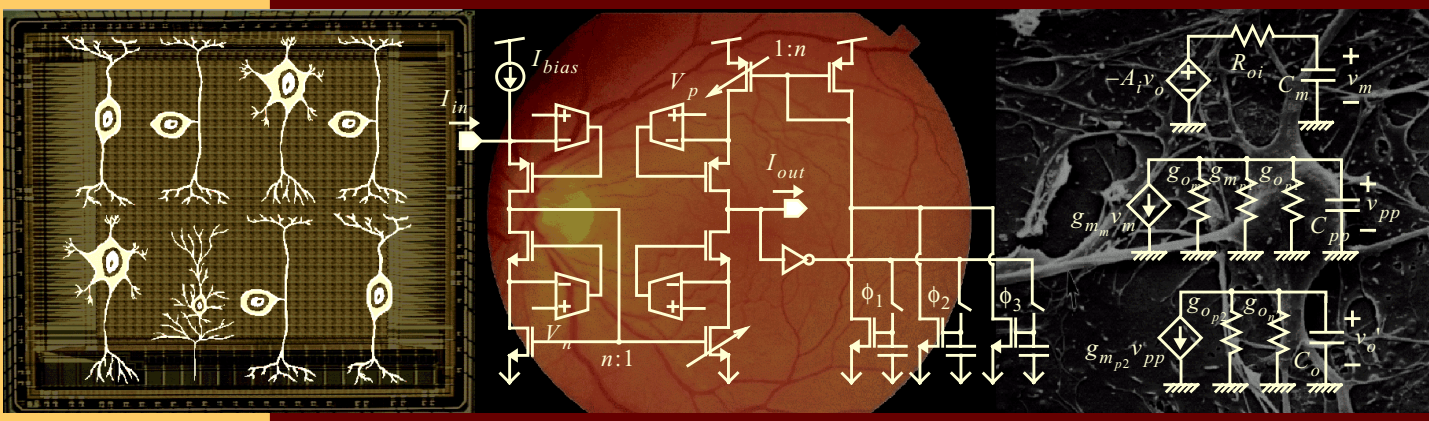
Análisis y Diseño de Hardware VLSI Basado en CNNs para el Procesamiento de Imágenes en Tiempo Real

Ricardo A. Carmona Galán



Departamento de Electrónica y Electromagnetismo
UNIVERSIDAD DE SEVILLA

Instituto de Microelectrónica de Sevilla-CNM-CSIC

UNIVERSIDAD DE SEVILLA

Departamento de Electrónica y Electromagnetismo

Análisis y Diseño de Hardware VLSI Basado en CNNs para el Procesamiento de Imágenes en Tiempo-Real

Memoria presentada por

RICARDO A. CARMONA GALÁN

para optar al grado de Doctor en Ciencias Físicas

Sevilla, 17 de Diciembre de 2001

Análisis y Diseño de Hardware VLSI Basado en CNNs para el Procesamiento de Imágenes en Tiempo-Real

Memoria presentada por

RICARDO A. CARMONA GALÁN

para optar al grado de Doctor en Ciencias Físicas

El Autor

El director

ÁNGEL RODRÍGUEZ VÁZQUEZ

UNIVERSIDAD DE SEVILLA

Departamento de Electrónica y Electromagnetismo

A Ricardo Carmona y Julio Maíllo,
in memoriam

A Reyes

Quisiera expresar mi agradecimiento más sincero a Ángel por su confianza y su dirección en este proceloso mundo de la investigación científica, a Servando y a Rafael, por sus consejos en el diseño analógico, a Gustavo, Ignacio y Paco por su inestimable colaboración, a Paco, Elisenda, José Manuel, Fernando, Rocío, Ángeles, Juan Manuel y a todos los demás compañeros que tanto en el IMSE-CNM como en la UC Berkeley generaron el entorno favorable para que el desarrollo de este trabajo fuera posible.

Ricardo Carmona Galán
Sevilla, 17 de Diciembre de 2001

“Porque donde unas cuencas vacías amanezcan,
ella pondrá dos piedras de futura mirada...”

Miguel Hernández
“El herido”– El Hombre Acecha, 1939

Índice

Prólogo

Durante las últimas décadas del siglo XX hemos asistido a dos oleadas revolucionarias sucesivas en el ámbito del tratamiento automático de la información y las comunicaciones. Estas se han materializado en el ordenador personal, PC, y la Internet, pero sin embargo tienen su origen en la aparición de avances tecnológicos relacionados con los dispositivos. Por un lado, el PC no es en sí mismo el germen de ninguna ola de informatización de los hogares del planeta, es, más bien, el producto de la revolución que en el procesamiento de la información significó la invención del microprocesador. Asimismo, la creación de la Internet no es el origen de la globalización del acceso a múltiples redes de ordenadores y enormes bases de datos remotas, es el resultado de la revolución de las conexiones y el almacenamiento de datos que ha supuesto la aparición de los láseres semiconductores. Estas dos olas tecnológicas, la revolución del procesamiento y la revolución del acceso a la información, impulsadas por la aparición de nuevos dispositivos, nos permiten hoy en día, según P. Saffo[i] describe en [Saff97], augurar una nueva revolución en el mundo de la tecnología de la información, capitaneada por nuevos dispositivos: los sensores. El desarrollo de nuevos sensores, y actuadores, basados en tecnologías existentes o por desarrollar, está ya empujando una nueva revolución que podemos llamar de la "interacción". Las máquinas que primero alcanzaron capacidades de procesamiento nunca antes imaginadas, y que luego se conectaron mediante fibra óptica a otras máquinas y a grandes bases de datos contenidas en soportes optoelectrónicos, van ahora a poseer la capacidad de interaccionar con el mundo que les rodea.

En un primer momento, estos sensores —dispositivos microelectromecánicos, materiales piezoeléctricos, micromáquinas, procesadores de video en un solo chip, microcámaras— supondrán una vía de conexión

i. Paul Saffo es un experto en nuevas tecnologías y su impacto en la economía y en la sociedad actual, ha trabajado como consejero en la industria, ha impartido clases en varias universidades, y es director del *Institute for the Future*, una fundación privada, con base en el *Silicon Valley* californiano, dedicada al pronóstico socioeconómico y a la planificación estratégica, cuya clientela está constituida por grandes corporaciones y agencias gubernamentales de Norteamérica, Europa y Asia.

eficiente y barata del mundo virtual de las máquinas digitales con el universo analógico que nos rodea. Pero el cambio en esta relación no va a detenerse ahí. El abaratamiento de estas habilidades conducirá, está conduciendo ya, a la aparición de artefactos inteligentes que realizarán sus tareas en continuo intercambio de información con su entorno. Las previsiones de P. Saffo van más allá, adelantando la aparición de estructuras y materiales formados por la aglomeración de dispositivos sensores/actuadores que adaptarían su comportamiento global a las condiciones del entorno. Por ejemplo, materiales inteligentes que recompusieran su estructura interna para incrementar o disminuir su resistencia a rozamientos, impactos, etc. Esto da lugar a un nuevo concepto de computación hiperdistribuida en el que multitud de procesadores elementales actúan de manera cooperativa y en el que el procesamiento analógico de la señal juega un papel capital, dadas sus características de alta velocidad, bajo perfil físico y bajo consumo de potencia, al ser comparados con su contrapartida digital.

En esta línea, el paradigma de las redes neuronales celulares (CNNs), introducidas por L. O. Chua[ii] [Chua93], supone un modelo de computación analógica distribuida, cuyas interacciones de carácter local permiten una realización física eficiente y rentable en tecnologías estándar de fabricación de circuitos integrados. Este modelo de computación cooperativa, propio de las redes neuronales en general, en las que la elevada interconexión entre sus elementos es responsable de comportamientos de muy alto orden, se ve aquí restringido a interacciones locales, de las que, aún así, emerge un comportamiento global muy complejo. Si a esto añadimos las posibilidades del substrato de silicio de las tecnologías CMOS como sensor y transductor del estímulo lumínico, nos encontramos ante una posibilidad fehaciente de producir microcámaras inteligentes en un solo chip, y a un coste muy por debajo de la alternativa digital convencional. Se han obtenido modelos, inspirados en la retina biológica, desarrollados por el Prof. Werblin[iii] en torno a la observación de la retina de los vertebrados [Werb95], y basados en el modelo de computación de las CNNs, que van a permitir la incorporación de los avances evolutivos a la implementación de un sistema de visión artificial compacto, eficiente y suficientemente eficaz.

Con el fin de adecuar la descripción algorítmica de dichos modelos a la posibilidad real de implementación física de los mismos, surgen arquitecturas que tratan de salvar la distancia impuesta por las limitaciones tecnológi-

ii. El Profesor Leon O. Chua, del *Department of Electrical Engineering and Computer Science* de la Universidad de California en Berkeley, es el inventor de las CNNs así como de circuitos que permiten la realización y el estudio del caos de manera controlada. Es autor de varios libros y multitud de artículos en el área de los circuitos y sistemas no lineales. Es poseedor de varias patentes y ha recibido numerosos premios internacionales y doctorados *Honoris Causa*.

iii. Frank Werblin es profesor de neurobiología en el *Department of Molecular and Cell Biology*, de la Universidad de California en Berkeley. Fue uno de los primeros científicos en desvelar los secretos de las contribuciones de las células a las funciones de la retina, usando técnicas electrofisiológicas. Durante 20 años ha trabajado en el estudio de la retina y el modelado de sus funciones.

cas. Por ejemplo, la Máquina Universal basada en CNNs (CNNUM), descrita por el Prof. Roska[iv] [Rosk93], supone una estructura totalmente programable y reconfigurable que, siendo compatible con las actuales tecnologías de fabricación, permite implementar los modelos de computación analógica distribuida referidos anteriormente. El último paso, relativamente pequeño en cuanto a lo conceptual, pero con toda una problemática práctica, de hondo calado, asociada, es la plasmación de dicho conjunto de reglas matemáticas en una analogía física, microelectrónica, en concreto, de muy alta escala de integración (VLSI).

En esta Tesis, pretendemos abordar la realización física VLSI de circuitos y sistemas encaminados al procesamiento de imágenes en tiempo real, a partir de la adaptación de los citados modelos de computación analógica distribuida al problema específico del tratamiento de bajo nivel de los estímulos visuales. Para ello iniciaremos el recorrido con el planteamiento del problema en términos eminentemente prácticos y con la sugerencia de una alternativa al enfoque convencional, basada en las CNNs. Estudiaremos también las limitaciones tecnológicas y su incidencia en los parámetros del sistema. A continuación, revisaremos las herramientas teóricas y prácticas que nos permitirán una mejora de la implementación desde el punto de vista de la robustez, y concluiremos con el estudio de dos prototipos en silicio de dos elementos fundamentales en la implementación física del procesamiento de imágenes basado en CNNs: un chip de memoria analógica que actuará como buffer o memoria caché dentro del sistema, y un chip de procesamiento basado en la dinámica espacio-temporal de una CNN de dos capas, con funciones de microprocesador visual, inspirado en los modelos de la retina antes mencionados.

Ricardo Carmona Galán
Sevilla, Diciembre 2001

iv. El Profesor Tamás Roska es el director del *Analogic and Neural Computing Laboratory* de la Academia de Ciencias de Hungría en Budapest. Es co-inventor de la CNNUM. Es autor de varios libros y multitud de artículos en las áreas de los circuitos neuronales y no lineales y de los sistemas de computación analógica. Es miembro de la Academia de Ciencias Europea y ha recibidos varios premios internacionales de reconocido prestigio.

Referencias del prólogo

[Chua93] L.O. Chua and T. Roska, “The CNN Paradigm”. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 40, No. 3, pp. 147-156, March 1993.

[Rosk93] T. Roska and L. O. Chua: “The CNN Universal Machine: An Analogic Array Computer”. *IEEE Transactions on circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 40, No. 3, pp. 163-173, March 1993.

[Saff97] Paul Saffo, “Sensors: The Next Wave of Infotech Innovation” in the *1997 Ten-Year Forecast*, pp. 115-122. Institute for the Future, Menlo Park, CA, 1997.

[Werb95] F. Werblin, T. Roska and L. O. Chua, “The Analogic Cellular Neural Network as a Bionic Eye”. *International Journal of Circuit Theory and Applications*, Vol. 23, No. 6, pp. 541-69, November-December 1995.

Análisis y Diseño de Hardware VLSI Basado en CNNs para el Procesamiento de Imágenes en Tiempo-Real

Procesar de imágenes y/o señales de video en tiempo real es una tarea que necesita de una elevada capacidad de cómputo. Fácilmente, velocidades de operación por encima de las gigaOPS (10^9 operaciones por segundo) pueden ser requeridas. El tamaño de las imágenes que deben procesarse en la práctica junto con la frecuencia de refresco de las mismas, en aplicaciones industriales e incluso domésticas, representan un enorme flujo de información. Por ejemplo, los sistemas de TV convencionales operan con imágenes de unos 644×483 puntos, o *pixels*, a una velocidad de 30 imágenes por segundo [Poyn96]. Esto representa una flujo de pixels de $9.3\text{Mpel}/\text{s}$, sin contar con los pixels extra que no transmiten información de la imagen en sí, sino que se utilizan para sincronizar el escaneado de la pantalla. Una simple operación de convolución de cada elemento de la imagen con una máscara espacial, por ejemplo, para realizar un promediado del valor del pixel con su vecindario, o cualquier otra operación de filtrado espacial lineal, en la cual, un conjunto de pesos se aplica a un vecindario de 3×3 -pixel alrededor de cada punto de la imagen, requiere 9 multiplicaciones y 8 sumas por pixel [Gonz87]. Esto representa una velocidad de procesamiento de 0.16GOPS para ser realizado en tiempo real. La misma operación en el ámbito de la HDTV (televisión de alta definición), para la cual el tamaño de la imagen es de 1920×1080 pixels y la frecuencia de refresco de 30fps (*frames per second*), significa que la potencia de cómputo requerida asciende a 1.06GOPS. Los algoritmos más complejos de procesamiento de imágenes, desarrollados para aplicaciones como la visión estereoscópica, la fusión de imágenes o la compresión de video, por ejemplo, requieren muchas más operaciones aritméticas y lógicas por pixel, excediendo ampliamente estas cifras, aproximándose al rango de las teraOPS (tera = 10^{12}).

La implementación VLSI (muy alta escala de integración) de semejante potencia de cálculo en el mundo digital, donde la paralelización de procesos representa una duplicación de hardware que suele ser prohibitiva, requiere que se adopten estrategias arquitecturales conducentes a dos

aproximaciones diferentes: arquitecturas dedicadas o programables. Por un lado, las arquitecturas dedicadas explotan la adaptación del *hardware* VLSI para la realización de tareas específicas. En el caso del procesamiento de imágenes a muy bajo nivel, nos referimos a la realización de tareas simples y repetitivas de naturaleza regular y paralela. Una arquitectura dedicada, en estas condiciones, resulta en una solución muy eficiente, en la que se minimiza la circuitería de control y el consumo de potencia a expensas de una pérdida notoria de flexibilidad. Por el contrario, una arquitectura programable, basada mayormente en el control realizado por el *software*, es capaz de realizar tareas en las que la demanda de cómputo y la complejidad del flujo de información son muy variables, sin embargo se muestran muy poco adecuadas para la ejecución de tareas de procesamiento intensivas de bajo nivel, las cuales, por desgracia, representan casi el 80% del trabajo a realizar en la mayoría de las aplicaciones de procesamiento de imágenes [Pirs98]. De modo que existe un compromiso a resolver entre la flexibilidad y la eficiencia, junto con las correspondientes limitaciones en el esfuerzo de diseño disponible y el *time-to-market* final en desarrollos industriales.

Una alternativa al enfoque digital convencional es el procesamiento analógico VLSI. Los circuitos analógicos resultan en realizaciones más compactas y eficientes de las funcionalidades que se necesitan en muchas de las tareas de procesamiento de imágenes. Su mayor desventaja es la reducida precisión de la que son capaces, especialmente en el caso de los *chips* de señal mixta, en los que circuitos analógicos y digitales conviven sobre el mismo substrato. De todos modos, una resolución equivalente de 8bits es suficiente para el procesamiento de imagen, según se establece en un amplio conjunto de estándares de video mundialmente aceptados [CCIR90a]. Más aun, las tareas de visión temprana, realizadas por chips de visión artificial inspirados en sistemas biológicos, pueden realizarse con resoluciones equivalentes de tan sólo 6-7 bits [Koch95], lo cual reduce aún más los requerimientos de precisión, ampliándose el área de aplicación de los circuitos analógicos VLSI.

Acerca del compromiso entre flexibilidad y eficiencia, una solución intermedia puede alcanzarse mediante una implementación analógica compacta en VLSI, que contenga cierto grado de programabilidad. En este contexto, se concibió la Máquina Universal basada en Redes Neuronales Celulares (*Cellular Neural Network Universal Machine*, CNNUM) [Rosk93a]. La CNNUM es un computador algorítmico programable analógico y paralelo, con una potencia de cómputo de un superordenador en un único chip. Puede verse como un procesador analógico paralelo especializado, adaptado para operar masivamente sobre gran cantidad de señales analógicas, y con suficiente flexibilidad para realizar algoritmos complejos mediante un programa que recoge instrucciones analógicas y lógicas. Se ha demostrado que la CNNUM es universal en el sentido de Turing [Crou96]. Realizaciones VLSI de la misma [Liña98] alcanzan velocidades de operación de hasta 0.4TOPS. Una cifra interesante cuando la comparamos con las $0.9\text{GIPS}^{\text{i}}$ obtenidas por DSPs multiprocesador de última generación [TI99].

En esta tesis pretendemos identificar y caracterizar las dificultades para la implementación VLSI de un sistema de procesamiento de imágenes en tiempo real basado en CNNs. Asimismo, propondremos una metodología de análisis de siste-

mas de procesamiento analógico paralelo e introduciremos las claves para una simulación y verificación eficiente del diseño. Abordaremos el diseño y la implementación VLSI de algunas piezas del hardware de un sistema de procesamiento de imágenes basado en CNNs, centrándonos en el estudio de dos prototipos: un chip de memoria analógica de corto término de apoyo al procesador visual, y un segundo chip, un procesador analógico y lógico paralelo capaz de realizar complejas evoluciones dinámicas y de implementar flujos de programa complicados, inspirado en descubrimientos sobre la operación de las capas más accesibles de la retina de los vertebrados.

Al final, habremos descubierto que estos métodos suponen una alternativa práctica para la realización del procesamiento de video en tiempo real, superando, en ciertas condiciones, las limitaciones tecnológicas que surgen en este campo, que es, por cierto, una de las más amplias áreas de aplicación y estudio para el procesamiento de señal con circuitos VLSI tanto analógicos como digitales.

1. Introducción al procesamiento de imágenes mediante CNNs

1. 1. Definiciones básicas y formatos de video analógico y digital

Modelo de la imagen

La representación de escenas tridimensionales de la vida real en forma de imágenes bidimensionales se basa en la proyección de la luz emitida y reflejada por los objetos sobre una superficie sensible a la misma. En la visión humana, esta superficie es la retina. Esta está compuesta por fotoreceptores que traducen los estímulos visuales en impulsos neuronales que pueden ser procesados por el sistema nervioso central [Hube88]. En fotografía, los rayos de luz se proyectan sobre una película sensible. El sistema de enfoque opera de igual manera que el sistema óptico del ojo humano. Esta película sensible es luego revelada mediante procesos físicos y químicos para ser luego impresa en un soporte manejable. En una videocámara, en lugar de la película sensible, la luz incide sobre un arreglo de fotoreceptores situados en el plano focal. Estos convierten la intensidad luminosa y el color de cada elemento de la imagen (*pixel*) en una señal eléctrica. La información visual queda codificada de este modo en un formato susceptible de procesamiento electrónico.

Para establecer un modelo de imagen sobre el cual trabajar, podemos

i. Aquí, la velocidad de operación de un procesador digital se mide IPS — instructions per second— las cuales consideraremos comparables a la operaciones analógicas por segundo (OPS) empleadas con anterioridad, a pesar de que esto puede resultar en una estimación pesimista ya que una simple multiplicación en el ámbito analógico puede suponer más de una micro-instrucción en un procesador digital.

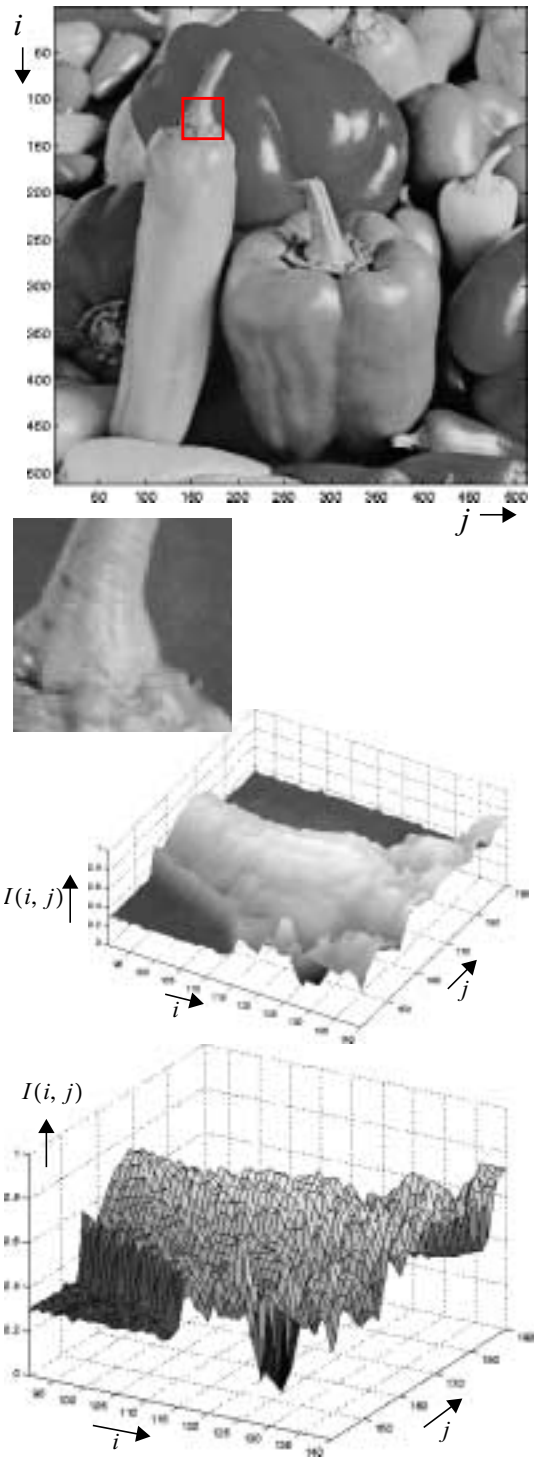


FIGURA 1. Imagen monocromática y representación de la superficie correspondiente a la intensidad luminosa del área recuadrada.

considerar que el término imagen se refiere a una función bidimensional de la intensidad luminosa. Aunque la luz que incide sobre los objetos en el mundo real puede tener componentes en un amplio rango de frecuencias, y la visión humana es capaz de percibir longitudes de onda entre los 400nm y los 700nm aproximadamente, vamos a restringirnos a la consideración de imágenes monocromáticas. En una imagen monocromática, cada pixel, de coordenadas (i, j) en el plano focal, donde i representa la posición vertical y j la horizontal en la matriz de puntos, se asocia a un valor de la *luminosidad* en una escala de grises. A este valor, E'_Y, se le conoce como *luminancia*. La imagen monocroma, I, queda definida entonces mediante:

$$I \equiv E'_Y(i, j) \tag{1}$$

que está normalizada a la unidad. La naturaleza discreta del soporte de la imagen —debido al tamaño y número finito de fotoreceptores en una cámara, o al número de puntos en una pantalla— conduce a la discretización del plano de la imagen, por lo que la función de la intensidad aparece como una función muestreada a intervalos regulares en las dos direcciones del plano. Por otro lado, la imagen está, naturalmente, limitada, de modo que las coordenadas del pixel podrán tomar valores, únicamente, de un conjunto finito:

$$\begin{aligned}(i, j) \in \{&(1, 1), \ldots, (1, N_j), (2, 1), \ldots \\ &\ldots, (N_i - 1, N_j), \ldots, (N_i, N_j)\}\end{aligned} \tag{2}$$

donde N_i y N_j representan el número total de pixels en ambas direcciones del plano. De este modo, la imagen será descrita por una matriz de $N_i \times N_j$ valores de intensidad luminosa, o sea, N_i filas, conteniendo cada una N_j muestras de la luminancia[ii]. La resolución espacial de la imagen, o sea, la agudeza de la representación, es directamente proporcional al tamaño de $E'_Y(i, j)$.

ii. Nótese que el tamaño de la imagen se expresa comúnmente como el producto de la base, si la imagen es rectangular, por la altura, al revés de como se indican las dimensiones de la matriz que la representa según este modelo.

Frecuencias de refresco, de línea y de muestreo

Las imágenes en movimiento se basan en la ilusión visual creada por una sucesión de imágenes estáticas, $\{I_k(i, j)\}_{k = 0, \ldots, N}$, cuando estas son capturadas y luego reproducidas a una determinada velocidad. Esto convierte a la función que representa la luminosidad en una función no sólo de las coordenadas espaciales, sino también del tiempo, $I(i, j, t)$. Supongamos que las imágenes estáticas son capturadas o expuestas a intervalos regulares de tiempo, esto es, en los instantes 0, T_V, $2T_V$, $3T_V$, ... El tiempo que dista entre dos capturas, T_V, define la frecuencia de refresco $f_V = 1/T_V$. También es conocida como frecuencia de sincronización vertical en el ámbito de los sistemas de monitorizado de imágenes.

Como indica f_V, una imagen completa (o al menos un campo de la imagen, segun el estándar) es capturada, transmitida o visualizada cada $1/f_V$ segundos. Durante este tiempo, la información contenida en el plano de la imagen es recorrida línea tras línea, de arriba a abajo. Puede definirse, en estas condiciones, una frecuencia de sincronización horizontal, f_H, o frecuencia de línea, a partir de f_V y el número total de líneas contenidas en la imagen, N_V:

$$f_H = N_V \cdot f_V \tag{3}$$

N_V suele ser diferente a N_i, el número de filas de la matriz que representa la intensidad luminosa de la imagen, debido a que no todas las líneas indicadas por N_V contienen información acerca de la luminancia. Algunas de estas líneas sirven únicamente para permitir que, en los sistemas de visualización de imágenes en movimiento, el agente físico que realiza el refresco de la imagen (p. ej. el cañón de electrones en un tubo de rayos catódicos, por ejemplo) retorne a la posición inicial antes de comenzar a describir una nueva imagen (Fig. 2). También existe una discrepancia entre el número total de muestras de la luminancia contenidas en una línea de la imagen, N_H, y el número real de elementos de la imagen por fila, N_j. Los motivos son análogos a los expresados para la sincronización vertical.

Puede definirse también una frecuencia de pixel, o frecuencia del reloj de pixel, o simplemente frecuencia de muestreo, f_S, que es la frecuencia a la que las muestras de la intensidad luminosa de la imagen son capturadas, transmitidas o visualizadas. A partir de f_H y N_H:

$$f_S = N_H \cdot f_H \tag{4}$$

Al intervalo $T_S = 1/f_S$ se le conoce como periodo de muestreo o periodo del reloj de pixel.

Hasta ahora hemos supuesto que el escaneado de la imagen se realiza línea tras línea desde lo más alto de la imagen hasta abajo. Esto es lo que se considera un escaneado progresivo de la imagen. Existe otro modo de realizar el escaneado de la imagen que consiste en dividirla en dos campos o más, cada uno de ellos conteniendo líneas alternativas de la imagen, que son transmitidos uno tras otro. Con esto se consigue transmitir a una

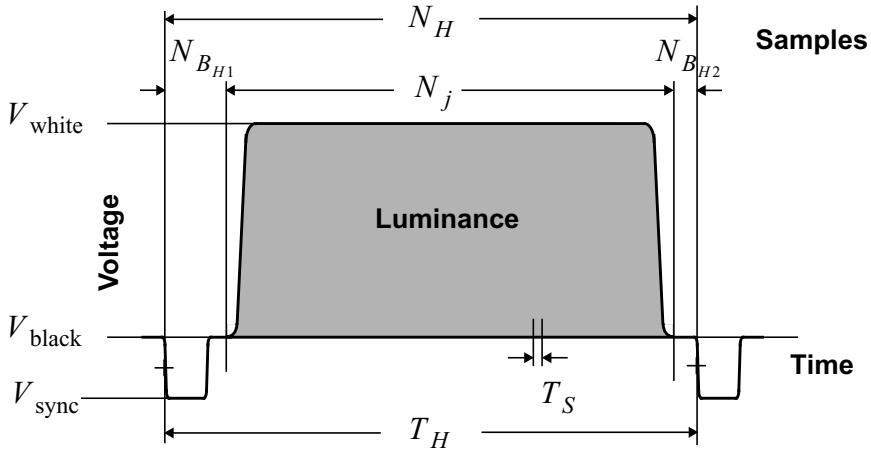


FIGURA 2. Forma de onda de una linea de la imagen.

menor frecuencia de refresco de la imagen completa, sin perder la percepción que el usuario final tiene de la resolución espacial de la imagen. A esto se le conoce como escaneado entrelazado de la imagen. También relacionada con la frecuencia de muestreo está la codificación del color. Puesto que el usuario final de un sistema de visualización es humano y dado que nuestro sistema de visión posee una menor capacidad para distinguir cambios en los colores que en la intensidad luminosa total, la información relativa al color puede sub-muestrearse (véase Apéndice 1).

Hay dos magnitudes que se emplean comúnmente para cuantificar la cantidad de información transmitida por unidad de tiempo en el procesamiento de señal de video. Para lo que llamamos video analógico, hablamos del ancho de banda de la señal, BW. Este ancho de banda de video analógico es la frecuencia máxima a partir de la cual no pueden encontrarse contribuciones importantes al espectro de la señal. Este límite suele estar impuesto por la frecuencia de muestreo. De acuerdo con el teorema de Nyquist [Oppe93], una señal limitada en banda $m(t)$, con un ancho de banda de f_m rad/s, debe ser muestreada al menos a una frecuencia de muestreo $2f_m$ para que pueda ser reconstruida a partir de sus muestras. Una lectura apropiada de este teorema sería que, cualquier dispositivo de adquisición o visualización de imágenes que opere a una frecuencia de muestreo f_S, no puede capturar o transmitir componentes del espectro la entrada que se encuentren más allá de $f_S/2$. Esto es, características de la imagen cuya periodicidad espacial en la dirección horizontal esté por debajo de $2w_p$ —dos veces la anchura de un pixel— se perderán tras pasar por dicho sistema. El ancho de banda de señal de video será:

$$BW = f_S/2 \tag{5}$$

El equivalente digital al ancho de banda de video en el terreno analógico es el flujo de datos, D. Para una secuencia de video monocromática, donde N sería el número de bits empleados en codificar cada muestra de la luminancia, tenemos:

$$D = N \cdot f_S \tag{6}$$

lo cual nos permite relacionar fácilmente el ancho de banda de video analógico con

el flujo de datos que este representa en el dominio digital:

$$BW = D/2N \tag{7}$$

Estándares de codificación y escaneado de video

Diferentes estándares de video analógico y digital han ido apareciendo como resultado de esfuerzos conjuntos entre la industria, las compañías de comunicación y diferentes organizaciones internacionales. El objetivo de la estandarización es compatibilizar la investigación y los avances con la estructura existente, con el fin de facilitar la producción y comercialización en masa de los nuevos dispositivos. Estos son algunos de los formatos más extendidos.

En cuanto a la televisión convencional, el documento Rec. 601-2 [CCIR90a] recoge el formato de las señales de video compuesto NTSC y PAL, así como sus estándares de escaneado asociados, $525/59.94$ y $625/50$ respectivamente. Esta notación indica el número total de líneas por imagen N_V, incluyendo las dedicadas a sincronización, y la frecuencia de refresco relativa a los campos por segundo, ya que ambos estándares son entrelazados (Tabla 1). Las frecuencias de refresco en estos sistemas son de 29.97Hz y 25Hz, respectivamente. A partir del tamaño de la imagen, incluyendo los periodos dedicados a sincronización, obtenemos, en ambos casos, una frecuencia de muestreo de la luminancia de 13.5MHz. Las señales que representan las diferencias de color son muestreadas a 6.75MHz. En ambos estándares, la tensión de estas señales va desde -300mV, que representa la parte más baja del pulso de sincronización horizontal V_{sync}, a los 700mV que corresponde al máximo de cada componente de la señal de video, V_{white}. El nivel correspondiente a una luminosidad cero de la imagen, V_{black}, es 0mV.

	525/60	625/50
f_V (Hz)	29.97	25
N_V	525	625
N_i	483	576
f_H (kHz)	15.750	15.625
N_H	858	864
N_{BH1}	122	132
N_j	720	
N_{BH2}	16	12
f_S	13.5MHz	

TABLA 1. Tamaño de la imagen y frecuencias de reloj en Rec. 601-2

En lo concerniente a formatos de video digital, para aplicaciones gráficas por ordenador, IBM introdujo en 1987 el VGA (*video graphic array*) que se convirtió en un estándar *de facto*. La *Video Electronics Standard Association* (VESA) generó algunas extensiones de este sistema como el SVGA (Super VGA) [VESA90] o el XGA (Extended graphic array) [VESA93]. El sistema de gráficos de un ordenador está compuesto por tres elementos: la tarjeta gráfica, o adaptador de video, el monitor y el *driver*, o sea, el software que controla la tarjeta. La tarjeta de video es una extensión de la placa madre, que incluye un chip de procesamiento gráfico, memoria RAM y una dispositivo RAMDAC (RAM-based DAC) que convierte los datos digitales en señales analógicas susceptibles de ser utilizadas por el monitor. Para más información acerca de diferentes estándares de video, véase Apéndice 1.

1. 2. Implementación VLSI del procesamiento de imágenes en tiempo real

Flexibilidad frente a eficiencia

La implementación del procesamiento de imágenes y video en tiempo real mediante circuitos VLSI abre un sinfín posibilidades y nuevas aplicaciones en los campos de la robótica, las telecomunicaciones y los dispositivos multimedia. La tecnología VLSI ofrece bajo consumo de potencia para dispositivos portátiles y aislados, alta velocidad y prestaciones en aplicaciones donde pueden ser críticas y, además, bajo coste para poder entrar en una comercialización masiva. Desde un punto de vista arquitectural, las dos alternativas para la implementación VLSI del procesamiento de imágenes en tiempo real serían, tanto en el ámbito digital como en el analógico, la realización de arquitecturas programables o dedicadas [Pirs98]. Conviene entonces analizar las ventajas y desventajas de cada una de ellas. Por un lado, los microprocesadores digitales de propósito general representan una implementación muy flexible, capaz de realizar algoritmos muy diversos y de muy diferente complejidad. Estos procesadores de propósito general pueden exhibir unas prestaciones magníficas en el ámbito del procesamiento de datos en general, pero están pobremente cualificados para el procesamiento de señal. Además, son caros y consumen mucha potencia como para ser utilizados en aplicaciones aisladas de multimedia o telecomunicación. Por otro lado, una arquitectura dedicada, basada en la adaptación a las necesidades específicas del algoritmo, el formato de la señal de video y la distribución paralela inherente de la información de la imagen, resulta en una implementación máximamente eficiente, a expensas, claro está, de una reducción en la flexibilidad y la programabilidad del sistema.

El procesamiento de imágenes y de video requiere de la concatenación de una gran variedad de algoritmos, de muy diferente complejidad y con muy distinta demanda computacional. Atendiendo a la complejidad, podemos clasificar la operaciones para procesado de imágenes en tres clases: tareas de alto, medio y bajo nivel. La complejidad de una operación está relacionada con la regularidad del flujo computacional, el tamaño de los datos envueltos y en número de los mismos, la simplicidad de las estructuras de datos y el nivel de abstracción [Pirs98]. Un flujo computacional muy regular contendría un conjunto de operaciones predefinidas que serían ejecutadas sobre una vasta cantidad de datos, con independencia de los valores representados por los mismos. Por contra, un flujo computacional irregular se caracteriza por la toma de decisiones dependientes de los datos y un alto nivel de impredecibilidad. Mientras que aquel puede ser realizado cómodamente mediante una arquitectura paralela, alcanzando altas cotas de adaptación y eficiencia, éste último se basa en la flexibilidad y programabilidad del procesador. Obviamente, esto precisa de una cierta circuitería dedicada al control de los datos y del flujo del programa. Además de las divergencias en cuanto a la complejidad del flujo computacional, las tareas de alto nivel se diferencian de las de bajo nivel en que operan sobre un número reducido de símbolos u objetos de la ima-

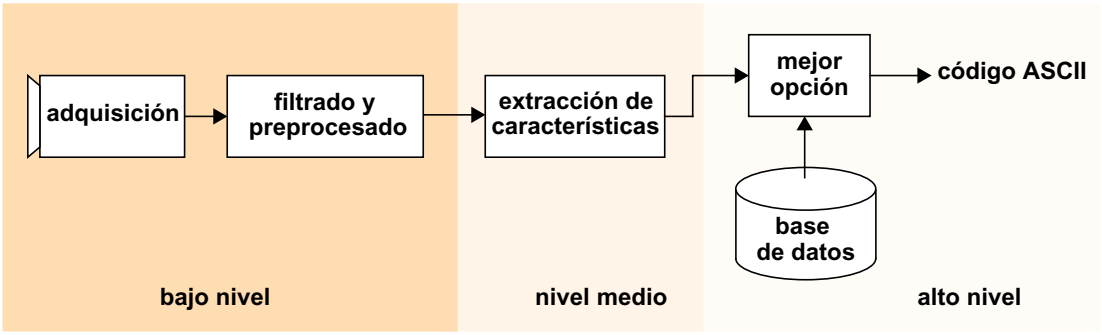


FIGURA 3. Diagrama conceptual de un sistema de OCR genérico.

gen, cuya estructura suele ser bastante más compleja que la de unos simples conjuntos de pixels. En cuanto a la potencia de cálculo, las tareas de bajo nivel demandan, por lo general, una potencia de cómputo mayor. Esto puede parecer, aparentemente, una contradicción, sin embargo las tareas de alto nivel, que dependen de la programabilidad del sistema, operan sobre un número reducido de elementos, complejos, pero pocos, mientras que las tareas de bajo nivel operan sobre todos los elementos de la imagen, que aunque poseen una estructura de datos simple, son excepcionalmente numerosos. En resumen, las tareas de bajo nivel admiten un alto grado de paralelización, operan sobre estructuras de datos muy simples, pero suponen un gran esfuerzo de cómputo por causa de la enorme cantidad de datos contenidos en una imagen. Por el contrario, las tareas de alto nivel, trabajan sobre un conjunto reducido de estructuras de datos complejas, de modo que requieren de arquitecturas muy flexibles mejor que arquitecturas adaptadas al cómputo intensivo (Tabla 2).

Como ejemplo, consideremos un sistema de reconocimiento óptico de caracteres (OCR). Esta aplicación incluye procesamiento de imágenes a diferentes niveles de abstracción. Un sistema OCR sirve para convertir texto impreso o incluso manuscrito en texto electrónico (Fig. 3). El primer paso consistiría en la adquisición de datos, mediante una cámara o un escáner. A continuación, o concurrentemente, la imagen es preprocesada, se filtra el ruido, se detectan líneas, etc. Todas estas operaciones son de bajo nivel. Son realizadas sobre todos los datos, independientemente del valor en sí que representan. Una vez completadas estas tareas, se procede a la extracción de características, lo que convierte al enorme conjunto de datos asociados a los pixels, en una representación simbólica que recoge ciertos

	Complejidad		
	BAJA	**MEDIA**	**ALTA**
Flujo computacional	regular	irregular	muy irregular
Estructuras de datos	simples	simples y/o complejas	complejas
Número de datos	grande	medio-grande	pequeño-mediano
Demanda de potencia	alta	media-variable	baja-variable

TABLA 2. Características de las tareas de procesamiento de imágenes atendiendo a su complejidad.

(a) (b) (c) (d)

FIGURA 4. Ejemplos de operaciones de pixel, locales y globales.

rasgos particulares de la imagen. Esta tarea puede considerarse de nivel medio. Las estructuras de datos se complican, a la vez que se reduce su número. Finalmente, esta información, más estructurada, se emplea para comparar con una base de datos y seleccionar la opción que ofrece un mayor grado de apareamiento. Esta última parte es, claramente, una tarea de alto nivel, que puede tener un flujo computacional bastante irregular, pero que opera sobre un conjunto muy reducido de datos.

Memoria de corto término

Una componente fundamental en el procesamiento de imágenes a bajo nivel es la memoria de corto término. Pensemos en que, por un lado, la salida correspondiente a cada pixel se obtiene a partir de un subconjunto de pixels de la imagen, o de la secuencia de imágenes, que puede ser arbitrariamente grande. Por otro lado, toda esta información no es transmitida de una vez, sino que sigue un esquema prescrito. De manera que cualquier sistema de procesamiento de imágenes debe estar provisto de algún mecanismo de almacenamiento temporal de datos y, eventualmente, de reorganización o adaptación de esos datos. Más adelante nos ocuparemos de materializar estos mecanismos, ahora vamos simplemente a concentrarnos en las necesidades del sistema.

La cantidad mínima de memoria necesaria para elaborar la respuesta correspondiente a un sólo pixel es, trivialmente, cero. Ya que si asignamos un valor fijo a todos los pixels de una imagen, no necesitamos memorizar nada. La utilidad de este operador es ciertamente restringida. En general, para computar la salida asociada a un determinado pixel de la imagen necesitaremos conocer los valores del mismo y/o de sus vecinos. A veces, toda la información de la imagen es requerida. Otras veces incluso, va a ser necesario disponer de información correspondiente a otras imágenes en una secuencia para poder elaborar la respuesta. Atendiendo al ámbito de la operación, esto es, a la cantidad de pixels envueltos en la obtención del valor de salida de un único pixel podemos hablar de

- operaciones de pixel
- operaciones locales
- operaciones globales
- procesamiento entre *frames*

En la Fig. 4 encontramos ejemplos de procesamiento con operadores de diferentes categorías. En primer lugar, un umbralizado de la imagen en Fig. 4(a), operación de pixel, produce la salida mostrada en Fig. 4(b). Como ejemplo de operación local, hemos aplicado los operadores de Sobel [Gonz87] para computar las componentes del gradiente de la imagen a partir de un vecindario de 3×3 pixels. Como resultado tenemos la imagen en Fig. 4(c). Como ejemplo de operador global hemos ecualizado el histograma de la imagen en Fig. 4(a), obteniendo Fig. 4(d). Esta operación puede ser considerada de nivel medio, ya que actuamos sobre el mapa de colores de la imagen, que puede ser contemplado como una estructura de datos de un nivel de abstracción superior. Finalmente, cualquier operación encaminada a la detección de movimiento sería un ejemplo de procesamiento entre *frames*, ya que necesitaría considerar información procedente de diferentes imágenes de una secuencia.

Así, dependiendo de las operaciones que tengamos previsto realizar, y también de los requerimientos impuestos por el formato de la señal de video, podemos establecer el tamaño de la memoria de corto término que vamos a precisar para implementar determinado algoritmo. Este *buffer* de datos, cuyo tamaño depende de la naturaleza de la operación, deberá llenarse antes de proceder al procesamiento, o bien deberá vaciarse antes de dar paso a la siguiente salida. Con el obtener el máximo rendimiento de las capacidades del sistema, nos planteamos realizar las tareas de adquisición, procesamiento y descarga de datos mediante tuberías (*pipeline*)

Pipeline para procesamiento de video

El procesamiento en tiempo real de señales de video requiere que las tasas de entrada y salida (E/S) de datos coincidan con los de transmisión de la señal de video. De este modo, las imágenes han de ser procesadas a una frecuencia de f_V imágenes por segundo, f_H líneas por segundo, o, de manera equivalente, f_S pixels por segundo. No obstante, cierto periodo de tiempo entre la llegada de los primeros datos y la disponibilidad de las primeras salidas es necesario. A este periodo se le denomina *latencia* del sistema de procesamiento. En particular, el procesamiento de imágenes en 2-D requiere de la anticipación de varias líneas de la imagen antes de elaborar la primera línea de la imagen de salida. Por ejemplo, los algoritmos de compresión MPEG aplican la transformada del coseno discreto (DCT) sobre macro-bloques de 8×8 pixels para cuantificar la correlación espacial entre puntos de la imagen relativamente cercanos [Siko97]. Por tanto, si la imagen se trasmite línea a línea, será preciso haber adquirido las 8 primeras líneas de la imagen antes de que comience el procesamiento. Si el numero de líneas necesarias para procesar una imagen es N_o, observaremos un retraso de $\tau_o = N_o T_H$. Del mismo modo, si el algoritmo incluye procesamiento entre *frames*, in necesitamos N_F imágenes previas antes de empezar a elaborar la respuesta, tendremos que añadir $N_F T_V$ segundos al retraso inicial. Una vez transcurrido dicho periodo de latencia, el sistema debe ser capaz de mantener la citadas tasas de E/S de datos, lo cual va a representar el requerimiento más estricto en

cuanto la velocidad de operación del sistema de procesamiento.

Consideremos que un sistema de procesamiento de video en tiempo real transmite a una frecuencia de línea de f_H. Esto significa que necesita un periodo de tiempo T_H para adquirir una línea de la entrada o para emitir la salida correspondiente a una línea de la imagen. Esto es así independientemente del número de líneas que deba existir en el *buffer* para poder procesar correctamente. Jugando con el periodo de latencia, podemos obtener un sistema que opere a una frecuencia f_H incluso si el tiempo de procesamiento es n_s veces mas largo que el periodo de línea, T_H. Para ello, es necesario dividir el procesamiento en etapas —lo que a veces significa añadir *hardware*, aunque esto va a conseguir reducir las limitaciones temporales— y luego establecer con estas etapas, y las de adquisición y de salida, una estructura de *pipeline*. En la Fig. 5 podemos ver un pipeline de $(n_s + 2)$ etapas, que añade un tiempo $n_s T_H$ al periodo de latencia original, que ahora será de $(N_o + n_s)T_H$, pero que permite mantener las tasas de E/S prescritas. Mediante un razonamiento análogo puede constituirse un pipeline que contemple procesamiento *interframe*. Observemos, en el ejemplo, que el procesamiento se realiza en el tiempo $n_s T_H$ requerido, y aún así, la tasa de E/S, f_H, se mantiene, lo cual a sido posible sólo permitiendo un cierto incremento de la latencia del sistema. Para optimizar el uso de la tubería, es necesario realizar un particionado razonable del proceso en etapas [Poll90]. Las mejores prestaciones se obtienen en casos de máxima adaptación de la arquitectura, lo cual impide la implementación de la flexibilidad suficiente para realizar tareas de alto nivel. De todos modos, nosotros vamos a estar interesados en una solución de compromiso que nos permita obtener gran velocidad de procesamiento con cierto grado de programabilidad y reconfigurabilidad, para así poder plantearnos la realización de sistemas completos en un sólo chip.

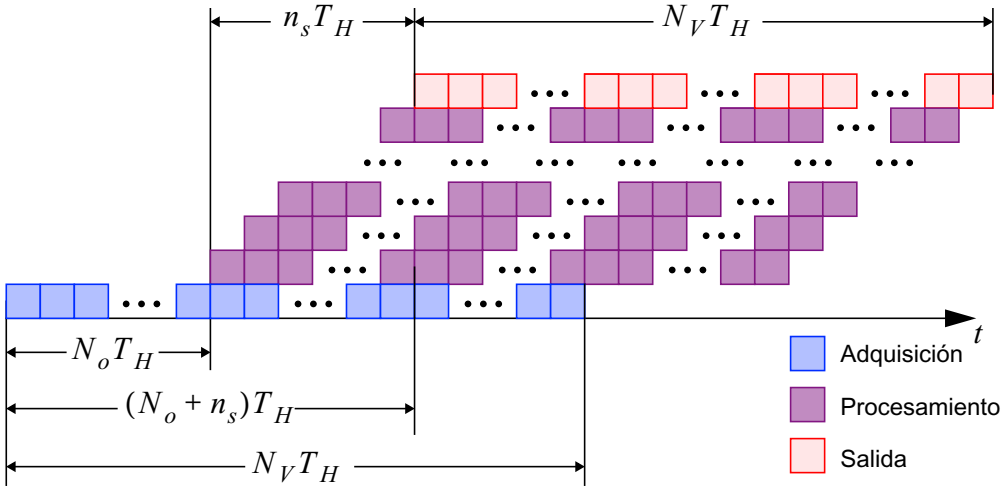


FIGURA 5. *Pipeline* para procesamiento de video (*intraframe*)

Arreglos de procesadores analógicos en paralelo (APAP)

En el ámbito de los circuitos digitales, la implementación VLSI de sistemas de procesamiento de imágenes y video en tiempo real se realiza de acuerdo a uno de estos dos enfoques: arquitecturas dedicadas o procesadores multimedia programables [Pirs98]. Ambas estrategias tratan de resolver el compromiso entre flexibilidad y eficiencia. Por un lado, las arquitecturas dedicadas ofrecen la mayor eficiencia y pueden encontrarse aplicaciones para las que resultan óptimas. Sin embargo, los algoritmos de procesamiento de imágenes de cierta complejidad, necesitan también de cierto grado de flexibilidad del cual estas carecen. Una integración factible de paralelismo y programabilidad, fácilmente transportable al ámbito de los procesadores analógicos en paralelo, como veremos más tarde, es la arquitectura SIMD (*single instruction stream–multiple data streams*). En un procesador SIMD [Flyn66], una unidad de control global gestiona varios caminos de señal paralelos (Fig. 6). En todos estos caminos se realiza la misma operación, sólo que sobre diferentes datos. Operaciones condicionales pueden realizarse mediante la inhibición de la operación en ciertos caminos a partir del enmascaramiento de las instrucciones.

Un caso particular de arquitectura multiprocesador, especialmente adecuada para el procesamiento de imágenes, sería una matriz de procesadores que, siguiendo una arquitectura SIMD, empleara un procesador básico por pixel. Esto permitiría mapear una señal 2-D, una imagen, sobre los nodos de la red de procesadores, de modo que esta realizara la misma operación sobre cada pixel de la imagen en paralelo. La gran ventaja de este sistema es que toda la imagen sería procesada en el mismo tiempo que un sistema serial tardaría en procesar un único pixel. Si combinamos esto con una cierta cantidad de memoria local, dentro de cada pixel, pueden alcanzarse velocidades de procesamiento realmente grandes, muy por encima de las gigaOPS.

Por desgracia, la implementación VLSI de un número suficientemente elevado de unidades básicas de procesamiento —de un número de procesadores que resulte práctico desde el punto de vista industrial— no es problema fácil de resolver. En primer lugar, el tamaño del chip está limitado por el rendimiento de la fabricación (*yield* en inglés). O sea, que una cierta cantidad de las muestras producidas deben ser completamente operacionales para que los costes de fabricación sean razonables. Para que

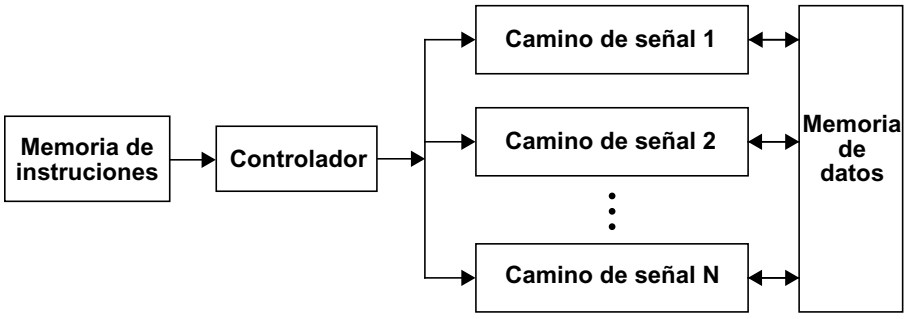


FIGURA 6. Diagrama conceptual de una arquitectura multiprocesador SIMD.

esto ocurra, el tamaño del chip debe estar constreñido a unos límites de seguridad, lo cual, inevitablemente, limita el número de procesadores básicos que pueden ser integrados en un mismo chip. Otra restricción sobre el número máximo de unidades de proceso que pueden incluirse en el mismo chip de silicio surge en conexión con los requerimientos de precisión. Tanto en realizaciones analógicas como digitales, el tamaño del procesador crece junto con la resolución prescrita. En el caso de los procesadores digitales, un bit más de resolución implica, al menos, la duplicación del área del circuito [McCl86]. En el caso de los circuitos analógicos para procesamiento de señal, cuya operación depende muy habitualmente de la cancelación de no-idealidades por parte de dispositivos apareados, es necesario utilizar dispositivos grandes, que ocupen un área apreciable, para obtener una precisión alta. Esto se debe a la relación inversa que existe entre la desviación estadística de los parámetros del proceso y el tamaño de los dispositivos [Pelg89]. De modo que para una mayor precisión, los procesadores unitarios deben ocupar más área, lo cual reduce la densidad de empaquetamiento de los mismos, por tanto menor será el número de procesadores básico por chip disponible. Aún así, sin dejar de tener en cuenta la presencia de limitaciones tecnológicas, todavía pueden adoptarse ciertas estrategias de diseño que permitan realizar el procesamiento de imágenes en tiempo real en VLSI. Por un lado, para una resolución moderada (7 u 8 bits), los circuitos VLSI analógicos representan una alternativa rápida, compacta y más eficiente en área y consumo de potencia que los circuitos digitales [Isma94]. Por otro lado, un diseño optimizado del sistema puede permitir el procesamiento en tiempo real —en el sentido descrito anteriormente de mantener unas determinadas tasas de E/S de datos, aún a costa de una mayor latencia— de imágenes grandes con procesadores más pequeños.

Comparación entre procesamiento analógico y digital en VLSI

De modo que, aunque el rol tradicional del procesamiento analógico en circuitos VLSI de señal mixta ha sido proveer al *core* digital del chip de la necesaria interfaz de E/S, la situación cambia en ciertas aplicaciones dado lo inapropiado de los circuitos digitales convencionales para procesar grandes cantidades de datos —analógicos en origen, por lo que necesitarían de una conversión A/D— en paralelo. Básicamente, cuando un flujo de información masivo, generalmente de naturaleza sensorial, debe ser tratado en paralelo, para cumplir ciertas restricciones temporales, los circuitos digitales resultan en una alternativa costosa en área y potencia debido a su reducida *densidad computacional* (potencia de cómputo por unidad de área). Para ilustrar esto, comparemos los ejemplos de procesadores analógico y digitales contenidos en la Tabla 3. Suponiendo que estos circuitos sean representativos de los diferentes enfoques del problema, lo primero que encontramos es una mayor potencia de cómputo por área en aquellos chips basados en APAP, comparados con sus contrapartidas digitales. Lo mismo puede decirse en cuanto a la potencia de cómputo por consumo de potencia. Los procesadores básicos analógicos son necesitan mucha menos potencia que los bloques digitales. De espe-

Ref.	Descripción del chip y de las técnicas de diseño	Proceso CMOS	Nº de celdas	celdas/ mm^2	XPS^i/ mm^2	XPS/ mW
ANALÓGICOS Y DE SEÑAL MIXTA						
[Espe94b]	Red neuronal celular reconfigurable basada en técnicas de modo de corriente.	1.6µm	1024	165	85.5G	17M
[Domí97]	CNN con E/S binaria, memoria de programa y sinapsis analógica de 4 trans.	0.8µm	440	27.5	8.25G	0.12G
[Mart98]	Procesador paralelo de señal mixta. Basado en conversión ADC-DAC cíclica.	0.8µm	25	5.26	4.2M	0.43M
[Liña98]	APAP con E/S en escala de grises, con memoria de programa y sinapsis de un único transistor	0.5µm	4096	81	7.93G	0.33G
DIGITALES						
[Yama94]	Procesador digital de señal de video con memoria de programa.	0.55µm (BiCMOS)	64	0.33	1.2M	68k
[Moll98]	Procesador digital de video (con una FPGA controladora externa).	1.2µm	N/A	N/A	5.0M	0.28M
[Sait98]	ASIC digital: Red neuronal de un millón de sinapsis.	0.25µm	N/A	N/A	44M	2.6M
[Geal99]	Procesador digital de pixels en paralelo con memoria local.	0.6µm	4096	66.7	4.0M	1.0M

TABLA 3. Comparación de procesadores paralelos analógicos y digitales

i. IPS: las Instrucciones Por Segundo son una medida típica de la velocidad de un procesador digital. Instrucciones comunes pueden ser la suma bit a bit, el complemento, el desplazamiento de bits, etc. XPS: la Operaciones Analógicas Por Segundo constituyen una medida equivalente que indica el número de operaciones aritméticas como la suma o la multiplicación. En los casos en los que se observa una evolución dinámica espacio-temporal, las XPS han sido calculadas siguiendo el método de integración de Runge-Kutta de 4º orden, tal como se propone en [Harr94].

cial interés es la comparación entre los chips presentados en [Geal99] y [Liña98]. Ambos emplean una estructura SIMD con un procesador masivo paralelo con el mismo número de procesadores unitarios. La densidad de celdas es similar, pero el procesador digital realiza tareas mucho más sencillas. Aún así, la potencia de cálculo por consumo de potencia es 2 o 3 ordenes de magnitud mayor en el chip analógico.

1. 3. Procesamiento de imágenes con redes neuronales celulares

Descripción y modelos de CNN

En el año 1988, los profesores L. O. Chua y L. Yang de la Universidad de California en Berkeley, definieron una nueva clase de procesadores analógicos dinámicos: las Redes Neuronales Celulares (CNN del inglés Cellular Neural Network) [Chua88]. Las unidades elementales de este arreglo de procesadores en paralelo poseen la característica distintiva de estar conectados únicamente con sus vecinos más cercanos, en contraposición a las redes neuronales artificiales (ANN), que están completamente conectadas. A diferencia de los autómatas celulares, que son procesadores celulares digitales, las CNNs están compuestas por procesadores analógicos que interaccionan mediante señales analógicas. Debido a que poseen un grado de conectividad exclusivamente local, las CNNs resultan especialmente adecuadas para la implementación VLSI. En especial, la clase de CNNs que posee invariancia traslacional, en las que todas las celdas son idénticas (salvo aquellas que se encuentran en el borde de la red, obviamente) y el patrón de interacción con los vecinos se mantiene constante a lo largo y ancho de la matriz de procesadores.

De acuerdo con lo expuesto en [Chua93], una CNN es un arreglo de procesadores dinámicos no-lineales, también llamados células o celdas, que ocupan los nudos de una rejilla bi-, tri- o n-dimensional . Estas celdas deben satisfacer las dos propiedades citadas anteriormente: las interacciones entre ellas están restringidas a un ámbito local, las variables de estado son señales continuas en amplitud y en el tiempo. El conjunto de celdas recibe el nombre de dominio de la red (*grid domain*, GD). Por comodidad, supongamos que se trata de una red bidimensional compuesta por $M \times N$ celdas. esto no supondrá una pérdida de generalidad puesto que todas las definiciones pueden extenderse fácilmente a redes con más dimensiones. Cada celda de la red interaccionará directamente sólo con aquellas celdas que se encuentren dentro de un radio r_c . La Fig. 7 ilustra el concepto de vecindario de una celda $C(i, j)$, para diferentes valores

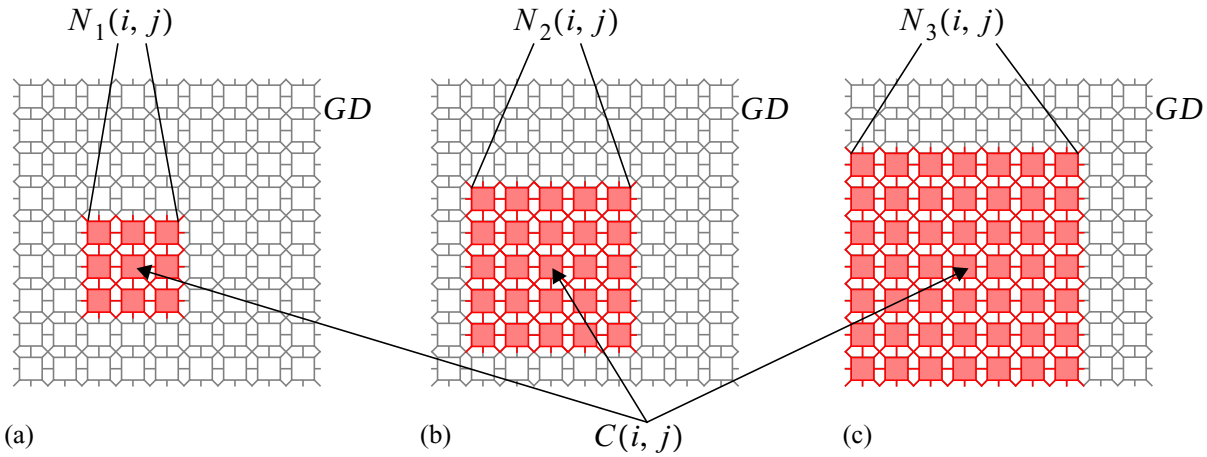


FIGURA 7. Vecindarios de radio (a) $r_c = 1$, (b) $r_c = 2$ y (c) $r_c = 3$ de una celda de CNN.

del radio: $N_{r_c}(i, j)$. La topología de la red queda de esta manera unívocamente definida por la forma de la rejilla (rectangular, hexagonal), el valor del radio y un conjunto de condiciones de contorno que afectan al perímetro de la red (*grid surrounding*, GS). Si consideramos una CNN compuesta por $M \times N$ celdas, desde la perspectiva del procesamiento de señal, la operación de cada celda de la red, $C(i, j)$, puede describirse a partir de tres variables:

- Entrada de la celda: $u_{ij}(t)$, que representa la excitación externa.
- El estado de la celda: $x_{ij}(t)$, que nos da una idea de la energía de la celda en función del tiempo[iii].
- La salida de la celda: $y_{ij}(t)$, obtenida a partir del estado mediante una transformación no lineal:

$$y_{ij} = f(x_{ij}) \tag{8}$$

Así que las CNNs pueden verse como dispositivos para procesamiento de señales multidimensionales, con un vector de entrada $\mathbf{u} = \{u_{ij}\}$, $\forall C(i, j) \in GD$ y un vector de estados iniciales $\mathbf{x}_o = \{x_{ij}(0)\}$, y cuya respuesta está representada por un vector de variables de salida $\mathbf{y} = \{y_{ij}\}$. Estos vectores, para una red bidimensional, pueden escribirse como matrices, de modo que resulta natural llamarlos imágenes de entrada, inicial y de salida. Para unas entrada, estado inicial y topología de la red dadas, el procesamiento realizado por la red viene determinado por:

- Los estados, $\mathbf{x}_s$, y entradas, $\mathbf{u}_s$, de las celdas de la periferia de la red.
- Una *ley de evolución,* idéntica para cada celda, descrita por ecuaciones diferenciales no lineales, o ecuaciones en diferencias finitas no lineales, o por una mezcla de ambas. Estas ecuaciones pueden ser, a priori, de cualquier orden, pero nosotros vamos a considerar aquí únicamente CNNs descritas por ecuaciones de primer orden, como esta:

$$\begin{aligned} \tau \frac{dx_{ij}(t)}{dt} = & -g[x_{ij}(t)] + z_{ij} + \sum_{k=-r}^{r} \sum_{l=-r}^{r} \{A_{ij,kl}[y_{(i+k)(j+l)}] + \\ & + B_{ij,kl}[u_{(i+k)(j+l)}] + C_{ij,kl}[x_{(i+k)(j+l)}] + \\ & + D_{ij,kl}[u_{(i+k)(j+l)}, x_{(i+k)(j+l)}, y_{(i+k)(j+l)}]\} \end{aligned} \tag{9}$$

En esta ecuación, a τ se la conoce como contante de tiempo de la red, $g(\bullet)$ es la función de disipación o término de pérdidas, z_{ij} es el término de *offset*, y $A_{ij,kl}(\bullet)$, $B_{ij,kl}(\bullet)$, $C_{ij,kl}(\bullet)$ y $D_{ij,kl}(\bullet)$ son los operadores de interacción. Estos serán no lineales en el caso más general, sin embargo, muchas tareas de procesamiento requieren exclusivamente interacciones lineales entre celdas, de modo que estos operadores reciben el nombre de pesos de interconexión, tal y como son referidos en el ámbito de las redes neuronales artificiales.

iii. La variable temporal puede ser continua, representada por t, o discreta, representada por n. En este último caso, las señales sólo son válidas en determinados instantes dentro de un conjunto discreto de valores, $t = nT$, donde $n = 0, 1, 2, 3, \ldots$

Un caso particular de CNN es el modelo descrito originalmente por Chua y Yang [Chua88]. Es un modelo de red con interacciones lineales e invariante a traslaciones, o sea con un radio de vecindad constante y un conjunto de pesos de conexión que no depende de la posición de la celda en el arreglo. El término de pérdidas es lineal, la función de la salida es una sigmoide lineal a tramos y los operadores $\mathbf{C}(\bullet)$ y $\mathbf{D}(\bullet)$ son nulos. De modo que la evolución de cada celda de la red viene descrita por:

$$\tau \frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + z_{ij} + \sum_{k=-r}^{r} \sum_{l=-r}^{r} [a_{kl} \cdot y_{(i+k)(j+l)} + b_{kl} \cdot u_{(i+k)(j+l)}] \tag{10}$$

donde:

$$y_{ij} = f(x_{ij}) = \frac{1}{2}(|x_{ij}+1| - |x_{ij}-1|) \tag{11}$$

La Fig. 8 muestra el diagrama esquemático del modelo. Aquí, las variables de entrada, estado y salida de la celda están representadas por las tensiones u_{ij}, x_{ij} y y_{ij}, respectivamente. Las contribuciones de los vecinos y la propia realimentación de la celda se realiza mediante corrientes inyectadas por fuentes controladas por tensión (*voltage controlled current sources*, VCCS), cuyas transconductancias son proporcionales a los valores nominales de los pesos de conexión. Estas corrientes se integran en el condensador de estado C_c. El término de pérdidas que encontramos en la Ec. 10 aparece aquí como la resistencia lineal $1/G_c$, que hace las funciones de transconductancia de normalización para las VCCS. La constante de tiempo de la red se define mediante $\tau = C_c/G_c$. Una fuente de intensidad en DC implementa el término de *offset* o de *bias,* o *umbral de corriente* (z_{ij}). Los operadores de interconexión son lineales, y, puesto que están restringidos a un radio finito, los operadores **A** y **B**, tienen forma matricial. Comúnmente se les denomina máscaras de conexión o *templates*. De modo que tenemos un template de rea-

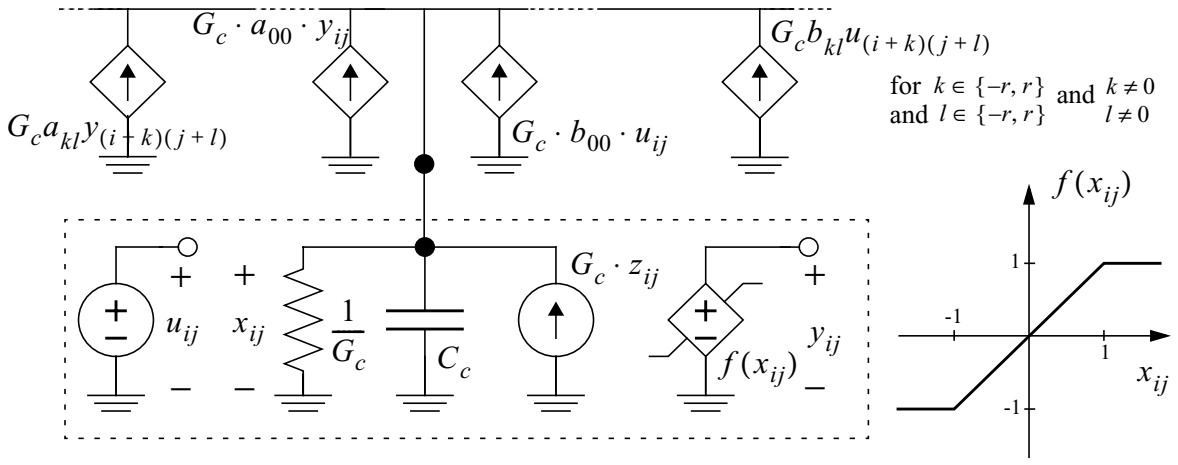


FIGURA 8. Modelo de CNN propuesto originalmente por Chua y Yang.

limentación, $\mathbf{A}$, y otro de control, $\mathbf{B}$. Para un radio de vecindad unitario:

$$\mathbf{A} = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix} \text{ y } \mathbf{B} = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix} \qquad (12)$$

Como sistema dinámico, podríamos evaluar las condiciones en las que una CNN converge a un punto de equilibrio estable (véase Apéndice 1).

Otros modelos de CNN han sido descritos en la literatura. Todos ellos están incluidos en el modelo general, pero, de todos modos, algunos merecen una mención especial. Es el caso del modelo de rango completo de señal (*full-signal-range*, FSR), desarrollado por el Prof. S. Espejo *et al*. [Espe94a]. Este modelo es especialmente adecuado para la implementación VLSI debido a que la variable de estado queda restringida al mismo rango de tensiones que la salida, y no progresa más allá de los raíles de saturación. Esto se consigue mediante un término de pérdidas no lineal:

$$g(x_{ij}) = \lim_{m \to \infty} \begin{cases} mx_{ij} & \text{if} \quad x_{ij} > 1 \\ x_{ij} & \text{if} \quad |x_{ij}| \le 1 \\ -mx_{ij} & \text{if} \quad x_{ij} < -1 \end{cases} \qquad (13)$$

En estas condiciones, no es necesario implementar la función de salida, puesto que las variables x_{ij} e y_{ij}, coinciden, y, lo que es más importante, el valor máximo de $|x_{ij}|$ ya no depende de los valores de los *templates*. Así prevenimos que la variable de estado sea recortada involuntariamente por la característica no lineal intrínseca de los bloques de circuito que utilizaremos en la implementación de la CNN.

Diseño de *templates* de CNN

El patrón de interconexión definido por los templates $\mathbf{A}$ y $\mathbf{B}$, el término de bias z_{ij}, $\forall C(i,j) \in GD$ y las condiciones iniciales y de contorno, determinan la evolución y el estado final de la CNN. Desde el punto de vista del procesamiento de señal, estos parámetros determinan el tipo de operación que se realiza sobre una imagen de entrada $\mathbf{u}$, para obtener una imagen de salida $\mathbf{y}$, asumiendo un estado inicial de la red $\mathbf{x}(0)$. De este modo, la CNN puede entenderse como un sistema de procesamiento con una dinámica espacio-temporal interna programable, que transforma señales bidimensionales (en el caso que nos ocupa) de entrada en una señal bidimensional también de salida. Los elementos de los *templates* constituyen el programa analógico de la CNN.

El diseño de templates para realizar una determinada operación puede realizarse de manera directa, expresando dicha operación en términos de reglas dinámicas locales, de modo que los coeficientes puedan ser derivados analíticamente [Chua91]. Como ejemplo podemos fijarnos en la

FIGURA 9. Detección de componentes conectados de la imagen de 64×64 pixels de un carácter manuscrito.

detección de componentes conectados (*connected component detection* CCDet)[Mats90], que consiste en contabilizar el número de segmentos conectados en cada línea de una imagen binaria (Fig. 9). Puesto que la propagación de la señal ocurre en la dirección horizontal, la operación pude definirse a partir de una CNN unidimensional. Por otro lado, puesto que propagamos la información, no estamos interesados en una influencia constante de la entrada, por lo que el *template* de control será nulo. A continuación, se evalúa el signo de la derivada de x_{ij} para una celda genérica $C(i, j)$ para cada situación diferente que puede enfrentar en la evolución de la red, en total ocho, dando lugar a otras tantas inecuaciones. Si se resuelven según se describe en [Häng99] (véase Apéndice 1):

$$\mathbf{A} = \begin{bmatrix} a & a+1 & -a \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{z} = 0 \tag{14}$$

donde $a > 0$. Para seleccionar finalmente unos valores particulares, convendría analizar la robustez de la implementación [King96].

Una aproximación diferente al diseño de *templates* de CNN es la aplicación de técnicas de aprendizaje [Noss93]. Este método se basa en un conjunto de L pares entrada-salida, que reciben el nombre de conjunto de entrenamiento (*training set*) y de los cuales no es fácil extraer un conjunto de reglas dinámicas de forma explícita. Cada pareja está compuesta por un vector de entrada (o de estado inicial), $\mathbf{u}^l$ (ó $\mathbf{x}^l(0)$), y un vector que representa la salida esperada $\mathbf{d}^l(\infty)$. Si definimos el vector de parámetros $\mathbf{p} = [a_{-1,-1}, a_{-1,0}, \ldots, b_{1,0}, b_{1,1}, z]^T$, el objetivo del aprendizaje es encontrar un vector de parámetros $\mathbf{p}$ tal que la salida de la red gobernada por dichos parámetros $\mathbf{y}^l(\infty)$ corresponda con la salida esperada $\mathbf{d}^l(\infty)$. Esto para todos los pares pertenecientes al *set* de entrenamiento. En otras palabras, tendremos que minimizar una función coste, o error, definida en el espacio de los parámetros a partir del training set. La hipersuperficie de error $\varepsilon(\mathbf{p})$ puede definirse como la suma de los errores individuales correspondientes a cada celda, evaluados para todos los pares entrada-salida del *set* de entrenamiento:

$$\varepsilon(\mathbf{p}) = \sum_{l=1}^{L} \varepsilon^l(\mathbf{p}) = \sum_{l=1}^{L} \sum_{i=1}^{M} \sum_{j=1}^{N} \left| d_{ij}^l(\infty) - y_{ij}^l(\infty) \right|^{\nu} \tag{15}$$

Sea cual sea el método mediante el cual se han obtenido los parámetros de la CNN, debe quedar claro que cada conjunto de *templates*, junto con las condiciones de contorno, determina el tipo de procesamiento que esta va a ejecutar sobre la entrada. Veamos algunos ejemplos, resultado de la operación de un solo *template*

Operación	A	B	z
Difusión (filtro paso de baja espacial)	$\begin{bmatrix} 0.25 & 0.50 & 0.25 \\ 0.50 & -2.00 & 0.50 \\ 0.25 & 0.50 & 0.25 \end{bmatrix}$	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	0.00
Umbralizado (saturación de la salida)	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 2.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	0.00
Detección de bordes (para entrada binaria)	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 2.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	$\begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & -2.00 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix}$	−1.50

TABLA 4. Ejemplos de templates lineales para procesamiento básico

lineal sobre la imagen de entrada. En la parte superior izquierda de la Fig. 10 tenemos la imagen inicial, de 512×512 pixels en una escala de grises de 256 niveles. La primera operación que realizamos es una difusión, controlada por los templates reflejados en la primera fila de la Tabla 4. que es equivalente a realizar un filtrado espacial de paso de baja [Jähn99a]. Como resultado obtenemos la imagen expuesta en la Fig. 10(b). El segundo ejemplo consiste en el umbralizado y binarización de la imagen difundida. Al ser una operación de pixel, los elementos de los templates que reflejaran alguna interacción con el vecindario son nulos. Como resultado tenemos la Fig. 10(c). Finalmente hemos realizado sobre esta última imagen una detección de bordes (Fig. 10(d))

La máquina universal de cómputo basada en CNNs

A partir del paradigma de las redes neuronales celulares [Chua93], ha sido posible definir la estructura de un procesador algorítmico paralelo y programable con la potencia de cálculo de un supercomputador en un solo chip: la máquina universal de CNNs (CNNUM) [Rosk93a]. La CNNUM es una arquitectura de cómputo híbrida que combina una dinámica espacio-temporal con eventos discretos [Rosk88]. Se ha demostrado que es universal en el sentido de Turing [Crou96]. Una de las características más interesantes de la CNNUM es su adecuación para la implementación VLSI, dada la regularidad espacial de su estructura y la conectividad local de sus procesadores básicos. Estas unidades elementales de proceso contienen un núcleo que implementa un nodo de la CNN, memorias analógicas y digitales locales, bloques de lógica programable y un control de la topología interna mediante *switches*. Algunas implementaciones monolíticas han sido reportadas [Cruz98] [Espe96b] [Paas97]y más recientemente [Liña98].

La Fig. 11 muestra un diagrama conceptual de la CNNUM y sus principales bloques. En primer lugar encontramos la matriz de procesadores básicos en paralelo. Estas celdas básicas contienen un núcleo de CNN que implementa una red lineal, de una capa, invariante a traslaciones. En un

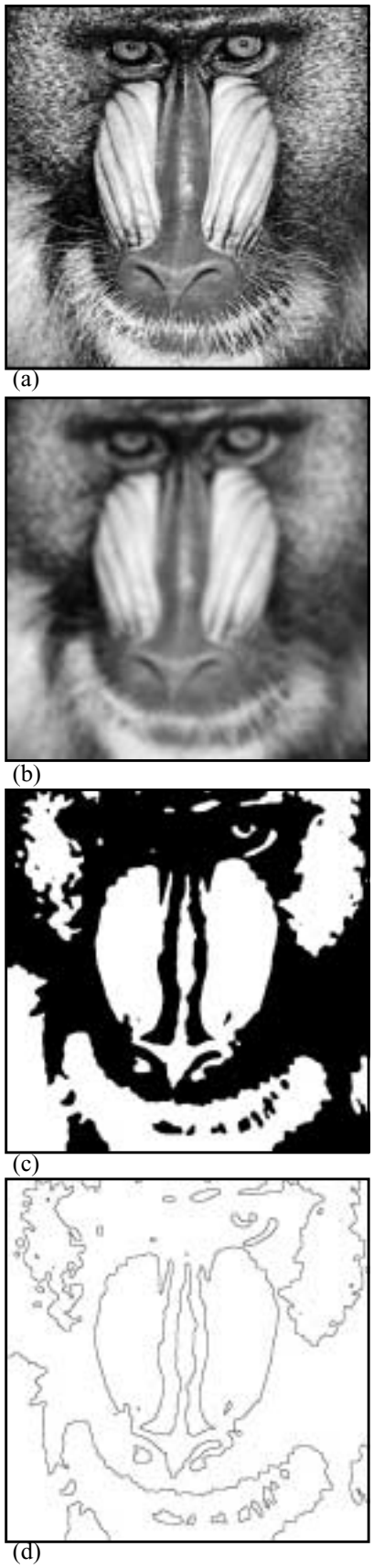


FIGURA 10. Ejemplo de operaciones de CNN con un único template lineal.

caso más general se incluirían operadores de conexión no lineales, una dinámica de orden superior, etc. Junto a este núcleo encontramos la unidad de procesamiento lógico local (*local logic unit*, LLU). El almacenamiento de resultados intermedios se realiza en las memorias locales analógicas y lógicas (*local analog* y *local logic memories*, LAMs y LLMs). El intercambio de datos con el exterior y el control de la operación son soportados localmente por la unidad de comunicación y control local (*local communication and control unit*, LCCU) y por la interfaz local para la salida (*local analog output unit*, LAOU) que puede incluir algunas habilidades aritméticas, como la adición o substracción de imágenes, etc. Fuera de la matriz de procesadores básicos, existe una unidad global de programación y control que a su vez contiene un bloque dedicado al almacenamiento del programa —que consiste en la memoria de programa analógico (*analog program registre*, APR), la memoria del programa lógico (*logic program registre*, LPR) y el registro de la configuración de la red (*switch configuration registre*, SCR)— y por otro lado un bloque dedicado al control, que decodifica las microinstrucciones almacenadas en la memoria de programa y las transmite al arreglo de procesadores, que es la unidad global de control (*global analogic control unit*, GACU). Nótese que la arquitectura de la CNNUM es la de un procesador SIMD sólo que en cada procesador unitario conviven bloques de procesamiento de señal analógicos junto con la lógica programable. Esta CNNUM puede realizar multitud de operaciones de procesamiento paralelo diferentes sin más que definir el programa analógico, el programa lógico y el flujo de datos dentro de la celda, o sea, los coeficientes que rigen las evoluciones de la CNN, las operaciones que tendrán lugar entre imágenes binarias y las configuraciones de la red para permitir el paso de datos de un bloque a otro. Para ilustrar las capacidades de la CNNUM consideremos un ejemplo de restauración de imágenes.Partimos de una imagen en tonos de gris (256 niveles), corrompida por un ruido impulsivo con una probabilidad de ocurrencia de un 20% (Fig. 13(a)). Este artefacto visual, también conocido como “ruido de sal y pimienta”, es común en los sistemas de visualización electrónicos y puede deberse a la saturación de dispositivos, a errores en la transmisión o adquisición, etc. Para restaurar la imagen, o al menos recuperar la mayor parte de la información contenida en el original, vamos a hacer uso de dos cosas. En primer lugar, del hecho de que la información en imágenes

naturales está bastante correlacionada [Gonz87]. Por otro lado, vamos a utilizar una propiedad que suele estar incluida en la CNNUM que es la máscara de estado fijo (*fixed state map*, FSM), descrito en [Rosk93a], y que no es más que la ya citada inhibición de la operación en ciertos caminos de señal mediante el enmascaramiento local de instrucciones globales en arquitecturas SIMD [Pirs98].

Teniendo en cuenta lo que hemos dicho acerca de la concentración de la información relevante en las bajas frecuencias espaciales, puede pensarse que un filtrado paso de baja eliminaría el ruido impulsivo, concentrado en las más altas frecuencias espaciales. Para realizar este filtrado paso de baja bidimensional utilizaremos el template de difusión expuesto en la Tabla 4.

Pues bien, si pasamos la imagen original por un filtro paso de baja, implementado mediante el *template* de difusión en una CNN, el resultado no es nada alentador, como puede verse en la Fig. 13(b), ya que los bordes de la imagen, componentes de alta frecuencia espacial, se han deteriorado notablemente. El filtro paso de baja ha cortado las componentes de alta frecuencia, haciendo que los bordes se difuminen. Al mismo tiempo, la información correspondiente a los impulsos se expande hacia sus vecinos, creando manchas inexistentes en la imagen original. En procesamiento digital, este problema se resuelve mediante un filtro de mediana, en lugar de un promediado, que es lo que hemos hecho. Sin embargo, obtener la mediana de un bloque de 3×3 pixels —o sea, en este grupo de 9 pixels,

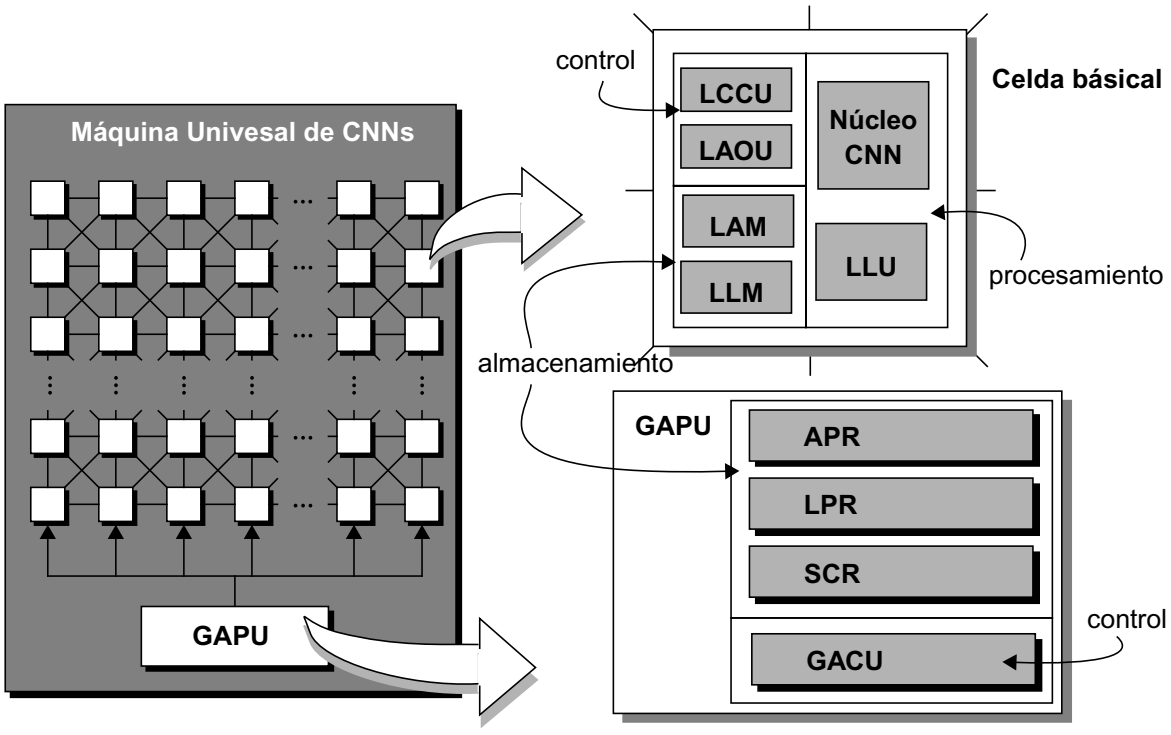


FIGURA 11. Arquitectura de la CNNUM, de la celda básica y de la GAPU.

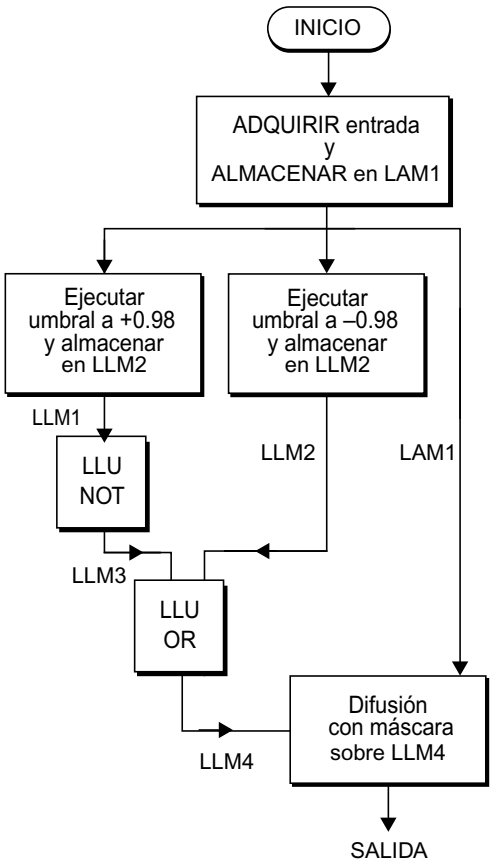


FIGURA 12. Diagrama de flujo para la restauración de una imagen corrupta con impulsos mediante la CNNUM.

encontrar el pixel cuya intensidad luminosa deja tantos pixels por arriba como por abajo— es una tarea con un flujo computacional bastante irregular, lo cual no es nada bueno para esta estructura altamente paralela.

Mediante la CNNUM podemos emular el procesamiento de los filtros de mediana. Por ejemplo, podemos prevenir los pixels no afectados por el ruido de recibir ninguna influencia, y, al mismo tiempo, sustituir el valor de la intensidad luminosa en aquellos pixels corruptos por la media de sus vecinos. Para esto nos puede servir el FSM. Supongamos que la imagen inicial es adquirida y almacenada en una memoria analógica (LAM1). Realizamos tres transformaciones diferentes sobre esta imagen (Fig. 12). Primero, umbralizamos a 0.98, que es el 99% del rango completo de la señal, obteniendo una máscara que incluye todos los impulsos "blancos", almacenándola en LLM1. Al invertir, mediante la LLU, obtenemos la imagen binaria de la Fig. 13(c), la cual se guarda en LLM3. Asimismo, umbralizamos la imagen corrupta -0.98, generando la máscara de los impulsos "negros" (Fig. 13(d)). Esta se almacena en LLM2. Luego se combinan LLM3 y LLM2 para obtener la máscara de todos los impulsos detectados (Fig. 13(e)) y se realiza una difusión en los pixels marcados pixels, impidiendo la modificación de los que no han sido detectados como corruptos. Como resultado obtenemos la imagen Fig. 13(f), la cual aparece limpia del ruido impulsivo y con sus bordes originales casi intactos.

2. Diseño de sistemas VLSI de procesamiento basados en CNNs

2. 1. Limitaciones tecnológicas sobre el tamaño de la imagen

Las mayores dificultades que encontramos en la implementación VLSI de sistemas de procesamiento masivo, paralelo y analógico son de tipo tecnológico. Dos fenómenos diferentes imponen restricciones sobre la densidad máxima de celdas y el tamaño del chip. Por un lado, la precisión, que en circuitos analógicos está fuertemente relacionada con el llamado *yield* (que puede traducirse por rendimiento de la producción) paramétrico [Pelg89], exige la implementación con dispositivos extensos en área. Por otro, el *yield* aleatorio, relacionado con la aparición de defectos en la fabricación, conmina a realizar chips más pequeños.

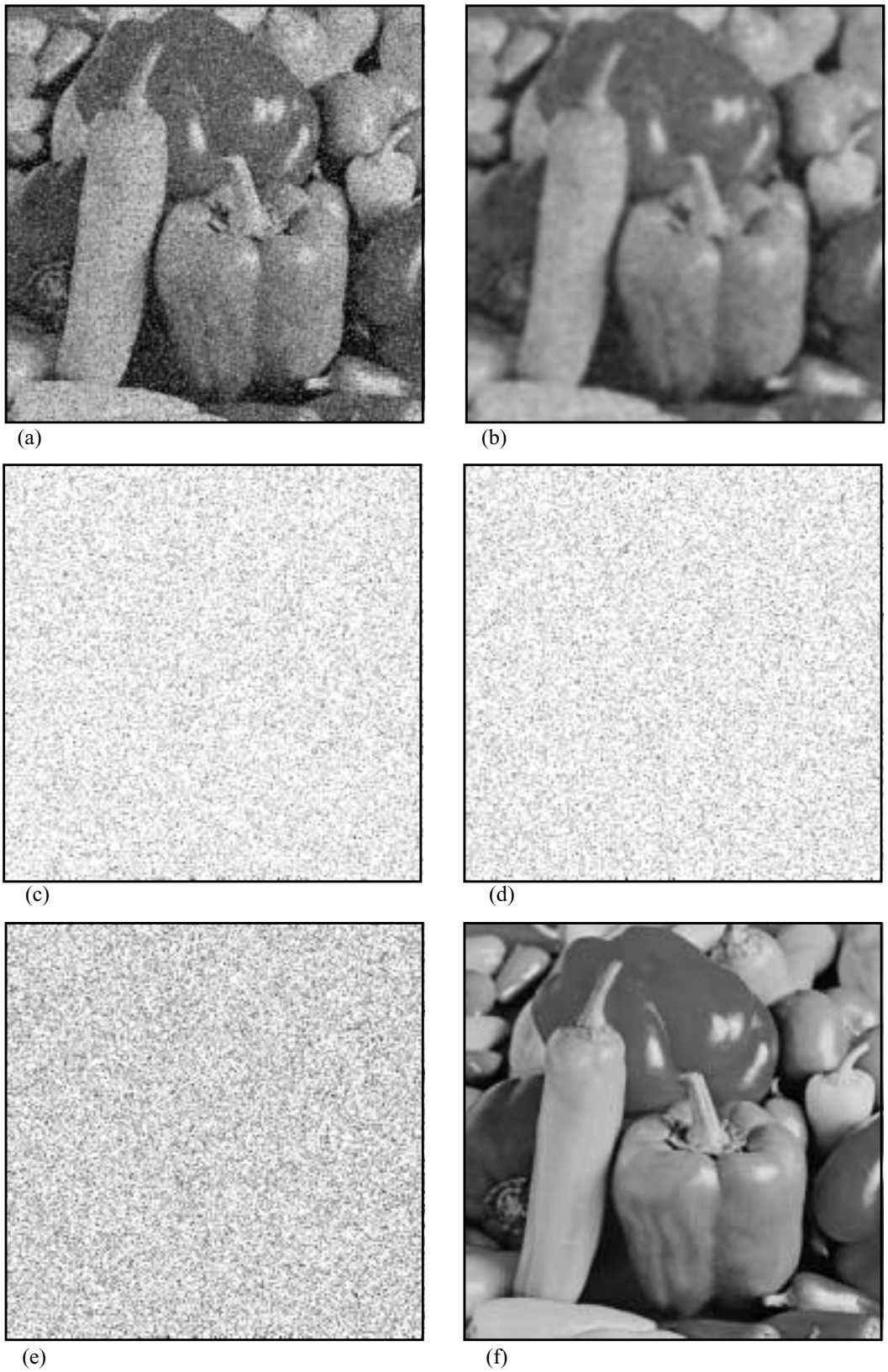


FIGURA 13. Ejemplo de procesamiento: restauración de imágenes corruptas por ruido impulsivo..

El efecto combinado de estos fenómenos resulta en un máximo en el número de procesadores básicos que pueden ser integrados en un mismo chip.

Defectos aleatorios en la fabricación de chips

El proceso de fabricación de circuitos integrados (ICs) esta sujeto a una serie de factores que pueden comprometer seriamente el rendimiento de la producción (*yield*). Una de las mayores fuentes de problemas es la incidencia de defectos aleatorios, los cuales surgen debido a que el proceso tiene un grado de limpieza finito. Un defecto es una distorsión local de las estructuras de silicio prescritas, causada principalmente por la presencia de una partícula de polvo, la contaminación del equipamiento de fabricación, etc. La incidencia de defectos puntuales, llamada así puesto que suelen tener ámbito local, es de naturaleza aleatoria. Se caracteriza mediante una magnitud estadística que relaciona el tamaño y la frecuencia de los defectos con la unidad de área, o sea, la densidad de defectos, D. Sin embargo, esta densidad de defectos no está muy bien definida, y habitualmente es inferida a partir de las medidas sobre el *yield*. Así que, la interacción entre los defectos y las estructuras de silicio genera fallos, y estos fallos provocan el descenso del *yield*. De modo que el *yield* decrece debido a la incidencia de defectos fatales, fallos, y un defecto es fatal si provoca un detrimento del *yield* [Kore89]. Las estimaciones del *yield* basadas en estas consideraciones constituyen lo que se conoce como *yield* aleatorio. Es conveniente distinguirlo del yield paramétrico, relacionado con la desviación de los parámetros del proceso, que veremos después. A pesar de la débil definición del *yield* aleatorio con la que contamos, se han desarrollado modelos muy aceptables del mismo, por ejemplo, la fórmula de Poisson [Warn74]:

$$Y_{\text{random}}(A_{\text{chip}}) = \exp(-DA_{\text{chip}}) \tag{16}$$

donde D es la densidad de defectos y A_{chip} representa el área del chip. Asumiendo una distribución estadística de los defectos distinta, obtenemos la fórmula binomial negativa [Stap91]:

$$Y_{\text{random}}(A_{\text{chip}}) = \left(1 + \phi\frac{DA_{\text{chip}}}{\alpha}\right)^{-\alpha} \tag{17}$$

aquí, D es la densidad de defectos, A_{chip} el área del chip, α es el grado de arracimamiento de los defectos, y ϕ es una medida de la probabilidad de un defecto de convertirse en fatal. Tanto D como α dependen del proceso, mientras que A_{chip} y ϕ dependen del diseño. En concreto, ϕ depende fuertemente de la distribución y el tamaño de las zonas críticas del *layout* del circuito. Para un *layout* denso, lo cual aplica en el caso de implementaciones analógicas VLSI de CNNs, este factor puede ser grande (más del 10% del área puede ser crítica para el tamaño medio de los defectos). Lo cual resulta en una pérdida considerable del yield para grandes chips, según predice la fórmula binomial negativa. Así que podemos concluir que, para

obtener un rendimiento de la producción razonable, el área del chip debe mantenerse por debajo de un valor máximo. Usando la fórmula binomial negativa, este máximo es:

$$A_{\text{chip}} \leq \frac{\alpha}{D\phi}\left(\frac{1}{\sqrt[\alpha]{Y_{\text{random}}}} - 1\right) \tag{18}$$

Indirectamente esto supone un máximo en el número de celdas básicas de CNN que pueden integrarse en un mismo chip.

Desviaciones de los parámetros del proceso

Otra fuente de pérdidas del *yield* está en la desviación, con respecto a los valores nominales, que son los que va a manejar el diseñador, de los parámetros del proceso de fabricación. Estos parámetros varían de un lote de fabricación a otro, de una oblea de silicio a otra dentro del mismo lote, o incluso de un chip a otro dentro de la misma oblea. Debido a la variación de estos parámetros del proceso, los parámetros eléctricos derivados de los mismos, también sufren una desviación. La influencia de estas desviaciones es de dos tipos, están, por un lado, las variaciones que afectan igualmente a todos los dispositivos de un mismo chip, y que resultan en una desviación de los valores medios de estos parámetros con respecto a los promedios que nominalmente se utilizaron durante la fase de diseño. y por otro lado, las desviaciones debidas a la granularidad del proceso a lo largo de un mismo chip, que se manifiesta en forma de diferencias entre las propiedades eléctricas de dos dispositivos que inicialmente fueron diseñados iguales. Esto es lo que se llama desapareamiento (*mismatch*).

Respecto de los cambios que podríamos llamar *de gran escala*, variaciones en los parámetros que afectan igualmente a todo el chip, lo único que el diseñador puede hacer es verificar el funcionamiento del circuito en las esquinas de la tecnología, o sea, para los valores extremos que el fabricante consigna de los parámetros, los cuales corresponderían con los peores casos de ciertas propiedades de los dispositivos. No hay ninguna actuación razonable sobre las dimensiones del chip que permita mitigar el efecto de las desviaciones a gran escala de los parámetros. De modo que no puede establecerse, basándonos en este fenómeno, una relación entre el tamaño del chip y el *yield*. Sin embargo, esta relación sí puede encontrarse si, en lugar de utilizar las desviaciones de los parámetros observadas entre un *batch* de fabricación y otro, o entre una oblea y otra dentro del mismo *batch*, o entre un chip y otro dentro de la misma oblea, consideramos el efecto de la desviaciones de los parámetros dentro de un mismo chip. Pensemos entonces en dos dispositivos que, perteneciendo al mismo chip, han sido diseñados originalmente idénticos. Estos dispositivos nunca van a mostrar exactamente las mismas características debido a las fluctuaciones de pequeña escala de los parámetros. Estas variaciones dentro del chip tienen un efecto perjudicial en la operación de los circuitos, ya que existe una gran variedad de circuitos analógicos lineales y no lineales cuya ope-

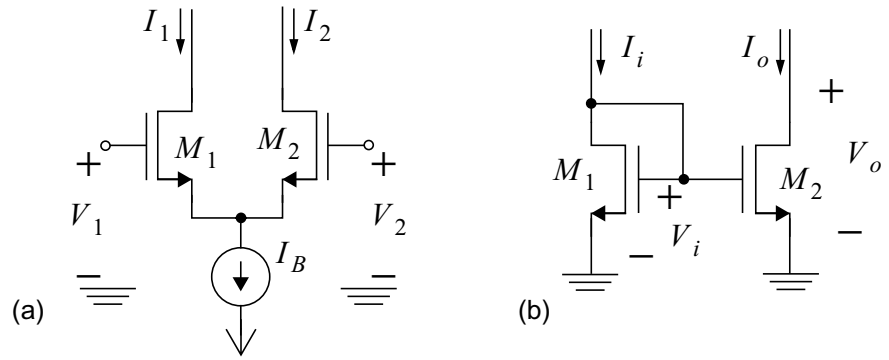


FIGURA 14. (a) Par diferencial MOS y (b) espejo de corriente MOS.

ración descansa en la cancelación de no idealidades entre dos dispositivos apareados. Por ejemplo, consideremos un par diferencial construido con dos transistores MOS (Fig. 14(a)) [Geig90]. Si definimos la tensión de entrada como la diferencia entre las tensiones aplicadas a cada una de la s puertas de los transistores del par, $V_i = V_1 - V_2$, y la corriente de salida como la intensidad de corriente diferencial $I_o = I_1 - I_2$, entonces, considerando estos transistores como perfectamente apareados, lo que supone que tienen la misma transconductancia y la misma tensión umbral[iv]:

$$I_o = \beta V_i \sqrt{2I_B/\beta - V_i^2} \tag{19}$$

mientras que la tensión diferencial de entrada se mantenga dentro de ciertos límites $|V_i| < \sqrt{I_B/\beta}$. De otro modo la intensidad de corriente de salida se satura a I_B o $-I_B$. Pues bien, si tenemos en cuneta que existe una cierta diferencia entre los valores de la tensión umbral de ambos transistores, $V_{T_1} - V_{T_2} = \Delta V_T$, y entre los factores de corriente $\beta_1 - \beta_2 = \Delta\beta$, entonces la Ec. 19 se transforma en:

$$I_o \approx \beta(V_i + \Delta V_T)\sqrt{\frac{2I_B}{\beta} - \left(1 + \frac{\Delta\beta}{\beta}\right)(V_i + \Delta V_T)^2} \tag{20}$$

lo cual significa la aparición de un cierto offset en la corriente de salida que antes, en el caso ideal de apareamiento perfecto, no existía ($I_o(V_i = 0) \neq 0$). Otro ejemplo de bloque analógico cuya operación se degrada por la inevitable presencia del desapareamiento es el espejo de corriente (Fig. 14(b)). Idealmente, si $\beta_1 = \beta_2 \equiv \beta$ y $V_{T_1} = V_{T_2} \equiv V_T$, la corriente de salida es igual a la de entrada $I_o = I_i$ mientras que se mantenga a ambos transistores en saturación. Pero si consideramos la existencia de un cierto desapareamiento entre ambos, llegamos a [Lake94]:

$$I_o \approx I_i\left(1 - \frac{\Delta\beta}{\beta} - \frac{2\Delta V_T}{V_i - V_T}\right) \tag{21}$$

Si obtenemos modelos de circuito que tengan en consideración los efectos del

iv. No estamos considerando la constante de efecto substrato, por simplificar, pero esta puede ser una importante fuente de error de desapareamiento.

desapareamiento de los dispositivos, será posible incluir una estimación del *yield* paramétrico en el flujo de diseño. Esto no va a ser sencillo ya que el análisis estadístico de los modelos conlleva un uso extensivo de la simulación y, si esta se realiza sobre descripciones de muy bajo nivel la demanda computacional es enorme, mientras que si se utilizan modelos de alto nivel que no sean suficientemente detallados, se corre el riesgo de perder fiabilidad en los resultados [Carm99a]. Este va ser un problema que exploraremos más adelante. Aquí nos vamos a limitar a consignar la relación empírica entre el *yield*, ahora llamado paramétrico, y la desviación de los parámetros del circuito, encontrada en el curso de nuestras simulaciones, que es:

$$Y_{\text{param}}(\sigma) = \frac{1}{2}\{1 - \tanh[m_0(\sigma - \sigma_0)]\} \tag{22}$$

donde σ_0 y m_0 han sido estimados mediante un ajuste no lineal de los resultados del análisis de Monte Carlo.

Una vez establecida la relación entre el yield y las desviaciones de los parámetros, faltaría encontrar el nexo entre el área del chip y estas desviaciones. Para ello contamos con el modelo de Pelgromm *et al*. en [Pelg89], que relaciona el desapareamiento con el área, la orientación y la distancia entre ambos dispositivos. Este modelo es válido si los procesos que dan lugar a determinado parámetro afectan a toda el área del dispositivo. Así, si el parámetro P satisface los requerimientos del modelo, y, para dos dispositivos rectangulares y con la misma orientación adquiere los valores P_1 y P_2, entonces, la varianza del desapareamiento entre estos valores ($\Delta P = P_1 - P_2$) viene dada por:

$$\sigma^2(\Delta P) = \frac{A_P^2}{WL} + S_P^2 D \tag{23}$$

donde W y L son la longitud y anchura nominales de los dispositivos, D es la distancia entre ellos y A_P y S_P son constantes del proceso. Pueden incluirse correcciones a este modelo para tener en cuenta los efectos de borde. Puesto que, en general, los dispositivos que, dentro del bloque analógico, deben estar apareados se encuentran relativamente cerca, sólo nos interesa el término dependiente del área en la Ec. 23. Si consideramos que el área total del chip es un múltiplo del área del dispositivo elemental, WL, en un ejercicio de extrapolación ciertamente especulativo, podemos decir que $\sigma(\Delta P) \propto 1/\sqrt{A_{\text{chip}}}$, y por tanto, reescribir la expresión aproximada del *yield* paramétrico como una función del área del chip:

$$Y_{\text{param}}(A_{\text{chip}}) = \frac{1}{2}\left\{1 - \tanh\left[k_0\left(\frac{1}{\sqrt{A_{\text{chip}}}} - \frac{1}{\sqrt{A_0}}\right)\right]\right\} \tag{24}$$

donde k_0 y A_0 se obtienen a partir de los parámetros de ajuste σ_0 y m_0, el parámetro tecnológico A_P, y la relación entre A_{chip} y WL.

Tamaño óptimo del chip

Las Eqs. 17 y 24 relacionan las dos estimaciones del *yield* —una basada en la incidencia de defectos aleatorios y otra debida a la naturaleza estadística de los parámetros del proceso de fabricación— con el área total del chip. Podemos ver que un incremento (o alternativamente una disminución) de A_{chip} tiene efectos contrapuestos en ambas estimaciones. Por un lado, mientras mayor es el chip, más probable es que un defecto fatal aparezca, aunque al mismo tiempo, mayores serán los dispositivos analógicos y por tanto más precisos, por lo cual mejor será la estimación del *yield* paramétrico. Si A_{chip} disminuye, decrece el *yield* paramétrico, pero aumenta el aleatorio, puesto que disminuye la probabilidad de que aparezcan un defecto fatal. Puesto que un fallo debido al desapareamiento y un fallo debido a un defecto aleatorio son dos fenómenos descorrelacionados, si F_{random} y F_{param} son las probabilidades de que ocurran, entonces la probabilidad de que una muestra del chip sufra un fallo por alguna de estas causas es:

$$F_{\text{total}} = F_{\text{random}} + F_{\text{param}} - F_{\text{random}}F_{\text{param}} \tag{25}$$

Y como el *yield* total es la probabilidad de que una muestra del chip, seleccionada de entre todas las fabricadas, opere correctamente, tendremos:

$$\begin{aligned} Y_{\text{total}} &= 1 - F_{\text{total}} = 1 + F_{\text{random}}F_{\text{param}} - F_{\text{random}} - F_{\text{param}} = \\ &= (1 - F_{\text{random}})(1 - F_{\text{param}}) = Y_{\text{random}}Y_{\text{param}} \end{aligned} \tag{26}$$

Por tanto, una estimación global del yield, en función del área del chip, puede obtenerse multiplicando los dos factores calculado previamente:

$$Y_{\text{total}} = \frac{1}{2}\left(1 + \phi\frac{D \cdot A_{\text{chip}}}{\alpha}\right)^{-\alpha}\left\{1 - \tanh\left[k_0\left(\frac{1}{\sqrt{A_{\text{chip}}}} - \frac{1}{\sqrt{A_0}}\right)\right]\right\} \tag{27}$$

Dada la dependencia antagónica de Y_{random} y Y_{param} con el área del chip, es posible encontrar un tamaño óptimo del mismo para un diseño particular. En esta ocasión hemos considerado una densidad de defectos de $D = 2$ def/cm^2, un factor de arracimamiento de defectos de $\alpha = 2$ y una probabilidad de defecto fatal $\phi = 0.15$, así como las tablas en [Pelg89] y las simulaciones en [Carm99a] para caracterizar el yield paramétrico. El máximo yield encontrado es 69.96%, y se obtiene para un área del chip de 1.175cm^2. Esto está muy por encima del tamaño habitual de un chip analógico sencillo, con una funcionalidad cerrada, pero no es un tamaño raro para un circuito APAP complejo.

2. 2. Multiplexado de sistemas basados en CNNs

Sistemas de CNN multichip

A pesar de las limitaciones tecnológicas, encontramos ejemplos de chips VLSI analógicos que realizan procesamiento en el plano focal y en

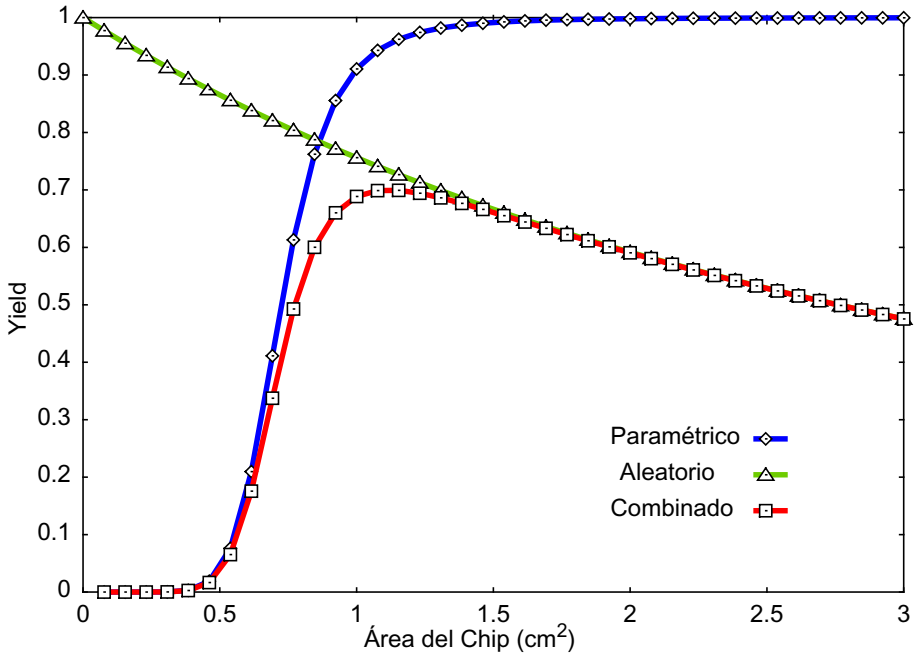


FIGURA 15. Estimación combinada del yield aleatorio y paramétrico frente al área del chip.

tiempo real de la imagen [Koch96]. Por otro lado, las capacidades de la CNNUM sólo se muestran de manera natural en una implementación monolítica. Así que nos plantearemos la superación de estas limitaciones mediante actuaciones en el nivel de sistema, como el multiplexado espacial o temporal del hardware, para realizar una implementación de la CNNUM. En una realización multichip, la interconexión de chips con un número reducido de celdas de CNN permitirá construir redes de gran tamaño. Cada módulo elemental operará sobre una fracción de la imagen original, la cual será, de esta manera, procesada en paralelo.A pesar de la enorme potencia de cómputo que este sistema sería capaz de desarrollar, existen ciertos aspectos de su implementación que no están bastante depurados. Por un lado, añadidos a los problemas derivados de las fluctuaciones de los parámetros dentro de un mismo chip, nos encontramos con el desapareamiento entre las diferentes muestras del mismo chip. Los parámetros de la red se desvían en las diferentes sub-redes, pudiendo causar una pérdida notable del *yield*. Como ejemplo, la Fig. 16 muestra el caso de una red de 32×32 celdas compuesta a partir de módulos de 8×8 CNNs de Chua y Yang. No se han considerado desviaciones dentro de las sub-redes. Como vemos, la desviación de uno de los elementos de los *templates*, en este caso el término de *offset*, de un chip a otro tiene como consecuencia una considerable pérdida de *yield*. Para evitar esto pueden adoptarse diferentes estrategias: rediseñar los templates para una implementación más robusta [Fold98], definir los parámetros de la red de manera independiente al proceso [Angu98] o corregir las desviaciones de manera adaptativa [Espe96b].

Por otro lado, la realización multichip tiene un impacto considerable en el coste de fabricación del sistema [v]. Por ejemplo, el sistema prototipo compuesto por 720 celdas descrito en [Sale97] contiene 20 módulos de 6×6 celdas en una configuración 5×4. Si pretendiéramos procesar completamente en paralelo una imagen de 640×480 pixels necesitaríamos más de 8×10^3 chips. Suponiendo que usára-

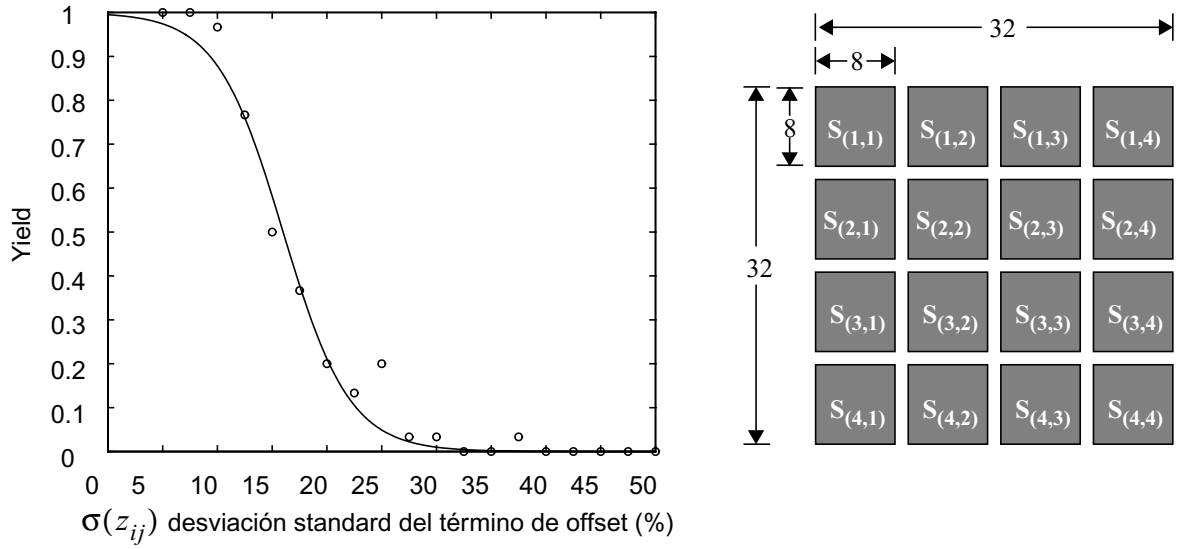


FIGURA 16. Yield frente a la desviación standard del término de offset, para un template de detección de bordes, en la arquitectura multichip de 32×32 celdas indicada.

mos módulos de 64×64 celdas, el número total de muestras necesarias aún sería de 75. El montaje de semejante sistema puede estar justificado por motivos científicos, pero resulta irrealizable desde el punto de vista de la producción en masa para un mercado amplio. También en el mismo sentido apuntarían las conclusiones derivadas del análisis del número de interconexiones entre los diferentes módulos que se requieren. Los costes de fabricación del sistema dependen del tipo de encapsulado elegido y de la tecnología de montaje, y no todas las cápsulas[vi] tienen las terminales suficientes ni todas las tecnologías de montaje [vii] ofrecen una densidad de cableado suficiente. Puede demostrarse que cada módulo de $M \times N$ celdas necesitará $2(M + N - 2)$ nudos de entrada y $2(3M + 3N - 2)$ conexiones de salida, lo que hace un total de $8(M + N - 1)$ terminales sólo para conectar las celdas de los bordes. Una CNN multichip de 640×480 celdas, compuesta por módulos de 64×64 necesitaría 1016 pins por cada uno de ellos y un total de 76.2×10^3 interconexiones.

Multiplexado temporal de chips de CNN

Un enfoque diferente consistiría en el multiplexado del *hardware* en el tiempo. La extraordinaria potencia de cálculo de la CNNUM [Chua96] permitiría que un único chip procesara una imagen completa operando sobre diferentes fracciones de la imagen sucesivamente. Una frecuencia de refresco de 60Hz, adecuada para aplicaciones de video de alta calidad [Poyn96], supondría, para un tamaño de imagen de 640×480 pixels, un

v. Aquí no vamos a definir con exactitud el coste de la realización del sistema, sino que nos limitaremos a realizar apreciaciones cualitativas basadas en características como el número de chips necesarios, el tipo de encapsulado, etc.

vi. p. ej. Dual-in Parallel, Pin-Grid-Array, Ball-Grid-Array

vii. p. ej. Printed-Circuit-Board, Chip-On-Board, Multi-Chip-Modules

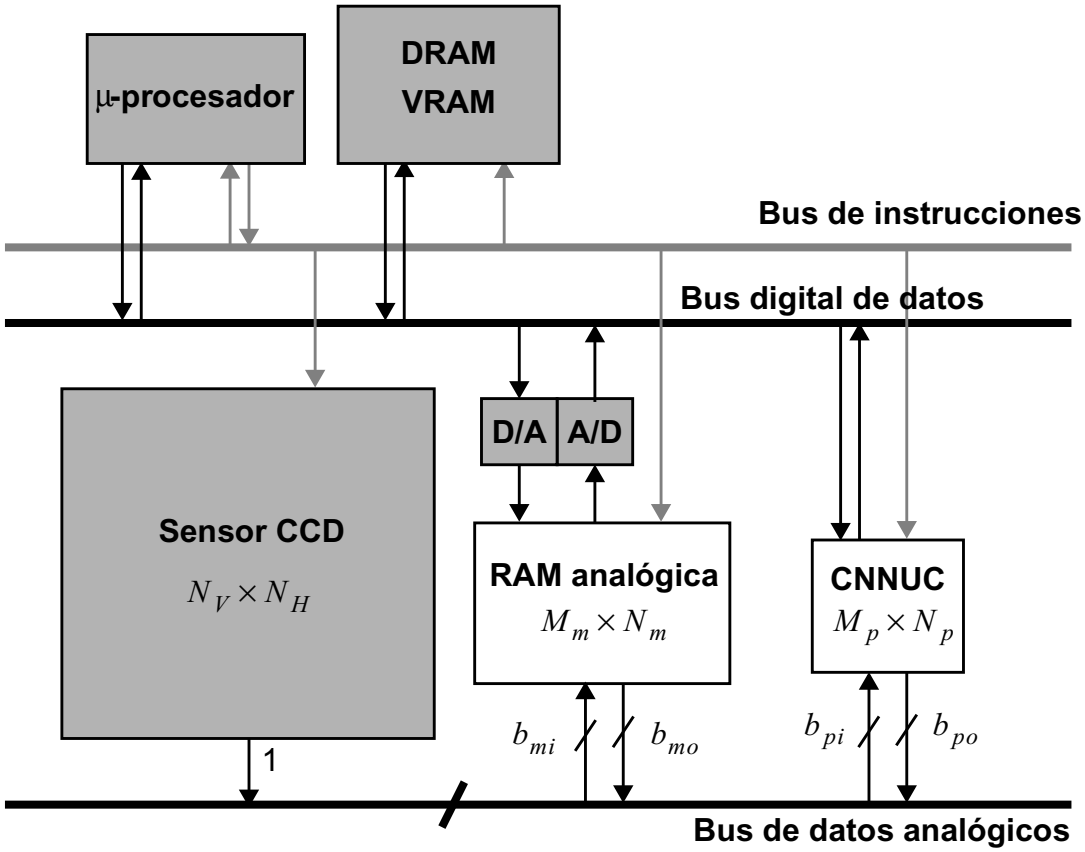


FIGURA 17. Diagrama de la arquitectura del *chipset* de CNN.

flujo de datos de 18.4×10^6 pixels por segundo. Procesar en tiempo real este flujo de pixels requiere un tiempo de procesamiento por pixel de 54ns. Si empleamos un chip de CNN de 64×64 celdas, y permitimos un solapamiento de 2 pixels entre dos sub-imágenes adyacentes, el tiempo total de procesamiento de la fracción de la imagen, incluida la E/S de datos, debe ser como máximo de $221\mu\text{s}$ para mantener dicho flujo de pixels. Esto no parece difícil para un APAP cuya dinámica está caracterizada por una constante de tiempo por debajo del microsegundo. En estas condiciones, esta aproximación resulta más factible y menos costosa que la implementación multichip. Para ponerla en práctica va a ser necesario definir el *chipset* apropiado [Rosk96] para dar soporte al multiplexado temporal del micro-procesador basado en CNNs, al que vamos a llamar CNNUC (CNN *universal chip*). Este chipset está compuesto por varios circuitos integrados analógicos, digitales y de señal mixta. Su función es, básicamente, actuar de interface entre el CNNUC y las fuentes de señal, por un lado, y el entorno digital de datos y control, por otro. EL CNNUC puede entenderse en este contexto como un microprocesador visual, que opera sobre señales analógicas y lógicas, controlado por un conjunto reducido de instrucciones digitales. La transmisión de datos y de señales de control entre el CNNUC y el sistema digital en el que reside, se realiza mediante tres buses: un bus de datos analógicos, un bus de datos digitales y un bus, digital, de instrucciones (Fig. 17). Conectados al bus de datos digitales encontramos un procesador digital, memoria RAM y una interfaz, con los consiguientes convertidores, con el bus de datos analógicos. COnectados al bus analógico, encontramos las

fuentes de la señal de video y los componentes no-estándar del sistema: el APAP y el chip de memoria analógica (aRAM).

Los puntos críticos del diseño del chipset de CNN son los *cuellos de botella* en la transmisión de datos. Supongamos que estamos recibiendo muestras de la imagen con una frecuencia del reloj de pixel f_S, que, según las definiciones de la sección anterior, se obtiene de:

$$f_S = N_H N_V f_V \tag{28}$$

Si no usamos el chip de memoria analógica (aRAM), y consideramos una arquitectura de *pipeline* para el APAP, podemos definir la frecuencia de adquisición de los pixels, como el cociente entre la anchura del bus de entrada b_{pi} y el tiempo de adquisición de una muestra t_{pi}:

$$f_{pi} = \frac{b_{pi}}{t_{pi}} \tag{29}$$

Si pretendemos que esta frecuencia no reduzca la eficiencia del *pipeline*:

$$f_{pi} \geq f_S \tag{30}$$

ya que de otro modo tendríamos un embotellamiento entre la fuente de señal de video y el procesador. A continuación, los $M_p \times N_p$ pixels son procesados en paralelo. La frecuencia de procesamiento de un pixel será:

$$f_{pp} = \frac{M_p N_p}{t_{pp}} \tag{31}$$

donde t_{pp} es el tiempo que el CNNUC dedica a procesar la sub-imagen. Mientras más grande sea el procesador, más pixels procesa en paralelo, luego mayor será la frecuencia de procesamiento de pixel. Para evitar cuellos de botella dentro del procesador, debe cumplirse que:

$$f_{pp} \geq f_{pi} \tag{32}$$

Finalmente, en cuanto a la salida de datos, podemos definir la frecuencia de salida de pixels como:

$$f_{po} = \frac{b_{po}}{t_{po}} \tag{33}$$

donde b_{po} es la anchura del bus analógico de salida y t_{po} el tiempo que se tarda en transmitir una muestra. Para evitar atropellos, debe darse que:

$$f_{po} \geq f_{pp} \tag{34}$$

El chip de memoria analógica (aRAM) funciona como caché de alta velocidad

del APAP. Permite implementar un *pipeline* en el caso de que la información capturada necesite reorganizarse antes de ser procesada, o incluso en el caso en que el APAP no posea una estructura de *pipeline*.

3. Metodologías de simulación y diseño de chips de CNNs

3. 1. Modelado de comportamiento y simulación de chips de CNN

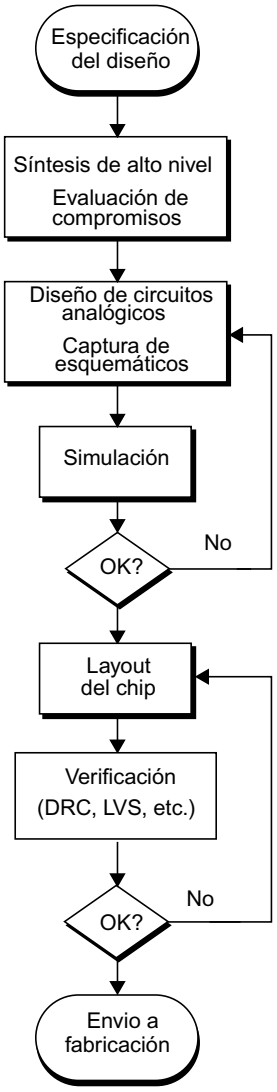


FIGURA 18. Diagrama de flujo genérico del diseño de un chip VLSI analógico.

Las excepcionales prestaciones que ofrecen las CNNs en el procesamiento en paralelo de señales masivas sólo pueden ser aprovechadas al cien por cien por realizaciones VLSI analógicas, y no por implementaciones software o mediante hardware digital acelerado [Fehe96]. Los chips de CNN típicos pueden contener hasta 200 transistores por celda, incluyendo los fotosensores y su circuitería de adaptación [Espe96a] [King95]. Además, las aplicaciones prácticas, orientadas al ámbito industrial, requieren arreglos de un tamaño importante (mínimo 100×100 celdas). De modo que los diseñadores de chips de CNNs tienen que afrontar niveles de complejidad por encima de los 10^6 transistores, la mayoría de ellos operando en modo analógico. La simulación juega un papel muy importante en el diseño de estos sistemas analógicos tan complejos (Fig. 18). debe ser suficientemente rápida para permitir varias iteraciones en el diseño de los bloques analógicos en un tiempo razonable. Debe ser fiable, reflejar con precisión el comportamiento de estos bloques, de modo que el diseñador cuente con una información sólida a partir de la cual tomar decisiones y resolver los compromisos de diseño. El problema está en que la simulación en el diseño de grandes chips analógicos, y de señal mixta, de procesamiento paralelo, está fuertemente condicionada por las limitaciones en la capacidad de cómputo de las herramientas de diseño. Por un lado, los paquetes de software de alto nivel no permiten implementar fácilmente modelos realistas de los circuitos electrónicos [Plon90]. Aunque suponen una herramienta de una simulación rápida, y se utilizan para diseñar nuevas aplicaciones de CNN mediante modelos de alto nivel, su falta de detalle los hace inadecuados para la simulación fiable de circuitos integrados. Por otro lado, los simuladores eléctricos de tipo SPICE, raramente son capaces de manejar circuitos con más de 10^5 transistores. Una simulación de un *netlist* que contenga alrededor de 10^6 transistores puede necesitar varios días de CPU en una Sun SPARCstation 10 [Meta88]. O sea que, las herramientas de simulación de muy bajo nivel son

extremadamente lentas cuando trabajas sobre circuitos con una gran número de dispositivos, haciendo que la simulación completa de un chip de CNN sea inconcebible. Para poder disponer de una herramienta que fuera a la vez fiable y suficientemente rápida, habría que buscar la manera de acercar ambos enfoques hasta una solución intermedia.

Una aproximación consistiría en configurar y programar los simuladores de alto nivel para una simulación más realista del hardware VLSI. Esto supondría un considerable esfuerzo de análisis y programación. Otra solución sería definir macromodelos y bloques de comportamiento establecido en herramientas de tipo SPICE, con el fin de disminuir el consumo de CPU. Esto resultaría en una descripción simplificada, pero aún precisa, del circuito, pero el núcleo del simulador tendría todavía que manejar un elevado número de interconexiones debido a la topología de la red. De modo que la metodología que proponemos trata de superar estas limitaciones con una herramienta específica que concentra los esfuerzos para el modelado detallado del circuito en la unidad básica de proceso, celda, y en su comportamiento, mientras que la topología de la red se encuentra soportada de manera implícita por la forma de las estructuras de datos y por el núcleo del simulador que opera con estas estructuras. Haciendo uso de las propiedades de las CNNs, como la regularidad y la uniformidad, y el alcance local de las conexiones, las red puede ser descrita a partir de ciertas variables que son una agregación de variables locales. La descripción de la CNN consiste en una ecuación diferencial en forma matricial, Las relaciones entre los elementos de las matrices, definidas en el ámbito local, se repiten en cualquier parte de la red. Aparte de estas definiciones de ámbito local, no va a ser necesaria ninguna otra consideración respecto a la topología de la red. Todo esto permite que la descripción de la CNN sea tan detallada como queramos, de modo que un modelo ideal puede ser refinado y mejorado progresivamente para reflejar algún aspecto importante derivado de una implementación específica. Siguiendo esta filosofía se ha desarrollado SIRENA.

Introducción a SIRENA

SIRENA es un entorno para el modelado y la simulación de CNNs orientado a la implementación VLSI [Carm99a]. Está escrito en C usando técnicas de programación orientada a objetos y funciona bajo UNIX y en el entorno gráfico X-Windows. Un lenguaje de programación propio, DECEL, permite describir modelos arbitrariamente complicados de CNN, desde la definición algorítmica hasta la descripción detallada de un circuito en concreto con un análisis profundo de los parásitos y una dinámica de orden superior. Estos modelos son posteriormente enlazados con el núcleo del simulador sin necesidad de recompilar el código de los principales componentes del sistema. La eficiencia de SIRENA reside en la forma en que los modelos de CNN se describen. Esta descripción se centra en las características de la celda básica mientras que la arquitectura global de la red se encuentra embebida en el núcleo del simulador, de manera que queda definida de manera implícita en la generación de los

modelos. En la Fig. 19 podemos ver la arquitectura de SIRENA, consistente en tres módulos principales: la herramienta de generación de modelos (GMS), el núcleo simulador (NSS) y la interfaz gráfica de usuario (GUI). Así que lo primero que habría que hacer es una descripción de la red en DECEL, para ser compilada y enlazada por el GMS. Luego, mediante los ficheros (scripts) de configuración adecuados, podemos lanzar diferentes tipos de simulación de la red mediante el NSS. Las salidas y las entradas de la simulación pueden editarse mediante la GUI.

En la descripción de la red en lenguaje DECEL distinguimos tres partes: la declaración de las variables, la descripción de la evolución de las capas y la descripción de la red. Tomemos como ejemplo el modelo original de CNN de Chua y Yang (ver página 18). Atendiendo a las ecuaciones que establecen la evolución de la variable de estado de la celda y la generación de la salida, confeccionamos el fichero de descripción de la red (**chua.cs** en la Fig. 20). En primer lugar aparecen todas las magnitudes implicadas en el comportamiento de una capa de CNN, declarándose también el tipo de datos al que pertenecen: matrices, templates, escalares, contornos o escalares. Una vez hecho esto, la sección siguiente describe la evolución de la capa, en forma parecida a como se escriben las fórmulas matemáticas en lenguaje C. De hecho sólo algunos operadores son específicos de DECEL, los demás pertenecen a las librerías matemáticas de C. En esta ocasión encontramos operadores de DECEL como el operador derivada (*derivative*), el evaluador de conexión (><) y el evaluador del contorno (#), junto a operadores de C: suma (+), producto (*), menos (–), división (/), condicional (:). Finalmente, la última sección define la estructura de la red a partir de las capas descritas anteriormente. En este caso tenemos una red de una sola capa. Es importante hacer notar que no se ha asignado ningún valor a las variables de la red durante la definición y la compilación del modelo. Estas magnitudes van a ser instanciadas en el momento de la simulación, de modo que un mismo modelo sirve para describir redes con la misma estructura de la celda pero diferentes tamaños, diferentes templates, diferentes constantes de tiempo.

Un modelo compilado de CNN puede simularse a partir de los *scripts* que confi-

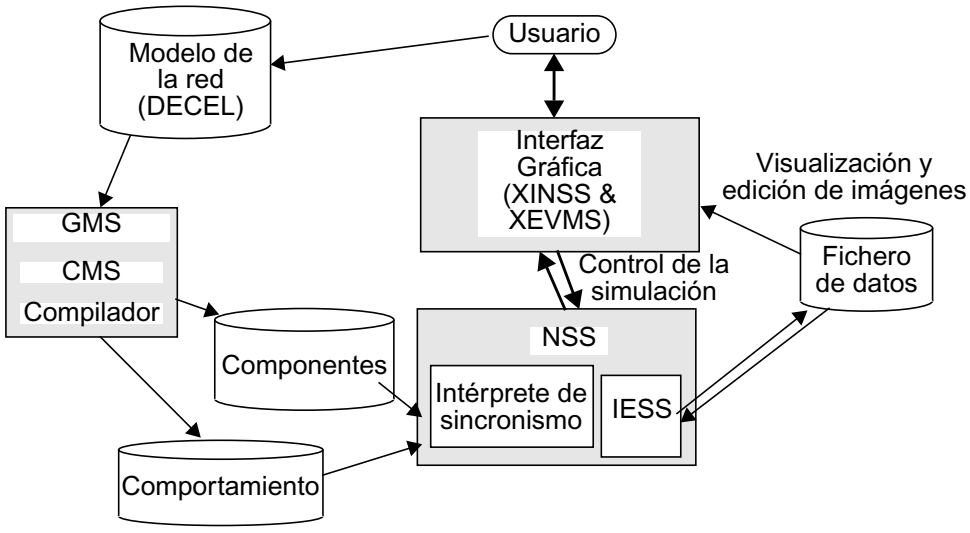


FIGURA 19. Diagrama de bloques conceptual de SIRENA.

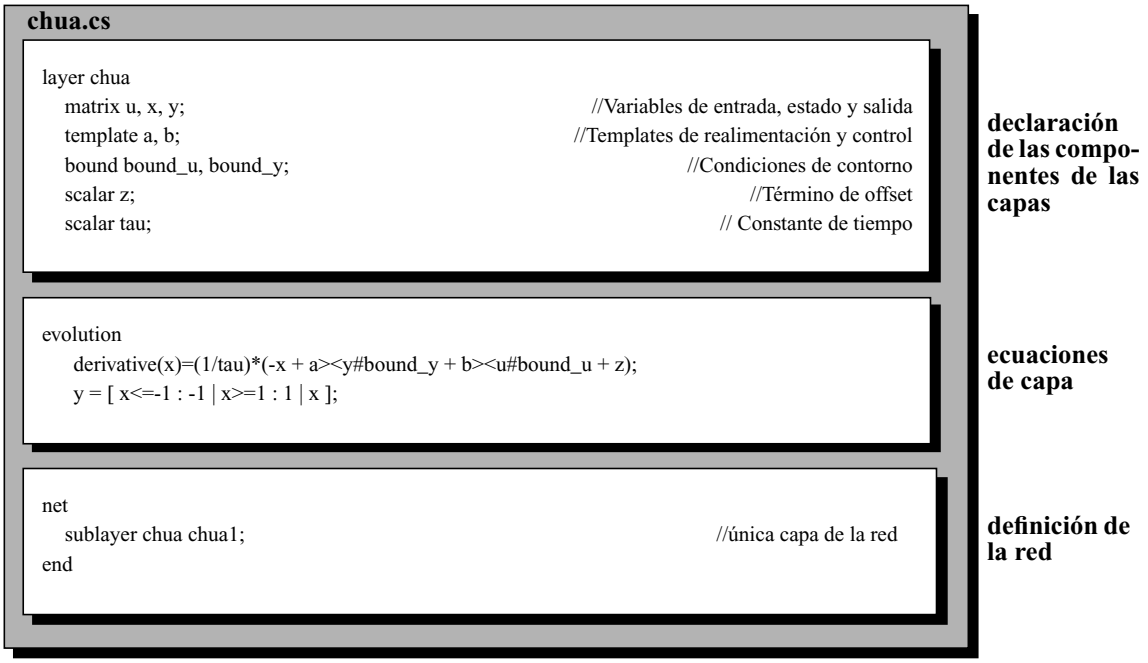


FIGURA 20. Descripción DECEL del modelo Chua-Yang de CNN.

guran el NSS y que detallan los valores de las variables del modelo, o bien mediante la GUI, ya que existe una interfaz gráfica para configurar el núcleo simulador (XINSS en Fig. 21) y otra para editar y visualizar las entradas y las salidas (XEVMS y XVMS, Fig. 21).

Núcleo del simulador

Como hemos visto, la evolución de una CNN con dinámica de tiempo continuo viene descrita por un sistema de ecuaciones diferenciales no lineales, para el que no puede encontrarse una forma cerrada de la solución. Así que tenemos el siguiente sistema de ecuaciones:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}[\mathbf{x}(t)] \qquad \text{ó} \qquad \mathbf{x}(t) = \mathbf{x}(t_0) + \int_{t_0}^{t} \mathbf{f}[\mathbf{x}(\tau)]d\tau \qquad (35)$$

que tendrá que ser resuelto mediante algún método numérico de integración. Estos métodos consisten en el mapeo del sistema a otro, de tiempo discreto, que emule la dinámica de tiempo continuo. La elección del método de integración, y por tanto de la forma del sistema de tiempo discreto, consistirá en determinar la manera en que la integral en la Ec. 35 es evaluada. Existen varios algoritmos de integración: *forward* y *backward* Euler, trapezoidal, gear, etc. Nosotros vamos a considerar sólo tres de ellos para establecer una comparación, siguiendo las indicaciones del Prof. H. Harrer [Harr94]. Así que examinaremos la fórmula explícita de Euler (más conocido como *forward* Euler), el algoritmo de predicción-correción (que realiza una predicción explícita y una correción implícita) y el Runge-Kutta de cuarto orden (método explícito de alto orden).

Con estos algoritmos de integración numérica hemos simulado la evo-

lución de una fila de 8 celdas de CNN que implemente el detector de componentes conectados [Mats90] de un patrón binario de entrada. Hemos hecho una estimación del paso máximo del algoritmo que aún permite la convergencia el sistema de tiempo discreto al mismo estado que el sistema de tiempo continuo, encontrando que el método de Runge-Kutta exhibe convergencia a mayores tamaños de paso. Esta información no sería de gran utilidad si al mismo tiempo no tuviéramos una idea de los recursos (en términos de consumo de CPU) que cada algoritmo requiere. Podemos ver, en la Fig. 22(a), que para una red de este tamaño, 1×8 celdas, no hay mucha diferencia en tiempo de CPU para los diferentes tamaños del paso. Un aspecto en el que el método de Runge-Kutta resulta mejor cualificado es en la precisión de la emulación. Como vemos en la Fig. 22(b), el Forward Euler se desvía claramente del comportamiento *real*[viii]. En estas condiciones, el algoritmo de integración seleccionado para el núcleo simulador de SIRENA es un Runge-Kutta de 4º orden. Para mejorar la eficiencia puede disponerse también de un mecanismo de control de paso, que permite alargar el paso de la integración en los tramos de la evolución que no son críticos, sin perder por ello precisión. Los parámetros que determinan el funcionamiento del algoritmo, incluidos los que fijan el

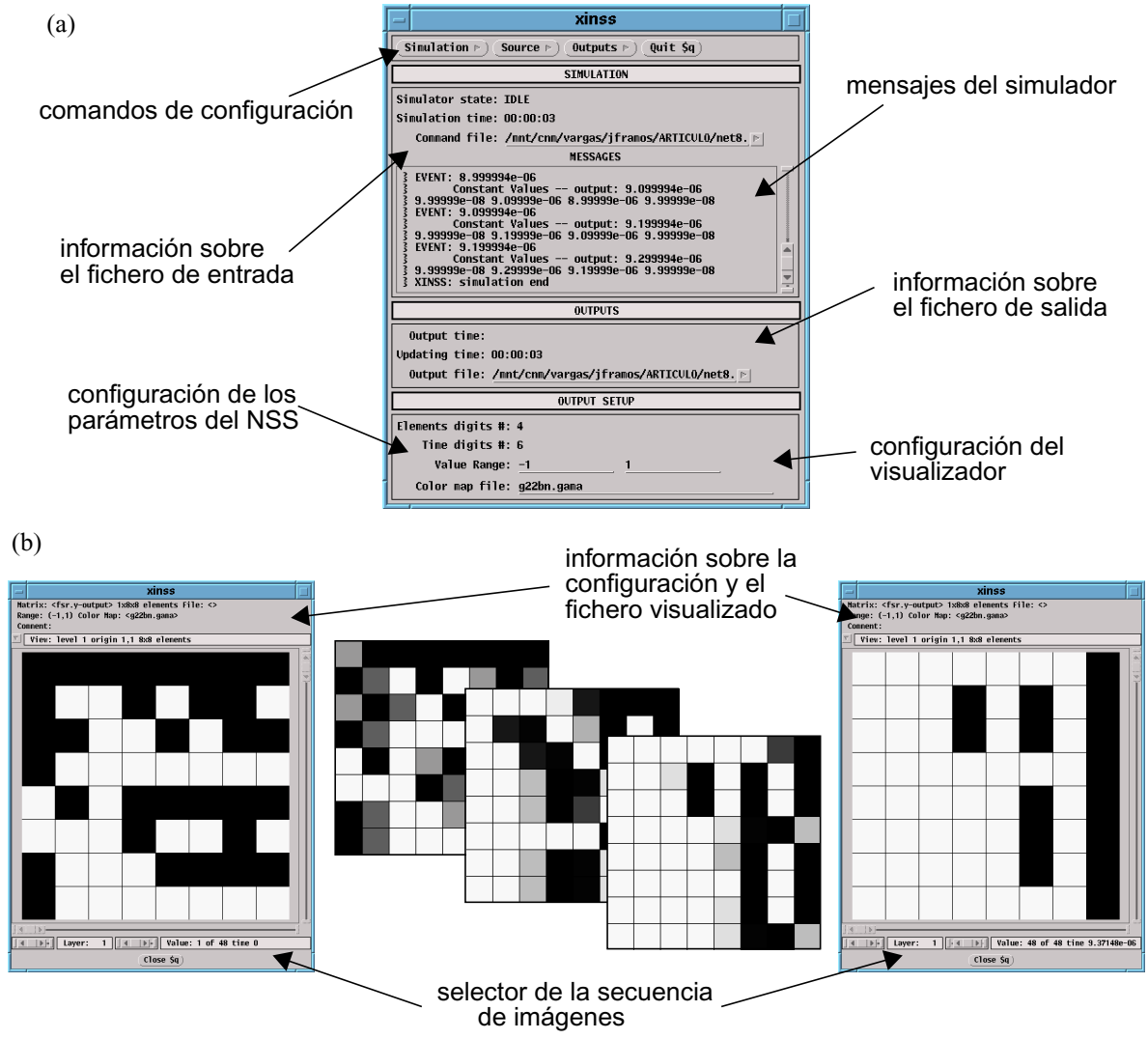


FIGURA 21. (a) XINSS: interfaz gráfica del NSS, (b) XVMS: visualizador de imágenes.

criterio de convergencia, pueden ser manipulados por el usuario a voluntad, desde un *shell* interactivo, desde un *script* o desde la GUI.

Ejemplos de modelado de CNNs

Siguiendo el procedimiento descrito anteriormente, hemos desarrollado algunos modelos de implementaciones reales de CNN. Un primer ejemplo es una realización en modo de corriente reportada en [Rodr93]. Se trata de una CNN de pesos fijos construida con espejos de corriente, que implementa la detección de componentes conectados (véase Apén-

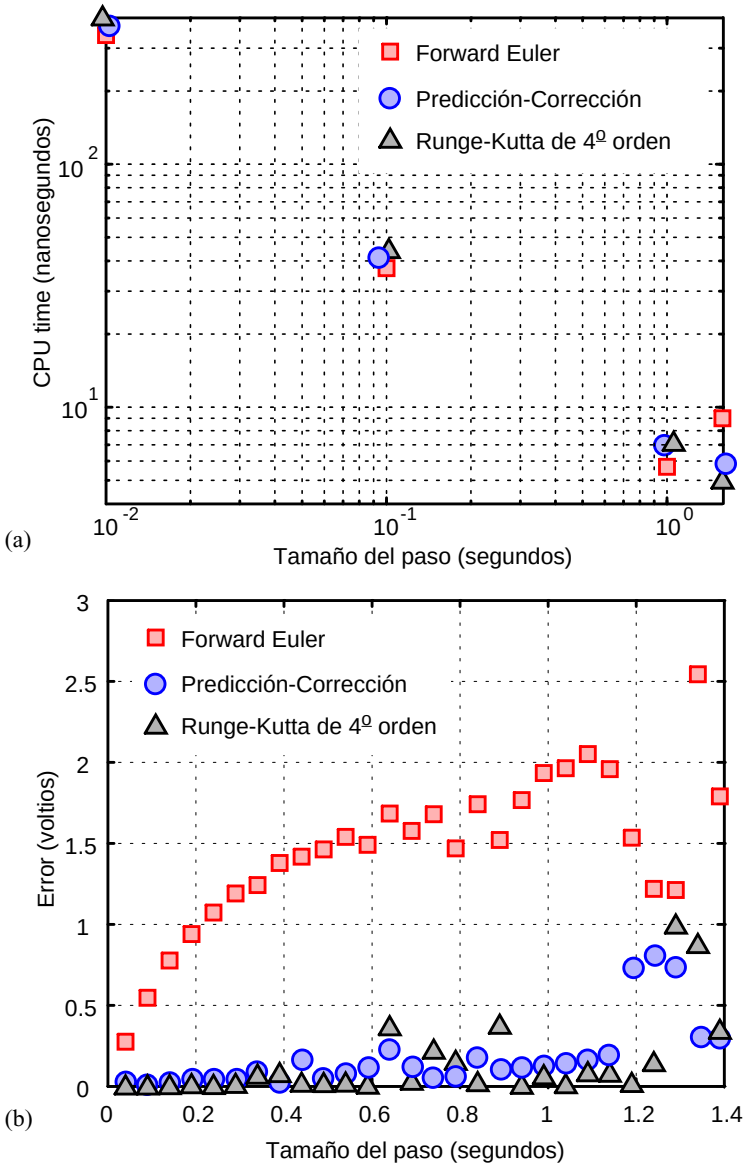


FIGURA 22. (a) Gráfico logarítmico del tiempo de CPU frente al tamaño del paso y (b) error máximo en la evaluación del vector de estados frente al tamaño del paso de integración.

viii. Lo que aquí llamamos comportamiento real sigue siendo una emulación del sistema de tiempo continuo. Corresponde a un método de Runge-Kutta con una paso muy pequeño, 0.001 segundos. Con este tamaño de paso, la precisión de cualquier método sería muy alta, a expensas de un grandísimo consumo de CPU.

dice 3). Si consideramos las tensiones en los condensadores C_x, C_{yp} y C_{yn} como las variables de estado x, y_p e y_n, tendremos tres constantes de tiempo asociadas a estos nudos: $\tau_x = C_x/G_c$, $\tau_{yp} = C_{yp}/G_c$ y $\tau_{yn} = C_{yn}/G_c$. La evolución de estas variables puede describirse con ayuda de algunas corrientes auxiliares:

$$\tau_x \frac{dx}{dt} = \frac{1}{G_c} \cdot [2(i_{cpc} - i_{cnc}) + (i_{cpl} - i_{cnl}) + (i_{cpr} - i_{cnr}) + (i_{cpx} - i_{cnx})]$$

$$\tau_{yp} \frac{dy_p}{dt} = \frac{1}{G_c} \cdot (2i_{cpyp} - i_{cn1yp} - i_{cn2yp}) \tag{36}$$

$$\tau_{yn} \frac{dy_n}{dt} = \frac{1}{G_c} \cdot (2i_{cpyn} - i_{cn1yn} - i_{cn2yn})$$

de modo que la corriente que fluye hacia C_x es la suma de la realimentación, las contribuciones de los vecinos y el termino de pérdidas. Cada una de estas contribuciones tiene una componente positiva, debido a las corrientes de desplazamiento de nivel de las fuentes PMOS, y un término negativo, introducido por los transistores de canal-n, y que vamos a modelar mediante las ecuaciones de nivel-1, de Schichman-Hodges, del transistor MOS. Computando todas las variables auxiliares que aparecen en las ecuaciones de la evolución de las variables de estado, llegamos a la descripción en DECEL expuesta en la Fig. 23. Asignando los valores apropiados a los parámetros del modelo, extraídos de los ficheros tecnológicos y del análisis del circuito, podemos simular esta red para poder comparar los resultados con los de HSPICE.

Otro modelo que hemos desarrollado está basado en la implementación reportada en [Cruz92]. Esta es una realización programable mediante OTAs (amplificadores de transconductancia). La no linealidad de estos bloques (véase Apéndice 3) permite realizar la limitación de la salida mientras que la multiplicación se realiza mediante un bloque basado en la celda de Gilbert [Gilb68]. La evolución de la celda viene dada por:

$$\tau \frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + z + \frac{1}{G_c}\left\{ \sum_{k=-r}^{r} \sum_{l=-r}^{r} [G_{m_A}(a_{kl}, x_{(i+k)(j+l)}) + \right. \tag{37}$$
$$\left. + G_{m_B}(b_{kl}, u_{(i+k)(j+l)})] \right.$$

En este modelo de CNN la multiplicación de los pesos de conexión se realiza en origen, ya que va a ser más fácil llevar las contribuciones de corriente a los vecinos, ya que deben concurrir en un mismo nudo, que transmitir las variables de estado en tensión. Esto precisará de una reorganización de los elementos de los *templates* [Espe94a]. Además, será necesario escalar los elementos de los templates diseñados para el modelo ideal, de acuerdo con los valores de los parámetros que caracterizan los bloques amplificadores y multiplicadores. Todo esto es tenido en cuenta en la descripción del modelo que se expone en la Fig. 24. Una vez compilado, simulamos este modelo con SIRENA para compararlo con la implementación en HSPICE.

cm.cs

```
layer layer_cm
   matrix x, yp, yn ;                                   //Variables de estado
   matrix yl, yr;                                       //Variables de estado de los vecinos
   matrix icx, icyp, icyn;                              //Corrientes auxiliares
                  ⋮                                              ⋮
   matrix icn1yn,icn2yn,icn1yp2, icn1yn2;
   template ayl,ayr;                                    //Operadores de evaluación de los vecinos
   boundary bound_yp,bound_yn;                          //Condiciones de contorono (vyp, vyn)
   scalar vdd, vg;                                      //Alimentación y tensión de referencia
   scalar sn,sp;                                        //Factor de geometría de los transistores MOS
   scalar betap,betan,vtp,vtn;                          //Parámetros tecnológicos
   scalar taux, tauyp, tauyn;                           //Constantes de tiempo
```

declaración de las componentes de la capa

```
evolution
        yl= ayl><yp#bound_yp;
        yr= ayr><yn#bound_yn;

        icpl= [ x < vg+vtp : (betap*sp/2)*(vdd-vg-vtp)*(vdd-vg-vtp) |
                betap*sp*((vdd-vg-vtp)*(vdd-x)-(vdd-x)*(vdd-x)/2) ];
        icnl2= [ x > (yl-vtn) : (betan/2)*sn*(yl-vtn)*(yl-vtn) |
                betan*sn*((yl-vtn)*x -x*x/2) ];
        icnl= [ yl > vtn : icnl2 | 0 ];
        icpr=icpl ;
        icnr2= [ x > (yr-vtn) : (betan/2)*sn*(yr-vtn)*(yr-vtn) |
                betan*sn*((yr-vtn)*x -x*x/2) ];
        icnr= [ yr > vtn : icnr2 | 0 ];
        icpc=2*icpl ;
        icnc2= [ x > (yp-vtn) : betan*sn*(yp-vtn)*(yp-vtn) |
                2*betan*sn*((yp-vtn)*x -x*x/2) ];
        icnc= [ yp > vtn : icnc2 | 0 ];
        icpx= icpl;
        icnx= [ x < vtn : 0 | (betan/2)*sn*(x-vtn)*(x-vtn) ];
                icpyp= [ yp < vg+vtp : (betap*sp/2)*(vdd-vg-vtp)*(vdd-vg-vtp) |
                betap*sp*((vdd-vg-vtp)*(vdd-yp)-(vdd-yp)*(vdd-yp)/2) ];
        icn1yp2= [ vyp > (x-vtn) : (betan/2)*sn*(x-vtn)*(x-vtn) |
                betan*sn*((x-vtn)*yp -yp*yp/2) ];
        icn1yp= [ x > vtn : icn1yp2 | 0 ];
        icn2yp= [ yp < vtn : 0 | (betan/2)*sn*(yp-vtn)*(yp-vtn) ];
        icyp= 2*icpyp-icn1yp-icn2yp;
        icpyn= [ yn < vg+vtp : (betap*sp/2)*(vdd-vg-vtp)*(vdd-vg-vtp) |
                betap*sp*((vdd-vg-vtp)*(vdd-yn)-(vdd-yn)*(vdd-yn)/2) ];
        icn1yn2= [ yn > (vyp-vtn) : (betan/2)*sn*(yp-vtn)*(yp-vtn) |
                betan*sn*((yp-vtn)*yn -yn*yn/2) ];
        icn1yn= [ yp > vtn : icn1yn2 | 0 ];
        icn2yn= [ yn < vtn : 0 | (betan/2)*sn*(yn-vtn)*(yn-vtn) ];
        icyn= 2*icpyn-icn1yn-icn2yn;

        derivative(x)= icx/(taux*gc);
```

ecuaciones de la capa

```
network
   sublayer layer_cm cm;
end
```

definición de la red

FIGURA 23. Decripción en DECEL de la CNN en modo de corriente.

ota.cs

```
layer layer_otasc
   matrix x,u;                                          //Variables de estado y entrada
   matrix i1x,i2x,i1u,i2u;                              //Corrientes del los pares diferenciales
   matrix i3a00,i4a00,i5a00,i6a00;                      //Corrientes de los multiplicadores
                    ⋮                                          ⋮
   matrix i3b22,i4b22,i5b22,i6b22;                      //Corrientes de los multiplicadores
   matrix iin, iout;                                    //Corrientes de salida y de entrada
   matrix io00x,io01x,io02x;                                  //Corrientes de salida
                    ⋮                                          ⋮
   matrix io20u,io21u,io22u;                                  //Corrientes de salida
   matrix ibias;                                        //Corriente de polarización
   matrix b3a00,b4a00,...,b4b22;                        //Parámetros de los multiplicadores
   matrix ibi1,ibi2;                                    // Corrientes del término de offset
   template p00,p01,...,p22;                                  //Evaluación del vecindario
   boundary ic00x,ic01x,ic02x;                                //Corrientes de los contornos
                    ⋮                                          ⋮
   boundary ic20u,ic21u,ic22u;                                //Corrientes de los contornos
   scalar a00,a01,a02;                                  // Tensiones de los templates
                    ⋮                                          ⋮
   scalar b20,b21,b22;                                  // Tensiones de los templates
   scalar vbias;                                        //Tensión del termino de offset
```

declaración de las componentes de la capa

```
evolution
//Corrientes del par diferencial de entrada
i1x = [ x >= sqrt(ibx/b1x) : ibx |x <= -sqrt(ibx/b1x) : 0 |
            ibx/2 + (1/2)*b1x*x*sqrt(2*ibx/b1x-x*x)];
                    ⋮                                          ⋮
i2u = [ u >= sqrt(ibu/b1u) : 0 |u <= -sqrt(ibu/b1u) : ibu |
            ibu/2 - (1/2)*b1u*u*sqrt(2*ibu/b1u-u*u)];
// Multiplicadores
i3a00 = [ a00 >= sqrt(i2x/b3a00) : i2x |a00 <= -sqrt(i2x/b3a00) : 0 |
            i2x/2 + (1/2)*b3a00*a00*sqrt(2*i2x/b3a00-a00*a00)];
                    ⋮                                          ⋮

i4b22 = [ b22 >= sqrt(i1u/b4b22) : 0 |b22 <= -sqrt(i1u/b4b22) : i1u |
            i1u/2 - (1/2)*b4b22*b22*sqrt(2*i1u/b4b22-b22*b22)];
// Corrinetes de salida
   iout=i1x-i2x;
   io00x=i6a00+i5a00-i3a00-i4a00;
                    ⋮
   io22u=i6b22+i5b22-i3b22-i4b22;
   ibias=ibi1-ibi2;
// Evolución de las variables de estado
    iin = p00><io00x#ic00x + p00><io00u#ic00u +
                    ⋮                    ⋮
            p22><io22x#ic22x + p22><io22u#ic22u + ibias;
```

ecuaciones de la capa

```
network
   sublayer layer_otasc otasc;
end
```

definición de la red

FIGURA 24. Descripción en DECEL de la CNN basada en OTAs

Comparación con HSPICE

Para comparar la operación de SIRENA con la de HSPICE, vamos a hacer simulaciones de los modelos referidos con ambas herramientas. En la Fig. 25 podemos ver las formas de onda obtenidas en la simulación con SIRENA y con HSPICE (v. 95.1) de una red de 4×4 celdas que implementa un template de detección de componentes conectados con diferentes modelos de CNN. En primer lugar, Fig. 25(a), tenemos los resultados para un modelo de Chua-Yang. Al ser un modelo ideal, ambos programas ofrecen resultados idénticos. Para el caso de la CNN en modo de corriente, Fig. 25(b), podemos ver alguna discrepancia en los valores alcanzados por la variable de estado durante su evolución, pero lo más importante es que cualitativamente no hay diferencias de comportamiento. Para el modelo del la CNN basada en OTAs encontramos igualmente un comportamiento muy semejante entre los resultados de SIRENA y HSPICE. De modo que de la inspección de estas formas de onda deducimos que la simulación de SIRENA, aún estando basada en modelos menos detallados, ofrece unos resultados fiables.

Es de interés comparar la utilización de recursos que hace cada una de las dos herramientas para obtener unos resultados muy parecidos, como hemos visto. Para ello nos valdremos de ciertos comandos de UNIX que permiten monitorizar el uso de memoria y el tiempo de CPU consumido por cada proceso. Hemos computado el tiempo de CPU empleado en simular, con ambas herramientas, diferentes redes, con tamaños que van desde las 2×2 a las 32×32 celdas, usando primero el modelo de Chua-Yang y luego el de la CNN en modo de corriente, en un Sun Microsystems SPARC Server 1000E con 4 CPUs y 512MB de RAM. Los resultados aparecen reflejados en la Fig. 26. Como vemos, la diferencia de tiempo de CPU consumido en la simulación de modelos complejos de CNN es de dos órdenes de magnitud, y esta distancia aumenta con el número de celdas.

3. 2. Robustez del algoritmo y centrado del diseño

Debido a las inevitables desviaciones de los parámetros del proceso de fabricación, las propiedades de los dispositivos que implementan la red se desvían igualmente de los valores asignados nominalmente durante la fase de diseño. En estas condiciones, interesa encontrar un conjunto de templates que ofrezca cierto margen para las desviaciones de la implementación física de la CNN. Mientras mayor sea este margen, más seguros estaremos de que la realización VLSI va a reflejar el comportamiento idealmente concebido a nivel algorítmico, ya que siendo el margen mayor, las desviaciones de los parámetros del proceso, y por tanto las desviaciones de los bloques analógicos de su comportamiento nominal, pueden ser mayores y todavía permitir un funcionamiento correcto de la red. Pretendemos entonces, partiendo de un punto en el espacio de los parámetros de la CNN —aquellos que determinan el comportamiento de la red a nivel algorítmico— llegar al centro de la zona, hiperplano, más distante de las

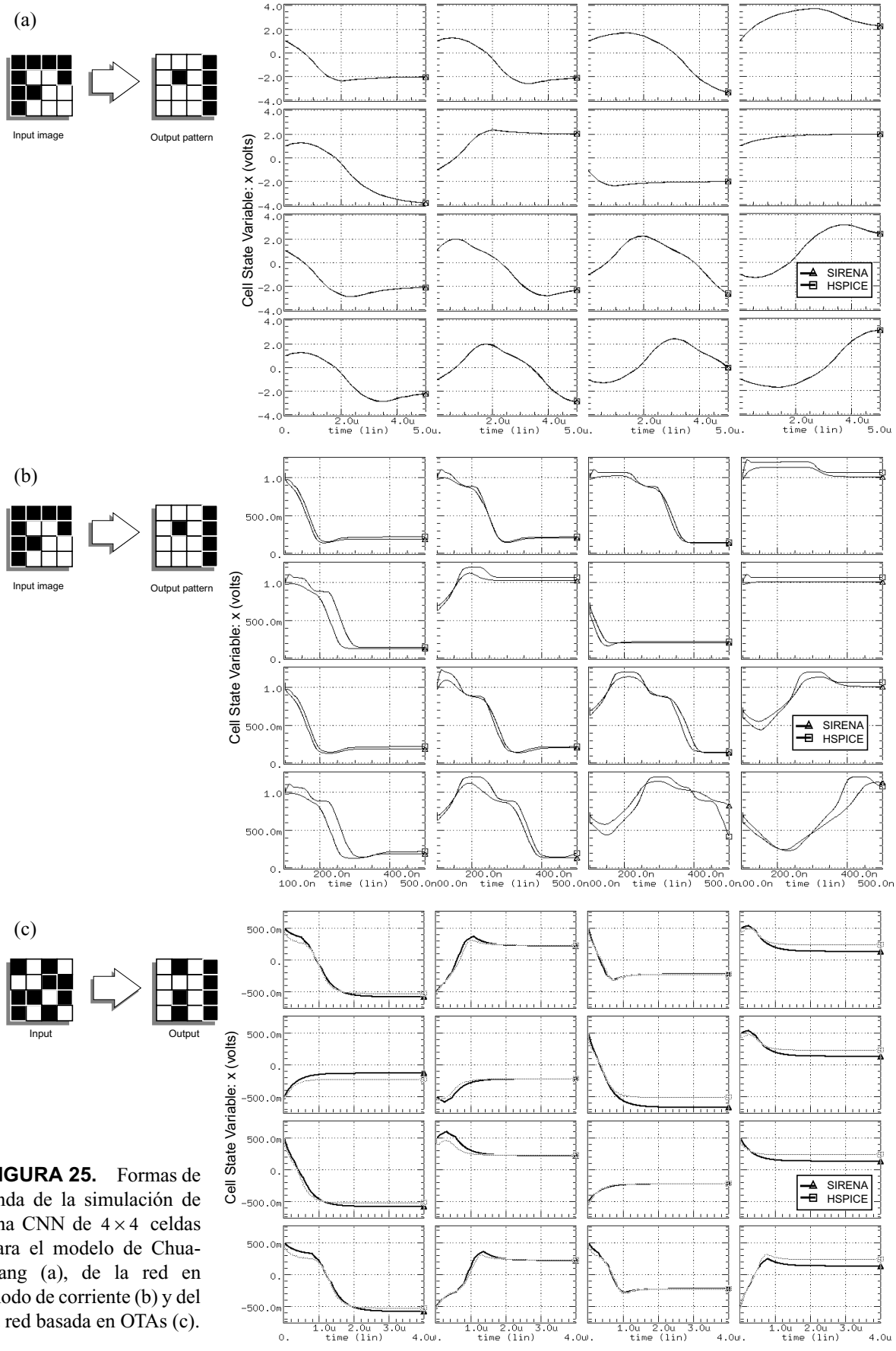


FIGURA 25. Formas de onda de la simulación de una CNN de 4×4 celdas para el modelo de Chua-Yang (a), de la red en modo de corriente (b) y del la red basada en OTAs (c).

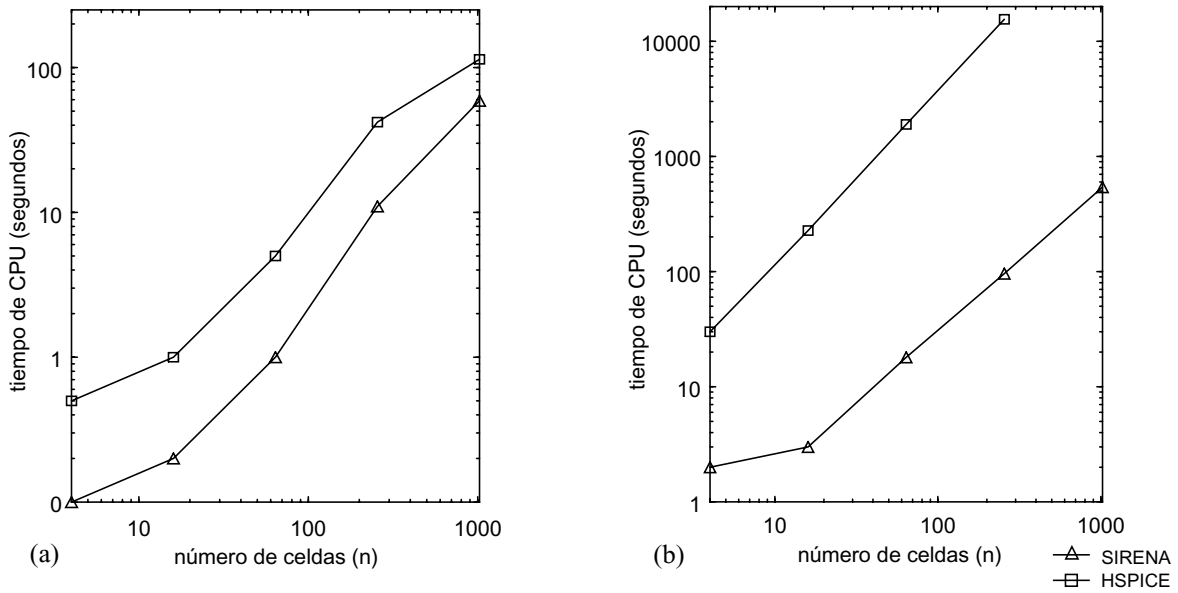


FIGURA 26. Representación en escala logarítmica del tiempo de CPU consumido por la simulación frente al numero de celdas para (a) el modelo de Chua-Yang y (b) CNN en modo de corriente.

fronteras de funcionamiento correcto. En dicho punto, el algoritmo se mostrará lo más robusto posible frente a imperfecciones en la implementación. Este proceso se conoce como centrado del diseño. Puede hacerse de forma analítica, en algunos casos, mediante el estudio detallado de las rutas dinámicas, y puede realizarse automáticamente con herramientas de cálculo numérico de las que disponemos.

Rutas dinámicas del modelo de Chua-Yang

Para estudiar las rutas dinámicas de una celda de la CNN hemos de suponer que el resto de la red ya se ha establecido a un estado estacionario [Chua88]. Garantizar que todas las celdas de la red, por separado, si las demás están quietas, van a converger hacia un estado estacionario, no va a ser condición suficiente para la convergencia de la red, pero sí que es condición necesaria. Así que, en estas condiciones, una celda de una CNN de Chua-Yang evolucionaría siguiendo esta ecuación:

$$\tau \frac{dx_{ij}(t)}{dt} = -Gx_{ij}(t) + a_{00} f(x_{ij}) + k_{ij} \tag{38}$$

donde las contribuciones de los vecinos, que son constantes por encontrarse en estado estacionario, y el término de bias se agrupan en k_{ij}:

$$k_{ij} = \sum_{\substack{k=-r \\ k\neq 0}}^{r} \sum_{\substack{l=-r \\ l\neq 0}}^{r} a_{kl} \cdot y_{(i+k)(j+l)} + \sum_{k=-r}^{r} \sum_{l=-r}^{r} b_{kl} \cdot u_{(i+k)(j+l)} + z_{ij} \tag{39}$$

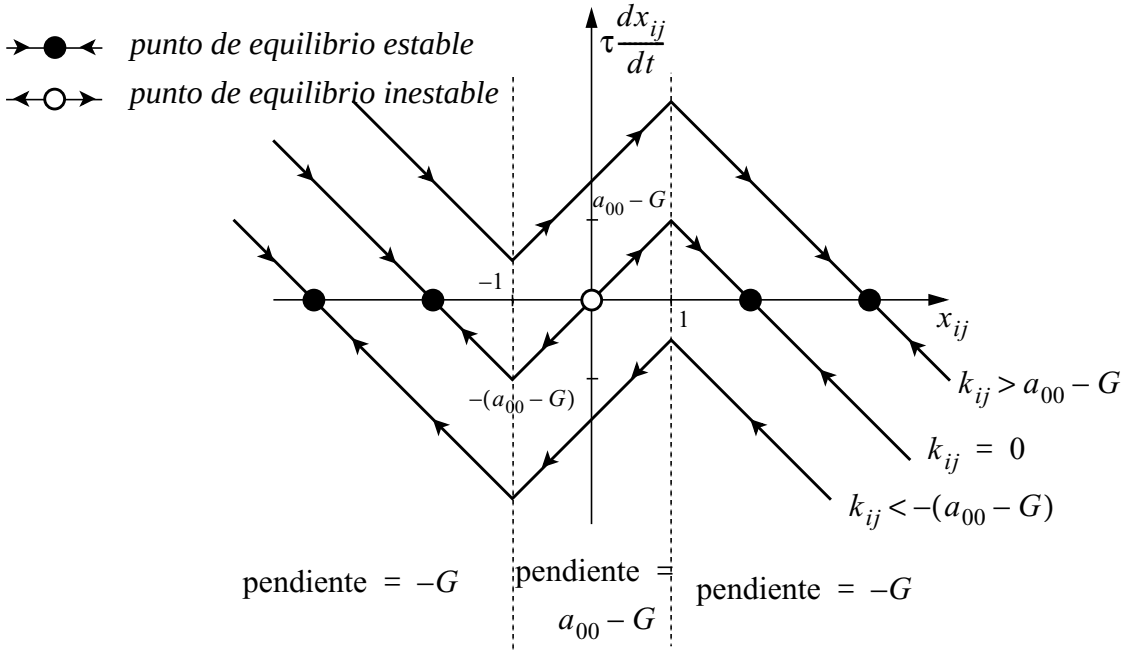


FIGURA 27. Rutas dinámicas en el modelo de Chua-Yang de CNN.

y hemos permitido que la pendiente del término de pérdidas difiera de la unidad, lo que correspondería al caso ideal. Esta celda evoluciona siguiendo las rutas dinámicas expuestas en la Fig. 27, donde hemos supuesto que $|a_{00}| > G$. Dependiendo del valor de k_{ij}, tendremos entre uno y tres puntos de equilibrio. Siempre tendremos, al menos, un punto de equilibrio estable.

Y puesto que el estado estacionario dependerá de la ruta dinámica seguida por la celda, y dado que este estado estacionario final puede considerarse como la salida del procesamiento realizado por la CNN, cualquier desviación en la realización de estas rutas dinámicas podría tener consecuencias nefastas para la funcionalidad del sistema. Estas desviaciones pueden ser, según [King96] y atendiendo a la Ec. 38, de tres tipos: desviaciones en la realimentación de los vecinos, el template de control o el término de bias, o sea, errores en k_{ij}, desviaciones en la implementación del término de perdidas, G, o desviaciones en la implementación de la realimentación propia, a_{00}.

En el primero de los casos, cualquier desviación de k_{ij} provoca un desplazamiento de las rutas dinámicas a lo largo del eje vertical. Esto puede provocar la aparición o desaparición de puntos de equilibrio estables sobre los que se fundamente el procesamiento de la señal, lo cual desemboca en una pérdida de la funcionalidad. En el caso de que los errores se manifiesten en forma de una realización incorrecta de G ó a_{00}, los puntos de equilibrio se desplazan en el eje horizontal pero no se crean ni se destruyen. De modo que para determinar si va a ocurrir una pérdida de funcionalidad, tendremos que evaluar la máxima desviación, con respecto de los valores nominales, que una ruta dinámica va a ser capaz de soportar antes de transformarse en una ruta de distinta naturaleza. Así que para una ruta dinámica, definimos un margen, $\Delta_{\text{máx}}$, más allá del cual cambia la cantidad y calidad de sus puntos de equilibrio.Este margen depende del conjunto de templates. Para una realización concreta de la red, la ruta dinámica real se distancia de la ideal

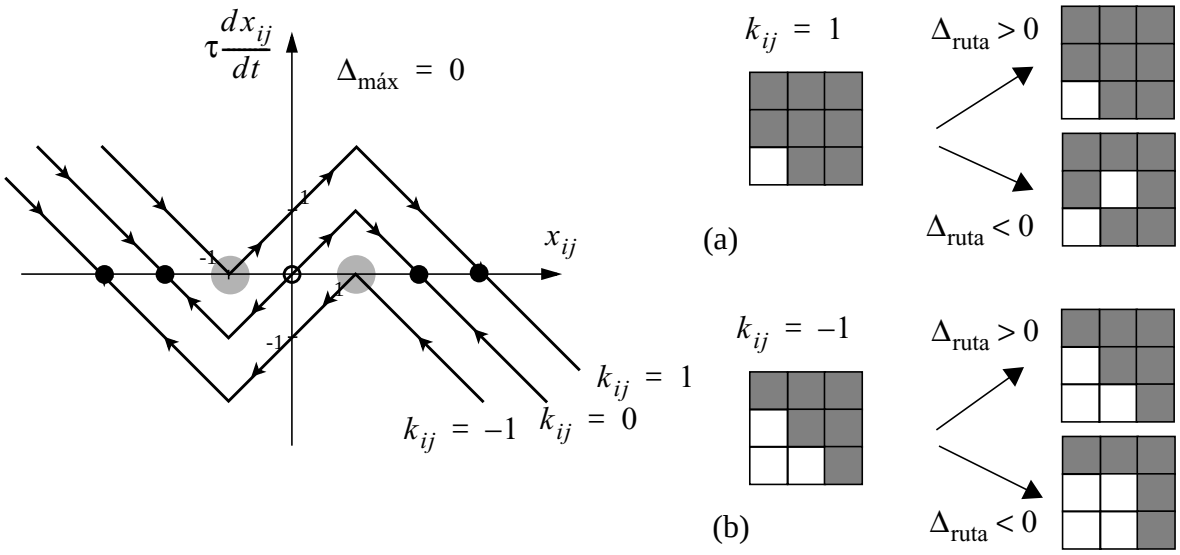


FIGURA 28. Rutas dinámicas para el detector de bordes en el modelo de Chua-Yang de CNN.

una cantidad Δ_{ruta}, que puede obtenerse a partir de los errores en la implementación de los parámetros:

$$\Delta_{ruta} = \Delta G + \Delta z + \sum_{k=-r}^{r} \sum_{l=-r}^{r} (a_{kl} + b_{kl}) \tag{40}$$

supuestos todos positivos. Debe cumplirse que:

$$\Delta_{ruta} < \Delta_{máx} \tag{41}$$

para preservar la funcionalidad de la CNN. Así que mientras mayor sea la $\Delta_{máx}$, mayor será la robustez del algoritmo y mayores pueden ser las tolerancias en la implementación de los templates, que aún la diferencia $\Delta_{máx} - \Delta_{ruta}$ seguirá siendo positiva. Como ejemplo de este análisis, veamos lo que ocurre con el template de detección de bordes en una imagen binaria reportado en [Rosk95]:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \qquad z = -1.5 \tag{42}$$

Lo primero que debemos hacer es identificar la ruta dinámica crítica, es decir, aquella con el menor $\Delta_{máx}$ teórico. La suma de las contribuciones de los vecinos más el término de offset, k_{ij}, podrá tomar diferentes valores según el estado de la propia celda y de los vecinos (véase Apéndice 3), pero fijémonos en los valores críticos. Por ejemplo, si la celda en cuestión se encuentra en estado alto, ■, y tiene tres vecinos, cualesquiera, en el mismo estado, y los otros cinco en estado bajo, □, entonces $k_{ij} = 0$, lo que corresponde a la ruta central de la Fig. 28. Si $|k_{ij}| = 1$, nos encontramos con una ruta dinámica con tolerancia cero. Esta circunstancia se

da en situaciones en las que la definición de borde del patrón binario no es muy clara, e incluso a nosotros nos costaría trabajo decidir si la celda central debería ser considerada borde o no. De todos modos el caso es que una vez definido un criterio, habría que mover las rutas dinámicas, por ejemplo ajustando b_{00} a 2.25, con el fin de obtener un conjunto de templates realizable. Ahora $\Delta_{máx} = 0.25$, que es infinitamente mejor que cero. En resumen, una inspección detallada de las rutas dinámicas permite determinar aquellos aspectos críticos que ponen en peligro la validez de una implementación de la CNN. Con esta información podemos redefinir los templates con el fin de abrir el espacio de diseño, mejorando la robustez del algoritmo frente a imperfecciones de la realización física. No obstante, en el caso de aplicaciones sobre imágenes en escala de grises, el conjunto de pares entrada-salida de entrenamiento puede ser colosal, de modo que no resulta práctico este estudio detallado de la rutas dinámicas, sino que parece conveniente buscar la manera de automatizar el centrado del diseño.

Centrado automático del diseño con SIRENA

Una de las tareas más importantes en la verificación de diseños analógicos VLSI de alta complejidad es el análisis estadístico de la implementación. Por causa de las desviaciones de los parámetros del proceso de fabricación CMOS —el espesor del óxido de puerta, la densidad de átomos dopantes, etc.—, las propiedades eléctricas de dispositivos —la tensión umbral extrapolada, el factor de transconductancia, la constante de efecto substrato— también se distancian de los valores nominales empleados en la fase de diseño, siguiendo una distribución estadística. Así, un parámetro P, cuyo valor era originalmente P_0, se mostrará como:

$$P = P_0 + \Delta P = P_0\left(1 + \frac{\Delta P}{P_0}\right) \tag{43}$$

siendo ΔP, el error absoluto cometido en la implementación, diferente para cada dispositivo. Para evaluar la influencia de estas desviaciones en el comportamiento global de la red, o sea, en la funcionalidad de la CNN, habría que realizar un análisis de la sensibilidad a los parámetros o bien un análisis de Monte Carlo. Hacer esto con las herramientas tradicionales de simulación de circuitos, del tipo SPICE, puede resultar inabarcable, en función de la cantidad de simulaciones que van a ser necesarias y del tamaño de la red y la complejidad del circuito. Sin embargo, sí podemos plantearnos el análisis estadístico de la implementación mediante una herramienta dedicada que facilite la simulación: SIRENA.

Así, para incrementar la robustez del algoritmo de detección de bordes presentado anteriormente, o sea, para aumentar el grado de inmunidad de dicho algoritmo frente a imperfecciones en su realización, podemos hacer un análisis de sensibilidad aprovechando las capacidades multianálisis de SIRENA. Para ello hemos evaluado la respuesta de redes de diferentes tamaños para este conjunto de templates, permitiendo que z adoptara

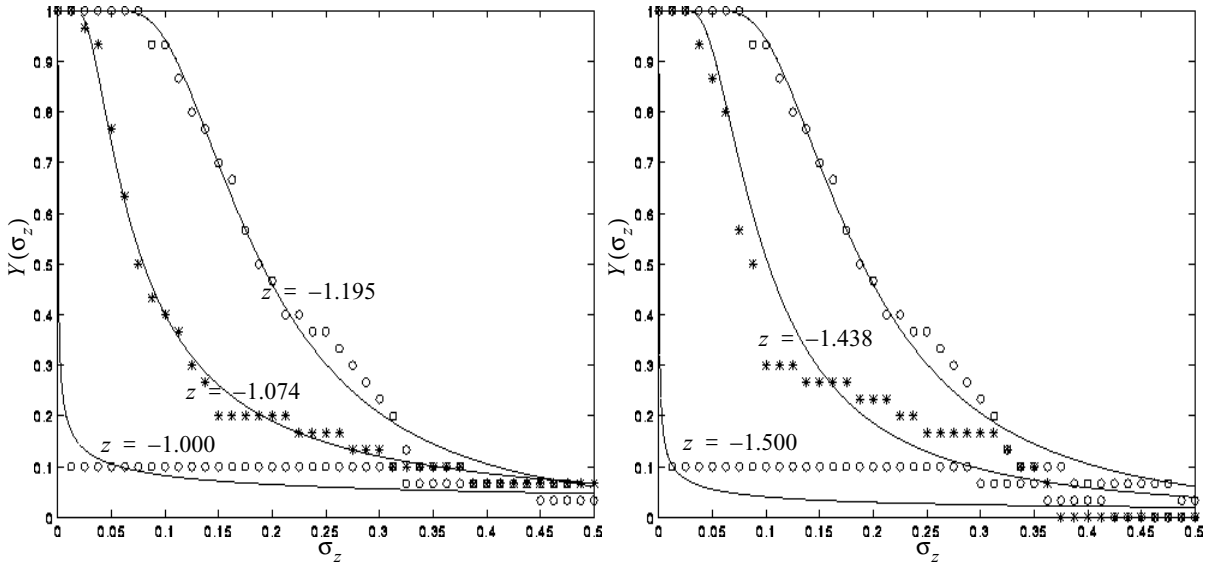


FIGURA 29. Yield paramétrico para diferentes valores del término de offset.

valores dentro de un intervalo centrado en $z = -1.5$ con un radio del 60% del valor nominal. Encontramos que la tolerancia, el rango de validez de la operación, a las variaciones de este parámetro está por debajo del 0.01%. Estos resultados están refrendados por el análisis de Monte Carlo alrededor de $z = -1.5$. Resulta evidente que este conjunto de templates se encuentra en el límite de la región de tolerancia.

Para centrar el diseño empleamos un algoritmo de optimización guiada, ya que suponemos que aunque no estamos en un extremo de la superficie de error sí estamos en la cuenca de atracción del mínimo. La función de error, o de coste, será la inversa del *yield* paramétrico. Para realizar la optimización, empezamos computando el yield en $z = -1.5$ para un conjunto de valores de σ_z (la desviación estándar de z) predeterminado (de 0 a 50% cada 1%), mediante un análisis de Monte Carlo en cada uno de esos puntos ($51 \times 30 = 1580$ simulaciones). Acto seguido, el algoritmo de optimización realizará las evaluaciones de la función de error pertinentes (a razón de 1580 simulaciones por evaluación) y avanzará hacia el valor del parámetro z que hace mínima esta función. La Fig. 29 muestra la evaluación realizada del *yield* paramétrico en función de σ_z, para algunos valores de z por encima y por debajo del valor que posee una mejor curva de *yield*, y que ha sido detectado por el algoritmo de optimización. En concreto $z = -1.195$.

Esta técnica que permite el centrado automático del diseño, necesita de una herramienta eficiente, suficientemente precisa pero con una demanda computacional moderada, para ser incluida en el flujo de diseño. Así, parámetros importantes desde el punto de vista de las especificaciones de alto nivel del problema, como el *yield* paramétrico, pueden utilizarse en el estudio de la viabilidad de la implementación del algoritmo.

Aprendizaje supervisado de *templates*

Como hemos mencionado antes, hay circunstancias en las que los métodos analíticos directos para el diseño de templates no resultan prácticos. Encontramos casos en los que la vasta cantidad de situaciones a considerar hace que el análisis de las rutas dinámicas sea difícilmente manejable. En otras ocasiones, la funcionalidad perseguida no está perfectamente definida, y nos encontramos con un conjunto reducido de pares de entrada y salida, a partir de los cuales resulta difícil extraer características comunes que los separe en clases bien definidas. La solución es el aprendizaje automatizado, que también contará con la disponibilidad de una herramienta eficiente de simulación. Al fin y al cabo, el aprendizaje va a consistir en la optimización de un cierto parámetro de operación: una función objetivo que mida la proximidad entre la salida obtenida y la deseada.

Varios autores han considerado la posibilidad e incorporar el aprendizaje no supervisado y la auto-organización a las CNNs, para poder implementar, por ejemplo, memorias asociativas [Liu93] [Szol93]. No obstante, las tareas de aprendizaje relacionadas con la adaptación de la CNN a la realización de determinado procesamiento de la imagen, están basadas en un modelo de aprendizaje supervisado. En este modelo, hay un profesor que conducirá a la red, adaptando ciertos parámetros, a la realización del mapeo entre entrada y salida deseado. La actualización del vector de parámetros de la red, $\mathbf{p} = \{a_{ul}, \ldots, a_{br}, b_{ul}, \ldots, b_{br}, z\}$, que consiste en la colección de templates que codifica una determinada dinámica, se realiza siguiendo un algoritmo de optimización que intentará encontrar el mínimo de una superficie de error $\varepsilon(\mathbf{p})$.

Un esquema de aprendizaje, basado en el método de la máxima pendiente, para una CNN de una capa aparece ilustrado en la Fig. 30. El objetivo de la optimización es encontrar un vector de parámetros $\mathbf{p}$ para el que la diferencia entre la salida de la red $\mathbf{y}^l(\infty)$ y la salida deseada $\mathbf{d}^l(\infty)$ para una entrada y un estado inicial concretos, $\mathbf{x}^l(0)$ y $\mathbf{u}^l$, y para todos los patrones del *training set* ($l = 1, \ldots, L$), sea mínima. El error en el mapeo E/S se define como una distancia métrica de noma ν, lo cual, para una celda de la red en particular viene a ser:

$$e_{ij}^l(\mathbf{p}) \equiv \left| d_{ij}^l(\infty) - y_{ij}^l(\infty) \right|^\nu \tag{44}$$

de modo que el error de toda la red, podemos definirlo como la suma de los errores particulares de todas las celdas, para todo el conjunto de patrones de entrenamiento:

$$\varepsilon(\mathbf{p}) = \sum_{l=1}^{L} \varepsilon^l(\mathbf{p}) = \sum_{l=1}^{L} \sum_{i=1}^{M} \sum_{j=1}^{N} e_{ij}^l(\mathbf{p}) \tag{45}$$

De modo que iniciamos el proceso con un vector $\mathbf{p}(0)$. Entonces la CNN genera $\{\mathbf{y}^l(\infty) | l = 1, \ldots, L\}$ mientras que el *profesor* produce $\{\mathbf{d}^l(\infty) | l = 1, \ldots, L\}$. Con esto se calculan $\varepsilon(\mathbf{p})$ y, a continuación el

gradiente $d\varepsilon/d\mathbf{p}$ —esto requerirá varias evaluaciones de la superficie de error en las proximidades de $\mathbf{p}$, aunque no lo hemos indicado en la Fig. 30 por facilitar su lectura. Así que una vez obtenidos $\varepsilon(\mathbf{p})$ y $d\varepsilon/d\mathbf{p}$, actualizamos el vector de parámetros:

$$\mathbf{p}(n+1) = \mathbf{p}(n) - \eta \frac{d\varepsilon(\mathbf{p})}{d\mathbf{p}} \tag{46}$$

donde η es la *tasa de aprendizaje*. Después de un cierto número de iteraciones, el algoritmo converge a un mínimo local de la superficie de error, en este punto, el gradiente se anula $d\varepsilon/d\mathbf{p} = 0$.

Existen diferentes métodos para la evaluación de este gradiente: retropropagación [Pine87] o retropropagación en el tiempo [Pear89], etc. Pero no debe olvidarse que $\varepsilon(\mathbf{p})$ es una función no lineal que además presenta singularidades en sus derivadas, por lo que el Jacobiano, y por tanto el gradiente de la misma, puede no estar bien definido en todos los puntos. Parece que una aproximación más conveniente, en lugar de la computación explícita del gradiente de la superficie de error, sería la observación directa de la influencia de la variaciones de los parámetros. Esto es, generar la actualización del vector $\mathbf{p}$ mediante métodos que no impliquen el cálculo del gradiente en dicho punto [Demb90]. Por ejemplo, utilizando un método de aproximación perturbativo [Cauw96]. Estos mecanismos de aprendizaje han sido puestos en práctica para realizar un entrenamiento de la red post-fabricación. Con un chip de CNN con capacidad de adquisición óptica de datos [Domí97], se presenta en [Szir94] una aplicación de clasificación de texturas. Se trata de obtener un conjunto adecuado de templates, un vector de parámetros, que aplicado a diferentes texturas de entrada, genere patrones de salida muy distintos. Así que

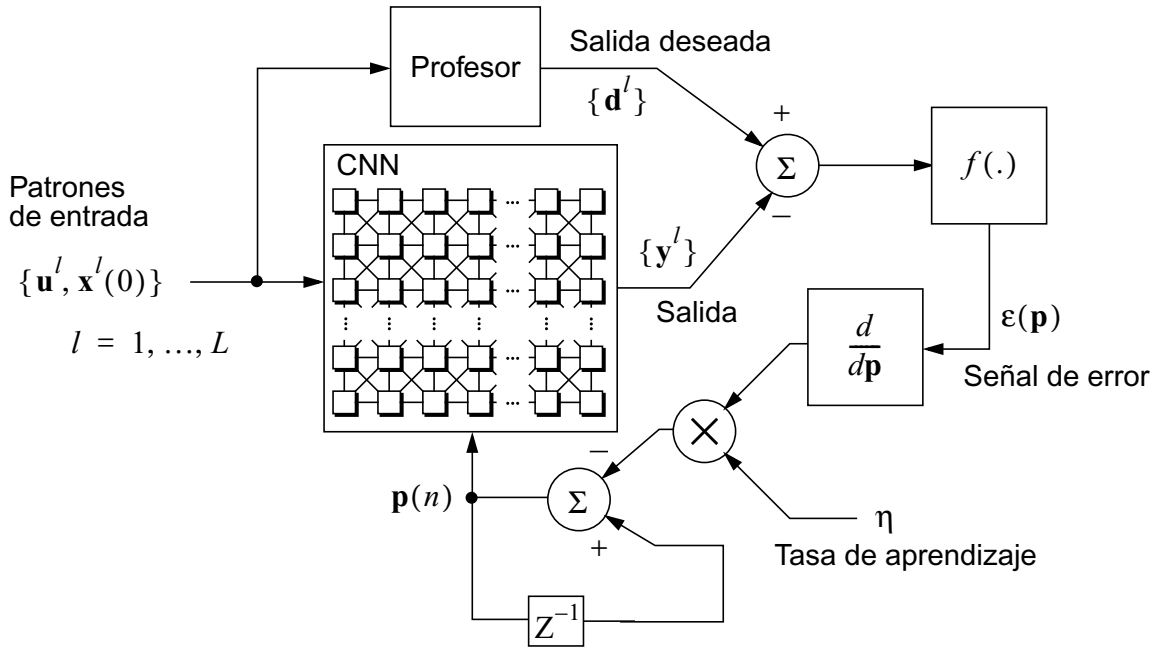


FIGURA 30. Algoritmo de aprendizaje basado en la evaluación del gradiente para una CNN.

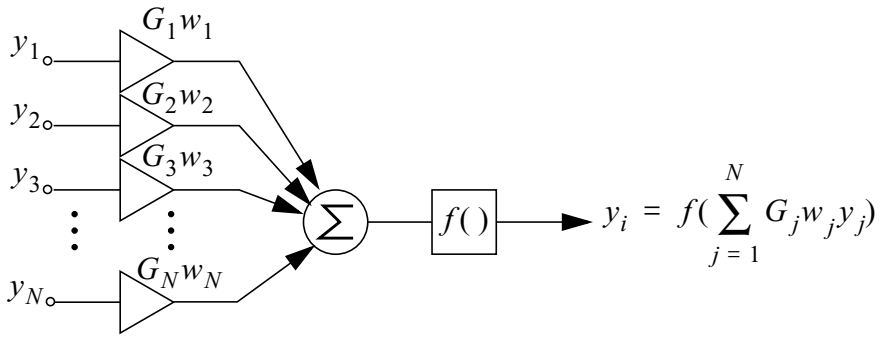


FIGURA 31. Modelo conceptual de una celda de una red neuronal.

mediante un procedimiento *chip-in-the-loop*, en el cual el chip de CNN realiza al evaluación de la función para cada patrón del *training set*, para un conjunto de parámetros, se busca un vector **p** que maximice las diferencias entre las salidas correspondientes a cada patrón de entrada. Esta búsqueda se realiza mediante algoritmos genéticos [Koze93] implementados en un ordenador. Los algoritmos genéticos son métodos de optimización estocástica basados en los mecanismos naturales de selección natural [Gold89]. No requieren la evaluación del gradiente de la superficie de error, y aseguran la convergencia incluso en espacios de búsqueda muy ruidosos.

Optimización del rango dinámico de los pesos

Para ilustrar otro método de mejora de la robustez de la implementación, tomemos como ejemplo el diagrama de la Fig. 31. En él se representa un celda de una red neuronal, que elabora señal de salida a partir de las contribuciones de las otras celdas. Supongamos que todos los bloques que implementan la conexiones sinápticas con otras celdas de la red tienen la misma fuerza. O sea, que para un peso w_i, ante una señal de entrada y_i, la salida de la sinapsis correspondiente será $G_0 w_i y_i$. En el caso de las CNNs, estos bloques conectan la celda con su vecinos más próximos. Para cada conjunto de templates que necesitamos implementar, podemos encontrar un valor máximo de entre sus elementos, $w_{\text{máx}}$, y un mínimo, $w_{\text{mín}}$, que puede ser, no sólo el más pequeño de los valores de peso a implementar, sino también, la diferencia más pequeña entre dos pesos adyacentes. La razón entre estos valores extremos:

$$\text{DR} = \frac{w_{\text{máx}}}{w_{\text{mín}}} \tag{47}$$

es el rango dinámico requerido para la implementación de dicho conjunto de templates. Un realización física con un nivel de precisión en la implementación de los pesos por debajo de DR, no va a exhibir el comportamiento que idealmente se espera de la red con estos *templates*. Supongamos que no estamos necesariamente interesados en realizar el mayor número de templates posibles, sino que pretendemos implementar un conjunto finito de templates, específicos de una cierta aplicación: $\mathbf{S} = \{S_i\}_{i=1,\ldots,N}$. Cada uno de los templates de este conjunto tendrá un

cierto requerimiento de rango dinámico: DR_i. Nótese que un escalado uniforme de todos los pesos no varia las especificaciones sobre el rango dinámico. Pues bien, si no existe ninguna dirección privilegiada en la red:

$$\mathrm{DR} = \text{máx}\{\mathrm{DR}_i\} = \text{máx}\left\{\frac{w_{\text{máx}_i}}{w_{\text{mín}_i}}\right\} \tag{48}$$

define el rango dinámico requerido para la implementación de dicho conjunto de templates. Pero, examinando grupos de templates relacionados con una misma aplicación [Rosk95], no es difícil encontrar que algunos de los pesos empleados muestran habitualmente unos valores más grandes que los que adoptan otros pesos. Sería interesante entonces tratar de codificar esta prevalencia de algunos pesos en la propia estructura de la red, con el fin de aliviar los requerimiento sobre el rango dinámico de los pesos. Pensemos en que cada bloque de la Fig. 31 está escalado por un factor distinto, en lugar de los G_0 tendremos G_1, G_2, …, G_N. a este factor lo llamaremos *fuerza* de la sinápsis. así la contribución de la sinápsis k-ésima será:

$$i_k = G_k w_k y_k \tag{49}$$

de modo que si G_k es n veces más grande lo que era antes, G_0, entonces, para una contribución idéntica a la anterior, necesitaremos un peso n veces más pequeño. O sea, que si concedemos más fuerza a las sinapsis que sueles estar pesadas por valores mayores de w_k, estos valores se reducirán consecuentemente, de modo que los requerimientos de DR disminuyen.

Un reforzamiento de una de las sinapsis de las que suelen soportar pesos grandes, no tiene por qué mejorar los requerimientos de la implementación, ya que a pesar de haberse reducido el DR correspondiente a algunos templates, puede haberse incrementado el máximo en otros templates. Así que nos plantearemos encontrar un vector de factores de escala $\mathbf{G} = \{G_1, G_2, \ldots, G_N\}$ que minimice el requerimiento de rango dinámico para los pesos, o sea, $\mathrm{DR}(\mathbf{G}) = \text{mín}\{\text{máx}[\mathrm{DR}_i(\mathbf{G})]\}$

Para encontrar este mínimo global tendremos que recorrer regiones del espacio de los parámetros para las que aparecerán multitud de mínimos locales. Con el fin de resolver este problema de optimización hemos empleado enfriamiento simulado (*simulated annealing*, SA) [Kirk83]. Este método de optimización de los requerimiento de rango dinámico de los pesos ha sido aplicado al diseño de las fuerzas relativas de las sinapsis en una CNN de dos capas (véase Apéndice 3). Haciendo uso de las simetrías que muestran los templates reportados en [Reke00b] tendremos sólo 6 elementos diferentes: los elementos de esquina y de centro de cara y central del template de realimentación a_{11}, a_{01}, a_{00}, el peso de conexión con la otra capa a_{12}, el término central del template de control b_{00}, que aquí se reduce a un sólo elemento, y el término de offset z. Si asignamos la misma fuerza a todas las sinapsis, necesitaremos un rango dinámico de $7/0.05 = 140$, o sea, una resolución equivalente en la implementación de los pesos de $7.13\,\text{bits}$, lo cual no es fácil de obtener en VLSI analógico. Partiendo entonces de este punto, y con una función coste de $\phi = 140$, iniciamos un proceso de optimización como el que

hemos descrito antes. Con esto llegamos a un mínimo de $\phi = 15.8114$, o lo que es lo mismo, $3.98\,\text{bits}$. Esto ocurre para los factores $\mathbf{G} = [1.0000, 1.1144, 12.3114, \quad 6.3246, 5.4063, 1.7265]$. Redondeando, para $\mathbf{G} = [1, 1, 12, 6, 5, 2]$ necesitaremos un rango dinámico de 16.67, o sea $4.01\,\text{bits}$. El problema es que el incremento en área derivado de estos factores de escala es muy grande, de modo que resulta conveniente inspeccionar los alrededores del mínimo hasta obtener un compromiso aceptable, por ejemplo, para el vector $\mathbf{G} = [1, 1, 3, 3, 3, 1]$, mucho menos costoso de implementar, obtenemos un rango dinámico de 33.3, $5.01\,\text{bits}$.

4. Almacenamiento de corto término en tecnología CMOS

El procesamiento de imágenes mediante algoritmos complejos, con resultados intermedios y flujos de cómputo irregulares, impone la necesidad de implementar en el CNN *chipset* un dispositivo de almacenamiento que permita la reorganización y adaptación de la información en un formato compatible con el procesador de CNN, y que no comprometa los requerimientos temporales de la aplicación.

4. 1. Errores en el proceso de almacenamiento de señales analógicas

Errores en la adquisición de datos

Generalmente, un circuito destinado a servir de memoria analógica, opera en dos modos diferentes. El modo de *adquisición*, *muestreo* o *seguimiento* (*sampling* ó *tracking*), durante el cual el circuito captura la señal, y el modo de *almacenamiento* o *retención* (*storage* ó *hold*), que es el periodo de tiempo que transcurre hasta que una nueva adquisición tiene lugar. Por causa de las desviaciones que alejan a los circuitos reales de su comportamiento ideal, los procesos de muestreo y retención de datos sufren de ciertos errores. Consideremos estas no idealidades en un circuito simple de muestreo y retención (*sample and hold*, S/H) como el formado por el transistor y el condensador de la Fig. 32(a). Tal como apunta [Nish93a], podemos encontrar las siguientes fuentes de error durante el modo de adquisición:

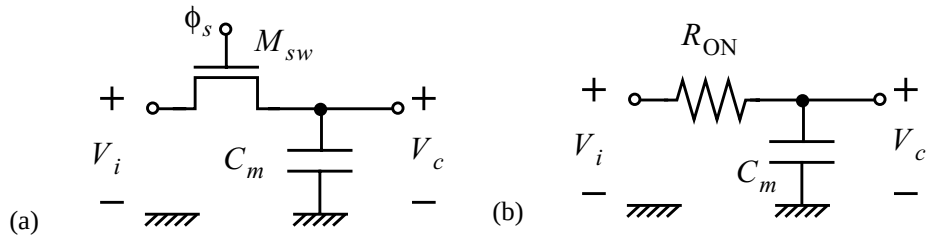


FIGURA 32. (a) Condensador y transistor de paso y (b) circuito equivalente en modo de adquisición.

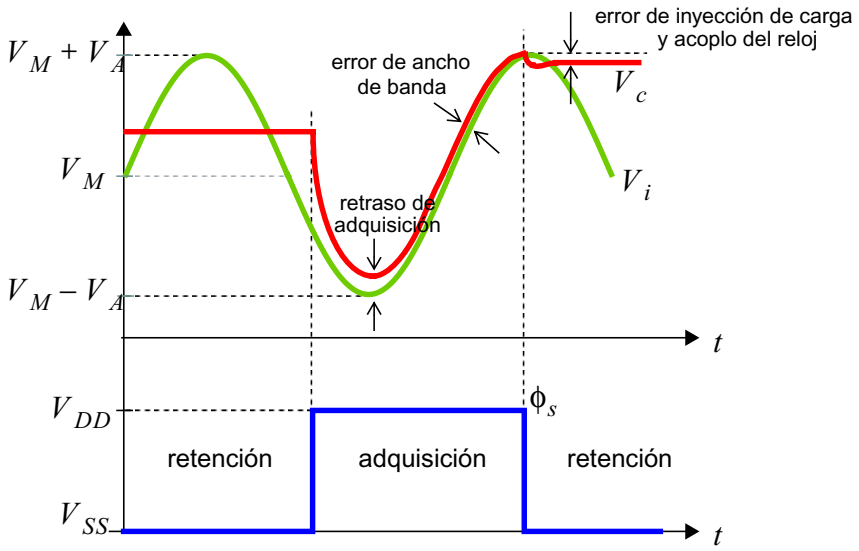


FIGURA 33. Errores en el muestreo de la señal.

Retraso de adquisición de la muestra: consistente en una ligera discrepancia entre la señal de entrada $v_i(t)$ y la señal que vamos a muestrear $v_c(t)$ (Fig. 33), que se debe al polo introducido por el circuito formado por la llave en ON, que podemos modelar como una resistencia R_{ON}, y el condensador C_m (Fig. 32(b)). Podemos hablar de una frecuencia de corte del filtro LP de primer orden dada por:

$$\omega_{-3\,\text{dB}} = (R_{\text{ON}} C_m)^{-1} = \tau^{-1} \tag{50}$$

Un sistema caracterizado por un polo a esta frecuencia, responderá a un escalón $v_i(t) = V_M + 2V_A u_o(t)$, como:

$$v_c(t) = V_M + V_A(1 - 2e^{-t/\tau})u_o(t) \tag{51}$$

de modo que el error entre $v_c(t)$ y $v_i(t)$ para cualquier instante será:

$$v_\varepsilon(t) = -2V_A e^{-t/\tau} \qquad \text{para} \quad t \geq 0^+ \tag{52}$$

lo que nos permite definir, a partir de la resolución equivalente objetivo (N bits), y sabiendo que $2V_A$ es el rango completo de la entrada, un tiempo máximo de adquisición, ya que debe cumplirse que $|v_\varepsilon(t_{\text{adq}})| \leq V_A/2^N$ para que el error cometido no supere el $1/2$ LSB. Así:

$$t_{\text{adq}} \geq \tau(N+1)\ln 2 \tag{53}$$

Por lo tanto, para reducir el tiempo de adquisición, podemos reducir el tamaño del condensador, con lo que C_m disminuye, o hacer más ancho el transistor de paso, reduciéndose R_{ON}. Ambas medidas contribuyen a, por un lado, aumentar la carga del reloj, y por otro, a un error debido a la conmutación (*switching error*) mayor.

Deplazamiento en la fase debido al ancho de banda finito: una vez que el transitorio de la adquisición se relaja, el circuito se encuentra de lleno en el modo de

seguimiento de la entrada. Pero debido al ancho de banda finito, observaremos, en régimen sinusoidal estacionario, un error de fase en la salida de:

$$\angle H(j\omega) = -\text{atan}(\omega\tau) \tag{54}$$

donde τ es la constante que definimos antes como $R_{\text{ON}}C_m$. Cualquier modificación introducida con el fin de disminuir el tiempo mínimo de adquisición, también tendrá efectos positivos en cuanto a este error de fase. Es conveniente recordar, de todos modos, que el uso de transistores de paso muy anchos para aumentar el ancho de banda en el modo de seguimiento (*tracking*) conlleva un aumento emparejado de la carga del reloj. Esto puede provocar una excesiva deformación del pulso del reloj (*clock skew*) y desembocar en un error de apertura (*aperture jitter*) grave.

Errores de inyección de carga y acoplo del reloj: al cerrar el acceso al condensador de almacenamiento mediante una llave real, observamos que la tensión retenida difiere de la que idealmente teníamos que haber muestreado. Esto se debe a la colaboración de dos fenómenos: la inyección de la carga (*charge injection*) que conformaba el canal hacia el condensador C_m una vez que el transistor entra en OFF, y el acoplo capacitivo de la entrada con la tensión almacenada después de la desconexión a través de las capacidades de solapamiento (*clock feedthrough*). El primero de estos fenómenos suele ser más importante en magnitud, y es de naturaleza aleatoria. La degradación que introduce en la tensión almacenada es:

$$\Delta V_{c_{ch}} = -\rho\frac{C_{ox}^{*}WL}{C_m}[V_{DD} - V_i - V_T(V_i - V_{SS})] \tag{55}$$

donde $V_T(V_{SB})$ es la tensión umbral para un determinado V_{SB} (véase Apéndice 4): En esta ecuación, el coeficiente ρ representa la fracción, no determinista, de la carga del canal que es inyectada en el condensador C_m, el peor caso sería $\rho = 1$.

En cuanto al *feedthrough*, hemos de observar que, una vez que el transistor de paso deja de estar en conducción, o sea, a partir de que la señal ϕ_s alcance el valor $V_i + V_T(V_i - V_{SS})$, se forma un divisor de tensiones capacitivo con el condensador parásito C_{gds} y C_m, entre la puerta de M_{sw} y V_c. El error introducido en la tensión almacenada es:

$$\Delta V_{c_{feed}} = -\frac{C_{gds}}{C_m + C_{gds}}[V_i + V_T(V_i - V_{SS}) - V_{SS}] \tag{56}$$

Ambos errores disminuyen al usar un condensador mayor o un transistor de paso más estrecho, justo al contrario que los errores de adquisición y de ancho de banda finito. De modo que habrá que alcanzar un compromiso. De todas formas, lo peor es que son dependientes de la señal.

Limitaciones debidas al ruido: además de las fuentes de error citadas, durante la fase de adquisición, el transistor de paso se comporta como una resistencia, lo que contribuye con un ruido blanco con una densidad espectral de potencia dada por [Gray93]:

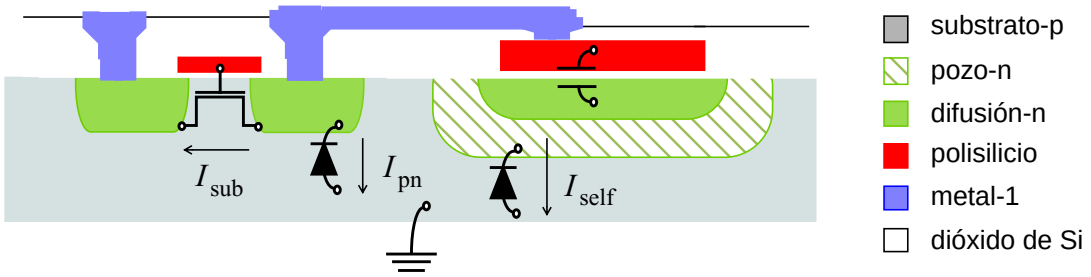


FIGURA 34. Perfil del condensador de polisilicio sobre difusión mostrando las fugas de corriente.

$$\overline{v_i^2} = 4kTR_{ON}(\Delta\omega/2\pi) \tag{57}$$

por lo que la potencia total de ruido a la salida será:

$$\overline{v_c^2} = \frac{1}{2\pi}\int_0^\infty |H(j\omega)|^2 \frac{\overline{v_i^2}}{\Delta\omega} d\omega = \frac{kT}{C_m} \tag{58}$$

que es independiente del tamaño del transistor. El ruido a la salida puede reducirse usando un condensador más grande, con lo que eso conlleva de reducción de la frecuencia máxima de muestreo. Así que $\sqrt{kT/C_m}$ constituye un límite para el rango dinámico del sistema. En nuestro caso, para un circuito S/H que opere sobre el rango de tensiones disponible para una alimentación de $3.3\,\mathrm{V}$, con un condensador de $0.1\,\mathrm{pF}$, obtenemos un rango dinámico máximo de $84\,\mathrm{dB}$, lo que no va a suponer un problema.

Fugas en el modo de retención

Durante el periodo de retención, y una vez descontados los errores de *switching*, las fugas de corriente provocan la degradación paulatina de la tensión almacenada en el condensador. Estrictamente, lo que se produce es una disminución de la carga acumulada y por tanto de la tensión entre las placas del condensador. Tendremos fugas asociadas al transistor de paso y fugas asociadas al condensador (Fig. 34):

Fugas a través del diodo de difusión del transistor de paso: este diodo, formado por la difusión n^+ del drenador/fuente de M_{sw} y el substrato tipo-p, está polarizado inversamente. Una vez cerrado el acceso al condensador C_m, el diodo extrae del mismo una corriente aproximadamente igual a la corriente inversa de saturación, dada por:

$$I_{pn} \approx I_0 = qA\left(\frac{D_p p_{n0}}{L_p} + \frac{D_n n_{p0}}{L_n}\right) \tag{59}$$

donde, q es la carga de un electrón ($1.602\times10^{-19}\,\mathrm{C}$ aprox.), A es el área de la unión pn, D_p y D_n son los coeficientes de difusión para huecos y electrones, L_p y L_n sus longitudes de difusión y p_{n0} y n_{p0} son las concentraciones de portadores minoritarios a cada lado de la unión [Sing94].

Fugas a través del transistor de paso: que corresponden a lo que sería la corriente subumbral del transistor MOS, o sea [Schro87]:

$$I_{\text{sub}} = I_{DS} = I_0 \exp\left(-\frac{qV_{GS}}{kT}\right)\left[1 - \exp\left(-\frac{qV_{DS}}{kT}\right)\right] \tag{60}$$

cuyo efecto se suma al de I_{pn} resultando en una fuga de corriente total en el rango de los pA.

Auto-descarga del condensador: en el caso de que el condensador esté implementado por una estructura del tipo mostrado en la Fig. 34, esto es, mediante una capa de polisilicio sobre difusión (separados por óxido fino), todo ello sobre un *well* débilmente dopado de tipo-n, entonces tenemos otra corriente de fuga asociada a la unión *well*-n/sustrato-p. Esta unión está inversamente polarizada, de manera que partiendo de la placa inferior del condensador y en dirección al substrato fluye una corriente:

$$I_{\text{self}} \approx qA_{\text{nwell}}\left(\frac{D_p p_{n0}}{L_p} + \frac{D_n n_{p0}}{L_n}\right) \tag{61}$$

donde los parámetros son los correspondientes a la unión citada. Esta I_{self} es del orden de los fA a la temperatura ambiente, lo cual limita el efecto de las fugas de la placa superior. De modo que la degradación en la tensión almacenada que se produce durante el periodo de retención viene dada por:

$$\frac{dv_c(t)}{dt} = -\frac{1}{C_m}\left(\frac{dq}{dt}\right) \approx -\frac{I_{\text{self}}}{C_a A_{\text{eff}}} \tag{62}$$

donde C_a es la capacidad por unidad de área de la estructura de polisilicio sobre difusión, y A_{eff} es el área del condensador ideal de placas plano-paralelas equivalente a la estructura en estudio. Si el condensador es suficientemente grande, $A_{\text{eff}} \approx A_{\text{nwell}}$, de modo que, reescribiendo la Ec. 62:

$$\frac{dv_c(t)}{dt} \approx -\frac{q}{C_a}\left(\frac{D_p p_{n0}}{L_p} + \frac{D_n n_{p0}}{L_n}\right) = -r_{\text{self}} \tag{63}$$

definimos una razón de auto-descarga que es independiente del tamaño del condensador, y que es propia de esta estructura. Un valor típico de esta razón en una tecnología CMOS estándar de $0.5\mu\text{m}$ es 50mV/s.

La integración de la Ec. 63 nos permitirá definir un tiempo de almacenamiento máximo, que será aquel para el cual ya no podemos asegurar que la tensión retenida haya sufrido una degradación por debajo de $1/2$ LSB. Si consideramos que el rango total de señal es $2V_A$, tenemos:

$$t_{\text{strg}} \leq V_A(2^N r_{\text{self}})^{-1} \tag{64}$$

o sea, 200ms si el error puede llegar a los 10mV. Estos datos deben tomarse como

orientativos, dado que estas fugas de corriente dependen fuertemente de la temperatura, pudiendo aumentar en tres órdenes de magnitud desde los $27C$ a los $70C$.

4. 2. Revisión de circuitos de memoria analógica

Características de una memoria analógica

Cualquier circuito, o dispositivo, concebido para funcionar como una memoria analógica (AM, *analog memory*) exhibirá ciertas características, como el tiempo de retención, la resolución, etc., que delimitan su comportamiento. Hagamos un repaso de estas características, particularizando las propiedades enumeradas en [Hori92] al diseño de un dispositivo de almacenamiento temporal de corto término en el *chipset* de CNN:

No volatilidad: es la habilidad del circuito, o dispositivo, de AM para retener la información hasta la próxima actualización. Existe no volatilidad con o sin alimentación. En nuestro caso nos interesa la no volatilidad de los datos mientras que la conexión a las fuentes de alimentación está habilitada, ya que pretendemos realizar una memoria caché [Poll90]. Los mecanismos para asegurar la no volatilidad de los datos pueden ser muy elaborados, sin embargo, debe tenerse en mente que la no volatilidad indefinida resulta irrealizable debido a la inevitable presencia de parásitos Así que hablaremos siempre de un tiempo de almacenamiento, que será el periodo de tiempo por encima del cual no puede asegurarse la precisión de los datos almacenados. En este caso, un periodo de $100\text{-}200\text{ms}$ es más que suficiente.

Resolución: este concepto se aplica habitualmente a los sistemas y herramientas de procesamiento digital de señal. Aquí vamos a extenderlo a la operación de circuitos analógicos. La resolución equivalente en circuitos analógicos no es más que el rango dinámico, el cociente entre el máximo de señal que el sistema puede manejar y el mínimo que va a ser capaz de discernir propiamente del ruido:

$$\mathrm{DR} = \frac{V_{\mathrm{máx}}}{V_{\mathrm{mín}}} \tag{65}$$

que puede expresarse tal y como está, o en decibelios, o, y de aquí su relación con la resolución en circuitos digitales, en bits, o sea, con el número de bits que necesitaríamos para representar semejante rango de señal:

$$\mathrm{N} = \log_2 \mathrm{DR} \tag{66}$$

Así que una precisión del $0.8\text{-}1.5\%$, que significa un rango dinámico entre 66.67 y 125, representa una resolución equivalente de $6\text{-}7\text{bits}$. Este va a ser el rango de precisión que vamos a necesitar en aplicaciones de procesamiento de imágenes a bajo nivel, en el plano focal. De hecho, el ojo humano dificilmente distingue entre dos niveles de gris adyacentes en una

escala de 256 niveles.

Acceso aleatorio, no destructivo y de alta velocidad: una característica relacionada tanto con la arquitectura como con el diseño del circuito es el acceso a la AM. En primer lugar, podemos suponer que mientras más estrecho sea el periodo de lectura o escritura, mejor. Estamos interesados en el procesamiento en tiempo real de gran cantidad de datos, por lo que el acceso a los mismo debe ser rápido, del orden de los 100ns. Este acceso, además de rápido tendrá que ser aleatorio, o al menos tener un cierto grado de aleatoriedad, por ser una memoria caché [Goor89]. Y finalmente, la recuperación de la información almacenada debe ser no destructiva. Está claro que toda transmisión de información requiere de una transacción energética, pero pueden disponerse medios para compensar la degradación de la información almacenada.

Tamaño de la celda de memoria: esta es otra característica que tendrá una fuerte incidencia en el diseño del sistema. Mientras más compacta sea la celda de memoria, mayor sea el número de celdas que podrán integrarse en un mismo chip y por tanto mayores serán las imágenes que pueden procesarse y almacenarse con un *hardware* muy reducido. Obviamente, como contrapartida, una celda pequeña será capaz de unos tiempos de retención menores y de una precisión restringida.

Comparación de alternativas tecnológicas

Las aplicaciones del almacenamiento de señales analógicas en el procesamiento analógico VLSI son muy diversas. Desde los pesos de una red neuronal, hasta líneas de retraso el procesamiento de video, pasando por moduladores sobremuestreados para conversión A/D y D/A en interfaces y comunicación. Existen por tanto diferentes alternativas, a distinto nivel, dispositivos, circuitos o sistemas, para la implementación de una memoria analógica fiable. Siguiendo la clasificación de [Hori92], tenemos:

Dispositivos de memoria digitales: pueden almacenarse las muestras de las señales analógicas como datos binarios, una vez convertidos, en dispositivos de memoria RAM convencionales. Tienen unas magníficas características de retención y fatiga, puesto que las señales digitales pueden ser reconstruidas con facilidad y precisión [Geig90]. Existen diferentes tipos de RAM CMOS, la RAM estática (SRAM), la RAM dinámica (DRAM), que utiliza una celda básica de memoria más sencilla pero necesita de un mecanismo de refresco que evite la degradación de los datos almacenados. Aunque el gran problema de usar estas memorias digitales para el almacenamiento de señales analógicas es la necesidad de circuitos de conversión A/D y D/A. Incluso empleando circuitos y técnicas del estado del arte, una implementación de este tipo requiere una enorme área de silicio y un aporte de potencia excesivo.

Memorias semiconductoras no volátiles (NVSM): se trata de una aproximación al problema desde el punto de vista del dispositivo. Mediante alguna modificación del proceso de fabricación CMOS pueden obtenerse dispositivos especialmente capacitados para el almacenamiento

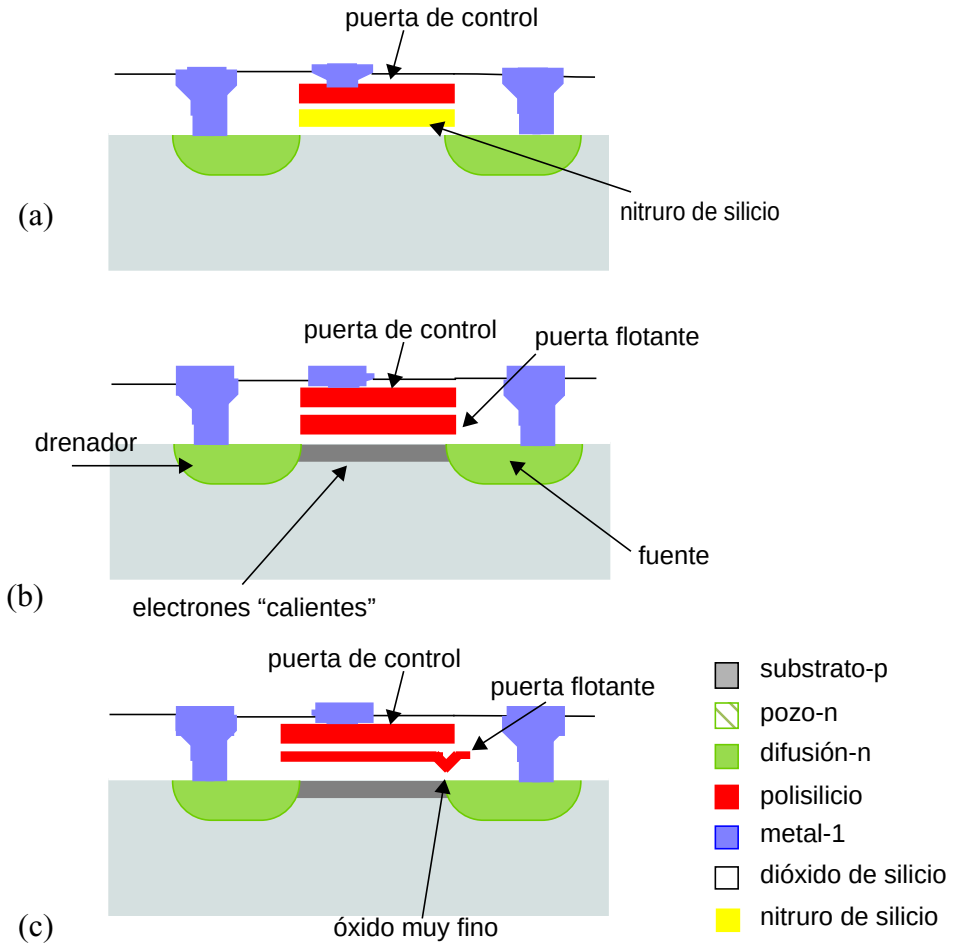


FIGURA 35. Estructura de (a) la trampa de nitruro de silicio, y los dispositivos (b) FAMOS y (c) FLOTOX.

de señales. Por ejemplo, insertando una capa de nitruro de silicio (Si_3N_4) entre la puerta y el óxido de puerta [Chan76] pueden quedar atrapadas algunas cargas en la entrefase de estos dos materiales, implementando así una cierta memoria al modificar la tensión umbral del transistor MOS [Tsiv87] (Fig. 35(a)). De manera similar, podemos tener diferentes tensiones umbrales en dispositivos con puerta flotante, como el FAMOS [Froh74] que opera por avalancha (Fig. 35(b)), presente en las EPROM (*erasable programmable* ROM), o el FLOTOX [Kolo86] que utiliza el efecto túnel, y que se usa en las EEPROM (*electrically erasable programmable* ROM) ya que el proceso de adquisición de cargas es reversible (Fig. 35(c)). El problema es que, en general para todas las NVSM, las tensiones requeridas para programar las memorias son relativamente altas, los tiempos de escritura y lectura son enormes y el control sobre la cantidad de carga inyectada no es muy fino, de hecho, estos dispositivos fueron concebidos como memorias digitales binarias.

Condensadores: un simple condensador con un transistor de paso, como en la Fig. 32, es la forma más sencilla de almacenar una tensión analógica, o digital, en un circuito VLSI. Es completamente compatible con tecnologías CMOS estándar, donde puede ser realizado mediante una estructura de placas planas y paralelas de polisilicio a dos niveles diferentes (poli- sobre poli-), o bien mediante polisilicio

sobre difusión, o bien simplemente mediante un transistor MOS cortocircuitado. A pesar de estar afectada por parásitos y efectos no contemplados por el modelo ideal, la operación de esta estructura es bien sencilla y está basada en la habilidad del condensador para almacenar carga. Según la ecuación constitutiva de un condensador ideal, existe una relación entre la cantidad de carga almacenada en las placas del condensador en un instante específico, $q_C(t)$, y la tensión entre dichas placas en dicho instante, $v_C(t)$. Esta relación es lineal e invariante en el tiempo, donde la constante de proporcionalidad es la capacidad del condensador, C_m. Así que en el momento en el que muestreamos la señal de entrada, la carga almacenada será $q_C(t_s)$, proporcional a la señal de entrada en dicho instante $v_i(t_s)$. Si podemos sensar esta carga almacenada sin que exista redistribución, entonces tenemos una memoria analógica. Para evitar esta redistribución será necesario aportar la carga que pueda perderse, como en los circuitos de muestreo y retención (S/H) de la Fig. 36. El primero de ellos realiza una lectura destructiva de los contenidos de la memoria, el segundo no. Debido a las fugas, de las que hemos hablado, a través de las llaves de paso y de los parásitos de las estructuras con las que se forman los condensadores, las tensiones almacenadas se degradan con el tiempo. Con el fin de compensar estas pérdidas de carga, pueden disponerse circuitos que corrijan esta desviación. Estas serían las memorias analógicas activas.

Memorias analógicas activas (AAM): se definen como memorias analógicas dinámicas de largo o medio término en las cuales se implementa *on-chip* algún tipo de mecanismo de compensación de la degradación de la tensión almacenada. Las características y operación de estas memorias son ajustables mediante el diseño del circuito. Suponen una aproximación desde el nivel de los circuitos a la realización de la AM, en contraposición a la aproximación desde en nivel de los dispositivos, como es el caso de las NVSM. Estas memorias son totalmente compatibles con las tecnologías CMOS estándar. Entre los diferentes tipos de AAM, que aparecen reportados en la literatura, tenemos la memoria analógica de condensadores gemelos [Hori90], en la que la diferencia entre las tensiones almacenadas en los condensadores C_1 and C_2 (Fig. 37(a)) se mantiene constante —suponiendo que las pérdidas no dependan del valor almacenado— y sirve para restituir los niveles originales de tensión, mediante un pulso de refresco originado cuando la referencia cae por debajo de un umbral. Otro método diferente consiste en utilizar un lazo de conversión A/D-D/A para compensar las pérdidas (Fig. 37(b)). Esta técnica con-

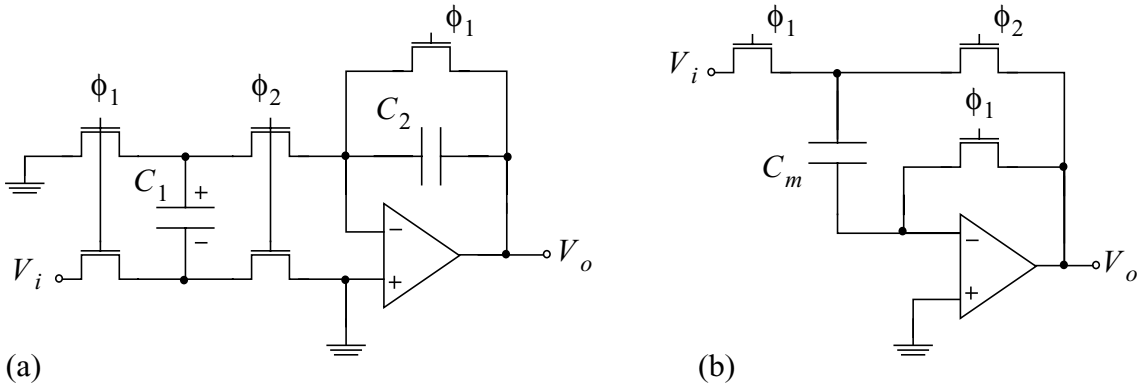


FIGURA 36. Circuitos de muestreo y retención (*sample and hold*, S/H).

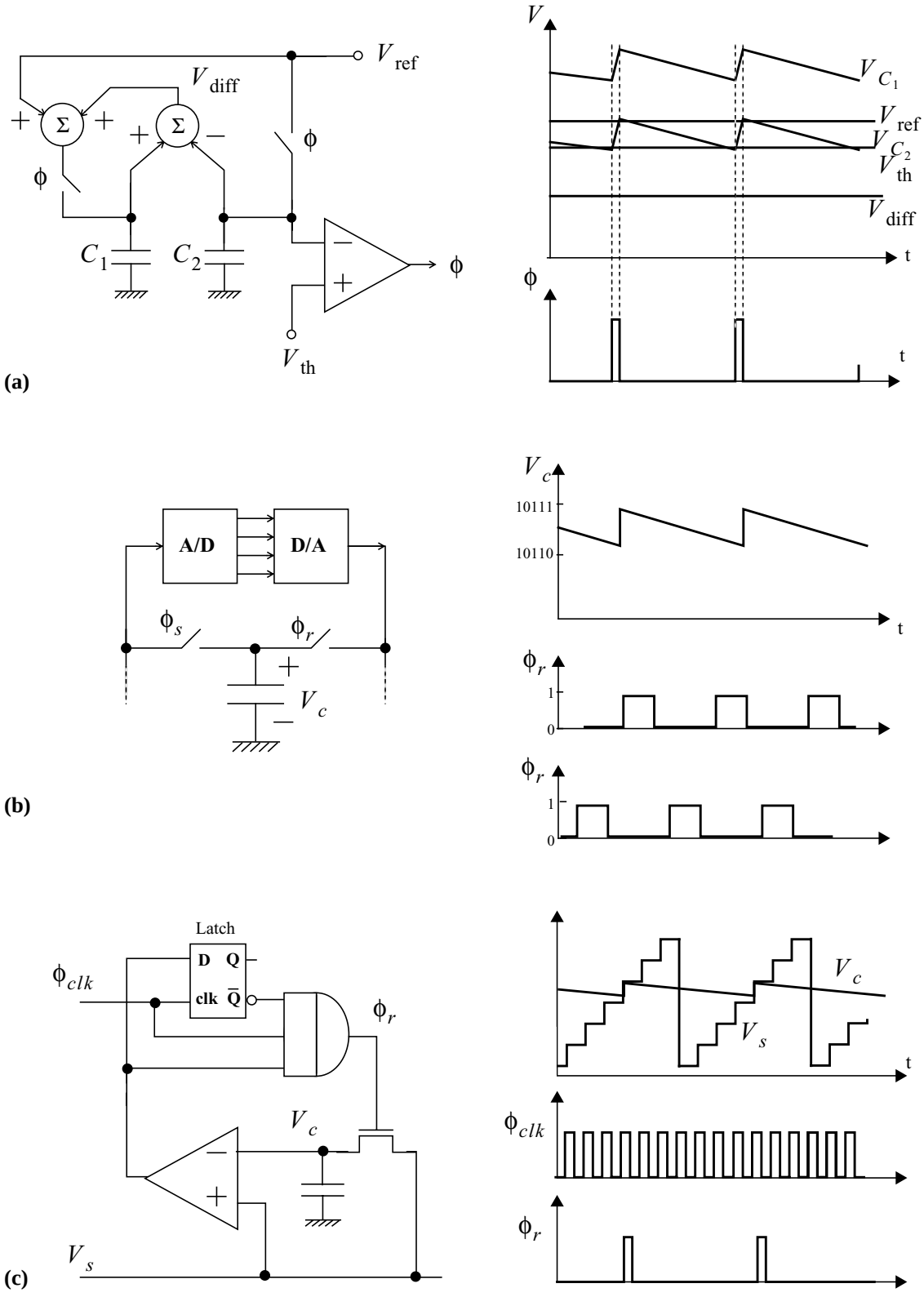


FIGURA 37. Ejemplos de memoria analógica activa (AAM): (a) condensadores gemelos, (b) lazo A/D-D/A, (c) señal de refresco en escalera.

siste en detectar la tensión almacenada, convertirla a un código digital y luego, volverla a convertir a una tensión analógica que regenera el valor almacenado. La frecuencia de refresco tiene que ser escogida de tal manera que la degradación sea menos que un LSB. Puede utilizarse el mismo lazo A/D-D/A para refrescar diferentes memorias sin más que establecer razonablemente el proceso de multiplexado de los convertidores [Lina91]. Un tercer ejemplo sería el circuito que emplea una señal de refresco en escalera, Fig. 37(c), en el que dicha señal que va recorriendo diferentes niveles preestablecidos (escalera) se compara con la señal almacenada y se utiliza para restaurar la señal degradada a uno de los niveles designados por la resolución prescrita [Vitt91]. Otras técnicas emplean detectores de fase, bombas de carga, etc. [Hori92].

4. 3. Un chip CMOS de RAM analógica

Existen varios trabajos reportados en la literatura en los que se implementa una memoria analógica en diferentes contextos. Algunos requieren de procesos especiales para la implementación de dispositivos NVSM, [Yong96] [Dior95] [Devo93]. Otros chips consideran la realización de una línea de retraso de video [Nish93b] [Hall89], de modo que aspectos como el acceso aleatorio y no destructivo no se contemplan. Otra alternativa presenta una RAM analógica (aRAM) cuya operación depende del apareamiento de ciertos dispositivos [Fran92]. Nosotros vamos a presentar un chip desarrollado en una tecnología CMOS de $0.5\mu m$ [Carm99b], que presenta claras ventajas al ser comparado con los anteriores.

Diseño del las líneas de S/H

Este chip de aRAM está compuesto por varios circuitos de muestreo y retención (S/H) basados en el S/H de Gregorian (Fig. 36(b)), con ganancia unidad y libre de *offset*, que puede utilizarse para muestreo de la señal mediante la placa de abajo del condensador. Esta técnica permite reducir notablemente la distorsión armónica que produciría un error de conmutación dependiente de la señal. Cada circuito S/H , contendrá varios condensadores de almacenamiento (Fig. 38) asociados al mismo amplificador de sensado. La lectura de los datos es no destructiva. En una primera aproximación, una tensión almacenada en el instante t_n en el condensador C_k, donde $1 \le k \le N$, es recuperada en el instante t_{n+1} sin verse afectada por el *offset* del *opamp*. O sea:

$$\left.\begin{aligned} V_{c_k}(n) &= V_i(n) - V_{OS} \\ V_o(n+1) &= V_{c_k}(n) + V_{OS} \end{aligned}\right\} \quad \text{así que} \quad V_o(n+1) = V_i(n) \quad (67)$$

Si consideramos que el opamp tiene una ganancia finita y que los parásitos no pueden despreciarse:

$$V_o = \frac{1}{1 + \frac{1}{A_o}\left(1 + \frac{C_p}{C_k}\right)}\left[V_i + \frac{1}{1 + A_o}\left(1 + \frac{C_p}{C_k}\right)V_{OS}\right] \tag{68}$$

de modo que error relativo cometido al recuperar una muestra puede aproximarse por:

$$\left|\frac{V_o(n+1) - V_i(n)}{V_i(n)}\right| \approx \frac{1}{A_o}\left(1 + \frac{C_p}{C_k}\right)\left|\frac{V_{OS}}{V_i(n)} - 1\right| \tag{69}$$

que se satura a $(1/A_o)(1 + C_p/C_k)$ para señales de entrada grandes, comparadas con el offset. Para señales de entrada pequeñas, el error se dispara, aunque esto ocurre a niveles en los que el ruido resulta determinante como factor de limitación de la amplitud de la entrada. Para una resolución equivalente de N bits, necesitaremos una ganancia mínima:

$$A_o > 2^{N+1}\left(1 + \frac{C_p}{C_k}\right) \tag{70}$$

por ejemplo, para tener 7bits de resolución, con parásitos tan grandes como el condensador nominal, necesitaremos un $A_o \geq 54.2\text{dB}$.

Otro aspecto importante del diseño de las líneas de S/H es el rango de salida (*output swing*, OS) limitado por el opamp. Para mantener un rango dinámico de 45dB, y que los errores de conmutación puedan ser del orden de 20mV, necesitaremos un OS de 2.6V. Conseguir esto con una tensión de alimentación de 3.3V requerirá usar un tipo de opamp que sea capaz de esto, por ejemplo, un Cascode plegado (*folded* Cascode) [Lake94]. Para concretar el diseño, utilizaremos las especificaciones dinámicas. Puesto que las señales a muestrear están limitadas en banda a un máximo de 4MHz, un

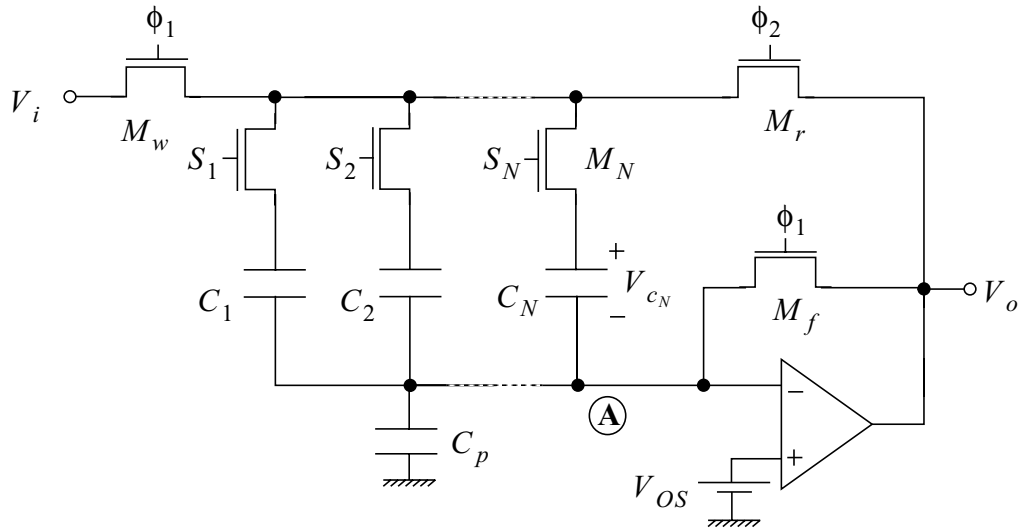


FIGURA 38. Esquemático de una línea S/H mostrando los parásitos y el offset del opamp.

$\mathrm{GB} = 20\mathrm{MHz}$ será más que suficiente. Dado que el GB viene dado por la razón g_{m_1}/C_L, donde g_{m_1} es la transconductancia de uno de los transistores del par diferencial de entrada del opamp, las dimensiones de estos transistores pueden calcularse mediante:

$$\frac{W}{L} = \frac{(C_L \mathrm{GB})^2}{2\beta_n^* I_B} \tag{71}$$

donde GB debe expresarse en rad/s. Para determinar el valor de I_B, usamos las especificaciones del *slew-rate*. De modo que para seguir una señal con componentes hasta los $4\mathrm{MHz}$, y con amplitud $2.6\mathrm{V_{p\text{-}p}}$, necesitamos un $\mathrm{SR} = 10.4\mathrm{V/\mu s}$. Dado que la capacidad de carga es dominante:

$$\mathrm{SR} = \frac{I_B}{C_L}. \tag{72}$$

Con esto podemos dimensionar el par diferencial, teniendo cuidado de no poner en peligro el margen de fase, al hacer estos transistores demasiado grandes y traer hacia menores frecuencias un polo no dominante.

Reducción del error de conmutación

Por causa de la dependencia del error de conmutación con la señal, tendremos una cierta distorsión armónica en la señal recuperada. Consideremos que la tensión almacenada es la suma de la entrada más un cierto error, $V_c = V_i + V_\varepsilon$, correspondiente a la inyección de carga y al *feedthrough*, despreciando los demás errores. V_ε puede expresarse como:

$$V_\varepsilon = k_0 + k_1[V_i + V_T(V_i - V_{SS})] \tag{73}$$

donde k_0 y k_1 son constantes independientes de V_i. Los primeros coeficientes de la distorsión armónica introducida son (véase Apéndice 4):

$$\begin{aligned} \mathrm{HD}_2 &= \frac{b_2}{b_1} = \frac{1}{2}A\frac{a_2}{a_1} = \frac{1}{4}A\frac{\partial^2 V_\varepsilon}{\partial v_i^2}\bigg|_Q \left(1 + \frac{\partial V_\varepsilon}{\partial v_i}\bigg|_Q\right)^{-1} \\ \mathrm{HD}_3 &= \frac{b_3}{b_1} = \frac{1}{4}A^2\frac{a_3}{a_1} = \frac{1}{24}A^2\frac{\partial^3 V_\varepsilon}{\partial v_i^3}\bigg|_Q \left(1 + \frac{\partial V_\varepsilon}{\partial v_i}\bigg|_Q\right)^{-1} \end{aligned} \tag{74}$$

Si ponemos en práctica la técnica conocida como muestreo por la placa baja (*bottom-plate sampling*) [Greg94], conseguiremos reducir notablemente dicha distorsión. Para ello, habrá que abrir la llave M_f algo antes que M_w, de manera que el error cometido, en primera instancia, no depende de v_i, por lo que se cancelarían los coeficientes HD_2, HD_3, etc., a cambio de un error constante, pedestal, en todas las muestras.

Tamaño del condensador: compromiso precisión-velocidad

Como hemos observado antes, cualquier actuación encaminada a mejorar la velocidad de operación del circuito básico formado por el condensador y el transistor de paso, tiene el efecto contrario en la precisión del muestreo, y viceversa. Tendremos que resolver este compromiso entre velocidad y precisión. Para una frecuencia de muestreo f_s, el error de adquisición será de:

$$V_{\varepsilon_{\text{acq}}} = -V_i e^{-1/\tau f_s} \tag{75}$$

donde V_i es la amplitud de la muestra, y $\tau = R_{\text{ON}} C_m$. Aquí, C_m es la capacidad del condensador de almacenamiento y R_{ON} es la resistencia de la llave, relacionada con sus dimensiones a través de:

$$R_{\text{ON}} = \frac{1}{\beta_n (V_{GS} - V_T - V_{DS})} \approx \frac{1}{\mu_0 C_{ox}^* (W_{\text{sw}}/L_{\text{sw}})(V_{GS} - V_T)} \tag{76}$$

Considerando que la contribución más importante al error de conmutación es la del error de inyección de carga:

$$V_{\varepsilon_{\text{sw}}} = -\left(\frac{C_{ox}^* W_{\text{sw}} L_{\text{sw}}}{C_m}\right)(V_{GS} - V_T) = -\frac{L_{\text{sw}}^2}{\mu_0 R_{\text{ON}} C_m} = -\frac{L_{\text{sw}}^2}{\mu_0 \tau} \tag{77}$$

Si usamos un transistor de paso de longitud mínima, podemos expresar el error total como función de τ:

$$|V_\varepsilon| = \frac{L_{\text{min}}^2}{\mu_0 \tau} + V_i \exp\left(-\frac{1}{\tau f_s}\right) \tag{78}$$

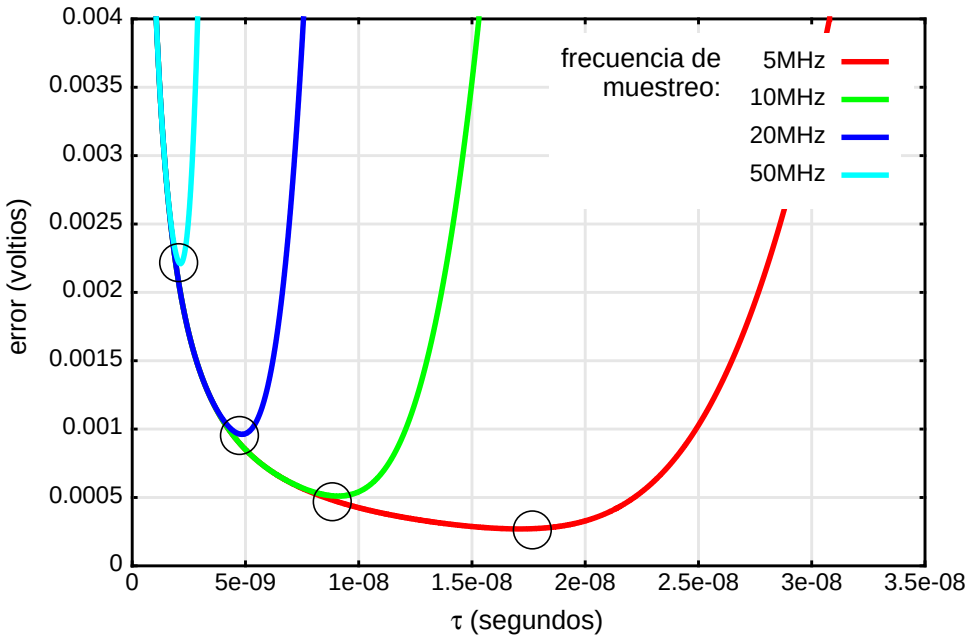


FIGURA 39. Compromiso entre velocidad y precisión.

Para frecuencias de muestreo altas, domina el error de adquisición, para bajas frecuencias es el error de conmutación. Como ambos dependen de τ, puede calcularse el valor de τ que hace mínima la suma. Si representamos el error total frente a τ (Fig. 39) podemos ver que existe un mínimo para cada frecuencia de muestreo:

$$\tau_{\min} = \left[f_s \ln\left(\frac{\mu_0 V_i}{f_s L_{\min}^2} \right) \right]^{-1} \tag{79}$$

Con esto tendríamos una relación entre R_{ON} y C_m que minimiza el error al muestrear. Si escogemos un transistor de paso de área mínima, sólo tendremos que dimensionar C_m para respetar esta relación. En esta ocasión tenemos un condensador de almacenamiento de $0.2\mathrm{pF}$.

Floorplan del chip de aRAM

El chip de aRAM propuesto consiste en una matriz de 32×256 celdas de memoria analógica, un total de 8192 registros organizados en 32 líneas, del tipo descrito anteriormente, con 256 condensadores cada una. Además de la matriz de celdas de memoria, el chip cuenta con los circuitos de interfaz necesarios para conectarlo a un chip que implemente una CNNUM, o a una fuente de señal de video, a través del bus analógico del *chipset* de CNN. Cada celda de memoria contiene un condensador y un transistor de paso, y, por grupos, algo de lógica local para decodificar la dirección. En la Fig. 40 podemos ver el *floorplan* del chip. El acceso a las celdas de memoria es totalmente aleatorio, sin más que apuntar convenientemente a la dirección del registro de interés mediante un código de 5 bits para indicar la fila y 8 bits para señalar la columna. Si se selecciona el modo paralelo de E/S, se descargan 16 filas en paralelo, empezando por la que indique el código. Fuera del *array* de celdas de memoria, las señales analógicas son *buffereadas* hacia el exterior del chip. Existe una señal de *strobe* global para evitar solapamientos en la selección de diferentes regis-

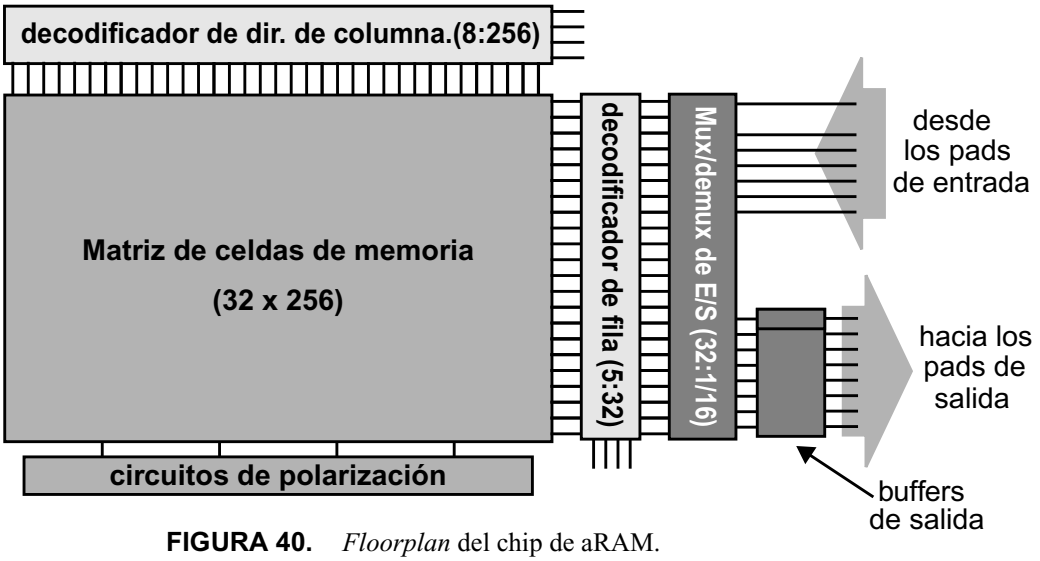


FIGURA 40. *Floorplan* del chip de aRAM.

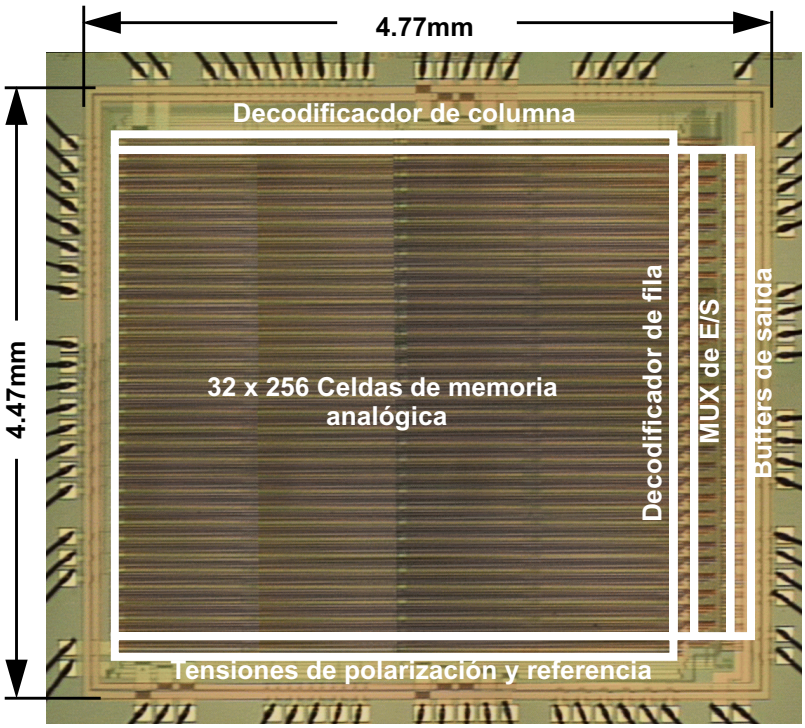


FIGURA 41. Fotografía del prototipo de chip de aRAM.

tros de memoria, lo que provocaría. Además se han implementado diferentes técnicas para evitar interferencias, como proveer caminos de retorno para las corrientes de substrato, apantallar las áreas sensibles al ruido de conmutación y separar las fuentes de ruido de los circuitos susceptibles de ser afectados por el ruido, bien en el espacio, o en el tiempo o en la frecuencia [Twom00] (véase Apéndice 4).

Resultados experimentales

El chip prototipo AM8K ha sido diseñado y fabricado en una tecnología CMOS de $0.5\mu m$ de resolución, un sólo nivel de polisilicio y tres niveles de metal. Contiene las mencionadas 32×256 celdas de memoria analógica que ocupan un área de $12.86 mm^2$. Lo que equivale a una densidad de $637 celdas/mm^2$. El área total ocupada por el chip, incluyendo la circuitería de interfaz, es de $21.32 mm^2$. La Fig. 41 muestra una microfotografía del prototipo, en la que se han indicado los diferentes bloques que componen el chip, que ha sido montado en una cápsula PGA-84. Un total de 24 muestras has sido testadas y han resultado ser funcionales. No se han observado discrepancias importantes entre los resultados obtenidos para cada una de ellas. Se ha obtenido una resolución equivalente de entre 7 y 8 bits, un tiempo de almacenamiento de $80\text{-}100 ms$, a temperatura ambiente en condiciones normales de operación. El consumo de potencia en DC es de $73 mW$ para una alimentación de $3.3 V$. Los tiempos de acceso a la memoria, tanto de escritura como de lectura, son de $100 ns$.

El montaje experimental utilizado para llevar a cabo los tests es el que se presenta en la Fig. 42. Para realizar el control digital del chip, hemos

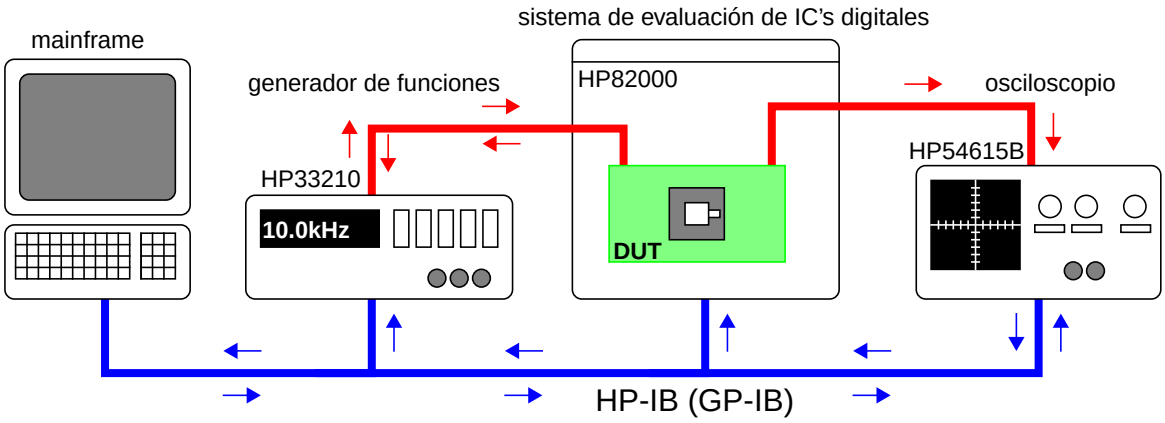


FIGURA 42. Montaje experimental para los tests y enlace de datos entre instrumentos.

empleado el sistema de evaluación digital HP82000 que nos ha servido, asimismo, para sincronizar los instrumentos. Las entradas analógicas se han originado mediante un generador de funciones arbitrarias HP33210. Las formas de onda de salida se han capturado mediante un osciloscopio digitalizador HP54615B. El control de los tests se realiza por ordenador, a través de un bus de propósito general (HP-IB), programado en el entorno VEE de Agilent. Los detalles se encuentran en el Apéndice 4.

Para medir la precisión del chip en la reproducción de la señal capturada, hemos empleado un tono de $1\,\mathrm{Vp\text{-}p}$ a $10\mathrm{kHz}$. La entrada ha sido adquirida a razón de $10\mathrm{Ms/s}$ (millones de muestras por segundo), o sea $100\mathrm{ns}$ de tiempo de acceso. La salida se ha descargado a la misma velocidad, $10\mathrm{Ms/s}$, ó $100\mathrm{ns}$ de tiempo de lectura desde fuera del chip. Una vez reconstruida la señal, hemos encontrado un offset medio de $-33.9\mathrm{mV}$. Una vez extraído este valor medio, el error cuadrático medio (RMSE) entre la entrada y la salida es de $3.9\mathrm{mV}$, lo que equivale a $8\mathrm{bits}$ de resolución equivalente (Fig. 43).Para cuantificar la eficiencia del método de muestreo propuesto a la hora de eliminar la distorsión armónica hemos obtenido el espectro de la señal de salida (suavizada para eliminar las componentes de muy alta frecuencia) obtenida a partir de las muestras (Fig. 44). Encontramos una diferencia de $46.8\mathrm{dB}$ entre el impulso tonal en los $10\mathrm{kHz}$ y la magnitud del segundo armónico.

También se han realizado medidas para caracterizar la linealidad del mapeo de E/S. Hemos medido la no linealidad diferencial y esta no supera el $1/2\,\mathrm{LSB}$ para ninguno de los 256 niveles de gris testados. En lo relativo a la no linealidad integral, suponiendo que la entrada está representada en 256 niveles de gris, las muestras de la salida correspondientes a dichos niveles de entrada, forman una recta con un coeficiente de correlación de 0.9999. Para más detalles, consultar el Apéndice 4.

En cuanto al tiempo de almacenamiento, hemos medido la degradación de la tensión almacenada en celdas diferentes de chips diferentes, componiendo la gráfica expuesta en la Fig. 45. Como estaba previsto, la tasa de auto-descarga del condensador conlleva una degradación lineal de la tensión almacenada confirmada ahora por las observaciones. Así para una resolución de $7\text{-}8\mathrm{bits}$, lo que supone una degradación de $5\text{-}11\mathrm{mV}$ en este contexto, la información puede ser retenida como mucho $60\text{-}80\mathrm{ms}$. Para $6\text{-}7\mathrm{bits}$, un error de $11\text{-}22\mathrm{mV}$, tendremos tiempos de almacenamiento de $80\text{-}100\mathrm{ns}$. Dado que para frecuencias de refresco de imágenes de $60\mathrm{Hz}$

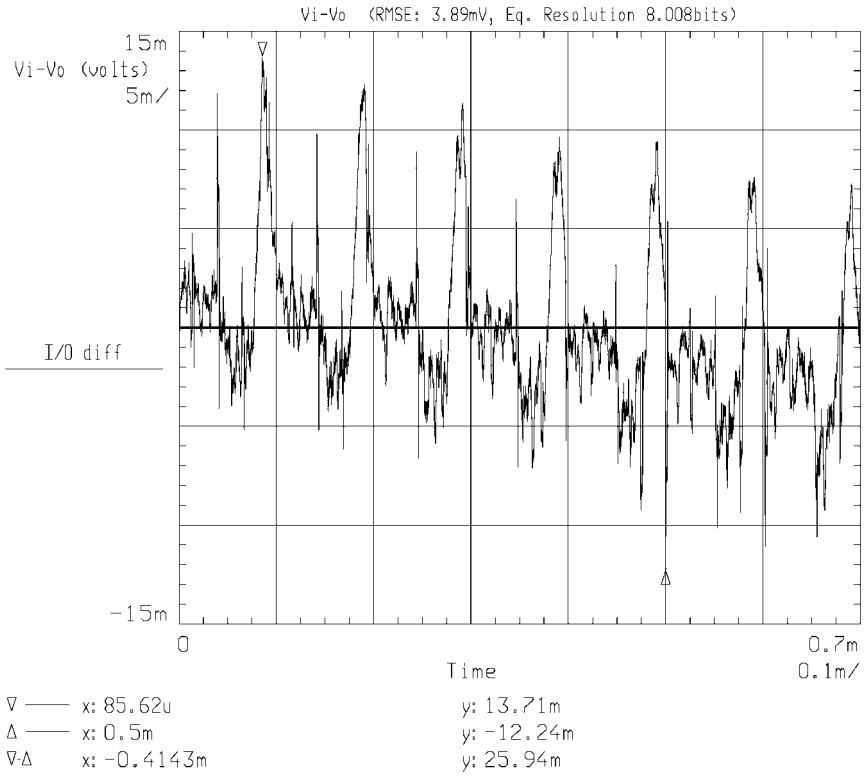


FIGURA 43. Diferencias entre la entrada y la salida (eliminando el valor medio del offset).

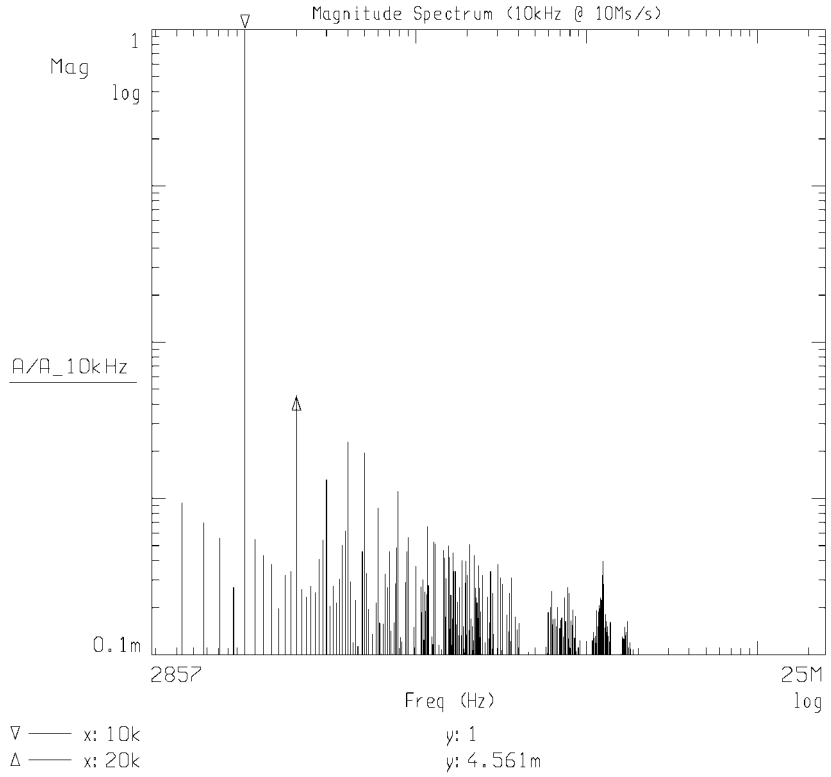


FIGURA 44. Espectro de la señal de salida (recortadas las componentes de mayor frecuencia).

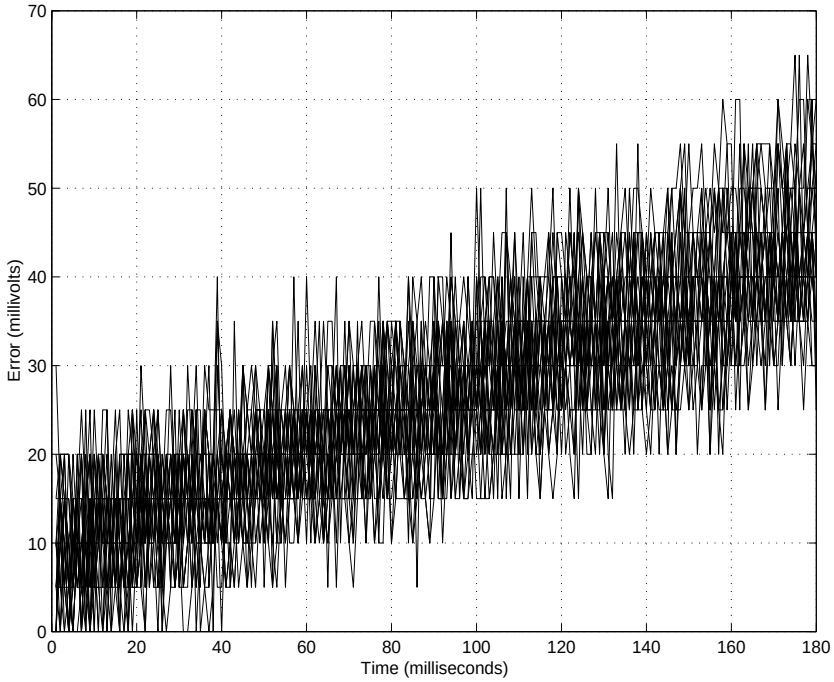


FIGURA 45. Degradación de la tensión almacenada (24 celdas)

tenemos 16.67ms por cada imagen, este chip de aRAM es capaz de mantener la precisión prescrita durante el tiempo suficiente.

Además de la caracterización a nivel eléctrico del prototipo, también se han realizado tests con imágenes reales[ix]. Por ejemplo las imágenes de 32×128 pixels, digitalizadas en diferentes niveles de gris, que pueden verse en la Fig. 46. Las imágenes (a), (c) y (e) son las entradas y (b), (d) y (f) son las salidas, las señales reconstruidas. Como puede apreciarse, las imágenes han sido adquiridas, almacenadas y descargadas sin pérdidas significativas de información. En el Apéndice 4 se muestran más tests con imágenes reales.

Comparación con chips de aRAM previos

Para establecer una comparación entre diferentes implementaciones CMOS de un chip de memoria RAM analógica, utilizaremos la figura de mérito (*figure of merit*, FOM) definida en [Good96]. Esta FOM relaciona el consumo medio de potencia, P_{med}, el retraso—para el que consideraremos la media entre el tiempo de adquisición (t_{adq}) y el de lectura (t_{lect})—y la precisión, 2^N, o sea, el número de niveles para tener una resolución equivalente de N bits:

$$\text{FOM} = \frac{P_{\text{med}}(t_{\text{adq}} + t_{\text{lect}})}{2^{N+1}} \tag{80}$$

ix. Resultados cedidos por el Dr. Péter Földesy del *Analogic and Neural Computing Laboratory* de la Academia de Ciencias de Hungría en Budapest

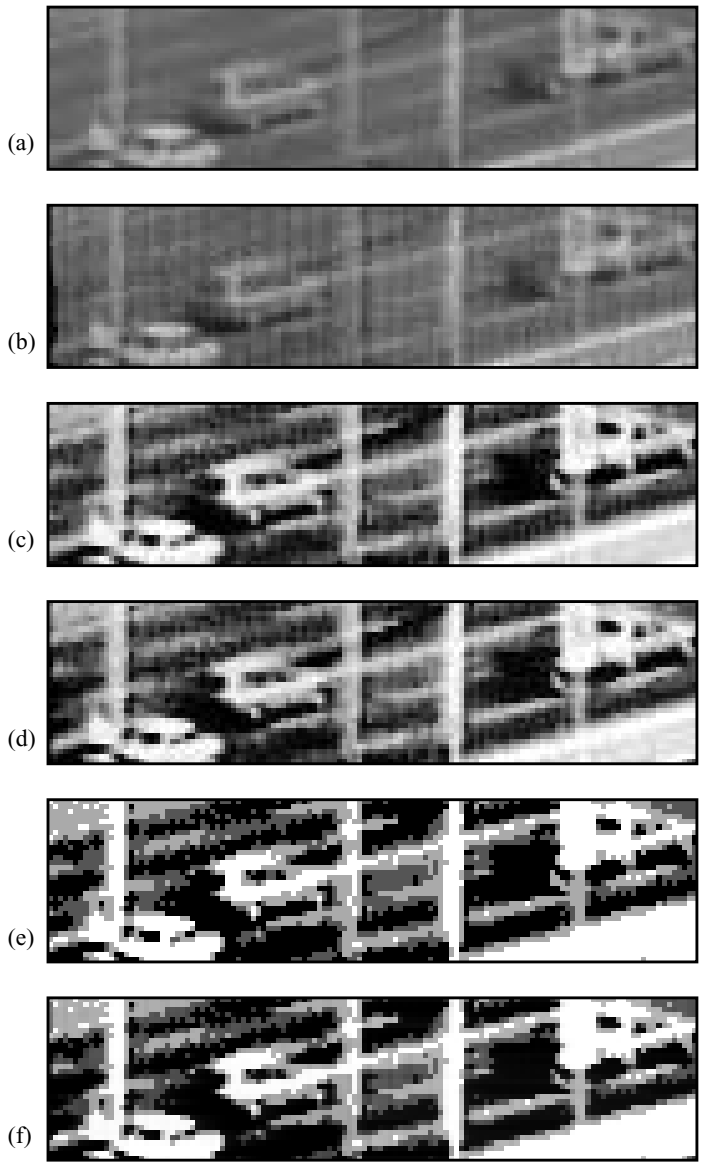


FIGURA 46. Ejemplos de tests con imágenes reales: (a), (c) y (e) entradas, y (b), (d) y (f) salidas.

Esta FOM tiene unidades de energía (J, julios en el SI). Mientras menos sea esta FOM, mejor será la implementación. Nótese que tanto una disminución en el consumo medio de potencia, una disminución en los tiempos de acceso a la memoria o un aumento de la precisión, redundan en una FOM menor. En la Tabla 5 podemos ver la FOM obtenida para diferentes implementaciones. La del chip presentado aquí es sensiblemente mejor, o sea, más pequeña.

TABLA 5. Comparación de chips de aRAM CMOS.

Autores del chip y año de publicación	[Franchi et al. 1992]	[Nishimura y Gray 1993]	[Simoni et al. 1995]	[Ping y Franca 1997]	Este chip
tecnología	CMOS 1.6µm/ 1P	CMOS 1.2µm/ 2P	CMOS 1.2µm/ 2P	CMOS 1µm/2P	CMOS 0.5µm/ 1P
tamaño (nº de celdas)	32 x 32	2 x 910	64 x 64	20 x 57	32 x 256
potencia	30mW	170mW	12mW	85mW	73mW
precisión	6-7b	8b	———	———	7-8b
rango/alimentación	1.9V@5V	2.6V@5V	———	———	1.4V@3.3V
acceso aleatorio	Yes	No	No	No	Yes
tiempo de escritura	800ns ♣♣	30ns ♣	16000ns ♣♣	56ns ♣	<100ns ♣♣
teimpo de lectura	300ns ♣♣,♠♠	<100ns ♣,♠	———	112ns♣,♠	<100ns ♣♣,♠♠
FOM [Good96]	0.1831pJ♦	0.0109pJ	0.1831pJ♦	0.0218pJ	0.0035pJ

♣. Dentro del chip ♠. Lectura destructiva
♣♣. Al exterior del chip ♠♠. Lectura no destructiva
♦. Asumiendo una resolución de 8b.

5. Una máquina de cómputo analógica celular en un chip

5. 1. Modelo bio-inspirado de CNN de dos capas

Esquema de la retina biológica

Después de millones de años de evolución, la naturaleza ha desarrollado uno de los mejores dispositivos para la visión existentes, en términos de consumo de potencia, rango dinámico y perfil físico: la retina de los vertebrados. Esta consiste en una agregación de fotosensores de amplio rango y de procesadores elementales que, si bien son simples, lentos y relativamente imprecisos, sí que muestran un comportamiento fuertemente cooperativo [Hube88]. La retina supone la solución natural al cuello de botella que produciría transmitir toda la información contenida en los estímulos visuales hasta la parte del cerebro que la va a procesar. Esta solución consiste en trasladar el procesamiento de bajo nivel al plano focal para así disponer de una información preprocesada, de un menor volumen de datos pero de mayor calidad, que transmitir al sistema nervioso central. Con la intención de imitar este logro de la naturaleza, se han desarrollado varios sistemas basados en silicio de estructura neuromórfica [Mead89], [Koch95], [Mead89]. Estos circuitos integrados suponen una alternativa bio-inspirada a los sistemas de visión artificial basados en el procesamiento digital convencional de la señal. Una secuencia de imáge-

nes de un tamaño habitual, 640×480 pixels, digitalizada en un esquema de $24\,\mathrm{b}$ por pixel y transmitidas a un ritmo de $30\,\mathrm{fps}$, supone un flujo de datos de más de $27\,\mathrm{MB/s}$. La alternativa analógica VLSI consiste en hacer concurrir la circuitería de procesamiento, de bajo nivel, con los dispositivos fotosensores, cosa que resulta más que posible en tecnologías CMOS convencionales. De este modo, el enorme grado de paralelización del procesamiento inherente a esta estructura permite obtener velocidades de operación mucho más altas que el procesamiento serializado.

Pero, esta emulación de la solución natural para el procesamiento a bajo nivel de la información visual no se limita a la asimilación de una estructura parecida a la de la retina de los vertebrados. Según demuestran recientes experimentos, un modelo basado en CNNs puede utilizarse para emular, con bastante fidelidad, el comportamiento de la retina [Reke00a]. al fin y al cabo, en ambos casos, en la retina biológica y en el modelo general de CNN, nos encontramos con una estructura bidimensional de red de procesadores elementales, que soportan una dinámica espacio-temporal que puede llegar a ser muy compleja, y que interaccionan entre ellos en un ámbito estrictamente local. Este modelo [Jaco96] ha sido desarrollado a partir de los estudios fisiológicos y farmacológicos iniciados por el Prof. F. Werblin en la Universidad de California, Berkeley [Werb91]. La estructura de la retina de los vertebrados es la que puede verse en la Fig. 47 [Werb95]. Una primera capa de fotosensores encima de todo, los conos —por otro lado están los bastones, que son fotosensores especializados en captar la luz en condiciones de muy poca iluminación, por lo que una vez acomodado el sistema a la iluminación circundante, los bastones se encuentran saturados— son los encargados de capturar el estímulo lumínico y convertirlo en un patrón de señales de activación. Las células bipolares transmiten estas señales a través de las diferentes capas de la retina hacia las ganglios, donde las señales de activación continuas se convierten en impulsos de potenciales de acción que pueden ser transmitidos a mayor distancia mediante el nervio óptico. En el camino hacia los ganglios, que es de unos micrómetros de longitud, las señales de activación se ven afectadas por la influencia de las celdas horizontales y amacrinas. Cuatro son, principalmente, las transformaciones que tienen lugar en esta estructura y que dan lugar al procesamiento del estímulo visual que realiza la retina: el control de la ganancia de los fotosensores, el control de ganan-

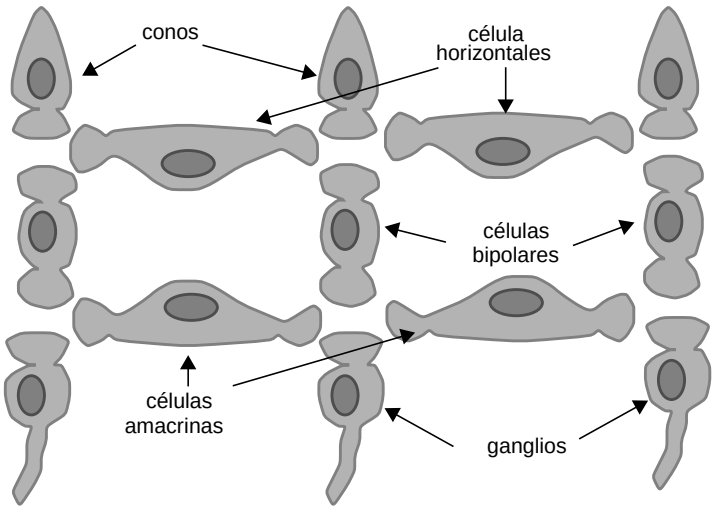


FIGURA 47. Diagrama esquemático simplificado de la retina de los vertebrados [Werb95].

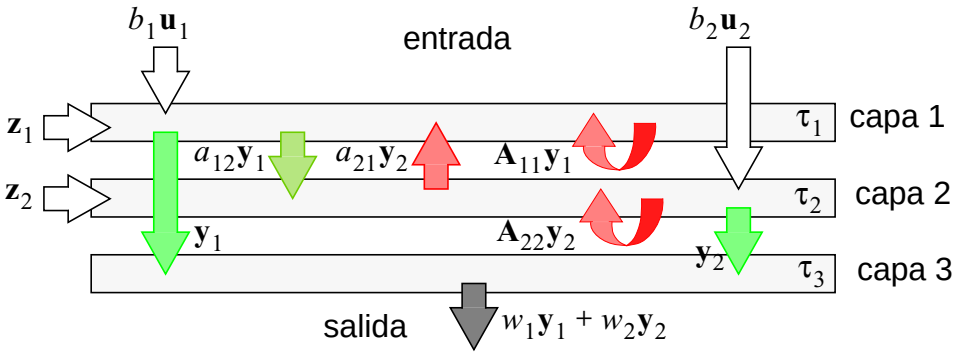


FIGURA 48. Diagrama conceptual de la CNN de 2do orden.

cia de las células bipolares, la generación de un patrón de actividad transitorio y la transmisión de un patrón transitorio de inhibición. Brevemente, los estímulos capturados se promedian para desplazar la característica de alta ganancia de los conos y de las células bipolares y así adaptarlas a las condiciones particulares de iluminación. Una vez que se ha conseguido la adaptación, se generan patrones de actividad de forma dinámica según se detecte la presencia o ausencia de ese estímulo visual. Al mismo tiempo, se ponen en funcionamiento los mecanismos de inhibición lateral desarrollados por las células horizontales y amacrinas. Así, muy afectados por la inhibición que reciben en las diferentes capas de células horizontales, los patrones de actividad llegan a los ganglios donde son convertidos en señales codificadas como pulsos, las cuales pueden ser enviadas al cerebro para ser interpretadas.

Como vemos, las coincidencias con el modelo general de CNN presentado al principio de este trabajo, son bastante patentes: red de procesadores elementales, no lineales, conexiones locales, pesos analógicos, etc. El siguiente paso es fijar un modelo de CNN que pueda concretarse en una realización VLSI analógica que implemente los procesos, algunos procesos, que se dan en la retina biológica.

Modelo para un chip de CNN de 2do orden y 3 capas

Tanto en la capa plexiforme interior (IPL) de la retina, como en la exterior (OPL) se ha observado la presencia de una estructura, a partir de la caracterización expuesta en [Reke00a], de tres capas. Cada una de estas capas tiene una estructura de CNN bidimensional por sí misma, con su propios patrones de interacción y su constante de tiempo particular. Y además se han encontrado ciertas interacciones dinámicas entre las señales que caracterizan el estado es de estas capas. Dada la relativa simplicidad de los modelos encontrados para la IPL y la OPL, se ha propuesto un chip que implemente una máquina universal basada en CNNs que incluya una dinámica de orden superior. En concreto, esta CNN de 2do orden y tres capas, va a consistir en dos capas de CNN acopladas mediante pesos de conexión inter-capas y una tercera capa adicional (Fig. 48), con una

constante de tiempo mucho más rápida que las anteriores, $\tau_3 \ll \tau_1, \tau_2$, que va a estar destinada a realizar operaciones aritméticas entre las variables de las capas enlazadas dinámicamente. La ley de evolución de una celda, $C(i, j)$, de esta CNN será:

$$\begin{aligned}
\tau_1 \frac{dx_{1,ij}(t)}{dt} &= -g[x_{1,ij}(t)] + \sum_{k=-r_1}^{r_1} \sum_{l=-r_1}^{r_1} a_{11,kl} \cdot y_{1,(i+k)(j+l)} + \\
&+ b_{11,00} \cdot u_{1,ij} + a_{12} \cdot y_{2,ij} + z_{1,ij} \\
\tau_2 \frac{dx_{2,ij}(t)}{dt} &= -g[x_{2,ij}(t)] + \sum_{k=-r_2}^{r_2} \sum_{l=-r_2}^{r_2} a_{22,kl} \cdot y_{2,(i+k)(j+l)} + \\
&+ b_{22,00} \cdot u_{2,ij} + a_{21} \cdot y_{1ij} + z_{2,ij}
\end{aligned} \tag{81}$$

donde, para el término de pérdidas y para la no linealidad de la salida, adoptaremos las funciones definidas para el modelo FSR de CNN (Apéndice 1).

En la Fig. 49 podemos ver un diagrama de bloques del modelo descrito por la Ec. 81. Cada elemento de proceso consiste en realidad en dos celdas, nodos, pertenecientes a las dos capas de la red, con una cierta interacción entre ellas. En este

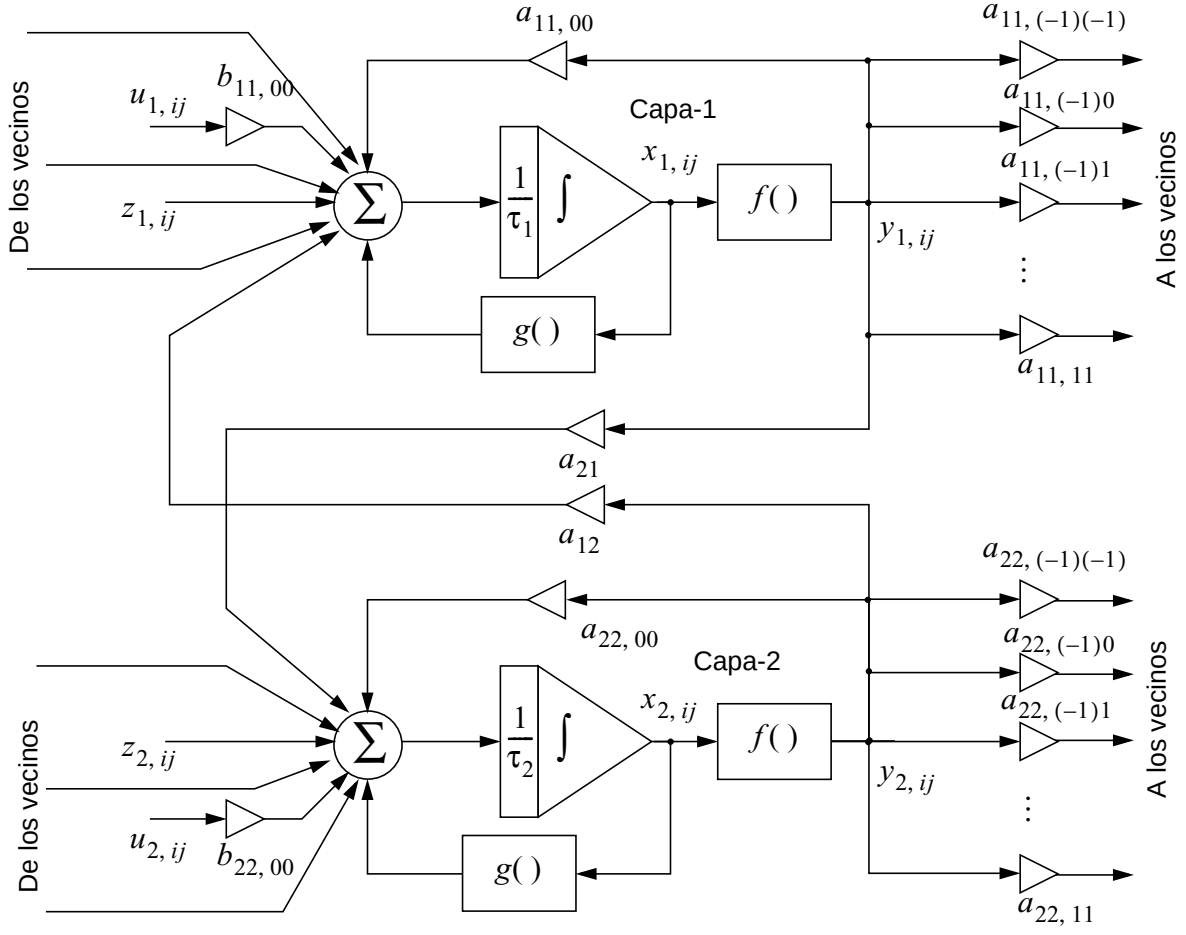


FIGURA 49. Diagrama de bloques de una celda básica con nudos en ambas capas de CNN.

modelo pueden programarse diferentes dinámicas sin más que ajustar convenientemente los pesos de conexión dentro de las capas y entre capas, así como las constantes de tiempo, o la relación entre ambas.

Ejemplos de operación

La dinámica de esta CNN de dos capas puede programarse para mostrar: fenómenos de propagación de ondas en dos dimensiones, generación de patrones, procesamiento de imágenes, etc. El Dr. Csaba Rekeczky de la Academia de Ciencias de Hungría en Budapest, ha obtenido un conjunto de *templates* [Reke00b], de los cuales vamos a presentar alguno a modo de ejemplo de operación. Para empezar, consideremos un conjunto de *templates* que provoca que diferentes estructuras 2D de la imagen original crezcan, o se propaguen, en todas direcciones, en la forma en la que lo hacen las ondas. Para ello, no necesitamos *template* de control, $b_1 = b_2 = 0$, ni lo que formalmente consideramos imágenes de entrada, $\mathbf{u}_1$ y $\mathbf{u}_2$. En su lugar, tendremos unos estados iniciales para ambas capas, $\mathbf{x}_1(0)$ y $\mathbf{x}_2(0)$, a partir de los cuales la red evolucionará. Los *templates* de realimentación $\mathbf{A}_{11}$ y $\mathbf{A}_{22}$, deben valer:

$$\mathbf{A}_{11} = \mathbf{A}_{22} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.00 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \tag{82}$$

y la relación que debe existir entre las constantes de tiempo es:

$$\frac{\tau_1}{\tau_2} = \frac{1}{5} \tag{83}$$

Nos faltan los pesos de conexión entre capas, elementos a_{12} y a_{21}, y el término de *offset* para cada una de ellas, z_1 y z_2. Para este ejemplo:

$$a_{21} = 3.00 \qquad a_{12} = -5.00 \qquad z_1 = -1.25 \qquad z_2 = 2.25 \tag{84}$$

Pues bien, en estas condiciones, la red muestra el comportamiento expuesto en la Fig. 50. Varios frentes de onda que se generan en diferentes puntos y que se aniquilan al colisionar.

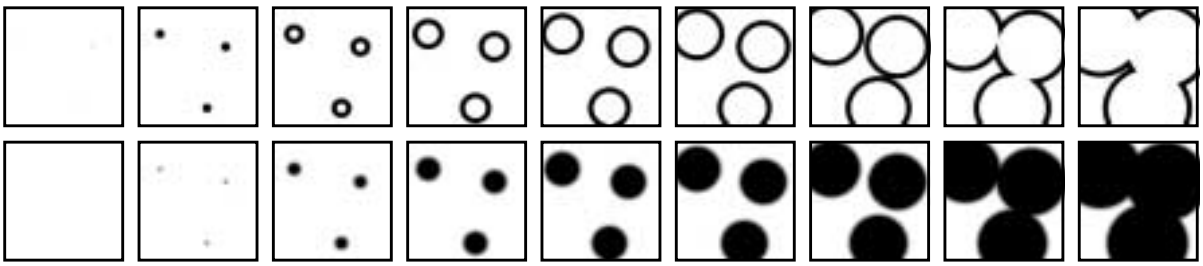


FIGURA 50. Ondas viajeras en el modelo de CNN de 2 capas ($\tau_1 = 0.2\mu s$ y $\tau_2 = 1\mu s$).

Otro ejemplo que ilustra las posibilidades de procesamiento de este circuito puede ser la detección de contornos. Esta operación, que es de vital importancia para la segmentación de imágenes, descansa en los siguientes *templates*:

$$A_{11} = \begin{bmatrix} 0.05 & 0.20 & 0.05 \\ 0.20 & 0.00 & 0.20 \\ 0.05 & 0.20 & 0.05 \end{bmatrix} \qquad A_{22} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.00 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix}$$

$$a_{21} = -2.50 \qquad a_{12} = -0.10$$

$$b_1 = 1.00 \qquad b_2 = 0.00 \tag{85}$$

$$z_1 = 0.00 \qquad \text{y} \qquad z_2 = 1.25$$

mientras que ambas capas tienen la misma constante de tiempo: $\tau_1 = \tau_2$. En esta ocasión, un punto en la imagen colocado como etiqueta del segmento que se quiere extraer, crece mientras que en la otra capa la imagen se va difundiendo para evitar que se computen como bordes los detalles pertenecientes a la textura del *parche* que queremos segmentar(Fig. 51).

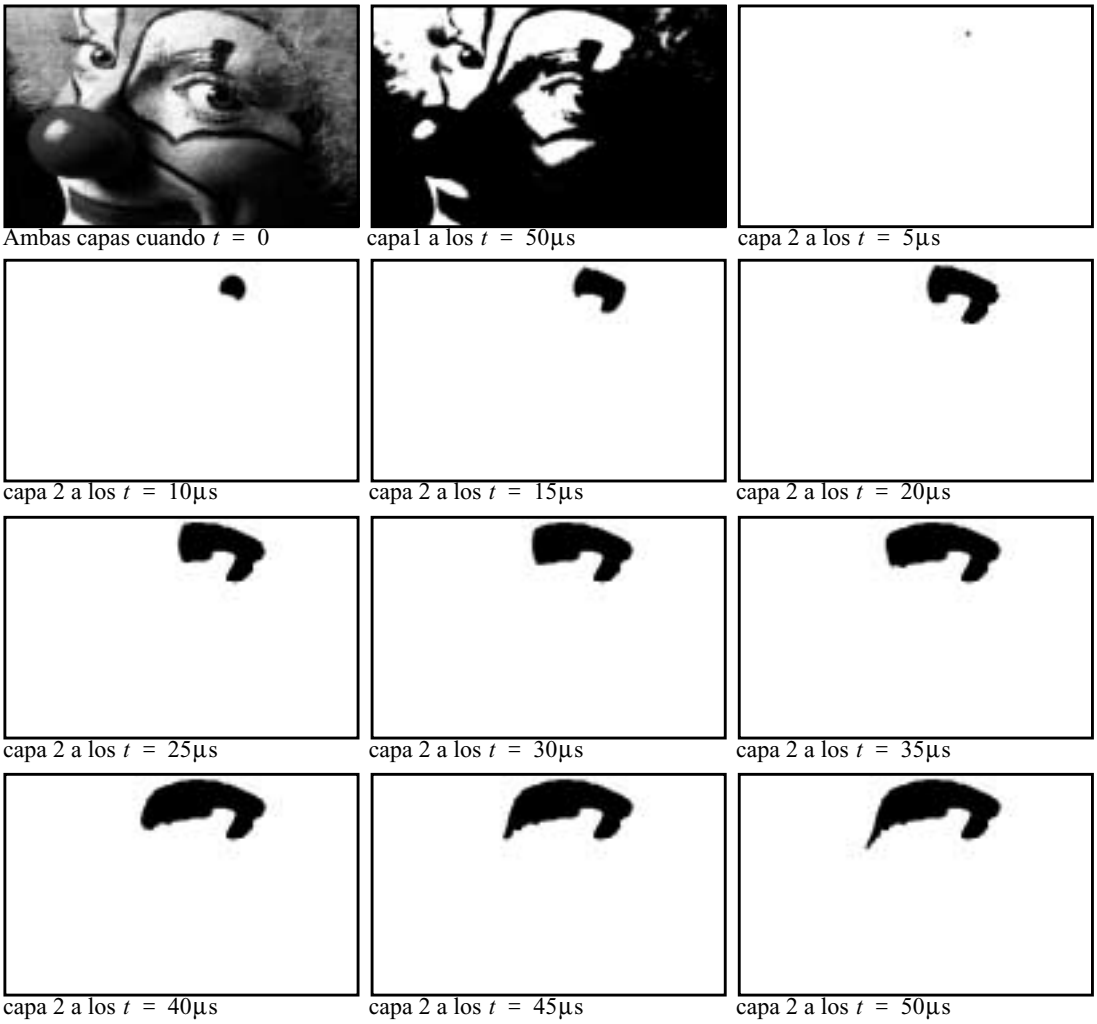


FIGURA 51. Detección de contorno en las CNN de 2 capas ($\tau_1 = 1\mu s$ y $\tau_2 = 1\mu s$).

Si utilizamos la información del movimiento en una secuencia de imágenes para generar los marcadores o etiquetas que disparan la segmentación de un parche de la imagen original, esto templates nos sirven para detectar contornos activos. esta operación tiene mucha importancia en aplicaciones médicas, p. ej. en ecografía donde la imagen obtenida no es de mucha resolución.

5. 2. Diseño de un chip CMOS del modelo bio-inspirado

Estructura de la celda básica

Basándonos en el modelo bio-inspirado antes descrito, vamos a abordar la realización de un sistema analógico VLSI, en tecnología CMOS, que permita emular ciertas funciones de la retina. El procesador elemental de esta red, la celda básica, tiene una estructura similar a la de la celda de la CNNUM, sólo que el *core* analógico, ahora implementa una CNN de 2 capas. En la Fig. 52, podemos ver que la celda básica contiene, aparte de las memorias lógicas y analógicas (LLMs y LAMs) y la unidad lógica (LLU), dos núcleos analógicos de procesamiento que constituyen los nudos pertenecientes a cada una de las dos capas de la red. Las dos capas difieren en que los nudos correspondientes a la primera capa tienen una constante de tiempo escalable, mientras que los de la otra capa la tienen fija. La estructura de estos nudos, que también se muestra en la Fig. 52, se basa en un nudo de entrada que recibe las contribuciones en corriente de los vecinos, sumándolas, y luego integrando la suma en el condensador de estado. Este condensador está afectado por un limitador que impide que la variable de estado se salga de determinado rango (modelo FSR). Para poder utilizar sinapsis de un solo transistor, como la que veremos a continuación, hace falta un dispositivo que retenga y elimine el offset de corriente. La Fig. 53 muestra el *layout* de la celda básica, indicando sus

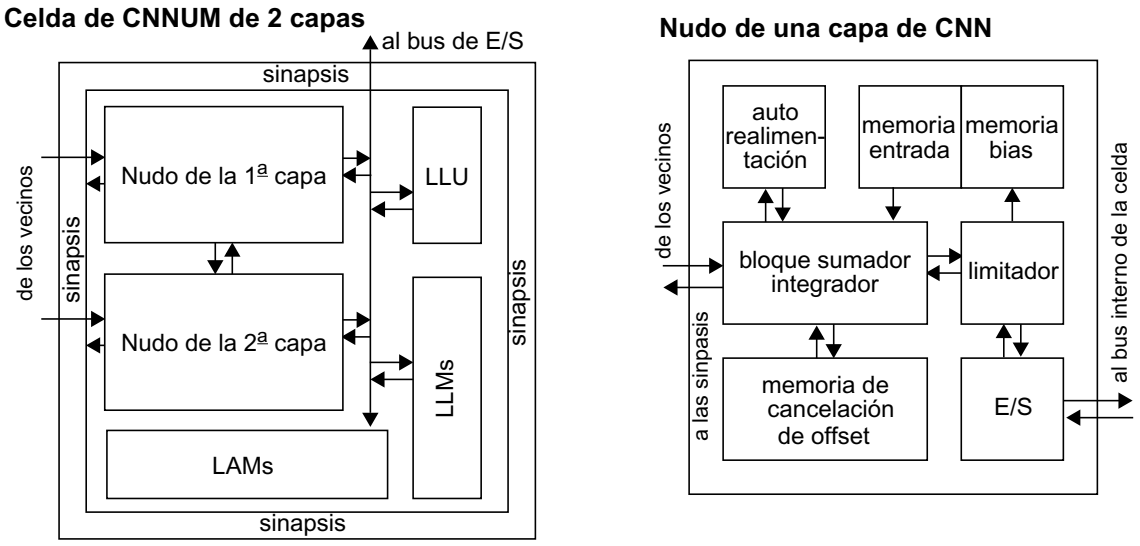


FIGURA 52. Diagrama conceptual de la celda básica y estrcutura interna de los nudos de las capas de CNN.

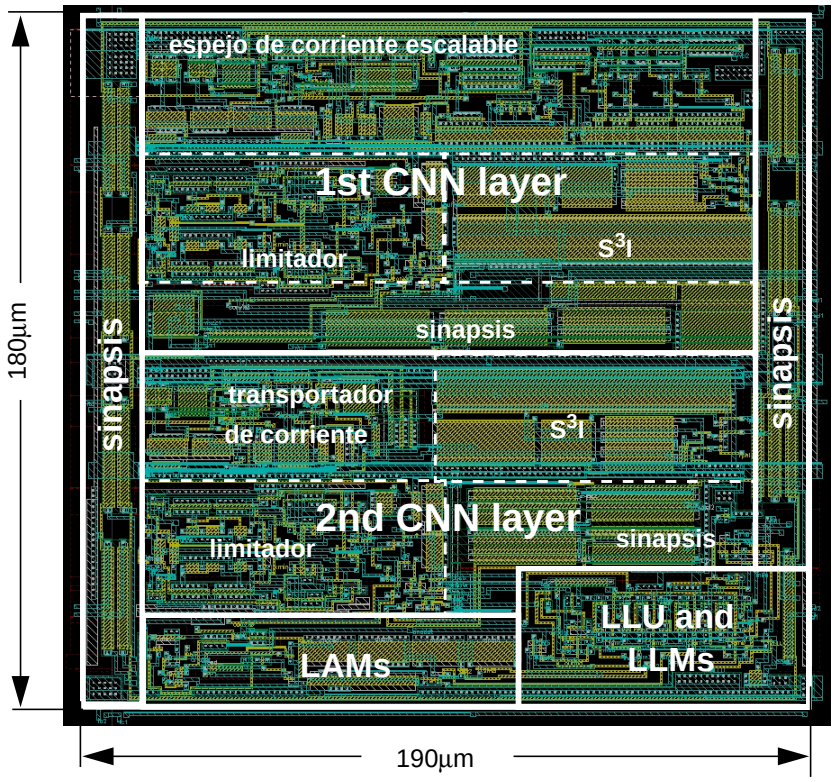


FIGURA 53. *Layout* de la celda básica.

dimensiones, no se incluyen las dos últimas capas de metal, con el fin de poder observar todos los bloques internos. Pueden apreciarse las diferencias entre la primera y la segunda capas.

Sinapsis de un único transistor

Uno de los bloques más importantes en la celda básica de la CNN es la sinapsis, dado que la mayoría de los circuitos que integran dependen de su implementación. Un bloque sináptico no es más que un multiplicador de cuatro cuadrantes. Sus entradas serán el estado de la celda y el peso de la conexión, mientras que la salida, una señal relacionada con el producto de las entradas, es la contribución al vecino. Por conveniencia, las entradas son en tensión y la salida en intensidad. así que, idealmente, la sinapsis tendrá la siguiente característica en gran señal:

$$I_o = kV_wV_x \tag{86}$$

donde V_w y V_x representan las señales correspondientes al peso y al estado de la celda, mientras que I_o es la corriente de salida. Estas señales tendrán sino positivo o negativo, y aunque va a ser necesario que I_o sea lineal con V_x, no hace falta que lo sea con V_w ya que, en este tipo de procesamiento, la señal de peso no cambia durante la evolución de la red. Además, cualquier desviación que no dependa de V_x podrá cancelarse con una calibración previa. Existen multiplicadores basados en las diferentes regiones de operación del transistor MOS [Espe94a]. Por su simplicidad y por

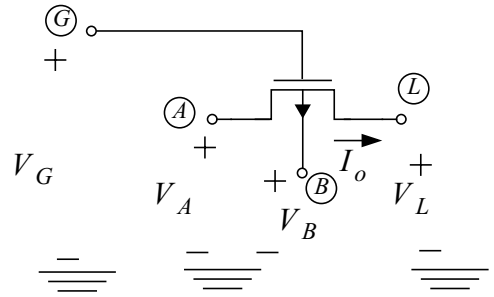


FIGURA 54. Multiplicador de un solo transistor MOS en zona óhmica.

la relación entre la potencia de la señal y la potencia de polarización, hemos usado aquí una sinapsis de un sólo transistor, operando en la zona óhmica [Domí98]. Para describir su operación consideremos un transistor de tipo-p, que es más resistivo, por tanto para un determinado nivel de intensidad, necesitaremos un transistor más corto que si fuera de tipo-n, operando en zona óhmica (Fig. 54). La intensidad de la corriente que va de la fuente al drenador de este dispositivo es:

$$I_o = \beta_p \left[V_A - V_G - \left| V_{T_p}(V_{DD} - V_A) \right| - \frac{V_A - V_L}{2} \right] (V_A - V_L) \tag{87}$$

si $V_A \geq V_L$, lo que significa que el nudo (A) es la fuente y el nudo (L) es el drenador. Aquí $\beta_p = \mu_o C_{ox}'(W/L)$ y la tensión umbral:

$$V_{T_p}(V_{BS}) = -\left| V_{T0_p} \right| - \gamma(\sqrt{\phi_B + V_{BS}} - \sqrt{\phi_B}) \tag{88}$$

Si por el contrario, $V_A \leq V_L$, el nudo (A) pasa a ser el drenador y (L) la fuente. De modo que I_o vendrá dado por:

$$I_o = -\beta_p \left[V_L - V_G - \left| V_{T_p}(V_{DD} - V_L) \right| - \frac{V_L - V_A}{2} \right] (V_L - V_A) \tag{89}$$

Ambas ecuaciones pueden resumirse en:

$$I_o = -\beta_p (V_A - V_L) V_G - \beta_p (V_A - V_L) \left(\left| \hat{V}_{T_p} \right| - \frac{V_A + V_L}{2} \right) \tag{90}$$

donde la tensión umbral será la que corresponda a cada caso:

$$V_{T_p} = \begin{cases} -\left| V_{T0_p} \right| - \gamma(\sqrt{\phi_B + V_{DD} - V_A} - \sqrt{\phi_B}) & \text{si } V_A \geq V_L \\ -\left| V_{T0_p} \right| - \gamma(\sqrt{\phi_B + V_{DD} - V_L} - \sqrt{\phi_B}) & \text{si } V_A \leq V_L \end{cases} \tag{91}$$

Antes de continuar, conviene recordar que la tensión V_L debe mantenerse constante, de modo que V_A y V_G puedan usarse como señales de un único raíl, e I_o pueda sensarse como la salida de la sinapsis. Esto va a conseguirse en la práctica

mediante un transportador de corriente [Smit68], que genera una referencia virtual en un nudo de baja impedancia, y reproduce la intensidad que es absorbida en ese nudo donde haga falta.

Volviendo a la Ec. 90, nótese que el segundo término en el miembro derecho de la ecuación no depende de V_G, por lo que $\textcircled{G}$ puede usarse para soportar la variable de estado de la celda. el único inconveniente es que V_G necesita ser siempre positivo para que el transistor opere por encima del umbral. Así que vamos a descomponer V_G en una señal de referencia y una señal correspondiente al estado de la celda superpuesta:

$$V_G \equiv V_X = V_{x_0} + V_x \tag{92}$$

y del mismo modo consideraremos que V_L, es la referencia para la señal de pesos V_{w_0}, y que V_A contiene la referencia y la señal particular:

$$V_A \equiv V_W = V_{w_0} + V_w \tag{93}$$

así que la Ec. 90 puede reescribirse como:

$$I_o = -\beta_p V_w V_x - \beta_p V_w \left(V_{x_0} + \left| \hat{V}_{T_p} \right| - V_{w_0} - \frac{V_w}{2} \right) \tag{94}$$

que es un multiplicador de cuatro cuadrantes con un término de *offset* invariante en el tiempo, al menos durante la evolución de la red, que no depende de la variable de estado:

$$I_o = -\beta_p V_w V_x + I_{\text{offset}}(V_w) \tag{95}$$

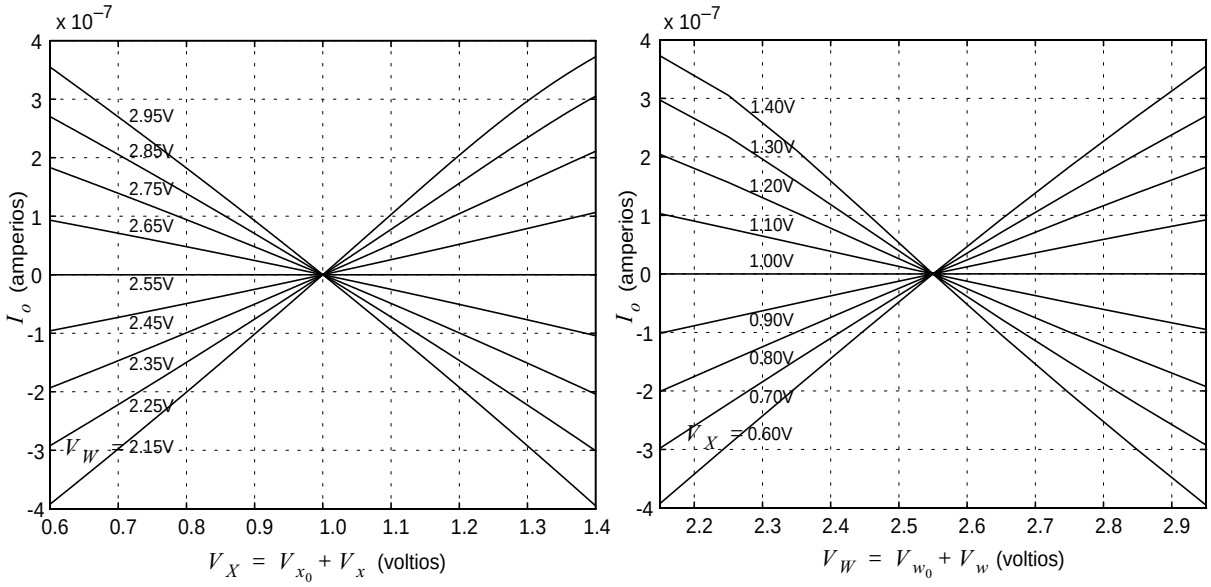


FIGURA 55. Salida del multiplicador (sin el *offset*) frente a V_X y V_W.

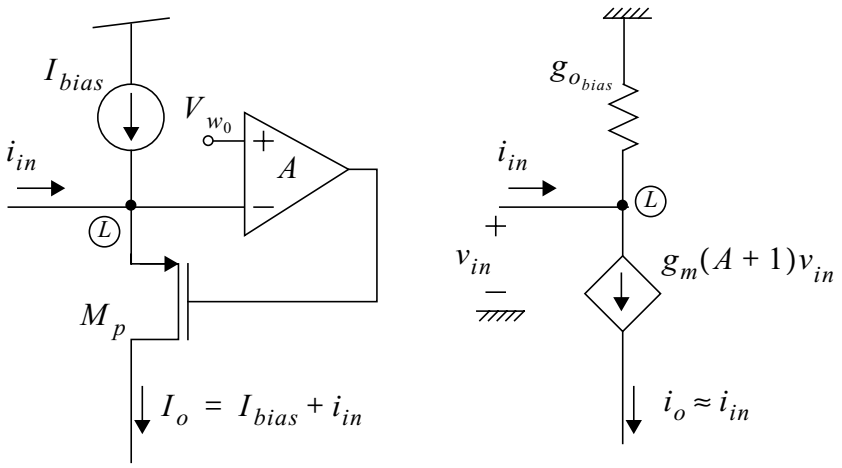


FIGURA 56. Realización del transportador de corriente y modelo en pequeña señal.

y este *offset* puede ser eliminado mediante una memoria de calibración. La simulación del multiplicador, usando Hspice v. 99.2 y un modelo del MOS de nivel 49 en una tecnología CMOS estándar de $0.5\mu m$ puede verse en la Fig. 55. La selección de los rangos de señal apropiados se explica en el Apéndice 5. En esta ocasión, para una corriente máxima por sinapsis de $1.4\mu A$, las dimensiones del transistor serán de $2\mu m / 25.9\mu m$.

Como hemos mencionado, es necesario que la tensión del nudo (L) se mantenga constante, para ello vamos a utilizar el transportador de corriente mostrado en la Fig. 56. Cualquier diferencia entre esta tensión y la referencia es realimentada negativamente, haciendo que el nudo (L) quede fijado a la tensión V_{w_0}. La impedancia de entrada de este bloque es bajísima, si consideramos que $g_{o_{bias}} \ll g_m(A+1)$, entonces:

$$Z_{in} = \frac{v_{in}}{i_{in}} = \frac{1}{g_m(A+1)} \tag{96}$$

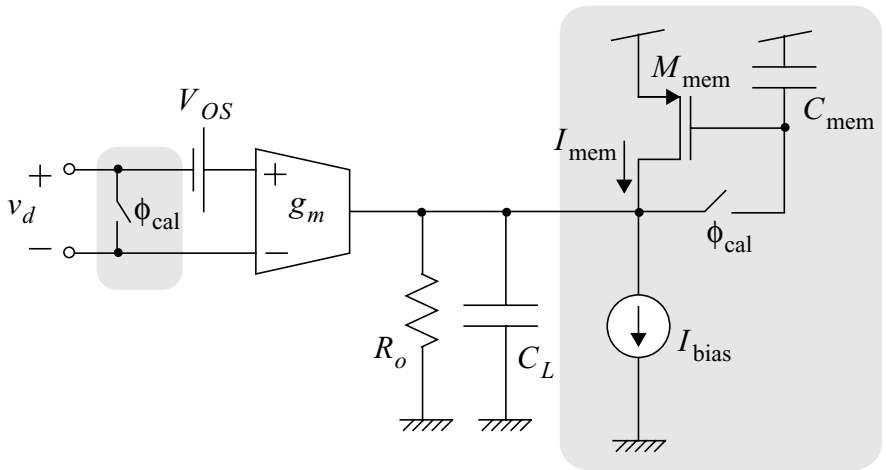


FIGURA 57. Mecanismo de calibración del offset para lo OTA críticos.

En esta realización es esencial reducir al máximo el offset del amplificador, que puede realizarse mediante un OTA simple, puesto que introduciría un error del mismo orden en la referencia virtual implementada en Ⓛ. Así que hemos usado el mecanismo de calibración representado en la Fig. 57, cuya operación se explica en el Apéndice 5.

Memoria de corriente S^3I

Todas las contribuciones de los vecinos, realizadas en corriente, acarrean un offset independiente del estado de las celdas que las generan. Este offset de corriente puede calcularse previamente para calcularlo durante la evolución de la red. Para ello utilizaremos una memoria de corriente. Así, antes de la operación de la CNN, todas las sinapsis, con los pesos correspondientes, se *resetean* con $V_x = 0$, de modo que su salida es exclusivamente el offset de corriente, y este se almacena en una memoria de corriente de gran precisión. Veamos, para las aplicaciones que vamos a programar en el chip, las suma las intensidades que constituyen las contribuciones de los vecinos va a estar entre los $18\mu A$ y los $46\mu A$. Por otro lado, el rango total de intensidad de una sinapsis es de $1\mu A$. Si la precisión del bloque multiplicador debe ser de 8b, entonces $(1/2)$LSB será de $2nA$. Para que la memoria de corriente sea capaz de distinguir $2nA$ de entre los $46\mu A$, tendrá que tener una resolución equivalente de $14.5b$. Por este motivo se utiliza una memoria del tipo S^3I [Toum93] como la que se muestra en la Fig. 58.Esta memoria está compuesta por tres etapas, cada una de ellas conteniendo un transistor que actúa como transconductor, un condensador y una llave de paso. En una primera etapa, la corriente que se pretende memorizar, I_B, se reparte entre los tres transconductores M_1, M_2 y M_3, ya que ϕ_1, ϕ_2 y ϕ_3 están en ON. Cerrando sucesivamente las llaves, llegamos a que el error (ver Apéndice 5) en la corriente final, I_F, sólo depende del error en la última etapa:

$$I_F = I_B - g_{m_3}\Delta V_3 \tag{97}$$

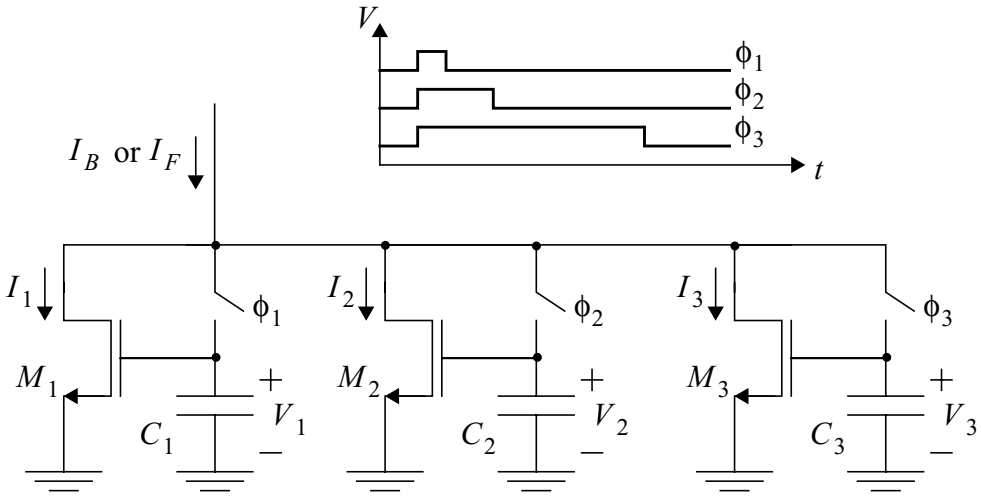


FIGURA 58. Esquemático de una memoria de corriente S^3I y señales de control.

Si la memoria S^3I se diseña de modo que la primera etapa almacena los bits más significativos de I_B, y la última los menos significativos, el error cometido al muestrear la corriente puede hacerse muy pequeño. Si la resolución que debe tener la memoria es de N bits, haciendo el reparto de corriente que hemos planteado (ver Apéndice 5) llegamos a que la transconductancia del transistor M_3 y el error en la tensión almacenada, que dependerá del tamaño del condensador y de la llave, están relacionados a través de:

$$\Delta V_3 \leq \sqrt{\frac{I_B}{\beta_3}} \cdot 2^{-\left(\frac{N_{\text{eff}}}{2}+\frac{N}{2}+2\right)} \text{ donde } N_{\text{eff}} = \log_2\left(\sum_{k=\frac{2N}{3}+1}^{N} 2^{N-k}\right) \quad (98)$$

Una vez que se ha elegido β_3 convenientemente, β_1 y β_2 salen de:

$$\beta_1 = \left(\frac{\beta_3}{2^{N_{\text{eff}}}}\right)\sum_{k=1}^{\frac{N}{3}} 2^{N-k} \qquad \beta_2 = \left(\frac{\beta_3}{2^{N_{\text{eff}}}}\right)\sum_{k=\frac{N}{3}+1}^{\frac{2N}{3}} 2^{N-k} \quad (99)$$

Aunque puede parecer que añadiendo más etapas podemos conseguir un nivel de precisión ilimitado, hay que tener en cuenta que a medida que vamos aumentando el orden de la memoria, las corrientes que tienen que ser sensadas por las últimas etapas son cada vez más pequeñas, del mismo orden que las fugas de las primeras etapas, lo que impide llegar a un estado estacionario para la cancelación de errores.

Escalado de la constante de tiempo

En la capa de CNN con constante de tiempo fija, la salida del transportador de corriente más una copia, con signo contrario, del error de corriente memorizado, son inyectadas en el condensador de estado, que en este chip está formado por las puertas de las sinapsis conectadas a este nudo. La constante de tiempo de esta capa consiste en:

$$\tau = C_c / G_c \quad (100)$$

que es el cociente entre la capacidad total de todas las sinapsis conectadas a este nudo, C_c, y la transconductancia de normalización, G_c, que se emplea para calcular las tensiones de los pesos y que tiene que ver con las dimensiones de la sinapsis y la potencia consumida por esta. Si nos planteamos realizar un escalado de la constante de tiempo, por un lado, no podemos tocar la constante G_c ya que perderíamos las referencias de los pesos y podemos perder el rango dinámico, ya que las señales de los pesos están restringidas a un determinado intervalo de tensiones para las que está asegurado el funcionamiento de la sinapsis y del transportador de corriente. Por otro lado, no podemos tener un capacidad en el nudo de la variable de estado por debajo de C_c, de modo que sólo podemos pensar en

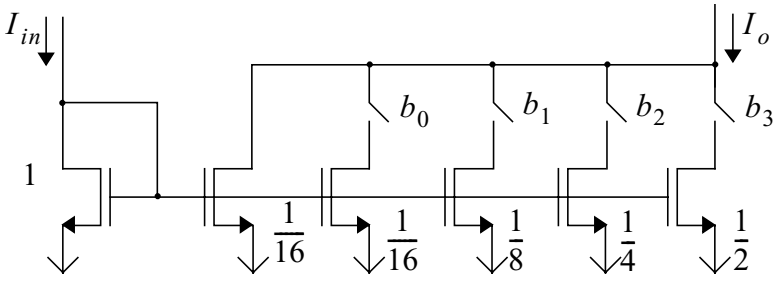


FIGURA 59. Espejo de corriente programable binario (4b).

una constante de tiempo mayor, incrementando progresivamente el condensador de estado. El inconveniente principal es que para tener un rango de constantes de tiempo como el que se pretende, $1{:}16$, necesitamos un área demasiado grande. Una alternativa viable puede ser el escalado de la corriente que llega al condensador de estado. Si nos fijamos en la ecuación que rige la evolución de la CNN, nos encontramos con una suma de intensidades que son inyectadas en el condensador de estado:

$$\tau \frac{dx_{ij}(t)}{dt} = \sum_k I_k \tag{101}$$

si escalamos dicha suma, el efecto es el de un escalado de la constante de tiempo. Este escalado puede ser continuo o discreto. Puesto que el escalado discreto, mediante un espejo de corriente pesado binariamente, presenta ventajas evidentes en cuanto a la reproducción del comportamiento nominal a lo largo y ancho del chip (ver Apéndice 5), y puesto que no necesitamos un control muy preciso del valor de la constante escalada (con 4 bits es suficiente) hemos optado por esta solución. Así, el espejo de corriente programable binario de 4b de la Fig. 59 produce una salida que está relacionada con la corriente de entrada I_{in} a través de:

$$I_o = (1 + b_0 + 2b_1 + 4b_2 + 8b_3)\frac{I_{in}}{16} \tag{102}$$

donde b_0, b_1, b_2 y b_3 son los valores binarios de los bits de control.

Sin embargo, con esto surge un nuevo problema, si la corriente de entrada puede escalares hasta sólo $1/16$ de su valor original, la memoria de corriente debería mantener la precisión operando sobre las corrientes más grandes y sobre las pequeñas, cosa difícil de asegurar mediante el diseño. Para evitar esto, podemos hacer que la memoria S^3I opere sobre la versión sin escalar de la corriente, y luego, escalamos la contribución y el offset memorizado juntos, antes de ser dirigidos al condensador de estado. El único problema está ahora en que si realizamos el escalado posteriormente a la memorización de los errores, el offset introducido por el bloque de escalado se suma a la señal, de modo que disminuye la precisión. Nuestra propuesta finalmente consiste en colocar el bloque de escalado (espejo programable) entre el transportador de corriente y la memoria de corriente (Fig. 60) de modo que durante la calibración, cualquier error que no dependa de la variable de estado, V_x, incluido el offset del bloque de escalado, será memorizado para cancelarse posteriormente (más detalles en el Apéndice 5).

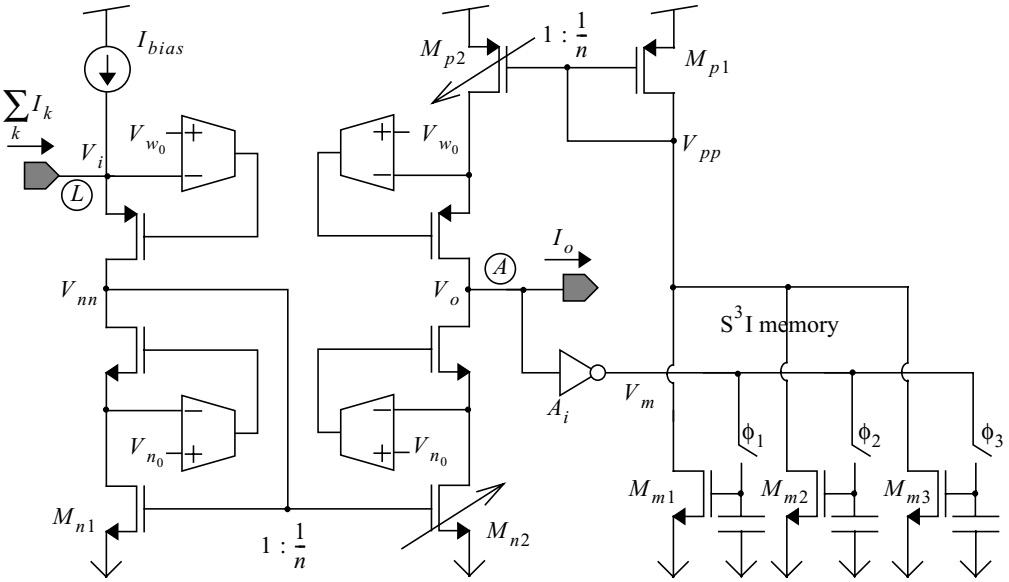


FIGURA 60. Bloque de entrada a la capa de constante de tiempo escalable.

Limitador de la variable de estado

El limitador de tensiones empleado para realizar la saturación de la variable de estado, en el modelo FSR de CNN, es el mismo que se utiliza en [Liña98]. Como puede verse en la Fig. 61(a), consiste en un limitador *suave*, compuesto por M_n y M_p, y un limitador *duro* formado por comparadores de tensión. El limitador blando tiene como misión evitar excursiones del nudo de entrada cuando este queda aislado por la acción del limitador duro. La implementación de los comparadores del limitador duro (Fig. 61(c)) está reportada en [Rodr95]. La impedancia de entrada de este comparador se comporta en parte como resistiva, evitando que la tensión del nudo de entrada realice desplazamientos importantes, y en parte como capacitiva, lo que incide en una gran precisión, dependiendo de la

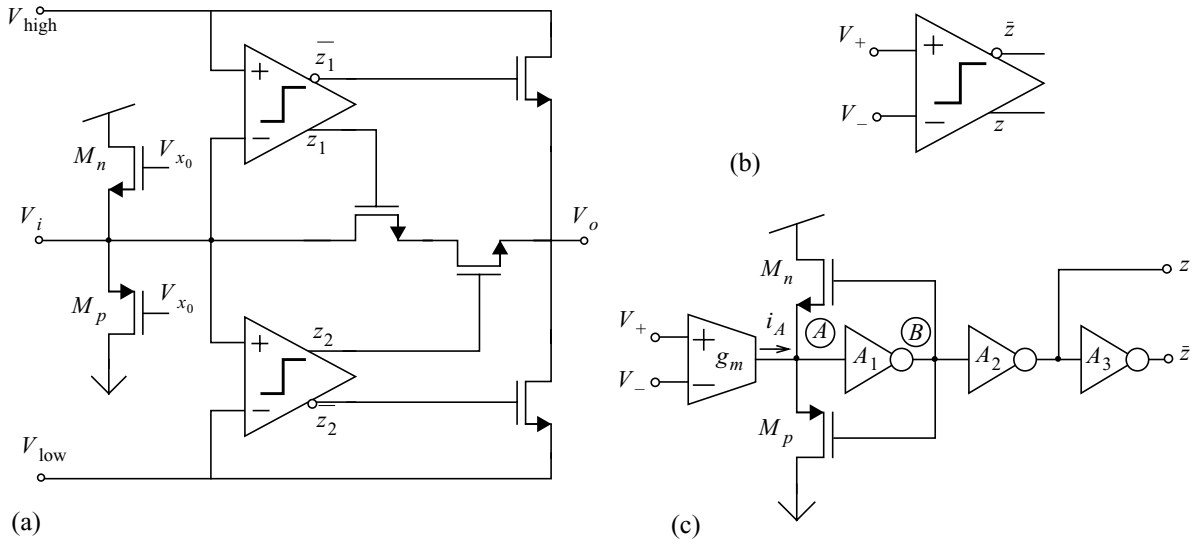


FIGURA 61. (a) Limitador de la variable de estado y comparador de tensiones: (b) simbolo y (c) esquemático.

zona de operación. En el punto equiescente, $V_+ - V_- = 0$, tendremos, si no existe un offset, que $i_A = 0$ y que M_n y M_p no conducen corriente. Cualquier pequeña variación en la tensión diferencial de entrada es transformada en una pequeña corriente i_A y detectada por la capacidad del nudo $\textcircled{A}$, teniéndose la máxima resolución. Si la corriente es grande, positiva o negativa, un lazo de realimentación mantiene la tensión V_A cerca del valor del punto equiescente. La impedancia de entrada se convierte en resistiva, ya que M_p o M_n conducen.

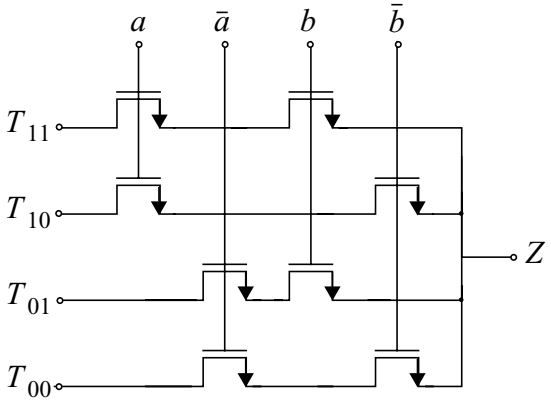
Este bloque es el último de los circuitos contenidos en los núcleos de CNN de la celda básica. Aparte de ellos, dentro del procesador elemental, encontramos la unidad lógica local (LLU) y las memorias analógicas y lógicas (LAMs y LLMs).

Unidad lógica y memorias locales

La LLU es una puerta lógica de dos entradas totalmente programable realizada mediante un esquema de lógica de transistor de paso, propuesto y validado en [Espe96a]. En esta unidad, las salidas están programadas por el usuario. En la Fig. 62 podemos ver el esquemático de esta realización, así como la tabla de verdad y el mapa de Karnaugh de la función combinacional. En cuanto a las LAMs, han sido realizadas mediante las mismas técnicas descritas para el chip de aRAM. La única diferencia es que aquí hemos usado transistores MOS cortocircuitados a modo de condensadores. Las memorias lógicas locales (LLMs) son celdas de memoria RAM estática convencionales.

Floorplan del chip prototipo y circuitería periférica

El chip propuesto consiste en un arreglo de procesadores analógicos en paralelo de 32×32 celdas, idénticas y del tipo descrito antes (Fig. 63). Estos procesadores están rodeados por un anillo de circuitos que implementan las condiciones de contorno de la CNN. Aparte de esto, existe una interface de E/S, una unidad de control y temporización y una unidad de programación. La interface de E/S (Fig. 64)consiste en un multiplexor analógico serializador-deserializador que conecta el canal analógico en



(a)

Tabla de verdad

a	b	z
0	0	T_{00}
0	1	T_{01}
1	0	T_{10}
1	1	T_{11}

(b)

Mapa de Karnaugh

b \ a	0	1
0	T_{00}	T_{10}
1	T_{01}	T_{11}

(c)

FIGURA 62. (a) Esquemático de la LLU, (b) tabla de verdad y (c) mapa de Karnaugh.

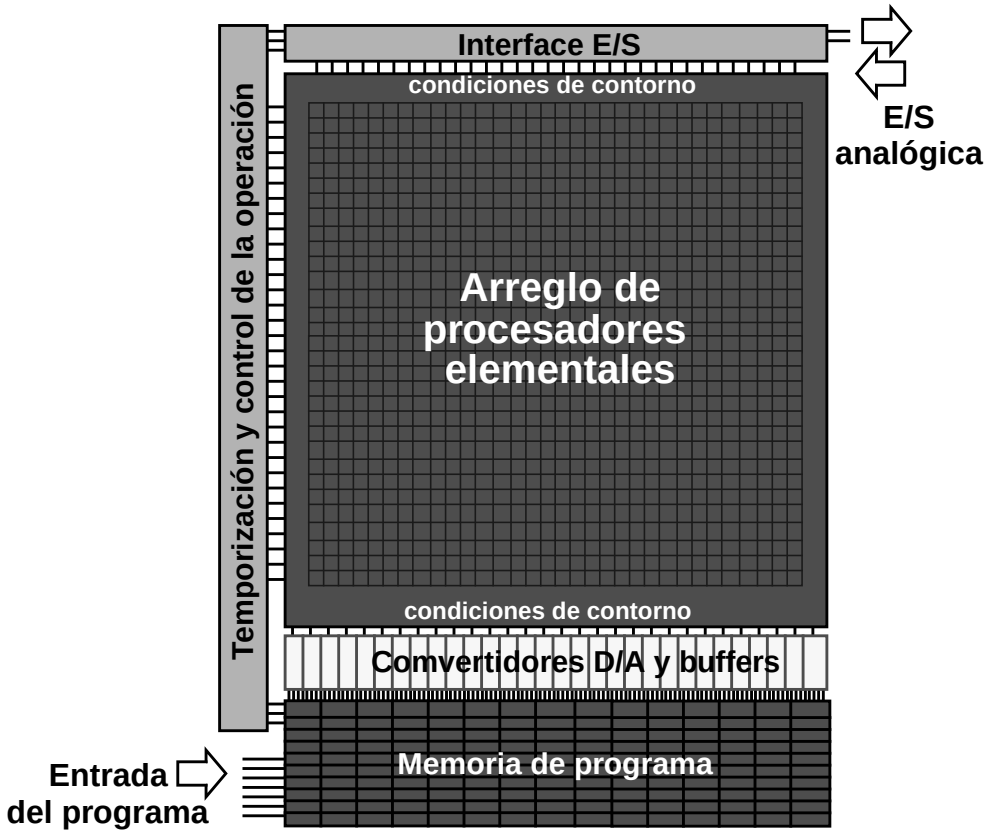


FIGURA 63. *Floorplan* del chip prototipo CACE1K..

serie de E/S del chip al bus de 32 líneas de E/S del *array* de procesadores. Esto se realiza mediante una batería de bloques S/H y los correspondientes decodificadores de fila de columna, controlados por la unidad de temporización. Al adquirir una imagen, cuando el bit de instrucción **ENRW** está en alto, el canal de E/S conecta el pad de entrada del chip a los nudos de entrada de los 32 S/H. Entonces, 32 muestras de la entrada son capturadas a 10Ms/s. Después, las siguientes 32 muestras son capturadas por la segunda fila de S/H mientras la primera descarga sus contenidos en la fila correspondiente del *array* de procesadores. Al hacerlo en paralelo,

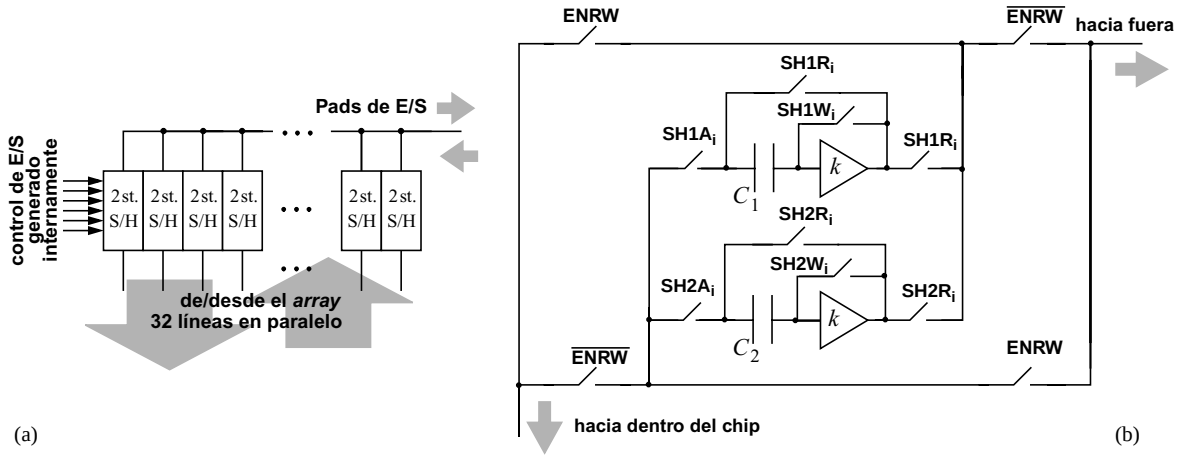


FIGURA 64. (a) Interface E/S y (b)circuito de muestra y retención de dos etapas.

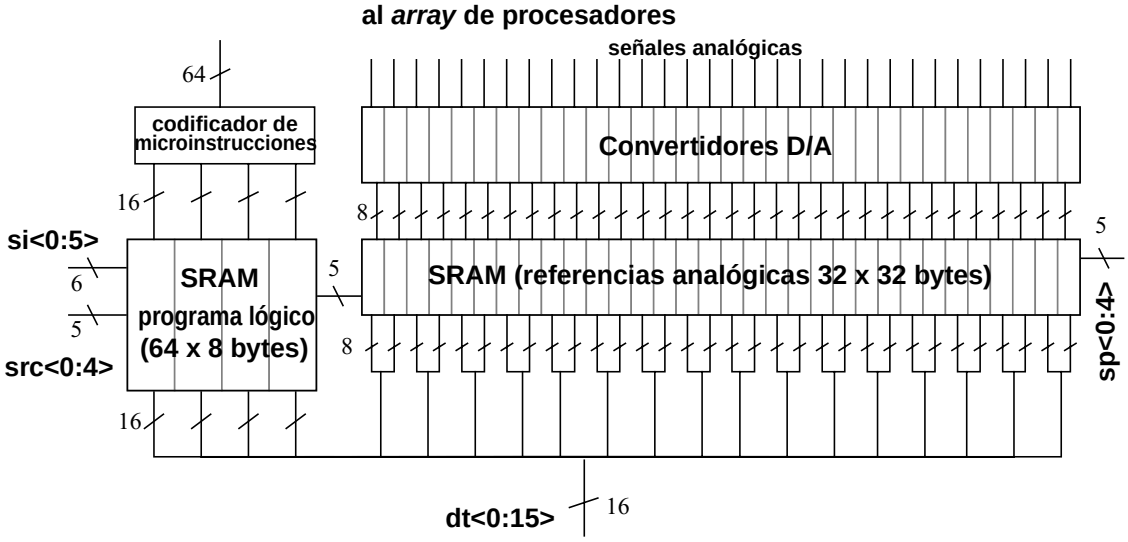


FIGURA 65. Block diagram of the program memory and program control

disponen de $3.2\mu s$. Una vez que se han adquirido la 32 segundas muestras, la segunda fila de S/H pasa a descargarlas hacia el array y la primera fila de S/H se dispone a adquirir las siguiente 32 muestras. Así hasta completar los 32×32 pixels que componen la imagen completa. Para descargar la salida de la red, el proceso se invierte. Las muestras se adquieren el paralelo, a través de las 32 líneas de E/S del *array*, y se descargan hacia el *pad* de salida de una en una.

La unidad de temporización está compuesta por un reloj/contador interno y un conjunto de máquinas de estados finitos que generan las señales internas para carga y descarga de imágenes, del programa etc. El control de la operación se realiza mediante un programa almacenado en la memoria de programa. A un ritmo marcado por el *host* digital, las instrucciones almacenadas son traducidas en micro-instrucciones que el arreglo de procesadores en paralelo ejecuta. Todo esto constituye la llamada GAPU, la unidad global de programa analógico y lógico, en la estructura de la CNNUM.

Por último, la memoria de programa está compuesta por 1kB de RAM estática (Fig. 65) que contiene el programa analógico (APR, *analog program register*), 0.5kB para el programa lógico (LPR, *logic program register*) y la configuración de la celda (SCR, *switch configuration register*), y alguna señal para la interface de E/S. Las señales **dt<0:15>** constituyen el bus de datos digitales. Las direcciones, vienen indicadas por la dirección del bloque de memoria, señales **src<0:4>**, y la selección de palabra, señales **si<0:5>** para el LPR o señales **sp<0:4>** para el APR. En el modo de ejecución, las instrucciones lógicas son transmitidas en forma de micro-instrucciones al array de procesadores, pero las instrucciones analógicas, tensiones de los pesos y de referencia, almacenadas en forma binaria, necesitan convertirse a tensiones analógicas para ser transmitidas a la matriz de celdas. Para ello disponemos de un banco de convertidores D/A. Distribuir señales analógicas a lo largo y ancho de un chip de casi 1cm^2 no es una tarea fácil. aparte de los problemas derivados de la interferencia de señales ruidosas, tenemos que contar con la caída de tensión en las líneas de metal que soportan una importante intensidad de

corriente. Para resolver este problema hemos construido un modelo que nos ha permitido hacer una estimación del error y disponer de una estrategia de *buffereado* que nos permitirá mantener los niveles de precisión requeridos (véase Apéndice 5).

5. 3. Datos de chip prototipo y simulaciones

Datos del chip CACE1K

Siguiendo las directrices expresadas en la sección anterior, hemos diseñado un chip prototipo que ha sido fabricado en una tecnología estándar CMOS de $0.5\mu m$ de resolución, con una sola capa de polisilicio y tres capas de metal. En la Fig. 66 podemos ver una microfotografía del prototipo al que hemos llamado CACE1K. Contiene un *array* central de 32×32 celdas, cada una con dos núcleos de CNN acoplados con dinámica programable, LLMs y LAMs y una LLU. Además del array, la interface de E/S, la circuitería de control y temporización y la memoria de programa. El sistema completo ocupa 9.27×8.45 mm^2, incluyendo el anillo de *pads*. Sin los *pads*, el área total de 8.77×7.94 mm^2. El arreglo de celdas de CNN ocupa 5.98×5.83 mm^2, lo que resulta en una densidad de 29.37celdas/mm^2. Para manejar estos datos con cuidado, hay que

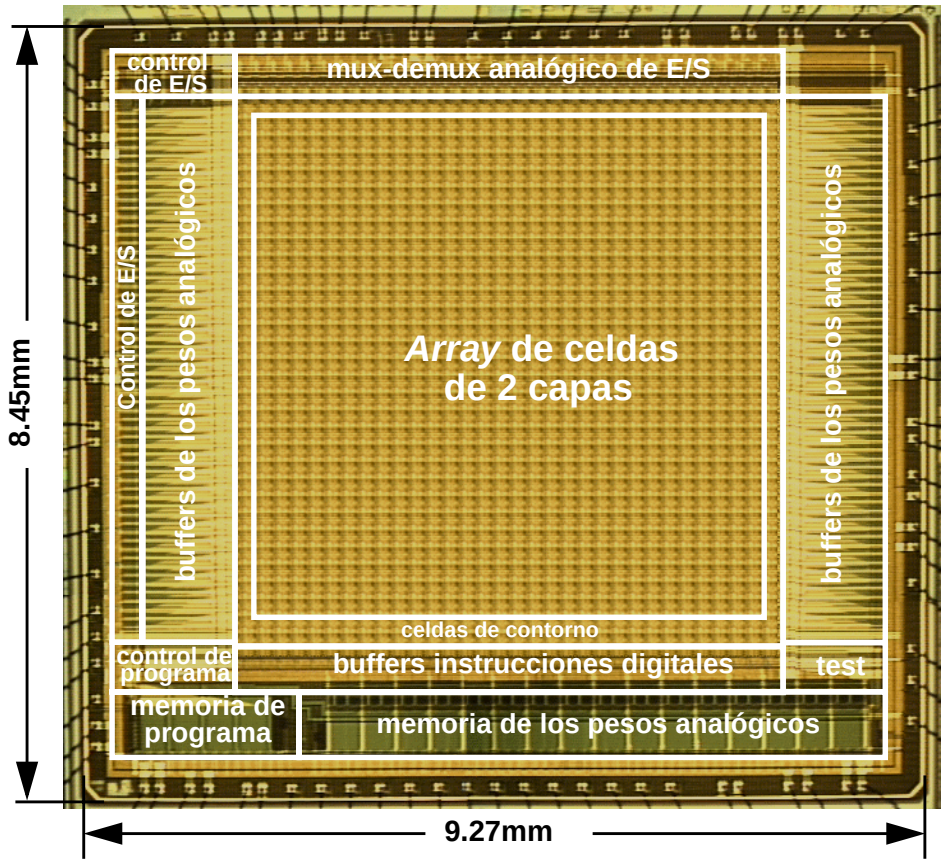


FIGURA 66. Microfotografía del chip prototipo.

tener en cuenta que el área ocupada por el array se escala linealmente con el número de celdas, mientras que la circuitería externa tiene a ocupar cada vez una fracción más pequeña del chip a medida que crece el número de procesadores elementales. Este chip ha sido encapsulado en un PGA-100. Los resultados que se esperan de los tests son $8b$ de precisión en los pesos, $7\text{-}8b$ de resolución en las imágenes, una tasa de E/S de datos analógicos de $10Ms/s$, y una constante de tiempo para la dinámica de la CNN por debajo de los $100ns$. Se están realizando en este momento diversos test que confirmen las características prescritas.

Resultados de la simulación

Para hacernos una idea de la manera en que este circuito desarrolla el procesamiento de la entrada, podemos examinar los resultados de la simulación. En primer lugar, podemos ver la evolución de las variables de estado y de la salida de una sola celda del chip, sometida a todo el proceso de carga de imágenes, calibración de los circuitos, inicialización de las capas de CNN, evolución de las capas y, finalmente, diferentes operaciones lógicas sobre los resultados de la evolución de las capas, que son enviadas a la salida del chip a través del bus de E/S de la columna correspondiente. Todo esto puede verse en la Fig. 67.

En la siguiente simulación (Fig. 68) vemos como evolucionan las variables de estado de 8 celdas de CNN en una fila, programadas para exhibir la propagación de una onda que empieza con un único pixel en valor alto en una de las capas (variable V_{x_1} en la celda C_{14}). Como vemos, esta señal se propaga a la otra capa y se expande hacia los extremos de la línea de celdas.

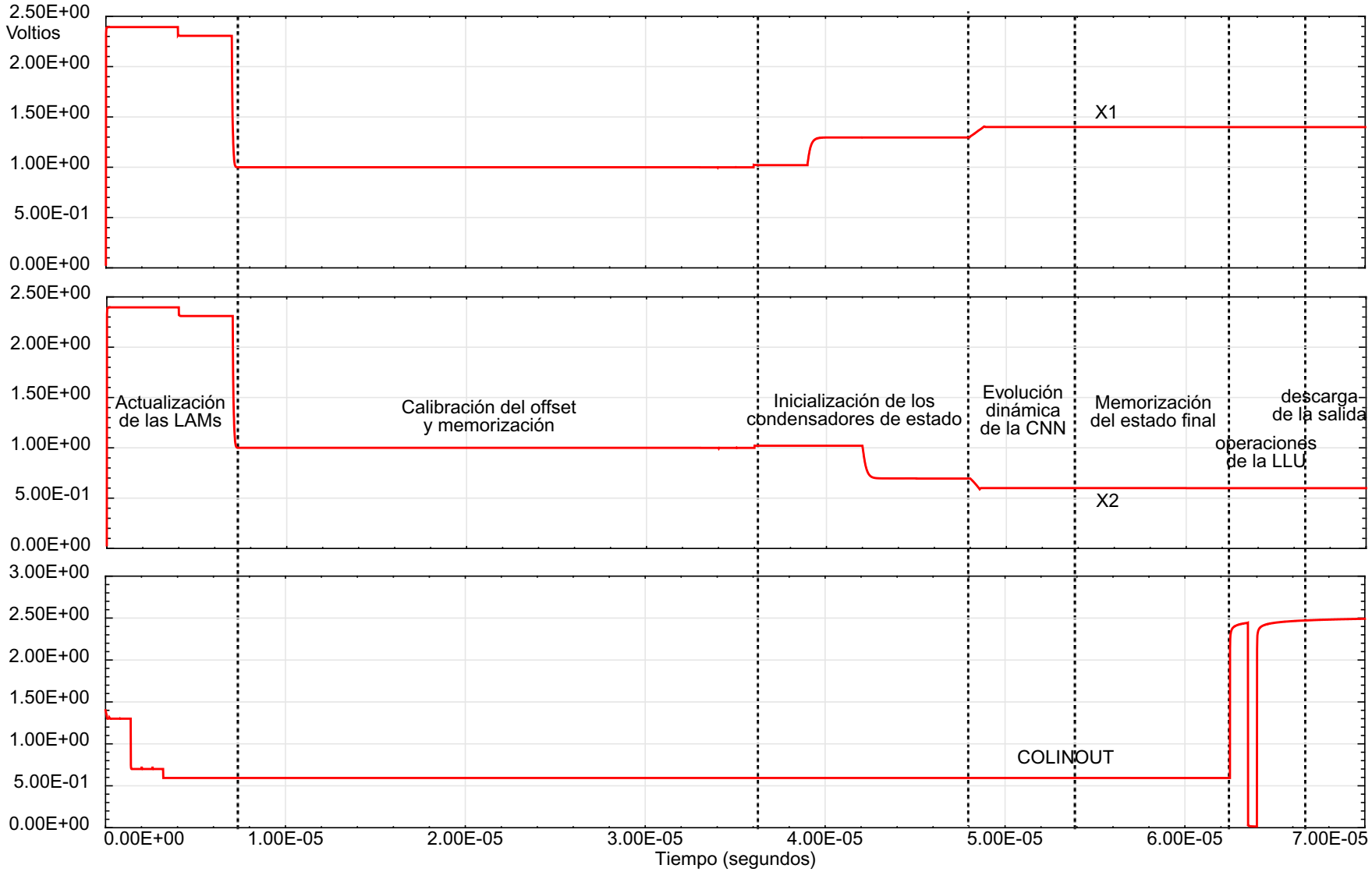


FIGURA 67. Evolución de las variables de estado y del nudop de E/S de una celda.

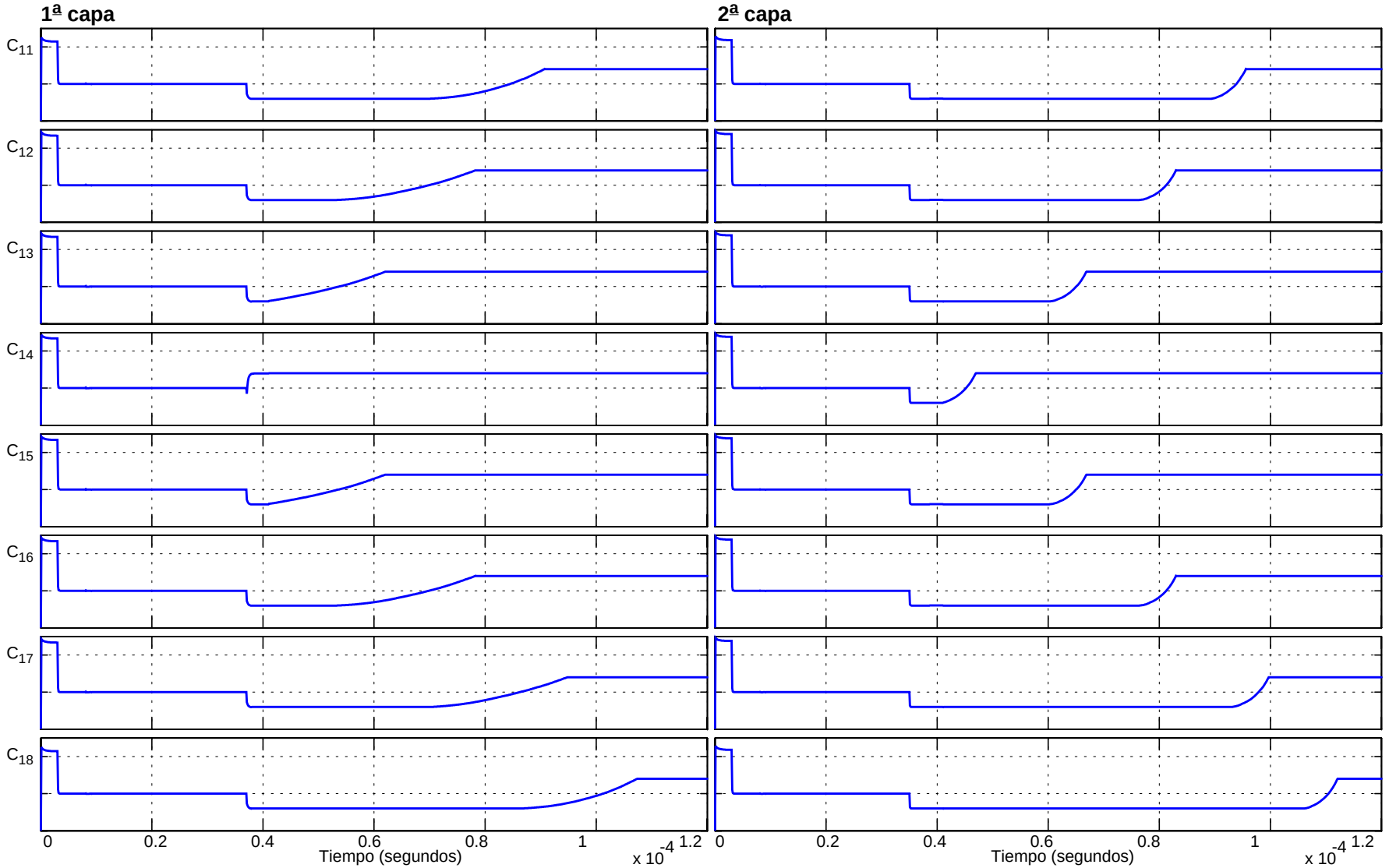


FIGURA 68. Evolución de las variables de estado en una fila de 8 celdas.

Apéndices

1 An Introduction to Real-Time Image and Video Signal Processing with CNNs

Real-time image and video signal processing is a highly demanding computational task that can easily require operating speeds beyond the gigaOPS (10^9 operations per second). Practical image sizes and frame rates for industrial and even domestic use represent a quite significant flow of information. For instance, conventional TV systems deal with picture frames of about 644×483 pixels at around 30 frames per second [Poyn96]. It represents a pixel rate of $9.3\,\mathrm{Mpel/s}$, without considering the necessary scanning overhead required for blanking and synchronization in the most common display systems. A simple mask operation, e. g. for averaging or linear spatial filtering, in which a set of weights is applied to a 3×3-pixel neighborhood of each image dot, requires 9 multiplications and 8 additions per pixel [Gonz87], what requires a processing speed of $0.16\,\mathrm{GOPS}$ if performed real-time. The same operation in HDTV (High Definition TV), e. g. 1920×1080 pixels per frame at 30 fps, means $1.06\,\mathrm{GOPS}$. Complex image processing algorithms developed for applications like stereoscopic artificial vision, image fusion or motion picture compression, for instance, involve a lot more arithmetic and logic operations per pixel, thus largely exceeding these figures and approaching the teraOPS range (tera $= 10^{12}$).

VLSI implementation of such an enormous computing power in the digital world, where parallelization represents a prohibitive duplication of hardware, requires of the adoption of some architectural strategies leading to either dedicated or programmable approaches. On one side, dedicated architectures exploit adaptation of the VLSI hardware to the realization of a specific algorithm. In the case of low-level image processing tasks, we are talking about simple and repetitive operations of an inherently regular and parallel nature. A dedicated architecture results in a high efficiency and the minimization of control circuitry and power consumption, at the expense of a lack of flexibility. On the contrary, programmable architectures, based mostly in software control, are able to handle computational tasks of largely varying complexity, but result significantly inadequate for

the execution of low-level computation-intensive processing tasks, which represent nearly 80% of the work in most of the image processing applications [Pirs98]. Flexibility and efficiency must be traded-off, together with time-to-market and design effort constraints for industrial development.

An alternative to conventional digital approaches is analog VLSI signal processing. Analog circuits result in a more compact and efficient realization of the functionalities involved in many image processing tasks. Their major disadvantage is the reduced accuracy achievable, especially in the case of mixed-signal chips in which analog and digital circuitry coexist on the same substrate. However, an 8-bit equivalent resolution is enough for image processing, as established by widely accepted video coding standards [CCIR90a]. Moreover, early vision tasks realized by biologically inspired artificial vision chips can be accomplished with an equivalent resolution of 6-7 bits [Koch95], thus reducing accuracy requirements and extending the area of application of analog VLSI circuits. Concerning the flexibility vs. efficiency trade-off, a compromise can be made for achieving enough programmability with an efficient and compact analog VLSI implementation. In this context, the Cellular Neural Network Universal Machine (CNNUM) was conceived [Rosk93a]. The CNNUM is an algorithmically programmable analog array computer with supercomputer power on a chip. It can be seen as a specialized analog array processor adapted for parallel and massive operation onto a large amount of analog data, with enough flexibility to implement highly complex algorithms by means of an encoded analogic —analog and logic— program. It has been demonstrated to be universal in the Turing sense [Crou96] and currently available analog VLSI implementations [Liña98] achieve a peak operation speed of 0.4TOPS. An astonishing figure when compared to the 0.9GIPS[i] attainable by last generation multiprocessor DSP's [TI99].

The main objectives of this chapter are, first, to assess system requirements for the implementation of real-time image and video signal processing, attending to speed and accuracy, by the revision of the most common analog and digital TV, computer graphics and video formats, and, secondly, to introduce the principles of CNN-based processing by a detailed revision of the CNN Universal Machine concept and its potential for high-speed massively parallel analog signal processing on a single chip. This introduction is the starting point for a feasibility study of the implementation of real-time video processing system on a chip with high-end performance at reduced fabrication costs. Design methodology, architectural considerations, analog VLSI implementation proposals and experimental results obtained from the test of some integrated prototypes will be disclosed in the following chapters. In the end, this approach represents a practicable solution for real-time video processing that overcomes existing technology limitations in this field, which, by the way, displays one of the wider areas of application for both analog and digital VLSI signal processing.

i. Here, the operation speed of a digital processor is measured in IPS —instructions per second— which has been considered comparable to the analog operations per second (OPS) employed before, although it can result in a pessimistic estimation, given that a simple multiplication in the analog world can involve more than one micro-instruction in a digital processor.

1. 1. Video signal formats and standards

1. 1. 1. Basic definitions in analog and digital video formats

The representation of real-life 3-D scenes in the form of 2-D images is based on the projection of light emitted and reflected by objects into a light-sensitive surface. In human vision, this surface is the retina. It is composed of photo-receptors that transduce visual stimuli into neuronal impulses that can be processed and handled by the central nervous system [Hube88]. In photographic cameras, light beams are projected onto a light-sensitive film. The focusing system operates much in the same way as the human eye. This film is developed and then printed into photographic paper by means of physical and chemical processes. Video cameras, in contrast, contain an array of photo-receptors at the focal-plane, which realize photo-transduction by converting brightness and colour information of each picture element (pixel) into electrical signals of a specific format. Visual information is coded in this way into a set of signals that can be handled by electronic equipment, and thus allowing for a wide variety of data processing operations like image enhancement and restoration, editing and modification, etc. These signals, that will be referred as video signals, can be stored in a tape, processed on- and off-line and displayed by appropriate devices (monitors, LCD panels, etc.). Thus, let us identify the main elements and characteristics of image capture and visualization processes and the signals involved, in order to establish afterwards the specifications for a real-time image processing electronic system.

Image model

The term image refers to a 2-D light-intensity function. Although light incident in real-life objects can have components on a wide frequency range, human vision is restricted to wavelengths in the range of 400nm to 700nm, approximately. Colour vision is based on the absorption and reflection of components of different frequencies by the part of the objects being illuminated. Colour encoding and representation will be the subject of one of the following sections, therefore we will restrict ourselves to the consideration of monochrome images, in which each pixel, of coordinates (i, j) in the focal-plane, or in the screen of the display (where i represents the position in the vertical scan direction and j in the horizontal direction), is associated to a picture brightness value in a gray scale. This value, E'_Y, is commonly referred as video luminance. A monochrome image, I, is therefore defined by:

$$I \equiv E'_Y(i, j) \tag{1.1}$$

which is assumed to be normalized to unity. The maximum value, 1, corresponds to a white pixel, while the minimum, 0, represents a black pixel

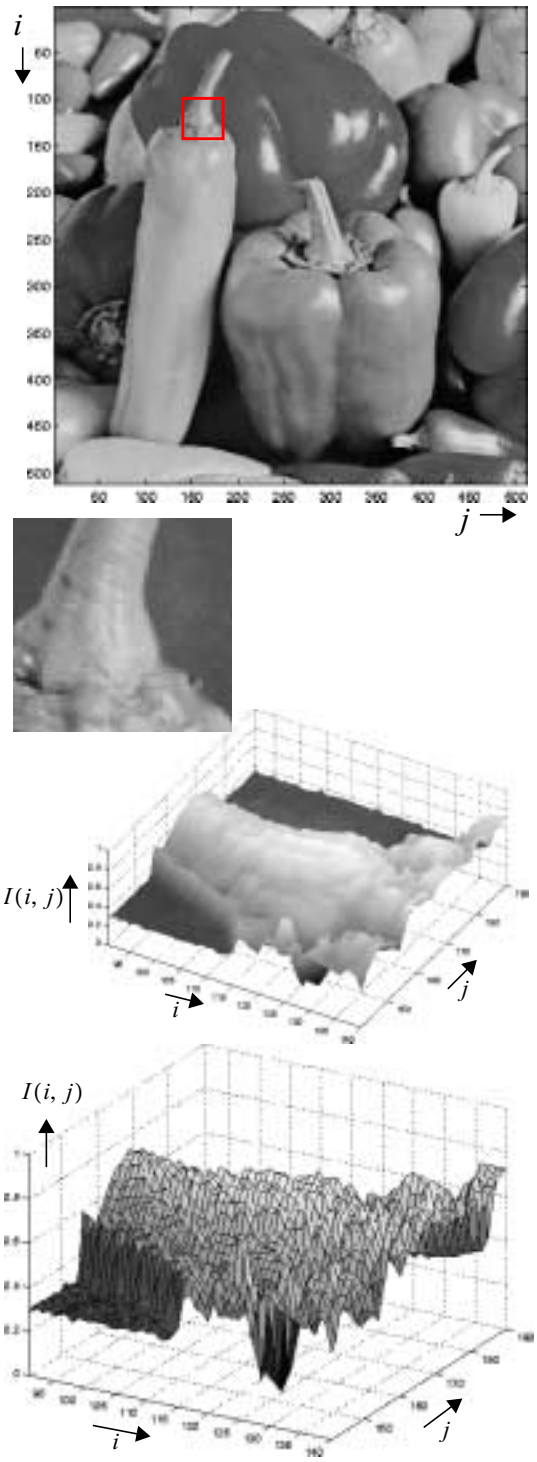


FIGURE 1.1. Monochrome image and picture brightness surface plot of the in-line detail.

(see Fig. 1.1). The prime symbol indicates that E'_Y is not a linear transformation of the physical light intensity. A nonlinear transformation, known as gamma correction, is required to convert the image brightness, that is linear in the optical domain, to a video luminance signal, that will be required to be linear in the electrical domain. Derivation of E'_Y from the additive primary colour signals, E'_R, E'_G and E'_B, and the definition of digital signals Y, C_R and C_B will be considered later on.

It is important to mention that a 2-D image (monochrome image) can be considered as a continuous function as long as the observer is not able to discriminate differences between adjacent light intensity values, only noticeable at the quantum level, therefore it is not quantized in principle. But discreteness of the supporting media —due to the finite number and size of photo-receptors in a camera, or image dots in a display device— leads to a discretization of the image plane and, consequently, the intensity function appears as sampled at equally spaced intervals in both scan directions. In addition, image size is naturally limited, therefore, pixel coordinates can only take discrete values from a finite set:

$$(i, j) \in \{(1, 1), \ldots, (1, N_j), (2, 1), \ldots \\ \ldots, (N_i - 1, N_j), \ldots, (N_i, N_j)\} \quad (1.2)$$

where N_i and N_j are the total number of pixels in both scan directions. A 2-D monochrome image can be represented, then, by a $N_i \times N_j$ matrix of brightness values, that is, a batch of N_i lines containing N_j luminance samples each[ii]. The resolution of the image, i. e. the sharpness of the visual representation, is directly proportional to the pixel array size.

It is common to employ square pixels in the capture and visualization of images, what means that the sampling grid has equal horizontal, w_P, and vertical, h_P, sample pitch, but some video

ii. Notice that picture size is commonly expressed as the product of the picture frame base times its height, this is, picture size is conventionally indicated by the dimensions of the transposed image matrix.

systems use sampling grids in which the horizontal and the vertical sample pitch differ ($w_p \neq h_p$). Picture width, w_I,and height, h_I, and pixel dimensions are related through:

$$w_I = N_j \cdot w_p \qquad \text{and} \qquad h_I = N_i \cdot h_p \tag{1.3}$$

Picture frame rate

Motion pictures are based on the visual illusion created by a succession of flashed still pictures, $\{I_k(i, j)\}_{k = 0, \ldots, N}$, captured and displayed at a sufficiently high rate. Now picture brightness is also a function of time, $I(i, j, t)$. If each still picture is flashed at times 0, T_V, $2T_V$, $3T_V$, $\ldots$, the image brightness for the pixel with coordinates (i, j) at time t is given by:

$$I(i, j, t) = I_n(i, j) \qquad \text{where} \qquad n = \text{floor}\left(\frac{t}{T_V}\right) \tag{1.4}$$

and where the function $\text{floor}(\cdot)$ represents the nearest smaller integer value (including zero).

The delay between two consecutively flashed still pictures, T_V, defines the flash rate $f_V = 1/T_V$, also known as frame rate or field rate, depending on the scanning method, that can be progressive of interlaced, as we will see later. f_V is also referred as the vertical frequency, especially in display systems, in which it represents how often the monitor screen is refreshed with a new picture information.

The quality of this motion representation depends on several factors like the illumination of the environment, the viewing angle, etc. Therefore, different flash rates apply for different viewing conditions. For instance, a flash rate of 50Hz or 60Hz is sufficient for watching television in a dim environment, such as a living room. In a bright environment, like in an office, flash rates higher than 70Hz are required for displaying motion. Nevertheless, objective criteria for the evaluation of motion portrayal quality can be derived from the relations between pixel and image sizes and refresh rate. In fact, the larger the vertical frequency the less flickering will be noticed in the screen and therefore, the higher realism is obtained in the representation of motion. However, the higher the picture resolution, the higher the frame rate required for maintaining the same quality in the representation of motion.

Horizontal frequency

As indicated by the vertical frequency, f_V, one complete picture frame (or field) is captured, transmitted or displayed each $1/f_V$ seconds. During this time, the image plane information is scanned line-by-line from left to right and from top to down. This will be a progressive scanning of the frame —we will see later what interlaced scanning is. In these conditions, a line rate, f_H, also known as horizontal frequency, can be defined from f_V and the total number of image lines contained in a picture frame, N_V:

$$f_H = N_V \cdot f_V \tag{1.5}$$

Notice that N_V, that has been referred as the total number of image lines, is usually different from N_i, the number of rows of the picture brightness matrix. The reason is that not all of the N_V lines contain actual image information. Some display devices, like cathodic ray tubes (CRTs), requires of some extra time, let us denote it as $N_{B_V} T_V$, for the picture refresh agent, i. e. the electron gun, to return to the original position before a new frame can be flashed. This can be seen as some extra N_{B_V} blank lines, for synchronization, added to the N_i actual image lines, thus conveying a total of $N_V = N_{B_V} + N_i$ lines. The same applies for the discrepancy between the total number of samples per line and the actual luminance samples per line, i. e. the number of columns of the image matrix, N_j. In this case, being $T_H = 1/f_H$ the line period, $N_{B_H} T_H$ seconds are required for the display device to recover and synchronize with the origin of the next image line. Therefore, N_{B_H} blank image samples added to the N_j actual luminance samples, makes a total number of samples per line of $N_H = N_{B_H} + N_j$.

The format of an analog video luminance scan line is shown in Fig. 1.2. Each line of the image is represented by an analog voltage signal that starts with a line synchronization pulse, in which the voltage goes down to V_{sync}. It is followed by a burst of image samples, with voltage levels between the lowest V_{black} and the highest voltage V_{white}. Then, after $N_{B_{H2}}$ extra blank samples, a new synchronization pulse belonging to the next image line is received. A pixel rate —pixel clock frequency or, simply, sampling frequency— f_S, can be defined as the rate at which luminance samples are taken, transmitted and received. It can be calculated from the line frequency, f_H, and the total number of samples per line, N_H:

$$f_S = N_H \cdot f_H \tag{1.6}$$

Correspondingly, $T_S = 1/f_S$ is the pixel clock period or sampling period. It is worth noting that spatial frequencies associated with the still image information are converted to temporal frequencies by means of picture scanning in video signal transmission and processing. In particular, image features in the horizontal scan direction with a spatial periodicity being a multiple of the pixel width, let us say $k \cdot w_p$, are now represented by time-varying signals with a significant frequency

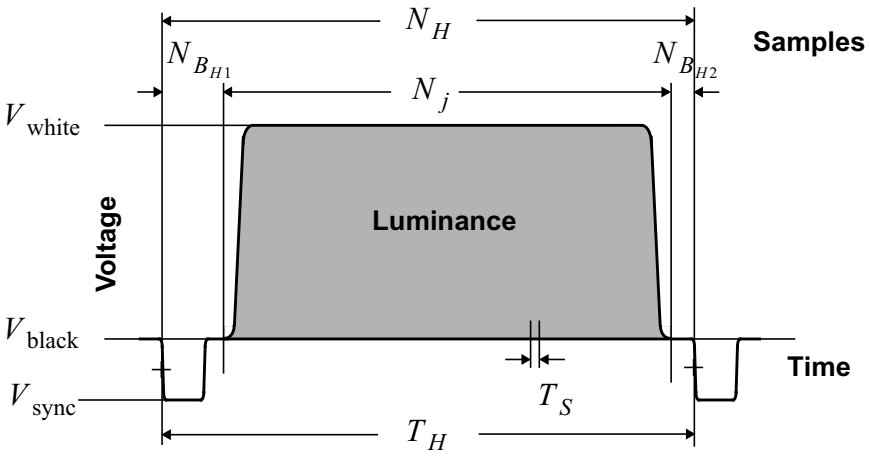


FIGURE 1.2. Scan line waveform of video luminance.

component at f_S/k. Periodic patterns in the vertical scan, with period $k \cdot h_p$, manifest in turn as f_H/k components in the video signal spectrum.

Progressive and interlaced scanning

Picture frames, as we have already seen, are captured and transmitted on a line-by-line basis from the uppermost line of the image to the bottom. When it is done throughout the frame, this is, lines are scanned consecutively, without skipping any of them, it is called progressive scanning (Fig. 1.3(a)). In contrast, interlaced scanning consists in dividing the frame into, generally, two fields, containing half of the lines in the image each. These fields are scanned one after the other, first the odd field, that comprises odd numbered image lines and then the even field, containing only the even numbered lines, as depicted in Fig. 1.3(b). In a formal way, progressive scanning is a rectilinear scanning process in which the distance from centre to centre of sucessively scanned lines is equal to the nominal line width. On the contrary, in interlaced scanning, that distance is two or more times the nominal line width. It is usually denoted as $n{:}1$, being n the number of different fields in which the picture frame is divided.

The main advantage of interlace scanning is the improved resolution for a fixed sampling rate. While a system employing progressive scanning captures, transmits or displays $N_V \times N_H$-pixel frames at a frame rate given by f_V, an equivalent system based on interlaced scanning delivers fields at the same rate, thus composing a picture frame double the size. Of course, it is done at half the frame rate, but where limitations on the analog video signal bandwidth do not allow a higher sampling rate, it is a method for achieving a higher perceived spatial resolution, as long as lowering the frame rate does not affect seriously the illusion of motion. On the other side, interlaced scanning can generate artifacts like the Kell-effect, what causes a consistent lower vertical resolution, subjectively evaluated, than the geometrically predicted resolution [Tong84].

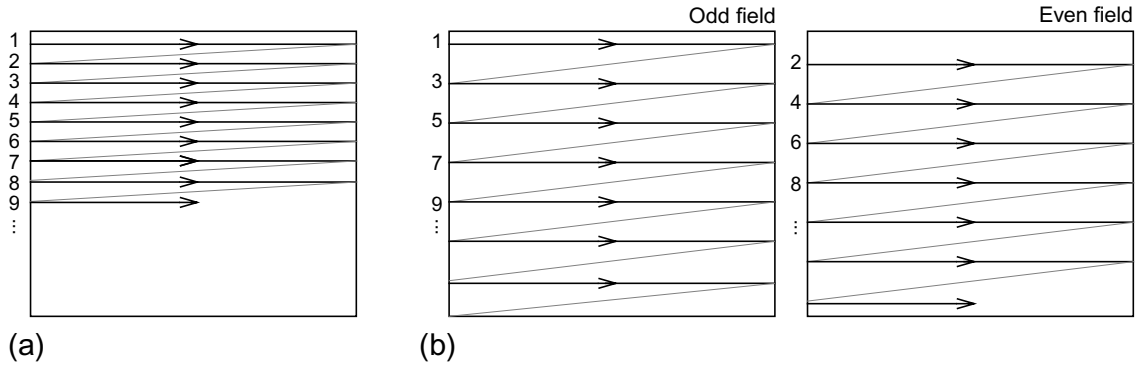


FIGURE 1.3. (a) Progressive and (b) $2{:}1$ interlaced scanning.

Colour encoding

A geometric representation of colour consists in a three dimensional space in which each point stands for a different colour. Its coordinates represent the intensities of light of the three different primary colours that combine to compose the colour corresponding to that specific point. A particular selection of the primary colours leads to a specific color space or colour description, although this selection is not entirely arbitrary. Due to the structure of the human eye, all colours are seen as a combination of the so-called additive primary colours: red (R), green (G) and blue (B). Therefore, the RGB description is a widely accepted selection, being the basis of most image capture techniques and self-luminous displays (CRT). Other selections, like the CMYK description (cyan, magenta, yellow, the substractive primary colours and, optionally, black), are better suited for different display techniques, like printing.

Summarizing, colour information of a picture is conveyed by the signals representing primary colour intensities in each pixel. In the case of a RGB description, signals E'_R, E'_G and E'_B represent the intensity of the three primary additive colours for each image dot, in the same way that E'_Y represents picture brightness as a function of the pixel coordinates (see). E'_R, E'_G and E'_B are normalized, with values between 0 and 1, and gamma pre-corrected in order to represent light intensity linearly in the electrical domain. These signals, however, must be scaled to match the voltage levels for a particular video encoding standard, being V_{black} (Fig. 1.2) the minimum value, corresponding to a zero intensity, and V_{white} the maximum signal voltage, maximum intensity value.

A different description of an image brightness and colour information can be sustained by signals E'_Y, the video luminance, and the so-called colour-difference signals $(E'_R - E'_Y)$ and $(E'_B - E'_Y)$. These signals can be derived from the additive primary colour signals of the RGB component video. Picture brightness is obtained from [CCIR90b]:

$$E'_Y = 0.299E'_R + 0.587E'_G + 0.114E'_B \tag{1.7}$$

while colour-difference signals are given by:

$$\left.\begin{aligned}(E'_R - E'_Y) &= 0.701E'_R - 0.587E'_G - 0.114E'_B \\ (E'_B - E'_Y) &= -0.299E'_R - 0.587E'_G + 0.886E'_B\end{aligned}\right\} \tag{1.8}$$

Notice that only two extra signals are required to support colour information, because it is partially included in the video luminance signal. Also notice that colour-difference signals have a different range than E'_Y. In order to maintain the same signal excursion for the three signals, two re-normalized colour-difference signals, E'_{C_R} and E'_{C_B}, are constructed:

$$\left.\begin{aligned}E'_{C_R} &= 0.500E'_R - 0.419E'_G - 0.081E'_B \\ E'_{C_B} &= -0.169E'_R - 0.331E'_G + 0.500E'_B\end{aligned}\right\} \tag{1.9}$$

which range from -0.5 to 0.5.

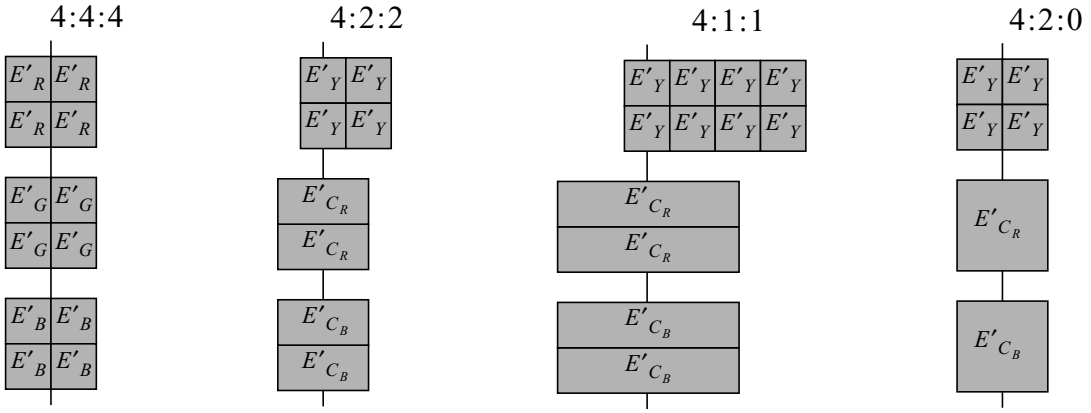


FIGURE 1.4. Colour subsampling schemes.

Due to the relatively poor colour acuity of human vision, the colour information carried by a video signal can be reduced, providing that full luminance information is kept. An image described with an equal number of samples of signals E'_R, E'_G and E'_B, is said to have a sample ratio of $4{:}4{:}4$ (Fig. 1.4). It is common to employ descriptions in E'_Y, E'_{C_R} and E'_{C_B} with colour subsampled signals with ratio $4{:}2{:}2$, i. e. one sample of E'_{C_R} and E'_{C_B} per each two luminance sample in the horizontal scan, where chrominance samples are co-sited (aligned) with alternate E'_Y samples. Other subsampling schemes are $4{:}1{:}1$, with one sample of each chrominance signal is taken for each four samples of E'_Y, also in the horizontal scan, and $4{:}2{:}0$, where subsampling is extended to the vertical scan, having one sample of E'_{C_R} and E'_{C_B} per each 2×2 block of E'_Y samples.

A video system based on three separate signals supporting brightness and colour image information, either E'_R, E'_G and E'_B or E'_Y, E'_{C_R} and E'_{C_B}, is referred as component video. In contrast to component video, composite video carries image luminance and chrominance simultaneously. The signals conveying brightness and colour information are not separated but present in the same channel together. In practice, this is accomplished by superimposing a phase-amplitude modulated sinusoid conveying chrominance information on top of the video luminance signal. Finally, video component signals have to be quantized and encoded to be processed in digital video. E'_R, E'_G, E'_B, E'_Y, E'_{C_R} and E'_{C_B} are then converted to R, G, B, Y, C_R and C_B, following the directions of the corresponding video standard concerning the number of quantization levels, encoding method and thus the number of bits per sample.

Analog video bandwidth and digital data rate

Two related magnitudes are employed to quantify the information rate in video signal processing. For analog video, we can talk about the video signal bandwidth, BW. The analog video bandwidth is the maximum frequency component of the signal beyond which no significant contribution to the signal magnitude spectrum is found. This limit is usually imposed

by the sampling frequency. According to Nyquist's theorem [Oppe93], a bandwidth limited signal, $m(t)$, with bandwidth f_m, is required to be sampled at least at a frequency $2f_m$ in order to be properly reconstructed from its samples. A different reading of Nyquist's theorem would be that any acquisition or display device that operates at a pixel clock of frequency f_S can not capture or deliver signal components with frequencies beyond $f_S/2$. Image features in the original picture with spatial periodicity in the horizontal scan direction below $2w_p$, or, what is equivalent, an spatial frequency above $1/2w_p$, will be missed in the final image. In this conditions, the analog bandwidth of the video signal is given by:

$$BW = \frac{f_S}{2} \tag{1.10}$$

Another consequence of sampling is the aliasing of video signal components above $f_S/2$, so it is convenient to filter them out before processing. Nevertheless, bandwidth limitations to the video signal, especially in TV broadcast, are established by the transmission channel.

The digital counterpart to analog signal bandwidth is the aggregated digital data rate, D. For a monochrome video sequence, D is obtained by multiplying the number of bits employed for coding each video luminance sample, N, times the total number of pixels per image line, N_H, including blanking and synchronization, times the total number of lines, N_V, also with the vertical sync, times the frame rate, f_V. By using the previous definitions, it can be expressed as a multiple of the pixel clock frequency:

$$D = N \cdot f_S \tag{1.11}$$

It is sometimes useful to define a baud rate, b, which is the rate at which symbols, composed of a fixed number of bits, N_b, are transmitted. It is, by definition, a submultiple of D:

$$b = \frac{D}{N_b} = \frac{N}{N_b} f_S \tag{1.12}$$

Analog signal bandwidth and digital data rate are thus related through f_S, yielding:

$$BW = \frac{D}{2N} = \frac{b}{2} \cdot \frac{N_b}{N} \tag{1.13}$$

1. 1. 2. Video signal scanning and encoding standards

Several standard for scanning and encoding analog and digital video signals have been developed as a joint effort between industry, broadcast and communication companies and organizations. Standardization aims compatibility and confluence of research and new achievements. Mass production and marketing obviously benefits from the definition and world-wide adoption of standards. Here, conven-

tional analog and digital TV and computer graphics standards are reviewed, together with HDTV and videoconference formats.

Analog and digital TV standards

NTSC and PAL colour-encoded composite video signals and their associated scanning standards, denoted by $525/59.94$ and $625/50$, are revised in Rec. 601-2 [CCIR90a]. Notation of the scanning standard depicts the number of lines per video frame, N_V, and the flash rate which, in this occasion, refers to the number of fields per second, being both formats $2{:}1$ interlaced (Table 1.1). Frame rates, f_V, for $525/59.94$ and $625/50$ systems are 29.97Hz and 25Hz, respectively. Correspondingly, line frequency, f_H, for each standard is 15.750kHz in $525/59.94$ systems and 15.625kHz in $625/50$. This means a nominal line period, T_H, of $63.5\mu\text{s}$ and $64\mu\text{s}$, respectively. Rec. 601-2 also assigns a total number of video luminance samples per line, including blanking and synchronization, of 858 and 864 samples, where only 720 of them contain actual luminance information. Therefore, the sampling frequency for the luminance signal results in 13.5MHz in both cases, *i. e.* a nominal pixel clock period of $T_S = 74\text{ns}$. Both standards employ $4{:}2{:}2$ colour subsampling, what means that colour-difference signals are sampled at half of f_S, this is 6.75MHz

	525/60	625/50
f_V (Hz)	29.97	25
N_V	525	625
N_i	483	576
f_H (kHz)	15.750	15.625
N_H	858	864
N_{BH1}	122	132
N_j	720	
N_{BH2}	16	12
f_S	13.5MHz	

TABLE 1.1. Picture size and clocks in Rec. 601-2

The actual picture size for these systems is approximately 644×483 and 768×576, respectively —recall here that image size and the total number of lines and samples per line do not match because of the required blanking intervals. Picture aspect ratio is $4{:}3$.

Signal voltages in both standards range from the -300mV, representing the tip of the horizontal synchronization pulse, V_{sync}, to 700mV that correspond to the maximum voltage for each video signal component, V_{white}. Black level, V_{black}, which stands for zero picture brightness, is 0mV. These signals are scaled versions of E'_Y, E'_R, E'_G and E'_B, which are normalized by definition. For digital TV, Rec. 601-2 covers quantization of both composite and component video signals. E'_Y is uniformly quantized into 220 levels, being 16 the corresponding black level. The digital luminance signal Y —decimal value— is obtained from:

$$Y = \text{int}(219E'_Y) + 16 \qquad (1.14)$$

Re-normalized colour-difference signals E'_{C_R} and

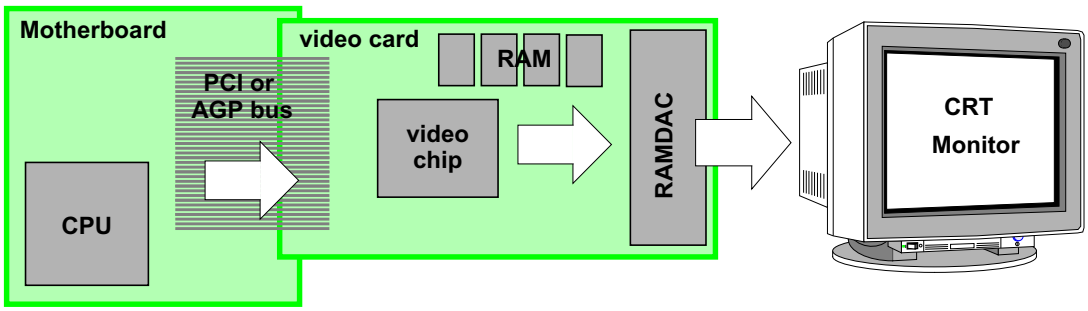


FIGURE 1.5. Computer graphics display system.

E'_{C_B}, in contrast, are quantized into 225 levels with zero level at 128, thus (recall that both signals range from -0.5 to 0.5):

$$\left.\begin{aligned} C_R &= \text{int}(224E'_{C_R}) + 128 \\ C_B &= \text{int}(224E'_{C_B}) + 128 \end{aligned}\right\} \tag{1.15}$$

For component video, the decimal values of R, G and B are computed in the same form as Y, that is, 220 quantization levels with black level at 16:

$$\left.\begin{aligned} R &= \text{int}(219E'_R) + 16 \\ G &= \text{int}(219E'_G) + 16 \\ B &= \text{int}(219E'_B) + 16 \end{aligned}\right\} \tag{1.16}$$

Computer graphics: VESA standards

In 1987, IBM introduced the Video Graphic Array (VGA) display system for PCs, which soon became a *de facto* standard. Extensions to this standard has been developed by the Video Electronics Standard Association (VESA) since then, like SVGA (Super VGA) [VESA90] or XGA (Extended graphic array) [VESA93]. The video display system in a computer embodies three elements, as depicted by Fig. 1.5:

- the video card, also referred as video adapter or graphics card,
- the monitor (most usually a CRT display device),
- and the software driver, that controls the video adapter.

The video card is an extension of the computer's motherboard that includes a video chip, some amount of RAM and a RAMDAC (RAM-based DAC) chip that converts digital data into the analog signals that the monitor can handle. The video chip is in charge of generating the data to compose images in the screen of the computer, it usually implements some functions for graphics acceleration, i. e. drawing lines, polygons, windows. The memory bank is required for storing one or several pictures when processing information involving different frames. A special type of RAM, called Video RAM (VRAM), is

	Resolution	f_H (kHz)	f_V (Hz)
VGA	640×480	31.5	60
VGA	640×480	43.3	85
SVGA	800×600	37.9	60
SVGA	800×600	46.9	75
SVGA	1024×768	60.0	75
SVGA	1024×768	68.7	85
XVGA	1152×864	44.8	47.5[i]
XVGA	1152×864	54.8	60
Vesa 1280	1280×1024	64.0	60

TABLE 1.2. Example of different display modes implemented in a multiscan monitor [Sony97].

i. 95Hz interlaced.

usually employed in video cards. Each VRAM memory cell is made up of two ordinary RAM cells, what allows for simultaneous reading and writing on the same memory address. If the computer generated images are visualized in a LCD (Liquid Crystal Display) device, there is no need for D/A conversion because it handle digital data. A digital graphic adapter can be employed [VESA99].

Concerning the video signal format, VGA and derived standards for computer graphics utilize RGB component video. CRT monitors usually implement different refresh rates by adapting to the received synchronization signals, thus achieving different resolutions (see example in Table 1.2). The number of colours that can be displayed depends on the number of bits employed in the encoding of signals R, G and B, also known as colour depth, which in turn depends on the relation between the total number of pixels (resolution) and the available amount of VRAM. Thus, a so-called true colour representation, 24 bits per pixel (8 bits per colour signal), has a colour palette of about 16.7 million colours. The storage of a 1280×1024-pixel screen in 24-bit RGB format will require nearly 4MB of RAM. A 16-bit description, 5 bits for R and B and 6 for G, also referred as *high colour* description, has 65536 colours available, and represents a 33% reduction in video memory requirements. Further reduction can be achieved by using an indexed colour table in which only a subset of the 16.7 million colours is employed. Only 8 bits are necessary to describe a colour table containing 256 colours, 4 bits for a 16-colour table. It means that less than 2MB and 1MB will be required, respectively, to store a 1280×1024-pixel screen in these colour schemes.

High Definition TV

Rec. 709 [CCIR90c] establishes the basic parameters for the HDTV (High Definition TV) standard, some of them still at a preliminary stage. Also the technical report 3271-E [EBU93] develops the interlaced version for the so-called $1250/50$ HDTV production standard. Not surprisingly, it shares some

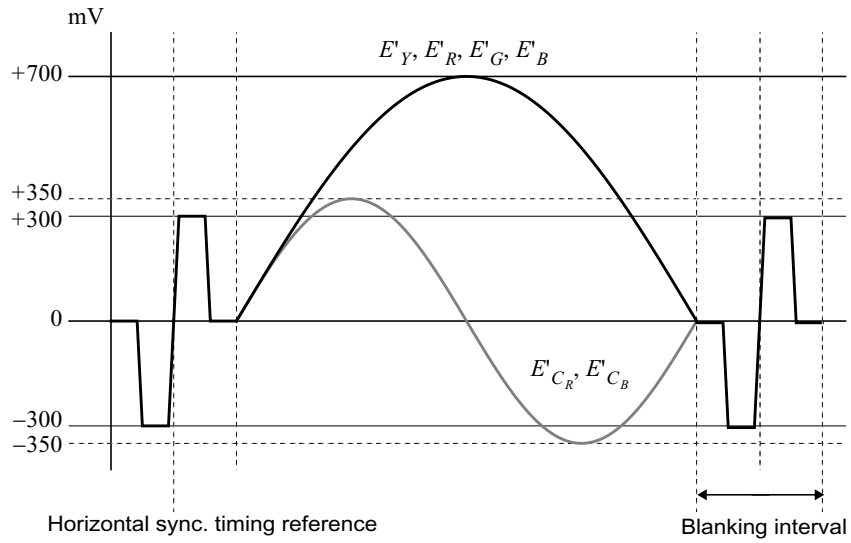


FIGURE 1.6. Component video scan line in HDTV.

common parameters with standard TV, indeed compatibility with existing standard TV systems has been aimed in the development. The first difference is the image size, now 1920×1080. The aspect ratio of the picture has been extended to $16{:}9$. Frame rate is kept at $f_V = 25\text{Hz}$, this is 50 fields per second in a $2{:}1$ interlaced scanning system. The total number of lines per frame, N_V, including blanking, is 1250. It means that the nominal line frequency is $f_H = 31.25\text{kHz}$, what represents a nominal line period, T_H, of $32\mu\text{s}$, half of the $64\mu\text{s}$ for each line in the $625/50$ scanning standard. The total number of luminance samples per line is $N_H = 2304$, and, correspondingly, the pixel clock frequency, f_S, is 72MHz. Colour information, encoded in colour-difference signals in composite video, is subsampled at half of this frequency, 36MHz, maintaining the same $4{:}2{:}2$ subsampling scheme, with chrominance samples co-sited with alternate luminance samples. Another $2{:}1$ interlaced version with 60Hz field refresh rate and 1125 lines per frame is defined in [SMPT94]. The frame rate is now 30Hz and the derived line rate is 33.75kHz, this is a nominal line period of $T_H \approx 30\mu\text{s}$. Being the total number of luminance samples per line equal to 2200, the sampling frequency is 74.25MHz, 37.125MHz for colour subsampling.

HDTV standards also cover either encoding with 8-bit per sample and the extension to 10 bits. Bipolar $\pm 300\text{mV}$ synchronization pulses are introduced, synchronizing at the zero-crossings (Fig. 1.6). Signals E'_Y, E'_R, E'_G and E'_B are scaled to a 700mV range with zero level at 0mV and maximum at 700mV. Colour-difference signals E'_{C_R} and E'_{C_B} are also scaled to a 700mV voltage range, but now between -350mV and $+350\text{mV}$.

Video-conferencing and others

Less restrictive specifications on the frame size and timing are established for videoconferencing by Rec. H.261 [ITU93]. The source coder in this document operates on pictures based on a Common Intermediate Format (CIF) (Fig. 1.7(a)). It consists in non-interlaced 352×288-pixel pictures transmitted at a frame rate of $29.97\,\text{fps}$, approximately $30\,\text{fps}$. Luminance and color encoding follows the same scheme as in Rec. 601, yet the scanning standard differs. The number of pixels per line is compatible with sampling the active portions of the luminance and color difference signals from 525- and 625-line sources at $6.75\,\text{MHz}$ and $3.375\,\text{MHz}$, respectively. Here, chrominance samples are co-sited with each other and with alternate luminance samples too. Rec. H. 261 proposes a smaller format as mandatory for the video source coder, in order to extend compatibility, known as QCIF (quarter-CIF), with a frame size of 176×144 pixels, half of the lines and half of the pixels per line found in a CIF frame. Luminance sampling frequency goes down to $1.688\,\text{MHz}$. Even smaller, sub-QCIF specs are a maximum of 96 lines with no more than 128 pixels each.

Another widely employed format for storage and transmission of digital video is the Standard (or Source) Input Format (SIF) (Fig. 1.7(b)). It constitutes the base resolution for MPEG compression algorithms [ISO93]. The frame size is 352×240 pixels for NTSC-encoded signals and 384×288 for PAL-encoded (recall that NTSC and PAL are color encoding schemes, different from what raster scanning standards $525/59.94$ and $625/50$ mean).

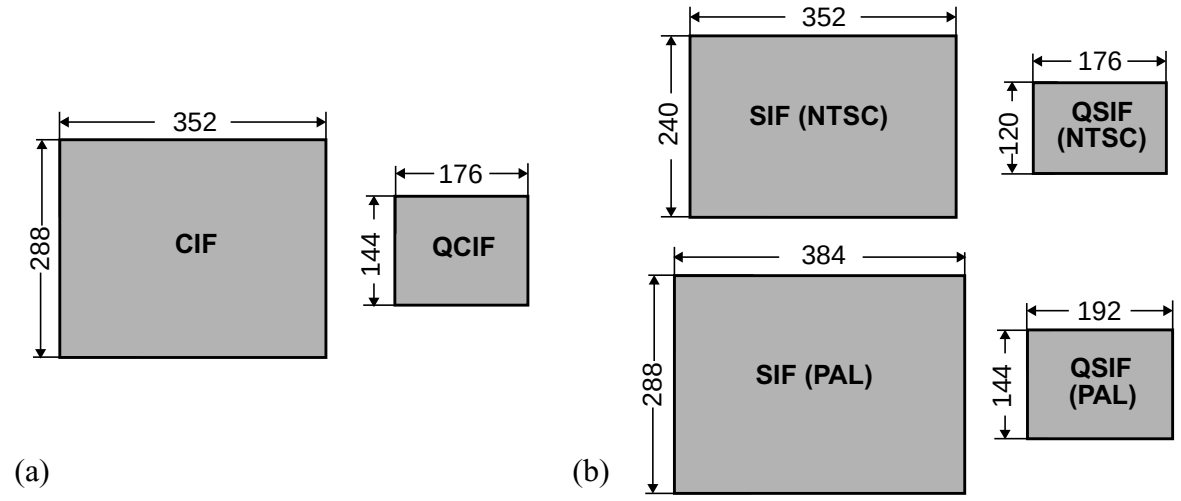


FIGURE 1.7. Frame size in the (a) Common Intermediate formats (CIF) and (b) Source Input Format (SIF).

1. 2. VLSI implementation of real-time image processing

1. 2. 1. Parallel processing architectures

VLSI implementation of real-time image and video signal processing is prompted by the continuing development of new and innovative applications, especially in the fields of robotics, multimedia and telecommunications. VLSI technology offers low power consumption for stand-alone and portable hardware, high speed and performance in time critical applications and low cost to meet mass-market demands. The two alternative approaches to the implementation of real-time image processing in VLSI, from an architectural point of view —both in the analog and digital worlds— are programmable and dedicated architectures [Pirs98]. There are, of course, several advantages and drawbacks in adopting either one or the other approach. On one side, programmable general-purpose digital microprocessors represent a highly flexible implementation, capable of realizing a diversity of image processing algorithms of different complexity. These general-purpose programmable processors may exhibit a significantly high performance level in desktop computing applications but, at the same time, they have rather weak signal processing capabilities. Added to this, they are too expensive and power consuming to be employed in stand-alone and portable multimedia and communications applications. On the other side, a dedicated architecture, based on adaptation to the special characteristics of the algorithm, the video signal format and the inherently parallel distribution of image information, results in a maximally efficient implementation at the expense, indeed, of a reduction in flexibility and algorithm programmability.

Flexibility versus efficiency

Image and video processing applications involve a wide variety of signal processing algorithms of a different complexity and having very dissimilar computational demands. These factors are extremely important in the selection of an efficient architecture. First of all, attending to complexity, image processing operations can be divided into three classes: high-, medium- and low-level tasks. The complexity of a particular image processing operation is related to the regularity of the computational flow, the data size and number, the simplicity of the data structures and the level of abstraction [Pirs98]. A regular computational flow contains a set of predefined operations that are performed repeatedly onto a vast amount of data, with independence of the particular values represented by these data. In contrast, an irregular computational flow is characterized by data-dependent decisions and unpredictability. While the former can be easily performed by a parallel architecture, thus reaching a high level of adaptation and efficiency, this latter counts on flexibility and programmability of the processor. Obviously, this happens at the expense of a considerable hardware overhead for controlling purposes. In addition to complex data flows, high-level tasks are also characterized by the operation onto a reduced number of symbols and image objects, conveying more complicated data structures than single pixels or image macroblocks —small groups of pixels. About

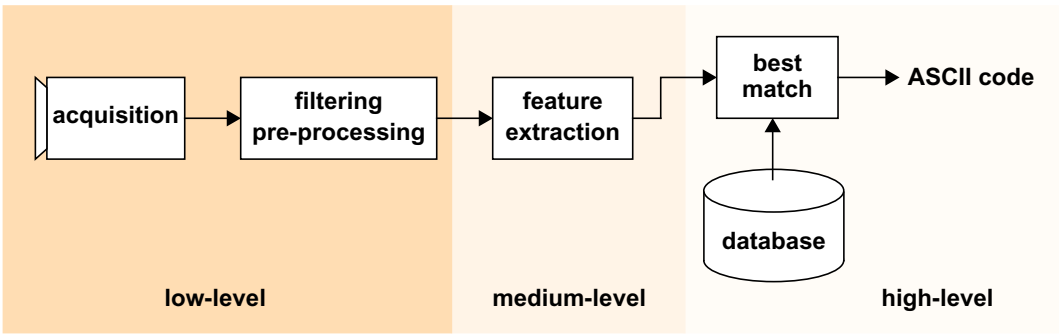


FIGURE 1.8. Conceptual diagram of a generic OCR system.

the computing power demanded by image processing operations of a different complexity, low-level tasks usually exhibit higher computational demands than high-level ones. There seems to be an apparent contradiction here but, actually, high-level operations, that depend heavily on programmability, are performed over a reduced number of elements, of a complex structure but in a smaller number, while the intensive low-level processing tasks are performed onto an exceptionally large number of simple image elements. Summarizing, low-level tasks admit a higher degree of parallelization operating onto simpler data structures, but demand a significant computational effort because of the enormous amount of data to be processed, while high-level tasks, with a very irregular computing flow but operating at a higher level of abstraction, and thus over a smaller data set, are more relying on flexibility than in computation intensive architectures. These arguments are condensed in Table 1.3, below:

	Complexity		
	LOW	**MEDIUM**	**HIGH**
Computational flow	regular	irregular	very irregular
Data structure	simple	simple-complex	complex
Data set	large	medium-large	small-medium
Comput. demand	high	medium-variable	low-variable

TABLE 1.3. Image processing tasks characteristics attending to complexity.

As an example of a complete application consider an OCR (optical character recognition) system, that includes image processing operations ranging from the lowest to the highest abstraction levels. OCR systems are employed to convert printed, or even hand-written, documents into electronic text (Fig. 1.8). The first step in the process, data acquisition, is realized by a camera or a scanner. After scanning, the image is pre-processed. This includes noise filtering and suppression, thresholding, lines detection, thinning, etc. All of these operations are considered low-level image processing tasks. They are performed over the complete data set, with independence of their particular values. Then, feature extraction converts the

raw pixel data into a symbolic representation conveying some important characteristics of the image. This is a medium-level task in which the information of the raw pixels is converted into more complex data structures associating a group of pixels to some distinctive characteristics. Finally, this extracted feature information is employed for comparison and selection of a best match in a character database. This last part of the application is a high-level task, which can have a rather irregular computing flow but operates into a reduced amount of data.

Short-term data storage

A key component of a real-time image processing system is the short-term memory. On one hand, every image processing algorithm computes each pixel's output by considering the information contained in a larger or smaller subset —usually in the neighborhood of that particular pixel— of the original image, or sequence of images. On the other hand, this information is not transmitted all at once, in most of the cases, but following a specific schedule. Therefore, some mechanism for short-term data storage and, eventually, re-organization, has to be provided. A survey on the different alternatives for short-term storage of image data will be given later in this book, by now we are going to concentrate on system requirements.

Let us evaluate the memory needs for different image processing operations. Trivially, the minimum amount of information that is required in the elaboration of one single pixel's output is zero. That is exactly the case if every pixel is set to a fixed value. However, the utility of this operator is certainly restricted. In general, the intensity values of a particular pixel and/or its neighbours' intensities are required to compute its output. Sometimes, the complete picture information is needed. Sometimes more than one picture frame is required. Attending to what we will call the *scope* of the operation, i. e. the amount of pixels required to be present to compute the output of a single pixel, we can establish four classes of image processing operators:

- Pixel-wise operators
- Local operators
- Global operators
- Interframe processing

Pixel-wise operators only depend on the specific pixel value. For instance, a thresholding operator, that compares the pixel luminance value to a preset threshold. For the "peppers" image (Fig. 1.9(a)), it produces an output like that in Fig. 1.9(b).

Local operators require of the information belonging to each pixel in a specific neighbourhood of the processed pixel. For instance, the Sobel operators [Gonz87] compute the vertical and horizontal components of the gradient vector for each pixel, G_x and G_y respectively, with the contributions of the pixels in a 3×3 neighborhood. These are the masks


(a)


(b)


(c)


(d)

FIGURE 1.9. Examples of pixel-wise, local and global operations.

applied to approximate gradient components:

$$M_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \text{ and } M_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (1.17)$$

Fig. 1.9(c) shows the magnitude of the gradient vector, $\text{sqrt}(G_x^2 + G_y^2)$, computed for the image example by this method.

Global operators require the whole image information to compute the output. Histogram equalization is an example of a global operation. It can be considered a medium-level task as long as it operates onto raw image information to generate a new image colour map, what can be seen as a data structure of a higher level. It requires the computation of some image statistics, in particular the accumulated probability, $p_r(\cdot)$, for each gray level, r. As explained in [Gonz87], each new gray level s is obtained from the corresponding old value r by the following transformation:

$$s = 2^N \int_0^r p_r(w)dw \quad (1.18)$$

where 2^N is the number of levels in the original colour map. The effects of this transformation can be seen on Fig. 1.9(d). This picture is encoded in a 8-bits gray scale, what means 256 gray levels. Fig. 1.10 displays the histograms of the original and processed image using both 16 and 256 data bins.

Finally, *interframe* operators —as opposed to intra-frame processing, represented by the previously described operators— require more than one frame to be considered in order to elaborate the output. For instance, it is common in motion detection applications to compute a difference image by substracting the pixel value of a previous frame from the value in the current frame. Fig. 1.11 illustrates this.

As reported in the previous section, still images and motion picture frames are not acquired nor transmitted all at once, what forces the use of a memory buffer. The size of the buffer depends on the specific algorithm. According to the nature of the operators we will require pixel memories, line buffers —recall that video frames are transmitted line-by-line—, frame buffers, etc. This memory buffer has to be updated before the output can be computed. In order to accelerate the process and meet real-time requirements,

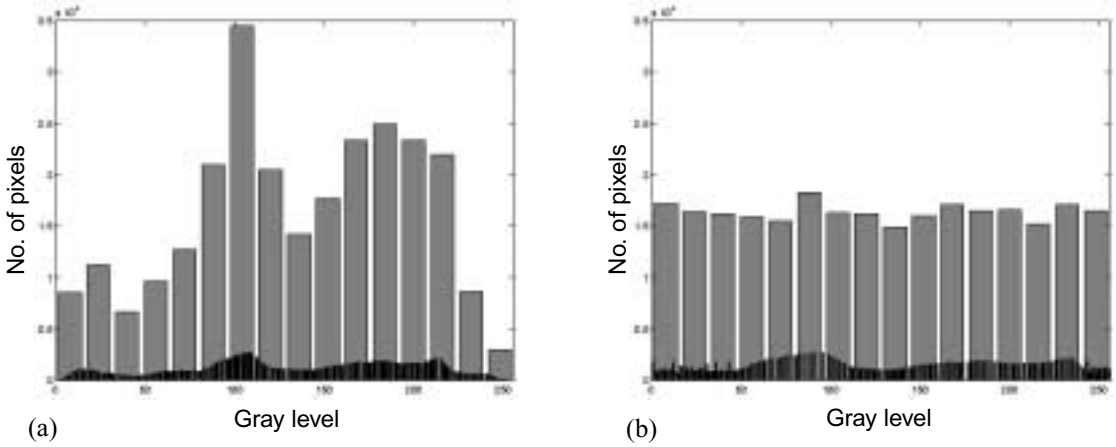


FIGURE 1.10. Histograms of the "peppers" image: (a) original, (b) after equalization.

memory uploading, signal processing and output delivery can be pipelined.

Video signal processing pipeline

Real-time processing of video signals requires the I/O rate of the system to match that of the video signal transmission. Therefore, images have to be processed at f_V frames per second, f_H lines per second, or, equivalently, at f_S pixels per second. Nevertheless, some delay between the acquisition of the first input image samples and the delivery of the first output data available is allowed. This is usually referred as the *latency* of the real-time image processing system. In particular, 2-D image processing usually requires the anticipation of several lines of the input image frame before any output information can be delivered. For instance, MPEG compression algorithms apply the discrete cosine transform (DCT) to 8×8-pixel macroblocks to quantify spatial correlation between neighbouring pixels within the same frame [Siko97]. This means that 8 complete lines —the information in one line is not accessible until the previous line has been completely transmitted— of the original image have to be available before any processing starts. If the total number of lines required to process one line is N_o, the delay introduced is $\tau_o = N_o T_H$. That is only taking into account intraframe processing, but similar considerations can be made for interframe processing. If we assume that N_F picture frames are needed to be present in order to gener-

FIGURE 1.11. Inter-frame processing example.

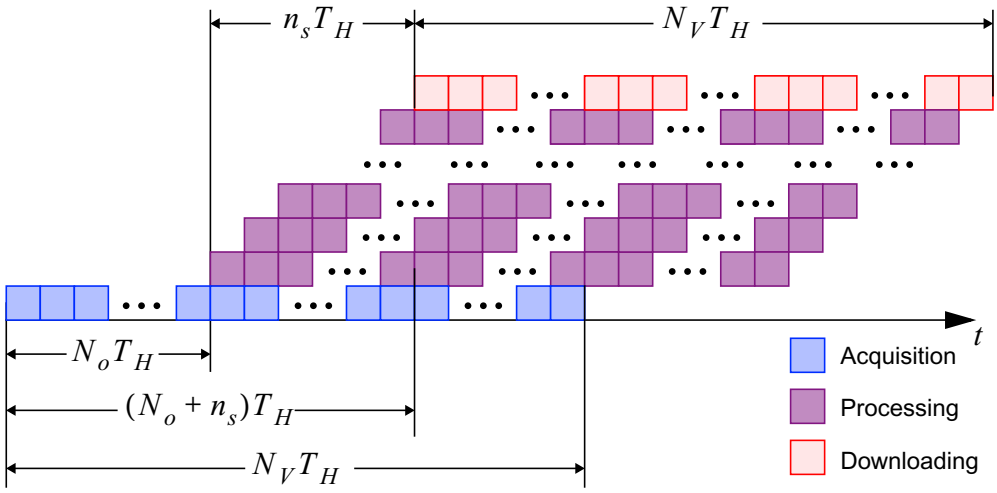


FIGURE 1.12. Pipeline for intraframe video signal processing.

ate one specific frame's output, then $N_F T_V$ seconds has to be added to the already computed initial delay. Once the latency period is passed off, the system must sustain the above mentioned I/O rates. Accepting that a reasonable latency, e. g. several lines or frames, can be allowed in a real-time video processing system, the major timing constraints are found in maintaining the appropriate data rate.

Consider the real-time processing of video signals transmitted at a line rate of f_H. It means that it takes a line period, T_H, to acquire one input line or to deliver the output corresponding to one line. This with independence of the number of lines that are currently stored in the buffer for correct processing. Playing with an arbitrary latency period, we still can design a system to operate at f_H, while processing time is n_s times larger than the line period. For this purpose, the processing system has to be divided into n_s stages —what means extra hardware, but will revert in softer time constraints— and then pipelined together with the acquisition and output stages (Fig. 1.12). It constitutes a $(n_s + 2)$-stage pipeline, what adds $n_s T_H$ to the initially computed delay, resulting in a latency period of $(N_o + n_s) T_H$. Adding the extra delay corresponding to the acquisition of previous video frames for interframe processing, the total system latency is finally given by:

$$\tau_o = N_F T_V + (N_o + n_s) T_H \tag{1.19}$$

Observe that in the example, in Fig. 1.12, the processing itself takes $n_s T_H$ to be realized, while input and output data are retrieved and delivered at f_H, what is only T_H. The advantage of using a pipelined architecture is clear. Real-time processing of a video signal transmitted at a rate f_H has been made possible, just at the expense of a determined initial delay τ_o. In order to optimize pipeline operation, the adequate partitioning of the process into different processing steps is imperative [Poll90]. Peak performance is achieved only by highly adapted architectures, which, however, lack enough flexibility to implement high-level processing tasks. Nevertheless, we are interested in a compromise solution that can achieve enough speed and allow a certain degree of programmability and reconfigurability.

Massively parallel array processors

Digital VLSI implementations of real-time image and video processing can be divided into two major groups with a different architectural approach: dedicated architectures and programmable multimedia processors [Pirs98]. Both strategies try to solve the referred efficiency vs. flexibility trade-off. On one side, dedicated implementations offer the highest efficiency and applications can be found in which they result in the best suited solution. On the other side, however, complex image processing algorithms requires of a certain amount of programmability and reconfigurability. A feasible integration of parallelism and programmability, which can be easily ported to analog array processors, as we will see later, is the SIMD architecture (single instruction stream–multiple data streams). In a SIMD processor [Flyn66], several parallel data paths are centrally conducted by a global control unit (Fig. 1.13). The same operation is executed in all the data paths, but onto different data. Conditional operations can be realized by excluding some data paths from the execution of an instruction by binary masking. The SIMD architecture is, basically, a multiprocessor arrangement in which control hardware has been concentrated into a common control unit. Each processor realize the same operations as the others, upon different data sets, thus exploiting the parallel nature of image processing algorithms. In a sense, it is multiplexing — replicating— hardware in order to gain speed.

A particular case of multiprocessor architecture which is especially suitable for image processing applications is the massively parallel array processor. It consists in a multiprocessor arrangement in which each unitary processor occupies a node of a 2-D lattice. It has also a SIMD architecture so as every processor in the 2-D grid realizes the same operation onto its particular data set. Applied to image processing, this architecture allows mapping a 2-D information signal, e. g. an image, onto the 2-D multiprocessor network and operate in parallel over the complete picture values. The great advantage for real-time image processing is that the whole image is processed in the same time that a serialized architecture realizes processing on a single pixel. If this is combined with a certain amount of local data memory (see Fig. 1.14) the high-speed processing capabilities of such an arrangement are well beyond the gigaOPS. This massively parallel array processor is able to efficiently realize complex

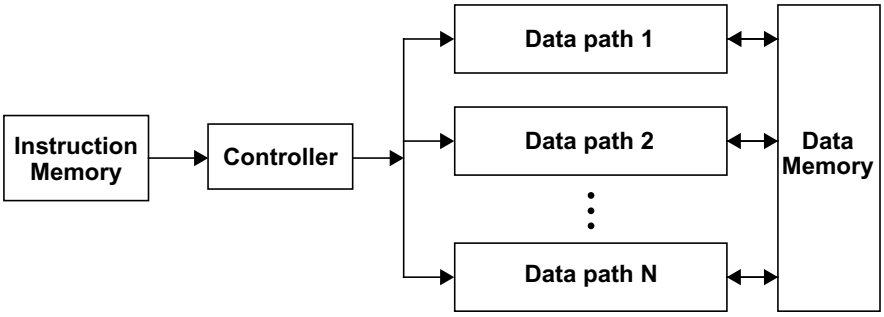


FIGURE 1.13. Conceptual diagram of a SIMD multiprocessor architecture.

image processing algorithms real-time by means of an appropriate synergy between adaptation to the nature of the information signals involved and programmability issues.

Unfortunately, VLSI implementation of a practical number of basic processing units —practical from the point of view of the picture size in industrial applications— is not an easy problem to solve. In the first place, chip size is limited by a desirable fabrication yield, this means that a certain quantity of the produced samples must be fully operational in order to keep fabrication costs under a reasonable bound. For this thing to happen, chip size must be kept under some safety margins, thus unavoidably limiting the number of basic processors to be implemented on-a-chip. Another restriction on the number processing units that can be integrated in the same silicon die is imposed by accuracy concerns. Both in analog and digital realizations, the processor size grows jointly with the accuracy requirements. For digital processors, one more bit of resolution implies at least a duplication of the circuit area [McCl86]. For analog signal processing circuitry, that usually relies on device matching for the cancellation of nonlinearities, larger devices are an obligation to achieve higher precision. It is mainly due to the inverse relation between the statistical deviations of the process parameters and the device size [Pelg89]. Therefore, for a higher accuracy, larger unitary processors are required, what results in a smaller packing density, and thus, a smaller number of basic processing units per chip.

Keeping in mind the pervasive presence of diverse technology limitations, still some strategies can be adopted in order to implement real-time image processing in VLSI. On one side, for a moderate resolution (7-8bits), analog VLSI signal processing represents a faster, more compact and consequently more efficient alternative than area and power consuming digital circuits [Isma94]. On the other side, an optimized system design will enable real-time operation over large images with much smaller array processors.

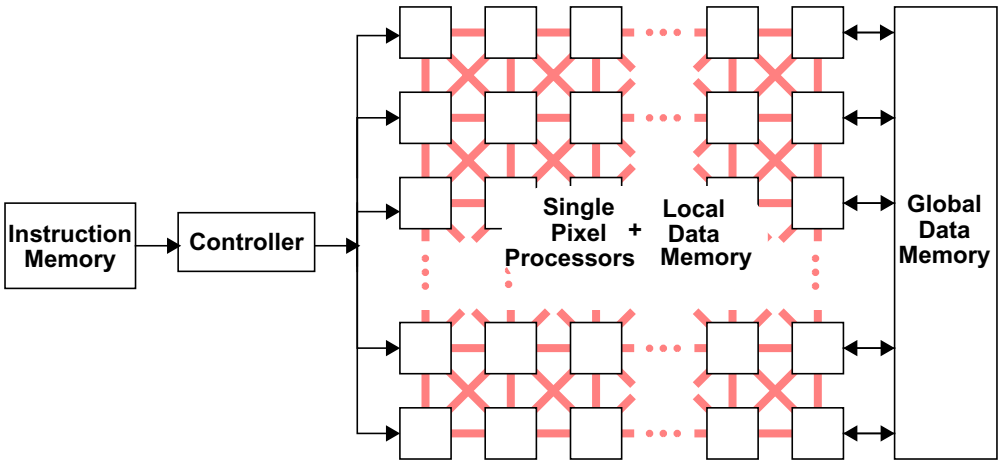


FIGURE 1.14. Conceptual schematic of a massively parallel array processor architecture.

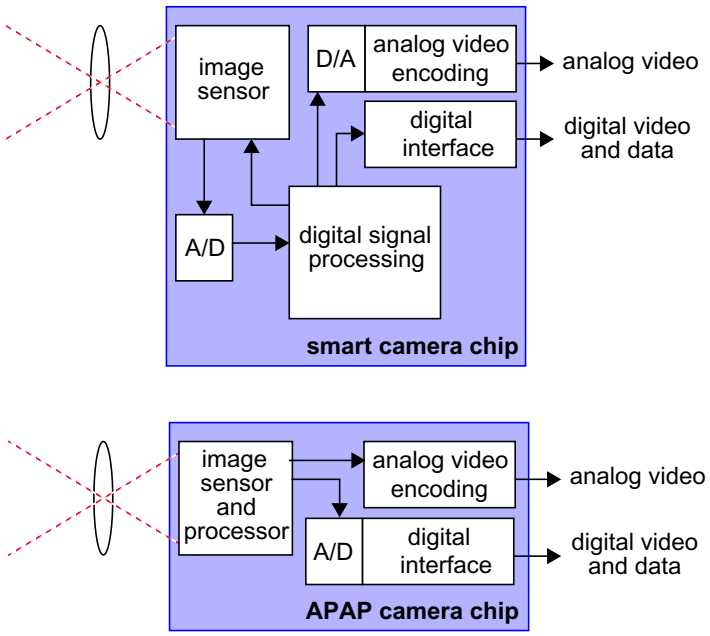


FIGURE 1.15. Conceptual diagrams of a digital smart camera chip and a camera chip based on analog parallel array processing.

1. 2. 2. Video signal processing on a single chip example

Architecture of a smart camera chip

Let us see how the above described architecture can be applied, for instance, to the case of a single chip containing a video signal capture and processing system. A generic case of a video signal processing system-on-a-chip (SoC) is the so-called smart camera [Loin99]. Smart camera chips take advantage of the ability of CMOS processes to integrate sensing and processing in the same silicon substrate —this ability is quite limited in CCD technology. Compared to a conventional video camera system, either based in a CCD or a CMOS sensor, these smart cameras eludes the use of different chips to realize real-time signal processing. As depicted in the upper diagram of Fig. 1.15, the image sensor, the DSP and the analog and/or digital video interfaces share the same silicon die. Also A/D and D/A conversion is provided on-chip in order to convert the analog samples coming from the pixels to digital signals that can be processed, or to generate sensor control signals from the DSP, etc.

A different approach to the smart camera chip is the use of an analog parallel array processor (APAP). In this case, each pixel on the sensor matrix is given processing capabilities. Using analog signal processing at the pixel level instead of digital, which can be less technology dependent and thus more robust, is justified by the difficulties found in integrating signal A/D conversion and power-hungry and area-consuming digital processing circuits together with the sensor at each pixel. The parallelization of the processing yields outstanding speed figures as compared to

sequential counterparts. But, of course, this advantages do not come for free. Maintaining a high accuracy level in analog processing that coexists in the same substrate with a large amount of digital circuitry —required for control, timing, etc.— is very unlikely. Therefore, smart camera chips, with their potential for high processing speed per unit area and power, are restricted to applications in which accuracy requirements are moderate. In performing this tasks, some APAP based chips have outdone pretty neat digital systems.

Analog vs. digital VLSI signal processing

Although the traditional role of analog VLSI processing in mixed-signal integrated circuits has been providing the I/O interface to the core of the chip, which was digital, the situation has essentially changed due to the unsuitability of conventional digital circuits for the processing of large amount of data in parallel. Basically, when massive information flows, usually of a sensory nature, have to be treated in parallel, digital circuits result in an area and power costly alternative due to their reduced computational density. In order to illustrate this, a table (Table 1.4) has been completed with some examples of analog and digital image processors reported in literature. By examining this data, several conclusions can be made, assuming that each of these chips are representative of different approaches to the same problem. The first thing is the appreciably higher computing power rates per area featured by those chips based in APAP, compared with the lower figures obtained for their digital counterparts. The same applies in what refers to computing power versus power consumption. Basic analog processing units are much more power saving than basic digital building blocks. It is of special interest the comparison between the chips presented in [Geal99] and [Liña98]. Both of them employs a massively parallel array processor with the same number of cells. Cell density is similar, but the digital processor perform much simpler tasks, thus this magnitude is not a good reference for the comparison. Taking a look at the processing power (number of operations per second) vs. area and power consumption, the analog array processor features a computing power that is 2 or 3 orders of magnitude above.

1. 3. Image processing with Cellular Neural Networks

In this section, we will introduce the basics of a class of locally coupled neural networks which resemble the already examined SIMD architecture. We will see also how image processing can be easily realize by programming the network analog interactions. These will be the foundations for the development of a hardware environment based on analog parallel array processors (APAP) with outstanding computing power figures when compared to conventional digital signal processors.

Chip ref.	Chip description & design techniques	CMOS process	No. of cells	Cells/ mm^2	XPS^i/ mm^2	XPS/ mW
ANALOG AND MIXED-SIGNAL						
[Espe94b]	Reconfigurable Cellular Neural Network based in Current-Mode techniques	1.6µm	1024	165	85.5G	17M
[Domí97]	Binary I/O CNN with program memory. Analog 4-transistor synapse.	0.8µm	440	27.5	8.25G	0.12G
[Mart98]	Programmable mixed-signal array processor. Based on cyclic ADC-DAC.	0.8µm	25	5.26	4.2M	0.43M
[Liña98]	Gray-scale I/O analog array processor with program memory, 1-transistor synapse.	0.5µm	4096	81	7.93G	0.33G
DIGITAL						
[Yama94]	Digital video signal processor with on-chip memory.	0.55µm (BiCMOS)	64	0.33	1.2M	68k
[Moll98]	Digital video signal processor (with a external FPGA controller).	1.2µm	N/A	N/A	5.0M	0.28M
[Sait98]	Digital ASIC: 1M Synapse Neural Network.	0.25µm	N/A	N/A	44M	2.6M
[Geal99]	Pixel-parallel digital processor with local memory.	0.6µm	4096	66.7	4.0M	1.0M

TABLE 1.4. Comparison between analog and digital programmable array processors

i. IPS: Instructions Per Second, is a typical measurement of a digital processor speed. Common instructions are bitwise addition, complement, shifting, etc. XPS: Analog Operations Per Second, is an equivalent measure indicating the number of analog arithmetic operations like addition, substraction, multiplication and division. In the case that some spatio-temporal dynamics takes place, XPS have been computed by considering a 4th-order Runge-Kutta integration method as proposed in [Harr94].

1. 3. 1. The Cellular Neural Network paradigm

Back in 1988, professors L. O. Chua and L. Yang, from the University of California at Berkeley, defined a new class of analog dynamic array processors: the Cellular Neural Network (CNN) [Chua88]. The elementary processing units of this analog array processor have the distinctive property of being only locally connected. In other words, they interact directly only with those processing units located within a finite local neighborhood, as opposed to the more general fully-connected neural network. In fact a CNN can be seen as a partially connected recurrent neural network (Fig. 1.16). Unlike cellular automata, that are a digital construction, CNNs are composed of analog processors which interact by means of analog signals. The interest of CNNs is grounded on the observation that many tasks related to the processing of multi-dimensional analog signals, such as

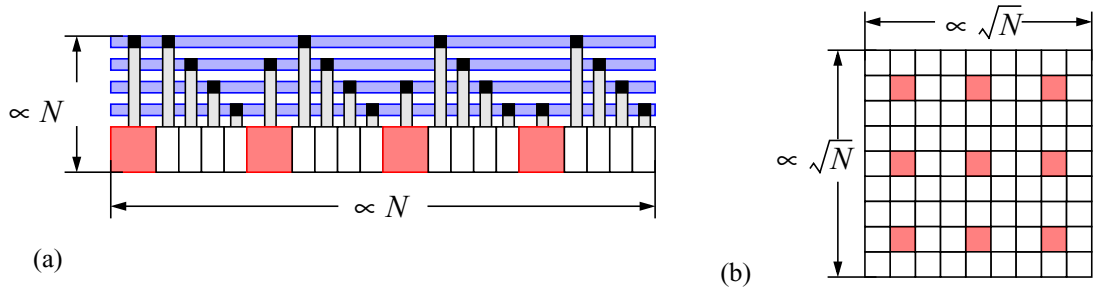


FIGURE 1.17. Conceptual layout of (a) a fully-connected neural network, and (b) a locally-connected network

images, can be formulated as well-defined operations between each individual (unidimensional) signal value (pixel), and the values placed within a local receptive field (neighborhood)—directly mappable onto CNNs for their solution. Furthermore, due to their local connection feature, CNNs are very well suited for VLSI implementation. Specially the class of translation-invariant CNNs, where all cells are identical (with the obvious exception of those in the array boundary), and the connection radius and the interaction pattern remain constant across the network. For instance, in a translation invariant CNN, the area of silicon dedicated to processing and interconnections —synaptic operators and routing— increases linearly with the total neuron count (N), while in a Hopfield neural network [Hayk94], this area is proportional to N^2 (Fig. 1.17). In this section we will consider the evolution undergone by the CNN paradigm since its invention, we will review the

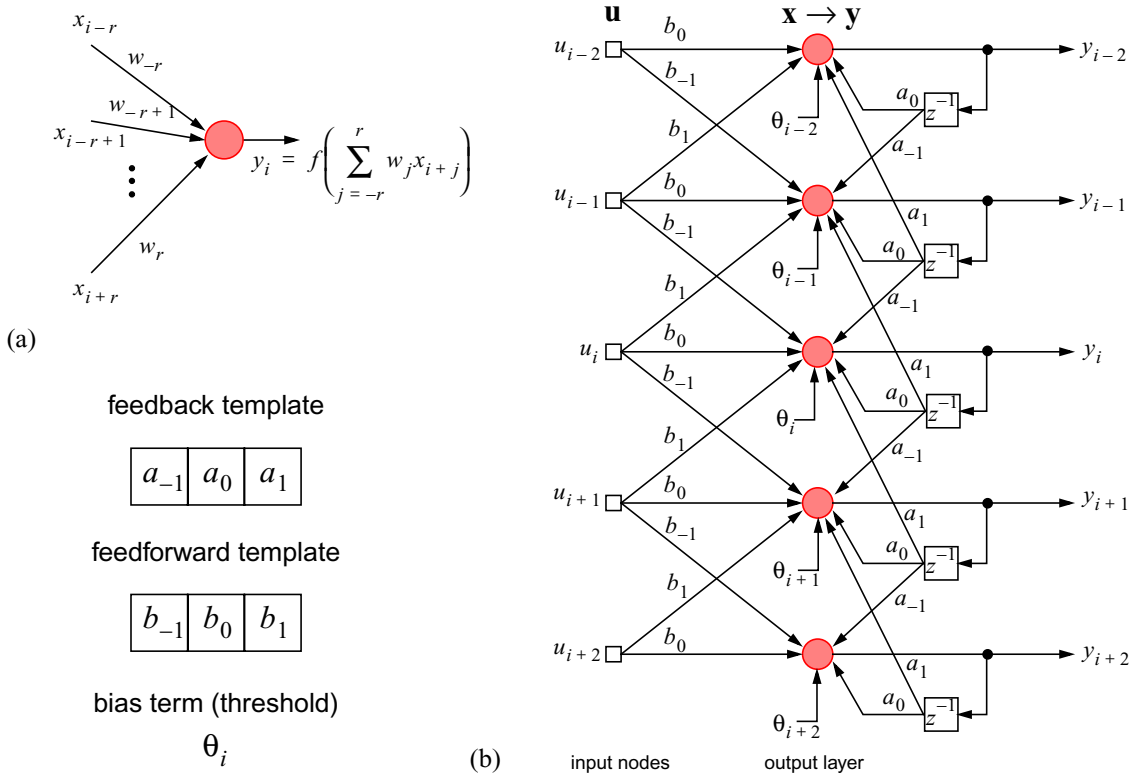


FIGURE 1.16. Architectural graph [Hayk94] of a 1-D CNN: (a) Functionalities represented by the connecting branches and the neuron body and (b) graph of the 1-D CNN. Notice that there are no hidden layers and the network is recurrent and partially connected. Feedforward and feedback contributions are summed and passed through a nonlinear transformation, $y_i = f(x_i)$, activated over a certain threshold $-\theta_i$.

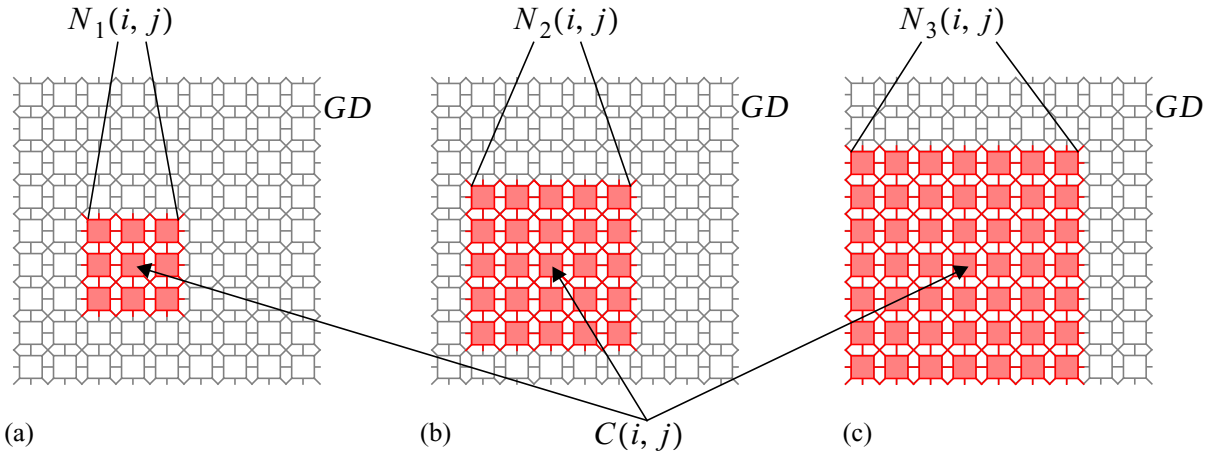


FIGURE 1.18. CNN cell neighborhoods of radius (a) $r_c = 1$, (b) $r_c = 2$ and (c) $r_c = 3$.

mathematical background underneath this paradigm and present different kind of templates —spatio-temporal interaction masks— and a taxonomy of the CNN world.

Definition and general model

According to [Chua93] a Cellular Neural Network (CNN) is an array of nonlinear dynamic processing units —also called cells—, which occupy the nodes of a 2-D, 3-D or n-dimensional grid. These cells satisfy two properties: interactions between them are restricted to a local scope and state variables are continuous-valued signals. The set of all cells in the network define the *grid domain*, GD. For commodity, we will consider a 2-D grid composed by $M \times N$ cells, without loss of generality because definitions and results can be easily extended to higher dimensional networks. Cells belonging to this grid domain are locally connected. This means that each cell of the array interacts directly only to those cells of the grid located within a finite radius r_c. Fig. 1.18 illustrates the concept of the neighborhood of cell $C(i, j)$ —where $i \in \{1, 2, \ldots, M\}$ and $j \in \{1, 2, \ldots, N\}$— for $r_c = 1$, $r_c = 2$ and $r_c = 3$. This neighborhood will be denoted by $N_{r_c}(i, j)$. In these conditions, the topology of the CNN becomes univocally defined by the grid shape (rectangular, hexagonal, etc.), the value of r_c [iii], and a set of boundary conditions affecting to the grid surrounding, GS, that is the set of cells situated less than r_c cells away from the border of the network.

Let us consider, then, a $M \times N$-cell CNN. From a signal processing point of view, the operation of each cell in the CNN, $C(i, j)$, is described by three variables:

- Cell *state*: $x_{ij}(t)$, which conveys cell energy information as a

iii. Notice that a constant radius, $r_c = r$, equal to the largest neighborhood present in a specific CNN model, can be selected. With this, all present neigbourhoods with smaller connection radius are represented by a r-neighborhood with non-existing connections filled with zeroes.

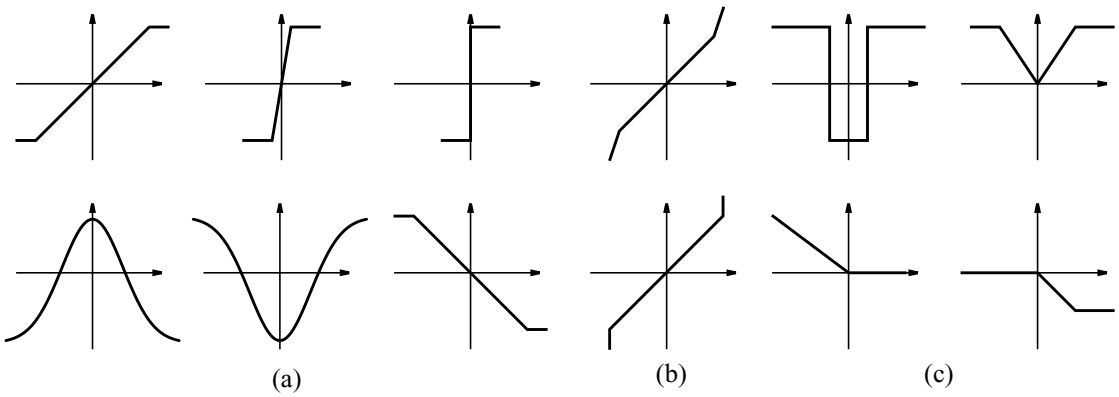


FIGURE 1.19. Nonlinearities found in CNN (a) output functions, (b) dissipative terms, (c) interactions.

function of time[iv].

- Cell *output*: $y_{ij}(t)$, obtained from cell state through a nonlinear transformation,

$$y_{ij} = f(x_{ij}) \tag{1.20}$$

  Fig. 1.19(a) shows some typical shapes observed in CNNs. Here we will restrict ourselves to consider *sigmoidal* approximations of the shapes depicted at the top in this figure.

- Cell *input*: $u_{ij}(t)$, representing external excitation.

Therefore, CNNs are multidimensional signal processing devices with an input vector $\mathbf{u} = \{u_{ij}\}$, $\forall C(i, j) \in GD$, and a vector of initial states $\mathbf{x}_o = \{x_{ij}(0)\}$ and whose outcome is represented by the vector of output variables $\mathbf{y} = \{y_{ij}\}$. These vectors, for a 2-D CNN, adopt the form of 2-D matrices, and this is why they will be referred later as input, initial and output images. For any given input, initial state and network topology, the processing performed by the CNN is determined by:

- The states, $\mathbf{x}_s$, and inputs, $\mathbf{u}_s$, of the cells in the grid surrounding.
- An *evolution law,* which is identical for every cell, described by nonlinear differential equations, nonlinear finite-difference equations, or a mixture of both. A priori these equations can be of any order, but here we consider only CNNs described at the cell level by a first-order differential equation:

$$\begin{aligned}\tau\frac{dx_{ij}(t)}{dt} = &-g[x_{ij}(t)] + z_{ij} + \sum_{k=-r}^{r}\sum_{l=-r}^{r}\{A_{ij,kl}[y_{(i+k)(j+l)}] + \\ &+ B_{ij,kl}[u_{(i+k)(j+l)}] + C_{ij,kl}[x_{(i+k)(j+l)}] + \\ &+ D_{ij,kl}[u_{(i+k)(j+l)}, x_{(i+k)(j+l)}, y_{(i+k)(j+l)}]\}\end{aligned} \tag{1.21}$$

iv. The time variable may either be continuous, represented by t, or discrete, represented by n. Signals in this latter case are valid only at discrete time instants, $t = nT$, where $n = 0, 1, 2, 3, \ldots$

In this equation, τ is referred as the network time constant[v], $g(\bullet)$ is the dissipation function, z_{ij} is the offset term, and $A_{ij,\,kl}(\bullet)$, $B_{ij,\,kl}(\bullet)$, $C_{ij,\,kl}(\bullet)$ and $D_{ij,\,kl}(\bullet)$ are the interaction operators. Fig. 1.19, extracted from [Chua93] and [Espe94a], shows typical shapes for the output and the dissipation functions and the interaction operators. Although the latter will be nonlinear in the more general case, many processing tasks require only linear interaction among cells, thus receiving the name of interconnection weights, as they were referred in artificial neural networks. Also for many applications, $\mathbf{C}(\bullet)$ and $\mathbf{D}(\bullet)$ operators are null[vi].

Continuous-time Chua-Yang CNN model

A particular case of the CNN general model already depicted is the original CNN model introduced by Chua and Yang [Chua88]. This model is a translation-invariant CNN —with a constant neighbourhood radius and a set of interconnection weights that does not depend on the position of the cell in the array. In this model, both the dissipative term and the interaction operators are linear. $\mathbf{C}(\bullet)$ and $\mathbf{D}(\bullet)$ operators are null. Finally, the output (nonlinear) function has a sigmoidal shape. Therefore, the evolution of each cell in the CNN is ruled by the following differential equation:

$$\tau\frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + z_{ij} + \sum_{k=-r}^{r}\sum_{l=-r}^{r}[a_{kl}\cdot y_{(i+k)(j+l)} + b_{kl}\cdot u_{(i+k)(j+l)}] \tag{1.22}$$

where the output nonlinear function has a sigmoidal shape:

$$y_{ij} = f(x_{ij}) = \frac{1}{2}(|x_{ij}+1| - |x_{ij}-1|) \tag{1.23}$$

Fig. 1.20 depicts the schematics for this original CNN model. Here, the cell input, state and output variables are represented by voltages u_{ij}, x_{ij} and y_{ij}, respectively. Neighbours' and self-feedback contributions are introduced in the form of currents by linear voltage controlled current sources (VCCS), with transconductances proportional to the corresponding weight values. These contributions are integrated in the state capacitor C_c. The losses term in Eq. 1.22 is represented by the linear resistance $1/G_c$, which is also the normalizing transconductance for the VCCS. A network time-constant can be defined by $\tau = C_c/G_c$. In principle, it is going to be considered constant along the entire network, although this is not usually the case in physical implementations of the CNN. We will see later in the text which are the

v. At this point we are considering this time constant to be the same for every cell in the network. This will not surely be the case for any physical realization of the CNN, but this is going to be considered later in one of the succeeding chapters.

vi. In the more general case, the $\mathbf{C}(\bullet)$ and $\mathbf{D}(\bullet)$ operators are non null, and the cell interactions are non-linear. However, in most practical cases the nonlinearity appears only at the scaling factor of an otherwise linear transformation.

effects of having a non-uniform time-constant along the array. The bias term (z_{ij}), is introduced by an independent DC current source. For most of the practical cases, the threshold current is considered uniform all over the array of cells, but there are situations in which a non-uniform z_{ij} can help, for instance, to compensate for the effect of process parameters deviation from a nominal central value. Finally, the output voltage is generated from x_{ij} through a linear voltage controlled voltage source (VCVS) with saturation limits, thus implementing a sigmoidal nonlinearity.

As we have already indicated, interconnection operators are linear. And, because connections are restricted to a fixed radius, operators **A** and **B**, have the form of matrices, commonly referred as cloning templates - being the term "cloning" hereby employed in order to emphasize that the network is translation-invariant. These templates are usually known as the feedback and control, or feedforward, templates, respectively. For an unity-radius neighborhood they can be expressed as:

$$\mathbf{A} = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix} \quad (1.24)$$

These templates, together with the offset term and the boundary conditions —fixed state and input of the cells in the grid surrounding— determine the CNN dynamics.

Stability of the Chua-Yang CNN

Being a dynamical system, we can evaluate the conditions for which a CNN converges to a stable equilibrium point after a transient evolution. The convergence properties of this CNN model are analysed in [Chua88] by using the Lyapunov's method [Guck83]. Here we are going to concen-

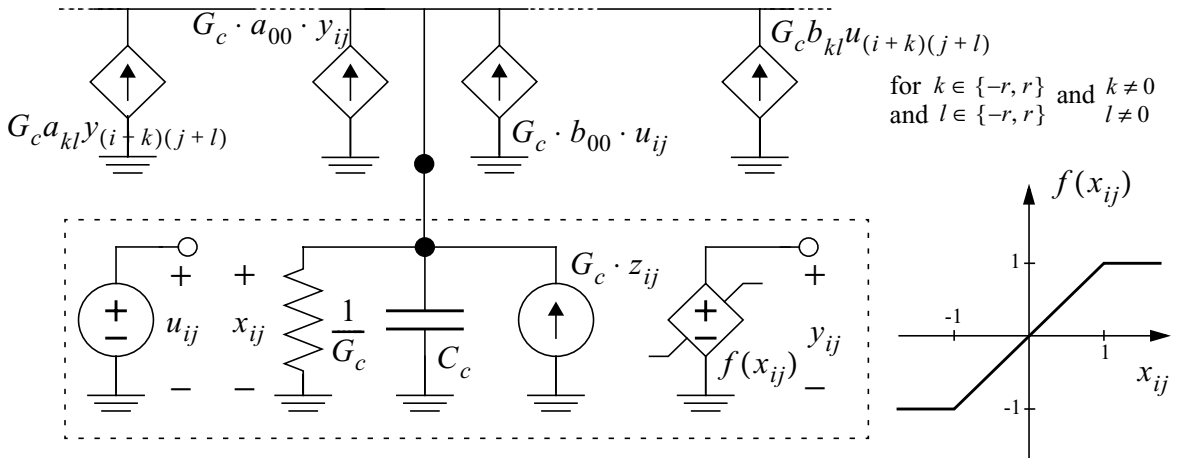


FIGURE 1.20. CNN model originally proposed by Chua and Yang.

trate in the study of the dynamic routes of a single cell in the CNN array, which will depend in the parameters referred above —the template elements, the bias term, the grid surrounding— and, certainly, on the input image ($\mathbf{u}$) and the initial state ($\mathbf{x}_o$) of the network. Let us first assume that every cell in the CNN but cell $C(i, j)$, the one under consideration, has converged to a steady-state. We can re-write equation Eq. 1.22 in the following form:

$$\tau \frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + a_{00} y_{ij} + k_{ij} \tag{1.25}$$

where k_{ij} groups the contributions of the neighborhood and the bias term, or, in other words, all the terms in the right side of Eq. 1.22 that do not explicitly depend on x_{ij}:

$$\begin{aligned} k_{ij} = & \sum_{\substack{k=-r \\ k\neq 0}}^{r} \sum_{\substack{l=-r \\ l\neq 0}}^{r} a_{kl} \cdot y_{(i+k)(j+l)} + \\ & + \sum_{k=-r}^{r} \sum_{l=-r}^{r} b_{kl} \cdot u_{(i+k)(j+l)} + z_{ij} \end{aligned} \tag{1.26}$$

Let us draw the dynamic routes for cell $C(i, j)$: $F(x_{ij}) = \tau(dx_{ij}/dt)$. For different values of k_{ij} and a_{00} we will find different equilibrium points ($F(x_{ij}) = 0$). Consider first that $a_{00} > 1$ (Fig. 1.21). This means that the central part of $F(x_{ij})$ has a positive slope. For this reason, if $k_{ij} = 0$, we have one unstable equilibrium point in $x_{e1} = 0$ —this is, $dx_{ij}/dt = 0$ with $dF/dx_{ij} > 0$. We have also two stable equilibrium points in $x_{e2} = -a_{00}$ and $x_{e3} = a_{00}$. By varying k_{ij} we introduce a vertical displacement of the dynamic route that can result in the loss of one or two of the equilibrium points. Namely, while k_{ij} increases but remains below $a_{00} - 1$, the unstable equilibrium point is given by $x_{e1} = -k_{ij}/(a_{00} - 1)$, while the two stable equilibrium points are located at $x_{e2} = -a_{00} + k_{ij}$ and $x_{e3} = a_{00} + k_{ij}$. As k_{ij} approaches $a_{00} - 1$, x_{e1} and x_{e2} come closer until they merge into a single unstable equilibrium point in $x_{ij} = -1$. If k_{ij} is increased further, x_{e1} and x_{e2} disappear. Some analogous phenomenon can be observed if k_{ij} decreases beyond $-(a_{00} - 1)$, now between points x_{e1} and x_{e3}.

From this discussion we can extract two important conclusions. The first one is that, as long as $a_{00} > 1$, none of the cells in a CNN has stable equilibrium points in the linear region of the sigmoidal output function —the output voltage at steady-state has a saturated value for every cell in the CNN. Thus, after the network reaches steady-state, the CNN output vector $\mathbf{y}$ is binary. For some applications, this result will allow the implementation of a very simple digital output interface[vii]. Secondly, a certain bound can be found for the state variable x_{ij} in the equilibrium. Consider the upper stable equilibrium point $x_{e3} = a_{00} - 1$. If the elements of the

vii. In fact, the first CNN chips that were designed to operate onto binary images, were thoroughly tested with a digital IC evaluation system (HP82000) [Espe93b].

input vector **u** are restricted to the interval $[-1, 1]$, then, using Eq. 1.26 and taking into account that the output of the rest of the cells is ± 1, there is a maximum for x_{ij} given by:

$$|x_{ij}(t \to \infty)| \leq \sum_{k=-r}^{r} \sum_{l=-r}^{r} (|a_{kl}| + |b_{kl}|) + |z_{ij}| \tag{1.27}$$

However, a similar bound can be found for any time instant [Chua88]:

$$|x_{ij}(t)| \leq 1 + \sum_{k=-r}^{r} \sum_{l=-r}^{r} (|a_{kl}| + |b_{kl}|) + |z_{ij}| \qquad \forall t \geq 0 \tag{1.28}$$

These results on the state variable boundedness can be easily extended to the cases in which $a_{00} > 1$ does not hold. On the other hand, the dynamic routes for $a_{00} < 1$ are completely different from those described before. Now, the central part of $F(x_{ij})$ has negative slope as the outer branches (Fig. 1.22) resulting in a single stable equilibrium point. The location of the equilibrium point depends on the magnitude of k_{ij}. It will be situated at $x_e = -k_{ij}/(a_{00} - 1)$ for $|k_{ij}| \leq 1 - a_{00}$, or $x_e = a_{00} + k_{ij}$ if $k_{ij} > 1 - a_{00}$, or $x_e = -a_{00} + k_{ij}$ if $k_{ij} < -(1 - a_{00})$. Notice that a stable equilibrium point can be found in the linear region of the output nonlinear function, i. e. in the interval $[-1, 1]$, in this occasion.

Finally, the case in which $a_{00} = 1$ is an extreme of the already described case. If $k_{ij} \neq 0$ we will have one stable equilibrium point at $a_{00} + k_{ij}$ for $k_{ij} > 0$, or at

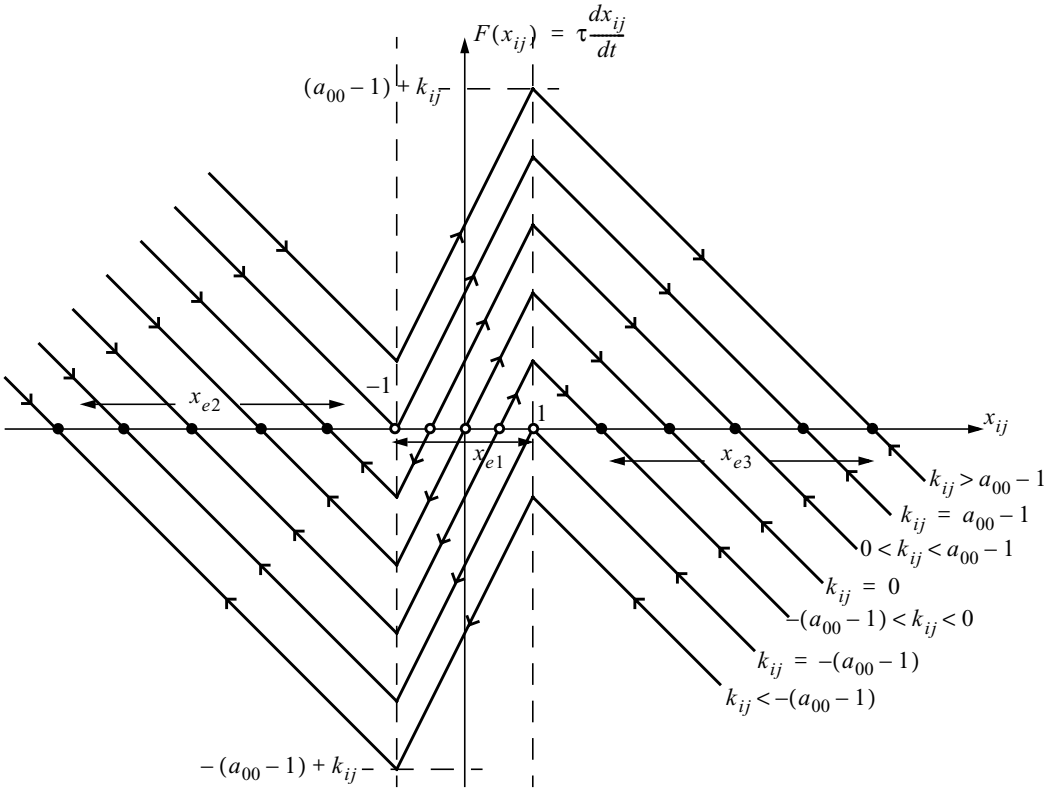


FIGURE 1.21. Dynamic routes of a Chua-Yang CNN cell for $a_{00} > 1$.

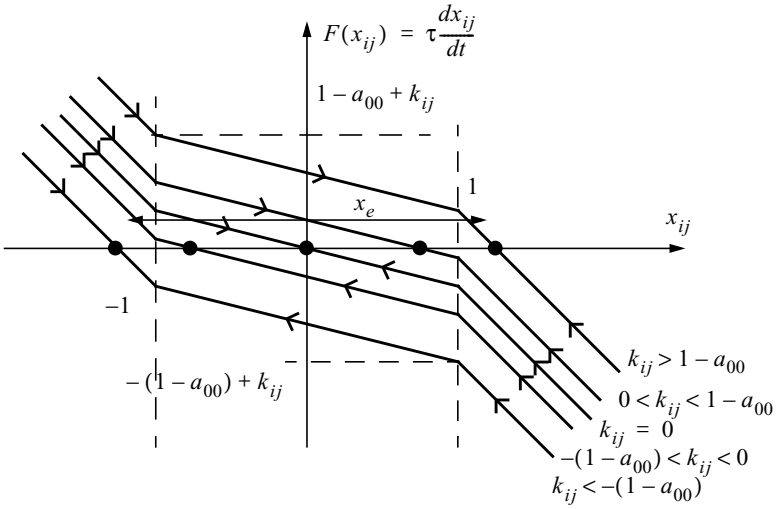


FIGURE 1.22. Dynamic routes of a Chua-Yang CNN cell for $a_{00} < 1$.

$-a_{00} + k_{ij}$ for $k_{ij} < 0$. If $k_{ij} = 0$ there is a complete interval, $x_{ij} \in [-1, 1]$, of stable points which does not represent any interesting dynamics, apart from the fact that, in practice, it would be very difficult to have exactly $a_{00} = 1$ or $k_{ij} = 0$.

Full-signal-range CNN model and others

In addition to the CNN model reported originally, a few more models can be found, all of them contained in the previously described general model for the CNN. However, some of them deserve particular attention. The most important from the VLSI implementation point of view is the Full-Signal-Range CNN model (FSR), developed by Prof. S. Espejo et al. [Espe94a]. The principal characteristic of the FSR model is that the state variable voltage, x_{ij}, is bounded, by means of a nonlinear dissipative term, $g(x_{ij})$. This restriction forces the state variable x_{ij} to remain in a central interval, that can be selected to be $[-1, 1]$. The continuous-time version of this model evolves following this differential equation:

$$\tau\frac{dx_{ij}(t)}{dt} = -g(x_{ij}) + z_{ij} + \sum_{k=-r}^{r}\sum_{l=-r}^{r}[a_{kl}\cdot y_{(i+k)(j+l)} + b_{kl}\cdot u_{(i+k)(j+l)}] \qquad \forall C(i,j) \in GD \tag{1.29}$$

where the nonlinear losses-term is given by (see Fig. 1.19(b)):

$$g(x_{ij}) = \lim_{m\to\infty}\begin{cases} mx_{ij} & \text{if} \quad x_{ij} > 1 \\ x_{ij} & \text{if} \quad |x_{ij}| \le 1 \\ -mx_{ij} & \text{if} \quad x_{ij} < -1 \end{cases} \tag{1.30}$$

Because the state, x_{ij}, and output, y_{ij}, variables have the same values in this model —remember that $y_{ij} = x_{ij}$ as long as $|x_{ij}| \le 1$—, they can

be identified, thus eliminating the need for implementing the nonlinear VCVS at the output of the cell (Fig. 1.20), and, what is more important, the maximum values of the state variable x_{ij}, during transient and at steady-state, do not depend on the template coefficients anymore. It means that voltage excursions of the state variable node will be the same no matter which set of templates we are applying, or, what is equivalent, independently of the CNN operation we are performing onto the input image. This is a very useful result from the VLSI implementation point of view, because, on one side, state variables are bounded —otherwise they will be unwantedly clipped by the nonlinear characteristics of the basic circuit blocks— and, on the other side, because programmability is certainly restricted if the signal range varies depending on the template elements. Convergence and stability of this model are reported in [Espe94b], where they are demonstrated to be similar to those of the original model.

Based on these continuous-time CNN models, discrete-time emulations can be defined. In particular, professors H. Harrer and J. A. Nossek describe their Discrete-Time CNN (DTCNN) by the following recursive formulae [Harr92b]:

$$x_{ij}(nT) = \sum_{k=-r}^{r} \sum_{l=-r}^{r} [a_{kl} \cdot y_{(i+k)(j+l)}(nT) + b_{kl} \cdot u_{(i+k)(j+l)}] + z_{ij} \qquad \forall C(i, j) \in GD \tag{1.31}$$

and:

$$y_{ij}(nT) = \begin{cases} 1 & \text{if} \quad x_{ij}[(n-1)T] > 0 \\ -1 & \text{if} \quad x_{ij}[(n-1)T] < 0 \end{cases} \tag{1.32}$$

They differ from the previous CNN models in that the system is clocked and only binary values for the state variable are permitted, i. e. the output function has a hard-nonlinearity (Fig. 1.19). Convergence and stability has been proven for a quite large set of templates. However, this model has the disadvantage of operating only with binary images. Physical implementations based on this model have a certainly restricted field of application because of this.

Finally, extensions to the original algorithm towards the general model described before were added promptly after the original model was released. For instance, professors T. Roska and L. O. Chua introduced nonlinear and delay-type template elements in [Rosk90]. Namely, they substitute the linear controlled sources of Eq. 1.22 by nonlinear and delayed sources, defining the following evolution law:

$$\tau \frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + z_{ij} + \sum_{k=-r}^{r} \sum_{l=-r}^{r} \{\hat{A}_{ij,kl}[y_{(i+k)(j+l)}, y_{ij}] + A^{\tau}_{ij,kl} \cdot y_{(i+k)(j+l)}(t-\tau) + \hat{B}_{ij,kl}[u_{(i+k)(j+l)}, u_{ij}] + B^{\tau}_{ij,kl} \cdot u_{(i+k)(j+l)}(t-\tau)\} \tag{1.33}$$

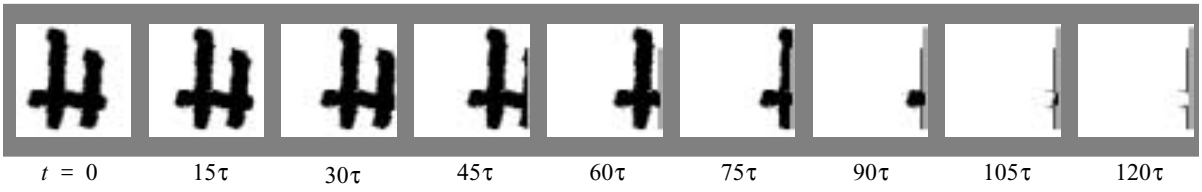


FIGURE 1.23. Connected component detection in a 64×64 -pixel handwritten character.

where it can be noticed that nonlinear template elements represent some functional interaction form that depends in both the source and the target cells' input and output, and that delay-type elements are applied to delayed values of the inputs and outputs of the neighborhood.

From a practical point of view, a thoughtful simplification of the general CNN model (Eq. 1.21) will eventually conduct to a more direct and feasible VLSI implementation. However, complex CNNs exhibit a larger and richer collection of dynamic behaviours, thus allowing for a higher parallelization degree in the implementation of highly complex image processing algorithms.

CNN templates design

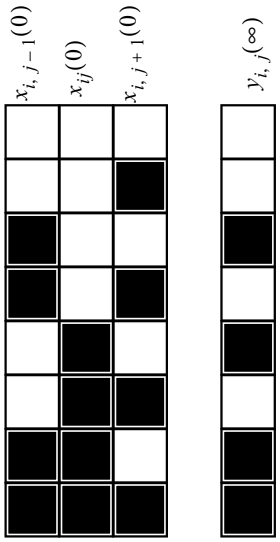
As it was succinctly referred before, the interconnection pattern, defined by the set of templates $\mathbf{A}$ and $\mathbf{B}$, and the bias term z_{ij}, $\forall C(i, j) \in GD$ —together with the particular boundary conditions— determine the evolution and the steady-state values of the network variables. From a signal processing point of view, these parameters determine the operation to be performed in order to map an input image $\mathbf{u}$ into an output image $\mathbf{y}$., assuming that the network has an initial state vector $\mathbf{x}(0)$. The CNN can be seen as a processing system with programmable internal spatio-temporal dynamics that transforms a 2-D (in the most common scheme, although it can be n-dimensional in general) input signal into a 2-D output. The time required for the network to reach steady-state can be seen as the latency of the processor. The set of cloning templates, the interconnection pattern, constitutes the *analog* program of the CNN.

Designing the CNN templates to realize a particular operation can be accomplished by following different approaches. The more direct method is mapping the operation to be performed into a set of local dynamic rules so as the coefficients can be derived analytically [Chua91]. Let us, for instance, consider the case of the Connected Component Detector [Mats90]. This operation consists in detecting the different horizontal segments in a binary picture and producing an output signal which has as many separated black pixels in a row as there were segments in the original image line. Basically, all the CNN does is to propagate the black segments to one end of the row, bearing in mind that if any overlap is permitted some segments will be missed. Fig. 1.23 shows the effect of applying the Connected Component Detector cloning templates to a handwritten character (letter H) scanned on a 64×64 -dot matrix. It can be

seen how information propagates towards the rightmost edge of the picture and each segment is compressed into one single black pixel. Let us now realize how these template elements are calculated. First of all, it seems convenient to examine the operation geometry. In this case it is clear that connected component detection takes place in each horizontal line of the image with independence on what is happening in the neighbouring rows. Therefore, templates will have only three elements —the operation can be defined on a 1-D CNN instead of the most common 2-D CNN— in a row. In addition, as information propagates through the network, we are not interested on having a constant influence of the input image pixels during the whole network evolution, thus, the feedforward (control) template ($\mathbf{B}$) is null. Therefore, connected component detection will be realized by a set of templates of the form:

$$\mathbf{A} = \begin{bmatrix} a_{0-1} & a_{00} & a_{01} \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{z} = z \tag{1.34}$$

The following step is to evaluate the sign of the time-derivative of the state variable x_{ij} of a generic cell $C(i, j)$, for each different combination of initial states of the neighbouring cells, x_{ij-1} and x_{ij+1}. Thus, eight different situations can be found, and, for each one of them, one of the inequalities in (1.35) is derived —keep in mind that -1 represents a white pixel while $+1$ stands for a black pixel:



$$\begin{aligned}
-a_{0-1} - (a_{00} - 1) - a_{01} + z &\le 0 \\
-a_{0-1} - (a_{00} - 1) + a_{01} + z &\le 0 \\
a_{0-1} - (a_{00} - 1) - a_{01} + z &> 0 \\
a_{0-1} - (a_{00} - 1) + a_{01} + z &\le 0 \\
-a_{0-1} + (a_{00} - 1) - a_{01} + z &\ge 0 \\
-a_{0-1} + (a_{00} - 1) + a_{01} + z &< 0 \\
a_{0-1} + (a_{00} - 1) - a_{01} + z &\ge 0 \\
a_{0-1} + (a_{00} - 1) + a_{01} + z &\ge 0
\end{aligned} \tag{1.35}$$

Following the methodology described in [Häng99], we arrive to the conclusion that the templates for realizing connected component detection must be of the form:

$$\mathbf{A} = \begin{bmatrix} a & a+1 & -a \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{z} = 0 \tag{1.36}$$

where $a > 0$. And to finally select some particular values from this family of templates, robustness of the implementation can be studied [King96].

A different approach to compute the necessary template coefficients for a particular operation is learning [Noss93]. This approach is based on a set of L input-

output pairs, from which is difficult to derive a set of dynamic rules in an explicit functional form, that receives the name of training set. Each pair in this training set is composed by an initial state and/or input, $\mathbf{x}^l(0)$ and/or $\mathbf{u}^l$, and the desired output vector $\mathbf{d}^l(\infty)$. Let us define the parameters vector $\mathbf{p} = [a_{-1,-1}, a_{-1,0}, \ldots, b_{1,0}, b_{1,1}, z]^T$. The objective is to find a vector of parameters $\mathbf{p}$, such that the network output $\mathbf{y}^l(\infty)$ matches the desired output $\mathbf{d}^l(\infty)$, for the corresponding initial state and input, for all the pairs belonging to the training set, i. e. $l = 1, \ldots, L$. In other words, this problem consists in the minimization of an error function defined on the parameters space. Consider that the error in the input-output mapping for cell $C(i, j)$, for a vector of parameters $\mathbf{p}$, is a distance metric of norm ν:

$$e_{ij}^l(\mathbf{p}) = \left| d_{ij}^l(\infty) - y_{ij}^l(\infty) \right|^\nu \tag{1.37}$$

We can define the error surface $\varepsilon(\mathbf{p})$ as the sum of all the individual cell errors over the complete training set:

$$\varepsilon(\mathbf{p}) = \sum_{l=1}^{L} \varepsilon^l(\mathbf{p}) = \sum_{l=1}^{L} \sum_{i=1}^{M} \sum_{j=1}^{N} e_{ij}^l(\mathbf{p}) \tag{1.38}$$

The application of the chain rule for obtaining the gradient of this error surface —because we are interested in locating the minimum of $\varepsilon(\mathbf{p})$— leads to algorithms of the backpropagation type. In case the error function is evaluated at some fixed points of the trajectory, e. g. the final states, it results in the recurrent backpropagation algorithm [Pine87]. If the error is computed along a prescribed trajectory we will require backpropagation through-time to solve the minimization of the error surface [Pear89]. However, the nonlinear characteristic of the CNN yields a pretty irregular error surface in which deterministic optimization methods can easily fail to converge to a global minimum. Therefore, the use of *stochastic* —like genetic algorithms [Koze93]—and *perturbative approximation* methods is justified [Cauw96].

Single-linear-template image processing examples

Templates can be found to realize a wide variety of operations onto a binary or a grayscale input image [Rosk95]. Some of them represents a nonlinear or delay-type relation between the state variable and the neighbourhood inputs and outputs [Rosk90]. Although much more interesting operations can be realized with nonlinear operators, we will see later that these nonlinear templates can be decomposed and implemented by linear operators if complex control configurations and local memories are provided at the cell level. Let us concentrate now in some examples of linear-template operations. Consider the pictures depicted in Fig. 1.24. In the upper left corner we have the input image, a grayscale picture with 512×512-pixels and 8-bits depth. The first operation performed over this input image is a linear diffusion controlled by the templates expressed

in the first row of Table 1.5 (see Section 1.3.2). It can be demonstrated that operating this mask onto a grayscale image during a time given by t_o is equivalent to a lowpass filter in the spatial-frequencies domain with bandwidth $1/t_o$ [Jähn99a]. Therefore Fig. 1.24(b) depicts a lowpass filtered version of the input image. The equilibrium point will be in the average value of the image pixels. In this example we have stopped the network evolution after several time constants. The second example, Fig. 1.24(c), is a thresholded and binarized version of the previously diffused image. This is a one-pixel operation and there is no interaction with the neighbourhood (see Table 1.5). In this occasion, $a_{00} > 1$ makes every cell saturate to either black or white depending if $x_{ij}(0)$ was positive or negative respectively. Finally, templates for binary edges detection are employed (Fig. 1.24(d)). Notice here that the cell evolution is driven by the neighbours and compensated by a bias term and the only mission of the feedback template, as it is uncoupled, is to cause the network to saturate to one of the two extremes of the linear region, thus the output will be binary.

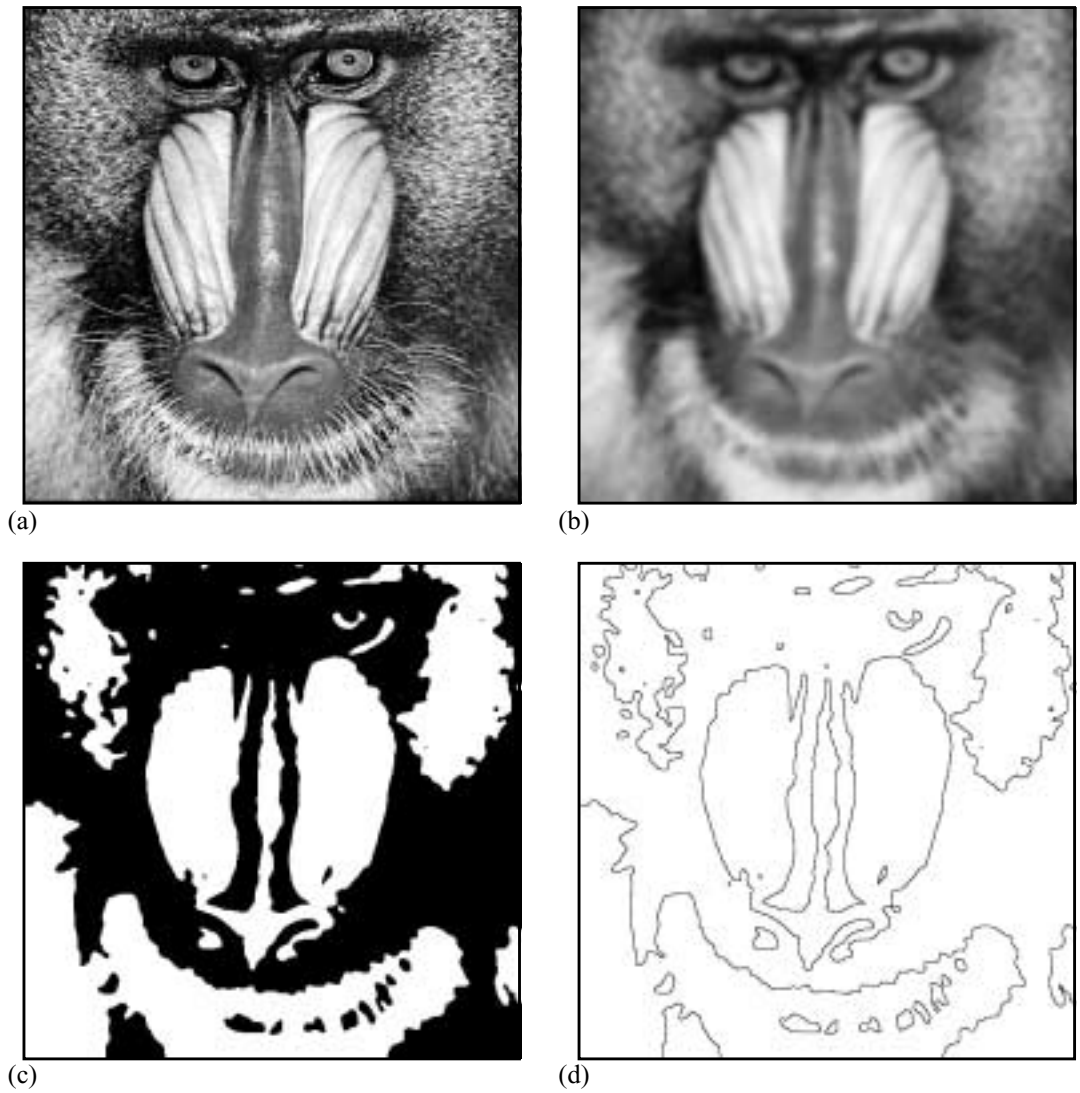


FIGURE 1.24. Examples of single-linear-template CNN operations

Operation	A	B	z
Diffusion (spatial-frequency lowpass filter)	$\begin{bmatrix} 0.25 & 0.50 & 0.25 \\ 0.50 & -2.00 & 0.50 \\ 0.25 & 0.50 & 0.25 \end{bmatrix}$	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	0.00
Threshold (saturated output)	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 2.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	0.00
Edge detector (for binary inputs)	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 2.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	$\begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & -2.00 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix}$	–1.50

TABLE 1.5. Examples of basic operations performed by linear template sets.

1. 3. 2. The CNN Universal Machine

The Cellular Neural Network paradigm has provided the framework for the development of a programmable algorithmic array processor with supercomputer power on a single chip: the CNN Universal Machine (CNNUM) [Rosk93a]. The CNNUM is a hybrid computing architecture that combines spatio-temporal analog dynamics with discrete events [Rosk88]. It has been demonstrated to be universal in the Turing sense [Crou96]. Besides, one the most remarkable characteristic of the CNNUM is its suitability for VLSI implementation, mainly due to its spatial regularity and its restricted local connectivity —the CNNUM implements a space-invariant cellular neural network. Although a considerable design effort must be realized in order to develop a CNNUM chip, it represents the synergy between the aforementioned massively parallel array processor architecture and the most versatile analog signal processing circuitry. The basic processing unit in the CNNUM contains the obvious CNN-core and, together with this, some local analog and logic memories, programmable logic and a reconfigurable control and internal topology. These added capabilities render the CNNUM as a powerful parallel processing device. Several implementations of a single-chip CNNUM have been reported up to now. Some of them feature only part of the characteristics of the CNNUM, because of VLSI implementation constraints [Cruz98] [Espe96b] [Paas97]. But the most recent developments [Liña98] implement a FSR-CNN model with improved accuracy characteristics, an extended set of boundary conditions, analog and digital I/O and local short-term analog memories.

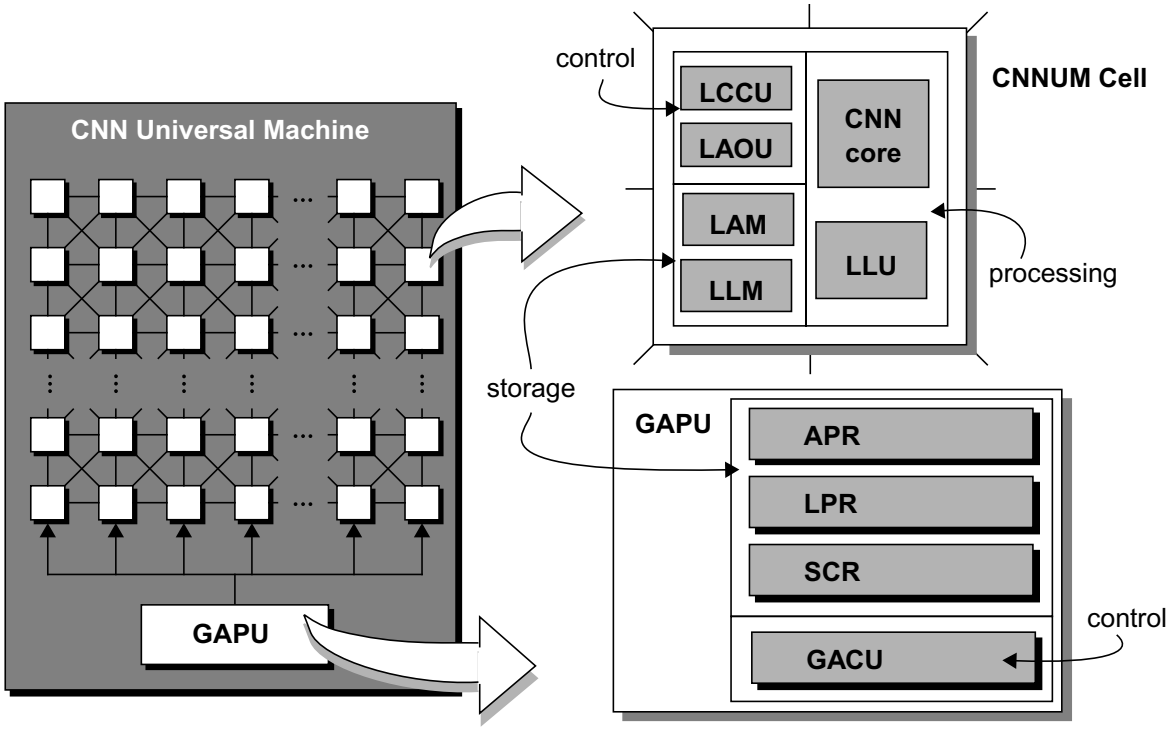


FIGURE 1.25. CNN Universal Machine architecture, basic cell and global analogic programming unit.

The dual computing concept and the CNNUM architecture

The CNN Universal Machine is a hybrid computing architecture that combines programmable spatio-temporal dynamics, supported by an analog processing circuit core, with programmable logic and control driven by digital signals [Rosk88]. In particular, in the CNNUM, the basic CNN cell (see the previous section) constitutes the analog processing core that coexists with local programmable logic circuits and local analog and logic memories. Complex image processing tasks can be implemented in the CNNUM by means of an analogic program (analog and logic) consisting in a sequence of analog and logic operations. In addition, local distributed analog and logic memories allows for the realization of highly complex, recursive and multithreaded algorithms. But let us first describe the architecture of the CNNUM [Rosk93a].

Fig. 1.25 depicts a conceptual diagram of the CNNUM and its main building blocks. In the first place we find the array of analogic processing cells. These cells contain a CNN-core that implements a programmable single-layer linear and time-invariant CNN. However, a more general model for the CNNUM can include nonlinear connection templates, higher order dynamics, etc. Together with this CNN-core, and also being part of the processing area of the cell is the *local logic unit* (LLU), that realizes local programmable logic. Storage of intermediate signals is done in a set of *local analog* and *local logic memories* (LAMs and LLMs). Data exchange and operation control are locally supported by the *local communication and control unit* (LCCU) and the *local analog output unit*

(LAOU) which can have extra processing capabilities like realizing addition and substraction of images, etc. Outside the array, the GAPU stores the analogic program and controls its execution. For this purpose, it is divided into two main functional blocks. On one side the storage unit — containing the machine code instructions for the analog and logic operators and switches configuration— consisting of an *analog program register* (APR), a *logic program register* (LPR) and a *switch configuration register* (SCR). On the other side, the *global analogic control unit* (GACU) that decodes these instructions into a microcode, that is transmitted to the basic cells. Notice that the CNNUM architecture is that of a massively parallel SIMD processor, only now there are analog signal processing blocks coexisting with the programmable logic inside each single-pixel processor.

The analogic CNNUM stored program

A diversity of parallel processing algorithms can be implemented in the CNNUM only by defining the analog program, i. e. the coefficients for controlling the CNN dynamics, the logic program, that is the logic operations to be performed between binary images, and the data flow, which is set by configuring the SCR. Let us see how this is accomplished by an example found in [Rosk93a]. In this example we are going to extract some distinct features from an input image. Character recognition applications are usually based on feature extraction. In this case we are going to localize the intersection between vertical and diagonal lines in a binary picture. Starting from a blurry binary picture (Fig. 1.26(a)), the first step of the algorithm is binarization in order to have a pure 2-level image (Fig. 1.26(b)). In the flowchart of Fig. 1.26, this is indicated by the

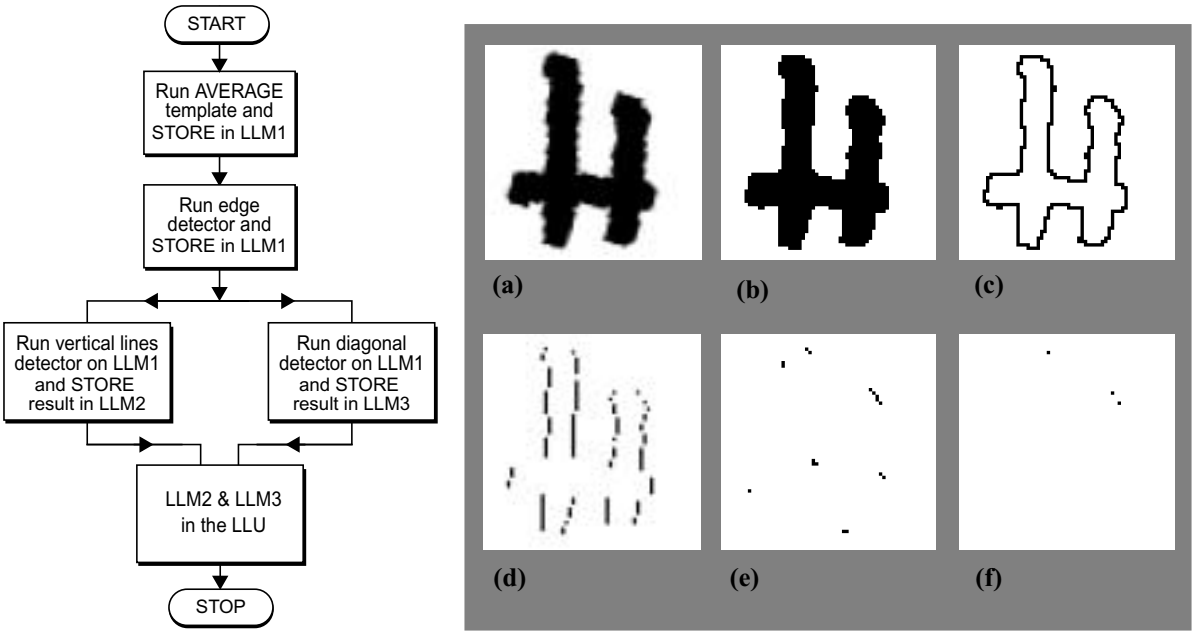


FIGURE 1.26. Flowchart of the CNN analogic algorithm for the detection of intersections between vertical and diagonal lines and input, intermediate and output images.

Operation	A	B	z
Average	$\begin{bmatrix} 0.00 & 1.00 & 0.00 \\ 1.00 & 2.00 & 1.00 \\ 0.00 & 1.00 & 0.00 \end{bmatrix}$	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	0.00
Binary edge detector	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 2.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	$\begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2.00 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix}$	−1.5
Vertical lines detector	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	$\begin{bmatrix} -0.25 & 1.00 & -0.25 \\ -2.00 & 2.00 & -2.00 \\ -0.25 & 1.00 & -0.25 \end{bmatrix}$	−6.25
Diagonals detector	$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix}$	$\begin{bmatrix} 1.00 & -0.25 & -2.00 \\ -0.25 & 2.00 & -0.25 \\ -2.00 & -0.25 & 1.00 \end{bmatrix}$	−6.25

TABLE 1.6. Templates for the flowchart in Fig. 1.26.

upmost rectangular box. The CNN template employed to realize this operation —the AVERAGE template [Rosk95]—is displayed in Table 1.6. The result of averaging is stored in local logic memory LLM1. After that, an edge detection is performed and the result (Fig. 1.26(c)) is send to LLM1 overwriting the previous stored, and now useless, image. The template for binary edge detection is also depicted in Table 1.6. Now, two different CNN operations are performed over the contents of LLM1. First, vertical lines are extracted (Fig. 1.26(d)) with the help of a vertical lines detector template (Table 1.6), and stored in LLM2. Then diagonals are detected (Fig. 1.26(e)) and stored in LLM3. Finally, the LLU is employed to realize a logic AND between the contents of LLM2 and LLM3. This results in the intersection of vertical and diagonal lines (Fig. 1.26(f))

Let us see how this algorithm is introduced in the GAPU. For the analog core, the APR must contain the four sets of templates expressed in Table 1.6, which are required to implement the already described algorithm. Let us call them TEM1, TEM2, TEM3 and TEM4. The operations of the LLU are also coded and stored in the LPR. In this occasion, only a logic AND is programmed, let us call it LLU1. Added to this, 15 different switch configurations must be coded and stored in the SCR in order to realize the data tranfers foreseen in Fig. 1.26. Table 1.7 assigns different names to those switch configurations that are going to be kept at the SCR.

Using a high-level language to describe the analogic program, it looks like that listed in Fig. 1.27 [Rosk93a]. At each step of the program either a different switch configuration is selected, or the CNN core is tuned to implement a new set of templates, or the LLU is programmed to realize a different logic operation. This higher-level description has to be translated to the CNNUM chip machine code by the appropriate compiler [Rosk99].

Name	Operation
SC0	Reset
SC1	Acquire input and store it in LAM1
SC2	Load LAM1 into the CNN core as input
SC3	Load LAM1 into the CNN core as initial state
SC4	Load LLM1 into the CNN core as input
SC5	Load LLM1 into the CNN core as initial state
SC6	Load LLM2 into one LLU input
SC7	Load LLM3 into one LLU input
SC8	Store CNN output in LLM1
SC9	Store CNN output in LLM2
SC10	Store CNN output in LLM3
SC11	Store LLU output in LLM4
SC12	Start CNN dynamics
SC13	Activate LLU output
SC14	Send LLM4 to LAOU

TABLE 1.7. Switch configurations for the algorithm in Fig. 1.26.

Mathematical morphology on the CNNUM

Mathematical morphology is a very helpful technique for the analysis of spatial structures in an image [Serr82]. It is called morphology because it aims at the shape and form of the objects. It results obvious that mathematical morphology analysis is of a tremendous interest in image segmentation, which is the key to content-based image compression and coding and other applications based on image understanding. Morphological operators for the analysis of images, like erosion, dilation, etc., belong to the class of nonlinear neighborhood operators [Jähn99b]. It makes sense, therefore, to consider the implementation of mathematical morphology in the CNNUM [Zará98]. In this section we are going to describe an example in which morphological operators are employed. It is important to notice that no data transaction with the outside is realized apart from input image acquisition and output delivery. The complete processing algorithm is internal to the CNNUM thus avoiding data transfer bottlenecks.

In this example, the analogic algorithm is intended to localize and display separately bubbles of a different size in a picture with many of them. Realizing this operation real-time —at standard video rates— with conventional digital serial algorithms can be at best too expensive in terms of area and power. Let us see how this can be easily accomplished by the CNNUM in parallel. The flowchart of the algorithm is depicted in

```
Begin                    — program starts
   Select SC0;           — reset the CNNUM
   Select SC1;           — acquire the input and store it in LAM1
   Load TEM1;            — adjust the CNN core to realize the averaging template
   Select SC2;           — load the content of LAM1 as the input image
   Select SC3;           — load the content of LAM1 as the initial image
   Select SC12;          — start CNN dynamics
   Select SC8;           — store the output into LLM1
   Load TEM2;            — adjust the CNN core to realize TEM2
   Select SC4;           — load the input image from LLM1
   Select SC5;           — load the initial state from LLM1
   Select SC12;          — start CNN dynamics
   Select SC8;           — store the output into LLM1
   Load TEM3;            — adjust the CNN core to realize TEM3
   Select SC4;           — load the input image from LLM1
   Select SC5;           — load the initial state from LLM1
   Select SC12;          — start CNN dynamics
   Select SC9;           — store the output into LLM2
   Load TEM4;            — adjust the CNN core to realize TEM4
   Select SC4;           — load the input image from LLM1
   Select SC5;           — load the initial state from LLM1
   Select SC12;          — start CNN dynamics
   Select SC10;          — store the output into LLM3
   Load LLU1;            — adjust the LLU to realize LLU1
   Select SC6;           — set LLM2 as the first LLU input
   Select SC7;           — set LLM3 as the second LLU input
   Select SC13;          — activate LLU
   Select SC14;          — send the LLU output to the LAOU
End                      — program termination
```

FIGURE 1.27. Analogic program for the detection of intersections between vertical and diagonal lines.

Fig. 1.28. First of all, the input image (Fig. 1.29(a)) is acquired and stored in LLM1. Then a mask with the shape of the different objects in the input (Fig. 1.29(f)) is created by the subroutine in Fig. 1.28. We will go over it later. Then this mask suffers and erosion (Fig. 1.29(g)) with the template in Table 1.8 (L-shape erosion). Then the RECALL template is applied and the result (Fig. 1.29(h)) is stored in LLM3. This image suffers another L-shape erosion (Fig. 1.29(i)) and the recalled image (Fig. 1.29(j)) is stored in LLM4. at this point, the CNN core has finished its job. The rest of the algorithm is realized by the local logic unit (LLU). First LLM2 and LLM3 are XORed to obtain a mask for the smallest bubbles (Fig. 1.29(k)). Then, LLM3 and LLM4 are XORed to obtain a mask for the medium bubbles (Fig. 1.29(n)), while LLM4 constitutes itself a mask for the largest size bubbles (Fig. 1.29(j)). Finally, these masks are ANDed with the original input in LLM1, in order to obtain OUT1 (Fig. 1.29(p)), OUT2

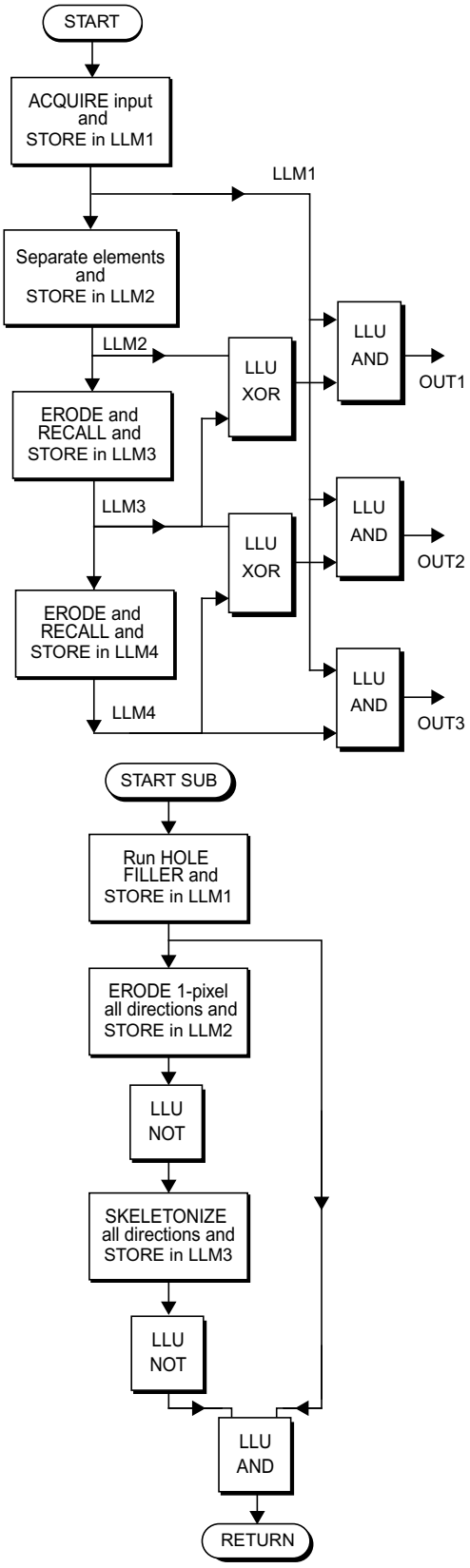


FIGURE 1.28. Analogic algorithm for object classification in the CNNUM and the object separation subroutine.

Operation	A	B	z
Hole Filler	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	−1
Erode up (rotate for left, down and right)	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	−2
Skeletonize NW (rotate for NE, SW, SE)	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 7 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	−3
Skeletonize up (rotate for left, down and right)	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 7 & 0 \\ -0.5 & -1 & -0.5 \end{bmatrix}$	−3.4
Erosion L-shape	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$	−2.5
Recall	$\begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 4 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	2.5

TABLE 1.8. Linear template sets for the mathematical morphology application.

(Fig. 1.29(q)) and OUT3 (Fig. 1.29(r)), which represent different versions of the input containing only the smallest, medium and largest bubbles. Concerning the object separation subroutine, it is based on an operation called skeletonization, that consists in peeling the objects 1 pixel in each direction but conserving the connectivity. If we have a look at the example, the holes in the input image (Fig. 1.29(a)) are first filled, ending in Fig. 1.29(b). Then the image is eroded one pixel in each direction (Fig. 1.29(c)), then inverted (Fig. 1.29(d)) and then skeletonized and inverted again, resulting in Fig. 1.29(e). If this image is ANDed with the hole-filled version of the input, it results in the mask Fig. 1.29(f), that contains the shape of every element in the original image but separated into disconnected patches.

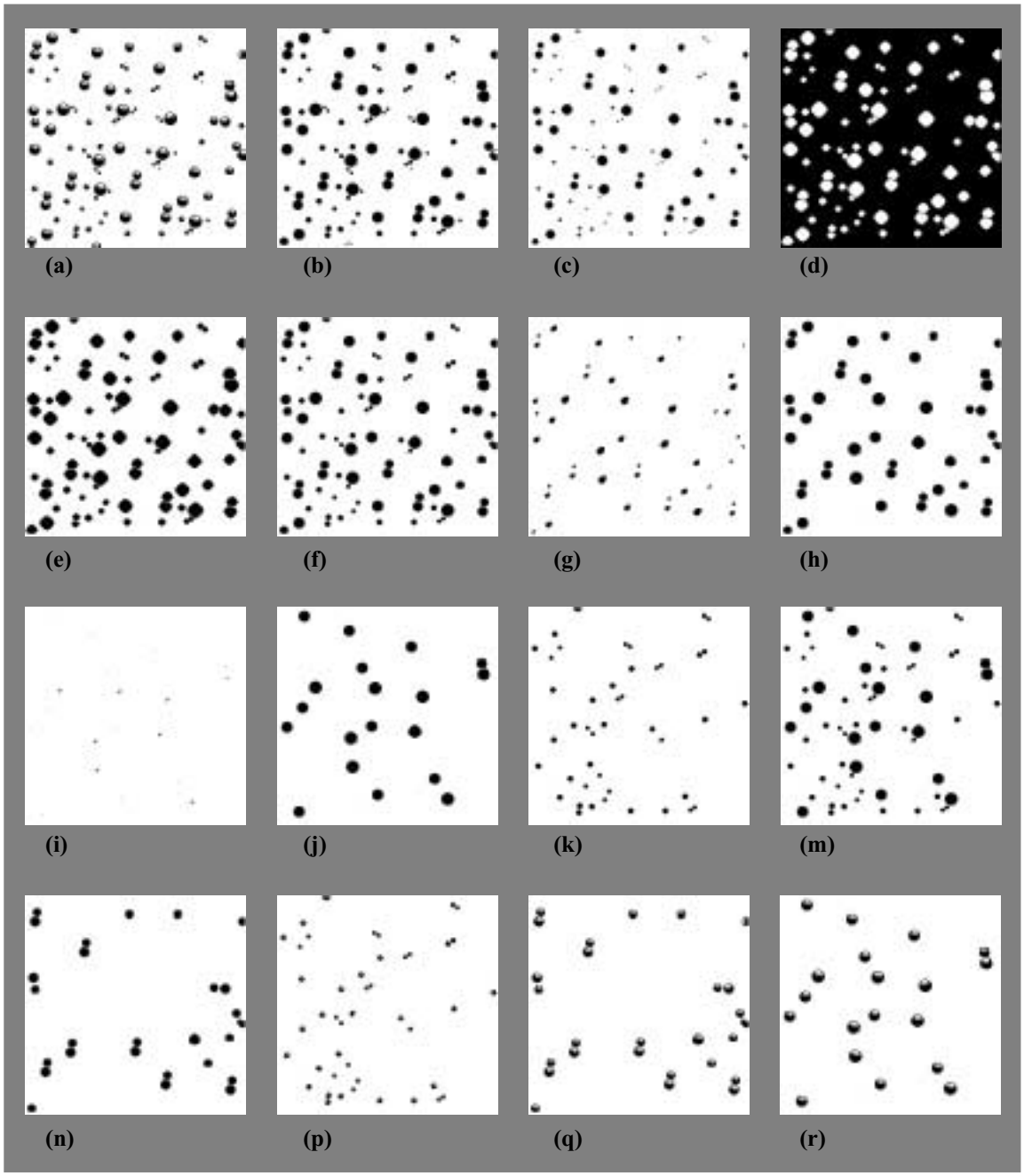


FIGURE 1.29. Input, outputs and intermediate images of the objects separation algorithm.

Fixed state map: a grayscale image restoration example

Finally, let us consider an operation example concerning grayscale image processing. In this occasion we have a 8-bit deep gray-scale input image corrupted by impulsive noise with a 20% probability of occurrence (Fig. 1.33(a)). This artifact —also known as "salt and pepper" noise— is common in electronic imaging due to saturation of nonlinear devices, errors in the acquisition or transmission of information packets, etc. In order to restore the original image, or recover as much as we can from the original image information, we are going to use another feature that can

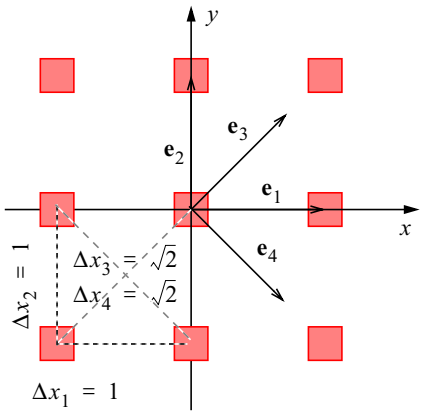


FIGURE 1.30. Base vectors for a rectangular grid

be implemented in the CNNUM: the fixed state map (FSM) [Rosk93a]. It consists in a binary mask that inhibits dynamic evolution of the state of the masked cells. In other words, only the cells that are marked (or unmasked) will evolve once the CNN dynamics are started with the presence of a FSM. This property is inherited from digital multiprocessor SIMD architectures in which data-dependent computational flows are implemented by binary masking of the desired data path [Pirs98].

Together with the FSM feature, this application will be based on the extensive use of the spatial-diffusion template (Table 1.9). Let us first discuss how this template set is derived. Consider a scalar spatio-temporal image property defined over a rectangular grid, e. g. pixel brightness $I(x_1, x_2, t)$ (see Fig. 1.30). The diffusion equation, also known as heat equation [Habe87], for this spatio-temporal function $I(x_1, x_2, t)$, can be expressed as:

$$\frac{\partial I}{\partial t} = D \cdot \nabla^2 I \tag{1.39}$$

where D is the diffusion coefficient. The Laplacian operator in the right-hand side of Eq. 1.39 can be approximated by the following finite-difference equivalent, bearing in mind that the space is discretized around the pixel under consideration and that there are neighbouring pixels in the four directions denoted by $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ and $\mathbf{e}_4$:

$$\nabla^2 I = \sum_{i=1}^{4} \frac{I(x_i + \Delta x_i) - 2I(x_i) + I(x_i - \Delta x_i)}{(\Delta x_i)^2} \tag{1.40}$$

Taking into account that $\Delta x_1 = \Delta x_2 = 1$ and $\Delta x_3 = \Delta x_4 = \sqrt{2}$, and the discreteness of the image plane (Eq. 1.2), thus picture brightness samples can be referred as $I(i, j)$, Eq. 1.39 results in:

$$\begin{aligned} \frac{1}{2D}\frac{d}{dt}I(i,j) = {} & \frac{1}{2}I(i+1,j) + \frac{1}{2}I(i-1,j) + \\ & + \frac{1}{2}I(i,j+1) + \frac{1}{2}I(i,j-1) + \\ & + \frac{1}{4}I(i+1,j+1) + \frac{1}{4}I(i-1,j-1) + \\ & + \frac{1}{4}I(i-1,j+1) + \frac{1}{4}I(i+1,j-1) - 3I(i,j) \end{aligned} \tag{1.41}$$

Making $D = 1/2\tau$ and comparing this to Eq. 1.22, what

we have here is a linear single-layer space invariant CNN with the set of templates for linear diffusion in Table 1.9. The effect of running this template during t_o seconds is equivalent to a convolution in the spatial domain of the original image with a 2-D Gaussian function $h(x, y)$ of the form [Jähn99a]:

$$h(x, y) = \frac{1}{4\pi D t_o} e^{-\frac{(x^2 + y^2)}{4Dt_o}} \Leftrightarrow (H(k_x, k_y) = e^{-4\pi^2 D t_o (k_x^2 + k_y^2)}) \qquad (1.42)$$

where D is the diffusion coefficient, the functional form $H(k_x, k_y)$ is the 2-D Fourier transform of $h(x, y)$, and k_x and k_y are the wave numbers in the reciprocal space. $H(k_x, k_y)$ is a lowpass Gaussian spatial filter with a bandwidth that is inversely proportional to t_o (Fig. 1.31).

These two elements, the fixed state map and the spatial-diffusion template set, will help us eliminate the impulsive noise from the corrupted image. It can be seen from Fig. 1.33(b) that lowpass spatial filtering of the corrupted image does not result in a very convenient output. On one side, the diffusion operator causes sharp edges of the image to disappear —they have been blurred because the lowpass filter has clipped the spatial spectrum of the image. At the same time, impulses information belonging to the corrupted pixels has expanded towards their neighbours causing some fuzzy areas to appear around the points originally affected by noise. The problem is that in computing the local average for every pixel in the image, those pixels in the neighbourhood of a noisy pixel present a distorted local average value. The impulse value pulls up (or down) the mean value for their neighbouring pixels to a great extent, what results in the diffusion of the impulsive noise towards their neighbouring areas. In digital image processing, this problem is solved by median filtering instead of averaging. But obtaining the median value of a 3×3-pixels neighbourhood —which is the pixel brightness that has equal number of neighbouring pixel values above than below itself— is a computational task with a not very regular flow, what is no sense good for its implementation in a highly-parallel architecture.

Therefore, within the CNN framework, median filtering must be emulated by an alternative algorithm. For instance, we can prevent correct pixels form receiving

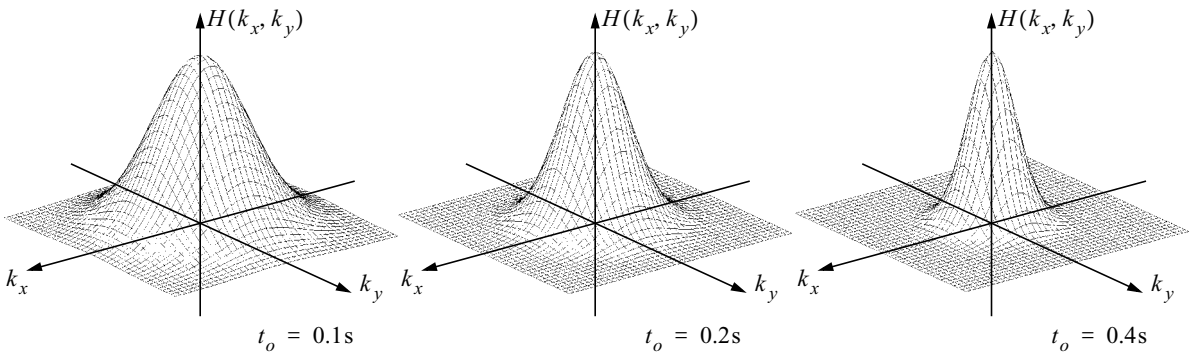


FIGURE 1.31. Spatial Fourier transform of the Gaussian function $h(x, y)$ for different values of t_o. $D = 0.01\text{m}^2/\text{s}$ in all cases.

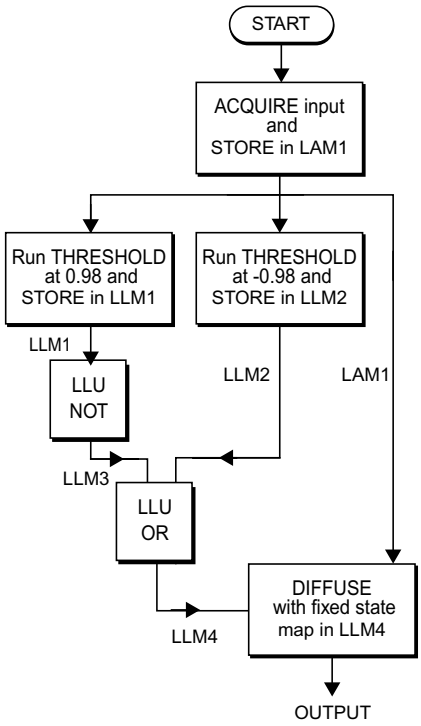


FIGURE 1.32. Flowchart for restoration from an impulse corrupted image with the CNNUM.

any influence from the corrupted pixels in their neighbourhood. At the same time, the impulses can be by an extrapolated correct value obtained by local averaging of the neighborhood—assuming the strong correlation in space that can be observed in the majority of natural images [Gonz87]. This can be accomplished with the help of the FSM feature. Let us consider the flowchart depicted in Fig. 1.32. Beforehand, the corrupted image (Fig. 1.33(a)) is acquired and stored in an analog memory (LAM1). This image suffers 3 different transformations. First, it is thresholded at 0.98 —i. e. pixels with brightness above a 99% of the full scale are detected—generating binary mask LLM1, which then is inverted to have LLM3 (Fig. 1.33(c)). This is realized by the set of templates depicted in the second row of Table 1.9:

Operation	A	B	z
Diffusion (spatial frequency lowpass filter)	$\begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & -2.0 & 0.50 \\ 0.25 & 0.5 & 0.25 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	0.0
Threshold at value z	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	z

TABLE 1.9. Examples of basic operations performed by linear template sets.

Then, the input image (LAM1) is thresholded at –0.98 —i. e. detection of pixels with brightness equal or under 1% of the full scale— creating mask LLM2 (Fig. 1.33(d)). After that LLM3 and LLM2 are composed (by a logic OR) to generate the binary mask LLM4 (Fig. 1.33(e)). This binary mask contains the position of all the possibly corrupted pixels. The final step is to allow diffusion of the image contents, this is local averaging, only to those marked pixels. The effect is that corrupted pixels are substituted by their averaged neighbouring pixel values while the correct pixels remain unchanged. Compare the result (Fig. 1.33(f)) with the original image (Fig. 1.9(a)).

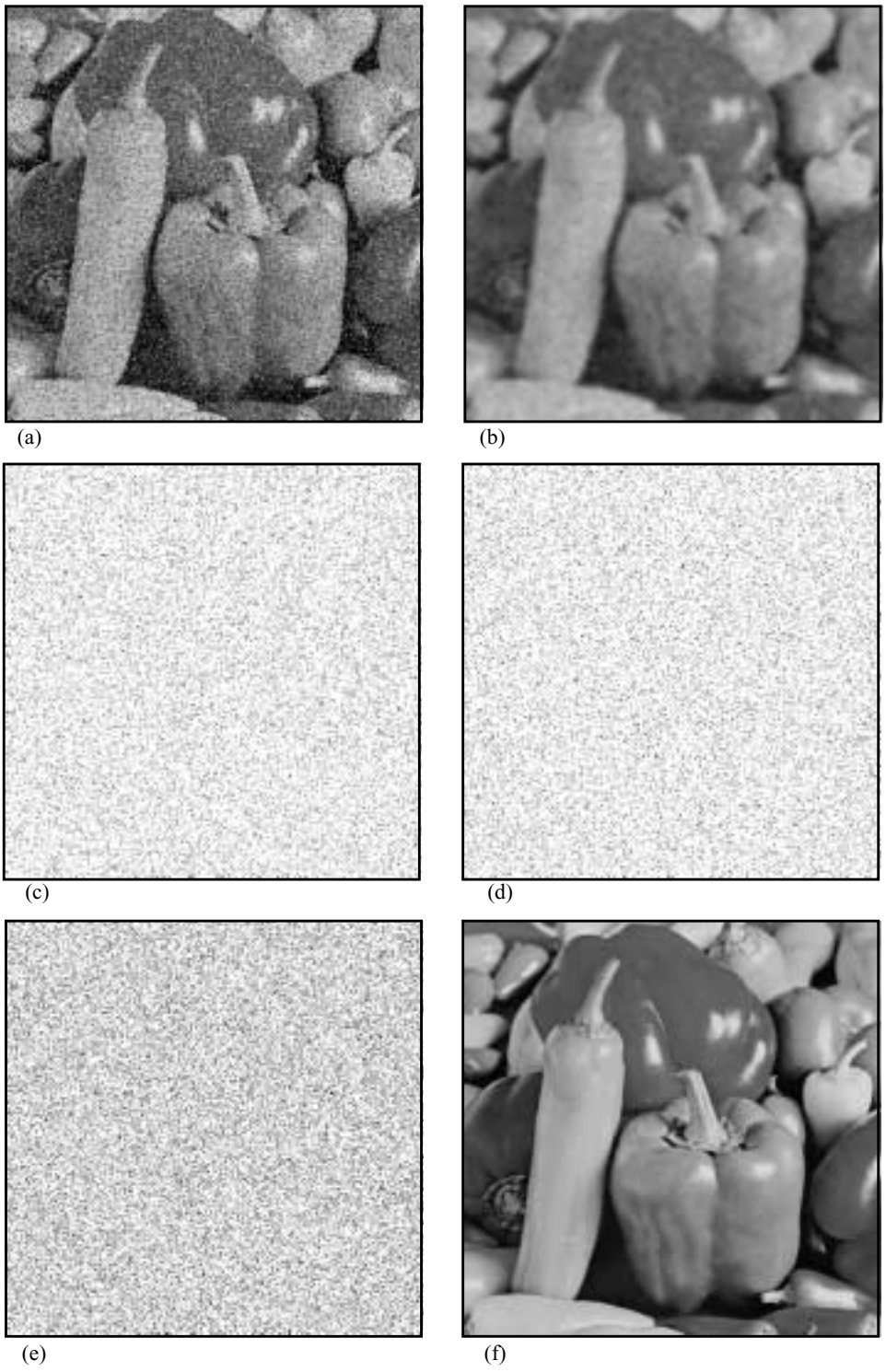


FIGURE 1.33. Grayscale image processing example: image restoration.

2 Analysis and Design of Parallel Processing Architectures based on CNNs

In the previous chapter, the Cellular Neural Network framework was outlined together with the comprehensive capabilities of the CNN Universal Machine. The major difficulties found in VLSI implementation of massively parallel analog array processors are of a technological nature. Two different phenomena pose limits on the maximum achievable cell density and the chip size, respectively. Analog accuracy, on one side, which is strongly related with the influence of the parametric yield [Pelg89], demands for larger devices. Fabrication yield, on the other, derived form the apparition of random defects [Kore89], pushes for reduced chip sizes. Their combined action drops the maximum number of cells that can be integrated together in the same chip. It represents an impediment for VLSI implementation of CNN-based processing for practical applications. In fact, the maximum number of cells per chip is certainly below the current available number of pixels in video camera CMOS sensor arrays [Yang99]. Of course, the little processing abilities that are commonly present in regular CMOS cameras at the moment are quite below the tremendous capabilities offered by the implementation of the CNN Universal Machine concurrently with the photosensing circuitry. A consequence of this is that the pixel size in a CNNUM chip —actually the cell size, of which the photosensor is not a large fraction— is definitely larger than that of the simple photodevice plus follower found in APS (Active Pixel Sensor) cameras [Seit99]. Thus a reduced fill-factor must be expected for chips incorporating a considerable amount of processing within the sensors. Charge-Coupled Devices (CCDs) are intentionally excluded from this discussion because of their reduced on-chip processing features. In these conditions, a careful architectural analysis is required for the application of the CNN framework to the solution of practical problems. In this chapter we will start by realizing the influence of fabrication yield and process parameter deviation on the CNN chip size. After that we will review a generic hardware architecture defined by Professor Tamás Roska, the CNN chip set [Rosk96], which can be seen as a suitable environment for the CNNUM. This architecture reproduces the

internal dual-computing nature of the CNNUM at the system level and interfaces this analogic array processor with the sensory devices, the video sources, short-, middle- and long-term memory devices and a conventional digital host computer. In addition, we will specify the timing constraints for each component of the CNN chip set and will point out the characteristics of the non-standard parts.

2. 1. VLSI technology limitations on image size

The integrated circuit fabrication process is exposed to a number of factors that can seriously compromise the production yield. On one side, random defects appear that can result in fabrication faults, because of a limit in the achievable degree of cleanliness of the process. This phenomenon, random defect incidence, reduces the so-called random yield and can be characterized by: the density of defects, their clustering properties and the probability of a random defect of becoming a fatal defect, as we will see later on. It will be concluded that the smaller the die area, the smaller the risk of suffering from fatal defects in fabrication, and thus, the larger the attainable random yield. On the other side, there will be the statistical nature of the IC fabrication process parameters. These physical parameters are affected by deviations of both deterministic and random sources, and so is the operation of the devices which, in principle, have been designed for the nominal values of this set of parameters. This deviation is the cause of a considerable accuracy loss in analog VLSI signal processing. Actually, the operation of many building blocks in VLSI analog signal processing counts on the cancellation of non-idealities between two equally-designed devices. A significant disparity between the values of process parameters for these devices results in a mismatch between their electrical characteristics, and hence, a defective cancellation of some non-idealities and second order effects. A parametric yield can be defined to quantify the influence of process parameter deviation on the final circuit performance. It can be concluded that for a higher accuracy, larger devices must be designed, resulting in a reduced cell density for the implementation of CNNs.

Both, random and parametric yield, limit the number of cells that can be integrated in a chip, reducing the applicability of CNN processing to VLSI solutions for practical problems. CNNUM chips of up to 64×64 processing cells have been reported [Liña98] while, for instance, conventional television employs around 640×480 pixels per frame. Consequently, fully-parallel image processing is still quite far from being implemented. In these conditions, assuming that a one-to-one mapping of the image plane pixels to the CNN processor array elements is impracticable, the proposed solutions necessarily involve either the replication of hardware to implement larger image sizes, or the time-multiplexing of a smaller processing core.

2. 1. 1. Random yield constraints on the chip size

Random defects in IC fabrication

One of the major sources of production yield loss in IC fabrication is the incidence of random defects. A defect is a local disturbance of the prescribed silicon structures mainly caused by particles of dust, contamination of the fabrication equipment or variations in the fabrication process, like loss of calibration, etc. The incidence of spot defects, so referred because of their local scope, is of a random nature. Its occurrence is characterized by an stochastic size and frequency per unit area, also referred as defect density, D. However, defect density is an ill-defined magnitude inferred by yield measurements. Interactions between defects and device structures cause faults to occur, thus reducing the production yield. Basically, yield decreases because of fatal defects —i. e. faults— and defects are fatal if they happen to be where they can cause yield degradation [Kore89]. Yield estimations based on the incidence of random defects in fabrication constitute the so-called random yield. It must be distinguished from the parametric yield, this is, the predicted production yield due to process parameters deviation, which will be reviewed later.

Nevertheless, in spite of the weak definition for defect density, widely accepted models for random yield prediction have been developed, such as the Poisson formula [Warn74]:

$$Y_{\text{random}}(A_{\text{chip}}) = \exp(-DA_{\text{chip}}) \tag{2.1}$$

where D is the defect density and A_{chip} stands for the die area.

Assuming a different statistical distribution of defects, we find the *negative binomial yield* formula [Stap91],

$$Y_{\text{random}}(A_{\text{chip}}) = \left(1 + \phi\frac{DA_{\text{chip}}}{\alpha}\right)^{-\alpha} \tag{2.2}$$

where D is the defect density, A_{chip} the chip area, α is a defect clustering factor, and ϕ stands for the probability of a defect to become a fatal defect. Both D and α are process dependent, while A_{chip} and ϕ depends on the design. Actually, the probability ϕ depends highly on the distribution and magnitude of critical areas in the circuit layout —i. e. those areas in which the interaction with a spot defect surely result in a fault. In a sense, this parameter also depends on the fabrication process. Each process has a set of design rules which establish minimum widths for the metal tracks and polygons made up of different materials, as well as the minimum distances between them. Shrinking rules increase the critical areas of the layout, unless the average defect size decreases jointly with the technology minimum feature size. Otherwise, parameter ϕ increases. For a dense layout, which unfortunately is exactly the case in analog VLSI CNN implementation, this factor can be extremely large (more than 10% of the circuit layout being critical for the average spot defect size). This results in a considera-

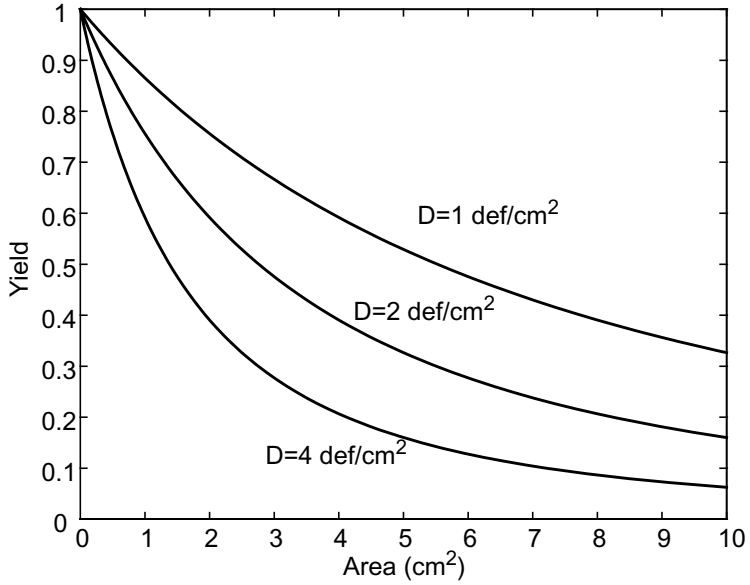


FIGURE 2.1. Predicted fabrication yield vs. die area for several random defects densities

ble yield loss predicted by the negative binomial model for large die areas. Fig. 2.1 displays the estimated yield for several defect densities. A defect clustering factor of $\alpha = 2$ and a probability of failure of $\phi = 0.15$ (15%) have been considered. For a process in which $D = 2$ def/cm^2, predicted yield is below 50% if the chip area exceeds 2.75 cm^2. In other words, less than half of the manufactured samples will pass functional tests in the conditions referred above. It is important to remind that random yield effects apply for both analog and digital VLSI circuits, while parametric yield loss will affect more strongly to analog circuits.

Chip area and cell number bounds due to random yield

Therefore, in order to achieve a reasonable fabrication yield, the chip size must be constrained to a certain limit, which can be derived from the above formulae. Using the negative binomial yield:

$$A_{\text{chip}} \leq \frac{\alpha}{D\phi}\left(\frac{1}{\sqrt[\alpha]{Y_{\text{random}}}} - 1\right) \tag{2.3}$$

An upper bound in the number of cells that can be safely integrated appears. For example, consider a CNN array composed by a total of N cells, each one of them occupying an area of silicon A_{cell}. If we consider that the analog references, digital control and interface circuits represent only 10% of the total die area —what is usual in CNN chips [Espe94a]— we find that $A_{\text{chip}} = NA_{\text{cell}}/0.9$, and thus the maximum number of cells in the CNN chip is fixed by the targeted production yield:

$$N \leq \frac{0.9\alpha}{A_{\text{cell}}D\phi}\left(\frac{1}{\sqrt[\alpha]{Y_{\text{targeted}}}} - 1\right) \tag{2.4}$$

2. 1. 2. Process parameter scattering

Another important source for yield loss in IC fabrication is found in the deviation of the process parameters from the nominal values handled by the designer. As in any other natural[i] process, the parameters that characterize integrated circuit technologies vary from one fabrication run to another, even if the instrumentation and machinery involved are adjusted every time to the same nominal values. Differences can be observed as well from one wafer to another, within the same fabrication batch, and even from one chip to another inside the same wafer. The electrical parameters that characterize the operation and behaviour of the different primitives found in IC technologies also suffer also from a deviation similar to that of the lower-level physical parameters that characterize the fabrication process. This is because these parameters —e. g. threshold voltage, intrinsic transconductance of MOS transistors— are derived from the lower-level parameters that characterize the process, like the oxide thickness, substrate doping, etc.

The influence of these variations over the final circuit performance is twofold. On one side, variations affecting equally to every device on the same chip, like batch-to-batch, wafer-to-wafer and chip-to-chip deviations, result in average parameter values which fail to coincide with the nominal values considered during the design phase. On the other, the granularity of the fabrication process across the die, i. e. the unevenness of the parameter surfaces, causes equally designed devices to show different electrical properties. This impairity between supposedly matched devices is known as parameters mismatch.

The effect of parameters mismatch in the circuit operation and its influence on the final chip size will be covered in the next section. Let us consider now the deviations in the parameter values that can be appreciated at a larger scale.

Large-scale variations of the process parameters

Because of unwanted offsets and fluctuations, different average values for the parameters can be detected in each fabrication run. Moreover, these differences can be observed also from wafer to wafer in the same fabrication batch and even between chips belonging to the same wafer (Fig. 2.2). Consider a set of N parameters which have been adjusted to their nominal values for the corresponding process. We can see it as a point in the parameters space (that is R^N), denoted by vector $\mathbf{p}_0$, which represents that particular set of nominal values. For each die in a specific wafer of a particular fabrication batch, there is a vector $\mathbf{p}$, composed by the actual average values for the N parameters in that particular chip, which usually deviates from $\mathbf{p}_0$ defining an error vector $\Delta\mathbf{p} = \mathbf{p}_0 - \mathbf{p}$. It is possible to define a hyper-cube centred around $\mathbf{p}_0$ that contains the vast majority of points $\mathbf{p}$ representing the average values for the process

i. meaning *real* as opposite of *virtual*.

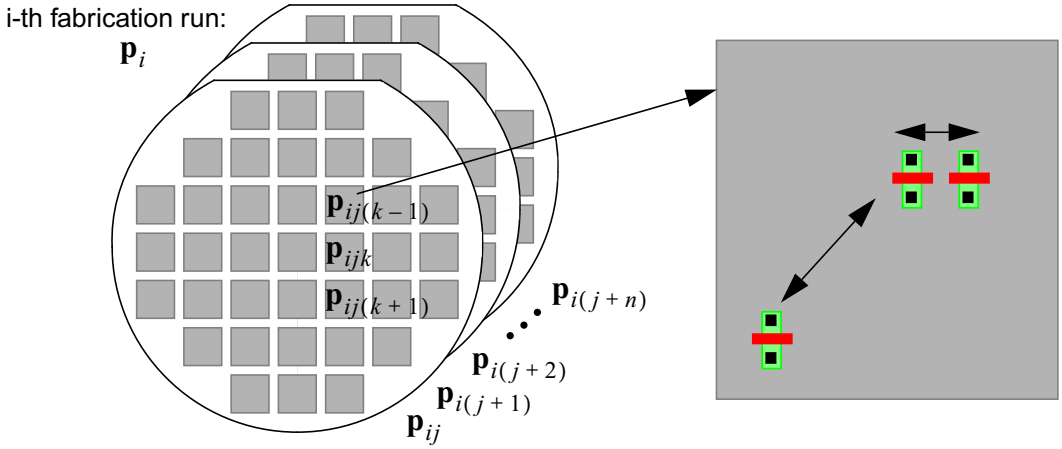


FIGURE 2.2. Illustrating batch-to-batch, wafer-to-wafer and chip-to-chip deviations of parameters $\mathbf{p}$.

parameters for different chips of different wafers from different runs. Think that although variations can have a strong tendency to modify some of the parameters in a particular direction in the parameters space, there are calibration cycles and process tuning protocols that periodically bring back the nominal parameters vector $\mathbf{p}$ to a position closer to the centre of the above-mentioned hyper-cube. The corners of this hyper-cube are the so-called technology worst cases. It is important to the designer to know, on one hand, the boundaries of this hyper-cube, and on the other hand, the sensitivity of the circuit features to each specific parameter, in order to devise a circuit that is robust enough to maintain the required performance levels for extreme cases of the process parameter average values. Suppose a performance index, F, of an analog building block, that depends on the value of two uncorrelated process parameters p_x and p_y. This is, $F = F(p_x, p_y)$, the particular functional form depending on the selected circuit primitives and circuit topology. Consider that absolute deviations of F beyond a prescribed ΔF will be labelled as defective. Therefore:

$$\left|S_{p_x}^F\right| \cdot \left|\frac{\Delta p_x}{p_x}\right| + \left|S_{p_y}^F\right| \cdot \left|\frac{\Delta p_y}{p_y}\right| \leq \left|\frac{\Delta F}{F}\right| \tag{2.5}$$

must hold for any value of p_x and p_y [Scha90]. Hence, the sensitivities of F with respect to the process parameters, $S_{p_x}^F$ and $S_{p_y}^F$, must be reduced to an extent delimited by the maximum relative error in the implementation of p_x and p_y. Therefore, technology worst cases must be a well established magnitude, normally warranted by the silicon vendor.

Amongst the causes for the deviation of the process parameters from their nominal values is possible to find both deterministic and random phenomena. On one side, offsets are introduced by instruments aging, temperature drift during operation and similar circumstances. On the other, microscopic scale fluctuations, like thermal agitation of molecules in a crystal lattice, are responsible for the random component of these variations. However —and this is a piece of reasoning that will be repeatedly alleged along the discussion— from the designer's point of view, both contribution to $\Delta\mathbf{p}$ can be considered purely random, as she does not know a priori which the status of the involved instrumentation and machinery will be by

the time the chip is being fabricated. Thus, because of the nature of the contributions to the final value of each component p_i of vector $\mathbf{p}$, it can be considered that each p_i follows a normal distribution around p_{0_i}. The referred worst cases, are thus defined by $p_{0_i} \pm 3\sigma(p_i)$.

In order to illustrate how large these deviations can be, let us consider the data extracted from the characterization files of one of the processes (CMOS $0.35\mu m$) offered by a a low cost prototyping and small-volume production service (MOSIS). These data files correspond to 11 different fabrication runs conducted during a period of two years. There is no information about the different wafers on each run, only the global batch characterization is given. However, it seems reasonable to think that batch-to-batch imparities are much larger than the differences that will be observed between wafer-bound average values in the same run, or chip-bound measurements inside the same wafer. Hence, for the thin oxide thickness, t_{ox}, a characteristic process parameter in CMOS technology [Sing94], it can be found that during the referred two-year long period t_{ox} has been oscillating around an average value of $\overline{t_{ox}} = 78.6\text{Å}$, with a standard deviation of $\sigma(t_{ox}) = 1.4\text{Å}$, this is 1.7% of the average value. This means that 66% of the chip samples will deviate $\pm 1.7\%$, or 99% of the samples has an error contained in $\pm 5.1\%$ of the average. Also for the same process, the average value for the threshold voltage, an electrical parameter derived from a set of process parameters, of a minimum sized n-channel MOS transistor has been $\overline{V_{th}} = 0.637\text{V}$ with a standard deviation of $\sigma(V_{th}) = 32\text{mV}$, the 5% of the nominal value.

These common deviations from the nominal values can seriously compromise circuit performance and operation, but no direct relation can be established between them and the chip size. In other words, the actuation over chip dimensions does not seem to mitigate the effect of these deviations. Let us now consider the mismatch observed in the parameters of supposedly identical devices. We will see that a connection between parameters mismatch and the active area of the primitive devices, and thus the final chip size, is confirmed.

Device mismatch due to process parameter deviation

In the previous section, we have seen that parameters characterizing circuit operation vary from batch to batch, from wafer to wafer and from chip to chip. Furthermore, two primitive devices designed to be identical, inside the same die, very rarely display identical characteristics, because of fluctuations of the parameters at scales smaller that the chip size. These chip-bound variations of the process parameters have a detrimental effect on circuit performance. Bear in mind that the operation of an important subset of linear and nonlinear analog building blocks counts on the cancellation of nonidealities between two ideally matched devices. For instance, consider a pair of MOS transistors constituting a differential pair (Fig. 2.3(a)) [Geig90]. If the differential input voltage is defined as $V_i = V_1 - V_2$, and the output current is computed from $I_o = I_1 - I_2$, then the following expression can be derived if both transistors, M_1 and

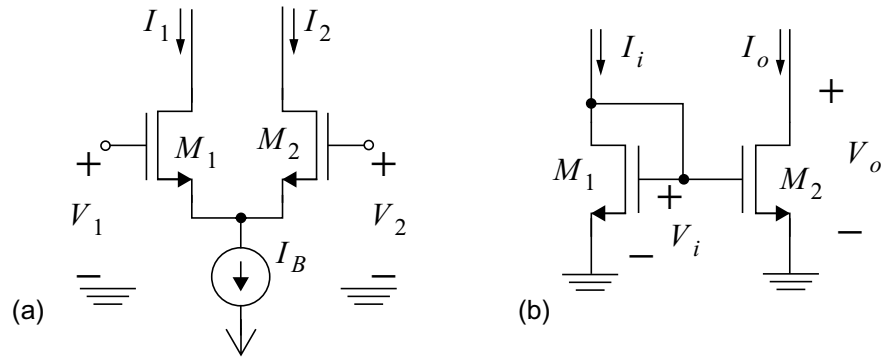


FIGURE 2.3. (a) MOS differential pair and (b) MOS current mirror.

M_2 are considered to be perfectly matched (this is, having equal transconductance parameters, $\beta_1 = \beta_2 \equiv \beta$, and threshold voltages $V_{T_1} = V_{T_2} \equiv V_T$[ii]):

$$I_o = \beta V_i \sqrt{2I_B/\beta - V_i^2} \tag{2.6}$$

as long as the differential input voltage remains $|V_i| < \sqrt{I_B/\beta}$, otherwise the differential pair saturates to either I_B or $-I_B$. If a certain amount of mismatch in the values of the threshold voltages, $V_{T_1} - V_{T_2} = \Delta V_T$, and the current factor, $\beta_1 - \beta_2 = \Delta\beta$, between both devices is taken into account, Eq. 2.6 transforms into:

$$I_o \approx \beta(V_i + \Delta V_T)\sqrt{\frac{2I_B}{\beta} - \left(1 + \frac{\Delta\beta}{\beta}\right)(V_i + \Delta V_T)^2} \tag{2.7}$$

where it can be seen, easily, that an offset current has been introduced ($I_o(V_i = 0) \neq 0$). Another example of an analog building block, whose operation is degraded by the unavoidable presence of mismatch between originally identical devices, is the current mirror in Fig. 2.3(b). Ideally, and because M_1 and M_2 are considered to have equal parameters ($\beta_1 = \beta_2 \equiv \beta$ and $V_{T_1} = V_{T_2} \equiv V_T$), the output current matches the input current, $I_o = I_i$, as long as both transistors operate in the saturation region $V_o \geq V_i - V_T$. But if a slight mismatch is considered between the parameters of both devices, the output current differs from I_i [Lake94]:

$$I_o \approx I_i\left(1 - \frac{\Delta\beta}{\beta} - \frac{2\Delta V_T}{V_i - V_T}\right) \tag{2.8}$$

Therefore, a relation can be established between the amount of correctly operating chip samples of a particular design and the predicted parameters mismatch for a specific technology. This is the so-called parametric yield. For instance, in order to visualize the influence of parameters mismatch in the operation of a CNN-based processor, let us consider the CMOS OTA-based CNN chip in [Cruz92]. It has been modelled and simulated with SIRENA [Carm99a] (see the detailed descrip-

ii. We are not considering here the body-effect constant, which can be the object of an important mismatch.

tion of this model in Chapter 3). Monte Carlo analysis over 30 samples of the network have been realized, considering that the mismatch in the threshold voltage, ΔV_T, between supposedly identical devices —constituting differential pairs and current mirrors— follows a normal distribution with different standard deviations. For each analysis, the associated yield has been computed as the fraction of correctly operating chips. Fig. 2.4 plots the corresponding yield figures versus the standard deviation of the threshold mismatch. This function can be roughly approximated by:

$$Y_{\text{param}}(\sigma) = \frac{1}{2}\{1 - \tanh[m_0(\sigma - \sigma_0)]\} \tag{2.9}$$

where σ_0 and m_0, represent the abscissa and the slope of the curve in the inflexion point and have been estimated by nonlinear least square fitting of the Monte Carlo simulation outputs.

Parametric yield limits to cell density

Once yield considerations associated with process parameters deviation have been theoretically established, let us see how this relates with the total chip area. The model proposed by Pelgromm et al. in [Pelg89] links the mismatch between the parameters of two ideally matched devices with the area, orientation and distance between both devices. This model is valid as long as the processes that generate a particular parameter value affect the entire device area, i. e. these processes are driven by area-related physical causes. For instance, the threshold voltage in a MOS transistor depends on the substrate doping under the channel area, thus mis-

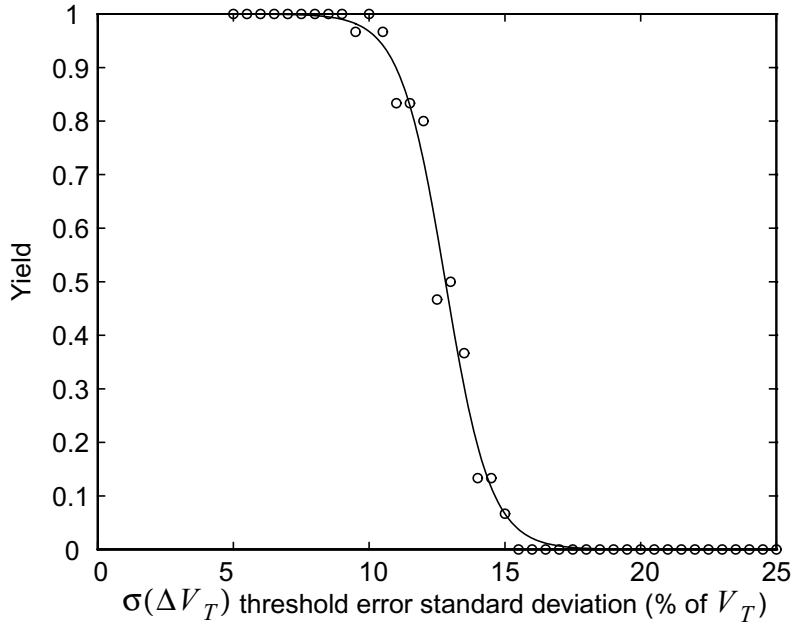


FIGURE 2.4. Percentage of correct samples vs. threshold voltage mismatch deviation.

matches in V_T will be covered by this model. On the contrary, the Early voltage depends only on processes affecting the transistor length, therefore Pelgromm's model is not applicable to Early voltage mismatch. The fluctuations of the process parameters reflected by this model are random and can have a short correlation distance —similar to device dimensions— or a long correlation distance effects, like gradients across the die, that have a deterministic cause but remain virtually random to the designer. It is important to notice that large scale variations of the process parameters, like batch-to-batch, wafer-to-wafer and chip-to-chip imparities have been excluded from these considerations. Only differences between devices inside the same chip are considered. Then, because of these local (meaning intra-die) random fluctuations of the process parameters, the value for one particular parameter P of a primitive device oscillates around an average value following a normal distribution. Consider two ideally matched devices. Although they where nominally thought to implement P_0, they actually score P_1 and P_2 for parameter P. If both devices are rectangular, have the same orientation and the parameter P satisfies the requirements for the application of Pelgromm's model, the variance of the mismatch in the implementation of P, given by $\Delta P = P_1 - P_2$, is determined by:

$$\sigma^2(\Delta P) = \frac{A_P^2}{WL} + S_P^2 D \tag{2.10}$$

where, W and L are the nominal width and length of the devices, D is the distance between them and A_P and S_P are process-related constants. Corrections to this model can be made in order to account for border effects. But, the capital consequence that can be extracted from this model is the relation between circuit performance and size. Usually the operation of the analog building blocks depends on matching of devices that occupy relatively close positions inside the chip. Therefore, long correlation distance effects can be neglected and only the area-dependent term in Eq. 2.10 counts. If the total chip size, A_{chip}, is a multiple of the elementary devices area, WL, it can be said that $\sigma(\Delta P) \propto 1/\sqrt{A_{\text{chip}}}$, and thus an approximated expression for the parametric yield as a function of the total chip area can be:

$$Y_{\text{param}}(A_{\text{chip}}) = \frac{1}{2}\left\{1 - \tanh\left[k_0\left(\frac{1}{\sqrt{A_{\text{chip}}}} - \frac{1}{\sqrt{A_0}}\right)\right]\right\} \tag{2.11}$$

where k_0 and A_0 are derived from the adjusted parameters σ_0 and m_0 in Eq. 2.9, the technology-dependent parameter A_P, and the relation between A_{chip} and WL.

2. 1. 3. Yield estimation inside the design cycle

Parametric yield modelling in circuit simulation

Considering the impact that parameters scattering will have on the system performance, it seems reasonable to include statistical characterization of the circuit into the design cycle. Obviously, fatal defects are a major cause for system failure, but it make no sense to characterize it at the circuit level because there is not a soft degradation of the system performance because of this. Suppose a circuit affected only by random yield, it would either malfunction if there is a defect or perfectly meet the electrical test specifications if not. Therefore it will be a issue to be considered at the wafer scale. On the contrary, an estimation of the expected parametric yield for a particular circuit implementation can be easily realized by simulation. These estimations can be employed to gauge different circuit alternatives, or to realize design centering, etc.

The characterization of the system tolerances, or, in other words, the evaluation of the robustness of a circuit implementation against process parameter deviation, is basically done in two steps. A fast analysis that considers technology corners, and a computation-intensive evaluation based on Monte Carlo analysis. A CMOS fabrication process is characterized by a set of physically measurable model parameters. These parameters are also known as skew parameters, as they usually follow a normal distribution which is not exactly symmetrical, but skewed to one end. Typical parameters in CMOS technology are [Meta88]: XL, the polysilicon layer critical dimension (CD), representing the difference between the drawn and the actual sizes, XW_n and XW_p, the active layer CD, TOX, gate oxide thickness, RSH_n and RSH_p, active layer resistivity, and DELVTO_n and DELVTO_p, representing the threshold voltage variation. These typical skew parameters are independent of each other, so the worst cases of the technology are derived from moving all of them in a direction in which their effects add up. Following this criterion, technology corners can be found. For instance, fast and slow processes are a common denomination for CMOS technology corners. The parameters corresponding to a fast process are obtained by substracting the maximum value for XL, RSH, DELVTO and TOX from the NMOS transistor model and XL, RSH and TOX from the PMOS model, while adding the maximum value for XW in the NMOS case and XW and DELVTO in the PMOS model. The converse operation is done to obtain the slow case parameters from the skew parameters model.

Therefore, as a quick estimation of the implementation robustness, simulation of the circuit in the technology corners is mandatory for validating the design. But still a detailed statistical characterization of the effect of parameter scattering is lacking. This is accomplished by Monte Carlo analysis. It consists in the generation of random values, following a particular distribution, for the device model parameters, thus mimicking what happens in the fabrication process. These parameters can be programmed to vary globally from the nominal values and locally, causing

mismatch between equally designed devices to appear. Then, circuit simulation is iterated, and on each iteration model parameters are computed again. Performance indexes can be defined and statistically treated from the simulation outputs. The approach to Monte Carlo analysis can be either physical or electrical, depending on whether the parameters selected to be randomly generated are the true low-level skew parameters referred above, or some derived magnitudes obtained from them. We will see later in this work how Monte Carlo analysis can be employed in electrical parameter tuning for yield optimization, that is design centering.

Optimum chip size: combined estimation of random and parametric yield

The dependence of both yield estimations —the one based on the characterized incidence of random defects and the other owed to the statistical nature of the process parameters— on the chip area is given by Eqs. (2.2) and (2.11). It can be seen that an increase (or alternatively a decrease) in the total chip area has conflicting effects in those yield estimations. On one side, the larger the chip, the more probable is a fatal defect to occur, thus the worse the random yield figures. On the other side, the larger the analog circuit primitives the more accurate they are, thus the larger the chip the better parametric yield estimations become. Added to this, a failure due to a fatal defect appearance and a failure due to parameters mismatch are two completely uncorrelated phenomena. Therefore, if we define F_{random} and F_{param} as the probabilities of each failure type to occur, the probability of having a failing chip sample because of, at least, one of these causes is given by:

$$F_{\text{total}} = F_{\text{random}} + F_{\text{param}} - F_{\text{random}}F_{\text{param}} \tag{2.12}$$

As the total yield is defined as the probability of having a correctly working sample, it can be stated that:

$$\begin{aligned} Y_{\text{total}} &= 1 - F_{\text{total}} = 1 + F_{\text{random}}F_{\text{param}} - F_{\text{random}} - F_{\text{param}} = \\ &= (1 - F_{\text{random}})(1 - F_{\text{param}}) = Y_{\text{random}}Y_{\text{param}} \end{aligned} \tag{2.13}$$

In this way, a global yield estimation, as a function of the total chip area, is obtained by multiplying the two yield factors, Y_{random} and Y_{param}. Thus, from Eqs. (2.2) and (2.11):

$$Y_{\text{total}} = \frac{1}{2}\left(1 + \phi\frac{D \cdot A_{\text{chip}}}{\alpha}\right)^{-\alpha}\left\{1 - \tanh\left[k_0\left(\frac{1}{\sqrt{A_{\text{chip}}}} - \frac{1}{\sqrt{A_0}}\right)\right]\right\} \tag{2.14}$$

Because of the antagonistic dependence of Y_{random} and Y_{param} on the total chip area, an optimum chip size can be found for a particular design, either by deriving the previous formula and equating the result to zero, in order to find the function extremes, or by graphical or numerical optimization of the combined yield formula. For the particular case employing the data of random defect incidence referred before, and the tables in

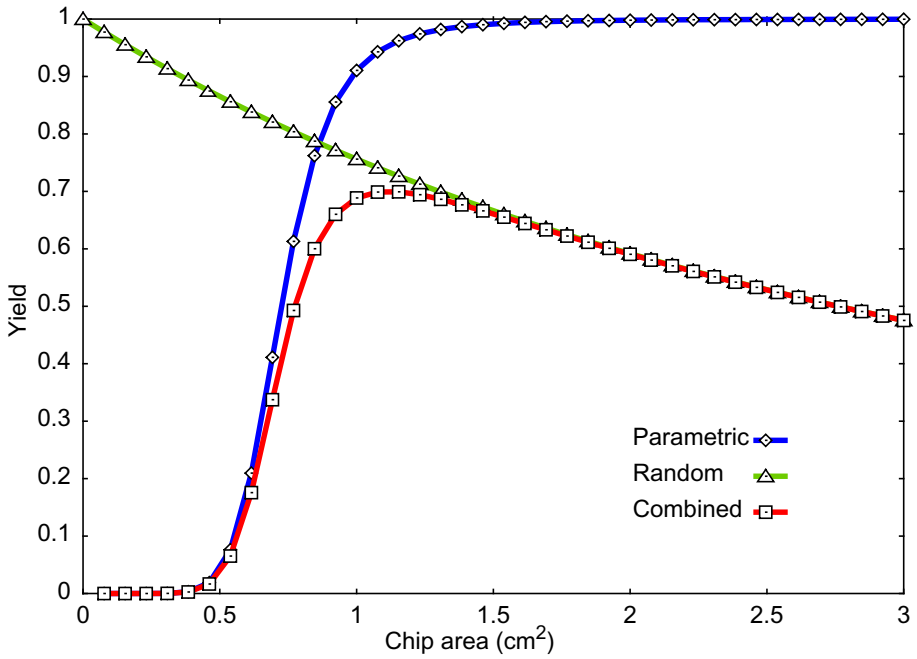


FIGURE 2.5. Combined random and parametric yield estimation vs. chip size.

[Pelg89], the random, parametric and combined yield predictions in Fig. 2.5 are obtained. The maximum combined yield found in this case is 69.96%, which occurs for an optimum chip size of 1.175cm^2. This is way beyond what is expected for simple analog circuits with restricted flexibility and a closed functionality, but with respect to complex APAP chips, it is not uncommon to manage silicon real estate on the square centimetre range. In an academic context, some yield figures can be more than acceptable, but a careful examination of these results is of extreme importance in evaluating industrial production costs and marketing revenues in the development of new products.

2. 2. Multiplexion of CNN systems

In spite of technology limitations, analog VLSI chips have displayed a remarkable performance in fields like focal-plane and real-time image processing [Koch96]. In addition, the outstanding processing capabilities of the CNN Universal Machine architecture can only be naturally exhibited by dedicated analog ICs. For this reason, different system-level solutions have been proposed in order to overcome the referred technology limitations. Multiplexed architectures, either multichip or time-multiplexed systems, have been developed to employ available analog array processors for an optimal high-performance CNNUM implementation.

2. 2. 1. Multichip CNN systems

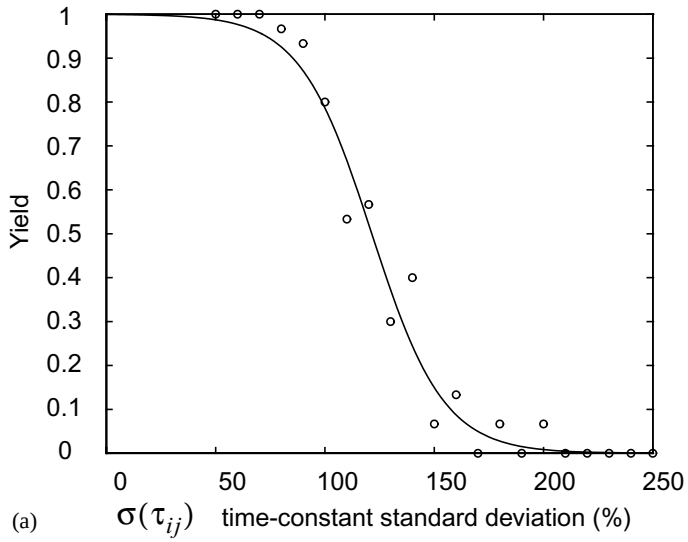
Parameters mismatch between modules

In a multichip CNN implementation, large arrays can be built by interconnecting chips with a smaller number of cells. Each module operates simultaneously onto a fraction of the input image which is, in this way, processed in parallel. Multichip processor arrays can accomplish an enormous computing power, due to their inherently parallel architecture. There are, however, some critical aspects in their design that deserve a careful analysis. In the first place, the influence of process parameter mismatch must be evaluated. Added to deviation and gradients within the same die, the averaged values of the fabrication process parameters vary from chip to chip, causing mismatch between different samples of the same circuit. Network parameters are, in this way, affected of a certain deviation from their nominal values in each of the modules incorporating the multichip CNN. Unmatched network parameters for the different subarrays can cause incorrect or inaccurate operation. As an example, a 32×32 -cells network, composed of 8×8 Chua-Yang's CNN modules, has been simulated under different circumstances. No intra-chip deviations of the network parameters has been considered this time. First of all, the time-constant of each subarray, $S_{(i,\,j)}$, has been deviated from its nominal value ($1\mu s$) following a normal distribution. In the case of non-propagating templates, like the edge detector, no significant degradation of the system performance is observed. On the contrary, a large mismatch on τ leads to convergence to an erroneous final state in the case of propagating templates, *e. g.* connected components detector. Fig. 2.6(a) displays the yield, computed as the fraction of the multichip networks that operates correctly, as a function of the standard deviation in the time-constant of the building blocks. In a second experiment, template elements have been deviated from their nominal values for each subarray. A strong effect on the system performance has been detected (see Fig. 2.6(b) and Fig. 2.6(c)). Several strategies can be adopted to reduce the influence of parameter mismatch in multichip systems. For instance, synaptic weights re-design for a more robust implementation of the algorithm [Fold98], process-independent definition of the network parameters [Angu98] with a possible post-fabrication tuning of the modules, or implementation of an adaptive algorithm for correcting parameter deviation in the analog weight generation, as it is done in [Espe96b].

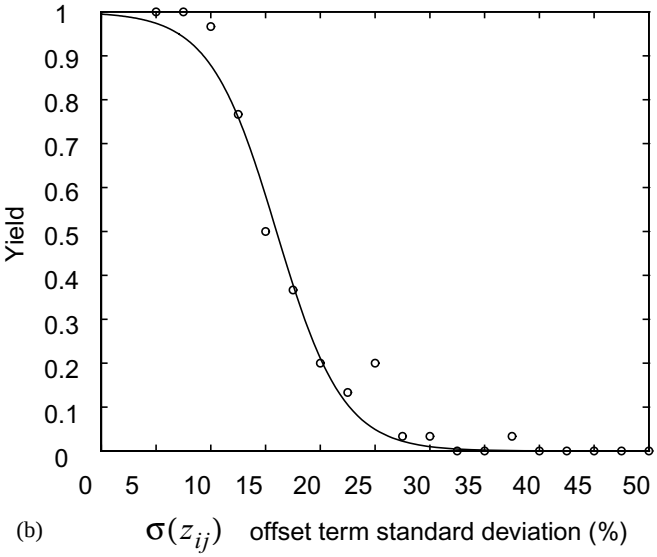
Cost-related drawbacks

Some other particular aspects of the multichip implementation have a significant impact in the system fabrication cost[iii]. In [Sale97] a prototype system composed of 720 digitally programmable cells is described. It contains 20 chips arranged in a 5×4 configuration, each one implement-

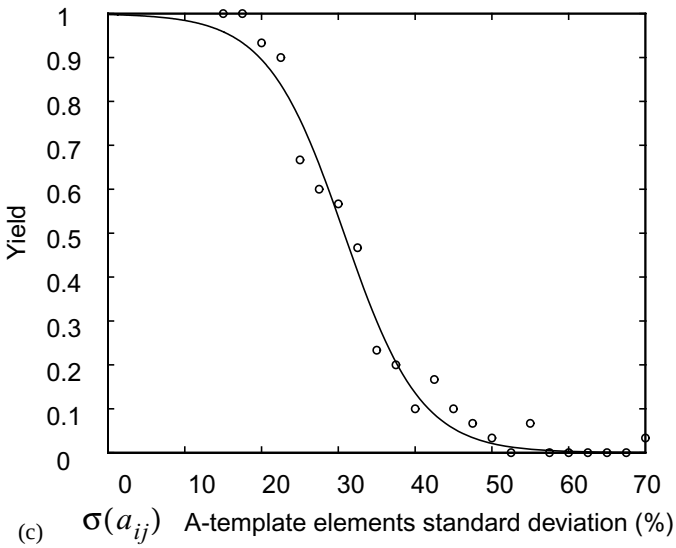
iii. Here, the fabrication cost is not strictly defined, instead, a qualitative analysis is realized based on different characteristics of the multichip implementation, e. g. number of chips, package type.



(a)



(b)



(c)

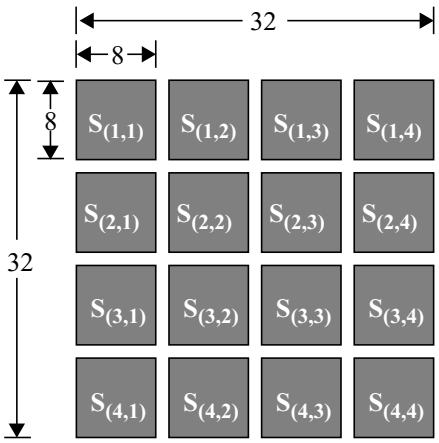


FIGURE 2.6. Yield vs.

(a) time-constant mismatch for connected component detection,

(b) offset term standard deviation, for an edge detector configuration,

(c) A-template elements standard deviation, for a connected comp. detector,

in the depicted 32×32 -cells multichip architecture.

ing a 6×6-cells CNN module. Experiments on this system are reported in [Tara98]. In order to process a 640×480-pixels video frame, more than 8×10^3 of these CNN chips will be required, 300 if the system is built by connecting 32×32-cells modules, 75 chips if 64×64-cells CNN modules are employed. Even using the larger CNN chips available, the implementation of a fully parallel processing system of a practical size requires a considerable number of modules. Building this multichip system can be justified by scientific research, though the associated fabrication cost renders infeasible mass production and low-end marketing.

Also related to manufacturing costs is the number of inter-chip connections required to achieve expandability of the system. System fabrication cost is affected by the selection of the chip package and mounting. Thus, on one side, not every chip package[iv] has the required amount of pins. On the other, each mounting technology[v] is characterized by a determined routing density, in other words, not every technology supports the required number of inter-chip connections. Consider the case of a CNN multichip implementation build from of $M \times N$-cells modules. Suppose that each cell generates contributions to neighbours, which is the common scheme in CNN monolithic implementation, and the contributions from the neighbourhood are summed up at the input node of the cell. Therefore, each $M \times N$ module will have $2(M + N - 2)$ input nodes, corresponding to the $2(M + N - 2)$ cells in the border of the array. Then, in order to contribute to the evolution of the border cells in the nearby modules, we have to generate $2(3M + 3N - 2)$ contributions. Each of them requires an output pin. This makes a total of $8(M + N - 1)$ pins, only for border cells interconnection (Fig. 2.7). A time-multiplexed scheme can be applied in some particular cases [Angu98], for instance, in spiking neurons networks or discrete-time CNNs —where the contribution to the neighbours are

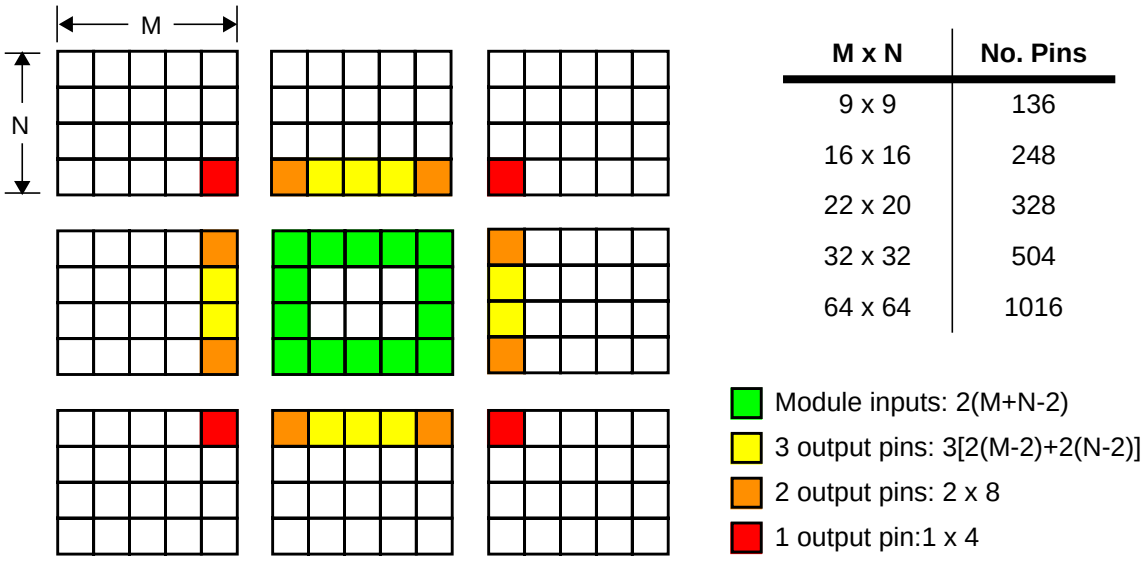


FIGURE 2.7. Conceptual floorplan of a modular architecture showing inter-chip connection needs

iv. e. g. Dual-in Parallel, Pin-Grid-Array, Ball-Grid-Array
v. e. g. Printed-Circuit-Board, Chip-On-Board, Multi-Chip-Modules

realized in discrete quantities or the network is settled at every timestep. Otherwise, the referred number of connections must be available. Therefore, a 640×480-cells CNN composed of 32×32-cells chips will require 504 pins per module dedicated only to inter-chip connections, and a total of 151.2×10^3 wires to support the CNN topology. In the case of a network made of 64×64-cells chips, the requirements are 1016 pins per module, as modules are bigger, and 76.2×10^3 wires for interconnection between them, as less modules are needed. Once again, this solution seems to be too expensive for some applications.

2. 2. 2. Time-division multiplexing of CNN chips

A different approach to the practical application of CNN chips to image processing is multiplexing in time. Making use of the extraordinary computing power of the CNN Universal Machine [Chua96], a single chip can process a whole video frame by operating on a fraction of the image at a time. A frame rate of 60Hz, which is adequate for high quality video applications [Poyn96], represents a data flow (for a 640×480 image size) of 18.4×10^6 pixels per second. Real-time processing of such a pixel rate demands 54ns processing time per pixel. A 32×32-cells CNN chip will be required to work faster than $55.3\mu\text{s}$ per fraction of the input. Processing time, including I/O, for a 64×64-cells chip must be less than $221\mu\text{s}$. Besides, there is a necessary overlapping between the input image pieces that has not been considered [Pine96]. Allowing a 2-pixel wide overlap, time requirements result in $43.3\mu\text{s}$ and $198\mu\text{s}$ per piece, for the 32×32-cells and 64×64-cells chips operation, respectively. These requirements do not seem to be a problem for a parallel array processor based on spatio-temporal dynamics characterized by a time-constant below the microsecond range. Therefore the time-multiplexed approach appears as a more feasible and cost-effective solution for the implementation of CNN image processing. The next step will be the definition and development of an appropriate chipset [Rosk96] to support multiplexion of the CNN processor. This chipset includes control of the operation, analog signals storage and an interface with the video sources and the digital environment.

The CNN chipset architecture

The CNN chipset has been conceived as a suitable hardware environment for the CNN Universal Chip [Rosk96]. It is composed of several analog, digital and mixed-signal integrated circuits. The function of the CNN chipset is, basically, interfacing the CNN processor to the sensors and the digital environment. Not only the signal format and codification can be different, but also information have to be re-ordered to be processed by the CNNUC in a block-by-block basis. In this context, the CNNUC can be seen a visual microprocessor with a reduced instruction set that operates onto analog and logic signals, controlled by digital instructions. Data transmission for interfacing the CNN processor to the

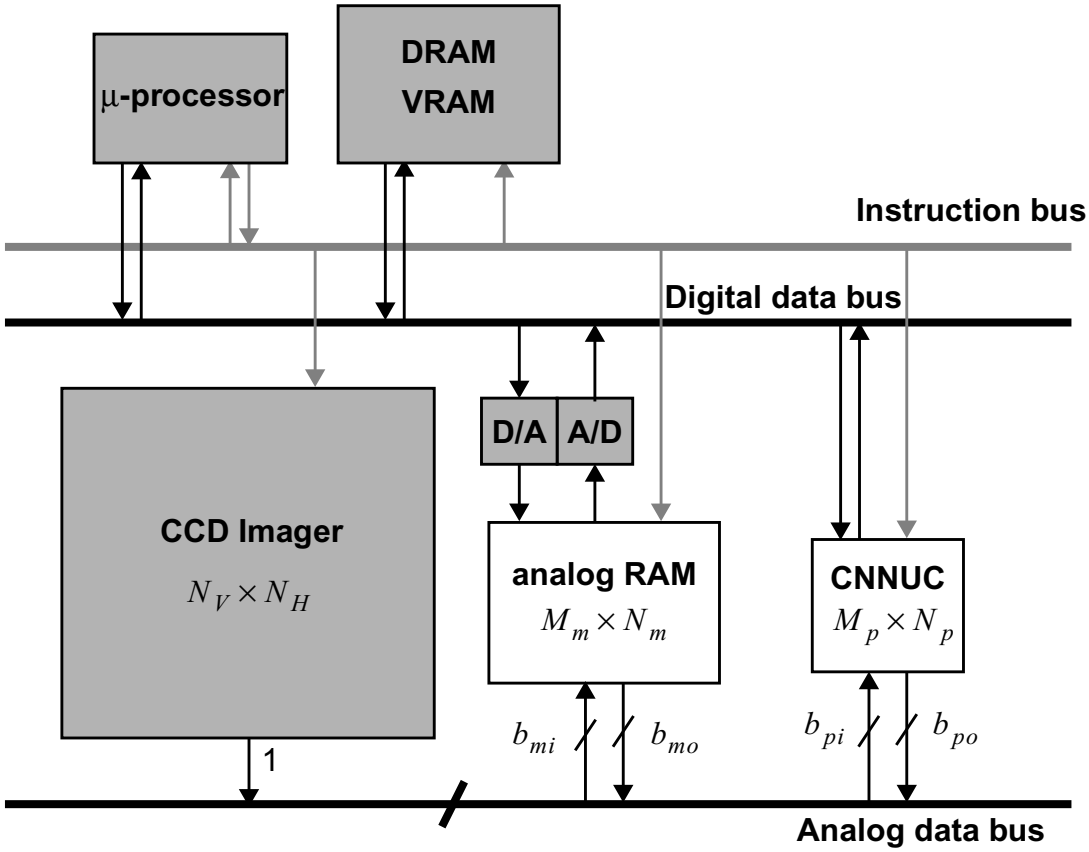


FIGURE 2.8. Diagram of the CNN chipset architecture.

sensory devices and the digital host circuitry is supported by three different buses: an analog data bus, a digital data bus and a digital instruction bus (Fig. 2.8). Connected to the digital data bus, the chipset contains a digital microprocessor, DRAM circuits and an interface, together with data converters, that bridge the analog and digital data buses. There is another digital bus, dedicated to the control of the operation of the whole system, which is the digital instruction. The third bus is analog, and it is shared by the video sources, and the non-standard components of the chipset. These are the analog parallel array processor (APAP), which in this case is a CNNUC, and the analog RAM chip (aRAM). Let us consider their theoretical processing and storage capabilities and local throughput. These specifications will be crucial in the design of the CNN chipset. In the first place, the CNNUC is composed of $M_p \times N_p$ cells. It usually incorporates two internal analog buses carrying the input and output signals. Their widths will be referred as b_{pi} and b_{po}, respectively. For the aRAM chip, a total of $M_m \times N_m$ analog memory cells are appointed. It will include I/O buses of b_{mi} and b_{mo} lines, respectively. We will see later how the widths of these buses are extremely important in meeting the specifications for some applications.

Timing and non-standard parts specification

If we consider a pipelined internal structure of the CNN processor, the critical points in the design of the CNN chipset are transmission bottlenecks. The optimum performance of the pipeline is obtained when it is full the most of the time [Poll90]. The stage with the lowest transition rate in the chain makes the rest of the pipelined stages to wait for it to accomplish its job. That leaves some of the pipeline stages empty every now and then, what has a detrimental influence on the efficiency of the pipeline. Recalling the definitions in Chapter 1, the rate at which the video source is delivering data is given by the product of the frame rate, f_V, or vertical refresh frequency, times the image height, N_V, which is the number of lines in the image frame, times the image width, N_H, that is the number of samples contained in each line. This product is known as the pixel clock frequency, or, simply, the sampling rate:

$$f_S = N_H N_V f_V \tag{2.15}$$

Let us suppose by now that we are not using the aRAM chip. Consider, as well, that t_{pi} and t_{po} are the input acquisition and output settling times of the processor I/O buses, of widths b_{pi} and b_{po}, respectively. We can define f_{pi} as the rate at which data is entering the CNN chip, which is the quotient between the input data bus width, b_{pi}, and the input acquisition time for one sample, t_{pi}:

$$f_{pi} = \frac{b_{pi}}{t_{pi}} \tag{2.16}$$

Obviously, the smaller the acquisition time the wider the analog data bus, the larger the input rate. Then, the first conclusion we can extract is that

$$f_{pi} \geq f_S \tag{2.17}$$

must hold. Otherwise we will have a bottleneck between the video signal source and the processor. Once the samples enter the CNNUC, they are processed in parallel during time t_{pp}. This period of time, corresponding to the execution of the analogic parallel algorithm, can be decomposed in:

$$t_{pp} = n_{ap} t_{ap} + n_{lp} t_{lp} + n_{mp} t_{mp} + t_{ov} \tag{2.18}$$

where, n_{ap} is the number of analog instructions — the programmed spatio-temporal dynamics— of the analogic program and t_{ap} is the average time required for complexion of an analog instruction, n_{lp} is the number of local logic operations and t_{lp} is the average time needed to perform each of them, n_{mp} is the number of accesses to the local memories during program execution and t_{mp} the average time required for that purpose, and finally t_{ov} is the timing overhead necessary to accomplish resetting and calibration before the analogic program starts.

Going back to the overall internal processing time t_{pp}, since the whole $M_p \times N_p$ sub-image will be processed in parallel, we can define an inter-

nal processing rate as:

$$f_{pp} = \frac{M_p N_p}{t_{pp}} \tag{2.19}$$

The larger the processor, the faster data processing rate attainable.In order to avoid data bottlenecks inside the processor, given that no inner buffering has been considered to be available at this point, the following assumption must hold:

$$f_{pp} \geq f_{pi} \tag{2.20}$$

Finally, concerning the rate at which the output data are delivered, an output rate for the processor can be defined as:

$$f_{po} = \frac{b_{po}}{t_{po}} \tag{2.21}$$

It must accomplish the following inequality to avoid data jamming:

$$f_{po} \geq f_{pp} \tag{2.22}$$

if this is not true, more samples are processed than delivered by the CNNUC, thus causing the bottleneck to be at the processor output.

Let us illustrate the above formulae with some real data. Consider a a 640×480-pixel image transmitted at 30Hz frame rate. This supposes a data rate of 9.2Ms/s, or, what is equivalent, a sampling frequency, f_S,of about 9.2MHz. Since the image frame is delivered pixel-by-pixel by the video signal source, the width of the input bus of the processor is 1. Therefore, the required acquisition time t_{pi} must be under 110ns in order to avoid input data jamming. If this is accomplished, the commonly attainable processing rates easily hold Eq. 2.20 because of the high degree of parallelism implemented by the processor core. For instance, in a 32×32-cell processor, t_{pp} can be 1024 times t_{pi} and still the inequality in Eq. 2.20 will be true. This yields several tens of microseconds to process each $M_p \times N_p$-pixel chunk of the image. What seems more difficult, in principle, to achieve is the last condition, Eq. 2.22. Usually b_{po} is also 1-line wide. If we have kept f_{pp} close enough to f_S, then, the limitations are the same as for the imager, delivering analog samples off-chip at video rates. It can be done by just investing enough power. But, if the processor operates at a much higher rate than f_S, there will be large periods of time in which the processor stops to wait either for a new input to arrive or for the output to be delivered. This reduces the efficiency of the pipeline because some stages, the processor, have idle periods.

An architectural solutions to enhance the pipeline efficiency can be tried. For instance, imagine internal data clogging at the processing array because of a small output rate, f_{po}. This can be solved by adding an

internal buffer of the appropriate size. Suppose that video samples acquisition and parallel processing occur exactly at the sampling rate, f_S. Each $M_p \times N_p$-pixel piece of the original image enters the chip in a time:

$$T_{\text{in}} = \frac{M_p N_p}{f_S} = T_{\text{proc}} \tag{2.23}$$

It is also, as we have assumed, the time it takes to process the whole piece. On the other side, the time the output stage takes to deliver the $M_p \times N_p$-pixel image block is given by:

$$T_{\text{out}} = \frac{M_p N_p}{f_{po}} \tag{2.24}$$

The difference between them is:

$$\Delta T = T_{\text{out}} - T_{\text{in}} = M_p N_p \left(\frac{1}{f_{po}} - \frac{1}{f_S} \right) \tag{2.25}$$

If this ΔT is multiplied by the rate at which data crowds into the processor, we will have the size, in pixels, of the buffer that must be placed between the processing array and the output stage in order to acquire images blocks at f_S and to deliver them at a slower rate:

$$N_{\text{buff}} = M_p N_p \left(\frac{f_S - f_{po}}{f_{po}} \right) \tag{2.26}$$

If the original image is of $N_H \times N_V$ pixels, and a certain overlap in the horizontal, n_{ho}, and the vertical directions, n_{vo}, is permitted to avoid border effects in the image processing, the original image frame must be split up in this many pieces:

$$N_{\text{pieces}} = \frac{(N_V - n_{vo})(N_H - n_{ho})}{(M_p - n_{vo})(N_p - n_{ho})} \tag{2.27}$$

making N_{buff} grow. And if a sequence of N_F frames is to be processed, the size of the buffer must increase N_F times. Thus for processing a sequence of N_F frames of size $N_V \times N_H$ pixels, acquired at a rate f_S, an internal buffer of size:

$$N_{\text{buff}} = N_F \frac{(N_V - n_{vo})(N_H - n_{ho})}{(M_p - n_{vo})(N_p - n_{ho})} M_p N_p \left(\frac{f_S - f_{po}}{f_{po}} \right) \tag{2.28}$$

is required if the delivery rate is limited to f_{po}. This solves compatibility between the sampling rate of the imager, and the maximum available delivery rate of the processor. If the buffer is not available, the system can process data, but at a lower speed.

Serializing and deserializing analog data with the aRAM chip

The majority of the analog array processors reported in the literature do not have a pipeline architecture. They do not allow simultaneous acquisition, processing and downloading of images. However, the aRAM chip in the CNN chipset can be employed for implementing a processing pipeline if it is designed so. In a processor in which analog samples are acquired, processed and downloaded sequentially, the times required for all these tasks add up to a total processing time per pixel:

$$T_p = \frac{1}{f_{pi}} + \frac{1}{f_{pp}} + \frac{1}{f_{po}} \tag{2.29}$$

If the video signal source delivers data at f_S, then the condition to be held for avoiding bottlenecks is:

$$f_S \leq \frac{1}{T_p} = \frac{f_{pi} f_{pp} f_{po}}{f_{pi} f_{pp} + f_{pp} f_{po} + f_{pi} f_{po}} \tag{2.30}$$

what results fairly difficult to achieve. Imagine that f_{pp} is, because of the parallel structure of the processor, so high that Eq. 2.30 reduces to:

$$f_S \leq \frac{1}{T_p} \approx \frac{f_{pi} f_{po}}{f_{pi} + f_{po}} \tag{2.31}$$

In these conditions, if $t_{pi} = t_{po} = 1/f_S$, and the I/O buses of the processor are of width 1 —analog samples arrive and depart serially, one-by-one— Eq. 2.31 does not hold. The only solution, to maintain the video signal data rate, f_S, is to increase f_{pi} and f_{po}, or, equivalently, to decrease the chip I/O times to $t_{pi} = t_{po} \leq 1/2f_S$.

However, from the definitions of f_{pi} and f_{po}, Eqs. (2.16) and (2.21), it can be seen that acting on the I/O buses widths, these frequencies can be modified. This can be done with the help of the aRAM chip. This chip is not standard, has to be custom designed. It communicates with the video signal source through a serial port while connecting to the processor via a wider analog bus. The aRAM interfaces the CNNUC with the signal source with a sample delivery rate of f_S. These data is now acquired by the aRAM chip, through an analog serial input, at that precise rate. Now, assuming that the parallel input and output buses of the CNN chip and the aRAM are compatible, this is $b_{pi} = b_{mo}$ and $b_{po} = b_{mi}$, the data is uploaded to the processor at a rate f_{pi} that can be made larger than f_S. If $b_{pi} > 1$, f_{pi} will be larger than f_S, even if the acquisition and delivery times for each sample, t_{pi} and t_{po}, are equal to the video source sampling time, $1/f_S$. Then comes parallel processing, which occurs at a rate f_{pp}, and after that, output image downloading at a rate f_{po}, which is also larger than f_S because of the width of the interface between the CNN processor and the aRAM. Finally, the aRAM chip will deliver the output image at the original video rate, f_S, one sample at a time, via its serial output channel.

Therefore, the critical points in the pipeline formed by the video source, the aRAM chip, the CNN processor and the video output are, again, the data rates at the interfaces. Between the aRAM and the video signal source and video output, the original data rate, f_S, is maintained. Then, in order to avoid data jamming, and considering that the processor job is not pipelined, the following relation must hold:

$$f_S \leq \left(\frac{1}{f_{pi}} + \frac{1}{f_{pp}} + \frac{1}{f_{po}} \right)^{-1} = \left(\frac{t_{pi}}{b_{pi}} + \frac{t_{pp}}{M_p N_p} + \frac{t_{po}}{b_{po}} \right)^{-1} \qquad (2.32)$$

which now makes sense, being b_{pi} and b_{po} larger than unity.

Let us consider a numerical example. For the processor, we will use the same data as before: I/O times of 100ns and 32×32 processing cells. The I/O buses widths are 16. Concerning the analogic program, let us assume that it consists of 20 analog instructions, each of them taking $1\mu\text{s}$ to be accomplished, 20 local logic instructions, each of them taking 100ns to be performed, 40 local memory accesses, what take 100ns each, and $20\mu\text{s}$ of timing overhead. This analogic program, which is not precisely a simple and small example, takes $46\mu\text{s}$ to be completed. Substituting these figures into Eq. 2.32, the global processing rate, T_p^{-1}, obtained is 17.41Ms/s, which is compatible with the original f_S of 9.2Ms/s.

Application-oriented CNN systems

The above application of an independent aRAM chip to the assembly of a processing pipeline for video signal processing on-the-fly makes sense at the prototyping and academic levels, where integrated circuits and systems of this potential are made general purpose in order to investigate for possible applications in parallel. The full exploitation of the CNN processing capabilities requires, however, the adaptation of the CNN chip and chipset architectures to the specific characteristics of a particular application field. This will allow the integration of more functionalities into the same die, reducing the CNN chipset, or at least the non-standard section, to a single chip (SoC) containing the processor, the analog storage, the image sensing devices, data converters, digital interfaces, etc. The difficulties are obvious: the chip area grows and so the yield decrease, the co-existence of complex analog and digital circuitry on the same substrate compromises signal integrity and system performance, etc. But the benefits are clear: if analog signals are confined to the interior of the chip, errors due to signal interaction can be reduced, better control of the operation can be realized with a full digital interface, autonomous and stand-alone systems can be developed, all with a tremendous impact on the system cost because of the integration of the most of it in a single IC.

For this to be accomplished, the observations made above have to be internalized by the CNN chip architecture. consider the case of a smart-camera consisting in $N_V \times N_H$ pixels each with photosensing and

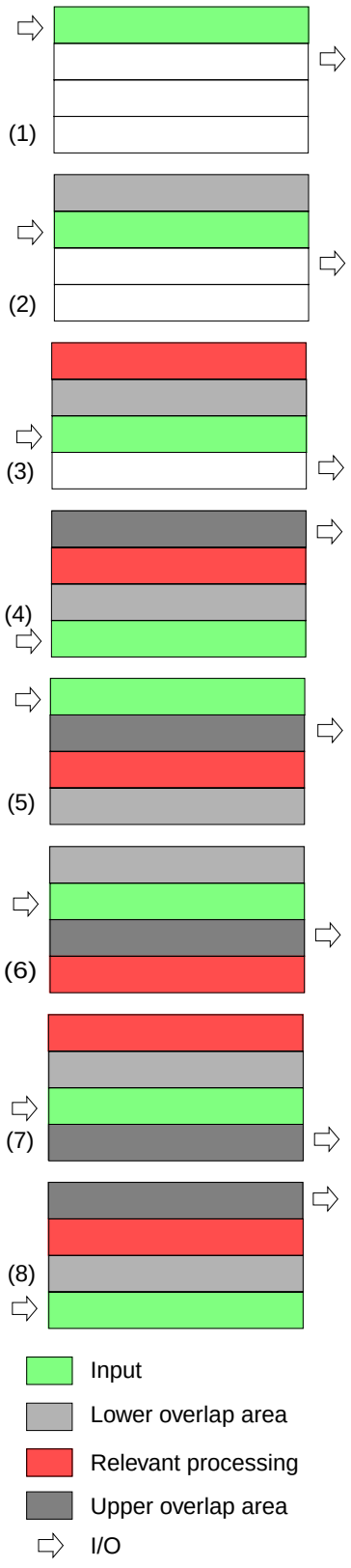


FIGURE 2.9. Cyclic array processor for real-time 2-D processing.

processing capabilities. In the specifications for such a device we find the frame rate, f_V, which is the numbers of pictures that will be taken, processed, or delivered per second, and this, together with the array size, help us to compute the pixel clock frequency, or, simply, the pixel rate:

$$f_S = N_H N_V f_V \tag{2.33}$$

In this conditions, we will have $1/f_V$ seconds to acquire and process the image, if the architecture of the system permits simultaneous acquisition and downloading of the image, this is, if the different tasks are properly pipelined. Adding one more stage to the pipeline, parallel processing can be done on the previous picture frame while the new one is being acquired.

Another application that can benefit from an adapted architecture could be CNN-based fusion of data coming from heterogeneous sources: infrared cameras, intensified video, thermal pictures, etc. In general, these signal sources transmit data serially in a line-by-line basis. On the other side, CNNs operate onto a 2-D grid, thus the whole neighborhood must be available before the processing can be realized. Apart from those CNN templates with a local scope, an estimation of the necessary overlapping of images to compute propagation CNN operations with a certain accuracy can be made. Based on this estimation, a system with only a few rows can be designed to perform on-the-fly data fusion. For instance, consider the cyclic array processor of Fig. 2.9. The inputs, arrive one line after the other. The rows of the array are grouped into four sections. At the beginning the fist section is loaded with the corresponding lines of the image, in a second step, they become the lower overlap area of a virtual upper group. After that, and because they are surrounded by the corresponding overlap areas, the first group of rows is processed (step 3). In the following step it acts as the upper overlap of the second section, which is processed in turn, and at the same time, its output is delivered (step 4). In the following step, the data stored in the first section is not useful any more, then it is updated with the contents of new image lines (step 5). And the process starts again until the last lines of the image are processed and delivered. If extra time is needed for the processing, because of the complexity of the analogic algorithm, overlap areas can be grown to increase the time for each stage. This increases the latency of the system, but has no effect on the I/O data rates.

3 A methodology for the simulation and design of CNN-based processors

VLSI design of CNN-based processors, being nonlinear analog circuits of a high-complexity —above 10^6 transistors in a chip implementing an array of practical size— is strongly influenced by technology considerations, particularly by the available parameter tolerances provided by the silicon vendor. In these conditions, it is important to improve the robustness of the CNNUM algorithm in order to overcome technology limitations for a successful physical implementation. A reasonable start to improve the robustness of the algorithms is a detailed analysis of the dynamic routes of the individual cells of the network for each specific set of templates. The accuracy requirements of the cell parameters responsible of a correct evolution will be a major constraint in the circuit design, thus the more relaxed they are the more robust the implementation will be. Therefore, on one hand, we are interested in selecting a more or less extensive set of templates with the largest tolerances possible, in order to provide the widest design room. This is known as template design centring. The circuit realization is protected, in this way, against inaccuracies in the physical and electrical parameters thus achieving a more robust implementation.

On the other hand, several problems are derived from the real implementation of the algorithms: nonlinearity and reduced accuracy in the template elements implementation, linearity of the multipliers, delay on the limiters response, etc. It means that a set of templates that has been properly centred with an ideal model of a CNN, would, at the end be totally irrealizable because of physical implementation constraints.

Thus, any improvement in the robustness of the implementation have to deal with, on one side, design centring and template element optimization, i. e. loosening dynamic range requirements in their implementation, etc. And, on the other side, an accurate modellisation of the non ideal effects introduced by real circuits. But, unfortunately, traditional optimization algorithms are not going to work perfectly in this context, because the error surfaces to be studied here are pretty rugged, and very badly

behaved —not only a plethora of local minima can be found, but also discontinuities and singularities of the error surface have to be confronted. This makes any guided optimization algorithm render a rather poor performance. Thus, if stochastical methods for optimization are going to be employed, how to realize a huge number of simulations to properly cover the design space, with enough accuracy to account for nonideal effects but with a moderate computing power waste, in a reasonable time to allow iterations in the design flow. This is the problem to be solved by the tools and methods reported in this chapter.

3. 1. Behavioral modeling and simulation of CNN chips

3. 1. 1. Some motivations for a behavioural modelling

It has been demonstrated that the outstanding processing capabilities of CNNs for high speed parallel processing [Chua96] are only fully exhibited by dedicated mixed-signal VLSI hardware realization [Chua93], compared to software or hardware accelerated digital implementation [Fehe96]. Typical CNN chips may contain up to about 200 transistors per pixel (including sensory and processing devices) [Espe96a] [King95]. Together with this, practical and industry-oriented applications require large enough grid sizes (around 100×100). Thus, CNN chip designers must confront complexity levels larger than 10^6 transistors, most of them operating in analog mode. Simulation plays an important role in the design of these complex analog systems (Fig. 3.1). It has to be *fast* enough to allow successive iterations in the analog blocks design in a reasonable time. Moreover, it must reflect accurately the behaviour of these blocks, and give the VLSI designer *reliable* information for making decisions and solving the design trade-offs. Unfortunately, in the design of CNN chips, and in general of analog parallel processing chips, simulation is strongly conditioned by computing power limitations. On one side, high-level simulation packages are unable to easily implement realistic models for electronic circuits [Plon90]. Although they are fast, and can be used to develop new CNN applications, their lack of detail makes them ill-suited for reliable IC simulation. On the other side, SPICE-type electrical simulators are barely capable to handle more than about 10^5 transistors and may take several days CPU time on a Sun SPARCstation 10 for circuit netlists containing about 10^6 transistors [Meta88]. In other words, low-level simulation tools become extremely slow when treating with rather large networks, making reliable simulation of CNN chips unfeasible. Therefore, it is necessary to bridge the gap between these approaches to achieve efficient simulation of CNN chips, an intermediate solution must be found.

In a straightforward approach, high-level simulators can be configured

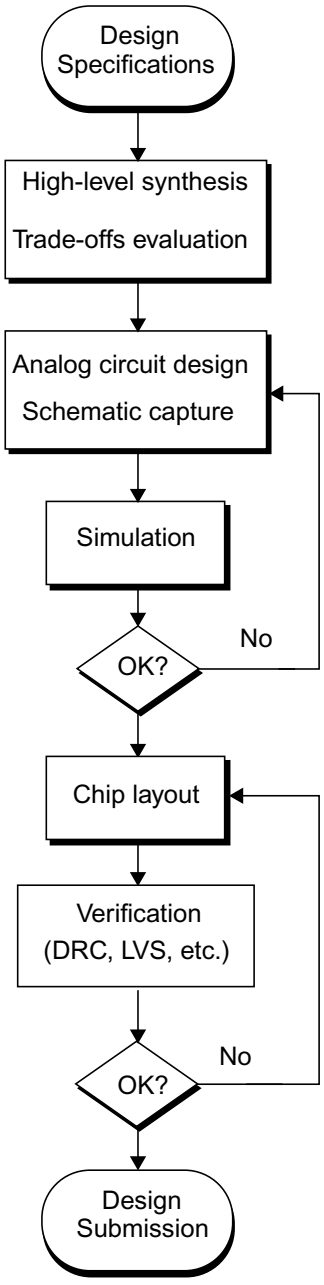


FIGURE 3.1. Generic analog VLSI chip design flow chart.

and programmed to achieve a more realistic simulation of CNN hardware, albeit at the expense of a considerable coding effort. On the other side, the use of macromodels and behavioural definitions in SPICE-like tools will decrease the CPU time consumption. This results in a simplified but still accurate description of the circuit, although the simulator core still has to handle the whole network interconnection topology. Again, the limitations on computing power make this approach inefficient when dealing with large CNN chips. Therefore, the following methodology is intended to overcome these limitations by focusing on the basic processing unit elements and behavior.

The network topology will be embedded within the simulator core of a dedicated tool, connections between cells and topological considerations are implicitly defined. Making use of the VLSI-oriented properties of CNNs, such as regularity, uniformity and local connection scope, the network can be expressed as a whole in which relevant variables result from an aggregation of local values. In other words, the CNN description consists in a matrix form differential equation in which inter-cell connections between different matrices elements are implicitly described and, thus, no further considerations upon circuit topology must be undertaken. This allows, as well, an arbitrarily detailed description of the network, which means that a primary idealistic model can be progressively refined and improved to reflect the effects of any relevant non-ideality that might arise in the behaviour of a specific CNN implementation. This approach has been successfully adopted in the development of SIRENA, a CAD environment for the behavioural modelling and simulation of mixed-signal VLSI parallel processing chips based on CNNs [Carm99a]. SIRENA has been used throughout the text to illustrate the efficiency of dedicated tools in the simulation of CNN chips. Consequently, a brief introduction to this environment will be given. The network models in the text have been developed with the help of SIRENA environment model generation tool (GMS), and the simulations have been performed by NSS (SIRENA's simulator core). However, these results can be extrapolated to a different software implementation based upon the same philosophy, e. g. a specially coded C++ program or a MATLAB script library supported by compiled executables, like in the case of [Reke97], a CNN simulation toolbox including different VLSI-oriented kernels encoded.

3. 1. 2. SIRENA: A VLSI-oriented environment

An introduction to SIRENA environment

SIRENA is a simulation environment for CNNs oriented towards VLSI implementation. It has been written in C language using object-oriented programming techniques, and currently runs under the UNIX operating system and the X-Windows framework. SIRENA has been developed in an attempt to overcome currently available non-specific CAD tools limitations in the field of CNN design. Restrictions on the model description have been mostly avoided so any network based on a cellular structure in which interconnection between the basic cells is restricted to a specific neighborhood (that could be as large as the network itself), and whose operation relays on some nonlinear dynamics taking place within the individual processors of the array, can be described and simulated by SIRENA. An especially developed high-level language (DECEL) has been defined to accomplish this. A careful object-oriented programming of the environment allows feasible operation with widely different CNN models without further modification and recompilation of the main tools source code.

Efficiency of SIRENA is related with the form in which CNN models are described, compiled and linked to other components of the system. The network description emphasizes the basic processing unit features, while the global architecture is embedded in the simulator core and so implicitly defined in the model generation process. Thus, CNN model description consists in a matrix form differential equation, its controlling parameters and a set of matrices representing those variables being relevant in the evolution of the network. Besides, another key for efficiency, also available in SPICE-like simulators, is the arbitrary complexity of the model. From a pure theoretical sketch of the circuit to a highly detailed description, with a deep analysis of parasitics and higher order dynamics.

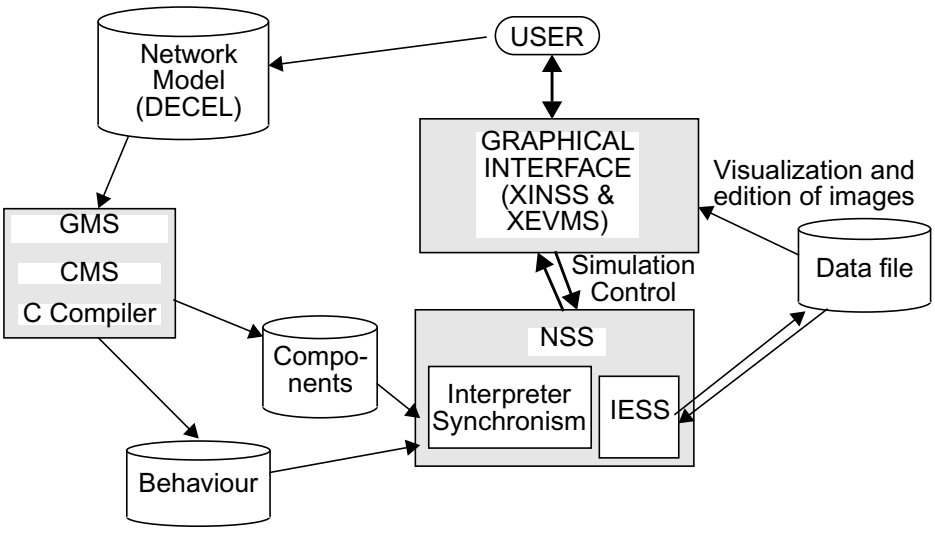


FIGURE 3.2. Conceptual block diagram of SIRENA.

DECEL allows a progressive insight on the network behaviour based on this systematic model refinement.

Fig. 3.2 depicts the architecture of SIRENA. It consists of three main modules: the Model Generation Tool (GMS), the Simulator Core (NSS) and the Graphical User Interface (GUI). The modularity of this system allows workable expansion and improvement based on the primal version of the executable code. Therefore, SIRENA makes use of a user-defined network description, written in DECEL, that is properly handled and compiled into a usable format by the model generation program, GMS. Specific auxiliary files describing network components and behaviour, to be used during simulation processes, are generated, and the linkage to the NSS is performed. User-defined data files contain the information concerning the particular characteristics of the experiment, *i. e.* size of the network, input data, parameter values, etc. A simulation script configures the NSS, setting integration method, convergence criterion, and type and timing of the analysis. Outputs are being appended to an output file that can be simultaneously parsed by the GUI (XEVMS) to visualize the evolution of the network simulation. The format of the output can be set in the simulation script before the execution time or modified after the simulation by any of the graphic format translation tools.

Network model description: DECEL language

One of the keys for an efficient software implementation of CNNs is the exploitation of both their regular topology and their hierarchical structure. In order to illustrate the methodology for describing this type of network, taking advantage of the inherent properties of CNNs, let us consider the original definition proposed by L. O. Chua and L. Yang in 1988 [Chua88]. In their initial model, the evolution of each individual cell C_{ij} in the array is expressed in terms of a network time-constant (τ), a neighborhood S_{ij}^{r} with radius of vicinity r, feedback and control linear template elements (**A** and **B**) that weight the influence of the neighbours' input and output voltages ($u_{(i+k)(j+l)}$ and $y_{(i+k)(j+l)}$) and an offset term z (as can be seen in Fig. 1.20, repeated in Fig. 3.3 for chapter's self-containment). Cell state, x_{ij}, is the integral of a sum of weighted contributions from the neighbouring processors, an offset and a losses term,

$$\tau\frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + z + \sum_{k=-r}^{r}\sum_{l=-r}^{r}[a_{kl}\cdot y_{(i+k)(j+l)} + b_{kl}\cdot u_{(i+k)(j+l)}] \tag{3.1}$$

and, on the other side, cell output, y_{ij}, is a sigmoidal non-linear function of the state voltage, as expressed by the following equations:

$$y_{ij} = f(x_{ij}) = \frac{1}{2}(|x_{ij}+1| - |x_{ij}-1|) \tag{3.2}$$

Some modified versions of this initial model introduce new features and algorithm extension, as nonlinear or delay-type templates [Rosk90], and discrete-time emulation [Harr92a]. Other works report a VLSI oriented model [Espe94b].

From these equations, it can be observed that the uniformity of the CNN allows the definition of a set of network parameters shared by every cell at the array, e. g. the network time constant, cloning connection template elements, bias term in the simpler model. Also, the relevant variables in the basic processor can be grouped into a set of matrices that represent the state, input, output and the bias map of the whole network. In this manner, the evolution law of the network can be represented by a matrix form differential equation in which the different variables involved are interrelated by some functional expressions in matrix form. The CNN model is then completely described by the declaration of the network variables (matrices), the parameters (scalar or template-type), the form of the differential equation and the shape of the relations between variables. Let us realize now how this is stated within SIRENA environment using DECEL language. The network description file (**chua.cs** in Fig. 3.4) is composed of three parts: variables declaration, evolution section and network definition. In the first part every magnitude implicated in the CNN behaviour is itemized indicating type from a pre-defined set, namely: matrix, template, bound or scalar. This is accomplished for an adequate memory space management. After that, the evolution section outlines the relations between those already declared variables. Equations stating these relations are written with the help of the C-language mathematical libraries as well as of special operators belonging to DECEL. Some of them are self-explanatory, like *derivative* operator, some other may need further illustration. For instance, connection evaluator (><) relates a template-type variable with a matrix-type one and performs multiplication of the each template element with the neighborhood of each of the elements of the matrix. The boundary conditions evaluator (#) helps evaluating the connection of elements in the border of the matrix. Conditional operator (:) is employed in piece-wise defined functions, like the piece-wise linear function

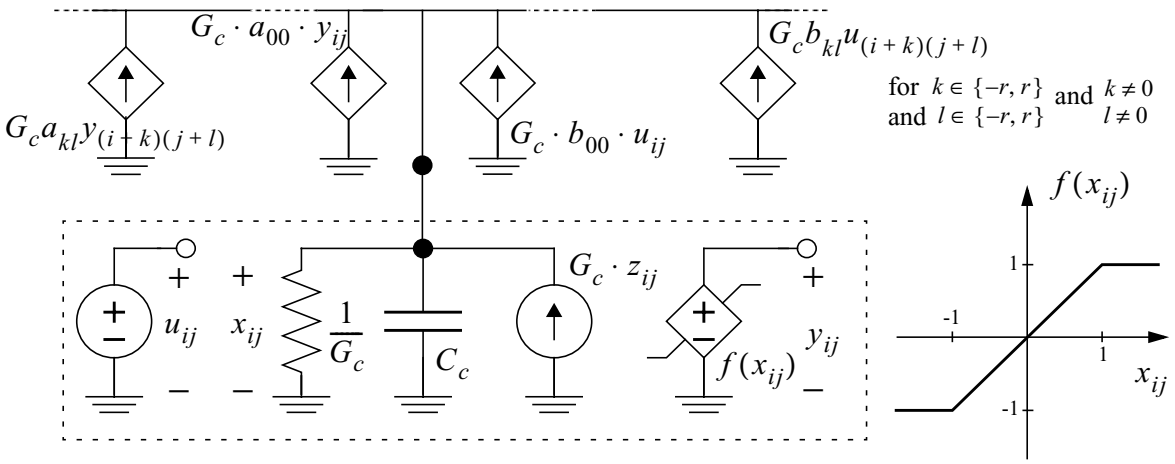


FIGURE 3.3. CNN model originally proposed by Chua and Yang.

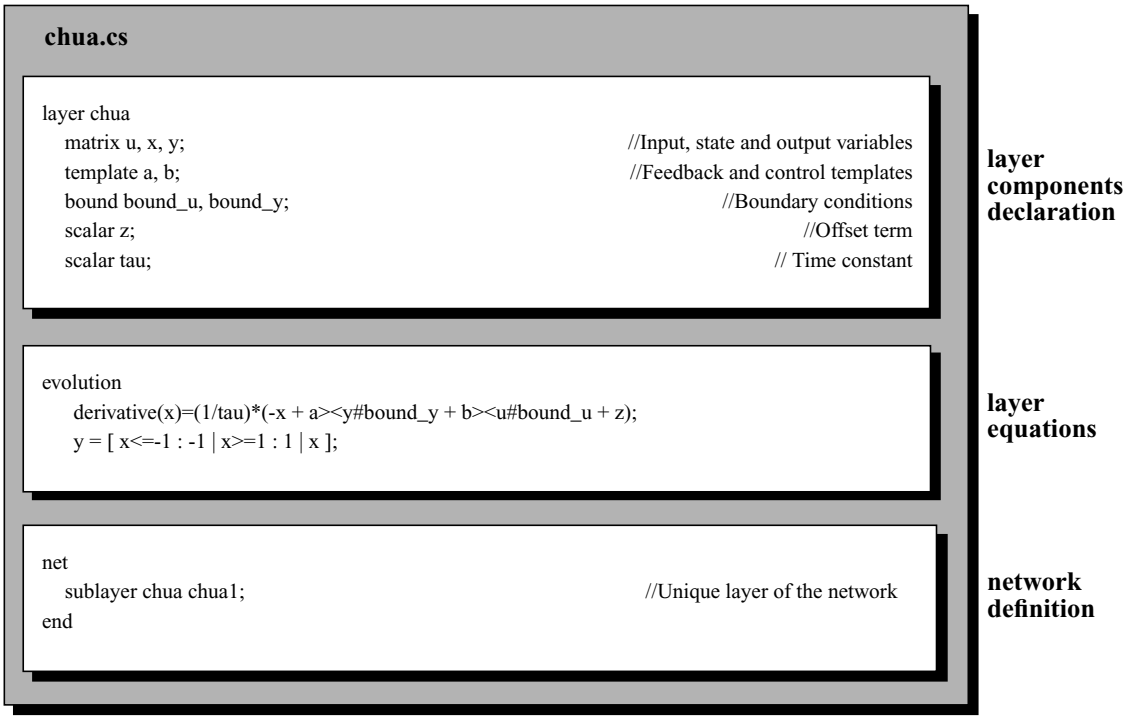


FIGURE 3.4. DECEL description of the Chua-Yang CNN model.

generating the cell output in the example. Once the equations are mapped onto DECEL syntax statements, a one-layer network is defined in the net section. It is important to mention that no specific values for the network variables are included in the model definition and compilation stages, these magnitudes will be instanced at simulation time.

Once DECEL description of the network has been made, it is compiled and linked to the simulator core for further operation. Two files are generated, **chua.des** and **chua.mro** in the example, which report network components and behaviour, respectively. Separation between functional description of the model and instances values assignment permits development of more general CNN models and building a model library that can be extensively utilized by a non-experienced in DECEL user. However, a skilled user can program the CNN model directly in C language to avoid any kind of limitations imposed by DECEL syntax.

The Graphical User Interface

Communication between the user and SIRENA's simulator core can be realized either in the foreground, using an interactive command shell for simulation configuration and control, or in the background with the help of a set of script files. In order to improve user interaction with the simulation control, inputs and outputs, a Graphical User Interface (GUI) for SIRENA was developed, in parallel to the NSS, under the X-Windows framework. The GUI is a collection of graphical tools and libraries which provides user-friendly communication with the simulator core. It allows the control and supervision of the NSS performance, facilitates input and

output visualization and edition, and permits data exchange with other widespread graphical tools running under UNIX operating system. Fig. 3.5 gives a conceptual view of the GUI and its components interactions.

These tools can be classified into three groups: communication programs, data translators and the properly called GUI. In the first class:

- IESS: an I/O functions library that manages data flow and exchange between the rest of the tools constituting the whole SIRENA package. Actually, the common user does not call it directly, but it is a necessary reference for developers.
- AFDS: data files parser, prints error messages if syntax errors are found inside input data files avoiding NSS and visualization tools faults derived from mistaken input processing.

Data translators allow data formatting to make use of different graphic tools (Fig. 3.6). As SIRENA's output viewer is intended to visualize matrices in a colour or gray scale, these routines make possible the representation of waveforms of any of the network variables.

- TFSXG: converts SIRENA output files, that can be ASCII or binary coded, to *xgraph* (*xvgr* compatible, used in Fig. 3.6(a)) manageable files.
- TFSGSI: converts SIRENA output files to *gsi* (*hsplot* compatible, Fig. 3.6(b)) compatible format.

Finally, the X-Windows based tools conforming the graphic front-end from which the user can control interactively the whole simulation:

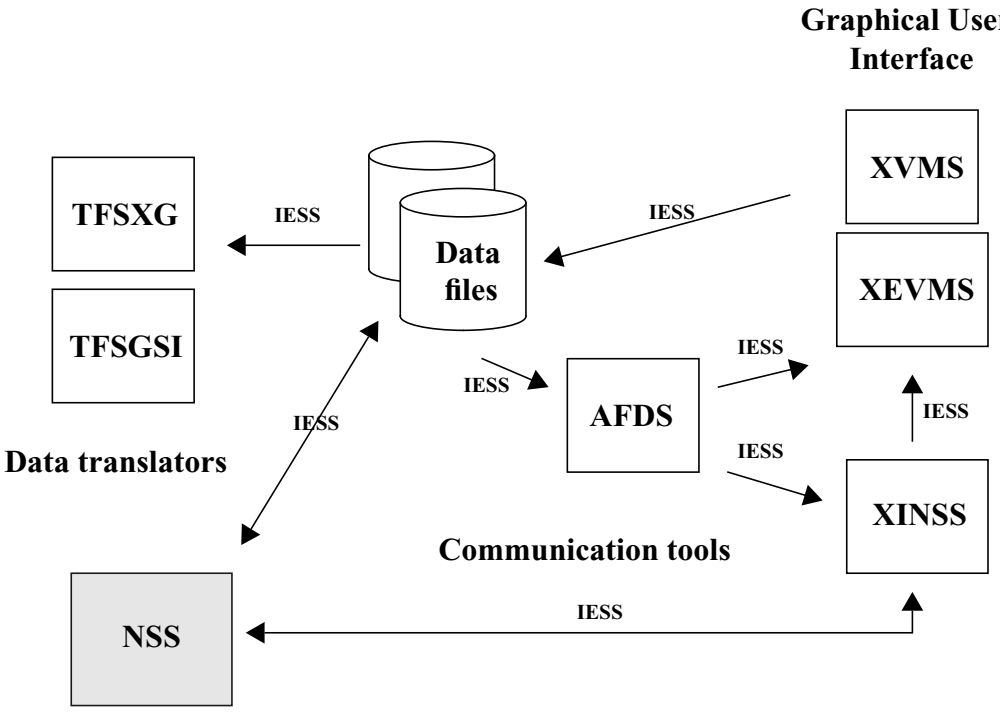


FIGURE 3.5. Conceptual block diagram of the Graphical User Interface.

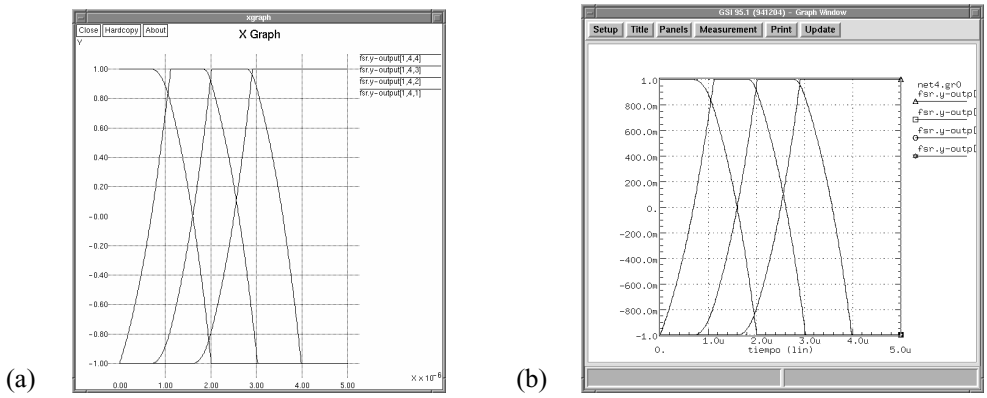


FIGURE 3.6. Waveform of network outputs visualization with (a) *xgraph* and (b) *gsi*.

- XINSS: graphical interface with the simulator core of SIRENA under the X-Windows system. Provides interactive access to the simulation control and output, allowing visualization of results during the simulation and configuration-parameters edition.
- XVMS: matrix viewer, allows image visualization and image-sequences animation. With the help of a command file (ASCII text format), XINSS starts the simulation (Fig. 3.7), and permits output monitoring using XVMS while it is running, as well as on-line simulation parameters modification. Fig. 3.8 shows the transient analysis of a 8×8 cells network for connected components detection as seen with XVMS.
- XEVMS: matrix editor, enhanced version of the visualizer with capabilities for the creation and modification of images, as well as import/export features (Fig. 3.9).

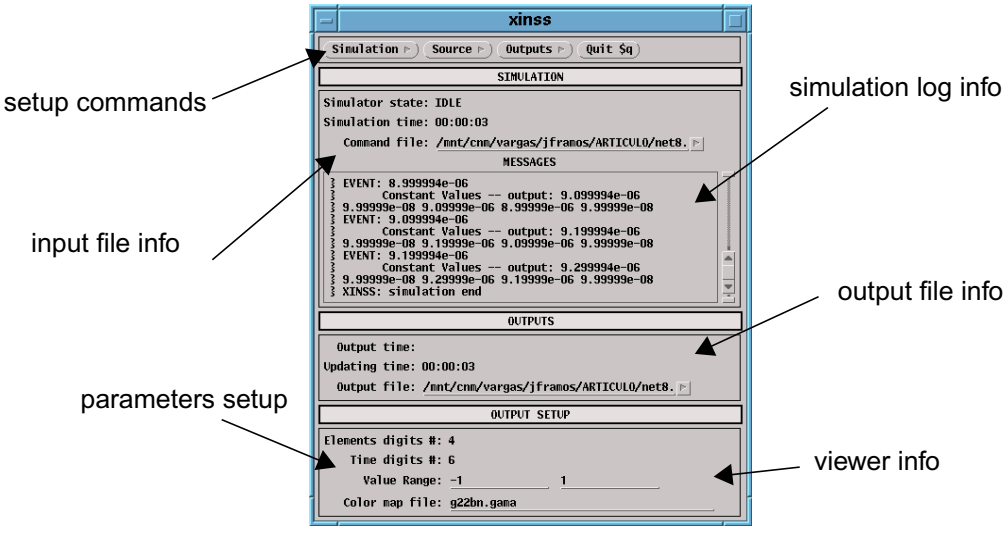


FIGURE 3.7. XINSS: graphical user interface with NSS

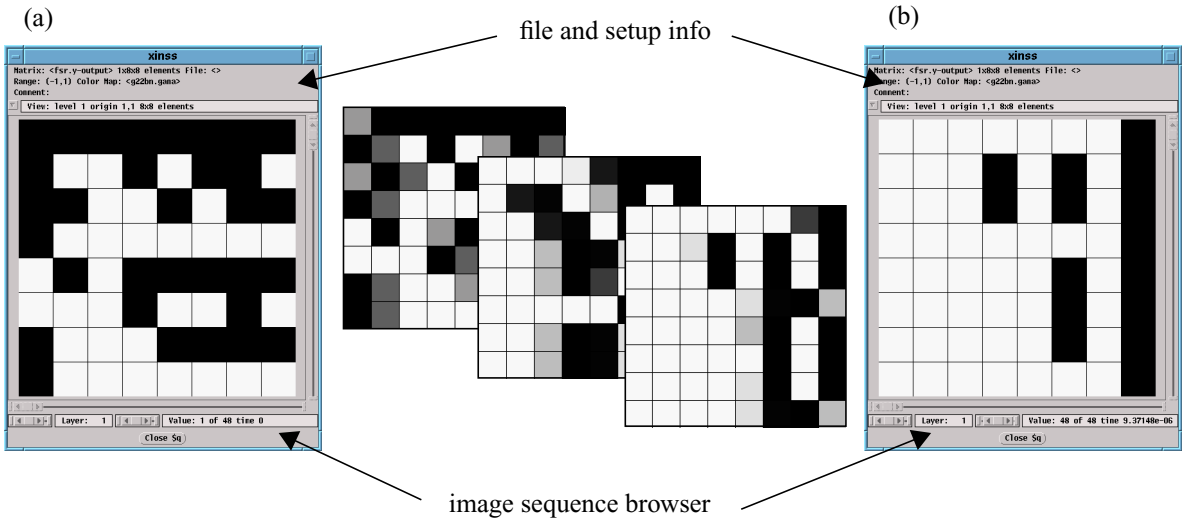


FIGURE 3.8. XVMS: Network input image (a) and output pattern (b) with a set of intermediate states.

3. 1. 3. SIRENA's simulator core

Selection of an appropriate integration algorithm

The output of a CNN model simulation is the final state reached by the network after evolving from an initial state under the influence of a specific input and boundary conditions. Let us concentrate on the case in which the network dynamics are described by a system of nonlinear first-order differential equations. Although finite differential equations, to describe discrete-time CNN models, or higher order dynamics, for higher order CNNs, can be required to track the evolution of the network, the results obtained for the first-order differential equations can be extended to these cases. Thus, the problem is to integrate this set of equations:

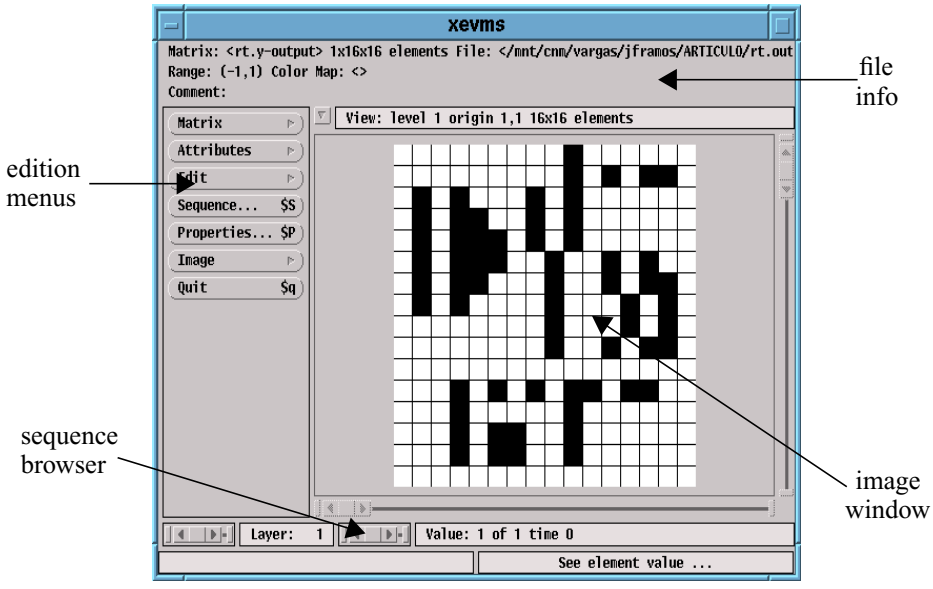


FIGURE 3.9. Input edition with the extended matrix viewer (XEVMS).

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}[\mathbf{x}(t)] \qquad \text{or} \qquad \mathbf{x}(t) = \mathbf{x}(t_0) + \int_{t_0}^{t} \mathbf{f}[\mathbf{x}(\tau)]d\tau \tag{3.3}$$

which must be integrated numerically, because a closed form of the solution can not be given. The simulation of such a system of equations on a digital computer requires mapping the system to a discrete-time system which emulates the continuous-time behaviour with similar dynamics and that converges to the same final state. The error committed by this emulation depends on the form of the discrete-time system selected, which, in turns, depends on the choice of the method of integration, *i. e.* the way in which the definite integral in Eq. 3.4 is calculated.

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + \int_{t_n}^{t_{n+1}} \mathbf{f}[\mathbf{x}(t)]dt \tag{3.4}$$

There is a wide variety of integration algorithms that can be used to perform this task, e. g. forward-Euler, backward-Euler, trapezoidal, gear methods, etc. However, only three of them are going to be considered here for a comparison, following the directions reported in H. Harrer's work [Harr94]. These methods are the explicit (forward) Euler's formula:

$$\int_{t_n}^{t_{n+1}} \mathbf{f}[\mathbf{x}(t)]dt = \Delta t \cdot \mathbf{f}[\mathbf{x}(t_n)] \tag{3.5}$$

the predictor-corrector algorithm, a mixed method with explicit-prediction and implicit-correction:

$$\int_{t_n}^{t_{n+1}} \mathbf{f}[\mathbf{x}(t)]dt = \frac{\Delta t}{2} \cdot \{\mathbf{f}[\mathbf{x}(t_n)] + \mathbf{f}[\mathbf{x}(t_{n+1})^{(0)}]\} \qquad \text{where} \quad \mathbf{x}(t_{n+1})^{(0)} = \mathbf{x}(t_n) + \Delta t \cdot \mathbf{f}[\mathbf{x}(t_n)] \tag{3.6}$$

and the fourth-order Runge-Kutta, a high order explicit method:

$$\int_{t_n}^{t_{n+1}} \mathbf{f}[\mathbf{x}(t)]dt = \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \qquad \text{with} \quad \begin{aligned} \mathbf{k}_1 &= \Delta t \cdot \mathbf{f}[\mathbf{x}(t_n)] \\ \mathbf{k}_2 &= \Delta t \cdot \mathbf{f}\left[\mathbf{x}(t_n) + \frac{1}{2}\mathbf{k}_1\right] \\ \mathbf{k}_3 &= \Delta t \cdot \mathbf{f}\left[\mathbf{x}(t_n) + \frac{1}{2}\mathbf{k}_2\right] \\ \mathbf{k}_4 &= \Delta t \cdot \mathbf{f}\left[\mathbf{x}(t_n) + \frac{1}{2}\mathbf{k}_3\right] \end{aligned} \tag{3.7}$$

These three numerical integration algorithms has been employed to simulate the evolution of a row of 8 CNN cells (Chua's model) intended to perform Connected Component Detection (CCDet) [Mats90] upon the binary input pattern depicted in Fig. 3.10. This operation has been selected because it implies propagation of the signals across the whole array, thus the more complex dynamics can be observed. The cloning templates for CCDet are:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & -1 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \qquad z = 0 \tag{3.8}$$

After several tests, these three algorithms of integration selected, which in principle exhibit approximately the same predictions for the evolution of the network, draw qualitatively different results.

Integration methods comparison

First of all, an estimation has been made on the maximum step size that still leads to convergence to the correct final state for each of the integration algorithms. The results are shown in Table 3.1. It can be concluded that a 4th-Order Runge-Kutta algorithm exhibits better convergence behaviour than the others. Some may think that, as this method can use larger step sizes and still converges to the correct final state, then this is the integration algorithm that requires less computing power to perform a sufficiently accurate simulation. However, this is inexact, because, looking at these methods' formulae, it is obvious that rather more operations per second are employed in one step of Runge-Kutta's methods, that in a

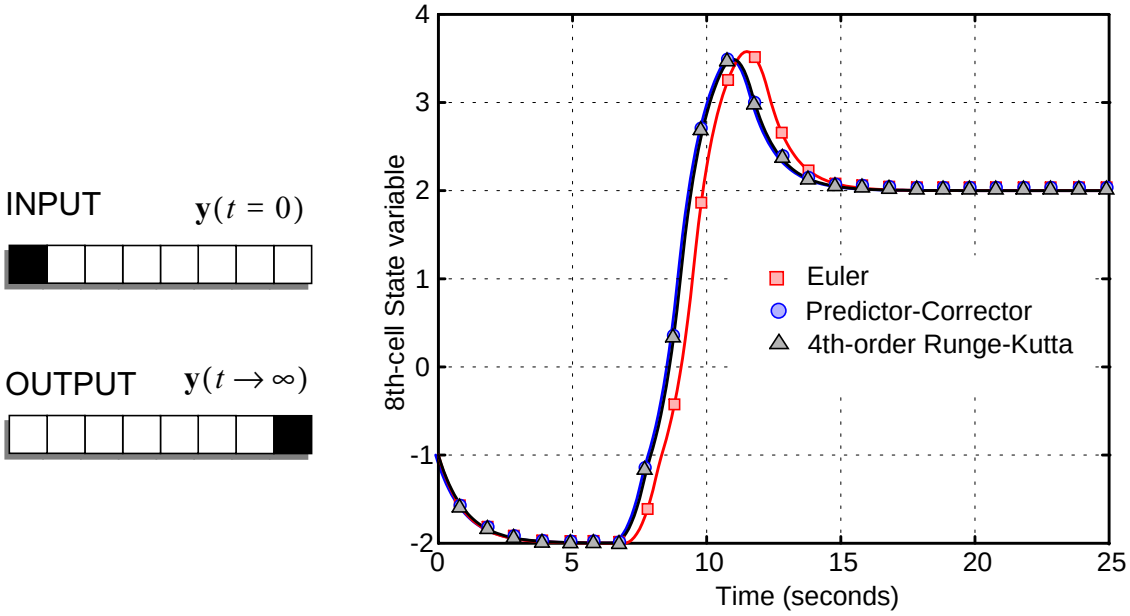


FIGURE 3.10. Waveforms of the state variable of the 8th-cell of the array plotted by the different methods.

step of any of the other two algorithms.

Method	Maximum step size (s.)
Forward Euler	1.575
Predictor-Corrector	1.867
4th-order Runge-Kutta	2.480

TABLE 3.1. Maximum time step for correct convergence.

This is the reason why if the CPU time consumption is evaluated for different step sizes (Fig. 3.11(a)), not a very large difference is found for this network size (1×8).

Advantages of the Runge-Kutta's method are however found in the accuracy on the emulation of the continuous-time system. Thus, the error committed in the calculation of the waveforms of the cell state vector components by each of the integration methods has been measured and represented together (Fig. 3.11(b)). For the error estimation, the reference has been generated by a 4th-Order Runge-Kutta integrator with an extremely fine time step (0.001 seconds). In these tests, Forward Euler's formula presents the less accurate performance.

Therefore, a fixed step 4th-order Runge-Kutta integration algorithm has been implemented in SIRENA's simulator core. But, in order to improve efficiency, reducing CPU time consumption, a step control Runge-Kutta method is included as well, thus allowing the algorithm to gain time by enlarging the step size in non-critical time points without losing accuracy. A relative convergence criterion is applied after each iteration in the simulation loop, so the process stops when the derivatives of the components of the cells state vector, containing the state variable of every cell in the $M \times N$ array, are smaller that an arbitrarily selected quantity (ε in equation Eq. 3.9)

$$\left| \frac{x_{ij}(t_{n+1}) - x_{ij}(t_n)}{\Delta t} \right| < \varepsilon \qquad \forall i \in \{1, \ldots, M\}, j \in \{1, \ldots, N\} \quad (3.9)$$

3. 1. 4. Some examples of modelling

Let us illustrate the modelling methodology —aimed to increase efficiency and simulation speed— depicted in the previous section with the help of the following examples in SIRENA:

Fixed-weight current-mode CNN

This current mode implementation [Rodr93] of a CNN is built on the extensive use of the current mirror. It is designed to perform CCDet upon a binary input pattern. Certainly, the implementation of this specific application CNN makes use of a simpler model, having zero input control tem-

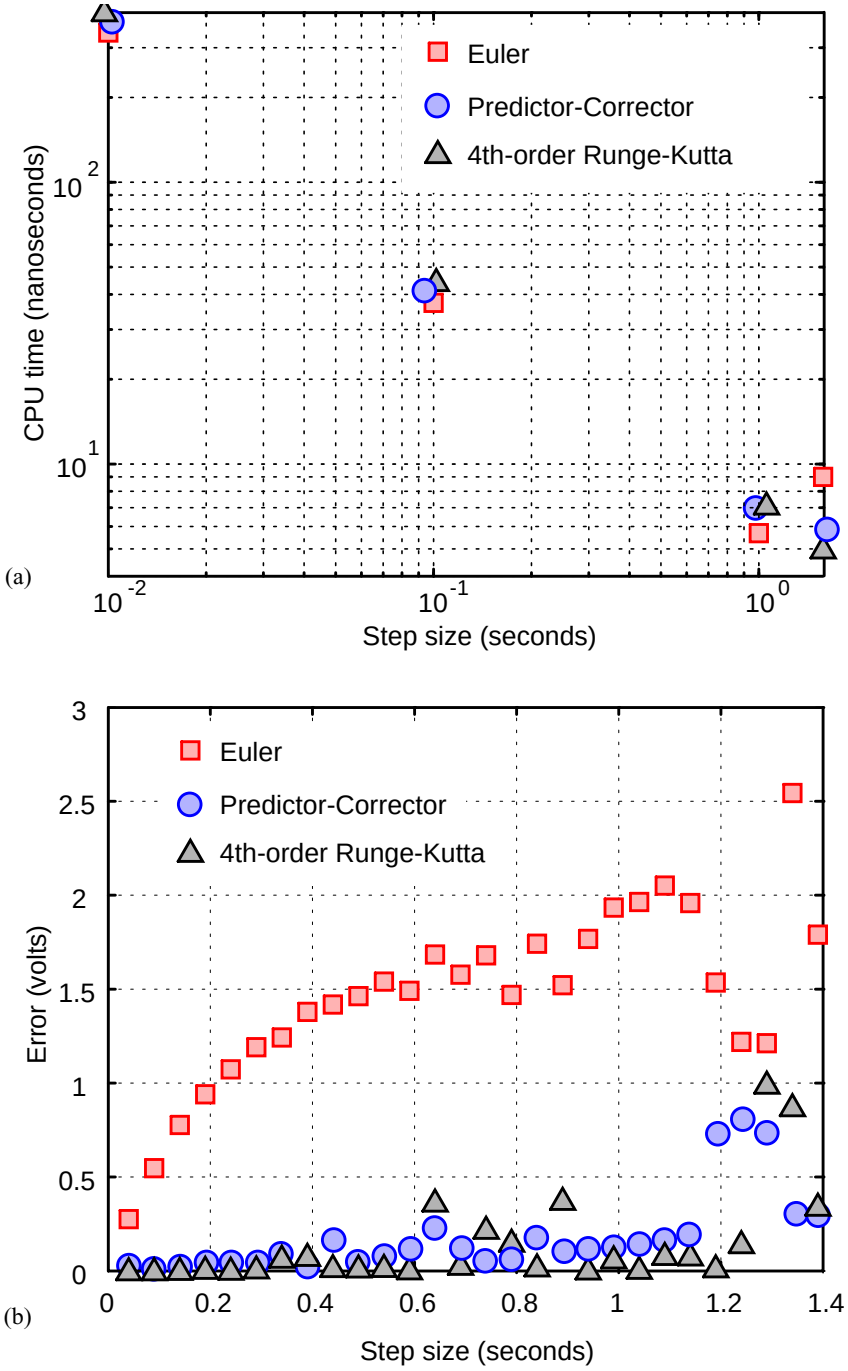


FIGURE 3.11. (a) Log-log graph of the CPU time consumption vs. time step and (b) maximum error in the evaluation of cell state vector vs. time step.

plate and offset term, and only two neighbours connected, but this will be useful enough to illustrate capabilities of SIRENA environment. Besides, this circuit has been successfully designed, integrated, proven and reported [Espe93a].

First of all, the state variables of a basic cell must be selected and their evolution laws must be established by inspection. Thus, consider the schematic of the basic processor depicted in Fig. 3.12. We have chosen the voltages across capacitors C_x,

C_{yp} and C_{yn} to be our state variable, namely: x, y_p and y_n. These two latter are the output variables of the cell, limited versions of the cell state (x) with different sign either. We define also a normalizing factor G_c, with transconductance dimensions, resulting in the following time constants: $\tau_x = C_x/G_c$, $\tau_{yp} = C_{yp}/G_c$ and $\tau_{yn} = C_{yn}/G_c$. In these conditions time evolution of these three state variables of the circuit is described, with the help of some auxiliary currents that will be defined later, by the following set of equations:

$$\tau_x \frac{dx}{dt} = \frac{1}{G_c} \cdot [2(i_{cpc} - i_{cnc}) + (i_{cpl} - i_{cnl}) + (i_{cpr} - i_{cnr}) + (i_{cpx} - i_{cnx})]$$

$$\tau_{yp} \frac{dy_p}{dt} = \frac{1}{G_c} \cdot (2i_{cpyp} - i_{cn1yp} - i_{cn2yp}) \qquad (3.10)$$

$$\tau_{yn} \frac{dy_n}{dt} = \frac{1}{G_c} \cdot (2i_{cpyn} - i_{cn1yn} - i_{cn2yn})$$

Now, the current flowing into C_x is the sum of a self-feedback term, double of the current generated by cell state variable, the neighbours contribution with the sign indicated in the feedback template, and the losses term. Each current contribution has a positive component, due to the PMOS current sources, and a negative term dragged from the n-channel devices, which in our model will be established by level-1 Schichman-Hodges MOS transistor equations. Once state variables and equations are extracted, the DECEL description of the model can be realized (Fig. 3.13). Let us examine the details of the **cm.cs** file. For the layer components declaration, the state and output (no input) variables are defined as matrices. After that, several auxiliary currents, necessary to evaluate contributions in the equations of the state variables are laid down. Two helpful templates are declared[1] in order to sense the neighbours' out-

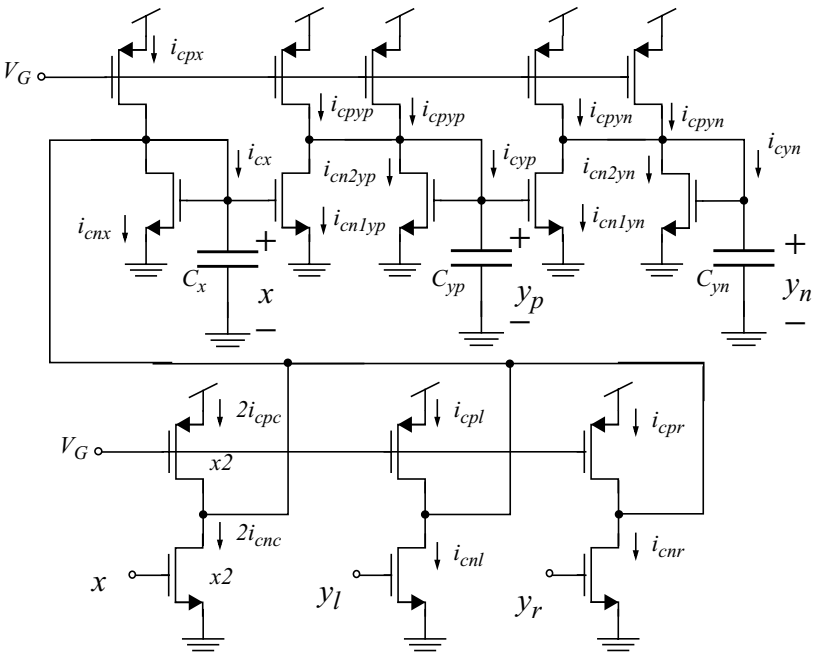


FIGURE 3.12. Schematic diagram of a current-mode CNN cell for Connected Component Detection based on bias-shifted current mirrors.

cm.cs

layer components declaration

```
layer layer_cm
   matrix x, yp, yn ;                                   //State Variables
   matrix yl, yr;                                       //Neighborhood State Variables
   matrix icx, icyp, icyn;                              //Auxiliary currents
                  ⋮                                              ⋮
   matrix icn1yn,icn2yn,icn1yp2, icn1yn2;
   template ayl,ayr;                                    //Template for neighbor state evaluation
   boundary bound_yp,bound_yn;                          //Boundary conditions (vyp, vyn)
   scalar vdd, vg;                                      //Power Supply and Bias Voltage
   scalar sn,sp;                                        //MOS Trans. Geometric aspect ratio
   scalar betap,betan,vtp,vtn;                          //Techn. Parameters
   scalar taux, tauyp, tauyn;                           //Time constants
```

layer equations

```
evolution
        yl= ayl><yp#bound_yp;
        yr= ayr><yn#bound_yn;

        icpl= [ x < vg+vtp : (betap*sp/2)*(vdd-vg-vtp)*(vdd-vg-vtp) |
                betap*sp*((vdd-vg-vtp)*(vdd-x)-(vdd-x)*(vdd-x)/2) ];
        icnl2= [ x > (yl-vtn) : (betan/2)*sn*(yl-vtn)*(yl-vtn) |
                betan*sn*((yl-vtn)*x -x*x/2) ];
        icnl= [ yl > vtn : icnl2 | 0 ];
        icpr=icpl ;
        icnr2= [ x > (yr-vtn) : (betan/2)*sn*(yr-vtn)*(yr-vtn) |
                betan*sn*((yr-vtn)*x -x*x/2) ];
        icnr= [ yr > vtn : icnr2 | 0 ];
        icpc=2*icpl ;
        icnc2= [ x > (yp-vtn) : betan*sn*(yp-vtn)*(yp-vtn) |
                2*betan*sn*((yp-vtn)*x -x*x/2) ];
        icnc= [ yp > vtn : icnc2 | 0 ];
        icpx= icpl;
        icnx= [ x < vtn : 0 | (betan/2)*sn*(x-vtn)*(x-vtn) ];
                icpyp= [ yp < vg+vtp : (betap*sp/2)*(vdd-vg-vtp)*(vdd-vg-vtp) |
                betap*sp*((vdd-vg-vtp)*(vdd-yp)-(vdd-yp)*(vdd-yp)/2) ];
        icn1yp2= [ vyp > (x-vtn) : (betan/2)*sn*(x-vtn)*(x-vtn) |
                betan*sn*((x-vtn)*yp -yp*yp/2) ];
        icn1yp= [ x > vtn : icn1yp2 | 0 ];
        icn2yp= [ yp < vtn : 0 | (betan/2)*sn*(yp-vtn)*(yp-vtn) ];
        icyp= 2*icpyp-icn1yp-icn2yp;
        icpyn= [ yn < vg+vtp : (betap*sp/2)*(vdd-vg-vtp)*(vdd-vg-vtp) |
                betap*sp*((vdd-vg-vtp)*(vdd-yn)-(vdd-yn)*(vdd-yn)/2) ];
        icn1yn2= [ yn > (vyp-vtn) : (betan/2)*sn*(yp-vtn)*(yp-vtn) |
                betan*sn*((yp-vtn)*yn -yn*yn/2) ];
        icn1yn= [ yp > vtn : icn1yn2 | 0 ];
        icn2yn= [ yn < vtn : 0 | (betan/2)*sn*(yn-vtn)*(yn-vtn) ];
        icyn= 2*icpyn-icn1yn-icn2yn;

        derivative(x)= icx/(taux*gc);
```

network definition

```
network
   sublayer layer_cm cm;
end
```

FIGURE 3.13. DECEL description of the current-mode CNN model.

put variable. Finally, boundary conditions and several technology and design parameters are declared as well. Then, the layer equations are stated beginning with the evaluation of the output variables of the neighbors. After that the auxiliary currents are calculated, and closing this *evolution* section, the time evolution laws of the state variables Eq. 3.10 are formulated. The last part of the model description is the network definition as a single-layer CNN. The following table displays the values assigned to each variable and parameter, extracted from HPICE models of the CMOS devices and the circuit analysis:

Variable, Parameter	Name	Value	Units
Initial state (max.)	x, yp, yn	0.98	volts
Initial state (min.)	x, yp, yn	0.70	volts
Power Supply voltage (V_{DD})	vdd	5.0	volts
Bias voltage (V_G)	vg	3.6	volts
NMOS transcond. ($\beta_n^* = \mu_{0n} \cdot C_{ox}^*$)	betan	45×10^{-6}	A/V^2
PMOS transcond. ($\beta_p^* = \mu_{0p} \cdot C_{ox}^*$)	betap	15.4×10^{-6}	A/V^2
NMOS threshold voltage (V_{T_n})	vtn	0.60	volts
PMOS threshold voltage (V_{T_p})	vtp	0.98	volts
Time constant (τ_x)	taux	22.5×10^{-9}	seconds
Time constant (τ_{yp})	tauyp	5.6×10^{-9}	seconds
Time constant (τ_{yn})	tauyn	2.1×10^{-9}	seconds
Positive boundary condition	bound_yp	0.98	volts
Negative boundary condition	bound_yn	0.70	volts
Normalizing transcond. (G_c)	gc	20×10^{-6}	mhos

TABLE 3.2. Current-mode CNN model parameters.

Programmable OTA-based CNN

This CNN implementation based on Operational Transconductance Amplifiers (OTAs) is reported in [Cruz92]. It is a straight-forward mapping of the differential equations defining the CNN dynamics onto some standard CMOS circuit primitives. In this approach, an OTA block performs the nonlinear operation over the state variable (G_{m_A} in Fig. 3.14). Multiplication of the input and output variables and the template elements is performed over the output current of the OTA. This current has been replicated a number of times to implement the whole set of template elements, a total of eighteen multipliers. As in [Rodr93], multiplication occurs inside each cell and the weighted contributions are passed to the neighborhood. Because of this, template elements should be reorganize to

i. The weights of the connections can be included here or left for a later description within the layer equations statement

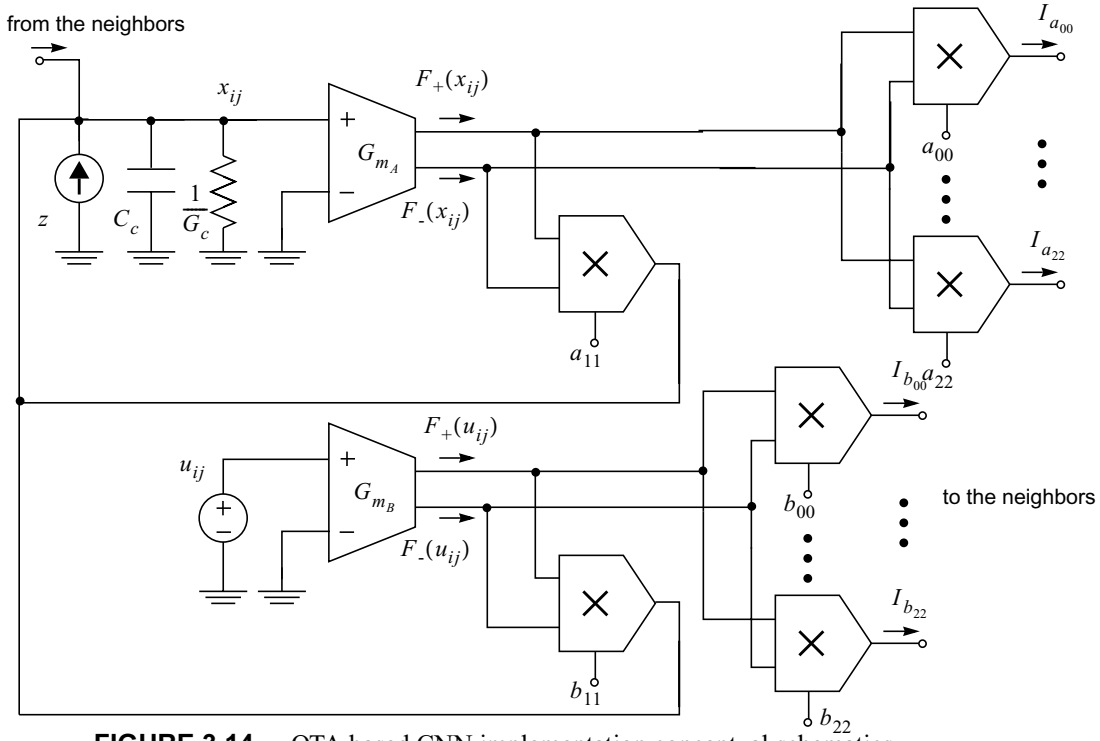


FIGURE 3.14. OTA based CNN implementation conceptual schematics.

achieve the task they were designed for [Espe94a]. Currents, representing these neighbour contributions and the self-feedback components, are added by wiring them together at the input node of each cell. Dynamic evolution of every cell within the grid is described by the following equation:

$$\tau\frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + z + \frac{1}{G_c}\left\{\sum_{k=-r}^{r}\sum_{l=-r}^{r}[G_{m_A}(a_{kl}, x_{(i+k)(j+l)}) + G_{m_B}(b_{kl}, u_{(i+k)(j+l)})]\right. \tag{3.11}$$

where G_{m_A} includes nonlinear operation and multiplication. Fig. 3.15 displays the schematic of the OTA-multiplier block implementation. Equations describing the large-signal characteristic of this circuit can be derived assuming that the transistors associated in pairs are matched. Thus, the output current of the kl-th multiplier of the ij-th can be approximated as a piece-wise-linear (PWL) function of the state variable with this form:

$$I_{a_{kl},\,ij} \cong \begin{cases} \sqrt{\beta_p \cdot I_B}\cdot a_{kl} & \text{if} \quad x_{ij} > \sqrt{\dfrac{2I_B}{\beta_n}} \\ \sqrt{\dfrac{\beta_p \cdot \beta_n}{2}}\cdot a_{kl}\cdot x_{ij} & \text{if} \quad |x_{ij}| \le \sqrt{\dfrac{2I_B}{\beta_n}} \\ -\sqrt{\beta_p \cdot I_B}\cdot a_{kl} & \text{if} \quad x_{ij} < -\sqrt{\dfrac{2I_B}{\beta_n}} \end{cases} \tag{3.12}$$

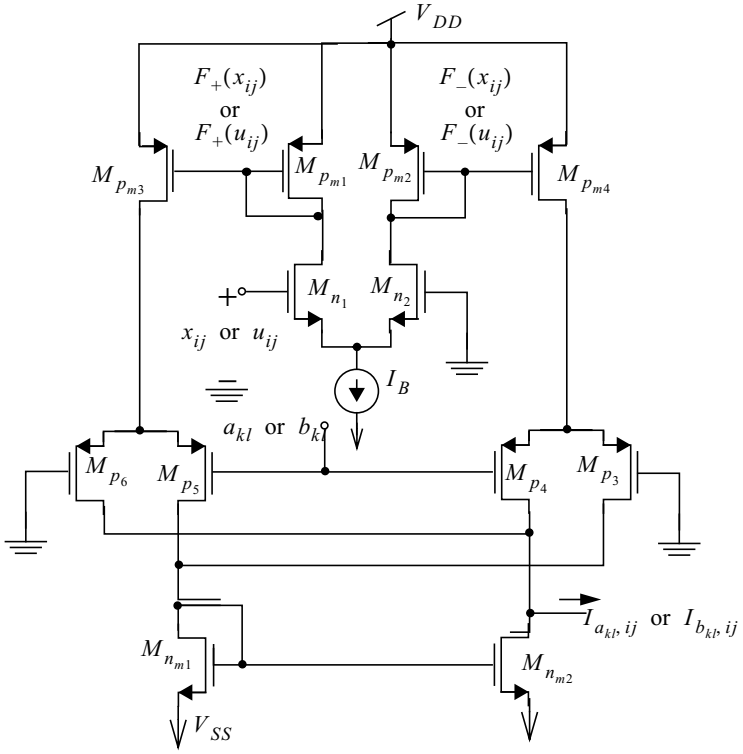


FIGURE 3.15. CMOS realization of the OTA-multiplier block.

and, alternatively, for the feedforward template (B template):

$$I_{b_{kl},ij} \cong \begin{cases} \sqrt{\beta_p \cdot I_B} \cdot b_{kl} & \text{if} \quad u_{ij} > \sqrt{\dfrac{2I_B}{\beta_n}} \\ \sqrt{\dfrac{\beta_p \cdot \beta_n}{2}} \cdot b_{kl} \cdot u_{ij} & \text{if} \quad |u_{ij}| \le \sqrt{\dfrac{2I_B}{\beta_n}} \\ -\sqrt{\beta_p \cdot I_B} \cdot b_{kl} & \text{if} \quad u_{ij} < -\sqrt{\dfrac{2I_B}{\beta_n}} \end{cases} \tag{3.13}$$

where the saturation limits has been set to a value that becomes the normalizing factor for Eq. 3.11 in order to represent the evolution of a CNN. Template elements have to be redesigned also because of this equation scaling [Fold96]. In order to describe the network, variables and parameters are first formally declared (Fig. 3.16 shows file **ota.cs** for SIRENA modelling). In this occasion, auxiliary matrices are created to evaluate neighbours' contribution. Also some CMOS parameters necessary for the OTAs equations are defined. After that the evolution section, that contains the relations between the network variables and the differential equation, is described. Finally the single-layer network definition is stated.

The following table shows the values assigned to each variable and parameter, extracted from HPICE models of the CMOS devices and the

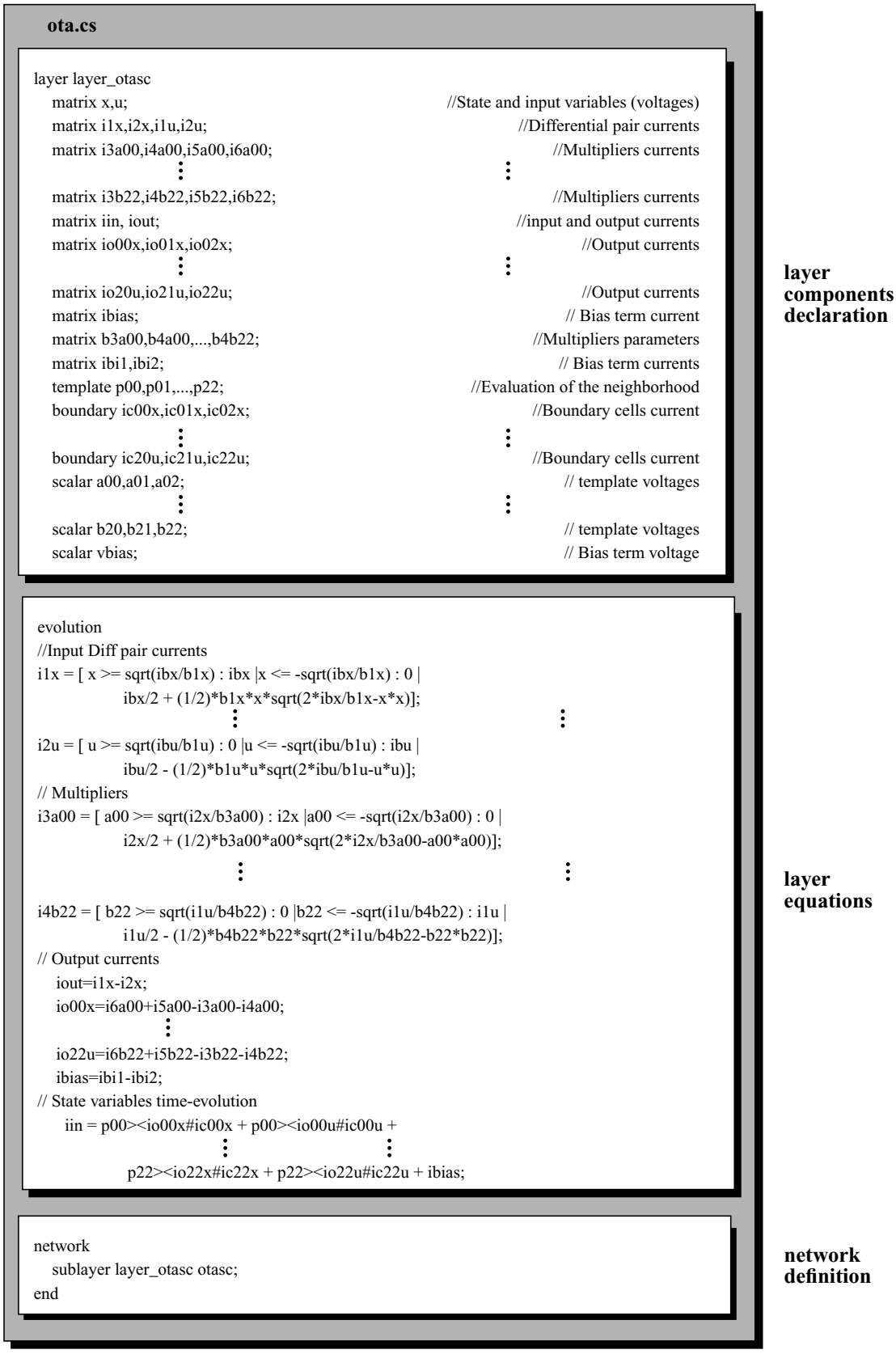


FIGURE 3.16. DECEL description of the OTA-based CNN model.

technology files:

Variable, Parameter	Name	Value	Units
Input and initial state (max)	x, u	0.50	volts
Input and initial state (min)	x, u	-0.50	volts
Diff. pairs bias currents (I_B)	ibx, ibu	5.0×10^{-6}	amps
NMOS transcond. ($\beta_n = \beta_n^* \frac{W}{L}$)	b1x,...,b2u	240×10^{-6}	A/V^2
PMOS transcond. ($\beta_p = \beta_p^* \frac{W}{L}$)	b3a00,...,b6b22	60×10^{-6}	A/V^2
Template elements (unity)	a00,...,b22	0.3	volts
Bias term voltage (unity)	vbias	0.3	volts
Boundary currents (unity)	ic00x,...,ic22u	3.0×10^{-6}	amps
Time constant (τ)	tau	250×10^{-9}	seconds
Normalizing transcond. (G_c)	gc	20×10^{-6}	mhos

TABLE 3.3. OTA based CNN model parameters.

Discrete-time CNN model with nonlinear interactions

Finally, in order to illustrate the ample variety of network types that can be described in DECEL language, let us consider the case of a discrete-time CNN chip to perform the Radon Transform operation [Espe93b]. Some peculiar properties of this algorithm include the hard-limitation of the cell output variable and the use of non-linear template elements —which depend of the cell state. This operation consists on computing the integral of intensity values along each row of the network. For binary images, this results in a histogram of the input pattern. The discrete-time evolution of every cell belonging to the network is now described by a system of nonlinear finite-differences equations,

$$x_{ij}(n+1) = \sum_{k=-r}^{r} \sum_{l=-r}^{r} a_{kl}[y_{ij}(n)] y_{(i+k)(j+l)}(n) \tag{3.14}$$

where the non-linear feedback template is described by

$$\mathbf{A}(y_{ij}) = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1-y_{ij}}{2} & 0 & \frac{1+y_{ij}}{2} \\ 0 & 0 & 0 \end{bmatrix} \tag{3.15}$$

and where the output variable is obtained through a hard-limiting non-linear function:

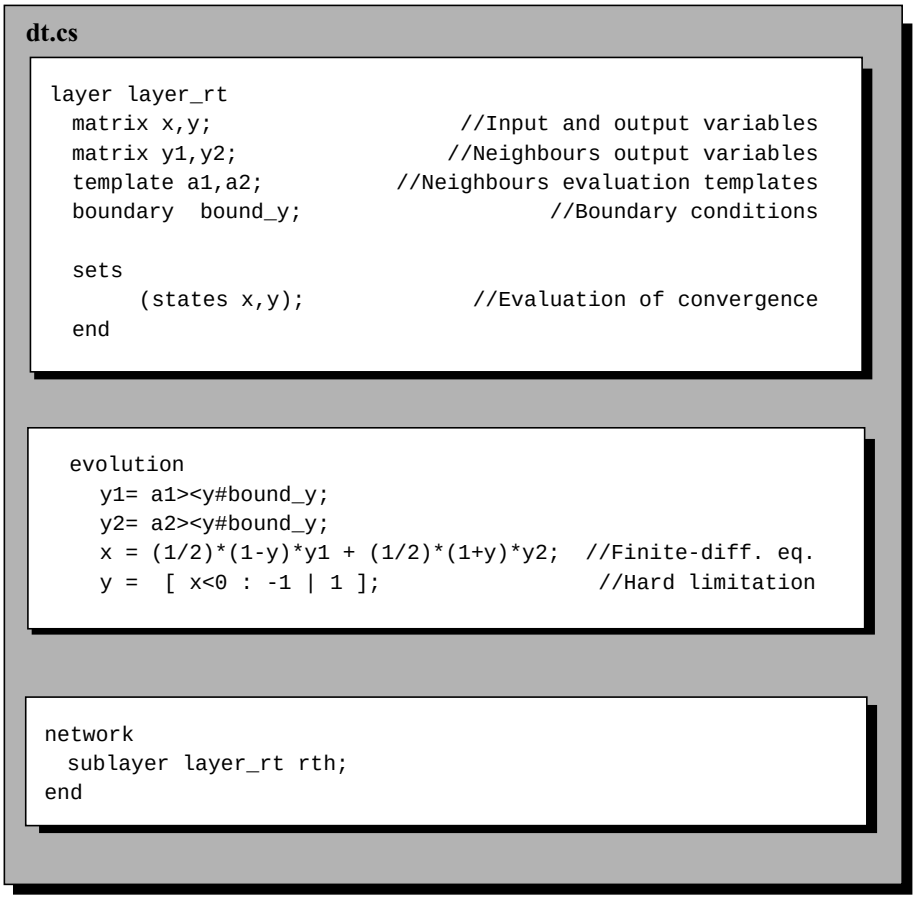


FIGURE 3.17. DECEL description of the DT-CNN model

$$y_{ij}(n) = f[x_{ij}(n)] = \begin{cases} 1 & \text{if} \quad x_{ij}(n) > 0 \\ -1 & \text{if} \quad x_{ij}(n) \leq 0 \end{cases} \tag{3.16}$$

Substantial differences can be observed between the model of this network and those models reported before. First of all, this is a discrete-time CNN. This forces the definition of a set of variables within the DECEL description of the model (file **dt.cs** in Fig. 3.17) in order to evaluate convergence during the simulation process. Feedback-template dependence on the cell output must be included in the evolution law of the model, and therefore the connection operator must store neighbours output variables into separate matrices. The last part of the model description is the network definition.

Input and output patterns of SIRENA simulation are shown in Fig. 3.9 as depicted by the Graphical User Interface.

3. 1. 5. Performance comparison

In this section, the results obtained by the simulation with SIRENA of the previously reported CNN models are compared to those given by HSPICE (v. 95.1). First, a qualitative analysis based on the inspection of the waveform plots validates the cor-

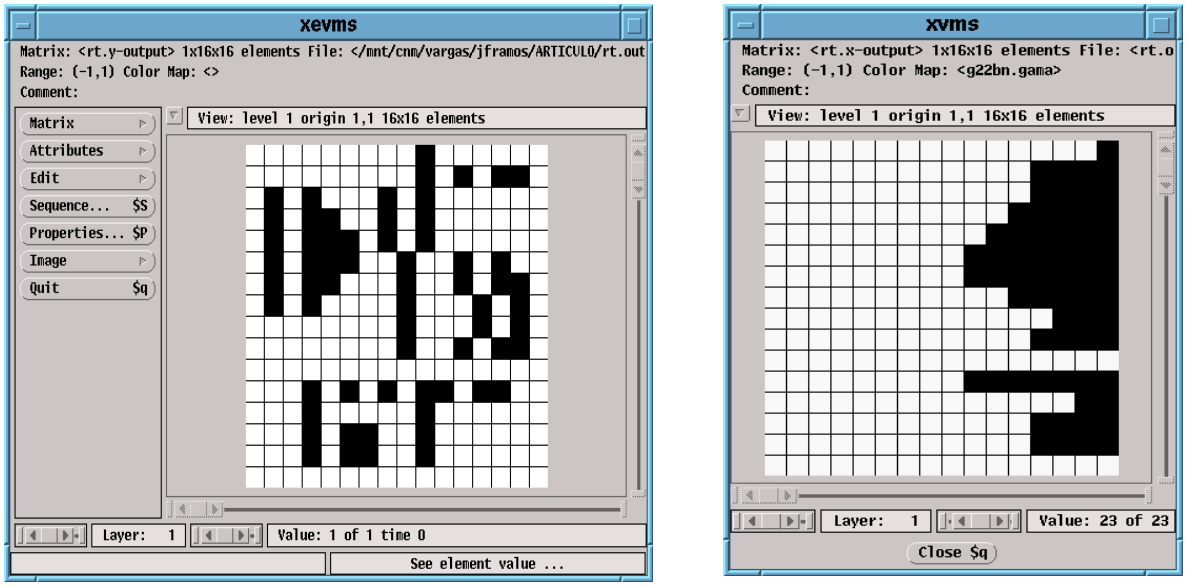


FIGURE 3.18. Input and output patterns of the Radon Transform CNN simulation as depicted by the Graphical User Interface.

rectness and accuracy levels of the proposed alternative simulation tool. After that, some comments about the use of the computing resources by both implementations are quantified with the help of some monitoring commands of the UNIX system. The evolution of different networks performing CCDet, with network sizes ranging from a 2×2 to a 32×32 matrix of cells, and based on the three models depicted before has been simulated in a Sun Microsystems SPARC Server 1000E with 4 CPUs and 512MB RAM.

Thus, Fig. 3.19 shows the waveform plots of the cell state variable (x) and the cell output variable (y) generated by HSPICE and SIRENA for the simulation of a 4×4 CNN based on the Chua-Yang model. G_c and C_c have been chosen to have a time constant of 1μs.

Fig. 3.20 follows with the waveforms of the state variable (x) and the output (y_p) of a 4×4 current-mode CNN given as results of the simulation of the circuit in HSPICE, using level-two models for MOS devices, and in SIRENA using the macro-model described before. Notice that the sign of the y_p signals is reversed from the state variable sign, due to the inversion of the current sense imposed by mirroring. Plots begin at 100ns because of initialization of the network.

In the first case, both simulators display quite the same output plot, the only difference found in the breaking points of the nonlinear output variable, due to a different integration method. In the current-mode case. Similarity between each of the sixteen pairs of curves gives an idea of the accuracy of SIRENA model of this circuit. However, a higher degree of precision can be reached with some model refining and the inclusion of some second order effects (channel-length modulation, body-effect, etc.).

Finally, a 4×4 OTA-based CNN simulation is shown in Fig. 3.21, again SIRENA output exhibit an strong resemblance of the HSPICE plots.

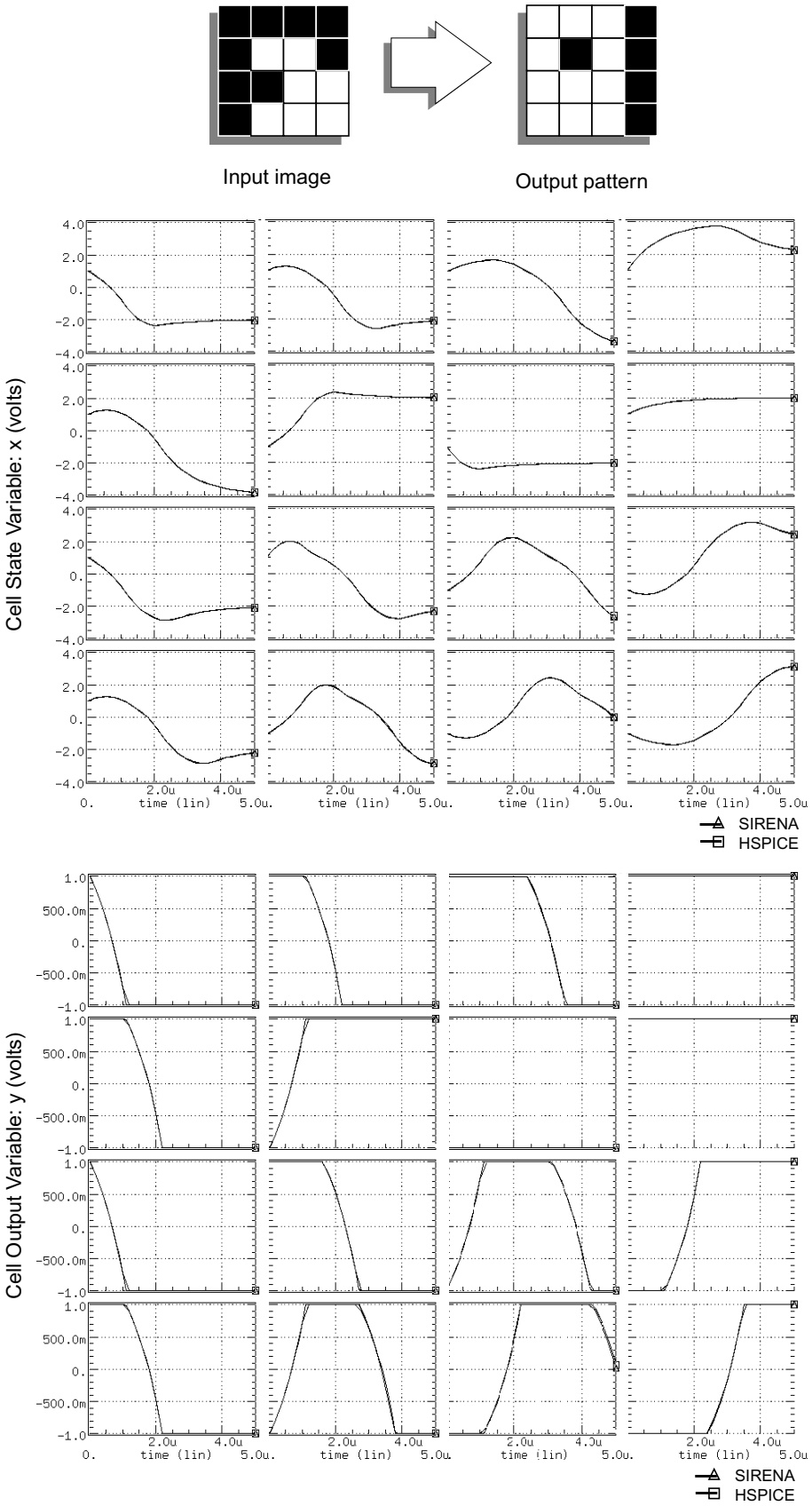


FIGURE 3.19. Simulation waveforms of a 4×4 cells CNN for Connected Component Detection using Chua-Yang original CNN model given by HSPICE (v.95.1) and SIRENA.

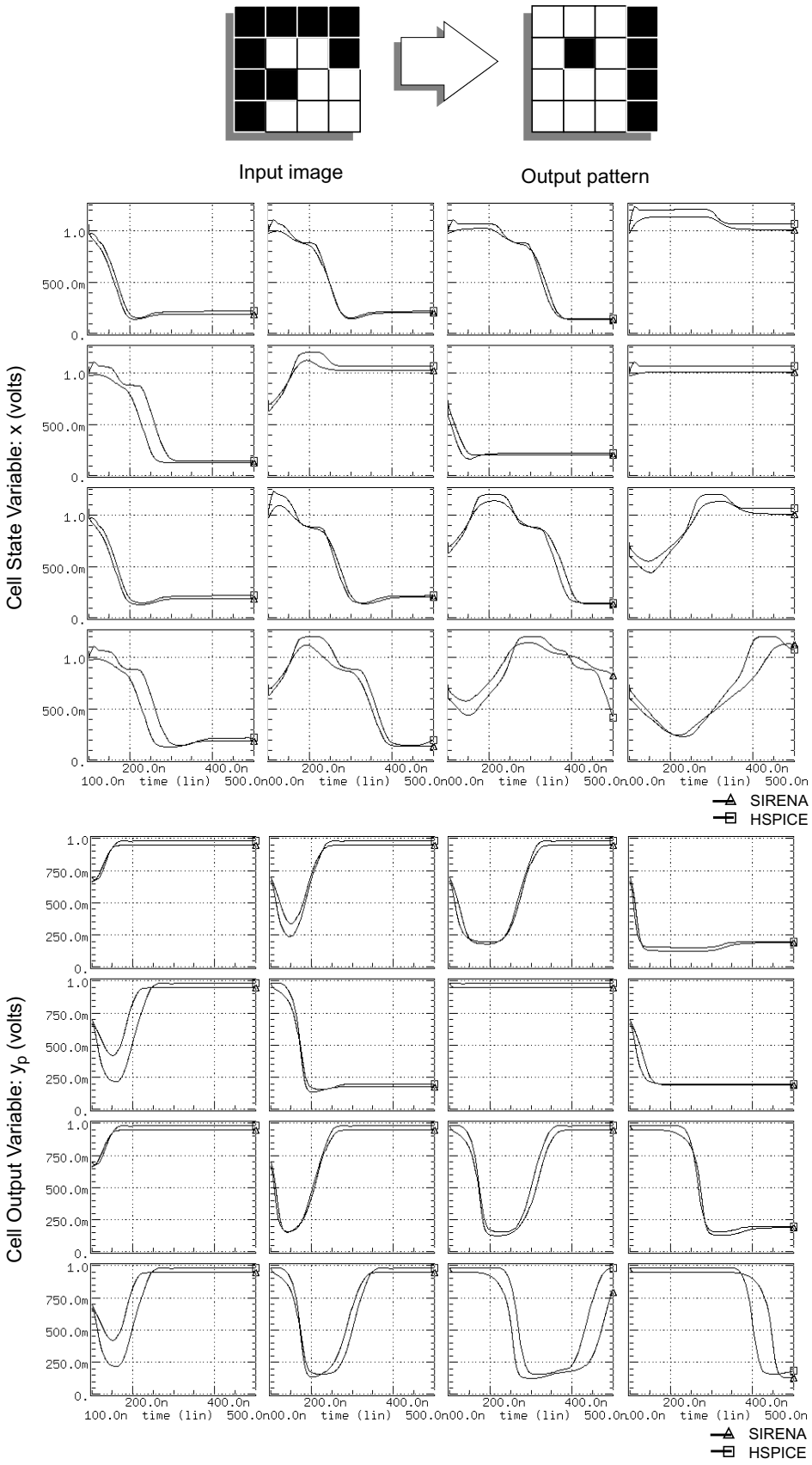


FIGURE 3.20. Simulation waveforms of a 4×4 cells Current-Mode CNN for Connected Component Detection given by HSPICE (v.95.1) and SIRENA.

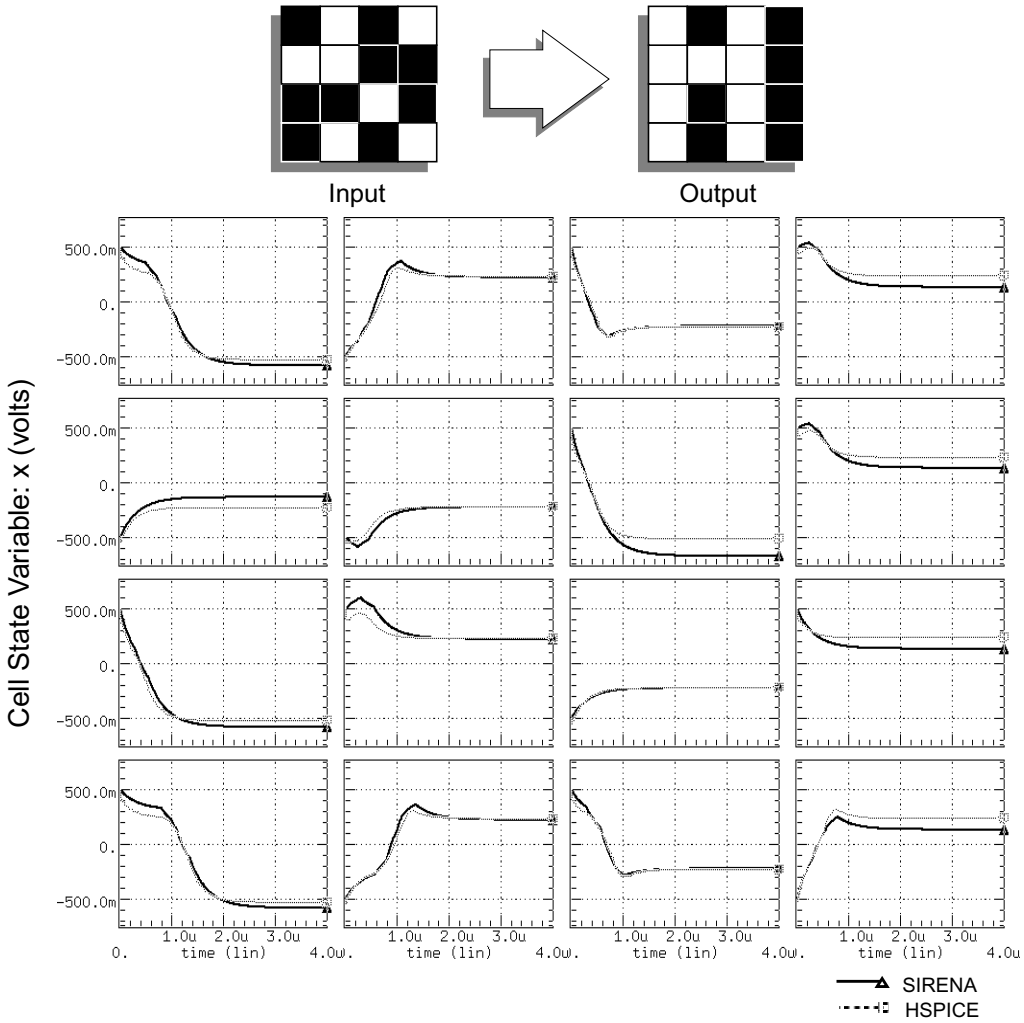


FIGURE 3.21. Waveforms of state variable of an OTA based 4×4 CNN performing Connected Component Detection of an input pattern, simulated by HSPICE and SIRENA.

After several simulations of different size CNNs, realized in the same machine (SPARC Server 1000E) with the help of HSPICE (v. 95.1) and SIRENA, a comparison of the CPU time consumption can be made (Fig. 3.22). From these data a considerable advantage on using SIRENA is revealed, reaching CPU time savings of even two orders of magnitude for the simulation of larger networks. Thus, the efficiency of this environment within the scope for which it was developed results evident. For the OTA-based model, a 4×4 cells CNN simulation in a SPARC Station 10/51 with 32 MB RAM, needs 924kB RAM and 61.5 CPU seconds for SIRENA while 7452kB and 279.8 CPU seconds are required by HSPICE.

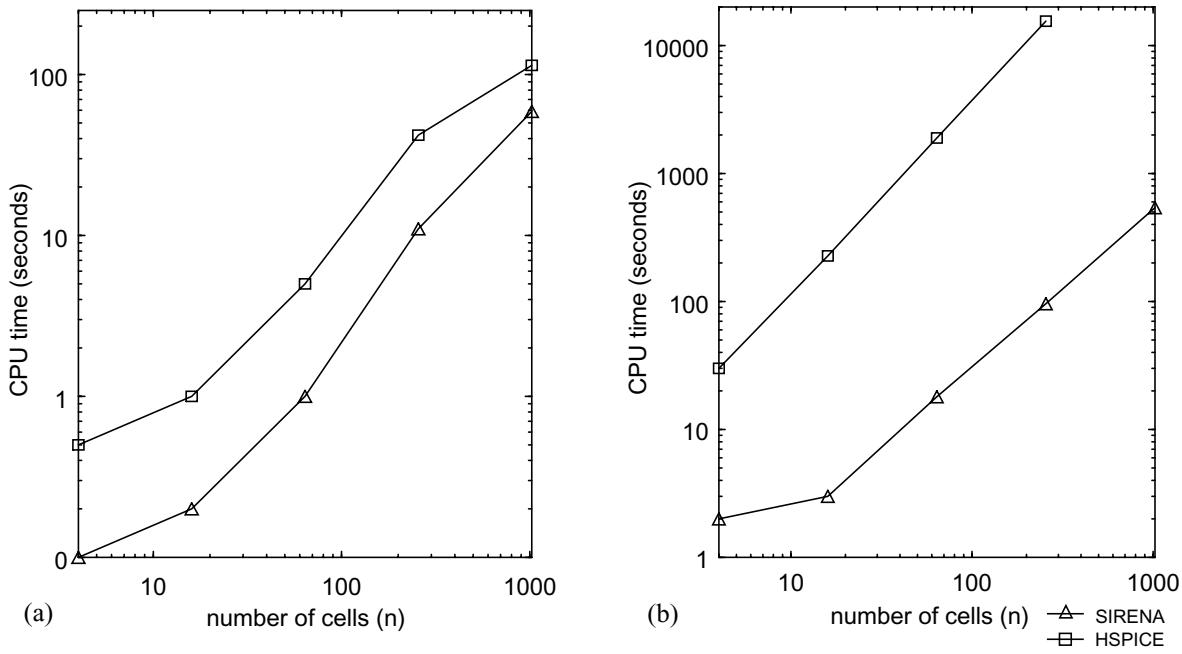


FIGURE 3.22. Log-log graphs of the CPU time vs. the number of cells, for (a) the Chua-Yang CNN model, and (b) the current-mode CNN model.

3. 2. CNN template robustness and design centering

Once the basic tools and methodology for the evaluation of the performance of a CNN-based system are presented, let us see how these tools can be employed for further improvement of the robustness of the implementation. For this purpose, one of the principal tasks to be developed is the generation of a set of templates —which are the tunable parameters that characterize the network operation— with tolerances large enough to ensure feasibility of the physical realization. In other words, any real implementation of an algorithm fails to match the ideally designed parameters to some extent, thus, let us provide a set of parameters that maintain the same network operation even in the case that some of them deviate from their nominal values. Actually, we will be interested in that set of parameters that allows larger deviations. This can be visualized as moving a point in the space of the network parameters to the centre of a region in which any point represents a set of parameters coding the same valid operation, what is design centring.

So, first, we will see how this intuitive concept relates with the dynamic routes of an actual network model, and then, we will employ the tools described before to perform design centring automatically. We will review also how this methodology can be extended to the automatic generation of templates for realizing some specific tasks.

3. 2. 1. Dynamic routes and robustness

Dynamics of the Chua-Yang model

As referred in Chapter 1, dynamic routes of a cell in a CNN can vary along the network evolution because of the dependence of these routes on the actual state of the neighbours in any time instant. This holds true for the more general case, since this occurs for coupled CNNs —those with more non-zero elements apart from the central element in the feedback template— but it does not happen for uncoupled CNNs. This fact leads to very complex dynamics, for which some preliminary assumptions must be realized in order to extract easily some practical information out of the mathematics involved. This primary assumption in the study of one single cell dynamic routes is to consider that the rest of the array has already arrived to steady-state. It can be pointed out that guaranteeing that all the cells separately converge to steady-state is not a sufficient condition for the convergence of the whole network, but it is a necessary condition indeed. Thus, consider that every cell in a Chua-Yang CNN but the one under study have converged to steady-state. Let us now re-write the equation that governs the cell dynamics:

$$\tau \frac{dx_{ij}(t)}{dt} = -Gx_{ij}(t) + a_{00} f(x_{ij}) + k_{ij} \tag{3.17}$$

As we did before, we have grouped the contributions of the neighborhood and the bias in the term k_{ij}:

$$\begin{aligned} k_{ij} = & \sum_{\substack{k=-r \\ k\neq 0}}^{r} \sum_{\substack{l=-r \\ l\neq 0}}^{r} a_{kl} \cdot y_{(i+k)(j+l)} + \\ & + \sum_{k=-r}^{r} \sum_{l=-r}^{r} b_{kl} \cdot u_{(i+k)(j+l)} + z_{ij} \end{aligned} \tag{3.18}$$

Also notice that the losses term slope have been chosen to be G, that can be different than 1, which is the ideal case. In these conditions, the network will evolve from its initial state following one of the routes depicted in Fig. 3.23. It has been considered that $|a_{00}| > G$, which is a necessary condition for convergence to one of the saturated states, as reported in [Chua88]. See that depending on the value of k_{ij}, which is fixed for uncoupled CNNs but varies through time for coupled CNNs —those exhibiting propagation of signals throughout the array, we will have routes with one to three equilibrium points. There will be always one stable equilibrium point at least, but there can be another if $-(a_{00} - G) < k_{ij} < a_{00} - G$ holds. In this case, the steady-state is also determined by the initial conditions.

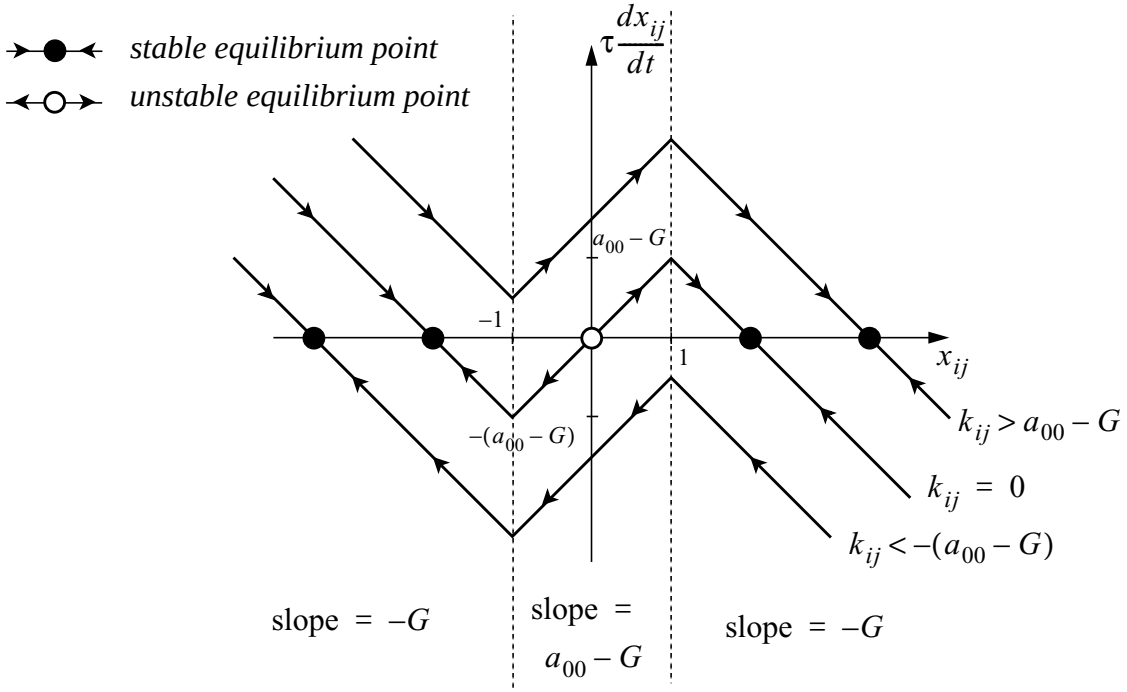


FIGURE 3.23. Dynamic routes of the state variable in the Chua-Yang CNN model.

Changes in the dynamic routes

To see how an inexact realization of the network can seriously compromise functionality, let us examine the effect that minute deviations on the values of the network parameters have on the predicted dynamic routes [King96]. This is because the functionality of a CNN with some specific connection weights is defined by the convergence to a some specific steady-states when evolving from some specific initial states. Thus, any qualitative changes in the dynamic routes can modify the steady-state corresponding to a specific initial state, and, in this manner, the functionality of the network can be lost.

Namely, any deviation in the value of k_{ij}, because any of the terms in Eq. 3.18 has been wrongly implemented, has the consequence of shifting dynamic routs in the vertical axis (Fig. 3.24(a)). On the other hand, any deviation in the losses-term slope, G, produces a shift in the position of the stable equilibrium points (Fig. 3.24(b)). Finally, changes in the central element of the feedback template, a_{00}, cause a different slope in the inner piece of the routes to appear and shift the stable equilibrium points (Fig. 3.24(c)).

In these conditions, it can be said that the network will evolve properly to a correct steady-state for a given set of templates if the shifting of the dynamic routes and the equilibrium points do not alter the observable behaviour of the cells. This is, as long the nature of the corresponding dynamic routes remains unchanged. There is a margin, for each dynamic route, beyond which, the quality and quantity of the equilibrium point can change. This margin, that we will call $\Delta_{\max}$, is characteristic for each set

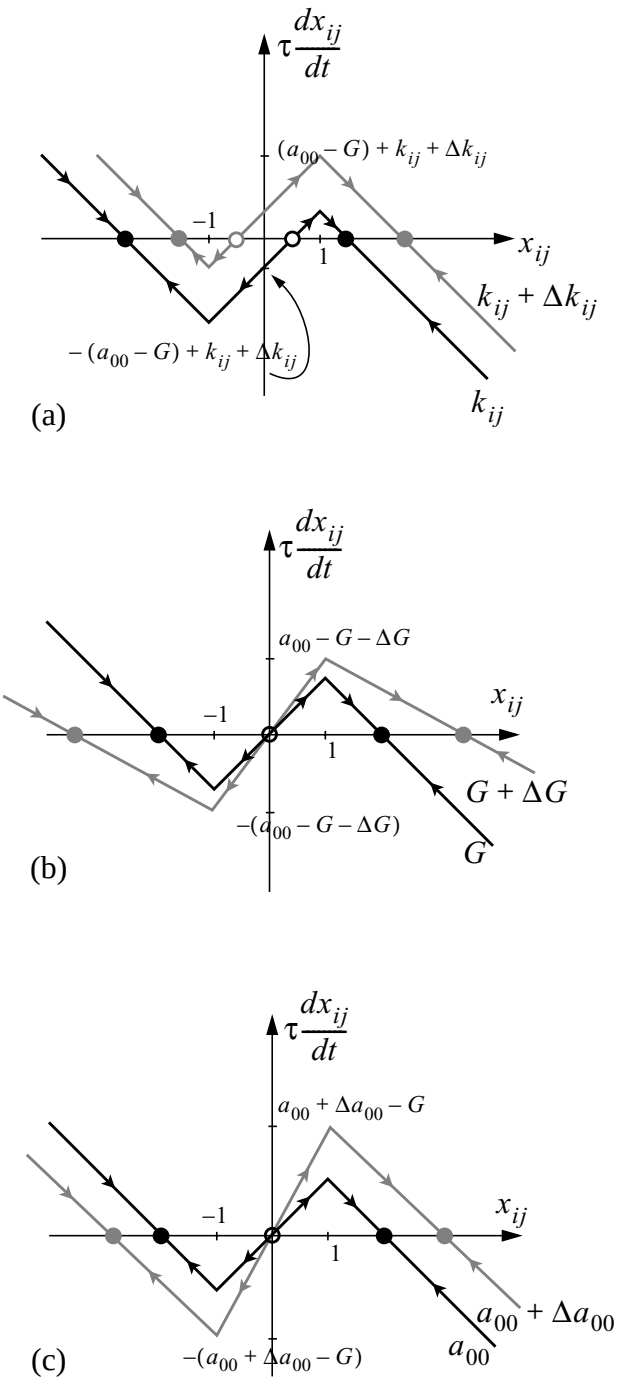


FIGURE 3.24. Effects of the deviations of some network parameters on the dynamic routes of the state variable: (a) deviation on feedback and feed-forward templates and the bias term, (b) deviation of the slope of the losses-term, y (c) deviation of the self-feedback coefficient.

of templates. The actual route described by the real network deviates from the ideal route by a quantity Δ_{route}, which, from what has been exposed, can be obtained by adding up the errors originated in the implementation of the template elements:

$$\Delta_{\text{route}} = \Delta G + \Delta z + \\ + \sum_{k=-r}^{r} \sum_{l=-r}^{r} (a_{kl} + b_{kl}) \tag{3.19}$$

where all the errors are supposed to push in the same direction. Therefore, as long as this relation holds:

$$\Delta_{\text{route}} < \Delta_{\text{max}} \tag{3.20}$$

the network functionality is preserved. Consequently, the larger Δ_{max} is, the larger the deviations of the template elements from their nominal values that can be permitted, thus, the more robust the implementation will be. Or, equivalently, the larger the difference $\Delta_{\text{max}} - \Delta_{\text{route}}$ is — where Δ_{route} have been computed considering the tolerances expected for a specific technology, the better the implementation will be. Hence, this is a measure of the goodness of an implementation in a particular technology based on inspection of dynamic routes.

As an example, let us analyse two cases, the CCDet [Mats90] and the Edge Detector [Rosk95]. For both of them, a critical route has to be identified, that is the one with a smaller theoretical Δ_{max}.

For the first case, the CCDet, with templates:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & -1 \end{bmatrix} \\ \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \qquad z = 0 \tag{3.21}$$

the Eqs. 3.17 and 3.18 can be re-written as:

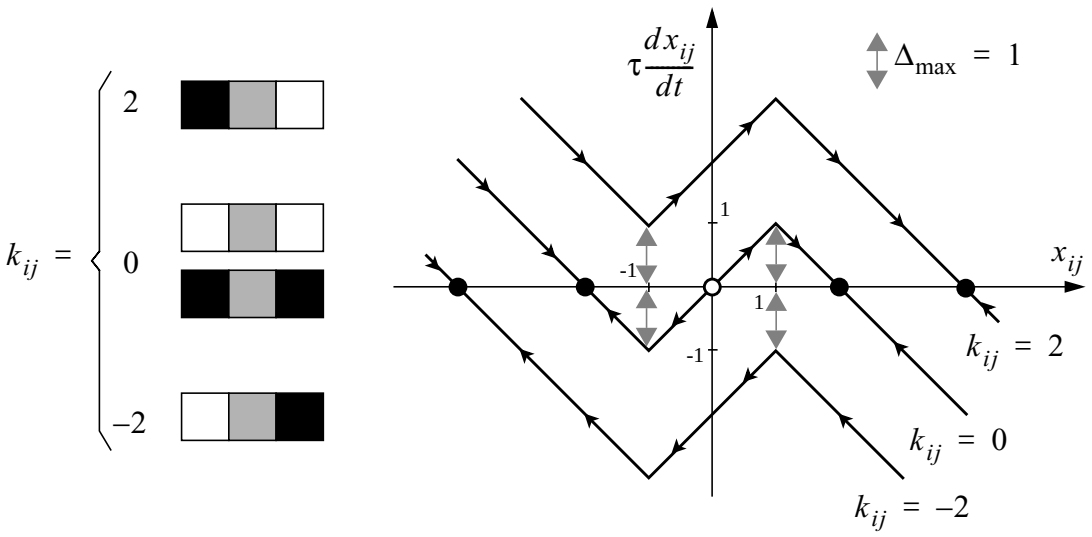


FIGURE 3.25. Dynamic routes for the CCDet in the Chua-Yang CNN model.

$$\tau\frac{dx_{ij}}{dt} = -x_{ij} + 2f(x_{ij}) + f(x_{i-1,\,j}) - f(x_{i+1,\,j}) \tag{3.22}$$

it means that a cell with a stationary neighbourhood can be in any of the four possible situations depicted in Fig. 3.25, independently of its own state. Each of the three possible routes displays a $\Delta_{max} = 1$, what means that if the three template elements deviate in a correlated fashion, a simultaneous deviation of even the 25% of the three elements can be permitted without causing the network to misfunction.

In the case of the edge extraction, the following templates are used:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \qquad z = -1.5 \tag{3.23}$$

Now the sum of the neighbours' contribution plus the bias term is:

$$k_{ij} = \sum_{k=-r}^{r} \sum_{l=-r}^{r} b_{kl} \cdot u_{(i+k)(j+l)} + z_{ij} \tag{3.24}$$

The following table can be computed for the values of k_{ij}:

Cell state	Neighbours	k_{ij}	Δ_{max}
■ = 1	□□□□□□□	1.5	0.5
■ = 1	□□□□□□■	1.0	0.0
■ = 1	□□□□□■■	0.5	0.5

TABLE 3.4. Values of the term k_{ij} for the Edge Extraction templates.

Cell state	Neighbours	k_{ij}	$\Delta_{\max}$
■ = 1		0.0	1.0
■ = 1		–0.5	0.5
■ = 1		–1.0	0.0
■ = 1		–1.5	0.5
■ = 1		–2.0	1.0
■ = 1		–2.5	1.5
□ = –1		≤ -1.5	≥ 0.5

TABLE 3.4. Values of the term k_{ij} for the Edge Extraction templates.

In order to understand the computed $\Delta_{\max}$ for each route, think that the route corresponding to $k_{ij} = 0$ (which is the consequence of having a high cell state and 3 black neighbours) is the one drawn in the middle of the diagram of Fig. 3.26. Then, it can be easily deducted that $|k_{ij}| = 1$ implies a critical route with zero tolerance, as can be seen in Fig. 3.26. This happens actually in cases in which the definition of the edge is not clear. Let us look at the situations depicted in Fig. 3.26(a) and Fig. 3.26(b) and decide whether the centre cell should be considered part of the picture border or not. The question is not easy to solve. Anyway, once we have established a criterion, a new set of templates can be generated with an improved tolerance for the critical routes. Yet by shifting the routes in Table 3.4 by 0.25, what can be achieved by tuning b_{00} to 2.25 for instance, we get $\Delta_{\max} = 0.25$, which is infinitely better than zero.

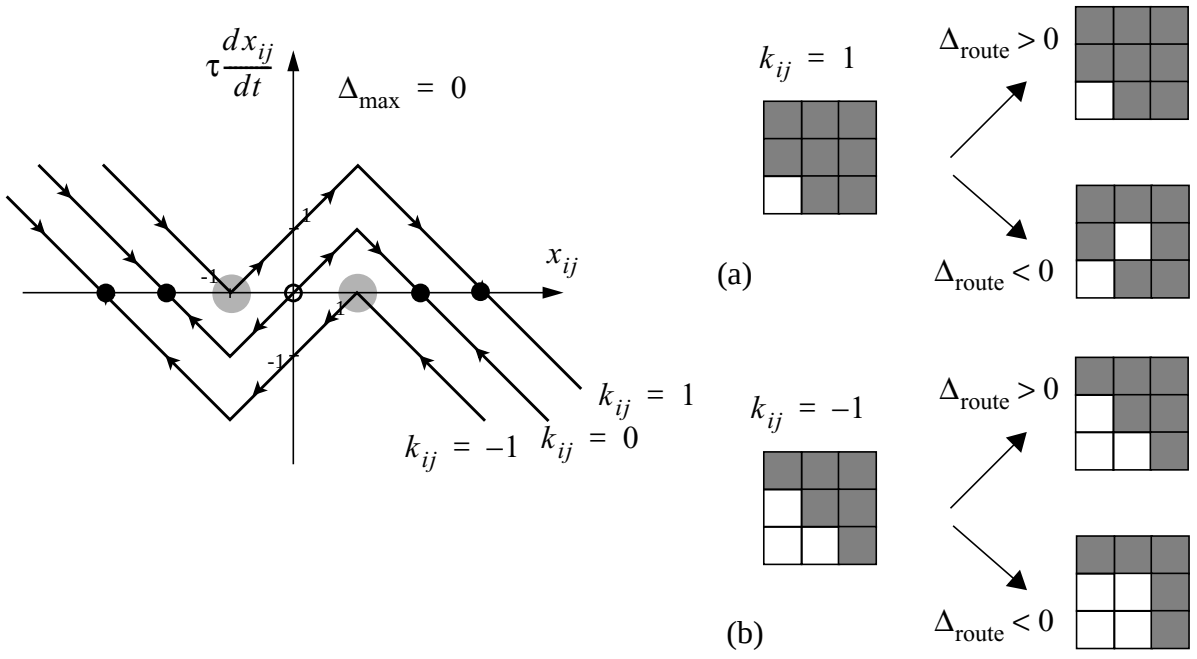


FIGURE 3.26. Dynamic routes for the edge detector in the Chua-Yang CNN model..

Summarizing, a detailed inspection of the dynamic routes can point out some critical aspects that endangers the physical implementation of the CNN operation. It is convenient to redefine the nominal values for network parameters in order to enlarge the design room, thus making the implementation more robust. However, sometimes this is not an easy task. Sometimes the CNN operates on a grayscale image and the training set of input-output pairs may be quite bulky. Automatic redefinition of a set of templates will be desirable. For this job, we are going to make intensive use of the previously described tools.

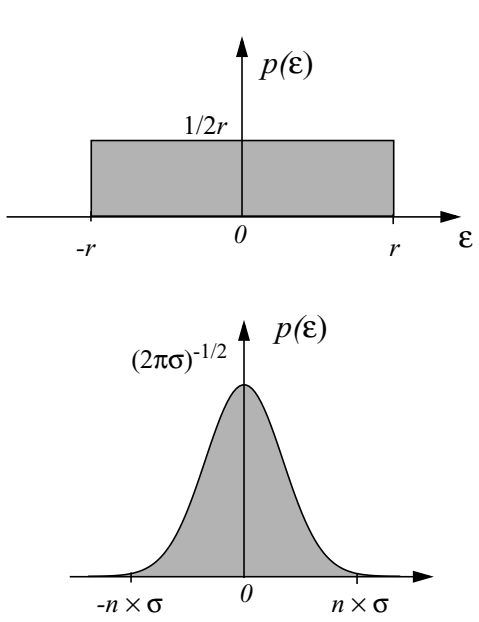
3. 2. 2. Automatic design centring with SIRENA

Parameters scattering in VLSI technologies

One of the most important tasks in the verification of high-complexity analog VLSI designs is the statistical analysis of the implementation. Special effort is required to consider the unavoidable deviation of IC fabrication process parameters that can lead to extremely low production yield (see Chapter 2). It is a well-known fact that performance of a large number of analog circuit primitives in modern IC technologies, like CMOS and BiCMOS, relay in the matched characteristics of highly nonlinear semiconductor devices. Assumptions based on the likelihood of equally designed transistors lead to simplified and manageable expressions for the circuit variables. But, unfortunately, matching of two identically designed devices is limited by the unevenness of the fabrication process. Deviation of the characteristic parameters of the standard CMOS process, e. g. oxide thickness, dopant atoms density, etc. induce an equivalent deviation of the electrical parameters of the different semiconductor devices, say, the extrapolated threshold voltage (V_{T0}), the current-factor (transconductance factor β) and the body-effect constant (γ), from their nominal values [Pelg89]. In addition, batch-to-batch and wafer-to-wafer impairities lead to slightly different nominal values, this time affecting the whole network in the same manner.

Consequently, statistical characterization of the technology parameters and their influence on the network performance is required to achieve a robust design and to ensure a fully functional and highly efficient hardware implementation. This can be realized via sensitivity analysis, in which critical parameters (P) are swept through an equally separated discrete set of values, or Monte Carlo analysis, where parameter values are generated randomly following a specific probability distribution, *e. g.* a normal or gaussian distribution, or a uniform distribution (Fig. 3.27). The error or deviation of each parameter from its nominal value (ΔP) is characterized by a certain standard deviation, while the mean value is zero:

$$P = P_0 + \Delta P = P_0\left(1 + \frac{\Delta P}{P_0}\right) \qquad \text{where} \quad \Delta P = \varepsilon_{abs} = P_0 \cdot \varepsilon_{rel} \tag{3.25}$$



Uniform error probability distribution:

$$\varepsilon \in (-r,r)$$

$$p(\varepsilon) = \frac{1}{2r}$$

Zero-mean gaussian distribution:

$$\varepsilon \in (-n \times \sigma, n \times \sigma)$$

$$p(\varepsilon) = \frac{1}{\sqrt{2\pi\sigma}} \cdot e^{-\frac{\varepsilon^2}{2\sigma^2}}$$

FIGURE 3.27. Error probability functions available in SIRENA for emulation of parameters deviation from their nominal values.

The evaluation of the tolerances than can be expected for a particular process and their influence on the behaviour of a specific circuit design is a prerequisite for the fabrication of high-complexity analog VLSI circuits, especially those employing small devices. It is however extremely expensive, in terms of CPU time, to perform a substantial number of simulations of large networks. Its realization with traditional SPICE-like electrical simulators is, sometimes, virtually impossible. Therefore, these tasks need to be realized with the help of efficient and specific simulation tools. Let us illustrate this with two examples developed with SIRENA.

Design centring example

SIRENA is able to perform multiple transient analysis of the network evolution, sweeping the value of a single parameter or variable, or a set of them, with a specified increment and range. In this way, sensitivity of the network to deviation of different parameters can be evaluated and tolerance margins can be defined to complete characterization of a specific hardware implementation. On the other side, random distribution of parameter values can be introduced into multiple simulations to perform Montecarlo analysis of the network. These outstanding simulation abilities can be used in the optimization of the robustness of the algorithm against the anomalies presented by a specific implementation, *i. e.* the referred design centring. Robustness of an algorithm means a certain degree of immunity from fabrication process unevenness. Thus, a specific algorithm is said to be robust against some particular implementation failures if there exists a wide enough interval of values that each algorithm parameter can deviates without causing loss of functionality nor improper performance. Let us consider the templates for binary images edge detection reported before. Nominal values

of the template elements given for that operation were:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \qquad z = -1.5 \quad (3.26)$$

This set of templates, as pointed out by the analysis of the critical dynamic routes, will lead to network misfunction in some particular situations —for some specific input-output pairs— for any real implementation of the CNN. Actually, during verification of multianalysis capabilities of SIRENA, the implementation of both Chua's or Full-Signal-Range model [Espe94b] based CNNs with this set of templates scored an unexpected, and extremely low, rate of unerring samples. Either using a globally or a locally distributed deviation of the offset term (z) to perform Monte Carlo analysis results were discouraging. Obviously, modifications of any of the rest of the parameters would lead to failure as well, as already observed in the previous section, but we are going to concentrate here in the offset term. Therefore, the problem was restated as finding the optimum centre value for z, that is, the one which will display a larger tolerance margin. It is confirmed by SIRENA simulation results that the value that has been applied to the offset term, $z = -1.5$, lays on the verge of the tolerance range of that specific parameter. From the sensitivity analysis of a set of different sizes CNNs ranging from 8×8 to 32×32 cells, a tolerance margin below 0.01% of the nominal value for that offset term was detected —theoretically it must be zero. In these analysis this parameter is swept, within a radius of 60% the nominal value, through 50 equidistant values. After that, the outputs are compared with the expected output pattern with the help of a simple program. This is employed to generate a new central value, which is now tested by Montecarlo analysis of the network to confirm the results given by the sensitivity analysis. With the results of Montecarlo analysis for different values of the offset term standard deviation (σ_z), the interpolated parametric yield curves of Fig. 3.28 are composed. They represent the probability of correct operation of the network versus σ_z. It can be seen that $z = -1.195$, that has been computed by a guided optimization algorithm, displays a much more robust behaviour than the initial value ($z = -1.50$).

Yield estimation based on the statistical characterization of the process

In a second experiment, the influence of the parameters deviation on the network performance is evaluated for an actual hardware implementation. Let us consider the OTA based CNN model described in Sect. 3. 1. 4. The output current of the OTA-block of Fig. 3.15 (given by Eqs. 3.12 and 3.13) has been derived under the assumption that transistors M_{n_1} and M_{n_2} have the same transconductance parameter ($\beta_n = \mu_o \cdot C^*_{ox} \cdot (W/L)$) and threshold voltage (V_{T_n})

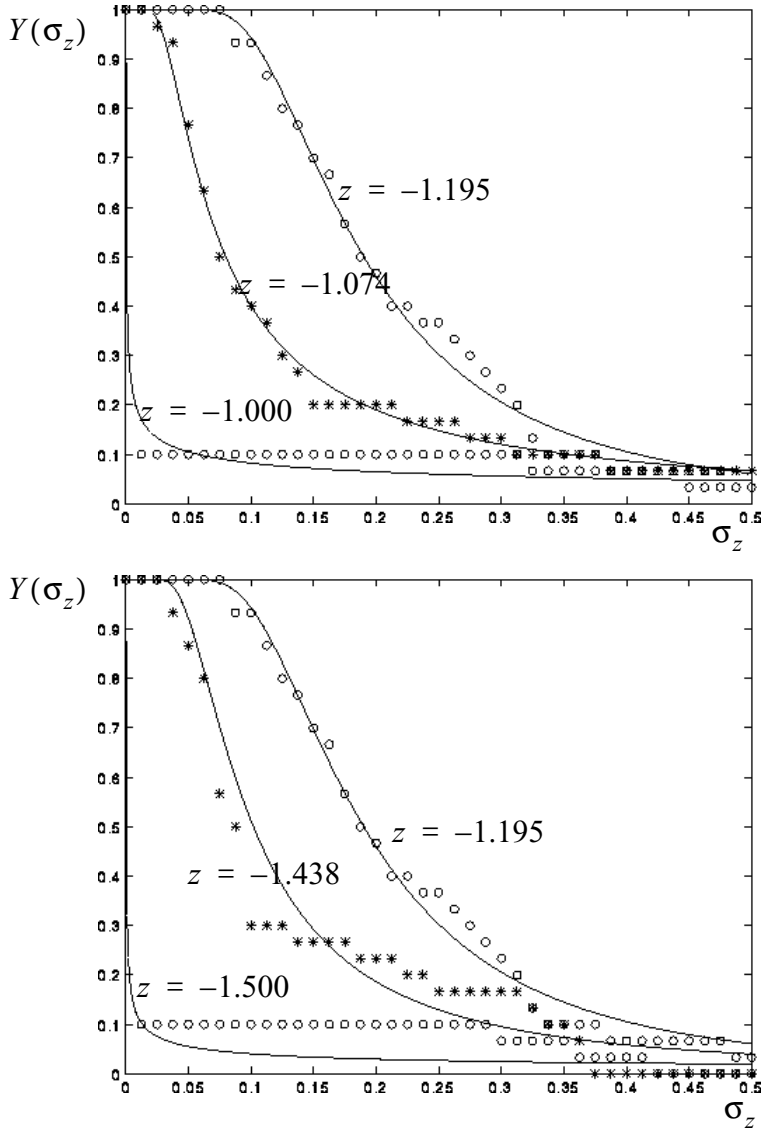


FIGURE 3.28. Parametric yield for different values of the offset term.

$$\beta_{n_1} = \beta_{n_2} = \beta_n \quad \text{and} \quad V_{T_{n_1}} = V_{T_{n_2}} = V_{T_n} \tag{3.27}$$

This, together with the fact that the sum of the currents through them equals the amount of current pulled by the tail source, I_B, yields the following equations for these currents:

$$I_{Mn_1} = \begin{cases} I_B & \text{if} \quad x_{ij} > \sqrt{\dfrac{2I_B}{\beta_n}} \\ \dfrac{I_B}{2} + \dfrac{\beta_n}{4} \cdot x_{ij} \cdot \sqrt{\dfrac{4I_B}{\beta_n} - x_{ij}^{\,2}} & \text{if} \quad |x_{ij}| \le \sqrt{\dfrac{2I_B}{\beta_n}} \\ 0 & \text{if} \quad x_{ij} < -\sqrt{\dfrac{2I_B}{\beta_n}} \end{cases} \tag{3.28}$$

and:

$$I_{Mn_2} = \begin{cases} 0 & \text{if} \quad x_{ij} > \sqrt{\dfrac{2I_B}{\beta_n}} \\ \dfrac{I_B}{2} - \dfrac{\beta_n}{4} \cdot x_{ij} \cdot \sqrt{\dfrac{4I_B}{\beta_n} - x_{ij}^{\,2}} & \text{if} \quad |x_{ij}| \le \sqrt{\dfrac{2I_B}{\beta_n}} \\ I_B & \text{if} \quad x_{ij} < -\sqrt{\dfrac{2I_B}{\beta_n}} \end{cases} \tag{3.29}$$

Now, if we consider that there are slight differences between the electrical parameters of these transistors:

$$\begin{aligned} \beta_{n_1} &= \beta_n + \frac{\Delta\beta_n}{2} \\ \beta_{n_2} &= \beta_n - \frac{\Delta\beta_n}{2} \end{aligned} \quad \text{and} \quad \begin{aligned} V_{T_{n_1}} &= V_{T_n} + \frac{\Delta V_{T_n}}{2} \\ V_{T_{n_2}} &= V_{T_n} - \frac{\Delta V_{T_n}}{2} \end{aligned} \tag{3.30}$$

the central term of the currents flowing through M_{n_1} and M_{n_2} in Eqs. 3.28 and 3.29, are now expressed by:

$$\begin{aligned} I_{M_{n_1}} &= \frac{1}{4\beta_n^2}\left(\beta_n + \frac{\Delta\beta_n}{2}\right)\left[-\frac{\beta_n \Delta\beta_n \Delta V_{T_n}^{\,2}}{2} + \frac{\Delta\beta_n^2 \Delta V_{T_n}^{\,2}}{4} + \right. \\ &+ 2\beta_n I_B + \beta_n \Delta\beta_n \Delta V_{T_n} x_{ij} - \frac{\Delta\beta_n^2 \Delta V_{T_n} x_{ij}}{2} - \frac{\beta_n \Delta\beta_n x_{ij}^{\,2}}{2} + \\ &+ \frac{\Delta\beta_n^2 x_{ij}^{\,2}}{4} + \left(\beta_n - \frac{\Delta\beta_n}{2}\right)(x_{ij} - \Delta V_{T_n}) \cdot \mathrm{sqrt}\left(\frac{\Delta\beta_n^2 \Delta V_{T_n}^{\,2}}{4} - \beta_n^2 \Delta V_{T_n}^{\,2} + \right. \\ &\left. \left. + 4\beta_n I_B + 2\beta_n^2 \Delta V_{T_n} x_{ij} - \frac{\Delta\beta_n^2 \Delta V_{T_n} x_{ij}}{2} - \beta_n^2 x_{ij}^{\,2} + \frac{\Delta\beta_n^2 x_{ij}^{\,2}}{4} \right) \right] \end{aligned} \tag{3.31}$$

and

$$I_{M_{n_2}} = \frac{1}{4\beta_n^2}\left(\beta_n - \frac{\Delta\beta_n}{2}\right)\left[\frac{\beta_n\Delta\beta_n\Delta V_{T_n}{}^2}{2} + \frac{\Delta\beta_n^2\Delta V_{T_n}{}^2}{4} + \right.$$

$$+ 2\beta_n I_B + \beta_n\Delta\beta_n\Delta V_{T_n}x_{ij} - \frac{\Delta\beta_n^2\Delta V_{T_n}x_{ij}}{2} - \frac{\beta_n\Delta\beta_n x_{ij}{}^2}{2} +$$

$$+ \frac{\Delta\beta_n^2 x_{ij}{}^2}{4} + \left(\beta_n + \frac{\Delta\beta_n}{2}\right)(x_{ij} - \Delta V_{T_n}) \cdot \mathrm{sqrt}\left(\frac{\Delta\beta_n^2\Delta V_{T_n}{}^2}{4} - \beta_n^2\Delta V_{T_n}{}^2 + \right. \tag{3.32}$$

$$\left.\left. + 4\beta_n I_B + 2\beta_n^2\Delta V_{T_n}x_{ij} - \frac{\Delta\beta_n^2\Delta V_{T_n}x_{ij}}{2} - \beta_n^2 x_{ij}{}^2 + \frac{\Delta\beta_n^2 x_{ij}{}^2}{4}\right)\right]$$

Up to this point, it can be observed that the introduction of these parameter deviations result in a substantially more complex model. Also notice that only threshold voltage mismatch (ΔV_{T_n}) and not the nominal value (V_{T_n}) influences the differential pair performance. A simplified version of this model, based on that described in Sect. 3. 1. 4, in which only threshold voltage variations has been considered, has been implemented in SIRENA. The effect of this deviation on a 32×32 CNN chip based on this model and realizing binary images edge detection can be evaluated. The following table shows the set of network parameters used in the simulation:

Variable, Parameter	Name	Value	Units
Input and initial state (max)	x, u	0.50	volts
Input and initial state (min)	x, u	-0.50	volts
Diff. pairs bias currents (I_B)	ibx, ibu	5.0×10^{-6}	amps
NMOS transcond. ($\beta_n = \beta_n^* \frac{W}{L}$)	b1x,...,b2u	295×10^{-6}	A/V^2
PMOS transcond. ($\beta_p = \beta_p^* \frac{W}{L}$)	b3a00,...,b6b22	60×10^{-6}	A/V^2
Nominal threshold (V_{T_n})	vtn	0.6	volts
Time constant (τ)	tau	1×10^{-9}	seconds
Normalizing transcond. (G_c)	gc	50×10^{-6}	mhos

TABLE 3.5. OTA based CNN model parameters.

Threshold voltage mismatch (ΔV_{T_n}) has been selected to follow a zero-mean normal distribution where the standard deviation has been swept from 0 to 25% the nominal value of V_{T_n}. Then Montecarlo analysis has been performed, based upon SIRENA simulation of the network, for each value of $\sigma(\Delta V_{T_n})$, the threshold voltage mismatch standard deviation, and the percentage of correctly-functioning samples has been computed, resulting in the yield curve of Fig. 3.29. This data can

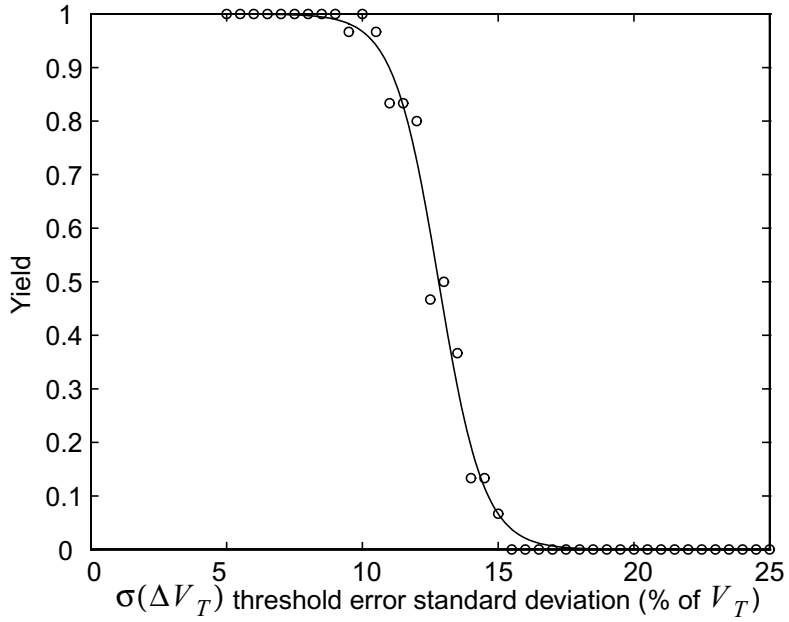


FIGURE 3.29. Percentage of correct samples vs. threshold voltage mismatch deviation.

be used now to resize the transistors in the differential pair, according to the relation between MOS transistors area and mismatching figures established in Chapter 2 (Eq. 2.10), that is:

$$\sigma^2(\Delta V_{T0}) = \frac{A_{V_{T0}}^2}{W \cdot L} + S_{V_{T0}}^2 \cdot D \tag{3.33}$$

now considering that this circuit will exhibit an extremely low yield beyond $\sigma(\Delta V_{T_n}) = 10\%$.

3. 3. Template learning and optimization

Although straightforward methods for template design has been described, there are circumstances in which applying this methodology is impractical. Sometimes, the vast amount of situations to be taken into account is not manageable by detailed analysis of the dynamic routes. Sometimes, the proper functionality of the network is not perfectly defined, having to cope with a reduced set of correct input-output pairs from which is not easy to extract, by inspection, common features, that is to generalize and conform well-defined classes from them. The solution is turning to automated learning. This can be realized within a reasonable time framework with a moderate computing power if a reliable, efficient simulation tools available. So let us examine how template design can be formulated as a supervised learning process —which in the end is just a matter of optimization of some performance index: an objective function that measures the proximity between the

actual response of the system and the desired response— and after that we will extend these concepts by reviewing some experiments on chip-in-the-loop training.

In addition, we will see here how hardware requirements can be relaxed if the implementation of only a reduced number of templates is pretended. The trade-off between robustness and flexibility shows up again. The elements of the templates required for a specific application will be redefined by scaling the synapses strength, resulting in a smaller dynamic range required in the implementation of the weights, thus broadening the room for the designer.

3. 3. 1. Designing templates by learning

Supervised learning on CNNs

Different authors have addressed the incorporation of unsupervised learning and self-organization to the CNNs so as to implement associative memories [Liu93] [Szol93]. However, learning tasks involved in adaptive image processing within the CNN framework are based on a *supervised* learning model. Here, the training process is conducted by a teacher and the network parameters adapt to convey the desired input-output mapping. The update of the parameters vector $\mathbf{p} = \{a_{ul}, \ldots, a_{br}, b_{ul}, \ldots \ldots, b_{br}, z\}$ is realized following an optimization algorithm which intends either to maximize an *objective function*, or, what is equivalent, to find the minimum of an *error surface* $\varepsilon(\mathbf{p})$.

Fig. 3.30 illustrates a learning scheme for a single-layer LTI CNN based on *gradient descent* method. The optimization goal is to find a value for the parameters vector $\mathbf{p}$ for which there is a minimum of the difference between the actual output of the network $\mathbf{y}^l(\infty)$ and the desired output $\mathbf{d}^l(\infty)$ for given initial state $\mathbf{x}^l(0)$ and input $\mathbf{u}^l$, and for all training patterns ($l = 1, \ldots, L$). The error in the input-output map is defined as a distance metric of norm ν, that for one specific cell of the array is:

$$e_{ij}^l(\mathbf{p}) \equiv \left| d_{ij}^l(\infty) - y_{ij}^l(\infty) \right|^\nu \tag{3.34}$$

and the network error is defined as the sum of the error yielded by each cell in the network, over all training patterns:

$$\varepsilon(\mathbf{p}) = \sum_{l=1}^{L} \varepsilon^l(\mathbf{p}) = \sum_{l=1}^{L} \sum_{i=1}^{M} \sum_{j=1}^{N} e_{ij}^l(\mathbf{p}) \tag{3.35}$$

The algorithm starts with a parameter vector $\mathbf{p}(0)$. Then, $\{\mathbf{y}^l(\infty) | l = 1, \ldots, L\}$ and $\{\mathbf{d}^l(\infty) | l = 1, \ldots, L\}$ are generated by the single-layer CNN and the teacher system, respectively. After that, $\varepsilon(\mathbf{p})$ is computed and so is the gradient $d\varepsilon / d\mathbf{p}$ —what will require several evaluations of the error surface in the vicinity of $\mathbf{p}$, and this is not indicated in

Fig. 3.30. Then, the vector of parameters is updated incrementally, according to the value of the gradient,

$$\mathbf{p}(n+1) = \mathbf{p}(n) - \eta \frac{d\varepsilon(\mathbf{p})}{d\mathbf{p}} \tag{3.36}$$

where η is the *learning rate*. After a number of iterations, the algorithm converges to a local minimum of the error surface in which the gradient vanishes, $d\varepsilon / d\mathbf{p} = 0$.

In order to avoid a premature convergence of the algorithm to any of the local minima not corresponding to appropriate outputs —due to the sigmoidal shape of the CNN cell output function the error surface happens to be pretty rugged and uneven— the error surface can be redefined out of the final state of the network instead of using the cells' output, as proposed in [Noss93],

$$\varepsilon(\mathbf{p}) = \sum_{l=1}^{L} \varepsilon^l(\mathbf{p}) = \sum_{l=1}^{L} \sum_{i=1}^{M} \sum_{j=1}^{N} \left| d_{ij}^l(\infty) - x_{ij}^l(\infty) \right|^{\nu} \tag{3.37}$$

The way in which the gradient of this error surface is computed, and this is a complicated nonlinear function for which the Jacobian may not be well-defined, classifies the learning algorithms into different types: backpropagation, recurrent backpropagation [Pine87], backpropagation through time [Pear89]. A more convenient approach seems to be using direct observation of the influence of the parameters variations in the error surface, instead of explicit computation of the gradient of the error surface [Demb90]. For instance, by using a *perturbative approximation* method like that reported in [Cauw96]. Consider π being a vector of random perturbations of the same size as $\mathbf{p}$, and whose components can be $\pm\delta$,

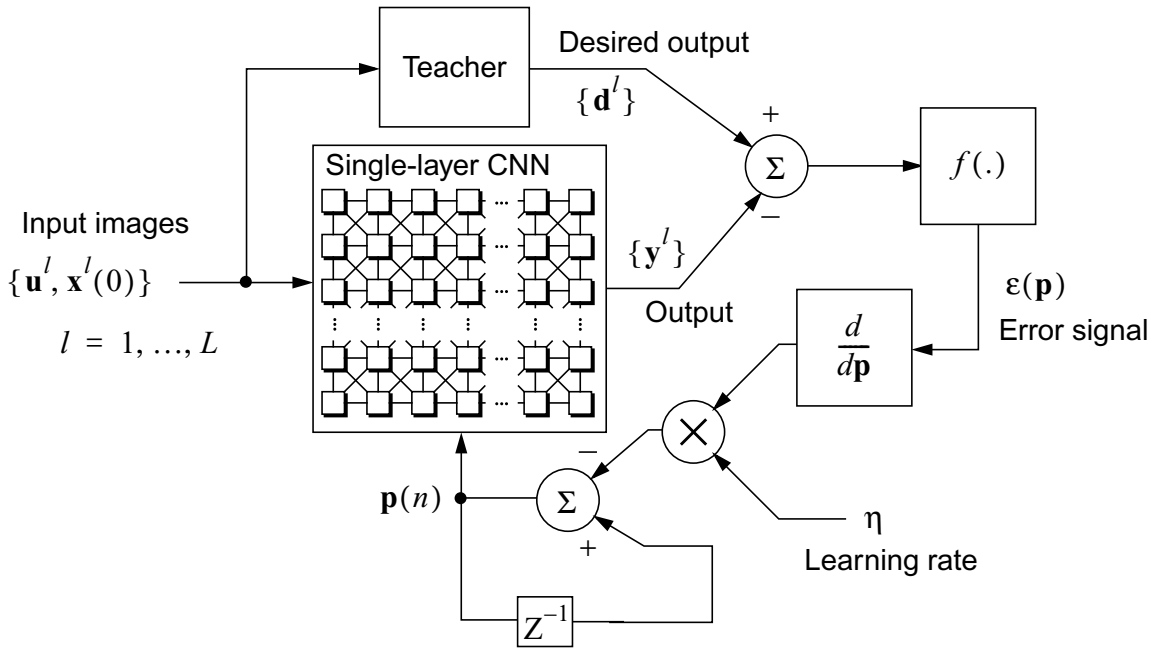


FIGURE 3.30. Gradient descent learning algorithm for a single-layer CNN.

this is, a fixed amplitude and a random sign, with equal probability to either sign. Then the differentially perturbed error is calculated as:

$$\hat{\varepsilon}(n) = \frac{1}{2}[\varepsilon(\mathbf{p}+\pi) - \varepsilon(\mathbf{p}-\pi)] \tag{3.38}$$

then, it is employed to compute the incremental updating of the parameters vector:

$$\mathbf{p}(n+1) = \mathbf{p}(n) - \eta \cdot \hat{\varepsilon}(n) \cdot \pi \tag{3.39}$$

what results in a random-direction error descent.

In addition, unconstrained optimization of the parameters vector in a CNN will lead to non-convergence of the algorithm because the stability of the network is not guaranteed for every $\mathbf{p}$. Therefore, optimization must be constrained to a certain region of the parameters space to ensure convergence, either by implementing those constraints in the error surface or by providing the mechanisms to stop the CNN dynamics and assign a large error value for vectors leading to unstable, oscillating or even chaotic behaviour of the network. This strategy has been implemented in a texture classification algorithm [Szir94] developed for the stored program 20×22-cell CNNUM chip referred before. Basically, it consists in the application of *genetic* algorithms to the design and optimization of CNN templates [Koze93]. Here, a set of different vectors of parameters is encoded into a set of binary strings, known as *chromosomes*. Then, performance of the network for each one of this chromosomes is evaluated deriving a fitness factor for all of them. After that, a new generation of $\mathbf{p}$-vectors is created by mating the chromosomes with higher fitness values, by the application of several operators like crossover and mutation, in an analogy of what happens in nature. At the end of the process, an optimum vector of CNN parameters to perform the desired input-output mapping will be obtained.

Post-fabrication chip training

The template coefficients of a CNN determine the network evolution specifying a definite performance. As discussed in [Noss93], the set of network parameters that encode a specific CNN operation can be found either by design or by learning. On one side, the design of a template set consists in the definition of the local dynamic rules that conveys the required functionality. On the other, the network is trained with a set of input-output pairs in order to realize a desired task, making use of the potential for learning and adaptation of the CNN framework.

At the empirical level, weight optimization in a particular hardware implementation of the CNNUM can be arrange as a chip-in-the-loop learning system aimed for adaptive signal processing. This learning system (Fig. 3.31) consists on one side of a programmable stored-weight CNN chip with optical image acquisition capabilities [Domí97], that is

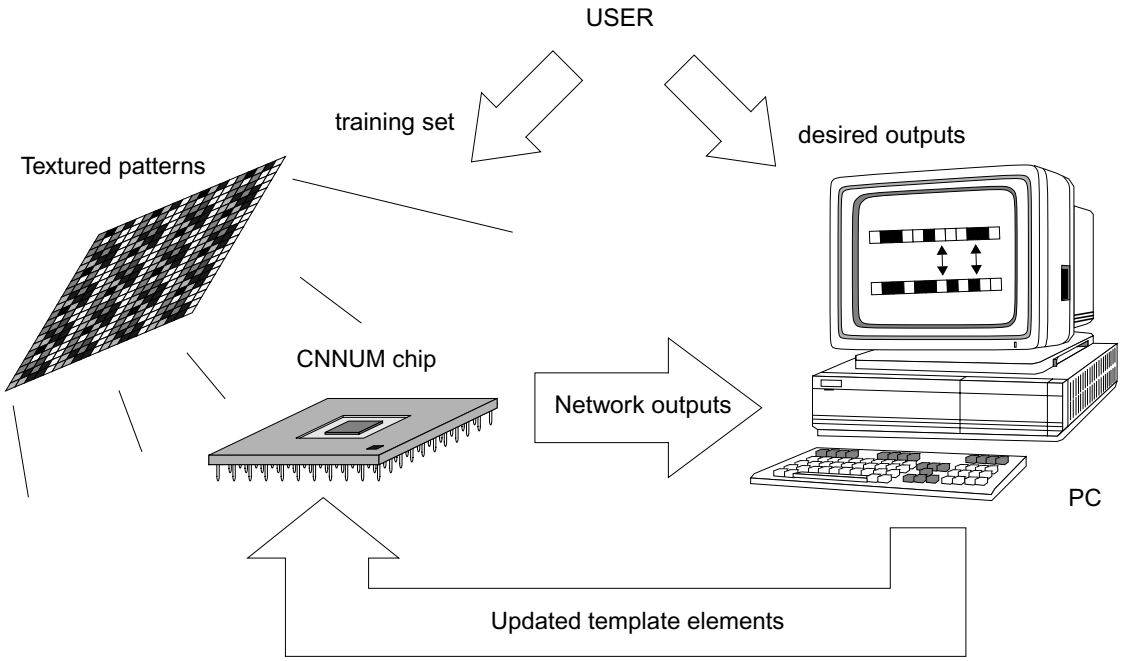


FIGURE 3.31. Chip-in-the-loop learning system for texture classification [Szir94].

going to be trained for a specific operation. On the other side a host computer implements the optimization algorithm, evaluating the error function and generating the updated weight values. After training, the network will operate onto a set of test images evolving to the proper steady-state by means of the adapted connections weights. Genetic algorithms (GA's) are going to be employed in the training phase for weight optimization. GA's are stochastic optimization algorithms, originally motivated by the mechanisms of natural selection and genetics [Gold89]. They have been successfully applied to the optimization and design of CNN templates [Koze93]. GA's do not require the computation of the gradient of the error surface and ensure convergence even in the case of very noisy and discontinuous search spaces.

In order to realize CNN weight optimization, each template element has to be coded into a binary string. These binary strings are combined into one single string representing the parameters vector $\mathbf{p}$. It is very important to realize a correct encoding, this is, bits that can be affected by a large amount of changes during optimization should not encode a large fraction of the template element value, otherwise it will cause non-convergence of the algorithm. This binary string that stands for $\mathbf{p}$ is its chromosome. The optimization begins with a population of N chromosomes, that is, a set of N vectors of parameters. Then, the network outputs for each coded $\mathbf{p}$ in the current population are evaluated, compared with the desired outputs, and a fitness factor, or a cost function, for each chromosome is computed, for instance:

$$\varepsilon(\mathbf{p}) = \sum_{l=1}^{L} \sum_{i=1}^{M} \sum_{j=1}^{N} [d_{ij}(\infty) - y_{ij}(\infty)]^2 \tag{3.40}$$

Now, only the $\mathbf{p}$-vectors with the higher fitness factors —correspondingly the smaller $\varepsilon(\mathbf{p})$, are selected for reproduction and the generation of a new population. Chromosomes of the more fitted vectors are mated with the help of several operators, that emulate the generation of new populations in nature. These operators are mutation and crossover. Once again, the mutation rate and the extent and randomness of the crossover operator are critical factors for the performance of the optimization algorithm. Then the evaluation-selection-reproduction cycle is repeated until a convergence criterion is accomplished.

This chip-in-the-loop learning has been employed to classify different natural Brodatz textures [Brod66] by computing a different histogram for each of them with a single CNN template [Szir94]. Starting in a set of textures with a similar ratio of black pixels (ROB) in their thresholded version, the CNN is trained to generate patterns with fairly different ROBs for each texture with a single template. The estimated Bayesian-error [McKa95] for the recognition of the 4 textures is about $4\text{-}5\%$ for each class, $2\text{-}4\%$ for the classification into 3 different texture classes and $0\text{-}4\%$ for the recognition of two of them. These results are independent of the texture-rotation and sheet-tilt in a wide range.

3. 3. 2. Minimization of the required dynamic range in the weights implementation

Synapse strength and dynamic range of the weights

One of the most important building blocks in a neural network is the synapse. In the particular context of the CNNs, the critical issues in the design of the synaptic block are accuracy and matching versus device dimensions. We will be interested in the implementation of the template elements with a certain precision, and it will be desirable to maintain a high level of similarity between weights implementation in different locations at the array. This can be achieved by using larger devices to realize the synapse circuits, what is just the opposite to the initial drive of integrating as many unitary processors as possible in a single chip (see Chapter 2). Therefore, any reduction on the accuracy requirements (the necessary dynamic range) for weight implementation leads to an improvement in cell packing density, or computing power per area unit.

Let us consider the summing node of one particular cell in a neural network (Fig. 3.32). There will be contributions coming from each connected neighbours. These contributions are weighted by the corresponding elements of a cloning template, in the case of a CNN. The scaling factor G_0 has been introduced here because it will be useful later, but, being the same for all synapses it can be considered part of the activation function $f(\,)$. Each template set (that includes feedback and feedforward templates, A and B, as well as the bias term z) will have a template element, or set of elements, with the maximum absolute value ($w_{\max}$). There is, correspondingly, an element or a set of them, with a minimum absolute

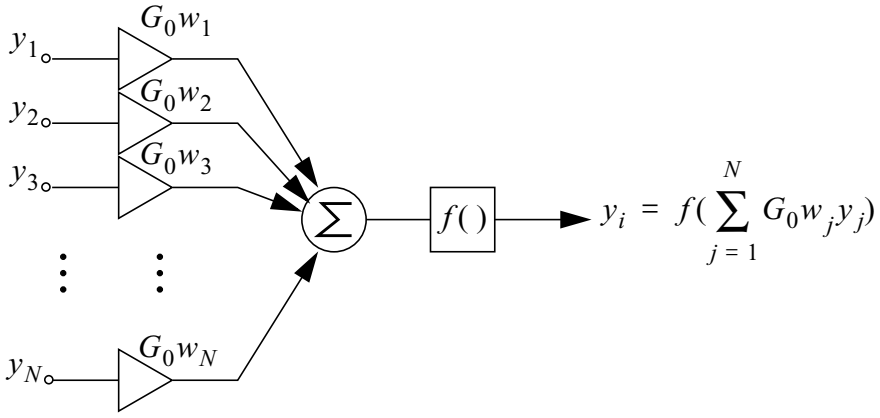


FIGURE 3.32. Conceptual model of a neural network cell.

value in the whole set. There is, as well, a minimum distance between two adjacent weight values, thus, the smallest of these last two quantities will be assigned to $w_{\min}$, which is the absolute minimum weight discrimination value. The ratio between these two extremes:

$$\mathrm{DR} = \frac{w_{\max}}{w_{\min}} \tag{3.41}$$

is the required dynamic range that a physical realization of the synaptic block must ensure for this template set to be implemented. Accuracy levels in the weight implementation that are below this DR, will not properly exhibit the ideally predicted behaviour of the network.

Suppose that we are interested in the implementation of a programmable CNN which has to be able to realize some specific application. This application consists in an *analogic* algorithm that makes use of a number (say N) of template sets. Thus the following set of template sets can be defined for each application: $\mathbf{S} = \{S_i\}_{i=1,\ldots,N}$. This set of template sets can be seen as a library of analog functions that have to be available for some analogic algorithm to be performed. Each template set (S_i), that is, each member of this library, will have its own required dynamic range, DR_i, computed from its elements with maximum and minimum absolute values. It is important to notice that scaling all the elements in a template set does not affect to the dynamic range requirements on the weights, as both $w_{\max_i}$ and $w_{\min_i}$ are scaled equally, no matter if $w_{\min}$ is an actual weight value or the smallest difference between two adjacent weights.

If the network is to be isotropic, *i. e.* no direction of the interconnections is privileged or somehow distinguished from the others, then there is little, if any, room for relaxing the requirements. The dynamic range necessary to implement the application will be the maximum range found between those computed for the individual sets:

$$\mathrm{DR} = \max\{\mathrm{DR}_i\} = \max\left\{\frac{w_{\max_i}}{w_{\min_i}}\right\} \tag{3.42}$$

However, and do not forget that we are centred in a finite number of template sets that we will wish to have available, an inspection of the set **S** finds that there are always some connections displaying stronger weights by far, especially in the most common operations related with image processing. Therefore it can be thought of finding a way in which this imbalance or lack of symmetry can be hard-coded in the connections implementation, thus alleviating the required DR.

Consider, for this purpose, the diagram in Fig. 3.33. Here, each synapse is scaled differently, thus favouring some connections in front of others. Let us call i_k the contribution of the k-th synaptic block, to the summing node:

$$i_k = G_k w_k y_k \tag{3.43}$$

If a different synapse, the l-th, have a scaling factor, or in other words a strength G_l that is n times the strength of the k-th synapse, then, if they have to implement the same contribution, the synaptic weights applied to both must hold the following relation:

$$w_k = n \cdot w_l \tag{3.44}$$

Therefore, if the synapses that are usually driven by larger weights are strengthened, i. e. are designed to have larger scaling factors, then the required weights will be smaller, approximating their values to the values of the weights of the weaker synapses, thus lowering the required DR.

Automatic optimization by simulated annealing

In order to minimize the requirements on the weights DR, the synapses which are usually driven by larger weights will be reinforced by increasing their scaling factors. Once this action is decided, the required dynamic range for the whole set of templates has to be calculated again, and it will not necessarily result in an improvement of the DR requirements, because our last change may eventually produce a larger maximum DR_i for a different template. Hence, we are interested in finding a vector of synapse

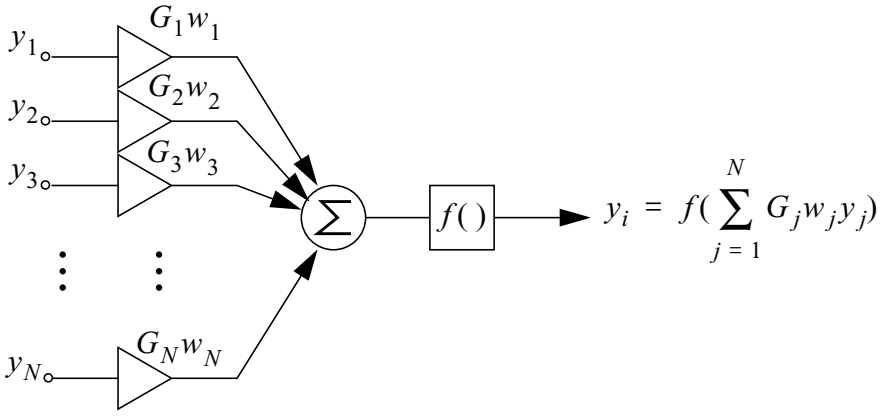


FIGURE 3.33. Conceptual model of a neural network cell with anisotropic connections.

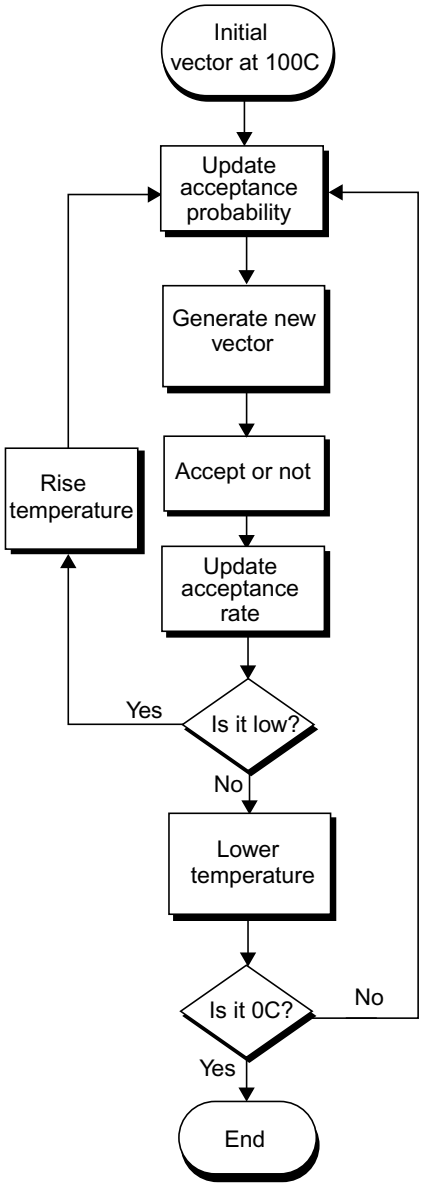


FIGURE 3.34. Simulated annealing flowchart.

scaling factors $\mathbf{G} = \{G_1, G_2, \ldots, G_N\}$ for which the required DR is minimum:

$$\mathrm{DR}(\mathbf{G}) = \min\{\max[\mathrm{DR}_i(\mathbf{G})]\} \tag{3.45}$$

This is a large scale optimization problem in which a global minimum has to be found in a region of the parameters space in which many local *extrema* will be present. The approach presented here is based in simulated annealing (SA) [Kirk83]. This method is based on an analogy with thermodynamics. At high temperatures, condensed matter is melted and its molecules move freely having weaker bonds between them. As the liquid is being cooled down slowly, these molecules loose thermal motion and they line up with themselves to form completely ordered crystals. This crystal is a minimum energy state naturally found by slowly cooling the melted material. If it is suddenly cooled a polycristalline of amorphous state, with a higher energy, is reached. Most of the guided minimization algorithms go downhill to the nearest minimum as quickly as they can, thus emulating what happens with rapidly cooled matter [Pres88]. Now, we are going to permit the system to move uphill sometimes, in order to escape from local minima, in a controlled manner so convergence to the global minimum is ensured. In Nature, the particles of a system in equilibrium possess an energy (E)that follows the Boltzmann probability distribution:

$$P(E) \propto \exp\left(-\frac{E}{kT}\right) \tag{3.46}$$

what means that even at very low temperatures —T is the absolute temperature and k is the Boltzmann constant— there is a very small but not null probability of finding a particle with in a quite high energy level. Incorporating this concept to numerical computations, first done in [Metr53], our optimization process will allow movements in the parameters space that go uphill in the minimization, according to the simulated temperature of the system in that moment. Thus as the process goes on, the temperature will be slowly lowered until reaching the frozen state. At this time the system should have escaped from all the poor local minima found in its way, and fallen into the global minimum.

Fig. 3.34 depicts the flowchart of the SA process, that has been implemented in MATLAB. This flowchart

represents a modification of the *traditional* simulated annealing methods that has been successfully applied in [Mede98]. Originally, optimization based in SA was programmed to have a slow-paced descent of temperature from the highest point, let us *normalize* it to $100C$, to frost, say $0C$. Some trials have to be made to adjust the scaling factors of the acceptance probability —the probability for an uphill movement to be accepted, which is a function of temperature of the shape of the Boltzmann formula. If not, the process will achieve extremely high or low acceptance rates, which is the ratio between the number of accepted uphill movements and the total number of them. In the first case, when too many of those movements are accepted, the process will spend too much time blindly wandering without finding a reasonable minimization path. On the other side, if the acceptance probability is small, the system might prematurely fall into the basin of a good enough minimum, but not the best. In order to avoid these problems, a desirable acceptance rate is programmed to control the minimization process.

Therefore, the minimization of the required DR following the flowchart in Fig. 3.34 starts at $100C$. If no better guess is available, the starting point is an isotropic network, with equal strength for all the synapses, let us say 1, and the weights nominally assigned for the templates belonging to the set under study ($\mathbf{S}$). An initial guess for the acceptance probability scaling factor has to be given. Then, a new vector of parameters is generated by perturbing the original vector. This new vector will be accepted or not. Then traditional methods skip the revision of the acceptance rate, but in our case, the actual rate is compared with the desired ratio. If it is below the rate thought to be convenient, the system is warmed until the prescribed acceptance rate is reached. If it is over the desired acceptance rate, the system is cooled to avoid wandering. After the whole process, the system will have frozen for a vector of parameters that renders the minimum DR. A gradient descent search is perform in addition to accurately find the minimum once we are sure that the system is in the basin of the global *extremum*.

Optimization example for a 2-layer CNN

There are some aspects that have to be considered before going on with the optimization procedure. On one side, the resulting anisotropic network will be improved to realize the specific set of templates employed in the DR minimization, but some other template sets, outside the one considered, will be difficult, if not impossible, to implement in this network. Keep in mind that we have traded part of the potential programmability of the CNN for robustness. Another thing is that, strengthening some synaptic blocks implies an unavoidable area penalty. Special care must be taken not to absurdly oversize connection blocks. All in all, at the beginning, we were interested in enhancing the robustness of the implementation while keeping a low area waste. And finally, symmetries found between the elements of each set of templates should be exploited to facilitate the computations and to avoid ending in unrealistic asymmetrical distributions of

synapse strengths.

Let us apply this methodology to the minimization of the DR requirements by redefining the strengths of the synapses in a 2-layer CNN which will be described in detail in Chapter 5. This bio-inspired CNN model is based on studies of the vertebrate retina [Werb94], [Jaco94], [Reke00a], [Rosk01]. There is a group of applications for which this chip is developed. It is capable of manifesting some wave and pattern formation phenomena, as well as realizing some low- and medium-level image processing operations. Thus the deal is to ensure robust implementation of the templates related with these applications. Table 3.6 shows a list of the elements of the templates reported in [Reke00b] to this respect. In this table, the elements with maximum and minimum absolute value has been shadowed. However, before optimization, we should take advantage of the symmetries in the template elements for these applications. First of all, both layers, should have equivalent operation, thus, instead of considering 14 applications with 24 weights each, we will arrange it into 28 set of templates with 12 elements each. Because of the symmetries inside each template, the A_{ii} feedback template, that initially counts in 9 elements, is reduced to 3, one representing the four corners, another representing the for elements in the middle of the sides, and a third one that stand for the central element of the template. Finally, adding the weight of the contribution to the other layer a_{ij}, the feedforward template —only the central element— b_i, and the bias weight, z_i, each application is represented by six elements (Table 3.7). If we assign, a priori, equal strength to all the synapses (unity) the required dynamic range is $7/0.05 = 140$. This is $7.13\,\text{bits}$, what is not easy to achieve in analog VLSI.

Now, we define a multidimensional cost function, ϕ, which is the required dynamic range. Its argument is a vector formed by the inverses of the strengths of the synapses. Only the six synapses mentioned above: corner, mid-side and central elements of the feedback template, interlayer coefficient, central element of the feedforward template and bias term coefficient, are going to be considered. Then we begin by evaluating the cost function in the starting point, say vector $\mathbf{m}_0 = [1, 1, 1, 1, 1, 1]$. After that, the simulated annealing process starts. Once the system reaches zero degrees, no thermal vibration will allow the system to escape from the minimum found. We have added a guided optimization process —driven by the simplex algorithm— to find the actual minimum, $\phi = 15.8114$, which is equivalent to $3.98\,\text{bits}$. That occurs at: $\mathbf{m}_1 = [1.0000, 0.8974, 0.0812, 0.1581, 0.1850, 0.5792]$. The corresponding strengths for the synapses is given by $\mathbf{G} = [1.0000, 1.1144, 12.3114, , 6.3246, 5.4063, 1.7265]$. It is convenient to round up or down this numbers, because it will be a lot easier to match devices which are integer multiples. Then, for $\mathbf{G} = [1, 1, 12, 6, 5, 2]$, a dynamic range of 16.67, this is $4.01\,\text{bits}$, is found. Although we have released $3\,\text{bits}$ in the specs for the weights implementation requirements, the increment in area represented by these synaptic strengths might be too much. Anyway this is a good starting point to find a compromise solution. For instance, allowing another bit, what means a dynamic range of 33.3, $5.01\,\text{bits}$, a less costly alternative vector is found: $\mathbf{G} = [1, 1, 3, 3, 3, 1]$.

Applications		A_{11}									A_{22}									a_{12}	a_{21}	b_1	b_2	z_1	z_2
Waves	triggered	ocd	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	1.00	0.00	0.00	0.00	**3.75**	**3.75**
	travelling	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	3.00	**-5.0**	0.00	0.00	-1.25	2.25
	autowaves	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	**5.5**	-5.0	0.00	0.00	-1.25	-1.25
	spiral	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	5.5	**-17.5**	0.00	0.00	-1.25	-1.25
Pattern Formation	1	-0.5	0.00	-0.5	0.00	**2.00**	0.00	-0.5	0.00	-0.5	-0.5	-1.0	-0.5	-1.0	**2.00**	-1.0	-0.5	-1.0	-0.5	0.00	0.00	0.00	0.00	0.00	0.00
	2	0.5	0.00	0.5	0.00	**2.00**	0.00	0.5	0.00	0.5	-0.5	1.00	-0.5	1.00	**2.00**	1.00	-0.5	1.00	-0.5	0.00	0.00	0.00	0.00	0.00	0.00
	3	0.5	1.00	0.5	1.00	2.00	1.00	0.5	1.00	0.5	-0.5	-1.0	-0.5	-1.0	2.00	-1.0	-0.5	-1.0	-0.5	**4.00**	0.00	0.00	0.00	0.00	0.00
	4	0.5	1.00	0.5	1.00	**2.00**	1.00	0.5	1.00	0.5	-0.5	0.00	-0.5	0.00	**2.00**	0.00	-0.5	0.00	-0.5	1.00	0.00	0.00	0.00	0.00	0.00
Image Processing	edge enh.	0.05	0.20	0.05	0.20	0.00	0.20	0.05	0.20	0.05	0.05	0.20	0.05	0.20	0.00	0.20	0.05	0.20	0.05	0.1	**-5.0**	1.00	0.00	0.00	0.00
	halftoning	-0.36	-0.60	-0.36	-0.60	1.05	-0.60	-0.36	-0.60	-0.36	0.05	0.20	0.05	0.20	0.00	0.20	0.05	0.20	0.05	1.0	-1.0	**7.00**	0.00	0.00	0.00
	active con.	0.05	0.20	0.05	0.20	0.00	0.20	0.05	0.20	0.05	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	**-2.5**	-0.1	0.00	0.00	0.00	0.00
	wave met.	0.25	0.25	0.25	0.25	**3.00**	0.25	0.25	0.25	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.5	0.00	2.00	0.00	1.75	-1.0
	edge det.	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	-0.1	-0.1	-0.1	-0.1	2.3	-0.1	-0.1	-0.1	-0.1	-1.0	0.00	0.00	0.00	**3.75**	0.00
	skeleton	0.25	0.25	0.25	0.25	3.00	0.25	0.25	0.25	0.25	-0.25	-0.25	-0.25	-0.25	**5.40**	-0.25	-0.25	-0.25	-0.25	-2.5	0.0	0.00	0.00	3.75	0.00

TABLE 3.6. Template elements for different applications in the 2-layer CNN

Applications		A_{11}			a_{12}	b_1	z_1
Waves	triggered	0.25	0.25	3.00	1.00	0.00	**3.75**
		0.25	0.25	3.00	0.00	0.00	**3.75**
	travelling	0.25	0.25	**3.00**	**3.00**	0.00	-1.25
		0.25	0.25	3.00	**-5.0**	0.00	2.25
	autowaves	0.25	0.25	3.00	**5.5**	0.00	-1.25
		0.25	0.25	3.00	**-5.0**	0.00	-1.25
	spiral	0.25	0.25	3.00	**5.5**	0.00	-1.25
		0.25	0.25	3.00	**-17.5**	0.00	-1.25
Pattern Formation	1	-0.5	0.00	**2.00**	0.00	0.00	0.00
		-0.5	-1.0	**2.00**	0.00	0.00	0.00
	2	0.5	0.00	**2.00**	0.00	0.00	0.00
		-0.5	1.00	**2.00**	0.00	0.00	0.00
	3	0.5	1.00	2.00	**4.00**	0.00	0.00
		-0.5	-1.0	**2.00**	0.00	0.00	0.00
	4	0.5	1.00	**2.00**	1.00	0.00	0.00
		-0.5	0.00	**2.00**	0.00	0.00	0.00
Image Processing	edge enh.	0.05	0.20	0.00	0.1	**1.00**	0.00
		0.05	0.20	0.00	**-5.0**	0.00	0.00
	halftoning	-0.36	-0.60	1.05	1.0	**7.00**	0.00
		0.05	0.20	0.00	**-1.0**	0.00	0.00
	active con.	0.05	0.20	0.00	**-2.5**	0.00	0.00
		0.25	0.25	**3.00**	-0.1	0.00	0.00
	wave met.	0.25	0.25	**3.00**	-0.5	2.00	1.75
		0.00	0.00	0.00	0.00	0.00	-1.0
	edge det.	0.25	0.25	3.00	-1.0	0.00	**3.75**
		-0.1	-0.1	**2.3**	0.00	0.00	0.00
	skeleton	0.25	0.25	3.00	-2.5	0.00	**3.75**
		-0.25	-0.25	**5.40**	0.0	0.00	0.00

TABLE 3.7. Template elements and time constants for different applications in the 2-layer CNN

4 Analog signals storage for the CNN chipset in CMOS technology

Complex analogic image processing algorithms raise the need for analog signal storage in the CNN chipset. On one side, technical limitations on the parallel array processor dimensions drive the demand for some compatible storage means, that permits the obliged picture partitioning without compromising the timing requirements of the application. On the other side, adaptation of video signals to a format that can be processed by the CNN chip also requires some workspace in which the picture samples can be reorganized and accommodated to the CNN processor format and ranges.

Thus, in the first place, we are going to make a survey of the problems associated with sampling and storing analog signals in CMOS technology. For this task we will analyse a simple circuit composed of a switch and a capacitor. After that, we will summarize the characteristics of a analog memory implementation to qualify as the analog cache memory of the CNN chipset. The circuit has to be simple, in order to have as many memory cells as possible in a single chip. It has to be accurate enough, as well, to maintain the 7-8bits aimed in CNN-based image processing. And it must keep a low power consumption, to allow system integration in standalone and portable applications. With the intention of finding an efficient implementation for the analog RAM, several alternatives will be inspected, however, with a strong bias towards a standard CMOS technology solution. This is for the obvious reasons of compatibility for further system integration and maturity and lower fabrication costs of the technology compared to special processes.

Finally we present a prototype for the analog RAM, designed, fabricated and verified in a $0.5\mu m$. Experimental results are shown, either from an electrical perspective, with a complete characterization of the chip features, and from the system design point of view, where some tests with real images are performed. Interesting conclusions are extracted from a comparison with state-of-the-art circuits in this field.

4. 1. Errors in the analog signal storage process

4. 1. 1. Sampling errors

Generally, analog data storage conveys two different modes of operation of the analog memory circuit. In the first one, the *sampling* or *tracking* mode, the signal is sampled at a specific point in time. In the second, called *storage* or *hold* mode, the sampled signal is held for a certain period, until the memory is refreshed or updated with a new sample. Because of the use of real physical devices, certain deviations from the ideal sample and hold process are observed. Let us examine these non-idealities in a simple S/H circuit composed of a pass transistor and a storage capacitor [Nish93a]. We will begin by reviewing errors introduced in the sampling mode and later on we will analyse signal degradation because of leakages in hold mode.

Consider the circuit in Fig. 4.1(a). It tracks the input signal while the pass transistor is ON, and holds the sampled voltage while it is OFF. In the track mode, the sampling process is affected by either deterministic and random errors, resulting in a stored voltage that slightly differs from the input: $V_c = V_i + V_\varepsilon$. The main errors introduced in the sampling process are the sample acquisition error, a phase shift caused by the finite track mode bandwidth and the clock feedthrough (Fig. 4.2). Also noise during the sampling period contributes to the error in the sampled voltage.

Sample acquisition delay

Because of the finite resistance of the pass transistor when it is ON the circuit behaves as a low-pass (LP) first order (one pole) system (Fig. 4.1(b)). The transfer function, $V_c(s)/V_i(s)$, being:

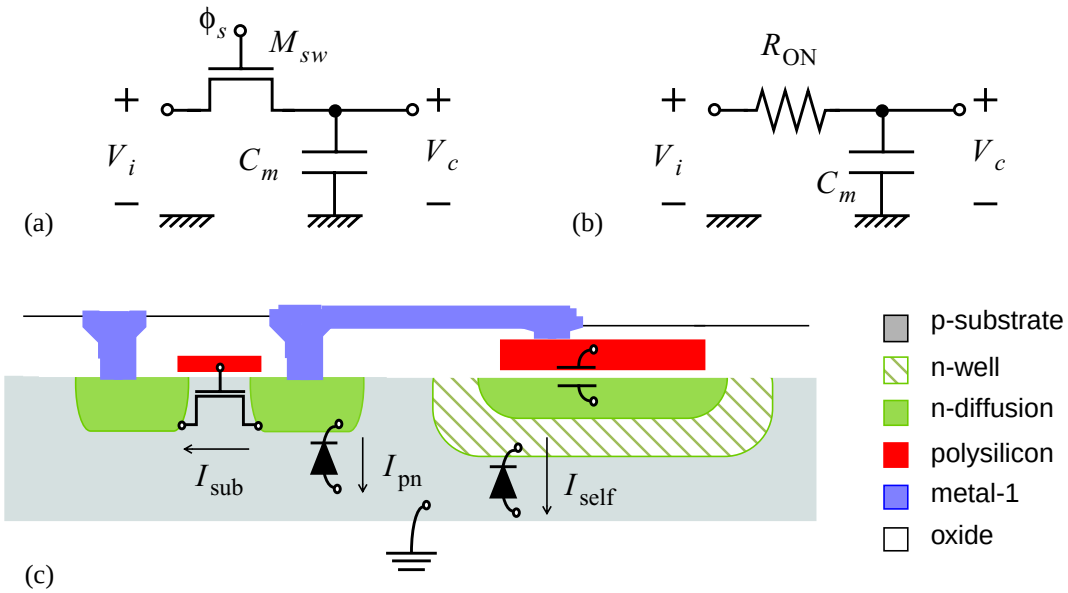


FIGURE 4.1. (a) Storage capacitor and pass transistor, (b) equivalent circuit in track mode and (c) profile of the poly-over-diffusion implementation of the capacitor showing leakage currents.

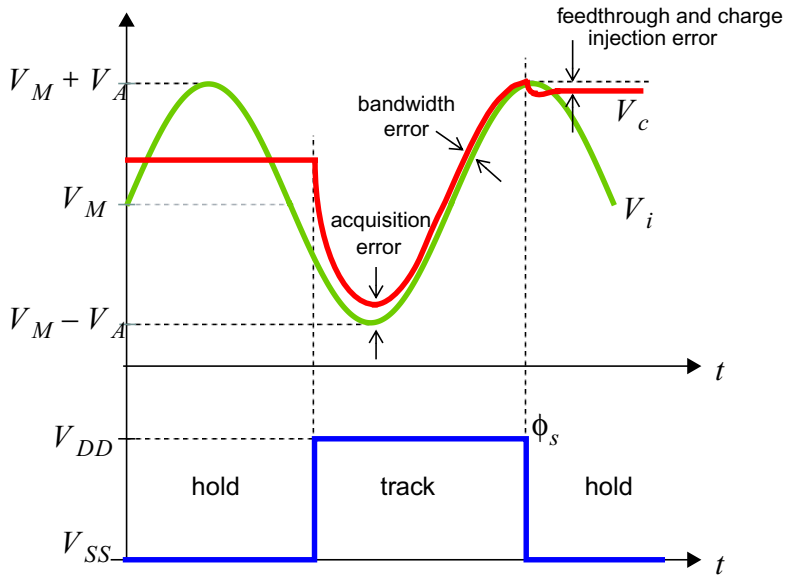


FIGURE 4.2. Errors in the sampling process.

$$H(s) = \frac{V_c(s)}{V_i(s)} = \frac{1}{1 + sR_{\mathrm{ON}}C_m} \tag{4.1}$$

where R_{ON} is the ON-resistance of the switch and C_m is the storage capacitance. Let us call τ to the inverse of the $-3\,\mathrm{dB}$ frequency of the LP filter:

$$\tau = R_{\mathrm{ON}}C_m = 1/\omega_{-3\,\mathrm{dB}} \tag{4.2}$$

The frequency response of the system can help us derive the time response to a step input of size E. In the Laplace domain, the trasform of the output is the product of the transfer function times the input's transform, which for a step of this magnitude is E/s, thus:

$$V_c(s) = \frac{E}{s(1 + s\tau)} \tag{4.3}$$

Computing the inverse Laplace-transform to obtain the time response of the system yields:

$$v_c(t) = E(1 - e^{-t/\tau})u_o(t) \tag{4.4}$$

where $u_o(t)$, the unitary step function starting at $t = 0$, is indicating that the expression is only valid for positive values of t. Suppose that in our particular case, the input signal is confined between $V_M - V_A$ and $V_M + V_A$. It means that the maximum step that we will observe will be $2V_A$, either positive of negative. Consider a step input of size $2V_A$, starting at $v_i(0) = V_M - V_A$, then the response of the circuit will begin at $v_c(0) = V_M - V_A$ and will finally end at $v_c(\infty) = V_M + V_A$, its exact shape given by:

$$v_c(t) = V_M + V_A(1 - 2e^{-t/\tau}) \tag{4.5}$$

As the input signal is $v_i(t) = V_M + V_A$ for any $t > 0^-$, the acquisition error, the difference between the sampled value at a specific point in time and the actual input, can be defined:

$$v_\varepsilon(t) = -2V_A e^{-t/\tau} \tag{4.6}$$

This quantity decreases exponentially in time, and we can make use of it to establish the minimum acquisition time, or, what is equivalent, the maximum sampling rate. If the total input range is $2V_A$ then $V_A/2^N$ represents $1/2$ LSB if the equivalent digital resolution is N. Therefore, the minimum acquisition time is the time required to track a step of the maximum size, with an error smaller than $1/2$ LSB. Hence, $|v_\varepsilon(t_{acq})| \leq V_A/2^N$ implies that, in order to maintain the desired accuracy:

$$t_{acq} \geq \tau(N+1)\ln 2 \tag{4.7}$$

In order to reduce the minimum acquisition time, the sample capacitor can be made smaller, what reduces C_m, and the access switch wider, what diminishes the value of R_{ON}. The undesirable effects of these measures are a heavier clock load and a larger feedthrough error. Therefore, access time and the switching error, which we will consider later on, will have to be traded-off to determine the optimum size for the pass transistor and the storage capacitor.

Finite track-mode bandwidth

Once the acquisition transient settles, the S/H circuit is in track mode. Now, the voltage between the capacitor plates attempts to follow the input voltage. The behaviour of the circuit formed by the pass transistor, that modelled here as a resistance R_{ON}, and the storage capacitor C_m, is described by the transfer function of Eq. 4.1, which in sinusoidal steady-state can be described in terms of the Fourier-transform of the impulse response of the circuit [Deso69]:

$$H(j\omega) = \frac{1}{1 + j\omega\tau} \tag{4.8}$$

Although it does not have an important incidence in the amplitude of the tracking output, as long as the frequency components of the input signal are well below of the cut frequency —the 3dB frequency of the first order LP filter, $1/\tau$, the phase shift introduced can be specially harmful when it is operating onto signals modulated in phase. For a given frequency, $f = \omega/2\pi$, this phase shift is given by:

$$\angle H(j\omega) = -\text{atan}(\omega\tau) \tag{4.9}$$

Again, reducing the capacitor size or the pass transistor resistance, by building a wider channel region, this error is cut down, because both actions cause τ to shrink. Hence, any reduction in the acquisition error will correspondingly diminish the influence of the finite track mode bandwidth. After all, both

effects have the same physical source.

It is important to mention, however, that introducing wider switching devices to attenuate the effect of the finite tracking bandwidth, can increase the clock loads considerably. This causes an excessive clock skew which results in a serious aperture jitter: a random variation of the delay between the edge of the gate signal and the actual time instant in which the circuit enters in hold mode. The switching noise thereby introduced can degrade the system SNR.

Charge injection and clock feedthrough errors

Switching off the access to the storage capacitor by means of a real switch has detrimental effects in the voltage already sampled (Fig. 4.2). A small discrepancy is observed between the ideally sampled voltage and the actually held value. This is caused by two different phenomena: channel charge injection and capacitive coupling of the clock. The fist one consists in the injection of the charge that constitutes the channel of the pass transistor, when it is turned OFF. Consequently, the voltage between the capacitor plates changes in order to accommodate this excess charge. The exact amount of charge injected into C_m (Fig. 4.1(a)) can not be easily computed given that it is not a deterministic phenomenon. However, it depends on the speed of the clock edges and the relative impedances seen at the access switch terminals. The channel charge when M_{sw} is ON is:

$$Q_{ch} = -C_{ox}^{*} WL[V_{DD} - V_i - V_T(V_i - V_{SS})] \tag{4.10}$$

where

$$V_T(V_{SB}) = V_{T0} + \gamma(\sqrt{\phi_B + V_{SB}} - \sqrt{\phi_B}) \tag{4.11}$$

stands for the body-effect. Here, V_{T0} is the extrapolated threshold voltage for $V_{SB} = 0$, γ is the body-effect constant, and ϕ_B is the surface potential in strong inversion while V_{SB} is the source-bulk voltage drop. The degradation in the stored voltage can be written as

$$\Delta V_{c_{ch}} = -\rho \frac{C_{ox}^{*} WL}{C_m}[V_{DD} - V_i - V_T(V_i - V_{SS})] \tag{4.12}$$

where ρ stands for the undeterministic fraction of the channel charge that is actually injected into the storage capacitor ($0 < \rho < 1$). For design purposes, a worst case can be considered in which $\rho = 1$, this is, all of the charge in the channel is injected into the storage capacitor.

For the second phenomenon, let us consider the capacitive divider formed by the gate-to-source overlap parasitic, C_{gds}, and the sampling capacitor C_m. When the input signal, V_i, is no longer driving the storage node, because the switch have been turned off, the clock may cause some extra voltage degradation by being fed through capacitive coupling to the

sampling capacitor. Once the signal ϕ_s reaches

$$\phi_s = V_i + V_T(V_i - V_{SS}) \tag{4.13}$$

the switch turns off and the voltage at C_m does not follow V_i any more. But the clock ϕ_s still goes further down to the negative power supply rail, feeding the storage node through the parasitic overlap capacitance between the gate and the source terminals. This results in an error in the stored voltage that can be expressed by

$$\Delta V_{c_{feed}} = -\frac{C_{gds}}{C_m + C_{gds}}[V_i + V_T(V_i - V_{SS}) - V_{SS}] \tag{4.14}$$

By evaluating this expressions (Eqs. 4.12 and 4.14) for a standard $0.5\mu m$ CMOS technology, it can be appreciated that charge injection causes more signal degradation than clock feedthrough. For this technology, a C^*_{ox} of $3.02 fF/(\mu m)^2$ is found, what makes $C^*_{ox}WL = 1.21 fF$. At the same time, $CGSO$, a SPICE parameter that stands for gate-to-source overlap capacitance, is found to be $1.42 \times 10^{-10} F/m$, what means that $C_{gds} = CGSO \cdot W$ yields $0.18 fF$. As both quantities are multiplied by similar numbers, it can be stated that charge injection error is one order of magnitude larger than clock feedthrough in this technology. For different technologies is about the same.

The worst part is that both contributions to the switching error are signal dependent, and therefore, they will appear as a considerable amount of harmonic distortion. In addition, charge injection and clock feedthrough are reduced by using larger sampling capacitor and a narrower pass transistor. This is exactly the opposite of what can be made for improving the acquisition time. Therefore, the limitations introduced by the switching errors compromise the attainable maximum sampling rate, unless some techniques for cancellation or reduction of the switching error are implemented. We will discuss some of them later.

Noise limitations

Apart from these effects, there is a random contribution to the error in the stored voltage that must be taken into account. It is the effect of thermal noise during the sampling period. During this time, the pass transistor can be considered as a resistance that introduces a thermal noise. Its noise power density, since it is a white noise with gaussian amplitude and distribution, is [Gray93]

$$\overline{v_i^2} = 4kTR_{ON}(\Delta\omega/2\pi) \tag{4.15}$$

On the other side, the single-pole LP filter formed by the access switch and the sampling capacitor limits the bandwidth of the noise to an equivalent noise bandwidth ω_N that can be computed from $H(j\omega)$, the voltage gain as a function of the frequency, as depicted in Eq. 4.8:

$$\omega_N = \frac{1}{H_o^2}\int_0^\infty |H(j\omega)|^2 d\omega = \frac{\pi}{2\tau} \tag{4.16}$$

where $\tau = R_{\mathrm{ON}} C_m$. This results in a total noise power at the output, the storage node in this case, given by:

$$\overline{v_c^2} = \frac{1}{2\pi}\int_0^\infty |H(j\omega)|^2 \frac{\overline{v_i^2}}{\Delta\omega} d\omega = \frac{kT}{C_m} \tag{4.17}$$

which is independent of the transistor size. This noise power at the output can be reduced by using, once more, a larger sampling capacitor. Notice, also, that $\sqrt{kT/C_m}$ constitutes a limit to the dynamic range of the system. For a S/H circuit that operates over the whole rail-to-rail scale ($3.3\,\mathrm{V}$ power supply voltage) with a $0.1\,\mathrm{pF}$ sampling capacitor, the maximum achievable dynamic range will be approximately $84\,\mathrm{dB}$. This is not a crucial issue in our case because a relatively low system resolution is required, but it can restrain the use of analog signal processing in other applications.

4. 1. 2. Leakages in hold mode

Pass transistor leakages

Finally, during the hold period, several leakage currents attempt to discharge the storage capacitor, contributing to degrade the sampled voltage value (Fig. 4.1(c)). On one side, there is a reverse-biased junction formed by the n^+ diffusion area of the source of the pass transistor and the p-type substrate. It pumps a nearly constant current that can be approximated by the reverse-biased saturation current of the diode [Sing94], out of the upper plate of the capacitor:

$$I_{\mathrm{pn}} \approx I_0 = qA\left(\frac{D_p p_{n0}}{L_p} + \frac{D_n n_{p0}}{L_n}\right) \tag{4.18}$$

where, q is the charge of an electron ($1.602\times10^{-19}\,\mathrm{C}$ approx.), A is the area of the pn-junction, D_p and D_n are the diffusion coefficients for holes and electrons, L_p and L_n their diffusion lenghts and p_{n0} and n_{p0} are the minority-carrier concentrations in each side of the junction.

Another leakage source is the subthreshold drain-to-source current of the pass MOS transistor [Schro87]:

$$I_{\mathrm{sub}} = I_{DS} = I_0 \exp\left(-\frac{qV_{GS}}{kT}\right)\left[1 - \exp\left(-\frac{qV_{DS}}{kT}\right)\right] \tag{4.19}$$

whose effects add up with I_{pn} resulting in a total leakage current in the range of the pA.

Self-discharge rate of the capacitor

If the capacitor is implemented by a poly-over-diffusion structure lying on top of a weakly-doped n-well (Fig. 4.3), which is the case of several analog CMOS processes developed by minor modifications of the standard digital CMOS process steps, another diode contributes to the stored voltage degradation. The n-well/p-substrate junction is reverse-biased and there is a current that flows out of the bottom plate of the capacitor. It is described by an expression of the same form of equation Eq. 4.18:

$$I_{\text{self}} \approx qA_{\text{nwell}}\left(\frac{D_p p_{n0}}{L_p} + \frac{D_n n_{p0}}{L_n}\right) \tag{4.20}$$

where now the diode area is the area of the surface between the n-well and the p-substrate, A_{nwell}, and the rest of the parameters, D_p, D_n, L_p, L_n, p_{n0} and n_{p0} are those corresponding to n-well and p-bulk. Since I_{self} is in the fA range at room temperature, it limits the effect of the upper plate leakage. Now voltage degradation during the hold period is given by

$$\frac{dv_c(t)}{dt} = -\frac{1}{C_m}\left(\frac{dq}{dt}\right) \approx -\frac{I_{\text{self}}}{C_a A_{\text{eff}}} \tag{4.21}$$

where C_a is the capacitance per unit area of the poly-over-diffusion structure, and A_{eff} is the area of an ideal capacitor with plane parallel plates, without border effects, with the same capacitance as the one under study. This area is somewhere between A_{poly}, the area of the polysilicon plate of

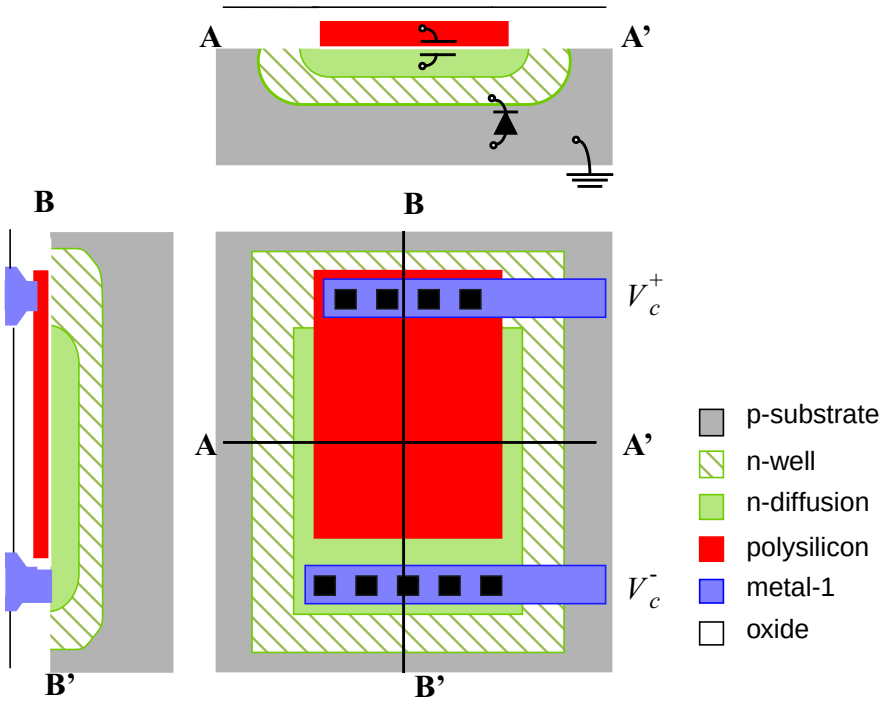


FIGURE 4.3. Polysilicon over n-diffusion capacitor.

the capacitor, and A_{ndiff}, the area of the underlying n-type diffusion plate. Thus, for a sufficiently large capacitor, A_{poly}, and hence A_{ndiff}, approximate A_{nwell}. This happens because the differences between them are fixed by layout design rules. As the capacitor grows, the area devoted to accomplish these rules represents a diminishing fraction of the total device area. Thus, let us consider that:

$$A_{\text{poly}} \approx A_{\text{eff}} \approx A_{\text{ndiff}} \approx A_{\text{nwell}} \tag{4.22}$$

Then Eq. 4.21 can be rewritten in this form:

$$\frac{dv_c(t)}{dt} \approx -\frac{q\left(\frac{D_p p_{n0}}{L_p} + \frac{D_n n_{p0}}{L_n}\right)}{C_a} = -r_{\text{self}} \tag{4.23}$$

where a self-discharge rate, independent of the capacitor size, is defined:

$$r_{\text{self}} = \frac{q}{C_a}\left(\frac{D_p p_{n0}}{L_p} + \frac{D_n n_{p0}}{L_n}\right) \tag{4.24}$$

A typical value for this ratio in a $0.5\mu\text{m}$ standard CMOS technology can be 50mV/s.

Then, the shape of the voltage across the sampling capacitor can be obtained by integrating Eq. 4.23, what yields

$$v_c(t) = v_c(0) - r_{\text{self}} t \tag{4.25}$$

It shows that $v_c(t)$ decays linearly during the hold period. A maximum storage time can be defined in terms of the accuracy requirements. For an equivalent resolution of N bits, and a full scale range of the input signal given by $2V_A$, the maximum storage time (t_{strg}) is the period in which the difference between V_c and the initially stored voltage does not exceed $V_A/2^N$, that is $1/2$ LSB:

$$v_c(0) - v_c(t_{\text{strg}}) = r_{\text{self}} t_{\text{strg}} \leq V_A/2^N \tag{4.26}$$

thus

$$t_{\text{strg}} = \frac{V_A}{2^N r_{\text{self}}} \tag{4.27}$$

that can be typically in the 200ms range for a 10mV error. These figures, however, must be understood only as orientative because of the strong influence of operation temperature in the leakage currents —they can increase even 3 orders of magnitude from 27C to 70C. Also, the incidence of light on the circuit surface can seriously degrade the contents of the memory because of the light induced generation of an extra amount of carriers.

4. 2. Analog memory circuits: a survey

In the previous section, the major sources for misbehaviour of a simple analog memory circuit have been reported. In order to overcome the effects of the non-idealities exposed, several alternatives for the implementation of the analog memory are going to be contemplated. Those alternative proposals try to cancel out some of the effects of the previously reviewed phenomena. But in most of the cases, a compromise must be taken and we will have to decide if the traded-off magnitudes still fall within the specifications. Therefore, before examining different implementation choices, we will analyse the characteristics of an analog memory device, keeping in mind that it will be intended for analog image and video signal processing.

4. 2. 1. Characteristics of an analog memory

Any circuit, or device, conceived to function as an analog memory (AM), must display some distinctive features —storage time, resolution, etc.— which help us characterize its behaviour. Before comparing different implementations of this functionality, it is convenient to enumerate the most important properties of an AM circuit [Hori92]. At the same time, and because we will be interested in image and video signal processing within the CNN framework, that is, we have a specific application field for which the circuit will be designed, it seems appropriate to evaluate these features from the point of view of the system's specifications.

Non-volatility

Its means, the ability to maintain the stored information from one update to another. We must distinguish here between power-on and power-off non-volatility. For instance, power-off non-volatility is shown by EEPROM devices and floating gates, in which disconnection of the power mains does not imply any information loss. Power-on non-volatility is what any RAM —either static or dynamic, as long as a refreshing scheme is provided— device exhibits. As long as the power supply is continued, any information stored in a particular memory location remains there until it is intentionally overwritten. If the power fails, information is lost, therefore, this is power-on non-volatility. The mechanisms to achieve non-volatility can be quite elaborated, but, ultimately, leakages through parasitics renders endless non-volatility impossible. We will always talk about a certain storage time, that is the maximum time for which the prescribed accuracy can be guaranteed. Beyond that point in time, the stored analog information will have degrade above the accuracy limits. For the applications contemplated in our dissertation, due to the high-speed of operation of the analog parallel array processors described, a storage time of 100-200ms should be enough. In addition, if the circuit is going to be a cache memory [Poll90], power-off non-volatility is not necessary.

Equivalent resolution

Although the term resolution is usually applied to digital signal processing systems and tools, it is convenient to extend this concept to the operation of certain analog circuits. The equivalent resolution in an analog circuit is just the dynamic range, the quotient between the maximum signal that the system can handle and the minimum signal that the circuit will properly discern from noise:

$$\mathrm{DR} = \frac{V_{\max}}{V_{\min}} \tag{4.28}$$

this ratio can be expressed as the raw figures, as a percentage, in decibels or in bits, that is, accounting for the number of bits that would be necessary to represent such a dynamic range:

$$\mathrm{N} = \log_2 \mathrm{DR} \tag{4.29}$$

Thus, consider an accuracy level in the $0.8\text{-}1.5\%$ range, understanding as accuracy the ratio between the allowed error and the full-scale range of the signal, in other words, the inverse of the dynamic range defined in Eq. 4.28. Then a system with a DR in between 66.67 and 125 is needed. This represents an equivalent resolution of $6\text{-}7\mathrm{bits}$.

These are the resolution, or, equivalently, accuracy levels required for early-vision applications and low-level focal plane image processing. In fact we are dealing with bio-inspired models for the parallel array processor, and the human eye finds it very difficult to perceive differences of only one level of gray out of 256 unless they are closely assembled, but this is because our ability to detect borders. For instance, the gray rectangles partially covered by the black shape in Fig. 4.4, are the 128th and 127th in a scale with 256 divisions between white and the black —this means an equivalent resolution of $8\mathrm{bits}$. These rectangles have been repeated below the first figure, but now the black mask have been taken away. It is clear that from the image above, those rectangles did not seem to be as different as they result to be from what is seen below.

Therefore, we will be aiming for a moderate resolution. In addition, cooperative phenomena derived from the parallel processing nature of CNNs, like hyperacuity [Werb95], will result in improved perceptual features even for moderate resolution analog hardware.

High-speed, random and non-destructive access

A characteristic that involves both architectural and circuit design considerations is the modality of the access. First of all, and being part of the system specifications, narrow access times to the memory device are expected. If the circuit deals with images of practical dimensions and they have to be processed real-time, updating and downloading times —think as information being up- and downloaded from a memory cell that acts as a container— around $100\mathrm{ns}$ will be common. As referred above, in order to meet these high-speed requirements both architectural, like the proper

FIGURE 4.4. Illustrating some perceptual abilities and limitations of the human eye.

sizing of the signal buses or the design of serialization and deserialization strategies, and circuital solutions, like trading-off memory cell packing for better driving capabilities, must be contemplated.

Another important aspect of the memory access is randomness. It can be either sequential or random, or it can have a certain degree of randomness between the strictly sequential and purely random. The type of access is strongly related with the intended access times and duration of the storage function. Long-term data storage is usually based on sequential devices in which access times are rather large. A cache memory [Goor89], on the opposite extreme, is a short-term memory which requires the highest access speed.

And finally, and this will be a matter of circuit design, recovering the stored information can be destructive or non-destructive, depending on the effects of reading on the stored samples. Although any transmission of information involves energy transactions, some circuits compensate for the degradation introduced by interacting with the stored information. Therefore, we will consider the reading non-destructive as long as the degradation of the sampled signal does not go beyond some specific limit.

Some analogic algorithms designed for the CNNUM require repeated reading and writing to a specific location of the memory. Thus, random access to any memory register should be provided. For the same reason, because access to some memory locations might be required several times in an analogic program, reading should not affect their contents.

Size of the memory cell

Finally, another characteristic with strong implications in the system design, the memory cell size. The larger the number of cells in memory array, the larger the pictures that can be stored and processed, or the more frames that can be maintained for recurrent algorithms. In addition, for a required size of the memory buffer, the smaller the memory cell, the more cells that can be accommodated in a single chip, the less number of chips are needed. Therefore, the memory cell should be small enough in order to integrate as many as possible in a single chip.

But this drive conflicts with the pretended retention times and the required accuracy levels. For smaller memory cells, the influence of parasitics either during signal acquisition or in the hold time is more noticeable, thus degrading performance. Even high-speed requirements can be affected. Although smaller cells are faster, they have less driving capabilities, thus in order to have faster writing periods we might be sacrificing reading times. This will ultimately depends on the application.

4. 2. 2. Comparison of technology alternatives

The applications of analog signal storage in analog VLSI signal processing are diverse. From synaptic weight storage in neural networks to scanning delay-lines in video signal processing and also oversampled modulators for A/D and D/A interfacing and communications. Therefore, many different alternatives for the implementation of a reliable analog memory device has been reported in literature. An overview of these circuit and device level solutions is given in [Hori92]. These alternatives for the short- and mid-term storage of analog signals can be classified into five groups: digital memories, non-volatile semiconductor devices, memory capacitors, active analog memories and system level memories.

Digital memory devices

Conventional digital random access memory (RAM) circuits can be employed for the storage of analog signals in binary format. They have a good retention and fatigue characteristic, since binary signals can be easily reshaped [Geig90]. The CMOS static RAM cell (SRAM) pictured in Fig. 4.5(a), consists in a pair of CMOS inverters constituting a latch. Each one's output drive the other's input closing a positive feedback loop that saturate the inverters —that can be seen as high-gain single-rail inverting amplifiers— to complementary logic levels. The power consumption is negligible during hold time and the stored logic levels cannot be degraded by construction. The dynamic RAM cells (Fig. 4.5 (b) and (c)) are named in this way because the charge stored in the capacitor is leaking during hold time, so the stored voltage is of a dynamic nature. In order to avoid degradation and loss of stored information, a process of memory refresh must be provided. Memory refreshing usually consists in reading and rewriting of the recovered data back into the memory cell. The three-transistor cell, in Fig. 4.5 (b), allocates some buffering capabilities inside the

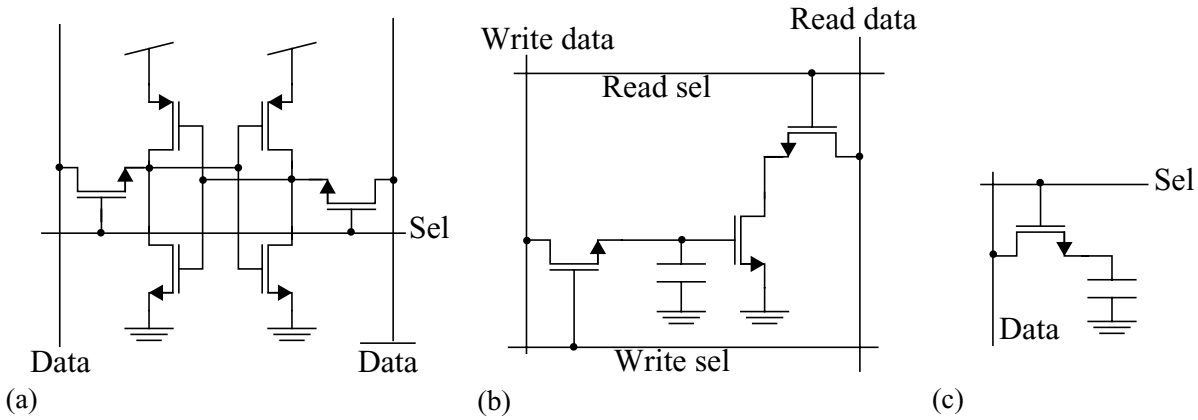


FIGURE 4.5. CMOS (a) SRAM, (b) 3-transistor and (c) one-transistor DRAM cells

memory cell, thus making it robust against reading accesses, though refreshing is still required as leakages exist. The one-transistor cell (Fig. 4.5 (c)) allows for larger densities, but rather involved sensing circuits and strategies must be implemented to compensate the lack of driving force at the cell level. Added to this, there are digital memories with power-off non-volatility. These are the ROM (read-only memory) devices, in which data is hard-coded by the presence of a pull-up or pull-down transistor in the proper node of a wire matrix (Fig. 4.6). Although ROM devices are random access, they can not be updated and, hence, result useless for the applications under consideration.

However, the main drawback for using digital memories for the implementation of an analog memory functionality is the unavoidable presence of A/D and D/A conversion circuitry. As we will see later, even the use of state-of-the-art circuits and techniques results in power-hungry implementations with quite large silicon area occupation.

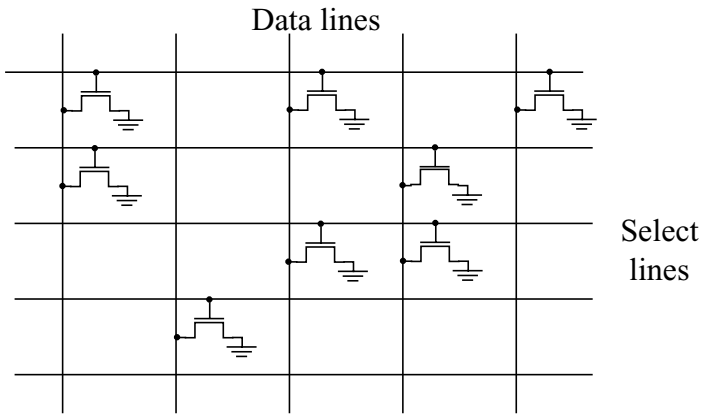


FIGURE 4.6. ROM memory array

Nonvolatile semi-conductor memories (NVSM)

A different approach to the problem of analog signal storage can be conducted from the device level. Some special technology processes can host a number of semiconductor devices with improved storage capabilities. They are usually built by modifying the conventional MOS structure [Tsiv87]. For instance, a layer of silicon nitride (Si_3N_4) inserted between the gate metal (or polycrystaline silicon) and the silicon dioxide in a n-channel MOS transistor [Chan76]. This structure (Fig. 4.7(a)) allows changes in the threshold voltage of the transistor by means of the trapped charges in the oxide-nitride interface.

Similarly, threshold voltage can be varied in floating-gate devices as the FAMOS, what stands for floating-gate avalanche-injection MOS [Froh74]. The FAMOS structure, seen in Fig. 4.7(b), consists in a MOS transistor with a second polysilicon gate that is buried in the silicon dioxide between the control gate and the transistor channel area. When a relatively large voltage ($15\text{-}20\,V$) is applied between the drain and the source of the device, a number of electrons with a large energy level is induced in the channel area. This hot electrons, carry enough energy to jump to the floating gate, triggering an avalanche injection of charge. This occurs with oxide as thick as $100\,nm$. Once this large programming voltage is

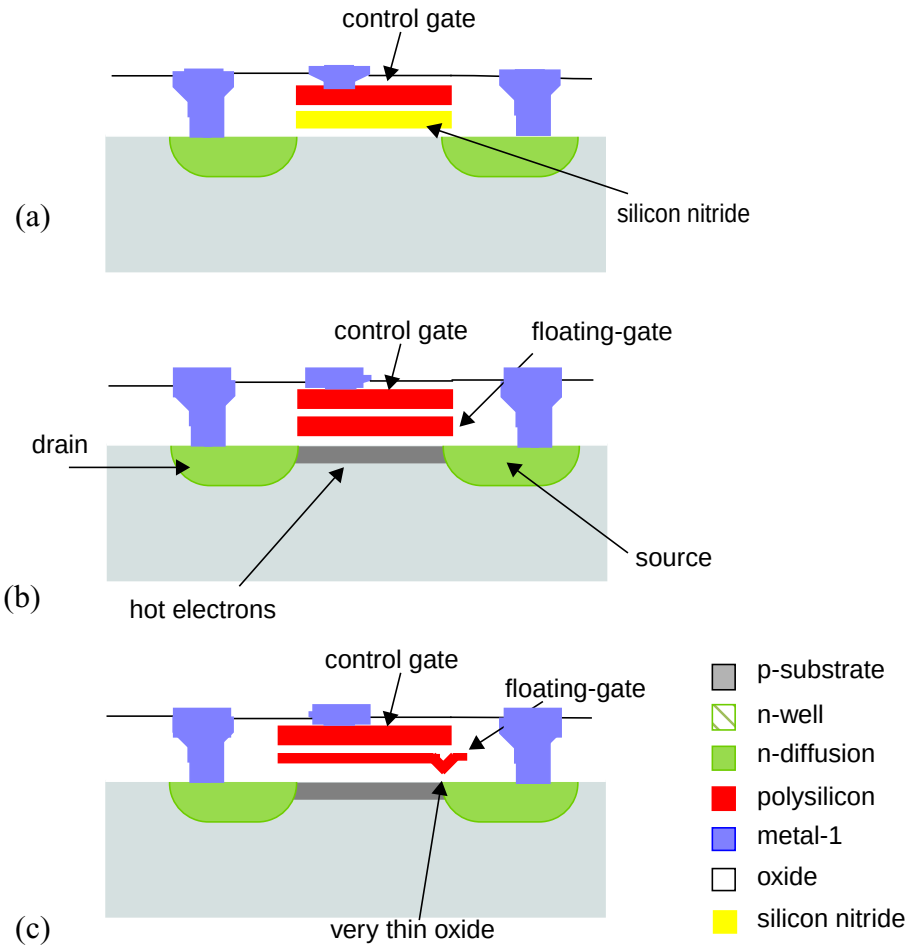


FIGURE 4.7. Structure of the (a) silicon nitride-oxide trap, (b) FAMOS device and (c) FLOTOX device.

released, the resulting threshold voltage is much higher than the original because of the electrons trapped in the floating-gate. Typical programming times are of 5-10μs. The leakages in this structure are so small that the discharge time constant for a FAMOS device can be of tens of years. In order to erase the memory contents, *i. e.* to release the trapped electrons, the floating-gate device must be exposed to UV-light for a period of up to several minutes. This device constitutes the core cells of EPROM (erasable-programmable ROM) chips, in which a large number of them are packed together, thus they are all erased in parallel.

Another floating-gate structure, depicted in Fig. 4.7(c), is the FLOTOX device. FLOTOX means floating-gate tunnelling oxide [Kolo86]. Its operation is based on the Fowler-Nordheim tunnelling effect. By using the appropriate programming voltages, typically smaller than those requires in FAMOS programming, sufficiently high energy electrons are introduced in the channel area. Those arriving to the very thin oxide region with enough energy tunnel to the floating-gate. This phenomenon takes place for oxide thicknesses on the 10nm range. It can be made reversible by the application of the appropriate voltages, thus the FLOTOX devices have been employed to implement EEPROM (electrically-erasable-programmable ROM) chips. The FLASH memory devices (FLASH EEPROM) uses avalanche injection for programming and Fowler-Nordheim tunnelling to erase the device.

The main drawbacks of using NVSM devices to implement an efficient analog memory derive from the fact that they were initially developed for being used as digital ROM devices. Therefore, accurate control of the stored charges is not easy to implement, large voltage pulses are required for introducing changes into the stored values. On top of that, because these devices were thought to be ROM devices, erasing and rewriting the memory contents can be extremely slow.

Capacitors

A capacitor with a pass transistor (Fig. 4.1(a)) provides the simplest way to store analog, and digital, information in VLSI circuits. We have made use of its simplicity to explain the most common error sources found in practice when sampling and holding signals. It is fully compatible with standard CMOS technologies, where it can be built by using poly-to-poly structures or poly over diffusion, or even by a short-circuited MOS transistor, as depicted in Fig. 4.8.

Although affected by the presence of parasitics and the influence of a number of non-ideal effects, reported in the previous sections, the nominal operation is quite simple, and based on the ability of the capacitor to store charges. Because of its constitutive equation, any change in the voltage applied to the terminals of the capacitor produces a change in the stored charge. In the ideal case, the relation between the instantaneous amount of charge in the capacitor plates, $q_C(t)$, and the instantaneous voltage applied to them, $v_C(t)$, is linear and time-invariant. The proportionality constant is called capacitance:

$$q_C(t) = C_m v_C(t) \tag{4.30}$$

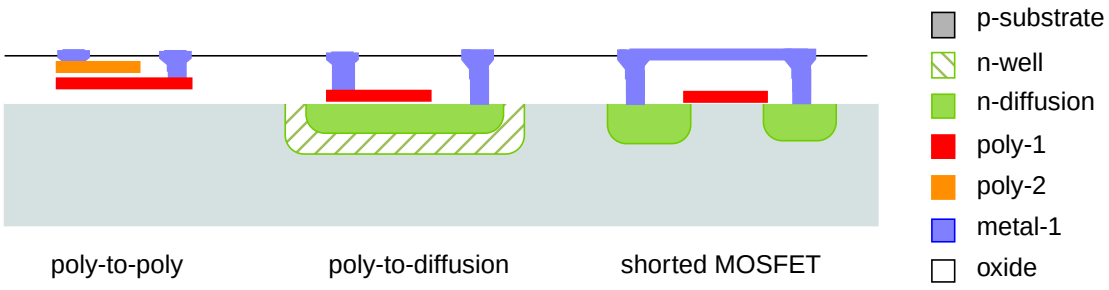


FIGURE 4.8. CMOS compatible structures for the implementation of capacitors

Then, ideally, if the capacitor is fed with signal $v_i(t)$, the amount of charge stored in its plates will track this input signal, ending in a $q_C(t_s)$ that will be proportional to the value of v_i at the sampling instant t_s. If this charge in the capacitor can be sensed without redistribution, then we will have an analog memory mechanism. In order to avoid signal degradation because of charge redistribution when sensing the stored analog value, a high-gain amplifier is employed to supply the necessary charge. This can be observed in the sample and hold circuit of Fig. 4.9(a). At the writing phase (ϕ_1 is active), the capacitor C_s is being updated by a voltage $-V_i(n)$, while the capacitor C_2 is reset. Charges in the capacitors at this phase are:

$$Q_{C_1}(n) = -C_1 V_i(n) \qquad \text{and} \qquad Q_{C_2}(n) = 0 \tag{4.31}$$

At the second phase (ϕ_2 in ON and ϕ_1 is OFF), $V_{C_1}(n+1)$ settles to 0V after a transient, therefore, $Q_{C_1}(n+1) = 0$. The difference in the charge in C_1 between the two phases can only be absorbed by C_2, thus:

$$\Delta Q_{C_2} = Q_{C_2}(n+1) - 0 = 0 - Q_{C_1}(n) = \Delta Q_{C_1} = C_1 V_i(n) \tag{4.32}$$

that results in:

$$V_o(n+1) = \frac{1}{C_2} Q_{C_2}(n+1) = \frac{C_1}{C_2} V_i(n) \tag{4.33}$$

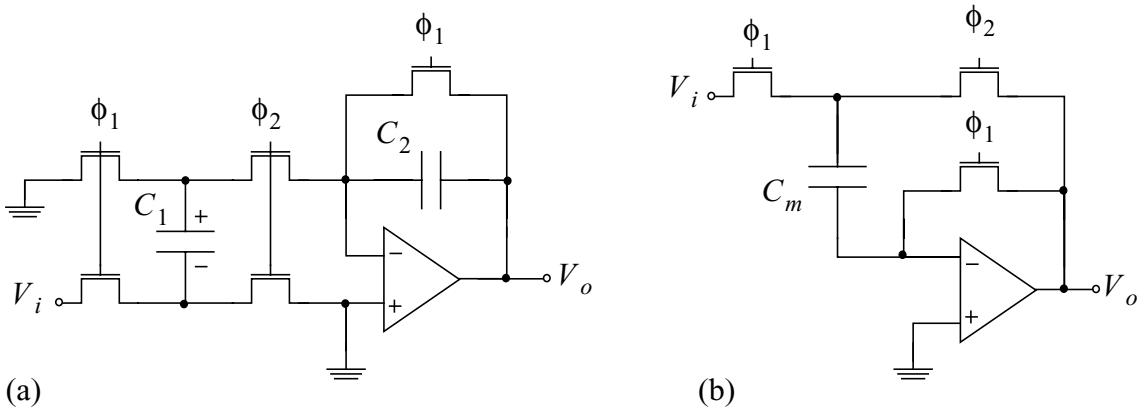


FIGURE 4.9. Sample and hold circuits.

In a first approximation, the output voltage follows the input. If V_o has to replicate V_i values, then C_1 and C_2 has to be matched. This can be avoided by using one single capacitor for sampling and sensing, like in the unitary-gain buffer of Fig. 4.9(b). This circuit is offset insensitive. In the first clock phase, ϕ_1, the capacitor C_m is charged by a voltage $V_i(n) - V_{OS}$, being V_{OS} the input-referred offset-voltage of the opamp [Solo74]. Therefore, the charge stored in the capacitor is:

$$Q_{C_m}(n) = C_m[V_i(n) - V_{OS}] \tag{4.34}$$

After that, once ϕ_1 turns off and ϕ_2 becomes active, the charge in the capacitor is given by:

$$Q_{C_m}(n+1) = C_m[V_o(n+1) - V_{OS}] \tag{4.35}$$

but, because the bottom plate of C_m is isolated in the second phase (ϕ_2), the charge in the capacitor does not change, thus $Q_{C_m}(n)$ and $Q_{C_m}(n+1)$ must be equal, rendering:

$$V_o(n+1) = V_i(n) \tag{4.36}$$

We have seen in the previous sections that the use of real MOS switches can cause an important deviation from the behaviour described above. Later, we will discuss several strategies to reduce the sampling errors in these circuits, but now we will consider another major limitation. Due to leakages in the switches and capacitors structures (Fig. 4.3) — reverse biased diodes and subthreshold channel conduction— the stored voltage degrades with time. All of these leakage sources are strongly dependent on the temperature and can be highly reduced by cooling. In order to compensate for this voltage degradation, circuit strategies like twin-capacitor differential storage can be employed. These solutions constitute the group of the active analog memories.

Active analog memories (AAM)

The AAMs are defined as dynamic long- or mid-term analog memories in which leakage compensation or memory refresh mechanisms are present on-chip. Memory performance and characteristics can be adjustable by design, as they comprise a circuit-level approach to analog signal storage, as opposed to the device-level approach of the NVSM. These memory circuits are fully compatible with standard CMOS technology. There is no need, hence, for additional steps in the fabrication process.

Amongst the different kinds of AAM that have been proposed, we have selected a few examples in order to realize the basic principles in which most of them are based. Let us consider the review, in the first place, of the twin-capacitor analog memory [Hori90]. The principle behind this circuit is the similarity of the voltage degradation occurring in two equally designed capacitors. If C_1 and C_2 (Fig. 4.10) are matched,

so their parasitics will be. Suppose that we have stored a voltage V_i in C_1[i]. And simultaneously, the reference voltage V_{ref} has been recorded in C_2. Despite the sampling errors, both stored values suffer identical degradation, if the parasitic diodes are matched, because they will most probably be at the same temperature. Then, although degradation can be severe, the difference signal $V_{\text{diff}}(t) = V_{C_2}(t) - V_{C_1}(t)$, remains constant and equal to the initial value: $V_{\text{diff}}(0) = V_i - V_{\text{ref}}$. In order to avoid degradation beyond a certain limit, the voltage in C_2 is compared with a certain threshold, V_{th}. If V_{C_2} falls below this threshold a refresh pulse is triggered. With this pulse, ϕ, V_{ref} is restored in C_2 while the difference signal, V_{diff}, which remains constant during the whole process, is sensed and added to the reference to generate a copy of the original signal $V_i = V_{\text{ref}} + V_{\text{diff}}$ which is restored into C_1.

A different method for leakage compensation is the use of a A/D-D/A loop, like the one depicted in Fig. 4.11(a). In this circuit, two non-overlapped clock signals allow successive sensing and restoring the sampled voltage. In the sense phase (ϕ_s ON), the stored voltage is detected and converted to a digital code. This code is converted back to an analog signal which is employed to update V_c in the refresh phase (ϕ_r ON). The frequency of the refresh (frequency of the clocks ϕ_s and ϕ_r) must be large enough to avoid the stored voltage degrading more than one LSB. If this occurs, if V_c falls below the level corresponding to the adjacent digital code before it is sensed, then the memory contents are overwritten with an incorrect value. This method permits by construction to establish a precise accuracy control. Care must be taken not to degrade the stored voltage because of charge redistribution when sensing V_c. Active sense circuits like those described in the previous sections can be applied. Consider also that this strategy can be applied to a relatively large number of analog memory registers by multiplexing the A/D-D/A loop in time. This method has been employed in [Lina91] for the implementation of a bidirectional associative memory (BAM) chip.

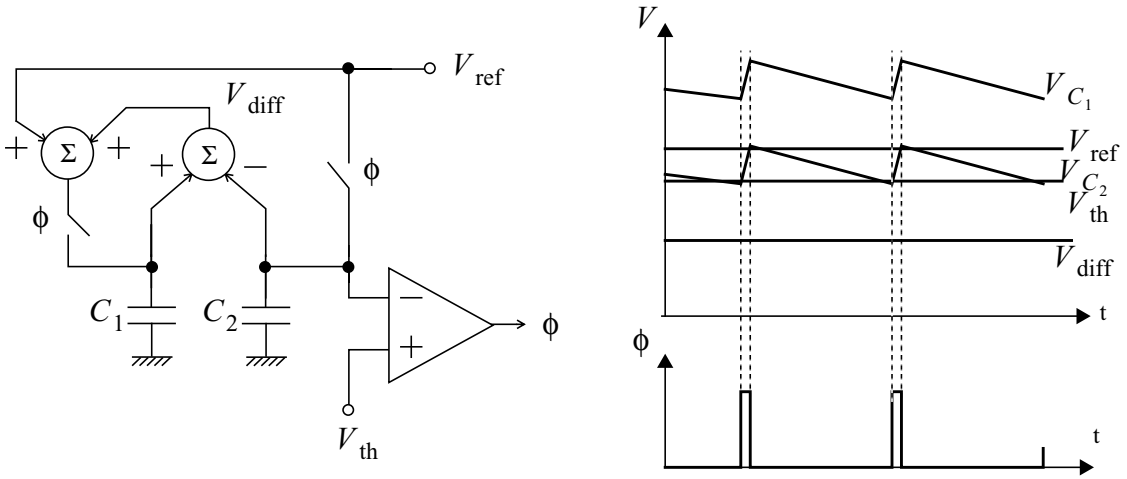


FIGURE 4.10. Conceptual diagram of the twin-capacitor active analog memory.

i. Of course, what we have stored is a certain amount of charge, but being a linear time-invariant capacitor, the voltage between its two plates is proportional to the amount of charge stored in it, so it is not strange to say the *stored voltage*.

A third example of leakage compensation methods is the staircase refresh technique of Fig. 4.11(b). Here, a reference staircase signal, V_s, is compared to the stored voltage in each clock cycle. If V_c drops below the value of the corresponding step of the stair, a comparator triggers a refresh pulse and the voltage in the capacitor is restored to the voltage level of last step. A clocked latch is employed to reset the refresh pulse. This principle is described in [Vitt91].

There are more methods for leakage compensation [Hori92]. Some apply a phase detector where the stored signal is compared with a triangular wave and the phase detector is employed to generate the refresh pulse. Others use a charge pump to transport the charge between two capacitors, both of them updated to the same voltage but degraded in opposite directions, then the charge pump restores the initial state. Other circuit employs a frequency locked loop composed by a voltage controlled oscillator (VCO) and a F/V (frequency-to-voltage) converter. The frequency, or the control voltage of the VCO, gets locked in the loop, which can be therefore employed as an analog memory. Some other circuit use a current latches, but these can be seen as current-mode digital memories.

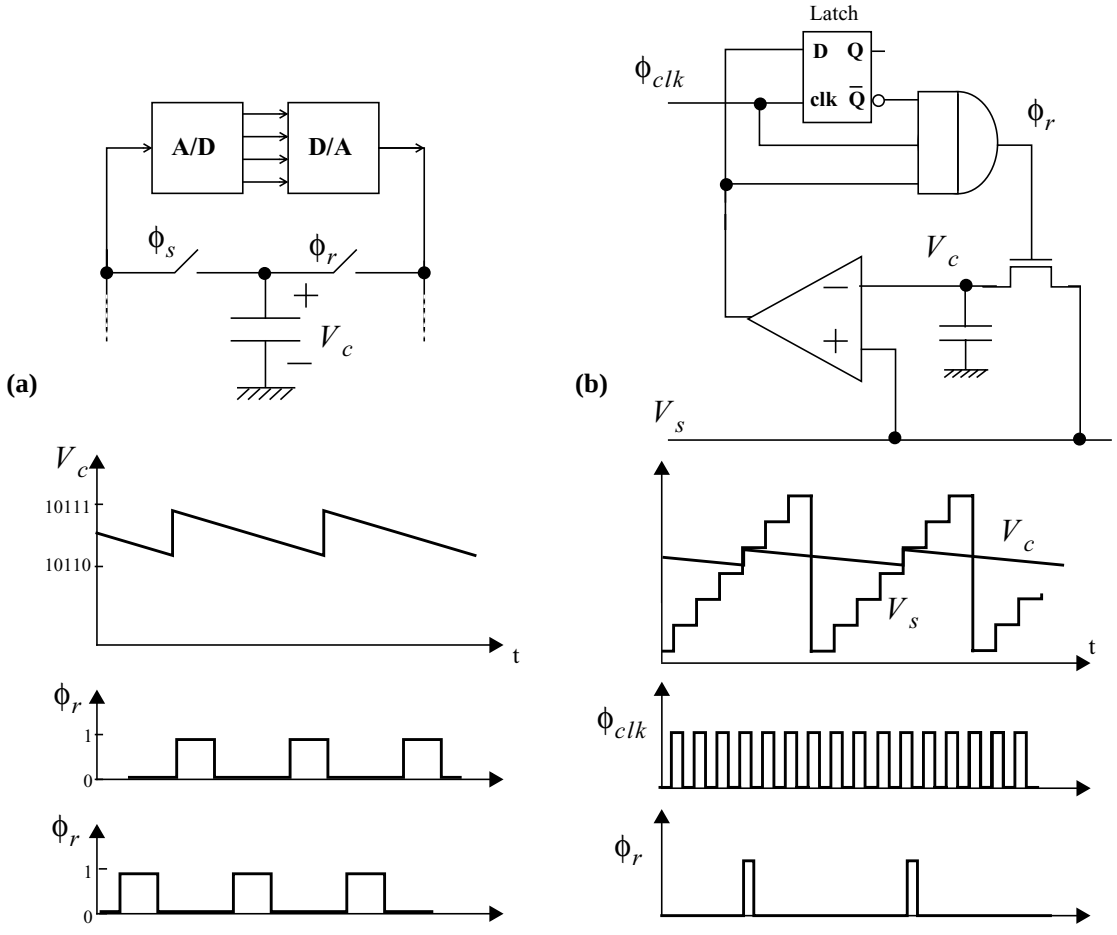


FIGURE 4.11. Active analog memories (a) A/D-D/A loop, (b) staircase refresh.

Network level memory

Finally, and for the sake of completion, we are going to cite a type of memory that is not implemented in a device or a circuit, but it has roots in the collective behaviour of a whole network. In the context of neural networks, memory refers to relatively enduring alterations of the neurons resulting from the interaction of a network with its environment [Hayk94]. Here, stability and non-volatility of the stored information is not implemented at device or circuit level. Instead, a brainlike distributed memory model is applied. Its operates by association, so it is called associative memory. Its fundamental property is its ability to map output patterns of neural activity onto input patterns. In other words, these networks are capable of associating a particular stimulus with its corresponding response pattern by means of an internal representation. This is achieved by changes in the synaptic connections between neurons. This changes are retained by each synapse with the help of the memory devices and circuits reviewed before. Once a input-output pattern has been internally apprehended, it can be recalled by the application of resembling stimuli. If an input pattern similar to the one employed in the learning phase is presented, the associative memory recalls the corresponding response pattern by adopting an internal configuration that resembles the learned state, despite imperfections in the input stimuli. An interesting property of this collective memory mechanism is that regardless of the neurons not being highly accurate, noise insensitive processing units, the associative memory is largely resistive to noise and damages. In other words, collective memories exhibit a more robust behaviour than can be expected from their moderate precision building units [Kosk92].

This network level analog memory, although an interesting phenomenon, does not seem to be applicable in the context of finding a short-term memory buffer implementation for the CNN chipset.

4. 3. A CMOS analog RAM chip

The cache memory is a concept extracted from the context of computers architecture [Poll90]. It is based on the locality in signal processing. Commonly, the pieces of information related to those already being processed are stored in a nearby location of the memory [Goor89]. Therefore, granting a high-speed connection between the processor and a privileged section of the memory —what is referred as cache memory and retains temporarily the data that, because of their proximity to those under processing, are suppose to be most probably going to be related with them— a higher speed of operation can be accomplished. Besides, some complex image processing tasks require the storage of intermediate results. Therefore, a reliable memory device, faster than massive longer-term memories, will highly improve the speed of the system when repeated access to some stored information is necessary, e. g. for the implementation of a multipath analogic algorithm. A straightforward real-

ization of the required functionality would be the use of conventional digital RAM. But, interfacing this digital memory to the analog processor, however, would require the use of A/D and D/A converters. This, on one side, could compromise the intended I/O rates, and on the other, outrun the low-power directives that target a higher integration of the CNN chipset. In order to realize direct data communications between the memory and the analog parallel array processor, avoiding data conversion, the implementation of an analog RAM chip is proposed.

Multiple chips related with analog signal storage have been reported in literature. These works cover different applications in which short- and mid-term storage of analog information is required, like image processing or neural networks. In some cases, a special technology process has been employed for the implementation of a nonvolatile memory device, like those reviewed before [Yong96], [Dior95], [Devo93]. The use of extra layers or customized processes, however, can seriously compromise production costs and the compatibility with a conventional digital environment. Some other chips contain a CMOS realization of a scanning-delay line for video signal processing [Nish93b]. Here, random and non-destructive reading of the memory contents is not provided because it was not foreseen by the application. A high-speed SC sampling circuit is reported in [Hall89]. This circuit captures analog waveforms originating at an array of sensory devices. No random access or non-destructive reading of the memory contents is available either. Besides, a small capacitor per pixel is employed in [Simo95] for the CMOS implementation of a motion-detection algorithm, which is a kind of local analog memory. Another reported work presents an analog RAM for early vision applications [Fran92]. In this case, accuracy relays on mismatch compensation and no switching error reduction strategies are adopted. In this section we will present a prototype of an analog RAM (aRAM) chip developed in a $0.5\mu m$ CMOS single-poly triple-metal technology [Carm98]. It will be compared to the above-mentioned implementations.

4. 3. 1. Circuit design

Sample and hold lines

The aRAM chip presented here is composed of several sample and hold (S/H) circuits based on the unity-gain offset-free circuit of Fig. 4.9(b). This circuit, also known as Gregorian's S/H can be employed for bottom-plate sampling of signals. This technique allows reducing the harmonic distortion introduced by signal-dependent switching errors, as we will see later. Hence, Fig. 4.12 displays a set of N analog memory registers associated to the same sensing amplifier. Its realizes a non-destructive recovery of the stored signal. It has been previously employed in image processing applications and reported in [Mats85]. The nominal behaviour of the S/H circuit has been already described, yielding an offset-free tracking of the input signal. For instance, if a sample of V_i is

stored in the k-th capacitor, C_k, being $1 \le k \le N$, and V_{OS} is the input referred offset voltage of the opamp, the output in a first approximation is:

$$\left.\begin{aligned} V_{c_k}(n) &= V_i(n) - V_{OS} \\ V_o(n+1) &= V_{c_k}(n) + V_{OS} \end{aligned}\right\} \quad \text{thus} \quad V_o(n+1) = V_i(n) \tag{4.37}$$

Assume now that the operational amplifier (opamp) has a large but finite gain, A_o, and parasitics are not negligible. In the first phase (ϕ_1 ON), and considering only one single memory capacitor, C_k, the charges stored at each capacitor, the nominal and the parasitic, are:

$$\left.\begin{aligned} Q_{C_k}(n) &= C_k\left[V_i(n) - \frac{A_o}{1+A_o}V_{OS}\right] \\ Q_{C_p}(n) &= C_p\left(\frac{A_o}{1+A_o}\right)V_{OS} \end{aligned}\right\} \tag{4.38}$$

In the second phase, ϕ_2 ON, the stored charge is:

$$\left.\begin{aligned} Q_{C_k}(n+1) &= C_k\left[\frac{1+A_o}{A_o}V_o(n+1) - V_{OS}\right] \\ Q_{C_p}(n+1) &= C_p\left[V_{OS} - \frac{1}{A_o}V_o(n+1)\right] \end{aligned}\right\} \tag{4.39}$$

Because the changes in the stored charge in C_k have been transmitted to C_p, it can be said that:

$$Q_{C_k}(n+1) - Q_{C_k}(n) = \Delta Q_{C_k} = \Delta Q_{C_p} = Q_{C_p}(n+1) - Q_{C_p}(n) \tag{4.40}$$

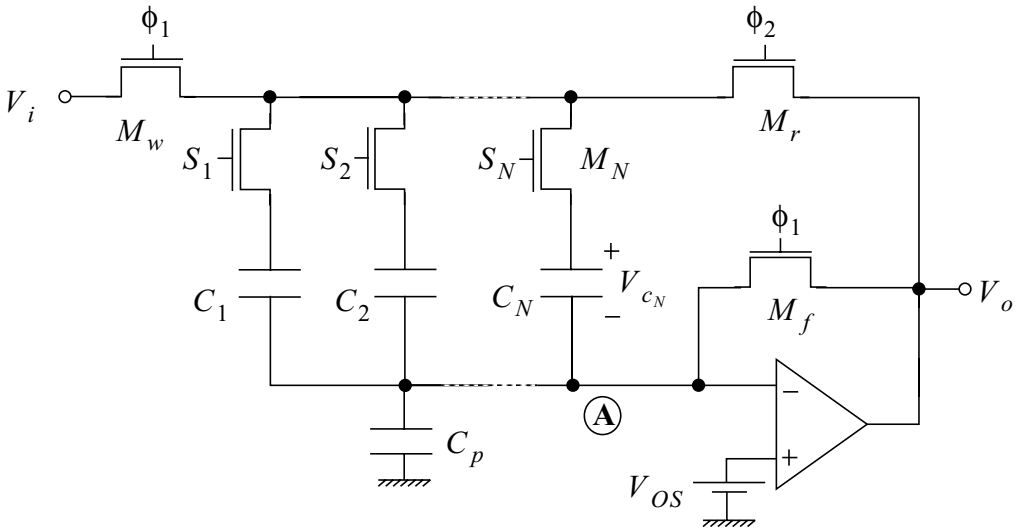


FIGURE 4.12. S/H line schematic including parasitic capacitance and opamp offset.

and this leads to:

$$V_o = \frac{1}{1+\frac{1}{A_o}\left(1+\frac{C_p}{C_k}\right)}\left[V_i+\frac{1}{1+A_o}\left(1+\frac{C_p}{C_k}\right)V_{OS}\right] \tag{4.41}$$

Making $\varepsilon = A_o^{-1}(1 + C_p/C_k)$ we have:

$$V_o \approx \frac{1}{1+\varepsilon}(V_i+\varepsilon V_{OS}) \qquad \text{or} \qquad V_o \approx (1-\varepsilon)(V_i+\varepsilon V_{OS}) \tag{4.42}$$

that can be approximated by:

$$V_o \approx (1-\varepsilon)V_i+\varepsilon V_{OS} \tag{4.43}$$

These errors in the recovered wave sample appear as a gain-error and an offset. The gain-error causes a little attenuation of the signal with respect to the original samples, i. e. the gain of voltage follower being less than unity. The second error term produces and offset. It proceeds from the opamp and is attenuated by the dc gain. The error committed in the recovery of the sample $V_i(n)$ is given by:

$$V_o(n+1)-V_i(n) \approx \frac{1}{A_o}\left(1+\frac{C_p}{C_k}\right)[V_{OS}-V_i(n)] \tag{4.44}$$

where it can be seen that, as $V_i(n)$ approaches V_{OS}, the offset term compensates for the gain error. Concerning the relative error,

$$\left|\frac{V_o(n+1)-V_i(n)}{V_i(n)}\right| \approx \frac{1}{A_o}\left(1+\frac{C_p}{C_k}\right)\left|\frac{V_{OS}}{V_i(n)}-1\right| \tag{4.45}$$

it saturates to $(1/A_o)(1 + C_p/C_k)$ for the larger amplitudes of the input signal. For the smaller signals, the relative error takes off, but this happens in a range where noise is the limiting factor, i. e. much smaller than the valid signal range. Thus, for an equivalent resolution of N-bits, that is, a relative error below $1/2^{N+1}$, then:

$$A_o > 2^{N+1}\left(1+\frac{C_p}{C_k}\right) \tag{4.46}$$

for example, 7bits of resolution with parasitics of about the same size of the nominal capacitor yields a required dc gain for the opamp of 54.2dB.

Another aspect of the S/H circuit that demands our consideration is the output swing, OS, of the opamp. On one side, an equivalent resolution around 7bits requires a dynamic range of 45dB. On the other, evaluating the formulae in the previous sections, the errors derived from switching can reach on average 20mV. In order to maintain such dynamic range with this lower level for the signals, an OS of 2.6V is required. In other to accomplish this with a 3.3V power supply voltage a folded Cas-

code implementation of the opamp [Lake94] has been proposed (Fig. 4.13).

In this opamp, the dc gain is the product of the transconductance of the transistors in the input differential pair and the output resistance:

$$A_o = g_{m_1} R_o \tag{4.47}$$

where R_o is obtained as the parallel connection of the resistance seen from V_o towards M_6 and the one seen towards M_{10}, that is:

$$R_o = \frac{1}{g_{\mathrm{OUT}_6} + g_{\mathrm{OUT}_{10}}} \tag{4.48}$$

and then:

$$g_{\mathrm{OUT}_6} = \frac{g_{o_6}(g_{o_4} + g_{o_2})}{g_{m_6} + g_{mb_6}} \qquad \text{and} \qquad g_{\mathrm{OUT}_{10}} = \frac{g_{o_{10}} g_{o_8}}{g_{m_{10}} + g_{mb_{10}}} \tag{4.49}$$

Added to this, dynamic specifications for this opamp are the gain-bandwidth product (GB) and the slew-rate (SR). On one side, the input signals will be band limited to 4MHz, but the main part of its spectrum is at much lower frequencies, thus a GB = 20MHz will be enough. As the only high-impedance node is V_o, the GB is obtained by:

$$\mathrm{GB} = A_o \omega_c \qquad \text{where} \qquad \omega_c = \frac{1}{R_o C_o} \tag{4.50}$$

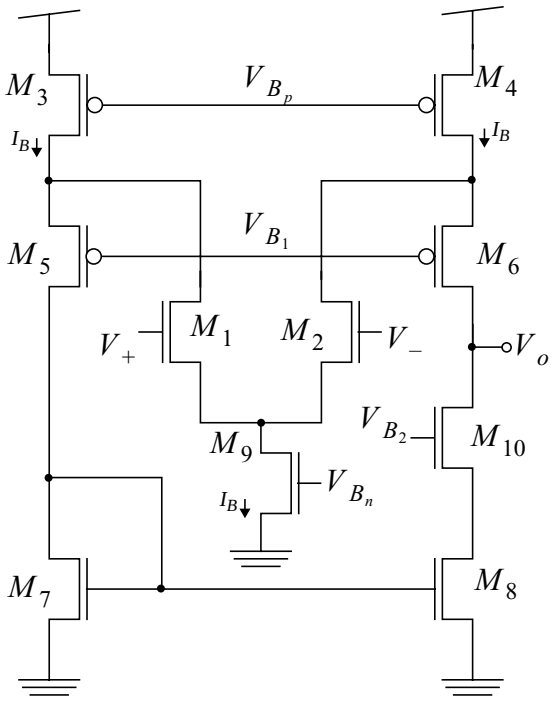


FIGURE 4.13. Folded-cascode opamp.

here, R_o is the output resistance and C_o is the capacitance associated with node V_o. This capacitance is the sum of the load capacitance, C_L which is going to be dominant, and the capacitances associated to the drains of M_6 and M_{10}. Operating, we arrive to:

$$\mathrm{GB} = \frac{g_{m_1}}{C_L} \tag{4.51}$$

where the GB is in rad/s. This help us dimension the differential pair:

$$\frac{W}{L} = \frac{(C_L \mathrm{GB})^2}{2\beta_n^* I_B} \tag{4.52}$$

A more critical specification is the slew-rate. In order to follow an input that is band limited to 4MHz, and can be up to 2.6V peak-to-peak amplitude, a slew-rate of 10.4V/μs is required. This fixes the necessary tail-current of the opamp, if the load capacitance is dominant, because:

$$\mathrm{SR} = \frac{I_B}{C_L} \tag{4.53}$$

and then $I_B = \mathrm{SR} C_L$.

Then, the differential pair can be dimensioned, but care must be taken not to jeopardize the phase margin, PM, being the first non-dominant pole at the node formed by the drains of M_2 and M_4 and the source of M_6. The only thing left is to tailor the R_o to match the desired dc gain. Final compromises are solved by matching considerations.

Switching error reduction scheme

The worst feature of the error caused by feedthrough and charge injection is its signal dependence. This causes an important amount of harmonic distortion to appear in the recovered waveform. In order to picture this effect, let us consider the large signal relationship between the input voltage V_i and the finally stored value V_c:

$$V_c = V_i + V_\varepsilon \tag{4.54}$$

where V_ε stands for the addition of the charge injection error and the feedthrough error:

$$V_\varepsilon = \Delta V_{c_{ch}} + \Delta V_{c_{feed}} \tag{4.55}$$

These errors were sketched in Eqs. 4.12 and 4.14. We can summarize them into:

$$V_\varepsilon = k_0 + k_1[V_i + V_T(V_i - V_{SS})] \tag{4.56}$$

where the two constants are $k_0 \approx (C_{gds}V_{SS} - \rho C_{ox}^{*}WLV_{DD})/C_m$ and $k_1 \approx (\rho C_{ox}^{*}WL - C_{gds})/C_m$. As can be seen, V_ε is input-dependent.

If V_c is expanded around the quiescent point $(V_i|_Q, V_c|_Q)$

$$V_c = a_0 + a_1 v_i + a_2 v_i^2 + a_3 v_i^3 + \ldots \tag{4.57}$$

$$\text{where } v_i = V_i - V_i|_Q \tag{4.58}$$

and

$$\begin{aligned}
a_0 &= V_c|_Q = V_i|_Q + V_\varepsilon|_Q \\
a_1 &= \left.\frac{\partial V_c}{\partial v_i}\right|_Q = 1 + \left.\frac{\partial V_\varepsilon}{\partial v_i}\right|_Q = 1 + k_1\left(1 + \frac{\gamma}{2\sqrt{\phi_B + V_i|_Q - V_{SS}}}\right) \\
a_2 &= \left.\frac{1}{2}\frac{\partial^2 V_c}{\partial v_i^2}\right|_Q = \left.\frac{1}{2}\frac{\partial^2 V_\varepsilon}{\partial v_i^2}\right|_Q = -\frac{k_1\gamma}{8}(\phi_B + V_i|_Q - V_{SS})^{-\frac{3}{2}} \\
a_3 &= \left.\frac{1}{6}\frac{\partial^3 V_c}{\partial v_i^3}\right|_Q = \left.\frac{1}{6}\frac{\partial^3 V_\varepsilon}{\partial v_i^3}\right|_Q = -\frac{3k_1\gamma}{48}(\phi_B + V_i|_Q - V_{SS})^{-\frac{5}{2}}
\end{aligned} \tag{4.59}$$

Notice that a linear dependence of V_ε on the input signal does not lead to harmonic distortion, as the second derivative of V_ε with respect to v_i is zero. It is the nonlinearity of the bulk-effect that ends promoting distortion. Therefore, as a preliminary result, a pass transistor without substrate-effect should be employed whenever the signal ranges allow it, because this eliminates harmonic distortion.

Now, consider that the input signal is of the form

$$V_i = V_i|_Q + v_i = V_i|_Q + A\cos\omega t \tag{4.60}$$

then the stored voltage can be expanded as a series of cosines of multiples of the fundamental frequency

$$V_c = b_0 + b_1\cos\omega t + b_2\cos 2\omega t + b_3\cos 3\omega t + \ldots \tag{4.61}$$

from where, by means of some trigonometric relations, we find that

$$\begin{aligned}
b_0 &= a_0 + \frac{1}{2}A^2 a_2 + \ldots \\
b_1 &= Aa_1 + \frac{3}{4}A^3 a_3 + \ldots \\
b_2 &= \frac{1}{2}A^2 a_2 + \ldots \\
b_3 &= \frac{1}{4}A^3 a_3 + \ldots
\end{aligned} \tag{4.62}$$

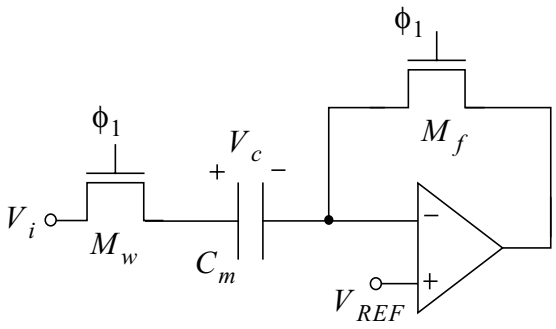


FIGURE 4.14. S/H circuit for bottom-plate sampling in the writing phase.

Then, the harmonic distortion corresponding to the n-th term is defined as the ratio between the amplitude of the n-th harmonic in Eq. 4.61 to that of the fundamental harmonic. For the first two we have:

$$\begin{aligned} \mathrm{HD}_2 &= \frac{b_2}{b_1} = \frac{1}{2}A\frac{a_2}{a_1} = \frac{1}{4}A\frac{\partial^2 V_\varepsilon}{\partial v_i^2}\bigg|_Q \left(1+\frac{\partial V_\varepsilon}{\partial v_i}\bigg|_Q\right)^{-1} \\ \mathrm{HD}_3 &= \frac{b_3}{b_1} = \frac{1}{4}A^2\frac{a_3}{a_1} = \frac{1}{24}A^2\frac{\partial^3 V_\varepsilon}{\partial v_i^3}\bigg|_Q \left(1+\frac{\partial V_\varepsilon}{\partial v_i}\bigg|_Q\right)^{-1} \end{aligned} \tag{4.63}$$

considering only the leading terms in Eq. 4.62.

A better protection against signal-dependent switching errors is a technique known as bottom-plate sampling [Greg94].Consider the circuit in Fig. 4.14, which is the S/H circuit of Fig. 4.9(b) in writing mode with a different dc reference. If some mechanism to switch transistor M_f off before M_w is turned down is provided, then harmonic distortion vanishes. The reason is that, switching off M_f before M_w isolates the bottom plate of the storage capacitor C_m. When M_w is turned off, there is no path for the charge in C_m to escape or enter, except from parasitics feedthrough. Therefore the signal-dependent feedthrough is greatly reduced. Instead, a constant pedestal is added to each sample:

$$V_\varepsilon = k_0 + k_1[V_{REF} + V_T(V_{REF} - V_{SS})] \tag{4.64}$$

but this is not signal-dependent and, therefore, the derivatives of V_ε with respect to v_i in Eq. 4.59 are null, and so are HD_2, HD_3, etc. This small pedestal error can be easily removed by correlated double-sampling techniques or using fully-differential S/H circuits.

Capacitor sizing: speed-accuracy trade-off

As it was pointed out in Sect. 4. 1, directions to enhance the speed of operation have antagonistic effects on the accuracy of the sampling. If the pass transistor is widened, what makes it more conductive, and the storage capacitor is made smaller, decreasing C_m, the minimum acquisition time, t_{acq}, decreases, but, at the same time, feedthrough error and charge injection increase, thus degrading the accuracy on the recovery of the input signal. Let us try to resolve this trade-off between speed and accuracy. On one side, the acquisition error for a sampling rate of f_s is given by:

$$V_{\varepsilon_{\text{acq}}} = -V_i e^{-1/\tau f_s} \tag{4.65}$$

where V_i is the amplitude of the sample, and $\tau = R_{\text{ON}} C_m$. Here, C_m is the storage capacitance, which is related with the area of the capacitor through the capacitance per unit area, C_m^*, via:

$$C_m = C_m^* W_{\text{cap}} L_{\text{cap}} \tag{4.66}$$

and R_{ON} is the ON resistance of the switch, which is related with the dimensions of the switch through:

$$R_{\text{ON}} = \frac{1}{\beta_n (V_{GS} - V_T - V_{DS})} \approx \frac{1}{\mu_0 C_{ox}^* \left(\frac{W_{\text{sw}}}{L_{\text{sw}}}\right)(V_{GS} - V_T)} \tag{4.67}$$

On the other side, considering that charge injection is the main contribution to the switching error, we have:

$$V_{\varepsilon_{\text{sw}}} = -\left(\frac{C_{ox}^* W_{\text{sw}} L_{\text{sw}}}{C_m}\right)(V_{GS} - V_T) \tag{4.68}$$

that is:

$$V_{\varepsilon_{\text{sw}}} = -\frac{L_{\text{sw}}^2}{\mu_0 R_{\text{ON}} C_m} = -\frac{L_{\text{sw}}^2}{\mu_0 \tau} \tag{4.69}$$

Selecting minimum L_{sw}, the absolute value of the total error for a given sampling frequency, f_s, can be expressed as a function of τ by:

$$|V_\varepsilon| = \frac{L_{\min}^2}{\mu_0 \tau} + V_i \exp\left(-\frac{1}{\tau f_s}\right) \tag{4.70}$$

For higher sampling frequencies, the second term, that corresponds to the acquisition error, dominates. For the lower frequencies, the switching error is the main contribution to the total error $|V_\varepsilon|$. Both terms in Eq. 4.70, depend on τ as well, which is the time constant of the single-pole circuit formed by the pass transistor and the capacitor. For a lower τ, what means

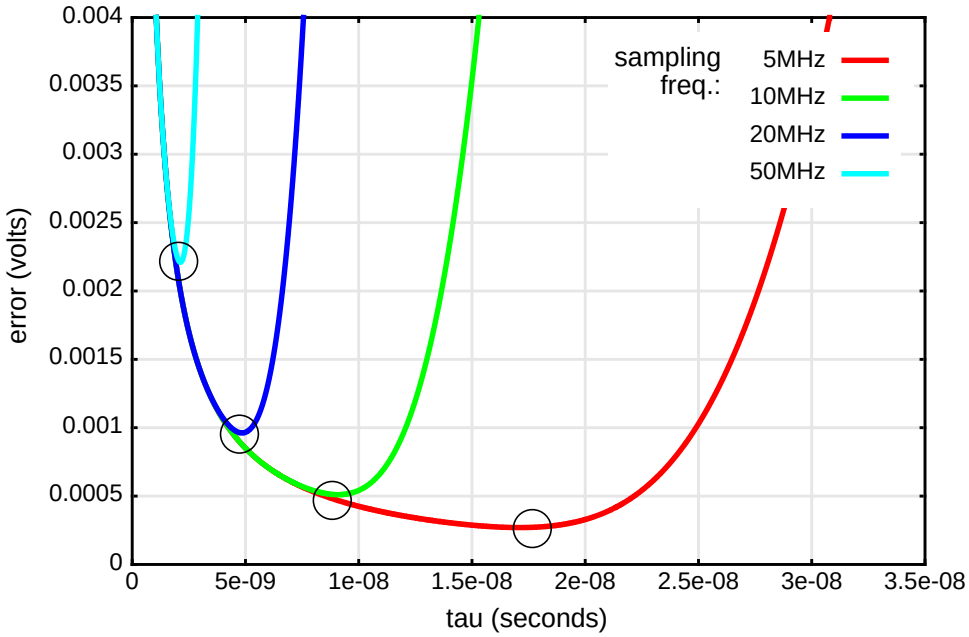


FIGURE 4.15. Speed-accuracy trade-off.

a highly conductive switch and a small capacitor, the term corresponding to the charge injection is larger. For higher τ, the contribution of the acquisition error grows saturating at V_i for an infinite time-constant. This makes sense because it means a switch with infinitely low conductivity and an infinitely large capacitor, therefore, it will take forever to track the signal. If the total error is plotted vs. τ (Fig. 4.15) it can be observed that this family of curves has a minimum for each sampling frequency. Its exact position can be obtained by deriving Eq. 4.70 and equating it to zero:

$$\frac{d|V_\varepsilon|}{d\tau} = -\frac{L_{\min}^2}{\mu_0 \tau^2}\left[1 - \frac{\mu_0 V_i}{L_{\min}^2 f_s}\exp\left(-\frac{1}{\tau f_s}\right)\right] = 0 \tag{4.71}$$

from here the value of τ for which the error is minimum, $\tau_{\min}$, is:

$$\tau_{\min} = \left[f_s \ln\left(\frac{\mu_0 V_i}{f_s L_{\min}^2}\right)\right]^{-1} \tag{4.72}$$

This equation gives us the relation between R_{ON} and C_m that minimizes the errors derived from sampling —acquisition and switching errors. In order to decide the final magnitudes of these devices, we can consider the total active area of the memory cell. This is

$$A_{maa} = W_{\mathrm{cap}} L_{\mathrm{cap}} + W_{\mathrm{sw}} L_{\mathrm{sw}} \tag{4.73}$$

that can be expressed as a function of C_m and R_{ON}, considering that the pass transistor has minimum length:

$$A_{maa} = \frac{C_m}{C_m^*} + \frac{L_{\min}^2}{\mu_0 C_{ox}^* R_{\mathrm{ON}}(V_{DD} - V_i - V_T)} \tag{4.74}$$

From here, it is seen that there is no extreme of this function. Logically, if we implement a minimum area transistor, which will be more resistive, we will need a smaller capacitor. Thus, allowing minimum size for the access switch, and keeping the relation between R_{ON} and C_m imposed by Eq. 4.72, a memory capacitor of $0.2\mathrm{pF}$ is employed.

4. 3. 2. System architecture and physical design

Floorplan of the aRAM chip

The aRAM chip proposed consists in an array of 32×256 analog memory cells, with the sensing, driving and control circuitry required to interface the memory array to either a parallel array processor chip (the CNNUM chip) or to serial analog sampled video signal sources, via the analog data bus of the CNN chipset. Each of the memory cells contains a capacitor, a pass transistor and some local logic for address decoding. The floorplan of the chip is shown in Fig. 4.16. The 8192 analog memory registers are arranged in 32 rows of 256 cells each. each row is also subdivided in smaller groups to reduce the capacitive load that have to be driven by the sensing amplifiers. Random access is featured, hence, each memory register of the array is addressed by a 5-bit row code and an 8-bit column code. With the help of 2 binary-to-one-hot custom designed decoders, one cell at a time is selected. This occurs if the serial I/O mode is selected. If parallel mode for memory updating and downloading is switched on, then 16 cells, belonging to 16 different rows are accessed in each I/O cycle. Analog channels are multiplexed with the control of the address and mode selection bits. The necessity for this I/O scheme is

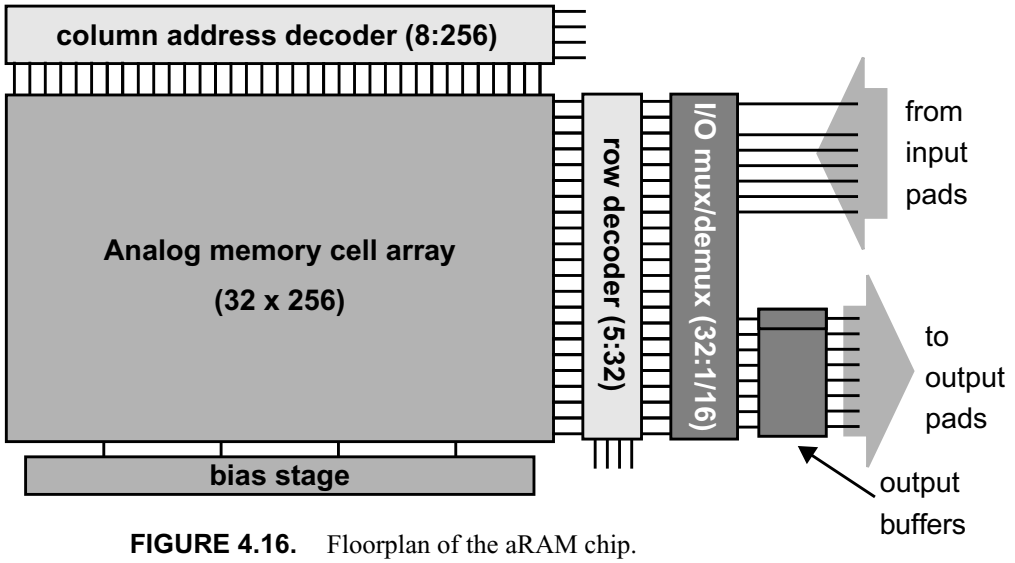


FIGURE 4.16. Floorplan of the aRAM chip.

revealed later, but its utility is evinced in Chapter 2, where the data transactions between the components of the CNN chipset are analysed.

In addition to this, buffering of the analog signals outside the memory array is provided, in order to drive the output pads. In this way, the I/O circuitry is mostly shared by the whole array, thus lowering the power consumption, which can skyrocket if such driving capabilities are ported to the individual memory registers. Also a global strobe signal, carefully transmitted to the whole array —we will see how it is done without interfering with the analog signal paths— provides tunable time-guarding for the address codes to change when writing or reading a whole sector of the memory in the same burst.

Signal interaction prevention and substrate noise decoupling

Directions concerning prevention of signal interactions and decoupling of the digital noise injected to the substrate have been followed in the development of the prototype. Metal lines driven by noisy digital signals interfere with nearby located analog lines, degrading the analog waves. On the other side, logic circuits introduce switching noise into the substrate. This circumstance can seriously affect analog blocks performance, by degrading the references and also by coupling to analog signal paths via the parasitics. Think that much of the spectral components of the logic transition currents are in the higher harmonics of the digital clocks. This converts this problem in a RF noise problem, and, at these frequencies, all parasitic capacitors become transmission paths.

Digital signal integrity becomes a problem when the magnitude of noise exceed 25% of the signal range. But for the case of analog and mixed-signal circuits, noise specifications are well under this limit. Therefore, strategies to reduce switching noise must be adopted. Because this phenomenon is distributed all along the die, it seems reasonable to apply a distributed methodology to this problem. Actions to be taken can be separated into four different categories [Twom00]: providing low-noise, low-impedance, connections for power, ground and substrate; making any listener circuit or signal less noise sensitive; suppressing or silencing the talker; and, finally, separating the talker and the listener either in space, or in time or in frequency. The best way to achieve enough reduction of the digital noise is to combine several techniques.

Let us analyse first, the signal interaction between nearby conductors, and explain the solution adopted. Consider the two parallel metal strips of Fig. 4.17(a). One of is driven by a digital source, $v_{i_D}(t)$, with source resistance R_s, and the other by an analog signal source $v_{i_A}(t)$, with the same resistance. The electric field lines departing from each of them arrive to the other, thus a picture of the vector field can be made by applying the superposition theorem. In this occasion, lumped circuits theory applies, thus these interactions can be modelled by a simple circuit. Each metal line is represented by a capacitor, C_L, and coupling between them is supported by parasitic capacitance C_p. Let us calculate the influence of

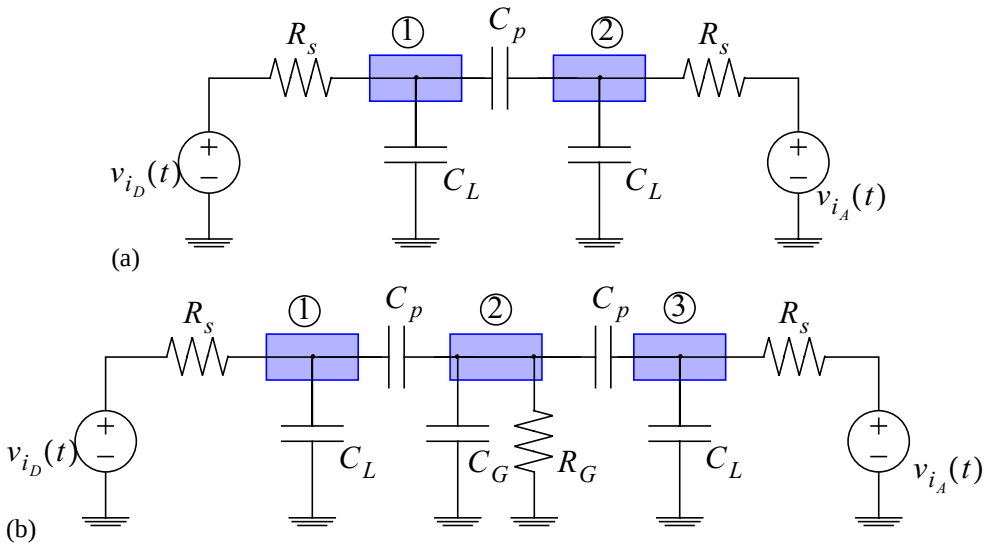


FIGURE 4.17. Profile of the parallel metal lines and schematics for the crosstalk analysis

the noisy digital signal on the neighbouring analog line. For this, let us assume that $v_{i_A}(t) = 0$. Notice that we are interested in the relation between V_2, that is the voltage at node ② and V_1, the voltage at node ①. Then, by means of node analysis, we arrive to:

$$H_{21}(s) = \frac{V_2(s)}{V_1(s)} = \frac{sR_sC_p}{1 + sR_s(C_p + C_L)} \tag{4.75}$$

which has a high-pass (HP) characteristic. It means that the higher frequency components of V_1 are passed to node ② with an approximate attenuation of $C_p/(C_p + C_L)$. Suppose that C_L is only 10 times larger than C_p. It makes barely –20dB. For instance, the response of the listener node, V_2 in this case, to a pulse of 2Vp-p applied in the talker, V_1 —what is not completely realistic, because the higher frequency components of a pulse introduced by $v_{i_D}(t)$, will be filtered out before arriving to V_1— can be seen in Fig. 4.18. This case correspond to the direct coupling of the noise source

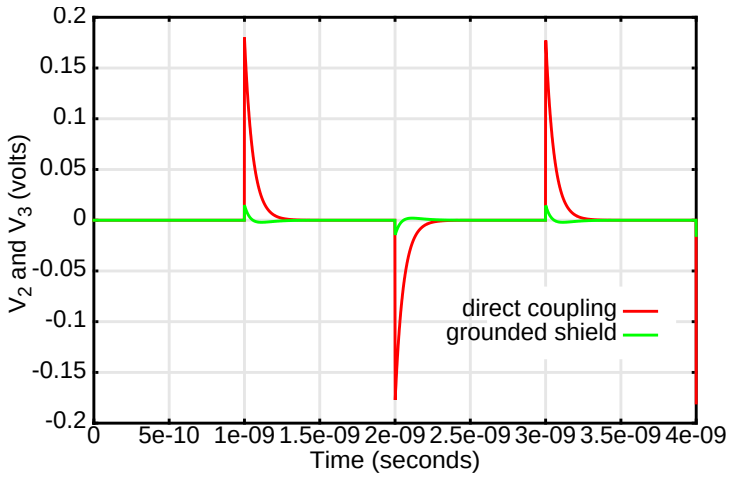


FIGURE 4.18. Response of the listener node to a pulse in the talker

and the listener through C_p. The maximum amplitude of the noise is somewhere between 150 and 200mV. For this simulation, we have chosen the following values for the elements: $R_s = 50\Omega$, $C_L = 1\text{pF}$ and $C_p = 0.1\text{pF}$.

In order to avoid this, a third metal strip connected to ground can be placed between both lines. The grounded metal line acts as a shield preventing the strongest electric field lines departing from the digital line to reach the analog metal line. In the lumped circuit model, the grounded strip has an associated capacitor C_G and a resistor R_G tied to ground (Fig. 4.17(b)). Node analysis of this circuit leads to the following transfer characteristic between node ①, and, now, node ③:

$$H_{31}(s) = \frac{V_3(s)}{V_1(s)} = \frac{s^2 R_s R_G C_p^2}{1 + s[R_s(C_p + C_L) + R_G(C_G + 2C_L)] + \ldots} \qquad (4.76)$$
$$\ldots + s^2 R_s R_G [C_p(C_p + 2C_L) + C_G(C_p + C_L)]$$

It has also a HP shape, but now is 2nd-order. This makes attenuation at lower frequencies larger. In addition, the better the contact to ground is, the better the shield performs. When $R_G \to 0$, practically $|H_{31}(j\omega)| \approx 0$. Thus making node ③ independent from ①. Notice that eliminating the shield from Eq. 4.76, thus making $R_G \to \infty$ and $C_G = 0$, we arrive back to Eq. 4.75 with half of the parasitic capacitance, as a result of connecting two C_p in series. The effect of the grounded shield can be appreciated in Fig. 4.18. Larger frequency components are now attenuated by:

$$H_{31}(s \to \infty) = \frac{C_p}{C_p + 2C_L + C_G + \dfrac{C_L G_G}{C_p}} \qquad (4.77)$$

which means, for $C_p = 0.1\text{pF}$ and $C_L = C_G = 1\text{pF}$, an attenuation of more than -40dB. Less than 20mV for an input of 2Vp-p in Fig. 4.18.

Let us now evaluate the effects of logic transients into the power, ground and substrate polarization. Consider the CMOS inverter of Fig. 4.19. Although CMOS logic does not consume any power while in steady-state, when a transient occurs between high and low states, or vice-versa, there is a small period of time in which both transistors are conducting. Thus some current is drawn from the power supply and driven to ground on each transition. At the same time, transitory currents can be observed at the bulk connections (either at the p- or n-bulk) because of the creation and destruction of the channel. These spiky currents would not be a problem if the voltage sources attached to power, ground and bulk, were ideal. The problem is that they have a *internal* resistance, R_s, which causes a voltage drop between the V_{dd} and V_{ss} sources and the actual point to which power, ground and subtrates are connected (nodes ①, ④, ② and ③, respectively). This will happen each time the inverter commutes from one logic state to the other. If only a R_s of 50Ω —an usual value for any controllable power supply found in the lab— is considered, the spikes in these voltages can be as large as 30mV for one single minimum size CMOS inverter (see the simulation outputs in Fig. 4.19). Add to this the corresponding resistances of the metal lines, metal-diffusion contacts, substrate diffusions, and the effect will propor-

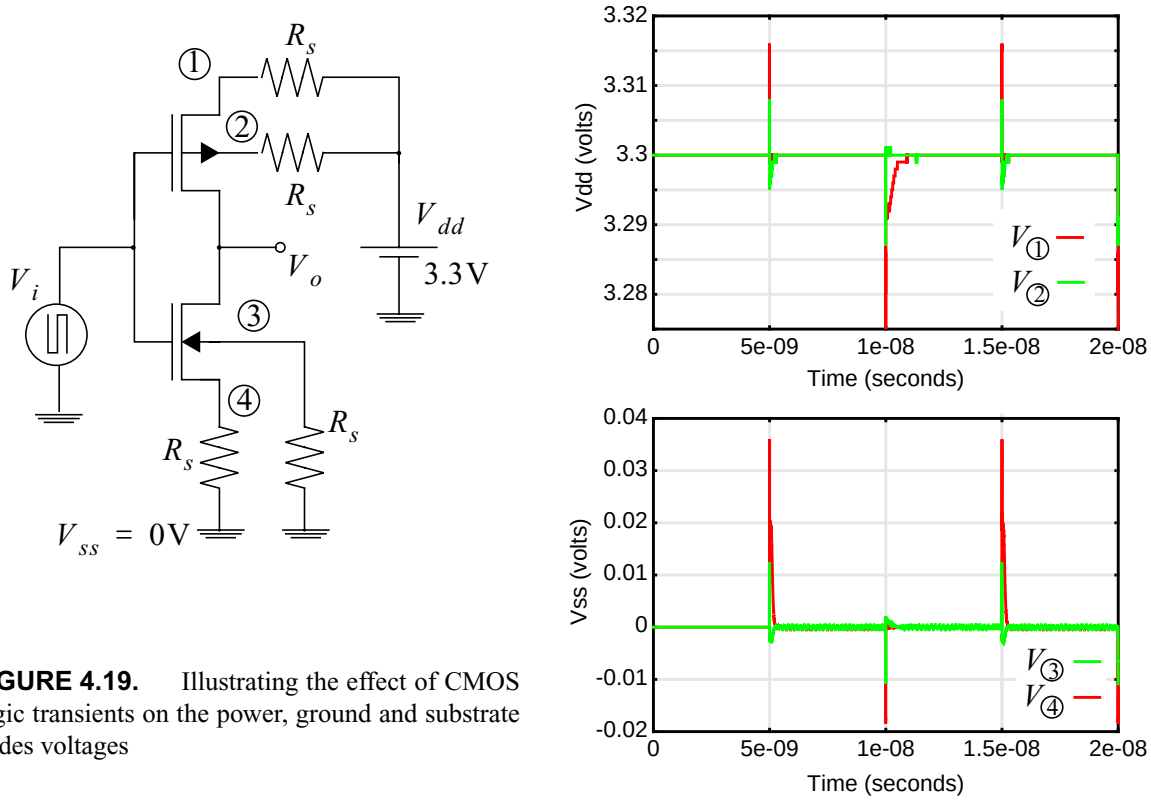


FIGURE 4.19. Illustrating the effect of CMOS logic transients on the power, ground and substrate nodes voltages

tionally grow. Now sum the contributions of all the logic circuits commuting at that precise time instant, and we will have what is commonly known as *ground bounce* or power supply noise. Actions to be taken can be found in any of the four above-mentioned categories. On one side, we can suppress or, at least, reduce the amount of switching noise. For instance, the use of balanced current-steering logic instead of a voltage-switching logic family, as CMOS static logic is, reduces considerably the current injected towards power and ground [Alls93]. Though, a high gate count renders this approach impractical since each current-steered circuit draws a constant amount of current even at steady-state. Another strategy will be a conservative layout style, with an extensive use of grounded guard rings. This reduces signal coupling by opening alternative return paths to the currents induced into the substrate [Schme94]. At the same time, it converts the problem into a local phenomenon, isolating, to a certain extent, the stronger noise sources and the more susceptible listeners. Although the physical design rules of the technology provide some directions about substrate connection to avoid latch-up [Bake98], these can be insufficient to manage with the injected currents. Thus, abundant substrate contacts are always recommended. Their effect can be reinforced by the implementation of separated power supply and ground connections for the analog and digital circuitry and guard rings [Stan95]. All these strategies have been incorporated to the design of the aRAM chip prototype.

I/O Scheme

As it was mentioned earlier, serialization and deserialization of the sampled analog data can be realize on-chip. For this purpose, a serial/parallel mode analog multiplexer has been placed beside the memory array. In this way, the I/O

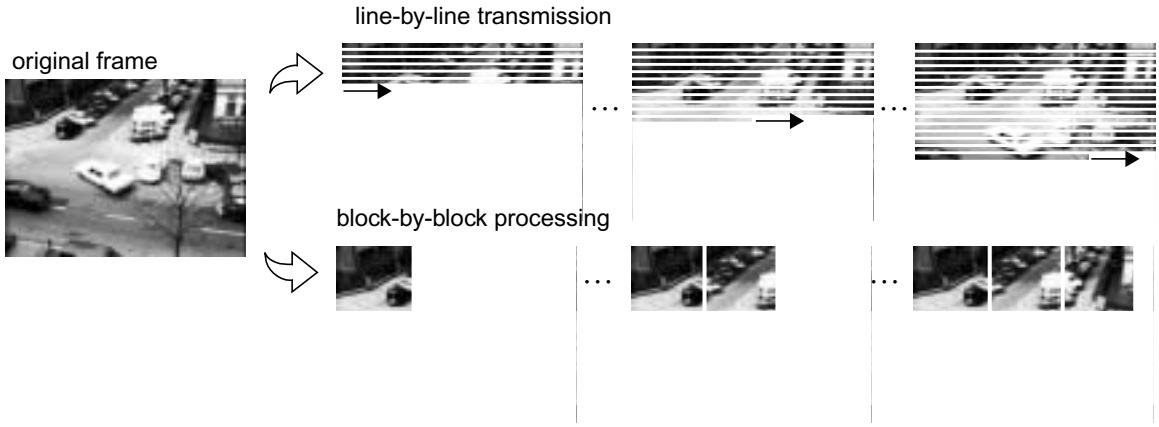


FIGURE 4.20. Video frame partitioning for transmission and processing.

interface scans the 32 memory rows in a 1-by-1 fashion, when serial mode is selected, or in a 16-by-16 basis. This multimodal I/O scheme enhances versatility of the aRAM for different architectures of the CNN chipset. The main reason to include this I/O plan can be found in the discrepancy between the way in which video signals are usually transmitted and the way in which processing based in local image properties is realized. As displayed in Fig. 4.20, transmission of video signals occurs on a line-by-line basis in the most spread standards (see Chapter 1) while any kind of 2-dimensional processing of the image, not necessarily based on CNNs, requires a block-by-block scheme for covering the original frame [Gero99]. This if fully-parallel processing is not available. Therefore, the inclusion of a versatile I/O layer for the aRAM chip prototype permits, on one side, to easily build a video signal interface for the CNN chipset (Fig. 4.21), and on the other side, to experiment with a CMOS compatible analog interface of the CNN processing core for a future integration of both functionalities on the same die. From the experimental results we will see that the aRAM implementation proposed features very promising capabilities to be included in a larger and more complex system-on-a-chip, especially when compared to more conventional implementations.

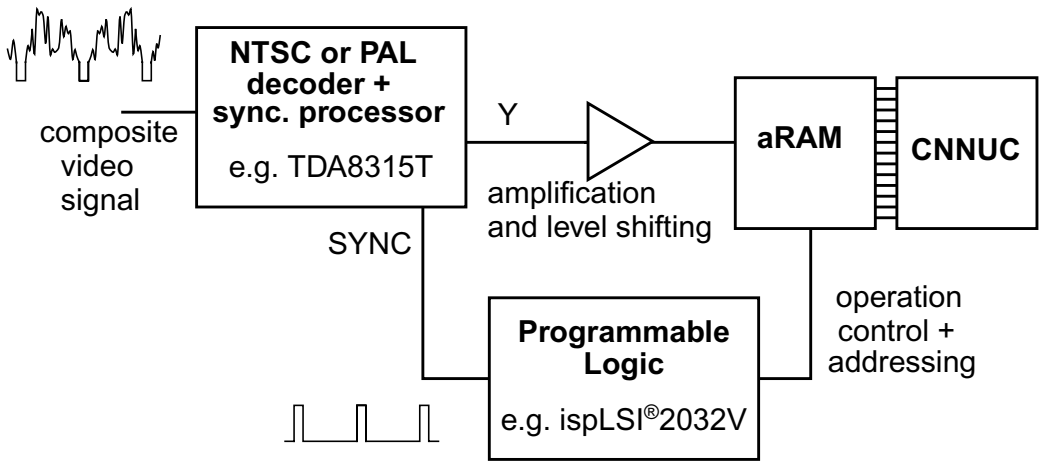


FIGURE 4.21. Composite-video signal interface to the CNN chipset.

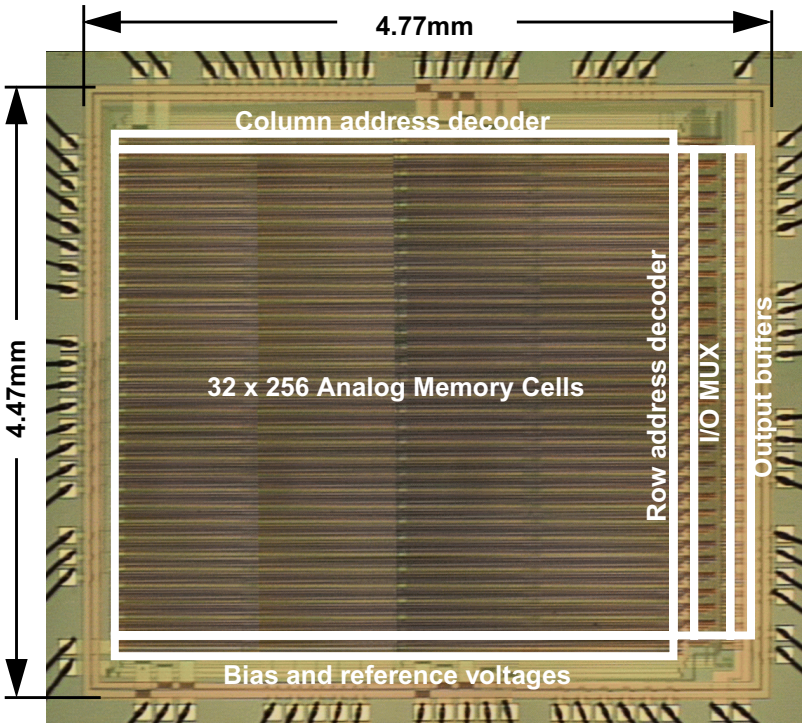


FIGURE 4.22. Photograph of the aRAM prototype chip.

4. 3. 3. Experimental results

Prototype chip data

The prototype chip, AM8K, described in the previous sections has been designed and fabricated in a $0.5\mu m$ CMOS single-poly triple-metal technology. It contains 32×256 analog memory cells that occupy $12.86 mm^2$. This means a cell-density of $637 cells/mm^2$. The total chip area, what includes the I/O mux-demux, output buffers and the pad ring, reaches $21.32 mm^2$. Fig. 4.22 shows photograph of the prototype in which the different blocks of the chip architecture have been highlighted. In Fig. 4.23, the bonding diagram of the prototype and the pin map of the packaged chip are displayed. It has been mounted in a ceramic PGA-84 package, but only 81 pins are used. A total of 24 samples have been tested and proven to be functional. No major discrepancies have been found during the test of the different samples. From the measurements realized in the laboratory, which will be reviewed in detail later, an equivalent resolution of 7-8bits has been obtained. Storage times in the 80-100ms range have been measured at room temperature and under normal operation conditions, but no exact measure of the chip temperature is available. DC power dissipation remains $73 mW$ for a $3.3 V$ power supply. Access times of $100 ns$ have been obtained for the prescribed accuracy. These data will be summarized in Table 4.2, at the end of this chapter, in which a comparison between different aRAMs CMOS technology is portrayed.

Before entering in the test setup and measurements, here are some indi-

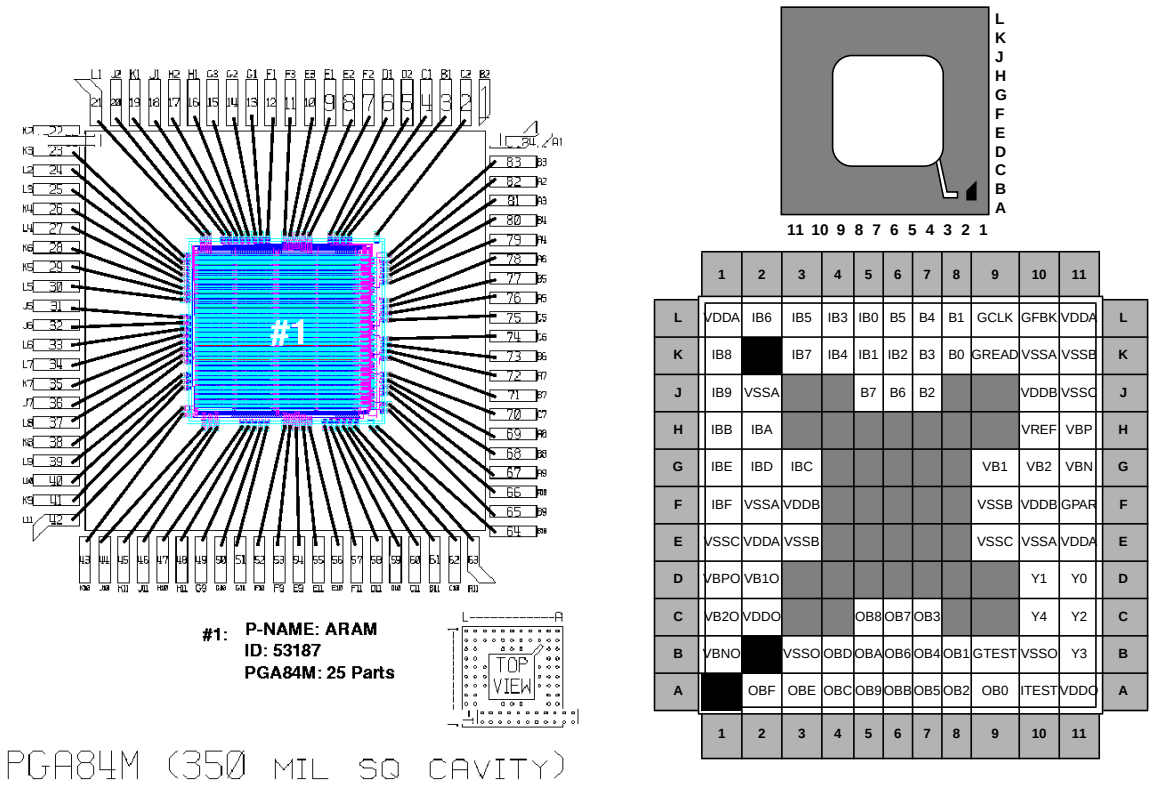


FIGURE 4.23. Bonding diagram and pin map of the packaged prototype chip.

cations about the chip connections and the signals required for the operation of the prototype. The following table describes the pin assignment, the prototype pad names and the nature of the signals that have to be attached to them:

Finger #	Pin #	Pad name	Description
1	B2		Not Connected
2	C2	VDDO#1	Output stage **power supply**: dc power source of 3.3V. External capacitor recommended for stabilization
3 4 5 6	B1 C1 D2 D1	VBNO VB2O VB1O VBPO	**Bias voltages** of the output buffers: Cascode references. Nominally they must be set to 0.98, 1.60, 1.70 and 1.77 volts, respectively.
7 8	F2 E2	VSSA#1 VDDA#1	Analog **ground connection** and **power supply**: dc 0.0V and 3.3V respectively. External capacitor recommended.
9 10 11	E1 E3 F3	VSSC#1 VSSB#1 VDDB#1	Digital **ground connections** and **power supply**. They must be connected to 0.0V, 0.0V and 3.3V respectively. External capacitors recommended.
12 13 14 15	F1 G1 G2 G3	IBF IBE IBD IBC	**Analog input bus**: input signals must be limited to a bandwidth of 3.83MHz (NTSC composite video compatible) and in the range of 0.8-2.0 volts.

TABLE 4.1. Pin assignment of the AM8K prototype and signals description

Finger #	Pin #	Pad name	Description
16 17 18 19	H1 H2 J1 K1	IBB IBA IB9 IB8	**Analog input bus**: input signals must be limited to a bandwidth of 3.83MHz (NTSC composite video compatible) and in the range of 0.8-2.0 volts.
20 21	J2 L1	VSSA#3 VDDA#3	Analog **ground connection** and **power supply**: dc 0.0V and 3.3V respectively. External capacitor recommended.
22	K2		Not Connected
23 24 25 26 27 28 29 30	K3 L2 L3 K4 L4 K6 K5 L5	IB7 IB6 IB5 IB4 IB3 IB2 IB1 IB0	**Analog input bus**: input signals must be limited to a bandwidth of 3.83MHz (NTSC composite video compatible) and in the range of 0.8-2.0 volts.
31 32 33 34 35 36 37 38	J5 J6 L6 L7 K7 J7 L8 K8	B7 B6 B5 B4 B3 B2 B1 B0	**Column address code**: Each pair— B7B6, B5B4, B3B2, B1B0— is Gray-coded: “00”-”01”-”11”-”10”.
39	L9	GCLK	**Global clock**: sets the duty cycle of the selection pulse width, avoiding any overlapping.
40	L10	GFBK	**Global feedback signal**: controls the sample and hold feedback loop for bottom-plate sampling. It must fall some time before the GCLK falling-edge.
41	K9	GREAD	**Global read signal**: “0” for test and memory updating, “1” for output downloading.
42 43	L11 K10	VDDA#4 VSSA#4	Analog **power supply** and **ground connection**: 3.3V and 0.0V respectively. External capacitors recommended.
44 45 46	J10 K11 J11	VDDB#3 VSSB#3 VSSC#3	Digital **power supply** and **ground connections**. They must be connected to 3.3V, 0.0V and 0.0V respectively. External capacitors recommended for stabilization.
47	H10	VREF	**S&H virtual ground reference**. It must be set to 0.8V.
48 49 50 51	H11 G9 G10 G11	VBP VB1 VB2 VBN	**Bias voltages** of the local OPAMPs. Nominally they must be set to: 2.00, 1.70, 1.60, 0.88, respectively.
52 53 54	F10 F9 E9	VDDB#2 VSSB#2 VSSC#2	Digital **power supply** and **ground connections**. They must be connected to 3.3V, 0.0V and 0.0V respectively. External capacitor recommended for stabilization.
55 56	E11 E10	VDDA#2 VSSA#2	Analog **power supply** and **ground connection**: 3.3V and 0.0V dc respectively. External capacitor recommended.
57	F11	GPAR	**Serial/Parallel mode selector**: “0”:Serial, “1”: Parallel

TABLE 4.1. Pin assignment of the AM8K prototype and signals description

Finger #	Pin #	Pad name	Description
58	D11	Y0	Row address code: 5b Gray-code
59	D10	Y1	
60	C11	Y2	
61	B11	Y3	
62	C10	Y4	
63	A11	VDDO#2	Output buffer **power supply** and **ground connection**: 3.3V
64	B10	VSSO#2	and 0.0V respectively. External capacitor recommended.
65	B9	GTEST	**Output buffer test enable**: “0”:Normal operation, “1”:Test.
66	A10	ITEST	**Output buffer test input**: analog signal to test the buffers.
67	A9	OB0	**Analog output bus**: Output buffers are designed to load a
68	B8	OB1	capacitance of 20pF maximum, including pads and package
69	A8	OB2	capacitances.
70	C7	OB3	
71	B7	OB4	
72	A7	OB5	
73	B6	OB6	
74	C6	OB7	
75	C5	OB8	
76	A5	OB9	
77	B5	OBA	
78	A6	OBB	
79	A4	OBC	
80	B4	OBD	
81	A3	OBE	
82	A2	OBF	
83	B3	VSSO#1	Output buffer **ground connection**: 0.0V.
84	A1		Not Connected

TABLE 4.1. Pin assignment of the AM8K prototype and signals description

And these (Figs. 4.24 and 4.25) are the timing diagrams for serial and parallel image acquisition and downloading.

Test setup and software

The laboratory setup employed to test the chip prototype is depicted in Fig. 4.26. We have used the HP82000 digital IC evaluation system to realize, on one side, the digital control of the device under test (DUT) mounted in its PCB, and on the other, to establish the synchronicity between the instruments. The analog input is originated with the help of a HP33210 arbitrary function generator. This instrument receives triggering from the HP82000. To sense the output a HP54615B digitizing oscilloscope is employed, also triggered by a synchronicity signal generated at the HP82000. Therefore, the function generator, the HP82000, and so the DUT, and the oscilloscope are hard-wired to communicate real-time. On the contrary, global control of the tests is performed by a computer, and instructions to the instruments are delivered via the general purpose instrument bus (GP-IB or HP-IB in its commercial form). The test control programs that run in the workstation have been programmed with the help of Agilent’s Visual Engineering Environment (VEE, formerly HP-VEE).

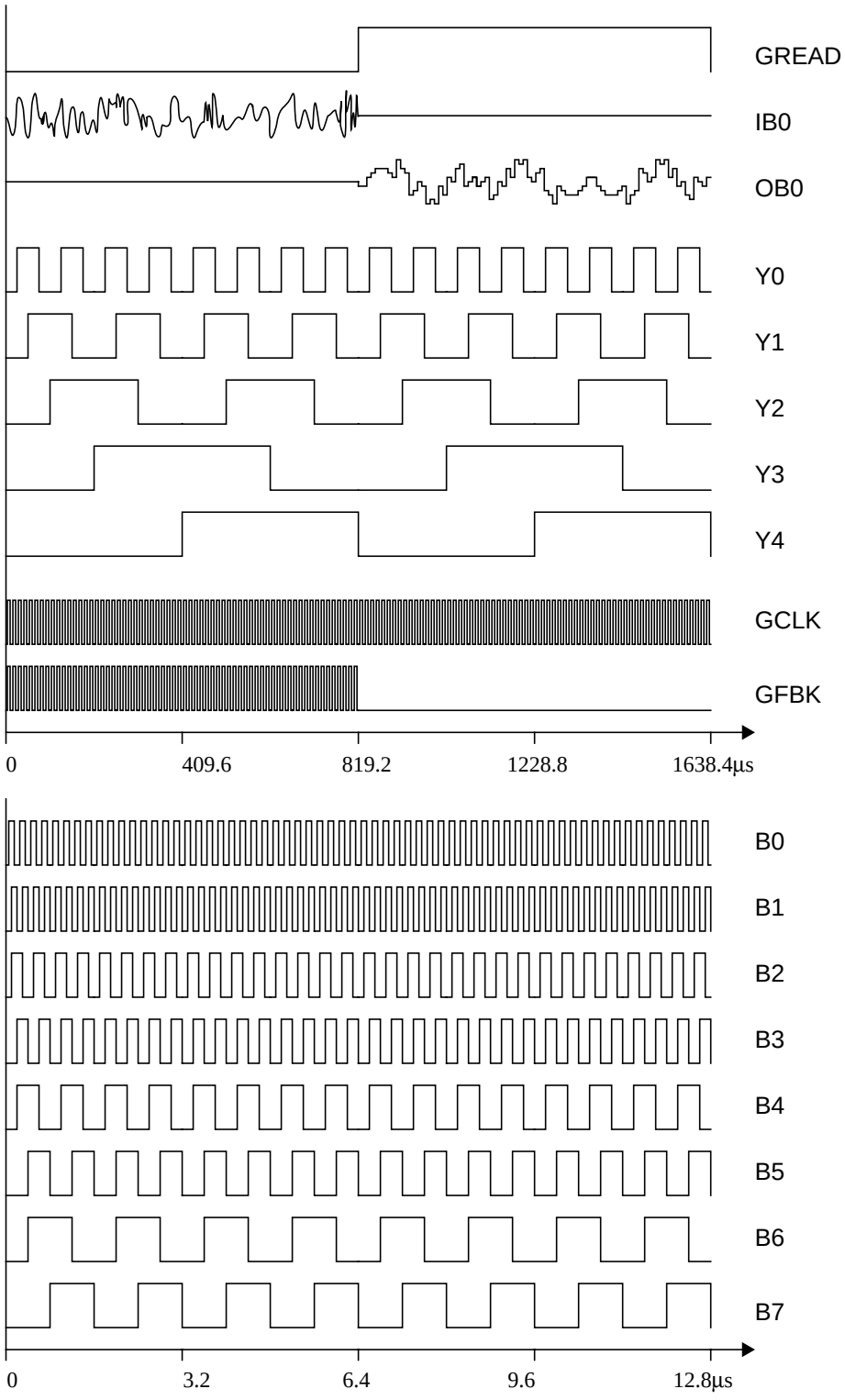


FIGURE 4.24. Timing diagram for serial image acquisition and delivery (GPAR=0).

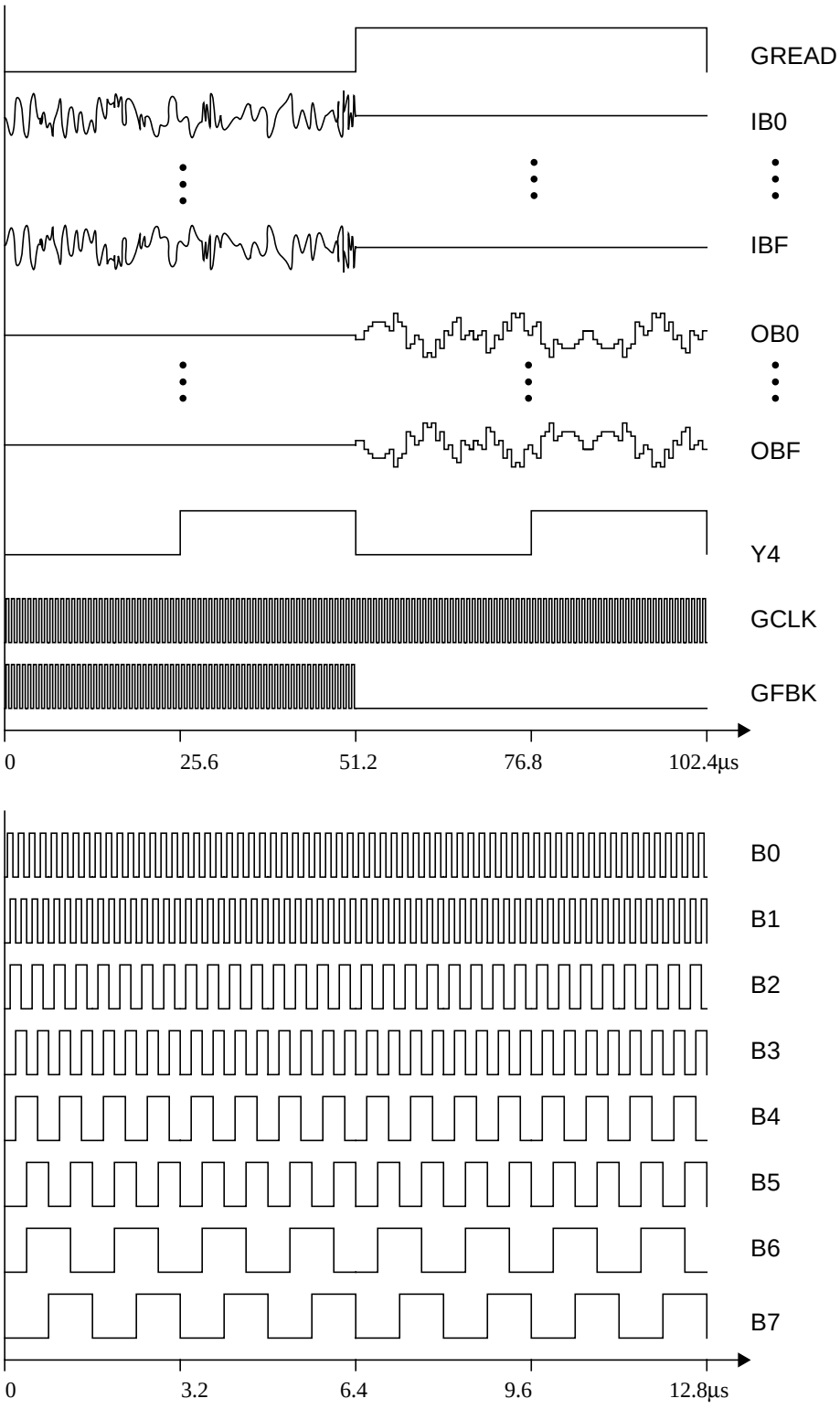


FIGURE 4.25. Timing diagram for parallel image acquisition and delivery (GPAR=1 and Y3Y2Y1Y0=0000).

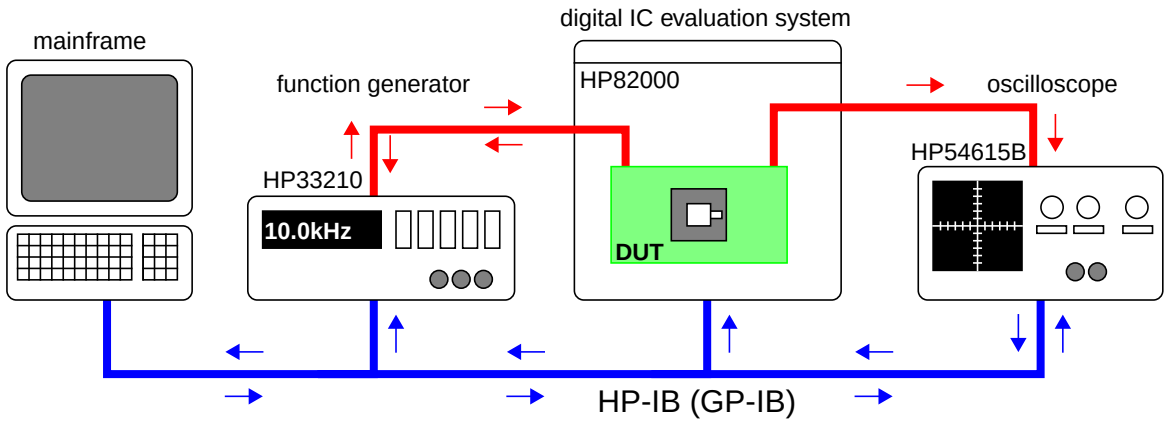


FIGURE 4.26. Test setup and data links between instruments.

These test programs have mostly the same structure: a first section for declaration and assignment of global variables (power voltages, sampling frequency, data file names, etc.). Then internal and auxiliary variables are calculated. After that the instruments are reset and configured for the new data sequentially. Once everything is set, all the instruments are commanded to wait for the master instrument to trigger the test run (in this occasion the HP82000 is the master) with a synchronicity signal. Once the system operation finish, it all happens in a burst, the data in the memory of the digitizing oscilloscope are retrieved. Depending on the type of test we are performing: access times, storage time, accuracy, etc., the corresponding data processing is done and the outputs are visualized. Fig. 4.27 displays one of the visual programs for the test. Comments are placed beside in order to identify the above-mentioned sections. Each of the blocks is decomposed into several boxes performing operations in a lower hierarchy level. We have unfolded one of these subroutines in Fig. 4.28, actually that realizing data retrieval from the oscilloscope memory. Also in Fig. 4.28, the control panel of the program is shown.

Equivalent resolution and linearity

A 1Vp-p single-tone input at 10kHz has been employed to test the accuracy of the aRAM in reproducing the stored signals. The input wave has been acquired at 10Ms/s, what means 100ns access time, and it has been downloaded at the same speed, 10Ms/s, or 100ns off-chip reading time. Fig. 4.29 displays a plot of the input and output waves put together in the same time basis. The output has been smoothed by stripping higher frequency components in the data processing block. The differences between them are plotted in Fig. 4.30. An average offset of –33.9mV has been found, which corresponds mainly to the constant pedestal added to each sample, as can be deducted from the differences computed after removal of the offset (Fig. 4.31). The root-mean-square error (RMSE) of the difference between each sample of the input and the corresponding sample of the output:

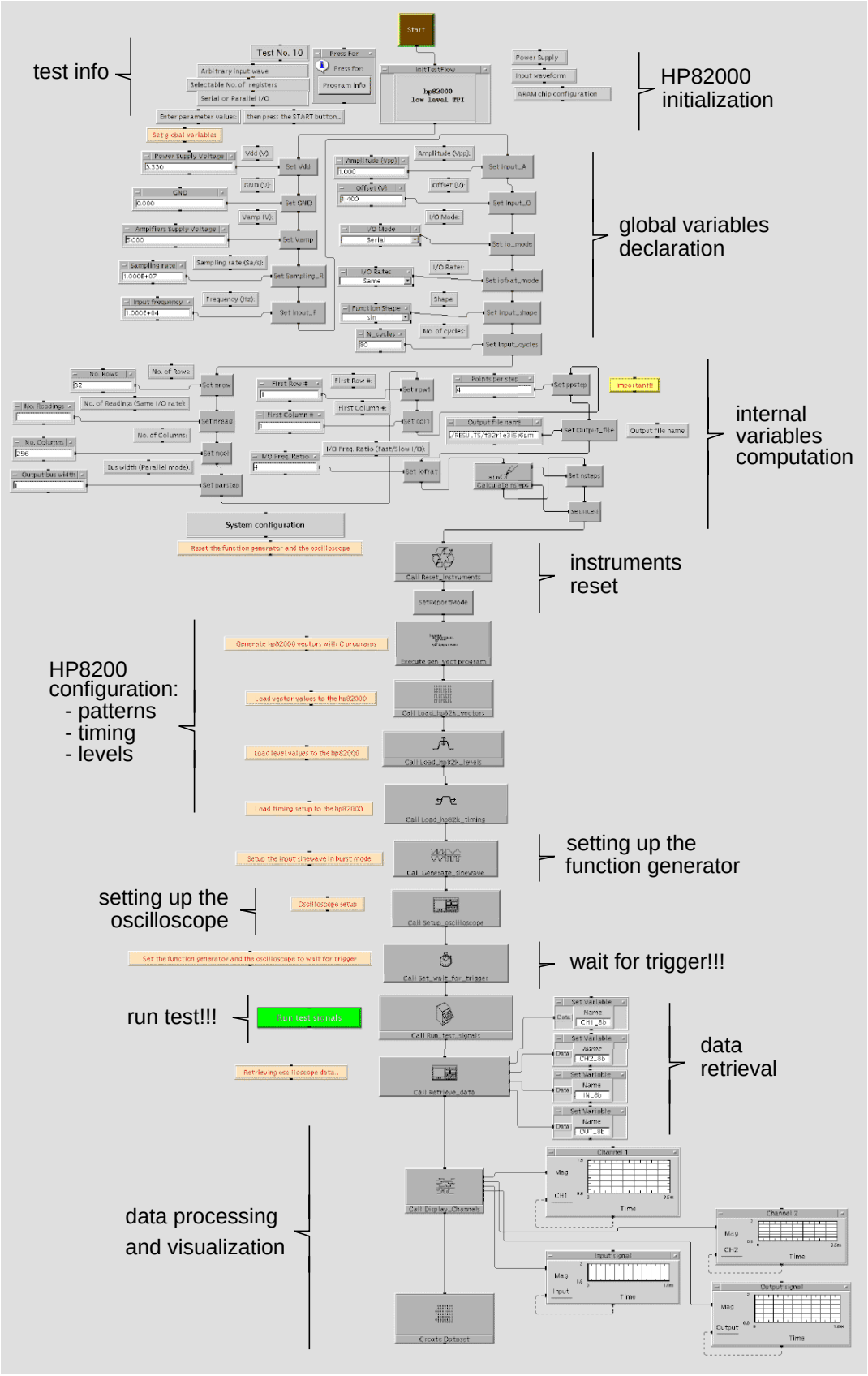
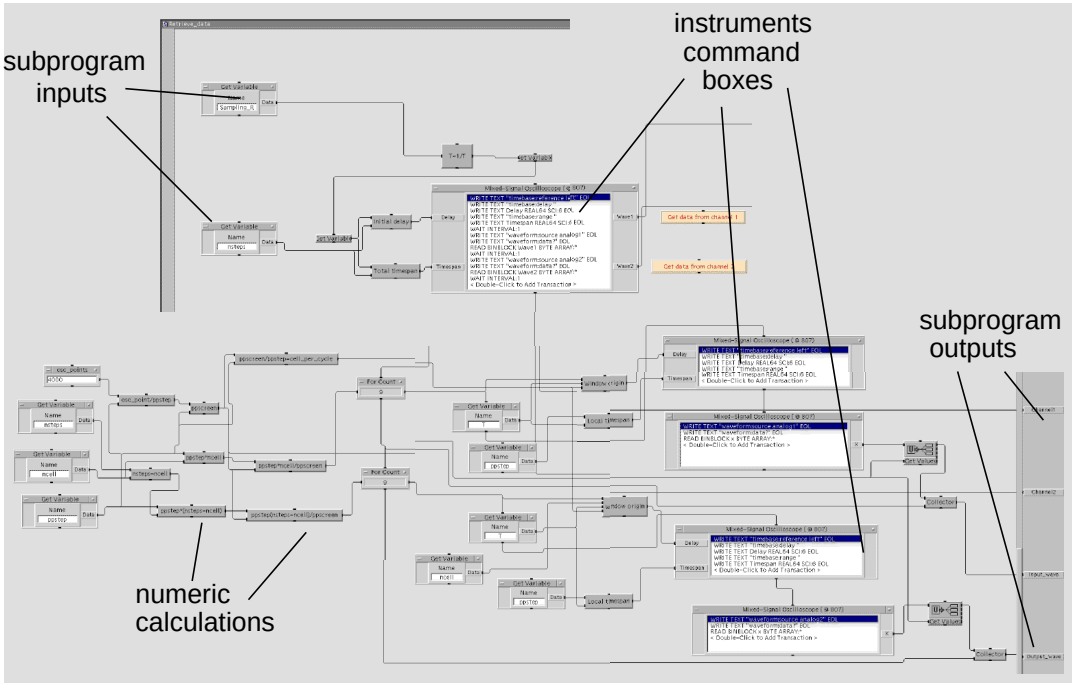


FIGURE 4.27. Visual description of one of the HP-VEE test programs.



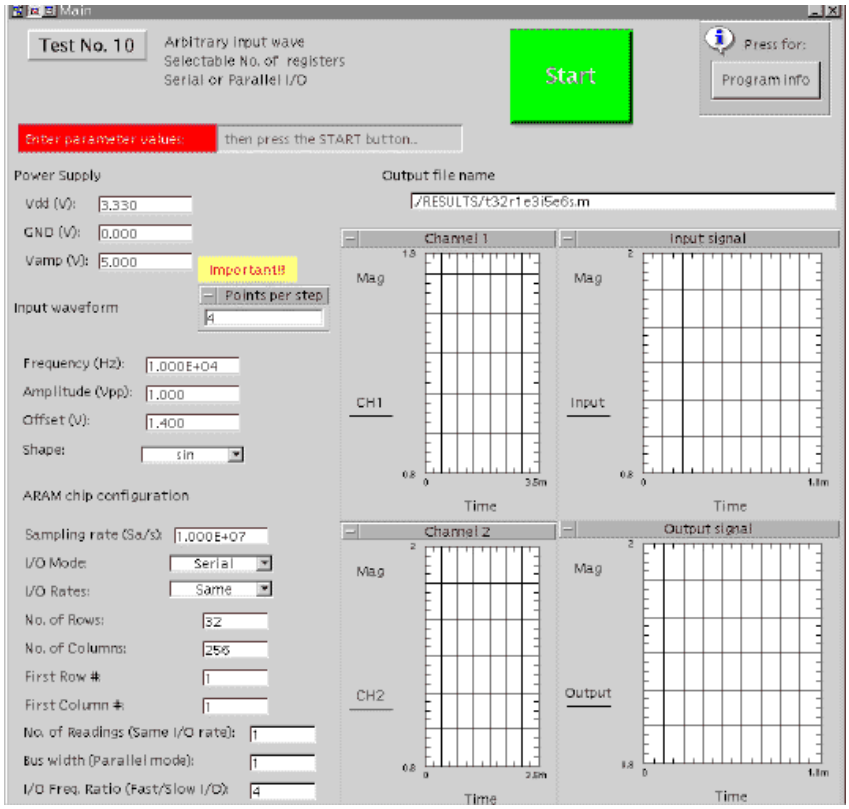


FIGURE 4.28. Detail of one of the subroutines and control panel of the test program

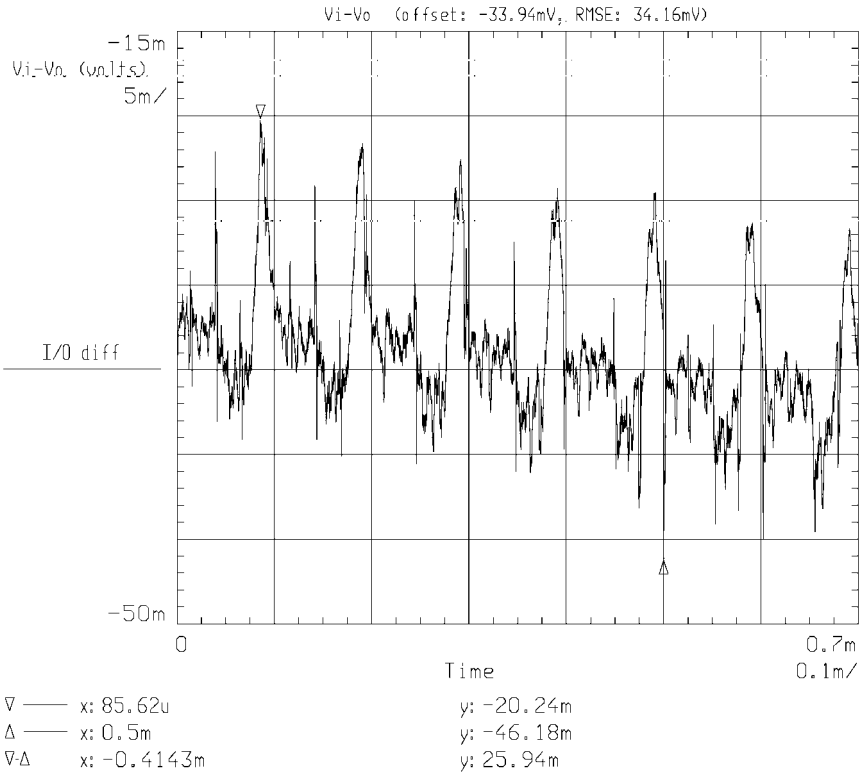


FIGURE 4.30. Differences between input and output

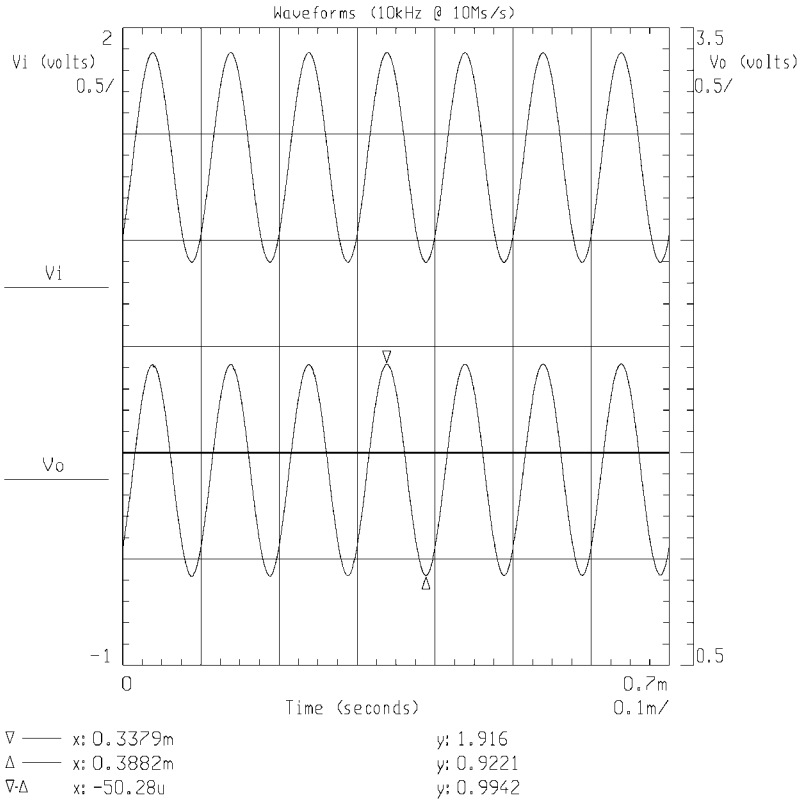


FIGURE 4.29. Input and smoothed output waves

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(V_{i_k} - V_{o_k})^2} \tag{4.78}$$

obtained for these waveforms is of 3.9mV, what means an equivalent resolution of 8bits.

In order to quantify the efficiency of the proposed sampling method in eliminating distortion, the magnitude spectrum of the output waveform have been computed from the retrieved samples with the help of the test software (Fig. 4.32). As can be appreciated, there is a difference of 46.8dB between the magnitude of the tonal impulse at 10kHz (normalized here to unity) and the magnitude of the second harmonic.

More results to illustrate the linearity of the I/O map can be found by computing its differential non-linearity (DNL). For this, the 1Vp-p amplitude of the input is divided into 256 levels and hence the voltage corresponding to 1LSB is computed, let us call it $V_{1\text{LSB}}$. Assume that an input voltage represented by decimal number n, this is $V_{i_n} = nV_{1\text{LSB}}$, has a corresponding output V_{o_n}. Then, the DNL for this pair is:

$$\text{DNL}_n = (V_{o_n}/V_{1\text{LSB}}) - n \tag{4.79}$$

This measure has been represented in Fig. 4.33 and it can be seen that it does not exceed $(1/2)\text{LSB}$ for any of the 256 levels.

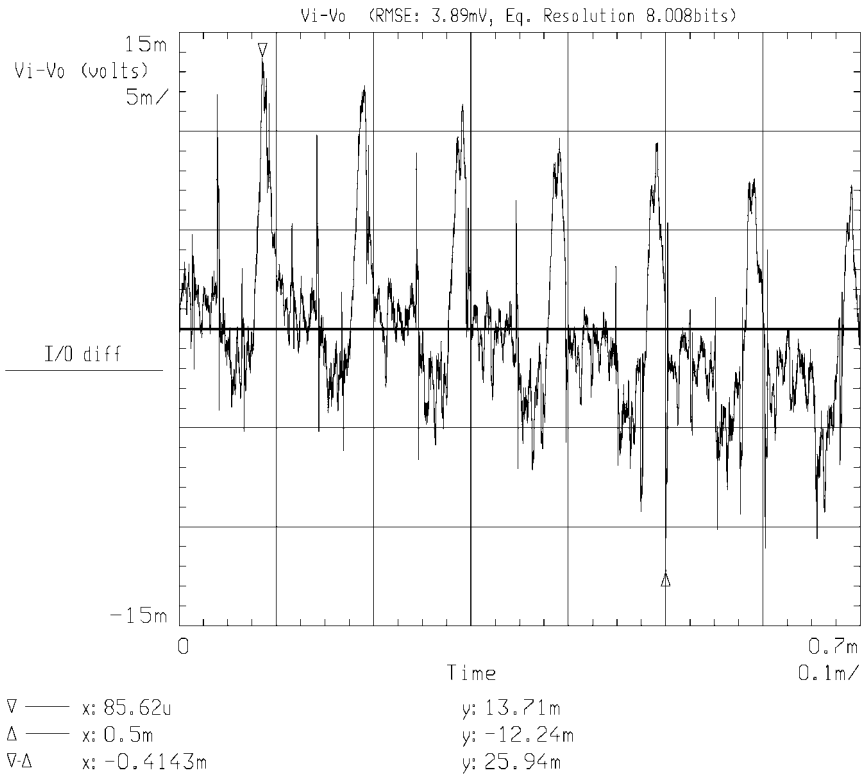


FIGURE 4.31. I-O Differences (offset removed).

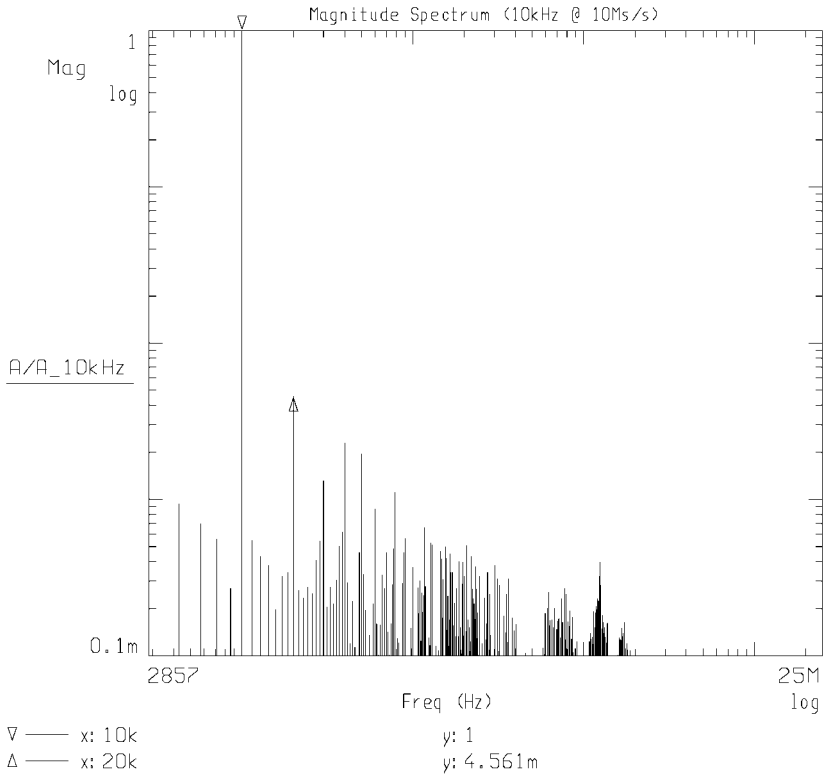


FIGURE 4.32. Magnitude spectrum of the output wave

Also with the object of measuring linearity and distortion, integral non-linearity (INL) can be computed. For the calculation of the INL, the best fitting straight line has to be computed by linear regression out of all the obtained I/O pairs. Once we have the best fitting line, which has a slope of 1.0047, a correlation coefficient of 0.9999 and an offset of 8.17LSB, as can be seen in Fig. 4.34, then the INL for each sample is calculated by substracting the corresponding point in the line from the actual output. The results are shown in Fig. 4.35. It can be seen that it is in the extremes where the actual output deviates more from the linear behaviour.

Storage time

An estimation of the storage time has been obtained by measuring the voltage degradation in time in different cells belonging to different chip samples. It has been measured at room temperature and no method for temperature reading nor control was available. Therefore, let us consider this measurements as orientative. Then, the voltage degradation in 24 memory cells, each one belonging to a one of the 24 different samples of the chip, has been measured. It has been done by selecting the analog memory cell, updating it with an initial voltage value and then taking successive readings of the output during a certain period of time.

As it was foreseen in Sect. 4. 1. 2, leakages in the poly-to-diffusion structure leads to a self-discharge rate for the capacitor. This linear degradation of the stored voltage has been confirmed by the experimental

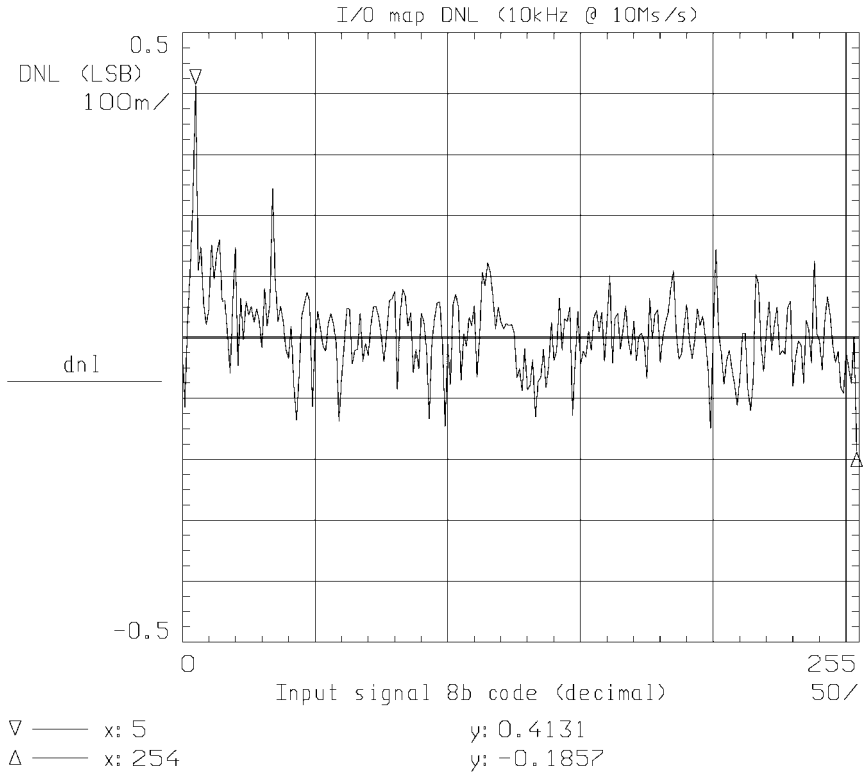


FIGURE 4.33. Differential non-linearity (DNL) of the I/O map

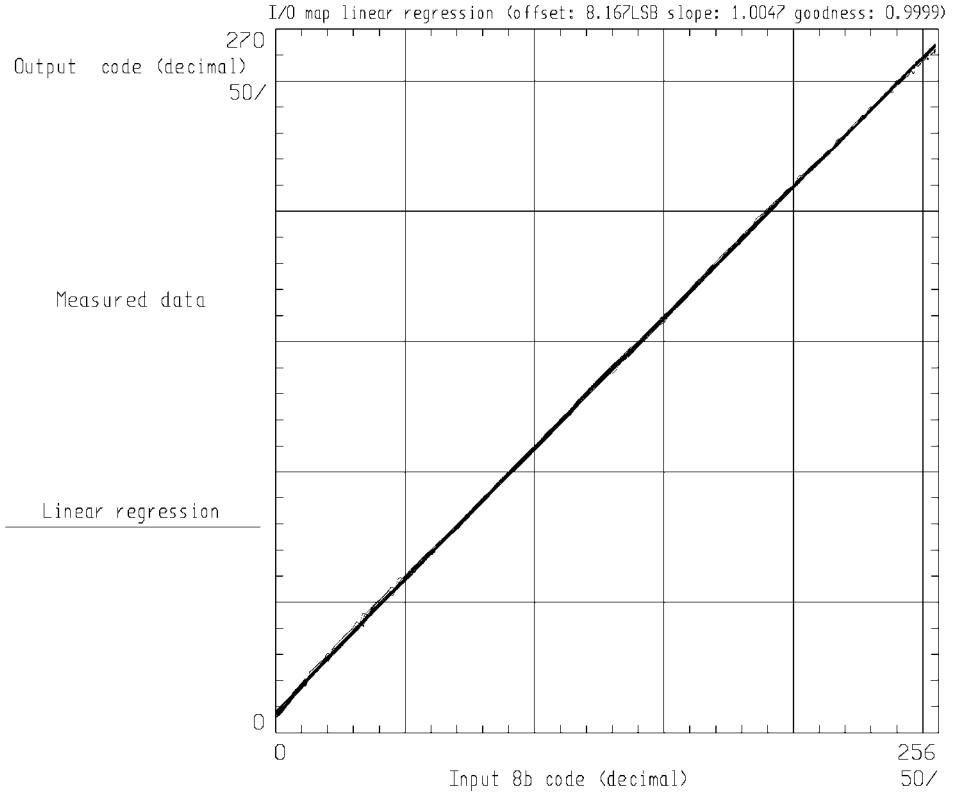


FIGURE 4.34. I-O linear regression

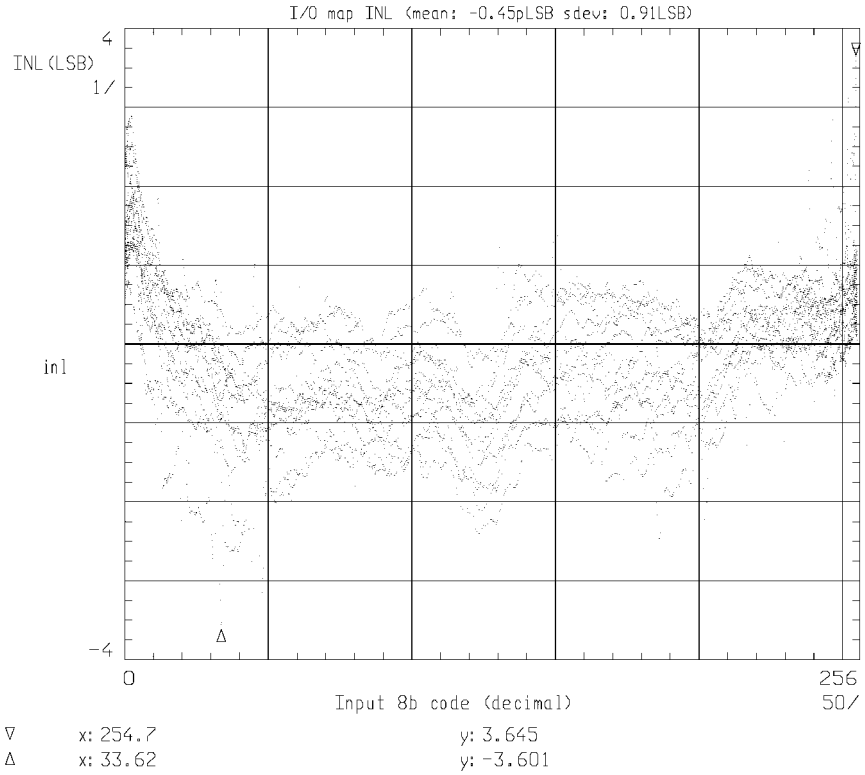


FIGURE 4.35. Integral non-linearity (INL)

observations. Fig. 4.36 represents the difference between the initially stored value and the actual voltage at each specific time instant. The errors start with a 9mV average at $t = 0$, and thus we will discount this initial error from the storage time estimation. As the full-scale signal range in this experiment was estimated to be 1.4V, the degradation of the input voltage has to be compared with this range. Then for a resolution of $7\text{-}8\text{bits}$, what means a voltage degradation of $5\text{-}11\text{mV}$ in this context, the information can be retained as much as $60\text{-}80\text{ms}$. For $6\text{-}7\text{bits}$, an error in the $11\text{-}22\text{mV}$ range, storage times of $80\text{-}100\text{ns}$ can be reached. These figures show that for the most usual video signal formats, with frame rates above 60Hz, this is 16.67ms per picture, this aRAM chip ca n perfectly maintain the accuracy levels during the frame processing.

In addition to this, no significant discrepancies in the readings have been detected when repeatedly accessing the same memory location.

Tests with real images

Besides electrical characterization of the aRAM prototype, some real images have been stored and downloaded from the chip. These images have been obtained by Dr. Péter Földesy at the Analogic and Neural Computing Laboratory of the Computer and Automation Institute of the Hungarian Academy of Science, in Budapest. They have been acquired with the help of the chip prototyping system developed in their group [Rosk99]. In the first place, a 32×128 -pixels image, digitized originally in different number of quantization levels, is portrayed in Fig. 4.37. Here,

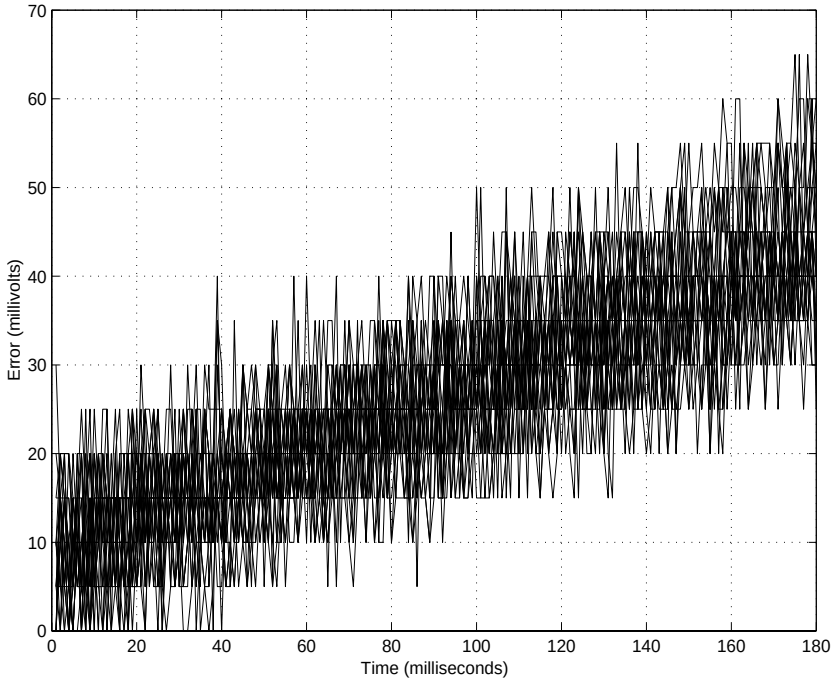


FIGURE 4.36. Degradation of the stored voltage (24 cells)

images (a), (c) and (e) are the inputs and (b), (d) and (f) are the recovered signals, these in an 8bit-coded grayscale. Qualitatively, the most of the image information has been acquired, stored and downloaded without significant losses. In the following illustration (Fig. 4.38) several images are displayed. The first two correspond to 512×512-pixels black and white pictures in a 256-level grayscale. The last is a 256×256-pixels colour picture. All of them have been cut into 32×128-pixels pieces because of limitations of the test equipment. Here, the first row, (a), represents the original inputs, row (b) stand for the output images and, in (c), the absolute difference between them has been represented in the same grayscale. The coloured sample has been separated into three images corresponding to red, green and blue information and processed separately. After that they have been fused again to compose the output. As can be seen from the pictures, some spatial noise have been detected in the outputs. It is partly due to image partitioning and on the other side, as we have detected, due to an improper tracking of the input at the beginning of each pixel group —the vertical lines at the 1st, 129th, 257th and 385th columns of pixels. Because of the strobing scheme adopted to avoid the selection of more than one memory cell at a time, the feedback loop of the opamp in the S/H stage is left open for some time after the last writing. As a consequence, the voltage of the output node can go up to the power supply voltage or down to the negative rail. And from there, the slew-rate of the opamp is insufficient to catch up with the input in the required acquisition time. This problem can be overcome by skipping this first reading, or, by letting the acquisition time of the first reading be larger.

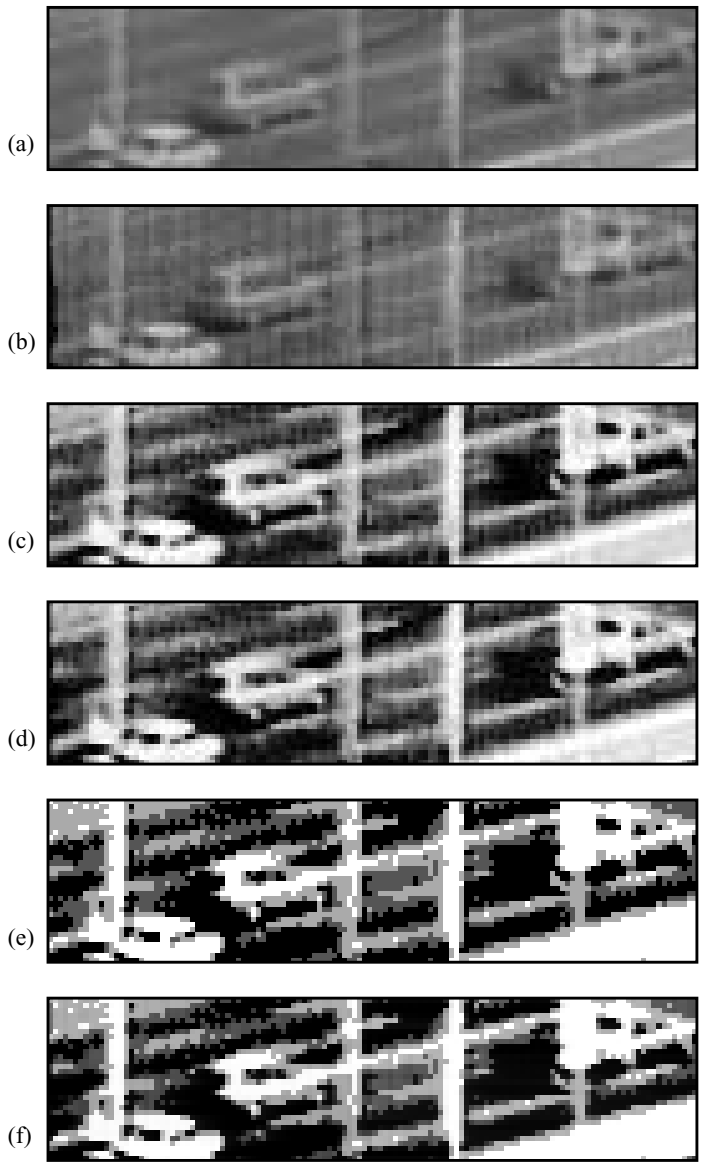


FIGURE 4.37. Real images test samples. (a), (c) and (e) inputs, and (b), (d) and (f) outputs.

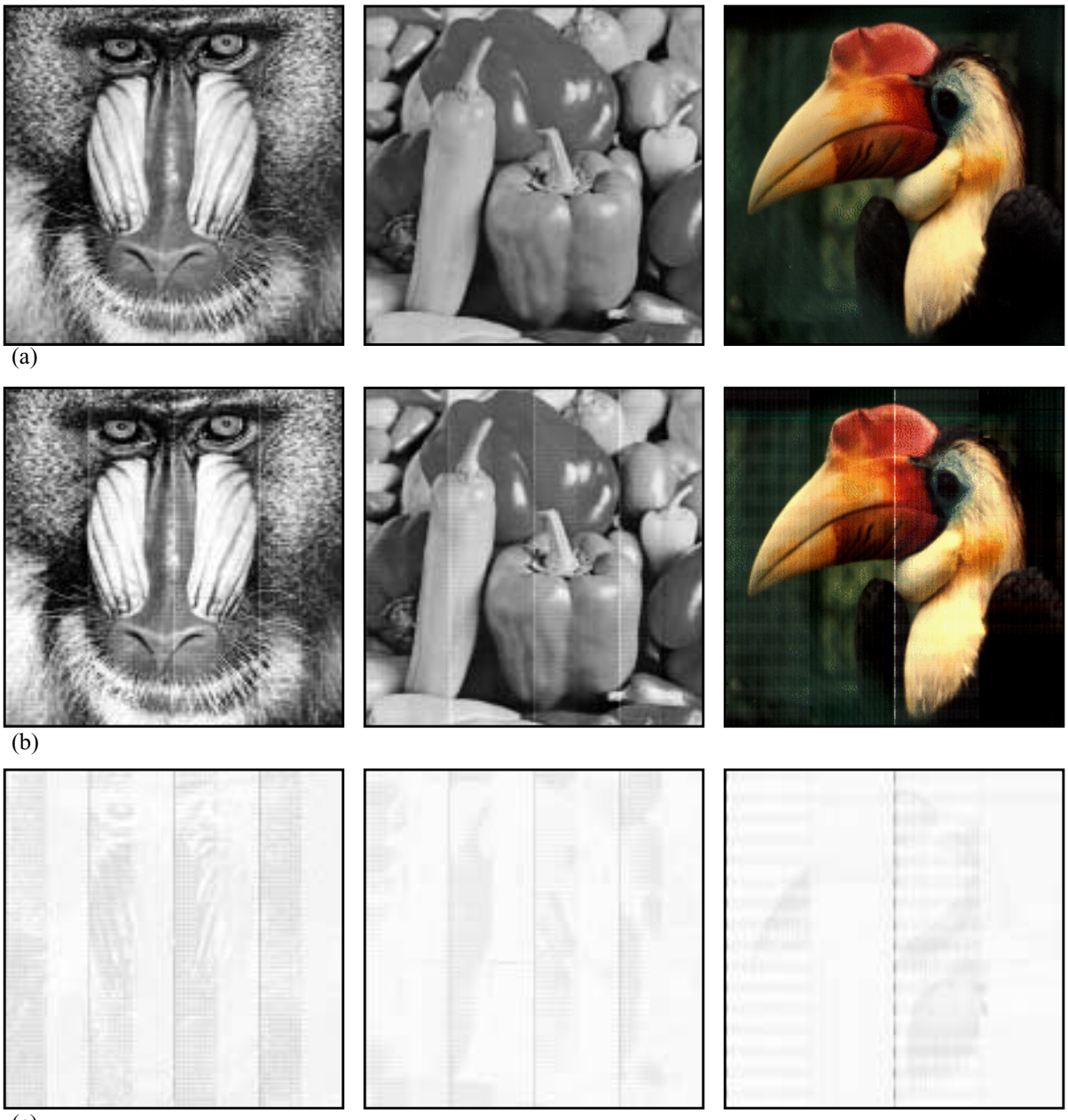
Comparison with previous aRAM chips

A comparison between different CMOS implementations of an aRAM chip can be made on the basis of the figure-of-merit (FOM) defined in [Good96]. This FOM relates the average power consumption, P_{avrg}, delay —for this, we will consider here the access time, the average between the acquisition (t_{acq}) and reading times (t_{read})— and accuracy, 2^N, which is the number of levels for an equivalent resolution of N bits:

$$\text{FOM} = \frac{P_{\text{avrg}}(t_{\text{acq}} + t_{\text{read}})}{2^{N+1}} \tag{4.80}$$

This FOM has units of energy (J, joules in the SI). The smaller the FOM,

the better the implementation. Notice that either lowering the power consumption, speeding up access times, or increasing the accuracy redound in a lower FOM. Table 4.2 displays the FOM computed for different implementations of the aRAM in CMOS technology. They are accompanied by the different chip data employed in the calculations. for those chips lacking accuracy measurements in their communications, 8bits have been assumed. From this data, the chip presented here features a much better FOM than the previous works. The only objection to this comparison is that power figures include consumption of the whole chip, what can, somehow, distort the computations. But it does not happen only in some of the chips having different functionalities included, also in this chip, the power measurements include the waste of the 15 output buffers that are not used in serial mode.



(a)

(b)

(c)

FIGURE 4.38. Image test samples: (a) input, (b) output, and (c) difference.

TABLE 4.2. CMOS aRAM chips comparison

Chip authors and year	[Franchi et al. 1992]	[Nishimura and Gray 1993]	[Simoni et al. 1995]	[Ping and Franca 1997]	This chip
technology	CMOS 1.6µm/ 1P	CMOS 1.2µm/ 2P	CMOS 1.2µm/ 2P	CMOS 1µm/2P	CMOS 0.5µm/ 1P
size (no. of cells)	32 x 32	2 x 910	64 x 64	20 x 57	32 x 256
power	30mW	170mW	12mW	85mW	73mW
accuracy	6-7b	8b	———	———	7-8b
range/supply	1.9V@5V	2.6V@5V	———	———	1.4V@3.3V
random access	Yes	No	No	No	Yes
writing time	800ns ♣♣	30ns ♣	16000ns ♣♣	56ns ♣	<100ns ♣♣
reading time	300ns ♣♣,♠♠	<100ns ♣,♠	———	112ns♣,♠	<100ns ♣♣,♠♠
FOM [Good96]	0.1831pJ♦	0.0109pJ	0.1831pJ♦	0.0218pJ	0.0035pJ

♣. Intra-chip ♠. Destructive reading
♣♣.Off-chip ♠♠. Non destructive
♦. Assuming 8b accuracy

5 A Complex-Cell Analog Computing Engine Chip

After millions of years of evolution, nature has developed one of the best vision device ever built, in terms of power consumption, dynamic range and physical profile: the vertebrate retina. It consists in an aggregation of wide range photodetectors and simple, relatively slow and fairly inaccurate but strongly cooperative, processing elements [Hube88]. In an attempt to mimic this achievement of nature, several neuromorphic [Mead89] silicon-based systems have been developed and reported in literature. Some of these works are listed and examined in [Koch95] and [Moin99]. The biological retina is the natural solution for the time-critical problem of processing the vast amount of data contained in visual images. Neuromorphic ICs, consisting of arrays of photosensors combined with analog signal processing at the pixel level [Mead89], constitute a bio-inspired alternative to artificial vision systems based on conventional digital signal processing. Depending on the application, digital systems can be unable to faithfully handle such an amount of information —a sequence of pictures of a standard size, 640×480 pixels, digitized in a 24b colour scheme and transmitted at 30fps, supposes a data flow of more than 27MB per second. The analog VLSI alternative consists in bringing the low-level image processing tasks to the focal plane, making the processing circuitry coexists with the photosensing devices at the pixel level. The inherently high degree of parallelization thereby obtained permits a much higher computing speed than in serialized processing.

But, analog parallel array processors do not only benefit from mimicking the structure of the biological retina. Behavioural models can be extracted from measurements on the natural vision system and adapted for analog VLSI implementation. Recent experiments demonstrate that a model based in CNNs can be hopefully employed to emulate retinal behaviour [Reke00a]. As we have seen in the previous chapters, the CNN framework provides the work-space for the design of complex spatio-temporal dynamics defined on a regular grid of elementary processors. The topology of the CNN resembles that of the retina. Both of them are com-

posed of layers of cells which are dynamically linked to their nearest neighbours. In both cases, inter-cell connections carry continuous signals. These coincidental facts prompt the feasibility of an analog VLSI implementation of such a model. It will be confirmed in this chapter by the design of a prototype chip in a standard CMOS technology.

5. 1. A bio-inspired 2-layer CNN model

5. 1. 1. Model description

Sketch of the biological retina

Based on physiological and pharmacological studies, started by the group of Prof. F. Werblin at UC Berkeley [Werb91], a CNN model has been developed which matches the observed behaviour of the vertebrate retina [Jaco96]. The retina is a specialized part of the nervous system brought to the sensory periphery, instead of having it integrated in the central nervous system. By performing early processing at the focal plane, the data flow to the visual cortex is greatly reduced, thus solving the problem of intelligent processing of visual information in a tight time frame.

The vertebrate retina has the structure displayed in Fig. 5.1 [Werb95]. A first layer of photodetectors at the top, the cone cells —a different type of cell, the rods, are specialized in sensing in very dim light conditions and saturate very easily—, captures light and converts it to activation signals. Bipolar cells carry these signals across the retina layers to the ganglion cells that interface the retina with the optical nerve, in a trip of several micrometers. The ganglion cells convert the continuous activation

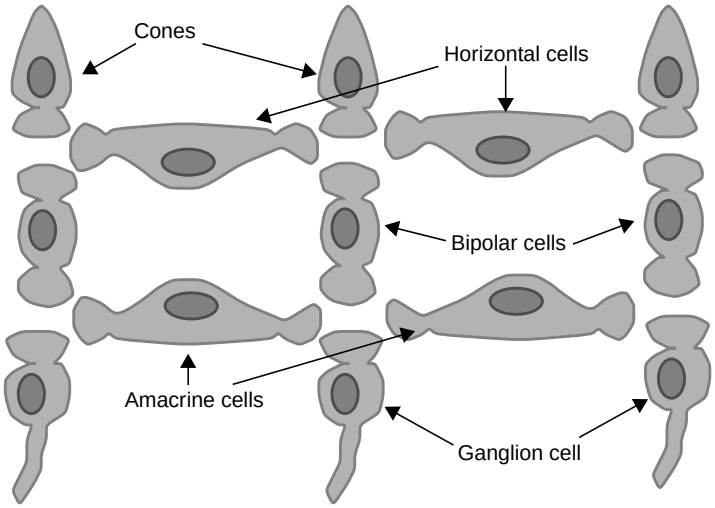


FIGURE 5.1. Schematic diagram of the vertebrate retina [Werb95] showing the layer of photosensors at the top and the ganglion cells connecting to the optic nerve. Between them there are several layers like that displayed in the picture.

signals proper of the retina to pulse-like action potential signals that can be transmitted over longer distances by the nervous system. In the way to the ganglion cells, the information carried by bipolar cells is affected by the operation of horizontal and amacrine cells. They form layers in which activation signals are weighted and promediated to bias photodetectors and to account for inhibition on the vertical pathway. The four main transformations that take place in this structure are: the photoreceptor gain control, the gain control of the bipolar cells, the generation of transient activity and the transmission of transient inhibition. Briefly, captured stimulus are promediated and the high-gain characteristics of the cones and the bipolar cells are shifted to adapt to the particular light conditions. These operations have a local scope and depend on the recent history of the cells. Once adaptation is achieved, patterns of activity are formed dynamically by the presence or absence of visual stimuli. Also inhibition is generated and transmitted laterally through the layers of horizontal and amacrine cells. As a result of these transformations, the patterns of activity reach the layer of ganglion cells. Then, they are converted into pulse-coded signals that are sent to the brain to be interpreted.

There are, in this description, some interesting aspects of each retinal layer that markedly resemble the characteristics of the cellular neural network: a 2D aggregation of continuous signals, local connectivity between elementary nonlinear processors, analog weighted interactions between them. Also, the complete signal pathway in the retina have the topology of a 3D, or, more properly $2\frac{1}{2}$D —a pile of 2D layers connected vertically— network. Motivated by these coincidences, a model for the operations of the biological retina based on CNNs have been developed [Werb95]. The observed behaviour of the retina have been successfully emulated by this model. Thus the following step is to derive a VLSI-oriented model that permits physical realization of an artificial retina on a CNN universal chip.

2nd-order 3-layer CNNUM chip

The inner and outer plexiform layers of the retina, IPL and OPL, are responsible for the generation of the retinal output and the capture of images, respectively. Both of them has been characterized by experimental measurement, leading to the results exposed below [Reke00a]. Let us consider first a CNN-based model for the OPL. The outer plexiform layer is composed of three different layers of cells. The first one, the photosensing layer, consists in an aggregation of cone cells. It is assumed here that the retina is adapted to lighting conditions and so the rods are saturated and thus remain silent. Added to the layer corresponding to the cones, there is a second layer composed of horizontal cells and a third layer of bipolar cells. Each of these layers has the structure of a 2D CNN itself. each of them has its own interaction patterns (templates) and its particular time constant. Cell dynamics are sustained by a first or a second order core. The structure of the OPL is depicted in Fig. 5.2(a), where interactions between layers of cells are portrayed. The input signal is captured by

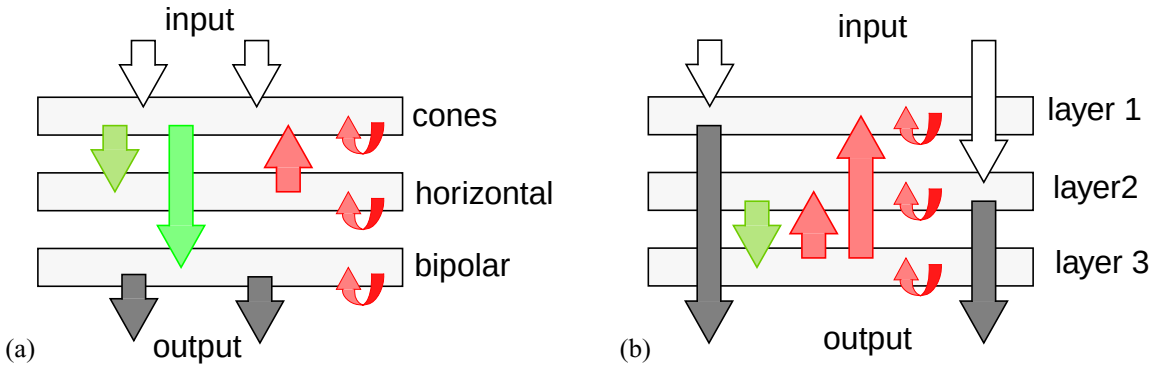


FIGURE 5.2. Conceptual diagram of the (a) OPL of the retina, and (b) the wide-field activity in the IPL.

the cones and feedforward to the layers of horizontal and bipolar cells. From the experiments it has been concluded that no feedforward connection exists between the horizontal cells and the layer of the bipolar cells. No feedback has been observed neither from the output of the bipolar cells to the previous layers. It has been deducted that the feedback connection of the horizontal cells to the layer of cones acts as a modulator of the feedforward functions rather than affecting directly to the cones state.

Concerning the inner plexiform layer (IPL), a simplified model is described in [Reke00a], that accounts for wide-field activity with three layers of cells. Wide-field activity is observed in certain amacrine cells of the IPL. It consists in the integration of the action potentials along a widely extended area previous to the ganglion cells. Based on the experimental records, a model has been conceived that consists in two layers of wide-field amacrine cells excited by the input signal, which in this occasion is the output of the bipolar cells, and a third layer that controls the dynamic of the previous layers by means of feedback signals. As before, the three layers are supposed to be 2D CNNs with their own internal coupling and their own time constant (see Fig. 5.2(b)).

Because of the relative simplicity of these models a complex-cell CNN universal machine chip has been proposed [Reke00b]. It incorporates some specific extensions to the original CNN model that will allow the VLSI implementation of the observed behaviour of the retina. This 2nd-order 3-layer CNN cell consists of 2 CNN layers coupled by some inter-layer weights and an additional layer embodying analog arithmetics to combine the outputs of the dynamically linked layers. The cells in the two first layers have a first order core, while the third layer, that can be also modelled in this way, has much faster dynamics ($\tau_3 \ll \tau_1, \tau_2$). Complex dynamics can be programmed via the adjustment of the intra- and inter-layer coupling strengths. Fig. 5.3 displays the conceptual diagram of the 2nd-order 3-layer (2 CNN layers and another one enabled for arithmetical combination of the others' states). Each connection, either feedback or feedforward is showing the signals transmitted by that specific channel. The evolution law of each cell, $C(i, j)$, following this CNN model is given by these two coupled differential equations:

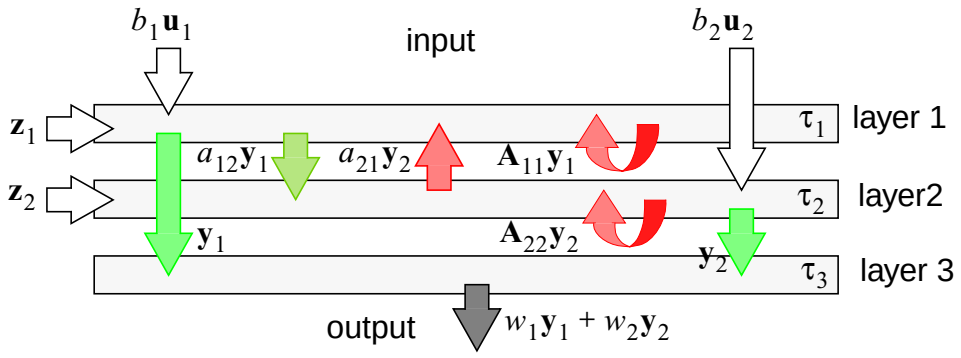


FIGURE 5.3. Conceptual diagram of the 2nd-order CNN.

$$\begin{aligned}\tau_1\frac{dx_{1,ij}(t)}{dt} &= -g[x_{1,ij}(t)] + \sum_{k=-r_1}^{r_1}\sum_{l=-r_1}^{r_1} a_{11,kl}\cdot y_{1,(i+k)(j+l)} + \\ &+ b_{11,00}\cdot u_{1,ij} + a_{12}\cdot y_{2,ij} + z_{1,ij} \\ \tau_2\frac{dx_{2,ij}(t)}{dt} &= -g[x_{2,ij}(t)] + \sum_{k=-r_2}^{r_2}\sum_{l=-r_2}^{r_2} a_{22,kl}\cdot y_{2,(i+k)(j+l)} + \\ &+ b_{22,00}\cdot u_{2,ij} + a_{21}\cdot y_{1ij} + z_{2,ij}\end{aligned} \tag{5.1}$$

where the nonlinear losses term and the output function in each layer are those of the FSR CNN model described in Chapter 1 and repeated here:

$$g(x_{n,ij}) = \lim_{m\to\infty}\begin{cases} mx_{n,ij} & \text{if} \quad x_{n,ij} > 1 \\ x_{n,ij} & \text{if} \quad |x_{n,ij}| \le 1 \\ -mx_{n,ij} & \text{if} \quad x_{n,ij} < -1 \end{cases} \tag{5.2}$$

and:

$$y_{n,ij} = f(x_{n,ij}) = \frac{1}{2}(|x_{n,ij}+1| - |x_{n,ij}-1|) \tag{5.3}$$

Although the proposed chip contains only 2 CNN layers, coupled and continuous-time, the evolution of more complicated dynamics binding more than 2 CNN layers can be emulated in discrete-time. This is done with the help of shared local analog memories (LAMs) and the built-in arithmetics. The CNN hardware is multiplexed in time and the intermediate results are stored in the LAMs. Tough not very efficient, it represents extended functionalities free of any charge.

Extended CNN model

Fig. 5.4 depicts the block diagram of the extended CNN model described by Eqs. 5.1, 5.2 and 5.3. Each processing element, cell, consists actually in two cells belonging to each of the two coupled layers. Synaptic connections between cells are linear. Each CNN layer incorporates the full set of feedback template elements $\{a_{nn,\,kl}\}$, where the indexes hold $k, l \in \{-r, -r+1, \ldots, -1, 0, 1, \ldots, r-1, r\}$. Only one feedforward connection $b_{nn,\,00}$, which is the central term of the control template, is available. Also a bias term $z_{n,\,ij}$, that can be different for each cell conforming a bias map. Finally there are coupling connections between both layers, one feeding the summing node of the second layer with the output of the first layer, $y_{1,\,ij}$, weighted by a_{21}, and the reverse connection, which pushes $y_{2,\,ij}$ into the first layer input node, weighted by a_{12}. Both layers have their own time-constant τ_n.

Programming different dynamics in this CNN model is possible by adjusting the template elements and the time-constants of the layers. The total number of synapses to be implemented on each cell is 22, plus the 2 bias maps multipliers, which will be treated as a second input image for each layer. Let us consider some operation examples. From these simulations, it has been found that the maximum required scaling of the time constant, if both layers realize equivalent functionality, is $10{:}1$.

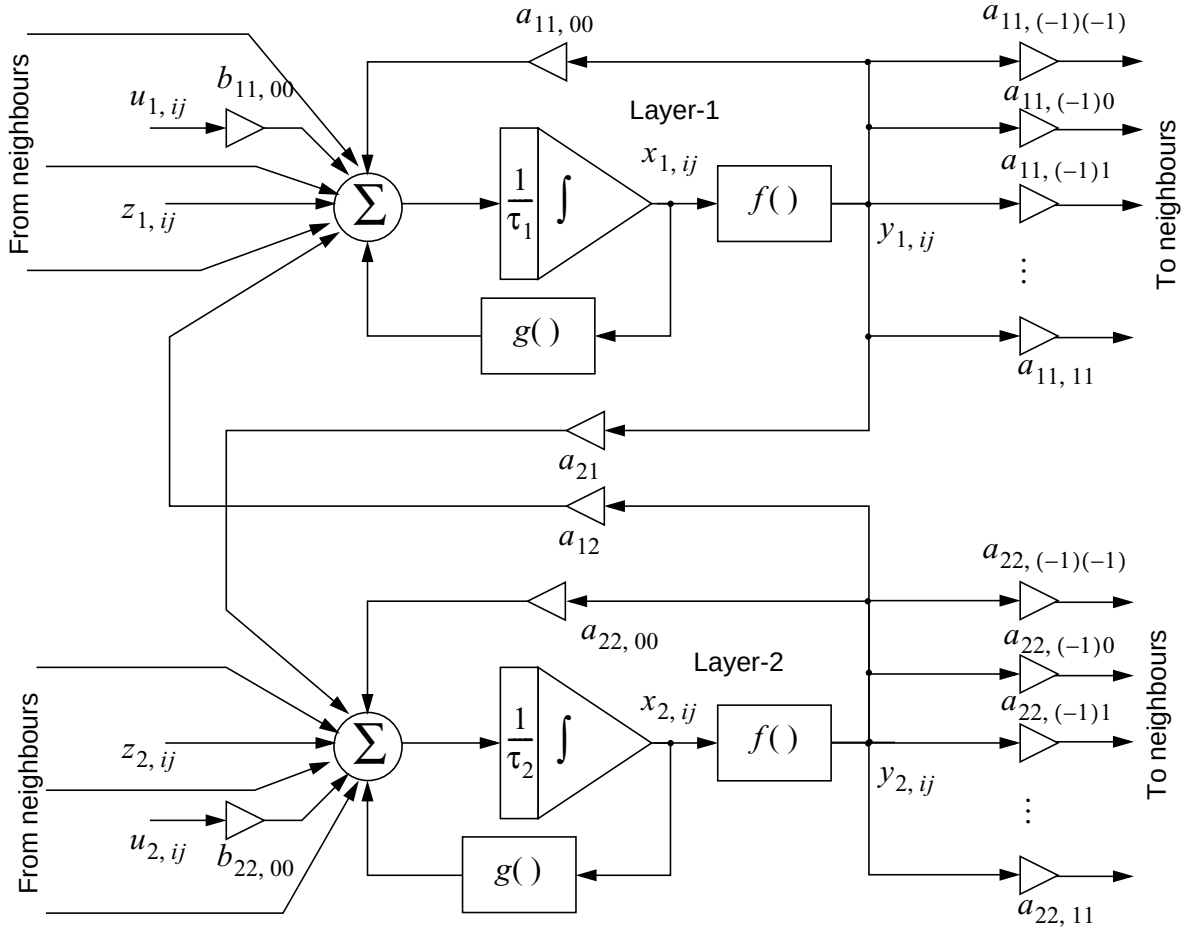


FIGURE 5.4. Block diagram of a cell belonging to two coupled CNN layers

5. 1. 2. Examples of operation

Let us consider some examples of the evolution of the 2-layer CNN depicted in Fig. 5.4. It has been programmed to display different phenomena in which the complex dynamics of the model are manifested. These phenomena have been studied by Dr. Csaba Rekeczky of the Analogic and Neural Computing Laboratory of the Computer and Automation Institute of the Hungarian Academy of Science, in Budapest. He has proposed the templates that we have employed here to illustrate the kind of operations that the proposed chip will perform [Reke00b]. These examples are classified into three different groups: 2D wave phenomena, pattern formation and image processing templates. These classes are, however, related in that similar dynamics lay beneath the different applications.

Wave phenomena

In these examples, different 2D structures in the original image grow or propagate all along the image plane in the way waves do. For all these wave phenomena, there is no control template, $b_1 = b_2 = 0$, hence, there are no formal input images, $\mathbf{u}_1$ and $\mathbf{u}_2$. Instead, there is an initial state for each of the two layers from which both evolve, $\mathbf{x}_1(0)$ and $\mathbf{x}_2(0)$. For the first layer, the initial state is constituted by black spots on a white background, that will trigger the wave fronts in all directions (see for instance the picture for $t = 0$ in Fig. 5.5), while the initial state for second layer is a totally empty frame. For all of the wave types that can be observed in this kind of network, the same feedback templates, $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$, apply being also equal for both layer:

$$\mathbf{A}_{11} = \mathbf{A}_{22} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.00 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \tag{5.4}$$

and the relation between the time constants, τ_1 and τ_2, is:

$$\frac{\tau_1}{\tau_2} = \frac{1}{5} \tag{5.5}$$

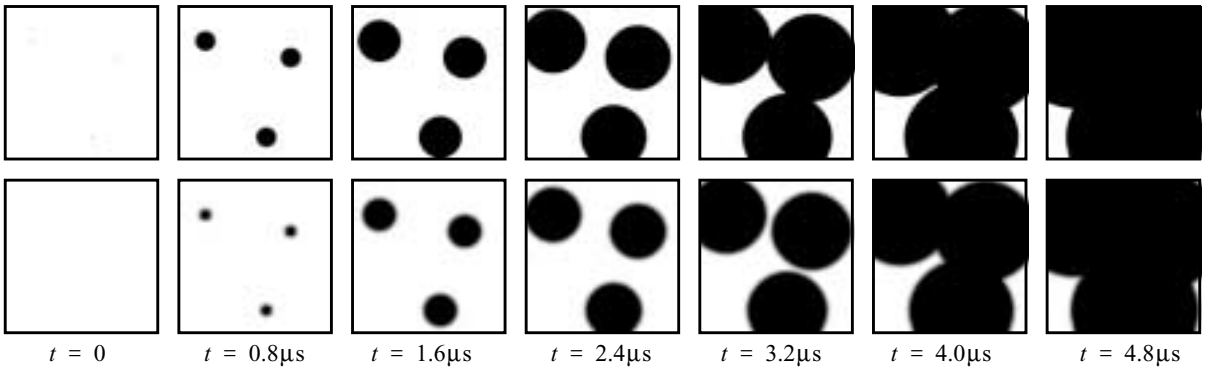


FIGURE 5.5. Triggered waves in the 2-layer CNN model ($\tau_1 = 0.2\mu s$ and $\tau_2 = 1\mu s$).

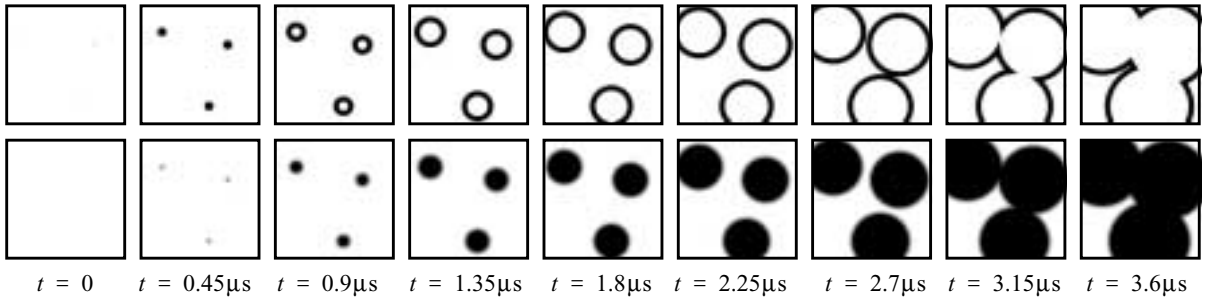


FIGURE 5.6. Travelling waves in the 2-layer CNN model ($\tau_1 = 0.2\mu s$ and $\tau_2 = 1\mu s$).

Now, it is the weight of the coupling between layers, elements a_{12} and a_{21} of the template set, and the magnitude of the bias terms, z_1 and z_2, that determines the type of wave that can be observed. For instance, the following values:

$$a_{21} = 1.00 \qquad a_{12} = 0.00 \qquad z_1 = z_2 = 3.75 \tag{5.6}$$

render the behaviour depicted in Fig. 5.5. It displays circular waves, triggered by the different black spots in the original image, the initial state of the first layer, that grow until they merge covering the whole image plane. These wave fronts are reflected in the second layer by means of a_{21}.

It can be played with the influence of the state of the second layer when fed back to the first one, e. g. programming the following values for the layers coupling and the bias terms, while maintaining the same values for the rest of the template elements and the two time constants:

$$a_{21} = 3.00 \qquad a_{12} = -5.00 \qquad z_1 = -1.25 \qquad z_2 = 2.25 \tag{5.7}$$

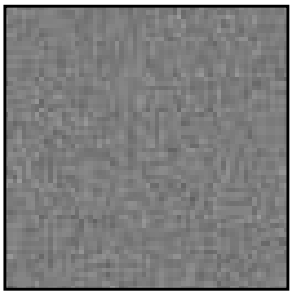
the second layer remains the same, but now the first layer displays wave fronts that travel across the image plane and annihilate each other after colliding (Fig. 5.6).

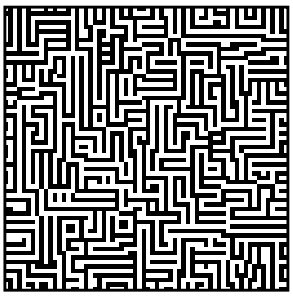
Pattern formation

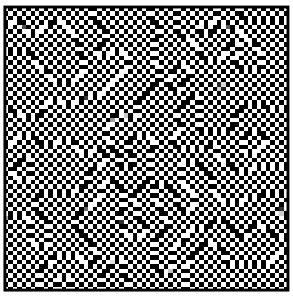
Pattern formation in autonomous —without inputs, $b_1 = b_2 = 0$, nor bias term, $z_1 = z_2 = 0$— CNNs has been studied in [Thir95] and [Crou95], where the principal templates for pattern formation were proposed. By not coupling both layers of the model under study, $a_{21} = a_{12} = 0$, the patterns depicted in (Fig. 5.7) can be obtained in either one or the other with the help of the following templates. This one:

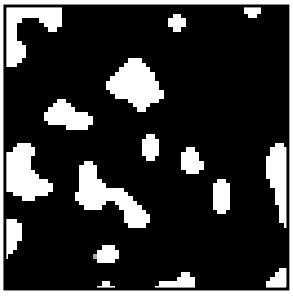
$$\mathbf{A}_{nn} = \begin{bmatrix} -0.5 & 0.0 & -0.5 \\ 0.0 & 2.0 & 0.0 \\ -0.5 & 0.0 & -0.5 \end{bmatrix} \tag{5.8}$$

allows generating the stripped pattern of Fig. 5.7(b), starting from the noisy rest-


(a)


(b)


(c)


(d)

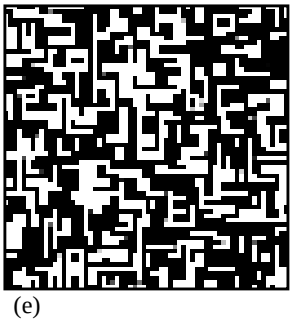

(e)

FIGURE 5.7. Initial state and output patterns formed in an autonomous CNN layer.

ing initial state depicted in Fig. 5.7(a). Starting from the same original image, a chequered pattern can be obtained (Fig. 5.7(c)), by using:

$$\mathbf{A}_{nn} = \begin{bmatrix} 0.5 & -1.0 & 0.5 \\ -1.0 & 2.0 & -1.0 \\ 0.5 & -1.0 & 0.5 \end{bmatrix} \tag{5.9}$$

These are the basic template structures. By the combination of these template elements distribution, some basic 1-layer patterns can be obtained. For instance, the following template:

$$\mathbf{A}_{nn} = \begin{bmatrix} 0.5 & 1.0 & 0.5 \\ 1.0 & 2.0 & 1.0 \\ 0.5 & 1.0 & 0.5 \end{bmatrix} \tag{5.10}$$

yields the patched cow-like pattern of Fig. 5.7(d). Also this one:

$$\mathbf{A}_{nn} = \begin{bmatrix} -0.5 & 1.0 & -0.5 \\ 1.0 & 2.0 & 1.0 \\ -0.5 & 1.0 & -0.5 \end{bmatrix} \tag{5.11}$$

results in the pattern composed of superposed rectangular shapes that can be seen in Fig. 5.7(e).

But the more interesting part is that, with the 2-layer CNN, combined patterns can be generated by allowing some coupling between the two layers. For example, if the feedback templates and the inter-layer coupling are programmed to be:

$$\mathbf{A}_{11} = \begin{bmatrix} 0.5 & 1.0 & 0.5 \\ 1.0 & 2.0 & 1.0 \\ 0.5 & 1.0 & 0.5 \end{bmatrix}$$

$$\mathbf{A}_{22} = \begin{bmatrix} -0.5 & -1.0 & -0.5 \\ -1.0 & 2.0 & -1.0 \\ -0.5 & -1.0 & -0.5 \end{bmatrix} \tag{5.12}$$

$$a_{21} = 4.0 \qquad \text{and} \qquad a_{12} = 0.0$$

then, the first layer, Fig. 5.8(b), ends in the same patched pattern obtained above (Fig. 5.7(d)) and the second layer exhibits a combined pattern with patches filled with dots (Fig. 5.8(c)).

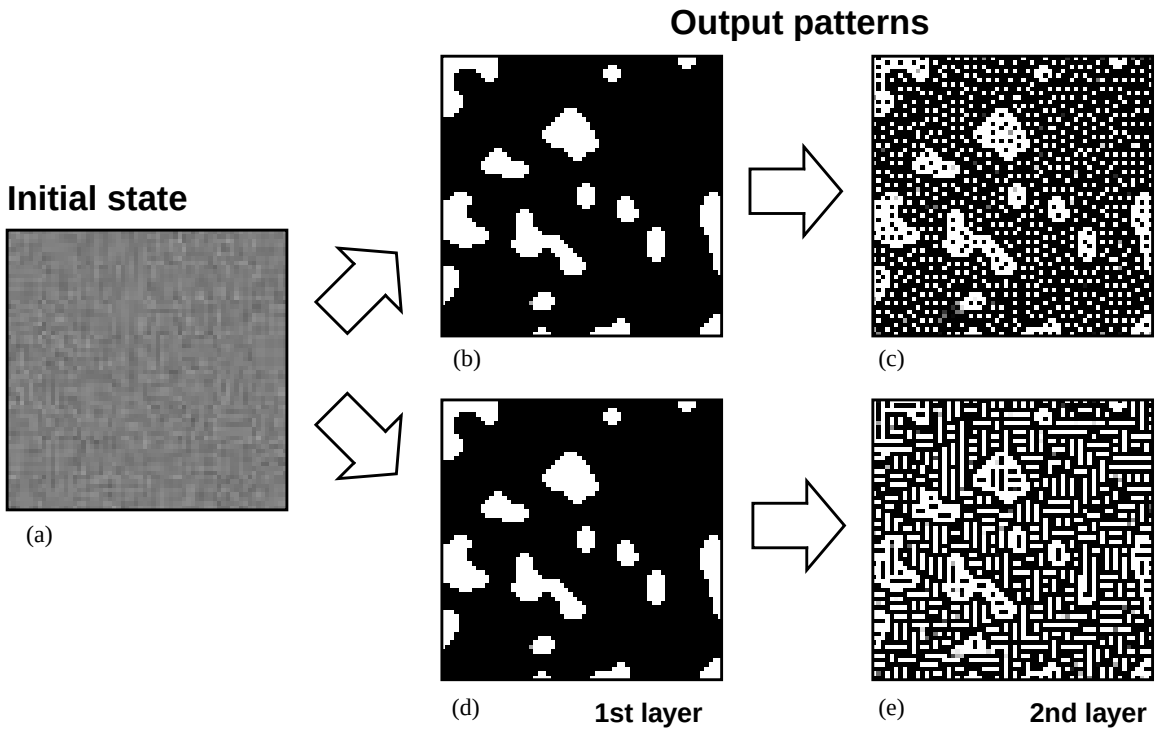


FIGURE 5.8. Combined patterns in the 2-layer CNN model.

A similar phenomenon can be observed if the feedback templates and the coupling between layers are governed by the following weights:

$$\mathbf{A}_{11} = \begin{bmatrix} 0.5 & 1.0 & 0.5 \\ 1.0 & 2.0 & 1.0 \\ 0.5 & 1.0 & 0.5 \end{bmatrix} \qquad \mathbf{A}_{22} = \begin{bmatrix} -0.5 & 0.0 & -0.5 \\ 0.0 & 2.0 & 0.0 \\ -0.5 & 0.0 & -0.5 \end{bmatrix} \tag{5.13}$$

$$a_{21} = 1.0 \qquad \text{and} \qquad a_{12} = 0.0$$

Now, while the first layer remains the same (Fig. 5.8(d)), the second layer depicts a combination of the cow-like patches with superimposed stripes (Fig. 5.8(e)).

Image processing

Pattern generation features can be applied to image processing to produce image halftoning. Image halftoning, or dithering, is a transformation that converts an image with greater amplitude resolution into an image with lesser amplitude resolution [Ulich87]. Particularly, a gray-scale image into a binary image in a way in which most of the elements of the original image — shapes, bodies, patches, borders, edges, etc.— are retained. Halftoning, or dithering, creates the illusion of tonal quality by sensible placement of the black and white dots. Being a process that can be established from a set of local rules, it seems reasonable to try to achieve image halftoning by means of a CNN. A 5×5 template has been employed in [Crou93] to realize this.

(a)

(b)

(c)

(d)

(e)

FIGURE 5.9. Original and dithered images obtained with the 2-layer CNN.

Also in [Reke00b], there is a set of 3×3 templates proposed to perform image halftoning with the 2-layer CNN model. The templates elements are:

$$\mathbf{A}_{11} = \begin{bmatrix} -0.36 & -0.60 & -0.36 \\ -0.60 & 1.05 & -0.60 \\ -0.36 & -0.60 & -0.36 \end{bmatrix}$$

$$\mathbf{A}_{22} = \begin{bmatrix} 0.05 & 0.20 & 0.05 \\ 0.20 & 0.00 & 0.20 \\ 0.05 & 0.20 & 0.05 \end{bmatrix} \qquad (5.14)$$

$$a_{21} = 1.00 \qquad a_{12} = -1.00$$

$$b_1 = 7.00 \qquad b_2 = 0.00$$

$$\text{and} \qquad z_1 = z_2 = 0.00$$

With these template values and the following relation between the layers' time constants:

$$\frac{\tau_1}{\tau_2} = 10 \qquad (5.15)$$

the results depicted in Fig. 5.9 are obtained. The original gray-scale image is displayed on top (Fig. 5.9(a)). The outputs of the first and second layers are Figs. 5.9(b) and (d). These images are still gray-scale but they have a much smaller pixel depth, i. e. amplitude resolution. Thresholding, an operation easily performed by CNNs as well, is applied to these pictures to obtain the pure black and white dithered images of Figs. 5.9(c) and (e).

Added to this, the programmable wave phenomena described at the beginning of this section can also be applied to realize image processing tasks —either on binary or gray-scale images. These operations relay on the propagation of waves triggered by image markers and constrained by the original image characteristics. For instance, let us see how the 2-layer CNN is able to detect the edges of a binary image, composed only of black and white pixels. For this operation, the following template elements are required:

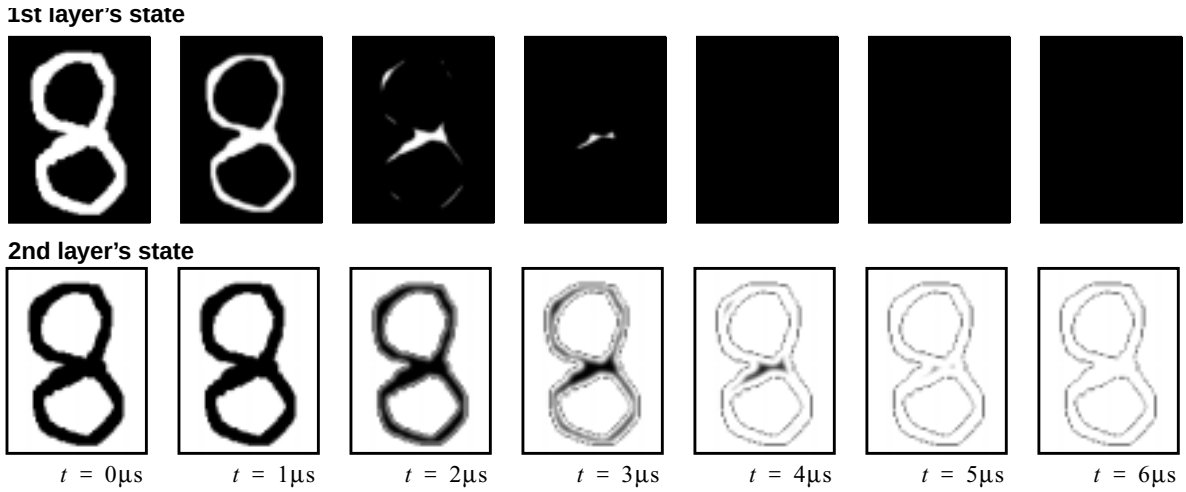


FIGURE 5.10. Wave-based edge detection in the 2-layer CNN ($\tau_1 = \tau_2 = 1\mu s$).

$$\mathbf{A}_{11} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.00 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \qquad \mathbf{A}_{22} = \begin{bmatrix} -0.10 & -0.10 & -0.10 \\ -0.10 & 2.30 & -0.10 \\ -0.10 & -0.10 & -0.10 \end{bmatrix}$$
$$a_{21} = -1.00 \qquad a_{12} = 0.00$$
$$b_1 = b_2 = 0.00 \tag{5.16}$$
$$z_1 = 3.75 \qquad \text{and} \qquad z_2 = 0.00$$

while the both time constants must be the same. It can be seen in Fig. 5.10 that the initial state (there are no formal inputs $\mathbf{u}_1$ and $\mathbf{u}_2$) for the first layer, $\mathbf{x}_1(0)$, is the inverse of the image to be processed, which is the initial state of the second layer, $\mathbf{x}_2(0)$. Then, the black areas that appear on the inverse image are grown by means of the programmed wave dynamics. This information is transmitted to the other layer, because of coupling, that use it to clean everything but the edges of the black figures.

Another image processing task that applies programmed wave phenomena, this time to gray-scale images, is contour detection. This operation, which is very important in image segmentation, relays in the following template elements:

$$A_{11} = \begin{bmatrix} 0.05 & 0.20 & 0.05 \\ 0.20 & 0.00 & 0.20 \\ 0.05 & 0.20 & 0.05 \end{bmatrix} \qquad A_{22} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.00 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix}$$
$$a_{21} = -2.50 \qquad a_{12} = -0.10$$
$$b_1 = 1.00 \qquad b_2 = 0.00 \tag{5.17}$$
$$z_1 = 0.00 \qquad \text{and} \qquad z_2 = 1.25$$

while both layers have the same time constant: $\tau_1 = \tau_2$.

In this occasion, a label spot in the second layer is grown, following the aforementioned principles, while the first layer diffuses and simultaneously increases progressively the contrast between the lighter and darker areas. This is realized with the intention of preventing the wave of the second layer to stop at the borders caused by minor details of the original image instead of advancing until the boundaries of the most significant elements (Fig. 5.11). The obtained patch can be now processed by the binary edge extractor in order to achieve contour detection.

If motion information in a sequence of images is employed to generate the markers that trigger the waves in this process, this set of templates can detect active contours. This operation is of importance in medical applications, given the low resolution and detail available, for instance, in ultrasound imaging —echography— or in X-ray screening (mammograms).

FIGURE 5.11. Contour detection in the 2-layer CNN ($\tau_1 = 1\mu s$ and $\tau_2 = 1\mu s$).

5. 2. CMOS chip design

With the intention of assessing the feasibility of the physical implementation of a complex-cell network, like that described in the previous sections, the design of a prototype chip in a standard CMOS technology has been accomplished. First, the elementary processor, cell, structure and building blocks will be analysed. Then, the system architecture will be reviewed, with a detailed description of the data and instruction I/O interfaces and protocols. Some of the building blocks of this highly complex IC —it contains more than 500,000 transistors, where 90% of them are working in analog mode— has been previously implemented and tested. Reports can be found in [Espe96a],[Domí97] and [Liña98]. Some of them will be described here for completion, but the emphasis will be put in the new functionalities and design challenges related with the implementation of the bio-inspired model.

5. 2. 1. The 2-layer CNNUM cell

Basic cell architecture

The elementary processor, i. e. the basic cell, of the CNN-based array processor has a similar architecture to that of the CNNUM cell (see Chapter 1). However, this complex-dynamics analog computing engine prototype includes two different continuous-time CNN layers. Therefore, as depicted in Fig. 5.12, the basic cell contains not only the local analog and logic memories (LAMS and LLMs) and a local logic unit (LLU), but, in this occasion, two different analog CNN core blocks. Each core block of the elementary processor belongs to one of the two different CNN layers of the network. Those CNN cores are coupled, following the above described model. The synaptic connections between processing elements

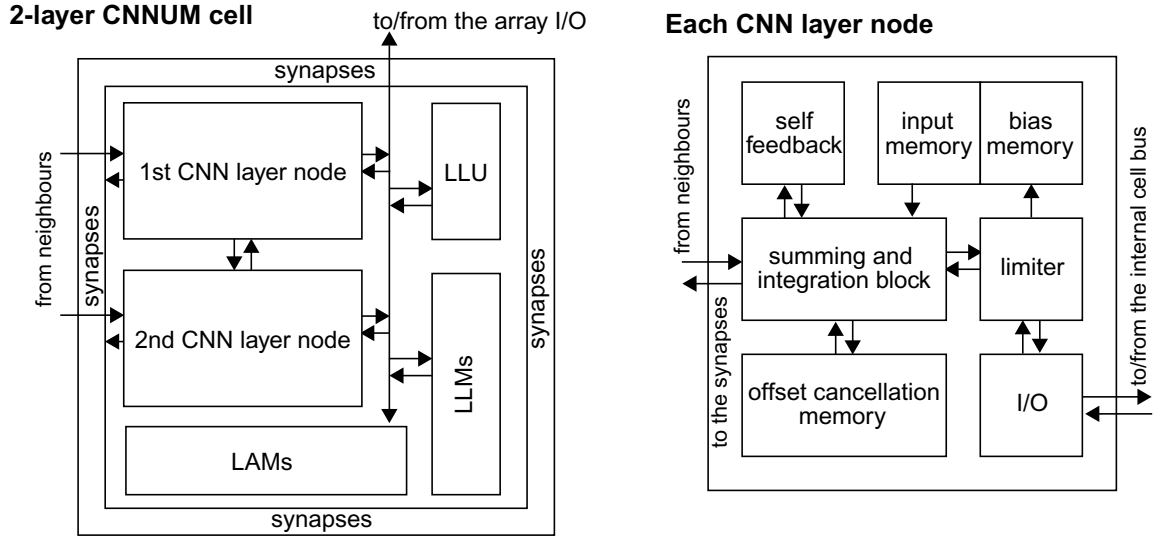


FIGURE 5.12. Conceptual diagram of the basic cell and the internal structure of each CNN layer node.

of the same layer are built around the cell core, as shown, while inter-layer coupling, kept within the pixel scope in this model, is placed inside the cell (represented by arrows between the processing layers in the diagram). All the blocks in the cell communicate via an intra-cell data bus, which is multiplexed to the array I/O interface. Control bits and switch configuration are passed to the cell directly from the global programming unit, which will be discussed later in this chapter.

The internal structure of each of the CNN cores of the cell is also depicted in the diagram of Fig. 5.12. They receive contributions from the rest of the processing nodes in the neighbourhood which are summed and integrated in the state capacitor. The two layers differ in that the first layer has a scalable time constant, controlled by the appropriate binary code, while the second layer has a fixed time constant. The evolution of the state variable is also driven by self-feedback and by the feedforward action of the stored input and bias patterns. There is a voltage limiter which helps to implement the limitation on the state variable of the FSR CNN model. This state variable is transmitted in voltage form to the synaptic blocks, in the periphery of the cell, where weighted contributions to the neighbours' are generated. There is also a current memory that will be employed for cancellation of the offset of the synaptic blocks. Initialization of the state, input and/or bias voltages is done through a mesh of multiplexing analog switches that connect to the cell's internal data bus.

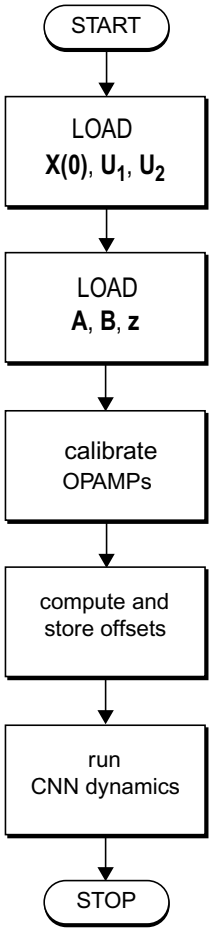


FIGURE 5.13. Flowchart of the complex network initialization and evolution.

Running complex spatio-temporal dynamics in this network requires following several initialization and calibration steps (Fig. 5.13). First of all, acquisition of the input image and auxiliary masks and/or patterns. For this purpose, the array I/O interface is directed to specific LAM locations in a row-by-row basis. After that, the analog instruction, i. e. the set of synaptic weights required for a specific operation, is selected and transmitted to all the cells in the array. Then, the offset of the critical OPAMPs is quenched in a calibration step. After this, the time-invariant offsets of the synaptic blocks are computed and stored in the current memories. Now the network is almost ready to operate. Then, the state capacitors and the feedforward synapses are initialized by means of the appropriate switch configuration, and the network evolution is run by closing the feedback loop in each processing element (according to the current freezing mask). Before stopping the network, the final state is stored in a LAM register for further operation.

Fig. 5.14 shows the layout of the basic cell of the array processor. The picture has been stripped of the two outermost metal layers in order to make visible all the underlying building blocks of the cell. The differences between the 1st and the 2nd layer CNN cores are quite noticeable.

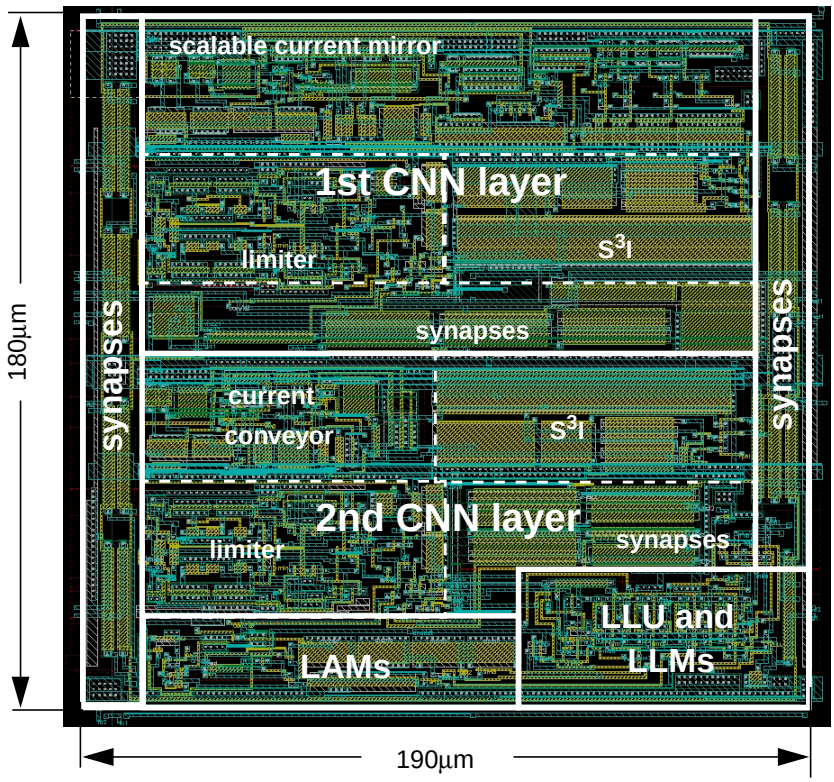


FIGURE 5.14. Layout of the basic cell.

One-transistor synapse

One of most important blocks in the CNN cell —because the majority of the circuit structures employed in the cell are strongly dependent on its implementation— is the synaptic block. The synapse is, *simply*, a four-quadrant analog multiplier. Their inputs will be the cell state or input variables and the corresponding weight signal, while the output will be the cell's contribution to a specific neighbouring cell. For convenience, the multiplier is chosen to have voltage inputs and current output. The first requirement arises from the fact that both the cell state and the weight signal, the signals to be multiplied, i. e. the multiplier inputs, must be distributed over different points in the circuit. The cell state must drive every synapse in the local scope while, at the same time, the weight signal must be transmitted to every cell in the array. If the cell state and the weight signals are represented by voltages, they can be easily conveyed to any high-impedance node by a simple wire. On the other side, the output of the multiplier should be represented by a current. This is because the contributions of all the neighbours are summed at the input of the processing core, and being in the form of currents, this summation can be readily achieved by wiring all these contributions concurrently to a low-impedance node. Therefore the synapse block will display the following large-signal characteristic:

$$I_o = kV_wV_x \tag{5.18}$$

where V_w and V_x represent the weight and cell state signals, I_o is the out-

put current, and k is a constant with dimensions of A/V^2. All signals in Eq. 5.18 can have either positive or negative sign. It is important to notice that the relation between the weight signal, V_w, and the output current, I_o, does not have to be linear. In fact, it is a question to be solved by the system software. However, the linearity with the cell state voltage, V_x, is a must. Thus our synaptic block will behave as:

$$I_o = G(V_w)V_x \tag{5.19}$$

where the function $G(.)$ has transconductance dimensions.

In addition, one thing that is common in this type of processing is that the weight signal does not change during the evolution of the network. It means that any deviation from the predicted behaviour that is not depending on V_x, e. g. an offset current, can be cancelled by auto-zeroing in a pre-processing calibration step.

With these assumptions about the form of the synaptic function, several CMOS circuit blocks for implementing the synapse have been reviewed and reported [Espe94a]. For instance, there are synapses with MOS transistors in weak inversion, exploiting the exponential law that governs this regime of operation to achieve multiplication [Socl85]:

$$ab = e^{(\ln a + \ln b)} \tag{5.20}$$

There are multipliers based on MOS transistor in strong inversion, operating in the saturation region, where their large-signal characteristic exhibits a quadratic law. Multiplication relies on this relationship:

$$(a+b)^2 - (a-b)^2 = 4ab \tag{5.21}$$

which is the principle behind the well-known Gilbert cell [Gilb68].

Finally, direct multiplication can be achieved by, for instance, a n-type MOS transistor operating in the ohmic region. Its low-frequency large-signal characteristic expressed by:

$$I_{DS} = \beta_n \left[V_{GS} - V_T(V_{SB}) - \frac{V_{DS}}{2} \right] V_{DS} \tag{5.22}$$

where $\beta_n = \mu_o C_{ox}'(W/L)$. A multiplication can be realized with this device as long as $V_{DS} \ll 2[V_{GS} - V_T(V_{SB})]$ holds [Tsiv94].

The advantages and inconveniences of using multipliers based in one or the other region of operation of the MOS transistor are exposed in [Espe94a]. Because of several reasons we are using a one-transistor synapse based in the MOS in triode region [Domí98]. On one side, the reduced area requirements, because four-quadrant behaviour is achieved with one single transistor. On the other side, a better relation between bias power and signal power is found thus leading to higher accuracy at lower power consumption (in the saturation region the information is car-

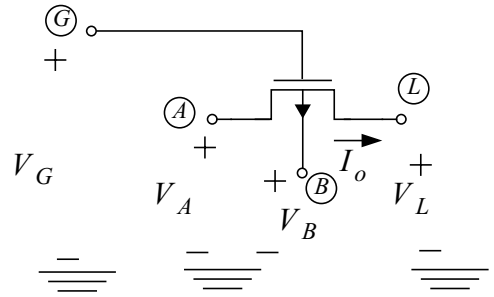


FIGURE 5.15. Multiplier using one single MOS transistor in ohmic region.

ried by a small fraction of the actual currents flowing through the devices). Let us describe how the one-transistor synapse works and how the signal ranges are established for proper operation. Consider a p-type MOS transistor operating in ohmic region (Fig. 5.15). The reason for selecting a p-type instead of a n-type MOS transistor is clear, the more resistive p-type channel allows smaller currents, and so power consumption, for the same transistor lenghts. Or, equivalently, for the same current levels, the required p-channel MOS is shorter than its n-type counterpart. Hence, the source-to-drain current of a PMOS transistor in the ohmic region is given by:

$$I_o = \beta_p\left[V_A - V_G - \left|V_{T_p}(V_{DD} - V_A)\right| - \frac{V_A - V_L}{2}\right](V_A - V_L) \tag{5.23}$$

if $V_A \geq V_L$, what means that node (A) is the source while node (L) is the drain. Here $\beta_p = \mu_o C_{ox}'(W/L)$ and the threshold voltage is given by:

$$V_{T_p}(V_{BS}) = -\left|V_{T0_p}\right| - \gamma(\sqrt{\phi_B + V_{BS}} - \sqrt{\phi_B}) \tag{5.24}$$

If, on the contrary, $V_A \leq V_L$, the node (A) becomes the drain while node (L) becomes the source. Thus, I_o is given by:

$$I_o = -\beta_p\left[V_L - V_G - \left|V_{T_p}(V_{DD} - V_L)\right| - \frac{V_L - V_A}{2}\right](V_L - V_A) \tag{5.25}$$

Both equations can be summarized in:

$$I_o = -\beta_p(V_A - V_L)V_G - \beta_p(V_A - V_L)\left(\left|\hat{V}_{T_p}\right| - \frac{V_A + V_L}{2}\right) \tag{5.26}$$

where the threshold adopts one of these two analogue forms:

$$\hat{V}_{T_p} = \begin{cases} -\left|V_{T0_p}\right| - \gamma(\sqrt{\phi_B + V_{DD} - V_A} - \sqrt{\phi_B}) & \text{if } V_A \geq V_L \\ -\left|V_{T0_p}\right| - \gamma(\sqrt{\phi_B + V_{DD} - V_L} - \sqrt{\phi_B}) & \text{if } V_A \leq V_L \end{cases} \tag{5.27}$$

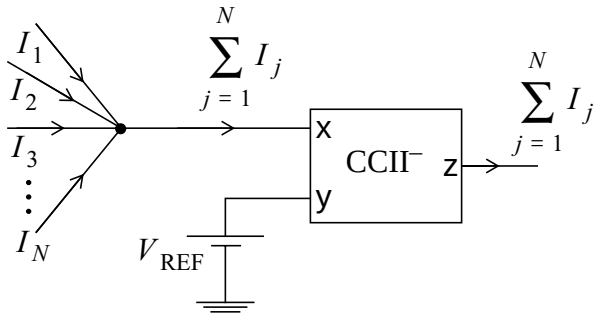


FIGURE 5.16. Current conveyor implementation of a virtual reference at the cell's input.

Now, the first thing is that V_L must be kept fixed in order to use V_A and V_G as single-ended input voltages, and to sense I_o as the output of the synapse. For this purpose we can employ a current conveyor [Smit68] at the current input node of each cell. This block is represented by the box labelled CCII^- in Fig. 5.16. This is a second-generation current conveyor in which a virtual ground is created between terminals x and y, holding:

$$V_{\mathsf{x}} = V_{\mathsf{y}} \tag{5.28}$$

while terminal y is a high-impedance node, $I_{\mathsf{y}} \equiv 0$. Node x has low-impedance and its current is *conveyed* to terminal z. The negative sign stands for the opposite directions of currents I_{x} and I_{z}. Thus:

$$I_{\mathsf{x}} = -I_{\mathsf{z}} \tag{5.29}$$

The current conveyor permits current sensing while maintaining a virtual reference at node $\textcircled{L}$. As depicted in Fig. 5.16, all the synapses contributing to the same cell can be connected to the same virtual reference. The only objection being that the impedance seen at this node must be well below the parallel of the output impedances of all the synaptic blocks.

Back to Eq. 5.26, notice that the second term in the right side of the equation does not depend on V_G, therefore node $\textcircled{G}$ is a strong candidate to hold the cell state variable voltage. But V_G must be always positive for the MOS transistor to operate above threshold, thus let V_G be composed of a reference voltage V_{x_0}, sufficiently high, and a superposed cell state signal V_x:

$$V_G \equiv V_X = V_{x_0} + V_x \tag{5.30}$$

Eq. 5.26 now becomes:

$$I_o = -\beta_p (V_A - V_L)(V_{x_0} + V_x) - \beta_p (V_A - V_L)\left(\left|\hat{V}_{T_p}\right| - \frac{V_A + V_L}{2}\right) \tag{5.31}$$

Now, in order to achieve four-quadrant multiplication (consider only the first summand in Eq. 5.31), V_A must be permitted to go up and below V_L. Let us select V_L

as the reference for the weight signal, V_{w_0}, being:

$$V_A \equiv V_W = V_{w_0} + V_w \tag{5.32}$$

Then Eq. 5.31 can be rewritten as:

$$I_o = -\beta_p V_w V_x - \beta_p V_w \left(V_{x_0} + \left| \hat{V}_{T_p} \right| - V_{w_0} - \frac{V_w}{2} \right) \tag{5.33}$$

which is a four-quadrant multiplier with an offset term that is time-invariant —at least during the evolution of the network— and not depending on the cell state. Therefore, we have arrive to a four-quadrant multiplier with single-ended voltage inputs and a current output, with a offset that can be eliminated by a calibration step, with the help of a current memory:

$$I_o = -\beta_p V_w V_x + I_{\text{offset}}(V_w) \tag{5.34}$$

Fig. 5.17 shows the simulated characteristic of the multiplier (using Hspice v. 99.2 and a MOS model of level 49) in a $0.5\mu m$ standard CMOS technology. Let us realize how the signal ranges are selected. The only limitations found in the first order model are the boundaries of the ohmic region in strong inversion. Assume that $V_w \geq 0$, this is, voltage at node $\textcircled{A}$ is above that of node $\textcircled{L}$, which is identically V_{w_0}. Then, for the transistor to operate in strong inversion (above threshold) it must hold:

$$V_{x_0} \leq V_{w_0} + V_w - V_x - \left| V_{T_p}(V_{DD} - V_{w_0} - V_w) \right| \tag{5.35}$$

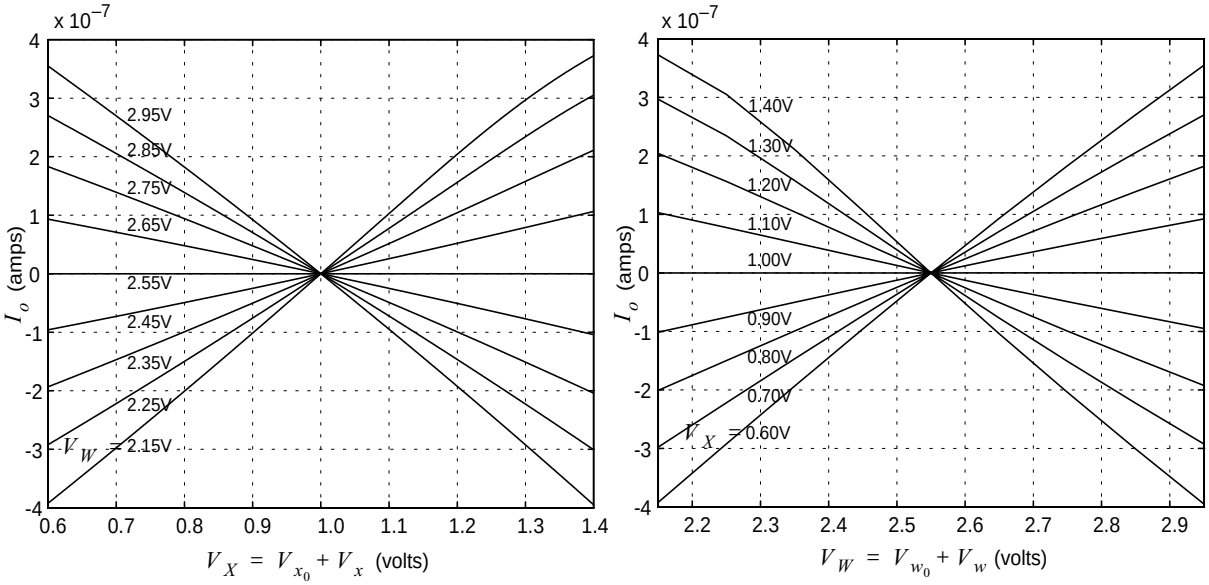


FIGURE 5.17. Multiplier output (without the offset term) vs. V_X and V_W.

where $V_{T_p}(.)$ can be computed by using Eq. 5.24. At the same time, for the transistor not to enter in saturation, it must hold:

$$V_{x_0} \leq V_{w_0} - V_x - \left| V_{T_p}(V_{DD} - V_{w_0} - V_w) \right| \tag{5.36}$$

with the same threshold. If, on the contrary, the voltage at node (A) is below that at node (L), this is $V_w \leq 0$, then

$$V_{x_0} \leq V_{w_0} - V_x - \left| V_{T_p}(V_{DD} - V_{w_0}) \right| \tag{5.37}$$

ensures that the transistor is operating above threshold and

$$V_{x_0} \leq V_{w_0} + V_w - V_x - \left| V_{T_p}(V_{DD} - V_{w_0}) \right| \tag{5.38}$$

is the condition for the transistor to operate in the ohmic region. Of these inequalities, the more restrictive is Eq. 5.38 when $V_x = V_{x_{\max}}$ and $V_w = -V_{w_{\max}}$, hence:

$$V_{x_0} \leq V_{w_0} - V_{w_{\max}} - V_{x_{\max}} - \left| V_{T_p}(V_{DD} - V_{w_0}) \right| \tag{5.39}$$

Another restriction to the operation of the single-transistor synapse can be found in the degradation of the mobility. The transversal electric field (normal to the surface of the channel) pushes the carriers towards the semiconductor surface where they suffer scattering, which renders a reduction in the speed of the carriers, thus degrading the mobility. This transversal electric filed depends on the gate voltage, thus the first summand in Eq. 5.34 will no longer be linear with V_x. A widely accepted model for this effect in a PMOS transistor is found in [Tsiv87]:

$$\mu = \frac{\mu_0}{1 + \theta(V_{SG} - \left| V_{T_P}(V_{SB}) \right|)} \tag{5.40}$$

where θ is a fitting parameter ($0.20\mathrm{V}^{-1}$ is a common value for a p-channel MOS in a $0.5\mu\mathrm{m}$ technology). Therefore, it can be thought of a maximum effective gate voltage:

$$V_{GE_{\max}} = (V_{SG} - \left| V_{T_P}(V_{SB}) \right|)\Big|_{\max} \tag{5.41}$$

beyond which the distortion introduced by mobility degradation exceeds the linearity requirements. This provides an upper limit for the gate voltage, then, if $V_w \geq 0$, it must be accomplished that:

$$V_{w_0} + V_w - V_{x_0} - V_x - \left| V_{T_p}(V_{DD} - V_{w_0} - V_w) \right| \leq V_{GE_{\max}} \tag{5.42}$$

and if $V_w \leq 0$, then:

$$V_{w_0} - V_{x_0} - V_x - \left| V_{T_p}(V_{DD} - V_{w_0}) \right| \leq V_{GE_{\max}} \tag{5.43}$$

Of these two inequalities, the more restrictive is Eq. 5.42 for $V_x = -V_{x_{\max}}$ and

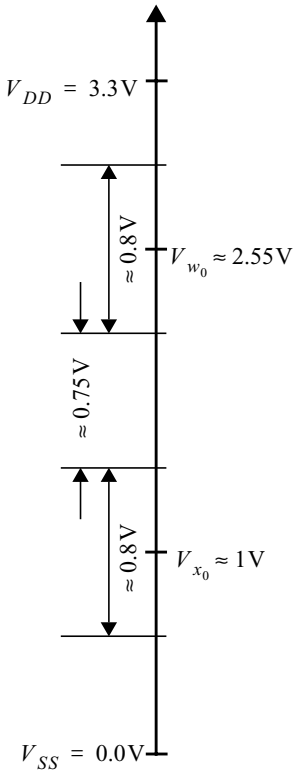


FIGURE 5.18. Voltage ranges for the state variable and the weight signals.

$V_w = V_{w_{\max}}$, yielding:

$$V_{w_0} + V_{w_{\max}} + V_{x_{\max}} - \left|V_{T_p}(V_{DD} - V_{w_0} - V_{w_{\max}})\right| \le V_{GE_{\max}} \quad (5.44)$$

Combining this equation with Eq. 5.39 we have:

$$V_{w_{\max}} + V_{x_{\max}} \le \frac{1}{2}V_{GE_{\max}} - \\ -\frac{1}{2}[\left|V_{T_p}(V_{DD} - V_{w_0})\right| - \left|V_{T_p}(V_{DD} - V_{w_0} - V_{w_{\max}})\right|] \quad (5.45)$$

For moderate linearity requirements, in a typical CMOS technology, the right hand side of Eq. 5.45 can be approximately 1V. Let us consider that V_x and V_w will be assigned the same voltage ranges, $\pm 400\text{mV}$ around their reference values. Thus $V_{x_{\max}} = V_{w_{\max}} = 400\text{mV}$. With this, we can go back to Eq. 5.39, where substituting the values of $V_{x_{\max}}$, $V_{w_{\max}}$ and $\left|V_{T_p}\right| \approx 0.8\text{V}$:

$$V_{x_0} \le V_{w_0} - 1.6\text{V} \quad (5.46)$$

Thus, V_{w_0} must be high enough to leave room for V_{x_0}, but not too large because the weight signal will progress up to $V_{w_{\max}}$ above V_{w_0}. In addition, we have to provide range for the current conveyor circuitry to maintain a virtual reference precisely at V_{w_0}, and for the circuits generating the weight voltages, which will have a limited output swing. If we select $V_{w_0} = 2.55\text{V}$, then there are 0.75V above V_{w_0} before hitting the power rail at 3.3V, what means one $\left|V_{T_p}\right|$, approximately. With this value, V_{x_0} results in 0.95V. Fig. 5.18 displays the current selection of voltage ranges for the PMOS synapse in a $0.5\mu\text{m}$ technology. These are the values employed in the previous simulations (Fig. 5.17).

Finally, once the voltage ranges are fixed, a maximum current per synapse is selected for meeting power requirements, in our case it will be $1.4\mu\text{A}$. With these values, the synapse is dimensioned. In our chip, it will be $2\mu\text{m}$ wide and $25.9\mu\text{m}$ long.

Current conveyor and level shifting

The current conveyor is implemented by the circuit of Fig. 5.19. Any difference between the voltage at node (L) and the reference V_{w_0} is amplified and the negative feedback corrects the deviation. For instance, if the voltage V_L moves slightly up V_{w_0}, because of a sudden increase in i_{in}, then the amplifier makes the gate voltage of M_p to decrease so as to absorb the extra amount of current, bringing back V_L to the value of the equilibrium. An analog behaviour is observed when V_L slightly falls below V_{w_0}.

The input impedance of this block is very low, as can be

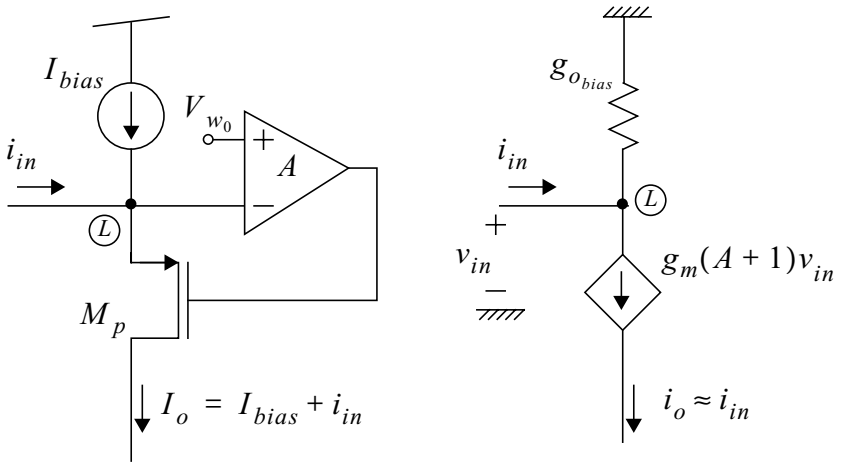


FIGURE 5.19. Current conveyor realization and small-signal equivalent.

obtained from its small-signal equivalent. Considering that $g_{o_{bias}} \ll g_m(A+1)$, then:

$$Z_{in} = \frac{v_{in}}{i_{in}} = \frac{1}{g_m(A+1)} \tag{5.47}$$

what means that small changes in the input current Δi_{in} does not disturb appreciably the virtual reference at node (L), this is $\Delta v_{in} \approx 0$.

The bias current is required to ensure that node (L) is always the source of transistor M_p. At the same time, this circuit permits the injection of a nearly exact copy of the input current at the state node, whose voltage range differs from that of the weight signals.

Notice that a voltage offset, V_{OS}, in the differential amplifier —that can be implemented with a simple OTA [Lake94] as it drives a very high impedance node, the gate of M_p— results in an error of the same amount in the reference

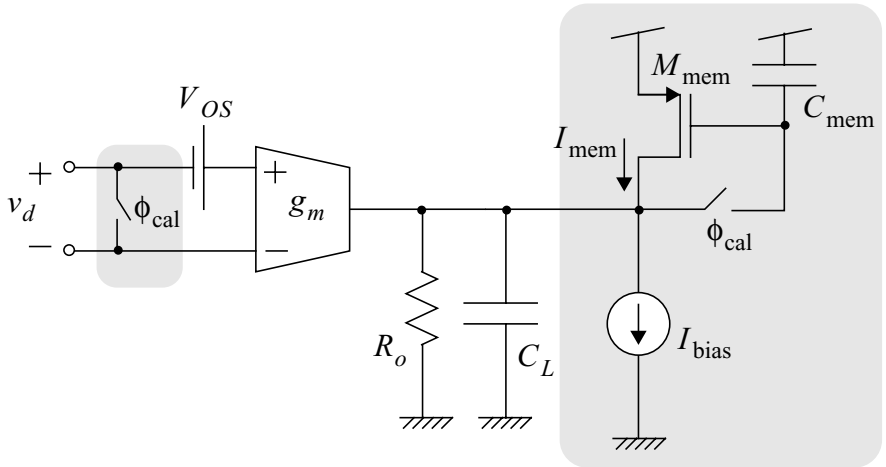


FIGURE 5.20. Offset calibration mechanism for the critical OTAs.

voltage implemented at node (L). Since the main contribution to the offset is random, this error will be distributed all along the array resulting in mismatched synaptic blocks that can degrade performance, e. g. a symmetrical propagation template yielding anisotropic evolution of the network. In addition, the drive for high cell-packing density forces the use of small devices, what aggravates the problem of the random offset. In order to avoid this, an offset calibration mechanism has been implemented at the critical OTAs' output. Fig. 5.20 represents the schematics for the offset cancellation circuit. The OTA input referred offset voltage, V_{OS}, and its output impedance has been taken out of the OTA block symbol, which is therefore supposed to have null offset and infinite output resistance. Let us forget for a moment the offset cancellation circuit (in the shadowed areas of Fig. 5.20). At low frequencies, the output voltage of the OTA is given by:

$$V_o = A(v_d + V_{OS}) \qquad \text{where} \qquad A = g_m R_o \tag{5.48}$$

Now, consider the error cancellation mechanism. When ϕ_{cal} is ON, then the inputs are shorted and M_{mem} is connected as a diode, so as its source-to-drain is, in steady-state:

$$I_{\text{mem}} = I_{\text{bias}} - g_m V_{OS} \tag{5.49}$$

After some time, ϕ_{cal} is turned off and, except from a remnant switching error, the current I_{mem} is memorized by means of the voltage stored in C_{mem}. Thus, if the output current of the OTA is now:

$$I_o = g_m(v_d + V_{OS}) \tag{5.50}$$

the total current injected into the load is:

$$I_L = I_o + I_{\text{mem}} - I_{\text{bias}} = g_m v_d \tag{5.51}$$

and therefore, at low frequencies:

$$V_o = \frac{g_m}{g_o} v_d = A v_d \tag{5.52}$$

S^3I current memory

As it has been referred, the offset term of the synapse current must be removed for its output current to represent the result of a four-quadrant multiplication. For this purpose, before the CNN operation, but after the new weights has been uploaded, all the synapses are reset to $V_x = 0$, this is $V_X = V_{x_0}$. Then the resulting current, which is the sum of the offset currents of all the synapses concurrently connected to the same node, is memorized. This value will be substracted on-line from the input current when the CNN loop is closed, resulting in a one-step cancellation of the errors of all the synapses. The validity of this method relies in the accuracy of the current memory. For instance, in this chip, the sum of all the contributions will

range, for the applications fro which it has been designed, from $18\mu A$ to $46\mu A$. On the other side, the maximum signal current to be handled is:

$$I_{\max} = \beta_{\text{ohm}} V_{x_{\max}} V_{w_{\max}} \approx 0.5\mu A \tag{5.53}$$

what means a total current range of $1\mu A$. If a signal resolution of $8b$ is pretended, then, $(1/2)\text{LSB} = 2\text{nA}$. In these conditions, our current memory must be able to distinguish 2nA out of the $46\mu A$. This represents an equivalent resolution of $14.5b$.

In order to achieve such accuracy levels, a so-called S^3I current memory will be employed [Toum93]. As depicted in Fig. 5.21, it is composed by three stages, each one consisting in a switch, a capacitor and a transistor. I_B is the current to be memorized. At the beginning, ϕ_1, ϕ_2 and ϕ_3 are ON, and the current I_B is divided into I_1, I_2 and I_3, which are diode-connected. Therefore, and being $V_1 = V_2 = V_3 = V_m$:

$$I_B = \sum_{i=1}^{3} I_i \qquad \text{where} \qquad I_i = \beta_i (V_m - V_{T0_n})^2 \tag{5.54}$$

or, equivalently:

$$V_m = V_{T0_n} + \sqrt{I_B / \left(\sum_{i=1}^{3} \beta_i \right)} \tag{5.55}$$

Once ϕ_1 is turned off, V_1 changes slightly because of the switching error, thus the current I_1, that was equal to $\beta_1 (V_m - V_{T0_n})^2$, now is approximately:

$$I_1 = \beta_1 (V_m - V_{T0_n})^2 - g_{m_1} \Delta V_1 \tag{5.56}$$

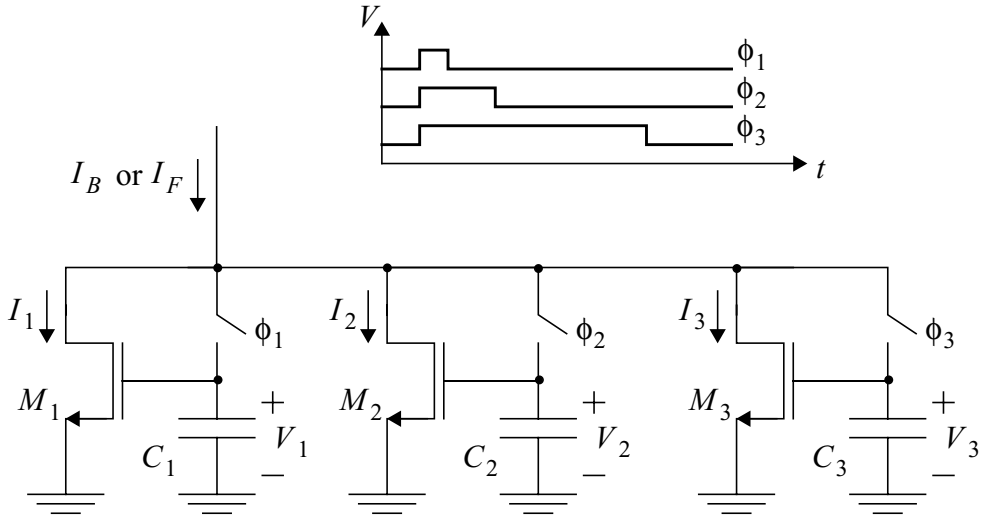


FIGURE 5.21. S^3I current memory schematics an timing

At the same time, I_2 and I_3 accommodate to absorb this error, as the sum of the currents has to be I_B. Hence, V_2 and V_3 change to V_m'. This voltage can be derived from the fact that $I_B = I_1 + I_2 + I_3$. Thus:

$$V_m' = V_{T0_n} + \sqrt{I_B / \left(\sum_{i=1}^{3} \beta_i \right) + (g_{m_1} \Delta V_1) / \left(\sum_{i=2}^{3} \beta_i \right)} \tag{5.57}$$

Then ϕ_2 is switched off. This makes the current I_2, which was equal to $\beta_2 (V_m' - V_{T0_n})^2$, become:

$$I_2 = \beta_2 (V_m' - V_{T0_n})^2 - g_{m_2} \Delta V_2 \tag{5.58}$$

and at the same time V_3 changes to a different value V_m'', because still $I_B = I_1 + I_2 + I_3$. Operating:

$$V_m'' = V_{T0_n} + \sqrt{I_B / \left(\sum_{i=1}^{3} \beta_i \right) + (g_{m_1} \Delta V_1) / \left(\sum_{i=2}^{3} \beta_i \right) + g_{m_2} \Delta V_2} \tag{5.59}$$

Finally ϕ_3 is turned off, introducing an error in I_3, that was given by $\beta_3 (V_m'' - V_{T0_n})^2$ and now becomes:

$$I_3 = \beta_3 (V_m'' - V_{T0_n})^2 - g_{m_3} \Delta V_3 \tag{5.60}$$

Now I_B can not be forced to enter the S^3I memory. Instead, there will be a final current which is the sum of the currents flowing through the transistors M_1, M_2 and M_3:

$$\begin{aligned} I_F = \sum_{i=1}^{3} I_i = \beta_1 (V_m - V_{T0_n})^2 - g_{m_1} \Delta V_1 + \\ + \beta_2 (V_m' - V_{T0_n})^2 - g_{m_2} \Delta V_2 + \beta_3 (V_m'' - V_{T0_n})^2 - g_{m_3} \Delta V_3 \end{aligned} \tag{5.61}$$

Substituting here the values of V_m, V_m' and V_m'', we find that:

$$I_F = I_B - g_{m_3} \Delta V_3 \tag{5.62}$$

the only error left is that corresponding to the last stage. The former stages do not contribute to the error in the memorized current. If the S^3I block is design so as to store the most significant bits in the first capacitor, and the less significant bits in the last one, then the error in the memorized current can be made quite small. Consider that the total resolution of the current memory is N. Let us assume that M_1 is conducting the most $N/3$ significant bits of the current I_B, then M_2 conducts the next $N/3$ and M_3 conducts the rest, thus:

$$I_1 = \left(\frac{I_B}{2^N}\right)\sum_{k=1}^{\frac{N}{3}} 2^{N-k} \qquad I_2 = \left(\frac{I_B}{2^N}\right)\sum_{k=\frac{N}{3}+1}^{\frac{2N}{3}} 2^{N-k}$$
$$\text{and} \qquad I_3 = \left(\frac{I_B}{2^N}\right)\sum_{k=\frac{2N}{3}+1}^{N} 2^{N-k} \tag{5.63}$$

then, an effective resolution for the last stage can be defined:

$$N_{\text{eff}} = \log_2\left(\sum_{k=\frac{2N}{3}+1}^{N} 2^{N-k}\right) \tag{5.64}$$

So I_3 can be rewritten in this form:

$$I_3 = \frac{2^{N_{\text{eff}}}}{2^N} I_B \tag{5.65}$$

but at the same time, because of the current division performed:

$$I_3 = \beta_3 I_B \Big/ \left(\sum_{i=1}^{3} \beta_i\right) \tag{5.66}$$

thus combining Eqs. 5.65 and 5.66:

$$\beta_3 \Big/ \left(\sum_{i=1}^{3} \beta_i\right) = \frac{2^{N_{\text{eff}}}}{2^N} \tag{5.67}$$

Back to the error in the memorized current, it has to be kept below the value of $(1/2)\text{LSB}$, this is:

$$g_{m_3}\Delta V_3 \leq \frac{I_B}{2^{N+1}} \tag{5.68}$$

Given that $g_{m_3} = 2\sqrt{\beta_3 I_3}$, and using Eqs. 5.65 and 5.67, it is found that:

$$\Delta V_3 \leq \sqrt{\frac{I_B}{\beta_3}} \cdot 2^{-\left(\frac{N_{\text{eff}}}{2}+\frac{N}{2}+2\right)} \tag{5.69}$$

And this is the design equation that relates the geometric aspect of transistor M_3,

through β_3, with the magnitude of the storage capacitor, via ΔV_3. Once we have β_3, β_1 and β_2 can be easily derived using Eqs. 5.54 and 5.63, resulting in:

$$\beta_1 = \left(\frac{\beta_3}{2^{N_{\text{eff}}}}\right)\sum_{k=1}^{\frac{N}{3}} 2^{N-k} \qquad \beta_2 = \left(\frac{\beta_3}{2^{N_{\text{eff}}}}\right)\sum_{k=\frac{N}{3}+1}^{\frac{2N}{3}} 2^{N-k} \tag{5.70}$$

From what has been explained, one can think that adding more stages to the current memory will endlessly increase accuracy. However, there is one factor that has not been accounted yet. As the order of the memory increase, the tinier the currents that have to be sensed by the last stages. There comes a point in which the leakages from the capacitors of the first stages are of the size of the current to be memorized by the last stages, thus making it impossible to reach a steady state current that corrects from the previous errors. This problem worsens as temperature rises. For instance, at 70C leakages can introduce changes in the memorized current in the order of $0.2\text{nA}/\mu\text{s}$. If the dynamics of the current memory require several μs to settle —because of the use of large capacitors and the tiny currents involved— the memorized current will display an error that is quite above the initial estimation.

Time-constant scaling

In the fixed time-constant CNN layer, the output of the current conveyor is injected, concurrently with an inverted copy of the memorized current offset, directly to the state capacitor, which is formed by the gates of all the synapses associated to this state variable. But if we are interested in scaling the time-constant, the input block is rather more involved. The time-constant of the CNN is defined in Sect. 1. 3. 1 as the ratio between the magnitude of the state-capacitor C_c, and the normalizing transconductance G_c employed to compute the actual weight voltages.

$$\tau = C_c / G_c \tag{5.71}$$

Because the physical implementation of C_c consist in the aggregation of the gate capacitances of the synapses, there is an initial value for C_c below which it can not be reached, unless any of synaptic blocks is removed. In addition, G_c is a parameter adjusted for the synapses implementation, and these are dimensioned attending to area and power concerns, therefore G_c can not be tweaked. So, at a first glance, the only possible action to take is to implement a adjustable capacitor, C_A, in one of the layers, whose value adds to the original C_c. Instead of the time-constant given by Eq. 5.71, the adjustable CNN layer will have a time constant of:

$$\tau_1 = \frac{C_c + C_A}{G_c} \tag{5.72}$$

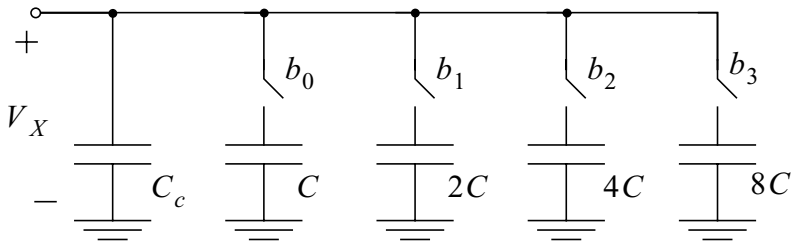


FIGURE 5.22. Scalable state capacitor implementation

Although it does not seem easy to implement a continuously adjustable capacitor, one showing a finite set of discrete values is quite simple, as can be seen in Fig. 5.22. The major drawback of this implementation is the excessive silicon real-state that must be reserved to allocate the scaled capacitors. For the example, if $C = C_c$, in order to have a time-constant up to 16 times larger, with 4b resolution, 16 times the area occupied by the synapses in the fixed time-constant layer will be required.

A better solution can be obtained by taking advantage of the properties of the FSR model [Espe94c]. In this CNN model, the losses term is implemented together with the central element of the feedback template. It means that there is no physical losses resistor in the cell core. Therefore, the differential equation that governs the evolution of the network (Eq. 1.29) is a sum of current contributions injected to the state capacitor:

$$\tau \frac{dx_{ij}(t)}{dt} = \sum_k I_k \tag{5.73}$$

Notice that scaling up/down this sum of currents is equivalent to speeding up/down the network dynamics. Therefore, sensing the input current, the sum of contributions, which is already done by the current conveyor, and scaling it with the help of a current mirror, for instance, will have the effect of scaling the time-constant. A circuit for continuously adjust the current gain of a mirror can be designed based on the active-input regulated-Cascode current mirror of [Serr94]. When transistors M_1 and M_2 are biased in saturation, then $I_1 \approx I_2$, assuming $\beta_1 = \beta_2 = \beta$, the differences due to the finite, although very high, output resistance of the mirror. But when they are operating in the ohmic region, this is, when $V_{w_0}, V_\tau \geq V_g + \left|V_{T0_p}\right|$, then, M_1 and M_2 currents are given by:

$$\begin{aligned} I_1 &\approx \beta(V_{DD} - V_g - \left|V_{T0_p}\right|)(V_{DD} - V_{w_0}) \\ I_2 &\approx \beta(V_{DD} - V_g - \left|V_{T0_p}\right|)(V_{DD} - V_\tau) \end{aligned} \tag{5.74}$$

as long as the voltages $V_{DD} - V_{w_0}$ and $V_{DD} - V_\tau$ are much smaller than $2(V_{DD} - V_g - \left|V_{T0_p}\right|)$, than can be easily accomplished in the ohmic region of operation. In these conditions, the ratio between the sensed and the reflected currents is a function of the controlling voltage V_τ:

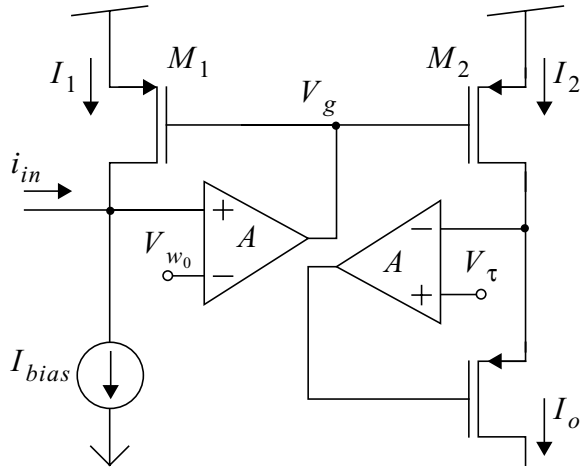


FIGURE 5.23. Active-input regulated-Cascode current mirror.

$$\alpha = \frac{I_2}{I_1} = \frac{V_{DD} - V_\tau}{V_{DD} - V_{w_0}} \tag{5.75}$$

The major disadvantage of using this circuit is the strong dependence on the power rail voltage. Computing the sensitivity of the scaling ratio with V_{DD}, we obtain:

$$S_\alpha^{V_{DD}} = \frac{V_{DD}}{\alpha}\left(\frac{d\alpha}{dV_{DD}}\right) = \frac{V_{DD}}{V_{DD} - V_\tau} \tag{5.76}$$

that can be around 10 for common values. If we compute now the variability of α:

$$\frac{\Delta\alpha}{\alpha} = S_\alpha^{V_{DD}}\left(\frac{\Delta V_{DD}}{V_{DD}}\right) \tag{5.77}$$

what means that a meagre 1% deviation on the value of the power rail voltage, ends up in a 10% deviation of the time-constant with respect to a cell in which this voltage matches the nominal. As we will see later, it is not strange that the power rail voltage deviates further more that this 1% is a densely packed 32×32 -cell parallel array processor chip. Although it is not of importance when dealing with non-propagative templates, the operations based on the propagation of waves across the image can be seriously damaged by a large mismatch in the time-constants of the layers (as was reported in Chapter 2).

An alternative to this is a binary programmable current mirror (Fig. 5.24). In this block, the input current, I_{in}, that must be always positive in the sense indicated in the figure, is scaled and reflected to the rest of the transistors, being the output current given by:

$$I_o = (1 + b_0 + 2b_1 + 4b_2 + 8b_3)\frac{I_{in}}{16} \tag{5.78}$$

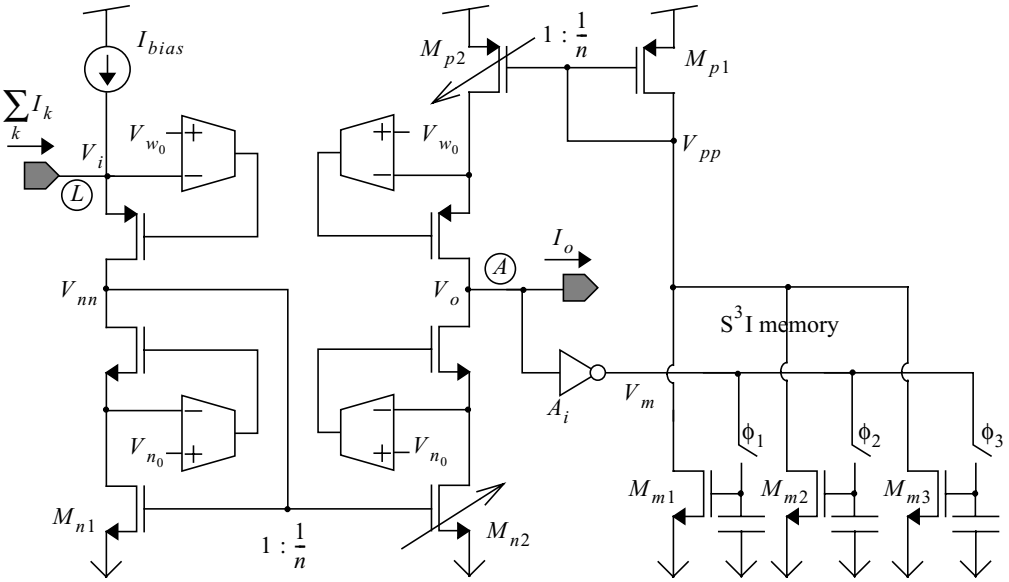


FIGURE 5.25. Input block with current scaling.

where b_0, b_1, b_2 and b_3 are the binary values, ones or zeroes, of the control bits. In this occasion, 4 bits will be more than enough to program the required relations between τ_1 and τ_2. The mismatch between the time constants of the different cells is now fairly attenuated.

A new problem arises, though, because of current scaling. If the input current can be reshaped to a 16-times smaller waveform, then the current memory has operate over, on one hand, the large signals when I_{in} is not scaled down, and on the other hand over the smaller signals when scaled. If designed to operate on large currents, the current memory will not work for the tiny currents of the scaled version of the input. If it is designed to run on small input currents, long transistors will be needed, and the operation will be unreliable for the larger currents. One way of avoiding this situation is to make the S^3I memory to work on the original unscaled version of the input current. Therefore, the adjustable-time-constant CNN core will be identical to the fixed-τ core, only that a binary programmable current mirror, properly biassed, follows the input stage formed by the current conveyor plus the S^3I current memory. The problem now is that the offsets introduced by the scaling circuitry add up to the signal and the required accuracy levels can be lost. Our proposal is depicted in Fig. 5.25. It consists in placing the scaling block

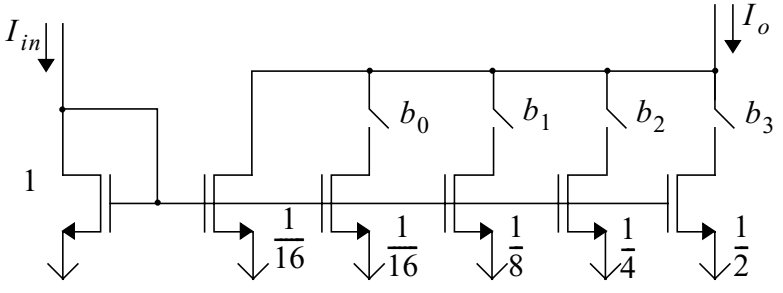


FIGURE 5.24. Binary programmable current mirror (4b).

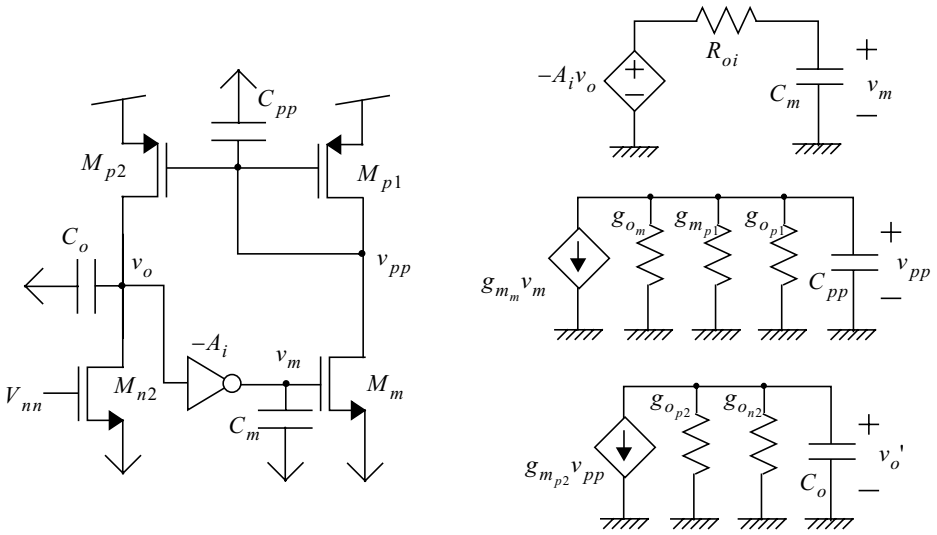


FIGURE 5.26. Simplified schematics of the feedback loop and its small signal equivalent.

(programmable mirror) between the current conveyor and the current memory. In this way, any error that is not dependent on the state voltage, V_x, will be cancelled at the zeroing phase. In the picture, the voltage reference generated with the current conveyor, the regulated-Cascode current mirrors and the S^3I memory can be easily identified. The inverter, A_i, driving the gates of the transistors of the current memory is required for stability. Without it, the output node, (A), will diverge from the equilibrium. Operation of this circuit is similar to what has been described in previous sections. Before running the CNN dynamics, the current offsets of all the synapses contributing to this cell are injected to the virtual reference at node (L). This current is scaled down to one n-th of its value by means of the adjustable current mirror formed by M_{n1} and M_{n2}. The arrow over M_{n2} stands for the binary programmability of this device. The value of n is given, in a 4b mirror, by:

$$n = 1 + b_0 + 2b_1 + 4b_2 + 8b_3 \tag{5.79}$$

Then, if all the transistors of the S^3I memory are conducting, this is ϕ_1, ϕ_2 and ϕ_3 are ON, then, the negative feedback loop makes M_{p_2} to conduct the same current as M_{n2}. M_{p2} is also adjustable so as to make M_{p1} and the current memory to work with the same current ranges than the input stage. The rest of the operation has been already described. The current memory stores successively the remaining most significant bits of the input current, plus the errors accumulated. When it is done, the CNN loop can be closed and the output current I_o represent the scaled sum of the contributions, substracted of the error terms.

The critical aspects of this circuit are related with the feedback loop formed by M_{p1}, M_{p2}, M_{n2}, the inverting amplifier A_i and the transistors M_m, when sensing the offset current. During this process the output current I_o is zero because the current path to the state capacitor is open. Later, during the CNN evolution, this feedback loop will be open and the CNN core loop is closed. Let us examine the stability of the feedback loop once the input current has been established. It means that V_{nn} can be considered a bias voltage.Fig. 5.26 depicts a simplified schematic

diagram of the critical feedback loop. First of all, it must be taken into account that during the three different phases in which the loop is closed (ϕ_1,ϕ_2 and ϕ_3 ON, ϕ_1 OFF and ϕ_2 and ϕ_3 ON, and, finally, ϕ_1 and ϕ_2 OFF and ϕ_3 ON) the values of g_{m_m} and C_m change, so the stability conditions must hold for any possible set of values. Let us consider the small-signal equivalent circuit pictured in Fig. 5.26. For the first stage we have:

$$\frac{v_m}{v_o} = -\frac{A_i}{1+\frac{s}{p_2}} \qquad \text{where} \qquad p_2 = \frac{1}{C_m R_{oi}} \tag{5.80}$$

then for the second stage, assuming $g_{o_{p1}}$, $g_{o_{m1}} \ll g_{m_{p1}}$:

$$\frac{v_{pp}}{v_m} = -\frac{g_{m_m}/g_{m_{p1}}}{1+\frac{s}{p_3}} \qquad \text{where} \qquad p_3 = \frac{g_{m_{p1}}}{C_{pp}} \tag{5.81}$$

and finally, for the third, making $g_{o_2} = g_{o_{p2}} + g_{o_{n2}}$:

$$\frac{v_o}{v_{pp}} = -\frac{g_{m_{p2}}/g_{o_2}}{1+\frac{s}{p_1}} \qquad \text{where} \qquad p_1 = \frac{g_{o_2}}{C_o} \tag{5.82}$$

Thus, breaking the loop at the inverter's input, in order to compute the loop gain, we arrive to:

$$T(s) = \frac{v_o'}{v_o} = -\frac{T_0}{\left(1+\frac{s}{p_1}\right)\left(1+\frac{s}{p_2}\right)\left(1+\frac{s}{p_3}\right)} \tag{5.83}$$

where $T_0 = A_i(g_{m_m}/g_{m_{p1}})(g_{m_{p2}}/g_{o_2})$ is the low-frequency loop gain.

Fig. 5.27 displays the Bode diagram of $T(j\omega)$. Notice the relative position of the poles in the frequency axis. The nearest pole, that is at node v_o, will be employed to compensate the loop for stability. As g_{m_m} and C_m decrease for the latest phases of the current memorization, the loop will be more stable because this causes T_0 to decrease and p_2 to grow, breaking away from p_1 and thus increasing the phase margin [Gray93]. Therefore, the worst situation will occur when ϕ_1,ϕ_2 and ϕ_3 are ON, and thus the circuit is designed to be stable in these conditions. It is also important that A_i is kept reasonably low, otherwise it will displace the magnitude plot up in the y-axis, moving the unity-gain frequency, ω_u, towards the value of the inversion ω_{180}. This means a loss of phase margin, and can compromise the loop stability.

As we commented before, leakage currents can degrade the S^3I memory operation especially as the operation temperature rises. Although the negative feedback moves the circuit towards the correction of the errors, it maybe too slow to settle at

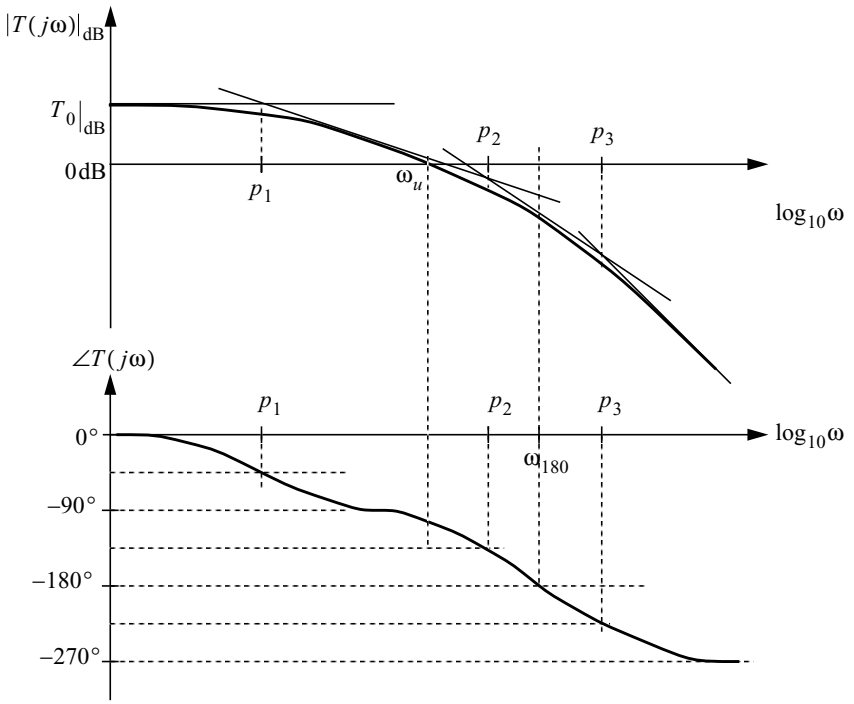


FIGURE 5.27. Bode diagram of the feedback loop gain $T(j\omega)$.

a value before leakages modify the position of the equilibrium point. Therefore, compensation must be kept under a limit to avoid slowing down the loop dynamics in excess.

State-voltage limiter

The voltage limiter employed to implement the CNN node state saturation is the same found in [Liña98]. As depicted in Fig. 5.28(a), it consists in a soft-limiter, composed by M_n and M_p, and a hard-limiter built with voltage comparators. The soft-limiter acts when the input node is isolated from the output. Its mission is to avoid excursions of V_i all the way up or

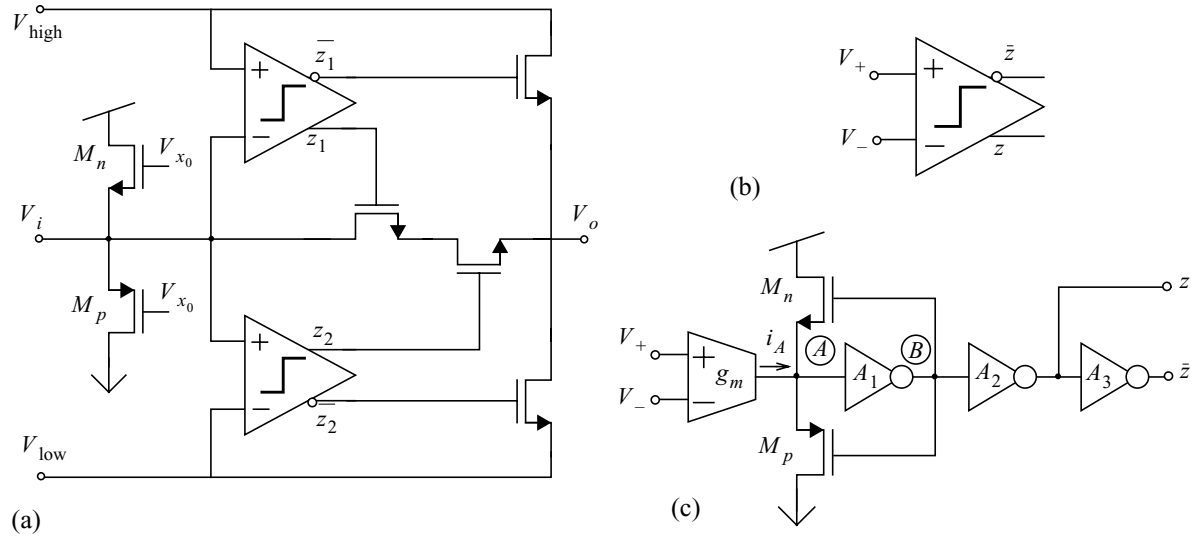


FIGURE 5.28. (a) State-voltage limiter and (b) voltage comparator symbol and (c) implementation.

down to the power rails. This, on one side, delays the limiter response, as the input node can be quite far from the comparators' references, and on the other, can cause malfunction of previous stages. With the soft-limiter, the input node is constrained between $V_{x_0} - V_{T_n}$ and $V_{x_0} + \left|V_{T_p}\right|$.

About the hard-limiter block, its operation results pretty obvious from the schematics. Each time V_i exceeds V_{high}, or falls below V_{low}, the corresponding voltage comparator disconnects the output node from V_i and attaches it to either V_{high} or V_{low}, respectively, which represents the saturation limits for the state variable. The state capacitor is connected to node V_o. The implementation of the voltage comparators consists in a simple OTA and a current comparator of the type presented in [Rodr95]. The transconductance amplifier converts the voltage difference to a current which is sensed by the current comparator. For this block, the better precision is obtained by a capacitive input impedance. But the problem is that integration of the input current can result in large voltage excursions at the input, what slows down the response of the comparator to sudden changes in the sense of the current. To avoid these excursions a resistive input impedance is needed, but this at the expense of losing part of the resolution accomplished by the capacitive input impedance. The current comparator employed here (Fig. 5.28(c)) behaves either as a capacitive or a resistive input comparator, depending on the region of operation. At the quiescent point, $V_+ - V_- = 0$, $i_A = 0$, neglecting the offset, and M_n and M_p are not conducting. Any small variation of the voltage difference around this point is transformed to a small i_A current and sensed by the capacitive impedance of node $\textcircled{A}$. Here, the maximum resolution is achieved. Then, for large positive (or, alternatively, negative) input currents, V_A grows (or decreases), making V_B fall (or rise) and thus permitting M_p (or M_n) to conduct. They close a negative feedback loop that clamps the input node voltage to a value that is not far from the quiescent voltage $V_A\big|_Q$. In particular:

$$\begin{aligned} V_A\big|_{\max} &= V_A\big|_Q + \frac{\left|V_{T_p}\right|}{1 + A_1} \qquad \text{if} \qquad i_A > 0 \\ V_A\big|_{\min} &= V_A\big|_Q - \frac{V_{T_n}}{1 + A_1} \qquad \text{if} \qquad i_A < 0 \end{aligned} \tag{5.84}$$

where A_1 is the voltage gain of the inverter, thus these voltages are very close to $V_A\big|_Q$. The inverters A_2 and A_3, placed after node $\textcircled{B}$, are required to recover the logic levels as V_B does not go far from the quiescent value neither. These inverters, as they spend most of the time at one the logic levels, can be implemented by CMOS logic. However, the inverter A_1 will spend most of the time in the high-gain region. If it is implemented by a complementary MOS inverter, its two transistor will be mostly ON. This will result in an excessive power draught, so it would be wise to realize it by a different technique, e. g. a single-stage inverting amplifier with a n-type MOS pull-down and an active load.

This completes the revision of the circuits comprised in the CNN cores of the basic cell. Outside them are the local logic unit (LLU) and the local analog and logic memories (LAMs and LLMs).

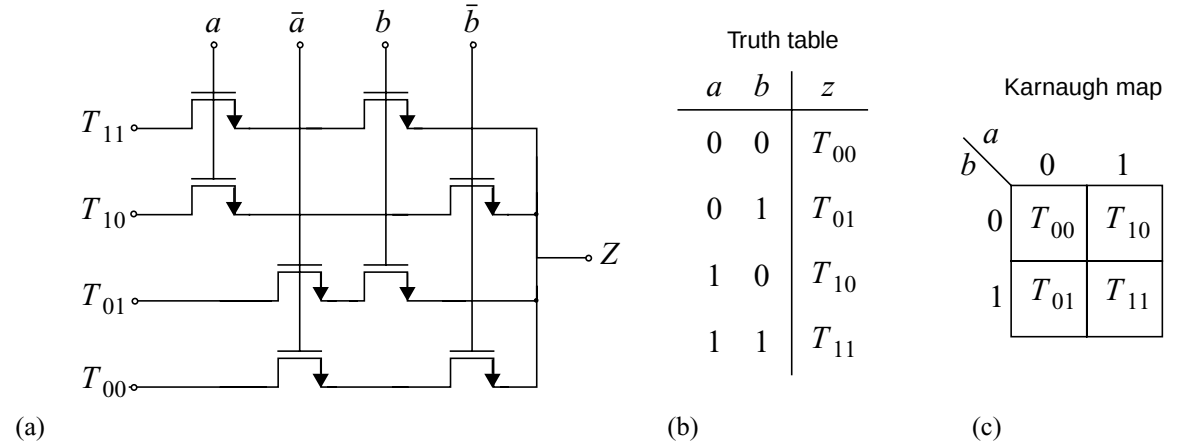


FIGURE 5.29. (a) Pass-transistor logic implementation of the LLU, (b) truth table and (c) Karnaugh map.

Local logic unit (LLU) and local memories (LAMs and LLMs)

The LLU is a fully-programmable 2-bits logic gate. It is implemented by a pass-transistor logic scheme, proposed and validated in [Espe96a], in which the outputs are programmed by the user. Fig. 5.29 displays the schematic of the implementation. As can be seen by the truth table or the Karnaugh map, any possible combinatorial function of two logic variables, a and b, can be obtained just by setting signals T_{ij} to the appropriate voltages. For instance, the exclusive OR function of the two inputs which is defined as $z = ab' + a'b$, can be realized by making T_{00} and T_{11} match the logic zero voltage while making T_{01} and T_{10} be equal to the logic one.

The LAMs are implemented with analog memory circuits of the type described in detail in Sect. 4. 3. The only difference being the use of shorted MOS transistors as capacitors. Concerning the LLMs they are just static RAM cells (see Fig. 4.5(a)).

5. 2. 2. System architecture and peripheral circuitry

Once the different building blocks for the implementation of the 2layer CNNUM cell has been reviewed, let us consider the system design and the circuits in the periphery of the array. These circuits are responsible for interfacing the parallel processing array with the world outside, of generating the appropriate references for the analog blocks and of storing and controlling the program flux.

Floorplan of the chip

The proposed chip consists in a analog parallel processing array of 32×32 cells, which are identical and of the type described in the previous section (Fig. 5.30). It is surrounded by a ring of circuits implementing the boundary conditions for the CNN dynamics. Apart from the main

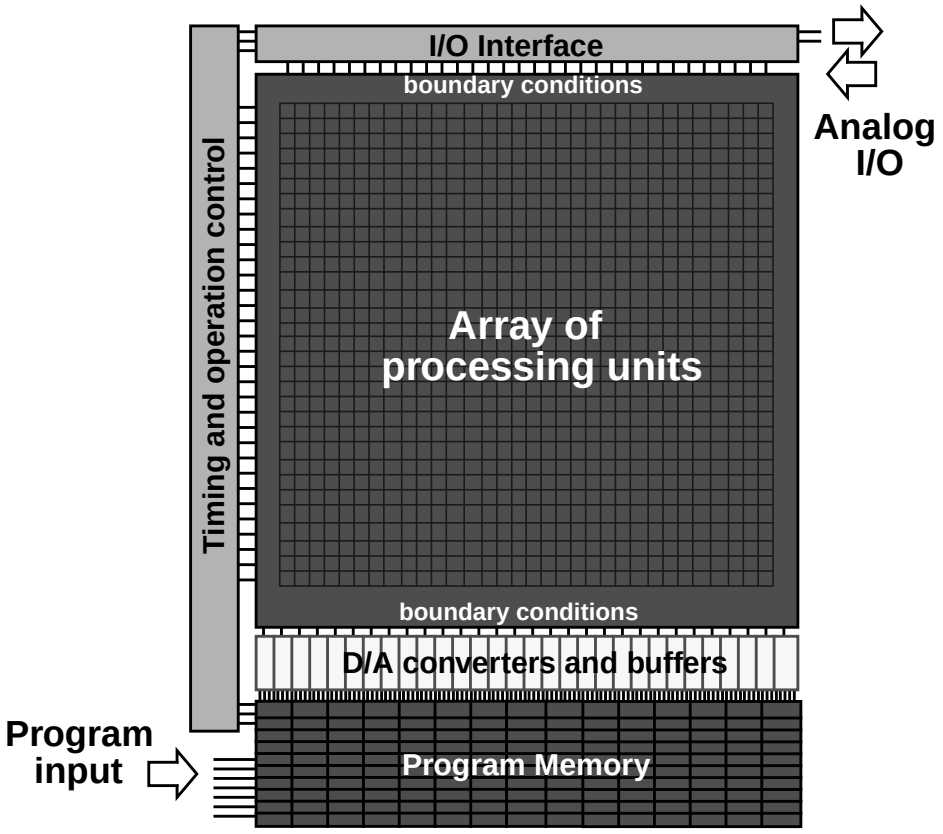


FIGURE 5.30. Floorplan of the CACE1K prototype chip.

array, there is a I/O interface, a timing and control unit and a program memory. The I/O interface consists in a serializing-deserializing analog multiplexor. It accommodates the serial analog I/O channel to the 32 I/O lines corresponding to the 32 columns of the array by means of a battery of S/H blocks. The corresponding row and column address decoders, controlled by the timing unit, are part of this block.

On the other hand, the operation control and program memory blocks constitute what in the CNNUM architecture was referred as the GAPU —the global analog-logic program unit. The program memory is composed, on one side, of 16 blocks of SRAM of 64 bytes of capacity dedicated to the analog program (the analog program register, or APR), this is 1kB, and on the other side, 4 blocks of 128 bytes each, this is 0.5kB, for the logic program. This is the logic program register (LPR) and the switch configuration register (SCR) and, in this occasion, some control signals for the I/O interface. The operation control unit constitutes the interface between the program memory and the processing array. It is composed by several logic circuits and transmission gates. Digital signal buffering can be considered part of the operation control unit. In addition, the analog instructions and reference signals, codified in one section of the program memory, need to be transmitted to every cell in the network in the form of analog voltages. Thus, a bank of D/A converters interfaces the APR with the processing array. Distributing analog references across large distances within a chip is not a trivial task. Apart from the

problems derived from electromagnetic interference, voltage drops in long metal lines carrying currents can be quite noticeable. Thus, signal buffering and low-resistance paths must be provided to avoid this, especially in the case of weight signals, that enter the synapses through a relatively low impedance node.

Finally, the timing unit is composed by an internal clock/counter and a set of FSMs (finite state machines [McCl86]) that generate the internal signals that enable the processes of images up/downloading and program memory accesses.

Power and references distribution

As already mentioned, conveying analog voltages from the boundaries of the array to the inner cells is not trivial. Especially if, on one side, the metal lines supporting the signals have to carry important current intensities, and on the other, the width of these lines must be reduced because of cell area compromises. These resistive lines carrying some current cause voltage signals to drop. In the case of the weight signals —that have to be transmitted to every synapse in the network, entering through a low-impedance node and, thus, dragging a quite perceptible amount of current— this voltage drop can seriously compromise the appointed resolution. Also for the power supply lines, that carry an important amount of current, voltage drops are a serious problem, as they can cause misfunction of the inner cell's circuitry. In consequence, it is important to develop a reliable model for this phenomenon. This will permit a precise design of the metal lines that convey the power and reference voltages. Let us start by considering a one-dimensional model (Fig. 5.31). The network, this time, is a row of cells that can be represented by a set of nodes from which a current I_o is drawn. This current represents the power needs of one single cell under normal operation conditions. This quantity (approximately $300\mu A$) is obtained by simulation. The cells in the row will be connected to each other by segments of the referred metal lines. Each of these segments is represented in the model by the resistance R. Because of the current pulled from every node, the reference voltage V_{REF} will degrade to different values at each node that will be referred as $V_1, V_2, \ldots, V_N$. In order to compute the effect of all the sources in the voltage of a particular node, V_i, we will calculate the effect of each source, voltage or current, separately and then we will apply superposition theorem [Deso69]. First of all, let us compute the effect of the voltage sources, V_{REF}, directly

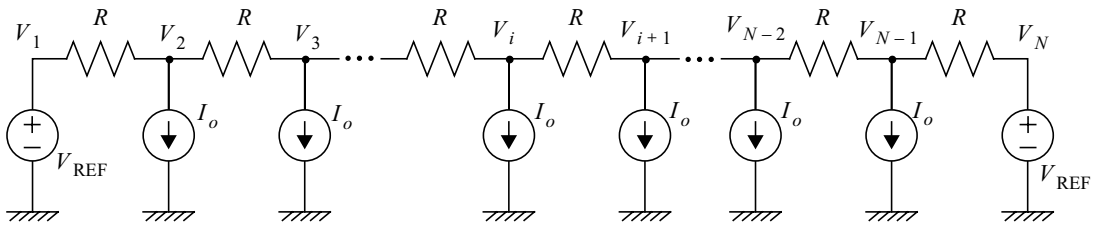


FIGURE 5.31. 1-D model for the voltage decay towards the centre of the network.

connected to nodes ① and Ⓝ. Assume that all the current sources disappear, are nulled, and that node Ⓝ is tied to ground. Hence, node ⓘ has $(i-1)$ resistors in series connecting it to V_{REF}, and $(N-i)$ from ⓘ to ground. Thus the effect of the source is given by voltage division:

$$V_i\big|_{V_{\mathrm{REF}①}} = \frac{N-i}{N-1}V_{\mathrm{REF}} \tag{5.85}$$

For the other voltage source, this renders:

$$V_i\big|_{V_{\mathrm{REF}Ⓝ}} = \frac{i-1}{N-1}V_{\mathrm{REF}} \tag{5.86}$$

Then, for a current source I_o at the left of node ⓘ —let us say at node ⓙ, while $j<1$ — we must eliminate the rest of the current sources and connect nodes ① and Ⓝ both to ground. The contribution, obtained by analysing the current division, is:

$$V_i\big|_{j<i} = -\frac{(N-j)(i-1)}{N-1}I_oR \tag{5.87}$$

For a current source at the right, $j>i$:

$$V_i\big|_{j>i} = -\frac{(N-i)(j-1)}{N-1}I_oR \tag{5.88}$$

Any of them applies for $j=i$. Then adding up the contributions:

$$V_i = V_i\big|_{V_{\mathrm{REF}①}} + V_i\big|_{V_{\mathrm{REF}Ⓝ}} + \sum_{j=2}^{i-1} V_i\big|_{j<i} + \sum_{j=i}^{N-1} V_i\big|_{j\geq i} \tag{5.89}$$

what results in:

$$V_i = V_{\mathrm{REF}} - \frac{(N-i)(i-1)}{2}I_oR \tag{5.90}$$

From this equation, the minimum occurs at $i=\mathrm{floor}[(N+1)/2]$, that is the centre cell in the 1-D array, what means that the maximum error in the propagation of the reference or power voltage V_{REF} is given by:

$$\left|\Delta V_{\max}\right| = \left|V_{\mathrm{REF}} - V_{\frac{N+1}{2}}\right| = \frac{(N-1)^2}{8}I_oR \tag{5.91}$$

This expression results quite useful in order to determine the appropriate width of the power lines that ride across the array. For instance, in the case of this prototype chip ($N=32$) each cell hauls $300\mu\mathrm{A}$ under normal operation conditions, but is not a bad idea to preview for a much higher consumption, let us say $800\mu\mathrm{A}$. Assume, as well, that the maximum error allowed in the power voltage, nominally

$3.3\mathrm{V}$, will be $50\mathrm{mV}$. If the power supply distribution coincides with that of the presented model —the voltage sources are tied to the ends of each row of the array, no vertical connection between horizontal power lines is considered—, each segment of the lines will have as much as $520\mathrm{m}\Omega$. This is:

$$R = R_{\square}\left(\frac{L}{W}\right) \leq 520\mathrm{m}\Omega \tag{5.92}$$

where $R_{\square}$ is the sheet resistance of the metal [Geig90], and L and W are the length and width of the metal track. For the uppermost metal layer in the CMOS process employed, the most conductive of the three, $R_{\square}$ is of $35\mathrm{m}\Omega/\square$ at room temperature, but can go up to $80\mathrm{m}\Omega/\square$ at $100\mathrm{C}$. If the length of the cells is $190\mu\mathrm{m}$, making a conservative estimation, employing the higher value for $R_{\square}$ —do not forget that direct connection of the power supply sources to the 1-D array is not realistic, thus the decay will be more pronounced— it is found that the minimum width needed to distribute the power supply voltage is approximately $30\mu\mathrm{m}$.

A similar approach is employed to derive the width of the metal lines carrying the weight signals. Now, currents are much lower, $I_o = 2.0\mu\mathrm{A}$. But the maximum error permitted is as low as $1.6\mathrm{mV}$, this is $(1/2)\mathrm{LSB}$ for an equivalent resolution of $8\mathrm{b}$ with a total signal range of $800\mathrm{mV}$. Tracing the weight lines with the same metal employed before, it is found that $R \leq 6.66\Omega$. With this, the minimum width required to maintain the accuracy levels is approximately $2.3\mu\mathrm{m}$.

In the real chip, the power supply network is 2-D. There are also vertical metal lines connecting the $M \times N$ nodes of the network. Although we have failed to find a closed form for the maximum error in a 2-D network of the described structure by analytical means, a good estimation can be made making some assumptions and with the help of some mathematical tools. First, suppose that the network is a square, $M = N$, and that V_{DD}, the power supply voltage is directly connected to every cell in the border:

$$V_{ij} = V_{\mathrm{DD}} \qquad \forall i, j \in \{1, N\} \tag{5.93}$$

It seems reasonable to think that the maximum drop will be observed at the centre of the array. An equation can be written for the inner nodes (Fig. 5.32), assuming equal resistances of the horizontal and vertical lines:

$$\begin{gathered} 4V_{ij} - V_{i-1,\,j} - V_{i+1,\,j} - V_{i,\,j-1} - V_{i,\,j+1} = -I_o R \\ \text{if} \qquad 1 < i, j < N \end{gathered} \tag{5.94}$$

This equations constitute a system of N^2 linear equation on N^2 variables, whose matricial form can be automatically computed:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & -1 & 0 & \dots & 0 & -1 & 4 & -1 & \dots & 0 & -1 & \dots & 0 \\ 0 & 0 & -1 & \dots & 0 & 0 & -1 & 4 & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & -1 & 0 & \dots & -1 & 4 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} V_{11} \\ V_{12} \\ V_{13} \\ \dots \\ V_{1N} \\ V_{21} \\ V_{22} \\ V_{23} \\ \dots \\ V_{31} \\ V_{32} \\ \dots \\ V_{NN} \end{bmatrix} = \begin{bmatrix} V_{\mathrm{DD}} \\ V_{\mathrm{DD}} \\ V_{\mathrm{DD}} \\ \dots \\ V_{\mathrm{DD}} \\ V_{\mathrm{DD}} \\ -I_o R \\ -I_o R \\ \dots \\ V_{\mathrm{DD}} \\ -I_o R \\ \dots \\ V_{\mathrm{DD}} \end{bmatrix} \tag{5.95}$$

Solving this system for the central term of the array with the help of a symbolic analysis tool [Math92], a maximum value for the error is found. The voltage drop from V_{DD} in the middle of the network as a function of the number of cells is plotted in Fig. 5.33. Black stars are the computed minima while the solid line is the best fitting 2nd-order curve. This approximation yields the following relation:

$$\left|\Delta V_{\max}\right| \approx \frac{N^2 - 2.0543N + 0.01221}{13.569} I_o R \tag{5.96}$$

It means that providing a second mesh of metal connections can reduce the voltage drop in more than a half. In particular, if the horizontal and vertical resistances are the same, the error at the centre cell is divided, approximately, by $\sqrt{3}$, —this figure is obtained by comparing Eqs. 5.95 and 5.95 for the same N.

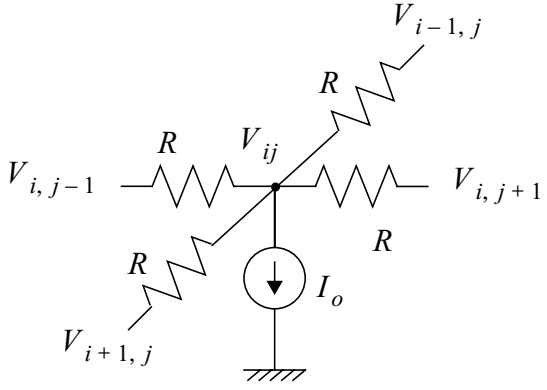


FIGURE 5.32. 2-D model for the voltage decay towards the centre of the network.

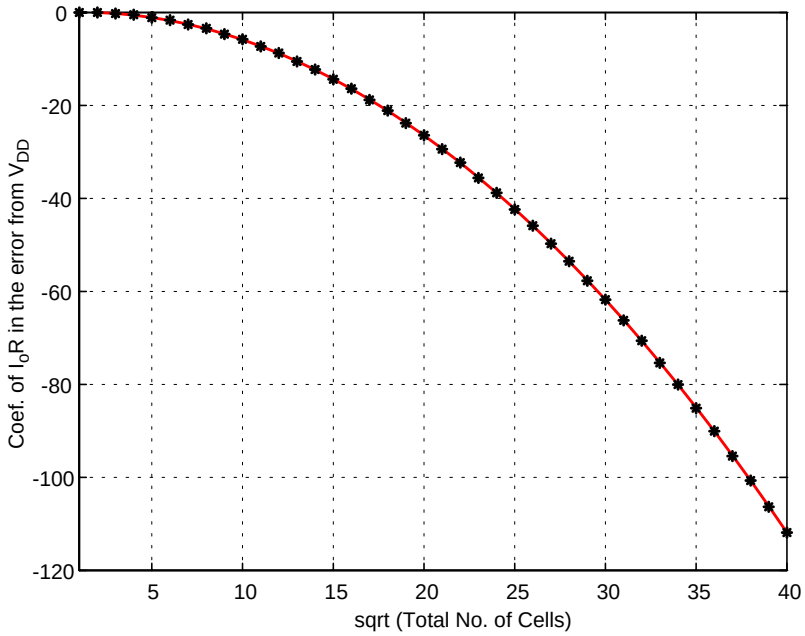


FIGURE 5.33. Voltage drop from V_{DD} experimented by the central cell in the grid.

Programming block

The programming block realizes the control of the operation of the chip. The chip has, basically, two modes of operation: the programming mode and the execution or running mode. When in programming mode, machine code instructions are passed by the user to the program storage memory. These instructions can be part of the logic program or the analog program. Protocol signals are generated on-chip to synchronize instructions download to the chip. In the execution mode, the instructions of the logic program are selected, interpreted and translated into microinstructions that are passed to the array processor to perform the corresponding operation. Physically, the memory of the programming block consists in 4 SRAM blocks of 64 words of 2 bytes each (Fig. 5.34), that constitute the logic program memory, and 16 SRAM blocks of 32 words of 2 bytes each (or 32 blocks of 32 words of one byte each, as in the figure), that store analog references, considered as the analog program.

Signals **dt<0:15>** constitute the digital data bus. It is employed in programming mode to set the contents of the memories, either the logic program memory or the analog program memory. The address for these data are separated in the block address, signals **src<0:4>**, and the word selection, signals **si<0:5>** for the logic program SRAM or signals **sp<0:4>** for the analog program. The block address, **src<0:4>**, selects one out of the 20 memory blocks (4 for the logic program and 16 for the analog). In the execution mode, buses **dt<0:15>** and **src<0:4>** are silent. Meanwhile **si<0:5>** and **sp<0:4>** select the corresponding logic and analog instructions to be passed to the processing array. One instruction of the logic program memory consists in 64bits, this is 8 bytes. Table 5.1 enumerates and describes the different digital signals that comprise the instruction of

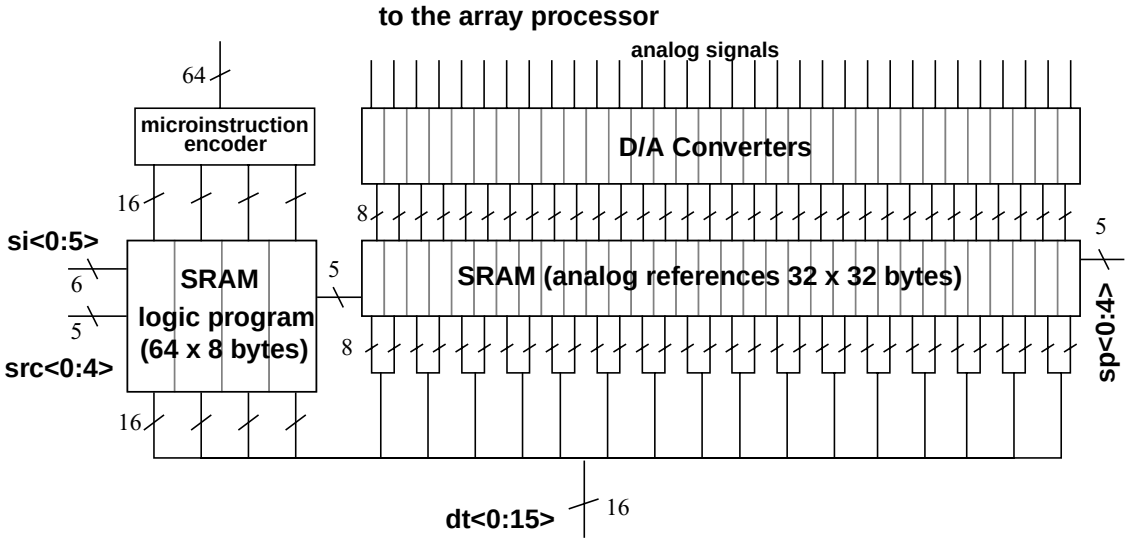


FIGURE 5.34. Block diagram of the program memory and program control

the logic program. These signals constitute the internal microcode, performing different functions within the chip. Some enable access and communication between the different blocks in the cell, some others trigger analog operations and dynamics, etc.

src#	bit #	Signal name	act.	null	Description
10011	0	UAM	1	0	Enable I/O access to the LAMs
“	1	AMR	1	0	Enable reading access to the LAMs
“	2	AMW	1	0	Enable writing access to the LAMs
“	3	sm0	1	0	Enable access to one of the four analog memories locations: LAM0, LAM1, LAM2 and LAM3, respectively
“	4	sm1	1	0	
“	5	sm2	1	0	
“	6	sm3	1	0	
“	7	feed	1	0	Enable the error correction capacitor.
“	8	CLOOP	1	1	Close LLM latches loops
“	9	RLLU	1	0	Read LLU results
“	10	T00	1	0	Truth table of the LLU (see Fig. 5.29)
“	11	T01	1	0	
“	12	T10	1	0	
“	13	T11	1	0	
“	14	edo	1	0	Enables data input to the LLU
“	15	edi	1	0	Enables data output from the LLU
10010	0	conVx0	1	0	Connects the state node to Vx0

TABLE 5.1. CACE1K internal microcode

src#	bit #	Signal name	act.	null	Description
“	1	conVxmax	1	0	Connects state node to Vxmax
“	2	conVxmin	1	0	Connects state node to Vxmin
“	3	dcco	1	1	Deviates current conveyor output when not running the CNN dynamics
“	4	lazo	1	0	Closes CNN cores feedback loops
“	5	frz	0	1	Enables the use of the freezing mask
“	6	Vphi1	1	0	Timing of the S^3I current memory
“	7	Vphi2	1	0	
“	8	Vphi3	1	0	
“	9	Soff	0	1	Short-circuits the inputs of the critical OTAs
“	10	Moff	0	1	Enables access to the critical OTAs autozeroing capacitors.
“	11	MoffON	0	0	Enables OTAs autozeroing strategy
“	12	sana1	1	0	Connects state node of the scalable time-constant layer (layer 1) to the cell’s analog bus
“	13	sbin1	1	0	Connects the binarized output of layer 1 to the cell’s digital bus
“	14	eai1	1	0	Enables the analog input to layer 1
“	15	eao1	1	0	Enables the analog output off layer 1
10001	0	sz1	1	0	Connects the limiter output to the bias term capacitor in layer 1
“	1	su1	1	0	Connects the limiter output to the input signal capacitor in layer 1
“	2	sx1	1	0	Connects the limiter output to the state capacitor in layer 1, must be active to close the CNN loop and run the netwrok dynamics.
“	3	bt0	1	1	Codify the curent scaling factor. All active means 1:1 mirroring, each LSB means 1/16 of the original current. All inactive means that the output of the mirror is only 1/16 of the original current.
“	4	bt1	1	1	
“	5	bt2	1	1	
“	6	bt3	1	1	
“	7	sana2	1	0	Connects state node of the fixed time-constant layer (layer 2) to the cell’s analog bus
“	8	sbin2	1	0	Connects the binarized output of layer 2 to the cell’s digital bus
“	9	eai2	1	0	Enables the analog input to layer 2
“	10	eao2	1	0	Enables the analog output off layer 2
“	11	sz2	1	0	Connects the limiter output to the bias term capacitor in layer 2
“	12	su2	1	0	Connects the limiter output to the input signal capacitor in layer 2
“	13	sx2	1	0	Connects the limiter output to the state capacitor in layer 2, must be active to close the CNN loop and run the netwrok dynamics.
“	14	Not employed			

TABLE 5.1. CACE1K internal microcode

src#	bit #	Signal name	act.	null	Description
“	15	“			
10000	0	b1c1	1	0	Enable different boundary conditions for the layers: boundary at Vxmin (00), at Vx0 (01), at Vxmax (10), at Vx of the nearest neighbour (11).
“	1	b1c0	1	1	
“	2	b2c1	1	0	
“	3	b2c0	1	1	
“	4	ColS	1	0	Connects to the column bus.
“	5	ENIO	1	0	Enables image I/O processes
“	6	ENRW	1	0	Activates reading or writing images
“	7	PRCH	1	0	Precharges global gates computation
“	8	RGG	1	0	Reads glogbal gates results
“	9	Not employed			
“	10	“			
“	11	“			
“	12	“			
“	13	“			
“	14	“			
“	15	“			

TABLE 5.1. CACE1K internal microcode

Each instruction of the analog program consists in 32 bytes, what are 256bits. Each byte codifies one analog voltage reference. These codes are passed to the bank of D/A converted that interfaces the analog program memory with the processing array. Table 5.2 shows the correspondence between the memory addresses and the internal analog references.

src#	dt #	Signal name	dt #	Signal name	Description
00000	0-7	wz1	8-15	wz2	weigh the bias image
00001	0-7	wu1	8-15	wu2	weigh the input image
00010	0-7	wxol1	8-15	wxol2	weigh the connection to the other layer
00011	0-7	wxid1	8-15	wxid2	connection to the bottom-right corner
00100	0-7	wxic1	8-15	wxic2	connection to the bottom-central cell
00101	0-7	wxii1	8-15	wxii2	connection to the bottom-left corner
00110	0-7	wxcd1	8-15	wxcd2	connection to the centre-right
00111	0-7	wxcc1	8-15	wxcc2	self-feedback
01000	0-7	wxci1	8-15	wxci2	connection to the centre-left
01001	0-7	wxsd1	8-15	wxsd2	connection to the upper-right corner
01010	0-7	wxsc1	8-15	wxsc2	connection to the upper-central cell
01011	0-7	wxsi1	8-15	wxsi2	connection to the upper-left corner
01100	0-7	Vxmin	8-15	Vn0	Analog references: 0.6V and 0.4V

TABLE 5.2. Analog program instruction

src#	dt #	Signal name	dt #	Signal name	Description
01101	0-7	Vwmin	8-15		Analog reference: 2.15V
01110	0-7	Vx0	8-15	Vxmax	References: 1.0V and 1.6V
01111	0-7	Vw0	8-15	Vwmax	References: 2.55V and 2.95V

TABLE 5.2. Analog program instruction

Voltages are codified by the 7MSBs of each byte, while the last one encodes the sign. The first 12 blocks (first 12 rows of Table 5.2) are connected to D/A converters with 800mV full-scale centred in 2.55V, nominally. These blocks store the weights of the synapses. The last 4 blocks drive D/A converters with 3.3V full-scale centred in 1.65V. It is important to remember here that some of the synapses have been resized to hard-code their strength, in order to reduce the prescribed resolution for the weight signals for a set of applications (see Chapter 3). This means that the template elements introduced at the beginning, which are the theoretical weight values $V_{W\text{theo}}$, must be scaled down by the synapse strength w_{stre}, in order to operate properly:

$$V_W = \alpha \frac{V_{W\text{theo}}}{w_{\text{stre}}} + V_{\text{off}} \tag{5.97}$$

where α and V_{off} are the scaling factor and voltage shifting necessary to implement the theoretical template values in a specific realization, i. e. $V_{W\text{theo}} = 0\text{V}$ corresponds to a V_W of 2.55V, and therefore $V_{\text{off}} = 2.55\text{V}$ in this implementation. Also, suppose the maximum value for $V_{W\text{theo}}$ is 4V, as the maximum V_W attainable is 2.95V (400mV above 2.55V) then the scaling factor, α, is 10, if the synapse strength is unity. In this chip, synapses associated with signals **wxcc1**, **wxcc2**, **wu1**, **wu2**, **wxol1**, **wxol2** have a strength of 3, while the rest have unity strength.

D/A converters and weight buffers

The D/A converters employed here are of an inherently monotonic type [Jaeg82]. As depicted in Fig. 5.35, this circuit consists in a long string of equally-valued resistors (apart from those at the extremes) running from the higher, $V_{\text{REF+}}$, to the lower voltage reference, $V_{\text{REF–}}$. This string is tapped at equally-spaced points. The access to this points is controlled by a tree of analog switched driven by the outputs of some decoding logic. This is, the 8b codifying each analog reference are the inputs of two binary-to-one-hot 4:16 decoders, one for the 4 less significant bits and the other for the most significant. These decoders generate signals $s_{L_0}, s_{L_1}, \ldots, s_{L_{15}}$ and $s_{H_0}, s_{H_1}, \ldots, s_{H_{15}}$ which control the analog switches. By using these decoders, the total number of transistors is not reduced at all. If a complete binary tree is implemented, 256 transistors are required for the first level, 128 for the second, etc. This makes a total of 512 transistors, assuming that the complementary bits are available at the output

of the memory blocks. For the scheme depicted in Fig. 5.35, we need $256 + 16 = 272$ transistors for the analog switches tree, plus 128 transistors for each 4:16 decoder. This makes a total of 528 transistors. However, the advantage of using this scheme is found in the resistance of the path from the resistor chain to the voltage follower input. The larger the resistance, the slower the settling.

Monotonicity is assured by construction, as every tap points to a higher voltage that the previous one. Differential nonlinearity in this circuit is introduced by the mismatch between the resistors, therefore, they must be sized to avoid important impairities in the converter steps. Integral nonlinearity is not a problem because it can be corrected by software.

The outputs of the D/A converters need buffering to be transmitted to the array processor. Those voltage references driving high-impedance nodes, as the gates of MOS transistors, whose impedance can be considered infinite at low-frequencies, do not require extra driving force other than that afforded by the voltage followers at the converters' output. Unfortunately, this is not the situation for the weight signals. Weights, as reported in the description of the single-transistor synapse, have to be transmitted to low-impedance nodes, 1024 sources of MOS transistors in parallel. The loading impedance, R_L, results unbearable for the voltage buffer especially if the resistance of the long tracks of metal conveying the signal from one end of the die to the other is taken into account. Basically, the resistance of the metal tracks add up to the buffer output resistance, thus boosting the voltage divi-

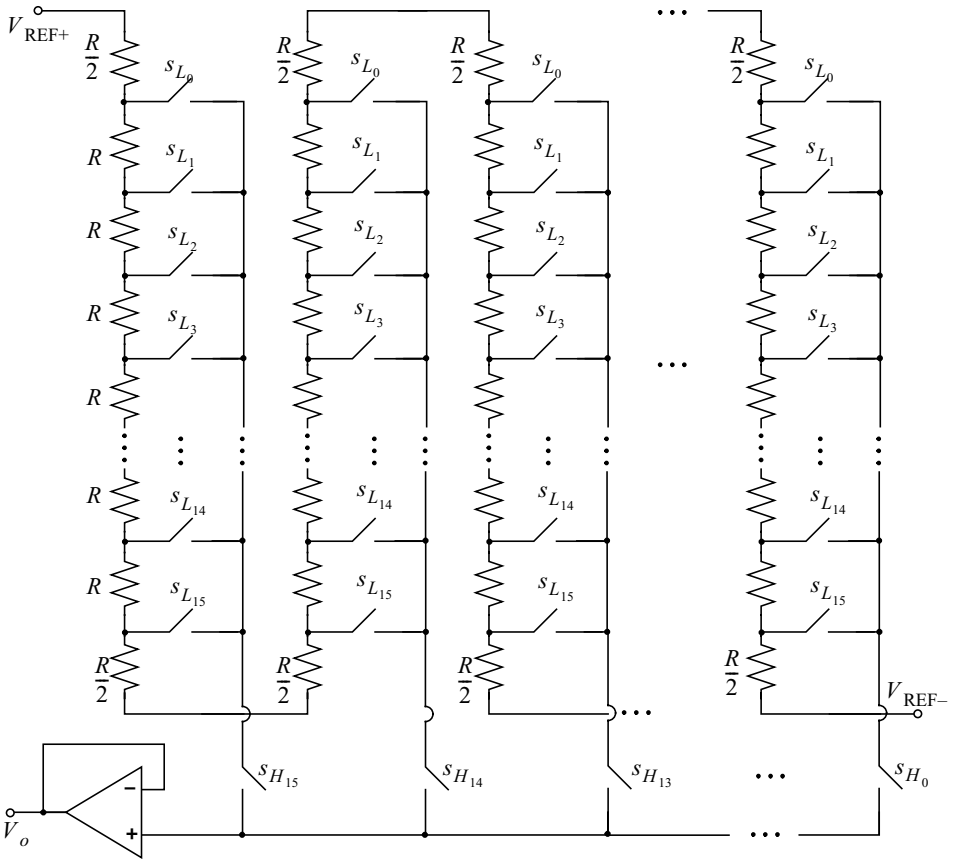


FIGURE 5.35. Inherently monotonic 256-level (8bits) D/A converter.

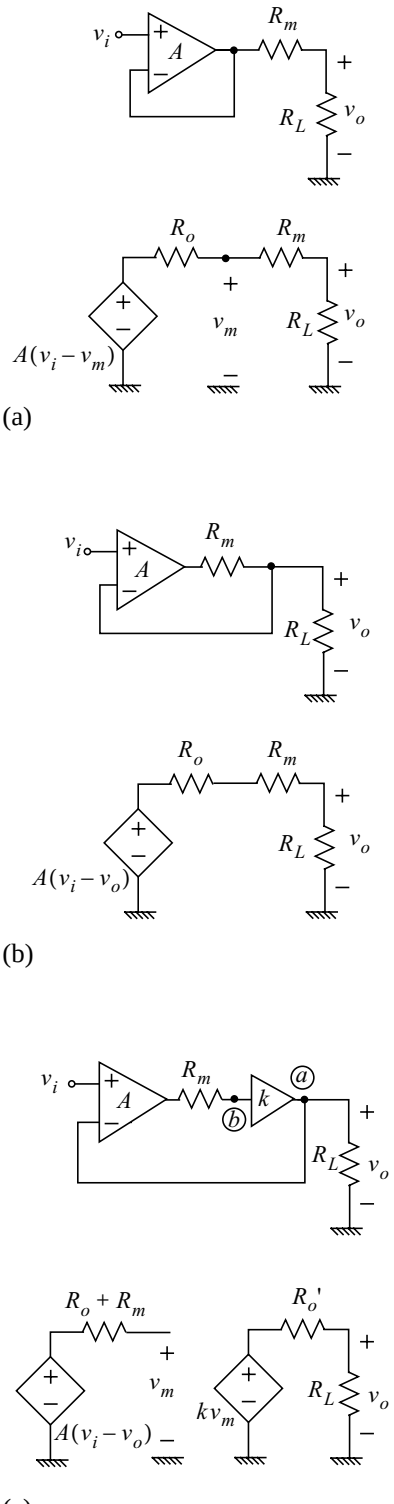


FIGURE 5.36. Weights buffering alternatives and low-frequency small signal models

sion. This is the situation in Fig. 5.36(a). Computing the output impedance of the buffer-plus-metal-tracks circuit, it adds up to:

$$Z_o = R_m + \frac{R_o}{1 + A} \tag{5.98}$$

hence, despite the fact that the output resistance of the amplifier is greatly attenuated by feedback, the resistance of the metal tracks, represented here by R_m, makes Z_o to be as large as the actual R_L. The resistance of the metal tracks can be as high as $1\text{k}\Omega$ while the resistance of one of the p-type MOS transistors operating in the ohmic region employed can be about $1\text{M}\Omega$. 1024 of them in parallel makes an R_L of $1\text{k}\Omega$, approximately, thus halving the dynamic range of the signals, and therefore losing one bit of resolution in the weights.

This error can not be afforded so a different scheme must be employed for the buffers of the weight signals. For instance, in the circuit in Fig. 5.36(b) is the actual output voltage the one that is fed back to the amplifier, causing the output impedance seen by the load to be:

$$Z_o = \frac{R_o + R_m}{1 + A} \tag{5.99}$$

what enhances the performance of the buffer. But still, propagation of signals through metal tracks carrying strong current intensities may end in appreciable disparities between the voltages transmitted to different points along the metal tracks, unless they are made inconceivably wide. In order to avoid this, we have adopted the solution proposed in [Liña98], in which voltage buffers are located nearer to the cell array. Working in parallel, they can be considered as a voltage amplifier with gain k, a high input impedance and a rather low output impedance (Fig. 5.36(c)). The output impedance of the complete circuit, at low-frequencies, is given by:

$$Z_o = \frac{R_o'}{1 + kA} \tag{5.100}$$

what certainly reduces any possible loading effect. The design challenge is to avoid the loss of phase margin in the feedback loop, being the dominant pole, associated with the output node ⓐ, and the second pole, which is associated with node ⓑ, too close to operate without compensation.

Extra capacitance must be added accordingly to node ⓐ to avoid instability, at the expense of a reduction on the circuit speed.

I/O interface

The last major subsystem of the prototype is the I/O interface. This circuit is provided for the acquisition and delivery of image samples, which must be analog voltages ranging from $0.6\mathrm{V}$ to $1.4\mathrm{V}$, nominally. The collection of 32×32 image samples, that will be acquired or delivered on the same batch, are transmitted through a serial channel but passed in parallel to the 32 columns of the array. This is achieved by the circuit structure depicted in Fig. 5.37(a). It consists in 32 sample and hold circuits, one for each column of the array, connected concurrently to the serial I/O channel. Each S/H circuit, represented by the schematic in Fig. 5.37(b), consists in 2 sample and hold stages —like those employed for the local analog memories in the array— and several transmission gates. When acquiring an image, **ENRW** = 1. The serial I/O channel connects the input pad to the input nodes of the sample and hold stages. Then, at $10\mathrm{Ms/s}$ (which is the intended acquisition speed, that was proven possible by the aRAM prototype), 32 samples of the signal are stored in the first row of S/H circuits by activating alternatively signals $\mathbf{SH1A_i}$ and $\mathbf{SH1W_i}$. Once the fist 32 samples are acquired, the following 32 samples of the input signal are stored in the second row of S/H circuits, by activating $\mathbf{SH2A_i}$ and $\mathbf{SH2W_i}$ for the 32 S/H stages alternatively. At the same time, all the signals $\mathbf{SH1R_i}$ corresponding to the first row of S/H's are activated together, thus uploading the stored samples to the first row of the array. This is realized during $32 \times 100\mathrm{ns} = 3.2\mu\mathrm{s}$, which is enough time for the driving capabilities of the S/H amplifiers to update the voltage at the prescribed local memory in the cells of the first row of the array. By the time the second row of S/H circuits is done with the acquisition of the second batch of 32 samples of the input, the first row starts acquiring the following 32 samples, while the second S/H row passes the stored sample to the second row of the array of cells. This process continues until the last row of the array is updated with the information from the

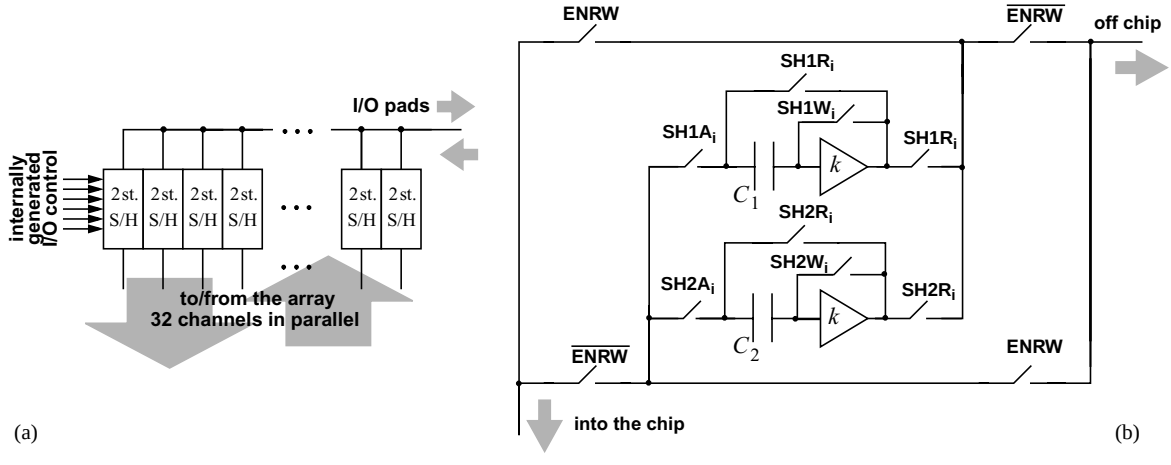


FIGURE 5.37. (a) Serializing-deserializing I/O interface and (b) 2-stage circuit for sample and hold.

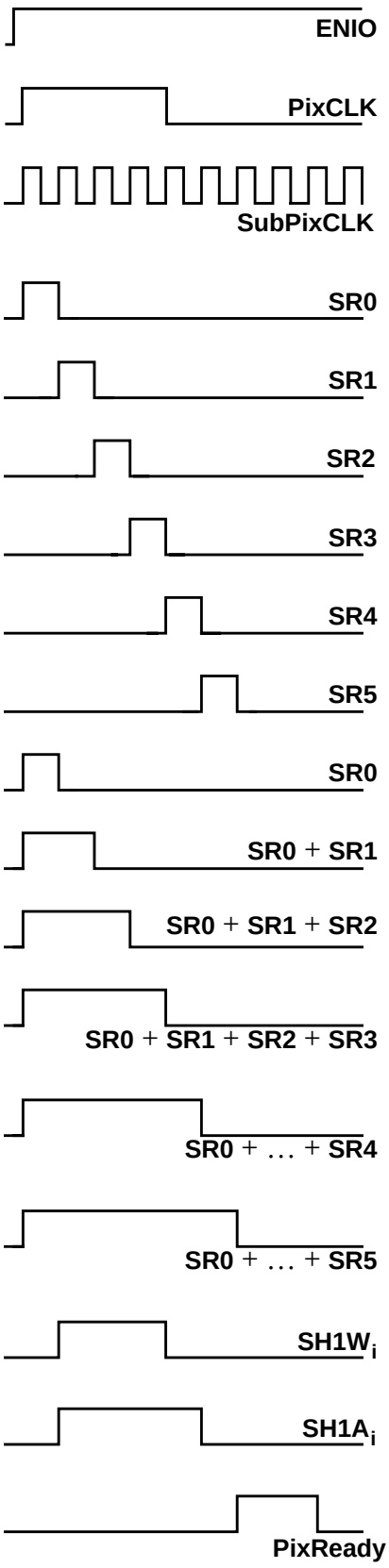


FIGURE 5.38. Internal generation of separated rising and trailing edges for the S/H control.

new image. For the delivery of the output image, the process is inverted, **ENRW** = 0, then the S/H circuits first read in parallel what is transmitted by the rows of the array, and then deliver the acquired samples of the output one by one, through the I/O channel that now is connected to an output buffer to send the voltages off-chip at 10Ms/s.

It must be regarded that most of the signals controlling the I/O processes are generated on-chip, hence, for the user, external control is reduced to follow a simple protocol. Let us draw an sketch of these synchronization signals. First of all, there is an internal clock, made up from a ring of inverters, with voltage controlled frequency, constituting a non-harmonic VCO. This is known as the sub-pixel clock, generating signal **SubPixCLK**. From the program memory, bits 54th and 55th of the logic program instruction indicate the initiation of an I/O process (the 54th bit corresponds to signal **ENIO**, as depicted in Table 5.1) and specify whether a 32×32-pixel image is to be acquired or delivered by the chip (what is done by signal **ENRW**, 55th bit of the logic program instruction). Then, when an instruction containing **ENIO** = 1 is selected, the I/O interface waits for a rising edge of an external signal named **PixCLK**, referred as the pixel clock. Once it occurs, an internal process controlled by **SubPixCLK** generates the appropriate internal signals (**SH1A$_i$**, **SH1W$_i$**, **SH1R$_i$**, **SH2A$_i$**, **SH2W$_i$** and **SH2R$_i$**, and the appropriate row selection) to acquire or deliver one voltage sample. When this is achieved, the I/O interface generates a pulse named **PixReady**, that must be sensed by the user, that means that the system is ready to receive, or send, the following pixel sample. The critical steps in the design of this scheme are two:

- overlapping in the S/H selection signals must be avoided. Guard times must be provided that do not rely on the internal delays that are not controlled precisely.
- control of the local memories access must be passed to the I/O interface while acquiring or delivering an image.

Therefore, with the activation of **ENIO** and the arrival of the first **PixCLK** rising edge, a shift-register clocked by **SubPixCLK** passes a pulse from the leftmost stage, say **SR0**, to the rightmost, **SR5** in Fig. 5.38, permitting the generation of synchronized edges that will

constitute the rising and trailing edges of the S/H control signals. In the figure, the registers outputs are combined by means of logic OR gates, generating pulses of increasing widths. These pulses can be combined as well to obtain the desired control signals. For instance,

$$\mathbf{SH1W_i} = \overline{\mathbf{SR0}}(\mathbf{SR0} + \mathbf{SR1} + \mathbf{SR2} + \mathbf{SR3}) \quad (5.101)$$

generates signal $\mathbf{SH1W_i}$, that controls the feedback path in the sample-and-hold circuit. At the same time:

$$\mathbf{SH1A_i} = \overline{\mathbf{SR0}}(\mathbf{SR0} + \ldots + \mathbf{SR4}) \quad (5.102)$$

allows generating $\mathbf{SH1A_i}$, which controls the access to the memory capacitor. The first signal, $\mathbf{SH1W_i}$, must fall necessarily before signal $\mathbf{SH1A_i}$, for the bottom-sampling operation to perform properly. The internal generation and separation of the signal edges prevent uncontrolled delays to alter the precedence between signals, almost independently of the clock frequency at which these circuits are operated. At the end of the count realized by the shift-register, a **PixReady** pulse is generated, notifying the user that the next voltage sample can be recorded in the following capacitor of the S/H battery.

Then, samples are first acquired and stored on top of each column of the array. When a complete row of samples is memorized (32 pixels), they are transmitted to the corresponding row of the array, while the next 32 samples are being acquired and stored in the second row of S/H's. Selection of the appropriate column and row is performed by two 5b counters that are re-started with the first pulse of the pixel clock. The counters' output pulses are employed internally in a similar manner as the already described shift-register output. By means of the appropriate logic, they generate the control signals for accessing the local analog memories (LAMs), whose control has been passed to the I/O interface during images up/downloading. Concerning downloading, the processes are inverted, the S/H batteries acquire the samples in parallel from each row of the array at a time and deliver them one by one through the serial I/O channel.

The combinational and sequential logic circuits employed to implement the I/O interface has been designed full-custom, in order to allow as much integration with the processing array as possible. Using a library of cells would not have permitted the intricate routing employed to tailor the I/O control.

5. 3. Prototype chip data and simulations

Following the directions detailed in the previous sections, a prototype chip, named CACE1K, has been designed and fabricated in a standard CMOS technology. By this time, the device is being tested to evaluate its performance. Prototype data are presented here together with some simulation results that illustrate the expected performance aspects of the chip.

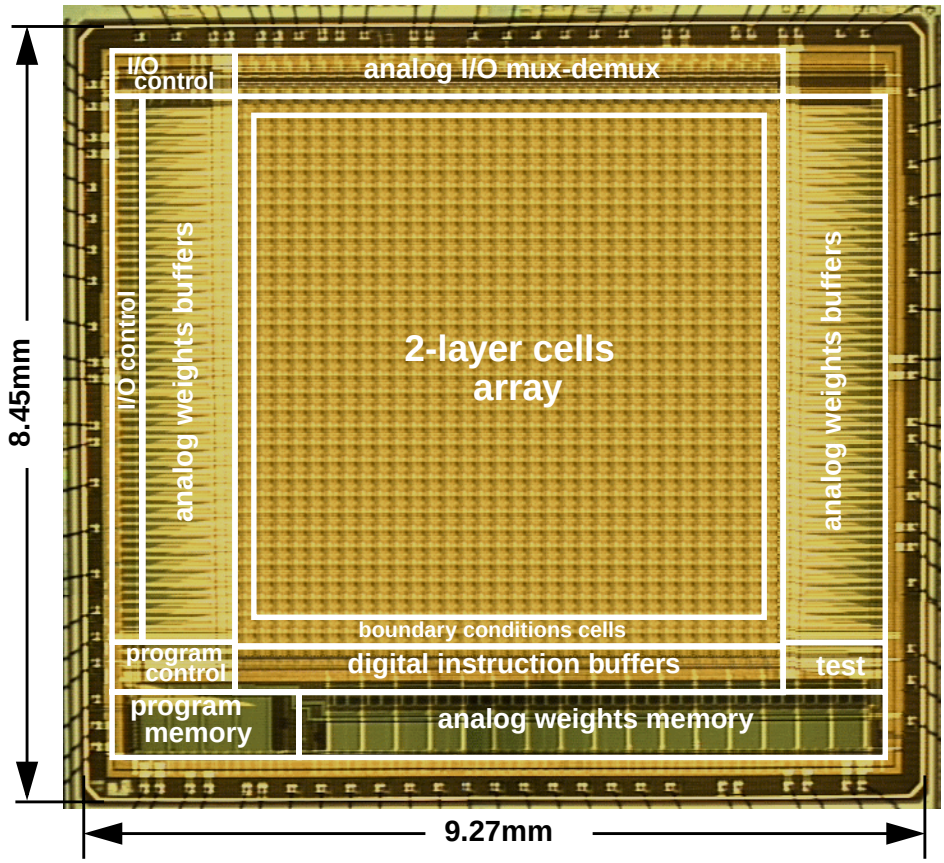
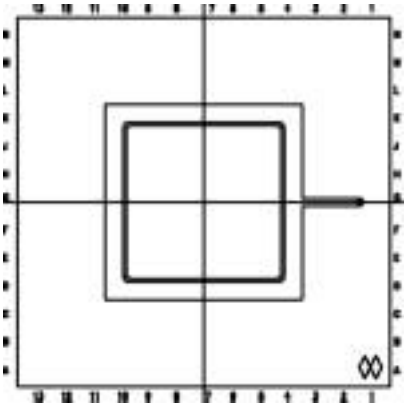


FIGURE 5.39. Microphotograph pf the prototype chip.

5. 3. 1. CACE1K data and pinage

Chip data

A prototype chip has been designed and fabricated in a standard $0.5\mu m$ CMOS technology with single-poly and triple-metal layers. Fig. 5.39 displays a microphotograph of the CACE1K chip. It contains a central array of 32×32 cells of the type described formerly, this is, with two coupled CNN cores with programmable dynamics, LLMs and LAMs and a LLU. Surrounding the array, a ring of boundary cells, implementing the contour conditions for the CNN dynamics, is found, together with the necessary buffers to transmit digital instructions and analog references to the array. On the lower part of the chip (see Fig. 5.39), the program control and memory blocks can be found. The last major subsystem is the I/O interface including S/H batteries, decoders, counters and different sequential logic. The whole system fits in 9.27×8.45 sq. mm., including the ring of bonding pads. Without pads, the total area is 8.77×7.94 sq. mm., this includes the CNN array and the necessary overhead to make it work, plus the program memory and control and the I/O interface. The array of CNN cells alone occupies 5.98×5.83 sq. mm., which is, roughly, the 50% of the total area of the chip. The resulting cell density, excluding circuits outside the array, is 29.37cells/mm^2. In order to cautiously handle this data, it is important to notice that the area occupied by the cell array scales line-



	13	12	11	10	9	8	7	6	5	4	3	2	1	
N	VSSA	VDDD	VDDP	dt13	dt11	dt8	dt7	dt4	dt1	VSSP	VSSD	VSSA	VSSD	N
M	NC	VDDA	VSSD	VSSP	dt12	dt9	dt5	dt3	dt0	VDDP	VDDD	VDDA	PBready	M
L	sp3	sp4				dt10	dt6	dt2				VDDD	si5	L
K	sp1	sp2										si4	si3	K
J	VSSD	sp0										si2	si1	J
H	GGNAND	VDDP	VDDD								si0	CLKext	CLKtest	H
G	VDDA	GGNOR	VSSP								VSSP	VSSA	VDDP	G
F	VSSA	TREFc0	TREFc1								CLKsel	CLKbias	VDDA	F
E	TREFc2	TREFc3										PBreset	prg	E
D	TREFc4	TREFstrb										VSSA	PBrw	D
C	TREFout	NC				src3	ENG	VSSP			NC	VSSD	VDDA	C
B	VSSA	dt15	VSSA	VDDP	VSSD	src2	src0	ImgOut	ImgIn	VSSD	PixReady	NC	VDDD	B
A	dt14	VDDA	VSSP	VDDD	src4	src1	VDDA	VSSA	VDDP	VDDD	VSSA	VDDA	PixCLK	A
	13	12	11	10	9	8	7	6	5	4	3	2	1	

FIGURE 5.40. Pinage of the packaged prototype chip.

arly with the total number of cells, what is not the case of the overhead circuitry, which tends to be a smaller fraction of the total chip size as long as the number of cells rises.

The prototype has been packaged in a PGA-100 ceramic capsule following the pin map depicted in Fig. 5.40. The bonding diagram sent to the manufacturer is shown in Fig. 5.41. Expected performance data are $8b$ accuracy on the weights, $7\text{-}8b$ accuracy on the image signals, $10Ms/s$ I/O rates, and a time constant for the CNN dynamics under $100ns$. Tests are now carried on the confirmation of these prescribed features.

The following table describes the pin assignment, the prototype pad names and the nature of the signals that have to be attached to them:

Finger #	Pin #	Pad name	Description
1	B2	CavGND	Gound connection (0V) of the cavity of the package
2 3	B1 C2	VDDD VSSD	Digital **power supply** and **ground**: dc power sources of 3.3V and 0V. External capacitors recommended.
4 5	C1 D2	VDDA VSSA	Analog **power supply** and **ground**: dc 3.3V and 0.0V respectively. External capacitor recommended.
6	D1	PBrw	Programming block **read/write**: selects program memory reading, logic-1 (3.3V), or memory writing, logic-0 (0V).
7	E2	PBreset	Programming block **reset**: initiates every type of access to the program memory.

TABLE 5.3. Pin assignment of the CACE1K prototype and signals description

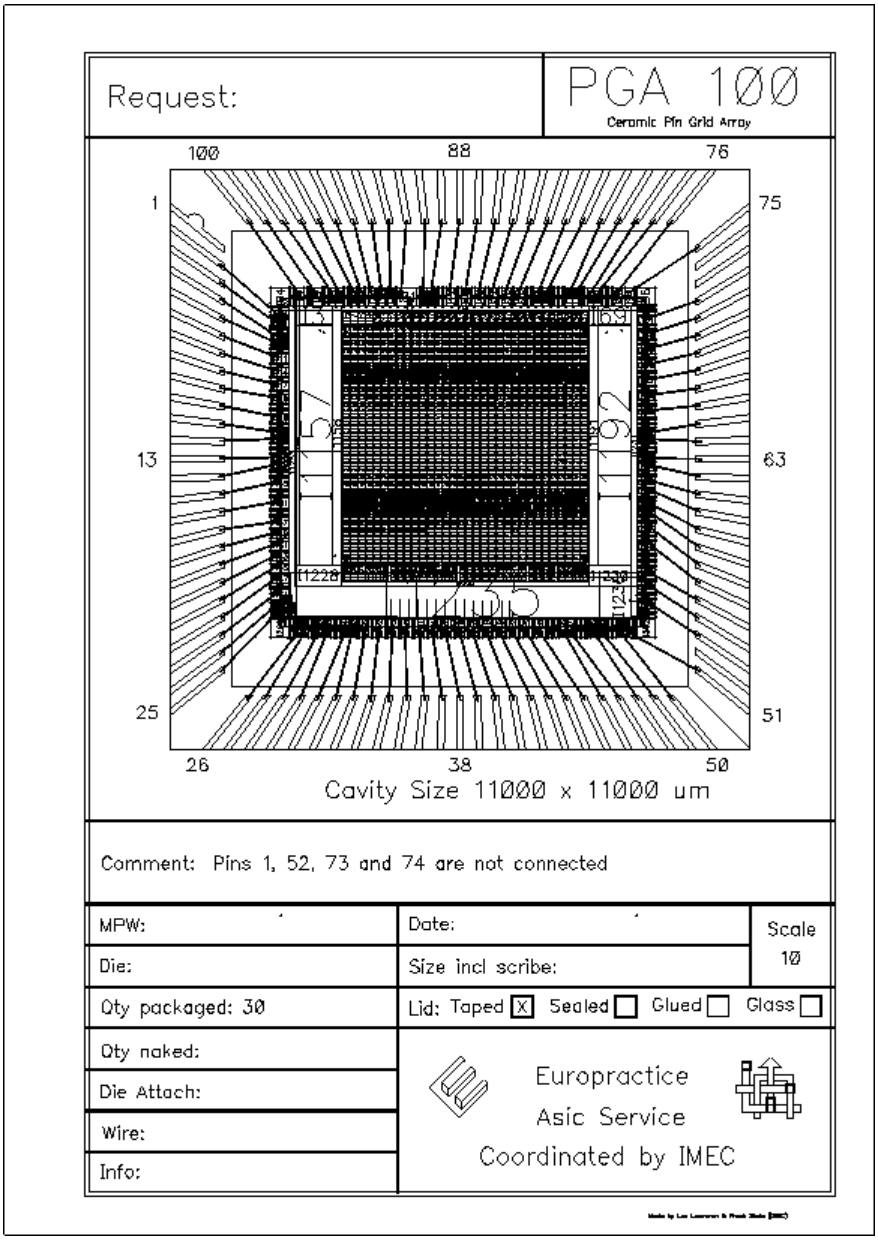


FIGURE 5.41. Bonding diagram of the CACE1K chip.

Finger #	Pin #	Pad name	Description
8	E1	prg	Programming **mode selection**: 1 programming, 0 processing.
9	F3	CLKsel	Clock **selector**: 0 for the internal clock, 1 for the external source. Pulled down by default.
10	F2	CLKbias	Internal clock **bias**: reference voltage for the internal oscillator. Designed to operate between 0V (which corresponds to 125.8MHz) and 2.5V (for 2.5MHz).

TABLE 5.3. Pin assignment of the CACE1K prototype and signals description

Finger #	Pin #	Pad name	Description
11 12	F1 G2	VDDA VSSA	Analog **power supply** and **GND**: dc 3.3V and 0.0V respectively. External capacitor recommended.
13 14	G3 G1	VSSP VDDP	Padring **GND** and **power supply**: dc 0.0V and 3.3V respectively. External capacitor recommended.
15	H1	CLKtest	Clock **test probe**: buffered clock output for test purpose.
16	H2	CLKext	External clock **input**: if required.
17 18 19 20 21 22	H3 J1 J2 K1 K2 L1	si0 si1 si2 si3 si4 si5	Digital **instruction address**: used for reading/writing the program memory. It points to one out of the 64 sub-blocks of 2 bytes of the selected digital instruction memory block.
23	M1	PBready	Prog. block operation **termination flag**: protocol signal that communicates to the outside of the chip that the operation with the prog. block was fulfilled.
24 25	L2 N1	VDDD VSSD	Digital **power supply** and **GND**: dc power sources of 3.3V and 0V. External capacitors recommended.
26 27	M2 N2	VDDA VSSA	Analog **power supply** and **GND**: dc 3.3V and 0.0V respectively. External capacitor recommended.
28 29	M3 N3	VDDD VSSD	Digital **power supply** and **GND**: dc power sources of 3.3V and 0V. External capacitors recommended.
30 31	M4 N4	VDDP VSSP	Padring **power supply** and **GND**: dc 3.3V and 0.0V respectively. External capacitor recommended.
32 33 34 35 36	M5 N5 L6 M6 N6	dt0 dt1 dt2 dt3 dt4	Digital **data bits**: first 5 bits of a digital or codified analog instruction to be stored in the program memory.
37 38 39 40 41 42 43 44 45	M7 L7 N7 N8 M8 L8 N9 M9 N10	dt5 dt6 dt7 dt8 dt9 dt10 dt11 dt12 dt13	Digital **data bits**: the following 9 bits of a digital or codified analog instruction to be stored in the program memory.
46 47	M10 N11	VSSP VDDP	Padring **GND** and **power supply**: dc 0.0V and 3.3V respectively. External capacitor recommended.
48 49	N12 M11	VDDD VSSD	Digital **power supply and GND**: dc power sources of 3.3V and 0V. External capacitors recommended.
50 51	N13 M12	VSSA VDDA	Analog **GND** and **power supply**: dc 0.0V and 3.3V respectively. External capacitor recommended.
52	M13		Not Connected

TABLE 5.3. Pin assignment of the CACE1K prototype and signals description

Finger #	Pin #	Pad name	Description
53 54 55 56 57	L12 L13 K12 K13 J12	sp4 sp3 sp2 sp1 sp0	Analog **instruction address**: used for reading/writing the program memory. It points to one out of the 32 sub-blocks of 2 bytes of the selected digital instruction memory block.
58 59	J13 H11	VSSD VDDD	Digital **GND** and **power supply**: dc power sources of 0.0V and 3.3V. External capacitors recommended.
60	H12	VDDP	Padring **power supply**: dc 3.3V. External capacitor to GND.
61	H13	GGNAND	**Output** of the global NAND gate.
62	G12	GGNOR	**Output** of the global NOR gate.
63	G11	VSSP	Padring **GND**: dc 0V.
64 65	G13 F13	VDDA VSSA	Analog **power supply** and **GND**: dc 3.3V and 0.0V respectively. External capacitor recommended.
66 67 68 69 70	F12 F11 E13 E12 D13	TREFc0 TREFc1 TREFc2 TREFc3 TREFc4	Analog references **test code**: selects which signal will be multiplexed to the test output pad. Pulled up by default.
71	D12	TREFstrb	Analog references **test strobe**: enables the analog reference signals test. Pulled down by default.
72	C13	TREFout	Analog references **test probe**: for sensing the actual internally generated analog references.
73	B13		Not Connected
74	C13		Not Connected
75 76	A13 B12	dt14 dt15	Digital **data bits**: last 2 bits of a digital or codified analog instruction to be stored in the program memory.
77 78	A12 B11	VDDA VSSA	Analog **power supply** and **GND**: dc 3.3V and 0.0V respectively. External capacitor recommended.
79 80	A11 B10	VSSP VDDP	Padring **GND** and **power supply**: dc 0.0V and 3.3V respectively. External capacitor recommended.
81 82	A10 B9	VDDD VSSD	Digital **power supply** and **GND**: dc power sources of 3.3V and 0V. External capacitors recommended.
83 84 85 86 87	A9 C8 B8 A8 B7	src4 src3 src2 src1 src0	Program **memory block address**: Selects one out of the 24 memory blocks. The first 4 are for digital instructions (the logic program) and the rest are for coded analog weights and references (the analog program).
88	C7	ENG	**Global enable** signal: pulled up by default.
89 90	A7 A6	VDDA VSSA	Analog **power supply** and **GND**: dc 3.3V and 0.0V respectively. External capacitor recommended.
91	B6	ImgOut	Analog **output pin:** buffered output gray-scale image samples ranging from 0.6V to 1.4V nominally.
92 93	C6 A5	VSSP VDDP	Padring **GND** and **power supply**: dc 0.0V and 3.3V respectively. External capacitor recommended.
94	B5	ImgIn	Analog **input pin:** gray-scale image samples input. Must be restricted to [0.6V,1.4V] interval, nominally.

TABLE 5.3. Pin assignment of the CACE1K prototype and signals description

Finger #	Pin #	Pad name	Description
95 96	A4 B4	VDDD VSSD	Digital **power supply** and **GND**: dc power sources of 3.3V and 0V. External capacitors recommended.
97 98	A3 A2	VSSA VDDA	Analog **GND** and **power supply**: dc 0.0V and 3.3V respectively. External capacitor recommended.
99	B3	PixReady	Sample up/downloading **termination flag:** protocol signal that transmit to the outside the fulfillment fo the acquisition or delivery of one pixel.
100	A1	PixCLK	Pixel **clock:** triggers the acquisition of one pixel of the in put image, or the delivery of a pixel of the output.

TABLE 5.3. Pin assignment of the CACE1K prototype and signals description

5. 3. 2. Simulation results

Single-cell evolution

In order to illustrate the operation of the circuit, let us display some simulations results concerning, in the first place, the evolution of one single cell of the array. In this process, controlled by multiple signals constituting the internal microcode of the array processor, the cell memories are updated and then the CNN is initialized and calibrated for the network dynamics to be run. After the evolution of the cell layers, the result is stored in a LAM for further processing with the programmable local logic. Thus, Fig. 5.42 displays the waveforms of all the signals that drive and configure the single cell in this process. To find out what their particular task is you can refer yourself to Table 5.1. Then Fig. 5.43 plots the evolution of the two state variables of the cell and the I/O node. It can be seen how the different steps of the process reflect in the state variable voltage.

Reduced network simulation

For illustrating the cooperative behaviour of the network, a simulation showing the propagation of signals from one layer to the other has been realized. Fig. 5.44 shows how a perturbation in a pixel of a row of 8 cells (in particular, the 1st layer of the cell C_{14}) is rapidly extended to the other layer and propagates in both towards the two ends on the line.

Finally, another propagation template has been programmed in a 4×4-cells network. This time, black pixels are projected towards the rightmost side of each row of cells. The two layers are now uncoupled and realize the same operation, but we can see here how a different time-constant results in a much slower evolution for layer-1, as compared to layer-2 (Fig. 5.45).

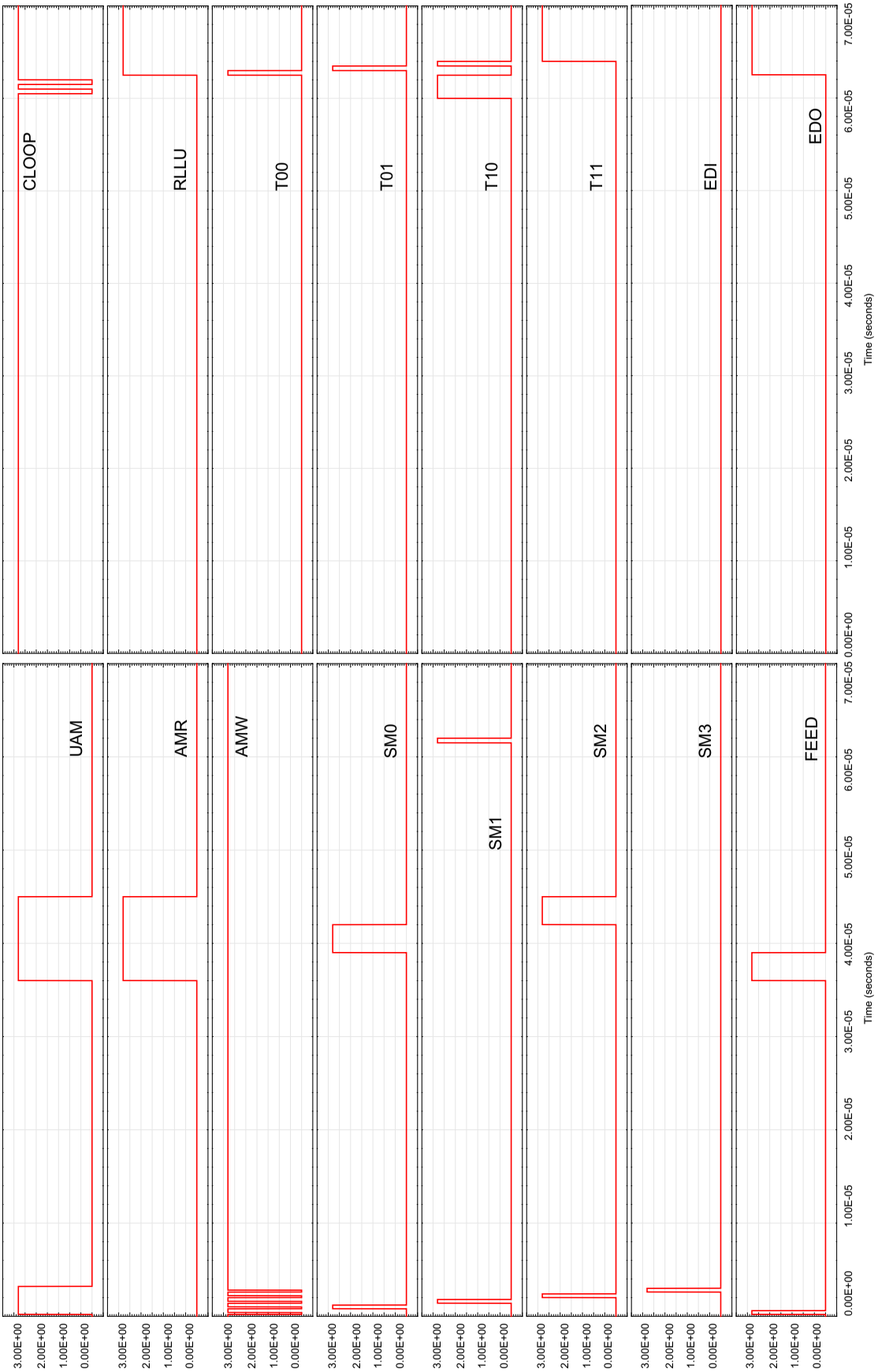


FIGURE 5.42. Control signals for the evolution of a single-cell alone.

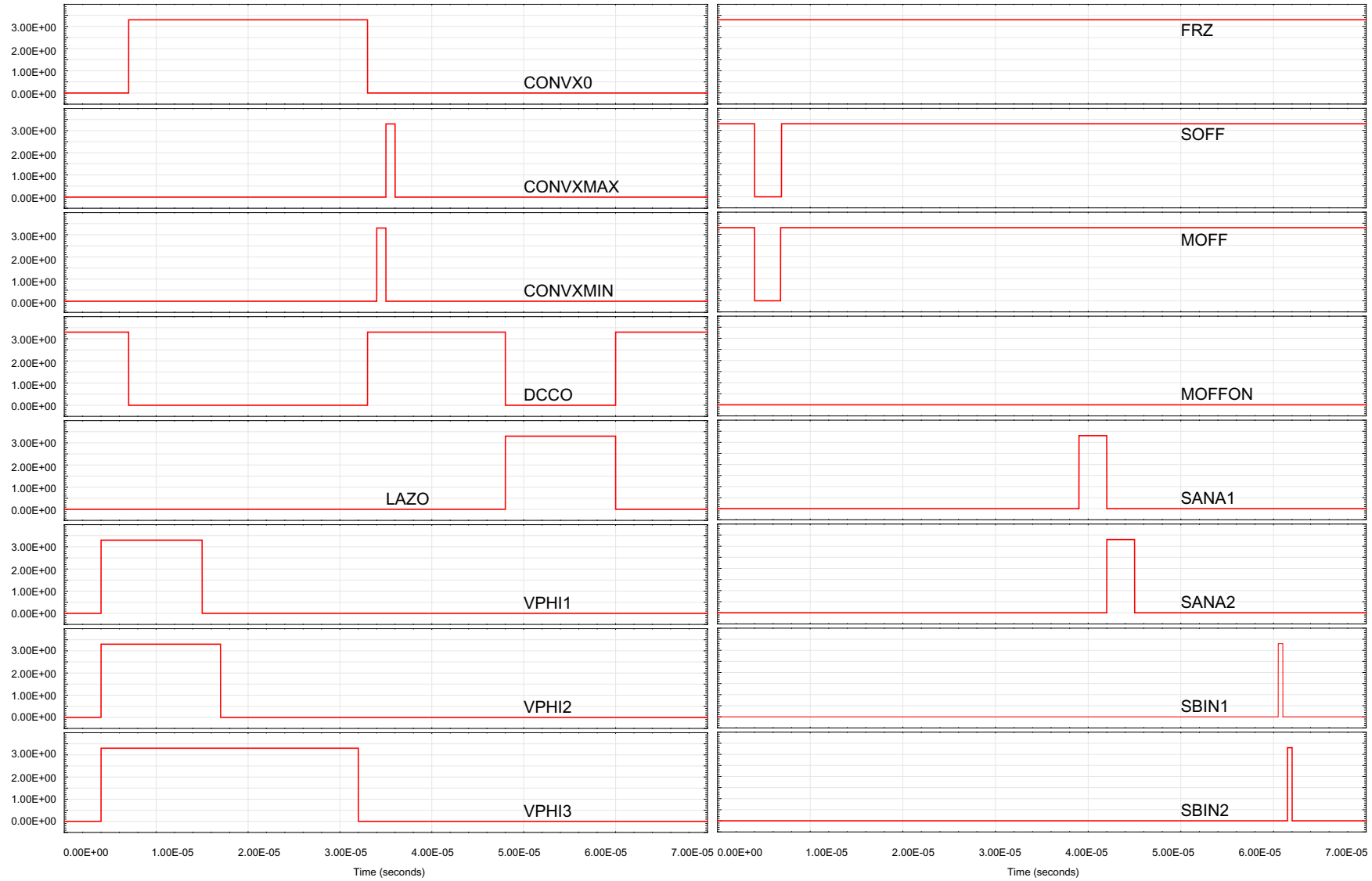


FIGURE 5.42. (cont.) Control signals for the evolution of a single-cell alone.

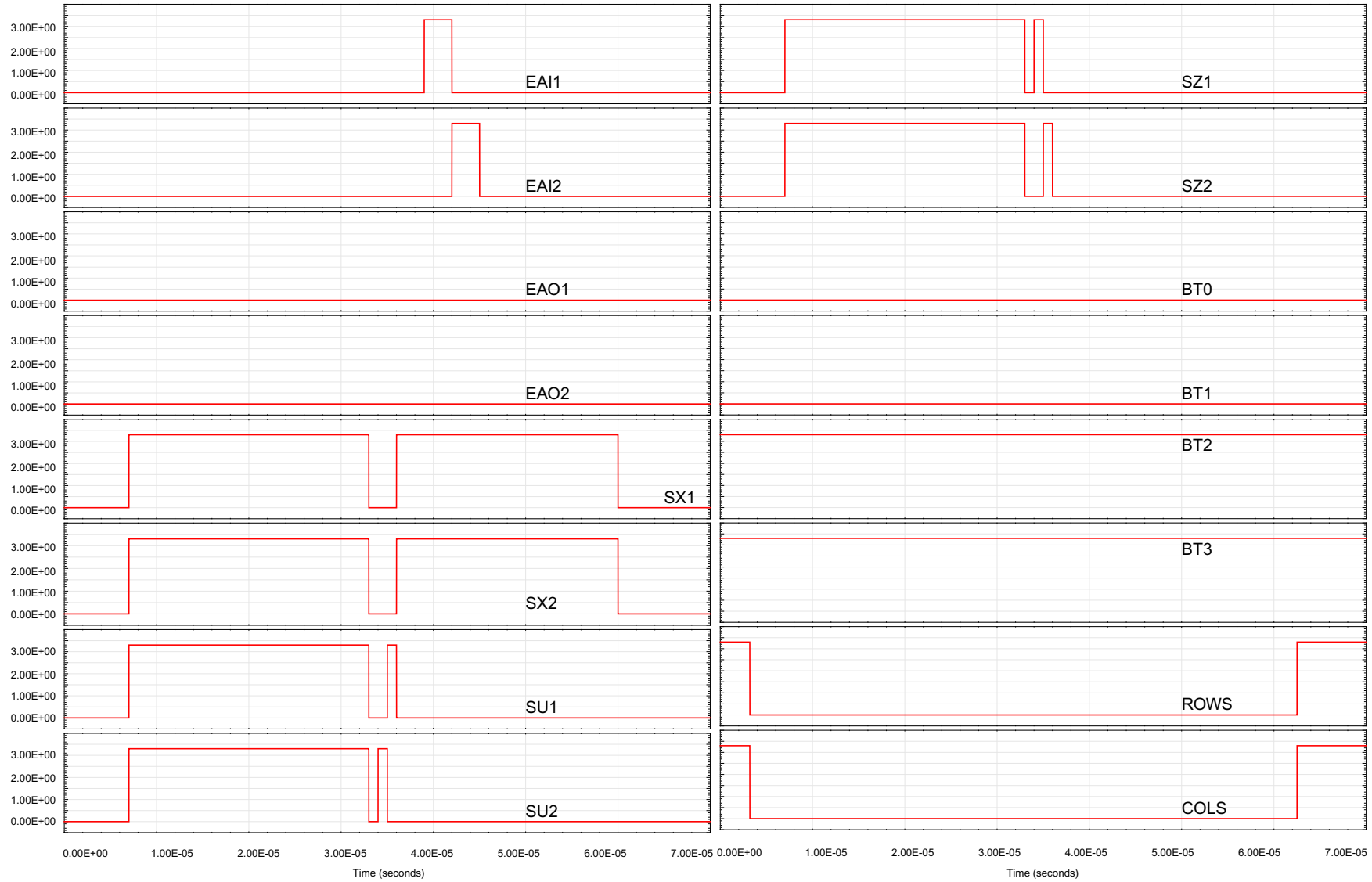


FIGURE 5.42. (cont.) Control signals for the evolution of a single-cell alone.

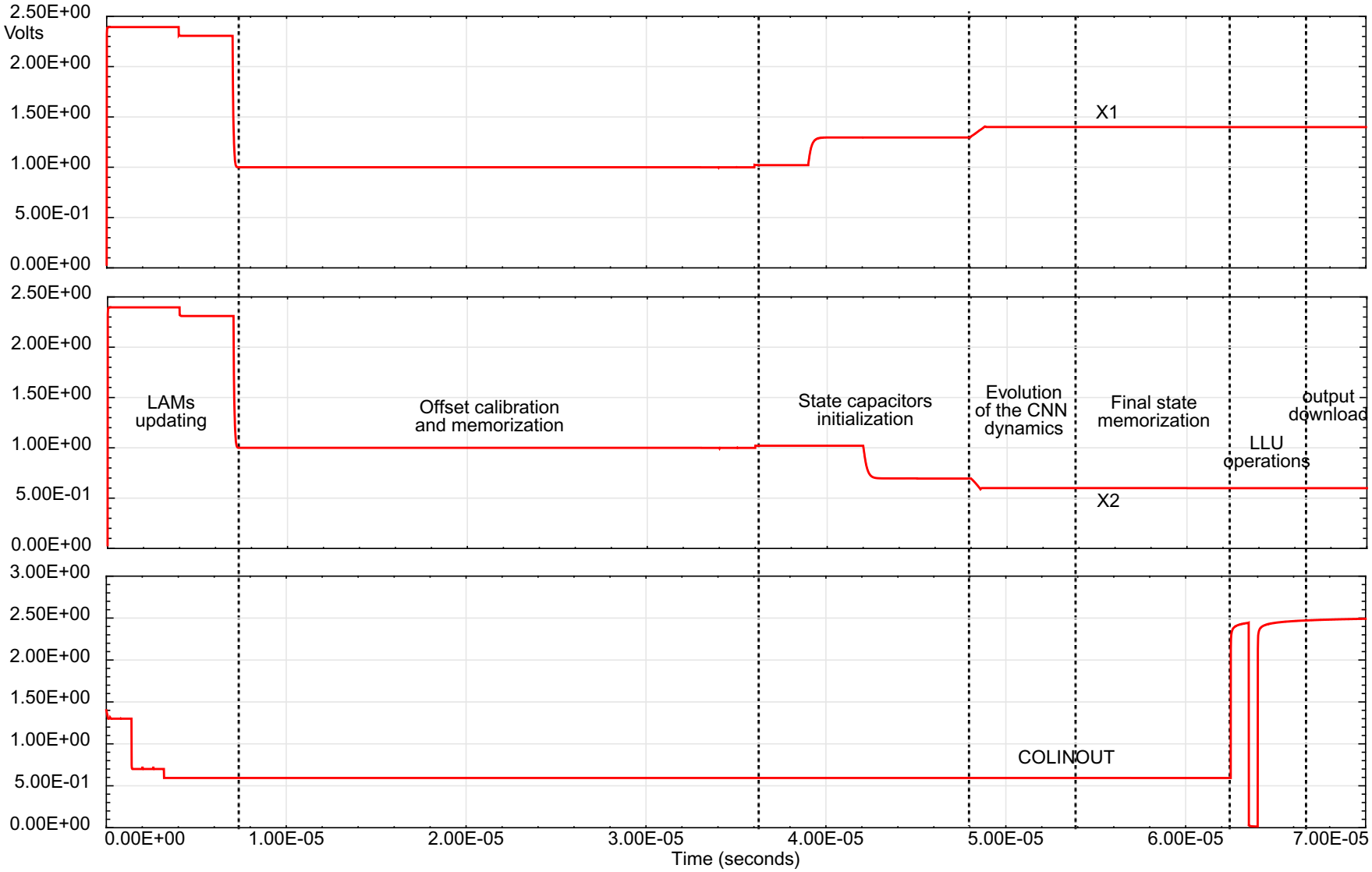


FIGURE 5.43. Evolution of the cell state voltages and the I/O node.

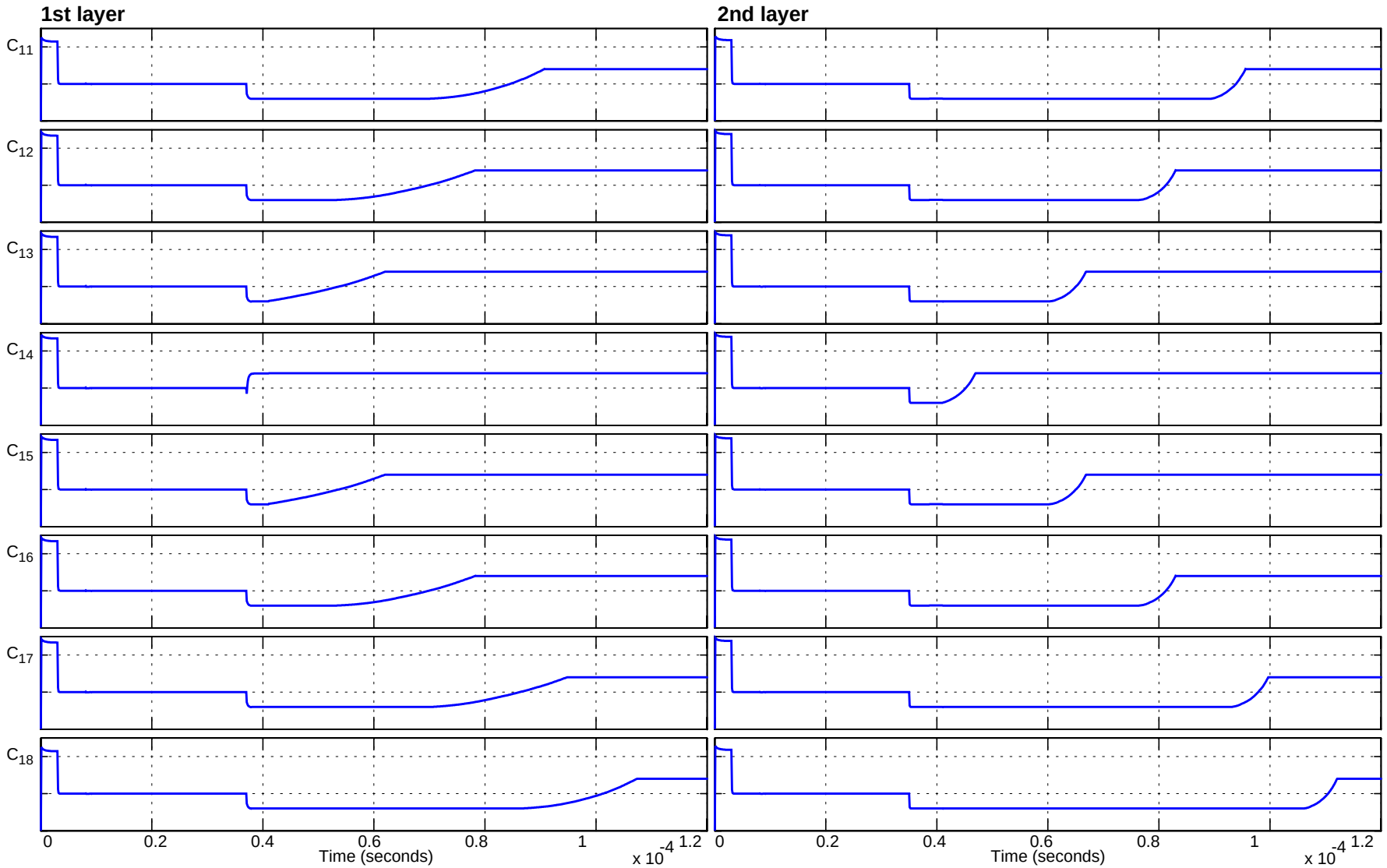


FIGURE 5.44. Evolution ofthe states of both layers in a row of 8 cells

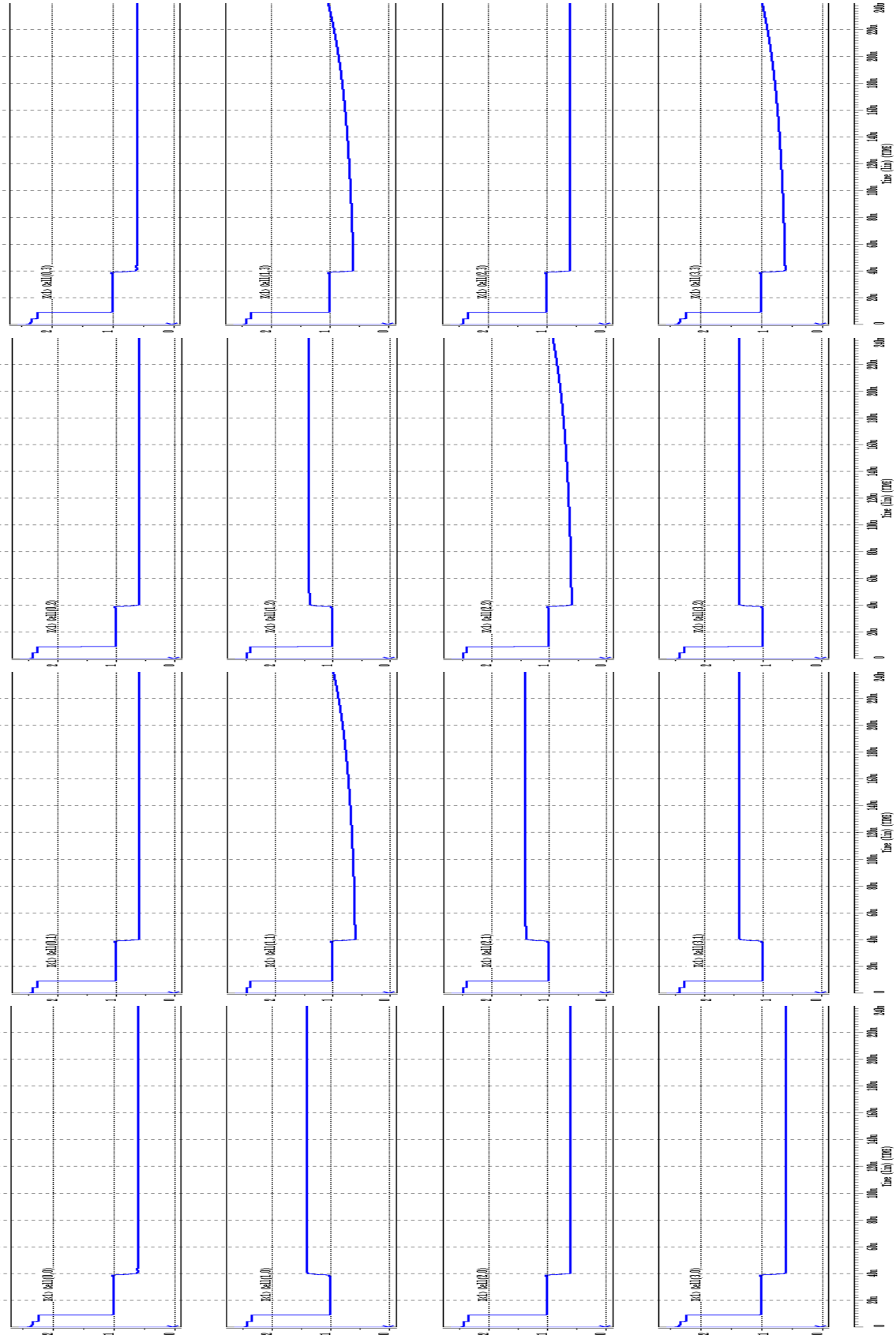


FIGURE 5.45. Evolution of the state variable of the 1st layer (the slower one) of a 4×4 -cells network.

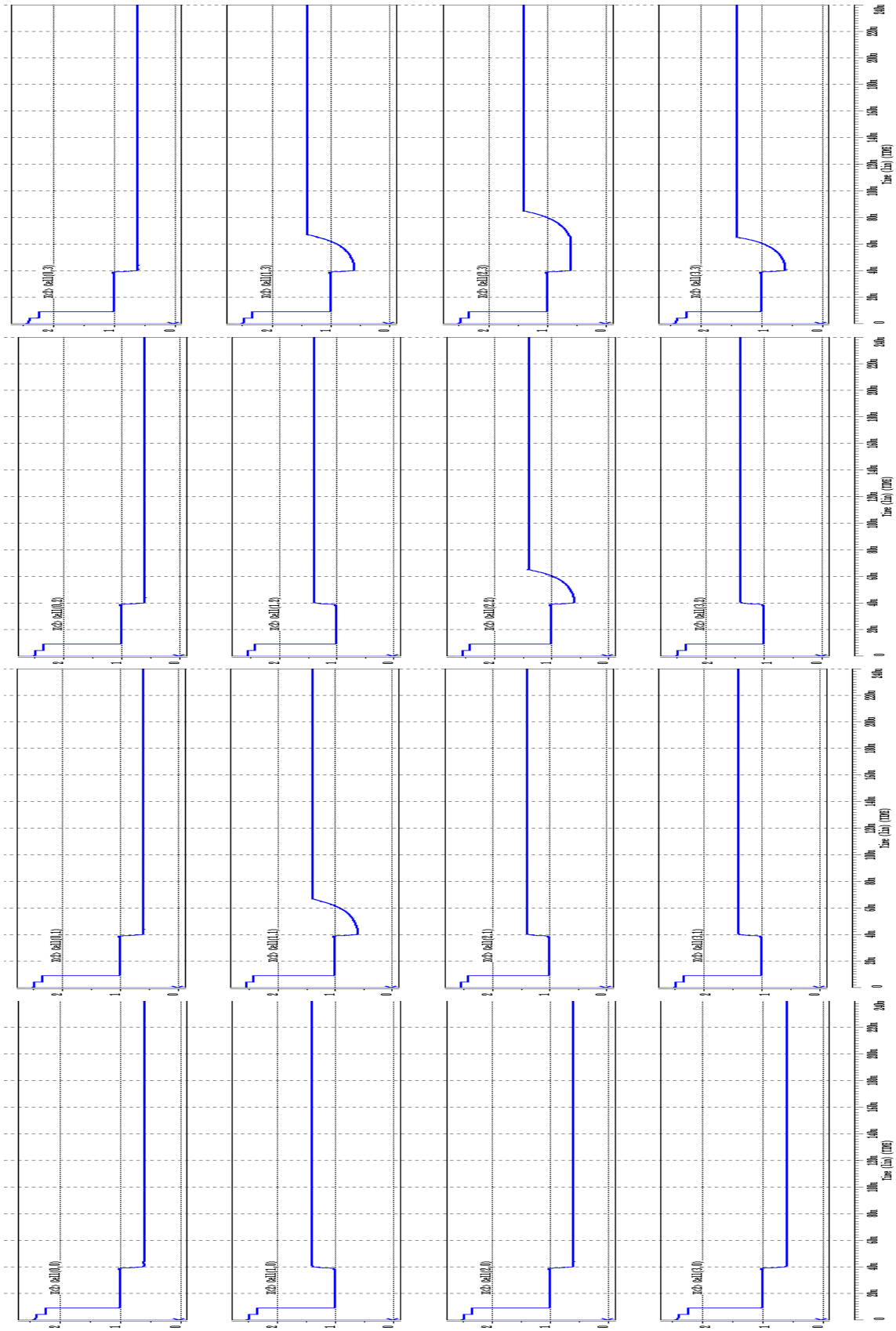


FIGURE 5.45. (cont.) Evolution of the state variable of the 2nd layer (the fast one) of a 4×4 -cells network.

Bibliografía

[Angu98] M. Anguita, F. J. Pelayo, E. Ros and D. Palomar, “Focal-Plane and Multiple Chip VLSI Approaches to CNNs”. *Analog Integrated Circuits and Signal Processing*, Vol. 15, No. 3, pp. 263-275, March 1998.

[Alls93] D. J. Allstot, S.-H. Chee, S. Kiaei and M. Shrivastawa, “Folded Source-Coupled Logic vs. CMOS Static Logic for Low-Noise Mixed-Signal ICs”. *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, Vol. 40, No. 9, pp. 553-563, September 1993.

[Bake98] R. Jacob Baker, Harry W. Li and David E. Boyce, *CMOS Circuit Design, Layout and Simulation*. IEEE Press, New York, 1998.

[Brod66] P. Brodatz*, Textures: A Photographic Album for Artists and Designers*, Dover Publications, New York, 1966.

[Carm98] R. Carmona, S. Espejo, R. Domínguez-Castro, A. Rodríguez-Vázquez, T. Roska, T. Kozek and L. O. Chua, “A 0.5µm CMOS CNN Analog Random Access Memory Chip for Massive Image Processing”. *Proc. of the Fifth IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 271-276, London, UK, April 1998.

[Carm99a] R. Carmona, I. García-Vargas, J. F. Ramos, R. Domínguez-Castro, S. Espejo and A. Rodríguez-Vázquez, “SIRENA: A CAD Environment for Behavioral Modeling and Simulation of VLSI CNNs”. *International Journal of Circuit Theory and Applications*, Vol. 27, No. 1, pp. 43-76, January-February 1999.

[Carm99b] R. Carmona, A. Rodríguez-Vázquez, S. Espejo, R. Domínguez -Castro, T. Roska, T. Kozek and L. O. Chua, “A 0.5um CMOS Analog Random Access Memory Chip for TeraOPS Speed Multimedia Video Processing”, *IEEE Transactions on Multimedia*, Vol. 1, No. 2, pp. 121-135. June 1999.

[Carm99c] R. Carmona, A. Rodríguez-Vázquez, S. Espejo and R. Domínguez-Castro, “A 0.5µm CMOS Analog Random Access Memory Chip for Real-Time Video Processing”. *25th European Solid-State Circuits Conference (ESSCIRC’99)*, pp. 162-165. Duisburg, Germany, September 1999. ISBN 2-86332-246-X.

[Carm99d] R. Carmona, A. Rodríguez-Vázquez, S. Espejo and R. Domínguez-Castro, "A CMOS Analog Memory Buffer Chip for Real-Time Image Processing". *XIV Design of Integrated Circuits*

and Systems Conf. (DCIS'99), pp. 807-810. Palma de Mallorca, Islas Baleares, España. Noviembre 1999. ISBN 84-7632-424-3.

[Cauw96] G. Cauwenberghs, "An Analog VLSI Recurrent Neural Network Learning a Continuous-Time Trajectory", *IEEE Transactions on Neural Networks*, Vol. 7, No. 2, pp. 346-361, March 1996.

[CCIR90a] "Rec. 601-2: Encoding Parameters of Digital Television for Studios". *Recommendations of the International Consultative Committee for Radio-communications (CCIR).* Vol. XI, pp. 95-104, Geneva, Switzerland 1990.

[CCIR90b] "Report 624-4: Characteristics of Television Systems". *Reports of the International Consultative Committee for Radio-communications (CCIR).* Annex to Vol. XI, Part 1-Broadcasting Service, pp. 1-31, Geneva, Switzerland 1990.

[CCIR90c] "Rec. 709: Basic Parameter Values for the HDTV Standard for the Studio and for International Programme Exchange". *Recommendations of the International Consultative Committee for Radio-communications (CCIR).* Annex to Vol. XI, Part 1-Broadcasting Service (Television), pp. 4-12, Geneva, Switzerland 1990.

[Chan76] J. J. Chang, "Nonvolatile Semiconductor Memory Devices". *Proceedings of the IEEE*, Vol. 64, No. 7, pp. 1039-1059, July 1976.

[Chua87] Leon O. Chua, Charles A. Desoer and Ernest S. Kuh, *Linear and Nonlinear Circuits*. McGraw-Hill, New York, 1987.

[Chua88] L.O. Chua and L. Yang, "Cellular Neural Networks: Applications", *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, Vol. 35, No. 10, pp. 1273-1290, October 1988.

[Chua91] L.O. Chua and P. Thiran, "An Analytic Method for Designing Simple Cellular Neural Networks", *IEEE Transactions on Circuits and Systems*, Vol. 38, No. 11, pp. 1332-1341, November 1991.

[Chua93] L.O. Chua and T. Roska, "The CNN Paradigm". *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 40, No. 3, pp. 147-156, March 1993.

[Chua96] L. O. Chua, T. Roska, T. Kozek and A. Zarandy, "CNN Universal Chips Crank up the Computing Power". *IEEE Circuits and Devices Magazine*, Vol. 12, No. 4, pp. 18-28, July 1996.

[Crou93] K. R. Crounse, T. Roska and L. O. Chua, "Image Halftoning with Cellular Neural Networks". *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Vol. 40, No. 4, pp. 267-283, April 1993.

[Crou95] K. R. Crounse and L. O. Chua, "Methods for Image Processing and Pattern Formation in Cellular Neural Networks". *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 42, No. 10, pp. 583-601, October 1995.

[Crou96] K. R. Crounse and L. O. Chua, "The CNN Universal Machine is as universal as a Turing Machine". *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 43, No. 4, pp. 353-355, April 1996.

[Crou97] Kenneth R. Crounse, *Image Processing Techniques for Cellular Neural Network Hardware*. Ph. D. Thesis, University of California, Berkeley, June 1997.

[Cruz92] J. M. Cruz and L.O. Chua, "Design of High-Speed, High-Density CNNS in CMOS Technology". *International Journal of Circuit Theory and Applications*, Vol. 20, No. 5, pp. 555-572, September-October 1992.

[Cruz98] J. M. Cruz and L. O. Chua, "A 16 x 16 Cellular Neural Network Universal Chip: the First Complete Single-Chip Dynamic computer Array with distributed Memory and Grayscale I/O", *Analog Integrated Circuits and Signal Processing*, Vol. 15, No. 3, pp 227-238, March 1998.

[Demb90] A. Dembo and T. Kailath, "Model-Free Distributed Learning", *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, pp. 58-70, January 1990.

[Deso69] Charles A. Desoer and Ernest S. Kuh, *Basic Circuit Theory*, McGraw-Hill, New york, 1969.

[Devo93] F. Devos, M. Zhang, Y. Ni, Y., J. F.Pone, "Trimming CMOS Smart Imager with Tunnel-Effect Nonvolatile Analogue Memory". *IEE Electronics Letters*, Vol. 29, No. 20, pp. 1766-1767, September 1993.

[Dior95] C. Diorio, S. Mahajan, P. Hasler, B. Minch, et al. "A High-Resolution Non-Volatile Analog Memory Cell". *Proc. of the 1995 IEEE Symposium on Circuits and Systems*, Vol. 3, pp. 2233-2236, Seattle, WA, USA, April 1995.

[Domí97] R. Domínguez-Castro, S. Espejo, A. Rodríguez-Vázquez, R. Carmona, P. Foldesy, A. Zarándy, P. Szolgay, T. Sziranyi and T. Roska, "A 0.8 μm CMOS Programmable Mixed-Signal Focal-Plane Array Processor with On-Chip Binary Imaging and Instructions Storage", *IEEE Journal of Solid State Circuits*, Vol. 32, No. 7, pp. 1013-1026, July 1997.

[Domí98] R. Domínguez-Castro, A. Rodríguez-Vázquez, S. Espejo and R. Carmona, "Four-Quadrant One-Transistor Synapse for High Density CNN Implementations". *Proc. of the Fifth IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 243-248, London, UK, April 1998.

[EBU93] "Interlaced Version of the 1250/50 HDTV Production Standard". *European Broadcasting Union (EBU) Technical Documents. Tech 3000 Series*, EBU Tech. 3271-E, Geneva, Switzerland 1990.

[Espe93a] S. Espejo, A. Rodríguez-Vázquez, R. Domínguez-Castro, J.L. Huertas and E. Sánchez-Sinencio, "An Analog Design Technique for Smart-Pixel CMOS Chips". *Proceedings of the 1993 European Solid-State Circuits Conference*, pp. 78-81, Sevilla, September 1993.

[Espe93b] S. Espejo, R. Carmona, R. Domínguez-Castro and A. Rodríguez-Vázquez. "Design of Sensory Processing CNN Chips". *Proc. of the International Symposium on Non-Linear Theory and its Applications, NOLTA'93.* Vol. 1, pp 5-10, Hawaii, December 1993.

[Espe94a] S. Espejo, *Cellular Neural Networks: VLSI Design and Modelling*. Ph. D. Thesis, Universidad de Sevilla, February 1994.

[Espe94b] S. Espejo, A. Rodríguez-Vázquez and R. Domínguez-Castro, “Smart-Pixel Cellular Neural Networks in Analog Current-Mode CMOS Technology”, *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 8, pp. 895-905, August 1994.

[Espe94c] S. Espejo, A. Rodríguez-Vázquez, R. Domínguez-Castro and R. Carmona, “Convergence and Stability of the FSR CNN Model”. *Proceedings of the 3rd International Workshop on Cellular Neural Networks and their Applications*, pp. 411-417, Rome, December 1994.

[Espe96a] S. Espejo, R. Carmona, R. Domínguez-Castro and A. Rodríguez-Vázquez, “A CNN Universal Chip in CMOS Technology”. *International Journal of Circuit Theory and Applications.* Vol 24, No. 1, pp. 93-110, January-February 1996.

[Espe96b] S. Espejo, R. Domínguez-Castro, R. Carmona and A. Rodríguez-Vázquez, “Hybrid Control of Synapse Circuits for Programmable Cellular Neural Networks”. *Proc. of IEEE International Symposium on Circuits and Systems*, Vol. 3, pp. 507-510, Atlanta, GA, USA, 12-15 May 1996.

[Fehe96] B. Feher, P. Szolgay, T. Roska, A. G. Radvanyi et al., “ACE: A Digital Floating Point CNN Emulator Engine”. *Proceedings of the 4th IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 273-278, Sevilla, June 1996.

[Flyn66] M. J. Flynn, “Very High Speed Computing Systems”. *Proceedings of the IEEE*, Vol. 54, pp. 1901-1909. December 1999.

[Fold96] P. Földesy, P. Szolgay and A. Zarándy, *Functional Testing of the cP 300 14 x 14 CNN Universal Chip*. Analogical and Neural Computing Laboratory, Computer and Automation Institute. Budapest, 1996.

[Fold98] P. Földesy, L. Kék, T. Roska, A. Zarándy and G. Bártfai, “Fault Tolerant CNN Template Design and Optimization Based on Chip Measurements”. *Proceedings of the Fifth IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 404-409, London, UK, April 1998.

[Fran92] E. Franchi, M. Tartagni, R. Guerrieri and G. Baccarani, “Random Access Analog Memory for Early Vision”. *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 7, pp. 1105-1109, July 1992.

[Froh74] D. Frohman-Bentchkowsky, “FAMOS—A New Semiconductor Charge Storage Device”. *Solid-State Electronics*, Vol. 17, No. 6, pp. 517-529, June 1974.

[Furth97] P. M. Furth and A. Andreou, “On Fault Probabilities and Yield Models for VLSI Neural Networks”. *IEEE Journal of Solid-State Circuits*, Vol. 32, No. 8, pp. 1284-1287, August 1997.

[Geal99] J. C. Gealow and C. G. Sodini, “A Pixel-Parallel Image Processor Using Logic Pitch -Matched to Dynamic Memory”. *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 6, pp. 831-839, June 1999.

[Geig90] Randall L. Geiger, Phillip E. Allen and Noel R. Strader. *VLSI Design Techniques for Analog and Digital Circuits*. McGraw-Hill Pub. Co. New York, 1990.

[Gero99] A. Gerosa, G. M. Cortelazzo, A. Baschirotto and E. Malavasi, “2D Video Rate SC FIR Filters Based on Analog RAMs”.

IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 46, No. 11, pp. 1348-1360, November 1999.

[Gilb68] B. Gilbert, "A Precise Four-Quadrant Multiplier with Subnanosecond Response". *IEEE Journal of Solid-State Circuits*, Vol. 3, No. 4, pp. 365-373, December 1968.

[Gold89] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading MA, 1989

[Gonz87] R. C. González and P. Wintz, *Digital Image Processing*, Addison-Wesley, Reading MA, 1987.

[Goor89] A. J. Van Der Goor, *Computer Architecture and Design*, *Electronic Systems Engineering Series*. Addison-Wesley, Wokingham, England, 1989.

[Good96] F. Goodenough, "Analog Technologies of All Varieties Dominate ISSCC". *Electron Design*, Vol. 44, pp. 96-111, 1996.

[Gray93] Paul R. Gray and Robert G. Meyer, *Analysis and Design of Analog Integrated Circuits*. 3rd edition. "Chapter 11: Noise in Integrated Circuits". Wiley, New York, 1993.

[Greg94] Roubik Gregorian and Gabor C. Temes, *Analog MOS Integrated Circuits for Signal Processing*. John Wiley & Sons, New York, 1994.

[Guck83] John Guckenheimer and Philip Holmes, *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields*. Springer-Verlag, New York, 1983.

[Habe87] Richard Haberman, *Elementary Applied Partial Differential Equations*. Prentice-Hall International Inc. Englewood Cliffs, NJ, 1987.

[Hall89] G. M. Haller and B. A. Wooley, "A 700-MHz Switched Capacitor Analog Waveform Sampling Circuit". *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 4, April 1989.

[Häng99] M. Hänggi and G. S. Moschytz, "An Exact and irect Analytical Method for the Design of Optimally Robust CNN Templates". *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, Vol. 46, No. 2, pp. 304-311, February 1999.

[Harr92a] H. Harrer, J. A. Nossek and R. Steltz, "An Analog Implementation of Discrete-Time Cellular Neural Networks". *IEEE Trans. Neural Networks*, Vol. 3, No. 5, pp. 466-476, May 1992.

[Harr92b] H. Harrer and J. A. Nossek, "Discrete-Time Cellular Neural Networks". *International Journal of Circuit Theory and Applications,* Vol. 20, No. 5, pp. 453-467, September-October 1992.

[Harr94] H. Harrer, *Comparison of Different Numerical Integration Methods for Simulating Cellular Neural Networks*. Institute for Network Theory and Circuit Design, Technical University of Munich. Report TUM-LNS-TR-90-9, September 1994.

[Hayk94] S. Haykin, *Neural Networks: A Comprehensive Foundation*. McMillan, New York, 1994.

[Hori90] Y. Horio, M. Yamamoto and s. Nakamura, "Active Analog Memories for VLSI Analog Neural Networks". Proceedings of the *International Conference on Fuzzy Logic and Neural Networks*, Vol. 2, pp. 655-660, 1990.

[Hori92] Y. Horio and S. Nakamura, “Analog Memories for VLSI Neurocomputing”, in *Artificial Neural Networks: Paradigms, Applications and Hardware Implementations*, pp. 344-363, IEEE Press, 1992.

[Hube88] D. H. Hubel, *Eye, Brain and Vision*. W. H. Freeman, New York, 1988.

[Isma94] Mohammed Ismail and Terri Fiez (eds.), *Analog VLSI: Signal and Information Processing*. McGraw-Hill, New York, 1994.

[ISO93] ISO/IEC 11172-2. *Information Technology –Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbits/s.– Part 2: Video*. Geneva 1993.

[ITU93] *Rec. H. 261: Video Codec for Audiovisual Services at p x 64kbit/s*. *Recommendations of the International Telecommunications Union*, Helsinki, Finland 1993.

[Jaco94] A. Jacobs, F. Werblin, and T. Roska, “Techniques for Constructing Physiologically Motivated Neuromorphic models in CNN”. *Proceedings of Third IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pp. 53-58, Rome, December 1994.

[Jaco96] A. Jacobs, T. Roska and F. S. Werblin, “Methods for Constructing Physiologically Motivated Neuromorphic Models in CNNs”. *International Journal of Circuit Theory and Applications*, Vol. 24, No. 3, pp. 315-339, May-June 1996.

[Jaeg82] Richard C. Jaeger, “Tutorial: Analog Data Acquisition Technology. Part I–Digital-to-Analog Conversion”. *IEEE Micro*, Vol. 24, No. 3, pp. 20-37, May 1982.

[Jähn99a] B. Jähne, “Multiresolutional Signal Representation”, Chapter 4 in Bernd Jähne, Horst Hauβecker and Peter Geiβler (Eds.), *Handbook of Computer Vision and Applications. Volume 2: Signal Processing and Pattern Recognition.* Academic Press, San Diego, 1999.

[Jähn99b] B. Jähne, “Neighborhood Operators”, Chapter 5 in Bernd Jähne, Horst Hauβecker and Peter Geiβler (Eds.), *Handbook of Computer Vision and Applications. Volume 2: Signal Processing and Pattern Recognition.* Academic Press, San Diego, 1999.

[Jeni98] Jen-I Pi, *MOSIS Scalable CMOS (SCMOS) Design Rules*, Revision 7.2, The MOSIS Service, USC/ISI, February 1998.

[King95] P. Kinget and M. S. J. Steyaert, “A Programmable Analog Cellular Neural Network CMOS Chip for High Speed Image Processing”. *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 3, pp. 235-243, March 1995.

[King96] P. Kinget and M. Steyaert, “Evaluation of CNN Template Robustness Towards VLSI Implementation”. *International Journal of Circuit Theory and Applications,* Vol. 24, No. 1, pp. 111-120, January-February 1996.

[Kirk83] S. Kirkpatrick, C. D. Gelatt Jr. and M. P. Vecchi, “Optimization by Simulated Annealing”. *Science*, Vol. 220, No. 4598, pp. 671-680, May 1983.

[Koch95] Christof Koch and Hua Li (Eds.), *Vision Chips: Implementing Vision Algorithms with Analog VLSI Circuits*, IEEE Computer Society Press, Los Alamitos, 1995.

[Koch96] C. Koch and B. Mathur, “Neuromorphic Vision Chips”. *IEEE Spectrum*, Vol. 33, No. 5, pp. 38-46, May 1996.

[Kolo86] A. Kolodny, S. T. K. Nieh, B. Eitan and J. Shappir, “Analysis and Modeling of Floating-Gate EEPROM Cells”. *IEEE Transactions on Electron Devices*, Vol. 33, No. 6, pp. 835-844, July 1986.

[Kore89] Israel Koren (Ed.): *Defect and Fault Tolerance in VLSI Systems*. Plenum Press, New York, 1989.

[Kosk92] Bart Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence.* Prentice Hall, englewood Cliffs, NJ, 1992.

[Koze93] T. Kozek, T. Roska and L.O. Chua, “Genetic Algorithm for CNN Template Learning”, *IEEE Transactions on Circuit and Systems I: Fundamental Theory and Applications*, Vol. 40, No. 6, pp. 392-402, June 1993.

[Krau97] W. H. Krautschneider, A. Kohlhase and H. Terletzki, “Scaling Down and Reliability Problems of Gigabit CMOS Circuits”. *Microelectronics Reliability*, Vol. 37, No. 1, pp. 19-37, January 1997.

[Lake94] Kenneth R. Laker and William M. C. Sansen, *Design of Analog Integrated Circuits and Systems*. McGraw-Hill, New York, 1994.

[Lina91] B. Linares-Barranco, E. Sánchez-Sinencio, A. Rodríguez-Vázquez and J. L. Huertas, “VLSI Implementation of a Transconductance Mode Continuous BAM with On-Chip Learning and Dynamic Analog Memory”, *Proceedings of the International Symposium on Circuits and Systems*, pp. 1283-1286, Singapore, June 1991.

[Liña98] G. Liñan, S. Espejo, R. Domínguez-Castro, E. Roca and A. Rodríguez-Vázquez, “A 64 x 64 CNN with Analog and Digital I/O”. *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems*, pp. 203-206, Lisbon, Portugal, September 1998.

[Liu93] D. Liu, “Cloning Template Design of Cellular Neural Networks for Associative Memories”, *IEEE Transactions on Circuit and Systems I: Fundamental Theory and Applications*, Vol. 44, No.7, pp. 646-650, July 1993.

[Loin99] M. Loinaz and B. Ackland, “Video Cameras: CMOS Technology Provides On-Chip Processing”. *Sensor Review*, Vol. 19, No. 1, pp. 19-26, January 1999.

[Mart98] D. A. Martin, H. S. Lee and I. Masaki, “A Mixed-Signal Array Processor with Early Vision Applications”. *IEEE Journal of Solid-State Circuits*, Vol. 33, No. 3, pp. 497-502, March 1998.

[Math92] *MATLAB Reference Guide.* The MathWorks Inc. 1984-1992.

[Mats85] K. Matsui, T. Matsura, S. Fukasawa, Y. Izawa, Y. Toba, N. Miyake and K. Nagasawa, “CMOS Video Filter Using Switched-Capacitors 14-MHz Circuits”. *IEEE Journal of Solid-State Circuits*, Vol. SC-20, pp. 1096-1102, Dec. 1985.

[Mats90] T. Matsumoto, L.O. Chua and H. Suzuki, “CNN Cloning Template: Connected Component Detector”. *IEEE Trans. Circuits and Systems*, Vol. 37, pp. 633-635, May 1990.

[Matt98] Timothy G. Mattson and Greg Henry, “An Overview of the

Intel TFLOPS Supercomputer". *Intel Technology Journal,* Issue No. Q1-98, January 1998.

[McKa95] David J. McKay, "Bayesian Methods for Supervised Neural Networks" in Michael A. Arbib (ed.), *The Handbook of Brain Theory and Neural Networks*, The MIT Press, pp. 144-149, Cambridge, Massachusets, 1995.

[McCl86] Edward J. McCluskey, *Logic Design Principles: with Emphasis on Testable Semicustom Circuits.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Mead89] Carver Mead, *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA, 1989.

[Mede98] Fernando Medeiro, Belén Pérez-Verdú and Ángel Rodríguez-Vázquez, *Top-Down Design of High-Performance sigma-Delta Modulators*, Kluwer Academic Publishers, Boston, 1998.

[Meta88] *HSPICE User Manual.* Meta Software Inc. 1988.

[Metr53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, "Equation of State Calculations by Fast Computing machines". Journal of Chemical Physics, Vol. 21, No. 6, pp. 1087-1092, 1953.

[Mich92] C. Michael and M. Ismael, "Statistical Modeling of Device Mismatch for Analog MOS Integrated Circuits". *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 2, pp- 154-166, February 1992.

[Moin99] Alireza Moini, *Vision Chips*. Kluwer Academic Publishers, Boston, 1999.

[Moll98] S. Molloy and R. Jain, "A 100-K Transistor 25-MPixels/s onfigurable Image Transform Processor Unit". *IEEE Journal of Solid-State Circuits*, Vol. 33, pp. 86-97, January 1998.

[Nish93a] K. A. Nishimura, *Optimum Partitioning of Analog and Digital Circuitry in Mixed-Signal Circuits for Signal Processing*. Ph. D. Dissertation, College of Engineering, University of California, Berkeley, July 1993.

[Nish93b] K. A. Nishimura and P. R. Gray, "A Monolithic Analog Video Comb Filter in 1.2µm CMOS". *IEEE Journal of Solid-State Circuits*, Vol. 28, No. 12, pp. 1331-1339, December 1993.

[Noss93] J. A. Nossek, "Design and Learning with Cellular Neural Networks", *International Journal of Circuit Theory and Applications*, Vol. 24, pp. 15-24, 1996.

[Oppe93] A. V. Oppenheim, A. S. Willsky and I. T. Young, *Signals and Systems*. Prentice-Hall, Englewood Cliffs, New Jersey 1983.

[Paas97] A. Paasio, A. Dawidziuk, K. Halonen and V. Porra, "Minimum Size 0.5µm CMOS Programmable 48 x 48 CNN Test Chip", *Proc. of the 1997 European Conference on Circuit Theory and Design*, pp. 154-156, Budapest, Hungary, September 1997.

[Pear89] B. A. Pearlmutter, "Learning State Space Trajectories in Recurrent Neural Networks". *Neural Computation*, Vol. 1, pp. 263-269, 1989.

[Pelg89] M. J. M Pelgrom, A. C. J. Duinmaijer and A. P. G. Welbers, "Matching Properties of MOS Transistors", *IEEE Journal Solid-State Circuits*, Vol. 24, No. 10, pp. 1433-1440, October 1989.

[Pine87] F. J. Pineda, “Generalization of Back-Propagation to Recurrent Neural Networks”. *Physical Review Letters*, Vol. 59, pp. 2229-2232, 1987.

[Pine96] J. Pineda de Gyvez, Lei Wang and E. Sanchez-Sinencio, “Large-image CNN hardware processing using a time multiplexing scheme”. *Proc. of the Fourth IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 405-410, Sevilla, Spain, June 1996.

[Ping97] W. Ping and J. E. Franca, “A CMOS 1μm 2-D Analog Multirate System for Real-Time Image Processing”. *IEEE Journal of Solid-State Circuits*, Vol. 32, No. 7, pp. 1037-1048, July 1997.

[Pirs98] P. Pirsch and H. J. Stolberg, “VLSI Implementation of Image and Multimedia Processing Systems”, *IEEE Transcations on Circuits and Systems for Video Technology*, Vol. 8, No. 7, pp. 878-891. November, 1998.

[Plon90] M. Plonski and C. Joyce, “RCS, GENESIS and SFINX: Three Public-Domain Simulators for Neural Networks”. *Neural Networks Review* 4, 1990.

[Poll90] L. Howard Pollard, *Computer Design and Architecture*, Prentice-Hall Inc., Englewood Cliffs, N. J., 1990.

[Poyn96] Charles A. Poynton, *A Technical Introduction to Digital Video*. John Wiley & Sons, New York, 1996.

[Pres88] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, *Numerical Recipes in C: the Art of Scientific Computing*. Cambridge University Press, New York 1988.

[Reke97] Csaba Rekeczky, *MATCNN, Analogic CNN Simulation toolbox for MATLAB. Version 1.0*. Analogic and Neural Computing Laboratory. Computer and Automation Institute (MTA-SzTAKI). Tech. Report DNS-11-1997. Budapest, November 1997.

[Reke00a] Cs. Rekeczky, B. Roska, E. Nemeth and F. Werblin, “Neuromorphic CNN Models for Spatio-Temporal Effects Measured in the Inner and Outer Retina of Tiger Salamander”. *Proc. of the Sixth IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 15-20, Catania, Italy, May 2000.

[Reke00b] Cs. Rekeczky, T. Serrano-Gotarredona, T. Roska and A. Rodríguez-Vázquez, “A Stored Program 2nd Order/3-Layer Complex Cell CNN-UM”. *Proc. of the Sixth IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 219-224, Catania, Italy, May 2000.

[Rodr93] A. Rodríguez-Vázquez, S. Espejo, R. Domínguez-Castro, J.L. Huertas and E. Sánchez-Sinencio, “Current-Mode Techniques for the Implementation of Continuous-Time and Discrete-Time Cellular Neural Networks”, *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, Vol. 40, No. 3, pp. 132-146, March 1993.

[Rodr95] A. Rodríguez-Vázquez, R. Domínguez-Castro, F. Medeiro, J. L. Huertas and M. Delgado-Restituto. "High Resolution CMOS Current Comparators: Design and Applications to Current Mode Function Generation". *Analog Integrated Circuits*

and Signal Processing, Vol. 7, No. 2, pp. 149-165, March 1995.

[Rosk88] T. Roska, "Analog Events and a Dual Computing Structure Using Analog and Digital Circuits and Operators". In P. Varaiya and A. B. Khurzansky (Eds.): *Discrete Events Systems: Models and Applications*, Springer Verlag, Berlin 1988.

[Rosk90] T. Roska and L. O. Chua, "Cellular Neural Networks with Nonlinear and Delay-Type Template Elements". *Proceedings of the First IEEE International Workshop on Cellular Neural Networks and Their Applications*, pp. 12-25, Budapest 1990.

[Rosk93a] T. Roska and L. O. Chua: "The CNN Universal Machine: An Analogic Array Computer". *IEEE Transactions on circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 40, No. 3, pp. 163-173, March 1993.

[Rosk93b] T. Roska and L. O. Chua and A. Zarandy, *Language, Compiler and Operating System for the CNN Supercomputer*. Report UCB/ERL M93/34, University of California, Berkeley, 1993.

[Rosk94] T. Roska, P. Szolgay, Á. Zarándy, P. L. Venetianer, A. Radványi and T. Szirányi, "On a CNN Chip-Prototyping System", *Proceedings of the 3rd IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 375-380, Rome, Italy, 1994.

[Rosk95] T. Roska and L. Kek (Eds.), *Analogic CNN Program Library (Version 6.1)*. Report DNS-6-1995, Analogical and Neural Computing Laboratory, Computer and Automation Institute of the Hungarian Academy of Sciences, Budapest, 1995.

[Rosk96] T. Roska: "CNN Chip Set Architectures and the Visual Mouse". *Proc. of the 4th IEEE Int. Workshop on Cellular Neural Networks and their Applications*, pp 487-492. Sevilla, Spain, June 1996.

[Rosk97] T. Roska, "Implementation of CNN Computing Technology". *Proceedings of the International Conference on Artificial Neural Networks*, pp. 1151-1155. Lausanne, Switzerland, October 1997.

[Rosk99] T. Roska, Á. Zarándy, S. Zöld, P. Földesy and P. Szolgay, "The Computational Infrastructure of Analogic CNN Computing—Part I: The CNN-UM Chip Prototyping System". *IEEE Transactions on Circuit and Systems I: Fundamental Theory and Applications*, Vol. 46, No. 2, pp. 261-268, February 1999.

[Rosk01] B. Roska and F. Werblin, "Vertical interactions between ten parallel, stacked representations in the mammalians retina". *Nature*, McMillan Publishers Ltd. No. 410, pp. 583-587, March 2001.

[Saff97] Paul Saffo, "Sensors: The Next Wave of Infotech Innovation" in the *1997 Ten-Year Forecast*. Institute for the Future, Menlo Park, CA, 1997.

[Sait98] O. Saito et al. "A 1M Synapse Self-Learning Digital Neural Network Chip". *Digest of Technical Papers of the IEEE International Solid-State Circuits Coference*, pp. 94-95. San Francisco, CA. February 1998.

[Sale97] M. Salerno, F. Sargeni and V. Bonaiuto, "A 720 Cells Interconnection-Oriented System for Cellular Neural Networks". *Pro-*

ceedings of the IEEE International Symposium on Circuits and Systems. Vol. 1, pp. 681-684, Hong Kong, June 1997.

[Scha90] Rolf Schaumann, Mohammed S. Ghausi and Kenneth R. Laker. *Design of Analog Filters: Passive, Active RC and Switched Capacitor*. Prentice Hall. Englewood Cliffs, New Jersey, 1990.

[Schme94] T. J. Schmerbeck, "Minimizing Mixed-Signal Coupling and Interaction". *Proc. of the 20th European Solid-State Circuits Conference*, pp. 28-37. Ulm, Germany, September 1994.

[Schro87] Dieter K. Schroder, *Advanced MOS Devices*. In G. W. Neudeck and R. F. Pierret (Eds.), Modular Series on Solid State Devices. Addison-Wesley Pub. Co. Reading, Mass. 1987.

[Seit99] P. Seitz, "Solid-State Image Sensing", Chapter 8 in B. Jähne, H. Hauβecker and P. Geiβler (eds.), *Handbook of Computer Vision and Applications. Volume 1: Sensors and Imaging*. Academinc Press, San Diego, California, 1999.

[Sejn92] T. J. Sejnowski and P. S. Churchland, "Silicon Brains". *BYTE Magazine*, pp. 137-146, October 1992.

[Serr82] J. Serra, *Image Analysis and Mathematical Morphology.* Academic Press, London 1982.

[Serr94] T. Serrano and B. Linares-Barranco: "The Active-Input Regulated-Cascode Current Mirror". *IEEE Transactions on circuits and Systems-I: Fundamental Theory and Applications*, Vol. 41, No. 6, pp. 464-467, June 1994.

[Siko97] T. Sikora, "MPEG Digital Video-Coding Standards". IEEE Signal Processing Magazine, Vol. 14, No. 5, pp. 82-100, September 1997.

[Simo95] A. Simoni, G. Torrelli, F. Maloberti, A. Sartori, S. E. Plevridis and A. N. Birbas, "A Single-Chip Optical Sensor with Analog Memory for Motion Detection". *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 7, pp. 800-806, July 1995.

[Sing94] J. Singh, *Semiconductor Devices: An Introduction*. McGraw-Hill, New York, 1994.

[Smit68] K. C. Smith and A. S. Sedra: "The Current Conveyor —A New Circuit Building Block". *IEEE Proceedings*, Vol. 56, pp. 1368-1369, August 1968.

[SMPT94] "Signal Parameters - 1125-Line High-Definition Production System". *Society of Motion Pictures and Television Engineers (SMPTE) Journal*, pp. 291-294, April 1994.

[Socl85] Sidney Soclof, Analog Integrated Circuits. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[Solo74] J. E. Solomon, "The Monolitic Opamp: A Tutorial Survey". *IEEE Journal of Solid-State Circuits*, Vol. 9, No. 6, pp. 314-332, December 1974.

[Sony97] *Trinitron Multimedia Computer Display: CPD-120VS, CPD-220VS, Operating Instructions.* Sony Corporation, 1997.

[Stan95] B. R. Stanisic, R. A. Rutenbar and L. R. Carley, "Addressing Noise Decoupling in Mixed-Signal IC's: Power Distribution Design and Cell Customization". *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 3, pp. 321-326, March 1995.

[Stap91] C. H. Stapper, " On Murphy's Yield Integral". *IEEE Transac-*

tions on Semiconductor Manufacturing. Vol. 4, No. 4, pp. 294-297, November 1991.

[Szir94] T. Szirányi, M. Csapodi, "Texture Classification by Cellular Neural Network and Genetic Learning", *1994 IEEE Int. Conference on Pattern. Recognition*, pp. 381-383, 1994.

[Szol93] P. Szolgay, I. Szatmári and K. László, "A Fast Fixed Point Learning Method to Implement Associative Memories in CNNs", *IEEE Transactions on Circuit and Systems I: Fundamental Theory and Applications*, Vol. 44, No. 4, pp. 362-366, April 1993.

[Tara98] S. Taraglio, A. Zanela, M. Salerno, F. Sargeni and V. Bonaiuto, "A CNN Stereo Vision Hardware System for Autonomous Robot Navigation". *Proc. of the Fifth IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 181-185, London, England, April 1998.

[Thir95] P. Thiran, K. R. Crounse, L. O. Chua and M. Hasler, "Pattern Formation Properties of Autonomous Cellular Neural Networks". *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 42, No. 10, pp. 757-774, October 1995.

[TI99] *TMS320C6711 Floating-Point Digital Signal Processor Datasheet*. Texas Instruments Inc. March 1999.

[Tong84] G. J. Tonge, "The Television Scanning Process". *SMPTE Journal*, Vol. 93, No. 7, pp. 657-666, July 1984.

[Toum93] C. Toumazou, J. B. Hughes and N. C. Battersby (Eds.), *Switched-Currents: An Analogue Thechnique for Digital Technology.* Peter Peregrinus, London, England, 1993.

[Tsiv87] Yannis Tsividis, *Operation and Modeling of the MOS Transistor*. McGraw-Hill, 1987.

[Tsiv94] Y. P. Tsividis, "Integrated Continuous-Time Filter Design — An Overview". *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 3, pp. 210-216, March 1994.

[Twom00] Jerry Twomey, " Noise Reduction is Crucial To Mixed-Signal ASIC Design Success". *Electronic Design*. Vol. 48, No. 22, pp. 123-131, October 2000.

[Ulich87] Robert Ulichney, *Digital Halftoning*. MIT Press, Cambridge, Mass. 1987.

[VESA90] *VESA Monitor Timing Standard for 800 x 600 with 72Hz Refresh Rate.* No. VS900603. Video Electronics Standard Association, San Jose, CA, June 1999.

[VESA93] *Monitor Timing Standard for 800x600 with 72Hz and 1024x768 with 70Hz Refresh Rate*. Video Electronics Standard Association, San Jose, October 1993.

[VESA99] *VESA Digital Flat Panel. Version 1*. Video Electronics Standard Association, San Jose, CA, February 1999.

[Vitt91] E. Vittoz, H. Oguey, M. A. Maher, O. Nys, E. Dijikstra and M. Chevroulet, "Analog Storage of Adjustable Synaptic Weights" in *VLSI Design of Neural Networks*, pp. 47-63, Kluwer Academic, Boston, 1991.

[Warn74] R. M. Warner Jr. "Applying a Composite Model to the IC Yield Problem". *IEEE Jorunal of Solid State Circuits*. Vol. 9, No. 3,

pp. 86-95, June 1974.

[Werb91] F. werblin, “Synaptic Connections, Receptive Fields and Patterns of Activity in the Tiger Salamander Retina”, *Investigative Ophthalmology and Visual Science*, Vol. 32, No. 3, pp. 459-483, March 1991.

[Werb94] F. Werblin and A. Jacobs, “Using CNN to Unravel Space-Time Processing in the Vertebrate Retina”. *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pp. 33-40, Rome, Italy. December 1994.

[Werb95] F. Werblin, T. Roska and L. O. Chua, “The Analogic Cellular Neural Network as a Bionic Eye”. *International Journal of Circuit Theory and Applications*, Vol. 23, No. 6, pp. 541-69, November-December 1995.

[Werb96] F. Werblin, A. Jacobs and J. Teeters, “The Computational Eye”. *IEEE Spectrum*, Vol. 33, No. 5, pp. 30-37, May 1996.

[Wong96] Hon-Sum Wong, “Technology and Device Scaling Considerations for CMOS Imagers”. *IEEE Transactions on Electron Devices*, Vol. 43, No. 12, pp. 2131-2142, December 1996.

[Yama94] N. Yamashita et al. “A 3.84GIPS Integrated Memory Array Processor with 64 Processing Elements and 2-MB SRAM”. IEEE Journal of Solid-State Circuits, Vol. 29, No. 11, pp. 1336-1342, November 1994.

[Yang99] D. X. D. Yang, A. El Gamal, B. Fowler and H. Tian, “A 640 x 512 CMOS Image Sensor with Ultrawide Dynamic Range Floating-Point Pixel-Level ADC”. *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 12, pp. 1821-1834, December 1999.

[Yong96] Chai Yong-Yong and L. G. Johnson, “A 2 x 2 Analog Memory Implemented with a Special Layout Injector”. *IEEE Journal of Solid-State Circuits*, Vol.31, No. 6, pp. 856-859, June 1996.

[Zará98] Á. Zarándy, A. Stoffels, T. Roska and L.O. Chua, “Implementation of Binary and Gray-Scale Mathematical Morphology on the CNN Universal Machine”, *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 45. No. 2. pp. 163-168, February 1998.

Conclusiones

Esta Tesis representa una contribución al diseño de sistemas en muy alta escala de integración (VLSI) para procesamiento masivo de información de carácter sensorial. En particular, al análisis y diseño de circuitos integrados de alta complejidad para el procesamiento de imágenes y señal de video en tiempo real, basados en redes neuronales celulares (CNN). El estudio aquí realizado aborda inicialmente la problemática derivada de la realización física monolítica de estos sistemas de procesamiento desde el punto de vista de la arquitectura del sistema y de las limitaciones tecnológicas. A continuación, el estudio se adentra en el modelado y simulación de comportamiento de estos sistemas con la intención de facilitar el progreso en el flujo de diseño mediante herramientas de CAD optimizadas y un análisis suficientemente detallado de las estructuras que componen el sistema. Como refrendo a las metodologías propuestas se presentan dos prototipos, uno de un chip de memoria analógica (aRAM) con funciones de memoria caché dentro del sistema completo de procesamiento, y un segundo que implementa una máquina universal, en el sentido de Turing, de cómputo analógico, que muestra una dinámica compleja basada en modelos bio-inspirados. El desarrollo de estos chips ha permitido afrontar toda una serie de retos de diseño, en cuya resolución se han aplicado técnicas avanzadas de diseño de circuitos analógicos. Por tanto el análisis de estos sistemas de procesamiento masivo se ha realizado a diferentes niveles, que van desde la arquitectura del sistema hasta el análisis de la problemática específica de determinados bloques de circuito que implementan funciones de muy bajo nivel, pasando por el análisis de los algoritmos analógicos que establecen el comportamiento del sistema. Asimismo, en conexión con los prototipos diseñados, se han realizado propuestas a diferentes niveles. Desde el diseño de la temporización de *pipelines* de procesamiento paralelo hasta detalles concretos de bloques básicos de procesamiento analógico.

En este escenario, la aportación realizada por los diferentes trabajos desarrollados en la elaboración de esta Tesis puede desgranarse en los siguientes puntos:

- Se ha realizado una revisión de la problemática planteada por el procesamiento de imágenes en tiempo real, partiendo de un análi-

sis de los modelos de imagen y de sistema de procesamiento tradicionales, para proponer una arquitectura de sistema adaptada a las necesidades específicas del problema, que se beneficiaría, al mismo tiempo, de las ventajas en cuanto a consumo de potencia y área, y en cuanto a velocidad de procesamiento, que ofrecen los circuitos analógicos VLSI en aplicaciones en las que los requerimientos de precisión son moderados (alrededor de 7 bits).

- Se ha realizado una revisión de las redes neuronales celulares, por resultar un modelo de procesamiento altamente cualificado para la realización física monolítica de sistemas de procesamiento paralelo y masivo. En esta revisión hemos cubierto con el detalle suficiente, todos aquellos aspectos de las CNNs con los que el diseñador de circuitos va a poder interactuar con el fin de llegar a una implementación posible y competitiva del procesamiento de imágenes en tiempo real.

- Se ha realizado un estudio de las limitaciones tecnológicas que se encuentran en la realización VLSI de sistemas de procesamiento masivo y paralelo. En este sentido se ha modelado la influencia conjunta de defectos aleatorios en el proceso de fabricación, junto con las desviaciones de los parámetros de dicho proceso, encontrándose la existencia de un límite en el rendimiento de la producción (*yield*).

- Se han evaluado las diferentes alternativas para la implementación de un sistema de procesamiento de imágenes en tiempo real basado en CNNs, habida cuenta de las limitaciones tecnológicas encontradas, mediante el multiplexado espacial o temporal del hardware disponible. Como consecuencia de esta evaluación, se han planteado esquemas de temporización de la operación del *array* de procesadores que redundan en una mejor adaptación al problema y por tanto a una solución óptima.

- Se ha planteado la problemática derivada del modelado y la simulación de comportamiento de bloques de procesamiento analógicos complejos. En esta línea, se presentan diferentes trabajos de implementación de modelos y evaluación de la simulación con herramientas de CAD específicas, e inclusión de estas herramientas en el flujo de diseño de los algoritmos y de los circuitos integrados.

- Se han apuntado diferentes métodos de mejora de la robustez del algoritmo frente a errores o desviaciones en la implementación física. Por un lado se han examinado las posibilidades de asegurar la robustez de manera analítica. Por otro lado se ha establecido un procedimiento de centrado del diseño basado en la caracterización estadística del modelo de circuito mediante las herramientas desarrolladas *ad hoc*.

- Se ha propuesto un método de minimización de los requerimientos sobre el rango dinámico de los pesos en una realización concreta de una máquina universal de CNN. Se trata de encontrar una distribución óptima de fuerzas de las sinapsis, de tal forma que, para un conjunto específico de templates,

la precisión con la que deben realizarse los pesos es mínima. Como resultado se obtiene una ampliación notable del espacio de diseño, al tiempo que se incrementa la robustez del algoritmo.

- Se han revisado las dificultades y fuentes de error que aparecen en la implementación VLSI del almacenamiento de señales analógicas, junto con las diferentes alternativas tecnológicas para su realización, basándonos en el estudio de las características que definen la estructura y el comportamiento de dispositivos y circuitos que actúan como memoria analógica.

- Se ha propuesto, diseñado, fabricado y testado un chip prototipo de memoria analógica en una tecnología CMOS estándar de $0.5\mu m$ de resolución, una capa de polisilicio y tres capas de metal. Según los resultados obtenidos, este chip supone una alternativa competitiva a anteriores realizaciones del almacenamiento de señales analógicas en sistemas integrados.

- Se han propuesto soluciones específicas a los problemas de diseño surgidos en el desarrollo del chip, en relación con la integridad de las señales analógicas en un substrato en el que los circuitos analógicos cohabitan con circuitería digital de control.

- Se ha revisado un modelo inspirado en la operación de ciertas partes de la retina de los vertebrados, que implementado en una máquina universal basada en CNNs podría utilizarse para emular muchas de las funciones de la retina biológica. En especial, se han estudiado los detalles de la programación de la dinámica de la red para conseguir que esta realice determinados procesamientos de imágenes.

- Se han estudiado los bloques y operadores analógicos básicos necesarios para la implementación del modelo referido, una CNN de dos capas acopladas con memorias locales y otros elementos de procesamiento dentro de cada procesador elemental, en una tecnología CMOS estándar. Se han propuesto soluciones concretas para los problemas derivados de la implementación de una constante de tiempo escalable para una de las capas.

- Se ha revisado el problema de la homogeneidad de las referencias así como otras cuestiones derivadas de la realización de un chip de gran tamaño y se han establecido los métodos para minimizar los errores.

- Se ha diseñado y fabricado un chip prototipo en una tecnología CMOS estándar de $0.5\mu m$ de resolución, una capa de polisilicio y tres capas de metal, que implementa una máquina universal de cómputo basada en CNNs. Este circuito consta de un arreglo de 32×32 procesadores elementales de dinámica compleja con facilidades para el almacenamiento lógico y analógico y para el procesamiento lógico y aritmético locales.

Esta Tesis representa una contribución al diseño de sistemas en muy alta escala de integración (VLSI) para procesamiento masivo de información de carácter sensorial. En particular, al análisis y diseño de circuitos integrados de alta complejidad para el procesamiento de imágenes y señal de video en tiempo real, basados en redes neuronales celulares (CNN). El estudio aquí realizado aborda inicialmente la problemática derivada de la realización física monolítica de estos sistemas de procesamiento desde el punto de vista de la arquitectura del sistema y de las limitaciones tecnológicas. A continuación, el estudio se adentra en el modelado y simulación de comportamiento de estos sistemas con la intención de facilitar el progreso en el flujo de diseño mediante herramientas de CAD optimizadas y un análisis suficientemente detallado de las estructuras que componen el sistema. Como refrendo a las metodologías propuestas se presentan dos prototipos: un chip de memoria analógica (aRAM) con funciones de memoria caché y una máquina universal de cómputo analógico, con una dinámica basada en modelos bio-inspirados. El desarrollo de estos chips ha permitido afrontar toda una serie de retos en cuya resolución se han aplicado técnicas avanzadas de diseño de circuitos analógicos. Por tanto, el análisis de estos sistemas de procesamiento masivo se ha realizado a diferentes niveles, que van desde la arquitectura del sistema hasta el análisis de la problemática específica de determinados bloques de circuito que implementan funciones de muy bajo nivel, pasando por el análisis de los algoritmos analógicos que establecen el comportamiento del sistema. Asimismo, en conexión con los prototipos diseñados, se han realizado propuestas a diferentes niveles. Desde el diseño de la temporización de *pipelines* de procesamiento paralelo hasta detalles concretos de bloques básicos de procesamiento analógico.

UNIVERSIDAD de SEVILLA

Instituto de Microelectrónica de Sevilla