SIMULADOR DE SISTEMAS AER BASADO EN EVENTOS

J. A. Pérez-Carrasco[(1)], C. Serrano[(2)], B. Acha[(2)], T. Serrano-Gotarredona[(1)], and B. Linares-Barranco[(1).]

jcarrasco@imse.cnm.es, cserrano@us.es, bacha@us.es, terese@imse.cnm.es, bernabe@imse.cnm.es

[(1)] Instituto de Microelectrónica de Sevilla (IMSE-CNM-CSIC) Avda. Reina Mercedes, s/n, Sevilla. CP41012.

[(2)] Dpto. Teoría de la Señal, ETSIT, Universidad de Sevilla. Avda de los Descubrimientos, s/n, CP41092

***Abstract*- Address-Event-Representation (AER) is a communications protocol for transferring (visual) information between chips, originally developed for bio-inspired vision and audition systems. Such systems may consist of a complicated multi-layer hierarchical structure with many chips that transmit events among them in real time, while performing some complex processing (for example, convolutions, competitions, etc). This sensing and processing technology is capable of very high speed throughput, because it does not rely on sensing and processing sequences of frames, and because it allows for complex hierarchically structured cortical-like layers for sophisticated processing.**

In this paper we present an effective tool that simulates the behaviour of such kind of structures. AER stream sources are fed to the software simulation tool and AER streams at all nodes of the network are computed. The tool has been developed in MATLAB and is event driven. It has been conceived as an open tool, so that any user can add extra functional blocks easily, or provide more elaborate or more simplified descriptions of already available blocks.

I. INTRODUCCIÓN

El protocolo AER fue propuesto por primera vez en 1991 por Sivilotti [1] para replicar el estado de un array de píxeles (neuronas o celdas) de un chip (emisor) en uno o múltiples chips (receptores) [2]. Es muy fácil añadir procesamiento digital extra mientras los datos están siendo transmitidos entre los chips [3]. La estructura puede ser expandida modularmente a estructuras de chips multietapa [4].

En una comunicación AER tradicional punto a punto [5], el estado de un chip emisor que contiene un array de celdas (como, por ejemplo, una cámara o un chip retina artificial), se replica en otro chip. La Fig. 1 explica las bases de una comunicación basada en AER. El chip emisor contiene un array de neuronas donde cada píxel o celda muestra un estado variante con el tiempo con una constante de variación temporal muy pequeña (del orden de *ms*). Este estado continuo en el tiempo es transformado en pulsos digitales (eventos) de anchura mínima (del orden de *ns*). El intervalo temporal entre eventos está en el orden de *ms*. Este hecho permite una ventajosa multiplexación en el tiempo. Cada vez que una neurona produce un pulso, su dirección o código se escribe en un bus inter-chip de alta velocidad haciendo uso de una comunicación asíncrona. Un elemento arbitrador en el chip resuelve las posibles colisiones debidas a neuronas que producen pulsos al mismo tiempo [5]. El bus inter-chip de alta velocidad mostrará un flujo continuo de direcciones de píxeles. El chip receptor lee las direcciones presentes en el bus, y dirige cada dirección (evento) al píxel con la misma coordenada. Este píxel recibirá el evento y lo integrará, recuperando de este modo la actividad original variante con el tiempo del píxel correspondiente emisor.

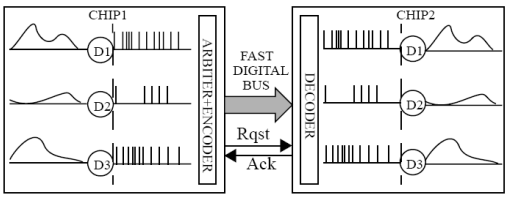


Fig. 1. Enlace de comunicación AER punto a punto

Esta es la comunicación inter-chip más simple basada en AER. Sin embargo, esta comunicación punto a punto puede ser extendida fácilmente a un esquema multireceptor [2], de modo que, por ejemplo, un chip retina emisor pueda enviar su actividad a muchos chips en paralelo de procesamiento. También, usando esquemas más complicados, múltiples chips emisores pueden multiplexar sus salidas y enviarlas a un conjunto menor de chips de procesamiento. Además, la información visual puede ser trasladada o rotada fácilmente simplemente cambiando las direcciones de los eventos al tiempo que viajan de un chip al siguiente [3].

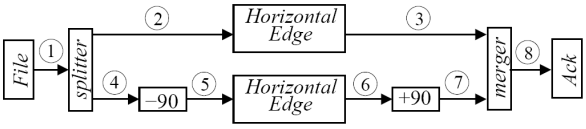
Existe una creciente comunidad de usuarios del protocolo AER para el diseño de aplicaciones de visión y audición bio-inspiradas, como ha sido demostrado por el éxito en los últimos años de los participantes en las '*Neuromorphic Engineering Workshop series*' [7]. El éxito de esta comunidad es diseñar sistemas grandes jerárquicamente estructurados multi-chip multi-etapa capaces de implementar procesamientos complejos de matrices en tiempo real. El éxito de tales sistemas dependerá grandemente de la disponibilidad de herramientas robustas y eficientes de diseño y depuración de sistemas AER [8]-[9].

Al mismo tiempo, la disponibilidad de un simulador para diseñar y concebir sistemas de procesamiento complejos basados en AER se está convirtiendo en una necesidad de fuerte importancia para la comunidad AER. Obviamente, tal simulador debería tomar ventaja de la naturaleza basada en eventos de los sistemas basados en AER, y evitar la filosofía tradicional presente en muchos simuladores de resolver un conjunto de ecuaciones diferenciales que describen el sistema por completo y actualizan el estado del mismo en cada paso de tiempo. En este artículo presentamos un simulador basado en MATLAB sencillo y abierto para estudiar complejos sistemas AER.

II. DESCRIPCIÓN DEL SIMULADOR

En este simulador, un sistema genérico AER es descrito mediante un *netlist* o fichero de conexiones que usa solamente dos tipos de elementos: *módulos* y *canales*. Un *módulo* es un bloque que genera y/o produce trenes de eventos (*streams*) AER. Por ejemplo, un chip retina sería una fuente que proporciona eventos AER a un sistema AER. Un chip de convolución [10] sería un módulo de procesamiento AER que recibiría como entrada un stream de eventos AER y que produciría un nuevo stream de eventos AER a la salida. Un splitter sería un módulo especial que replica los eventos recibidos por el puerto de entrada en varios puertos de salida. Similarmente, un merger es otro módulo especial que recibiría varios streams AER y los multiplexaría en un único stream AER en el puerto de salida. Los streams AER constituyen los nodos del netlist en un sistema AER y son llamados *canales*. La herramienta de simulación impone la restricción de que un *canal* conecta una única salida AER de un módulo con una única entrada AER de otro (o del mismo) módulo. De este modo, los canales representan conexiones punto a punto. Para replicar o multiplexar canales, se deben incluir en el netlist módulos splitter o merger.

La Fig. 2 muestra un netlist de ejemplo y su descripción mediante un archivo ASCII.




```
%First, we declare sources to the system
%              SOURCES          SOURCES_DATA
sources          {1}               {s1}

%Next, we declare priorities
priorities     {0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2}

%Next, we declare blocks
%NAME            IN-CHANNELS   OUT-CHANNELS  PARAMS      STATES
splitter         {1}           {2,4}         {params1}   {state1}
n_sobel          {2}           {3}           {params2}   {state2}
imrotate90       {4}           {5}           {params3}   {state3}
n_sobel          {5}           {6}           {params4}   {state4}
imrotate-90      {6}           {7}           {params5}   {state5}
merger           {3,7}         {8}           {params6}   {state6}
ack_only         {8}           {8}           {params7}   {state7}
```

Fig. 2. (arriba) Ejemplo de diagrama de módulos de un sistema AER. (abajo) Fichero Netlist que lo describe.

El netlist contiene 7 módulos y 8 canales. La descripción del netlist es proporcionada a la herramienta de simulación mediante un fichero de texto, que se muestra en la parte inferior de la Fig. 2. El canal 1 es un canal fuente. Todos sus eventos están disponibles a priori como archivo de entrada al simulador. Puede haber un número arbitrario de canales fuente en el sistema. Cada canal fuente necesita una línea en el fichero netlist, empezando con la palabra reservada *sources*, seguida por el número de canal y el nombre del archivo que contiene sus eventos. Una línea que empiece con la palabra reservada *priorities* proporciona un número de prioridad para cada canal. Su uso será explicado más adelante. Las líneas siguientes describen cada uno de los módulos, con una línea por cada módulo presente en el sistema. El primer campo que aparece en la línea es el nombre del módulo, seguido por sus canales de entrada, canales de salida, nombre de la estructura que contiene sus parámetros de configuración, y nombre de la estructura que contiene sus variables de estado. Cada módulo es descrito mediante una función MATLAB cuyo nombre es el propio nombre del módulo. El simulador no impone ninguna restricción en el formato de las estructuras de parámetros y estados. Estos formatos están abiertos al usuario que escribe el código de la función de cada módulo. El simulador sólo necesita saber el nombre de los archivos donde están almacenadas estas estructuras.

Los canales están descritos mediante matrices de dos dimensiones. Cada fila en la matriz corresponde a un evento y tiene seis componentes

$$\left\lfloor x, y, sign, T_{preRqst}, T_{Rqst}, T_{Ack} \right\rfloor \quad (1)$$

'x' e 'y' representan las coordenadas o direcciones del evento y 'sign' es el signo del evento. '$T_{preRqst}$' representa el tiempo en el cual el evento fue creado por el módulo emisor, 'T_{Rqst}' representa el tiempo en el cual el evento es procesado por el módulo receptor y 'T_{Ack}' representa el tiempo en el cual el evento es finalmente asentido por el módulo receptor. En nuestra aplicación distinguimos entre tiempo pre-Request $T_{preRqst}$ y tiempo de Request efectivo T_{Rqst}. El primero sólo depende del módulo emisor, mientras que el segundo requiere que el módulo receptor esté preparado para procesar la señal *request* de un evento. De este modo, podemos proporcionar como fuente al sistema una lista entera de eventos que están descritos sólo mediante sus direcciones, signo y tiempos $T_{preRqst}$. Una vez que los eventos son procesados por el simulador, sus tiempos T_{Rqst} y T_{Ack} son establecidos.

Antes de empezar la simulación, los eventos de los canales fuente deben ser proporcionados mediante un fichero de entrada. Durante la simulación, se procesan los eventos pertenecientes a los canales fuente y nuevos eventos pueden ser creados por los módulos de procesamiento y enviados a sus canales de salida. Tras la simulación, el usuario puede visualizar los flujos de eventos resultantes en los diferentes canales.

La ejecución del simulador es como sigue. Inicialmente se lee el archivo netlist conjuntamente con todos los ficheros de parámetros y variables de estado pertenecientes a los diferentes módulos. A partir de este punto, el programa entre en un bucle de procesamiento continuo que realiza los siguientes pasos:

1. Todos los canales son examinados. El simulador selecciona aquél con el evento no procesado con menor tiempo de creación. Se considera que un evento no ha sido procesado cuando solamente su tiempo pre-Request $T_{preRqst}$ ha sido establecido, pero no el tiempo de Request T_{Rqst} ni el tiempo de asentimiento T_{Ack}.

   En caso de que varios canales tengan eventos sin procesar con el mismo tiempo pre-Request, se acudirá a la información de prioridad de canal proporcionada en el archivo netlist (*priorities*, ver Fig. 2). Esta línea asigna un valor de prioridad a cada canal. El evento perteneciente al canal con el valor de prioridad más alto será procesado primero. Nótese que eventos pertenecientes a canales diferentes pueden ocurrir simultáneamente, ya que físicamente los canales son independientes. Este vector de prioridades es usado solamente por el simulador, pero no altera el comportamiento final del sistema.
2. Una vez que un canal es seleccionado para procesamiento, la información de su evento con inferior tiempo de creación es suministrado como entrada al

módulo con el cual el canal está conectado. En ese momento, el módulo actualiza su estado interno a partir de la información recibida del evento y del resto de parámetros de configuración del módulo. Este hecho puede ocasionar la generación de nuevos eventos no procesados en el módulo que serán proporcionados a los puertos de salida correspondientes del módulo. Estos eventos tendrán información de dirección, signo y tiempo de creación, pero no tendrán asignados (por haber sido creados, no procesados) valores en los tiempos T_{Rqst} y T_{Ack}. A partir de este momento, el simulador actualiza todos los canales, almacena el nuevo estado para el módulo que se acaba de procesar y regresa al punto 1.

La funcionalidad de un módulo es descrita mediante una función MATLAB independiente cuyo nombre es idéntico al que aparece en el netlist. Un usuario puede añadir y describir nuevos módulos si lo necesita. La única restricción impuesta es respetar el formato de llamada de la función

$$[Event\_In, Events\_Out, State\_New, Time\_out] = \\ = function(Event\_In, Params, State, Time\_In) \quad (2)$$

'Event_In' corresponde a la información del evento entrante al módulo (ver Eq. (1)). A la salida, el evento tiene actualizados los valores correspondientes a los tiempos de Request efectivo y tiempo de asentimiento. 'State' y 'State_New' representan las variables de estado internas al módulo antes y después del procesamiento del evento respectivamente. 'Time_In' y 'Time_Out' son los tiempos globales del sistema para este módulo antes y después del procesamiento. Y 'Event_Out' es una lista de eventos de salida producidos por el módulo en sus correspondientes canales de salida. Estos eventos de salida (que están aún sin procesar) son incluidos por el simulador en los canales correspondientes, y serán procesados en un tiempo posterior por los módulos de destino correspondientes a esos canales.

Como ilustración, el código de algunos de los módulos más elementales se muestra a continuación. El módulo *splitter* simplemente replica el evento en su puerto de entrada en los puertos de salida, introduciendo un retraso (descrito en los parámetros de entrada del módulo):

```
function [channel,out,state,timeact]=splitter(channel,params,state,timeact)
% LEE LOS PARÁMETROS PARA ESTE MÓDULO
delay_time=params.delay;
tdelayack=params.tack;
% OBTÉN COORDENADAS Y SIGNO DEL EVENTO
x=channel(1);
y=channel(2);
sig=channel(3);
% CALCULA TIEMPOS Rqst Y Ack
tpre=timeact+delay_time;
tack=timeact;
% ESTABLECE VALOR DE TIEMPO Ack EN EL EVENTO ENTRANTE
channel(6)=timeact+tdelayack;
% COPIA EL EVENTO DE ENTRADA EN EL CANAL DE SALIDA
out=[x y sig tpre 0 -1];
```

El evento de entrada es copiado a la salida. Un valor de tiempo de asentimiento T_{Ack} se le asigna al evento, y un valor igual a '-1' se le da al evento de salida. De este modo, el simulador sabe que los canales correspondientes a estos eventos de salida tienen que ser procesados.

El módulo *merger* copia todos los eventos en los puertos de entrada al único puerto de salida, introduciéndoles un pequeño retraso. El código es igual que el descrito para el módulo *splitter*, lo único que cambia es el nombre del módulo y el sentido de los eventos. En el módulo *splitter*, los eventos proceden de un único canal y se replican en los canales de salida. Sin embargo, en el módulo *merger* hay varios canales de entrada y cada uno de los eventos correspondientes a uno de estos canales se replica en el único canal de salida. El formato de llamada a este módulo sería el siguiente:

```
function [channel,out,state,timeact]=merger(channel,params,state,timeact)
```

El módulo *Acknowledge_only* asigna simplemente un valor temporal T_{Ack} a los eventos entrantes, de modo que la comunicación no se mantenga en espera:

```
function [channel,out,state,timeact]=ack_only(channel,params,state,timeact)
out=[];
channel(6)=timeact;
```

El módulo extractor de bordes horizontales implementa simplemente un campo de proyección. Para cada evento entrante en la coordenada *(x,y)* se genera un campo de proyección de eventos (máscara de convolución) de tamaño 3x3 alrededor de esta coordenada. Esto es equivalente a obtener en la salida la convolución entre los eventos de entrada y la máscara de convolución. La máscara de convolución es la siguiente:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (3)$$

El módulo genera 8 eventos de salida para cada evento de entrada: uno de coordenadas *(x-1, y+1)* y signo '+', otro de coordenadas *(x-1,y-1)* y signo '-', dos para las coordenadas *(x, y+1)* y signo '+' y así sucesivamente. Como éste es un módulo idealizado, no se introducen retrasos. En una situación más realista, cada píxel debería ser implementado como una neurona de integración y disparo con un determinado umbral y se deberían considerar retrasos realistas correspondientes a una implementación física del sistema.

III. RESULTADOS

A continuación se muestra un ejemplo de simulación para el sistema descrito por el esquemático y netlist de la Fig. 2. El canal 1 es un canal fuente que contiene todos los eventos que describen la imagen de entrada estática mostrada en la Fig. 3(a). La imagen es de tamaño 128x128 y 16 niveles de gris. El flujo de eventos del canal 1 ha sido obtenido usando el método uniforme [11] y se han generado un total de 150K eventos. Si la frecuencia máxima de píxel es 1*Kevento por segundo* (para el píxel más brillante), entonces todos esos eventos serán transferidos en 16*ms*, lo que conlleva una tasa promedia de eventos de 9.38*Meventos por segundo* para este canal.

Los módulos en la Fig. 2 son un *'splitter'*, un *'merger'*, dos *'rotators'* (rotan la coordenada del evento), y dos detectores de bordes horizontales (*'horizontal sobel edge detectors'*). En primer lugar, el flujo de información visual del Canal 1 es replicado en los Canales 2 y 4 para implementar procesamientos independientes. El flujo AER en el Canal 2 es alimentado a un detector de bordes horizontales, cuya salida aparece en el Canal 3 y se muestra en la Fig. 3(b). El Canal 3 tiene un total de 1.2M eventos. El

flujo del Canal 4 es rotado primero -90º, después procesado por otro detector horizontal idéntico al anterior y finalmente rotado de nuevo +90º, generándose un flujo nuevo AER de información visual en el Canal 7, mostrado en la Fig. 3(c). El Canal 7 tiene un total de 1.2Meventos. Finalmente, los flujos de los canales 3 y 7 son reunidos con el módulo *merger* en el Canal 8, resultando en la información visual mostrada en la Fig. 3(d). El canal 8 tiene un total de 2.4Meventos. El simulador computa los flujos de información en los diferentes canales usando la información de los canales fuente y de los módulos. Al final de la simulación, el simulador proporciona la lista de eventos que han aparecido en todos los canales con sus tiempos finales Reques*t* y Acknowledge. Una vez que todos los eventos de todos los canales han sido procesados, éstos pueden ser visualizados como imágenes 2D. En este caso particular, como el flujo de eventos de entrada al sistema en el Canal 1 representa una imagen estática, todos los flujos de todos los canales representan imágenes estáticas también. Para analizar o visualizar la información contenida en los eventos que viajan a través de los diferentes canales, se requiere una rutina de visualización. En general, los flujos de eventos representan información visual correspondiente a objetos en movimiento, de modo que la rutina de visualización necesita saber una tasa de frames (o tiempo de frame) para reconstruir una imagen a partir de los eventos que pertenezcan a un tiempo de frame determinado.

IV. CONCLUSIONES

Se ha introducido el concepto de AER y el procesamiento de imágenes basado en eventos. Esta tecnología permite trabajar a muy altas tasas de procesamiento sin la necesidad de esperar un tiempo de frame artificial. Es factible el procesamiento complejo de imágenes en paralelo y en tiempo real. Se muestra un ejemplo muy simple de simulación a partir de una imagen estática. A pesar de la simplicidad del ejemplo, es posible observar el esquema de procesamiento totalmente diferente a los esquemas clásicos basados en frames y cómo no es necesario disponer de un frame completo a la entrada del sistema para empezar a procesar la imagen y obtener resultados en la salida. Se ha diseñado una herramienta de simulación de software abierto y basada en eventos para simular sistemas AER jerárquicamente estructurados. Un netlist genérico de canales AER y módulos se define a partir de un simple fichero de texto. Uno de los objetivos de la herramienta es que sea transparente al usuario, de modo que los usuarios pueden añadir muy fácilmente descripciones de nuevos módulos. El trabajo en el futuro se concentrará en la implementación de una librería de módulos de procesamiento AER y en el diseño de aplicaciones bastante más complejas y funcionales pero siguiendo el esquema de procesamiento mediante eventos y no los clásicos esquemas de procesamiento basados en frames.

AGRADECIMIENTOS

Este trabajo ha sido financiado en parte por los proyectos españoles 2006-11730-C03-01 (Samanta2) y el proyecto EU IST-2001-34124 (Caviar). JAPC ha sido financiado por el proyecto del gobierno andaluz P06-TIC-01417 (Brain System).

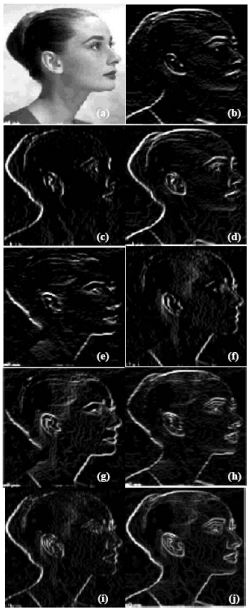


Fig. 3. (a) Imagen de entrada usada como fuente en el Canal 1, visualización de eventos positivos en Canal 3 (b) (hor), (c) Canal 7 (vert), (d) canal 8 (hor+vert); de eventos negativos en Canal 3 (e) (hor), (f) Çanal 7 (vert), (g) Canal8 (hor+vert); eventos positivos más negativos en (h) Canal 3(hor), (i) Canal 7 (vert), y (j) Canal 8 (hor+vert).

REFERENCES

[1] M. Sivilotti, Wiring Considerations in analog VLSI Systems with Application to Field-Programmable Networks, *Ph D.Thesis*, California Institute of Techonology, Pasadena CA, 1991.

[2] J. P. Lazzaro and J.Wawrzynek, "A Multi-sender asynchronous extension to the address-event protocol," *16th Conference on Advanced Research in VLSI*, W. J. Dally, J. W. Poulton, and A. T. Ishii (Eds.), pp. 158-169,1995.

[3] D. H. Goldberg, G. Cauwenberghs, and A. G. Andreou, "Analog VLSI spiking neural network with address domain probabilistic synapses," *Proc. Of the IEEE 2001 Int. Symp. Circ. Syst* (ISCAS 2001), vol. 2, pp. 241-244, May 2001.

[4] T. Serrano-Gotarredona, A. G. Andreou, and B. Linares-Barranco, "AER image filtering architecture for vision processing Systems," *IEEE Trans. Circuits and Systems Part-II*, vol. 46, No. 9, pp. 1064-1071, September 1999.

[5] K. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Trans. on Circuits and Systems Part-II*, vol. 47, No. 5, pp. 416-434, May 2000.

[6] W. Maass, and C. M. Bishop (Eds.), *Pulsed Neural Networks*, MIT Press, Boston MA, 1999.

[7] A. Cohen, R. Etienne-Cummings, P. Hasler, T. Horiuchi, G. Indiveri, S. Shamma, A. Van Shaik, R. Douglas, C. Koch and T. Sejnowski, *Report on the 2005 Workshop on Neuromorphic Engineering*, Telluride, CO, June 26 to July 16, 2005.

[8] F. Gomez-Rodriguez, R. Paz, A. Linares-Barranco, M. Rivas, L. Miro, S. Vicente, G. Jimenez, A. Civit, "AER tools for Communications and Debugging," *IEEE 2006 Proc. Int. Symp. Circ. Syst.,* (ISCAS 2006), pp. 3253-3256, Kos (Greece), May 2006.

[9] E. Chicca, V. Dante, A. M. Whatley, P. Lichtsteiner, T. Delbrück, G. Indiveri, P. Del Giudice, and R. J. Douglas, "Multi-chip pulse based neuromorphic infrastructure and its application to a cortical model of orentiation selectivity," *IEEE Transactions on Circuits and Systems I,* 54(5):981-993*,* 2007.

[10] R.Serrano-Gotarredona, T.Serrano-Gotarredona, A.J.Acosta-Jiménez and B.Linares-Barranco, "An Arbitrary Kernel Convolution AER-Transceiver Chip for Real-Time Image Filtering," *IEEE 2006 Proc. Int. Symp. Circ. Syst.,* (ISCAS 2006),pp. 3145-3148, Kos (Greece), May 2006.

[11] A. Linares-Barranco, G. Jimenez-Moreno, B. Linares-Barranco and A. Civit-Ballcels, "On Algorithmic Rate-Coded AER Generation," *IEEE Transactions on Neural Networks*, vol. 17, No. 3, pp. 771-788, May 2006.