Proyecto Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Desarrollo de un sistema hardware de control basado en tecnologías embebidas para aplicaciones de convertidores electrónicos de potencia.

Autor: María Muñoz Quijada
Tutor: Sergio Vázquez Pérez

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Desarrollo de un sistema hardware de control basado en tecnologías embebidas para aplicaciones de convertidores electrónicos de potencia.

Autor:
María Muñoz Quijada

Tutor:
Sergio Vázquez Pérez
Profesor contratado

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia y amigos el apoyo y dedicación mostrados durante todos estos años de estudios, de alegrias y desilusiones, sacrificios y recompensas. Nada de esto hubiera sido posible sin su ayuda.

También me gustaría agradecer a los integrantes del departamento de electrónica por la acogida en su zona de trabajo durante el proceso de elaboracion de este proyecto, en especial a mi tutor, Sergio Vazquez Pérez, y a Abraham Márquez Alcaide, por la paciencia mostrada y la ayuda proporcionada, que sin duda ha sido clave para el desarrollo del proyecto.

Por último, quiero dar las gracias a mis profesores por todo lo que he podido aprender durante estos años gracias a sus explicaciones.

Índice General

Lista de símbolos y acotaciones

ACIB: Analog common interface bus.

ADC: Analog to digital Converter.

CAN: Controller Area Networks.

CCS: Code Composer Studio.

CPU: Central Processing Unit

CRC: Comprobación de redundancia cíclica.

DAC: Digital to Analog Converter.

DIMM: Dual in-line memory module.

DSP: Digital Signal Processor.

ePWM: enhanced pulse width modulation.

FIFO: First input first output.

FPU: Floating-Point Unit.

GPIO: General purpose input output.

I2C: Inter-Integratted circuit.

IGBT: Insulated Gate Bipolar Transistor.

ISO: Organización internacional de normalización.

JTAG: Joint test action group.

LED: Light-emitting diode.

LSB: Least significant bit

McBSP: Multichannel Buffered serial port.

MCU: Microcontrolador.

MSB: Most significant bit.

NRZ: Non-return to zero.

PCB: Printed Circuit Board

PWM: Pulse width modulation.

RAM: Random access memory.

RS232: Recommended standard 232.

SAR: Successive approximation register.

SCI: Serial communication interface.

SOC: Start of Conversion.

SPI: Serial Peripheral Interface.

UART: Universal asynchronous receiver-transmitter.

USB: Universal Serial Bus.

Lista de ilustraciones y tablas

1 INTRODUCCIÓN

En el documento aquí expuesto, se quiere mostrar la configuración a realizar del DSP Concerto de Texas Instruments para el control de un convertidor de potencia trifásico, que puede hacer las funciones de inversor o rectificador según se realice la conexión, y cuyo hardware fue diseñado en el departamento de ingeniería electrónica de la escuela.

Este convertidor consta de una serie de IGBT's cuyo disparo ha de ser controlado mediante un MCU, el cual se decidió inicialmente que fuera el DSP Delfino, fabricado y diseñado por la misma empresa electrónica que el DSP objeto de este proyecto.

Tras la implantación del DSP en el convertidor mencionado, se decidió hacer un cambio del dispositivo, ya que, tras el uso del mismo, se observaban interferencias en las medidas de corriente tomadas a partir del ADC (convertidor analógico-digital) y que las comunicaciones externas con el dispositivo no eran todo lo rápidas que se querían, debido al alto rendimiento computacional que se le exigía al MCU. Además, la disposición de los pines en la placa del dispositivo con respecto a la placa de control que permite la conexión entre el DSP y el convertidor no parecía la más adecuada.

El sustituto propuesto, Concerto, es un sistema que consta de dos núcleos: un subsistema de control en tiempo real (control C28 subsystem) y un subsistema maestro (master M3 subsystem).

Esto permite solucionar los problemas de retrasos, ya que se podría enfocar el M3 a las comunicaciones y el C28 al control, de manera que en lugar de tener un procesador encargándose de varias tareas a la vez, se tiene a cada núcleo encargado de tareas más específicas. Así mismo, dispone de un ADC (convertidor analógico digital) externo a ambos núcleos a diferencia del DSP utilizado anteriormente (Delfino), por lo que se evitan así las interferencias observadas por proximidad del ADC al procesador.

Este proyecto tiene como objetivo final la familiarización con el dispositivo en cuestión, en cuanto a programación, conexiones internas y externas y comunicación con otros dispositivos, centrándose en la configuración hardware y software del mismo para realizar las tareas de control de los disparos del convertidor mediante los módulos PWM, obtención de medidas a través del ADC para su posterior tratamiento y comunicación entre el dispositivo y la interfaz Labview, utilizando el subsistema C28 tanto para el control como para las comunicaciones y partiendo del código implementado en el dispositivo antiguo.

Como ampliación futura, se pretende que sea el subsistema M3 el encargado del tratamiento de las comunicaciones, de manera que el subsistema C28 se encargue únicamente del control y se consiga obtener así el rendimiento óptimo del dispositivo elegido.

2 FAMILIARIZACIÓN DEL ENTORNO Y COMPONENTES UTILIZADOS

Un sistema embebido es un dispositivo electrónico con capacidad de procesamiento (incluye una CPU) y conexión en red que es capaz de interactuar con sistemas similares.

Estos son utilizados en aplicaciones industriales, ya sea, en sistemas automóviles, en el control de energías renovables o para comunicaciones de red.

La arquitectura de estos sistemas se podría resumir en la de un sistema microprocesador (MCU), el cual está compuesto por un Maestro, que es capaz de ejecutar instrucciones y programas y manejar datos, unos buses de transmisión de datos y ordenes y una serie de esclavos o periféricos que se comunican con el maestro y lo complementan a la hora de realizar tareas.

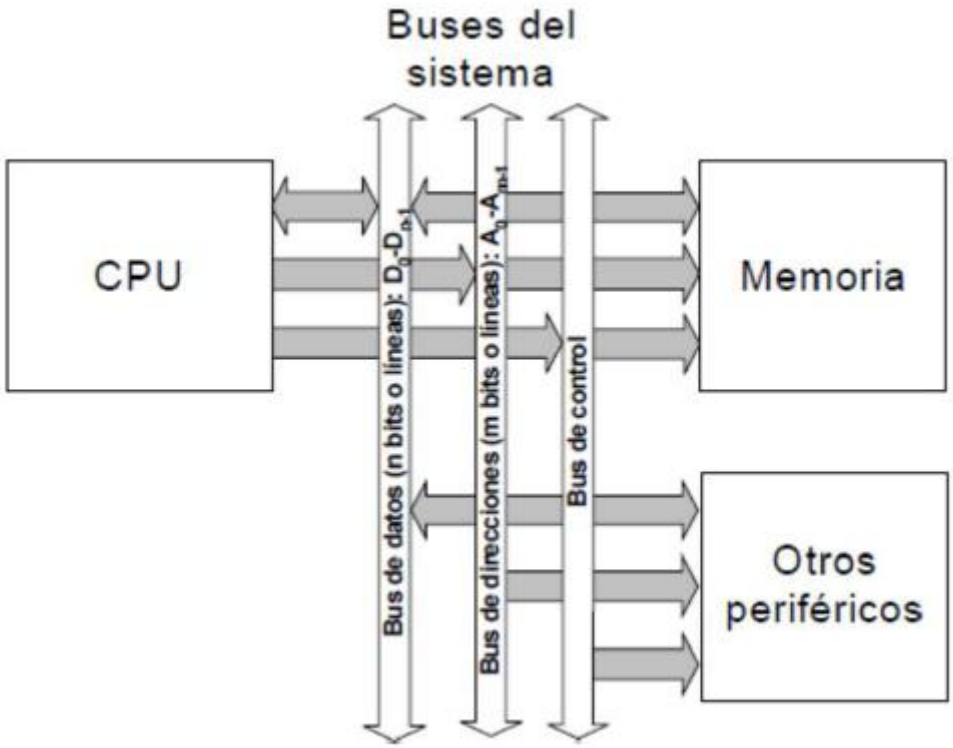


Ilustración 1: Arquitectura.

A la hora de trabajar con sistemas embebidos, es importante tener un conocimiento general del funcionamiento del sistema (periféricos que dispone, configuración de los mismos, conexiones...) a través del data sheet que proporciona el fabricante, y posteriormente profundizar en el conocimiento de aquellos periféricos que se necesitan utilizar para obtener el funcionamiento deseado del sistema. Es necesario también saber manejar distintos programas informáticos, como compiladores, máquinas virtuales o sistemas de programación grafica, que permiten configurar el dispositivo según se requiera, o visualizar los datos que el sistema proporciona mediante un ordenador.

En el siguiente apartado se explicaran las características del DSP elegido y las herramientas que se utilizaron para facilitar el aprendizaje del uso del mismo.

Además se expondrán los distintos equipos de laboratorio y materiales auxiliares que fueron utilizados para realizar las pruebas necesarias para comprobar el funcionamiento del sistema, así como una serie de software que fueron útiles para desarrollar el proyecto.

2.1 Dispositivo

Como se comentó al inicio del documento, el dispositivo que será objeto de este proyecto es el DSP F28M35H52C, cuyo nombre comercial es Concerto, perteneciente a la gama C2000 de Texas Instruments.

A continuación se exponen las características de este sistema y una de las herramientas que ofrece Texas Instruments para aprender a utilizar el DSP.

2.1.1 Características

Con el DSP Concerto, Texas Instruments ofrece una solución para aunar rendimiento, sencillez y precio, ya que en un mismo dispositivo se encuentran dos subsistemas independientes que son capaces de comunicarse entre sí, con lo que se evita dificultad en el software y el compromiso entre características y funcionamiento que supondría el uso de un solo MCU, y por otro lado, la complejidad, el alto precio y el elevado tiempo de respuesta que supondría el uso de dos MCU.

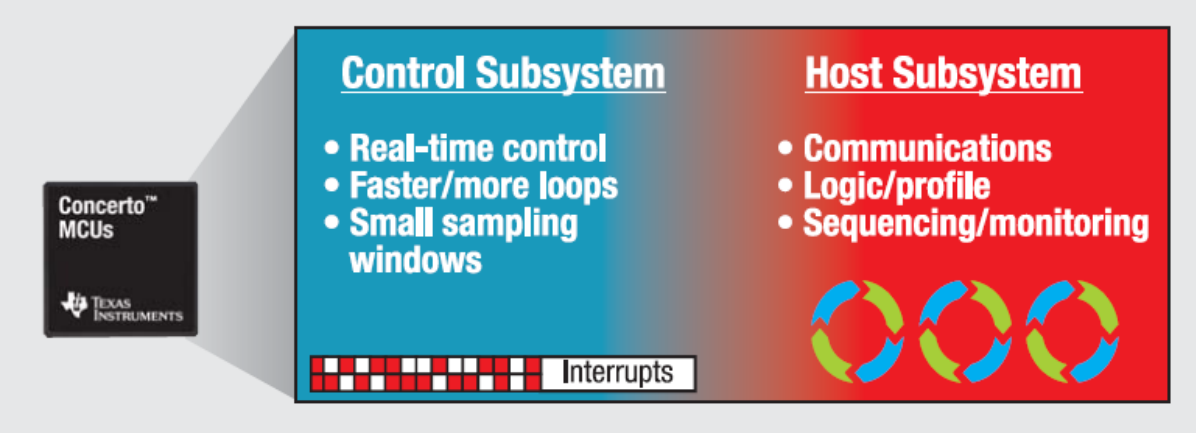


Ilustración 2: Concerto MCU solucion.

El dispositivo cuenta con un subsistema maestro M3 (ARM Cortex) para propósito general y comunicaciones, las cuales puede realizar a través de los distintos puertos que reúne (Ethernet, USB, CAN, puertos serie…).

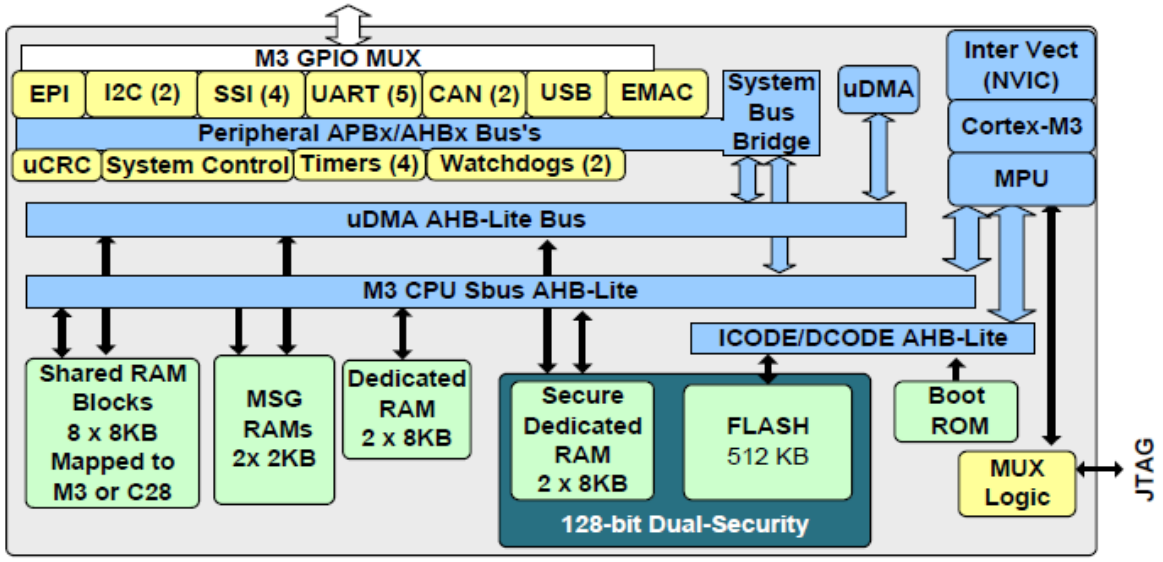


Ilustración 3: Estructura del subsistema maestro M3.

Por otra parte, el subsistema de control (TMS320C28x) permite una alta velocidad gracias a la FPU que incorpora. Además cuenta con 9 módulos PWM lo que resulta bastante útil para aplicaciones de electrónica de potencia. Al igual que el subsistema maestro, es capaz de realizar tareas de comunicación a través de los puertos serie (SPI, SCI).

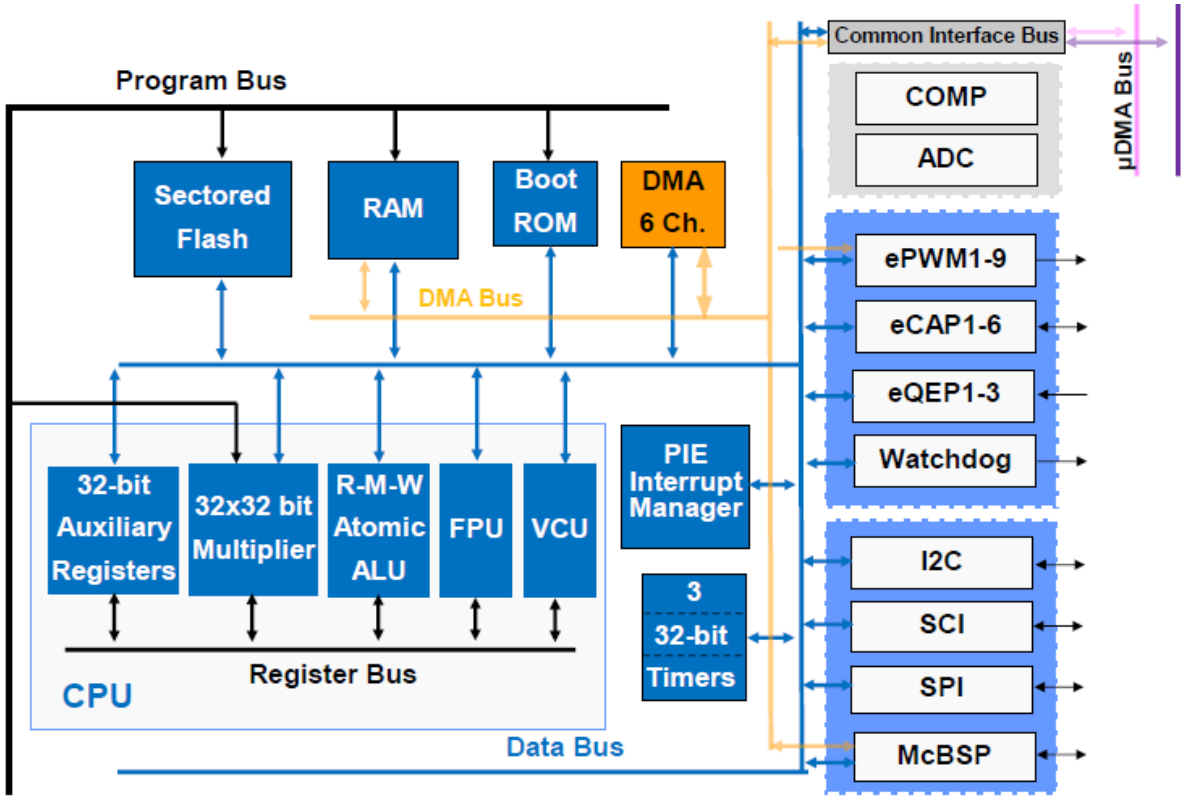


Ilustración 4: Arquitectura del subsistema C28.

Una de las razones por las que se elige este dispositivo es el subsistema analógico que ofrece, el cual es externo y accesible por ambos núcleos mediante el ACIB (Analog common interface bus) y cuenta con 2 módulos ADC de 12 bits y 6 comparadores con 6 DAC de 10 bits internos.

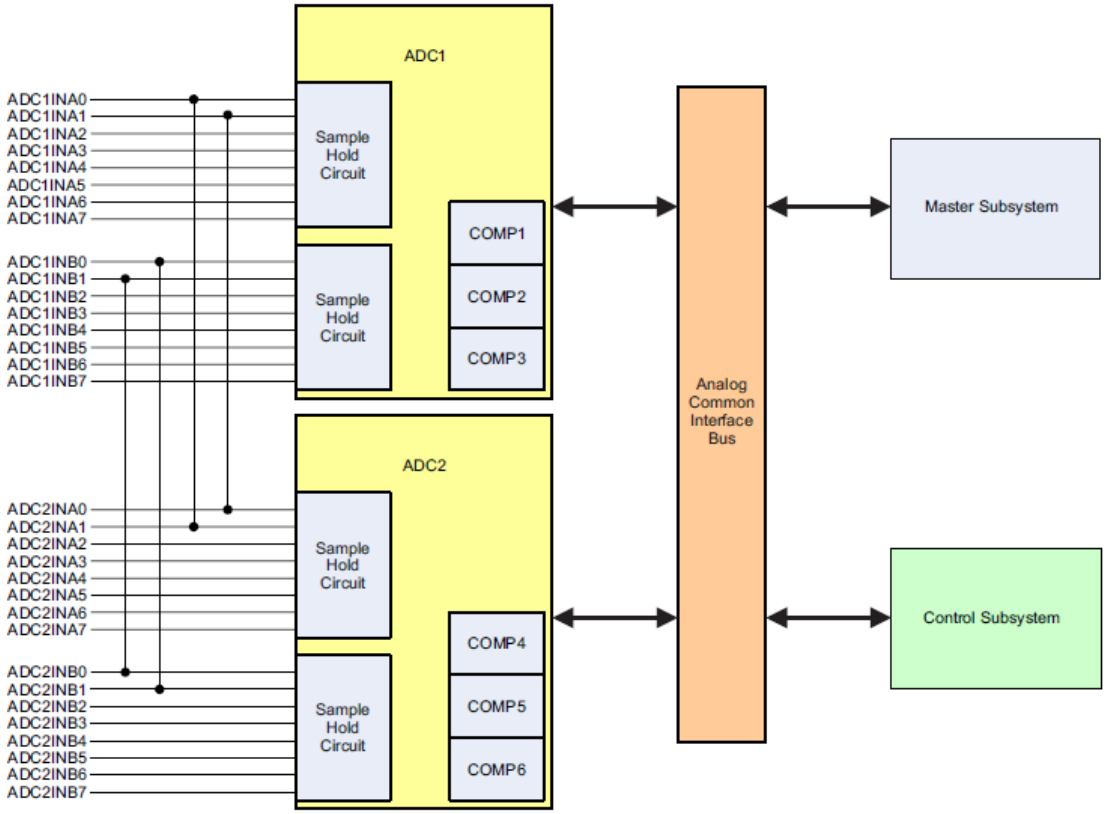


Ilustración 5: Diagrama de bloques del subsistema analógico.

Adicionalmente, el dispositivo cuenta con 64 pines GPIO que funcionan a 3’3V.

A continuación, se presenta una tabla con las características específicas del DSP y una imagen de las conexiones internas que encontramos entre los tres subsistemas que lo componen.

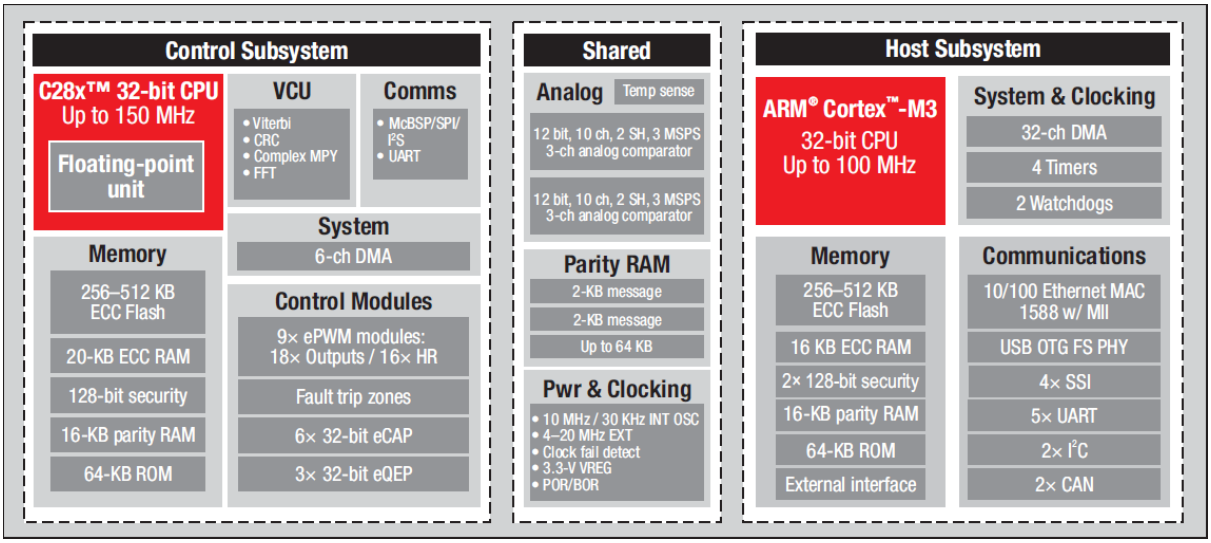


Tabla 1: Características especificas del DSP Concerto.

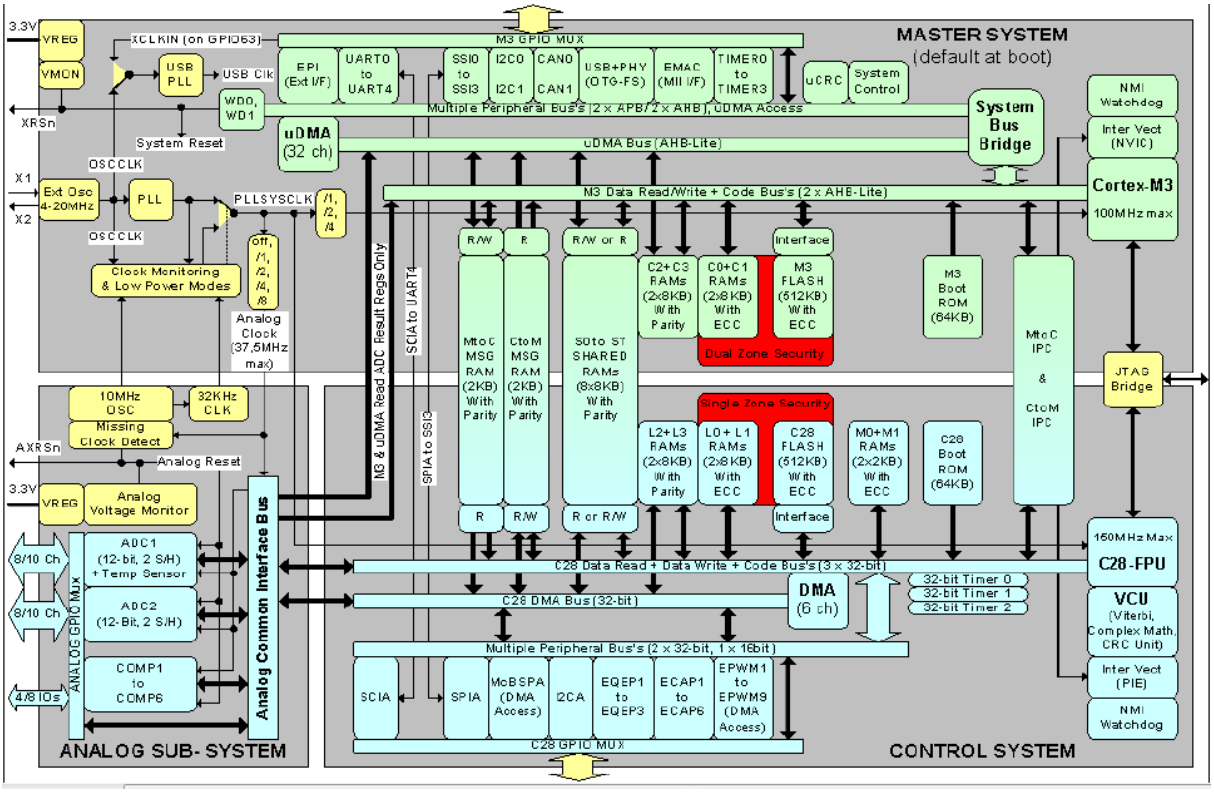


Ilustración 6: Conexión entre los distintos subsistemas.

Para trabajar con el dispositivo se utiliza el Experimenter's kit, que incluye una placa (controlCARD) con el DSP integrado, además de diferentes puertos de conexión y una dockstation, en la que se conecta esta placa y que permite un acceso sencillo a los pines de la placa.

Incluye un puerto JTAG que permite una fácil configuración del dispositivo con Code Composer Studio, cuya licencia se ofrece.

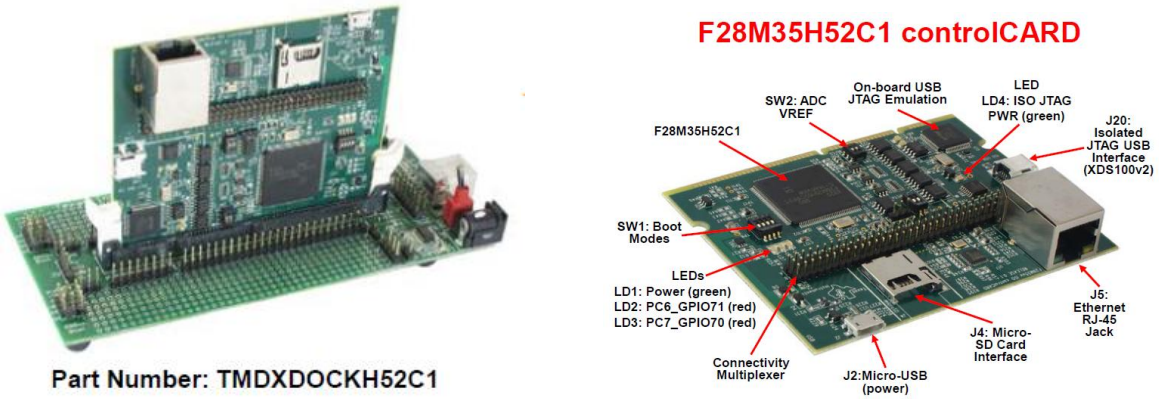


Ilustración 7: Dockstation y ControlCard.

No obstante, algunos de los pines GPIO del DSP no son directamente accesibles desde la dockstation, ya que la conexión de estos pines desde el MCU al DIMM de la controlCARD no es directa, por lo que habrá que usar jumpers como se verá más adelante.

2.1.2 Workshop

Antes de empezar a programar el sistema, es de utilidad conocer la configuración básica de los distintos periféricos que nos ofrece el dispositivo.

Con este fin, Texas Instruments ofrece una herramienta que incluye una serie de ejemplos básicos para el manejo del DSP: el Workshop.

Tras la descarga e instalación, se podrá encontrar una carpeta que contiene librerías para el dispositivo, documentos de información técnica tanto hardware como software, y un manual donde se explica paso a paso una serie de ejemplos que se comentaran a continuación.

Inicialmente se explica cómo utilizar CCS para crear un nuevo proyecto, generar la configuración de la tarjeta que permite decirle al programa como conectarse con el dispositivo y mostrar las herramientas del entorno.

Una vez que se tenga el código que se quiera correr, se conecta la tarjeta y se programa la flash del subsistema M3 pulsando el botón "debug". Una vez hecho esto, el subsistema C28 aparece desconectado, por lo que habrá que pulsar en él con el botón derecho del ratón y clicar en Conectar, conectandose así este sistema.

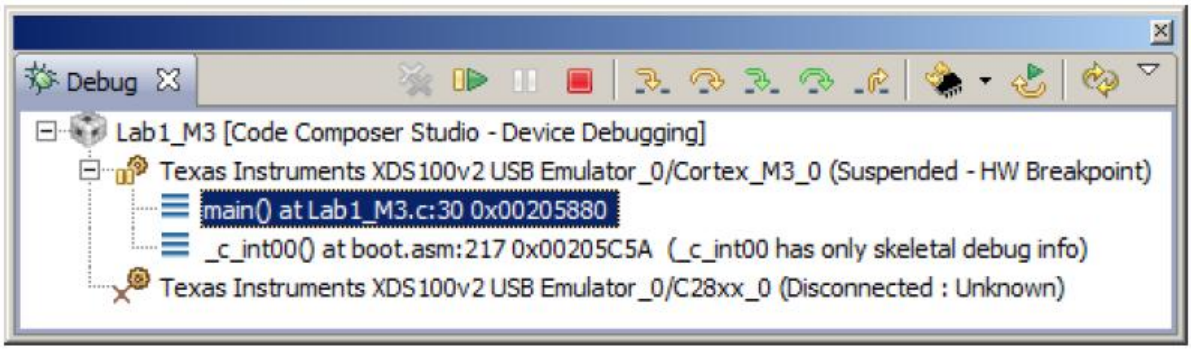


Ilustración 8: Conexión del subsistema C28.

Para cargar el código del subsistema C28, se pulsará el botón "load" con el subsistema C28 marcado y posteriormente "load programm" eligiendo el archivo .out que se quiera correr.

Después habrá que realizar la secuencia de arranque:

- Resetear C28
- Resetear M3
- Run C28
- Run M3

Para todos los ejercicios que se explican a continuación, se tendrá seleccionado el modo BOOT como arranque desde la Flash del M3. Para ello, el switch SW1 tendrá que tener todos sus interruptores hacia abajo.

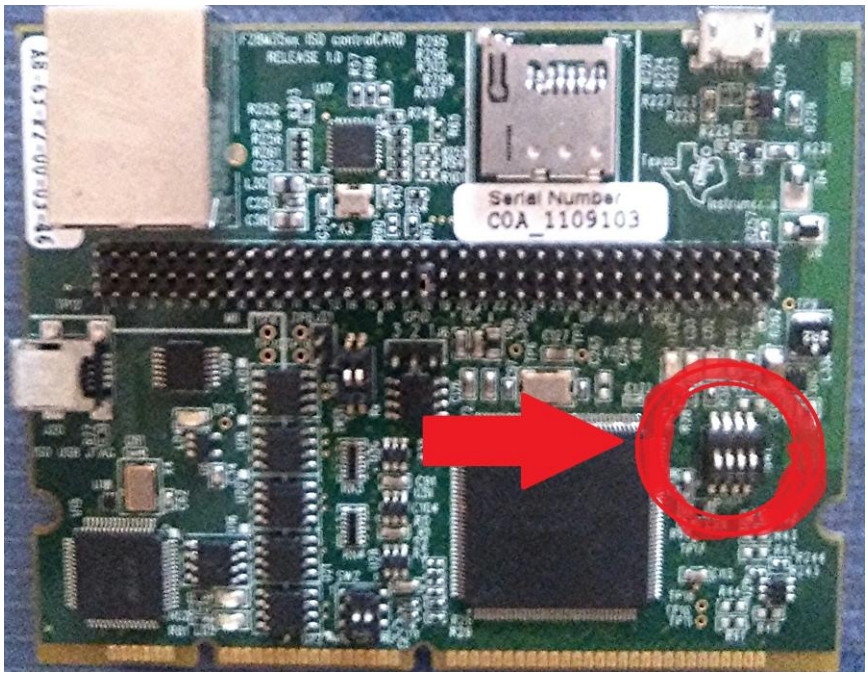


Ilustración 9: El switch SW1 permanece con todos sus interruptores hacia abajo

Ahora se verá cada uno de los ejemplos que contiene el Workshop.

- En el laboratorio 1 se expone un ejemplo sencillo en el que ambos núcleos trabajan de forma independiente, haciendo parpadear el subsistema M3 el led LD3 y el C28 el led LD2.

  Primero se crea un nuevo proyecto para el subsistema máster y se selecciona el dispositivo, en este caso, la familia ARM, eligiendo el dispositivo F28M35H52C1.

  Una vez hecho esto, habrá que añadir los archivos fuente del proyecto, haciendo clic derecho en la carpeta del proyecto que aparece en el explorador y pulsando "Add files".

  Para este ejemplo se añaden los siguientes archivos que se encuentran en la carpeta "device support":

  Startup_CCS.c, para las interrupciones, F28M35H52C1_m3.cmd, para localizar las variables dentro de la memoria física y driverlib.lib, para la inicialización del núcleo y los periféricos.

  Después se añaden las librerías necesarias, con clic derecho en la carpeta del proyecto y seleccionando propiedades. En este caso, deberá incluirse PROJECT_ROOT/../../../Device_support/MWARE.

  El M3 tendrá que inicializar el reloj del sistema, configurando el registro "SysCtlClockConfigSet", habilitar las interrupciones con la función "IntMasterEnable" y configurar el pin GPIO_ C7 (corresponde al GPIO 71 en el subsistema C28) como output, a través del registro "GPIOPinTypeGPIOOutput", para posteriormente encenderlo o apagarlo en un bucle infinito con "GPIOPinWrite"[1].

  Inicialmente, el subsistema maestro tiene el control de los pines GPIO, por lo que para usarlos en el C28 se tendrá que ceder el control de estos desde el M3, usando la función "IPCMtoCBootSystem" y dando el control del pin en cuestión mediante "GPIOPinConfigureCoreSelect".

[1] Todas las funciones usadas por el M3 se pueden encontrar en el archivo DRL_UG dentro de la carpeta Pdf del Workshop.

Se seguiran los mismos pasos para crear el proyecto del C28, esta vez seleccionando la familia C2000 y añadiendo los archivos F28M35x_CodeStartBranch.asm, F28M35x_SysCtrl.c, F28M35x_GlobalVariableDefs.c,F28M35x_Headers_nonBIOS.cmd, F28M35H52C1_c28.cmd y las librerías headers y common.

El main tendrá que inicializar el sistema del C28 con "InitSysCtrl" (se encuentra en el archivo SysCtrl), configurar el pin GPIO_70 como output, poniendo el registro "GpioG1CtrlRegs.GPCDIR.bit.GPIO70" a nivel alto y apagar y encender el led modificando el registro GPCDAT[2].

Realizando la secuencia de arranque explicada anteriormente, se puede observar como el led LD2 parpadea controlado por el C28 y el led LD3 por el M3.

- En el laboratorio 2 se pretende realizar la misma función que en el laboratorio 1 con el M3 pero esta vez utilizando una interrupción para hacer parpadear el led.

  Para ello se partirá de una copia del proyecto anterior y se modificará el main del mismo.

  Con respecto al código anterior, se añade la definición de la interrupción que se encargara de encender y apagar el led y que se activa con el timer0, el cual se configura para que cuente 75000000/2 ciclos de entrada de reloj de 13'33 nanosegundos y la interrupción salte cada 500 ms.

  Así, se habilitará el timer con la función "SysCtlPeripheralEnable (SYSCTL_PERIPH_TIMER0)" y con timerconfigure y timerloadset se configurará como timer periódico de 32 bit.

  Posteriormente, se habilita la interrupción y el timer con las funciones "IntEnable", "TimerIntEnable", "IntRegister" y "TimerEnable".

  Como consecuencia de esta configuración, el programa saltara cada 500ms apagando y encendiendo el led, el cual está controlado únicamente por el subsistema maestro.

- Al igual que en el laboratorio 2, en el laboratorio 3 se configura el subsistema de control para que realice mediante una interrupción el parpadeo del led LD2.

  Como antes, se hará una copia de la parte del C28 del laboratorio 1 y se añadirán las fuentes necesarias para que la modificación del código principal sea funcional.

  En este caso, se tendrá que añadir los archivos F28M35_CpuTimers.c, F28M35_DefaultIsr.c, F28M35_PieCtrl.c y F28M35_PieVect.c, necesarios para manejar la interrupción.

  Antes del main, se declara la función de la interrupción para el timer 0.

  Después, se inician los vectores de interrupción PIE con las funciones "InitPieCtrl" y InitPieVectTable" con funciones por defecto, y se asigna a la entrada del timer 0 la función de la rutina de interrupción (en este caso, la rutina tiene por nombre cpu_timer0_isr) que se creó antes con la siguiente línea:

  PieVectTable.TINT0=&cpu_timer0_isr

  Posteriormente, se inicializa el timer 0 con un periodo de 0.125 segundos y se habilita la interrupción, usando las funciones "InitCpuTimer" y "ConfigCpuTimers" y "PieCtrlRegs.PIEIER1.bit.INTx7=1", "IER|=1", "EINT" y "ERTM" respectivamente.

  Antes del bucle infinito, se arranca el temporizador 0 mediante "CpuTimer0Regs.TCR.bit.TSS=0".

  Finalmente, fuera del main, se añade la rutina de la interrupción.

  Para correr el código, se ha de seguir la secuencia de arranque, utilizando una copia del laboratorio 2 modificada para dar control al subsistema C28 desde el M3, consiguiendo así que el M3 tenga el led LD3 parpadeando y el C28 el led LD2, ambos usando una interrupción.

[2] Todas las funciones usadas por el C28 se pueden encontrar en el manual técnico del dispositivo

- En el laboratorio 4 se introduce el subsistema analógico y los módulos PWM.

  La onda generada por el PWM será una simétrica de 2KHz con un duty del 50%, la cual será capturada posteriormente por el convertidor ADC cuya configuración se verá en la segunda parte del laboratorio 4.

  Se partirá pues, de una copia del proyecto C28 del laboratorio 3, y se añadiran las fuentes necesarias para poder configurar el modulo PWM.

  Inicialmente, se habilita el GPIO0 para que funcione como ePWM1A, modificando el registro "GPAMUX1".

  Después, se configura el ePWM1A para que opere con una frecuencia de 2KHz usando el registro "TBPRD". Para ello, hay que tener en cuenta que:

$$\text{TBPRD}=\frac{fcpu}{2*fpwm*CLKDIV*HSPCLKDIV}$$

  Donde, fcpu es la frecuencia de la cpu, fpwm es la frecuencia del PWM (2 KHz), CLKDIV es parte de la pre escala del reloj de base (TBCLK) y se asigna previamente mediante "EPwm1Regs.TBCLT.bit.CLKDIV", y HSPCLKDIV es la otra parte de la pre escala del reloj base y se asigna previamente mediante "EPwm1Regs.TBCLT.bit.HSPCLKDIV".

  Además, para que la onda sea simétrica, habrá que configurar el registro CTRMODE en el modo up-down y se configuraran las acciones a realizar cuando la onda alcance al periodo, o al cero, con los registros "PRD" y "ZRO", ya sea forzando el PWM a nivel alto o bajo.

  Utilizando una copia del proyecto M3 del laboratorio 3, modificada para dar el control del GPIO0 al C28 como se vio en el laboratorio 1, y realizando la secuencia de arranque, se tendrá el código funcionando, de tal manera que, si conectamos un osciloscopio al GPIO0, se podrá observar la siguiente onda:

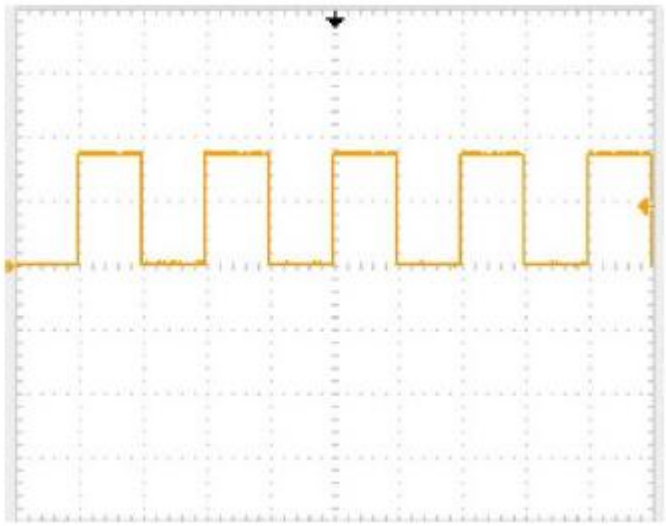


Ilustración 10: Onda PWM obtenida con osciloscopio.

Para la segunda parte de este ejercicio, se configurara el ADC para que capture la onda PWM generada en el apartado anterior, las medidas se guardaran en un buffer circular y se mostraran en una grafica creada con el CCS.

Como se ha ido haciendo, se parte de una copia de la primera parte del laboratorio y se añaden los archivos fuente necesarios para la configuración del ADC. En este caso, se añadirán los archivos F28M35x_Adc.c y F28M35x_usDelay.asm.

En el código principal, se cambia la función de la interrupción del temporizador por una nueva interrupción del ADC.

Se añaden 3 nuevas variables globales, "ConversionCount", "Voltage_C28[100]" y "GPIO70_count", las dos primeras para el buffer y la última para el parpadeo del led.

Habrá que cambiar la línea que asignaba la interrupción del timer a la entrada de la tabla de interrupciones, por la interrupción del ADC(a "PieVectTable.ADCINT1" se le asigna la interrupción del ADC) y se habilita la interrupción mediante el registro "PIEIER1".

Para que la frecuencia de muestreo del ADC sea de 50KHz, se cambia el periodo del timer0 a 20 microsegundos.

Antes de entrar en el bucle infinito, habrá que configurar el ADC de tal forma que se habilite el modo de no superposición, poniendo el bit "ADCNONOVERLAP" del registro "ADCCTL2" a nivel alto, la interrupción salte tras obtener un valor en el registro de resultados (ADCRESULTS) cambiando el bit "INTPULSEPOS" del registro "ADCCTL1" a uno, se habilite la interrupción, se deshabilite el modo continuo y se mande el pulso que habilita la interrupción con los bits "INT1E", "INT1CONT" e "INT1SEL" del registro "INTSEL1N2".

Dentro del registro "ADCSOC0CTL", se configura el bit "CHSEL" para asignar la entrada A0 del ADC a la SOC0, el bit "TRIGSEL" para que la conversión se dispare con el trigger1, al cual se le ha asignado previamente la interrupción 0 (AnalogSysCtrlRegs.TRIG1SEL) y el bit "ACQPS" para que cuente 7 ciclos de reloj.

Finalmente, se añadirá la rutina de interrupción del ADC fuera del main.

Una vez que se ha arrancado el código como se ha ido comentando, se conecta el PWM1A que se encuentra en el pin 0 con la entrada A0 del ADC (pin ADC-A0) y se deja correr el código unos segundos, para posteriormente parar el subsistema C28.

Si el usuario añade la variable "Voltage_C28" a la ventana de "Expressions" del CCS, podrá ver que esta varía entre 0 y 4095 correspondiendo a la variación de la entrada del ADC (entre 0 y 3,3 V).

Para generar la grafica dentro del CCS, habrá que hacer clic en Tools->Graph->Single Time y asignar los siguientes valores:

Acquisition Buffer Size	100
DSP Data Type	16-bit unsigned integer
Sampling Rate (Hz)	50000
Start Address	Voltage_C28
Display Data Size	100
Time Display Unit	µs

Tabla 2: Datos para la representación en CCS.

La grafica mostrara una señal periódica de 1 KHz y 50% de duty.

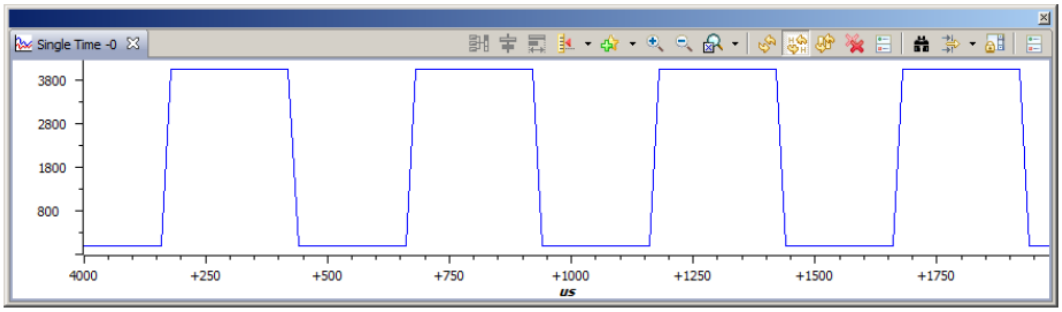


Ilustración 11: Grafica señal PWM obtenida con CCS.

- En el laboratorio 5, se quiere mostrar la comunicación entre los dos procesadores.

  Así, el C28 corre el mismo programa que en laboratorio 4, solo que esta vez los resultados se escribirán en la RAM compartida y el subsistema maestro leerá esos datos y los guardara en un buffer circular. Finalmente, se mostraran la grafica resultante con el CCS.

  Para ello, al inicio del código se añadirá un directivo "pragma" que emplaza la variable "LatestAdcResult" en la sección de datos "CTOM".

  Dentro de la rutina de interrupción del ADC, se carga el valor obtenido de la conversión en la CTOM y se habilita la bandera "IPC1" para el subsistema M3 indicando la existencia de un dato.

  Para este laboratorio, se tendrá que crear un archivo linkador, pinchando en File->New->Source file e introduciendo el siguiente código:

```
SECTIONS
{
        CtoM: > CTOMRAM, PAGE = 1
}
```

Ilustración 12: Código linkador del C28

Para el subsistema M3 se parte de una copia del laboratorio 4 y se añade una nueva librería (“inc/hw_ipc.h”), además de 3 nuevas variables globales para manejar los datos recibidos.

Dentro del bucle while, se esperará la activación de la bandera, leyendo en ese caso el dato y cargándolo desde la CtoM a la RAM.

Al igual que antes, habrá que crear un archivo linkador con el siguiente código:

```
SECTIONS
{
        CtoM: > CTOMRAM, TYPE = DSECT
}
```

Ilustración 13: Código linkador M3

Por último, se lanza el programa en ambos subsistemas y se dibujan las graficas con el CCS.

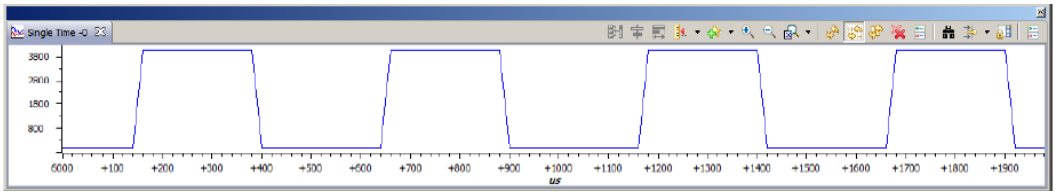


Ilustración 14: Grafica PWM C28

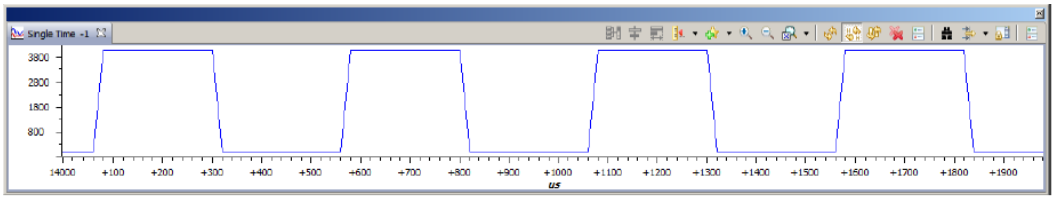


Ilustración 15: Grafica PWM vista desde el M3.

2.2 Equipos auxiliares

Además de las herramientas explicadas en los apartados anteriores, fueron necesarios distintos equipos de laboratorio, tanto para el contacto inicial con el sistema, como para conseguir el objetivo final del proyecto.

Uno de los equipos que se requirieron fue el osciloscopio, el cual es una de las herramientas más útiles a la hora de comprobar que el código que se esta ejecutando está realizando de verdad lo que se quiere, sobre todo para observar que la onda que se ha pretendido configurar mediante una serie de registros se corresponde con la que proporciona el sistema en realidad.

Ilustración 16: Osciloscopio.

A la hora de utilizar este equipo, es importante ajustar la sonda y el osciloscopio con la misma atenuación, y ajustar la escala temporal para poder visualizar correctamente la onda, teniendo en cuenta que la frecuencia de las ondas que se van a generar rondaran los 10KHz.

Como se verá más adelante, a la hora de comprobar las comunicaciones, será necesario un cable USB a puerto RS232, una placa puente[3], que conectara el puerto con los pines de transmisión y recepción del MCU, y una fuente de alimentación.

El convertidor USB a RS232 utilizado es de la empresa Prolific Technology Inc. y resultó de utilidad para la conexión del ordenador portátil con el DSP a través de la placa puente que se mostrara más adelante, permitiendo el envío de los caracteres hacia el MCU y la recepción desde el mismo y pudiendo visualizarlos mediante el uso adicional de un par de programas informáticos que se comentaran en el siguiente apartado.

Ilustración 17: Convertidor USB a Puerto serie.

[3] Esta placa ha sido diseñada y fabricada por miembros del departamento de ingeniería electrónica de la escuela.

La placa mencionada anteriormente, permite la conexión del puerto serie por un lado y la de los pines de recepción (Rx) y transmisión (Tx) del MCU por otro, de manera que ambos sistemas, ordenador y MCU, se conectan físicamente, permitiendo la comunicación a través de la UART del M3 o del SCI del C28.

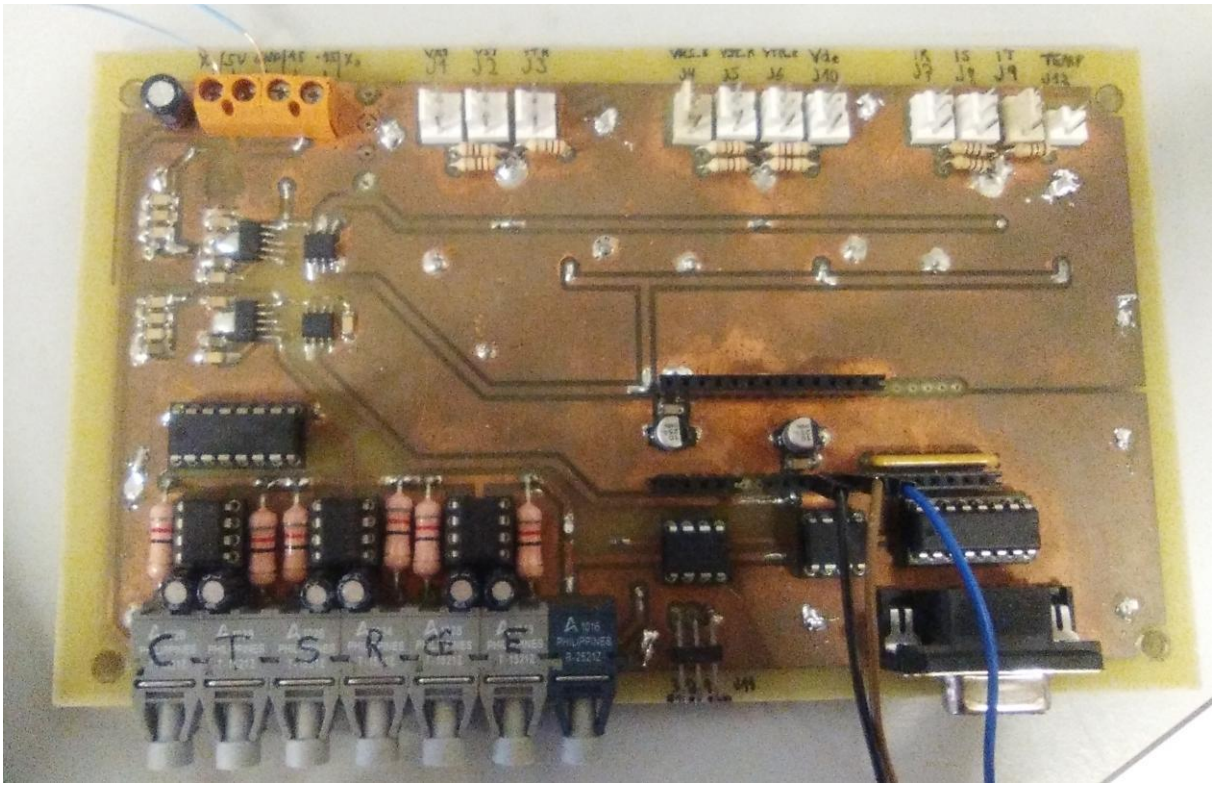


Ilustración 18: Placa de adaptación.

Para que la placa pueda funcionar, necesita una alimentación de 5V y para ello se utilizó una fuente de alimentación regulable de laboratorio.

Ilustración 19: Fuente de alimentación variable.

Por último, para realizar las pruebas finales, antes de montar el dispositivo en el sistema de potencia, se utilizó un generador de ondas, que permitió la simulación de las corrientes y tensiones que llegan al sistema final.

Ilustración 20 : Generador de ondas.

2.3 Programas utilizados

A la hora de visualizar los datos obtenidos, se utilizaron dos programas, uno de adquisición de datos, Labview, y un terminal virtual, TeraTerm.

Labview es un software de desarrollo de sistemas para aplicaciones científicas e industriales, creado por la empresa National Instruments, que permite la adquisición, control, análisis y presentación de datos de forma fácil.

Tiene una interfaz sencilla, que permite una programación muy visual, por lo que no es excesivamente complicado de usar.

Para crear un archivo nuevo, inicialmente se abren dos ventanas: una para los diagrama de bloques y otra es el panel frontal.

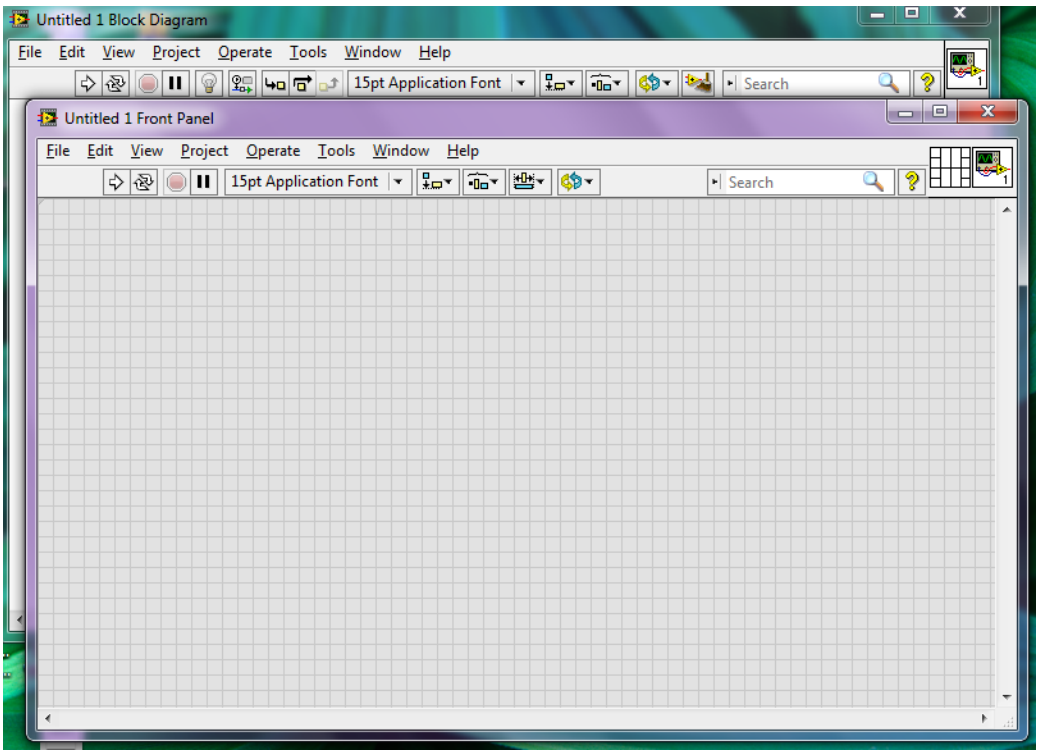


Ilustración 21: Visualización Labview.

El panel frontal es la interfaz grafica con el usuario. Está formado por una serie de botones, controles e indicadores que el usuario define y recoge las entradas introducidas por el usuario, mostrando las salidas correspondientes generadas por el programa.

Por otra parte, la pantalla de diagrama de bloques es aquella en la cual se realiza la implementación del programa para controlar y procesar las entradas y salidas creadas en el panel frontal.

Ya que la programación es visual, esta consistirá en la unión de diferentes bloques de funciones proporcionados por las librerías de Labview y ajustarlos con los parámetros que se requieran.

En este proyecto, se utiliza Labview para la obtención y visualización de las medidas que proporciona el sistema real a través del DSP, además de la comprobación del funcionamiento adecuado del mismo.

Para ello, como se verá en apartados próximos, se parte de un código en Labview específico para el convertidor de potencia que se esta tratando, por lo que a priori, no se tendrá que programar en Labview.

A la hora de realizar la configuración de la UART o el SCI del DSP para las comunicaciones entre este y el ordenador, se utilizó el programa TeraTem, una maquina virtual de uso más simple y con menos funcionalidad que Labview, con objeto de simplificar la comprobación de las configuraciones realizadas en la realización de ejemplos de comunicación sencillos para aprender a manejar los distintos registros dedicados a la transmisión/recepción de datos.

A diferencia de Labview, TeraTerm no permite la programación de un código de forma visual, simplemente muestra lo que se recibe del MCU y permite el envío desde el ordenador, lo que es suficiente para trabajar con los ejemplos más sencillos de comunicación.

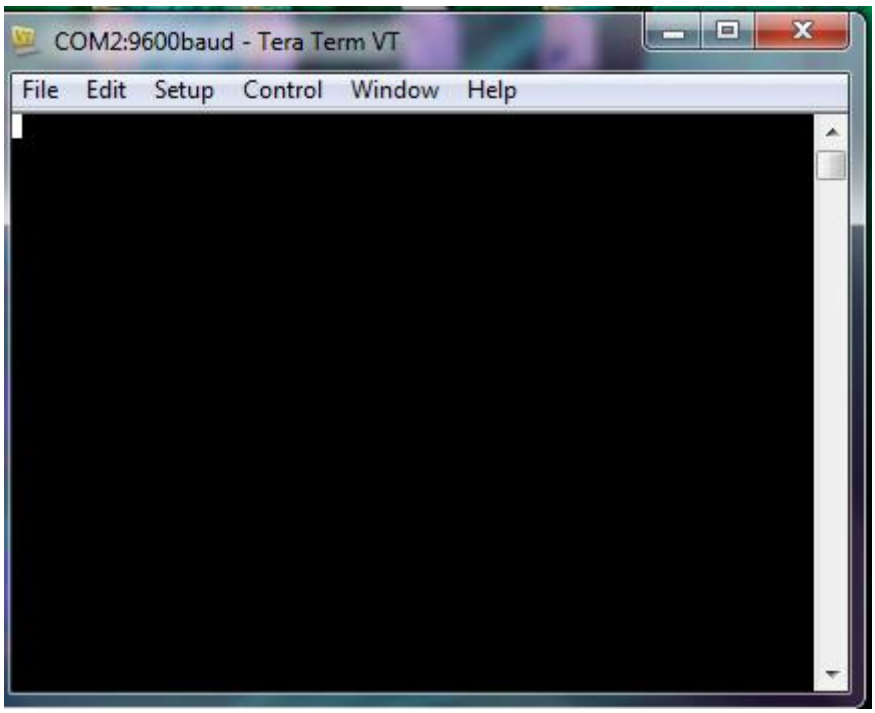


Ilustración 22: Visualización TeraTerm.

El uso del programa es bastante simple.

Dentro de la pestaña setup, en la opción Serial port, se pueden configurar las características del puerto al que el usuario se quiere conectar (numero del puerto, velocidad de transmisión, numero de datos recibidos, paridad…).

Así mismo, en la opción terminal, el usuario puede configurar la visualización del terminal, además de permitir el envío de caracteres mediante la pestaña local echo (eco local).

El uso de ambos programas se verá reflejado en el apartado de comunicaciones, donde se verá los distintos ejemplos realizados y la configuración final de las mismas para el sistema.

3 CONFIGURACIÓN DEL DISPOSITIVO

Hasta ahora se ha visto los pasos previos para aprender a manejar el dispositivo, así como, las herramientas y equipos que se han utilizado o van a utilizar para la configuración del mismo.

El objetivo principal de este proyecto consiste en la adaptación del código utilizado para controlar y gestionar el dispositivo de potencia que se quiere controlar con el DSP anterior (Delfino) para el nuevo DSP (Concerto).

No obstante, en los primeros apartados de este capítulo, se partirá de uno de los códigos de los laboratorios que se explicaron en el capitulo anterior, y posteriormente en el apartado de comunicaciones, se introducirán los archivos fuentes, librerías y código pertenecientes al anterior proyecto y que siguen siendo de utilidad para este.

En este capítulo, se pretende explicar la configuración del DSP realizada para que el sistema real funcione correctamente.

3.1 Módulo PWM

El PWM es una herramienta esencial para el control de mucho de los dispositivos de electrónica de potencia que se encuentran en equipos industriales y comerciales.

El ePWM utilizado en este dispositivo desempeña la función de un DAC, donde el duty cycle es equivalente al valor analógico que proporciona el DAC.

El dispositivo cuenta con 9 módulos PWM, los cuales representan un canal completo PWM compuesto de dos salidas: EPWMxA y EPWMxB.

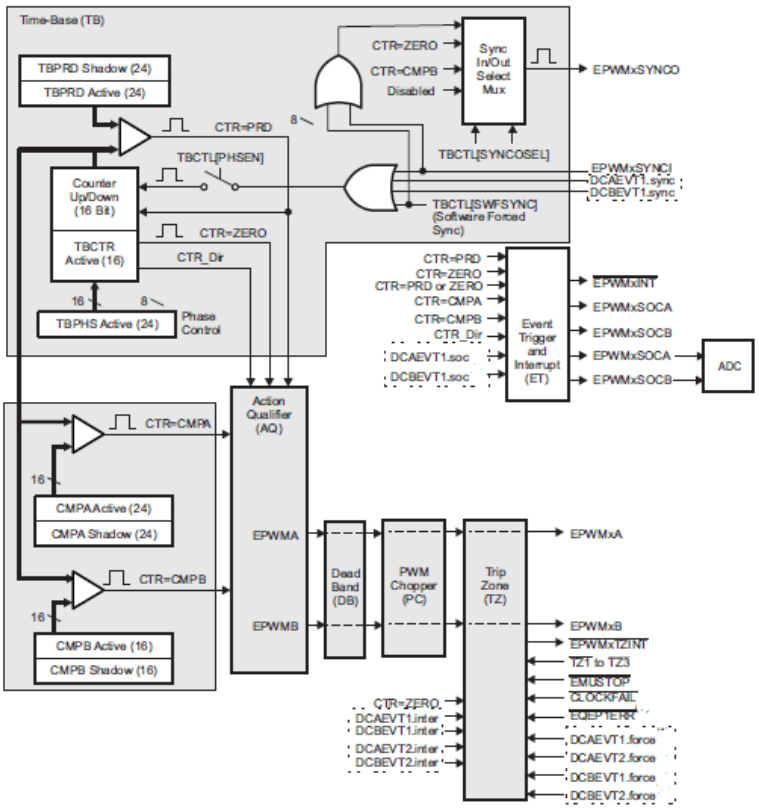


Ilustración 23: Modulo ePWM.

Los módulos van sincronizados mediante la señal de reloj del sistema (SYSCLKOUT) escalada y están compuestos cada uno por un contador de 16 bit con control de periodo y frecuencia. Son capaces de lanzar consignas de comparación y permiten que se fuercen los estados lógicos de sus salidas (alto, bajo, alta impedancia) y lanzar interrupciones de la CPU y/o ADC SOC's.

Para el control de los disparos de los IGBT's y al ser el sistema trifásico, se tendrán 3 módulos PWM, uno por cada fase (R, S, T) y un modulo PWM adicional para el chopper.

Las ondas de estos módulos serán simétricas (Up-Down) con una frecuencia en principio de 5-10 KHz, las cuales generaran una onda cuadrada con un duty determinado según la consigna de comparación que reciba cada modulo. Esta consigna es la misma para los tres primeros módulos pero desfasada entre ellos en 120 grados. Si la onda generada está por encima de dicha consigna, la onda cuadrada del PWM permanece a nivel bajo. Por el contrario, si dicha onda se encuentra por debajo de la consigna, la onda cuadrada cambiara a nivel alto.

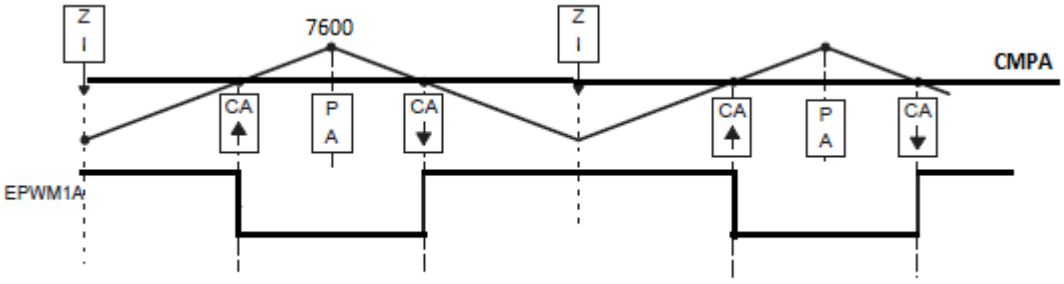


Ilustración 24: Onda PWM.

Una vez que se sabe cómo ha de ser la onda, se pasará a la configuración de los módulos.

Inicialmente, habrá que configurar los pines GPIO para que funcionen como PWM. En este caso, el pin GPIO0 corresponderá a la salida EPWM1A, el GPIO2 al EPWM2A, el GPIO4 al EPWM3A y el GPIO6[4] al EPWM4A, correspondiendo respectivamente a las salidas R, S, T y CH del dispositivo de potencia.

Para ello, dentro del main del subsistema M3 se dará control al c28 de los pines 0, 2, 4 y 6, primero habilitando el puerto GPIOA y posteriormente, dando el control de los sucesivos pines:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
GPIOPinConfigureCoreSelect(GPIO_PORTA_BASE, GPIO_PIN_0, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTA_BASE, GPIO_PIN_2, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTA_BASE, GPIO_PIN_4, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTA_BASE, GPIO_PIN_6, GPIO_PIN_C_CORE_SELECT);
```

[4] El GPIO6 no está directamente conectado al DIMM de la placa (controlCARD), por lo que para tener acceso a él a través del dockstation tendremos que conectar un jumper entre los pines J13 pin 3(fila B) y J17 pin 3(fila A) de la tarjeta.

En el código del C28, se configuraran estos pines como salidas PWM:

GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1; // GPIO0 = ePWM1A R

GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1; // GPIO2 = ePWM2A S

GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1; // GPIO4 = ePWM3A T

GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1; // GPIO6 = ePWM4A CH

Después se configura cada modulo PWM:

- Modulo 1:

  En el modulo 1 se configura el registro fase para que la onda comience en 0 y no se carga el registro del contador con el registro de fase, de tal forma que a partir de la onda del modulo uno se generaran las otras tres.

  EPwm1Regs.TBPHS.half.TBPHS = 0;

  EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;

  Posteriormente se pone a cero el contador y se activa el registro sombra.

  El registro sombra guarda el nuevo valor de la consigna de comparación y lo carga una vez que la onda PWM ha terminado el ciclo.

  EPwm1Regs.TBCTR = 0x0000;// Clear counter

  EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;

  Se sincroniza la señal de salida cuando el contador este a cero y se ajusta la onda PWM para que cuando alcance la consigna de comparación se ponga a nivel bajo y viceversa.

  EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; //tb_ctr=0

  EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR; //cuando alcanza CMPA se pone a BAJO(incrementado)

  EPwm1Regs.AQCTLA.bit.CAD = AQ_SET;//cuando alcanza CMPA se pone a ALTO el pwm(decrementando)

  La onda simétrica (up-down) estará configurada con una frecuencia de 10KHz mediante el registro TBPRD, ajustando las escalas de tiempo (CLKDIV y HSPCLKDIV) a 1, y se configurará el registro sombra para que guarde el valor de la consigna de comparación (CMPA) y lo cargue en cero y al finalizar el periodo del PWM, para que se produzca la conversión del ADC en esos dos instantes.

  EPwm1Regs.TBCTL.bit.CLKDIV = 0; // CLKDIV = "div by 1";

  EPwm1Regs.TBCTL.bit.HSPCLKDIV = 0; // HSPCLKDIV = "div by 1"

  EPwm1Regs.TBCTL.bit.CTRMODE = 2; // count mode = up-down mode

  EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;

  EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO_PRD; // load on CTR=Zero y CTR=Prd

  EPwm1Regs.TBPRD = PERIOD;

El módulo uno es el que se encarga de habilitar la interrupción SOC, que lanza la conversión en cero y cuando se alcanza el periodo del PWM, para ello se modificarán los registros ETSEL y ETPS:

EPwm1Regs.ETSEL.bit.SOCAEN = 1;//enable Soc en A

EPwm1Regs.ETSEL.bit.SOCASEL = 0x3; // se genera el pulso en cero y prd.

EPwm1Regs.ETSEL.bit.SOCASELCMP = 0x0;//CMPA

EPwm1Regs.ETPS.bit.SOCAPRD = 1; //generar pulso en el primer evento

- Los otros 3 módulos se configuran de igual forma, cambiando su registro respectivo, Epwm2 para el 2, Epwm3 para el 3 y Epwm4 para el 4.

Por último, se asignan las consignas de comparación obtenidas a cada módulo dentro de la rutina del ADC:

EPwm1Regs.CMPA.half.CMPA = ref.pwm.r;

EPwm2Regs.CMPA.half.CMPA = ref.pwm.s;

EPwm3Regs.CMPA.half.CMPA = ref.pwm.t;

EPwm4Regs.CMPA.half.CMPA = ref.pwm.chp;

Una vez hecho esto, conectando un osciloscopio a los pines 0, 2, 4 y 6 el usuario podrá observar las ondas PWM generadas que ha configurado.

3.2 ADC

El convertidor analógico-digital (ADC) permite la conversión de señales analógicas, en este caso las tensiones e intensidades que se generan en el dispositivo de potencia, en señales digitales, para que puedan ser tratadas fácilmente por el DSP.

El modulo ADC del dispositivo consta de 16 canales[5] con una resolución de 12 bit, y su estructura es parte registro de aproximaciones sucesivas (SAR) parte pipeline.

Está compuesto por una parte analógica que contiene circuitos multiplexadores(MUX), circuitos de sample and hold(S/H), circuitos reguladores de voltaje, un circuito de conversión y otros circuitos de apoyo, y una parte digital formada por conversores programables, registros de resultados y el bus de comunicaciones ACIB.

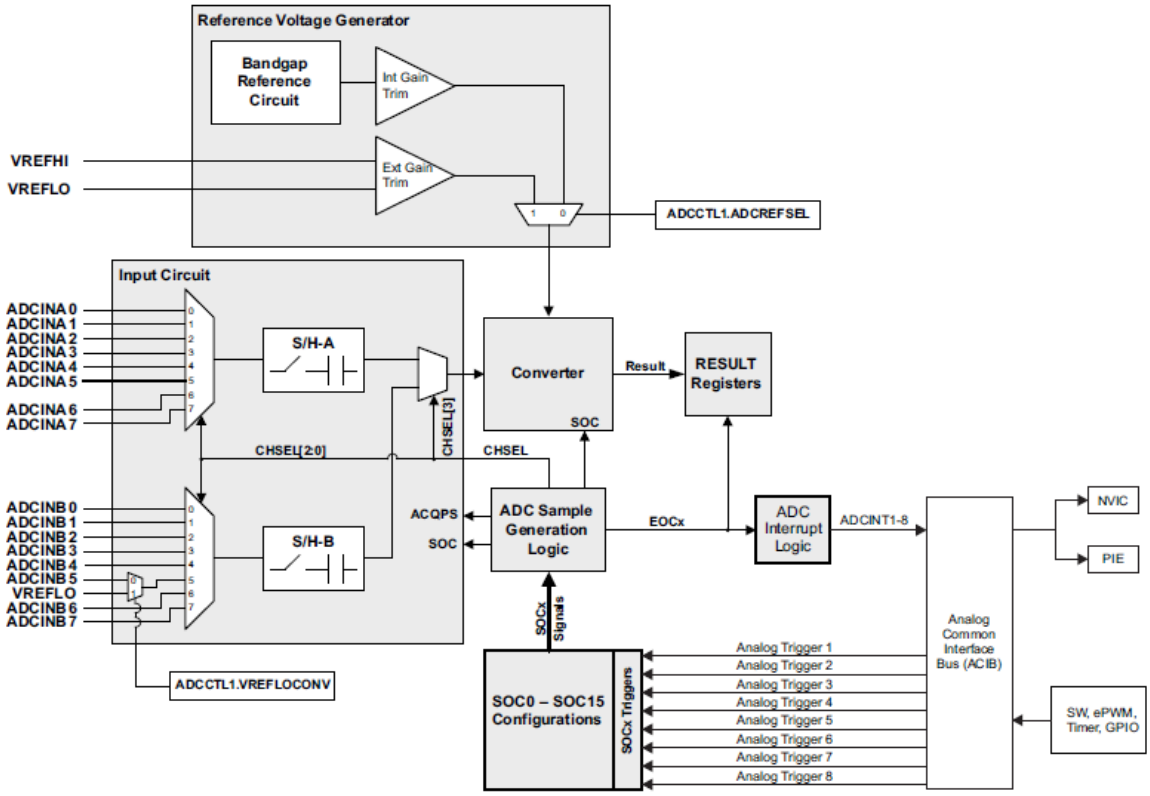


Ilustración 25: Modulo ADC.

Para el sistema, se utilizaran los canales del 1 al 7 del puerto A del ADC1[6] para que la conversión de todos los canales comience con la señal de trigger que genera el modulo PWM1 configurado en el apartado anterior.

Para ello, inicialmente se creará una interrupción del ADC y se configurará el reloj del sistema tal y como se vio en el laboratorio 4 del apartado 2.1.2 de este documento.

Esta interrupción actualizará los valores de la consigna de comparación de los PWM, ajustando así los disparos de los IGBT´s.

Posteriormente, se configuraran los registros del ADC para la conversión de los canales.

[5] De estos canales solo 10 son accesibles en este dispositivo.
[6] Con respecto a la dockstation.

Así, se modificará el registro ADC1 para que las conversiones no se solapen, haciendo que la interrupción del ADC salte una vez que se haya obtenido un resultado de conversión y que la señal que active esta interrupción sea EOC0.

```
Adc1Regs.ADCCTL2.bit.ADCNONOVERLAP = 1; // Enable non-overlap mode
Adc1Regs.ADCCTL1.bit.INTPULSEPOS = 1; // ADCINT1 trips after AdcResults latch
Adc1Regs.INTSEL1N2.bit.INT1SEL = 0; // setup EOC0 to trigger ADCINT1 to fire
```

Después, el usuario elegirá que SOC se encarga de convertir cada canal. Así, el SOC0 se encargara de la conversión del canal de entrada A1, el SOC1 del A2 y así sucesivamente[7].

```
Adc1Regs.ADCSOC0CTL.bit.CHSEL = 8; //B0/A1/It
Adc1Regs.ADCSOC1CTL.bit.CHSEL = 2;//A2/A2/ Is
Adc1Regs.ADCSOC2CTL.bit.CHSEL = 3;//A3/A3/Ir
Adc1Regs.ADCSOC3CTL.bit.CHSEL = 4;//A4/A4/Vdc
Adc1Regs.ADCSOC4CTL.bit.CHSEL = 0xC;//B4/A5/Vt
Adc1Regs.ADCSOC5CTL.bit.CHSEL = 6;//A6/A6/Vs
Adc1Regs.ADCSOC6CTL.bit.CHSEL = 7;//A7/A7/Vr
```

Para indicar que la señal que lanza la conversión sea el PWM1 con el SOCA, se modifica el registro TRIG1SEL para que se sincronice con la señal PWM:

```
AnalogSysctrlRegs.TRIG1SEL.all = 5;
```

Y posteriormente, se hace que esta señal sea la que active cada uno de los canales SOC antes seleccionados.

```
Adc1Regs.ADCSOC0CTL.bit.TRIGSEL = 5; // Set SOC0 start trigger to
                                     // ADC Trigger 1 of the ADC
Adc1Regs.ADCSOC1CTL.bit.TRIGSEL = 5; // Set SOC1 start trigger to
                                     // ADC Trigger 1 of the ADC
Adc1Regs.ADCSOC2CTL.bit.TRIGSEL = 5; // Set SOC2 start trigger to
                                     // ADC Trigger 1 of the ADC
Adc1Regs.ADCSOC3CTL.bit.TRIGSEL = 5; // Set SOC3 start trigger to
                                     // ADC Trigger 1 of the ADC
Adc1Regs.ADCSOC4CTL.bit.TRIGSEL = 5; // Set SOC4 start trigger to
                                     // ADC Trigger 1 of the ADC
Adc1Regs.ADCSOC5CTL.bit.TRIGSEL = 5; // Set SOC5 start trigger to
                                     // ADC Trigger 1 of the ADC
Adc1Regs.ADCSOC6CTL.bit.TRIGSEL = 5; // Set SOC6 start trigger to
                                     // ADC Trigger 1 of the ADC
```

[7] El valor del registro dependerá de la conexión interna entre la dockstation y la controlCARD. Por ejemplo, el pin A1 de la dockstation corresponde al canal ADC1_B0 de la controlCARD.

Por último, se indica durante cuantos ciclos se quiere que permanezca el dato de conversión mediante el registro ACQPS. El valor escrito en este registro es uno menos del número de ciclos deseados.

```
Adc1Regs.ADCSOC0CTL.bit.ACQPS = 6; // set SOC0 S/H Window to 7 ADC
                                   // Clock Cycles, (6 ACQPS + 1)
Adc1Regs.ADCSOC1CTL.bit.ACQPS = 6; // set SOC1 S/H Window to 7 ADC
                                   // Clock Cycles, (6 ACQPS + 1)
Adc1Regs.ADCSOC2CTL.bit.ACQPS = 6; // set SOC2 S/H Window to 7 ADC
                                   // Clock Cycles, (6 ACQPS + 1)
Adc1Regs.ADCSOC3CTL.bit.ACQPS = 6; // set SOC3 S/H Window to 7 ADC
                                   // Clock Cycles, (6 ACQPS + 1)
Adc1Regs.ADCSOC4CTL.bit.ACQPS = 6; // set SOC4 S/H Window to 7 ADC
                                   // Clock Cycles, (6 ACQPS + 1)
Adc1Regs.ADCSOC5CTL.bit.ACQPS = 6; // set SOC5 S/H Window to 7 ADC
                                   // Clock Cycles, (6 ACQPS + 1)
Adc1Regs.ADCSOC6CTL.bit.ACQPS = 6; // set SOC6 S/H Window to 7 ADC
                                   // Clock Cycles, (6 ACQPS + 1)
```

Una vez se ha cambiado los registros, se inicia el adc1 con la función InitAdc1, que se encuentra definida en las librerias del dispositivo.

3.3 Comunicaciones

Cuando se esta controlando un sistema, se necesita una interfaz que permita visualizar lo que está ocurriendo y actuar en consecuencia. Para ello, es precisa la comunicación entre el DSP y la interfaz del usuario, que en este caso será Labview.

El MCU que se esta utilizando tiene muchos puertos de comunicación (I2C, CAN, Ethernet, McBSP…) pero este capítulo, y este proyecto se centrará en dos de ellos: UART y SCI.

3.3.1 UART

La UART (Universal asynchronous receiver-transmitter) es un periférico que controla los puertos y dispositivos serie, y se encarga de convertir los datos en formato paralelo, que reciben los buses del sistema, a datos en formato serie, para que puedan ser transmitidos por los puertos y viceversa.

Normalmente se usa en conjunción con otros estándares de comunicación, como son RS-232 que es el que se utilizará más adelante.

Permite el envío y recepción de datos en distintos formatos y velocidades, de ahí su carácter universal.

La UART del dispositivo en cuestión se encuentra en el subsistema M3 y permite velocidades entre la velocidad del reloj del M3/16, para velocidades normales, y la velocidad del reloj del M3/ 8, para altas velocidades.

Consta de dos FIFO de 16 * 8 independiente con longitud programable, una de transmisión (Tx) y otra de recepción (Rx)) para agilizar la carga de la CPU.

Incluye los bits estándar de comunicación asíncrona para el comienzo, parada y la paridad y detección de fallos de comunicación.

Se consigue una transferencia eficiente mediante la DMA, la cual permite que la transmisión y recepción sea independientes una de otra mediante el uso de canales separados, y que las peticiones de recepción y transmisión se acepten únicamente cuando haya datos en la FIFO o haya hueco respectivamente.

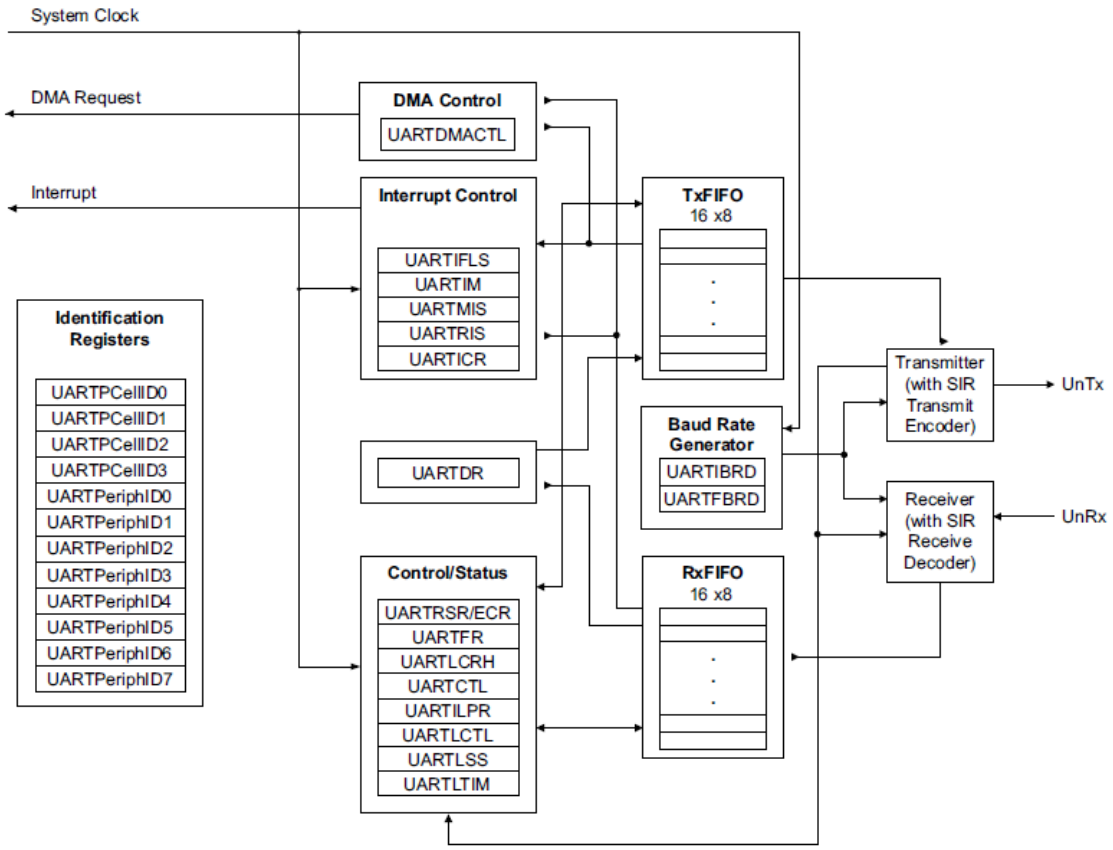


Ilustración 26: Bloque de diagrama de la UART.

La transmisión se realiza mediante la conversión del dato que se encuentra en la FIFO de transmisión de paralelo a serie. Esta empieza con un bit de comienzo y posteriormente se envían los datos desde el bit menos significativo (LSB) al más significativo (MSB), después el bit de paridad y por último los bits de parada.

La recepción por el contrario, convierte el dato de serie a paralelo, después de que se haya producido un pulso de comienzo valido.

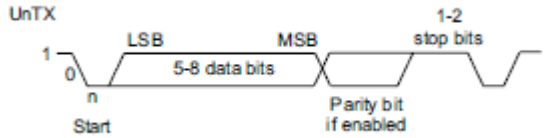


Ilustración 27: Secuencia de transmisión UART.

La velocidad de transmisión/recepción de los datos se configura mediante el Baud Rate, que indica el número de bits transmitidos/recibidos por segundo.

Para configurar la UART se parte de un ejemplo de código que proporciona el ControlSuite de Texas Instruments, y se utiliza el terminal TeraTerm, comentado en el apartado 2.3 de este documento, para visualizar los datos recibidos desde el DSP y enviar datos al mismo.

El ejemplo consiste en el envío de una cadena de caracteres desde el DSP a través de la UART que muestra el mensaje "Enter text:" y la espera a la recepción de un carácter, para volver a transmitirlo mediante interrupción.

Para ello, inicialmente se incluye la librería que contiene las funciones usadas por la UART, "driverlib/uart.h", que incluye las funciones "UARTCharGetNonBlocking" y "UARTCharPutNonBlocking" que permiten leer y escribir un dato respectivamente.

Después se habilitan los periféricos que va a usar el usuario, en este caso la UART0 y el puerto E de la GPIO donde se encuentran los pines GPIO28 (pin de recepción rx) y GPIO29 (pin de transmisión tx).

SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

Y se configuran dichos pines para que funcionen como Rx y Tx:

GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);

GPIOPinConfigure(GPIO_PE4_U0RX);

GPIOPinConfigure(GPIO_PE5_U0TX);

Posteriormente, se configura la UART para que transmita a 9600, con un bit de stop, sin bit de paridad y 8 bits de datos:

UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(SYSTEM_CLOCK_SPEED), 9600,

(UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |

UART_CONFIG_PAR_NONE));

Y se habilitan las interrupciones y las FIFO de transmisión y recepción indicando a que interrupción atienden:

UARTFIFOEnable(UART0_BASE);

UARTFIFOLevelSet(UART0_BASE, UART_FIFO_TX1_8,UART_FIFO_RX1_8);

UARTIntEnable(UART0_BASE, UART_INT_RX);

UARTIntRegister(UART0_BASE, UARTIntHandler);

IntMasterEnable();

Una vez configurada la UART y antes de entrar en el bucle while se envia la cadena de caracteres con la función UARTSend, la cual se define en el código, y recibe una cadena de caracteres y su tamaño y envía carácter a carácter a la UART mediante la función "UARTCharPutNonBlocking" .

Por último, en la rutina de interrupción (UARTIntHandler), se espera a que haya un dato disponible en la FIFO de recepción mediante la función "UARTCharsAvail", se lee el dato con la funcion "UARTCharPutNonBlocking" y se vuelve a transmitir con "UARTCharPutNonBlocking".

Esta interrupción saltara cada vez que se envie un dato por el TeraTerm.

Una vez entendido el código, se comunicará el TeraTerm con el DSP a través del adaptador de USB a RS-232 y la placa de adaptación comentada en el apartado 2.2 y se configurará el terminal para que atienda al puerto en el que se tenga conectado el adaptador a una velocidad de 9600 sin paridad, con un bit de stop.

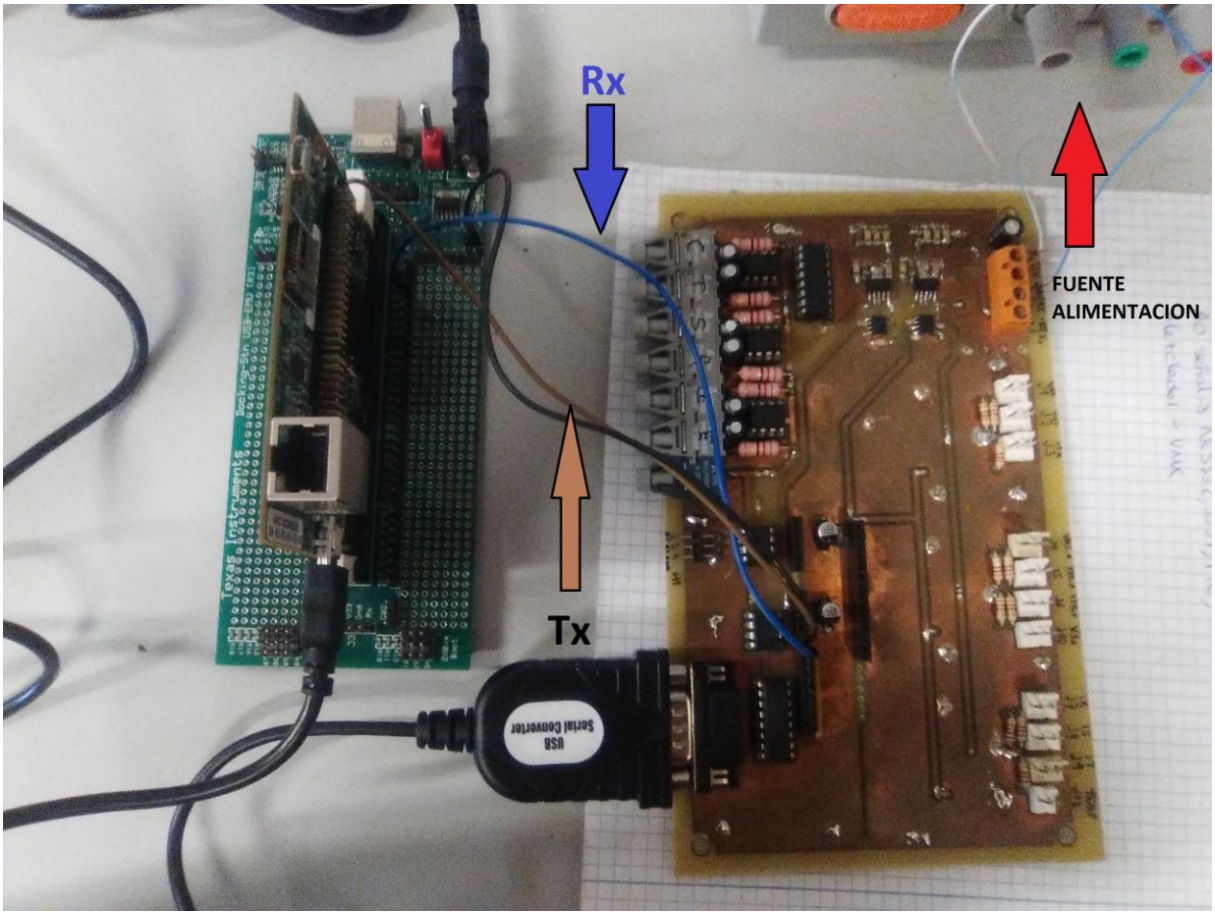
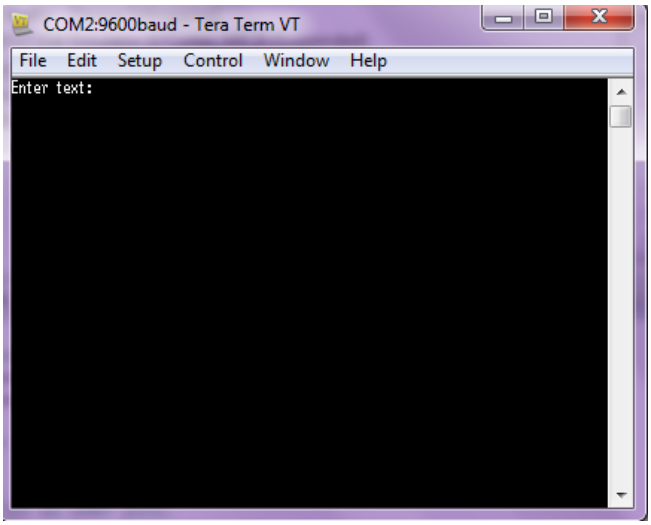


Ilustración 28: Conexión de equipos.

Una vez se tiene todo conectado, se carga el código en el DSP siguiendo la secuencia de arranque vista en los laboratorios del apartado 2.1.2 y se podrá ver que el TeraTerm muestra el mensaje que enviamos:



Y si el usuario escribe una letra, esta se vuelve a enviar.

3.3.2 SCI

El puerto SCI (Serial Comunication Interface) es un periférico que al igual que la UART, permite las comunicaciones serie.

Consta de dos líneas y soporta comunicaciones asíncronas entre la CPU y otros periféricos que usen el estándar NRZ.

Utiliza dos pines externos para la comunicación (Rx y Tx), que en este dispositivo corresponden a los pines 28 y 29 de la GPIO.

Cuenta con dos FIFO independientes para la transmisión y recepción de 16 niveles, cada una con un registro buffer asociado, y la velocidad de transmisión es ajustable mediante el Baud rate:

$$SCI\ asynchronous\ baud = \frac{LSPCLK}{(BRR + 1) * 8}$$

Donde BRR es el valor de 16bit en decimal del registro de selección del Baud y LSPCLK el reloj periférico de baja velocidad (37.5 MHz).

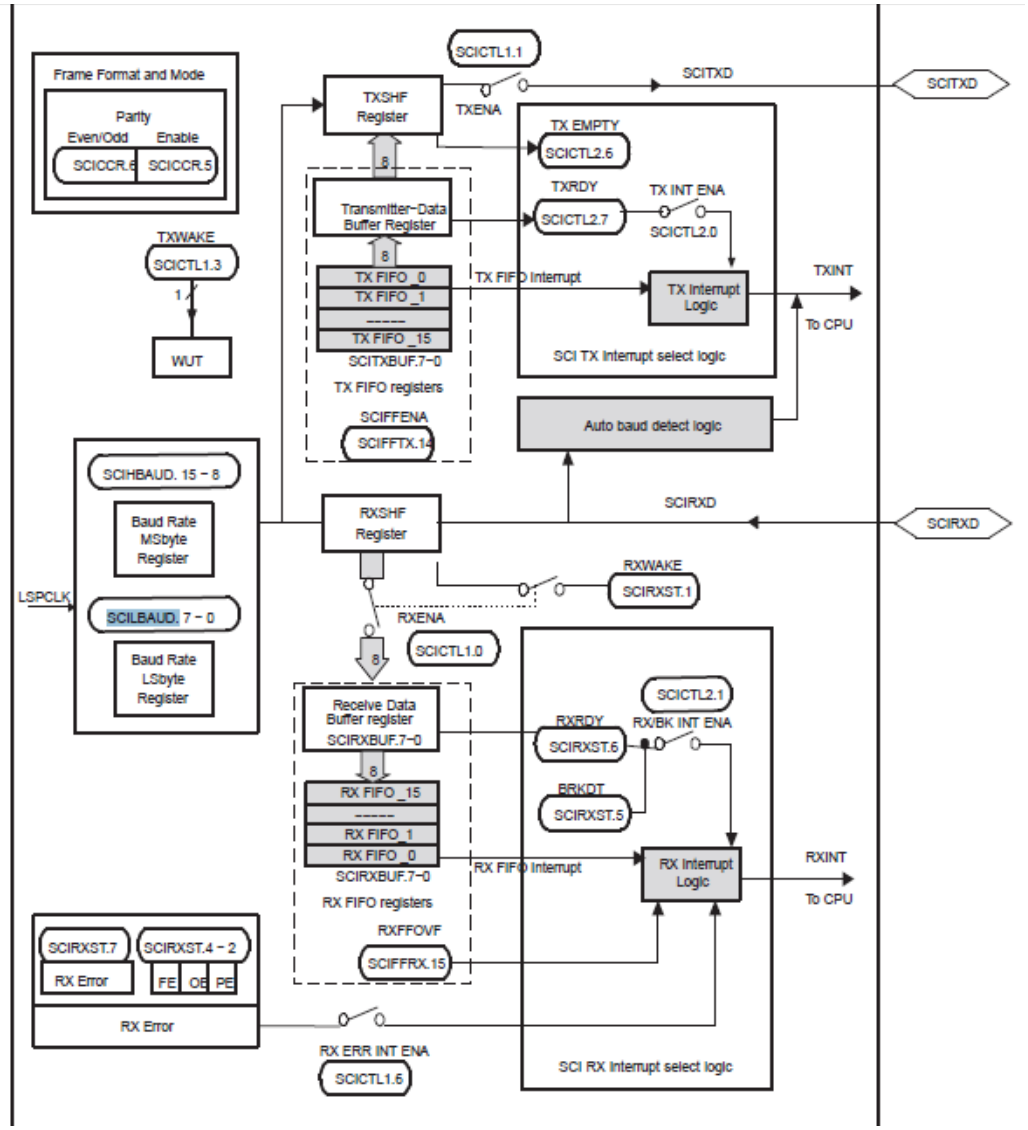


Ilustración 29: Bloque de diagrama SCI.

El formato de la transmisión es similar al de la UART, con un bit de comienzo, seguido de los datos a transmitir (de 1 a 8 bits), opción del bit de paridad, uno o dos bits de parada y un bit extra para distinguir entre dirección o dato si se configura en el modo de bit de dirección.

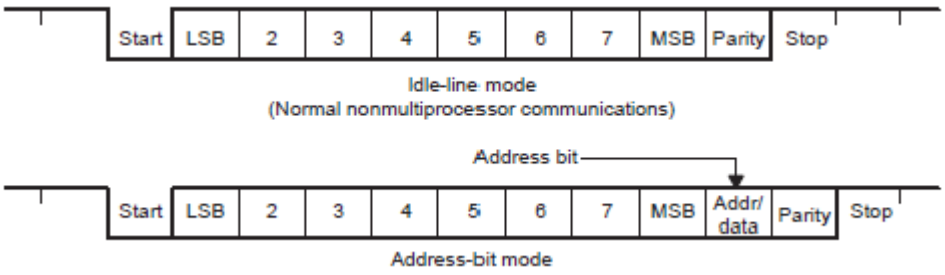


Ilustración 30: Secuencia de transmisión SCI.

Como se comento al principio, el objetivo del proyecto es tener un código funcional utilizando los periféricos del subsistema C28 (y el subsistema analógico), por lo que para las comunicaciones del sistema final, se utilizará el SCI.

Inicialmente, al igual que se hizo con la UART en el apartado anterior, se realizará un ejemplo más sencillo para la configuración del SCI y la comunicación entre el DSP y el TeraTerm, cuyo código se explica a continuación.

El ejemplo consiste en un bucle en el que se envía un carácter y se espera a recibir otro para enviar el siguiente.

Para ello, primero el subsistema maestro deberá dar control al C28 de los pines 28 y 29:

GPIOPinConfigureCoreSelect(GPIO_PORTE_BASE, GPIO_PIN_4, GPIO_PIN_C_CORE_SELECT); //28

GPIOPinConfigureCoreSelect(GPIO_PORTE_BASE, GPIO_PIN_5, GPIO_PIN_C_CORE_SELECT); //29

Y en el C28 configuramos dichos pines para que actúen como Rx (GPIO28) y Tx (GPIO29):

GpioG1CtrlRegs.GPADIR.bit.GPIO28 = 0; //set as an input

GpioG1CtrlRegs.GPADIR.bit.GPIO29 = 1; //set as an output

GpioCtrlRegs.GPAQSEL2.bit.GPIO28 = 3; // Asynch input GPIO28 (SCIRXDA)

GpioG1CtrlRegs.GPAMUX2.bit.GPIO28 = 1; //gives it bits 01 to make it the SCIRXDA (I)

GpioG1CtrlRegs.GPAMUX2.bit.GPIO29 = 1; //sets 01 to make it SCITXDA (O)

Posteriormente, se configuran los registros del SCI para que envíen y reciban datos de longitud 8 bits, con 1 bit de parada, sin paridad y el modo IDLE compatible con RS-232:

SciaRegs.SCICCR.bit.SCICHAR =(0x0008-1);

SciaRegs.SCICCR.bit.STOPBITS = 0;

SciaRegs.SCICCR.bit.PARITYENA = 0;

SciaRegs.SCICCR.bit.ADDRIDLE_MODE = 0;

Además, se habilitan Rx y Tx para que los datos se reciban y transmitan directamente por los pines 28 y 29 y se habilitan los bits que controlan las peticiones de interrupción causadas por las banderas TXRDY y BRKDT que indican que la FIFO está disponible o que tiene un dato respectivamente.

Ajustamos así mismo la velocidad de transmisión para 57600:

```
SciaRegs.SCICTL1.bit.RXENA = 1;
SciaRegs.SCICTL1.bit.TXENA = 1;
SciaRegs.SCICTL2.bit.TXINTENA =1;
SciaRegs.SCICTL2.bit.RXBKINTENA =1;
SciaRegs.SCIHBAUD    =0x0000;
SciaRegs.SCILBAUD    =0x0050;
```

El dispositivo permite la conexión interna entre los pines de transmisión y recepción. Para deshabilitar esta opción, se mantiene a cero el bit LOOPBKENA del registro de control.

```
SciaRegs.SCICCR.bit.LOOPBKENA =0;
```

Para finalizar la configuración del SCI, se libera el SCI del reset:

```
SciaRegs.SCICTL1.all =0x0023;
```

Después se inician las FIFO:

```
SciaRegs.SCIFFTX.all=0xE040;
SciaRegs.SCIFFRX.all=0x2044;
SciaRegs.SCIFFCT.all=0x0;
```

Dentro del bucle, se envia el carácter mediante la siguiente secuencia, la cual espera a que la FIFO de transmisión tenga un dato valido y entonces pone el dato en el buffer de transmisión, del cual llega directamente al pin de transmisión y se muestra en el terminal:

```
while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
SciaRegs.SCITXBUF=a;
```

Una vez enviado el dato, se espera a recibir otro dato desde el terminal. Cuando la FIFO de recepción está disponible el dato llega al DSP y se incrementa el valor del carácter inicialmente enviado.

```
while (SciaRegs.SCIFFRX.bit.RXFFST !=1) { }
ReceivedChar = SciaRegs.SCIRXBUF.all;
//if (ReceivedChar != SendChar) error();
SendChar++;
```

Una vez hecho esto, conectando los equipos y configurando el TeraTerm para que reciba y transmita con una velocidad de 57600, se podrá observar que cada vez que se envia un dato a través del TeraTerm el dato recibido se incrementa.

La configuración del SCI para el dispositivo final es muy similar, si bien, el procesado de los datos enviados y recibidos es diferente.

En este punto, se utilizará el código del dispositivo anterior (Delfino) modificando todos los registros para adaptarlos al nuevo dispositivo.

Este capítulo se centrará en la parte de dicho código relacionada con las comunicaciones, y el resto de modificaciones se verá en el próximo apartado.

Como se vio en el ejemplo anterior, inicialmente habrá que dar el control de los pines 28 y 29 desde el M3 al C28 y después configurar dichos pines en el código del C28 para que actúen como Rx y Tx

La configuración del SCI y las FIFO es la misma que la del ejemplo (57600, 8 bit de datos, sin paridad, 1 bit de parada...), salvo que esta vez no se habilitan las interrupciones ya que no son necesarias.

Al final de dicha configuración, se inicializará una tabla necesaria para obtener el CRC en el MODBUS, que es el protocolo visto desde el lado del esclavo que lleva a cabo el procesamiento de los datos recibidos.

En el main principal, dentro del bucle while, existen tres maquinas de estados:

La máquina de recepción (maq_est_rxon) que se encarga de la recepción de mensajes, la máquina de estados de transmisión (maq_est_txon) que se encarga del envío de mensajes y el modbus que ya se mencionó anteriormente.

Para la máquina de estados de recepción se actua igual que en el ejemplo antes mencionado, esperando a que la FIFO tenga un dato y almacenando dicho dato posteriormente en el bus de recepción, incrementando a su vez un contador que permitirá el avance de la máquina de estados del modbus.

```
if( (SciaRegs.SCIFFRX.bit.RXFFST != 0) )  //si no esta vacia la fifo
{
buf_rxon_28335[cont_rxon_28335++] = SciaRegs.SCIRXBUF.all;  //recibe dato
cont_rxon_28335++;

}
```

Se procede igual con la máquina de estados de transmisión:

```
while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
//FIFO de transmision no vacia
cont_txon_28335--;
SciaRegs.SCITXBUF = buf_txon_28335[ind_txon_28335++]; //cargo dato
```

En la máquina de estados del modbus se comprueba inicialmente si el dato recibido concuerda con la dirección del esclavo, si es así se pasa al siguiente estado, inicializando previamente los parámetros, si no, se mantiene en el estado de reposo.

El siguiente dato que llegue deberá ser el código de función del modbus para escritura o lectura en los registros. Si no es una instrucción valida, se generará un código de error, y si lo es, se irá comprobando en los siguientes estados si el mensaje recibido es válido o no.

Finalmente, si el dato que llegó es válido, se realizará la función de lectura o escritura en los registros según el código de la función que llegó.

Una vez configurado el registro, se podrá utilizar un código de Labview el cual está programado para probar este dispositivo de potencia.

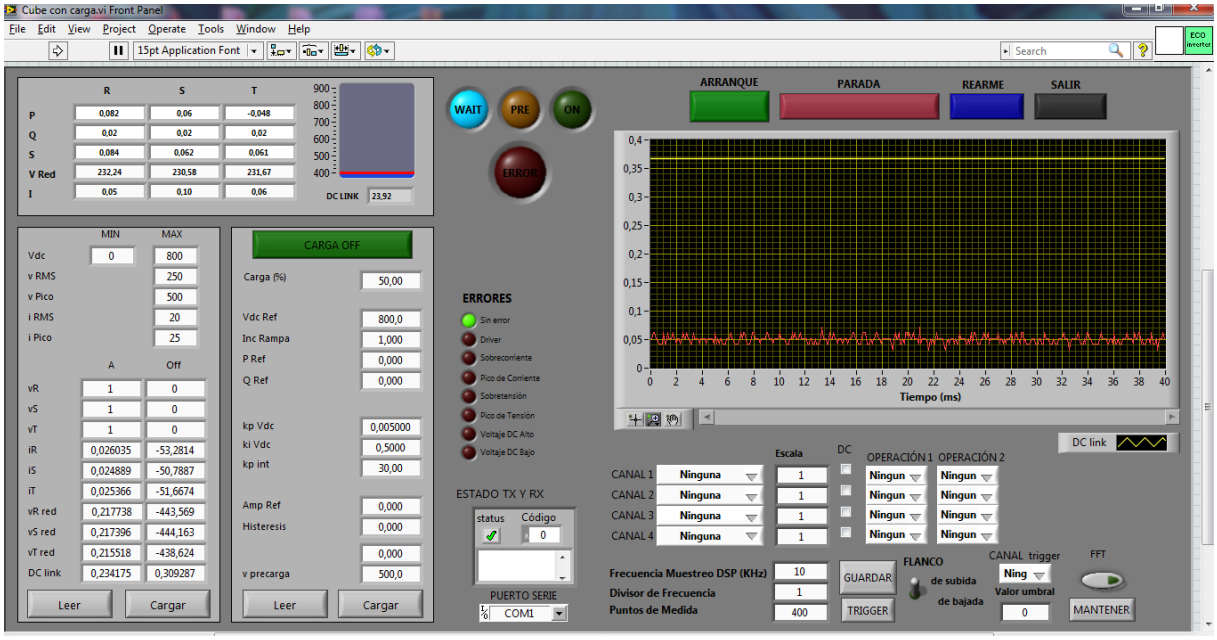


Ilustración 31: Código Labview para el dispositivo de potencia.

Así, configurando el puerto serie para que se corresponda con el que se comunica el dispositivo, se podrá observar que esta comunicación se produce.

4 PROBLEMAS ENCONTRADOS

A la hora de programar un DSP que no es suficientemente conocido, e incluso aunque el usuario tenga cierto manejo del mismo, es normal que en las primeras pruebas del código surjan errores a la hora de compilar o este no realice las tareas que se pretendía que hiciera.

En este capítulo se muestran los errores que surgieron a medida que avanzaba la configuración del sistema final, con el fin de evitar posibles errores futuros en proyectos similares, ya sean trabajos con el mismo dispositivo aquí usado, o bien fallos genéricos en la programación de un sistema embebido.

4.1 Accesibilidad de los pines GPIO

Para trabajar con el MCU se utiliza una tarjeta (ControlCARD) en la que este está integrado y que permite un acceso fácil a los pines de entrada y salida y la programación del MCU mediante la inserción de dicha tarjeta en la dockstation.

Esta tarjeta tiene una conexión DIMM de 100 pines que conectan directamente pines del MCU con los pines de la placa de trabajo.

No obstante, no todos los pines del DSP tienen un acceso directo a esta placa de trabajo y puede que a la hora de intentar obtener señal desde un pin que se ha configurado correctamente (dando el control al subsistema C28 desde el M3 y configurando el pin GPIO para que funcione como se requiere dentro del código del C28), no se consiga.

Es esto mismo lo que pasó al configurar el GPIO6 como salida PWM, ya que al configurar los cuatro módulos PWM como se explico en el capítulo 3.1, y observar dichas ondas mediante un osciloscopio, se podían observar las ondas que generaban los tres primeros módulos (asociados a los pines 0, 2 y 4) pero no la del cuarto modulo (asociada al pin 6).

Para solucionarlo, fue necesario conocer el conexionado interno entre los pines del DSP y el DIMM.

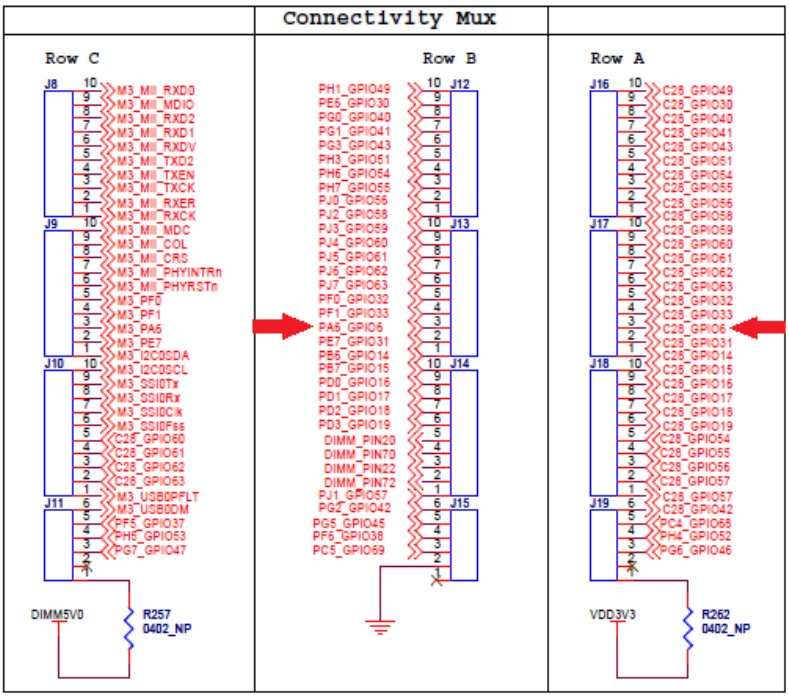


Ilustración 32: Pines de la ControlCARD.

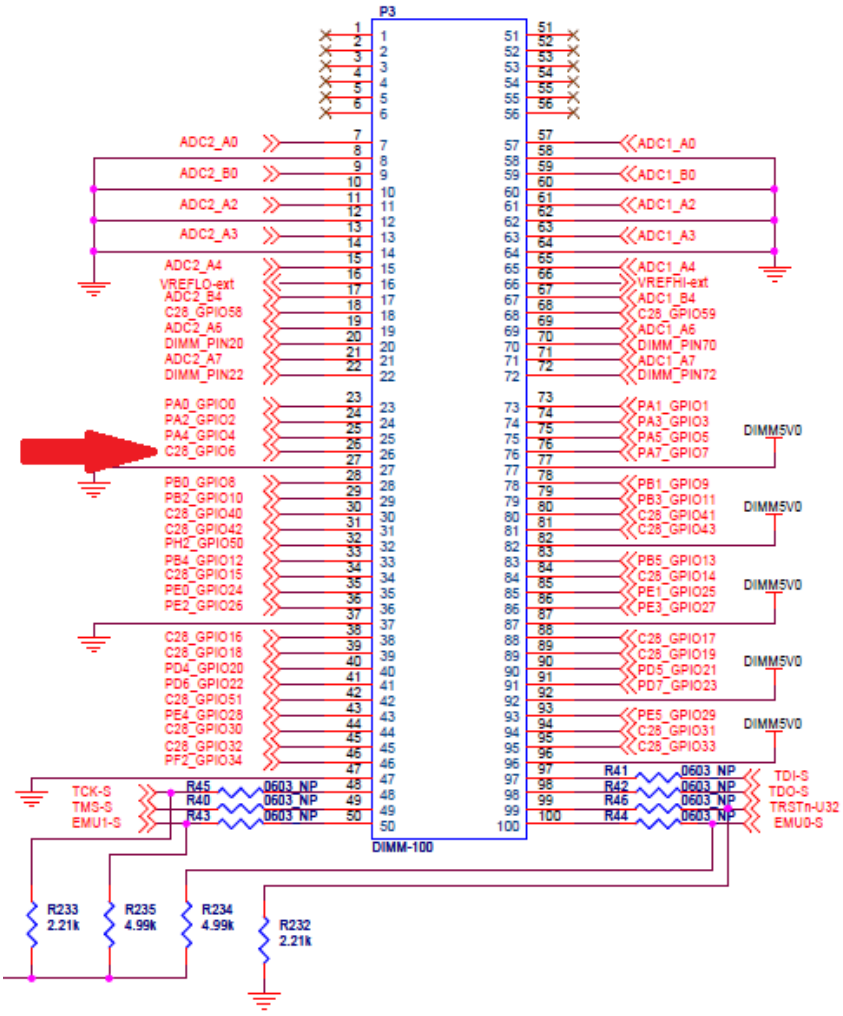


Ilustración 33: DIMM 100.

Como se puede observar en la ilustraciones, el GPIO6 no está conectado directamente al DIMM, si no que este va del MCU a la fila de pines de la ControlCARD (fila B) y es el pin 3 del puerto J17 de la fila A el que está conectado al DIMM 100 de la placa.

La solución pasa por conectar un jumper entre dichos pines para conectar ambas líneas.

4.2 Inserción del código antiguo.

Como se comento con anterioridad, para desarrollar el código final se utilizó un código previo diseñado para el DSP anterior (Delfino).

Ambos dispositivos disponen de periféricos similares, pero se acceden a ellos desde diferentes registros en uno u otro MCU.

Es por ello que a la hora de introducir el código anterior, se tendrá que cambiar los registros que dan acceso a los periféricos del Delfino por aquellos que dan acceso a los mismos periféricos del Concerto. Además, se observan una serie de librerías que también habran de modificarse o cambiarse por otras adecuadas para nuestro MCU.

Inicialmente, ya que el objetivo del proyecto está pensado para que el código sea funcional dentro del subsistema C28, habrá que dar control a este subsistema de todos los pines GPIO que se vayan a utilizar para la conexión entre el DSP y el dispositivo de potencia.

Para ello, se habilitan los pines desde el M3 que aparecen en la siguiente tabla, en la que se muestra la función que realizara cada uno, como ya se ha visto en capítulos anteriores del documento.

PIN	PUERTO	ENTRADA/SALIDA	FUNCION
GPIO0	A	SALIDA	EPWM1A (CANAL R)
GPIO2	A	SALIDA	EPWM2A, (CANAL S)
GPIO4	A	SALIDA	EPWM3A, (CANAL T)
GPIO6	A	SALIDA	EPWM4A, (CANAL CH)
GPIO1	A	SALIDA	PIN DE TESTEO
GPIO7	A	SALIDA	K2
GPIO22	D	SALIDA	K1
GPIO13	B	SALIDA	DRIVER
GPIO21	D	ENTRADA	ERROR
GPIO27	E	SALIDA	CLR_ERROR
GPIO28	E	ENTRADA	Rx(RECEPCION COMUNICACIONES)
GPIO29	E	SALIDA	Tx(TRANSMISION COMUNICACIONES)

Tabla 3: GPIO's utilizados.

Una vez configurados los pines GPIO, se introduciran en el bucle infinito del main del C28 las máquinas de estados correspondientes a las comunicaciones ya comentadas anteriormente (maq_est_rxon, maq_est_txon y modbus) y una maquina de estados que se encarga del control del sistema (StateMachine).

Así mismo, en la interrupción del ADC se introduciran una serie de funciones que permitirán ajustar el valor obtenido de las corrientes y tensiones a través del convertidor y asignar un nuevo Duty a los canales R, S, T y CH.

Para poder utilizar dichas funciones, habrá que añadir una serie de archivos (.c) y librerías (.h) que se comentan a continuación.

Inicialmente, se creó una carpeta con todas las librerías de las que se partía modificadas para su uso en el nuevo MCU, para agregar posteriormente dicha carpeta al nuevo código (tal y como se vio en el capitulo 2.1.2). Las modificaciones consistieron en el cambio del acceso de estas librerías a otras propias del dispositivo antiguo, por las del nuevo dispositivo.

Dichas librerías son las siguientes:

- "background.h": Incluye las declaraciones de las funciones StateMachine (maquina de estados de control del sistema), ClearError (resetea el estado de error a través del pin asociado al mismo), ResetCtrl (resetea las variables de control), SlideWindow, CheckAlarm y pot (funciones para adaptar los datos obtenidos para la visualización en Labview). Todas estas funciones están definidas en el archivo background.c.

- “config.h”: Inicialmente, contenía las declaraciones de las funciones de configuración del ADC, PWM, GPIO, y UART del sistema, pero se modifico para que solo incluyera las de la UART, y otras funciones para la inicialización del sistema. Estas se encuentran definidas en el archivo Config.c.
- “cubeV2.h”: Es la librería que se utiliza en el main y que incluye todas las demás que se van a usar.
- “def.h”: Contiene definiciones de funciones básicas, como encender o apagar un pin, para un uso mas rápido en el código principal. Además define los estados y constantes para la máquina de estados, así como los códigos de error.
- “e_main.h”: Define las cabeceras del main y una serie de constantes que se utilizaran.
- “e_memfija.h”: Permite el uso del archivo e_memfija.c que junto al archivo MEMFIJA.cmd permite ubicar las variables en un espacio de memoria física concreto. En el siguiente apartado se comentara más en profundidad.
- “e_modbus_v_5.h”, “e_monitorizacion_auxiliar.h” y “e_var_comp_modbus.h”: Contienen definiciones para las funciones de comunicación comentadas en el capitulo anterior.
- “mod.h”: Contiene las definiciones de las funciones que se encuentran en el archivo mod.c para controlar los módulos PWM según queramos configurar el dispositivo de potencia (inversor, rectificador, back to back…).
- “var.h”: Incluye las definiciones de una serie de estructuras, rellenadas posteriormente en var.c, que contienen las medidas tomadas por el sistema, las correcciones para esas medidas, valores de referencia, estados de errores…

4.3 Archivos CMD

Como se comentaba en el apartado anterior, existen unos archivos cuya extensión es CMD (comando), que permiten alojar las variables del programa en un espacio de memoria concreto.

Estos archivos dividen la memoria en dos páginas y estas dos páginas están a su vez divididas en secciones de tamaño variable. La pagina 0 por defecto alberga la memoria de programa y la pagina 1 la memoria de datos. Cada una de las diferentes secciones de memoria tiene una longitud definida y está localizada a partir de una dirección de memoria concreta, las cuales se pueden modificar, por ejemplo, uniendo dos secciones consecutivas para obtener una de mayor tamaño.

Al compilar nuestro código, se observaba en CCS un error de espacio de memoria con una serie de variables concretas.

El fallo estaba en el archivo MEMFIJA.cmd, que provenía del código anterior y por tanto, colocaba las variables en zonas de memoria ya ocupadas en el dispositivo.

Para solucionarlo, se buscó una sección disponible dentro de la pagina 1 (memoria de datos) en el archivo “F28M35HM52C1_c28.cmd” proporcionado en las librerías del dispositivo. Esta correspondía al registro “RAML0” que inicialmente comenzaba en la dirección 0x008000 de memoria y tenía una longitud de 1000. Sin embargo, el espacio de ocupación de las variables era superior al tamaño de esta sección, por lo que se eliminó la sección “RAML1” consecutiva a esta y también disponible y se cambió la longitud inicial a 2000.

Así mismo, se escogió la sección “RAMS0” junto con la “RAMS1” para guardar las variables locales, y hubo que ampliar la sección “FLASHA” de la página de memoria de programa anexándola con la sección “FLASHD”.

Dentro del archivo cmd (MEMFIJA.cmd), las variables se colocan a partir de la dirección 8000, indicando en cada una de ellas la longitud que ocupan.

Ciertas partes del código de la parte de comunicaciones hacian referencias directas a las direcciones de estas variables, por lo que para el acceso correcto a dichas variables se tuvo que cambiar dentro del código estas direcciones a las actuales, solucionandose así el problema.

4.4 Comunicaciones

En la comunicación entre el dispositivo y Labview surgieron varios problemas que se comentan a continuación.

Inicialmente, una vez configurado el registro SCI y conectado los elementos externos (adaptador del puerto serie, placa de adaptación y cables en Rx y Tx) se probó comunicar el dispositivo con Labview sin éxito, ya que este ultimo mostraba errores de timeout.

Se decidió por tanto realizar los ejemplos comentados en el apartado 3.3 del documento, con el fin de comprobar si el fallo estaba en la comunicación de los dispositivos y no en otra parte del código.

Tras probar dichos ejemplos depurándolos mediante breakpoints y utilizando el terminal TeraTerm, se observaba como el dispositivo ni transmitía aquello que se suponía debía transmitir ni llegaba a recibir ningún dato en el buffer de recepción.

Uno de los problemas fue el ajuste de la velocidad de transmisión entre el TeraTerm y el dispositivo, ya que este estaba configurado para una transmisión de 115200 y el TeraTerm recibía a 9600, por lo que se cambio el código y la configuración del terminal para que ambas fueran de 57600, ya que esta es la velocidad a la que se comunica Labview.

Aunque Labview seguía emitiendo fallos de timeout, el terminal si mostraba los datos enviados, es decir, la transmisión desde el DSP funcionaba, pero la recepción no.

Sabiendo que la configuración de los registros de recepción era la correcta, el problema se reducía a un fallo de hardware.

Por ello se comprobó inicialmente mediante un osciloscopio que efectivamente el dato enviado a través del TeraTerm se recibía y que todos los cables estaban en buen estado.

El problema estaba en uno de los switches de la ControlCARD, el cual se había conmutado accidentalmente.

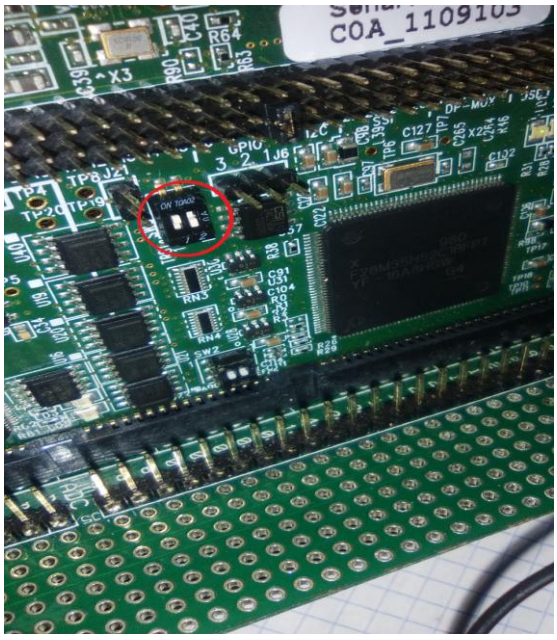


Ilustración 34: Switch SW3

El switch en concreto era el SW3, el cual se encarga de habilitar las señales de comunicación TRST/ISO SCI. Este switch tiene dos interruptores, el primero se encarga de conectar o desconectar la señal TRST del circuito JTAG al DSP, y el segundo, motivo del fallo, se encarga de activar la comunicación Rs-232 a través de los pines 2 y 42 del DIMM bloqueando el GPIO28 en alto nivel en posición de ON o habilitar el pin GPIO28 como GPIO. Esta última opción es la que interesa, ya que es este pin el que se quiere conectar con el dispositivo de potencia y dentro del código se puede configurar como Rx.

Una vez realizado esto, los ejemplos de comunicación con la UART y el SCI eran funcionales, así que se volvió a comprobar el código final.

En este punto y tras el uso de breakpoints, se observaba que el dispositivo llegaba a recibir datos en el buffer de recepción, y por tanto la máquina de estados asociada a la recepción funcionaba correctamente, sin embargo, la máquina de estados del modbus, comentada en el apartado 3.3.2, no avanzaba del estado de reposo.

Este bloqueo del código es debido a que el incremento de la variable "cont_rxon_28335" no se produce una vez recibido el dato y por tanto la máquina de estados del modbus, que inicialmente esta en reposo, no entra en el "if" que comprueba si el dato recibido es válido, ya que esta variable permanece a cero siempre.

Al no observarse cambio en esta variable, se pensó que el problema estaba nuevamente en la localización de las variables globales en memoria.

La seccion de memoria que albergaba las variables locales(S0 en este caso) era una seccion de memoria compartida por ambos subsistemas (M3 y C28). Estos bloques de memoria son accesibles en escritura y lectura por el subsistema maestro, sin embargo solo son accesibles en modo lectura para el subsistema de control.

Es por ello que la variable en cuestion no cambiaba de estado, ya que no podiamos reescribirla a traves del C28.

Para habilitar la escritura en dicha seccion de memoria, se dió permiso desde el subsistema maestro mediante la funcion "RAMMReqSharedMemAccess" , y añadiendo la librería ram.h y el archivo ram.c donde se encuentran las definiciones y el codigo de dicha función (localizada en la driverlib).

5 CONCLUSIONES Y RESULTADOS

Antes de probar el dispositivo en el equipo real hay que realizar pruebas del correcto funcionamiento del DSP mediante el uso de un generador de ondas, un osciloscopio, la placa de adaptación y el software Labview.

Inicialmente, se hizo el conexionado entre la placa y el dispositivo, de acuerdo con la PCB de la placa, conectando las distintas salidas de los canales de tensión e intensidad de la placa a los pines del ADC del dispositivo y conectando el generador de ondas con una frecuencia de 50Hz y una tensión de pico a pico no superior a 2 voltios a la placa, con el fin de evitar quemar los pines GPIO del DSP.

Mediante el osciloscopio se comprobó que las ondas PWM generadas eran correctas y estas se transmitian en la placa correctamente.

Posteriormente, utilizando Labview, se visualizaron los distintos canales de intensidades y tensiones del ADC[8], observandose el cambio en los valores de cada canal, como se ve en la parte superior izquierda de la imagen y en la gráfica de la derecha, que muestra la onda senoidal que se ha introducido con el generador de ondas.

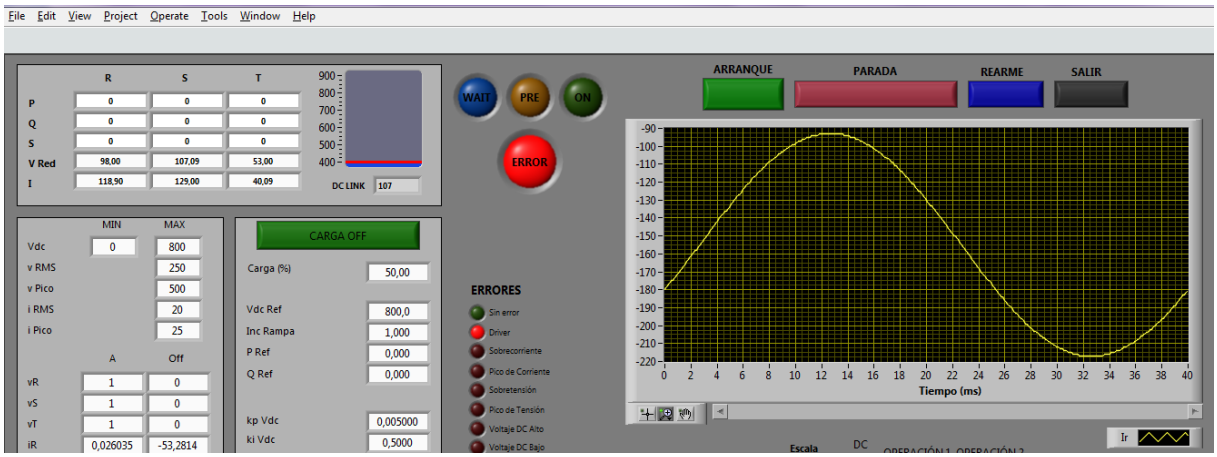


Ilustración 35: Canal Ir.

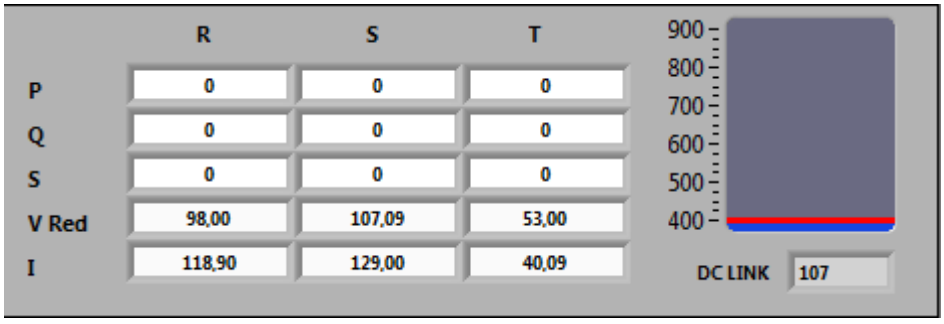


Ilustración 36: Medidas Ir

[8] Se pueden observar errores de driver o sobrecorriente, ya que estos pines no estan conectados. Esto no impide observar las medidas obtenidas que es el objetivo de esta prueba.

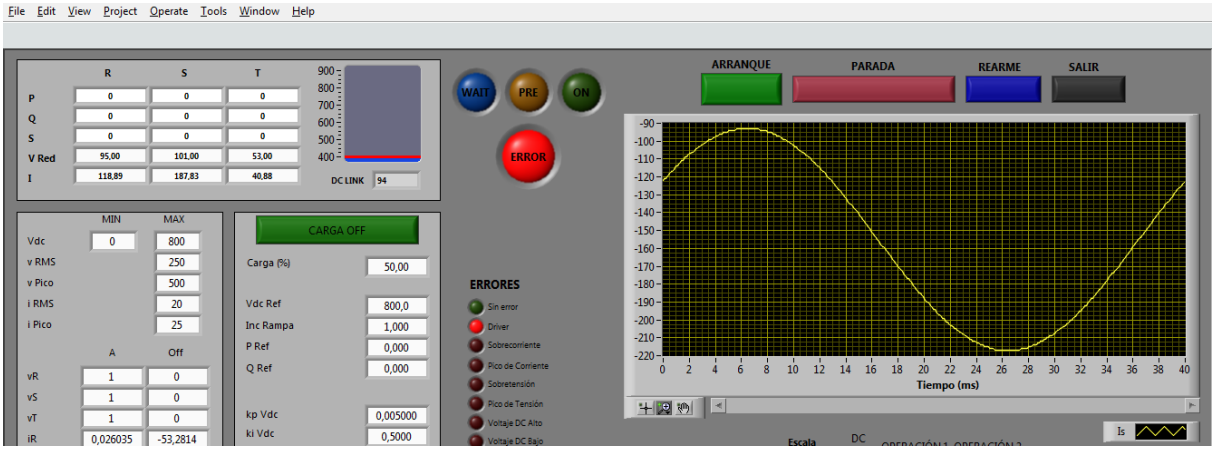


Ilustración 37: Canal Is

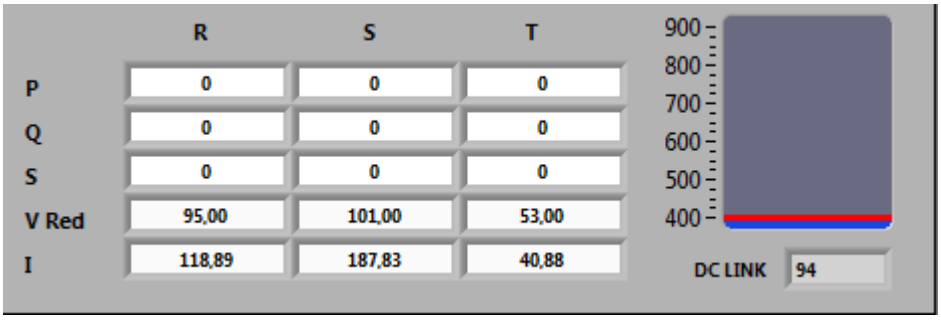


Ilustración 38: Medidas Is.

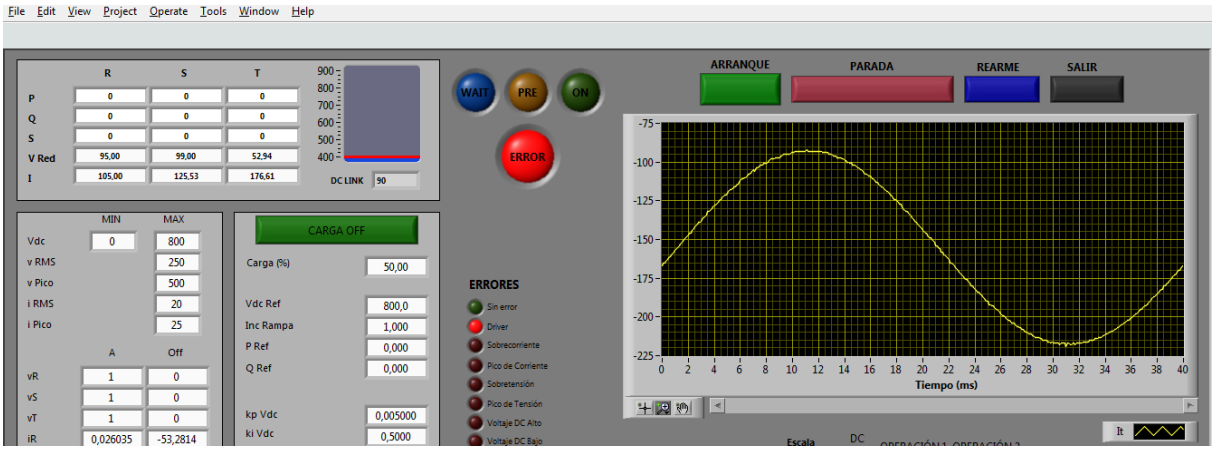


Ilustración 39: Canal It.

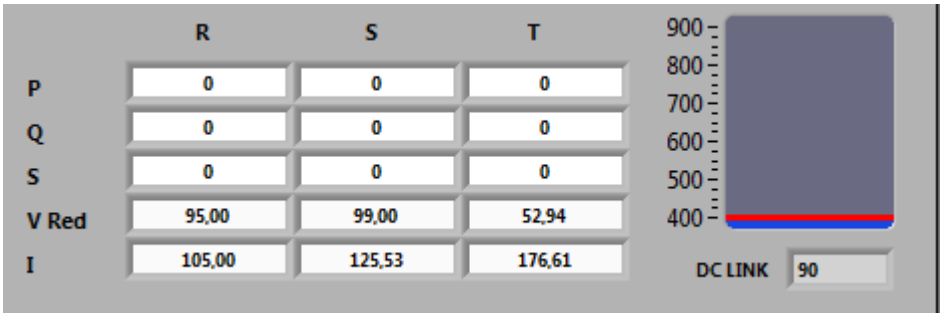


Ilustración 40: Medidas It.

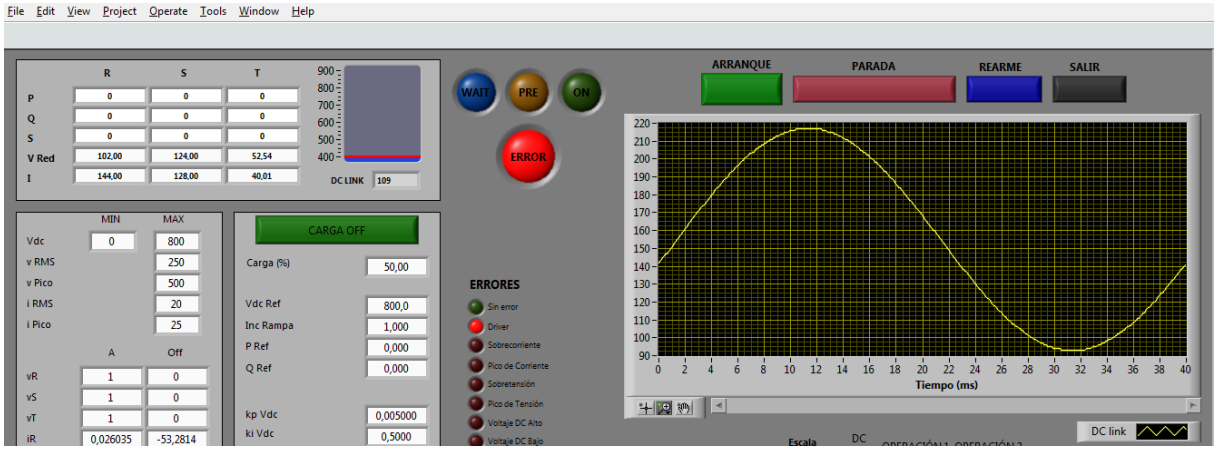


Ilustración 41: Canal Vdc.

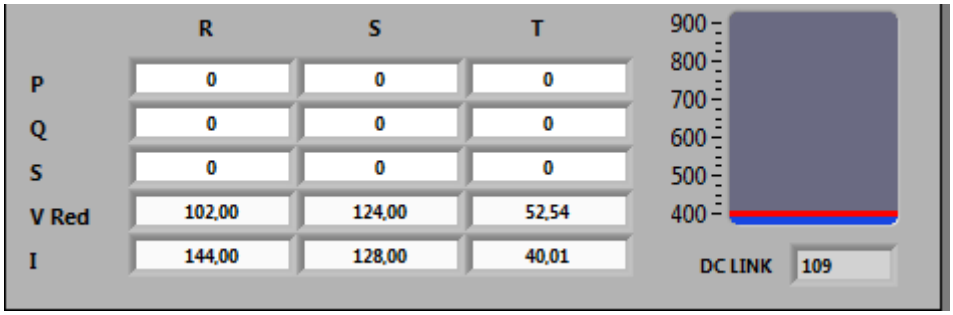


Ilustración 42: Medidas Vdc.

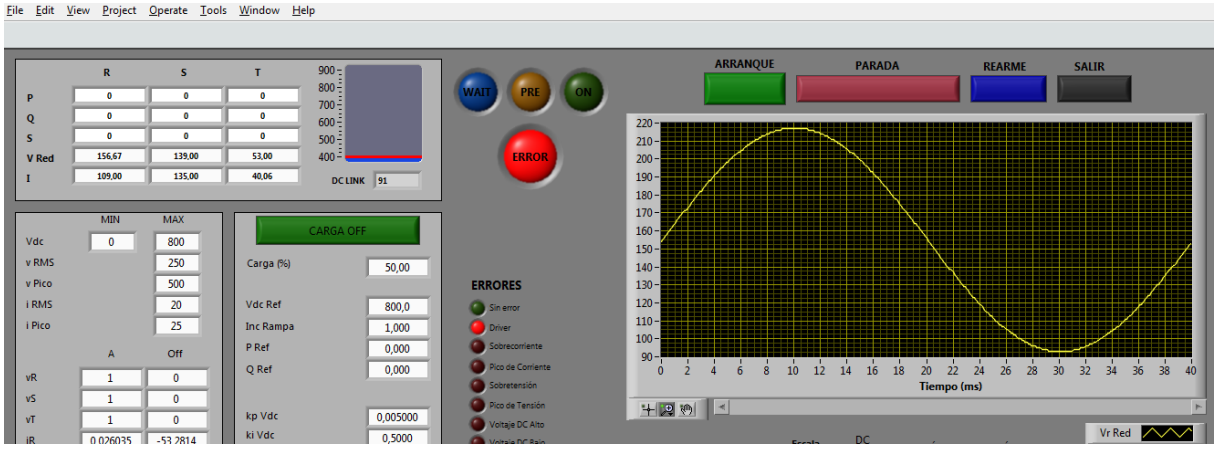


Ilustración 43: Canal VrRed.

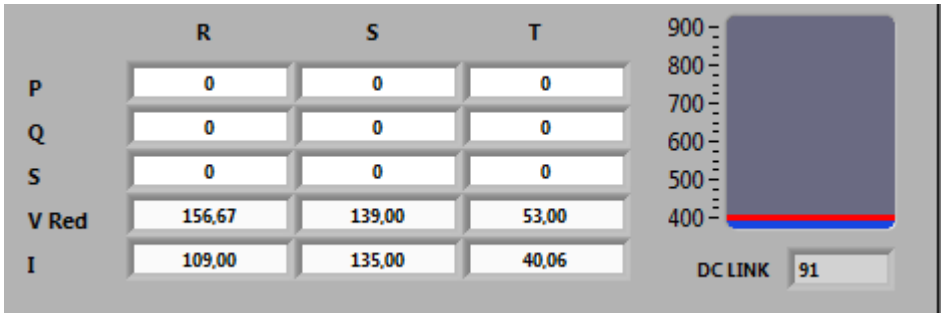


Ilustración 44: Medidas VrRed

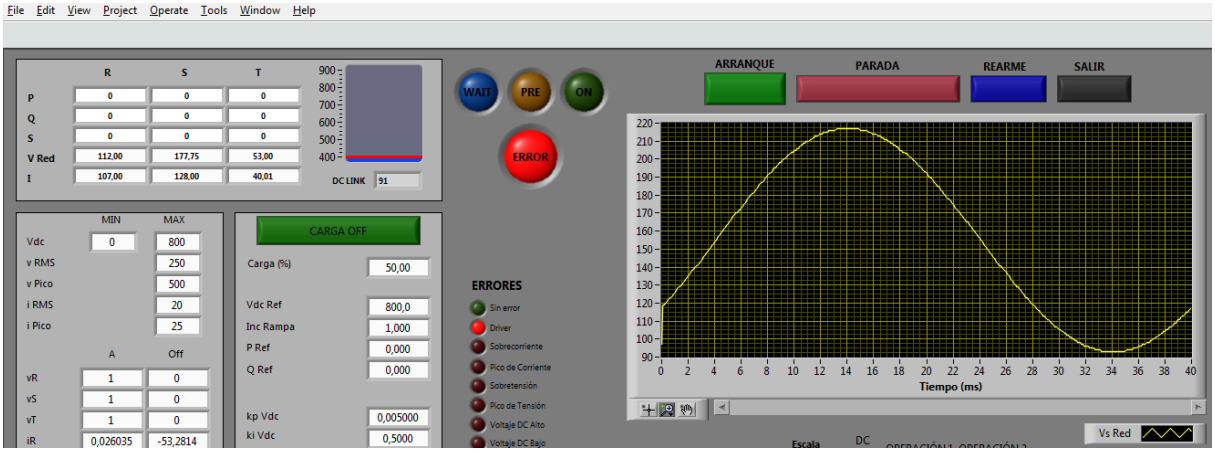


Ilustración 45: Canal VsRed.

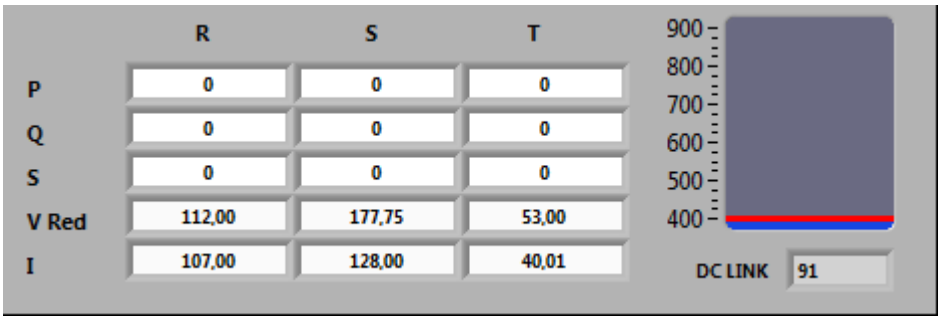


Ilustración 46: Medidas VsRed.

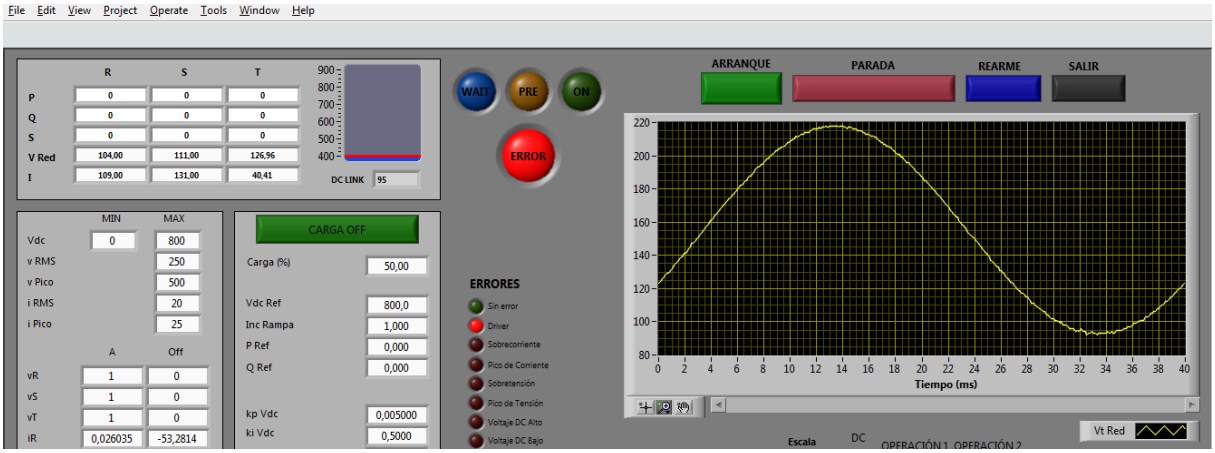


Ilustración 47: Canal VtRed.

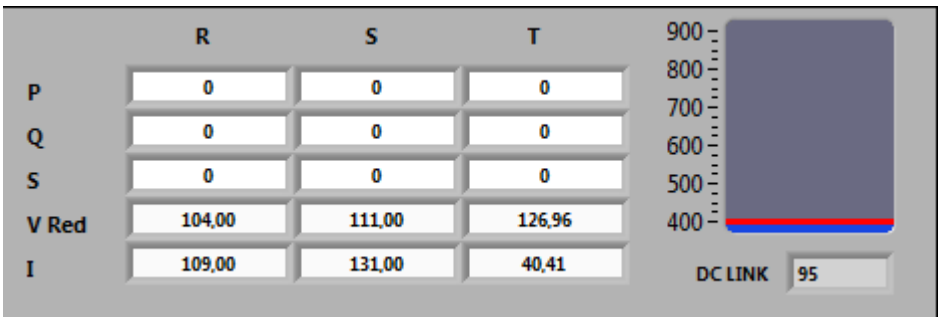


Ilustración 48:Medidas VtRed.

6 ACCIONES FUTURAS

Como se ha ido comentando a lo largo del proyecto, el DSP que se esta manejando incluye dos núcleos independientes, que permiten que la carga computacional sea menor que si fuera un solo núcleo.

En este sentido, el subsistema M3 está diseñado para encargarse de las comunicaciones y el C28 para el control.

Aunque inicialmente se plantea el uso de este dispositivo como solución a la elevada carga de la CPU del dispositivo anterior y a las interferencias en el convertidor ADC, el desarrollo del proyecto se limita a la funcionalidad del código en el subsistema C28.

Como acciones futuras, se propone realizar el código correspondiente a las comunicaciones desde el subsistema M3 y a través del puerto Ethernet, dejando al subsistema C28 encargado de los módulos PWM y del código de control del dispositivo de potencia, de manera que el aprovechamiento del dispositivo sea el mayor posible y se puedan cumplir los requisitos temporales que se mencionan.

7 BIBLIOGRAFÍA

- Sergio Toral Marín: Apuntes electrónica industrial. Departamento de ingeniería electrónica, Universidad de Sevilla.
- Texas Instruments: C200 Concerto Microcontrollers (Datasheet).
- Texas Instruments: Concerto F28M35x. Technical Reference Manual.
- Texas Instruments: Workshop Guide and Lab Manual
- Texas Instruments: F28M35x Peripheral Driver Library. User's Guide.
- National Instruments: Pagina web disponible en: <http://www.ni.com/labview/esa/>
- Universidad de Sevilla: Tutorial Labview.
- Texas Instruments: ControlSUITE.
- Texas Instruments: Concerto F28m35xx ControlCARD (infosheet).

Algunas de las imagenes presentadas se han tomado del buscador Google.