Universidad Nacional Autónoma de México
Facultad de Ingeniería
División de Ingeniería Eléctrica

Tutorial de la Tarjeta LaunchPad Delfino TMS320F28377S y Code Composer Studio

Compilación

M.I. Larry Escobar Salguero
Ing. Iván Menéndez Rosas
Ing. Luis Álvarez Fernández
Ing. Michel Olvera Zambrano
M.C. Samuel Vázquez Sánchez

Laboratorio de Procesamiento Digital de Señales
Departamento de Procesamiento de Señales, abril 2016

Índice general

Índice general I

1. Introducción **1**

2. MCU Delfino TMS320F28377S **2**

3. Tarjeta LaunchPad Delfino **5**

4. Code Composer Studio **8**
4.1. Creación de un proyecto en CCS 10

5. Ejemplos en lenguaje ensamblador **15**
5.1. Suma varias constantes en aritmética de punto fijo a 16 Bits 15
5.1.1. Direccionamiento Inmediato 15
5.1.2. Direccionamiento Directo 16
5.1.3. Direccionamiento Indirecto 17
5.1.4. Uso de la instrucción RPT 18
5.2. Promedio de una secuencia de datos 19
5.3. Manejo de los LEDs D9 y D10 como salidas 19

6. Resumen **21**

Bibliografía **22**

Apéndice 1 **23**

1. Introducción

El Procesamiento Digital de Señales (PDS) juega un papel importante en los sistemas electrónicos actuales, tales como computadoras personales, teléfonos celulares, tabletas digitales, entre otros. En prácticamente todos ellos el PDS está presente en aplicaciones multimedia, biomédicas, videojuegos, televisión y radio digital, por mencionar algunas. Por su parte, los Procesadores Digitales de Señales (DSPs) son herramientas dedicadas a la implementación de soluciones a problemas de PDS en tiempo real.

Texas Instruments (TI) es una compañía estadounidense fundada en 1951 dedicada al diseño y fabricación de semiconductores, la cual tiene una fuerte presencia en el mercado del cómputo embebido. Provee una amplia gama de microcontroladores (MCUs) y Procesadores Digitales de Señales (DSPs) para una gran diversidad de aplicaciones relacionadas con control y procesamiento de señales. Entre estos dispositivos se encuentra la familia C2000 que posee características orientadas al control e instrumentación de alto rendimiento computacional, al mismo tiempo que tiene los periféricos más comunes de un microcontrolador.

En este tutorial se presentan las características generales del MCU Delfino TMS320F28377S, la tarjeta de evaluación (LaunchPad) del mismo, así como una introducción al ambiente de desarrollo Code Composer Studio (CCS). Se incluyen ejemplos simples con el fin de comenzar a desarrollar aplicaciones de PDS.

2. MCU Delfino TMS320F28377S

La familia F2837x es un MCU C28x con una Unidad de Punto Flotante (C28x + FPU) basada en controladores con arquitectura de punto entero a 32 bits. La FPU efectúa operaciones de punto flotante en precisión simple utilizando el estándar IEEE 754. Esto permite la implementación eficiente de algoritmos de control y PDS a mayor precisión numérica, prescindiendo de un segundo procesador. Estas familias son las más avanzadas de la gama C28x.

Actualmente existen dos versiones de esta familia: una de dos nucleos (F2837xD) capaz de realizar hasta 800 millones de instrucciones por segundo (MIPS), y la de un núcleo (F2837xS) con capacidad de 400 MIPS. Sus principales características son:

- Arquitectura tipo Harvard modificada.
- Unidad Central de Proceso de 32-bits
  - Reloj de 200 [MHz].
  - Unidad de punto flotante (FPU) IEEE 754 de precisión simple.
  - Unidad trigonométrica (TMU).
  - Unidad de matemática compleja Viterbi (VCU-II).
- Acelerador programable de control (CLA)
  - Reloj de 200 [MHz].
  - Unidad de punto flotante (FPU) IEEE 754 con precisión simple.
  - Instrucciones en punto flotante con precisión simple.
  - Ejecuta código independiente del CPU principal.
- Memoria interna
  - Hasta 1 [MB] de memoria Flash.
  - Hasta 164 [KB] de memoria RAM.
- Pipeline de 8 niveles, que permiten un solapamiento máximo de 8 instrucciones en niveles de ejecución diferentes:
  - Búsqueda de instrucción: F1 y F2.
  - Decodificación: D1 y D2.

  - Lectura de operandos: R1 y R2.
  - Ejecución: X.
  - Escritura: W.
- Subsistema analógico
  - Cuatro convertidores analógico-digital (ADCs) con hasta 24 canales de 12-bits a 3.5 MSPS cada uno.
  - Tres salidas de convertidor digital-analógico (DAC) de 12 bits.
  - Ocho comparadores de ventana con referencias digitales a analógicas de 12 bits.
- Periféricos del Sistema
  - Dos interfaces externas de memoria (EMIFs) con soporte ASRAM y SDRAM.
  - Acceso directo a memoria (DMA) de seis canales.
  - Hasta 169 Pines de entrada/salida de propósito general (GPIOs) multiplexados e individualmente programables.
  - Controlador de interrupción de periféricos (ePIE).
  - Soporte para múltiples modos de baja energía (LPM).
- Periféricos de comunicación
  - USB 2.0 (MAC + PHY).
  - Soporte para puerto paralelo universal de 12 pines
  - Dos módulos para control de red de área (CAN).
  - Tres puertos SPI de alta velocidad (hasta 50 [MHz]).
  - Dos puertos seriales multicanal (McBSPs).
- Ejecuta instrucciones de 32 bits para mejorar la precisión numérica.
- Ejecuta instrucciones de 16 bits para mejorar la eficiencia en el código.
- Unidad aritmética lógica (ALU) de 32 bits.
- Unidad aritmética de registros auxiliares (ARAU), genera direcciones de memoria dato, realiza aritmética entre apuntadores en paralelo con operaciones de la ALU.
- Registro de corrimiento, ejecuta corrimientos hacia la derecha o izquierda de hasta 16 bits.
- Ejecuta multiplicaciones de 32 x 32 bits con resultado de 64 bits.
- Efectúa una operación multiplicación acumulación (MAC) de 32 x 32 bits en un ciclo de reloj.
- Efectúa dos operaciones MAC de 16 x 16 bits (DMAC) en un ciclo de reloj.
- Emulación de su funcionamiento en tiempo real.
- Protección de código.
- En un ciclo de instrucción puede ejecutar instrucciones que leen, modifican y escriben en memoria.

- Respuesta de interrupciones rápida con salvado automático del contexto.
- Sincronía de eventos con latencia mínima.

En la figura 2.1 se muestra el diagrama de bloques del DSP, donde se pueden observar los diversos periféricos que se mencionaron anteriormente.

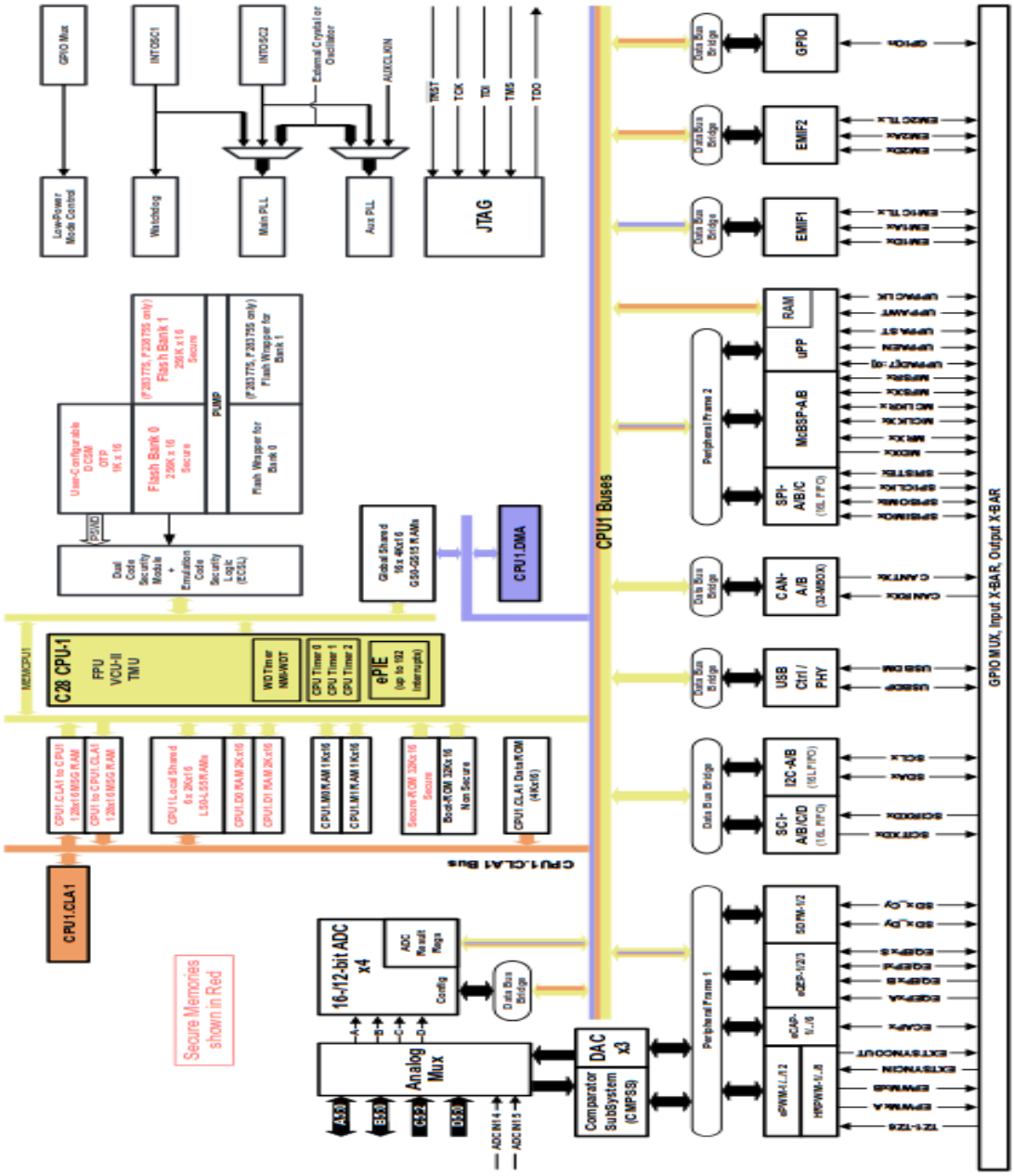

Figura 2.1: *Diagrama de bloques del TMS320F28337S*

3. Tarjeta LaunchPad Delfino

La tarjeta LauchPad Delfino C2000 es una plataforma de evaluación de bajo costo conveniente para empezar a desarrollar aplicaciones de procesamiento de señales o de control en tiempo real. Esta tarjeta contiene un MCU TMS320F28377S, así como un emulador JTAG XDS100 para cargar programas y sesiones de depuración desde una PC con el software Code Composer Studio.

En la figura 3.1 se muestra una imagen de la tarjeta LaunchPad donde se aprecian sus principales componentes. Entre ellos LEDs para uso del usuario (D2 y D3), headers para conexión de tarjetas de expansión (BoosterPacks) (J1/J3, J2/J4, J5/J7 y J6/J8). Cuenta con pines de alimentación externa (J10), y un botón para reiniciar el procesador (S3).

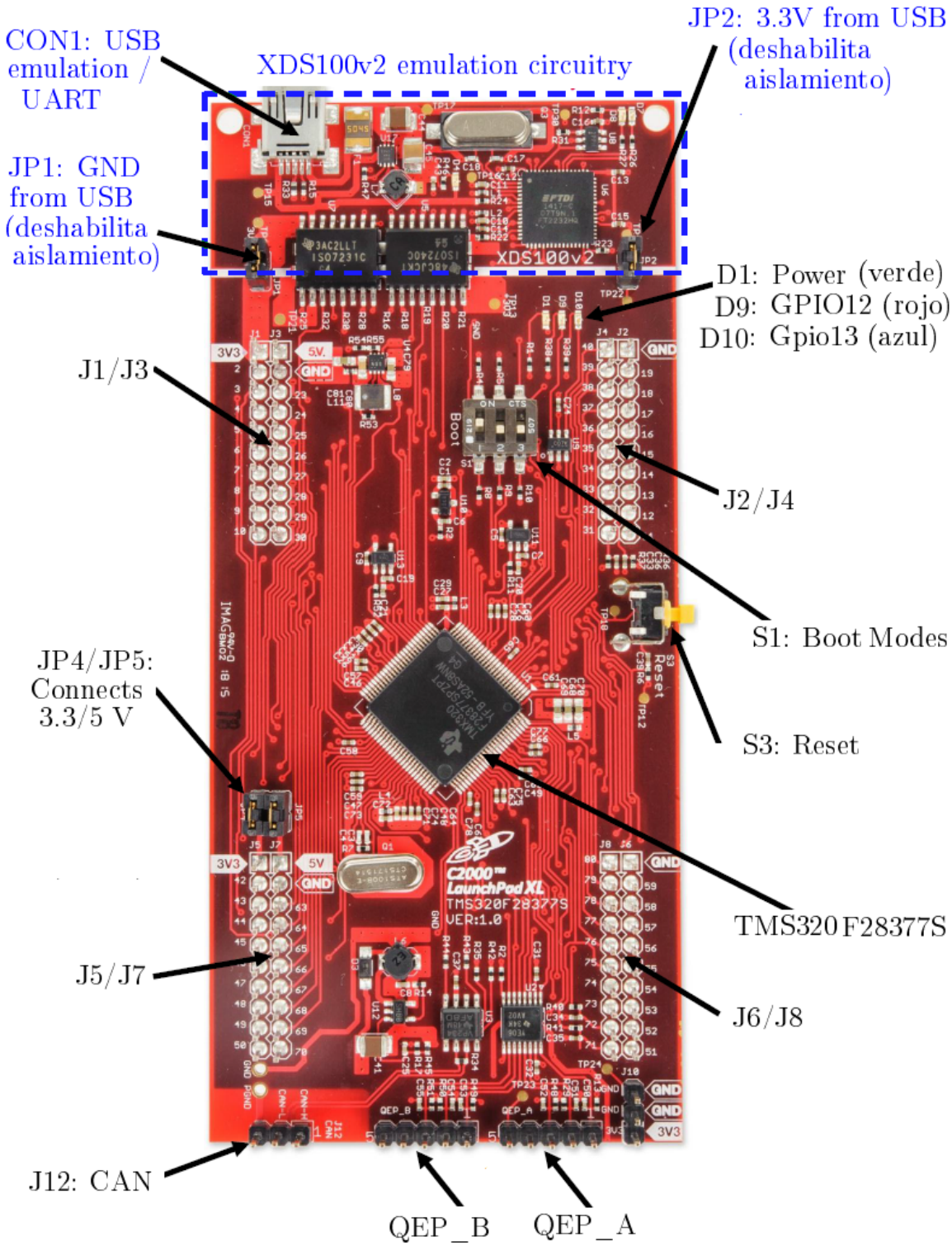

Figura 3.1: *Tarjeta LaunchPad Delfino.*

En la figura 3.2 se muestra el mapa de pines del LaunchPad. Los pines marcados como Pxx corresponden a los pines de entrada y salida (GPIO), así mismo se muestran los periféricos que se encuentran conectados a cada uno de los pines. La sección marcada como *BoosterPack Standard*, corresponde a los pines empleados en las diferentes tarjetas de expansión (Booster Packs) fabricadas por Texas Instruments.

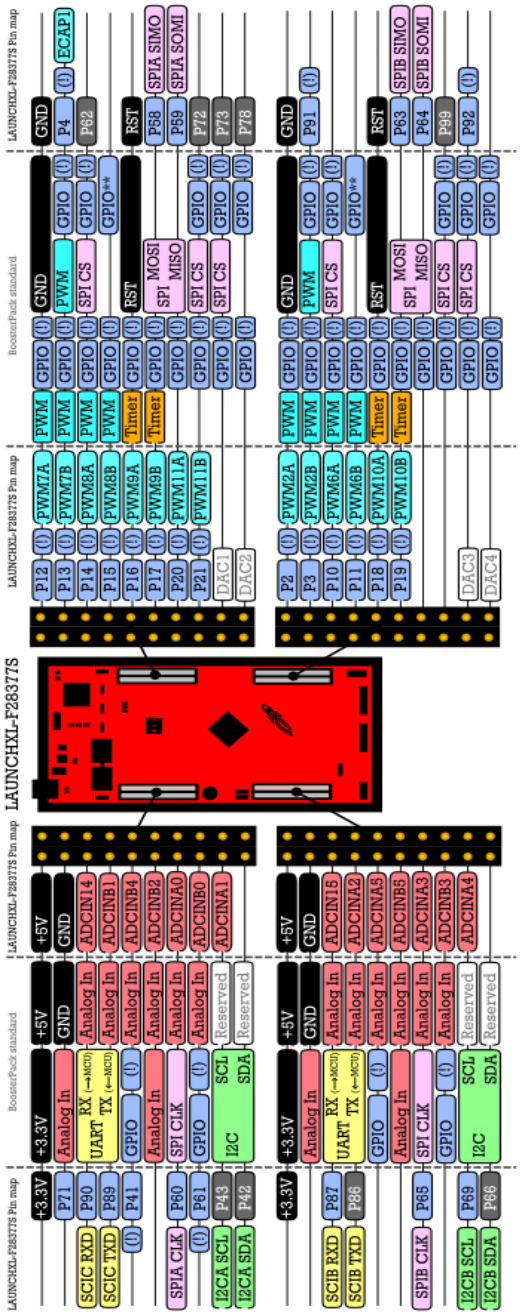

Figura 3.2: *Mapa de pines del LaunchPad Delfino [8]*

4. Code Composer Studio

Code Composer Studio (CCS) es un Ambiente de Desarrollo Integrado (IDE) que ofrece una interfaz de usuario con las herramientas necesarias para desarrollar y depurar aplicaciones para los productos de TI.

Sus principales características son:

- Compiladores optimizados.
- Editor de código fuente.
- Ambiente para la construcción de un proyecto.
- Depurador completo para C/C++ y código ensamblador.
- Esta basada en Eclipse.
- Puede instalarse en los sistemas operativos Windows y Linux.

La comunicación entre la tarjeta de evaluación y CCS se realiza mediante el módulo emulador XDS100, el cual provee acceso JTAG al MCU desde un puerto USB en la PC. Dicha comunicación puede mantener acceso en tiempo real a la memoria y los registros de control, además de permitir la contabilización de ciclos de reloj entre dos segmentos de código, graficar segmentos de memoria, visualizar el *desensamble* del código, entre otras funciones. En la figura 4.1 se muestran las principales ventanas que conforman el programa.

1. **Navegador de proyectos:** organiza todos los archivos que pertenecen a un proyecto con extensiones .c, .asm, .cmd, .obj o cualquier otro.
2. **Panel de control de una sesión de depuración:** cuando se está en una sesión de depuración o *debug*, permite manipular la forma de ejecución del programa, ya sea pausando, reiniciando, ejecutando paso a paso, etcétera.
3. **Ventana de edición:** es la ventana donde se tiene acceso a los archivos del proyecto. En la figura 4.1 se muestra un programa en lenguaje C.
4. **Ventana de monitoreo:** sirve para observar y modificar las variables en memoria RAM.
5. **Ventana de registros:** permite monitorear, y en algunos casos modificar los registros del dispositivo, ya sean del CPU, interrupciones, módulo ADC, módulo PWM, entre otros.

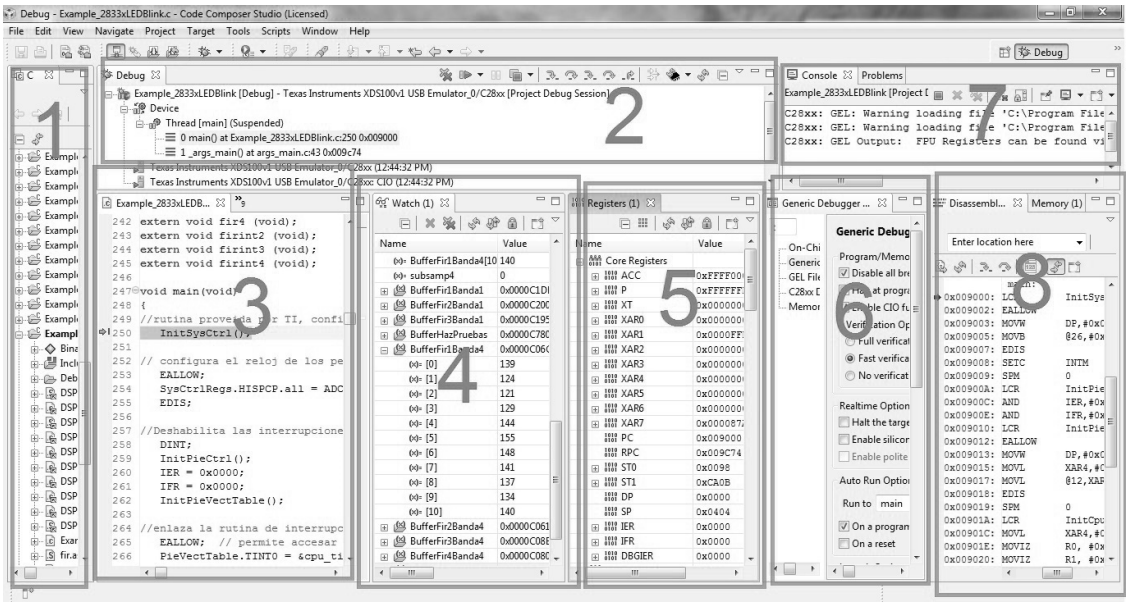

Figura 4.1: *Ventanas de CCS.*

6. **Panel de configuración de la sesión de depuración:** permite elegir el grado de interacción de la PC y el dispositivo en una sesión de depuración.
7. **Ventana de notificaciones:** muestra la actividad del CCS y notificaciones respecto a la compilación de código o la comunicación con la tarjeta.
8. **Ventana de código desensamblado:** muestra el código en ensamblador tal como se cargó en el dispositivo.

Dentro del entorno se puede modificar la cantidad de ventanas que se desean visualizar, así como su distribución en la pantalla, tanto para el modo *CCS Edit* como para el modo *CCS Debug*.

Por su parte, la barra de herramientas provee acceso a todas las funciones del ambiente.

- **File:** despliega opciones de creación de nuevos proyectos o archivos, así como abrir, cerrar, importar, exportar, visualizar propiedades y una lista de archivos recientes.
- **Edit:** provee funciones como copiar, cortar, pegar, deshacer, rehacer, seleccionar, así como herramientas de búsqueda.
- **View:** despliega una lista de las ventanas disponibles para mostrar en la interfaz.
- **Navigate:** provee funciones para la navegación en el código.
- **Project:** contiene funciones de creación, construcción e importación de proyectos, archivos y ejemplos.
- **Run:** despliega funciones de depuración que permiten la carga del proyecto a la tarjeta.

- **Scripts:** permite el acceso a *scripts* en caso de existir.
- **Window:** ofrece herramientas para la creación y edición de ventanas y perspectivas. También permite el acceso a las preferencias del programa.
- **Help:** provee acceso a las herramientas de ayuda, así como identificación e instalación de actualizaciones. Aquí se encuentra también la información de la versión del programa bajo la etiqueta *About CCS*.

4.1. Creación de un proyecto en CCS

Al ejecutar el software CCS lo primero que nos solicita es seleccionar un directorio de trabajo, como se muestra en la figura 4.2. Una vez seleccionado se puede evitar que vuelva a aparecer esta ventana seleccionando la casilla de *usar este directorio por defecto y no preguntar nuevamente*.

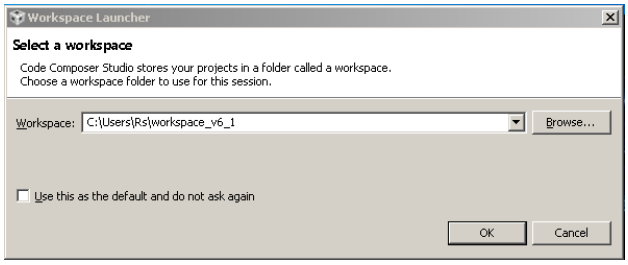

Figura 4.2: *Selección del directorio de trabajo.*

Al iniciar CCS por primera vez se observa una pantalla como la que se muestra en la figura 4.3, donde se muestran botones para crear un nuevo proyecto, buscar ejemplos instalados, importar proyectos, etc.

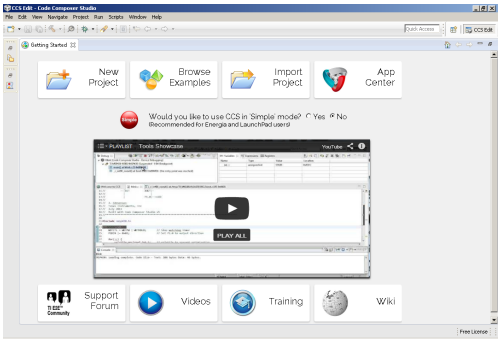

Figura 4.3: *Pantalla de inicio*

Seleccionando el icono *New Project* (o bien File→New→CSS Project) aparece una ventana como la mostrada en la figura 4.4. Para crear un proyecto que utilice la tarjeta LaunchPad Delfino, es necesario que quede configurada como se muestra en dicha figura.

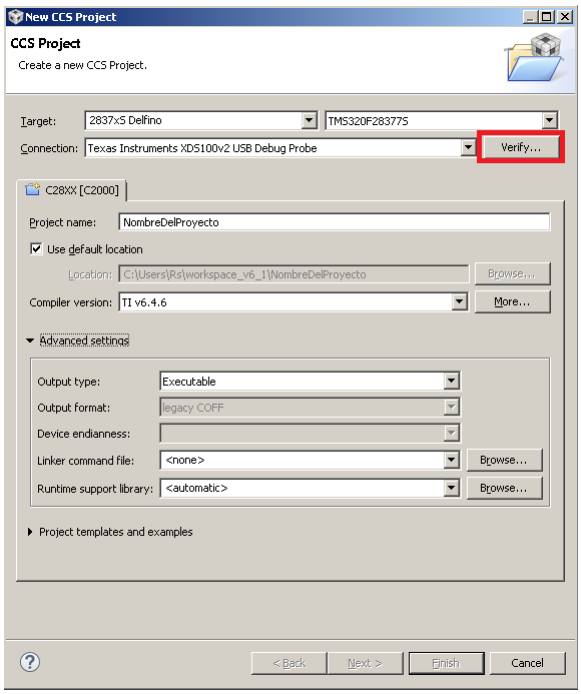

Figura 4.4: *Nuevo proyecto con la tarjeta LaunchPad Delfino*

El botón de *Verify...* que se encuentra a la derecha del campo *Connection* genera una prueba que verifica la correcta comunicación entre CCS y la tarjeta. Al conectar la tarjeta con el cable USB proporcionado en el Kit de evaluación y presionar el botón de *Verify...*, aparecerá una ventana emergente. Si la prueba fue exitosa, la ventana mostrará hasta el final del reporte un mensaje como el que se muestra en la figura 4.5.

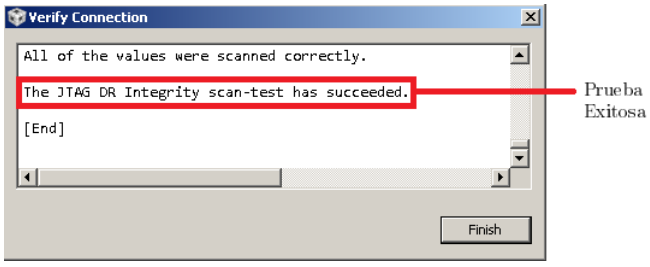

Figura 4.5: *Prueba de conexión exitosa*

En caso de que la conexión falle, será necesario revisar el cable de conexión hacia la computadora, así como la correcta selección del emulador XDS100v2 en el campo de *Connection* de la figura 4.4.

Por último, en la sección de *Project templates and examples* se selecciona el tipo de archivo que se usará en el proyecto. Los ejemplos de este tutorial se encuentran escritos en lenguaje ensamblador, por lo que se selecciona *Empty Assembly-only Project*, como se muestra en la figura 4.6, y se da click en el botón *finish*.

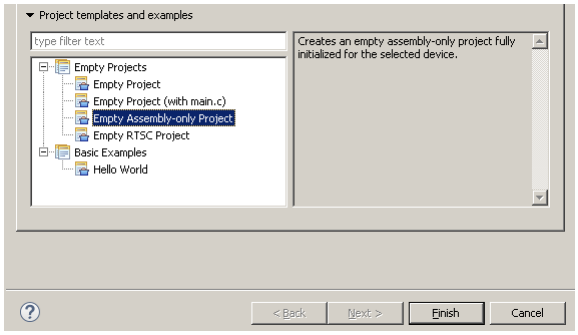

Figura 4.6: *Selección de proyecto para lenguaje ensamblador*

Una vez creado el proyecto, la interfaz cambiará su apariencia y se verá como se muestra en la figura 4.7. Aparecerán dos ventanas: el explorador de proyectos a la izquierda y la ventana de notificaciones en la parte inferior.

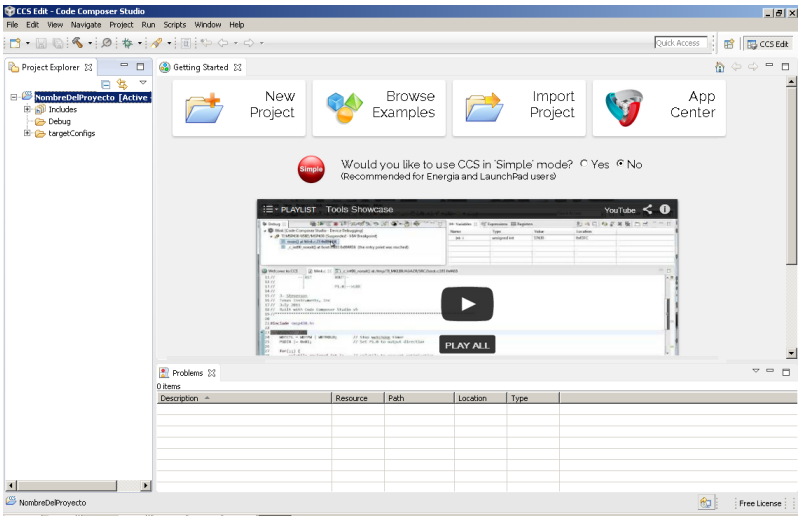

Figura 4.7: *Interfaz con un proyecto*

Debido a que se seleccionó *<none>* en el campo *Linker command file* (figura 4.4), el proyecto carecerá de un archivo .cmd, por lo que será necesario agregar dicho archivo en el proyecto, el cual contiene la definición de los bloques de memoria, como son *.text* y *.data*. Para cargar este archivo, haga clic derecho en la carpeta del proyecto, ubicada en el explorador de proyectos, y seleccione *Add Files...*, aparecerá una ventana donde se deberá buscar el archivo *mi_28377s.cmd*. El código de dicho archivo se encuentra en el Apéndice 1.

Para crear el archivo donde se escribirá el código, haga clic derecho sobre la carpeta del proyecto y seleccione *new* → *file*. Hecho lo anterior, emergerá una ventana como la que se muestra en la figura 4.8. En *file name* se escribe el nombre del archivo y se termina con *.asm*.

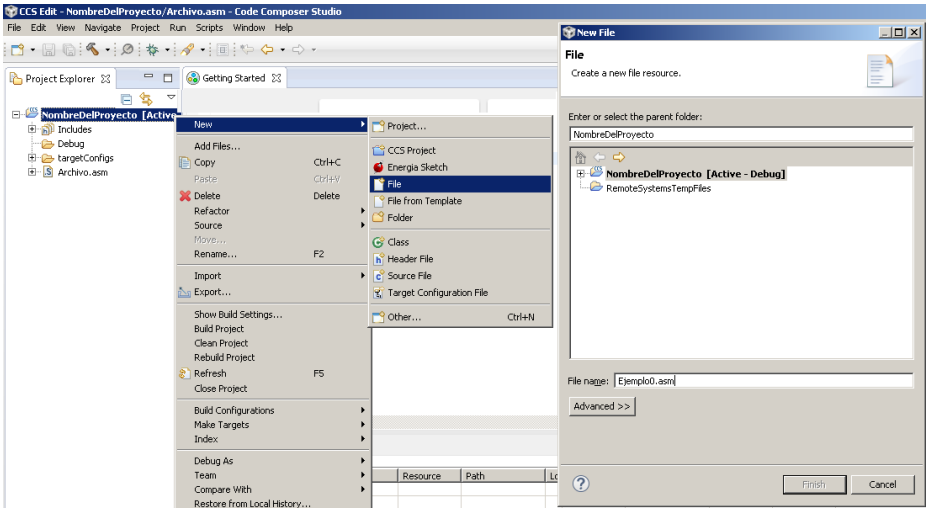

Figura 4.8: *Creación de un nuevo archivo en ensamblador*

Una vez creado el archivo .asm, copie el contenido del ejemplo de la sección 5.1.1 para continuar con esta prueba. Posteriormente será necesario construir el proyecto, para ello, se hace clic en el icono de *Build*, el cual se muestra en la figura 4.9. Si existen errores en el código, estos se mostrarán en la ventana de notificaciones. Al realizar la corrección, será necesario construir el proyecto nuevamente.

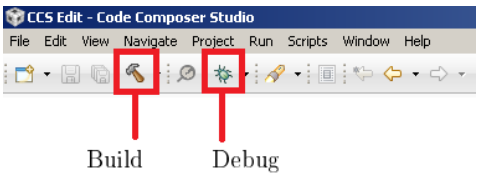

Figura 4.9: *Herramienta 'Build' y 'Debug'*

Una vez que se obtiene la compilación exitosa del proyecto, mediante el icono de *Debug* (figura 4.9) se carga el programa a la tarjeta. Al presionar el botón, aparecerá una ventana emergente que notifica al usuario que se está cargando el proyecto e inmediatamente cambia la interfaz de *CCS Edit* a *CCS Debug* como se muestra en la figura 4.10.

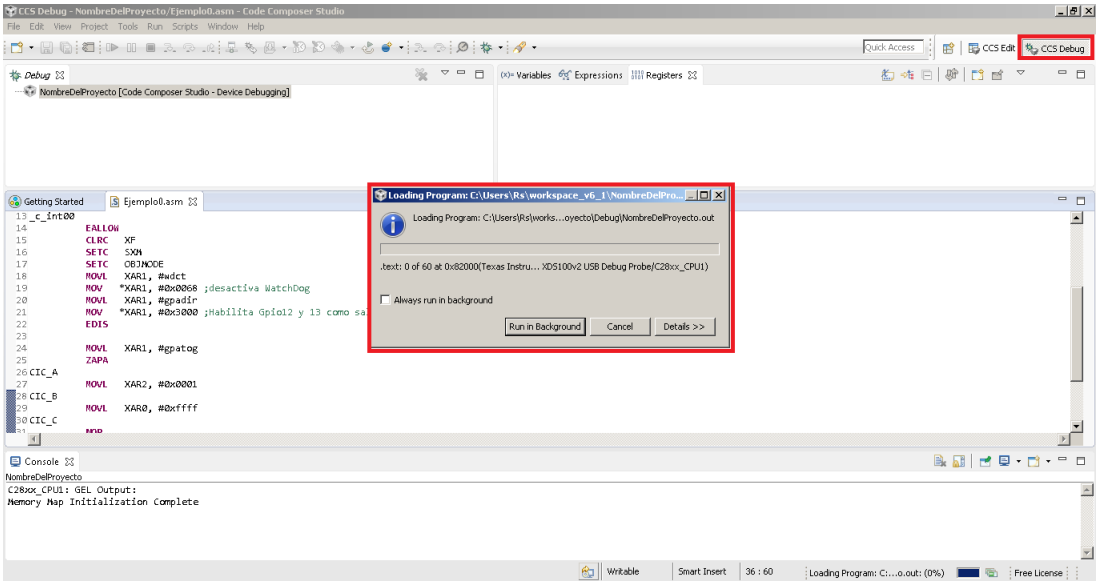

Figura 4.10: *Interfaz CCS Debug al cargar el programa*

Al terminar la carga del programa, éste puede ejecutarse instrucción por instrucción mediante el icono *Step Into* (figura 4.11), o bien puede ejecutarse de forma continua mediante el icono *Resume*.

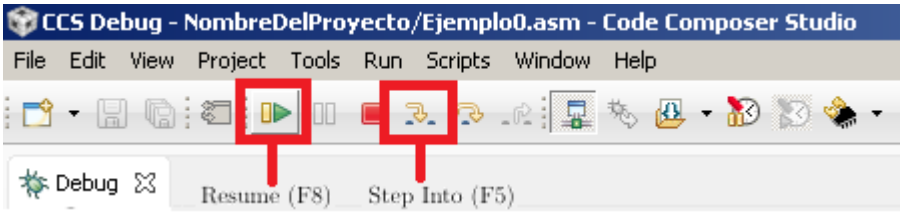

Figura 4.11: *Herramienta 'Resume' para ejecución de código*

5. Ejemplos en lenguaje ensamblador

El propósito de estos ejemplos es introducir al lector en la realización e implementación de programas en lenguaje ensamblador utilizando los diferentes modos de direccionamiento del DSP. En todos los ejemplos se seguirá el procedimiento mostrado en la sección 4.1.

5.1. Suma varias constantes en aritmética de punto fijo a 16 Bits

En estos ejemplos se realiza la suma de cinco constantes y el resultado se salva en modo directo. La operación a realizar es la sumatoria que se muestra en la ecuación (5.1).

$$total = \sum_{i=0}^{N-1} D_i \tag{5.1}$$

5.1.1. Direccionamiento Inmediato

En este modo de direccionamiento las constantes se codifican en la instrucción, por lo que no es posible cambiar sus valores durante la ejecución del programa, siendo este tipo de direccionamiento poco flexible. En este primer ejemplo se puede observar que existe un bloque de desactivación del *WatchDog* para evitar que se interrumpa al CPU (en los ejemplos subsecuentes se prescindirá de este bloque).

```
;*
;*    Suma varias constantes en modo inmediato
;*
        .global   _c_int00 ; Símbolo global para inicio de código
        .data              ; Sección de datos
WDCR    .set   07029h      ; Dirección registro de control WatchDog
CTE_WD  .set   0068h       ; Constante para desactivar el  WatchDog
D1      .set  1            ; Se define  D1 = constante = 1 entero
D2      .set  2            ; Se define  D2 = constante = 2
D3      .set  3            ; Se define  D3 = constante = 3
D4      .set  4            ; Se define  D4 = constante = 4
D5      .set  5            ; Se define  D5 = constante = 5
```

```
total   .word 0            ; Aparta una localidad para variable total
                           ; y la inicializa con 0

;* SECCION DE CODIGO
        .text              ; Sección de código
_c_int00                   ; Inicio de código

;* Deshabilitación del WatchDog
         EALLOW            ; Habilita escritura a registros protegidos
         MOVL XAR1, #WDCR  ; Registro XAR1 apunta dir, WDCR
         MOV  *XAR1,#0068h ; Desactiva WatchDog, escribe en WDCR
         EDIS              ; Deshabilita escritura a registros protegidos
;*
         MOVW  DP, #total  ; Apuntador de página de datos en página de
                           ; variable total
         MOV   ACC, #D1    ; Mueve D1 a acumulador
         ADD   ACC, #D2    ; Suma D2 a acumulador
         ADD   ACC, #D3    ; Suma D3 a acumulador
         ADD   ACC, #D4    ; Suma D4 a acumulador
         ADD   ACC, #D5    ; Suma D5 a acumulador
         MOV   @total, AL  ; Salva la suma AL en localidad total

REGRESA  NOP               ; Ciclo infinito para fin de programa
         LB    REGRESA
        .end               ; Fin de ensamblado
```

5.1.2. Direccionamiento Directo

El direccionamiento directo escribe los valores de las variables en la memoria de datos, lo que permite modificar su valor durante la ejecución del programa, lo que hace a este modo más flexible que el modo inmediato. Para hacer uso de este tipo de direccionamiento es necesario especificar la dirección de la página donde se encuentran los datos (DP).

```
;*
;*    Suma varios datos en modo directo
;*
        .global   _c_int00 ; Símbolo global para inicio de código
        .data              ; Sección de datos
x1      .word 1            ; Aparta una localidad para variable x1
                           ; y la inicializa con el valor 1
x2      .word 2            ; Aparta una localidad para variable x1
                           ; y la inicializa con el valor 2
```

```
x3      .word 3
x4      .word 4
x5      .word 5
total   .word 0            ; Aparta una localidad para variable total
                           ; y la inicializa con 0
        .text              ; Sección de código

_c_int00
         MOVW  DP, #total  ; Apuntador de página de datos en página de total
         MOV   ACC, @x1    ; Mueve dato x1 a acumulador
         ADD   ACC, @x2    ; Suma x2 a acumulador
         ADD   ACC, @x3    ; Suma x3 a acumulador
         ADD   ACC, @x4    ; Suma x4 a acumulador
         ADD   ACC, @x5    ; Suma x5 a acumulador
         MOV   @total, AL  ; Salva la suma AL en localidad total

REGRESA  NOP               ; Ciclo infinito para fin de programa
         LB    REGRESA
         .end              ; Fin de ensamblado
```

5.1.3. Direccionamiento Indirecto

El direccionamiento indirecto emplea los registros auxiliares XAR0 a XAR7 como apuntadores para direccionar la memoria de datos, por lo que las variables se pueden escribir como componentes de un vector. Este modo permite realizar modificaciones a los registros apuntadores (XARn) en paralelo con la ejecución de la instrucción, es decir, se puede post-incrementar y post/pre-decrementar el contenido de los registros auxiliares mientras se ejecuta cualquier otra instrucción.

```
;*
;*    Suma varios datos en modo indirecto
;*
        .global _c_int00
        .data
N       .set    5
D       .word   1,2,3,4,5 ; Aparta cinco localidades para el vector
                          ; de datos D y le escribe los valores 1,2,3,4,5
total   .word   0
        .text

_c_int00
         MOVL XAR1,#D      ; Registro XAR1 apunta al arreglo D
```

```
        MOV  ACC, *XAR1++ ; Carga ACC con el dato apuntado por XAR1
                          ; Postincrementa XAR1: XAR1 = XAR1 + 1
        ADD ACC,*XAR1++   ; ACC = ACC + dato apuntado por XAR1
                          ; XAR1 = XAR1 + 1
        ADD ACC,*XAR1++   ; ACC = ACC + dato apuntado por XAR1
                          ; XAR1 = XAR1 + 1
        ADD ACC,*XAR1++   ; ACC = ACC + dato apuntado por XAR1
                          ; XAR1 = XAR1 + 1
        ADD ACC,*XAR1++   ; ACC = ACC + dato apuntado por XAR1
                          ; XAR1 = XAR1 + 1, apunta a loc. total
        MOV *XAR1,AL      ; Guarda la parte baja del acumulador en total

REGRE   NOP
        LB    REGRE
        .end
```

De este ejemplo, se puede observar que, al incrementar el apuntador XAR1 durante las sumas, este queda situado una posición después del último valor del arreglo D. Dicha localidad es la asignada a la variable total, por lo que para guardar el resultado de la sumatoria solo es necesario mover el acumulador a la dirección apuntada por XAR1.

5.1.4. Uso de la instrucción RPT

De los ejemplos anteriores, hemos visto que para sumar N datos se requiere $N-1$ instrucciones suma. Si N es muy grande, resulta impráctico declarar una instrucción por cada operación. Una forma de evitar esto es a través de un ciclo que repita la instrucción suma. Esto se logra empleando el modo de direccionamiento indirecto del programa anterior y utilizando la instrucción de repetición RPT.

```
;*
;*    Suma varios datos en modo indirecto con instrucción RPT
;*
        .global _c_int00
        .data
N       .set    5
D       .word   1,2,3,4,5 ; Aparta cinco localidades para el vector de
                          ; datos D y le escribe los valores  1,2,3,4,5
total   .word   0
        .text

_c_int00
        MOVL XAR1,#D      ; Registro XAR1 apunta al arreglo D
```

```
          MOV  ACC, #0      ; Se limpia acumulador

          RPT #N-1          ; Repite N veces la siguiente instrucción
       || ADD ACC,*XAR1++   ; ACC = ACC + dato apuntado por XAR1
                            ; XAR1 = XAR1 + 1

         MOVW DP,#total     ; Ubica al DP donde se encuentra la página total
         MOV @total,AL      ; Guarda la parte baja del acumulador en total

REGRESA NOP
         LB   REGRESA
         .end
```

5.2. Promedio de una secuencia de datos

Para obtener el promedio de un conjunto o una secuencia de datos es necesario efectuar la suma de los datos y dividirla entre la cantidad de datos. Matemáticamente se muestra en la ecuación (5.2).

$$x_{med} = \frac{1}{N} \sum_{i=0}^{N-1} x_i \tag{5.2}$$

En los ejemplos anteriores hemos realizado la suma de datos, por lo que sería suficiente multiplicar la suma por el inverso de N, es decir, agregar el siguiente código al final del programa:

```
      MOV   T,@total    ; Carga en T la suma total
      MPY   ACC,T,@Ninv ; ACC = total*Ninv
```

Al final, se salva el resultado en el acumulador y es necesario ajustarlo al formato adecuado realizando los corrimientos necesarios.

5.3. Manejo de los LEDs D9 y D10 como salidas

Los LEDs D9 y D10 de la tarjeta se encuentran conectados a los terminales de los GPIOs 12 y 13, respectivamente. El siguiente código de ejemplo hace parpadear el LED D10 a mitad de la frecuencia que lo hace el LED D9.

```
          .global _c_int00
          .data
wdct      .set 0x7029 ; Dirección del registro WatchDog
```

```
gpadir    .set 0x7c0a ; Dirección del registro GpaDir
gpatog    .set 0x7f06 ; Dirección del registro GpaTog


          .text
_c_int00
          EALLOW               ; Habilita escritura a registros protegidos
          MOVL   XAR1, #wdct   ; Registro xar1 apunta a wdct
          MOV   *XAR1, #0x0068 ; Desactiva WatchDog
          MOVL   XAR1, #gpadir ; Registro xar1 apunta a gpadir
          MOV   *XAR1, #0x3000 ; Habilita Gpio12 y 13 como salidas
          EDIS                 ; Deshabilita escritura a registros protegidos

          MOVL   XAR1, #gpatog ; Registro xar1 apunta a gpatog
CIC_A
          MOVL   XAR2, #0x0001 ; Carga registro xar2 con 0x0001
CIC_B
          MOVL   XAR0, #0xffff ; Carga registro xar0 con 0xffff
CIC_C
          NOP                  ; No operación
          NOP                  ; No operación
          BANZ CIC_C, AR0--    ; Salta a cic_c si ar0 no es igual a cero
                               ; Decrementa registro AR0
          MOV   *XAR1, #0x1000 ; Cambia el estado del Gpio12
          BANZ CIC_B, AR2--    ; Salta a cic_b si ar2 no es igual a cero
                               ; Decrementa registro AR2
          MOV   *XAR1, #0x2000 ; Cambia el estado del Gpio13
          LB CIC_A             ; Salta a cic_a
          .end                 ; Fin de ensamblado
```

6. Resumen

Este tutorial conforma una breve guía introductoria para los interesados en el PDS. Para ello, se presentó una descripción de la tarjeta de evaluación LaunchPad Delfino, de la compañía Texas Instruments, que resulta un excelente medio para comenzar de inmediato a desarrollar e implementar códigos relacionados con el PDS. Posteriormente se presentó una introducción al Software Code Composer Studio y la metodología a seguir para la creación de un proyecto. Finalmente, se mostraron algunos códigos de ejemplo relacionados con aritmética de punto fijo.

Bibliografía

[1] Escobar S. L. *Arquitecturas de DSP TMS320F28xxx y aplicaciones*. Facultad de Ingeniería, UNAM, marzo de 2014. 282 pags.

[2] IEEE *IEEE Standar for Binary Floating-Point Arithmetic*. IEEE Standar 754-1985.

[3] Texas Instruments. *TMS320F28x CPU and Instruction Set Reference Guide*. SPRU430. USA 2015.

[4] Texas Instruments. *TMS320F28x Assembly Language Tools v.15.9.0.STS. User's Guide*.SPRU513. USA 2015.

[5] Texas Instruments. *TMS320F2837xS Delfino Microcontrolles. Technical Reference Manual*. SPRUHX5. USA 2015.

[6] Texas Instruments. *TMS320F2837xS Delfino Microcontrolles. Data Sheet*. SPRS881. USA 2015.

[7] Texas Instruments. *TMS320C28x Extended Instruction Sets Technical Reference Manual*. SPRUHS1. USA 2014.

[8] Texas Instruments. *TMS320F28377S Launchpad Quick Start Guide*. SPRUI26. USA 2015.

Apéndice 1

A continuación se presenta el código del archivo mi_28377S.cmd.

```
MEMORY
{
PAGE 0 :  /* Program Memory */
          /* Memory (RAM/FLASH) blocks can be moved to PAGE1 for data allocation */
          /* BEGIN is used for the "boot to Flash" bootloader mode   */

   BEGIN            : origin = 0x080000, length = 0x000002
   RAMM0            : origin = 0x000122, length = 0x0002DE
   RAMD0            : origin = 0x00B000, length = 0x000800
   RAMLS0           : origin = 0x008000, length = 0x000800
   RAMLS1           : origin = 0x008800, length = 0x000800
   RAMLS2           : origin = 0x009000, length = 0x000800
   RAMLS3           : origin = 0x009800, length = 0x000800
   RAMLS4           : origin = 0x00A000, length = 0x000800
   RAMGS14          : origin = 0x01A000, length = 0x001000
   RAMGS15          : origin = 0x01B000, length = 0x001000
   RESET           : origin = 0x3FFFC0, length = 0x000002

   /* Flash sectors */
   FLASHA           : origin = 0x080002, length = 0x001FFE /* on-chip Flash */
   FLASHB           : origin = 0x082000, length = 0x002000 /* on-chip Flash */
   FLASHC           : origin = 0x084000, length = 0x002000 /* on-chip Flash */
   FLASHD           : origin = 0x086000, length = 0x002000 /* on-chip Flash */
   FLASHE           : origin = 0x088000, length = 0x008000 /* on-chip Flash */
   FLASHF           : origin = 0x090000, length = 0x008000 /* on-chip Flash */
   FLASHG           : origin = 0x098000, length = 0x008000 /* on-chip Flash */
   FLASHH           : origin = 0x0A0000, length = 0x008000 /* on-chip Flash */
   FLASHI           : origin = 0x0A8000, length = 0x008000 /* on-chip Flash */
   FLASHJ           : origin = 0x0B0000, length = 0x008000 /* on-chip Flash */
   FLASHK           : origin = 0x0B8000, length = 0x002000 /* on-chip Flash */
   FLASHL           : origin = 0x0BA000, length = 0x002000 /* on-chip Flash */
```

```
   FLASHM           : origin = 0x0BC000, length = 0x002000 /* on-chip Flash */
   FLASHN           : origin = 0x0BE000, length = 0x002000 /* on-chip Flash */
   FLASHO           : origin = 0x0C0000, length = 0x002000 /* on-chip Flash */
   FLASHP           : origin = 0x0C2000, length = 0x002000 /* on-chip Flash */
   FLASHQ           : origin = 0x0C4000, length = 0x002000 /* on-chip Flash */
   FLASHR           : origin = 0x0C6000, length = 0x002000 /* on-chip Flash */
   FLASHS           : origin = 0x0C8000, length = 0x008000 /* on-chip Flash */
   FLASHT           : origin = 0x0D0000, length = 0x008000 /* on-chip Flash */
   FLASHU           : origin = 0x0D8000, length = 0x008000 /* on-chip Flash */
   FLASHV           : origin = 0x0E0000, length = 0x008000 /* on-chip Flash */
   FLASHW           : origin = 0x0E8000, length = 0x008000 /* on-chip Flash */
   FLASHX           : origin = 0x0F0000, length = 0x008000 /* on-chip Flash */
   FLASHY           : origin = 0x0F8000, length = 0x002000 /* on-chip Flash */
   FLASHZ           : origin = 0x0FA000, length = 0x002000 /* on-chip Flash */
   FLASHAA          : origin = 0x0FC000, length = 0x002000 /* on-chip Flash */
   FLASHAB          : origin = 0x0FE000, length = 0x002000 /* on-chip Flash */

PAGE 1 : /* Data Memory */
         /* Memory (RAM/FLASH) blocks can be moved to PAGE0 for program allocation */

   BOOT_RSVD       : origin = 0x000002, length = 0x000120 /* Part of M0, BOOT rom
                                                             will use this for stack */
   RAMM1           : origin = 0x000400, length = 0x000400 /* on-chip RAM block M1 */
   RAMD1           : origin = 0x00B800, length = 0x000800

   RAMLS5      : origin = 0x00A800, length = 0x000800

   RAMGS0      : origin = 0x00C000, length = 0x001000
   RAMGS1      : origin = 0x00D000, length = 0x001000
   RAMGS2      : origin = 0x00E000, length = 0x001000
   RAMGS3      : origin = 0x00F000, length = 0x001000
   RAMGS4      : origin = 0x010000, length = 0x001000
   RAMGS5      : origin = 0x011000, length = 0x001000
   RAMGS6      : origin = 0x012000, length = 0x001000
   RAMGS7      : origin = 0x013000, length = 0x001000
   RAMGS8      : origin = 0x014000, length = 0x001000
   RAMGS9      : origin = 0x015000, length = 0x001000
   RAMGS10     : origin = 0x016000, length = 0x001000
   RAMGS11     : origin = 0x017000, length = 0x001000
   RAMGS12     : origin = 0x018000, length = 0x001000
   RAMGS13     : origin = 0x019000, length = 0x001000

}
```

```
SECTIONS
{
   /* Allocate program areas: */
   .cinit           : > FLASHB     PAGE = 0, ALIGN(4)
   .pinit           : > FLASHB,    PAGE = 0, ALIGN(4)
   .text            : >> FLASHB | FLASHC | FLASHD | FLASHE   PAGE = 0, ALIGN(4)
   codestart        : > BEGIN      PAGE = 0, ALIGN(4)
   ramfuncs         : LOAD = FLASHD,
                      RUN = RAMLS0 | RAMLS1 | RAMLS2 |RAMLS3,
                      LOAD_START(_RamfuncsLoadStart),
                      LOAD_SIZE(_RamfuncsLoadSize),
                      LOAD_END(_RamfuncsLoadEnd),
                      RUN_START(_RamfuncsRunStart),
                      RUN_SIZE(_RamfuncsRunSize),
                      RUN_END(_RamfuncsRunEnd),
                      PAGE = 0, ALIGN(4)

   /* Allocate uninitalized data sections: */
   .stack              : > RAMM1        PAGE = 1
   .ebss               : >> RAMLS5 | RAMGS0 | RAMGS1       PAGE = 1
   .data               : >> RAMLS0                         PAGE = 0
   .esysmem            : > RAMLS5       PAGE = 1

   /* Initalized sections go in Flash */
   .econst             : >> FLASHF | FLASHG | FLASHH      PAGE = 0, ALIGN(4)
   .switch             : > FLASHB      PAGE = 0, ALIGN(4)

   .reset              : > RESET,     PAGE = 0, TYPE = DSECT /* not used, */

}

/*
//===========================================================================
// End of file.
//===========================================================================
*/
```