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

Ín	dice general	I						
1.	Introducción 1							
2.	MCU Delfino TMS320F28377S							
3.	Tarjeta LaunchPad Delfino	5						
4.	Code Composer Studio 4.1. Creación de un proyecto en CCS	8 10						
5.	 Ejemplos en lenguaje ensamblador 5.1. Suma varias constantes en aritmética de punto fijo a 16 Bits	 15 15 16 17 18 19 19 						
6.	Resumen	21						
Bi	bliografía	22						
Aŗ	péndice 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.

Debug - Example	e_2833xLEDBlink.c - Code Composer Studio (Licensed)	and the second second	100 C 100 C 100 C 100	1-	_ D X
File Edit View	Navigate Project Target Tools Scripts Window He	lp			
	□ ∞□∞ ☆ • 9. • 19 A 9	• □ • ♥ Φ • ↔ •		-	🖽 🏇 Debug
te c 🛛 🗖 🗆	Debug 🛛	🍇 🕪 🕶 🖩 🖬 💌 🔍 🗇) ¬ ⊂ ⊟ 🗞 + 🏈 ⊟ [∨] ¬ □	Console 🛛 Problems	- D
4	Example_2833xLEDBlink [Debug] - Texas Instruments	XDS100v1 USB Emulator_0/C28xx [Project Debug	Session	 Example_2833xLEDBlink [Project I 	
2 - 2 - 2	⊜-@® Device			C28xx: GEL: Warning lo	ading file 'C:\Program File ^
	Inread [main] (Suspended) Optical and a second se	0.00000		E C28xx: GEL: Warning lo	ading file 'C:\Program File
- 4°	= 1 arcs main() at arcs main c43 0x009c74	0009000		C28xx: GEL Output: FP	U Registers can be found vi
Example 4	Texas Instruments XDS100v1 USB Emulator 0/C2	3xx (12:44:32 PM)			
Example	Texas Instruments XDS100v1 USB Emulator_0/02	8xx: CIO (12:44:32 PM)	1		P
Example	E Evample 2823vl EDR S? "	1 (60 Watch (1) 22	1919 Requirters (1) 22	Generic Debugger 😚 🗖 🗖	Dirarramhi 😚 Mamonr (1) 🗖 🗖
Example	242 antenn mid fint (mid)				
🕀 🥵 Example	243 extern void firint2 (void):		E \$^ \$5 @ [] .	Generic Debug	· · · · · · · · · · · · · · · · · · ·
🗄 😅 Example	244 extern void firint3 (void);	Name Value ^	Name Value 📤	On-Chi	Enter location here 🔹
🔅 😅 Example	245 extern void firint4 (void);	(x): BufferFir1Banda4[10 140	E 🎆 Core Registers	- Generic Program/Memo	A A A A A A A
Example	246	(x)+ subsamp4 0		GEL File Disable all bre	
🕀 🦳 Example	247 [©] void main(void)	⊞ BufferFir1Banda1	I III P OxFFFFFF		DOX009000: LC ThitSvs
🔅 🗁 Example	248 {	⊞ BufferFir2Banda1		Memor Memor De CIO fu =	0x009002: EALLow
🗈 📴 Example	249 //rutina proveida p r TI, confi	BufferFir3Banda1 0x0000C195		Verif ation Op	0x009003: MOVW DP,#0x0
😑 🧁 Exampl	<pre>pl250 InitSysCtrl();</pre>	BufferHazPruebas 0x0000C780		Trull verificat	0х009005: МОVВ 026,#0х
⊕- ♦ Binz	251	BufferFir1Banda4 0x0000C06C	⊞ ### XAR2 0x000000	Eart varifica	0x009007: EDIS
B-@ Incl	252 // configura el reloj de los pe	(4)= [0] 139	H H KAR3 0x000000	O Past vernica	0x009008: SETC INTM
🗄 🗁 Deb	253 EALLOW; 254 SwaCtriBogg HISDCD all = ADC	(x)= [1] 124		O No verificat	Ox009003; ICP TritPic
⊕-B DSP	255 FDIS.	(x)= [2] 121		D. 10. D. 1	0x00900C: AND TER.#0x
⊞- Le DSP	256	(x)= [3] 129		Realtime Option	0x00900E: AND IFR. #0x
⊞-LS DSP	257 //Deshabilita las interrupcione	(x)= [4] 144	⊞ 1000 XAR7 0x000087;	🖾 Halt the targe	0x009010: LCR InitPie
⊞Le DSP	258 DINT:	(x)= [5] 155	1010 PC 0x009000	Enable silicor	0x009012: EALLOW
⊞- Le DSP	259 InitPieCtrl();	(x)= [6] 148	1010 RPC 0x009C74	Enable polite	0x009013: MOVW DP,#0x0
⊕- 👰 DSP	260 IER = 0x0000;	(×)= [7] 141			0x009015: MOVL XAR4,#C
⊞-⊔e DSP	261 IFR = 0x0000;	(x)= [8] 137	Hill ST1 0xCA0B	Auto Run Optio	0x009017: MOVL @12,XAF
⊞Le DSP	<pre>262 InitPieVectTable();</pre>	(x)= [9] 134	1010 DP 0x0000	Run to main	OXODODIS: EDIS
🕀 🖳 😥 DSP	263	(×)= [10] 140	1010 SP 0x0404	and a second second	Ov000013: JCD InitCon
🗈 🖓 DSP	264 //enlaza la rutina de interrupc	🗉 🙆 BufferFir2Banda4 0x0000C061		🗹 On a program	0x00901C: MOVL XAR4.#C
⊕-⊡ Exar	265 EALLOW; // permite accesar	⊞ 🕮 BufferFir3Banda4 0x0000C08E	⊞ ### IFR 0x0000	🔄 On a reset	0x00901E: MOVIZ R0, #0x
⊞-S fir.a .	<pre>266 PieVectTable.TINT0 = &cpu_ti.</pre>	🗉 🙆 BufferFir4Banda4 0x0000C08C 🚽	B ISIN DBGIER 0x0000 -		0x009020: MOVIZ R1, #0x -
< >	< >>	۲	• III •		< <u> </u>
D°					

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*.

🐨 Workspace Launcher	×
Select a workspace	
Code Composer Studio stores your projects in a folder called a workspace. Choose a workspace folder to use for this session.	
Workspace: C:\Users\Rs\workspace_v6_1	▼Browse
Use this as the default and do not ask again	
	OK Cancel

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.

• () () (§ • (Ø)	☆・ ∥ • ⊕ ⊕						Quick Access
🚳 Getting Started 💠							h 🗢 🕁
*	New Project	Browse Examples		Import Project	V	App Center	
	_	Would you like to u Recommended for Ene	se CCS in 'Sim Igla and Launchi	iple' mode? - C ' Padusers?	Yes ®No		
	DIPPLATEDT TO	ola Showcaso Showcaso Showcaso Showcaso	Product Witeman B Sec. Sec. R.1 arguint	Bagans By ris Nata Codes Nata Natari	YouTube 🔨	0	
	A state of the second secon	Control for column an include					
	0.000000000000000000000000000000000000		Lanes.			11 D	
	Manuscription Annual Statement Annual						
			Play All		1 3 40 H O - P		
	a construction of the second s	Fig. (ed. suid a suid a suid fuel de la classificación de la classifi	PLAY ALL		13,0104.047		

Figura 4.3: Pantalla de inicio

Seleccionando el icono New Project (o bien File \rightarrow New \rightarrow 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.

💱 New CCS Pi	roject					
CCS Project Create a new	CCS Project.					
Target:	2837x5 Delfin	D	•	TM5320F283775		•
Connection:	Texas Instrum	ents XD5100v2 US	8 Debug Probe		-	Verify
😭 C28XX [C2000]					
Project nam	ne: Nombre	eDelProyecto				
🔽 Use def	ault location					
Loc	ation: C:\Use	rs\Rs\workspace_	/6_1\NombreDell	Proyecto		Browse
Compiler ve	rsion: TI v6.4	.6			•	More
Advanced Output ty	d settinas pe:	Executable			•	
Output fo	rmat:	legacy COFF			Ŧ	
Device en	dianness:				Ψ.	
Linker con	nmand file:	<none></none>			•	Browse
Runtime s	upport library:	<automatic></automatic>			•	Browse
Project te	emplates and ex	camples				
?			< <u>B</u> ack	Next >	Einish	Cancel

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.

Ę	Verify Connection	×	
	All of the values were scanned correctly.		
	The JTAG DR Integrity scan-test has succeeded.		- Prueba
	[End]	T	Exitosa
		F	
		Finish	

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.

	an peripes without help								
📑 • 🔛 🕼 🔦 • 🖉 🎋 •	🛷 • 📵 🐤 🔶 •	\$ •					Quick	Access	🗒 CCS Edit
🏠 Project Explorer 🔀 📃 🗖	🚳 Getting Started 🖂							🟠 🗇	⇔ = □
MombreDelProyecto [Active MombreDelProyecto [Active Debug Debug Debug Debug	2	New Project	🍄 e	Browse Examples		Import Project	V	App tenter	-
		Simp	Would (Recom	l you like to u mended for Ene	se CCS in 'S rgia and Laur	Simple' mode? O` nchPad users)	″es €No		
		PLAYLIST Binksg : Binksg :	TOOLS SHOW Will a Will strength, a Tools I train theory PRO Committee And Instance of Society (1) of Soci	en 1920 1. Sel (al main de la company) 1. post ess multed)	Network 12 40 September 1997	aine El Japano () ofi (n Vala Gradion gued et 1948 ()			
		Andrew (n)(3) Andrew	Held II (20.4.1490, peed) at	novit, water of a second state					-
		For (a) (PLAT ALL		11		
									VOF
	Problems 🔀								_
	Problems 23 0 items Description ~		Resource	Path	Location	Туре			
	Problems 🔀 0 items Description A		Resource	Path	Location	Туре			
	Problems X 0 items Description A		Resource	Path	Location	Туре			
	Problems Otems Description		Resource	Path	Location	Туре			
	Problems O items Description		Resource	Path	Location	Туре			

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 $mi_228377s.cmd$. 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 $new \rightarrow file$. 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*.

🜍 CCS Edit - NombreDelProyecto/A	rchivo.asm - Code Compose	Studio				
File Edit View Navigate Project F	Run Scripts Window Help			😚 New File		
📑 • 🔚 🕼 🔦 • 🖉 🎋 •	<i>∦</i> • ■ *	*		File		
陷 Project Explorer 🛛 📃 🗖	Getting Started 🔀			Create a new file resource.		
BombrethelProyecto [Activ Big] Includes Big] Includes Big] Encludes Big] Encludes Big] Encludes Big] Activo.asm	New Add Files Copy Paste Stopper Paste Refactor Source Rome Rename Import Export	Project Cri+C Cri+V Delete Pile Pile Pile Pile Pile Pile Pile Pil		Enter or select the parent folder: NombreDeProyecto		
	Show Build Settings Build Project Clean Project Rebuild Project SetTesh Close Project Build Configurations Male Targets Index Debug As Teom Compare With	F5	Ctrl+N Ctrl+N Resource Path	File name: Etemplo0.asm Advanced >>	Finish Cancel	

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.

CCS Debug - NombreDelProyecto/Ejemplo0.asr File Edit View Project Tools Run Scripts Wind	km - Code Composer Studio dow Hélp	
📑 • 🖫 🕼 💭 🕪 💷 🖷 3. 🖓 .t .	≅ % @ + 10 10 % + & ● + 12. ⊙ Ø ♦ + / +	Quick Access 🔡 📑 CCS Edit 🎭 CCS Debug
体 Debug 23 一 🎲 NombreDeProyecto [Code Composer Studio - De	evce Debugging]	∰ 44 ⊟ (\$P) [1] m ▼
🚳 Getting Started 🛛 🔝 Ejemplo0.asm 🔀	Coading Program: C:\Users\Rs\workspace_v6_1\NombreDelPro	
13	sactiva WatchDog xilita Gpiol2 y 13 como sa	ے پر پر
📮 Console 🔀		🗎 🛃 🛛 🛫 💭 🕶 🗖 🗖
NombrobelProyecto C2800_CPUI: GEL Output: Hemory Hap Initialization Complete		<u>م</u> ۲
	😥 Writable Smart Insert 36 : 60	Loading Program: C:o.out: (0%) 💼 🐑 Free License 📋

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

total	.word	0	;	Aparta una localidad para variable total y la inicializa con O
;* SECCI	ION DE	CODIGO		
	.text		;	Sección de código
_c_int00)		;	Inicio de código
;* Desha	abilita	ación del Wato	chl	Dog
	EALL	WC	;	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
;*
                   _c_int00 ; Símbolo global para inicio de código
        .global
        .data
                            ; Sección de datos
        .word 1
                            ; Aparta una localidad para variable x1
x1
                            ; y la inicializa con el valor 1
        .word 2
                            ; Aparta una localidad para variable x1
x2
                            ; y la inicializa con el valor 2
```

xЗ	.word	3	
x4	.word	4	
x5	.word	5	
total	.word	0	; Aparta una localidad para variable total ; y la inicializa con O
	.text		; Sección de código
_c_int0	0		
	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
Ν
        .set
                 5
D
                 1,2,3,4,5 ; Aparta cinco localidades para el vector
         .word
                           ; de datos D y le escribe los valores 1,2,3,4,5
total
                 0
        .word
        .text
_c_int00
         MOVL XAR1,#D
                            ; Registro XAR1 apunta al arreglo D
```

ACC, *XAR1++ ; Carga ACC con el dato apuntado por XAR1 MOV ; Postincrementa XAR1: XAR1 = XAR1 + 1 ADD ACC, *XAR1++ ; ACC = ACC + dato apuntado por XAR1 ; XAR1 = XAR1 + 1ADD ACC, *XAR1++ ; ACC = ACC + dato apuntado por XAR1 ; XAR1 = XAR1 + 1ADD ACC, *XAR1++ ; ACC = ACC + dato apuntado por XAR1 ; XAR1 = XAR1 + 1ADD 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
Ν
         .set
                 5
D
                 1,2,3,4,5 ; Aparta cinco localidades para el vector de
         .word
                            ; datos D y le escribe los valores 1,2,3,4,5
total
                 0
        .word
         .text
_c_int00
                           ; Registro XAR1 apunta al arreglo D
         MOVL XAR1,#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, #wd	ct ;	Registro xar1 apunta a wdct
	MOV *XAR1, #Ox(0068;	Desactiva WatchDog
	MOVL XAR1, #gpa	adir ;	Registro xar1 apunta a gpadir
	MOV *XAR1, #0x3	3000 ;	Habilita Gpio12 y 13 como salidas
	EDIS	;	Deshabilita escritura a registros protegidos
	MOVL XAR1, #gpa	atog ;	Registro xar1 apunta a gpatog
CIC_A			
	MOVL XAR2, #Ox(0001 ;	Carga registro xar2 con 0x0001
CIC_B			
	MOVL XARO, #0x:	ffff ;	Carga registro xarO con Oxffff
CIC_C			
	NOP	;	No operación
	NOP	;	No operación
	BANZ CIC_C, ARO	- ;	Salta a cic_c si ar0 no es igual a cero
		;	Decrementa registro ARO
	MOV *XAR1, #0x1	1000 ;	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, #Ox2	2000;	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 O :	/* Program Memory */
	/* Memory (RAM/FLASH) blocks can be moved to PAGE1 for data allocation \ast
	/* BEGIN is used for the "boot to Flash" bootloader mode
BEGIN	: origin = 0x080000, length = 0x000002
RAMMO	: origin = $0x000122$, length = $0x0002DE$
RAMDO	: origin = 0x00B000, length = 0x000800
RAMLSO	: origin = 0x008000, length = 0x000800
RAMLS1	: origin = 0x008800, length = 0x000800
RAMLS2	: origin = 0x009000, length = 0x000800
RAMLS3	: origin = 0x009800, length = 0x000800
RAMLS4	: origin = 0x00A000, length = 0x000800
RAMGS1	: origin = 0x01A000, length = 0x001000
RAMGS1	: origin = 0x01B000, length = 0x001000
RESET	: origin = 0x3FFFC0, length = 0x000002
/* Fla	sh 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 : /* Da /* Me	ta Memory */ emory (RAM/FLASH) blocks can be moved to PAGEO 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
RAMGSO	: 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	: $origin = 0x017000$, $length = 0x001000$
DAMCG10	6 , 6
RAMODIZ	: origin = 0x018000, length = 0x001000

}

```
SECTIONS
{
  /* Allocate program areas: */
                              PAGE = 0, ALIGN(4)
                 : > FLASHB
  .cinit
  .pinit
                 : > FLASHB,
                              PAGE = 0, ALIGN(4)
                 : >> FLASHB | FLASHC | FLASHD | FLASHE
                                                    PAGE = 0, ALIGN(4)
  .text
  codestart
                 : > BEGIN
                              PAGE = 0, ALIGN(4)
                 : LOAD = FLASHD,
  ramfuncs
                   RUN = RAMLSO | 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: */
                   : > RAMM1
                                  PAGE = 1
  .stack
                    : >> RAMLS5 | RAMGS0 | RAMGS1
  .ebss
                                                   PAGE = 1
                    : >> RAMLSO
                                                   PAGE = 0
  .data
  .esysmem
                    : > RAMLS5
                                  PAGE = 1
  /* Initalized sections go in Flash */
                    : >> FLASHF | FLASHG | FLASHH
                                                  PAGE = 0, ALIGN(4)
  .econst
                                 PAGE = 0, ALIGN(4)
  .switch
                    : > FLASHB
  .reset
                                PAGE = 0, TYPE = DSECT /* not used, */
                    : > RESET,
}
/*
// End of file.
*/
```