ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO DE PRÁCTICAS DE TRANSMISIÓN DE DATOS BASADO EN EL PROCESADOR DIGITAL DE SEÑAL (DSP) TMS320F2812"

INFORME DE PROYECTO INTEGRADOR

Previo a la obtención del Título de:

INGENIERO EN ELECTRICIDAD ESPECIALIZACIÓN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

Presentado por:

CHRISTOPHER DANIEL LOOR GARCIA

GABRIEL ANDRES NIVELA CORREA

GUAYAQUIL – ECUADOR

AÑO: 2016

AGRADECIMIENTOS

Agradezco a Dios de antemano por haberme permitido alcanzar este logro, a mis padres y a mi hermana querida que no sólo fueron el sostén de mis estudios a lo largo de mi carrera universitaria, sino también en el colegio y escuela donde siempre buscaron lo mejor para mí, aunque no esté dentro de su alcance.

A mi enamorada le agradezco por su apoyo incondicional, por sus consejos que me ayudaban a tomar las mejores decisiones. A mis amigos les agradezco por haber hecho está carrera más fácil, con sus bromas y estudios en grupo, los semestres se pasaban fáciles. Y por último a los profesores que nos brindaban sus conocimientos, y nos relataban sus experiencias, aunque a veces eran un poco estrictos, sabemos que su objetivo era prepararnos para la vida profesional fuera de la universidad.

Christopher Daniel Loor Garcia

Agradezco a Dios, quien ha estado conmigo en todo momento y ha forjado mi camino poniendo a las personas correctas en cada etapa de estos años, a mis padres y hermano, cimientos de mi vida, por su esfuerzo en darme lo mejor y el sustento diario. Además, agradezco al excelente grupo de amigos y compañeros con los que día a día compartimos alegrías y preocupaciones para conseguir nuestra anhelada meta, finalmente a mis maestros quienes marcaron mi crecimiento académico con sus enseñanzas y preparación de calidad.

Gabriel Andres Nivela Correa

DEDICATORIA

Dedicamos el presente proyecto a nuestros padres y hermanos que son nuestro ejemplo de lucha y perseverancia diaria, también agradecemos a aquellas personas que contribuyeron con el desarrollo del mismo, tanto en lo académico como en lo anímico.

TRIBUNAL DE EVALUACIÓN

Ph.D. Wilton Edixon Agila Gálvez
PROFESOR EVALUADOR

Msc. Damián Alberto Larco Gómez
PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual".

Christopher Daniel Loor Garcia

Gabriel Andrés Nivela Correa

RESUMEN

La industria moderna se constituye de sistemas digitales altamente sofisticados, capaces de procesar todo tipo de señales y con la capacidad de formar redes de comunicación industrial para transferencia de datos. Es allí donde los Procesadores Digitales de Señales intervienen.

Actualmente, el Laboratorio de Electrónica de Potencia cuenta con algunos kits de DSP (Procesador digital de señal), los cuales no han podido ser aprovechados al máximo por los estudiantes. Además, se debe mencionar que el DSP integra unidades periféricas de comunicación diseñadas para construir sistemas de comunicación industrial para transferencia de datos.

El Proyecto implementado abarca el diseño de prácticas para el manejo de las unidades periféricas de comunicación del procesador digitales de señales *TMS320F2812* de Texas Instruments.

La metodología de aprendizaje del presente proyecto está basada en el desarrollo de diez prácticas, mediante las cuales los estudiantes se introducen a la tecnología del procesamiento digital de señales y se fortalece sus conocimientos en la transmisión y recepción de información por interfaces SPI, SCI y por protocolo CAN.

En el primer capítulo se describe el problema que tienen sobre este tema los estudiantes de la carrera Ing. en Electricidad, Electrónica y Automatización Industrial. Y por último se da una solución a dicho problema mediante el diseño de prácticas para fortalecer el conocimiento de los estudiantes en sistemas de comunicación industrial con DSP.

En el segundo capítulo se presentan los conceptos básicos y diagrama de bloques del procesador (DSP) *TMS320F2812*. Además, se presentan la tarjeta *eZdsp™F2812* que tiene embebido el DSP, y la tarjeta *KSPS-0504 Adaptor Board* (desarrollada por la Universidad de Zwickau). Por último, se presenta el programa *Code Composer Studio V3.3*

En el tercer capítulo se describe los objetivos de cada práctica y una breve explicación de la práctica.

En los anexos se presentan la guía de prácticas desarrollada detalladamente, y la fundamentación teórica de los temas a desarrollar en las prácticas.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO 1

MARCO GENERAL

En este capítulo se pretende dar a conocer la existencia de los Kits de DSP (Procesador digital de señales) que se encuentran en el Laboratorio de Electrónica de Potencia de la Escuela Superior Politécnica del Litoral, se describe el problema que tienen sobre este tema los estudiantes de la carrera Ing. en Electricidad, Electrónica y Automatización Industrial. Y por último se da una solución a dicho problema mediante el uso de los Kits de DSP para fortalecer el conocimiento de los estudiantes en sistemas de comunicación industrial con dicha tecnología.

1.1 Descripción del problema

La industria moderna se constituye de sistemas digitales altamente sofisticados, capaces de procesar todo tipo de señales y con la capacidad de formar redes de comunicación industrial para transferencia de datos. El procesamiento analógico es reemplazado por los procesadores digitales con alto nivel de integración, mayor potencia, mayor rapidez, menor tamaño y costo. El curso de Laboratorio de Electrónica de Potencia en conjunto con el de Automatización Industrial II apuntan a contribuir con estos conocimientos en la formación profesional de cada estudiante.

Actualmente, el Laboratorio de Electrónica de Potencia cuenta con algunos kits de DSP (Procesador digital de señal), los cuales no han podido ser aprovechados al máximo por los estudiantes que toman el curso, debido a que dichos kits no se los emplean en las prácticas de laboratorio, trayendo como secuela una ausencia de estudiantes capacitados en aplicaciones de este tipo de procesamiento como lo son sistemas embebidos y control en tiempo real. Además, se debe mencionar que el DSP integra unidades periféricas de comunicación diseñadas para construir sistemas de comunicación industrial para transferencia de datos.

1.2 Justificación del Proyecto

El procesamiento digital de señal está a la vanguardia de la tecnología ya que constantemente se desarrollan métodos para análisis y tratamiento de señales de

forma digital, los cuales han provisto de una valiosa herramienta que es la programación en lenguaje de alto nivel, tanto para diseño como para modificación del código y además han brindado valiosos beneficios en cuanto a rapidez, costo, potencia y tamaño.

Ante los requerimientos de hoy en día en procesamiento digital de señal y en sistemas de comunicación para transferir datos con esta tecnología, el estudiante de electrónica y automatización en su pregrado debe ser capaz de incursionar en el desarrollo de software bajo plataformas de fabricantes de DSP dado que son estos procesadores los aptos para dichas condiciones.

El estudiante logrará la comprensión y el discernimiento de estos temas valiéndose de sus conocimientos en: programación en lenguaje C, microcontroladores, microprocesadores y comunicaciones industriales en conjunto con una guía que se provee en este proyecto.

1.3 Propuesta de Solución

Una vez recopiladas las limitaciones presentes en la formación de los estudiantes de Electrónica y Automatización Industrial de la Escuela Superior Politécnica del Litoral, este proyecto aportará con conocimientos acerca del diseño de sistemas de comunicación para transferencia de datos empleando un Controlador Digital de Señal(DSC) que integra en su interior un DSP.

Los fabricantes de DSP actualmente disponen de estos nuevos controladores (DSC), que gracias a sus características pueden tratar con el mundo real sin dificultad, pues en resumen mezclan las interesantes características de un microcontrolador con el DSP.

1.4 Objetivos

1.4.1 Objetivo General

- Diseñar e implementar prácticas para el aprendizaje de sistemas de comunicación para transferencia de datos con el procesador digital de señales *TMS320F2812* utilizando el software *Code Composer Studio* y las tarjetas ***eZdsp™F2812 y KSPS-0504-1002.***

1.4.2 Objetivos Específicos

- Introducir al estudiante en la tecnología del procesamiento digital de señales y la transferencia de datos usando DSPs.
- Familiarizar al estudiante con la herramienta de desarrollo *Code Composer Studio* y su programación en lenguaje C.
- Aprender a usar y configurar unidades periféricas del procesador *TMS320F2812* mediante los elementos de la tarjeta *KSPS-0504 Adaptor Board.*
- Aprender a usar y configurar las unidades periféricas de comunicación (SPI, SCI, CAN) del procesador *TMS320F2812* mediante los elementos de la tarjeta *KSPS-0504 Adaptor Board.*
- Analizar el procesamiento digital mediante la recepción y transmisión de datos en tiempo real sin afectación en cálculos.
- Distribuir el uso de las unidades periféricas de comunicación según sus propiedades y su capacidad para resolver una aplicación específica, dado sus requerimientos.

CAPÍTULO 2

DESCRIPCIÓN DE LA HERRAMIENTA DE SOFTWARE CODE COMPOSER STUDIO Y DE LAS TARJETAS *eZdsp™F2812* Y *KSPS-0504* QUE SE EMPLEAN EN EL DESARROLLO DE PRÁCTICAS

En este capítulo se presentan los conceptos básicos y diagrama de bloques del procesador *TMS320F2812* con el cual se va a trabajar. Además, se presentan la tarjeta *eZdsp™F2812* que tiene embebido el DSP, y la tarjeta *KSPS-0504 Adaptor Board* (desarrollada por la Universidad de Zwickau) que es donde se adapta la tarjeta *eZdsp™F2812.* Se dará una breve explicación para realizar un correcto uso de las tarjetas. Por último, se presenta el programa *Code Composer Studio V3.3* con sus respectivos drives que se usaran para la comunicación y programación del procesador *TMS320F2812.*

2.1 Procesador Digital de Señales (DSP)

2.1.1 Introducción al Controlador Digital de Señales (DSC)

Partiendo de la novedosa existencia de los microcontroladores, se diseña uno nuevo al que se le denomina Controlador Digital de señal, el cual abarca las mejores características para ser un diseño de altamente potenciado en sus funciones, dichas características pueden resumirse en un procesamiento al nivel de DSP, periféricos embebidos y memoria externa. El DSC lo integra todo en un solo chip, lo cual permite que su calificativo de controlador sea en correcto juicio. El DSP es la parte principal del DSC, es decir es la unidad de Procesamiento Digital de Señal del chip integrado. El procesador digital de señal es un tipo de microprocesador apto para manipular intensas operaciones matemáticas típicas. [1]

Dadas sus cualidades de interacción con el mundo real, los DSP propiamente dichos tienen una gran variedad de aplicaciones, siendo la de mayor demanda su uso en sistemas embebidos, es decir sistemas autónomos, por ejemplo,

celulares, impresoras, cámaras fotográficas, antenas digitales y en controladores digitales. En general estos elementos tienen gran demanda en las Tecnologías de la Información y las Comunicaciones, ya que cada vez se necesitan más recursos para procesar grandes volúmenes de datos.

En cuanto a la memoria, este controlador posee dos tipos, tal como los DSP, lo que le permite hacer uso de los datos simultáneamente a través de los dos tipos de buses que disponemos en su diseño. En cuanto a velocidad e interrupciones sofisticadas, el controlador es capaz de procesar algoritmos complejos programados y sus métodos de depuración son más robustos, lo cual brinda seguridad, en cuando a periféricos, el controlador cuenta con salidas y entradas digitales, así como analógicas, además de interfaces de comunicación, etc.

2.1.2 Tipo de Arquitectura

Básicamente el interior de un procesador viene dado por múltiples “bloques” que realizan funciones complementarias y que en conjunto permiten que se diseñe un procesador. Estos bloques tienen que estar conectados para que el proceso secuencia se dé de manera correcta, ya sea independiente o simultáneamente. Este diseño de bloques y conexiones es a lo que llamamos la arquitectura del procesador, es por eso que se menciona un poco de lo que se encuentra en el procesador *TMS320F2812*.

La arquitectura “Harvard” da la capacidad al procesador de recibir y transmitir datos en tiempo real, sin interrupción en sus operaciones matemáticas internas. Esto se debe a la partición de memorias que contiene en su interior, sus buses que trabajan simultáneamente para adquirir y escribir datos y la eficiencia al realizar el cálculo de multiplicación y acumulación en un solo ciclo. Cabe recalcar que este tipo de arquitectura es moderna, dejando de lado la convencional, denominada “Von Neumann”, la cual, emplea el mismo dispositivo de almacenamiento para instrucciones de código y para datos, lo cual implica a su vez que únicamente emplee un bus y por ende que no se puedan realizar varias funciones en un solo ciclo, con esto queda demostrada

la deficiencia de velocidad ante la arquitectura “Harvard”. Sin embargo, hoy en día se encuentra esta arquitectura convencional en la mayoría de los equipos debido a la flexibilidad que nos brinda la operación e implementación de un programa en vez de dos y programación libre.

2.1.3 Familia de Procesadores de Texas Instruments

Al valerse de equipos diseñados por Texas Instruments (TI), se introducirá al breve conocimiento de las familias de procesadores que fabrica TI, con lo que también se encontrara ciertas diferencias que serán de ayuda para la selección del indicado dado los requerimientos.

TMS320 se puede diferenciar entre tres familias, que se denominan C6000, C5000 y C2000.

C6000 es la familia de mayor rendimiento en cuanto a potencia de cálculos matemáticos, utiliza tanto punto fijo como punto flotante, por ende, sus aplicaciones son para procesamiento de gran cantidad de información como imágenes, audio, estaciones para antenas, etc. [1]

C5000 se usa en sistemas móviles como telefonía celular, por lo que tiene un eficiente consumo. No es tan potente como la primera familia, pero también se puede encontrar en reproductores de audio y cámaras fotográficas de baja gama. [1]

C2000 es la familia dedicada al potente control en tiempo real, y en la cual se encuentra el controlador con el que se realizarán las practicas diseñadas en el apartado. Se caracteriza por tener un costo mínimo integrando memoria y periféricos a diferencia de las dos familias mencionadas anteriormente. [1]

2.1.4 Procesador Digital de Señales TMS320F2812

Figura 2.1 Procesador Digital de Señal TMS320F2812 [2]

El dispositivo *TMS320F2812* es un Controlador Digital de Señales (DSC) que tiene embebido un procesador perteneciente a la generación de los DSP *TMS320C28X™,* de la familia C2000 y fabricado por Texas Instruments con una tecnología CMOS para un alto rendimiento, con un diseño para un consumo de baja potencia y una frecuencia de ciclo de 150 MHz [3]. El procesador tiene un CPU de 32 bit para un mayor rendimiento en aplicaciones de control, con una arquitectura de bus tipo “Harvard”, e integración de memorias, y unidades periféricas como de comunicación, I/O, convertidores ADC, manejadores de eventos, entre otros.

2.1.4.1 Diagrama de Bloques del procesador F2812

El diagrama de bloques del procesador *TMS320F2812* se muestra en la Figura 2.2 y se lo puede dividir en 4 partes:

- Bus interno y externo
- Unidad central de proceso (CPU)
- Memoria
- Unidades periféricas

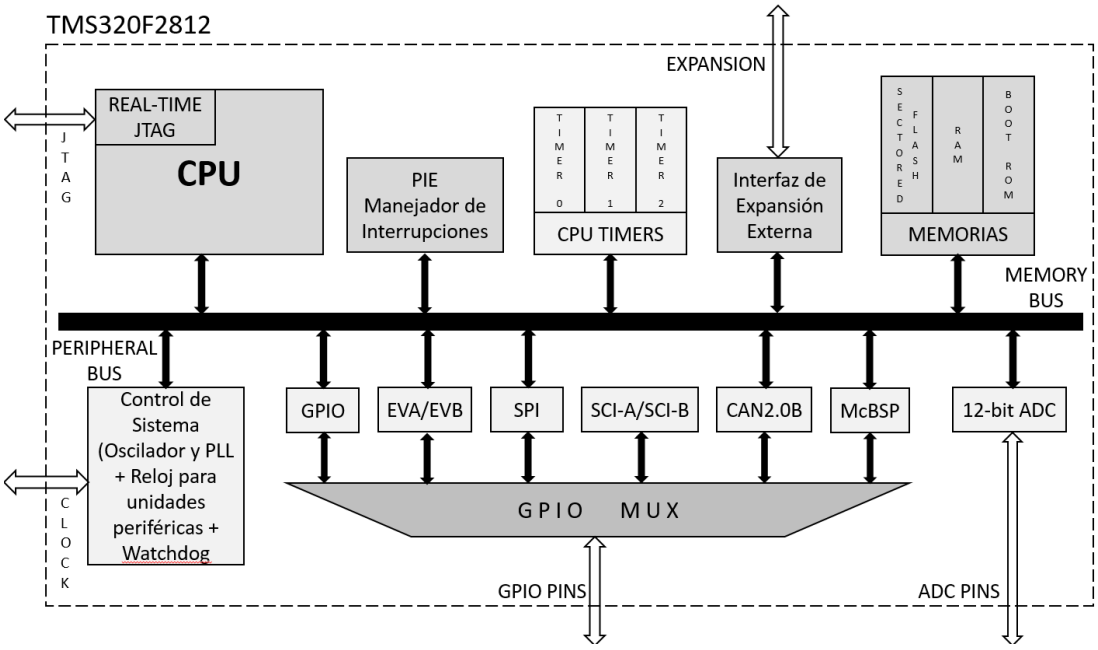


Figura 2.2 Diagrama de bloques del DSP TMS320F2812

Al integrar memorias y periféricos, el procesador *TMS320F2812* cumple eficientemente con funciones de un controlador, y para ello usa un tipo de arquitectura de bus Harvard que le permite al procesador conducir una instrucción, leer un dato y escribir un dato en un solo ciclo de reloj. El procesador *TMS320F2812* es considerado como un dispositivo de alto rendimiento debido al acceso a ambas memorias (de programa y datos) en el mismo ciclo de reloj.

La CPU puede ejecutar la mayoría de las instrucciones matemáticas gracias a los registros internos que posee, también posee tres temporizadores de 32 bits cada uno, un gestor de interrupciones para respuestas rápidas a eventos externos o internos, un multiplicador y una unidad aritmética lógica (ALU) de 32 bits, pudiendo las dos últimas funcionar en paralelo para ejecutar una multiplicación y suma simultáneamente.

El procesador *TMS320F2812* implementa la interfaz estándar JTAG (IEEE1149.1). Adicionalmente, el procesador incluye (mediante hardware) el modo de operación en tiempo real, por lo que el contenido de la memoria,

periféricos y registros puede ser modificado mientras el procesador está en funcionamiento.

Hay varios tipos de memorias dentro del procesador *F2812* como la memoria FLASH, RAM y ROM.

El bloque de expansión de interrupciones periféricas PIE cumple con funciones de multiplexar una numerosa cantidad de fuentes de interrupción para convertirlas en un conjunto pequeño de ellas. Se pueden agrupar hasta 96 interrupciones, de las cuales 45 son utilizadas por los periféricos. Las interrupciones se estructuran en grupos de 8 y cada grupo se identifica en la CPU (INT1 a INT12). Para acceder a una de ellas se necesita el vector que direcciona a su propio grupo. Toma 8 ciclos de reloj de procesador buscar la dirección que contiene el vector y guardar los registros críticos del procesador por lo que es considerado un procesador de respuesta rápida a eventos de interrupción [3]. Cada interrupción individual puede ser activada o desactivada dentro del bloque PIE.

La mayoría de las señales periféricas son multiplexadas con las señales GPIO (General-Purpose I/O). Esto nos permite seleccionar la función de cada pin según la aplicación. Los pines del procesador *TMS320F2812* se pueden configurar como entradas o salidas, se ajustan como entradas al reiniciar el procesador. Para ciertos pines configurados como entrada, también se les puede especificar el número de ciclos que debe transcurrir accionada para que sea activada

Con el objetivo de asociarse con los periféricos, el procesador *TMS320F2812* se encarga de manejar la interconexión de los buses de los periféricos, para ello, adopta un bus de periféricos estándar junto con un multiplexor. Con esto se consigue generar el bus de memoria del procesador. La prioridad de los accesos a la memoria de datos es la de escribir datos.

Entre los periféricos se tiene:

- Dos Event Manager (EVA, EVB)
- Un módulo convertidor de Analógico a Digital (ADC) de 12 bits

- 56 pines compartidos para entradas y salidas de propósito general o para funciones primarias
- Tres contadores de 32 bits (Timers 0, 1 y 2)
- Periféricos de comunicación serial

Los periféricos de comunicación serial que implementa el procesador *TMS320F2812* son:

- Unidad Periférica de Comunicación SPI
- Unidad Periférica de Comunicación SCI
- Unidad Periférica de Comunicación eCAN

2.1.4.2 Unidad Periférica de Comunicación SPI

SPI (Serial Peripheral Interface): La Interfaz Periférica Serial es un medio de alta velocidad que permite el intercambio de un flujo de bits (hasta 16 bits) de manera serial síncrona a una determinada velocidad. Esta interfaz es demandada en campos de comunicación entre un DSP y periféricos externos. Soporta una configuración de la red del tipo maestro/esclavo.

2.1.4.3 Unidad Periférica de Comunicación SCI

SCI (Serial Communication Interface): La Interfaz de Comunicación Serial es un medio de doble hilo que permite la comunicación entre un procesador y un periférico asíncrono. Brinda confianza y seguridad al usuario debido al trato íntegro que se le da al dato en todo momento de la comunicación. Usa el formato de trama de datos de la interfaz RS232C configurable con un bit de inicio, de 1 a 8 bits de datos, un bit opcional de paridad, y uno o dos bits de parada. El procesador *TMS320F2812* incluye dos módulos SCI (A y B). Con el objetivo de reducir la sobrecarga de servicio del procesador, esta unidad cuenta con 16 niveles independientes FIFO para transmisión y recepción. También es conocida como UART (Universal Asynchronous Receiver/Transmitter).

2.1.4.4 Unidad Periférica de Comunicación eCAN

eCAN (enhanced Controller Area Network): Es una versión mejorada basada en la estandarización de las capas 1 y 2 de la interfaz CAN 2.0B, que permite la comunicación serial con otros nodos (de 0 a 8 bytes de datos por trama) de manera rápida y segura en entornos de interferencia eléctrica. Cuenta con 32 buzones (Mailboxes) configurables para transmisión o recepción. Implementa sofisticados mecanismos de chequeo de errores y un manejador de acceso al bus de datos que evita las colisiones no deseadas. Ante problemas de comunicación, eCAN maneja trama de datos remotas con las cuales se puede notificar y reestablecer la comunicación.

2.2 Tarjeta *eZdsp™F2812*

La tarjeta *eZdsp™F2812* es un módulo independiente que permite evaluar las características del Procesador Digital de Señal *TMS320F2812* y así determinar su utilidad en las diferentes aplicaciones que se vaya a utilizar.

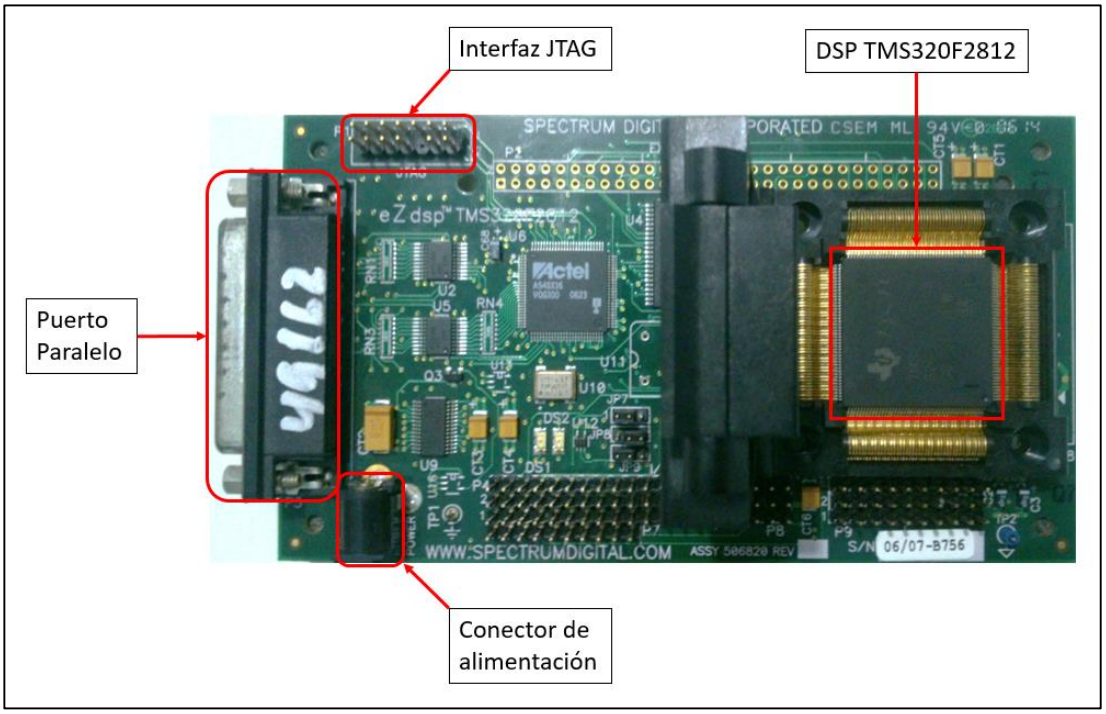


Figura 2.3 Descripción de la Tarjeta eZdsp™F2812

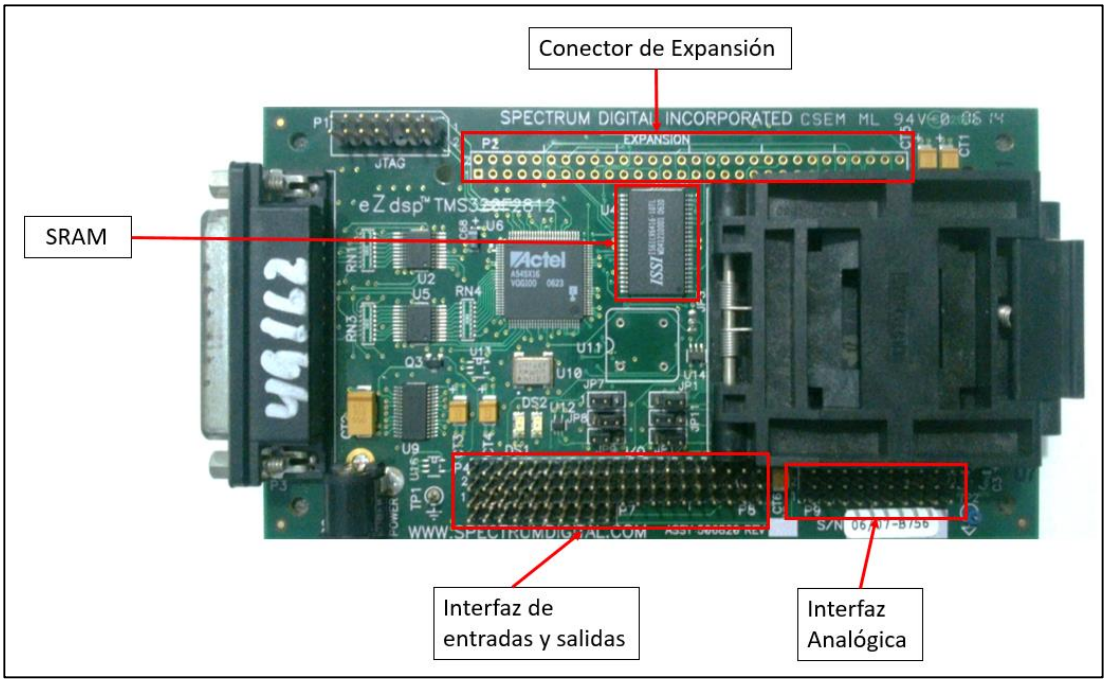


Figura 2.4 Descripción de la tarjeta eZdsp™F2812

2.2.1 Características de la tarjeta eZdsp™F2812

En las Figura 2.3 y Figura 2.4 se observa la tarjeta *eZdsp™F2812,* la cual tiene las siguientes características:

- Tiene embebido el Procesador Digital de Señal *TMS320F2812*
- Velocidad de operación de 150 MIPS (Millones de instrucciones por segundo)
- Una RAM con capacidad de numero de palabras de 18K words
- Una Flash con capacidad de numero de palabras de 128K words
- Una SRAM con capacidad de numero de palabras de 64K words
- 30 MHz de reloj
- Dos conectores de expansión (Analógica, I/O)
- Conector IEEE 1149.1 JTAG
- Puerto paralelo de 25 pines
- Requiere solamente 5 voltios para operar

2.2.2 Diagrama de bloques del eZdsp™F2812

La Figura 2.5 muestra el diagrama de bloques de la configuración básica de la tarjeta *eZdsp™F2812*. La cual tiene embebido el procesador digital de señal (DSP) *TMS320F2812*.

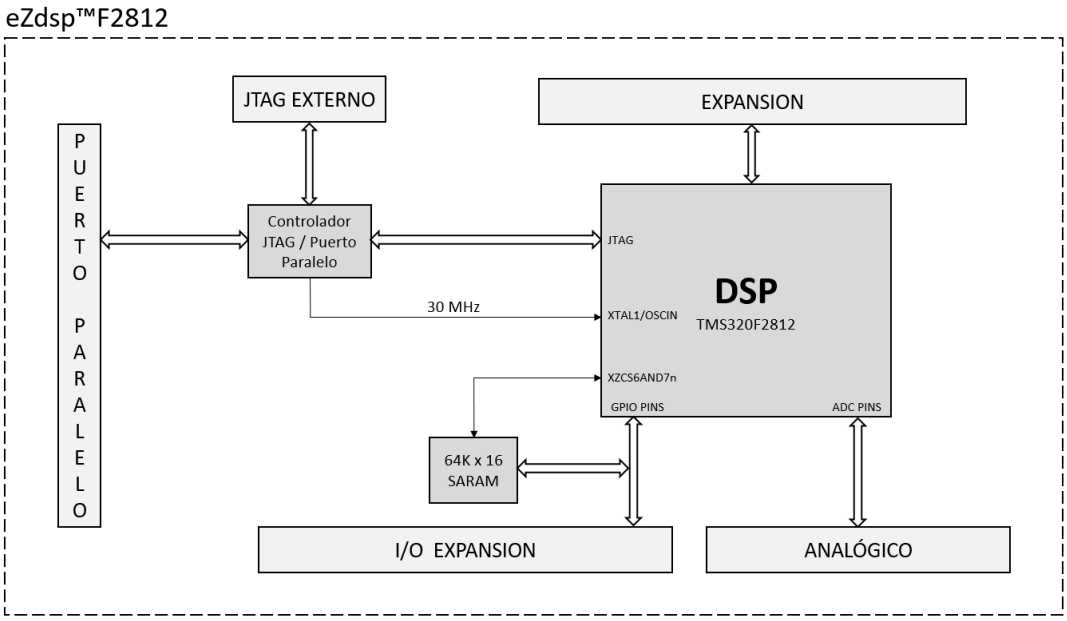


Figura 2.5 Diagrama de Bloques de la Tarjeta eZdsp™F2812 [4]

La tarjeta contiene un chip controlador de los puertos para realizar la conexión con el DSP, los puertos que contiene la tarjeta *eZdsp™F2812* son JTAG o Paralelo. Además, posee tres conectores para expansión: Externa, I/O, Analógico.

2.2.3 Conectores del *eZdsp™F2812*

La Figura 2.6 muestra la ubicación de cada conector en la tarjeta.

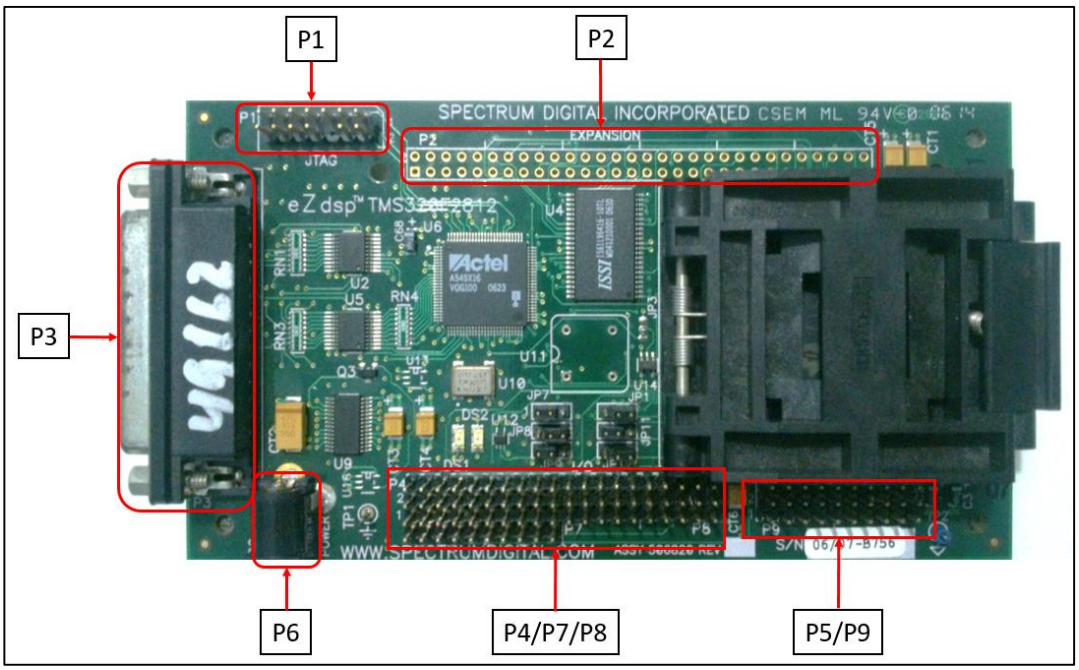


Figura 2.6 Conectores de la Tarjeta eZdsp™F2812

La *eZdsp™F2812* tiene cinco conectores como se puede observar en la Figura 2.6, la función de cada conector se muestra en la Tabla 1.

Conector	Función
P1	Interfaz JTAG
P2	Expansión
P3	Puerto Paralelo/ interfaz de controlador JTAG
P4/P8/P7	Interfaz I/O
P5/P9	Interfaz analógica
P6	Conector de energía

Tabla 1: Función de los conectores de la Tarjeta eZdsp™F2812 [4]

2.2.3.1 Conector P1, interfaz JTAG (Joint Test Action Group)

El conector P1 es una interfaz JTAG la cual consta de 14 pines. Esta interfaz estándar JTAG es muy usado por Texas Instruments para los DSPs. El interfaz JTAG es una herramienta que facilita el intercambio de datos en tiempo real, es decir, es posible observar variables en tiempo real, mientras se está ejecutando la aplicación. [4]

La localización de los pines en el conector P1 se muestra en la Figura 2.7

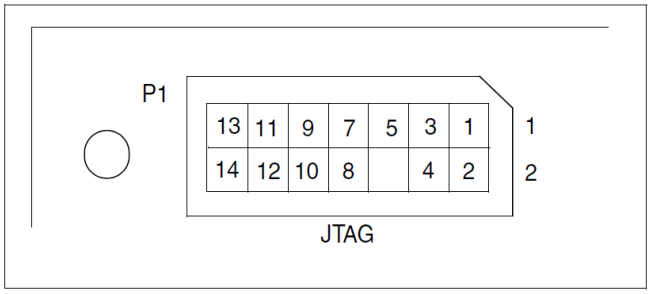


Figura 2.7 Conector P1, Interfaz JTAG [4]

La Tabla 2 muestra las señales de la interfaz JTAG relacionadas a cada pin del conector P1.

# Pin	Señal	# Pin	Señal
1	TMS	2	TRST-
3	TDI	4	GND
5	PD (+5V)	6	No pin
7	TDO	8	GND
9	TCK-RET	10	GND
11	TCK	12	GND
13	EMU0	14	EMU1

Tabla 2: Señales de la Interfaz JTAG [4]

2.2.3.2 Conector P2, interfaz de expansión

El conector P2 es una interfaz de expansión la cual consta de 60 pines. El presente proyecto no hace uso de este conector por lo que no se entrara en detalles.

2.2.3.3 Conector P3, puerto paralelo

El Conector P3 es un Puerto Paralelo. Este conector incorpora una interfaz de puerto paralelo que soporta ECP, EPP, y SPP8/Comunicaciones bidireccionales. Este conector tiene acceso directo a la interfaz integrada del JTAG [4].

2.2.3.4 Conector P4/P8/P7, Interfaz de entrada y salida

En los pines de los conectores P4, P8 y P7 se encuentran las señales de entradas y salidas del DSP, la numeración se muestra en la Figura 2.8

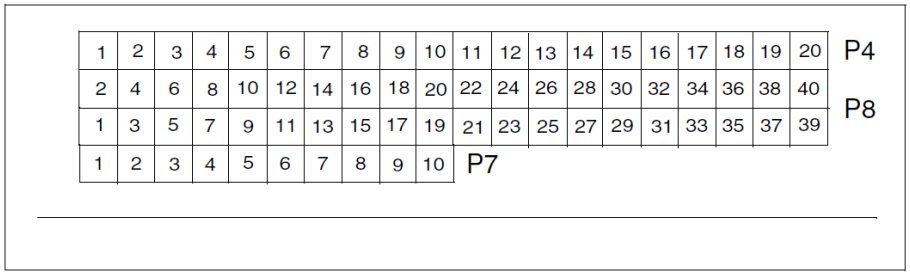


Figura 2.8 Conector de Interfaz de entrada y salida [4]

La Tabla 3 muestra las señales relacionadas a cada pin de los conectores P4 y P8, mientras la Tabla 4 muestra las señales relacionadas a cada pin del conector P7.

P4		**P8**			
# Pin	**Señal**	**# Pin**	**Señal**	**# Pin**	**Señal**
1	+3.3V/+5V/NC*	1	+3.3V/+5V/NC*	2	+3.3V/+5V/NC*
2	XINT2/ADCSOC	3	SCITXDA	4	SCIRXDA
3	MCLKXA	5	XINT1n/XBIOn	6	CAP1/QEP1
4	MCLKRA	7	CAP2/QEP2	8	CAP3/QEPI1
5	MFSXA	9	PWM1	10	PWM2
6	MFSRA	11	PWM3	12	PWM4
7	MDXA	13	PWM5	14	PWM6
8	MDRA	15	T1PWM/T1CMP	16	T2PWM/T2CMP
9	No conectado	17	TDIRA	18	TCLKINA
10	GND	19	GND	20	GND
11	CAP5/QEP4	21	No conectado	22	XINT1n/XBIOn
12	CAP6/QEPI2	23	SPISIMOA	24	SPISOMIA
13	T3PWM/T3CMP	25	SPICLKA	26	SPISTEA
14	T4PWM/T4CMP	27	CANTXA	28	CANRXA
15	TDIRB	29	XCLKOUT	30	PWM7
16	TCLKINB	31	PWM8	32	PWM9
17	XF/XPLLDISn	33	PWM10	34	PWM11
18	SCITXDB	35	PWM12	36	CAP4/QEP3
19	SCIRXDB	37	T1CTRIP/PDPINTAn	38	T3CTRIP/PDPINTBn
20	GND	39	GND	40	GND

Tabla 3: Señales de Interfaz de entrada y salida [4]

P7	
# Pin	**Señal**
1	C1TRIPn
2	C2TRIPn
3	C3TRIPn
4	T2CTRIPn/EVASOCn
5	C4TRIPn
6	C5TRIPn
7	C6TRIPn
8	T4CTRIPn/EVBSOCn
9	No conectado
10	GND

Tabla 4: Señales de Interfaz de entrada y salida [4]

2.2.3.5 Conector P5/P9, interfaz analógica

En los pines de los conectores P5 y P9 se encuentran las señales de la interfaz analógica del DSP, son un total de 30 pines y su numeración se muestra en la Figura 2.9

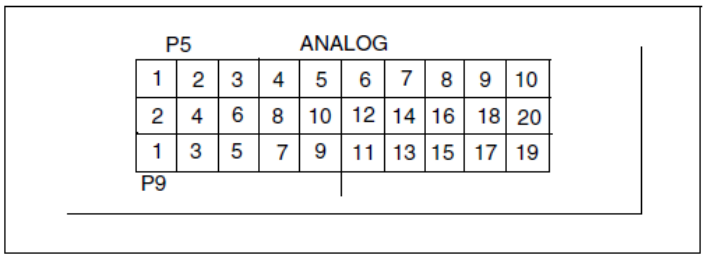


Figura 2.9 Conector de Interfaz Analógica [4]

La Tabla 5 muestra las señales relacionadas a cada pin del conector P5 y P9.

P5		P9			
# Pin	**Señal**	**# Pin**	**Señal**	**# Pin**	**Señal**
1	ADCINB0	1	GND	2	ADCINA0
2	ADCINB1	3	GND	4	ADCINA1
3	ADCINB2	5	GND	6	ADCINA2
4	ADCINB3	7	GND	8	ADCINA3
5	ADCINB4	9	GND	10	ADCINA4
6	ADCINB5	11	GND	12	ADCINA5
7	ADCINB6	13	GND	14	ADCINA6
8	ADCINB7	15	GND	16	ADCINA7
9	ADCREFM	17	GND	18	VREFLO*
10	ADCREFP	19	GND	20	No conectado

Tabla 5: Señales de Interfaz Analógica [4]

2.2.3.6 Conector P6, Conector de alimentación

La tarjeta *eZdsp™F2812* se alimenta con 5 voltios mediante el conector P6. El conector tiene un diámetro exterior de 5.5 milímetros. Y un diámetro interior de 2 mm [4].

El conector P6 de forma frontal se observa en la Figura 2.10

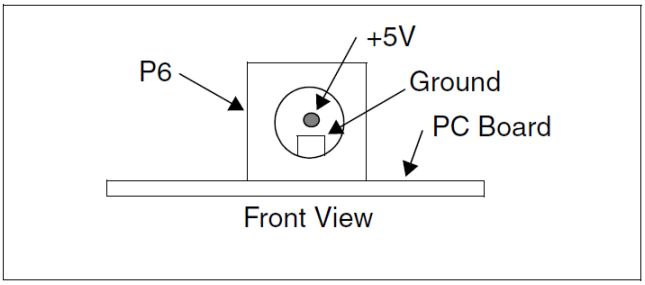


Figura 2.10 Descripción del conector de alimentación [4]

2.2.4 Jumpers, LEDs y puntos de prueba

La Figura 2.11 muestra la ubicación de los jumpers, LEDs y puntos de prueba de la parte superior de la tarjeta.

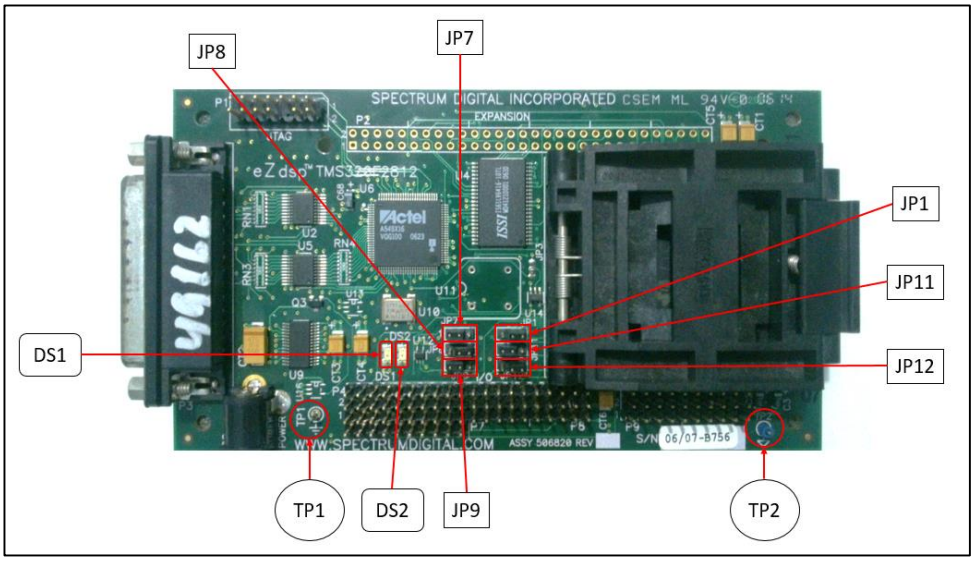


Figura 2.11 Jumpers de la tarjeta eZdsp™F2812

La Figura 2.12 muestra la ubicación de los jumpers de la parte inferior de la tarjeta.

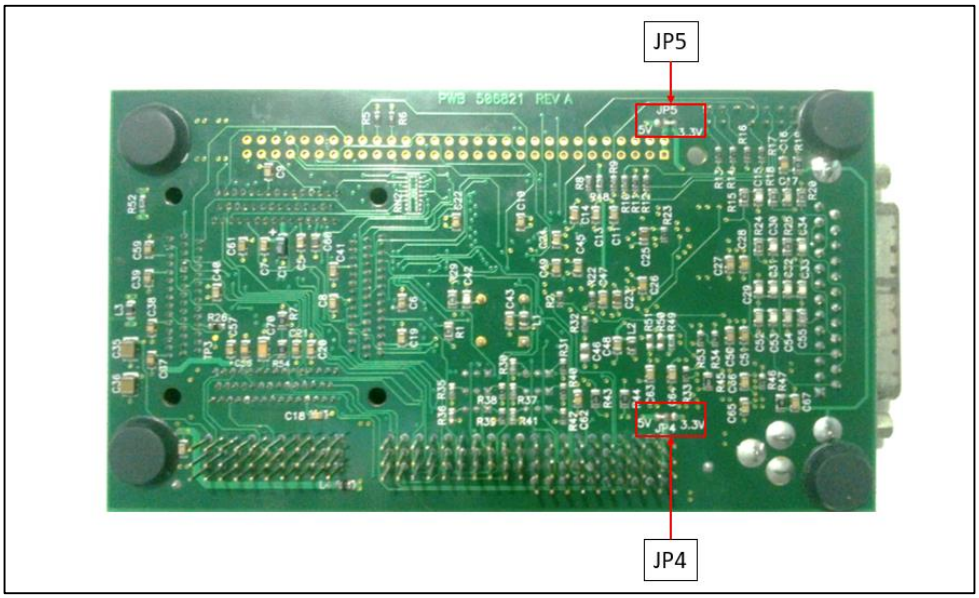


Figura 2.12 Jumpers de la tarjeta eZdsp™F2812

La *eZdsp™F2812* tiene ocho jumpers como se observa en las Figuras 2.11 y 2.12, la función de cada jumper se muestra en la Tabla 6.

# Jumper	Tamaño	Función	Posición *default*
JP1	1 x 3	XMP/MCn	2-3
JP4	1 x 3	+3.3/5 Volts a P8, P4	No conectado
JP5	1 x 3	+3.3/5 Volts a P2	No conectado
JP7	1 x 3	Boot Mode 3	2-3
JP8	1 x 3	Boot Mode 2	2-3
JP9	1 x 3	Deshabilitar PLL	1-2
JP11	1 x 3	Boot Mode 1	1-2
JP12	1 x 3	Boot Mode 0	2-3

Tabla 6: Funciones de los jumpers de la Tarjeta eZdsp™F2812 [4]

2.2.4.1 Jumper JP1, selección XMP/Mcn

El jumper JP1 es usado para seleccionar el modo XMP/MCn, ver Tabla 7. La selección en la posición 1-2 permite al DSP operar en el modo microcontrolador. La opción 2-3 permite al DSP operar en el modo Microprocesador [4]. La opción 2-3 es la que viene de fábrica.

Posición	Función
1-2	Modo Microcontrolador
2-3	Modo Microprocesador

Tabla 7: Función Jumper JP1 [4]

2.2.4.2 Jumper JP4, selección de voltaje

Jumper JP4 proporciona 3.3 o 5 voltios a los pines 1 y 2 del conector P8, y al pin 1 del conector P4. De fábrica no viene conectado este jumper [4]. La configuración de este jumper se muestra en la Tabla 8.

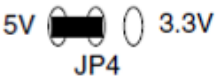
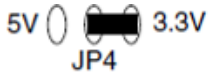
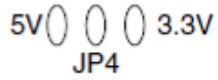
Posición	Función	Posición del Jumper
1-2	+5V a los pines 1 y 2 de P8 y al pin 1 de P4	
2-3	3.3V a los pines 1 y 2 de P8 y al pin 1 de P4	
No conectado*		

Tabla 8: Función Jumper JP4 [4]

2.2.4.3 Jumper JP5, selección de voltaje

Jumper JP5 proporciona +3,3 o +5 voltios a los pines 1 y 2 del conector P2. De fábrica no viene conectado este jumper [4]. La configuración de este jumper se muestra en la Tabla 9.

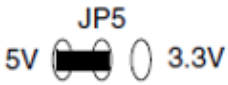
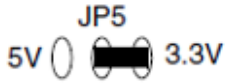
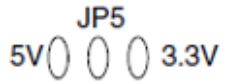
Posición	Función	Posición del Jumper
1-2	+5 V a los pines 1 y 2 de P2	
2-3	3.3V a los pines 1 y 2 de P2	
No conectado*		

Tabla 9: Función Jumper JP5 [4]

2.2.4.4 Jumpers JP7, JP8, JP11 y JP12, selección del modo de arranque

Los jumpers JP7, JP8, JP11 y JP12 son usados para determinar en qué modo se cargará el código a la DSP durante el arranque. Para establecer un nivel

de voltaje alto, se debe colocar el jumper en la posición 1-2. Para un nivel de voltaje bajo, utilice la posición 2-3 [4]. Las opciones se muestran en la Tabla 10.

JP7, BOOT3 SCITXDA	JP8, BOOT2 MDXA	JP11, BOOT1 SPISTEA	JP12, BOOT0 SPICLKA	MODO
1	x	x	x	FLASH
0	1	x	x	SPI
0	0	1	1	SCI
0	0	1	0	H0*
0	0	0	1	OTP
0	0	0	0	PARALLEL

Tabla 10: Configuración de Jumpers JP7, JP8, JP11 y JP12 [4]

2.2.4.5 Jumper JP9, deshabilitar PLL

El jumper JP9 se emplea para habilitar o deshabilitar el uso del *Phase Lock Loop* (PLL) en el DSP [4]. De fábrica el PLL viene habilitado. Las posiciones de JP9 se muestran en la Tabla 11.

Posición	Función
1-2*	Habilitar PLL
2-3	Deshabilitar PLL

Tabla 11: Función Jumper JP9 [4]

2.2.4.6 LEDs

La tarjeta ***eZdsp™F2812*** tiene dos diodos LED. El diodo DS1 indica la presencia de +5 voltios en la tarjeta. El diodo DS2 está bajo un control de software y está ligado al pin XF del DSP a través de un buffer [4]. Ver Tabla 12.

# LED	Color	Señal
DS1	Green	+5 voltios
DS2	Green	XF bit

Tabla 12: LEDs de la Tarjeta eZdsp™F2812 [4]

2.2.4.7 Puntos de Prueba

La tarjeta *eZdsp™F2812* tiene dos puntos de prueba que se describen en la Tabla 13.

Puntos de Prueba	Señal
TP1	Ground
TP2	Referencia analógica

Tabla 13: Puntos de prueba de la tarjeta eZdsp™F2812 [4]

2.3 Tarjeta *KSPS-0504* (Zwickau Adaptor Board)

La tarjeta de la Figura 2.13 fue diseñada por el Dr. Frank Bormann de la Universidad de Zwickau para ser utilizado con las tarjetas *eZdsp™F2812 o eZdsp™F2808* (Desarrolladas por Spectrum Digital).

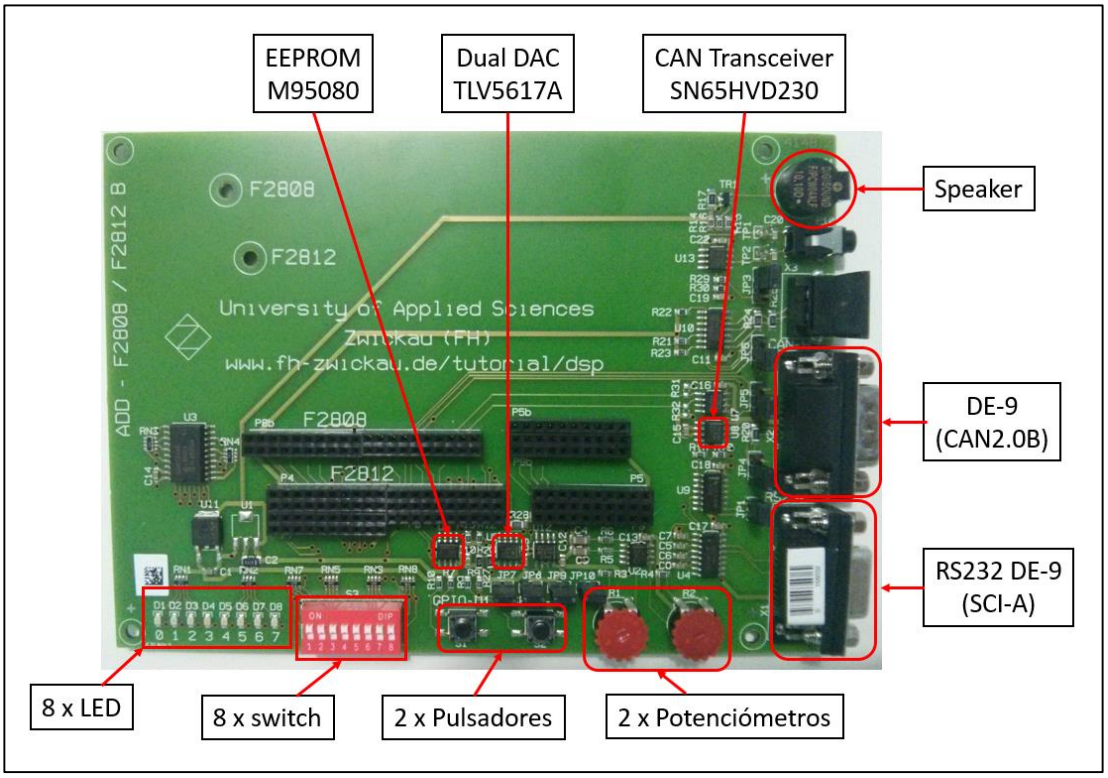


Figura 2.13 Elementos de la Tarjeta KSPS-0504 Adaptor Board

Esta tarjeta consta de las siguientes características:

- 8 LEDs conectados a los pines GPIOB7 hasta B0
- 8 DIP switches conectados a los pines GPIOB15 hasta B8
- 2 pulsadores conectados a los pines GPIOD1 y GPIOD6
- Loudspeaker conectado a la salida T1PWM
- 2 Potenciómetros de 0-3V aplicados a los canales ADC A0 y B0
- Puerto SCI-A conectado a transceptores RS232 y salida con conector DE-9 hembra.
- Salida SPI conectada a una EEPROM serial (M95080) (Selección programable por pin GPIOD5) y a una DAC serial (TLV5617A) (Selección programable por pin GPIOD0)
- Módulo CAN2.0B conectado a chip SN65HVD230 y salida con conector DE-9 macho.

2.4 Software Code Composer Studio

Figura 2.14 Herramienta de desarrollo Code Composer Studio™ V3.3

Texas Instruments ofrece una herramienta de software para la familia C2000, conocida como *Code Composer Studio*™, siendo este programa el más completo para el ambiente de desarrollo de proyecto con DSP. En *Code Composer Studio* se puede programar tanto en lenguaje C/C++ como en Ensamblador.

En todo ambiente de desarrollo de sistemas basados en controladores y procesadores, es importante adquirir habilidades y comprensión en la herramienta de software que se empleará para llevar a cabo los objetivos del proyecto.

Todo proyecto contiene los siguientes archivos fuente:

- Código fuente (C, ensamblador)
- Librerías
- Configuración DSP/BIOS
- Archivos de comandos de enlace.

Asimismo, en todo proyecto se realizan ajustes de programa, siendo los siguientes los más importantes:

- Build Options (compilar y ensamblar)
- Build configurations
- DSP/BIOS
- Linker

2.5 Diagrama de bloques de conexiones entre hardware y software

Una vez hecha la descripción de las tarjetas y el software que se emplearán en el desarrollo del proyecto, se describirá brevemente mediante la Figura 2.15 las etapas y los elementos que intervienen dentro de cada una de ellas.

Figura 2.15 Conexiones entre Hardware y Software

El computador debe contar con los programas Code Composer Studio™ IDE y Setup Code Composer Studio™ de la versión 3.3, además de ello un puerto físico de comunicación como lo son los puertos USB.

A continuación, se tiene un bloque de conversión, en el que interviene el emulador XDS510LC USB JTAG que cumplirá con la función de interpretar los datos de una interface JTAG (Joint Test Action Group) hacia una USB (Universal Serial Bus) y viceversa, cabe recalcar que al bloque del computador debe instalársele los drivers respectivos que se proporciona en el kit en un CD.

Para lograr la completa comunicación entre el mundo real y el mundo digital es necesario configurar el puerto USB que se utilizará, para ello se emplea el software Setup Code Composer Studio™, los detalles de configuración se describen en el anexo de la Práctica # 1.

Finalmente se tienen las tarjetas electrónicas (*eZdsp™F2812* y *KSPS-0504 Adaptor Board*) que ya se han descrito anteriormente, siendo la *eZdsp™F2812* quien recibe la información de programa y de datos a través del Conector P1 (JTAG). Por otro lado, la tarjeta *KSPS-0504 Adaptor Board* consta de elementos para hacer posible el diseño de las prácticas de transferencia de datos mediante SPI, SCI y CAN. Ambas tarjetas se enlazan directamente mediante espadines por los Conectores P4, P5, P7, P8 y P9 de la tarjeta *eZdsp™F2812*.

CAPÍTULO 3

APORTE DEL DISEÑO DE LAS PRÁCTICAS SOBRE TRANSFERENCIA DE DATOS CON EL PROCESADOR DIGITAL DE SEÑALES.

3.1 Descripción de la Práctica # 1

"Introducción a la herramienta de desarrollo de programa Code Composer Studio mediante la creación de un proyecto para el manejo de entradas y salidas digitales del DSP *TMS320F2812*"

3.1.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Conectar el DSP *TMS320F2812* con el programa *Code Composer Studio*.
- Familiarizar al estudiante con la herramienta de desarrollo *Code Composer Studio* y su programación.
- Crear, compilar un proyecto en *Code Composer Studio.*
- Aprender a configurar el módulo de reloj y el perro guardián del procesador.
- Aprender a configurar los pines del procesador para que sean usados como entradas o salidas digitales.

3.1.2 Breve explicación de la practica

En el Procesador Digital de Señales (DSP) no solo tenemos un núcleo de 32 bits, sino también algunas unidades periféricas necesarias para poder resolver varios problemas en distintos tipos de aplicaciones. Entre las unidades periféricas tenemos el módulo de entrada y salidas digitales.

Lo primero que se debe configurar en los DSPs es el módulo de reloj. El *TMS320F2812* es temporizado por un oscilador externo por los pines *X1* y *X2*, este oscilador es más lento para así reducir el ruido electromagnético. Se debe programar el circuito interno PLL del módulo de reloj para que genere una velocidad interna de 150 MHz al CPU del procesador a partir del oscilador externo de 30 MHz.

Luego se debe configurar los pines del procesador para que funciones como entradas y salidas digitales, según las conexiones entre la tarjeta *eZdsp™F2812* y *KSPS-0504 Adaptor Board*. Ver Figura 3.1

Finalmente se debe programar que cuando se presione el pulsador S1 (Conectado a GPIOD1), los estados de los switches (Conectados desde GPIO B15 a B8) se los visualizará en los leds (Conectados desde GPIO B7 a B0). Por otro lado, cuando se presione el pulsador S2 (Conectado a GPIOD6), se iniciará una secuencia de encendido de los leds (Conectados desde GPIO B7 a B0), en el cual el encendido se desplaza entre led y led.

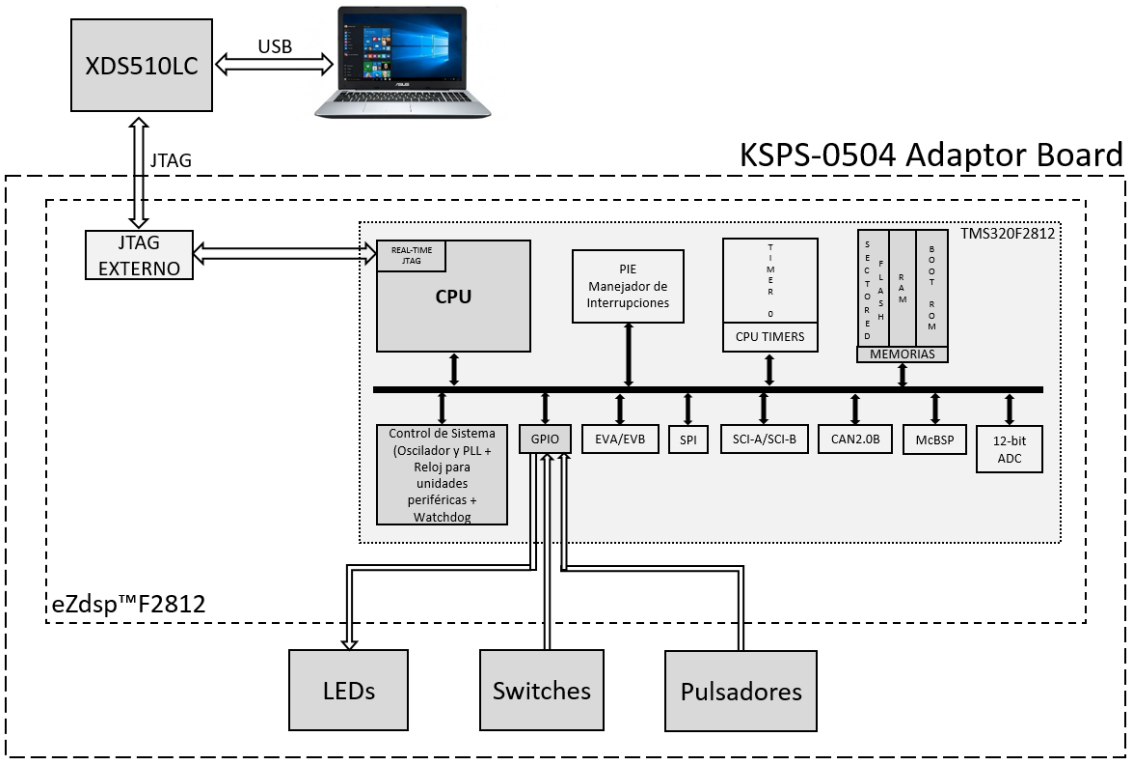


Figura 3.1 Diagrama de Bloques del manejo de entradas y salidas digitales

3.2 Descripción de la Práctica # 2

"Manejo del tiempo en el desplazamiento del encendido de los leds usando las interrupciones de un temporizador interno (Timer0) del DSP *TMS320F2812*"

3.2.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Habilitar y configurar el sistema de interrupciones para poder usar cualquier fuente de interrupción de las unidades periféricas del procesador *TMS320F2812*.
- Aprender a usar el "Timer0" interno del procesador para generar interrupciones cada 50 milisegundos.
- Diferenciar las ventajas entre el uso de retardos por hardware que usando retardos por software.

3.2.2 Breve explicación de la practica

Las interrupciones permiten detectar eventos en cualquier momento sin importar en que sección de código se encuentre. Debido a esto el uso de interrupciones es de importancia para el correcto manejo de las unidades periféricas de los DSP como lo son: el manejador de eventos, convertidor analógico a digital, temporizadores internos, SPI, SCI, CAN, entre otros.

Se debe programar que cuando se presione el pulsador S1 (Conectado a GPIOD1), arranque la secuencia de desplazamiento de los leds (Conectados desde GPIO B7 a B0). Por otro lado, cuando se presione el pulsador S2 (Conectado a GPIOD6), se detendrá la secuencia de desplazamiento de los leds (Conectados desde GPIO B7 a B0). Los pulsadores S1 y S2 funcionan como arranque (Start) y parada (Stop) respectivamente. Ver Figura 3.2

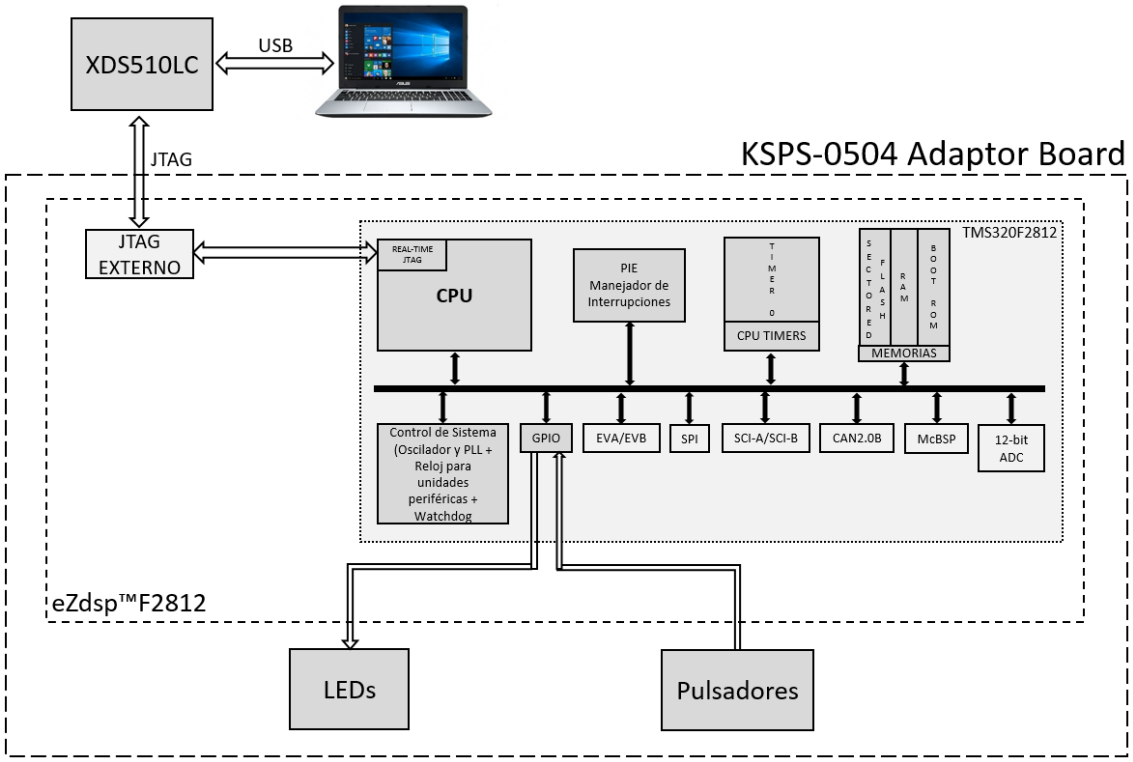


Figura 3.2 Diagrama de Bloques para el uso de Interrupciones

3.3 Descripción de la Práctica # 3

"Comunicación mediante la interfaz SPI entre el procesador TMS320F2812 y el Convertidor Digital a Analógico TLV5617"

3.3.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Establecer comunicación SPI entre el procesador *TMS320F2812* y el convertidor digital a analógico TLV5617.
- Generar y transmitir señales digitales de forma diente de sierra mediante el procesador *TMS320F2812* para sea convertida por el DAC TLV5617.
- Observar el valor analógico que convierte el chip integrado TLV5617 a través del visualizador de gráficos y el modo Tiempo Real de Code Composer Studio.

3.3.2 Breve explicación de la practica

El procesador *TMS320F2812* embebe unidades periféricas de comunicación, entre ellas la unidad SPI, la misma que establecerá una interfaz SPI entre el procesador y el chip DAC TLV5617 que es parte de la tarjeta *KSPS-0504 Adaptor Board*. La configuración para el uso de la unidad SPI se la realiza en lenguaje C mediante el software Code Composer Studio en conjunto con las librerías y archivos código fuente provistos por Texas Instruments para el manejo de unidades periféricas del procesador *TMS320F2812*.

Se generan dos señales de forma diente de sierra mediante el incremento y decremento de dos variables, estas señales se trasmiten desde el procesador hacia el chip DAC TLV5617 mediante la interfaz SPI. El periférico externo TLV5617 convierte la señal digital en una señal analógica periódicamente en función del reloj dado por la unidad SPI del procesador *TMS320F2812*.

Al final de la práctica se podrán evidenciar en tiempo real los valores de los pines de salida del chip TLV5617 con ayuda del ADC interno del procesador y con las herramientas de visualización gráfica y numérica del software. Ver Figura 3.3

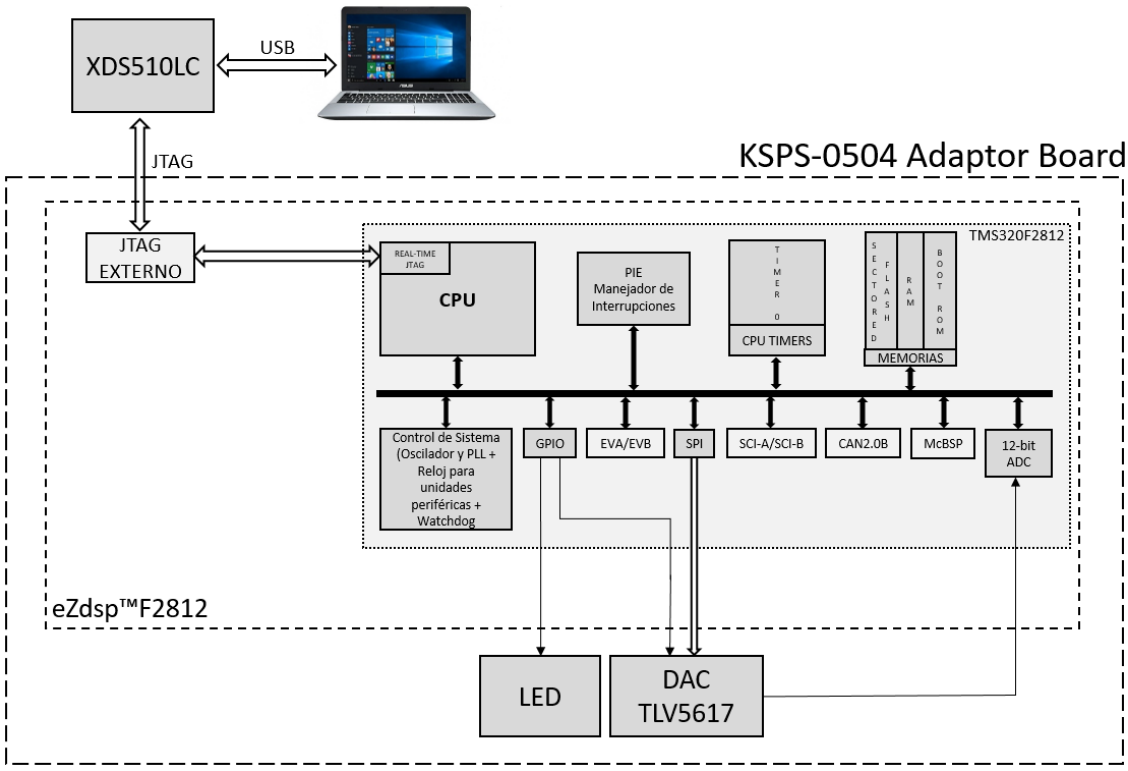


Figura 3.3 Diagrama de Bloques de Comunicación SPI entre DSP y Chip DAC

3.4 Descripción de la Práctica # 4

"Comunicación mediante interfaz SPI entre el procesador TMS320F2812 y la memoria EEPROM M95080"

3.4.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Establecer comunicación SPI entre el procesador *TMS320F2812* y la memoria EEPROM serial M95080.
- Transmitir y almacenar datos de las entradas (GPIO B15 a B8) del procesador *TMS320F2812* hacia una dirección de la EEPROM.
- Leer datos almacenados en la misma dirección de la memoria EEPROM utilizada en el objetivo anterior y mostrar los datos en las salidas (GPIO B7 a B0) del procesador *TMS320F2812*.

3.4.2 Breve explicación de la practica

La memoria EEPROM es conocida como memoria no volátil, porque ante una interrupción de la fuente de energía, los datos almacenados no son borrados. Por lo que generalmente se usan para almacenar información sobre programación, estados de las entradas y salidas, fallas, entre otros.

Utilizando la unidad periférica de comunicación SPI, se establecerá la comunicación entre la interfaz SPI del procesador *TMS320F2812* y la memoria EEPROM que es parte de la tarjeta *KSPS-0504 Adaptor Board*. La configuración para el uso de la unidad SPI se la realiza en lenguaje C mediante el software Code Composer Studio en conjunto con las librerías y archivos código fuente provistos por Texas Instruments para el manejo de unidades periféricas del procesador.

Cuando se presiona el pulsador S1 (Conectado GPIO D1), se transmitirá una instrucción para poder escribir en la memoria EEPROM los estados de las entradas GPIO B15 a B8 del procesador *TMS320F2812*. Cuando se presione

el pulsador S2 (Conectado GPIO D6), se transmitirá una instrucción para poder leer en la memoria EEPROM los estados de las entradas que se escribieron previamente, y así mostrar los estados en las salidas GPIO B7 a B0. Ver Figura 3.4

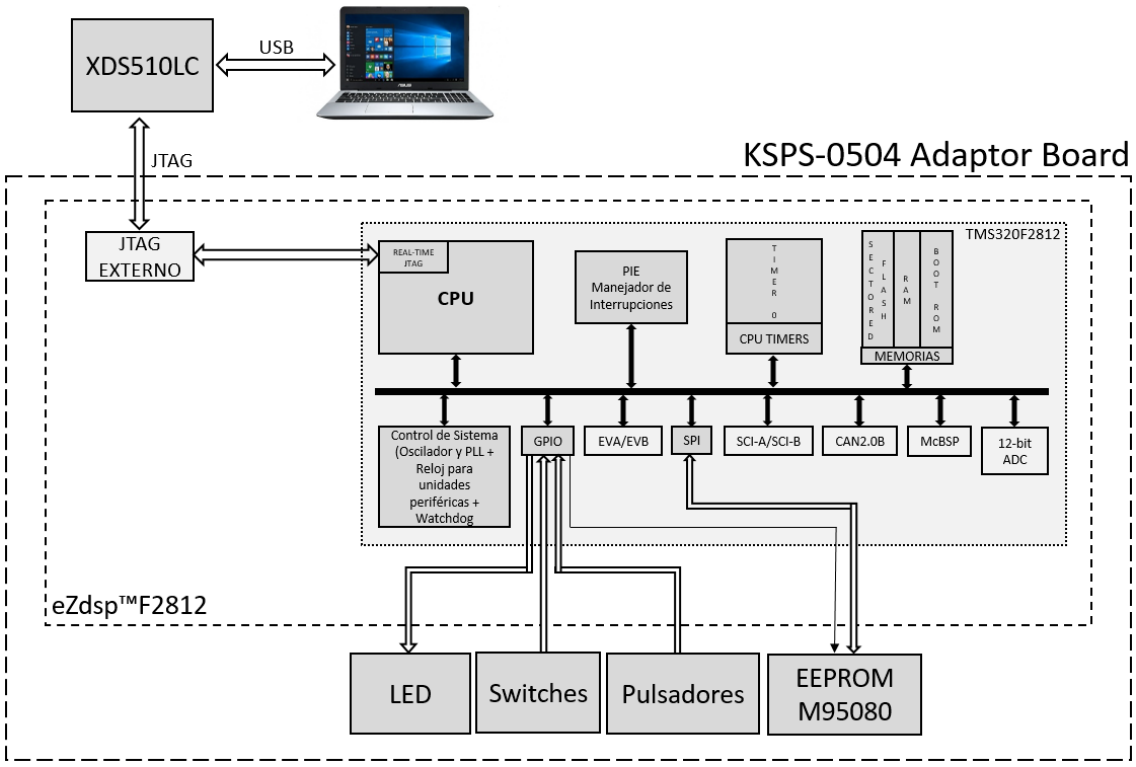


Figura 3.4 Diagrama de Bloques de Comunicación SPI entre DSP y EEPROM

3.5 Descripción de la Práctica # 5

"Transmisión de datos mediante interfaz SCI entre el procesador TMS320F2812 y la PC"

3.5.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Configurar el formato de la trama y la velocidad para una comunicación SCI.
- Establecer comunicación SCI entre el procesador *TMS320F2812* y la PC.
- Transmitir un texto desde el procesador *TMS320F2812* hacia el puerto seria de la computadora.

3.5.2 Breve explicación de la practica

Serial Communication Interface (SCI) es una interfaz asíncrona también conocida como UART (Universal Asynchronous Receiver-Transmitter). La unidad periférica SCI soporta una comunicación digital entre el CPU y otros periféricos asíncronos externos que usen el formato estándar non-return-to-zero (NRZ). La unidad SCI puede operar tanto en comunicación half-duplex como full-duplex.

Utilizando la unidad periférica SCI, se establecerá la comunicación entre el procesador *TMS320F2812* y el puerto serial de la PC. La configuración para el uso de la unidad periférica SCI se la realiza por programa mediante el software Code Composer Studio. Para poder establecer la comunicación SCI se debe configurar también la trama y velocidad en el puerto COM de la PC y en el programa HyperTerminal.

Se transmitirá la frase "F2812 está comunicando \n\r" mediante la interfaz RS232 el cual se podrá visualizar en la ventana principal del programa HyperTerminal cada 2 segundos aproximadamente. El retardo de tiempo se

generará por software. El LED conectado a GPIO B0 será el indicador de la transmisión de datos, es decir, cada vez que se esté transmitiendo caracteres del texto el LED estará prendido. Ver Figura 3.5

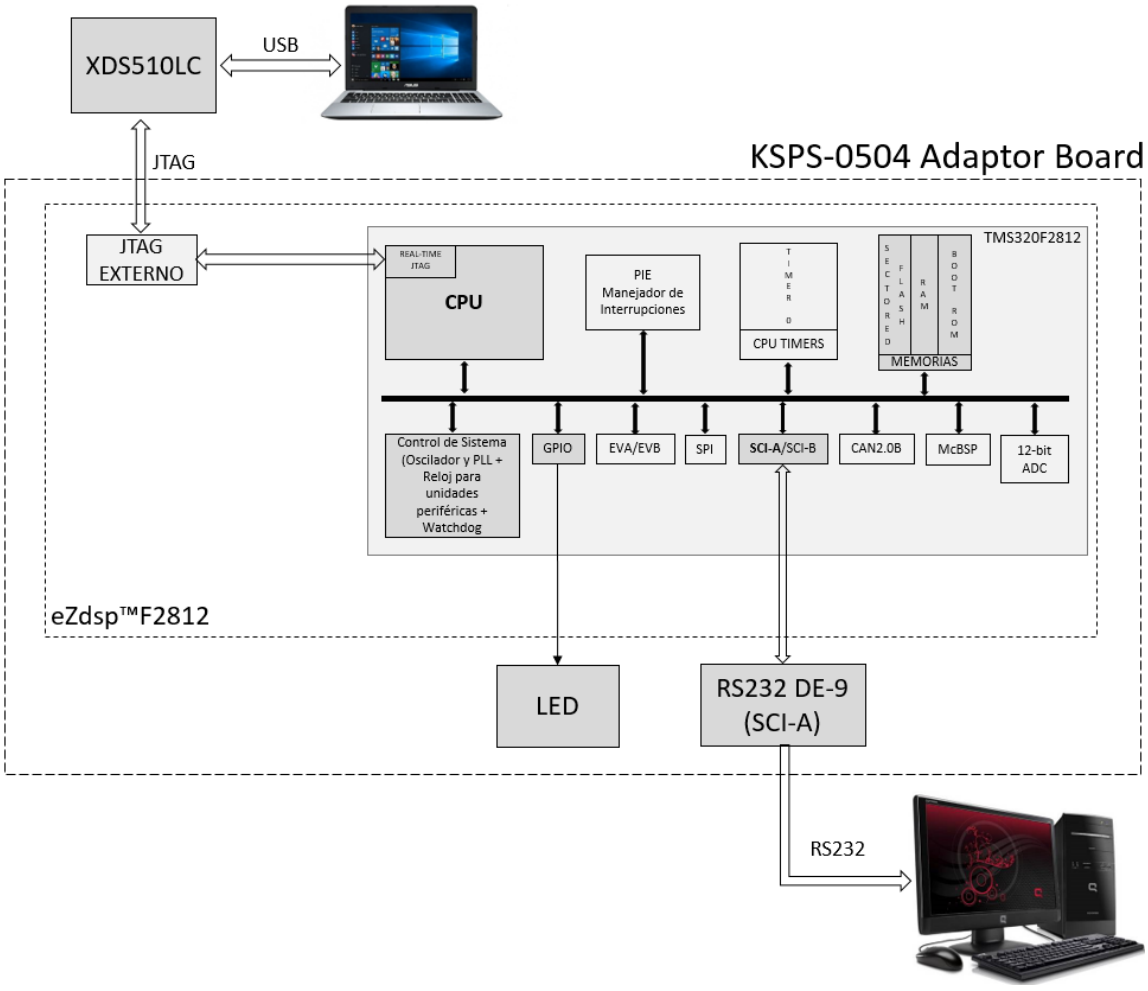


Figura 3.5 Diagrama de Bloques de Comunicación SCI entre DSP y PC

3.6 Descripción de la Práctica # 6

"Transmisión de datos a una PC mediante la interfaz de comunicación SCI y el sistema de interrupciones del procesador TMS320F2812"

3.6.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Configurar el formato de la trama, la velocidad, y habilitar el sistema de interrupciones para una comunicación SCI.
- Establecer comunicación SCI entre el procesador *TMS320F2812* y la PC.
- Transmitir un texto desde el procesador *TMS320F2812* hacia el puerto serial de la computadora cada 2 segundos.

3.6.2 Breve explicación de la practica

Las interrupciones permiten detectar eventos en cualquier momento sin importar en que sección de código se encuentre. Debido a esto el uso de interrupciones es de importancia para el correcto manejo de las unidades periféricas de los DSP como lo son: el manejador de eventos, convertidor analógico a digital, temporizadores, SPI, SCI, CAN, entre otros.

Utilizando la unidad periférica SCI, y el manejo de interrupciones se establecerá la comunicación entre en procesador *TMS320F2812* y el puerto serial de la PC. La configuración para el uso de la unidad periférica SCI se la realiza por programa mediante el software Code Composer Studio. Para poder establecer la comunicación SCI se debe configurar también la trama y velocidad en el puerto COM de la PC y en el programa HyperTerminal.

Se transmitirá la frase "F2812 esta comunicando \n\r" mediante la interfaz RS232 el cual se podrá visualizar en la ventana principal del programa HyperTerminal cada 2 segundos. El retardo de tiempo se generará por hardware usando los timers internos del DSP. El LED conectado a GPIO B0

será el indicador de la transmisión de datos, es decir, cada vez que se esté transmitiendo caracteres del texto el LED estará prendido. Ver Figura 3.6

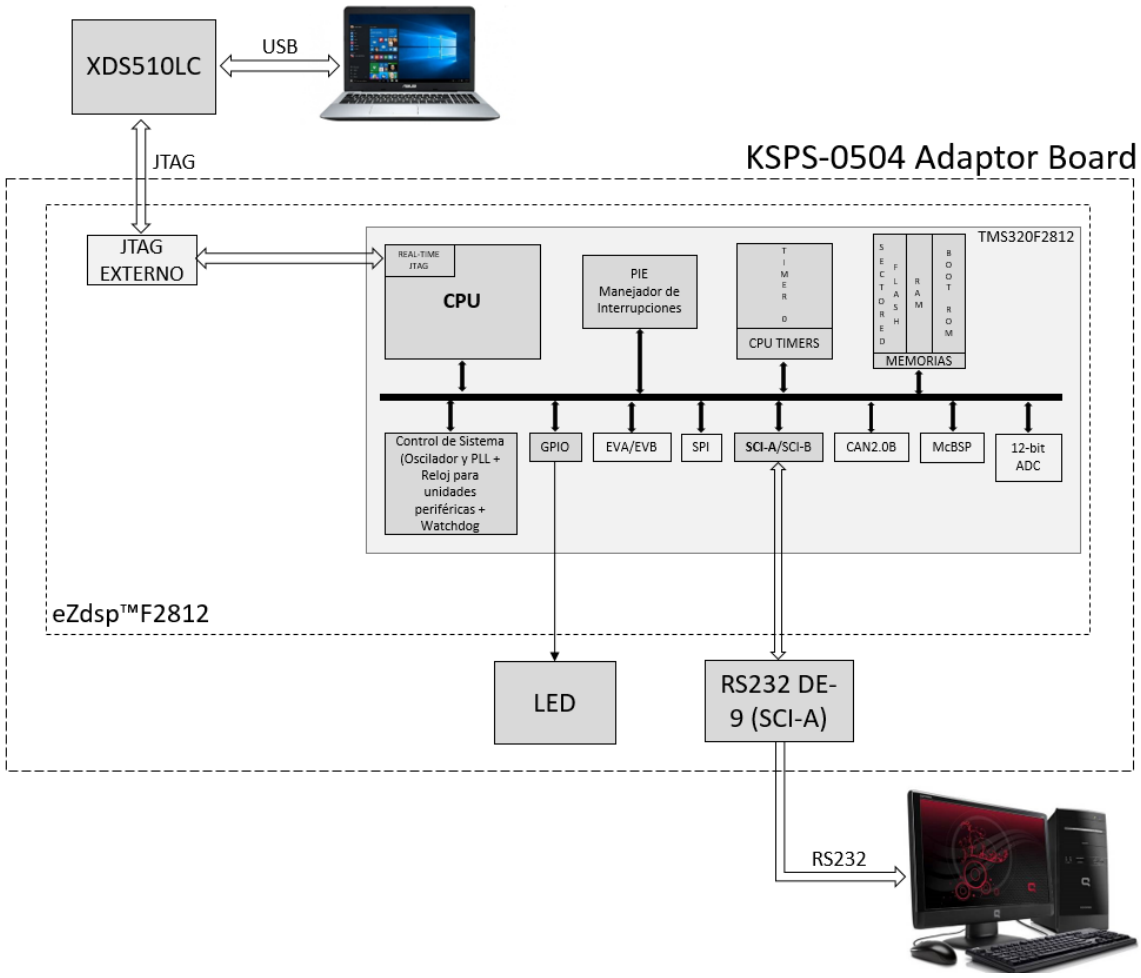


Figura 3.6 Diagrama de Bloques de Comunicación SCI entre DSP y PC usando Interrupciones

3.7 Descripción de la Práctica # 7

"Transmisión de datos a una PC mediante la unidad FIFO de la interfaz de comunicación SCI y el sistema de interrupciones del procesador TMS320F2812"

3.7.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Mejorar la Práctica 5 mediante la implementación del método FIFO de la comunicación SCI
- Configurar el formato de la trama, la velocidad, y habilitar el sistema de interrupciones para una comunicación SCI.
- Establecer comunicación SCI entre el procesador *TMS320F2812* y la PC.
- Transmitir un texto desde el procesador *TMS320F2812* hacia el puerto serial de la computadora cada 2 segundos.

3.7.2 Breve explicación de la practica

En lugar de generar una gran cantidad de interrupciones SCI como en la práctica anterior, se va a utilizar la técnica de transmisión de ráfaga para llenar los 16 caracteres en el interior de la FIFO de transmisión SCI, con esto se consigue reducir la cantidad de servicios de interrupciones a una sola interrupción por frase.

Utilizando la unidad periférica SCI, y el manejo de interrupciones se establecerá la comunicación entre el procesador *TMS320F2812* y el puerto serial de la PC. La configuración para el uso de la unidad periférica SCI se la realiza por programa mediante el software Code Composer Studio. Para poder establecer la comunicación SCI se debe configurar también la trama y velocidad en el puerto COM de la PC y en el programa HyperTerminal.

Se cargará la FIFO de transmisión SCI con los caracteres de la frase “COMUNICANDO ! \n\r” para luego ser transmitidos mediante la interfaz RS232 y visualizados en la ventana principal del programa HyperTerminal cada 2 segundos. El retardo de tiempo se generará por hardware usando los timers internos del DSP. El LED conectado a GPIO B0 será el indicador de la transmisión de datos, es decir, cada vez que se esté transmitiendo caracteres del texto el LED estará prendido. Ver Figura 3.7

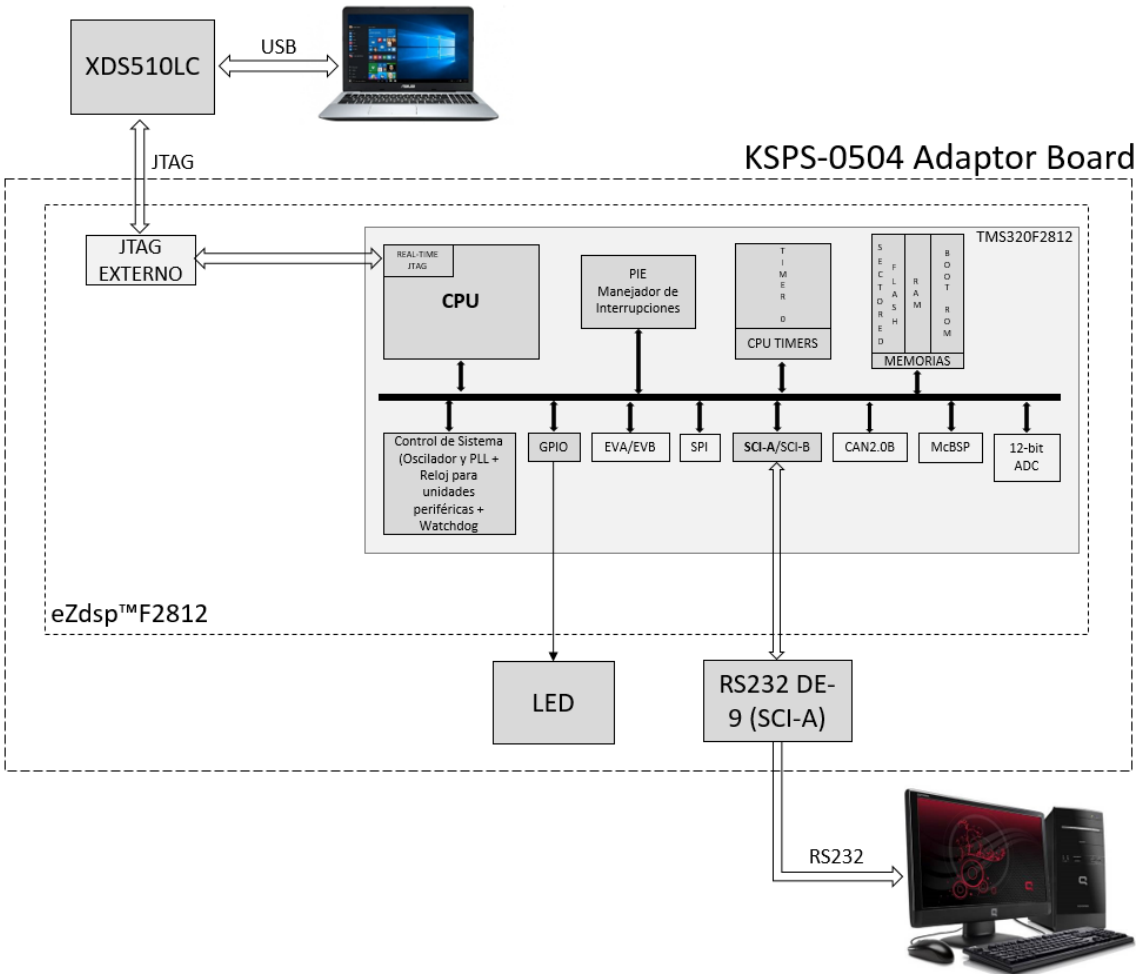


Figura 3.7 Diagrama de Bloques de Comunicación SCI entre DSP y PC usando Interrupciones y el método FIFO

3.8 Descripción de la Práctica # 8

"Transmisión y Recepción de datos mediante interfaz SCI entre el procesador TMS320F2812 y una PC"

3.8.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Configurar el formato de la trama, la velocidad, y habilitar el sistema de interrupciones para una comunicación SCI.
- Establecer comunicación SCI entre el procesador *TMS320F2812* y la PC.
- Establecer la velocidad del "Knight-Rider" (secuencia de encendido de leds) mediante la recepción de datos desde la PC.

3.8.2 Breve explicación de la practica

Utilizando la unidad periférica SCI, y el manejo de interrupciones se establecerá la comunicación entre el procesador *TMS320F2812* y el puerto serial de la PC. La configuración para el uso de la unidad periférica SCI se la realiza por programa mediante el software Code Composer Studio. Para poder establecer la comunicación SCI se debe configurar también la trama y velocidad en el puerto COM de la PC y en el programa HyperTerminal.

Cuando el botón S1 (Conectado a GPIO D1) es pulsado se cargará la FIFO de transmisión SCI con los caracteres de la frase "Ingrese tiempo: ", para luego ser transmitidos mediante la interfaz RS232 y visualizados en la ventana principal del programa HyperTerminal. Luego se transmitirá desde la PC al procesador *TMS320F2812* el tiempo el cual será almacenado en la unidad FIFO de recepción SCI. Cabe recalcar que el valor ingresado debe tener 5 caracteres en el orden de los milisegundos, en un rango de 0 a 9999 ms y que

el ultimo carácter debe ser un espacio para así poder validad el tiempo y modificar la velocidad de la secuencia de encendido de los leds (Conectados desde GPIO B0 a B8). Ver Figura 3.8

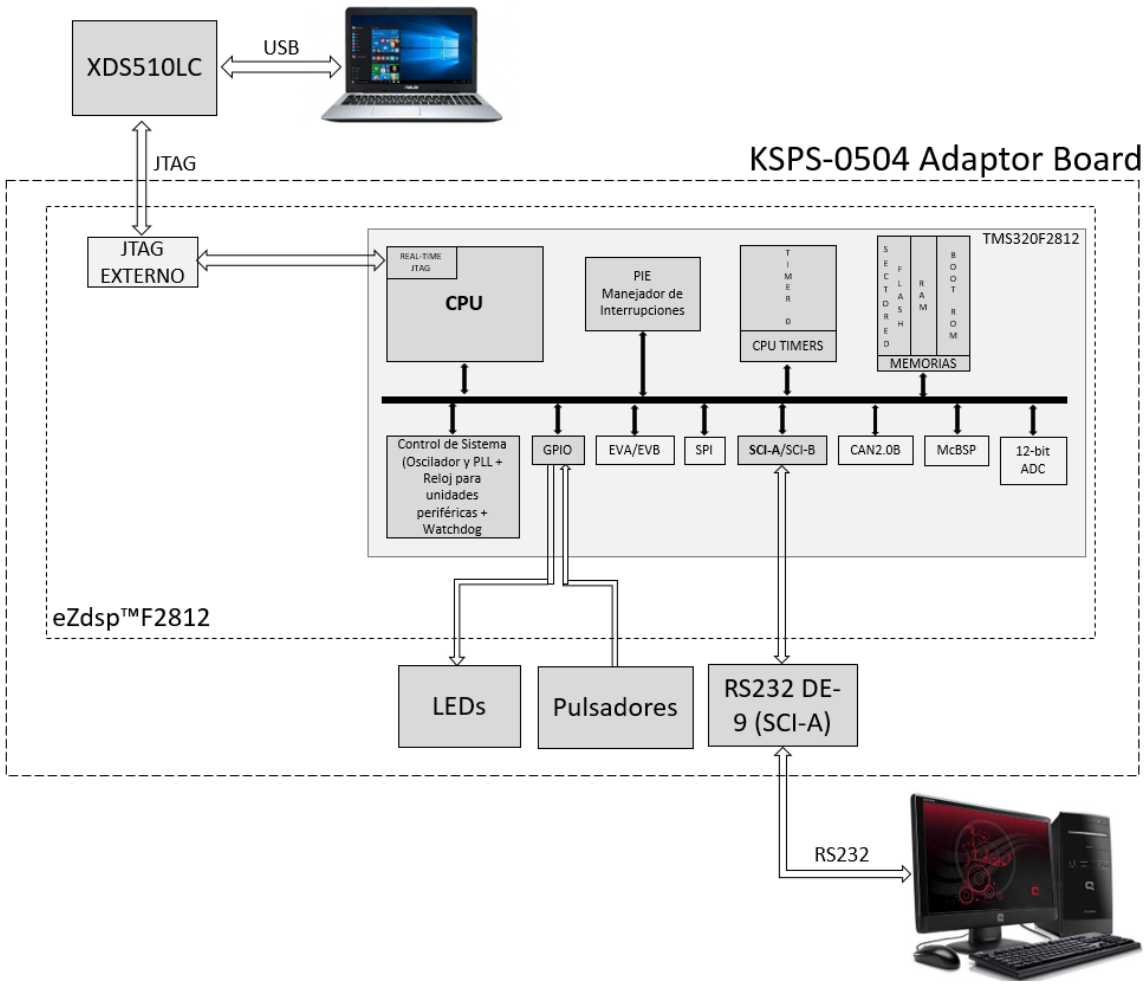


Figura 3.8 Diagrama de Bloques de Comunicación SCI entre DSP con PC y viceversa

3.9 Descripción de la Práctica # 9

"Transmisión de trama de datos desde el procesador TMS320F2812 usando el protocolo de comunicación CAN y el transceptor SN 65 HVD 230"

3.9.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Transmitir con el procesador *TMS320F2812* un byte de datos en la trama cada segundo mediante protocolo CAN.
- Configurar el formato de la trama, la velocidad y el Mailbox.
- Establecer comunicación CAN entre dos procesadores *TMS320F2812*.
- Usar "CPU Timer0" para generar un intervalo de un segundo.

3.9.2 Breve explicación de la practica

El procesador *TMS320F2812* embebe la unidad periférica de comunicación CAN, la cual es compatible con el estándar CAN2.0B. Su uso establece un protocolo para comunicarse de forma serial con otros procesadores en ambientes de permanente ruido eléctrico. Esta unidad periférica provee una robusta y versátil interfaz de comunicación serial.

El procesador (DSP) se configurará para transmitir una trama de un byte de datos, el cual contiene el estado de los switches de entrada (conectados a GPIOB15…GPIOB8), también se configura el uso del identificador extendido 0x1000 0000 para transmitir el mensaje, el uso del Mailbox#5 como Mailbox de transmisión, la velocidad de transmisión de 100Kbps. La trama se debe transmitir cada segundo. Ver Figura 3.9

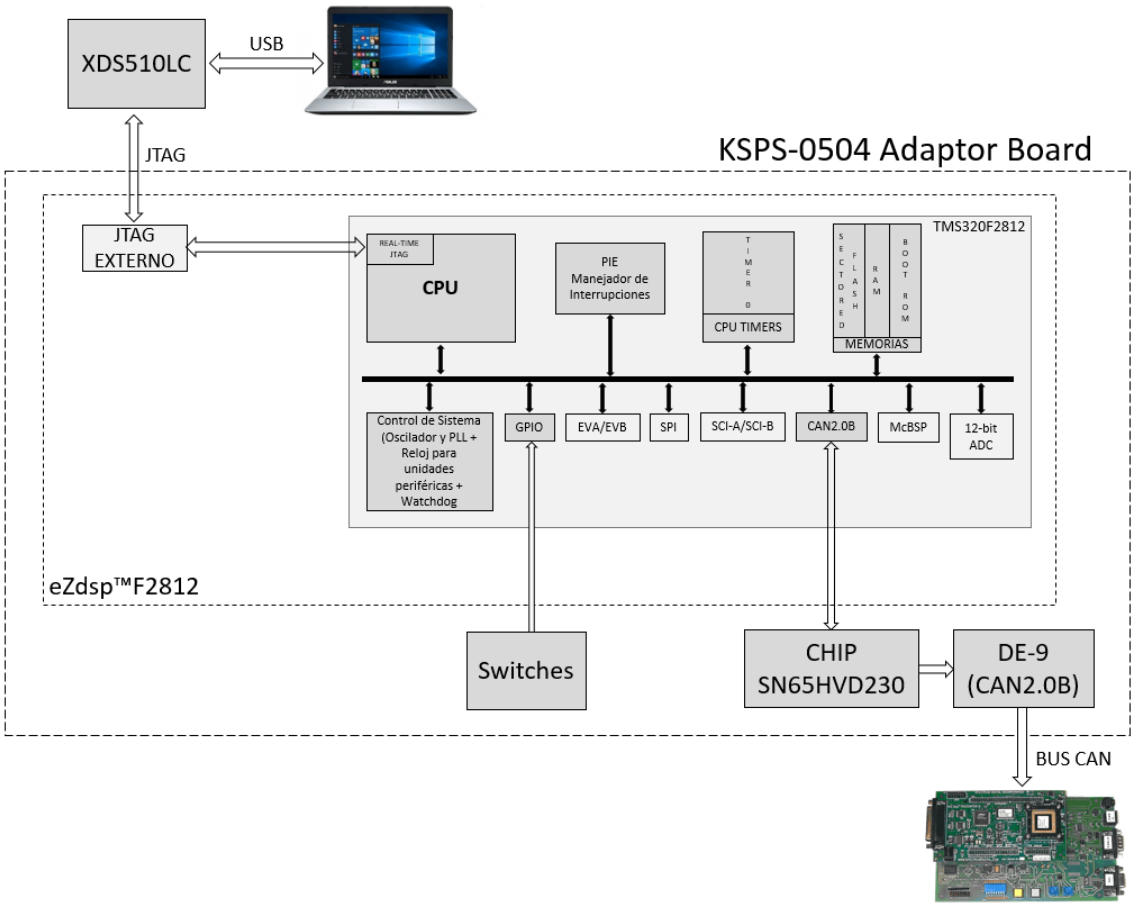


Figura 3.9 Diagrama de Bloques de Transmisión mediante protocolo CAN

3.10 Descripción de la Práctica # 10

"Recepción de trama de datos en el procesador TMS320F2812 usando el protocolo de comunicación CAN y el transceptor SN 65 HVD 230"

3.10.1 Objetivos:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Recibir con el procesador F2812 un byte de datos en la trama mediante protocolo CAN.
- Configurar el formato de la trama, la velocidad y el Mailbox.
- Establecer comunicación CAN entre dos procesadores TMS320F2812.

3.10.2 Breve explicación de la practica

El procesador *TMS320F2812* embebe la unidad periférica de comunicación CAN, la cual es compatible con el estándar CAN2.0B. Su uso establece un protocolo para comunicarse de forma serial con otros procesadores en ambientes de permanente ruido eléctrico. Esta unidad periférica provee una robusta y versátil interfaz de comunicación serial.

El procesador se configurará para recibir una trama de un byte de datos y mostrarlos en los leds de salida conectados a GPIOB15…GPIOB8, también se configura el uso del identificador extendido 0x1000 0000 para recibir el mensaje, el uso del Mailbox#1 como Mailbox de recepción, la velocidad de transmisión de 100Kbps. La trama se recibirá cada segundo. Ver Figura 3.10

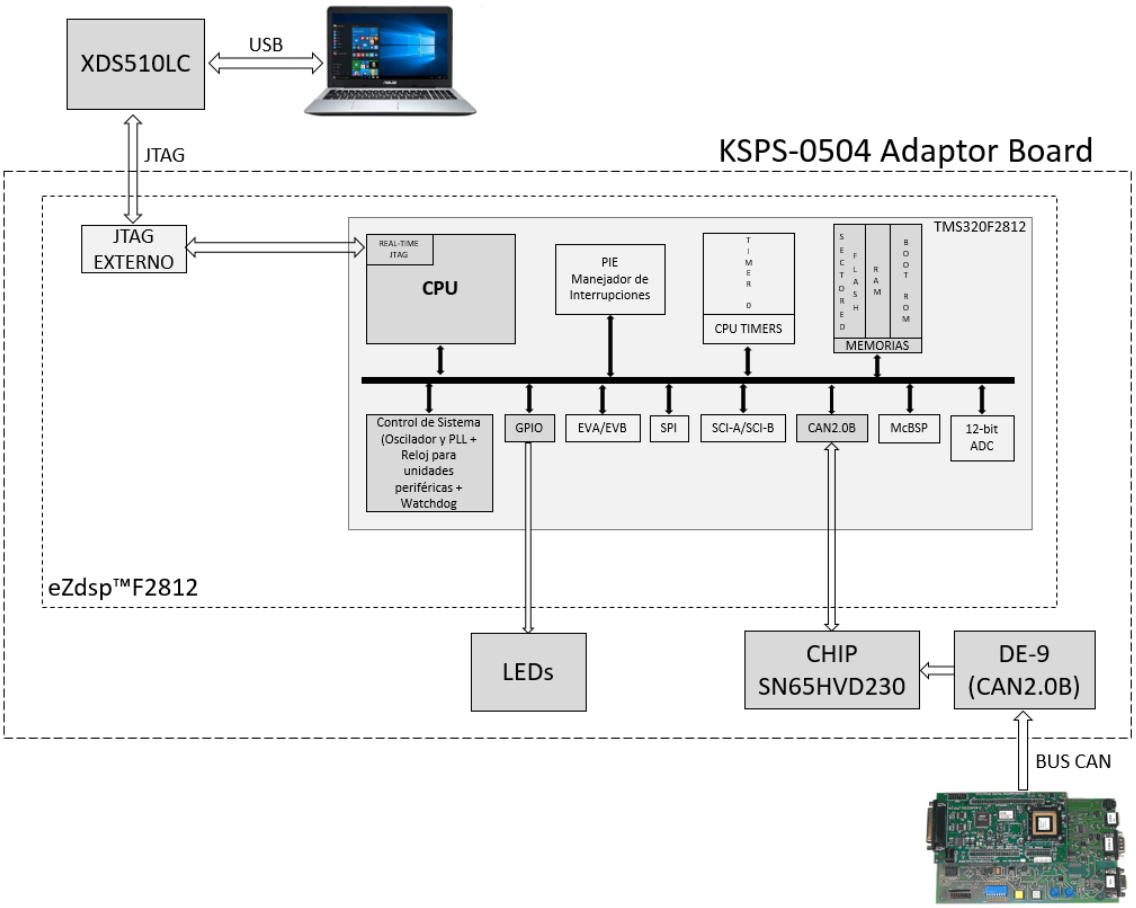


Figura 3.10 Diagrama de Bloques de Recepción mediante protocolo CAN

CONCLUSIONES Y RECOMENDACIONES

Los Procesadores Digitales de Señales cada vez tienen una mayor demanda en las Tecnologías de la información y las Comunicaciones, debido a la necesidad de requerir más recursos para procesar grandes volúmenes de datos. Gracias a sus cualidades de interacción con el mundo real, los DSP propiamente dichos tienen una gran variedad de aplicaciones, siendo la de mayor demanda su uso en sistemas embebidos, es decir en sistemas autónomos; por ejemplo, celulares, impresoras, cámaras fotográficas, antenas digitales y en controladores digitales.

Los retardos de tiempo en los Procesadores Digitales de Señales pueden ser generados por software o por hardware. Aunque a simple vista el resultado no difiere entre ambos métodos, existe una gran diferencia. El intervalo de tiempo generado por Hardware es muy preciso con respecto al tiempo, y el DSP no se sobrecarga de conteos de una variable tantas veces.

En el módulo de reloj se pueden habilitar todos los relojes de las unidades periféricas y el proyecto funcionaria sin ningún problema, sin embargo, se recomienda que, si una unidad periférica no será usada, entonces el reloj de dicha unidad periférica debe ser deshabilitado para minimizar el consumo de potencia y calentamiento del Procesador Digital de Señal.

La interfaz SPI es un medio serial síncrono que permite el intercambio de flujo serial de bits a una alta velocidad. El campo de aplicación de la interfaz SPI abarca comunicaciones a velocidades muy altas entre DSP y periféricos externos, para intercambio de información en tiempo real.

La Interfaz de comunicación serial (SCI) es un medio serial asincrónico, mientras la interfaz SPI es un medio serial síncrono, Pero la diferencia entre las dos interfaces no solo es el hecho de que una es sincrónica y la otra asincrónica, sino también que la velocidad de transmisión de datos es mucho mayor en la interfaz SPI, lo que permite que el intercambio de información sea en tiempo crítico.

La interfaz de comunicación eCAN se constituyó como la unidad periférica de comunicación más robusta y confiable del procesador F2812 cuando se trata de llevar a cabo transferencia y recepción de mensajes ya que implementa mecanismos

sofisticados de detección de errores en los diferentes tipos de tramas, pudiendo inclusive autocorregir el envío/transmisión de mensaje, además contiene un manejador de acceso al bus de datos que impide colisiones de mensajes y brinda seguridad a aquel de mayor prioridad.

Durante la configuración de la transmisión mediante el protocolo CAN se debe especificar el tamaño de los datos a transmitir, sin embargo, en la configuración de recepción no es necesario decir el tamaño de los datos a recibir.

BIBLIOGRAFÍA

[1]http://cegt201.bradley.edu/projects/proj2011/pkohnen/Resources/28857722-F2812-DSP-Full-Tutorial

[2] http://www.aliexpress.com/item-img/Free-shipping-5pcs-lot-TMS320F2812PGFA-TMS320F2812-DSP-signal-device-new-original/32688290745.html?spm=2114.10010108.1000017.2.dqN0vV

[3] http://www.ti.com/lit/ds/symlink/tms320f2812.pdf

[4] http://c2000.spectrumdigital.com/ezf2812/docs/ezf2812_techref.pdf

[5] http://www.ti.com/lit/ug/spru078g/spru078g.pdf

[6] http://www.ti.com/lit/ug/spru059e/spru059e.pdf

[7] http://www.ti.com/lit/ds/symlink/tlv5617a.pdf

[8]http://www.st.com/content/ccc/resource/technical/document/datasheet/28/42/21/c1/13/bf/47/9a/DM00043274.pdf/files/DM00043274.pdf/jcr:content/translations/en.DM00043274.pdf

[9] http://www.ti.com/lit/ug/spru051d/spru051d.pdf

[10] http://pcexpertos.com/2010/01/conector-serial-com-1-o-conector-db9.html

[11] http://www.picprojects.net/rs232_communication/

[12] https://es.wikipedia.org/wiki/C%C3%B3digos_NRZ

[13] http://www.ti.com/lit/ug/spru074f/spru074f.pdf

[14]http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/moduleComponents/CanBus_canover.pdf

[15] http://ecee.colorado.edu/~mcclurel/CANPRES.pdf

[16] http://www.ti.com/lit/an/slla270/slla270.pdf

ANEXO 1. GUÍA DE PRÁCTICAS SOBRE TRANSMISIÓN Y RECEPCIÓN DE DATOS CON EL PROCESADOR DIGITAL DE SEÑALES

PRÁCTICA # 1

TEMA:

"Introducción a la herramienta de desarrollo de programa Code Composer Studio mediante la creación de un proyecto para el manejo de entradas y salidas digitales del DSP *TMS320F2812*"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Conectar el DSP *TMS320F2812* con el programa *Code Composer Studio*.
- Familiarizar al estudiante con la herramienta de desarrollo *Code Composer Studio* y su programación.
- Crear, compilar un proyecto en *Code Composer Studio.*
- Aprender a configurar el módulo de reloj y el perro guardián del procesador.
- Aprender a configurar los pines del procesador para que sean usados como entradas o salidas digitales.

REQUISITOS MÍNIMOS:

- Tener Instalado el software Code Composer Studio (versión 3.3) con su respectivo drive para el manejo del convertidor JTAG a USB.
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos de programación en Lenguaje C.
- Lectura y comprensión del archivo de fundamentación teórica: "CONFIGURACIONES DEL CONTROL DE SISTEMA DEL DSP TMS320F2812"
- Lectura y comprensión del archivo de fundamentación teórica: "MANEJO DE ENTRADAS Y SALIDAS DIGITALES DEL DSP TMS320F2812".

BREVE EXPLICACIÓN DE LA PRACTICA

En el Procesador Digital de Señales (DSP) no solo tenemos un núcleo de 32-bit, sino también algunas unidades periféricas necesarias para poder resolver varios problemas en distintos tipos de aplicaciones. Entre las unidades periféricas tenemos el módulo de entrada y salidas digitales.

Lo primero que se debe configurar en los DSPs es el módulo de reloj. El *TMS320F2812* es temporizado por un oscilador externo por los pines *X1* y *X2*, este oscilador es más lento para así reducir el ruido electromagnético. Se debe programar el circuito interno PLL del módulo de reloj para que genere una velocidad interna de 150 MHz al CPU del procesador a partir del oscilador externo de 30 MHz.

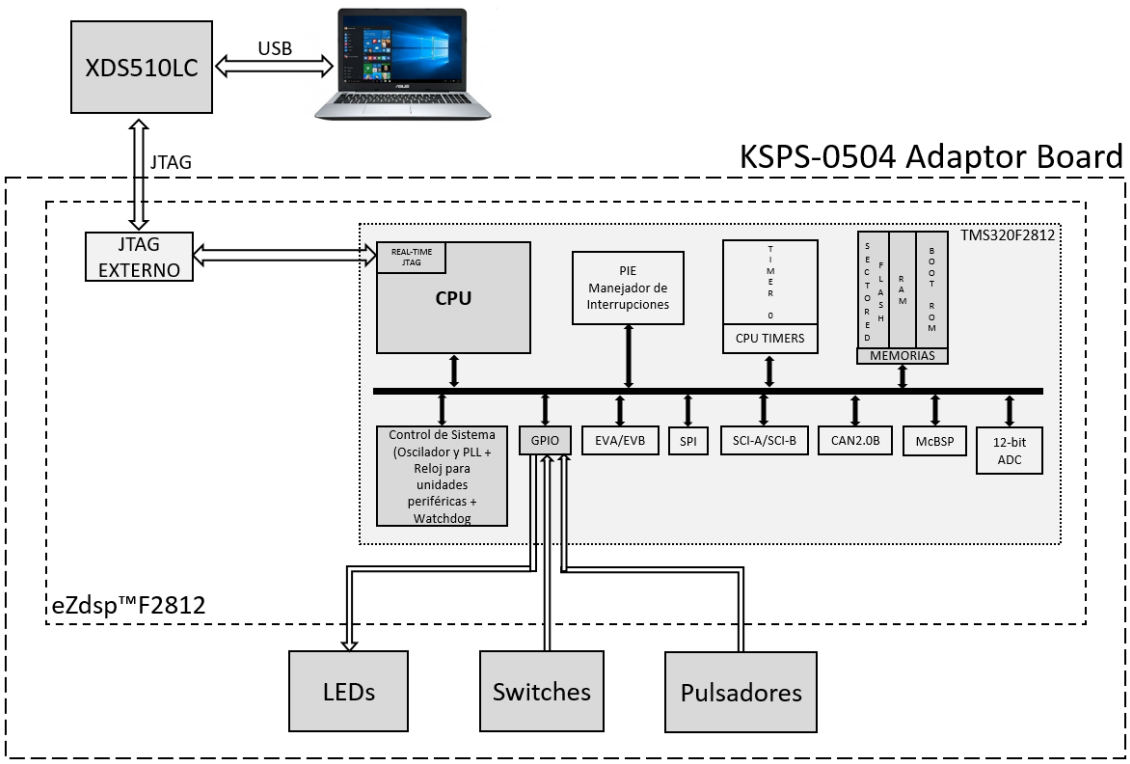


Figura 4.1 Diagrama de Bloques del manejo de entradas y salidas digitales

Luego se debe configurar los pines del procesador para que funciones como entradas y salidas digitales, según las conexiones entre la tarjeta *eZdsp™F2812* y *KSPS-0504 Adaptor Board*.

Finalmente se debe programar que cuando se presione el pulsador S1 (Conectado a GPIOD1), los estados de los switches (Conectados desde GPIO B15 a B8) se los visualizará en los leds (Conectados desde GPIO B7 a B0). Por otro lado, cuando se presione el pulsador S2 (Conectado a GPIOD6), se iniciará una secuencia de encendido de los leds (Conectados desde GPIO B7 a B0), en el cual el encendido se desplaza entre led y led. Ver Figura 4.1

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Setup Code Composer Studio V3.3*, dando doble clic en el icono igual al de la Figura 4.2

Figura 4.2 Icono del programa "Setup CCStudio V3.3"

3. En la ventana del programa *Setup Code Composer Studio V3.3* (Figura 4.3), se procederá a realizar la configuración para conectarse con el Hardware de la tarjeta.

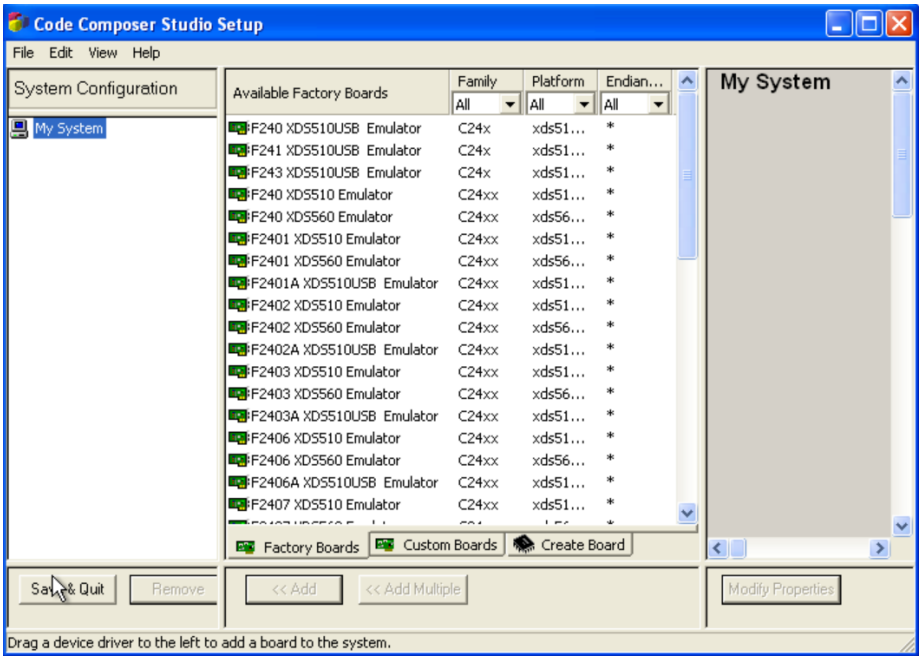


Figura 4.3 Ventana principal del programa "Setup CCStudio"

4. Filtrar todas las tarjetas disponibles en base a las siguientes características:

- Family: C28xx
- Platform: xds510usb emulator

Finalmente, de la lista de tarjetas se selecciona la “F2812” tal como se observa en la Figura 4.4

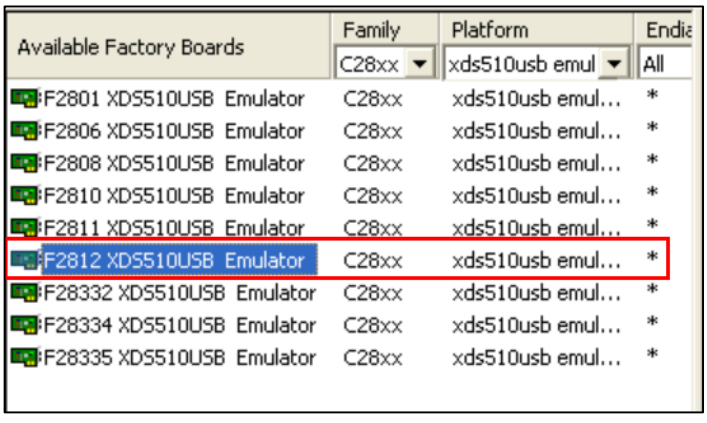


Figura 4.4 Selección de tarjeta F2812 en el programa "Setup CCStudio"

5. Dar clic en **<<Add.** Luego de haber agregado la tarjeta, en la configuración del sistema del programa aparecerá la tarjeta con su respectivo DSP, tal como se observa en la Figura 4.5

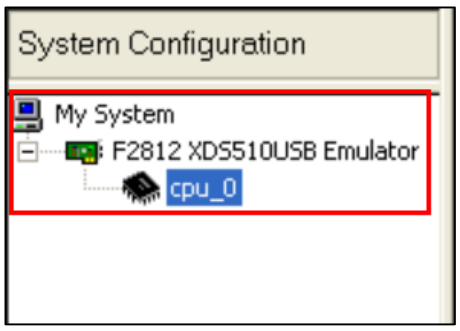


Figura 4.5 Configuración del sistema hecha con tarjeta F2812

6. Dar clic en el botón **Save & Quit**, para guardar los cambios y cerrar el programa. Se desplegará una ventana igual a la Figura 4.6 en la cual se dará clic en **Yes** para abrir el programa *Code Composer Studio*.

Figura 4.6 Ventada de confirmación para apertura de programa "CCStudio"

7. En la Figura 4.7 se puede observar la ventana principal del programa *Code Composer Studio*.

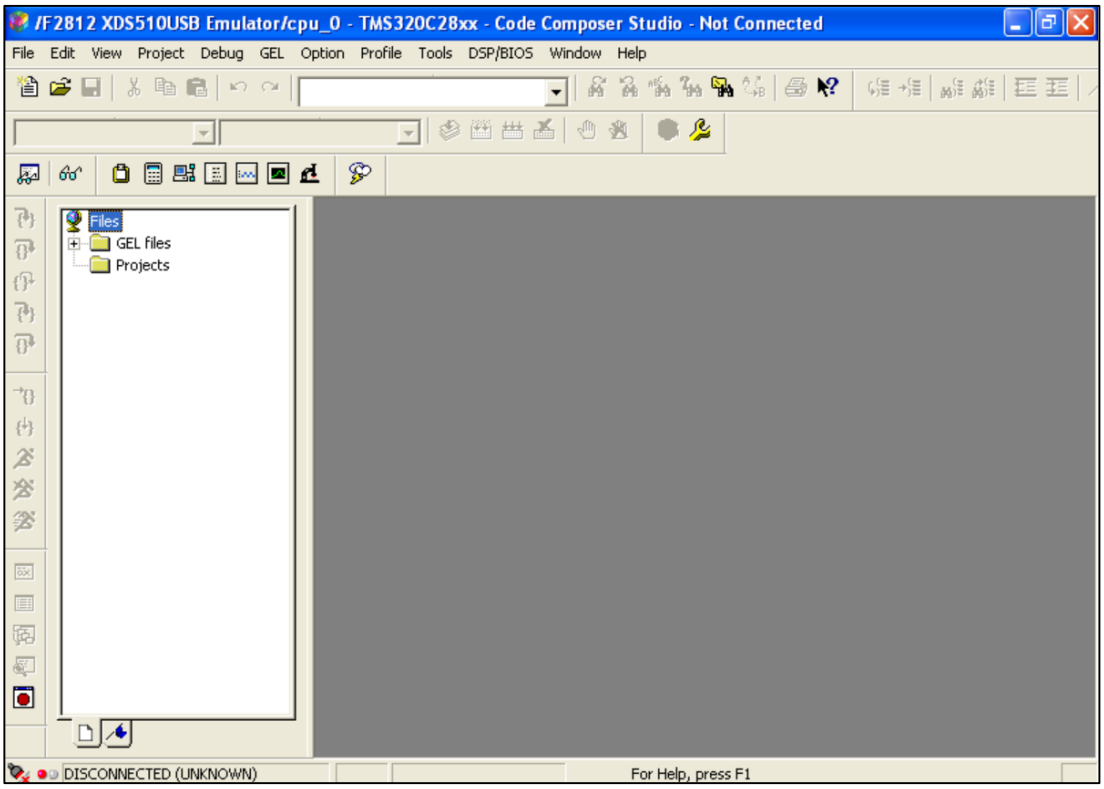


Figura 4.7 Ventana principal del programa "Code Composer Studio" (CCStudio)

8. Crear un proyecto nuevo (*Project -> New...)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica1
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** de la ventana para crear el proyecto. Ver Figura 4.8

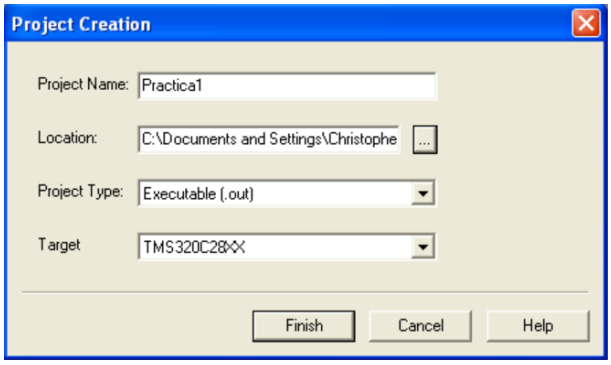


Figura 4.8 Ventana para nuevo proyecto en "CCStudio"

9. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

// Prototipos de funciones

void main(void)
{
        while(1)
        {
        }
}
```

10. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica1.c"***

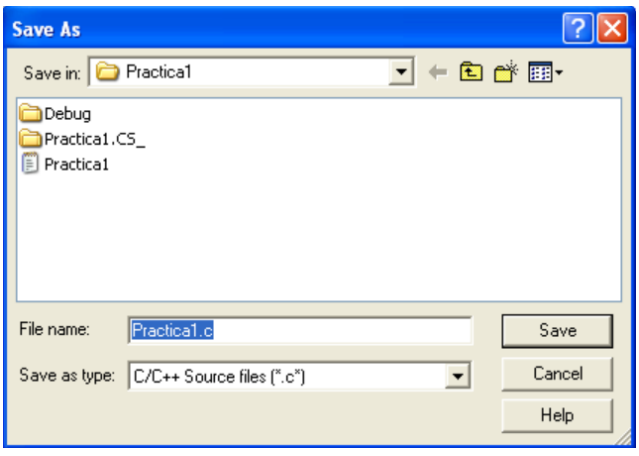


Figura 4.9 Ventana para guardar archivo de código fuente

11. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica1.c"***. Ver Figura 4.10

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

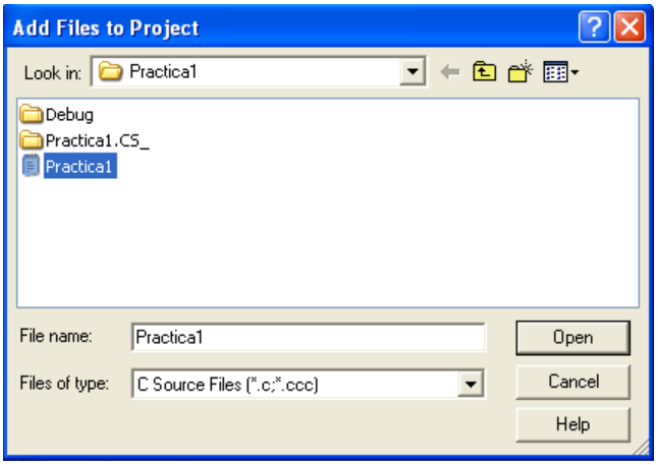


Figura 4.10 Ventana para agregar archivos al proyecto

12. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **DSP281x_GlobalVariableDefs.c**

13. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **F2812_Headers_nonBIOS.cmd**

14. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

15. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL “BUILD OPTIONS”

16. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options...*
- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include
  en el campo *Include Search Path*. Ver Figura 4.11

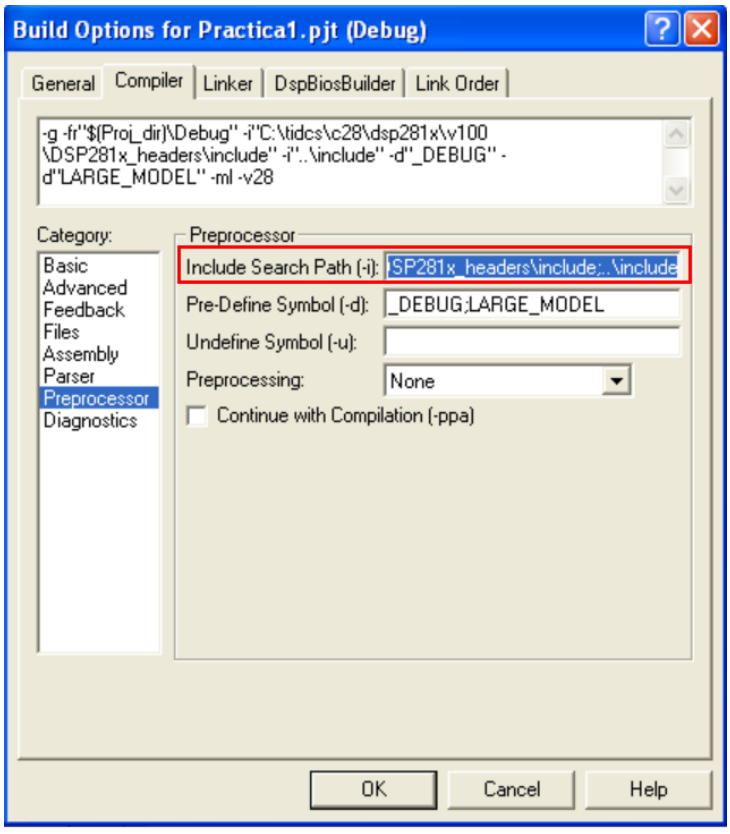


Figura 4.11 Configuración de la ruta de búsqueda de los archivos de encabezado

17. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **"400"** en el campo *Stack Size.* Ver Figura 4.12
- Dar clic en el botón **OK** para terminar la configuración del *Build Options*

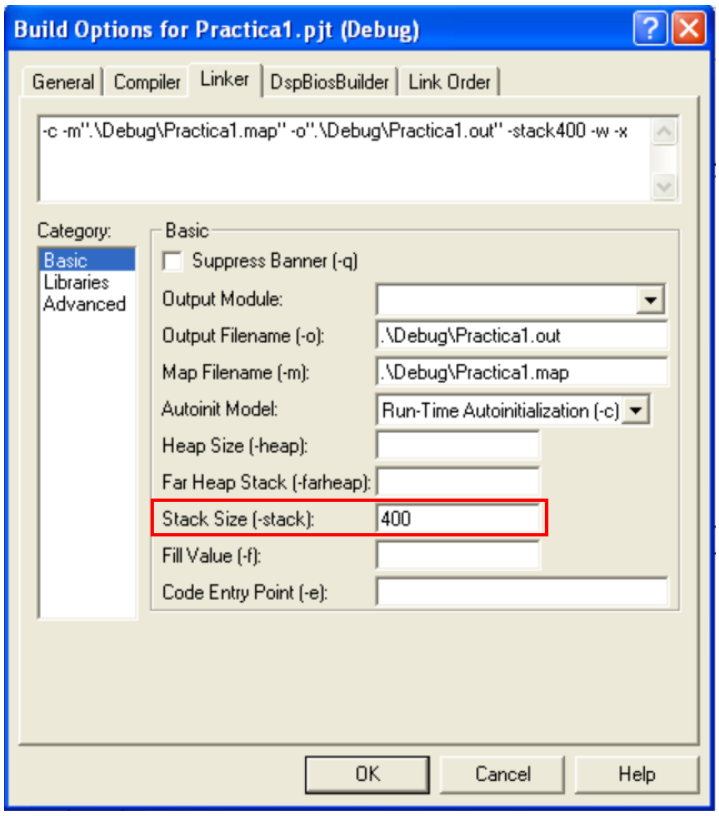


Figura 4.12 Configuración del tamaño de la pila

MODIFICAR EL ARCHIVO DE CÓDIGO FUENTE

18. Al inicio del archivo de código fuente ***"Practica1"***, antes del "main" agregar el prototipo de función **"void InitSystem(void)"**, como se observa en la Figura 4.13

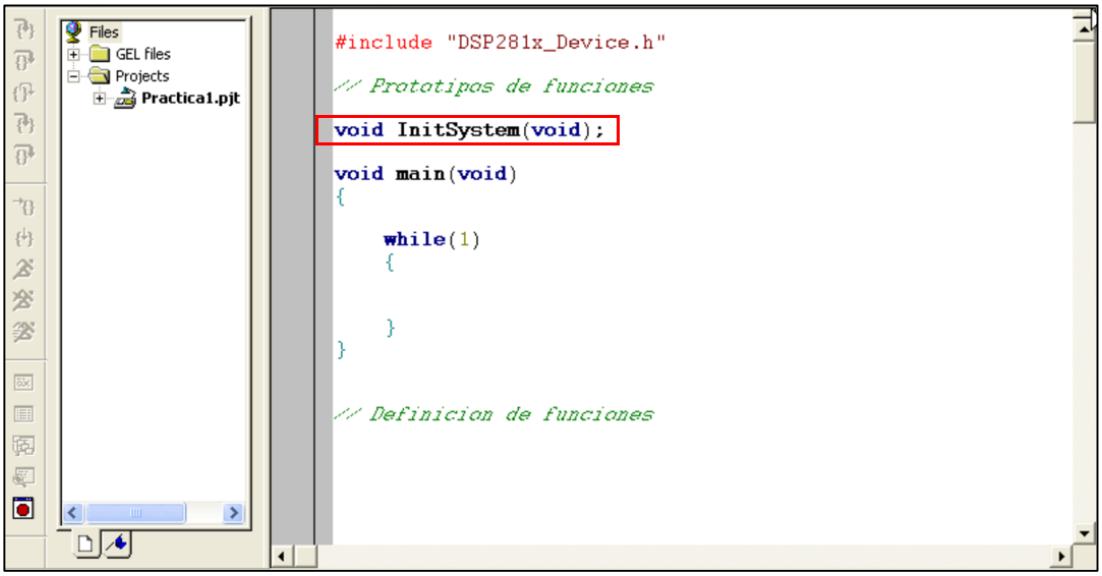


Figura 4.13 Prototipo de función "InitSystem( )"

19. Al final del archivo de código fuente ***"Practica1"***, agregar la definición de la función **"void InitSystem(void)"**, en base a lo siguiente:

Para el registro WDCR:

- Habilitar el módulo Watchdog
- Limpiar la bandera del WD para que pueda causar un Reseteo del sistema
- Colocar los correctos bits de lógica de verificación para que no se genere un Reseteo inmediatamente.
- Escoger el pre escalador de 64

Para el registro SCSR:

- Configurar el bit "WDENINT" para habilitar que el módulo Watchdog pueda generar una interrupción (Reseteo del sistema).

Para el registro PLLCR:

- Configurar los bits “DIV” para determinar la frecuencia de 150 Mhz en el CPU del procesador *TMS320F2812*.

Para el registro HISPCP:

- Configurar una frecuencia de 75 Mhz en el reloj adicional HSPCLK

Para el registro LOSPCP:

- Configurar una frecuencia de 37.5 Mhz en el reloj adicional LSPCLK

Para el registro PCLKCR:

- Deshabilitar todos los relojes de las unidades periféricas.

En la Figura 4.14 se muestra la programación de los registros para configurarlos según lo mencionado anteriormente en la definición de la función “InitSystem( )”.

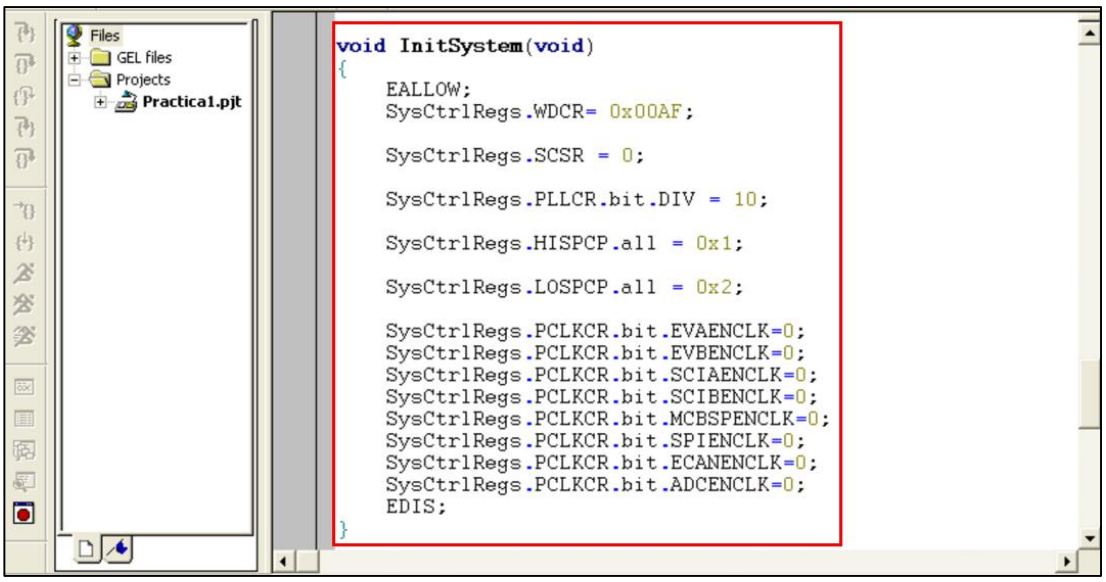


Figura 4.14 Definición de función "InitSystem( )"

20. Al inicio del archivo de código fuente ***"Practica1"***, antes del "main" agregar el prototipo de función **"void Gpio_select(void)"**, como se observa en la Figura 4.15

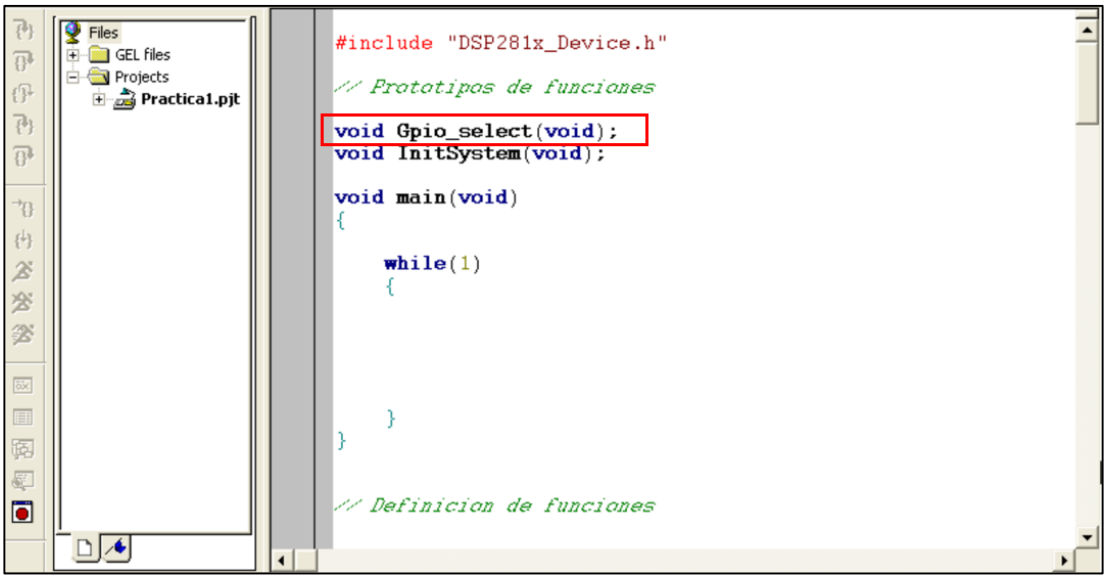


Figura 4.15 Prototipo de función "Gpio_select( )"

21. Al final del archivo de código fuente ***"Practica1"***, agregar la definición de la función **"void Gpio_select(void)"**, en base a lo siguiente:

Para el registro GPxMUX:

- Configurar todos los puertos como entradas y salidas digitales

Para el registro GPxDIR:

- Configurar los pines de los puertos A, D, E, F y G como entradas.
- Configurar los pines B15 a B8 del puerto B como entradas, y los pines B7 a B0 como salidas.

Para el registro GPxDAT:

- Ajustar en '0' todos los bits del puerto B, para apagar los leds al iniciar el programa.

En la Figura 4.16 se muestra la programación de los registros para configurarlos según lo mencionado anteriormente en la definición de la función “Gpio_select( )”.

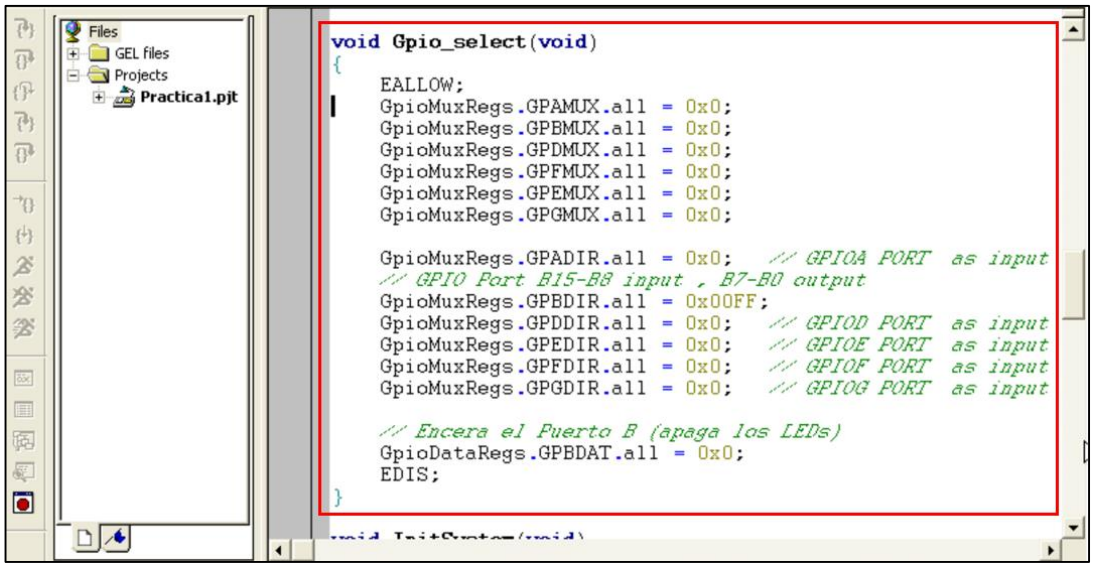


Figura 4.16 Definición de función "Gpio_select( )"

22. Al inicio del archivo de código fuente agregar el prototipo de función **“void delay_loop(long end)”** y al final definir la función en base a los siguiente:

- Generar un retardo mediante un lazo “for” que se repite “end” veces
- Dentro del lazo “for”, escribir la secuencia de clave para resetear periódicamente el módulo Watchdog.

El objetivo de esta función es generar un retardo por software para poder apreciar el desplazamiento de los leds. Ver Figura 4.17

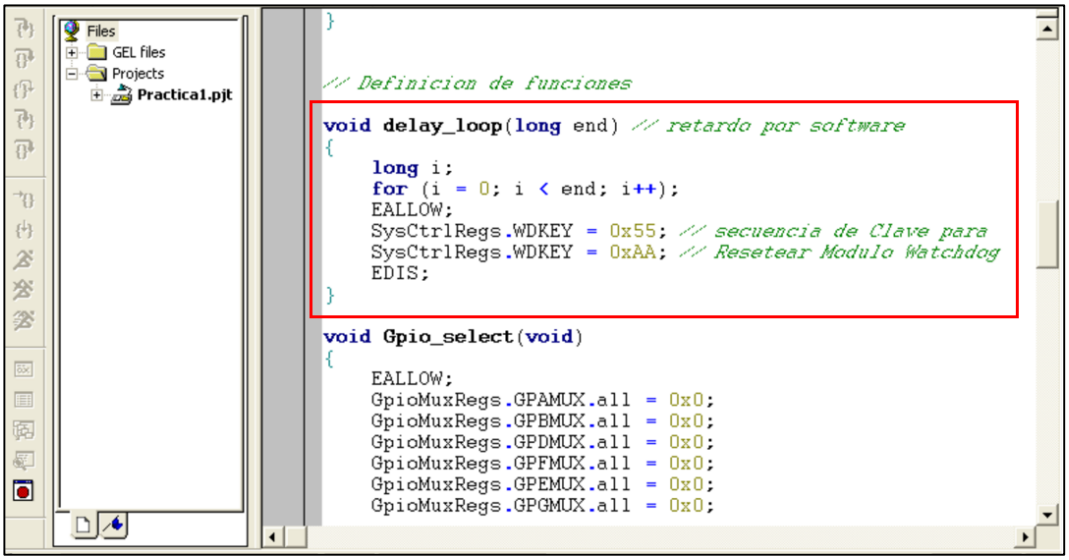


Figura 4.17 Definición de función "delay_loop( )"

PROGRAMACIÓN DEL “MAIN”

23. Al inicio del “main”, declarar e inicializar las variables a usar en la práctica. Ver Figura 4.18

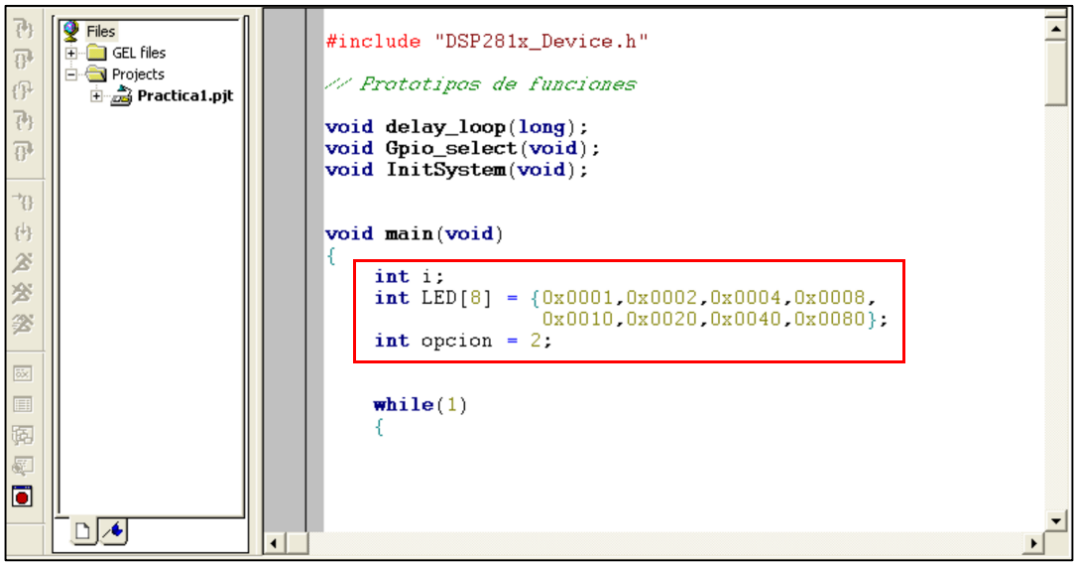


Figura 4.18 Declaración de Variables

24. En el “main”, antes de ingresar en lazo “while(1)”, llamar a las funciones **“InitSystem()”** y **“Gpio_select()”**, para inicializar el módulo de reloj y el Watchdog y configurar los pines del DSP, tal como se observa en la Figura 4.19

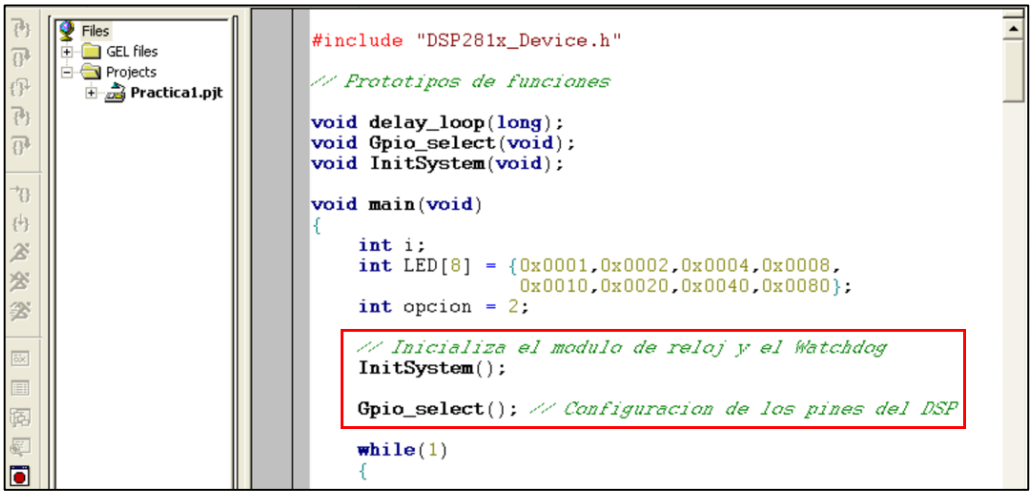


Figura 4.19 Llamado de funciones "InitSystem( )" y "Gpio_select( )"

25. Dentro del “while(1)”. Cuando es pulsado el botón S1 (GPIOD1 = 0), el estado de los switches (Conectados en GPIO B15 a B8) se pasarán a los leds (Conectados en GPIO B7 a B0). Ver Figura 4.20

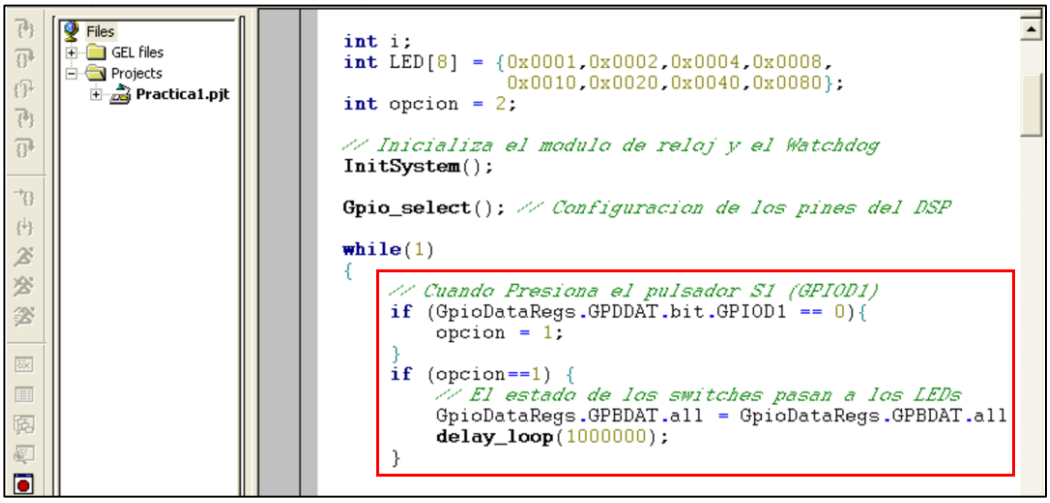


Figura 4.20 Programación del "while(1)"cuando presiona S1

26. Dentro del “while(1)”. Cuando es pulsado el botón S2 (GPIOD6 = 0), se establece una secuencia de encendido de los leds (Conectados en GPIO B7 a B0) en forma de desplazamiento. Ver Figura 4.21

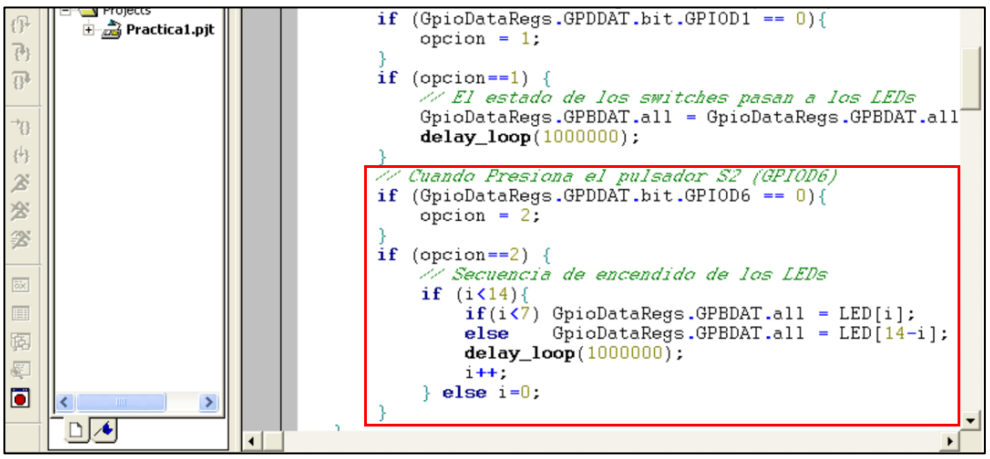


Figura 4.21 Programación del "while(1)" cuando presiona S2

COMPILAR, CARGAR Y EJECUTAR EL PROGRAMA

27. Compilar el proyecto seleccionando*: Project -> Rebuild All*. En la Figura 4.22 se observa que la compilación fue exitosa, pero en caso de haber errores se deben solucionar

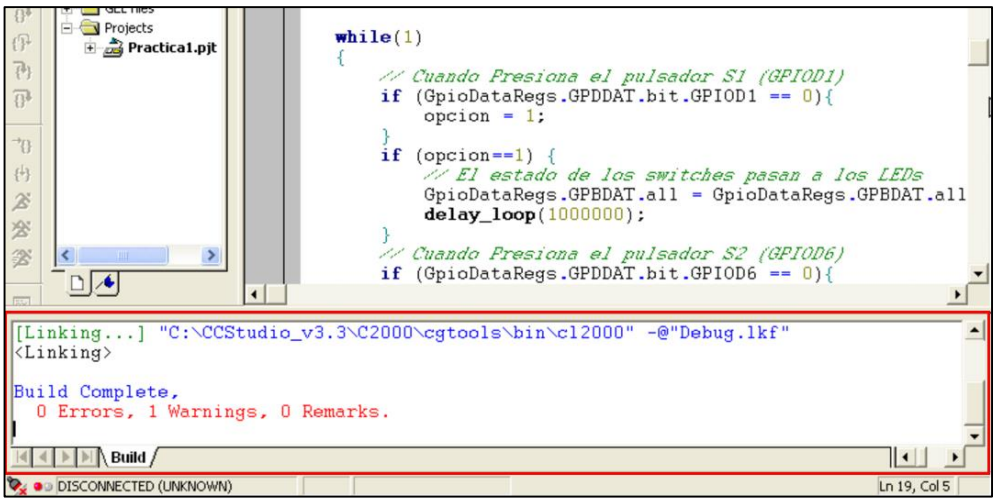


Figura 4.22 Compilación exitosa de la práctica

28. alimentar la tarjeta *eZdsp™F2812*. Luego conectarse con el procesador: *Debug -> Connect*

29. Cargar el programa (*File -> Load Program…)* **Practica1.out** que se genera dentro de la carpeta Debug al compilar el proyecto. Ver Figura 4.23

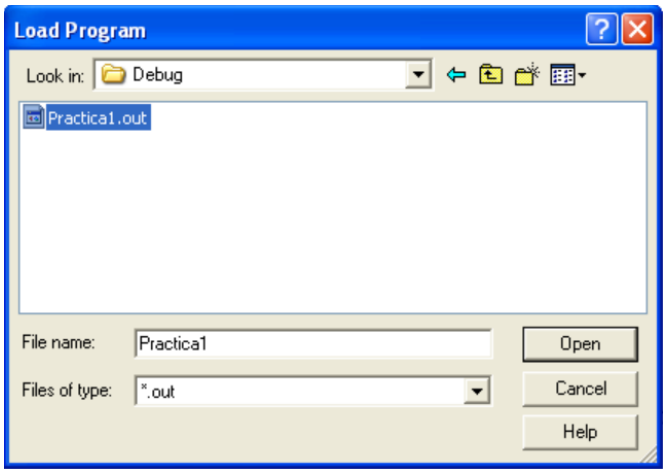


Figura 4.23 Ventana para cargar el programa al DSP

30. Ejecutar el programa cargado: *Debug -> Run*

Cuando se presiona el pulsador S1 (Conectado a GPIOD1), el estado de los switches se lo visualizará en los leds, tal como se observa en la Figura 4.24

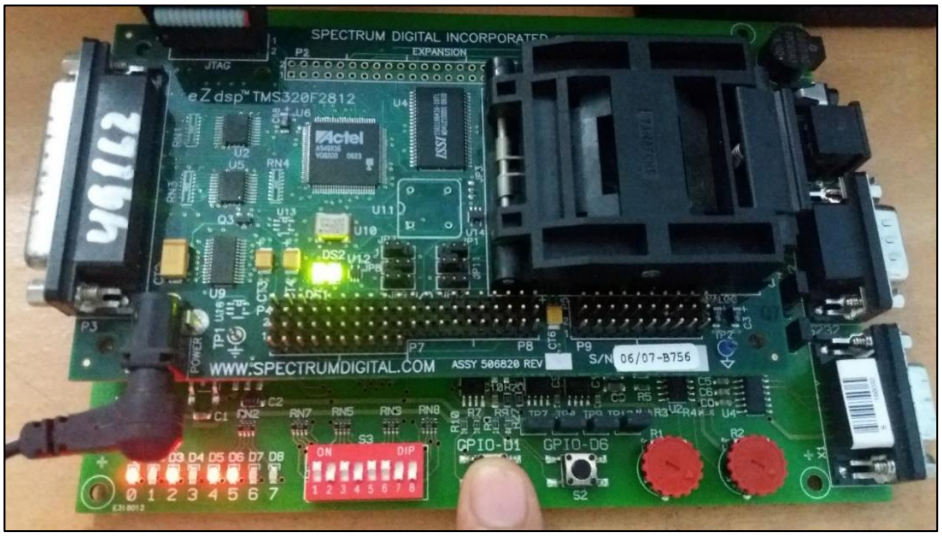


Figura 4.24 Foto del Kit cuando se presiona S1

Cuando se presiona el pulsador S2 (Conectado a GPIOD6), se iniciará una secuencia de encendido de los leds, en el cual el encendido se desplaza entre led y led, tal como se observa en la Figura 4.25

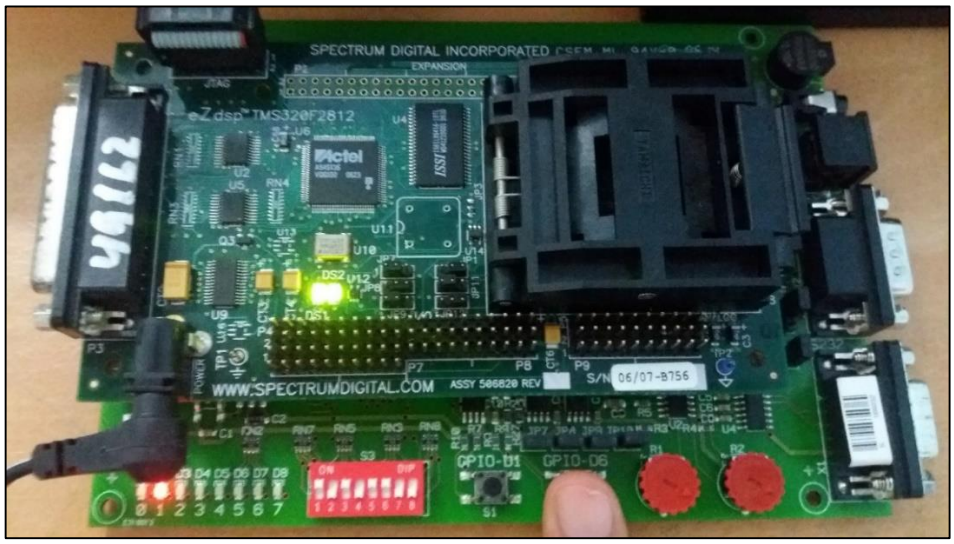


Figura 4.25 Foto del Kit cuando se presiona S2

31. Al finalizar la práctica, detener la ejecución del programa: *(Debug -> Halt)*

32. Resetear el CPU *(Debug -> Reset CPU)* para finalmente desconectarse de la tarjeta *(Debug -> Disconnect)*

33. Quitar la alimentación de la tarjeta *eZdsp™F2812*.

PRÁCTICA # 2

TEMA:

"Manejo del tiempo en el desplazamiento del encendido de los leds usando las interrupciones de un temporizador interno (Timer0) del DSP *TMS320F2812*"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Habilitar y configurar el sistema de interrupciones para poder usar cualquier fuente de interrupción de las unidades periféricas del procesador *TMS320F2812*.
- Aprender a usar el "Timer0" interno del procesador para generar interrupciones cada 50 milisegundos.
- Diferenciar las ventajas entre el uso de retardos por hardware que usando retardos por software.

REQUISITOS MÍNIMOS:

- Tener Instalado el software Code Composer Studio (versión 3.3) con su respectivo drive para el manejo del convertidor JTAG a USB.
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos de programación en Lenguaje C.
- Haber culminado satisfactoriamente la Práctica # 1.
- Lectura y comprensión del archivo de fundamentación teórica: "SISTEMAS DE INTERRUPCIONES DEL DSP TMS320F2812"

BREVE EXPLICACIÓN DE LA PRACTICA

Las interrupciones permiten detectar eventos en cualquier momento sin importar en que sección de código se encuentre. Debido a esto el uso de interrupciones es de importancia para el correcto manejo de las unidades periféricas de los DSP como lo son: el manejador de eventos, convertidor analógico a digital, temporizadores internos, SPI, SCI, CAN, entre otros.

Se debe programar que cuando se presione el pulsador S1 (Conectado a GPIOD1), arranque la secuencia de desplazamiento de los leds (Conectados desde GPIO B7 a B0). Por otro lado, cuando se presione el pulsador S2 (Conectado a GPIOD6), se detendrá la secuencia de desplazamiento de los leds (Conectados desde GPIO B7 a B0). Los pulsadores S1 y S2 funcionan como arranque (Start) y parada (Stop) respectivamente. Ver Figura 4.26

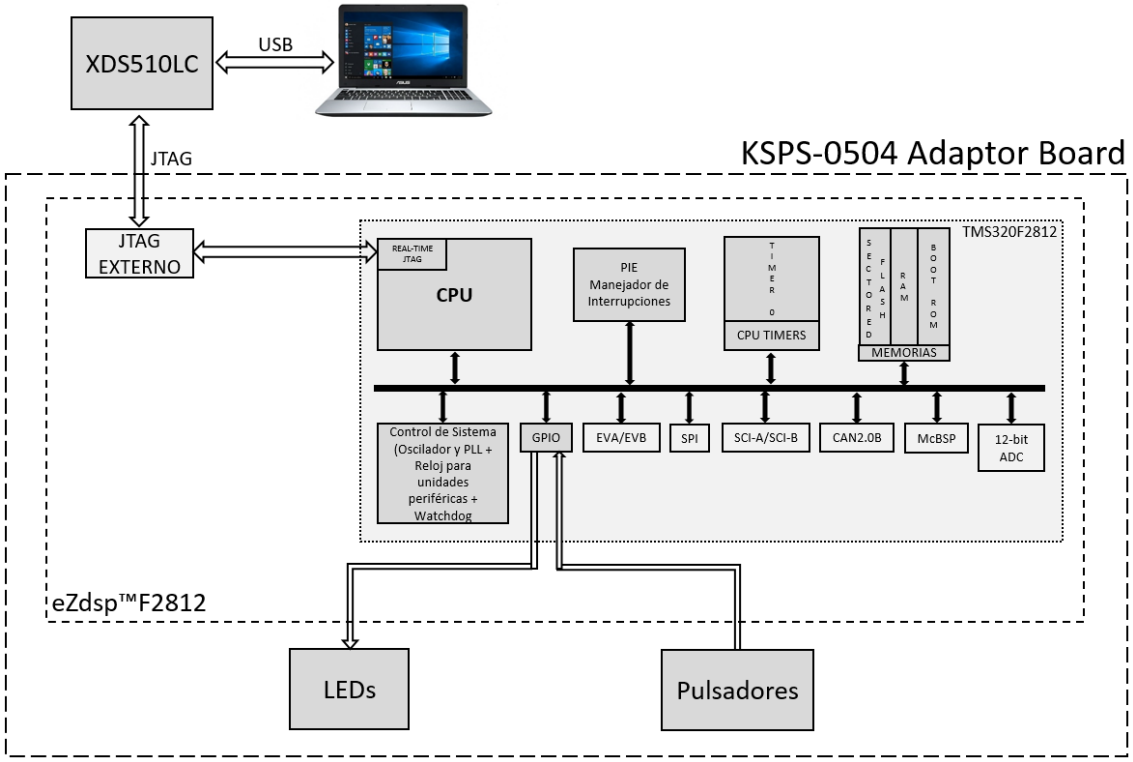


Figura 4.26 Diagrama de Bloques para el uso de Interrupciones

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Code Composer Studio*.

NOTA: Si al abrir el programa sale algún error de conexión con la PC, el motivo podría ser que no esté configurado el *Setup CCStudio V3.3* tal como se ve en la practica 1. Si el estudiante no ha realizado la *practica 1* se recomienda desarrollarla para evitar tener problemas de comunicación entre su PC y las tarjetas, para luego continuar con esta práctica.

3. Crear un proyecto nuevo (*Project -> New...)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica2
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** para crear el proyecto

4. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

// Prototipos de funciones
void Gpio_select(void);
void InitSystem(void);

void main(void)
{
        InitSystem();
```

```
        Gpio_select();

        while(1)
        {
        }
}

// Definicion de funciones

void Gpio_select(void)
{
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;
        GpioMuxRegs.GPBMUX.all = 0x0;
        GpioMuxRegs.GPDMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;

        GpioMuxRegs.GPADIR.all = 0x0;
        GpioMuxRegs.GPBDIR.all = 0x00FF;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
        GpioMuxRegs.GPFDIR.all = 0x0;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioDataRegs.GPBDAT.all = 0x0;
        EDIS;
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00E8;

        SysCtrlRegs.SCSR = 0;

        SysCtrlRegs.PLLCR.bit.DIV = 10;

        SysCtrlRegs.HISPCP.all = 0x1;
        SysCtrlRegs.LOSPCP.all = 0x2;

        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
}
```

5. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica2.c"***

6. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica2.c".***

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

7. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **DSP281x_GlobalVariableDefs.c**

8. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **F2812_Headers_nonBIOS.cmd**

9. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

10. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\source"* agregar al proyector los siguientes archivos:

- **DSP281x_PieCtrl.c**
- **DSP281x_PieVect.c**
- **DSP281x_DefaultIsr.c**
- **DSP281x_CpuTimers.c**

11. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL “BUILD OPTIONS”

12. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options…*
- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include
  en el campo Include Search Path.

13. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **“400”** en el campo *Stack Size.*
- Dar clic en el botón **OK** para terminar la configuración del *Build Options*

MODIFICAR EL ARCHIVO DE CÓDIGO FUENTE

14. Al inicio del archivo de código fuente ***“Practica2”***, antes del “main*”* agregar el prototipo de función **“interrupt void cpu_timer0_isr(void)”**, como se observa en la Figura 4.27

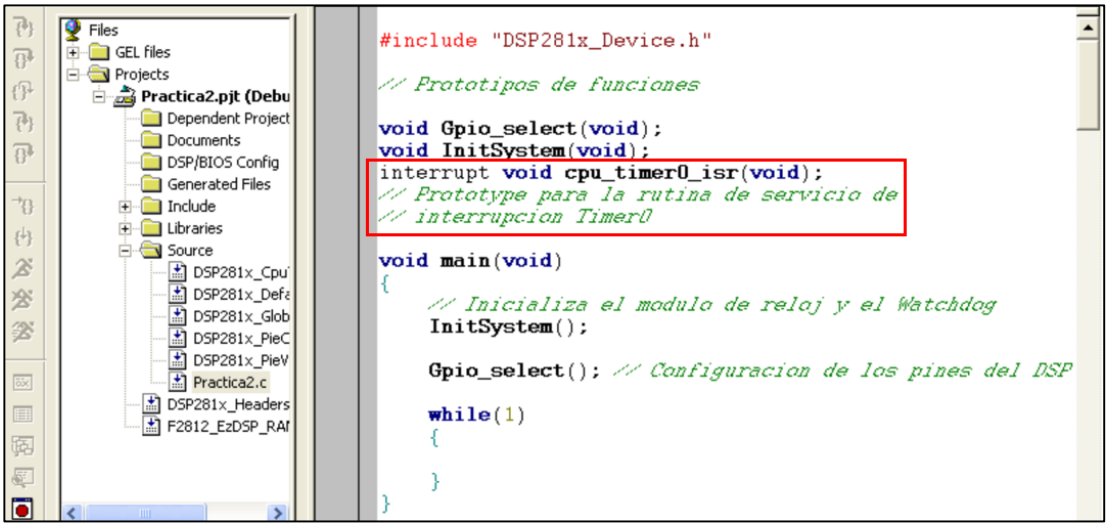


Figura 4.27 Prototipo de función "cpu_timer0_isr( )"

15. Al final del archivo de código fuente ***"Practica2"***, agregar la definición de la función **"interrupt void cpu_timer0_isr(void)"** (Ver Figura 4.28), en base a lo siguiente:

- Incrementar la variable **"CpuTimer0.InterruptCount"** para contar las interrupciones provocadas por el desbordamiento del "Timer0"
- Reconocer el servicio de interrupción de la línea de interrupción a la cual pertenece el "Timer0" mediante el registro PIEACK. Este paso es necesario para permitir una siguiente interrupción de esta misma fuente.

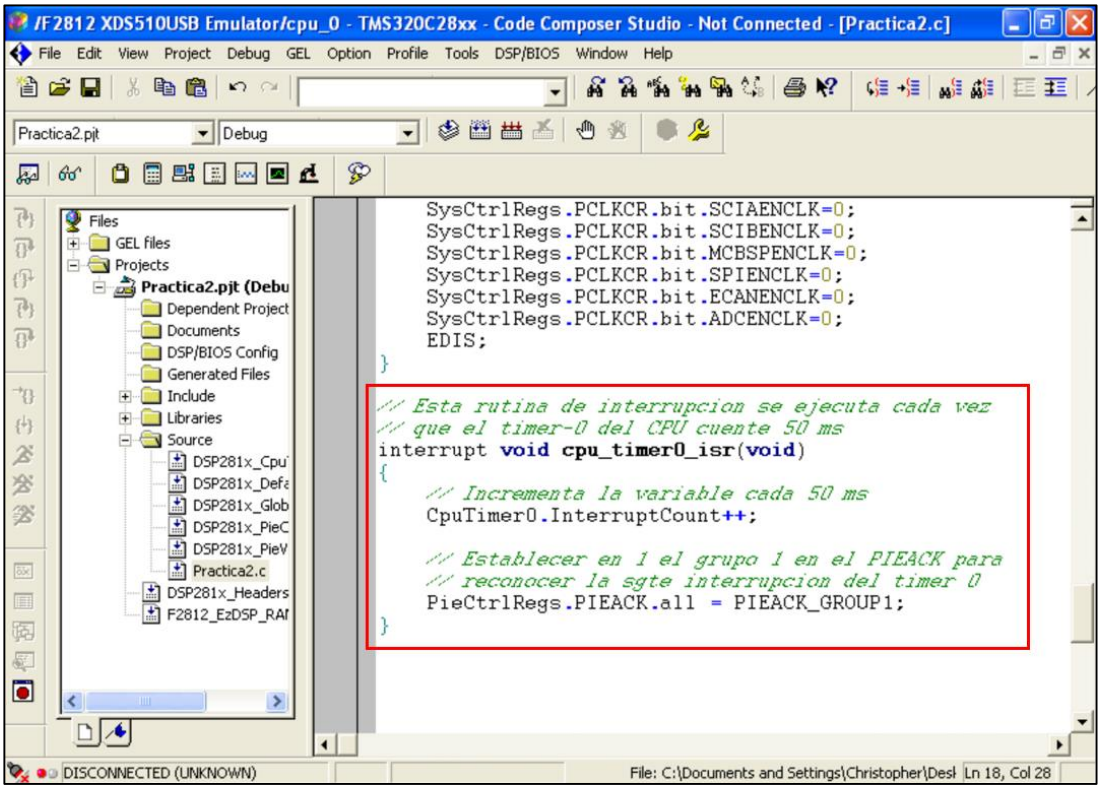


Figura 4.28 Definición de función "cpu_timer0_isr( )"

Cada vez que ocurra una interrupción por el “Timer0”, se incrementara la variable “CpuTimer0.InterruptCount”. Más adelante se configurará el periodo del “Timer0” para que la interrupción ocurra cada 50 milisegundos.

PROGRAMACIÓN DEL “MAIN”

16. Al inicio del “main”, declarar e inicializar las variables a usar en la práctica. Ver Figura 4.29

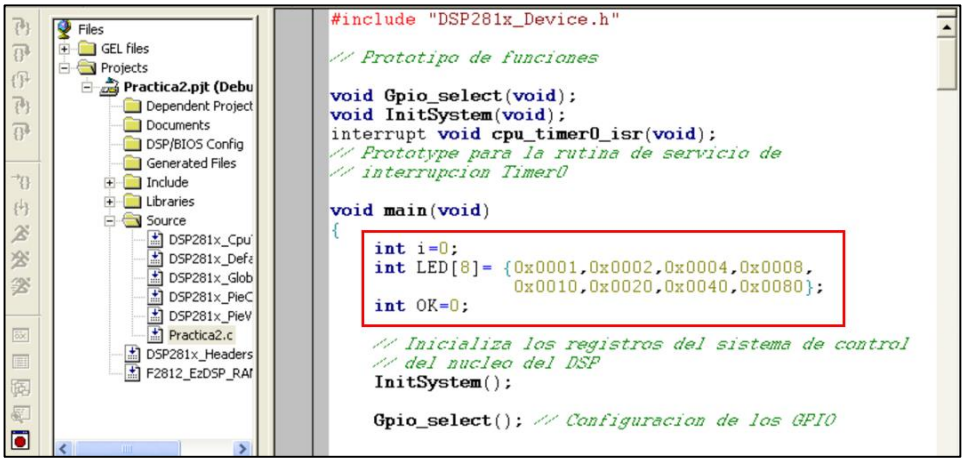


Figura 4.29 Declaración de Variables

17. En el “main”, antes de ingresar en lazo “while(1)”, llamar a la función **“InitPieCtrl()”**, para inicializar la unidad PIE, tal como se observa en la Figura 4.30.

Al llamar esta función se limpia todas las interrupciones pendientes y se deshabilita las líneas de interrupciones del PIE. La función está definida dentro del archivo *DSP281x_PieCtrl.c* que se agregó al proyecto al inicio de la práctica.

Figura 4.30 Inicializar la unidad PIE

18. En el “main”, antes de ingresar en lazo “while(1)”, llamar a la función **“InitPieVectTable()”**, tal como se observa en la Figura 4.31.

Al llamar esta función se copia la tabla de vectores del PIE en la variable global “PieVectTable”, para así luego poder re-mapear las fuentes de interrupciones que se vayan a usar. La función está definida dentro del archivo *DSP281x_PieVect.c* que se agregó al proyecto al inicio de la práctica.

El archivo *DSP281x_DefaultIsr.c* que se agregó al proyecto al inicio de la práctica, contiene algunas rutinas de servicio de interrupción.

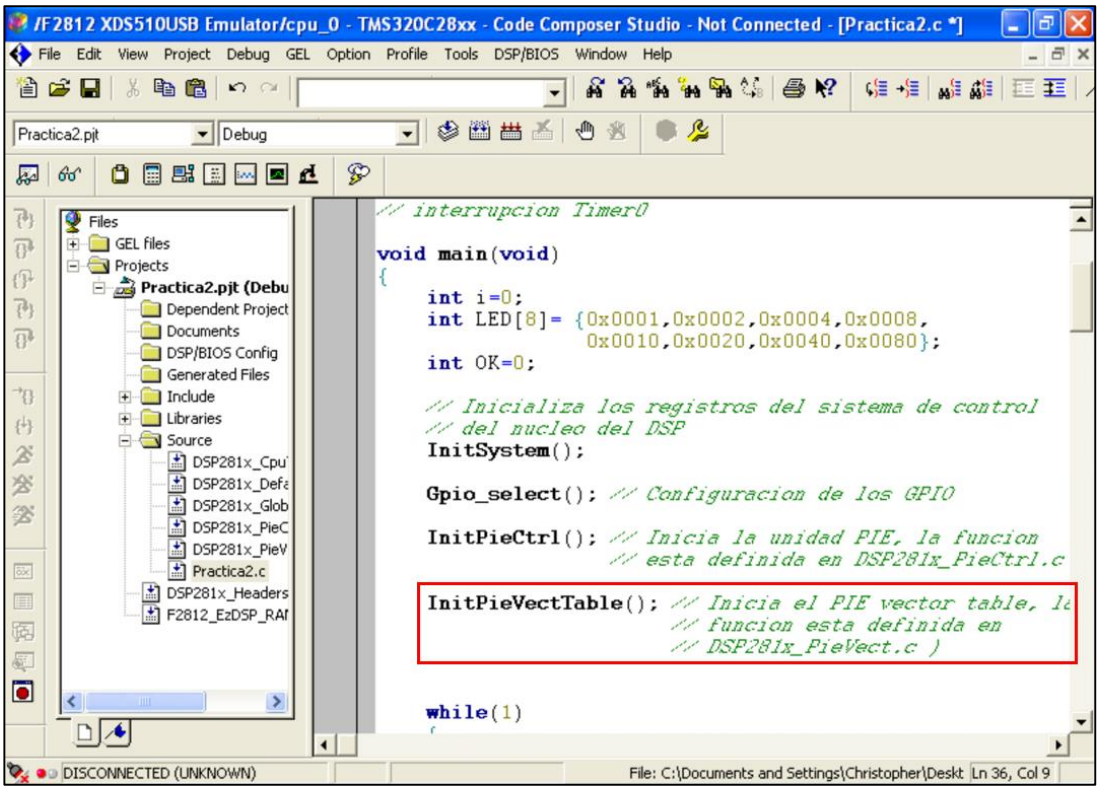


Figura 4.31 Copia la Tabla de Vectores del PIE a las variables globales

19. Re-mapear la dirección de la rutina de servicio de interrupción que se creó, al campo correspondiente del “Timer0” en la variable global “PieVectTable”. Ver Figura 4.32

Dentro del archivo *DSP281x_DefaultIsr.c* se puede acceder y agregar el código de la rutina de servicio de interrupción para el “Timer0”, pero no sería una buena práctica de programación, por lo que se re-mapeará la dirección de la rutina de servicio de interrupción **“interrupt void cpu_timer0_isr(void)”** que se había creado.

La variable global “PieVectTable” está protegida por los macros EALLOW y EDIS.

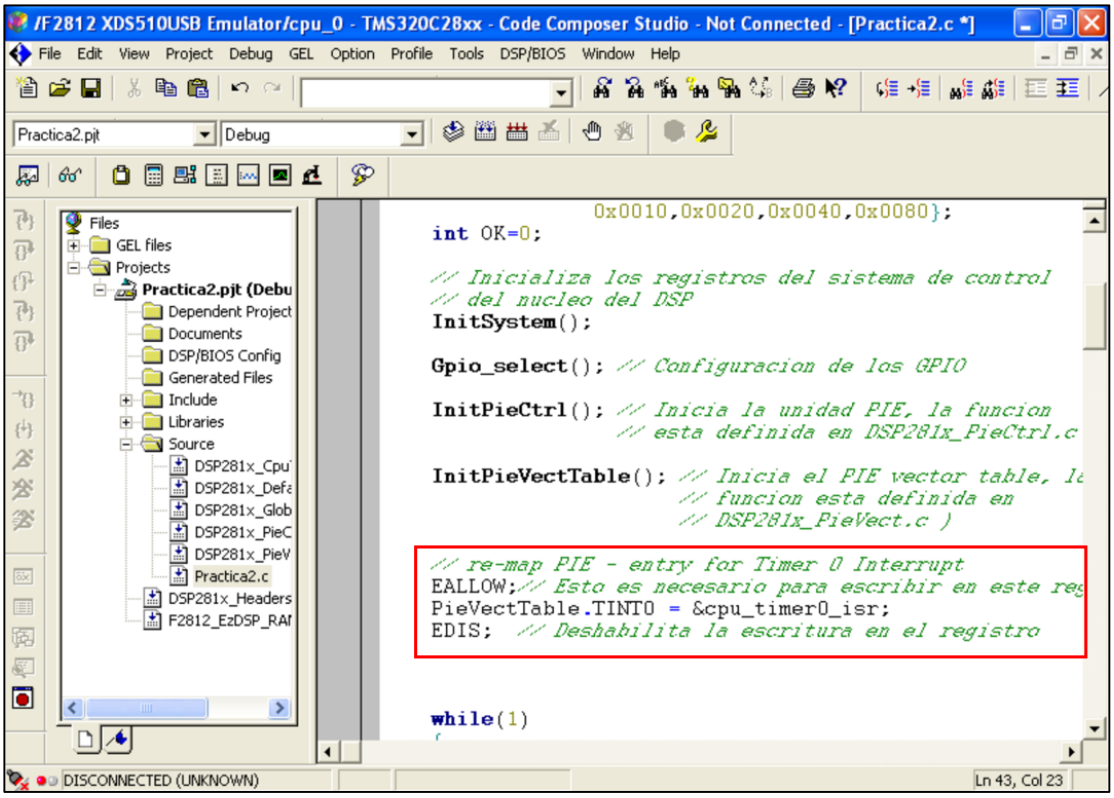


Figura 4.32 Re-mapeo de la posición de la tabla de vectores con la dirección de la función creada

20. En el “main”, antes de ingresar en lazo “while(1)”, llamar a la función **“InitCpuTimers()”**, tal como se observa en la Figura 4.33

Al llamar esta función se inicializa la configuración básica del “Timer0” para habilitarlo y que genere interrupciones. La función está definida dentro del archivo *DSP281x_CpuTimers.c* que se agregó al proyecto al inicio de la práctica.

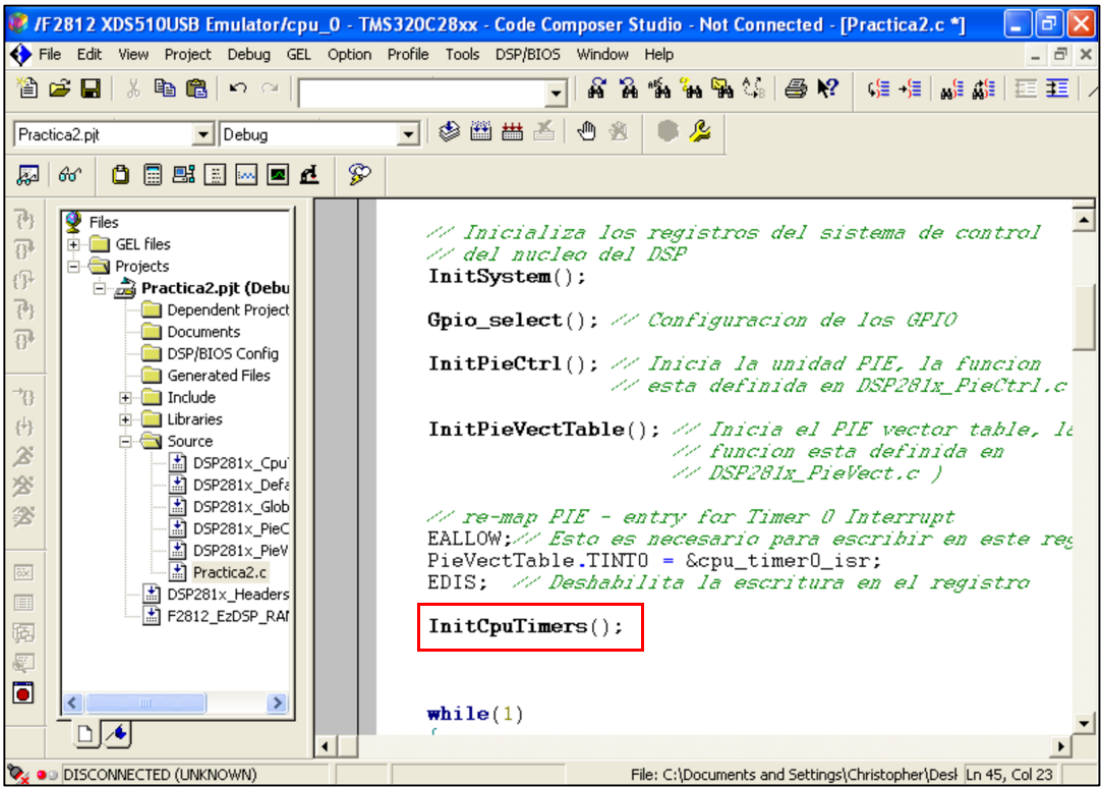


Figura 4.33 LIamar a la función "InitCpuTimers( )"

21. En el “main”, antes de ingresar en lazo “while(1)”, llamar a la función **“ConfigCpuTimer( )”**, tal como se observa en la Figura 4.34

En esta función se configura el “Timer0” para generar un periodo de 50 milisegundos (ms), es decir que cada 50 ms generará una fuente de interrupción. Esta función recibe tres argumentos para configurar el “Timer0” que son:

- El parámetro 1 es la dirección de la estructura del “Timer0”, la cual es **“CpuTimer0”**.
- El parámetro 2 es la frecuencia del núcleo del DSP en mega Herz, la cual es **“150”**.
- El parámetro 3 es el periodo en micro segundos en el cual se desbordará el “Timer0”, el cual es **“50000”**.

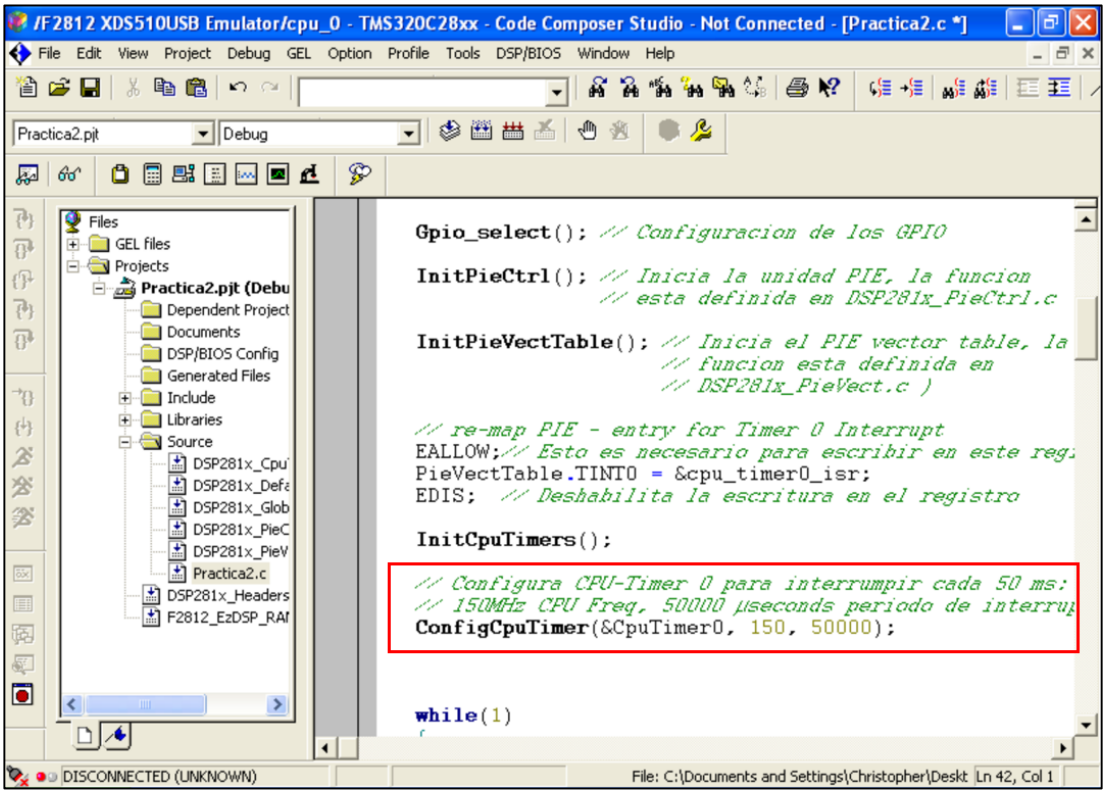


Figura 4.34 Configuración del Temporizador Interno del CPU del procesador

22. En el “main”, antes de ingresar en lazo “while(1)”, habilitar en el PIE la fuente de interrupción del “Timer0”, tal como se observa en la Figura 4.35

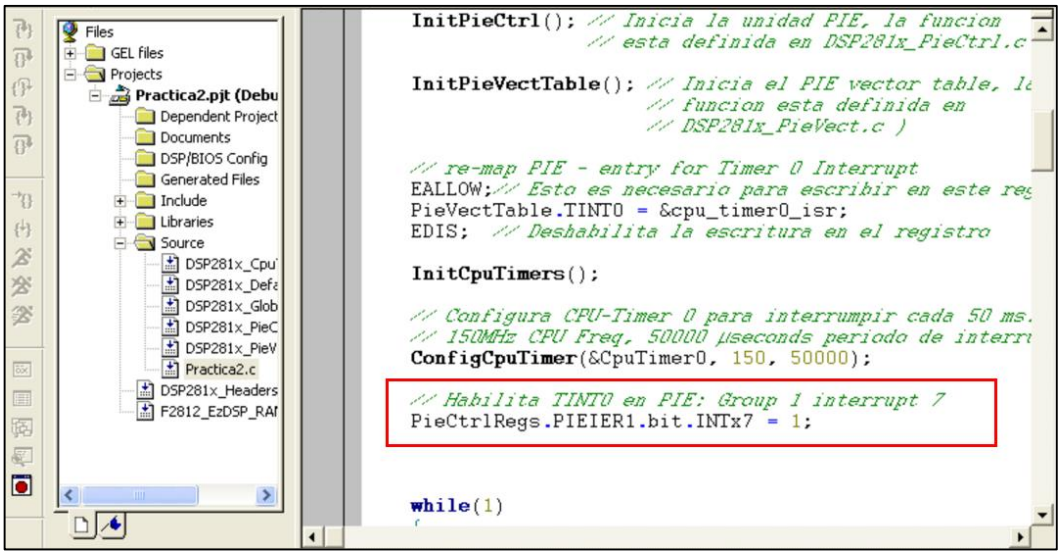


Figura 4.35 Habilita fuente de interrupción del Timer0

23. En el “main”, antes de ingresar en lazo “while(1)”, habilitar la primera línea de interrupción, habilitar también las interrupciones globales del DSP, tal como se observa en la Figura 4.36

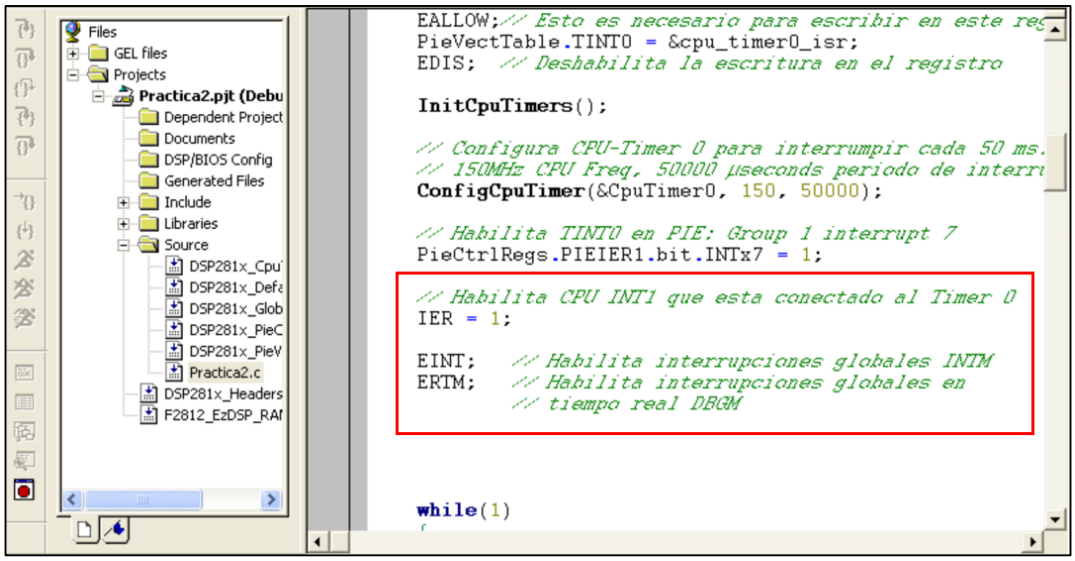


Figura 4.36 Habilita línea de interrupción del Timer0 e Interruptores globales

24. En el “main”, antes de ingresar en lazo “while(1)”, se dará inicio al conteo del “Timer0”, ajustando en ‘0’ el bit “TSS” del registro TCR, tal como se observa en la Figura 4.37

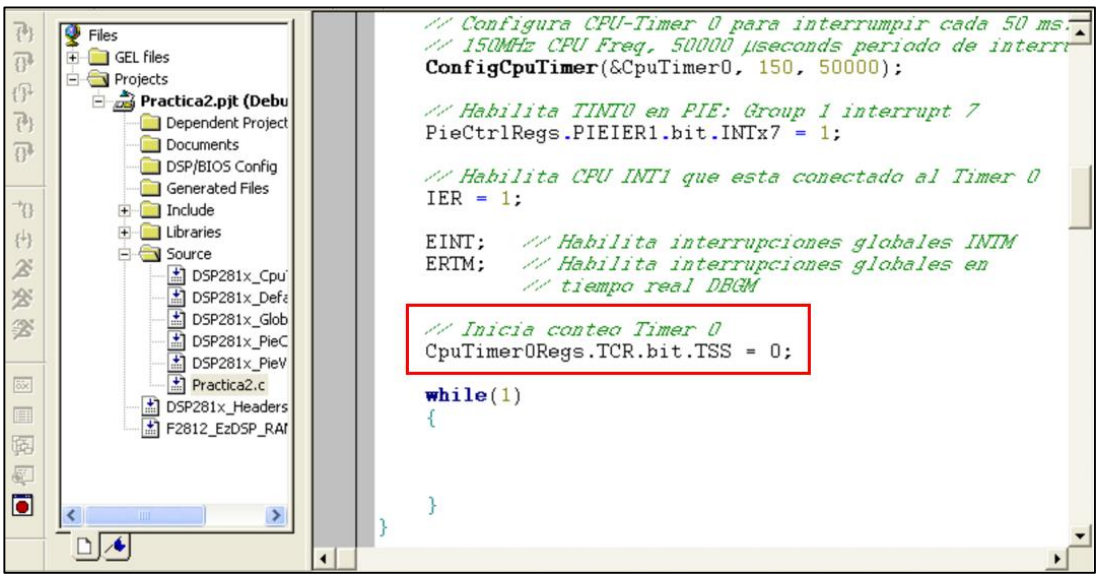


Figura 4.37 Inicia el conteo del Timer0

25. Dentro del “while(1)” se debe programar que cuando sea pulsado el botón S1 (GPIOD1 = 0), arranque la secuencia de desplazamiento de los leds (Conectados desde GPIO B7 a B0). Y cuando se presione el pulsador S2 (Conectado a GPIOD6), se detenga la secuencia de desplazamiento de los leds. Ver programación en Figura 4.38

Para hacer el desplazamiento de encendido de los leds, en la práctica 1 se usaba un retardo por software para poder apreciar dicho desplazamiento. En esta práctica se usa un retardo por hardware mediante la implementación del sistema de interrupciones del “Timer0” para poder apreciar el desplazamiento de los leds.

La variable “CpuTimer0.InterruptCount” se incrementa cada 50 ms debido a la interrupción del “Timer0”. El tiempo que tomará en desplazar la secuencia de

encendido entre led y led se lo establecerá en 150 ms, por lo que se debe esperar que la variable “CpuTimer0.InterruptCount” llegue a 3.

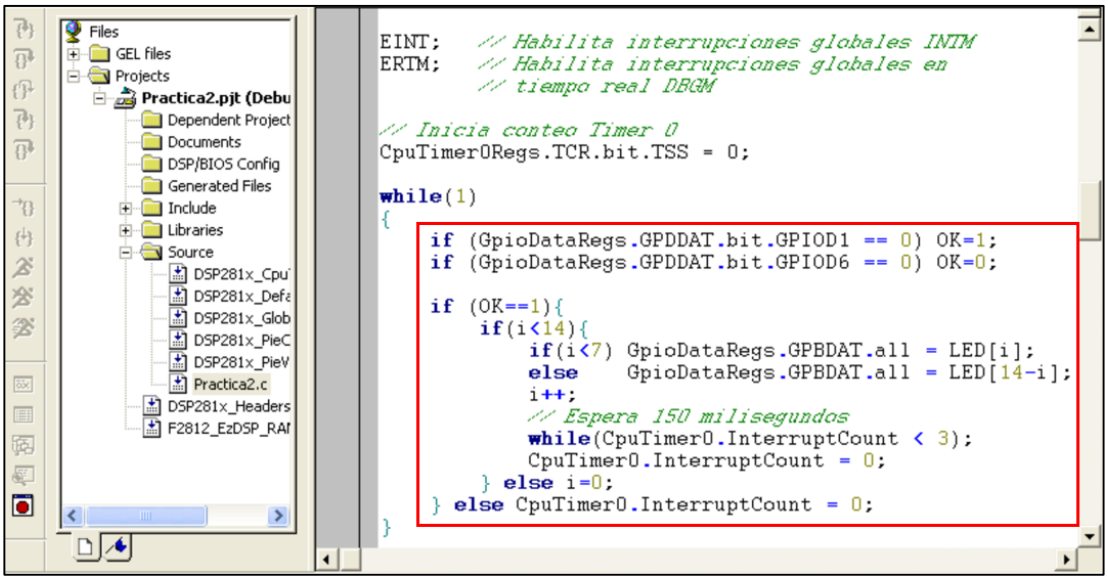


Figura 4.38 Programación del "while(1)"

COMPILAR, CARGAR Y EJECUTAR EL PROGRAMA

26. Compilar el proyecto seleccionando*: Project -> Rebuild All*. Solucionar los errores en caso de haberlos.

27. alimentar la tarjeta *eZdsp™F2812*.

28. Conectarse con el procesador: *Debug -> Connect*

29. Cargar el programa (*File -> Load Program…)* **Practica1.out** que se genera dentro de la carpeta Debug al compilar el proyecto

30. Ejecutar el programa cargado: *Debug -> Run*

En la Figura 4.39 se observa como arranca la secuencia de encendido de los leds cuando se presiona el pulsador S1 (GPIOD1)

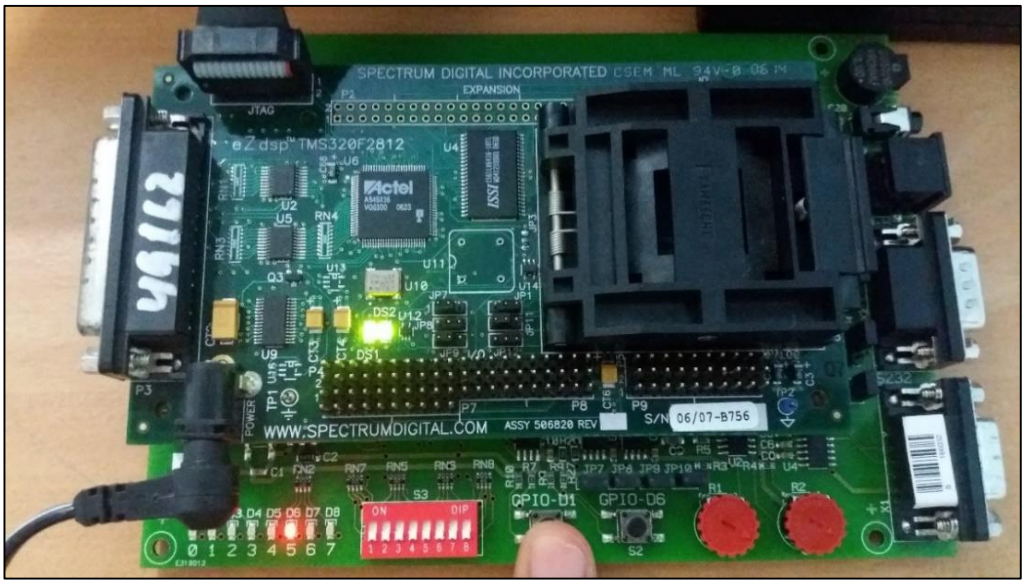


Figura 4.39 Foto del Kit cuando se presiona S1

En la Figura 4.40 se observa cómo se detiene la secuencia de encendido de los leds cuando se presiona el pulsador S2 (GPIOD6)

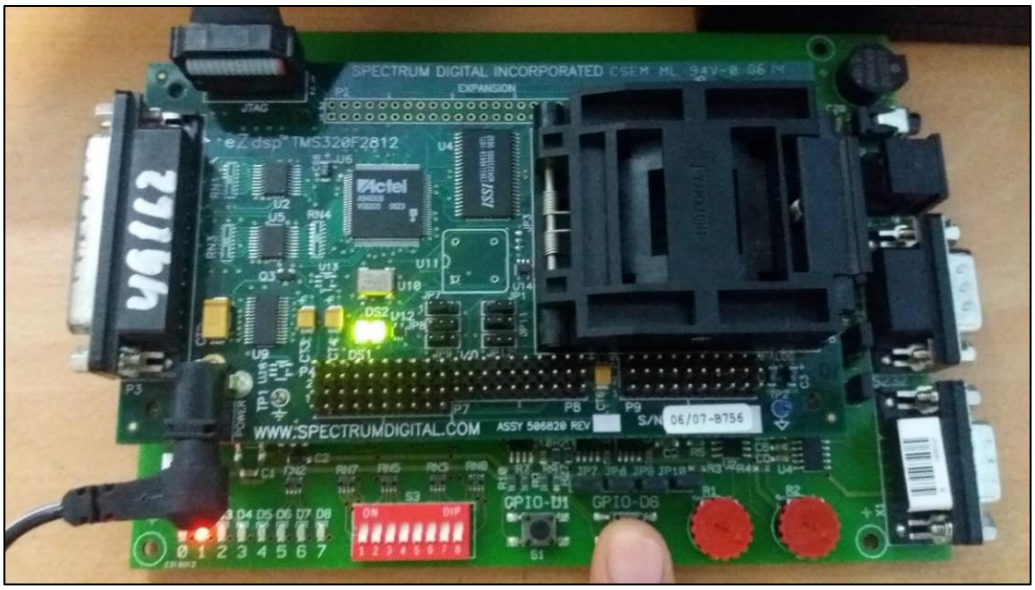


Figura 4.40 Foto del Kit cuando se presiona S2

31. Al finalizar la práctica, detener la ejecución del programa: *(Debug -> Halt)*

32. Resetear el CPU *(Debug -> Reset CPU)*, luego desconectarse de la tarjeta *(Debug -> Disconnect)*.

33. Quitar la alimentación de la tarjeta *eZdsp™F2812*.

PRÁCTICA # 3

TEMA:

"Comunicación mediante la interfaz SPI entre el procesador TMS320F2812 y el Convertidor Digital a Analógico TLV5617"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Establecer comunicación SPI entre el procesador *TMS320F2812* y el convertidor digital a analógico TLV5617.
- Generar y transmitir señales digitales de forma diente de sierra mediante el procesador *TMS320F2812* para sea convertida por el DAC TLV5617.
- Observar el valor analógico que convierte el chip integrado TLV5617 a través del visualizador de gráficos y el modo Tiempo Real de Code Composer Studio.

REQUISITOS MÍNIMOS:

- Tener Instalado el software Code Composer Studio (versión 3.3) con su respectivo drive para el manejo del convertidor JTAG a USB.
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos de programación en Lenguaje C
- Haber culminado satisfactoriamente la Práctica # 2.
- Lectura y comprensión del archivo de fundamentación teórica:
  "SERIAL PERIPHERAL INTERFACE (SPI)"

BREVE EXPLICACIÓN DE LA PRÁCTICA

El procesador *TMS320F2812* embebe unidades periféricas de comunicación, entre ellas la unidad SPI, la misma que establecerá una interfaz SPI entre el procesador y el chip DAC TLV5617 que es parte de la tarjeta *KSPS-0504 adaptor board*. La configuración para el uso de la unidad SPI se la realiza en lenguaje C mediante el

software Code Composer Studio en conjunto con las librerías y archivos código fuente provistos por Texas Instruments para el manejo de unidades periféricas del procesador *TMS320F2812*.

Se generan dos señales de forma diente de sierra mediante el incremento y decremento de dos variables, estas señales se trasmiten desde el procesador *TMS320F2812* hacia el chip DAC TLV5617 mediante la interfaz SPI. El periférico externo TLV5617 convierte la señal digital en una señal analógica periódicamente en función del reloj dado por la unidad SPI del procesador.

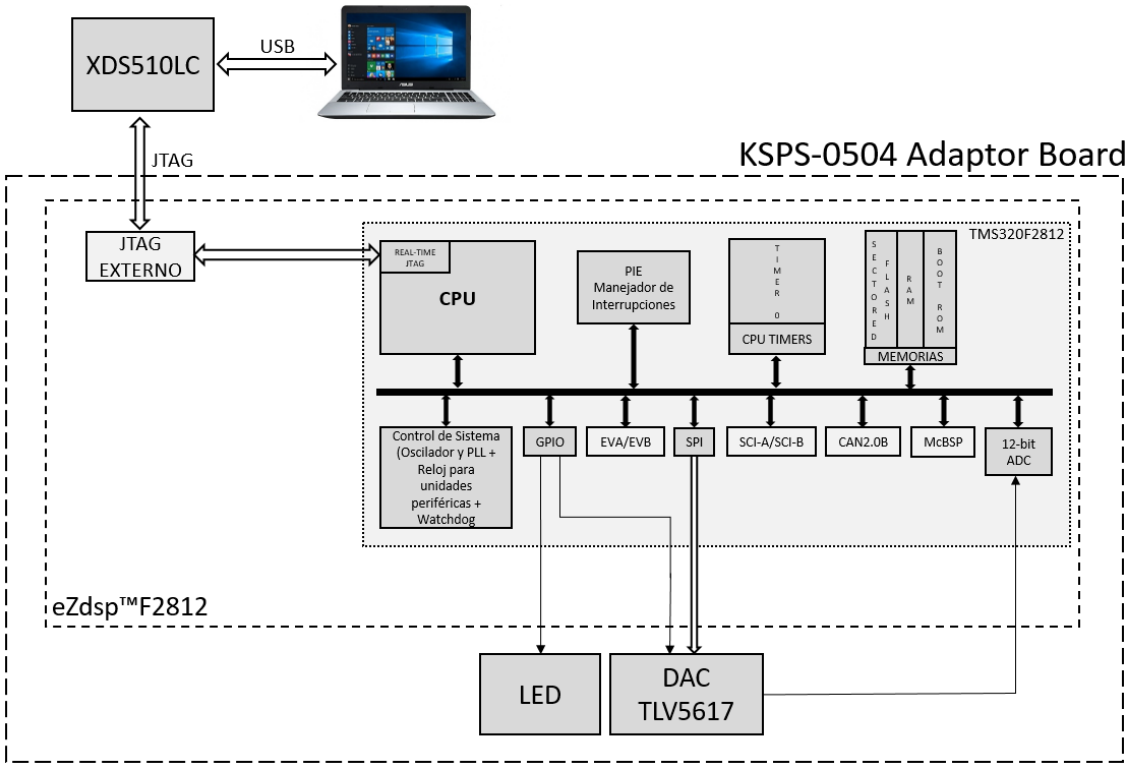


Figura 4.41 Diagrama de Bloques de Comunicación SPI entre DSP y Chip DAC

Al final de la práctica se podrán evidenciar en tiempo real los valores de los pines de salida del chip TLV5617 con ayuda del ADC interno del procesador *TMS320F2812* y con las herramientas de visualización gráfica y numérica del software. Ver Figura 4.41

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Code Composer Studio*.

NOTA: Si al abrir el programa sale algún error de conexión con la PC, el motivo podría ser que no esté configurado el *Setup CCStudio V3.3* tal como se ve en la practica 1. Si el estudiante no ha realizado la *practica 1* se recomienda desarrollarla para evitar tener problemas de comunicación entre su PC y las tarjetas, para luego continuar con esta práctica.

3. Crear un proyecto nuevo (*Project -> New..)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica3
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** para crear el proyecto

4. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

// Prototipo de funciones
void Gpio_select(void);
void InitSystem(void);
interrupt void cpu_timer0_isr(void);

void main(void)
{
        InitSystem();

        Gpio_select();

        InitPieCtrl();
        InitPieVectTable();
```

```
        EALLOW;
        PieVectTable.TINT0 = &cpu_timer0_isr;
        EDIS;

        InitCpuTimers();
        ConfigCpuTimer(&CpuTimer0, 150, 50000);

                PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
        IER = 1;
        EINT;   // Habilita interrupciones globales INTM
        ERTM;   // Habilita interrupciones globales en
                        // tiempo real DBGM

        CpuTimer0Regs.TCR.bit.TSS = 0;

        InitAdc();
        AdcRegs.ADCTRL1.bit.SEQ_CASC = 0;           // Dual Sequencer Mode
        AdcRegs.ADCTRL1.bit.CONT_RUN = 0;           // No Continuous run
        AdcRegs.ADCTRL1.bit.CPS = 0;                // prescaler = 1
        AdcRegs.ADCMAXCONV.all = 0x0001;
        AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x1;.
        AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x9;.
        AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1 = 0;
        AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;
        AdcRegs.ADCTRL3.bit.ADCCLKPS = 2;

        EALLOW;// Esto es necesario para escribir en este registro
        PieVectTable.ADCINT = &ADC_ISR;
        EDIS;  // Deshabilita la escritura en el registro

        PieCtrlRegs.PIEIER1.bit.INTx6 = 1;

        while(1)
        {
           while(CpuTimer0.InterruptCount < 3);
        }
}

void Gpio_select(void)
{
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;
        GpioMuxRegs.GPBMUX.all = 0x0;
        GpioMuxRegs.GPDMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;


        GpioMuxRegs.GPADIR.all = 0x0;
        GpioMuxRegs.GPBDIR.all = 0x00FF;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
```

```
        GpioMuxRegs.GPFDIR.all = 0x0;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioDataRegs.GPBDAT.all = 0x0;
        EDIS;
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00E8;

        SysCtrlRegs.SCSR = 0;

        SysCtrlRegs.PLLCR.bit.DIV = 10;

        SysCtrlRegs.HISPCP.all = 0x1;
        SysCtrlRegs.LOSPCP.all = 0x2;

        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
}

interrupt void cpu_timer0_isr(void)
{
        CpuTimer0.InterruptCount++;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

5. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica3.c"***

6. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica3.c".***

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

7. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **DSP281x_GlobalVariableDefs.c**

8. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_Headers_nonBIOS.cmd**

9. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

10. De la *dirección "C:\tidcs\c28\dsp281x\v100\DSP281x_common\source"* agregar al proyector los siguientes archivos:

- **DSP281x_PieCtrl.c**
- **DSP281x_PieVect.c**
- **DSP281x_DefaultIsr.c**
- **DSP281x_CpuTimers.c**
- **DSP281x_Adc.c**
- **DSP281x_usDelay.asm**

11. Agregar el archivo de código fuente **ADC_ISR.c** otorgado por el profesor

12. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL "BUILD OPTIONS"

13. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options..*
- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include
  en el campo *Include Search Path*

14. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **400** en el campo *Stack Size.*
- Dar clic en el botón *OK* para terminar la configuración del *Build Options*

AGREGAR CÓDIGO DE INICIALIZACIÓN SPI

15. Habilitar el reloj de la unidad periférica SPI en la función “InitSystem()”. Como se observa en la Figura 4.42

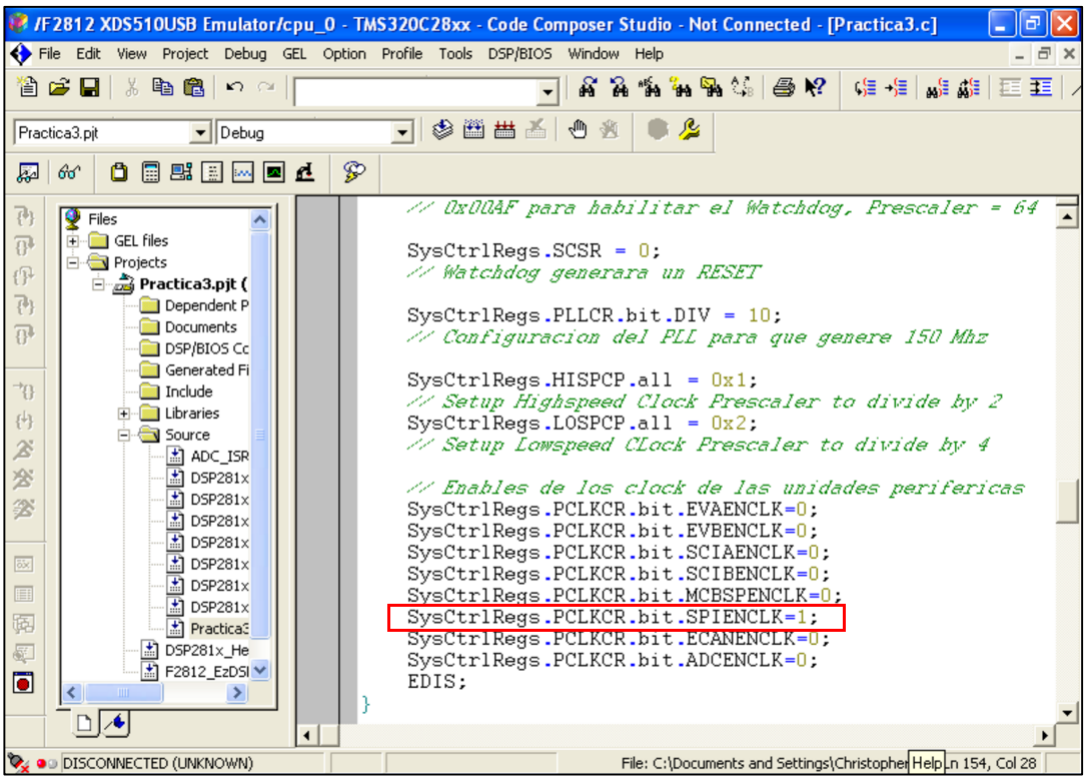


Figura 4.42 Habilita Reloj de la unidad periférica de comunicación SPI

16. Configurar como función primaria los pines del puerto F en la función “Gpio_select()”, los cuales están asociados a la unidad periférica de comunicación SPI. Ver Figura 4.43

Figura 4.43 Selecciona función primaria en todos los pines del puerto F

17. Configurar los pines GPIO D0 y D5 como pines de salida, los cuales serán los chip-select para el TLV5617A y la EEPROM M95080 respectivamente. Ver Figura 4.44

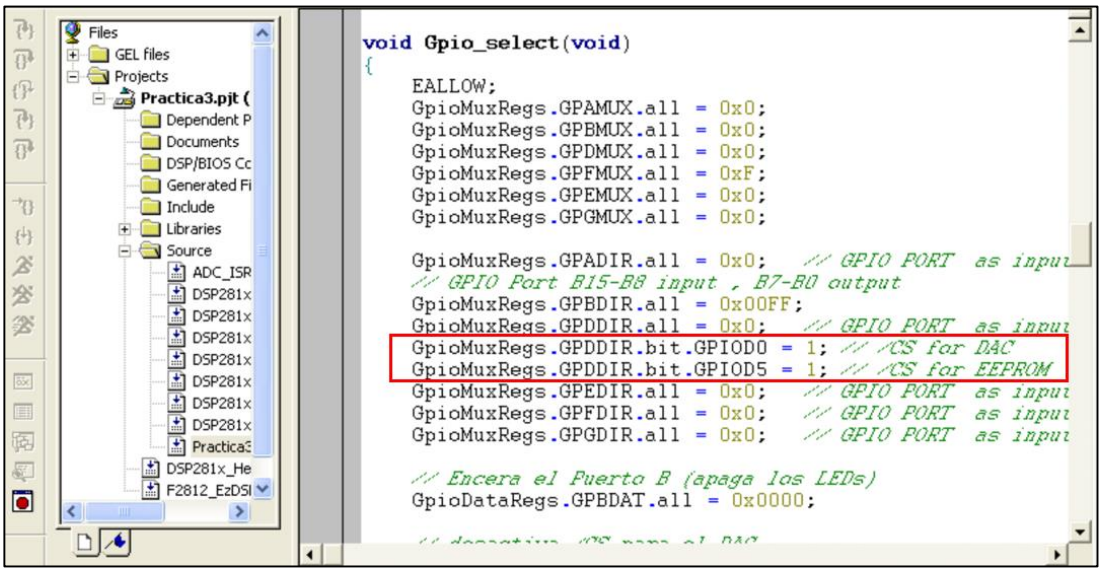


Figura 4.44 Configura como salida los pines GPIOD0 y GPIOD5

18. Inicializar los pines GPIO D0 y D5 en ‘1’, para deshabilitar el chip-select para el TLV5617A y la EEPROM M95080 respectivamente. Ver Figura 4.45

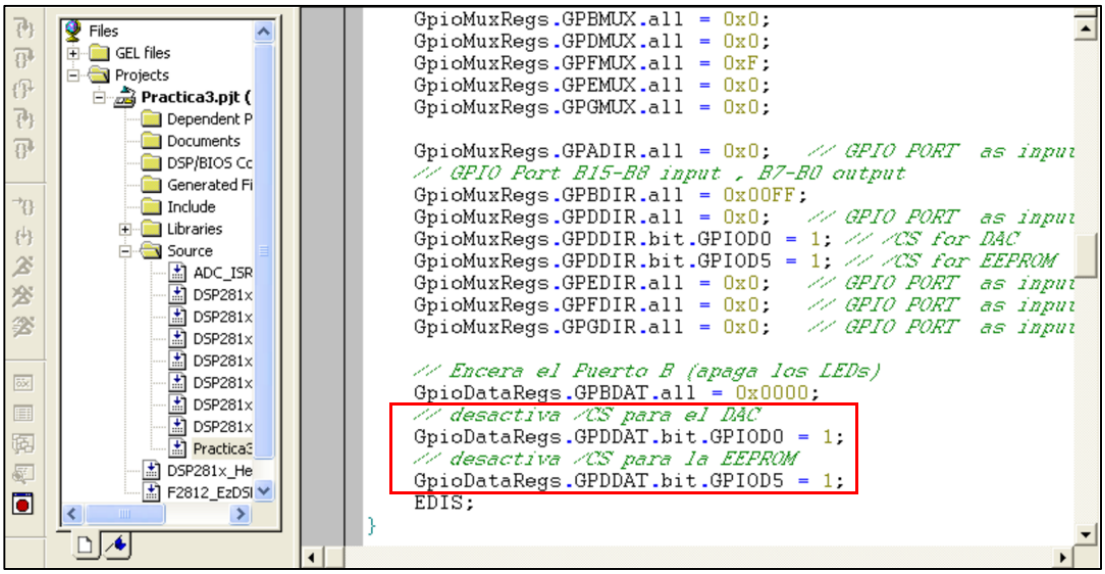


Figura 4.45 Establece en '1' para desactivar los Chip-select del DAC y EEPROM

19. Al inicio del archivo de código fuente ***“Practica3”***, antes del “main” agregar el prototipo de función **“void SPI_Init(void)”**, como se observa en la Figura 4.46

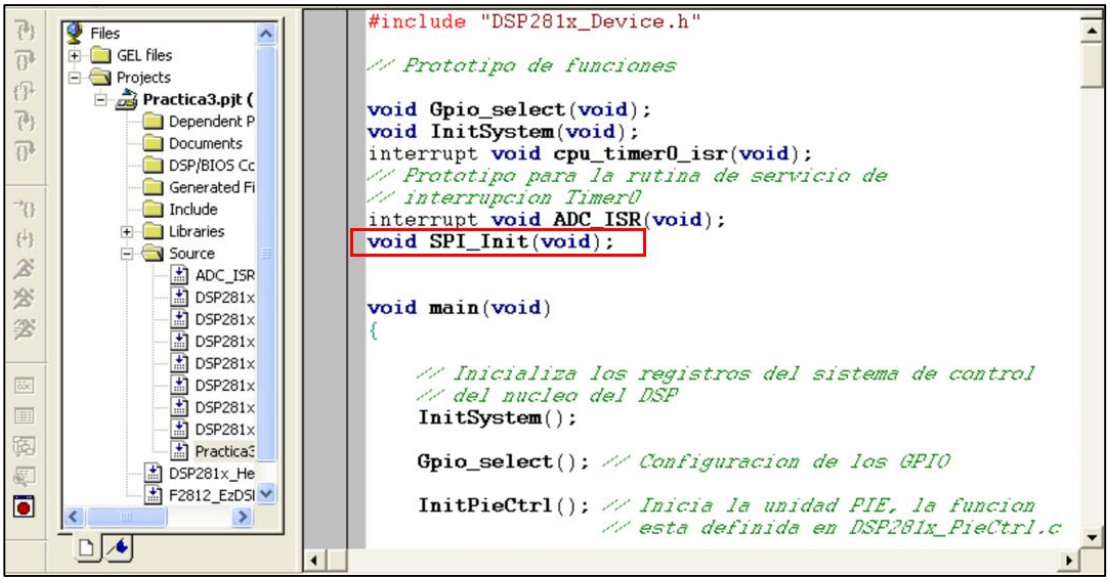


Figura 4.46 Prototipo de función "SPI_Init( )"

20. En el “main”, antes de la línea de código “InitAdc()”, llamar a la función **“SPI_Init()”**, tal como se observa en la Figura 4.47

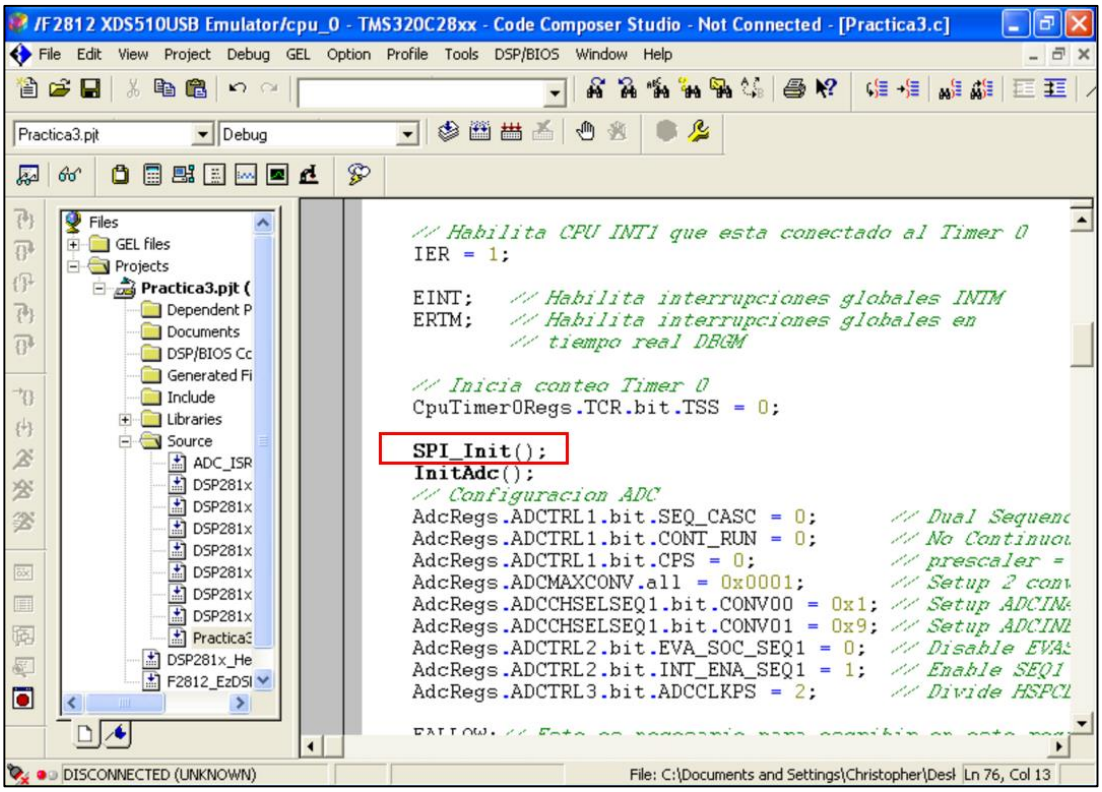


Figura 4.47 Inicializa la Unidad Periférica de Comunicación SPI

21. Al final del archivo código fuente ***“Practica3”***, agregar la definición de la función **“void SPI_Init(void)”** en base a la siguiente configuración:

Para el registro SPICCR:

- Resetear las banderas del SPI
- Transferir datos cuando se detecte flanco de bajada del reloj
- Configurar 16 bits de datos por trama

Para el registro SPICTL:

- Deshabilitar la interrupción por desbordamiento para recepción de datos
- Generar un retardo de medio ciclo

- Configurarlo como maestro
- Habilitar Transmisión
- Deshabilitar las interrupciones del SPI para transferencia de datos

Para el registro SPIBRR:

- Configurar la velocidad a 300 KHz, sabiendo que la frecuencia de LSPCLK es de 37.5 MHz.

NOTA: para los bits reservados de los registros colocar el valor de “0”

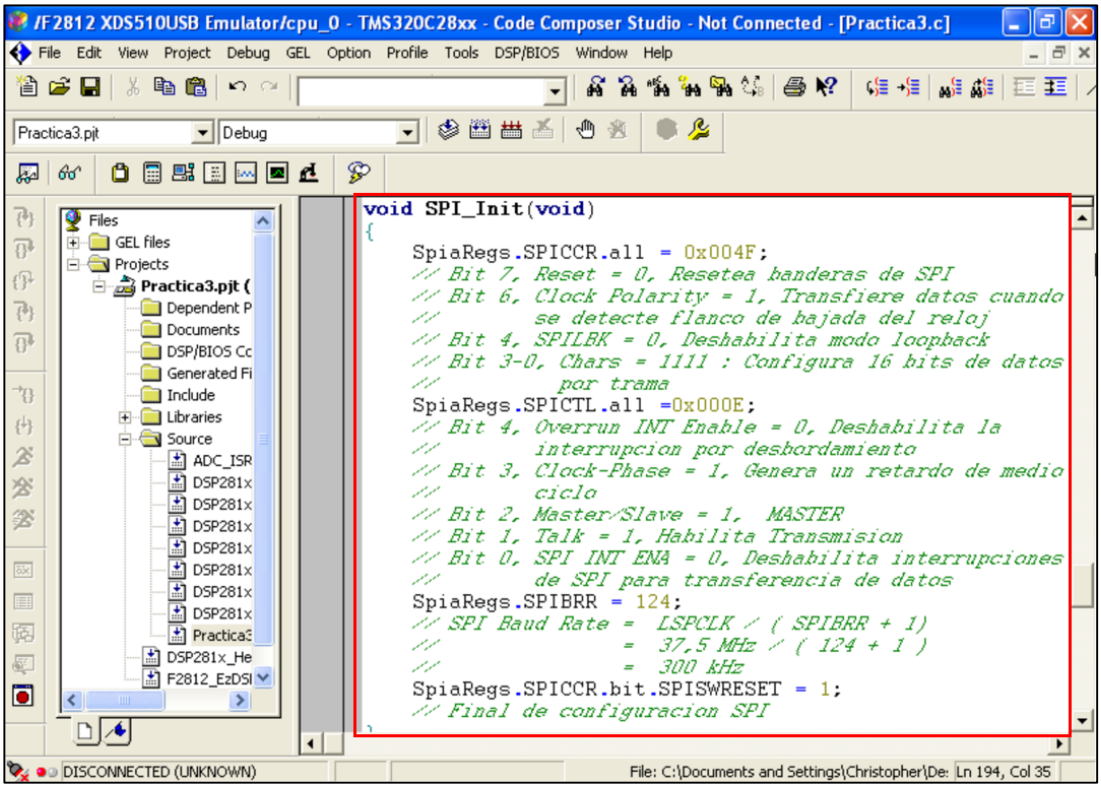


Figura 4.48 Definición de la función "SPI_Init( )"

AGREGAR CÓDIGO DE ACTUALIZACIÓN DEL DAC

22. Al inicio del “main”. Agregar una variable entera “Voltage” para almacenar el valor digital actual que se convertira en el DAC. El valor inicial para la variable “Voltage” es de 0. Ver Figura 4.49

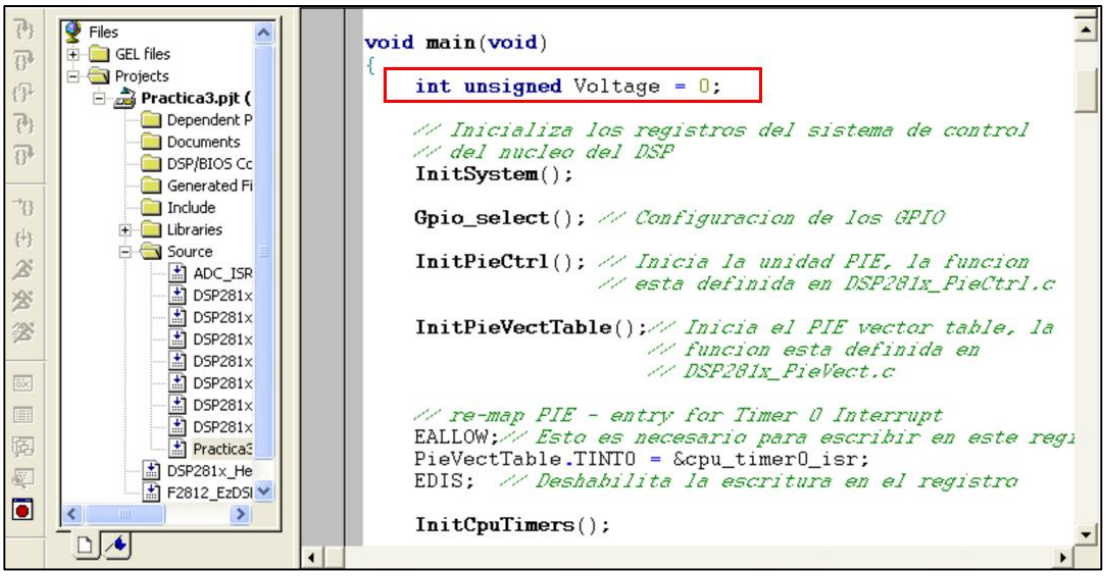


Figura 4.49 Declaración de Variable a usar en la práctica

23. Habilitar el reloj de la unidad periférica ADC en la función “InitSystem()”. Ver Figura 4.50

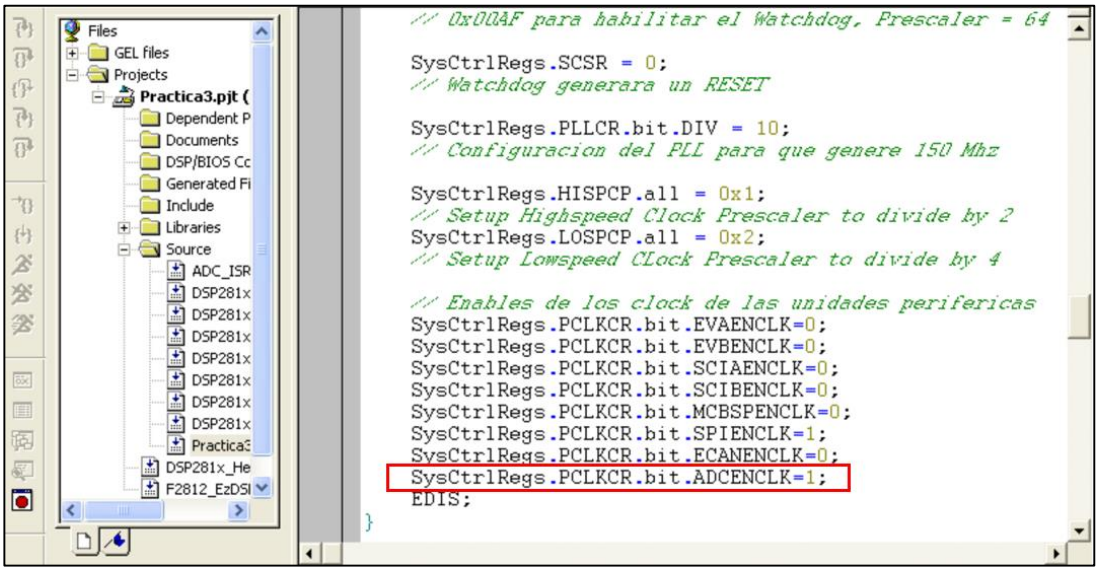


Figura 4.50 Habilitación del reloj de la unidad periférica ADC

24. Para actualizar el DAC una buena solución sería escribir una función que se llame "DAC_Update" con un parámetro de entrada (value). Agregar el prototipo **"void DAC_Update(int value)"** al inicio del archivo de código fuente. Ver Figura 4.51

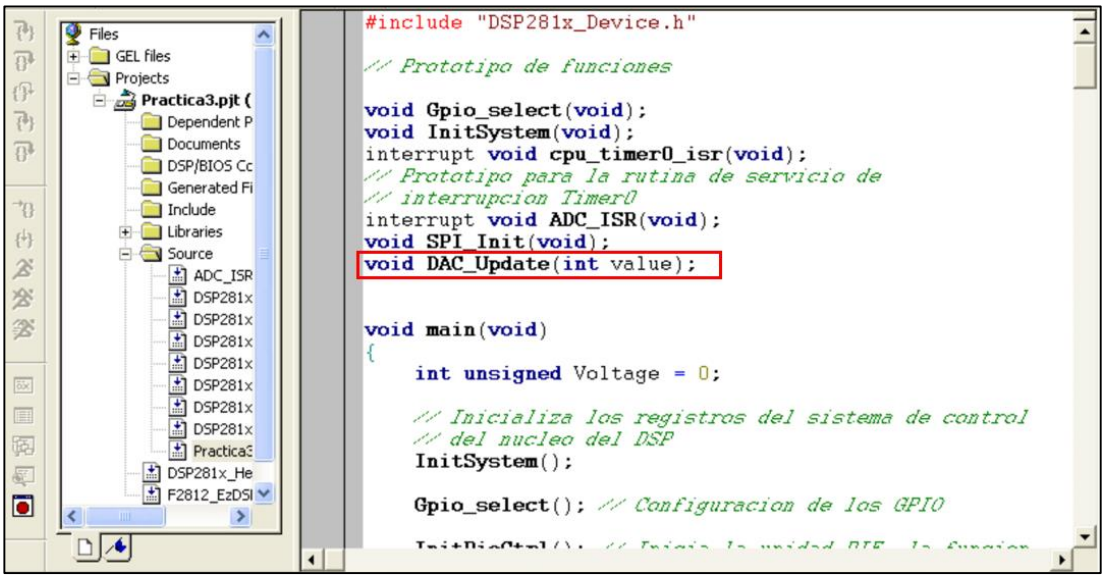


Figura 4.51 Prototipo de función "DAC_Update( )"

25. Al final del archivo código fuente, agregar la definición de la función **"void DAC_Update(int value)"** (Ver Figura 4.52), tomando en cuenta lo siguiente:

- Habilitar el DAC TLV5617A, encerando el bit **GpioDataRegs.GPDDAT.bit.GPIOD0**
- Transmitir el valor que se desea convertir tomando en cuenta como debe estar configurado los 16 bits de transferencia de datos para que el DAC pueda convertir el valor correcto y que solo use el canal A.
- Esperar a que termine de enviar toda la información
- Generar un retardo para que el DAC termine de convertir
- Deshabilitar el DAC TLV5617A, colocando un 1 en el bit **GpioDataRegs.GPDDAT.bit.GPIOD0**

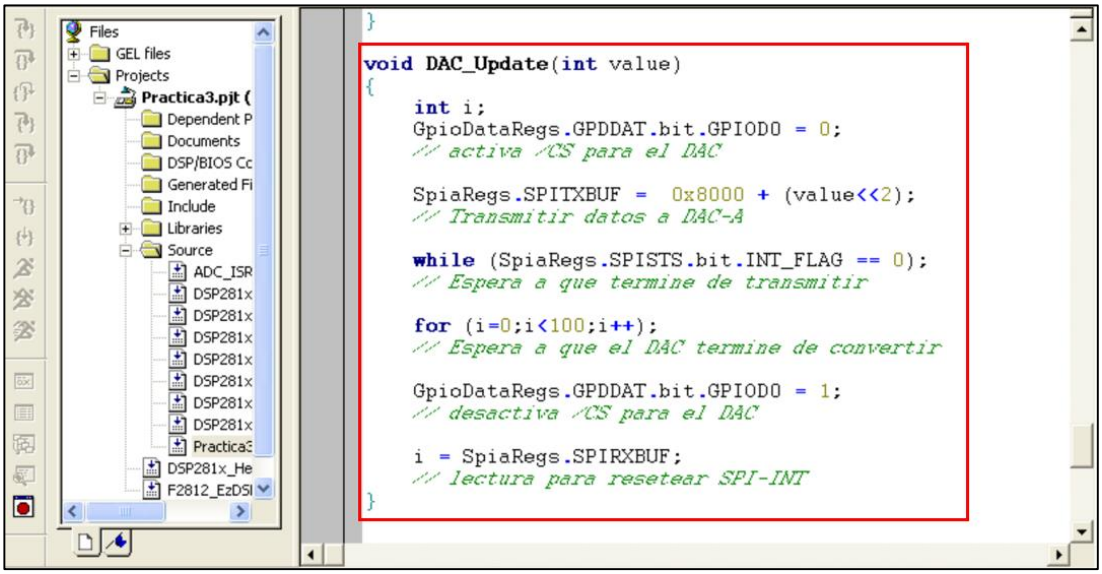


Figura 4.52 Definición de función "DAC_Update( )"

26. En el “main”, se llama a la función “DAC_Update” dentro del lazo “while(1)”, después de la línea: “GpioDataRegs.GPBTOGGLE.bit.GPIOBO = 1”. Ver Figura 4.53

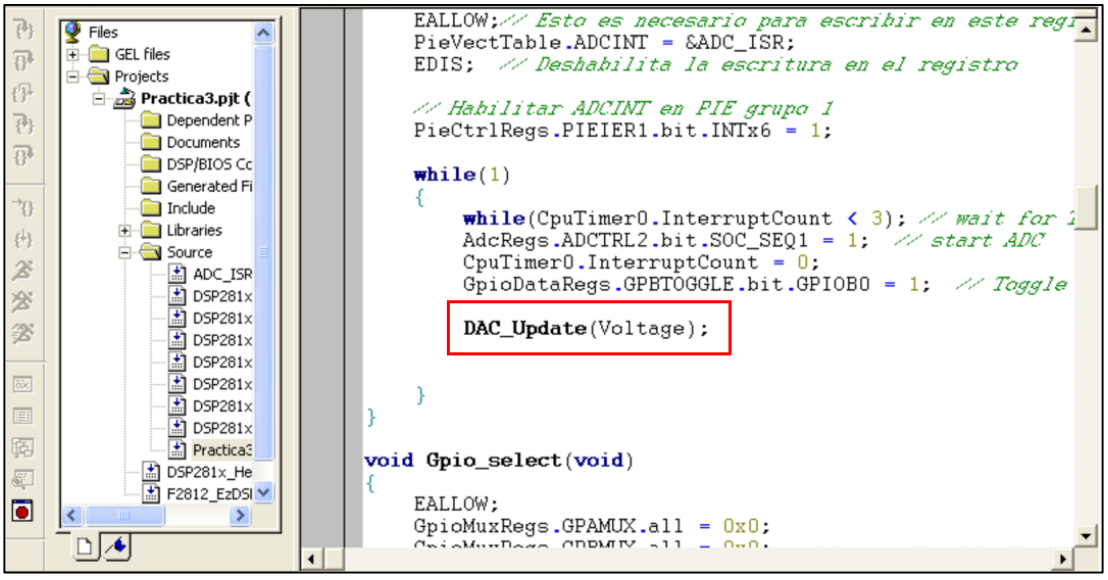


Figura 4.53 llamado a la función "DAC_Update( )" para actualizar transferencia de datos al Chip

Cada vez que se llame a la función “DAC_Update” se actualizara el valor de voltaje de salida del canal A del TLV5617A, pero como la variable “Voltage” en el programa es inicializada con 0, y no se altera su valor en el resto de código, entonces no se puede apreciar la conversión.

27. Para poder apreciar la conversión, se hará que la variable “Voltage” tome la forma de una señal diente de sierra tal como se muestra en la Figura 4.54:

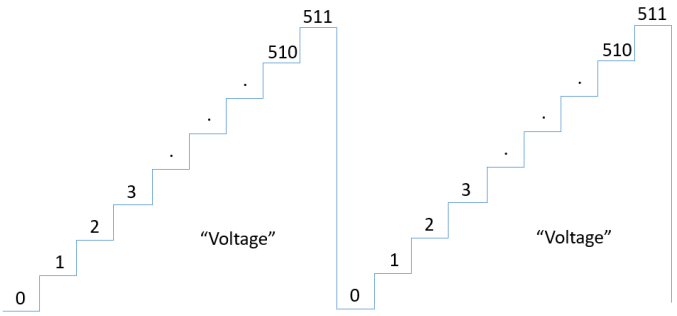


Figura 4.54 Señal diente de sierra digital

Para generar la señal diente de sierra digitalmente, en lenguaje C se puede incrementar la variable “Voltage” cada vez que complete un lazo en el “while(1)”, y encerando la variable “Voltage” cada vez que llegue a 511. Como se observa en la Figura 4.55

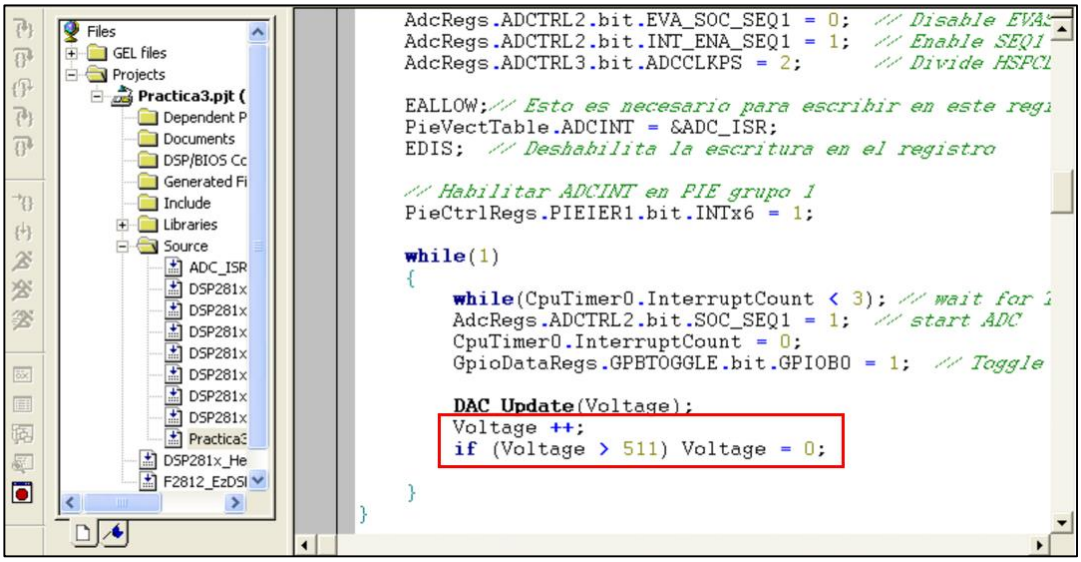


Figura 4.55 Código para formar señal diente de sierra digital

COMPILAR Y CARGAR EL PROGRAMA

28. Compilar el proyecto seleccionando*: Project -> Rebuild All*. Solucionar los errores en caso de haberlos.

29. alimentar la tarjeta *eZdsp™F2812*, y conectarse con el procesador: *Debug -> Connect*

30. Cargar el programa (*File -> Load Program…)* **Practica3.out** que se genera dentro de la carpeta Debug al compilar el proyecto

REAL-TIME MODE

31. Seleccionar el modo en tiempo real: *Debug -> Real-time Mode*

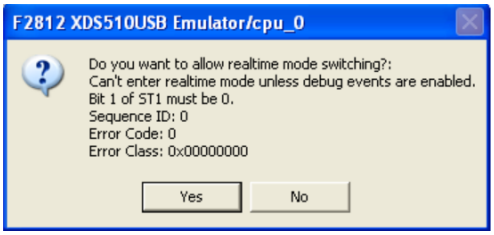


Figura 4.56 Ventana para confirmación del modo en tiempo real

Aparecerá una ventana en la que se procederá a dar clic en **Yes**. Ver Figura 4.56

32. Ejecutar el programa cargado: *Debug -> Run*

Figura 4.57 Foto del Kit mientras se ejecuta el programa

En la Figura 4.57 se observa una foto del Kit, donde el led D1 encendido indica que se está transmitiendo datos a la DAC mediante la interfaz SPI.

33. Abrir una ventana de Watch Window para ver el resultado de la conversión de la lectura del TLV5617A *(GEL -> Watch ADC Registers -> ADCRESULT_0_to_3)*

El registro *ADCRESULT0* es el que mostrara el valor en hexagesimal, para cambiar el valor a binario, en la fila de la variable damos clic derecho en “Radix” y damos clic en *dec.*

34. Abrir la ventana de gráfico para visualizar la variable convertida en decimal (View -> Graph -> Time/Frequency), y configurar los parámetros según la Figura 4.58

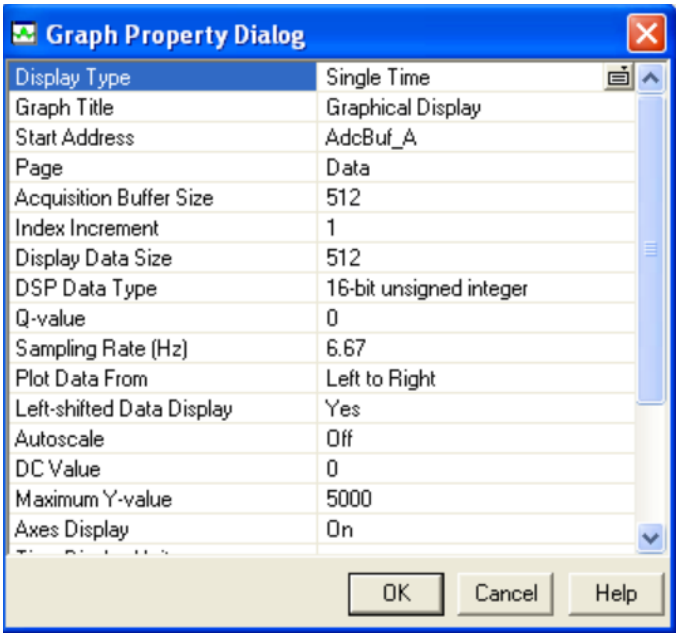


Figura 4.58 Configuración de la gráfica en "CCStudio"

35. Dar clic en **OK**, aparecerá una gráfica como se observa en la Figura 4.59, donde se dará clic derecho y se seleccionará la opción ***Continuous Refresh***.

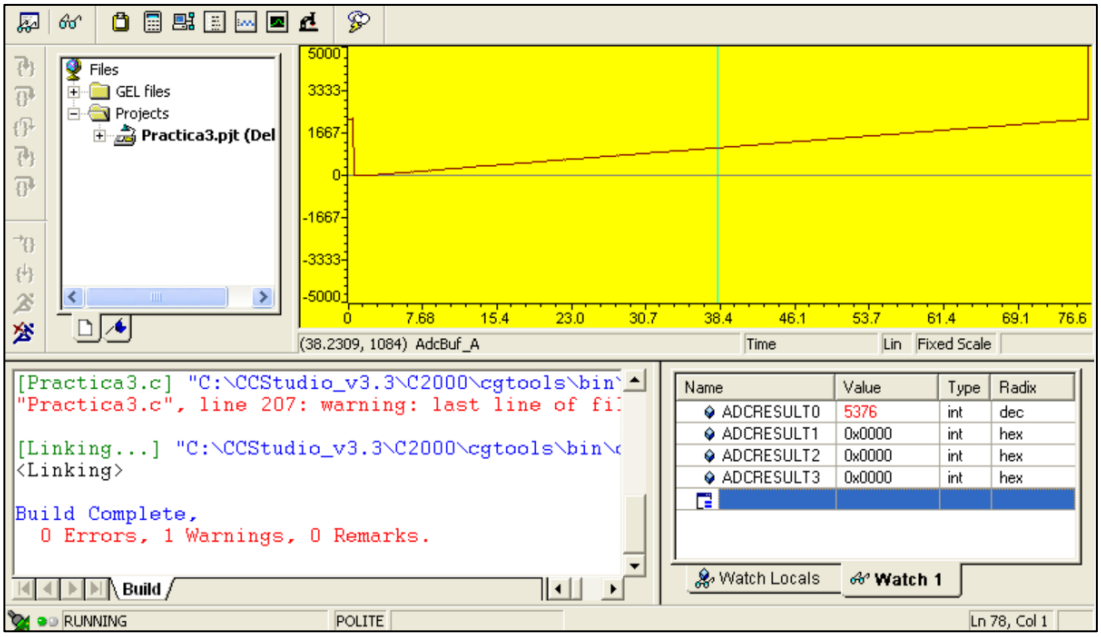


Figura 4.59 Observación en tiempo real de la gráfica y el valor del diente de sierra

36. Al finalizar la práctica, quitar el modo de *Continuous Refresh* dando clic derecho sobre la gráfica y deshabilitando el *Continuous Refresh*, luego detener la ejecución del programa: *(Debug -> Halt)*, finalmente deshabilitar el modo en tiempo real *(Debug -> Real-Time Mode)*

37. Resetear el CPU *(Debug -> Reset CPU)*, luego desconectarse de la tarjeta *(Debug -> Disconnect).* Y quitar la alimentación de la tarjeta *eZdsp™F2812*.

PRÁCTICA # 4

TEMA:

"Comunicación mediante interfaz SPI entre el procesador TMS320F2812 y la memoria EEPROM M95080"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Establecer comunicación SPI entre el procesador *TMS320F2812* y la memoria EEPROM serial M95080.
- Transmitir y almacenar datos de las entradas (GPIO B15 a B8) del procesador *TMS320F2812* hacia una dirección de la EEPROM.
- Leer datos almacenados en la misma dirección de la memoria EEPROM utilizada en el objetivo anterior y mostrar los datos en las salidas (GPIO B7 a B0) del procesador *TMS320F2812*.

REQUISITOS MÍNIMOS:

- Tener Instalado el software Code Composer Studio (versión 3.3) con su respectivo drive para el manejo del convertidor JTAG a USB.
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos de programación en Lenguaje C
- Haber culminado satisfactoriamente la Práctica # 2.
- Lectura y comprensión del archivo de fundamentación teórica: "SERIAL PERIPHERAL INTERFACE (SPI)"

BREVE EXPLICACIÓN DE LA PRÁCTICA

La memoria EEPROM es conocida como memoria no volátil, porque ante una interrupción de la fuente de energía, los datos almacenados no son borrados. Por lo que generalmente se usan para almacenar información sobre programación, estados de las entradas y salidas, fallas, entre otros.

Utilizando la unidad periférica de comunicación SPI, se establecerá la comunicación entre la interfaz SPI del procesador *TMS320F2812* y la memoria EEPROM que es parte de la tarjeta *KSPS-0504 Adaptor Board*. La configuración para el uso de la unidad SPI se la realiza en lenguaje C mediante el software Code Composer Studio en conjunto con las librerías y archivos código fuente provistos por Texas Instruments para el manejo de unidades periféricas del procesador.

Cuando se presiona el pulsador S1 (Conectado GPIO D1), se transmitirá una instrucción para poder escribir en la memoria EEPROM los estados de las entradas GPIO B15 a B8 del procesador *TMS320F2812*. Cuando se presione el pulsador S2 (Conectado GPIO D6), se transmitirá una instrucción para poder leer en la memoria EEPROM los estados de las entradas que se escribieron previamente, y así mostrar los estados en las salidas GPIO B7 a B0. Ver Figura 4.60

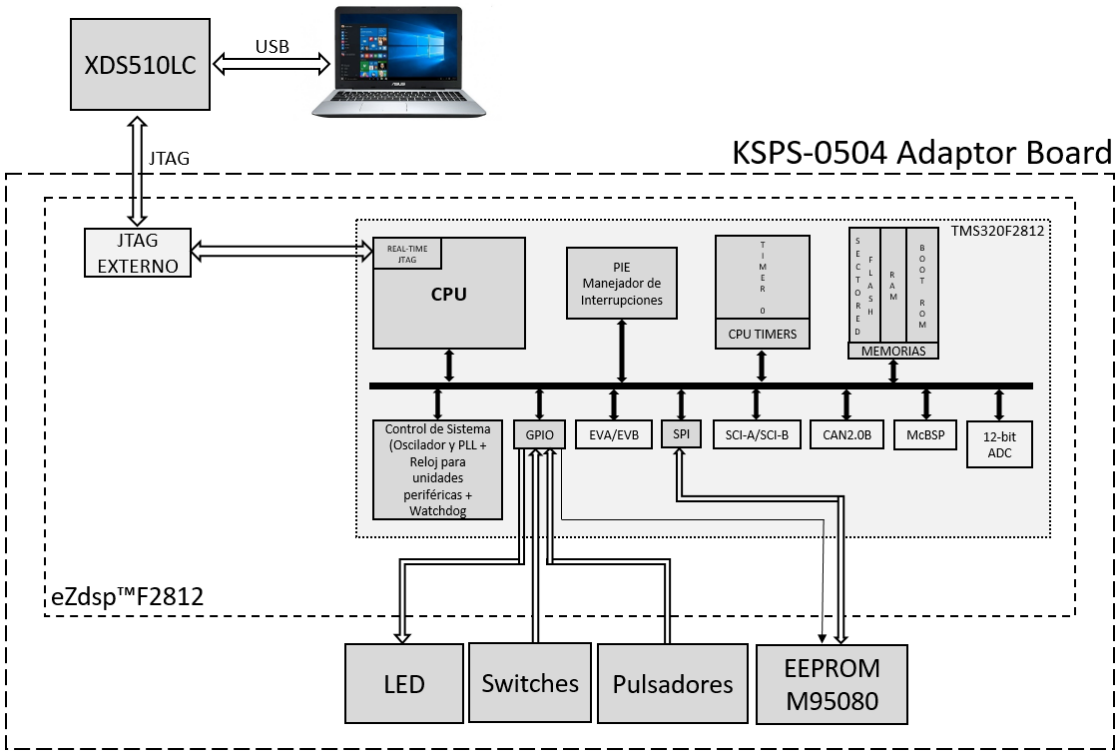


Figura 4.60 Diagrama de Bloques de Comunicación SPI entre DSP y EEPROM

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Code Composer Studio*.

NOTA: Si al abrir el programa sale algún error de conexión con la PC, el motivo podría ser que no esté configurado el *Setup CCStudio V3.3* tal como se ve en la practica 1. Si el estudiante no ha realizado la *practica 1* se recomienda desarrollarla para evitar tener problemas de comunicación entre su PC y las tarjetas, para luego continuar con esta práctica.

3. Crear un proyecto nuevo (*Project -> New..)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica4
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** para crear el proyecto

4. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

// Prototipo de funciones
void Gpio_select(void);
void InitSystem(void);
interrupt void cpu_timer0_isr(void);

// Variables Globales
int dummy;

void main(void)
{
        InitSystem();
        Gpio_select();

        InitPieCtrl();
```

```
        InitPieVectTable();

        EALLOW;
        PieVectTable.TINT0 = &cpu_timer0_isr;
        EDIS;

        InitCpuTimers();
        ConfigCpuTimer(&CpuTimer0, 150, 50000);

        PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
         IER = 1;
        EINT;
        ERTM;

        CpuTimer0Regs.TCR.bit.TSS = 0;

        while(1)
        {
                while(CpuTimer0.InterruptCount < 4);
                CpuTimer0.InterruptCount = 0;
        }
}

void Gpio_select(void)
{
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;
        GpioMuxRegs.GPBMUX.all = 0x0;
         GpioMuxRegs.GPDMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;

        GpioMuxRegs.GPADIR.all = 0x0;
        GpioMuxRegs.GPBDIR.all = 0x00FF;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
        GpioMuxRegs.GPFDIR.all = 0x0;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioDataRegs.GPBDAT.all = 0x0000;

        EDIS;
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00E8;

        SysCtrlRegs.SCSR = 0;

        SysCtrlRegs.PLLCR.bit.DIV = 10;
```

```
        SysCtrlRegs.HISPCP.all = 0x1;
        SysCtrlRegs.LOSPCP.all = 0x2;


        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
}

interrupt void cpu_timer0_isr(void)
{
        CpuTimer0.InterruptCount++;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

5. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica4.c"***

6. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica4.c".***

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

7. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **DSP281x_GlobalVariableDefs.c**

8. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_Headers_nonBIOS.cmd**

9. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

10. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\source"* agregar al proyector los siguientes archivos:

- **DSP281x_PieCtrl.c**
- **DSP281x_PieVect.c**
- **DSP281x_DefaultIsr.c**
- **DSP281x_CpuTimers.c**

11. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL "BUILD OPTIONS"

12. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options..*
- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include
  en el campo *Include Search Path*

13. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **400** en el campo *Stack Size.*
- Dar clic en el botón *OK* para terminar la configuración del *Build Options*

AGREGAR CÓDIGO DE INICIALIZACIÓN SPI

14. Habilitar el reloj de la unidad periférica SPI en la función “InitSystem()”. Ver Figura 4.61

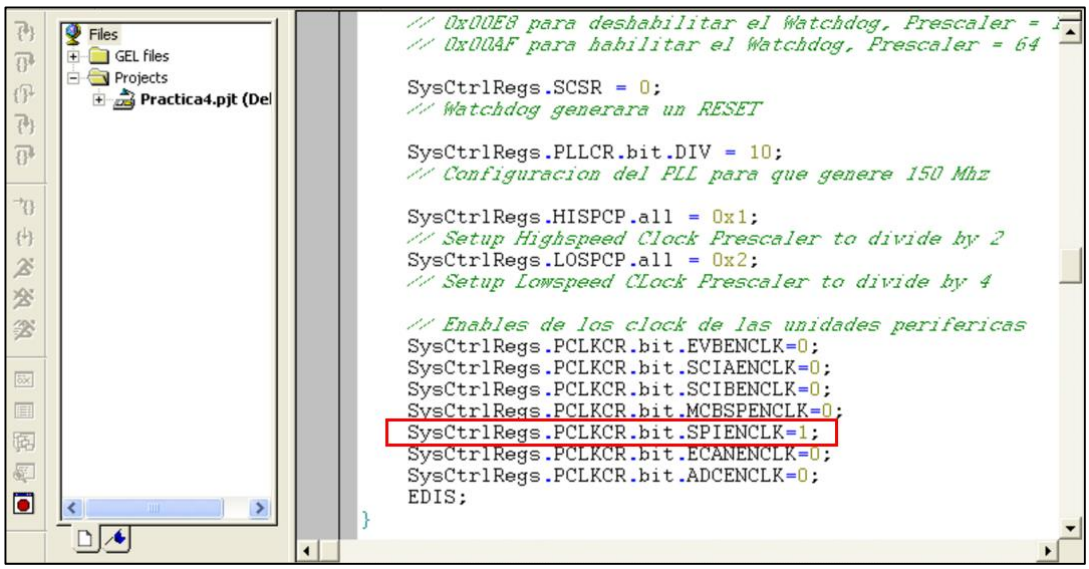


Figura 4.61 Habilitación de Reloj de unidad periférica de comunicación SPI

15. Configurar como función primaria los pines del puerto F en la función “Gpio_select()”, los cuales están asociados a la unidad periférica de comunicación SPI. Ver Figura 4.62

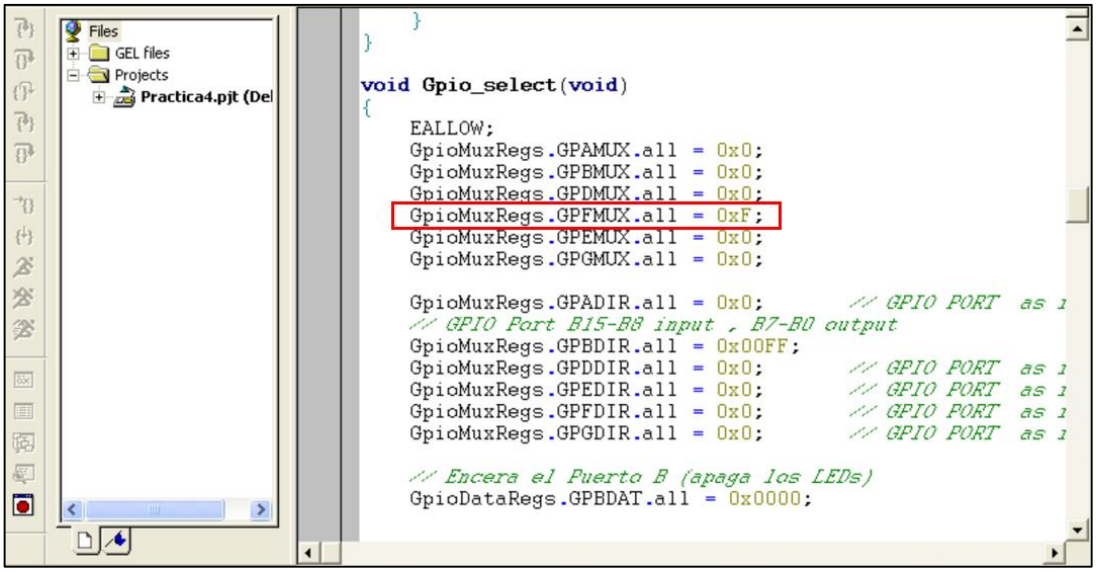


Figura 4.62 Configuración de función primaria en todos los pines del puerto F

16. Configurar los pines GPIO D0 y D5 como pines de salida, los cuales serán los chip-select para el TLV5617A y la EEPROM M95080 respectivamente, como se observa en la Figura 4.63

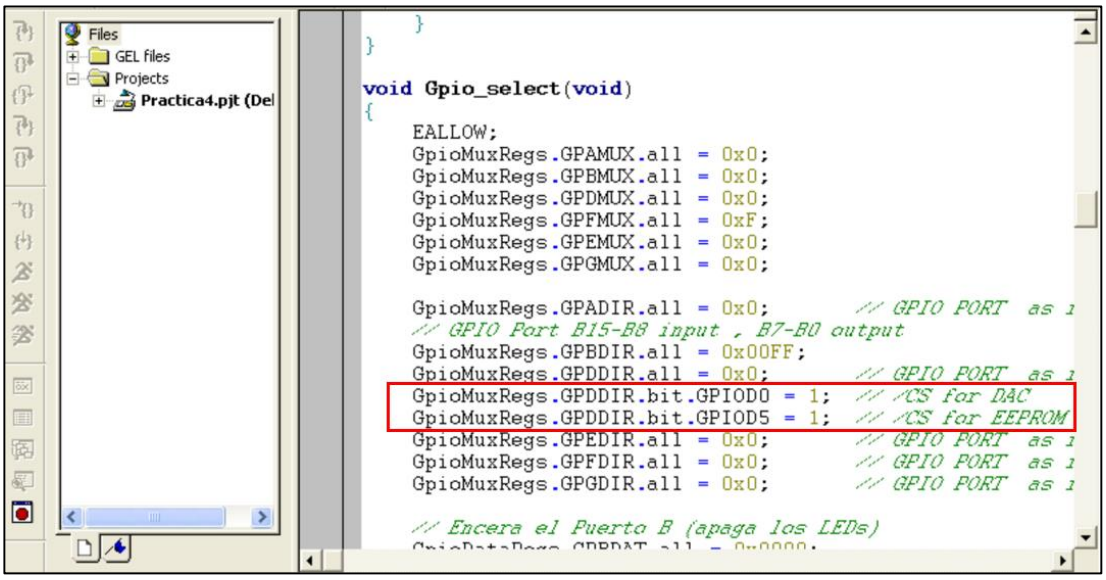


Figura 4.63 Configura como salida los pines GPIOD0 y GPIOD5

17. Inicializar los pines GPIO D0 y D5 en '1', para deshabilitar el chip-select para el TLV5617A y la EEPROM M95080 respectivamente. Ver Figura 4.64

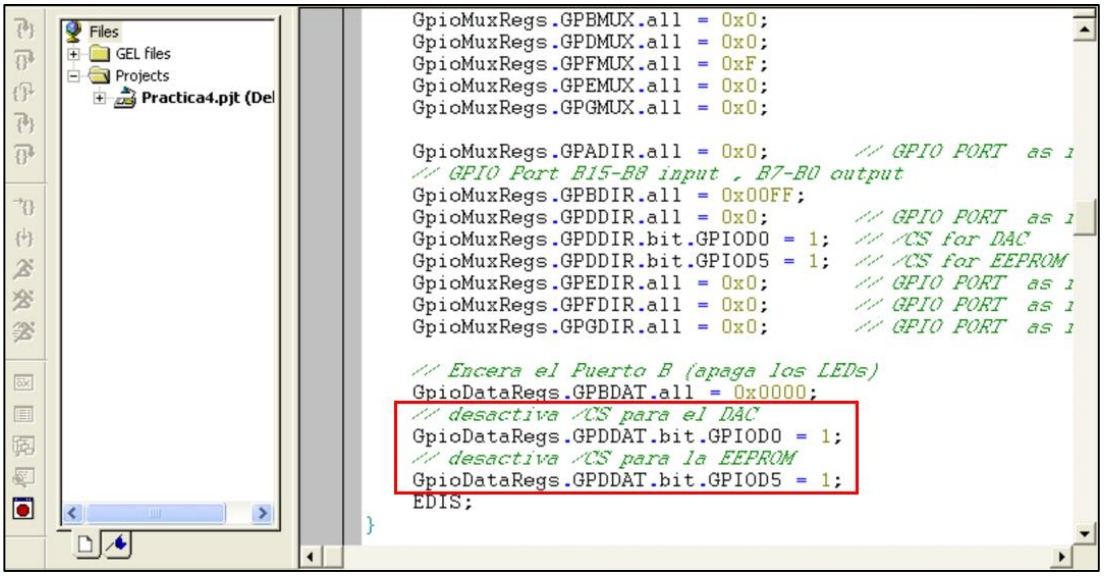


Figura 4.64 Ajuste en '1' de los pines para desactivar los Chip-select del DAC y EEPROM

18. Al inicio del archivo de código fuente ***"Practica4"***, antes del "main" agregar el prototipo de función **"void SPI_Init(void)"**, como se observa en la Figura 4.65

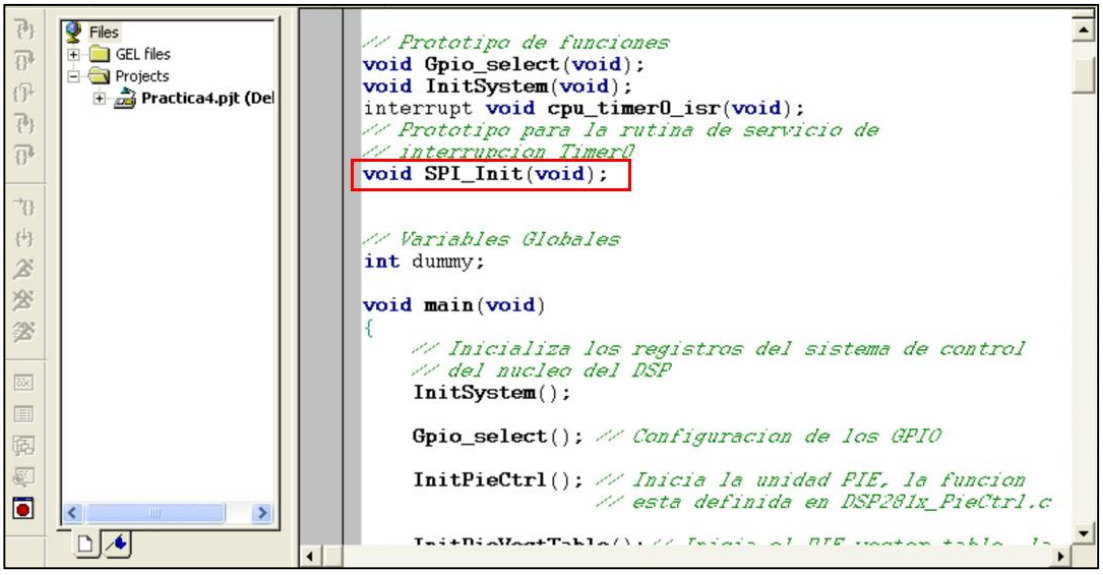


Figura 4.65 Prototipo de función "SPI_Init( )"

19. Agregar la función ***"SPI_Init()"*** en el "main" antes de ingresar al lazo "while(1)". Ver Figura 4.66

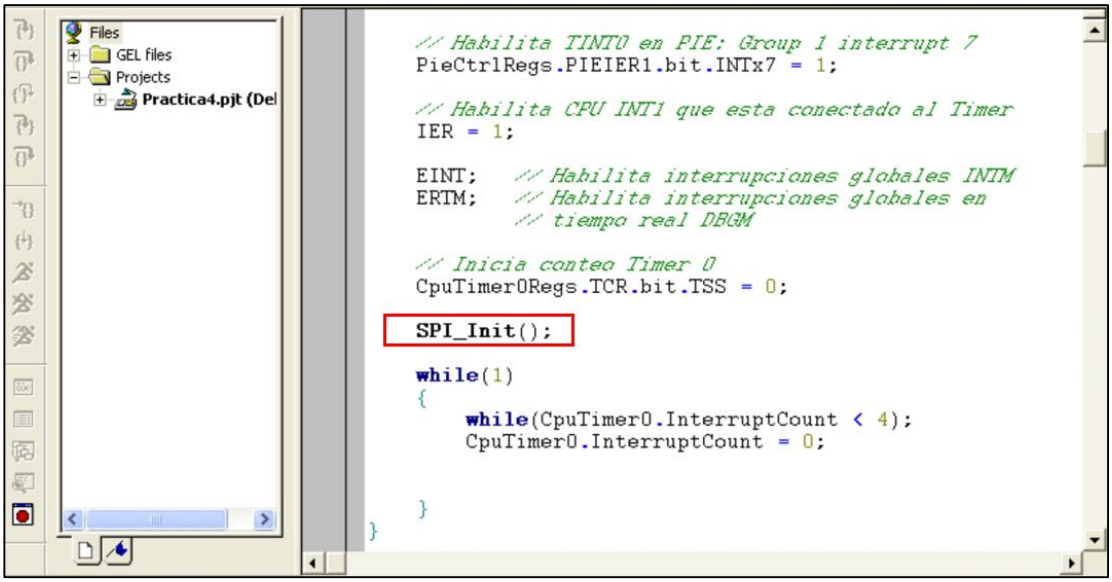


Figura 4.66 Inicialización del módulo SPI

20. Al final del archivo código fuente ***"Practica4"***, agregar la definición de la función **"void SPI_Init(void)"** (Ver Figura 4.67) en base a la siguiente configuración:

Para el registro SPICCR:

- Resetear las banderas del SPI
- Transferir datos cuando se detecte flanco de bajada del reloj
- Configurar 16 bits de datos por trama

Para el registro SPICTL:

- Deshabilitar la interrupción por desbordamiento para recepción de datos
- No generar retardo en SPICLK
- Configurarlo como maestro
- Habilitar Transmisión
- Deshabilitar las interrupciones del SPI para transferencia de datos

Para el registro SPIBRR:

- Configurar la velocidad a 300 KHz, sabiendo que la frecuencia de LSPCLK es de 37.5 MHz.

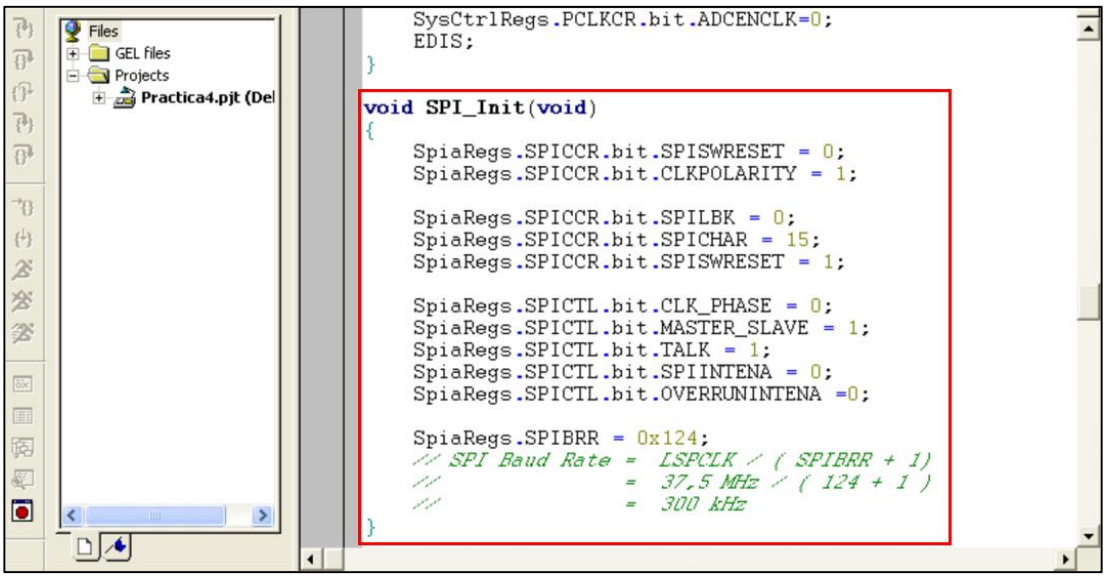


Figura 4.67 Definición de la función "SPI_Init( )"

CREAR FUNCIONES DE ACCESO A EEPROM

21. Definir las variables globales al inicio del archivo de código fuente para el manejo de la EEPROM. Ver Figura 4.68

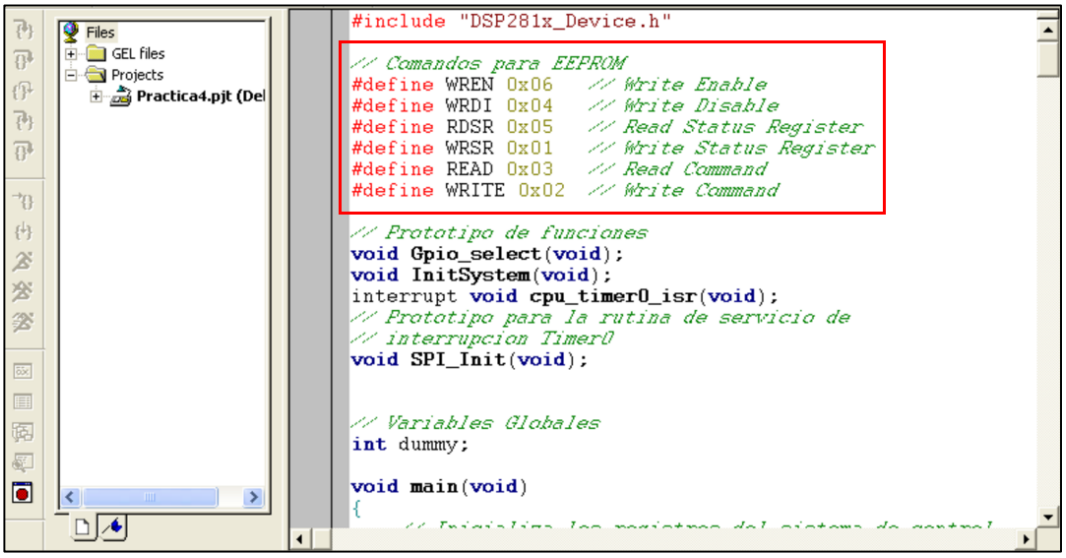


Figura 4.68 Definición de las Variables globales para manejo de la EEPROM

22. Para acceder a la EEPROM, se crearán cuatro funciones específicas, por lo que se debe declarar el prototipo de cada una de ellas antes del “main”. Ver Figura 4.69

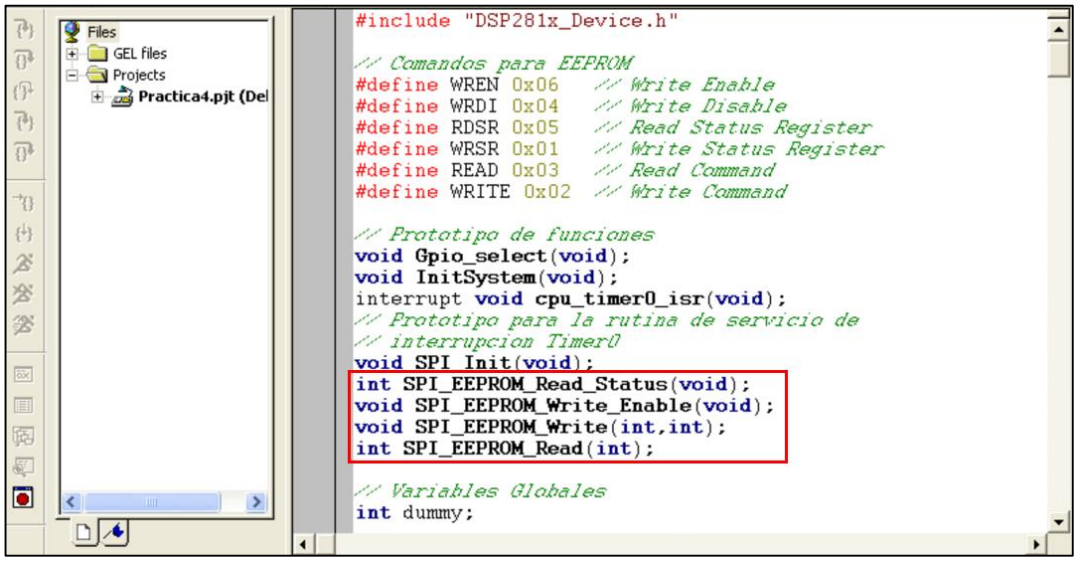


Figura 4.69 Prototipos de funciones para manejo de EEPROM

23. Al final del archivo código fuente, agregar la función **“int SPI_EEPROM_Read_Status(void)”** (Ver Figura 4.70), luego definirlo en base a los siguiente:

- Al inicio de la función, activar la señal “Chip Select” de la EEPROM. Para ello: **GpioDataRegs.GPDDAT.bit.GPIOD5 = 0;**
- Transmitir la instrucción para habilitar la lectura del registro de estado mediante la carga de RDSR en SPITXBUF. Se debe tener cuidado con la correcta alineación de los 8 bits de esta instrucción al ser cargada en SPITXBUF.
- Antes de poder continuar se debe esperar que termine la transmisión SPI. Para ellos preguntaremos por el estado de la bandera SPI INT de la siguiente manera: **while(SpiaRegs.SPISTS.bit.INT_FLAG = = 0);**
- Ahora ya se puede leer el “Status Register” en SPIRXBUF
- Desactivar la señal “Chip Select” de la EEPROM. Para ello: **GpioDataRegs.GPDDAT.bit.GPIOD5 = 1;**
- Retornar la lectura del “Status Register”.

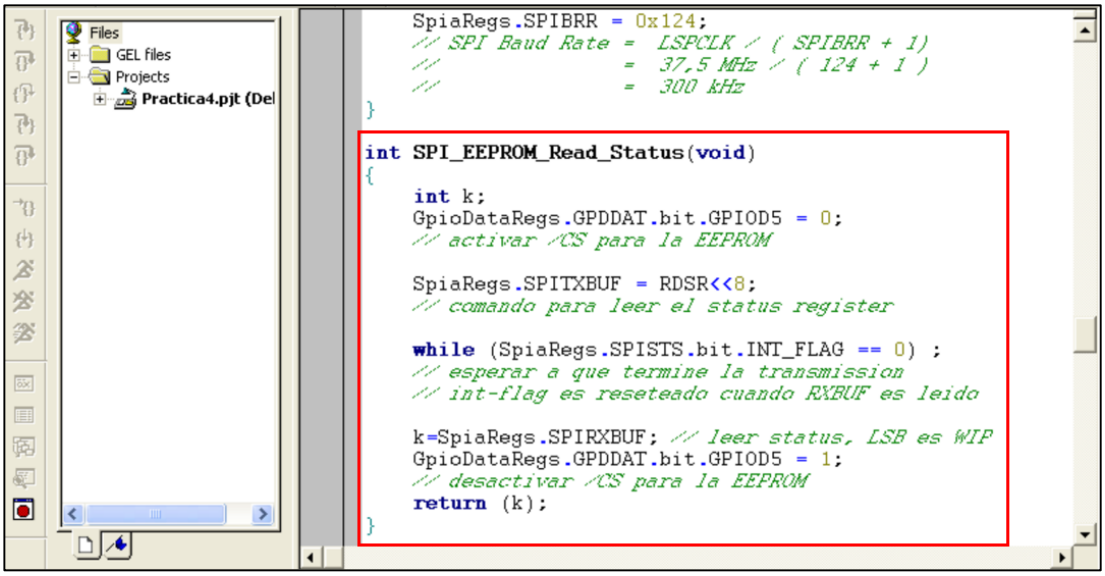


Figura 4.70 Definición de función "SPI_EEPROM_Read_Status( )"

24. Al final del archivo código fuente, agregar la función **“void SPI_EEPROM_Write_Enable(void)”** que servirá para habilitar la escritura en la EEPROM (Ver Figura 4.71). En su definición se debe tomar en cuenta lo siguiente:

- Al inicio de la función, activar la señal “Chip Select” de la EEPROM. Para ello: **GpioDataRegs.GPDDAT.bit.GPIOD5 = 0;**
- Transmitir la instrucción para habilitar la escritura mediante la carga de WREN en SPITXBUF. Se debe tener cuidado con la correcta alineación de los 8 bits de esta instrucción al ser cargada en SPITXBUF.
- Antes de poder continuar se debe esperar que termine la transmisión SPI. Para ellos preguntaremos por el estado de la bandera SPI INT de la siguiente manera: **while(SpiaRegs.SPISTS.bit.INT_FLAG == 0);**
- Ejecutar la lectura de SPIRXBUF para resetear la bandera SPI INT
- Desactivar la señal “Chip Select” de la EEPROM. Para ello: **GpioDataRegs.GPDDAT.bit.GPIOD5 = 1;**

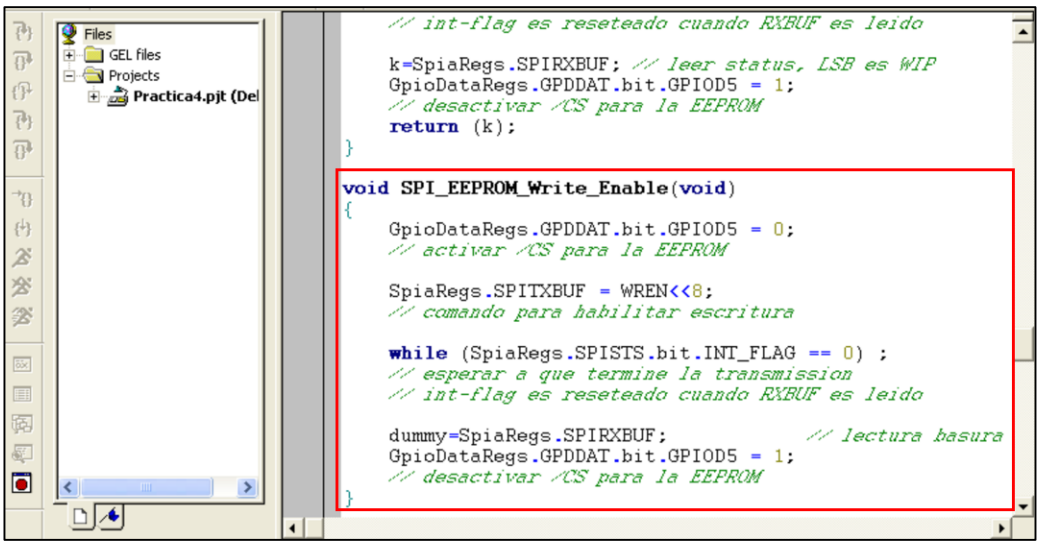


Figura 4.71 Definición de función "SPI_EEPROM_Write_Enable( )"

25. Al final del archivo código fuente, agregar la **función “void SPI_EEPROM_Write(int address, int data)”** que servirá para escribir un caracter de 8 bits en la EEPROM (Ver Figura 4.72). El caracter de 8 bits lleva la información del

estado de los switches (conectados a GPIO B15…B8). En su definición se debe tomar en cuenta lo siguiente:

- Al inicio de la función, activar la señal “Chip Select” de la EEPROM. Para ello: **GpioDataRegs.GPDDAT.bit.GPIOD5 = 0;**
- Transmitir la instrucción de escritura mediante la carga de WRITE, adicionado de los 8 bits más significativos de la dirección en SPITXBUF. Se debe tener cuidado con la correcta alineación de los 8 bits de esta instrucción al ser cargada en SPITXBUF.
- Antes de poder continuar se debe esperar que termine la transmisión SPI. Para ellos preguntaremos por el estado de la bandera SPI INT de la siguiente manera: **while(SpiaRegs.SPISTS.bit.INT_FLAG = = 0);**
- Ejecutar la lectura de SPIRXBUF para resetear la bandera SPI INT
- Transmitir los 8 bits menos significativos de la dirección, adicionado del dato en SPITXBUF. Se debe tener cuidado con la correcta alineación de los 8 bits de esta instrucción al ser cargada en SPITXBUF.
- Antes de poder continuar se debe esperar que termine la transmisión SPI. Para ellos preguntaremos por el estado de la bandera SPI INT de la siguiente manera: **while(SpiaRegs.SPISTS.bit.INT_FLAG = = 0);**
- Ejecutar la lectura de SPIRXBUF para resetear la bandera SPI INT
- Desactivar la señal “Chip Select” de la EEPROM. Para ello: **GpioDataRegs.GPDDAT.bit.GPIOD5 = 1;**

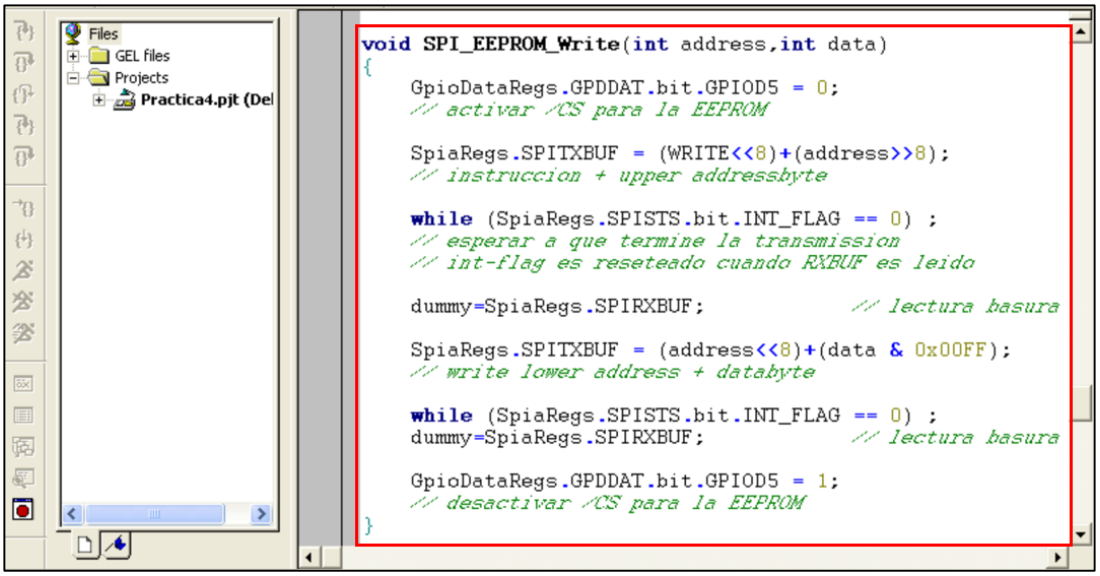


Figura 4.72 Definición de función "SPI_EEPROM_Write( )"

26. Al final del archivo código fuente, agregar la función **“int SPI_EEPROM_Read(int address)”**, esta función retorna la lectura del dato guardado en la dirección 0x40 de la EEPROM (Ver Figura 4.73). En su definición se debe tomar en cuenta lo siguiente:

- Al inicio de la función, activar la señal “Chip Select” de la EEPROM. Para ello: **GpioDataRegs.GPDDAT.bit.GPIOD5 = 0;**
- Transmitir la instrucción de lectura mediante la carga de READ, adicionado de los 8 bits más significativos de la dirección en SPITXBUF. Se debe tener cuidado con la correcta alineación de los 8 bits de esta instrucción al ser cargada en SPITXBUF.
- Antes de poder continuar se debe esperar que termine la transmisión SPI. Para ellos preguntaremos por el estado de la bandera SPI INT de la siguiente manera: **while(SpiaRegs.SPISTS.bit.INT_FLAG = = 0);**
- Ejecutar la lectura de SPIRXBUF para resetear la bandera SPI INT
- Transmitir los 8 bits menos significativos de la dirección en SPITXBUF. Se debe tener cuidado con la correcta alineación de los 8 bits de esta instrucción al ser cargada en SPITXBUF.

- Antes de poder continuar se debe esperar que termine la transmisión SPI. Para ellos preguntaremos por el estado de la bandera SPI INT de la siguiente manera: **while(SpiaRegs.SPISTS.bit.INT_FLAG = = 0);**
- Ejecutar la lectura de SPIRXBUF para leer los datos requeridos de la EEPROM
- Desactivar la señal “Chip Select” de la EEPROM. Para ello: **GpioDataRegs.GPDDAT.bit.GPIOD5 = 1;**
- Retornar la lectura del dato

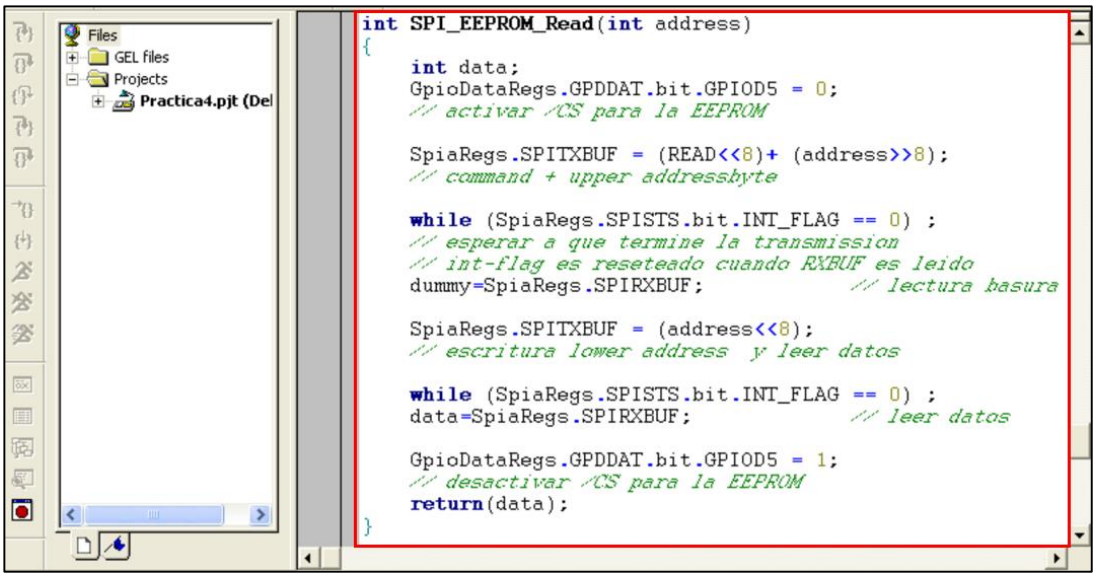


Figura 4.73 Definición de función "SPI_EEPROM_Read"

PROGRAMACIÓN DEL “MAIN LOOP”

El objetivo de esta práctica de laboratorio es almacenar el estado de los 8 switches de entrada (GPIO B15 a B8) en la dirección 0x40 de la EEPROM cuando sea pulsado el botón S1 (GPIO D1). Si S2 (GPIO D6) es pulsado, el programa debe leer la dirección 0x40 de la EEPROM y cargar el valor en los 8 led’s (GPIO B7 a B0)

27. Cuando es pulsado el botón S1:

- Habilitar la escritura en la EEPROM llamando la función “SPI_EEPROM_Write_Enable”

- Verificar que la bandera “WEL” sea igual a 1, llamando a la función “SPI_EEPROM_Read_Status”
- Escribir el estado de los 8 switches en la dirección 0x40 de la EEPROM con la ayuda de la función “SPI_EEPROM_Write”
- Esperar a que termine la escritura preguntando por la bandera “WIP” del “Status Register”. Ver Figura 4.74

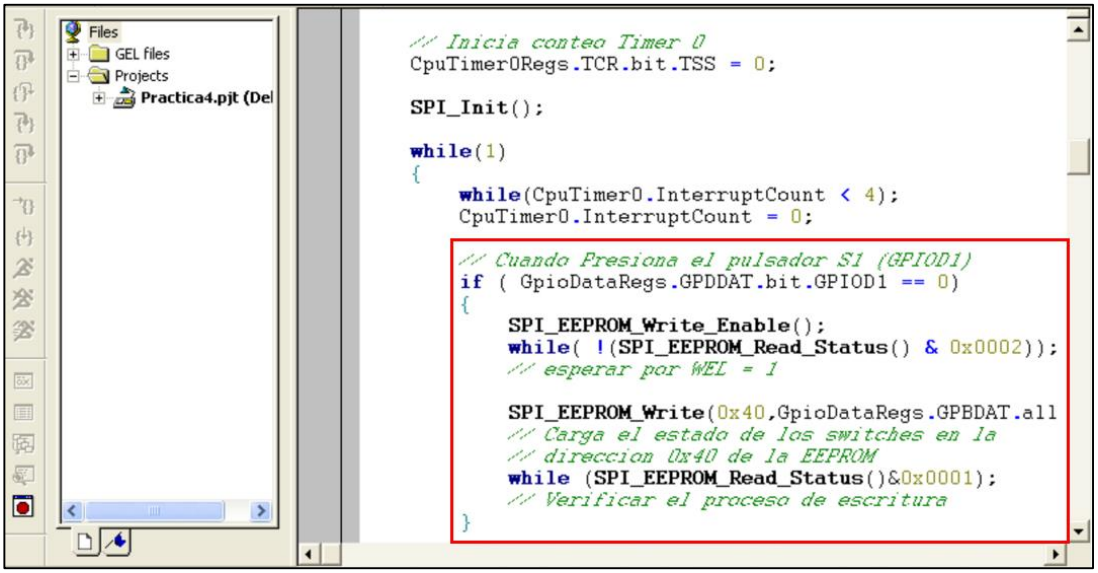


Figura 4.74 Programación de “while(1)" cuando se presiona S1

28. Cuando es pulsado el botón S2:

- Llamar a la función “SPI_EEPROM_Read( )” y transferir los 8 bits retornados a los 8 led’s conectados en GPIO B7 a B0. Ver Figura 4.75

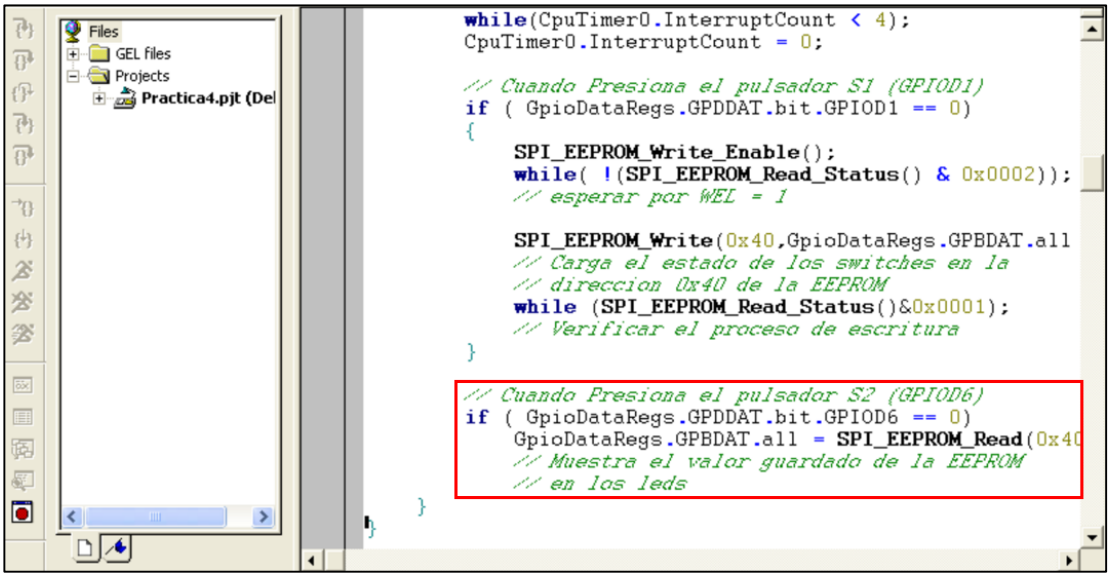


Figura 4.75 Programación del "while(1)" cuando se presiona S2

COMPILAR, CARGAR Y EJECUTAR EL PROGRAMA

29. Compilar el proyecto seleccionando*: Project -> Rebuild All*. Solucionar los errores en caso de haberlos.

30. alimentar la tarjeta *eZdsp™F2812*.

31. Conectarse con el procesador: *Debug -> Connect*

32. Cargar el programa (*File -> Load Program…)* **Practica4.out** que se genera dentro de la carpeta Debug al compilar el proyecto

33. Ejecutar el programa cargado: *Debug -> Run*

En la Figura 4.76 se puede observar que se presiona el pulsador S1 (Conectado GPIO D1), cuando esto sucede se transmitirá una instrucción para poder escribir en la memoria EEPROM los estados de las entradas GPIO B15 a B8 del procesador *TMS320F2812*.

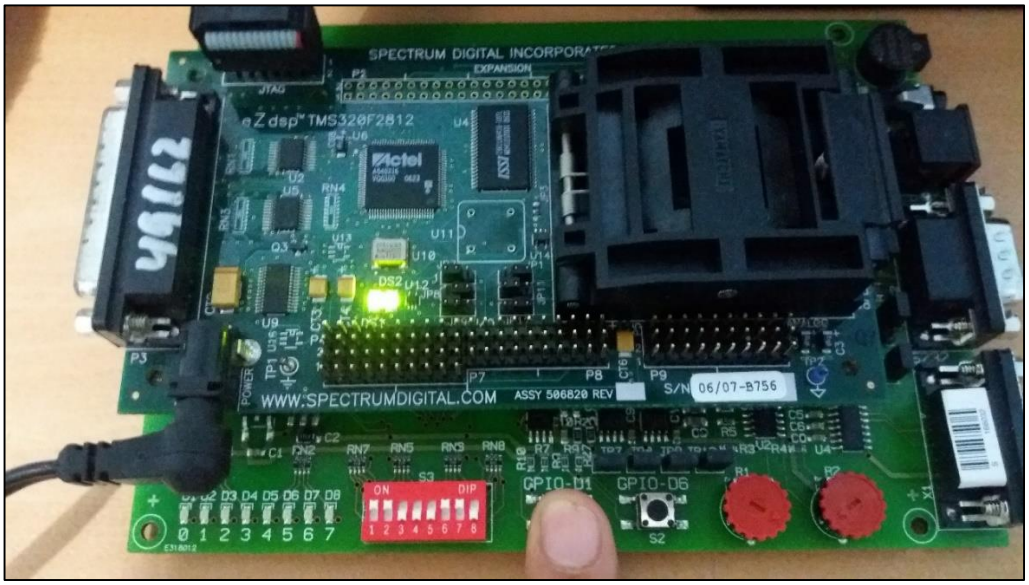


Figura 4.76 Foto del Kit cuando se presiona S1

Cuando se presione el pulsador S2 (Conectado GPIO D6), se transmitirá una instrucción para poder leer en la memoria EEPROM los estados de las entradas que se escribieron previamente, y así mostrar los estados en las salidas GPIO B7 a B0. Ver Figura 4.77

Figura 4.77 Foto del Kit cuando se presiona S2

34. Al finalizar la práctica, detener la ejecución del programa: *(Debug -> Halt)*

35. Resetear el CPU *(Debug -> Reset CPU)*, luego desconectarse de la tarjeta *(Debug -> Disconnect).* Y quitar la alimentación de la tarjeta *eZdsp™F2812*.

PRÁCTICA # 5

TEMA:

"Transmisión de datos mediante interfaz SCI entre el procesador TMS320F2812 y la PC"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Configurar el formato de la trama y la velocidad para una comunicación SCI.
- Establecer comunicación SCI entre el procesador *TMS320F2812* y la PC.
- Transmitir un texto desde el procesador *F2812* hacia el puerto seria de la computadora.

REQUISITOS MÍNIMOS:

- Tener Instalado el software Code Composer Studio (versión 3.3) con su respectivo drive para el manejo del convertidor JTAG a USB.
- Tener instalado el programa *HyperTerminal*
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos de programación en Lenguaje C
- Haber culminado satisfactoriamente la Práctica # 1.
- Lectura y comprensión del archivo de fundamentación teórica: "SERIAL COMMUNICATION INTERFACE (SCI)".

BREVE EXPLICACIÓN DE LA PRÁCTICA

Serial Communication Interface (SCI) es una interfaz asíncrona también conocida como UART (Universal Asynchronous Receiver-Transmitter). La unidad periférica SCI soporta una comunicación digital entre el CPU y otros periféricos asíncronos

externos que usen el formato estándar non-return-to-zero (NRZ). La unidad SCI puede operar tanto en comunicación half-duplex como full-duplex.

Utilizando la unidad periférica SCI, se establecerá la comunicación entre el procesador *TMS320F2812* y el puerto serial de la PC. La configuración para el uso de la unidad periférica SCI se la realiza por programa mediante el software Code Composer Studio. Para poder establecer la comunicación SCI se debe configurar también la trama y velocidad en el puerto COM de la PC y en el programa HyperTerminal.

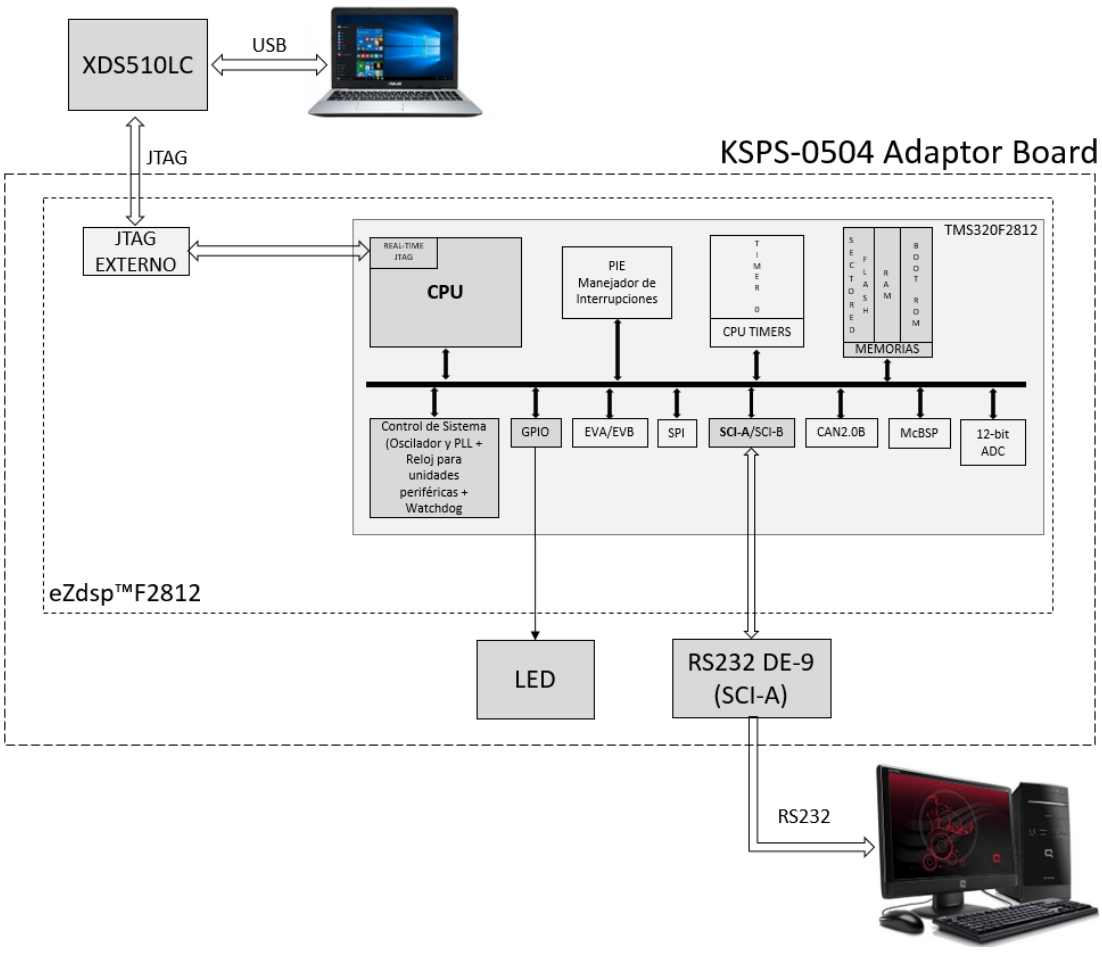


Figura 4.78 Diagrama de Bloques de Comunicación SCI entre DSP y PC

Se transmitirá la frase “F2812 está comunicando \n\r” mediante la interfaz RS232 el cual se podrá visualizar en la ventana principal del programa HyperTerminal cada 2 segundos aproximadamente. El retardo de tiempo se generará por software. El LED conectado a GPIO B0 será el indicador de la transmisión de datos, es decir, cada vez que se esté transmitiendo caracteres del texto el LED estará prendido. Ver Figura 4.78

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Code Composer Studio*.

NOTA: Si al abrir el programa sale algún error de conexión con la PC, el motivo podría ser que no esté configurado el *Setup CCStudio V3.3* tal como se ve en la practica 1. Si el estudiante no ha realizado la *practica 1* se recomienda desarrollarla para evitar tener problemas de comunicación entre su PC y las tarjetas, para luego continuar con esta práctica.

3. Crear un proyecto nuevo (*Project -> New..)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica5
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** para crear el proyecto

4. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

//Prototipo de Funciones
```

```
void Gpio_select(void);
void InitSystem(void);

void main(void)
{
        InitSystem();

        Gpio_select();

        while(1)
        {
        }
}

void Gpio_select(void)
{
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;
        GpioMuxRegs.GPBMUX.all = 0x0;
        GpioMuxRegs.GPDMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;

        GpioMuxRegs.GPADIR.all = 0x0;
        GpioMuxRegs.GPBDIR.all = 0x00FF;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
        GpioMuxRegs.GPFDIR.all = 0x0;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioDataRegs.GPBDAT.all = 0x0000;
        EDIS;
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00E8;

        SysCtrlRegs.SCSR = 0;
        SysCtrlRegs.PLLCR.bit.DIV = 10;

        SysCtrlRegs.HISPCP.all = 0x1;
        SysCtrlRegs.LOSPCP.all = 0x2;
        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;// reloj para SCI-A
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
```

```
        EDIS;
}
```

5. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica5.c"***

6. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica5.c".***

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

7. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **DSP281x_GlobalVariableDefs.c**

8. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_Headers_nonBIOS.cmd**

9. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

10. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL "BUILD OPTIONS"

11. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options..*

- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include
  en el campo *Include Search Path*

12. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **400** en el campo *Stack Size.*
- Dar clic en el botón *OK* para terminar la configuración del *Build Options*

AGREGAR CÓDIGO DE INICIALIZACIÓN SCI

13. Habilitar el reloj de la unidad periférica SCI en la función “InitSystem()”. Ver Figura 4.79

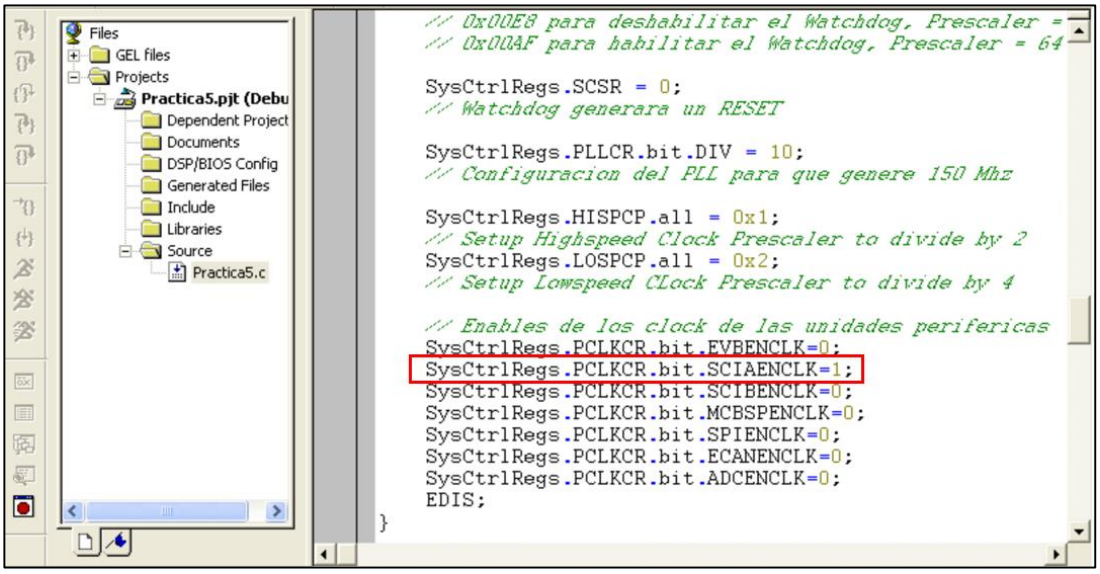


Figura 4.79 Habilitación de Reloj de unidad periférica de comunicación SCI

14. Configurar como función primaria los pines del puerto F en la función “Gpio_select()”, los cuales están asociados a la unidad periférica de comunicación SCI. Ver Figura 4.80

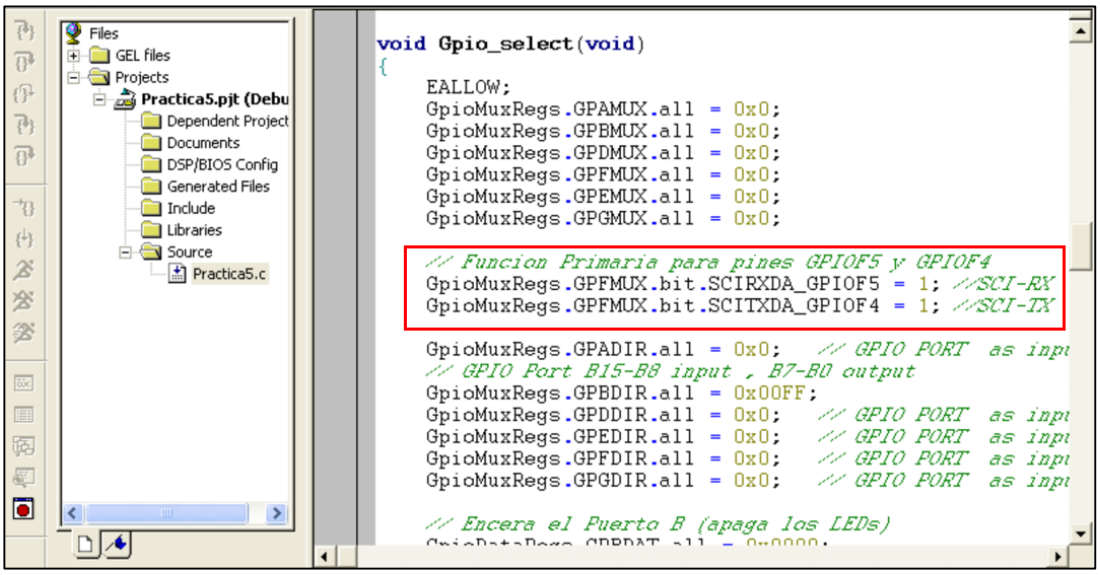


Figura 4.80 Configuración de función primaria en los pines GPIOF4 y GPIOF5

15. Definir las variables en el “main” que se usaran en la práctica. Ver Figura 4.81

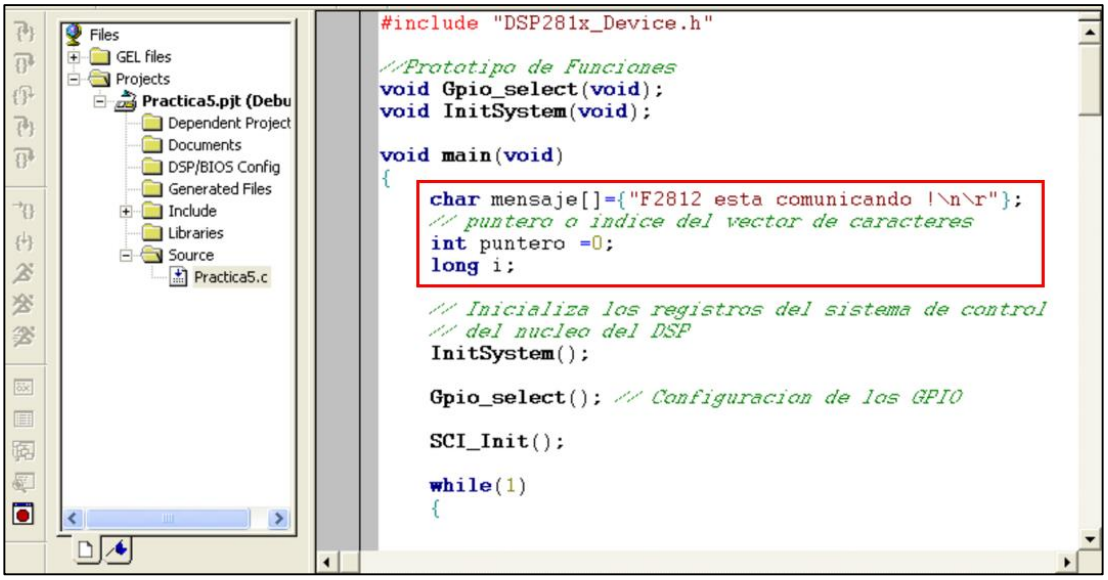


Figura 4.81 Declaración de Variables a usar en la práctica

16. Al inicio del archivo de código fuente ***"Practica5"***, antes del "main" agregar el prototipo de función **"void SCI_Init(void)"**, como se observa en la Figura 4.82

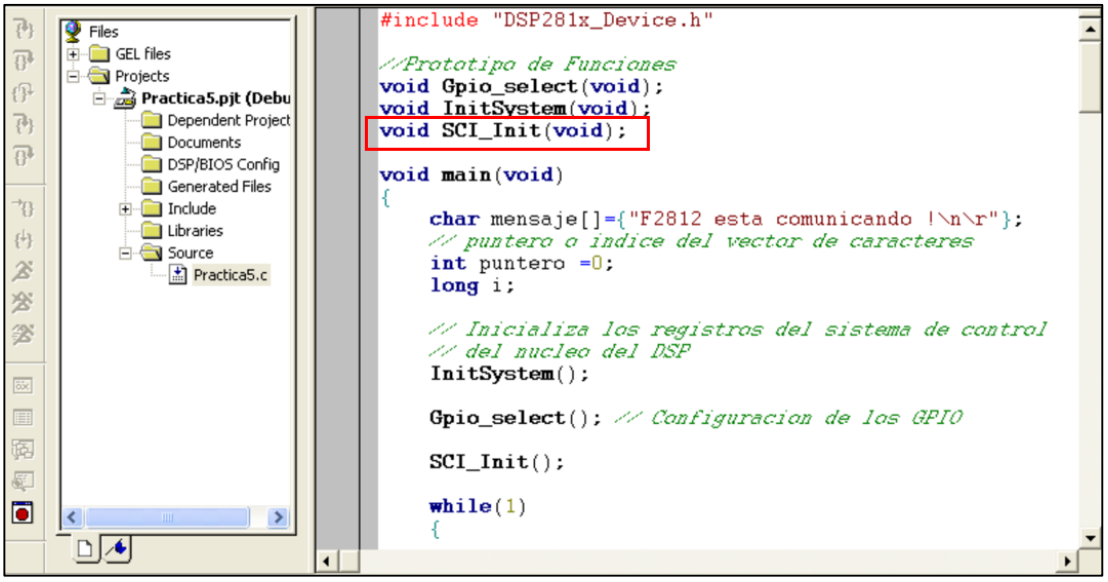


Figura 4.82 Prototipo de función "SCI_Init( )"

17. Agregar la función ***"SCI_Init()"*** en el "main" antes de ingresar al lazo "while(1)". Como se observa en la Figura 4.83

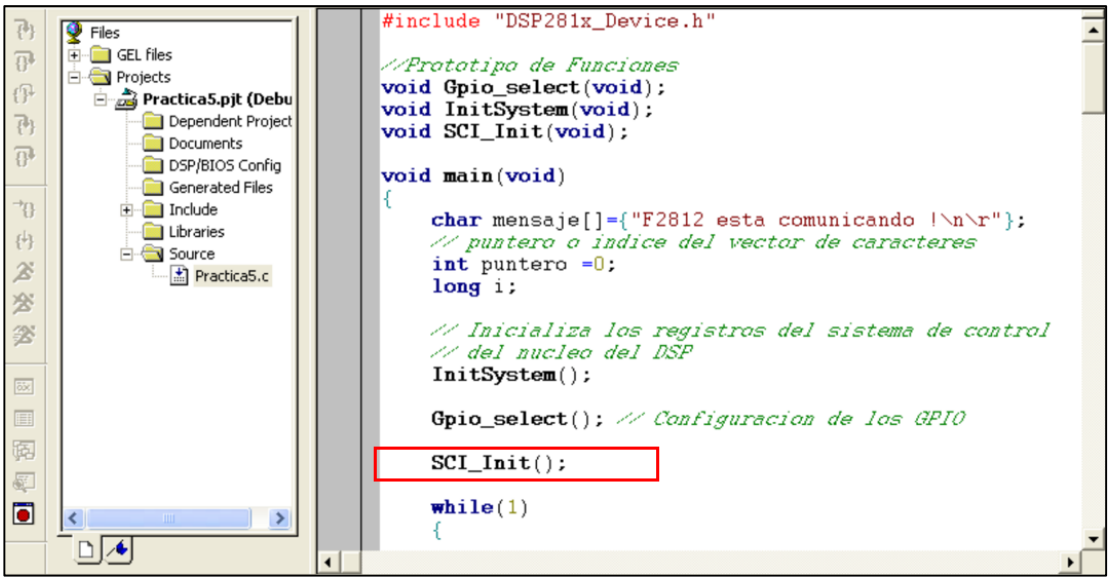


Figura 4.83 Llamado de función "SCI_Init( )" en el "main"

18. Al final del archivo código fuente ***"Practica5"***, agregar la definición de la función **"void SCI_Init(void)"** (Ver Figura 4.84) en base a la siguiente configuración:

Para el registro SCICCR:

- Configurar un bit de STOP
- Deshabilitar paridad y loop back
- Configurar 8 bits de datos por carácter
- Seleccionar modo Idle-line

Para el registro SCICTL1:

- Resetear SCI
- Habilitar transmisión y recepción
- Deshabilitar modo SLEEP, TXWAKE e interrupciones por errores de recepción.

Para el registro SCIHBAUD / SCILBAUD:

- Configurar la velocidad a 9600 baud rate, sabiendo que la frecuencia de LSPCLK es de 37.5 MHz.

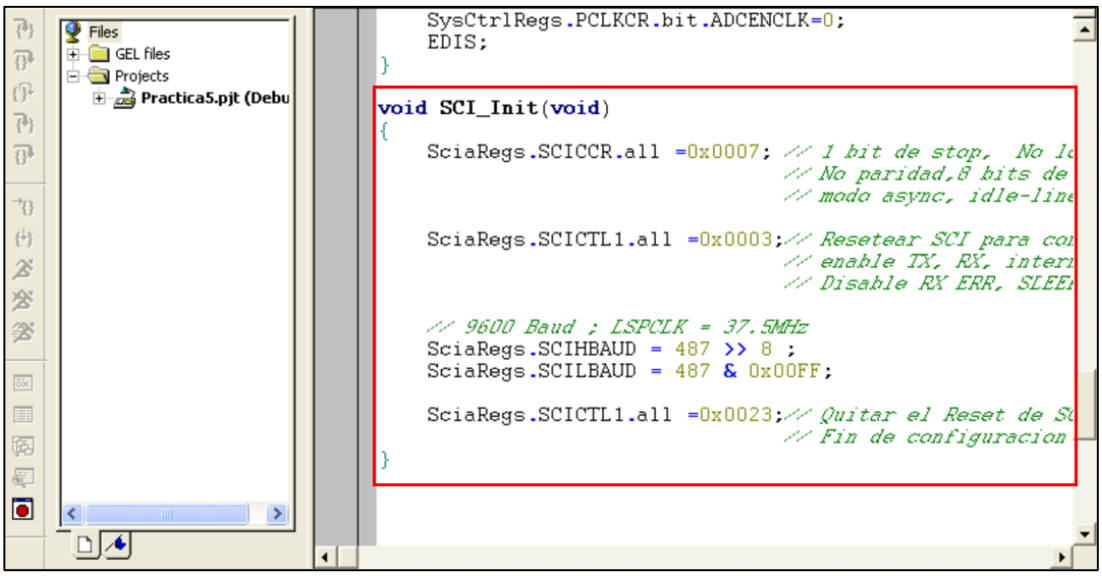


Figura 4.84 Definición de función "SCI_Init( )"

PROGRAMACIÓN DEL “MAIN LOOP”

19. En el interior del “while(1)”:

- Cargar el caracter a transmitir en el registro SCITXBUF
- Esperar que haya terminado la transmisión mediante la verificación del bit TXEMPTY
- Incrementar en uno el valor del índice que recorre el string para poder cargar el siguiente caracter en el registro SCITXBUF y esperar a que termine la transmisión. Esta actualización se utilizará para transmitir el resto de caracteres del string.
- Cuando termine la transmisión del último caracter, generar un retardo de tiempo de aproximadamente 2 segundos por software mediante la siguiente instrucción:

  for(i=0;i<15000000;i++);.

En la Figura 4.85 se muestra la programación en lenguaje C para transmitir el texto cada 2 segundos aproximadamente.

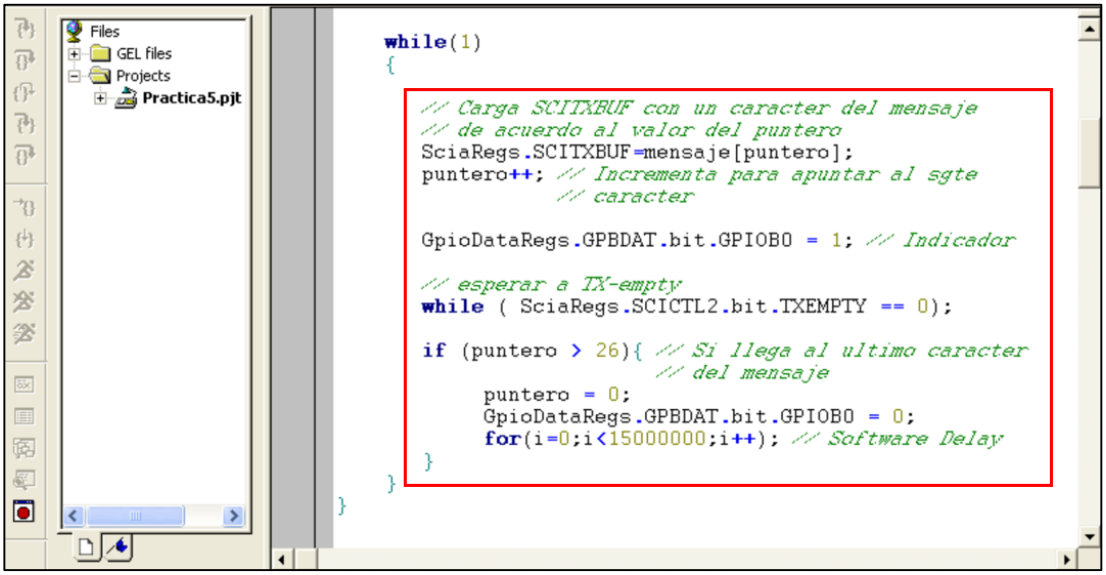


Figura 4.85 Programación del lazo "while(1)"

COMPILAR, CARGAR Y EJECUTAR EL PROGRAMA

20. Compilar el proyecto seleccionando*: Project -> Rebuild All*. Solucionar los errores en caso de haberlos.

21. Alimentar la tarjeta *eZdsp™F2812*.

22. Conectarse con el procesador: *Debug -> Connect*

23. Cargar el programa (*File -> Load Program…)* **Practica5.out** que se genera dentro de la carpeta Debug al compilar el proyecto

24. Ejecutar el programa cargado: *Debug -> Run*

En la Figura 4.86 se observa la ejecución de la práctica, donde el led D1 se encenderá cada vez que se esté transmitiendo datos al PC.

Figura 4.86 Foto del Kit mientras se ejecuta la práctica

25. Configurar las propiedades del puerto COM de la PC de acuerdo al formato de la trama SCI. Como se observa en la Figura 4.87

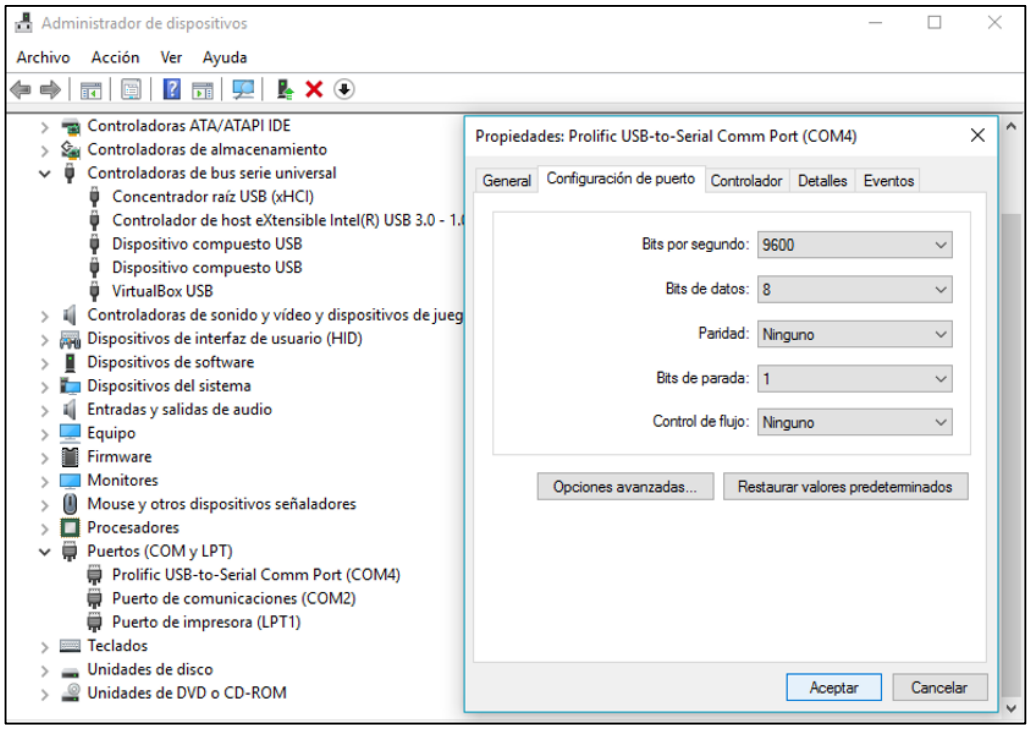


Figura 4.87 Configuración del puerto COM de la PC

26. Abrir programa HyperTerminal para colocar nombre del proyecto y luego dar click en **OK**. Como puede observar en la Figura 4.88

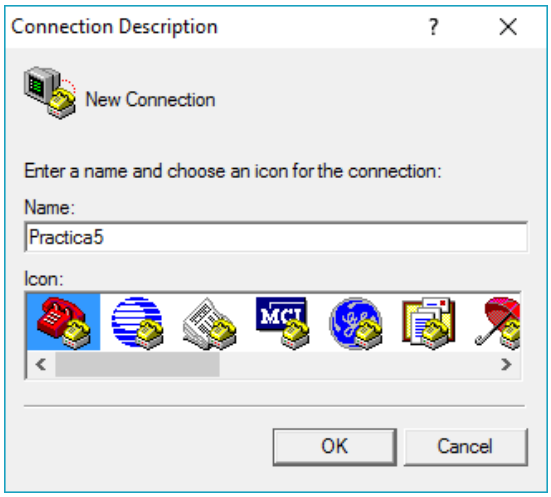


Figura 4.88 Creación de proyecto en programa HyperTerminal

27. Al abrirse la ventana “Connect To”, se debe seleccionar el puerto por el cual se realizará la comunicación para luego dar click en **Configure…**

28. Configurar las propiedades del puerto COM del programa HyperTerminal de acuerdo al formato de la trama SCI. Como se observa en la Figura 4.89

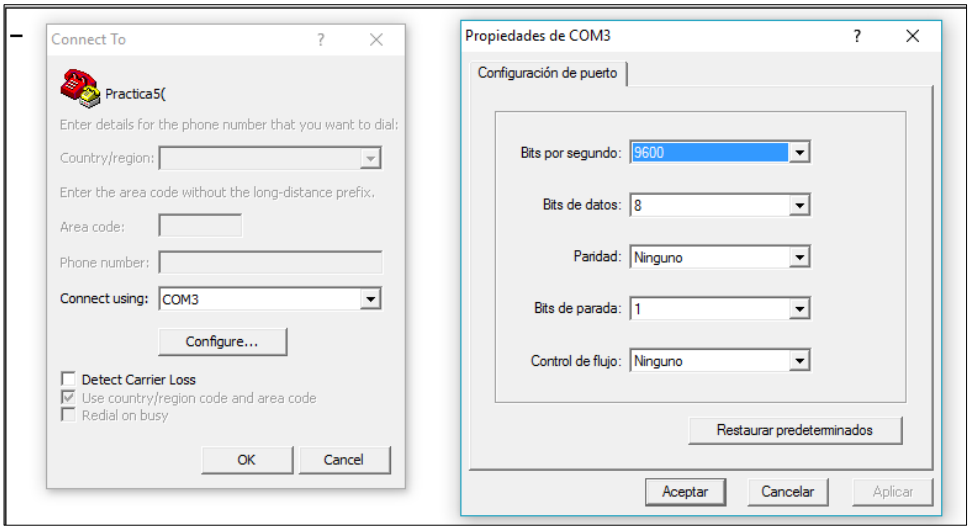


Figura 4.89 Establecer para que HyperTerminal use el puerto COM de la PC

29. Dar click en **Aceptar** y luego en **OK** para poder visualizar el mensaje recibido que ha sido transmitido por la DSP.

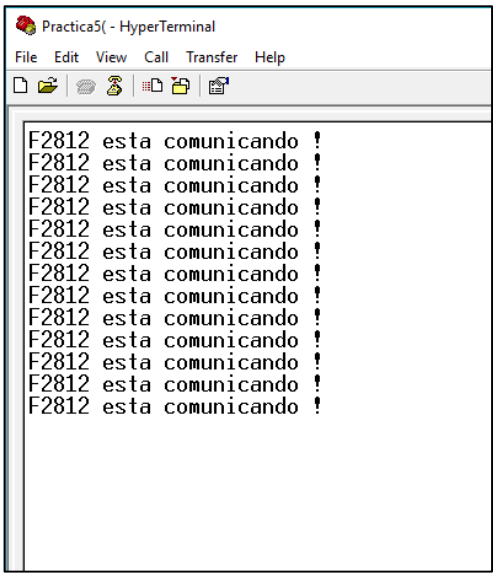


Figura 4.90 Visualización de los caracteres transmitidos

En la Figura 4.90 se observa los caracteres del mensaje transmitido desde el DSP *TMS320F2812* hacia el Computador mediante la unidad periférica de comunicación SCI del procesador.

30. Al finalizar la práctica, detener la ejecución del programa: *(Debug -> Halt)*

31. Resetear el CPU *(Debug -> Reset CPU)*, luego desconectarse de la tarjeta *(Debug -> Disconnect).*

32. Quitar la alimentación de la tarjeta *eZdsp™F2812*.

PRÁCTICA # 6

TEMA:

"Transmisión de datos a una PC mediante la interfaz de comunicación SCI y el sistema de interrupciones del procesador TMS320F2812"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Configurar el formato de la trama, la velocidad, y habilitar el sistema de interrupciones para una comunicación SCI.
- Establecer comunicación SCI entre el procesador *TMS320F2812* y la PC.
- Transmitir un texto desde el procesador *F2812* hacia el puerto serial de la computadora cada 2 segundos.

REQUISITOS MÍNIMOS:

- Tener Instalado el software Code Composer Studio (versión 3.3) con su respectivo drive para el manejo del convertidor JTAG a USB.
- Tener instalado el programa *HyperTerminal*
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos de programación en Lenguaje C
- Haber culminado satisfactoriamente la Práctica # 2 y 5.
- Lectura y comprensión del archivo de fundamentación teórica: "SISTEMAS DE INTERRUPCIONES DEL DSP TMS320F2812"
- Lectura y comprensión del archivo de fundamentación teórica: "SERIAL COMMUNICATION INTERFACE (SCI)"

BREVE EXPLICACIÓN DE LA PRÁCTICA

Las interrupciones permiten detectar eventos en cualquier momento sin importar en que sección de código se encuentre. Debido a esto el uso de interrupciones es de

importancia para el correcto manejo de las unidades periféricas de los DSP como lo son: el manejador de eventos, convertidor analógico a digital, temporizadores, SPI, SCI, CAN, entre otros.

Utilizando la unidad periférica SCI, y el manejo de interrupciones se establecerá la comunicación entre en procesador *TMS320F2812* y el puerto serial de la PC. La configuración para el uso de la unidad periférica SCI se la realiza por programa mediante el software Code Composer Studio. Para poder establecer la comunicación SCI se debe configurar también la trama y velocidad en el puerto COM de la PC y en el programa HyperTerminal.

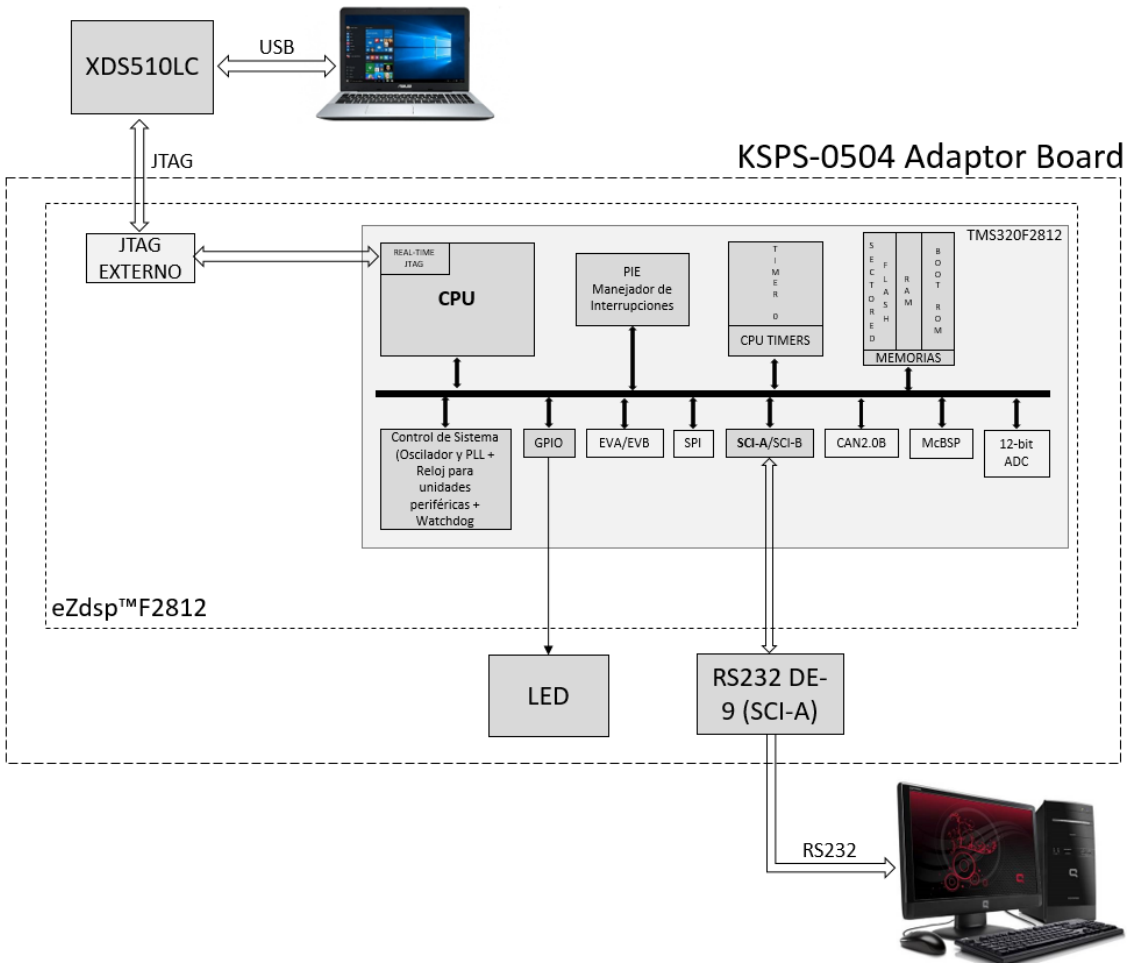


Figura 4.91 Diagrama de Bloques de Comunicación SCI entre DSP y PC usando Interrupciones

Se transmitirá la frase “F2812 esta comunicando \n\r” mediante la interfaz RS232 el cual se podrá visualizar en la ventana principal del programa HyperTerminal cada 2 segundos. El retardo de tiempo se generará por hardware usando los Timers internos del DSP. El LED conectado a GPIO B0 será el indicador de la transmisión de datos, es decir, cada vez que se esté transmitiendo caracteres del texto el LED estará prendido. Ver Figura 4.91

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Code Composer Studio*.

NOTA: Si al abrir el programa sale algún error de conexión con la PC, el motivo podría ser que no esté configurado el *Setup CCStudio V3.3* tal como se ve en la practica 1. Si el estudiante no ha realizado la *practica 1* se recomienda desarrollarla para evitar tener problemas de comunicación entre su PC y las tarjetas, para luego continuar con esta práctica.

3. Crear un proyecto nuevo (*Project -> New..)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica6
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** para crear el proyecto

4. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

// Prototipo de Funciones
```

```
void Gpio_select(void);
void InitSystem(void);
interrupt void cpu_timer0_isr(void);


void main(void)
{
	InitSystem();
	Gpio_select();

	InitPieCtrl();
	InitPieVectTable();

	EALLOW;
	PieVectTable.TINT0 = &cpu_timer0_isr;
	EDIS;

	InitCpuTimers();
	ConfigCpuTimer(&CpuTimer0, 150, 50000);

	PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
	IER = 1;
	EINT;
	ERTM;

	CpuTimer0Regs.TCR.bit.TSS = 0;

	while(1)
	{
		while(CpuTimer0.InterruptCount < 40);
		CpuTimer0.InterruptCount = 0;
	}
}

void Gpio_select(void)
{
	EALLOW;
	GpioMuxRegs.GPAMUX.all = 0x0;
	GpioMuxRegs.GPBMUX.all = 0x0;
	GpioMuxRegs.GPDMUX.all = 0x0;
	GpioMuxRegs.GPFMUX.all = 0x0;
	GpioMuxRegs.GPEMUX.all = 0x0;
	GpioMuxRegs.GPGMUX.all = 0x0;

	GpioMuxRegs.GPADIR.all = 0x0;
	GpioMuxRegs.GPBDIR.all = 0x00FF;
	GpioMuxRegs.GPDDIR.all = 0x0;
	GpioMuxRegs.GPEDIR.all = 0x0;
	GpioMuxRegs.GPFDIR.all = 0x0;
	GpioMuxRegs.GPGDIR.all = 0x0;

	GpioDataRegs.GPBDAT.all = 0x0000;
	EDIS;
```

```
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00E8;

        SysCtrlRegs.SCSR = 0;

        SysCtrlRegs.PLLCR.bit.DIV = 10;

        SysCtrlRegs.HISPCP.all = 0x1;
        SysCtrlRegs.LOSPCP.all = 0x2;


        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
}

interrupt void cpu_timer0_isr(void)
{
        CpuTimer0.InterruptCount++;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

5. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica6.c"***

6. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica6.c".***

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

7. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **DSP281x_GlobalVariableDefs.c**

8. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_Headers_nonBIOS.cmd**

9. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

10. De la dirección "*C:\tidcs\c28\dsp281x\v100\DSP281x_common\source"* agregar al proyecto los archivos:

- **DSP281x_PieCtrl.c**
- **DSP281x_PieVect.c**
- **DSP281x_DefaultIsr.c**
- **DSP281x_CpuTimers.c**
- **DSP281x_usDelay.asm**

11. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL "BUILD OPTIONS"

12. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options..*
- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include
  en el campo *Include Search Path*

13. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **400** en el campo *Stack Size.*
- Dar clic en el botón *OK* para terminar la configuración del *Build Options*

AGREGAR CÓDIGO DE INICIALIZACIÓN SCI

14. Habilitar el reloj de la unidad periférica SCI en la función “InitSystem()”. Como se observa en la Figura 4.92

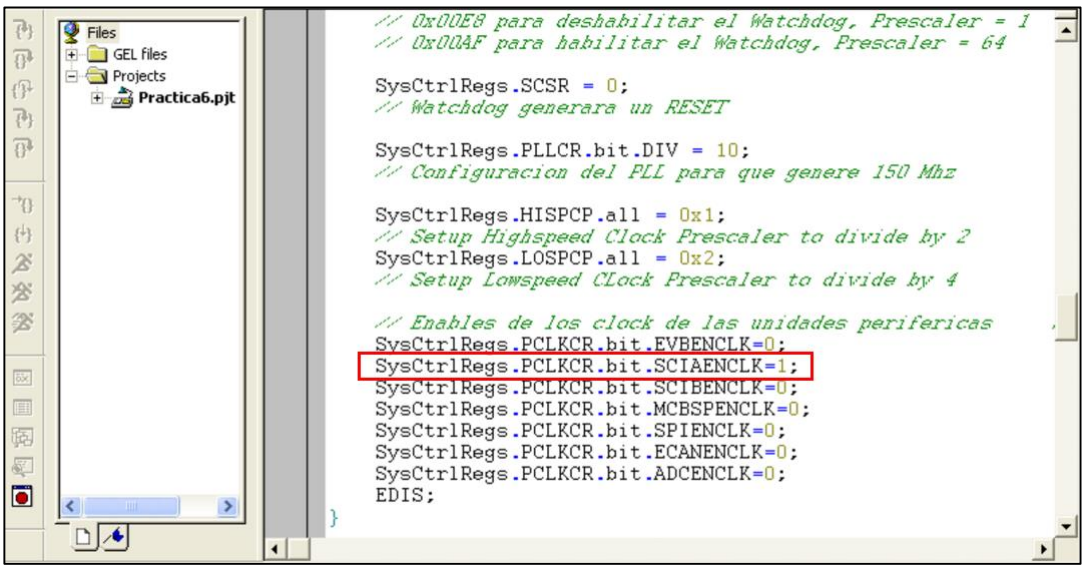


Figura 4.92 Habilitación de Reloj de unidad periférica de comunicación SCI

15. Configurar como función primaria los pines del puerto F en la función “Gpio_select()”, los cuales están asociados a la unidad periférica de comunicación SCI. Como se observa en la Figura 4.93

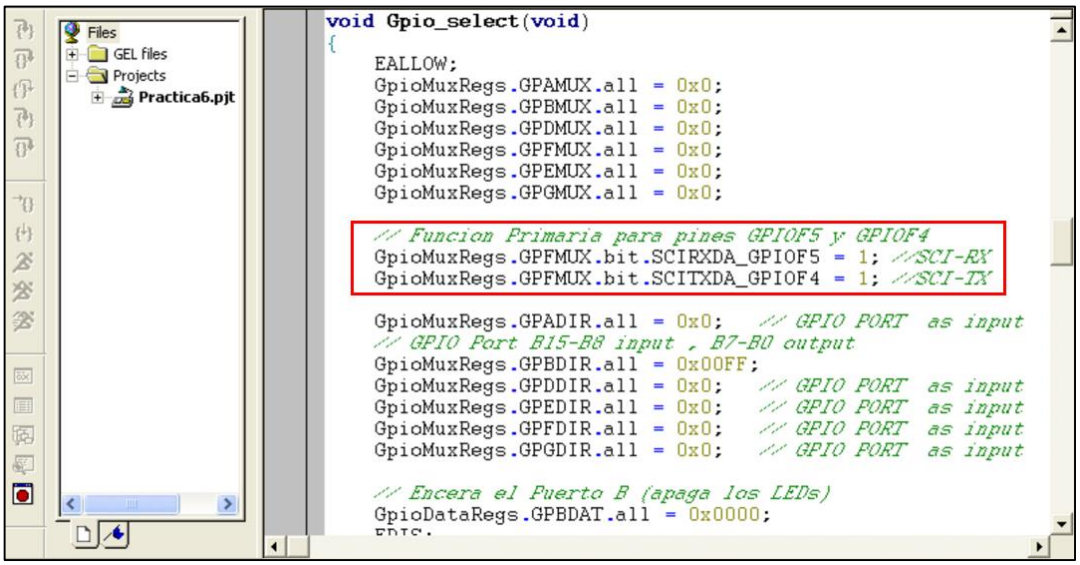


Figura 4.93 Configuración de función primaria en los pines GPIOF4 y GPIOF5

16. Al inicio del archivo de código fuente ***"Practica6"***, antes del "main" agregar el prototipo de función **"void SCI_Init(void)"**, como se observa en la Figura 4.94

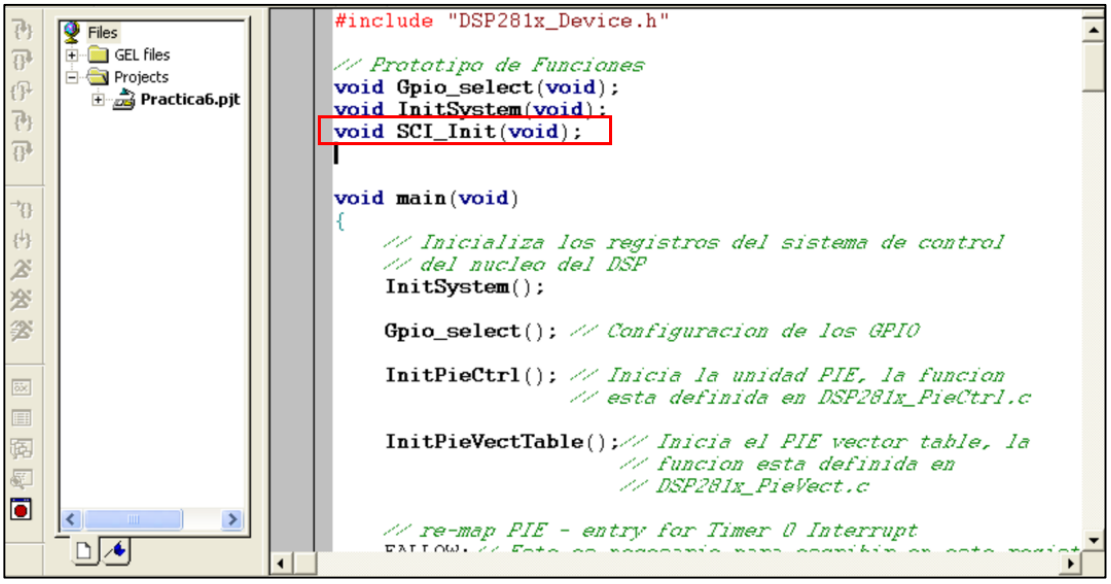


Figura 4.94 Prototipo de función "SCI_Init( )"

17. Agregar la función ***"SCI_Init( )"*** en el "main" antes de ingresar al lazo "while(1)". Como se observa en la Figura 4.95

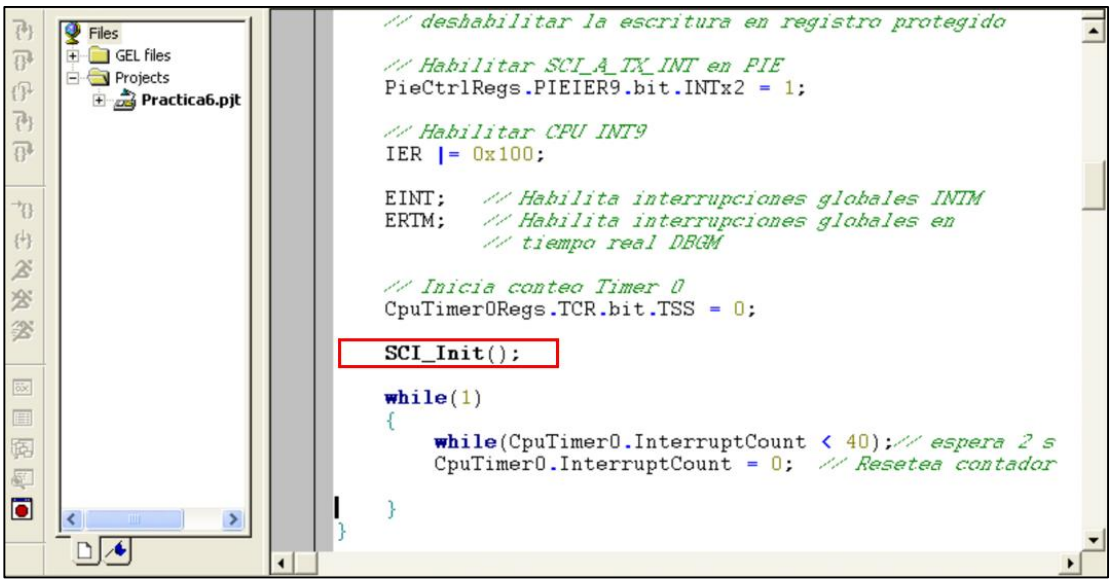


Figura 4.95 LLamado de función "SCI_Init( )" en el "main"

18. Al final del archivo código fuente ***"Practica6"***, agregar la definición de la función **"void SCI_Init(void)"** (Ver Figura 4.96) en base a la siguiente configuración:

Para el registro SCICCR:

- Configurar un bit de STOP
- Deshabilitar paridad y loop back
- Configurar 8 bits de datos por carácter
- Seleccionar modo Idle-line

Para el registro SCICTL1:

- Resetear SCI
- Habilitar transmisión y recepción
- Deshabilitar modo SLEEP, TXWAKE e interrupciones por errores de recepción.

Para el registro SCITL2:

- Habilitar la interrupción de transmisión SCI

Para el registro SCIHBAUD / SCILBAUD:

- Configurar la velocidad a 9600 baud rate, sabiendo que la frecuencia de LSPCLK es de 37.5 MHz.

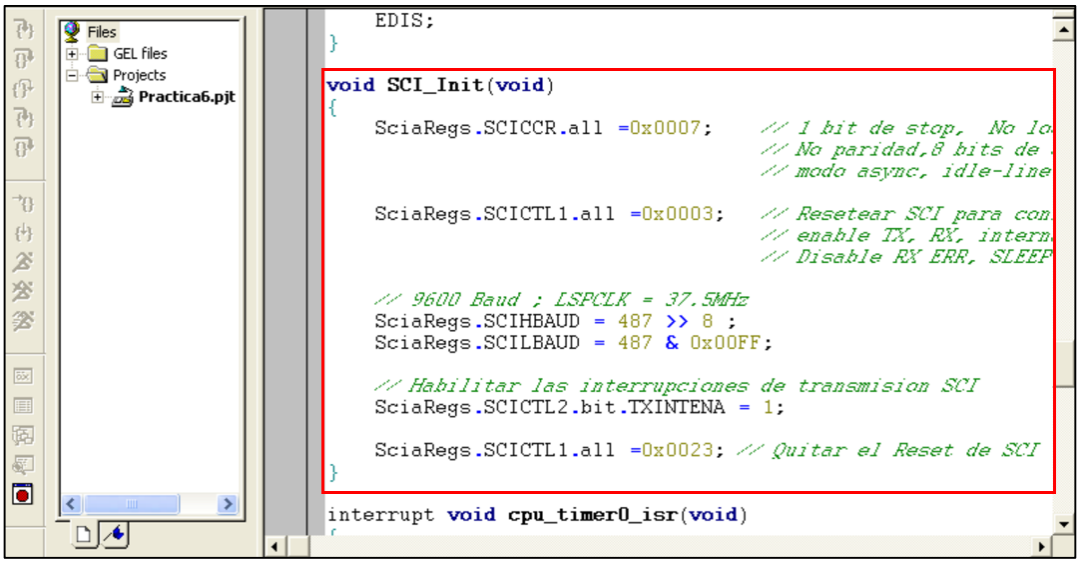


Figura 4.96 Definición de función "SCI_Init( )"

19. Una vez habilitada la interrupción SCI-TX, definir el servicio de interrupción “SCI_TX_isr( )” (Ver Figura 4.97). Al inicio del programa se debe declarar esta función y al final del código se la define en base a lo siguiente:

- Se debe cargar el siguiente carácter del string en SCITXBUF siempre que no hayamos terminado de recorrer y transmitir el string en su totalidad
- Llamar a la función de reconocimiento para resetear el registro PIEACK

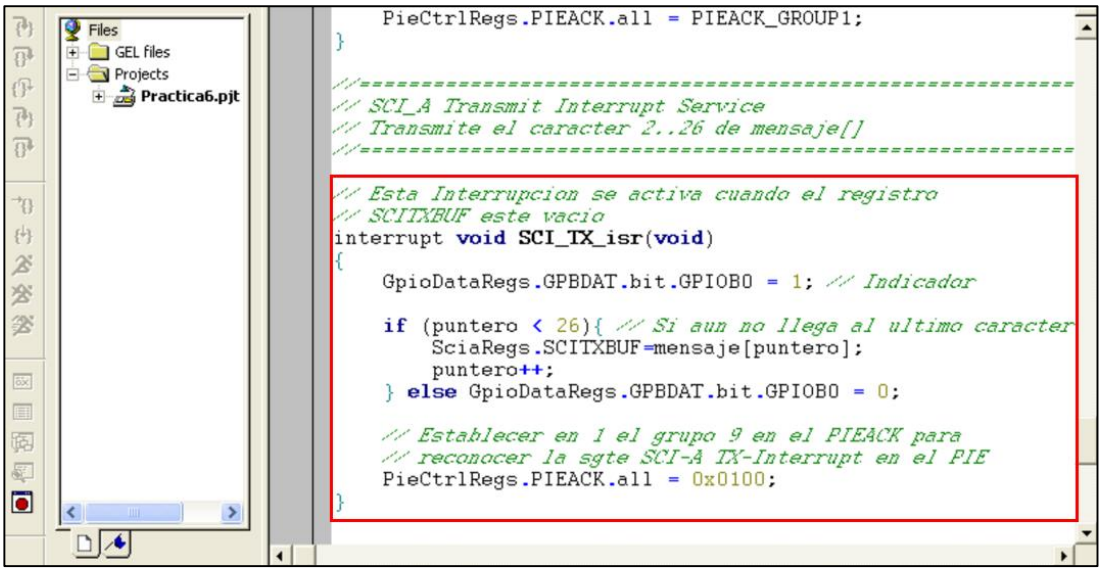


Figura 4.97 Definición de rutina de servicio de interrupción para transmisión SCI

20. Habilitar la interrupción de transmisión SCI en el registro de control PIE y escribir la dirección de la rutina del servicio de interrupción en la tabla de vectores PIE. Antes de la línea del habilitador global de interrupciones “EINT” como se aprecia en la Figura 4.98

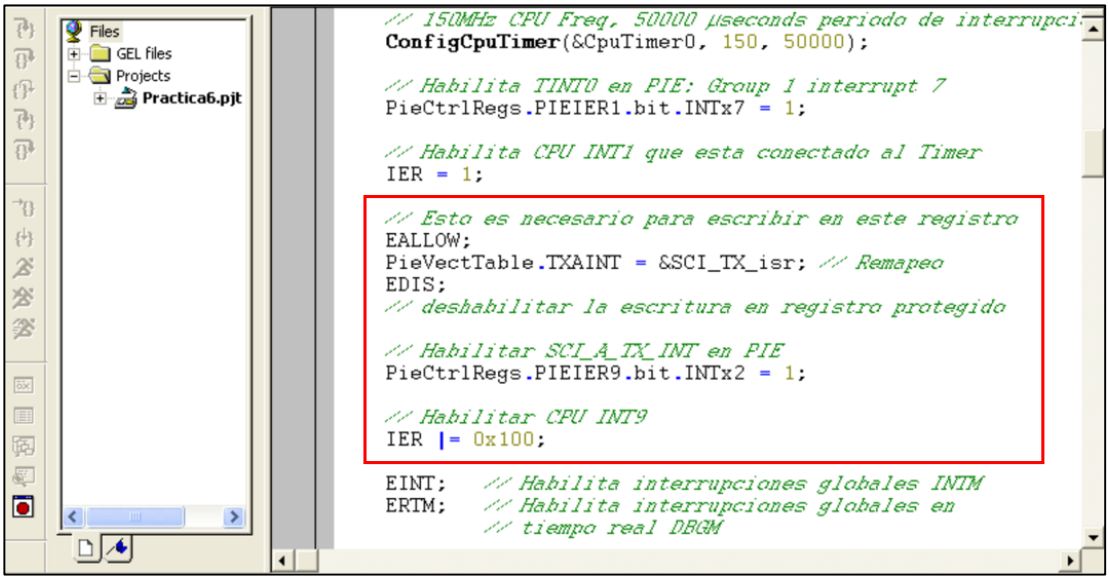


Figura 4.98 Configuración de interrupción por transmisión SCI

21. Declarar las variables “index” y “messagge” como variables globales debido a que no forman parte del “main” sino que son parte de “SCI_TX_isr()”. Esta declaración se la debe especificar en el inicio del código como se aprecia en la Figura 4.99

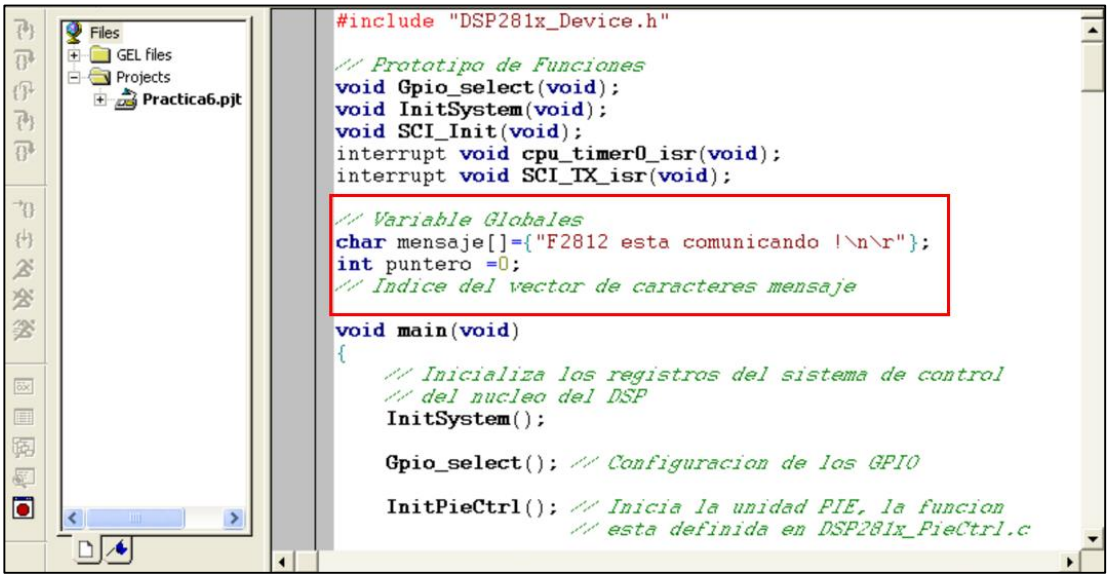


Figura 4.99 Declaración de Variables globales a usar en la práctica

PROGRAMACIÓN DEL “MAIN LOOP”

22. En el interior del “while(1)”:

- Debido a que se desea transmitir la frase cada dos segundos y sabiendo que “CPUTimer0.InterrupCount” se incrementa en 1 cada 50 ms, se codifica una espera en la que “CPUTimer0.InterrupCount” incrementa hasta 40 (2000ms)
- Luego del transcurso de los dos segundos, se le asigna el valor 0 a “CPUTimer0.InterrupCount” para resetearlo
- Resetear la variable “index” asignándole el valor 0
- Cargar el primer caracter del mensaje en SCITXBUF
- Incrementar la variable “index”

En la Figura 4.100 se muestra la programación que se debe hacer dentro del lazo “while(1)” para hacer el retardo por hardware de dos segundos y transmitir el primer carácter del mensaje.

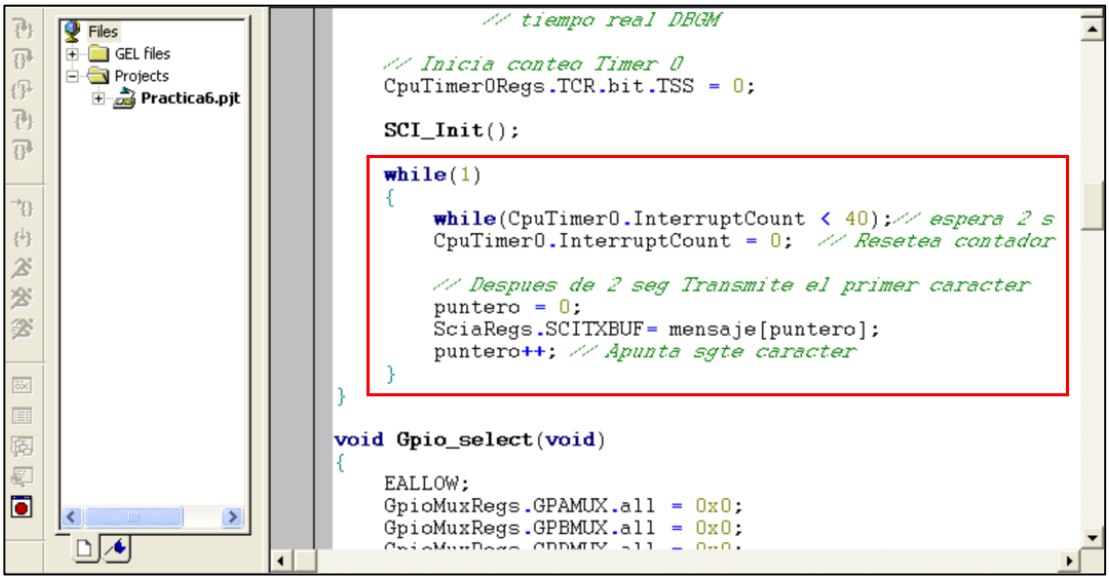


Figura 4.100 Programación dentro del lazo "while(1)"

COMPILAR, CARGAR Y EJECUTAR EL PROGRAMA

23. Compilar el proyecto seleccionando*: Project -> Rebuild All*. Solucionar los errores en caso de haberlos.

24. Alimentar la tarjeta *eZdsp™F2812*.

25. Conectarse con el procesador: *Debug -> Connect*

26. Cargar el programa (*File -> Load Program…)* **Practica6.out** que se genera dentro de la carpeta Debug al compilar el proyecto

27. Ejecutar el programa cargado: *Debug -> Run*

En la Figura 4.101 se observa la ejecución de la práctica, donde el led D1 se encenderá cada vez que se esté transmitiendo datos al PC.

Figura 4.101 Foto del Kit mientras se ejecuta la práctica

28. Configurar las propiedades del puerto COM de la PC de acuerdo al formato de la trama SCI.

29. Abrir programa Hyper Terminal para colocar nombre del proyecto y luego dar click en **OK**:

30. Al abrirse la ventana “Connect To”, se debe seleccionar el puerto por el cual se realizará la comunicación para luego dar click en **Configure…**

31. Configurar las propiedades del puerto COM del programa HyperTerminal de acuerdo al formato de la trama SCI.

32. Dar click en **Aceptar** y luego en **OK** para poder visualizar el mensaje recibido que ha sido transmitido por la DSP

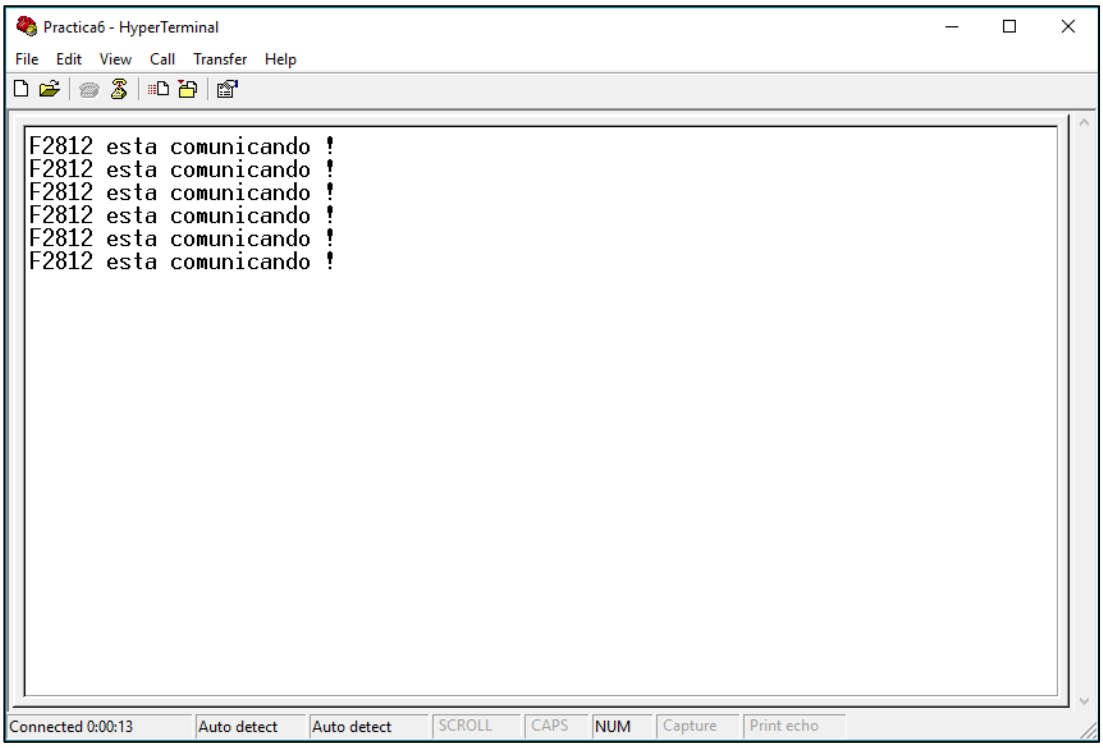


Figura 4.102 Visualización de los caracteres transmitidos

En la Figura 4.102 se observa los caracteres del mensaje transmitido desde el DSP *TMS320F2812* cada 2 segundos hacia el Computador mediante la unidad periférica de comunicación SCI del procesador usando un retardo por hardware.

33. Al finalizar la práctica detener la ejecución del programa: *(Debug -> Halt)*

34. Resetear el CPU *(Debug -> Reset CPU)*, luego desconectarse de la tarjeta *(Debug -> Disconnect).* Y quitar la alimentación de la tarjeta *eZdsp™F2812*.

PRÁCTICA # 7

TEMA:

"Transmisión de datos a una PC mediante la unidad FIFO de la interfaz de comunicación SCI y el sistema de interrupciones del procesador TMS320F2812"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Mejorar la Práctica 5 mediante la implementación del método FIFO de la comunicación SCI
- Configurar el formato de la trama, la velocidad, y habilitar el sistema de interrupciones para una comunicación SCI.
- Establecer comunicación SCI entre el procesador *TMS320F2812* y la PC.
- Transmitir un texto desde el procesador *F2812* hacia el puerto serial de la computadora cada 2 segundos.

REQUISITOS MÍNIMOS:

- Haber Instalado el software Code Composer Studio (versión 3.3) con su respectivo drive para el manejo del convertidor JTAG a USB.
- Haber instalado el programa *HyperTerminal*
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos sobre el manejo de interrupciones
- Conocimientos de programación en Lenguaje C
- Haber culminado satisfactoriamente la Práctica # 6.
- Lectura y comprensión del archivo de fundamentación teórica: "SERIAL COMMUNICATION INTERFACE (SCI)"

BREVE EXPLICACIÓN DE LA PRÁCTICA

En lugar de generar una gran cantidad de interrupciones SCI como en la práctica anterior, se va a utilizar la técnica de transmisión de ráfaga para llenar los 16 caracteres en el interior de la FIFO de transmisión SCI, con esto se consigue reducir la cantidad de servicios de interrupciones a una sola interrupción por frase.

Utilizando la unidad periférica SCI, y el manejo de interrupciones se establecerá la comunicación entre el procesador *TMS320F2812* y el puerto serial de la PC. La configuración para el uso de la unidad periférica SCI se la realiza por programa mediante el software Code Composer Studio. Para poder establecer la comunicación SCI se debe configurar también la trama y velocidad en el puerto COM de la PC y en el programa HyperTerminal.

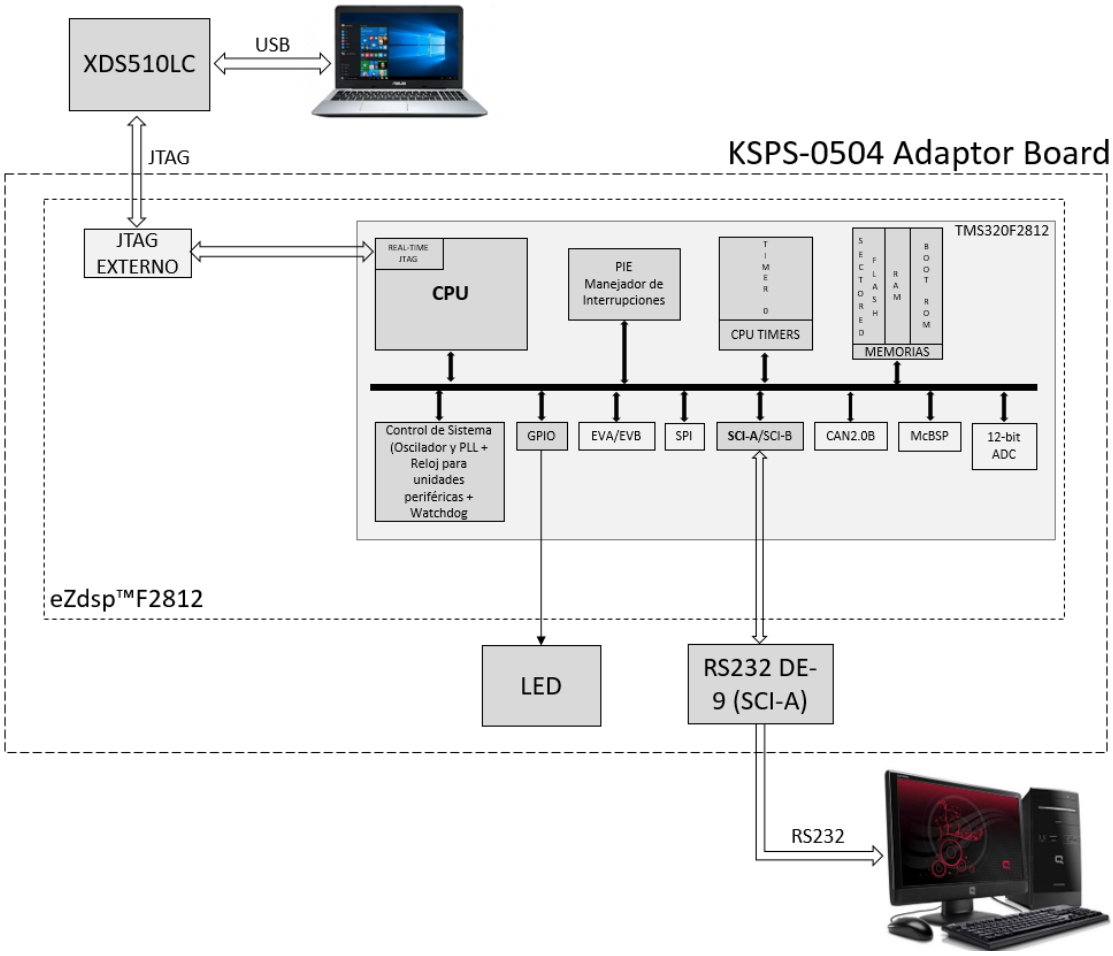


Figura 4.103 Diagrama de Bloques de Comunicación SCI entre DSP y PC usando Interrupciones y el método FIFO

Se cargará la FIFO de transmisión SCI con los caracteres de la frase “COMUNICANDO ! \n\r” para luego ser transmitidos mediante la interfaz RS232 y visualizados en la ventana principal del programa HyperTerminal cada 2 segundos. El retardo de tiempo se generará por hardware usando los Timers internos del DSP. El LED conectado a GPIO B0 será el indicador de la transmisión de datos, es decir, cada vez que se esté transmitiendo caracteres del texto el LED estará prendido. Ver Figura 4.103

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Code Composer Studio*.

NOTA: Si al abrir el programa sale algún error de conexión con la PC, el motivo podría ser que no esté configurado el *Setup CCStudio V3.3* tal como se ve en la practica 1. Si el estudiante no ha realizado la *practica 1* se recomienda desarrollarla para evitar tener problemas de comunicación entre su PC y las tarjetas, para luego continuar con esta práctica.

3. Crear un proyecto nuevo (*Project -> New..)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica7
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** para crear el proyecto

4. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

// Prototipo de funciones
void Gpio_select(void);
void InitSystem(void);
interrupt void cpu_timer0_isr(void);

void main(void)
{
        InitSystem();
        Gpio_select();

        InitPieCtrl();
        InitPieVectTable();

        EALLOW;
        PieVectTable.TINT0 = &cpu_timer0_isr;
        EDIS;

        InitCpuTimers();
```

```
        ConfigCpuTimer(&CpuTimer0, 150, 50000);

        PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
        IER = 1;
        EINT;
        ERTM;
        CpuTimer0Regs.TCR.bit.TSS = 0;

        while(1)
        {
                while(CpuTimer0.InterruptCount < 40){

                }
                CpuTimer0.InterruptCount = 0;
        }
}

void Gpio_select(void)
{
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;
        GpioMuxRegs.GPBMUX.all = 0x0;
        GpioMuxRegs.GPDMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;

        GpioMuxRegs.GPADIR.all = 0x0;
        GpioMuxRegs.GPBDIR.all = 0x00FF;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
        GpioMuxRegs.GPFDIR.all = 0x0;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioDataRegs.GPBDAT.all = 0x0000;
        EDIS;
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00E8;

        SysCtrlRegs.SCSR = 0;
        SysCtrlRegs.PLLCR.bit.DIV = 10;

        SysCtrlRegs.HISPCP.all = 0x1;
        SysCtrlRegs.LOSPCP.all = 0x2;


        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
```

```
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
}

interrupt void cpu_timer0_isr(void)
{
        CpuTimer0.InterruptCount++;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

5. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica7.c"***

6. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica7.c".***

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

7. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **DSP281x_GlobalVariableDefs.c**

8. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_Headers_nonBIOS.cmd**

9. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

10. De la dirección "*C:\tidcs\c28\dsp281x\v100\DSP281x_common\source"* agregar al proyecto los archivos:

- **DSP281x_PieCtrl.c**
- **DSP281x_PieVect.c**
- **DSP281x_DefaultIsr.c**
- **DSP281x_CpuTimers.c**

- **DSP281x_usDelay.asm**

11. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL "BUILD OPTIONS"

12. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options..*
- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include
  en el campo *Include Search Path*

13. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **400** en el campo *Stack Size.*
- Dar clic en el botón *OK* para terminar la configuración del *Build Options*

AGREGAR CÓDIGO DE INICIALIZACIÓN SCI

14. Habilitar el reloj de la unidad periférica SCI en la función "InitSystem()". Tal como se observa en la Figura 4.104

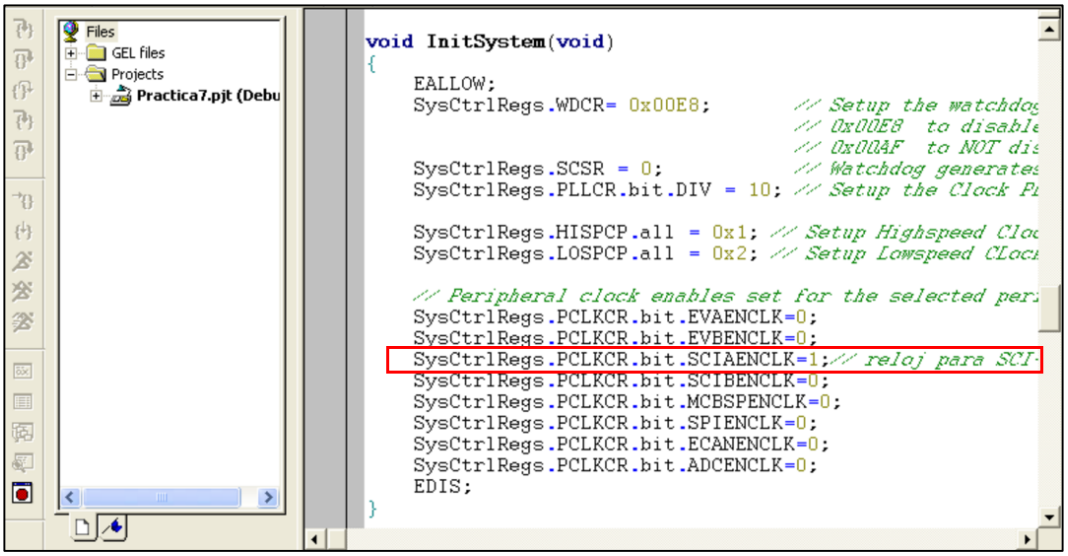


Figura 4.104 Habilitación de Reloj de unidad periférica de comunicación SCI

15. Configurar como función primaria los pines del puerto F en la función "Gpio_select()", los cuales están asociados a la unidad periférica de comunicación SCI. Ver Figura 4.105

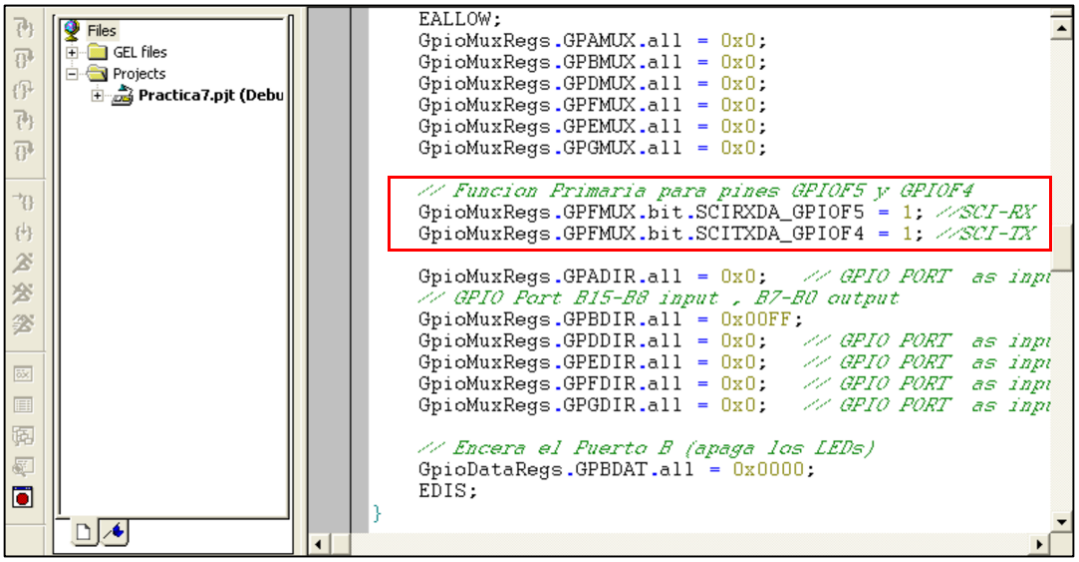


Figura 4.105 Configuración de función primaria en los pines GPIOF4 y GPIOF5

16. Al inicio del archivo de código fuente ***"Practica7"***, antes del "main" agregar el prototipo de función **"void SCI_Init(void)"**, como se observa en la Figura 4.106

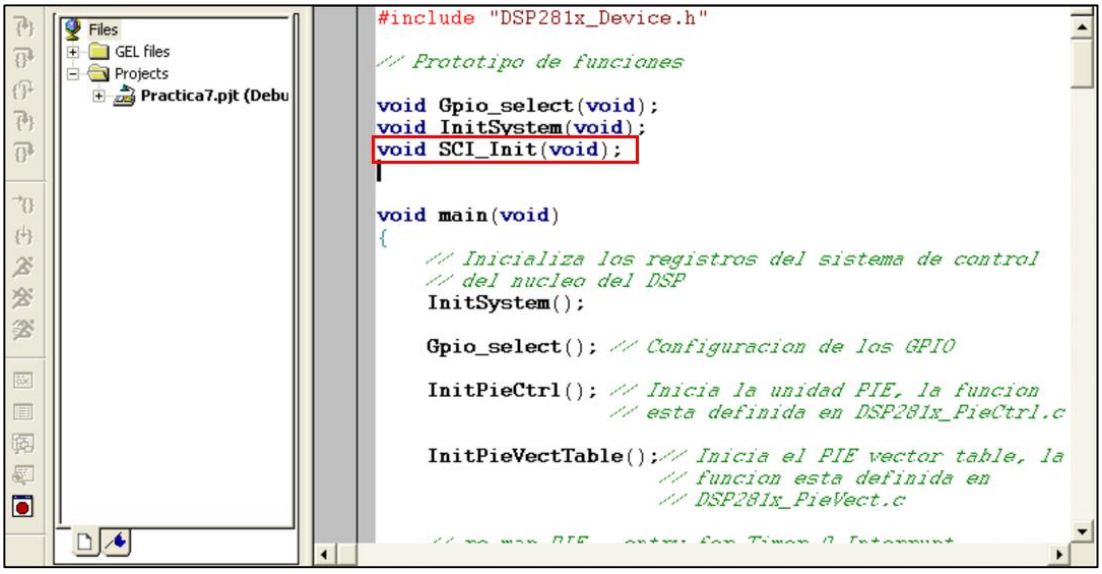


Figura 4.106 Declaración del prototipo de función "SCI_Init( )"

17. Agregar la función ***"SCI_Init( )"*** en el "main" antes de ingresar al lazo "while(1)", como se observa en la Figura 4.107

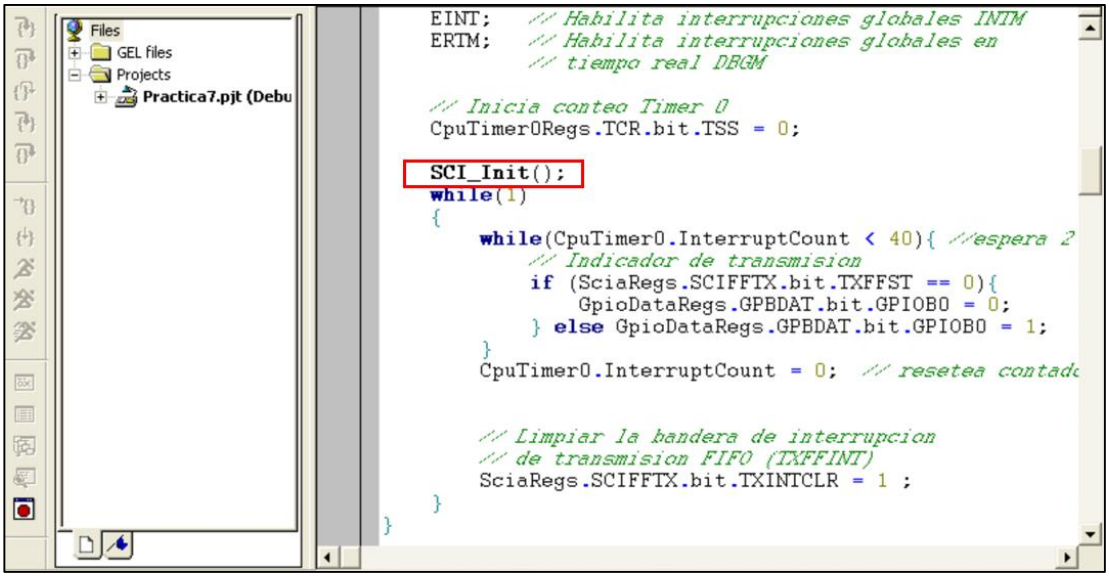


Figura 4.107 Llamado a la función "SCI_Init( )" en el "main"

18. Al final del archivo código fuente, agregar la definición de la función "SCI_init( )" (Ver Figura 4.108) en base a la siguiente configuración:

Para el registro SCICCR:

- Configurar un bit de STOP
- Deshabilitar paridad y loop back
- Configurar 8 bits de datos por carácter
- Seleccionar modo Idle-line

Para el registro SCICTL1:

- Resetear SCI
- Habilitar transmisión y recepción
- Deshabilitar modo SLEEP, TXWAKE e interrupciones por errores de recepción.

Para el registro SCITL2:

- Habilitar la interrupción de transmisión SCI

Para el registro SCIHBAUD / SCILBAUD:

- Configurar la velocidad a 9600 baud rate, sabiendo que la frecuencia de LSPCLK es de 37.5 MHz.

Para el registro SCIFFTX:

- Reanudar la unidad FIFO de transmisión y recepción
- Habilitar las mejoras de la unidad FIFO
- Habilitar la operación de transmisión FIFO
- Limpiar el indicador de interrupciones de transmisión FIFO
- Habilitar la condición TXFFIVL para que ocurra una interrupción de transmisión FIFO
- Establecer el nivel de interrupción FIFO para interrumpir cuando FIFO esté vacío

NOTA: En caso de querer colocar un valor hexadecimal a to registro, a los bits reservados de los registros se le debe colocar el valor de “0”

Figura 4.108 Definición de la funcion "SCI_Init( )"

19. Declarar como variable global “mensaje” debido a que no será utilizada en el “main” principal sino en otra función. Esta declaración se la debe especificar en el inicio del código como se aprecia en la Figura 4.109

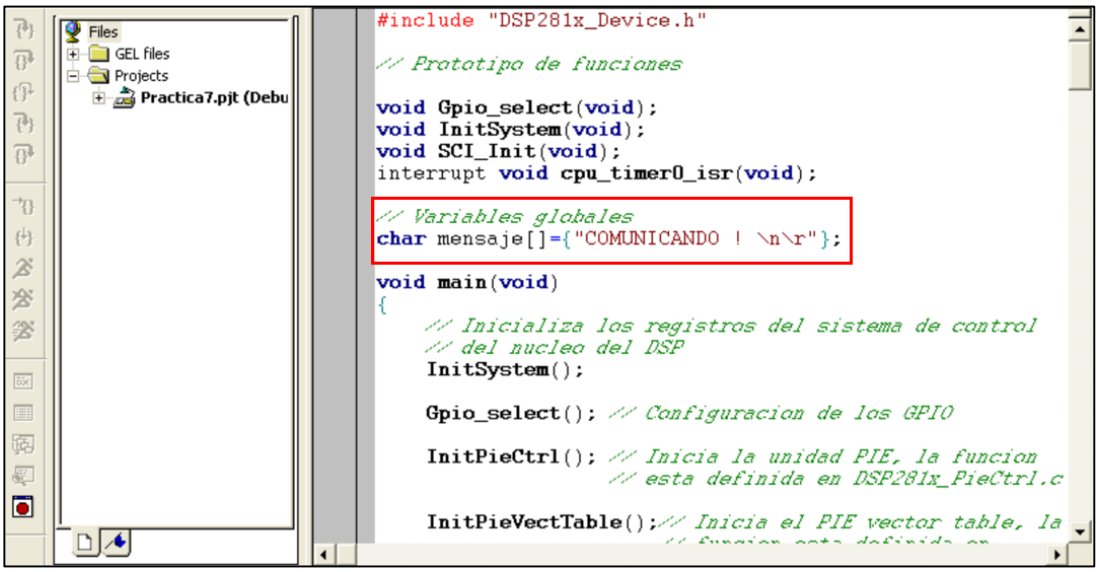


Figura 4.109 Declaración de las variables globales a usar en la práctica

Como se puede apreciar, la frase para esta práctica contiene 16 caracteres a diferencia de las anteriores en que tenía 26, esto se debe a que la unidad FIFO de transmisión tiene 16 niveles en la que puede colocar un caracter por nivel.

20. Definir el servicio de interrupción “SCI_TX_isr( )”. Al inicio del programa se debe declarar esta función y al final del código se la define en base a lo siguiente:

- Mediante un lazo, cargar los 16 caracteres dentro la unidad de transmisión FIFO mediante el registro SCITXBUF que únicamente sirve ahora de intermediario. Ver Figura 4.110

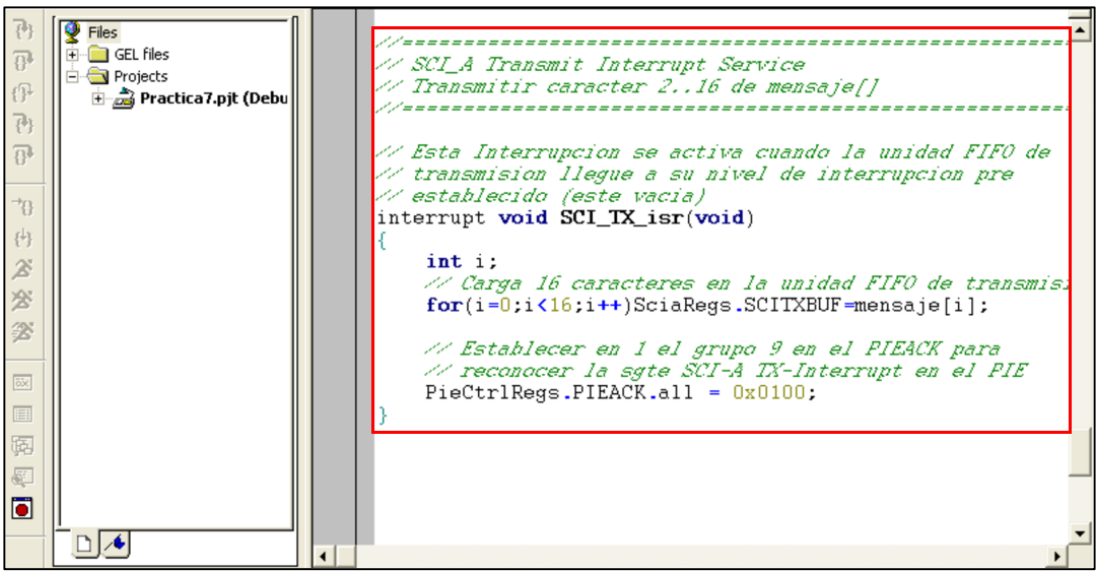


Figura 4.110 Definición de rutina de servicio de interrupción para transmisión SCI con método FIFO

La rutina de servicio será llamada cuando el nivel de interrupción FIFO sea alcanzado. Debido a que la configuración del nivel de interrupción FIFO fue establecido en cero, se podrán cargar los 16 caracteres en de la unidad de transmisión FIFO.

21. Habilitar la interrupción de transmisión SCI en el registro de control PIE y escribir la dirección de la rutina del servicio de interrupción en la tabla de vectores PIE. Antes de la línea del habilitador global de interrupciones “EINT” como se aprecia en la Figura 4.111

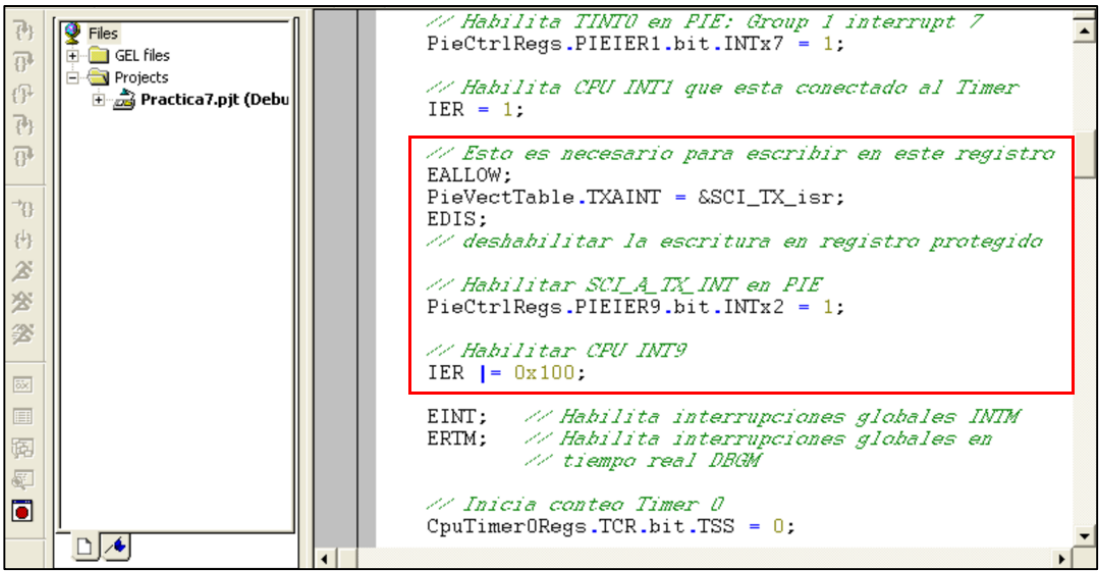


Figura 4.111 Configuración de interrupción por transmisión SCI usando método FIFO

PROGRAMACIÓN DEL “MAIN LOOP”

22. En el interior del “while(1)”:

- Debido a que se desea transmitir la frase cada dos segundos y sabiendo que “CPUTimer0.InterrupCount” se incrementa en 1 cada 50 ms, se codifica una espera en la que “CPUTimer0.InterrupCount” incrementa hasta 40 (2000ms)
- Luego del transcurso de los dos segundos, se le asigna el valor 0 a “CPUTimer0.InterrupCount” para resetearlo
- Limpiar la bandera de interrupción FIFO

En la Figura 4.112 se muestra la programación que se debe hacer dentro del lazo “while(1)” para limpiar la bandera de interrupciones por transmisión SCI usando método FIFO.

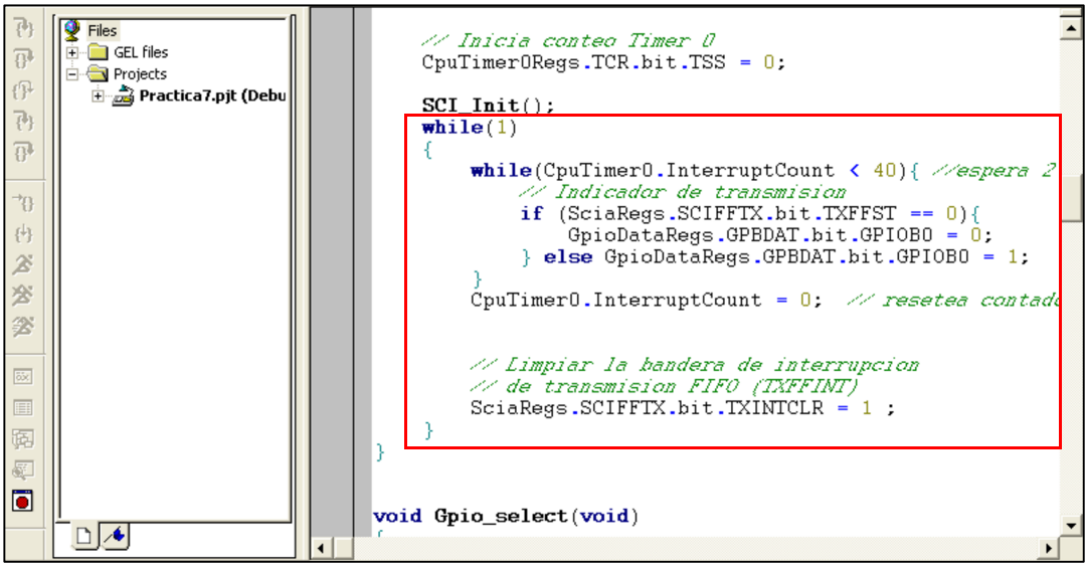


Figura 4.112 Programación dentro del lazo "while( )"

En esta práctica se habilitará la interrupción de transmisión FIFO para solicitar servicio de rutina cuando el nivel FIFO sea cero. Inmediatamente después de la inicialización del SCI se llevará a cabo la primera interrupción de transmisión FIFO. La siguiente interrupción será llamada solamente cuando se establezca en 1 el bit TXINTCLR del registro SCIFFTX, esta configuración limpia la bandera de interrupción de transmisión FIFO. Limpiando dicha bandera pasados dos segundos, la frase se transmitirá cada dos segundos (una frase por interrupción).

COMPILAR, CARGAR Y EJECUTAR EL PROGRAMA

23. Compilar el proyecto seleccionando*: Project -> Rebuild All*. Solucionar los errores en caso de haberlos.

24. Alimentar la tarjeta *eZdsp™F2812*.

25. Conectarse con el procesador: *Debug -> Connect*

26. Cargar el programa (*File -> Load Program…)* **Practica7.out** que se genera dentro de la carpeta Debug al compilar el proyecto

27. Ejecutar el programa cargado: *Debug -> Run*

En la Figura 4.113 se observa la ejecución de la práctica, donde el led D1 se encenderá cada vez que se esté transmitiendo datos al PC.

Figura 4.113 Foto del Kit mientras se ejecuta la práctica

28. Configurar las propiedades del puerto COM de la PC de acuerdo al formato de la trama SCI.

29. Abrir programa Hyper Terminal para colocar nombre del proyecto y luego dar click en **OK**:

30. Al abrirse la ventana “Connect To”, se debe seleccionar el puerto por el cual se realizará la comunicación para luego dar click en **Configure…**

31. Configurar las propiedades del puerto COM del programa HyperTerminal de acuerdo al formato de la trama SCI.

32. Dar click en **Aceptar** y luego en **OK** para poder visualizar el mensaje recibido que ha sido transmitido por la DSP

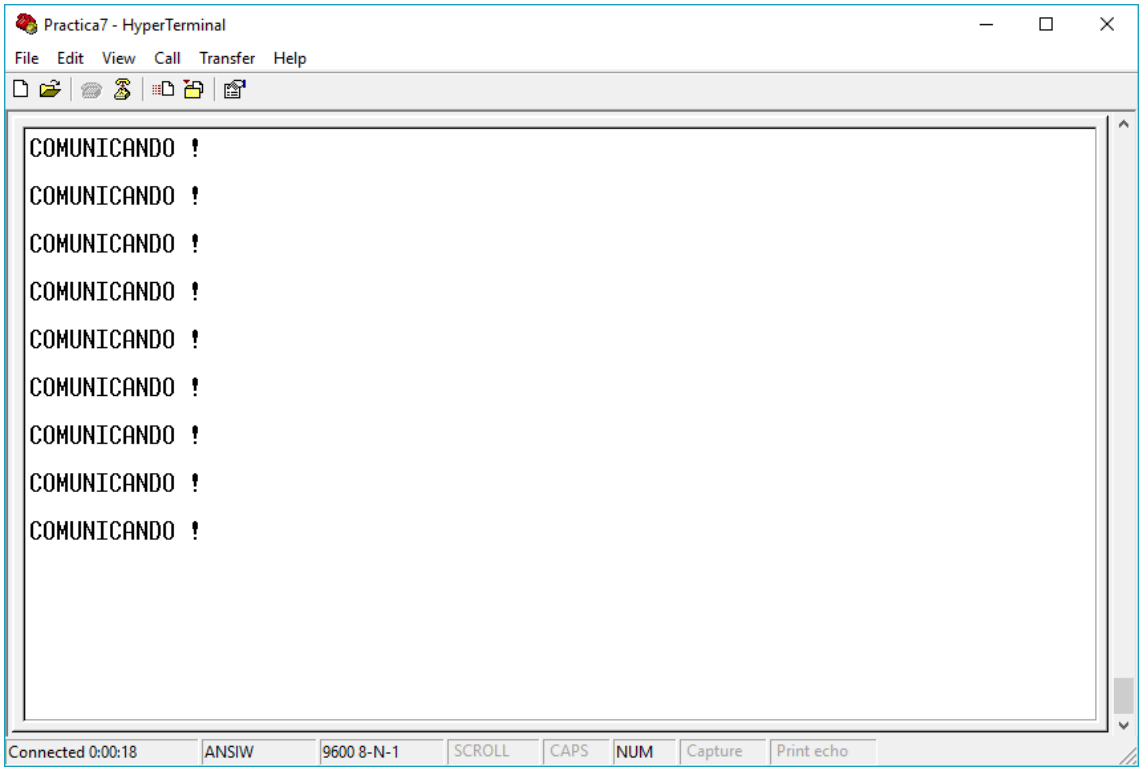


Figura 4.114 Visualización de los caracteres transmitidos

En la Figura 4.114 se observa los caracteres del mensaje transmitido desde el DSP *TMS320F2812* hacia el Computador mediante la unidad periférica de comunicación SCI del procesador.

33. Al finalizar la práctica detener la ejecución del programa: *(Debug -> Halt)*

34. Resetear el CPU *(Debug -> Reset CPU)*, luego desconectarse de la tarjeta *(Debug -> Disconnect).* Y quitar la alimentación de la tarjeta *eZdsp™F2812*.

PRÁCTICA # 8

TEMA:

"Transmisión y Recepción de datos mediante interfaz SCI entre el procesador TMS320F2812 y una PC"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Configurar el formato de la trama, la velocidad, y habilitar el sistema de interrupciones para una comunicación SCI.
- Establecer comunicación SCI entre el procesador *TMS320F2812* y la PC.
- Establecer la velocidad del "Knight-Rider" (secuencia de encendido de leds) mediante la recepción de datos desde la PC.

REQUISITOS MÍNIMOS:

- Haber Instalado el software Code Composer Studio (versión 3.3) con su respectivo drive para el manejo del convertidor JTAG a USB.
- Haber instalado el programa *HyperTerminal*
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos de programación en Lenguaje C
- Haber culminado satisfactoriamente la Práctica # 7.
- Lectura y comprensión del archivo de fundamentación teórica: "SERIAL COMMUNICATION INTERFACE (SCI)"

BREVE EXPLICACIÓN DE LA PRÁCTICA

Utilizando la unidad periférica SCI, y el manejo de interrupciones se establecerá la comunicación entre el procesador *F2812* y el puerto serial de la PC. La configuración para el uso de la unidad periférica SCI se la realiza por programa mediante el software Code Composer Studio. Para poder establecer la comunicación SCI se debe

configurar también la trama y velocidad en el puerto COM de la PC y en el programa HyperTerminal.

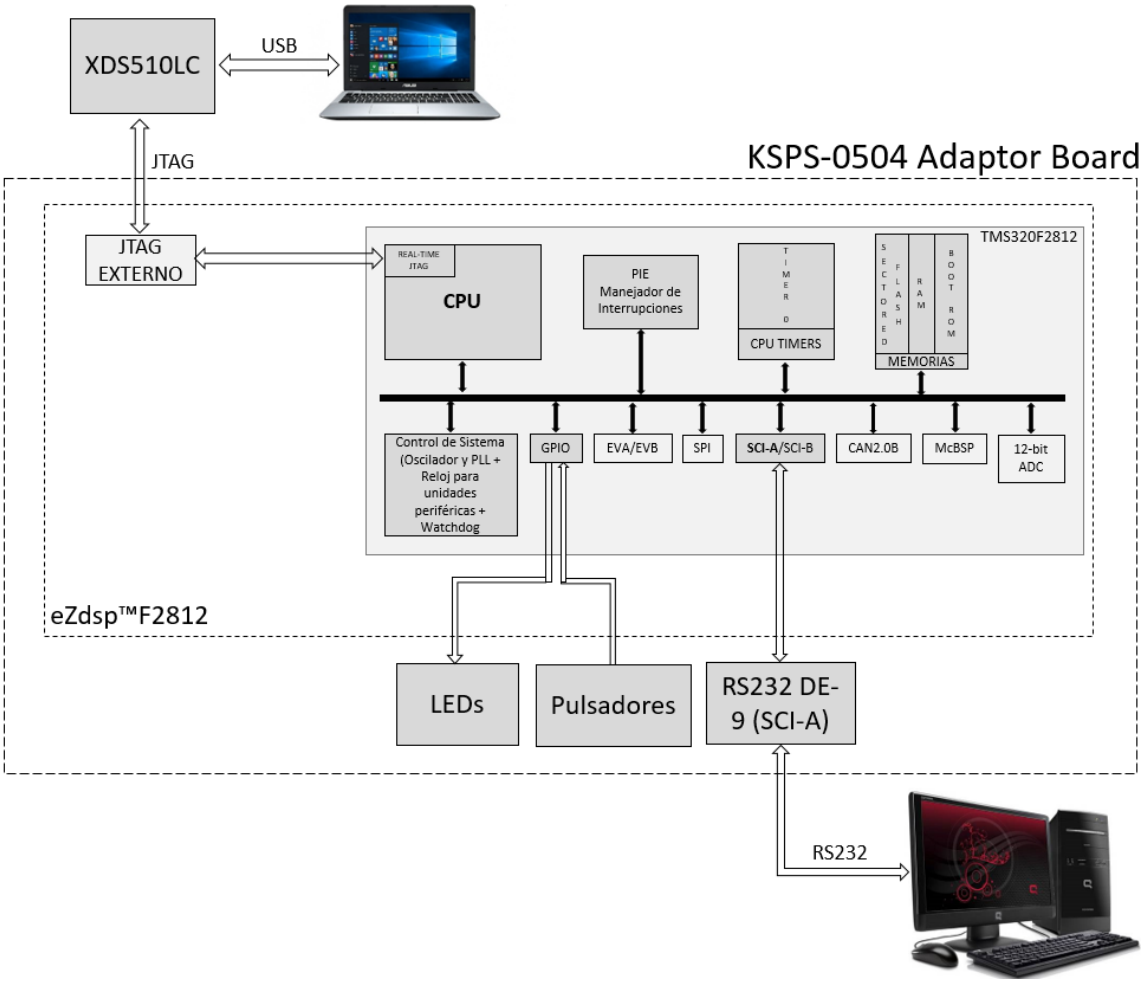


Figura 4.115 Diagrama de Bloques de transmisión y recepción de datos mediante SCI entre DSP y PC

Cuando el botón S1 (Conectado a GPIO D1) es pulsado se cargará la FIFO de transmisión SCI con los caracteres de la frase “Ingrese tiempo: ”, para luego ser transmitidos mediante la interfaz RS232 y visualizados en la ventana principal del programa HyperTerminal. Luego se transmitirá desde la PC al procesador *TMS320F2812* el tiempo el cual será almacenado en la unidad FIFO de recepción SCI. Cabe recalcar que el valor ingresado debe tener 5 caracteres en el orden de los milisegundos, en un rango de 0 a 9999 ms y que el ultimo carácter debe ser un

espacio para así poder validad el tiempo y modificar la velocidad de la secuencia de encendido de los leds (Conectados desde GPIO B0 a B8). Ver Figura 4.115

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Code Composer Studio*.

NOTA: Si al abrir el programa sale algún error de conexión con la PC, el motivo podría ser que no esté configurado el *Setup CCStudio V3.3* tal como se ve en la practica 1. Si el estudiante no ha realizado la *practica 1* se recomienda desarrollarla para evitar tener problemas de comunicación entre su PC y las tarjetas, para luego continuar con esta práctica.

3. Crear un proyecto nuevo (*Project -> New..)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica8
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** para crear el proyecto

4. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

// Prototipo de funciones
void Gpio_select(void);
void InitSystem(void);
interrupt void cpu_timer0_isr(void);

void main(void)
{
        InitSystem();
```

```
        Gpio_select();

        InitPieCtrl();
        InitPieVectTable();


        EALLOW;
        PieVectTable.TINT0 = &cpu_timer0_isr;
        EDIS;

        InitCpuTimers();
        ConfigCpuTimer(&CpuTimer0, 150, 50000);

        PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
        IER = 1;
        EINT;
        ERTM;

        CpuTimer0Regs.TCR.bit.TSS = 0;

        while(1){

        }
}

void Gpio_select(void)
{
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;
        GpioMuxRegs.GPBMUX.all = 0x0;
        GpioMuxRegs.GPDMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;

        GpioMuxRegs.GPADIR.all = 0x0;
        GpioMuxRegs.GPBDIR.all = 0x00FF;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
        GpioMuxRegs.GPFDIR.all = 0x0;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioDataRegs.GPBDAT.all = 0x0000;
        EDIS;
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00E8;

        SysCtrlRegs.SCSR = 0;
```

```
        SysCtrlRegs.PLLCR.bit.DIV = 10;

        SysCtrlRegs.HISPCP.all = 0x1;
        SysCtrlRegs.LOSPCP.all = 0x2;

        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
}

interrupt void cpu_timer0_isr(void)
{
        CpuTimer0.InterruptCount++;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

5. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica8.c"***

6. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica8.c".***

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

7. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **DSP281x_GlobalVariableDefs.c**

8. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_Headers_nonBIOS.cmd**

9. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

10. De la dirección "*C:\tidcs\c28\dsp281x\v100\DSP281x_common\source"* agregar al proyecto los archivos:

- **DSP281x_PieCtrl.c**
- **DSP281x_PieVect.c**
- **DSP281x_DefaultIsr.c**
- **DSP281x_CpuTimers.c**
- **DSP281x_usDelay.asm**

11. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL "BUILD OPTIONS"

12. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options..*
- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include
  en el campo *Include Search Path*

13. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **400** en el campo *Stack Size.*
- Dar clic en el botón *OK* para terminar la configuración del *Build Options*

AGREGAR CÓDIGO DE INICIALIZACIÓN SCI

14. Habilitar el reloj de la unidad periférica SCI en la función "InitSystem()", como se observa en la Figura 4.116

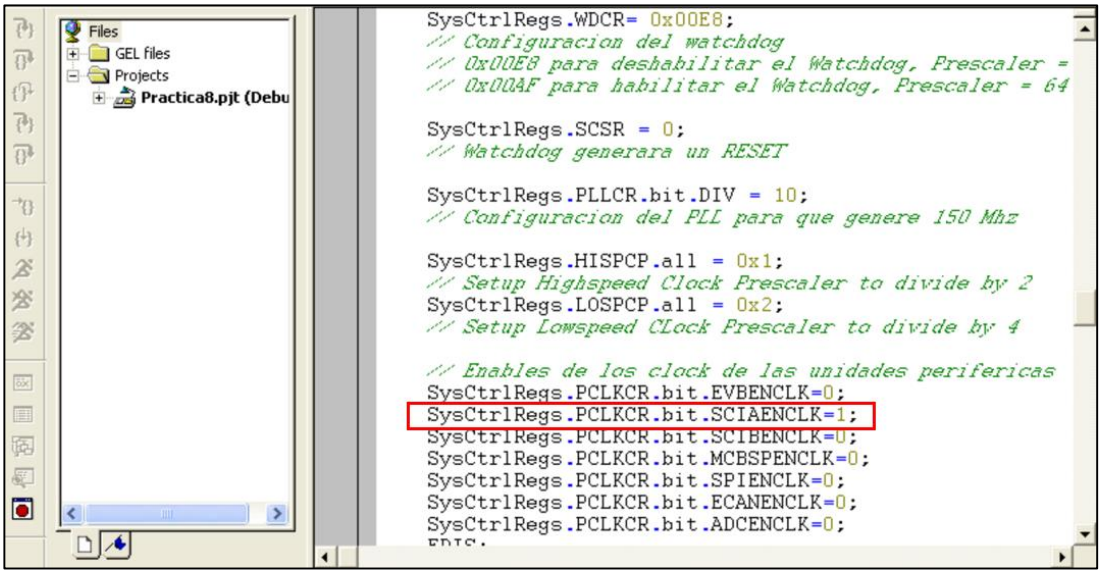


Figura 4.116 Habilitación de Reloj de unidad periférica de comunicación SCI

15. Configurar como función primaria los pines del puerto F en la función “Gpio_select()”, los cuales están asociados a la unidad periférica de comunicación SCI. Ver Figura 4.117

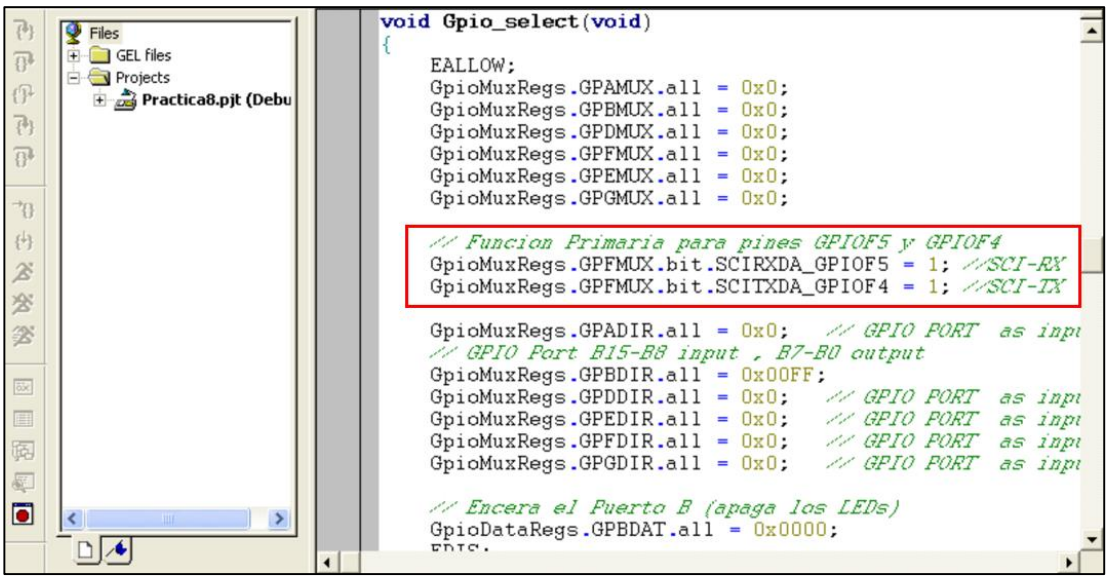


Figura 4.117 Configuración de función primaria en los pines GPIOF4 y GPIOF5

16. Declarar el prototipo **“void SCI_Init(void)”** al inicio del archivo código fuente y al final definirlo (Ver Figura 4.118) en base a lo siguiente:

Para el registro SCICCR:

- Configurar un bit de STOP
- Deshabilitar paridad y Loop back
- Configurar 8 bits de datos por carácter
- Seleccionar modo Idle-line

Para el registro SCICTL1:

- Resetear SCI
- Habilitar transmisión y recepción
- Deshabilitar modo SLEEP, TXWAKE e interrupciones por errores de recepción.

Para el registro SCITL2:

- Habilitar la interrupción de transmisión SCI
- Habilitar la interrupción de recepción SCI

Para el registro SCIHBAUD / SCILBAUD:

- Configurar la velocidad a 9600 baud rate, sabiendo que la frecuencia de LSPCLK es de 37.5 MHz.

Para el registro SCIFFTX:

- Reanudar la unidad FIFO de transmisión y recepción
- Habilitar las mejoras de la unidad FIFO
- Resetear el puntero de la unidad FIFO a cero
- Limpiar el indicador de interrupciones de transmisión FIFO
- Habilitar la condición TXFFIVL para que ocurra una interrupción de transmisión FIFO

- Establecer el nivel de interrupción FIFO para que se accione cuando FIFO esté vacío

Para el registro SCIFFRX:

- Limpiar el indicador de desbordamiento RXFFOVF
- Habilitar la operación de recepción FIFO
- Limpiar el indicador de interrupciones de recepción FIFO
- Habilitar la condición RXFFIVL para que ocurra una interrupción de recepción FIFO
- Establecer el nivel de interrupción de recepción FIFO para que se accione cuando FIFO llegue al nivel configurado de 5

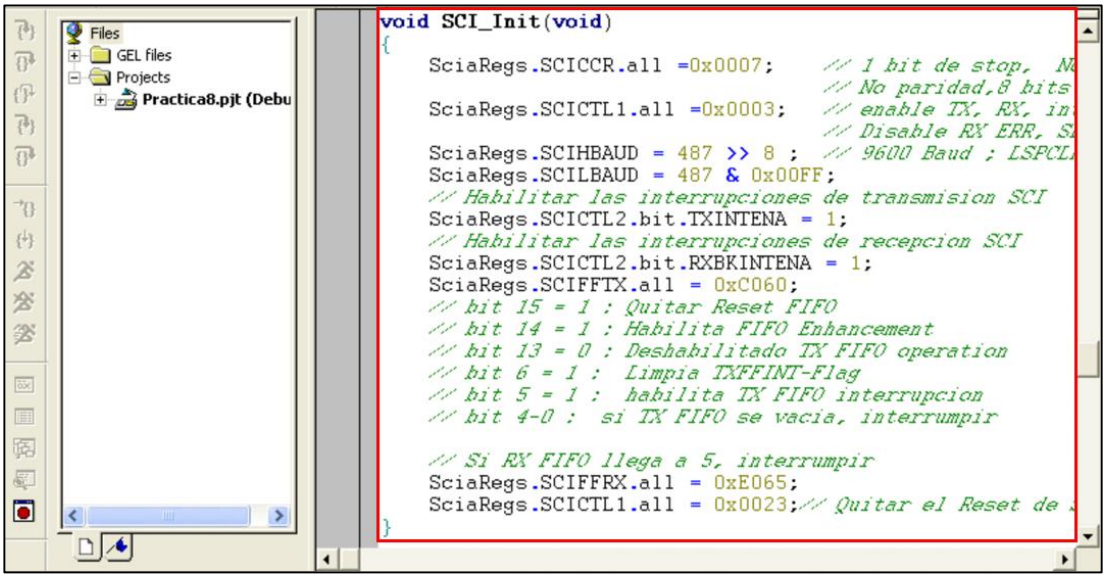


Figura 4.118 Definición de función "SCI_Init( )"

17. Agregar la función **"SCI_Init( )"** en el "main" antes de ingresar al lazo "while(1)", como se observa en la Figura 4.119

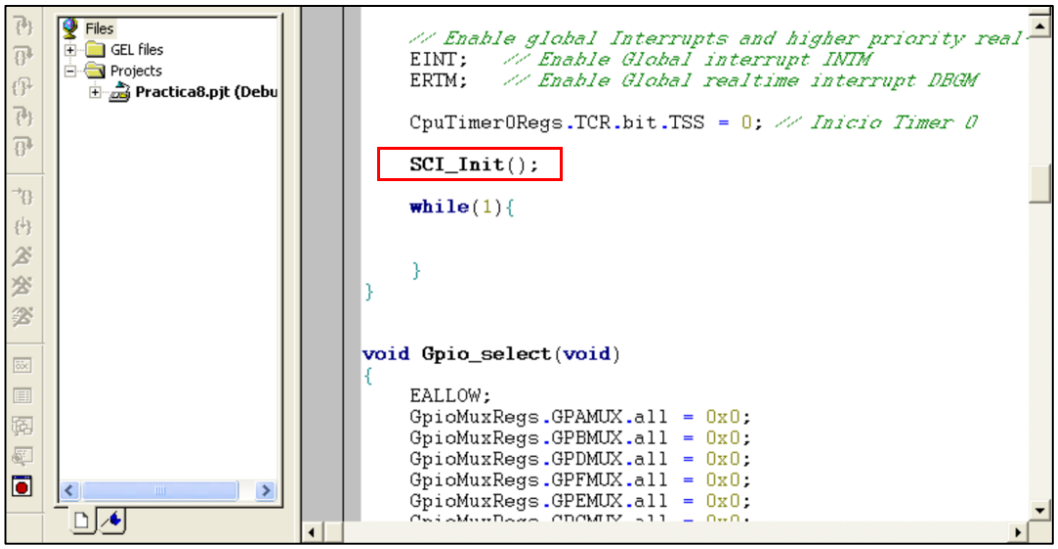


Figura 4.119 Llamado a la función "SCI_Init( )" en el "main"

18. Declarar como variables globales "mensaje", "speed", "envio" y "recepción" debido a que no serán utilizadas en el "main" sino en funciones adicionales. Esta declaración se debe especificar en el inicio del código como se aprecia en la Figura 4.120

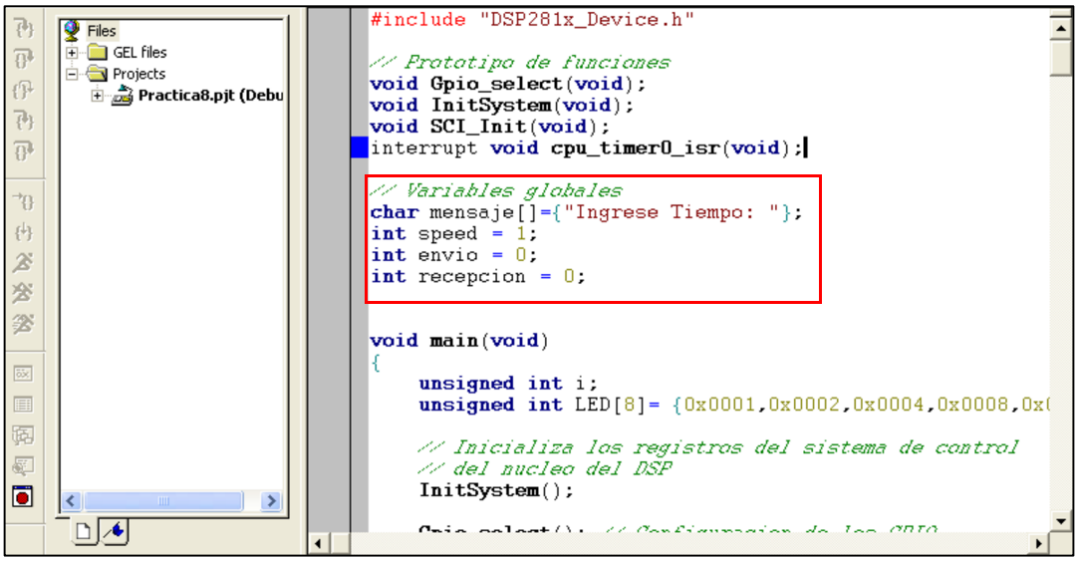


Figura 4.120 Declaración e Inicialización de variables globales a usar en la práctica

Como se puede apreciar, la frase para esta práctica contiene 16 caracteres, esto se debe a que la unidad FIFO de transmisión tiene 16 niveles en la que puede colocar un caracter por nivel.

19. Definir el servicio de interrupción “SCI_TX_isr( )”. Al inicio del programa se debe declarar esta función y al final del código se la define (Ver Figura 4.121) en base a lo siguiente:

- Mediante un lazo, cargar los 16 caracteres dentro la unidad de transmisión FIFO mediante el registro SCITXBUF que únicamente sirve ahora de intermediario
- Resetear el puntero de recepción de la unidad FIFO y habilitar la operación de recepción de la unidad. Esto se realiza para no adquirir lectura errónea en la unidad FIFO de recepción a la hora de pedir los datos
- Llamar a la función de reconocimiento para resetear el registro PIEACK

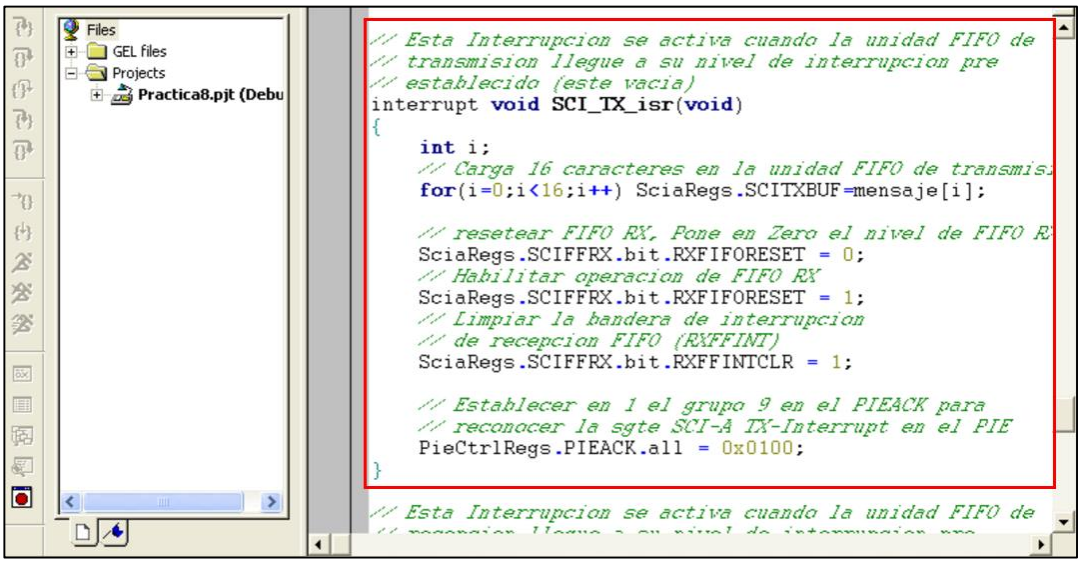


Figura 4.121 Definición de rutina de servicio de interrupción para transmisión SCI usando FIFO

La rutina de servicio será llamada cuando el nivel de interrupción FIFO sea alcanzado. Debido a que la configuración del nivel de interrupción FIFO fue establecido en cero, se podrán cargar los 16 caracteres en de la unidad de transmisión FIFO.

20. Definir el servicio de interrupción “SCI_RX_isr( )”. Al inicio del programa se debe declarar esta función y al final del código se la define (Ver Figura 4.122) en base a lo siguiente:

- Mediante un lazo, leer los caracteres de la unidad de transmisión FIFO mediante el registro SCIRXBUF
- Tomar el dato leído, convertirlo a un dato numérico con la función “atoi” siempre y cuando el quinto caracter sea un espacio (el caracter espacio representará la confirmación del valor ingresado) y guardarlo en la variable “*speed*”
- Limpiar la bandera de interrupción de recepción de la unidad FIFO
- Llamar a la función de reconocimiento para resetear el registro PIEACK

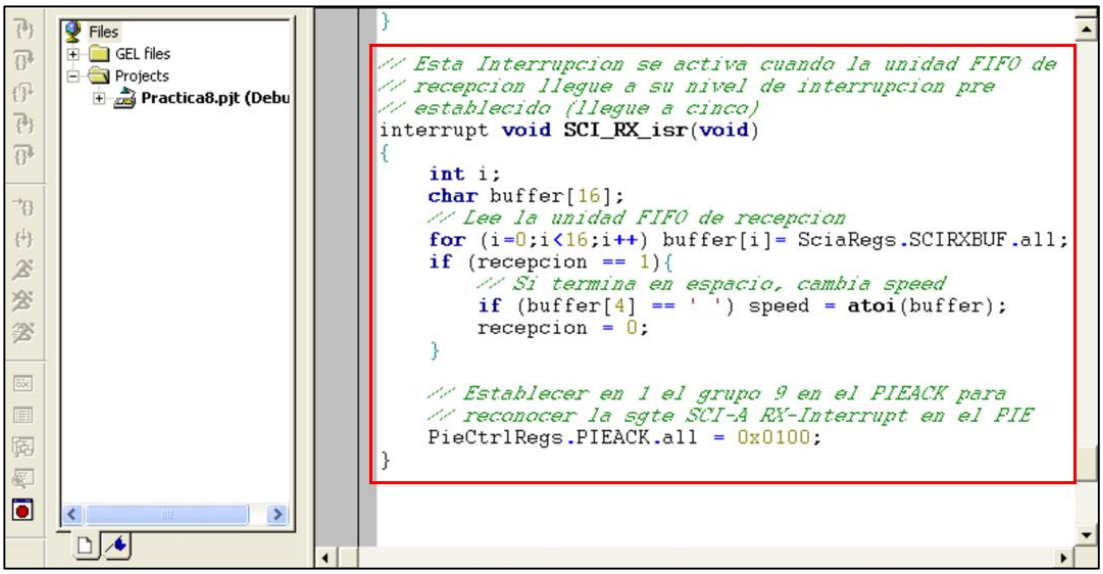


Figura 4.122 Definición de rutina de servicio de interrupción para recepción SCI usando FIFO

La rutina de servicio será llamada cuando el nivel de interrupción FIFO sea alcanzado. Debido a que la configuración del nivel de interrupción FIFO fue establecido en cinco, se podrán leer 5 caracteres de la unidad de recepción FIFO, donde cuatro primeros serán datos y el último será de confirmación.

21. Habilitar la interrupción de transmisión SCI en el registro de control PIE y escribir la dirección de la rutina del servicio de interrupción en la tabla de vectores PIE. Antes de la línea del habilitador global de interrupciones “EINT” como se aprecia en la Figura 4.123

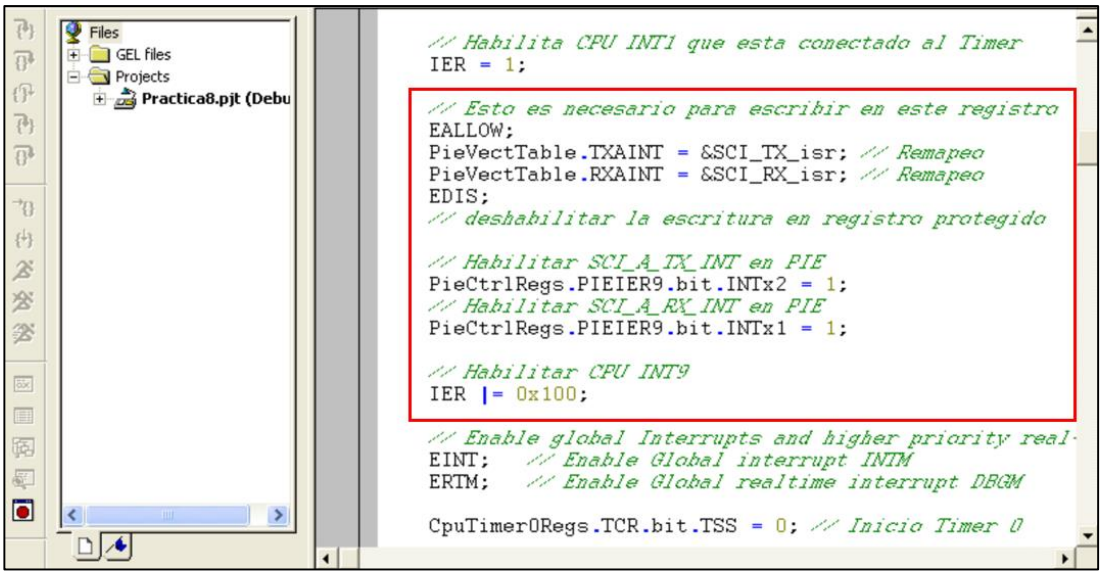


Figura 4.123 Configuración de interrupciones por transmisión y recepción SCI

PROGRAMACIÓN DEL “MAIN LOOP”

22. En el interior del “while(1)”:

- Mediante un lazo, se recorre la secuencia de encendido de los leds. El cambio entre el encendido de un led y otro dependerá de la variable “*speed*”
- Dentro del lazo, habilitar la operación de transmisión FIFO y limpiar la bandera de interrupción de transmisión FIFO, todo esto siempre y cuando se haya presionado el pulsador conectado a GPIOD1
- Dado que se desea desplazar el encendido de los leds en función de la variable “speed” y sabiendo que “CPUTimer0.InterrupCount” se incrementa en 1 cada 10 ms, se codifica una espera en la que “CPUTimer0.InterrupCount” incrementa hasta llegar al mismo valor de “speed”
- Luego del desplazamiento de un led, se le asigna el valor 0 a “CPUTimer0.InterrupCount” para resetearlo.

En la Figura 4.124 se muestra la programación que se debe hacer dentro del lazo “while(1)” para el manejo de la transmisión y recepción, y del control de velocidad en la secuencia de encendido de los leds.

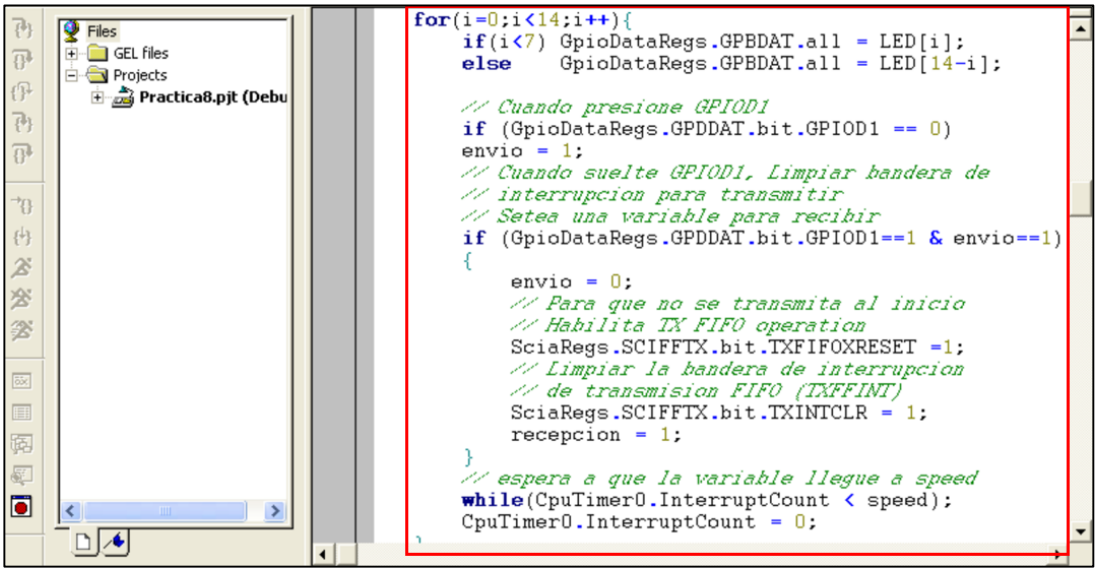


Figura 4.124 Programación dentro del lazo "while(1)" en el “main”

COMPILAR, CARGAR Y EJECUTAR EL PROGRAMA

23. Compilar el proyecto seleccionando*: Project -> Rebuild All*. Solucionar los errores en caso de haberlos.

24. Alimentar la tarjeta *eZdsp™F2812*.

25. Conectarse con el procesador: *Debug -> Connect*

26. Cargar el programa (*File -> Load Program…)* **Practica8.out** que se genera dentro de la carpeta Debug al compilar el proyecto

27. Ejecutar el programa cargado: *Debug -> Run*

28. Configurar las propiedades del puerto COM de la PC de acuerdo al formato de la trama SCI.

29. Abrir programa Hyper Terminal para colocar nombre del proyecto y luego dar click en **OK**

30. Al abrirse la ventana “Connect To”, se debe seleccionar el puerto por el cual se realizará la comunicación para luego dar click en **Configure…**

31. Configurar las propiedades del puerto COM del programa HyperTerminal de acuerdo al formato de la trama SCI. Para finalizar dar click en **Aceptar** y luego en **OK**.

32. Se tiene que configurar la transmisión de caracteres por parte del programa HyperTerminal. Para ello acceder a las propiedades en el programa HyperTerminal. (*File -> Properties).*

33. Ir a la pestaña “Setting” de la ventana de propiedades (Ver Figura 4.125), y dar clic sobre el botón **“ASCII Setup”**

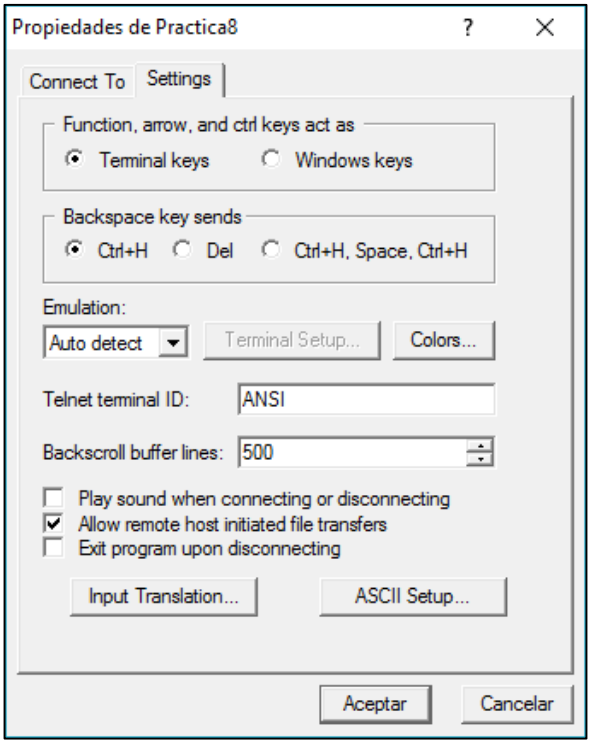


Figura 4.125 Propiedades del programa HyperTerminal

34. En la ventana de configuraciones ASCII, seleccionamos los dos casilleros de “ASCII Sending”, como se observa en la Figura 4.126

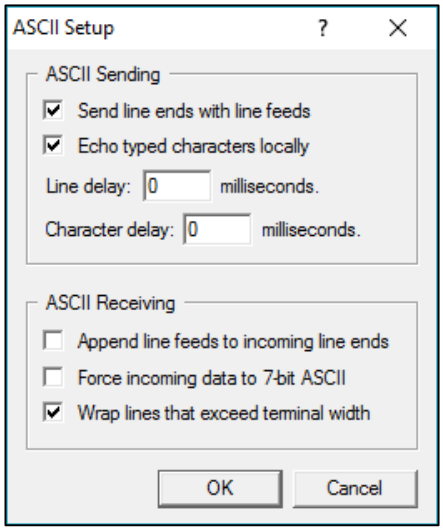


Figura 4.126 Configuraciones ASCII en el programa HyperTerminal

35. Para finalizar la configuración de transmisión en el programa HyperTerminal, dar clic en **OK** en la ventada de “ASCII Setup” y luego en **Aceptar** en la ventada de propiedades

36. Comprobar el funcionamiento del programa:

En la Figura 4.127 se observa que al ejecutar el programa los leds parecen estar prendidos, pero en verdad estas desplazando su encendido de 1 milisegundo entre led y led.

Figura 4.127 Foto del Kit inicia con un tiempo de desplazamiento de 1 milisegundo

Cada que se presione el pulsador S1, se pedirá por pantalla el ingreso de un valor de tiempo en milisegundos (0 y 9999 ms) que se debe validar con el caracter espacio. Al ingresar el valor, se visualizará el cambio de tiempo de encendido entre led y led de la secuencia.

En la Figura 4.128 se puede observar el mensaje transmitido por el DSP cuando se presiona el pulsador S1, solicitando un tiempo.

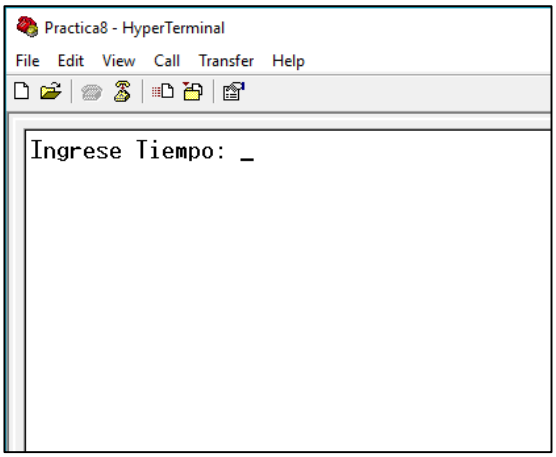


Figura 4.128 Petición de tiempo por parte del DSP

En la Figura 4.129 se observa la petición de tres tiempos, y la respuesta por parte del usuario. En las Figuras 4.130, 4.131, 4.132 se observa Fotos del Kit en el cual se aprecia la velocidad de secuencia de los leds a 1000, 100 y 10 ms (milisegundos) respectivamente.

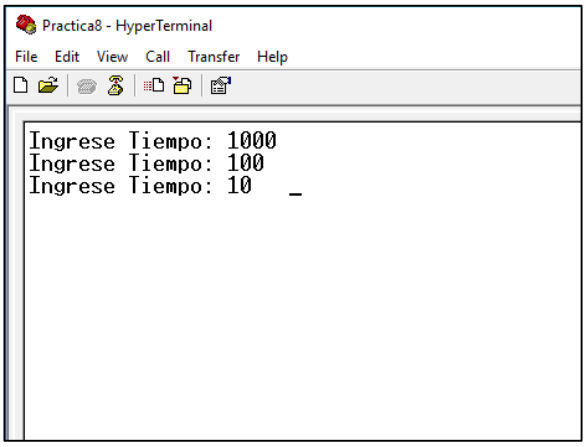


Figura 4.129 Pruebas para transmisión de varios tiempos

Figura 4.130 Foto del Kit cuando ha recibido 1 segundo

Figura 4.131 Foto del Kit cuando ha recibido 100 milisegundos

Figura 4.132 Foto del Kit cuando ha recibido 10 milisegundos

37. Al finalizar la práctica, detener la ejecución del programa: *(Debug -> Halt)*

38. Resetear el CPU *(Debug -> Reset CPU)*, luego desconectarse de la tarjeta *(Debug -> Disconnect)*

39. Quitar la alimentación de la tarjeta *eZdsp™F2812*.

PRÁCTICA # 9

TEMA:

"Transmisión de trama de datos desde el procesador TMS320F2812 usando el protocolo de comunicación CAN y el transceptor SN 65 HVD 230"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Transmitir con el procesador *F2812* un byte de datos en la trama cada segundo mediante protocolo CAN.
- Configurar el formato de la trama, la velocidad y el Mailbox.
- Establecer comunicación CAN entre dos procesadores *TMS320F2812*.
- Usar "CPU Timer0" para generar un intervalo de un segundo.

REQUISITOS MÍNIMOS:

- Computador con el software Code Composer Studio (versión 3.3) instalado y su respectivo drive para el manejo del convertidor JTAG a USB.
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos básicos en sistemas de comunicación industrial
- Conocimientos de programación en Lenguaje C
- Haber culminado satisfactoriamente la Práctica # 2.
- Lectura y comprensión del archivo de fundamentación teórica: "CONTROLLER AREA NETWORK (CAN)"

BREVE EXPLICACIÓN DE LA PRÁCTICA

El procesador *TMS320F2812* embebe la unidad periférica de comunicación CAN, la cual es compatible con el estándar CAN2.0B. Su uso establece un protocolo para comunicarse de forma serial con otros procesadores en ambientes de permanente

ruido eléctrico. Esta unidad periférica provee una robusta y versátil interfaz de comunicación serial.

El procesador (DSP) se configurará para transmitir una trama de un byte de datos, el cual contiene el estado de los switches de entrada (conectados a GPIOB15…GPIOB8), también se configura el uso del identificador extendido 0x1000 0000 para transmitir el mensaje, el uso del Mailbox#5 como mailbox de transmisión, la velocidad de transmisión de 100Kbps. La trama se debe transmitir cada segundo. Ver Figura 4.133

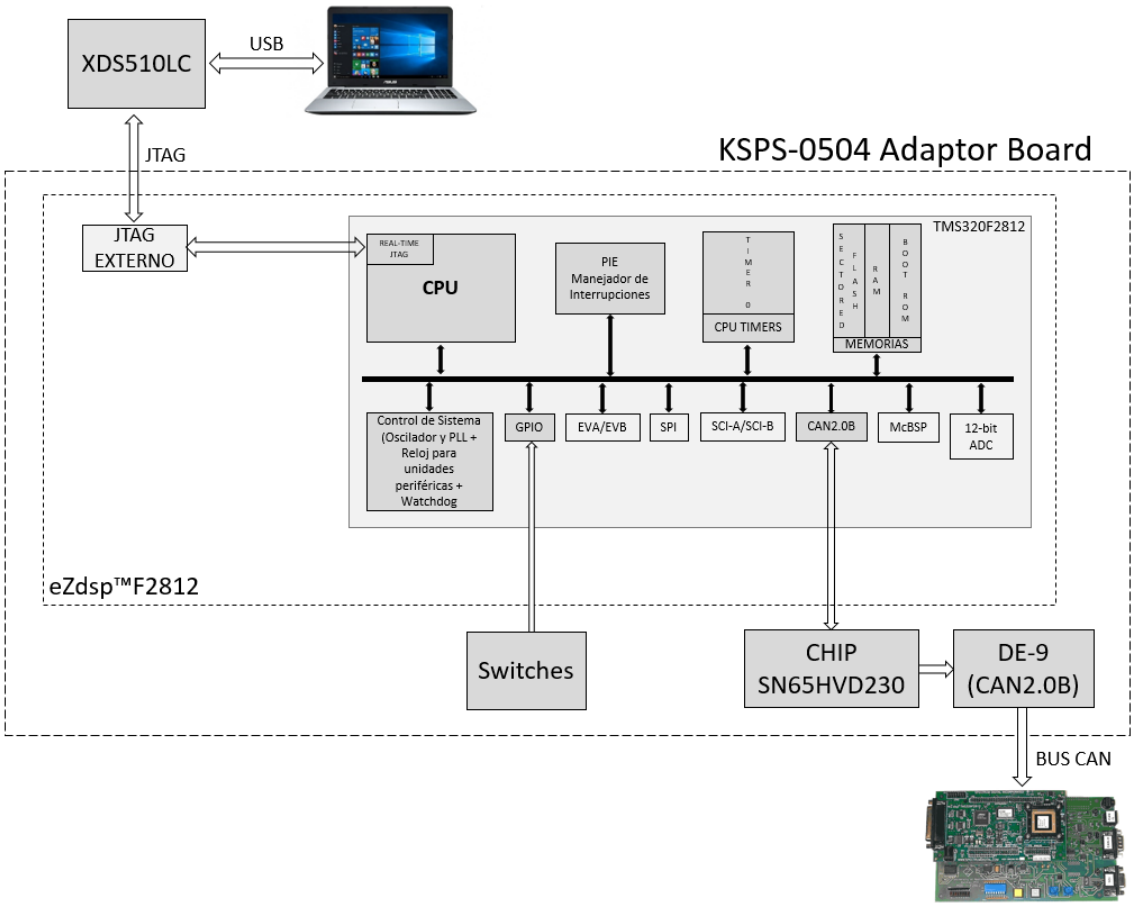


Figura 4.133 Diagrama de Bloques de Transmisión mediante protocolo CAN

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Code Composer Studio*.

NOTA: Si al abrir el programa sale algún error de conexión con la PC, el motivo podría ser que no esté configurado el *Setup CCStudio V3.3* tal como se ve en la practica 1. Si el estudiante no ha realizado la *practica 1* se recomienda desarrollarla para evitar tener problemas de comunicación entre su PC y las tarjetas, para luego continuar con esta práctica.

3. Crear un proyecto nuevo (*Project -> New..)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica9
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** para crear el proyecto

4. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

// Prototipo de funciones
void Gpio_select(void);
void InitSystem(void);
interrupt void cpu_timer0_isr(void);

void main(void)
{
        InitSystem();
        Gpio_select();

        InitPieCtrl();
        InitPieVectTable();

        EALLOW;
```

```
        PieVectTable.TINT0 = &cpu_timer0_isr;
        EDIS;

        InitCpuTimers();

        ConfigCpuTimer(&CpuTimer0, 150, 50000);

        PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
        IER = 1;
        EINT;
        ERTM;

        CpuTimer0Regs.TCR.bit.TSS = 0;

        while(1)
        {
           while(CpuTimer0.InterruptCount < 20);
           CpuTimer0.InterruptCount = 0;
        }
}

void Gpio_select(void)
{
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;
        GpioMuxRegs.GPBMUX.all = 0x0;
        GpioMuxRegs.GPDMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;

        GpioMuxRegs.GPADIR.all = 0x0;
        GpioMuxRegs.GPBDIR.all = 0x00FF;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
        GpioMuxRegs.GPFDIR.all = 0x0;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioDataRegs.GPBDAT.all = 0x0000;
   EDIS;
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00E8;

        SysCtrlRegs.SCSR = 0;

        SysCtrlRegs.PLLCR.bit.DIV = 10;

        SysCtrlRegs.HISPCP.all = 0x1;
        SysCtrlRegs.LOSPCP.all = 0x2;
```

```
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
}

interrupt void cpu_timer0_isr(void)
{
        CpuTimer0.InterruptCount++;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

5. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica9.c"***

6. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica9.c".***

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

7. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **DSP281x_GlobalVariableDefs.c**

8. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_Headers_nonBIOS.cmd**

9. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

10. De la dirección "*C:\tidcs\c28\dsp281x\v100\DSP281x_common\source"* agregar al proyecto los archivos:

- **DSP281x_PieCtrl.c**

- **DSP281x_PieVect.c**
- **DSP281x_DefaultIsr.c**
- **DSP281x_CpuTimers.c**

11. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL "BUILD OPTIONS"

12. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options..*
- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include
  en el campo *Include Search Path*

13. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **400** en el campo *Stack Size.*
- Dar clic en el botón *OK* para terminar la configuración del *Build Options*

MODIFICAR CÓDIGO FUENTE DE ENCABEZADO ECAN

14. Antes de empezar con la edición del código, se debe modificar una porción del archivo de encabezado **"DSP281x_ECan.h"**, Versión 1.0, localizado en *C:\tidcs\c28\dsp281x\v100\DSP281x_headers\include* y provisto por Texas Instruments. Se deben buscar y modificar las estructuras "CANMDL_BYTES" y "CANMDH_BYTES" para que queden como las Figuras 4.134 y 4.135 respectivamente.

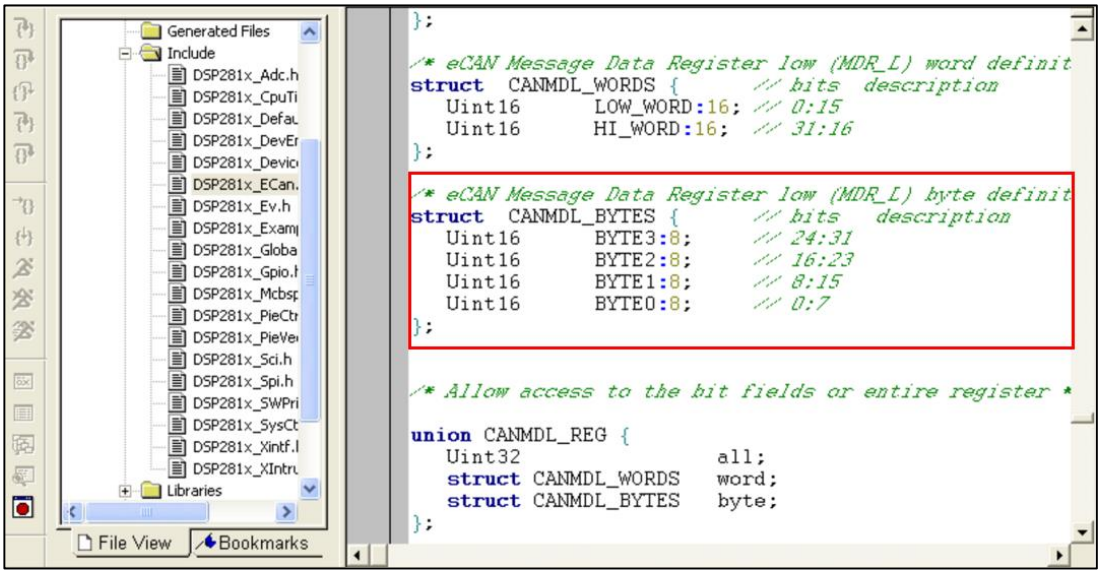


Figura 4.134 Edición de la estructura "CANMDL_BYTES"

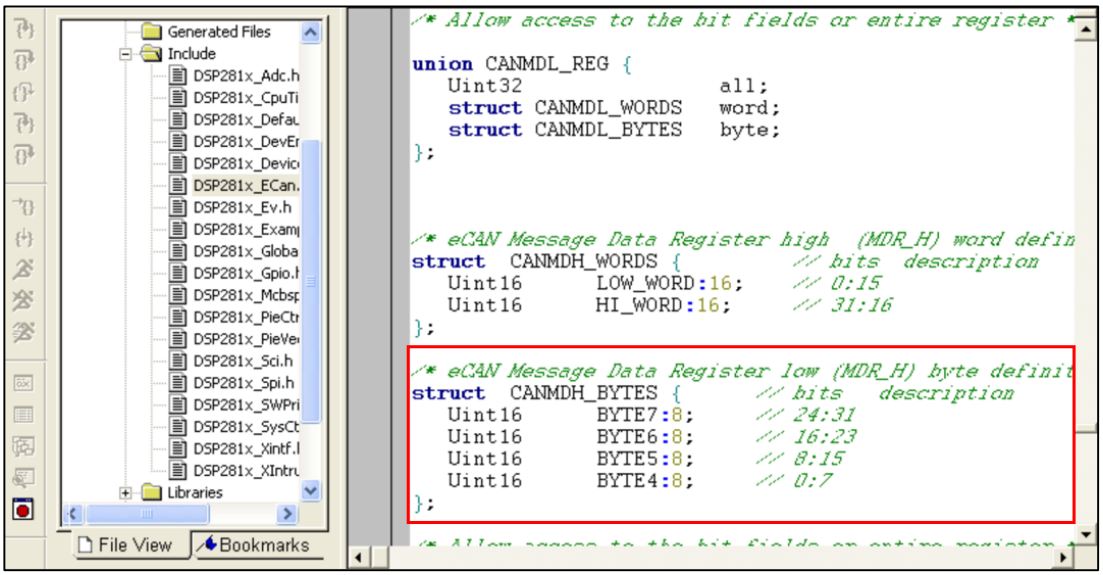


Figura 4.135 Edición de la estructura "CANMDH_BYTES"

COMPILAR EL PROGRAMA

15. Compilar el proyecto seleccionando*: Project -> Rebuild All*.

MODIFICAR CONTENIDO DE CÓDIGO FUENTE

16. Añadir una nueva estructura "ECanaShadow" como una variable local en el “main”. Ver Figura 4.136

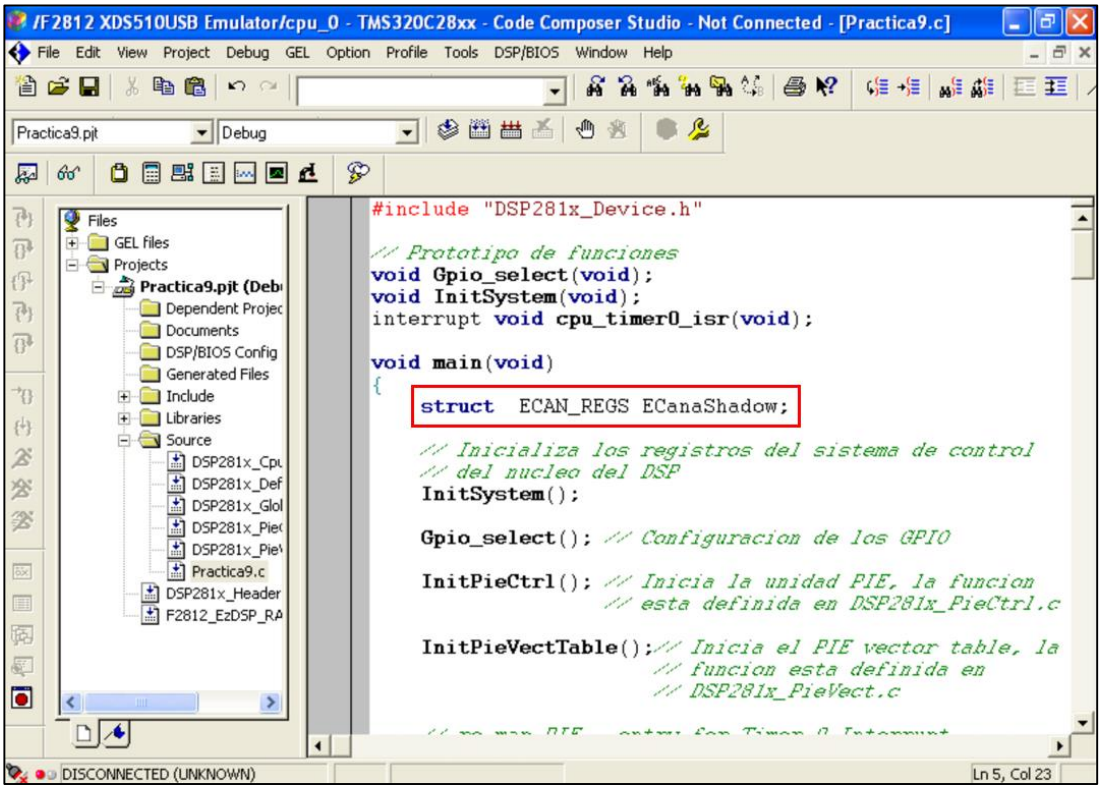


Figura 4.136 Declaración de variable local como estructura de registros CAN

Esta estructura será usada como una copia local de los registros CAN originales. Su creación es necesaria para manipular bits individuales en algunos registros que únicamente son accesibles mediante escritura de todos sus bits a la vez. La manipulación individual se la podrá realizar en la copia creada para luego cargarla a los registros originales CAN.

17. En la definición de la función “InitSystem()” habilitar la unidad de reloj para el modulo CAN, como se observa en la Figura 4.137

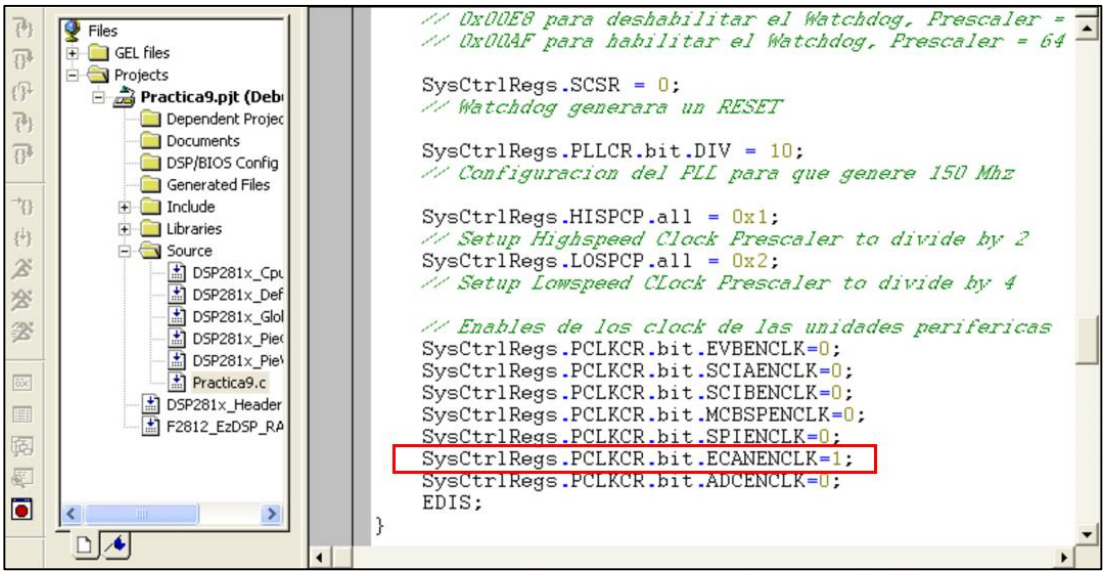


Figura 4.137 Habilitación de Reloj de unidad periférica de comunicación CAN

18. Dentro de la función “Gpio_select( )” seleccionar la función primaria de los bits 6 y 7 (CANTxA y CANRxA) del puerto F. Ver Figura 4.138

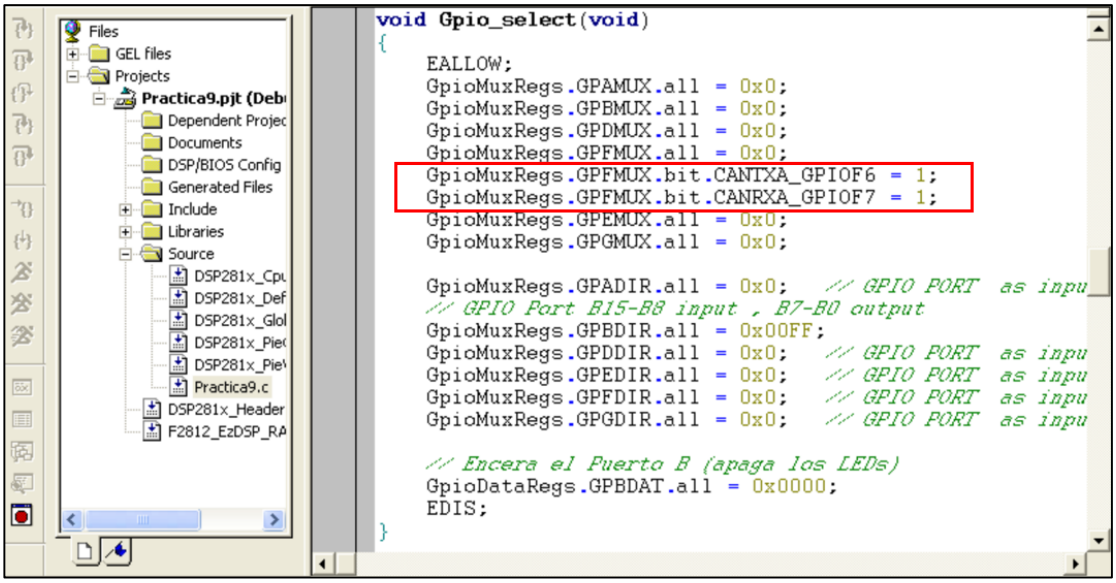


Figura 4.138 Configuración de función primaria en los pines GPIOF6 y GPIOF7

AGREGAR CÓDIGO DE INICIALIZACIÓN DE CAN

19. Declarar el prototipo ***"void InitCan(void)"*** al inicio del archivo código fuente y al final del mismo definirlo (Ver Figura 4.139) en base a lo siguiente:

Para el registro CANTIOC:

- Configurar TXFUNC para que el pin CANTxA sea usado para funciones de transmisión CAN

Para el registro CANRIOC:

- Configurar RXFUNC para que el pin CANRxA sea usado para funciones de recepción CAN.

Para el registro CANMC:

- Habilitar el modo HECC (High−end CAN Controller) del módulo CAN
- Ajustar el bit CCR en 1 para permitir el acceso a la configuración del registro CANBTC

Nota: Antes de poder continuar con la inicialización, nosotros debemos esperar a que el modulo CAN haya procesado la solicitud. En este caso el indicador CCE del registro CANES se ajustará en 1 cuando el modulo CAN esté listo. Mediante una línea de código, configurar una espera para que se establezca en uno este indicador y así iniciar la configuración del tiempo de bit.

Para el registro CANBTC:

- Configurar los parámetros BRP, TSEG1 y TSEG2 para transmitir a 100 kbps a un punto de muestreo del 80% del tiempo de bit.

Nota: Para el registro CANMC ajustar el bit CCR en 0 para deshabilitar el acceso a la configuración del registro CANBTC, luego configurar una espera para que se encere la bandera del indicador CCE del registro CANES.

Para el registro CANME:

- Deshabilitar todos los mailboxes (ajustar en 0 todos los bits) para así poder permitir la escritura de los registros MSGID, en estos registros se almacenarán los identificadores de mensaje.

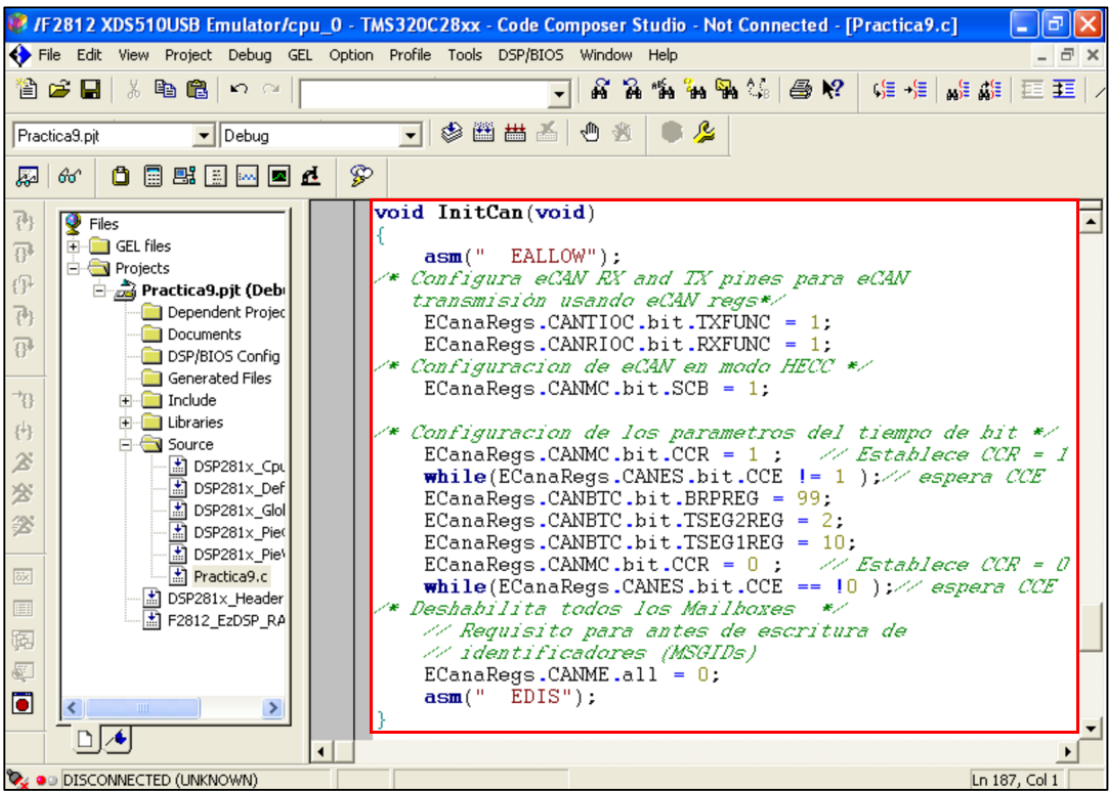


Figura 4.139 Definición de la función "InitCan( )"

DENTRO DEL MAIN, CONFIGURAR EL MAILBOX #5 PARA TRANSMITIR

20. Llamar a la función "InitCan( )" en el "main" antes de inicializar el conteo del CPU timer0, como se observa en la Figura 4.140

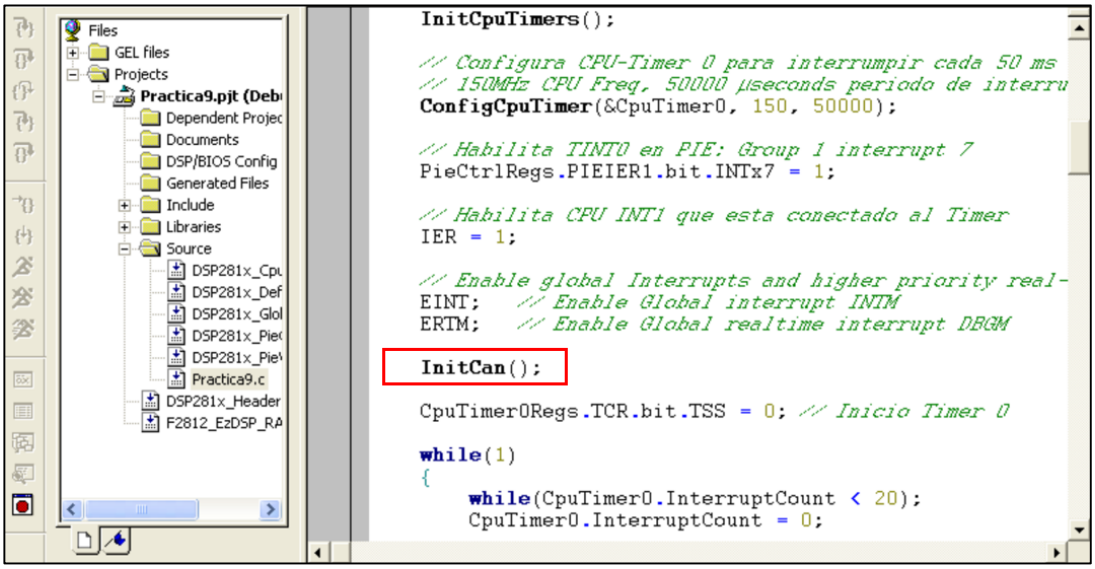


Figura 4.140 Llamado a la función "InitCan( )"

21. En el interior del "main", luego del llamado a la función "InitCan( )", añadir código para preparar la transmisión del Mailbox (Ver Figura 4.141). En este ejercicio, se utilizará el Mailbox #5, una identificación extendida de 0x10000000 y un tamaño de dato de 1 byte. Para ello:

- Escribir el identificador en el interior del registro MSGID.
- Establecer el bit "IDE" del registro MSGID para transmitir con identificación extendida.
- Configurar el Mailbox #5 como Mailbox de transmisión. Para ello se ajusta en 0 el bit MD5 del registro CANMD. Debido a la estructura interna de la unidad CAN, no se puede acceder a modificar un bit de los registros originales CAN, una buena práctica es copiar todos los valores de un registro hacia un "Shadow Register", manipularlo y copiar los 32 bits del "Shadow Register" con las modificaciones ajustadas al registro original.
- Habilitar el Mailbox #5, utilizar la ayuda de "Shadow Register".

- Establecer en 1 el campo del tamaño de datos (DLC) del mensaje en el registro MSGCTRL.

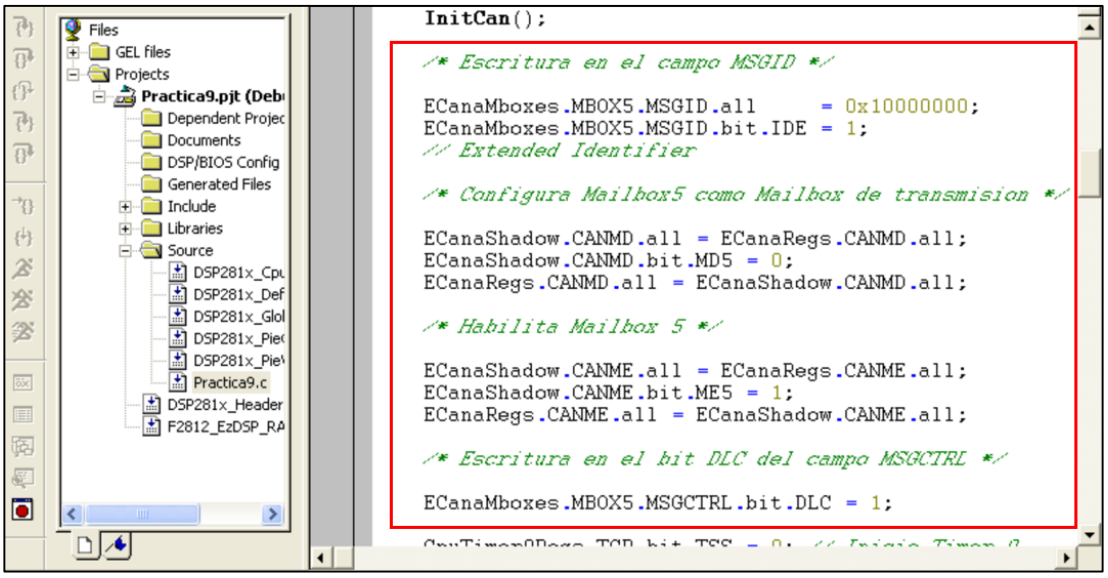


Figura 4.141 Configuración del Mailbox # 5 como Mailbox de transmisión

DENTRO DEL MAIN, TRANSMITIR 1 BYTE DE DATOS

En esta parte del código se agregará el estado de los switches (conectados a GPIO B15…B8) en el byte 0 del campo de datos del Mailbox #5.

22. Dentro del lazo “while(1)” del “main” se debe codificar para transmitir el estado de los switches cada segundo, para ello:

- Dado que se desea transmitir cada segundo, y sabiendo que “CPUTimer0.InterrupCount” se incrementa en 1 cada 50 ms, se codifica una espera en la que “CPUTimer0.InterrupCount” incrementa hasta llegar al valor de 20 (generando 1 segundo).
- Luego de haber transcurrido el segundo, se le asigna el valor 0 a “CPUTimer0.InterrupCount” para iniciar un nuevo conteo.
- Cargar el estado actual de los 8 switches de entrada en el registro **“ECanaMboxes.MBOX5.MDL.byte.BYTE0”**

- Solicitar la transmisión del Mailbox #5. En el registro CANTRS de la estructura ECanaShadow se ajustará en 1 el bit TRS5 y los demás 31 bits en 0. Luego se carga todo el registro CANTRS de la estructura ECanaShadow al registro CANTRS de la estructura ECanaRegs.
- Esperar hasta que la unidad CAN reconozca la solicitud de transmisión. La bandera “ECanaRegs.CANTA.bit.TA5” se establecerá en 1 si la solicitud se reconoce.
- Limpiar la bandera “ECanaRegs.CANTA.bit.TA5” (asignando 1 al bit), para ello utilizar la ayuda de “Shadow Register”.

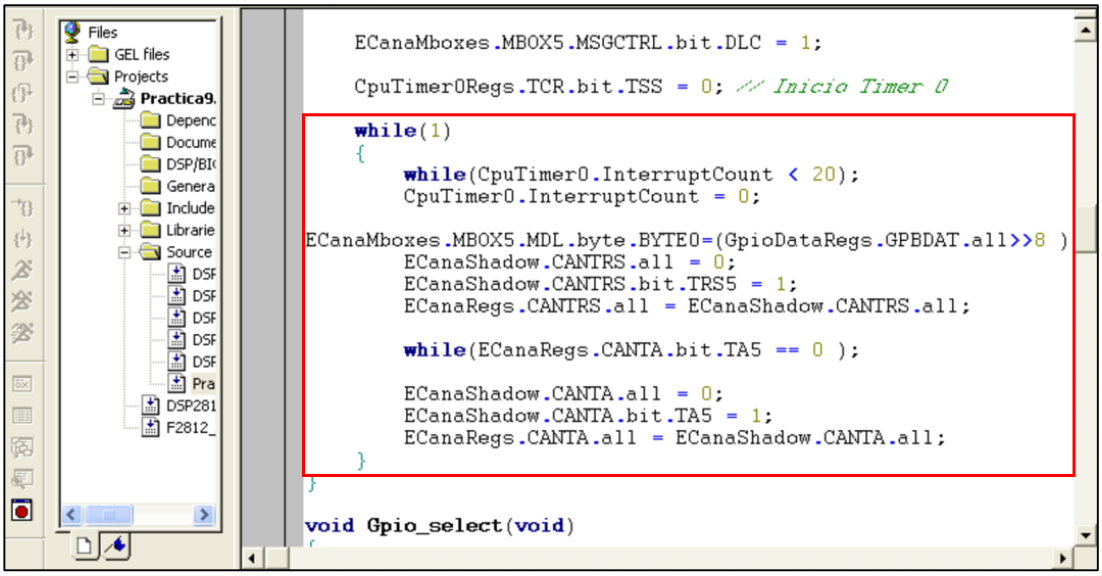


Figura 4.142 Transmisión del estado de los switches mediante protocolo CAN

COMPILAR, CARGAR Y EJECUTAR EL PROGRAMA

23. Colocar los jumpers JP5 y JP6 de la tarjeta *KSPS-0504 Adaptor board* en la posición 1-2 con el objetivo de seleccionar el integrado SN65HVD230 que generará el bus CAN.

24. Colocar el jumper JP4 de la tarjeta *KSPS-0504 Adaptor board* en la posición 2-3 con el objetivo de habilitar la resistencia de terminación de 120 [ohms]

Para comprobar la transmisión de la trama CAN, se debe emplear un segundo kit de prácticas con el procesador TMS320F2812, el cual debe estar programado para recibir la trama CAN y mostrarla en sus respectivos leds (Práctica #10).

25. Enlazar ambas tarjetas, conectando punto a punto los pines 2, 7 y 3 de los conectores X2 (DE9 macho) de ambas tarjetas *KSPS-0504 Adaptor board.*

26. Compilar el proyecto seleccionando*: Project -> Rebuild All*. Solucionar los errores en caso de haberlos.

27. Alimentar la tarjeta *eZdsp™F2812*.

28. Conectarse con el procesador: *Debug -> Connect*

29. Cargar el programa (*File -> Load Program…)* **Practica9.out** que se genera dentro de la carpeta Debug al compilar el proyecto

30. Ejecutar el programa cargado: *Debug -> Run*

31. Desconectarse del procesador: *Debug -> Disconnect*

32. Comprobar el funcionamiento del programa:

Para comprobar la práctica 9, se debe programar el segundo Kit para recibir la trama CAN y mostrarla en sus respectivos leds (Práctica #10).

33. Al finalizar la práctica, conectarse con el procesador: *Debug -> Connect*

34. Detener la ejecución del programa: *(Debug -> Halt)*

35. Resetear el CPU *(Debug -> Reset CPU)*, luego desconectarse de la tarjeta *(Debug -> Disconnect).* Y quitar la alimentación de la tarjeta *eZdsp™F2812*.

PRÁCTICA # 10

TEMA:

"Recepción de trama de datos en el procesador TMS320F2812 usando el protocolo de comunicación CAN y el transceptor SN 65 HVD 230"

OBJETIVOS:

Al concluir satisfactoriamente la práctica usted estará en capacidad de:

- Recibir con el procesador TMS320F2812 un byte de datos en la trama mediante protocolo CAN.
- Configurar el formato de la trama, la velocidad y el Mailbox.
- Establecer comunicación CAN entre dos procesadores *TMS320F2812*.

REQUISITOS MÍNIMOS:

- Computador con el software Code Composer Studio (versión 3.3) instalado y su respectivo drive para el manejo del convertidor JTAG a USB.
- Conocimientos básicos en microprocesadores y microcontroladores.
- Conocimientos básicos en sistemas de comunicación industrial
- Conocimientos de programación en Lenguaje C
- Haber culminado satisfactoriamente la Práctica # 2.
- Lectura y comprensión del archivo de fundamentación teórica: "CONTROLLER AREA NETWORK (CAN)"

BREVE EXPLICACIÓN DE LA PRÁCTICA

El procesador *TMS320F2812* embebe la unidad periférica de comunicación CAN, la cual es compatible con el estándar CAN2.0B. Su uso establece un protocolo para comunicarse de forma serial con otros procesadores en ambientes de permanente

ruido eléctrico. Esta unidad periférica provee una robusta y versátil interfaz de comunicación serial.

El procesador se configurará para recibir una trama de un byte de datos y mostrarlos en los leds de salida conectados a GPIOB15…GPIOB8, también se configura el uso del identificador extendido 0x1000 0000 para recibir el mensaje, el uso del Mailbox#1 como Mailbox de recepción, la velocidad de transmisión de 100Kbps. La trama se recibirá cada segundo. Ver Figura 4.143

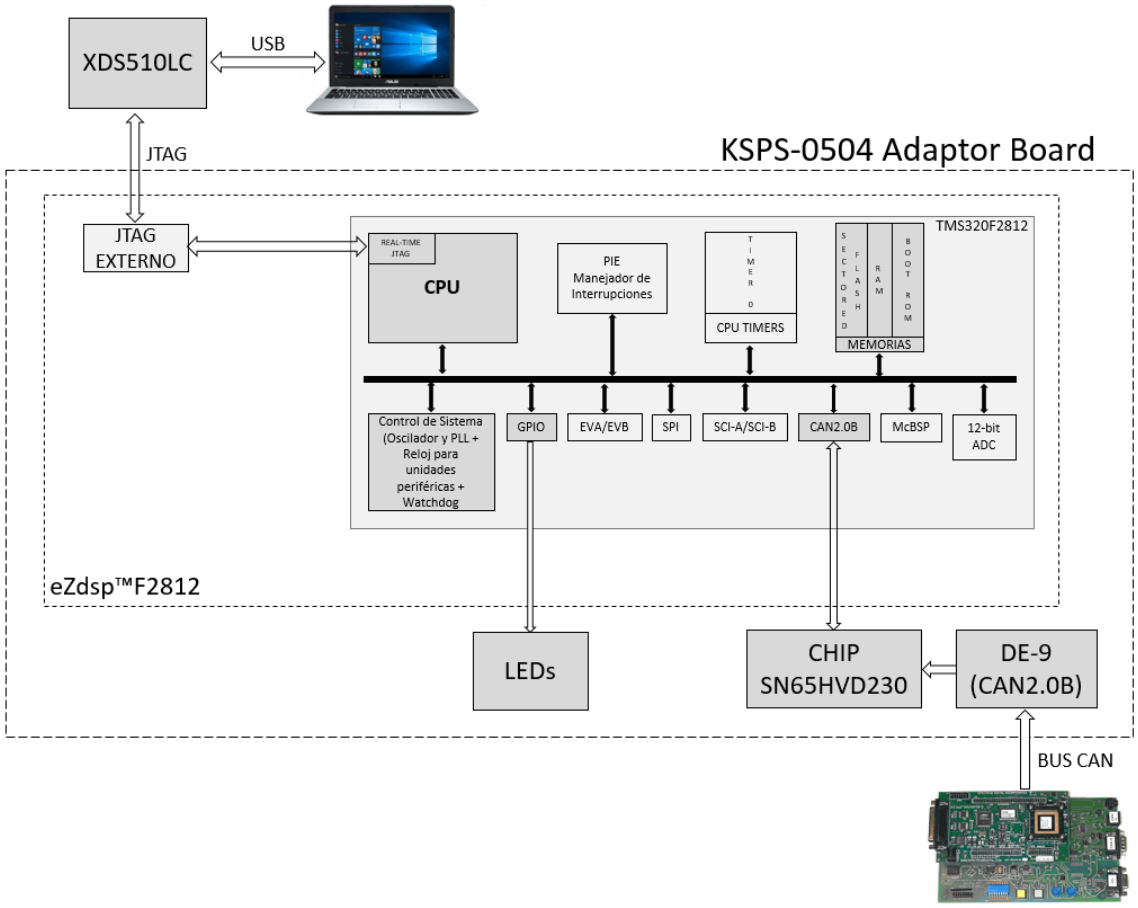


Figura 4.143 Diagrama de Bloques de Recepción mediante protocolo CAN

DESARROLLO DE LA PRÁCTICA

CREAR PROYECTO Y AGREGAR ARCHIVOS AL PROYECTO

1. Conectar el bus del puerto JTAG del convertidor *XDS510LC* al conector P1 de la tarjeta *eZdsp™F2812*, y conectar el puerto USB del convertidor *XDS510LC* a un puerto USB de la PC que tenga instalado el programa Code Composer Studio.

2. Abrir el programa *Code Composer Studio*.

NOTA: Si al abrir el programa sale algún error de conexión con la PC, el motivo podría ser que no esté configurado el *Setup CCStudio V3.3* tal como se ve en la practica 1. Si el estudiante no ha realizado la *practica 1* se recomienda desarrollarla para evitar tener problemas de comunicación entre su PC y las tarjetas, para luego continuar con esta práctica.

3. Crear un proyecto nuevo (*Project -> New..)* en *Code Composer Studio* con los siguientes campos:

- Project Name: Practica10
- Project Type: Executable (.out)
- Target: TMS320C28xx

Dar clic en el botón **Finish** para crear el proyecto

4. Crear un nuevo archivo de código fuente *(File -> New -> Source File)* y copiar el código otorgado por el profesor en esta área de trabajo.

```
#include "DSP281x_Device.h"

// Prototipo de funciones
void Gpio_select(void);
void InitSystem(void);

void main(void)
{
        InitSystem();
        Gpio_select();

        while(1)
        {
        }
}
```

```
void Gpio_select(void)
{
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;
        GpioMuxRegs.GPBMUX.all = 0x0;
        GpioMuxRegs.GPDMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;

        GpioMuxRegs.GPADIR.all = 0x0;
        GpioMuxRegs.GPBDIR.all = 0x00FF;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
        GpioMuxRegs.GPFDIR.all = 0x0;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioDataRegs.GPBDAT.all = 0x0000;
        EDIS;
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00E8;

        SysCtrlRegs.SCSR = 0;

        SysCtrlRegs.PLLCR.bit.DIV = 10;

        SysCtrlRegs.HISPCP.all = 0x1;
        SysCtrlRegs.LOSPCP.all = 0x2;

        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
}
```

5. Guardar este archivo *(File -> Save)* y colocar como nombre ***"Practica10.c"***

6. Agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo de código fuente ***"Practica10.c".***

Dar clic en el botón **Open** para agregar el archivo de código fuente seleccionado.

7. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **DSP281x_GlobalVariableDefs.c**

8. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_Headers_nonBIOS.cmd**

9. De la dirección *"C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd"* agregar al proyecto (Project ->Add Files to Proyect...) el archivo:

- **F2812_ExDSP_RAM_lnk.cmd**

10. De la dirección *"C:\CCStudio_v3.3\c2000\cgtools\lib"* agregar al proyecto (*Project ->Add Files to Proyect...)* el archivo:

- **rts2800_ml.lib**

CONFIGURACIÓN DEL "BUILD OPTIONS"

11. Configurar la ruta de búsqueda para incluir los archivos de cabecera de los registros periféricos, para ello:

- Dar clic en *Project -> Build Options..*
- Seleccionar la categoría *Preprocessor* de la pestaña *Compiler* e incluir la dirección
  C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include en el campo Include Search Path

12. Configurar el Tamaño de la Pila, para ello:

- En la misma ventana de *Build Options*, Seleccionar la categoría *Basic* de la pestaña *Linker* y colocar el valor de **400** en el campo *Stack Size.*
- Dar clic en el botón *OK* para terminar la configuración del *Build Options*

MODIFICAR CÓDIGO FUENTE DE ENCABEZADO ECAN

13. Antes de empezar con la edición del código, se debe modificar una porción del archivo de encabezado **"DSP281x_ECan.h"**, Versión 1.0, localizado en *C:\tidcs\c28\dsp281x\v100\DSP281x_headers\include* y provisto por Texas Instruments. Se deben buscar y modificar las estructuras "CANMDL_BYTES" y "CANMDH_BYTES" para que queden de la siguiente manera:

```
struct CANMDL_BYTES { // bits description
        Uint16 BYTE3:8; // 31:24
        Uint16 BYTE2:8; // 23:16
        Uint16 BYTE1:8; // 15:8
        Uint16 BYTE0:8; // 7:0
        };
struct CANMDH_BYTES { // bits description
        Uint16 BYTE7:8; // 63:56
        Uint16 BYTE6:8; // 55:48
        Uint16 BYTE5:8; // 47:40
        Uint16 BYTE4:8; // 39:32
        };
```

COMPILAR EL PROGRAMA

14. Compilar el proyecto seleccionando*: Project -> Rebuild All*.

El objetivo de compilar el proyecto en este punto es para cuando se programe el registro, se habilite la ayuda de búsqueda de los campos dentro de los registros o estructuras.

MODIFICAR CONTENIDO DE CÓDIGO FUENTE

15. Añadir una nueva estructura "ECanaShadow" como una variable local en el “main”, como se observa en la Figura 4.144

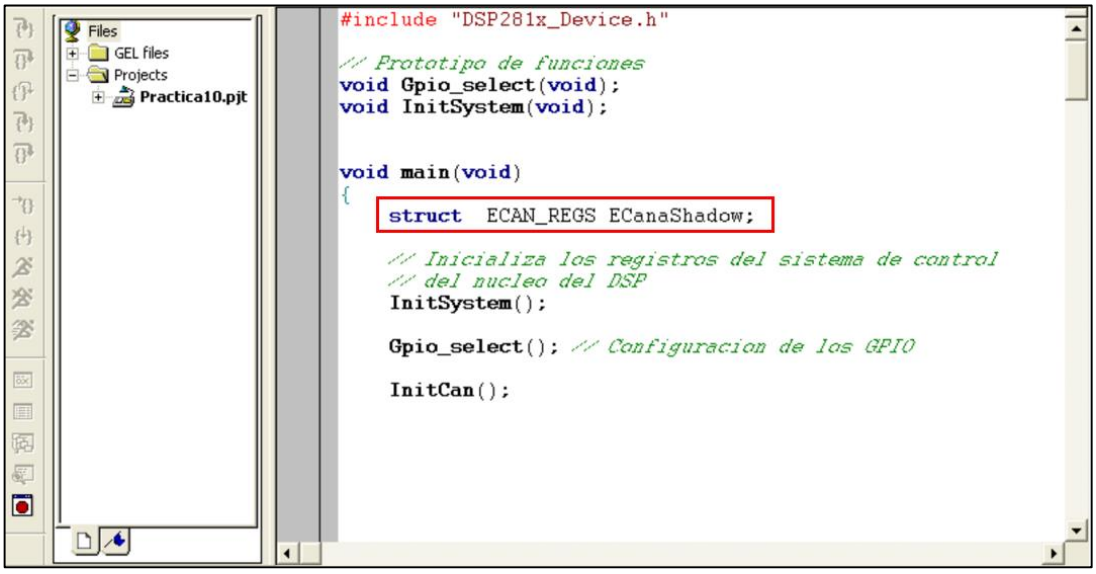


Figura 4.144 Declaración de variable local como estructura de registros CAN

Esta estructura será usada como una copia local de los registros CAN originales. Su creación es necesaria para manipular bits individuales en algunos registros que únicamente son accesibles mediante escritura de todos sus bits a la vez. La manipulación individual se la podrá realizar en la copia creada para luego cargarla a los registros originales CAN.

16. En la definición de la función “InitSystem( )” habilitar la unidad de reloj para el modulo CAN, como se observa en la Figura 4.145

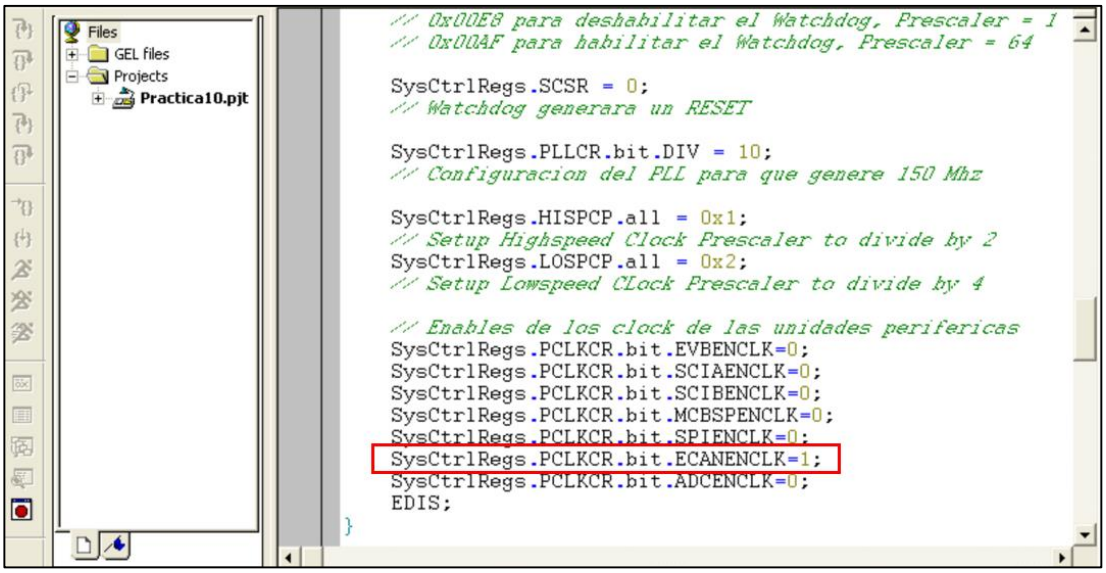


Figura 4.145 Habilitación de Reloj de unidad periférica de comunicación CAN

17. Dentro de la función “Gpio_select( )” seleccionar la función primaria de los bits 6 y 7 (CANTxA y CANRxA) del puerto F. Ver Figura 4.146

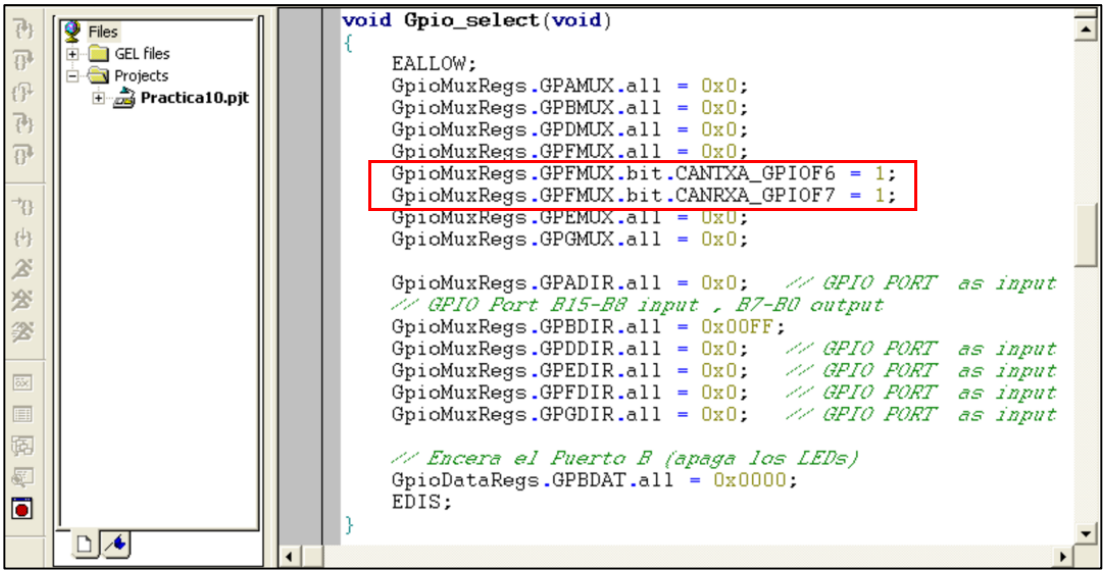


Figura 4.146 Configuración de función primaria en los pines GPIOF6 y GPIOF7

AGREGAR CÓDIGO DE INICIALIZACIÓN DE CAN

18. Declarar el prototipo ***"void InitCan(void)"*** al inicio del archivo código fuente y al final del mismo definirlo (Ver Figura 4.147) en base a lo siguiente:

Para el registro CANTIOC:

- Configurar TXFUNC para que el pin CANTxA sea usado para funciones de transmisión CAN

Para el registro CANRIOC:

- Configurar RXFUNC para que el pin CANRxA sea usado para funciones de recepción CAN.

Para el registro CANMC:

- Habilitar el modo HECC (High−end CAN Controller) del módulo CAN.
- Ajustar el bit CCR en 1 para permitir el acceso a la configuración del registro CANBTC.

Nota: Antes de poder continuar con la inicialización, nosotros debemos esperar a que el modulo CAN haya procesado la solicitud. En este caso el indicador CCE del registro CANES se ajustará en 1 cuando el modulo CAN esté listo. Mediante una línea de código, configurar una espera para que se establezca en uno este indicador y así iniciar la configuración del tiempo de bit.

Para el registro CANBTC:

- Configurar los parámetros BRP, TSEG1 y TSEG2 para recibir a 100 kbps a un punto de muestreo del 80% del tiempo de bit.

Nota: Para el registro CANMC ajustar el bit CCR en 0 para deshabilitar el acceso a la configuración del registro CANBTC, luego configurar una espera para que se encere la bandera del indicador CCE del registro CANES.

Para el registro CANME:

- Deshabilitar todos los mailboxes (ajustar en 0 todos los bits) para así poder permitir la escritura de los registros MSGID, en estos registros se almacenarán los identificadores de mensaje.

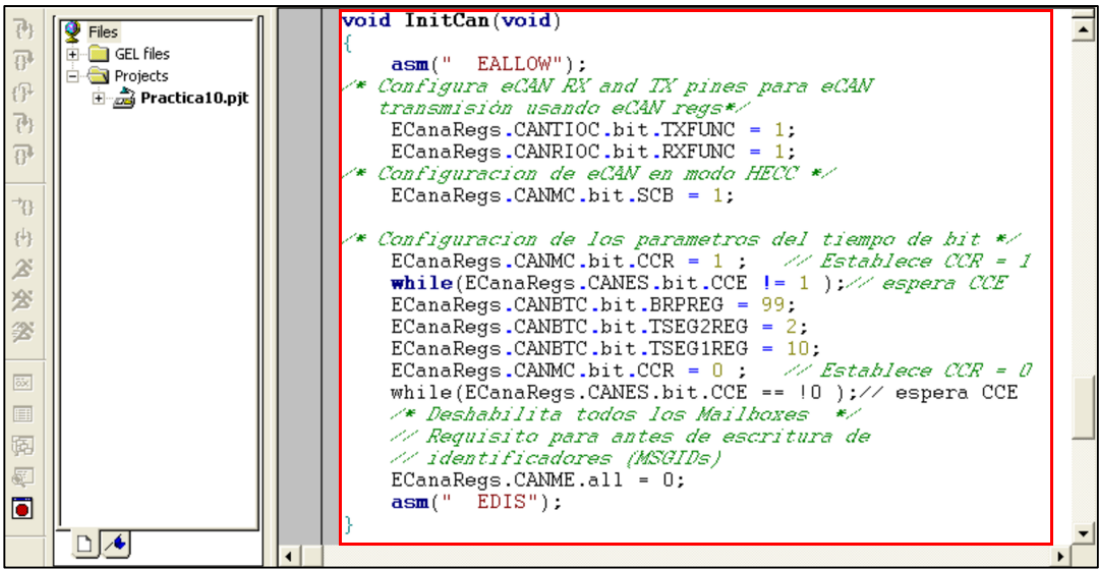


Figura 4.147 Definición de la función "InitCan( )"

DENTRO DEL MAIN, CONFIGURAR EL MAILBOX #1 PARA RECIBIR

19. Llamar a la función "InitCan( )" en el "main" antes del lazo "while(1)", como se observa en la Figura 4.148

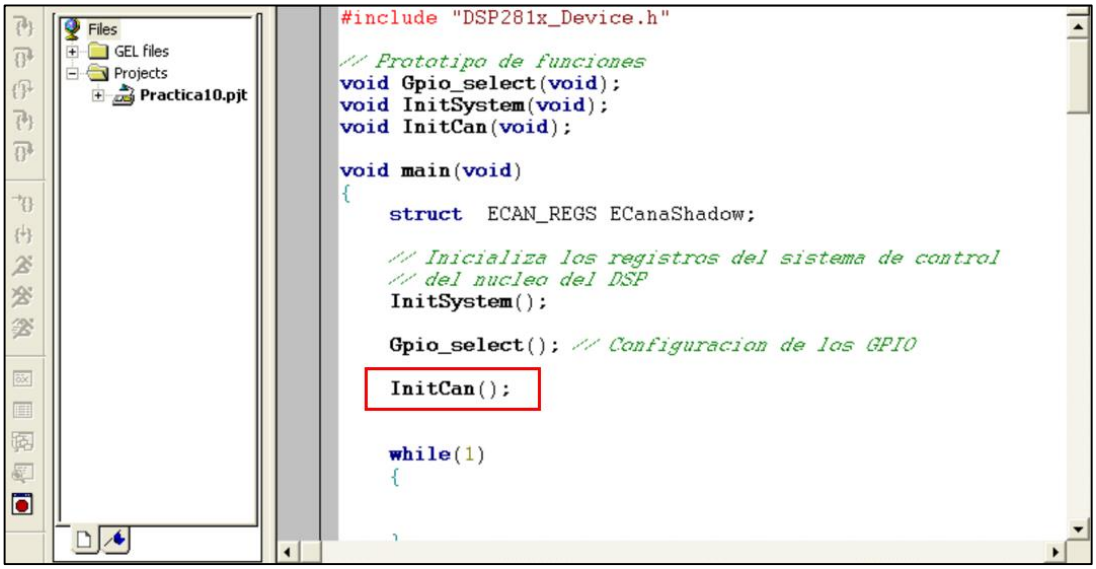


Figura 4.148 Llamado a la función "InitCan( )"

20. En el interior del "main", luego del llamado a la función "InitCan()", añadir código para preparar al Mailbox como receptor de la trama (Ver Figura 4.149). En este ejercicio, se utilizará el Mailbox #1, una identificación extendida de 0x10000000 y un tamaño de dato de 1 byte. Para ello:

- Escribir el identificador en el interior del registro MSGID.
- Establecer el bit "IDE" del registro MSGID para recibir con identificación extendida.
- Configurar el Mailbox #1 como Mailbox de recepción. Para ello se ajusta en 1 el bit MD1 del registro CANMD. Debido a la estructura interna de la unidad CAN, no se puede acceder a modificar un bit de los registros originales CAN, una buena práctica es copiar todos los valores de un registro hacia un "Shadow Register", manipularlo y copiar los 32 bits del "Shadow Register" con las modificaciones ajustadas al registro original.
- Habilitar el Mailbox #1, utilizar la ayuda de "Shadow Register".

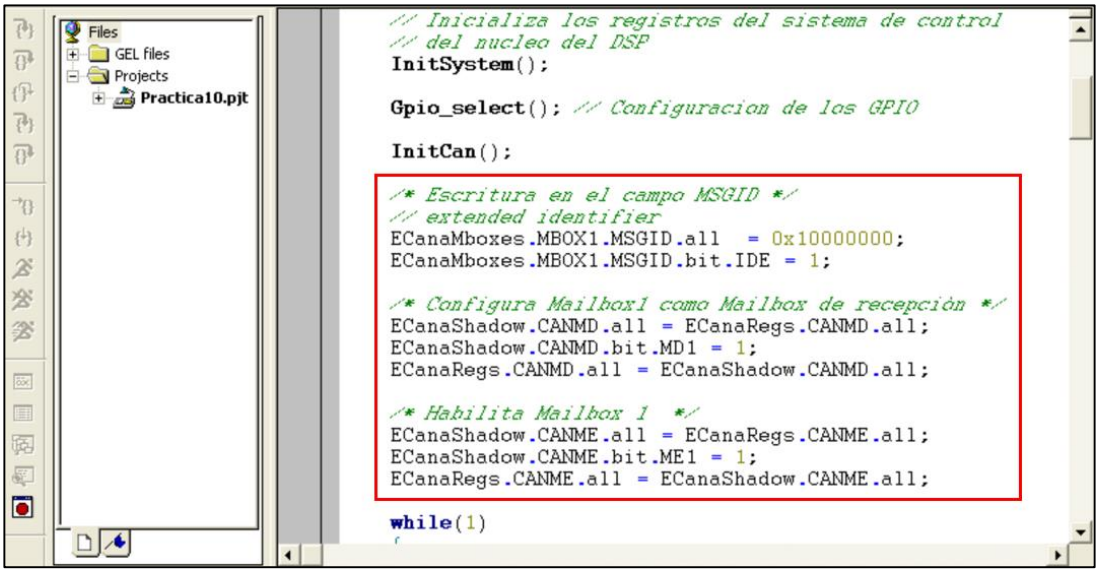


Figura 4.149 Configuración de Mailbox # 1 como Mailbox de recepción

DENTRO DEL MAIN, RECIBIR EL BYTE DE DATOS

21. Dentro del lazo “while(1)” del “main”:

- generar una espera mediante un lazo “do-while” hasta que el bit “RMP1” del registro CANRMP de la estructura ECanaRegs se establezca en 1, lo cual indicará que un mensaje válido ha sido recibido.

Nota: Es recomendable copiar los valores de los bits del registro CANRMP de la estructura ECanaRegs en el registro CANRMP de la estructura ECanaShadow para luego verificar el bit “ECanaShadow.CANRMP.bit.RMP1”.

- Si el bit RMP1 es establecido en 1 por la unidad CAN, se tomará el byte 0 de datos del Mailbox #1 para mostrarlo en los 8 leds conectados a GPIO B7...B0.
- Limpiar la bandera “ECanaRegs.CANRMP.bit.RMP1” (asignando 1 al bit), para ello utilizar la ayuda de “Shadow Register”.

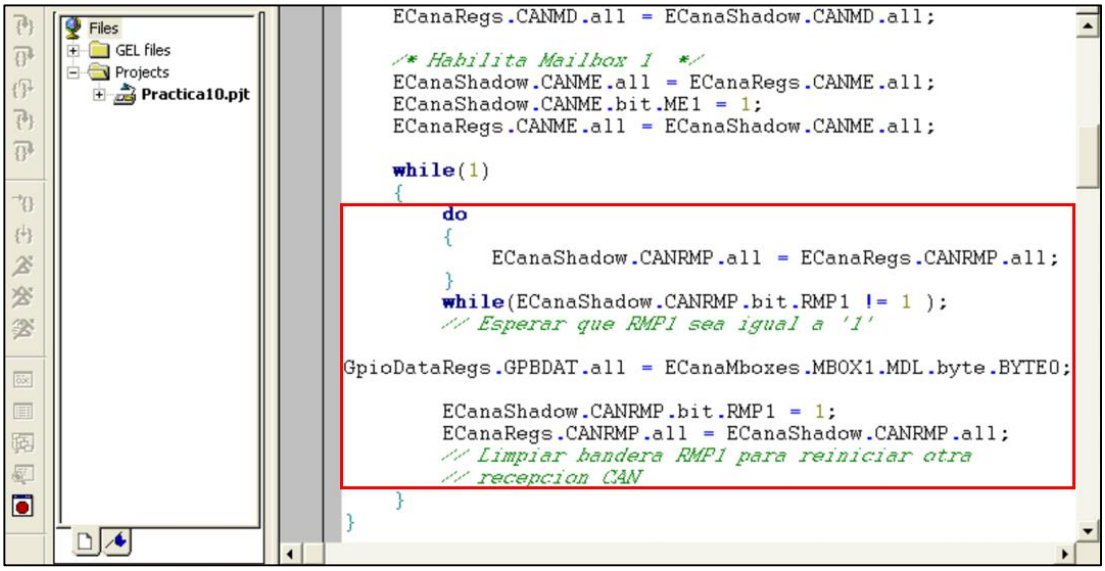


Figura 4.150 Recepción de trama de datos CAN y carga en los leds

COMPILAR, CARGAR Y EJECUTAR EL PROGRAMA

22. Colocar los jumpers JP5 y JP6 de la tarjeta *KSPS-0504 Adaptor board* en la posición 1-2 con el objetivo de seleccionar el integrado SN65HVD230 que generará el bus CAN.

23. Colocar el jumper JP4 de la tarjeta *KSPS-0504 Adaptor board* en la posición 2-3 con el objetivo de habilitar la resistencia de terminación de 120 [ohms]

Para comprobar la recepción de una trama CAN, se debe emplear un segundo kit de prácticas con el procesador *TMS320F2812*, el cual debe estar programado para transmitir una trama CAN (Práctica #9).

24. Enlazar ambas tarjetas, conectando punto a punto los pines 2, 7 y 3 de los conectores X2 (DE9 macho) de ambas tarjetas *KSPS-0504 Adaptor board.*

25. Compilar el proyecto seleccionando*: Project -> Rebuild All*. Solucionar los errores en caso de haberlos.

26. Alimentar la tarjeta *eZdsp™F2812*.

27. Conectarse con el procesador: *Debug -> Connect*

28. Cargar el programa (*File -> Load Program…)* **Practica10.out** que se genera dentro de la carpeta Debug al compilar el proyecto

29. Ejecutar el programa cargado: *Debug -> Run*

30. Desconectarse del procesador: *Debug -> Disconnect*

31. Comprobar el funcionamiento del programa:

En este literal ya se puede comprobar la práctica 9, con ayuda de la práctica 10.

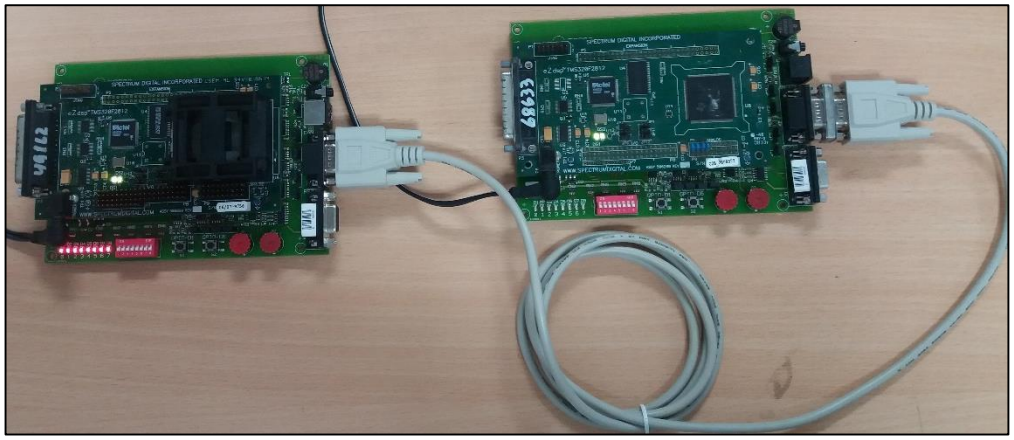


Figura 4.151 Ejecución de la práctica sobre comunicación CAN

En la Figura 4.151 se presenta al lado izquierdo el primer Kit DSP (Práctica #9), el cual ha sido configurado para transmitir el estado de los switches hacia el segundo Kit DSP (Practica # 10) que se encuentra al lado derecho y el cual ha sido configurado para recibir la trama de datos y así mostrarla en sus respectivos leds.

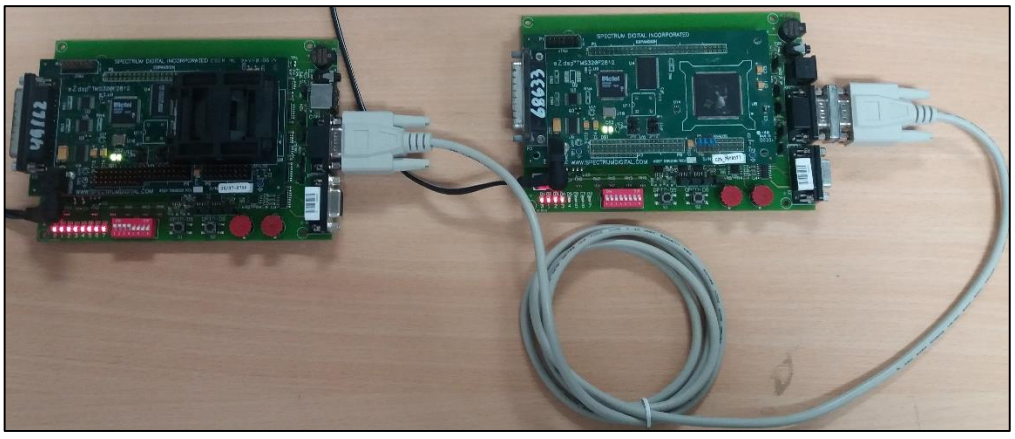


Figura 4.152 Demostración de la práctica sobre comunicación CAN

En la Figura 4.152 se presenta una pequeña demostración, en la cual se cambian el estado de los cuatro primeros switches (Figura 4.153) y se transmite esta información. En el DSP de recepción, recibe la información y la muestra en sus respectivos leds (Figura 4.154).

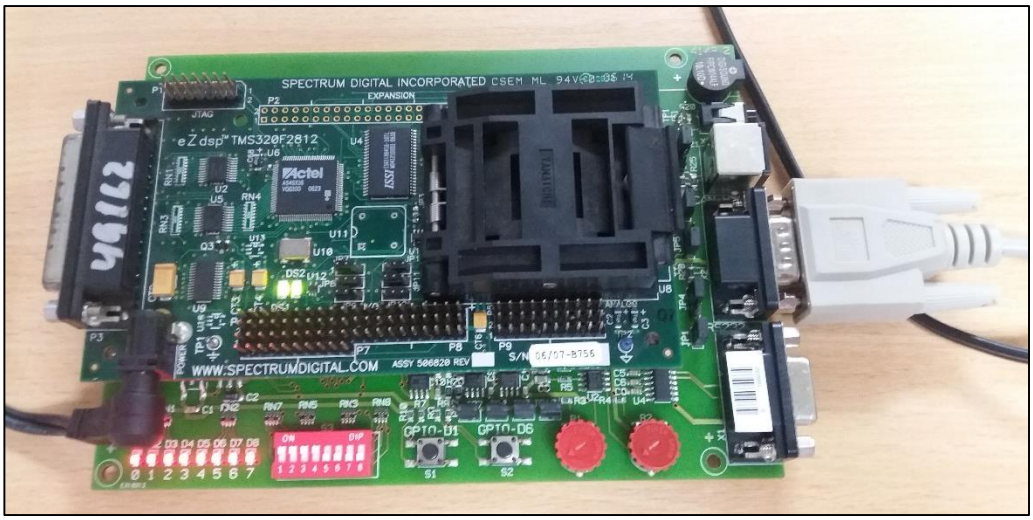


Figura 4.153 Cambio de estado de los 4 primeros switches en el primer Kit DSP

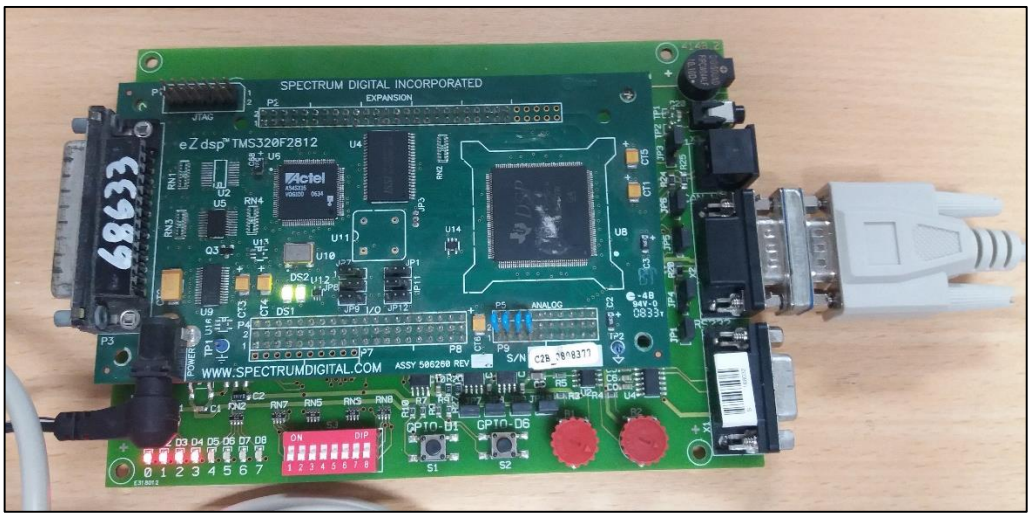


Figura 4.154 Visualización de la información recibida en el segundo Kit DSP

32. Al finalizar la práctica, conectarse con el procesador: *Debug -> Connect*

33. Detener la ejecución del programa: *(Debug -> Halt)*

34. Resetear el CPU *(Debug -> Reset CPU)*, luego desconectarse de la tarjeta *(Debug -> Disconnect).* Y quitar la alimentación de la tarjeta *eZdsp™F2812*.

ANEXO 2. FUNDAMENTACIÓN TEÓRICA DE LAS PRÁCTICAS

CONFIGURACIONES DEL CONTROL DE SISTEMA

MODULO DE RELOJ

Lo primero que se debe configurar en los DSPs es el módulo de reloj. El *TMS320F2812* es temporizado por un oscilador externo por los pines *X1* y *X2*, este oscilador es más lento para así reducir el ruido electromagnético. Entonces un circuito interno PLL es el que generara la velocidad interna a partir del oscilador externo. [1]

La tarjeta *eZdsp™F2812* contiene un cristal, el cual proporciona una frecuencia de reloj externo de 30 MHz, entonces para lograr alcanzar la frecuencia interna de 150 MHz que necesita el núcleo (CPU) del DSP hay que multiplicar el reloj externo por 10 y dividirlo para 2. El factor de multiplicación se puede programar en el circuito interno PLL.

Diagrama de Bloques del Módulo de Reloj

En la Figura 5.1 se puede observar el diagrama de bloques del módulo de reloj del procesador *TMS320F2812*

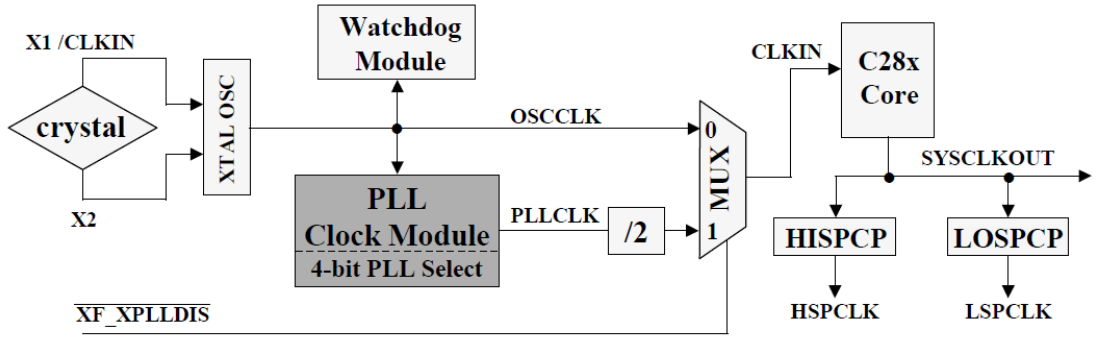


Figura 5.1 Diagrama de Bloques del módulo de reloj [1]

El Modulo del perro guardián (Watchdog Timer) tendrá la misma frecuencia que el oscilador externo. Como la tarjeta *eZdsp™F2812* contiene un cristal de 30 MHz, ese será el valor de frecuencia de la señal “OSCCLK”.

El PLL toma la frecuencia de oscilación externa y la multiplica por un valor programable en el registro PLLCR. La frecuencia resultante “PLLCLK” luego es dividida para dos para entrar al MUX.

El pin “XF_XPLLDIS” controlara la salida del MUX. Entonces si en el pin se ingresa una señal de bajo, la frecuencia de “CLKIN” será igual a la frecuencia de salida del PLL. Por otro lado, si en el pin se ingresa una señal de alto, la frecuencia de “CLKIN” será igual a la frecuencia del oscilador externo (cristal).

La señal “CLKIN” a la entrada del núcleo (CPU) del DSP tiene la misma frecuencia que la señal “SYSCLKOUT” que está a la salida del mismo. [1]

Los registros HISPCP (High-speed Clock Pre-scaler) y LOSPCP (Low speed Clock Pre-scaler) son usados como unos adicionales divisores de reloj. Las salidas de estos pre-escaladores son usadas como fuentes de reloj para las unidades periféricas. [1].

Para usar una unidad periférica, se debe habilitar la distribución de reloj en el campo individual de los bits del registro PCLKCR. Las entradas y salidas digitales no tienen esta característica, aunque también es una unidad periférica. [1]

WATCHDOG TIMER

El “Watchdog timer” es un contador independiente del CPU que desencadena un reseteo del procesador, si este llega a desbordarse. Si se desea evitar el reseteo del procesador, se debe encerar periódicamente el contador. El módulo de perro guardián (Watchdog Timer) es usado para reconocer eventos de fallo en el programa, por ejemplo, si el programa se queda inhibido en alguna línea de código. [1]

Diagrama de Bloques del Módulo Watchdog Timer

En la Figura 5.2 se puede observar el diagrama de bloques del módulo de perro guardián (Watchdog Timer) del procesador *TMS320F2812.*

El Pre escalador del Watchdog (“WDPS2-0” del registro WDCR) puede ser usado para aumentar el periodo de desbordamiento del Watchdog por medio de un contador de 6 bits. El bit “WDDIS” permite el conteo del Watchdog.

Los bits para lógica de verificación (“WDCHK2-0” del registro WDCR) es un campo de bits de seguridad. Todas las configuraciones que se hagan en el registro WDCR debe incluir la combinación “101” en este campo de 3 bits, de lo contrario se genera un RESET inmediatamente. [1]

La bandera “WDFLAG” del registro WDCR puede ser utilizado para distinguir entre un Reset normal (WDFLAG=0) y un Reset debido al Watchdog (WDFLAG=1). Para limpiar esta bandera por software, se debe escribir un “1” en este bit. [1]

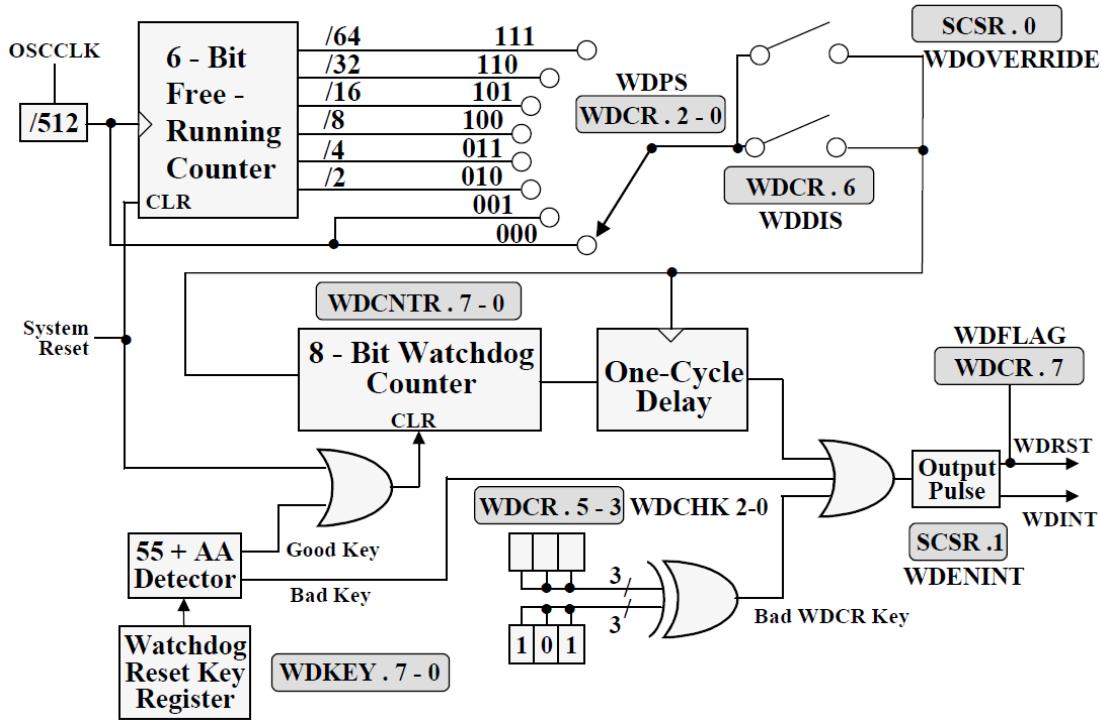


Figura 5.2 Diagrama de Bloques del módulo Watchdog Timer [1]

Para encerar el contador del Watchdog antes de que desencadene un Reseteo del sistema, se debe escribir la secuencia (55h luego AAh) en los 8 bits menos significativos del registro WDKEY.

REGISTROS DEL MÓDULO DE RELOJ Y WATCHDOG TIMER

La Tabla 14 presenta los registros del Módulo de Reloj y Watchdog Timer que se usaran en las prácticas.

Dirección	Registro	Nombre
0x007021	PLLCR	Registro de control PLL
0x00701A	HISPCP	Registro para reloj “HSPCLK”
0x00701B	LOSPCP	Registro para reloj “LSPCLK”
0x00701C	PCLKCR	Registro de control del reloj en unidades periféricas
0x007029	WDCR	Registro de control del Watchdog Timer
0x007023	WDCNTR	Contador del Watchdog Timer
0x007025	WDKEY	Reseteo de Watchdog Timer
0x007022	SCSR	Registro de Sistema de Control y Estados

Tabla 14: Registros del Módulo de reloj y Watchdog Timer [5]

Todos estos registros son parte de la estructura “SysCtrlRegs”. Si se necesita usar alguno de estos registros de la Tabla 14 se deberá primero llamar a la estructura “SysCtrlRegs”, para luego acceder al campo con el nombre del registro a usar. Por ejemplo, si se requiere cargar el valor de 0x0001 en el registro HISPCP, se lo hace de la siguiente manera: “SysCtrlRegs.HISPCP.all = 0x0001”.

Registro de Control PLL (PLLCR)

El registro PLLCR permite programar el factor por el cual se multiplicará el valor de la frecuencia del oscilador externo. La configuración en este registro está protegida por el registro EALLOW. Por lo que, si se requiere cargar el valor de 0x0001 en el registro PLLCR, se lo hace de la siguiente manera en lenguaje C:

- EALLOW; // Permite escribir en el registro
- SysCtrlRegs.PLLCR.all = 0x0001; // Escritura en el registro
- EDIS; // Deshabilita escritura en el registro

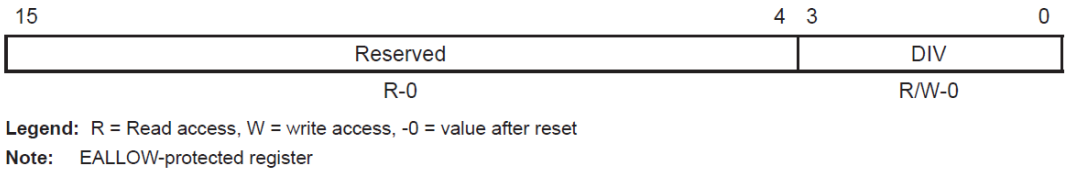


Figura 5.3 Registro de Control PLL (PLLCR) [5]

En la Tabla 15 se presenta la descripción de los bits mostrados en la Figura 5.3

Bit(s)	Nombre	Descripción
15-4	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto
3-0	DIV	0000 CLKIN = OSCCLK/2 (PLL Bypass) 0001 CLKIN = (OSCCLK*1)/2 0010 CLKIN = (OSCCLK*2)/2 0011 CLKIN = (OSCCLK*3)/2 0100 CLKIN = (OSCCLK*4)/2 0101 CLKIN = (OSCCLK*5)/2 0110 CLKIN = (OSCCLK*6)/2 0111 CLKIN = (OSCCLK*7)/2 1000 CLKIN = (OSCCLK*8)/2 1001 CLKIN = (OSCCLK*9)/2 1010 CLKIN = (OSCCLK*10)/2

Tabla 15: Descripción de los bits del registro PLLCR [5]

Registro para reloj “HSPCLK” (HISPCP)

Permite configurar la frecuencia del reloj adicional HSPCLK. La configuración en este registro está protegida por el registro EALLOW.

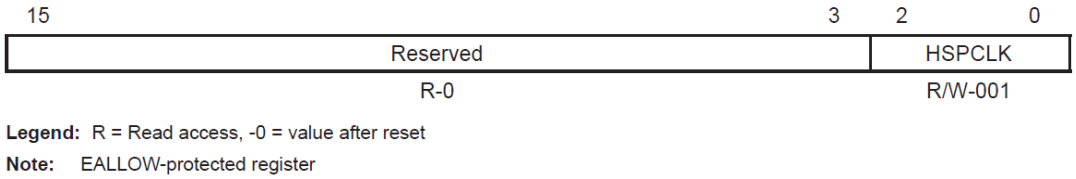


Figura 5.4 Registro para reloj "HSPCLK" (HISPCP) [5]

En la Tabla 16 se presenta la descripción de los bits mostrados en la Figura 5.4

Bit(s)	Nombre	Descripción
15-3	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto
2-0	HSPCLK	Estos bits permiten configurar la frecuencia del reloj HSPCLK 000 HSPCLK = SYSCLKOUT/1 001 HSPCLK = SYSCLKOUT/2 (reset default) 010 HSPCLK = SYSCLKOUT/4 011 HSPCLK = SYSCLKOUT/6 100 HSPCLK = SYSCLKOUT/8 101 HSPCLK = SYSCLKOUT/10 110 HSPCLK = SYSCLKOUT/12 111 HSPCLK = SYSCLKOUT/14

Tabla 16: Descripción de los bits del registro HISPCP [5]

Registro para reloj “LSPCLK” (LOSPCP)

Permite configurar la frecuencia del reloj adicional LSPCLK. La configuración en este registro está protegida por el registro EALLOW.

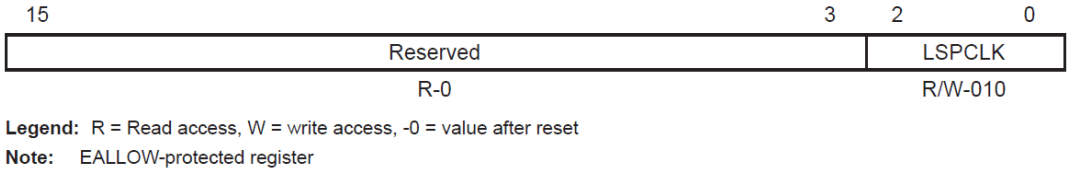


Figura 5.5 Registro para reloj "LSPCLK" (LOSPCP) [5]

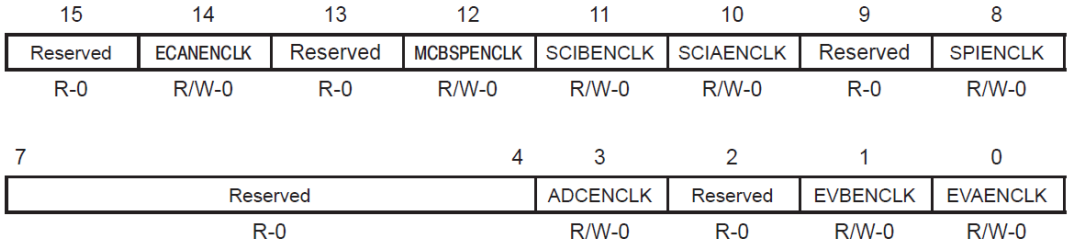
En la Tabla 17 se presenta la descripción de los bits mostrados en la Figura 5.5

Bit(s)	Nombre	Descripción
15-3	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto
2-0	LSPCLK	Estos bits permiten configurar la frecuencia del reloj LSPCLK 000 HSPCLK = SYSCLKOUT/1 001 HSPCLK = SYSCLKOUT/2 010 HSPCLK = SYSCLKOUT/4 (reset default) 011 HSPCLK = SYSCLKOUT/6 100 HSPCLK = SYSCLKOUT/8 101 HSPCLK = SYSCLKOUT/10 110 HSPCLK = SYSCLKOUT/12 111 HSPCLK = SYSCLKOUT/14

Tabla 17: Descripción de los bits del registro LOSPCP [5]

Registro de control del reloj en unidades periféricas (PCLKCR)

Este registro permite habilitar o deshabilitar los relojes en las unidades periféricas. La configuración en este registro está protegida por el registro EALLOW.



Legend: R = Read access, -0 = value after reset

Notes: 1) EALLOW-protected register

2) If a peripheral block is not used, then the clock to that peripheral can be turned off to minimize power consumption.

Figura 5.6 Registro de control del reloj en unidades periféricas (PCLKCR) [5]

En la Tabla 18 se presenta la descripción de los bits mostrados en la Figura 5.6

Bit(s)	Nombre	Descripción
15	Reservado	
14	ECANENCLK	Unidad periférica funciona con reloj del sistema “SYSCLKOUT” 0 Deshabilita Reloj en unidad periférica 1 Habilita Reloj en unidad periférica
13	Reservado	
12	MCBSPENCLK	Unidad periférica funciona con reloj “LSPCLK” 0 Deshabilita Reloj en unidad periférica 1 Habilita Reloj en unidad periférica
11	SCIBENCLK	Unidad periférica funciona con reloj “LSPCLK” 0 Deshabilita Reloj en unidad periférica 1 Habilita Reloj en unidad periférica
10	SCIAENCLK	Unidad periférica funciona con reloj “LSPCLK” 0 Deshabilita Reloj en unidad periférica 1 Habilita Reloj en unidad periférica
9	Reservado	
8	SPIAENCLK	Unidad periférica funciona con reloj “LSPCLK” 0 Deshabilita Reloj en unidad periférica 1 Habilita Reloj en unidad periférica
7-4	Reservado	
3	ADCENCLK	Unidad periférica funciona con reloj “HSPCLK” 0 Deshabilita Reloj en unidad periférica 1 Habilita Reloj en unidad periférica

Bit(s)	Nombre	Descripción
2	Reservado	
1	EVBENCLK	Unidad periférica funciona con reloj “HSPCLK” 0 Deshabilita Reloj en unidad periférica 1 Habilita Reloj en unidad periférica
0	EVAENCLK	Unidad periférica funciona con reloj “HSPCLK” 0 Deshabilita Reloj en unidad periférica 1 Habilita Reloj en unidad periférica

Tabla 18: Descripción de los bits del registro PCLKCR [5]

Registro de control del Watchdog Timer (WDCR)

Este registro permite configurar el modulo del perro guardián (Watchdog Timer). La configuración en este registro está protegida por el registro EALLOW.

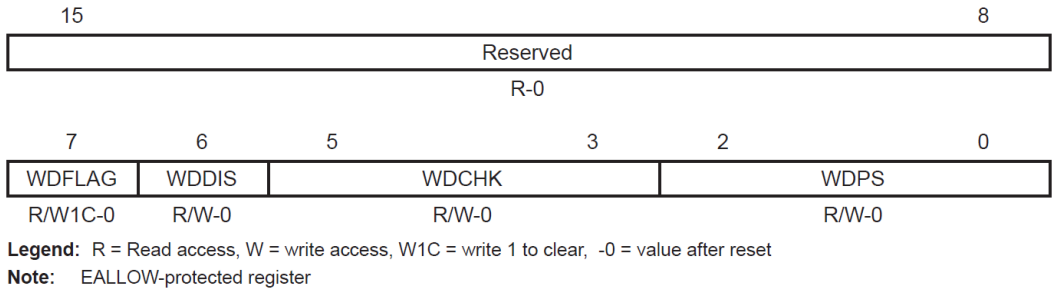


Figura 5.7 Registro de control del Watchdog Timer (WDCR) [5]

En la Tabla 19 se presenta la descripción de los bits mostrados en la Figura 5.7

Bit(s)	Nombre	Descripción
15-8	Reservado	
7	WDFLAG	0 No tiene efecto 1 Limpia este bit para que el WD cause un reset
6	WDDIS	0 Habilita el módulo Watchdog 1 Deshabilita el módulo Watchdog
5-3	WDCHK(2-0)	101 No tiene efecto Otro valor Desencadena un Reset en el procesador

Bit(s)	Nombre	Descripción
2-0	WDPS(2-0)	Estos bits permiten configurar el reloj del contador del Watchdog 000 WDCLK = (OSCCLK/512) /1 001 WDCLK = (OSCCLK/512) /1 010 WDCLK = (OSCCLK/512) /2 011 WDCLK = (OSCCLK/512) /4 100 WDCLK = (OSCCLK/512) /8 101 WDCLK = (OSCCLK/512) /16 110 WDCLK = (OSCCLK/512) /32 111 WDCLK = (OSCCLK/512) /64

Tabla 19: Descripción de los bits del registro WDCR [5]

Contador del Watchdog Timer (WDCNTR)

La configuración en este registro está protegida por el registro EALLOW.

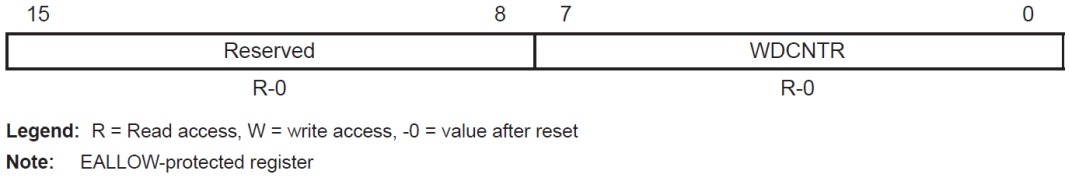


Figura 5.8 Contador del Watchdog Timer (WDCNTR) [5]

En la Tabla 20 se presenta la descripción de los bits mostrados en la Figura 5.8

Bit(s)	Nombre	Descripción
15-8	Reservado	
7-0	WDCNTR	Estos bits contienen el valor actual del Watchdog timer. Si el contador se desborda entonces se generará un reset en el procesador. Si el registro WDKEY es escrito con una valida combinación entonces el contador se encera.

Tabla 20: Descripción de los bits del registro WDCNTR [5]

Reseteo de Watchdog Timer (WDKEY)

La configuración en este registro está protegida por el registro EALLOW.

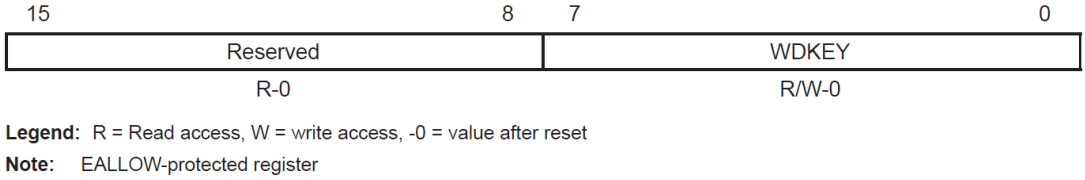


Figura 5.9 Reseteo de Watchdog Timer (WDKEY) [5]

En la Tabla 21 se presenta la descripción de los bits mostrados en la Figura 5.9

Bit(s)	Nombre	Descripción
15-8	Reservado	
7-0	WDCNTR	Escribiendo la secuencia 0x55 seguido de 0xAA ocaciona que el contador WDCNTR se encere. Escribiendo cualquier otro valor causa un reset en el procesador.

Tabla 21: Descripción de los bits del registro WDKEY [5]

Registro de Sistema de Control y Estados (SCSR)

La configuración en este registro está protegida por el registro EALLOW.

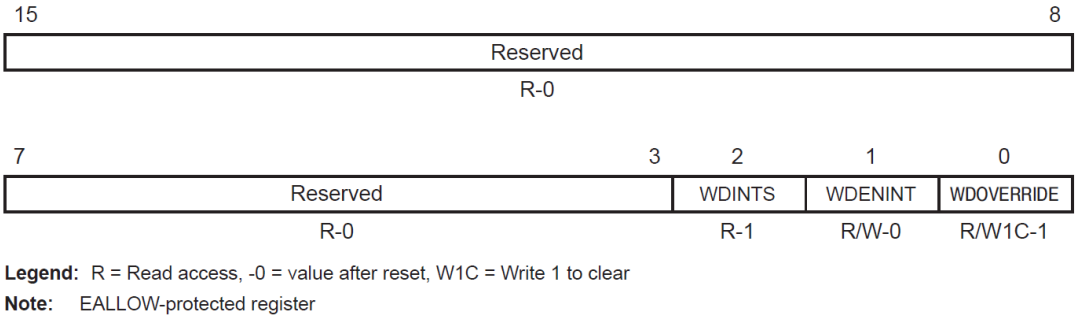


Figura 5.10 Registro de Sistema de Control y Estado (SCSR) [5]

En la Tabla 22 se presenta la descripción de los bits mostrados en la Figura 5.10

Bit(s)	Nombre	Descripción
15-3	Reservado	
2	WDINTS	Bit de estado de interrupción por el Watchdog timer 0 Ha ocurrido una interrupción 1 No ha ocurrido una interrupción
1	WDENINT	0 Deshabilita Interrupción por el Watchdog y habilita el reset del procesador por el Watchdog 1 Habilita Interrupción por el Watchdog y deshabilita el Reset del procesador por el Watchdog
0	WDOVERRIDE	0 No tiene efecto 1 Permite la escritura en el bit WDDIS del registro WDCR. (Default)

Tabla 22: Descripción de los bits del registro SCSR [5]

MANEJO DE ENTRADAS Y SALIDAS DIGITALES

En el Procesador Digital de Señales (DSP) no solo tenemos un núcleo de 32-bit, sino también algunas unidades periféricas necesarias para construir un sistema de control que pueda resolver varios problemas en distintos tipos de aplicaciones. En este anexo se explicará cómo configurar y usar la unidad periférica de entradas y salidas digitales.

ASIGNACIÓN DE PINES

Todas las entradas y salidas digitales son agrupadas en puertos, llamados GPIO-A, GPIO-B, GPIO-D, GPIO-E, GPIO-F, GPIO-G. GPIO significa “entrada y salida de propósito general”. Los pines físicos del procesador *TMS320F2812* pueden ser usado para 2 diferentes funciones, en algunos pines hasta 3 funciones. [1] El programador puede seleccionar la función con la que trabajará en el registro GPxMUX (donde x puede ser A, B, D, E, F o G).

En las Tablas de 23 y 24 se muestra la función I/O y la función primaria de los pines del procesador *TMS320F2812* agrupados por puertos.

GPIO-A	GPIO-B	GPIO-D
GPIOA0 / PWM1	GPIOB0 / PWM7	GPIOD0 / T1CTRIP_PDPINTA
GPIOA1 / PWM2	GPIOB1 / PWM8	GPIOD1 / T2CTRIP / EVASOC
GPIOA2 / PWM3	GPIOB2 / PWM9	GPIOD5 / T3CTRIP_PDPINTB
GPIOA3 / PWM4	GPIOB3 / PWM10	GPIOD6 / T4CTRIP / EVBSOC
GPIOA4 / PWM5	GPIOB4 / PWM11	
GPIOA5 / PWM6	GPIOB5 / PWM12	
GPIOA6 / T1PWM_T1CMP	GPIOB6 / T3PWM_T3CMP	
GPIOA7 / T2PWM_T2CMP	GPIOB7 / T4PWM_T4CMP	
GPIOA8 / CAP1_QEP1	GPIOB8 / CAP4_QEP3	
GPIOA9 / CAP2_QEP2	GPIOB9 / CAP5_QEP4	
GPIOA10 / CAP3_QEP11	GPIOB10 / CAP6_QEP12	
GPIOA11 / TDIRA	GPIOB11 / TDIRB	
GPIOA12 / TCLKINA	GPIOB12 / TCLKINB	
GPIOA13 / C1TRIP	GPIOB13 / C4TRIP	
GPIOA14 / C2TRIP	GPIOB14 / C5TRIP	
GPIOA15 / C3TRIP	GPIOB15 / C6TRIP	

Tabla 23: Asignación de pines de los puertos A, B y D [1]

GPIO-E	GPIO-F	GPIO-G
GPIOE0 / XINT1_XBIO	GPIOF0 / SPISIMOA	GPIOG4 / SCITXDB
GPIOE1 / XINT2_ADCSOC	GPIOF1 / SPISOMIA	GPIOG5 / SCIRXDB
GPIOE2 / XNMI_XINT13	GPIOF2 / SPICLKA	
	GPIOF3 / SPISTEA	
	GPIOF4 / SCITXDA	
	GPIOF5 / SCIRXDA	
	GPIOF6 / CANTXA	
	GPIOF7 / CANRXA	
	GPIOF8 / MCLKXA	
	GPIOF9 / MCLKRA	
	GPIOF10 / MFSXA	
	GPIOF11 / MFSRA	
	GPIOF12 / MDXA	
	GPIOF13 / MDRA	
	GPIOF14 / XF	

Tabla 24: Asignación de pines en los puertos E, F y G [1]

Los pines del procesador *TMS320F2812* se conectarán a los distintos elementos de la tarjeta *KSPS-0504 Adaptor Board* a través de los espadines de la tarjeta *eZdspF2812*. En la Figura 5.11 se muestra la tarjeta *KSPS-0504 Adaptor Board* con los elementos que se usarán en las prácticas.

- Los 8 switches estarán conectados a los pines GPIOB15 a B8
- Los 8 LEDs estarán conectados a los pines GPIOB7 a B0
- Los 2 pulsadores estarán conectados a los pines GPIOD1 y GPIOD6
- EEPROM serial (M95080) conectada a los pines GPIOD6, GPIOF0, F1, F2 y F3.
- DAC serial (TLV5617A) conectado a los pines GPIOD0, GPIOF0, F1, F2 y F3.
- Conector DE-9 hembra (SCI-A) conectado a los pines GPIOF4 y GPIOF5.
- Conector DE-9 macho (CAN2.0B) conectado al chip SN65HVD230, y el chip conectado a los pines GPIOF6 y GPIOF7.

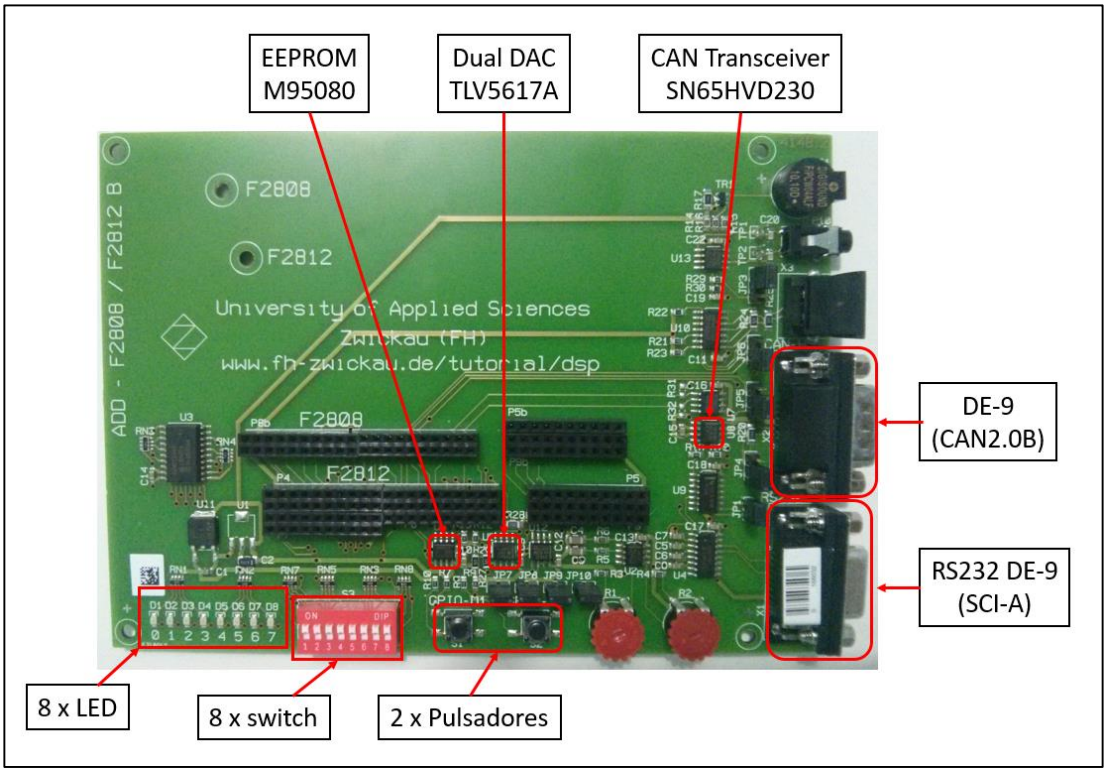


Figura 5.11 Elementos involucrados en las prácticas

DIAGRAMA DE BLOQUES

En la Figura 5.12 se puede observar el diagrama de bloques para realizar la configuración de cada pin del procesador *TMS320F2812*.

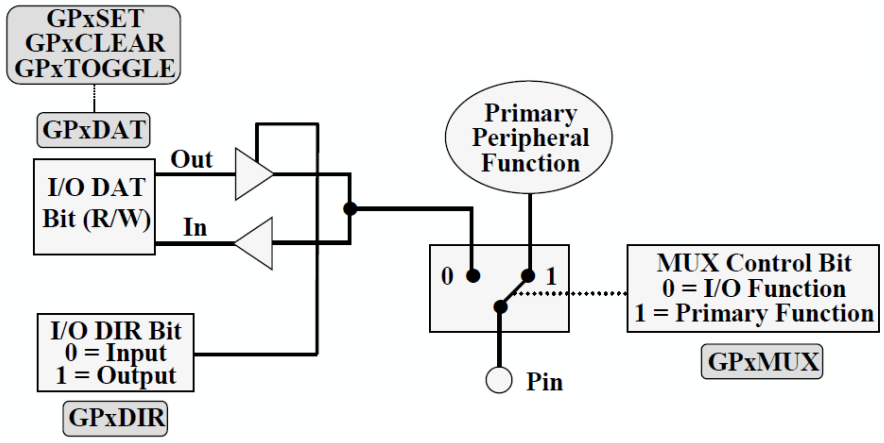


Figura 5.12 Diagrama de bloques para configuración de un pin [1]

Todos los seis puertos son controlados por su propio registro multiplexor, llamado GPxMUX (donde x puede ser A, B, D, E, F o G) [1]. Asignando el valor de '0' a un bit del registro GPxMUX significará la selección de la función I/O para el pin físico, por otro lado, si se asigna el valor de '1' a un bit del registro GPxMUX significará la selección de la función primaria para el pin físico.

Cuando la función I/O ha sido seleccionada, entonces el grupo de registros GPxDIR (donde x puede ser A, B, D, E, F o G) determinará si el pin será utilizado como entrada o salida [1]. Asignado el valor de 0 a un bit del registro GPxDIR significará que el pin será utilizado como entrada, por otro lado, si se asigna el valor de 1 a un bit del registro GPxDIR significará que el pin será utilizado como salida.

REGISTROS DE LA ESTRUCTURA "GpioMuxRegs"

La Tabla 25 presenta los registros de la estructura "GpioMuxRegs" que se usaran en las prácticas para la configuración de los pines del procesador *TMS320F2812*.

Dirección	Registro	Nombre
0x0070C0	GPAMUX	Registro de Control GPIO-A MUX
0x0070C1	GPADIR	Registro de Control de Dirección GPIO-A
0x0070C4	GPBMUX	Registro de Control GPIO-B MUX
0x0070C5	GPBDIR	Registro de Control de Dirección GPIO-B
0x0070CC	GPDMUX	Registro de Control GPIO-D MUX
0x0070CD	GPDDIR	Registro de Control de Dirección GPIO-D
0x0070D0	GPEMUX	Registro de Control GPIO-E MUX
0x0070D1	GPEDIR	Registro de Control de Dirección GPIO-E
0x0070D4	GPFMUX	Registro de Control GPIO-F MUX
0x0070D5	GPFDIR	Registro de Control de Dirección GPIO-F
0x0070D8	GPGMUX	Registro de Control GPIO-G MUX
0x0070D9	GPGDIR	Registro de Control de Dirección GPIO-G

Tabla 25: Registros de la estructura "GpioMuxRegs" [5]

Todos estos registros son parte de la estructura "GpioMuxRegs". Si se necesita usar alguno de estos registros de la Tabla 25 se deberá primero llamar a la estructura "GpioMuxRegs", para luego acceder al campo con el nombre del registro a usar. Por

ejemplo, si se requiere cargar el valor de 0x00FF en el registro GPBDIR, se lo hace de la siguiente manera: “GpioMuxRegs.GPBDIR.all = 0x00FF”.

La configuración de estos registros está protegida por el registro EALLOW. Por lo que, si se requiere cargar algún valor de en los registros de esta estructura, se lo hará de la siguiente manera en lenguaje C:

- EALLOW; // Permite escribir en el registro
- GpioMuxRegs.GPBDIR.all = 0x00FF; // Escritura en el registro
- EDIS; // Deshabilita escritura en el registro

Registros de Control GPxMUX

Cada puerto I/O tiene un registro multiplexor. Los registros multiplexores son usados para seleccionar entre la función primaria y la función I/O de cada uno de los pines del procesador *TMS320F2812* [5]. Después de un Reseteo del procesador todos los pines son configurados como pines de entrada o salida.

- Si GPxMUX.bit = 0, entonces el pin es configurado como una entrada o salida.
- Si GPxMUX.bit = 1, entonces el pin es configurado como la función primaria

La configuración en este registro está protegida por el registro EALLOW.

Registros de Control de dirección GPxDIR

Cada puerto I/O tiene un registro de control de dirección. Los registros GPxDIR son usados para seleccionar el estado que tendrá el pin del procesador *TMS320F2812* [5], con respecto al estado del pin se refiere a que puede ser configurado como una entrada o como una salida. Después de un Reseteo del procesador todos los pines son configurados como pines de entrada.

- Si GPxDIR.bit = 0, entonces el pin es configurado como una entrada.
- Si GPxDIR.bit = 1, entonces el pin es configurado como una salida.

La configuración en este registro está protegida por el registro EALLOW.

REGISTROS DE LA ESTRUCTURA “GpioDataRegs”

La Tabla 26 presenta los registros de la estructura “GpioDataRegs” que se usaran en las prácticas para el manejo de las entradas y salidas del procesador *TMS320F2812*.

Dirección	Registro	Nombre
0x0070E0	GPADAT	Registro de Datos GPIO-A
0x0070E4	GPBDAT	Registro de Datos GPIO-B
0x0070EC	GPDDAT	Registro de Datos GPIO-D
0x0070F0	GPEDAT	Registro de Datos GPIO-E
0x0070F4	GPFDAT	Registro de Datos GPIO-F
0x0070F8	GPGDAT	Registro de Datos GPIO-G

Tabla 26: Registros de la estructura "GpioDataRegs" [5]

Todos estos registros son parte de la estructura “GpioDataRegs”. Si se necesita usar alguno de estos registros de la Tabla 26 se deberá primero llamar a la estructura “GpioDataRegs”, para luego acceder al campo con el nombre del registro a usar. Por ejemplo, si se requiere cargar el valor de 0x00FF en el registro GPBDAT, se lo hace de la siguiente manera: “GpioDataRegs.GPBDAT.all = 0x00FF”.

Registros de Datos GPxDAT

Cada puerto I/O tiene un registro de datos. Los registros de datos es un registro de lectura y escritura que refleja el estado actual de los pines.

Si el pin es de entrada, con el registro de datos podemos leer el estado de este pin. La escritura en un pin de entrada no tiene efecto.

Si el pin es de salida, con el registro de datos podemos escribir es este pin:

- Si GPxDAT.bit = 0, entonces en el pin de salida se ve un nivel de voltaje bajo.
- Si GPxDAT.bit = 1, entonces en el pin de salida se ve un nivel de voltaje alto.

SISTEMA DE INTERRUPCIONES

Las interrupciones permiten detectar eventos en cualquier momento sin importar en que sección de código se encuentre. Debido a esto el uso de interrupciones es de importancia para el correcto manejo de las unidades periféricas de los DSP como lo son: el manejador de eventos, convertidor analógico a digital, temporizadores internos, SPI, SCI, CAN, entre otros.

LINEAS DE INTERRUPCIONES DEL CPU

El CPU del procesador *TMS320F2812* tiene 16 líneas de interrupciones como se observa en la Figura 5.13, donde dos de ellas son llamadas no enmascarables (RESET, NMI). Las otras 14 son enmascarables [1], lo que significa que estas líneas de interrupciones pueden ser habilitada o deshabilitada por programación.

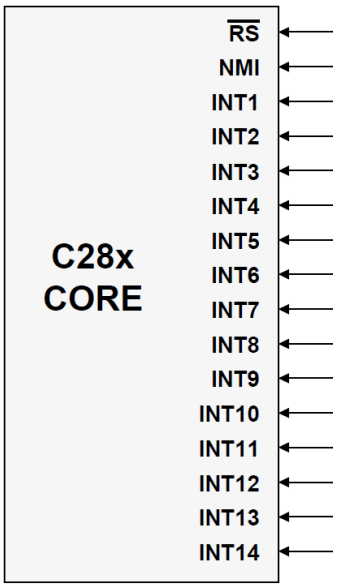


Figura 5.13 Líneas de Interrupciones del CPU [1]

En la Figura 5.14 se observa el diagrama de bloques del procedimiento para habilitación de las líneas de interrupciones

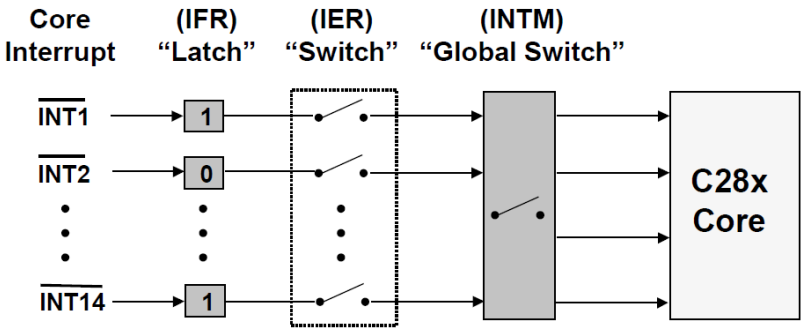


Figura 5.14 Procedimiento de habilitación de líneas de Interrupción [1]

Un evento de interrupción por alguna Línea de interrupción provocará que el registro IFR muestre un '1' en su respectivo bit. Si el respectivo bit del registro IER y el Switch global INTM están habilitados, entonces permitirá que el CPU procese el evento de interrupción.

FUENTES DE INTERRUPCIONES

El procesador *TMS320F2812* tiene una gran cantidad de fuentes de interrupciones (96 interrupciones), pero solo tiene 14 líneas de interrupciones enmascarables [1]. Por lo tanto, la unidad PIE (Peripheral Interrupt Expansion) será la encargada de manejar que por una línea de interrupción puedan circular múltiples fuentes de interrupciones; como se observa en la Figura 5.15.

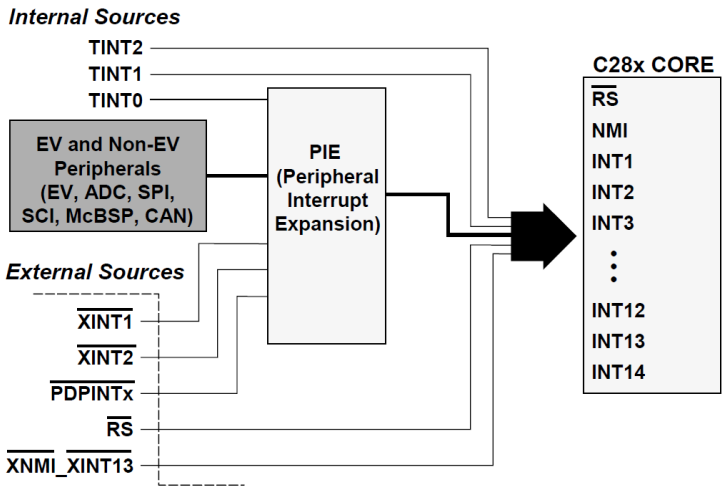


Figura 5.15 Fuentes de Interrupciones del DSP TMS320F2812 [1]

Las fuentes de interrupciones pueden ser generadas por fuentes externas, fuentes internas, o por las unidades periféricas que posee el procesador *TMS320F2812*.

Todas las 96 posibles fuentes de interrupciones son agrupadas en 12 líneas de interrupciones enmascarables, donde a cada línea le corresponderá 8 fuentes de interrupciones. Las 2 líneas de interrupciones enmascarables sobrantes estarán asociadas a los CPU Times 1 y 2. [1]

En la Figura 5.16 se observa el diagrama de bloques del procedimiento para habilitación de las fuentes de interrupciones individuales.

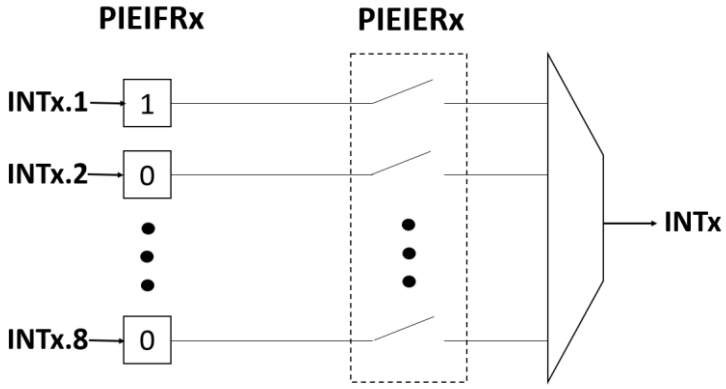


Figura 5.16 Procedimiento de habilitación de fuente de interrupción [1]

Un evento en alguna fuente de interrupción provocará que el registro PIEIFRx (donde x puede ser 1, 2, ... 12) muestre un ‘1’ en su respectivo bit. Si el respectivo bit del registro PIEIERx (donde x puede ser 1, 2, … 12) está habilitado, entonces permitirá que la fuente de interrupción pase a la línea de interrupción.

Todas las fuentes de interrupciones son conectadas a las 12 líneas de interrupciones de acuerdo a la siguiente Tabla 27. Esta tabla es denominada tabla de vectores de la unidad PIE.

CPU Interrupts	PIE Interrupts							
	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1.y	$\overline{\text{WAKEINT}}$ (LPM/WD)	TINT0 (TIMER 0)	ADCINT (ADC)	XINT2	XINT1	Reserved	PDPINTB (EV-B)	PDPINTA (EV-A)
INT2.y	Reserved	T1OFINT (EV-A)	T1UFINT (EV-A)	T1CINT (EV-A)	T1PINT (EV-A)	CMP3INT (EV-A)	CMP2INT (EV-A)	CMP1INT (EV-A)
INT3.y	Reserved	CAPINT3 (EV-A)	CAPINT2 (EV-A)	CAPINT1 (EV-A)	T2OFINT (EV-A)	T2UFINT (EV-A)	T2CINT (EV-A)	T2PINT (EV-A)
INT4.y	Reserved	T3OFINT (EV-B)	T3UFINT (EV-B)	T3CINT (EV-B)	T3PINT (EV-B)	CMP6INT (EV-B)	CMP5INT (EV-B)	CMP4INT (EV-B)
INT5.y	Reserved	CAPINT6 (EV-B)	CAPINT5 (EV-B)	CAPINT4 (EV-B)	T4OFINT (EV-B)	T4UFINT (EV-B)	T4CINT (EV-B)	T4PINT (EV-B)
INT6.y	Reserved	Reserved	MXINT (McBSP)	MRINT (McBSP)	Reserved	Reserved	SPITXINTA (SPI)	SPIRXINTA (SPI)
INT7.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
INT8.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
INT9.y	Reserved	Reserved	ECAN1INT (CAN)	ECAN0INT (CAN)	SCITXINTB (SCI-B)	SCIRXINTB (SCI-B)	SCITXINTA (SCI-A)	SCIRXINTA (SCI-A)
INT10.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
INT11.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
INT12.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

Tabla 27: Tabla de Vectores de la Unidad PIE [5]

De las 96 fuentes de interrupciones, solo 45 tienen asignado una funcionalidad. Las 51 fuentes de interrupciones sobrantes son reservadas para mejoras en el dispositivo, sin embargo, estas interrupciones pueden ser usada como interrupciones por software. [5]

RE-MAPEO DE LA TABLA DE VECTORES DEL PIE

Todas las fuentes de interrupciones mapeadas en la tabla de vectores del PIE, están protegidas por el registro EALLOW.

En el re-mapeo de la tabla de vectores del PIE se asigna la dirección de una rutina de servicio de interrupción a una fuente de interrupción de la tabla de vectores del PIE. La rutina de servicio de interrupción se ejecutará cada vez que sea procesada la respectiva fuente de interrupción por el CPU del procesador.

MANEJO DEL SISTEMA DE INTERRUPCIONES

En la Figura 5.17 se muestra el diagrama de bloques para el manejo del sistema de interrupciones del procesador *TMS320F2812*.

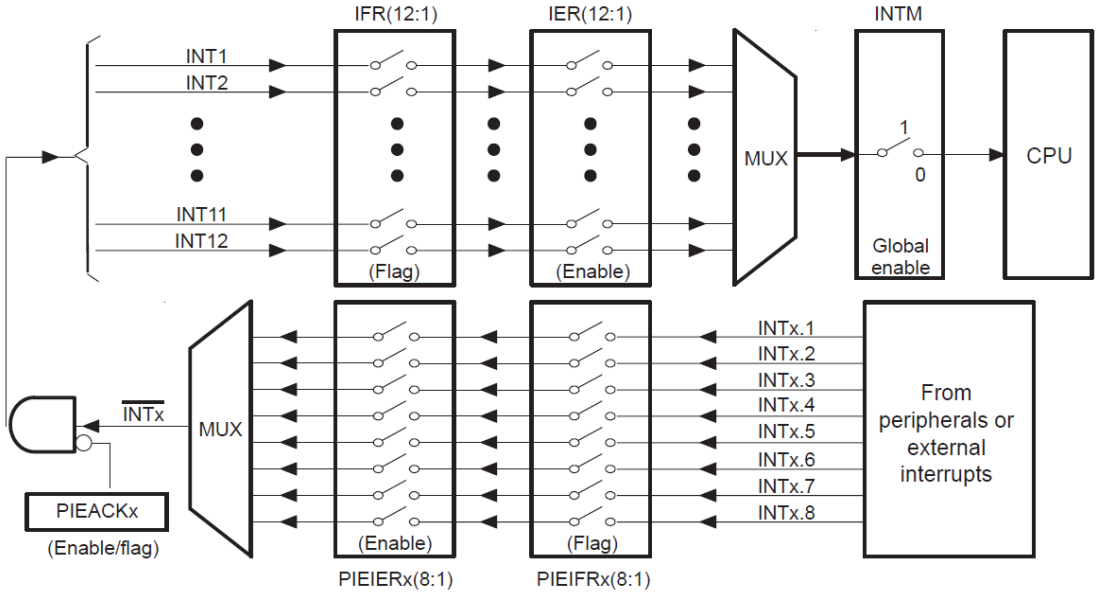


Figura 5.17 Diagrama de Bloques del sistema de interrupciones [5]

Nivel Periférico

Cuando una interrupción asociada a una unidad periférica ocurre, el bit indicador de la interrupción (IF) correspondiente a ese evento es ajustado en '1' en un registro de la unidad periférica [1]. Luego de la interrupción, el bit indicador debe ser limpiado manualmente mediante programación, para que se genere otra solicitud de interrupción.

Si el bit que habilita la interrupción (IE) es ajustado en '1', entonces la unidad periférica generará una solicitud de interrupción al PIE. [5]

Nivel PIE

El bloque PIE asigna a una línea de interrupción ocho fuentes de interrupciones. Por una línea de interrupción solo puede haber una fuente de interrupción a la vez, por lo que las fuentes de interrupciones son multiplexadas. [5]

Para que las fuentes de interrupciones sean multiplexadas, cada grupo de fuentes de interrupciones en el PIE tiene asociado un bit indicador (PIEIFRx.y) y un bit habilitador (PIEIERx.y). Además, tiene un bit de reconocimiento (PIEACK) para cada línea de interrupción (INT1 a INT12).

Una vez que se hace la petición de interrupción al bloque PIE, el respectivo bit indicador (PIEIFRx.y) es ajustado. Si el bit habilitador (PIEIERx.y) correspondiente ha sido previamente habilitado para generar una interrupción, entonces el PIE revisará el respectivo bit de reconocimiento en el registro PIEACK, para determinar que el CPU está listo para recibir una interrupción de este grupo. Si el bit correspondiente a ese grupo en el registro PIEACK es limpiado, entonces el PIE enviara una petición de interrupción al CPU.

Nivel CPU

Una vez que la petición ha sido enviada al CPU, el respectivo bit indicador de la línea de interrupción (INTx) en el registro IFR es ajustado [5]. La interrupción no será atendida hasta que sea habilitada, esto se lleva a cabo, ajustando el correspondiente bit de la línea de interrupción (INTx) en el registro IER y habilitando los Switches de interrupciones globales INTM y DBGM.

Para habilitar los switches de interrupciones globales INTM y DBGM se debe escribir en lenguaje C los siguientes comandos:

- EINT; // Habilita INTM
- ERTM; // Habilita DBGM

REGISTROS DEL SISTEMA DE INTERRUPCIONES

La Tabla 28 presenta los registros involucrados en el sistema de interrupciones y que se usaran en las prácticas.

Dirección	Registro	Nombre
----------	IER	Registro habilitador de interrupciones
----------	IFR	Registro indicador de interrupciones
0x000CE0	PIECTRL	Registro de Control PIE
0x000CE1	PIEACK	Registro de Reconocimiento PIE
0x000CE2	PIEIER1	Registro habilitador PIE del grupo INT1
0x000CE3	PIEIFR1	Registro indicador PIE del grupo INT1
0x000CE4	PIEIER2	Registro habilitador PIE del grupo INT2
0X000CE5	PIEIFR2	Registro indicador PIE del grupo INT2
0X000CE6	PIEIER3	Registro habilitador PIE del grupo INT3
0X000CE7	PIEIFR3	Registro indicador PIE del grupo INT3
0X000CE8	PIEIER4	Registro habilitador PIE del grupo INT4
0X000CE9	PIEIFR4	Registro indicador PIE del grupo INT4
0X000CEA	PIEIER5	Registro habilitador PIE del grupo INT5
0X000CEB	PIEIFR5	Registro indicador PIE del grupo INT5
0X000CEC	PIEIER6	Registro habilitador PIE del grupo INT6
0X000CED	PIEIFR6	Registro indicador PIE del grupo INT6
0X000CEE	PIEIER7	Registro habilitador PIE del grupo INT7
0X000CEF	PIEIFR7	Registro indicador PIE del grupo INT7
0X000CF0	PIEIER8	Registro habilitador PIE del grupo INT8
0X000CF1	PIEIFR8	Registro indicador PIE del grupo INT8
0X000CF2	PIEIER9	Registro habilitador PIE del grupo INT9
0X000CF3	PIEIFR9	Registro indicador PIE del grupo INT9
0X000CF4	PIEIER10	Registro habilitador PIE del grupo INT10
0X000CF5	PIEIFR10	Registro indicador PIE del grupo INT10
0X000CF6	PIEIER11	Registro habilitador PIE del grupo INT11
0X000CF7	PIEIFR11	Registro indicador PIE del grupo INT11
0X000CF8	PIEIER12	Registro habilitador PIE del grupo INT12
0X000CF9	PIEIFR12	Registro indicador PIE del grupo INT12

Tabla 28: Registros del sistema de Interrupciones [5]

Todos los registros que contienen las siglas PIE pertenecen a la estructura “PieCtrlRegs”. Si se necesita usar alguno de estos registros se deberá primero llamar a la estructura “PieCtrlRegs”, para luego acceder al campo con el nombre del registro

a usar. Por ejemplo, si se requiere cargar el valor de 64 en el registro PIEIER1, se lo realiza de la siguiente manera: “PieCtrlRegs.PIEIER1.all = 64”.

Registro habilitador de interrupciones (IER)

El registro IER permite habilitar las líneas de interrupciones enmascarables.

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

Note: R = Read access, W = Write access, –0 = value after reset

Figura 5.18 Registro Habilitar de Interrupciones (IER) [5]

En la Tabla 29 se presenta la descripción de los bits mostrados en la Figura 5.18

Bit(s)	Nombre	Descripción
15-14	Reservado	
13	INT14	0 La línea de interrupción INT14 es deshabilitada 1 La línea de interrupción INT14 es habilitada
12	INT13	0 La línea de interrupción INT13 es deshabilitada 1 La línea de interrupción INT13 es habilitada
11	INT12	0 La línea de interrupción INT12 es deshabilitada 1 La línea de interrupción INT12 es habilitada
10	INT11	0 La línea de interrupción INT11 es deshabilitada 1 La línea de interrupción INT11 es habilitada
9	INT10	0 La línea de interrupción INT10 es deshabilitada 1 La línea de interrupción INT10 es habilitada
8	INT9	0 La línea de interrupción INT9 es deshabilitada 1 La línea de interrupción INT9 es habilitada
7	INT8	0 La línea de interrupción INT8 es deshabilitada 1 La línea de interrupción INT8 es habilitada
6	INT7	0 La línea de interrupción INT7 es deshabilitada 1 La línea de interrupción INT7 es habilitada
5	INT6	0 La línea de interrupción INT6 es deshabilitada 1 La línea de interrupción INT6 es habilitada
4	INT5	0 La línea de interrupción INT5 es deshabilitada 1 La línea de interrupción INT5 es habilitada

Bit(s)	Nombre	Descripción
3	INT4	0 La línea de interrupción INT4 es deshabilitada 1 La línea de interrupción INT4 es habilitada
2	INT3	0 La línea de interrupción INT3 es deshabilitada 1 La línea de interrupción INT3 es habilitada
1	INT2	0 La línea de interrupción INT2 es deshabilitada 1 La línea de interrupción INT2 es habilitada
0	INT1	0 La línea de interrupción INT1 es deshabilitada 1 La línea de interrupción INT1 es habilitada

Tabla 29: Descripción de los bits del registro IER [5]

Registro indicador de interrupciones (IFR)

El registro IFR indica el estado de las líneas de interrupciones enmascarables. Después del reconocimiento de la línea de interrupción mediante el registro PIEACK, el bit correspondiente del registro IFR será limpiado indirectamente.

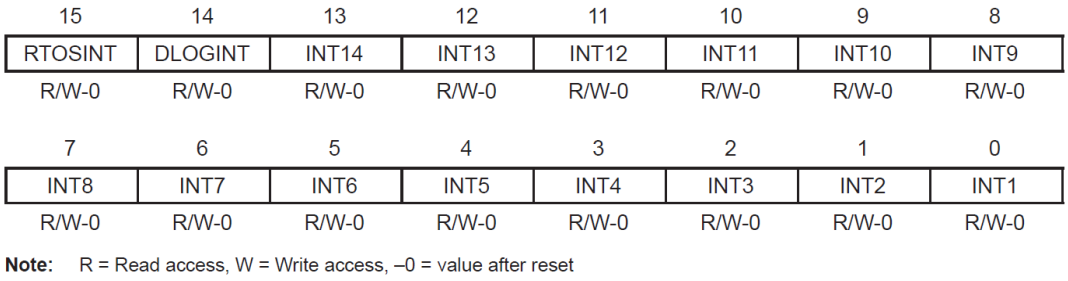


Figura 5.19 Registro Indicador de Interrupciones (IFR) [5]

En la Tabla 30 se presenta la descripción de los bits mostrados en la Figura 5.19

Bit(s)	Nombre	Descripción
15	RTOSINT	
14	DLOGINT	
13	INT14	0 No hay una interrupción INT14 pendiente 1 Al menos una interrupción INT14 está pendiente
12	INT13	0 No hay una interrupción INT13 pendiente 1 Al menos una interrupción INT13 está pendiente
11	INT12	0 No hay una interrupción INT12 pendiente 1 Al menos una interrupción INT12 está pendiente

Bit(s)	Nombre	Descripción
10	INT11	0 No hay una interrupción INT11 pendiente 1 Al menos una interrupción INT11 está pendiente
9	INT10	0 No hay una interrupción INT10 pendiente 1 Al menos una interrupción INT10 está pendiente
8	INT9	0 No hay una interrupción INT9 pendiente 1 Al menos una interrupción INT9 está pendiente
7	INT8	0 No hay una interrupción INT8 pendiente 1 Al menos una interrupción INT8 está pendiente
6	INT7	0 No hay una interrupción INT7 pendiente 1 Al menos una interrupción INT7 está pendiente
5	INT6	0 No hay una interrupción INT6 pendiente 1 Al menos una interrupción INT6 está pendiente
4	INT5	0 No hay una interrupción INT5 pendiente 1 Al menos una interrupción INT5 está pendiente
3	INT4	0 No hay una interrupción INT4 pendiente 1 Al menos una interrupción INT4 está pendiente
2	INT3	0 No hay una interrupción INT3 pendiente 1 Al menos una interrupción INT3 está pendiente
1	INT2	0 No hay una interrupción INT2 pendiente 1 Al menos una interrupción INT2 está pendiente
0	INT1	0 No hay una interrupción INT1 pendiente 1 Al menos una interrupción INT1 está pendiente

Tabla 30: Descripción de los bits del registro IFR [5]

Registro de control PIE (PIECTRL)

El registro PIECTRL permite habilitar el bloque PIE (Peripheral Interrupt Expansion).

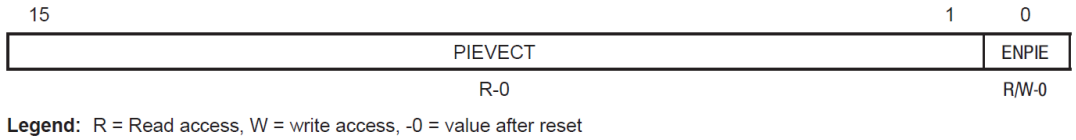


Figura 5.20 Registro de control PIE (PIECTRL) [5]

En la Tabla 31 se presenta la descripción de los bits mostrados en la Figura 5.20

Bit(s)	Nombre	Descripción
15-1	PIEVECT	Estos bits indican la dirección en la tabla de vectores PIE de la que el vector fue obtenido.
0	ENPIE	0 Deshabilita el bloque PIE 1 Habilita el bloque PIE

Tabla 31: Descripción de los bits del registro PIECTRL [5]

Registro de reconocimiento PIE (PIEACK)

El registro PIEACK reconoce la línea de interrupción y le dice al CPU que la procese.

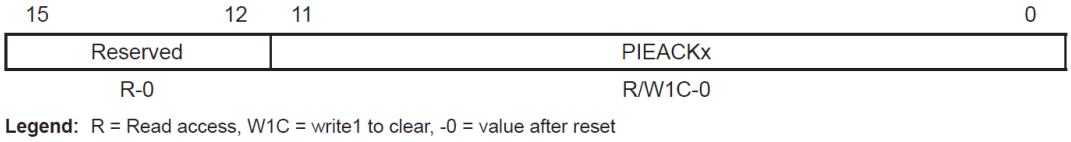


Figura 5.21 Registro de reconocimiento PIE (PIEACK) [5]

En la Tabla 32 se presenta la descripción de los bits mostrados en la Figura 5.21

Bit(s)	Nombre	Descripción
15-12	Reservado	
11-0	PIEACKx	0 No hace nada 1 Permite que el CPU procese la línea de interrupción. El bit 0 se refiere a la línea de interrupción INT1, por lo tanto el bit 11 se refiere a la línea de interrupción INT12.

Tabla 32: Descripción de los bits del registro PIEACK [5]

Registro indicador de interrupciones PIE (PIEIFRx)

El registro PIEIFRx (donde x va de 1 hasta 12) indica el estado de las fuentes de interrupciones asociada a la línea de interrupción “x”.

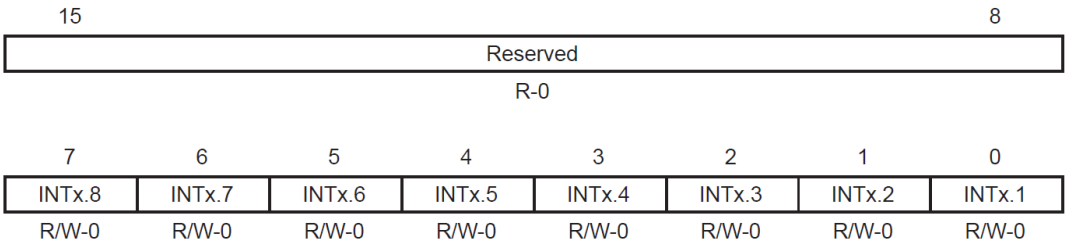


Figura 5.22 Registro Indicador de Interrupciones PIE (PIEIFRx) [5]

En la Tabla 33 se presenta la descripción de los bits mostrados en la Figura 5.22

Bit(s)	Nombre	Descripción
15-8	Reservado	
7	INTx.8	0 No ha ocurrido esta fuente de interrupción 1 Ha ocurrido esta fuente de interrupción
6	INTx.7	2 No ha ocurrido esta fuente de interrupción 3 Ha ocurrido esta fuente de interrupción
5	INTx.6	4 No ha ocurrido esta fuente de interrupción 5 Ha ocurrido esta fuente de interrupción
4	INTx.5	6 No ha ocurrido esta fuente de interrupción 7 Ha ocurrido esta fuente de interrupción
3	INTx.4	8 No ha ocurrido esta fuente de interrupción 9 Ha ocurrido esta fuente de interrupción
2	INTx.3	10 No ha ocurrido esta fuente de interrupción 11 Ha ocurrido esta fuente de interrupción
1	INTx.2	12 No ha ocurrido esta fuente de interrupción 13 Ha ocurrido esta fuente de interrupción
0	INTx.1	14 No ha ocurrido esta fuente de interrupción 15 Ha ocurrido esta fuente de interrupción

Tabla 33: Descripción de los bits del registro PIEIFRx [5]

Registro habilitador de interrupciones PIE (PIEIERx)

El registro PIEIERx (donde x va de 1 hasta 12) permite habilitar las fuentes de interrupciones asociada a la línea de interrupción “x”.

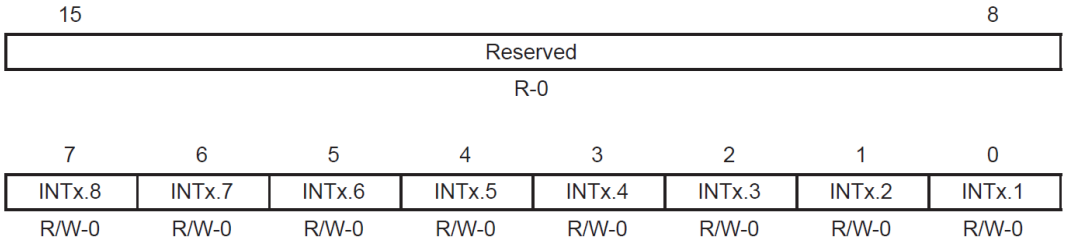


Figura 5.23 Registro habilitador de interrupciones PIE (PIEIERx) [5]

En la Tabla 34 se presenta la descripción de los bits mostrados en la Figura 5.23

Bit(s)	Nombre	Descripción
15-8	Reservado	
7	INTx.8	0 Deshabilita esta fuente de interrupción 1 Habilita esta fuente de interrupción
6	INTx.7	0 Deshabilita esta fuente de interrupción 1 Habilita esta fuente de interrupción
5	INTx.6	0 Deshabilita esta fuente de interrupción 1 Habilita esta fuente de interrupción
4	INTx.5	0 Deshabilita esta fuente de interrupción 1 Habilita esta fuente de interrupción
3	INTx.4	0 Deshabilita esta fuente de interrupción 1 Habilita esta fuente de interrupción
2	INTx.3	0 Deshabilita esta fuente de interrupción 1 Habilita esta fuente de interrupción
1	INTx.2	0 Deshabilita esta fuente de interrupción 1 Habilita esta fuente de interrupción
0	INTx.1	0 Deshabilita esta fuente de interrupción 1 Habilita esta fuente de interrupción

Tabla 34: Descripción de los bits del registro PIEIERx [5]

CPU TIMERS

El procesador *TMS320F2812* tiene tres temporizadores de 32-bit. El diagrama de bloques para un temporizador de los tres existentes se muestra en la Figura 5.24

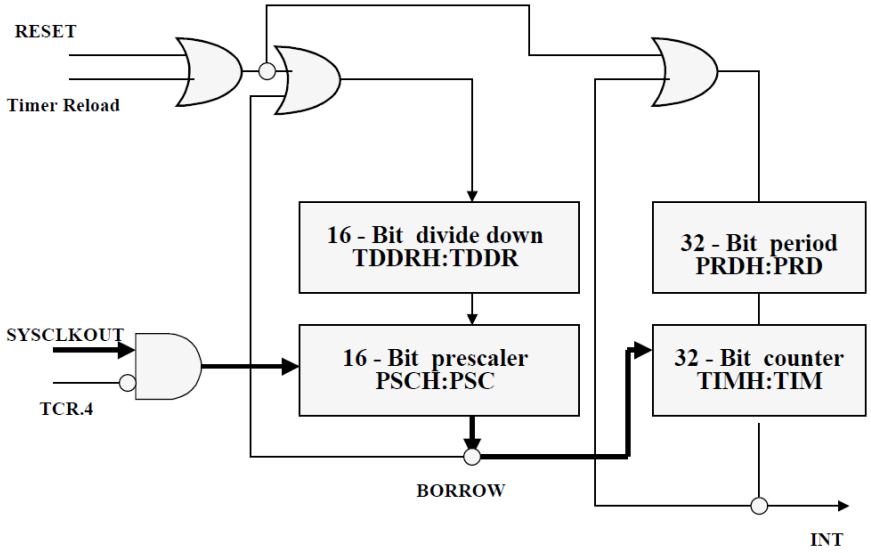


Figura 5.24 Diagrama de Bloques de un CPU Timer [1]

Cuando el bit “TSS” (quinto bit) del registro TCR se cargue con el valor de ‘0’, el temporizador comenzara a contar hasta llegar a un valor pre establecido. En el momento que se llegue a ese valor pre establecido se desencadenará una interrupción.

Los Timer1 y Timer2 son usualmente usados para operación en tiempo real en sistemas “DSP/BIOS”, mientras el Timer0 es libre para ser usado de manera general. En la práctica se usará el Timer0. [1]

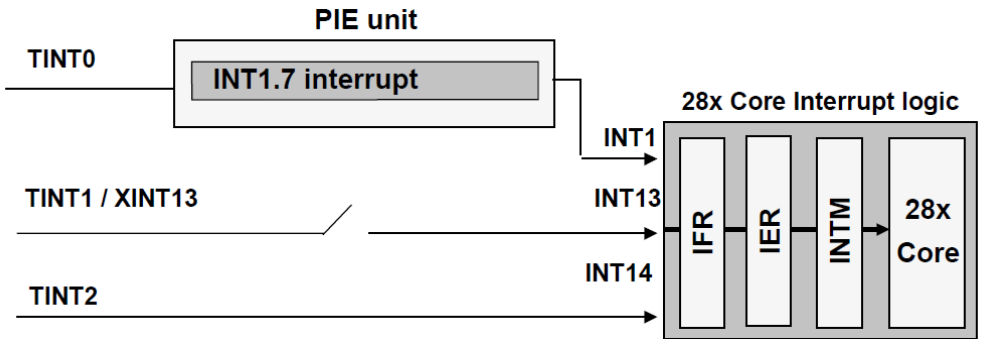


Figura 5.25 Fuente de Interrupción CPU Timer0 [1]

En la Figura 5.25 se observa que la fuente de interrupción debido al Timer0 esta mapeada en la primera línea de interrupción y es la fuente de interrupción número siete de esta línea.

Registro de control CPU Timer0 (TCR)

El registro TCR permite habilitar el conteo de los temporizadores. El registro TCR está asociado a la estructura “CpuTimer0Regs”, la cual permite habilitar el conteo del Timer0. Para habilitar el conteo se debe escribir la siguiente línea de código en lenguaje C:

- CpuTimer0Regs.TCR.bit.TSS = 0; // Inicia conteo del Timer0

SERIAL PERIPHERAL INTERFACE (SPI)

El DSP *TMS320F2812* integra funciones que permiten intercambio de datos y comunicación de varios métodos (SPI, SCI y CAN) entre el procesador y otros dispositivos. En esta sección se explicará lo necesario para realizar comunicación mediante Serial Peripheral Interface (SPI).

La interfaz SPI es un medio serial síncrono de entrada/salida y de alta velocidad que permite el desplazamiento de un flujo serial de bits de longitud programable a una cierta velocidad. Es usado frecuentemente para comunicar controladores basados en DSPs y periféricos externos como los de entrada/salida, registros de desplazamiento, convertidores ADC, etc. Es capaz de operar bajo el soporte maestro/esclavo cuando se tiene comunicación entre múltiples dispositivos. Para reducir la sobrecarga de servicio de CPU, el procesador tiene 16 niveles de recepción y transmisión FIFO.

El intercambio de datos se realiza cuando se haya configurado la operación de un dispositivo como maestro y la del resto como esclavos. El maestro es quien maneja la señal de reloj para todos los esclavos en el autobús.

DIAGRAMA DE BLOQUES

Los bits de datos son desplazados entre los dispositivos SPI a través del registro SPIDAT. Para transmitir una trama de datos, primero se debe escribir el mensaje de 16 bits en el buffer SPITXBUF para que luego sea desplazado a su destino mediante SPIDAT, por otro lado, una trama se recibe mediante SPIDAT y se almacena en el buffer SPIRXBUX para ser leída cuando se desee. En nuestras prácticas, escribiremos directamente en el registro SPITXBUF y leeremos del registro SPIRXBUF.

El maestro puede iniciar la transferencia de datos en cualquier instante de tiempo debido a que controla la señal SPICLK.

La interfaz SPI ofrece características FIFO que permiten construir hasta 16 niveles de transmisión y recepción, ampliando la capacidad de los buffers SPI para transmitir

y recibir hasta 16 veces. Para especificar el nivel deseado, se dispone de interrupciones que dependerán del estado de llenado de las dos FIFO.

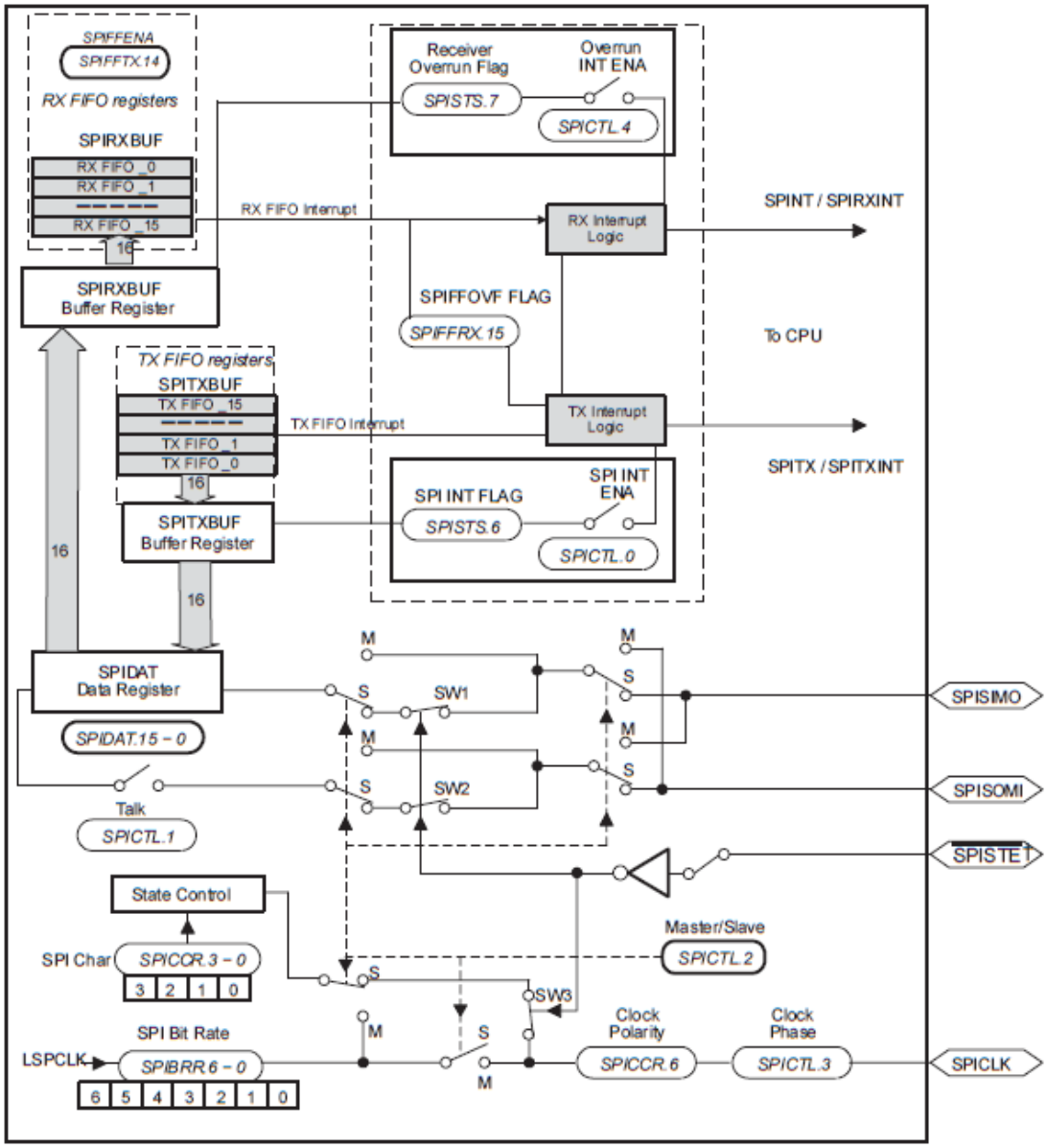


Figura 5.26 Diagrama de bloques de la unidad SPI [6]

El maestro es el responsable de generar la velocidad en la comunicación SPI. Esta velocidad es la que provee el reloj SPICLK el cual:

- Se deriva del prescalador interno de baja velocidad LSPCLK

- Está sujeto a ajustes en su valor gracias a la configuración de los registros SPIBRR, SPICCR y SPICTL

Este ajuste es implementado por TI debido a que la interfaz SPI no se encuentra estandarizada y existe una gran cantidad de dispositivos esclavos como EEPROM, ADC, sensores de temperatura y otros, que traen requisitos de configuración de sincronización variadas. Dependiendo de esta sincronización de relojes, se podrán tener diferentes eventos de envío de datos reales y ficticios por parte del maestro y del esclavo.

Una transmisión SPI siempre comienza con el MSB (bit más significativo) del registro SPIDAT y el dato recibido será desplazado en el dispositivo receptor hasta ubicarse en la posición del bit más significativo también. La transmisión y recepción es simultánea y realiza un desplazamiento de bit cada periodo de reloj SPI (SPICLK).

RESUMEN DE SEÑALES DEL MÓDULO SPI

Nombre	Descripción
SPICLK	Reloj SPI
SPISIMO	Esclavo SPI in, Maestro SPI out
SPISOMI	Esclavo SPI out, Maestro SPI in
/SPISTE	Transmisión de esclavo SPI habilitada
SPI Clock Rate	LSPCLK
SPIRXINT	Interrupción de transmisión/recepción Interrupción de recepción FIFO
SPITXINT	Interrupción de transmisión FIFO

Tabla 35: Resumen de señales del módulo SPI [6]

Las GPIO del procesador *TMS320F2812* deben ser multiplexadas para ser utilizadas como funciones de la interfaz SPI.

MODO DE OPERACIÓN SPI DE PROCESADOR F2812

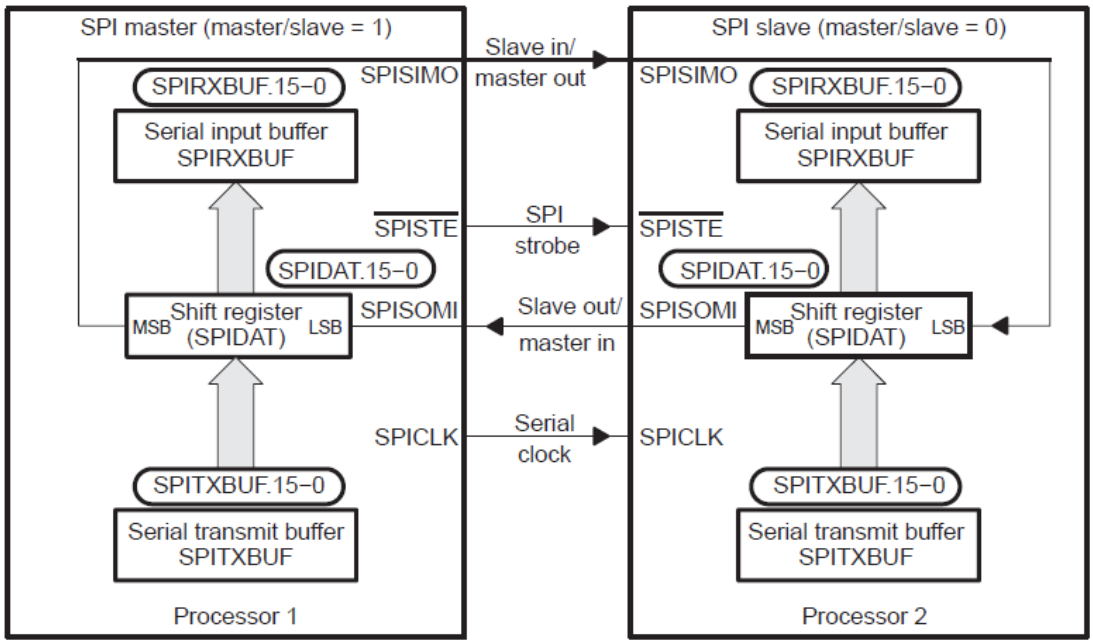


Figura 5.27 Modo de operación SPI del procesador F2812 [6]

En las prácticas desarrolladas, el procesador *TMS320F2812* se configura como maestro en la comunicación SPI (SPICTL.2 = 1), por lo tanto, es quien provee el reloj para la señal SPICLK de la comunicación. Los datos del maestro parten del pin SPISIMO, mientras que los datos que llegan al maestro ingresan por el pin SPISOMI.

La existencia de datos escritos en SPIDAT o SPITXBUF inicia la transmisión de datos en el pin SPISIMO como se observa en la Figura 5.27. Simultáneamente, el maestro puede recibir datos, los cuales ingresan por el pin SPISOMI y se desplazan por el registro SPIDAT desde el bit menos significativo (LSB).

Cuando la cantidad de bits de datos especificado ha sido desplazada a través de SPIDAT, ocurre lo siguiente:

- El contenido de SPIDAT es transferido a SPIRXBUF.
- El bit SPI INT FLAG (SPISTS.6) se establece en ‘1’.
- Si el dato en SPITXBUF es válido, se transfiere a SPIDAT para ser transmitido al esclavo, caso contrario, el reloj SPICLK se detiene.

- Si el bit SPI INT ENA (SPICTL.0) se establece en cero, se confirma una interrupción.

INTERRUPCIONES SPI

Para ejercer el control de la comunicación empleando interrupciones, se utilizan 4 bits los cuales inicializan y monitorean las interrupciones SPI.

- Bit SPI INT ENA (SPICTL.0)
- Bit SPI INT FLAG (SPISTS.6)
- Bit OVERRUN INT ENA (SPICTL.4)
- Bit RECEIVER OVERRUN FLAG (SPISTS.7)

Bit SPI INT ENA (SPICTL.0)

Permite habilitar las interrupciones sobre el llenado de los registros, para que ellas puedan ser confirmadas con las banderas.

0. Interrupción SPI deshabilitada
1. Interrupción SPI habilitada

Bit SPI INT FLAG (SPISTS.6)

Indica que un caracter ha sido desplazado en su totalidad del registro SPIDAT y cargado en el buffer de recepción SPIRXBUF en caso de haberlo programado. Esta bandera permanece en alto hasta que sea encerada por uno de los siguientes eventos:

- La interrupción sea admitida
- La CPU lea el buffer SPIRXBUF
- Se limpie el bit SPI SW RESET mediante software
- Ocurra un reseteo de sistema

Bit OVERRUN INT ENA (SPICTL.4)

Permite habilitar las interrupciones sobre el desbordamiento de los registros, para que ellas puedan ser confirmadas con la bandera RECEIVER OVERRUN.

0. Interrupción por desbordamiento deshabilitada

1. Interrupción por desbordamiento habilitada

Bit RECEIVER OVERRUN FLAG (SPISTS.7)

Indica que un nuevo carácter es recibido y cargado en SPIRXBUF antes que el previo haya sido leído desde SPIRXBUF. Esta bandera permanece en alto hasta que sea encerada por software.

REGISTROS SPI

La configuración SPI es ajustada y controlada por registros, la Tabla 36 muestra los más importantes para la comunicación SPI:

Dirección	Registro	Nombre
0x007040	SPICCR	Registro de control de comunicación SPI
0x007041	SPICTL	Registro de control de operación SPI
0x007042	SPISTS	Registro de estados SPI
0x007044	SPIBBR	Registro de velocidad de transmisión SPI
0x007047	SPIRFBUF	Registro buffer de recepción SPI
0x007048	SPITXBUF	Registro buffer de transmisión SPI
0x007049	SPIDAT	Registro de dato SPI

Tabla 36: Registros SPI [6]

Registro de control de configuración (SPICCR)

7	6	5	4	3	2	1	0
SPI SW Reset	CLOCK POLARITY	Reserved	SPILBK	SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0
R/W-0	R/W−0	R−0	R−0	R/W−0	R/W−0	R/W−0	R/W−0

Legend: R = Read access; W = Write access; −x = value after reset

Figura 5.28 Registro de control de configuración (SPICCR) [6]

En la Tabla 37 se presenta la descripción de los bits mostrados en la Figura 5.28

Bit(s)	Nombre	Descripción
7	SPI SW RESET	SPI software reset.: 1 El puerto SPI está listo para transmitir o recibir el siguiente carácter 0 Inicializa la operación de las banderas SPI a sus condiciones de reseteo
6	CLOCK POLARITY	Shift Clock Polarity. Controla la polaridad de la señal SPICLK: 1 Transferencia de datos en flanco de bajada 0 Transferencia de datos en flanco de subida
5	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto.
4	SPILBK	SPI loopback. Este modo permite validar el módulo durante el testeo de dispositivo. Es válido solo cuando se opera como maestro. 1 Modo loopback habilitado 0 Modo loopback deshabilitado
3-0	SPICHAR3-SPICHAR0	Bits de control de tamaño del carácter. Estos 4 bits especifican la cantidad de bits (1 a 16) que posee el dato. Esta información permite determinar en qué momento ha terminado el procesamiento de dicho carácter en su totalidad. Tamaño de carácter= SPICHAR[3..0]+1. Ejemplos: 0000b --> tamaño=1, 1111b --> tamaño=16

Tabla 37: Descripción de los bits del registro SPICCR [6]

Registro de control de operación SPI (SPICTL)

7 5	4	3	2	1	0
Reserved	OVERRUN INT ENA	CLOCK PHASE	MASTER/ SLAVE	TALK	SPI INT ENA
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Legend: R = Read access; W = Write access; −x = value after reset

Figura 5.29 Registro de control de operación SPI (SPICTL) [6]

En la Tabla 38 se presenta la descripción de los bits mostrados en la Figura 5.29

Bit(s)	Nombre	Descripción
7-5	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto.
4	Overrun INT ENA	Overrun Interrupt Enable. Habilita la interrupción sobre el desbordamiento de los registros. 1 Interrupción por desbordamiento habilitada 0 Interrupción por desbordamiento deshabilitada.
3	CLOCK PHASE	SPI Clock Phase Select. Controla la fase de la señal SPICLK: 1 Retraso en la señal SPICLK de medio ciclo, la polaridad es determinada por el bit CLOCK POLARITY 0 Operación normal de la señal SPICLK, dependerá únicamente de la polaridad según la configuración del bit CLOCK POLARITY
2	MASTER/ SLAVE	SPI Network Mode Control. Controla el tipo de operación de F2812 en la red configurada, es decir, maestro o esclavo. 1 Configuración como maestro 0 Configuración como esclavo
1	TALK	Master/Slave Transmit Enable. Controla la transmisión de datos mediante el manejo del dato de salida: 1 Transmisión habilitada. F2812 funcionará como dispositivo de recepción/transmisión 0 Transmisión deshabilitada. F2812 funcionará como dispositivo de recepción únicamente
0	SPI INT ENA	SPI Interrupt Enable. Controla la habilidad SPI para generar interrupciones de transmisión/recepción: 1 Interrupciones habilitadas 0 Interrupciones deshabilitadas

Tabla 38: Descripción de los bits del registro SPICTL [6]

Registro de estados SPI (SPISTS)

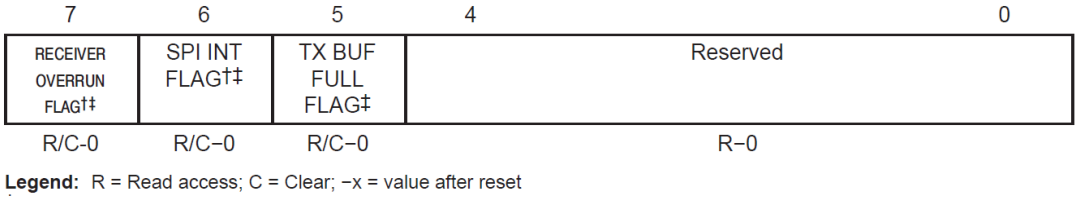


Figura 5.30 Registro de estados SPI (SPISTS) [6]

En la Tabla 39 se presenta la descripción de los bits mostrados en la Figura 5.30

Bit(s)	Nombre	Descripción
7	RECEIVER OVERRUN FLAG	SPI Receiver Overrun Flag. Esta bandera se establece en 1 cuando una operación de transmisión o recepción es completada antes de que el caracter previo haya sido leído del buffer. Este bit nos indica que el último caracter recibido ha sido sobrescrito y por lo tanto perdido. En conjunto con el bit OVERRUN INT ENA, podrán causar una interrupción que debe ser atendida. Este bit se encera cuando: - Se escribe 1 en este bit - Se escribe 0 en SPI SW RESET (SPICCR.7)
6	SPI INT FLAG	SPI Interrupt Flag. Esta bandera es configurada por hardware para indicar que la transmisión o recepción ha sido completada y que el hardware está listo para ser usado. Al mismo tiempo que se establece en uno esta bandera, el caracter recibido se traslada al buffer de recepción. En conjunto con el bit SPI INT ENA, podrán causar una interrupción que debe ser atendida. Este bit se encera cuando: - Se lee el registro SPIRXBUFF - Se escribe 0 en SPI SW RESET (SPICCR.7)
5	TX BUF FULL FLAG	SPI Transmit Buffer Flag. Este bit de sólo lectura se establece un 1 cuando el caracter es escrito en el buffer SPITXBUFF y en 0 cuando el caracter es cargado en SPIDAT (se carga automáticamente cuando el dato previo en SPIDAT se ha desplazado en su totalidad hacia afuera).
4-0	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto.

Tabla 39: Descripción de los bits del registro SPISTS [6]

Registro de velocidad de transmisión SPI (SPIBBR)

7	6	5	4	3	2	1	0
Reserved	SPI BIT RATE 6	SPI BIT RATE 5	SPI BIT RATE 4	SPI BIT RATE 3	SPI BIT RATE 2	SPI BIT RATE 1	SPI BIT RATE 0
R-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Legend: R = Read access, W = Write access, -0 = value after reset

Figura 5.31 Registro de velocidad de transmisión SPI (SPIBBR) [6]

En la Tabla 40 se presenta la descripción de los bits mostrados en la Figura 5.31

Bit(s)	Nombre	Descripción
7	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto.
6-0	SPI BIT RATE 6- SPI BIT RATE 0	SPI Bit Rate (Baud) Control. Estos bits determinan la velocidad de transferencia de un dispositivo maestro SPI. Mediante esta configuración se genera una señal de reloj para SPICLK, siempre y cuando se haya habilitado el prescalador LSPCLK como reloj base en la comunicación SPI. Son 125 velocidades a disposición. Un bit de dato será desplazado cada ciclo de SPICLK. Si el dispositivo es configurado como esclavo, recibirá esta señal en el pin SPICLK del maestro y estos bits no tendrán efecto. La velocidad de transferencia es determinada en base a: - Para SPIBRR = 3 a 127 $SPICLK = \frac{LSPCLK}{(SPIBRR + 1)}$ - Para SPIBRR = 0, 1 o 2 $SPICLK = \frac{LSPCLK}{4}$

Tabla 40: Descripción de los bits del registro SPIBBR [6]

Registro buffer de recepción SPI (SPIRXBUF)

15	14	13	12	11	10	9	8
RXB15	RXB14	RXB13	RXB12	RXB11	RXB10	RXB9	RXB8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1	RXB0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Legend: R = Read access, -0 = value after reset

Figura 5.32 Registro buffer de recepción (SPIRXBUF) [6]

En la Tabla 41 se presenta la descripción de los bits mostrados en la Figura 5.32

Bit(s)	Nombre	Descripción
15-0	RXV15-RXV0	Received Data. Una vez que SPIDAT haya recibido el caracter completo, se transferirá a este registro SPIRXBUF, donde podrá ser leído. Al mismo tiempo, el bit de la bandera SPI INT FLAG se establece en 1.

Tabla 41: Descripción de los bits del registro SPIRXBUF [6]

Registro buffer de transmisión SPI (SPITXBUF)

15	14	13	12	11	10	9	8
TXB15	TXB14	TXB13	TXB12	TXB11	TXB10	TXB9	TXB8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Legend: R = Read access, W = Write access, -0 = value after reset

Figura 5.33 Registro buffer de transmisión SPI (SPITXBUF) [6]

En la Tabla 42 se presenta la descripción de los bits mostrados en la Figura 5.33

Bit(s)	Nombre	Descripción
15-0	TXV15-TXV0	Transmit Data Buffer. Es aquí donde el siguiente carácter a transmitirse es almacenado. Cuando la transmisión de un caracter se lleva a cabo en su totalidad y siempre que TX BUF FULL Flag se establezca en uno, el contenido de este registro se transfiere automáticamente a SPIDAT y TX BUF FULL se encera.

Tabla 42: Descripción de los bits del registro SPITXBUF [6]

Registro de dato SPI (SPIDAT)

15	14	13	12	11	10	9	8
SDAT15	SDAT14	SDAT13	SDAT12	SDAT11	SDAT10	SDAT9	SDAT8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
SDAT7	SDAT6	SDAT5	SDAT4	SDAT3	SDAT2	SDAT1	SDAT0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Legend: R = Read access, W = Write access, -0 = value after reset

Figura 5.34 Registro de dato SPI (SPIDAT) [6]

En la Tabla 43 se presenta la descripción de los bits mostrados en la Figura 5.34

Bit(s)	Nombre	Descripción
15-0	SDAT15-SDAT0	Seria data. Escribiendo en SPIDAT se realizan dos funciones: - Proveer el dato de salida serial siempre que el bit TALK(SPICTL.1) se haya ajustado en 1 - Cuando se opera como maestro, inicia la transferencia de datos. Se debe tener en cuenta el bit CLOCK POLARITY y el bit CLOCK PHASE.

Tabla 43: Descripción de los bits del registro SPIDAT [6]

DUAL DAC TEXAS INSTRUMENTS TLV5617A

El TLV5617 es un convertidor digital a analógico dual de 10 bits de salida en su voltaje. Es programado con un registro serial de 16 bits, de los cuales 4 son destinados para control y 10 bits son de datos.

La interfaz entre el procesador *TMS320F2812* y la DAC es el estándar SPI y usa las siguientes conexiones para su ejecución:

Nombre de señal en TLV5617	No. Pin	Descripción	Conectado a…
AGND	5	Tierra	GND
/CS	3	Chip - Select	F2812 - GPIO D0
DIN	1	Dato de entrada	F2812 - SPISIMO
SCLK	2	Reloj SPI	F2812 - SPICLK
REF	6	Entrada de Referencia Análoga	3.3V
VDD	8	Alimentación	3.3V
OUT A	4	Salida A de DAC	JP7 - 1
OUT B	7	Salida B de DAC	JP7 - 1

Tabla 44: Conexiones entre el procesador F2812 y DAC TLV5617 [1]

Diagrama de bloques

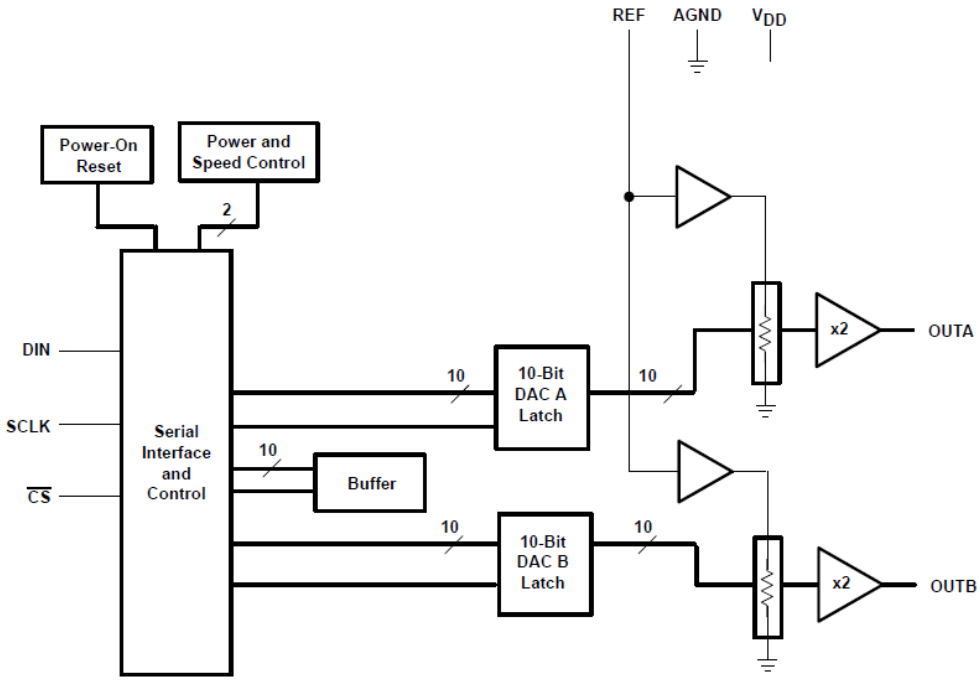


Figura 5.35 Diagrama de bloques de DAC TLV 5617 [7]

El convertidor TLV5617 es un dispositivo SPI que únicamente “escucha”, es decir, no envía datos de vuelta al procesador F2812. El voltaje REF define el valor máximo de la escala para los voltajes análogos de salida. Este voltaje es conectado a 3.3V en el set de prácticas. A pesar que TLV5617 es un convertidor digital a analógico de 10 bits, usa un multiplicador interno en cada una de sus salidas, por lo que la entrada digital se encuentra en el rango de 0 a 511. Ver Figura 5.35

Requerimientos de sincronización

Los requerimientos de sincronización se pueden entender observando el siguiente diagrama de tiempo:

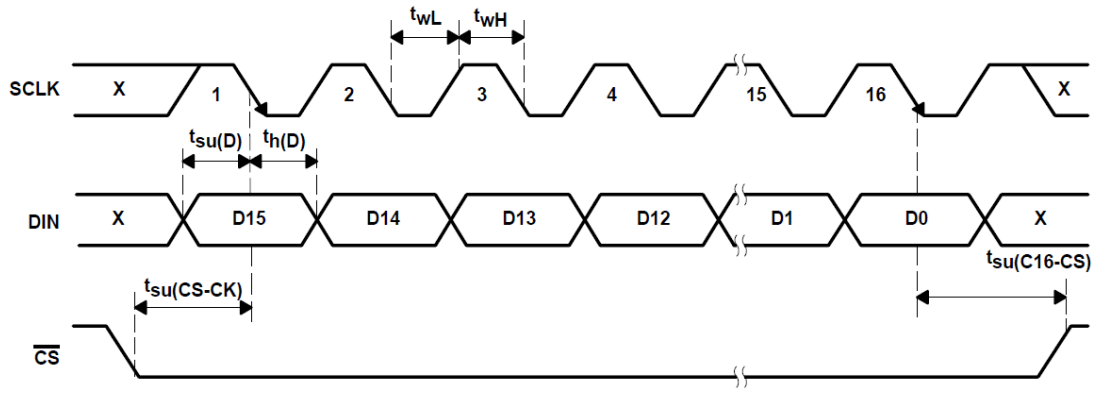


Figura 5.36: Diagrama de tiempo de requerimientos de sincronización [7]

La trama a transmitir está cubierta por un nivel de bajo voltaje de la señal /CS. MSB (bit más significativo) es el primero en transmitirse, este bit dirige el flanco descendente de la señal SCLK por mitad de un período de reloj. La operación interna de la DAC se inicia con el flanco descendente de /CS.

Formato de datos serial

La palabra de dato contiene 16 bits para el convertidos TLV5617 y se constituye en dos partes:

- Bits de control(15…12)
- Nuevo dato (11…2)

15	14	13	12	11	10	9	8
R1	SPD	PWR	R0	DATA9	DATA8	DATA7	DATA6
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
DATA5	DATA4	DATA3	DATA2	DATA1	DATA0	0	0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0

Figura 5.37 Formato de datos serial DAC TLV5617 [7]

En la Tabla 45 se presenta la descripción de los bits mostrados en la Figura 5.37

Bit(s)	Nombre	Descripción
15	R1	Register select bit[1]. Permite configurar el registro de selección del modo.
14	SPD	Speed control bit. Bit de control de velocidad. 1 Modo rápido 0 Modo lento
13	PWR	Power Control Bit. Bit de control de potencia. 1 Apagado 0 Operación normal
12	R0	Register select bit[0]. Permite configurar el registro de selección del modo.
11-2	DATA9- DATA0	Received Data. Una vez que SPIDAT haya recibido el caracter completo, se transferirá a este registro SPIRXBUF, donde podrá ser leído. Al mismo tiempo, el bit de la bandera SPI INT FLAG se establece en 1.
1-0	0	Reservado. Al leer, se obtendrá el valor de cero. La escritura no tiene efecto.

Tabla 45: Descripción de los bits del formato de datos serial [7]

La siguiente tabla enlista las posibles combinaciones para los bits del registro de selección del modo:

R1	R0	Registro
0	0	Escribir datos en DAC B y Buffer
0	1	Escribir datos en Buffer
1	0	Escribir datos en DAC A y actualizar DAC B con el contenido de Buffer
1	1	Reservado

Tabla 46: Posibles combinaciones para los bits del registro de selección de modo [7]

La elección de la combinación deseada permitirá interpretar los 12 bits de datos en el registro ya que podrían guardarse en alguna de las salidas del convertidor o actualizarse al buffer.

EEPROM M95080

Estas memorias programables y borrables eléctricamente, emplean una organización de 1024 x 8 bits y se puede acceder a ellas a través del bus SPI.

La interface entre el procesador F2812 y la EEPROM de 1024 x 8 Bits es el estándar SPI y usa las siguientes conexiones:

Nombre de señal en M95080	No. Pin	Descripción	Conectado a…
VSS	4	Tierra	GND
VCC	8	3.3V	3.3V
C	6	Reloj SPI	F2812 - SPICLK
D	5	Dato SPI de entrada	F2812 - SPISIMO
Q	2	Dato SPI de salida	F2812 - SPI SOMI
/S	1	Chip Select	F2812 - GPIO D5
/W	3	Protección de escritura	3.3V
/HOLD	7	Sostener comunicación	3.3V

Tabla 47: Conexiones entre el procesador F2812 y EEPROM M95080 [1]

Diagrama de tiempo

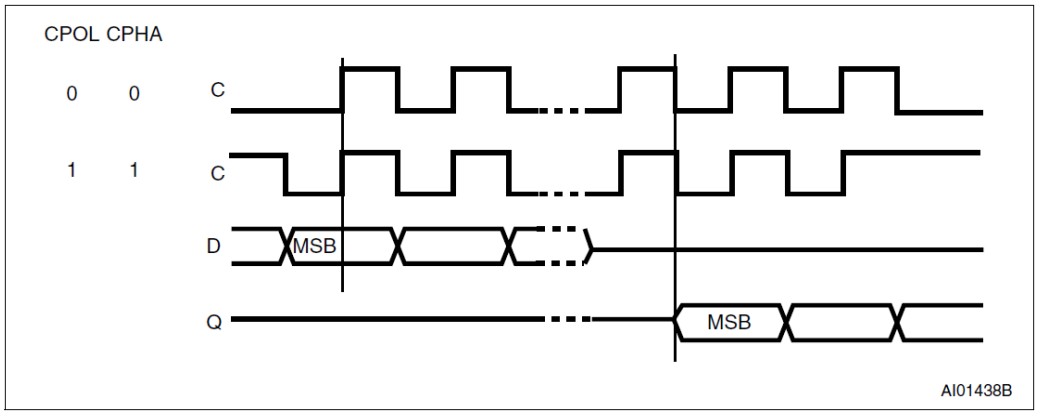


Figura 5.38 Diagrama de tiempo SPI de EEPROM M95080 [8]

Para escribir datos en la EEPROM, el DSP tiene que generar el primer bit de datos. Un retardo de ½ ciclo junto con un flanco de subida de la señal de reloj en el pin C, avisan a la EEPROM que debe almacenar datos. Por otro lado, para leer la EEPROM, el flanco de bajada del mismo reloj en el pin C causa que se envíen datos afuera.

Cuando la señal de entrada /S (Chip Select) está en un nivel alto, el dispositivo no está seleccionado y la señal en Q (Serial Data Output) será de alta impedancia. El dispositivo se encontrará en modo “Standby Power” a no ser que un ciclo de escritura se procese. Ajustando la señal /S a un nivel bajo, el dispositivo queda seleccionado y se encontrará en el modo “Active Power”.

La señal /HOLD es usada para pausar cualquier comunicación serial con algún dispositivo sin tener que usar la señal /S. Durante la condición “Hold”, la señal Q se ajusta en alta impedancia, mientras que las señales D y C no importan.

Registro de estados

El registro de estados contiene los bits de control y de estado que pueden ser ajustados o leídos dependiendo de su tipo. Este registro controla el acceso a la escritura en la memoria interna.

b7	b6	b5	b4	b3	b2	b1	b0
SRWD	0	0	0	BP1	BP0	ERL	WIP
R/W-0	R-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R-0

Figura 5.39 Registro de estados M95080 [8]

En la Tabla 48 se presenta la descripción de los bits mostrados en la Figura 5.39

Bit(s)	**Nombre**	**Descripción**
b7	SRWD	Status Register Write Disable. Permite deshabilitar cualquier acceso de escritura al registro de estados. Funciona en conjunto con la señal Write Protect (/W): 1 Sin acceso de escritura a registro de estado 0 Operación normal
b6-b4	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto.
b3-b2	BP1 - BP0	Block Protect. Estos bits no volátiles se usan para definir el área de la memoria que debe ser protegida contra cualquier acceso de escritura. Esta protección se lleva a cabo en base a lo siguiente: 00 Sin protección 01 Protección a 0x300 - 0x3FF 10 Protección a 0x200 - 0x3FF 11 Protección a 0x000 - 0x3FF

Bit(s)	Nombre	Descripción
b1	WEL	Write Enable Latch. Es un bit de control que debe ajustarse a 1 en cada acceso de escritura a la EEPROM. Luego de un ciclo de escritura exitoso, el bit es limpiado por la EEPROM. 1 Escritura habilitada 0 Escritura deshabilitada
b0	WIP	Write in progress. Indica si un ciclo de escritura interna está en progreso o no. Los ciclos de escritura inician al final de los comandos de instrucción WRITE y WRSR. 1 Ciclo de escritura en progreso 0 No existe ciclo de escritura al momento

Tabla 48: Descripción de los bits del registro de estados [8]

Conjunto de instrucciones

Para comunicarse con la EEPROM M95080, se debe tener en cuenta la siguiente tabla de instrucciones. Una instrucción constituye la primera parte de toda secuencia serial de datos entre el procesador F2812 y la EEPROM.

En la Tabla 49 se enlistan las instrucciones más importantes y que serán utilizadas en nuestro laboratorio.

Instrucción	Descripción	Código
WREN	Habilitar escritura	0000 0110
RDSR	Leer Registro de Estados	0000 0101
READ	Leer datos	0000 0011
WRITE	Escribir datos	0000 0010

Tabla 49: Conjunto de instrucciones de EEPROM M95080 [8]

WREN

Esta instrucción debe aplicarse a la EEPROM antes de cada instrucción de escritura, ya sea de datos o del Registro de Estados. El comando es una secuencia SPI que toma 8 ciclos de reloj en ser reconocida.

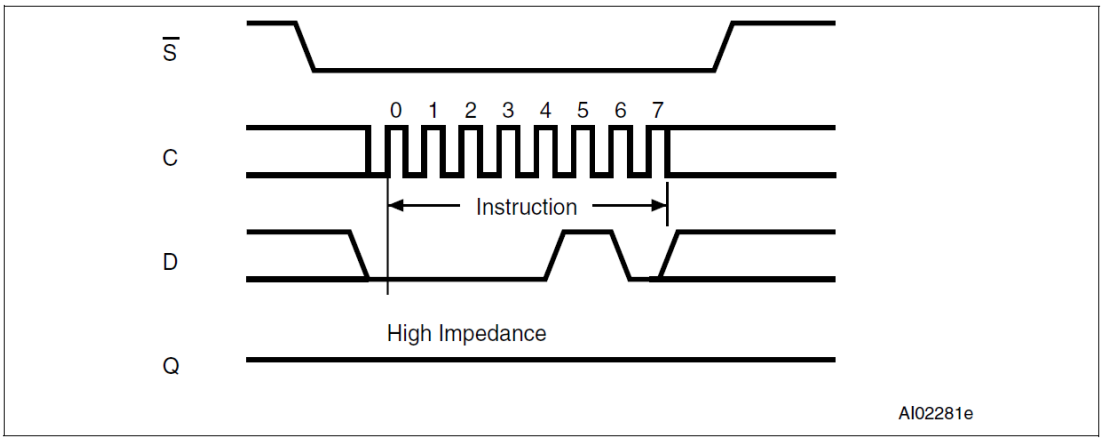


Figura 5.40 Diagrama de tiempo de la instrucción WREN [8]

RDSR

Permite la lectura del Registro de Estado. Puede ser leído en cualquier momento y se recomienda el uso de esta instrucción para chequear la bandera “Write In Progress” (WIP) antes de enviar una nueva instrucción a la EEPROM.

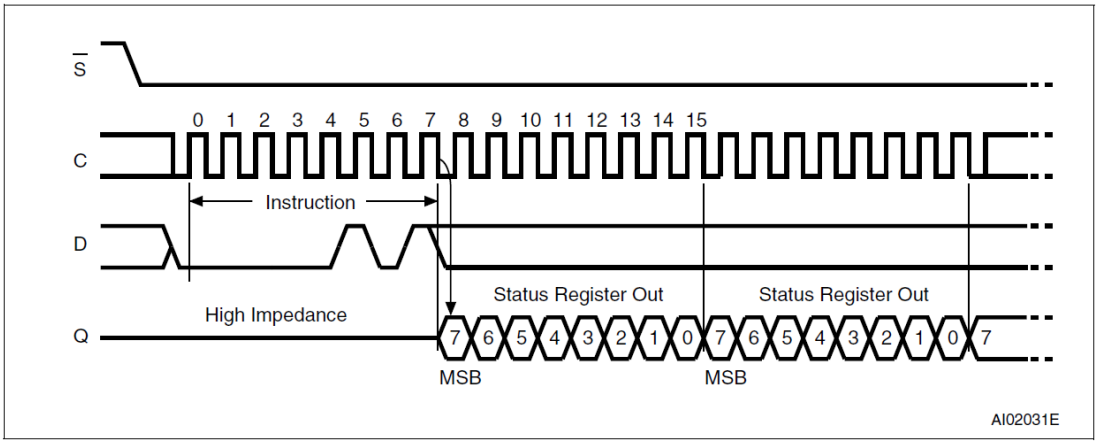


Figura 5.41 Diagrama de tiempo de instrucción RDSR [8]

READ

La instrucción de lectura es usada para leer datos de la EEPROM y enviar la información por el pin de salida. El rango de dirección para M95080 es de 0 a 1023. La dirección se especifica en 16 bits, sin embargo, los 6 bits más significativos (A15-

A10) no llevarán información que se tome en cuenta. Cabe recalcar que esta operación toma 32 ciclos de reloj para que sea completada y que en todo momento la señal “Chip Select” (/S) debe permanecer con un nivel bajo.

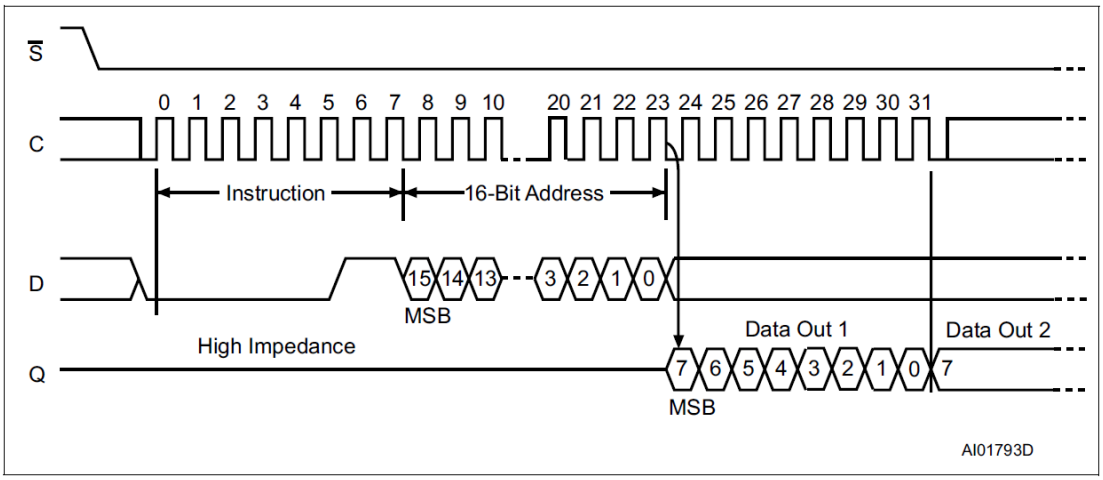


Figura 5.42 Diagrama de tiempo de instrucción READ [8]

WRITE

La instrucción de escritura se utiliza para escribir datos en la EEPROM. Al igual que READ, esta instrucción finaliza con el ajuste de la señal “Chip Select” (/S) a un nivel alto. En este punto es cuando el ciclo de escritura interna inicia, y su fin, la bandera WIP del Registro de Estado se encera.

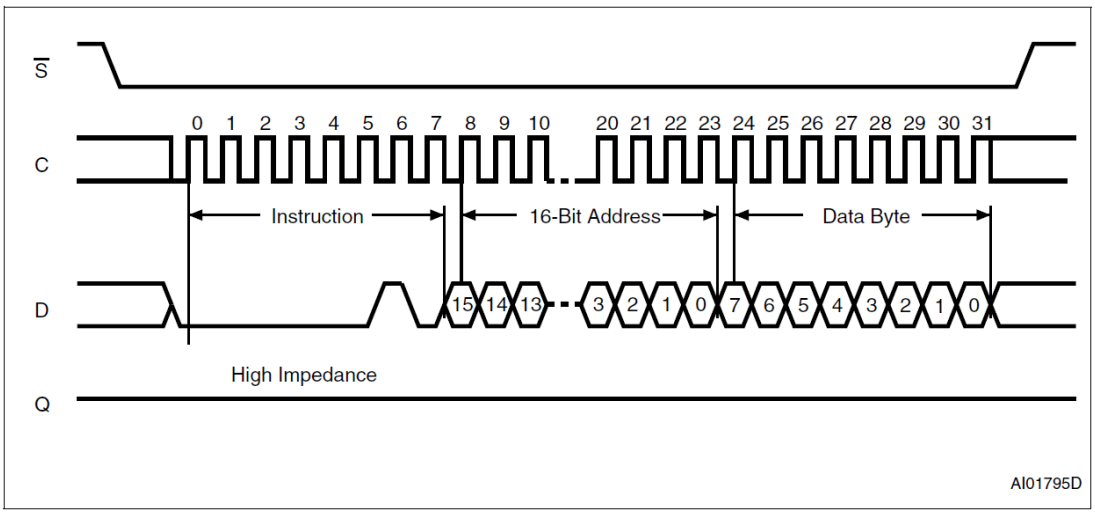


Figura 5.43 Diagrama de tiempo de instrucción WRITE [8]

SERIAL COMMUNICATION INTERFACE (SCI)

La interfaz de comunicación serial (SCI) es un puerto serial asincrónico, también conocido como UART (Universal Asynchronous Receiver-Transmitter). El módulo SCI del procesador *TMS320F2812* puede establecer comunicación digital entre su CPU y otros periféricos asincrónicos que usen el formato estándar para comunicaciones seriales RS-232-C con codificación de bits Non-Return-to-Zero (NRZ). [9] Cabe recalcar que el procesador *TMS320F2812* tiene dos módulos SCI, para el desarrollo de las prácticas se empleara el módulo SCI-A.

La Unidad periférica de comunicación SCI usa una forma de comunicación serie. La comunicación serie utiliza tres hilos para transmitir y recibir la información bit a bit, uno tras otro hasta completar un carácter. Los bits pueden tener dos estados, ya sea un nivel alto (1 lógico) y un nivel bajo (0 lógico).

INTERFAZ RS-232

Es una interfaz que designa una norma para el intercambio de datos binarios en forma serial. Esta norma se ocupa del aspecto físico para establecer la conexión tales como los tipos de conectores, niveles de señales. El conector básico definido por la norma es el DB-25, el cual consta de 25 líneas, aunque también se encuentra la versión DE-9 (Mal denominado DB-9), el cual consta de 9 líneas y es el comúnmente más usado para esta interfaz.

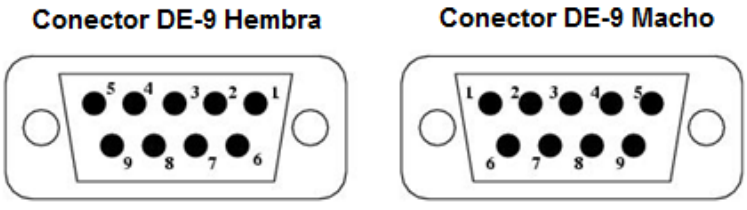


Figura 5.44 Numeración de pines de Conector DE-9 [10]

En la Figura 5.44 se puede apreciar el conector DE-9, el cual consta de 9 pines donde a cada pin se le asigna una señal. Existen variantes degeneradas de la norma,

denominada enlace RS-232 donde solo se usan 3 pines de los 9 del conector DE-9 como se puede apreciar en la Tabla 50

# Pin	Señal
2	Transmitir Datos (TXD)
3	Recibir Datos (RXD)
5	Señal de Tierra (SG)

Tabla 50: Señales de los pines del Conector DE-9 Hembra de la Tarjeta eZdspF2812

En el desarrollo de las practicas se usará esta variante degenerada para transmitir y recibir datos entre la unidad periférica de comunicación SCI del procesador *TMS320F2812* a través del conector de la tarjeta *KSPS-0504 Adaptor Board* y el puerto COM de un PC. En la Figura 5.45 se puede observar la conexión física entre el conector de la tarjeta *KSPS-0504 Adaptor Board* y el Puerto COM de la PC.

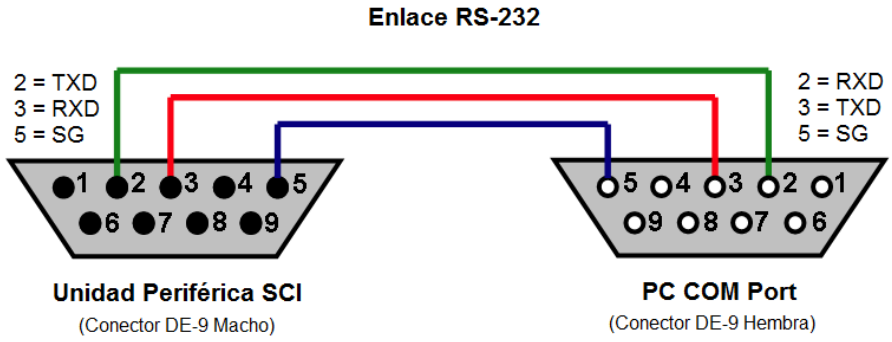


Figura 5.45 Conexión entre eZdspF2812 y puerto COM de la PC [11]

CODIFICACIÓN FÍSICA DE BITS NON-RETURN-TO-ZERO (NRZ)

La unidad SCI del procesador *TMS320F2812* utiliza una codificación NRZ, lo que significa que, en un estado de inactividad de las líneas TXD y RXD se mantendrán en un nivel alto (1 lógico. Además, se denomina NRZ porque el voltaje no cambia a cero entre bits consecutivos de valor uno. [12] Como se puede ver en la Figura 5.46

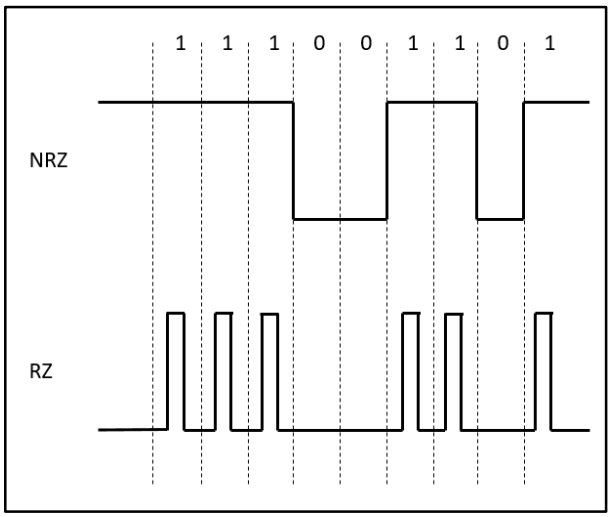


Figura 5.46 Codificación de bits Non-Return-to-Zero

DIAGRAMA DE BLOQUES DEL MÓDULO SCI

En la Figura 5.47 se observa el diagrama de bloques de la unidad periférica SCI del DSP *TMS320F2812* en el cual podemos resaltar las siguientes características:

- Dos Pines Externos: SCITXD (Pin de salida para transmisión SCI) y SCIRXD (Pin de entrada para recepción SCI)
- La velocidad de transferencia puede ser programada a diferentes valores en los registros SCIHBAUD y SCILBAUD.
- El formato para la trama de datos es configurable
- Operación en Full-Duplex es permitida
- Registros de desplazamiento independientes, uno para transmisión (TXSHF) y otro para recepción (RXSHF)
- Buffer para almacenamiento independientes, uno para transmisión (SCITXBUF) y otro para recepción (SCIRXBUF).
- Consta de unidades FIFO independientes, uno para transmisión y otro para recepción.
- Bloques lógicos para interrupciones independientes, uno para transmisión (TX Interrupt Logic) y otro para recepción (RX Interrupt Logic).

- Consta de Indicadores de detención de errores en los datos. Los cuales están conectado al bloque lógico de interrupciones RX (recepción) y que permitirá ejecutar una interrupción RXINT, siempre y cuando se encuentre habilitado RX ERR INT ENA (SCICTL1.6).

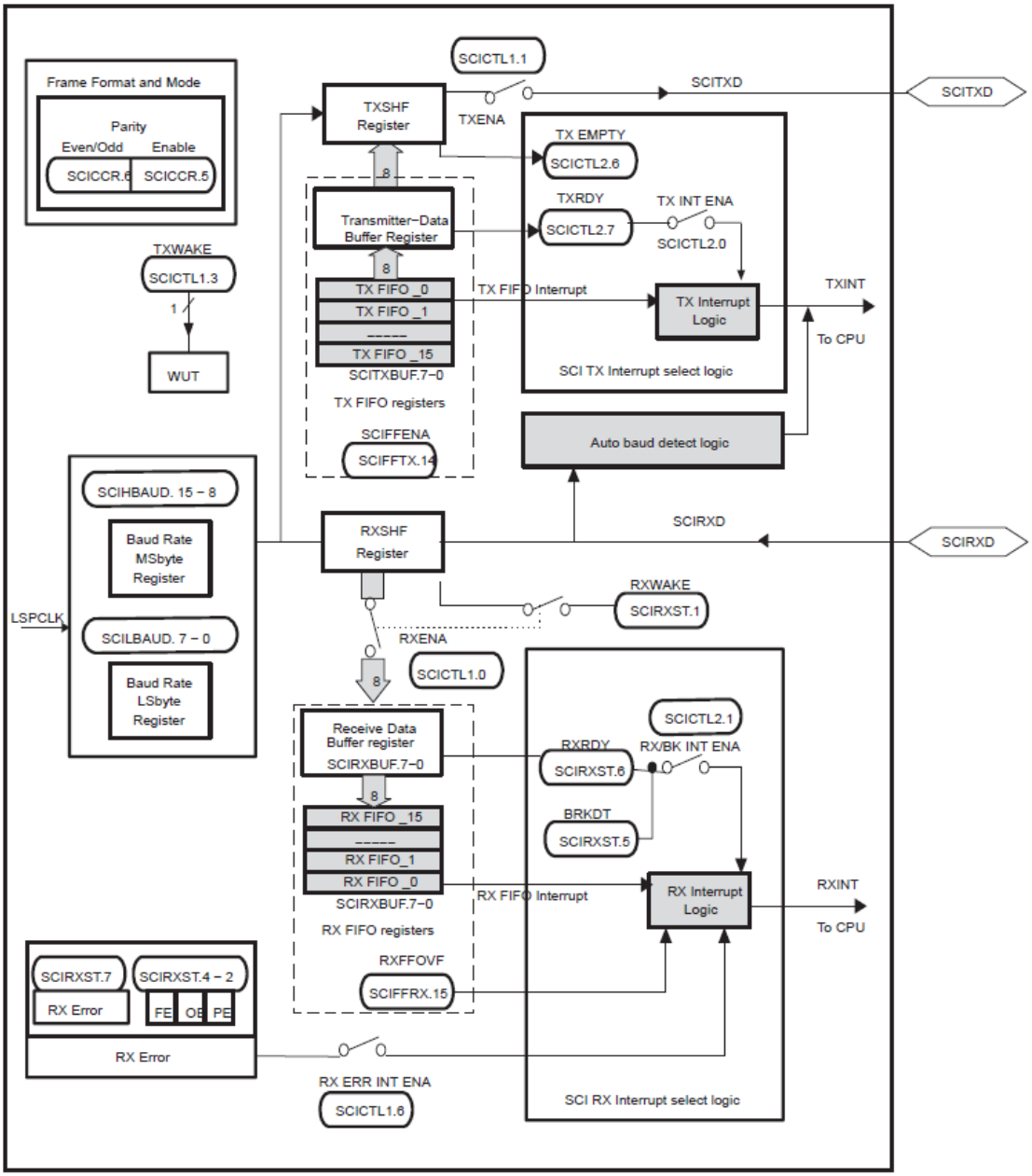


Figura 5.47 Diagrama de Bloques de la Unidad SCI [9]

La unidad básica de datos es llamada “caracter” y puede tener de uno a ocho bits, los cuales pueden ser programados en el registro SCICCR.

Para transmitir una trama de datos, se debe cargar los bits en el registro SCITXBUF (El programador tiene acceso a este registro), para luego mover los datos al registro de desplazamiento TXSHF. El registro de desplazamiento TXSHF es el encargado se desplazar los bits por el pin externo SCITXD un bit a la vez, siempre y cuando se encuentre habilitado TXENA (SCICTL1.1). El programador no tiene acceso al registro TXSHF.

La unidad FIFO de transmisión ofrece características que permiten construir hasta 16 niveles donde se almacenarán caracteres a transmitir, con esta unidad se puede transmitir hasta 16 caracteres con una sola interrupción, un carácter a la vez. El programador no tiene acceso a la carga directa de la unidad FIFO de transmisión, sino que lo hace indirectamente mediante el registro SCITXBUF, siempre y cuando sea habilitada la unidad FIFO de transmisión.

Para recibir una trama de datos, se debe leer los bits en el registro SCIRXBUF (El programador tiene acceso a este registro), el registro SCIRXBUF previamente debió haber sido cargado con los bits mediante del registro de desplazamiento RXSHF, siempre y cuando se encuentre habilitado RXENA (SCICTL1.0). El registro de desplazamiento RXSHF es el encargado de almacenar los bits que llegan por el pin externo SCIRXD un bit a la vez. El programador no tiene acceso al registro RXSHF.

La unidad FIFO de recepción también permite construir 16 niveles donde se almacenarán caracteres recibidos, con esta unidad se pueden recibir hasta 16 caracteres con una sola interrupción, un carácter a la vez. El programador no tiene acceso directo a la lectura de la unidad FIFO de recepción, sino que lo hace indirectamente mediante el registro SCIRXBUF, siempre y cuando sea habilitada la unidad FIFO de recepción.

El Modulo SCI tiene dos protocolos multiprocesadores; el modo multiprocesador *Idle-line* y el modo multiprocesador *address-bit*. Si la conexión es punto a punto, es decir no se trabaja con múltiples procesadores, entonces el modo que se usa es el *Idle-line.*

FORMATO DE TRAMA DE DATOS SCI

El formato de trama de datos SCI como se observa en la Figura 5.48 está formado de:

- Un bit de Start
- Uno a ocho bits de datos programables
- Un bit para paridad par o impar, este bit es opcional
- Uno o dos bits de Stop

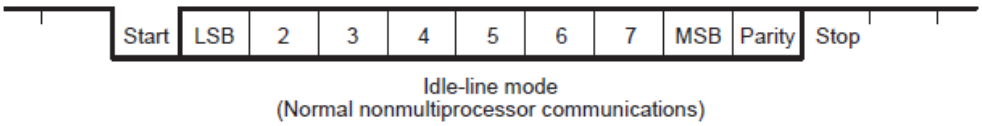


Figura 5.48 Formato de trama de datos SCI modo Idle-line [9]

Cada bit de dato será presentado durante ocho periodos del reloj SCICLK. En la Figura 5.49 se puede observar que la operación de recepción inicia validando el bit de start mediante la identificación de un nivel bajo en los cuatro primeros periodos del reloj SCICLK. Para los siguientes bits después del inicio, el procesador determinara su valor mediante el muestreo de tres valores por bit de datos en los periodos cuarto, quinto y sexto del reloj SCICLK. La longitud de cada bit de dato será de 8 periodos de reloj SCICLK. [9]

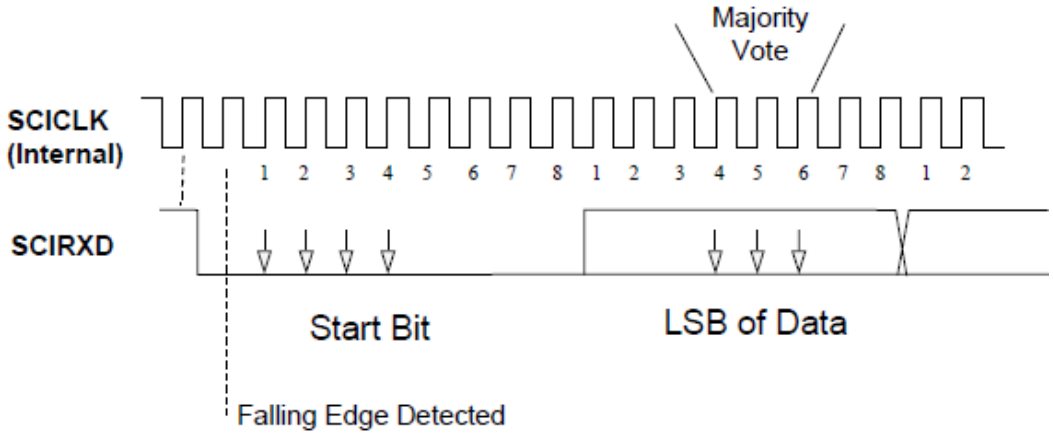


Figura 5.49 Determinación de valores de la trama de datos [1]

El receptor no necesita un reloj externo sincronizado para poder recibir los datos, por eso la comunicación es llamada asíncrona. El reloj SCICLK con el que hace el muestreo para la recepción de datos es un reloj propio de la unidad receptora.

MODO Idle-line

En el modo Idle-line los bloques de tramas son separados por largos tiempos de inactividad entre los bloques y entre las tramas en los bloques [9], tal como se ve en la Figura 5.50. Entre los bloques habrá un tiempo de inactividad de 10 o más bits en un nivel alto. Entre las tramas de un mismo bloque habrá un tiempo de inactividad de menos de 10 bits en un nivel alto

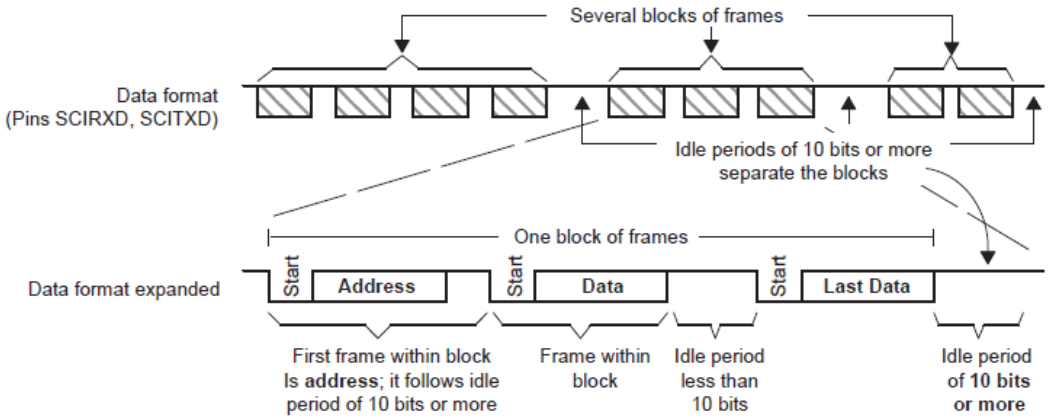


Figura 5.50 Tiempos de Inactividad entre bloques y tramas de datos [9]

CALCULO DE LA VELOCIDAD DE TRANSMISIÓN SCI

El reloj interno SCICLK que se genera, es determinado por el reloj LSPCLK del procesador y los registros SCIHBAUD y SCILBAUD. [9] Con los 16 bits de los registros SCIHBAUD y SCILBAUD (cada registro contiene 8 bits) podemos seleccionar diferentes velocidades de transmisión, según la siguiente formula:

$$BRR = \frac{LSPCLK}{Baud\ Rate\ deseado\ \times\ 8} - 1$$

El valor BRR entero calculado es el que se coloca en forma binaria en los registros SCIHBAUD y SCILBAUD. La fórmula es aplicable solo si el resultado de BRR está en el siguiente intervalo:

$$1 \leq BRR \leq 65535$$

INTERRUPCIONES SCI

En la Figura 5.51 se observa el diagrama de bloques para entender mejor el manejo de interrupciones del módulo SCI del procesador *TMS320F2812*.

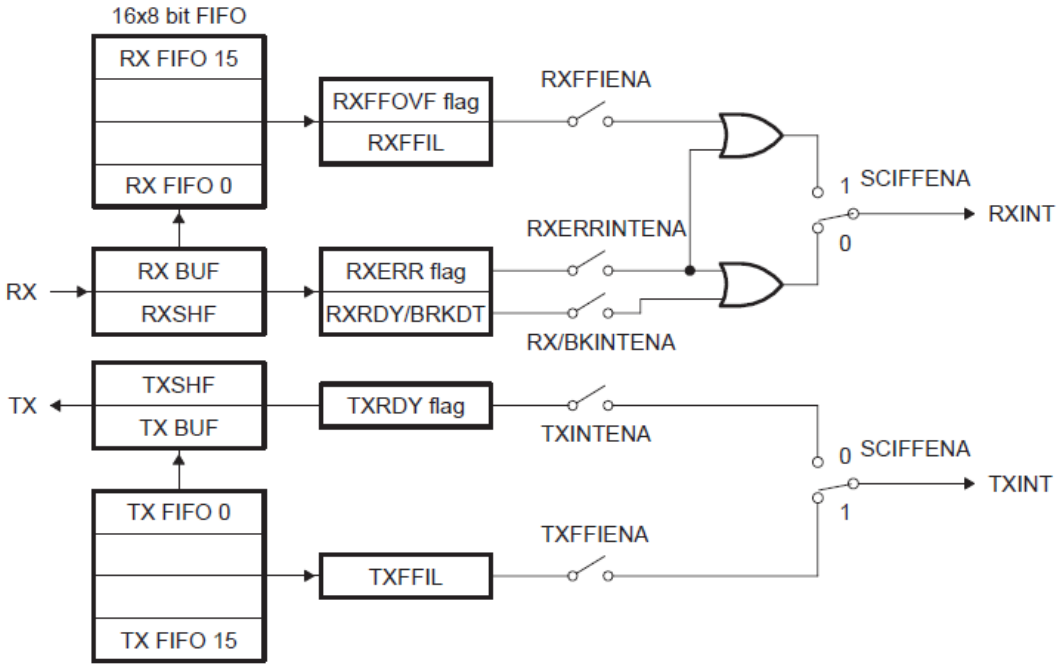


Figura 5.51 Diagrama de Bloques de Interrupciones SCI [9]

Como se había mencionado previamente, el módulo SCI contiene dos bloques lógicos de interrupciones, uno para transmisión y otro para recepción. De acuerdo al valor establecido en el bit SCIFFENA (SCIFFTX.14) se seleccionará para ambos bloques lógicos la unidad que generará la interrupción. Con respecto a la unidad que generará la interrupción se refiere a los estados de la unidad FIFO o a los estados de los registros de almacenamiento.

El boque lógico de interrupciones TX (transmisión) puede desencadenar una interrupción TXINT, por medio de los indicadores TXRDY o TXFFIL.

El indicador TXRDY desencadenará una interrupción de transmisión siempre que se cumpla lo siguiente:

- El registro SCITXBUF este vacío
- El Switch TXINTENA (SCICTL2.0) este habilitado.
- El Selector SCIFFENA (SCIFFTX.14) este en la posición de cero.

El indicador TXFFIL desencadenará una interrupción de transmisión siempre que se cumpla lo siguiente:

- Los bits del estado de la unidad de transmisión FIFO (TXFFST4-0) sean menor o igual a los bits del nivel de interrupción de transmisión FIFO (TXFFIL4-0), estos bits pueden ser observados y configurados respectivamente en el registro SCIFFTX.
- El Switch TXFFIENA (SCIFFTX.5) este habilitado.
- El Selector SCIFFENA (SCIFFTX.14) este en la posición de uno.

El boque lógico de interrupciones RX (recepción) puede desencadenar una interrupción RXINT, por medio de los indicadores RXRDY/BRKDT, RXERR Flag, RXFFOVF Flag o RXFFIL.

El indicador RXRDY/BRKDT desencadenará una interrupción de recepción siempre que se cumpla lo siguiente:

- El registro SCIRXBUF esté lleno
- El Switch RXERRINTENA (SCICTL2.1) este habilitado.
- El Selector SCIFFENA (SCIFFTX.14) este en la posición de cero.

El indicador RXERR Flag desencadenará una interrupción de recepción siempre que se cumpla lo siguiente:

- Se detecte algún error en la recepción de los datos
- El Switch RX/BKINTENA (SCICTL1.6) este habilitado.

El indicador RXFFOVF Flag desencadenará una interrupción de recepción siempre que se cumpla lo siguiente:

- La Unidad de recepción FIFO se desborde.
- El Switch RXFFIENA (SCIFFRX.5) este habilitado.
- El Selector SCIFFENA (SCIFFTX.14) este en la posición de uno.

El indicador RXFFIL desencadenará una interrupción de recepción siempre que se cumpla lo siguiente:

- Los bits del estado de la unidad de recepción FIFO (RXFFST4-0) sean mayor o igual a los bits del nivel de interrupción de recepción FIFO (RXFFIL4-0), estos bits pueden ser observados y configurados respectivamente en el registro SCIFFRX.
- El Switch RXFFIENA (SCIFFRX.5) este habilitado.
- El Selector SCIFFENA (SCIFFTX.14) este en la posición de uno.

REGISTROS SCI

La unidad periférica de comunicación SCI-A es configurada y controlada por registros, la Tabla 51 presenta los registros que se usaran en las prácticas.

Dirección	Registro	Nombre
0x007050	SCICCR	Registro de control de comunicación SCI-A
0x007051	SCICTL1	Registro 1 de control SCI-A
0x007052	SCIHBAUD	Registro Baud SCI-A, Bits más significativo
0x007053	SCILBAUD	Registro Baud SCI-A, Bits menos significativos
0x007054	SCICTL2	Registro 2 de control SCI-A
0x007057	SCIRXBUF	Registro buffer de recepción SCI-A
0x007059	SCITXBUF	Registro buffer de transmisión SCI-A
0x00705A	SCIFFTX	Registro de transmisión FIFO SCI-A
0x00705B	SCIFFRX	Registro de recepción FIFO SCI-A

Tabla 51: Registros de la unidad SCI [9]

Todos estos registros son parte de la estructura “SciaRegs”. Si necesitamos usar alguno de estos registros de la Tabla 51 se deberá primero llamar a la estructura “SciaRegs”, para luego acceder al campo con el nombre del registro a usar. Por ejemplo, si necesitamos cargar el valor de 0x0007 en el registro SCICCR, lo hacemos de la siguiente manera: “SciaRegs.SCICCR.all = 0x0007”.

Registro de Control de Comunicación SCI-A (SCICCR)

El registro SCICCR permite configurar el formato de la trama de datos y el modo de comunicación que será usado por el módulo SCI-A.

7	6	5	4	3	2	1	0
STOP BITS	EVEN/ODD PARITY	PARITY ENABLE	LOOPBACK ENA	ADDR/IDLE MODE	SCICHAR2	SCICHAR1	SCICHAR0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Legend: R = Read access, W = Write access, -0 = value after reset

Figura 5.52 Registro de Control de Comunicación SCI-A (SCICCR) [9]

En la Tabla 52 se presenta la descripción de los bits mostrados en la Figura 5.52

Bit(s)	Nombre	Descripción
7	STOP BITS	Selección de número de bits de parada: 1 Dos bits de parada 0 Un bit de parada
6	EVEN/ODD PARITY	Selección de paridad par o impar: 1 Paridad par 0 Paridad impar
5	PARITY ENABLE	Habilitar paridad: 1 Paridad es habilitado 0 Paridad es deshabilitado
4	LOOP BACK ENA	Habilitar el modo “Loop Back test” 1 Habilitar modo Loop Back 0 Deshabilitar modo Loop Back
3	ADDR/IDLE MODE	Selección del protocolo para múltiples procesadores: 1 Selección del modo *Address-bit* 0 Selección del modo *Idle-line*
2-0	SCICHAR 2-0	Selección de la cantidad de bits que tendrá el caracter en la trama de datos: 000 Un bit por caracter 001 Dos bits por caracter 010 Tres bits por caracter 011 Cuatro bits por caracter 100 Cinco bits por caracter 101 Seis bits por caracter 110 Siete bits por caracter 111 Ocho bits por caracter

Tabla 52: Descripción de los bits del registro SCICCR [9]

Registro 1 de control SCI-A (SCICTL1)

El registro SCICTL1 permite controlar la habilitación de la transmisión y recepción, el reseteo por software del módulo SCI-A y las funciones del TXWAKE y el modo SLEEP.

7	6	5	4	3	2	1	0
Reserved	RX ERR INT ENA	SW RESET	Reserved	TXWAKE	SLEEP	TXENA	RXENA
R-0	R/W-0	R/W-0	R-0	R/S-0	R/W-0	R/W-0	R/W-0

Legend: R = Read access, W = Write access, -0 = value after reset

Figura 5.53 Registro 1 de control SCI-A (SCICTL1) [9]

En la Tabla 53 se presenta la descripción de los bits mostrados en la Figura 5.53

Bit(s)	Nombre	Descripción
7	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto
6	RX ERR INT ENA	Habilitar las interrupciones por errores en el momento de la recepción: 1 Habilita estas interrupciones 0 Deshabilita estas interrupciones
5	SW RESET	SCI software reset 1 Resetea el módulo SCI-A 0 Quita el reset del módulo SCI-A
4	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto
3	TXWAKE	Selecciona el método *wake-up* para la transmisión SCI 1 La característica de transmisión seleccionada depende modo, *Idle-line* o *address-bit* 0 La característica de transmisión no es seleccionada
2	SLEEP	Habilitar modo *Sleep* 1 El modo *Sleep* es habilitado 0 El modo *Sleep* es deshabilitado
1	TXENA	Habilitar la transmisión SCI-A: 1 Habilita la Transmisión 0 Deshabilita la Transmisión
0	RXENA	Habilitar la recepción SCI-A: 1 Habilita la Recepción 0 Deshabilita la Recepción

Tabla 53: Descripción de los bits del registro SCICTL1 [9]

Registros Baud-Select SCI-A (SCIHBAUD, SCILBAUD)

Los valores en los registros SCIHBAUD y SCILBAUD determinan el baud-rate (Velocidad de transmisión) del módulo SCI-A.

15	14	13	12	11	10	9	8
BAUD15 (MSB)	BAUD14	BAUD13	BAUD12	BAUD11	BAUD10	BAUD9	BAUD8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
BAUD7	BAUD6	BAUD5	BAUD4	BAUD3	BAUD2	BAUD1	BAUD0 (LSB)
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Legend: R = Read access, W = Write access, -n = value after reset

Figura 5.54 Registros Baud-Select SCI-A (SCIHBAUD y SCILBAUD) [9]

En la Tabla 54 se presenta la descripción de los bits mostrados en la Figura 5.54

Bit(s)	Nombre	Descripción
15-0	BAUD15 – BAUD0	Los Registros SCIHBAUD (más significativos) y SCILBAUD (Menos significativos) son concatenados para formar un valor de 16-bit. En estos registros se coloca el valor de BRR que fue calculado para una determinada velocidad de transmisión, según la siguiente formula: $$BRR = \frac{LSPCLK}{Baud\ Rate\ deseado\ \times 8} - 1$$

Tabla 54: Descripción de los bits de los registros SCIHBAUD y SCILBAUD [9]

Registro 2 de control SCI-A (SCICTL2)

El registro SCICTL2 permite habilitar las interrupciones para transmisión y recepción, y visualizar el estado de los indicadores de transmisión.

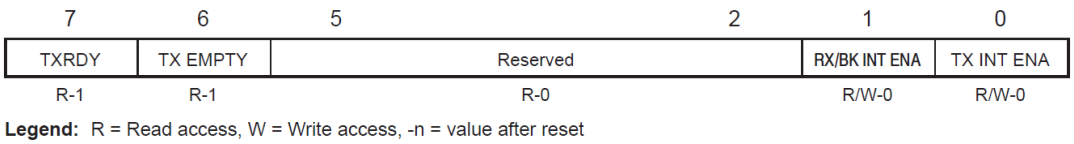


Figura 5.55 Registro 2 de control SCI-A (SCICTL2) [9]

En la Tabla 55 se presenta la descripción de los bits mostrados en la Figura 5.55

Bit(s)	Nombre	Descripción
7	TXRDY	Estado del registro SCITXBUF: 1 SCITXBUF está vacío, por lo que está listo para recibir un nuevo carácter 0 SCITXBUF está lleno
6	TX EMPTY	Estado de los registros SCITXBUF y TXSHF: 1 Ambos registros están vacíos 0 Ambos registros están cargados con datos
5-2	Reservado	Al leer, se obtendrá el valor de cero. La escritura no tiene efecto
1	RX/BK INT ENA	Habilitar interrupciones de recepción por estado de SCIRXBUF: 1 Habilita las interrupciones por RXRDY/BRKDT 0 Deshabilita las interrupciones por RXRDY/BRKDT
0	TX INT ENA	Habilitar interrupciones de transmisión por estado de SCITXBUF: 1 Habilita las interrupciones por TXRDY 0 Deshabilita las interrupciones por TXRDY

Tabla 55: Descripción de los bits del registro SCICTL2 [9]

Registro buffer de recepción SCI-A (SCIRXBUF)

El registro SCIRXBUF permite almacenar el carácter recibido.

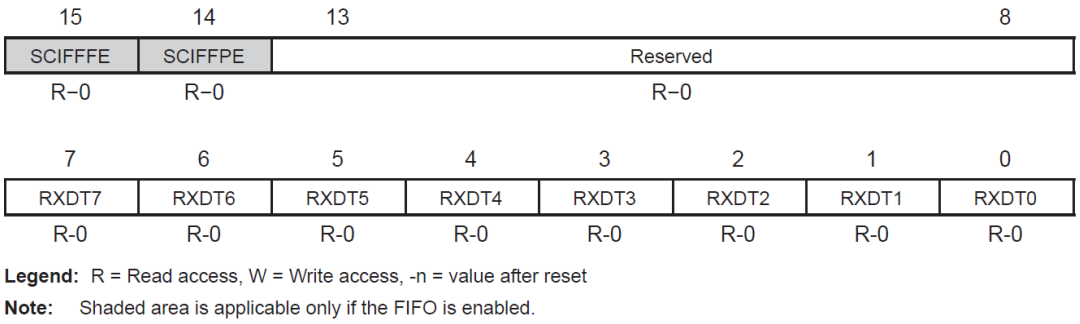


Figura 5.56 Registro buffer de recepción SCI-A (SCIRXBUF) [9]

En la Tabla 56 se presenta la descripción de los bits mostrados en la Figura 5.56

Bit(s)	Nombre	Descripción
15	SCIFFFE	Indica si en la unidad de recepción FIFO ha ocurrido un error de trama: 1 Un error de trama ha ocurrido 0 No ha ocurrido un error de trama
14	SCIFFPE	Indica si en la unidad de recepción FIFO ha ocurrido un error de paridad: 1 Un error de paridad ha ocurrido 0 No ha ocurrido un error de paridad
13-8	Reservado	
7-0	RXDT 7-0	Bits del carácter recibido

Tabla 56: Descripción de los bits del registro SCIRXBUF [9]

Registro buffer de transmisión SCI-A (SCITXBUF)

El registro SCITXBUF permite almacenar el carácter que se transmitirá.

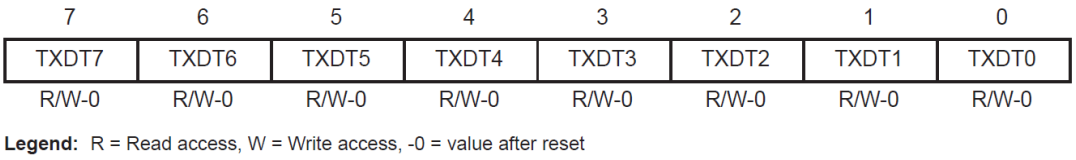


Figura 5.57 Registro Buffer de transmisión SCI-A (SCITXBUF) [9]

En la Tabla 57 se presenta la descripción de los bits mostrados en la Figura 5.57

Bit(s)	Nombre	Descripción
7-0	TXDT 7-0	Bits del carácter que será transmitido

Tabla 57: Descripción de los bits del registro SCITXBUF [9]

Registro de transmisión FIFO SCI-A (SCIFFTX)

El registro SCIFFTX permite habilitar y configurar la unidad de transmisión FIFO.

15	14	13	12	11	10	9	8
SCIRST	SCIFFENA	TXFIFO Reset	TXFFST4	TXFFST3	TXFFST2	TXFFST1	TXFFST0
R/W-1	R/W-0	R/W-1	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
TXFFINT Flag	TXFFINT CLR	TXFFIENA	TXFFIL4	TXFFIL3	TXFFIL2	TXFFIL1	TXFFIL0
R-0	W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Legend: R = Read access, W = Write access, -0 = value after reset

Figura 5.58 Registro de transmisión FIFO SCI-A (SCIFFTX) [9]

En la Tabla 58 se presenta la descripción de los bits mostrados en la Figura 5.58

Bit(s)	Nombre	Descripción
15	SCIRST	0 Resetea los canales de transmisión y recepción FIFO. la configuración de los registros FIFO se mantiene 1 Reanuda la transmisión y recepción FIFO
14	SCIFFENA	0 Deshabilita las interrupciones de las unidades de transmisión y recepción FIFO 1 Habilita las interrupciones de las unidades de Transmisión y recepción FIFO
13	TXFIFO Reset	Resetear transmisión FIFO 0 Resetea el puntero FIFO a cero y mantiene el reset 1 Habilita la operación de transmisión FIFO

Bit(s)	Nombre	Descripción
12-8	TXFFST 4-0	0000 La unidad de transmisión FIFO está vacía 0001 La unidad de transmisión FIFO tiene 1 caracter 0010 La unidad de transmisión FIFO tiene 2 caracteres 0xxx La unidad de transmisión FIFO tiene x caracteres 1000 La unidad de transmisión FIFO tiene 16 caracteres
7	TXFFINT	Interrupción de transmisión FIFO 0 No ha ocurrido una interrupción de transmisión FIFO 1 Ha ocurrido una interrupción de transmisión FIFO
6	TXFFINT CLR	0 No tiene efecto 1 Limpia la bandera de la interrupción de transmisión FIFO (TXFFINT)
5	TXFFIENA	0 Deshabilita la interrupción de transmisión FIFO 1 Habilita la interrupción de Transmisión FIFO
4-0	TXFFIL 4-0	Nivel de interrupción de transmisión FIFO. Una transmisión FIFO genera una interrupción cuando el valor de caracteres en la unidad de transmisión FIFO (TXFFST4-0) es menor o igual al nivel de interrupción de transmisión FIFO (TXFFIL4-0). Con 0x00000 la unidad de transmisión FIFO podrá transmitir 16 caracteres.

Tabla 58 Descripción de los bits del registro SCIFFTX [9]

Registro de recepción FIFO SCI-A (SCIFFRX)

El registro SCIFFRX permite habilitar y configurar la unidad de recepción FIFO.

15	14	13	12	11	10	9	8
RXFFOVF	RXFFOVR CLR	RXFIFO Reset	RXFIFST4	RXFFST3	RXFFST2	RXFFST1	RXFFST0
R-0	W−0	R/W−1	R−0	R−0	R−0	R−0	R−0

7	6	5	4	3	2	1	0
RXFFINT Flag	RXFFINT CLR	RXFFIENA	RXFFIL4	RXFFIL3	RXFFIL2	RXFFIL1	RXFFIL0
R-0	W−0	R/W−0	R/W−1	R/W−1	R/W−1	R/W−1	R/W−1

Note: R = Read access, W = Write access, -0 = value after reset

Figura 5.59 Registro de recepción FIFO SCI-A (SCIFFRX) [9]

En la Tabla 59 se presenta la descripción de los bits mostrados en la Figura 5.59

Bit(s)	Nombre	Descripción
15	RXFFOVF	Interrupción por desbordamiento de la unidad de recepción FIFO 0 La unidad de recepción FIFO no se ha desbordado 1 La unidad de recepción FIFO se ha desbordado, es decir que más de 16 caracteres han sido recibidos
14	RXFFOVF CLR	0 No tiene efecto 2 Limpia la bandera de la interrupción por desbordamiento de la unidad de recepción FIFO
13	RXFIFO Reset	Resetear recepción FIFO 0 Resetea el puntero FIFO a cero y mantiene el reset 1 Habilita la operación de recepción FIFO
12-8	RXFFST 4-0	0000 La unidad de recepción FIFO está vacía 0001 La unidad de recepción FIFO tiene 1 caracter 0010 La unidad de recepción FIFO tiene 2 caracteres 0xxx La unidad de recepción FIFO tiene x caracteres 1000 La unidad de recepción FIFO tiene 16 caracteres
7	RXFFINT	Interrupción de recepción FIFO 0 No ha ocurrido una interrupción de recepción FIFO 1 Ha ocurrido una interrupción de recepción FIFO
6	RXFFINT CLR	0 No tiene efecto 1 Limpia la bandera de la interrupción de recepción FIFO (RXFFINT)
5	RXFFIENA	0 Deshabilita la interrupción de recepción FIFO 1 Habilita la interrupción de recepción FIFO
4-0	RXFFIL 4-0	Nivel de interrupción de recepción FIFO. Una recepción FIFO genera una interrupción cuando el valor de caracteres almacenados en la unidad de recepción FIFO (RXFFST4-0) es mayor o igual al nivel de interrupción de recepción FIFO (RXFFIL4-0). Con 0x10000 la unidad de recepción FIFO podrá leer hasta 16 caracteres.

Tabla 59: Descripción de los bits del registro SCIFFRX [9]

CONTROLLER AREA NETWORK (CAN)

CAN es un sistema de bus digital estándar, que ha sido diseñado para la industria automotriz y que transmite de manera serial síncrona. CAN fue introducido y patentado por Robert Bosch, Alemania. Hoy en día es el sistema de red de comunicación básico en casi todos los vehículos de gama alta. Cuenta con una alta y confiable tasa de intercambio de datos, detección de errores y precios rentables para controladores, por tales razones, en la actualidad su uso se extiende hasta aplicaciones industriales como bus de campo, que tiene gran demanda en la industria.

El módulo mejorado eCAN (enhanced Controller Area Network) implementado en el DSP TMS320F2812 es compatible con el estándar CAN 2.0B. Utiliza el protocolo establecido para comunicarse de manera serial con otros controladores en entornos de ruido eléctrico y emplea 32 buzones (Mailboxes) totalmente configurables, tomando en cuenta estas mejoras, el módulo eCAN se convierte en una interfaz de comunicación serial robusta y a la vez versátil.

CARACTERÍSTICAS BÁSICAS DE CAN

- CAN no hace uso de direcciones físicas para direccionar, sino que cada mensaje se envía con un identificador que es reconocido por los diferentes nodos de la red. Además de ello, el identificador tiene una segunda función que es la de indicar prioridad en los mensajes, esto significa si dos o más nodos quieren transmitir al mismo tiempo, el identificador determinará la prioridad para dicha transmisión.
- La cantidad de nodos no está limitada por el protocolo CAN. A una distancia máxima del bus de 40 metros, CAN emplea una alta tasa de transferencia de datos de 1000kilobits por segundo. El tamaño del mensaje es pequeño (8 bytes de datos máximo).

- CAN provee un sofisticado mecanismo de detección de errores como lo es el chequeo cíclico redundante (CRC) y una alta inmunidad contra interferencia electromagnética, garantizando la consistencia de los datos.
- CAN se encuentra estandarizado internacionalmente por la Organización Internacional de Estandarización (ISO) y la asociación de Ingenieros Automotriz (SAE)
- El protocolo CAN hace uso de la capa física y de la capa de enlace de datos en el modelo de referencia OSI.
- El acceso al bus es manejado por el protocolo de comunicación serial “Carrier Sense Multiple Access/Collision Avoidance” (CSMA/CA).
- Existe una gran variedad de implementaciones físicas de CAN como uso de par trenzado diferencial (CAN de alta velocidad), línea simple (CAN baja velocidad) o fibra óptica.
- Según la implementación en silicón, un módulo CAN pertenece a una de las siguientes versiones.
    - CAN-Básico: Un único buffer de transmisión y dos para recepción. Un único filtro para mensajes entrantes.
    - CAN-Full: Cuenta con un servidor de mensajes, una amplia aceptación de filtrado en los mensajes entrantes, buzones (Mailboxes) configurables y con un avanzado reconocimiento de errores.

FORMATOS DE TRAMA CAN

Existen 5 formatos de trama que maneja el protocolo CAN:

1. Trama de datos.
2. Trama de sobrecarga
3. Trama remota
4. Espacio entre tramas
5. Trama de error

TRAMA DE DATOS CAN

La trama de datos es generada por el nodo CAN cuando debe transmitir un dato. Existen dos tipos de tramas de datos que permiten la transmisión y recepción de datos, la trama estándar y la trama extendida. Con una adecuada configuración, cada implementación en silicón puede manejar ambos formatos de trama.

1. CAN-Estándar: CAN-Versión 2.0A. Mensajes con identificador de 11 bits.
2. CAN-Extendido: CAN-Versión 2.0B. Mensaje con identificador de 29 bits.

Dado que en las prácticas se utilizará el formato extendido, se procederá a explicar dicha trama.

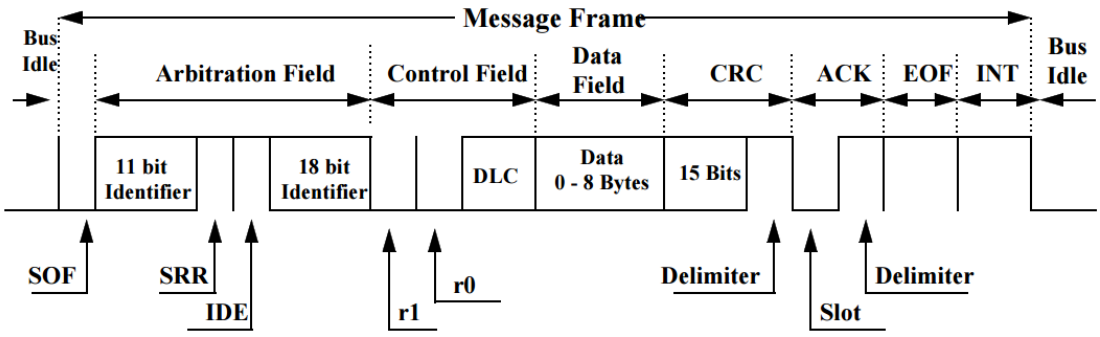


Figura 5.60 Trama de datos CAN 2.0B [14]

- Start of Frame (1 bit – dominante): La trama inicia con un bit de inicio que indica el comienzo de un mensaje causando una sincronización en todos los nodos.
- Identifier (11 bits): Refleja el nombre del mensaje y su prioridad
- SRR/RTR (1 bit – recesivos): Si SRR=1, el dato dentro de la trama no es válido, esta solicitud es para los receptores con el objetivo de enviar sus mensajes.
- IDE (1 bit): Extensión de identificador. Si IDE=1, después de este campo se especifican los 18 bits restantes de identificación para completar las características de la trama de datos extendida.
- r1 (1 bit): Reservado. Con este bit se inicia el campo de control de la trama.
- r0 (1 bit): Reservado.

- CDL (4 bits): Código de longitud de datos. Los últimos 4 bits del campo de control especifican el número de bytes de datos contenidos en el mensaje (0 – 8 bytes).
- Data (0...8bytes): Contiene el dato del mensaje de acuerdo a las especificaciones de CDL.
- CRC (15 bits): Campo de chequeo cíclico, sirve para detectar posibles errores de transmisión sin corregir los mismos.
- ACK (2 bits): Durante el tiempo de reconocimiento, cada nodo que ha recibido un mensaje libre de errores debe transmitir de vuelta un bit de reconocimiento dominante durante este mismo tiempo.
- EOF (7 bits - recesivos): Final de la trama.
- IFS (3 bit): Espacio entre tramas, espacio de tiempo para copiar un mensaje recibido del manejador de bus en el buffer.

TRAMA REMOTA CAN

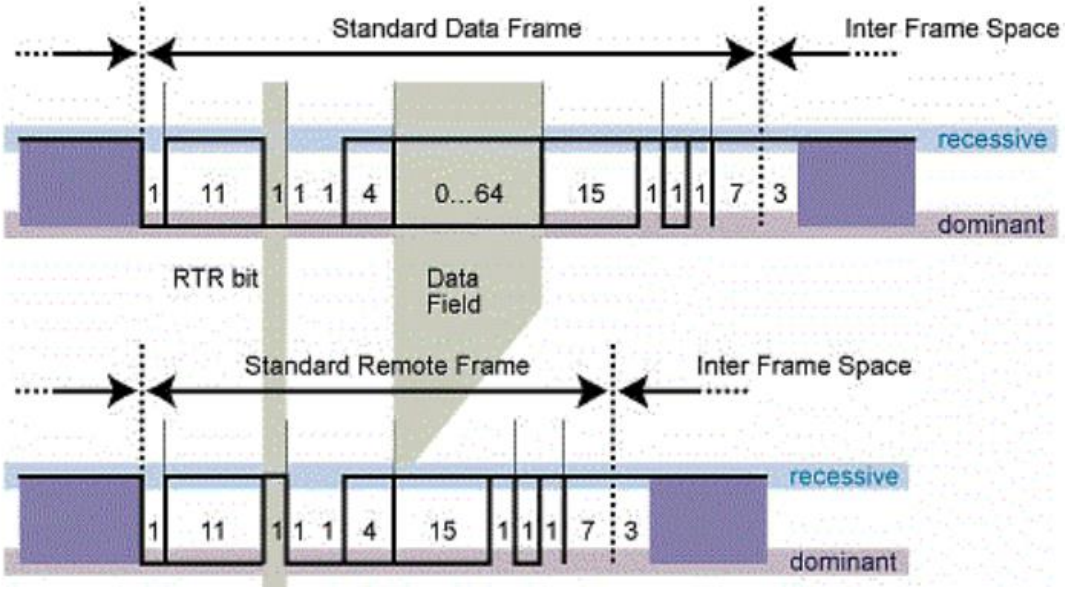


Figura 5.61 Trama remota CAN 2.0B [15]

En general, la transmisión de datos viene dada por un diseño autónomo en el que un nodo origina la trama de datos y la envía sin importar qué nodo la necesite, sin

embargo, un nodo de destino puede solicitar aquel dato del nodo fuente enviando una trama remota.

Existen dos diferencias entre la trama de datos y la trama remota, la primera es el valor que se debe ajustar en el bit SRR, y la segunda es que la trama remota no tiene campo de datos. En algún caso extraordinario que se deba transmitir una trama de datos y una trama remota al mismo tiempo, se logrará transmitir la de datos debido a su bit dominante en SRR.

ESTANDARIZACIÓN CAN

- CAN es un sistema abierto.
- ISO europea ha elaborado estándares equivalentes.
- El estándar CAN sigue el modelo de referencia OSI – ISO para interconexiones de sistemas abiertos.
- En redes de comunicación automotriz, únicamente emplea las capas 1, 2 y 7 del modelo de referencia OSI.
- La implementación de la capa 7 no está estandarizada.

LISTA DE ESTANDARIZACIONES CAN:

- ISO 11519 – 2: Especifica características de la capa de enlace de datos y la capa física de Controller Area Network (CAN) en redes de comunicación de hasta 125kbit/s.
- ISO 11519 – 3: Especifica características de la capa de enlace de datos y la capa física de Vehicle Area Network (VAN) en redes de comunicación de hasta 125kbit/s.
- ISO 11519 – 4: Especifica información de redes de comunicación de datos clase B en interfaz J1850.
- ISO 11898 – 1: Especifica características de señalización de la capa física de Controller Area Network (CAN).

CAN EN EL MODELO DE REFERENCIA OSI

Los fundamentos de la tecnología CAN que se describen en esta sección son basados en las normas ISO 11898 que describen requerimientos de las capas físicas y de enlace de datos para estructuras multipunto.

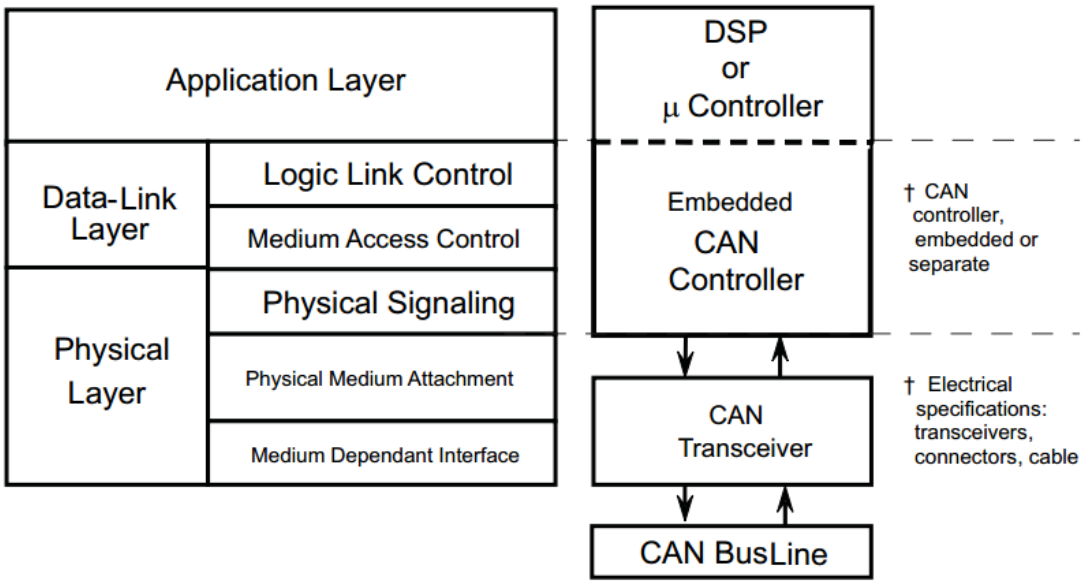


Figura 5.62 Arquitectura estándar CAN en base a modelo de referencia OSI [16]

La Figura 5.62 muestra la arquitectura estándar usando las dos capas inferiores del modelo de referencia OSI, la capa de enlace de datos y la capa física, todo esto según la norma ISO 11898

La capa de enlace de datos tiene como objetivo que la transferencia de mensajes que parten de un nodo y se dirigen a la red se dé sin errores. Es decir, se ocupa de las sumas sucesivas de comprobación y en caso de que sea necesario rellenar espacios con bits, también lo realiza. Además de ello, en esta capa se espera por el mensaje de reconocimiento del receptor.

- ✓ Protocolo de formato y transmisión de mensajes
- ✓ CSMA/CA

La capa física consiste en la arquitectura de hardware básica y necesaria para que una red de comunicación CAN funcione exitosamente, todo lo que debe cumplir se

detalla en la norma mencionada anteriormente. En general, convierte valores digitales en impulsos eléctricos que salen de un nodo y que al final de la secuencia ingresarán de la misma manera al nodo destino. La capa física siempre se implementa en hardware, a diferencia de las demás capas que pueden ser implementadas tanto en hardware como en software.

- ✓ Interfaz de transmisión de doble hilo diferencial, par trenzado (CAN-High y CAN-Low), e inclusive fibra óptica
- ✓ NRZ, PWM

En la Figura 5.62 se especifica como la capa de aplicación proporciona las funciones de comunicación de alto nivel que demanda el sistema de referencia OSI. La implementación de estas funciones puede ser por software o manejadas por un protocolo de capa superior (integrado en un DSP o un microcontrolador) como lo son:

- ✓ CANopen
- ✓ DeviceNet
- ✓ Smart Distributed Systems (SDS)
- ✓ CAN Kingdom
- ✓ OSEK/VDX

CSMA/CA

El protocolo de comunicación serial “Carrier Sense Multiple Access/Collision Avoidance” (CSMA/CA) se basa en un principio de múltiples maestros, es decir, todos los nodos están permitidos a usar CAN como un nodo maestro. Este procedimiento brinda seguridad en casos de conflictos de acceso, protegiendo el mensaje de mayor prioridad ante colisiones por transmisiones simultáneas.

El bus utiliza un mecanismo de lógica llamado “Wired-AND” (Ver Figura 5.63) en el cual existen dos estados para el bus, estado dominante (equivalente a cero lógico) y recesivo (equivalente a uno lógico).

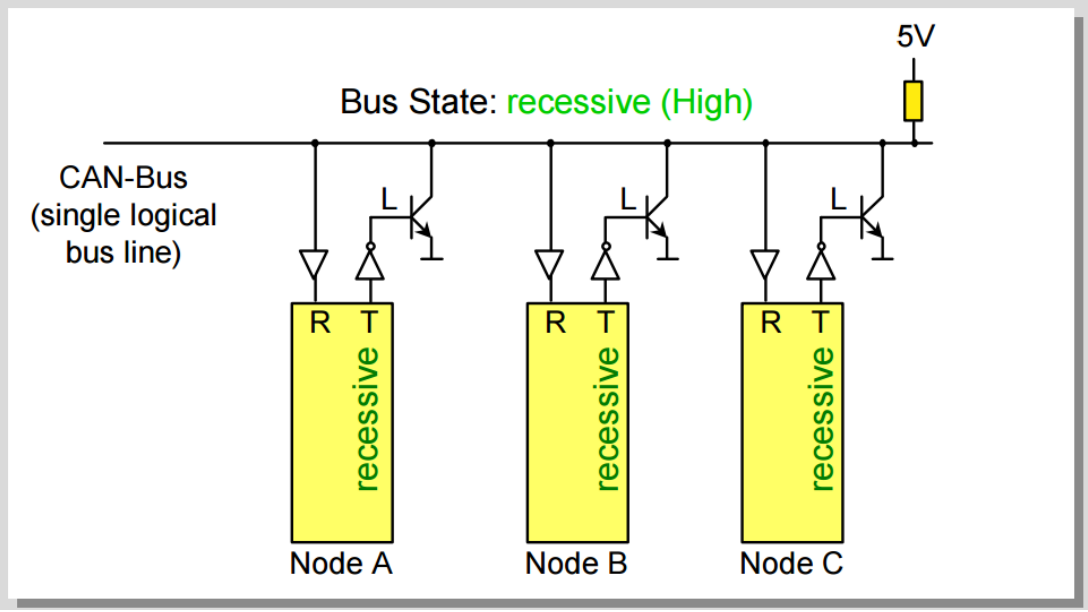


Figura 5.63 Mecanismo lógico Wired-AND en estado recesivo [15]

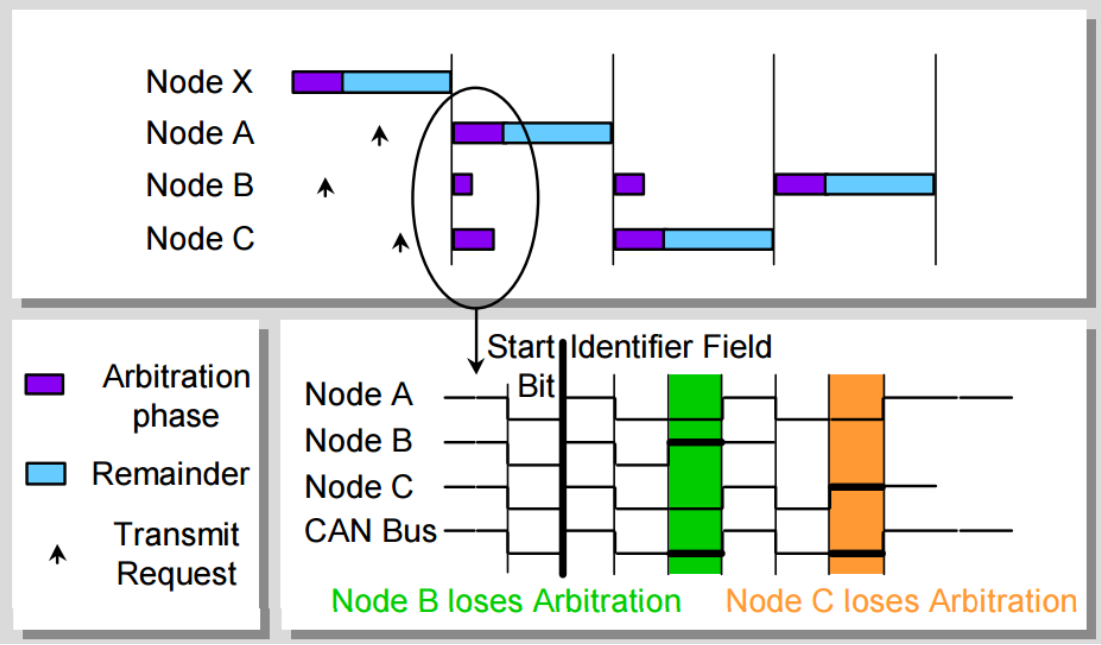


Figura 5.64 Ejemplo de una red "Wired-AND" [15]

En la Figura 5.64 se ha trazado la transmisión de 3 buses llamados A, B y C, y el resultado del estado del bus de acuerdo al principio “Wired-AND”

Si dos o más nodos de bus empiezan su transmisión al mismo tiempo, la colisión del mensaje es evitada por el concepto de arbitrariedad.

En un determinado momento, los nodos A y C envían un bit de identificación dominante. El nodo B por otro lado envía un bit de identificación recesivo, pero leerá de vuelta un bit dominante. El nodo B pierde el bus y se ajusta a modo de receptor. Luego del transcurso de algunos bits, el nodo C también pierde el bus, ajustándose a modo de receptor. Al finalizar el campo de identificación de todos los nodos, se puede concluir que A mantuvo la dominancia y por lo tanto se constituyó como el mensaje de mayor de prioridad el enviado por él.

Los nodos B y C automáticamente intentarán repetir la transmisión una vez que el bus retorne al estado de reposo. Debido a que el nodo B perdió el bus antes que el nodo C, será el C el siguiente en transmitirse, seguido finalmente de B.

No es permitido que los diferentes nodos de la red contengan el mismo identificador dado que causarán problemas al momento de solucionar errores de colisión.

CAN-HIGH

Para las líneas de transmisión CAN, se debe escoger un medio que permita transmitir dos posibles bits de estados, el dominante y el recesivo, por tal razón y debido a su bajo coste, la mejor opción se ha convertido en el uso del par trenzado. Estos dos niveles de voltaje en las líneas serán conocidos como CAN-High y CAN-Low y son manejados por nodos con señales diferenciales. El par trenzado termina con una resistencia de terminación en cada fin de línea de bus, usualmente de 120[ohms]. Ver Figura 5.65.

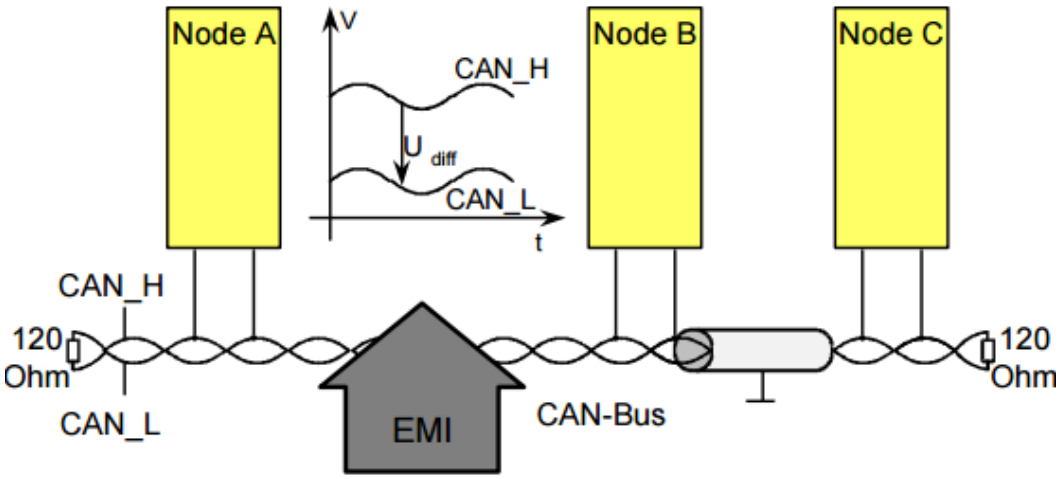


Figura 5.65 Esquema de transmisión diferencial CAN [15]

La transmisión diferencial de CAN es insensible a la interferencia electromagnética debido a que ambas líneas del par trenzado son afectadas en la misma manera, lo cual al restarse elimina la interferencia y deja la señal sin afectación.

Para generar niveles de voltaje de transmisión diferencial acorde a CAN High, se necesita un dispositivo transceptor adicional, en nuestro caso es el SN65HVD23x.

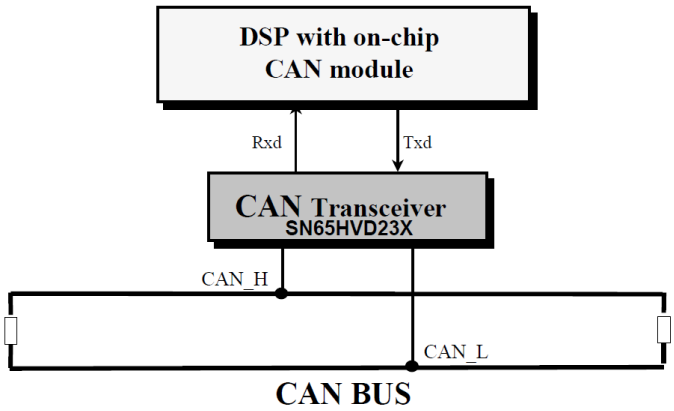


Figura 5.66 Esquema del uso del transceptor SN65HVD23x en una transmisión diferencial CAN [1]

MANEJADOR DE ERRORES CAN

La mayor parte de los errores deben detectarse y autocorregirse por el propio procesador F2812.

Una notificación automática informará al resto de nodos que un error ha sido hallado

Todos los nodos tienen que cancelar el último mensaje que hayan recibido.

La transmisión se repite automáticamente gracias al manejador de bus.

Cada nodo CAN permanece en uno de los siguientes estados de error:

1. Error activo
2. Error pasivo
3. Bus apagado

El error activo es el estado al que usualmente se ajusta el nodo al resetearlo. Mediante el bus, el nodo puede recibir o trasmitir mensajes e inclusive tramas de error sin ninguna restricción. Durante la comunicación CAN, el contador CAN se va actualizando acorde a las siguientes reglas:

a) Existen dos conteos de errores:

   1. Conteo de errores de transmisión (REC)

   2. Conteo de errores de recepción (TEC)

b) El nodo transmisor reconoce un error: TEC =TEC+8
c) El nodo receptor reconoce un error: REC=REC+1
d) El nodo receptor reconoce un error, después de enviar una trama de error: REC=REC+8
e) En el estado de error activo, el nodo encuentra un bit de error durante la transmisión de una trama de error: TEC=TEC+1
f) Recepción exitosa: REC=REC-1
g) Transmisión exitosa: TEC=TEC-1

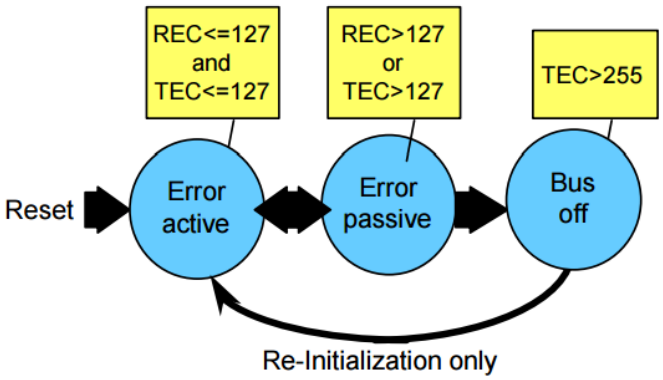


Figura 5.67 Esquema lógico del Manejador de Errores CAN [15]

El estado de error activo será el actual en los nodos mientras los conteos del mismo sean menores o igual a 127 (Ver Figura 5.67). Cuando sobrepasen esta cifra, los nodos cambiarán a estado de error pasivo. En estado de error pasivo, cualquier transmisión o recepción debe continuar hasta su fin, sin embargo, acabada dicha acción, el nodo debe suspender la transmisión.

Solo tramas de error pueden ser transmitidas por un nodo que se encuentre en estado de error pasivo, siempre y cuando haya esperado el transcurso de 8 bits.

Si ambos contadores presentan valores menores a 128 debido a comunicaciones de bus exitosas, entonces el nodo podrá volver al estado de error activo.

El estado de bus apagado se presenta cuando el contador de error de transmisión excede el valor de 255. Ante esta situación, todas las actividades se detienen, impidiendo que el nodo participe en la comunicación. Para retornar al estado de error activo, se debe reinicializar el nodo y por lo tanto el contador.

El protocolo CAN consta de sofisticados mecanismos de detección de errores:

1. Chequeo cíclico redundante (CRC): El transmisor calcula una suma de chequeo para la secuencia de bits desde el inicio de la trama hasta el final del campo de datos. La secuencia CRC es transmitida en el campo CRC de la trama de datos CAN. El receptor también realiza sus cálculos de la secuencia de bits CRC usando el mismo principio para luego comparar los valores con la secuencia recibida. En caso de no coincidir se genera el error (transmisión

o recepción) por CRC, el receptor descarta el mensaje y transmite una trama de error para solicitar la retransmisión de la trama.

2. Chequeo de reconocimiento: El transmisor chequea el campo ACK de la trama (Ver Figura 5.68) para determinar si el mensaje que es enviado contiene bits dominantes. Si este es el caso, al menos habrá un nodo que reciba la trama correctamente. Caso contrario, un error de reconocimiento ocurrirá y el mensaje debe ser repetido. No se genera una trama de error en este chequeo.

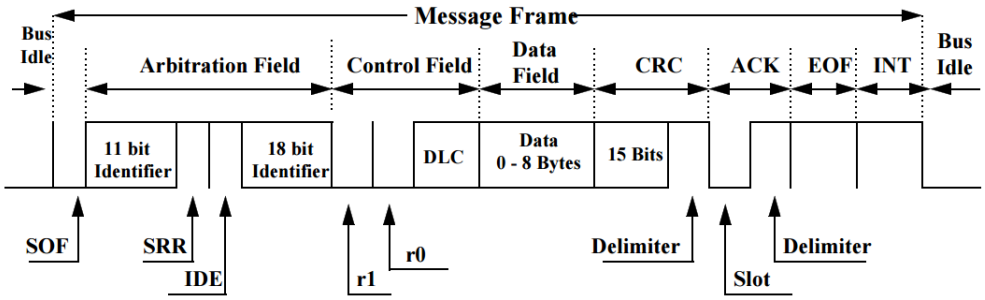


Figura 5.68 Trama de datos CAN2.0B [14]

3. Chequeo de trama: Este mecanismo consiste en detectar un bit dominante en alguno de los siguientes campos del transmisor:
   - ✓ CRC Delimiter,
   - ✓ ACK Delimiter,
   - ✓ End Of Frame (EOF)
   - ✓ Espacio entre tramas

   Ante alguno de estos eventos, se genera un error de trama, por lo que se debe repetir el mensaje.

4. Monitoreo de bit: Todos los nodos realizan este mecanismo, un error de bit ocurre cuando el transmisor envía un bit dominante, pero detecta un bit recesivo o viceversa. Se genera un error de trama por lo que se debe repetir el mensaje. Cabe recalcar que durante el campo de arbitrariedad y el “Slot” de reconocimiento no se generará un error si se llega a detectar esta variación.

5. Chequeo de bit de relleno: Si seis bits consecutivos son detectados con la misma polaridad durante el transcurso entre el inicio de la trama y el delimitador CRC, la regla de relleno de bit habrá sido violada. Ante dicho evento, se genera un error de trama y el mensaje debe ser repetido.

MODULO CAN EN F2812

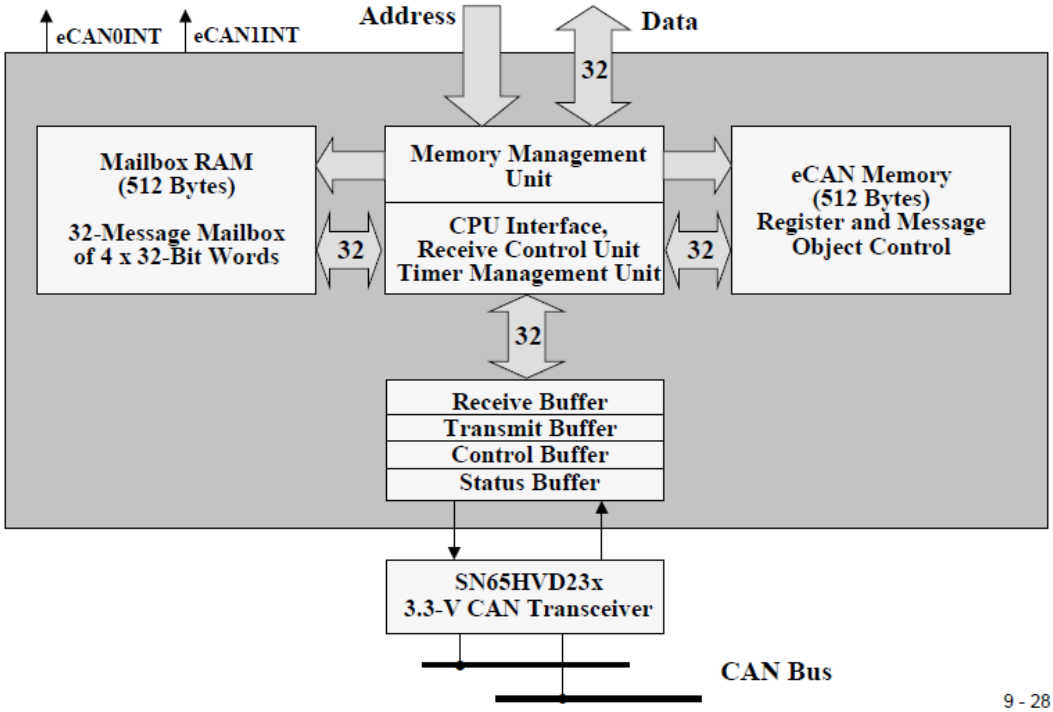


Figura 5.69 Diagrama de bloques del módulo CAN de F2812 [1]

- El modulo CAN en el procesador F2812 es un controlador versión 2.0B (Ver Figura 5.69)
- Soporta velocidades de hasta 1Mbps
- El módulo controlador CAN cuenta con 32 Mailboxes (buzones) para mensajes con tamaños de datos de 0 a 8 bytes.
  - ✓ Configurables para transmitir o recibir.
  - ✓ Configurables para hacer uso del identificador estándar o el extendido
  - ✓ Un Mailbox soporta una trama de datos o una trama remota
  - ✓ Cada Mailbox dispone de esquema de interrupciones

- ✓ Cada Mailbox dispone de alarmas

➢ El módulo CAN contiene registros que se dividen en 5 grupos. Estos registros se localizan en la memoria de datos, de 0x006200 a 0x0063FF (Ver Figura 5.70). Los 5 grupos son:

- ✓ Registros de Control y Estado
- ✓ Máscaras de aceptación local
- ✓ Marcas de tiempo del mensaje
- ✓ Tiempo de espera del mensaje
- ✓ Mailboxes (buzones)

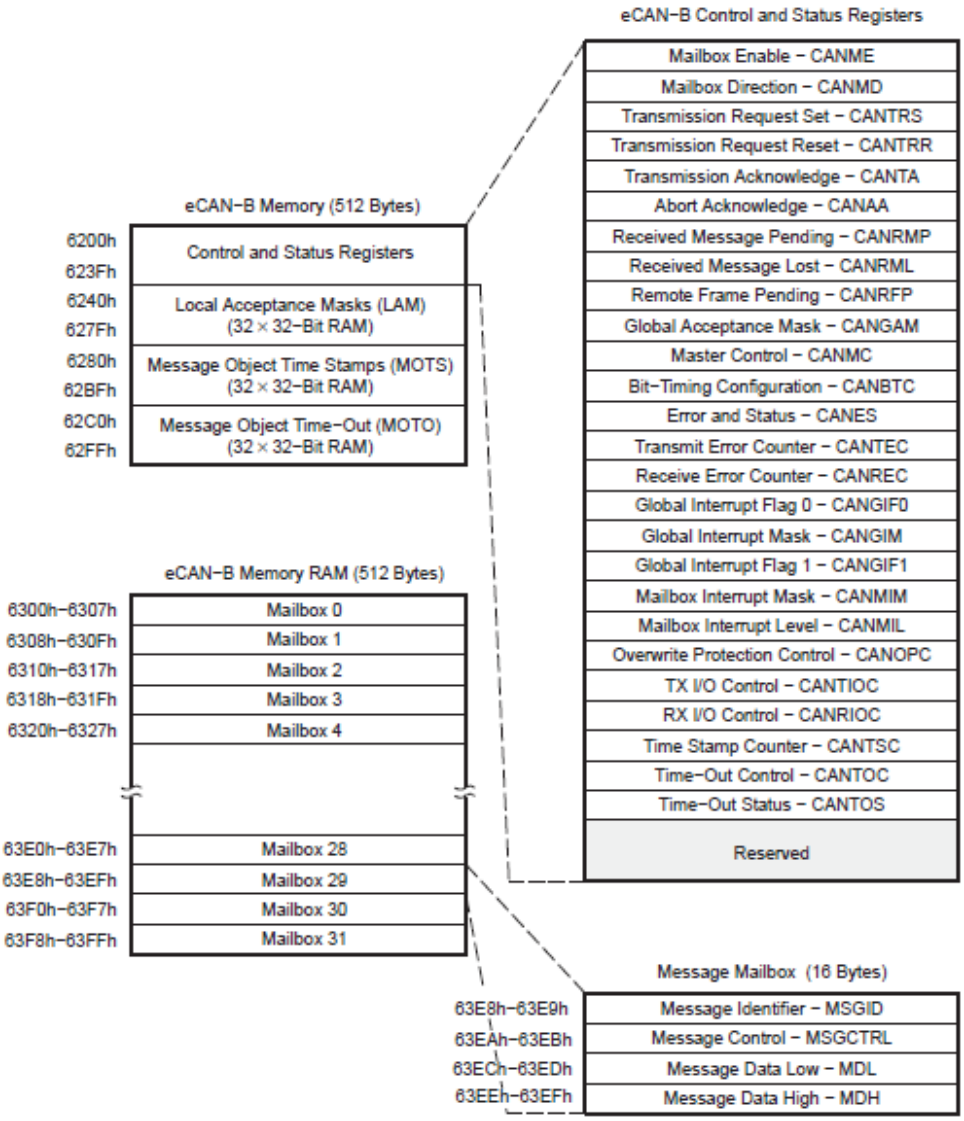


Figura 5.70 Mapa de memoria eCAN 2.0B [13]

REGISTROS MODULO CAN EN F2812

De los 5 grupos de registros que se mencionaron previamente, los de mayor utilidad son los que se encuentran dentro del grupo de Registros de Control y Estado y los Mailboxes.

MAILBOX

Los Mailboxes tienen un espacio en la memoria RAM donde se pueden almacenar después de su recepción o antes de su transmisión

Cada Mailbox contiene:

- ✓ Identificador de mensaje
  - Identificador extendido
  - Identificador estándar
- ✓ Bit de extensión de Identificador, IDE (MSGID.31)
- ✓ Bit habilitar la máscara de aceptación, AME (MSGID.30)
- ✓ Bit modo de respuesta automática, AAM (MSGID.29)
- ✓ Nivel de prioridad de transmisión, TPL (MSGCTRL.12−8)
- ✓ Bit de solicitud de transmisión remota, RTR (MSGCTRL.4)
- ✓ Código de longitud de datos, DLC (MSGCTRL.3−0)
- ✓ Hasta 8 bytes para el campo de datos

Cada Mailbox comprende 4 registros de 32 bits cada uno:

1. MSGID: Almacena la identificación del mensaje.
2. MSGCTRL: Define la cantidad de bits, la prioridad del mensaje y la posible operación como trama remota
3. MDL: 4 bytes de datos menos significativos
4. MDH: 4 bytes de datos más significativos

Registro de Identificación de Mensaje (MSGID)

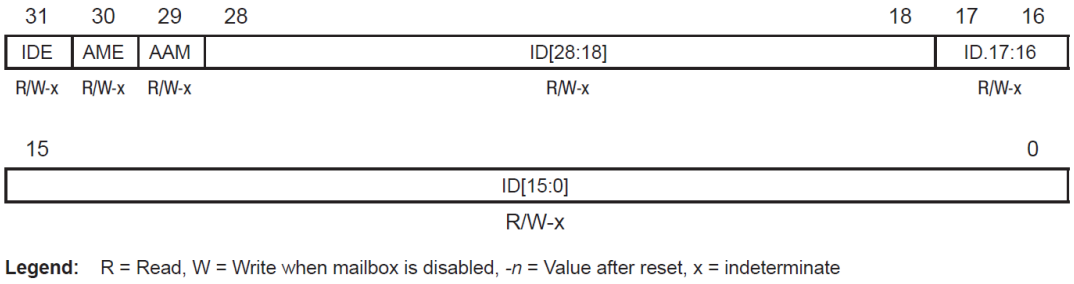


Figura 5.71 Registro de Identificación de Mensaje (MSGID) [13]

En la Tabla 60 se presenta la descripción de los bits mostrados en la Figura 5.71

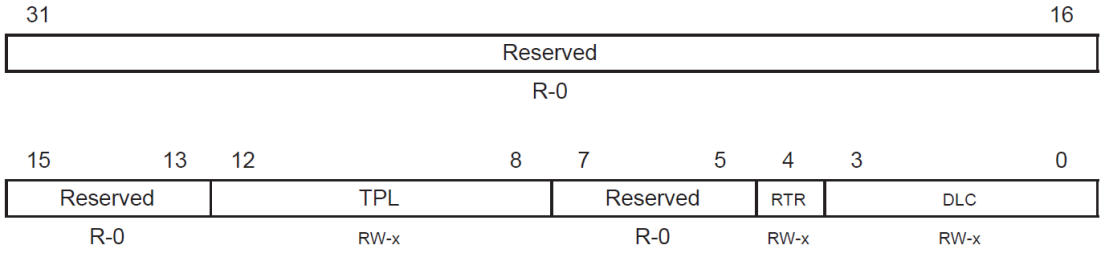
Bit(s)	Nombre	Descripción
31	IDE	Bit de extensión de identificador. Dependerá del ajuste del valor en el bit AMI. Cuando AMI=1: 1 El mensaje recibido tiene identificador extendido 0 El mensaje recibido tiene identificador estándar Cuando AMI=0: 1 El mensaje a ser recibido debe tener identificación extendida 0 El mensaje a ser recibido debe tener identificación estándar
30	AME	Bit habilitar la máscara de aceptación. Es utilizado únicamente por Mailboxes receptores: 1 La máscara de aceptación correspondiente es usada 0 Sin uso de máscara de aceptación.
29	AAM	Bit modo de respuesta automática. Únicamente es válido para Mailboxes que son configurados como trasmisores, en los receptores no tiene efecto alguno: 1 Modo de respuesta automática activado. Si una solicitud remota es recibida, el módulo CAN responde la solicitud enviando el contenido del Mailbox 0 Modo de transmisión normal. Ante una solicitud remota, no se genera respuesta.

Bit(s)	Nombre	Descripción
28-0	ID	Identificación de mensaje - En modo de identificador estándar, la identificación se almacena en los bits 28…18, quedando los bits 17…0 sin significado. - En modo de identificador extendido, la identificación se almacena en todos los bits del campo (28…0).

Tabla 60: Descripción de los bits del registro MSGID [13]

Registro de Control de Mensaje (MSGCTRL)

Como parte del proceso de inicialización del módulo CAN, todos los bits de este registro deben ser inicializados al valor de cero, luego de aquello se podrán configurar los campos con valores deseados.



Legend: RW = Read any time, write when mailbox is disabled or configured for transmission. *-n* = Value after reset, x = indeterminate

Figura 5.72 Registro de Control de Mensaje (MSGCTRL) [13]

En la Tabla 61 se presenta la descripción de los bits mostrados en la Figura 5.72

Bit(s)	Nombre	Descripción
31:13	Reservado	
12:8	TPL	Nivel de prioridad de transmisión. Este campo de 5 bits define la prioridad del mensaje comparándose con el resto de Mailboxes. El mayor número obtendrá la más alta prioridad.
7:5	Reservado	

Bit(s)	Nombre	Descripción
4	SRR/RTR	Bit modo de solicitud de transmisión remota: 1 Para Mailboxes de transmisión: Si la bandera TRS se enciende, la trama remota es transmitida y recibida por el Mailbox de recepción correspondiente. 0 Sin solicitud de trama remota.
3:0	DLC	Código de tamaño de dato. La cantidad especificada en estos bits determinan cuántos bytes serán enviados o recibidos. Valores a partir de 9 hasta 15 no son permitidos.

Tabla 61: Descripción de los bits del registro MSGCTRL [13]

Registro de dato de mensaje (MDL, MDH)

8 bytes son usados para almacenar el campo de datos del mensaje CAN. La configuración de DBO (MC.10) determina el orden del almacenamiento de datos.

La transmisión o recepción de bus CAN, inicia con el Byte 0 del registro.

- Cuando DBO=1, el dato es almacenado o leído empezando con el byte menos significativo del registro CANMDL y termina con el bit más significativo del registro CANMDH. Ver Figuras 5.73 y 5.74.

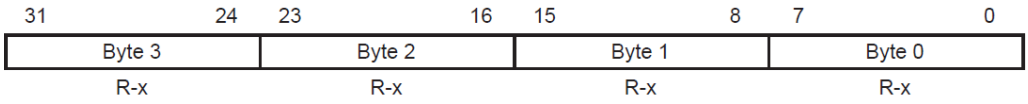


Figura 5.73 Registro MDL con DBO = 1 [13]

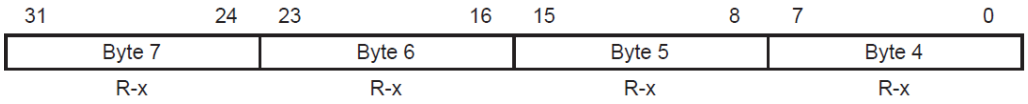


Figura 5.74 Registro MDH con DBO = 1 [13]

- Cuando DBO=0, el dato es almacenado o leído empezando con el byte más significativo del registro CANMDL y termina con el bit menos significativo del registro CANMDH. Ver Figuras 5.75 y 5.76.

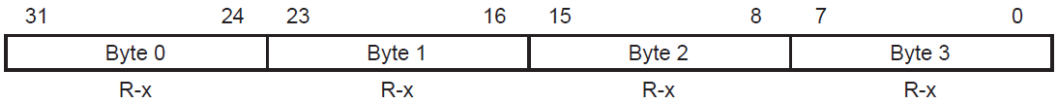


Figura 5.75 Registro MDL con DBO = 0 [13]

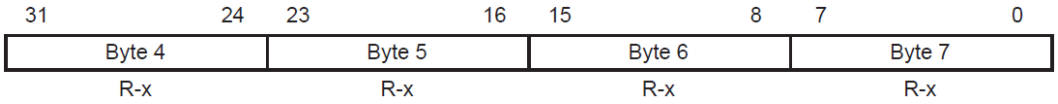


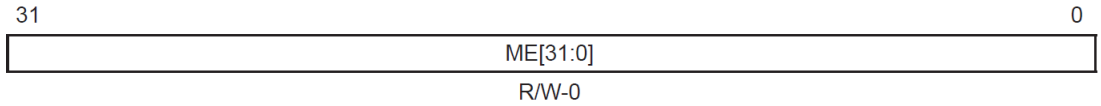
Figura 5.76 Registro MDH con DBO = 0 [13]

Los registros MDL y MDH pueden ser escritos únicamente si son configurados para transmitir o son deshabilitados.

Registros de control y estado

Registro de activación de Mailbox (CANME)

Es usado para habilitar/deshabilitar los Mailboxes de manera individual.



Legend: R/W = Read/Write, *-n* = Value after reset

Figura 5.77 Registro de activación de Mailbox (CANME) [13]

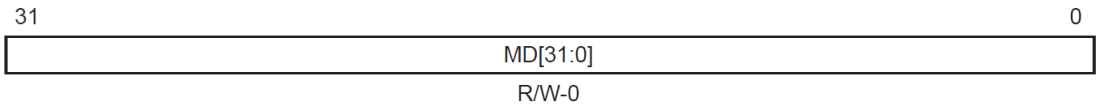
En la Tabla 62 se presenta la descripción de los bits mostrados en la Figura 5.77

Bit(s)	Nombre	Descripción
31:0	ME	Mailbox enable bits. Al encenderse, todos los bits son limpiados. Al deshabilitar los Mailboxes, el CPU puede usar esta memoria como adicional. 1 El correspondiente Mailbox es habilitado por el módulo CAN 0 El espacio de memoria RAM ocupada por el Mailbox queda a disposición del CPU para su uso.

Tabla 62: Descripción de los bits del registro CANME [13]

Registro de dirección de Mailbox (CANMD)

Es usado para configurar la operación de un Mailbox, es decir, transmisor o receptor.



Legend: R/W = Read/Write, *-n* = Value after reset

Figura 5.78 Registro de dirección de Mailbox (CANMD) [13]

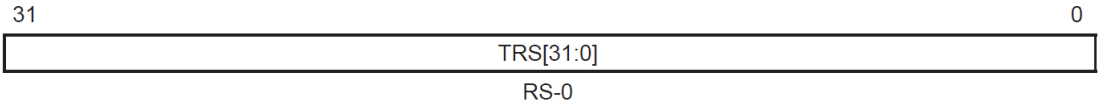
En la Tabla 63 se presenta la descripción de los bits mostrados en la Figura 5.78

Bit(s)	Nombre	Descripción
31:0	MD	Mailbox direction bits. 1 El correspondiente Mailbox es definido como receptor 0 El correspondiente Mailbox es definido como transmisor

Tabla 63: Descripción de los bits del registro CANMD [13]

Registro de ajuste de solicitud de transmisión (CANTRS)

Cuando el Mailbox *n* está listo para ser transmitido, el CPU ajusta el bit TRS[*n*] a 1 para empezar la transmisión. Puede ser ajustado a 1 por el modulo CAN para solicitar una trama remota. Por otra parte, se ajusta se encera cuando una transmisión se lleva a cabo de manera exitosa o es abortada. Si el Mailbox es configurado como receptor, el bit correspondiente del registro CANTRS es ignorado a no ser que sea configurado como manejador de tramas remotas (SRR/RTR=1). Por lo tanto, un Mailbox receptor de esta característica puede enviar una trama remota si su correspondiente bit TRS es ajustado en 1. Una vez que haya enviado la trama remota, el bit TRS se encerará gracias al módulo CAN.



Legend: RS = Read/Set, *-n* = Value after reset

Figura 5.79 Registro de ajuste de solicitud de transmisión (CANTRS) [13]

En la Tabla 64 se presenta la descripción de los bits mostrados en la Figura 5.79

Bit(s)	Nombre	Descripción
31:0	TRS	Transmit-Request-set bits. Normalmente el CPU asigna el valor de 1 en el bit del Mailbox listo a ser transmitido mientras que se encera a través del módulo CAN. 1 El correspondiente Mailbox es definido como receptor 0 Sin efecto

Tabla 64: Descripción de los bits del registro CANTRS [13]

Registro de Reconocimiento de Transmisión (CANTA)

Si el mensaje del Mailbox *n* fue enviado exitosamente, el bit TA[*n*] se ajusta en 1. Escribir 0 no tiene efecto alguno.

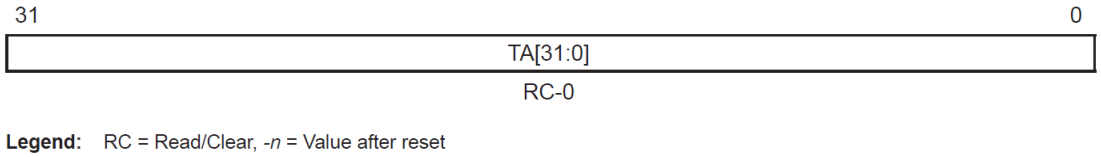


Figura 5.80 Registro de Reconocimiento de Transmisión (CANTA) [13]

En la Tabla 65 se presenta la descripción de los bits mostrados en la Figura 5.80

Bit(s)	Nombre	Descripción
31:0	TA	Transmit-acknowledge bits. 1 El mensaje del Mailbox *n* fue enviado de manera exitosa 0 Sin efecto

Tabla 65: Descripción de los bits del registro CANTA [13]

Registro de Mensaje Recibido Pendiente (CANRMP)

Si el Mailbox *n* contiene un mensaje recibido, el bit RMP[*n*] se ajusta en 1. Los bits del registro pueden ser reseteados únicamente por el CPU y ajustados en 1 por la lógica del módulo CAN.

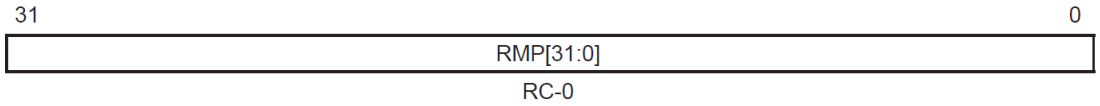


Figura 5.81 Registro de Mensaje Recibido Pendiente (CANRMP) [13]

En la Tabla 66 se presenta la descripción de los bits mostrados en la Figura 5.81

Bit(s)	Nombre	Descripción
31:0	RMP	Received-message bits. 1 El Mailbox *n* contiene un mensaje recibido. 0 El Mailbox no contiene ningún mensaje

Tabla 66: Descripción de los bits del registro CANRMP [13]

Registro de Trama Remota Pendiente (CANRFP)

Siempre que una solicitud de trama remota es recibida por el módulo CAN (normalmente es recibida por un Mailbox Transmisor), el correspondiente bit RFP[*n*] del registro de trama remota pendiente se ajusta en 1. Si una trama remota es almacenada en un Mailbox receptor, el correspondiente bit RFP[*n*] del registro de trama remota pendiente permanece en 0.

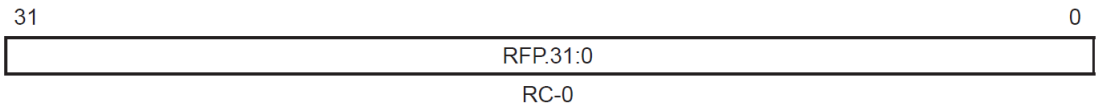


Figura 5.82 Registro de Trama Remota Pendiente (CANRFP) [13]

En la Tabla 67 se presenta la descripción de los bits mostrados en la Figura 5.82

Bit(s)	Nombre	Descripción
31:0	RFP	Remote-frame-pending. 1 Una solicitud de trama remota ha sido recibida por el módulo. 0 Solicitud de trama remota no recibida. Se encera mediante CPU

Tabla 67: Descripción de los bits del registro CANRFP [13]

Registro de Control Maestro (CANMC)

Es usado para controlar los ajustes del módulo CAN. Ciertos bits del registro son protegidos con EALLOW.

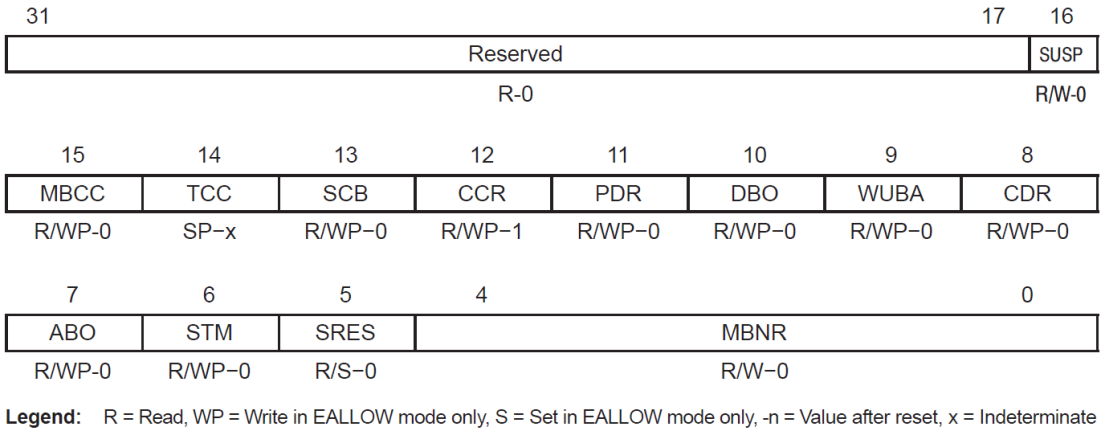


Figura 5.83 Registro de Control Maestro (CANMC) [13]

En la Tabla 68 se presenta la descripción de los bits mostrados en la Figura 5.83

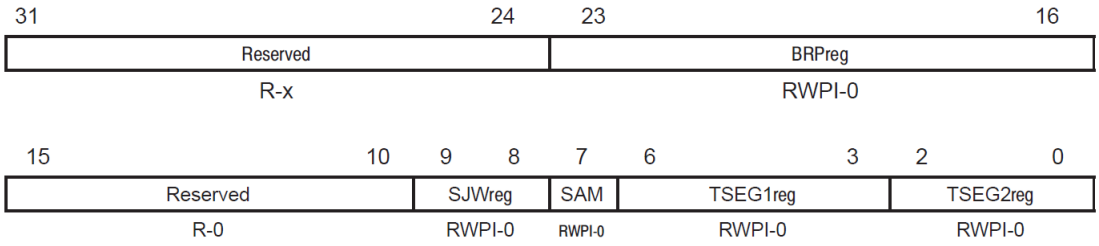
Bit(s)	Nombre	Descripción
31:17	Reservado	Leer no está definido y la escritura no tiene efecto.
16	SUSP	SUSPEND. En modo Suspendido se simula una parada como punto de interrupción o paso a paso. 1 Modo “FREE”. El periférico continúa sus funciones en modo Suspendido. 0 Modo “SOFT”. El periférico se apaga durante modo Suspendido.
15	MBCC	Mailbox timestamp counter clear bit. Este bit es reservado en modo SCC y es protegido por EALLOW. 1 El contador de marcas de tiempo se encera luego de una transmisión o recepción exitosa 0 El contador de marcas de tiempo no será reseteado
14	TCC	Time stamp counter MSB clear bit. Este bit es reservado en modo SCC y es protegido por EALLOW. 1 El bit más significativo del contador de marcas de tiempo se encera. Este bit se resetea luego de un ciclo del reloj interno. 0 El contador de marcas no cambia
13	SCB	SCC compatibility bit. Este bit es reservado en modo SCC y es protegido por EALLOW. 1 Selecciona modo eCAN 0 Solo Mailboxes del 15 al 0 pueden ser usados

12	CCR	Change-configuration request. Este bit es protegido por EALLOW. 1 El CPU solicita acceso de escritura para configurar el registro CANBTC y los registros de máscara de aceptación. 0 El CPU solicita funcionamiento normal.
11	PDR	Power down request. Este bit es limpiado automáticamente por el módulo eCAN tras la activación del modo de bajo consuma de energía. 1 Se solicita el apagado de manera local 0 No se solicita el apagado de manera local (operación normal)
10	DBO	Data byte order. Este bit selecciona el orden de los bytes del campo de datos del mensaje. 1 El dato se recibe o transmite con el bit menos significativo primero 0 El dato se recibe o transmite con el bit menos significativo primero
9	WUBA	Wake up on bus activity. Este bit es protegido por EALLOW 1 El módulo deja el modo de apagado luego de detectar alguna actividad del bus 0 El módulo deja el modo de apagado solo después de escribir 0 en el bit PDR
8	CDR	Change data field request. Este bit permite la actualización rápida del dato de mensaje. 1 El CPU solicita acceso de escritura para el campo de datos del Mailbox especificado. 0 El CPU solicita funcionamiento normal
7	ABO	Auto bus on. Este bit es protegido con EALLOW 1 Luego del estado de bus apagado, el módulo regresa al estado de bus encendido. 0 Sin acción
6	STM	Self-test mode. Este bit es protegido por EALLOW 1 Módulo en modo “self-test”. En este modo, el módulo CAN genera su propia señal de reconocimiento (ACK) 0 Módulo en modo normal
5	SRES	Este bit solo puede ser escrito y siempre es leído como cero. 1 Un acceso de escritura a este registro causa un reseteo de software del módulo. 0 Sin efecto
4:0	MBNR	Mailbox number. 1 El bit MBNR.4 es únicamente para eCAN, y es reservado en SCC 0 Número de Mailbox para el cual el CPU solicitará la escritura al campo de datos. Este bit se utiliza en conjunto con el bit CDR

Tabla 68: Descripción de los bits del registro CANMC [13]

Registro de Configuración Bit-Timing (CANBTC)

Es usado para configurar el nodo CAN con los apropiados parámetros de tiempo de la red. El registro debe ser programado antes de su uso por el módulo CAN.



Legend: RWPI = Read in all modes, write in EALLOW mode during initialization mode only, -*n* = Value after reset

Figura 5.84 Registro de Configuración Bit-Timing (CANBTC) [13]

En la Tabla 69 se presenta la descripción de los bits mostrados en la Figura 5.84

Bit(s)	Nombre	Descripción
31:24	Reservado	Leer no está definido y la escritura no tiene efecto.
23:16	BRPreg	Baud rate prescaler. Este registro define el prescalador para la configuración de la velocidad de transmisión. La longitud de un TQ se define por: $TQ = \frac{1}{SYSCLKOUT}(BRPreg + 1)$ Donde SYSCLKOUT es la frecuencia del sistema de reloj del módulo CAN
15:10	Reservado	
9:8	SJWreg	Synchronization jump width. Este parámetro indica para cuantas unidades de TQ es permitida la longitud cuando es sincronizada. $SJW = SJWreg + 1$
7	SAM	Este parámetro ajusta el número de muestras usadas por el módulo CAN para determinar el nivel actual del bus CAN. 1 El módulo CAN muestrea tres veces y toma la mejor decisión 0 El módulo CAN muestre una vez.
6:3	TSEG1	Time segnment 1. La longitud del bit en el bus CAN es determinada por los parámetros TSEG1, TSEG2 y BRP. Este parámetro especifica la longitud del segmento TSEG1 en unidades TQ. TSEG1 combina segmentos PROP_SEG y PHASE_SEG1: *TSEG1=PROP_SEG+PHASE_SEG1* Donde PROP_SEG y PHASE_SEG1 son longitudes de estos dos segmentos en unidades TQ.

Bit(s)	Nombre	Descripción
2:0	TSEG2reg	Time Segment 2. TSEG2 define la longitud del segmento PHASE_SEG2 en unidades TQ. TSEG2 es programable en el rango de 1TQ a 8 TQ. TSEG2 debe ser tan pequeña o igual a TSEG1.

Tabla 69: Descripción de los bits del registro CANBTC [13]

Registro de Estados y Error (CANES)

El estado del módulo CAN se muestra en el registro CANES y en los registros de conteo de errores.

El registro CANES comprende información sobre el estado actual de los indicadores de error del módulo CAN y muestra banderas de error del bus.

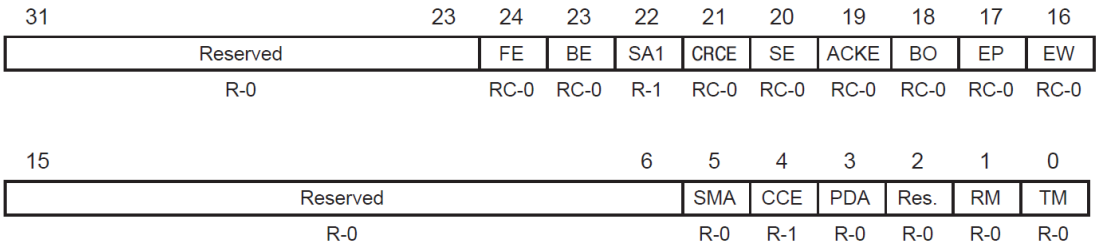


Figura 5.85 Registro de Estados y Error (CANES) [13]

En la Tabla 70 se presenta la descripción de los bits mostrados en la Figura 5.85

Bit(s)	Nombre	Descripción
31:25	Reservado	Leer no está definido y la escritura no tiene efecto.
24	FE	Form error flag. 1 Un error de forma ha ocurrido en el bus. Quiere decir que, uno o más campos de error de forma tiene un nivel equivocado de bus. 0 No se ha detectado errores de forma.

Bit(s)	Nombre	Descripción
23	BE	Bit error flag. 1 El bit recibido no coincide con el bit transmitido fuera del campo de arbitraje o durante la transmisión del campo de arbitraje, es decir, se envió un bit dominante, pero se ha recibido un bit recesivo. 0 No se ha detectado bit de error
22	SA1	Stuck at dominant error. Siempre está en 1 después de un reinicio de hardware, un restablecimiento de software, o una condición de inactividad del bus. Se limpia cuando se detecta en un bit recesivo. 1 El módulo CAN nunca detectó un bit recesivo 0 El módulo CAN detectó un bit recesivo
21	CRCE	CRC error. 1 El módulo recibió un CRC incorrecto 0 El módulo recibió nunca recibió un CRC incorrecto
20	SE	Stuff error. 1 Se ha producido error de bit de relleno 0 No hay error de bit de relleno
19	ACKE	Acknowledge error. 1 El módulo CAN ha recibido mensajes no reconocidos 0 Todos los mensajes se han reconocido correctamente
18	BO	Bus-off status. El módulo CAN viene a este estado. 1 Indica la existencia de una tasa anormal de errores en el bus CAN 0 Operación normal
17	EP	Error-passive state. 1 El módulo CAN está en modo error-pasivo. CANTEC ha alcanzado el valor de 128 0 El módulo CAN está en modo error-activo
16	EW	Warning status. 1 Uno de los dos contadores de error ha alcanzado el nivel de advertencia de 96 0 Los valores de ambos contadores están por debajo de 96
15:6	Reservado	Leer no está definido y la escritura no tiene efecto.
5	SMA	Suspend mode acknowledge. 1 El módulo ha entrado en modo Suspendido 0 El modulo no entra en modo Suspendido
4	CCE	Change configuration enable. Muestra la configuración de acceso. 1 El CPU tiene acceso a escritura para configuración de registros 0 El CPU tiene denegado el acceso a escritura para configuración de registros

Bit(s)	Nombre	Descripción
3	PDA	Change configuration enable. Muestra la configuración de acceso. 1 El módulo CAN ha entrado a modo apagado 0 Operación normal
2	Reservado	Leer no está definido y la escritura no tiene efecto.
1	RM	Change configuration enable. Muestra la configuración de acceso. 1 El módulo CAN está recibiendo un mensaje. 0 El módulo CAN no está recibiendo un mensaje.
0	TM	Transmit mode. El módulo CAN está en modo transmisión. 1 El módulo CAN está transmitiendo un mensaje. 0 El módulo CAN no está transmitiendo un mensaje.

Tabla 70: Descripción de los bits del registro CANES [13]

Registro de Control I/O (CANTIOC, CANRIOC)

Los pines CANTX y CANRX deben ser configurados para ser usados por CAN. Esto se realiza usando los registros CANTIOC y CANRIOC.

Registro de Control TX I/O (CANTIOC)

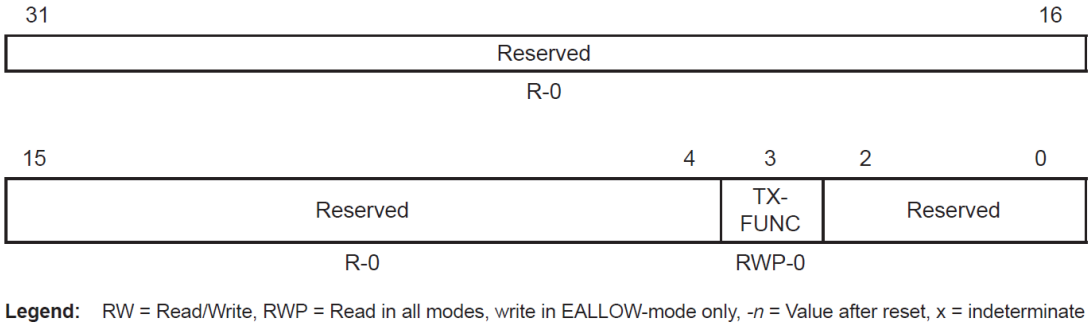


Figura 5.86 Registro de Control TX I/O (CANTIOC) [13]

En la Tabla 71 se presenta la descripción de los bits mostrados en la Figura 5.86

Bit(s)	Nombre	Descripción
31:4	Reservado	Leer no está definido y la escritura no tiene efecto.
3	TXFUNC	Este bit debe ser ajustado por el módulo CAN. 1 El pin CANTX para funciones de transmisión CAN 0 Reservado
2:0	Reservado	Reservado

Tabla 71: Descripción de los bits del registro CANTIOC [13]

Registro de Control RX I/O (CANRIOC)

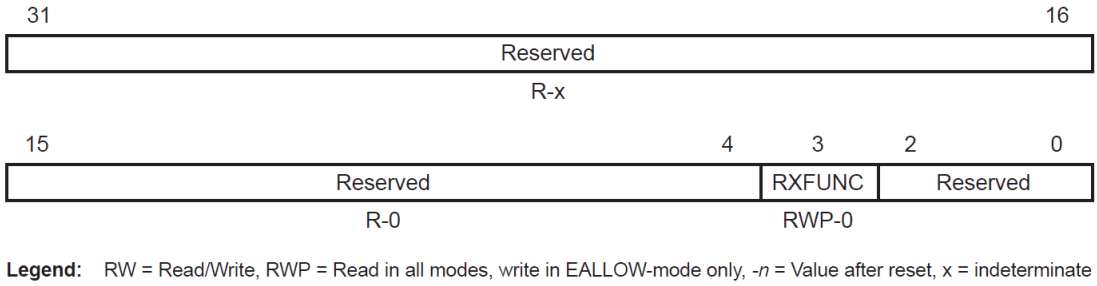


Figura 5.87 Registro de Control RX I/O (CANRIOC) [13]

En la Tabla 72 se presenta la descripción de los bits mostrados en la Figura 5.87

Bit(s)	Nombre	Descripción
31:4	Reservado	Leer no está definido y la escritura no tiene efecto.
3	RXFUNC	Este bit debe ser ajustado por el módulo CAN. 1 El pin CANRX para funciones de recepción CAN 0 Reservado
2:0	Reservado	Reservado

Tabla 72: Descripción de los bits del registro CANRIOC [13]

AJUSTES DE COMPORTAMIENTO DE TIPO DE MENSAJE

En la Tabla 73 se puede apreciar la configuración para establecer al Mailbox como uno de los 4 tipos de mensaje:

Comportamiento de tipo de mensaje	CANMD	AAM	SRR
Mensaje de transmisión	0	0	0
Mensaje de recepción	1	0	0
Mensaje de solicitud	1	0	1
Mensaje de respuesta	0	1	0

Tabla 73: Ajustes de comportamiento de tipo de mensaje CAN [13]

Los Mailboxes configurados como mensajes de transmisión y transmisión son usados para intercambiar datos entre el transmisor y múltiples receptores. Por otro lado, los que son configurados como mensaje de respuesta y de solicitud, son usados para crear enlaces de comunicación uno a uno.