UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"DISEÑO DE UN COPROCESADOR MATEMÁTICO BASADO EN LÓGICA PROGRAMABLE"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELÉCTRICO ELECTRÓNICO

(MÓDULO ELECTRÓNICA)

PRESENTA:

ANDRÉS VIVEROS WACHER

Director de Tesis:

M.I. MIGUEL ÁNGEL BAÑUELOS SAUCEDO

CIUDAD UNIVERSITARIA, MÉXICO, D.F. 2010

Agradecimientos

A mis padres, por impulsar mi sed de conocimiento y por su gran apoyo a lo largo de la carrera y de mi vida.

A mi familia, por su ayuda en toda la carrera.

A la Facultad de Ingeniería, por la formación que me otorgó.

Al M.I. Miguel Ángel Bañuelos Saucedo, por permitirme trabajar en el laboratorio de electrónica del CCADET y hacer posible este trabajo.

A mis amigos de la escuela: Carmen, Oso, Boni, Akira, Memo, Sophia, Manolo, Tico, Farid, Fer, Abn y un gran etcétera.

A mis amigos de la facultad: Fercho, Kenji, Arturo, César, Guillermo, Quiroz, Nando, Eros, Armi, Steph, Mariana, Quike y todos con los que viví grandes momentos dentro y fuera de la facultad.

A mis amigos de la Facultad de Química, por su valiosa amistad y apoyo a lo largo de la carrera.

A Pamela, por ayudarme en mis momentos de desesperación con este proyecto y por tu apoyo incondicional.

A ti, por interesarte en leer este trabajo, aunque sea esta página.

Índice

Abreviaciones

CPU Unidad Central de Procesamiento (*Central Processing Unit*)

DSP Procesador Digital de Señales (*Digital Signal Processor*)

HDL Lenguaje de Descripción de Hardware (*Hardware Description Language*)

PLD Dispositivo Lógico Programable (*Programmable Logic Device*)

SPLD Dispositivo Lógico Programable Simple (*Simple Programmable Logic Device*)

CPLD Dispositivo Lógico Programable Complejo (*Complex Programmable Logic Device*)

FPGA Matriz de Compuertas Programable en Campo (*Field Programmable Gate Array*)

ROM Memoria de solo lectura *(Read Only Memory*)

PAL Lógica de Matrices Programables (*Programmable Array Logic*)

PLA Matriz Lógica Programable (*Programmable Logic Array*)

GAL Lógica de Matrices Genérica (*Generic Array Logic*)

LUT Tabla de Consulta (*Look up table*)

PIC Controlador de Interfaz Periférico (*Peripheral Interface Controller*)

VHDL Lenguaje de Descripción de Hardware de Circuitos Integrados de Velocidad Muy Alta (*Very High Speed Integrated Circuit Hardware Description Language)*

MPGA Matriz de Compuertas programables por Máscara (*Mask Programmable Gate Array)*

ASIC Circuito Integrado de Aplicación Específica (*Application Specific Integrated Circuit*)

CAD Diseño Asistido por Computadora (*Computer Aided Design*)

LAB Bloque de Matrices Lógico (*Logic Array Block*)

PIA Matriz de Interconexiones Programable (*Programmable Interconnect Array*)

LSI Gran Escala de Integración (*Large Scale of Integration*)

PC Contador de programa (*Program Counter*)

SP Apuntador de Stack (*Stack Pointer*)

BSR Registro de Selección de Banco (*Bank Select Register*)

LCD Display de Cristal Líquido (*Liquid Crystal Display*)

IDE Ambiente de Desarrollo Integrado (*Integrated Development Environment*)

PCB Tarjeta de Circuito Impreso (*Printed Circuit Board*)

Introducción

La multiplicación es una de las operaciones básicas en aplicaciones de procesado digital de señales y control digital. Es habitual que en la realización práctica de este tipo de aplicaciones se utilicen microprocesadores (µP), microcontroladores (µC) o procesadores digitales de señales (DSP) que incorporan la instrucción de multiplicación. Cuando las exigencias de la aplicación requieren una velocidad de cálculo superior a la que puede dar este tipo de componentes, es necesario utilizar dispositivos de lógica programable en los que se implementarán directamente los circuitos hardware que realizan las operaciones necesarias por el proceso.

En el presente trabajo se explica el desarrollo de multiplicadores de alta velocidad a implementarse en dispositivos lógicos programables, haciendo uso de VHDL para su descripción. Así mismo, se hace una investigación y un estudio comparativo de las distintas arquitecturas y algoritmos de multiplicación rápida aplicables a PLDs para escoger las mejores opciones dentro de éstos.

Puesto que el desarrollo del multiplicador está pensado para ser un coprocesador, en este proyecto también se hace un desarrollo de una interfaz con un microcontrolador PIC para mostrar la comunicación entre los dos dispositivos y demostrar el funcionamiento correcto del multiplicador.

Justificación

Las multiplicaciones se emplean en prácticamente cualquier procesamiento digital de una señal. En la actualidad, las aplicaciones en tiempo real requieren de procesamientos muy veloces, por lo que se requieren mutiplicadores de alta velocidad. El presente trabajo pretende exponer un coprocesador que realice multiplicaciones a velocidades muy altas, comunicando tales operaciones con el procesador principal, que en este caso se trata de un microcontrolador PIC.

Se tiene como objetivo diseñar y desarrollar un multiplicador de alta velocidad implementado en un dispositivo de lógica programable que logre actuar como coprocesador para un microcontrolador.

Así mismo, se diseña y desarrolla un circuito que sirva para demostrar el funcionamiento correcto del multiplicador, para lo cual se debe incluir un microcontrolador PIC.

Antecedentes

El procesador es un componente electrónico que interpreta instrucciones y procesa los datos contenidos en los programas. Son conocidos también como CPU (Central Processing Unit) y es uno de los componentes necesarios dentro de una computadora, junto con la unidad de memoria y los dispositivos de entrada/salida.

Los primeros procesadores fueron diseñados a la medida, como parte de una computadora más grande, sin embargo este método de diseño para una aplicación en particular ha desaparecido y se ha sustituido por el desarrollo de procesadores baratos y estandarizados adaptados para uno o muchos propósitos. La popularización de los circuitos integrados permitió el diseño y fabricación de procesadores más complejos y en espacios menores.

Desde la aparición en la década de los ‘70s de los microprocesadores de Intel, este tipo de procesadores tomó el control del mercado, desplazando casi totalmente al resto de las alternativas de implementación de una unidad central de proceso en esa época. Actualmente compañías como AMD han ganado bastante mercado en este rubro.

Los microprocesadores deben cumplir con ciertas capacidades, la primera es leer y escribir información en la memoria. Esto es decisivo ya que las instrucciones del programa que ejecuta el microprocesador y los datos sobre los cuales trabaja están almacenados en esa memoria. La otra capacidad es reconocer y ejecutar una serie de comandos o instrucciones proporcionados por los programas. La tercera capacidad es decirle a otras partes del sistema lo que deben de hacer, para que el procesador pueda dirigir la operación a realizar. En pocas palabras los circuitos de control del microprocesador tienen la función de decodificar y ejecutar

el programa, que en sí es un conjunto de instrucciones para el procesamiento de los datos.

Por otro lado, un coprocesador es un procesador utilizado como suplemento de las funciones principales del procesador principal. Las operaciones ejecutadas por un coprocesador pueden ser aritméticas, procesamiento gráfico, procesamiento de señales, criptografía, entre muchas otras.

La principal razón para construir coprocesadores es para evitar que el procesador principal realice tareas intensivas, acelerando así el rendimiento del sistema. Debido a que los coprocesadores son más especializados, las tareas para las que son diseñados se realizan más eficientemente.

Los coprocesadores no son procesadores de propósito general, requieren de un procesador principal que lea las respuestas del coprocesador y maneje todas las operaciones junto con las funciones del procesador.

Actualmente los coprocesadores se pueden implementar principalmente de tres formas, dependiendo de su fin:

- Empleando un microprocesador adicional que cumpla con los requerimientos del coprocesador que se desea diseñar, que se comunique con el procesador principal y siga un programa preestablecido para poder llevar a cabo las tareas para las cuales es requerido.
- Por medio de un DSP (*Digital Signal Processor*), en el caso de tareas que requieran una velocidad bastante alta, o un nivel de procesamiento matemático bastante elevado.
- Usando lógica programable. En este caso, se pueden construir coprocesadores a la medida y hechos para casos específicos de acuerdo a las necesidades del sistema.

Lógica programable y HDLs

Al escoger lógica programable como base para implementar un coprocesador, uno tiene mayor libertad sobre el número de entradas y salidas, líneas de control y cualquier puerto que se vaya a ocupar para comunicarse con el procesador principal o con otros elementos externos que se requieran. Esta libertad, comparada con microprocesadores, es mayor ya que en los µPs se tiene preestablecido el número de puertos de entrada/salida y el número de bits en cada puerto. No hay que olvidar, sin embargo, que en los dispositivos lógicos programables también se tienen limitaciones en cuanto a número de bits por bus, dependiendo de su arquitectura interna, y de los pines disponibles ya que siempre se tienen algunos preestablecidos.

Ahora, existen diferentes formas de diseño para implementar un coprocesador mediante lógica programable; la más arcaica y de un bajo nivel de abstracción sería mediante diagramas lógicos, empleando la herramienta software pertinente. El usuario crea su esquemático, o diagrama RTL empleando desde compuertas básicas hasta bloques parametrizados y realizando las conexiones pertinentes se puede obtener el sistema requerido, sin embargo dicha captura no es estandarizada, por lo que no existe posibilidad de portabilidad. En otras palabras, no es 100% seguro el traslado del diseño de un dispositivo de un fabricante a otro.

Por esta razón, se crearon los HDLs (Hardware Description Languages), que permiten documentar interconexiones y el comportamiento de un circuito sin necesidad de utilizar diagramas esquemáticos.

El flujo de diseño empleando HDLs suele ser típico:

1. Definir las tareas que tiene que hacer el circuito
2. Escribir el programa usando un HDL.
3. Comprobación de sintaxis y simulación de la descripción.

4. Programación del dispositivo y comprobación del funcionamiento.

Los rasgos comunes entre todos los HDLs suelen ser la independencia del hardware y la modularidad o jerarquía. Esta última se refiere a que una vez hecho un diseño, éste se puede usar dentro de otro diseño más complicado y con otro dispositivo compatible. Por ejemplo, en el caso del objetivo de esta tesis, que se refiere al diseño de un multiplicador, éste puede ser usado para un diseño más complejo que involucre cálculos en los que se requiera una multiplicación.

Actualmente los principales HDLs son SystemC, Verilog y VHDL. El primero está basado en un conjunto de librerías implementadas en C++. Los otros dos son lenguajes de descripción de hardware de alto nivel y son populares para dispositivos complejos, sin embargo, pueden ser empleados también para diseños de baja complejidad.

Verilog es un lenguaje de descripción de hardware (HDL), desarrollado más o menos al mismo tiempo que VHDL, por Gateway Design Automation y se convirtió en un estándar IEEE en 1995. Las bases sintácticas de VHDL están basadas en ADA (lenguaje de programación orientada a objetos), mientras que las de Verilog están basadas en C. Debido a esto, algunos consideran a Verilog más fácil de aprender (o menos intimidante). VHDL y Verilog mantienen más o menos el 50% del mercado cada una. Ambos lenguajes pueden satisfacer los requerimientos de diseño digital. Si se conoce alguno de los dos lenguajes, no es difícil hacer la transición de uno a otro.

Por estas razones, se empleó en este proyecto el VHDL como lenguaje para describir el coprocesador dentro de un dispositivo de lógica programable. En los capítulos siguientes se hablará sobre el VHDL, así como de dispositivos lógicos programables, desde los más básicos hasta algunos más complejos. Posteriormente se hablará sobre microprocesadores, en especial de PICs, ya que en este proyecto se empleará uno de ellos como procesador principal del sistema.

Capítulo 1. Lenguaje VHDL

En este capítulo se abordan la definición y las principales características del lenguaje VHDL.

1.1 Definición de VHDL

VHDL es el acrónimo de VHSIC (*Very High Speed Integrated Circuit) Hardware Description Language*. VHDL es un lenguaje de descripción y modelado diseñado para describir la funcionalidad y la organización de sistemas *hardware* digitales.

VHDL fue desarrollado como un lenguaje para el modelado y simulación lógica dirigida por eventos de sistemas digitales, y actualmente se utiliza también para la síntesis automática de circuitos.

El VHDL es un lenguaje que fue diseñado inicialmente para ser usado en el modelado de sistemas digitales. Es por esta razón que su utilización en síntesis no es inmediata, aunque lo cierto es que la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

La síntesis a partir de VHDL constituye hoy en día una de las principales aplicaciones del lenguaje con una gran demanda de uso. Las herramientas de síntesis basadas en el lenguaje permiten en la actualidad ganancias importantes en la productividad de diseño.

Un lenguaje de descripción de hardware permite diseñar y depurar sistemas digitales con un nivel de abstracción más alto que con una captura esquemática de compuertas, flip flops e incluso circuitos MSI.

VHDL fue desarrollado originalmente bajo financiamiento del departamento de defensa de Estados Unidos, para tener un método uniforme para especificar sistemas digitales. El propósito principal del VHDL cuando fue desarrollado, era tener un mecanismo para describir y documentar *hardware* sin ambigüedades. La síntesis del hardware de descripciones de alto nivel no fue uno de los propósitos originales. Desde entonces, el VHDL se ha convertido en un estándar IEEE y se usa bastante en la industria. IEEE creó el estándar VHDL en 1987, el cual fue modificado en 1993, y se hicieron revisiones nuevamente en 2000 y 2002.

1.2 Características del lenguaje VHDL

La característica más importante del VHDL es que, al ser un lenguaje de descripción, si se definen leyes o reglas que simbolicen conexiones, dichas reglas se hacen efectivas al mismo tiempo, a diferencia de otros lenguajes en los cuales la ejecución es en serie, es decir, se ejecuta una instrucción tras otra.

Sin embargo, tiene también la posibilidad de ejecutar algunos bloques de forma serie, haciendo así la descripción en dicho lenguaje más sencilla y con un nivel de abstracción más alto que si se crearan *netlists*.

Capítulo 2 Dispositivos Lógicos Programables

En este capítulo se abordan los distintos tipos de Dispositivos Lógicos Programables, incluyendo su definición, su clasificación y una breve explicación de cada tipo de PLD.

2.1 Definiciones

Un Dispositivo Lógico Programable (PLD) es un componente electrónico usado para construir circuitos digitales reconfigurables. A diferencia de una compuerta lógica que tiene una función fija, los PLDs salen de fábrica sin una función en específico, por lo tanto necesitan ser programados o reconfigurados antes de poder ser usados.

Los PLDs tienen varias ventajas. La primera es la habilidad de integración, que permite integrar una gran cantidad de funcionalidad en un solo chip. Los PLDs eliminan el uso de múltiples chips así como la inconveniencia y desconfianza de usar cableado externo. La segunda ventaja es el hecho de poder cambiar el diseño. Muchos PLDs permiten ser reprogramados o reconfigurados.

Existen dos ramas principales dentro de los dispositivos lógicos programables, la lógica programable de campo y la de fábrica. El término campo en este contexto implica que los dispositivos puedan ser programados en el “campo” del usuario, mientras que la lógica de fábrica puede ser programada en la misma fábrica donde se construyen, de acuerdo a los requerimientos del cliente. En este caso, la tecnología de programación usa procesos irreversibles, por lo que solo es posible hacerlo una vez.

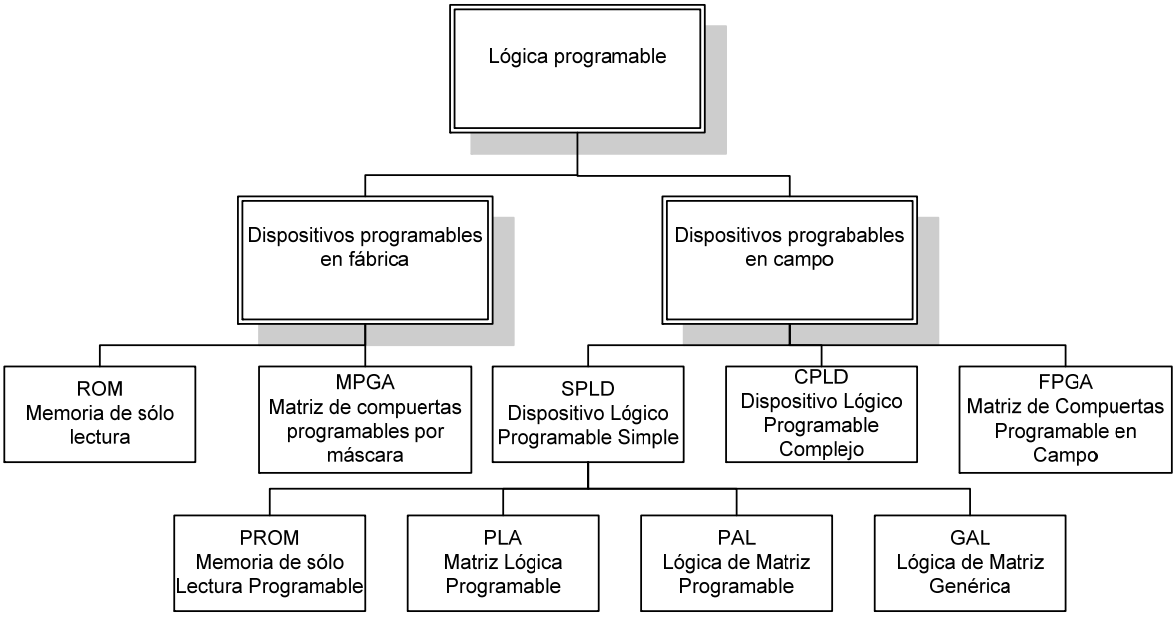


Figura 2.1 Árbol de clasificación de PLDs.

Algunos ejemplos de lógica programable de fábrica son los MPGAs y memorias de sólo lectura (ROMs). Las primeras generaciones de muchos dispositivos programables también fueron programados únicamente en fábrica. Las ROMs son consideradas como lógica programable porque, aunque fueron concebidas como unidades de memoria, también sirven para implementar cualquier circuitería combinacional. Los MPGAs son arreglos de compuertas tradicionales que requieren una máscara para ser diseñados. Los MPGAs son también llamados simplemente gate arrays y han sido la tecnología popular para crear ASICs (*Application Specific Integrated Circuits*).

La lógica programable por el usuario basada en compuertas AND y OR fue desarrollada al inicio de la década de los ‘70s. Para 1972-73 ya estaban disponibles arreglos lógicos programables una sola vez que permitían personalizaciones instantáneas para diseñadores. Algunos se refieren a estos circuitos como FPLAs (*Field Programmable Logic Arrays*).

Monolithic Memories Inc. (MMI), una compañía comprada por Advanced Micro Devices (AMD) creó circuitos integrados llamados PLAs (*Programmable Logic Arrays*) que podían tener el mismo rendimiento y funcionalidad que 5 a 20 chips comerciales. Un dispositivo similar es el PAL (*Programmable Array Logic*).

PALs y PLAs contienen arreglos de compuertas. En el PLA se tiene un arreglo de compuertas AND programable y un arreglo de compuertas OR programable, permitiendo a los usuarios implementar funciones combinacionales en dos niveles de compuertas. El PAL es un caso especial del PLA, ya que el arreglo de ORs es fijo y el único arreglo programable es el de compuertas AND. Muchas PALs también contienen flip flops.

Los primeros dispositivos programables permitían ser programados una sola vez. El siguiente avance tecnológico fue el poder borrar los dispositivos, lo cual al inicio se hacía por medio de luz ultravioleta, que significaba remover el dispositivo del circuito para ponerlo en un ambiente ultravioleta. Este proceso era lento (de 10 a 15 minutos) y no permitía borrar la información en el circuito. El siguiente avance fue la tecnología de borrado eléctrico, que permitió la creación de dispositivos que se podían borrar rápida y fácilmente así como ser reprogramados sin necesidad de removerlo del circuito.

2.2 SPLDs

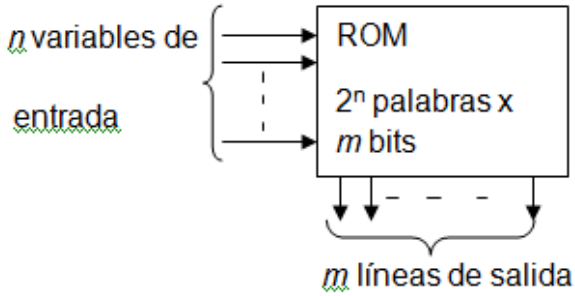
Los PLAs, PALs, GALs y ROMs son llamados SPLDs (*Simple Programmable Logic Devices*) a partir del surgimiento de los CPLDs (*Complex Programmable Logic Devices*) los cuales básicamente contienen múltiples PLDs en el mismo chip.

En esta sección se hablará brevemente sobre los principales SPLDs.

2.2.1 ROM

Una ROM consiste en un arreglo de dispositivos semiconductores que están interconectados para almacenar de datos binarios. Una vez almacenada la información, puede ser leída cuando se requiera, pero no puede ser modificada bajo condiciones normales de operación.

Las ROMs tienen combinaciones de entradas, que generalmente son llamadas direcciones, y patrones de salidas, llamadas palabras. Una ROM que tiene n líneas de entrada y m líneas de salida contiene un arreglo de 2^n palabras, cada una de m bits de longitud. La dirección sirve para seleccionar una de las 2^n palabras, por lo que cuando una combinación de entrada es aplicada a la ROM, el patrón de ceros y unos almacenados en la palabra correspondiente aparece en las líneas de salida.



n variables de entrada	m variables de salida
00...00	100...011
00...01	010...100
00...10	101...010
00...11	011...111
...	...
11...00	001...011
11...01	100...101
11...10	101...011
11...11	111...010

Figura 2.2 Ejemplo de una ROM y su tabla de verdad.

Una ROM consiste básicamente de un decodificador y un arreglo de memoria. Cuando un patrón de entrada se aplica a las entradas del decodificador, una de las 2^n salidas de dicho decodificador se activa, seleccionando una de las palabras almacenadas en la memoria y se transfiere a las líneas de salida.

Los tipos básicos de ROM incluyen ROMs programables por máscara, ROMs programables por el usuario (PROMs), ROMs programables borrables (EPROMs), ROMs programables borrables eléctricamente (EEPROMs) y memorias flash.

En las ROM programables por máscara, el arreglo de datos se almacena permanentemente a la hora de la manufactura. Este proceso resulta caro debido a que se requiere preparar una máscara especial, usada en la fabricación del dispositivo.

Las EPROMs se programan mediante un programador que provee pulsos de voltaje apropiados para almacenar cargas eléctricas en los lugares de memoria y se borran generalmente usando luz ultravioleta, mientras que las EEPROMs se borran por medio de pulsos eléctricos. Las EEPROMs tienen un número limitado de veces que pueden ser borradas y reprogramadas, típicamente entre 100 y 1000 veces. Las memorias flash son similares a las EEPROMs salvo que usan un mecanismo diferente de carga y almacenaje.

Una ROM puede implementar cualquier circuito combinacional. Si las salidas de todas las combinaciones de entradas son almacenadas en la ROM, pueden ser buscadas (“looked up” en inglés) en la tabla de verdad almacenada. Por esto, el método que emplea una ROM es también conocido como *look-up table* (LUT).

2.2.2 PLAs

Un arreglo lógico programable (PLA) realiza la misma función que una ROM. Un PLA con n entradas y m salidas puede realizar m funciones de n variables. La organización interna del PLA difiere de la de la ROM, el decodificador se reemplaza por un arreglo de ANDs que realiza los términos producto seleccionados de las variables de entrada. El arreglo de ORs realiza la operación OR a los términos producto necesarios para formar las funciones de salida.

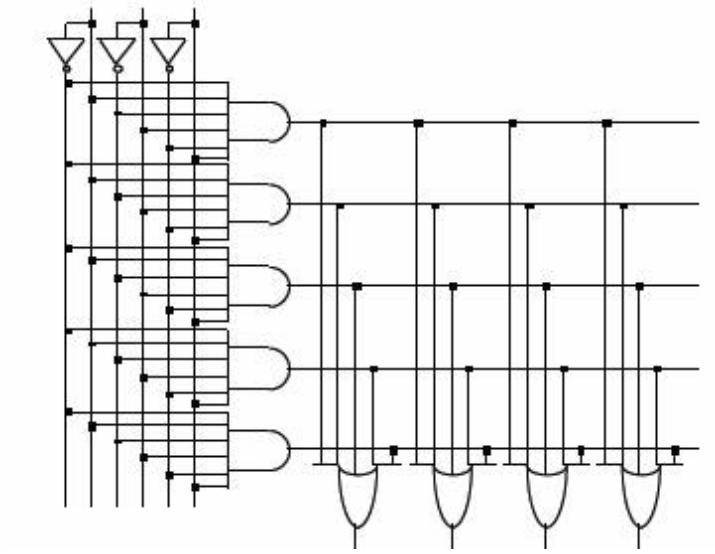


Figura 2.3 Diagrama de un PLA.

Para determinar la información que se graba en un PLA se realiza una tabla para PLA, que es diferente a una tabla de verdad para una ROM. En una tabla de verdad cada fila representa un mintérmino, por lo tanto exactamente una fila se selecciona por cada combinación de valores de entrada, mientras que en cada fila de una tabla para PLA representa un término producto general. Por lo tanto cero, una o más filas se seleccionan por cada combinación de valores de entrada. Para determinar el valor de la función para cierta combinación de entrada, a los valores de la función en las filas seleccionadas de la tabla para PLA se les debe aplicar la operación OR.

término producto	entradas			salidas			
	A	B	C	F0	F1	F2	F3
A'B'	0	0	_	1	0	1	0
AC'	1	_	0	1	1	0	0
B	_	1	_	0	1	0	1
BC'	_	1	0	0	0	1	0
AC	1	_	1	0	0	0	1

Figura 2.4 Ejemplo de una tabla PLA.

2.2.3 PALs

El PAL (Programmable Array Logic) es un caso especial del PLA en el que el arreglo de ANDs es programable y el de ORs es fijo. Sus estructuras son iguales, pero el hecho de que únicamente el arreglo de ANDs sea programable hace más barato y fácil de programar el PAL en comparación con el PLA.

Cuando se diseña con PALs se deben simplificar las ecuaciones lógicas para que quepan en uno (o más) de los PALs existentes. Los términos AND no se pueden compartir entre dos o más compuertas OR, por lo tanto cada función puede ser simplificada por si misma sin importar los otros términos. En cualquier PAL el número de términos AND que alimentan cada compuerta OR es fijo y limitado. Los PALs también pueden contener flip flops D con sus entradas provenientes del arreglo combinacional. Estos se llaman PALs secuenciales. Los PALs fueron desapareciendo con el desarrollo de otros dispositivos, como GALs, CPLDs y FPGAs.

2.2.4 Dispositivos lógicos programables/ Generic Array Logic

Conforme avanzaba la tecnología de circuitos integrados, una gran variedad de dispositivos lógicos programables aparecieron. Los PALs tradicionales no son reprogramables, sin embargo existen ahora PALs borrables y reprogramables con tecnología flash. A veces, a éstos se les llama PLDs.

El 22CEV10 es un PLD con tecnología CMOS borrable eléctricamente que puede ser usado para hacer tanto circuitos combinacionales como secuenciales.

Además de los arreglos AND y OR, la mayoría te los PLDs tienen algún tipo de macrobloque que contiene multiplexores y otros bloques programables adicionales. Estos PLDs se llaman de acuerdo a sus capacidades de entrada/ salida. Por ejemplo, el 22CEV10 tiene 12 pines de entrada más 10 pines que se pueden programar como entrada o salida (22 en total). Contiene también 10 flop flops D y 10 compuertas OR. Cada compuerta OR dirige una macrocelda lógica de salida. Cada macrocelda contiene uno de los 10 *flip flops* D, los cuales comparten un reloj común, un *reset* asícrono de entrada, y un *preset* síncrono de entrada.

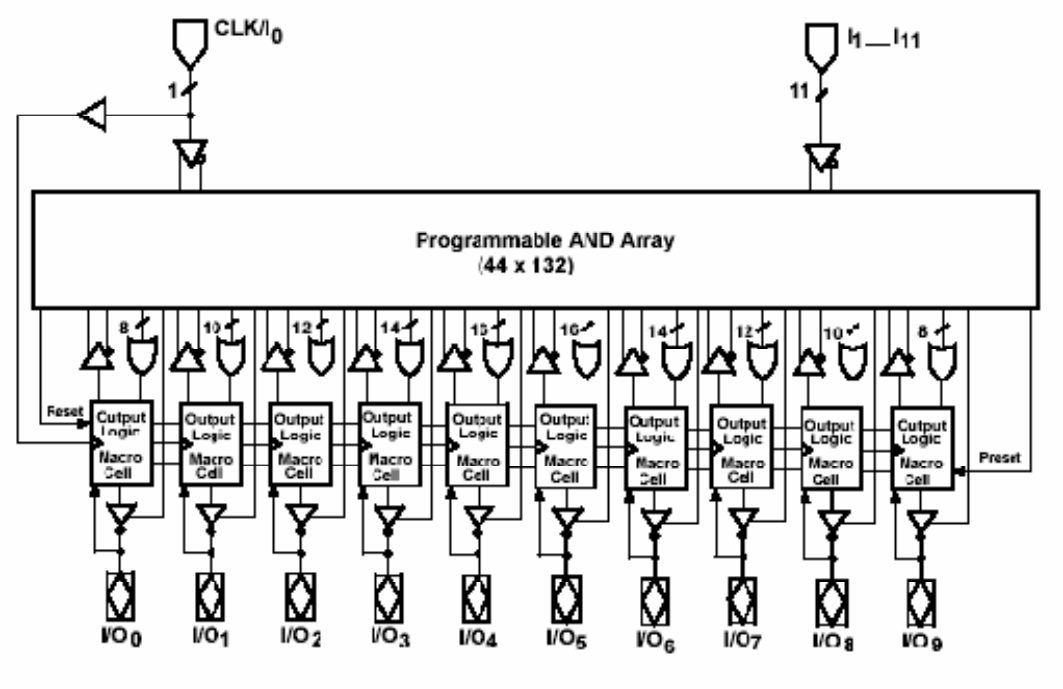


Figura 2.5 Esquema de una GAL 22V10.

La compañía Lattice Semiconductor creó dispositivos similares que tienen la capacidad de ser programados dentro del circuito (*in-circuit programming*) y lo llamó *Generic Array Logic* (GAL). Las GALs son perferctas para implementar pequeñas cantidades de lógica de interfaz. La mayoría de los PLDs, como PALCE22V10, PALCE20V8 entre otros, tienen sus equivalentes en GAL, llamados GAL22V10, GAL20V8, etc.

Existen programas CAD (*Computer Aided Design*) para PALs y PLDs. Estos programas aceptan ecuaciones lógicas, tablas de verdad, gráficas de estados y demás como entrada para generar automáticamente los patrones de bits que se requieren. Posteriormente, un programador puede descargar dichos patrones a los dispositivos para crear las conexiones necesarias. PALASM y ABEL son ejemplos de lenguajes que fueron populares como lenguaje de diseño para PALs y PLDs, aunque en estos días es posible hacer diseños para GALs en lenguajes como VHDL y Verilog.

2.3 CPLDs

Los avances en tecnología han hecho posible la creación de circuitos integrados programables equivalentes a varios PLDs en el mismo chip. A estos circuitos integrados se les llaman dispositivos lógicos programables complejos (CPLDs por sus siglas en inglés).

Un CPLD es un circuito integrado que consiste en un número de bloques lógicos parecidos a un PAL, incluyendo además una matriz programable de interconexiones entre estos bloques.

Algunos CPLDs se basan en la arquitectura del PAL, en cuyo caso cada macrocelda contiene un flip flop y una compuerta OR, cuyas entradas están asociadas a un arreglo de compuertas AND fijo, mientras que los CPLDs que se

basan en PLAs cada salida de compuertas AND en un bloque se puede conectar a la entrada de cualquier compuerta OR en ese bloque.

Los más grandes fabricantes de CPLDs hoy en día son Xilinx, Altera, Lattice Semiconductor, Cypress y Atmel. Algunos de estos vendedores especifican sus productos en términos de cantidad de compuertas, mientras que otros lo hacen en términos de elementos lógicos.

Por ejemplo, Altera vende tres series de CPLDs, las cuales son MAX II, MAX 3000 Y MAX 7000. Cada una de éstas tiene especificaciones en general y los dispositivos dentro de cada serie se diferencian de acuerdo a sus capacidades lógicas y el número de pines de entrada/ salida.

Dentro de la serie MAX 7000 existen dispositivos que van de las 600 compuertas (32 macroceldas) hasta 5000 compuertas (256 macroceldas). Su arquitectura esta basada en módulos de arreglos lógicos, llamados Logia Array Blocks (LABs) que consisten en arreglos de 16 macroceldas. Los LABs se conectan por medio del *Programmable Interconnect Array* (PIA) alimentado por todas las entradas, pines de entrada/salida así como por las macroceldas.

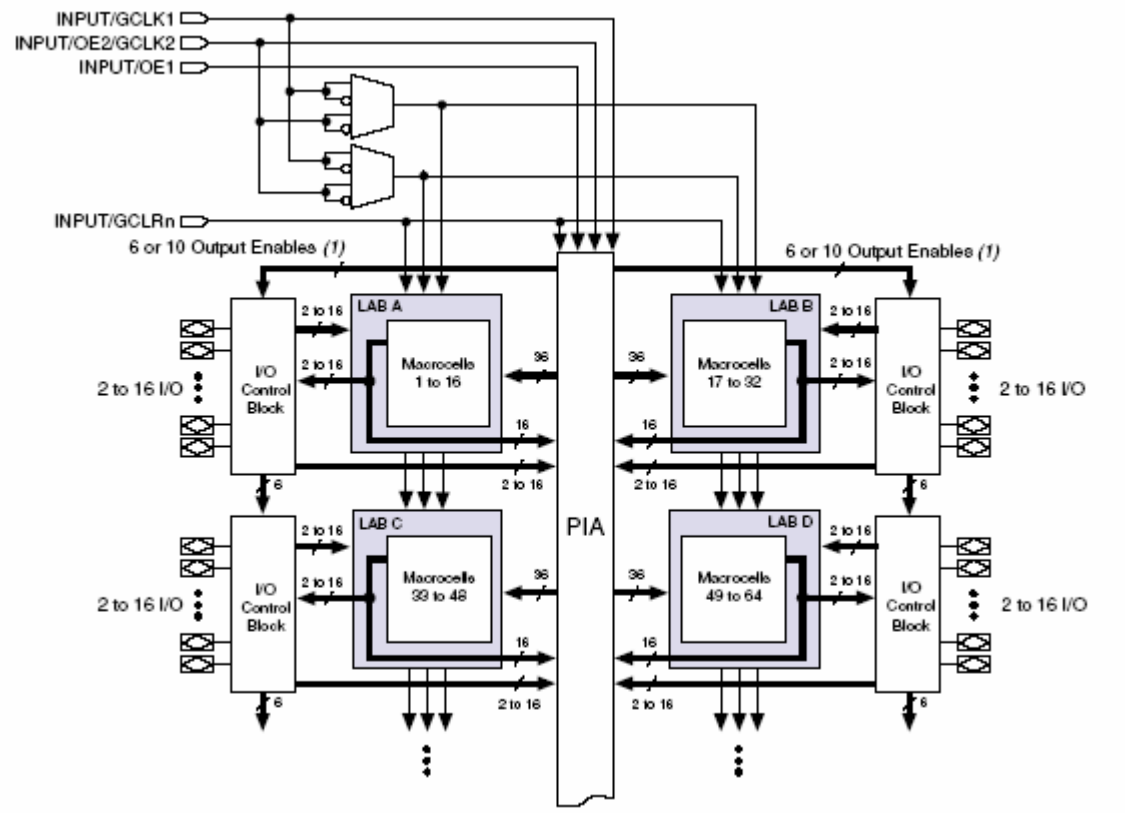


Figura 2.6 Diagrama de bloques del MAX 7000.

Las macroceldas de la serie MAX 7000 pueden ser configuradas individualmente para operar con lógica combinacional o secuencial. Tienen tres bloques funcionales: el arreglo lógico, la matriz de selección de términos producto y el registro programable.

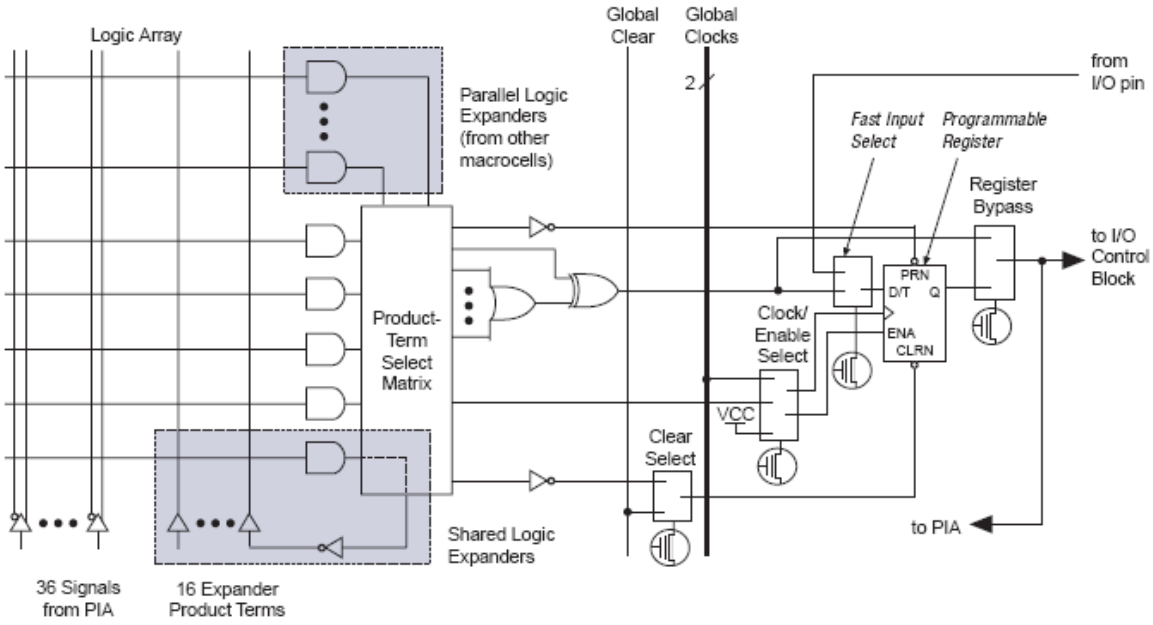


Figura 2.7 Diagrama de una macrocelda de la serie MAX 7000.

La lógica combinacional se implementa en el arreglo lógico que provee cinco términos producto por cada macrocelda. El registro de cada macrocelda se puede configurar para trabajar como *flip flop* D, T, JK o SR con un control de reloj programable. En el caso de tener únicamente lógica combinacional, el *flip flop* se puede ignorar.

2.4 FPGAs

Los FPGAs son circuitos integrados que contienen un arreglo de bloques lógicos idénticos con interconexiones programables, en los que el usuario puede programar tanto las funciones realizadas por cada bloque lógico como las conexiones entre bloques.

Los FPGAs tienen varias ventajas con respecto a MPGAs. Un arreglo de compuertas tradicional puede ser usado para implementar cualquier circuito, pero

sólo se puede programar en fábrica ya que se requiere hacer una máscara específica para un circuito en particular y el tiempo de diseño para un circuito integrado basado en arreglo de compuertas es de algunos meses. Por otro lado, los FPGAs son productos comerciales, el tiempo de manufactura se puede reducir de meses a algunas horas cambiando de MPGAs a FPGAs. De la misma forma, se vuelve más fácil y más barato corregir errores en los diseños. A volúmenes no tan altos, los FPGAs son más baratos que los MPGAs.

El interior de los FPGAs contiene típicamente tres elementos programables: los bloques lógicos, los bloques de entrada/ salida y las interconexiones. Se considera que los bloques de entrada/ salida se encuentran en la periferia del circuito integrado, éstos conectan las señales lógicas a los pines del chip. Los bloques lógicos se encuentran distribuidos dentro del FPGA y el espacio entre ellos se usa para mandar conexiones entre bloques.

La programabilidad de campo se logra por los elementos que pueden ser reconfigurables por el usuario. Los bloques lógicos se crean usando multiplexores, look-up tables y arreglos de compuertas AND-OR o NAND-NAND, y cualquiera de estas cosas puede ser programada (o configurada) por el usuario.

Lo que diferencia un FPGA de un CPLD es la interconexión flexible de propósito general. En un CPLD la interconexión es bastante restringida, mientras que en un FPGA es muy flexible, lo cual a veces puede resultar ser una desventaja ya que mandar una conexión de una parte del chip a otra muy alejada puede hacer el diseño más lento.

Los bloques lógicos entre FPGAs varían en los componentes básicos que emplean. Algunos FPGAs usan bloques basados en LUTs, mientras que otros usan multiplexores y compuertas lógicas. Existen también bloques lógicos que simplemente consistían en pares de transistores. En los primeros FPGAs de Altera, los bloques consistían en PLDs.

Los bloques lógicos también varían en el tamaño. Algunos FPGAs usan bloques básicos grandes, capaces de implementar varias funciones de cuatro o cinco variables, con algunos flip flops. En contraste, también existen FPGAs con bloques que sólo permiten una función de tres variables y un flip flop en cada bloque. Los distintos fabricantes usan nombres diferentes para sus bloques, por ejemplo, en Xilinx un bloque lógico programable se llama *Configurable Logic Block;* Altera los llama *Logic Elements* (LEs) y una colección de ocho o diez de ellos se llama *Logic Array Block* (LAB).

Un elemento importante en los FPGAs es la interconexión programable entre bloques lógicos. Existen diferentes tipos de conexiones en FPGAs comerciales. Algunos usan matrices de *switches*, en las que hay un *switch* en cada intersección de “cables”. Una matriz de *switches* soporta cualquier conexión entre cable y cable, pero resulta muy cara esta tecnología, además de que no todas las conexiones pueden existir al mismo tiempo.

Otros FPGAs usan conexiones especiales entre bloques lógicos adyacentes. Este tipo de conexiones son rápidas porque no necesitan pasar por una matriz de ruteo. En este tipo de FPGAs las interconexiones directas se dan hacia los cuatro bloques vecinos (arriba, abajo, izquierda y derecha), en otros casos se dan hacia los ocho vecinos, incluyendo así a los diagonales.

Los pines de un FPGA están conectados a bloques programables de entrada/ salida que facilitan conectar las señales de los bloques lógicos al mundo externo.

Cada bloque de entrada salida tiene un número de opciones. El pin puede ser configurado para ser entrada o salida mediante un buffer triestado. El bloque contiene *flip flops* para guardar los valores de entrada o de salida. La señal de salida puede ser invertida si se desea, mediante una compuerta XOR.

Los FPGAs recientes tienen también bloques especializados. Dentro de estos bloques existen los de memoria RAM, que van de 16k a 10M bits los cuales pueden servir para almacenar datos necesarios en un proceso. El vendedor puede incluir también bloques de procesamiento digital de señales con *hardware* para realizar transformadas rápidas de Fourier, filtros FIR e IIR, entre otras cosas. Existen también bloques embebidos de procesadores dentro de los FPGAs modernos, como el MicroBlaze de Xilinx y el Nios de Altera.

Capítulo 3. Microcontroladores

En este capítulo se define el microcontrolador, con una breve semblanza histórica sobre procesadores. Se habla más detenidamente sobre los PICs y se muestran características del PIC seleccionado para el proyecto.

3.1 Definiciones

3.1.1 Un sistema computacional incluye una Unidad Central de Procesamiento (CPU), una unidad de memoria que contiene el programa y datos y una interfaz de entrada/ salida con dispositivos de entrada y salida asociados.

Antes de que Intel presentara el primer microprocesador en 1971, el CPU era constituido por varios componentes. De hecho, en 1958 la computadora SAGE de la fuerza aérea norteamericana requería 40,000 pies cuadrados, 3 MW de potencia y tenía una unidad de memoria magnética de 30,000 tubos con 4 mil palabras de 32 bits cada uno. El término microprocesador se llegó a usar por primera vez en Intel en 1972 y se refiere generalmente a la implementación de de las funciones del CPU en un solo circuito LSI (*Large Scale of Integration*). Una microcomputadora, por lo tanto, es una computadora construida con un microprocesador como CPU y otros pocos componentes para memoria y dispositivos de entrada/salida. La microcomputadora que contenía el Intel 4004 de 1972 consistía de 4 circuitos integrados: el CPU, una memoria ROM para el programa, memoria RAM para datos y un chip de corrimiento de registros como expansión de salida.

3.1.2 Un microcontrolador es una microcomputadora que, en el mismo encapsulado, incluye el CPU, la memoria y las unidades de entrada/salida. Se usan para aplicaciones específicas de control que se pueden encontrar en productos tan variados como hornos microondas, tostadores, reproductores de mp3 y automóviles.

3.1.3 PICs

Los PICs (*Peripheral Interface Controller*) son microcontroladores fabricados por la compañía Microchip Technologies. Los PICs son microcontroladores de tipo RISC (*Reduced Instruction Set Computer*) con arquitectura Harvard.

Hoy en día los PICs vienen con varios periféricos integrados, como módulos de comunicación serie, convertidores analógico-digitales, *timers*, módulos de captura y comparación, etc.

Algunas de las características más relevantes de los PICs se muestran a continuación:

- Área de código y de datos separadas (Arquitectura Harvard).
- Un reducido número de instrucciones de largo fijo.
- La mayoría de las instrucciones se ejecutan en un solo ciclo de ejecución (4 ciclos de reloj).
- Un solo acumulador (W), cuyo uso es implícito (no está especificado en la instrucción).
- Todas las posiciones de la RAM funcionan como registros de origen y/o de destino de operaciones matemáticas y otras funciones.
- Una pila de hardware para almacenar instrucciones de regreso de funciones.
- Una relativamente pequeña cantidad de espacio de datos direccionable (típicamente, 256 bytes), extensible a través de manipulación de bancos de memoria.
- El espacio de datos está relacionado con el CPU, puertos, y los registros de los periféricos.
- El contador de programa esta también relacionado dentro del espacio de datos, y es posible escribir en él (permitiendo saltos indirectos).

Los PICs se pueden clasificar de acuerdo al tamaño de palabra que emplean. Siendo así, las familias básicas tienen un tamaño de palabra de 8 bits, la gama media tiene un tamaño de palabra de 16 bits, mientras que la gama más alta trabaja con palabras de 32 bits.

3.2 Elección del microcontrolador

Se escogió la serie PIC18FXX2 ya que, dentro de las gamas bajas de los PICs (con datos de 8 bits y sin llegar a los dsPICs) esa serie logra hacer una multiplicación por hardware en un ciclo de ejecución, lo cual, empleando la frecuencia más alta de reloj que permiten, implica obtener el resultado de una multiplicación en 100 nanosegundos, por lo que imponía el reto a vencer en cuanto a rapidez del coprocesador.

Se escogió el PIC18F452, cuyas características son las siguientes:

- Memoria de programa de 32 kbytes, que equivale a 16384 instrucciones de una palabra.
- Memoria RAM de 1536 bytes.
- Memoria EEPROM para datos de 256 bytes.
- 16 bits de tamaño de palabra para instrucciones y 8 bits para datos
- Frecuencia de operación de 40 MHz
- Multiplicador hardware de un ciclo de 8x8 bits.
- Set de 75 instrucciones.
- Módulo de comunicación serie, módulo de PWM y captura y comparación, módulo de conversión analogico-digital de 10 bits y módulo de temporizador como periféricos
- 5 puertos de entrada/salida.
-

El diagrama de bloques del PIC18F452 se muestra a continuación:

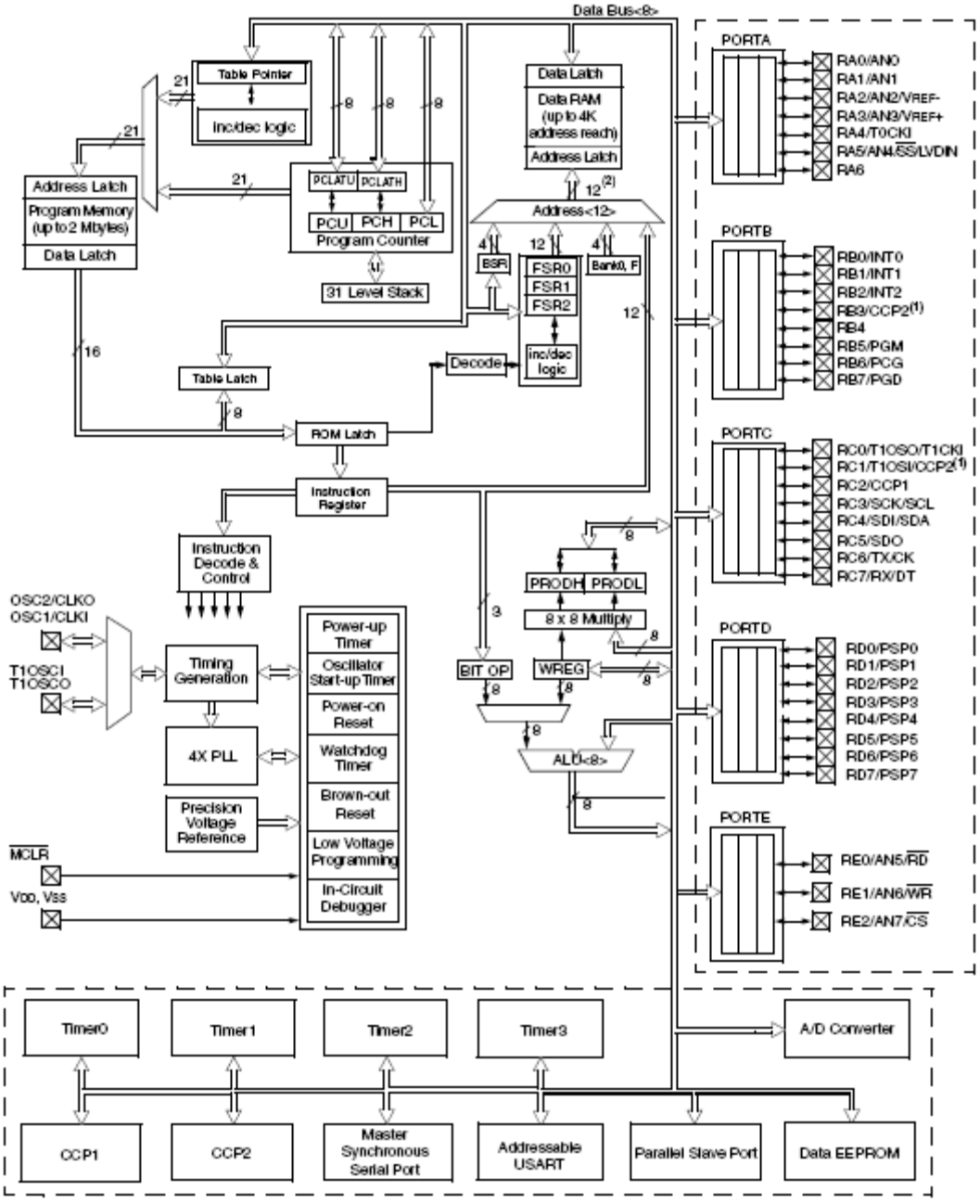


Figura 3.1 Diagrama de bloques del PIC18F45.

Organización de memoria en el PIC18F452

Existen tres bloques de memoria dentro del microcontrolador:

- Memoria de Programa
- Memoria RAM de datos
- Memoria EEPROM de datos

El contador de programa (PC, *program counter*) de 21 bits es capaz de direccionar todas las localidades de memoria de programa. La dirección del vector de RESET se encuentra en la localidad 0000h.

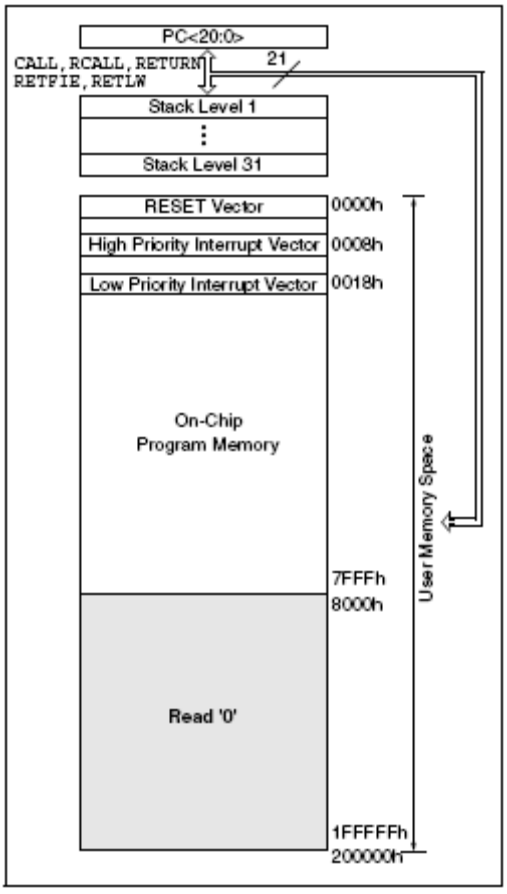


Figura 3.2 Mapa de memoria de programa del PIC18F452.

El *stack* de dirección de regreso permite que pueda ocurrir una combinación de hasta 31 llamadas de programa e interrupciones. El PC se empuja al *stack* cada que se ejecuta una instrucción CALL o RCALL, o cuando se reconoce una interrupción. El valor de PC se recupera del *stack* con instrucciones como RETURN, RETLW o RETFIE.

El *stack* opera como RAM de 31 balabras de 21 bits, con un apuntador de *stack* (*stack* pointer, SP) de 5 bits inicializado en 00000b después de todos los RESETs. Durante una instrucción de tipo CALL el *stack pointer* se incrementa y en la localidad a la que apunta se escribe el valor del PC. Por otro lado, al ejecutarse una instrucción de tipo RETURN, los contenidos de la localidad de la RAM apuntada por el SP se transfieren al *program counter* y el *stack pointer* se decrementa.

Ciclo de reloj y ciclo de instrucción

La entrada de reloj se divide internamente en cuatro pulsos, Q1, Q2, Q3 y Q4. Internamente el PC se incrementa cada Q1, se busca la instrucción de la memoria de programa y se transfiere al registro de instrucción en Q4. La instrucción se decodifica y se ejecuta durante los siguientes Q1 a Q4. La siguiente figura muestra las señales previamente explicadas.

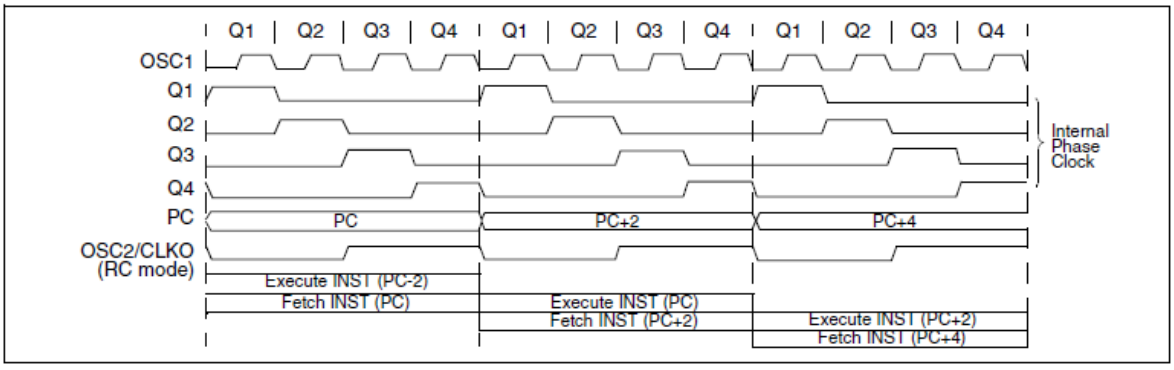


Figura 3.3 Ciclo de reloj e instrucción.

El ciclo de instrucción toma cuatro ciclos Q. La búsqueda de la instrucción (*fetch*), la decodificación y la ejecución tardan un ciclo de instrucción en realizarse, sin embargo por el método *pipeline*, cada instrucción se ejecuta efectivamente en un ciclo.

Instrucciones en la memoria de programa

La memoria de programa se direcciona en bytes. Las instrucciones se almacenan en dos o cuatro bytes dentro de la memoria de programa por lo que el PC se incrementa de dos en dos y así mantener alineada la lectura de las instrucciones.

Las instrucciones CALL y GOTO tienen una dirección de memoria de programa absoluta incrustada dentro de la misma instrucción. Los datos almacenados en dichas instrucciones son direcciones que escriben sobre el PC, el cual accesa directamente el byte de dirección requerido de la memoria de programa.

Organización de la memoria de datos

La memoria de datos está implementada como RAM estática. Cada registro tiene una dirección de 12 bits permitiendo los 4096 bytes de la memoria de datos. El mapa de memoria de datos se divide en 16 bancos que contienen 256 bytes cada uno. Los cuatro bits bajos del registro de selección de banco (*bank select register*, BSR) seleccionan el banco que será accesado, los bits altos no se implementan.

La memoria de datos puede ser accesada directa o indirectamente. El direccionamiento directo puede requerir el uso del registro BSR. El direccionamiento indirecto requiere el *File Select Register* (FSRn) y el correspondiente *Indirect File Operand* (INDFn). Cada FSR mantiene el valor de 12 bits de la dirección que puede ser usada para accesar cualquier localidad de la memoria de datos.

El set de instrucciones y la arquitectura del PIC permiten operaciones entre todos los bancos. Esto puede ser logrado mediante direccionamiento indirecto, o mediante la instrucción MOVFF, que mueve un valor de un registro a otro.

Capítulo 4 Algoritmos de multiplicación rápida

En este capítulo se habla sobre las distintas arquitecturas y algoritmos de multiplicación rápida que se encontraron en la bibliografía.

Los sistemas en un solo chip ofrecen grandes ventajas para desarrollar nuevos productos electrónicos. El uso de procesadores embebidos es cada vez mayor y uno de los principales bloques funcionales del procesador es el multiplicador.

Las operaciones de multiplicación y división son más complejas que las de adición y sustracción. Originalmente las multiplicaciones y divisiones se hacían por medio de *software,* haciendo programas por medio de sumas o restas sucesivas. Hoy en día, debido al avance tecnológico, se opta por realizar diseños en *hardware* ya que pueden conseguir velocidades más rápidas en dispositivos de bajo costo.

Las reglas de multiplicación binaria son sencillas:

1. Si el dígito multiplicador es 1, el multiplicando simplemente se copia y representa un producto parcial.
2. Si el dígito multiplicador es 0 entonces el producto es cero.

Para diseñar un multiplicador se debe tener la circuitería necesaria para lograr los siguientes propósitos:

1. debe ser capaz de identificar si el multiplicador es 0 ó 1.
2. debe ser capaz de desplazar los productos parciales.
3. debe ser capaz de sumar los productos parciales
4. como propósito adicional, debe examinar el signo de los números a multiplicar para determinar el signo del producto.

La mayoría de los algoritmos de multiplicación se basan en el método de suma y desplazamiento, por ello se analizarán los desplazamientos previamente.

Un desplazamiento aritmético es una operación en la que se mueve un número binario, ya sea a la derecha o a la izquierda. Un desplazamiento a la izquierda equivale a multiplicar el número por dos, y un desplazamiento a la derecha equivale a dividir dicho número por dos. Los desplazamientos no deben afectar el signo del número ya que las multiplicaciones o divisiones por 2 no afectan el signo.

Hablando de números binarios se trata de una operación que desplaza todos los bits del operando; cada bit del operando simplemente se mueve un número determinado de posiciones y las posiciones vacías se van rellenando. En los desplazamientos a la izquierda se introduce al bit menos significativo un cero, mientras que en el desplazamiento a la derecha, el bit más significativo, que representa el signo del número, se replica para llenar las posiciones vacías.

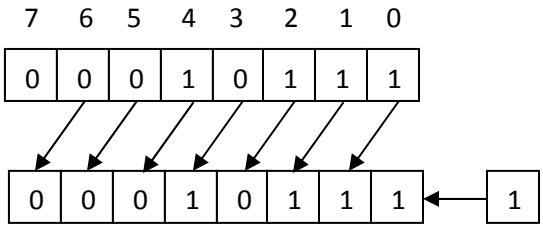


Figura 4.1 desplazamiento a la izquierda de un número de 8 bits.

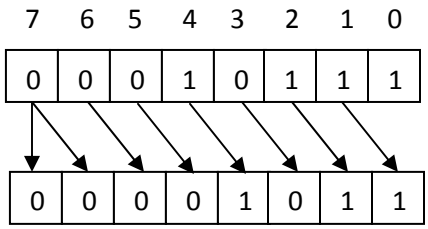


Figura 4.2 Desplazamiento a la derecha de un número de 8 bits.

4.1 Algoritmo clásico de multiplicación

El algoritmo clásico de multiplicación funciona bajo el conocido método de multiplicación decimal. Consiste de un multiplicador a, un multiplicando b y un producto p = a x b. la figura x muestra la multiplicación completa de dos palabras de cuatro bits.

$$
\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & \times & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & & b_0a_3 & b_0a_2 & b_0a_1 & b_0a_0 \\
 & & & b_1a_3 & b_1a_2 & b_1a_1 & b_1a_0 & \\
 & & b_2a_3 & b_2a_2 & b_2a_1 & b_2a_0 & & \\
 & b_3a_3 & b_3a_2 & b_3a_1 & b_3a_0 & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

Figura 4.3 multiplicación de dos palabras de 4 bits.

Al igual que en la multiplicación decimal, el primer bit del multiplicando b_0 se combina con todos los bits de a para formar el primer producto parcial. Los demás productos parciales se generan de la misma forma y el producto final es la suma de los productos parciales. Dentro de la multiplicación binaria, cuando se requiere hacer productos con números negativos (en formato de complemento a dos) se debe de considerar que los bits que se encuentran después del bit de signo son importantes y deben tomarse en cuenta. De esta manera el tamaño de palabra aumenta al doble de su tamaño original. Tomando en cuenta esto, el proceso de multiplicación con signo o sin signo es idéntico pero el tamaño de palabra se duplica.

Si x e y son números naturales representados por cadenas de n dígitos X e Y en un sistema base r, la operación de multiplicar produce p = x*y, representándose P

mediante una cadena de dígitos P. La multiplicación convencional se realizaría de la siguiente forma:

$$
\begin{array}{cccccccc}
 & & & X_3 & X_2 & X_1 & X_0 & \\
 & & & Y_3 & Y_2 & Y_1 & Y_0 & \\
\hline
 & & & X_3Y_0 & X_2Y_0 & X_1Y_0 & X_0Y_0 & \times Y_0 \\
 & & X_3Y_1 & X_2Y_1 & X_1Y_1 & X_0Y_1 & & \times Y_1\, r \\
 & X_3Y_2 & X_2Y_2 & X_1Y_2 & X_0Y_2 & & & \times Y_2\, r^2 \\
X_3Y_3 & X_2Y_3 & X_1Y_3 & X_0Y_3 & & & & \times Y_3\, r^3 \\
\hline
P7 & P6 & P5 & P4 & P3 & P2 & P1 & P0
\end{array}
$$

Del lado derecho se muestran los renglones de la multiplicación en una expresión más simplificada. El producto p se puede describir también de la siguiente forma:

$$x * y = x\sum_{i=0}^{n-1} Y_i r^i = \sum_{i=0}^{n-1} xY_i r^i \qquad \text{Ecuación 4.1}$$

Esta operación indica que primero se calculan los n términos Xr^iY_i y finalmente se calcula la suma. La mecanización de este método no es adecuada ya que requiere excesiva memoria y la capacidad de realizar sumas con operandos múltiples.

4.2 Multiplicador secuencial basado en sumas y desplazamientos

El multiplicador secuencial de números binarios sin signo de N bits basado en sumas y desplazamientos sucesivos utiliza un sumador completo y registros de almacenamiento y/o de desplazamiento, para acumular los productos parciales.

Debido a que se trabaja con números signados, se requiere también incluir dentro del algoritmo un método por el cual se discrimine el signo de los multiplicandos de tal manera que se determine el signo del producto final.

El algoritmo de sumas y desplazamientos genera, empleando la notación de la sección pasada, los términos xY_j y se van sumando uno a uno, teniendo en cuenta que antes de sumar los productos parciales, hay que desplazarlos.

Este algoritmo puede definirse mediante la siguiente recurrencia:

$p^{(0)} = 0$

$p^{(j+1)} = (p^{(j)} + r^n xY_j)1/r$ para j=0,1,…,n-1 Ecuación 4.2

En el caso de trabajar con 4 bits, tenemos que:

$p^{(0)} = 0$

$p^{(1)} = (r^4 xY_0)1/r = r^3 xY_0$

$p^{(2)} = (r^3 xY_0 + r^4 xY_1)1/r = r^2 xY_0 + r^3 xY_1$

$p^{(3)} = (r^2 xY_0 + r^3 xY_1 + r^4 xY_2)1/r = r^1 xY_0 + r^2 xY_1 + r^3 xY_2$

$p^{(4)} = (r^1 xY_0 + r^2 xY_1 + r^3 xY_2 + r^4 xY_3)1/r = xY_0 + r^1 xY_1 + r^2 xY_2 + r^3 xY_3$

Por lo tanto, el producto de dos números de n bits se obtiene en n pasos mediante este método. Los productos parciales se suman en los bits más significativos, y el resultado se desplaza un bit a la derecha (1/r). Para esto se necesita un sumador de n bits.

La siguiente figura representa un paso del algoritmo:

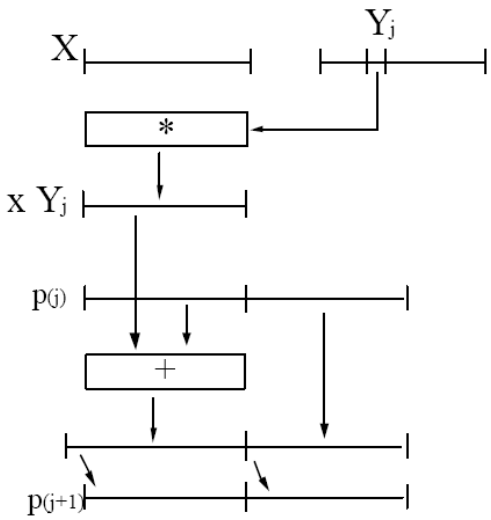


Figura 4.4 Un paso del algoritmo secuencial baso en sumas y desplazamientos a la derecha.

Se toma el bit j de Y y se multiplica por X, obteniendo XY_j, a continuación se suma con la parte alta de p(j) y se le aplica un desplazamiento.

La implementación de este algoritmo resulta atractiva con números de base r=2, es decir, binarios, ya que xY_j resulta 0 o bien x, dependiendo de que Y_j sea 0 ó 1.

A continuación se presenta una prueba de escritorio del algoritmo en cuestión con dos números de 4 bits.

n =5	r =2	x=23 X = 1 0 1 1 1	y =26 Y = 1 1 0 1 0
xY_0	0 0 0 0 0		
$p^{(0)}$	0 0 0 0 0		
	0 0 0 0 0		
xY_1	1 0 1 1 1		
$p^{(1)}$	0 0 0 0 0	0	
	1 0 1 1 1		
xY_2	0 0 0 0 0		
$p^{(2)}$	0 1 0 1 1	1 0	
	0 1 0 1 1		
xY_3	1 0 1 1 1		
$p^{(3)}$	0 0 1 0 1	1 1 0	
	1 1 1 0 0		
xY4	1 0 1 1 1		
p(4)	0 1 1 1 0	0 1 1 0	
	1 0 0 1 0 1		
p(5)	**1 0 0 1 0**	**1 0 1 1 0**	**= 598**

Figura 4.5 Prueba de escritorio del algoritmo

El diagrama de bloques del algoritmo para el multiplicador secuencial es el siguiente:

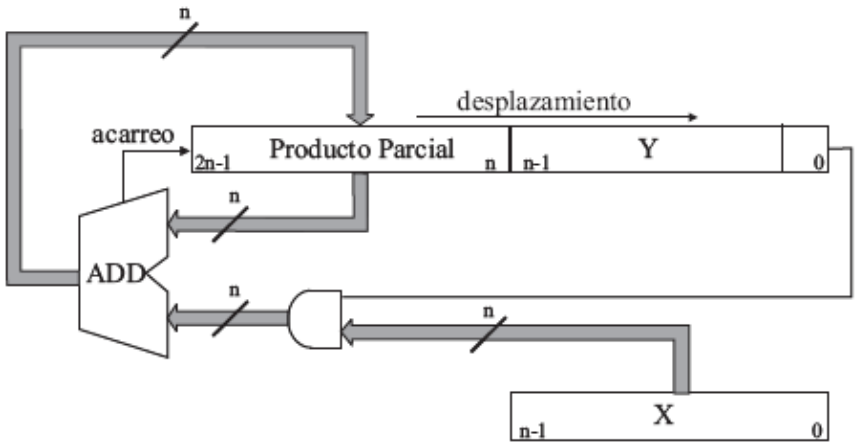


Figura 4.6 Diagrama de bloques del multiplicador secuencial basado en sumas y desplazamientos.

Los multiplicandos son X e Y, dos números de n bits, y el producto total de 2n bits. Dentro del registro del producto total se guarda en la parte baja el multiplicando Y, mientras que la parte alta se llenará de ceros. El diagrama nos muestra que se hacen las multiplicaciones parciales del bit menos significativo de Y con todos los bits de X. En seguida se realiza una suma del producto parcial con la parte alta del producto total y el resultado se guarda en la misma parte alta del producto total. Finalmente se hace un desplazamiento del registro del producto total, obteniendo así un nuevo bit menos significativo de Y con el cual se obtiene un nuevo producto parcial. Este proceso se repite n veces y al final se obtiene el resultado en el registro del producto final.

4.3 Multiplicador Ripple Carry

El multiplicador basado en un arreglo de sumadores de acarreo propagado es una aproximación para implementar el algoritmo de sumas sucesivas y desplazamientos; este tipo de multiplicador tiene la característica de transferir la propagación del acarreo a la siguiente suma parcial hasta que terminen los productos parciales correspondientes a esta fila, en este instante el acarreo generado se propaga al último producto de la siguiente fila de productos parciales y así sucesivamente hasta culminar la multiplicación.

El bloque principal es un sumador completo de 1 bit, y el diagrama de bloques de un multiplicador ripple carry de 4 bits se muestra en la Figura 4.7.

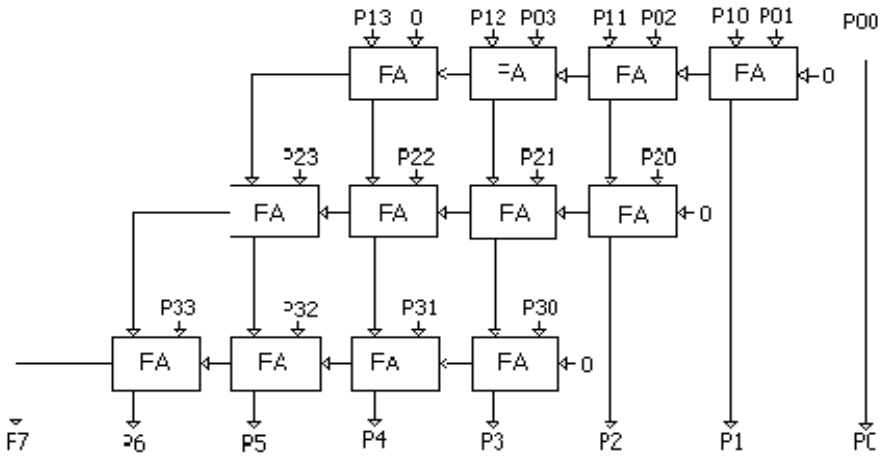


Figura 4.7 Multiplicador Ripple Carry.

4.4 Multiplicador Carry Save

El multiplicador basado en el arreglo de sumadores con acarreo salvado es otra aproximación para implementar el algoritmo de sumas sucesivas y desplazamientos. La idea es romper la cadena de acarreo propagado del sumador ripple para disminuir el retardo de cada suma, lo cual permite acelerar la multiplicación. El multiplicador tiene la característica de permitir salvar el acarreo generado en las sumas parciales y transferirlo como acarreo de entrada a la siguiente suma parcial de la fila de productos parciales siguiente. Este multiplicador es también conocido con el nombre de Multiplicador de Braun. El diagrama de un multiplicador carry save de 4 bits, se muestra en la Figura 4.8.

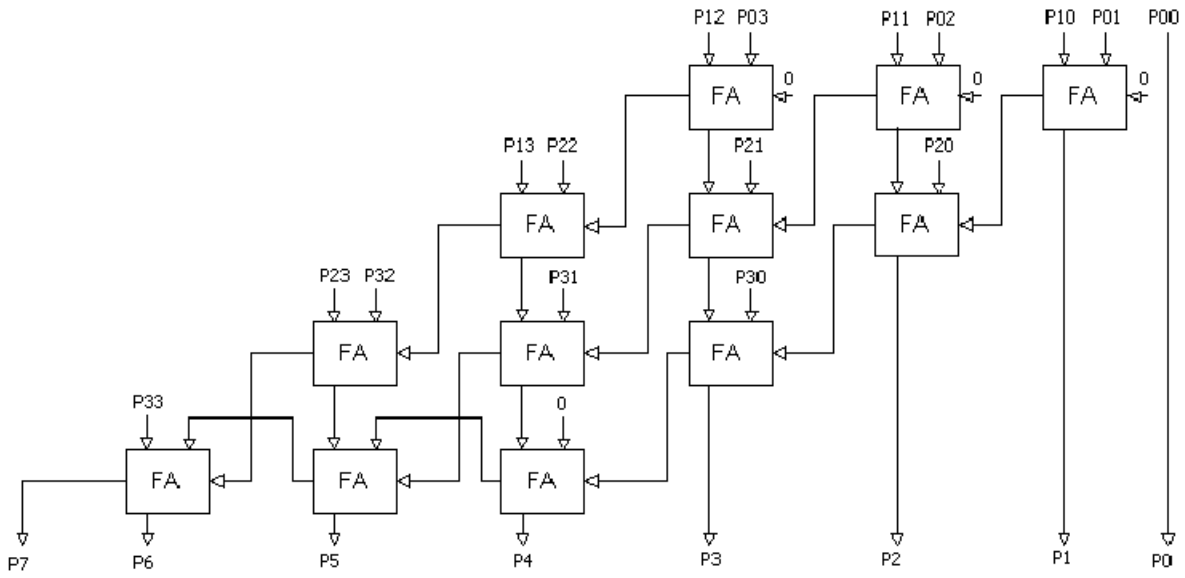


Figura 4.8 Multiplicador Carry Save.

4.5 Multiplicador de Wallace

El multiplicador de Wallace es una variante del algoritmo de sumas sucesivas y desplazamientos, donde se utilizan los bloques de sumadores completos con sus tres entradas recibiendo términos productos y generando un término suma que se adiciona con otro termino suma. El diagrama de bloques de un multiplicador de Wallace de 4 bits se muestra en la Figura 4.9.

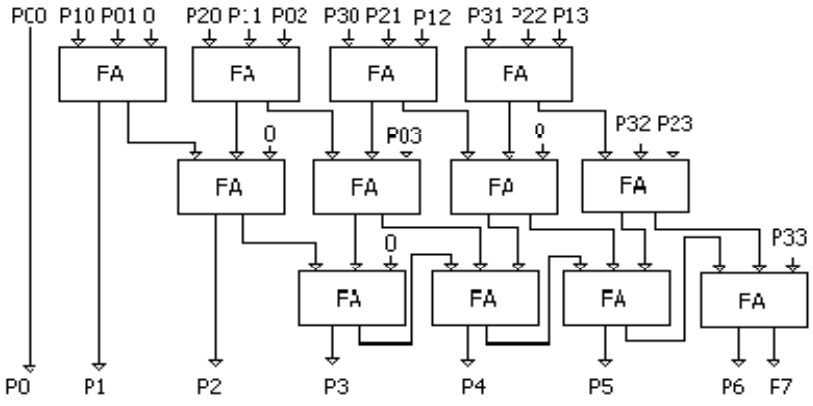


Figura 4.9 Multiplicador de Wallace.

4.6 Multiplicador de Baugh-Wooley

El multiplicador de Baugh-Wooley permite realizar la multiplicación de números con signo usando la representación en complemento a dos. Este multiplicador es una adecuación del algoritmo de sumas sucesivas y desplazamientos que permite realizar la multiplicación de números con signo. Este tipo de multiplicador está basado en arreglos de sumadores de acarreo salvado. El diagrama de un multiplicador de Baugh–Wooley de 4 bits se muestra en la Figura 4.10.

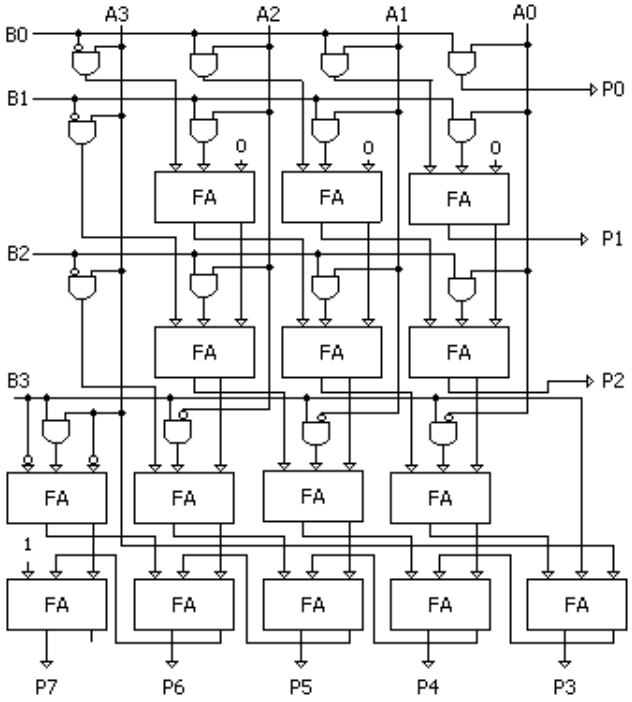


Figura 4.10 Multiplicador Baugh-Wooley.

4.7 Algoritmo de Booth

Como se ha visto, la multiplicación se basa en dos operaciones básicas: la suma y el desplazamiento. De estas dos, la que consume más tiempo es la suma. El algoritmo de Booth pretende minimizar el número de sumas que se deben realizar durante el proceso de multiplicación para reducir el tiempo en el que se realiza.

El algoritmo de Booth realiza multiplicaciones de números binarios con signo. De esta forma, los números negativos deben tener el formato de complemento a dos.

El algoritmo de Booth funciona haciendo una recodificiación del multiplicador, analizando los últimos dos bits (B_i y B_{i-1}) de acuerdo a la tabla que se muestra a continuación.

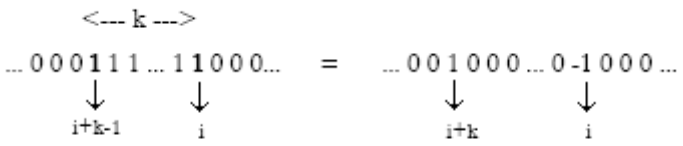
Bits del multiplicador B_i	B_{i-1}	Dígito recodificado	Operación
0	0	0	0 x multiplicando
0	1	1	1 x multiplicando
1	0	-1	-1 x multiplicando
1	1	0	0 x multiplicando

Tabla 4.1 Recodificación del algoritmo de Booth

Como se puede observar de la tabla, el algoritmo de Booth agiliza la multiplicación al encontrar cadenas de unos consecutivos. Esto se basa en que una cadena de unos se puede reducir usando la siguiente equivalencia:

$$2^{i+k-1} + \ldots + 2^{i+1} + 2^{i} = 2^{i+k} - 2^{i} \qquad \text{Ecuación 4.3}$$

Si se encuentra una cadena de k bits a 1, desaparece y se sustituye por una más simple que tiene un 1 al inicio (dígito más significativo en la cadena sustituida) y un -1 al final, dejando los dígitos intermedios en 0.



En el número recodificado, el peso del dígito D_i sigue siendo 2^i para cualquier dígito, sólo que ahora se aceptan números negativos.

Ejemplos:

109	→	0110 1101	→	10-11 0-11-1 (128-32+16-4+2-1 = 109)
-109	→	1001 0011	→	-101-1 010-1 (-128+32-16+4-1= -109)

Al realizar la multiplicación se usa la nueva codificación de modo que si se trata de un cero sólo se desplaza, si es un 1, se realiza una suma y el desplazamiento, y si se encuentra un -1, lo que se hace es una resta y el desplazamiento. Todas las iteraciones son iguales. El número de iteraciones que se realizan es igual al número de bits de los multiplicandos.

A continuación se muestra una prueba de escritorio del algoritmo de Booth.

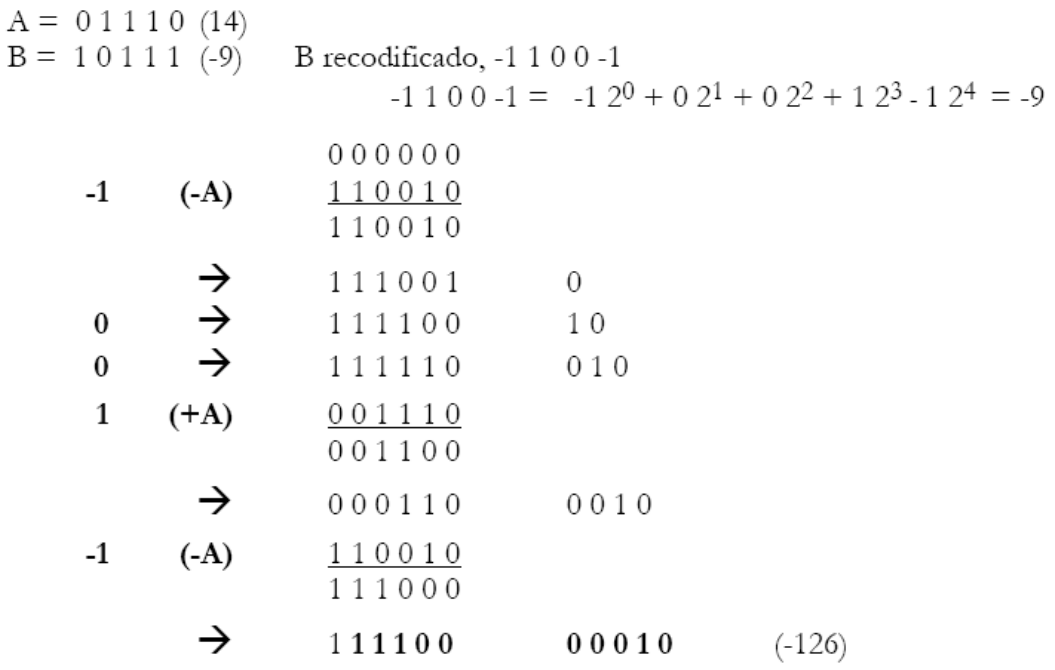


Figura 4.11 Prueba de escritorio del algoritmo de Booth

En la figura anterior, del lado izquierdo se muestra el dígito recodificado en cada paso, con lo que se decide si se hace una suma, una resta o simplemente un desplazamiento. Los desplazamientos se pueden observar en la figura, cada vez que se encuentra una flecha, lo cual produce un corrimiento de bits a la derecha.

Se puede observar que al encontrarse varios unos o ceros seguidos, se omiten sumas o restas con lo que se ahorra el tiempo que tomaría realizarlas, reduciendo el tiempo total de ejecución. El peor caso en este algoritmo resultaría ser claramente el que no incluya cadenas de ceros seguidos o de unos seguidos, ya que se realizaría la suma, o resta, en cada paso del proceso. Es por esto que el peor caso del algoritmo de Booth se asemeja mucho al tiempo que tarda el algoritmo de sumas y desplazamientos, sin embargo este tiempo se puede ver reducido significativamente en otros casos.

La siguiente figura muestra el diagrama de flujo del algoritmo de Booth.

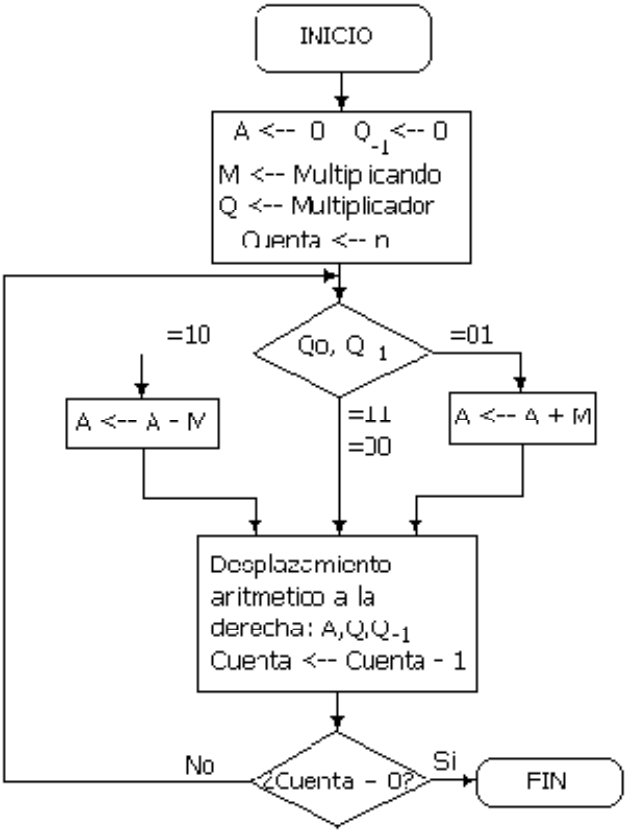


Figura 4.12 Algoritmo de Booth.

En el estado inicial se mandan a cero los bits del vector A, y al multiplicador Q se le agrega un 0 como bit menos significativo para realizar la primera comparación. Se tiene también un contador de ciclos inicializado en n= número de bits de los multiplicandos. En cada ciclo se leen los últimos dos bits de Q y se decide si se hace la suma o resta de A y M, o simplemente se omite esa operación. Posteriormente se hace un desplazamiento a la derecha del vector que contiene A y Q, el cual contendrá el resultado final, y se disminuye el contador. Finalmente se checa si ya se realizaron todos los ciclos, en caso de que no, se vuelve a realizar

otro, y en el caso de que el contador haya llegado a cero, ya se ha terminado la multiplicación.

Mega función de Altera

Altera provee una librería de mega funciones parametrizadas, dentro de la cual se encuentra la mega función *lpm_mult*, la cual es un bloque parametrizado descrito en lenguaje de alto nivel. Sin embargo no se recomienda usarla, puesto que utiliza demasiada área (elementos lógicos) dentro del dispositivo lógico programable.

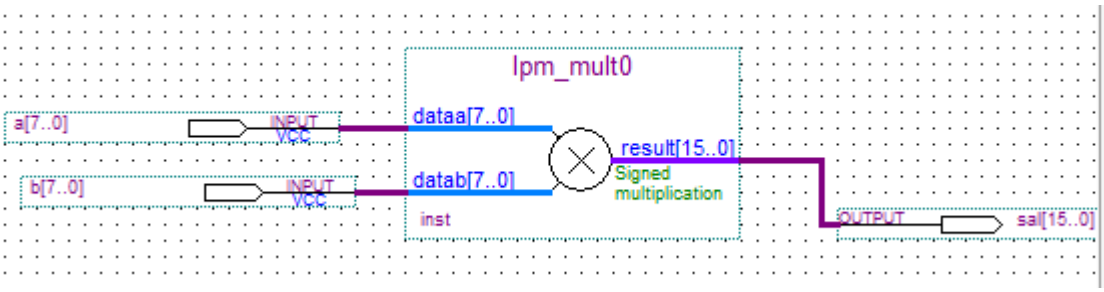


Figura 4.13 Bloque de la mega función LPM_MULT.

Capítulo 5 Metodología

En este capítulo se muestra la descripción de todos los pasos que se siguieron para la realización total de este proyecto.

5.1 Descripción de circuitos lógicos

Los circuitos lógicos son los que contienen, en sí, el multiplicador rápido. Se decidió realizar las descripciones de los circuitos mediante la herramienta Quartus II, de la compañía Altera. Dicho *software* permite describir en VHDL circuitos lógicos para que después se puedan sintetizar en algún dispositivo específico. Asimismo, permite simular los circuitos generados y realizar algunas verificaciones de tiempo de respuesta y área utilizada (o bien, a utilizar) dentro del dispositivo entre otras opciones. Posteriormente, con el mismo *software* y alguna interfaz, es posible programar y/o configurar los circuitos descritos dentro del dispositivo que se desea emplear.

Después de haber realizado la investigación sobre arquitecturas y algoritmos de multiplicación rápida, se optó por realizar un par de ellos. El primero fue el multiplicador secuencial basado en sumas y desplazamientos, puesto que implicaba una relativa facilidad en la descripción en VHDL además de que es el algoritmo de sumas y desplazamientos sin un método que acelere la multiplicación por lo que era factible tomarlo como referencia a la hora de comparar tiempos con otras arquitecturas.

El otro algoritmo que se decidió implementar fue el de Booth, ya que dentro de lo recabado en la bibliografía, apuntaba a ser el algoritmo más rápido para el objetivo específico de este proyecto, debido a que la mayoría de los algoritmos o arquitecturas disminuyen el tiempo en el que se realizan las sumas mediante varios métodos que implican en su mayoría diferentes usos del carry generado por los elementos que realizan las adiciones; sin embargo el algoritmo de Booth,

debido a la recodificación antes vista, logra evitar la realización de varias sumas, que como sabemos es la operación dentro de la multiplicación que más tiempo consume.

Cabe mencionar que dentro de las descripciones de los circuitos lógicos no sólo se encuentran los multiplicadores, sino que se debieron hacer adecuaciones para que fuera posible realizar la comunicación posteriormente con el PIC. A lo que se refiere lo anterior es que debido a que el PIC no tiene suficientes puertos de entrada y salida como para albergar los 16 bits de los multiplicandos así como los 16 bits del resultado del multiplicador, aparte de la interfaz que se escoja para mostrar la multiplicación al usuario del sistema completo, se agregaron ciertos elementos a la descripción lógica tanto a la entrada como a la salida del sistema configurado en el FPGA.

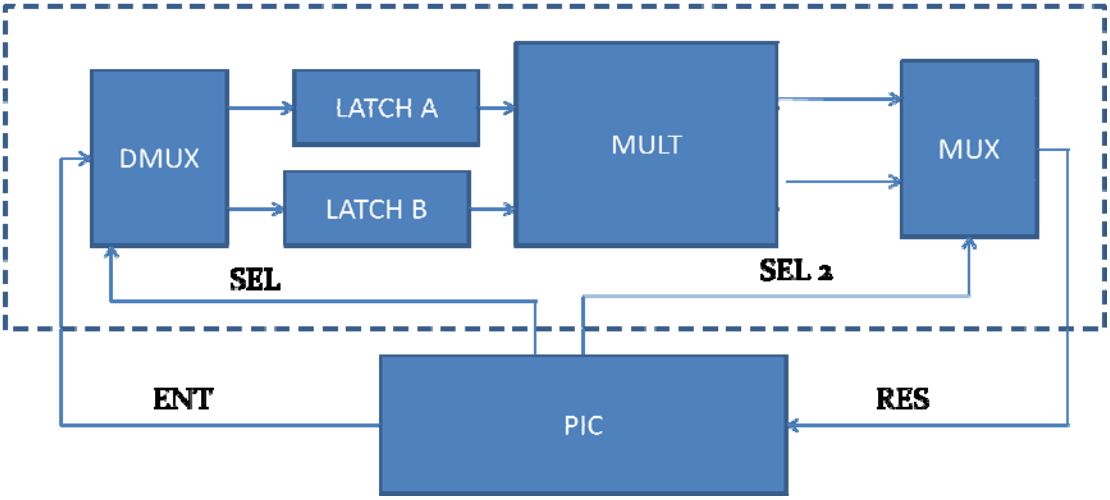


Figura 5.1 Diagrama de bloques del sistema completo.

La figura anterior muestra el diagrama de bloques del sistema FPGA-PIC. El sistema descrito en VHDL se encuentra encerrado por la línea punteada. Se observa que como elemento central tenemos el multiplicador y a su entrada y salida tenemos otros elementos lógicos. Para disminuir el número de pines

utilizados del PIC se decidió multiplexar tanto las entradas como las salidas al multiplicador, de esta forma, el PIC sólo manda 8 bits de entrada y recibe 8 bits del resultado. Con este propósito se crearon también dos señales de control, que seleccionan en la entrada si se trata del multiplicando A o B y dependiendo de esto lo registrado se guarda en el latch A o en el latch B de donde posteriormente el multiplicador obtendrá sus números a multiplicar. La segunda señal de control selecciona si se va a leer en el PIC la parte alta o la parte baja del resultado de la multiplicación.

Por lo tanto, dentro de la descripción lógica se agregaron un demultiplexor junto con dos latches de 8 bits para determinar si la entrada se refiere al multiplicando A o B, y un multiplexor a la salida que envía la parte alta o la parte baja del resultado al PIC, dependiendo de la señal de selección. Se decidió que las señales de control las generara el PIC para mantener la parte de lectura y de muestra de los números totalmente sincronizada.

Las descripciones en VHDL del multiplicador secuencial basado en sumas y desplazamientos, así como del multiplicador basado en el algoritmo de Booth se encuentran en la sección de anexos de este trabajo.

Con las descripciones realizadas en VHDL Quartus traduce esto a circuitería que realiza lo descrito.

5.2 Simulación de los circuitos lógicos

Una vez que se realizaron las descripciones de los circuitos lógicos, se pasó a realizar las simulaciones de los mismos. Quartus tiene una herramienta que realiza simulaciones de los circuitos realizados. Con esta herramienta es posible manipular las señales de entrada al sistema y observar el comportamiento de las señales de salida e inclusive de señales internas, con el propósito de comprobar que el comportamiento del circuito es el deseado.

Primero se simularon los multiplicadores aislados, para determinar si realizaban correctamente las multiplicaciones y en tiempos correctos.

Una vez que se determinó que sí funcionaban los multiplicadores se procedió a agregarles los elementos de entrada y salida que se explicaron en la sección anterior y se simuló el sistema completo. Se comprobó que los nuevos elementos interactuaban de forma correcta con el multiplicador, de acuerdo a los valores de las señales de selección que posteriormente se generarían por medio del PIC.

5.3 Verificación de los circuitos generados

Quartus crea varios archivos de verificación cada vez que se utiliza la herramienta de compilación. Dentro de la compilación se ejecutan varias herramientas: análisis y síntesis, fitter, assembler y analizador de tiempos. Para poder hacer uso de estas herramientas es necesario determinar el circuito integrado para el que está destinado el circuito que se describió, ya que varios de los análisis se realizan con base en el comportamiento específico de cada dispositivo. Por lo tanto, se debió escoger el dispositivo con el cual se iba a trabajar.

El laboratorio de electrónica del CCADET cuenta con una tarjeta UP1 de Altera, en la cual se encuentran dos dispositivos, uno de ellos es un CPLD de la familia MAX 7000 que originalmente se planeaba usar para los fines de esta tesis. Sin embargo los análisis de verificación demostraron que no tenía suficiente área para programar el multiplicador dentro de él, mucho menos podría ser posible programar el sistema completo. Por lo tanto se optó por usar el otro dispositivo dentro de la tarjeta UP1: un FPGA FLEX10k20RC240-4.

Dicho dispositivo es un FPGA de 240 pines, de los cuales 189 están disponibles para el usuario como entrada/salida, en un encapsulado tipo RQFP. Los dispositivos EPF10k20 contienen 1152 LEs y 144 LABs y son capaces de trabajar a 5 volts.

Dentro de la tarjeta UP1 se tienen unos dispositivos externos como *dip switches*, LEDs, *displays* de siete segmentos y otras interfaces, las cuales ya tienen asignadas sus pines de entrada/salida en el FPGA. Los pines asignables por el usuario tienen terminales en lo que Altera llama líneas de expansión, las cuales son filas dobles de hoyos disponibles para el acceso a señales globales, tierra, Vcc y pines de entrada/salida.

En la etapa de verificación de los circuitos lógicos, habiendo escogido ya el dispositivo a usar, Quartus pasa la descripción por varias etapas. La primera es análisis y síntesis, en la cual analiza la descripción hecha, de tal forma que si existe algún error en sintaxis de acuerdo a estándares de VHDL detiene la compilación y señala en qué lugar se encuentra el error o discrepancia y por qué se considera como tal. Este módulo del compilador también construye una base de datos, optimiza el diseño para el dispositivo elegido y *mapea* la tecnología para el diseño lógico. El reporte que ofrece Quartus sobre esta parte de la compilación incluye un resumen sobre los ajustes hechos para lograr la síntesis del circuito dentro del dispositivo, como la versión de VHDL con la que se trabaja, los archivos necesarios para sintetizar el circuito (dentro de los cuales se incluye el .vhd que se creó para la descripción, y las funciones necesarias para crear el circuito descrito). El reporte habla también sobre los *latches* creados por el usuario o inferidos por el compilador y los recursos del dispositivo a usar, entre otras cosas.

La segunda etapa de la compilación de Quartus es la herramienta llamada *fitter*, la cual construye el circuito lógico sobre el dispositivo seleccionado para el proyecto y coloca y conecta el diseño. El *fitter* entrega datos como el número de elementos lógicos usados, el número de pines de entrada/salida usados así como la ubicación de los mismos en el dispositivo, la lista de todos los pines del encapsulado y las señales que se encuentran en cada uno y el *fan out* de las señales de entrada, entre otras cosas.

Esta herramienta fue la que reportó que no era posible sintetizar el multiplicador dentro del CPLD de la tarjeta UP1 ya que no tenía el área lógica necesaria, por lo que hubo que decidir emplear el FPGA.

La tercera etapa que realiza Quartus se llama *Assembler*, la cual crea una imagen de programación del dispositivo para configura el dispositivo. Se generan los archivos para configurar el FPGA, ya sea mediante JTAG[1], en cuyo caso se le asigna un código de usuario para reconocer el dispositivo en la cadena y se genera el archivo .sof, o mediante un dispositivo de configuración, para lo cual se genera un archivo de tipo .pof que se emplea para programar el dispositivo que configurará el FPGA.

La cuarta y última etapa es el *timing analyzer*, que en el caso específico para FPGAs nos entrega cuatro tiempos específicos y una lista de señales involucradas en dichos tiempos. A continuación se describen los tiempos que entrega Quartus.

Tiempo t_{SU}

t_{SU} especifica la cantidad de tiempo que los datos necesitan para llegar y ser estables en un pin externo de entrada antes de de una transición de reloj en un pin de entrada/salida asociado a un reloj.

[1] Una interfaz JTAG (*Join Test Action Group*) es una interfaz de 5 pines diseñada para que múltiples circuitos integrados dentro de una tarjeta puedan ser conectados en cadena para ser probados, programados o configurados.

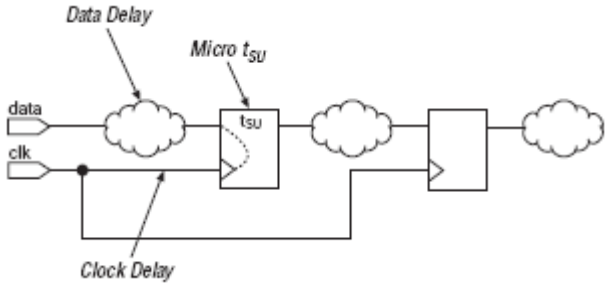


Figura 5.2 Tiempo de establecimiento de reloj.

El micro t_{SU} es el tiempo de establecimiento interno del registro. Es una característica del registro y no se afecta por las señales que alimentan al registro.

Tiempo t_H

t_H especifica la cantidad de tiempo que los datos necesitan mantenerse estables en un pin externo de entrada después de una transición de reloj en un pin de entrada/salida asociado al reloj.

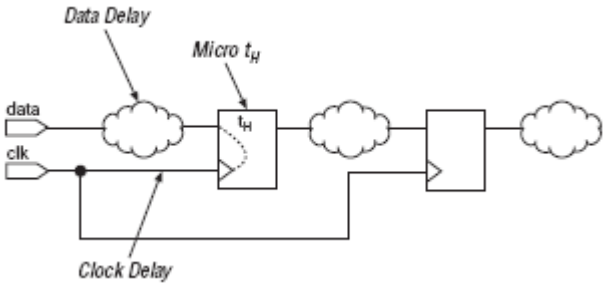


Figura 5.3 Tiempo de mantenimiento de reloj.

Tiempo t_{CO}

t_{CO} es el retraso denominado reloj a salida (*clock to output delay*), y es el máximo tiempo requerido para obtener una salida válida en un pin de salida alimentado por un registro, después de una transición de reloj en la entrada de reloj del registro. Micro t_{CO} es el retraso de reloj a salida interno del registro.

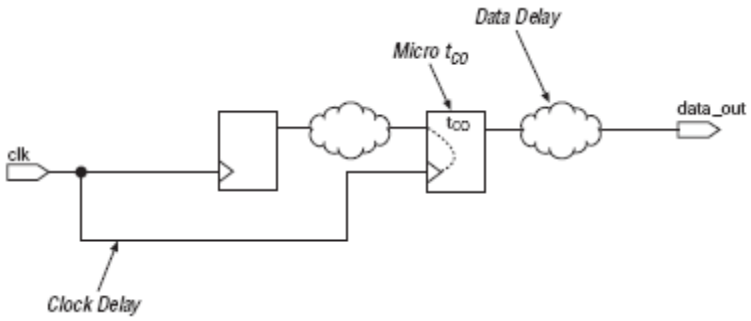


Figura 5.4 Retraso de reloj a salida.

Tiempo t_{PD}

El retraso de pin a pin (t_{PD}) es el tiempo requerido para que una señal de un pin de entrada se propague a través de lógica combinacional y aparezca en un pin de salida.

Puesto que no usamos reloj en los multiplicadores, el tiempo que nos concierne es el t_{PD}, ya que implica que los valores de entrada del multiplicador se propagan por toda la lógica generada y nos entrega el valor del producto en la salida. Es decir, debido a que el circuito descrito es combinacional, el único tiempo que nos concierne es el t_{PD}.

5.4 Implementación física

Una vez descritos los circuitos, habiéndolos verificado mediante Quartus y después de realizar las simulaciones correspondientes para constatar mediante el software su correcto funcionamiento, se realizó la implementación física. Como ya se mencionó se empleó el FPGA de la famila EPF10k20 incrustado en la tarjeta UP1 de Altera. Primero que nada se comprobó el funcionamiento correcto del

multiplicador únicamente, mediante 2 dip switches de ocho líneas cada uno actuando como entradas variables al FPGA y observando el producto obtenido a través de 16 LEDs conectados a las salidas correspondientes del dispositivo.

En esta etapa de las pruebas, el FPGA se configuró mediante JTAG empleando la interfaz ByteBlasterMV de Altera conectada al puerto paralelo de la PC.

Posteriormente para las pruebas finales con el sistema de prueba completo, se debió usar una memoria de configuración. Era necesario conseguir un dispositivo de configuración con encapsulado PDIP de 8 pines puesto que la tarjeta UP1 únicamente aceptaba ese tipo de circuitos integrados ya que tiene una base con esas características conectada al FPGA dispuesta de tal forma que logre configurar el FPGA cada que se alimente la tarjeta.

Por tal motivo, se optó por usar el EPC1PC8, fabricado también por Altera. Esta memoria es de tipo EPROM. Su forma de operar consiste en almacenar las interconexiones de la descripción realizada en VHDL y a la hora de que el sistema se enciende, este dispositivo manda dicha información al FPGA de forma serial. El proceso de configuración dura unos pocos milisegundos, después de los cuales el FPGA funciona de la manera en que fue especificado. El dispositivo contiene un oscilador interno que funciona como reloj tanto para él mismo como para el FPGA para que la etapa de configuración esté completamente sincronizada.

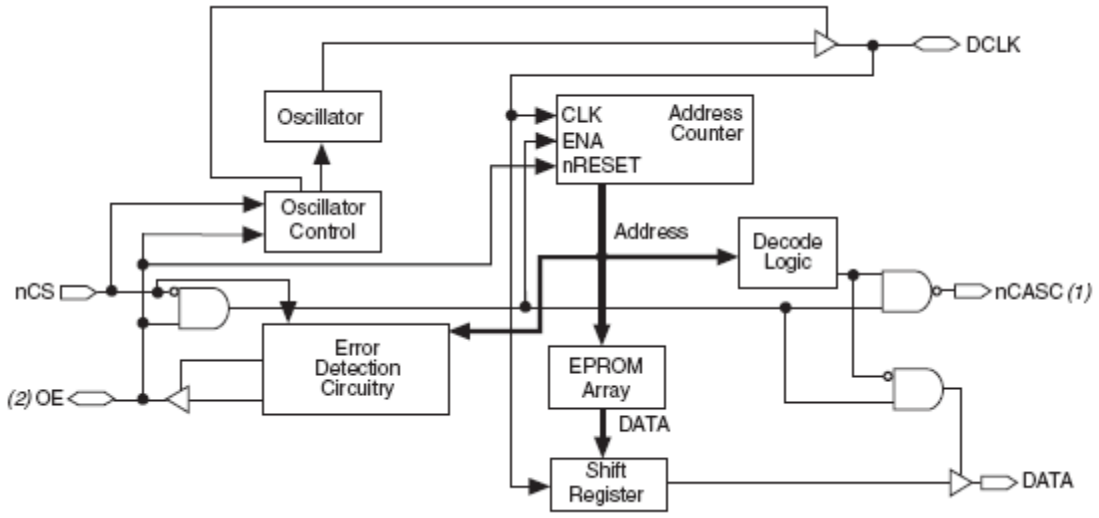


Figura 5.5 Diagrama de bloques del EPC1.

Las desventajas de este tipo de memorias de configuración es que sólo pueden ser programadas una sola vez, puesto que son de tipo EPROM, y no soportan JTAG, por lo que para ser programadas es necesario tener un programador con compatibilidad necesaria, o una interfaz de Altera, como el BitBlaster. En el caso de este proyecto, las memorias fueron programadas en el laboratorio abierto de la Facultad de Ingeniería.

Por otro lado, su costo es bajo, comparado con otras memorias de configuración e inclusive con otros métodos de configuración que pueden requerir un microcontrolador y/o memorias de otro tipo.

En la memoria de configuración se programó el sistema completo, de forma que ya se pudiera usar aunado con la segunda parte del proyecto, puesto que dichos dispositivos no son reprogramables ni muy fáciles de conseguir.

5.5 Programación del microcontrolador PIC

La segunda parte del proyecto trata de un PIC para que actuara como procesador principal del multiplicador. En este trabajo se usó simplemente para fines didácticos, de forma que sirviera para recibir los datos de entrada que se quieren multiplicar, así como el resultado proveniente del FPGA y de alguna manera mostrar toda la operación al usuario.

Se decidió usar un display de cristal líquido para mostrar tanto los multiplicandos como el resultado. Esto es importante ya que requiere de la programación de varias subrutinas específicas para el display dentro del programa principal.

Como se explicó en capítulos anteriores, se eligió el PIC18F452 para cumplir con los objetivos propuestos. Primero que nada, se realizó un diagrama de flujo de lo que debía realizar el PIC. El diagrama se muestra a continuación:

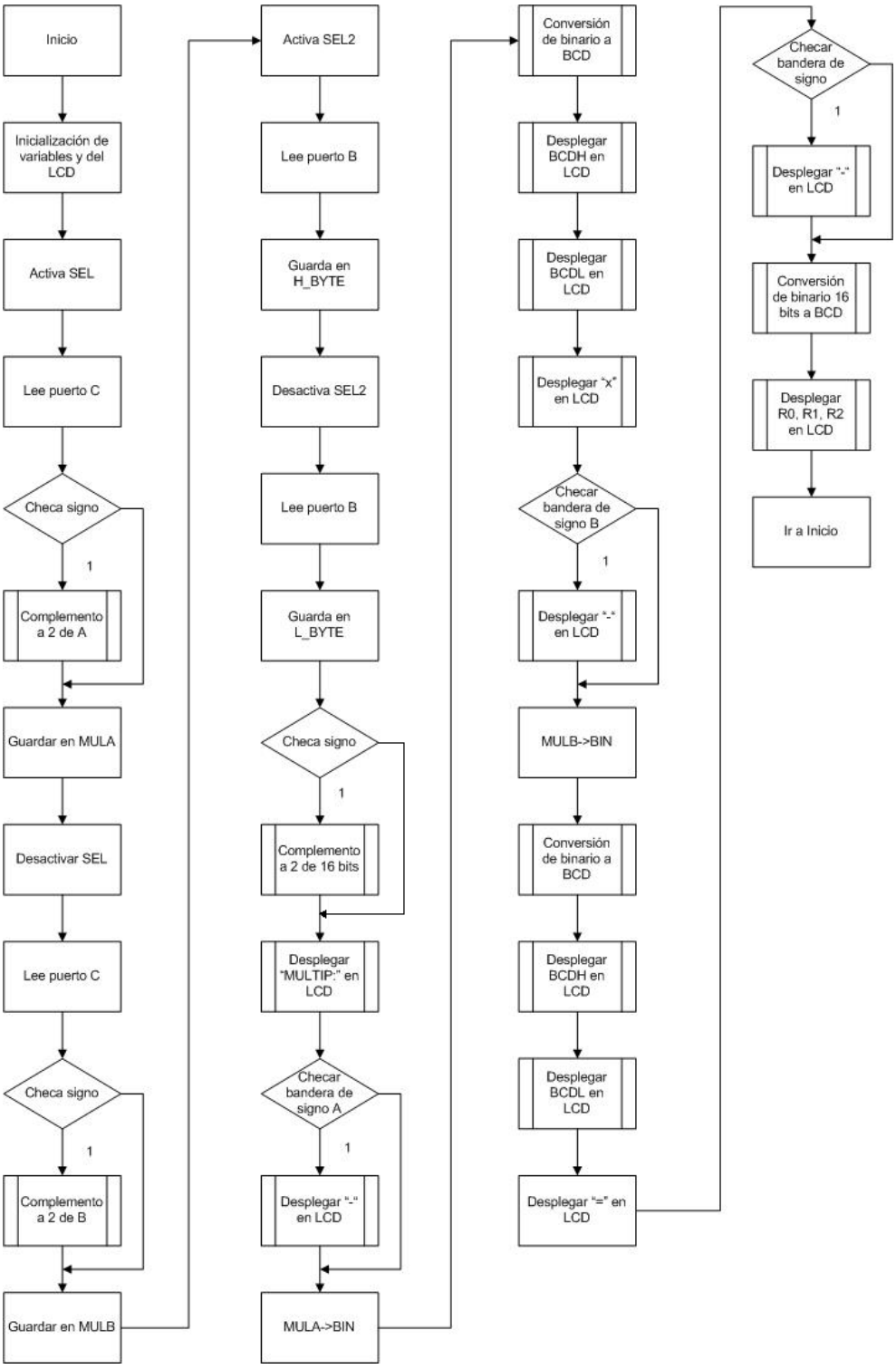


Figura 5.6 Diagrama de flujo del programa del PIC.

Como se puede observar, se tiene una etapa de inicialización del sistema, en el que se limpian variables y se inicializa el display, configurándolo de la forma en la que va a trabajar. Posteriormente se pasa a la etapa de lectura de datos. En esta etapa se manipulan las líneas de selección que se van a enviar al FPGA al mismo tiempo que se leen los multiplicandos, en primer lugar, y las partes alta y baja del resultado de la multiplicación. Durante esta misma etapa, se discrimina entre números positivos o negativos, de forma que si se trata de un número negativo, se le haga un complemento a 2 para poder mostrarlo como lo entendería el usuario. Si se trata de un número negativo, también se activa la respectiva bandera de signo para que pueda ser desplegado el signo "-" en el display, y de esta forma el usuario pueda corroborar que se trata de un número negativo.

A continuación se pasa a desplegar en el display la leyenda "MULTIP: " y enseguida el multiplicando A junto son su signo negativo si es que se trata de un número de esa índole, seguido del signo "x" y después el multiplicando B acompañado, si es necesario, de su signo negativo. Finalmente, se despliega el signo "=" y el resultado, con su signo si es que lo requiere. De esta forma, lo desplegado en el display tomaría la siguiente forma:

```
MULTIP:02x

-03=-0006
```

Para poder desplegar los números en el display, es necesario convertirlos a un formato que pueda entender dicho dispositivo. Por tal motivo, se manda llamar a las subrutinas BIN2BCD en el caso de los multiplicandos, la cual transforma el número binario en un número en código BCD, que más adelante empleará la subrutina BYTEDIS para finalmente enviar al display la información. En el caso del resultado, se debe usar otra subrutina, llamada B2_BCD, ya que ésta trabaja con números de 16 bits dividido en dos partes (alta y baja) que se colocan en dos

registros. Esta subrutina nos regresa tres valores que nuevamente con ayuda de la subrutina BYTEDIS se enviarán al display.

Finalizando la etapa del desplegado de la multiplicación en el display, se repite nuevamente todo el proceso.

Algunas otras subrutinas necesarias para el programa, son la de escritura de datos al display (LCDDAT), escritura de comandos al display (LCDCOM), la propia inicialización del display, las rutinas para realizar el complemento a dos, tanto de los números de 8 bits, como el resultado de 16 bits, y algunas subrutinas de retrasos, necesarias para la lectura de datos y el correcto funcionamiento del display.

El programa se realizó utilizando la herramienta MPLAB IDE de Microchip. El lenguaje empleado fue ensamblador. Se debieron crear todas las subrutinas que se han mencionado, así como el programa principal que realizara lo que se requería.

Para poder asegurar que las subrutinas funcionaban correctamente, previo a la programación del propio dispositivo, se empleó el simulador que se encuentra dentro de MPLAB, llamado MPLAB SIM, con el cual es posible correr el programa paso por paso y analizar el comportamiento del mismo. MPLAB permite observar los registros y unidades de memoria que se requiera, por lo que al correr el programa, se puede observar si se obtiene el resultado esperado en el registro en el que se quiere.

Posteriormente, cuando se finalizó el programa y habiendo probado las subrutinas, el siguiente paso fue programar el PIC y comprobar su funcionamiento.

El hardware de programación que se usó fue el PICSTART PLUS, que permite programar microcontroladores PIC, incluyendo la memoria de programa y bits de configuración entre otras cosas. El PICSTART PLUS permite ser operado

mediante el *software* MPLAB IDE. El *software* y el *hardware* de la programación del PIC se comunican a través de un cable RS232 estándar conectado a la PC.

Figura 5.7 Programador PICSTART PLUS.

5.6 Diseño del circuito de prueba

Para probar el funcionamiento del sistema completo se ideó un circuito de prueba que incluye al PIC, el display de cristal líquido para mostrar la multiplicación al usuario, un par de *dip switches* de ocho líneas cada uno, para modificar el valor de los multiplicandos, y las conexiones hacia el FPGA.

Como el diseño del sistema completo requirió que las entradas de los multiplicandos tanto para el PIC como para el FPGA fueran únicamente de 8 bits, y se fuera conmutando la lectura entre el multiplicando A y el B de acuerdo a la línea de selección generada por el PIC, el *hardware* para realizar las pruebas requería también responder a tal necesidad.

Se decidió utilizar dos circuitos integrados con ocho buffers digitales cada uno, controlados por la línea de selección y un circuito inversor de tal forma que cuando se encuentre en alto dicha señal de control, se active uno de los chips y habilite la lectura de uno de los multiplicandos. La misma señal de selección pasa por un circuito inversor para que cuando la señal se encuentre en bajo, se active el segundo chip y de esta forma leer el otro multiplicando.

El siguiente diagrama muestra lo anterior.

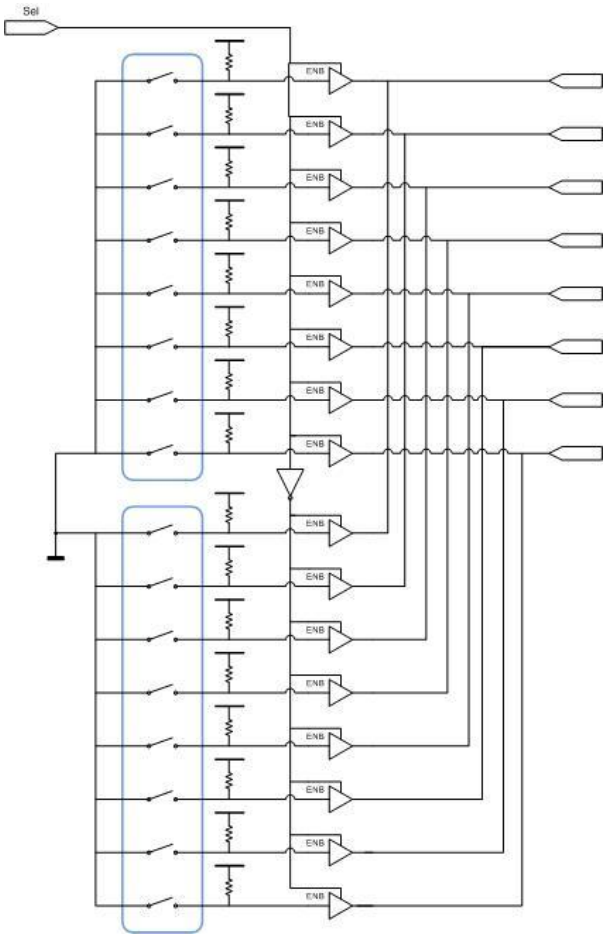
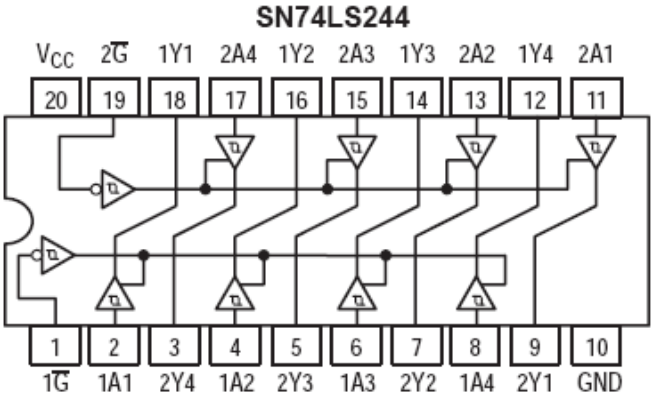


Figura 5.8 Diagrama del circuito de selección de multiplicandos.

Los *buffers* empleados son 74LS244, cada encapsulado incluye 8 *buffers* con su línea de activación. En el circuito integrado se tienen únicamente 2 líneas de selección.

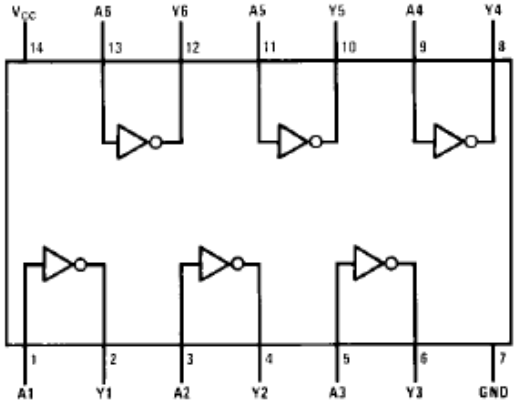


SN74LS244

INPUTS		OUTPUT
1G, 2G	D	
L	L	L
L	H	H
H	X	(Z)

Figura 5.9 Configuración interna y tabla de verdad del 74LS244.

El circuito inversor empleado para conmutar la línea de selección y diferenciar la entrada entre un multiplicando y otro fue el tradicional 74LS04.



$Y = \overline{A}$

Input	Output
A	Y
L	H
H	L

H = HIGH Logic Level
L = LOW Logic Level

Figura 5.10 Configuración interna y tabla de verdad del 74LS04.

De esta forma el circuito de prueba, está compuesto por el PIC, con los componentes externos que éste necesita (cristal, capacitores, etc), los *dip switches* con sus correspondientes buffers y pull ups, así como el circuito inversor para conmutar entre un *dip switch* y otro, el LCD para mostrar la multiplicación, y las bases para lograr la conexión entre el circuito de prueba y la tarjeta UP1 de Altera que contiene el FPGA usado.

.

5.7 Implementación del sistema

La parte final del proyecto fue crear el sistema físico completo. Con este fin, se realizó el diseño del circuito impreso a partir del diagrama esquemático del circuito de prueba. La paquetería PCAD se utilizó para realizar el diagrama de conexiones y corroborar la implementación correcta de los dispositivos electrónicos. A partir de dicho diagrama, se generaron las *netlists* para trasladar las conexiones hacia la herramienta de diseño de PCBs. Dentro de ésta, se acomodaron los componentes de la manera más conveniente posible y se realizaron las conexiones pertinentes.

Finalmente, se realizaron una serie de análisis dentro de la misma paquetería para asegurar que las conexiones realizadas en el diseño del circuito impreso correspondieran íntegramente a las que se encuentran en el diagrama esquemático, así como la verificación de parámetros preestablecidos de diseño de PCBs, como puede ser el de *clearance*, que precisa tener un mínimo de distancia entre líneas de cobre.

El diseño del circuito impreso se realizó con doble capa, puesto que en algunas partes del circuito, especialmente en el área de los buffers, se cruzaban las líneas de conexión. A continuación se muestra el diseño de las caras superior e inferior del PCB.

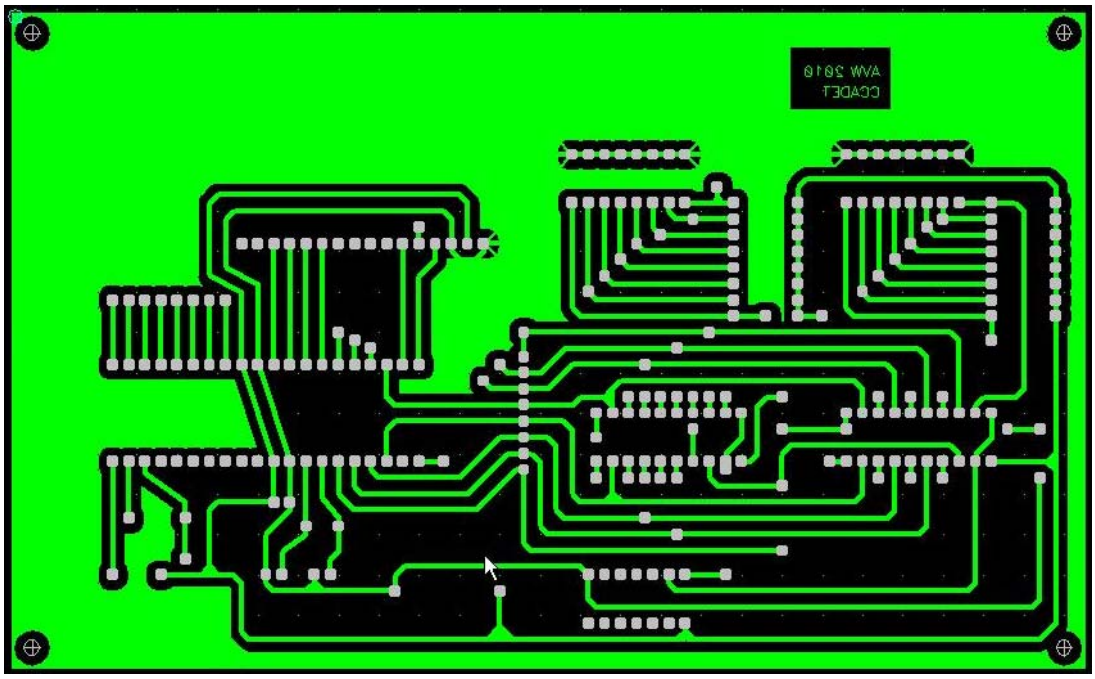


Figura 5.11 Cara inferior del circuito impreso.

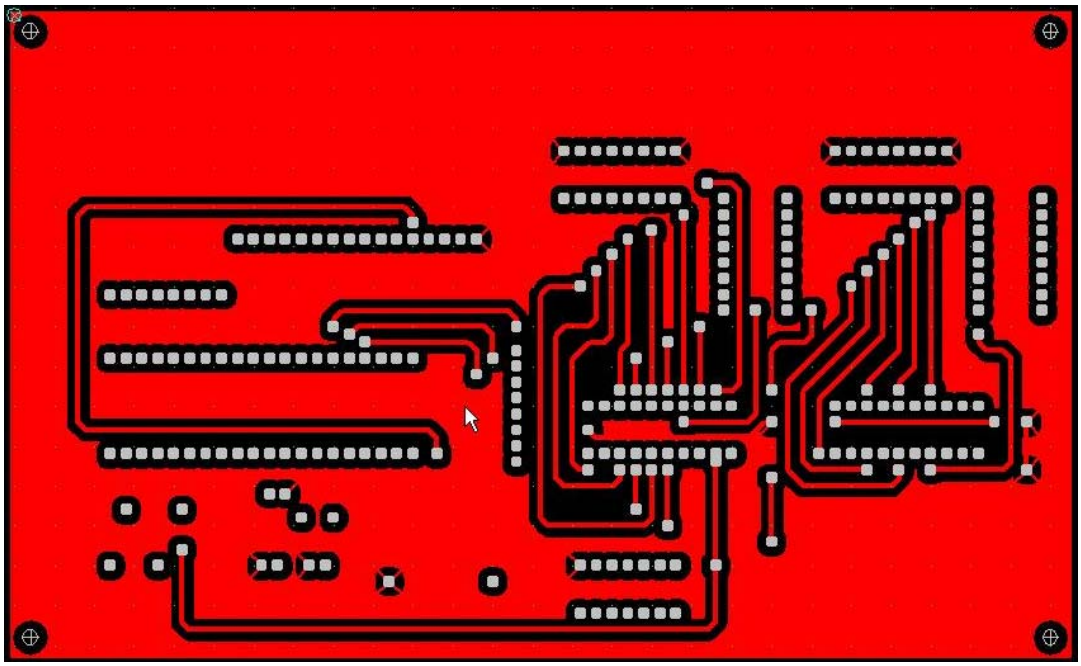


Figura 5.12 Cara superior del circuito impreso.

Una vez diseñado el circuito impreso, se mandó a hacer en el departamento de publicaciones del CCADET, se le soldaron los componentes, y se hicieron los cables de conexión entre el circuito de prueba y la tarjeta UP1 de Altera.

La etapa final de este proyecto consistió en realizar pruebas del funcionamiento del sistema completo, introduciendo números mediante los dip switches y obteniendo los resultados correctos de la multiplicación en el display incrustado en el circuito impreso que se hizo.

Las descripciones de los circuitos realizados, que se encuentran en la parte de anexos, fueron traducidas por Quartus para generar los circuitos físicos que se requería implementar. Las siguientes imágenes muestran los sistemas descritos en VHDL.

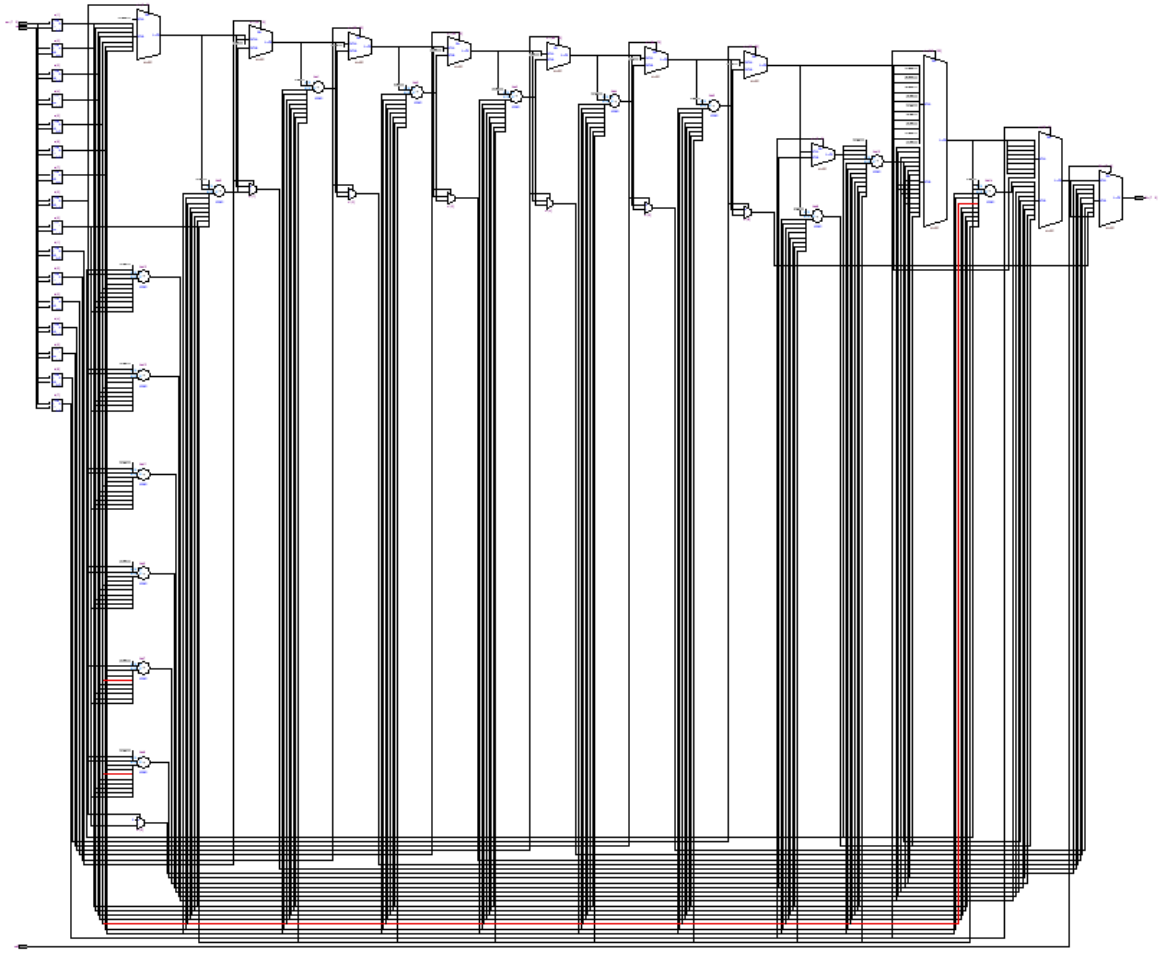


Figura 6.1 Diagrama RTL del multiplicador secuencial basado en sumas y desplazamientos.

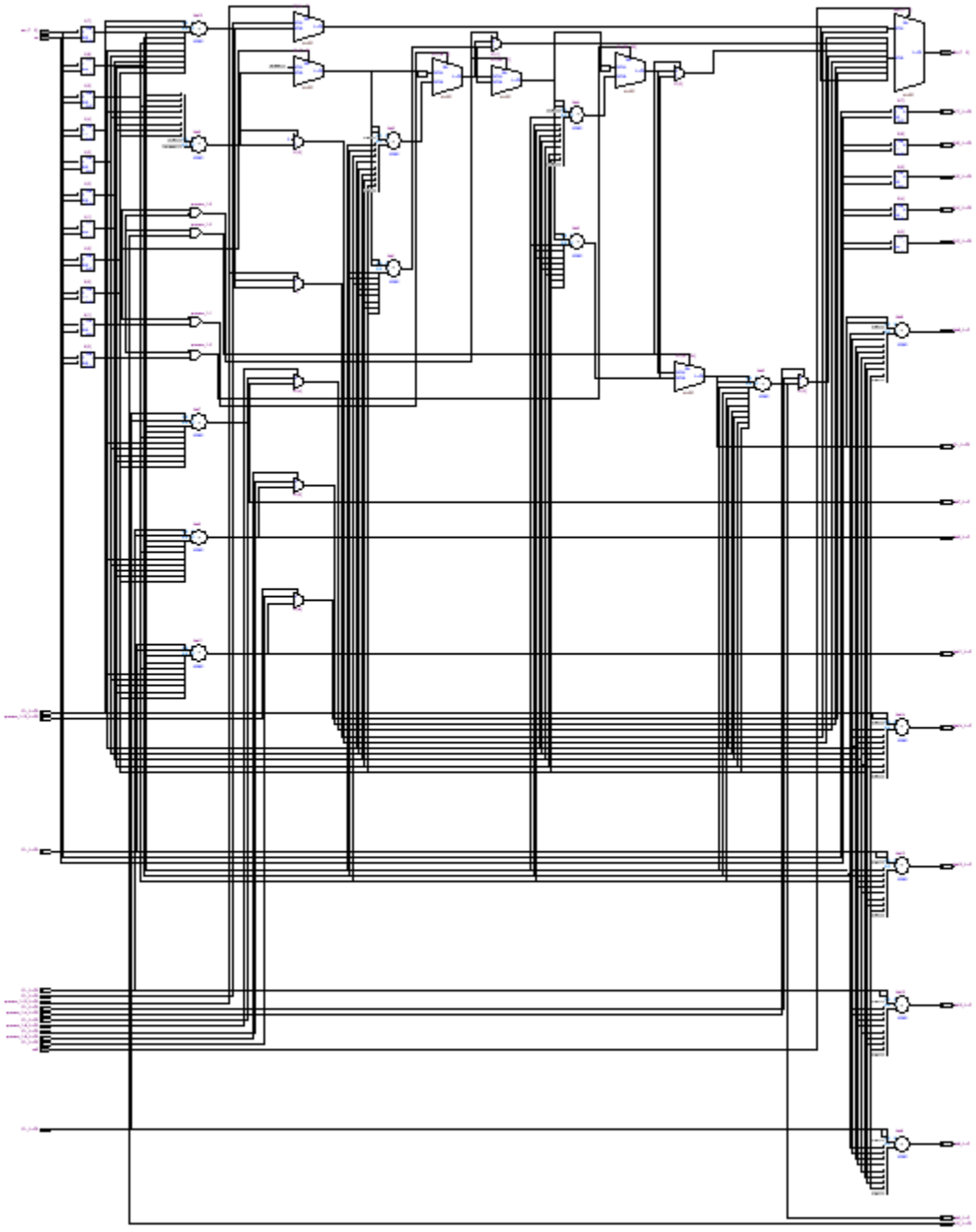


Figura 6.2 Diagrama RTL del multiplicador basado en el algoritmo de Booth.

Posteriormente, como ya fue explicado en el capítulo anterior, se simularon los circuitos para comprobar su funcionamiento. Algunas de las simulaciones realizadas se encuentran a continuación.

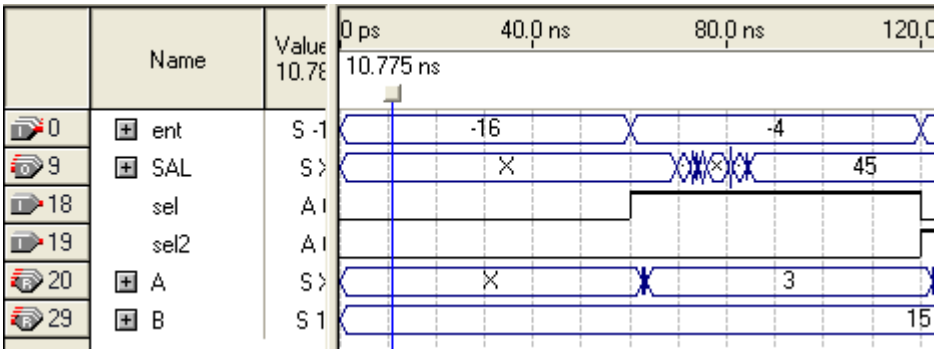


Figura 6.3 Simulación de la multiplicación 15*3=45.

En la figura anterior el valor -16 en el vector “ent” indica la entrada correspondiente al número 15 (en el vector B), con los valores invertidos puesto que la entrada se lee de dicha forma debido a la arquitectura de los componentes externos. De la misma forma el valor -4 corresponde al número 3 (en el vector A). En el vector de salida “SAL” se observa que después de alrededor de unos 20 ns se obtiene el valor adecuado (45).

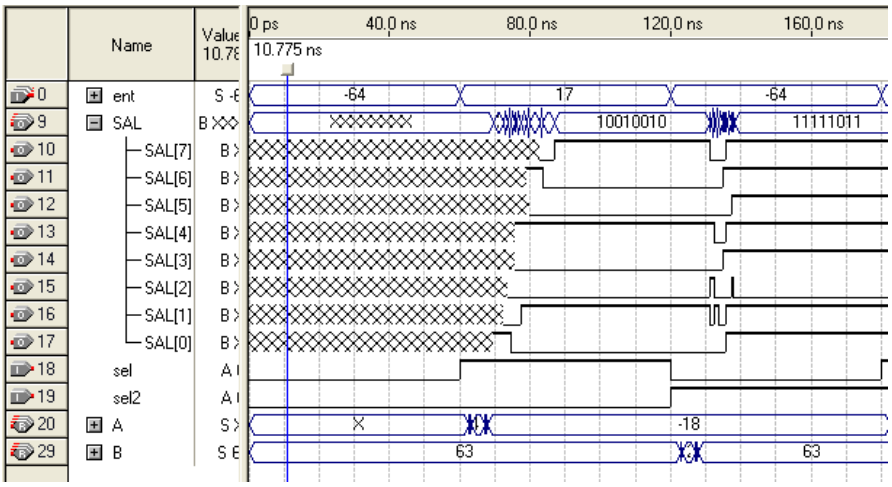


Figura 6.4 Simulación de la multiplicación 63x-18 = -1134.

La figura 6.4 muestra la simulación de la multiplicación 63 x -18, que corresponde a las entradas invertidas -64 y 17 respectivamente. Se observa que en el vector de salida se obtiene primero la parte baja y al activar sel2 se obtiene la parte alta. El número resultante de 16 bits corresponde a -1134 en el formato complemento a 2.

El objetivo principal de este proyecto era realizar un multiplicador de 8x8 bits con lógica programable, que fuera capaz de realizar una multiplicación de forma exitosa en menos de 100 nanosegundos. A continuación se muestran los resultados arrojados por el *timing analyzer* de Quartus, tanto para el multiplicador secuencial basado en sumas y desplazamientos como para el multiplicador de Booth.

Multiplicador	Tiempo [ns]	Tiempo completo [ns]
Secuencial	52	60
Booth	44.6	52.4

Tabla 6.1 Tiempos de ejecución de los multiplicadores.

El tiempo de la primera columna se refiere al multiplicador únicamente, mientras que el tiempo completo hace referencia al sistema que incluye las etapas de multiplexaje de entradas y salidas del multiplicador dentro del FPGA.

Como se puede observar, el objetivo inicial se cumple, se obtuvo un tiempo del sistema, empleando el algoritmo de Booth, de 52.4 ns, que es casi la mitad del tiempo que se quería reducir.

Las siguientes imágenes muestran los oscilogramas del tiempo de respuesta del multiplicador únicamente.

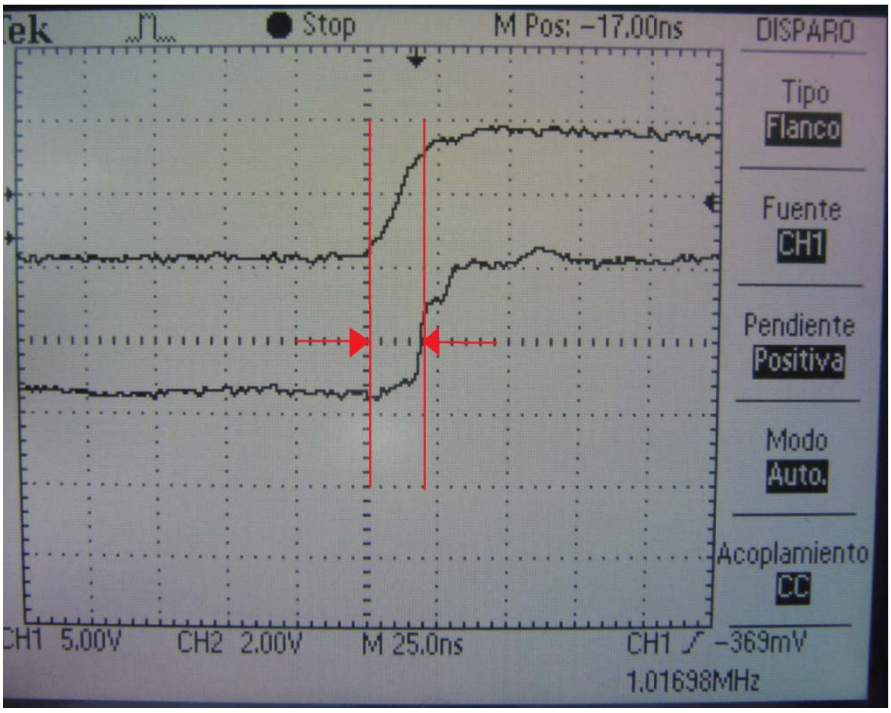


Figura 6.5 Tiempo de retardo entre una señal de entrada y una de salida. La señal superior es un bit de un multiplicando, mientras que la inferior es un bit del resultado.

En la figura anterior se observa, con ayuda de las acotaciones, que el tiempo de ejecución, es decir el tiempo en que tarda en responder el circuito, es de aproximadamente 20 ns. Este experimento muestra el mejor caso, ya que se probó teniendo al multiplicando con un bit encendido (el oscilograma muestra el momento en que se enciende) mientras los demás se mantenían en cero, lo que ocasiona en el algoritmo de Booth que se reduzcan al mínimo la cantidad de sumas a realizar.

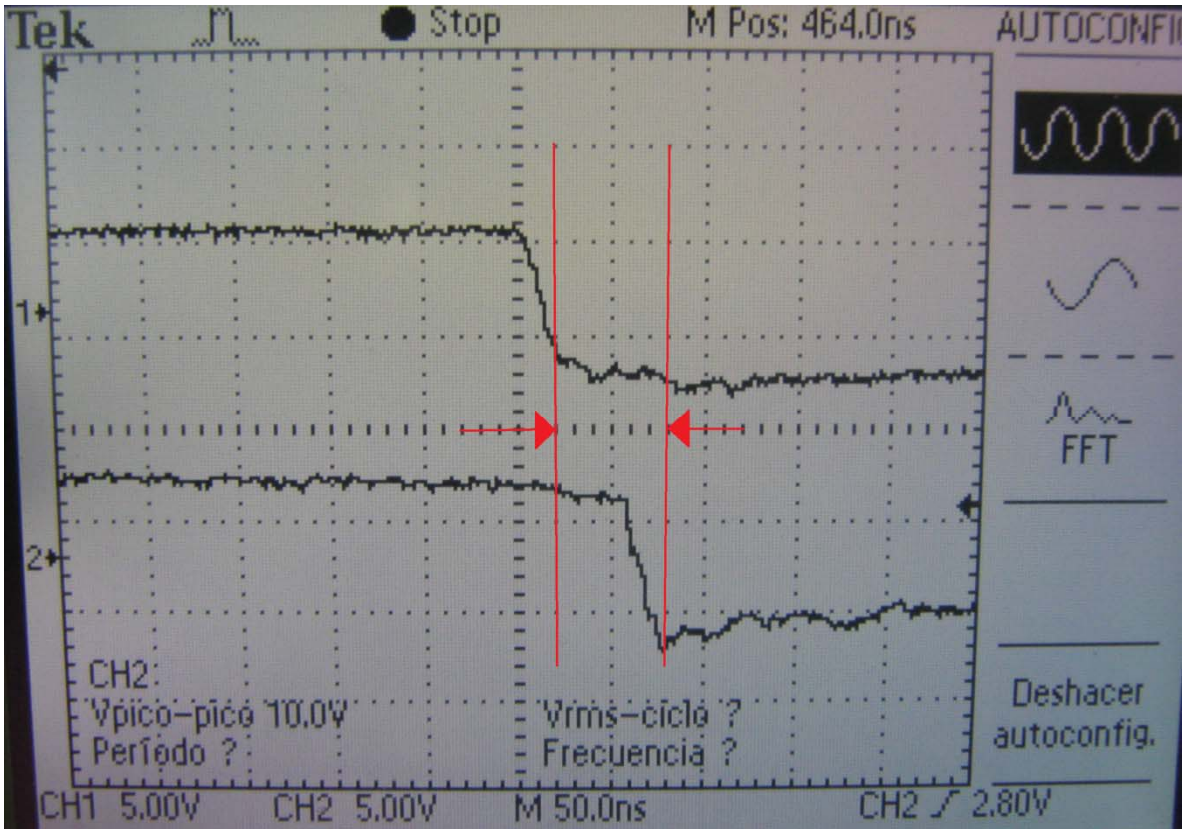


Figura 6.7 Tiempo de respuesta de la multiplicación. La señal superior muestra un bit de un multiplicando mientras que la inferior muestra un bit del resultado.

La figura anterior muestra el tiempo de retardo del peor caso de multiplicación para el algoritmo de Booth. Se observa con ayuda de las acotaciones que dicho tiempo es de aproximadamente 50 ns. Este experimento se toma como peor caso ya que el multiplicando presenta un “1” y un “0” intercalados en los 8 bits, con lo que se fuerza a realizar la mayor cantidad de sumas y restas haciendo más tardada la ejecución de la multiplicación.

El sistema completo, incluyendo el FPGA y el circuito de prueba, funcionó satisfactoriamente, entregando los multiplicandos establecidos en los dip switches y el resultado proveniente del FPGA desplegados en el LCD.

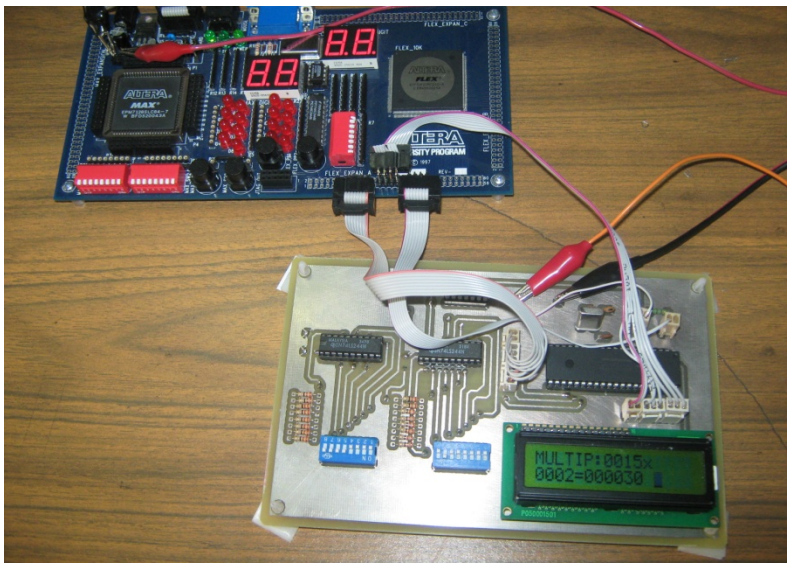


Figura 6.7 Sistema completo con el circuito impreso y la tarjeta UP1.

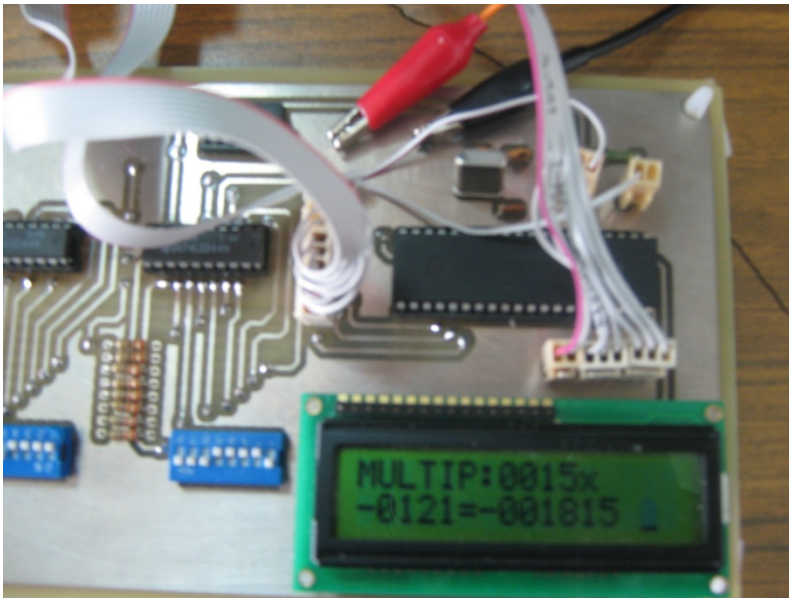


Figura 6.8 Circuito impreso con la multiplicación 15x-121= -1815 en el LCD.

Ejemplo de una aplicación del proyecto realizado

Este proyecto puede ser empleado en aplicaciones que involucren análisis matemáticos que requieran multiplicaciones. En un caso particular, se puede emplear para obtener la correlación entre dos señales. En procesamiento de señales, la correlación cruzada (o a veces denominada "covarianza cruzada") es una medida de la similitud entre dos señales, frecuentemente usada para encontrar características relevantes en una señal desconocida por medio de la comparación con otra que sí se conoce. Es función del tiempo relativo entre las señales, a veces también se la llama *producto escalar desplazado*, y tiene aplicaciones en el reconocimiento de patrones y en criptoanálisis.

La correlación cruzada de dos señales x(n) e y(n) está definida por cualquiera de las siguientes ecuaciones:

$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-1)\ l = 0 \pm 1, \pm 2 \ldots$ Ecuación 7.1

$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n+l)y(n)\ l = 0 \pm 1, \pm 2 \ldots$ Ecuación 7.2

Más específicamente, la correlación puede ser empleada en la caracterización de materiales, mediante la obtención de sus propiedades elásticas como por ejemplo el Módulo elástico de Young. Si se plantea realizar un instrumento electrónico que mida y obtenga dicha propiedad de los materiales, mediante transductores y un microcontrolador que realice las operaciones necesarias para obtener mediciones, convertir las señales previamente acondicionadas a datos digitales y procesarlas para obtener el resultado deseado, el coprocesador realizado en este proyecto puede resultar útil por las siguientes razones:

- Se adapta fácilmente a las necesidades del procesador principal.

- Obtiene las correctas multiplicaciones a velocidades rápidas de tal forma que el procesador principal no pierda tiempo en dichas operaciones y pueda obtener en el tiempo requerido el dato correcto.

Claramente se pueden vislumbrar tres opciones iniciales para confrontar la necesidad del usuario:

- Realizar el sistema completo empleando únicamente un DSP. Es decir, emplear dicho dispositivo como procesador principal y, debido a que los DSPs están diseñados para realizar operaciones de procesamiento de señales a velocidades altas, sería posible realizar la correlación en él mismo.
- Emplear un FPGA en el que se pueda integrar el sistema completo. Para esto se debería conseguir un FPGA que incluya emebedido un núcleo de procesamiento que actúe como procesador principal y contenga además lógica disponible para poder configurar el coprocesador diseñado en este trabajo.
- Diseñar el instrumento de medición deseado haciendo uso de un microcontrolador y un FPGA, éste último como coprocesador, tal y como se plantea en el proyecto realizado.

Tomando en cuenta que se tienen los conocimientos necesarios para realizar el proyecto de las tres maneras, y que todas las opciones logran satisfacer completamente las necesidades del usuario, se podrían evaluar las opciones desde un punto de vista financiero para elegir la mejor opción.

Analizando en primer lugar el uso de un FPGA en el que se pueda implementar el sistema completo, debemos encontrar un FPGA que tenga las capacidades necesarias para tal propósito. Los precios de dispositivos con estas características ascienden a unos cuantos cientos de dólares, llegando incluso a costar miles de

dólares. Ejemplos de éstos son las series Stratix, Arria y Nios de Altera, o las series MicroBlaze y Virtex de Xilinx.

Los DSPs más básicos, que trabajan con palabras de 16 bits, con una velocidad de procesamiento que se encuentra alrededor de los 20 MIPS y contienen el número necesario de entradas y salidas tienen un precio de entre 20 y 60 dólares.

Estas dos opciones probablemente requieran componentes externos para lograr la realización del proyecto, como convertidores analógico-digitales que pueden aumentar por lo menos unos 10 dólares el costo del sistema.

Por último tenemos la opción de emplear el sistema creado en este trabajo, en el que se usa un microcontrolador, capaz de realizar conversiones analógico-digitales y un FPGA que actúa como coprocesador realizando las multiplicaciones necesarias para la correlación, e incluso podría realizarse la correlación completa sin cambiar de dispositivo. El costo del PIC empleado es de 5 dólares, mientras que un FPGA capaz de actuar como coprocesador tiene un precio de 12 dólares.

Es cierto que al tener dos dispositivos la comunicación entre ellos involucra posiblemente capacitancias parásitas en las vías de comunicación que ocasionen que se reduzca la velocidad de procesamiento, sin embargo el sistema funciona correctamente y se observa que es el más viable económicamente.

Conclusiones

En este trabajo se presenta el desarrollo de multiplicadores rápidos descritos por medio de VHDL con la finalidad de sintetizarlos en un FPGA de tal forma que actúen como coprocesador de un microcontrolador PIC.

Los resultados demuestran que se logró el objetivo de reducir el tiempo de multiplicación originalmente establecido en 100 nanosegundos un 47.6%, si se toma en cuenta el tiempo del multiplicador basado en el algoritmo de Booth empleando el multiplexaje a la entrada y salida del multiplicador para lograr la correcta comunicación entre el FPGA y el PIC. Si se requiriera únicamente el multiplicador para una aplicación en específico en la que no se tienen tan limitados los pines de multiplicandos y resultado en el procesador principal, se observa que el tiempo de procesamiento de la operación de multiplicación es de 44.6 nanosegundos.

Estos tiempos, arrojados por la herramienta de analizador de tiempos de Quartus, se corroboraron por medio del osciloscopio, lo cual se puede observar en los oscilogramas que se muestran en el capítulo de resultados del presente trabajo.

Como comparación del rendimiento del coprocesador, se realizó una simulación de la mega función LPM_MULT de Altera; el tiempo arrojado por Quartus, sin incluir los módulos de multiplexaje a la entrada y a la salida del sistema, fue de 20 ns. Sin embargo, la megafunción utiliza mucha área del dispositivo lógico, y consume más energía que los módulos hechos por el usuario. Un sistema con funciones predefinidas de Altera, sin embargo, no puede modificarse de forma tan fácil ya que los únicos cambios posibles que se le pueden hacer a dichas mega funciones son aquellos que modifiquen los parámetros bajo los que trabaja la función, mas no la arquitectura interna ya que ésta está protegida por propiedad

intelectual. Por otro lado, no todos los PLDs permiten que se les programen mega funciones, y, en dado caso que se trabaje con Max Plus II, un software de Altera menos avanzado que Quartus, algunas mega funciones no pueden ser utilizadas, cosa que me sucedió en este proyecto.

De igual manera, se comprobó el correcto funcionamiento del sistema completo empleando la tarjeta de prueba conectada a la tarjeta de UP1 de Altera que incluye el FPGA. Para cualquier combinación de números signados de 8 bits, se desplegaba en el display de cristal líquido la multiplicación de éstos, aunada al resultado correcto de la operación.

En la etapa de simulaciones dentro de Quartus, se observa que mientras se ejecuta la multiplicación se tienen una serie de *glitches*[1] que duran básicamente el tiempo en que se tarda el módulo de multiplicación en obtener un resultado, ya que las salidas del multiplicador están conectadas con los pines de salida del FPGA. Sin embargo, si se tiene una correcta sincronía entre el procesador principal y el coprocesador, estos fenómenos no afectan el resultado que sería leído.

[1] Un glitch es un error considerado como una característica no prevista, una falla de poca duración en un sistema.

Bibliografía

- Pardo Carpio, Fernando; **VHDL, Lenguaje para modelado y descripción de circuitos**; Universidad de Valencia, 1997
- Roth Jr, Charles H., Lizy Kurian, John; **Digital Systems Design Using VHDL**, ed Thomson, USA 2008
- Cady, Frederick M.; **Microcontrollers and Microcomputers, Principles of software and Hardware Engineering**, Oxford University Press, 1997
- Angulo Usategui, José María *et al*; **Microcontroladores PIC, diseño práctico de aplicaciones**, McGraw Hill, 2006
- Ordoñez Fernández, Gustavo *et al*; **Diseño de multiplicadores Paralelos de 16 bits para FPGAs**; Universidad del Valle, Cali, Colombia, 2004
- Ramos Lara, Rafael *et al*; **Diseño de multiplicadores Digitales usando PLDs de Altera**; Ediciones Técnicas Rede, España, 2001
- Mendías Cuadros, José Manuel; **Estructura y Tecnología de Computadores**; Universidad Complutense de Madrid, 2001
- Funes, M. *et al*; **Estudio comparativo de multiplicadores secuenciales implementados en FPGAs,** Universidad Nacional del Mar de Plata, Facultad de Ingeniería, 2007

Datasheets

- **Flex 10k Embedded programmable logic device family**, Altera
- **Configuration Devices for SRAM-Based LUT Devices Data Sheet,** Altera
- **Quartus II Clasic Timing Analyzer**, Altera

- **University Program Design Laboratory Package**, Altera
- **Max 7000 Programmable Logic Design Family,** Altera
- **PIC18FXX2 Datasheet**, Microchip
- **JHD162A LCD Datasheet**, ETC
- **74LS244, Octal buffer/line driver with 3-state outputs**, ON Semiconductor
- **74LS04, Hex inverting gates**, Fairchild Semiconductor
- **BCD Routines, AN544 Application Note**, Microchip

Páginas de internet

- **Programmable Logic Arrays,** FPGA Central, http://www.fpgacentral.com/pld-types/pla-programmable-logic-arrays, consulta: Febrero 2010.
- Shobha, K. R., **Programmable Array Logic**, Ramiah Institute of Technology, http://forum.vtu.ac.in/~edusat/vhdl/krs/PROGRAMMABLE_ARRAY_LOGIC_VHDL_eNotes.pdf, consulta: Febrero 2010.
- **Algoritmos y procesadores Aritméticos**, Universidad del País Vasco, http://www.ehu.es/acwruvac/Arki1/5Aritmet/arit2cas.pdf, consulta: Marzo 2010.

Anexos

Descripción en VHDL del multiplicador secuencial basado en sumas y desplazamientos

```
library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY long IS
PORT(ent:IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      sel,sel2:IN STD_LOGIC;
      SAL:OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END ENTITY;

ARCHITECTURE ALGOR OF long IS
SIGNAL inn,a,b,SALH,SALL: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
inn<= not ent;
PROCESS(sel,inn)
BEGIN
      IF SEL='1' THEN a<=inn;
                        ELSE
                              b<=inn;
      END IF;
END PROCESS;

process(a,b)
variable dataa,datab:std_logic_vector (7 downto 0);
variable c:std_logic_vector (15 downto 0):=(others=>'0');
variable cf:std_logic_vector (31 downto 0);
begin

if a(7)='1' then
dataa:=(others=>'1');
else
dataa:=(others=>'0');
end if;

if b(7)='1' then
datab:=(others=>'1');
else
datab:=(others=>'0');
end if;

cf:=c&datab&b;

for i in 1 to 16 loop
      if cf(0)='1' then
```

```
            cf(31 downto 16):=cf(31 downto 16)+(dataa&a);
            end if;
      cf(31 downto 0):='0'&cf(31 downto 1);
      end loop;

--sal<=cf(15 downto 0);
SALH<=CF(15 DOWNTO 8);
SALL<=CF(7 DOWNTO 0);

end process;

PROCESS(SEL2)
BEGIN
      IF SEL2='0'THEN
            SAL<=SALL;
            ELSE
            SAL<=SALH;
            END IF;
END PROCESS;

END ALGOR;
```

Descripción en VHDL del multiplicador basado en el algoritmo de Booth

```
library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY booth IS
PORT(ent:IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      sel,sel2:IN STD_LOGIC;
      SAL:OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END ENTITY;

ARCHITECTURE ALGOR OF booth IS
SIGNAL inn,A,B,SALH,SALL: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
inn<= not ent;
PROCESS(sel,inn)
BEGIN
      IF SEL='1' THEN A<=inn;
                        ELSE
                              B<=inn;
      END IF;
END PROCESS;
```

```
PROCESS(A,B)

VARIABLE C:STD_LOGIC_VECTOR (8 DOWNTO 0):=(OTHERS=>'0');
VARIABLE CF:STD_LOGIC_VECTOR (16 DOWNTO 0);
VARIABLE X,T:STD_LOGIC;
VARIABLE DATAA:STD_LOGIC_VECTOR (8 DOWNTO 0);

BEGIN

IF A(7)='1' THEN
DATAA:='1'&A;
ELSE
DATAA:='0'&A;
END IF;

X:='0';
T:='0';
CF:=C&B;

FOR I IN 1 TO 8 LOOP
      IF CF(0)='0' AND X='1' THEN
      CF(16 DOWNTO 8):=CF(16 DOWNTO 8)+DATAA;
      ELSIF CF(0)='1' AND X='0' THEN
      CF(16 DOWNTO 8):=CF(16 DOWNTO 8)-DATAA;
      END IF;
      T:=CF(16);
      X:=CF(0);
      CF(16 DOWNTO 0):=T&CF(16 DOWNTO 1);
END LOOP;

--SAL<=CF(15 DOWNTO 0);

SALH<=CF(15 DOWNTO 8);
SALL<=CF(7 DOWNTO 0);


END PROCESS;

PROCESS(SEL2)
BEGIN
      IF SEL2='0'THEN
            SAL<=SALL;
            ELSE
            SAL<=SALH;
            END IF;
END PROCESS;

END ALGOR;
```

Programa en lenguaje ensamblador empleado para programar el PIC

```
;*********************************************************************
******

      LIST P=18F452          ;directiva    para    definir    procesador
especifico
      #include <P18F452.INC>  ;Archivo  de  definicion  de  variables  del
procesador

;*********************************************************************
******


;Registros de Configuracion  SINTAXIS AL FINAL DE ESTE ARCHIVO

;  SELECCION DE OSCILADOR:
      CONFIG    OSC = HSPLL             ; OSCILADOR alta velocidad pll
;   CAMBIO DE OSCILADOR:
      CONFIG      OSCS = OFF           ;DESHABILITADO
;   Power-up Timer DE RESET AL ENCENDIDO:
      CONFIG    PWRT = ON        ;HABILITADO
;   RESET Brown-out POR BAJO VOLTAJE:
      CONFIG      BOR = OFF          ;HABILITADO
;   VOLTAJE Brown-out:
      CONFIG    BORV = 45       ; 4.5V
;   Watchdog Timer:
      CONFIG    WDT = OFF       ; HABILITADO
;  POSTESCALADOR DEL Watchdog:
      CONFIG    WDTPS = 128   ; 1:128
;   CCP2 MUX:
      CONFIG    CCP2MUX = OFF      ;DESHABILITADO=(RB3) HABILITADO= (RC1)
;   RESET POR DESBORDE DE Stack:
      CONFIG    STVR = OFF    ; DESHABILITADO
;   PROGRAMACION ICSP CON BAJO VOLTAJE:
      CONFIG    LVP = OFF       ;DESHABILITADO
;   Background Debugger:
          CONFIG  DEBUG = OFF     ;DESHABILITADO
;   PROTECCION DE CODIGO BLOQUE 0:
      CONFIG      CP0 = OFF       ;DESHABILITADO
;   PROTECCION DE CODIGO BLOQUE 1:
      CONFIG      CP1 = OFF       ;DESHABILITADO
;   PROTECCION DE CODIGO BLOQUE 2:
      CONFIG      CP2 = OFF       ;DESHABILITADO
;   PROTECCION DE CODIGO BLOQUE 3:
```

```
      CONFIG      CP3 = OFF      ;DESHABILITADO
;   PROTECCION DE CODIGO SECTOR DE BOOT:
      CONFIG      CPB = OFF      ;DESHABILITADO
;   PROTECCION DE DATOS EEPROM:
      CONFIG      CPD = OFF      ;DESHABILITADO
;   PROTECCION DE ESCRITURA BLOQUE 0:
      CONFIG      WRT0 = OFF     ;DESHABILITADO
;   PROTECCION DE ESCRITURA BLOQUE 1:
      CONFIG      WRT1 = OFF     ;DESHABILITADO
;   PROTECCION DE ESCRITURA BLOQUE 2:
      CONFIG      WRT2 = OFF     ;DESHABILITADO
;   PROTECCION DE ESCRITURA BLOQUE 3:
      CONFIG      WRT3 = OFF     ;DESHABILITADO
;   PROTECCION DE ESCRITURA BLOQUE DE BOOT:
      CONFIG      WRTB = OFF     ;DESHABILITADO
;   PROTECCION DE ESCRITURA DEL BLOQUE DE CONFIGURACION:
      CONFIG      WRTC = OFF     ;DESHABILITADO
;   PROTECCION Datos EEPROM :
      CONFIG      WRTD = OFF     ;DESHABILITADO
;   PROTECCION DE LECTURA DE TABLA BLOQUE 0:
      CONFIG      EBTR0 = OFF    ;DESHABILITADO
;   PROTECCION DE LECTURA DE TABLA BLOQUE 1:
      CONFIG      EBTR1 = OFF    ;DESHABILITADO
;   PROTECCION DE LECTURA DE TABLA BLOQUE 2:
      CONFIG      EBTR2 = OFF    ;DESHABILITADO
;   PROTECCION DE LECTURA DE TABLA BLOQUE 3:
      CONFIG      EBTR3 = OFF    ;DESHABILITADO
;   Boot Block Table Read Protection:
      CONFIG      EBTRB = OFF    ;DESHABILITADO


;***********************************************
;definicion de constantes
;***********************************************

VALOR1      EQU         0x08
VALOR2      EQU         0x09
DATO  EQU         0x0A


;*****************************************************************************
******
;Definicion de Variables en RAM.

            CBLOCK      0x000
            AUX1
            MULA
            MULB
            BCDH
```

```
            BCDL
            CUENTA
            H_BYTE
            L_BYTE
            GTSDFVUH
            R0
            BASURA2
            BFTRS
            R1
            BASURA
            R2
            count
            FSR
            temp
            BIN
            BCD_TEMP
            TEMP1
            TEMP2
            RETRA1
            RETRA2
            RETRA3
            RETRA4
            ENDC


; EJEMPLO DE UN STACK PARA LA INTERRUPCION DE BAJA PRIORIDAD

            CBLOCK      0x080
            WREG_TEMP   ;VARIABLE PARA RESPALDO DE CONTEXTO
            STATUS_TEMP ;VARIABLE PARA RESPALDO DE CONTEXTO
            BSR_TEMP    ;VARIABLE PARA RESPALDO DE CONTEXTO
            ENDC


;*****************************************************************************
******
;DATOS DE EEPROM
; LOS DATOS A PROGRAMAR EN LA EEPROM SE PUEDEN DEFINIR DESDE AQUI

      ;     ORG   0xf00000

      ;     DE    "Test Data",0,1,2,3,4,5,0X23,0XFB

;*****************************************************************************
******
;VECTOR DE RESET
; AL OCURRIR UN RESET EL CODIGO A PARTIR DE 0X0000 SERA EJECUTADO
```

```
            ORG   0x0000

            goto  INICIO            ;BRINCA AL INICIO DEL CODIGO
                                ;ESTE BRICO SE PONE PARA NO COLISIONAR
                                ;CON LOS VECTORES DE INTERRUPCION
                                ;SI NO HAY INTERRUPCIONES SE PUEDE OMITIR EL
SALTO.

;*********************************************************************
******
;VECTOR DE INTERRUPCION DE ALTA PRIORIDAD

            ORG   0x0008

            goto  HighInt           ;BRINCO A LA RUTINA DE INTERRUPCION
                                ;DE ALTA PRIORIDAD

;*********************************************************************
******
; vECTOR DE INTERRUPCION DE BAJA PRIORIDAD
; This code will start executing when a low priority interrupt occurs.

            ORG   0x0018
            goto  LowInt   ;BRINCO A LA RUTINA DE INTERRUPCION DE BAJA
PRIORIDAD


;SI LA MEMORIA DE PROGRAMA ES CRITICA, NO SE USARIAN LOS SALTOS PARA
REDIRIGIR LAS INTERRUPCIONES
;Y LAS RUTINAS COMENZARIAN EN LAS DIRECCIONES DE LOS VECTORES USANDO UN
SALTO PARA LLEGAR A INICIO




;*********************************************************************
******
;INICIO DEL PROGRAMA PRINCIPAL
; EL CODIGO DEL PROGRAMA PRINCIPAL COMIENZA AQUI.

INICIO

            CLRF        TRISD ;PUERTO D SALIDAS (DISPLAY)
            MOVLW 0XFF
            MOVWF TRISC ;PUERTO C ENTRADAS (MULTIPLICANDOS)
            MOVWF TRISB ;PUERTO B ENTRADAS (RESULTADO)
            BCF         TRISA,0     ; PORTA 0 SALIDA (SEL2)
            BCF         TRISA,1     ; PORTA 1 SALIDA (SEL)
```

```
            CALL  INILCD                        ;INICIALIZAR DISPLAY
START BSF         PORTA,1
            CLRF  MULA

            CLRF  MULB
            CLRF  H_BYTE
            CLRF  L_BYTE


AQUI
            BCF         AUX1,0                  ;LIMPIAR BANDERAS DE SIGNO
            BCF         AUX1,1
            BCF         AUX1,2

            BSF         PORTA,1
            CALL  RET100MS
            CALL  RET100MS
            CALL  RET100MS
            CALL  RET100MS


LEEA  MOVF   PORTC,W                ;LEER MULA
            COMF  WREG
            btfsc WREG,7                  ;CHECAR SIGNO DE MULA
            CALL  COM2A
            MOVWF MULA

            CLRF  WREG

            CALL  RET100MS
            CALL  RET100MS


            BCF         PORTA,1

            CALL  RET100MS
            CALL  RET100MS



LEEB  MOVF   PORTC,W                ;LEER MULB
            COMF  WREG
            btfsc WREG,7                  ;CHECAR SIGNO DE MULB
            CALL  COM2B
            MOVWF MULB

      ;     BSF         PORTA,1
            BSF         PORTA,0
```

```
            CALL  RET100MS



LEEH  MOVF   PORTB,W

            MOVWF H_BYTE

            CALL  RET100MS
            CALL  RET100MS


            BCF         PORTA,0

            CALL  RET100MS
            CALL  RET100MS
            CALL  RET100MS



LEEL  MOVF   PORTB,W
            MOVWF L_BYTE

            CALL  RET100MS

            BSF         PORTA,0

            MOVF  PORTB,W
            btfsc WREG,7
            call  COM216

            BCF         PORTA,0



            MOVLW 0X80
            CALL LCDCOM

            MOVLW "M"                           ;MOSTRAR "MULTIPLICACION"
            CALL  LCDDAT
            MOVLW "U"
            CALL  LCDDAT
            MOVLW "L"
            CALL  LCDDAT
            MOVLW "T"
            CALL  LCDDAT
            MOVLW "I"
            CALL  LCDDAT
```

```
            MOVLW "P"
            CALL  LCDDAT
            MOVLW ":"
            CALL  LCDDAT



            BTFSC AUX1,0
            CALL  MENOS                 ;SI  AUX1,0  ESTA  ENCENDIDO  SE
IMPRIME "-"

            MOVFF MULA,BIN
            CALL  BIN2BCD
            MOVF  BCDH,W
            CALL  BYTEDIS
            MOVF  BCDL,W
            CALL BYTEDIS                      ;MOSTRAR MULA


            MOVLW "x"
            CALL  LCDDAT

            MOVLW " "
            CALL  LCDDAT

            MOVLW 0XC0
            CALL  LCDCOM

            BTFSC AUX1,1
            CALL  MENOS                 ;SI  AUX1,0  ESTA  ENCENDIDO  SE
IMPRIME "-"

            CLRF  BIN


            MOVFF MULB,BIN
            CALL  BIN2BCD
            MOVF  BCDH,W
            CALL  BYTEDIS
            MOVF  BCDL,W
            CALL  BYTEDIS                     ;MOSTRAR MULB

            MOVLW "="
            CALL  LCDDAT

            BTFSC AUX1,2
            CALL  MENOS

            CALL  B2_BCD
```

```
            MOVF  R0,W
            CALL  BYTEDIS
            MOVF  R1,W
            CALL  BYTEDIS
            MOVF  R2,W
            CALL  BYTEDIS                ;MOSTRAR RESULTADO

            MOVLW " "
            CALL  LCDDAT

            GOTO START
            NOP
            NOP

            GOTO INICIO
            nop
            nop


;TRABA      GOTO  TRABA


;*********************************************************************
******
;                       SUBRUTINAS
;*********************************************************************
******

MENOS MOVLW "-"
            CALL LCDDAT
            RETURN

INILCD      CLRF  TRISD             ; Puerto D como salida
            MOVLW b'00000000'       ; RS = 0, RW = 0
            MOVWF PORTD
            MOVLW b'00101000'       ; LCD en 4 bits, E=1
            MOVWF   PORTD
            NOP
            NOP                                 ; Minimo 220 ns
            MOVLW b'00100000'       ; E=0
            MOVWF   PORTD

            MOVLW b'00101000'       ; Funtion Set: 4 bits, 2 line, 5x7 dot
            CALL  LCDCOM
            ;MOVLW      b'00001110'       ;         Display         ON/OFF
Control:Display/Cursor ON/Blink ON
        MOVLW     b'00001111'       ;            Display            ON/OFF
Control:Display/Cursor OFF/Blink O
```

```
            CALL  LCDCOM
            MOVLW b'00000001'       ; Clear Display
            CALL  LCDCOM
            RETURN

LCDDAT      MOVWF DATO
            CALL  BUSSY
            MOVLW b'00000100'       ; RS=1, RW=0
            MOVWF   PORTD
            MOVF  DATO, W
            ANDLW b'11110000'       ; Parte Alta del dato en W
            IORLW   b'00001100'           ;  W  con  parte  alta  del  dato,
RW=0, RS=1, E=1
            MOVWF PORTD
            ANDLW b'11110111'
            NOP
            MOVWF PORTD             ; E=0
            SWAPF DATO,W                  ; Inversion Nibble
            ANDLW b'11110000'       ; Parte baja en W
            IORLW   b'00001100'           ;  W  con  parte  baja  del  dato,
RW=0, RS=1, E=1
            MOVWF PORTD
            ANDLW b'11110111'
            NOP
            MOVWF PORTD             ; E=0
            RETURN

LCDCOM      MOVWF DATO
            CALL  BUSSY
            MOVLW b'00000000'       ; RS=0, RW=0
            MOVWF   PORTD
            MOVF  DATO, W
            ANDLW b'11110000'       ; Parte Alta del dato en W
            IORLW   b'00001000'           ;  W  con  parte  alta  del  dato,
RW=0, RS=0, E=1
            MOVWF PORTD
            ANDLW b'11110111'
            NOP
            MOVWF PORTD             ; E=0
            SWAPF DATO,W                  ; Inversion Nibble
            ANDLW b'11110000'       ; Parte baja en W
            IORLW   b'00001000'           ;  W  con  parte  baja  del  dato,
RW=0, RS=0, E=1
            MOVWF PORTD
            ANDLW b'11110111'
            NOP
            MOVWF PORTD             ; E=0
            RETURN
```

```
BUSSY MOVLW 0x00A0
            MOVWF VALOR1
            MOVLW 0x00A0
            MOVWF VALOR2
LOOP  DECFSZ      VALOR1
            GOTO  LOOP
            DECFSZ      VALOR2
            GOTO  LOOP
            RETURN

;*********************************************************************
***
BIN2BCD
            clrf BCDH
            clrf BCDL
BCD_HIGH
            movlw .100
            subwf BIN,f
            btfss STATUS,C
            goto SUMA_100
            incf BCDH,f
            goto BCD_HIGH
SUMA_100
            movlw .100
            addwf BIN,f
            movlw 0x0F
            movwf BCDL
BCD_LOW movlw .10
            subwf BIN,f
            btfss STATUS,C
            goto SUMA_10
            incf BCDL,f
            movlw 0x0F
            iorwf BCDL,f
            goto BCD_LOW
SUMA_10 movlw .10
            addwf BIN,f
            movlw 0xF0
            andwf BCDL,f
            movf BIN,w
            iorwf BCDL,f
            return
;####################################################################


B2_BCD                                          ;CONVERSION  DE  BINARIO  16
BITS A BCD
    bcf      STATUS,C
    CLRF    count
```

```
    movlw    .16
    movwf    count
    clrf     R0
    clrf     R1
    clrf     R2
Loop16
    rlCf      L_BYTE
    rlCf      H_BYTE
    rlCf      R2
    rlCf      R1
    rlCf      R0
    decfsz   count
    goto     adjDEC
    RETURN
adjDEC
    movlw    R2
    movWf    FSR0L
    call     adjBCD
;
    movlw    R1
    movwf    FSR0L
    call     adjBCD
;
   movlw    R0
    movwf    FSR0L
    call     adjBCD
;
    goto     Loop16
;
      ;movlw R2 ; load R2 as indirect address ptr
      ;movwf FSR0L
      ;call adjBCD

      ;incf FSR0, F
      ;call adjBCD

       ;incf FSR0, F
      ;call adjBCD

      ; goto Loop16
adjBCD
   ; movlw    3
    ;addwf    0,W
  ;  movwf    temp
   ; btfsc    temp,3
   ; movwf    0
    ;movlw    30
    ;addwf    0,W
    ;movwf    temp
```

```
    ;btfsc    temp,7
    ;movwf    0
    ;RETURN
    movfF INDF0,WREG
      addlw 0x03
      btfsc WREG,3 ; test if result > 7
      movwf INDF0
      movfF INDF0,WREG
      addlw 0x30
      btfsc WREG,7 ; test if result > 7
      movwf INDF0 ; save as MSD
      return
;###############################################################################
##

COM2A COMF  WREG                    ;RUTINA PARA HACER COMPLEMENTO A 2 DE
A, Y ACTIVAR BANDERA DE SIGNO
            ADDLW 0X01
            BSF         AUX1,0
            RETURN

COM2B COMF  WREG                    ;RUTINA PARA HACER COMPLEMENTO A 2 DE
B, Y ACTIVAR BANDERA DE SIGNO
            ADDLW 0X01
            BSF         AUX1,1
            RETURN

COM216      COMF  L_BYTE,F                      ;RUTINA     PARA     HACER
COMPLEMENTO A 2 DE RESULTADO Y ACTIVAR BANDERA DE SIGNO
            INCF  L_BYTE,F
            BTFSC STATUS,Z
            DECF  H_BYTE,F
            COMF  H_BYTE,F
            BSF         AUX1,2
            RETURN

;*********************************************
;subrutina de escritura de un byte hex a display
;ENTRADA : DATO EN W. SALIDA: NINGUNA
;*********************************************
;SUBRUTINA QUE CONVIERTE DATOS DEL CONVERTIDOR A HEXADECIMAL Y LOS MANDA
AL DISPLAY

BYTEDIS
      CLRF  TEMP1
      CLRF  TEMP2
      MOVWF TEMP1
      SWAPF TEMP1,W
      ANDLW 0X0F
```

```
      MOVWF TEMP2
      MOVLW 0X0A
      CPFSLT      TEMP2
      GOTO  BYTEDIS1
      MOVF  TEMP2,W
      ADDLW 0X30
      CALL  LCDDAT
      GOTO  BYTEDIS2
BYTEDIS1    MOVF  TEMP2,W
      ADDLW 0X37
      CALL  LCDDAT
BYTEDIS2    MOVF  TEMP1,W
      ANDLW 0X0F
      MOVWF TEMP2
      MOVLW 0X0A
      CPFSLT      TEMP2
      GOTO  BYTEDIS3
      MOVF  TEMP2,W
      ADDLW 0X30
      CALL  LCDDAT
      RETURN
BYTEDIS3    MOVF  TEMP2,W
      ADDLW 0X37
      CALL  LCDDAT
      RETURN


BYTEDISHH   MOVWF TEMP1
      SWAPF TEMP1,W
      ANDLW 0X0F
      MOVWF TEMP2
      MOVLW 0X0A
      CPFSLT      TEMP2
      GOTO  BYTEDIS1H
      MOVF  TEMP2,W
      ADDLW 0X30
      CALL  LCDDAT
      RETURN
BYTEDIS1H   MOVF  TEMP2,W
      ADDLW 0X37
      CALL  LCDDAT
      RETURN


BYTEDISLH   MOVWF TEMP1
      MOVF  TEMP1,W
      ANDLW 0X0F
      MOVWF TEMP2
      MOVLW 0X0A
```

```
      CPFSLT      TEMP2
      GOTO  BYTEDIS3L
      MOVF  TEMP2,W
      ADDLW 0X30
      CALL  LCDDAT
      RETURN
BYTEDIS3L   MOVF  TEMP2,W
      ADDLW 0X37
      CALL  LCDDAT
      RETURN


RET20MS
      MOVLW 0X9A
      MOVWF RETRA2,A
      CLRF  RETRA1,A
RET20_1     DECFSZ      RETRA1,F,A
      GOTO  RET20_1
      DECFSZ      RETRA2,F,A
      GOTO  RET20_1
      RETURN

RET100MS
      MOVLW 0X05
      MOVWF RETRA3,A
RET100MS_1  CALL  RET20MS
             DECFSZ    RETRA3,F,A
          GOTO      RET100MS_1
          RETURN

                NOP
                NOP
                RETURN



;*********************************************************************
******
;                 RUTINAS DE INTERRUPCION
;*********************************************************************
******
; RUTINA DE INTERRUPCION DE ALTA PRIORIDAD


HighInt:

;     *** CODIGO DE RUTINA DE ALTA PRIORIDAD VA AQUI ***
```

```
            retfie      FAST


;*********************************************************************
******
; RUTINA DE INTERRUPCION DE BAJA PRIORIDAD


LowInt            movff STATUS,STATUS_TEMP      ;save STATUS register
            movff WREG,WREG_TEMP          ;save working register
            movff BSR,BSR_TEMP            ;save BSR register

;     *** CODIGO DE RUTINA DE BAJA PRIORIDAD VA AQUI ***


            movff BSR_TEMP,BSR            ;restore BSR register
            movff WREG_TEMP,WREG          ;restore working register
            movff STATUS_TEMP,STATUS      ;restore STATUS register
            retfie


;*********************************************************************
******
;FIN DEL PROGRAMA. REQUIERE LA DIRECTIVA END

            END
```