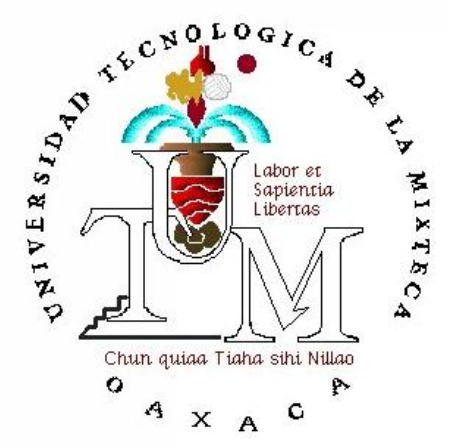


UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"Diseño de módulos para el manejo de Puertos, Temporización e Interrupciones para el Núcleo KCPSM3 e Implementación en el FPGA XC3S500 de Xilinx"

Tesis para obtener el título de
Ingeniero en Electrónica

Presenta
José Peña Toledo

Director de Tesis
M.C. Felipe Santiago Espinosa

Huajuapan de León, Oaxaca
Junio 2008

Tesis presentada el 20 de junio de 2008
ante los sinodales:

Dr. Antonio Orantes Molina
M.C. Enrique Guzmán Ramírez
M.C. Jacob J. Vásquez Sanjuán

Director de Tesis:
M.C. Felipe Santiago Espinosa

Dedicatoria

A mis padres Lucy e Isauro.

José.

Agradecimientos

A mi asesor, el profesor Felipe Santiago Espinosa por su apoyo incondicional en el desarrollo de esta tesis, siempre me pregunté cómo le hacía para soportarme y no ahorcarme a pesar de mis bromas y preguntas incoherentes.

A mis padres, que me apoyaron en toda mi formación académica.

A mis sinodales por la valiosa aportación a este trabajo, así como también a mis maestros de licenciatura.

Pido unas disculpas adelantadas porque me es imposible mencionar todos los nombres de las personas con las que estoy agradecido y que también les dedico la tesis, la lista sería interminable, pero ahí les voy: A mis compañeros de clase: Al Conejo, Wilo, Karo, Selene, Dui, Nacho, Canek, Jahery, Uriel, Wordo, Kisi, Goya, Koreano, Moto, Nasty, Pepito, el Men, Gordis, Rondanas, Chitus, Huasolo, Fabiruchis, Gauss, Kano, Liborio, Chole, Tania, Ursus, Chaquets, Esdras, Bailyss , Gama, Wendo, CENTEC. Mis compañeros de grupo: Hirán P., Tacho, Mary C., Rigo R., Paco T., Jorge L., Jorge Z. Paty G., Alberto D., Pedro H., Leobardo Ch., Giovanni S., Olga S., León M., Bety, José E., Martín L., Manuel M., Lupita S., Eutiquio, Pedro G., Temo, Groover, Bill y Bob, AA y Mov. BVNA. Mis hermanos: Hirán, Luis y Raúl. Mi cuñada Ivonne y mi sobrino David, mis primos: Pablito, Ñito, Alex, Erick, Claudina, Oscar y Fernando. Tíos: Rey, Claudina, Queño y Rosalba. Mis amigas: Marlen, Dalia, Tezo, Zenaida, Dulce Ma. López, Isabel, Anita, Liliana Escobedo, Adriana y Alejandra Budar, Adalí, Cesy, Gaby, Jojo, etc. Varios: Don Beto (mi entrenador de lucha libre), Tino, Pozole, Kano, Lirong, H. Suarez, Moreno, Linares, Heriberto, Felipe, Kike, Tello, Raúl Juarez, Manuel Saldívar, Santana, Tirso, Esteban. Autores e iluminados: Giusseppe Amara, Rius, Osho, Krishnamurti Jiddu, Krishnamurti U.G., Miguel Ruiz, Buda, Quetzalcoatl, Chuchito (Jesucristo), Lao Tse, Sócrates, Nietzsche. Luchadores: Puñetrón, Santo, Blue Demon, Eddie Guerrero, los Villanos, Mil máscaras, Canek, Último Dragón, Rey Misterio Jr., Bas Routen, Evan Pantazi. Capoeira: Giovanni. Yoga: Osho y Alejandro Maldonado. Cuates de la secundaria: Carlos Zaragoza, Carlin, Asun, Zaid, Tuga, Chico, Eliseo, Mario, Ale Vera, Vianey, Lety Peña, Liu, Lola, Liliana, etc. A todo Salina Cruz y sobre todo agradecimientos a mí mismo por mi "gran esfuerzo".

Índice General

Índice de Figuras

Índice de Tablas

Introducción

El núcleo KCPSM3 es de un microprocesador de 8 bits que puede implementarse en dispositivos Spartan-3, Virtex-II o dispositivos más recientes. También es llamado PicoBlaze, es de distribución libre y fue desarrollado por Ken Chapman; ingeniero de desarrollo de Xilinx, Inc. Aunque puede ser usado para procesamiento de datos, generalmente es empleado para aplicaciones que requieren una compleja máquina de estados. Un factor importante es su tamaño, ocupa solo el 3% de los recursos en un dispositivo XC3S500 [1].

Requiere de un único bloque de RAM para formar una ROM en donde es posible almacenar un programa de hasta 1024 instrucciones. Los programas con requerimientos mayores pueden ser redireccionados de manera que utilicen múltiples procesadores KCPSM3, cada uno con su respectivo bloque de memoria, para distribuir tareas en diversos sistemas. Su desempeño es de aproximadamente 43 a 66 millones de instrucciones por segundo (MIPS, *Millions Instruction Per Second*) y el núcleo puede acondicionarse para su síntesis en el lenguaje de descripción de hardware para circuitos integrados de muy alta velocidad (VHDL es el acrónimo que incorpora la combinación de VHSIC y HDL: VHSIC, *Very High Speed Integrated Circuit*; HDL *Hardware Description Language*) y en Verilog.

Su repertorio de instrucciones contiene los comandos suficientes para procesamiento de datos de 8 bits y está formado por 42 instrucciones con que puede ejecutar instrucciones lógicas, aritméticas y de transferencia. Incluye memoria de datos, 16 registros de propósito general, banderas, unidad aritmética lógica (ALU, *Aritmethic Logic Unity*), una interrupción externa, la posibilidad de manejar hasta 256 puertos de entrada y 256 puertos de salida [1].

La macro que se genera del KCPSM3 es proveída como un recurso, un componente de VHDL o Verilog, el cual puede ser instanciado y sometido a simulación. Otra ventaja de ser implementado en un FPGA es que es inmune a ser obsoleto debido a que puede ser reasignado a futuras generaciones de arreglos de compuertas programables en campo (FPGA, *Field Programmable Gate Array*). Evitando así cambiar el hardware como consecuencia de un cambio o actualización en el software. En el FPGA las aplicaciones pueden escribirse de forma que reconfiguren el hardware para satisfacer sus necesidades específicas, asegurando así su funcionamiento óptimo [2].

KCPSM3 es solo un núcleo, para que se pueda ocupar como microcontrolador necesita tener temporizadores para controlar tiempos, sistemas de interrupciones capaces de detectar eventos o sucesos especiales y puertos de propósito general, para resolver una gran parte de las aplicaciones típicas de los microcontroladores.

Si las labores de temporización y conteo de eventos se asignaran al programa principal, restarían mucho tiempo al procesador en atender otras actividades. Por esta razón, el presente trabajo se enfoca al diseño de recursos en VHDL, específicamente para las funciones de temporización, interrupciones y manejo de puertos, con lo que se incrementan las capacidades del núcleo [3]. Empleando una metodología de diseño descendente [19].

Una de las características interesantes es que se obtuvo un microcontrolador totalmente empotrado en un FPGA y no requiere de conexiones externas. El hecho de que cualquier sistema lógico puede ser conectado a un FPGA, significa que adicionalmente se le pueden agregar recursos al núcleo, proporcionando mayor flexibilidad, no sólo el módulo del Picoblaze es versátil, también el ambiente lo es [2].

A pesar de que el microcontrolador resultante es compacto, de bajo costo, con velocidad de respuesta rápida y capacidad considerable de procesamiento de datos, es fácilmente superado por dispositivos comerciales, sin embargo, el presente trabajo persigue fines educativos, específicamente en el área de arquitectura de computadoras. El propósito no es competir con los microcontroladores comerciales o de uso industrial, cada uno tiene sus ventajas y desventajas. Si sólo se dispusiese de un modelo de microcontrolador, éste debería tener muy potenciados todos sus recursos para poderse adaptar a las exigencias de las diferentes aplicaciones. Esta potenciación supondría en muchos casos un derroche.

Los microcontroladores están presentes en muchos de los productos electrónicos que se emplean en la vida cotidiana. Su enseñanza es una necesidad actual debido al auge y a su uso masivo en la industria, a la variedad de modelos existentes en el mercado, a la rapidez con la que surgen nuevas tecnologías y a la gran cantidad de aplicaciones posibles. Sin embargo, a pesar de su diversidad, existe un común denominador en los principios de funcionamiento y las arquitecturas de los microcontroladores. Esta tesis muestra un diseño básico, sencillo y funcional basado en ese común denominador.

Se busca la claridad en la exposición de los conceptos. Para que el alumno adquiera los conocimientos, los elabore, transforme y contribuya con nuevas aportaciones**.** Se busca aportar información útil en torno a los microcontroladores (a sus periféricos asociados, Temporizadores y sistemas de interrupciones), ya que la teoría es compleja y la comprensión de la misma requiere de un alto grado de abstracción por parte del estudiante, esto no lo hace sencillo para la mayoría.

La tesis se dirige especialmente a estudiantes y al personal docente de la electrónica, pero también resultará útil a los lectores interesados en el tema. Una de las áreas de trabajo que contribuye al perfil académico de todo ingeniero, ya sea en la especialidad de informática, electrónica, o de telecomunicaciones, es la de los microcontroladores. Ya que puede abrir las puertas del mercado laboral a muchos egresados.

Objetivos

Objetivo general

Diseñar un conjunto de módulos para el manejo de intervalos de tiempos concretos, un sistema de interrupciones para detección de eventos especiales y puertos de entrada o salida para el núcleo KCPSM3. Implementarlos y evaluarlos en un FPGA Spartan-3E.

Objetivos particulares

1. Evaluar al núcleo KCPSM3

2. Desarrollar un módulo de temporización con las siguientes características:
    - Temporizador y contador de eventos
    - Temporización por comparación
    - Modulación por ancho de pulso (PWM, *Pulse Width Modulation*)

3. Desarrollar un módulo de interrupciones que permita el manejo de:
    - Interrupciones externas
    - Interrupción por comparación
    - Interrupción por desborde

4. Desarrollar un módulo de puertos que incluya:
    - Cuatro puertos de propósito general
    - La multiplexación de algunas terminales para la inclusión de funciones alternas

5. Realizar la implementación y evaluación en el FPGA

6. Documentar los resultados

Organización del Documento

El documento comprende 4 capítulos, el contenido de cada uno de ellos se describe a continuación.

Capítulo 1. Se da una breve descripción sobre VHDL, los FPGA's y seguidamente se particulariza sobre el microprocesador KCPSM3. Se explican las cuestiones propias del tema que son comunes a la mayoría de los microcontroladores como son los puertos, la temporización y las interrupciones.

Capítulo 2. Explica la metodología de diseño empleada para el desarrollo de los módulos de puertos, de temporización y de interrupciones, así como la descripción de los mismos. Detalla la organización del sistema y el uso de registros para el manejo de recursos.

Capítulo 3. Describe las herramientas de desarrollo y la implementación de los módulos, su estructura individual. Así como la integración del sistema.

Capítulo 4. Está dedicado a los resultados y conclusiones obtenidas con los ejemplos de evaluación (generación de intervalos de tiempo, contador de eventos, prueba de PWM). Y se consideran los trabajos futuros.

1. Marco Teórico

Este capítulo contiene una breve descripción de los conceptos fundamentales para la realización del presente trabajo de tesis. Se ofrece una introducción acerca de los dispositivos lógicos programables, la organización del núcleo KCPSM3 y sobre el lenguaje de descripción de hardware para circuitos integrados de muy alta velocidad (VHDL es el acrónimo que incorpora la combinación de VHSIC y HDL: VHSIC, *Very High Speed Integrated Circuit*; HDL, *Hardware Description Language*), lenguaje en el que se diseñaron los módulos de interés. Para comprender el desarrollo de los módulos, se describe el funcionamiento de los periféricos comunes a los microcontroladores (Puertos, Temporización e Interrupciones).

1.1 Historia de los dispositivos lógicos programables

En 1854 George Boole desarrolla y publica el Álgebra de Boole. El sistema de Boole redujo argumentos lógicos a permutaciones de tres operadores básicos algebraicos: AND, OR y NOT [4].

En 1947, John Bardeen, Walter Brattain y William Shockley inventaron el transistor que es un dispositivo electrónico semiconductor que cumple funciones de amplificador, oscilador, conmutador o rectificador. Con el transistor se desarrollan "circuitos lógicos" o "circuitos digitales" que realizan operaciones análogas a las que indican los operadores lógicos [4].

El primer Circuito Integrado fue desarrollado en 1958 por el ingeniero Jack Kilby, era un dispositivo de germanio que consistía en seis transistores en una misma base semiconductora para formar un oscilador [4].

Un sistema digital es cualquier dispositivo destinado a la generación, transmisión, procesamiento o almacenamiento de señales digitales. Cuando se diseñaba un sistema digital, se tenía que realizar la interconexión de varios circuitos lógicos integrados. Esto no representaba un inconveniente cuando se trataba de aplicaciones sencillas, el problema surgía en aplicaciones con un elevado número de circuitos integrados, se elevaba el costo de construcción del circuito y se incrementaba la dificultad para implementar el diseño, un solo error en la interconexión desembocaba en la falla de todo el sistema. El circuito final no podía ser reconfigurado, sólo era útil para una aplicación específica.

En los años setentas surge una solución a la problemática antes descrita, se desarrollan los dispositivos lógicos programables (PLD, *Programmable Logic Device*).

La historia de los PLDs comenzó con las primeras memorias programables de sólo lectura (PROM, *Programmable Read-Only Memory*) que constaban de un arreglo o matriz de compuertas AND fijas y un arreglo OR programable como se muestra en la figura 1.1.

El arreglo AND genera 2^n productos de n entradas, el arreglo OR permite incluir cualquier combinación de términos producto en cada término suma [5].

En cada línea vertical del arreglo AND fijo de la figura 1.1, el símbolo • representa la entrada que forma parte del producto que realiza la compuerta AND. Las líneas horizontales del arreglo OR programable representan las salidas que se pueden programar para un producto de entradas realizadas por cada compuerta AND.

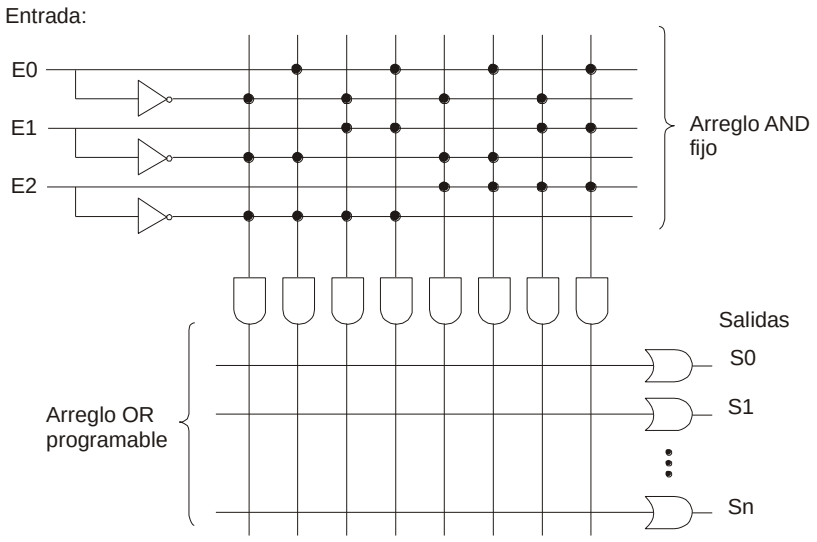


Figura 1. 1 Arreglo AND fijo y OR programable de una PROM [5].

Después se les añadió versatilidad con los arreglos lógicos programables **(**PLA, *Programmable Logic Array*) que contienen un arreglo AND programable seguido de un arreglo OR programable, éstos pueden configurarse para realizar operaciones lógicas AND y OR, esto se puede observar en la figura 1.2.

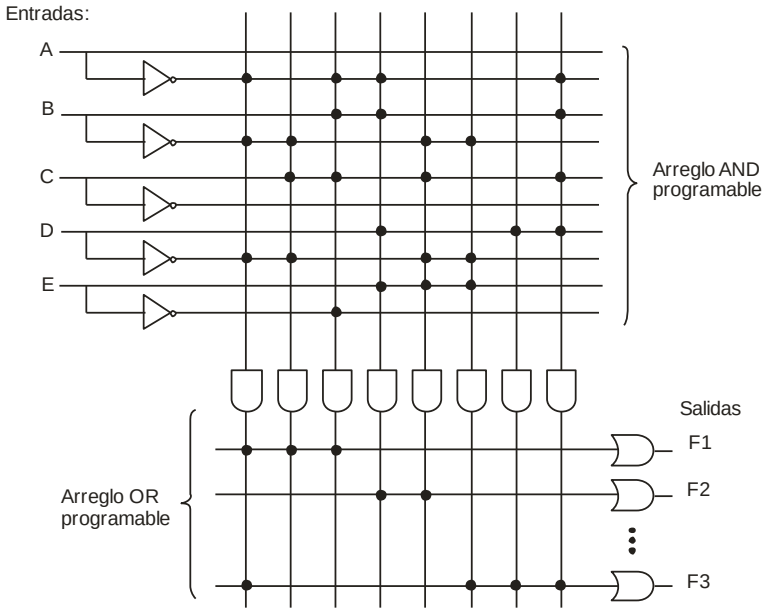


Figura 1. 2 Arreglos AND y OR programables de un PLA [5].

La figura 1.2 muestra los 2 arreglos programables de un PLA en el cual, a manera de ejemplo, el símbolo x representa la selección de entradas programadas que forman parte del producto realizado por la compuerta AND, así como la suma de productos realizados por cada compuerta OR. Los PLA han continuado creciendo en tamaño y potencia. Posteriormente se incluyó la programación utilizando fusibles, antifusibles o celdas de memoria de acceso aleatorio (RAM, *Random Access Memory*) [5].

A finales de la década de los 70's surge el arreglo lógico programable (PAL, *Programmable Array Logic*). El PAL consta de un arreglo AND programable y un arreglo OR fijo, como se muestra en la figura 1.3.

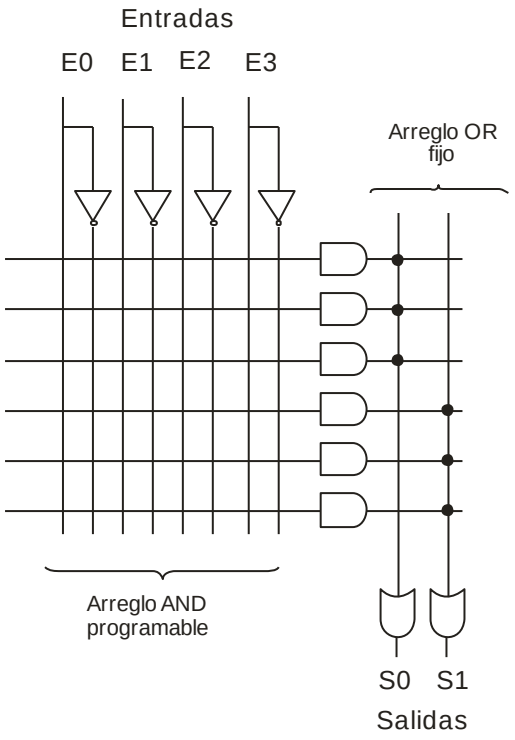


Figura 1. 3 Arreglos AND y OR de un PAL [5].

Para diseños lógicos de mayor complejidad se desarrollaron los circuitos lógicos configurables (CLC, *Configurable Logic Circuit*). Estos se caracterizan por tener recursos lógicos divididos en bloques y por contar con recursos de interconexión entre los bloques lógicos. A todos los dispositivos basados en estas tendencias se les denomina dispositivos lógicos programables.

A partir de matrices lógicas PAL y PLA con un registro de entrada y salida, se dio lugar a los secuenciadores lógicos programables (PLS, *Programmable Logic Secuencer*) los cuales son la base de los PLDs. Los PLDs cuentan con recursos de interconexión concentrados o de manera matricial, un ejemplo se muestra en la figura 1.4.

A partir de las arquitecturas empleadas en los circuitos integrados digitales a semi-medida, los cuales se caracterizan por contar con bloques funcionales que pueden ser configurados

mediante las interconexiones entre ellos, se dió lugar al arreglo de compuertas configurable en campo (FPGA, *Field Programmable Gate Array*).

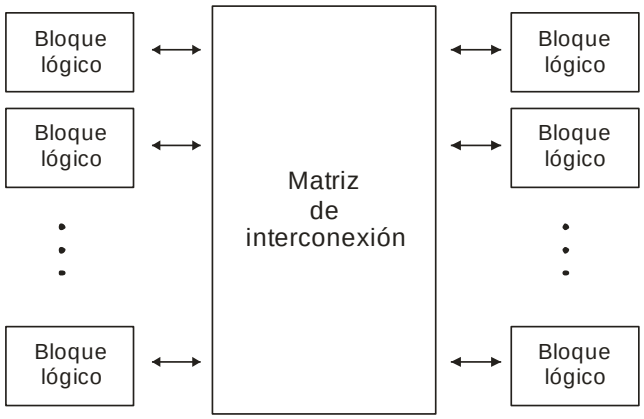


Figura 1. 4 Arreglo matricial de los PLDs [5].

Un FPGA es un circuito integrado que contiene componentes lógicos e interconexiones, ambas programables. Los componentes lógicos pueden ser programados para duplicar la funcionalidad de compuertas lógicas básicas o se pueden crear funciones combinacionales más complejas tales como decodificadores o funciones matemáticas simples [5]. La figura 1.5 muestra los elementos que conforman un FPGA.

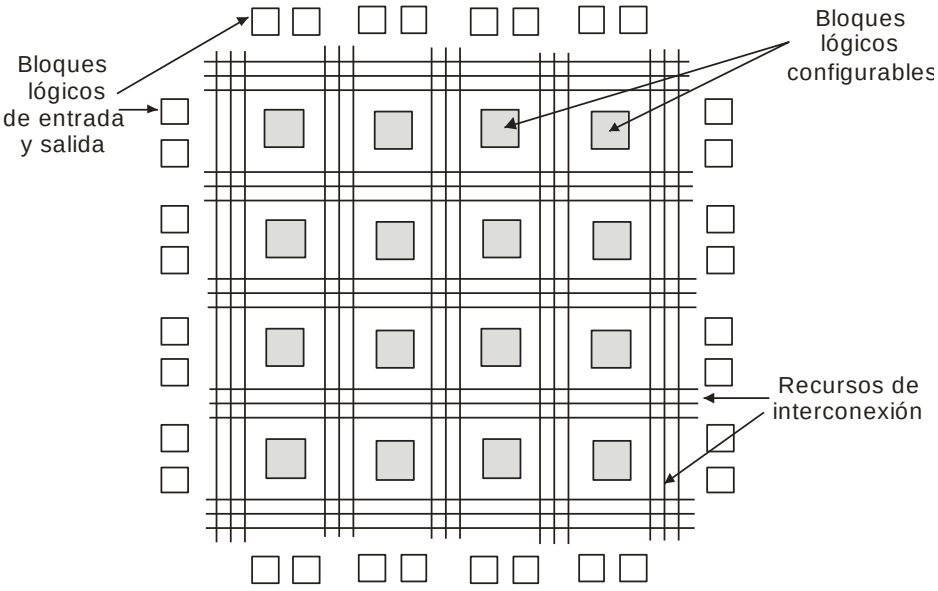


Figura 1. 5 Elementos que contiene un FPGA [5].

También se pueden incluir elementos de memoria, los cuales pueden ser simples flip-flops o bloques de memoria más elaborados. La tendencia de los FPGA's ha sido ahora el integrar procesadores empotrados como el Picoblaze, que puede ser expandido para tener algunos recursos que tienen los microcontroladores.

Las aplicaciones de los FPGA's van desde un sencillo contador hasta complejos sistemas digitales para la industria, investigación, medicina, comunicaciones alámbricas e inalámbricas, procesamiento y almacenamiento de datos, etc.

1.2 Estado del Arte de los Procesadores Empotrados en FPGA's

Un sistema empotrado es un sistema informático (hardware y software) de tiempo real integrado en un sistema o dispositivo mayor, en el que realiza funciones de monitoreo, control o procesamiento. Una de sus ventajas es su bajo costo, puesto que es posible fabricarlos por miles o por millones de unidades. Los sistemas empotrados suelen ser pequeños, de bajo consumo de energía, usan un procesador y una memoria para almacenar el programa para el cual fueron diseñados. Ejecutan un único programa, repetidamente. Reaccionan continuamente a los cambios en el ambiente o sistema en que están inmersos [6].

Los sistemas electrónicos modernos demandan cada día mayor capacidad de cómputo. Por ejemplo, las aplicaciones de red requieren dispositivos capaces de ofrecer un buen rendimiento a velocidades de transferencia del orden de gigabits por segundo. También los sistemas multimedia están aumentando sus necesidades de potencia de cálculo debido principalmente tanto al incremento de sus funciones (codificación/ decodificación de audio y video, conectividad a redes, etc.) como a la aparición de nuevos algoritmos de compresión/descompresión de audio y video que ofrecen mayor calidad a costa de mayor complejidad y costo computacional [6].

Con el aumento de las necesidades computacionales de los sistemas empotrados actuales, los procesadores de propósito general se ven desbordados y se hace necesario buscar soluciones alternativas. Una posible solución es mejorar el procesador que se está utilizando introduciendo mejoras estructurales que permiten obtener un mayor rendimiento y aumentar la frecuencia del reloj, o sustituir el procesador por uno más moderno que ofrezca un rendimiento más elevado. Esta solución presenta algunos inconvenientes, por ejemplo: aumentar la frecuencia del reloj implica un mayor consumo de energía y se necesitaría cambiar el software para adaptarlo al nuevo procesador [6].

Otra solución consiste en desarrollar nuevos coprocesadores aunque se incrementaría el tiempo de desarrollo, lo que llevaría un aumento de su costo en el mercado.

Una tercera solución consiste en utilizar múltiples procesadores en un solo chip (MPSoC, *Multiprocesor System on Chip*), bien de propósito general o específico.

Los FPGA's ofrecen gran cantidad de recursos, millones de compuertas lógicas, bloques de memoria e incluso uno o varios procesadores. Por ejemplo Xilinx ofrece FPGA's de las familias Virtex 2 Pro y Virtex 4 con uno o varios procesadores PowerPC, este tipo de procesadores son implementados en hardaware (HCP, *Hard-Core Processors*). Además de los procesadores HCP, existen los procesadores desarrollados con software (SCP, *Soft-Core Processors*) que se pueden implementar utilizando recursos lógicos del FPGA. Xilinx distribuye 2 procesadores SCP: PicoBlaze y MicroBlaze. La empresa Altera ofrece el

procesador SCP Nios y Nios II. Existe también una serie de procesadores de código abierto como el OpenRISC y el Leon [6].

El rendimiento de los SCP es menor que el que pueden ofrecer los HCP, pero tienen la ventaja de que su número no está fijado de antemano: se pueden utilizar todos los que sean necesarios y sólo se está limitado por la capacidad del FPGA. Otra ventaja es la configurabilidad, ya que se pueden incluir los diferentes módulos con los que se dispongan, como la unidad punto flotante, por ejemplo, lo que permite crear sistemas heterogéneos a pesar de utilizar procesadores de la misma familia [6].

A continuación se mencionan algunas aplicaciones desarrolladas con procesadores empotrados en FPGA's.

1.2.1 DSPuva16

En el Departamento de Tecnología Electrónica en la Universidad de Valladolid, España, se desarrolla el proyecto llamado OpenDSP, se inició en Marzo de 2001. En este Proyecto se diseña en Verilog un Procesador Digital de Señales (DSP, *Digital Processor Signals*), denominado "DSPuva16", que opera con enteros de 16 bits con precisión extendida hasta 24 bits. Tanto el código fuente del procesador en Verilog, como las aplicaciones de desarrollo (ensamblador, simulador, controlador remoto desde PC) y toda la documentación están disponibles en la página web del proyecto [URL1].

El procesador está orientado a aplicaciones de filtrado en las que no es necesario manipular demasiada información simultáneamente. Se han desarrollado aplicaciones de Electrónica de Potencia (convertidores de corriente continua/corriente alterna (CC/CA), control de motores CA, filtros activos, etc.), donde se manejan pocas magnitudes y todas están acotadas.

El DSPuva16 es un procesador de cálculo de 16 bits con arquitectura RISC. Su diseño está orientado a manejar directamente señales de entrada y salida. Dispone de 16 registros internos de 24 bits. Ocupa aproximadamente 250 slices en una Spartan-II de Xilinx [URL1].

1.2.2 CORDIC

A lo largo de los últimos años han cobrado especial interés las técnicas basadas en codiseño en el entorno de los sistemas empotrados. Mediante estas técnicas es posible dividir las tareas a realizar de manera que parte de ellas son implementadas mediante un hardware de propósito específico y otras mediante la programación de un microprocesador de propósito general. De esta manera, las tareas más complejas y críticas se implementan en hardware mientras que aquellas que se adaptan mejor a la ejecución de un microprocesador y en las que se requiere mayor flexibilidad se programan.

Un ejemplo de esta aplicación fue realizado en la Universidad de Sevilla, en procesamiento digital de señales, donde los alumnos desarrollan un pequeño coprocesador de imágenes capaz de rotar una imagen recibida por el puerto serie y enviarla de nuevo al PC. Para ello,

disponen de la especificación VHDL de un procesador 8051, de un módulo transmisor-receptor asíncrono universal (UART, *Universal Asynchronous Receiver-Transmitter*) y de un módulo de rotación digital coordinada por computadora (CORDIC, Cordinate Rotation Digital Computer). Se desarrolló un programa C que previamente compilado es ejecutado por el 8051 empotrado. Para ello deberá actuar con la UART y el CORDIC. El origen de rotación será el centro de la imagen [URL2].

1.2.3 Blackfin H8606

La empresa HV Sistemas implementó el Kit de Desarrollo para Procesador Embebido Blackfin H8606. La unidad H8606 es el módulo de un procesador embebido, basado en un procesador digital de señales, opera a 400MHz. El módulo ocupa poco espacio, es de bajo costo y bajo consumo de corriente, proporciona una gran potencia de procesamiento, siendo capaz de procesar señales de vídeo en tiempo real.

Ha sido diseñado específicamente para utilizar Linux Embebido (uClinux) como sistema operativo, e incorpora una FPGA Xilinx Spartan-3E que puede proporcionar una amplia variedad de periféricos, confiriendo a este módulo una gran flexibilidad y haciéndolo adaptable a una amplia variedad de aplicaciones. El código del FPGA se carga durante el proceso de arranque desde una memoria Flash [URL3].

1.2.4 MicroBlaze

El núcleo MicroBlaze es de un procesador de 32 bits, optimizado para su implementación en FPGA's. Contiene 32 registros de propósito general, se le pueden acondicionar módulos para: un controlador Ethernet, interrupciones, temporizador, generador de modulación por ancho de pulso (PWM, *Pulse Width Modulation*), controlador para periféricos genéricos, módulo UART para comunicación tipo serie, compatibilidad para comunicación vía bus serie universal (USB, *Universal Series Bus*), y es compatible con el protocolo controlador de red de área (CAN, *Controller Area Net*) [7], [8].

En la escuela Politécnica Superior de Madrid se desarrolló un prototipo de un robot ápodo modular, constituido por la unión en cadena de 8 módulos iguales, utilizando un procesador MicroBlaze empotrado en un FPGA de Xilinx. Se desplaza en línea recta, por medio de ondas que recorren su cuerpo desde la cola hasta la cabeza. El robot calcula las posiciones de las articulaciones a partir de los parámetros de la onda: forma, amplitud y longitud de onda.

Los robots modulares reconfigurables prometen ofrecer mayor versatilidad, robustez y menor costo. Los FPGAs dotan a los robots modulares de mayor versatilidad, al no depender de un procesador convencional concreto ni de una arquitectura hardware determinada [9].

1.2.5 PicoBlaze

Existen proyectos que se han elaborado con el procesador KCPSM3 e implementados en el FPGA Spartan 3 ó 3E, a continuación se mencionan algunos de ellos.

En la Universidad de Extremadura, en España, se diseña por software un contador ascendente-descendente utilizando el procesador empotrado Picoblaze. A partir de su repertorio de instrucciones se crea un código ensamblador que programa un contador. [URL4]

En la Escuela Politécnica Superior en España, se diseña un cronómetro digital con precisión de centésimas de segundos, utilizando el PicoBlaze, se decodifican los números hexadecimales para poder ser mostrados en un display de 7 segmentos [URL5].

Ken Chapman, ingeniero de desarrollo de la empresa Xilinx, ha desarrollado los siguientes proyectos, implementándolos en la tarjeta de desarrollo *Spartan 3E Starter Board*, manufacturtada por Digilent, Inc.:

- Diseño de un reloj que mide el tiempo en horas, minutos y segundos junto la posibilidad de programar una alarma. El diseño contiene un módulo UART para establecer comunicación serial y se utiliza para observar y/o programar el tiempo y la alarma a través de comandos simples y mensajes; para ello se utiliza el HyperTerminal de Windows. El diseño también puede ser conectado como una UART enlazada a un periférico de un procesador MicroBlaze o PowerPC. La señal de alarma puede ser usada para disparar una interrupción en el procesador principal [10].

- Diseño de una aplicación que proporciona una introducción general a los puertos de entrada analógicos, resalta las funciones básicas del convertidor analógico-digital LTC1407A-1 y del amplificador programable LTC6912-1. El PicoBlaze es utilizado para controlar los dispositivos mencionados y presentar los resultados en el display LCD.[11]

- Proyecto para implementar las funciones básicas del convertidor digital-analógico LTC2624. El dispositivo es controlado por el PicoBlaze para proveer una introducción general a los convertidores analógico-digital. [12]

- Implementación de un contador de frecuencia, capaz de medir frecuencias de hasta 200MHz. Puede ser utilizado para realizar pruebas de equipos o simplemente como práctica introductoria a los osciladores que dispone la tarjeta, los resultados son mostrados en el display LCD [13]

- Diseño de un módulos para mostrar un mensaje rotatorio a través del display LCD y permite controlar los LED’s utilizando switches y push buttons. Todo ello controlado por el PicoBlaze [14].

- Implementación por software de la modulación de ancho de pulso utilizando el procesador KCPSM3, permite manejar 12 canales, 8 canales controlan la intensidad de los leds, los 4 canales restantes pueden ser enviados al conector J4 para ser observadas a través del osciloscopio. El PWM implementado tiene una resolución

de 8 bits y el ciclo de trabajo puede ser determinado a través del HyperTerminal [15].

1.2.6 CompactRIO

Los Controladores de Automatización Programable (PAC, *Programmable Automotion Controller*) se utilizan en aplicaciones que requieren E/S análogas y digitales altamente integradas, procesamiento de punto flotante y conectividad directa a múltiples nodos de procesamiento.

La empresa National Instruments desarrolla el controlador de automatización programable CompactRIO, es un sistema embebido de control y adquisición de datos. Es de tamaño pequeño, robusto y flexible. Se puede usar hardware comercial para construir sistemas embebidos personalizados. El dispositivo es compatible con LabVIEW, donde es posible diseñar, programar y personalizar el sistema embebido con herramientas de programación gráfica.

CompactRIO combina un procesador embebido en tiempo real, un FPGA de alto rendimiento y módulos de E/S intercambiables. Cada módulo de E/S se conecta directamente al FPGA, proporcionando personalización de bajo nivel para temporización y procesamiento de señales de E/S. El FPGA es conectado al procesador vía bus PCI de alta velocidad. Esto representa una arquitectura con acceso abierto a recursos de hardware de bajo nivel.

LabVIEW contiene mecanismos integrados para transferencia de datos para pasar datos desde los módulos de E/S al FPGA y también desde el FPGA al procesador para análisis o procesamiento posterior, registro de datos o comunicación a un servidor conectado en red. Al usar el hardware embebido en el FPGA es posible implementar disparo, sincronización, control y procesamiento de señales analógica y digital.

El sistema embebido tiene un procesador de 400 MHz. LabVIEW ofrece 600 funciones integradas para construir un sistema embebido de hilos múltiples para control, análisis, registro de datos y comunicación en tiempo real. Se puede integrar código C/C++ [URL6].

1.2.7 Super Computadora Maxwell

En la Universidad de Edimburgo, en Escocia, se construye Maxwell. Maxwell es una supercomputadora y utiliza la tecnología de los FPGA’s en vez de los procesadores convencionales. Tiene una eficiencia en el consumo de energía 10 veces superior y 300 veces más rápida que los sistemas equivalentes. Es poco probable su uso comercial debido a la dificultad para programarla, pero ofrece ventajas en aplicaciones militares, medicina, sismología y cálculos financieros [URL7]

1.2.8 PowerPC

PowerPC es un procesador HCP de arquitectura RISC de 32 bits, empotrado en FPGA’s de la familia Virtex, opera a 450MHz. Provee 32 registros de propósito general. Una sencilla

aplicación que se puede implementar es conectar el bus local del procesador con los periféricos tales como el controlador de interrupciones, el puerto serie o el puerto ethernet para procesar la información que se envía o recibe a través de ellos [URL8].

1.2.9 JOP: Un procesador Java Optimizado para Sistemas Empotrados en Tiempo Real

En este trabajo se presenta el diseño de un procesador de Java para tareas en tiempo real. JOP (*Java Optimized Processor*) es la implementación en hardware de la máquina virtual de Java. Fue desarrollado en VHDL e implementado en un FPGA Spartan 3 de Xilinx, aunque el código puede ser implementado en otros FPGA's. Tiene una frecuencia de 83MHz y ocupa 1844 celdas lógicas [URL9].

1.3 VHDL

En el decenio de 1980, los rápidos avances en la tecnología de los circuitos integrados impulsaron el desarrollo de prácticas estándar de diseño para los sistemas digitales. VHDL se creó como parte de tal esfuerzo y se convirtió en el lenguaje estándar industrial para describir circuitos digitales, principalmente porque es un estándar oficial de la IEEE. En 1987 se adoptó la norma original para VHDL, llamada IEEE 1076. En 1993 se adoptó una norma revisada, la IEEE 1164 [16].

VHDL es el acrónimo que representa la combinación de VHSIC y HDL, donde **VHSIC** es el acrónimo de circuito integrado de muy alta velocidad (*Very High Speed Integrated Circuit*) y **HDL** es a su vez el acrónimo de lenguaje de descripción de hardware (*Hardware Description Language*). Es un lenguaje utilizado para el diseño y modelado de sistemas digitales, además de ser un estándar muy usado en los sistemas educativos. Aunque puede ser usado de forma general para describir cualquier circuito, se usa principalmente para programar circuitos integrados de aplicación específica (ASIC, *Application Specific Integrated Circuit*), PLD's, FPGA's, y similares.

Una característica de VHDL es la independencia del hardware y su modularidad o jerarquía, es decir, una vez hecho un diseño, éste puede ser usado dentro de otro diseño más complicado o con otro dispositivo compatible.

Existen otros lenguajes de descripción de hardware, como Verilog o Abel, y junto con VHDL comparten el flujo de diseño que suele ser:

- Definir la tarea o tareas que tiene que hacer el sistema.
- Escribir el programa usando un HDL.
- Comprobar la sintaxis y simular el programa.
- Programar el dispositivo y verificar su funcionamiento.

Los diseños en VHDL cuentan con tres partes principales: la entidad, la arquitectura y los paquetes, esto se muestra en la figura 1.6.

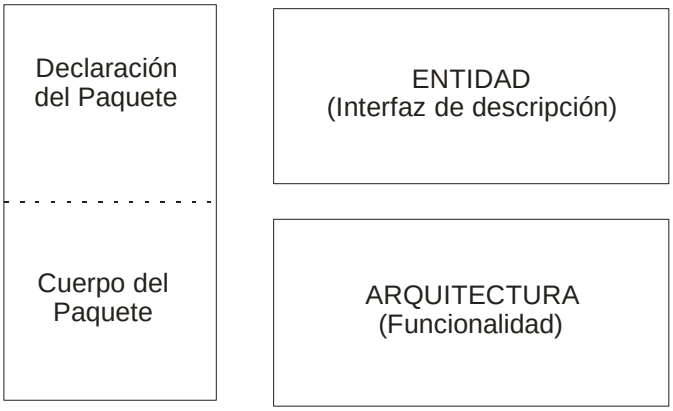


Figura 1. 6 Elementos básicos de un modelo de VHDL [17].

En la **entidad** se definen las entradas y salidas del sistema, es decir, mediante la entidad es posible realizar la interfaz de un sistema a otro.

Una **arquitectura** define la funcionalidad de la entidad a la que está asociada. Describe las operaciones que se efectúan sobre las entradas de la entidad y que determinan el valor de sus salidas en cada momento.

Un **paquete** es una colección de definiciones, tipos de datos, subprogramas, constantes, etc., que se agrupan y se hacen visibles para otros diseños. Los paquetes pueden ser estándares o definidos por el usuario [5].

La figura 1.7 ilustra la relación existente entre la entidad, la arquitectura y los paquetes.

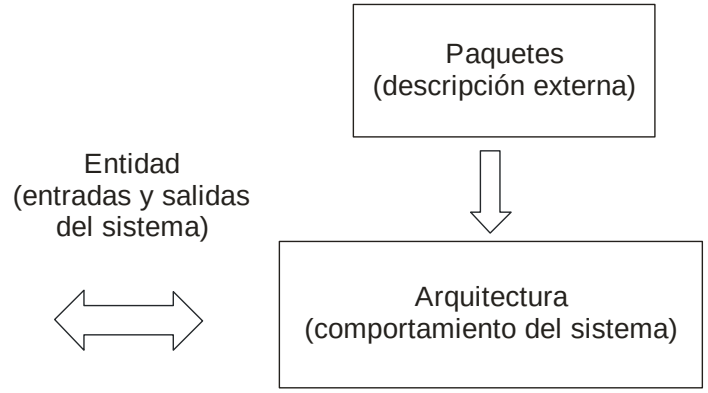


Figura 1. 7 Relación de la entidad, arquitectura y paquetes en VHDL [5].

En VHDL las **señales** se emplean para comunicar bloques o circuitos, representan niveles lógicos y pueden ser individuales o buses. Las señales pueden ser externas (declaradas en la entidad) o internas (declaradas en la arquitectura). Las señales externas se deben especificar de acuerdo a su modo de operación como: entradas, salidas, salidas retroalimentadas o bidireccionales.

En **VHDL** hay diferentes formas de diseñar el mismo circuito y es tarea del diseñador elegir la más apropiada. Esto queda establecido en cada arquitectura, las cuales pueden ser de los siguientes tipos:

- **Funcional**: Se describe la forma en que se comporta el circuito. Similar a los *algoritmos de programación*, ya que la descripción es secuencial. Estas sentencias secuenciales se encuentran dentro de **procesos**. Los procesos son ejecutados en paralelo entre sí, en paralelo con asignaciones concurrentes de señales y con las instancias a otros componentes. Las señales son el medio por el cual se comunican los procesos. Los procesos cuentan con una lista de sensitividad, es decir, una lista de las señales que activan su ejecución.

- **Flujo de datos**: Describe asignaciones concurrentes (en paralelo) de señales.

- **Estructural**: Se describe el circuito con instancias de componentes. Estas instancias forman un diseño de jerarquía superior, al conectar los puertos de estas instancias con las señales internas del circuito, o con puertos del circuito de jerarquía superior.

- **Mixta**: Combinación de todas o algunas de las anteriores [18].

1.3.1 Metodología de desarrollo

Cuando se construye un sistema con VHDL, normalmente se divide el diseño principal en módulos separados, siguiendo las metodologías de diseño descendente (*Top-Down*) ó ascendente (*Bottom-Up*).

La metodología descendente formula un resumen del sistema, sin especificar detalles, luego se propone dividir el sistema en diversos bloques de tal manera que se puedan solucionar los problemas por separado, igualmente, cada bloque a su vez se puede dividir en otras unidades si es necesario. El objetivo es que cada bloque tenga una función específica representada mediante un componente que desempeñe dicha función [19].

El enfoque descendente enfatiza la planificación y conocimiento completo del sistema. Se entiende que la codificación no puede comenzar hasta que no se haya alcanzado un nivel de detalle suficiente.

En contraste, en el diseño ascendente las partes individuales se diseñan con detalle, se caracterizan los componentes básicos del circuito y con estos se forman bloques de mayor tamaño que representen un circuito más complejo que sus partes individuales y luego se enlazan para formar componentes más grandes, que a su vez se enlazan hasta que se forma el sistema completo.

El diseño ascendente se basa en el conocimiento de todas las variables que pueden afectar los elementos del sistema. Hace énfasis en la programación y pruebas tempranas, que pueden comenzar tan pronto se ha especificado el primer módulo. La reutilización de código es uno de los mayores beneficios del enfoque ascendente [19].

Por lo general, para el desarrollo de un sistema con VHDL se utiliza un enfoque descendente. Durante la fase de diseño el proceso de desarrollo es manual, una vez que se ha alcanzado una descripción completa del sistema, se procede a la implementación utilizando herramientas de software.

1.4 El Núcleo KCPSM3

El KCPSM3 ó PicoBlaze es un microprocesador RISC de 8 bits, compacto y de distribución libre, completamente empotrable, optimizado para las familias de FPGA Spartan-3, Virtex-II y Virtex-II Pro. Útil para control y procesamiento de datos simples. Ocupa 91 Slices de un FPGA Spartan 3E. El núcleo requiere de un bloque de RAM, en el que es posible almacenar un programa de hasta 1024 instrucciones, que se cargan automáticamente durante la configuración del FPGA. El microprocesador es flexible, no requiere recursos externos. Su funcionalidad básica puede ser extendida conectándole puertos de entrada y salida u otros recursos, realizados con lógica del FPGA. En la figura 1.8 se observa la conexión del núcleo con su memoria de código.

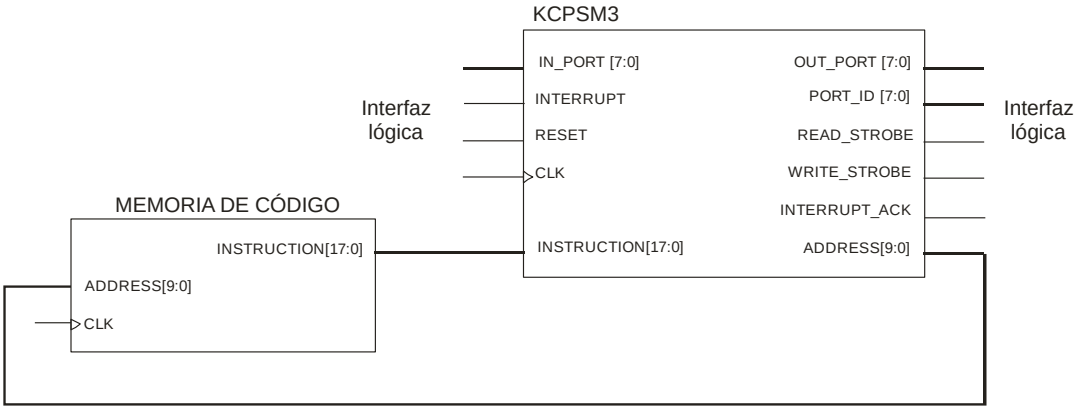


Figura 1. 8 Conexión del núcleo KCPSM3 con su memoria de código [1].

El PicoBlaze es un recurso que se presenta como código VHDL y que puede ser sintetizado en algún FPGA, el núcleo puede emigrarse a futuras arquitecturas de FPGAs, eliminando el riesgo de que la aplicación quede obsoleta, debido a nuevas generaciones de FPGAs. Como el KCPSM3 queda integrado en el FPGA de una tarjeta de desarrollo, reduce la necesidad de espacio físico externo. Además, es posible sintetizar más de un núcleo dentro de un FPGA.

1.4.1 Organización interna

En la figura 1.9 se muestra la organización del núcleo y en las siguientes secciones se describen los diferentes elementos que lo conforman.

1.4.1.1 Registros de Propósito General

Cuenta con 16 registros de propósito general de 8 bits, designados con los nombres de s0 a sF. Los registros pueden ser renombrados usando una directiva de ensamblador. Todas las

operaciones con registros pueden ser intercambiables, no existen registros reservados para tareas especiales ni registros con prioridades. No hay un acumulador dedicado, cada resultado es depositado en el registro que se especifique, según la sintaxis del ensamblador.

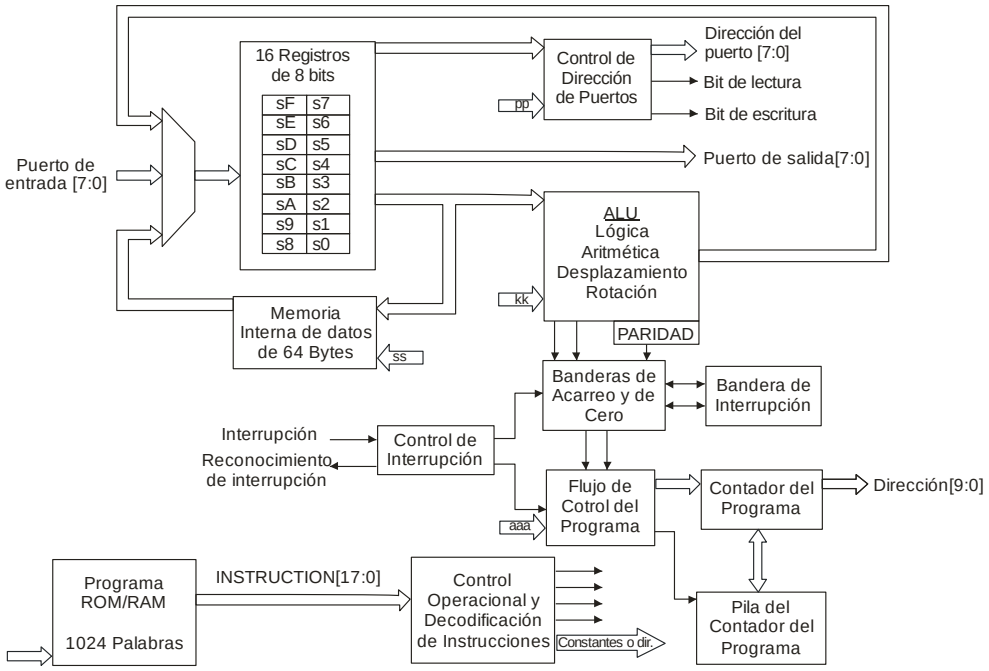


Figura 1. 9 Arquitectura del núcleo KCPSM3 [1].

1.4.1.2 Memoria de Código

Es capaz de almacenar hasta 1024 instrucciones de 18 bits. Todas las instrucciones requieren de 2 ciclos de reloj para su ejecución.

1.4.1.3 ALU

Incluye una unidad aritmética lógica de 8 bits. La cual puede ejecutar las operaciones siguientes:

- Operaciones aritméticas básicas, como adición y substracción.
- Operaciones lógicas tales como AND, OR, XOR entre dos registros.
- Comparación aritmética.
- Rotación y corrimiento de bits.

Las operaciones aritméticas y lógicas se realizan entre registros, o entre un registro y una constante.

1.4.1.4 Banderas

Las operaciones de la ALU afectan las banderas de CARRY y de ZERO. La bandera de ZERO indica cuando el resultado de la última operación resultó en cero. La bandera de CARRY indica diferentes condiciones, dependiendo de la instrucción ejecutada, indica un

sobreflujo para instrucciones aritméticas, la captura de un bit por desplazamientos o rotaciones o bien si un resultado temporal tiene paridad par.

1.4.1.5 Memoria de Datos

El PicoBlaze provee 64 Bytes de memoria RAM para propósito general, la cual se puede accesar directa o indirectamente usando las instrucciones FETCH y STORE.

La instrucción STORE escribe el contenido de uno de los 16 registros a cualquiera de las 64 localidades de memoria RAM. La instrucción FECTH lee el contenido de cualquiera de las 64 localidades de memoria y deposita el valor en uno de los 16 registros. Esto permite a un programa la disponibilidad de un número grande de variables. Para direccionamiento indirecto, se ocupa otro registro como apuntador a la localidad a acceder en la memoria.

1.4.1.6 Puertos I/O

Los puertos de entrada/salida permiten conectar al microprocesador a otros módulos o a los periféricos del FPGA. El microprocesador admite manejar 256 puertos de entrada y 256 puertos de salida, utilizando las siguientes señales:

- PORT_ID: provee la dirección del puerto.
- IN_PORT: Bus de entrada.
- OUT_PORT: Bus de salida.
- READ_STROBE: Activación de una operación de lectura.
- WRITE_STROBE: Activación de una operación de escritura.

La entrada de datos se realiza con la instrucción INPUT, el PicoBlaze coloca en PORT_ID la dirección a leer, genera un pulso en READ_STROBE por un ciclo de reloj, lee el dato del bus IN_PORT y lo transfiere a un registro especificado.

Para la salida de datos se tiene a la instrucción OUTPUT, con ésta el PicoBlaze coloca en PORT_ID la dirección a escribir, coloca un dato en el bus OUT_PORT proveniente de un registro especificado y genera un pulso en WRITE_STROBE por un ciclo de reloj.

Las señales PORT_ID, READ_STROBE y WRITE_STROBE son utilizadas para la decodificación de hardware externo, haciendo posible que diferentes recursos de hardware sean mapeados en diferentes direcciones. PORT_ID es válido por dos ciclos del reloj proporcionando tiempo adicional para decodificación lógica externa. READ_ STROBE y WRITE_ STROBE se proveen en el segundo ciclo de reloj.

En la figura 1.10 se muestra el funcionamiento de las operaciones de entrada y salida, para la escritura se asume que el contenido del registro sA es 42.

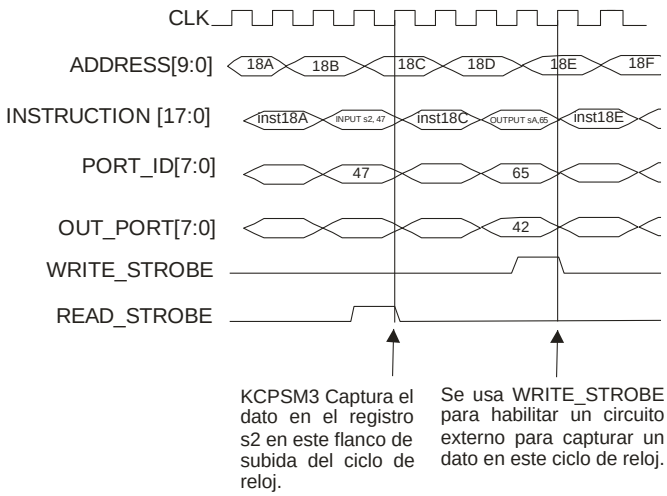


Figura 1. 10 Señales utilizadas para las operaciones de entrada y salida.

1.4.1.7 Contador de Programa (PC, *Program Counter*)

El PC contiene la dirección de la instrucción que va a ser ejecutada. El PC se incrementa automáticamente para apuntar a la localidad de la siguiente instrucción cuando se ejecuta una instrucción.

La secuencia de ejecución de las instrucciones puede ser modificada usando instrucciones de control de flujo condicional o incondicional, la instrucción JUMP puede llevar la ejecución del programa a una dirección específica. La instrucción CALL es para realizar llamadas a subrutinas y RETURN devuelve el flujo del programa a su curso original, antes de que fuera llamada la subrutina.

1.4.1.8 Pila del PC

El KCPSM3 contiene una pila que almacena hasta 31 direcciones, habilitando secuencias de CALL de hasta 31 niveles de profundidad. Se debe considerar que la pila reserva un nivel cuando se usa una interrupción.

1.4.1.9 Interrupciones

El PicoBlaze tiene un módulo hardware para el manejo de interrupciones, permitiendo atender eventos externos asíncronos. Existe sólo una entrada dedicada a la detección de eventos (INTERRUPT) por lo que para el reconocimiento de diferentes eventos, se deberá acondicionar el hardware externo necesario. La respuesta a la interrupción se da después de cuatro ciclos de reloj, y mientras eso ocurre, se pone en alto la señal INTERRUPT_ACK.

1.4.1.10 Reset

El núcleo KCPM3 contiene un circuito de control de *reset* interno para asegurar su correcto inicio. El *reset* está sincronizado con la señal de reloj y dura 4 ciclos. Se ocupa para formar una señal de control que reinicie el sistema.

1.4.2 Repertorio de Instrucciones

El repertorio de instrucciones del núcleo KCPSM3 se muestra en la tabla 1.1 y se compone de 57 instrucciones, que de acuerdo con su función, se organizan en 7 grupos.

GRUPO	INSTRUCCIÓN
Control de Programa	JUMP aaa JUMP Z, aaa JUMP NZ, aaa JUMP C, aaa JUMP NC, aaa CALL aaa CALL Z, aaa CALL NZ, aaa CALL C, aaa CALL NC, aaa
Control de Programa	RETURN RETURN Z RETURN NZ RETURN C RETURN NC

Aritmético	ADD sX, kk
	ADDCY sX, kk
	SUB sX, kk
	SUBCY sX, kk
	COMPARE sX, kk
	ADD sX, sY
	ADDCY sX, sY
	SUB sX, sY
	SUBCY sX, sY
	COMPARE sX, sY

Interrupciones	RETURNI ENABLE RETURNI DISABLE ENABLE INTERRUPT DISABLE INTERRUPT
Lógico	LOAD sX, kk AND sX, kk OR sX, kk XOR sX, kk TEST sX, kk LOAD sX, sY AND sX, sY OR sX, sY XOR sX, sY TEST sX, sY

Tabla 1. 1 Repertorio de instrucciones del núcleo KCPSM3

GRUPO	INSTRUCCIÓN
Acceso a Memoria	STORE sX, ss STORE sX, (sY) FETCH sX, ss FETCH X, (sY)
Desplazamientos y Rotaciones	SR0 sX SR1 sX SRX sX SRA sX RR sX SL0 sX SL1 sX SLX sX SLA sX RL sX
Entrada y Salida	INPUT sX, pp INPUT sX, (sY) OUTPUT sX, pp OUTPUT sX, (sY)

Tabla 1.1. Repertorio de instrucciones del núcleo KCPSM3 (continuación).

La notación empleada para los operandos es:

sX Uno de los 16 registros entre el rango de s0 a sF.
sY Uno de los 16 registros entre el rango de s0 a sF.

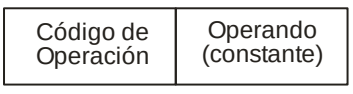
‘kk’ representa un valor constante en el rango de 00 a FF.
‘aaa’ representa una dirección en el rango de 000 a 3FF.
‘pp’ representa la dirección de un puerto en el rango de 00 a FF
‘ss’ Representa una dirección de almacenamiento interno en el rango de 00 a 3F.

En el apéndice A se muestra una referencia rápida al repertorio de instrucciones, mientras que el apéndice B se describe de manera detallada el funcionamiento de cada una de las instrucciones.

1.4.3 Modos de Direccionamiento

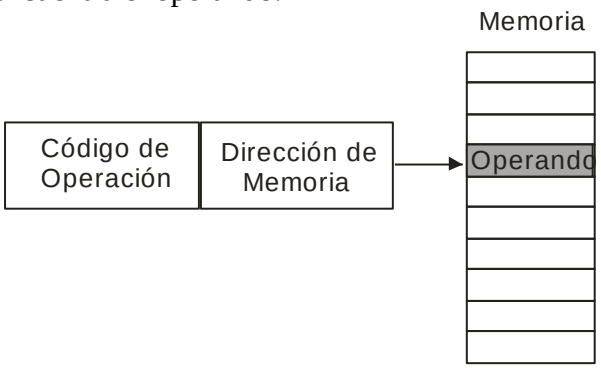
Los Modos de Direccionamiento hacen referencia a las diferentes maneras de especificar un operando dentro de una instrucción. Los modos de direccionamiento que puede manejar el núcleo son:

Direccionamiento Inmediato: En la instrucción está incluido un operando constante, por lo que se conoce de forma inmediata.



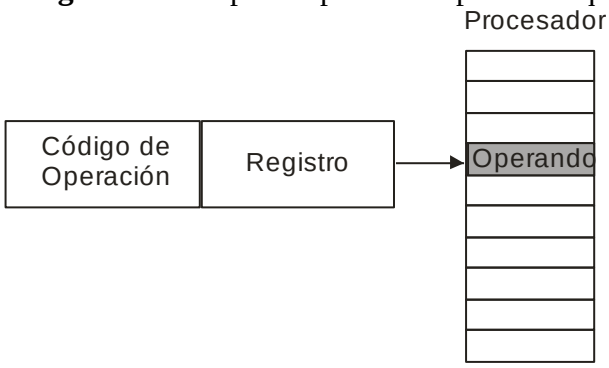
Ejemplos: ADD sX, kk
SUB sX, kk

Direccionamiento Directo: El campo de operando en la instrucción contiene la dirección en memoria donde se encuentra el operando.



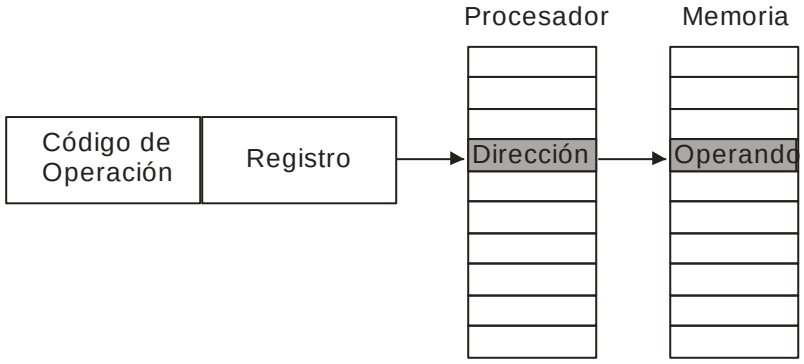
Ejemplo: FETCH sX, ss

Direccionamiento por Registro: Sirve para especificar operandos que están en registros.



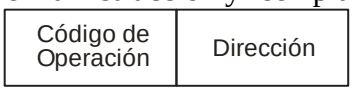
Ejemplos: ADD sX, sY
SUB sX, sY

Direccionamiento Indirecto: El campo de operando de la instrucción contiene un registro en el que se encuentra la dirección efectiva del operando, es decir, un registro funciona como apuntador.



Ejemplos: STORE sX, (sY)
FETCH sX, (sY)

Direccionamiento Absoluto: Es utilizado por las instrucciones de bifurcaciones (JUMP y CALL), la dirección destino está en la instrucción y reemplaza al contador del programa.



Ejemplos: JUMP aaa
CALL aaa

1.5 Periféricos Comunes a los Microcontroladores

Un controlador es un dispositivo que se emplea para el gobierno de uno o varios procesos. Aunque el concepto de controlador ha permanecido invariable a través del tiempo, su implementación física ha variado frecuentemente.

Hace tres décadas, los controladores se construían exclusivamente con componentes de lógica discreta, posteriormente se emplearon los microprocesadores, que se rodeaban con chips de memoria y E/S sobre una tarjeta de circuito impreso.

En la actualidad, todos los elementos del controlador se han podido incluir en un circuito integrado, el cual recibe el nombre de **microcontrolador**. Realmente consiste en un sencillo pero completo sistema contenido en un solo circuito integrado [3].

Puesto que todos los microcontroladores están integrados en un chip, su estructura fundamental y sus características básicas son muy parecidas. Todos deben disponer de los bloques esenciales: Procesador, memoria de datos y de instrucciones, líneas de E/S, oscilador de reloj, módulos controladores de periféricos, etc. Sin embargo, cada diseño intenta enfatizar los recursos más idóneos para las aplicaciones a las que se destinan.

Un periférico es un recurso interno o externo que intercambia datos con el procesador. La comunicación entre el procesador y el periférico está regulada por el procesador de acuerdo con los métodos:

- Sondeo *(Polling)*: El procesador revisa ordenadamente todos los periféricos para atender a cada uno de ellos secuencialmente.

- Interrupciones: El periférico que está listo para ser atendido por el procesador produce una “interrupción” de la ejecución del programa solicitando que el procesador lo atienda.

En los siguientes apartados se describen los recursos comunes a todos los microcontroladores.

1.5.1 Puertos de Entrada/Salida

Cualquier aplicación de un sistema digital basado en un microprocesador o microcontrolador requiere la transferencia de datos entre sí mismo y entre circuitos

externos. Estas transferencias constituyen las operaciones llamadas ENTRADA y SALIDA, (*input /output*) o E/S (*I/O*).

Los puertos son los medios por las cuales el procesador se comunica con los periféricos externos. Proporcionan un canal de datos por el cual circula la información. El procesador genera las señales de control que permiten habilitar a cada uno de sus periféricos. Cuando un periférico es habilitado, éste pone información en el bus de datos y esta información es leída en el puerto correspondiente. Un proceso similar sucede con la escritura, donde el puerto destinado a dicho fin se habilita y la información que se encuentra en las termines pasa al bus de datos, para que sea procesada por algún periférico externo.

Los puertos de entrada/salida básicamente son registros externos o internos. Algunos microcontroladores proporcionan señales de control que permiten que los registros que forman los puertos, ocupen un espacio de direcciones distinto de los registros que componen la memoria. Más sin embargo, la mayoría de microcontroladores mapean a los puertos en su espacio de registros, permitiendo su configuración o el intercambio externo de información, como si se manipulara cualquier registro del microcontrolador.

1.5.2 Temporización

Los temporizadores están presentes en casi todos los circuitos electrónicos y son uno de los recursos comúnmente utilizados por la electrónica de control. Se ocupan para regular el tiempo en un sistema digital para que funcione en un momento dado o en momentos determinados.

Un temporizador básicamente es un registro que funciona como un contador, incrementándose automáticamente en cada flanco de subida de la señal de reloj o cuando ocurre algún evento externo, si la fuente de temporización es externa, se le suele llamar Contador de eventos. Cuando el contador alcanza su máximo valor, en el siguiente evento, ya sea interno o externo, iniciará su cuenta en cero y generará alguna señalización, poniendo en alto una bandera que puede ser evaluada por software o configurada para que genere una interrupción, con lo cual se pueden realizar tareas en intervalos de tiempo preestablecidos o cuando ha ocurrido un número determinado de eventos.

Los temporizadores pueden ser utilizados como base para generar PWM, que es una técnica en la que se modifica el ciclo de trabajo de una señal periódica.

La construcción típica de un circuito PWM se lleva a cabo mediante un comparador con dos entradas y una salida. Una de las entradas se conecta a un temporizador, mientras que la otra queda disponible para depositar el valor a comparar. En la salida la frecuencia es generalmente igual a la de la señal del temporizador y el ciclo de trabajo está en función del valor de comparación.

Otra aplicación similar consiste en la manipulación de la resolución del contador, reiniciándolo cuando hay una coincidencia en la comparación de un valor determinado (que se puede configurar en otro registro) con el contador. Esto es, se realiza la limpieza del temporizador tras una coincidencia en la comparación (CTC, *Clear Timer on Compare*

Match). Algunos microcontroladores permiten la configuración de recursos, haciendo posible la puesta en alto, en bajo o la conmutación de un terminal de un puerto, proporcionando una respuesta automática ante eventos de temporización.

1.5.3 Interrupciones

Una interrupción es una señal que se origina en un dispositivo hardware (por ejemplo, un periférico), para indicar al procesador que un evento externo requiere su atención inmediata; se solicita al procesador que suspenda lo que está haciendo para atender la petición.

Las interrupciones juegan un papel fundamental en la operación de dispositivos E/S, ya que permiten enviar peticiones a la Unidad Central de Procesamiento. Sin las interrupciones el sistema debería monitorear constantemente los dispositivos para comprobar su actividad. Las interrupciones permiten que los dispositivos puedan permanecer funcionando independientemente, hasta el momento en que requieren la atención del procesador, con lo que aparenta que un microcontrolador puede realizar dos o más tareas en forma simultánea.

Las peticiones pueden ser generadas por hardware o por software, interrumpen el flujo de control del programa principal, dando paso a la ejecución de otro segmento de programa típicamente conocido como una rutina de servicio a la interrupción (ISR, *Interrupt service routine*) situado en un lugar predefinido de la memoria de código. Cuando se concluye el servicio de la interrupción, el flujo de control regresa al programa principal.

En algunas arquitecturas se cuenta con una dirección única, a la que bifurcará el programa sin importar el tipo de evento, por lo que lo primero que debe hacer la ISR es la evaluación de diferentes banderas, para determinar el origen de la interrupción. Otras arquitecturas incluyen un vector de interrupciones, en donde cada evento tiene asignada una dirección única, con lo cual se conoce de antemano, por la ubicación de la ISR, la fuente de interrupción.

1.5.4 Convertidor Analógico/Digital

Dispositivo que recibe una señal analógica y la muestrea con cierta frecuencia para generar un valor digital representativo de la señal al momento de la toma de la muestra. Tiene un voltaje de referencia que se utiliza para definir la escala de valores digitales. La salida se presenta como un código de varios bits, estos se leen todos al mismo tiempo. Están diseñados para generar una señal de interrupción cada vez que han concluido una conversión a digital.

1.5.5 Convertidor Digital/Analógico

Transforma los datos digitales obtenidos del procesador en su correspondiente señal analógica, esta señal se envía al exterior mediante un puerto del microcontrolador.

1.5.6 Perro Guardián (Watchdog Timer)

Cuando una computadora se bloquea por un fallo del software u otra causa, se pulsa el botón de reset y se reinicia el sistema. Pero un microcontrolador funciona sin el control de un supervisor y de forma continuada las 24 horas del día. El perro guardián consiste en un temporizador que, cuando se desborda, provoca un reset automáticamente en el sistema.

Se debe diseñar el programa que controla la tarea de forma que refresque o inicialice al perro guardián antes de que provoque el reset. Si falla el programa o se bloquea, no se refrescará al perro guardián y, al completar su temporización, provocará el reset.

1.5.7 Protección ante fallo de alimentación (Brownout)

Se trata de un circuito que reinicia al microcontrolador cuando el voltaje de alimentación (V_{DD}) es inferior a un voltaje de referencia (brownout). El dispositivo se mantiene reiniciado mientras el voltaje de alimentación sea inferior al de referencia, regresando a funcionar normalmente cuando sobrepasa dicho valor.

1.5.8 Comparador Analógico

Algunos microcontroladores disponen internamente de un amplificador operacional que actúa como comparador entre dos señales que se aplican por los puertos de entrada del microcontrolador. La salida del comparador proporciona un nivel lógico alto ó bajo según sea la relación entre ellas.

1.5.9 Modo de Bajo Consumo de Energía

En muchas aplicaciones el microcontrolador debe esperar, sin alguna actividad, a que se produzca algún acontecimiento externo para reanudar su funcionamiento. Para ahorrar energía algunos microcontroladores disponen de una instrucción especial, que los lleva al estado de reposo o de bajo consumo de energía, en el cual los requerimientos de potencia son mínimos. En dicho estado se detiene el reloj principal y se suspenden sus circuitos asociados. Al activarse la interrupción producida por el evento esperado, el microcontrolador reanuda su funcionamiento.

1.5.10 Transmisor-Receptor Asíncrono Universal

La UART es un circuito utilizado en comunicaciones de tipo serie, donde la información es enviada bit por bit. Hay parámetros que son necesarias configurar en los dispositivos que se van a comunicar, como son:

- La velocidad: Medida en baudios o bits por segundo.
- Bit de paridad: Es un bit adicional que se utiliza para prevenir errores en la transmisión. Normalmente ese noveno bit es configurable, y suele estar en función de los otros ocho bits que conforman al byte.

- Bit de paro: Se refiere a la utilización de un bit adicional como separación entre dos bytes consecutivos (bytes que pueden ser de nueve bits, si es que existe bit de paridad).

1.5.11 Controlador de Red de Área (CAN, *Controller Area Network*)

CAN es un protocolo de comunicaciones basado en una topología bus para la transmisión de mensajes en ambientes distribuidos en tiempo real, con un alto nivel de seguridad y multiplexación. Útil en la comunicación entre múltiples procesadores. Este protocolo es comúnmente utilizado en la industria automotriz.

2. Diseño de los Módulos

En el presente capítulo se describe el componente de recursos que se desarrollará así como los tres módulos que lo conforman y las funciones que deben cumplir.

KCPSM3 es un núcleo de un procesador y es un sistema abierto. Para que se pueda ocupar como microcontrolador necesita tener temporizadores para controlar tiempos, sistemas de interrupciones capaz de detectar eventos o sucesos especiales y puertos de propósito general para resolver gran parte de las aplicaciones típicas de los microcontroladores.

Si un procesador no cuenta con recursos de hardware suficientes, algunas tareas, como el monitoreo del estado de dispositivos externos o el manejo de intervalos de tiempo, se deben realizar por software, como parte del programa principal, restándole tiempo de procesamiento al procesador [3].

Es por eso que se propone diseñar recursos en VHDL, específicamente para las funciones de temporización, interrupciones y manejo de puertos. Los recursos deberán ser configurados y evaluados a través de registros, lo que permitirá seguir expandiendo al núcleo sin restarle flexibilidad.

2.1 Metodología de Diseño

De acuerdo con lo expuesto en la sección 1.2.1, durante el proceso de diseño se empleará la metodología descendente, que consiste en dividir el sistema en diferentes bloques que puedan resolver los problemas por separado, cada bloque se puede dividir si es necesario. El objetivo es que cada bloque tenga una función específica representada mediante un componente que desempeñe dicha función [19].

2.2 Organización del Sistema

Se plantea un componente que trabajará al mismo nivel jerárquico que el núcleo y su memoria de código, como se muestra en la figura 2.1. Se aprovecha de la disponibilidad de señales para expansión, las cuales se conservan para el agregado de nuevos elementos de hardware.

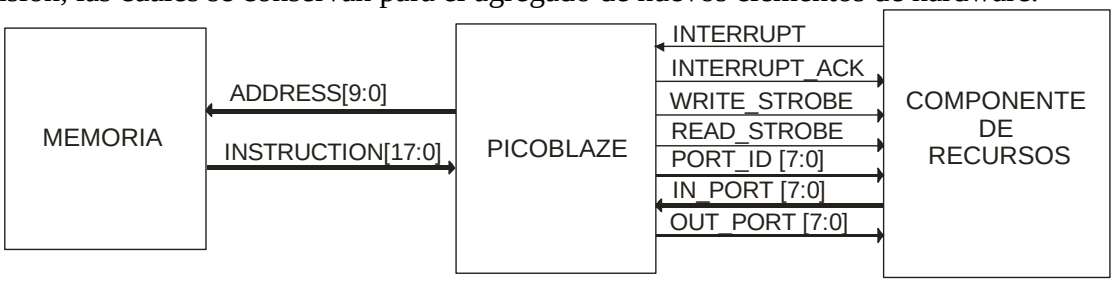


Figura 2. 1 Diagrama de bloques del sistema.

Este componente a su vez estará compuesto por tres módulos: El primero dedicado al manejo de los puertos de entrada y salida. El segundo dedicado al funcionamiento del Temporizador/Contador y el último atenderá a las interrupciones como se muestra en la Figura 2.2.

Figura 2. 2 Diagrama de bloques del componente de recursos.

El módulo de puertos incluirá:

- Cuatro puertos de propósito general.
- La multiplexación de algunas terminales para la inclusión de funciones alternas.

El módulo de temporización tendrá las siguientes características:

- Temporizador y contador de eventos.
- Temporización por comparación.
- Modulación por ancho de pulso (PWM).

El módulo de interrupciones permitirá el manejo de:

- Interrupciones externas.
- Interrupción por comparación.
- Interrupción por desborde.

2.2.1 Módulo de Puertos

Se incluirán cuatro puertos de propósito general, cada uno de ellos contendrá un bloque como el que se muestra en la figura 2.3, en el cual se observa un bus de entrada llamado BITS_IN1 y un bus de salida nombrado BITS_OUT1, el número al final de la etiqueta o nombre de los buses indica el puerto al cual pertenecen, ambos son de 8 bits, éstos son los que comunicarán al microcontrolador con el exterior. Además de este bloque, para cada terminal se requiere de un buffer de tres estados, que se conectará como se muestra en la figura 2.4. Es necesario definir por separado al bloque de cada puerto y a los buffers de tres estados, ya que en el momento de la implementación, los bloques formarán parte de los bloques lógicos configurables (CLB, *Configurable Logic Block*) de un FPGA y los buffers de tres estados se implementan en los bloques de entrada/salida (IOB, *Input output block*).

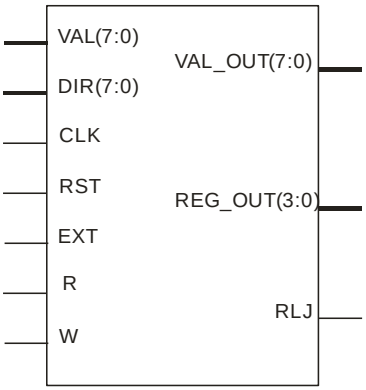


Figura 2. 3 Diagrama del bloque de puertos.

El bus CTRL1 determinará si las terminales de un puerto son de entrada o salida, es decir, maneja la señal de control del buffer de tercer estado colocado en cada terminal de los puertos. Su valor proviene de un registro interno para la configuración.

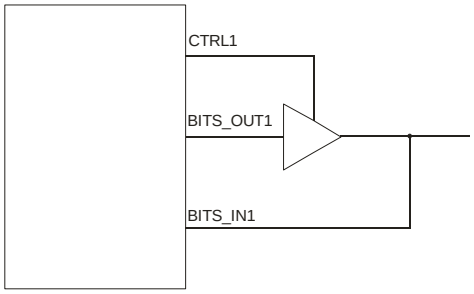


Figura 2. 4 Configuración del módulo de puertos para que pueda ser utilizado como entrada-salida.

Los buses VAL y VAL_OUT son para comunicarse con el núcleo. A través de ellos el núcleo podrá leer y escribir los registros de entrada, salida y configuración.

La entrada DIR leerá la dirección que envíe el procesador, este dato sólo será válido por dos ciclos de reloj. Los pulsos que reciban las entradas R y W son usados para confirmar las operaciones de entrada y salida. RST se ocupará para reiniciar los registros del módulo.

CLK recibirá la señal de reloj para sincronizar al módulo. Cabe aclarar que CLK, RST, DIR, R y W son terminales comunes a varios bloques, por tanto su funcionamiento es el mismo.

Para el manejo de cada uno de los puertos se requiere de la definición de 3 registros de 8 bits, el número al final del registro indica a que puerto pertenece:

- REG_CONF1: Registro para configuración del puerto, definirá individualmente si cada terminal será utilizada como entrada o salida.
- REG_IN1: Registro de entrada, retendrá el último valor leído.
- REG_OUT1: Registro de salida, retendrá el último valor escrito.

Debe considerarse que algunas de las terminales de los puertos estarán multiplexadas para que puedan realizar una función alterna, para: Contador de eventos, PWM, Respuesta automática ante coincidencias de comparación e Interrupción externa. Un multiplexor se encargará de definir como se ocuparán los puertos.

En cada flanco de subida de la señal de reloj, se monitorea si el bit de escritura W está activo y si el valor que se encuentra en el bus DIR es igual a la dirección de uno de los tres registros definidos para el manejo de cada puerto, cuando ambas condiciones ocurran, se deberá escribir el valor del bus VAL en el registro correspondiente.

Se pueden configurar individualmente las terminales de cada puerto (El bit cero configura la terminal 0, el bit uno a la terminal 1, etc.). Cuando el puerto está configurado como salida, el valor que esté depositado en el registro REG_OUT1, se envía a las terminales de salida de BITS_OUT1.

Cuando el puerto está configurado como entrada, los datos presentes en las terminales, automáticamente se capturan en el registro REG_IN1. Si la dirección corresponde a la de un registro y la terminal de lectura R está habilitada, el valor del registro se envía al bus VAL_OUT, en caso contrario, en este bus se presenta una alta impedancia.

El PicoBlaze cuenta con la capacidad de manejo de hasta 256 puertos de entrada/salida u otros recursos externos de hardware, a los cuales se les puede hacer referencia a través de su dirección. Puesto que se dispondrá de 4 puertos de propósito general y cada puerto requiere de 3 registro para su manejo, para el diseño del módulo de puertos de reservarán 12 direcciones, como se muestra en la tabla 2.1.

PUERTO 1		
NOMBRE DEL REGISTRO	DIR. BINARIO	DIR. HEXADECIMAL
REG_CONF1	1111 0000	F0
REG_IN1	1111 0010	F2
REG_OUT1	1111 0001	F1
PUERTO 2		
NOMBRE DEL REGISTRO	DIR. BINARIO	DIR. HEXADECIMAL
REG_CONF2	1111 0011	F3
REG_IN2	1111 0101	F5
REG_OUT2	1111 0100	F4
PUERTO 3		
NOMBRE DEL REGISTRO	DIR. BINARIO	DIR. HEXADECIMAL
REG_CONF3	1110 1101	ED
REG_IN3	1110 1111	EF
REG_OUT3	1110 1110	EE

Tabla 2. 1 Direcciones reservadas para leer y escribir en los registros de los puertos.

PUERTO 4		
NOMBRE DEL REGISTRO	DIR. BINARIO	DIR. HEXADECIMAL
REG_CONF4	1110 1010	EA
REG_IN4	1110 1100	EC
REG_OUT4	1110 1011	EB

Tabla 2.1 Direcciones reservadas para leer y escribir en los registros de los puertos (continuación)

2.2.2 Módulo de Temporización

Este módulo cuenta con tres bloques fundamentales, un contador que es la base para la temporización, un preescalador para obtener rangos mayores de tiempo y un generador de PWM, la relación básica entre estos bloques se muestra en la figura 2.5, a continuación se describirá cada uno de ellos:

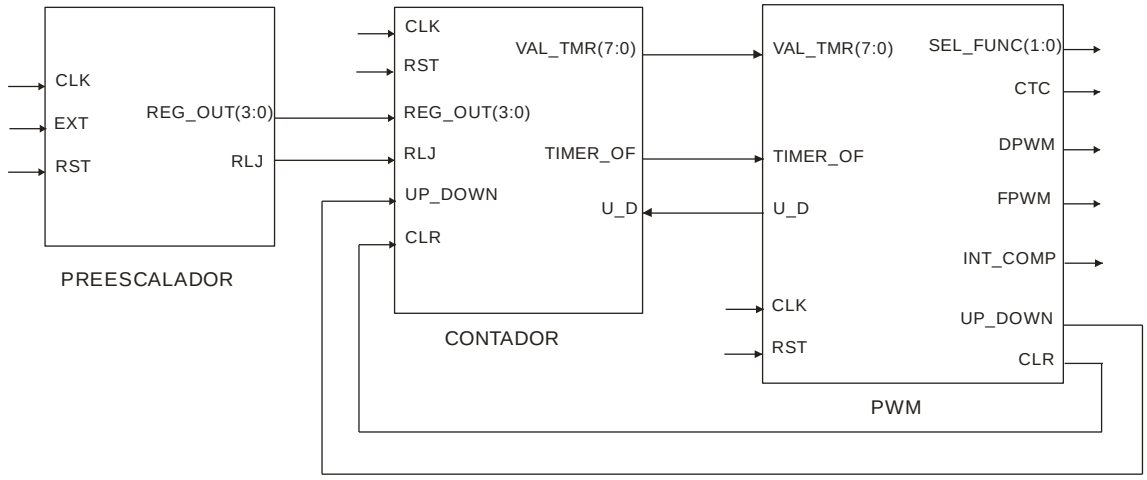


Figura 2. 5 Relación básica entre los bloques del módulo de temporización.

2.2.2.1 Preescalador del Contador

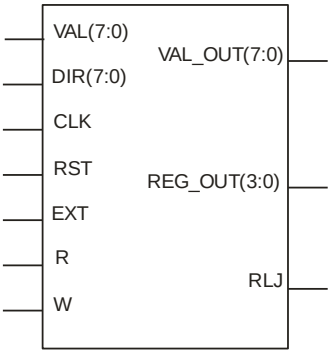


Figura 2. 6 Bloque del preescalador.

Este bloque permitirá alcanzar intervalos de tiempo más grandes, dividiendo la señal del reloj principal entre diferentes factores. En la figura 2.6 se muestran sus entradas y salidas. Las preescalas serán clk/8, clk/32, clk/64, clk/128, clk/256, clk/1024, clk/2^{17}, clk/2^{25} definidas en un registro de configuración. De acuerdo a la configuración también se podrá anular la salida del reloj, permitir su salida sin modificación ó utilizar un reloj externo. La salida del preescalador será proporcionada en la señal RLJ.

La señal de entrada VAL recibirá el valor que se quiera escribir en el registro de configuración y VAL_OUT permitirá leerlo. El registro tendrá una dirección reservada para su acceso, ésta se muestra en la tabla 2.2.

PREESCALADOR		
NOMBRE DEL REGISTRO	DIR. BINARIO	DIR. HEXADECIMAL
VAL_REG	1111 1100	FC

Tabla 2. 2 Dirección reservada para leer y escribir en el registro de configuración del preescalador.

Para lograr las preescalas, se tendrá un contador de 25 bits, del cual se tomará un bit para que funcione como divisor de frecuencia. Dependiendo del valor que se escriba en el registro VAL_REG, se determinara que bit se tomará o si se tomará una fuente externa, la señal de reloj tal cual o si se anulará la salida.

La manera en como se puede configurar al registro se muestra en la tabla 2.3 y la forma en que se organiza el preescalador se observa en la figura 2.7.

VAL_REG	**RLJ**
0000	0
0001	Reloj externo
0010	Reloj principal (clk)
0011	clk/8
0100	clk/32
0101	clk/64
0110	clk/128
0111	clk/256
1000	clk/1024
1001	clk/2^{17}
1010	clk/2^{25}
*Cualquier otro caso	0

Tabla 2. 3 Opciones del registro de configuración VAL_REG.

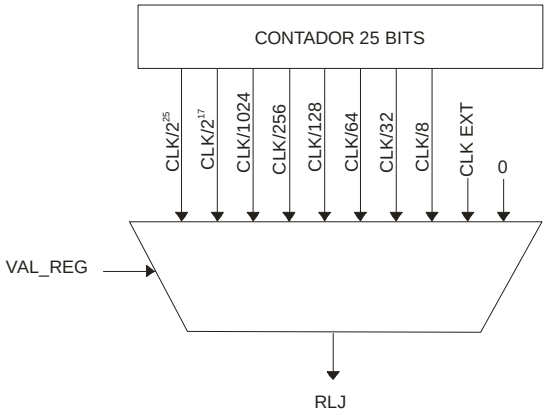


Figura 2. 7 Bloque del preescalador.

La entrada EXT estará designada para recibir la señal de reloj externa. Al pasar por el preescalador, permite un alcance mayor en el conteo de eventos. La terminal REG_OUT envíará los 4 bits menos significativos del registro VAL_REG al módulo del contador para conocer si se ocupa o no, algún factor de preescala.

2.2.2.2 Contador

Un contador es un circuito secuencial construido a partir de flip flops y compuertas lógicas, capaz de realizar el cálculo de los impulsos que recibe en la entrada, almacenar datos o actuar como divisor de frecuencia.

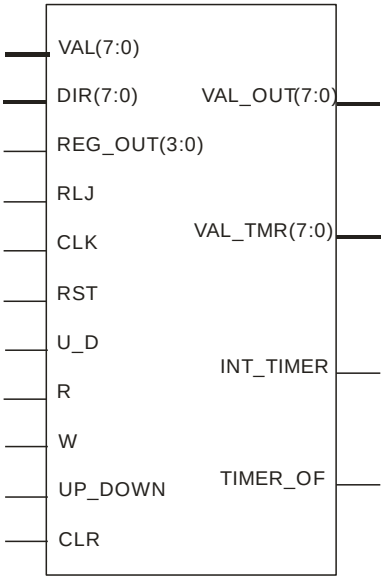


Figura 2. 8 Bloque del contador.

Se implementará un contador ascendente/descendente de 8 bits, su valor inicial será cero y se modificará en cada flanco de subida de la señal de reloj, dependiendo del valor presente en la terminal UP_DOWN. Si el valor es ‘1’ el contador se incrementará, en caso contrario se decrementará. Se podrá modificar el valor del contador escribiendo a través del bus de entrada VAL. Cuando se realice una lectura, el valor se mostrará a través del bus VAL_OUT, si no existe

la operación de lectura se mostrará alta impedancia. Esto es para evitar conflictos con otros datos que se envíen al KCPSM3, ya que si no se mostrara la alta impedancia, el dato quedaría retenido y "chocaría" con datos de otros módulos. El bloque del contador se muestra en la figura 2.8.

El contador tendrá un bit adicional que se pondrá en uno al ocurrir un desborde, este bit se enviará como una señal hacia el módulo de interrupciones, a través de la terminal TIMER_OF.

RLJ recibe la señal que sincronizará todos los procesos internos del contador. Esta señal proviene del preescalador. CLR se utilizará para limpiar contador.

VAL_TIMER envía constantemente el valor del contador para que sea comparado y utilizado en el bloque de modulación por ancho de pulso y de limpieza por comparación. En la tabla 2.4 se muestra el registro para el manejo del contador, al cual se puede tener acceso usando su dirección.

CONTADOR		
NOMBRE DEL REGISTRO	DIR. BINARIO	DIR. HEXADECIMAL
CONT_8B	1111 1111	FF

Tabla 2. 4 Registros del bloque contador.

2.2.2.3 PWM

El bloque PWM, que se muestra en la figura 2.9 se dedicará a generar las funciones especiales de limpieza del contador por coincidencia en comparación, modulación por ancho de pulso de rampa sencilla y rampa doble. Se requerirán 2 registros para implementar dichas funciones.

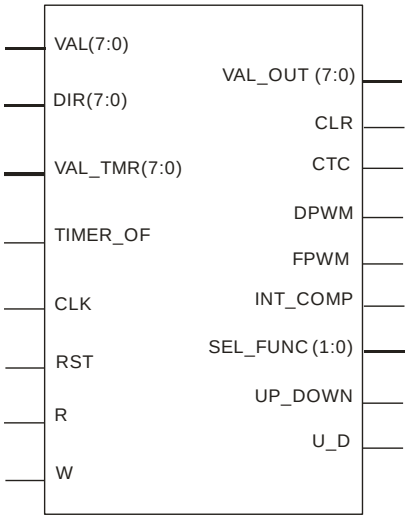


Figura 2. 9. Bloque del PWM.

El bloque comparará el valor del bus VAL_TMR con el que se deposite en el registro VAL_COMP, cuando haya una igualdad se disparará una señal a través de la terminal INT_COMP, que podrá ser utilizado para generar una interrupción por comparación. El registro

CONF_COMP se utilizará para configurar las funciones. Los detalles de los registros se muestran en la tabla 2.5.

PWM		
NOMBRE DEL REGISTRO	DIR. BINARIO	DIR. HEXADECIMAL
VAL_COMP	1111 1001	F9
CONF_COMP	1111 1000	F8

Tabla 2. 5 Registros del bloque PWM.

En el bus VAL se debe depositar el valor que se quiera escribir en cualquiera de los registros internos, lo que distinguirá a los registros será la dirección que se reciba en el bus DIR. Complementariamente VAL_OUT permitirá leer los registros.

El bus VAL_TMR recibirá constantemente el valor del contador, sin necesidad de realizar la lectura con el PicoBlaze. Este valor es el que se ocupará para generar las funciones especiales que se describen más adelante.

La terminal TIMER_OF recibirá la señal del bloque del contador que indica que hubo un desborde.

La terminal de salida CLR se utilizará para enviar una señal que indique al bloque del contador que debe reiniciar su cuenta. La terminal CTC enviará la señal generada por la limpieza por comparación.

La terminal DPWM enviará la señal generada por la modulación por ancho de pulso con doble rampa, mientras que la terminal FPWM enviará la señal generada por la modulación por ancho de pulso con una rampa.

La señal SEL_FUNC sirve para seleccionar si el puerto será de propósito general o de una función específica. Esto se hará utilizando un multiplexor, que al recibir la señal de la terminal SEL_FUNC en “10” permitirá el flujo de la terminal FPWM hacia el puerto de salida, siempre y cuando la terminal CTRL1 esté en ‘1’. Esto se ilustra en la figura 2.7. El mismo principio de operación se aplicará a las funciones de PWM de rampa doble y CTC con sus respectivas señales de selección.

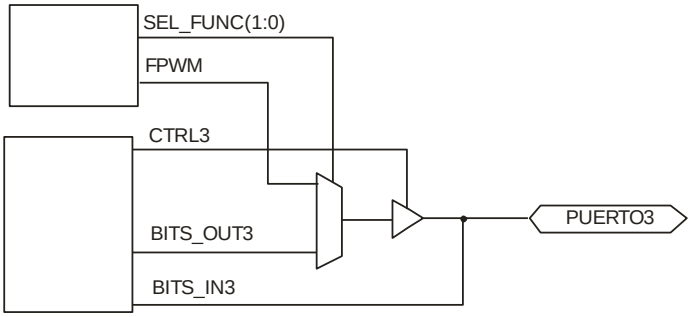


Figura 2. 10 Diagrama de interconexión de los bloques PWM y de puertos.

La terminal UP_DOWN será una señal para controlar al contador para que se incremente o decremente, ya que al generar la función PWM es necesario que el contador sea ascendente y para generar PWM de rampa doble es preciso un comportamiento ascendente/descendente del contador.

2.2.3. Módulo de Interrupciones

Una interrupción es una señal que recibe un microprocesador, indicando que debe "interrumpir" el curso de ejecución actual y ejecutar un código específico para tratar un evento especial, al final de atender la interrupción, el programa retoma su curso original [21]. A continuación se describe como se implementará esta función.

2.2.3.1 Módulo de Interrupciones

El módulo de interrupciones que se muestra en la figura 2.11, contendrá un registro de estado y otro de configuración.

El registro de configuración se utilizará para habilitar o deshabilitar la interrupción por desborde, interrupción por comparación e interrupción externa. Cada interrupción podrá ser habilitada o deshabilitada independientemente y también se tendrá un habilitador global. Se podrá configurar si la interrupción externa será habilitada por un flanco de subida, flanco de bajada, nivel lógico alto o nivel lógico bajo.

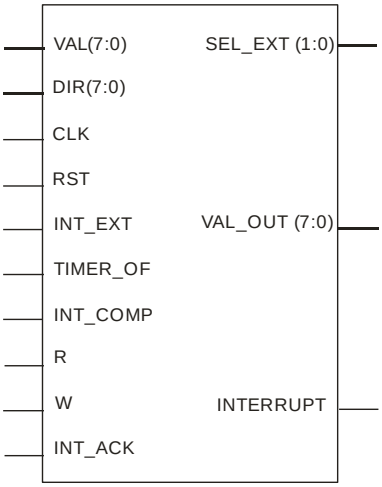


Figura 2. 11 Módulo de interrupciones.

El núcleo incluye una entrada para el manejo de una interrupción externa, se acondicionará por hardware para que pueda ser disparada por diferentes eventos. Por software se determinará la fuente de la interrupción a partir del contenido del registro de estado.

Una interrupción obliga al KCPSM3 a iniciar un llamado a una subrutina localizada en la última localidad de memoria del programa, donde el usuario debe definir un salto a una Rutina de Servicio de Interrupción (ISR, *Interrupt Service Routine*). En este momento, un pulso es generado

en la salida INTERRUPT_ACK del núcleo, para indicar que el procesador está atendiendo al evento, por lo que debe evitarse una nueva interrupción.

Las condiciones para que una interrupción se active serán que el Habilitador Global, el Permiso individual y la Bandera de estado sean iguales a '1'. Bajo cualquier otra condición, no se disparará la interrupción. El bus de entrada VAL se ocupará para escribir en los registros del bloque de interrupciones y el bus VAL_OUT se utilizará para leer el dato contenido en ellos. Las direcciones se muestran en la tabla 2.6.

INTERRUPCIONES		
NOMBRE DEL REGISTRO	DIR. BINARIO	DIR. HEXADECIMAL
INT_CONF	1111 1011	FB
INT_STATE	1111 1010	FA

Tabla 2. 6 Registros del bloque interrupciones.

Las funciones que realiza el registro REG_CONF se describen en la tabla 2.7.

BIT	**DESCRIPCION**
0	Permiso individual para la interrupción por desborde. Activo en alto.
1	Permiso individual para la interrupción externa. Activo en alto.
2	Permiso individual para la interrupción por comparación. Activo en alto.
3	--
4 y 5	00 Interrupción externa activa por flanco de subida. 01 Interrupción externa activa por flanco de bajada. 10 Interrupción externa activa por nivel lógico alto. 11 Interrupción externa activa por nivel lógico bajo.
6	--
7	Permiso global de Interrupción. Activo en alto.

Tabla 2. 7 Descripción del registro de configuración.

El registro de estado indicará los eventos que han ocurrido. Y podrá ser escrito por software o por hardware. Cada evento envía un pulso a la entrada correspondiente del módulo: la terminal INT_EXT recibirá el pulso que indique que ha ocurrido un evento externo. TIMER_OF tomará la señal que indique que ha ocurrido un desborde y INT_COMP capturará el dato que indique que ha habido una coincidencia en la comparación del registroVAL_COMP con el valor del contador. Estas entradas estarán conectadas directamente con el registro de estado. La ubicación de los bits del registro de estado se encuentra en la tabla 2.8.

BIT	**DESCRIPCION**
0	Evento de sobreflujo. Activo en alto por la terminal TIMER_OF.
1	Evento externo. Activo en alto Activo en alto por la terminal INT_EXT.
2	Evento de igualdad en comparación. Activo en alto por la terminal INT_COMP.
3:7	--

Tabla 2. 8 Descripción del registro de estado.

2.2.3.4 SEL_EXT

Se necesita un hardware auxiliar para configurar cómo será la activación por interrupciones externas, esto lo realizará el bloque SEL_EXT que se muestra en la figura 2.9.

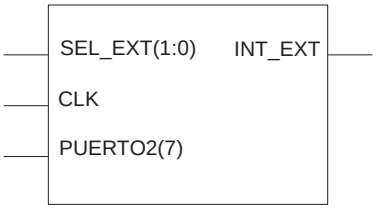


Figura 2. 12 Bloque de interrupciones.

La entrada PUERTO2(7) recibe el estado y eventos de la terminal externa, pero su contenido se evaluará dentro del bloque para ver si es un flanco de subida o de bajada, un nivel lógico alto o bajo, y de acuerdo en el contenido del bus SEL_EXT se determinará si debe o no generarse una interrupción.

2.3 Registros para el Manejo de Recursos

En la tabla 2.13 se presenta un resumen de los registros de 8 bits que se ocuparán para implementar el componente de recursos, se muestra la dirección hexadecimal y binaria correspondiente a cada registro así como una breve descripción. La única excepción es el registro CONT_8B que contiene un noveno bit, al cual no se tendrá acceso mediante la dirección del registro, sino un proceso interno al bloque del contador se encargará de habilitarlo o deshabilitarlo.

Son un total de 18 registros, de los cuales 12 son para el manejo de los puertos, 4 para el módulo del temporizador y 2 para el módulo de interrupciones. En las tablas 2.9-12 se desglosa el significado del nombre de los bits de los registros.

VAL_REG		
BIT	NOMBRE DEL BIT	DESCRIPCION
0	BCP0	Bit 0 de Configuración del preescalador
1	BCP1	Bit 1 de Configuración del preescalador
2	BCP2	Bit 2 de Configuración del preescalador
3	BCP3	Bit 3 de Configuración del preescalador
4	-	-
5	-	-
6	-	-
7	-	-

Tabla 2. 9 Detalles del registro VAL_REG

INT_CONF		
BIT	NOMBRE DEL BIT	DESCRIPCION
0	HIS	Habilitador individual de interrupción por sobreflujo
1	HIE	Habilitador individual de interrupción externa
2	HIC	Habilitador individual de interrupción por comparación
3	-	-
4	BS2	Bit de selección 2 de flanco o nivel para interrupción externa
5	BS1	Bit de selección 1 de flanco o nivel para interrupción externa
6	-	-
7	HGI	Habilitador Global de Interrupciones

Tabla 2. 10 Detalles del registro INT_CONF

INT_STATE		
BIT	NOMBRE DEL BIT	DESCRIPCION
0	BIS	Bandera de Interrupción por sobreflujo
1	BIE	Bandera de Interrupción externa
2	BIC	Bandera de Interrupción por comparación
3	-	-
4	-	-
5	-	-
6	-	-
7	-	-

Tabla 2. 11 Detalles del registro INT_STATE

CONF_COMP		
BIT	NOMBRE DEL BIT	DESCRIPCION
0	-	-
1	CTC	Habilita la función de limpieza del contador por comparación.
2	FPWM	Habilita la función Fast PWM o PWM de rampa sencilla.
3	DPWM	Habilita la función del PWM de doble rampa.
4	-	-
5	-	-
6	-	-
7	-	-

Tabla 2. 12 Detalles del registro CONF_COMP

DIR. HEX	DIR. BINARIO	MODULO	NOMBRE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FF	1111 1111	TEMPORIZADOR	CONT_8B	Registro del valor del contador							
FD	1111 1101	-	-	-	-	-	-	-	-	-	-
FC	1111 1100	TEMPORIZADOR	VAL_REG	-	-	-	-	BCP3	BCP2	BCP1	BCP0
FB	1111 1011	INTERRUPCIONES	INT_CONF	HGI	-	BS1	BS2	-	HIC	HIE	HIS
FA	1111 1010	INTERRUPCIONES	INT_STATE	-	-	-	-	-	BIC	BIE	BIS
F9	1111 1001	TEMPORIZADOR	VAL_COMP	Registro del valor a comparar							
F8	1111 1000	TEMPORIZADOR	CONF_COMP	-	-	-	-	DPWM	FPWM	CTC	-
F7	1111 0111	-	-	-	-	-	-	-	-	-	-
F6	1111 0110	-	-	-	-	-	-	-	-	-	-
F5	1111 0101	PUERTOS	REG_IN2	Registro de entrada del puerto 2							
F4	1111 0100	PUERTOS	REG_OUT2	Registro de salida del puerto 2							
F3	1111 0011	PUERTOS	REG_CONFIG2	Registro de configuración del puerto 2							
F2	1111 0010	PUERTOS	REG_IN1	Registro de entrada del puerto 1							
F1	1111 0001	PUERTOS	REG_OUT1	Registro de salida del puerto 1							
F0	1111 0000	PUERTOS	REG_CONFIG1	Registro de configuración del puerto 1							
EF	1110 1111	PUERTOS	REG_IN3	Registro de entrada del puerto 3							
EE	1110 1110	PUERTOS	REG_OUT3	Registro de salida del puerto 3							
ED	1110 1101	PUERTOS	REG_CONFIG3	Registro de configuración del puerto 3							
EC	1110 1100	PUERTOS	REG_IN4	Registro de entrada del puerto 4							
EB	1110 1011	PUERTOS	REG_OUT4	Registro de salida del puerto 4							
EA	1110 1010	PUERTOS	REG_CONFIG4	Registro de configuración del puerto 4							

Tabla 13. Resumen de los registros empleados en el componente de recursos.

3. Implementación

En el presente capítulo se describe la manera en cómo se llevó a cabo la implementación de los módulos, así como una explicación del código generado.

3.1 Herramientas de desarrollo

3.1.1 Ensamblador KCPSM3

El ensamblador KCPSM3 es proveído como un archivo ejecutable de MS DOS, junto con tres archivos de plantillas. Se copian todos los archivos KCPSM3.EXE, ROM_form.vhd, ROM_form.v y ROM_form.coe dentro del directorio de trabajo que se esté utilizando.

Los programas pueden ser escritos en editores de texto estándares, como el Bloc de notas o el WordPad. Los archivos tienen un límite de 8 caracteres en su nombre y deben ser guardados con la extensión ".psm"

Se abre una ventana de MS DOS y se navega al directorio de trabajo para ejecutar el ensamblador "kcpsm3 <nombre_archivo>.psm", entonces se ensambla el programa generando los archivos:

- <nombre_archivo>.vhd
- <nombre_archivo>.v
- <nombre_archivo>.coe
- <nombre_archivo>.m.

El ensamblador se detendrá tan pronto detecte un error. Un mensaje corto se desplegará para ayudar a determinar la razón del error. El ensamblador también despliega la línea que va analizando cuando detecta el problema. El usuario debe corregir cada error reportado, guardar nuevamente el código y ejecutar nuevamente el ensamblador.

3.1.2 Plantillas

El archivo ROM_form.vhd provee la plantilla para el archivo VHDL generado por el ensamblador y es apropiado para síntesis y simulación. Este archivo debe ser colocado en el directorio de trabajo.

La plantilla suministrada por el archivo ROM_form.vhd define un bloque de memoria RAM para la Spartan-3, Spartan-3E, Virtex-II o Virtex-II PRO, que es configurado para ser utilizado como un bloque de memoria ROM.

El ensamblador lee la plantilla y copia la información en el archivo de salida "nombre_archivo.vhd". No se realiza verificación de sintaxis en esta parte del proceso, así que cualquier alteración del código es responsabilidad del usuario.

Hay dos plantillas para implementar la memoria, aunque ambas tienen el mismo nombre. La primera implementa una memoria estática, la configuración se muestra en la figura 3.1.

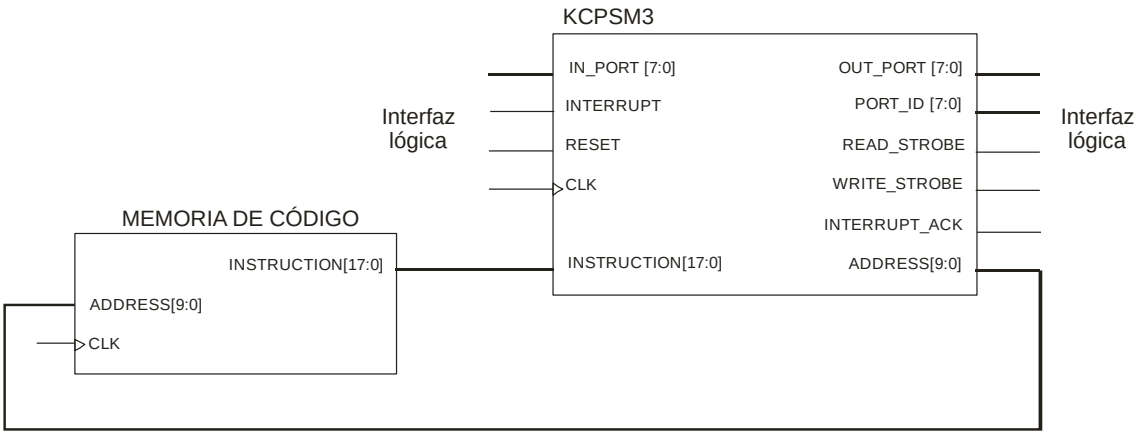


Figura 3. 1 Integración del procesador con memoria de código estática.

Para esta implementación se necesita ejecutar el ensamblador KCPSM3 para generar el archivo con extensión .vhd e importarlo al directorio de trabajo. Sintetizarlo e implementarlo con las herramientas adecuadas. Cualquier cambio en el código del programa que ejecutará el Picoblaze, requerirá de la repetición del proceso.

La segunda plantilla genera una memoria dinámica, la cual incluye un conjunto de señales internas así como una señal externa de inicialización (reset). También se genera un archivo con extensión .vhd, el cual se debe conectar como se muestra en la figura 3.2 para que desde el bloque de memoria se pueda reiniciar al PicoBlaze. Las señales internas hacen posible, a través de un programa denominado JTAG_LOADER, la descarga de otro código a la memoria. Con este procedimiento no es necesario sintetizar e implementar al sistema cada vez que se quiera remplazar el código a ejecutar, ya que no se hace una modificación del hardware, sólo un refresco de la memoria de código. La síntesis e implementación se requieren únicamente con modificaciones del hardware, reduciendo el tiempo de desarrollo.

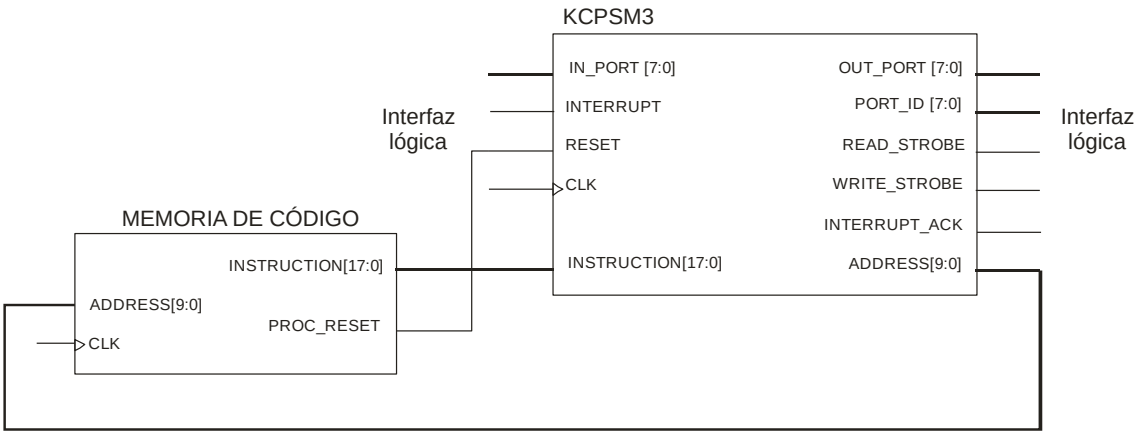


Figura 3. 2 Integración del procesador con memoria de código recargable.

3.1.3 Declaración e Instanciación del KCPSM3

La macro KCPSM3 es provista como un recurso de VHDL (kcpsm3.vhd). El código está disponible con una arquitectura descrita en bajo nivel (*low_level_definition*), adecuada para implementación y simulación y no debe ser modificado de ninguna manera. La implementación del núcleo se muestra en la figura 3.3

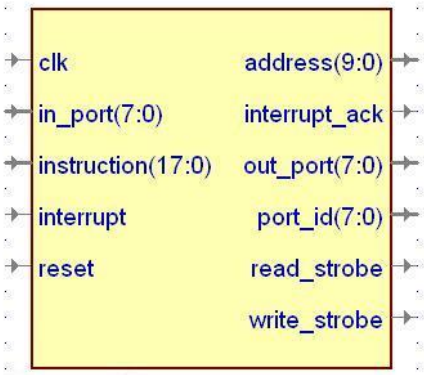


Figura 3. 3 Diagrama del núcleo KCPSM3 implementado en VHDL.

3.1.4 Declaración e Instanciación de la Memoria ROM

El ensamblador del KCPSM3 genera un archivo VHDL para implementarse en un bloque de RAM y define su contenido inicial. El nombre del componente ROM depende del nombre del archivo del programa en ensamblador. Por ejemplo, si el archivo se llama "buffer1.psm" entonces el ensamblador generará un archivo llamado "buffer1.vhd". La Figura 3.4 muestra la implementación de la memoria ROM.

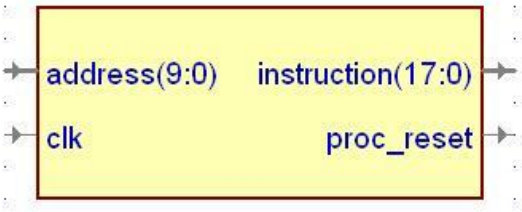


Figura 3. 4 Diagrama de la memoria ROM implementado en VHDL.

3.2 Implementación del Módulo de Puertos

La función que debe cumplir el Módulo de Puertos es la de proporcionar los medios necesarios para introducir datos al microprocesador, así como enviar datos del microprocesador a dispositivos externos.

El diseño del Hardware se muestra en la figura 3.5. Se definen 3 señales para que sean utilizadas como registros de 8 bits, éstas se muestran en la tabla 3.1. *REG_CONF1* es la señal reservada para el registro de configuración, *REG_OUT1* es el registro en el que se

depositan los valores que se enviarán al exterior, *REG_IN1* es el registro que retendrá los valores leídos en las terminales del puerto de entrada.

```
1  signal REG_CONF1:  STD_LOGIC_VECTOR (7 downto 0):="00000000";
2  signal REG_OUT1:   STD_LOGIC_VECTOR (7 downto 0):="00000000";
3  signal REG_IN1:   STD_LOGIC_VECTOR (7 downto 0):="00000000";
```

Tabla 3. 1 Definición de las señales utilizadas en el módulo de puertos.

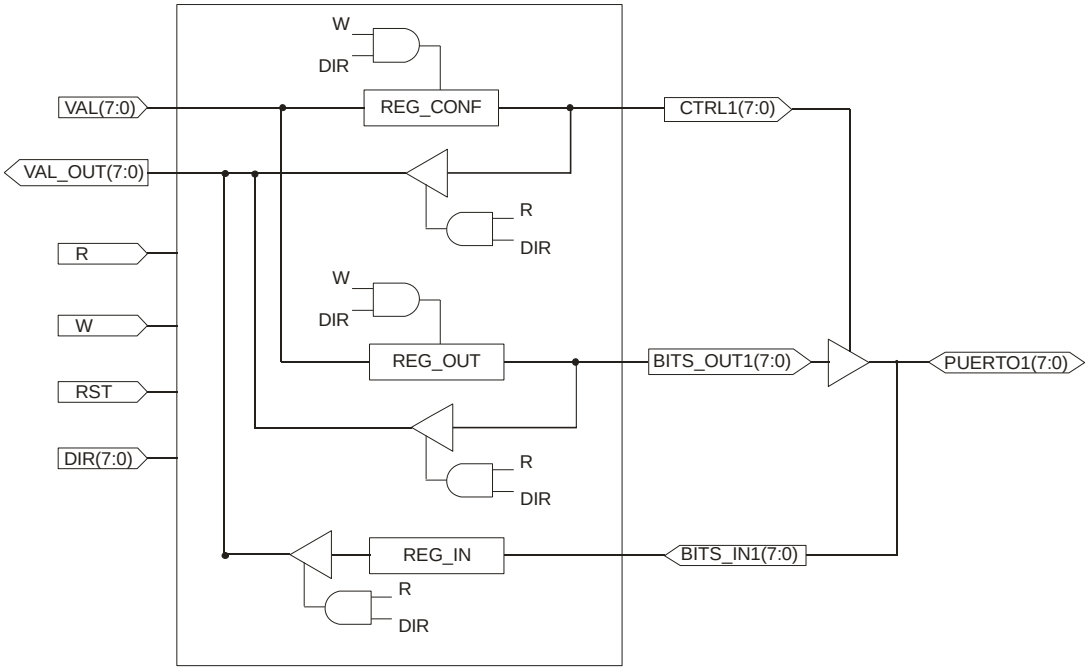


Figura 3. 5 Diagrama de bloques del módulo de puertos.

Existen tres procesos de escritura, los cuales verifican si está habilitado el bit de escritura y si la dirección pertenece a alguno de los registros. En caso de cumplirse estas condiciones, el valor presente en el bus *VAL* es asignado al registro correspondiente. En el caso del proceso *write3*, solo se monitorea si existe algún valor presente en el bus *BITS_IN1* para ser asignado al registro *REG_IN1*. El código se muestra en la tabla 3.2.

```
1  write1: process(clk)
2  begin
3     if  clk'event  and clk = '1' then
4        if  RST='1' then
5           REG_CONF1<="00000000";
6        elsif  DIR="11110000" and W='1' then
7           REG_CONF1<=INPUT;
8        end if;
9     end if;
10 end process write1;
```

Tabla 3. 2 Procesos de escritura utilizados en el módulo de puertos.

```
11 write2: process(clk)
12 begin
13    if  clk'event  and clk = '1' then
14       if  RST='1' then
15          REG_OUT1<="00000000";
16       elsif  DIR="11110001" and W='1' then
17          REG_OUT1<=INPUT; end if; end if;
18 end process write2;
19
20 write3: process(clk)
21 begin
22    if  clk'event  and clk = '1' then
23       if  RST='1' then
24          REG_IN1<="00000000";
25       else
26          REG_IN1<=bits_in1;
27       end if;
29    end if;
30 end process write3;
```

Tabla 3.2 Procesos de escritura utilizados en el módulo de puertos (continuación).

Un proceso se dedica a la lectura, en donde se implementa un multiplexor, el código se muestra en la tabla 3.3. Si existe una lectura y la dirección corresponde a un registro, éste se asigna a la salida *VAL_OUT*. En cualquier otro caso se mostrará alta impedancia para evitar conflictos de lectura.

```
1  read_process: process(R, DIR,REG_CONFIG1, REG_OUT1,REG_IN1)
2  begin
3     if  R='1' then
4        case DIR is
5           when "11110000" => VAL_OUT<=REG_CONF1;
6           when "11110001" => VAL_OUT<=REG_OUT1;
7           when "11110010" => VAL_OUT<=REG_IN1;
8           when others =>    VAL_OUT <= "ZZZZZZZZ";
9        end case;
10    else
11       VAL_OUT <= "ZZZZZZZZ";
12    end if;
13 end process read_process;
```

Tabla 3. 3 Proceso de lectura utilizado en el módulo de puertos.

El valor que se deposita en el registro de configuración se envía al bus *CTRL1* para que habilite como lectura o escritura las terminales del puerto. El contenido del registro *REG_OUT1* se envía al bus *BITS_OUT1*. Observe la tabla 3.4.

```
1  CTRL1<=REG_CONF1;
2  BITS_OUT1<=REG_OUT1;
```

Tabla 3. 4 Asignación de señales del módulo de puertos.

Externo a cada bloque de puertos, se generan buses de 8 líneas que se dirigen a un buffer de tercer estado, el bus *CTRL1* envía las señales que harán que el buffer se comporte como alta Impedancia permitiendo la lectura de datos o como corto circuito permitiendo la escritura de datos. El código se muestra en la tabla 3.5.

```
1  bus_construction_out:
2  for i in 0 to 7 GENERATE
3  begin
4      PUERTO1(i) <=BITS_OUT1(i) when CTRL1(i)='1' else 'Z';
5      BITS_IN1(i)<=PUERTO1(i);
6  end GENERATE;
7
8  PUERTO1<=PUERTO1;
```

Tabla 3. 5 Construcción de bus externo al módulo de puertos.

El mismo código se puede reutilizar *n* veces modificando la dirección de los registros, el nombre de *BITS_IN*, *BITS_OUT* y *CTRL* para implementar otros bloques de puertos. En la presente tesis se implementaron 4 puertos de propósito general.

El puerto 3 es multiplexado con una función alterna que se detalla en la sección 3.3.3, en la figura 3.6 se muestra la adaptación en las conexiones para la doble funcionalidad.

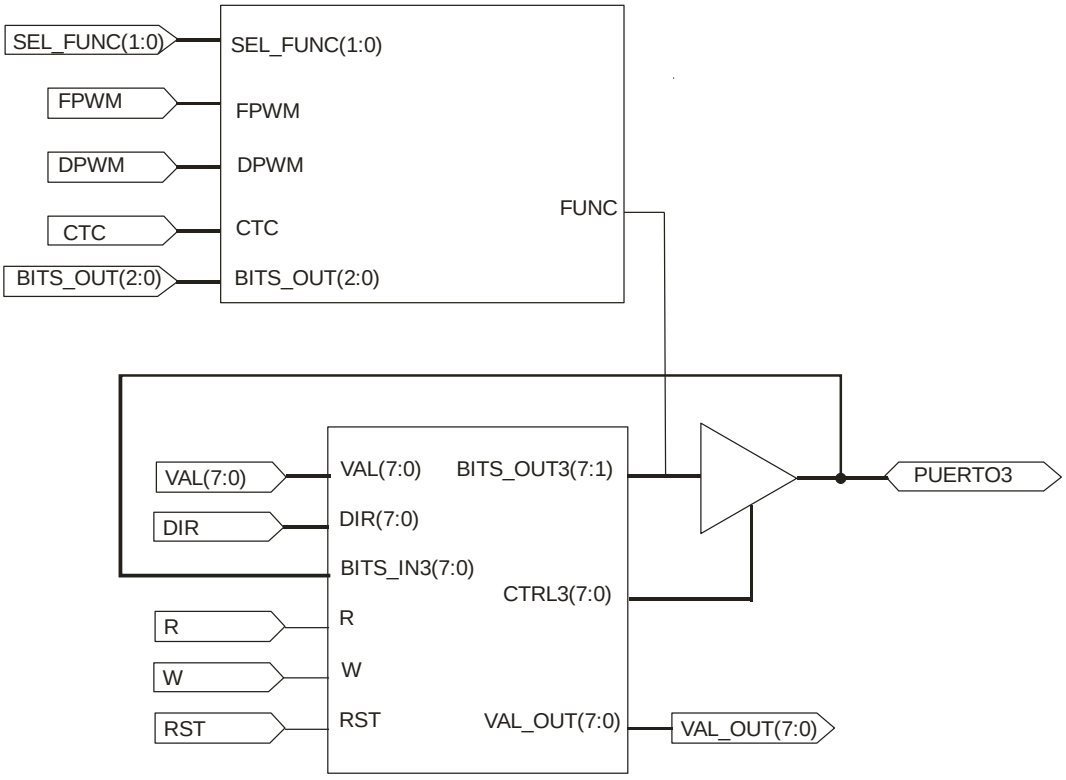


Figura 3. 6 Diagrama de bloques para adaptar el puerto 3 a funciones especiales.

3.3 Implementación del Módulo de Temporización

Este módulo proporciona funciones que implican controlar tiempos o contar eventos. Genera las señales para la interrupción por sobreflujo y por comparación, proporciona divisores de frecuencia predeterminados, contador de eventos, limpieza del temporizador por comparación, funciones PWM de rampa sencilla y doble.

3.3.1 Preescalador del Contador

El Preescalador utiliza la señal del reloj principal aunque puede utilizar una señal externa, necesita buses para enviar y recibir los datos que se escribirán en el registro de configuración llamado *VAL_REG* y para enviar la señal de reloj dividida por el preescalador. El diagrama de bloques del preescalador del contador se muestra en la figura 3.7.

La tabla 3.6 contiene la declaración de una señal de 25 bits que se utilizará como un divisor de frecuencia y una señal que servirá como registro del preescalador, dependiendo del contenido de éste se asignará la frecuencia de la señal de salida.

```
1  signal CONT: std_logic_vector(24 downto 0);
2  signal VAL_REG: STD_LOGIC_VECTOR (7 downto 0);
```

Tabla 3. 6 Definición de las señales utilizadas en el bloque del preescalador.

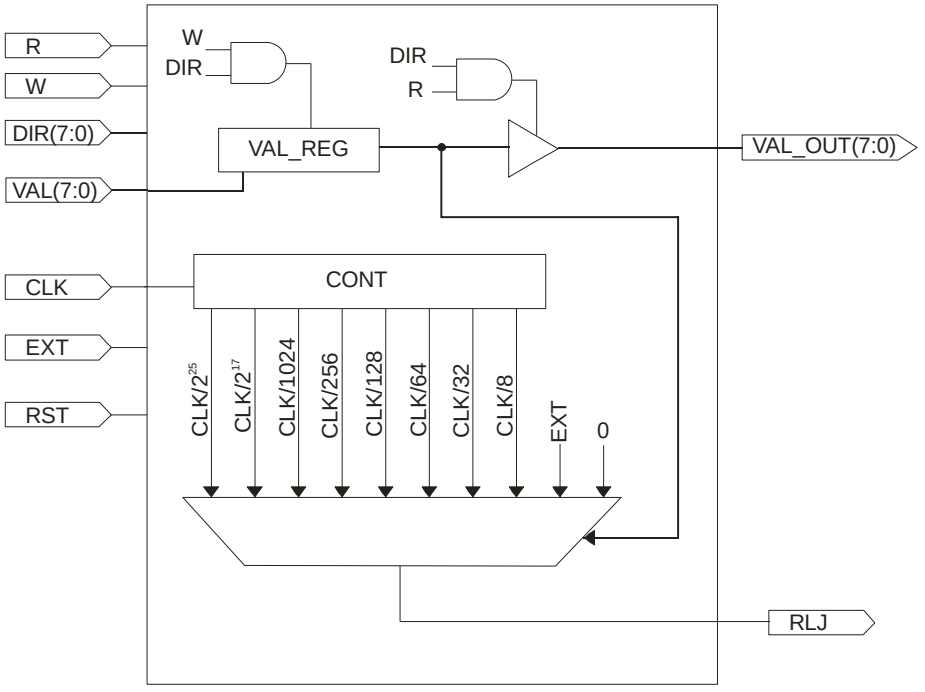


Figura 3. 7 Diagrama de bloques del preescalador.

El bloque del preescalador consta principalmente de 4 procesos, 2 de ellos son dedicados a leer y escribir en el registro del preescalador. Un proceso implementa un contador ascendente, donde a cada ciclo de reloj se incrementa en uno el valor del registro *CONT*.

Un multiplexor selecciona un bit del divisor de frecuencia de acuerdo al registro *VAL_REG* para enviar la señal a través de *RLJ*, esto se muestra en la tabla 3.7.

```
1   read: process(DIR, R)
2   begin
3      if  DIR="11111100" and R='1' then
4         VAL_OUT <= val_reg;
5      else
6         VAL_OUT <= "ZZZZZZZZ";
7       end if;
8   end process read;
9
10  write: process(clk)
11  begin
12  if  clk'event  and clk = '1' then
13         if  RST='1' then
14            VAL_REG<="00000000";
15         elsif  DIR="11111100" and W='1' then
16            VAL_REG<=VAL;
17         end if;
18     end if;
19  end process write;
20
21  contador: process(clk)
22  begin
23     if  clk' event and clk = '1' then
24         if  RST='1' then
25            CONT<="00000000000000000000000000";
26         else
27            CONT <= CONT + 1;
25         end if;
26     end if;
27  end process contador;
28
29  multiplexor: process(VAL_REG, CLK, EXT, CONT)
30  begin
31     case VAL_REG ( 3 downto 0) is
32         when "0000" => RLJ <= '0';
33         when "0001" => RLJ <= EXT;
34         when "0010" => RLJ <= CLK;
35         when "0011" => RLJ <= cont(2);-- clk/8
36          when "0100" => RLJ <= cont(4);-- clk/32
37         when "0101" => RLJ <= cont(5);-- clk/64
38         when "0110" => RLJ <= cont(6);-- clk/128
39         when "0111" => RLJ <= cont(7);-- clk/256
40         when "1000" => RLJ <= cont(9);-- clk/1024
41         when "1001" => RLJ <= cont(16);-- clk/2^17
42         when "1010" => RLJ <= cont(24); -- clk/2^25
43         when others => RLJ <= '0';
44     end case;
45  end process multiplexor;
```

Tabla 3. 7 Código de los procesos utilizados en el bloque del preescalador.

3.3.2 Contador

Este bloque es el único que se sincroniza con una señal diferente a la del reloj principal, ya que utiliza la señal que envía el módulo del Preescalador. Tiene terminales para inicializar

el contador, para leer o escribir en sus registros y conocer su estado. En la figura 3.8 se muestra su implementación.

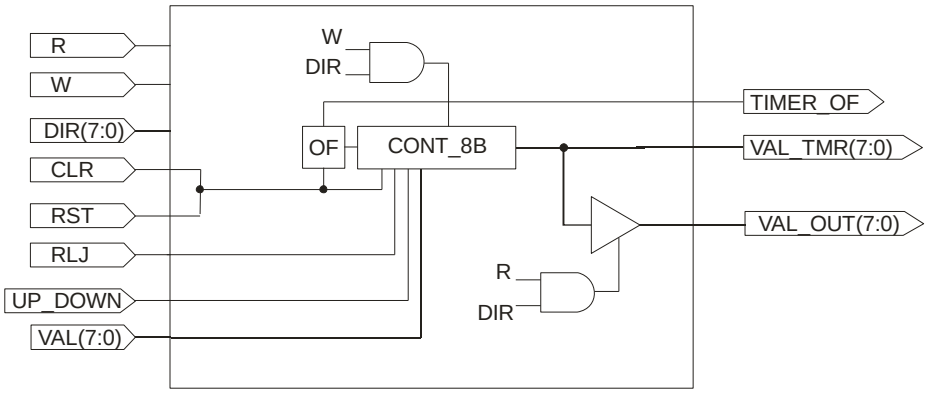


Figura 3. 8 Diagrama de bloques del contador.

Se define un registro de 9 bits para implementar un contador de 8 bits, el noveno bit servirá como bit de sobreflujo. El proceso de lectura solo mostrará los 8 bits menos significativos del registro.

El proceso principal *contador* inicializará en cero el registro *CONT_8B* cuando la terminal *CLR* o *RST* esté en '1'. En caso de que se indique la dirección del registro del contador y la terminal *w* esté en '1', se le asignará el valor presente en el puerto *VAL*.

Si no hay valor que escribir, el registro se incrementa o decrementa en cada ciclo de reloj o en cada flanco de subida de la señal *RLJ* que proviene del módulo del preescalador, de acuerdo el valor presente en la terminal *UP_DOWN*. La limpieza del noveno bit se realiza dentro de este mismo proceso al ocurrir un desborde. La señal de sobreflujo proviene del noveno bit del contador y se asigna a la terminal *TIMER_OF*, mientras que los bits restantes se envían a través del bus *VAL_TMR* para ser utilizadas por el módulo *PWM*. En la tabla 3.8 se muestra el código correspondiente.

```
1  flanco_subida: process(clk)
2  begin
3      if clk'event and clk='1' then
4        delay_RLJ <= RLJ;
5        if RLJ='1' and delay_RLJ='0' then
6            flanco <= '1';
7         else
8            flanco <= '0';
9        end if;
10 end if; end process flanco_subida;
11
12 contador: process(clk)
13 begin
14    if clk' event and clk= '1' then
15           if rst='1' or clr='1' then
16              cont_8b<="000000000";
17           else
```

Tabla 3. 8 Código de los procesos utilizados en el bloque del contador.

```
18              if DIR = "11111111" and W='1' then
19                 cont_8b(7 downto 0)<=VAL;
20              else
21                 if U_D='1' then        --DPWM; asc/desc
22                    if UP_DOWN='1' then
23                       if cont_8b(8)='1' then
24                          cont_8b<="011111111";
25                       elsif REG_OUT = "0010" or flanco='1' then
26                          cont_8b<= cont_8b + 1;
27                       end if;
28                    else
29                       if cont_8b(8)='1' then
30                          cont_8b<="000000001";
31                       elsif REG_OUT = "0010" or flanco='1' then
32                          cont_8b<= cont_8b - 1;
33                       end if;
34                    end if;
35                 else                   --asc;fpwm,ctc,cont
36                    if cont_8b(8)='1' then
37                       cont_8b<="000000001";
38                    elsif REG_OUT = "0010" or flanco='1' then
39                       cont_8b<= cont_8b + 1;
40                    end if;
41                 end if;
42              end if;
43           end if;
44        end if;
45 end process contador;
46
47 Read: process(DIR,R)
48 begin
49    if DIR = "11111111" and R='1' then
50       COUT <= cont_8b(7 downto 0);
51    else
52       COUT <="ZZZZZZZZ";
53    end if;
54 end process Read;
55
56 val_tmr   <= cont_8b(7 downto 0);
57 int_timer    <= cont_8b(8);
58 timer_of  <= cont_8b(8);
```

Tabla 3.8 Código de los procesos utilizados en el bloque del contador (continuación).

3.3.3 PWM

El bloque *PWM* necesita de la señal de sobreflujo y recibir continuamente los valores del contador de 8 bits. Requiere terminales para leer y escribir sus registros. Envía la señal de que ha ocurrido una igualdad al comparar un valor específico con el contador. Trabaja en conjunto con el bloque del Puerto3 utilizando señales de selección para multiplexar las funciones que genera. El diagrama del bloque *PWM* se muestra en la figura 3.9.

En la tabla 3.9 se declaran 8 señales, *VAL_COMP* y *CONF_COMP* funcionarán como registros de 8 bits mientras que *VAL_REG* se ocupará como registro temporal.

```
1  signal  MATCH: STD_LOGIC;
2  signal  CTC: STD_LOGIC:='0';
3  signal  CLR: STD_LOGIC:='0';
4  signal  FPWM: STD_LOGIC:='0';
5  signal  DPWM: STD_LOGIC:='0';
6  signal  VAL_REG : std_logic_vector(7 downto 0):="00000000";
7  signal  VAL_COMP : std_logic_vector(7 downto 0):="00000000";
8  signal  CONF_COMP : std_logic_vector(7 downto 0):="00000000";
```

Tabla 3. 9 Definición de señales utilizadas en el bloque PWM.

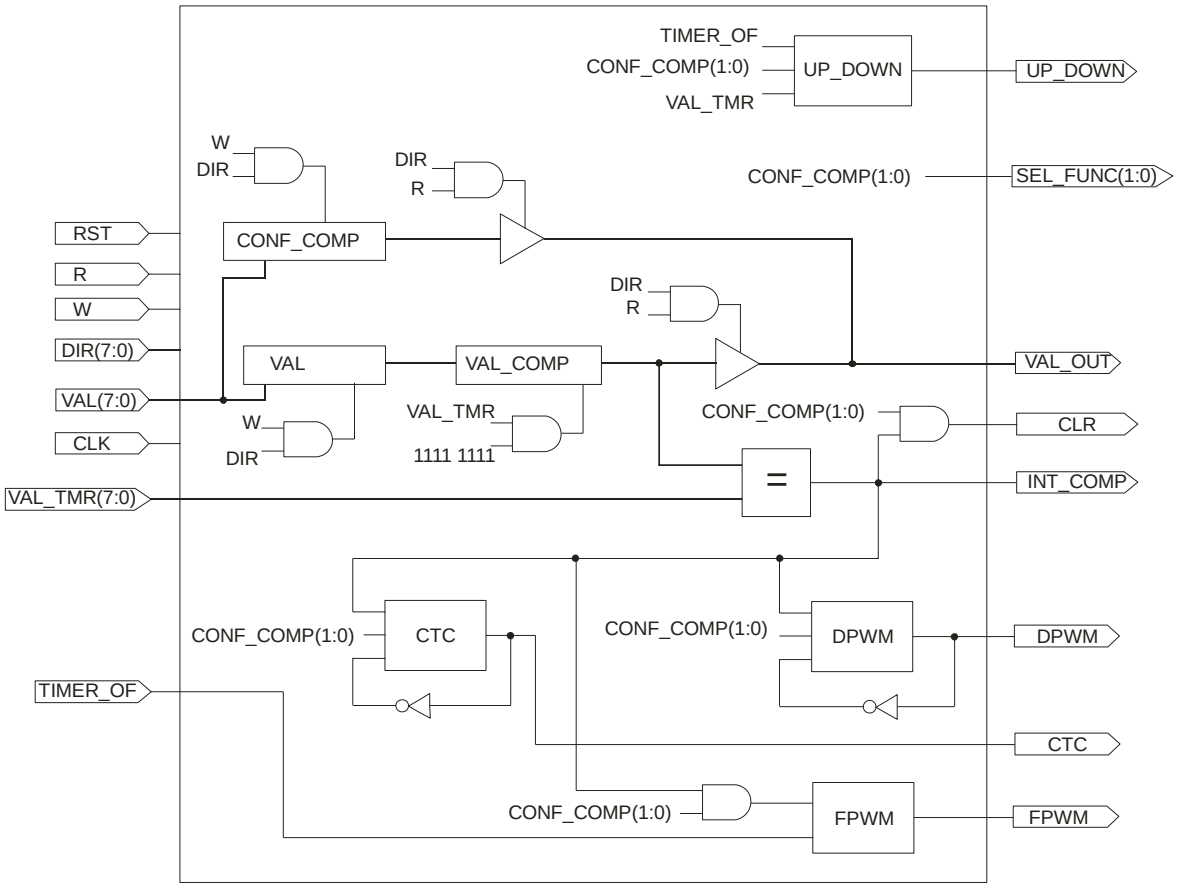


Figura 3. 9 Diagrama de bloques del PWM.

El registro *CONF_COMP* tiene un proceso de lectura y otro de escritura. El registro *VAL_COMP* puede leerse, más no escribirse inmediatamente, su valor se deposita en el registro *VAL_REG* y es hasta que el contador alcanza su máximo valor que se asigna al registro *VAL_COMP*. El código se muestra en la tabla 3.10.

```
1  write1: process(CLK)
2  begin
3     if clk' event and clk= '1' then
4        if  RST='1' then
5           VAL_S<="00000000";
6        elsif DIR = "11111001" and W='1' then
7              VAL_S<=VAL;
8        end if;
9     end if;
10 end process write1;
11
12 write2: process(CLK)
13 begin
14    if clk' event and clk= '1' then
15       if  RST='1' then
16          CONF_COMP_S<="00000000";
17       elsif DIR = "11111000" and W='1' then
18             CONF_COMP_S<=VAL;
19       end if;
20    end if;
21 end process write2;
22
23 actualizacion: process(CLK)
24 begin
25    if clk' event and clk= '1' then
26       if  RST='1' then
27          VAL_COMP_S<="00000000";
28       elsif VAL_TMR= "11111111" then
29          VAL_COMP_S<=VAL_S;
30       end if;
31    end if;
32 end process actualizacion;
33
34 read1: process(DIR,R)
35 begin
36    if DIR = "11111001" and R='1' then
37       VAL_OUT<=VAL_COMP;
38    else
39       VAL_OUT<="ZZZZZZZZ";
40    end if;
41 end process read1;
42
43 read2: process(DIR,R)
44 begin
45    if DIR = "11111000" and R='1' then
46       VAL_OUT<=CONF_COMP;
47    else
48       VAL_OUT<="ZZZZZZZZ";
49    end if;
50 end process read2;
```

Tabla 3. 10 Código de los procesos de lectura y escritura utilizados en el bloque PWM.

Todas las funciones tienen en común la comparación del registro *VAL_COMP* con el valor del contador. En la tabla 3.11 se muestra el código del proceso de comparación, donde al existir una igualdad se activa en alto la señal *MATCH*, en caso contrario la señal permanece en bajo.

```
1  comparacion: process(VAL_COMP,VAL_TMR)
2  begin
3      if VAL_COMP = VAL_TMR then
4          MATCH<='1';
5      else
6          MATCH<='0';
7      end if;
8  end process comparacion;
```

Tabla 3. 11 Código del proceso de comparación utilizado en el bloque PWM.

El proceso *CLR* modifica la resolución del contador cuando hay una igualdad en el proceso de comparación y cuando el bit 1 del registro *CONF_COMP* está en alto, es decir, cuando está habilitada la función de limpieza del contador por comparación. Bajo las mismas condiciones otro proceso conmuta la señal *CTC*. El diagrama de tiempo se muestra en la figura 3.10 y el código en la tabla 3.12.

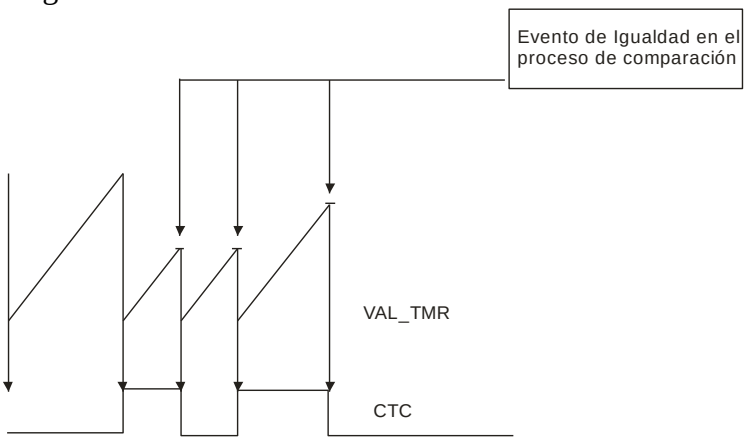


Figura 3. 10 Diagrama de tiempo de la función CTC.

```
1  CLR_process: process (MATCH,CONF_COMP)
2  begin
3         if MATCH = '1' and CONF_COMP= "00000001" then
4            CLR_S<='1';
5         else
6            CLR_s<='0';
7         end if;
8  end process CLR_process;
9
10 ctc_process: process(MATCH,CONF_COMP)
11 begin
12        if MATCH = '1' and CONF_COMP="00000001"  then
13           CTC_S <= not(CTC_S);
14       end if;
15 end process ctc_process;
```

Tabla 3. 12 Código de los procesos empleados para generar la función CTC.

Para generar la función de modulación por ancho de pulso se genera un ‘0’ lógico cuando existe una igualdad en el proceso de comparación y el bit 2 del registro *CONF_COMP* esté habilitado, este valor permanece hasta que se presenta un evento de sobreflujo y conmuta a ‘1’ lógico.

El diagrama de tiempo se observa en la figura 3.11 y el código en la tabla 3.13.

La generación de PWM de doble rampa es implementada con 2 procesos que verifican si está habilitado el bit 3 del registro *CONF_COMP*.

Un proceso controla al contador, al iniciar su operación, éste va en sentido ascendente; cuando alcanza el tope, se envía un '0' lógico a través de *up_down* que configura al contador en forma descendente. Cuando alcanza el valor de cero, envía un '1' lógico para configurarlo en forma ascendente.

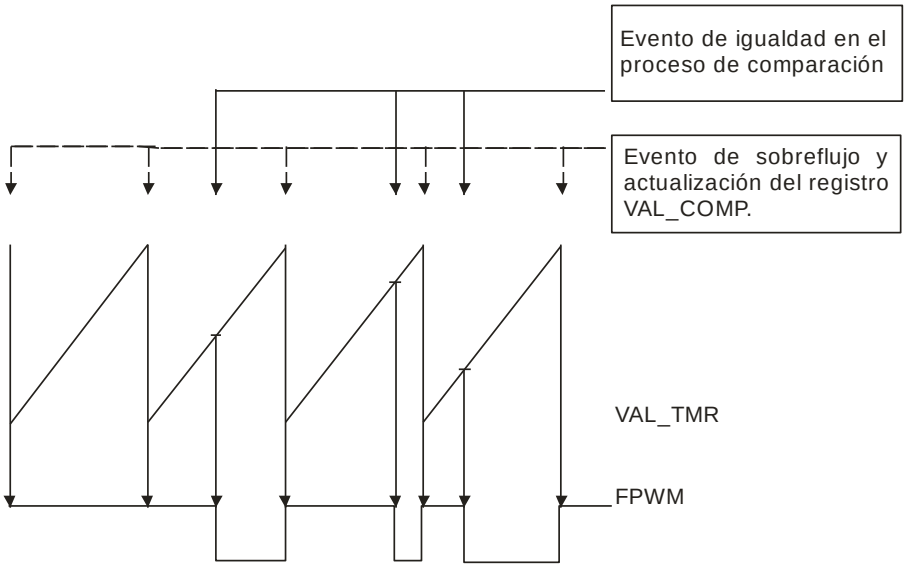


Figura 3. 11 Diagrama de tiempo de la función PWM.

```
1  fast_pwm: process (MATCH,CONF_COMP,TIMER_OF)
2  begin
3     if match = '1' and CONF_COMP="00000010"  then
4        fpwm<='0';
5     elsif  TIMER_OF='1' then
6        fpwm <= '1';
7     else
8        fpwm<=fpwm;
9     end if;
10 end process fast_pwm;
```

Tabla 3. 13 Código del proceso empleado para generar la función PWM.

El segundo proceso conmuta la señal *dpwm* cuando existe una igualdad en el proceso de comparación. El diagrama de tiempo se muestra en la figura 3.12 y el código en la tabla 3.14.

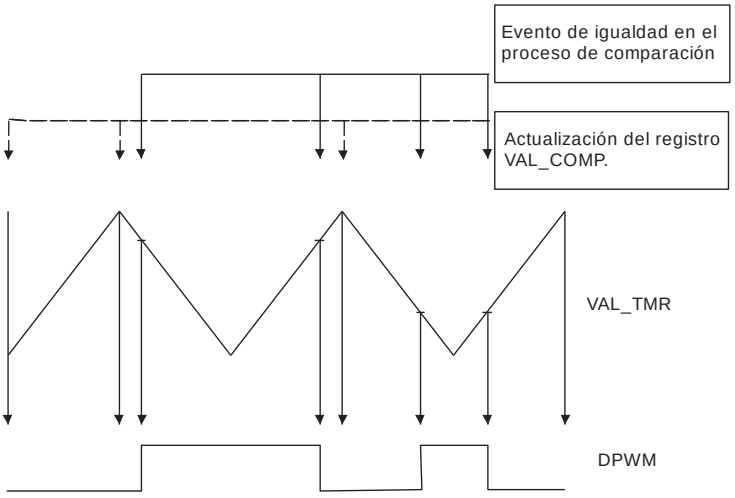


Figura 3. 12 Diagrama de tiempo de la función PWM de rampa doble.

```
1   up_down_process: process(CONF_COMP, VAL_TMR, TIMER_OF)
2   begin
3       if conf_comp(3)='1' then
4           if VAL_TMR="11111111" then
5               UP_DOWN <= '0' ;
6           end if;
7           if VAL_TMR="00000000" then
8               UP_DOWN <= '1';
9           end if;
10      else
11          UP_DOWN<='1';
12      end if;
13  end process up_down_process;
14
15  double_pwm: process (MATCH,CONF_COMP)
16  begin
17        if match = '1' and CONF_COMP="00000011"  then
18              dpwm <= not(dpwm);
19        end if;
20  end process double_pwm;
```

Tabla 3. 14 Código del proceso empleado para generar la función PWM de rampa doble.

Para la asignación de las señales a los puertos correspondientes, el registro de configuración contiene los bits que habilitan o deshabilitan las funciones especiales, estas señales las recibe el módulo de puertos. El código se muestra en la tabla 3.15.

```
1   temp: process(clk)
2      begin
3          if  clk'event  and clk = '1' then
4              SEL_FUNC(0)<=CONF_COMP_S(0);
5              SEL_FUNC(1)<=CONF_COMP_S(1);
6          end if;
7   end process temp;
8
9   UP_DOWN<=UP_DOWN;CTC<=CTC_S; CLR<=CLR_S; FPWM<=FPWM_S;
10  DPWM<=DPWM_S; INT_COMP<=MATCH;
```

Tabla 3. 15 Asignación de señales del bloque PWM.

3.3.4 Integración del Módulo de Temporización

En la figura 3.13 se muestra la relación que existe entre los diferentes bloques que integran al módulo de temporización.

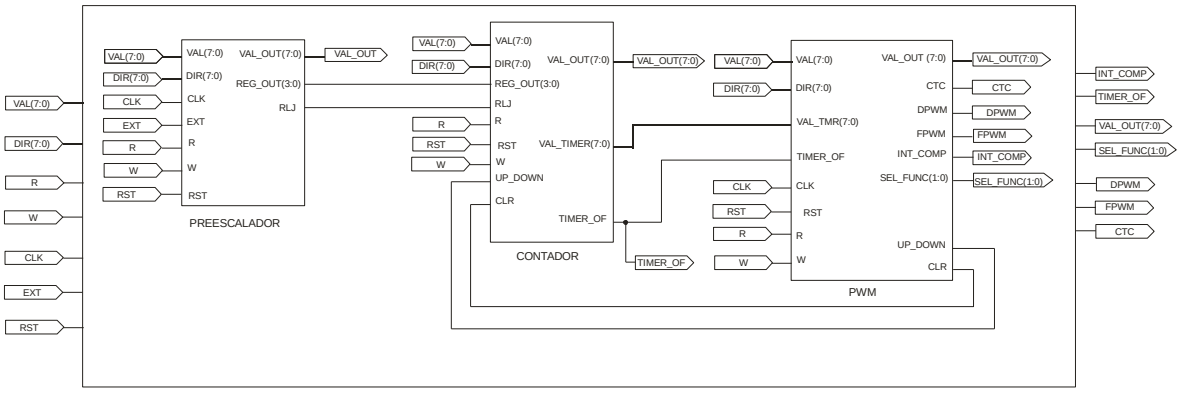


Figura 3. 13 Diagrama de bloques del módulo temporizador.

3.4 Implementación del Módulo de Interrupciones

El módulo de interrupciones determina si un evento externo, de sobreflujo o de igualdad en una comparación, es válido, para disparar una interrupción evaluando los registros de estado y de configuración. Contiene un bloque auxiliar para determinar si la interrupción externa se habilitará por flanco o por nivel.

3.4.1 Bloque de Interrupciones

El bloque de interrupciones recibe las señales que indican que han ocurrido eventos que disparan una interrupción, pero es necesaria la evaluación de los registros de estado y de configuración antes de enviar una señal que active la interrupción del procesador. El diagrama del bloque de interrupciones se muestra en la figura 3.14.

Se definen dos señales que se utilizarán como registros de 8 bits, estos registros son la parte principal del módulo de interrupciones. El código se muestra en la tabla 3.16

```
1  signal  int_conf : std_logic_vector (7 downto 0):="00000000";
2  signal  int_state : std_logic_vector (7 downto 0):="00000000";
```

Tabla 3. 16 Definición de señales del bloque de interrupciones.

Para que una interrupción sea válida deben estar habilitados su permiso individual y el habilitador global, además de haberse activado su bit de estado. Un circuito lógico combinacional toma la información de los registros *int_conf* y de *int_state* para verificar las condiciones antes mencionadas. Una máquina de estados enviará una señal para indicar que ha ocurrido una interrupción, la señal se suspende cuando el procesador la reconoce. Esto se realiza con el código que se muestra en la tabla 3.17.

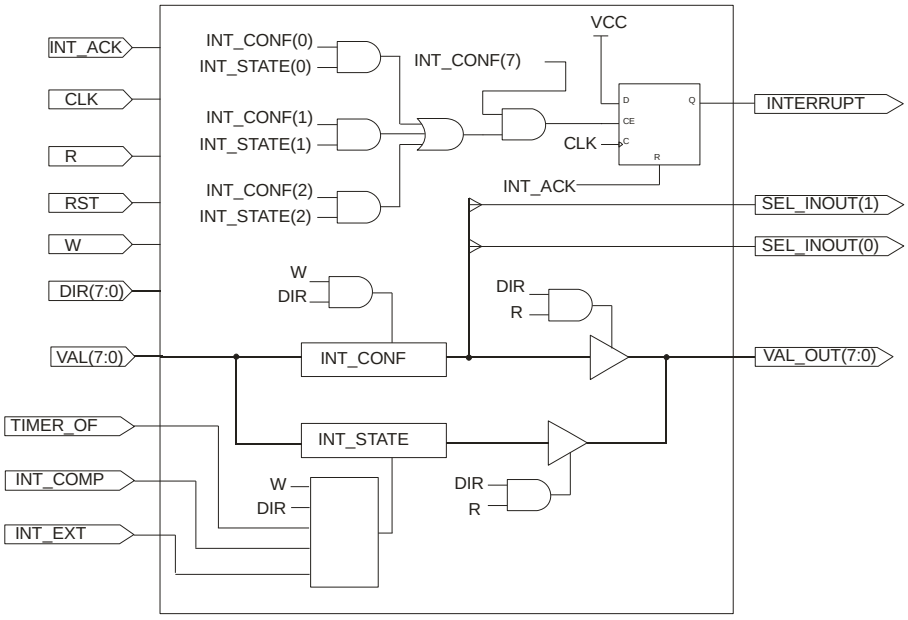


Figura 3. 14 Diagrama del bloque de interrupciones.

```
1  Sreg0_machine: process (CLK)
2  begin
3  if CLK'event and CLK = '1' then
4     if Rst='1' then
5        Sreg0 <= S1;
6        INTERRUPT <= '0';
7     else
8        case Sreg0 is
9           when S1 =>
10             INTERRUPT <= '0';
11             if combinacional='1' then
12                Sreg0 <= S2;
13             elsif combinacional='0' then
14                Sreg0 <= S1;
15             end if;
16          when S2 =>
17             INTERRUPT <= '1';
18             if int_ack='1' then
19                Sreg0 <= S3;
20                INTERRUPT<= '0';
21             elsif int_ack='0' then
22                Sreg0 <= S2;
23             end if;
24          when S3 =>
25             if combinacional='0' then
26                Sreg0 <= S1;
27             elsif combinacional='1' then
28                Sreg0 <= S3;
29 end if;when others =>null;end case;end if;end if;end process;
30
31 combinacional <='1' when int_conf(7)='1' and ((int_conf(0)='1' and
32 int_state(0)='1')or  (int_conf(1)='1'  and  int_state(1)='1')  or
33 (int_conf(2)='1' and int_state(2)='1')) else '0';
```

Tabla 3. 17 Código del proceso int_timer.

Existen procesos de lectura y escritura para cada registro. El registro de estado se puede escribir por Hardware o por software. Observe el código en la tabla 3.18.

El registro de configuración también determina si la interrupción externa será activada por flanco de subida, flanco de bajada, nivel lógico alto o bajo, por ello se envía el valor de los bits 4 y 5 al puerto de salida *SEL_EXT*. La señal *aux* utilizada en el proceso *int_timer* se asigna al puerto *INT_TIMER* para comunicar al procesador de una interrupción válida. El código se muestra en la tabla 3.19.

```
1  write_reg_conf: process(clk)
2  begin
3        if  rst='1' then
4           INT_CONF<="00000000";
5        elsif DIR = "11111011" and W='1' then   -- DIR=FB  and W=1
6              int_conf<=VAL;
7        end if;
8  end process write_reg_conf;
9
10 write_reg_state: process(clk)
11 begin
12    if clk'event and clk='1' then
13          if timer_of='1' then
14             int_state(0)<='1';
15          end if;
16
17          if int_ext='1' then
18             int_state(1)<='1';
19          end if;
20
21          if int_comp='1' then
22             int_state(2)<='1';
23          end if;
24
25       if  rst='1' then
26          INT_STATE<="00000000";
27       elsif DIR = "11111010" and W='1' then   -- DIR=FA  and W=1
28             int_state<=VAL;
29       end if;
30    end if;
31 end process write_reg_state;
32
33 read_reg_conf: process(DIR,R)
34 begin
35    if DIR = "11111011" and R='1' then   -- DIR=FB  and R=1
36       VAL_OUT<= int_conf;
37    else
38       VAL_OUT <="ZZZZZZZZ";
39    end if;
40 end process read_reg_conf;
```

Tabla 3. 18 Código de los procesos de lectura y escritura del bloque de interrupciones.

```
41 read_reg_state: process(DIR,R)
42 begin
43    if DIR = "11111010" and R='1' then   -- DIR=FA  and R=1
44       VAL_OUT<= int_state;
45    else
46       VAL_OUT <="ZZZZZZZZ";
47    end if;
48 end process read_reg_state;
```

Tabla 3.18 Código de los procesos de lectura y escritura del bloque de interrupciones (continuación)

```
1 extra: process(clk)
2 begin
3    if clk='1' and clk'event then
4       sel_ext(1)<=int_conf(5);
5       sel_ext(0)<=int_conf(4);
6    end if;
7 end process extra;
```

Tabla 3. 19 Asignación de señales del bloque de interrupciones.

3.4.2 Selección de Interrupción Externa

El bloque de selección de interrupción externa (*SEL_EXT*) recibe y evalúa la señal de la terminal 7 del puerto 3 (reservado para la interrupción externa) para determinar si ha sucedido un flanco de subida, bajada, nivel lógico alto o bajo. El diagrama del bloque *SEL_EXT* se muestra en la figura 3.15.

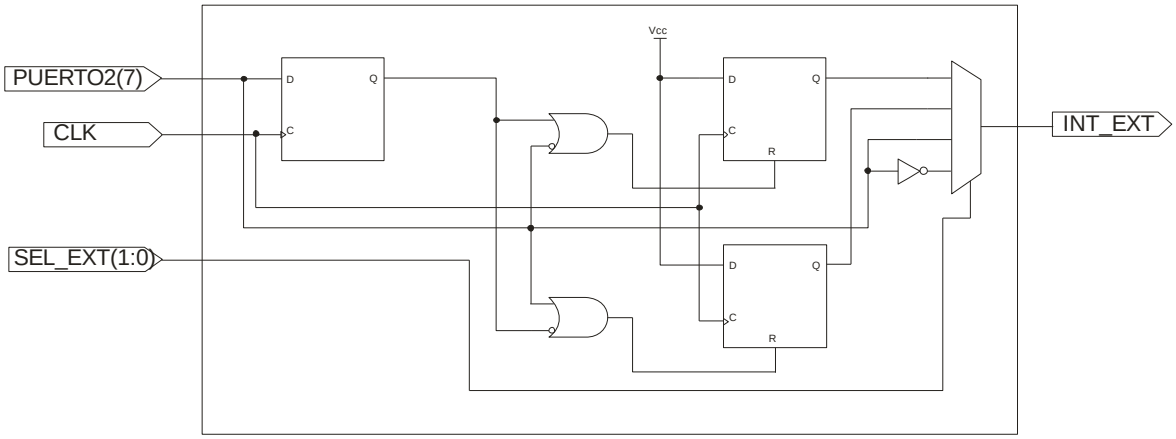


Figura 3. 15 Diagrama del bloque SEL_EXT.

Interiormente el bloque toma la señal externa y la almacena en un flip-flop tipo D, esto generará un retardo entre la señal original y la señal de salida del flip flop, si la señal original se encuentra en nivel lógico alto y la salida del flip flop contiene un nivel lógico bajo, es porque ha ocurrido un flanco de subida, pero si dichos valores se encuentran invertidos es porque ha ocurrido un flanco de bajada. El código de estos procesos se muestra en la tabla 3.20.

```
1  flanco_subida: process(clk)
2    begin
3     if clk'event and clk='1' then
4        delay_senal1_s <= senal;
5       if senal='1' and delay_senal1_s='0' then
6        flanco_subida_s <= '1';
7        else
8        flanco_subida_s <= '0';
9     end if;
10 end if;
11 end process flanco_subida;
12
13 flanco_bajada: process(clk)
14  begin
15     if clk'event and clk='1' then
16       delay_senal2_s <= senal;
17       if senal='0' and delay_senal2_s='1' then
18        flanco_bajada_s <= '1';
19        else
20        flanco_bajada_s <= '0';
21    end if;
22 end if;
23 end process flanco_bajada;
```

Tabla 3. 20 Código de los procesos que detectan flancos de subida o de bajada.

Un multiplexor que es controlado por la terminal *SEL_EXT* que son los bits 4 y 5 del registro *INT_CONF* (mayores detalles en la tabla 2.7) determinará cuál será la señal de salida del multiplexor y por tanto de la terminal *INT_EXT*. Las entradas del multiplexor serán las señales del detector de flanco de subida, de baja, la señal externa y su inverso. El código se muestra en la tabla 3.21.

```
1  detecta:   process(SEL_EXT,   flanco_subida_s,   flanco_bajada_s,
2  PUERTO2(7))
3  begin
4     case SEL_EXT is
5        when "00" => INT_EXT <=flanco_subida_s;
6        when "01" => INT_EXT <=flanco_bajada_s;
7        when "10" => INT_EXT <= PUERTO2(7); --nivel alto;
8        when "11" => INT_EXT <= NOT (PUERTO2(7));  --nivel BAJO;
9        when others =>INT_EXT <='0';
10    end case;
11 end process detecta;
```

Tabla 3. 21 Código del proceso que genera un multiplexor dentro del bloque SEL_EXT.

3.4.3 Integración del Módulo de Interrupciones

El módulo de interrupciones quedará integrado sólo por dos bloques, en la figura 3.16 se muestra la relación que entre ellos se establece.

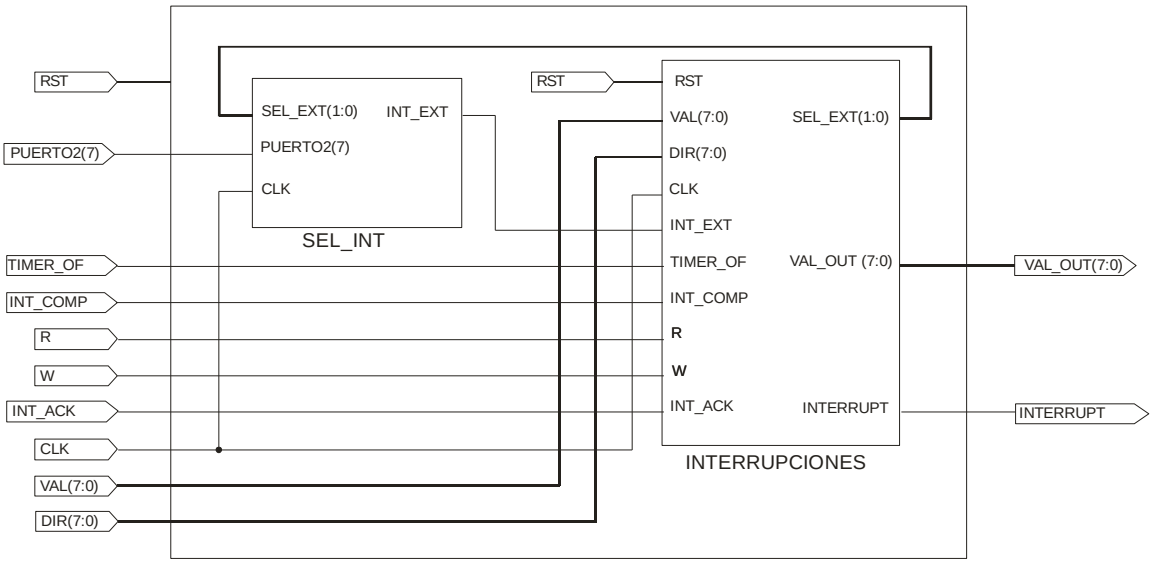


Figura 3. 16 Diagrama de bloques del módulo de interrupciones.

3.5 Integración del Sistema

Los puertos no se integran dentro de un módulo específico, sino que se colocan en el mismo nivel jerárquico en donde todos los módulos trabajan en conjunto con el procesador y la memoria de código, como se observa en la figura 3.18. Esto debido a que se requiere mantener a los buffers de tres estados en los IOBs.

En la figura 3.17 se muestra al sistema completo como un bloque, con cuatro puertos disponibles para diferentes aplicaciones. En el siguiente capítulo se muestra la evaluación del sistema y el análisis de los resultados obtenidos.

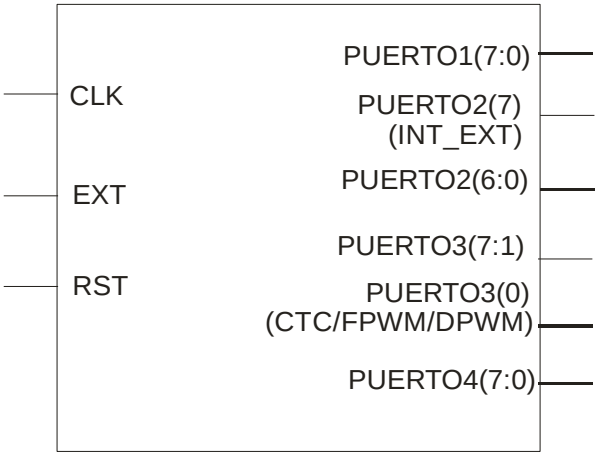


Figura 3. 17 El sistema completo

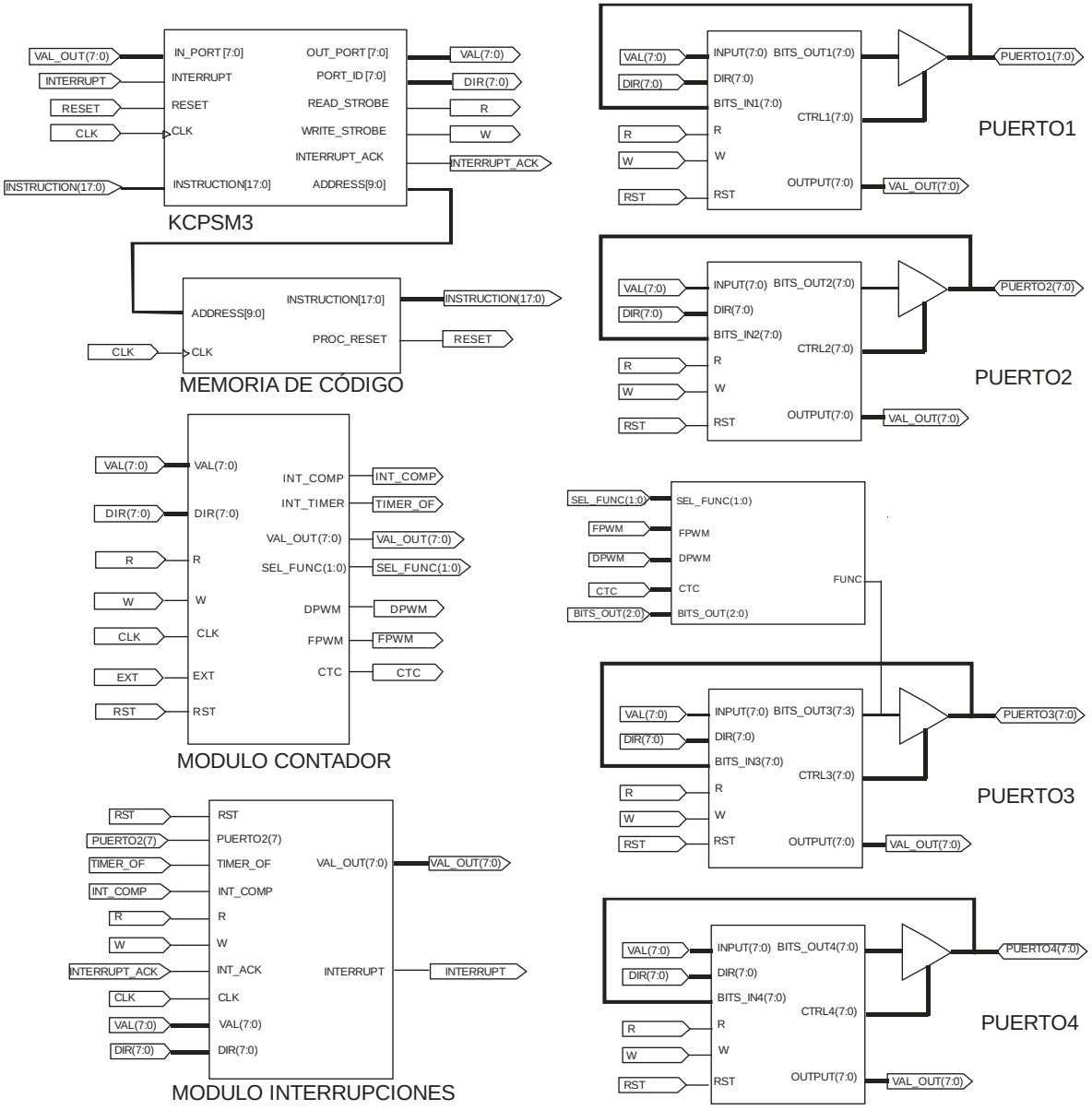


Figura 3. 18 Diagrama de bloques de las conexiones de los módulos temporizador, interrupciones con los puertos, el procesador y la memoria de código.

4. Resultados y Conclusiones

En este capítulo se exponen los resultados de las pruebas realizadas sobre los diferentes módulos desarrollados, para verificar su adecuado funcionamiento. Para ello se implementó el sistema que se muestra en la figura 3.18 en una tarjeta de desarrollo *Spartan 3E Starter Board*, manufacturada por Digilent, cuya frecuencia de su oscilador es de 50MHz, es decir, con un período de 20ns. En el apéndice C se listan las principales características de esta tarjeta de desarrollo.

4.1 Ejemplos de evaluación

A continuación se muestra un código de prueba seguido de una breve explicación, evaluando cada módulo, así como los diagramas de tiempo resultante de las simulaciones.

4.1.1 Prueba del bloque de Puertos

Se propone implementar un buffer de 4 bits utilizando el puerto 1. Tendrá configurado 4 terminales como entrada y 4 como salida, los datos de entrada quedarán en los bits más significativos del registro *REG_IN1*, con rotaciones de bits se generarán las salidas. El valor que se presente en las terminales de entrada se mostrará en las terminales de salida. El código ensamblador se muestra en la tabla 4.1.

```
1  ;BUFFER UTILIZANDO EL PUERTO 1
2
3  CONSTANT    REG_CONF1,  F0
4  CONSTANT    REG_OUT1,   F1
5  CONSTANT    REG_IN1,    F2
6
7  LOAD   S1,0F           ;CARGA EL VALOR 00001111, 7-4 R, 3-0 W
8  OUTPUT S1,REG_CONF1 ;MANDA EL VALOR AL REG_CONF DEL PUERTO1
9
10 loop:
11    INPUT S1,REG_IN1 ;LEE REG_IN1
12    RL  S1     ;ROTACION A LA IZQUIERDA
13    RL  S1     ;PARA DESPLEGAR BITS
14    RL  S1
15    RL  S1
16    OUTPUT S1,REG_OUT1 ;DEPOSITA VALOR EN REG_OUT1
17 JUMP   loop
```

Tabla 4. 1 Código ensamblador de la implementación de un buffer para probar el módulo de puertos.

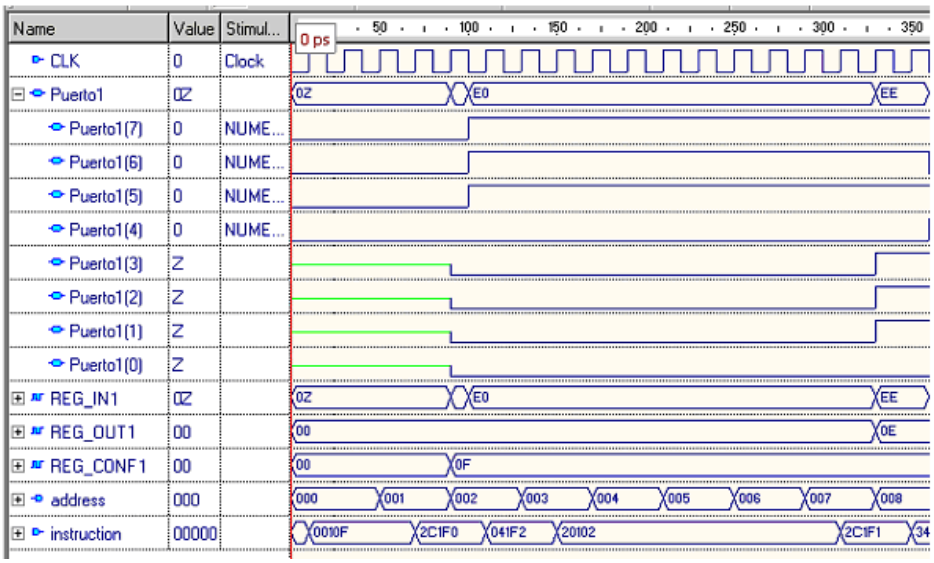


Figura 4. 1 Diagrama de tiempos de la simulación de un buffer con el puerto 1.

En la figura 4.1 se muestra el diagrama de tiempos al realizar una simulación de la prueba del buffer. Las entradas y salidas del puerto se configuran simultáneamente cuando se escribe un valor en el registro *REG_CONF1*. El puerto 1 lee el valor Eh, el cual tarda 11 ciclos de reloj en desplegar el resultado en las terminales configuradas como salida, este retardo es debido al tiempo utilizado en ejecutar el corrimiento de bits.

4.1.2 Prueba del bloque del Preescalador

Se configura al preescalador para que divida la señal de reloj entre un factor de 8, es posible cambiar el valor del registro *s1* de acuerdo a la tabla 3.8 para modificar el factor de preescala. El código se muestra en la tabla 4.2

```
1  ;CONFIGURA EL PREESCALADOR CLK/8
2  CONSTANT    VAL,  FC
3  LOAD     S1,03
4  OUTPUT   S1,VAL      ;CONFIGURA EL PREESCALADOR
```

Tabla 4. 2 Código ensamblador para probar el bloque del preescalador.

En la figura 4.2 se observa la salida del Preescalador *RLJ* dividiendo la señal de reloj en un factor de 8, generando una señal con un período de 160ns.

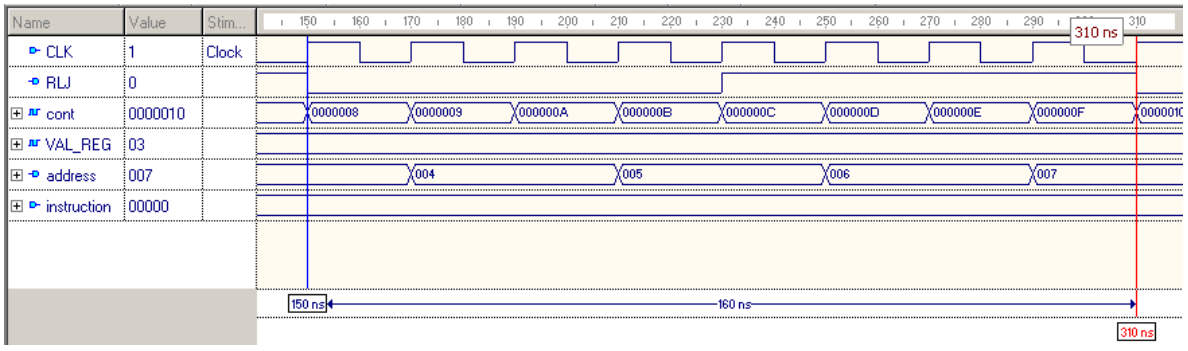


Figura 4. 2 Diagrama de tiempos de la simulación del preescalador.

4.1.3 Prueba del bloque del Contador

En esta práctica el contador trabaja conjuntamente con el Preescalador y uno de los puertos. Primero se configura al puerto 1 como entrada-salida y al preescalador para que divida la señal de reloj entre 8. El PicoBlaze tomará lectura del valor del contador y los bits menos significativos se mostrarán en las terminales de salida del puerto 1. El código se muestra en la tabla 4.3

```
1  ;CONFIGURA EL PREESCALADOR Y DESPLIEGA EL CONTADOR EN PUERTO1
2
3  CONSTANT    REG_CONF1,  F0
4  CONSTANT    REG_OUT1,   F1
5  CONSTANT    VAL_REG,    FC
6
7  LOAD         S1,0F            ;CARGA EL VALOR 00001111,7-4 R,3-0 W
8  OUTPUT    S1,REG_CONF1  ;MANDA EL VALOR AL REG_CONF1
9
10 LOAD         S1,03
11 OUTPUT    S1,VAL_REG       ;CONFIGURA EL PREESCALADOR
12
13 loop:
14    INPUT     S1,FF     ;LEE EL CONTADOR
15    OUTPUT S1,REG_OUT1  ;DEPOSITA VALOR EN REG_OUT1
16    JUMP   loop
```

Tabla 4. 3 Código ensamblador para probar el bloque del contador.

En la figura 4.3 se muestra el diagrama de tiempos de la prueba realizada al contador que se incrementa de acuerdo a la señal que recibe del preescalador. El puerto 1 muestra los 4 bits menos significativos del contador cada 8 ciclos de reloj, este retardo es debido a las operaciones de lectura y escritura.

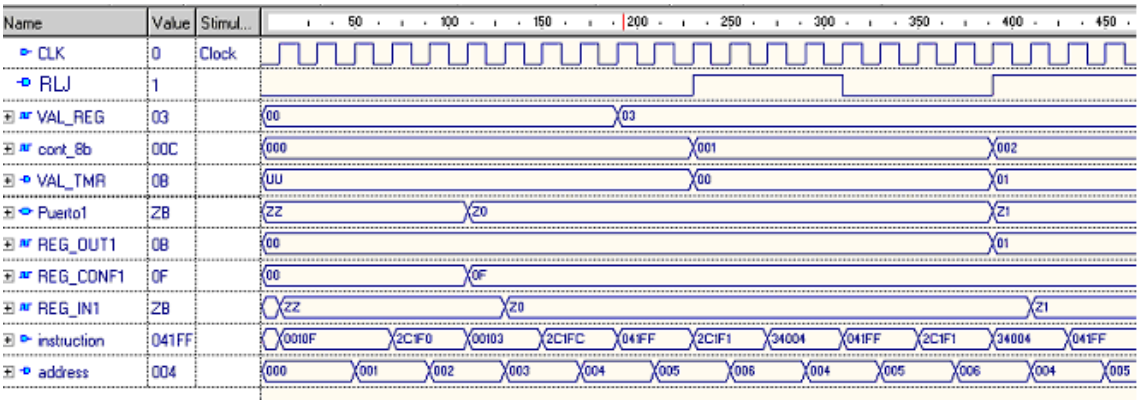


Figura 4. 3 Diagrama de tiempos de la simulación del contador.

4.1.4 Prueba del bloque PWM

Para generar la función de limpieza del contador por comparación, PWM de rampa sencilla o rampa doble, se configuran los 4 bits menos significativos del puerto 3 como salida. Se habilita el bit correspondiente a la función que se desea en el registro *REG_COMP* de

acuerdo a la tabla 2.9 y se deposita un valor de comparación, se muestra un ejemplo del código de cada función en las tablas 4.4 a 4.6.

En la tabla 4.4 se muestra el código para probar la función CTC para que genere una señal con una frecuencia de 1MHz. Se propone trabajar sin preescala. El valor de comparación se calcula con la ecuación 1 [22]:

$$f_{salida}=f_{clk}/(2N(1+REG\_COMP)) \quad (1)$$

Donde:

f_{salida} = Frecuencia de salida
f_{clk} = Frecuencia del reloj principal
N = Factor de preescala
REG_COMP = Valor depositado en el registro de comparación

Donde se despeja la variable que representa al registro de comparación, resultando la ecuación 2:

$$REG\_COMP = (f_{clk}/(2N\, f_{salida}))-1 \quad (2)$$

Sustituyendo los valores resulta REG_COMP = 24 = 18h.

```
1  ;PRUEBA CTC GENERA FREC DE 1MHz
2
3  CONSTANT    REG_CONF3,  ED
4  CONSTANT    CONF_COMP,  F8
5  CONSTANT    VAL_COMP,   F9
6  CONSTANT    VAL_REG,    FC
7
8  LOAD         S1,02       ;CONFIGURA PREESCALADOR ANTICIPADAMENTE
9  OUTPUT       S1,VAL_REG  ;FREC = CLK
10
11 LOAD         S1,0F          ;CARGA EL VALOR 00001111, 7-4 R, 3-0 W
12 OUTPUT       S1,REG_CONF3 ;MANDA 0F AL REG_CONF DEL PUERTO3
13
14 LOAD         S1,02       ;HAB EL CTC DEL CONF_COMP
15 OUTPUT       S1,CONF_COMP
16
17 LOAD         S1,18       ;VALOR DE COMPARACION
18 OUTPUT       S1,VAL_COMP
```

Tabla 4. 4 Código ensamblador para generar la función CTC.

En la figura 4.4 se observa que en un principio se activa la señal de igualdad de comparación *(match)* debido a que el valor del registro *VAL_COMP* es cero, igual que el valor del contador *cont_8b*. La señal *match* se habilita un ciclo de reloj después de ocurrir la coincidencia. Pero no se activa la señal de limpieza del contador *CLR* ni se genera una señal CTC debido a que aún no se ha habilitado la función en el registro *CONF_COMP*.

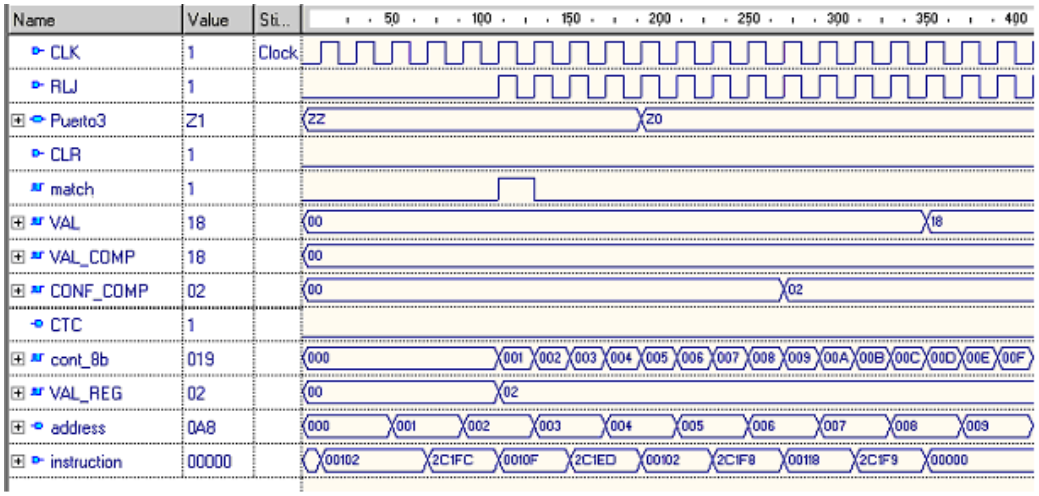


Figura 4. 4 Diagrama de tiempos de la simulación de la función CTC.

En la figura 4.5 se observa la señal generada en la terminal de salida CTC del módulo temporizador y simultáneamente en la terminal 0 del puerto 3, reservada para dicha señal, que tiene una frecuencia aproximada de 976KHz, es decir, un período de 1040ns, esto debido a que la señal *CLR* tarda un ciclo de reloj en actualizarse. Acumulándose dos retardos en cada ciclo de la señal. Cabe aclarar que el proceso de comparación depende del contador de 8 bits, que en este caso coincide con la frecuencia del reloj principal.

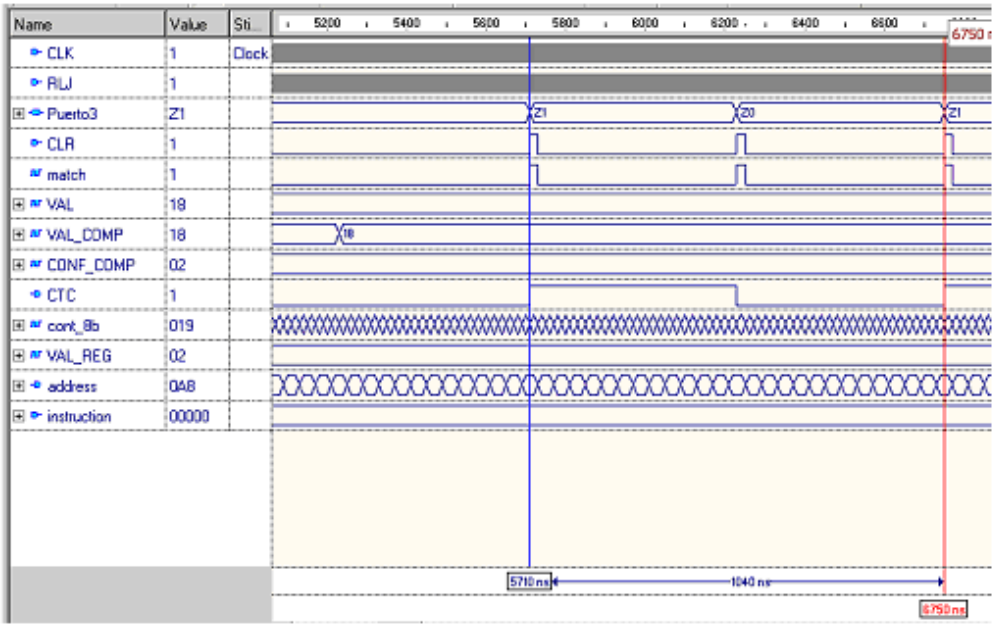


Figura 4. 5 Diagrama de tiempos de la simulación de la función CTC sin preescalador.

Si se genera la misma frecuencia, pero con un factor de preescala de 8, al sustituir los valores en la ecuación 2, el valor de comparación resultaría 2. Al realizar la simulación se obtiene un período de 1280ns. En este caso el valor del contador tarda 8 ciclos de reloj principal en actualizarse, o un ciclo de la señal RLJ, es decir, 160ns. Dicha señal proviene del Preescalador. Cada período de la señal CTC dura 8 ciclos de la señal RLJ, resultando 160ns*8 = 1280ns. Observe la figura 4.6.

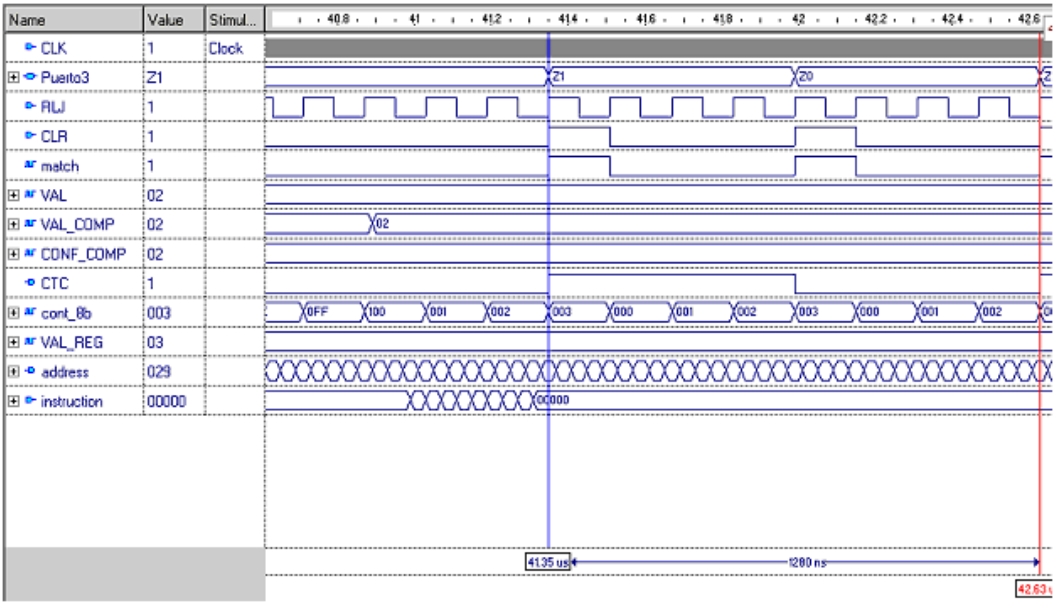


Figura 4. 6 Diagrama de tiempos de la simulación de la función CTC con preescalador igual a 8 veces la frecuencia del reloj principal.

En la figura 4.7 se muestra la imagen obtenida en el osciloscopio que corresponde a la señal CTC generada a través del FPGA Spartan 3E. La señal tiene una frecuencia de 960KHz aproximadamente, lo que corresponde a los cálculos realizados

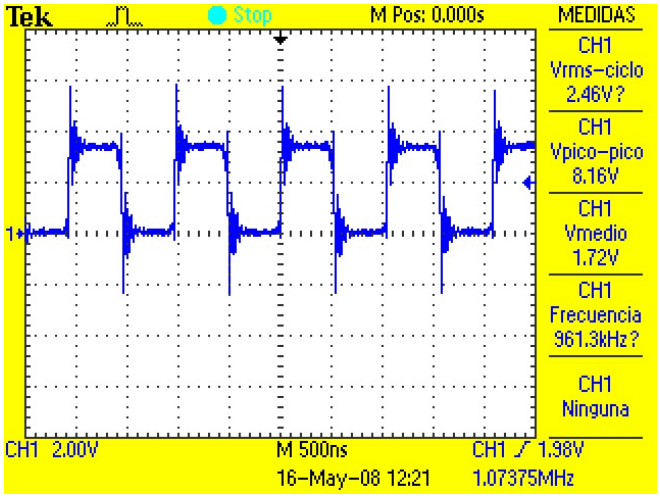


Figura 4.7 Imagen obtenida del osciloscopio de la función CTC.

Otra función que genera este bloque, es la función pwm de rampa sencilla, cuya frecuencia se determinada por la ecuación 3 [22].

$$f_{pwm} = f_{clk}/(256N) \quad (3)$$

Donde:

f_{pwm} = frecuencia de la señal pwm.
f_{clk} = frecuencia de la señal de reloj principal
N = factor de preescala

Despejando la variable N se obtiene:

$$N = f_{clk}/(256\, f_{pwm}) \qquad (4)$$

Si se desea generar una frecuencia de 195KHz, sustituimos los valores en la ecuación 4 y obtenemos un valor aproximado de N=1.

El valor de comparación modificará el ciclo de trabajo. Un valor de comparación de 256 resultaría un ciclo de trabajo de 100%. Para obtener un ciclo de trabajo de 80% se calcula la proporción, generándose un valor de comparación aproximado de 204d = CCh. La tabla 4.5 muestra el código para probar la función pwm de rampa sencilla con los datos obtenidos anteriormente.

```
1   ;PRUEBA PWM. FREC=195KHz, CICLO DE TRABAJO 80%
2   CONSTANT    REG_CONF3,  ED
3   CONSTANT    CONF_COMP,  F8
4   CONSTANT    VAL_COMP,   F9
5   CONSTANT    VAL_REG,    FC
6
7   LOAD         S1,0F          ;CARGA EL VALOR 00001111, 7-4 R, 3-0 W
8   OUTPUT    S1,REG_CONF3 ;MANDA EL VALOR AL REG_CONF DEL PUERTO3
9   LOAD         S1,04       ;HABILITA EL PWM DE RAMPA SENCILLA
10  OUTPUT    S1,CONF_COMP
11  LOAD         S1,CC       ;VALOR DE COMPARACION VAL_COMP
12  OUTPUT    S1,VAL_COMP
13  LOAD         S1,02
    OUTPUT    S1,VAL_REG     ;CONFIGURA EL PREESCALADOR
```

Tabla 4. 5 Código ensamblador para generar la función PWM.

La figura 4.8 muestra el inicio de la simulación de la prueba de PWM, donde se aprecia la señal *SEL_FUNC*, esta señal controla el multiplexor que permite que la Terminal 0 del puerto 3 no sea de propósito general, sino que envíe la función PWM. También se observa un evento de igualdad en la comparación, pero no se genera alguna señal en la terminal FPWM, ésta se comenzará a generar a partir de que exista un desborde y se actualice el registro *VAL_COMP* al siguiente ciclo de reloj. Esto último se observa en la figura 4.9.

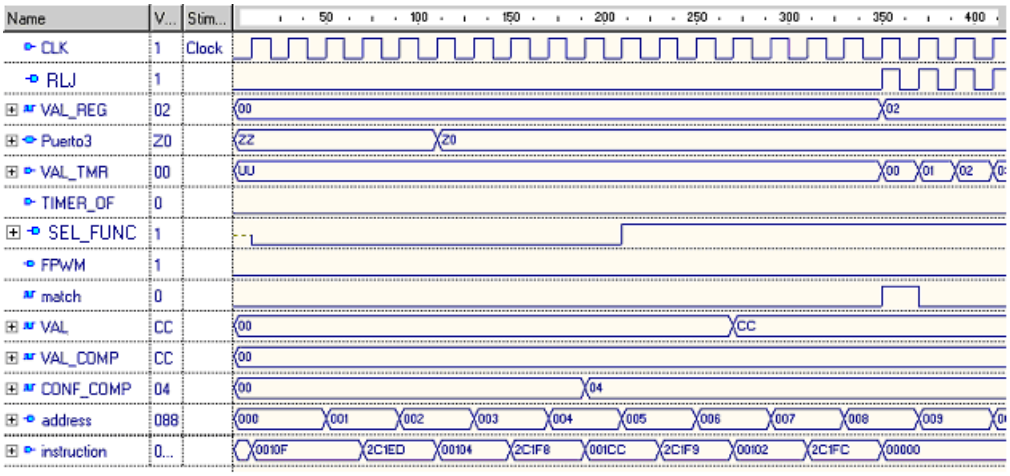


Figura 4. 8 Diagrama de tiempos al inicio de la simulación de la función PWM de rampa sencilla.

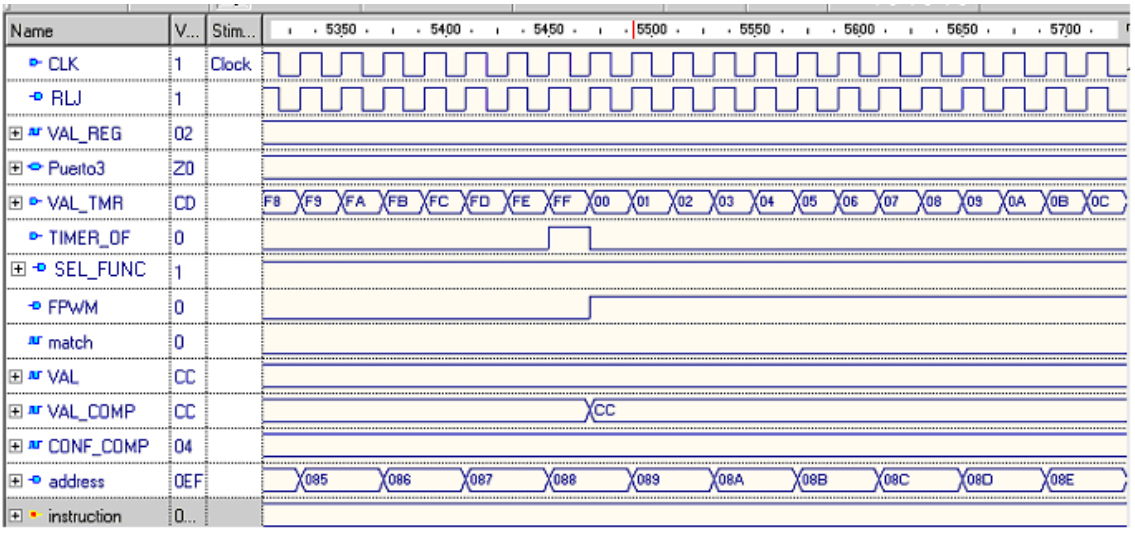


Figura 4. 9 Evento de sobreflujo y actualización del registro VAL_COMP.

La figura 4.10 muestra el pulso de la señal *match*, que indica una igualdad en la comparación del valor contenido en el registro *VAL_COMP* y el valor del contador. Al siguiente ciclo de reloj conmuta la señal *PWM*. La figura 4.11 muestra el evento de sobreflujo y reinicio de la función *PWM* en el siguiente ciclo de reloj.

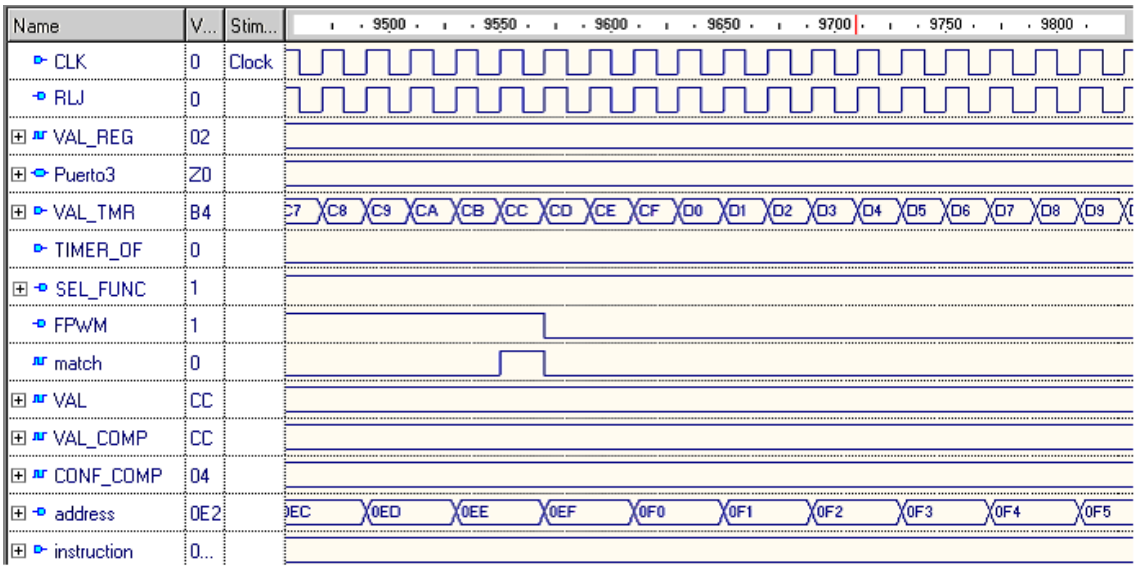


Figura 4. 10 Evento de igualdad en la comparación y cambio en la señal PWM.

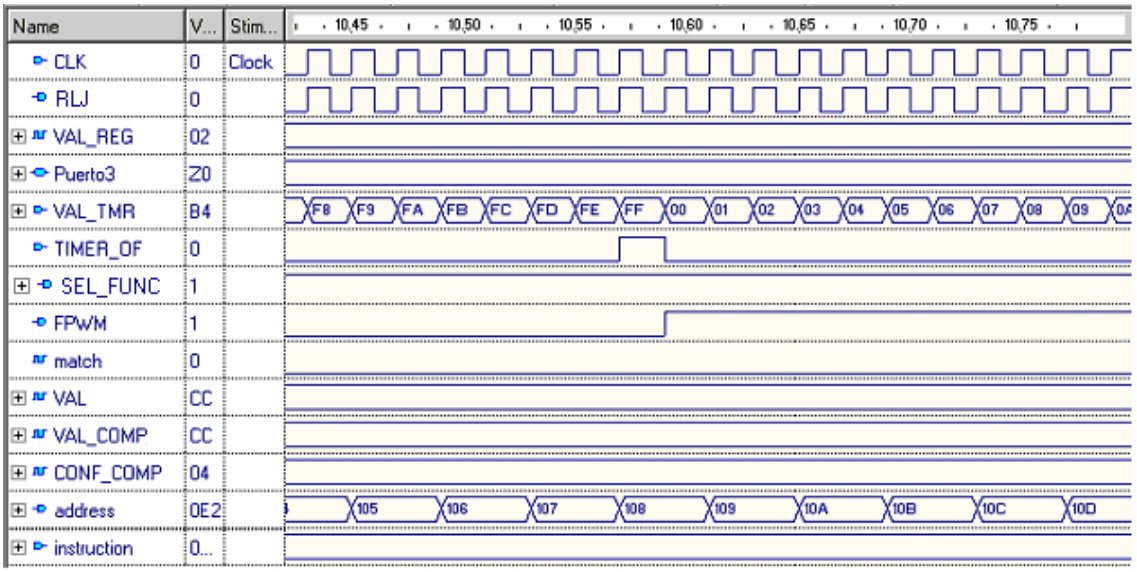


Figura 4. 11 Evento de sobreflujo y reinicio de la función PWM.

La figura 4.12 muestra la señal generada de 5120ns, que es igual a 195.3KHz. El pulso en alto es de 4100ns el cual representa el 80% del período de la señal.

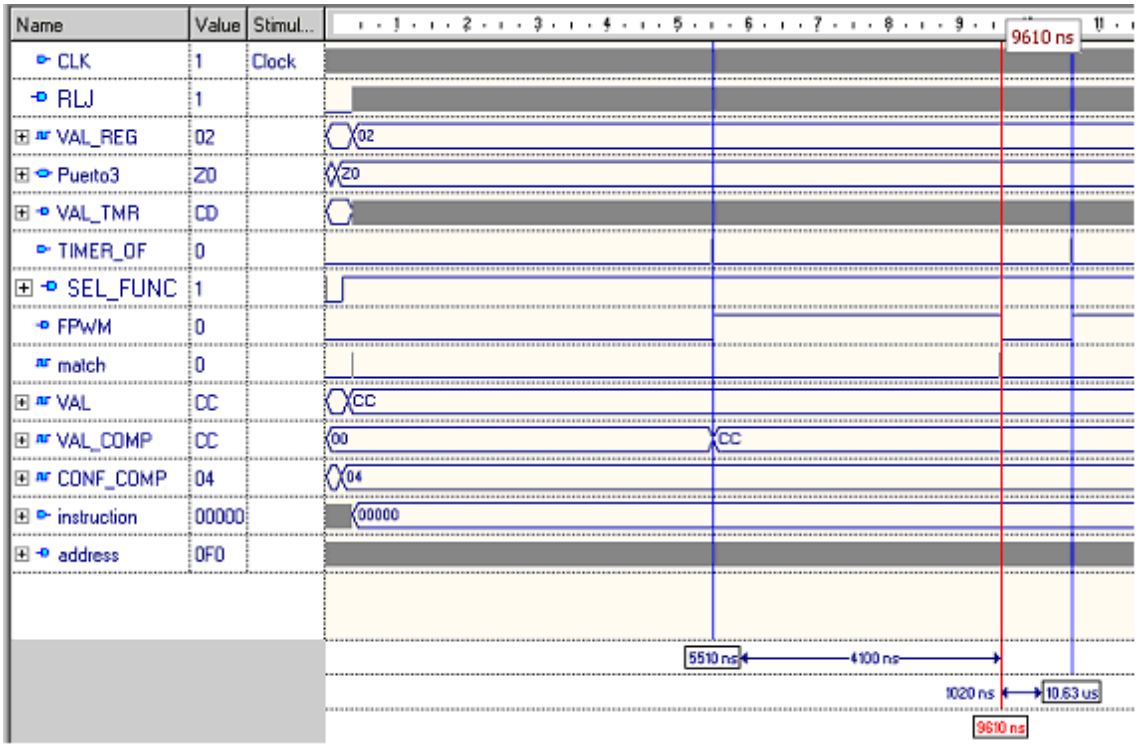


Figura 4. 12 Función PWM de rampa sencilla.

En la figura 4.13 se muestra la imagen obtenida en el osciloscopio que corresponde a la función PWM de rampa sencilla generada a través del FPGA Spartan 3E. La señal tiene una frecuencia de 195KHz aproximadamente, lo que corresponde a los cálculos realizados

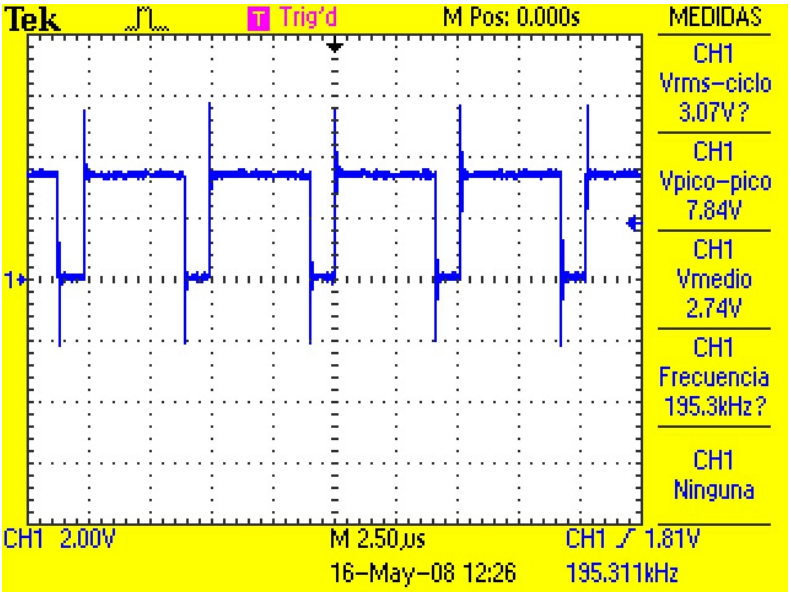


Figura 4. 13 Imagen obtenida del osciloscopio de la función PWM de rampa sencilla.

La tercera función que se genera es la de PWM de rampa doble, cuya frecuencia se determinada por la ecuación 5.

$$fd_{pwm} = f_{clk}/(510N) \quad (5)$$

Donde:

fd_{pwm} = frecuencia de la señal pwm de rampa doble
f_{clk} = frecuencia de la señal de reloj principal
N = factor de preescala

Despejando la variable N se obtiene:

$$N = f_{clk}/(510\, fd_{pwm}) \quad (6)$$

Si se desea obtener una frecuencia 98KHz en la función de PWM de rampa doble, se sustituyen los valores $fclk$ = 50x10e6 y $fdpwm$ = 9,8x10e3 en la ecuación 6 y se obtiene el valor de N=1.0004 o aproximadamente N=1.

Debido a que la función PWM de rampa doble requiere de un contador ascendente-descendente de 8 bits, el 100% del ciclo de trabajo lo representa el valor 512, si se propone un ciclo de trabajo de 6% se debe obtener la proporción para obtener el valor de comparación, el cual resulta con 30.7, pero ese valor debe ser divido entre dos, ya que la comparación se realiza 2 veces debido al contador ascendente-descendente, por tanto el valor de comparación es 30.7/2=15.35. Se utilizará el valor de comparación aproximado de 15=0Fh.

En la tabla 4.6 se muestra un código para generar la función PWM de rampa doble sin preescalador y un valor de comparación 0Fh.

```
1   ;PRUEBA PWM DE RAMPA DOBLE
2
3   CONSTANT    REG_CONF3,  ED
4   CONSTANT    CONF_COMP,  F8
5   CONSTANT    VAL_COMP,   F9
6   CONSTANT    VAL_REG,    FC
7
8   LOAD          S1,0F          ;CARGA EL VALOR 00001111, 7-4 R, 3-0 W
9   OUTPUT    S1,REG_CONF3 ;MANDA EL VALOR AL REG_CONF3
10  LOAD          S1,08       ;HAB EL PWM DE DOBLE RAMPA
11  OUTPUT    S1,CONF_COMP
12
13  LOAD          S1,0F       ;VALOR DE COMPARACION
14  OUTPUT    S1,VAL_COMP
15
16  LOAD          S1,02
17  OUTPUT    S1,VAL_REG     ;CONFIGURA EL PREESCALADOR
```

Tabla 4. 6 Código ensamblador para generar la función PWM de rampa doble.

La figura 4.14 muestra la actualización del registro de configuración *CONF_COMP* para que se realice la función PWM de rampa doble y un ciclo de reloj después la señal *SEL_FUNC* habilita el control para multiplexar la Terminal 0 del puerto 3. Observe que debido a que ya se habilitó la función y ocurre un evento de igualdad, se envía la señal de la función a través de la terminal DPWM.

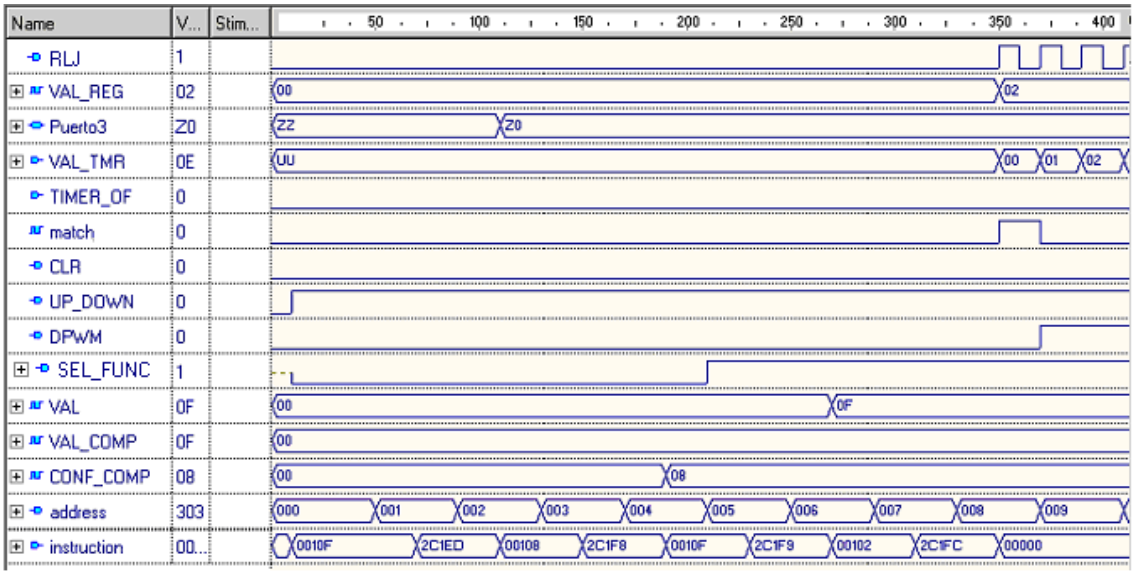


Figura 4. 14 Inicio del diagrama de tiempo de la simulación de la función PWM de rampa doble.

En la figura 4.15 se muestra el evento de sobreflujo en la terminal *TIMER_OF* y el cambio de la señal *UP_DOWN* de '1' a '0', lo cual modifica al contador de modo ascendente a descendente. Se observa también la actualización del registro de comparación *VAL_COMP* un ciclo de reloj después del evento de sobreflujo.

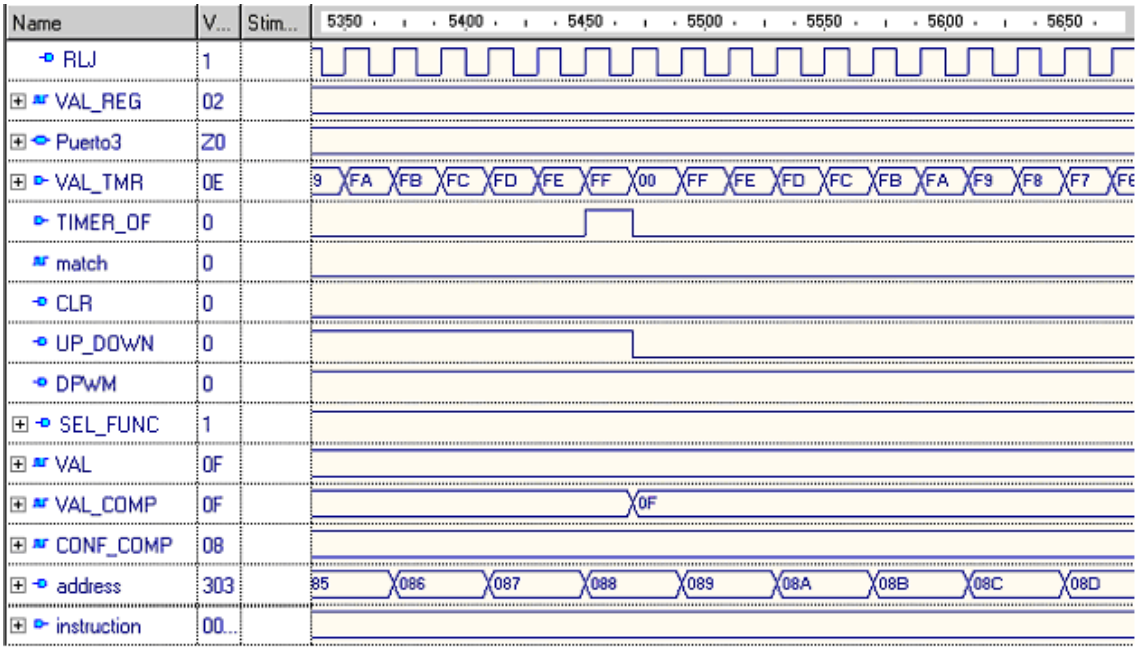


Figura 4. 15 Evento de sobreflujo y actualización del registro *VAL_COMP*.

En la figura 4.16 se muestra el valor del contador en modo descendente en la terminal *VAL_TMR* y la señal *match_s* que indica que hubo un evento de igualdad en el proceso de comparación, la cual conmuta el valor de la señal en la terminal *DPWM*.

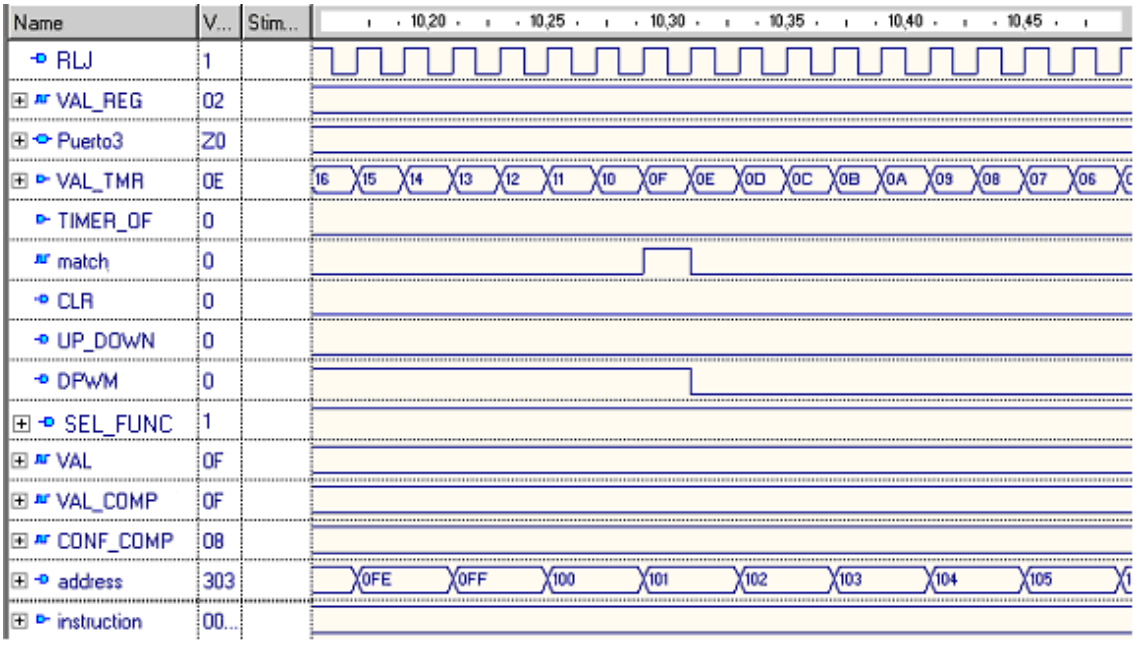


Figura 4. 166 Evento de igualdad en el proceso de comparación y conmutación de la señal PWM de doble rampa.

La figura 4.17 muestra el segundo evento de sobreflujo y se observa el cambio del contador de modo descendente a ascendente.

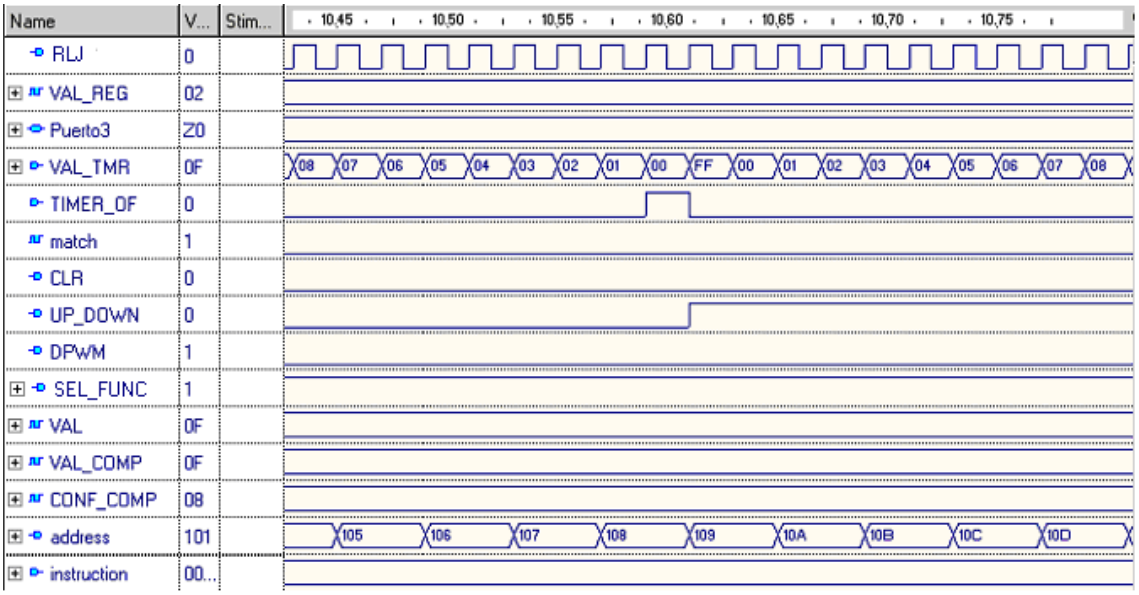


Figura 4. 17 Segundo evento de sobreflujo.

La figura 4.18 muestra el segundo evento de igualdad en el proceso de comparación. En la figura 4.19 se muestra en el mismo diagrama de tiempo, los eventos descritos en las figuras 4.16 a la 4.18.

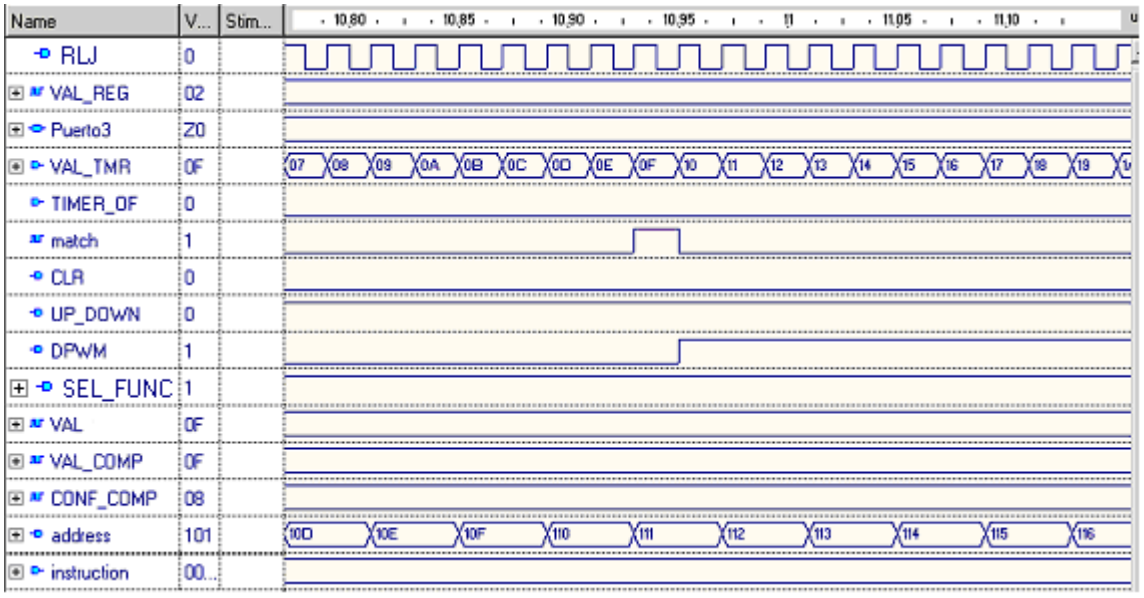


Figura 4. 188 Segundo evento de igualdad en el proceso de comparación.

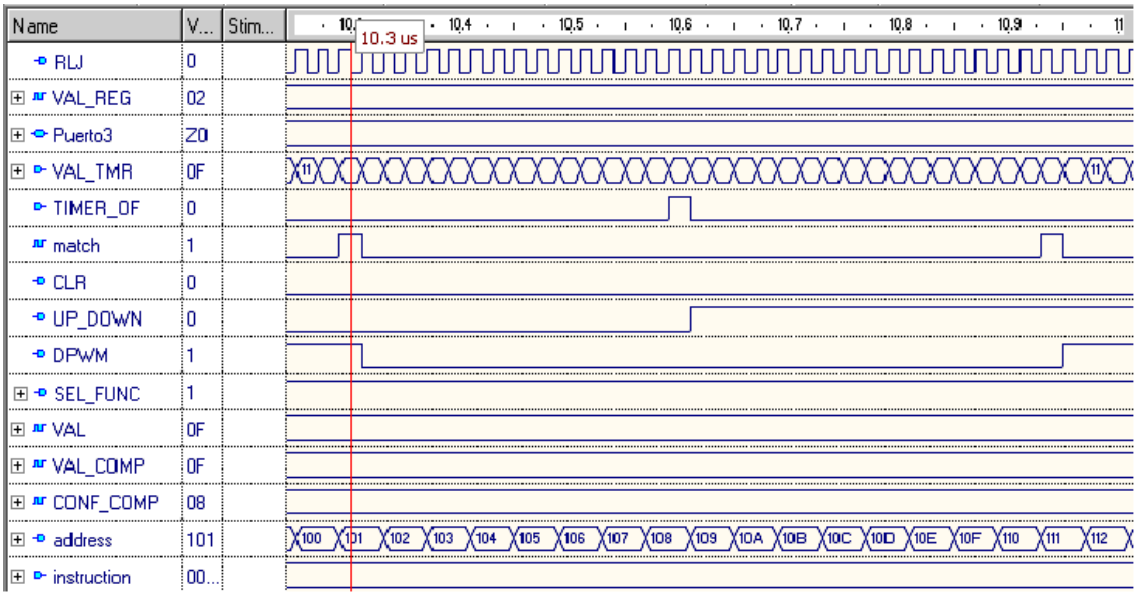


Figura 4. 19 Evento de sobreflujo y conmutación de la terminal DPWM en cada evento de igualdad en el proceso de comparación.

En la figura 4.20 se observa el período de la señal de PWM de rampa doble que es de 10.28uS, es decir de una frecuencia de 97.27KHz y una relación en el ciclo de trabajo de 93.8% que es el complemento de 6.2%. La señal de salida está invertida, dado que se esperaba un ciclo útil al 6 %. Esto es debido a que el preescalador se configuró después de habilitar la función PWM de rampa doble, lo que ocasionó que ocurriera un evento de igualdad cuando el contador se encontraba en 00h.

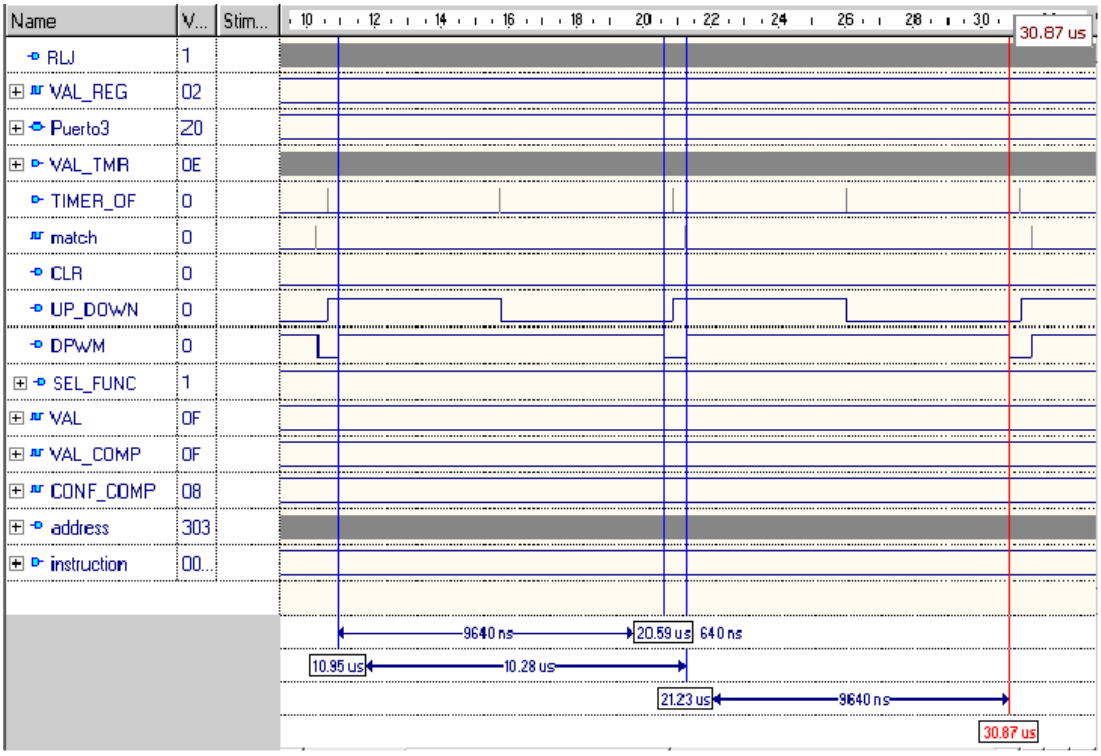


Figura 4. 20 Diagrama de tiempo de la función PWM de rampa doble con ciclo de trabajo de 93.8%.

Para corregir esta situación, se debe configurar al preescalador antes de habilitar la función de PWM, como se observa en la figura 4.21, donde se muestra que el contador inicia antes de que se habilite la función PWM de rampa doble, incluso ocurre un evento de igualdad en el proceso de comparación, pero no inicia la función PWM y finalmente en la figura 4.22 se observa la señal sin ser invertida y con un ciclo de trabajo de 6.2%.

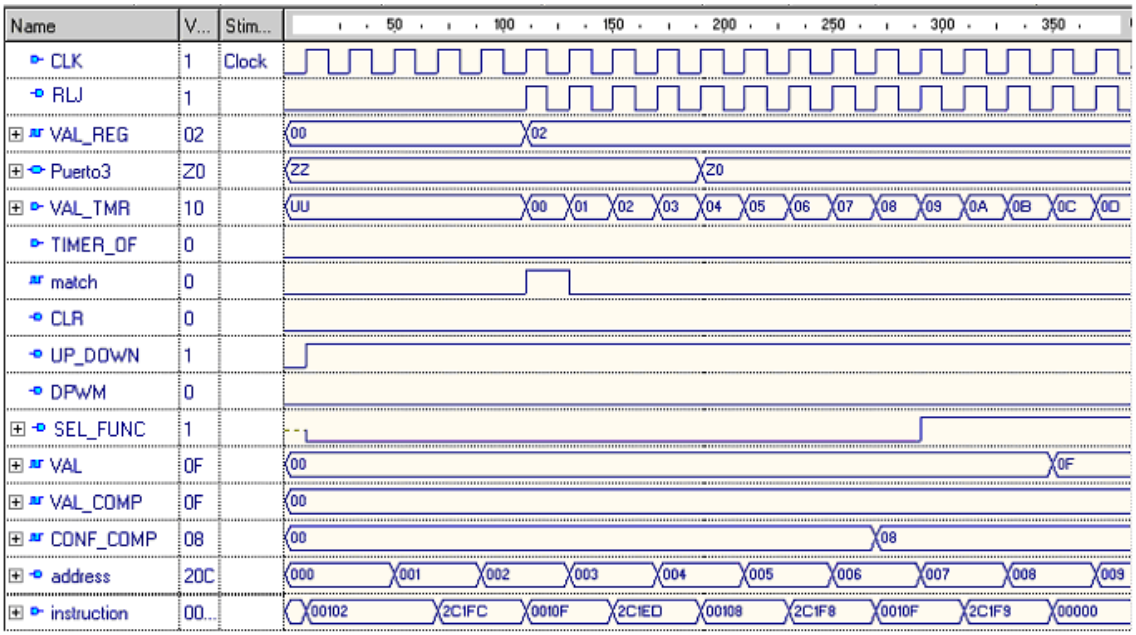


Figura 4. 21 Inicio de la segunda simulación de la función PWM de rampa doble.

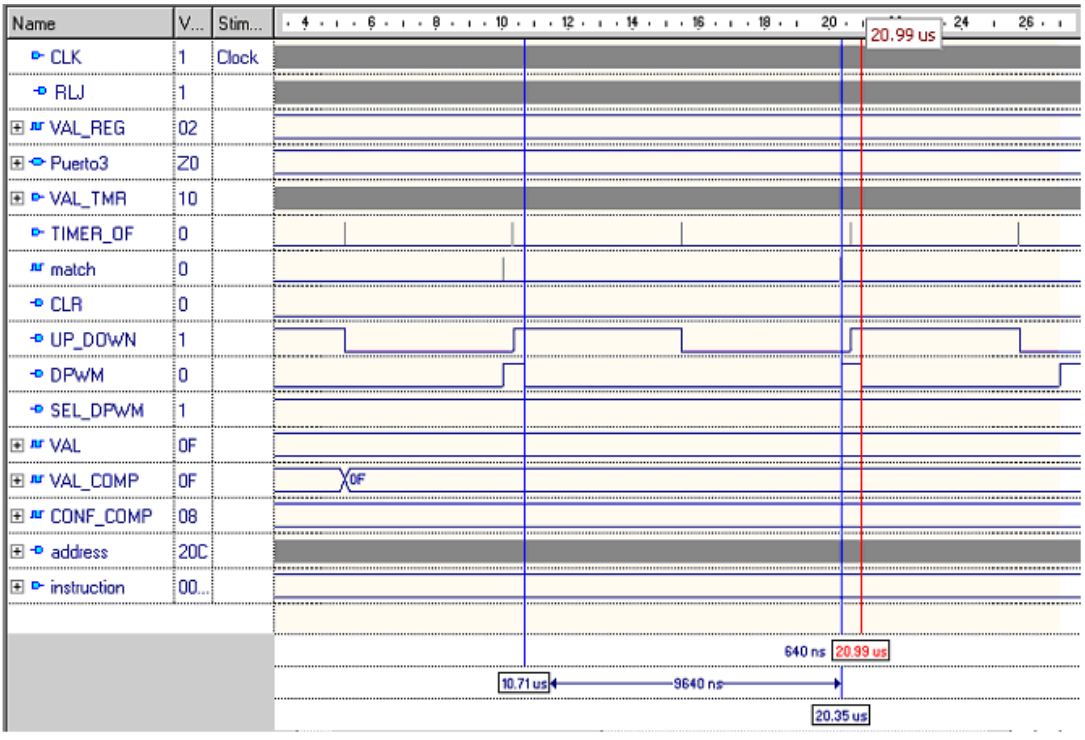


Figura 4. 22 Diagrama de tiempo de la función PWM de rampa doble con ciclo de trabajo de 6.2%.

En la figura 4.23 se muestra la imagen obtenida en el osciloscopio que corresponde a la función PWM de rampa doble generada a través del FPGA Spartan 3E. La señal tiene una frecuencia de 97KHz aproximadamente, lo que corresponde a los cálculos realizados

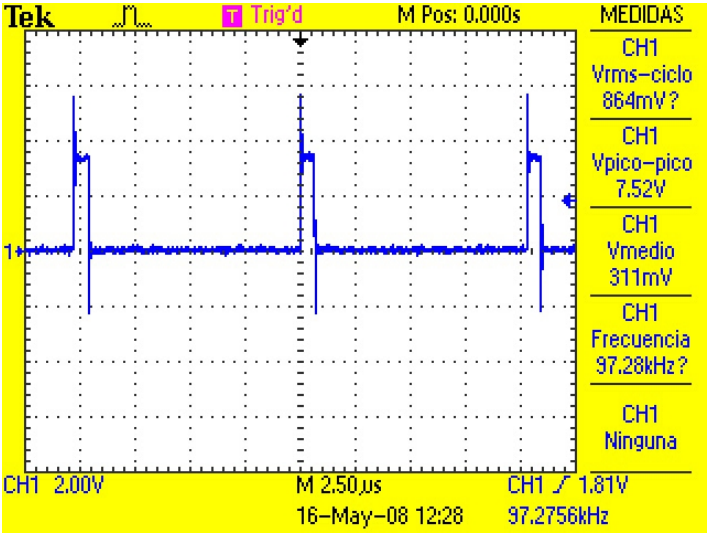


Figura 4. 23 Imagen obtenida del osciloscopio de la función PWM de rampa doble.

4.1.5 Prueba del Módulo de Interrupciones

En la tabla 4.7 se muestra un código ensamblador donde se configura el preescalador para funcionar sin preescala, el registro *CONF_INT* para habilitar la interrupción por desborde y el puerto 1 para que funcione como puerto de entrada y salida. Las interrupciones se habilitan por software con la instrucción *ENABLE INTERRUPT*. Posteriormente se implementa un ciclo infinito, que no realiza alguna acción, ya que carga el contenido de un registro dentro de sí mismo. Cuando ocurre un desborde del bloque del contador, se ejecuta la rutina ISR que envía el valor *0F* al puerto 1, esto es únicamente con la finalidad de mostrar una señal que el usuario pueda visualizar, por ejemplo, a través de un LED en la tarjeta de desarrollo empleada. Antes de que retorne del servicio de interrupción, es necesario limpiar en el registro estado, el bit que indica la causa de la interrupción, ya que en caso contrario se volvería a ejecutar inmediatamente la ISR.

```
1  ;INTERRUPCION POR DESBORDE
2  CONSTANT    REG_CONF1,  F0
3  CONSTANT    REG_OUT1,   F1
4  CONSTANT    REG_IN1,    F2
5  CONSTANT    INT_STATE,  FA
6  CONSTANT    INT_CONF,   FB
7  CONSTANT    VAL_REG,    FC
8  LOAD         S1,02
9  OUTPUT    S1,VAL_REG    ;CONFIGURA EL PREESCALADOR
10 LOAD         S1,81       ;CARGA 10000011 ->HAB GLOBAL, INT OF
11 OUTPUT    S1,INT_CONF   ;CONFIGURA INTERRUPCION
12 LOAD         S1,0F       ;CARGA EL VALOR 00001111, 7-4 R, 3-0 W
13 OUTPUT    S1,REG_CONF1 ;MANDA EL VALOR AL REG_CONF1
14 ENABLE INTERRUPT
```

Tabla 4. 7 Código ensamblador para implementar una rutina de servicio de interrupción por desborde.

```
17 LOOP:
18     LOAD   S9,S9
19     JUMP LOOP
20
21 ISR:
22 INT_OF:
23     LOAD         S2,0F          ;VALOR PARA ENVIAR AL HABER INT
24     OUTPUT    S2,REG_OUT1    ;DEPOSITA EL VALOR EN REG_OUT1
25     LOAD   S4,04
26
27 RETARDO:
28     SUB S4,01
29     JUMP   NZ, RETARDO
30
31     LOAD   S2,00
32     OUTPUT S2,REG_OUT1    ;LIMPIA EL VALOR AL ACABAR LA INT
33     INPUT  S5,INT_STATE ;LEE EL REG DE ESTADO DE LAS INT
34     AND       S5,FE
35     OUTPUT    S5,INT_STATE ;LIMPIA EL REG DE ESTADO DE LAS INT
36     RETURNI  ENABLE
37 ;
38 ;*********************************************************************
39 ;Interrupt Vector
40 ;*********************************************************************
41 ;
42 ADDRESS 3FF
43 JUMP ISR
```

Tabla 4.7 Código ensamblador para implementar una rutina de servicio de interrupción por desborde (continuación).

Al código de la tabla 4.7 se le agregaron instrucciones para poder detectar la interrupción externa, además de la interrupción por sobreflujo. La terminal 7 del puerto 2 está reservada para la interrupción externa y es necesaria su habilitación para que pueda ser utilizada.

Dado que ahora se manejarán dos eventos, cuando ocurra una interrupción, se debe comparar al registro de estado con el valor que representa al bit que probablemente haya disparado la interrupción. En caso de existir una coincidencia, se da un salto a la rutina correspondiente. Esto se observa en el código de la tabla 4.8, donde la interrupción por desborde envía el valor *0F* al puerto 1 y la interrupción externa envía el valor "04" al puerto 2.

```
1  ;INTERRUPCION EXTERNA Y POR SOBREFLUJO
2
3  CONSTANT    REG_CONF1,  F0
4  CONSTANT    REG_OUT1,   F1
5  CONSTANT    REG_CONF2,  F3
6  CONSTANT    REG_OUT2,   F4
7  CONSTANT    INT_STATE,  FA
8  CONSTANT    INT_CONF,   FB
9  CONSTANT    VAL_REG,    FC
10
11 ENABLE INTERRUPT
```

Tabla 4. 8 Código ensamblador para implementar una rutina de servicio de interrupción externa.

```
14 LOAD         S1,02
15 OUTPUT    S1,VAL_REG   ;CONFIGURA EL PREESCALADOR
16 LOAD         S1,83       ;10000011->HAB GLOBAL, INT EXT, INT OF
17 OUTPUT    S1,INT_CONF   ;CONFIGURA INT
18 LOAD         S1,0F       ;CARGA EL VALOR 00001111, 7-4 R, 3-0 W
19 OUTPUT    S1,REG_CONF1 ;MANDA EL VALOR AL REG_CONF1
20 OUTPUT    S1,REG_CONF2 ;MANDA EL VALOR AL REG_CONF2
21 loop:
22    INPUT  S1,FF   ;LEE EL VALOR DEL CONTADOR
23    OUTPUT S1,REG_OUT1 ;DEPOSITA EL VALOR EN EL PUERTO 1 (F1)
24    JUMP    loop
25
26 ISR:
27    INPUT  S1,INT_STATE
28    AND S1,0F
29    TEST   S1,01
30    JUMP   NZ,INT_OF   ;ATIENDE LA INTERRUPCIÓN POR SOBREFLUJO
31    TEST   S1,02
32    JUMP   NZ,INT_EXT  ;ATIENDE LA INTERRUPCIÓN EXTERNA
33    RETURNI  ENABLE
34
35 INT_EXT:
36    LOAD    S2,04 ;VALOR PARA ENVIAR AL HABER INT
37    LOAD    S4,02
38 RETARDO2:
39    OUTPUT S2,REG_OUT2 ;DEPOSITA EL VALOR EN PUERTO2 (F4)
40    SUB S4,01
41    JUMP   NZ, RETARDO2
42    LOAD   S2,00
43    OUTPUT   S2,REG_OUT2   ;ENVIA '0' AL ACABAR LA INT
44    INPUT    S5,INT_STATE  ;LEE EL REG DE ESTADO DE INT
45    AND   S5,FD
46    OUTPUT   S5,INT_STATE  ;LIMPIA EL REG DE ESTADO
47       RETURNI ENABLE
48
49 INT_OF:
50    LOAD         S2,0F        ;VALOR PARA ENVIAR AL HABER INT
51    OUTPUT    S2,REG_OUT1 ;DEPOSITA EL VALOR EN REG_OUT1
52    LOAD   S4,04
53 RETARDO:
54    SUB S4,01
55    JUMP   NZ, RETARDO
56    LOAD   S2,00
57    OUTPUT S2,REG_OUT1      ;LIMPIA EL VALOR DEL PUERTO1
58    INPUT  S5,INT_STATE ;LEE EL REG DE ESTADO DE INT
59    AND      S5,FE
60    OUTPUT    S5,INT_STATE ;LIMPIA EL REGISTRO DE ESTADO
61    RETURNI  ENABLE
62 ;*****************************************************************
63 ;Interrupt Vector
64 ;*****************************************************************
65 ;
66 ADDRESS 3FF
67 JUMP ISR
```

Tabla 4.8 Código ensamblador para implementar una rutina de servicio de interrupción externa (continuación).

La figura 4.24 muestra el evento de sobreflujo dado por la señal *TIMER_OF* y el reconocimiento de la interrupción indicado por la señal *int_ack*. En el transcurso de la interrupción, existe también un evento de interrupción externa y un evento de igualdad en el proceso de comparación debido a que el registro *VAL_COMP* y el contador están en cero, observe el cambio del registro de estado *INT_STATE*, de 05 a 07. La interrupción por comparación no se atiende por no estar habilitada. El evento externo no es atendido sino hasta que finaliza la interrupción por sobreflujo y cambia nuevamente el valor del registro de estado como se muestra en la figura 4.25.

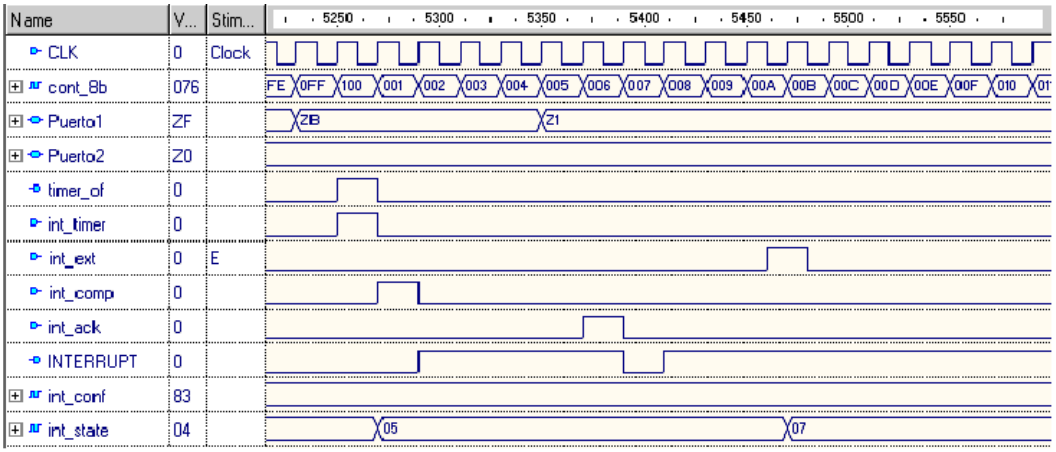


Figura 4. 24 Inicio de interrupción por sobreflujo.

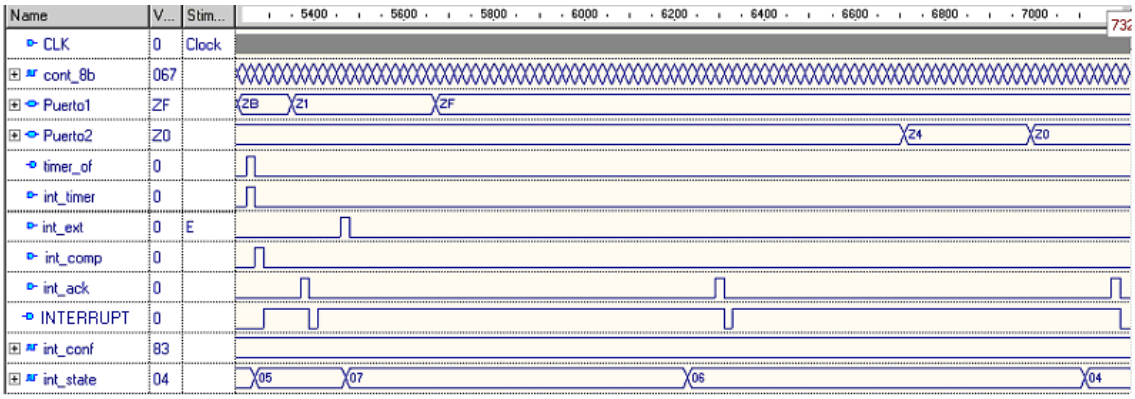


Figura 4. 25 Evento y reconocimiento de interrupción externa y por sobreflujo.

La tabla 4.9 muestra un código para implementar una rutina de servicio de interrupción externa habilitada por un flanco de bajada, la diferencia está en la configuración del registro *INT_CONF*.

```
1  ;INT FLANCO DE BAJADA
2  CONSTANT    REG_CONF1,   F0
3  CONSTANT    REG_OUT1,    F1
4  CONSTANT    REG_CONF2,   F3
5  CONSTANT    REG_OUT2,    F4
6  CONSTANT    INT_STATE,   FA
7  CONSTANT    INT_CONF,    FB
8  CONSTANT    VAL_REG,     FC
```

Tabla 4. 9 Código ensamblador para implementar una rutina de servicio de interrupción externa habilitada por un flanco de bajada

```
ENABLE INTERRUPT

LOAD         S1,02
OUTPUT       S1,VAL_REG    ;CONFIGURA EL PREESCALADOR

LOAD         S1,92         ;10010010 ->HAB GLOBALFLANCO BAJADA, EXT
OUTPUT       S1,INT_CONF      ;CONFIGURA INTERRUPCION

LOAD         S1,0F       ;CARGA EL VALOR 00001111, 7-4 R, 3-0 W
OUTPUT       S1,REG_CONF1 ;MANDA EL VALOR AL REG_CONF1
OUTPUT       S1,REG_CONF2 ;MANDA EL VALOR AL REG_CONF2

loop:
   LOAD   s9,s9
   JUMP   loop

ISR:

INT_EXT:    ;INTERRUPCIÓN EXTERNA
   LOAD   S2,04  ;VALOR PARA ENVIAR AL HABER INT
   LEDS3:
      OUTPUT  S2,REG_OUT2 ;DEPOSITA EL VALOR EN REG_OUT2
      SUB  S3,01
      JUMP     NZ, LEDS3

      LOAD     S2,00
      OUTPUT  S2,REG_OUT2 ;ENVIA 00

      INPUT   S5,INT_STATE       ;LEE EL REG  DE EDO.
      AND  S5,FD
      OUTPUT  S5,INT_STATE       ;LIMPIA REGISTRO DE EDO

   RETURNI ENABLE
;***************************************************************
;Interrupt Vector
;***************************************************************
;
ADDRESS 3FF
JUMP ISR
;
;
```

Tabla 4.9 Código ensamblador para implementar una rutina de servicio de interrupción externa habilitada por un flanco de bajada (continuación).

La figura 4.26 muestra el pulso de la señal *INT_EXT*, que de acuerdo a la señal *SEL_EXT*, ha detectado un flanco de bajada, esto se refleja en el siguiente ciclo de reloj en el registro de estado *INT_STATE*, bajo estas condiciones se habilita la interrupción externa, porque se encuentran habilitados sus permisos individual y global en el registro *INT_CONF*. La rutina que atiende la interrupción envía un pulso en la terminal 2 del puerto 2.

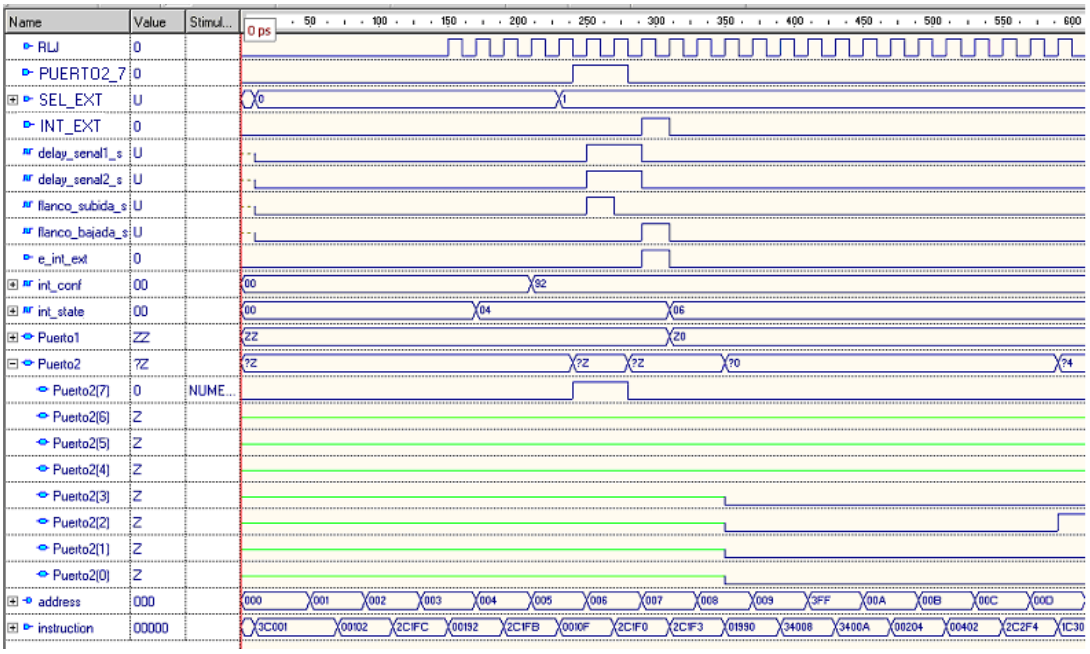


Figura 4. 26 Diagrama de tiempos de la simulación de interrupción externa habilitada por un flanco de bajada.

La tabla 4.10 muestra un código para implementar una rutina de servicio de interrupción por comparación y la lógica de programación es similar a los ejemplos anteriores, se habilita el permiso global e individual de la interrupción, a excepción de que se deposita un valor a comparar en el registro *VAL_COMP*, cuando dicho valor sea igual al del bus *VAL_TMR* se ejecuta la ISR que envía el valor "0F" al puerto 3.

```
1   ; INTERRUPCION POR COMPARACION
2
3   CONSTANT    REG_CONF3,  ED
4   CONSTANT    REG_OUT3,   EE
5
6   CONSTANT    REG_CONF1,  F0
7   CONSTANT    REG_OUT1,   F1
8
9   CONSTANT    VAL_COMP,   F9
10  CONSTANT    INT_STATE,  FA
11  CONSTANT    INT_CONF,   FB
12  CONSTANT    VAL_REG,    FC
13
14  ENABLE INTERRUPT
15
16  LOAD          S1,84       ;HABILITA INT COMPARACION
17  OUTPUT    S1,INT_CONF     ;CONFIGURA INT
18  LOAD          S1,40       ;VALOR DE COMPARACION VAL_COMP
19  OUTPUT    S1,VAL_COMP
20  LOAD          S1,0F       ;CARGA EL VALOR 00001111, 7-4 R, 3-0 W
21  OUTPUT    S1,REG_CONF1 ;MANDA EL VALOR AL REG_CONF1(F0)
22  OUTPUT    S1,REG_CONF3 ;MANDA EL VALOR AL REG_CONF3(F3)
23  LOAD          S1,09
24  OUTPUT    S1,VAL_REG      ;CONFIGURA EL PREESCALADOR
```

Tabla 4. 10 Código ensamblador para implementar una rutina de servicio de interrupción por comparación.

```
29  loop:
30              LOAD         S9,S9
31              JUMP        loop
32
33  ISR:
34              INT_CMP:
35              LOAD        S2,0F          ;VALOR PARA ENVIAR AL HABER INT
36              OUTPUT       S2,REG_OUT3   ;DEPOSITA EL VALOR EN REG_OUT3
37              LOAD        S3,02
38
39  RETARDO:
40              SUB          S3,01
41              JUMP         NZ,RETARDO
42              LOAD         S2,00
43              OUTPUT           S2,REG_OUT3       ;LIMPIA EL REGISTRO REG_OUT3
44              INPUT            S5,INT_STATE  ;LEE EL REG DE EDO DE INT
45              AND          S5,FB
46              OUTPUT    S5,INT_STATE ;LIMPIA EL REG DE EDO DE INT
47  RETURNI   ENABLE
48  ;
49  ;*****************************************************************************
50  ;Interrupt Vector
    ;*****************************************************************************
    ;
    ADDRESS 3FF
    JUMP ISR
```

Tabla 4.10 Código ensamblador para implementar una rutina de servicio de interrupción por comparación (continuación).

La figura 4.27 muestra un evento de sobreflujo dado por la terminal *timer_of*, pero no existe interrupción. Cuando el contador alcanza el valor 0Fh, que es igual al valor del registro *REG_COMP*, se envía el pulso de la terminal *INT_COMP* indicando un evento de igualdad en el proceso de comparación, generando el reconocimiento de la interrupción después de dos ciclos de reloj.

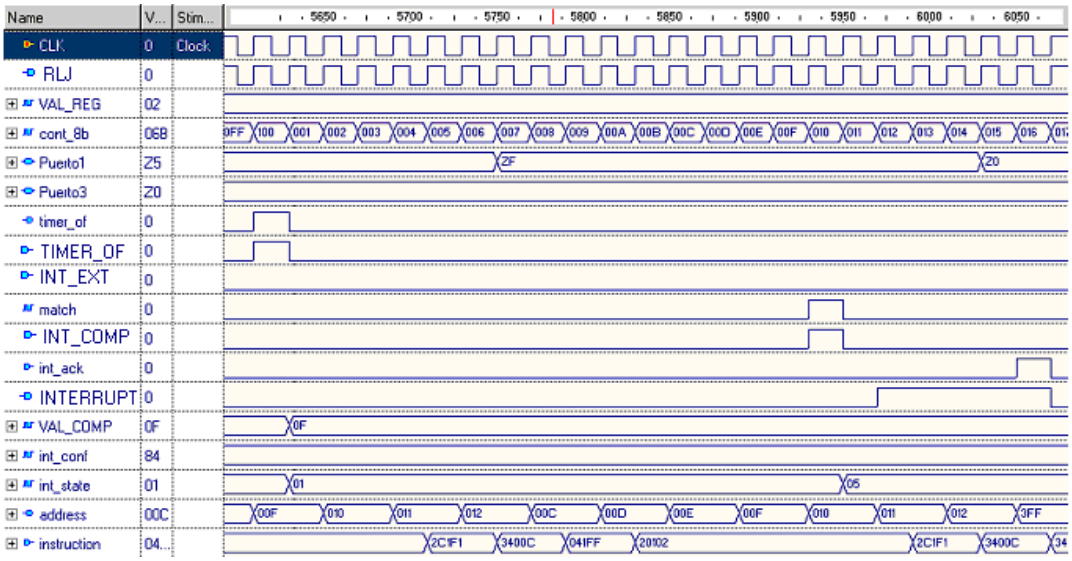


Figura 4. 27 Reconocimiento de la interrupción por comparación.

En la figura 4.28 la rutina de servicio de la Interrupción externa envía el valor 0*Fh* en las terminales de salida del puerto 3. Antes de finalizar la interrupción cambia el valor del registro de estado *INT_STATE* de 05 a 01, el 1 es debido al evento de sobreflujo.

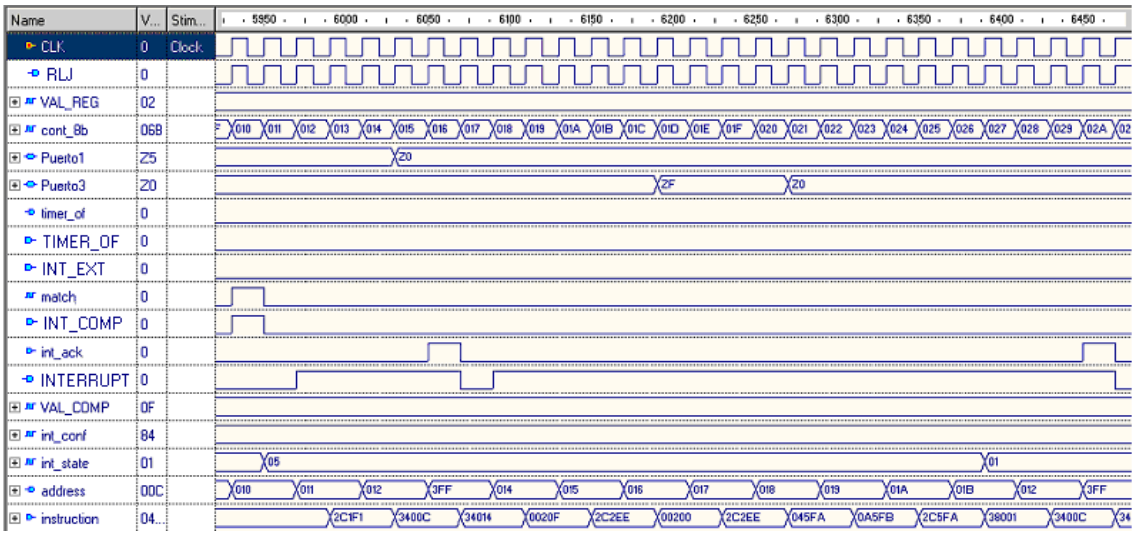


Figura 4. 28 Ejecución de la rutina de servicio de la interrupción externa.

Los ejemplos de código que se han mostrado no persiguen una aplicación específica, su objetivo es mostrar como se pueden configurar y emplear los módulos de hardware desarrollados en este trabajo de tesis.

4.2 Conclusiones

El objetivo de esta Tesis fue el diseño de módulos para el manejo de puertos, temporización e interrupciones para el Núcleo KCPSM3 y su implementación en el FPGA XC3S500 de Xilinx. El objetivo se cumplió y se comprobó que es posible diseñar un microcontrolador a partir de un procesador utilizando metodologías que enseñan en la UTM y dispositivos disponibles en la misma, permitiendo al estudiante realizar el diseño para asimilar la teoría de un microcontrolador y comprender su funcionamiento a través de las prácticas propuestas.

En el mercado es posible encontrar microcontroladores con más recursos y una capacidad de almacenamiento mayor al desarrollado en la presente Tesis, pero el propósito no fue competir en la industria o el mercado, sino que la información expuesta sea un material didáctico de apoyo tanto para el estudiante como para el personal académico, ya sea de las ingenierías de electrónica, mecatrónica o computación, específicamente en el área de Arquitectura de Computadoras.

Los FPGAs son eficaces para la implementación de algoritmos o el prototipado de circuitos y sistemas digitales, aunado a la ventaja de emplear lenguajes de alto nivel en la especificación de los diseños. Al contar con un microcontrolador empotrado, se amplía la gama de aplicaciones que con estos dispositivos pueden desarrollarse.

Respecto a los resultados obtenidos en las pruebas, es necesario mencionar que conviene configurar el preescalador antes de habilitar las funciones especiales, esto con el fin de que no interfiera el valor inicial del contador "00" con el valor predeterminado del registro de comparación VAL_COMP, que es "00", ya que esto origina que inicie la función. El registro VAL_COMP se actualiza cada que ocurre un evento de sobreflujo.

El porcentaje de error en las funciones de PWM de rampa sencilla y doble depende del redondeo que se realiza en el cálculo del valor de comparación, ya que no se utilizan valores fraccionarios. En el caso de la función CTC existe un rango de error debido a los retardos acumulados al actualizarse los registros, cada retardo equivale a 2 ciclos de la señal de salida del Preescalador. Puesto que ya se ha cuantificado, en las aplicaciones debe considerarse este error para la generación de las frecuencias requeridas por diferentes aplicaciones.

RECURSOS	KCPSM3 Y MEMORIA DE CÓDIGO	SISTEMA INTEGRADO
Número de IOBs externos	14 de 190 7.3%	34 de 190 19%
Number of RAMB16s	1 de 20 5%	1 de 20 5%
Número de Slices	91 de 4656 1.9%	329 de 4656 7%

Tabla 4. 11 Tabla comparativa de los recursos utilizados antes y después de realizar el sistema.

En la tabla 4.11 se muestran los recursos utilizados antes y después de agregar los módulos de recursos al procesador con su memoria de código. Se implementó en el dispositivo XC3S500 Spartan-3E. En ambas implementaciones se utiliza uno de los veinte bloques de memoria RAM, equivalente al 5% del total de memoria. El número de Slices se multiplica cuatro veces aproximadamente, de un 1.9 a un 7%, mientras que los bloques de entrada-salida (IOBs) externos aumentan de un 7.3 a un 19%. Estos datos muestran que el sistema implementado requiere pocos recursos del FPGA, lo que permite aún seguir expandiendo el sistema o agregar más de un microcontrolador a alguna aplicación.

Otra ventaja es la modularidad del sistema, la cual hace posible utilizar solamente los bloques que resulten útiles para alguna aplicación.

4.3 Trabajos Futuros

La electrónica es una ciencia que exige un constante aprendizaje de nuevas tecnologías o actualización de las ya aprendidas, esto es un proceso permanente. En el tiempo actual, una de las tecnologías que se encuentran en auge son los microcontroladores. En este sentido, el sistema desarrollado puede ser la base para la enseñanza de la organización y funcionamiento interno de este tipo de dispositivos.

La implementación realizada no es un proyecto cerrado, puede ser expandido agregando recursos como Comparadores y Convertidores A/D para poder trabajar con señales analógicas, Módulo de Puerto serie para poder realizar la comunicación con otro microcontrolador o con una PC. Agregar módulos que permitan realizar operaciones con punto flotante. Esto es respecto a mejoras en el diseño y quedan aún abiertas líneas de trabajo futuras, como el desarrollar aplicaciones con fines didácticos.

Apéndice A: Resumen del Repertorio de Instrucciones del Núcleo KCPSM3.

Instrucción	Descripción	Función	ZERO	CARRY
ADD sX, kk	Suma el registro sX con la constante kk.	sX ← sX + kk	?	?
ADD sX, sY	Suma el registro sX con el registro sY.	sX ← sX + sY	?	?
ADDCY sX, kk	Suma el registro sX con la constante kk con bit de acarreo.	sX ← sX + kk + CARRY	?	?
ADDCY sX, sY	Suma el registro sX con el registro sY con bit de acarreo.	sX ← sX + sY + CARRY	?	?
AND sX, kk	Operación AND lógica entre el registro sX y la constante kk.	sX ← sX AND kk	?	0
AND sX, sY	Operación AND lógica entre el registro sX y el registro sY.	sX ← sX AND sY	?	0
CALL aaa	Llamada incondicional a la subrutina en aaa.	TOS ← PC PC ← aaa	-	-
CALL C, aaa	Si la bandera CARRY es igual a '1', llama a la subrutina en aaa.	If CARRY = 1,{TOS ← PC, PC ← aaa}	-	-
CALL NC, aaa	Si la bandera CARRY es igual a '0', llama a la subrutina en aaa.	If CARRY = 0,{TOS ← PC, PC ← aaa}	-	-
CALL NZ, aaa	Si la bandera ZERO es igual a '0', llama a la subrutina en aaa.	If ZERO = 0,{TOS ← PC, PC ← aaa}	-	-
CALL Z, aaa	Si la bandera ZERO es igual a '1', llama a la subrutina en aaa.	If ZERO = 1,{TOS ← PC, PC ← aaa}	-	-
COMPARE sX,kk	Compara el registro sX con la constante kk. Pone las banderas como sea apropiado. Los registros no son afectados.	If sX=kk, ZERO ← 1 If sX<kk, CARRY ←1	?	?
COMPARE sX,sY	Compara el registro sX con el registro sY. Pone las banderas como sea apropiado. Los registros no son afectados.	If sX=sY, ZERO ← 1 If sX<sY, CARRY ←1	?	?
DISABLE INTERRUPT	Deshabilita la entrada de las interrupciones.	INTERRUPT_ENABLE←0	-	-
ENABLE INTERRUPT	Habilita la entrada de las interrupciones.	INTERRUPT_ENABLE←1	-	-
Interrupt Event	Entrada de Interrupción Asíncrona. Preserva las banderas y el PC. Limpia la bandera INERRUPT_ENABLE. Salta al vector de Interrupciones en la dirección 3FF.	ZERO ← ZERO CARRY ← CARRY INTERRUPT_ENABLE←0 TOS ← PC PC ← 3FF	-	-
FETCH sX,(sY)	Lee la memoria de datos interna apuntada por el registro sY y lo deposita en el registro sX.	sX ← RAM [(sY)]	-	-
FETCH sX, ss	Lee la localidad ss de memoria de datos interna y lo deposita en el registro sX.	sX ← RAM [ss]	-	-

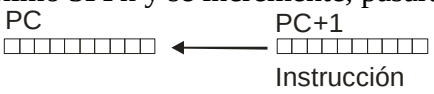
Instrucción	Descripción	Función	ZERO	CARRY
INPUT sX, (sY)	Lee el valor del puerto de entrada apuntado por el registro sY y lo deposita en sX.	PORT_ID ← sY sX←IN_PORT	-	-
INPUT sX, pp	Lee el valor del puerto de entrada apuntado por pp y lo deposita en sX.	PORT_ID ← pp sX←IN_PORT	-	-
JUMP aaa	Salto incondicional a la dirección aaa.	PC ← aaa	-	-
JUMP C, aaa	Si la bandera CARRY es igual a '1', salta a la dirección aaa.	If CARRY=1, PC ← aaa	-	-
JUMP NC, aaa	Si la bandera CARRY es igual a '0', salta a la dirección aaa.	If CARRY=0, PC ← aaa	-	-
JUMP NZ, aaa	Si la bandera ZERO es igual a '0', salta a la dirección aaa.	If ZERO=0, PC ← aaa	-	-
JUMP Z, aaa	Si la bandera ZERO es igual a '1', salta a la dirección aaa.	If ZERO=1, PC ← aaa	-	-
LOAD sX, kk	Carga el registro sX con la constante kk.	sX ← kk	-	-
LOAD sX, sY	Carga el registro sX con la constante sY.	sX ← sY	-	-
OR sX, kk	Operación lógica OR entre el registro sX y la constante kk.	sX ← sX OR kk	?	0
OR sX, sY	Operación lógica OR entre el registro sX y el registro sY.	sX ← sX OR sY	?	0
OUTPUT sX,(sY)	Escribe el registro sX en el puerto de salida apuntado por el registro sY.	PORT_ID ← sY OUT_PORT←sX	-	-
OUTPUT sX, pp	Escribe el registro sX en el puerto de salida apuntado por pp.	PORT_ID ← pp OUT_PORT←sX	-	-
RETURN	Retorno incondicional de subrutina.	PC ← TOS+1	-	-
RETURN C	Si la bandera CARRY es igual a '1', retorna de la subrutina.	If CARRY=1, PC←TOS+1	-	-
RETURN NC	Si la bandera CARRY es igual a '0', retorna de la subrutina.	If CARRY=0, PC←TOS+1	-	-
RETURN NZ	Si la bandera ZERO es igual a '0', retorna de la subrutina.	If ZERO=0, PC←TOS+1	-	-
RETURN Z	Si la bandera ZERO es igual a '1', retorna de la subrutina.	If ZERO=1, PC←TOS+1	-	-
RETURNI DISABLE	Retorna de la rutina de servicio a interrupción y deshabilita las interrupciones.	PC ← TOS ZERO ← ZERO CARRY← CARRY INTERRUPT_ENABLE←0	?	?
RETURNI ENABLE	Retorna de la rutina de servicio a interrupción y habilita las interrupciones.	PC ← TOS ZERO ← ZERO CARRY← CARRY INTERRUPT_ENABLE←1	?	?
RL sX	Rotación a la izquierda del registro sX.	sX ← {sX[6:0],s[7]} CARRY←sX[7]	?	?
RR sX	Rotación a la derecha del registro sX.	sX ← {sX[0],s[7:1]} CARRY←sX[0]	?	?
SL0 sX	Rotación a la izquierda del registro sX. Rellena con ceros.	sX←{sX[6:0],0} CARRY←sX[7]	?	?

Instrucción	Descripción	Función	ZERO	CARRY
SL1 sX	Rotación a la izquierda del registro sX. Rellena con unos.	sX←{sX[6:0],1} CARRY←sX[7]	0	?
SLA sX	Desplaza todos los bits del registro sX a la izquierda, incluyendo el acarreo.	sX ← {sX[6:0],CARRY} CARRY ← sX[7]	?	?
SR0 sX	Rotación a la derecha del registro sX. Rellena con ceros.	sX←{0, Sx[7:1]} CARRY←sX[0]	?	?
SR1 sX	Rotación a la derecha del registro sX. Rellena con unos.	sX←{1, Sx[7:1]} CARRY←sX[0]	?	?
SRA sX	Desplaza todos los bits del registro sX a la derecha, incluyendo el acarreo.	sX ← {CARRY, sX[7:1]} CARRY ← sX[0]	?	?
SRX sX	Desplazamiento Aritmético a la derecha del Registro sX con extensión de signo. El Bit sX[7] no es afectado.	sX ← {sX[7],sX[7:1]} CARRY ← sX[0]	?	?
STORE sX, (sY)	Escribe el registro sX en la memoria de datos interna en la localidad apuntada por sY.	RAM[(sY)] ← sX	-	-
STORE sX, ss	Escribe el registro sX en la memoria de datos interna en la localidad apuntado por ss.	RAM[(ss)]← sX	-	-
SUB sX, kk	Substrae el valor kk del registro sX.	sX ←sX - kk	?	?
SUB sX, sY	Substrae el valor sY del registro sX.	sX ←sX – sY	?	?
SUBCY sX, kk	Substrae el valor kk del registro sX con acarreo.	sX ←sX – kk – CARRY	?	?
SUBCY Sx, sY	Substrae el valor sY del registro sX con acarreo.	sX ←sX – sY - CARRY	?	?
TEST sX, kk	Prueba los bits del registro sX con la literal kk. Actualiza las banderas CARRY y ZERO. Los registros no son afectados.	If (sX AND kk)=0, ZERO←1 CARRY ←paridad de (sX AND kk)	?	?
TEST sX, sY	Prueba los bits del registro sX con el registro sY. Actualiza las banderas CARRY y ZERO. Los registros no son afectados.	If (sX AND sY)=0, ZERO←1 CARRY ←paridad de (sX AND sY)	?	?
XOR sX, kk	Operación lógica XOR entre sX y kk	sX ← sX XOR kk	?	0
XOR sX, sY	Operación lógica XOR entre sX y sY	sX ← sX XOR sY	?	0

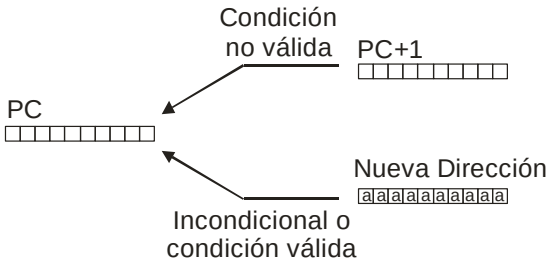
Apéndice B: Repertorio de Instrucciones del Núcleo KCPSM3

JUMP

Bajo condiciones normales, el contador del programa (PC, *Program Counter*) incrementa el apuntador a la siguiente instrucción. El espacio disponible es de 1024 localidades (000h a 3FFh) además de que el contador del programa tiene un ancho de 10 bits. Cuando el contador alcance el valor máximo 3FFh y se incremente, pasará al valor 000h.



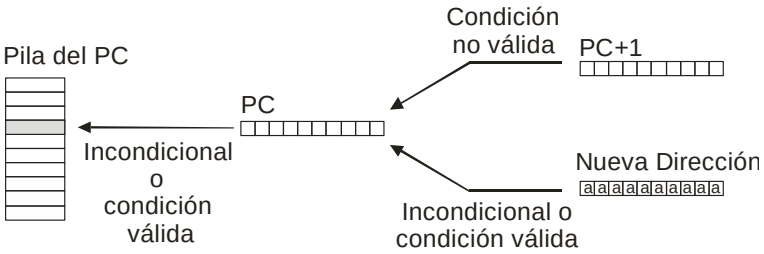
La instrucción JUMP puede ser utilizada para modificar esta secuencia especificando una nueva dirección. La instrucción JUMP puede ser condicional, un salto condicional será ejecutado dependiendo de evaluar la bandera ZERO y de CARRY. La instrucción JUMP no tiene efecto en el estado de las banderas.



Cada instrucción JUMP debe especificar una dirección de 3 dígitos hexadecimales. El ensamblador soporta etiquetas para simplificar este proceso.

CALL

La instrucción CALL es similar en funcionamiento a la instrucción JUMP, en el sentido de que modifica la ejecución normal de la secuencia del programa especificando una nueva dirección. La instrucción CALL también puede ser condicional. Además de suministrar una nueva dirección, la instrucción CALL también provoca que el valor del contador del programa se almacene dentro de la pila del contador de programa. La instrucción CALL no tiene efecto en los estados de las banderas.

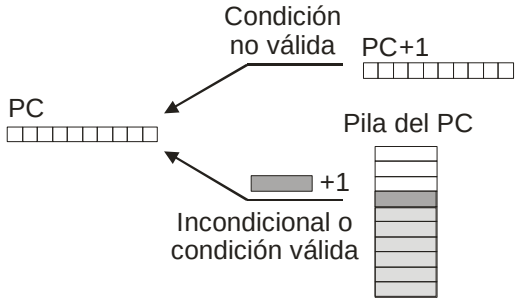


La pila del contador de programa soporta una profundidad de 31 valores de direcciones. Esto da la posibilidad de ejecutar secuencias de llamado a funciones a través de CALL con una profundidad de 31 niveles. Cuando se realiza una interrupción, también se reserva un espacio en la pila del contador de programa.

La pila es implementada como un buffer cíclico separado. Cuando la pila se llena, simplemente se sobrescribe el valor menos reciente. Por ello no es necesario inicializar al apuntador de la pila. Esto explica por qué ninguna otra localidad de memoria necesita ser reservada para el funcionamiento de la pila. Cada instrucción CALL debe especificarse como un valor hexadecimal de 3 dígitos.

RETURN

La instrucción RETURN es el complemento a la instrucción CALL. RETURN puede ser también una instrucción condicional. En el caso de un RETURN incondicional o con condición válida, el nuevo valor del PC será formado internamente incrementando el último valor de la última dirección que se introdujo a la pila. Esto asegura que el programa ejecute la instrucción que sigue después de CALL. La instrucción RETURN no tiene efecto en el estatus de las banderas.

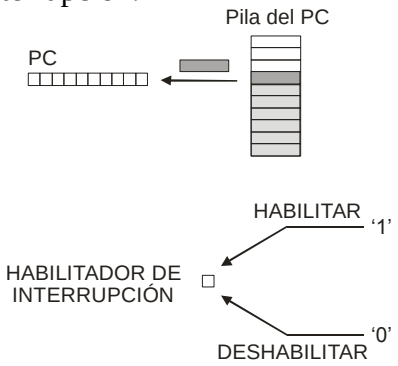


Es responsabilidad del programador asegurar que un RETURN sea ejecutado en respuesta a una previa instrucción CALL tal que el contador del programa contenga una dirección válida. En caso contrario, la implementación cíclica de la pila continuará proveyendo valores erróneos para el RETURN.

RETURNI

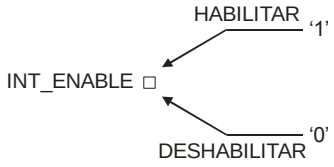
La instrucción RETURNI es una variación especial de la instrucción RETURN la cual debe ser usada para concluir una Rutina de Servicio de Interrupción. RETURNI es incondicional y además cargará el PC con la última dirección que se introdujo a la pila del contador del

programa. La instrucción RETURNI reestablece las banderas a la condición donde ellas se encontraban en el punto de la interrupción. RETURNI también determina la habilitación o deshabilitación de interrupciones usando ENABLE o DISABLE respectivamente, como un operando. Es responsabilidad del programador asegurar que RETURNI sea ejecutado solamente en repuesta a una interrupción.



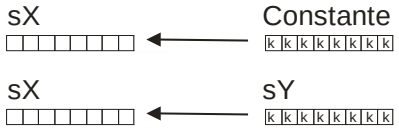
ENABLE/DISABLE INTERRUPT

Estas instrucciones son usadas para habilitar o deshabilitar la bandera INT_ENABLE. Antes de usar ENABLE INTERRUPT, una rutina de interrupción debe ser asociada con la dirección del vector de interrupciones (localizado en la dirección 3FF). Las Interrupciones nunca deben ser habilitadas mientras se ejecuta una rutina de servicio de interrupción.
Las interrupciones son enmascaradas cuando la bandera INT_ENABLE está en bajo. Este es el estado predeterminado de la bandera después de reprogramar o reiniciar el KCPSM3. INT_ENABLE es deshabilitado durante una interrupción activa.



LOAD

La instrucción LOAD, provee de un método para especificar el contenido de cualquier registro. El nuevo valor puede ser una constante, o el contenido de cualquier otro registro. La instrucción LOAD no tiene efecto en los estados de las banderas.

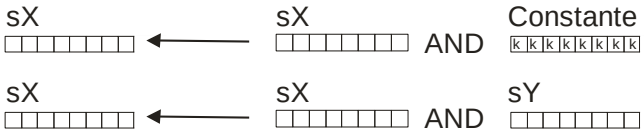


Como la instrucción LOAD no tiene efecto sobre las banderas puede ser usado para reordenar o asignar contenidos de registros en cualquier etapa del programa de ejecución. La habilidad para asignar una constante sin que tenga impacto en el tamaño del programa o la ejecución significa que la instrucción LOAD es la manera mas obvia para asignar un valor o limpiar un registro.

El primer operando de la instrucción LOAD debe especificar el registro a cargar como una ‘s’ seguida de un digito hexadecimal. El segundo operando debe especificarse con un segundo registro o con un valor usando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar el proceso.

AND

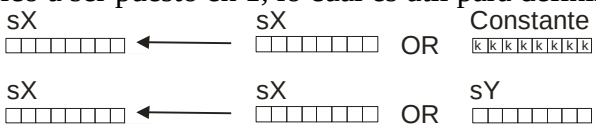
La instrucción AND ejecuta la operación lógica AND entre dos operandos, el primer operando es cualquier registro y en este registro se almacena el resultado de la operación. Un segundo operando puede ser cualquier registro o un valor constante de ocho bits. Las banderas serán afectadas por esa operación. La operación AND es útil para limpiar bits de un registro y evaluar sus contenidos. Entonces, el estado de la bandera ZERO controlará el flujo del programa.



Cada instrucción AND debe ser especificada con un registro como primer operando designado por ‘s’ seguido por un dígito hexadecimal. El segundo operando debe ser especificado por un segundo registro o una constante utilizando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

OR

La instrucción OR ejecuta la operación lógica OR entre dos operandos, el primer operando es cualquier registro y en este registro se almacena el resultado de la operación. Un segundo operando puede ser cualquier registro o un valor constante de ocho bits. Las banderas serán afectadas por esa operación. La operación OR provee una manera de forzar cualquier bit de un registro específico a ser puesto en 1, lo cual es útil para definir señales de control.

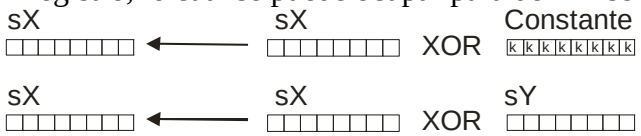


Cada instrucción OR debe ser especificada con un registro como primer operando designado por ‘s’ seguido por un dígito hexadecimal. El segundo operando debe ser especificado por un segundo registro o una constante utilizando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

XOR

La instrucción XOR ejecuta la operación lógica XOR entre dos operandos, el primer operando es cualquier registro y en este registro se almacena el resultado de la operación. Un segundo operando puede ser cualquier registro o un valor constante de ocho bits. Las

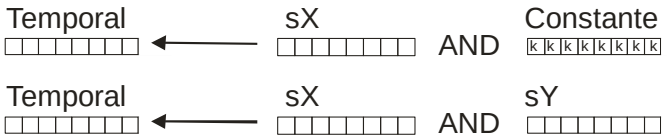
banderas serán afectadas por esa operación. La operación XOR es útil para invertir los bits contenidos en un registro, lo cual se puede ocupar para definir señales de control.



Cada instrucción XOR debe ser especificada con un registro como primer operando designado por ‘s’ seguido por un dígito hexadecimal. El segundo operando debe ser especificado por un segundo registro o una constante utilizando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

TEST

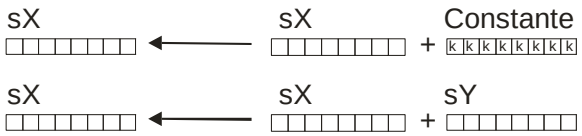
La instrucción TEST ejecuta la operación lógica AND entre dos operandos, a diferencia de la instrucción AND, el resultado es descartado y sólo las banderas son afectadas. La bandera ZERO es puesto en 1 si todos los bits del resultado temporal están en bajo. La bandera de CARRY es usada para indicar paridad del resultado temporal. La comprobación de paridad típicamente hace un TEST sobre todos los bits.



Aplicar el TEST es típicamente usado para aislar y evaluar un bit. Cada instrucción TEST debe ser especificada con un registro como primer operando designado por ‘s’ seguido por un dígito hexadecimal. El segundo operando debe ser especificado por un segundo registro o una constante utilizando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

ADD

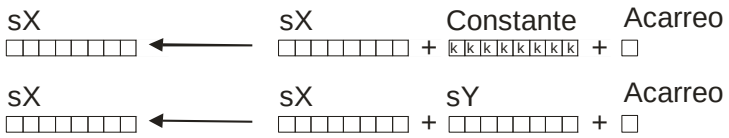
La instrucción ADD ejecuta la adición de dos operandos de ocho bits. El primer operando es cualquier registro, y en este registro donde será almacenado el resultado de la operación. El segundo operando puede ser cualquier registro o un valor constante de ocho bits. Las banderas serán afectadas por esta operación. Note que esta instrucción no ocupa el acarreo como una entrada, por ello no es necesario condicionar las banderas antes de usarla. La habilidad para especificar cualquier constante es útil para formar secuencias de control y contadores.



Cada instrucción ADD debe ser especificada con un registro como primer operando designado por ‘s’ seguido por un dígito hexadecimal. El segundo operando debe ser especificado por un segundo registro o una constante utilizando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

ADDCY

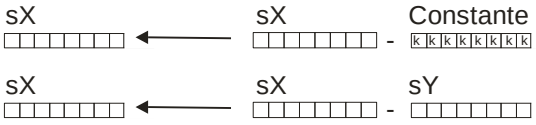
La instrucción ADDCY ejecuta una adición de dos operandos de ocho bits, junto con el contenido de la bandera de CARRY. El primer operando es cualquier registro, y en este registro será almacenado el resultado de la operación. El segundo operando puede ser cualquier registro o un valor constante de ocho bits. Las banderas serán afectadas por esta operación. La operación ADDCY puede ser usado para la formación de sumadores y contadores que excedan ocho bits.



Cada instrucción ADDCY debe ser especificada con un registro como primer operando designado por ‘s’ seguido por un dígito hexadecimal. El segundo operando debe ser especificado por un segundo registro o una constante utilizando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

SUB

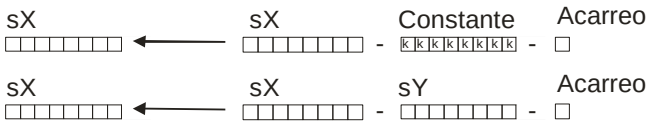
La instrucción SUB ejecuta una sustracción de dos operandos de ocho bits. El primer operando es cualquier registro, y en este registro será almacenado el resultado de la operación. El segundo operando puede ser cualquier registro o un valor constante de ocho bits. Las banderas serán afectadas por esta operación. Note que esta instrucción no ocupa el acarreo como una entrada, por ello no es necesario condicionar las banderas antes de usarla. La bandera de CARRY indica cuando ha ocurrido un sobreflujo.



Cada instrucción SUB debe ser especificada con un registro como primer operando designado por ‘s’ seguido por un dígito hexadecimal. El segundo operando debe ser especificado por un segundo registro o una constante utilizando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

SUBCY

La instrucción SUBCY ejecuta una sustracción de dos operandos de ocho bits, junto con el contenido de la bandera CARRY. El primer operando es cualquier registro, y en este registro será almacenado el resultado de la operación. El segundo operando puede ser cualquier registro o un valor constante de ocho bits. Las banderas serán afectadas por esta operación. La operación SUBCY puede ser usado para contadores descendentes que no excedan ocho bits.

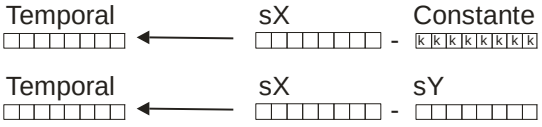


Cada instrucción SUBCY debe ser especificada con un registro como primer operando designado por 's' seguido por un dígito hexadecimal. El segundo operando debe ser especificado por un segundo registro o una constante utilizando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

COMPARE

La instrucción COMPARE ejecuta una substracción de dos operandos de ocho bits. A diferencia de la instrucción SUB el resultado de la operación es descartado y solo las banderas son afectadas. La bandera ZERO es puesto en uno 1 cuando todos los bits del resultado temporal están en bajo e indica que ambos operandos eran idénticos.

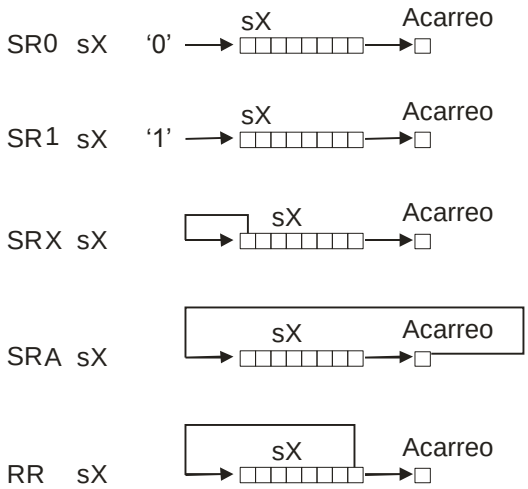
La bandera de CARRY indica que ha ocurrido un desbordamiento y que el segundo operando fue mayor que el primero



Cada instrucción COMPARE debe ser especificada con un registro como primer operando designado por 's' seguido por un dígito hexadecimal. El segundo operando debe ser especificado por un segundo registro o una constante utilizando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

SR0, SR1, SRX, SRA, RR

Todo el grupo de rotación y desplazamiento a la derecha modifica el contenido de un registro. Todas las instrucciones en el grupo tienen efecto en la bandera de acarreo.



SR0 sX : Hace un desplazamiento de los bits hacia la derecha, con acarreo. El espacio vacío lo rellena con un cero.

SR1 sX: Hace un desplazamiento de los bits hacia la derecha, con acarreo. El espacio vacío lo rellena con un uno.

SRX sX: Hace un desplazamiento de los bits hacia la derecha, con acarreo. El espacio vacío lo rellena con el bit más significativo.

SRA sX: Hace un desplazamiento de los bits hacia la derecha, con acarreo. El espacio vacío lo rellena con el valor que se encuentre en la bandera de CARRY.

RR sX: Hace una rotación hacia la derecha con acarreo.

Cada instrucción debe ser especificada con un registro como primer operando designado por 's' seguido por un dígito hexadecimal. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

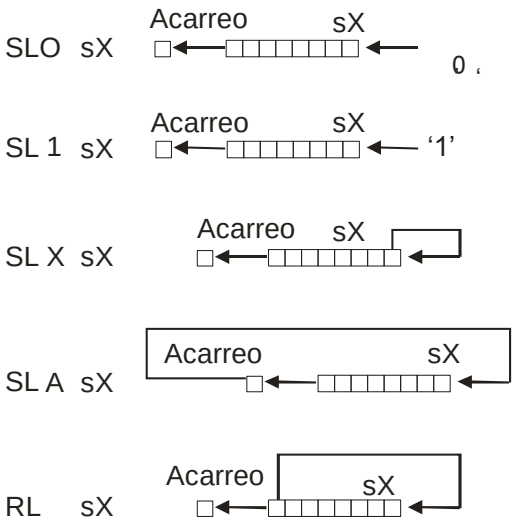
SL0, SL1, SLX, SLA, RL

Todo el grupo de rotación y desplazamiento a la izquierda modifica el contenido de un registro. Todas las instrucciones en el grupo tienen efecto en la bandera de acarreo.

SL0 sX : Hace un desplazamiento de los bits hacia la izquierda, con acarreo. El espacio vacío lo rellena con un cero.

SL1 sX: Hace un desplazamiento de los bits hacia la izquierda, con acarreo. El espacio vacío lo rellena con un uno.

SLX sX: Hace un desplazamiento de los bits hacia la izquierda, con acarreo. El espacio vacío lo rellena con el bit más significativo.



SLA sX: Hace un desplazamiento de los bits hacia la izquierda, con acarreo. El espacio vacío lo rellena con el valor que se encuentre en la bandera de CARRY.

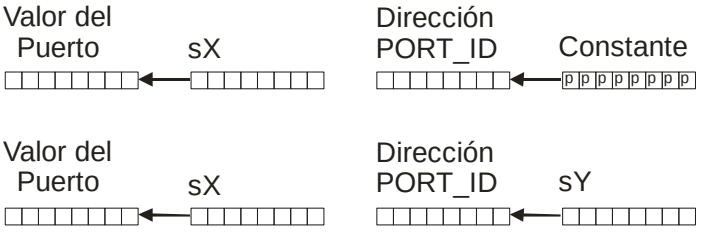
RL sX: Hace una rotación hacia la izquierda con acarreo.

Cada instrucción debe ser especificada con un registro como primer operando designado por 's' seguido por un dígito hexadecimal. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

OUTPUT

La instrucción OUTPUT transfiere el contenido de cualquier registro a la lógica externa del KCPSM3. La dirección del puerto (en el rango de 00 a FF) puede ser definida por un valor constante o indirectamente como el contenido de cualquier otro registro.

Las banderas no son afectadas por esta operación.



Una interfaz lógica de usuario es requerida para decodificar el valor de la dirección del puerto PORT_ID y capturar el dato proveído en OUT_PORT. El pin de WRITE_STROBE es puesto en 1 durante una operación de salida, y debe ser sincronizado con la señal de reloj que recibe el núcleo.

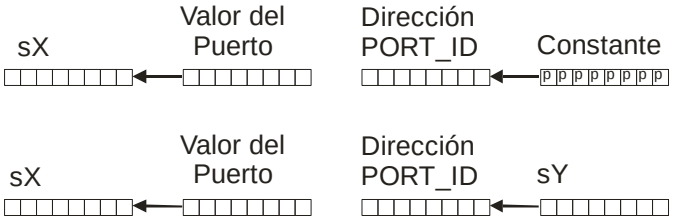
Cada instrucción OUT_PORT debe especificar el registro fuente como una 's' seguido de un digito hexadecimal. Y debe especificarse la dirección del puerto de salida usando el

valor de un registro de manera similar o especificando un identificador de puerto con una constante de ocho bits usando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

INPUT

La instrucción INPUT recibe un dato externo para ser ubicado dentro de cualquiera de los registros internos. La dirección del puerto (en el rango de 00 a FF) puede ser definido por un valor constante o indirectamente como el contenido de cualquier otro registro.

Las banderas no son afectadas por esta operación.



Una interfaz lógica de usuario es requerida para decodificar el valor de la dirección del puerto PORT_ID y suministrar el dato correcto a IN_PORT. El pin de READ_STROBE es puesto en 1 durante una operación de entrada, pero no siempre es necesario para la interfaz lógica decodificar, esta señal.

Cada instrucción IN_PORT debe especificar el registro fuente como una ‘s’ seguido de un digito hexadecimal. Y debe especificarse la dirección del puerto de entrada usando el valor de un registro de manera similar o especificando un identificador de puerto con una constante de ocho bits usando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

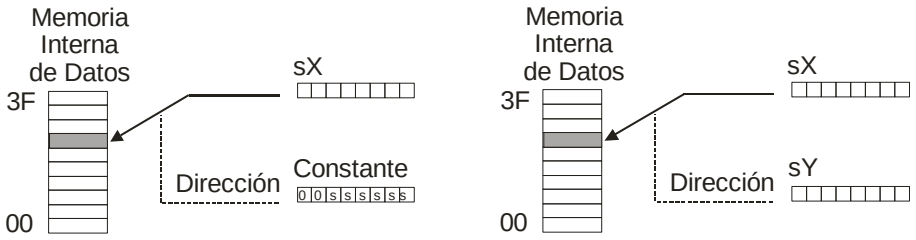
STORE

La instrucción STORE habilita que el contenido de cualquier registro sea transferido a una localidad de la memoria interna de 64 bytes. La dirección de almacenamiento (en el rango de 00 a 3F) puede ser definida por un valor constante o indirectamente como el contenido de cualquier otro registro. Las banderas no son afectadas por esta operación.

Cada instrucción STORE debe especificar el registro fuente como una ‘s’ seguido por un digito hexadecimal.

Debe especificarse la dirección de almacenamiento usando un registro de una manera similar o especificando una constante de seis bits usando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

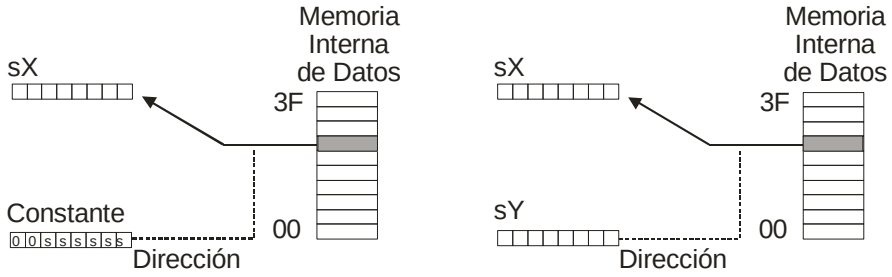
Aunque el ensamblador rechazará todas las constantes mayores que 3F, es responsabilidad del programador asegurarse que el valor del registro ’sY’ esté dentro de este rango de direcciones



FETCH

La instrucción FETCH habilita que el dato contenido en una localidad de la memoria interna de 64 bytes sea transferido a cualquiera de los registros. La dirección de almacenamiento (en el rango de 00 a 3F) puede ser definida por un valor constante o indirectamente como el contenido de cualquier otro registro.

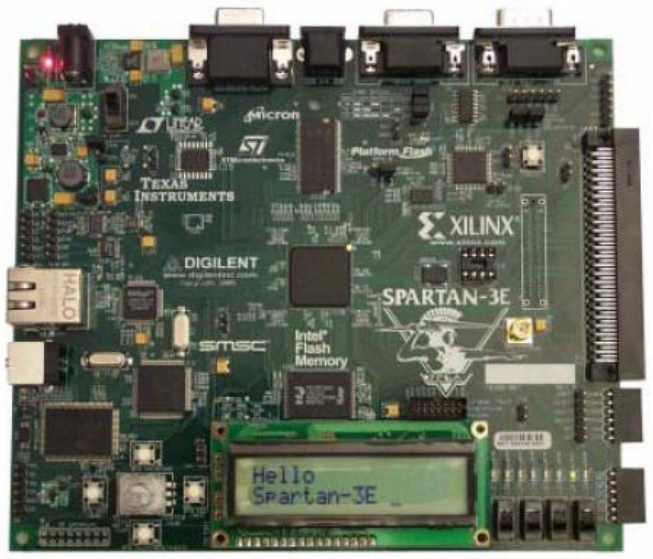
Las banderas no son afectadas por esta operación.



Cada instrucción FETCH debe especificar el registro fuente como una ‘s’ seguido por un digito hexadecimal. Y debe especificarse la dirección de almacenamiento usando un registro de una manera similar o especificando una constante de seis bits usando dos dígitos hexadecimales. El ensamblador soporta nombres de registros y etiquetas de constantes para simplificar los procesos.

Aunque el ensamblador rechazará todas las constantes mayores que 3F, es responsabilidad del programador asegurarse que el valor del registro ’sY’ esté dentro de este rango de direcciones

Apéndice C: Características de la tarjeta de desarrollo *Spartan 3E Starter Board.*



Las características principales de la *Spartan 3E Starter Board* son:

- Incluye un FPGA XC3S500E, el cual tiene aproximadamente 10,000 celdas lógicas y 232 pines de entrada-salida.
- 64 Mb de memoria DDR SDRAN
- Pantalla LCD de 2 líneas de 16 caracteres.
- Tres conectores de expansión Digilent de 6 pines.
- Dos entradas SPI con convertidor analógico a digital con pre-amplificador de ganancia programable, además cuatro salidas SPI con convertidor digital a analógico.
- Incluye codificador rotatorio, 4 Switches deslizables, 4 Switches Push-Button y 8 LEDS discretos.
- Oscilador de 50MHz de frecuencia y contiene 2 entradas para relojes externos: Entrada SMA y Socket de 8-pines.
- Incluye los siguientes puertos:
  - PS/2
  - VGA
  - 10/100 Ethernet PHY
  - RS-232 (Conectores Macho y Hembra)
  - USB

Referencias

[1] CHAPMAN, Ken: “PicoBlaze KCPSM3 8 bit Micro Controler”; Xilinx, Inc; June, 2005.

[2] CHAPMAN, Ken: “PicoBlaze 8-bit Embedded Microcontroller User Guide”; Xilinx, Inc; November, 2005.

[3] ANGULO USATEGUI, José María; ANGULO MARTÍNEZ, Ignacio: “Microcontroladores PIC Diseño práctico de aplicaciones”; McGraw-Hill.

[4] MANDADO, Enrique: “Dispositivos Lógicos Programables”; Editorial THOMSON

[5] TEPOZÁN RÍOS, Juan Carlos: “Diseño de controladores para la tarjeta de desarrollo XSA-100”; Tesis para obtener el título de Ingeniero en Electrónica; Universidad Tecnológica de la Mixteca; Mayo 2004.

[6] HUERTA, Pablo : “Sistemas MPSoC en FPGA’s”; Departamento de Arquitectura y Tecnología de Computadoras; Universidad Rey Juan Carlos; España.

[7] “MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 9.2i”; Xilinx, Inc; 2007.

[8] WANG, George: “Connect LCD to MicroBlaze Using an OPB Customer Peripheral”; Xilinx, Inc.

[9] GONZÁLEZ GÓMEZ: “Locomoción de un Robot Ápodo Modular con el Procesador MicroBlaze”; Escuela Politécnica Superior, Universidad Autónoma de Madrid, España.

[10] CHAPMAN, Ken: “UART Real Time Clock”; KCPSM3 Reference Design; Xilinx, Inc; September, 2003.

[11] CHAPMAN, Ken: “Amplifier and A/D Converter Control for Spartan-3E Starter Kit”; Xilinx, Inc; February, 2006.

[12] CHAPMAN, Ken: “D/A Converter Control for Spartan-3E Starter Kit”; Xilinx, Inc; February, 2006.

[13] CHAPMAN, Ken: “Frequency Counter for Spartan-3E Starter Kit (with test oscillators)”; Xilinx, Inc; March, 2006.

[14] CHAPMAN, Ken: “Initial Design for Spartan-3E Starter Kit (LCD Display Control)”; Xilinx, Inc; March, 2006.

[15] CHAPMAN, Ken: “Software Implementation of Pulse Width Modulation (PWM)”; Xilinx, Inc; May, 2006.

[16] BRONW, Stephen; VRANESIC, Zvonco: “Fundamentos de Lógica Digital con Diseño VHDL”; Segunda Edición; McGraw-Hill.

[17] GEIBLER, Richard; BULACH, Slavek: “VHDL manual”; September, 1998.

[18] DOUGLAS L., Perry: “Programing VHDL by Example”; Fourth Edition; McGraw-Hill.

[19] ROMERO, René: “Sistemas Digitales con VHDL”; LEGARIA EDICIONES

[20] ARIAS ESTRADA, Miguel; CUMPLIDO PARRA René “Curso Intensivo de FPGA’s y VHDL”; INAOE; Marzo del 2004.

[21] MACKENZIE, I. Scout: “The 8051 Microcontroller”; Edit. Thomson Delmar Learning; Second Edition.

[22] ATMEL, AVR: “ATmega8 DataSheet”, 2002.

Referencias URL

[URL1] "OpenDSP"; Departamento de Tecnología Electrónica de la Universidad de Valladolid, España; <http://www.dte.eis.uva.es/OpenProjects/OpenDSP/indice.htm>; [Consulta: 28 de mayo del 2008].

[URL2] "Sistema empotrado basado en codiseño para el procesamiento de imágenes"; Departamento de Arquitectura y Tecnologia de Computadores; Universidad de Sevilla, España; <http://icaro.eii.us.es/descargas/Practica4.pdf>; [Consulta: 28 de mayo del 2008].

[URL3] "Tarjeta Procesadora Blackfin"; HVSistemas; España; <http://www.hvsistemas.es/es/prod/H8606.html>; [Consulta: 28 de mayo del 2008].

[URL4] GÓMEZ PULIDO,Juan Antonio; "Práctica de diseño y prototipado sobre circuito fpga, Contador modelado mediante el procesador Picoblaze"; Universidad de Extremadura, España; < http://arco.unex.es/ ise6_xsv/contador_pico.htm>; [Consulta: 29 de mayo del 2008].

[URL5] "Sistemas embebidos. Implementación en un FPGA del microcontrolador PicoBlaze."; Universidad Autónoma de Madrid, España; <http://arantxa.ii.uam.es/~edcd/lab/Practica2.pdf> [Consulta: 29 de mayo del 2008].

[URL6] "Sistema de Control y Adquisición NI CompactRIO"; National Instruments, Inc; <http://www.ni.com/compactrio/esa/whatis.htm>; [Consulta: 30 de mayo del 2008].

[URL7] FERGUSON, Tim; "Scots build green supercomputer"; London, England; <http://news.cnet.com/Scots-build-green-supercomputer/2100-1010_3-6169254.html>; [Consulta: 30 de mayo del 2008].

[URL8] JAMES, Lucero; "Reference System: PowerPC 440 System Simulation"; Xilinx, Inc; April,2008; <http://www.xilinx.com/support/documentation/application_notes/xapp1003.pdf>; [Consulta: 30 de mayo del 2008].

[URL9] SCHOEBERL, Martin; "Java Optimized Processor" February ,2008; <http://www.jopdesign.com/index.jsp>; [Consulta: 30 de mayo del 2008].