UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN TEMPORIZADOR PARA UN SISTEMA EN CHIP (SOC) EN LENGUAJE VERILOG

TESIS

PRESENTADA POR:

JOSE LUIS CUEVA MAMANI

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PUNO – PERÚ

2018

UNIVERSIDAD NACIONAL DEL ALTIPLANO - PUNO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN TEMPORIZADOR PARA UN SISTEMA EN CHIP (SOC) EN LENGUAJE VERILOG

TESIS PRESENTADA POR:

JOSE LUIS CUEVA MAMANI

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

FECHA DE SUSTENTACIÓN: 28-12-2018

APROBADO POR EL JURADO CONFORMADO POR:

PRESIDENTE: _____

Dr. IVAN DELGADO HUAYTA

PRIMER MIEMBRO: _____

M.Sc. GAVINO JOSÉ FLORES CHIPANA

SEGUNDO MIEMBRO: _____

Dr. MAXIMO AMANCIO MONTALVO ATCO

DIRECTOR / ASESOR: _____

Mg. LUIS ENRIQUE BACA WIESSE

Área : Microelectrónica

Tema : Diseño de temporizador

DEDICATORIA

Este trabajo va dedicado a mis padres Jesús y Gregoria.

A mis amigos y a mi familia por todo su apoyo y confianza.

.

AGRADECIMIENTOS

A los docentes universitarios de la escuela profesional de Ingeniería electrónica.

Al programa de capacitación en diseño circuitos integrados CI BRASIL.

A NXP semiconductores Brasil.

.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE ACRÓNIMOS

HDL	Hardware Description Language
FPGA	Field Programable Gate Array
RTL	Register Transfer Level
CMOS	Complementary Metal Oxid Semiconductor
PWM	Pulse-width modulation

ÍNDICE DE ANEXOS

RESUMEN

La temporización electrónica digital ha sido usada para tareas de control de tiempo desde la invención de los circuitos integrados, existen temporizadores individuales encapsulados en un único chip, otros son incluidos dentro de sistemas mayores, el estudio de esta investigación es diseñar un módulo temporizador para formar parte de un sistema en chip (SOC). Estos sistemas en chip son muy comunes en el mercado de la informática móvil, control de vehículos e internet de las cosas (IoT), debido a su bajo costo, función específica y en general bajo consumo de energía, construidos alrededor de un microprocesador junto a otros periféricos. En este trabajo se hace el diseño del temporizador digital para generar y tratar diferentes eventos relacionados con el tiempo que incluye también generación de señales PWM (modulación por amplitud de pulso), las aplicaciones de este dispositivo son: generación de señales de tiempo (Triggers) para inicio de conversiones analógico/digitales en un ADC, circuito de protección Watch Dog (perro guardián), inicio de secuencias de transmisión de datos UART (Comunicación serial), conteo de eventos en los puertos de entrada de un SoC, temporización para despertador de modo de bajo consumo en un microcontrolador, generación de señal PWM, este temporizador incluye una hoja de características como fuente de datos para que pueda ser usado e integrado correctamente dentro de un sistema en chip (SOC) junto con otros componentes. Este documento contiene las características de funcionamiento del temporizador descritos en capítulos posteriores, Se cuenta también con las pruebas respectivas con el software NCSIM de Cadence y a nivel de hardware en un FPGA DE2-115 Altera.

Palabras Clave: Microelectrónica, Verilog, Temporizador, Circuitos Integrados, Sistema en chip, PWM, SOC, FPGA.

ABSTRACT

Digital electronic timers have been used for time control tasks since the invention of integrated circuits, there are individual timers encapsulated in a single chip, others are included within other systems, the study of this research is to design a timer module to form part of a system on chip (SOC). These on-chip systems are very common in the market of mobile computing, vehicle control and internet of things (IoT), due to their low cost, specific function and generally low power consumption, built around a microprocessor together with other peripherals. In this work the design of the digital timer is made to generate and treat different events related to the time that also includes generation of PWM signals (modulation by pulse amplitude), the applications of this device are: generation of time signals (Triggers) for initiation of analog / digital conversions in an ADC, Watch Dog protection circuit (watchdog), start of UART data transmission sequences (serial communication), event counting in the input ports of a SoC, timing for wakeup from low consumption mode in a microcontroller, generation of PWM signal, this timer has a data sheet for data reference so it can be used to integrate this module correctly within a system on chip (SOC) along with other components. This document contains the operating characteristics of the timer described in later chapters. It also has the respective tests with the Cadence NCSIM software and hardware level in an Altera FPGA DE2-115.

Keywords: Microelectronics, Verilog, Timer, Integrated Circuits, System on Chip, PWM, SOC, FPGA.

CAPITULO I

1.1. INTRODUCCIÓN

Para establecer la motivación de esta tesis, es necesario analizar el estado de la tecnología microelectrónica actual y de los microsistemas en particular. La tecnología microelectrónica está entrando a formar cada vez mas parte de la vida cotidiana. Así además del ordenador PC, aparatos convencionales tanto en los vehículos, como en las viviendas es difícil encontrar equipos o subsistemas eléctricos que no incluyan algún tipo de circuito integrado para su control. Aspectos como la automatización de viviendas y edificios se empiezan a considerar una cuestión clave para el futuro de la industria informática. La respuesta a esta necesidad la han dado los microsistemas, que permiten incluir en una oblea de silicio sensores, actuadores y la electrónica de control necesaria dando lugar a los llamados sensores y actuadores inteligentes. Solo es necesario estudiar la evolución del mercado de los microsistemas, para ver, que es una industria con un fuerte crecimiento. Por lo que en esta investigación (el diseño de un módulo temporizador) se aborda la necesidad de diseñar componentes para sistemas electrónicos propios para tener una menor dependencia en relación la importación de los mismos a la vez de proporcionar una base para futuros diseños incluyendo proyectos que no involucren exclusivamente temporizadores, las aplicaciones son múltiples considerando el control de tiempo; este proyecto se desarrollará de forma que sus propiedades y características podrán ser puestas a prueba.

En el capítulo I se presenta el planteamiento del problema del diseño de componentes digitales para circuitos integrados, la motivación del proyecto y los objetivos de la presente investigación.

En el capítulo II se definen conceptos relacionados al proyecto tales como el área de actuación, microelectrónica, los transistores MOSFET y términos utilizados para medir y probar el funcionamiento del sistema a implementarse.

En el capítulo III se describe la proyección e implementación del proyecto, tenemos el diagrama de bloques, el diagrama de señales, el diagrama de flujo sobre el funcionamiento del sistema y la descripción de las características del temporizador.

En el capítulo IV Se muestran los resultados gráficos de formas de onda sobre el funcionamiento del sistema en base a simulaciones realizadas con el presente sistema, incluye también una sección de discusión sobre las diferencias y similitudes de este proyecto con los proyectos mencionados en antecedentes.

1.2. PLANTEAMIENTO DEL PROBLEMA

Los sistemas microelectrónicos actuales crecen en complejidad y sofisticación para cumplir diversos requerimientos como son entre otros la potencia de procesamiento y el menor consumo de energía, la participación directa en el planeamiento y diseño de componentes específicos nos proporciona un medio de proyectar tecnología independiente de componentes importados y la vez crear sistemas que se adapten al uso común y cotidiano de nuestro entorno local.

1.2.1. DEFINICIÓN DEL PROBLEMA PRINCIPAL

La planificación y el conocimiento sobre el diseño de sistemas electrónicos ajustados a tareas específicas, por ejemplo, componentes digitales adaptados para operaciones específicas, como poco consumo de energía, escalabilidad y facilidad de uso para el usuario final es una motivación para esta tesis.

El diseño que centrara en elaborar un componente básico para tareas de temporización.

Un circuito electrónico descrito en lenguaje HDL que cumple requisitos específicos es una necesidad para el diseñador de componentes digitales, el planteamiento de este proyecto es implementar un diseño digital en lenguaje HDL cumpliendo con las características definidas a través de la hoja de especificaciones.

La hoja de datos y características es entregada al usuario final para ser una fuente de información para que el componente pueda ser integrado dentro de sistemas mayores, en este caso sistemas en chip que incluyen procesadores y periféricos para diversas tareas.

La simulación de este diseño es necesaria para comprobar su validez, en el presente proyecto se aborda la simulación en software de pago licenciado, así como software libre, se incluye datos de la puesta en prueba del diseño en una matriz de puertas programables FPGA.

1.2.2. FORMULACIÓN DEL PROBLEMA PRINCIPAL

Se puede diseñar un sistema digital propio, ¿bajo propias características para uso general dentro de dispositivos más complejos como sistemas en chip?, se puede implementar y probar a nivel lógico el funcionamiento de este dispositivo para así validar las características de este?, ¿se puede redactar una hoja de datos para que pueda ser

compartido y usado por cualquier persona o proyecto que así lo requiera para fines de estudio o aplicación?

1.2.3. PROBLEMAS ESPECÍFICOS

- Diseño del diagrama de bloques de un componente electrónico digital.
- Implementación y pruebas de un componente electrónico digital.
- Implementación de una hoja de características (Datasheet) del componente electrónico digital.
- Simulación de un componente electrónico digital en un sistema FPGA.

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Diseñar e implementar un temporizador en lenguaje Verilog para ser utilizado en un sistema en chip (SOC).

1.3.2. OBJETIVOS ESPECÍFICOS

- Diseñar un diagrama de bloques con las funcionalidades del temporizador.
- Implementar Funciones de Modulación por Ancho de Pulsos (PWM), contador de pulsos, generador de interrupciones en lenguaje Verilog, los registros de control para definir los modos de operación y hacer las pruebas respectivas.
- Redactar la hoja de todas las características y la guía de integración del módulo temporizador.

- Probar las características del circuito temporizador en una matriz de puertas de campo programable (FPGA).

1.4. HIPÓTESIS

1.4.1. HIPÓTESIS GENERAL

El diseño e implementación de un temporizador en lenguaje Verilog permitirá ser utilizado en un sistema en chip (SOC) para las funciones relacionadas con el tiempo.

1.4.2. HIPÓTESIS ESPECÍFICAS

- El diseño de un diagrama de bloques provee la funcionalidad del temporizador para un sistema en chip (SOC).

- La implementación de Funciones de Modulación por Ancho de Pulso, contador de pulsos, generador de interrupciones y registros de control, así como padrones de prueba garantizaran el funcionamiento correcto del sistema.

- La redacción de la hoja de todas las características y la guía de integración del módulo temporizador permitirá conozca las características y modos de funcionamiento, además de los pasos necesarios para configurar las diferentes funcionalidades.

- Será posible verificar las características del circuito en una matriz de puertas de campo programable (FPGA) para obtener resultados.

1.5. MATRIZ DE CONSISTENCIA

PROBLEMA GENERAL	OBJETIVO GENERAL	HIPOTESIS GENERAL	VARIABLES
Diseño de un componente digital en lenguaje HDL que cumplirá diversas tareas en el control relacionadas a control de tiempos	Diseñar e implementar un temporizador en lenguaje Verilog para ser utilizado en un sistema en chip (SOC).	El diseño de un diagrama de bloques provee la funcionalidad del temporizador para un sistema en chip (SOC).	**Variable independiente:** El diseño y la implementación de la estructura lógica del temporizador. **Variable dependiente:** Facilidad de uso del temporizador **Variable dependiente:** Cantidad de elementos de puertas lógicas y biestables requeridos para la implementación del sistema **Variable dependiente:** Numero de errores a corregir debido a la complejidad o simplicidad de la arquitectura de diseño
PROBLEMAS ESPECIFICOS	**OBJETIVOS ESPECIFICOS**	**HIPOTESIS ESPECIFICAS**	
Diseño del diagrama de bloques de un componente electrónico digital	Diseñar un diagrama de bloques de la funcionalidad del temporizador para un sistema en chip.	El diseño de un diagrama de bloques provee la funcionalidad del temporizador para un sistema en chip (SOC).	
Implementación y pruebas de un componente electrónico digital.	Implementar Funciones de Modulación por Ancho de Pulsos (PWM), contador de pulsos, generador de interrupciones en lenguaje Verilog, los registros de control para definir los modos de operación y hacer las pruebas respectivas.	La implementación de Funciones de Modulación por Ancho de Pulso, contador de pulsos, generador de interrupciones y registros de control, así como padrones de prueba garantizaran el funcionamiento correcto del sistema.	

PROBLEMAS ESPECIFICOS	OBJETIVOS ESPECIFICOS	HIPOTESIS ESPECIFICAS	VARIABLES
Implementación de una hoja de características (Datasheet) de un componente electrónico digital.	Redactar la hoja de todas las características y la guía de integración del módulo temporizador.	La redacción de la hoja de todas las características y la guía de integración del módulo temporizador permitirá conozca las características y modos de funcionamiento, además de los pasos necesarios para configurar las diferentes funcionalidades.	
Simulación de un componente electrónico digital en un sistema FPGA.	Probar las características del circuito temporizador en una matriz de puertas de campo programable (FPGA).	Será posible verificar las características del circuito en una matriz de puertas de campo programable (FPGA) para obtener resultados.	

CAPITULO II

REVISIÓN DE LITERATURA

2.1. MICROELECTRÓNICA

Tecnología mediante la cual se diseñan dispositivos electrónicos; Conjunto de reglas, normas, requisitos, materiales y procesos que, aplicados en una secuencia determinada, permite obtener como producto final un circuito integrado. Gordon Moore en 1965 analizó datos de producción de chips y notó que la cantidad de elementos que la tecnología acomodaba dentro de un chip se duplicaba aproximadamente cada 18 meses. Esa tendencia se ha mantenido hasta el presente. Todo esto está basado en el transistor, dispositivo que se ha usado en configuraciones complementarias CMOS para lograr construir lógica digital compleja. (Brown, 2013)

2.2. TRANSISTOR MOSFET

El transistor es un componente electrónico que juega un papel crucial dentro del diseño de componentes microelectrónicos, en el diseño de componentes digitales se usa la configuración CMOS para construir las principales compuertas lógicas existentes.

El transistor tiene tres partes. Una que emite electrones (emisor), otra que los recibe o recolecta (colector) y otra con la que se modula el paso de dichos electrones (base). Una pequeña señal eléctrica aplicada entre la base y emisor modula la que circula entre emisor y receptor. La señal base emisor puede ser muy pequeña en comparación con la emisora receptora. La señal emisor-receptor es aproximadamente la misma que la base-emisor, pero amplificada. El transistor se utiliza, por tanto, como amplificador. Además, todo amplificador oscila así que puede usarse como oscilador y también como rectificador y como conmutador on-off. El transistor también funciona por tanto como un

interruptor electrónico, siendo esta propiedad aplicada en la electrónica en el diseño de algunos tipos de memorias y de otros circuitos. (Cavanagh, 2017)

El transistor descrito en el párrafo anterior ha tenido versiones optimizadas para el consumo de potencia, este sería el transistor de efecto de campo de metal óxido semiconductor (MOSFET), este es el dispositivo que está en el centro de la revolución microelectrónica. Tiene en su interior una estructura compuesta de una puerta de poli silicio (anteriormente de metal), una capa fina de óxido de silicio, situada encima de la placa de silicio. Aplicando un voltaje a la puerta por encima de un cierto umbral, atrae electrones a la interfaz del semiconductor y el óxido. Estos electrones pueden entonces cerrar el interruptor entre otros dos grupos de electrones llamados fuente y sumidero ubicados en las proximidades, de forma que la corriente pueda fluir de uno a otro.

En esencia, un MOSFET es un interruptor que se activa electrónicamente. Si el voltaje aplicado a la puerta está por debajo del umbral, el interruptor se abre. Si está por encima, el interruptor se cierra. De esta forma se pueden hacer operaciones lógicas. Metal-óxido semiconductor complementario. Esto significa que realmente hay no uno sino dos tipos de MOSFET, uno que se activa con un voltaje por encima de un cierto valor y otro complementario que se activa con un voltaje por debajo de un valor diferente. Esto permite lo que se denomina lógica complementaria lo que básicamente quiere decir que cuando un interruptor está cerrado, el otro está abierto.

Esta configuración permite realizar operaciones lógicas sin tener una derivación directa entre la fuente de potencia y tierra evitando por lo tanto la corriente continua. Ninguna otra familia lógica lo hace. Este simple atributo es la clave que hay detrás de las enormes densidades de transistores de los modernos microprocesadores. Los semiconductores son el eje de la industria electrónica digital moderna, son dispositivos

electrónicos con elevado grado de miniaturización e integración de los transistores capaces de comportamientos electrónicos variables; posibilitando la realización de una serie de operaciones matemáticas a gran velocidad, el almacenamiento de información en forma digital o el control de una serie de tareas. (Clietti, 2017)

2.3. SISTEMA DE NUMERACION HEXADECIMAL

El sistema hexadecimal es un tipo de sistema de numeración posicional que utiliza como base el número 16. Sus números están representados por los 10 primeros dígitos de la numeración decimal, y el intervalo que va del número 10 al 15 están representados por las siguientes letras del alfabeto de la A – B – C – D – E y F. El uso que de la damos hoy en día al sistema hexadecimal se encuentra estrechamente ligado a la rama de la informática y las ciencias de la computación en las cuales, las diferentes operaciones del CPU usan el byte u octeto como la unidad básica de su memoria. Al ser éste un sistema numérico con Base-16, el sistema de numeración hexadecimal usa dieciséis dígitos diferentes con una combinación de números que van del 0 al 15. En otras palabras, hay 16 símbolos de dígitos posibles. El sistema hexadecimal se usa comúnmente en computadoras y sistemas digitales para reducir grandes cadenas de números binarios en conjuntos de cuatro dígitos para que podamos entenderlos fácilmente. Su uso actual está muy vinculado a la informática pues los computadores suelen utilizar el byte u octeto como unidad básica de memoria. El sistema hexadecimal actual fue introducido en el ámbito de la computación por primera vez por IBM en 1963. Una representación anterior, con 0–9 y u–z, fue usada en 1956 por la computadora Bendix G-15. Debido a que el sistema de numeración normal es en base decimal o base diez, se adoptó la idea de usar las seis primeras letras del alfabeto para poder suplir los números que hacían falta. De esta manera, los símbolos que se utilizan en el sistema hexadecimal son los siguientes: 0,

1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Se debe tener en cuenta que A= 10, B= 11, C= 12, D= 13, E= 14 y la letra F corresponde al número 15. (Li, 2015)

Los números hexadecimales y los binarios son ampliamente usados en el diseño electrónico digital ya que son fácilmente observables y el cálculo con los mismos se hace de forma más natural mientras son representados en registros de memoria.

2.4. SISTEMA DE NUMERACION BINARIA

El sistema numérico binario usa únicamente dos dígitos, 0 o 1, para representar todos los números ya sean naturales o fraccionarios entonces es ideal para ser usado como sistema numérico en sistemas electrónicos digitales, usando los transistores complementarios CMOS. (Magda, 2017)

2.5. REGISTROS

Un registro de desplazamiento es un circuito digital secuencial (es decir, que los valores de sus salidas dependen de sus entradas y de los valores anteriores) consistente en una serie de biestables, generalmente de tipo D, conectados en cascada, que basculan de forma sincrónica con la misma señal de reloj. Según las conexiones entre los biestables, se tiene un desplazamiento a la izquierda o a la derecha de la información almacenada. Un desplazamiento a la izquierda de un conjunto de bits es como multiplicar por 2, mientras que uno a la derecha, divide entre 2. Existen registros de desplazamiento bidireccionales, que pueden funcionar en ambos sentidos. Los registros universales, además de bidireccionales permiten la carga en paralelo. (Mishra, 2013)

2.6. SoC SISTEMA EN CHIP

Un sistema en chip o SoC (del inglés system on a chip o system on chip), describe la tendencia cada vez más frecuente de usar tecnologías de fabricación que integran todos

o gran parte de los módulos que componen un computador o cualquier otro sistema informático o electrónico en un único circuito integrado o chip. El diseño de estos sistemas puede estar basado en circuitos de señal digital, señal analógica, o incluso de señal mixta (tanto analógica como digital), y a menudo módulos o sistemas de radiofrecuencia (módulos de comunicación inalámbrica: Wi-Fi, Bluetooth, y otros). Un ámbito común de aplicación de la tecnología SoC son los sistemas embebidos. La diferencia principal de un SoC con un microcontrolador tradicional no debe pasarse por alto, puesto que estos rara vez disponen de más de 100 kilobytes de memoria RAM (de hecho, lo más frecuente es que las memorias, tanto la RAM como la flash, de un microcontrolador consten de unos pocos kilobytes), mientras que el término SoC es usado para procesadores más potentes y de arquitectura más compleja, como son los que integran los ordenadores y dispositivos actuales que dependen de chips o módulos de memoria externos para ser eficaces. Para sistemas más grandes y complejos sería impropio hablar de SoC, convirtiéndose el término en tal caso, más en una mera referencia o directiva a seguir que en la propia realidad de éstos: Aumentar la integración en un mismo chip con el objetivo de reducir costes y construir sistemas cada vez más reducidos (capaces de lo mismo o más que sistemas más antiguos y voluminosos). (Mehler, 2014)

2.7. MICROCONTROLADOR

Un microcontrolador es un sistema electrónico que incluye un microprocesador para ejecutar instrucciones, evidentemente incluye una memoria para almacenar el código del programa a ejecutar, también una memoria volátil para almacenar elementos temporales, usado generalmente para efectuar cálculos, aparte de estos componentes incluye también dispositivos periféricos tales como control de puertos de entrada/salida, temporizador, controlador de interrupciones, elementos conversores Analógico/Digital, Digital/Analógico entre otros. (Monk, 2016)

Podemos mencionar que los primeros microcontroladores incluían básicamente un microprocesador con elementos de acceso a memoria que podían ser usados para leer y ejecutar instrucciones, actualmente los microcontroladores modernos incluyen elementos tan complejos como comunicación inalámbrica, Wifi, bluetooth, y controladores de protocolos de comunicaciones. (Nano, 2017)

2.8. FRECUENCIA

La frecuencia es una magnitud que mide el número de repeticiones por unidad de tiempo de cualquier fenómeno o suceso periódico. Para calcular la frecuencia de un suceso, se contabilizan un número de ocurrencias de éste, teniendo en cuenta un intervalo temporal, y luego estas repeticiones se dividen por el tiempo transcurrido. Según el Sistema Internacional (SI), la frecuencia se mide en hercios (Hz), en honor a Heinrich Rudolf Hertz. Un hercio es la frecuencia de un suceso o fenómeno repetido por segundo. Así, un fenómeno con una frecuencia de dos hercios se repite dos veces por segundo. Esta unidad se llamó originalmente «ciclo por segundo» (cps). Otras unidades para indicar frecuencias son revoluciones por minuto (rpm o r/min según la notación del SI); las pulsaciones del corazón se miden en latidos por minuto (lat/min) y el tempo musical se mide en «pulsos por minuto» (bpm, del inglés “beats per minute”). (Rajewski, 2017)

2.9. TEMPORIZADOR

Un temporizador es un dispositivo electrónico, que permite medir el tiempo. Se han usado sistemas mecánicos hasta que se comenzó a usar circuitos electrónicos con mayor precisión y de menor tamaño. Cuando trascurre el tiempo configurado se hace saltar una alarma o alguna otra función a modo de advertencia o con el fin de ejecutar alguna acción, estos tiempos pueden ser cíclicos o de una sola ejecución dependiendo de la configuración dada por el usuario. Generalmente todo circuito microcontrolador o más

complejo como un SoC incluye uno a varios temporizadores. el componente de temporización dentro de un circuito integrado incluye varias funciones como conteo, división de frecuencia, generación de interrupciones incluso generando señales moduladas como (PWM modulación por amplitud de pulso), entre diseños complejos que incluyen diversas funcionalidades enfocados a equipos de telecomunicaciones, sistemas de control para automóviles y dispositivos de bajo consumo que son usados actualmente para acompañar al usuario tradicional en todo momento ayudando a medir elementos de información de salud como presión, temperatura y alarmas para sedentarismo. (Readler, 2014)

2.10. ICARUS VERILOG

Icarus Verilog es un software para simulación de circuitos electrónicos digitales escritos en lenguaje verilog, sobre el lenguaje de descripción de hardware Verilog. Admite las versiones de 1995, 2001 y 2005 del estándar definido por la IEEE, partes de SystemVerilog y algunas extensiones. Icarus verilog está disponible para Linux, FreeBSD, OpenSolaris, AIX, Microsoft Windows y Mac OS X. Lanzado bajo la Licencia Pública General de GNU, Icarus Verilog es un software libre. A partir de la versión 0.9, Icarus se compone de un compilador Verilog (que incluye un preprocesador Verilog) con soporte para complementos, y una máquina virtual que simula el diseño. La versión v10.0, además de mejoras generales y correcciones de errores, agrega soporte preliminar para VHDL. La versión actualizada a la fecha (2018) es la 11 que contiene algunas mejoras en cuanto a la posibilidad de usar señales vectoriales. Este compilador/simulador puede ser instalado tanto en Windows y Linux. (Romano, 2016)

2.11. CADENCE INCISIVE

Incisive es un conjunto de herramientas de Cadence Design Systems relacionadas con el diseño y la verificación de ASIC, SoC y FPGA. Incisive es comúnmente referido por el nombre NCSim en referencia al motor de simulación del núcleo. A fines de la década de 1990, el conjunto de herramientas era conocido como ldv (diseño lógico y verificación). (Rotch, 2015)

Esta herramienta permite la compilación elaboración y simulación de circuitos electrónicos representados en lenguajes HDL como verilog o vhdl inclusive modelos analógicos representados en lenguaje verilog ams (Analog Mixed Signal), se puede realizar un procedimiento en 3 pasos, compilación (ncvlog), elaboración (ncelab), simulación (ncsim) o usar un único paso en el cual el software ejecuta todo el proceso automáticamente mediante un solo comando denominado (irun). (Sutherland, 2017)

2.12. GESTIÓN

Conjunto de operaciones que se realizan para dirigir y administrar un negocio o una empresa. La noción de gestión, por lo tanto, se extiende hacia el conjunto de trámites que se llevan a cabo para resolver un asunto o concretar un proyecto. La gestión es también la dirección o administración de una compañía o de un negocio. El segundo pilar básico es la cultura o lo que es lo mismo el grupo de acciones para promover los valores de la empresa en cuestión, para fortalecer la misma, para recompensar los logros alcanzados y para poder realizar las decisiones adecuadas. A todo ello, se une el tercer eje de la gestión: la estructura. Bajo este concepto lo que se esconde son las actuaciones para promover la cooperación, para diseñar las formas para compartir el conocimiento y para situar al frente de las iniciativas a las personas mejores cualificadas. El cuarto y último pilar es el de la ejecución que consiste en tomar las decisiones adecuadas y

oportunas, fomentar la mejora de la productividad y satisfacer las necesidades de los consumidores. Es importante resaltar que existen distintos tipos de gestión. La gestión social, por ejemplo, consiste en la construcción de diferentes espacios para promover y hacer posible la interacción entre distintos actores de una sociedad. (Taraate, 2016)

2.13. DIAGRAMA DE PINES

También llamado Pinout es un término anglosajón, que en traducción libre significa patillaje, o más correctamente asignación de patillaje o "disposición de los pines". Se utiliza en electrónica para determinar la función de cada pin (patilla) en un circuito integrado, o bien en un dispositivo electrónico discreto, en cuanto a los circuitos representados en lenguajes HDL se usa de la misma forma representando la función de cada señal de interfaz de entrada o salida. (Thomas, 2016)

El diagrama de pines del temporizador indicara las señales que son usadas para comunicarse con otros componentes tales como microprocesador, tratador de interrupciones, controlador de reloj, controlador de reset entre otros que cumplen un papel importante en la operación del temporizador y en la operación en general del conjunto dentro de un SoC por ejemplo en un microcontrolador (MCU). (Unsalan, 2017)

2.14. DIAGRAMA DE BLOQUES:

Un diagrama de bloques es una representación gráfica de los componentes de un sistema. En muchos casos, estos diagramas permiten entender el comportamiento y la conexión del sistema. En un diagrama de bloques se unen todas las variables del sistema, mediante bloques funcionales, el cual es un símbolo para representar la operación matemática que sobre la señal de entrada hace el bloque para producir la salida. (Williams, 2014)

2.15. VERILOG

Es un lenguaje de descripción de hardware usado para modelar sistemas electrónicos. El lenguaje, algunas veces llamado Verilog HDL, soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señal mixta a diferentes niveles de abstracción.

Este lenguaje fue estandarizado como IEEE 1364, es un lenguaje de descripción de hardware (HDL) utilizado para modelar sistemas electrónicos. Se utiliza más comúnmente en el diseño y verificación de circuitos digitales en el nivel de abstracción de registro-transferencia. También se utiliza en la verificación de circuitos analógicos y circuitos de señal mixta, En 2009, el estándar Verilog (IEEE 1364-2005) se fusionó en el Estándar SystemVerilog, creando bajo el estándar IEEE 1800-2009. Desde entonces, Verilog es oficialmente parte del lenguaje SystemVerilog. La versión actual es la norma IEEE 1800-2017. Los diseñadores de Verilog querían un lenguaje con una sintaxis similar al lenguaje de programación C, que ya era ampliamente utilizado en el desarrollo de software de ingeniería. Al igual que C, Verilog distingue entre mayúsculas y minúsculas y tiene un preprocesador básico (aunque menos sofisticado que el de ANSI C / C ++). Sus palabras clave de flujo de control (if / else, for, while, case, etc.) son equivalentes, y su prioridad de operadores compatible con C. Las diferencias sintácticas incluyen: anchos de bits requeridos para declaraciones de variables, demarcación de bloques de procedimientos (Verilog usa el principio / final en lugar de llaves {}) y muchas otras diferencias menores. Verilog requiere que las variables tengan un tamaño definido. En C, estos tamaños se asumen a partir del 'tipo' de la variable (por ejemplo, un tipo entero puede ser de 8 bits). (Wilson, 2015)

2.16. NANOELECTRÓNICA

Es la rama de la electrónica referente a los circuitos electrónicos miniaturizados integrados en chips semiconductores, siendo su elemento de base el transistor. Hasta hace poco, el tamaño de los transistores se medía en micrómetros (µm: microelectrónica), pero hoy en día se fabrican transistores de 90 o 65 nanómetros (nm: nanoelectrónica). La Nanoelectrónica se refiere al uso de la nanotecnología en componentes electrónicos especialmente en transistores. Aunque el termino nanotecnología se usa normalmente para definir la tecnología de menos de 100 nm de tamaño, la electrotecnia se refiere a menudo a transistores de tamaño tan reducido que se necesita un estudio más exhaustivo de las interacciones interatómicas y de las propiedades mecánico-cuánticas. Es por ello por lo que transistores actuales como por ejemplo CMOS90 de TSMC o los procesamientos Pentium 4 de Intel, no son instalados en esta categoría a pesar de contar con un tamaño menor que 90 o 65 nm. A los dispositivos Nanoelectrónica se les considera una tecnología disruptiva ya que los ejemplos actuales son sustancialmente diferentes que los transistores tradicionales. Entre ellos cabe destacar la electrónica de semiconductor de moléculas híbridas nano turbos/nano hilos de una dimensión o la electrónica molecular avanzada. El sub-voltaje y la nanoelectrónica de sub-voltaje profundo son campos específicos e importantes de I+D, y la aparición de nuevos circuitos integrados operando a un nivel de consumo energético por procesamiento de un bit próximo al teórico fundamentalmente, tecnológico diseño, metodológico arquitectónico, algorítmico es inevitable. Una aplicación de importancia que pueda beneficiarse finalmente de esta tecnología en lo referente a operaciones lógicas es la computación reversible. (Brown, 2013)

2.17. FPGA

Una FPGA o matriz de puertas programables (del inglés field-programmable gate array) es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada en el momento mediante un lenguaje de descripción especializado. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip. Las FPGA se utilizan en aplicaciones similares a los ASIC sin embargo son más lentas, tienen un mayor consumo de energía y no pueden abarcar sistemas tan complejos como ellos. A pesar de esto, las FPGA tienen las ventajas de ser reprogramables (lo que añade una enorme flexibilidad al flujo de diseño), sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor. Ciertos fabricantes cuentan con FPGA que sólo se pueden programar una vez, por lo que sus ventajas e inconvenientes se encuentran a medio camino entre los ASIC y las FPGA re Históricamente las FPGA surgen como una evolución de los conceptos desarrollados en las PAL y los CPLD. Una jerarquía de interconexiones programables permite a los bloques lógicos de un FPGA ser interconectados según la necesidad del diseñador del sistema, algo parecido a una placa de inserción (es una placa de uso genérico reutilizable o semipermanente) programable. Estos bloques lógicos e interconexiones pueden ser programados después del proceso de manufactura por el usuario/diseñador, así que la FPGA puede desempeñar cualquier función lógica necesaria. Una tendencia reciente ha sido combinar los bloques lógicos e interconexiones de las FPGA con microprocesadores y periféricos relacionados para formar un sistema programable en un chip. Ejemplo de tales tecnologías híbridas pueden ser encontradas en los dispositivos Virtex-II PRO y Virtex-4 de Xilinx, los cuales incluyen uno o más procesadores PowerPC embebidos junto con la lógica de la FPGA.

El FPSLIC de Atmel es otro dispositivo similar, el cual usa un procesador AVR en combinación con la arquitectura lógica programable de Atmel. Otra alternativa es hacer uso de núcleos de procesadores implementados haciendo uso de la lógica de la FPGA. Esos núcleos incluyen los procesadores MicroBlaze y PicoBlaze de Xlinx, Nios y Nios II de Altera, y los procesadores de código abierto LatticeMicro32 y LatticeMicro8. Cualquier circuito de aplicación específica puede ser implementado en una FPGA, siempre y cuando esta disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan las FPGA incluyen a los DSP (procesamiento digital de señales), radio definido por software, sistemas aeroespaciales y de defensa, prototipos de ASIC, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, entre otras. Cabe notar que su uso en otras áreas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo. Existe código fuente disponible (bajo licencia GNU GPL) de sistemas como microprocesadores, microcontroladores, filtros, módulos de comunicaciones y memorias, entre otros. Estos códigos se llaman cores. (Cavanagh, 2017)

2.18. HDL (LENGUAJE DE DESCRIPCION DE HARDWARE)

Un lenguaje de descripción de hardware (HDL, hardware description language) es un lenguaje de programación especializado que se utiliza para definir la estructura, diseño y operación de circuitos electrónicos, y más comúnmente, de circuitos electrónicos digitales, como un microprocesador, temporizador, el convertidor analógico-digital o cualquier componente electrónico que pueda ser representado en función del tiempo. Así, los lenguajes de descripción de hardware hacen posible una descripción formal de un circuito electrónico, y posibilitan su análisis automático y su simulación.

Los lenguajes de descripción de hardware se parecen mucho a otros lenguajes de programación de ordenadores tales como el C o Java: básicamente consisten en una descripción textual con expresiones, declaraciones y estructuras de control. Sin embargo, una importante diferencia entre los HDL y otros lenguajes de programación está en que el HDL incluye explícitamente la noción de tiempo.

Cabe señalar que no se debe confundir con los lenguajes usados para la elaboración de software ya que estos modelan una secuencia de comandos que ejecuta un microprocesador mientras que HDL representa la secuencia o flujo de datos de todo el sistema electrónico en general. (Clietti, 2017)

2.19. INTERRUPCIONES

En el contexto de la informática, una interrupción (del inglés interruptor requeté, en español «petición de interrupción») es una señal recibida por el procesador de una computadora, para indicarle que debe «interrumpir» el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación. Una interrupción es una suspensión temporal de la ejecución de un proceso, para pasar a ejecutar una subrutina de servicio de interrupción, la cual, por lo general, no forma parte del programa, sino que pertenece al sistema operativo o al BIOS. Una vez finalizada dicha subrutina, se reanuda la ejecución del programa. Las interrupciones son generadas por los dispositivos periféricos habilitando una señal del CPU (llamada IRQ del inglés "interruptor requeté") para solicitar atención de este. Por ejemplo. cuando un disco duro completa una lectura solicita atención al igual que cada vez que se presiona una tecla o se mueve el ratón. La primera técnica que se empleó para esto fue el polling, que consistía en que el propio procesador se encargara de sondear los dispositivos periféricos cada cierto tiempo para averiguar si tenía pendiente alguna comunicación para él. Este método presentaba el inconveniente de

ser muy ineficiente, ya que el procesador consumía constantemente tiempo y recursos en realizar estas instrucciones de sondeo. El mecanismo de interrupciones fue la solución que permitió al procesador desentenderse de esta problemática, y delegar en el dispositivo periférico la responsabilidad de comunicarse con él cuando lo necesitara. El procesador, en este caso, no sondea a ningún dispositivo, sino que queda a la espera de que estos le avisen (le "interrumpan") cuando tengan algo que comunicarle (ya sea un evento, una transferencia de información, una condición de error, etc.). Todos los dispositivos que deseen comunicarse con el procesador por medio de interrupciones deben tener asignada una línea única capaz de avisar al CPU cuando le requiere para realizar una operación. Esta línea se denomina IRQ. Las IRQ son líneas que llegan al controlador de interrupciones, un componente de hardware dedicado a la gestión de las interrupciones, y que puede estar integrado en el procesador principal o ser un circuito separado conectado al mismo. El controlador de interrupciones debe ser capaz de habilitar o inhibir las líneas de interrupción y establecer prioridades entre las mismas. Cuando varias líneas de petición de interrupción se activan a la vez, el controlador de interrupciones utilizará estas prioridades para escoger la interrupción sobre la que informará al procesador principal. También puede darse el caso de que una rutina de tratamiento de interrupción sea interrumpida para realizar otra rutina de tratamiento de una interrupción de mayor prioridad a la que se estaba ejecutando; aunque hay interrupciones que no se pueden deshabilitar (conocidas como interrupciones no enmascarables o NMI). Un procesador principal que no tenga un controlador de interrupciones integrado suele tener una única línea de interrupción llamada habitualmente INT. Esta línea es activada por el controlador de interrupciones cuando tiene una interrupción que servir. Al activarse esta línea, el procesador consulta los registros del controlador de interrupciones para averiguar cuál IRQ hay que atender. A partir del número del IRQ busca en la tabla de vectores de

interrupción la dirección de la rutina a la que debe llamar para atender la petición del dispositivo asociado a dicha IRQ. El bus de control de la placa base dispone de líneas específicas para el sistema de interrupciones. Un PC típico dispone en su placa base de un controlador de interrupciones 8259 de Intel o de un circuito integrado análogo. Este dispositivo electrónico dispone de hasta 16 líneas IRQ, numeradas desde el 00 hasta el 15. En las nuevas placas base este circuito está integrado junto con el resto del chipset y permite hasta 24 interrupciones. En el IBM PC y XT existían 8 líneas de petición de interrupción manejadas por el controlador de interrupciones Intel 8259. Estas líneas están numeradas del 0 al 7, las dos primeras están asignadas al timer tick del temporizador Intel 8253, y al teclado. Solo quedaban 6 líneas para otros dispositivos, que aparecen como tales en el bus de control (IRQ2 - IRQ7). A partir del modelo AT se añadieron otras 8 líneas, numeradas del 8 al 15, mediante un segundo controlador de interrupciones (PIC), aunque la tecnología empleada exigió colgarlo de la línea IRQ2 del primero, de forma que esta línea se dedica a atender las interrupciones del segundo controlador a través de la línea 9 de este último, y la línea 8 se dedicó al reloj de tiempo real, un dispositivo que no existía en los modelos XT. Aunque internamente se manejan 16 líneas, no todas tienen contacto en los zócalos del bus externo (son las marcadas con asterisco en la tabla que sigue). La razón de esta ausencia en los zócalos de conexión es que son de asignación fija, y solo son usadas por ciertos dispositivos instalados en la propia placa base. En concreto la línea NMI está asignada al mecanismo de control de paridad de la memoria, la línea 0 está asignada al cronómetro del sistema y la línea 1 al chip que controla el teclado (dispositivos que pueden requerir atención urgente por parte del procesador). Es costumbre denominar IRQx a las que tienen prolongación en el bus. Teóricamente las restantes líneas podrían ser asignadas a cualquier nuevo dispositivo, pero en la práctica algunas están reservadas a dispositivos estándar. Por ejemplo, IRQ3 está casi siempre

asignado al puerto serie COM2 y el IRQ4 al COM1; IRQ6 al controlador estándar de disquetes y IRQ7 al puerto de impresora LPT1. La tabla 1 muestra las asignaciones clásicas para el XT y el AT. En sistemas más modernos utilizan la arquitectura APIC de Intel con 24 líneas y 8 extra para enrutar las interrupciones PCI. Las interrupciones de hardware son aquellas interrupciones que se producen como resultado de, por lo general, una operación de E/S. No son producidas por ninguna instrucción de un programa sino por las señales que emiten los dispositivos periféricos para indicarle al procesador que necesitan ser atendidos. Cuando el microprocesador accede a un periférico (disco duro, puerto de comunicación...), puede transcurrir algún tiempo antes de que los datos sean obtenidos o transmitidos. La solución más simple es esperar hasta recibir los datos o hasta que se haya efectuado la transmisión (polling), pero esta solución bloquea todos los programas en ejecución, y eso no puede admitirse en un sistema multitarea. Por ello, en los sistemas modernos se prefiere un funcionamiento mediante interrupciones, ya que éstas permiten mejorar la productividad del procesador, de forma que este último puede ordenar una operación de entrada o salida y, en lugar de tener que realizar una espera activa, se puede dedicar a atender a otro proceso o aplicación hasta que el dispositivo esté de nuevo disponible, siendo dicho dispositivo el encargado de notificar al procesador mediante la línea de interrupción que ya está preparado para continuar o terminar la operación de entrada o salida. Las interrupciones por software, también denominadas llamadas al sistema son aquellas generadas por un programa mientras este está ejecutándose. En general, actúan de la siguiente manera: Un programa en ejecución llega a una instrucción que requiere del sistema operativo para alguna tarea, por ejemplo, para leer un archivo en el disco duro (cuando un programa necesita un dato exterior, se detiene y pasa a cumplir con las tareas de recoger ese dato). En ese momento por tanto llama al sistema y se interrumpe virtualmente hasta recibir respuesta, en el ejemplo anterior hasta

que no se haya leído el disco y el archivo esté en memoria principal. Durante esa espera las instrucciones que se ejecutarán no serán del programa, sino del sistema operativo. Una vez éste termine su rutina ordenará reanudar la ejecución del programa auto interrumpido en espera. Por último, la ejecución del programa se reanuda. (Li, 2015)

2.20. XILINX

Xilinx diseña, desarrolla y comercializa productos lógicos programables, incluidos los circuitos integrados (CI), herramientas de software de diseño, funciones de sistema predefinidas entregados como núcleos de propiedad intelectual (IP), servicios de diseño, formación del cliente, ingeniería de campo y soporte técnico. Xilinx vende FPGAs y CPLDs para fabricantes de equipos electrónicos en los mercados finales, así como a los de comunicaciones, industrias, consumidores, automoción y procesamiento de datos. Los FPGAs de Xilinx se han utilizado para el ALICE (A Large Ion Collider Experiment) en el laboratorio europeo CERN, en la frontera francosuiza para mapear y desentrañar las trayectorias de miles de partículas subatómicas. Xilinx también se ha involucrado en una asociación con el Laboratorio de Investigación de Vehículos espaciales de la Fuerza Aérea de los Estados Unidos para el desarrollo de FPGAs para resistir los efectos dañinos de la radiación en el espacio, que son 1000 veces menos sensibles a la radiación espacial que su equivalente comercial, para el despliegue de nuevos satélites. Las familias de FPGAs Virtex-II Pro, Virtex-4, Virtex-5, y Virtex-6, que incluyen hasta dos núcleos IBM PowerPC embebidos, están dirigidos a las necesidades de diseñadores del sistema en chip (SoC, del inglés system-on-chip). Los FPGAs de Xilinx pueden ejecutar un sistema operativo incrustado regular (como Linux o vxWorks) y se pueden aplicar procesadores periféricos en la lógica programable. Los núcleos IP de Xilinx incluyen IP para funciones simples (codificadores BCD, contadores, etc.), para núcleos específicos de dominio (núcleos de procesamiento de señal digital, FFT y FIR) a

los sistemas complejos (núcleos de redes multi-gigabit, el microprocesador suave MicroBlaze y el microcontrolador compacto Picoblaze). Xilinx también crea núcleos personalizados por una tarifa. El kit de herramientas de diseño principal que Xilinx proporciona a los ingenieros es el Vivado Design Suite, un entorno de diseño integrado (IDE) con un sistema de herramientas IC de nivel, construidos en un modelo de datos escalable compartida y un entorno de depuración común. Vivado incluye nivel de sistema electrónico (ESL); herramientas de diseño para sintetizar y verificar IP algorítmica basada en C; normas de embalaje basado tanto en IP algorítmica y RTL para su reutilización; normas de costura (stitching) y sistemas IP de integración de todos los tipos de bloques de construcción del sistema basado; y la verificación de bloques y sistemas. Una versión gratuita de Edición WebPACK del Vivado proporciona a los diseñadores con una versión limitada del entorno de diseño. El Kit Embedded Developer's de Xilinx (EDK) apoya a los núcleos embebidos de PowerPC 405 y 440 (en chips de Virtex-II Pro y algunos Virtex-4 y -5) y el núcleo Microblaze. El Sistema Generador de Xilinx para DSP implementa diseños DSP en FPGAs de Xilinx. Una versión gratuita de su software EDA llamada ISE WebPack se utiliza con algunos de sus chips que carecen de alto rendimiento. Xilinx es el único (a partir del 2007) proveedor de FPGA para distribuir un programa gratuito de herramientas de cadena de síntesis nativo de Linux. En abril del 2012, Xilinx introdujo un rediseño de su conjunto de herramientas para los sistemas programables, llamada Vivado Design Suite. Esta IP y el software de diseño centrado en el sistema es compatible con los nuevos dispositivos de alta capacidad, y acelera el diseño de la lógica programable y I/O.Vivado proporciona una integración e implementación más rápida para los sistemas programables en dispositivos con tecnología 3D de interconexión de silicio apilado, sistemas de procesamiento ARM, señal analógica mixta (AMS), y muchos de los núcleos de semiconductores de la propiedad intelectual (IP). (Mishra, 2013)

2.21. CIRCUITO INTEGRADO

El Intel 8253 y el 8254 son temporizadores de intervalos programables (PIT), que realizan funciones de temporización y conteo. Fueron primariamente diseñadas para el Intel 8080/8085, pero usadas más tarde en el IBM PC y en los sistemas x86. Se encuentran en todos los sistemas compatibles de IBM. El timer 8253 tiene 3 contadores independientes, llamados canales. Cada contador puede ser programado para operar en uno de seis modos. Una vez programados, los contadores pueden realizar la tarea asignada según el modo. Adicionalmente hay un registro de control (Control Word Register) que permite programar cada uno de los tres contadores independientemente. Por medio de este registro de control puede programarse un contador en uno de seis modos de operación. Físicamente, el 8253 es accedido por cuatro puertos de entrada/salida a partir de una dirección base, que es definida mediante circuitos externos. En la dirección base + 0, base + 1, y base + 2 se encuentran los tres contadores, y en la dirección base + 3 se encuentra el registro de control. El valor de los contadores puede leerse y escribirse, y el registro de control es de solo escritura. En el IBM PC, el timer 8253 se encuentra localizado en la dirección base de entrada/salida 40h. Por lo tanto, los tres contadores están en los puertos de entrada/salida 40h, 41h y 42h, y el registro de control se encuentra en el puerto 43h. Los bits 7 y 6 de la palabra de control definen cuál contador se está programando. Para seleccionar un contador se coloca en estos bits el número del contador que se desea programar, 00, 01, 10 (en binario), para los contadores 0, 1 y 2 respectivamente. El valor 11 (en binario) es inválido, ya que no existe el contador 3. Los bits 3, 2, y 1, de la palabra de control, indican el modo de operación del contador. Cada contador puede programarse en uno de seis modos de operación. Para programarlo, se envía al registro Control Word del chip la información de cómo se quiere que trabaje el contador y luego se envía al contador el valor inicial de conteo. Entonces el contador

comenzará a funcionar de acuerdo a lo previsto en el modo de operación. Al programar los modos 2 y 3, los bits más significativos del modo de operación, bit 3 de la palabra de control, es ignorado, y no importa su valor. El timer 8253 fue usado como temporizador en el computador IBM PC desde su introducción en 1981, y a partir de entonces, en los compatibles. Su funcionalidad es heredada hasta los computadores clones compatibles de hoy en día (de la arquitectura x86). El IBM AT usaba el Intel 8254 como timer programable. Luego el timer desapareció como un circuito integrado individual y pasó a formar parte de chipset de la tarjeta madre. Hoy en día los computadores compatibles tienen esta funcionalidad en el chip southbridge. En algunos chipsets modernos, este cambio puede aparecer como diferencias de tiempo medibles en acceder un PIT usando el espacio de direcciones de entrada/salida x86. Lecturas y escrituras a tales registros del PIT en el espacio de dirección de entrada/salida pueden completarse mucho más rápidamente. Las tarjetas madre más nuevas también incluyen un contador a través del Advanced Configuration and Power Interface (ACPI), un contador en el Local Advanced Programmable Interrupt Controller (Local APIC), y un High Precision Event Timer. El CPU en sí mismo también proporciona la facilidad del Time Stamp Counter (TSC). (Mehler, 2014)

2.22. QUARTUS II

Quartus II es una herramienta de software producido por Altera para el análisis y la síntesis de los diseños en HDL. Quartus II permite al desarrollador compilar sus diseños, realizar análisis temporales, examinar diagramas RTL y configurar el dispositivo de destino con el programador. La Edición Web es una versión gratuita de Quartus II que puede ser descargada o enviada gratuitamente por correo. Esta edición permite la compilación y la programación de un número limitado de dispositivos Altera. La familia de FPGAs de bajo coste Cyclone, está soportada por esta edición, por lo que los pequeños

desarrolladores no tienen problemas en el coste del desarrollo de software. La Edición Web es una versión gratuita de Quartus II que puede ser descargada o enviada gratuitamente por correo. Esta edición permite la compilación y la programación de un número limitado de dispositivos Altera. La familia de FPGAs de bajo coste Cyclone, está soportada por esta edición, por lo que los pequeños desarrolladores y desarrolladoras no tendrán problemas por el coste del desarrollo de software. Se requiere un registro de licencia para utilizar la Edición Web de Quartus II, la cual es gratuita y puede ser renovada ilimitadamente o de pago. La Edición de Suscripción de Quartus II también está disponible para ser descargada gratuitamente, pero se debe pagar una licencia para utilizar todas las funciones de la aplicación. Se puede utilizar la licencia gratuita de la Edición Web en esta edición, restringiendo el número de dispositivos que pueden ser utilizados con la aplicación. (Monk, 2016)

2.23. FLIP-FLOPS

Un biestable (flip-flop en inglés), es un multivibrador capaz de permanecer en uno de dos estados posibles durante un tiempo indefinido en ausencia de perturbaciones.1 Esta característica es ampliamente utilizada en electrónica digital para memorizar información. El paso de un estado a otro se realiza variando sus entradas. Dependiendo del tipo de dichas entradas los biestables se dividen en: Asíncronos: solamente tienen entradas de control. El más empleado es el biestable RS. Síncronos: además de las entradas de control posee una entrada de sincronismo o de reloj.

Si las entradas de control dependen de la de sincronismo se denominan síncronas y en caso contrario asíncronas. Por lo general, las entradas de control asíncronas prevalecen sobre las síncronas. La entrada de sincronismo puede ser activada por nivel (alto o bajo) o por flanco (de subida o de bajada). Dentro de los biestables síncronos

activados por nivel están los tipos RS y D, y dentro de los activos por flancos los tipos JK, T y D. Los biestables síncronos activos por flanco (flip-flop) se crearon para eliminar las deficiencias de los latches (biestables asíncronos o sincronizados por nivel). Dispositivo de almacenamiento temporal de 2 estados (alto y bajo), cuyas entradas principales permiten al ser activadas:

R: el borrado (reset en inglés), puesta a 0 ó nivel bajo de la salida.

S: el grabado (set en inglés), puesta a 1 ó nivel alto de la salida

Si no se activa ninguna de las entradas, el biestable permanece en el estado que poseía tras la última operación de borrado o grabado. En ningún caso deberían activarse ambas entradas a la vez, ya que esto provoca que las salidas directa (Q) y negada (Q') queden con el mismo valor: a bajo, si el flip-flop está construido con puertas NOR, o a alto, si está construido con puertas NAND. El problema de que ambas salidas queden al mismo estado está en que al desactivar ambas entradas no se podrá determinar el estado en el que quedaría la salida. Por eso, en las tablas de verdad, la activación de ambas entradas se contempla como caso no deseado (N. D.). (Nano, 2017)

2.24. PWM

La modulación por ancho de pulsos (también conocida como PWM, siglas en inglés de pulse-width modulation) de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga. 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 oscilador de onda dientes de sierra, mientras

que la otra queda disponible para la señal moduladora. En la salida la frecuencia es generalmente igual a la de la señal dientes de sierra y el ciclo de trabajo está en función de la portadora. La principal desventaja que presentan los circuitos PWM es la posibilidad de que haya interferencias generadas por radiofrecuencia. Estas pueden minimizarse ubicando el controlador cerca de la carga y realizando un filtrado de la fuente de alimentación. En la actualidad existen muchos circuitos integrados en los que se implementa la modulación PWM, además de otros muy particulares para lograr circuitos funcionales que puedan controlar fuentes conmutadas, controles de motores, controles de elementos termoeléctricos, choppers para sensores en ambientes ruidosos y algunas otras aplicaciones. Se distinguen por fabricar este tipo de integrados compañías como Texas Instruments, National Semiconductor, Maxim, y algunas otras más. La modulación por ancho de pulsos es una técnica utilizada para regular la velocidad de giro de los motores eléctricos de inducción o asíncronos. Mantiene el par motor constante y no supone un desaprovechamiento de la energía eléctrica. Se utiliza tanto en corriente continua como en alterna, como su nombre lo indica, al controlar: un momento alto (encendido o alimentado) y un momento bajo (apagado o desconectado), controlado normalmente por relès (baja frecuencia) o MOSFET o tiristores (alta frecuencia). Otros sistemas para regular la velocidad modifican la tensión eléctrica, con lo que disminuye el par motor; o interponen una resistencia eléctrica, con lo que se pierde energía en forma de calor en esta resistencia. Otra forma de regular el giro del motor es variando el tiempo entre pulsos de duración constante, lo que se llama modulación por frecuencia de pulsos. En los motores de corriente alterna también se puede utilizar la variación de frecuencia. La modulación por ancho de pulsos también se usa para controlar servomotores, los cuales modifican su posición de acuerdo al ancho del pulso enviado cada un cierto período que depende de cada servo motor. Esta información puede ser enviada utilizando un microprocesador

como el Z80, o un microcontrolador (por ejemplo, un PIC 16F877A, 16F1827, 18F4550, etc. de la empresa Microchip). Otra aplicación es enviar información de manera analógica. Es útil para comunicarse de forma analógica con sistemas digitales. Para un sistema digital, es relativamente fácil medir cuánto dura una onda cuadrada. Sin embargo, si no se tiene un conversor analógico digital no se puede obtener información de un valor analógico, ya que sólo se puede detectar si hay una determinada tensión, 0 o 5 voltios por ejemplo (valores digitales de 0 y 1), con una cierta tolerancia, pero no puede medirse un valor analógico. Sin embargo, el PWM en conjunción con un oscilador digital, un contador y una puerta AND como puerta de paso, podrían fácilmente implementar un ADC. (Rajewski, 2017)

2.25. ANTECEDENTES DEL PROYECTO

Se encontraron investigaciones que pueden ser de ayuda. Estos antecedentes ayudarán a tener opciones al implementar este sistema, estos serán mencionadas a continuación:

SIMULACIÓN DE CIRCUITOS BASADA EN LA IMPLEMENTACIÓN DE MODELOS AVANZADOS DE DISPOSITIVOS Y SENSORES ELECTRÓNICOS EN VERILOG

Andrés Roldan Aranda – Granada - Valencia 2012

RESUMEN:

Esta tesis describe el proceso de modelado de diferentes dispositivos electrónicos, la elaboración de modelos compactos para simuladores de circuitos electrónicos y la simulación de numerosas aplicaciones circuitales usando estos modelos compactos.

En el Capítulo 1 se introducen las herramientas para la implementación de modelos avanzados de dispositivos. Se recorre la evolución histórica y los diferentes tipos de simuladores de circuitos electrónicos existentes. Se presentan los modelos compactos diseñados tanto en código fuente como en Verilog-A y sus ventajas e inconvenientes.

En el Capítulo 2 describe el simulador de circuitos desarrollado en el Grupo de Nanoelectrónica de la Universidad de Granada y el procedimiento para la modificación de modelos compactos ya existentes y la incorporación de nuevos efectos físicos en la simulación de circuitos.

Con el Capítulo 3 se desarrollan modelos compactos para dispositivos multipuerta donde se incluyen diferentes efectos físicos esenciales en dispositivos nanométricos (efectos cuánticos, del overshoot de la velocidad, de la velocidad de saturación, de canal corto). Se estudian las implicaciones de estos efectos en el comportamiento de diferentes circuitos. Se desarrollan los modelos de control de carga en inversión, corriente y capacidades para transistor SGT y doble puerta.

En el Capítulo 4 se introducen los diferentes dispositivos magnetorresistivos y su utilización como sensores de corriente. Se realiza la caracterización experimental de los dispositivos incluyendo su comportamiento térmico (efectos de autocalentamiento y de ruido). Mediante el uso de varios dispositivos se construyen puentes de Wheatstone que son igualmente modelados.

En el Capítulo 5 se presentan los resultados experimentales de un conjunto de aplicaciones con sensores magnetorresistivos de corriente (conversores de resistencia a frecuencia, convertidores generalizados de impedancias y medidores de potencia). Se realizan simulaciones de los circuitos incluyendo los modelos compactos obtenidos en el

capítulo 4. Finalmente, se extraen ordenadamente las conclusiones más importantes de este trabajo en la sección correspondiente.

En el apartado de Publicaciones se listan todos los resultados de comunicaciones a congresos y publicaciones en revistas incluidas en el índice JCR obtenidas durante el desarrollo de este trabajo.

En el Apéndice se han reunido todos los contenidos que por su nivel de detalle hemos extraído de los capítulos anteriores. Durante la lectura del presente documento se realizarán repetidas referencias a los contenidos incluidos en el apéndice.

CONCLUSIONES

Uno de los resultados más importantes presentados en esta tesis está relacionado con la demostración de que los modelos compactos codificados en Verilog-A son una alternativa eficaz para sustituir los modelos basados en código fuente. La implementación en Verilog-A presenta innumerables ventajas como la facilidad de codificación, su naturaleza multiplataforma, etc. A lo largo de los cinco capítulos que componen este trabajo se desarrollan diferentes modelos compactos para un conjunto variado de dispositivos electrónicos, tanto transistores como sensores, en particular, sensores de corriente. Todos ellos han sido implementados en diferentes simuladores de circuitos, principalmente en Verilog-A, para estudiar el comportamiento de circuitos electrónicos basados en los dispositivos modelados. Los modelos introducidos incorporan las relaciones entre tensiones y corrientes, los efectos térmicos, las contribuciones de los efectos capacitivos, y también las fuentes de ruido.

DISEÑO E IMPLEMENTACIÓN EN FPGA DE UN SISTEMA DE CONTROL DE ORIENTACIÓN PARA UN SIMULADOR DE VUELO SATELITAL

JOSE FRANCISCO OSORIO JIMENEZ – México DF - 2013

RESUMEN

Los alcances de la temática abordada en esta tesis al momento son vastos y excederían el esquema tradicional de un trabajo de titulación. Sin embargo, como primera aproximación en la integración de un sistema de simulación y validación en tierra, se muestran resultados que sin bien, aún están sujetos a una serie de ajustes y cambios, representan un avance importante en el desarrollo de tecnología de estabilización y control de apuntamiento en México, máxime donde actualmente a nivel mundial sólo países tan selectos como la India y algunos otros de la Unión Europea han tenido gran injerencia, y que invierten varios miles de dólares en su investigación y desarrollo cada año.

El contenido de la tesis está estructurado de la siguiente forma:

Capítulo 1: Se abordan las generalidades de las plataformas ADCS o de simulación satelital, sus principales características de simulación de vuelo, acoplamiento, movimiento, instrumentación, entre otras.

Capítulo 2: Trata de las generalidades las plataformas de desarrollo FPGA, sobre su arquitectura, su configuración, algunos de sus componentes y diferencias de fabricación, así como de los fabricantes de algunas plataformas de desarrollo que las contienen.

Capítulo 3: Se exponen los elementos básicos para el estudio y análisis del modelo matemático de la plataforma de simulación para la validación de esquemas de control, y también, se plantea el desarrollo de la ecuación de movimiento de la MSA con base en los elementos de estudio expuestos, esto es la dinámica del sistema.

Capítulo 4: Se propone una discusión para abordar las diferentes estrategias y propuestas de implementación e integración del sistema de simulación para la validación de esquemas de control y orientación, y se resumen en dos vertientes: la primera propone integrar toda la arquitectura de cómputo sobre un solo dispositivo FPGA; y la segunda propone utilizar las técnicas HIL para simular el coprocesamiento con ayuda de una computadora externa, permitiendo una ejecución del sistema como si estuviese totalmente integrado a bordo.

Capítulo 5: Se abordan los detalles del primer segmento del sistema que será implementado en dos bloques principales, con base en las técnicas HIL, el cual comprende el módulo de adquisición de datos (AD) y el control de actuadores, los cuáles han sido embebidos en la plataforma FPGA, se explican las configuraciones realizadas, los diagramas de flujo de los algoritmos implementados, y se expone una revisión de todo el segmento y sus elementos que lo componen.

Capítulo 6: Se explican los detalles del segundo segmento del sistema, e cual consiste en la implementación de los algoritmos de determinación y corrección de la orientación (EKG y TRAID), y la ley de control que con base en ella podrán determinarse los ciclos de trabajo de las ruedas inerciales y el sentido de giro de las mismas.

Capítulo 7: Una vez realizada la verificación de los componentes y que hayan sido validados en integrados, en este capítulo se muestran los resultados experimentales de las pruebas realizadas y sus resultados. Se discuten de forma breve estos resultados obtenidos

a partir de las gráficas obtenidas del comportamiento del sistema completo en funcionamiento.

Capítulo 8: Finalmente, en este capítulo, se realiza una discusión de los resultados obtenidos y de los esperados, de los alcances de esta primera aproximación, así como de las recomendaciones que se hacen para la continuación del trabajo, así como un panorama de lo que sería el trabajo futuro.

CONCLUSIONES

En la propuesta de esta tesis se plantearon e indicaron objetivos, donde el principal fue diseñar, implementar y validar operativamente un sistema de control de orientación en tres ejes sobre un simulador para pruebas de control para satélites pequeños que llamamos Mesa Suspendida en Aire (MSA), la cual es una plataforma de pruebas basada en un cojinete de aire tipo mesa. Este sistema es un sistema de validación en tierra de un ADCS de un satélite, donde éste realiza las funciones de determinación de la orientación y el control del vehículo en el espacio y debe cumplir ciertos requisitos que lo caracterizan en función de la misión del satélite; por su parte, la MSA tiene como finalidad ser una plataforma abierta y didáctica, que permita validar diversos sistemas de control sobre una plataforma instrumentada con componentes genéricos y simular condiciones de no fricción.

En este capítulo se exponen las conclusiones sobre la integración que se hizo del sistema, y también se darán recomendaciones para el trabajo futuro teniendo en cuenta la experiencia obtenida en el desarrollo de esta primera aproximación del sistema de simulación satelital para la validación de esquemas de control.

Por lo tanto:

Se integró una plataforma de simulación satelital para validar la lógica y hardware del ADCS (sensores, actuadores, computadora de vuelo), software (algoritmos de procesamiento) y la instrumentación necesaria para realizar las maniobras de control, así como demás accesorios necesarios para el funcionamiento del sistema.

Se utilizó la técnica hardware in the loop (HIL) para la aceleración del desarrollo del sistema de simulación satelital, y con ello se realizaron pruebas de validación de cada uno de los bloques que integran el sistema: por un lado el segmento implementado en la plataforma FPGA, que se encarga de la adquisición de datos del magnetómetro, acelerómetro y giróscopo, así como de recibir los comandos de control y escribirlos en los registros del núcleo PWM para las maniobras de control; y por otro, el segmento implementado en MATLAB que ejecuta sobre una PC externa a la plataforma, los algoritmos de procesamiento TRIAD, EKF y CONTROL, gestionados por un programa principal.

Se resolvieron los problemas de comunicación inalámbrica entre los dos segmentos que componen el sistema de simulación satelital, mediante la incorporación de radios módem, ajustando la velocidad y formato de envío entre los diversos puertos de las plataformas FPGA y PC, teniendo una velocidad de transmisión de 9600 bps.

Se realizaron pruebas parciales para la validación de la instrumentación de la MSA, realizando la configuración de los sensores para la lectura de datos, así como la activación de un filtro integrado pasa bajas para la reducción del ruido en la señal del sensor giróscopo, y también se validó el desempeño de las ruedas inerciales logrando calcular la velocidad angular de cada una de ellas en función del ciclo de trabajo inducido, así como el momento angular generado de acuerdo a esa velocidad.

El modelo dinámico de la MSA se retomó de un trabajo previo con el fin de obtener la ecuación de movimiento del sistema, que en el caso de un ADCS es una dinámica de la orientación, donde se relaciona la velocidad angular del sistema con los momentos de las fuerzas que actúan sobre él. Esta relación viene dada por la velocidad angular mencionada y los tensores de inercia que se refieren a distribución de masa en los ejes principales del sistema, que como se expuso en el Capítulo 3, el centro de masa del sistema es coincidente con el centro del sistema de referencia, esto nos permitió estudiar el cuerpo en su movimiento rotacional y simplificar el análisis. No obstante, como se ha dicho, desarrollar completamente la ecuación de movimiento permitirá mejorar el esquema de control y estabilización, así como permitirá tomar el análisis y realizar una simulación virtual del mismo, ampliando el espectro de aplicaciones y alcances del sistema en la investigación.

Mediante el uso de herramientas de software de Xilinx, se desarrollaron núcleos personalizados y se hizo uso de núcleos propietarios para realizar las funciones siguientes: núcleos personalizados PWM para la interpretación de comandos de ciclo de trabajo; uso de núcleos propietarios para el control y ejecución de interrupciones para la lectura de datos de los sensores magnetómetro, acelerómetro y giróscopo, y uso del núcleo I2C como protocolo de comunicación con estos sensores.

Con base en las herramientas de software Xilinx se generó una arquitectura de cómputo embebida haciendo uso del microprocesador Microblaze, el cual se encarga de gestionar los núcleos periféricos y además incluye rutinas en lenguaje de alto nivel para la configuración de la IMU, adquisición de datos de los sensores de navegación, envío de datos de los sensores con el formato adecuado hacía la PC externa y recepción de los comandos de control desde la PC externa.

Se implementaron en lenguaje de programación MATLAB los algoritmos siguientes: TRIAD para la determinación de la orientación del simulador satelital; EKF para la corrección y estimación de la orientación del simulador satelital, así como para la medición de la velocidad angular del sistema; y la ley de control que determina los ciclos de trabajo y el sentido de giro para cada una de las ruedas inerciales.

Se realizaron análisis y discusiones sobre las mejores estrategias de trabajo para realizar la implementación del sistema, así como reducir el tiempo de desarrollo del mismo, con lo que justificó y se argumentó el uso de las técnicas HIL, así como las herramientas con las que se contaba para integrar el sistema, sentando las bases para la continuidad del trabajo sobre la misma línea.

Se realizaron pruebas experimentales suficientes con las que se validó, verificó y ajustó el funcionamiento del sistema, teniendo como primera aproximación estabilización en los tres ejes ante una perturbación del mismo, y aunque las maniobras de apuntamiento aún no pueden lograr la inclinación en los tres ejes debido a las limitaciones en el hardware e instrumentación, como el tamaño de las ruedas, no se cuenta con bobinas de par magnético, entre otras, se obtuvieron resultados positivos. El comportamiento gráfico que se obtuvo de las diversas pruebas experimentales muestra una curva amortiguada y la disminución del movimiento y velocidad angular del sistema hasta su estabilización, y posteriormente el apuntamiento dentro de una ventana en la cual permanece el sistema teniendo un margen de error entre 10 y 15 grados.

DESARROLLO DE UN SIMULADOR DE CIRCUITOS DIGITALES INTEGRADO EN UN COMPILADOR DE DIAGRAMAS ASM

David Sanz Cabreros - Valladolid – España 2017

RESUMEN

El proyecto se centra en el diseño y desarrollo de un simulador de circuitos digitales para su integración en un compilador de diagramas ASM, capaz de simular un circuito digital descrito en lenguaje Verilog, mostrando al usuario tras su ejecución el resultado correcto o incorrecto de una serie de verificaciones descritas en el diagrama.

El objetivo principal es el desarrollo en lenguaje C++ de un simulador integrable en un compilador de diagramas ASM, capaz de a partir de un archivo con extensión. vdo, simular un circuito digital descrito en lenguaje Verilog y realizar una serie de verificaciones.

El objetivo secundario es desarrollar un interfaz de usuario empleando librerías Qt en el cual esté integrado el simulador y que permita al usuario seleccionar el archivo de simulación deseado, comprobar el proceso de simulación y obtener un archivo con extensión .txt con el resultado final de la simulación.

CONCLUSIONES

En los capítulos anteriores se ha descrito por completo toda la información relativa al proyecto, tanto teóricamente como mostrando el desarrollo de la programación del simulador.

Además, se han enumerado las diferentes especificaciones marcadas para la realización del proyecto, las cuales marcaban el objetivo final del simulador: obtener una

primera versión de un simulador de circuitos digitales para diagramas ASM++, integrado dentro de un interfaz de usuario, y desarrollado en código C++ empleando las librerías Qt, con el objetivo de poder ser integrable posteriormente en el compilador de diagramas ASM++.

Como se comprueba en el desarrollo de la programación del simulador, todas las especificaciones marcadas se cumplen correctamente:

Se dispone de una primera versión de un simulador de circuitos digitales descritos con diagramas ASM++ fácilmente integrable en un compilador de diagramas ASM++.

El simulador tiene como punto de partida un archivo con extensión. vdo, y como salida un archivo de texto .txt.

El simulador es capaz de obtener el resultado de la simulación del diseño descrito en la figura 28.

El simulador es capaz de interpretar circuitos descritos en lenguaje Verilog.

El simulador dispone de un interfaz gráfico que permita al usuario interaccionar con él.

DISEÑO E IMPLEMENTACIÓN DE UN TEMPORIZADOR PROGRAMABLE EN FPGA UTILIZANDO LENGUAJE VERILOG

Daniel Valgañon Medecigo – Ciudad de México 2017

RESUMEN

En este trabajo se pretende implementar un temporizador con características similares a las del circuito integrado 8253. Para su diseño se utilizará un dispositivo FPGA

por sus características que nos permiten reprogramarlo incluso ya instalado en campo y la capacidad de integración con otro sistema en un mismo dispositivo. Se trabajará el diseño y programación del dispositivo en el lenguaje de descripción de hardware Verilog debido a una de las grandes ventajas que presenta este lenguaje: la realización de simulaciones previas a la implementación en físico. Evitando, de esta manera, cualquier mal funcionamiento y/o avería del dispositivo FPGA.

Esta tesis consta de cuatro capítulos en los que se expone el desarrollo de un temporizador implementado en un FPGA utilizando lenguaje Altera.

Capítulo I: En el capítulo se aborda una breve introducción a los temporizadores, a los FPGAs y al lenguaje de descripción de hardware Verilog. También se incluye una pequeña investigación de trabajos similares realizados.

Capítulo II: En este capítulo se aborda el desarrollo del diseño del temporizador realizado y las simulaciones realizadas de los módulos del diseño en funcionamiento.

Capítulo III: En esta sección se incluyen leos resultados de la compilación y diversas aplicaciones realizadas para poner a prueba el diseño realizado.

CONCLUSIONES

Se logró cumplir con el objetivo general y los objetivos específicos planteados inicialmente realizando un circuito temporizador funcional capaz de trabajar con diferentes configuraciones.

Al terminar el trabajo el resultado final resultó ser: Diseñar un temporizador digital basado en el circuito integrado 8253 de la familia Intel e implementarlo en la tarjeta

DE2 de Altera con diversas aplicaciones con la finalidad de comprobar la funcionalidad del diseño realizado.

La tecnología FPGA permite realizar pruebas de diseños con eficiencia además de brindar muchas facilidades para realizarlas.

Los circuitos temporizadores permiten una inmensa cantidad de tareas que necesitan ser sincronizadas o, de alguna manera, limitadas por una señal en el tiempo.

Las características de los FPGA hacen que a pesar de usar los mismos módulos repetidamente estos no funcionen con la misma frecuencia máxima de operación y que se tuvieran que corregir propuestas del diseño por el funcionamiento erróneo se ocasionaba.

El conocimiento de electrónica digital facilito el aprendizaje del diseño en FPGA y de igual manera el manejo de lenguaje C hizo que entender Verilog HDL fuera algo sencillo. Por lo que en lo que más se centró la investigación del lenguaje de programación fue en las palabras reservadas, la manera de crear las instancias, y los módulos Always.

Al aprender e investigar sobre el circuito integrado 8253, fue más fácil entender la conexión de dispositivos en un sistema mínimo. Aunque realizar la conexión con el procesador Lagarto no es parte de este trabajo, es importante entender las conexiones de los sistemas computacionales ya que de esta manera se pueden realizar las modificaciones pertinentes según las necesidades del mismo.

El aprendizaje de dispositivos programables (GAL, PAL, FPGA, etc.) son una buena opción para realizar diseños más complejos de circuitos y sistemas digitales sin la necesidad de las conexiones complejas que pueden resultar de usar las compuertas utilizadas en los cursos de circuitos digitales y electrónica digital.

Es importante saber en qué casos es mejor usar cada tipo de dispositivo y tecnología. Hay aplicaciones en las que es mejor usar un microcontrolador, otras en las que una GAL es mejor.

Considero que el aprendizaje y experimentación con dispositivos programables es algo que debería implementarse en escuelas de ingeniería con afín a la electrónica. Y que empezar a desarrollar diseños propios es un gran inicio al progreso de las tecnologías. También, al utilizar estas tecnologías, abre un gran abanico de soluciones a posibles problemáticas además de las ya existentes.

• Durante la realización de este proyecto se tuvieron diversas problemáticas. Empezando por tener que aprender de 0 el lenguaje Verilog y el entorno de trabajo de Quartus II. Además de ser la primera vez que trabajaba con FPGA. Afortunadamente con la ayuda de mi asesor técnico y las múltiples lecturas realizadas pude acercarme sin mayor dificultad al diseño en FPGA utilizando Verilog HDL. Cuando estuve realizando la lectura y comprensión del funcionamiento del circuito 8253 no fue el idioma el problema, sino que los diagramas de tiempo no me eran claros en algunas condiciones de funcionamiento por lo que terminé leyendo las hojas de especificaciones del sucesor del 8253, el 8254. En las hojas de especificaciones de este último dispositivo se incluían diagramas de tiempo más detallados y fáciles de comprender. En la programación del funcionamiento fue donde más batallé pues, aunque entendía lo que tenía que suceder me costó un poco de trabajo pasar de la comprensión de lo que sucedía en la señal a cómo lograr dicha respuesta con dispositivos lógicos (Flip-Flops, compuertas lógicas entre otros). Tras largas horas de pensar di con algunas soluciones, pero terminando usando las más sencillas. Finalmente, durante las pruebas y aplicaciones del diseño se tuvieron que desarrollar los pequeños programas para la simulación que, aunque sencillos de realizar,

se vuelven laboriosos dependiendo de que tanto se quiera abarcar en una sola simulación. Añado, a pesar de que no fue realmente una dificultad, los problemas que se tuvieron con la tarjeta de desarrollo. Ésta al ya tener tiempo de uso tenía muy sensibles los push buttons lo que generaba que una simple pulsación la detectara como si fueran varias. Al final opté por usar mejor los DPDT switches que, aunque fuera más lento no me generaban rebote en la señal. Más que muchas problemáticas en el desarrollo de este proyecto me llevo muchas enseñanzas y conocimientos que pondré en práctica en un futuro y que espero me sirvan para muchas cosas.

CAPITULO III

MATERIALES Y MÉTODOS

3.1. MATERIALES

3.1.1. HARDWARE

3.1.1.1. Ordenador para programación (laptop)

- Modelo: DELL LATITUDE
- Procesador: Intel(R) Core (TM) i7-2430 2.40GHz
- Memória instalada (RAM): 8.00GB de RAM.
- Tipo de sistema: Sistema Operativo de 64 bits Windows 10

3.1.1.2. Servidor Linux

- Modelo: PowerEdge 2850 de Dell
- Procesador: Intel® Xeon™ 3,6 GHz
- Memória instalada (RAM): 64.00GB de RAM.
- Tipo de sistema: Sistema Operativo RedHat

3.1.1.3. FPGA – DE2-115 ALTERA

- Procesador Cyclone IV EPC4CE115
- 2 puertos Ethernet
- 128MB SDRAM

- 2MB SRAM
- 8MB Flash
- Puerto de video VGA DAC (Alta velocidad triple DACs)
- Conector RS232 DB-9

3.1.2. SOFTWARE

- Sistema Operativo de 64 bits Windows 10.
- Microsoft Word 2013 profesional.
- ICARUS VERILOG COMPILER
- CADENCE INCISIVE RTL COMPILER && CADENCE SIMVISION
- ALTERA QUARTUS – FPGA

3.2. DISEÑO, NIVEL Y TIPO DE LA INVESTIGACION

3.2.1. DISEÑO DE LA INVESTIGACIÓN

Esta investigación es experimental ya que se utilizará experimentos y los principios encontrados en el método científico. Los experimentos pueden ser llevados a cabo en los laboratorios de microelectrónica. Estos generalmente involucran un número relativamente pequeño de personas y abordan una pregunta bastante enfocada. Los experimentos son más efectivos para la investigación explicativa y frecuentemente están limitados a temas en los cuales el investigador puede manipular la situación en la cual las personas se hallan.

3.2.2. NIVEL DE LA INVESTIGACIÓN

El nivel de investigación se refiere a la profundidad del conocimiento que se busca lograr con la investigación, por tanto, el nivel de la presente investigación es exploratoria, señalando que las investigaciones exploratorias buscan abrir nuevos caminos en el desarrollo del conocimiento humano. Y la presente investigación siendo el diseño e implementación de un temporizador para un sistema en chip (SOC) en lenguaje VERILOG.

3.3. POBLACIÓN Y MUESTRA DE LA INVESTIGACIÓN

3.3.1. POBLACIÓN

Siendo la población el conjunto de las entidades, o cosas respecto a las cuales se basa las conclusiones de una investigación, para nuestro caso la población está definida por todos los microcontroladores, microprocesadores o sistemas que requieran un módulo temporizador interno.

3.3.2. MUESTRA

Se define la muestra como parte que se estudia y es representativa de la población, es decir un segmento que tiene las características y propiedades de la población, por tanto, se considerará como un muestreo no probabilístico y estará conformada por el sistema FPGA, es decir que en este sistema se desarrollaran las pruebas. La muestra no servirá para hacer generalizaciones, pero sí para el estudio exploratorio, se ha elegido a los individuos utilizando diferentes criterios relacionados con las características de la investigación y está determinado por el autor.

3.4. UBICACIÓN Y DESCRIPCION DE LA INVESTIGACIÓN

3.4.1. UBICACIÓN

La investigación, tanto pruebas y diseño se desarrolló en Campinas, Sao Paulo, Brasil en el laboratorio personal, propio del autor.

3.4.2. TECNICAS E INSTRUMENTOS DE RECOLECCION DE DATOS

La recolección de datos se refiere a cómo y qué medios se usan para la obtención de la información que será de utilidad para la corroboración de nuestras hipótesis, por lo tanto, resumimos este apartado de la siguiente forma:

Tabla 3.1 Técnicas e instrumentos para recolección de datos

Técnicas	**Instrumentos**
Consultas bibliográficas y de bases de datos.	Papers, foros, blogs, libros, videotutoriales y más fuentes de información publicadas en Internet. Los más importantes mostrados en ANEXOS.
Observación de pruebas finales en software	Capturas a la pantalla, apuntes simulaciones y fotos.
Observación de pruebas finales en hardware	Uso del FPGA con capturas a la pantalla, apuntes de simulaciones y fotos.

Elaboración propia.

3.5. IMPLEMENTACIÓN

El temporizador de 8 bits es un módulo de contador simplificado de propósito general, que incluye un contador bidireccional, un bloque de generación de interrupciones, un modo de conteo de pulsos de entrada o modo de reloj externo; también proporciona la generación de una señal modulada en PWM y una señal de activación (Trigger) generada según los eventos del temporizador.

3.5.1. VISIÓN GENERAL

El temporizador contiene un contador que tiene 2 modos de operación, contador ascendente o descendente, el valor inicial se define con un registro de configuración de valor inicial, del mismo modo el máximo o mínimo valor permitido, existen registros de valor de coincidencia que se pueden ajustar fácilmente para generar señales de interrupción periódicamente de acuerdo a la condición que un valor predefinido es igual al valor actual del contador.

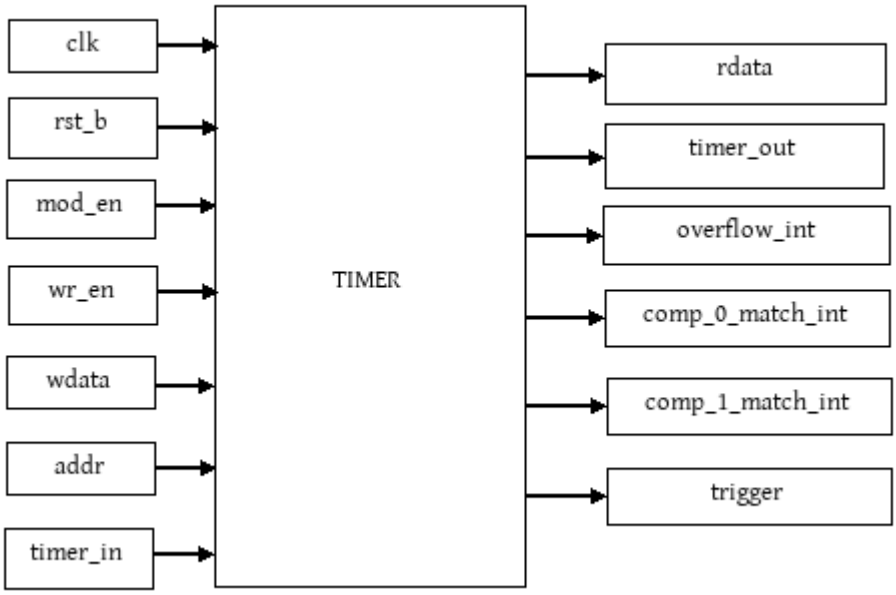
La señal PWM se puede generar usando el registro Count Match 0 y Count Match 1, la señal de disparo se puede activar de acuerdo con los mismos eventos para producir un pulso digital de 1 ciclo.

3.5.2. DIAGRAMA DE PINES

El temporizador tiene una interfaz para el acceso a los registros, las entradas de reloj y reinicio, también tiene la señal timer_in donde puede conectarse una señal de reloj externa o una fuente de pulsos asíncrono.

La interfaz de salida incluye los datos de lectura (rdata) que proporcionan un valor de lectura de acceso de registro, 3 señales de interrupción, timer_out es la salida para la señal PWM y el disparador trigger se comparte entre los 3 eventos: desbordamiento (overflow), coincidencia 0 (Match 0), coincidencia 1 (Match1).

Figura 3.1 Diagrama de Pines



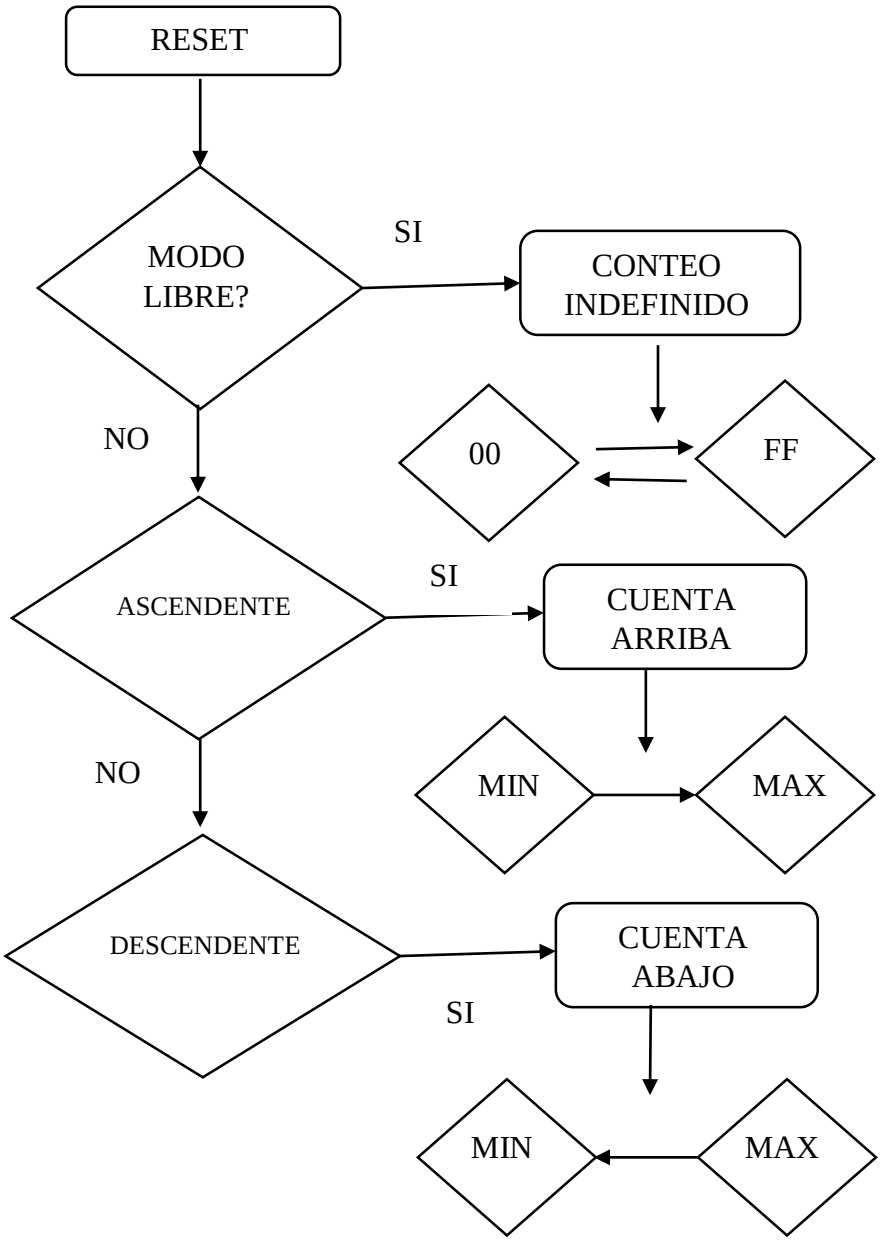
Elaboración Propia

3.5.3. DIAGRAMAS DE FUNCIONAMIENTO

A continuación, se presenta los diagramas que ilustran el funcionamiento de cada subcomponente, los criterios de decisión y ejecución de tareas durante el transcurso del tiempo y en base a las señales en la interfaz de entrada y controles proporcionados mediante lo registros de configuración.

3.5.3.1. CONTADOR

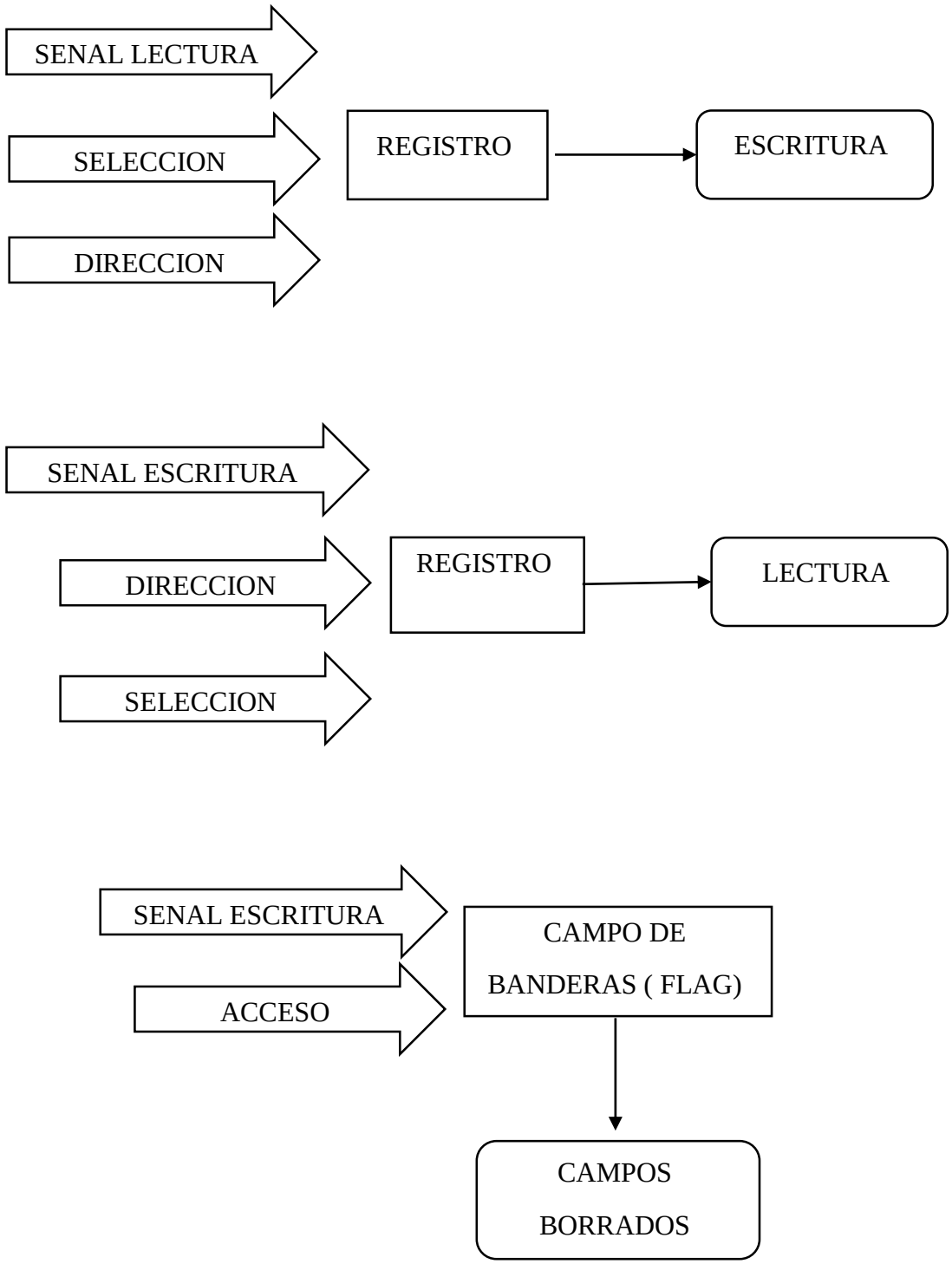
Figura 3.2 Diagrama de funcionamiento del contador



Elaboración Propia

3.5.3.2. REGISTROS

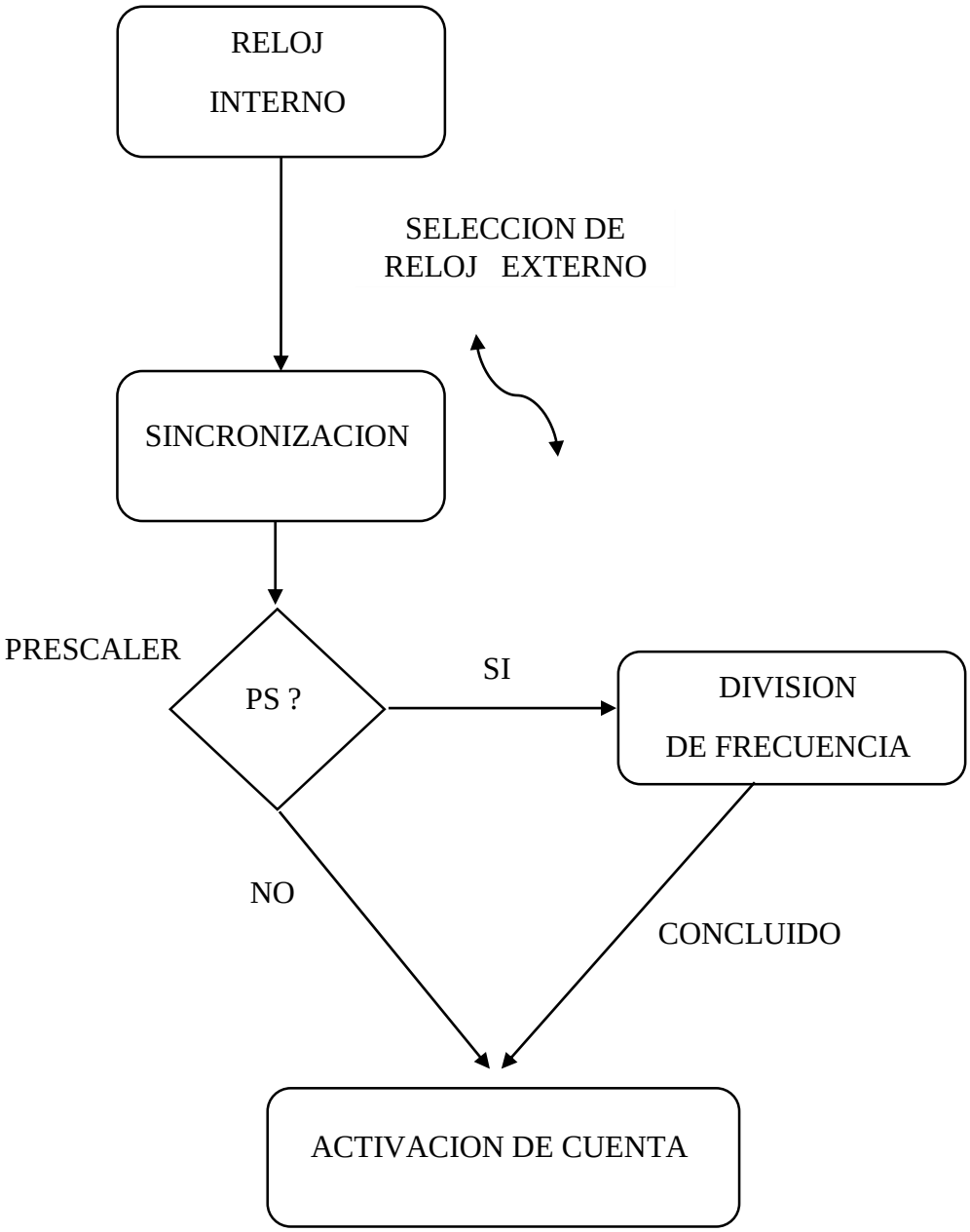
Figura 3.3 Diagrama de acceso a registros



Elaboración Propia

3.5.3.3. BLOQUE ENTRADA

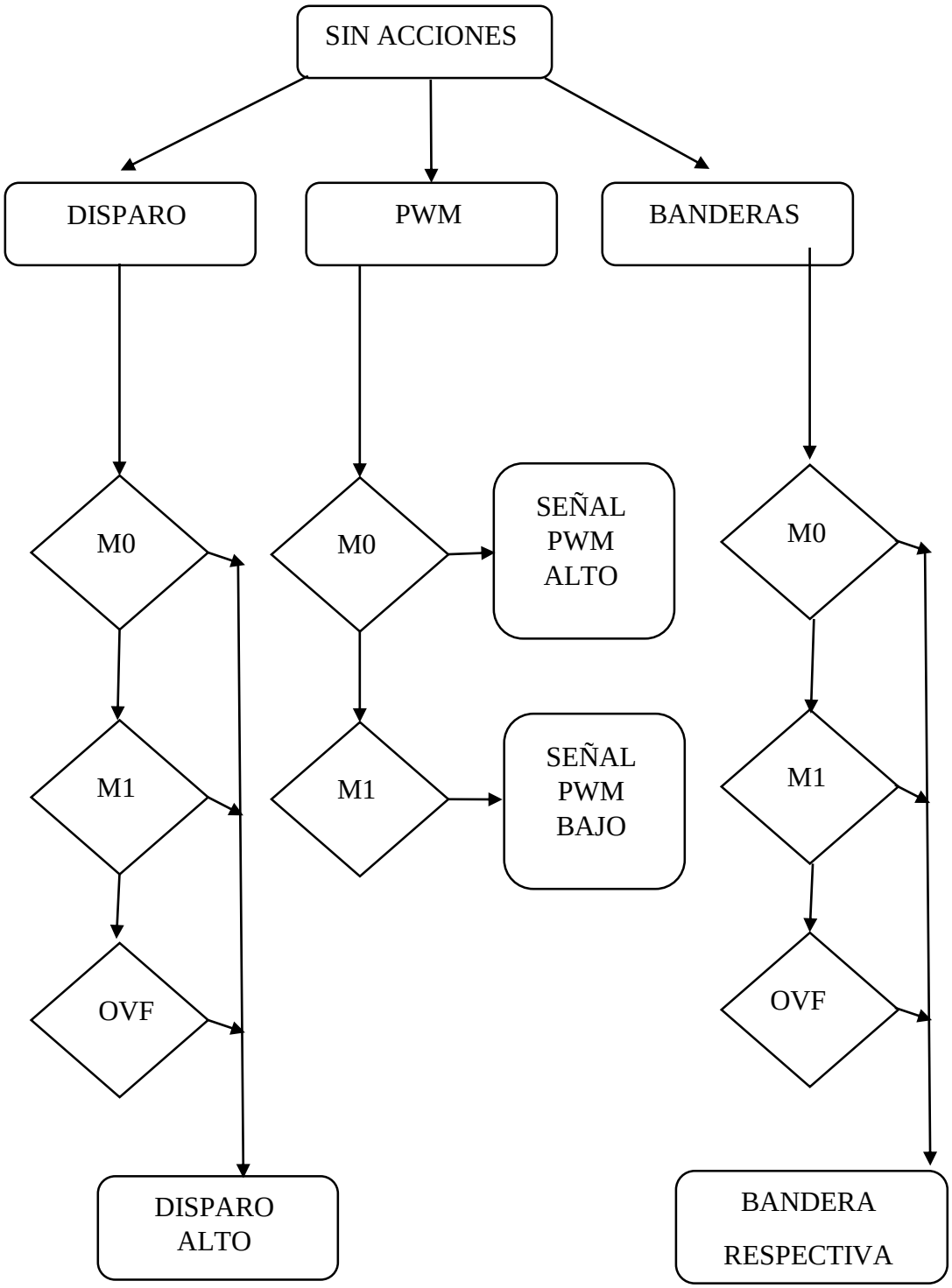
Figura 3.4 Diagrama de funcionamiento, bloque de entrada



Elaboración Propia

3.5.3.4. BLOQUE DE SALIDA

Figura 3.5 Diagrama de funcionamiento, bloque de salida

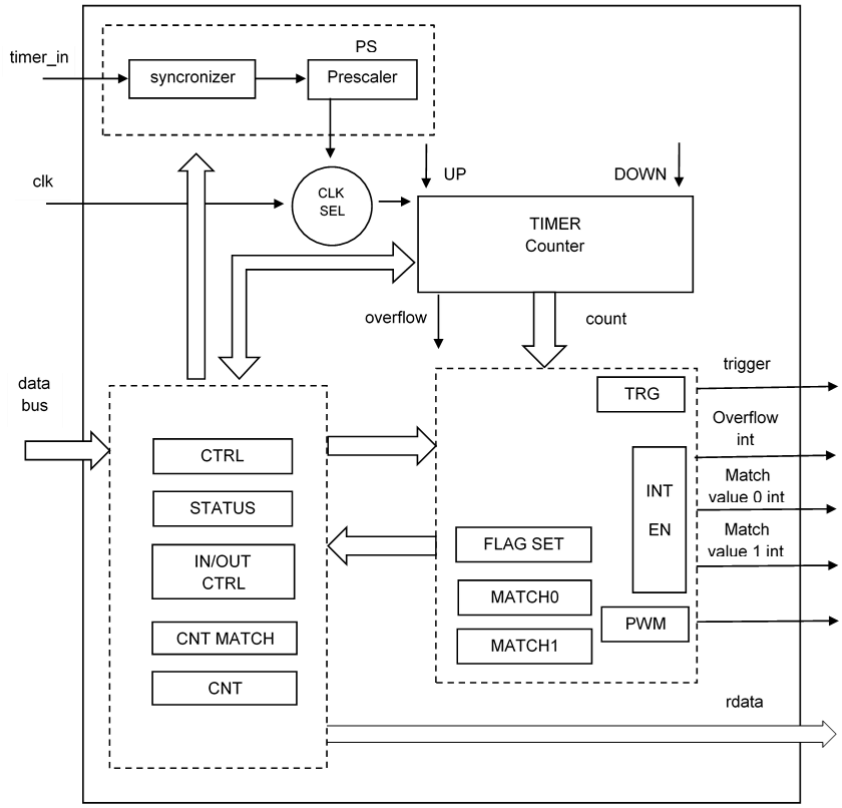


Elaboración Propia

3.5.4. DIAGRAMA DE BLOQUES

El siguiente diagrama muestra las características del componente en función de las partes involucradas en su funcionamiento, se puede apreciar el bloque de contador TIMER, el bloque de registros con los registros CTRL, STATUS, etc. El bloque de entrada con el sincronizador y el Prescaler, el bloque de salida que genera las interrupciones, la señal PWM y de disparo Trigger.

Figura 3.6 Diagrama de bloques



Elaboración propia.

3.5.4.1. Contador

El contador es de 8 bits y bidireccional, funciona en modo ascendente o descendente, admite el modo de conteo de ejecución libre (Free Run) y los modos de conteo restringidos a los valores MIN a MAX.

3.5.4.2. Bloque de entrada

Incluye detectores de borde y división de frecuencia.

- Prescaler: puede dividir la frecuencia de entrada por factores de 2,4,8,16,32,64,128.

- Sincronizador: Toma la señal de entrada y lo pasa por dos Flip Flops para que los bordes de subida bajada queden sincronizados con el reloj clk del temporizador.

3.5.4.3. Bloque de salida

Incluye las señales de interrupción, PWM y generación de disparo (trigger).

3.5.4.4. Bloque de registros

El bloque de registros interno contiene todo el campo de registros usados en el temporizador en una sola jerarquía.

3.5.5. CARACTERÍSTICAS

3.5.5.1. Pre-escala

Una señal de reloj externa se puede dividir por los factores 2,4,8,16,32,64,128, para obtener periodos más largos de reloj, de esta forma los eventos pueden suceder en tiempos igualmente ampliados.

3.5.5.2. Contador de pulsos de entrada

Se puede usar una señal de entrada como disparador de cuenta, el flanco de subida o bajada queda seleccionado mediante el registro CTRL_IN.

3.5.5.3. Soporte de reloj externo

La señal del reloj externo se puede conectar a la entrada timer_in y se puede usar para aumentar o disminuir el valor del contador, la frecuencia se puede dividir de acuerdo con el valor de Prescaler definido en el registro CTRL_IN.

3.5.5.4. Generación de interrupciones

Se pueden generar interrupciones para los siguientes eventos del temporizador:

- Overflow: La señal de interrupción se puede generar utilizando la configuración de registro de control OVF_EN habilitando el bit, cada vez que haya una transición desde el valor máximo (en 0xFF) al valor mínimo del contador 0x0 se generara una señal de interrupción se será mantenida hasta que el FLAG guardado en el registro STATUS sea borrado vía software.

- Interrupción en Match 0 y 1: las señales de interrupción se generan utilizando la configuración del registro de control CTRL en los bits CTN_M1_EN y CTN_M2_EN, cada coincidencia entre el valor real del contador y el valor de coincidencia configurado en los registros CNT_M1 y CNT_M2 dispara la señal de interrupción respectiva.

3.5.5.5. Inicialización del contador

El valor inicial del contador es 0 por defecto, pero se puede reemplazar para un valor definido por el usuario, escribiendo en el registro CNT_INT un valor particular antes de habilitar el funcionamiento del temporizador.

3.5.5.6. Contador modo inverso

La dirección del contador está definida por el bit de configuración CTRL del registro de control, el modo predeterminado es el modo de cuenta ascendente donde MODE = 0, si este bit está establecido MODE = 1 la dirección de conteo se invertirá.

Se espera que esta configuración se realice durante el modo sin operación, el bit de registro de control está con el valor predeterminado START = 0.

3.5.5.7. Generación de disparo (Trigger)

El contador genera una señal de disparo de acuerdo con los eventos de desbordamiento o coincidencia 0/1, el evento se selecciona a través del registro CTRL_OUT.

3.5.6. DESCRIPCION DE SEÑALES

Las señales se detallarán con las tablas presentadas en tablas, separándose por las señales de entrada y salida.

Tabla 3.2 Señales de entrada

Nombre	Tamaño (bits)	Descripción	Observación
clk	1	Entrada de reloj del sistema	
rst_b	1	Señal de reset o reinicio	Activa baja

addr	6	dirección de registro	
mod_en	1	Selecciona el módulo, habilita el acce a los registros.	Activo Alto
wr_en	1	1: módulo de acceso para escritura 0: módulo de acceso para lectura	
timer_in	1	Entrada de reloj y pulsos externos	
wdata	8	Datos para escritura de registro	

Elaboración propia.

Las señales de interface de entrada incluyen las señales de reset y de reloj, las señales de acceso a registradores internos como addr mod_en wr_en addr y wdata. La interface incluye la señal de entrada de impulsos externos o reloj externo timer_in.

Para tener acceso a un registrador del temporizador se levanta la señal wr_en en alto junto con module_en la dirección deseada vía addr y el dato a ser escrito vía wdata, para la lectura se mantiene la señal wr_en en bajo y se lee los datos mediante la señal de salida rdata descrita a continuación.

Tabla 3.3 Señales de salida.

Nombre	**Tamaño (bits)**	**Descripción**	**Observación**
rdata	8	Entrada de reloj	
overflow_int	1	Interrupción del evento d desbordamiento del temporizador	Activo Alto
comp_0_match_int	1	Comparador de coincidencia en even 0	Activo Alto

comp_1_match_int	1	Comparador de coincidencia en even 1	Activo Alto
timer_out	1	Salida PWM	
desencadenar	1	Salida de Trigger	

Elaboración propia.

3.5.7. MAPA DE MEMORIA Y DEFINICIÓN DE REGISTROS.

El módulo temporizador tiene 6 registros de configuración listados y resumidos en la siguiente tabla

Tabla 3.4 Resumen de registros

Dirección de registro	**Nombre de registro**	**Descripción**	**Anchura (bits)**	**Acceso**	**Restablecer valor**
0x0	CTRL	Registro de control general	8	R / W	0
0x1	CTRL_IN	Registro de Control de entrada	8	R / W	0
0x2	CTRL_OUT	Registro de Control de salida	8	R / W	0
0x4	STATUS	Registro de estado del temporizador	8	R / W	0
0x8	CNT_INIT	Registro de inicialización del temporizador	8	R / W	0
0x9	CNT_MIN	Registro de configuración de valor mínimo del contador	8	R / W	0

0xA	CNT_MAX	Registro de configuración de valor máximo del contador	8	R / W	0
0xB	CNT	Valor actual del contador	8	RO	0
0xC	CNT_M1	Registro de configuración de del valor de coincidencia 0	8	R / W	0
0xD	CNT_M2	Registro de configuración de del valor de coincidencia 1	8	R / W	0

Elaboración propia.

3.5.8. DESCRIPCIÓN DE REGISTROS

3.5.8.1. Registro de Control (CTRL)

El registro de control configura las características principales como; modo de operación, fuente de reloj y tiene el bit de inicio para habilitar la operación del contador. Descritos a continuación:

Tabla 3.5 Diagrama de registro de control

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R W	FREE	M1_INT	M0_INT	OVF_INT	CLK_SEL		MODO CNT	START
RESET	0	0	0	0	0	0	0	0

Elaboración propia.

Y la descripción de estos campos:

Tabla 3.6 Descripción de campos del registro de banderas

Campo	Función
START	Inicia la operación del contador, Se escribe 1 en este bit después de la configuración.
FREE	0 modo no libre, la operación de conteo considera los valores mínimo y máximo 1 forzar modo libre, desconsidera los valores mínimo y máximo en la cuenta, el contador cuenta desde 0 hasta el máximo valor 0xFF hexadecimal y retorna a 0x00 y continua así indefinidamente
CNT_MODE	0 conteo ascendente 1 conteo descendente
CLK_SEL	0 selecciona reloj del sistema para el contador 1 selecciona reloj externo para el contador
M0_INT	Habilita la generación de interrupción cuando hay una coincidencia entre el valor del contador y el valor configurado en el registro CNT_M0
M1_INT	Habilita la generación de interrupción cuando hay una coincidencia entre el valor del contador y el valor configurado en el registro CNT_M1
OVF_INT	Permite la interrupción de la coincidencia entre el valor CNT del registro del contador con el valor máximo del registro contador 0xFF: ESTADO OVF

Elaboración propia.

3.5.8.2. Registro de Control de entrada (CTRL_IN)

Establecido con el siguiente diagrama:

Tabla 3.7 Diagrama de control de entrada

Bits	7	6	5	4	3	2	1	0
R		PS						IN_EDGE
W								
Reset	0	0	0	0	0	0	0	0

Elaboración propia.

La descripción de estos campos se detalla en la siguiente tabla:

Tabla 3.8 Descripción de campos del registro de control de entrada.

Campo	Función
IN_EDGE	Selecciona el borde positivo o negativo de la señal de entrada timer_in para incrementar el contador, cuando es seleccionado el modo de contador usando señal de reloj externo y/o entrada de pulsos externo 0, borde positivo 1, borde negativo
PD	Valor del factor de división de frecuencia para reloj externo 000 - Divide por 1 001 - Divide por 2 010 - Divide por 4 011 - Divide por 8 100 - Divide por 16 101 - Divide por 32 110 - Divide por 64 111 - Divide por 128

Elaboración propia.

3.5.8.3. Registro de Control de salida (CTRL_OUT)

Establecido con el siguiente diagrama:

Tabla 3.9 Diagrama de control de salida

Bits	7	6	5	4	3	2	1	0
R		M1	M0	OVF			INV	PWM
W		TRG	TRG	TRG				
Reset	0	0	0	0	0	0	0	0

Elaboración propia.

La descripción de estos campos se detalla en la siguiente tabla:

Tabla 3.10 Descripción de campos del control de salida.

Campo	**Función**
PWM	El pin de salida timer_out alterna su valor de acuerdo con los valores MIN y MAX de la configuración de valores límite para el contador, cuando hay una coincidencia entre el valor del contador y el valor MIN se establece un nivel alto la salida, cuando hay una coincidencia entre el valor MAX y el valor del contador se establece un valor bajo en la salida. 0 generación de PWM desactivada 1 generación de PWM habilitada
INV	Invertir salida timer_out 0 la salida no está invertida 1 la salida está invertida
OVF TRG	Desbordamiento Cuando está habilitado, se genera un pulso con duración de un ciclo cuando hay un desbordamiento del contador (el valor del contador cambia de 0xFF a 0x00)
M0 TRG	Match 0: disparo de salida

	Cuando está habilitado, se genera un pulso con duración de un ciclo cuando hay una coincidencia entre el valor del contador y el valor en el registro CNT_M0
M1 TRG	Match 1: disparo de salida Cuando está habilitado, se genera un pulso con duración de un ciclo cuando hay una coincidencia entre el valor del contador y el valor en el registro CNT_M1

Elaboración propia.

3.5.8.4. Registro de estado (STATUS)

El registro de estado muestra indicadores para valores de coincidencia de contador y eventos de desbordamiento.

Tabla 3.11 Diagrama de estado

Bits	7	6	5	4	3	2	1	0
R						M1F	M0F	OVF
W1C*								
Reset	0	0	0	0	0	0	0	0

Elaboración propia.

Campos:

Tabla 3.12 Descripción de campos del diagrama de estado

Campo	**Función**
OVF	Bandera de desbordamiento 1 si hubo un desbordamiento desde la última vez que se borró este bit 0 si el contador de desbordamiento no se ha producido desde la última vez que se borró este bit. Acerca de borrar esta bandera: Al escribir 1 en esta posición se borra la bandera y las señales de interrupción se desactivan.

	Al escribir 0 en esta posición no se tiene ningún efecto.
M0F	Bandera de coincidencia Match Counter 0 1 el valor del contador ha coincidido con el valor de registro CTN_M0 desde la última vez que se borró este bit 0 el valor del contador no coincide o coinciden con el valor de registro CTN_M0 desde la última vez que se borró este bit. Acerca de borrar esta bandera: Al escribir 1 en esta posición se borra la bandera y las señales de interrupción se desactivan. Al escribir 0 en esta posición no se tiene ningún efecto
M1F	Bandera de coincidencia Match Counter 1 1 el valor del contador ha coincidido con el valor de registro CTN_M1 desde la última vez que se borró este bit 0 el valor del contador no coincide o coinciden con el valor de registro CTN_M1 desde la última vez que se borró este bit. Acerca de borrar esta bandera: Al escribir 1 en esta posición se borra la bandera y las señales de interrupción se desactivan. Al escribir 0 en esta posición no se tiene ningún efecto

*** W1C: Escribe 1 para borrar.**

Elaboración propia.

3.5.8.5. Registro de inicialización de contador (CNT_INIT)

El valor inicial del contador se puede definir utilizando este registro.

Tabla 3.13 Diagrama de inicialización de campo

Bits	7	6	5	4	3	2	1	0
R	INIT_VAL							
W								
Reset	0	0	0	0	0	0	0	0

Elaboración propia.

Campos:

Tabla 3.14 Descripción de diagrama de inicialización de campo.

Campo	Función
INIT_VAL	Valor inicial Este valor define el valor inicial para la operación del contador.

Elaboración propia.

3.5.8.6. Registro de valor mínimo del contador (CNT_MIN)

Registro de configuración de valor mínimo del contador.

Tabla 3.15 Diagrama de contador de valor mínimo.

Bits	7	6	5	4	3	2	1	0
R	MIN_VAL							
W								
Reset	0	0	0	0	0	0	0	0

Elaboración propia.

Campos:

Tabla 3.16 Descripción del diagrama de contador de valor mínimo

Campo	Función
MIN_VAL	Valor mínimo El mínimo valor que el contador puede asumir, cuando se alcanza el valor máximo, el contador vuelve a este valor a excepción de cuando se usa el modo de operación libre (Free run). En modo de cuenta descendente el contador cuenta hasta alcanzar este valor y luego retorna al valor MAX_VAL

Elaboración propia.

3.5.8.7. Registro de valor máximo del contador (CTN_MAX)

Registro de configuración del valor máximo del contador.

Tabla 3.17 Diagrama del contador de valor máximo

Bits	7	6	5	4	3	2	1	0
R	MAX_VAL							
W								
Reset	1	1	1	1	1	1	1	1

Elaboración propia.

Campos:

Tabla 3.18 Descripción del diagrama del contador de valor máximo.

Campo	Función
MAX_VAL	Valor máximo El contador sube hasta alcanzar este valor como máximo, a excepción de cuando se usa el modo de operación libre (Free run). En modo de cuenta descendente el contador vuelve a este valor luego de alcanzar el valor mínimo.

Elaboración propia.

3.5.8.8. Registro de valor actual del contador (CNT)

Este registro contiene el valor de conteo en el momento de lectura del registro.

Tabla 3.19 Diagrama de contador

Bits	7	6	5	4	3	2	1	0
R	CNT_VAL							
W								
Reset	0	0	0	0	0	0	0	0

Elaboración propia.

Campos:

Tabla 3.20 Descripción de campos del contador

Campo	Función
CNT_VAL	Valor del contador El valor actual del contador **Nota**: este registro solo se puede leer, no es posible modificar este valor

Elaboración propia.

3.5.8.9. Registro de valor de coincidencia 0, Match 0 (CNT_M0)

Contiene el valor de coincidencia 0.

Tabla 3.21 Diagrama de contador match 0.

Bits	7	6	5	4	3	2	1	0
R	CNT_MATCH							
W								
Reset	0	0	0	0	0	0	0	0

Elaboración propia.

Campos:

Tabla 3.22 Descripción del diagrama de contador match 0

Campo	Función
CNT_MATCH	Valor de coincidencia Este valor se utiliza para la comparación con el valor del contador y establecer el indicador M0F en el registro STATUS y la señal de interrupción correspondiente siempre que esté habilita la generación de interrupción mediante el bit M0_INT.

Elaboración propia.

3.5.8.10. Registro de valor de coincidencia 1, Match 1 (CNT_M1)

Contiene el valor de coincidencia 1.

Tabla 3.23 Diagrama de contador match 1

Bits	7	6	5	4	3	2	1	0
R	CNT_MATCH							
W								
Reset	0	0	0	0	0	0	0	0

Elaboración propia.

Campos:

Tabla 3.24 Descripción del diagrama de contador match 1

Campo	Función
CNT_MATCH	Valor de coincidencia Este valor se utiliza para la comparación con el valor del contador y establecer el indicador M0F en el registro STATUS y la señal de interrupción correspondiente siempre que esté habilita la generación de interrupción mediante el bit M1_INT.

Elaboración propia.

3.5.9. DESCRIPCION FUNCIONAL

El temporizador se puede configurar para los siguientes modos de operación:

3.5.9.1. Modo contador libre

El temporizador empieza la cuenta de 0x0 a 0xFF, al alcanzar 0xFF la cuenta vuelve a 0 y continua así indefinidamente.

Este modo de uso es el más simple ya que permite

3.5.9.2. Modo de conteo normal

El temporizador se configura para contar entre dos valores, uno superior MAX y uno inferior MIN, si el usuario desea inicializar en un valor diferente el contador puede usar el valor INIT, estas configuraciones son hechas mediante los registros respectivos.

3.5.9.3. Modo de conteo inverso

El temporizador se configura de la misma manera que el modo de conteo normal, pero el conteo comienza desde el valor máximo y continúa la cuenta regresiva hasta que alcanza el valor mínimo, cuando se alcanza el valor mínimo el conteo comienza nuevamente.

Nota: cuando el valor máximo no está configurado (0x0), el valor máximo es 0xFF.

3.5.9.4. Modo de contador de borde de entrada

El temporizador cuenta según los impulsos de los pines de entrada, el borde negado o el disparo de borde positivo se pueden seleccionar según el registro de control 2.

3.5.9.5. Generación de disparo (trigger)

Se genera una señal de activación en función de los eventos del temporizador, los eventos de desbordamiento y valor de coincidencia de contador pueden configurarse para generar un pulso de disparo en el pin timer_out.

3.5.10. INICIALIZACIÓN Y CONFIGURACIÓN

3.5.10.1. Operación de conteo libre

- Se escribe 1 en el bit START del registro de control CTRL.

El contador comienza a funcionar hasta que alcanza el valor máximo 0xFF y luego comienza de nuevo.

3.5.10.2. Operación de conteo normal

- Escribir los valores mínimo y máximo deseados en los registros respectivos.

- Escribir en el registro de inicialización del contador.
- Escribir los registros de coincidencia de ser necesario.
- Configurar los bits de habilitación de interrupciones M0_INT, M1_INT y OVF_INT si es necesario.
- Ajustar el bit de inicio START del registro de control.

3.5.10.3. Operación de conteo inverso

- Escribir los valores mínimo y máximo deseados en los registros respectivos.
- Escribir en el registro de inicialización del contador.
- Escribir los registros de coincidencia de ser necesario.
- Configurar los bits de habilitación de interrupciones M0_INT, M1_INT y OVF_INT si es necesario.
- Escribir 1 en el bit CNT_MODE del registro de control
- Ajustar el bit de inicio START del registro de control.

3.5.10.4. Interrupciones

- Interrupción de desbordamiento: se genera cuando el indicador OVF del registro de estado STATUS se establece y el bit OVF_INT se establece en el registro de control.
- Contador de interrupción Match 0: se genera cuando se establece el indicador M0F del registro de estado STATUS y se activa el bit de CTRL del registro de control M0F_INT.

- Contador de interrupción Match 1: se genera cuando se establece el indicador M0F del registro de estado STATUS y se activa el bit M1F_INT del CTRL del registro de control.

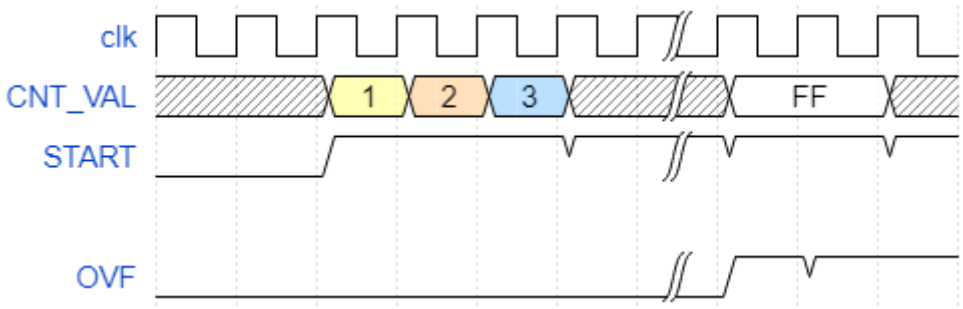
CAPITULO IV

RESULTADOS Y DISCUSIÓN

Se muestra los diagramas de tiempo de acuerdo con el modo de operación del contador, se ha creado las formas de onda como referencia para el diseño, se especifica el estado de los controles involucrados y los resultados esperados en los siguientes diagramas:

4.1. MODO DE CONTADOR LIBRE

Figura 4.1 Diagrama de contador en modo libre

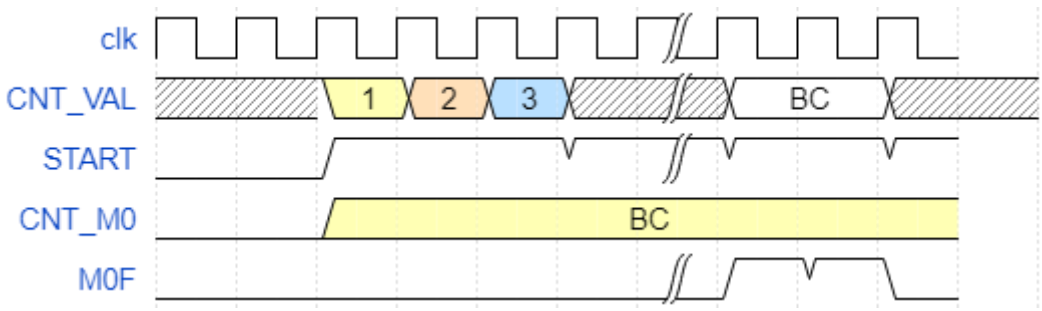


Elaboración propia.

En la figura arriba podemos visualizar el modo de funcionamiento denominado libre, contador libre, free run. El valor del contador empieza a ser incrementando a cada ciclo de reloj a partir del momento que el bit de control START es escrito con el valor 1, después de un tiempo este valor llega a ser 0xFF (hexadecimal FF) en ese momento se produce el evento de desbordamiento que es señalizado mediante la bandera OVF en el registro de control CTRL, la cuenta empieza de nuevo y sigue sucediendo así periódicamente.

4.2. MODO DE CONTADOR ASCENDENTE

Figura 4.2 Diagrama de tiempo en conteo normal

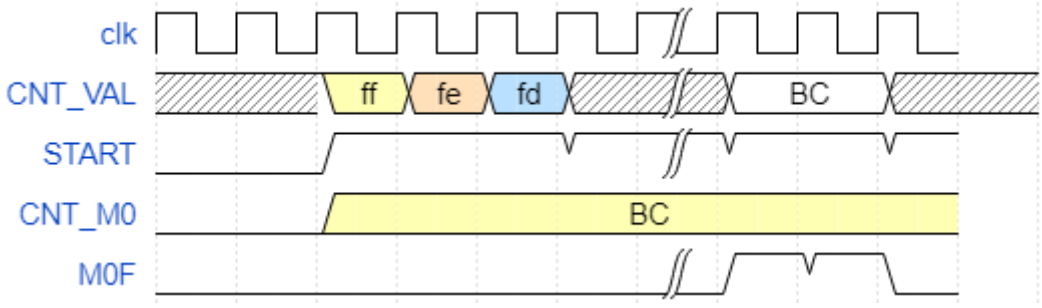


Elaboración propia.

En el modo de funcionamiento ascendente vemos que el valor máximo no es 0xFF como el modo de contador libre, el valor máximo es definido por el usuario para un valor especifico como por ejemplo en este caso 0xBC, en este diagrama también vemos que la bandera de señalización sube cuando el valor del contador alcanza el valor definido en el registro de coincidencia CNT_M0.

4.3. MODO DE CONTADOR DESCENDENTE

Figura 4.3 Diagrama de tiempo cuenta inversa



Elaboración propia.

A diferencia del contador en modo ascendente en este modo la cuenta es inversa, se va decrementando el valor del contador, sin embargo, el valor de coincidencia puede ser el mismo 0xBC, una vez llegada la cuenta a dicho valor la bandera M0F del registro

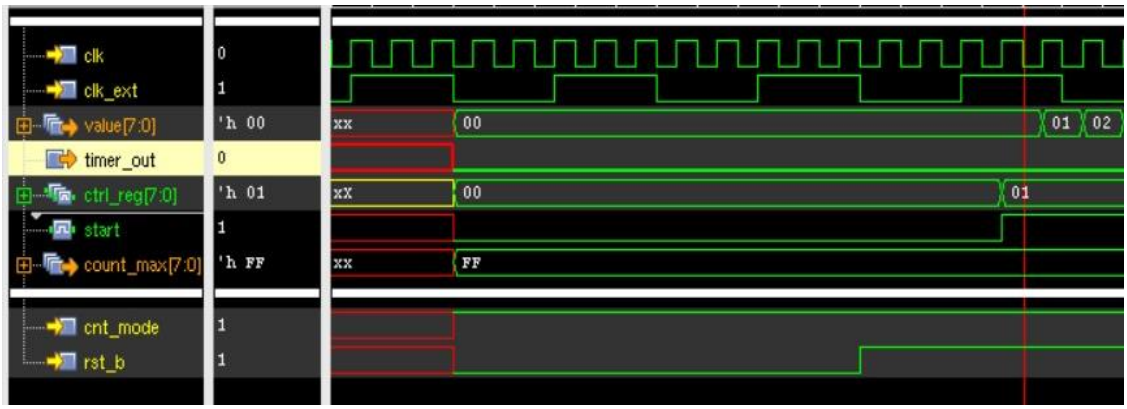
de estado STATUS es activada, la cuenta llegado a un valor mínimo definido retorna al valor más alto definido también por el usuario.

4.4. DIAGRAMAS DE SIMULACIÓN DE TIEMPO.

A continuación, se presentan los resultados de la implementación como también el código HDL del presente trabajo, el documento con las informaciones del componente implementado en este proyecto se redactó en idioma ingles y se encuentra disponible en la página https://github.com/joselcuevam/timer, se puede clonar el código mediante la aplicación de control de versión de archivos Git y simular el comportamiento del mismo usando el software libre Icarus verilog en un ambiente Linux. Para el presente trabajo se utilizó el sistema operativo Ubuntu, considerando la versión de Icarus verilog v.11. Se puede desarrollar estas simulaciones también en un ambiente Windows sin embargo las simulaciones del presente proyecto han sido realizados en el sistema operativo Linux Ubuntu.

4.4.1. MODO DE CONTADOR LIBRE

Figura 4.4 Inicio de ejecución en modo libre



Elaboración propia.

En este diagrama de tiempo vemos las señales de reloj interno o de sistema (clk) y el reloj externo (clk_ext), el valor del contador representado a través de la señal interna

value, el siguiente la señal de salida timer_out que está en cero debido a que no está siendo usado en el presente modo, el valor del registro de control CTRL mediante la señal ctrl_reg, el bit de inicio de conteo (start), la señal de reset rst_b y el modo de contador cnt_mode.

Inicialmente durante el estado de reset (cuando la señal rst_b está en nivel bajo 0) el sistema está inicializado en un estado conocido y no hay ejecución de tareas. Después de salir del estado de reset (cuando rst_b está en nivel alto 1) se escribe 1 en el campo START del registro CTRL de esta forma el contador empieza a contar de forma ascendente y una vez que alcanza el valor máximo la cuenta se repite indefinidamente.

4.4.2. MODO DE CONTADOR LIBRE, DESBORDAMIENTO

Figura 4.5 Desbordamiento del modo de ejecución libre

Elaboración propia.

Cuando la cuenta se aproxima al valor máximo 0xFF la bandera de desbordamiento, Overflow, cambia para un valor alto indicando que el contador tuvo un desborde de valor, en la figura está representada mediante la señal overflow_status_flag, este indicador estará señalizando el desborde hasta que sea borrado por software mediante la escritura en el registro STATUS y el bit OVF.

Tabla 4.1 Estado de registros desborde contador libre

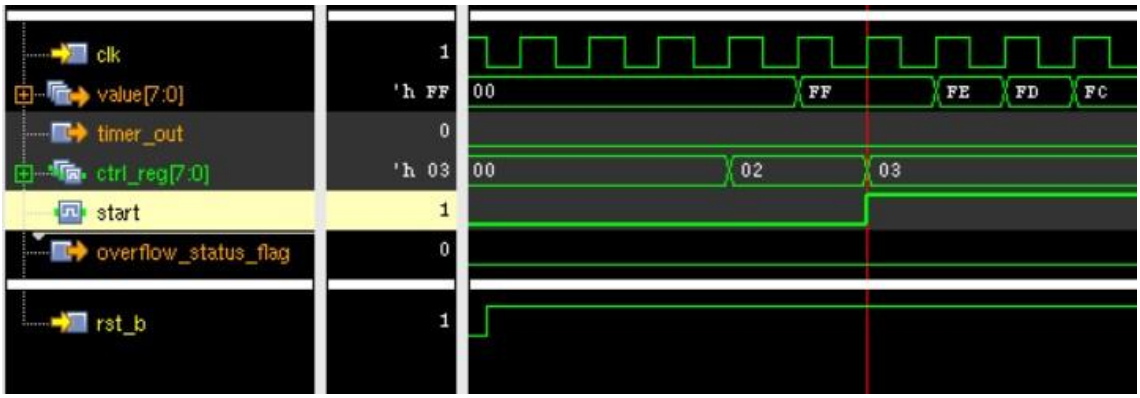
Estado inicial registros	Estado final registros
CTRL = 0x1	CTRL = 0x1
STATUS = 0x0	STATUS = 0x1 Indicador de desborde
CNT = 0x0	CNT = 0xFF

Elaboración propia

4.4.3. MODO CONTADOR LIBRE DESCENDENTE

Para este modo, se configura el modo de cuenta descendente escribiendo 1 en el campo CNT_MODE del registro de control CTRL, luego se escribe 1 en el bit de control START.

Figura 4.6 Ejecución libre, inicio del modo de cuenta regresiva



Elaboración propia.

Podemos apreciar en la figura que el valor inicial del contador cambia de 0x00 a 0xFF al configurar el modo de cuenta descendente, en seguida así que el bit de inicio START es habilitado la cuenta comienza.

Tabla 4.2 Estado de registros cuenta regresiva

Estado inicial registros	Estado final registros
CTRL = 0x2	CTRL = 0x3
STATUS = 0x0	STATUS = 0x0
CNT = 0x0	CNT = 0xFF y luego menos 1 a cada ciclo de reloj

Elaboración propia

4.4.4. MODO DE CONTADOR LIBRE DESCENDENTE, DESBORDAMIENTO

Así como se vio en la anterior figura la cuenta empezó a disminuir el valor del contador a cada ciclo de reloj, una vez que la cuenta llega al valor 0x0 se produce un desbordamiento de cuenta de 0x0 para 0xFF momento en el que también la bandera de señalización OVF es activada.

Figura 4.7 Ejecución libre, desbordamiento modo de cuenta descendente.

Elaboración propia.

La señal de bandera se mantiene indicando que se produjo un desbordamiento en el contador, se mantendrá así hasta que el usuario escriba en el mismo campo de control para deshabilitarlo, así como se mencionó anteriormente el registro CTRL y el bit OVF.

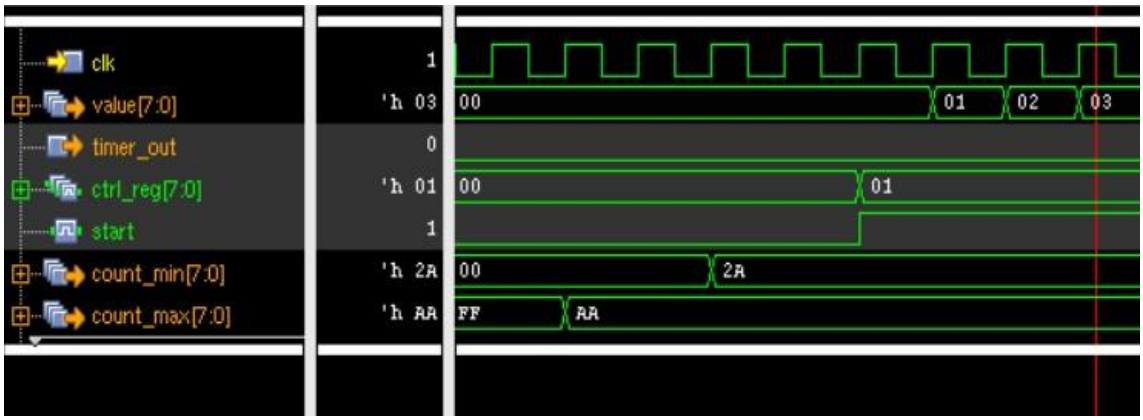
Tabla 4.3 Estado de registros desborde cuenta libre inversa

Estado inicial registros	Estado final registros
CTRL = 0x3	CTRL = 0x3
STATUS = 0x0	STATUS = 0x1 Indicador de desborde
CNT = 0xFF	CNT = 0x00 el valor del contador alcanza el valor mínimo 0x0 para luego retomar el valor 0xFF

Elaboración propia

4.4.5. MODO CONTADOR ASCENDENTE, INICIALIZACION DE VALORES MINIMO Y MAXIMO

Figura 4.8 Inicio del conteo ascendente.



Elaboración propia.

En secuencia se inicializa el valor de contador máximo MAX, el valor de contador mínimo MIN, se habilita el bit START, el contador empieza a funcionar. Cabe señalar que el valor de cuenta empieza la cuenta en 0 si el valor inicial no está configurado.

Tabla 4.4 Estado de registros desborde cuenta libre inversa

Estado inicial registros	Estado final registros
CTRL = 0x0	CTRL = 0x1
STATUS = 0x0	STATUS = 0x0
CNT = 0x00	CNT = 0x01 y luego el valor del contador se incrementa a cada ciclo de reloj
CNT_MAX = 0x00	CNT_MAX = 0xAA
CNT_MIN = 0x00	CNT_MAX = 0x2A

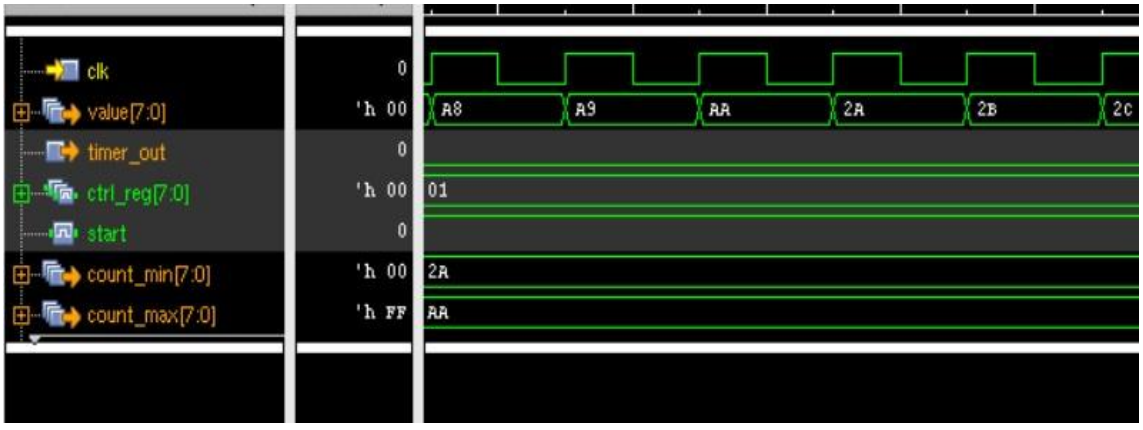
Elaboración propia

En esta configuración el contador no ha sido inicializado por lo tanto el valor inicial es 0, la cuenta, una vez que se alcance el valor máximo volverá al valor mínimo configurado.

4.4.6. MODO CONTADOR ASCENDENTE EN OPERACIÓN, DESBORDE

Al continuar en operación y alcanzar el valor máximo MAX, el contador regresa al valor mínimo MIN. Esto se muestra en la siguiente figura.

Figura 4.9 Contador en modo ascendente, desborde.



Elaboración propia.

Se reinicia la cuenta cuando se ha alcanzado el valor máximo configurado a través de los registros correspondientes mencionados en la tabla abajo, CNT_MAX, CNT_MIN.

Tabla 4.5 Contador Ascendente, retorno a valor mínimo

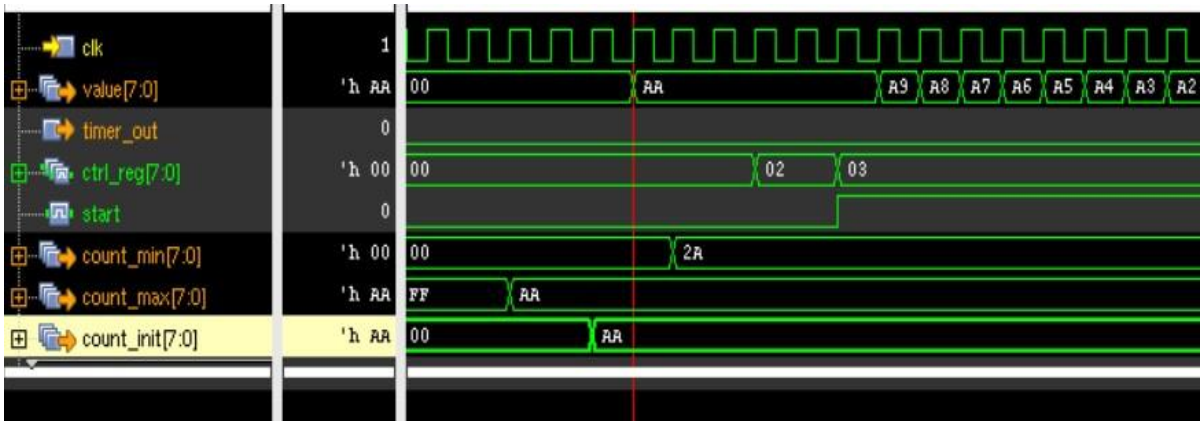
Estado inicial registros	Estado final registros
CTRL = 0x1	CTRL = 0x1
STATUS = 0x0	STATUS = 0x0 No se ha configurado valores de Match para generar banderas
CNT = 0xAA, este valor es el máximo valor disponible luego se retoma la cuenta desde el valor MIN	CNT = 0x2A y luego el valor del contador se incrementa a cada ciclo de reloj
CNT_MAX = 0xAA	CNT_MAX = 0xAA
CNT_MIN = 0x2A	CNT_MAX = 0x2A

Elaboración propia

4.4.7. MODO DE CONTADOR DESCENDENTE COMENZANDO

Se han configurado los valores CNT_MAX y CNT_MIN vía los registros respectivos, adicionalmente se configura el valor inicial CNT_INIT con el mismo valor que CNT_MAX para evitar que el contador comience con el valor inicial de cero, de esta forma la cuenta inicia en el valor máximo y va disminuyendo hasta alcanzar el valor mínimo.

Figura 4.10 Modo de cuenta descendente comenzando.



Elaboración propia.

Tabla 4.6 Contador Descendente, inicio de secuencia

Estado inicial registros	Estado final registros
CTRL = 0x0	CTRL = 0x3 Cuenta inversa y empezando a contar
STATUS = 0x0	STATUS = 0x0 No hay eventos
CNT_INIT = 0x00	CNT_INIT = 0xAA
CNT_MAX = 0xAA	CNT_MAX = 0xAA
CNT_MIN = 0x2A	CNT_MIN = 0x2A

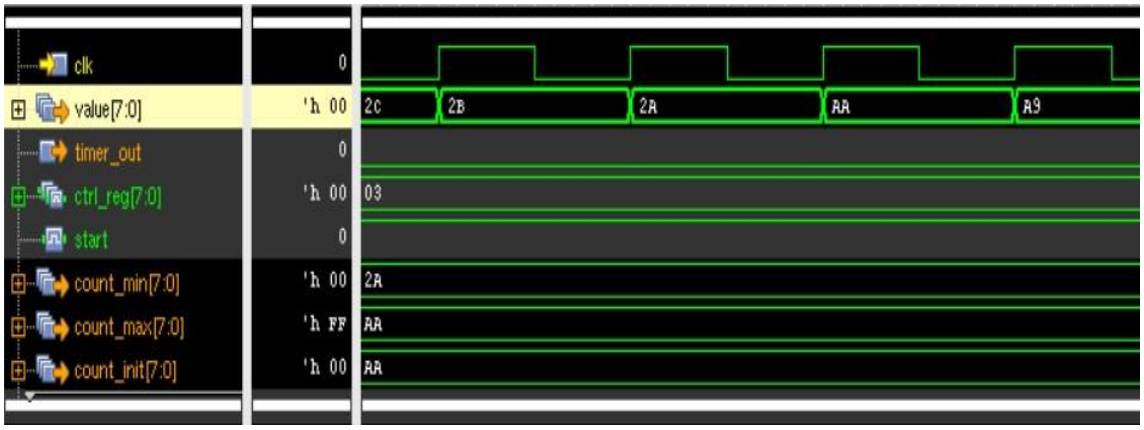
Elaboración propia

El contador es inicializado con el valor 0xAA la cuenta se prolonga hasta alcanzar el valor mínimo CNT_MIN y vuelve a repetirse indefinidamente hasta que el contador sea deshabilitado.

4.4.8. MODO DE CONTADOR DESCENDENTE, TRANSICION MIN-MAX

El contador reajusta de valor de la cuenta una vez alcanzado el valor CNT_MIN como siguiente valor a CNT_MAX, de esta forma retorna al valor inicial y reinicia el conteo.

Figura 4.11 Modo de cuenta regresiva



Elaboración propia.

Tabla 4.7 Contador Descendente, retorno a valor máximo

Estado inicial registros	Estado final registros
CTRL = 0x0	CTRL = 0x3 Cuenta inversa y empezando a contar
STATUS = 0x0	STATUS = 0x0 No hay eventos
CNT_MIN = 0x2A	CNT_MIN= 0x2A
CNT_MAX = 0xAA	CNT_MAX = 0xAA
CNT = 0x2A	CNT = 0xAA

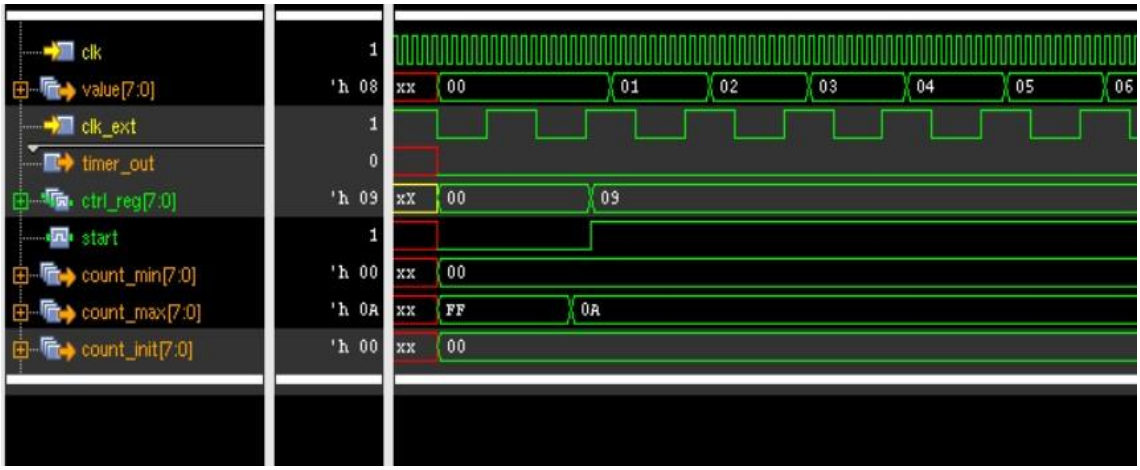
Elaboración propia

El retorno se realiza correctamente y la secuencia se repite nuevamente de acuerdo a la configuración mostrada en la tabla, el registro de estado STATUS no muestra cambios ya que no se han configurado los valores de coincidencia.

4.4.9. MODO CONTADOR DE PULSOS DE ENTRADA

En este modo, el contador aumenta según los impulsos en el pin de entrada (timer_in), en la figura aparece como clk_ext (renombrado a timer_in), el valor máximo que puede alcanzar en esta configuración especifica en este caso es 10 de acuerdo al valor CNT_MAX definido previamente, alcanzado este valor se reinicia la cuenta, este valor 10 o en hexadecimal 0xA se puede modificar con otro valor deseado.

Figura 4.12 Contador activado por pulsos de entrada



Elaboración propia.

Tabla 4.8 Contador funcionando con pulsos de entrada

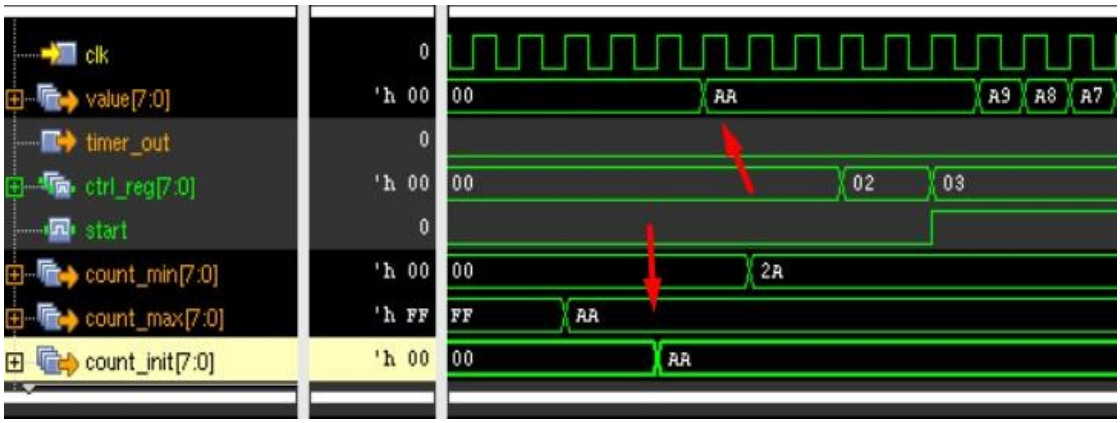
Estado inicial registros	Estado final registros
CTRL = 0x0	CTRL = 0x9 Cuenta usando reloj externo, contador habilitado
STATUS = 0x0	STATUS = 0x0 No hay eventos
CNT_MIN = 0x00	CNT_MIN = 0x00
CNT_MAX = 0x0A	CNT_MAX = 0x0A
CTRL_IN = 0x0	CTRL_IN = 0x0 Positive Edge

Elaboración propia

4.4.10. INICIALIZACIÓN DEL VALOR DEL CONTADOR

El valor inicial del contador se establece de acuerdo con las necesidades del usuario con el registro CNT_INIT, el valor de cuenta inicial predeterminado es cero. La figura muestra una inicialización con el valor 0xAA.

Figura 4.13 Inicialización del valor del contador.



Elaboración propia.

Tabla 4.9 Definición de valor inicial del contador

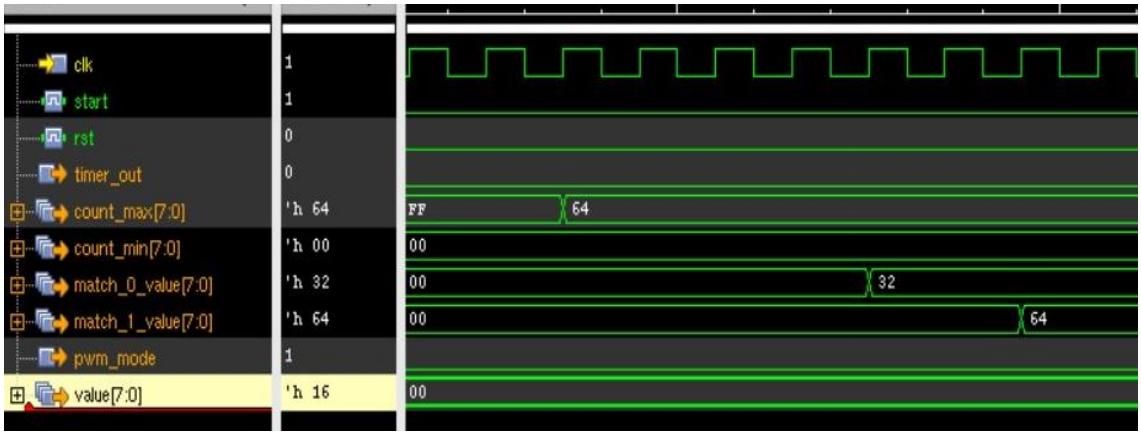
Estado inicial registros	Estado final registros
CNT_INIT = 0x0	CNT_INIT = 0xAA Modificando el valor inicial

Elaboración propia

4.4.11. GENERACIÓN DE SEÑAL MODULADA POR AMPLITUD DE PULSO PWM

La generación de una señal modulada por amplitud de pulso se puede realizar a través de los registros para valores de coincidencia CNT_M0 y CNT_M1, estos proveen un mecanismo para generar esta señal de salida alternando los valores de timer_out de acuerdo con los eventos de match 0/1, configura la señal de salida timer_out en alto 1 con match 0 mientras que los eventos de match 1 devuelven el valor de la señal a nivel bajo.

Figura 4.14 Registros para la generación de PWM.



Elaboración propia.

Tabla 4.10 Definición de valores para generación de señal PWM

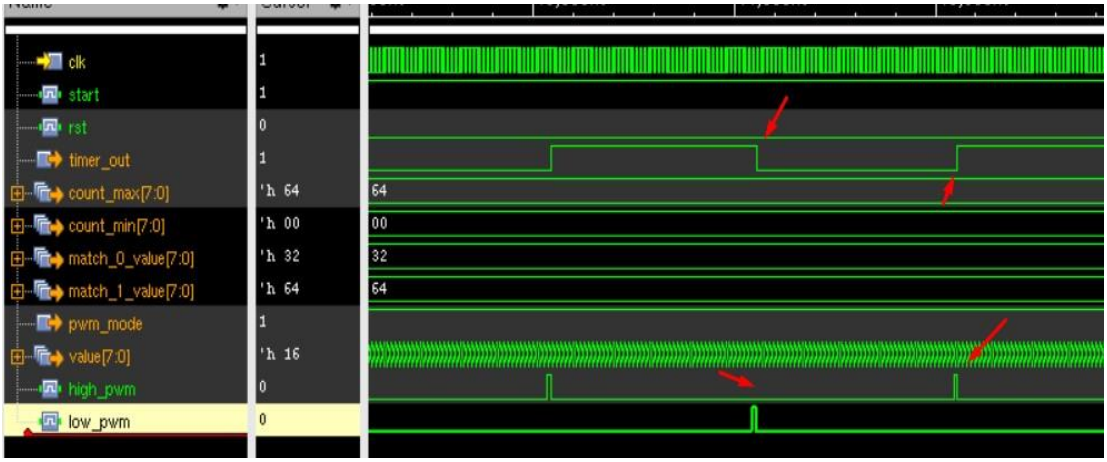
Estado inicial registros	Estado final registros
CNT_MAX= 0x0	CNT_MAX = 0x64
CNT_M0 =0x0	CNT_M0 =0x32
CNT_M1 =0x0	CNT_M1 =0x64
CTRL_OUT =0x0	CTRL_OUT = 0x1

Elaboración propia

Configuración de los registros de acuerdo a la tabla arriba, el valor de cuenta máximo se define como 0x64 mientras que los valores de coincidencia se configuran de tal forma que la señal tenga una modulación del 50%, CNT_M0 =0x32 y CNT_M1 = 0x64 respectivamente.

El bit PWM se habilita en el registro CTRL_OUT para que la generación de la señal sea efectiva en la interface de salida timer_out.

Figura 4.15 Generación de PWM.



Elaboración propia.

En la figura anterior vemos la señal timer_out con la forma de onda PWM siendo generada a través de los eventos de coincidencia de valor de contador con los registros CTN_M0 CTN_M1.

Tabla 4.11 Generación de señal PWM

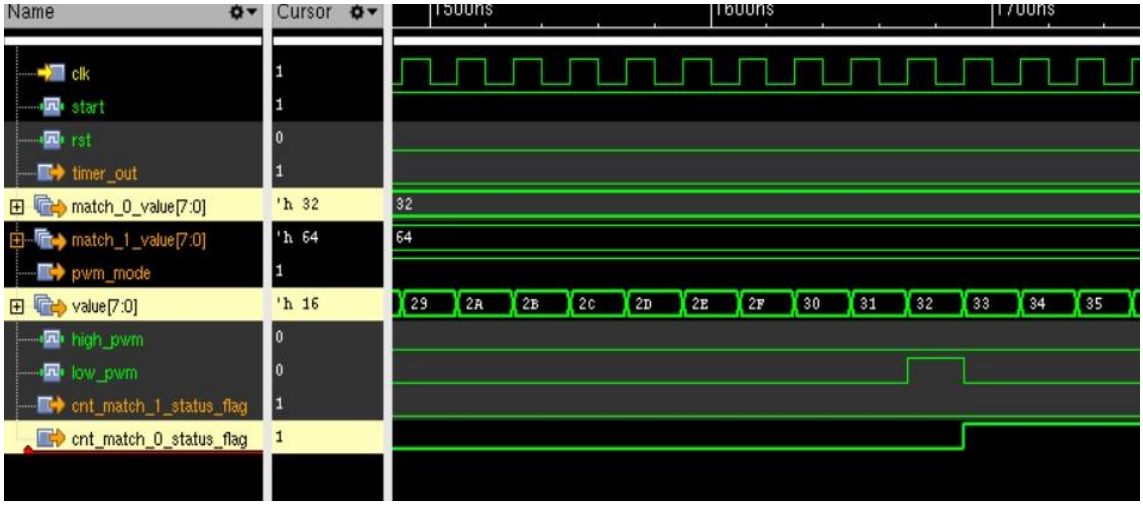
Estado inicial registros	Estado final registros
CNT_MAX= 0x64	CNT_MAX = 0x64
CNT_M0 =0x32	CNT_M0 =0x32, Alcanzado este valor la señal de salida va para un nivel lógico alto
CNT_M1 =0x32	CNT_M1 =0x64, Alcanzado este valor la señal de salida va para un nivel lógico bajo
CTRL_OUT =0x0	CTRL_OUT = 0x1, Generación de la señal PWM habilitada

Elaboración propia

4.4.12. BANDERAS DE CONTADOR MATCH

El siguiente gráfico muestra el momento en el que la bandera de coincidencia 0 es habilitada.

Figura 4.16 Bandera de conjuntos de Match 0.



Elaboración propia.

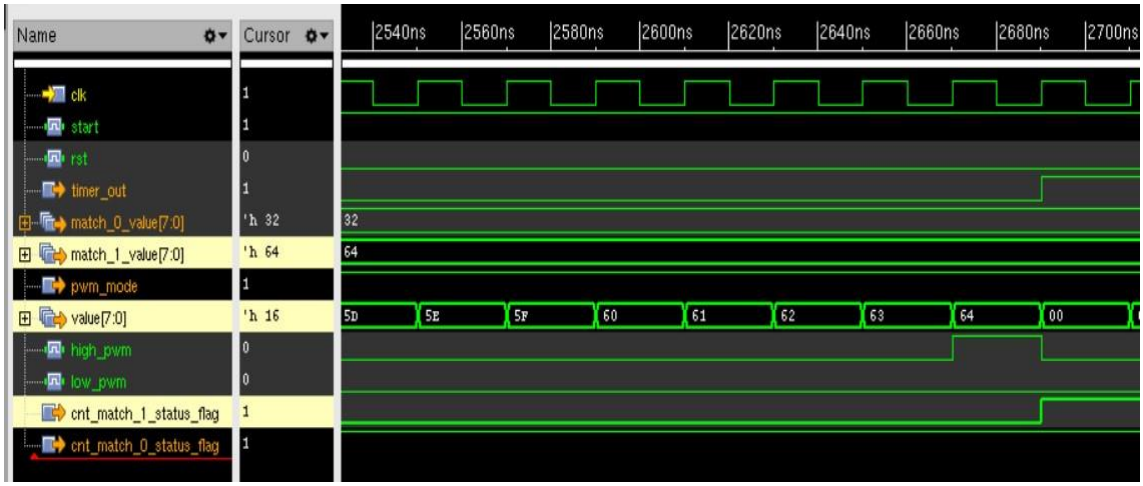
En el grafico el valor de coincidencia es 0x32, cuando este valor es alcanzado por el contador la bandera es habilitada inmediatamente un ciclo después.

Tabla 4.12 Generación de señales bandera M0 (Flag)

Estado inicial registros	Estado final registros
CNT_M0= 0x32	CNT_M0= 0x32
CRTL =0x1	CTRL = 0x1
STATUS =0x0	STATUS = 0x2, La bandera M0F está habilitada, bit posición 2

Elaboración propia

Figura 4.17 Bandera de conjuntos de Match 1.



Elaboración propia.

En la siguiente figura ocurre un funcionamiento similar, pero para el caso de configuración de coincidencia usando el registro CNT_M1.

Tabla 4.13 Generación de señales bandera M1 (Flag)

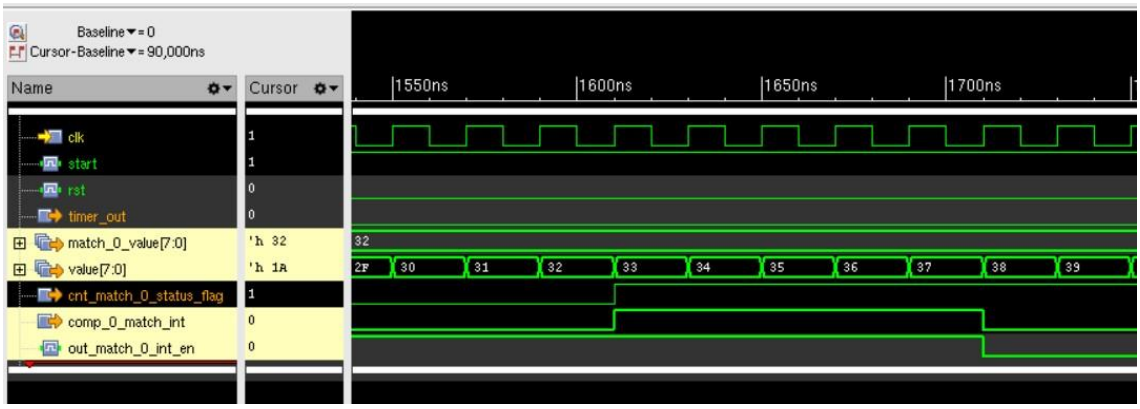
Estado inicial registros	Estado final registros
CNT_M1= 0x64	CNT_M1= 0x64
CRTL =0x1	CTRL = 0x1
STATUS =0x0	STATUS = 0x4, La bandera M1F está habilitada, bit posición 3

Elaboración propia

En este caso particular la configuración para el registro CNT_M1 es 0x64, entonces la bandera es generada un ciclo de reloj luego de que se produce la coincidencia, el detector de coincidencia lleva un ciclo para detectar este cambio y habilitar el señalizador M1F.

4.4.13. GENERACIÓN DE INTERRUPCIONES

Figura 4.18 Interrupción del valor de coincidencia.



Elaboración propia.

La función de interrupción, como se muestra en la figura, es generada simultáneamente con la señal de bandera sin embargo siempre que este habilitado la opción de generar interrupción ya sea para coincidencia Match 0 o coincidencia Match 1.

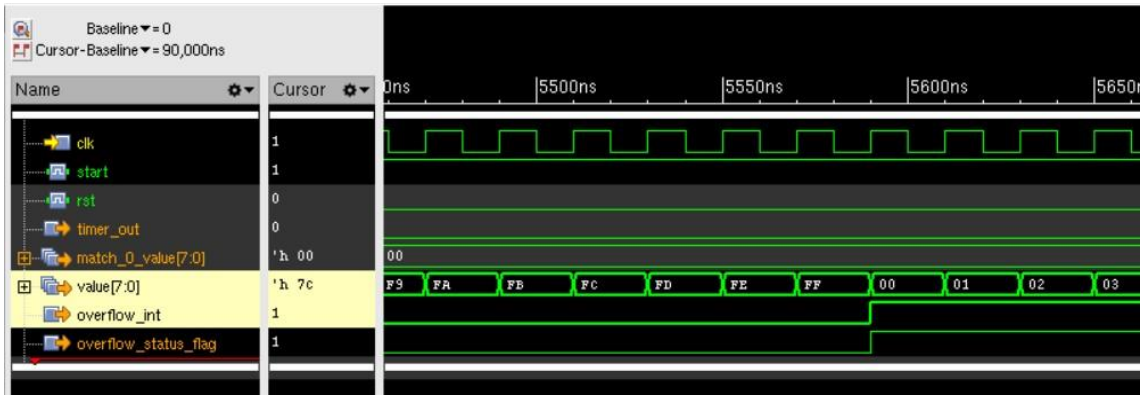
Tabla 4.14 Generación de interrupciones para Match 0

Estado inicial registros	Estado final registros
CNT_M0= 0x32	CNT_M0= 0x32
CRTL =0x0	CTRL = 0x21, Habilitamos la generación de interrupciones M0_INT, bit 5, y el funcionamiento START, bit 0
STATUS =0x0	STATUS = 0x2, La bandera M0F está habilitada, bit posición 2

Elaboración propia

4.4.14. INTERRUPCIÓN DE DESBORDAMIENTO

Figura 4.19 Interrupción por desbordamiento.



Elaboración propia.

La interrupción de desbordamiento del modo contador libre se genera cuando se alcanza la transición del valor máximo al mínimo o viceversa para el caso del contador en sentido descendente.

Tabla 4.15 Generación de interrupcion para desborde.

Estado inicial registros	Estado final registros
CRTL =0x0	CTRL = 0x11, Habilitamos la generación de interrupciones OVF_INT, bit 4, y el funcionamiento START, bit 0
STATUS =0x0	STATUS = 0x10, La bandera OVF está habilitada, bit posición 1

Elaboración propia

El registro de control es configurado para la generación de la interrupción previamente y el registro de estado muestra a la vez la generación de la señal de bandera que será usada posteriormente para poder deshabilitar la interrupción una vez que esta es tratada por el software.

4.4.15. PRESCALER DE ENTRADA

El Prescaler de entrada o divisor de frecuencia ayuda a generar tiempos más largos de cuenta, en esta figura apreciamos que se incrementa el contador a cada 8 ciclos del reloj externo, el valor del campo de control PS en el registro CTRL_IN es igual a 3, división por 8.

Figura 4.20 Prescaler habilitado.

Elaboración propia.

Tabla 4.16 funcionamiento del preescaler.

Estado inicial registros	Estado final registros
CRTL =0x0	CTRL = 0x9, Habilitamos el reloj externo CLK_EXT, bit 3, y el funcionamiento START, bit 0
CTRL_IN=0x0	CTRL_IN=0x30 el valor de Prescaler PS igual a 3 en la posición bit 4

Elaboración propia

Una vez que el Prescaler es habilitado internamente se activa la lógica del contador de entrada en el bloque (input_block) que se incrementa a cada pulso de entrada y una vez que se alcanza el valor de Prescaler genera una señal de disparo interno para incrementar por 1 el valor del contador principal.

4.4.16. MODO CONTADOR LIBRE, GENERACIÓN DE DISPARO TRIGGER

En el modo de contador libre o modo normal, se produce la señal de disparo (trigger) durante un ciclo de reloj para poder desencadenar una función del sistema en chip, este es un pulso con una duración de un ciclo de reloj. Una vez terminado el evento que lo desencadeno la señal vuelve a su estado original, nivel bajo.

Figura 4.21 Desbordamiento de modo de ejecución libre de generación de disparador.

Elaboración propia.

Tabla 4.17 Generación de disparo por desbordamiento.

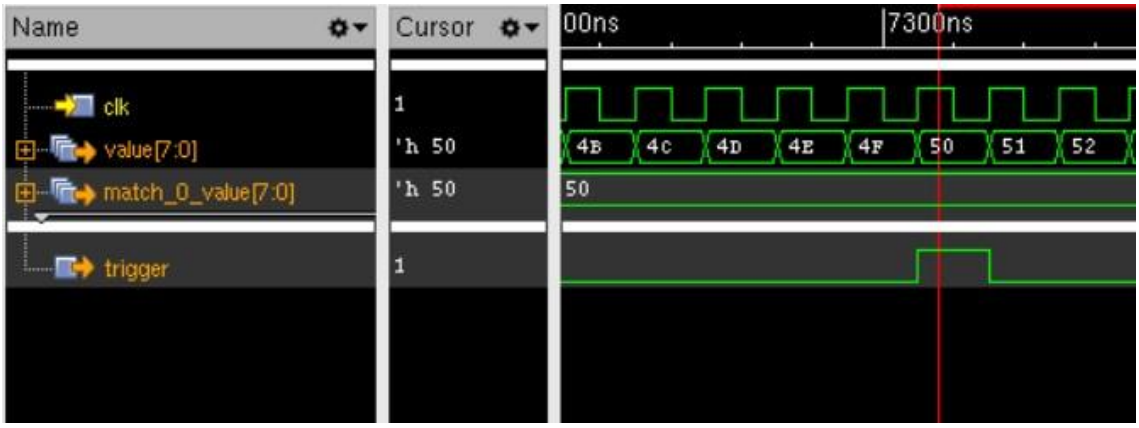
Estado inicial registros	Estado final registros
CRTL =0x0	CTRL = 0x1, Habilitado el funcionamiento con el bit START, bit 0
CTRL_OUT =0x0	CTRL_OUT =0x10, habilitado el bit 4, OVF_TRG, para permitir la generación de la señal de disparo

Elaboración propia

4.4.17. MODO CONTADOR ASCENDENTE/DESCENDENTE GENERACIÓN DE DISPARO

El disparo (trigger) se genera también mediante los registros de coincidencia de valores, cuando el valor del contador alcanza em mismo valor que los registros CNT_M0 o CTN_M1 y está habilitado mediante el registro CTRL_OUT.

Figura 4.22 Generación de disparo o trigger.



Elaboración propia.

Tabla 4.18 Generación de disparo para Match 0

Estado inicial registros	Estado final registros
CNT_M0= 0x50	CNT_M0= 0x50
CTRL= 0x0	CTRL= 0x1, Habilitamos el funcionamiento START, bit 0
CTRL_OUT =0x0	CTRL_OUT = 0x20, La generación de disparo M0_TRG, bit posición 5

Elaboración propia

Cada vez que el registro contador alcance el valor CNT_M0 la señal de disparo será habilitada durante un ciclo de reloj.

4.5. DISCUSIÓN

4.5.1. SIMULACIONES DEL DISEÑO

Basado en los antecedentes Antecedente 1, del autor Andrés Roldan Aranda se ha considerado el uso herramientas pagas, basado en la confiabilidad y soporte que estos proporcionan al estándar de Verilog IEEE 1364-2001, el autor mencionado usó el sistema para realizar simulaciones con componentes mixtos analógicos digitales, en este caso en este proyecto particular del temporizador se ha usado simulaciones netamente digitales, en complementariedad de estas simulaciones en el presente proyecto mencionamos y usamos también Software CAD libre como es Icarus Verilog, esto teniendo en cuenta que se necesita tomar ciertos criterios y cuidados puesto que no todas las especificaciones del estándar están implementados en el sistema libre esto puede producir errores de compilaciones como por ejemplo para las estructuras de señales vectoriales no tienen soporte en versiones de software gratuito.

4.5.2. IMPLEMENTACIÓN DEL SISTEMA EN FPGA

Basado en los antecedentes Antecedente 2, del autor Jose Francisco Osorio Jimenez, se toma en cuenta la implementación realizada en el sistema FPGA de acuerdo con los resultados mostrados que son aplicados a terceros sistemas como una mejora del presente proyecto, puesto que se realizó una fase de pruebas básicas de funcionamiento sin tener pruebas de uso general con dispositivos externos.

Las pruebas realizadas por el autor mencionado en la referencia son similares y tienen un énfasis diferente ya que la aplicación del proyecto mencionado es de carácter específico y el temporizador tiene una aplicación de mayor alcance puesto que puede ser reconfigurado para diferentes usos.

4.5.3. COMPARATIVAS CON TEMPORIZADORES

Basado en los antecedentes Antecedente 3, del autor Daniel Valgañon Medecigo

El temporizador implementado por el autor tiene características similares a este proyecto, el enfoque tiene ciertas particularidades, el uso que el proyecto referenciado propone es para ser implementado a nivel de FPGA, sin embargo, el presente trabajo tiene también aspiraciones de ser incluido en sistemas que puedan ser programados en FPGA o tanto asi como ser fabricados en silicio.

Los proyectos tienen bastante similitud en cuanto a los objetivos la diferencia está en la implementación y la arquitectura de estos.

CONCLUSIONES

Se implementó un temporizador digital bajo especificaciones propias en lenguaje Verilog y se logró hacer la simulación en software libre como Icarus verilog y de pago como Cadence Incisive, este diseño será divulgado e integrado en diseños más complejos como sistemas en chip (SOC), por ejemplo, dentro de un microcontrolador.

Se ha elaborado un diagrama de bloques que muestra la principales etapas y funciones del temporizador de forma gráfica.

Se ha logrado implementar funciones de Modulación por Ancho de Pulsos (PWM), contador de pulsos, generador de interrupciones, los registros de control para seleccionar y definir los datos necesarios para el modo de funcionamiento: INT_EN (habilitar interrupción), MODE (modo de funcionamiento), TIMER_START (habilitar el contador), MAX_VALUE (valor máximo del contador) INIT_VALUE (valor inicial) entre otros campos necesarios para definir los modos de operación también han sido implementados.

Se redactó la hoja de características y guía de integración de este módulo temporizador en función de los resultados de esta investigación, estos datos se han elaborado separadamente en idioma ingles por ser de mayor divulgación.

Se verificó las características de este diseño, cuenta ascendente, cuenta descendente, interrupciones, cuenta de pulsos externos, generación de señal PWM en una matriz de puertas programables (FPGA).

RECOMENDACIONES

Se recomienda realizar diseños de componentes similares, UART, Controlador de interrupciones, usando como referencia este diseño en particular.

El diagrama de bloques puede ser actualizado en base a nuevas características, por ejemplo, incrementar la resolución del contador y adicionar nuevas funcionalidades.

Se recomienda adicionar nuevas características a las ya existentes como por ejemplo decodificador de cuadratura, captura de valor de contador en base a señales de disparo en el puerto de entrada timer_in, actualización en vivo de valores mínimo y máximo de cuenta durante funcionamiento mediante un esquema de sincronización sofisticado, entre otros que pudieran ser adicionamos o actualizados en base a revisiones posteriores.

Se recomienda actualizar la hoja de características con más detalles sobre el funcionamiento de cada característica, así como adicionar restricciones sobre configuraciones invalidas.

Es posible crear más pruebas en el dispositivo FPGA y en base a esto corregir posibles errores no encontrados durante el diseño de este módulo, también sería útil tener estos datos registrados en la hoja de características.

REFERENCIAS BIBLIOGRÁFICAS

Brown S. (2013). Fundamentals of Digital Logic with Verilog Design 3rd Edition, Toronto, United States of America, The McGraw Hill Companies.

Cavanagh J. (2017) Verilog HDL Design Examples 1st Edition, Florida, United States of America, CRC Press.

Clietti M. (2017). Advanced Digital Design With The Verilog Hdl, 2Nd Edn Paperback, India, Pearson India.

Li Y. (2015). Computer Principles and Design in Verilog HDL 1st Edition, Solaris South Tower Singapore, Wiley.

Magda Y. (2017). A Practical Introduction to Verilog using Mojo V3 Kindle Edition, , United States of America, Amazon Digital Services LLC.

Mishra K. (2013). Advanced Chip Design, Practical Examples in Verilog Paperback, Silicon Valley, United States of America, TIE Silicon Valley.

Mehler R. (2014) Digital Integrated Circuit Design Using Verilog and Systemverilog 1st Edition, Massachusetts, United States of America, Newnes.

Monk, S. (2016). Programming FPGAs: Getting Started with Verilog 1st Edition. Columbus, United States of America: McGraw-Hill Education

Nano M. Ciletti M. (2017). Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog (6th Edition), United States of America, Pearson.

Rajewski J. (2017) Learning FPGAs: Digital Design for Beginners with Mojo and Lucid HDL 1st Edition, California, United States of America, O'Reilly Media.

Readler B. (2014). Vhdl By Example, United States of America, Full Arc Press.

Romano D. (2016). Make: FPGAs: Turning Software into Hardware with Eight Fun and Easy DIY Projects 1st Edition, California, United States of America, Maker Media.

Roth C; Jhon L; Lee B. (2015). Digital Systems Design Using Verilog (Activate Learning with these NEW titles from Engineering!) 1st Edition, Texas, United States of America, CL Engineering

Sutherland S. (2017). RTL Modeling with SystemVerilog for Simulation and Synthesis: Using SystemVerilog for ASIC and FPGA Design 1st Edition, Oregon, United States of America, CreateSpce, An Amazon,com Company.

Taraate V. (2016). Digital Logic Design Using Verilog: Coding and RTL Synthesis 1st ed; Oregon, United States of America, Springer.

Thomas D. (2016). Logic Design and Verification Using SystemVerilog (Revised) Paperback, Pittsburgh, United States of America, CreateSpce, An Amazon,com Company.

Unsalan C; Tar B. (2017) Digital System Design with FPGA: Implementation Using Verilog and VHDL 1st Edition, New York, United States of America, The McGraw Hill Companies.

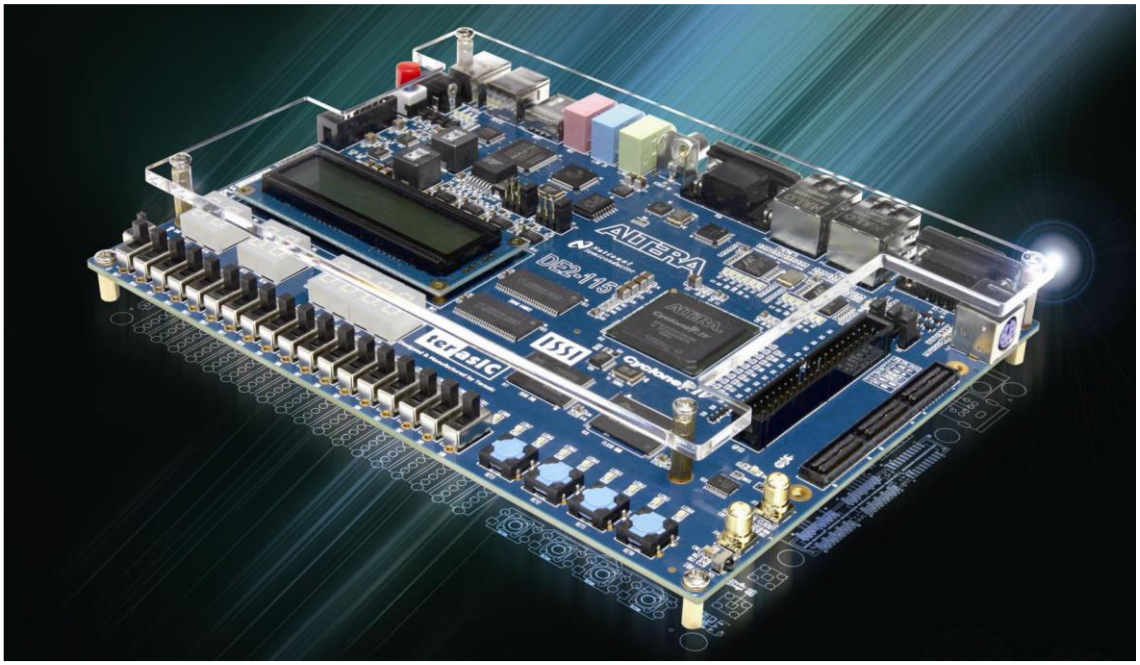
Williams J. (2014). Digital VLSI Design with Verilog: A Textbook from Silicon Valley Polytechnic Institute 2nd ed. Oregon, United States of America, Springer.

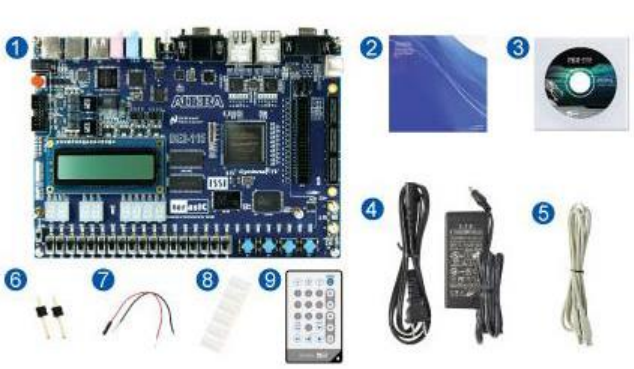
Wilson P. (2015) Design Recipes for FPGAs, Second Edition: Using Verilog and VHDL 2nd Edition, United Kingdom, Newnes.

Verilogcode.com (2015). Digital Logic RTL & Verilog Interview Questions Paperback, United States of America, Independent Publishing Platform.

ANEXOS

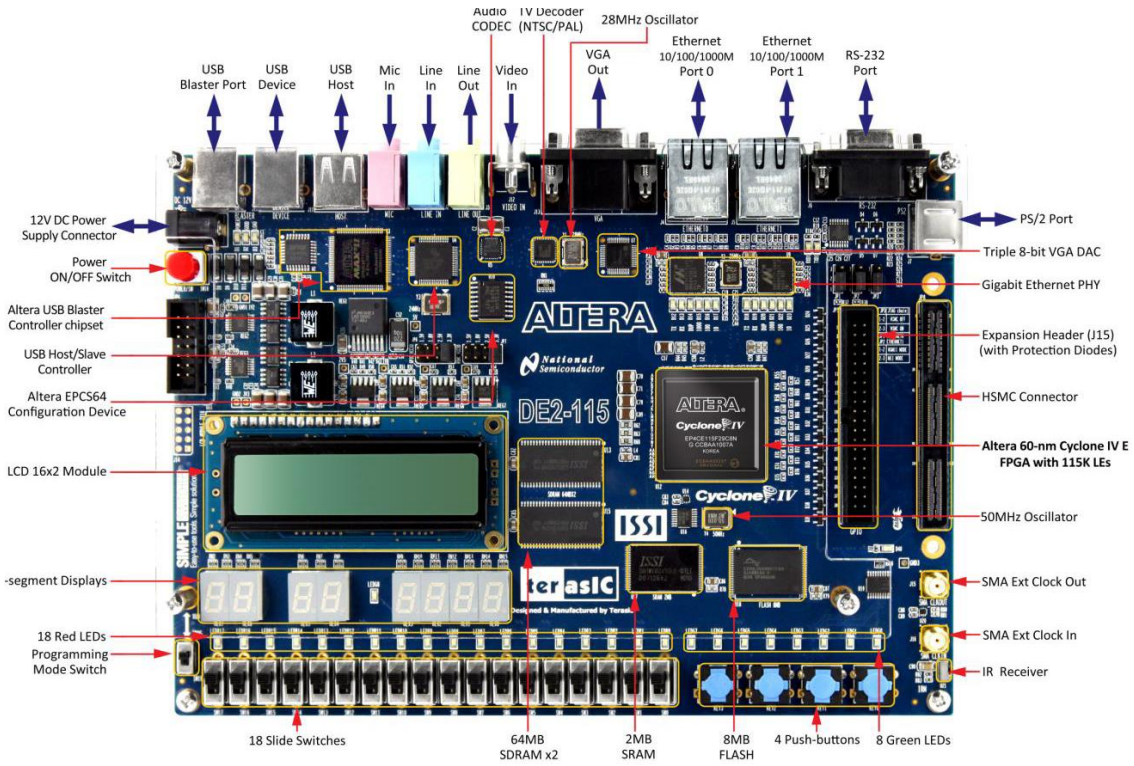
ANEXO 1. FPGA DE2-115 Altera

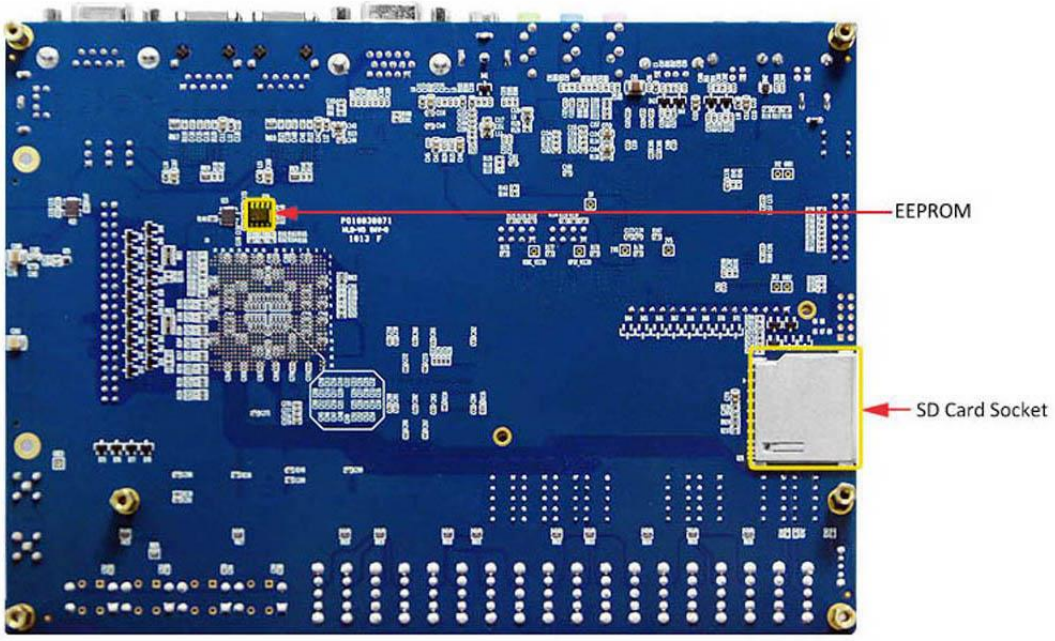




1. Altera DE2-115 Board
2. Altera Complete Design Suite DVD for Windows
3. DE2-115 System CD
4. Power DC Adapter 12V/2A
5. USB Programming Cable
6. Two 1-pin Headers
7. Two Wire Strips (black and red)
8. Six Silicon Footstands
9. Remote Controller

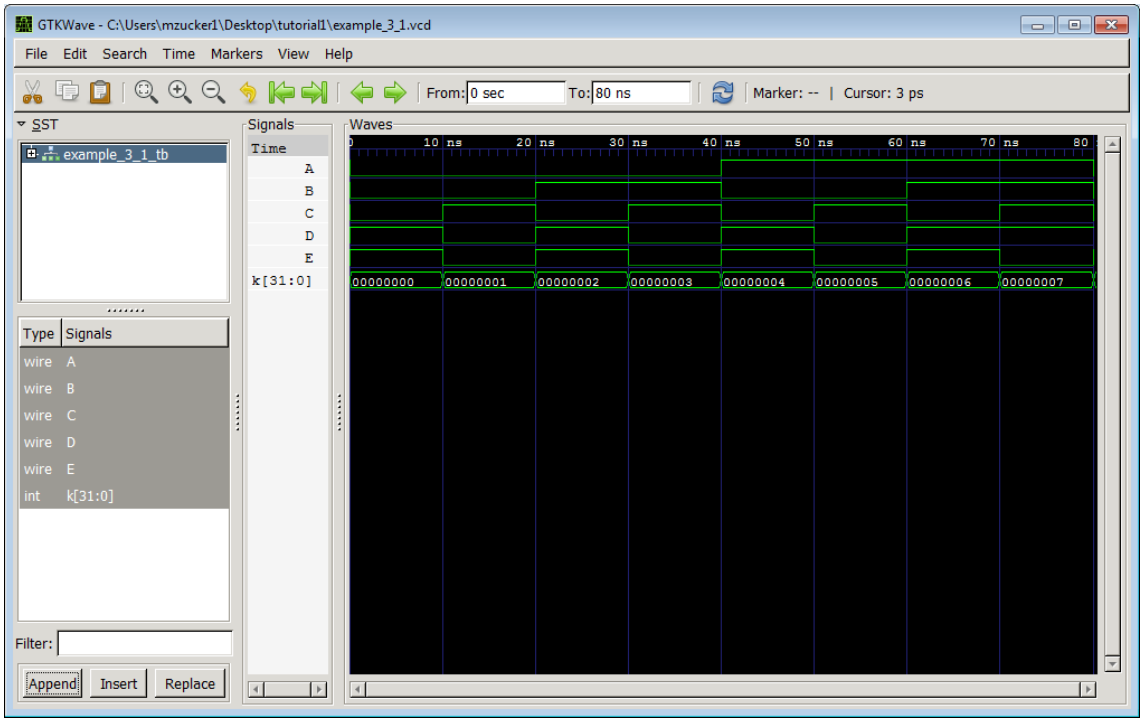
Fuente: https://www.terasic.com.tw/en/

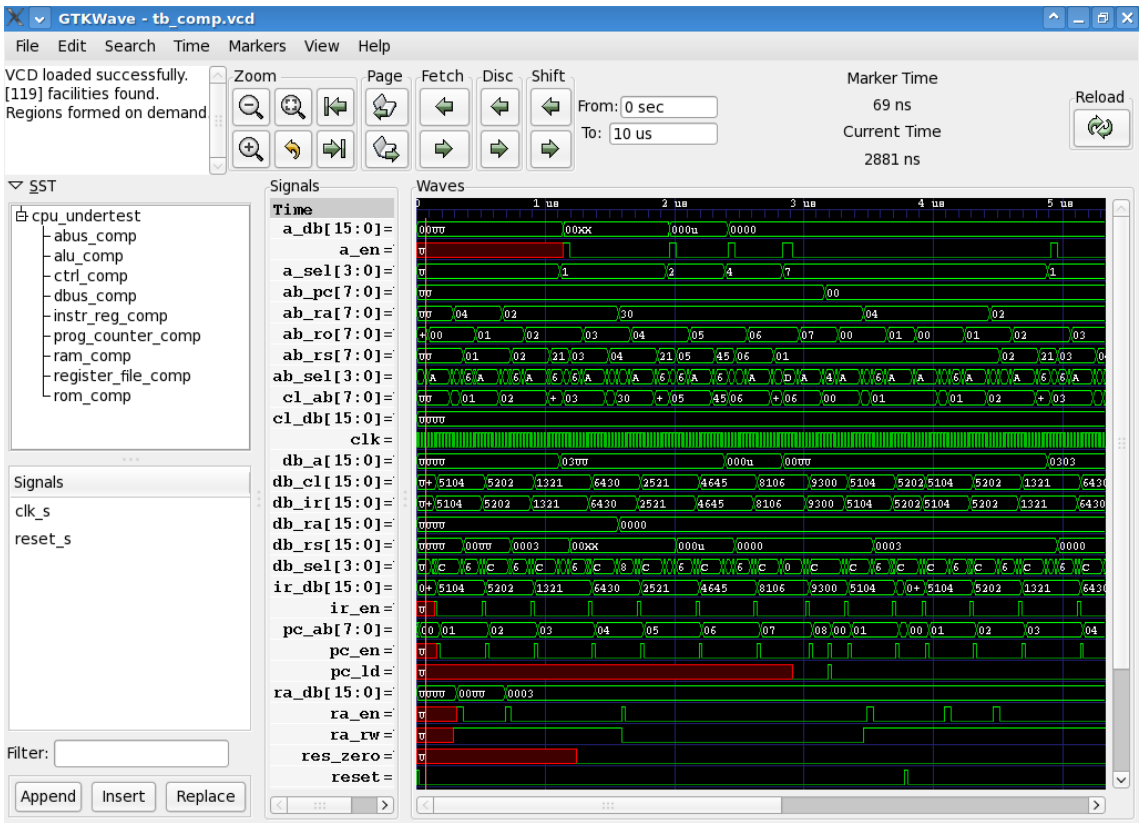




Fuente: https://www.terasic.com.tw/en/

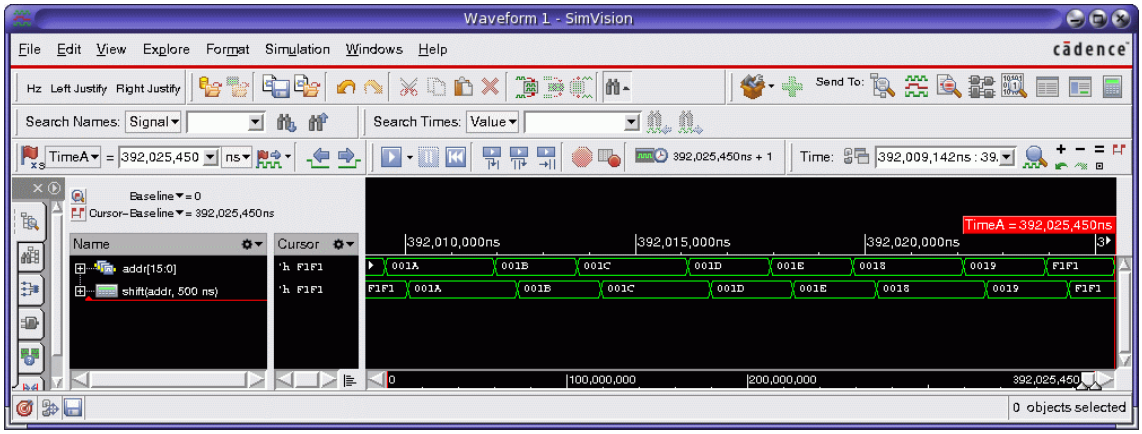
ANEXO 2. Icarus verilog & GTKWave

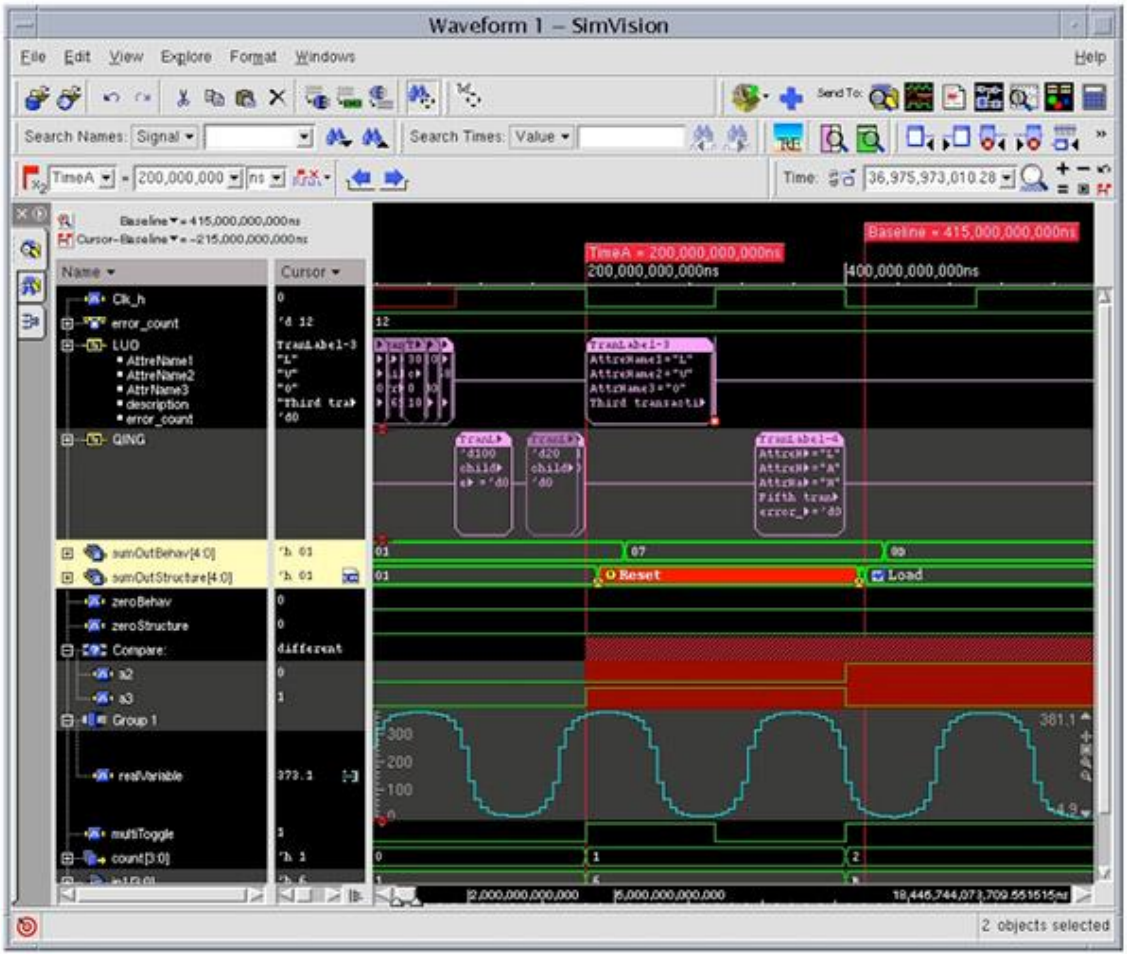




Fuente: http://iverilog.icarus.com/

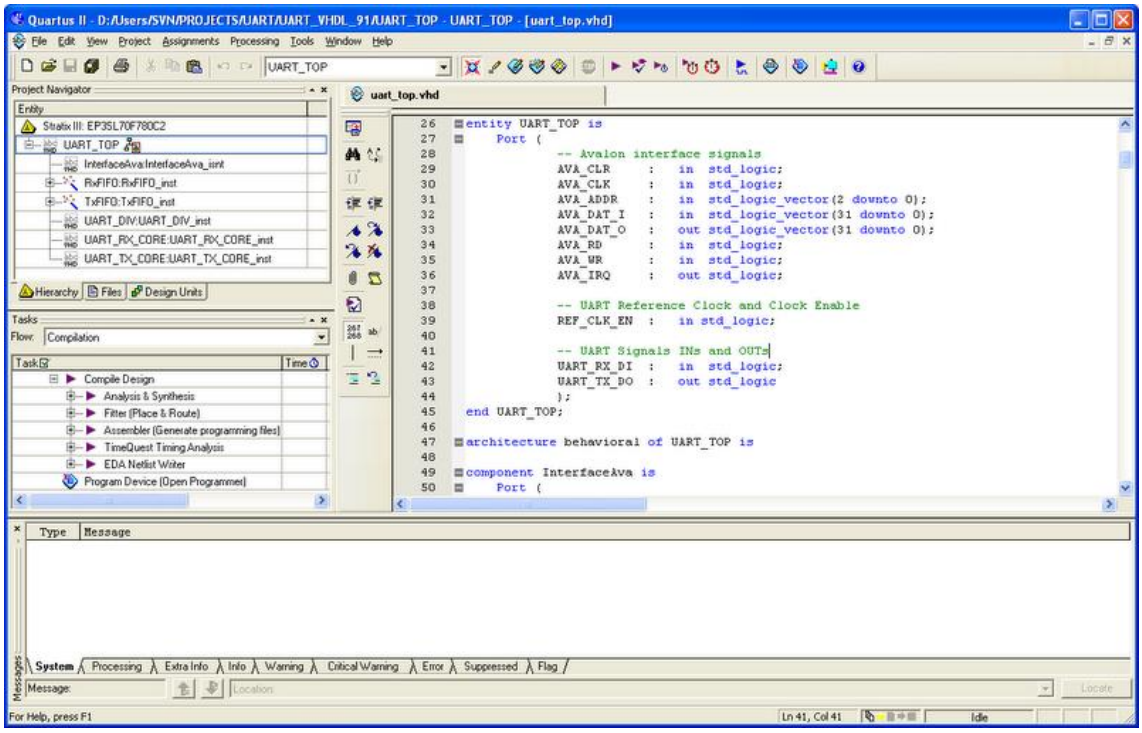
ANEXO 3. Cadence Incisive





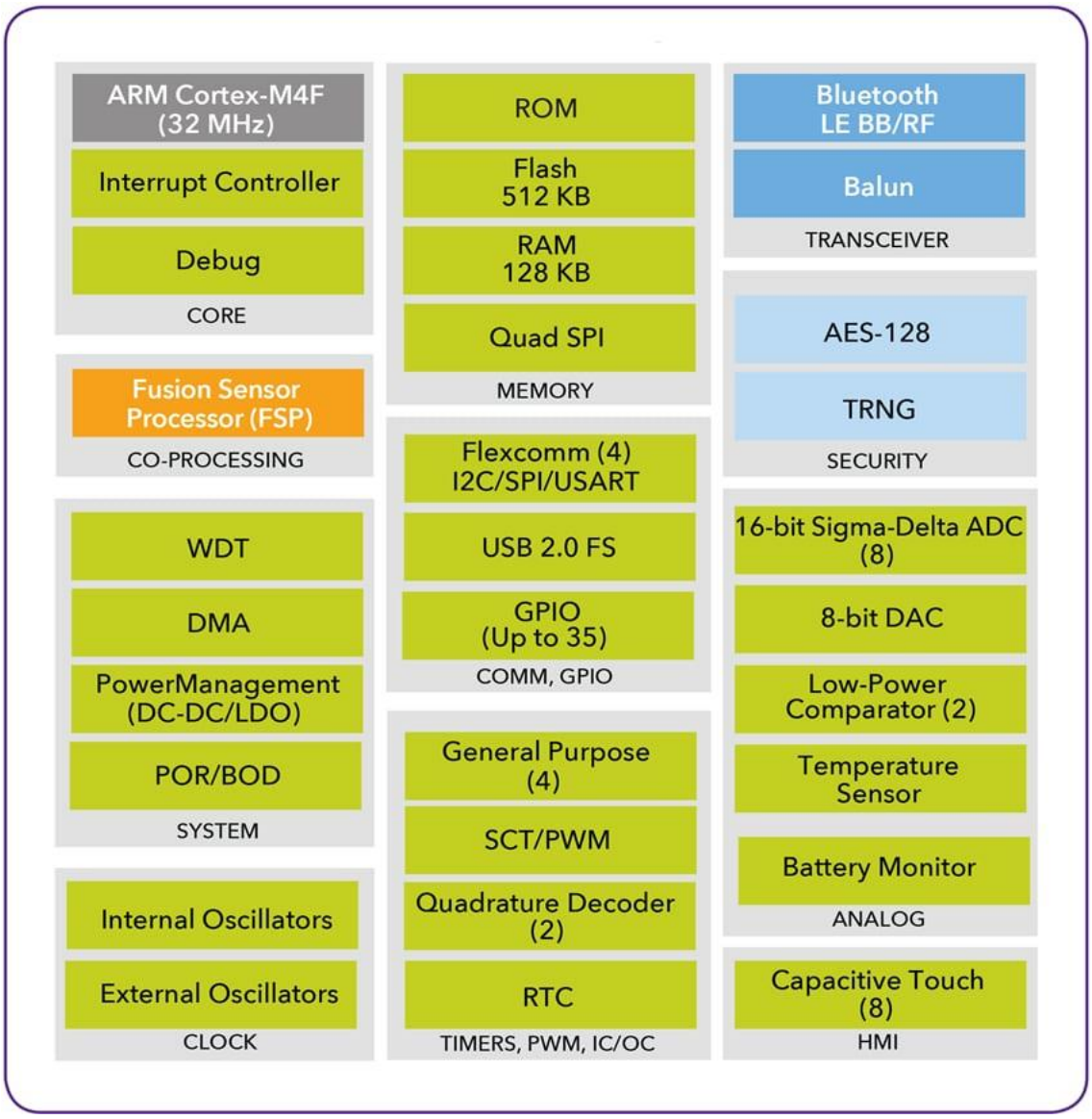
Fuente: https://www.cadence.com/

ANEXO 4. Altera Quartus II



Fuente: http://fpgasoftware.intel.com/

ANEXO 5. Sistema SoC



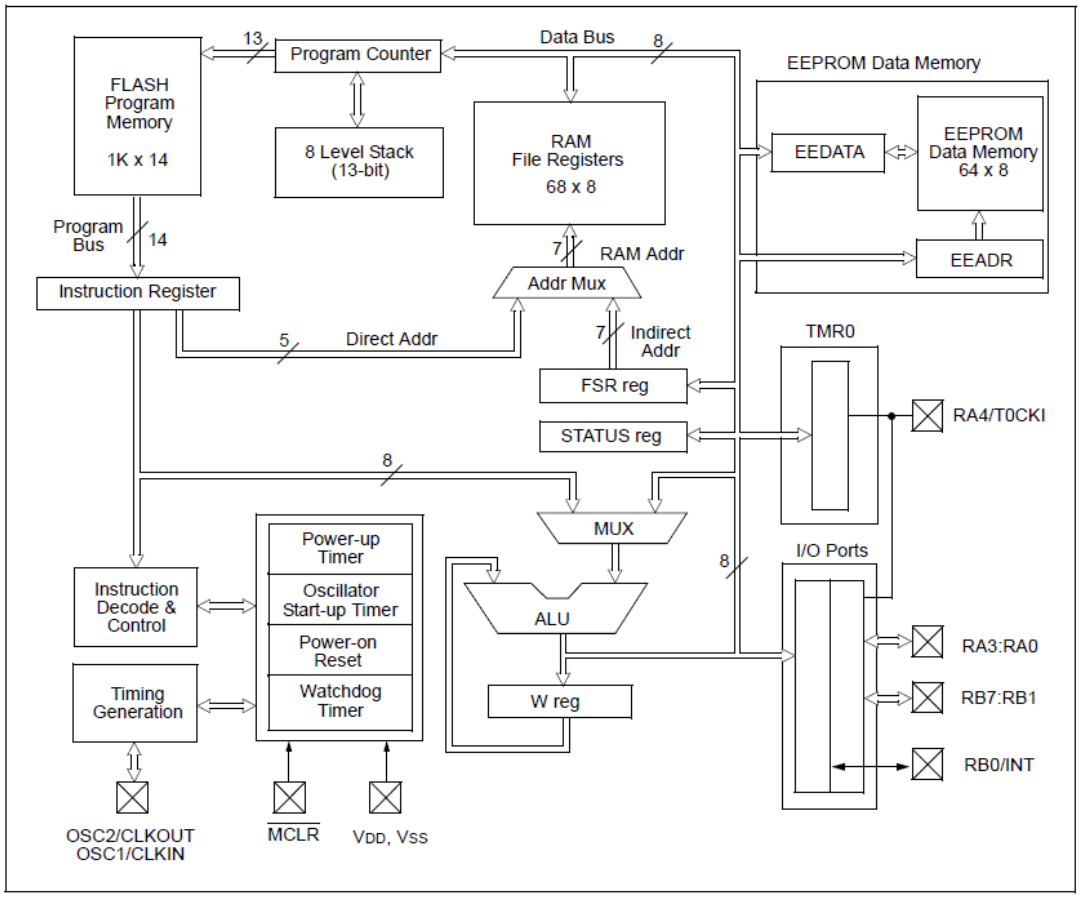
Fuente: www.nxp.com

Aplicaciones

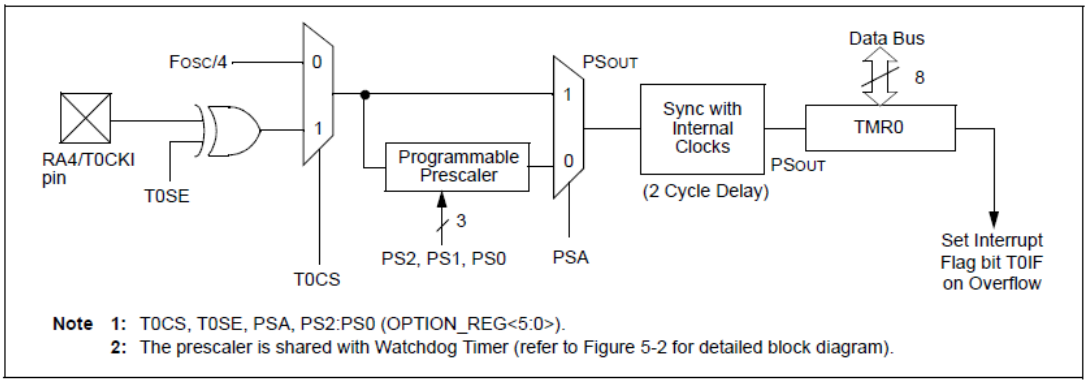
- Deportes, control.
- Control de signos vitales.
- Control remoto.

- Accesorios para teléfono inteligente.
- Periféricos de computadores.
- Redes de sensores inalámbricos.

ANEXO 6. Referencia temporizador del PIC16F84



Fuente: www.microchip.com



Fuente: www.microchip.com

Características:

- 8-bit timer/counter

- Readable and writable
- Internal or external clock select
- Edge select for external clock
- 8-bit software programmable prescaler
- Interrupt-on-overflow from FFh to 00h

Registros

REGISTER 2-2: OPTION REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Fuente: www.microchip.com

- Selección de fuente de reloj
- Selección de borda para contar
- Selección de preescaler
- Selección de tamaño de preescaler

Funcionamiento

Existen dos modos, modo TIMER, modo COUNTER, en el primer modo el temporizador incrementa el contador interno usando el reloj interno, durante el segundo modo usa la señal de entrada (seleccionable entre borde de subida y borde de bajada) para incrementar el contador.

Interrupción

La señal de interrupción es generada una vez que el contador alcanza el valor 0xFF y vuelvo a 0x00, el registro de STATUS marca el evento estando el FLAG de interrupción el cual necesita ser limpiado para ser usado nuevamente.

REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE Write Complete interrupts
0 = Disables the EE Write Complete interrupt

bit 5 **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Fuente: www.microchip.com

ANEXO 7. Implementación del Código HDL

La implementación del código se divide en los siguientes componentes:

- Bloque principal (d_ip_timer)
- Bloque de entrada (input_block)
- Bloque de salida (output_block)
- Bloque de registros (timer_registers)
- Bloque de contador (counter)

Las operaciones de conteo se realizan dentro del módulo counter, en cambio las operaciones de escritura y lectura de registros se realizan en el módulo timer_registers, las operaciones de sincronización y control de prescaler se encuentran en el módulo input_block, las operaciones de control de interrupciones se encuentran en el módulo output_block.

- Módulo de conteo (counter)

```
module timer_counter
    #(
      parameter COUNTER_SIZE  = 8    // counter size
     )(
       input                          clk,
                                      en,
                                      rst,
                                      cnt_mode,
                                      free,
                                      init_cnt,
       input    [COUNTER_SIZE-1:0]  min,
       input    [COUNTER_SIZE-1:0]  max,
       input    [COUNTER_SIZE-1:0]  init,
       output   reg [COUNTER_SIZE-1:0]  value,
       output                        overflow_set
```

```
      );


wire up,down;
wire overflow_up_set;
wire overflow_down_set;


assign up   = cnt_mode ==1;
assign down = cnt_mode ==0;


always @(posedge clk or posedge rst)
begin
 if (rst)
    value = 0;
 else if (init_cnt)
    value = init;
 else
 begin
  if (en)
  begin
    if ( (up|free) & ~down )
    begin
      if (~free)
      begin
        if (value < max)
          value = value + 1;
        else if (value >= max)
          value = min;
      end
      else
      begin
        value = value + 1;
      end
    end
    else if ( (down|free) & ~up )
    begin
      if (~free)
      begin
        if (value > min)
          value = value - 1;
        else if (value <= min)
          value = max;
      end
      else
      begin
```

```
            value = value - 1;
          end
        end
        else
          value = value;
    end
  end
end

// as max/min value reached and enabled
// next cycle count restarts
// overflow set

assign overflow_up_set  = (value == {COUNTER_SIZE{1'b1}}) & en;
assign overflow_dwn_set = (value == {COUNTER_SIZE{1'b0}}) & en;

assign overflow_set =  cnt_mode?overflow_up_set:overflow_dwn_set;

endmodule
```

Elaboración propia

- Módulo input_block

```
module input_block(

input         clk_ext,
              clk,
              rst,
input  [2:0]  ps,
input         edge_mode,


output        clk_pulse

);

reg [7:0] clk_ext_past;
wire      pos_edge;
wire      neg_edge;

reg   [6:0] ps_counter;
wire      in_pulse;
wire      clk_ps;
```

```
reg        clk_ps_past;


//Input edge detector
always @(posedge clk)
if (rst)
  clk_ext_past = 8'b0;
else
  clk_ext_past = clk_ext_past << 1|clk_ext;

assign pos_edge = ( (&(~clk_ext_past[3:2])) & (&clk_ext_past[1:0]) );
assign neg_edge = ( (&(~clk_ext_past[1:0])) & (&clk_ext_past[3:2]) );

assign in_pulse = edge_mode?neg_edge:pos_edge;


//prescaler
always @(posedge clk or posedge rst)
begin
  if (rst)
    ps_counter = 0;
  else
    if (in_pulse)
      ps_counter = ps_counter + 1;
end

assign clk_ps    =  (ps == 3'd0)? in_pulse
                 :(ps == 3'd1)? ps_counter[0]
                 :(ps == 3'd2)? ps_counter[1]
                 :(ps == 3'd3)? ps_counter[2]
                 :(ps == 3'd4)? ps_counter[3]
                 :(ps == 3'd5)? ps_counter[4]
                 :(ps == 3'd6)? ps_counter[5]
                 :(ps == 3'd7)? ps_counter[6]:0;

//counter edge detector
always @(posedge clk)
if (rst)
  clk_ps_past = 8'b0;
else
  clk_ps_past = clk_ps;

assign  clk_pulse = ~clk_ps_past & clk_ps;

endmodule
```

Elaboración propia

- Módulo output_block

```
module output_block
#(
   parameter NUM_COMP  = 3
)
(
input       rst,
input       clk,
input       en,
input      [NUM_COMP-1:0] [7:0] match_value,
input      [7:0] counter_value,
input      [NUM_COMP-1:0] intr_en,
input      [NUM_COMP-1:0] trg_en,
input      [NUM_COMP-1:0] flag,
input       pwm_mode,
       inv,

output  timer_out,
       trigger,
output       [NUM_COMP-1:0] intr,
output       [NUM_COMP-1:0] match

);

 reg timer_out_internal;

 wire high_pwm;
 wire low_pwm;
 reg [NUM_COMP-1:0] match_past;
 wire [NUM_COMP-1:0] match_rise;
 wire [NUM_COMP-1:0] trigger_internal;

 assign trigger = |trigger_internal;

 genvar a;

 for (a = 0;a < NUM_COMP; a=a+1 )
 begin
   assign match[a] = (match_value[a] == counter_value);
   assign match_rise[a] = (~match_past[a] & match[a]);
 end

 for (a = 0;a < NUM_COMP; a=a+1 )
```

```
  begin
    assign trigger_internal[a] = (match[a] & trg_en[a]);
  end

  always @ (posedge clk or posedge rst)
  begin
    if (rst)
      match_past =0;
    else
      match_past = match;
  end
  //PWM

  //set if comp match 0
  //unset if comp match 1
  assign high_pwm = match_rise[0] & en & pwm_mode;
  assign low_pwm  = match_rise[1] & en & pwm_mode;

  always @ (posedge clk or posedge rst)
  begin
    if (rst)
      timer_out_internal =0;
    else
        if (high_pwm)
          timer_out_internal =1;
        else if (low_pwm)
          timer_out_internal =0;
        else
          timer_out_internal = timer_out_internal;
  end

  assign intr = (intr_en & flag );
  assign timer_out = inv?~timer_out_internal:timer_out_internal;

endmodule
```

- Módulo timer_registers

```
module timer_registers(
        input   logic      clk,
        rst,
        module_en,
        wr,

        input [7:0]  wdata,
        input [5:0]  addr,
        output   [7:0]  rdata,
        output reg overflow_int_en,
        output reg out_match_0_int_en,
        output reg out_match_1_int_en,
        output reg clock_select,
        output reg force_free,
        output reg count_mode,
        output reg [7:0] match_1_value,
        output reg [7:0] match_0_value,
        output reg       start,
        output reg [2:0] prescaler,

        output reg overflow_trg_en,
        output reg out_match_1_trg_en,
        output reg out_match_0_trg_en,

        input   overflow,
        input   match_0,
        input   match_1,

        output reg overflow_status_flag,
        output reg cnt_match_0_status_flag,
        output reg cnt_match_1_status_flag,

        output reg [7:0] count_init,
        output reg [7:0] count_min,
        output reg [7:0] count_max,

        output reg       edge_mode,
        output reg       pwm_mode,
        output reg       cnt_init_wr,
        output reg       inv

    );

localparam CTRL_REG_RST        = 8'h0;
```

```
localparam CTRL_IN_REG_RST      = 8'h0;
localparam CTRL_OUT_REG_RST     = 8'h0;
localparam STATUS_REG_RST       = 8'h0;
localparam CNT_INIT_REG_RST     = 8'h0;
localparam CNT_MIN_REG_RST      = 8'h0;
localparam CNT_MAX_REG_RST      = 8'hFF;
localparam CNT_REG_RST          = 8'h0;
localparam CNT_REG_MATCH_0_RST  = 8'h0;
localparam CNT_REG_MATCH_1_RST  = 8'h0;

localparam CTRL_REG_ADDR        = 8'h0;
localparam CTRL_IN_REG_ADDR     = 8'h1;
localparam CTRL_OUT_REG_ADDR    = 8'h2;
localparam STATUS_REG_ADDR      = 8'h4;
localparam CNT_INIT_REG_ADDR    = 8'h8;
localparam CNT_MIN_REG_ADDR     = 8'h9;
localparam CNT_MAX_REG_ADDR     = 8'hA;
localparam CNT_REG_ADDR         = 8'hB;
localparam CNT_REG_MATCH_0_ADDR = 8'hC;
localparam CNT_REG_MATCH_1_ADDR = 8'hD;

wire ctrl_reg_sel     = (addr == CTRL_REG_ADDR);
wire ctrl_in_reg_sel  = (addr == CTRL_IN_REG_ADDR);
wire ctrl_out_reg_sel = (addr == CTRL_OUT_REG_ADDR);
wire status_reg_sel   = (addr == STATUS_REG_ADDR);
wire cnt_init_reg_sel = (addr == CNT_INIT_REG_ADDR);
wire cnt_min_reg_sel  = (addr == CNT_MIN_REG_ADDR);
wire cnt_max_reg_sel  = (addr == CNT_MAX_REG_ADDR);
wire cnt_reg_sel      = (addr == CNT_REG_ADDR);
wire cnt_match_1_sel  = (addr == CNT_REG_MATCH_1_ADDR);
wire cnt_match_0_sel  = (addr == CNT_REG_MATCH_0_ADDR);

wire        wr_en;
wire        rd_en;
reg  [7:0]  rdata_mux_out;

assign wr_en = module_en &  wr;
assign rd_en = module_en & ~wr;

//register ctrl
wire  [7:0] ctrl_reg;
assign ctrl_reg = {
          force_free,
          overflow_int_en,
          out_match_1_int_en,
          out_match_0_int_en,
          clock_select,
```

```
            1'b0,
            count_mode,
            start
        };
always @(posedge clk or posedge rst)
begin
    if (rst)
        start = CTRL_REG_RST[0];
    else
        if (wr_en & ctrl_reg_sel)
            start = wdata[0];
end
always @(posedge clk or posedge rst)
begin
    if (rst)
        count_mode = CTRL_REG_RST[1];
    else
        if (wr_en & ctrl_reg_sel)
            count_mode = wdata[1];
end
always @(posedge clk or posedge rst)
begin
    if (rst)
        clock_select = CTRL_REG_RST[3];
    else
        if (wr_en & ctrl_reg_sel)
            clock_select = wdata[3];
end
always @(posedge clk or posedge rst)
begin
    if (rst)
        overflow_int_en = CTRL_REG_RST[4];
    else
        if (wr_en & ctrl_reg_sel)
            overflow_int_en = wdata[4];
end
always @(posedge clk or posedge rst)
begin
    if (rst)
        out_match_0_int_en = CTRL_REG_RST[5];
    else
        if (wr_en & ctrl_reg_sel)
            out_match_0_int_en = wdata[5];
end
always @(posedge clk or posedge rst)
begin
    if (rst)
```

```
        out_match_1_int_en = CTRL_REG_RST[6];
    else
        if (wr_en & ctrl_reg_sel)
            out_match_1_int_en = wdata[6];
end
always @(posedge clk or posedge rst)
begin
    if (rst)
        force_free = CTRL_REG_RST[7];
    else
        if (wr_en & ctrl_reg_sel)
            force_free = wdata[7];
end

//register ctrl_in

wire [7:0] ctrl_in_reg;


assign ctrl_in_reg = {
            1'b0,
            prescaler,
            3'b0,
            edge_mode
        };

always @(posedge clk or posedge rst)
begin
    if (rst)
        edge_mode = CTRL_IN_REG_RST[0];
    else
        if (wr_en & ctrl_in_reg_sel)
            edge_mode = wdata[0];
end

always @(posedge clk or posedge rst)
begin
    if (rst)
        prescaler = CTRL_IN_REG_RST[6:4];
    else
        if (wr_en & ctrl_in_reg_sel)
            prescaler = wdata[6:4];
end


//register ctrl_out
```

```
wire [7:0] ctrl_out_reg;


assign ctrl_out_reg = {
          1'b0,
          out_match_1_trg_en,
          out_match_0_trg_en,
          overflow_trg_en,
          2'b0,
          inv,
          pwm_mode
       };

always @(posedge clk or posedge rst)
begin
   if (rst)
      pwm_mode = CTRL_OUT_REG_RST[0];
   else
      if (wr_en & ctrl_out_reg_sel)
         pwm_mode = wdata[0];
end
always @(posedge clk or posedge rst)
begin
   if (rst)
      inv = CTRL_OUT_REG_RST[1];
   else
      if (wr_en & ctrl_out_reg_sel)
         inv = wdata[1];
end

always @(posedge clk or posedge rst)
begin
   if (rst)
      overflow_trg_en = CTRL_OUT_REG_RST[4];
   else
      if (wr_en & ctrl_out_reg_sel)
         overflow_trg_en = wdata[4];
end
always @(posedge clk or posedge rst)
begin
   if (rst)
      out_match_0_trg_en = CTRL_OUT_REG_RST[5];
   else
      if (wr_en & ctrl_out_reg_sel)
         out_match_0_trg_en = wdata[5];
end
always @(posedge clk or posedge rst)
```

```
begin
   if (rst)
      out_match_1_trg_en = CTRL_OUT_REG_RST[6];
   else
      if (wr_en & ctrl_out_reg_sel)
         out_match_1_trg_en = wdata[6];
end

//register status

wire  [7:0] status_reg;

always @(posedge clk or posedge rst)
begin
   if (rst)
      overflow_status_flag = 0;
   else
      if (overflow)
         overflow_status_flag = 1;
      else if (wr_en & status_reg_sel & wdata[0])
         overflow_status_flag = 0;

end

always @(posedge clk or posedge rst)
begin
   if (rst)
      cnt_match_0_status_flag = 0;
   else
      if (match_0 & start)
         cnt_match_0_status_flag = 1;
      else if (wr_en & status_reg_sel & wdata[1])
         cnt_match_0_status_flag = 0;

end

always @(posedge clk or posedge rst)
begin
   if (rst)
      cnt_match_1_status_flag = 0;
   else
      if (match_1 & start)
         cnt_match_1_status_flag = 1;
      else if (wr_en & status_reg_sel & wdata[2])
         cnt_match_1_status_flag = 0;

end
```

```
assign status_reg =
{5'b0,cnt_match_1_status_flag,cnt_match_0_status_flag,overflow_status_flag};

//register count_init

always @(posedge clk or posedge rst)
begin
   if (rst)
      count_init = CNT_INIT_REG_RST;
   else
      if (wr_en & cnt_init_reg_sel)
        count_init = wdata;
end


always @(posedge clk or posedge rst)
begin
   if (rst)
     cnt_init_wr =0;
   else if  (wr_en & cnt_init_reg_sel)
    cnt_init_wr = 1;
   else
    cnt_init_wr = 0;
end


//register count_min

always @(posedge clk or posedge rst)
begin
   if (rst)
      count_min = CNT_MIN_REG_RST;
   else
      if (wr_en & cnt_min_reg_sel)
         count_min = wdata;
end

//register count_max

always @(posedge clk or posedge rst)
begin
   if (rst)
      count_max = CNT_MAX_REG_RST;
   else
      if (wr_en & cnt_max_reg_sel)
         count_max = wdata;
```

```
end


//register count

logic [7:0] counter;
wire [7:0] counter_reg;
always @(posedge clk or posedge rst)
begin
   if (rst)
      counter = CNT_REG_RST;
   else
      if (wr_en & cnt_reg_sel)
         counter = wdata;
end

always @(posedge clk or posedge rst)
begin
   if (rst)
      counter = CNT_REG_RST;
   else
      if (wr_en & cnt_reg_sel)
         counter = wdata;
end


assign counter_reg = counter;


//register output match_0 value
wire [7:0] cnt_match_0_reg;
always @(posedge clk or posedge rst)
begin
   if (rst)
      match_0_value = CNT_REG_MATCH_0_RST;
   else
      if (wr_en & cnt_match_0_sel)
         match_0_value = wdata;
end
assign cnt_match_0_reg = match_0_value;
//register output match_1 value
wire [7:0] cnt_match_1_reg;
always @(posedge clk or posedge rst)
begin
   if (rst)
      match_1_value = CNT_REG_MATCH_1_RST;
   else
```

```
        if (wr_en & cnt_match_1_sel)
            match_1_value = wdata;
end
assign cnt_match_1_reg = match_1_value;

//read register
always @ (*)
begin : MUX
   case(addr )
      CTRL_REG_ADDR : rdata_mux_out = ctrl_reg;
      CTRL_IN_REG_ADDR : rdata_mux_out = ctrl_in_reg;
      CTRL_OUT_REG_ADDR : rdata_mux_out = ctrl_out_reg;
      STATUS_REG_ADDR : rdata_mux_out = status_reg;
      CNT_REG_ADDR : rdata_mux_out = counter_reg;
      CNT_INIT_REG_ADDR : rdata_mux_out = count_init;
      CNT_MIN_REG_ADDR : rdata_mux_out = count_min;
      CNT_MAX_REG_ADDR : rdata_mux_out = count_max;
      CNT_REG_MATCH_1_ADDR : rdata_mux_out = cnt_match_1_reg;
      CNT_REG_MATCH_0_ADDR : rdata_mux_out = cnt_match_0_reg;
      default: rdata_mux_out = 8'b0;
   endcase
end
assign rdata = rdata_mux_out & {8{rd_en}};


endmodule
```

Elaboración propia

- Módulo principal d_ip_timer

```
module d_ip_timer(

  input  clk,
  input  rst_b,
  input  [5:0] addr,
  input  wr_en,
  input  mod_en,
  input  timer_in,
  input  [7:0] wdata,
  output [7:0] rdata,
  output timer_out,
  output trigger,
  output overflow_int,
         comp_1_match_int,
         comp_0_match_int

);

localparam CNTR_ADDR=6'h04;
localparam COUNTER_SIZE=8;
localparam NUM_COMP=2;

wire                    overflow_trg_en;
wire                    out_match_1_trg_en;
wire                    out_match_0_trg_en;
wire                    overflow_int_en;
wire                    out_match_1_int_en;
wire                    out_match_0_int_en;
wire                    clock_select;
wire                    count_mode;
wire                    start;
wire [COUNTER_SIZE-1:0]  counter_value;
wire                    counter_overflow;

wire [COUNTER_SIZE-1:0]  match_1_value;
wire [COUNTER_SIZE-1:0]  match_0_value;

wire [COUNTER_SIZE-1:0]  count_min;
wire [COUNTER_SIZE-1:0]  count_max;
wire [COUNTER_SIZE-1:0]  count_init;

wire overflow_status_flag;
```

```
wire cnt_match_0_status_flag;
wire cnt_match_1_status_flag;
wire cnt_match_0_status_flag_set;
wire cnt_match_1_status_flag_set;


reg start_1;
wire start_rise;


wire up_count;
wire down_count;
wire free_mode;
wire clk_pulse;
wire pwm_mode;
wire edge_mode;
wire inv;
wire trigger_int;
wire enable_operation;
wire [2:0] prescaler;

 //decode
 assign  ext_clock_select = clock_select;

 //edge detector
 always @ (posedge clk) begin
   start_1 <= start;
 end
 assign start_rise = start & ~start_1;


 assign enable_operation = clk_pulse & ext_clock_select & start| ~ext_clock_select
& start;

 timer_registers timer_registers(
 .clk (clk),
 .rst (~rst_b),
 .module_en (mod_en),
 .wr (wr_en),
 .wdata (wdata),
 .addr (addr),
 .overflow_int_en(overflow_int_en),
 .out_match_1_int_en(out_match_1_int_en),
 .out_match_0_int_en(out_match_0_int_en),
 .overflow_trg_en(overflow_trg_en),
 .out_match_1_trg_en(out_match_1_trg_en),
 .out_match_0_trg_en(out_match_0_trg_en),
```

```
.clock_select(clock_select),
.count_mode(count_mode),
.force_free (free_mode),
.count_init(count_init),
.count_min(count_min),
.count_max(count_max),
.match_1_value(match_1_value),
.match_0_value(match_0_value),
.overflow_status_flag(overflow_status_flag),
.cnt_match_0_status_flag(cnt_match_0_status_flag),
.cnt_match_1_status_flag(cnt_match_1_status_flag),
.overflow(counter_overflow),
.match_0(cnt_match_0_status_flag_set),
.match_1(cnt_match_1_status_flag_set),
.start(start),
.inv(inv),
.prescaler(prescaler),
.pwm_mode(pwm_mode),
.edge_mode(edge_mode),
.cnt_init_wr(cnt_init_wr),
.rdata (rdata)
);

timer_counter #(
  .COUNTER_SIZE (COUNTER_SIZE)
)
timer_counter(
  .clk(clk),
  .en(enable_operation),
  .rst(~rst_b),
  .cnt_mode ( ~count_mode ),
  .value(counter_value),
  .min (count_min),
  .max (count_max),
  .init (count_init),
  .free (free_mode),
  .init_cnt(cnt_init_wr),
  .overflow_set(counter_overflow)
);

assign overflow_int = overflow_int_en && overflow_status_flag;

input_block  input_block (

.clk       (clk),
.clk_ext   (timer_in),
.rst       (~rst_b),
```

```
  .edge_mode (edge_mode),
  .clk_pulse (clk_pulse),
  .ps        (prescaler)

  );

  output_block  #(
    .NUM_COMP (NUM_COMP)
  )    output_block (

  .clk           (clk),
  .rst           (~rst_b),
  .counter_value (counter_value),
  .pwm_mode      (pwm_mode),
  .timer_out     (timer_out),
  .match_value   ({ match_1_value,
                   match_0_value  }),
  .match         ({ cnt_match_1_status_flag_set,
                   cnt_match_0_status_flag_set}),
  .flag          ({ cnt_match_1_status_flag,
                   cnt_match_0_status_flag}),
  .intr_en       ({ out_match_1_int_en,
                   out_match_0_int_en }),
  .trg_en        ({ out_match_1_trg_en,
                   out_match_0_trg_en   }),
  .intr          ({ comp_1_match_int,
                   comp_0_match_int   }),
  .inv           (inv),
  .en            (enable_operation),
  .trigger       (trigger_int)
  );

  assign trigger = trigger_int | (counter_overflow & overflow_trg_en);

endmodule
```

Elaboración propia

ANEXO 8. Hoja de características

8-bit Timer Module

Document Information:
Author: Jose Cueva
Revision: 0.2
Modify date: 11 November 2018

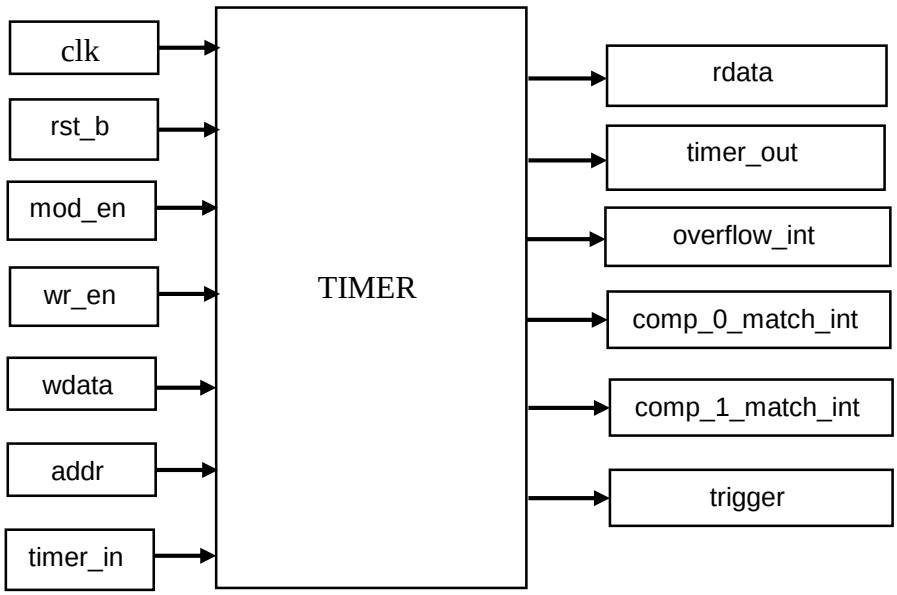
1 Introduction

The 8-bit timer is a general purpose simplified counter module, it includes a bidirectional counter, an interruption generation block, input pulse or external clock mode count mode, provides a method to generate a PWM signal and a trigger signal generated according timer events.

2 Overview

The counter has 2 operation modes, normal and inverse counter modes with initial value configuration register and match value registers can be easily adjusted to generate interruption signals periodically.
PWM can be generated using Match count 0 and Match count 1 timer events, trigger signal can be triggered according same events to produce a 1 cycle pulse, it can be used to start some other block actions as ADC conversions or UART transmissions.

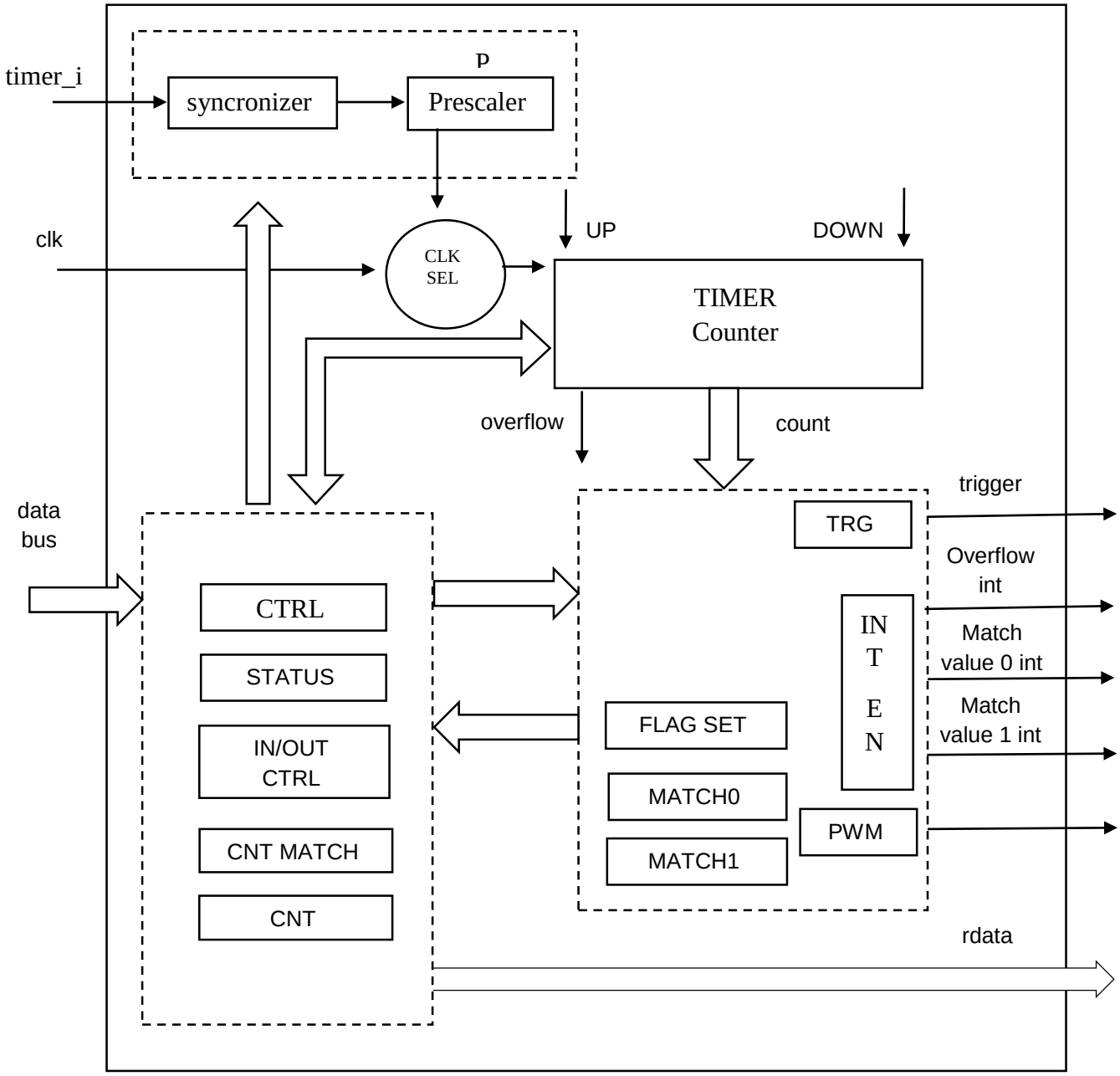
3 Pin diagram



Timer has an interface for register access and the clock and reset inputs, it also has the timer_in that can be connected to a pulse input pad or to an external clock signal.

Output interface includes the read data (rdata) providing register access read value, 3 interruption signals, timer_out is the PWM signal output and trigger is shared between the 3 events: overflow, match 0, match 1.

4 Block diagram



4.1 Counter

Counter is bidirectional, Up or Down mode, it supports free running counting mode and restricted MIN to MAX count modes.

4.2 Input block

Includes edge detectors and prescaler components.

4.2.1 Prescaler

Prescaler can divide the input frequency by 2,4,8,16,32,64,128 factor

4.2.2 Synchronizer

Takes input signal and passes it by two flip flops to synchronize with Timer clock.

4.3 Output block

Includes interruption/flag, PWM, trigger generation.

4.4 Registers block

Internal register block contains all the register fields in the same component.

5 Features

5.1 Prescaler

An external clock signal can be divided by half factors to get longer times.

5.2 Input pulse counter

An input signal rising or falling edge can be used to increment or decrement the counter.

5.3 External clock support

External clock signal can be connected to timer_in input and can be used to increment/decrement the counter, the frequency can be divided according the prescaler.

5.4 Interruption generation

Interruptions can be generated for the next timer events:

5.5 Overflow

Interruption signal can be generated using the control register configuration OVF_EN enable bit, every time there is a transition from max value (Actually 0xFF) to minimal counter value, 0x0 if CNT_INIT register not configured otherwise the value configured at CNT_INIT.

5.6 Match 0,1 interruption

Interruption signals are generated using the control register configuration CTN_M1_EN and CTN_M2_EN enable bits, every match between counter actual value and Match value configured in registers CNT_M1 and CNT_M2 triggers the respective interruption signal.

5.7 Counter Initialization

The counter initial value is 0 by default but it can be replaced for a user defined value writing at CNT_INT register

5.8 Counter inverse mode

Counter direction is defined by control register CTRL configuration bit MODE, default mode is up count mode MODE=0 if this bit is set MODE=1 the count direction will be reversed.
This configuration is expected to be done during no operation mode, control register bit is with default value START=0.

5.9 Trigger generation

Counter generates a trigger signal according overflow or match 0/1 event, depending which is selected.

6 Signal description

Below tables show the input and output signals

6.1 Input signals

Name	Size (bits)	Description	Notes
clk	1	Clock input	
rst_b	1	reset	Active low
addr	6	register address	
mod_en	1	module is selected for register access	Active High
wr_en	1	1: module access for writing 0: module access for reading	
timer_in	1	External clock or pulse input	
wdata	8	register write data	

Table 1

Output signals

Name	Size (bits)	Description	Notes
rdata	8	Clock input	
overflow_int	1	Timer overflow event interruption	Active High
comp_0_match_int	1	Timer comparator 0 match event interruption	Active High
comp_1_match_int	1	Timer comparator 1 match event interruption	Active High
timer_out	1	PWM output	
trigger	1	Trigger output	

Table 2

Memory map and register definition

6.2 Register summary

Timer module has 6 configuration registers listed in the table below

Offset	Register name	Description	Width (bits)	Access	Reset value
0x0	CTRL	Timer enable/configuration	8	R/W	0
0x1	CTRL_IN	Control input	8	R/W	0
0x2	CTRL_OUT	Control output	8	R/W	0
0x4	STATUS	Timer status	8	R/W	0
0x8	CNT_INIT	Timer initialization	8	R/W	0
0x9	CNT_MIN	Counter min	8	R/W	0
0xA	CNT_MAX	Counter max	8	R/W	0
0xB	CNT	Actual counter value	8	RO	0
0xC	CNT_M1	Counter match value 0	8	R/W	0
0xD	CNT_M2	Counter match value 1	8	R/W	0

6.3 Register description

6.3.1 Control register CTRL

Control register configures main features as operation mode, clock source and has the start bit to enable counter operation.

Diagram

Bits	7	6	5	4	3	2	1	0
R	FREE	M1_INT	M0_INT	OVF_INT	CLK_SEL		CNT MODE	START
W								
Reset	0	0	0	0	0	0	0	0

Fields

Field	Function
START	Starts counter operation, set this bit after configuration
FREE	• 0 Non-free run count mode, it considers MAX MIN values • 1 Force free run count, don't apply MAX MIN configuration
CNT MODE	• 0 Count up • 1 Count down
CLK_SEL	• 0 System clock • 1 External clock
M0_INT	Enables interruption for match between counter register CNT value with counter match 0 register value: M0F
M1_INT	Enables interruption for match between counter register CNT value with counter match 1 register value: M1F
OVF_INT	Enables interruption for match between counter register CNT value with max counter register value 0xFF: STATUS OVF

6.3.2 Control register input CTRL_IN

Diagram

Bits	7	6	5	4	3	2	1	0
R		PS						IN_EDGE
W								
Reset	0	0	0	0	0	0	0	0

Fields

Field	Function
IN_EDGE	Select between positive or negative edge to trigger counter when input counter mode is selected • 0 Positive edge

	• 1 Negative edge
PS	Prescaler factor value for external clock 000 - Divide by 1 001 - Divide by 2 010 - Divide by 4 011 - Divide by 8 100 - Divide by 16 101 - Divide by 32 110 - Divide by 64 111 - Divide by 128

6.3.3 Control register output CTRL_OUT

Diagram

Bits	7	6	5	4	3	2	1	0
R		M1 TRG	M0 TRG	OVF TRG			INV	PWM
W								
Reset	0	0	0	0	0	0	0	0

Fields

Field	Function
PWM	Output pin timer_out toggles according counter MIN and MAX value, when there is a match of MIN value and counter CNT value output is set, when there is a match of MAX value and CNT value output is cleared. • 0 PWM generation disabled • 1 PWM generation enabled
INV	Invert output • Output is not inverted • Output is inverted
OVF TRG	Overflow trigger

	When enabled a pulse is generated when there is a counter overflow
M0 TRG	Match 0 output trigger When enabled a pulse is generated when there is a counter match 0 event
M1 TRG	Match 1 output trigger When enabled a pulse is generated when there is a counter match 1 event

6.3.4 Status register STATUS

Status register shows flags for counter match values and overflow events

Diagram

Bits	7	6	5	4	3	2	1	0
R						M1F	M0F	OVF
W1C*								
Reset	0	0	0	0	0	0	0	0

Fields

Field	Function
OVF	Overflow Flag • 1 Counter overflow since the last time this bit has been cleared • 0 Counter overflow has not occurred since last time this bit cleared About Clearing this flag: Writing 1 to this position clears the flag and the interruption signals is consequently disabled Writing 0 to this position does nothing
M0F	Match Counter 0 Flag • 1 Counter value has matched with CTN_M0 register value since last time this bit has been cleared • 0 Counter value did not match or matches with CTN_M0 register value since last time this bit has been cleared

	About Clearing this flag: Writing 1 to this position clears the flag and the interruption signals is consequently disabled Writing 0 to this position does nothing
M1F	Match Counter 1 Flag • 1 Counter value has matched with CTN_M1 register value since last time this bit has been cleared • 0 Counter value did not match or matches with CTN_M1 register value since last time this bit has been cleared About Clearing this flag: Writing 1 to this position clears the flag and the interruption signals is consequently disabled Writing 0 to this position does nothing

* **W1C:** Write 1 to clear

6.3.5 Counter initialization register CNT_INIT

Counter initial value can be initialized using this register

Diagram

Bits	7	6	5	4	3	2	1	0
R	INIT_VAL							
W								
Reset	0	0	0	0	0	0	0	0

Fields

Field	Function
INIT_VAL	Initial Value This value defines the initial value for counter operation

6.3.6 Counter minimal value register CNT_MIN

Counter value minimal count value register

Diagram

Bits	7	6	5	4	3	2	1	0
R	MIN_VAL							
W								
Reset	0	0	0	0	0	0	0	0

Fields

Field	Function
MIN_VAL	Minimal Value When max value is reached counter returns to this value when non-free running mode is selected

6.3.7 Counter maximum value register CTN_MAX

Counter value maximum count value register

Diagram

Bits	7	6	5	4	3	2	1	0
R	MAX_VAL							
W								
Reset	1	1	1	1	1	1	1	1

Fields

Field	Function
MAX_VAL	Maximum Value Counter goes up until this value is reached when non-free running mode is selected

6.3.8 Counter register CNT

Contains the actual counter count value

Diagram

Bits	7	6	5	4	3	2	1	0
R	CNT_VAL							
W								
Reset	0	0	0	0	0	0	0	0

Fields

Field	Function
CNT_VAL	Counter Value The actual timer's counter value **Note**: This register can only be read, it is not possible to modify this value

6.3.9 Counter Match 0 CNT_M0

Contains the match value 0

Diagram

Bits	7	6	5	4	3	2	1	0
R	CNT_MATCH							
W								
Reset	0	0	0	0	0	0	0	0

Fields

Field	Function
CNT_MATCH	Counter match value 0 This value is used to compare with counter value and set M0F flag in STATUS register and the corresponding interruption signal when enabled interruption generation.

6.3.10 Counter Match 1 CNT_M1

Contains the match value 1

Diagram

Bits	7	6	5	4	3	2	1	0
R	CNT_MATCH							
W								
Reset	0	0	0	0	0	0	0	0

Fields

Field	Function
CNT_MATCH	Counter match value 1 This value is used to compare with counter value and set M1F flag in STATUS register and the corresponding interruption signal when enabled interruption generation.

7 Functional description

Timer can be configured for the next operation modes

7.1 Free counter mode

Timer counts from 0x0 to 0xFF and starts again indefinitely

7.2 Normal count mode

Timer is configured with an initial count value and a top count maximum value, count starts from this initial value and runs up until maximum value is reached, when maximum value is reached count starts again.

7.3 Inverse count mode

Timer is configured in the same way as normal count mode, but the count starts from maximum value and continues counting down until reaches minimum value, when minimum value is reached count starts again.

Note: when maximum value is not configured (0x0) the maximum value is 0xFF

7.4 Input edge counter mode

Timer counts according input pin pulses, negated edge or positive edge trigger can be selected according control register 2.

7.5 Prescaler for external clock

Prescaler provides a way to generate a frequency divider, it is implemented to operate 2 factor proportional.

7.6 PWM generation

Using Match 0 and Match 1 registers it is posible to generate a PWM signal at timer_out

7.7 Trigger generation

A trigger signal is generated based in the timer events, overflow and counter match value events can be configured to generate a trigger signal.

8 Initialization & Configuration

8.1 Free run operation

- Write to the Count match 0 and Count match 1 registers according desired match values
- Set the Control register bits M0_INT, M1_INT and OVF_INT if required
- Set the Control register bit START bit

Counter starts running until reach max value 0xFF then starts again.

8.2 Normal count operation

- Write to the count match 0 and count match 1 registers according desired match values
- Write to counter initialization register and counter max value register according expected values.
- Set the control register bits M0_INT, M1_INT and OVF_INT if required
- Set the control register bit START bit

8.3 Inverse count operation

- Write to the Count match 0 and Count match 1 registers according desired match values
- Write to counter initialization register and counter max value register according expected values.
- Set the control register bits M0_INT, M1_INT and OVF_INT if required

- Set the control register bit MODE
- Set the control register bit START bit

8.4 Interruptions

8.4.1 Overflow interruption

Overflow interruption is generated when STATUS register OVF flag is set and the OVF_INT bit is set in Control register.

8.4.2 Counter match 0 interruption

Counter match 0 interruption is generated when STATUS register M0F flag is set and the Control register CTRL bit M0F_INT is enabled.

8.4.3 Counter match 1 interruption

Counter match 1 interruption is generated when STATUS register M0F flag is set and the Control register CTRL bit M1F_INT is enabled.

9 Counter/Event time diagrams

9.1 Free run count mode

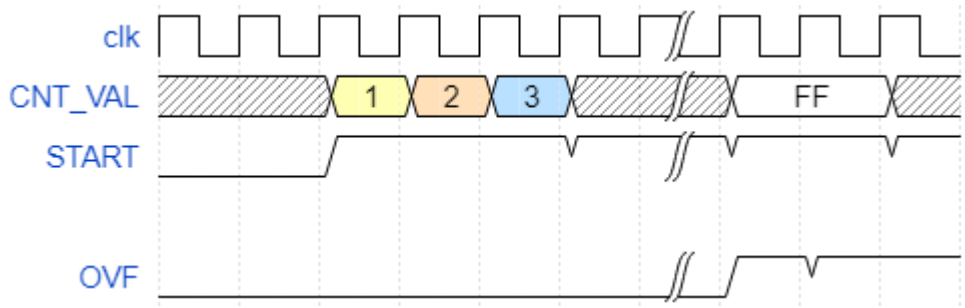


Figure 1

Normal count mode

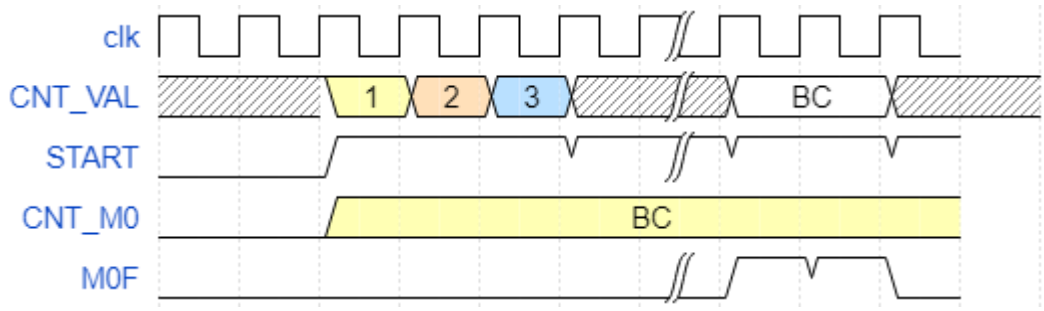


Figure 2

Reverse count mode

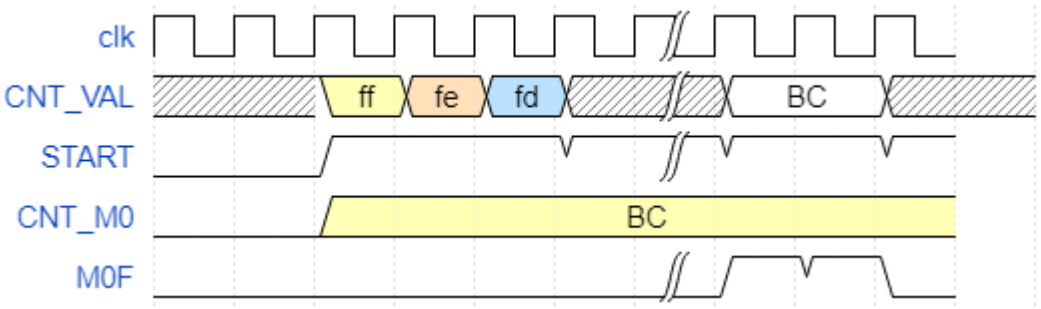


Figure 3

Additional

Folders organization

9.2 Host

This design files can be accessed/copied or cloned from Github.
https://github.com/joselcuevam/timer

9.3 Structure

Files are organized according the next parts

1.1.1.1 RTL

Contains all design data files, top design RTL and sub components.

1.1.1.2 TESTBENCH

Contains testbench related files, includes top testbench instantiation, test components as macros, modules, tasks etc.

1.1.1.3 doc

Includes this file and additional documents

1.1.1.4 make

Includes scripts for compilation, elaboration and simulation.

1.1.1.5 tool_data

Include files used by tools.

10 Simulation time diagrams

10.1.1 Free run mode starting

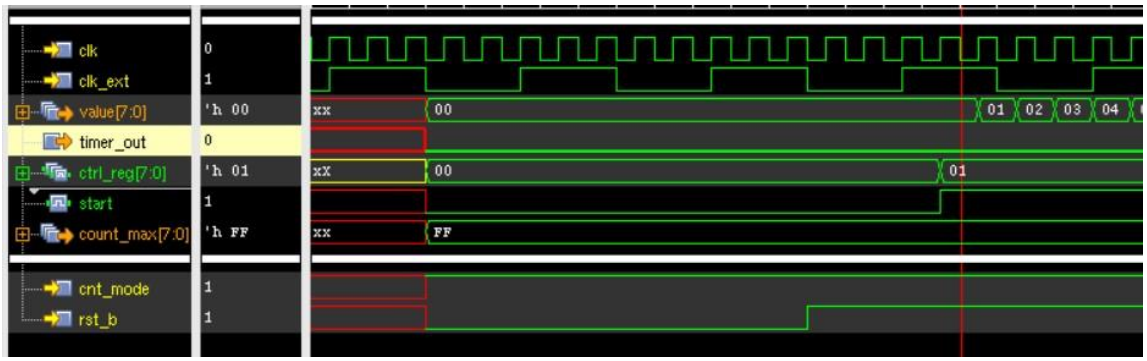


Figure 4 free run mode starting

10.1.2 Free run mode overflow

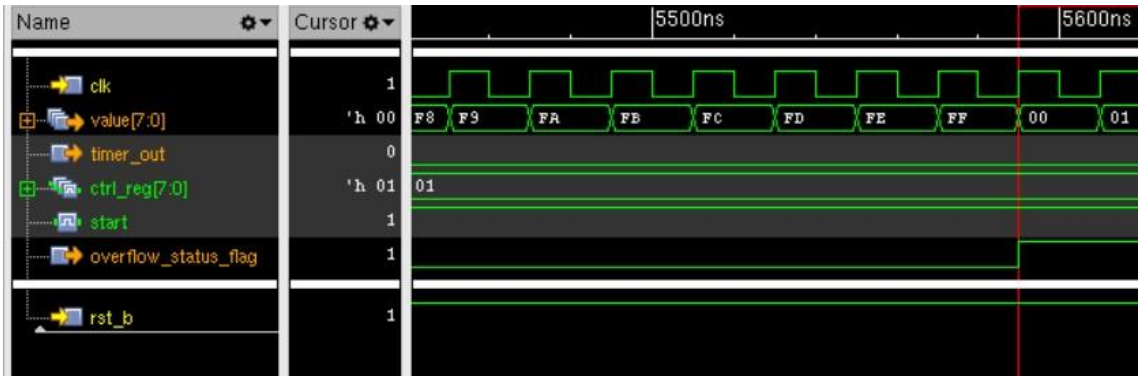


Figure 5 timer free run mode overflow

10.1.3 Free run mode (down count mode) starting

For this mode configure first CNT_INIT = FF then the CNT_MODE bit before setting START bit.

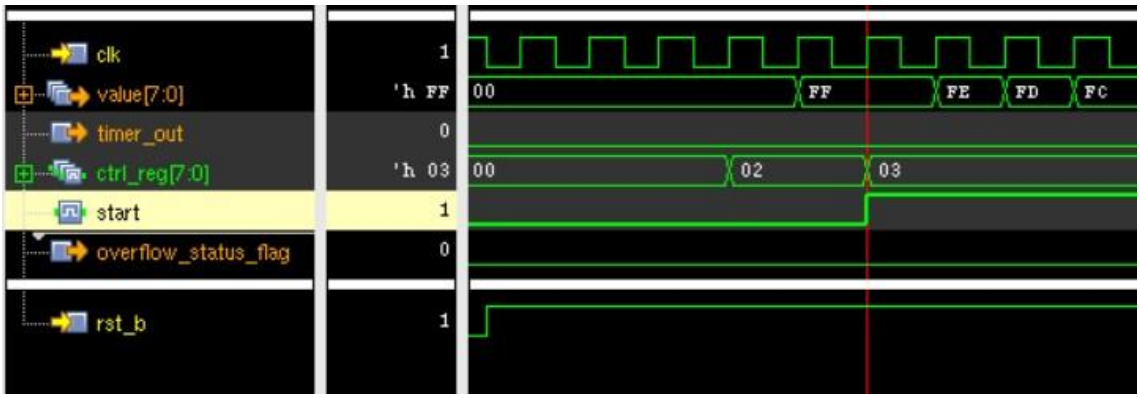


Figure 6 free run, down count mode starting

10.1.4 Free run mode (down count mode) overflow

For this mode configure first CNT_INIT == FF value then the CNT_MODE bit before setting START bit.

Figure 7 Free run, down count mode overflow

10.1.5 Up count mode starting

MAX and MIN count values have been configured before starting counter

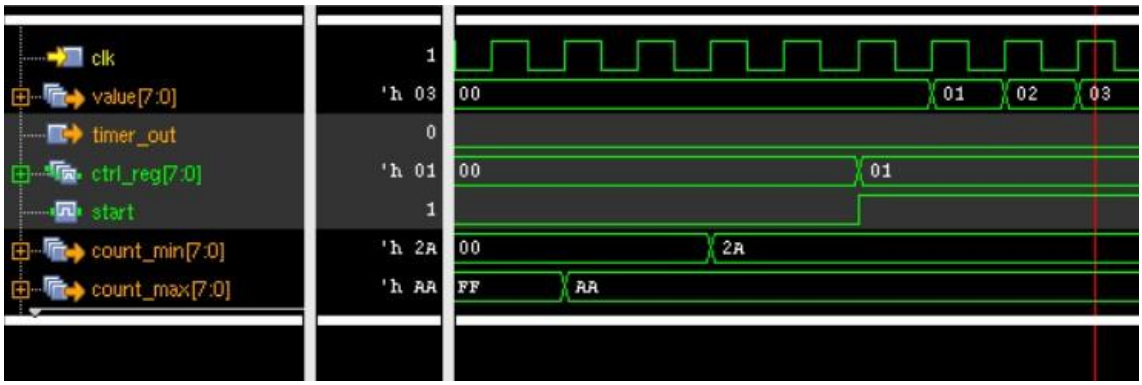


Figure 8 Up count starting

10.1.6 Up count mode returns

When MAX value is reached counter returns to MIN value

Figure 9 Up count mode returns.

10.1.7 Down count mode starting

MAX and MIN values are configured, and initial value is set same as MAX value to avoid the counter starting with initial value of zero.

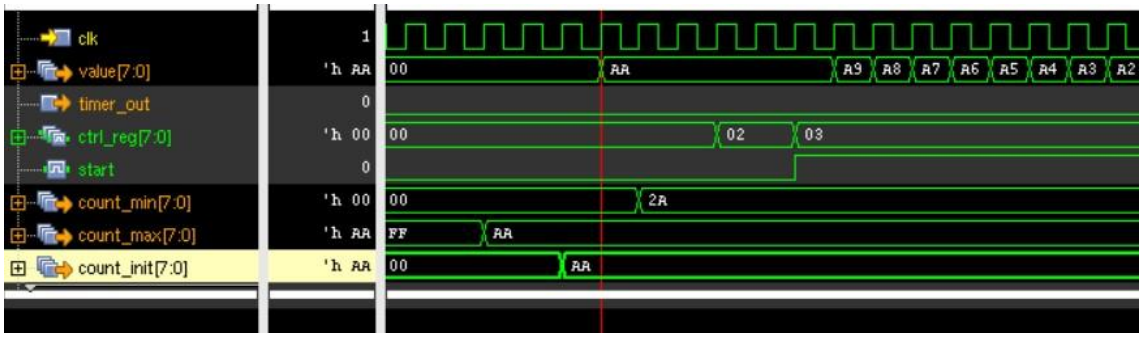


Figure 10 Down count mode starting

10.1.8 Down count mode returns

Counter Wraps from MIN to MAX value and count is restarted

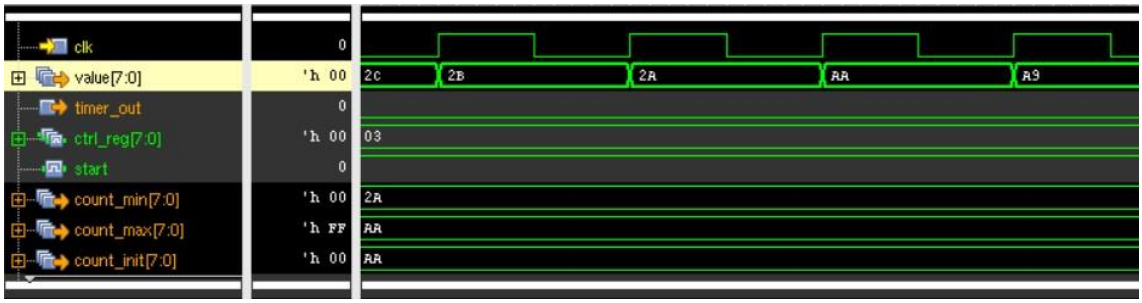


Figure 11 Down count mode returns

10.1.9 Counter value initialization

Counter initial value is set according user needs with the CNT_IN register, by default initial count value is zero.

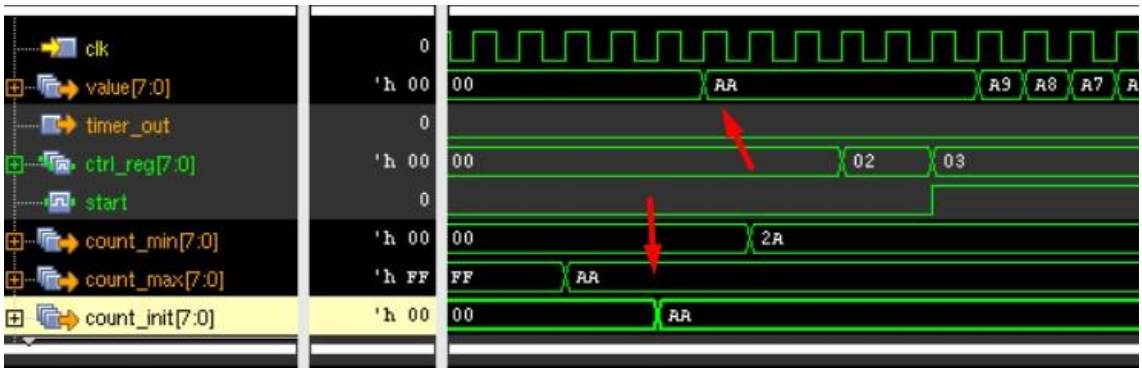


Figure 12 Counter value initialization

10.1.10 Input pulses counter

In this mode counter increases according pulses in the input pin, max value it can go is 10.

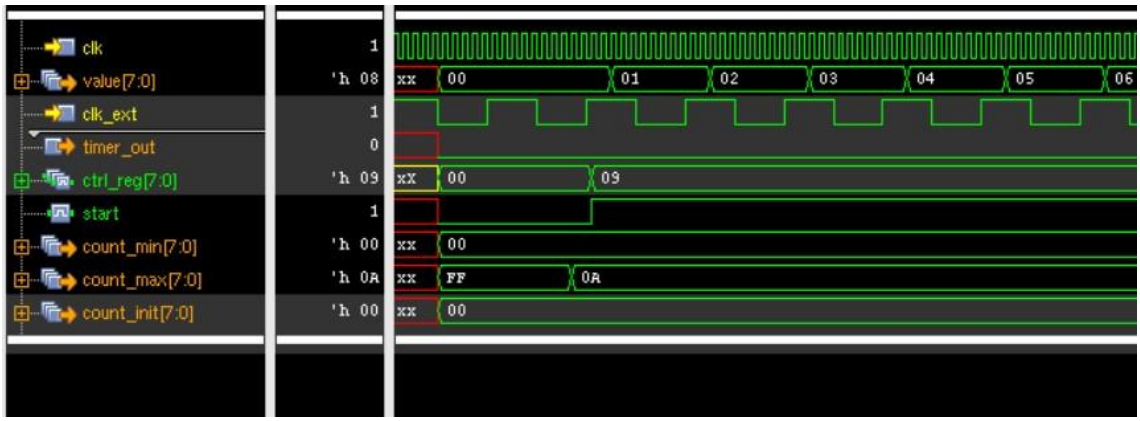


Figure 13 Counter triggered by input pulses

10.1.11 PWM generation

Matching CNT_M0 and CNT_M1 toggles output generating the PWM waveform, match 0 event sets output and match 1 events clears output signals.

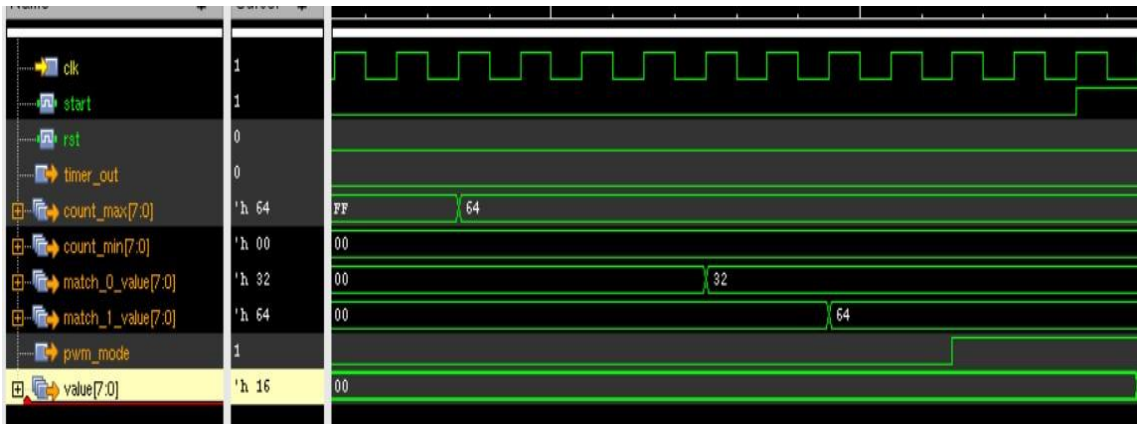


Figure 14 Register values configured according

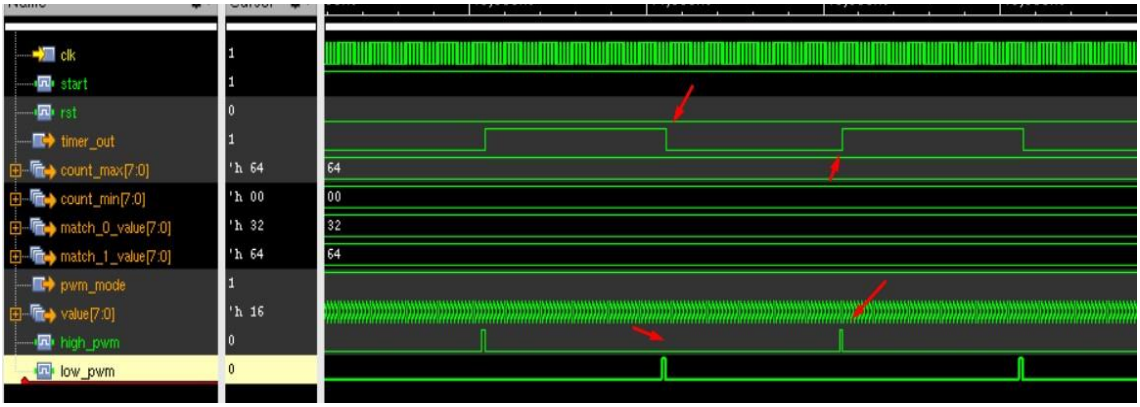


Figure 15 Timer out is set/clear according matches

10.1.12 Counter match flags

The next graphic sow the time diagram Flags are set

Figure 26 Match 0 sets flag

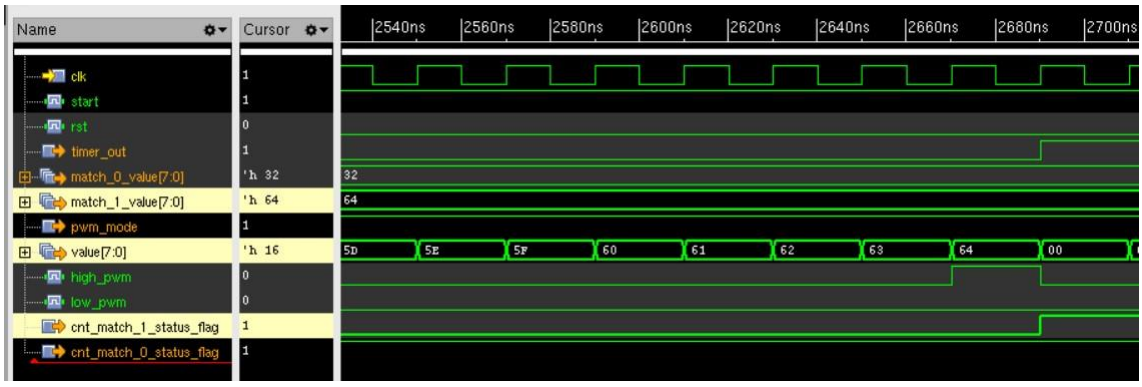


Figure 37 Match 1 sets flag

10.1.13 Interruption generation

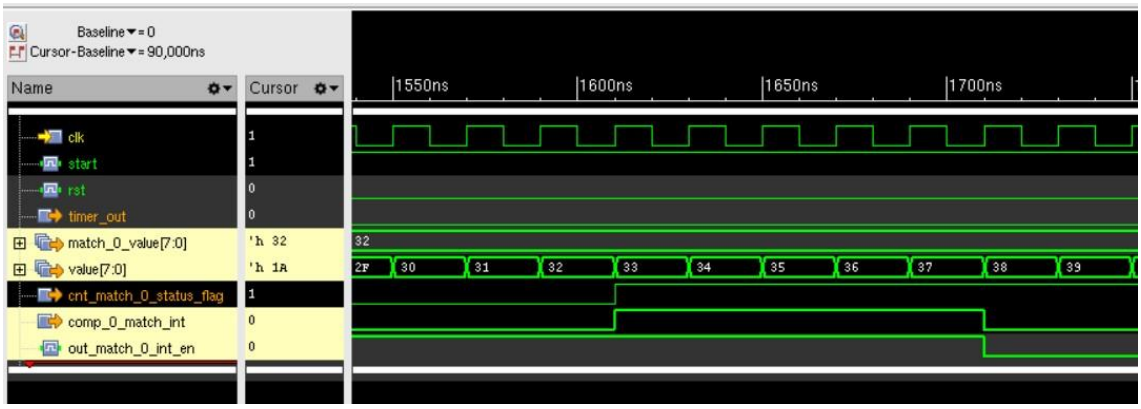


Figure 48 Match value interruption

10.1.14 Overflow interruption

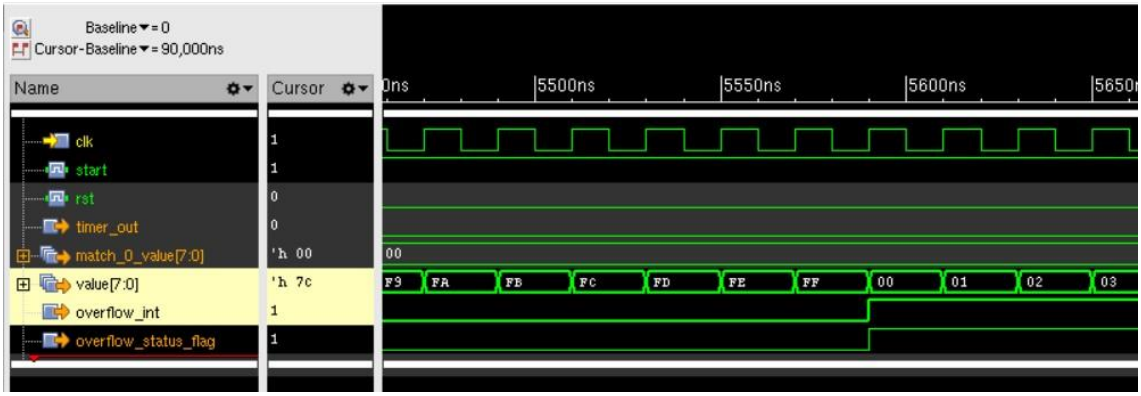


Figure 59 Overflow interruption

10.1.15 Input Prescaler

Figure 20 Prescaler enabled with PS=3

10.1.16 Trigger generation free run mode overflow

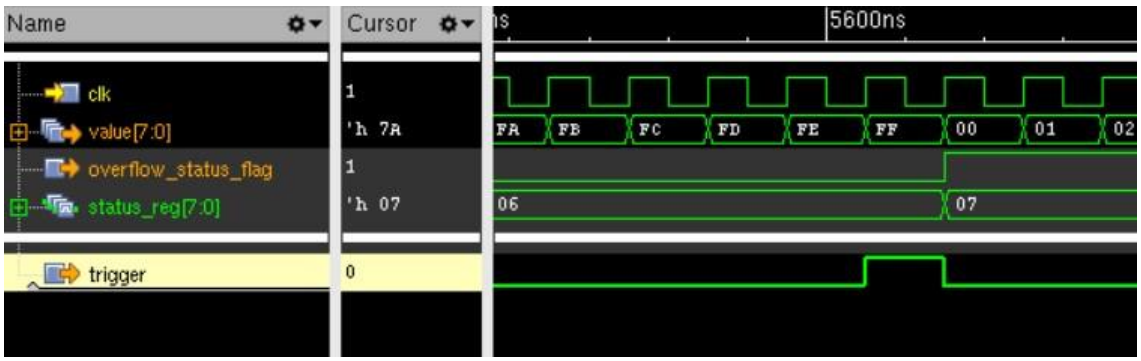


Figure 21 Trigger with overflow

10.1.17 Trigger generation match counter

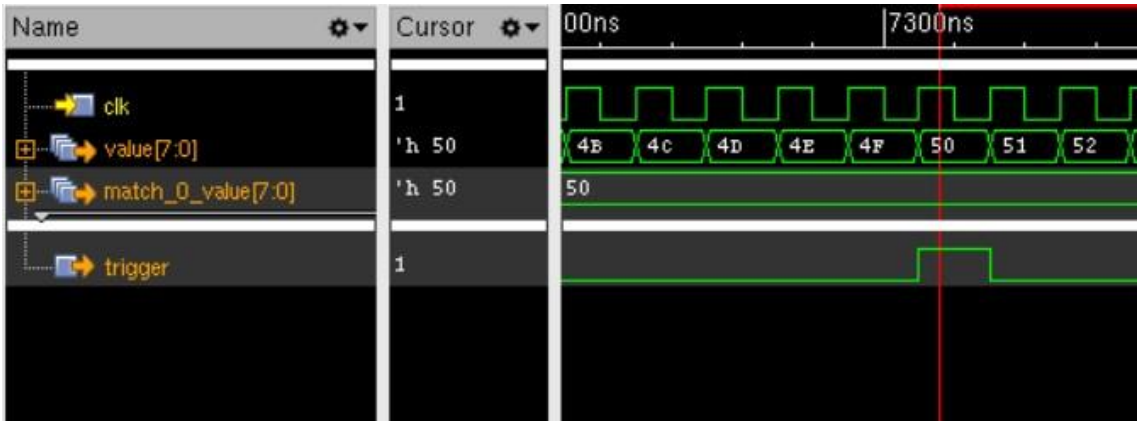


Figure 22 Trigger generation with match counter