ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Diseño y pruebas experimentales de una arquitectura de microcontrolador embebido para el desarrollo de prácticas para el Laboratorio de Sistemas Digitales Avanzados

PROYECTO DE TITULACIÓN

Previo la obtención del Título de:

Ingeniero en Electricidad especialización Electrónica y Automatización Industrial

Presentado por:

Arnold Steven Pinto Quintanilla

Carlos Luis Espinoza Molina

GUAYAQUIL – ECUADOR

Año: 2021

DEDICATORIA

A Dios por la sabiduría y fuerzas que me brinda para seguir adelante, a mis padres, y hermanos, en especial a mi madre Janeth que con su esfuerzo, ejemplo y sabias palabras no me permitieron rendirme aun cuando la situación se complicaba. Además, para aquellas particulares personas que conocí al inicio de mi etapa universitaria y que actualmente no se encuentran conmigo.

Los amo.

Arnold Steven Pinto Quintanilla

Dedico este trabajo a mi familia por el apoyo incondicional que siempre me brindaron, principalmente a mi mamá y hermano por ser fundamentales a lo largo del desarrollo de mi carrera universitaria.

Carlos Luis Espinoza Molina

AGRADECIMIENTOS

Nuestros más sinceros agradecimientos a todos los docentes que nos brindaron sus conocimientos en cada una de las materias que conformaron la carrera. Especialmente, queremos agradecer a nuestra tutora, MSc. Sara Ríos Orellana, por su orientación y consejos para llevar a cabo la realización de este proyecto.

Arnold Steven Pinto Quintanilla

Carlos Luis Espinoza Molina

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; **ARNOLD STEVEN PINTO QUINTANILLA, CARLOS LUIS ESPINOZA MOLINA** y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

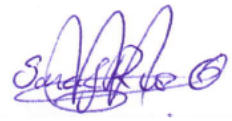
ARNOLD STEVEN
PINTO QUINTANILLA

CARLOS LUIS
ESPINOZA MOLINA

EVALUADORES

PhD. WILTON AGILA GÁLVEZ

PROFESOR DE LA MATERIA



MSc. SARA RÍOS ORELLANA

PROFESOR TUTOR

RESUMEN

De acuerdo a la comparación que existe entre la adquisición de un Controlador Lógico Programable (PLC) y un microcontrolador, para el desarrollo de pequeñas tareas lógicas, al igual que el incentivo del estudio y diseño de microcontroladores hacia los estudiantes, se ha propuesto el desarrollo de una arquitectura de microcontrolador de 8 bits tipo HARVARD embebido en una tarjeta FPGA DE10-Nano mediante VHDL a través del software Quartus Prime Lite Edition.

Con este diseño implementado se comprobó el correcto funcionamiento de 24 códigos de instrucciones implementadas en el diseño del microcontrolador dentro del Simulation Waveform Editor, por medio de la visualización de señales internas de la arquitectura. Igualmente se generó una señal PWM con frecuencia de 200 Hz y un duty cycle del 32%. Por último, mediante 3 displays de 7 segmentos se visualizó uno de los resultados de un set de instrucciones de la memoria ROM principal del diseño.

Los componentes utilizados para el desarrollo de este proyecto comprenden una tarjeta Terasic FPGA DE10-Nano, que simula la arquitectura diseñada, el software Quartus II Prime Lite Edition para la codificación y carga del archivo de la arquitectura diseñada, un ordenador de 8GB de RAM, un osciloscopio digital de 200 MHz de ancho de banda, un multímetro digital, y un protoboard con un set de resistencias de 220 Ω, jumpers, displays de 7 segmentos y un DIP switch. Todo esto último para la medición de los resultados durante las pruebas experimentales de la arquitectura diseñada.

El diseño de la arquitectura final del microcontrolador está conformado por 33 bloques internos, con topología Harvard, donde la memoria del programa y de datos se encuentran separadas y gobernadas por una unidad de control, que opera 24 tipo de instrucciones diferentes. Donde cada ciclo de instrucción se ejecuta por una señal de reloj negativa. Se hace uso de 168 bloques de memoria de bits de 5,662.720 bloques que dispone la tarjeta DE10-Nano, lo que equivale a menos del 1%, permitiendo implementar a futuro un mayor número de instrucciones, operaciones de nivel complejo y periféricos como puertos I2C o USB para una mejora del diseño propuesto.

Palabras Clave: FPGA DE10-Nano, direccionamiento, Harvard, bits, VHDL.

ABSTRACT

According to the comparison that exists between the acquisition of a Programmable Logic Controller (PLC) and a microcontroller, for the development of small logic tasks, as well as the incentive of the study and design of microcontrollers to students, we have proposed the development of an 8-bit microcontroller architecture HARVARD type embedded in a DE10-Nano FPGA card using VHDL through the Quartus Prime Lite Edition software.

With this design implemented, the correct operation of 24 instruction codes implemented in the microcontroller design within the Simulation Waveform Editor was verified by means of the visualization of internal signals of the architecture. A PWM signal with a frequency of 200 Hz and a duty cycle of 32% was also generated. Finally, by means of three 7-segment displays, one of the results of a set of instructions of the main ROM memory of the design was visualized.

The components used for the development of this project comprise a Terasic FPGA DE10-Nano board, which simulates the designed architecture, the Quartus II Prime Lite Edition software for coding and loading the designed architecture file, an 8GB RAM computer, a 200 MHz bandwidth digital oscilloscope, a digital multimeter, and a breadboard with a set of 220 Ω resistors, jumpers, 7-segment displays and a DIP switch. All the latter for measuring the results during the experimental tests of the designed architecture.

The final architecture design of the microcontroller consists of 33 internal blocks, with Harvard topology, where the program and data memory are separated and governed by a control unit, which operates 24 different types of instructions. Each instruction cycle is executed by a negative clock signal. Use is made of 168-bit memory blocks out of 5,662,720 blocks available on the DE10-Nano board, which is equivalent to less than 1%, allowing future implementation of a larger number of instructions, complex level operations and peripherals such as I2C or USB ports for an improvement of the proposed design.

Keywords: DE10-Nano FPGA, addressing, Harvard, bits, VHDL.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. INTRODUCCIÓN

Hoy en día la automatización posee un papel fundamental para el desarrollo de actividades que nos involucran de manera colectiva, siendo sus principales objetivos (Siemens, 2021):

- Mayor eficiencia de trabajo
- Mejores tiempos de respuesta
- Previsibilidad de resultados
- Reducción de errores

Las personas encargadas de llevar a cabo la automatización de procesos generalmente entran en contacto con los equipos involucrados durante la realización de estudios académicos superiores, resultando un requisito indispensable el conocer los instrumentos con los que se trabajará, principalmente en el ámbito ingenieril.

Considerando el limitado tiempo de aprendizaje enfocado a esta área y que “los conocimientos acerca de metodologías, técnicas y avances tecnológicos relacionados a la medición y control de procesos pareciesen no tener fin” (Festo, 2021), resulta en una ardua tarea la enseñanza y aprendizaje de temáticas relacionadas a la automatización de procesos.

Para la impartición de conocimientos sólidos relacionados con el campo de la automatización, las instituciones de educación superior se ven en la misión de adquirir equipos que se adapten a las tecnologías emergentes y demandadas en el mercado laboral, y que al mismo tiempo puedan cubrir la mayor cantidad de conocimientos con la elaboración de prácticas experimentales, teniendo en cuenta el presupuesto asignado para la obtención de dichos conocimientos.

Debido al amplio rango de opciones que existen en el mercado para realizar la automatización de una tarea o proceso, ya sea por la confiabilidad que ofrece un fabricante u otro, o la forma en cómo operan determinados equipos, es

indispensable realizar su adecuada selección. Al escoger las partes que formarán la solución, la variable costo de adquisición resulta determinante, puesto que por mayores funciones adicionales que posea un artefacto, éste puede estar sobredimensionado para la escala que aborda el proyecto.

1.1. Descripción del Problema

En la flexibilidad de las industrias existen procesos que requieren de la automatización de ciertos eventos, por lo que es importante elegir el hardware encargado del proceso de automatización. Éste puede ser dirigido por un Controlador Lógico Programable (PLC) industrial o una tarjeta embebida que posea un microcontrolador, entre otros. La selección de dicho hardware se realizará acorde a los requerimientos del proceso a automatizar y del presupuesto que se disponga en el proyecto, teniendo en cuenta que el precio de un PLC en el mercado es muy elevado en comparación de una tarjeta embebida.

Los métodos para la automatización de procesos se encuentran focalizados en el manejo de la electrónica digital, siendo el área de conocimiento que permite controlar las funciones de equipos de mayor tamaño mediante arreglos de pequeños circuitos lógicos. En la carrera de Electrónica y Automatización Industrial de la ESPOL se aborda la enseñanza de este campo en los últimos cursos, donde se aprende inicialmente el manejo de compuertas lógicas y programación de sistemas digitales, finalizando con aplicaciones complejas que utilizan sistemas embebidos y comunicaciones entre dispositivos.

El estudio teórico de sistemas embebidos se realiza en al menos dos materias dentro de la malla de la carrera. Una de dichas materias es Sistemas Digitales II, que posee el Laboratorio de Sistemas Digitales Avanzados para el desarrollo de su componente experimental. Dentro del material que se estudia en esta materia existe mucho interés en el desarrollo de microcontroladores embebidos, dado que en un capítulo se estudian las diferentes arquitecturas que se pueden construir experimentalmente.

Bajo este contexto, se propone diseñar una arquitectura modificada basada en FPGA, la misma que servirá para la impartición de prácticas de laboratorio de Sistemas Digitales II, materia fundamental para potenciar los conocimientos

ingenieriles y de uso en la industria local de los futuros ingenieros de la carrera de Electrónica y Automatización Industrial de la Facultad de Ingeniería en Electricidad y Computación (FIEC).

1.2. Justificación del Problema

Luego de realizar ligeramente una comparación de la tecnología, primordialmente de costos, se determina que un PLC Industrial es mucho más costoso que una tarjeta de desarrollo. Adicionalmente, las FPGA cumplen dos condiciones fundamentales, resultan ser accesibles por su bajo costo, teniendo en cuenta el presupuesto del laboratorio, y no requieren de espacio considerable para establecer una estación de trabajo.

Además, la tarjeta a programar nos permitirá desarrollar arquitecturas embebidas reconfigurables, que podremos modificar conforme la dificultad del diseño va en aumento y la necesidad de la aplicación lo requiera.

Por otro lado, la herramienta utilizada (Quartus II) resulta didáctica para el Laboratorio de Sistemas Digitales Avanzados, donde el estudiante, utilizando el microcontrolador desarrollado en la presente tesis, probará las arquitecturas vistas en clase de una manera más versátil utilizando tarjetas de desarrollo basadas en Arreglos de Compuertas Programables por Campos (FPGA) y programadas en Lenguaje de Descripción de Hardware de Alta Velocidad (VHDL).

La estructura del microcontrolador propuesto permitirá potenciar los conocimientos de los estudiantes para proponer soluciones que requieren el uso de tecnologías emergentes a casos reales que presenta el país. Lográndolo a través de proyectos que prueben los saberes obtenidos en cursos anteriores y los impartidos por el Laboratorio de Sistemas Digitales Avanzados.

1.3. Objetivos

1.3.1. Objetivo General

Diseñar la arquitectura de un microcontrolador embebido en una tarjeta FPGA DE10-Nano mediante VHDL y herramienta CAD para uso didáctico dentro del Laboratorio de Sistemas Digitales Avanzados.

1.3.2. Objetivos Específicos

- Seleccionar las funciones fundamentales que cumplirá el microcontrolador en base a las especificaciones del Laboratorio de Sistemas Digitales Avanzados.
- Programar los bloques necesarios para el microcontrolador mediante lenguaje VHDL.
- Diseñar la arquitectura y el plano circuital del microcontrolador mediante el software Quartus II.
- Comprobar el funcionamiento del microcontrolador embebido por medio de la respectiva programación de la tarjeta FPGA DE10-Nano y pruebas experimentales.
- Generar una guía de usuario que permita la realización de pruebas con la arquitectura desarrollada.

1.4. Marco Teórico

1.4.1. Antecedentes

Diversos trabajos académicos detallan las ventajas de incluir las FPGA como parte del proceso de aprendizaje de habilidades en el campo de los sistemas embebidos durante cursos prácticos. Además, existen otros artículos en los que se fundamenta la importancia de implementar arquitecturas propias para obtener funcionalidades determinadas gracias a la característica reprogramable de las tarjetas de desarrollo.

En el ámbito educativo pueden encontrarse diversas ventajas y alcances al incluir las FPGA como parte del curso de laboratorio. Según explican (Balid & Abdulwahed, 2013) las FPGA poseen un amplio rango de aplicaciones para aprendizaje en el aula, como puede ser: robots basados en FPGA,

diseño de sistemas programados en chips (SoCP), diseño de hardware/software, arquitecturas de computadores y procesamiento de señales por hardware. Las facilidades que ofrecen las FPGA permite a los estudiantes trabajar en proyectos de alcance significativo con la ayuda de la gran cantidad de puertas lógicas con las que cuenta sin dejar de lado los fundamentos del diseño digital.

De igual manera, las FPGA pueden ser implementadas para la enseñanza de conocimientos que no solo involucra el ámbito de la automatización de procesos, tal como mencionan (Charte et al., 2017) el constante incremento de facilidades para la obtención de circuitos FPGA, junto con la amplia disponibilidad de software gratuitos y de libre acceso para su manipulación, permiten considerar la innovación de posibilidades en el campo de los procesadores, especialmente en la enseñanza de arquitecturas.

Por otro lado, las arquitecturas que pueden diseñarse de manera independiente son utilizadas para la resolución de problemas que requieran de características específicas para abarcar una solución, como un trabajo realizado en la Universidad de Waterloo, en el que la arquitectura propuesta utiliza una FPGA comercial para la linealización de transmisores 5G de banda ancha (Huang et al., 2019).

Así mismo, pueden plantearse arquitecturas capaces de cubrir necesidades que surgen en cualquier ámbito donde se requiera mejoras continuas, como en el caso de un trabajo realizado en la Universidad de Lanzhou donde se busca mejorar la capacidades de procesadores RISC (del inglés *Reduced Instruction Set Computer*) de 16 bits con un modelo que utiliza acumuladores múltiples (Jing Yu et al., 2013), cuyos resultados finales se probaron en una FPGA, obteniendo el éxito esperados.

Cabe resaltar, la aplicación de arquitecturas embebidas en FPGA para fines de investigación pertenecientes a instituciones de educación superior, tomando como referencia un trabajo de la Universidad Distrital Francisco José De Caldas, en el que se plantea la estructura de un microprocesador de 8 bits con doble acumulador diseñado para ser usado en FPGA (Sáenz

Rodríguez et al., 2018), en este se incluyen 28 instrucciones específicas que el microprocesador es capaz de realizar para propósitos determinados.

1.4.2. Conceptos Relacionados

- ***Microcontrolador***

  Un microcontrolador es un circuito integrado compacto diseñado para gobernar una operación específica en un sistema embebido. Un microcontrolador incluye típicamente un procesador, memoria y periféricos de entrada/salida en un solo chip. Principalmente se trata de computadoras miniatura diseñadas para controlar pequeñas funciones de componentes más grandes, sin la necesidad de un sistema operativo con interfaces de usuario complejas. (Lutkevich, 2019)

- ***Lenguaje de Descripción de Hardware de Alta Velocidad (VHDL)***

  VHDL (del inglés *Very High-Speed Integrated Circuit Hardware Description Language*) es un lenguaje de programación utilizado para la descripción de hardware en la automatización de diseño electrónico para expresar sistemas digitales y de señal mixta, como circuitos integrados y FPGA. (Cadence, 2021)

- ***Arreglos de Compuertas Programables por Campos (FPGA)***

  Una FPGA (del inglés *Field Programmable Gate Arrays*) es un dispositivo semiconductor basado en arreglos de bloques lógicos configurables que utilizan conexiones programables. Estos pueden ser reprogramados para la aplicación deseada o para cumplir requisitos de funcionalidad. (Xilinx, 2021)

- ***Quartus II***

  Es un software utilizado para la realización de diseños en FPGA, SoC (del inglés *System on Chip*) y CPLD (dispositivos lógicos programables complejos), abarcando desde el diseño de entradas y la síntesis, hasta la optimización, verificación y simulación (Intel, 2021). Es seleccionado este

software para realizar la programación de los componentes del microcontrolador propuesto. Además, cuenta con la herramienta que permite embeber la arquitectura en la FPGA, teniendo la oportunidad de poner a prueba cada una de las instrucciones desarrolladas y corregirlas en caso de errores.

- ***Unidad aritmética lógica (ALU)***

  Una ALU (del inglés *Arithmetic Logic Unit*) representa el componente principal de la unidad de procesamiento de información que poseen los sistemas informáticos. En ésta se realizan diversas operaciones referentes a lógica y aritmética, dependientes de la cantidad de bits configurados, con operadores que provienen del decodificador de instrucciones. (Brown & Vranesic, 2006)

- ***Máquina secuencial sincrónica (MSS)***

  Una MSS puede definirse como un programa computacional que utiliza señales de reloj para controlar su operación. En esta se realiza la lectura de señales de entrada, para generar un cambio de estado y/o producir una salida correspondiente, dependiendo del estado interno en el que se encuentre en ese instante. (Brown & Vranesic, 2006)

- ***Memoria de acceso aleatorio (RAM)***

  La memoria RAM (del inglés *Random Access Memory*) se refiere al elemento principal de almacenamiento de un dispositivo que requiere de datos y programas informáticos, permitiendo el acceso a cualquier ubicación de sus registros. Esta es de tipo volátil, dado que requiere de un suministro de energía para mantener almacenada la información, por lo que solo es utilizada de manera temporal. (Micron, 2020)

- ***Memoria de solo lectura (ROM)***

  La memoria ROM (del inglés *Read Only Memory*) solo permite realizar operaciones de lectura, lo que indica que, para un sistema informático, la información en ésta se encuentra previamente configurada para su uso. Es

del tipo no volátil, por lo que no requiere de un suministro de energía para mantener almacenada la información, permitiendo que los datos se guarden de forma permanente. (Micron, 2021)

- ***Decodificador de instrucciones***

  El decodificador de instrucciones es el encargado de leer las instrucciones provenientes de la memoria, obtener sus componentes a manera de información y enviar a los lugares donde se requiera. En los decodificadores de instrucciones solo es posible tener una salida a la vez y esta depende de las entradas. (Brown & Vranesic, 2006)

- ***Registro de instrucciones***

  Los registros son utilizados para el almacenamiento de información, y en el caso del registro de instrucciones éste guarda el código de la instrucción que se encuentra ejecutando. La información le llega al registro desde la memoria ROM, que almacena el programa principal, que contiene las instrucciones. (Brown & Vranesic, 2006)

CAPÍTULO 2

2. METODOLOGÍA UTILIZADA Y DISEÑO DEL MICROCONTROLADOR

En este capítulo se describe el funcionamiento de la arquitectura del microcontrolador que cumple con las especificaciones que requiere el Laboratorio de Sistemas Digitales Avanzados. La arquitectura a desarrollar se encuentra fundamentada en el material que se revisa en la asignatura Sistemas Digitales II, precisamente en la sección de sistemas embebidos, y en el trabajo antes mencionado de la Universidad Distrital Francisco José De Caldas.

Se requiere de la unión de conceptos revisados en materias previas y de artículos de la literatura recopilada:

- Del material de estudio, fue seleccionada una arquitectura práctica con la que se enseña el manejo de instrucciones de una ALU, que posee: controlador (MSS), registro de instrucciones, memorias RAM y ROM, decodificador de instrucciones, selectores, acumuladores, contadores y registros de propósitos generales.
- Del trabajo de la Universidad Distrital Francisco José De Caldas, fueron tomadas como referencia diversas instrucciones que utilizaron para su desarrollo. Específicamente 19 de las 28 instrucciones que posee dicho trabajo, con las cuales se cumplirán objetivos específicos.

El presente desarrollo no busca emular arquitecturas de microcontroladores comerciales, dado la masiva cantidad de instrucciones que estos manejan. Por otro lado, no se trata de algún chip para ser comercializado. Con las 19 instrucciones se plantea el enfoque hacia la enseñanza de conocimientos básicos en el área de sistemas digitales.

2.1. Tipos de Arquitecturas

Las arquitecturas utilizadas para el desarrollo de sistemas computacionales se encuentran fundamentadas principalmente en dos tipos: Von-Neumann y Harvard.

2.1.1. Arquitectura Von-Neumann

Esta configuración posee una única memoria y utiliza un solo flujo de información para el almacenamiento y ejecución de datos e instrucciones respectivamente, por lo que en cada ciclo de reloj solo puede ejecutarse una instrucción y un dato (Baba, 2019).

El disponer de un único medio para el almacenamiento y manejo de información resulta en un inconveniente para el procesador, ya que no puede acceder a la memoria de instrucciones y a la memoria de datos al mismo tiempo, resultando en el “cuello de botella Von-Neumann” que disminuye el rendimiento del sistema. La figura 2.1 muestra esta estructura.

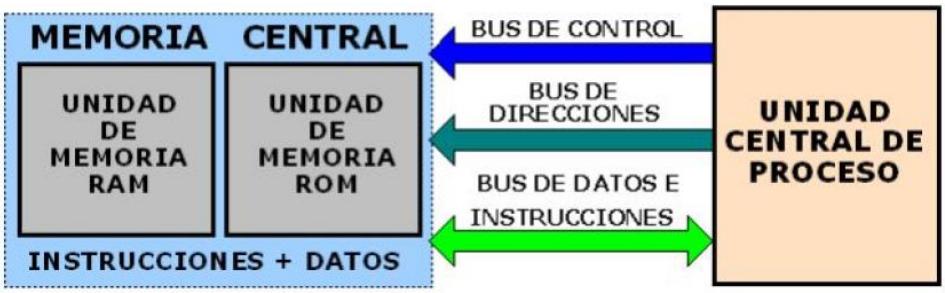


Figura 2.1: Diagrama de la arquitectura Von-Neumann (Ríos, 2020, p.5)

2.1.2. Arquitectura Harvard

La configuración Harvard emplea diferentes memorias y buses de información para el almacenamiento y manejo de datos e instrucciones. Con esta distribución interna de recursos se evita el problema del cuello de botella que generaba el modelo Von-Neumann (Baba, 2019).

Al encontrarse separadas las secciones de almacenamiento para las instrucciones y los datos, es posible obtener al mismo tiempo la información que requiere el procesador cuando se ejecuta una instrucción, incrementando el rendimiento del sistema al permitir guardar el resultado de una operación y realizar la búsqueda y ejecución de una nueva instrucción por medio de rutas de información diferentes. La figura 2.2 muestra la estructura de la arquitectura Harvard.

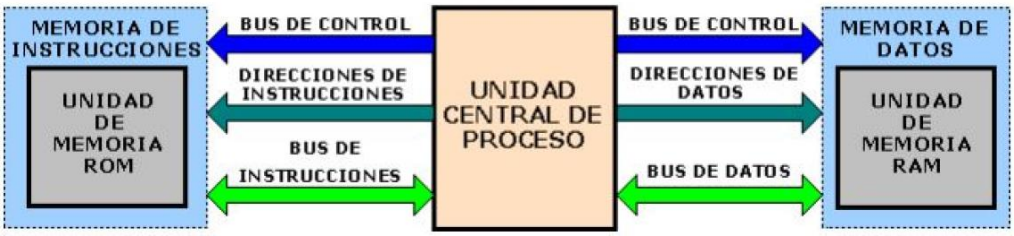


Figura 2.2: Diagrama de la arquitectura Harvard (Ríos, 2020, p.5)

2.2. Modos de Direccionamiento

Los modos de direccionamiento hacen referencia a la forma en que puede especificarse la ubicación de los datos y los métodos para acceder a ellos (Mendías, 2018, p. 16). Existen múltiples métodos para el direccionamiento, de manera general son: inmediato, directo e indirecto.

2.2.1. Direccionamiento inmediato

En el direccionamiento inmediato el dato se ubica en la dirección siguiente, a continuación del código de la operación, en la memoria del programa.

Comprendiéndose gráficamente como se muestra en la figura 2.3.

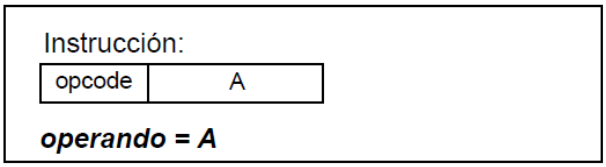


Figura 2.3: Direccionamiento inmediato gráficamente (Mendías, 2018, p. 17)

La ventaja de este direccionamiento es la rapidez, debido a que no es necesaria información adicional para la extracción del dato. Resulta como inconveniente que esta forma no es muy flexible a cambios al momento de realizar la localización de datos (E. U. P Valladolid, 2016, p. 1).

2.2.2. Direccionamiento directo

El direccionamiento directo puede realizarse de dos tipos: directo a registro o directo a memoria.

- **Directo a registro:** Debe especificarse la dirección del registro donde radica el dato o bien donde es necesario almacenar un

resultado (E. U. P Valladolid, 2016, p. 2). Su representación gráfica es como se muestra en la figura 2.4.

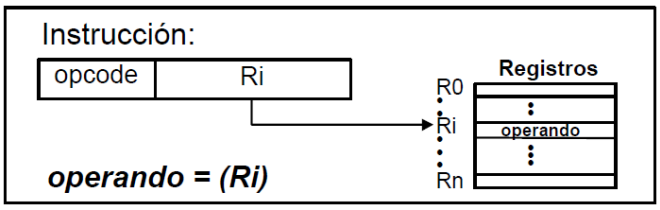


Figura 2.4: Direccionamiento directo por registro gráficamente
(Mendías, 2018, p. 17)

- **Directo a memoria:** También conocido como direccionamiento absoluto. En éste se especifica la dirección de memoria donde se encuentra el dato o donde debe almacenarse un resultado (E. U. P Valladolid, 2016, p. 3). La figura 2.5 muestra su representación gráfica.

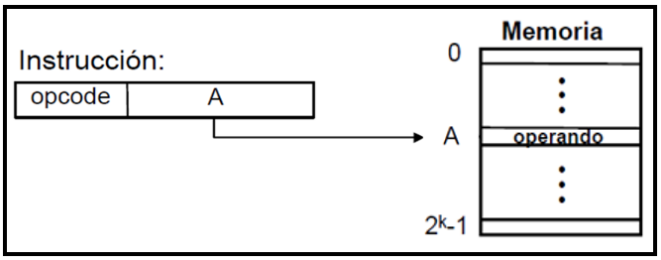


Figura 2.5: Direccionamiento directo por memoria gráficamente
(Mendías, 2018, p. 17)

2.2.3. Direccionamiento Indirecto

De igual manera al directo, el direccionamiento indirecto cuenta con diversos tipos, los cuales son: indirecto a registro e indirecto a memoria.

- **Indirecto a registro:** Debe de especificarse la dirección del registro en el que se encuentra almacenada la dirección de memoria donde está el dato o donde hay que dejar un resultado (E. U. P Valladolid, 2016, p. 3). Su representación gráfica es como la mostrada en la figura 2.6.

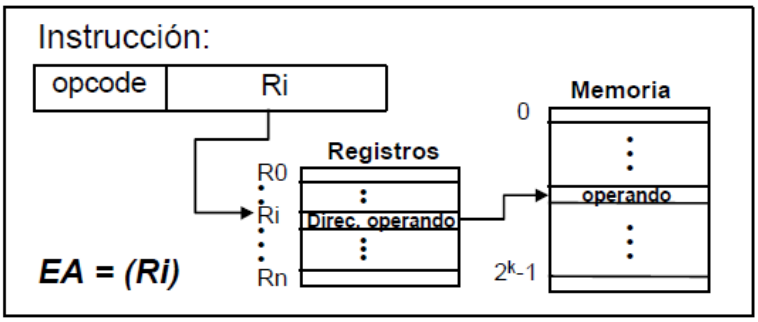


Figura 2.6: Direccionamiento indirecto por registro gráficamente
(Mendías, 2018, p. 18)

- **Indirecto a memoria:** En este se especifica una dirección de memoria, que al mismo tiempo contiene otra dirección de memoria, en donde se encuentra el dato o donde debe ser almacenado un resultado (E. U. P Valladolid, 2016, p. 4). Su representación gráfica es mostrada en la figura 2.7.

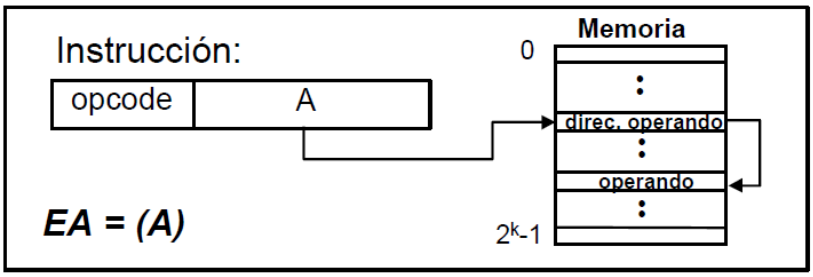


Figura 2.7: Direccionamiento indirecto por memoria gráficamente
(Mendías, 2018, p. 18)

2.3. Arquitectura Base

Para el diseño del microcontrolador, se tiene como conocimiento el funcionamiento de la arquitectura mostrada en la figura 2.8. En este caso la configuración es utilizada para realizar dos operaciones con la ALU: suma de dos operandos y permitir el paso de un dato de entrada. Implementa el controlador (MSS), decodificador de instrucciones, memorias RAM y ROM, ALU, registros de almacenamiento, contador y selector.

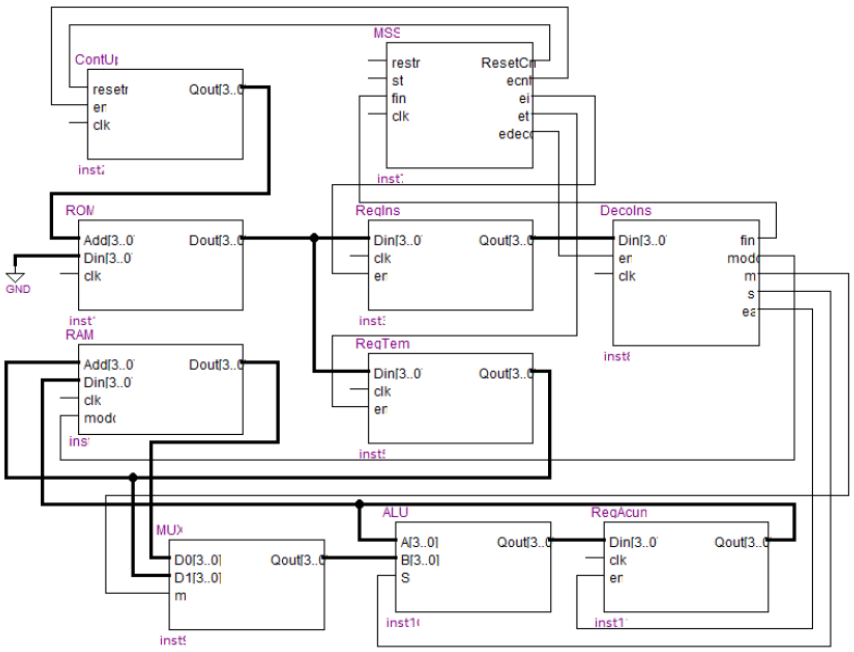


Figura 2.8: Diagrama de la arquitectura base del microcontrolador (Ríos, 2020)

Cada bloque cuenta con una función específica para el funcionamiento general que se requiere.

- La MSS se encarga de enviar las señales de salida que se requieren en distintas secciones del circuito, principalmente habilitadores y señales de reinicio, dependiendo de las señales de entrada y el flanco de reloj.
- El contador ascendente permite llevar un registro de conteo que se realiza cada vez que se detecte su señal habilitadora de entrada, generada por la MSS, que sirve para el manejo de las memorias ROM y RAM.
- La memoria ROM posee los códigos (formato hexadecimal) con los que se realizarán las instrucciones. Para este diseño la ROM contiene el código del programa almacenado en función de la codificación de las instrucciones y su operación se dará por medio del contador ascendente, que controla las direcciones.
- El registro de instrucciones permite almacenar la instrucción que se está ejecutando en el circuito durante un determinado intervalo de tiempo.

- El decodificador de instrucciones funciona de tal manera que, al recibir los códigos de la memoria ROM, sus salidas generan señales que permiten la operación específica de los demás componentes.
- La ALU se encarga de ejecutar las instrucciones que en ésta se encuentran configuradas para generar salidas en función de sus entradas, donde una de estas proviene del decodificador para la selección de la instrucción.
- El registro acumulador permite almacenar temporalmente valores para su uso en las operaciones de la ALU (realimentación) y funciona como dato de entrada para la memoria RAM.
- En la memoria RAM se realiza el almacenamiento de los datos provenientes del registro acumulador, utilizando direcciones de memoria que coinciden con los valores que registra el contador ascendente.
- El selector permite la comunicación de señales provenientes de diversas secciones del circuito, dependiendo de la cantidad de bloques que necesite enrutar, y utiliza una de éstas según corresponda, por medio de una señal generada por el decodificador.

2.3.1. Funcionamiento de la MSS

La MSS es la encargada de controlar las funciones de los demás elementos del circuito. Por lo que su estructura interna debe cumplir con requisitos específicos, dada una señal entrada debe ser capaz de producir una salida en un tiempo oportuno.

La estructura de una MSS puede ser comprendida por medio de un diagrama ASM (del inglés *Algorithmic State Machine*), con el cual es posible describir, de manera sencilla y ordenada, las operaciones de un sistema digital.

Las entradas son representadas por rombos, si se encuentran activas o no se toma una decisión u otra, mientras que las salidas son generadas dependiendo de los estados, los cuales se representan con rectángulos. El manejo del cambio de estado se da por medio de señales de reloj.

Para el esquema base, con el cual iniciamos el diseño de este microcontrolador, la MSS se maneja por el diagrama mostrado en la figura 2.9.

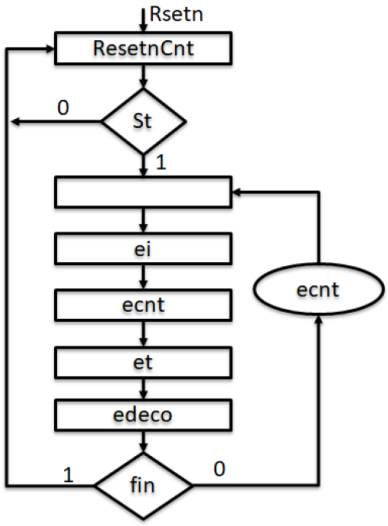


Figura 2.9: Diagrama ASM de la MSS del esquema base (Ríos, 2020)

El proceso de operación de la MSS se encuentra dividido en tres etapas: búsqueda, lectura, decodificación y ejecución de las instrucciones almacenada en la memoria ROM.

Empezando con la lectura del dato de entrada que permite el funcionamiento del sistema, en esta sección se añaden señales o estados específicos que resultan necesarios para una adecuada operación:

- El estado inicial de la MSS genera la señal que reinicia al contador (ResetnCnt), de esta forma es seguro que el sistema empiece desde cero el conteo para acceder a los códigos de las instrucciones.
- La primera señal de entrada que se requiere, St o Start, es utilizada para conocer si el sistema se encuentra en marcha. Si así fuere se activarán los estados siguientes para generar salidas, caso contrario se mantiene al sistema en espera repitiendo esta acción.
- El segundo estado permite implementar un retardo en la generación de señales. Este recurso es principalmente utilizado para que las

señales de las demás secciones del circuito puedan sincronizarse y llegar a los lugares correspondientes en el tiempo adecuado.

El proceso de búsqueda de los códigos de cada instrucción es realizado en los siguientes tres estados, en conjunto a operaciones adicionales que requiere el sistema:

- El tercer estado genera la señal que habilita el registro de instrucciones (ei), permitiendo el almacenamiento de la instrucción actual que se está decodificando.
- El cuarto estado permite habilitar al contador (ecnt) para que incremente su valor interno y manejar el siguiente código de la memoria ROM.
- El quinto estado habilita un registro temporal (señal et) en el que se almacena información para ser utilizada por la ALU.

A partir del siguiente estado se realiza la ejecución de la instrucción. Adicionalmente presenta la validación que permite al sistema conocer si ha culminado su operación:

- El sexto estado genera la señal que habilita el funcionamiento del decodificador de instrucciones (edeco), las señales de este último resultan indispensables para la ejecución. En este estado los bloques que necesitan la información del decodificador son capaces de realizar su función y obtener un resultado de la instrucción ejecutada.
- La señal de entrada Fin es utilizada para identificar si el último código de la memoria ROM ha sido ejecutado, siendo esta señal generada por el decodificador de instrucciones. Si se encuentra activa, el sistema termina su operación y regresa al estado inicial; caso contrario, se realiza nuevamente el proceso de búsqueda y ejecución de instrucciones desde el segundo al sexto estado.

2.4. Instrucciones del microcontrolador

Las instrucciones implementadas en la arquitectura propuesta se encuentran divididas en dos secciones: las instrucciones manejadas por la ALU y las que no requieren de esta.

Tabla 2.1: Instrucciones que utilizan la ALU

Descripción	Mnemotécnico	Código	Afecta	
			Cero	Acarreo
Suma	ADD	100000	1	1
Suma con acarreo	ADDC	100010	1	1
Resta	SUB	100100	1	1
Resta con acarreo	SUBC	100110	1	1
Incrementar	INC	101000	1	1
Decrementar	DEC	101010	1	1
Desplazamiento a la izquierda (sin acarreo)	SHL	101100	1	0
Desplazamiento a la derecha (sin acarreo)	SHR	101110	1	0
Desplazamiento circular a la izquierda	ROL	110000	1	1
Desplazamiento circular a la derecha	ROR	110010	1	1
Operadores Lógicos	AND	110100	1	0
	OR	110110	1	0
	XOR	111000	1	0
	NOT	111010	1	0
Cargar valor en el acumulador	LOAD	111100	1	0

Mover valores de acumuladores	MOVE	010100	1	0
Multiplicación	MULT	100001	1	1
División	DIV	100011	1	1
Porcentaje PWM	PDC	100101	1	1

Cada una de las instrucciones de la tabla 2.1 contiene su nombre, mnemotécnico para identificarla en la programación con Quartus II, su respectivo código para ser almacenado en la memoria ROM y si esta utiliza las funciones del cero, del acarreo o ambas.

A continuación, se describe cada una de estas instrucciones:

- **Suma:** Permite la adición de dos números, utilizando el acarreo siempre que el resultado sea de mayor escala que la programada en el bloque.
- **Suma con acarreo:** Permite la adición de dos números, considerando la información del acarreo como entrada que formará parte de la operación.
- **Resta:** Permite la sustracción entre dos números. Utiliza el acarreo si el resultado es de mayor escala que la programada en el bloque.
- **Resta con acarreo:** Permite la sustracción entre dos números, considerando la información en el acarreo como parte de la operación.
- **Incrementar:** Aumenta en una unidad el valor de un número seleccionado. Utiliza el acarreo si el resultado es de mayor escala que la programada en el bloque.
- **Decrementar:** Disminuye en una unidad el valor de un número seleccionado. Utiliza el acarreo si el resultado es de mayor escala que la programada en el bloque.
- **Desplazamiento a la izquierda (sin acarreo):** Incorpora un dígito (0 o 1 a elección del usuario) en la posición del bit menos significativo en un arreglo de bits, resultando en la pérdida de información del bit desplazado (más significativo).

- **Desplazamiento a la derecha (sin acarreo):** Incorpora un dígito (0 o 1 a elección del usuario) en la posición del bit más significativo en un arreglo de bits, resultando en la pérdida de información del bit desplazado (menos significativo).
- **Desplazamiento circular a la izquierda:** Permite mover la información del bit más significativo a la posición del bit menos significativo en un arreglo de bits, utilizando la bandera de acarreo como almacenamiento temporal durante el tiempo de desplazamiento de los demás bits del arreglo.
- **Desplazamiento circular a la derecha:** Permite mover la información del bit menos significativo a la posición del bit más significativo en un arreglo de bits, utilizando la bandera de acarreo como almacenamiento temporal durante el tiempo de desplazamiento de los demás bits del arreglo.
- **Operadores lógicos:** Corresponde a las cuatro operaciones lógicas básicas AND, OR, XOR y NOT. Estas operaciones pueden manejar datos tipo bit y vector.
- **Cargar valor en el acumulador:** Permite la carga de datos en el acumulador del sistema, pudiendo utilizarse en operaciones de las demás instrucciones.
- **Mover valores de acumuladores:** Permite establecer en la salida de la ALU un valor que se encuentre presente en los acumuladores.
- **Multiplicación:** Permite la multiplicación entre dos números. Utiliza el acarreo si el resultado es de mayor escala que la programada en el bloque.
- **División:** Obtiene la relación entre dos números, considerando la información del acarreo como entrada que formará parte de la operación.
- **Porcentaje PWM:** Realiza las operaciones matemáticas necesarias en el dato ingresado para determinar el tanto por ciento de la señal PWM.

Tabla 2.2: Instrucciones que no requieren de la ALU

Descripción	Mnemotécnico	Código
Salto si la bandera de cero es verdadera	JIFZ	000000
Salto si la bandera de acarreo es verdadera	JIFC	000010
Salto incondicional	JUMP	000100
Llamada a subrutina	CALL	000110
Retorno de Subrutina	RETURN	001000
Almacena el valor del acumulador en la RAM	STORE	001010
Lectura en puerto	INPUT	001100
Escritura en puerto	OUTPUT	001110
Fin	FIN	010100
Clear	CLEAR	010110
Activa PWM	PWM	011000
Retardo	DELAY	011110

La tabla 2.2 presenta las instrucciones que no requieren ser ejecutadas en conjunto con la ALU, estas no afectan las banderas del cero y del acarreo, pero el estado de estas si puede utilizarse para ejecutar una instrucción, ejemplo de esto son los saltos. A continuación, se describe cada una de estas instrucciones:

- **Salto si la bandera de cero es verdadera:** Salto a otra dirección de memoria cuando el estado de la bandera de cero es verdadero.
- **Salto si la bandera de acarreo es verdadera:** Salto a otra dirección de memoria cuando el estado de la bandera del acarreo es verdadero.

- **Salto incondicional:** Salto a otra dirección de memoria que no depende de banderas.
- **Llamada a subrutina:** Permite la ejecución de una subrutina programada, que se encuentre almacenada en registros de memoria.
- **Almacena el valor del acumulador en la RAM:** Posibilita el poder almacenar información en la memoria RAM proveniente de un acumulador, pudiendo ser el resultado de alguna operación realizada por la ALU.
- **Retorno de subrutina:** Luego de ejecutada una subrutina, permite retornar a una dirección de memoria para continuar con la lectura de la siguiente instrucción.
- **Lectura en puerto:** Permite obtener información presente en un puerto de la FPGA, consiguiendo datos del exterior para el sistema.
- **Escritura en puerto:** Permite enviar información hacia un puerto de la FPGA, transfiriendo datos desde el sistema hacia el exterior.
- **Clear:** Reinicia la información presente en los buses de datos.
- **Fin:** Genera la señal que permite al controlador terminar la operación del programa ejecutado.
- **Activa PWM:** Permite crear en uno de los pines de la tarjeta FPGA una señal PWM con frecuencia y ciclo de trabajo variable.
- **Retardo:** Permite programar un tiempo de espera para la recepción de datos provenientes del puerto de entrada físico.

2.5. Tarjeta de desarrollo DE10-Nano

En la ejecución del proyecto resulta necesario realizar pruebas experimentales con el dispositivo físico en el que se embeberá el microcontrolador desarrollado.

Se ha escogido la tarjeta de desarrollo DE10-Nano, mostrada en la figura 2.10, de la familia Cyclone V SE 5CSEBA6U23I7, que cuenta con tres generadores de reloj a 50Mhz, los cuales servirán para sincronizar todos los bloques creados para el funcionamiento del microcontrolador.

Adicionalmente, cuenta con el USB Blaster II, que es un sistema integrado para programación de la tarjeta en modo JTAG dentro del software Quartus II.

Posee dos conectores de 20 pines GPIO de uso múltiple, los que servirán para configurarlos como puertos de entrada y salida para el diseño propuesto.

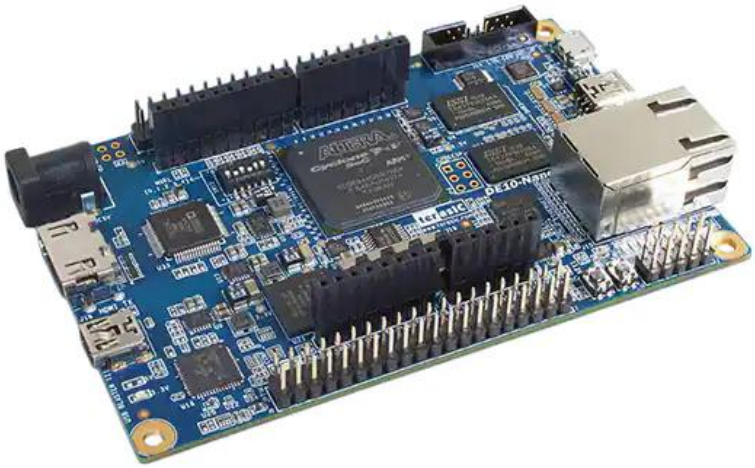


Figura 2.10: FPGA DE10-Nano

2.5.1. Comunicación de la FPGA DE10-Nano

La tarjeta FPGA DE10-NANO cuenta con un pin de entrada USB 2.0 mini B, el cual permite tener una comunicación bidireccional entre la tarjeta y la PC. Su ubicación en la tarjeta es señalada en la figura 2.11.

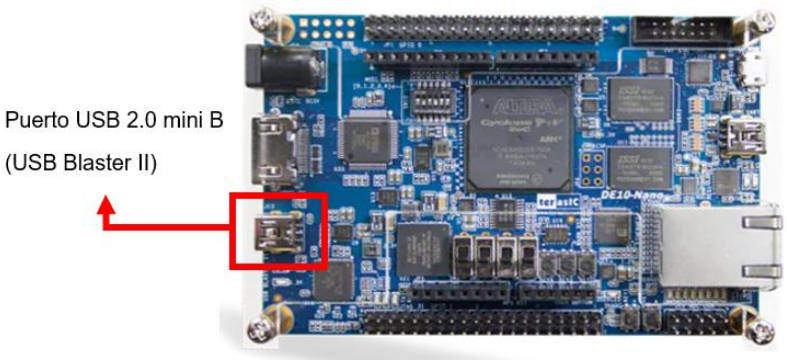


Figura 2.11: Puerto de comunicación de la FPGA DE10-Nano

Cabe resaltar, que al instalar el software Quartus II el firmware del USB Blaster II se encuentre instalado en la PC, debido a que este Driver posibilita el intercambio de datos por medio de la interfaz JTAG (Joint Test Access Group) facilitando la depuración de códigos, programación y pruebas de cualquier sistema embebido.

En la ruta donde se encuentra instalado Quartus II se verifica que la carpeta (usb-blaster-ii) se encuentre instalada. El seguimiento de la ruta de instalación es mostrado en la figura 2.12.

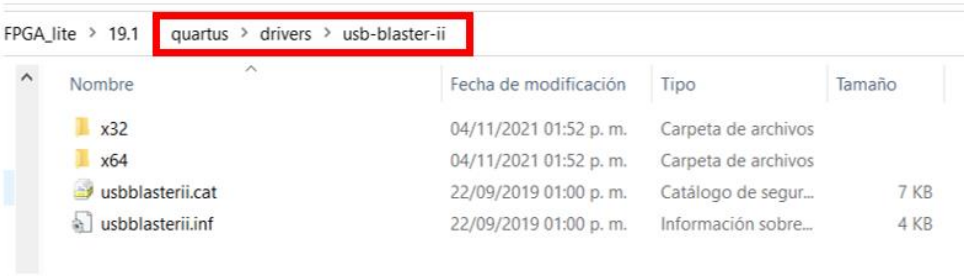


Figura 2.12: Verificación del driver para comunicación con FPGA DE10-Nano

2.6. Conector FPGA a Computador

El tipo de conector depende de las características de los componentes entre los que se tendrá la comunicación. La FPGA cuenta con el puerto USB 2.0 mini B, y los computadores convencionales con puertos USB tipo A, por lo que se requiere un cable USB tipo A a USB tipo mini B. La figura 2.13 muestra la apariencia física del conector al que se hace referencia.

Este conector posee terminales macho-macho tipo USB A y USB mini B, permitirá realizar la conexión entre el USB Blaster II que viene integrado en la tarjeta DE10-Nano y el ordenador. Los niveles de voltaje a los que operan los pines de este cable son de 5 Vdc y de 1.5 Vdc provenientes de la placa.

Figura 2.13: Cable USB tipo A a USB Mini B

2.6.1. Detección de la FPGA en el Computador

Teniendo en cuenta la instalación del firmware usb-blaster-ii en el computador, y la alimentación de la tarjeta DE10-Nano, es posible visualizar la conexión dentro del administrador de dispositivo del PC una vez conectado el cable de programación por medio del puerto JTAG.

A continuación, en la figura 2.14, se observa el administrador de dispositivos del computador donde no se encuentra configurado ningún dispositivo JTAG, debido a que la tarjeta no se encuentra conectada.

El acceso a esta ventana puede realizarse utilizando el buscador del ordenador, colocando “Administrador de dispositivos” y seleccionando la primera opción, perteneciente al panel de control.

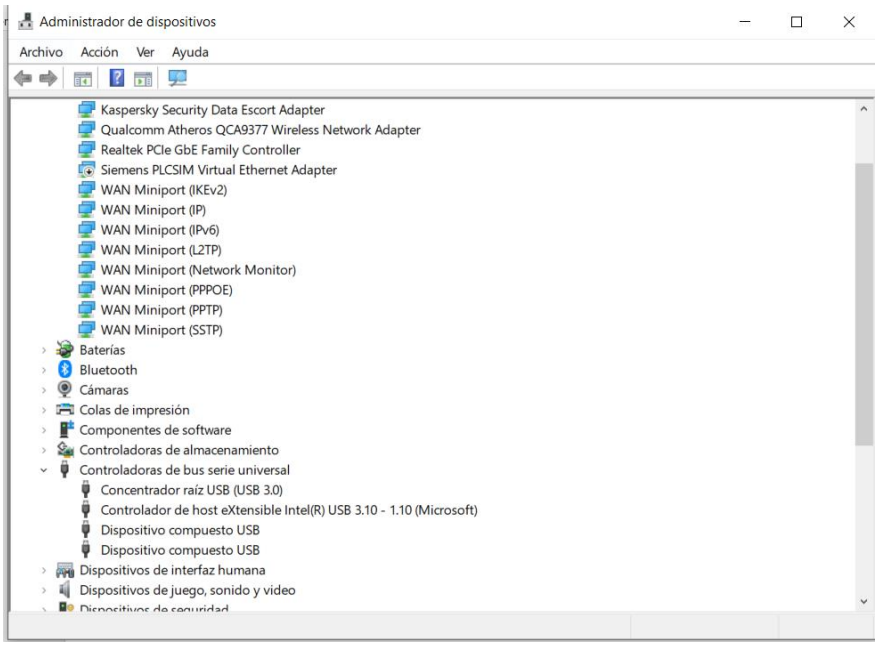


Figura 2.14: Dispositivo JTAG cables no detectado

Una vez conectada la tarjeta, se puede observar en la figura 2.15 la pestaña ‘JTAG cables’ que contiene el ‘Altera USB-Blaster II (JTAG interface)’ y el ‘Altera USB-Blaster II (System Console interface).

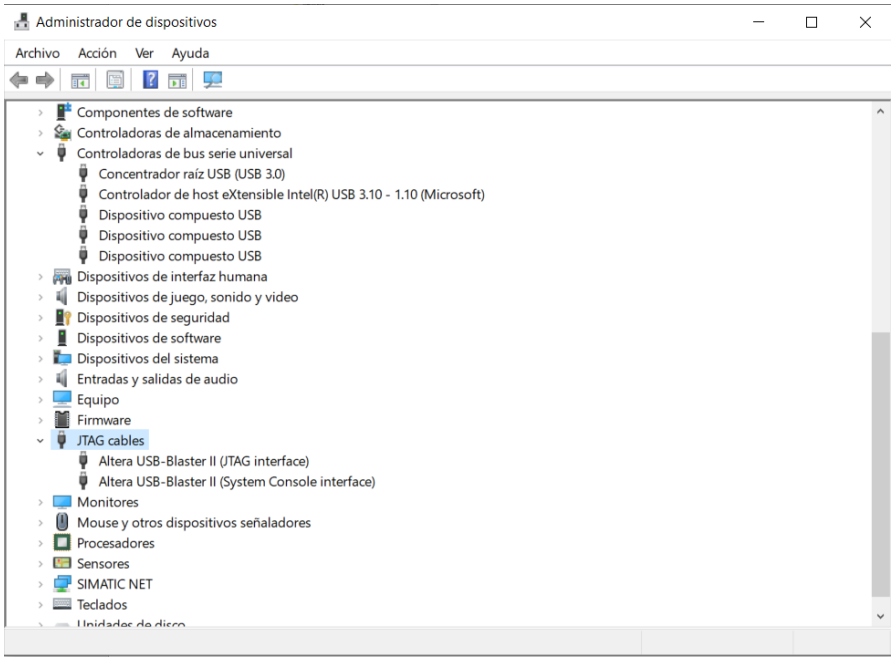


Figura 2.15: Dispositivo JTAG cables detectado

2.7. Configuración de pines de la FPGA DE10-Nano

La FPGA DE10-Nano se puede configurar desde la EPCS (EPCS128) o HPS (del inglés *Hard Processor System*), en este caso se realiza desde la EPCS 128, por ende, se utilizan los pines MSEL [4:00] para seleccionar el tipo de configuración. MSEL está representado en la placa DE10-Nano como un interruptor DIP de 6 pines (SW10), tal como se indica en la figura 2.16:

Figura 2.16: Interruptor DIP de la FPGA DE10-Nano

A continuación, en la tabla 2.3, se muestra la configuración que posee SW10, DIP de 6 pines:

Tabla 2.3: Configuración del SW10, DIP de 6 pines

Referencia de la Tarjeta	Nombre del Pin	Descripción	Valor Lógico
SW10.1	MSEL0	Pines para la configuración de la FPGA	ON<= '0', OFF<= '1'
SW10.2	MSEL1		
SW10.3	MSEL2		
SW10.4	MSEL3		
SW10.5	MSEL4		
SW10.6	N/A	N/A	N/A

En base a la tabla 2.4, la configuración que se realiza para que la FPGA funcione desde la EPCS 128 se dan en el SW10, la sección MSEL [4:0] debe ser "010010", luego de esto la tarjeta puede ser encendida normalmente.

Tabla 2.4: Ajustes de los pines MSEL para el modo de configuración de la FPGA DE10-Nano

Configuración	SW10.1 MSEL 0	SW10.2 MSEL 1	SW10.3 MSEL 2	SW10.4 MSEL 3	SW10.5 MSEL 4	SW10.6	Descripción
EPCS 128	ON	OFF	ON	ON	OFF	N/A	La FPGA configurada desde la EPCS.
HPS- LXDE Linux	ON	OFF	ON	OFF	ON	N/A	FPGA configurada desde el software HPS con una imagen cargada desde una tarjeta SD usando el escritorio LXDE de Linux
HPS	ON	ON	ON	ON	ON	N/A	FPGA configurada desde el software HPS con una imagen cargada desde una tarjeta SD.

2.8. Configuración del dispositivo en el software Quartus II

Dentro del software Quartus II es necesario definir el tipo y familia del dispositivo con el que se realizarán las pruebas del microcontrolador. En este caso, como se definió anteriormente en las características de la FPGA DE10-Nano, se seleccionó la familia Cyclone V (SE) con el respectivo nombre del dispositivo 5CSEBA6U23I7.

Si se definiese un dispositivo equivocado, los archivos que generaría el software no podrían ejecutarse correctamente en la FPGA que se posee físicamente, por lo que se obtendrían resultados erróneos al momento de probar el funcionamiento del microcontrolador embebido.

La figura 2.17 muestra la pantalla de selección en Quartus II con el dispositivo que corresponde a la FPGA DE10-Nano.

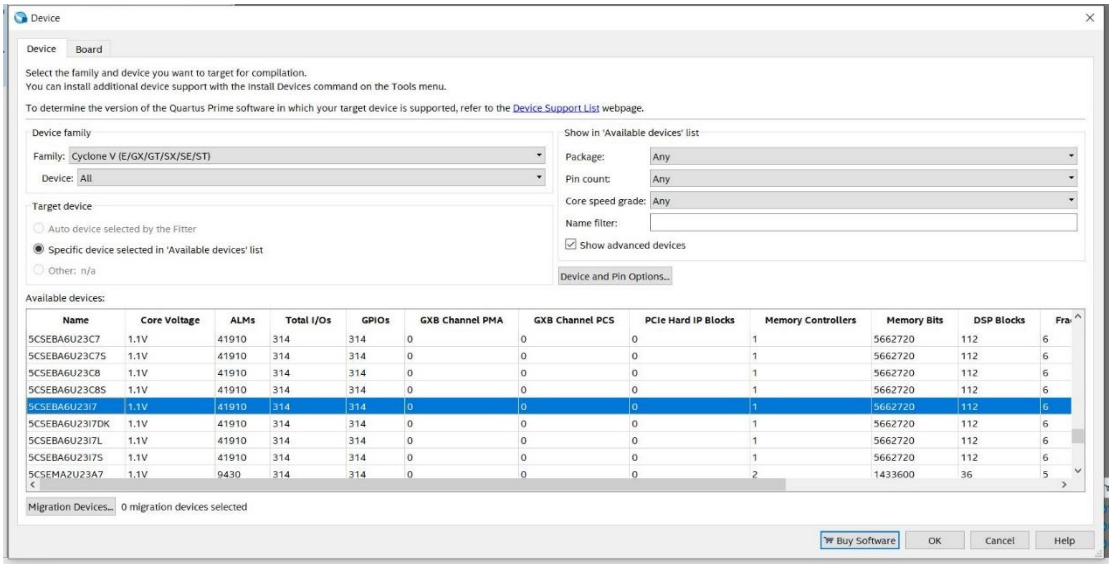


Figura 2.17: Selección de dispositivo en Quartus II

2.9. Configuración de la ventana EDA Tool Settings

Como se lo mencionó anteriormente, la arquitectura del microcontrolador será descrita con un lenguaje de descripción de hardware como lo es VHDL, por lo que Quartus II permitirá ejecutarlo. También es importante configurar la opción que le permita visualizar la simulación, por lo que tiene que seguir los siguientes pasos:

- Una vez configurada la tarjeta de nuestro dispositivo se le abre la siguiente ventana de la configuración del 'EDA Tool Settings', se dirige a Simulation y en la columna Tool Name y seleccione 'ModelSim-Altera'. Esto le va a permitir realizar simulaciones en modo Timing y Funtional.

- Ahora en la columna Formats y fila de Simulation, seleccione la estructura de lenguaje VHDL, debido a que la programación se desarrollará en este lenguaje.

- Una vez realizado lo anterior, se da click en Next.

La figura 2.18 muestra la configuración del EDA Tool Settings del proyecto.

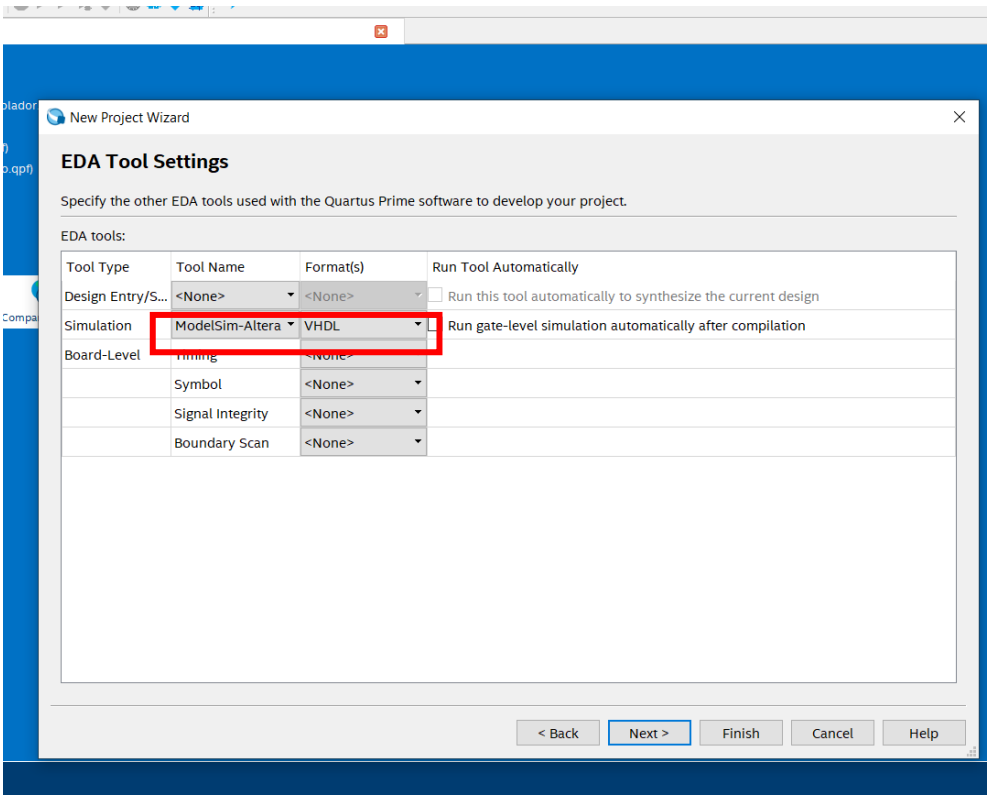


Figura 2.18: Configuración de la ventana EDA Tool Settings

2.10. Archivo VHDL

Un archivo VHDL es una hoja de trabajo que le permite crear la programación de un elemento lógico sincrónico o asincrónico, por lo que está escrito en dicho lenguaje. Consiste en un programa que realiza la descripción de circuitos

digitales, es decir toma el código fuente que le permite diseñar circuitos que operan a una señal de reloj, permitiendo detectar errores o problemas en el diseño antes de su implementación física, facilitando el diseño del circuito. En este caso para la creación de un nuevo archivo VHDL se tiene que tener en cuenta los siguientes pasos:

- Una vez configurado el archivo del proyecto creado, se mostrará la ventana de trabajo de Quartus II, la cual permite crear archivos VHDL, Block Diagram y VWF.
- A continuación, se dirige al ícono denominado NEW y se mostrará la ventana de la figura 2.19, la cual contiene todos los archivos para la creación de componentes electrónicos digitales.

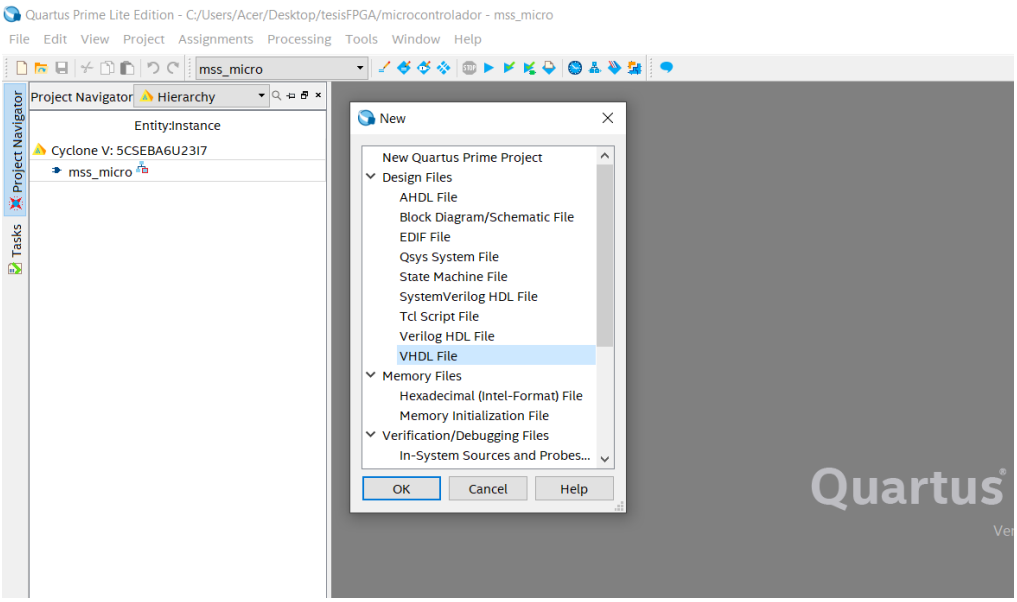


Figura 2.19: Generación de archivo VHDL

- Se busca el nombre del archivo VHDL y se lo selecciona, click en OK.
- Se mostrará una ventana como en la figura 2.20, que contiene la hoja de trabajo lista para la programación de cada uno de los bloques.

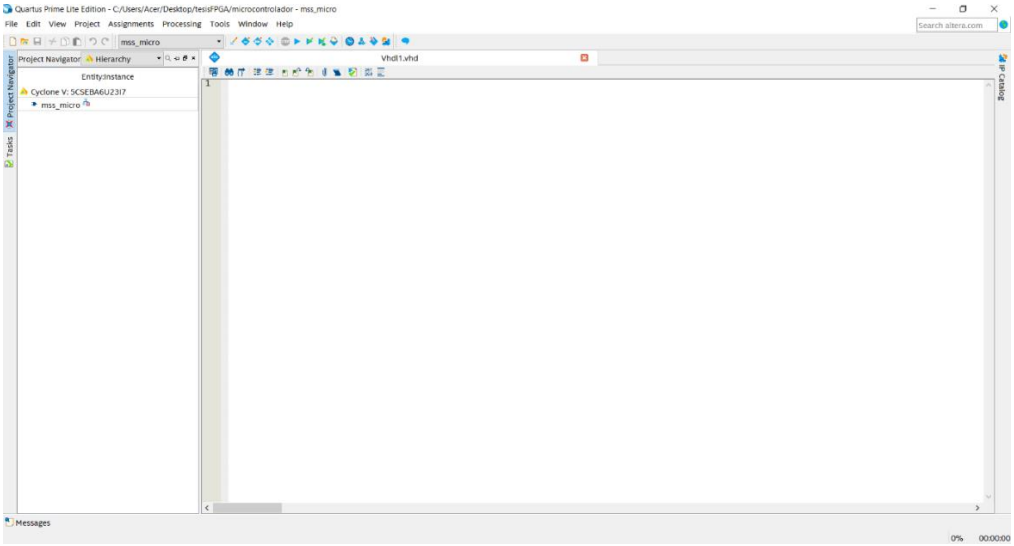


Figura 2.20: Hoja de trabajo del archivo VHDL

2.11. Proceso de programación de bloques

Como se mencionó anteriormente, la arquitectura del microcontrolador estará conformada de dos memorias separadas, como lo es la memoria del programa (ROM) y la memoria de datos (RAM) que estarán conectadas por medio de señales de datos y señales de control, todas estas controladas por el decodificador de instrucciones. Siendo este último gobernado por un contador de programa, el cuál otorgará las direcciones de la memoria del programa. Las operaciones quedarán descritas por la ALU y serán enviadas a los registros acumuladores según sea el caso de la programación. La figura 2.21 muestra el diseño de uno de los bloques de nuestra arquitectura, el bloque contador de programa.

Un sistema digital está compuesto de manera general por las entradas, las salidas y la retroalimentación que las componen. Para VHDL es necesario describir los componentes externos e internos del circuito y la relación entre sus entradas y salidas que lo componen. Por ello se lo divide en dos estructuras importantes, el ‘entity’, y el ‘architecture’. En el entity se describirán los puertos de entrada y salida que componen el contador de programa, para este caso estará conformado de un puerto de entrada de 8 bits denominado Ent, y 4 señales de un solo bit denominadas Load, En, resetn y clock y el puerto de salida Q que igualmente es de 8 bits.

Para este caso particular la función del contador incrementará su valor de salida siempre y cuando la señal ‘En’ coincida con el flanco negativo de la señal ‘clock’ y ‘resetn’ tenga un alto lógico. En cambio, cargará el valor de entrada siempre que la señal ‘Load’ tenga un alto lógico y coincida con el flanco negativo de nuestro ‘clock’. En base al conocimiento anterior se procedió a realizar la programación usando condicionales con la estructura if/elsif/else.

La ilustración siguiente muestra el código correspondiente para el contador de programa, se puede visualizar las señales del puerto del bloque dentro del ‘entity’ y dentro de la ‘architecture’ el funcionamiento del programa.

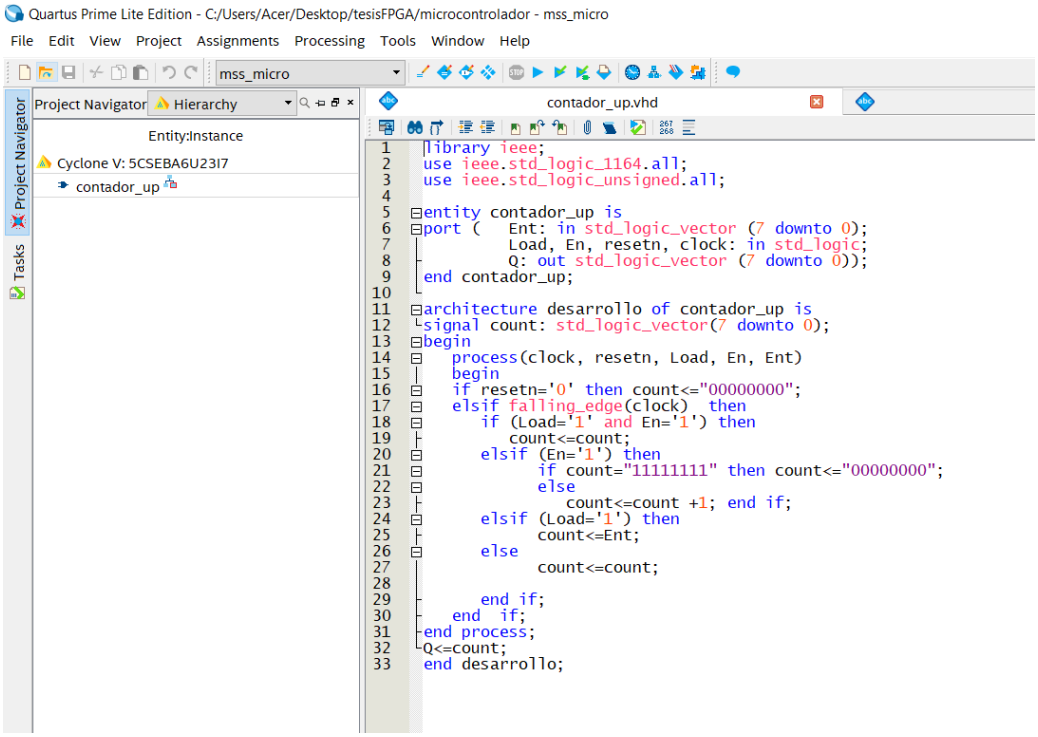


Figura 2.21: Código VHDL del contador up

Una vez culminado el proceso de programación, se procedió a realizar la respectiva compilación por medio de estos pasos.

- Guarde el archivo con extensión. vhdl con el nombre agregado dentro del ‘entity’, si no es el mismo arrojará un error en la compilación.
- Diríjase al ‘Project Navigator’ y seleccione con click derecho el archivo y escoja la opción que dice ‘Set as Top-Level Entity’, como se muestra en la figura 2.22, esto le permitirá que su archivo posea el nivel alto en la jerarquía de programación.

- Luego en la barra de herramientas seleccione el ícono ‘Start Compilation’, ahora comenzará la compilación del programa.
- Al finalizar la compilación puede observar el reporte generado dentro de la hoja del trabajo denominado ‘Compilation Report-nombre_proyecto’, en este caso el proceso fue realizado satisfactoriamente.

En la parte inferior, dentro del bloque de texto denominado ‘Messages’, en la opción ‘Processing’ se puede observar los comentarios de la compilación, como ejemplifica la figura 2.23, entre los mensajes que se pueden obtener pueden ser de advertencia o de error, para el último caso Quartus II automáticamente describe el error y arroja una posible solución para evitarlo, dándole una ventaja más, que lo hace una herramienta más útil para la programación de circuitos lógicos sincrónicos.

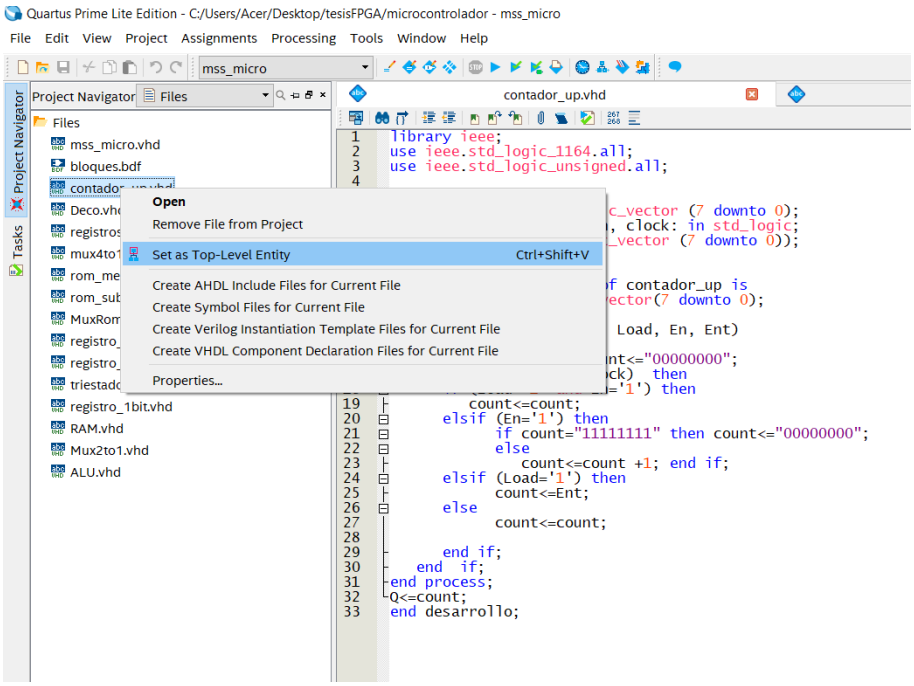


Figura 2.22: Asignación del Top-Level Entity del archivo VHDL del contador

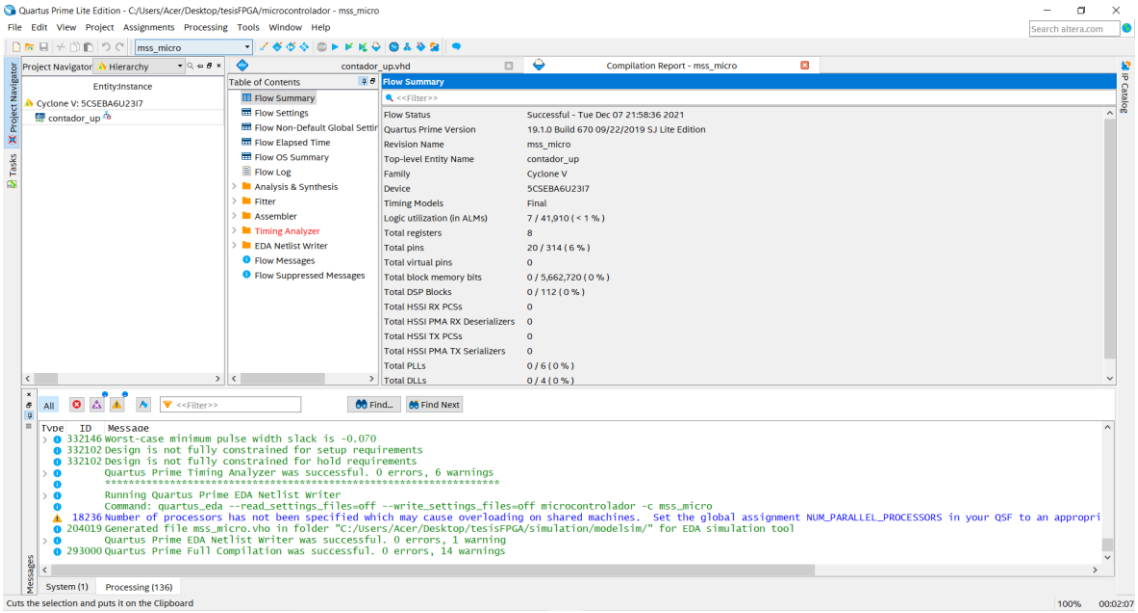


Figura 2.23: Visualización de la compilación del contador

2.11.1. Generación del archivo Symbol del bloque VHDL

Para la programación gráfica con diagrama de bloques es necesario crear el bloque correspondiente del archivo vhdl anteriormente compilado, para ello realizará lo siguiente:

- Nuevamente se dirige al 'Project Navigator' y haga clic sobre el archivo a generar el bloque, en este caso corresponde al count_up, click derecho y seleccione la opción denominada 'Create Symbol Files for Current File'.
- Inmediatamente comenzará la generación del archivo, el tiempo que le toma es muy inferior a la compilación del archivo. La figura 2.24 muestra el proceso de selección y generación del archivo.

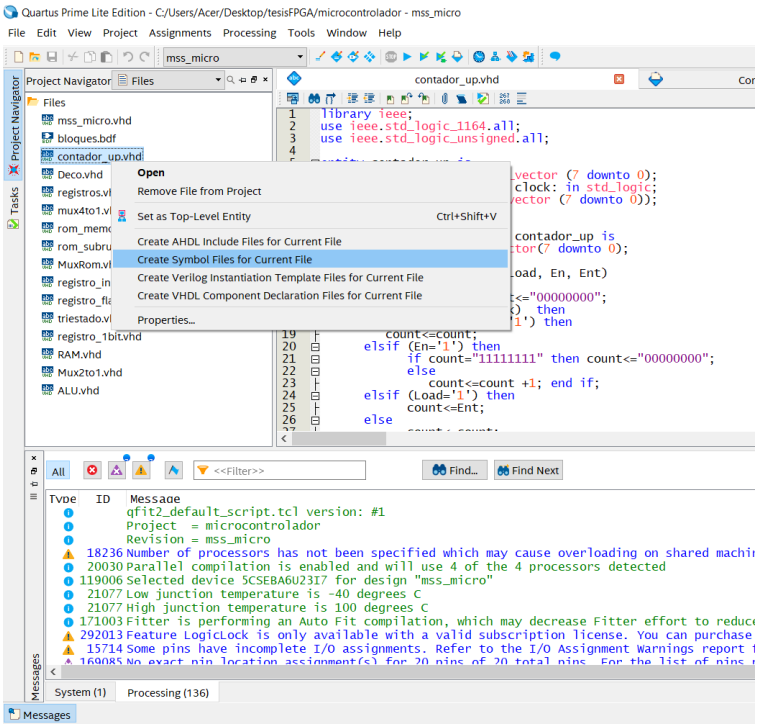


Figura 2.24: Creación del archivo de símbolo del contador

2.11.2. Generación del Block Diagram/Schematic File

Terminado de generar el bloque del contador_up se dirige a la opción New ubicada en la barra de herramientas del software y seleccione la pestaña Designs File, la opción ‘Block Diagram/Schematic File’. A continuación, se le generará automáticamente el archivo .bdf.

Para agregar el bloque count_up creado haga doble click izquierdo dentro de la hoja de trabajo denominado ‘Block1.bdf’ y se mostrará un cuadro llamado ‘Symbol’, allí diríjase a la carpeta ‘Project’ y se presentará el archivo creado, selecciónelo y de manera inmediata se agregará en la hoja de trabajo, este proceso se ilustra en la figura 2.25.

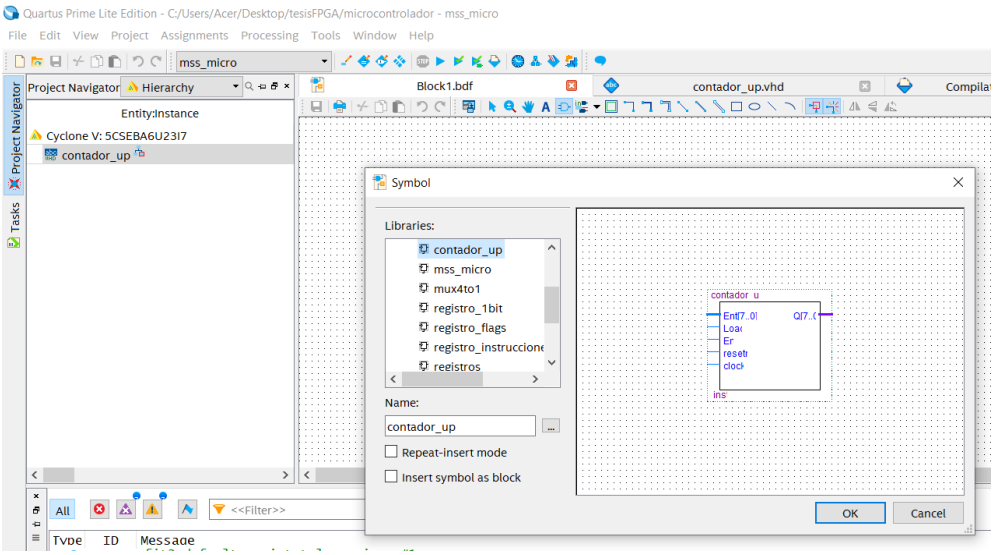


Figura 2.25: Ventana de selección del bloque a colocar

Esta metodología se repite para todos los demás bloques que conformarán la arquitectura del microcontrolador creado. Se debe tener en cuenta que para cada bloque debe asignarle el nivel alto de jerarquía para la respectiva simulación 'entity level'. Una vez que se hayan creado todos los bloques e importado a la hoja de trabajo del archivo .bdf se procede a realizar su respectiva conexión entre los terminales, tanto para las señales de un solo bit emitidas por la MSS y el decodificador de instrucciones, al igual que las señales del bus, dadas entre los multiplexores, decodificadores, registros, memorias y MSS.

2.12. Arquitectura Propuesta

El microcontrolador a desarrollar debe ser capaz de cumplir con la ejecución de las instrucciones detalladas en la sección 2.4. Teniendo como base el diagrama de la figura 2.8, fueron realizadas múltiples alteraciones en este diseño, entre lo que se incluye haber agregado registros, selectores y modificado la cantidad de bits de los buses de datos y direcciones.

Adicionalmente, el diagrama ASM del controlador (MSS) fue alterado ligeramente para permitir la generación de más retardos, consiguiendo con esto la sincronización de señales del circuito. Tomando en cuenta que fueron

añadidos elementos con respecto al diseño base, la programación del decodificador de instrucciones también fue modificada, permitiendo agregar señales de salida para controlar el funcionamiento de los nuevos bloques y poder operarlos en conjunto con los ya existentes.

El bloque de la memoria del programa está conformado por un set de instrucciones que consta de 19 bits, mostrado en la figura 2.26, donde se almacena información como lo es el código de operación, modos de direccionamiento, manejo de registros y de multiplexores, adicional un byte de propósito general, que puede ser usado para direcciones u operandos.

Carry In	Instruction	Purpose General	Addressing	Source	Destination
1 bit	6 bits	1 byte	2 bits	1 bit	1 bit

Figura 2.26: Arquitectura del set de instrucciones

De manera general la arquitectura muestra el programa principal que se encuentra almacenado en la memoria ROM. Los datos estarán almacenados en la memoria RAM, se han colocado los multiplexores que permiten direccionar los datos entre los diversos bloques (los presentes en la arquitectura base y los añadidos), registros acumuladores y temporales para el almacenamiento de datos y las señales para representar la información de entrada y salida provenientes de los puertos físicos de la tarjeta de desarrollo.

El diseño de la arquitectura se encuentra conformado por 11 bloques principales, los cuales son el contador de programa, ROM de instrucciones, registro de instrucciones, decodificador de instrucciones, unida de control o MSS, registros acumuladores, memoria RAM, generador PWM, ALU, temporizador y los puertos de E/S. Como se muestra en la figura 2.27, la arquitectura posee buses de dirección, buses de datos e instrucciones y señales de control provenientes de la MSS y del decodificador de instrucciones.

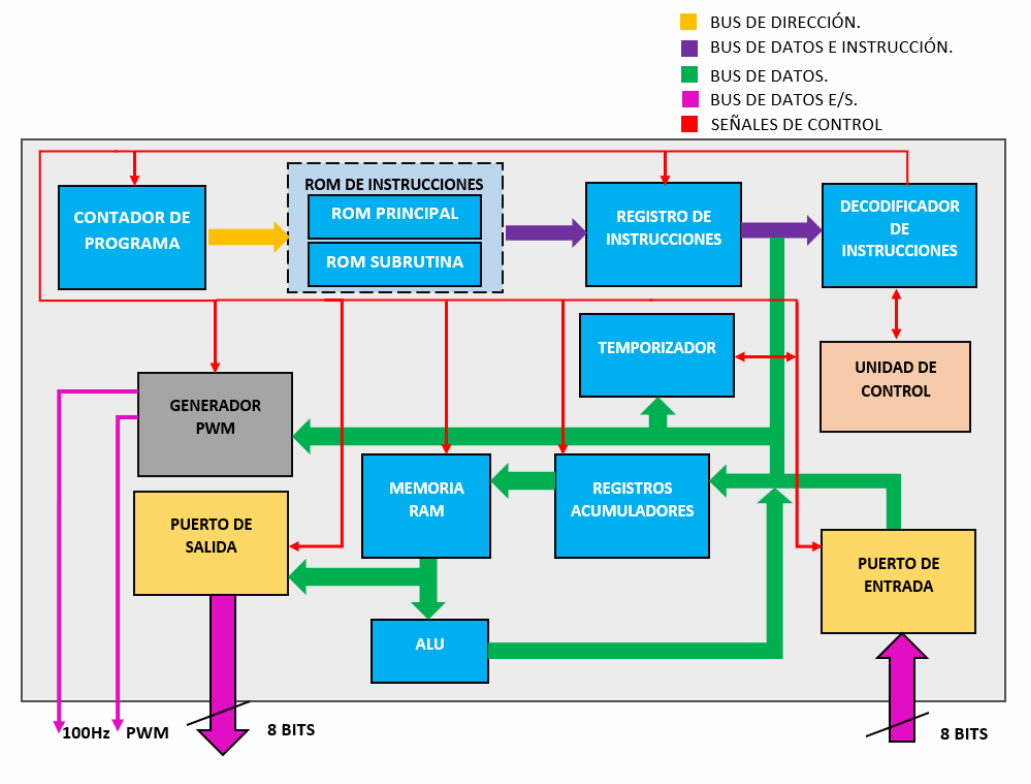


Figura 2.27: Diagrama de bloques de la arquitectura diseñada del microcontrolador

A continuación, en la figura 2.28, se muestra el diagrama completo de la arquitectura propuesta desarrollada en el software Quartus II, donde se visualiza cada uno de los bloques con su respectivo bus de datos, instrucciones y señales provenientes del decodificador de instrucciones y de la MSS interconectados entre ellos.

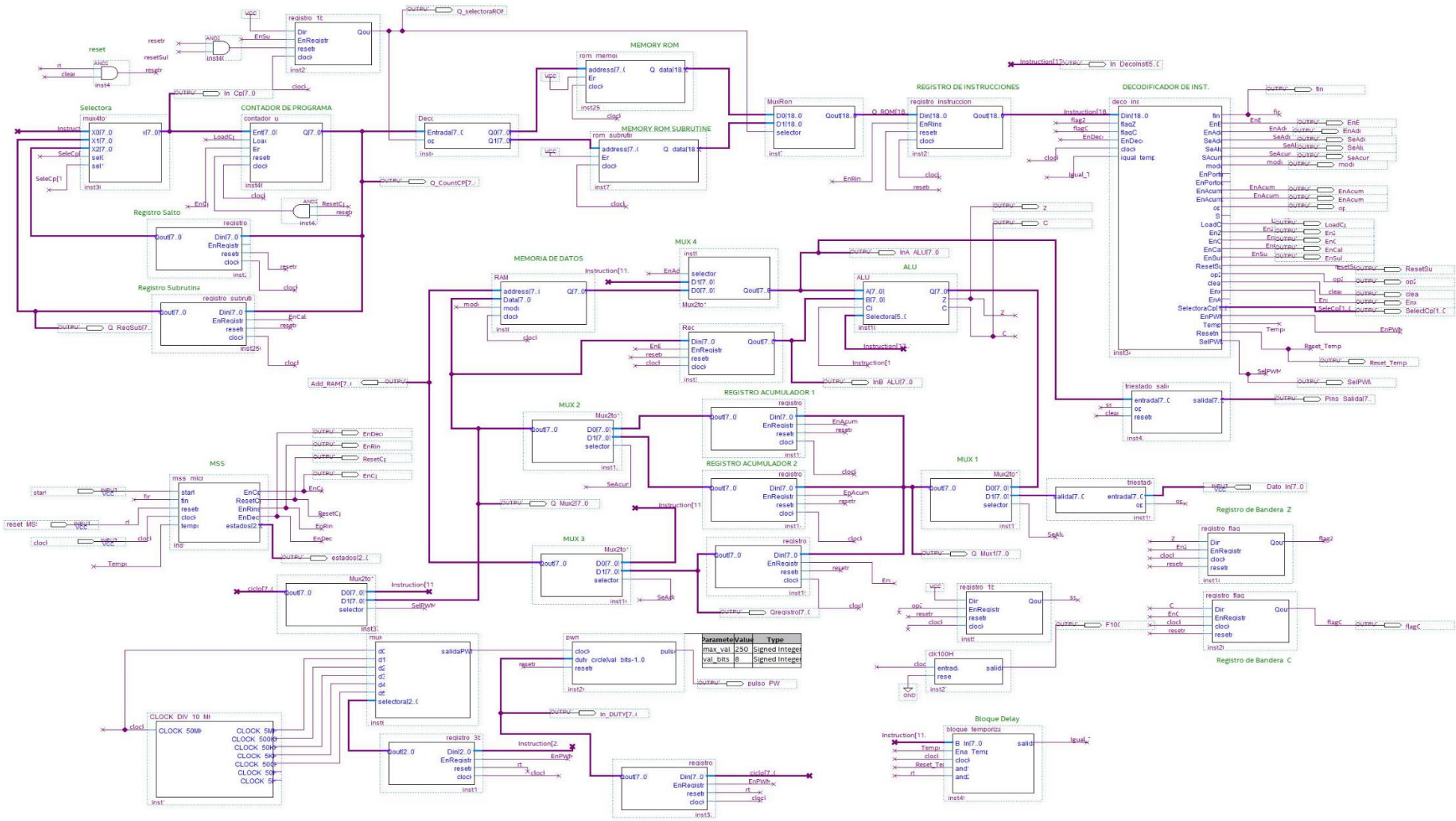


Figura 2.28: Diagrama de la arquitectura del microcontrolador propuesto

2.12.1. Funcionamiento del microcontrolador

Para su funcionamiento, el microcontrolador sigue una secuencia como la descrita en la figura 2.29. Realiza lectura, búsqueda, decodificación y ejecución de las instrucciones que se encuentren en la memoria ROM.

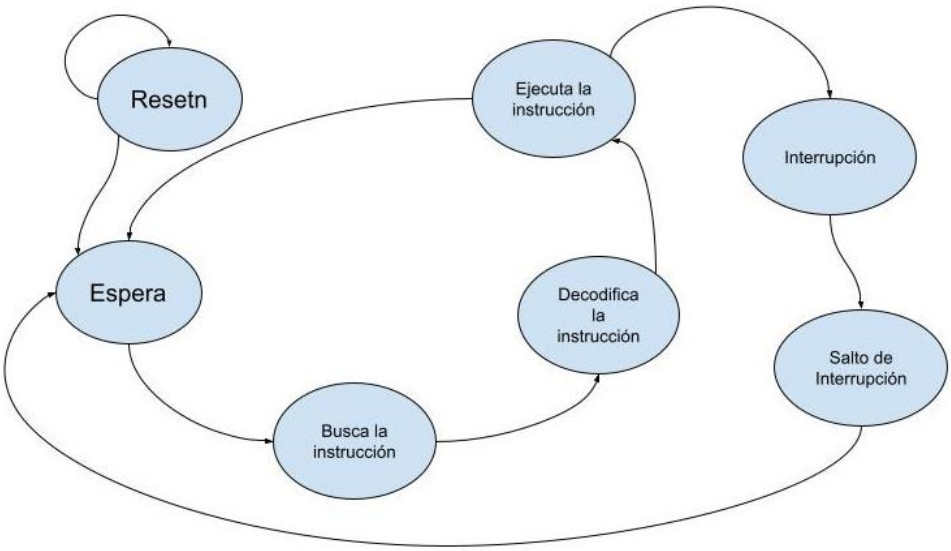


Figura 2.29: Diagrama de funcionamiento del microcontrolador

Con el fin de cumplir con algunas instrucciones, fue necesario añadir secciones en las que el microcontrolador pudiera realizar interrupciones, específicamente para las funciones de salto implementadas, las cuales permiten que se ejecute código de la memoria ROM desde una dirección especificada por el salto de la instrucción.

2.12.2. MSS de la arquitectura propuesta

Debido al aumento de nuevos elementos requeridos para la arquitectura propuesta, es necesario que las señales para activar cada uno de los componentes lleguen de manera oportuna cuando son requeridas, existiendo problemas con la sincronización de los bloques del microcontrolador si no es corregido este problema.

En los casos de falta de sincronización, se requiere que la MSS sea capaz de generar retardos para cumplir con la sincronía de señales. Por este motivo se agregó un estado vacío en la programación destinada a la

arquitectura propuesta. Adicionalmente, fue añadida una señal de entrada que permite establecer un retardo específico dentro del programa para el ingreso de datos por el puerto de entrada. La figura 2.30 presenta el diagrama de la MSS.

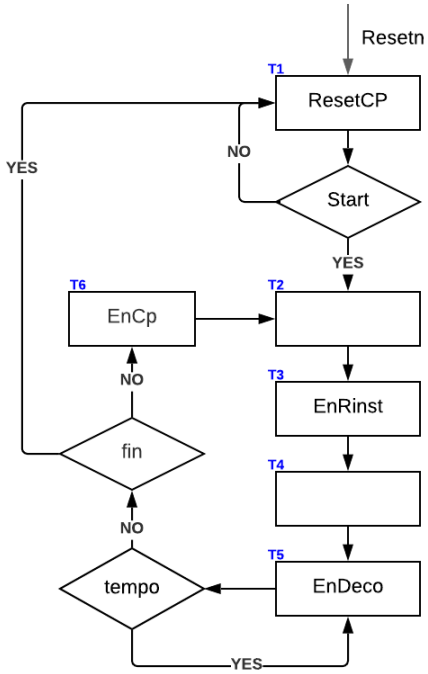


Figura 2.30: Diagrama ASM de la MSS de la arquitectura propuesta

El funcionamiento de la MSS para la arquitectura propuesta resulta ser similar al descrito en la sección 2.3.1, presentando su carácter cíclico con el que se activan las señales necesarias en el circuito para generar la búsqueda, lectura, decodificación y ejecución de las instrucciones. Además, requiere de la señal ‘Start’ para iniciar su funcionamiento y de ‘fin’ para detenerse. En el estado ‘T5’ se requiere conocer el estado de dos entradas. La primera corresponde a ‘tempo’, señal proveniente del decodificador de instrucciones que permite establecer un retardo dentro del código de programa, cuya funcionalidad se especificó anteriormente, y la segunda es ‘fin’, señal con la que se da por terminado el código de programa. Si ambas señales no se encuentran activadas, la MSS se dirige al estado T6 para habilitar nuevamente el contador de programa, caso contrario se dirige al estado T1 para finalizar la lectura del programa.

2.12.3. Manejo de la memoria ROM

En la realización de las pruebas para comprobar el funcionamiento de la arquitectura, es necesaria la compilación y ejecución de cada uno de los bloques que la componen, con el fin de obtener resultados de la codificación presente en la memoria ROM. Esta metodología estándar utiliza una significativa cantidad de tiempo con cada comprobación que realiza el software Quartus II, por lo que se opta por una vía que simplifique el poder ingresar el código de la tarea en la memoria del programa de manera rápida y sencilla.

Para cumplir con este cometido se utilizan archivos de extensión .mif, en los cuales se encuentra presente la codificación, similar a un archivo de texto común, que corresponde al programa que se desea embeber en la FPGA. De esta forma, solo se requiere ejecutar el archivo que comprende la totalidad de la arquitectura y en la memoria ROM se especifica la dirección del archivo .mif, facilitando el cambio de código entre pruebas.

La creación de este tipo de archivos es de la siguiente manera:

1. En la pestaña File, se escoge la opción New.
2. Se busca en la sección Memory Files la opción Memory Initialization File y se presiona OK, como muestra la figura 2.31.

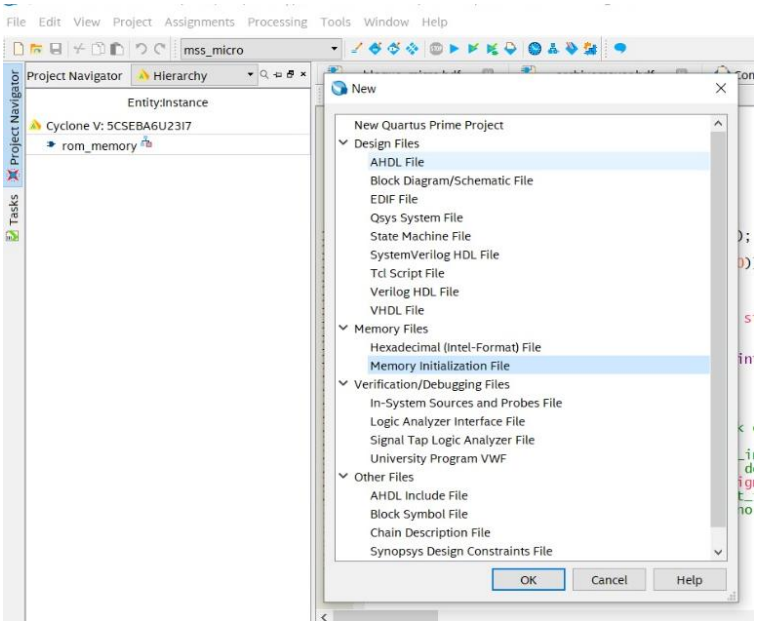


Figura 2.31: Creación del archivo con extensión .mif

3. En la ventana que aparece se ingresa la cantidad de palabras y el tamaño de estas, en base a las especificaciones que se requiere para el manejo de la codificación del proyecto, esta configuración se realiza como lo muestra la figura 2.32.

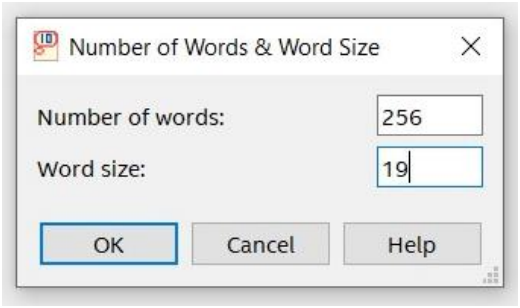


Figura 2.32: Configuración del archivo .mif

CAPÍTULO 3

3. SIMULACIONES DEL MICROCONTROLADOR Y RESULTADOS EXPERIMENTALES

3.1. Simulaciones

Para comprobar el funcionamiento de los bloques configurados en el microcontrolador, se procedió con la simulación de la forma de onda (waveform simulation) que representa cambios de estado y activación de salidas. En este caso, se añadieron salidas adicionales con las que se monitoreaban los estados en los que se encontraban diversos bloques de la arquitectura creada.

Para las simulaciones presentadas a continuación dentro del 'Waveform Simulation Editor' se mostrarán algunas señales internas de los bloques que constituyen la arquitectura del microcontrolador diseñado. Entre dichas señales se tiene:

- La señal 'clock' representa los flancos de subida y bajada que son utilizados por los bloques para detectar cuando es necesario dar paso a la siguiente línea de código de su programación interna. Puede observarse que los cambios ocurren durante los flancos descendentes.
- 'Start' es utilizado para poner en funcionamiento el sistema, extendiendo su duración para asegurar la recepción de la señal por el sistema durante el flanco correspondiente.
- 'Resetn' se mantiene lo largo de la simulación, excepto durante el instante inicial en que la señal de inicio es detectada. Esto debido a la lógica de programación utilizada en esta señal, con la que el sistema es reseteado si no se encuentra activa.
- 'EnCp' muestra los instantes en que el habilitador del contador del programa es activado.

- En la salida ‘Q_Cp’ se puede visualizar el número en el que se encuentra el contador durante cada intervalo de tiempo, se puede notar que el cambio de esta salida se da cada que ‘EnCp’ coincide con el flanco descendente del reloj.
- La salida ‘estados’ muestra de manera numérica el estado en el que se encuentra la MSS. Puede notarse que inicialmente se realiza la transición de estados desde el 1 al 6, y que para cada conteo sucesivo esto ocurre del 2 al 6, esto debido a la propiedad cíclica del sistema. En el caso de la última secuencia, esta va desde el 2 hasta el 5. Se debe a que la señal de finalización fue generada y el sistema regresa a su estado inicial (el 1) a la espera de una nueva puesta en marcha.
- ‘Q_ROM’ se encarga de mostrar la información que se encuentra en la salida del bloque de la memoria ROM. Cada código de la ROM presenta una duración que concuerda con un ciclo completo de la MSS, debido al control que esta ejerce sobre el contador que permite la lectura de un siguiente código en la memoria del programa.
- Las señales habilitadoras que utilizan los bloques del sistema (EnB, EnAdd, SeAdd, SeAlu, SeAcum, modo, EnAcum1, EnAcum2, op, LoadCp, EnZ, EnC, EnCall, EnSub, ResetSub, op2, clear, EnX y EnPWM) se encuentran activadas durante pequeños intervalos en los que se requieren de acuerdo al código de instrucción ejecutado. Se aprecia que cada una de ellas se activa durante el respectivo estado de la MSS, estado 5, tal como se programó en un inicio.
- La señal ‘fin’ es el indicador que permite a la MSS terminar con su operación. Su activación se da una única vez y con esta el controlador retorna a su estado inicial, así como también el código de la memoria ROM regresa a ser el primero que se encuentre configurado.
- ‘Q_ALU’ muestra los resultados que se obtienen en la salida de la ALU, en caso de que el programa integre instrucciones que requieran de esta, durante la ejecución del programa.

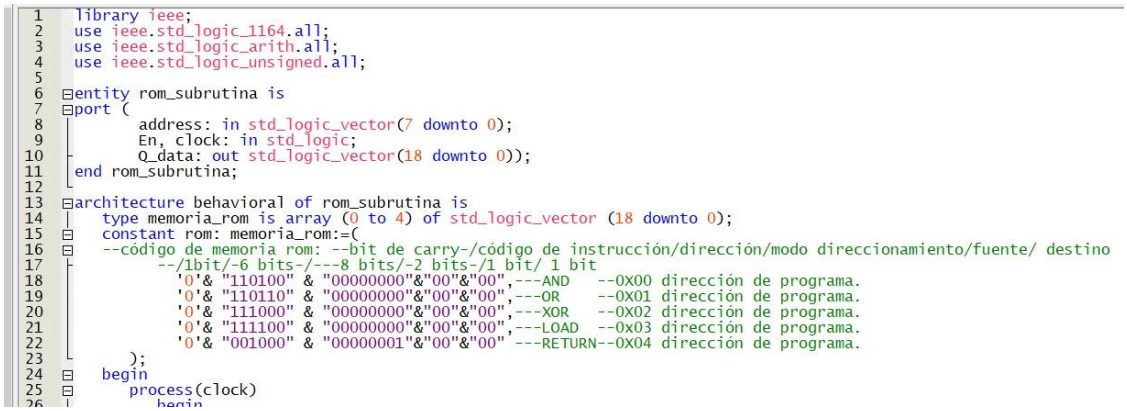
3.1.1. Primera simulación dentro del Waveform Editor

Para esta simulación se probarán algunas de las instrucciones enlistadas en la Tabla 2.1 y Tabla 2.2, adicional esperando tener como resultado el número 7 procedente del código de operación suma. Para ello se grabarán dos archivos de código de set de instrucciones, en la memoria rom principal y en la memoria rom de subrutina respectivamente. La figura 3.1 muestra el código correspondiente para la memoria principal. Aquí se visualiza el ingreso y almacenamiento de los datos, al igual que la operación de suma, el llamado de subrutina, la activación de salida y la finalización de la programación. Para lo indicado se utilizaron las instrucciones INPUT, STORE, LOAD, MOVE, ADD, CALL, OUTPUT y FIN.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity rom_memory is
port (

       address: in std_logic_vector(7 downto 0);
       En, clock: in std_logic;
       Q_data: out std_logic_vector(18 downto 0));
end rom_memory;

architecture behavioral of rom_memory is
   type memoria_rom is array (0 to 10) of std_logic_vector (18 downto 0);
   constant rom: memoria_rom:=(
   --código de memoria rom: acarreo de entrada/código de instrucción/dirección/modo direccionamiento/fuente/ destino
        --1 bit--/---6 bits----------/---8 bits/-------2 bits--------/-1 bit/ 1 bit--
         '0' & "001100" & "00000000"&"00"&"00",--INPUT   --0x00 dirección del programa--
         '0' & "001010" & "00000000"&"01"&"00",--STORE   --0x01 dirección del programa--
         '0' & "001100" & "00000000"&"00"&"00",--INPUT   --0x02 dirección del programa--
         '0' & "001010" & "00000001"&"01"&"00",--STORE   --0x03 dirección del programa--
         '0' & "111110" & "00000000"&"00"&"00",---MOVE   --0x04 dirección del programa--
         '0' & "111100" & "00000000"&"01"&"00",--LOAD    --0x05 dirección del programa--
         '0' & "100000" & "00000000"&"01"&"00",---ADD    --0x06 dirección del programa--
         '0' & "001010" & "00000010"&"01"&"00",---STORE  --0x07 dirección del programa--
         '0' & "000110" & "00000000"&"00"&"00",---CALL   --0x08 dirección del programa--
         '0' & "001110" & "00000010"&"00"&"00",---OUTPUT --0x09 dirección del programa--
         '0' & "010100" & "00000010"&"00"&"00"---FIN     --0x10 dirección del programa--
```

Figura 3.1: Código del programa para la memoria ROM principal

Cuando se ejecute la instrucción de llamado de subrutina inmediatamente se almacenará el valor del contador actual para posteriormente hacer la lectura en el archivo rom_subrutina.vhd que corresponde a una memoria de subrutina que contiene un programa de prueba mostrado en la figura 3.2. Al final del código se tiene la instrucción RETURN, que permite regresar a la memoria principal del programa y continuar en el instante en el que se abandonó el mismo.



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity rom_subrutina is
port (
       address: in std_logic_vector(7 downto 0);
       En, clock: in std_logic;
       Q_data: out std_logic_vector(18 downto 0));
end rom_subrutina;

architecture behavioral of rom_subrutina is
   type memoria_rom is array (0 to 4) of std_logic_vector (18 downto 0);
   constant rom: memoria_rom:=(
   --código de memoria rom: --bit de carry-/código de instrucción/dirección/modo direccionamiento/fuente/ destino
        --/1bit/-6 bits-/---8 bits/-2 bits-/1 bit/ 1 bit
          '0'& "110100" & "00000000"&"00"&"00",---AND    --0X00 dirección de programa.
          '0'& "110110" & "00000000"&"00"&"00",---OR     --0X01 dirección de programa.
          '0'& "111000" & "00000000"&"00"&"00",---XOR    --0X02 dirección de programa.
          '0'& "111100" & "00000000"&"00"&"00",---LOAD   --0x03 dirección de programa.
          '0'& "001000" & "00000001"&"00"&"00" ---RETURN--0X04 dirección de programa.
     );
   begin
      process(clock)
         begin
```

Figura 3.2: Código de programa para la memoria ROM de subrutina

Una vez ejecutado el Simulation Waveform Editor se pueden observar las señales activadas dentro de dicha ventana. En la figura 3.3 se observa la simulación correspondiente al código antes mencionado. El microcontrolador funciona de manera síncrona con una señal de reloj, en este caso es de 10 ns. Luego, la señal reset_MSS tiene que estar en un nivel lógico alto. Para que el sistema inicie la señal 'start' se mantendrá en nivel alto por un instante caso contrario el estado en que se encontraría la MSS correspondería al estado 1 del diagrama ASM. Por la señal Dato_in se ingresarán los dos números a guardar dentro de la memoria RAM. Una vez en marcha el sistema, la salida del contador comienza con cero por lo que buscará la dirección 0x00 de la memoria ROM principal, que contiene la instrucción 'INPUT', a su vez los estados de la MSS comienzan a incrementar. Esto se puede observar en la señal estados que contiene los estados del 1 al 6. A su vez Q_CountCP se encuentra en cero, como antes se mencionó. Cuando se llega al estado 3, se activa en el flanco positivo la señal EnRins que es la señal habilitadora del registro de instrucciones. Posteriormente, en el estado 5 se habilita la señal EnDeco, habilitando la señal 'op' que permite el ingreso del primer valor binario. Luego en el siguiente flanco ascendente se dirige al estado 6 donde se habilita la señal 'EnCp' que incrementa la señal 'Q_CountCP' a 1, lo que regresa nuevamente al estado 2 y ahora se ejecuta el set de instrucción en la dirección 0x01, que contiene la instrucción 'STORE', por ende, observe que se activa la señal 'modo' para guardar dicho valor antes del ingreso en la RAM. De esa manera cíclica se ejecuta cada una de las instrucciones hasta que se llega a la instrucción 'CALL' que corresponde a la salida Q_CountCP igual a

8. Dicho valor se guarda y el contador ahora se dirige a la dirección 0X00 de la memoria de subrutina hasta llegar a la dirección 0x04. Una vez allí se encuentra con la instrucción ‘RETURN’ que regresa a la memoria ROM principal y Q_CountCP continúa con el valor antes almacenado. Por último, en Pins_Salida se muestra el valor de 7, correspondiente a la instrucción ‘ADD’ y se finaliza el programa con la instrucción ‘FIN’ con Q_CountCP igual a 10.

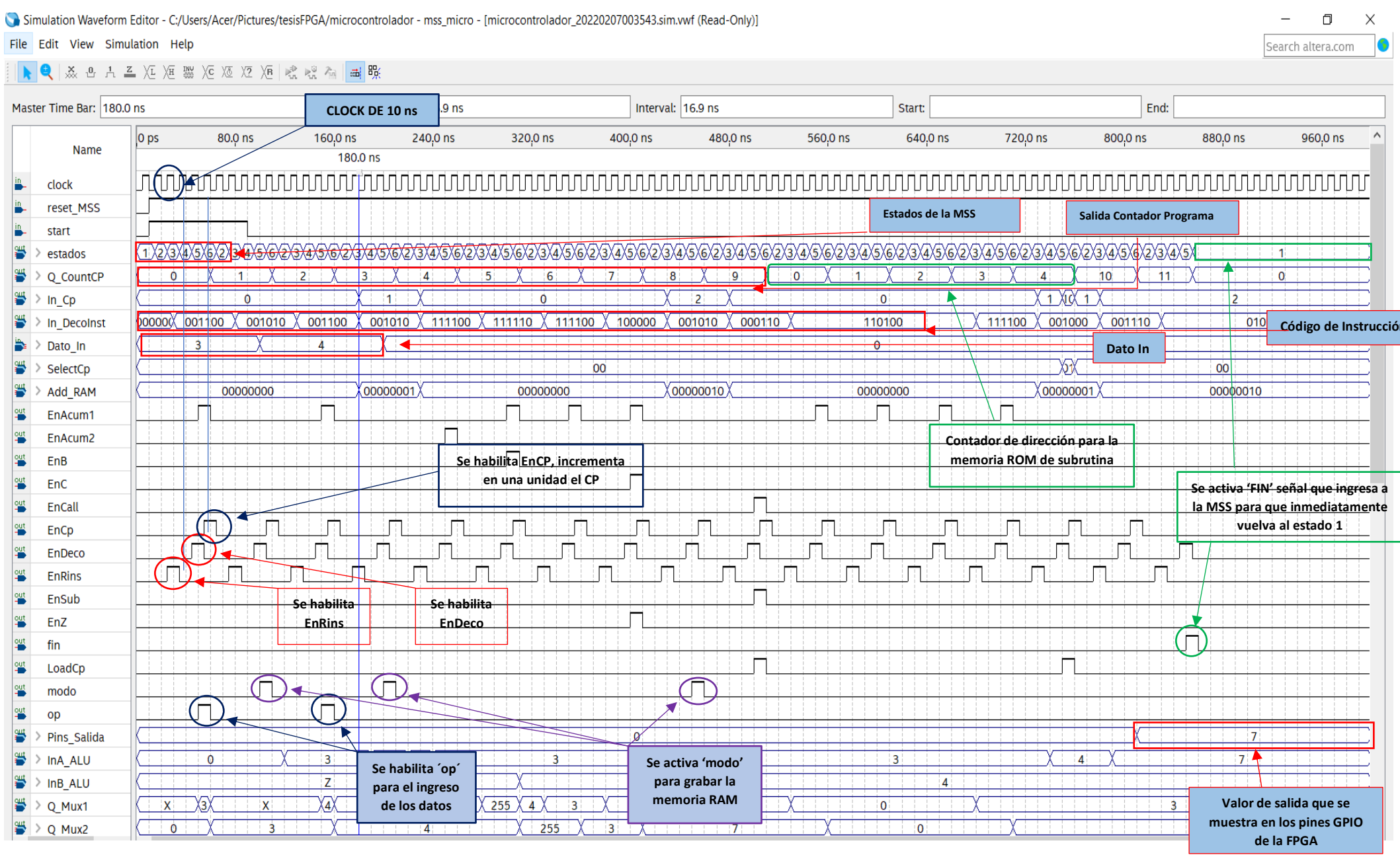


Figura 3.3: Waveform de la primera simulación

3.1.2. Segunda simulación dentro del Waveform Editor

Para esta simulación se probarán un grupo de instrucciones enlistadas en la Tabla 2.1 y Tabla 2.2. Adicional, se espera tener como resultado el número 226, proveniente de una multiplicación, división y algunas operaciones lógicas. En la figura 3.4 se observa el código correspondiente a la memoria ROM principal del microcontrolador. El programa contenido está formado por algunas instrucciones ya antes mencionadas en la simulación anterior y por otras nuevas como SHIFT LEFT, OR, SHIFT RIGHT, INC, NOT y SUBB.

```
10          address: in std_logic_vector(7 downto 0);
11          En, clock: in std_logic;
12          Q_data: out std_logic_vector(18 downto 0));
13  end rom_memory;
14
15  architecture behavioral of rom_memory is
16      type memoria_rom is array (0 to 11) of std_logic_vector (18 downto 0);
17      constant rom: memoria_rom:=(
18      --código de memoria rom: acarreo de entrada/código de instrucción/dirección/modo direccionamiento/fuente/ destino
19          -- /1 bit/-6 bits-/-------8 bits/2 bits/1 bit/1 bit--
20              '0' & "001100" & "00000000"&"00"&"00",--INPUT--0x00 dirección del programa--
21              '0' & "001010" & "00000000"&"01"&"00",--STORE--
22              '0' & "001100" & "00000000"&"00"&"00",--INPUT--
23              '0' & "001010" & "00000001"&"01"&"00",--STORE--
24              '0' & "111110" & "00000000"&"00"&"00",---MOVE
25              '0' & "111100" & "00000000"&"01"&"00",--LOAD--
26              '0' & "100000" & "00000000"&"01"&"00",---ADD--
27              '0' & "001010" & "00000010"&"01"&"00",---STORE
28              '0' & "100000" & "00000000"&"00"&"00",---ADD
29              '0' & "001110" & "00000010"&"00"&"00",---OUTPUT----------
30              '0' & "011000" & "01010000"&"00"&"11",---ACTIVE PWM -80% DUTYCYCLE
31              '0' & "010100" & "00000010"&"00"&"00"---FIN
```

Figura 3.4: Código para la memoria del programa de la segunda simulación

El programa inicia con reset_Mss y Start activados, dando paso al contador de programa a incrementar. En la figura 3.5 se observa que el primer set de instrucciones comienza con el ingreso del valor 8 en binario por medio del direccionamiento inmediato. La señal Q_CountCP comienza a incrementar para leer instrucción por instrucción dentro de la memoria ROM. Una vez ingresado dicho valor, éste se almacena en la memoria de datos y luego se realiza un desplazamiento a la izquierda, cuyo resultado da 16. Con este resultado parcial se realiza la operación OR con el dato ingresado por direccionamiento inmediato. Con este resultado de la operación OR se realiza un desplazamiento a la derecha. Por último, el valor actual es incrementado una unidad, después negado y finalmente restado con el inicial, dando como resultado $(226)_{10}$. En la señal INA_ALU, que corresponde al puerto de entrada A de la ALU, se puede observar todo el proceso matemático antes mencionado.

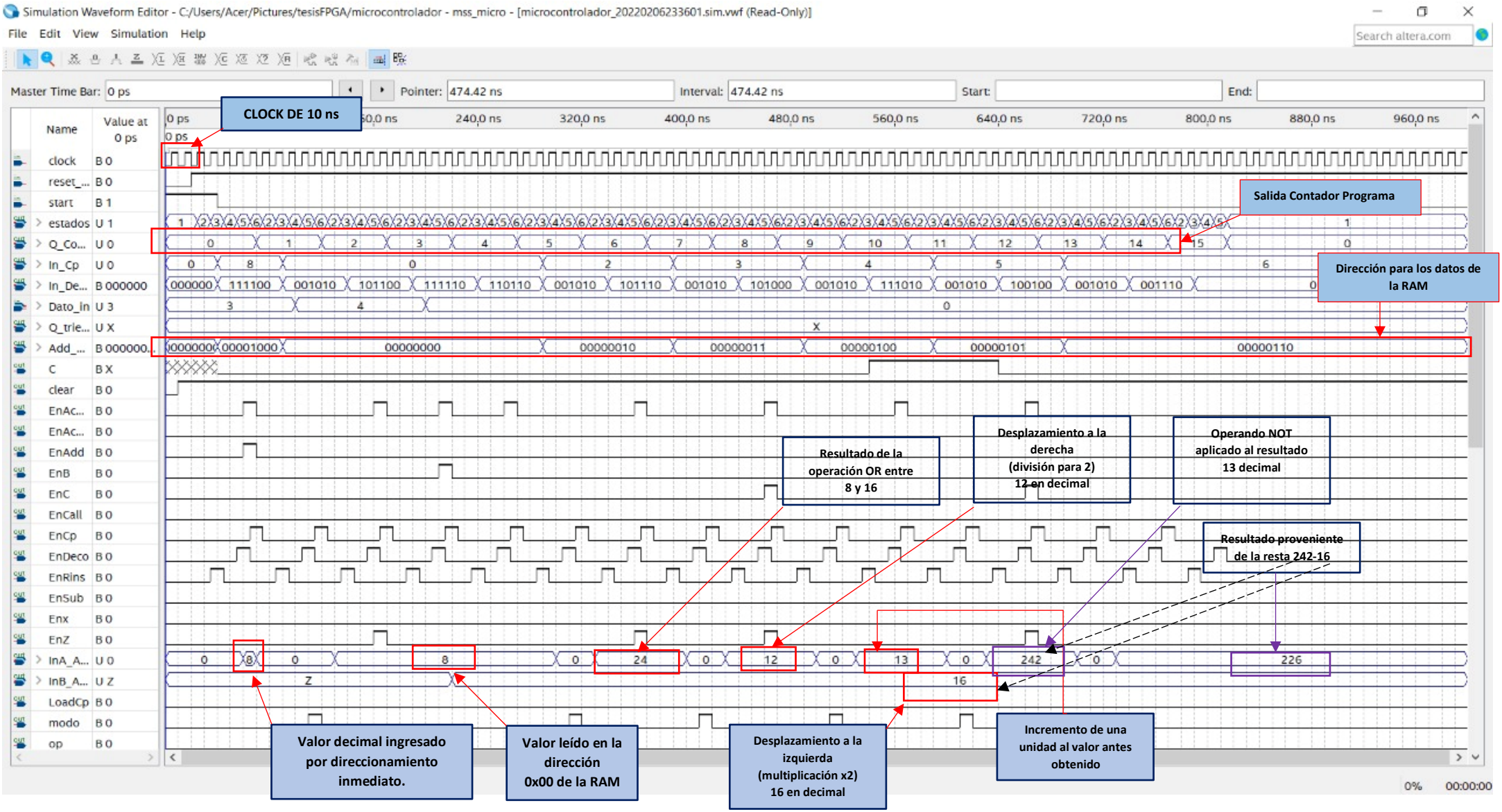


Figura 3.5: Waveform de la segunda simulación

3.2. Pruebas Experimentales

3.2.1. Uso de la memoria ROM en la primera prueba experimental

Como se había mencionado en la sección 2.12.3, para el desarrollo de las pruebas experimentales se hará uso de los archivos con extensión .mif, para facilitar el ingreso de la codificación en la memoria ROM.

En este caso, la figura 3.6 muestra el archivo que contiene los códigos para la generación de una señal PWM, denominándolo informacion.mif, restando como último paso ejecutar el archivo que posee un único bloque con la arquitectura en su interior, esta generación es explicada en secciones posteriores, para obtener los resultados del programa.

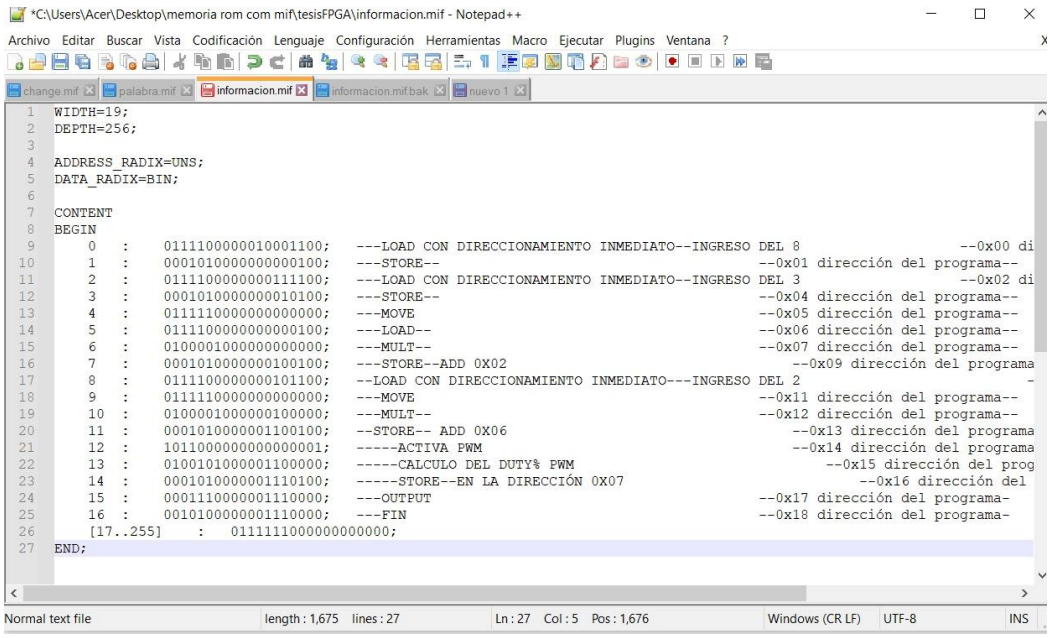


Figura 3.6: Archivo .mif para la primera prueba experimental

3.2.2. Prueba de la generación de la señal PWM

Uno de los bloques diseñados en el microcontrolador es el de PWM, que es capaz de generar una señal PWM con una frecuencia y ciclo de trabajo variable. El valor de la frecuencia del PWM es asignada de acuerdo a la salida que seleccione el bloque selector a partir de un bloque divisor de frecuencia que contiene 6 frecuencias distintas (5MHz, 500KHz, 50KHz, 5KHz, 500Hz), incluyendo la frecuencia de 50 MHz de operación del sistema, como se visualiza en la figura 3.7.

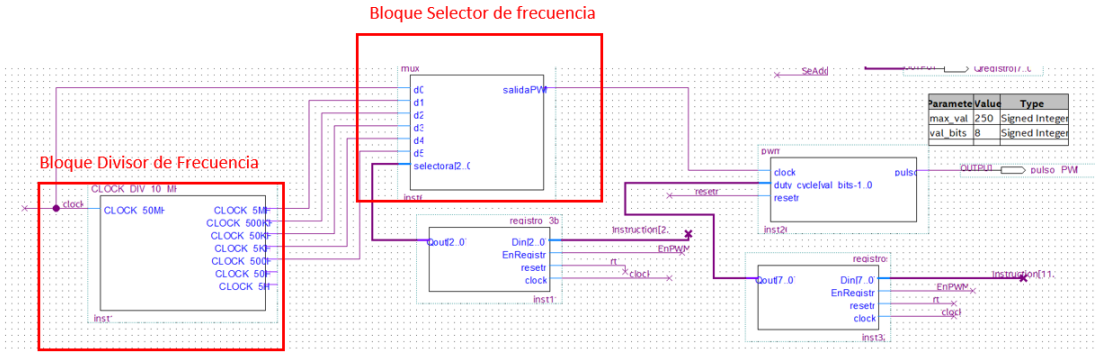


Figura 3.7: Diagrama de bloques del generador PWM

Mientras que la señal del ciclo de trabajo puede ser cambiada directamente desde el set del código de instrucciones, es decir de los 19 bits que corresponde al set de código de instrucciones, el bit 11 hasta el 4 corresponde al valor del duty cycle, siendo el mínimo 1 y el máximo 250. Para variar el duty cycle el código de la instrucción (bit 17 hasta bit 12) debe tener la siguiente combinación “011000”, lo que indica la activación del PWM dentro del decodificador de instrucción. A continuación, la tabla 3.1 muestra la combinación correspondiente a esta instrucción PWM.

Tabla 3.1: Configuración del set de instrucciones para la activación del bloque PWM

			Código para frecuencia PWM				
Carry In	Instruction	Valor del Duty Cycle	Addressing		Source	Desti.	
1: Dato por puerto de entrada 0: Dato por código	011000	Min: 00000001	N/A	0	0	0	200kHz
			N/A	0	0	1	20 kHz
			N/A	0	1	0	2 KHz
		Máx: 11111010	N/A	0	1	1	200 Hz
			N/A	1	0	0	20 Hz
			N/A	1	0	1	2 Hz

En la memoria ROM principal del microcontrolador, se tiene grabado un arreglo de set de instrucciones de 11 filas, en donde la fila 10 se tiene al set de instrucción correspondiente para la activación del PWM, concentrándose en dicha fila se puede observar que del bit 17 al bit 12 pertenece al código de instrucción denominado ‘Active PWM’ mientras que

del bit 11 al bit 4 corresponde al valor del duty cycle expresado en binario, en este caso es de $(80)_{10}=(01010000)_2$, y del bit 2 al bit 0 permite seleccionar la frecuencia, según la tabla anterior corresponde a una frecuencia PWM de 200 Hz, como lo muestra la figura 3.8.

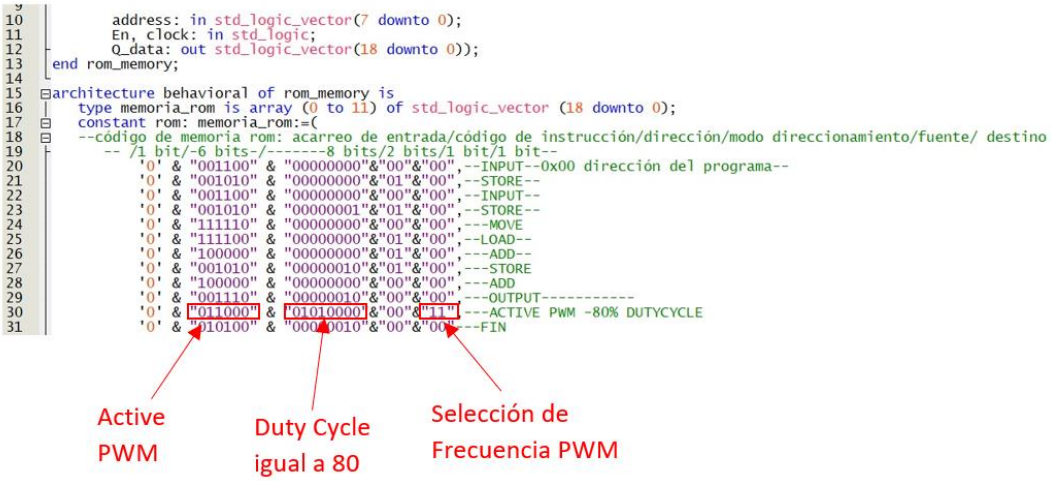


Figura 3.8: Código ingresado dentro de la memoria ROM principal del microcontrolador

Cabe indicar que el valor que se ingresa en el byte correspondiente al Duty Cycle no está dado en porcentaje, para obtener dicho valor en 'tanto por ciento' se hace uso de la siguiente ecuación:

$$\%Duty = \frac{valor\ duty\ decimal}{250} * 100 \qquad (1)$$

Lo que se esperaría, según la ecuación anterior, es un duty cycle de 32% en las simulaciones. Para la selección de la frecuencia PWM se debe tener en cuenta la relación entre la frecuencia de entrada del bloque PWM y el valor máximo del contador interno que posee, definiéndose de la siguiente forma:

$$frequency\ PWM = \frac{frequency\ in}{250} \qquad (2)$$

El valor de 250 corresponde a la salida máxima del contador del bloque interno PWM e indica la resolución que puede llegar a tener el Duty Cycle. En base a la codificación del set de instrucción anterior se espera tener una frecuencia PWM igual a:

$$frequency\ PWM = \frac{50kHz}{250} = 200\ Hz \qquad (3)$$

Una vez compilado y generado el archivo de bloques del microcontrolador denominado ‘bloques_micro.bdf’ se procede a crear el ‘Create Symbol Files for Current File’ el cual permite comprimir todos los bloques de la arquitectura en un solo bloque principal con sus respectivas entradas y salidas, en este caso particular se lo ha definido como ‘bloque_principal.bsf’ dentro de otro archivo mayor tipo bdf, llamado ‘archivomayor.bdf’ como se muestra a continuación en la figura 3.9.

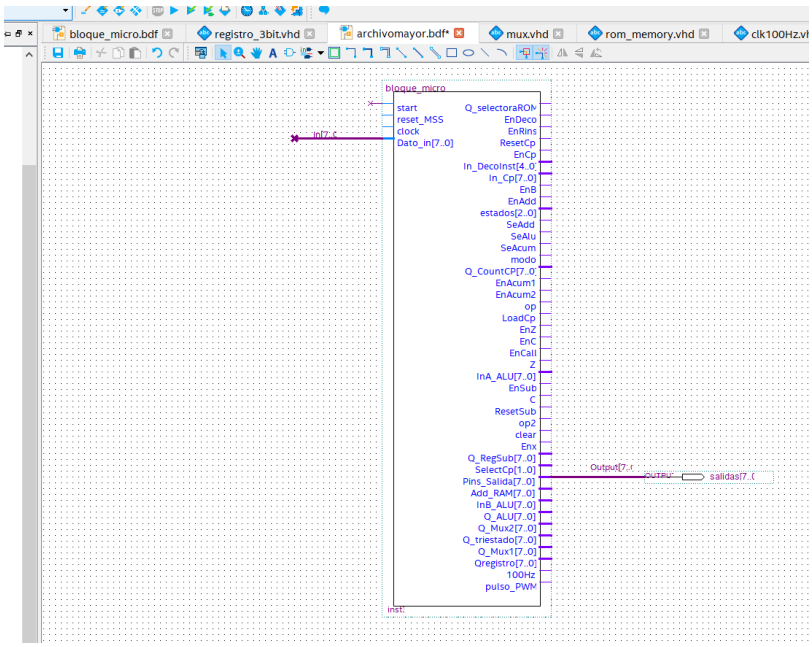


Figura 3.9: Generación del bloque total del microcontrolador diseñado dentro de otro archivo. bdf mayor

Por el momento en las señales de entrada nos interesa ‘start’, ‘reset_MSS’ y ‘clock’ y en la señal de salida ‘pulso_PWM’. Para la señal ‘start’ se ha agregado un bloque denominado ‘anti rebote’ debido a que se trata de una señal externa proveniente de un pulsador físico integrado en la tarjeta. La señal ‘clock’ y ‘reset_MSS’ proviene de un reloj interno y de un interruptor físico de la tarjeta respectivamente. La señal de salida ‘pulso_PWM’ se ha asignado a un pin externo de la tarjeta FPGA DE10-Nano. A continuación, la figura 3.10 muestra el esquemático final para la prueba de la generación

PWM del microcontrolador considerando un duty cycle del 32% a una frecuencia de 200 Hz.

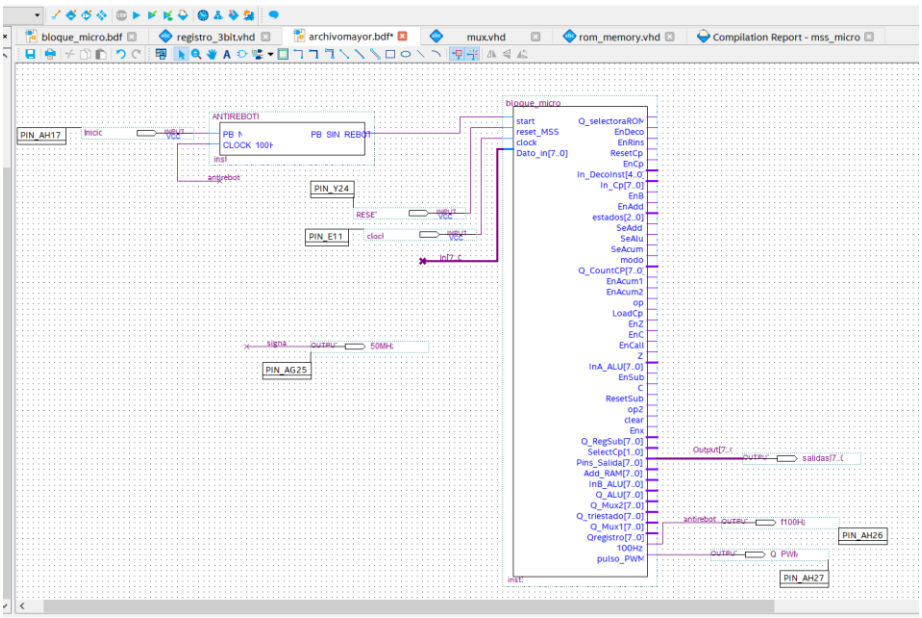


Figura 3.10: Asignación del bloque anti rebote y asignación de pines de entrada como salida del sistema

La asignación de los pines ha sido realizada por medio del 'Pin Planner'. Dentro de la mencionada ventana, se han asignado los pines de las señales de entrada y salida que se desean visualizar, en este caso las señales de entrada provienen de elementos discretos de la tarjeta FPGA DE10-Nano que corresponde al pin AH17, Y24 y E11 para el clock interno de 50 MHz. Para la señal de salida PWM se asignará el pin AH27 que pertenece al GPIO_1[9] de la tarjeta física. La figura 3.11 muestra la respectiva asignación dentro del Pin Planner.

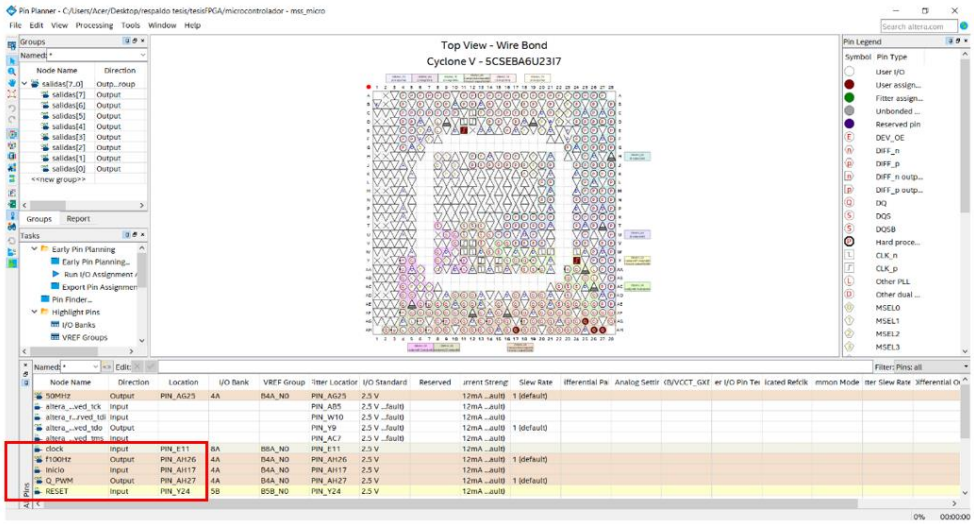


Figura 3.11: Asignación de pines de entrada y salida del sistema dentro de la ventana de Pin Planner de Quartus II

Una vez asignado los pines y compilado el ‘archivomayor.bdf’ se procede a realizar la programación de la tarjeta. Para ello se dirige a ‘Tools’, luego seleccionar ‘Programmer’ y a continuación se desplegará una venta como muestra la figura 3.7, donde debe seleccionar el dispositivo de la tarjeta FPGA DE10-Nano que es el 5CSEBA6U23 y cargar el archivo ‘.sof’. En este caso ‘mss_micro.sof’ luego se presiona ‘Start’ para iniciar el proceso de carga del programa. La figura 3.12 muestra que el progreso de carga ha sido 100% satisfactorio.

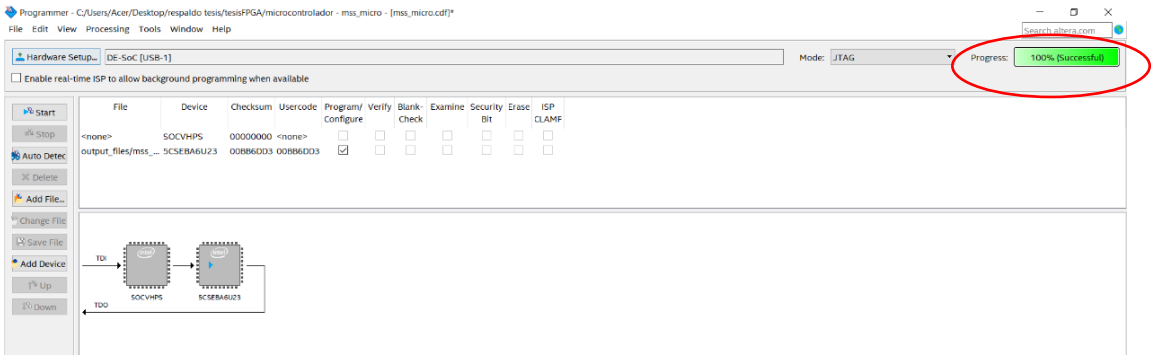


Figura 3.12: Programación del archivo dentro de la tarjeta FPGA DE10-Nano a través de la opción Programmer

3.2.3. Análisis de Resultados correspondiente a la generación de la señal PWM

Como se describió en los párrafos anteriores, las señales de entrada como lo son el ‘resetn’ y el ‘start’ provienen físicamente de la tarjeta, por lo tanto, hay que tener en cuenta la ubicación de dichos elementos. A continuación, en la figura 3.13 se muestra la ubicación física de las entradas al igual que los pines de salida para esta prueba. El PIN_AH27 ha sido asignado para la salida PWM, mientras el pin AH17 y el PIN_Y24 corresponden a ‘Start’ y ‘Reset MSS’ respectivamente. Además, se puede observar el led color naranja enmarcado que indica que la tarjeta FPGA DE10-Nano ha sido configurada y programada correctamente.

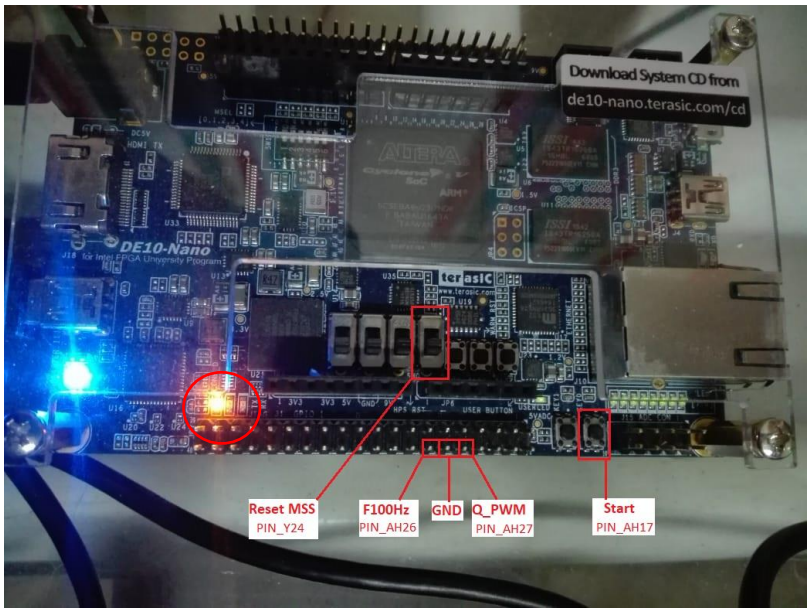


Figura 3.13: Ubicación física de los pines de entrada y de salida del diseño planteado

Para la prueba de la señal de salida PWM se hará uso de una sonda con un osciloscopio y un multímetro Fluke para la medición del ciclo de trabajo, tal como en la figura 3.14, y corroborar con los resultados teóricos enunciados en párrafos anteriores (%Duty cycle igual al 32% y frecuencia de la salida PWM igual a 200 Hz).

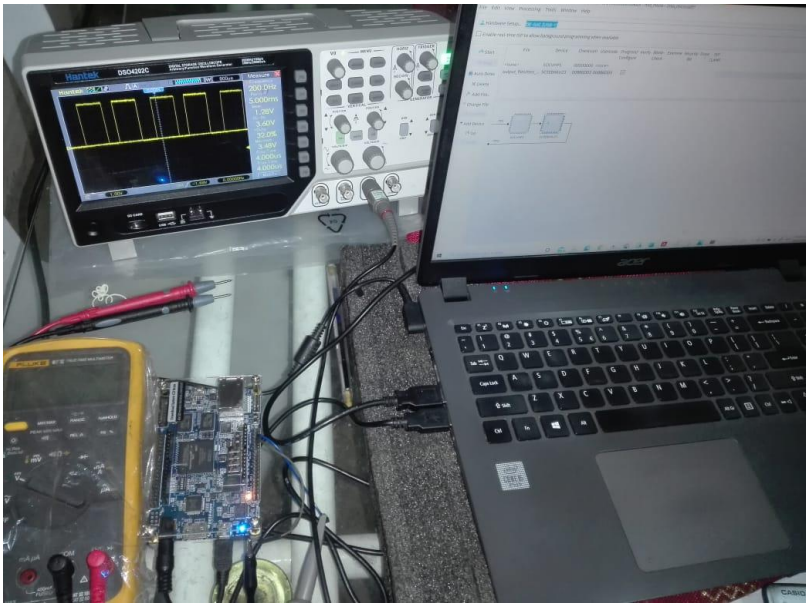


Figura 3.14: Equipos utilizados para la medición de los resultados

La figura 3.15 muestra la interfaz para PC del osciloscopio donde se visualiza la señal PWM captada del PIN_AH27, correspondiente al pin físico GPIO_1[9] de la tarjeta FPGA, que posee una frecuencia de 200.00 Hz, un ciclo de trabajo positivo del 32% y un ciclo de trabajo negativo del 68%.

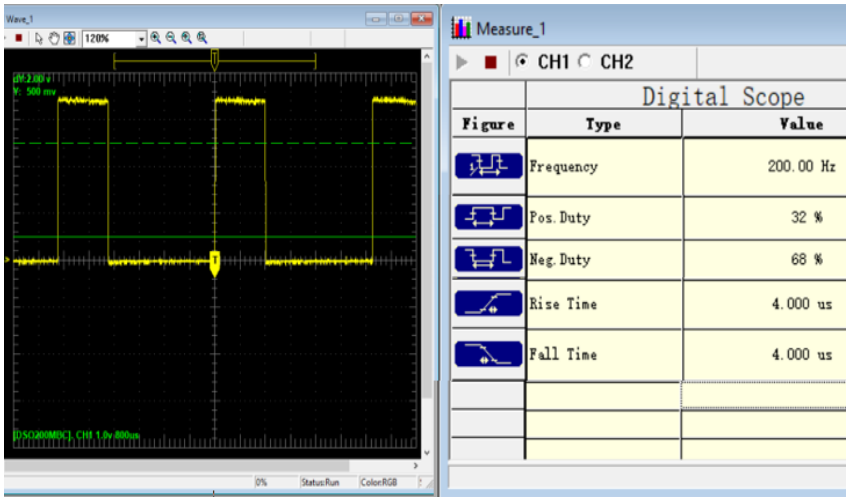


Figura 3.15: Captura de pantalla del osciloscopio donde se visualiza la frecuencia y duty cycle de la señal PWM captada

Respecto a la visualización en la pantalla física del osciloscopio, en la figura 3.16 se observa un ciclo de trabajo positivo igual a 32% y una frecuencia PWM igual a 200.00Hz según lo establecido en la sección anterior.

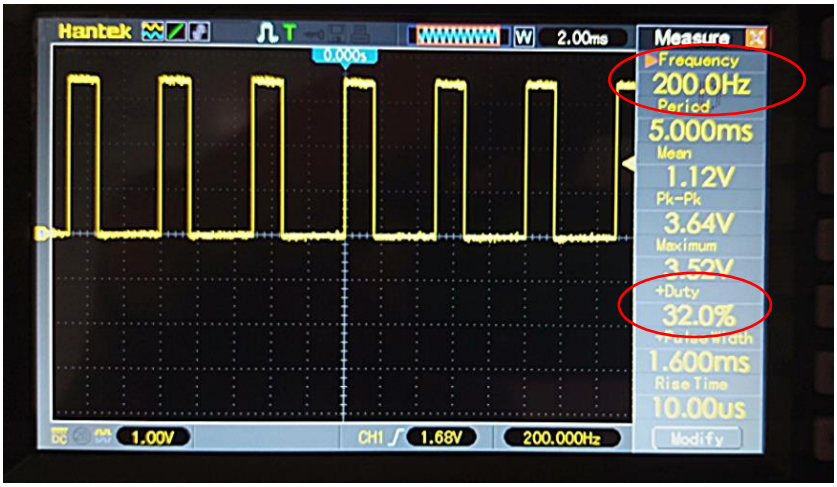


Figura 3.16: Pantalla del osciloscopio donde se visualiza la señal PWM generada del microcontrolador diseñado.

Para la lectura captada por el multímetro, se tiene un valor de ciclo de trabajo positivo igual al 32.0%, valor idéntico al obtenido en el osciloscopio y mostrado en la figura 3.17.

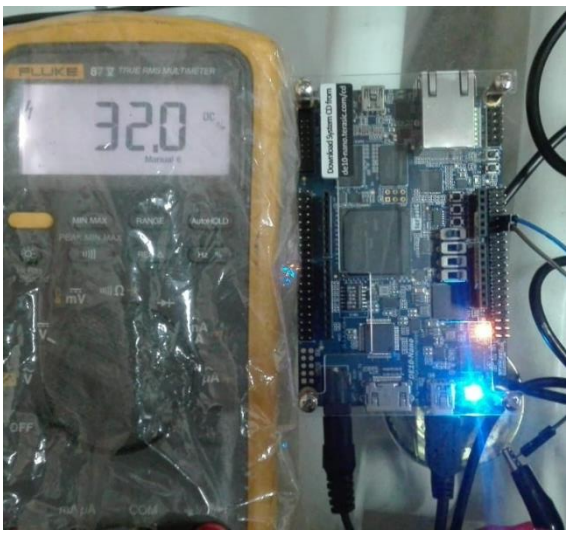


Figura 3.17: Lectura del Duty Cycle de la señal PWM generada por el microcontrolador a través del multímetro

Como se observa en la figura 3.18 se tiene una frecuencia de salida PWM igual a 200.0 Hz medida a partir de un multímetro, y que corresponde al mismo valor obtenido en la medición anterior.

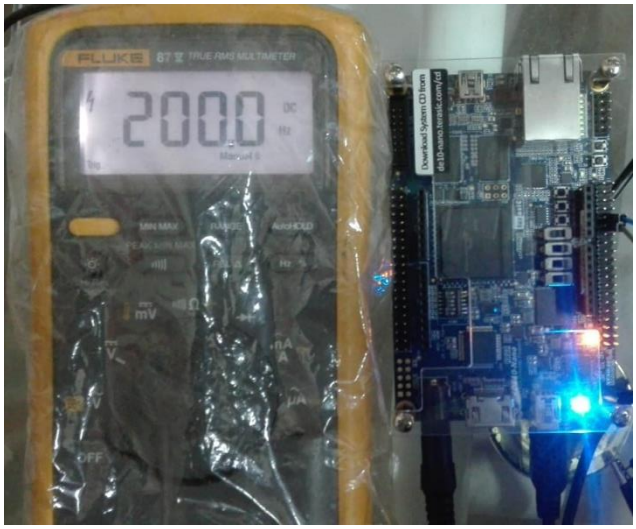
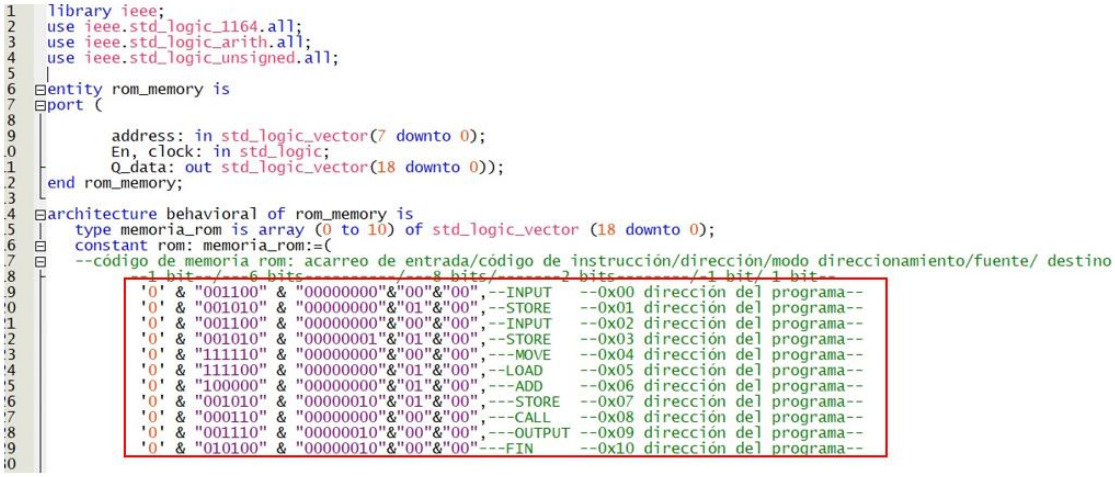


Figura 3.18: Lectura de la frecuencia de la señal PWM generada por el microcontrolador a través del multímetro

3.2.4. Prueba correspondiente a la validación del funcionamiento de los códigos de instrucciones

Para este caso se ha programado dentro de la ROM principal del microcontrolador un arreglo de set de instrucciones de 10 filas, en donde cada una de ellas posee un código de instrucción que corresponde a un operando, ya sea aritmético o no, los cuales son: input, store, load, move, add, call, output y fin. Sus respectivos códigos, mostrados en la figura 3.19, se pueden encontrar en el apartado 2.4. Reduciéndose a una suma binaria con un llamado de subrutina, la cual se encuentra en otra memoria ROM denominada ‘rom_subrutina.vhd’ que a su vez contiene la instrucción denominada ‘retorno subrutina’ que obliga a regresar a la lectura de la memoria ROM principal.



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity rom_memory is
port (

       address: in std_logic_vector(7 downto 0);
       En, clock: in std_logic;
       Q_data: out std_logic_vector(18 downto 0));
end rom_memory;

architecture behavioral of rom_memory is
   type memoria_rom is array (0 to 10) of std_logic_vector (18 downto 0);
   constant rom: memoria_rom:=(
   --código de memoria rom: acarreo de entrada/código de instrucción/dirección/modo direccionamiento/fuente/ destino
      --1 bit--/---6 bits----------/---8 bits/-------2 bits--------/-1 bit/ 1 bit--
        '0' & "001100" & "00000000"&"00"&"00",--INPUT   --0x00 dirección del programa--
        '0' & "001010" & "00000000"&"01"&"00",--STORE   --0x01 dirección del programa--
        '0' & "001100" & "00000000"&"00"&"00",--INPUT   --0x02 dirección del programa--
        '0' & "001010" & "00000001"&"01"&"00",--STORE   --0x03 dirección del programa--
        '0' & "111110" & "00000000"&"00"&"00",---MOVE   --0x04 dirección del programa--
        '0' & "111100" & "00000000"&"01"&"00",--LOAD    --0x05 dirección del programa--
        '0' & "100000" & "00000000"&"01"&"00",---ADD    --0x06 dirección del programa--
        '0' & "001010" & "00000010"&"01"&"00",---STORE  --0x07 dirección del programa--
        '0' & "000110" & "00000000"&"00"&"00",---CALL   --0x08 dirección del programa--
        '0' & "001110" & "00000010"&"00"&"00",---OUTPUT --0x09 dirección del programa--
        '0' & "010100" & "00000010"&"00"&"00"---FIN     --0x10 dirección del programa--
```

Figura 3.19: Código de set de instrucciones grabadas dentro de la memoria ROM principal del microcontrolador diseñado

Al igual que la aplicación anterior, una vez compilado y generado el archivo ‘bloques_micro.bdf’ se crea el ‘Create Symbol Files for Current File’ permitiendo comprimir cada uno de los bloques del microcontrolador en un bloque general con sus entradas y salidas asignadas anteriormente.

Posteriormente se procede a crear una nueva hoja de trabajo ‘.bdf’ donde se busca el archivo antes creado. En este caso corresponde al archivo ´bloque_principal.bsf’. Para esta simulación se concentrará por el momento en las señales de entrada que necesita el microcontrolador al igual que las señales de salida, las cuales se encuentran enmarcadas con un rectángulo rojo, ejemplificado en la figura 3.20. A estas señales se le asignará cada uno de los pines físico e internos que presenta la tarjeta FPGA DE10-Nano.

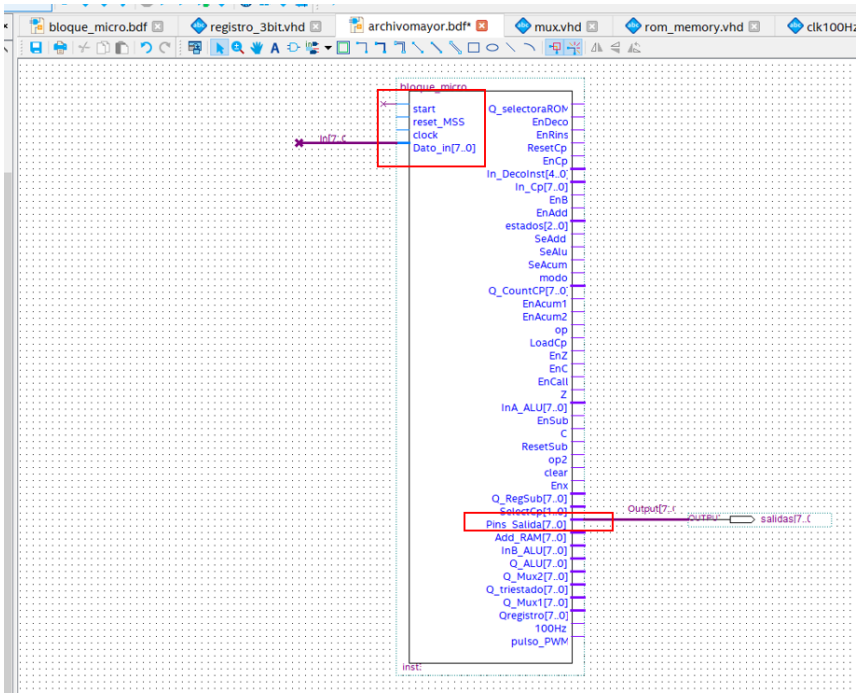


Figura 3.20: Bloque del archivo general del microcontrolador

3.2.5. Asignación dentro del Pin Planner

Para la asignación correcta de los pines hay que dirigirse al datasheet de la FPGA y tener en cuenta la ubicación del pin a configurar, en este caso se ha trabajado con los 8 primeros pines del GPIO_1 (JP7) de la DE-10 Nano para la salida de 8 bits de microcontrolador diseñado. En la figura 3.21 se ha seleccionado el pin V11 para el clock que es un clock de 50 MHz interno, el pin Y24 que corresponde a un interruptor en la tarjeta, y el Pin AH16 para el ‘start’ que es un push button. Mientras que, para el bus de datos de entrada, más adelante se la realizará dentro del bloque ‘Probes and Sources’.

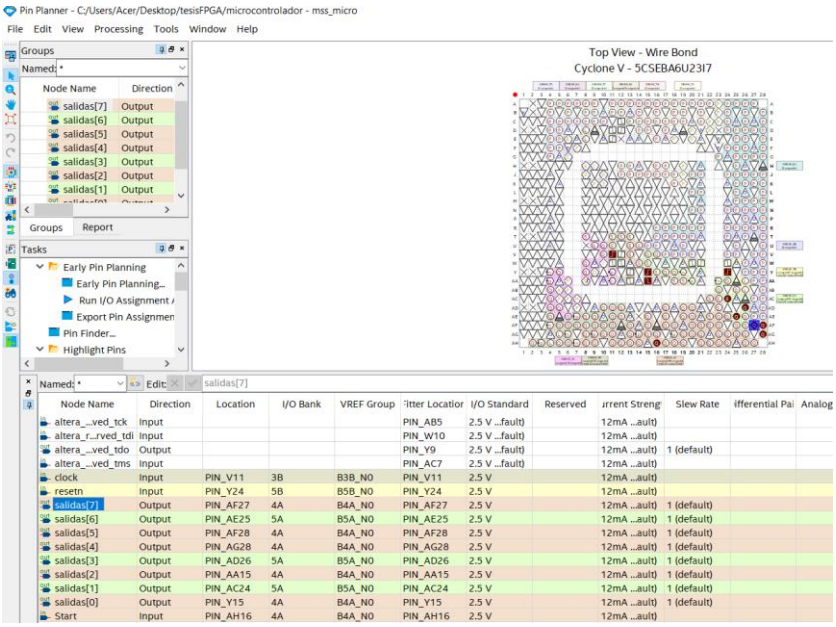


Figura 3.21: Configuración de la ventana Pin Planner para la asignación de cada uno de los pines físicos de la FPGA

En la figura 3.22 se muestra la asignación de pines físicos respecto a la asignación de pines internos del Pin Planner para la salida de 8 bits del microcontrolador diseñado.

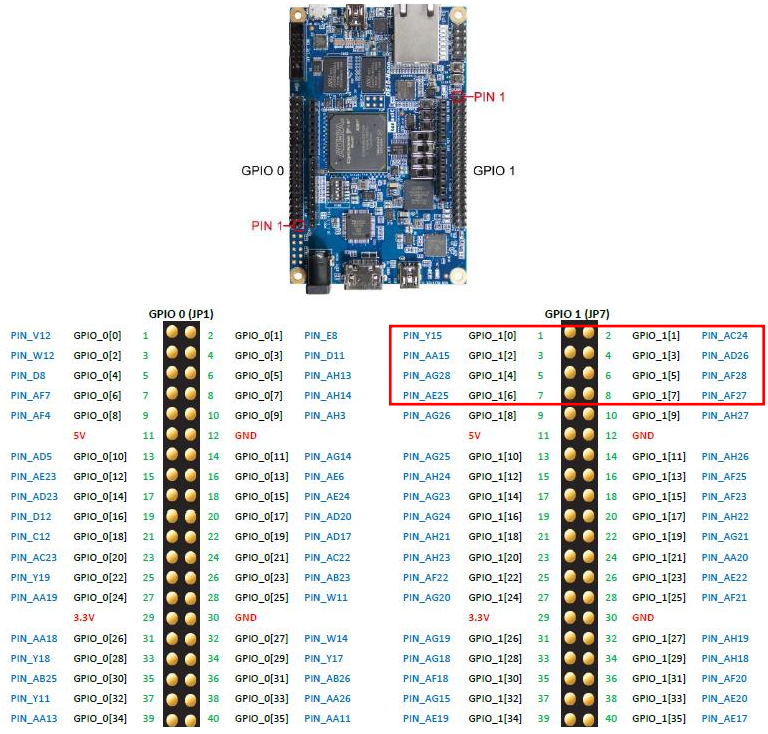


Figura 3.22: Correspondencia de pines en la FPGA DE10-Nano (Intel, 2016)

3.2.6. Creación del Intel FPGA In-System Sources

Para realizar el ingreso de los datos en el microcontrolador, es necesario crear un bloque de prueba y fuente, el cual permite leer y escribir datos según las entradas seleccionadas, en este caso las entradas principales del sistema corresponden a 'DataIn' y las salidas a 'Salidas [7...0]'. Adicionalmente, fue creado un bloque total del microcontrolador por medio de la opción 'Create Symbol File For Current File' el cual permite crear un archivo adicional. bdf de jerarquía superior para evitar realizar cambios en el diseño de la arquitectura del microcontrolador, lo cual muestra la figura 3.23.

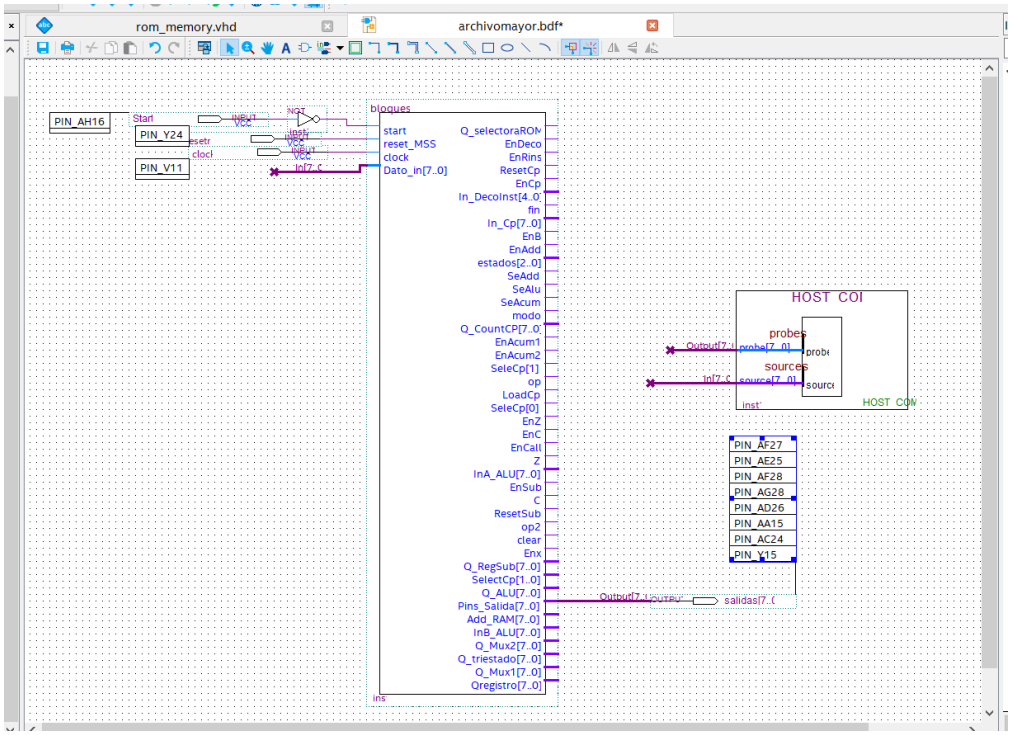


Figura 3.23: Generación del bloque Probes y Sources

3.2.7. Ejecución del In-System Sources and Probes Editor

A continuación, se puede observar la carga del archivo. sof que se ha generado y que se programa en la ventana In System Sources and Probes Editor, el cual permitirá leer y escribir datos en las señales seleccionadas del diseño. La figura 3.24 muestra el proceso de carga del diseño dentro de la tarjeta FPGA.

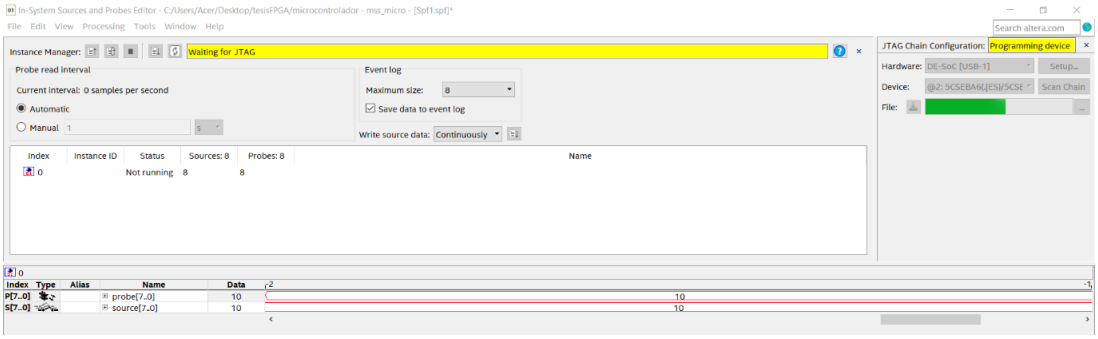


Figura 3.24: Ajuste de la ventana In-System Sources and Probes Editor

A diferencia de la simulación anterior, no se ejecuta la ventana 'Progammer', sin embargo se dirige a Tools, luego se despliega una ventana la cual se selecciona la opción In-System Sources and Probes Editor, se desplegará una ventana como en la gráfica superior indicada, la cual tiene que archivarse el archivo '.sof' para la programación del dispositivo, este archivo se lo busca dentro de la carpeta 'out files' con extensión '.sof'. Una vez agregado se selecciona el ícono 'Program Device' y el archivo se grabará inmediatamente.

3.2.8. Análisis de Resultados correspondiente a la verificación del funcionamiento de los códigos de instrucciones

En la figura 3.25 se muestra la asignación de pines físicos correspondientes al Pin Planner, en este caso la suma corresponde a 2 números binarios por lo que según el programa grabado dentro de la memoria ROM el resultado se almacenará dentro de la memoria RAM en la dirección 0x02. La salida del microcontrolador corresponde a una arreglo de 8 bits denominado salidas[7..0] donde cada uno de los bits se ubica según la ilustración mostrada a continuación.

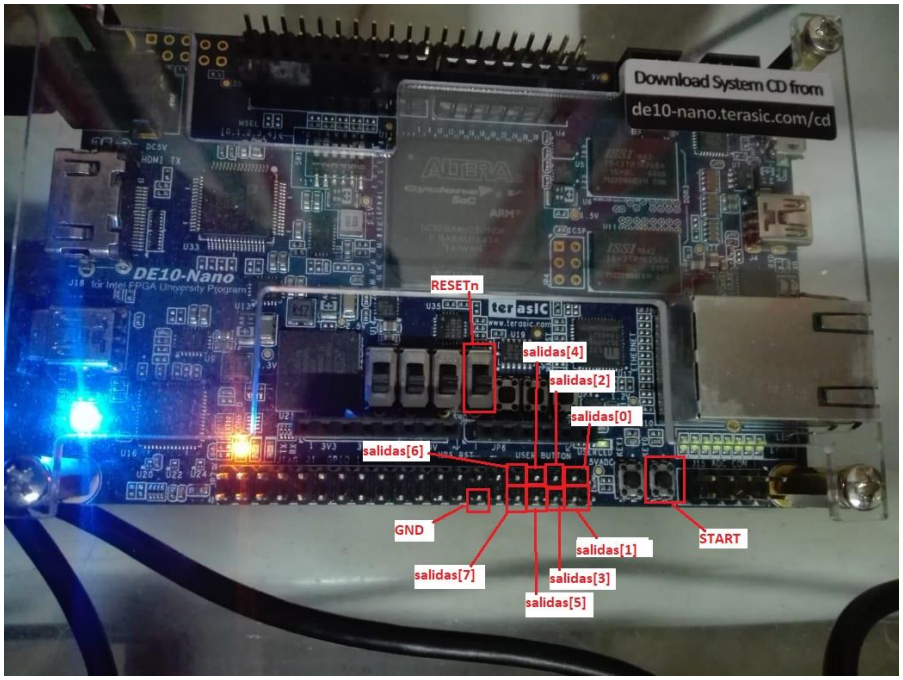


Figura 3.25: Asignación de los pines físicos de entrada y salida de la FPGA

Una vez iniciado el sistema con Resetn=1 y Start presionado, el programa se comienza a ejecutar por lo que el valor almacenado corresponde a 6, lo que equivale en binario a “00000110”. Los valores individuales ingresados por el ‘Sources Editor’ corresponde al $(00000001)_2$ y $(00000101)_2$. Posteriormente se procedió a medir cada uno de los pines correspondientes de salidas[7..0].

En la Tabla 3.2 se puede observar cada una de las salidas con su respectivo nivel lógico y voltaje Vdc medidos por el multímetro Fluke.

Tabla 3.2: Valor lógico de salida en cada bit

Salidas [7..0]									
Número\Bits		[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
$(6)_{10}$	Nivel Lógico	0	0	0	0	0	1	1	0
	Valor Vdc	0 V	0 V	0 V	0 V	0 V	3.339 V	3.340 V	0 V

La figura 3.26 muestra la lectura del pin GPI_1[2] de la tarjeta física FPGA que tiene un voltaje de 3.339 Vdc, valor que se ubica en el bit 2 del byte llamado ‘salidas’.

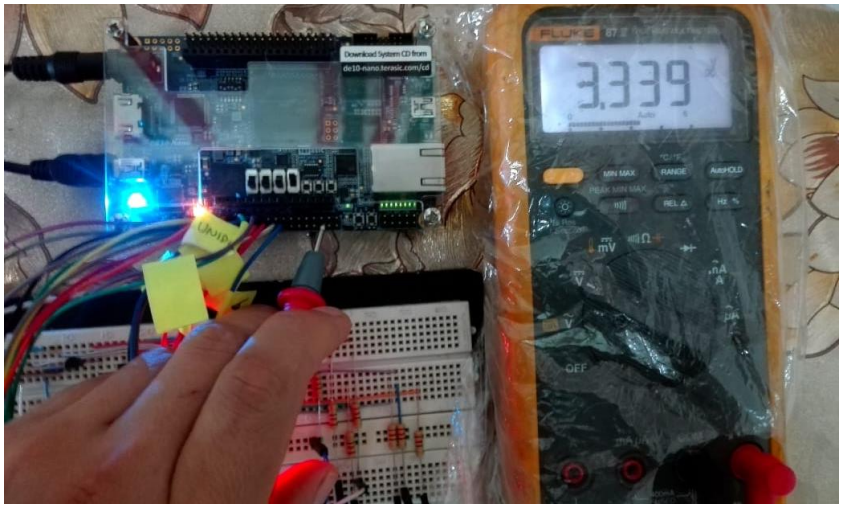


Figura 3.26: Medición del pin salida[2] que posee un valor de 3.33 V aproximadamente

Sin embargo, en la figura 3.27 se muestra la lectura del pin GPI_1[0] de la tarjeta física FPGA que tiene un voltaje de 0 Vdc, valor que se ubica en el bit 0 del byte llamado ‘salidas’.

Figura 3.27: Medición del pin salida[0] que posee un valor de 0 V aproximadamente

Los respectivos niveles de voltaje para el valor binario de 6, resultado de la prueba, se muestran en la figura 3.28 con su correcto nivel DC en los pines anteriormente configurados.

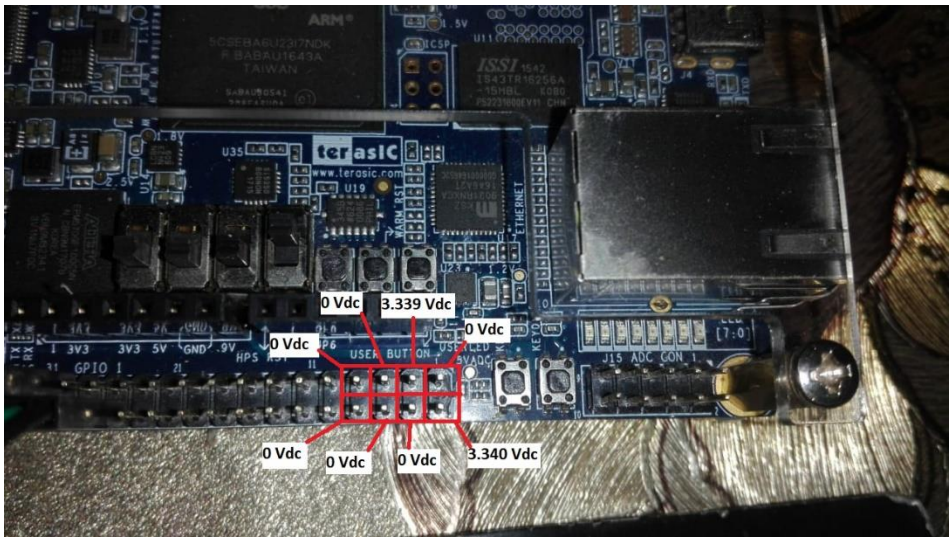


Figura 3.28: Niveles de voltaje en cada pin configurado

Para obtener una mejor visualización del resultado anterior, se creó un bloque decodificador que toma el valor de salida en binario y lo convierte para visualizarse a través de un display de 7 segmentos de 3 dígitos de ánodo común. La figura 3.29 muestra la implementación del bloque decodificador dentro del archivo .bdf de mayor jerarquía.

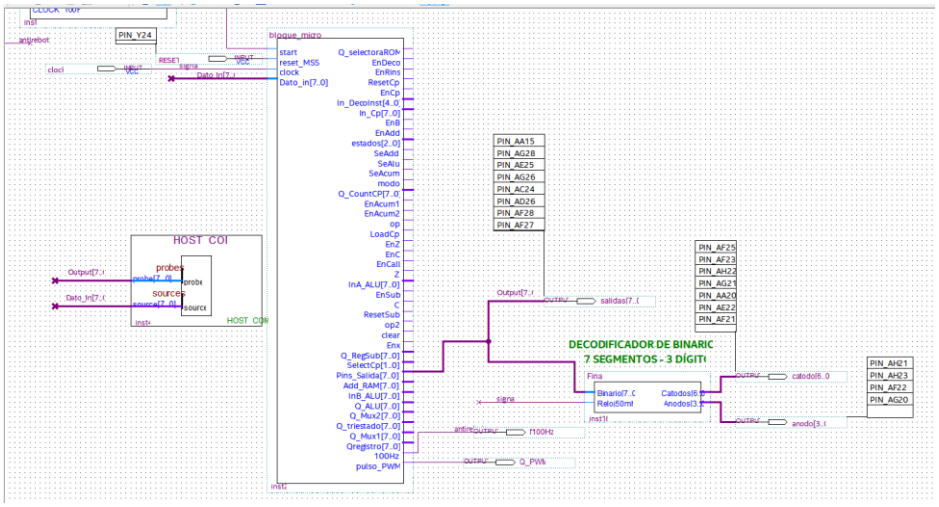


Figura 3.29: Bloque decodificador de binario a 7 segmentos

En la figura 3.30 se observa el valor decimal de 6 anteriormente mencionado, como resultado de la prueba que se refleja en un display de 7 segmentos.

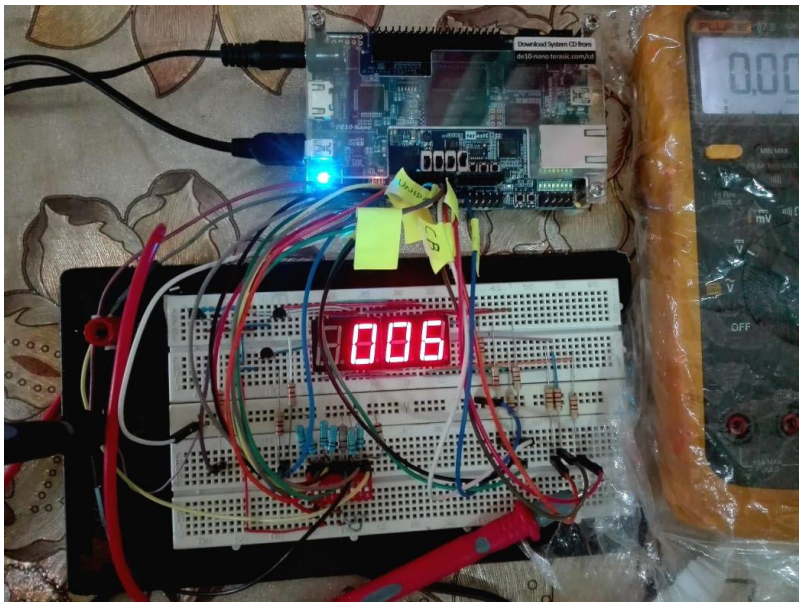


Figura 3.30: Visualización del resultado en display de 7 segmentos

De manera general se puede apreciar, en la figura 3.31, en el resumen del flujo un reporte proveniente de la compilación del archivo principal que indica el estatus, pines y memoria utilizada, entre otros. Existe un total de 265 registros usados, 14 pines usados de 314 lo que equivale al 4%, y 168 bits de bloque de memoria de 5,662.720 quedando por debajo del 1% de uso.

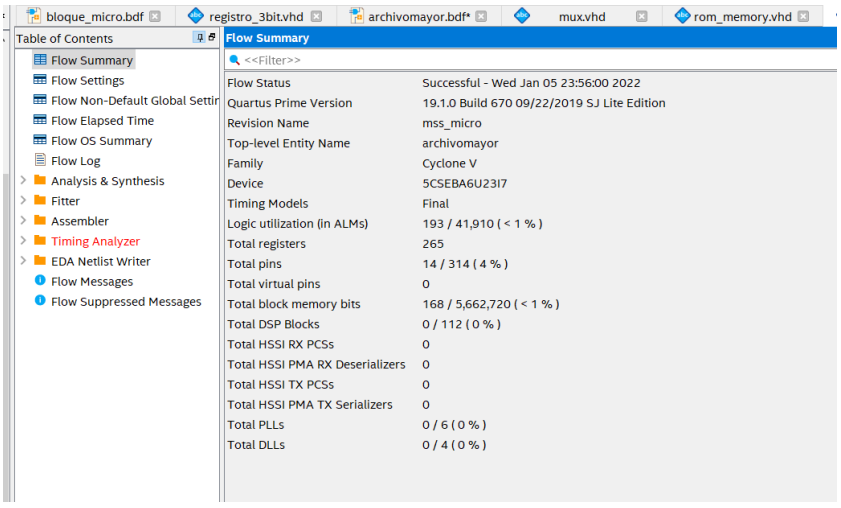


Figura 3.31: Reporte de compilación del archivo principal

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Se diseñó e implementó la arquitectura de un microcontrolador en lenguaje VHDL conformado por 33 bloques internos, de arquitectura tipo HARVARD, separando la memoria de instrucciones de la memoria de datos, por ende, le permite al estudiante o programador poder conocer más en detalle el funcionamiento de cada uno de los bloques internos que lo conforman.

- Mediante la creación de 2 estados vacíos dentro del diagrama ASM y el comando 'falling_edge' fue posible sincronizar cada uno de los bloques internos que conforman el microcontrolador para que activen sus respectivas salidas ante la respuesta de un flanco de bajada, a diferencia de la MSS principal que opera a flancos positivos.

- A través del código de operación de la instrucción 'Active PWM' se logró generar una salida física PWM por parte del microcontrolador que posee una frecuencia y ciclo de trabajo variable, con valores ingresados por medio del set de instrucciones, que le permite al usuario utilizar 5 frecuencias distintas para control de sistemas mediante PWM.

- Las operaciones que realiza el microcontrolador son solo de carácter digital debido a que la programación de cada uno de los bloques fue realizada en VHDL, por lo tanto, si se requiere operar con señales analógicas, es necesario el uso de un convertidor ADC ubicado previamente en el byte de entrada del microcontrolador diseñado.

- La memoria ROM principal y la memoria ROM de subrutina del diseño tiene una capacidad de 256X18, es decir está formado por un arreglo de 4608 bits, por lo tanto, permite ingresar solo hasta 255 set de instrucciones, una vez superado dicho número de dirección la lectura de la memoria inicia nuevamente.

- La memoria ROM de la arquitectura diseñada permite ser grabada de manera automática por medio de una PC, a través del archivo de extensión .mif, por lo que requiere ser grabado y guardado, necesitando solo compilar el archivo de mayor jerarquía del proyecto.

- Los resultados obtenidos en las pruebas experimentales fueron satisfactorios y concuerdan con las simulaciones dentro del Waveform, por lo que el prototipo queda totalmente operativo y listo para su uso en el Laboratorio de Sistemas Digitales Avanzado.

- Este trabajo se encuentra abierto para ser expandido hacia un mayor número de instrucciones, operaciones de nivel complejo y periféricos como puertos I2C o USB para una mejora del diseño propuesto.

- La profundidad de la memoria ROM principal y de subrutinas diseñada es de hasta 255 set de instrucciones, debido a que el contador de programa diseñado está conformado por una salida de 1 byte, formando en total 255 posibles combinaciones.

- Para el ingreso de datos por el puerto de entrada del microcontrolador se generó una instrucción de retardo por software, el cual permite ingresar un byte de datos antes de que se ejecute todo el programa debido a que el tiempo de ejecución de todo el programa está dado en nanosegundos mientras que el tiempo de ingreso de datos por el usuario está dado en segundos.

- En las simulaciones, dentro de la ventana de Simulation Waveform Editor se observa que cada instrucción tiene un tiempo de ejecución de 50 nanosegundos, es decir de 5 ciclos de reloj por instrucción. Por lo tanto, el tiempo de ejecución del primer programa para un set de 14 instrucciones grabadas en el microcontrolador es de 3.5 milisegundos, mientras que para el segundo programa que contiene un set de 16 instrucciones, el tiempo de ejecución es de 4 milisegundos. Mostrando que un mayor número de instrucciones conlleva un mayor tiempo de ejecución.

- El modelo del microcontrolador propuesto corresponde a un microcontrolador monociclo debido a que los recursos como la memoria de programa, de datos, registros y ALU se utilizan una sola vez por cada ciclo de reloj, lo que hace que las instrucciones se procesen de manera lenta a diferencia de un microprocesador multiciclo, donde tiene un menor tiempo de ejecución y mayor complejidad de diseño.

4.2. RECOMENDACIONES

- Utilizar otra familia de tarjeta Cyclone V FGPA, pero con una mayor frecuencia de 50 MHz debido a que las acciones de cada bloque se ejecutarían en menos de 20 us, haciendo que el microcontrolador tenga una mayor respuesta y tiempo de ejecución.

- Antes de realizar la Simulation Waveform Editor, del bloque total del microcontrolador, realizar la ejecución particular de cada uno de los bloques que lo componen para evitar errores en la compilación y simulación del mismo, lo que garantiza un mejor entendimiento para el programador y el usuario que profundice en este microcontrolador.

- Tener en cuenta la dimensión de cada uno de los buses de datos de los bloques al momento de realizar la conexión interna entre estos y evitar errores de conexión de la arquitectura.

- Si se desea tener una mayor capacidad de memoria de programa, es posible aumentar la cantidad de 8 a 16 bits en la salida del contador de programa, lo que permitirá almacenar 65535 set de instrucciones.

- Al final de cada código dentro de la memoria del programa, terminar con la instrucción fin, debido a que el contador de programa se finaliza y evita arrojar instrucciones que no se encuentran programadas dentro del memoria ROM.

- Al momento de realizar la programación dentro de la ventana del PIN PLANNER, tener a disposición el datasheet de la tarjeta FPGA y la diferencia entre la asignación del pin físico y el propio pin planner, para evitar errores de compilación.

REFERENCIAS

Baba, A. (2019). *Computer Architecture Von-Neumann vs Harvard*. *February*.

Balid, W., & Abdulwahed, M. (2013). A novel FPGA educational paradigm using the next generation programming languages case of an embedded FPGA system course. *IEEE Global Engineering Education Conference, EDUCON*, 23–31. https://doi.org/10.1109/EduCon.2013.6530082

Brown, S., & Vranesic, Z. (2006). Fundamentos de lógica digital con diseño en VHDL. En *Departamento de Ingeniería Eléctrica y Computación University of Toronto* (Segunda). McGraw Hill.

Cadence. (2021). *Hardware Description Languages: VHDL vs Verilog, and Their Functional Uses*. PCB Design & Analysis. https://resources.pcb.cadence.com/blog/2020-hardware-description-languages-vhdl-vs-verilog-and-their-functional-uses

Charte, F., Espinilla, M., Rivera, A., & Pulgar, F. (2017). Uso de dispositivos FPGA como apoyo a la ensenanza de asignaturas de arquitectura de computadores. *Enseñanza y Aprendizaje de Ingeniería de Computadores*, 37–52. https://doi.org/10.30827/digibug.47371

Escuela Universitaria Politécnica Valladolid (Departamento de Informática). (2016). *Modos de Direccionamiento* (pp. 1–12).

Festo. (2021). *Introducción a conceptos básicos (Automatización de Procesos)*. https://www.festo.com/mx/es/e/educacion/sistemas-de-aprendizaje/automatizacion-de-procesos/introduccion-y-conceptos-basicos-id_33984/

Huang, H., Xia, J., & Boumaiza, S. (2019). Parallel-Processing-Based Digital Predistortion Architecture and FPGA Implementation for Wide-band 5G Transmitters. En *2019 IEEE MTT-S International Microwave Conference on Hardware and Systems for 5G and Beyond, IMC-5G 2019*. https://doi.org/10.1109/IMC-5G47857.2019.9160360

Intel. (2016). *DE10-Nano User Manual*.

Intel. (2021). *Quartus*. FPGA Design Tools and Software. https://www.intel.la/content/www/xl/es/software/programmable/quartus-prime/overview.html

Jing Yu, H., Li-Li, L., Yan Chao, Z., Wen Tao, Y., & Jian Hong, Y. (2013). Multiply-accumulator using modified booth encoders designed for application in 16-bit RISC processor. *Proceedings - 2013 2nd International Symposium on Instrumentation and Measurement, Sensor Network and Automation, IMSNA 2013*, 416–419. https://doi.org/10.1109/IMSNA.2013.6743304

Lutkevich, B. (2019). *Microcontroller (MCU)*. TechTarget. https://internetofthingsagenda.techtarget.com/definition/microcontroller

Mendías, J. (2018). *Estructura y Tecnología de Computadores. Tema 4. Modos de direccionamiento y tipos de datos*.

Micron. (2020). *What is Computer Memory (RAM) and What Does It Do?* Crucial. https://www.crucial.com/articles/about-memory/support-what-does-computer-memory-do

Micron. (2021). *What is computer Rom*. Criucial. https://www.crucial.com/articles/about-memory/what-is-the-difference-between-ram-and-rom

Ríos, S. (2020). *FUNDAMENTOS DE MICROPROCESADORES Y SISTEMAS EMBEBIDOS [Diapositivas de PowerPoint], Facultad de Ingeniería en Electricidad y Computación, Escuela Superior Politécnica del Litoral (ESPOL)*.

Sáenz Rodríguez, W., Rivera Sánchez, F., & Martínez Santa, F. (2018). Dual-accumulator softcore 8-bit microprocessor designed to be used on FPGAs. *Tecnura*, *22*(56), 40–50.

Siemens. (2021). *Automation and Optimization: The Importance of the Health Check*. Siemens Healthinners. https://www.siemens-healthineers.com/ec/laboratory-automation/importance-of-healthcheck

Xilinx. (2021). *What is an FPGA?* Xilinx. https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

ANEXOS

Anexo 1: Código VHDL de la MSS

```
library ieee;
use ieee.std_logic_1164.all;

entity mss_micro  is
port (start, fin, resetn, clock, tempo: in std_logic;
            EnCp, ResetCp, EnRinst, EnDeco: out std_logic;
            estados: out std_logic_vector(2 downto 0));
end mss_micro;

architecture funcionamiento of mss_micro is
type estado is (T1, T2, T3, T4, T5, T6);
signal y: estado;

begin
process(clock, resetn, start, fin, tempo)
            begin
                  if resetn='0' then y<=T1;
                  elsif (rising_edge (clock)) then
                        case y is
                              When T1=> if start='1' then y<= T2;
                                                else y<= T1; end if;
                              when T2=> y<=T3;
                              when T3=> y<=T4;
                              when T4=> y<=T5;
                              when T5=> if tempo='1'then y<=T5;
                                                elsif (tempo='0' and fin='1')
then y<=T1;
                                                else y<=T6;
                                                end if;

                              when T6=> y<=T2;
                        end case;
                  end if;
end process;

process(y, start, tempo, fin)
begin
EnCp<='0'; ResetCp<='1'; EnRinst<='0'; EnDeco<='0'; estados<="000";
      case y is
            when T1 => estados<="001";ResetCp<='0';
            when T2 => estados<="010";
            when T3 => estados<="011";EnRinst<='1';
            when T4 => estados<="100";
            when T5 => estados<="101"; EnDeco<='1';
            when T6 => estados<="110"; if (tempo='0' and fin='0') then
EnCp<='1'; end if;
      end case;
end process;
end funcionamiento;
```

Anexo 2: Código VHDL de la memoria RAM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity RAM is
port(
            address: in std_logic_vector(7 downto 0);
            Data: in std_logic_vector(7 downto 0);
            modo, clock: in std_logic;
            Q: out std_logic_vector(7 downto 0));
end RAM;

architecture comp of RAM is
type ram_type is array (20 downto 0) of std_logic_vector (7 downto 0);
signal memram: ram_type;
begin
process(clock)
begin
      if (falling_edge(clock)) then
            if modo='1' then
                  memram(conv_integer(address)) <= Data;
            else
                  Q <= memram(conv_integer(address));
            end if;
      end if;
end process;
end comp;
```

Anexo 3: Código VHDL de la memoria ROM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity rom_memory is
port (

             address: in std_logic_vector(7 downto 0);
             En, clock: in std_logic;
             Q_data: out std_logic_vector(17 downto 0));
end rom_memory;

architecture behavioral of rom_memory is
      type memoria_rom is array (0 to 11) of std_logic_vector (17 downto 0);
      constant rom: memoria_rom:=(
      --código de memoria rom: acarreo de entrada/código de
instrucción/dirección/modo direccionamiento/fuente/ destino
                                                      --1 bit--/---5
bits----------/---8 bits/-------2 bits--------/-1 bit/ 1 bit--
                  '0' & "01111" & "00001000"&"00"&"00",--DELAY-- 8 SEGUNDOS--
0x00
                  '0' & "00110" & "00000000"&"00"&"00",--INPUT--0x00
dirección del programa--0x01
                  '0' & "00101" & "00000000"&"01"&"00",--STORE--0x02
```

```
                    '0' & "01111" & "00001000"&"00"&"00",--DELAY-- 8 SEGUNDOS
                    '0' & "00110" & "00000000"&"00"&"00",--INPUT--
                    '0' & "00101" & "00000001"&"01"&"00",--STORE--
                    '0' & "11111" & "00000000"&"00"&"00",---MOVE
                    '0' & "11110" & "00000000"&"01"&"00",--LOAD--
                    '0' & "10000" & "00000000"&"01"&"00",---ADD--
                    '0' & "00101" & "00000010"&"01"&"00",---STORE
                    '0' & "00111" & "00000010"&"00"&"00",---OUTPUT----
                    '0' & "01010" & "00000010"&"00"&"00"---FIN--0x11
                  --- '0' & "01111" & "01010000"&"00"&"11",
                  --x"0A",--0x01 dirección del programa
                  --x"8B",--0x02   '''
                  --x"05",--0x03    '''
                  --x"B7",--0x04    '''
                  --x"0B",--0x05    '''
                  --x"8B",--0x06    '''
                  --x"12",--0x07    '''
                  --x"B7",--0x08    '''
                  --x"0C",--0x09    '''
                  --x"AA",--0x0A   '''
                  --x"00" --0x0B   '''
            );
      begin
            process(clock)
                  begin
                        if falling_edge(clock) then --clock event?
                              if (en='1') then
                                    --output_interno <=
ROM(conv_integer(unsigned(address)));
                                    --Q_data<= output_interno(16 downto 0);

      Q_data<=rom(conv_integer(unsigned(address)));
                                 --else output_interno <= (output_interno'range
=> 'Z'); -- Todos los bits de Data_out se ponen a 'Z'
                                          --data_out<= output_interno(7 downto
0);
                                    end if;
                              end if;
            end process;
end behavioral;
```

Anexo 4: Código VHDL de la nueva ROM con lectura de archivo .mif

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
--use ieee.numeric_std.all;

entity rom_memory is
port (

            address: in std_logic_vector(7 downto 0);
            En, clock: in std_logic;
            Q_data: out std_logic_vector(18 downto 0));
end rom_memory;

architecture behavioral of rom_memory is
```

```
	type memoria_rom is array (255 downto 0) of std_logic_vector (18 downto
0);
	signal rom: memoria_rom;
	attribute ram_init_file: string;
	attribute ram_init_file of rom: signal is "informacion.mif";

	begin
		process(clock, address, En)
			begin
				if falling_edge(clock) then --clock event?
					if (en='1') then
					Q_data<=rom(conv_integer(unsigned(address)));
						end if;
					end if;
		end process;
end behavioral;
```

Anexo 5: Código VHDL del registro de saltos y de subrutinas

```
library ieee;
use ieee.std_logic_1164.all;

entity registros_salto is
port (
		Din: in std_logic_vector(7 downto 0);
		EnRegistro, resetn, clock: in std_logic;
		Qout: out std_logic_vector(7 downto 0));
end registros_salto;

architecture desarrollo of registros_salto is
signal temp: std_logic_vector(7 downto 0);
begin
process(clock,EnRegistro)
	begin
	if resetn='0' then temp<="00000000";
	elsif (falling_edge(clock)) then
		if EnRegistro='1' then temp<=Din;
			else temp<=temp;
		end if;
	end  if;
end process;
Qout<=temp;
end desarrollo;
```

Anexo 6: Código VHDL del decodificador de instrucciones

```
library ieee;
use ieee.std_logic_1164.all;

entity deco_inst is
port(
		Din: in std_logic_vector (18 downto 0);
		flagZ, flagC: in std_logic;
		EnDeco, clock, igual_tempo: in std_logic;
		fin,EnB,EnAdd, SeAdd, SeAlu, SAcum, modo, EnPortin, EnPortout,
EnAcum1, EnAcum2, op, S, LoadCp, EnZ, EnC, EnCall,
```

```
            EnSub,ResetSub, op2, clear,Enx,EnA: out std_logic;
            SelectoraCp: out std_logic_vector (1 downto 0);
            EnPWM,Tempo, Resetn_T,SelPWM: out std_logic);
end deco_inst;

architecture behavioral of deco_inst is
signal Instruccion: std_logic_vector (5 downto 0);
signal modo_direcc: std_logic_vector(1 downto 0);
begin
Instruccion<=Din(17 downto 12);
modo_direcc<=Din(3 downto 2);
      process(EnDeco, clock,igual_tempo, Din, Instruccion, modo_direcc)
      begin
            if (falling_edge(clock)) then

                  if EnDeco='1' then

            case modo_direcc is
                  when "11"=> EnAdd<='1';
            ----direccionamiento inmediato
                  when "01"=>
-----direccionamiento directo
                  when "10"=> SeAdd<='1'; Enx<='1';                        ----
direccionamiento indirecto  a memoria.
                  when others=>
            end case;
            if Din(17)='1' then if Din(0)='1' then EnAcum2<='1';          -----
DESTINO
                                                      else EnAcum1<='1';
                                                      end if;
                                                      end if;
            if Din(17 downto 12) /= "011000" then --- si no son iguales =>
entras
              if Din(1)='1' then SAcum<='1';                 ------FUENTE
                                                    else SAcum<='0';
                                                    end if;
                                                    end if;

            case Instruccion is
            ---ALU INSTRUCTIONS---
                  when "100000"=> EnZ<='1';EnC<='1';
------ADD
                  when "100010"=> EnZ<='1';EnC<='1';
      ----ADDC
               when "100100"=> EnZ<='1';EnC<='1';                              --
---SUB
               when "100110"=> EnZ<='1';EnC<='1';                    ---SUBC
                  when "101000"=> EnZ<='1';EnC<='1';                   ----INC
                  when "101010"=> EnZ<='1';EnC<='1';                   ----DEC
                  when "101100"=> EnZ<='1';
----SHL
                  when "101110"=> EnZ<='1';
---SHR
                  when "110000"=> EnZ<='1';EnC<='1';                    ---ROL
with carry
                  when "110010"=> EnZ<='1';EnC<='1';
      ----ROR with carry
                  when "110100"=>
                              ----AND
                  when "110110"=>
            ----OR
```

```
                when "111000"=>
                   ----XOR
                when "111010"=>
                      ----NOT
                when "111100"=>                                          -
---LOAD
                when "111110"=>   EnB<='1';
            ----MOVE
                when "100001"=> EnZ<='1';EnC<='1';                 -----
MULTIPLICACION
                when "100011"=> EnZ<='1';EnC<='1';                  ----
DIVISION
                when "100101"=> EnZ<='1';EnC<='1';                 -----%
duty cycle
            ---NON ALU INSTRUCTIONS-----
                when "000000"=> if flagZ<='1' then LoadCp<='1';      -----
JIFZ
                                                                        else
LoadCp<='0';

SelectoraCp<="00";
                                                                        end
if;

                when "000010"=>   if flagC<='1' then LoadCp<='1';--1       -
----JIFC
                                                                        else
LoadCp<='0';--0

SelectoraCp<="00";
                                                                        end
if;
                when "000100"=> LoadCp<='1';                            -----
-------------   -----JUMP
                when "000110"=> Encall<='1';LoadCp<='1';
EnSub<='1';SelectoraCp<="00";    -----CALL
                when "001000"=> ResetSub<='0';LoadCp<='1';SelectoraCp<="01";
                                  -----RETURN
                when "001010"=> modo<='1';
                      -----STORE
                when "001100"=> SeAlu<='1';op<='1';
                -----INPUT
                                              if Din(0)='1' then
EnAcum2<='1';
                                                                        else
EnAcum1<='1';
                                                                        end
if;
                when "001110"=> op2<='1';
----OUTPUT
                when "010000"=>
                                    -----EINT
                when "010010"=>
                                         -----DINT
                when "010100"=> fin<='1';
                -----FIN INSTRUCCION
                when "010110"=> clear<='0';
                          -----Limpia los registros
                when "011000"=> EnPWM<='1';                               --
---Activa PWM
```

```
                                                if Din(18)='1' then
SelPWM<='1';

else SelPWM<='0';

end if;
                  when "011110"=> if igual_tempo<='0' then Tempo<='1';      --
---Temporizador
                                                else
                                                Tempo<='0';Resetn_T<='0';
                                                end if;
                  when others => null; LoadCp<='0';
            end case;
            else modo<='0';EnB<='0';EnAcum1<='0';EnAcum2<='0';EnPortin<='0';
SeAdd<='0'; EnPortout<='0';SeAlu<='0';SAcum<='0';fin<='0';op<='0';S<='0';
LoadCp<='0';EnZ<='0'; EnC<='0';
            EnCall<='0';EnSub<='0'; EnAdd<='0';ResetSub<='1';
SelectoraCp<="00";op2<='0';clear<='1';Enx<='0';EnA<='0';
EnPWM<='0';Tempo<='0';Resetn_T<='1';SelPWM<='0';

            end if;
      end if;
      end process;
end behavioral;
```

Anexo 7: Código VHDL de la ALU

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity ALU is
      port (A, B: in std_logic_vector(7 downto 0);
                  Ci: in std_logic;
                  Selectora: in std_logic_vector(5 downto 0);
                  Q: out std_logic_vector(7 downto 0);
                  Z, C: out std_logic);
end ALU;

architecture behavioral of ALU is
signal Ao, Bo, Qo: std_logic_vector(8 downto 0);
      begin
                  Ao<= '0'& A;
                  Bo<= '0'& B;
                  process(Selectora, A, B)
                  begin
                        case Selectora is
                        --------coding---------------Mnemonic
                              when "100000"=> Qo<= Ao+Bo; ------ADD
                              when "100010"=> Qo<= Ao+Bo+Ci;----ADDC
                              when "100100"=> Qo<= Ao-Bo;  -----SUB
                              when "100110"=> Qo<= Ao-Bo-Ci; ---SUBC
                              when "101000"=> Qo<= Ao+1;    ----INC
                              when "101010"=> Qo<= Ao-1;     ----DEC
                              when "101100"=> Qo<= Ao(7 downto 0) &'0'; ----
SHL
                              when "101110"=> Qo<= Ao(8)&Ao(8 downto 1); ---
SHR
```

```
                              when "110000"=> Qo<= Ao(6 downto 0)& Ci & Ao(7);
---ROL with carry
                              when "110010"=> Qo<= Ci&Ao(7 downto 0); ------
ROR with carry

                        --Logic Operations----
                              when "110100"=> Qo<= Ao and Bo;  -----AND
                              when "110110"=> Qo<= Ao or Bo;  ------OR
                              when "111000"=> Qo<= Ao xor Bo;  -----XOR
                              when "111010"=> Qo<= not Ao;   -------NOT

                        ------------------------
                           when "111100"=> Qo<= Ao;       -------LOAD
                              when "111110"=> Qo<= Bo;       -------MOVE
                              when "100001"=> Qo<=
std_logic_vector(to_unsigned(to_integer(unsigned(Ao))*to_integer(unsigned(Bo)
),9));      ----MULTIPLICACION
                              when "100011"=> Qo<=
std_logic_vector(to_unsigned(to_integer(unsigned(Ao))/to_integer(unsigned(Bo)
),9));  ---DIVISION
                              when "100101"=> Qo<=
std_logic_vector(to_unsigned(to_integer((unsigned(Ao))*2/5),9));----DUTY PWM
                              when others =>
                        end case;
                        if (Qo(7 downto 0)="00000000") then
                               Z<='1'; else Z<='0';
                               end if;
                  end process;
                  C<=Qo(8);
                  Q<=Qo(7 downto 0);
end behavioral;
```

Anexo 8: Código VHDL del generador PWM

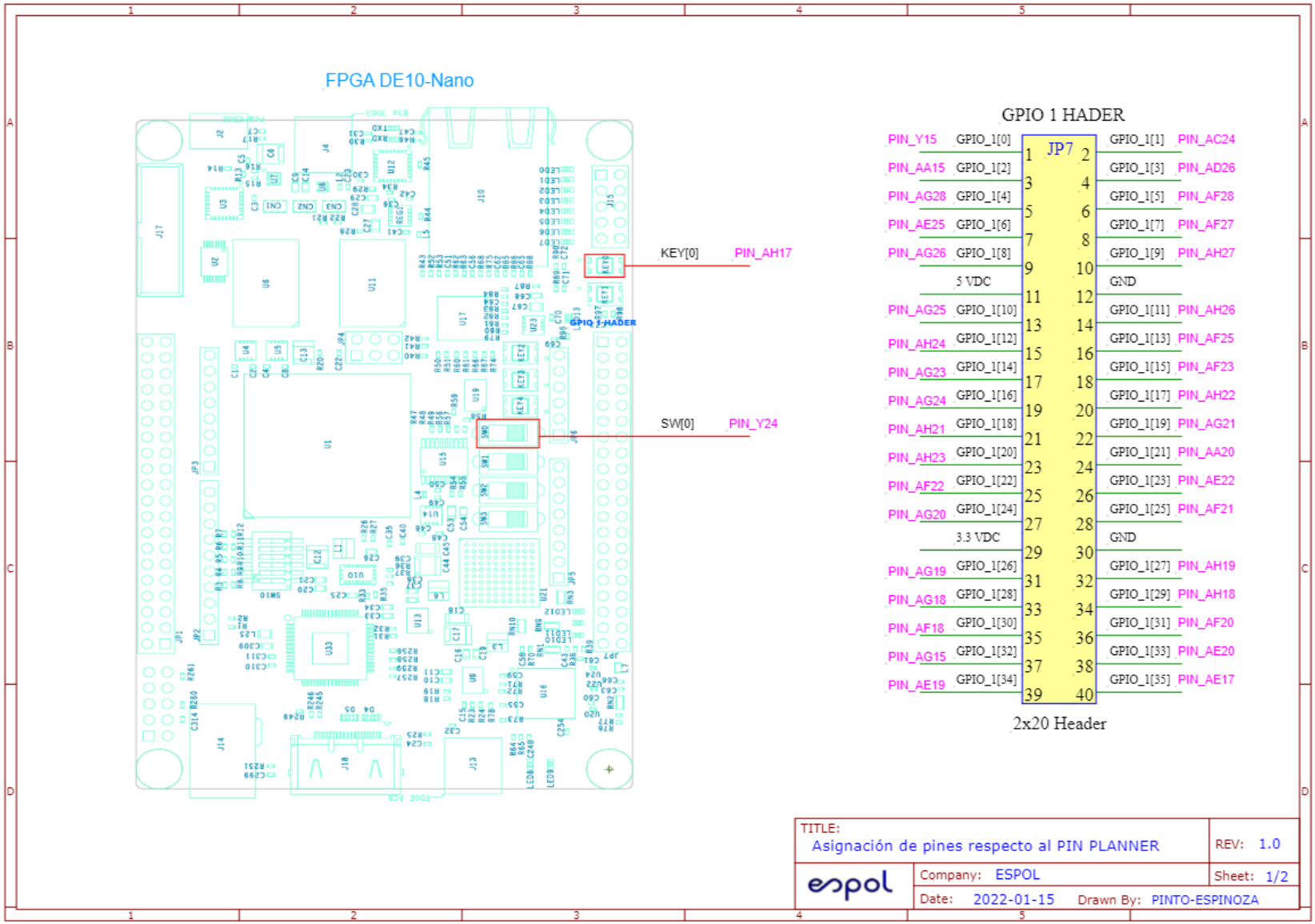
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity pwm_generator is
      generic(
            max_val: integer:= 250;
            val_bits: integer:=8
            );
      port (
            clock: in std_logic;
            duty_cycle: in std_logic_vector((val_bits-1) downto 0);
            resetn: in std_logic;
            pulso: out std_logic
            );
end entity;
 architecture system_pwm of pwm_generator is
      signal cont: std_logic_vector((val_bits - 1) downto 0);
begin
process (clock, resetn) ----conteo
begin
      if resetn='0' then
            cont<= (others =>'0');
```

```
      elsif (clock'event and clock='1') then
                  if (cont<(max_val-1)) then
                        cont<=cont+1;
                  else
                        cont<=(others=>'0');
                  end if;
      end if;
end process;
---asignación de señales---
process (clock, resetn)
begin
  if resetn='0' then pulso<='0';
  elsif (clock'event and clock='1') then
            if (duty_cycle > cont) then
            pulso<='1';
            else
            pulso<='0';
            end if;
      end if;
end process;
end system_pwm;
```

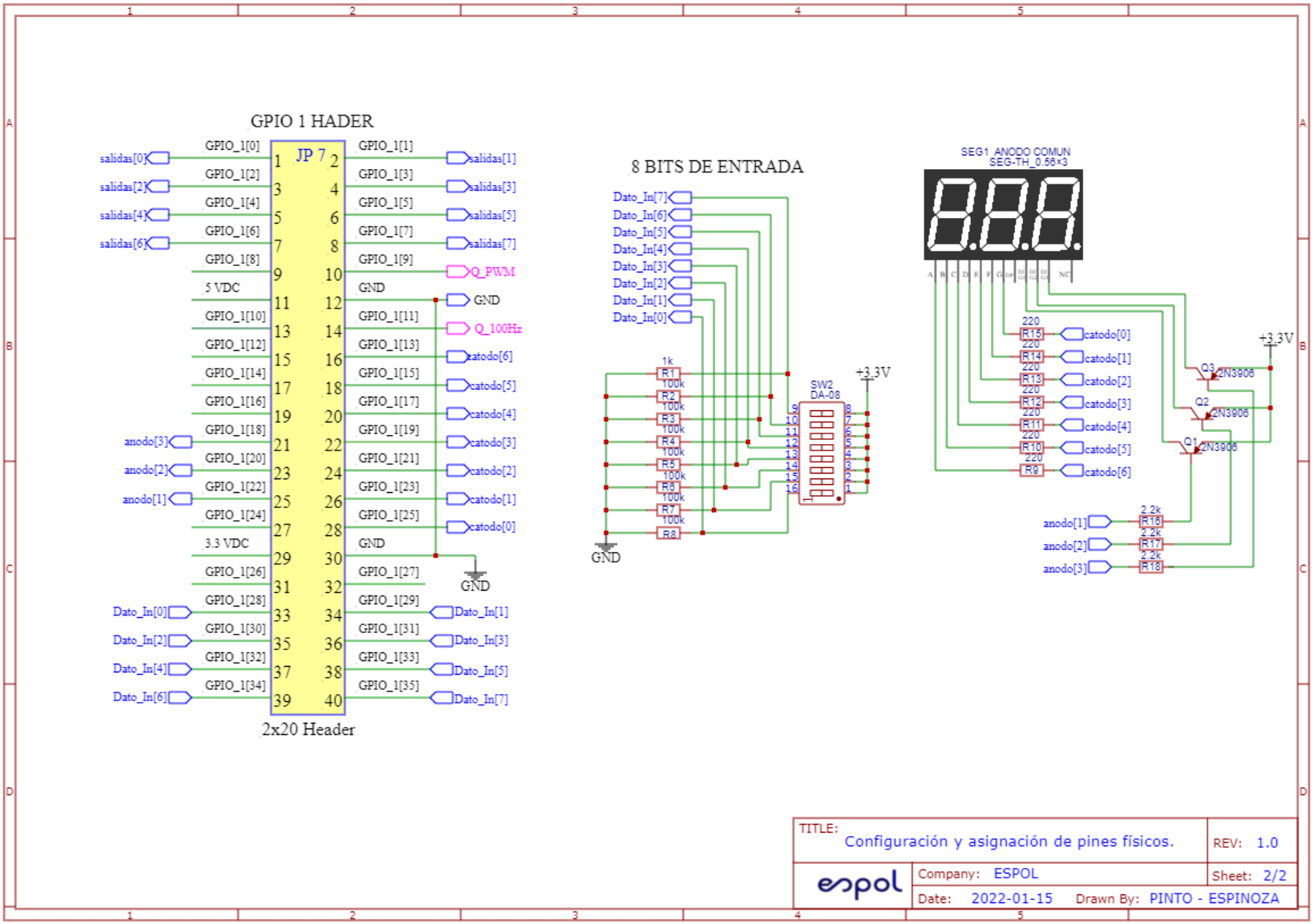
Anexo 9: Coste de la tarjeta de desarrollo

Producto	Precio
FPGA DE10-Nano	$208

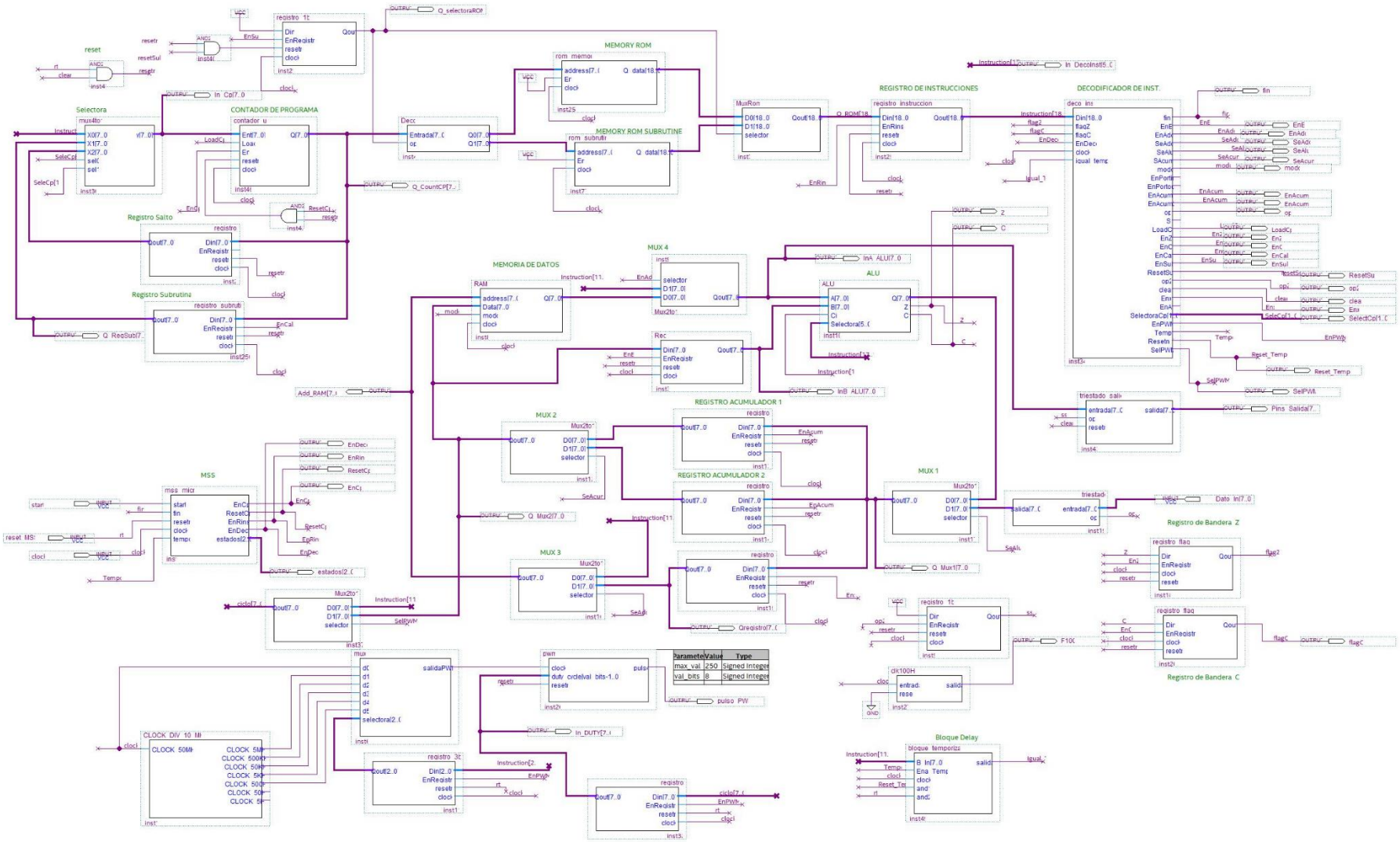
Anexo 10: Asignación de pines respecto al Pin Planner



Anexo 11: Configuración y asignación de pines físicos



Anexo 12: Diagrama del microcontrolador desarrollado



Anexo 13: Guía de práctica para el uso del microcontrolador desarrollado

PRÁCTICA

CAPÍTULO DEL CURSO: FUNDAMENTOS DE MICROCONTROLADORES Y SISTEMAS EMBEBIDOS

TEMA DE LA ACTIVIDAD: SISTEMA DIGITAL BASADO EN MICROCONTROLADOR

OBEJTIVOS DE APRENDIZAJE:

- Ejecutar una memoria rom de una arquitectura de un microcontrolador a través de un archivo de texto con extensión .mif
- Analizar la simulación en un diagrama de tiempo de las principales señales que conforman la arquitectura de un microcontrolador.
- Generar una señal PWM con un duty cycle variable por medio del ingreso de los dos últimos dígitos de la matricula del estudiante a través de los pines de la tarjeta FPGA DE10-Nano.
- Probar la generación de una señal PWM en un motor DC.
- Obtener de manera teórica y experimental el duty cycle de la señal PWM generada por el código ingresado en la memoria del microcontrolador.
- Diseñar el diagrama ASM del bloque de control del microcontrolador en vhdl, utilizando las señales del microcontrolador para la verificación de los estados de la máquina secuencial sincrónica.

DURACIÓN: 120 minutos

MATERIALRES Y HERRAMIENTAS:

- Software Intel Quartus Prime versión 18.1
- Tarjeta FPGA DE10-Nano
- Osciloscopio Digital y/o multímetro (con capacidad de medir frecuencia)
- Protoboard
- Display de 7 segmentos ánodo común de 3 dígitos
- DIP switch de 8 pines
- 3 transistores 2n3906
- Motor de 12 Vdc
- TIP 41C
- Fuente de 12Vdc
- Jumpers

MARCO TEÓRICO:

Un microcontrolador con arquitectura tipo Harvard es una arquitectura de computadora en la que está físicamente separadas por las instrucciones y los datos. La arquitectura de este microcontrolador está básicamente formada por una memoria ROM que contiene el archivo del programa donde se almacenan las instrucciones y una memoria RAM que permitirá almacenar la información ya sea ingresada por el usuario a través de los pines de entrada del microcontrolador o

generada por las operaciones que ejecute la Unidad Aritmética Lógica ALU. Adicional está conformada por bloques de registros que permite almacenar valores para propósito particular, por bloques multiplexores, un contador de programa y un decodificador de instrucciones. Se cuenta con una Unidad de Control, que constituye un bloque, el cual permitirá realizar el proceso secuencial del microcontrolador que es buscar, decodificar, y ejecutar la instrucción.

DESCRIPCIÓN DE LA PRÁCTICA: Se ingresarán los dos últimos dígitos que constituyen el número de matrícula a través del puerto de entrada del microcontrolador diseñado, luego estos valores se multiplicarán y el resultado se duplicará para posteriormente ingresar como el duty cycle de la señal PWM que será activada por código. Además, se analizarán los bloques del sistema digital al igual que el diagrama de tiempo donde se visualizarán las principales señales que componen la arquitectura del microcontrolador.

PROCEDIMIENTO:

1. Ejecute el programa Quartus Prime 19.1 Standard Edition que se encuentra en el escritorio representado por el ícono 

.

2. Proceda a crear un nuevo proyecto, seleccione la opción que se le mostrará en la ventana de Quartus, llamada **New Project Wizard,** como se muestra en la figura 1.

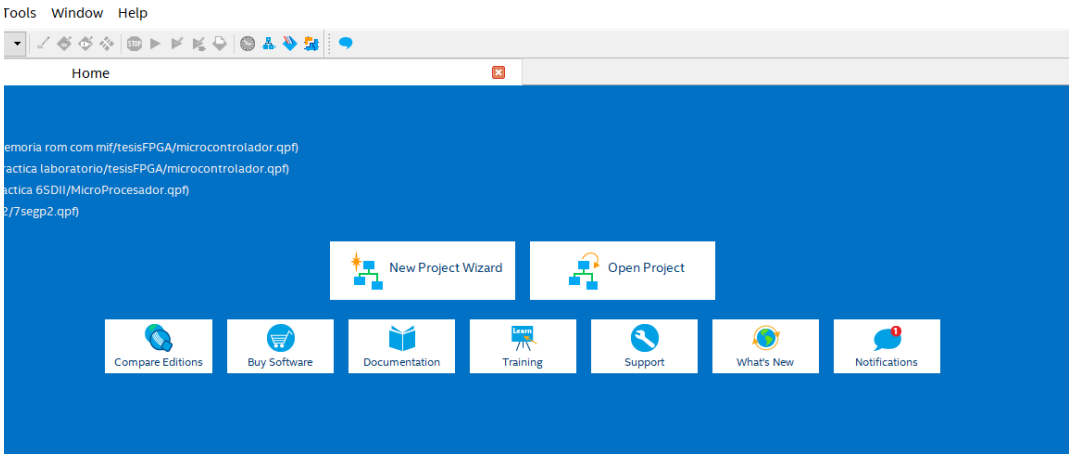


Figura 1. Ventana principal del programa Quartus Prime 18.1 Standard Edition

3. Proceda a guardar la dirección del proyecto en una carpeta vacía, en este caso la carpeta se llamará **PracticaMicro.** Asigne el nombre del proyecto como **MicroPrueba,** tal como se muestra en la figura 2

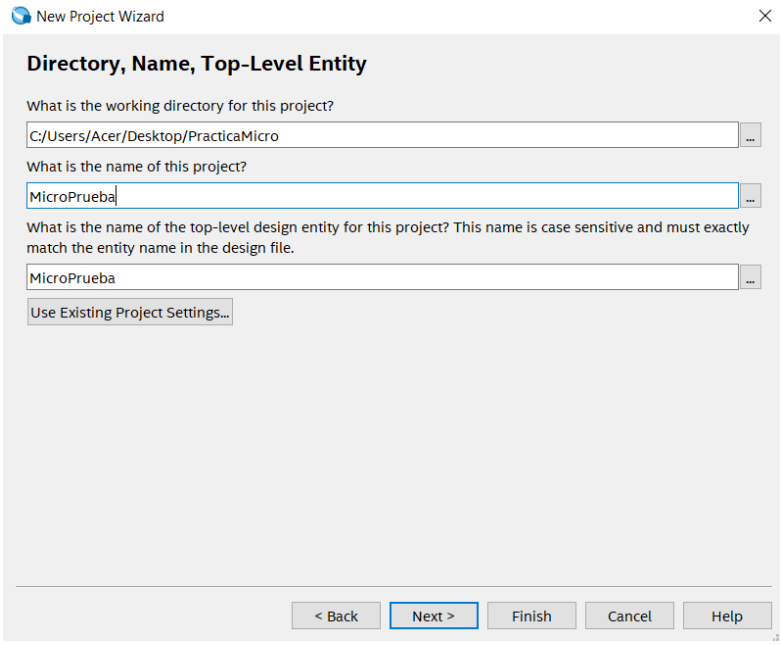


Figura 2. Ventana del New Project Wizard, en donde se especifica la ruta de dirección del proyecto a crear

4. Luego de seleccionar **Next** en la ventana anterior, se le mostrará la siguiente ventana como en la figura 3 donde debe seleccionar el dispositivo la FPGA de la familia **Cyclone V (E/GX/GT/SX/SE/ST)** y el nombre del dispositivo de la tarjeta, en este caso corresponde a 5CSEBA6U2317.

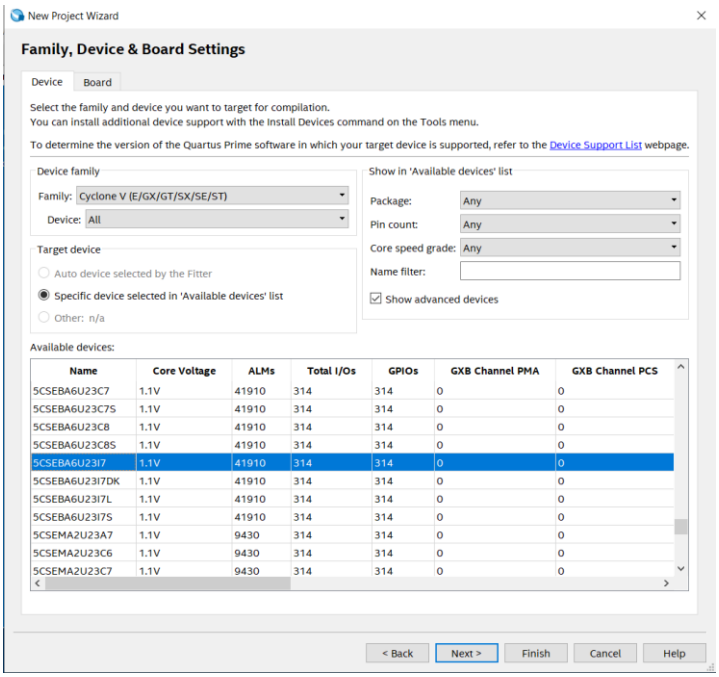


Figura 3. Selección del dispositivo y familia de la FPGA DE10-Nano

5. A continuación, en la siguiente ventana recuerde habilitar la opción de **ModelSim-Altera** en la columna **Tool Name** de Simulation y en la columna **Format(s)** VHDL, como se muestra en la figura 4. Luego click en Next.

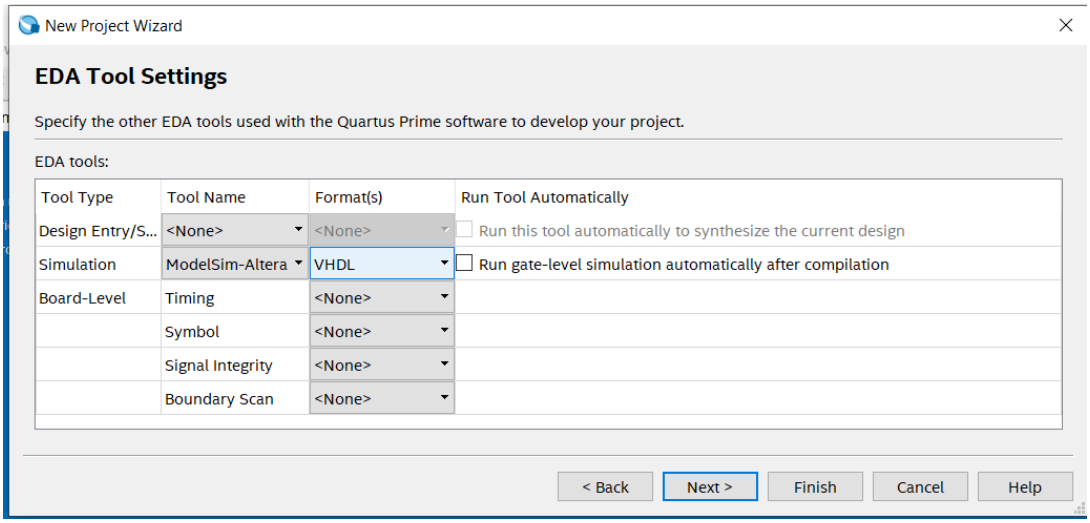


Figura 4. Selección del modo ModelSim-Altera y formato VHDL para la simulación del proyecto

6. Una vez creado el proyecto, integre todos los archivos entregados en la carpeta compartida de la práctica, tal como se muestra en la figura 5. Recuerde que el archivo debe de descomprimirse dentro de la carpeta de su proyecto, que en este caso es **PracticaMicro**.

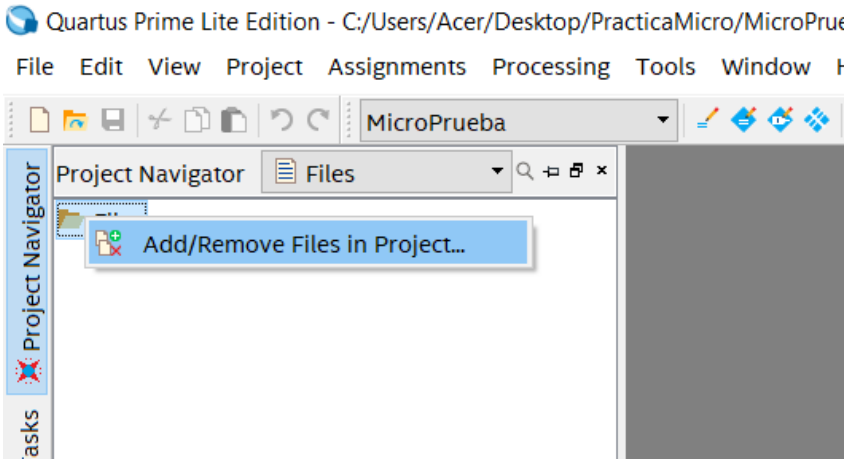


Figura 5. Proceso en el cual se muestra el agregado de todos los archivos en el proyecto

7. A continuación, se dirige a la carpeta donde se encuentra guardado el proyecto en este caso es **PracticaMicro** y selecciona el archivo con extensión .mif llamado **informacion.mif** tal como se muestra en la figura 6. Lo abre o ejecuta como una nota de texto o con Notepad++, grabará y guardará la información del documento que se encuentra en la carpeta compartida de recursos del laboratorio.

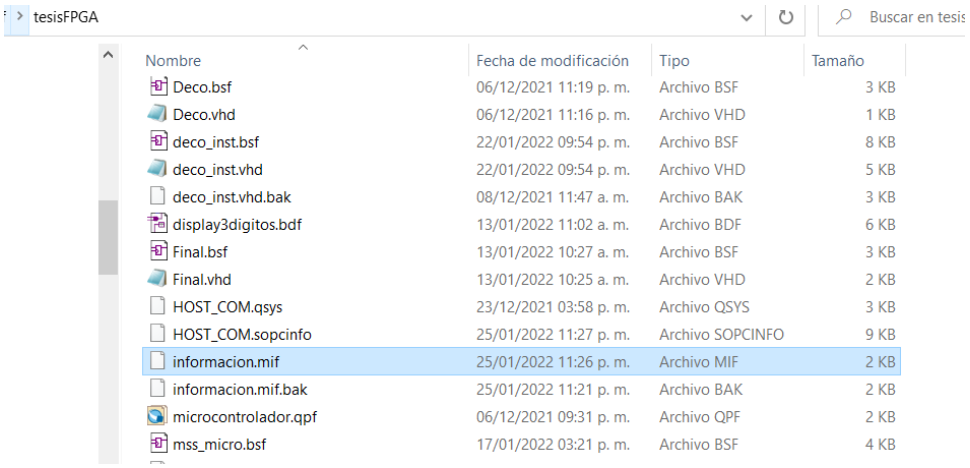


Figura 6. Selección y grabado del archivo con extensión .mif que contiene la información del programa

8. Ahora establezca como más alta jerarquía al archivo **bloque_micro.vhd**, para ello de click derecho sobre el archivo y selecciona la opción **Set as Top-Level Entity** tal como se indica en la figura 7.

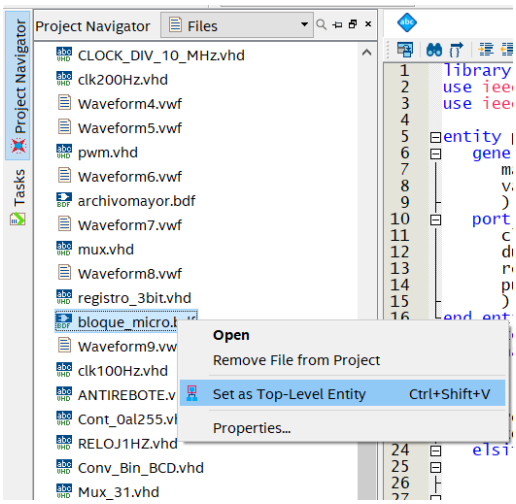


Figura 7. Asignación de la más alta jerarquía al archivo bloque_micro.bdf

9. Luego ejecute la compilación del archivo **bloque_micro.vhd** dando click en el ícono y luego genere un archivo de simulación llamado University Program Waveform, para ello diríjase a **File→New→University Program VWF** como se muestra en la figura 8.

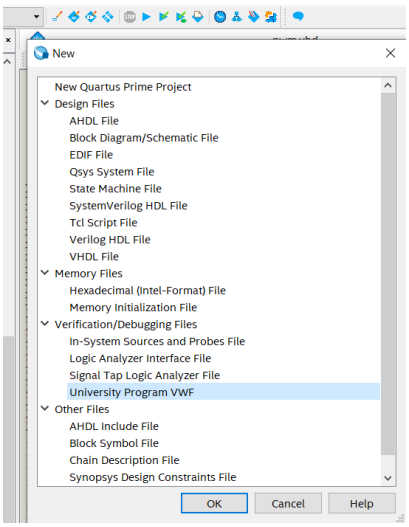


Figura 8. Creación del archivo de University Program VWF

10. A continuación, proceda a colocar todas las señales en orden de prioridad dentro del diagrama de tiempo como se observa en la siguiente figura 9.

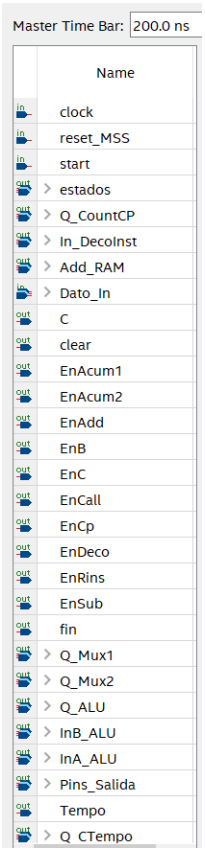


Figura 9. Señales correspondientes al archivo bloque_micro.bdf

11. Para la señal del **clock** establezca un clock de 10 [ns], mientras que a la señal **reset_MSS** establézcalo en un valor alto y para la señal **start** asígnele un valor alto por un tiempo aproximado de 100 ns, tal como se indica en la figura 10.

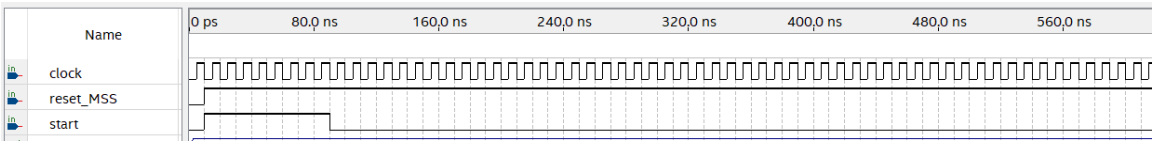


Figura 10. Asignación de valores lógicos a las principales señales de entrada del archivo bloque_micro.bdf

12. Ahora guarde el archivo de la simulación con el nombre que viene asignado por default como **Waveform.vwf** y proceda a realizar la simulación del archivo dando click en el ícono **Run Functional Simulation.** 

13. De las señales de la simulación del **Simulation Waveform Editor** que se obtienen, proceda a contestar las siguientes preguntas del reporte: pregunta#1, pregunta#2, pregunta #3, pregunta#4, pregunta#5 y pregunta#6.

14. Ahora para esta parte diríjase nuevamente a la carpeta donde se encuentra guardado el proyecto, en este caso es **PracticaMicro** y selecciona el archivo con extensión .mif llamado **informacion.mif** tal como se muestra en la figura 6. Lo abre o ejecuta como una nota de texto o con Notepad++, ahora grabará y guardará la información del documento que se encuentra en la carpeta compartida de recursos del laboratorio.

15. Establezca como más alta jerarquía al **archivomayor.bdf**, para ello haga clic derecho sobre el archivo y seleccione la opción **Set as Top-Level Entity** tal como se indica en la figura 11, luego compile el archivo.

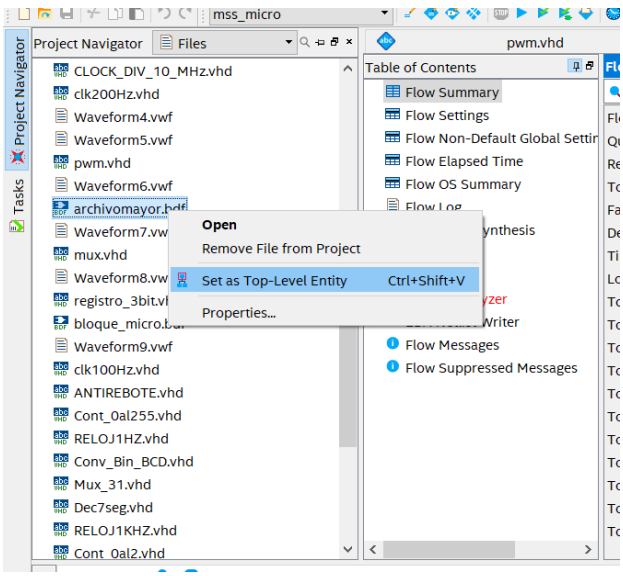


Figura 11. Asignación de la más alta jerarquía al archivo ***archivomayor.bdf***

16. Una vez finalizada de manera exitosa la compilación proceda a asignar los pines de la tarjeta de acuerdo a las señales de entrada y salida del microcontrolador propuesto. Para aquello haga clic en el ícono **Pin Planner**  ubicado en la barra de herramientas de Quartus.

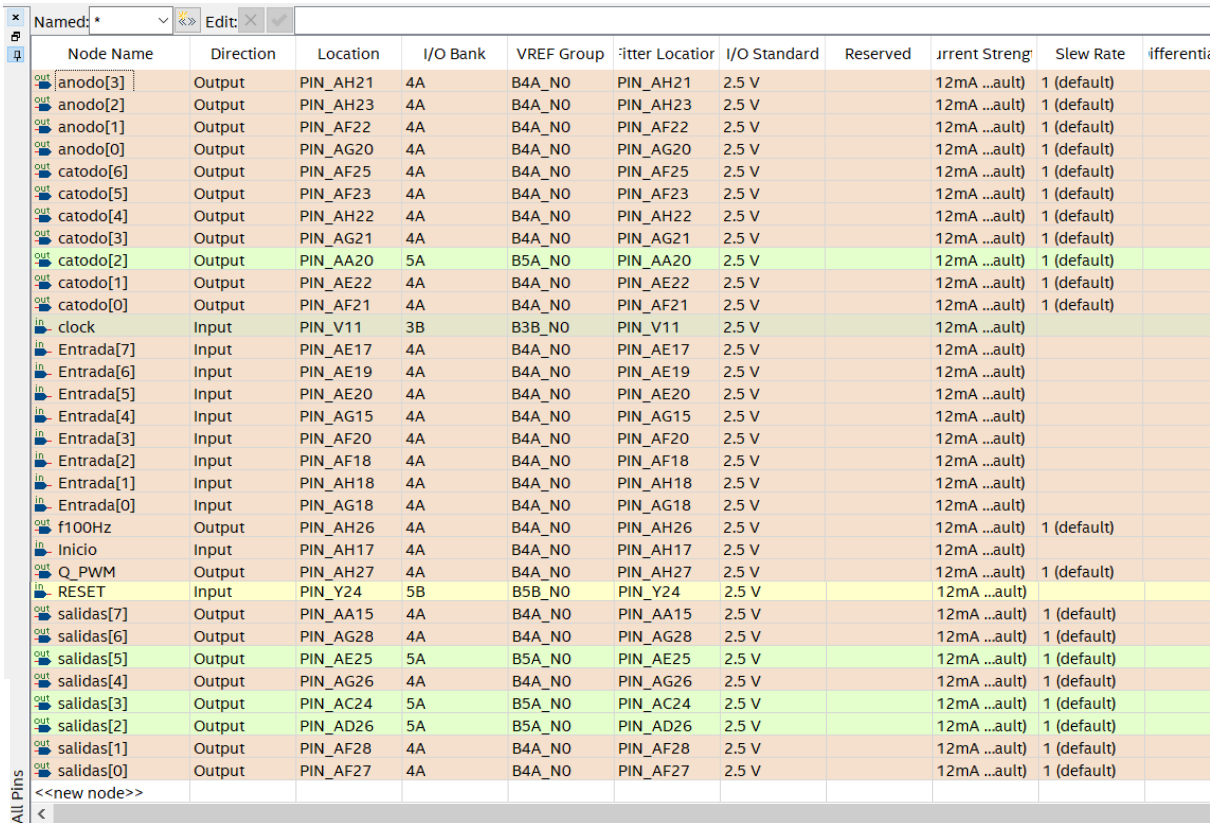
17. En la columna **Location** proceda a colocar cada uno de los pines de la tarjeta de acuerdo a las señales y pines indicados en la Tabla 1 y Tabla2. La asignación final de los pines se muestra en la figura 12.

SEÑAL	PIN FPGA
Inicio	AH17
RESET	Y24
Clock	V11
Entrada[7]	AE17
Entrada[6]	AE19
Entrada[5]	AE20
Entrada[4]	AG15
Entrada[3]	AF20
Entrada[2]	AF18
Entrada[1]	AH18
Entrada[0]	AG18
salidas[7]	AA15
salidas[6]	AG28
salidas[5]	AE25
salidas[4]	AG26
salidas[3]	AC24
salidas[2]	AD26
salidas[1]	AF28
salidas[0]	AF27

Tabla 1. Distribución 1 de los pines en la FPGA

SEÑAL	PIN FPGA
anodo[3]	AH21
anodo[2]	AH23
anodo[1]	AF22
anodo[0]	AG20
catodo[6]	AF25
catodo[5]	AF23
catodo[4]	AH22
catodo[3]	AG21
catodo[2]	AA20
catodo[1]	AE22
catodo[0]	AF21
f100Hz	AH26
Q_PWM	AH27

Tabla 2. Distribución 2 de los pines en la FPGA.



Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	:itter Locatior	I/O Standard	Reserved	ırrent Streng	Slew Rate	ifferenti
anodo[3]	Output	PIN_AH21	4A	B4A_N0	PIN_AH21	2.5 V		12mA ...ault)	1 (default)	
anodo[2]	Output	PIN_AH23	4A	B4A_N0	PIN_AH23	2.5 V		12mA ...ault)	1 (default)	
anodo[1]	Output	PIN_AF22	4A	B4A_N0	PIN_AF22	2.5 V		12mA ...ault)	1 (default)	
anodo[0]	Output	PIN_AG20	4A	B4A_N0	PIN_AG20	2.5 V		12mA ...ault)	1 (default)	
catodo[6]	Output	PIN_AF25	4A	B4A_N0	PIN_AF25	2.5 V		12mA ...ault)	1 (default)	
catodo[5]	Output	PIN_AF23	4A	B4A_N0	PIN_AF23	2.5 V		12mA ...ault)	1 (default)	
catodo[4]	Output	PIN_AH22	4A	B4A_N0	PIN_AH22	2.5 V		12mA ...ault)	1 (default)	
catodo[3]	Output	PIN_AG21	4A	B4A_N0	PIN_AG21	2.5 V		12mA ...ault)	1 (default)	
catodo[2]	Output	PIN_AA20	5A	B5A_N0	PIN_AA20	2.5 V		12mA ...ault)	1 (default)	
catodo[1]	Output	PIN_AE22	4A	B4A_N0	PIN_AE22	2.5 V		12mA ...ault)	1 (default)	
catodo[0]	Output	PIN_AF21	4A	B4A_N0	PIN_AF21	2.5 V		12mA ...ault)	1 (default)	
clock	Input	PIN_V11	3B	B3B_N0	PIN_V11	2.5 V		12mA ...ault)		
Entrada[7]	Input	PIN_AE17	4A	B4A_N0	PIN_AE17	2.5 V		12mA ...ault)		
Entrada[6]	Input	PIN_AE19	4A	B4A_N0	PIN_AE19	2.5 V		12mA ...ault)		
Entrada[5]	Input	PIN_AE20	4A	B4A_N0	PIN_AE20	2.5 V		12mA ...ault)		
Entrada[4]	Input	PIN_AG15	4A	B4A_N0	PIN_AG15	2.5 V		12mA ...ault)		
Entrada[3]	Input	PIN_AF20	4A	B4A_N0	PIN_AF20	2.5 V		12mA ...ault)		
Entrada[2]	Input	PIN_AF18	4A	B4A_N0	PIN_AF18	2.5 V		12mA ...ault)		
Entrada[1]	Input	PIN_AH18	4A	B4A_N0	PIN_AH18	2.5 V		12mA ...ault)		
Entrada[0]	Input	PIN_AG18	4A	B4A_N0	PIN_AG18	2.5 V		12mA ...ault)		
f100Hz	Output	PIN_AH26	4A	B4A_N0	PIN_AH26	2.5 V		12mA ...ault)	1 (default)	
Inicio	Input	PIN_AH17	4A	B4A_N0	PIN_AH17	2.5 V		12mA ...ault)		
Q_PWM	Output	PIN_AH27	4A	B4A_N0	PIN_AH27	2.5 V		12mA ...ault)	1 (default)	
RESET	Input	PIN_Y24	5B	B5B_N0	PIN_Y24	2.5 V		12mA ...ault)		
salidas[7]	Output	PIN_AA15	4A	B4A_N0	PIN_AA15	2.5 V		12mA ...ault)	1 (default)	
salidas[6]	Output	PIN_AG28	4A	B4A_N0	PIN_AG28	2.5 V		12mA ...ault)	1 (default)	
salidas[5]	Output	PIN_AE25	5A	B5A_N0	PIN_AE25	2.5 V		12mA ...ault)	1 (default)	
salidas[4]	Output	PIN_AG26	4A	B4A_N0	PIN_AG26	2.5 V		12mA ...ault)	1 (default)	
salidas[3]	Output	PIN_AC24	5A	B5A_N0	PIN_AC24	2.5 V		12mA ...ault)	1 (default)	
salidas[2]	Output	PIN_AD26	5A	B5A_N0	PIN_AD26	2.5 V		12mA ...ault)	1 (default)	
salidas[1]	Output	PIN_AF28	4A	B4A_N0	PIN_AF28	2.5 V		12mA ...ault)	1 (default)	
salidas[0]	Output	PIN_AF27	4A	B4A_N0	PIN_AF27	2.5 V		12mA ...ault)	1 (default)	
<<new node>>										

All Pins

Figura 12. Pines asignados de acuerdo a la tabla 1 y tabla 2

18. Una vez completa la asignación de los pines para todas las señales indicadas anteriormente, cierre la ventana **Pin Planner** y nuevamente realice la compilación del archivo con la opción **Start Compilation**, una vez finalizada de manera exitosa la compilación, de clic sobre el ícono **Progammer** para grabar la tarjeta FPGA DE10-Nano.

19. En este paso se realizará la programación de la tarjeta, por lo tanto, la tarjeta debe estar encendida y conectada a la PC.

    I. Una vez ejecutado **Programmer** se le mostrará la siguiente ventana como en la figura 13 en donde tiene que buscar el dispositivo FPGA DE10-Nano.
    II. Diríjase a la opción **Add Device,** se le desplegará una ventana y seleccione en Device Family **Cyclone V** y en Device Name **5CSEBA6U23**. Luego clic en OK, como se indica en la figura 14.

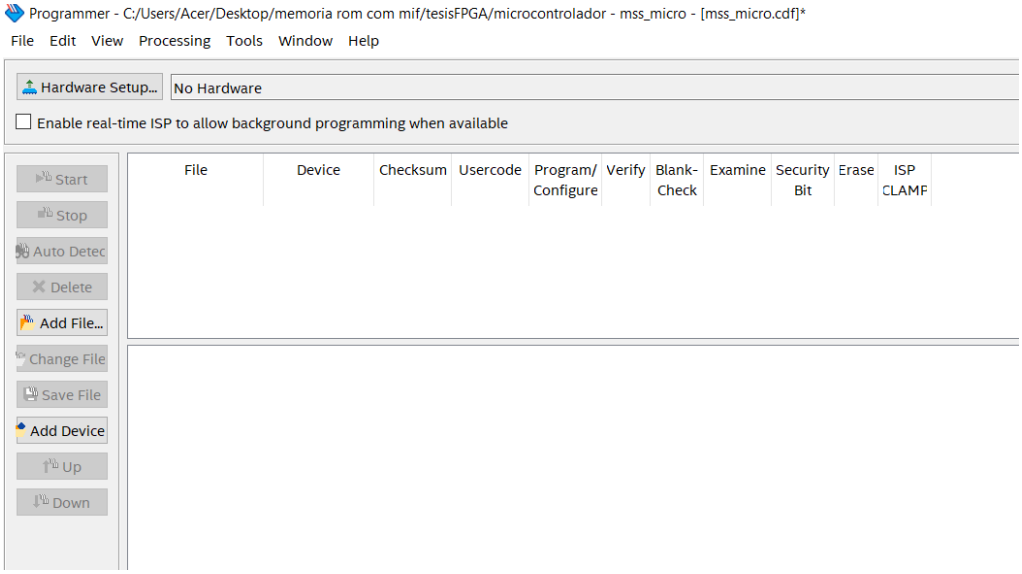


Figura 13. Venta Programmer en donde no se visualiza el dispositivo FPGA DE 10-Nano

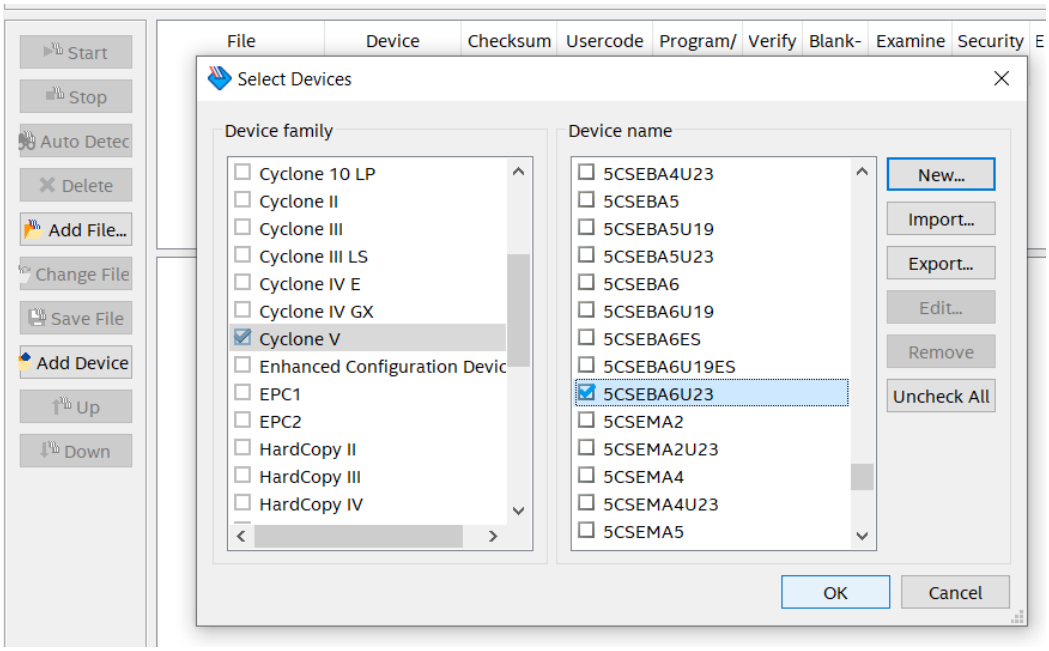


Figura 14. Selección de la familia y nombre del dispositivo FPGA DE10-Nano

III. Se le mostrará los dos dispositivos que conforman la FPGA DE10-Nano. Asegúrese que en la opción **Hardware Setup** se encuentre seleccionado **DE-SoC [USB-1]** que indica que la tarjeta ha sido reconocida por el ordenador, tal como se muestra en la figura 15.

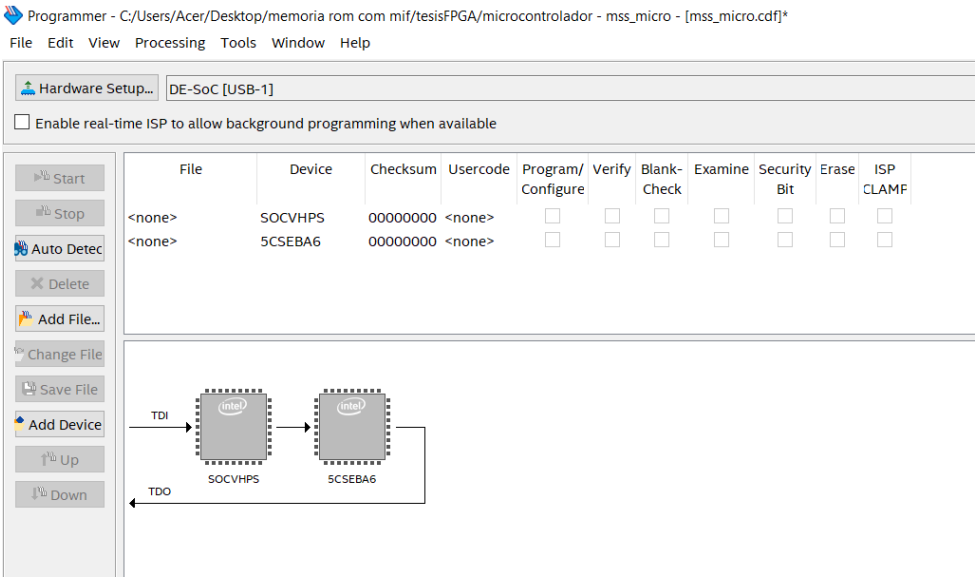


Figura 15. Reconocimiento de los dispositivos SOCVHPS y 5CSEBA6 de la tarjeta FPGA DE10-Nano

IV. Para grabar la información dentro de la tarjeta FPGA DE10-Nano diríjase al Device **5CSEBA6,** clic derecho sobre <none> y se le desplegará una pestaña en donde debe seleccionar la opción **Change File** como se indica en la figura 16

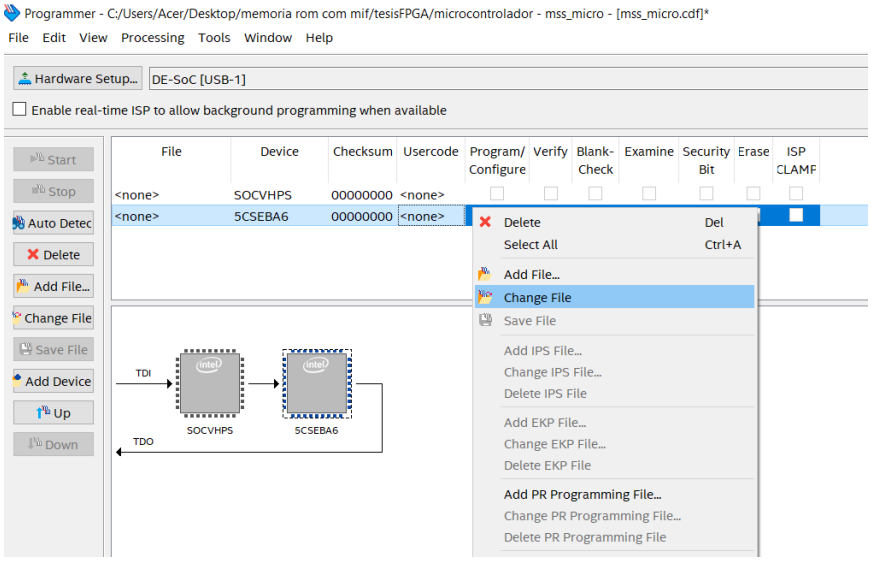


Figura 16. Selección del dispositivo 5CSEBA6 para el grabado de la información

V. Se le ejecutará una venta llamada **Select New Programming File** en el cual seleccione la carpeta **output_files** que contiene el archivo **mss_micro.sof**. Clic sobre el mismo archivo y luego en **Open,** como se indica en la figura 17.

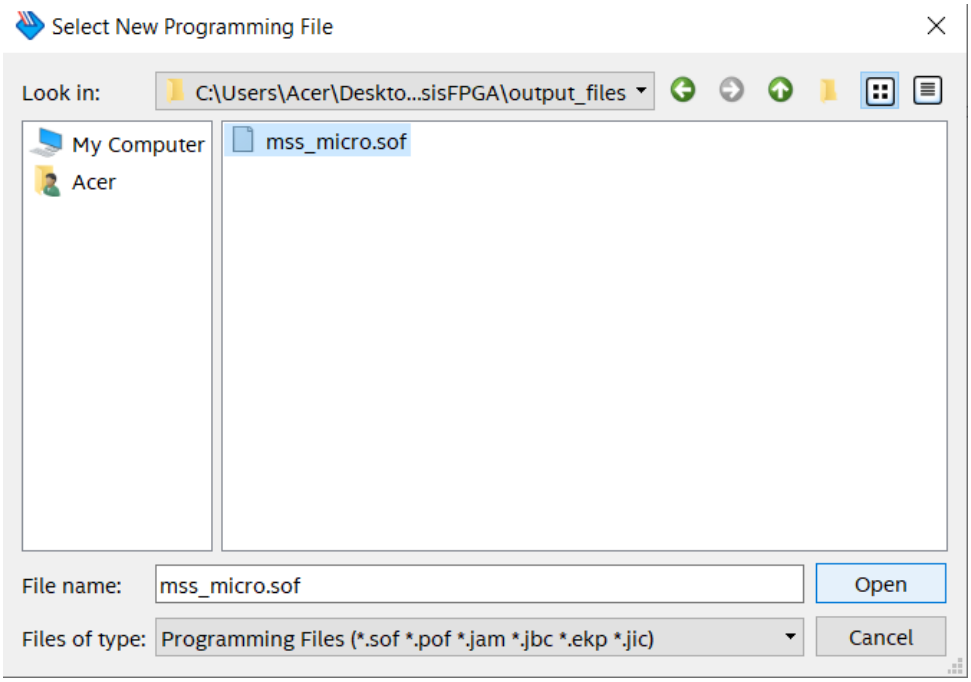


Figura 17. Selección del archivo con extensión .sof que contiene la arquitectura del microcontrolador

VI. A continuación, asegúrese de que la casilla ***Program/Configure*** se encuentre habilitada, luego haga clic sobre el ícono ***Start***, para iniciar el proceso de carga de información hacia la tarjeta FPGA DE10-Nano, tal como se muestra en la figura 18.

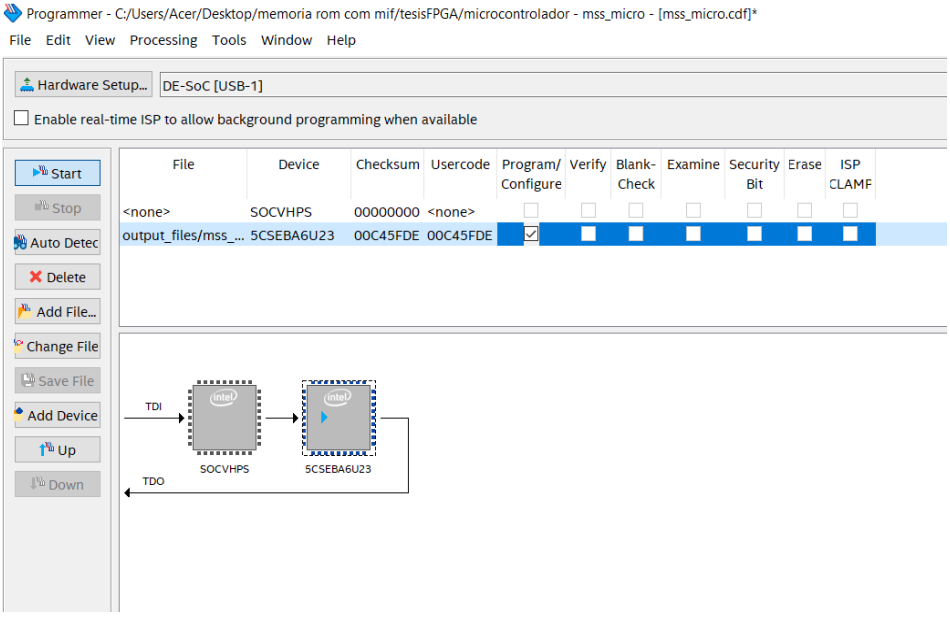


Figura 18. Proceso de grabado de la información hacia la tarjeta FPGA DE10-Nano

VII. Verifique que dentro de la barra ***Progress*** se presente el mensaje de **100%(*Successful*)** como se muestra en la figura 19.

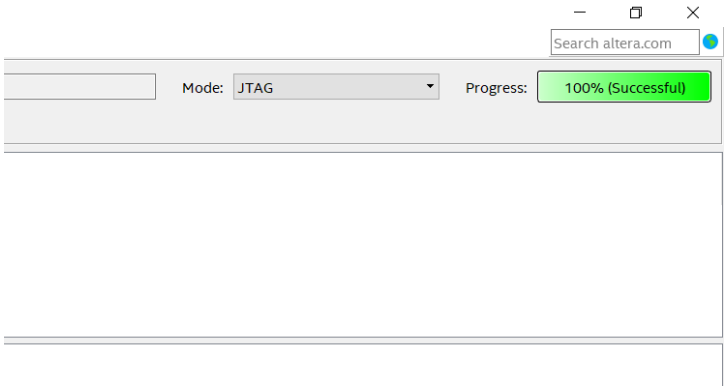


Figura 19. Proceso satisfactorio de carga del programa hacia la tarjeta física FPGA DE10-Nano

20. Realice la conexión de todos los elementos como lo son resistencias, DIP switch, transistores y display de 7 segmentos en el Protoboard de manera correcta tal como se muestra en la figura 20. Tenga en cuenta que la conexión de la alimentación del motor de 12 Vdc debe ser tomada de una fuente externa de 12 Vdc.

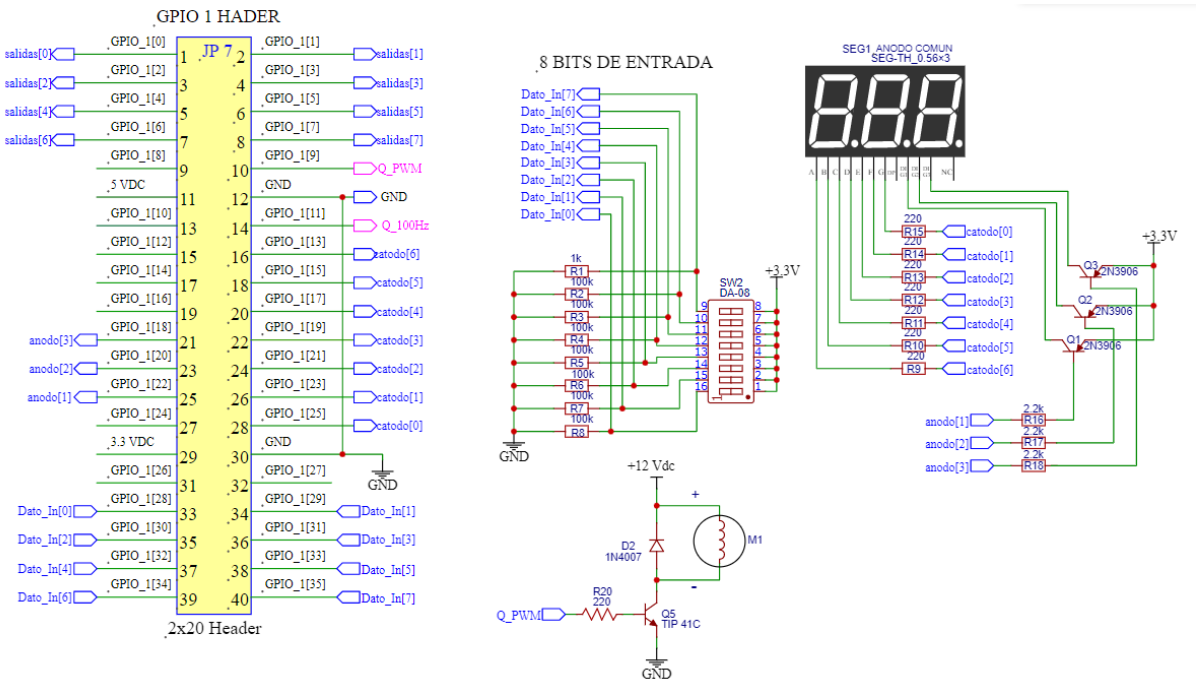


Figura 20. Diagrama de conexión de los pines, resistencias, transistores, DIP switch, displays y motor DC

21. Teniendo en cuenta los dos últimos dígitos de su matrícula, deberá calcular el valor del duty cycle para la señal PWM a generar. En este caso multiplique los dos dígitos a ingresar y luego a este resultado sáquele el duplo. Al valor final

que obtiene, ingréselo a la ecuación 2 para obtener el duty cycle de manera teórica.

$$valor\ duty\ decimal = primer\ digito\ \cdot segundo\ dígito \cdot 2 \qquad (1)$$

$$\%Duty = \frac{valor\ duty\ decimal}{250} * 100 \qquad (2)$$

22. Inicialice el sistema activando en primer lugar el switch **Reset_MSS**, luego presione el push botton **Start** como se muestra en la figura 21**.** Ahora ingrese el primer digito y el segundo a través del DIP SWITCH conectado en el protoboard. Recuerde que una vez presionado **Start** tiene 8 segundos para el ingreso del primer dígito y 8 segundos más para el ingreso del segundo dígito.

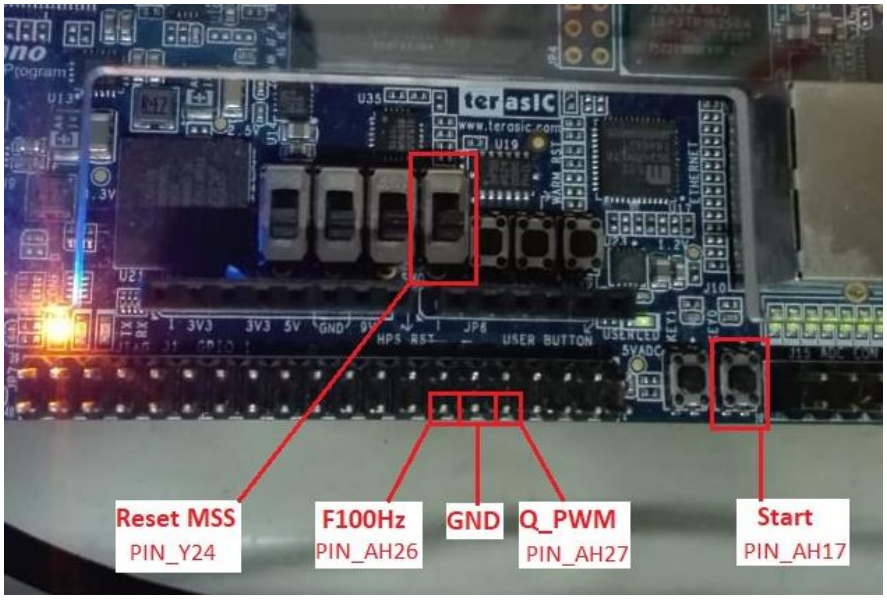


*Figura 21. Ubicación del **Reset_MSS** y del **Start** dentro de la tarjeta FPGA DE10-Nano*

23. Visualice en pantalla el resultado obtenido y compárelo con el calculado por la ecuación anterior. Adicional con un osciloscopio o un multímetro tome la lectura de la frecuencia y ciclo de trabajo positivo de la salida Q_PWM.

24. Repita el proceso de compilación anterior pero ahora con el archivo de la memoria ROM generado en el primer archivo compartido de la práctica desde la carpeta de recursos compartidos del laboratorio.

25. Anote los resultados obtenidos y responda las preguntas 7, 8, 9, 10, 11, y describa las conclusiones y recomendaciones de la práctica.

PREGUNTAS:

1. Adjunte una captura de pantalla del reporte de la simulación de la ventana del Simulation Waveform Editor y realice la respectiva explicación de cada una de las señales.

2. Adjunte captura de pantalla de la partición funcional (visor RTL) del archivo bloque_micro.bdf.

3. En base a la pregunta anterior determine los bloques utilizados del microcontrolador y realice una breve descripción de cada uno de ellos en donde indica su función.

[Click or tap here to enter text.]

4. Elabore el diagrama ASM de la unidad de control del microcontrolador en base al archivo waveform.vwf, indicando para que sirve cada uno de los estados.

[Click or tap here to enter text.]

5. Del bus de señal denominado **In_DecoInst** enliste cada una de las instrucciones que ejecuta el programa y en base a la hoja de instrucciones que posee el microcontrolador, identifique cada una de ellas.

[Click or tap here to enter text.]

6. Del primer código de programa grabado en la rom, calcule el tiempo de ejecución del programa por medio de la siguiente ecuación, tenga en cuenta el clock, y las instrucciones generadas por el archivo waveform.vwf.

$$tiempo\ ejecución = (\#instrucciones)(\frac{ciclos}{instruccion})(Periodo\ del\ clock)$$

[Click or tap here to enter text.]

7. Adjunte imagen del circuito PCB armado en donde se visualice el funcionamiento del microcontrolador, adicional imagen donde se muestra el desarrollo y resultado de la ecuación 1.

8. Adjunte imagen de la medición del osciloscopio o multímetro y comente el resultado que se ha obtenido del pin Q_PWM.

9. Comente que sucedió con el giro del motor al cargar el primer código de la rom y el segundo código de la rom en la tarjeta FPGA DE10-Nano.

   [Click or tap here to enter text.]

10. Indique otras aplicaciones que puede tener la generación de una señal PWM en el campo de la industria.

    [Click or tap here to enter text.]

11. Ahora ingrese nuevamente un valor de 15 y 12 en binario y teniendo en cuenta la fórmula para el porcentaje del duty cycle, explique lo que sucede. (*Tenga en cuenta que todas las operaciones internas aritméticas que realiza el microcontrolador son de 1 byte.)*

    [Click or tap here to enter text.]

CONCLUSIONES Y RECOMENDACIONES

[Mínimo 3 conclusiones y 3 recomendaciones]