UNIVERSIDAD TECNICA FEDERICO SANTA MARIA
SEDE CONCEPCION – REY BALDUINO DE BELGICA

PROGRAMACION DE DISPOSITIVOS LOGICOS PROGRAMABLES FPGA, UTILIZANDO EL LENGUAJE VHDL

Trabajo de titulación para optar al título profesional de Ingeniero de ejecución en control e Instrumentación Industrial.

Alumno:
Mauricio Eugenio Muñoz Armijo

Profesor Guía:
Nelson Vásquez Concha.

2026

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: PROGRAMACION DE DISPOSITIVOS LOGICOS PROGRAMABLES FPGA, UTILIZANDO LENGUAJE VHDL.

Nombre del candidato(a): MAURICIO EUGENIO MUÑOZ ARMIJO

Carrera / Grado: INGENIERIA DE EJECUCION EN CONTROL E INSTRUMENTACION

Campus: CONCEPCION **Departamento:** ELECTRONICA E INDUSTRIAL INFORMATICA

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Nelson Vásquez C., en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción):**

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 26-08-2026 **Firma:**

Estudiante o Candidato(a):

Fecha: 26/08/2026 **Firma:**

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

A mi Madre, hijos y todos a quienes me dieron una palabra de aliento en los momentos difíciles.

RESUMEN

Por medio del presente trabajo de título, se realiza la programación de diversas aplicaciones sobre dispositivos lógicos programables FPGA, utilizando el lenguaje VHDL.

INDICE DE CONTENIDOS

INTRODUCCION 1
DESCRIPCION DEL PROBLEMA 2
PROPUESTA DE SOLUCION 2
OBJETIVO PRINCIPAL 3
OBJETIVOS ESPECIFICOS 3
RESTRICCIONES 3
CAPITULO 1_ Marco teórico 4
1. DISPOSITIVOS LOGICOS PROGRAMABLES 5
1.1 DISPOSITIVOS PROGRAMABLES SIMPLES 5
1.2 DISPOSITIVOS LOGICOS PROGRAMABLES DE ALTA DENSIDAD 6
1.3 DISPOSITIVOS LOGICOS PROGRAMABLES COMPLEJOS CPLD 6
1.4 DISPOSITIVOS FPGA 6
1.5 ARQUITECTURA INTERNA DE UN FPGA 8
1.6 HERRAMIENTAS NECESARIAS PARA PROGRAMAR FPGA 9
2. EL LENGUAJE VHDL 10
2.1 ASPECTOS BASICOS 11
2.2 ESTRUCTURA DE UN PROGRAMA EN VHDL 12
2.21 ARCHIVOS DE BIBLIOTECA 13
2.211 PAQUETE STD_LOGIC 1164 14
2.212 PAQUETE NUMERIC STD 14
2.213 PAQUETE MATH REAL 15
2.214 DECLARACION DE ARCHIVOS DE BIBLIOTECA Y PAQUETES 15
2.22 ESTRUCTURA ENTITY (ENTIDAD) 16
2.23 ESTRUCTURA ARCHITECTURE 17
2.3 EL PRIMER PROGRAMA 18
2.4 ELEMENTOS LEXICOS 19
2.41 COMENTARIOS 19
2.42 IDENTIFICADORES 19
2.5 TIPOS DE DATOS 20

2.51 BITS 20
2.52 BIT_VECTOR 20
2.53 STD_LOGIC 20
2.54 STD_LOGIC_VECTOR 21
2.55 SIGNED/UNSIGNED 22
2.56 NATURAL 23
2.57 INTEGER 23
2.58 REAL 24
2.59 STRING 24
2.6 OBJETOS 25
2.61 SIGNAL (SEÑALES) 25
2.62 VARIABLE 26
2.63 CONSTANT 26
2.7 OPERADORES ARITMETICOS 27
2.71 OPERADORES ARITMETICOS CON NUMEROS ENTEROS 27
2.72 OPERADORES ARITMETICOS CON PUNTO FLOTANTE 28
2.8 OPERADORES COMPARATIVOS 30
2.9 OPERADORES LOGICOS 31
2.10 CIRCUITOS COMBINACIONALES Y SECUENCIALES 32
2.11 CIRCUITOS COMBINACIONALES 32
2.111 DECLARACION CONCURRENTE WITH-SELECT-WHEN 34
2.12 CIRCUITOS SECUENCIALES, PROCESS 35
2.121 ESTRUCTURA DE CONTROL IF-ELSIF-ELSE 37
3. ENTORNO DE DESARROLLO QUARTUS II, 17.1 39
3.1 GENERALLIDADES 40
3.2 INSTALACION 40
3.3 CREACION DE PROYECTOS 41
3.4 CREACION DE ARCHIVOS VHDL 46
3.5 COMPILACION, ANALISIS Y SINTETIZADO 48
3.51 ANALISIS Y SINTESIS 49
3.52 DIAGRAMA RTL 50
3.6 SIMULACION DE CODIGOS VHDL 52
3.7 CONFIGURACION DE PINES (PIN PLANER) 55

3.8 CREACION DE DIAGRAMAS DE BLOQUES 57
3.9 PROGRAMACION DE LA PLACA FPGA 63

4. EXPERIENCIAS PRACTICAS DE CIRCUITOS COMBINACIONALES Y SECUENCIALES CON FPGA ENTORNO GENERAL 66
4.1 CIRCUITOS COMBINACIONALES 67
4.11 COMPUERTA LOGICA NAND DE CUATRO ENTRADAS 67
4.12 DECODIFICADOR BCD A SIETE SEGMENTOS 71
4.13 MULTIPLEXOR DE CUATRO A UNA LINEA 75
4.14 MULTIPLEXOR DE CUATRO A UNA LINEA Y DEMULTIPLEXOR UNA A CUATRO LINEAS 79
4.2 CIRCUITOS SECUENCIALES 81
4.21 DIVISOR DE FRECUENCIA, SALIDA DE 1 HZ 81
4.22 CONTADOR BCD 86
4.23 MODULADOR PWM 89
4.24 VISUALIZACION DE MENSAJES EN DISPLAY LCD 96
4.241 INICIALIZACION Y CONFIGURACION 98
4.242 MAQUINAS DE ESTADOS 101
4.243 CODIGO VHDL LCD 2X16 106
4.3 INTERCONECCION DE PROYECTOS MEDIANTE BLOQUES 113
4.31 MODULADOR PWM CONTROLADO POR UN CONTADOR UP/DOWN 113
4.32 CONTADOR DE EVENTOS, HASTA 9999 118

CONCLUSION 121
BIBLIOGRAFIA 122
APENDICE A: ESPECIFICACIONES TECNICAS PLACA MAX-10 123
APENDICE B: EL USB BLASTER 132
APENDICE C: CARACTERES Y COMANDOS DEL MODULO HD 44780 134
APENDICE D: CODIGOS VHDL USADOS EN LOS BLOQUES 140

INDICE DE FIGURAS

FIGURA 1-1_ CLASIFICACION GENERAL DE LOS DISPOSITIVOS LOGICOS PROGRAMABLES.....7

FIGURA 1-2_ JERARQUIA DE SISTEMAS EMBEBIDOS, SEGÚN SU VELOCIDAD 7

FIGURA 1-3_ ARQUITECTURA INTERNA DE UN DISPOSITIVO FPGA.....8

FIGURA 1-4_ ELEMENTOS REQUERIDOS PARA PROGRAMAR CPLD Y FPGA 9

FIGURA 2-1_ ESTRUCTURA BASICA DE UN CODIGO VHDL.....13

FIGURA 2-2_ENTIDAD EN UN CODIGO VHDL 16

FIGURA 2-3_ARQUITECTURA DE UN CODIGO VHDL.....17

FIGURA 2-4_EL PRIMER PROGRAMA VHDL.....18

FIGURA 2-5_ OBJETOS, CONSTANTES Y SEÑALES 25

FIGURA 2-6_ DECLARACION DEL OBJETO CONSTANT 27

FIGURA 2-7_CIRCUITO COMBINACIONAL REQUERIDO PARA ESCRIBIR CODIGO VHDL 33

FIGURA 2-8_ CODIGO VHDL CORRESPONDIENTE AL HARDWARE DE LA FIGURA 2-7 34

FIGURA 2-9_CODIGO BASICO DE UN PROYECTO 36

FIGURA 3-1_ CENTRO DE DESCARGAS.....40

FIGURA 3-2_DESCARGA DE QUARTUS II, VERSION 17.1.....41

FIGURA 3-3_VISTA GENERAL DE QUARTUS II, VERSION 17.1 42

FIGURA 3-4_BARRAS DE HERRAMIENTAS DE QUARTUS II.....43
FIGURA 3-5_CREACION DE CARPETA Y NOMBRE DEL PROYECTO 43
FIGURA 3-6_ PARA UN NUEVO PROYECTO SE DEBE INDICAR
VACIO (EMPTY) 44
FIGURA 3-7_ SELECCIÓN DE LA FAMILIA MAX-10 Y DISPOSITIVO
FPGA 44
FIGURA 3-8_ REPORTEDE LA CREACION DE PROYECTO.....45

FIGURA 3-9_ PROYECTO CREADO, SE VISUALIZA LA VENTANA
SIN ARCHIVOS CREADOS.....45
FIGURA 3-10_CREACION DE UN NUEVO ARCHIVO VHDL.....46
FIGURA 3-11_ SELECCIÓN DEL TIPO DE ARCHIVO, VHDL.....47
FIGURA 3-12_ CAMPO DE TRABAJO EN BLANCO DISPONIBLE
PARA ESCRIBIR CODIGO VHDL 47
FIGURA 3-13_ PRIMER PROGRAMA 48
FIGURA 3-14_ GUARDADO DE ARCHIVO VHDL EN UN PROYECTO.....49
FIGURA 3-15_ ANALISIS Y SINTESIS DEL ARCHIVO VHDL,
DENTRO DEL PROYECTO.....49
FIGURA 3-16_RESUMEN ANALISIS Y SINTETIZADO DEL
ARCHIVO VHDL 50
FIGURA 3-17_ RUTA DE LOS DIAGRAMAS RTL 51
FIGURA 3-18_ DIAGRAMA RTL RESULTANTE, INDICANDO
LAS ENTRADAS, SEÑALES Y SALIDA 54

FIGURA 3-19_ CREACION DE ARCHIVOS DE SIMULACION.....52
FIGURA 3-20_ CAMPO DE TRABAJO, DESPLEGADO POR
SIMULADOR.....52
FIGURA 3-21_ NOMBRE DEL NUEVO ARCHIVO CREADO
POR EL SIMULADOR.....53
FIGURA 3-22_ VENTANA DEL NAVEGADOR CON EL ARCHIVO
DEL SIMULADOR.....53
FIGURA 3-23_ PROCEDIMEINTO PARA DESPLEGAR EL SIMULADOR
DE QUARTUS II.....54
FIGURA 3-24_ SIMULADOR CON 6 ENTRADAS Y LOS DIAGRAMAS
DE TIEMPO 55
FIGURA 3-25_ SALIDA 4 DEL CIRCUITO DESPLEGANDO EL
EL DIAGRAMA DE TIEMPO.....55
FIGURA 3-26_ ACCESO AL PIN PLANER SELECCIONANDO BARRA
DE HERRAMIENTAS.....56

FIGURA 3-27_ CUADRO DESPLEGADO DE PIN PLANER.....56
FIGURA 3-28_ CREACION DE ARCHIVOS PARA DIAGRAMAS
DE BLOQUES.....57
FIGURA 3-29_ GRILLA DE TRABAJO PARA REALIZAR LOS
DIAGRAMA DE BLOQUES.....58
FIGURA 3-30_ CREACION DE UN BLOQUE 58
FIGURA 3-31_ REPORTE DE CREACION DE UN SIMBOLO.....59

FIGURA 3-32_ INSERTAR SIMBOLOS CREADOS EN LA GRILLA.....59
FIGURA 3-33_ GENERACION DE PINES I/O.....60
FIGURA 3-34_ GUARDADO DEL BLOQUE GENERADO CON EXTENSION. Bdf.....60
FIGURA 3-35_ ANALISIS Y SINTETIZADO DEL ARCHIVO DEL BLOQUE.....61
FIGURA 3-36_ RESULTADO DE LA COMPILACION DEL BLOQUE CREADO.....61
FIGURA 3-37_COMPILADO DEL PROYECTO COMPLETO.....62
FIGURA 3-38_ RESUMEN DE LA COMPILACION.....62
FIGURA 3-39_ SELECCIÓN DE ICONOS PARA PROGRAMAR EL DISPOSITIVO FPGA.....63
FIGURA 3-40_ PROGRAMACION DE DISPOSITIVO FPGA.....64
FIGURA 4-1_ PLACA DE DESARROLLO MAX-10.....66
FIGURA 4-2_ HARDWARE DE UNA COMPUERTA NAND DE 4 ENTRADAS.....68
FIGURA 4-3_ DISTRIBUCION DE PINES DE DIP SWITCH DE LA LA PLACA MAX-10.....68
FIGURA 4-4_ SALIDA DEL PIN 93, CONECTADO CON D1.....68
FIGURA 4-5_CODIGO VHDL CORRESPONDIENTE A UNA COMPUERTA NAND DE 4 ENTRADAS.....69
FIGURA 4-6_ DIAGRAMA RTL DE UNA COMPUERTA NAND DE 4 ENTRADAS.....70

FIGURA 4-7_ DIAGRAMA PIN PLANER 71
FIGURA 4-8_ DECODIFICADOR BCD A SIETE SIETE SEGMENTOS 72
FIGURA 4-9_DISTRIBUCION DE PINES DEL FPGA, HACIA
LOS DISPLAY DE SIETE SEGMENTOS 72
FIGURA 4-10_CODIGO VHDL, CORRESPONDIENTE
AL DECODIFICADOR BCD A 7 SEGMENTOS 74
FIGURA 4-11_CONCEPTO BASICO DE UN MULTIPLEXOR
Y DEMULTIPLEXOR 76
FIGURA 4-12_CODIGO VHDL DE UN MULTIPLEXOR DE
4 A 1 LINEA 77
FIGURA 4-13_DIAGRAMA RTL DEL MULTIPLEXOR 4 A 1 LINEAS 78
FIGURA 4-14_REPRESENTACION DEL HARDWARE DE UN
MULTIPLEXOR Y DEMULTIPLEXOR 79
FIGURA 4-15_CODIGO VHDL CORRESPONDIENTE A UN
MULTIPLEXOR Y DEMULTIPLEXOR 80
FIGURA 4-16_REPRESENTACION DE UN DIVISOR DE FRECUENCIA 82
FIGURA 4-17_CAMBIO DE ESTADO EN LA SALIDA DE 1Hz
CADA 25.000.0000 DE PULSOS DE CLOCK 84
FIGURA 4-18_CODIGO DIVISOR DE FRECUENCIA SALIDA DE 1 Hz 84
FIGURA 4-19_RESUMEN DEL RESULTADO DE LA COMPLILACION
DEL PROYECTO 85
FIGURA 4-20_PINES UTILIZADOS EN EL DISPOSITIVO FPGA 85
FIGURA 4-21_CODIGO VHDL DE CONTADOR BCD 87

FIGURA 4-22_PIN PLANER CONTADOR BCD 88
FIGURA 4-23_RESUMEN DE LA COMPILACION DEL PROYECTO 88
FIGURA 4-24_PLACA MAX-10, CONECTADA A DIP SWITCH 8 DIP 89
FIGURA 4-25_HARDWARE PROPUESTO DE CONTROL PWM 90
FIGURA 4-26_FORMA DE ONDA DEL PWM PROPUESTO 90
FIGURA 4-27_CODIGO VHDL DEL PWM 93
FIGURA 4-28_PINES ASIGNADOS AL FPGA 94
FIGURA 4-29_RESUMEN DE LA COMPILACION DEL PROYECTO 94
FIGURA 4-30_MUESTRA PWM EN OSCILOSCOPIO 95
FIGURA 4-31_DIAGRAMA DE CONEXIONADODE UN LCD 96
FIGURA 4-32_DIAGRAAMA DE FLUJO DE ESCRITURA EN LCD 101
FIGURA 4-33_REPRESENTACION DE UNA MAQUINA DE ESTADO 102
FIGURA 4-34_DIAGRAMA DE ESTADO, CON LOGICA
DE SEMAFORO DE CUATRO ESTADOS 103
FIGURA 4-35_MAQUINA DE ESTADOS CON CAMBIOS DE CLOCK 104
FIGURA 4-36_SECUENCIAS DE ESTADOS PARA ENVIAR CARACTERES
A UN LCD 105
FIGURA 4-37_CODIGO VHDL LCD 2x16 108
FIGURA 4-38_DIAGRAMA DE PINES PARA CONECTAR LCD 112
FIGURA 4-39_ESQUEMA DE MODULADOR PWM, CONTADOR
UP/DOWN 114
FIGURA 4-40_AGREGADO DE ARCHIVOS A UN PROYECTO 114
FIGURA 4-41_SECUENCIA DE AGREGAR ARCHIVOS A PROYECTO 115

FIGURA 4-42_VISUALIZACION DE NUEVOS ARCHIVOS 116
FIGURA 4-43_CREACION DE NUEVA MALLA DE TRABAJO PARA EL DIAGRAMA DE BLOQUES 117
FIGURA 4-44_CONTADOR DE 0 A 9999 A IMPLEMENTAR CON DIAGRAMA DE BLOQUES.....119
FIGURA 4-45_ARCHIVOS RECOPILADOS DE OTROS PROYECTOS EXISTENTES.....119
FIGURA 4-46_DIAGRAMA DE BLOQUES DEL CONTADOR DE 0 A 9999.....120

INDICE DE TABLAS

TABLA 2-1_ CODIGO VHDL CORRESPONDIENTE AL TIPO DE COMPUERTA 31

TABLA 4-1_ EQUIVALENTE DEL CODIGO BCD A 7 SEGMENTOS 73

TABLA 4-2_ FUNCION SET 98

TABLA 4-3_ COMANDO SHOW 99

TABLA 4-4_ COMANDO CLEAR DISPLAY 99

TABLA 4-5_ COMANDO ENTRY MODE 100

TABLA 4-6_ CARÁCTER “O” 100

TABLA 4-7_ TABLA DE VERDAD CON TRANSISCION DE ESTADOS 104

SIGLA Y SIMBOLOGIA

PC_ Personal Computer

I/O_ Input/Output (puertos entrada/salida)

USB_Bus Serial Universal

Hz_Herthz.

BCD_Binary Coded Decimal.

SW_Switch.

CLK_Clock.

PWM_Pulse Wide Modulation.

IEEE_Institute of Electrical and Electronics Engineers.

CB_Communication Block.

SB_Switch Block

MSB_Most Significant Bit.

LSB_Least Significant Bit.

SIGNED_Con signo

USIGNED_Sin signo

U/D_Up/Down

PULL UP_Resistencia Conectada a Vcc

DIP SW_Grupo de interruptores

CLB_Conjunto de elementos logicos

Z_Estado de Alta impedancia

U_Estado Desconocido

LCD_Liquid Crystal Diaplay

INTRODUCCION

En las diversas áreas del control de procesos, la incorporación de procesadores digitales lo han convertido en un componente fundamental para la realización de las diversas tareas requeridas, siendo la velocidad de procesamiento entre otros un bien muy preciado en el mundo de hoy. Muchas son las alternativas disponibles actualmente, sin embargo, la búsqueda de nuevas tecnologías no cesa y una de las aún no muy difundidas son los FPGA. El presente trabajo tiene como objetivo principal, entregar un material que sirva de apoyo a los estudiantes de la especialidad, interesados en aprender a programar e implementar proyectos con FPGA. Es de vital importancia comenzar a comprender que esta tecnología es un concepto totalmente diferente a los sistemas embebidos tradicionales, en los cuales existe un microcontrolador con periféricos o una CPU con memoria y puertos I/O, donde se realiza un proceso de ir a buscar un código de instrucción a una posición de memoria, decodificarla y ejecutarla, en uno o más ciclos de instrucción. Hasta aquí nada nuevo, este proceso se ejecuta con un set de instrucciones perteneciente al microprocesador especifico. En los FPGA no existe nada de lo anterior, es decir no hay instrucciones en memoria, no hay una CPU que realice un trabajo según lo indicado en memoria, el concepto es diferente. Los FPGA son dispositivos con un hardware abierto, configurable a la medida del requerimiento del programador, todo a través de un lenguaje o código portable para cualquier dispositivo FPGA que soporte el software de desarrollo en el cual se trabaja.

DESCRIPCION DEL PROBLEMA

La aparición de los FPGA en el mercado ha generado la necesidad por parte de los especialistas en el área de la electrónica y control, en adquirir los conocimientos necesarios para trabajar y programar estos dispositivos. En la actualidad la información de los FPGA es muy dispersa, en consecuencia, requiere mucho tiempo para que un estudiante o especialista adquiera el conocimiento necesario para aplicarlo al mundo real, en tiempos acotados.

PROPUESTA DE SOLUCION

El presente trabajo, es un material de estudio que entregara el conocimiento necesario para programar y realizar aplicaciones prácticas con los FPGA, utilizando diversas herramientas de software y hardware.

OBJETIVOS GENERALES

Entregar los conocimientos necesarios, para que el lector pueda conocer y programar los dispositivos FPGA.

OBJETIVOS ESPECIFICOS

- Conocer los aspectos básicos de los dispositivos FPGA y las herramientas necesarias para la creación de proyectos.
- Conocer y trabajar con el lenguaje VHDL
- Conocer y trabajar con el entorno de desarrollo Quartus II.
- Realizar experiencias prácticas, utilizando los conocimientos Adquiridos en el presente trabajo.

RESTRICCION

- Se trabajará con versión gratuita (versión light) de Quartus II
- Las experiencias practicas se realizarán con tarjetas de bajo costo lo que podría ser una limitante, si el proyecto requiere muchos elementos lógicos.

CAPITULO 1_Marco teórico.

1.0 DISPOSITIVOS LOGICOS PROGRAMABLES CPLD y FPGA

Para comprender la naturaleza de los FPGA, cuyo acrónimo es Field Programable Gate Array (campo de arreglo de compuertas programables) y poder visualizar su entorno, es importante identificar la familia de estos dispositivos lógicos programables. Un dispositivo lógico programable PLD (programable logic device), es un dispositivo electrónico en el cual se puede implementar un diseño para crear un circuito lógico, reconfigurable cuantas veces sea necesario. Estos PLD le entrega al diseñador una gran flexibilidad para crear una variedad de circuitos electrónicos. Los PDL se dividen en dos grandes grupos: En el primer grupo se encuentran los SPDL, que cumplen una función específica y que son programados una única vez. En el segundo grupo se agrupan todos aquellos que son programados por el usuario, la veces que sea necesario. En el segundo grupo, los HDPLD (High density programmable logic device), se clasifican en CPLD y FPGA. Como una primera aproximación, el presente trabajo se centrará en el segundo grupo. En la figura 1-1 se aprecia una clasificación general de los dispositivos lógicos programables.

1.1 DISPOSITIVOS PROGRAMABLES SIMPLES

Los dispositivos programables simples (SPLD), son básicamente una matriz de compuertas lógicas, las cuales ascienden a centenar y tiene la capacidad de almacenar la configuración entregada por el diseñador. Poseen un bajo número de conexiones, alta velocidad de respuesta y bajo consumo de energía. De este subgrupo se desprenden dos tipos, las PLA (Programable Logic Array), con tecnología bipolar y las PAL (Programable Array Logic), con tecnología CMOS.

1.2 DISPOSITIVOS LOGICOS PROGRAMABLES DE ALTA DENSIDAD

Los HDLPD, están formados por arreglos de compuertas lógicas de alta densidad agrupadas en bloques (elementos lógicos) con una gran flexibilidad. Este tipo de dispositivo le entrega una gran flexibilidad al diseñador para realizar los proyectos. Estos bloques lógicos, están formados por compuertas lógicas básicas.

El presente documento basa todo el trabajo con dispositivos perteneciente a este subgrupo, los cuales son CPLD y los FPGA.

1.3 DISPOSITIVOS LOGICOS PROGRAMABLES COMPLEJOS (CPLD)

Son dispositivos con un funcionamiento similar a una memoria EEPROM, es decir son de lógica no volátil. Son de bajo costo y tienen una cantidad de elementos lógicos inferior a un FPGA, sin embargo, estos dispositivos son ideales para proyectos con baja demanda de elementos lógicos

1.4 FPGA

Un FPGA, tal como se menciona al comienzo de este capítulo, es una matriz de compuertas lógicas, similar a los CPLD, pero con mayor cantidad de elementos lógicos, lo que permite sintetizar una mayor cantidad de circuitos complejos, con un mayor costo.

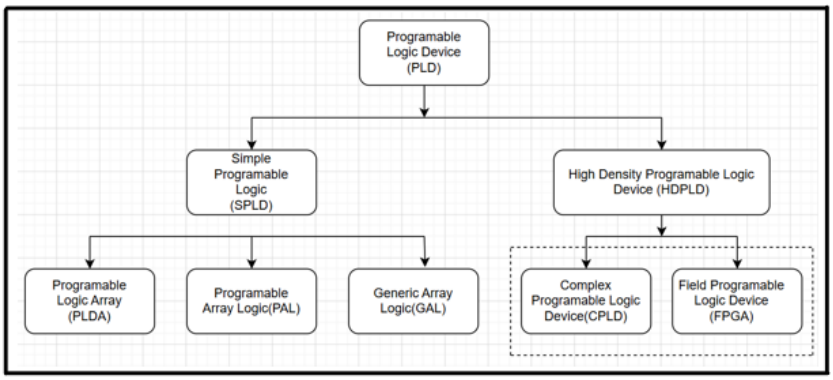


Figura 1-1_ Clasificación general de dispositivos lógicos programables.

En la figura 1-1 se visualiza la clasificación general de los dispositivos lógicos programables.

Una de las principales características de los FPGA, es su gran velocidad para ejecutar los procesos requeridos. En la figura 1-2 aprecia un esquema jerárquico en el cual se entrega una idea de diversos sistemas embebidos con respecto a la velocidad de procesamiento, utilizando diversos dispositivos programables. En un microcontrolador común se escribe un programa basado a un set de instrucciones acorde con el hardware del microcontrolador. Caso contrario en un FPGA, se escribe un código que arma un hardware dentro del FPGA.



Figura 1-2_ Jerarquía de sistemas embebidos, según su velocidad.

A modo de introducción en este trabajo, se requiere cierto cuidado al hablar de procesamiento de datos, compilación, programación entre otros términos y conceptos usados en la programación. Los FPGA y toda su familia no son dispositivos programables sino configurables. Específicamente los CPLD y FPGA

1.5 ARQUITECTURA INTERNA DE UN DISPOSITIVO FPGA

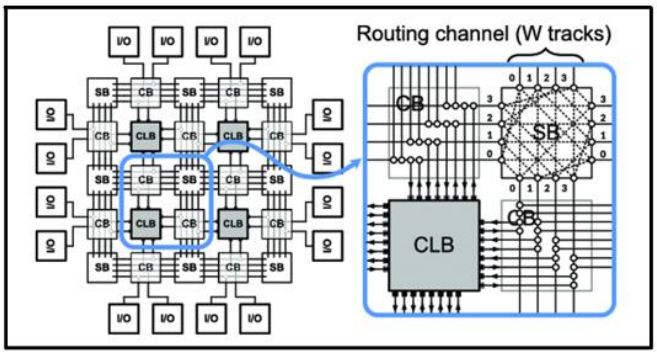


Figura 1-3 _Arquitectura interna de un dispositivo FPGA.

La figura 1-3 muestra la arquitectura de un FPGA, donde se aprecia los CLB que son los elementos lógicos, Configurable Logic Block´s, además están los SB o switch lógicos que son los encargados de gestionar la comunicación entre bloques, CB son los bloques de comunicación y los puertos I/O encargados de comunicar con el mundo exterior. Se observa que la arquitectura es totalmente abierta, es decir el lenguaje descriptor de hardware se encarga de construir el hardware requerido, utilizando los elementos lógicos que sean necesarios.

1.6 HERRAMIENTAS NECESARIAS PARA TRABAJAR CON FPGA.

Para trabajar con los FPGA se requieren algunos elementos tal como lo indica la figura 1.4. El primero es un lenguaje que permita generar los códigos que posteriormente serán cargados en el FPGA, en este trabajo se utilizó el lenguaje VHDL. En segundo lugar, un ambiente de desarrollo que permita, trabajar con el lenguaje VHDL para analizar, sintetizar (se asemeja a compilar), simular y cargar el código en la placa. El software Quartus II, 17.1 de la compañía ALTERA fue el elegido para este trabajo. Para la realización física de los diseños, se requiere un programador llamado USB Blaster y finalmente una placa de desarrollo llamada MAX-10 con el FPGA 10M08CSE144CG8, la cual tiene implementado algunos led's, display, botoneras entre otros. En el apéndice A se encuentra el catálogo de la placa que contiene los pines de entrada y salida asociados a dipw switch, display y otros puertos. Un punto muy importante es que todos los ensayos aquí descritos, pueden ser implementados en otra placa de desarrollo que contenga un dispositivo FPGA o CPLD perteneciente a la familia de Altera.

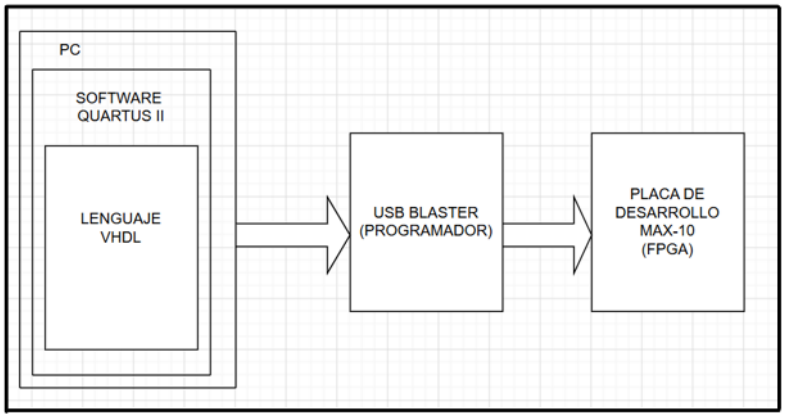


Figura 1-4_Elementos requeridos para programar CPLD y FPGA

CAPITULO 2- LENGUAJE VHDL

2.1 ASPECTOS BASICOS

Como se ha mencionado anteriormente, los dispositivos FPGA son capaces de emular circuitos lógicos complejos a través de sus matrices de compuertas o elementos lógicos, que se interconectan entre ellos, utilizando para esto el lenguaje de programación descriptivo que permita establecer el comportamiento del circuito lógico a implementar.

VHDL es una combinación de dos acrónimos: VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language), es uno de los tantos lenguajes de descripción de hardware, el cual describe el funcionamiento y la estructura de un circuito lógico que será posteriormente cargado en un FPGA. En el presente trabajo todos los programas y ensayos serán implementados en VHDL, con un detallado análisis, que permita una comprensión de esta tecnología.

En una primera aproximación se puede afirmar que VHDL permite la síntesis de circuitos (lo que en lenguajes de programación convencionales se conoce como compilación) así como la simulación de circuitos. La síntesis es la traducción de un código fuente a una estructura de hardware que implementa la funcionalidad deseada, mientras que la simulación es un procedimiento opcional de prueba, para asegurar la funcionalidad del circuito deseado.

Anteriormente se mencionó que VHDL se utiliza para describir el funcionamiento de circuitos digitales por lo que inevitablemente se trabaja con señales digitales; las cuales solo pueden existir dos posibles valores: 1 y 0 lógico.

VHDL permite también ejecutar operaciones aritméticas (ejem. sumar, restar, dividir y multiplicar), es capaz de sintetizar números enteros y con punto flotante usando los archivos de biblioteca adecuadas. Es importante tener presente que, a mayor complejidad en las operaciones matemáticas, mayor será el 'consumo' de elementos lógicos disponibles en el dispositivo FPGA a programar.

Debido a su naturaleza descriptiva, es posible ejecutar estructuras lógicas usadas en otros lenguajes de programación (orientados a objetos) tales como *if/else y case,* así como estructuras cíclicas como *for* y *while*, sumado a lo anterior tiene la capacidad de declarar funciones y ser llamadas en cualquier parte del código.

Para trabajar en VHDL y poder analizar, sintetizar, simular y programar un dispositivo FPGA, se utilizará una herramienta llamada Quartus II, versión 17.1. Este entorno de desarrollo ofrece muchos recursos entre ellos, el poder trabajar en diagramas de bloques (modo arquitectura), que ayudara a integrar e interconectar diferentes programas en forma gráfica, para consolidar un proyecto global. En el siguiente capitulo revisará en detalle Quartus II

2.2 ESTRUCTURA DE VHDL

El lenguaje de descripción de hardware está estructurado en módulos o unidades funcionales, que se identifican a través de palabras reservadas. Cada módulo tiene una secuencia de sentencias e instrucciones las cuales, con las

declaraciones de las unidades en el programa, permiten la descripción, comprensión, evaluación y solución de un sistema digital. La estructura fundamental del lenguaje VHDL, es decir los pilares que conforman un programa son tres: Los archivos de biblioteca, la entidad y la arquitectura. En la figura 2.1 se aprecia un formato de la estructura mínima requerida para un programa en VHDL.

```
1  --esto es comentario y no afecta al codigo VHDL
2  --Estructura basica de un programa VHDL
3
4   library ieee;                       --modulo librerias
5   use ieee.std_logic_1164.all;
6  entity  nombre_entidad  is
7  port (
8
9                                       -- modulo entidad
10
11
12        );
13 end entity;
14 architecture nombre_arquitectura  of nombre_identidad is
15 begin
16
17                                      --modulo arquitectura
18
19
20 end architecture;
21 --como veremos mas adelante, el nombre del archivo debe
22 --ser el mismo al de la entidad-
23  --m.m.a.
```

Figura 2-1_Estructura básica de un código VHDL.

2.21 ARCHIVOS DE BIBLIOTECA

Es considerado como el primer módulo o pilar fundamental de un programa VHDL y consiste en la declaración de los archivos de bibliotecas que habilitan el uso de tipos de datos, operadores y funciones. En VHDL son muy usados los paquetes de biblioteca IEEE, los cuales poseen archivos de código que se declaran de manera individual para usar su contenido en el código principal. Existen diversos paquetes para diferentes propósitos por lo que se considera para este trabajo

usar tres paquetes que cubren lo necesario para el funcionamiento de las actividades propuestas a realizar. Sin embargo, es importante indicar que existen otros paquetes no oficiales IEEE, que funcionan perfectamente y pueden agilizar algunas soluciones. Es de vital importancia mencionar que VHDL tiene sus propios paquetes estándar, sin necesidad de declararlos, sin embargo, tienen sus limitaciones, siendo necesario agregar en forma adicional el paquete IEEE.

2.211 PAQUETE STD_LOGIC_1164

Contiene el tipo de variable **STD_LOGIC** el cual permite declarar números de un solo bit. También incluye la variante **STD_LOGIC_VECTOR** que son arreglos o vectores de bits que permiten representar números de dos o más bits. El paquete define los valores que representan los posibles estados lógicos: “1” (alto) y “0” (bajo). El paquete permite sintetizar otros estados lógicos, como “u” (no asignado), “z” (alta impedancia) y “-” (no importa el valor).

2.212 PAQUETE NUMERIC_STD

Este paquete permite manipular números sin signo **(**unsigned), que se representan como arreglos de bits similar a los **STD_LOGIC_VECTOR**, y con signo (**SIGNED)** que, usando el método de complemento a 2, usa el bit más significativo **(MSB)** al arreglo para definir si el número es positivo o negativo. Además, permite manipular números naturales y números enteros (con y sin signo). Este paquete también contiene a los operadores aritméticos (suma, resta, multiplicación y división) para realizar operaciones con números enteros, además se pueden escribir en notación científica y ejecutar funciones que puedan realizar la conversión los **STD_LOGIG_VECTOR** a números enteros y viceversa.

2.213 PAQUETE MATH_REAL

Este paquete trabaja de manera similar a **NUMERIC_STD** con la diferencia que trabaja con números con punto flotante, incluye constantes matemáticas tales como el número Pi y la constante de Euler entre otras. El paquete contiene las operaciones matemáticas básicas (suma, resta multiplicación y división) para operar con números flotantes y añadiendo operaciones matemáticas avanzadas como potencias, raíces y funciones trigonométricas.

2.214 DECLARACION DE LOS ARCHIVOS DE BIBLIOTECAS Y PAQUETES.

Para declarar un archivo de biblioteca primero debe escribir en las primeras líneas de código siguiendo la siguiente sintaxis:

library [library_name]; (1)

Mientras que para declarar los paquetes de los archivos de biblioteca se usa la siguiente estructura:

use [library_name].[package_name].all; (2, 3 y 4)

En el siguiente ejemplo se muestra la declaración del archivo de biblioteca **ieee** y los paquetes mencionados anteriormente, según los requerimientos del programador:

library ieee; **(1)**

use ieee.std_logic_1164.all; **(2)**

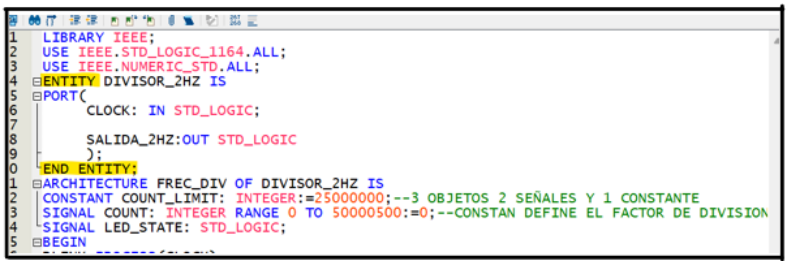
use ieee.numeric_std.all; **(3)**

use ieee.math_real.all; **(4)**

2.22 ESTRUCTURA ENTITY (entidad)

La estructura entity (entidad), representa las entradas y salidas de los procesos que se realizan internamente. La entidad es referida como un puerto de entrada y salida. Cada puerto tiene un nombre, un modo y tipo de dato y se declaran previamente. El nombre de cada puerto, el programador lo define con un identificador. El modo define si el puerto es de entrada in, salida out o bidireccional inout. Finalmente, el tipo de datos define si es un bit (std_logic), un grupo de bits (std_logic_vector) u otro tipo de datos.

En la figura 2.2 se muestra el comienzo de un programa ensayado previamente. Entre las líneas 1 y 3 se ubica las librerías a utilizar en el programa. La línea 2 tiene las librerías de la lógica binaria y la línea 3 tiene todas las librerías numéricas. En la línea 4 comienza la entidad y se declara el nombre 'DIVISOR_2HZ'. Entre las líneas 5 y 9, se declaran los puertos con los nombres, CLOCK y SALIDA_2HZ, los modos in y out respectivamente y los tipos de datos, ambos datos de 1 bit es decir std_logic**.** En la línea 10 finaliza la entidad.



```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4  ENTITY DIVISOR_2HZ IS
5  PORT(
6        CLOCK: IN STD_LOGIC;
7
8        SALIDA_2HZ:OUT STD_LOGIC
9        );
0  END ENTITY;
1  ARCHITECTURE FREC_DIV OF DIVISOR_2HZ IS
2  CONSTANT COUNT_LIMIT: INTEGER:=25000000;--3 OBJETOS 2 SEÑALES Y 1 CONSTANTE
3  SIGNAL COUNT: INTEGER RANGE 0 TO 50000500:=0;--CONSTAN DEFINE EL FACTOR DE DIVISION
4  SIGNAL LED_STATE: STD_LOGIC;
5  BEGIN
```

Figura 2-2_Entidad en código VHDL.

2.23 ESTRUCTURA ARCHITECTURE

Architecture es la sección del código, que describe el funcionamiento del circuito lógico propiamente a utilizar. En la figura 2-3, en la línea 12 comienza la arquitectura, **comp** es el nombre de la arquitectura y **decod4_7seg** es el nombre de la entidad y el archivo (archivo e identidad llevan el mismo nombre), ambos son identificadores definidos por el programador. La línea 13, Begin indica que comienza el proceso y termina en la línea 27 con la sentencia end architecture.

```
11  end entity;
12  architecture comp of deco4_7seg is
13  begin
14      with SBIT select
15      SEG<= "0000001" when "0000",
16            "1001111" when "0001",
17            "0010010" when "0010",
18            "0000110" when "0011",
19            "1001100" when "0100",
20            "0100100" when "0101",
21            "1100000" when "0110",
22            "0001111" when "0111",
23            "0000000" when "1000",
24            "0001100" when "1001",
25            "ZZZZZZZ" when  OTHERS;
26       S_D<="0111";
27      end architecture;
```

Figura 2-3_Arquitectura de un código VHDL.

La sección declarativa (situada entre architecture y begin), entre la linea 12 y 13 de la figura 2-3, se puede insertar de manera opcional los objetos (señales, variables o constantes) según la necesidad del programador.

En resumen, en el hardware a diseñar sus entradas y salida son la entidad (entity), es decir la interfaz con el mundo exterior. Por otra parte, lo que está dentro del hardware será la arquitectura (architecture), aquí es donde se realizan todos los procesos necesarios para configurar el hardware requerido.

2.3 EL PRIMER PROGRAMA

Hasta el momento se han revisado los elementos básicos de un programa en VHDL, entidad, arquitectura y librerías. A continuación, se visualiza un primer programa muy elemental que realizara la función básica **or** dentro de la arquitectura mostrada en la figura 2-4.

```
1  --ESTO ES UN COMENTARIO
2  --PROBAMOS NUESTRO PRIMER PROGRAMA
3   library ieee;                      --LIBRERIA STANDAR A USAR SIEMPRE
4   use ieee.std_logic_1164.all;       --LIBRERIA USADA EN OPERACIONES LOGICAS
5  entity  compuerta  is
6  port (
7         A: in std_logic;            --ENTIDAD
8         B: in std_logic;
9         C: in std_logic;
10        salida: out std_logic
11        );
12 end entity;
13 architecture NOMBRE_ARQUITECTURA of compuerta is
14 begin
15 salida <= A or B or C;      --aqui se realiza el proceso
16  end architecture;
```

Figura 2-4_El primer programa VHDL.

En la figura 2-4, las líneas 1 y 2 son comentarios indicados con dos guiones, las líneas 3 y 4 son las librerías por utilizar (en la mayoría de las ocasiones se usan). La librería **ieee.std_logic_1164.all** alberga todas las funciones lógicas, para todo tipo de datos binarios. En línea 5 comienza la entidad (entity), con el nombre de la entidad llamada **compuerta** (un identificador), línea siguiente comienza la declaración de los puertos a, b c en el modo de entrada **in, salida** en línea 10 en modo salida **out**, todos ellos con datos de tipo **std_logic,** es decir bit's(datos de un solo bit). El término de la entidad se visualiza en la línea 12, dando comienzo en la línea 13 al módulo de arquitectura, donde se realiza el proceso de la operación **or**, propiamente tal (línea 15). Begin indica el comienzo del proceso que termina en la línea 16 junto al programa. Un detalle muy importante que tener en cuenta es que la última puerta de entrada/salida de la identidad declarada no termina en punto y coma (ver línea 10).

2-4 ELEMENTOS LEXICOS VHDL

Léxico se conoce como el vocabulario propio de un idioma. En los lenguajes de programación se conoce como las unidades básicas de ellos.

2.41 COMENTARIOS

Un comentario comienza con dos guiones seguidos y lo escrito posterior a estos guiones, no tiene ningún efecto en el código. El objetivo es solo para documentar algún dato al programa, tal como en la figura 2.3 en el programa ‘compuerta’.

2.42 IDENTIFICADORES

Son los nombres o etiquetas que se usan para hacer referencia a las variables, constantes, señales, procesos, nombres de archivos y proyectos entre otros. Pueden ser letras mayúsculas y minúsculas, sin distinción entre ellas, guion bajo y números. No tienen restricción de longitud y no pueden comenzar con un número. Hay que recordar que no permite ningún símbolo, salvo el guion bajo (_).

Los identificadores en VHDL deben cumplir ciertas condiciones para su validez, que se detallan a continuación:

El primer carácter debe ser una letra, mayúscula o minúscula.

El segundo carácter no puede ser guion bajo.

No permite guiones juntos.

No puede usar ningún tipo de símbolo.

Algunos ejemplos:

5suma => No permitido	Suma5 => permitido
S_5bits => No permitido	S5_bits=> permitido
Resta__5=> No permitido	Resta_5_ =>permitido
Clear#2 => No permitido	Clear_2 =>permitido

2.5 TIPOS DE DATOS

En VHDL existen dos tipos de librerías, la estándar y la correspondiente a la IEEE. La librería estándar va inserta en VHDL, no siendo necesario declarar previamente dichas librerías, a diferencia de los paquetes IEEE que deben ser declarados previamente. Existe una diversidad de tipos de datos en VHDL, donde cada uno cumple una función específica que permiten al diseñador optimizar el proyecto. Para usarlos basta con declarar los archivos de bibliotecas (los pertenecientes a IEEE) y los paquetes que permitan usarlos, y después declararlos según sea el caso tanto dentro de la entidad y/o arquitectura. Diversos son los tipos de datos que revisaran en este texto y que se detallan a continuación:

2.51 BITS

Entrega 2 valores que son ‘0’ y ‘1’ lógico. Corresponde a los paquetes estándar.

2.52 BIT_VECTOR

Es un vector de bits, es decir es un grupo de bits usado en un bus de datos, pertenecientes a los paquetes estándar.

2.53 STD_LOGIC

Proviene del standart IEEE **std_logic_1164**, que a diferencia de los datos del tipo Bit (0 y 1), este standart agrega los siguientes tipos de datos:

- 0 y 1: Consiste en forzar a cero lógicos (0) y forzar a uno lógico (1)

-Z: Significa estado de alta impedancia, normalmente asociado a un buffer de tres Estados.

- H: Estado 1 lógico débil.

- L: Estado 0 lógico débil.

- ‘X’ y ‘W’: Se indica como una señal desconocida (x) y desconocida débil (w), que corresponde a un valor intermedio entre 0 y 1 lógico.

- ‘-‘: El guion indica un valor o estado que no importa.

- ‘U’: Significa no inicializado y es usado en simulación.

Para declarar este tipo de dato, se usa la sintaxis:

[variable_type] [name] : std_logic := '[bit_value]';

Ejemplo:
signal my_bit : std_logic:= '0';

2.54 STD_LOGIC_VECTOR

Trabaja de manera similar **std_logic** con la diferencia que alberga un vector de bits y es similar a los arreglos en otros lenguajes de programación. Puede representar buses de datos de más de dos bits, así como números binarios. La sintaxis para declararlos es la siguiente:

[variable_type] [name] : **std_logic_vector**([size **- 1**] **downto 0**):= "[binary_value]";

En el ejemplo, la variable **“PORT_A”** contiene un valor **std_logic_vector** que representa un bus de 8 bits:

signal PORT_A: **std_logic_vector** (**7 downto 0**): = "10101111";

Aquí el bit MSB a la izquierda y LSB a la derecha.

signal PORT_A: **std_logic_vector** (**0 to 7**): = "11110101";

Aquí el MSB va a la derecha y el LSB a la izquierda (**downto** y **to** hacen la diferencia).

A continuación, se muestra una forma típica de acceder a un bit específico de la variable "**PORT_A**" utilizada anteriormente, se asigna a **"my_bit"** declarada como **STD_LOGIC**. Se quiere acceder al bit situado en la posición 3 (contando de derecha a izquierda contando desde 0) y el valor obtenido es 1.

my_bit <= **PORT_A** (3);

En ocasiones se requiere obtener un conjunto de bits de un **STD_LOGIC_VECTOR**. En el siguiente ejemplo **"PORT_B"** se ha declarado como un **STD_LOGIC_VECTOR** de 4 bits y el valor asignado corresponde a los bits de las posiciones 7, 6, 5 y 4 de la variable **"PORT_A"** y el resultado es el número binario "1010".

PORT_B<= **PORT_A** (7 downto 4); -- transfiere los 4 bits MSB de PORT_A

-- hacia PORT_B (comentario)

2.55 SIGNED / UNSIGNED

Los tipos de dato con UNSIGNED trabajan de manera similar a los **std_logic_vector** debido a que representan números binarios con más de dos bits mientras que los tipos SIGNED representan números binarios con signo donde, de acuerdo con el método de complemento a 2, el bit más representativo indica el signo del número.

En el siguiente ejemplo en las variables "num_signed" y "num_unsigned" se declaran números con y sin signo de 4 bits respectivamente:

signal num_signed : signed(**3 downto 0**) := "1011" ;

signal num_unsigned : unsigned(**3 downto 0**): = "0011";

Notar un detalle muy importante, la variable “num_signed” almacena el número binario de 1011 que, de acuerdo con el método de complemento a 2, equivale al valor numérico -3 mientras que para la variable “num_unsigned” representa el número 11.

2.56 NATURAL

Este tipo de dato almacena y permite operar los números naturales. Estos números son enteros positivos, cuyo rango por defecto va de 0 a 2.147.483.645 es decir, utiliza el estándar de un numero de 32 bit, pero sin signo (0 a 2 elevado a 31-1). La forma de declarar un identificador con un tipo de numero natural es el siguiente:

[**variable_type]** [name] :natural := [value];
Por ejemplo:
signal natural_name : natural := 32; -- el identificador “natural_name” vale 32
-- y el tipo de dato es natural (comentario).

Se debe tener presente que la declaración anterior, no tiene rango definido, en consecuencia, el numero 32 utilizara 31 bits, lo que no es recomendable por el uso excesivo de elementos lógicos a momento de sintetizar el programa. Para evitar lo anterior se limita el rango de la siguiente forma:

signal natural_name: natural range 0 to 32: = 0;

2.57 INTEGER

Este tipo de dato permite trabajar con el conjunto de los números enteros con signo, por defecto con 32bits y un rango desde -2.147.483.647 a +2.147.483.647, siendo recomendable establecer un rango para mayor eficiencia de los elementos lógicos en el dispositivo FPGA a momento de sintetizar un programa.

[variable_type] [name] : integer range 0 to value:= [value];

Ejemplo:

signal int_num : integer range -100 to 100 := -100;

2.58 REAL

En VHDL, los números reales se manejan mediante el tipo de dato nativo real, el cual representa números de punto flotante en un rango de -1.0 x 10^38 a +1.0 x10^38. Sin embargo, el tipo real **no es sintetizable** y solo se utiliza para simulación. En el presente trabajo no se prioriza trabajar con números reales.

2.59 STRING

Son tipos de datos que almacenan arreglos de caracteres. En VHDL, las cadenas se declaran con un número finito de caracteres y cuando se realiza la asignación de un valor, debe coincidir con el tamaño de la cadena asignado.

[variable_type] [name]: **string(1 to** [string_lenght]):= "[string]";

La asignación de la longitud de la cadena se declara de manera similar a los **STD_LOGIC_VECTOR**, aunque en éste ultimo los bits se cuentan de derecha a izquierda iniciando desde 0, mientras que las cadenas los caracteres se cuentan de izquierda a derecha iniciando desde 1:

signal my_string: **string(1 to 9)**:= "HOLA FPGA"; --(el espacio es un carácter)

En este otro ejemplo se muestra cómo acceder a un carácter o una parte de la cadena de caracteres.

my_char <= my_string(2); -- entrega el segundo carácter “O”

string_2 <= my_string(4 to 8); -- entrega los caracteres “A FPG”

Donde el valor de “my_char” es el carácter “o”, siguiendo el conteo mencionado anteriormente. Mientras que para “string_2” el valor es una sección de la cadena “my_string” cuyo contenido es la cadena “A FPG”.

2.6 OBJETOS

Son elementos de datos que se utilizan para almacenar información durante la ejecución del circuito digital. En VHDL tres son los tipos de objetos más usados; señales, variables y constantes. Estos objetos su uso es opcional y se pueden requerir según la necesidad del proyecto. Los objetos se ubican inmediatamente después del comienzo de la arquitectura y finalizan justo antes de begin (línea 15). En la figura 2-5, entre las líneas 12 y 14 se ubican los objetos, en este caso 2 señales y una constante.

```
10  END ENTITY;
11  ARCHITECTURE FREC_DIV OF DIVISOR_2HZ IS
12  CONSTANT COUNT_LIMIT: INTEGER:=25000000;--3 OBJETOS 2 SEÑALES Y 1 CONSTANTE
13  SIGNAL COUNT: INTEGER RANGE 0 TO 50000500:=0;--CONSTAN DEFINE EL FACTOR DE DIVISION
14  SIGNAL LED_STATE: STD_LOGIC;
15  BEGIN
16  BLINK:PROCESS(CLOCK)
```

Figura 2-5_ Objetos, línea 12 una constante, líneas 13 y 14 señales

2.61 SIGNAL

Este tipo de objeto tiene el propósito de representar las señales físicas internas del circuito que se está programando, aunque también pueden realizar conexiones entre otras señales internas. En las siguientes líneas se muestran algunos tipos de datos declarados como señales internas, así como también en las líneas 13 y 14 de la figura 2-4. A continuación dos ejemplos:

```
signal mi_senal1: std_logic:= '0';
signal mi_senal2: std_logic_vector(3 downto 0);
```

2.62 VARIABLE

Se usan para el almacenamiento local de datos que se pueden almacenar y modificar su valor, dentro de un proceso. Su valor es válido en el momento de su asignación. En otras palabras, para VHDL es una memoria simbólica, es decir no existe un mapeo entre la variable y el hardware.

Son similares a **signal** y se declaran de igual forma forma:

```
variable my_variable_1 : std_logic := '0';
variable my_variable_2 : std_logic_vector(3 downto 0);
```

Su funcionamiento es casi idéntico con la gran diferencia de que **signal** se declara al inicio de la architecture lo que da el rol de una variable **global,** mientras que **variable** se declara dentro de la unidad secuencial en la cual se requiere su uso, por lo tanto, una variable es de uso local. En otras palabras, una variable es válida dentro de un proceso determinado. Mas adelante se explicará cómo trabaja un proceso (process) y se observará detalle la diferencia de una variable y una señal (signal).

2.63 CONSTANT

Un tipo de objeto **constant** almacena un valor de cualquier tipo de dato, no se permite el cambio del valor, por lo que siempre será el mismo en cualquier parte del código. Para declarar una constante, se llama a la palabra reservada **constant,** seguido del nombre que asigna a esta constante, posteriormente se indica el tipo de dato, para finalmente agregar el valor asignado. Es indispensable que exista un valor por defecto asignado, a diferencia de los otros objetos que es opcional.

```
constant my_constant_1: std_logic:= '1';
constant my_constant_2: std_logic_vector(3 downto 0): = "1101";
constant my_constant_3: integer := 100;
```

En la figura 2-6 se aprecia en la línea 12 la forma de declarar este objeto.

```
11  ARCHITECTURE FREC_DIV OF DIVISOR_2HZ IS
12    CONSTANT COUNT_LIMIT: INTEGER:=25000000;--3 OBJETOS 2 SEÑALES Y 1 CONSTANTE
13    SIGNAL COUNT: INTEGER RANGE 0 TO 50000500:=0;--CONSTAN DEFINE EL FACTOR DE DIVISION
```

Figura 2-6_ Declaración del objeto constant.

2.7 OPERADORES ARITMETICOS

En VHDL se pueden encontrar diversos operadores que realizan operaciones con los tipos de datos mencionados en la sección anterior.

2.71OPERADORES ARITMÉTICOS CON NÚMEROS ENTEROS.

Estos operadores permiten realizar operaciones matemáticas con los números enteros cuyo resultado son números enteros. Para hacer uso de ellos, se tiene que haber declarado antes el paquete **numeric_std.** En los siguientes ejemplos se muestra la forma en cómo se escriben los operadores matemáticos en VHDL y su resultado mostrado en el comentario.

Suma

a <= **10** + **6**; -- a = 16 (no olvidar que después de -- es un comentario)

Resta

b <= **6** - **18**; -- b = -12

Multiplicación

c <= -**5** * **9**; -- c = -45

División

d <= **5** / **2**; -- d = 2

e <= **100** / **3**; -- e = 33

Para este caso, la división solo puede regresar números enteros, por lo que, si matemáticamente el resultado es un número con punto flotante, el operador truncará el resultado.

Módulo

f <= 123 mod 10; -- f = 3;

Potencia

g <= 4 ** 5; -- g = 4^5 = 1024

Notación científica h <= **14**e3; -- d = 14*10^3 = 14000

2.72 OPERADORES ARITMÉTICOS DE NÚMEROS CON PUNTO FLOTANTE

Funcionan de igual forma que los mencionados en el apartado anterior, con la diferencia que el resultado de estos operadores también es un número con punto flotante y pueden operar operaciones matemáticas complejas como las funciones trigonométricas, exponentes y logaritmos. Es importante mencionar que, si el número es entero, pero se quiere operar con números con punto flotante, se debe agregar el punto seguido de un cero (ej. Si es el numero entero 2, en números con punto flotante se escribe 2.0).

Suma

a <= 4.5 + 10.0; -- a = 14.5

Resta

b <= 2.78 - 41.688; -- b = -38.908

Multiplicación

c <= 100.0 * 0.333; -- c = 33.3

División

d <= 5.0 / 151.3; -- d = 0.0330469

Exponente

e <= 5.6 ** 2.1; -- e = 37.2558826

Raíz cuadrada

f <= sqrt(0.56); -- f = 0.74833

Raíz cúbica

g <= cbrt(27.0); -- g = 3.0

Notación científica

h <=1.5153e-4; -- h = 0.00015153

Exponencial base e

i <= exp(3.25); -- i = 25.79034

Logaritmo natural

j <= log(12.69) -- j = 2.5408

Logaritmo base 10

k <= log10(156.4) -- k = 2.194237

Seno

l <= sin(2.36) -- l = 0.041178

Coseno

m <= cos(-2.36) -- m = 0.9991518

Tangente

n <= tan(0.423) -- n = 0.007383

Notación científica

h <=1.5153e-4; -- h = 0.00015153

Aproximar

p <= round(1.4896) -- p = 1.0

q <= round(-3.713) -- q = 4.0

2.8 OPERADORES COMPARATIVOS

Realizan las comparaciones matemáticas conocidas, pueden tener argumentos tanto números enteros como números flotantes y el resultado es un booleano que

puede ser true (verdadero) o false (falso). A continuación, se muestra el símbolo asignado a cada operador de comparación.

```
A = B          -- A igual a B
A /= B         -- A diferente de B
A < B          -- A menor que B
A > B          -- A mayor que B
A <= B         -- A menor o igual a B
A >= B         -- A mayor o igual a B
```

En el siguiente ejemplo se ilustra que la variable “condition” ha sido declarado variable booleana que solo guarda los valores ture o false.

```
condition <= 20 < 7;   false
condition <= 0.561 >= 0;  true
```

En la primera línea se obtiene que la condición es falsa ya que 20 no es mayor a 7, mientras que en la segunda línea la condición es verdadera porque 0.561 es mayor a cero.

2.9 OPERADORES LOGICOS

Los operadores lógicos son muy similares a los operadores comparativos del apartado anterior. Existe una diferencia entre ambos, los operadores comparativos trabajan con valores numéricos y devuelven un dato de tipo

booleano, mientras que los operadores lógicos trabajan con el tipo de dato STD_LOGIC. Por este motivo, los operadores lógicos se consideran equivalentes a las compuertas lógicas usadas en los circuitos digitales. En la tabla 2-0 se observa el código VHDL correspondiente al tipo de compuerta lógica.

CODIGO EN VHDL	TIPO DE COMPUERTA
not A	NOT
A and B	AND
A or B	OR
A xor B	XOR
A nand B	NAND

Tabla 2-1_ Código VHDL correspondiente al tipo de compuerta.

El siguiente ejemplo es parte de un código VHDL en el cual C es una variable correspondiente a una operación booleana, representada por compuertas lógicas.

```
C<= not((A and B) xor (A or B));      -- si A=1 y B=0, entonces C=1.
```

2.10 CIRCUITOS COMBIANCIONALES Y SECUENCIALES EN VHDL

Para el diseño de hardware con VHDL, es importante separar los dos tipos de hardware a revisar en este trabajo. El primer tipo son los circuitos combinacionales y en segundo término los circuitos secuenciales.

2.11 CIRCUITOS COMBINACIONALES

Son sistemas digitales donde la salida depende únicamente de la combinación actual de las entradas, sin el uso de memoria. Sus principales aplicaciones son los multiplexores, sumadores y decodificadores, fundamentalmente para la conversión de códigos. La implementación física se realiza con compuertas lógicas, sus operadores lógicos y su función se describe con tablas de verdad y el algebra de Boole.

La implementación de estos circuitos combinacionales en los dispositivos FPGA se realiza a través del lenguaje de descripción de hardware VHDL, en cual se implementa un programa, se sintetiza, en forma opcional se puede simular y finalmente una vez compilado el proyecto se carga en el dispositivo FPGA. En el siguiente capitulo se analizará en detalle el software Quartus II, versión 17.1.

Para comenzar con el diseño de circuitos combinacionales, dado el siguiente circuito mostrado en la figura 2-7, se describe el código en VHDL, mostrado en la figura 2-8.

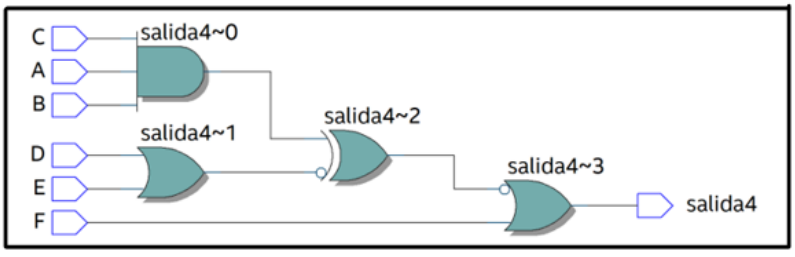


Figura 2-7_Circuito combinacional requerido para escribir el código VHDL.

Con los conocimientos adquiridos en el “primer programa”, en las primeras líneas se identifican las librerías necesarias, en este caso se utilizó **std_logic** (operaciones con un bit), que es parte de **ieee.std_logic_1164.all**. En la línea 5 comienza la entidad y en ella se declaran todos los puertos de entradas, es decir, a, b, c, d, e y f y la única salida salida4, todos en un bit. En la línea 17 comienza la arquitectura, es decir el corazón del proceso, para llegar a la línea 19 donde comienza el código y operadores lógicas para describir el circuito antes mencionado. Nótese que la estructura básica de escribir código en VHDL siempre es la misma, es decir las librerías, la entidad y la arquitectura.

```
1  --probamos con 4 compuertas
2  library ieee;
3  use ieee.std_logic_1164.all;
4  entity  compuerta4 is
5  port (
6        A: in std_logic;
7        B: in std_logic;
8        C: in std_logic;
9        D: in std_logic;
10       E: in std_logic;
11       F: in std_logic;
12
13
14       salida4: out std_logic
15       );
16 end entity;
17 architecture com of compuerta4 is
18 begin
19 salida4 <=   (f or(not((a and b and c) xor(not( D or E)))));
20
21 end architecture;
```

Figura 2-8_ Código VHDL correspondiente al hardware de la figura 2-7.

2.111 DECLARACION CONCURRENTE SELECTIVA WITH- SELEC- WHEN

La declaración **with-select-when** es fundamental para trabajar en circuitos combinacionales, tanto en multiplexores y decodificadores. Se utiliza para obtener una o más salidas, cuyo valor previamente establecido, sea seleccionado a través de otro valor. Estos valores pueden ser tanto en forma de bit o vectores de bit. El siguiente ejemplo ilustra en forma práctica a **with-select-when.**

```
With      identificador_entrada    Selec
Indentificador_salida <= ”0111” when “00”,  -- el identificador salida tomara
                         “1000” when “01”,  -- el valor “xxxx” cuando selec
                         “1001” when “10”,  -- valga “xx”.
                         “1010” when “11”;
```

La idea es la siguiente, cuando la palabra binaria de 2 bits (identificador de entrada) se encuentre “00”, la salida entregara el valor “0111”, caso contrario si toma el valor “01”, la salida tomara el valor “1000” y así sucesivamente hasta llegar a la salida “1010” una vez que nuestro seleccionador se encuentre en “11”. El texto seguido de dos guiones es un comentario que explica lo anterior. En los próximos capítulos se presenta este concepto con ejemplos prácticos tanto en un multiplexor y un decodificador de 7 segmentos.

2.12 CIRCUITOS SECUENCIALES

Los circuitos digitales secuenciales son sistemas digitales, en los cuales la salida depende de las entradas actuales y a estados pasados. Aquí entra el concepto de memoria (latches y flip-flop’s), que almacenan datos de eventos pasados.

Para trabajar y diseñar con circuitos secuenciales usando VHDL, es fundamental trabajar con los llamados “**procesos”,** los cuales se encuentran dentro de la arquitectura. Cada proceso representa una unidad secuencial que está sujeta al cambio de una o varias señales lógicas (flancos de subida o bajada) en los puertos de entrada de la entidad y cuando sucede el cambio por lo menos en una señal, se ejecuta el código contenido en el proceso.

Dentro de cada proceso, es posible trabajar con variables (objeto) las cuales son solo locales, es decir son válidas dentro de un determinado proceso.

La forma general de declarar un proceso dentro del módulo de arquitectura es como se indica a continuación.

La declaración de proceso comienza con la palabra clave **process** y la **lista de** sensibilidad son los eventos que activan el proceso, estos eventos son cambios de nivel o flancos de subida o bajada. En un proceso síncrono la lista de sensibilidad más utilizada es el reloj y señal de reset. En este trabajo, el reloj que lleva inserta la placa de desarrollo del FPGA será la principal lista de sensibilidad. El proceso es parte fundamental realizar divisores de frecuencia, para contar eventos en los circuitos secuenciales y funciones de temporización. Por otro lado, para la declaración de variables, son opcionales y locales, es decir, son válidas solo dentro de un determinado proceso. A continuación, se observa la forma general de hacer un process en VHDL.

```
PROCESS (LISTA_DE_SENSIBILAD) -- lista sensibilidad, clock, reset o un
                                -- puerto.
BEGIN                           --inicializa la tarea a realizar.
--codigo –                      --cuerpo del proceso, por lo general
                                -- es un bucle.
END PROCESS;                    --fin del proceso.
```

Como se indicó recientemente, la lista de sensibilidad funciona en base a cambios de nivel es decir flancos de subida o bajada y para diferenciar entre los tipos de flanco, se usan las siguientes palabras claves a escribir un código. **Rising_edge**, se asocia a un flanco de subida mientras que **Falling_edge** indica que es un flanco de bajada.

En la figura 2-9 se aprecia un código de ejemplo muy simple de un PROCESS. Como se aprecia, en la primera línea se comienza con la palabra process, seguido de la sensibilidad a revisar, para este caso es la señal de reloj (CLK).

Begin indica el comienzo del proceso. En este caso, la salida **'q'** tomara el valor **'d'** una vez que ocurra un flaco de subida (rising_edge) en la entrada de clock.

```
process (clk)
begin
  if rising_edge(clk) then
    q <= d;
  end if;
end process;
```

Figura 2-9_ Código básico de un proceso.

2.121 ESTRUCTURA DE CONTROL IF-ELSIF-ELSE

En VHDL, la estructura de control **if-elseif-else**, es una de las estructuras de ciclo más utilizadas y es similar a las existentes en otros lenguajes de programación. Utilizada en diferentes bloques de códigos basándose en una o más condiciones.

If: Evalúa la condición. Si la condición es verdadera, ejecuta el siguiente código.

Elseif: es una abreviatura de "Else if". Evalúa otra condición si la condición anterior if es falsa. Puede tener múltiples declaraciones elsif después de un if.

Else: Se ejecuta al final de la estructura if-elsif. Se ejecuta si ninguna de las condiciones anteriores (if y elsif) son verdaderas. A continuación, se muestra la forma general de un código VHDL con una estructura **if-elseif-else**.

```
Process(a,b)                          -- el proceso son las entradas a y b.
    Begin

      If (a='1' and b='0') then       -- si a=1 y b=0 entonces la salida=1.
        Salida<='1';
      Elsif(a= '0' and b='1') then    --si lo anterior es falso, pero si a=0 y
        Salida<='1';                  -- b=1, entonces la salida=1

       Else                           -- si ambas condiciones son falsas
        Salida<= '0';                 -- salida=0
     End if;                          -- fin del bucle
End process;                          -- fin del proceso.
```

Es muy importante indicar que las estructuras lógicas de ciclos deben estar dentro de un proceso. En el capítulo correspondiente a la implementación y ensayo de programas reales, se analizará en detalle la estructura **Process** con el ciclo **if-elsif.**

CAPITULO 3

ENTORNO DE DESARROLLO QUARTUS II (VERSION 17.11)

3.1 GENERALIDADES

La herramienta de desarrollo a utilizar en este trabajo es perteneciente a empresa Altera y su nombre es Quartus II, la cual es un programa de diseño lógico. El software es capaz de trabajar con lenguajes de programación tales como VHDL y Verilog, los cuales son usados para describir el funcionamiento de los circuitos a trabajar que serán implementados en un FPGA o CPLD. En este capítulo se indicará la forma de instalar este software, así como también cómo se crea, simula y carga un código VHDL, para posteriormente implementar dicho hardware en la placa FPGA.

3.2 INSTALACIÓN

Para instalar este software se accede a la página de FPGA Software Download www.altera.com y en la lista que se muestra se busca “Intel Quartus II Design Software versión 171. La Figura 3-1 muestra una sección de la lista que la página de descargas, donde puede seleccionar el tipo de software, la versión y el sistema operativo. En este caso, es recomendable usar la versión 17.1.

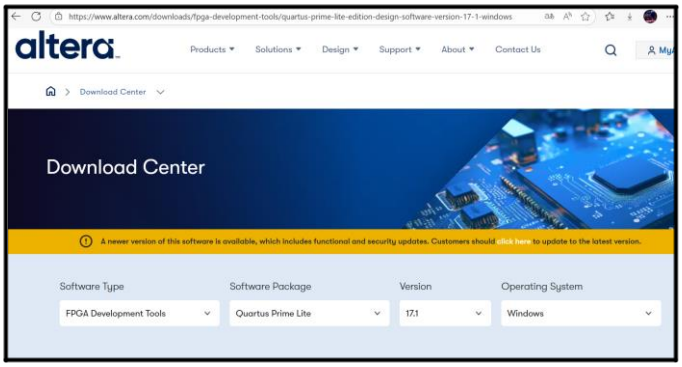


Figura 3-1_Centro de descarga.

En la misma página avanzando más abajo, se despliega una imagen similar a la figura 3-2, en la cual se descarga Quartus II 17.1, haciendo click en download, donde se abrirá una ventada de acuerdo de licencia de software, presionando aceptar comenzará la descarga. Una vez realizada la descarga se realiza la descomprensión del archivo RAR siendo posible ejecutar la instalación.

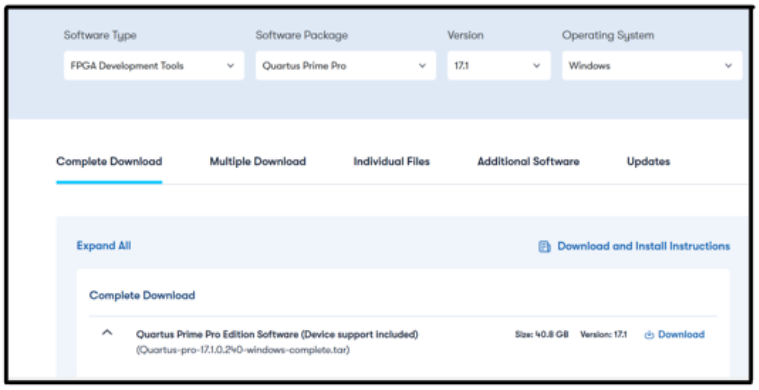


Figura 3-2_Descarga de Quartus II, versión 17.1

3.3 CREACION DE PROYECTOS

Una vez terminada la instalación, se encontrará habilitado para ejecutar Quartus II, haciendo click en el icono con el logo del software, según la ubicación en el computador. Una vez abierto el programa, aparece una imagen como la mostrada en la figura 3-3 en la cual se visualizan cinco secciones.

En la zona celeste, se ubica la barra de herramientas, donde se guardan, abren y crean los archivos. En la zona anaranjada, está la barra del navegador. Aquí se observan todos los archivos de un proyecto, con sus jerarquías y prioridades.

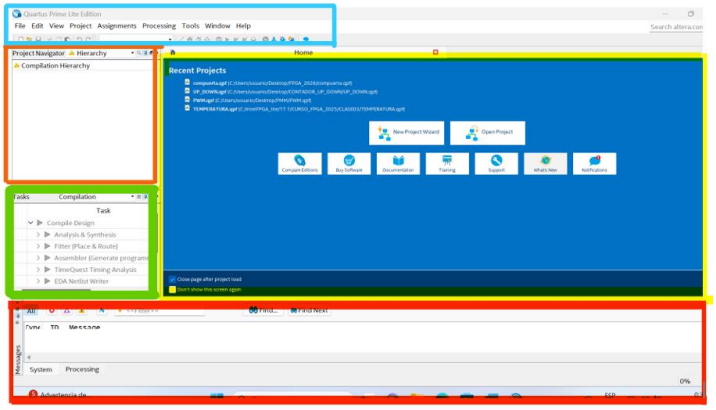


Figura 3-3- Vista general de Quartus II

En la zona amarilla está la zona de trabajo. En esta zona se escribe y edita el código fuente de los proyectos. El área verde muestra el estado del análisis y síntesis de un archivo, así como también el estado de compilación del proyecto completo. Finalmente, la zona roja, entrega todos los mensajes de advertencias, errores y resultados de una compilación.

Una vez revisadas las cinco secciones, están dadas las condiciones para crear un nuevo proyecto.

Para crear un proyecto hay que ubicar el cursor en la barra de herramientas, File, click en new proyect wizard, tal como lo indica la figura 3-4. El paso siguiente es crea una carpeta de trabajo en el PC, dicha carpeta se selecciona y guarda el nombre del proyecto en ella. En la figura 3-5 se visualiza el nombre de la carpeta creada (FPGA_2026), encerrada dentro de la elipse roja. En la zona verde se escribe el nombre del nuevo proyecto llamado MI_PRIMER_PROGRAMA. Paso siguiente presiona next, demarcada por el circulo celeste.

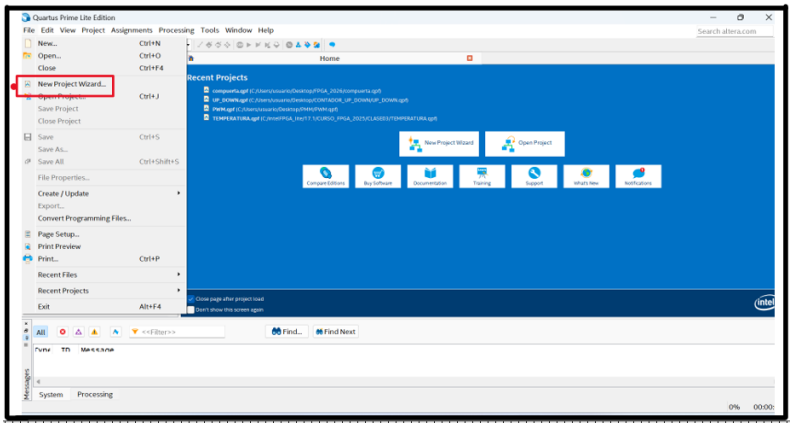


Figura 3-4 Barra de herramientas de Quartus II.

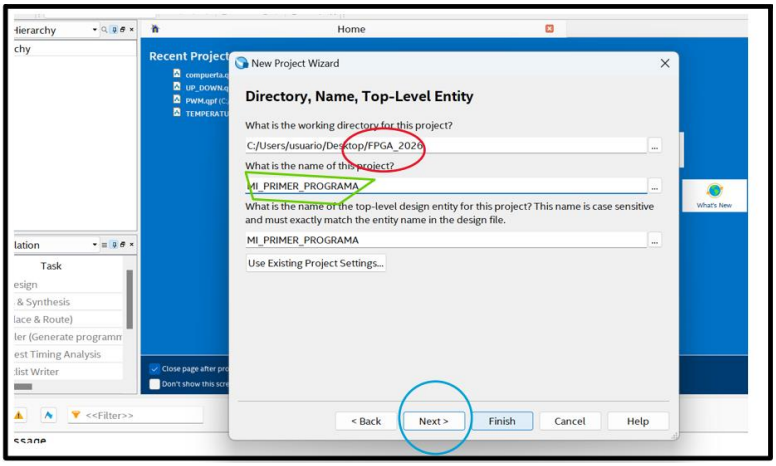


Figura 3-5_Creacion de carpeta y nombre del proyecto.

Una vez presionado Next de la figura 3-5, aparece una nueva ventana similar a la indicada en la figura 3-5 para seleccionar **Empty proyect y next,** encerrado en un círculo rojo.

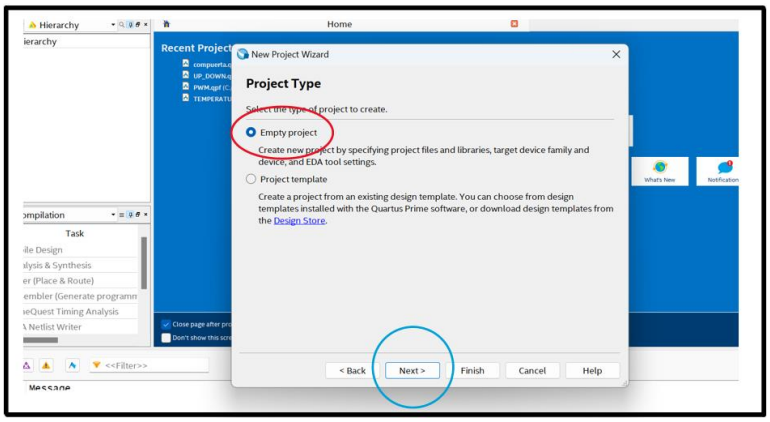


Figura 3-6_ Para un nuevo proyecto se debe indicar proyecto vacío (empty).

Paso siguiente, presionar next hasta que aparece la siguiente ventana, figura 3-7. Aquí se debe revisar y seleccionar del listado, para seleccionar la familia y FPGA a programar en el proyecto recién creado. En este caso seleccionar la familia MAX 10 y la FPGA 10M08SCE144C8G, para presionar next hasta que aparezca el resumen del proyecto en la figura 3-8.

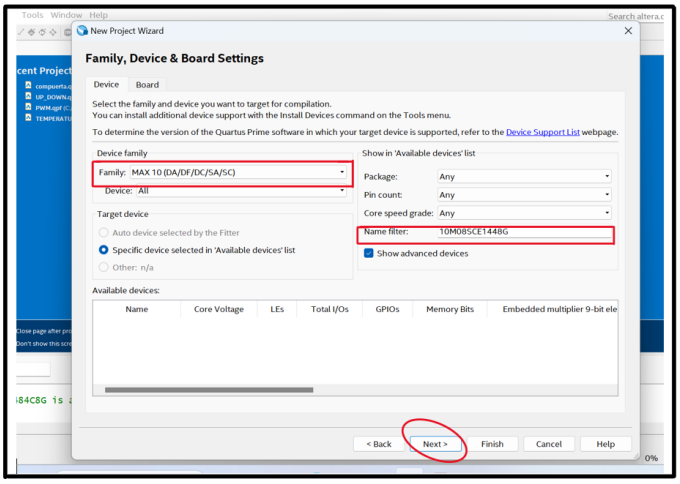


Figura 3-7_Seleccion de la familia MAX 10 y dispositivo FPGA.

El resumen entrega la información tal como la ubicación de la carpeta, el nombre del proyecto, la familia y el código de la FPGA a programar. Es muy importante revisar toda la información antes de presionar Finish.

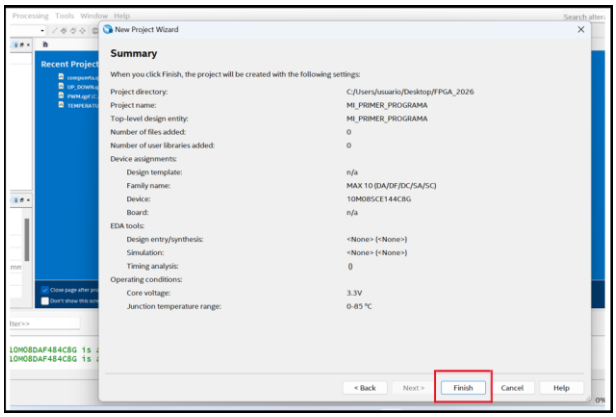


Figura 3-8_ Reporte de la creación del proyecto.

Como consecuencia de la creación del proyecto MI_PRIMER_PROGRAMA, en la figura 3-9 aparece en el árbol del proyecto el número de FPGA y el nombre del proyecto. Notar que se encuentra vacío, aun no hay archivos creados.

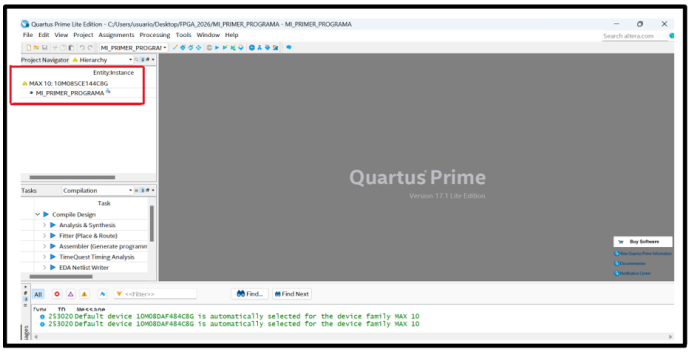


Figura 3-9_ Proyecto creado, se visualiza la ventana sin archivos creados.

3.4 CREACION DE ARCHIVOS EN VHDL

Una vez creado el proyecto, están las condiciones para escribir el código fuente en VHDL, es decir un archivo con extensión .vhd. Este primer Archivo debe tener el mismo nombre del proyecto, independiente que al proyecto se agreguen más archivos en el futuro. Mas adelante se revisará como sintetizar (compilar) diversos archivos en un mismo proyecto, para ello se le otorga alta prioridad al archivo escogido para sintetizar.

Para crear un archivo VHDL, seleccionar FILE y NEW, tal como indica la figura 3-10.

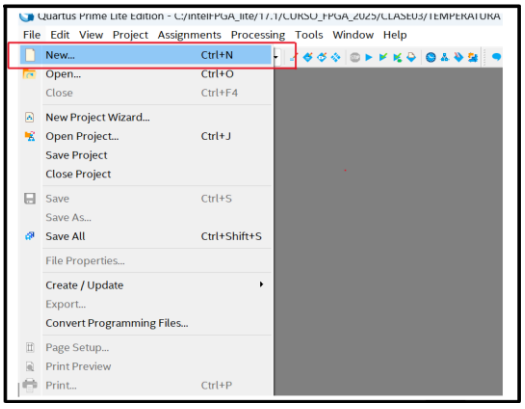


Figura 3-10_Creacion de un nuevo archivo de VHDL.

Posteriormente se desplegará una ventana mostrada en la figura 3-11, para seleccionar VHDL files (tipo de archivo) y presionar OK.

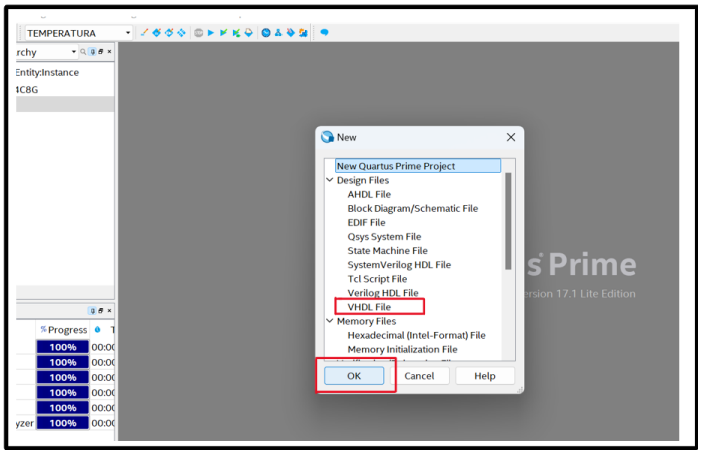


Figura 3-11_ Selección del tipo de archivo. Para este caso VHDL.

Una vez creado el archivo, aparece un campo de trabajo en blanco, tal como muestra la figura 3-12, donde se escribe el código VHDL. Una vez escrito el código VHDL, se selecciona File y presionar Save As (guardar como), se guarda dicho archivo con el mismo nombre del proyecto y el primer programa ha sido creado.

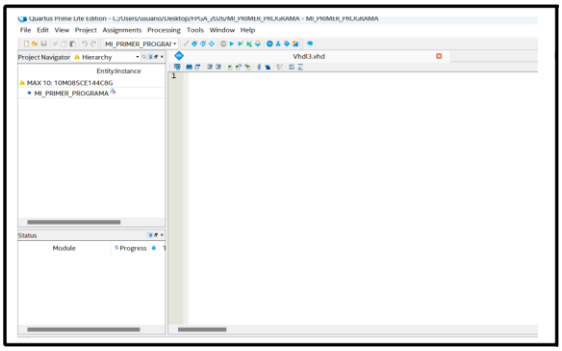


Figura 3-12_ Campo de trabajo en blanco disponible para escribir código VHDL.

Con el propósito de hacer más ilustrativo todos los procesos, se trabajará con el archivo llamado **PRIMER_PROGRAMA** que se muestra en la figura 3-13.

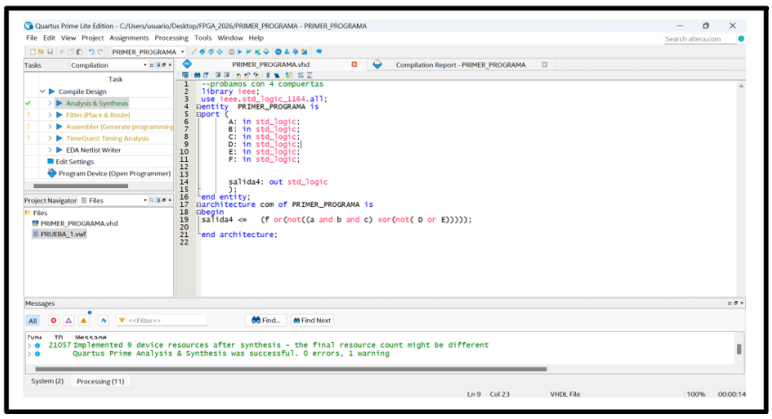


Figura 3-13_ Primer programa.

En el capítulo cuatro se revisarán programas prácticos, en los cuales será muy habitual usar esta misma metodología.

3.5 COMPILACION, ANALISIS Y SINTETIZADO

Antes de continuar hay que señalar que VHDL es un lenguaje descriptor de hardware tal como se ha indicado anteriormente y no es un lenguaje de programación tradicional, es un concepto muy diferente a un sistema de un microcortrolador (cpu- memoria-puertos e/s), con lo que hay que ser cuidadosos al hablar de programas y compilación. Al momento de escribir un código para VHDL el revisar errores, tanto de sintaxis y de lógica, se conoce como de análisis y sintetizado (lo que en lenguaje tradicional seria compilación), hasta aquí se está sintetizando y analizando un archivo.

En VHDL el concepto de compilación abarca un todo, es decir la totalidad del proyecto tal como el o los archivos dentro de un proyecto, el diagrama **pin planer**(conexión de pines al FPGA) y diagramas de bloques, tal como se aplicará más adelante.

3.51 ANALISIS Y SINTESIS

Una vez escrito el código VHDL en el editor, se procede a guardar el archivo con extensión “.vhd”, en la misma carpeta del proyecto, como se aprecia en la figura 3-14.

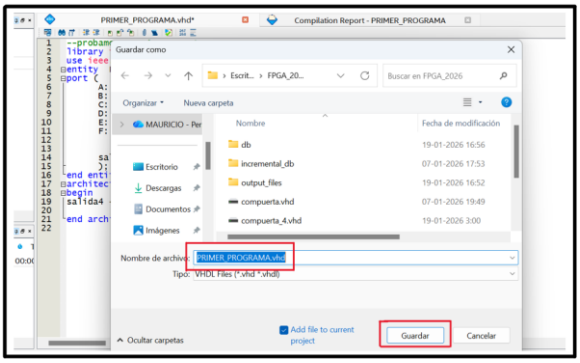


Figura 3-14_Guardado de archivo VHDL en un proyecto.

Paso siguiente, se debe seleccionar el comando de análisis y síntesis, esperar unos segundos y entregara el resultado de la existencia o no de errores, así como un resumen de dicho análisis. La figura 3-15 ilustra el icono a seleccionar para realizar el analisis y sintetizado del archivo de trabajo.

En la parte inferior entregara los mensajes de errores y advertencias que pudiese detectar dicho proceso.

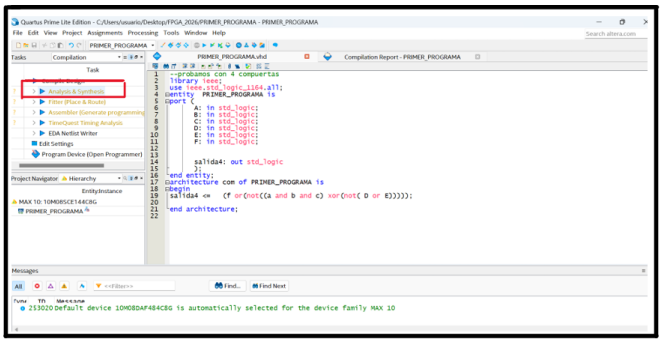


Figura 3-15_Analisis y sintesis del archivo VHDL, dentro de un proyecto.

La figura 3.16 indica, el resumen del resultado del análisis y síntesis, se aprecia el tipo de FPGA, el total de elementos lógicos requeridos para programar el

dispositivo FPGA, así como también el número de pines necesarios para implementar el hardware.

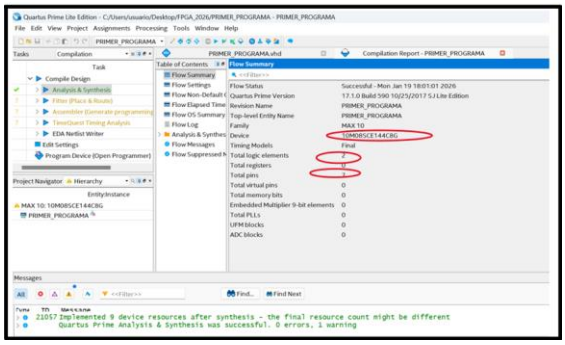


Figura 3-16_Resumen del análisis y sintetizado del archivo VHDL.

3.52 DIAGRAMAS RTL

Dentro de las herramientas importantes que posee Quartus II, está disponible la creación de diagramas RTL en forma opcional. Estos diagramas RTL muestran el hardware resultante producto del código escrito y sintetizado de VHDL, en el editor de texto indicado en el apartado anterior.

Los diagramas RTL ayudan en ocasiones a optimizar la descripción de hardware requerido para el proyecto deseado, así como también visualizar la entidad, arquitectura y las señales (conexiones internas dentro de la arquitectura). Una vez sintetizado un archivo, se debe seleccionar la barra de herramientas, seleccionando **Tools**, luego **Nestlist Viewers** y finalmente **RTL Viewer**, tal como indica la figura 3-17.

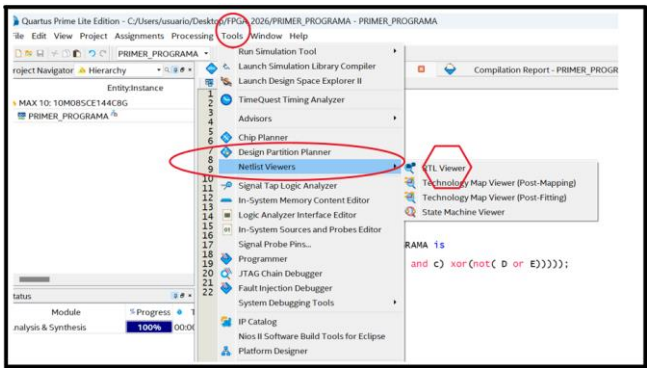


Figura 3-17_ Ruta de los diagramas RTL.

Posterior a la selección de **RTL viewer**, aparece una nueva ventana con el hardware resultante tal como se ilustra en la figura 3-18. Notar que las zonas encerradas en rojo corresponden a la entidad del código (revisar capitulo anterior), la zona amarilla indica la arquitectura y las encerradas en verde son las señales. En ocasiones estos diagramas serán de gran utilidad, así como la interpretación y la utilidad obtenida de estos diagramas, dependerá de la habilidad y experiencia adquirida del programador.

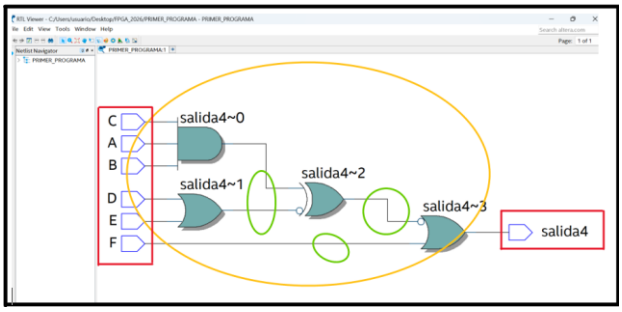


Figura 3-18_ Diagrama RTL resultante, indicando, las entradas, señales y salida.

3.6 SIMULACION DE CODIGOS VHDL

La simulación es otro recurso disponible en QUARTUS II, que puede ser utilizado en forma opcional por parte del usuario. Esta herramienta permite simular el circuito diseñado, comprobando el correcto funcionamiento y poder corregir o depurar el código recientemente analizado y sintetizado.

Antes de simular se debe crear el archivo llamado como waveform (extensión. VWF) y para ello seleccionar **File**, **New** y **university program vwf** tal como muestra la figura 3-19.

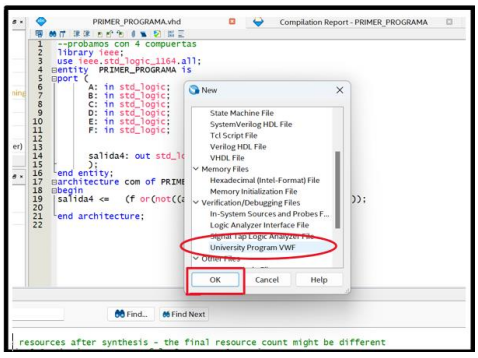


Figura 3.19_Creacion de archivo de simulación.

Una vez realizado lo anterior se desplegará una nueva ventana como la desplegada en la figura 3-20, la cual representa el campo de trabajo del simulador.

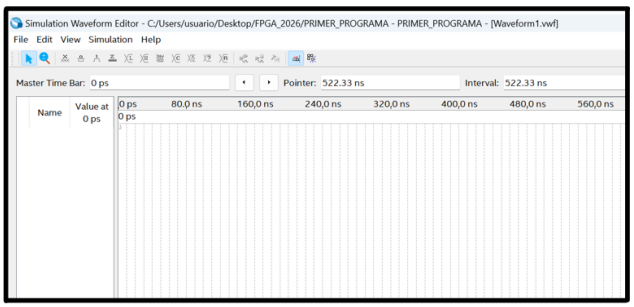


Figura 3-20_Campo de trabajo desplegado del simulador.

Paso siguiente se debe guardar en forma de archivo con extensión. vwf dentro de la carpeta del proyecto la simulación. Para este caso en particular fue nombrada PRUEBA_1.VWF. La figura 3-21 muestra en la imagen generada, la forma de guardar el nuevo archivo del simulador creado.

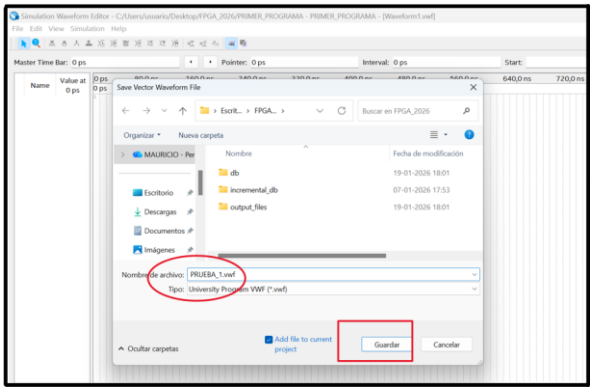


Figura 3.21_ Nombre del nuevo archivo creado del simulador.

Al regresar a la vista principal, se comprueba que el archivo recién creado aparece en la ventana del navegador con la extensión. VWF en la figura 3-22.

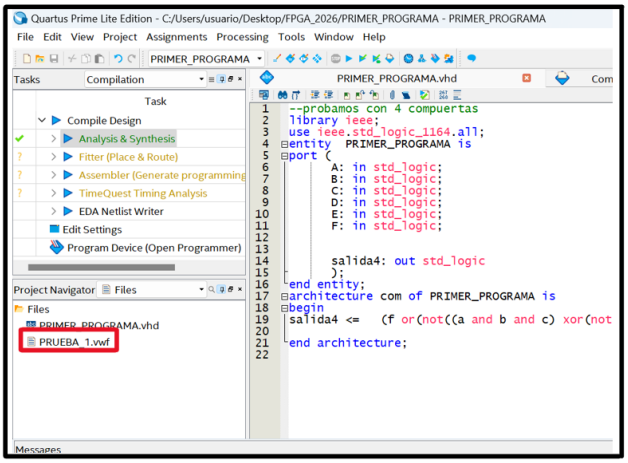


Figura 3-22_Ventana del navegador con el archivo del simulador.

Una vez realizado todos estos pasos, están las condiciones de realizar una simulación. Este archivo, **primer programa** es un circuito combinacional idóneo para aplicar algunos estados lógicos en las entradas y verificar el estado lógico

en la salida. Para ello se debe hacer Doble click en el campo de trabajo del simulador ya desplegado, para este caso el archivo llamado PRUEBA_1.VWF indicado en 1 en la figura 3-23, posteriormente seleccionar **Node Finder** (2), donde aparecerá un segundo recuadro, seleccionamos **List** (3), **>>** (4) y finalmente en **ok** (5) y **ok** (6).

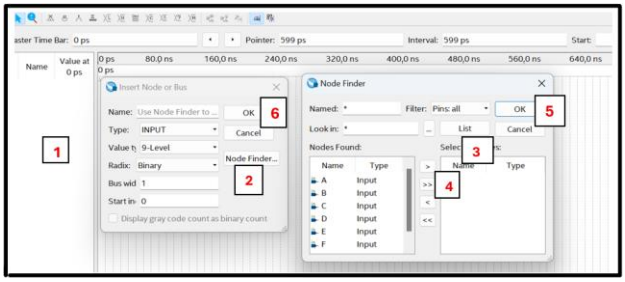


Figura 3-23_Procedimiento para desplegar el simulador de Quartus II.

Una vez realizado correctamente los pasos anteriores se desplegará una ventana similar a la ilustrada en la figura 3-24. Al momento de seleccionar correr simulación, presionando click en zona cerrada en cuadrado rojo, comienza a correr el simulador y arroja la forma de onda de la salida en el campo desplegado e indicada en la elipse roja, tal como lo muestra la figura 3-25. Notar que ambas figuras a la plantilla del simulador se aprecian los estados binarios en cada entrada, al frente de cada línea de tiempo.

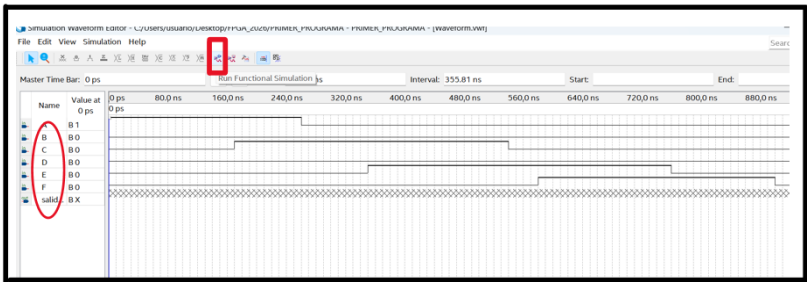


Figura 3-24_Simulador con 6 entradas y los diagramas de tiempo. La salida aún no se despliega hasta presionar RUN.

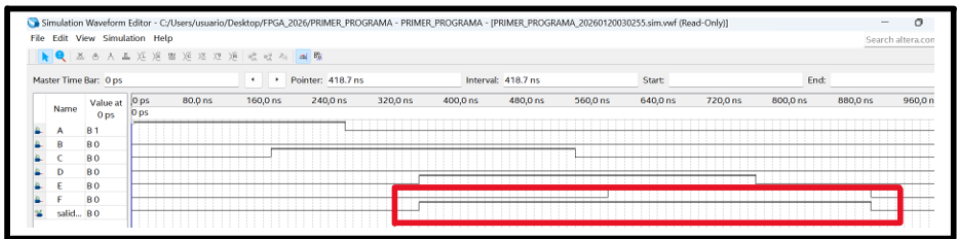


Figura 3-25_Salida 4 del circuito desplegando el diagrama de tiempo encerrado en el rectángulo.

Con lo anterior se puede concluir que la simulación entrega el resultado en la salida_4 del circuito combinacional, según los estados lógicos en el tiempo, introducidos en las entradas A, B, C, D, E Y F.

3.7 CONFIGURACION DE PINES (PIN PLANER)

Previo a cargar le código VHDL a la FPGA, es necesario configurar los pines a utilizar por dicho dispositivo FPGA, según la disponibilidad en la placa de desarrollo. En el Apéndice A se encuentra la información técnica de la placa MAX 10 con la que se probarán los programas aquí descritos. Para ello se selecciona en barras de herramientas, **Assignments** y **pin planer** según lo indicado en la figura 3-26.

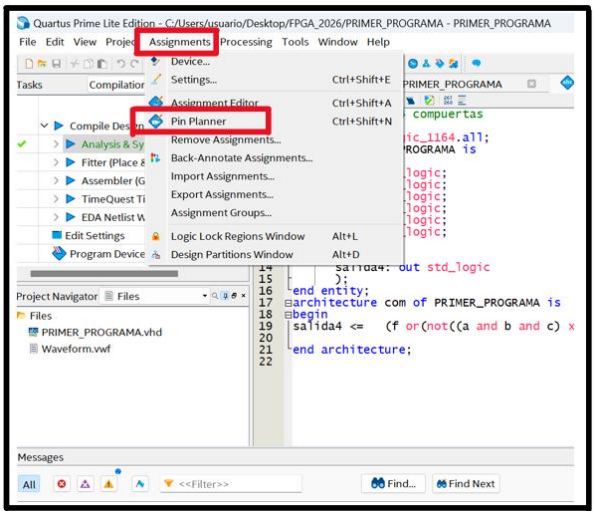


Figura 3-26_Acceso al pin planer, seleccionando la barra de herramientas.

Posteriormente se despliega una imagen figura 3-26, la cual muestra la información correspondiente al dispositivo FPGA 10M08SCE144C8G, entre otros datos que se revisarán más adelante. En las casillas indicadas por el rectángulo vertical de la figura 3-27, se ingresan los pines a usar en la FPGA, una vez llenado las casillas se minimiza la ventana, para posteriormente compilar el proyecto.

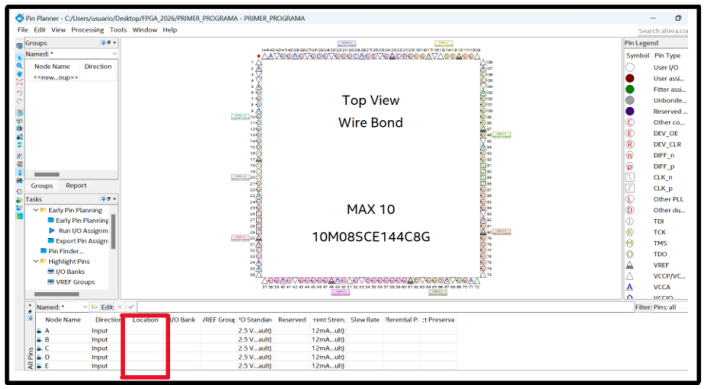


Figura 3-27_Cuadro desplegado de pin planer, en la zona del rectángulo vertical se ingresan los pines a usar del dispositivo FPGA.

3.8 CREACION DE DIAGRAMAS DE BLOQUES.

En los proyectos en VHDL es muy común trabajar con más de un archivo en consecuencia, ello requiere interconectar o asociar dichos archivos para conformar un solo proyecto con una determinada tarea a realizar. La razón de trabajar en bloques entrega una mayor portabilidad de ciertos archivos para ser utilizados en otros proyectos. La forma de asociar dichos archivos es a través de la creación de diagramas de bloques también conocido por algunos autores como el modo arquitectura. En resumen, se pretende armar un diagrama de bloques (archivo de extensión.bfd), interconectando todos los archivos de la forma planificada. El primer paso consiste en generar un campo de trabajo o una malla para insertar los bloques necesarios a interconectar. Para ello Seleccionar **File**,

New, se despliega otro recuadro y seleccionar **Block Diagram/Schematic File** y finalmente **ok.** La figura 3-28 ilustra lo anterior.

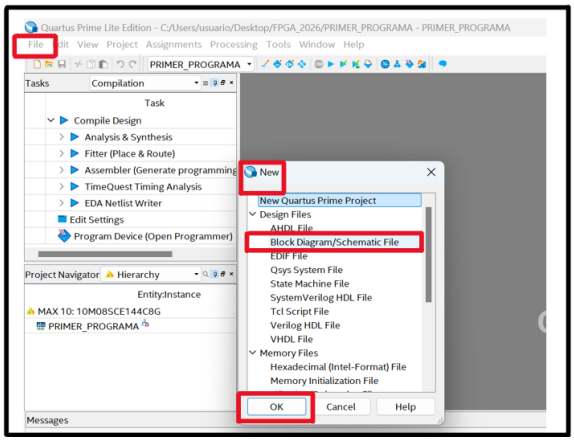


Figura 3-28_Creacion de un archivo para armar diagramas de bloques.

Una vez ejecutados todos estos pasos se despliega una malla de trabajo, tal como muestra la figura 3-29.

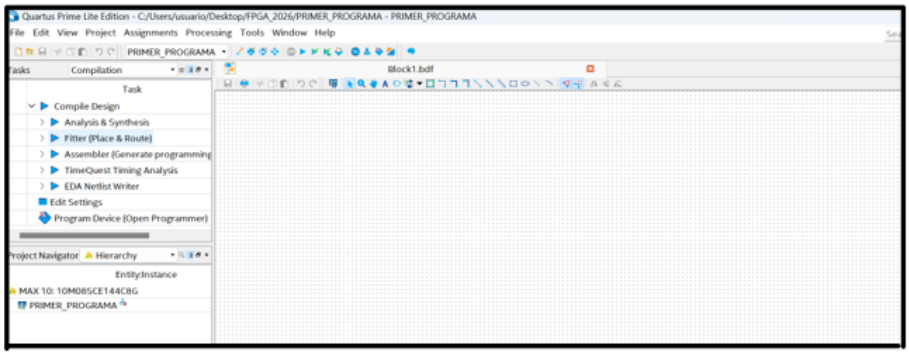


Figura 3-29_Grilla de trabajo para realizar diagramas de bloques.

Una vez desplegada la grilla de trabajo para la creación de bloques, debe quedar abierta y se debe crear los bloques individuales para posteriormente insertarlos a la grilla, interconectarlos y terminar de armar el proyecto.

Para crear el símbolo de cada archivo, es decir un bloque, se debe seleccionar el archivo deseado, en este caso **PRIMER_PROGRAMA,** click botón derecho, para abrir un recuadro open y seleccionar **Create Symbol Files Current File**, tal como lo indica la figura 3-30. Importante, Proyect Navigator debe permanecer en **Files**.

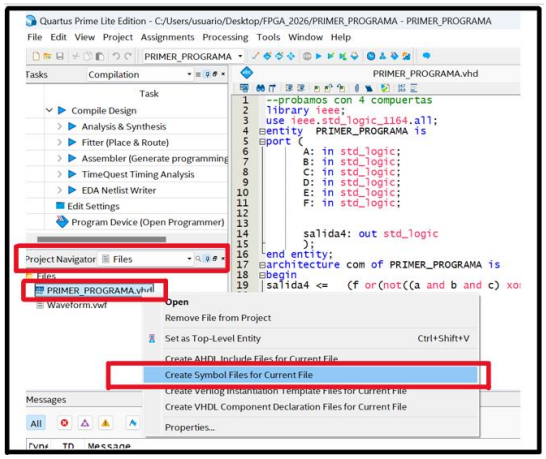


Figura 3-30_Creacion de un bloque correspondiente a un determinado archivo.

Una vez creado el bloque, se abre una ventana con el reporte de la creación del bloque, similar a la figura 3-31.

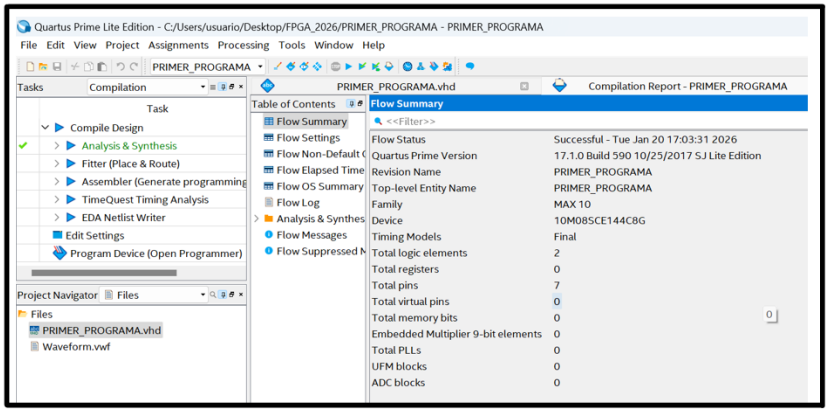


Figura 3-31_Reporte de creación de un símbolo.

Como etapa final en la creación de bloques se debe regresar a la grilla, para insertar en ella los símbolos creados anteriormente, en la cual se hace doble click sobre la zona de grilla indicada con el numero 1 tal como se indica en la figura 3-

32, luego de ello aparece el recuadro symbol (indicado en punto 2), en el cual se hace doble click en proyect, desplegando el listado de todos los archivos ya en bloques (símbolos). Para este caso existe solo el archivo **primer _programa,** doble click sobre él, click en ok y el bloque queda en la grilla.

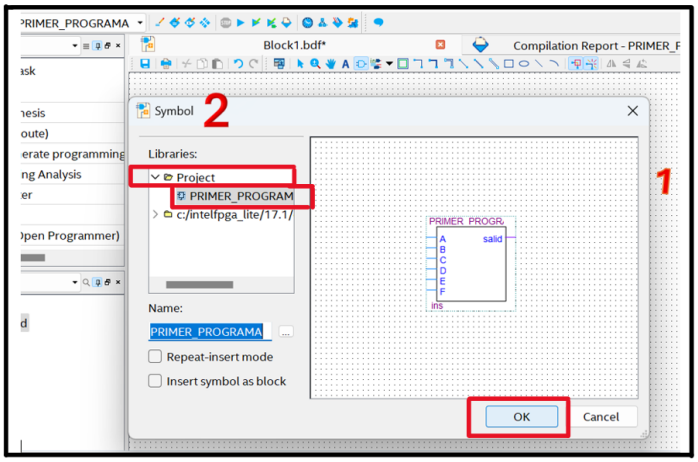


Figura 3-32_Se insertan los simbolos creados en la grilla, para construir el proyecto total.

Posteriormente aparecerá el bloque en la grilla, para poder así seleccionarlo con click botón derecho, generando los pines de entrada y salida tal como lo indica la elipse roja, en la figura 3-33.

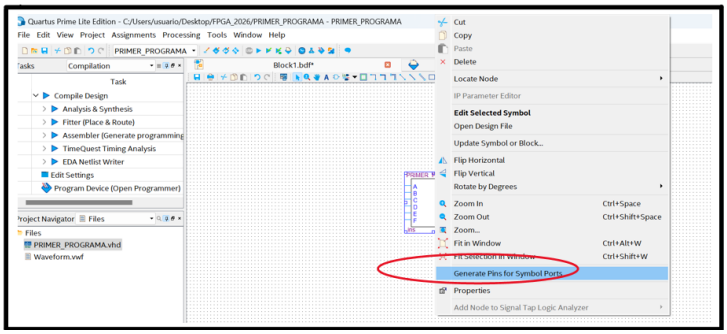


Figura 3-33_Generacion de pines I/O del bloque creado con extensión bfd.

En la figura 3-34 se puede observar el bloque con sus pines, paso seguido corresponde guardar el diagrama de bloques como un archivo de extensión del tipo “. bfd”, tal como está indicado en la figura antes mencionada.

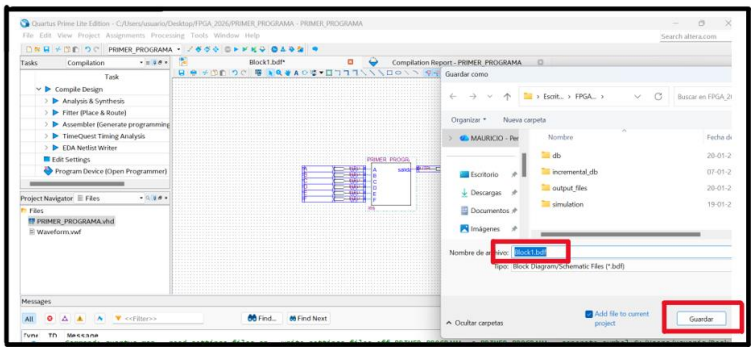


Figura 3-34_Guardado del bloque generado, con la extensión. bfd.

Una vez creado el archivo con extensión “. bfd”, se debe sintetizar el archivo creado en estos cuatro pasos, seleccionando la ventana de navegador, cambiado de **hiperactivity a Files** (1), seleccionar **block1.bdf** (2), **set as top level priority** (3), **análisis y síntesis** (4), tal como en a figura 3.35. Recordar que todo archivo a sintetizar siempre debe ser puesto en alta prioridad (**set as top level priority).**

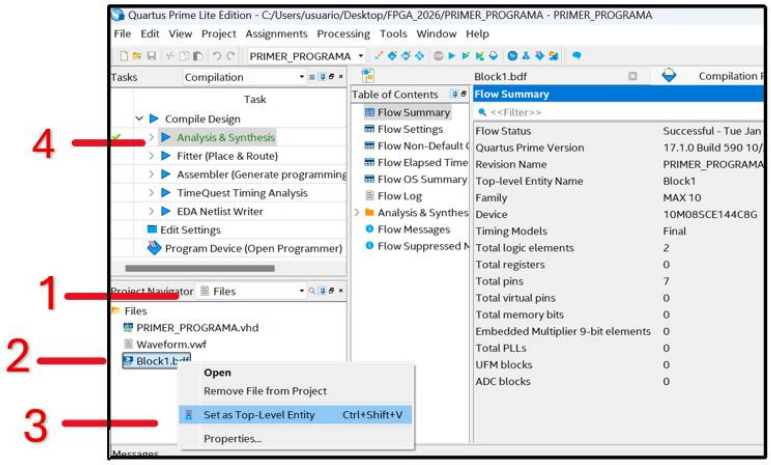


Figura 3-35_analisis y sintetizado del archivo correspondiente al bloque creado.

Una vez analizado y sintetizado, en la figura 3-36 se aprecia el resultado de la compilación y en detalle se observa el nombre del archivo (block 1), la cantidad de elementos lógicos y la cantidad de pines a utilizar el mencionado bloque.

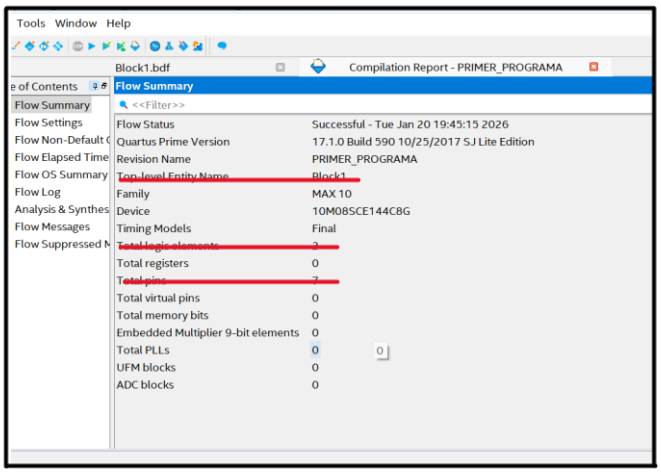


Figura 3-36_Resultado de la compilación del bloque creado.

Finalmente, analizado y sintetizado el o los archivos VHDL, el **pin planer** con todos los pines definidos del dispositivo FPGA y el diagrama de bloques, se procede a compilar el proyecto completo, seleccionando **Compile Design** tal como lo indica la figura 3-37.

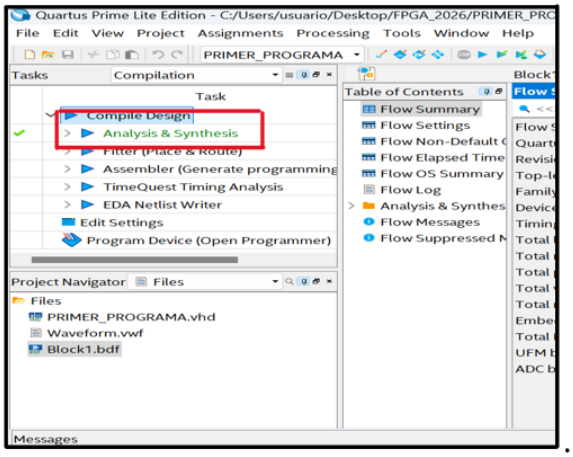


Figura 3-37_Compilado del proyecto completo.

Finalmente, en la figura 3-38, se observa el resumen de la compilación del proyecto completo. Notar que, al compilar todos los tipos de archivos de un proyecto, se aplica el concepto de compilación en lugar de análisis y sintetizado utilizado cuando se trabaja con los archivos en forma individual.

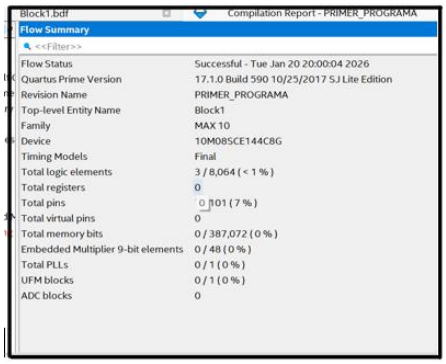


Figura 3-38_Resumen de compilación del proyecto completo.

3.9 PROGRAMACION DE LA PLACA FPGA.

Es importante indicar que para programar el proyecto debe estar debidamente compilado y haber chequeado atentamente la distribución de pines de la placa (pin planer), así como también revisar que la placa se encuentre energizada y conectado el usb blaster al PC. Para programar seleccionar barra de herramientas, **Tools** y **Programmmer,** como lo indica la figura 3-39.

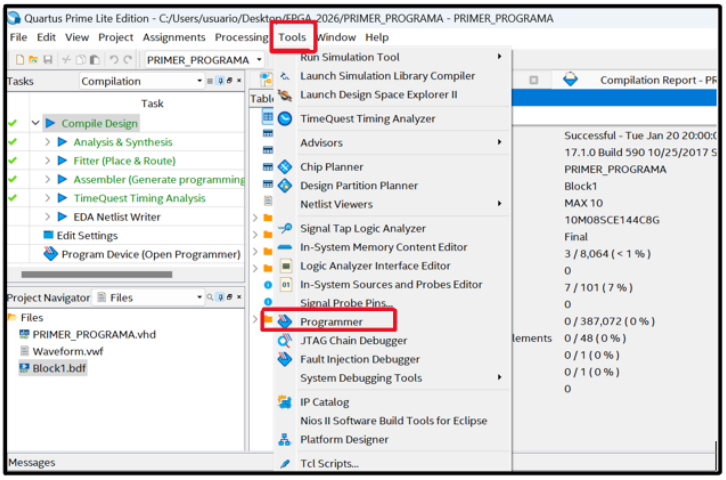


Figura 3-39_Seleccion de iconos para programar el dispositivo FPGA.

La figura 3-40 muestra la ventana desplegada para programar, verificar que el **USB BLASTER** haya sido reconocido y presionar **Start** para realizar la programación. Una vez programado el chip indicara un mensaje de programación exitosa.

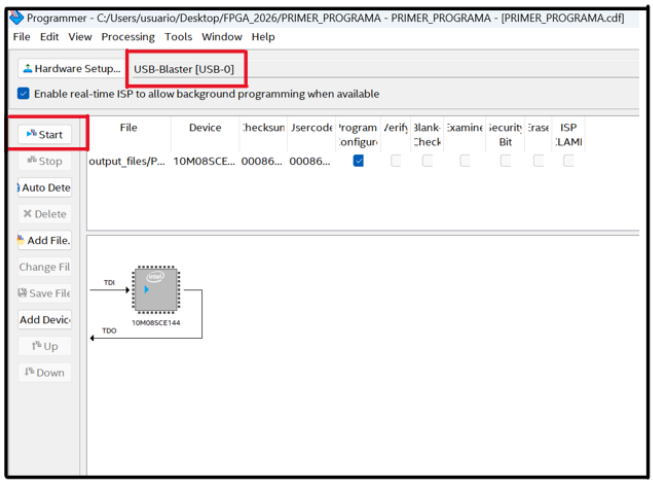


Figura 3-40_Programacion de dispositivo FPGA.

CAPITULO 4

REALIZACIONES PRACTICAS DE CIRCUITOS COMBINACIONALES Y SECUENCIALES CON DISPOSITIVOS FPGA

4.0 ENTORNO GENERAL

En este capítulo se realizarán los primeros ensayos prácticos, utilizando los conocimientos aprendidos en los capítulos anteriores en VHDL y Quartus II. Los ensayos de estos códigos se implementarán en circuitos combinaciones y circuitos secuenciales. Primeramente, se comienza con los códigos de circuitos combinacionales que realizan funciones específicas tales como codificadores, multiplexores entre otros, en forma individual, para posteriormente revisar los circuitos secuenciales como, divisores de frecuencia, contadores, moduladores PWM, todos como programas individuales, que posteriormente serán interconectados entre sí en forma de bloques, para revisar proyectos de mayor envergadura. Esta modalidad se conoce como el método de arquitectura y será revisado más adelante.

Como se ha indicado, se utilizará una placa de desarrollo MAX_10, en la cual se ven los detalles técnicos en el apéndice A. La figura 4-1 muestra es aspecto físico de la placa MAX_10, indicando los terminales de entrada y salida que se usaran en las practicas.

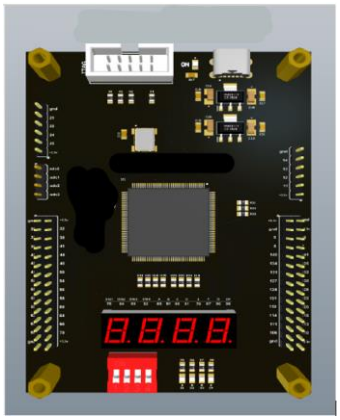


Figura 4-1_Placa de desarrollo MAX-10.

Un punto importante para recordar es que todos los códigos desarrollados aquí, son portables a otras placas de FPGA y solo se modifica el pin planer (los números de pines del dispositivo FPGA en particular). En la figura 4-1, en la parte inferior izquierda tiene 4 interruptores (dipswitch), al costado derecho 4 leds, en la parte inmediatamente superior 4 Dislplay de 7 segmentos y 60 puertos I/O entre otros recursos (ver Apéndice A).

4.1 CIRCUITOS COMBINACIONALES

Los circuitos combinacionales son sistemas lógicos digitales, en los cuales el estado de sus salidas depende únicamente de sus entradas en un determinado momento. Estos circuitos carecen de memoria, por lo que su salida no depende de estados anteriores.

4.11 COMPUERTA LOGICA NAND DE 4 ENTRADAS

El primer ensayo será una simple compuerta lógica de cuatro entradas, es decir escribirá un código en VHDL que una vez sintetizado y compilado el proyecto completo, el código configurará la compuerta lógica de 4 entradas dentro del dispositivo FPGA.

La figura 4-2 muestra una representación del hardware deseado que ejecutara el FPGA como el primer practico. Se utilizará el Dip switch de 4 pines disponible en la placa MAX-10. En la figura 4-3 se aprecia la distribución de pines de la placa. El circuito de entrada está formado con resistencias pull up’s de 10k, los Switchs en OFF es un 1 lógico y en ON es un 0 lógico. Por otro lado, la salida que se ocupará será el pin número 93 de la placa de desarrollo (ver figura 4-4).

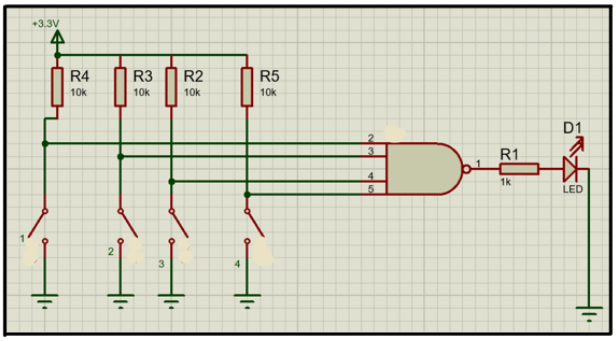
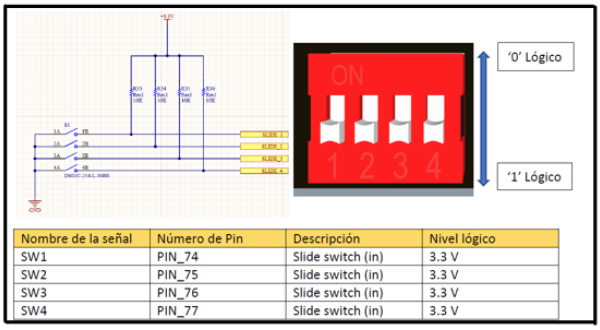


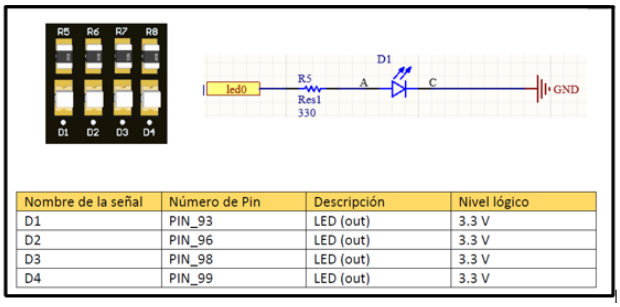
Figura 4-2_Hardware de una compuerta NAND de 4 entradas a ensayar.



Nombre de la señal	Número de Pin	Descripción	Nivel lógico
SW1	PIN_74	Slide switch (in)	3.3 V
SW2	PIN_75	Slide switch (in)	3.3 V
SW3	PIN_76	Slide switch (in)	3.3 V
SW4	PIN_77	Slide switch (in)	3.3 V

Figura 4-3_Distribucion de pines de Dip switch de la placa MAX-10.

La información de la figura 4-3 será de vital importancia al momento de compilar el proyecto completo, en el sentido de ingresar el número de pines en el PIN planer una vez trabajando en Quartus II.



Nombre de la señal	Número de Pin	Descripción	Nivel lógico
D1	PIN_93	LED (out)	3.3 V
D2	PIN_96	LED (out)	3.3 V
D3	PIN_98	LED (out)	3.3 V
D4	PIN_99	LED (out)	3.3 V

Figura 4-4_Salida del pin 93, conectada a D1.

La figura 4-4 muestra la disponibilidad de 4 leds que posee la placa de desarrollo y el primer ensayo, se usará el diodo led 1(D1) para visualizar el estado lógico de la compuerta NAND a implementar. En pocas palabras los Dip switch (pines 74,75,76 y 77), serán las entradas lógicas y led 1(93) en la salida.

En la figura 4-5 se aprecia el primer código, muy básico, correspondiente a la compuerta NAND de cuatro entradas. La estructura básica de un código en VHDL como se ha mencionado anteriormente, cuenta con librerías, entidad y arquitectura. En la línea 1, en ella se aprecia un comentario, cada comentario comienza con dos guiones seguidos y el compilador ignora el contenido de este. Las líneas 2 y 3 describen la librería y el paquete a usar (paquete usado para trabajar con bits 0 y 1 lógico). Entre las líneas 4 y 12 se escribe la entidad con sus puertos de entrada- salida desde A hasta D, salida4, el modo entrada o salida y el tipo de datos, en este caso bits. Desde la línea 13 a 17 se ubica el cuerpo de la arquitectura. En la línea 13 se debe nombrar a la arquitectura, para este caso **com,** así como el nombre del proyecto, **compuerta4**. Antes de entrar al cuerpo de la arquitectura, se debe agregar **begin,** tal como se aprecia en la línea 14. En la línea 15 se aprecia el cuerpo de la arquitectura es decir la función a realizar propiamente tal por parte del código a programar en el dispositivo FPGA, para este caso una función NAND, de 4 entradas.

```
1  --probamos una compuerta nand con 4 entradas
2  library ieee;
3  use ieee.std_logic_1164.all;
4  entity  compuerta4 is
5  port (
6          A: in std_logic;
7          B: in std_logic;
8          C: in std_logic;
9          D: in std_logic;
10         salida4: out std_logic
11         );
12 end entity;
13 architecture com of compuerta4 is
14 begin
15 salida4 <= NOT (A AND B AND C AND D);
16
17 end architecture;
```

Figura 4-5_Codigo VHDL correspondiente a una compuerta NAND 4 entradas.

Es importante tener presente que Quartus II, permite trabajar con las tres compuertas lógicas básicas, es decir AND, OR y NOT. La función NAND se obtiene, anteponiendo un negador (NOT) a la expresión AND, tal como se aprecia en el código. Es muy útil en ciertos casos visualizar el diagrama RTL del código tal como lo indica la figura 4-6, una vez analizado y sintetizado el código. La línea 17 pone termino a la arquitectura y el código de este primer proyecto.

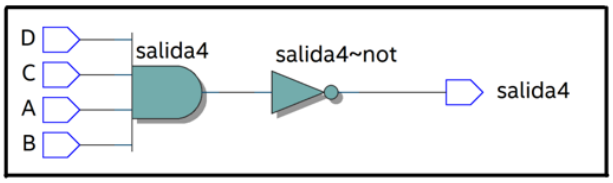


Figura 4-6_Diagrama RTL de una compuerta NAND de 4 entradas.

Una vez escrito, analizado y sintetizado el código y por ende sin errores de “compilación”, se procede a programar el dispositivo FPGA FPGA, sin olvidar de configurar los pines de entradas y salidas de la placa de desarrollo Max 10. Las figuras 4-3 y 4-4 contienen la información de los pines a utilizar. Para ello, se debe seleccionar en menú, **assignements**, **pin planer** y se desplegara una imagen tal como indica la figura 4-7 (revisar más detalles en el capítulo 3 en programación de FPGA).

En la imagen desplegada de la figura 4-7, se puede observar el diagrama del FPGA con el tipo dispositivo utilizado el 10M08CSE144C8G, con sus correspondientes pines. En la parte inferior hay una tabla con alguna información a completar y otra que aparece producto de la “compilación” realizada con anterioridad. La columna **node name** muestra los pines indicados en la entidad del codigo, es decir A, B, C, D y salida4. La columna **direction** indica el modo de

los puertos, para este caso 4 en modo entrada y 1 en modo salida. Finalmente, la columna **location** se ubican las celdas a escribir el número de los pines a

utilizar de acuerdo con la información de las figuras 4-3 y 4-4. Para este caso puntual los pines 74, 75, 76, 77 como entradas y el pin 93 como salida. Antes de programar, se debe minimizar la ventana **pin planer** y proceder a compilar el proyecto completo, para mayores detalles se sugiere revisar el capítulo 3.

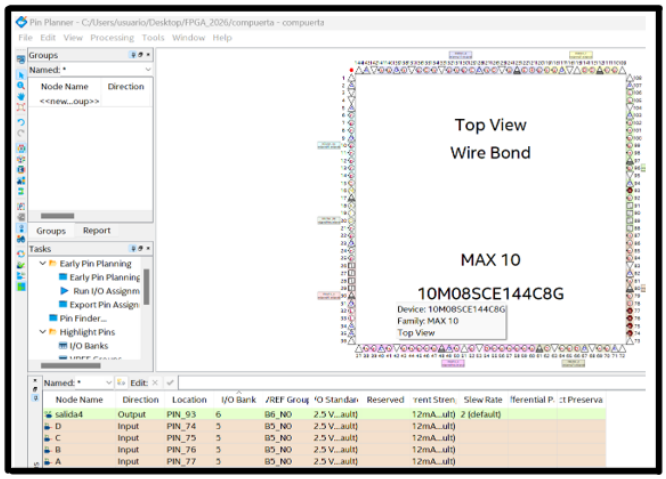


Figura 4-7_ Diagrama Pin Planer. En location se asignan los pines a usar del FPGA.

4.12 DECODIFICADOR BCD A SIETE SEGMENTOS

El decodificador BCD a siete segmentos a ensayar, consiste en ingresar un código de 4 bits, a través de un Dip switch de cuatro pines existente en la placa MAX-10 y para poder visualizar el equivalente al código BCD en un Display de siete segmentos, se utiliza uno de los cuatro Display disponible en la placa. En la figura 4-8 se visualiza una primera aproximación de la práctica a realizar.

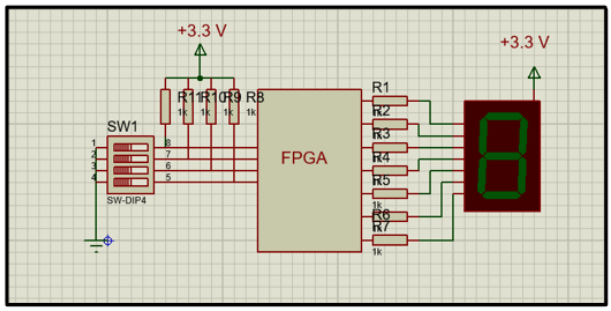


Figura 4-8_Decodificador BCD a siete segmentos.

Para ensayar el circuito anterior utilizando la placa MAX 10, se recurre a la información de la figura 4-3 para los pines relacionados con el Dip switch y para usar uno de los 4 Display's de siete segmentos de ánodo común contenidos en la placa, se obtiene la información de la figura 4-9. En la figura mencionada, se visualiza DG1 hasta DG4, los pines encargados de seleccionar el Display deseado, a través de un transistor 9012. La correspondencia de los pines se visualiza en la parte inferior de DG1 a DG4. Por otra parte, se observan las 7 líneas desde A hasta G, sumado DP (decimal point), con la correspondencia de los pines del FPGA en la parte inferior.

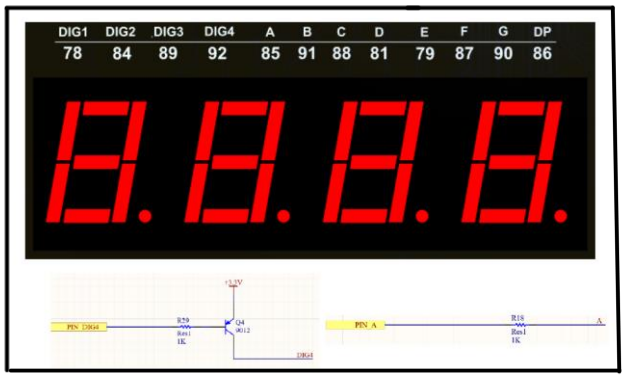


Figura 4-9_Distribucion de pines del FPGA hacia los Display de 7 segmentos.

Es Importante tener presente que los segmentos van conectados en paralelo de los 4 Display. Un transistor PNP actúa como switch electrónico para habilitar el display deseado (seleccionar DG1, DG2, DG3 o DG4) de ánodo común, con cero lógico en la base. Las líneas de cada segmento, es decir PIN A hasta PIN G van conectadas a través de una resistencia de 1 k, tal como lo muestra nuevamente la figura 4-9.

Antes de escribir código VHDL, es fundamental plantear una tabla de verdad con la conversión requerida. En la tabla 4-1 se observa que las cuatro primeras columnas de izquierda a derecha, se encuentra el código BCD, seguido de las siete columnas correspondientes a los pines del Display, notar que el Display está conectado en ánodo común, en consecuencia, la lógica es inversa, es decir, para que enciendan los segmentos se aplica 0 lógico, contrariamente a un Display de cátodo común. Finalmente, la última columna indica el valor a mostrar en decimal. Importante mencionar que para valores BCD no incluidos en la tabla, en la salida el Display no mostrara carácter alguno, por ende, se codifica con 1 lógico todos los pines de A hasta G.

TABLA DE VERDAD											
ENTRADA BCD				SALIDA DE SIETE SEGMENTOS							VALOR A
D	C	B	A	A	B	C	D	E	F	G	VISUALIZAR
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0	2
0	0	1	1	0	0	0	0	1	1	0	3
0	1	0	0	1	0	0	1	1	0	0	4
0	1	0	1	0	1	0	0	1	0	0	5
0	1	1	0	1	1	0	0	0	0	0	6
0	1	1	1	0	0	0	1	1	1	1	7
1	0	0	0	0	0	0	0	0	0	0	8
1	0	0	1	0	0	0	1	1	0	0	9
	OTROS			1	1	1	1	1	1	1	X

Tabla 4-1_Equivalente del código BCD a 7 segmentos.

En la figura 4-10 se visualiza el código VHDL con una de las tantas formas de realizar la conversión a siete segmentos. Para este caso se utilizó **with selec**, cuya idea consiste en que la salida tendrá un determinado valor binario “cuando” la entrada tenga un determinado valor en BCD.

```
1  library ieee;
2  use  ieee.std_logic_1164.all;
3  entity SEVENSEG is
4  port(
5       ENT: in std_logic_vector( 3 downto 0);
6       SAL: out std_logic_vector( 6 downto 0);    --Salida 7 seg
7       CONT: out std_logic_vector(3 downto 0)     --control display anodo comun
8  );
9  end entity;
10 architecture comp of SEVENSEG is
11   begin
12      with Ent select
13 SAL<="0000001" when "0000",--0
14       "1001111" when "0001",--1
15       "0010010" when "0010",--2
16       "0000110" when "0011",--3
17       "1001100" when "0100",--4
18       "0100100" when "0101",--5
19       "1100000" when "0110",--6
20       "0001111" when "0111",--7
21       "0000000" when "1000",--8
22       "0001100" when "1001",--9
23       "1111111" when others;--apaga todo con otros valores
24       CONT<="1110";--SELECCIONA CON CERO EL DIGITO DESEADO.
25 end architecture;
```

Figura 4-10_Codigo VHDL correspondiente al decodificador BCD a 7 segmentos.

Al revisar el código VHDL. En las líneas 1 y 2 se describen las librerías y los paquetes a usar. Como todos los datos a trabajar son grupo de bits o arreglos se utiliza el paquete **std_logic_vector.** La entidad comienza en la línea 3 y termina en la 9. Dentro de la entidad se definen 3 puertos; ENT**,** SAL y CONT, el primero en modo de entrada (in) y los siguientes en modo de salida (out), los tres trabajando con vectores de bits como se ha mencionado. El puerto ENT y CONT trabajan con cuatro bits (3 downto 0) y SAL con 7 bits (6 downto 0), es decir los bits más significativos están a la izquierda (revisar el capítulo 2 en los tipos de datos).

La arquitectura comienza en la línea 10, indicando el nombre de la arquitectura y del proyecto. En la línea 12 comienza proceso propiamente tal, utilizando **with**

select, comenzando en la línea 13. La interpretación es simple de entender, SAL tendrá el valor “0000001” cuando la entrada ENT sea igual a “0000” y el Display desplegará el cero en decimal. Dicha revisión se realizará para cada línea, hasta el número 9. La línea 23 indica que el Display permanece apagado para valores distintos a los de la tabla (OTHERS). En la línea 24 se selecciona el digito 1 (DG1) para mostrar los valores decimales, DG2, DG3 Y DG4 permanecen apagado, para esta práctica. Finalmente, el fin de la arquitectura y código termina en la línea 25.

4.13 MULTIPLEXOR DE CUATRO A UNA LINEA

Dentro del grupo de los circuitos combinacionales, tanto los multiplexores y demultiplexores son muy utilizados dentro del diseño de hardware para diversas aplicaciones. La presente practica muestra un multiplexor de cuatro a una línea. Antes de revisar la práctica, se revisará brevemente en que consiste el circuito multiplexor y demultiplexor.

Un multiplexor y demultiplexor son circuitos electrónicos combinacionales usados para gestionar el flujo de datos. Básicamente el multiplexor es un circuito digital formado por 3 tipos de buses, es decir el bus de entrada, el bus de salida y el bus de control. La función principal es seleccionar un bit del bus de entrada (dos o más bits) y transferir la información contenida al bus de salida (generalmente de 1 bit). La selección del bit de entrada lo determinara el valor binario del bus de control.

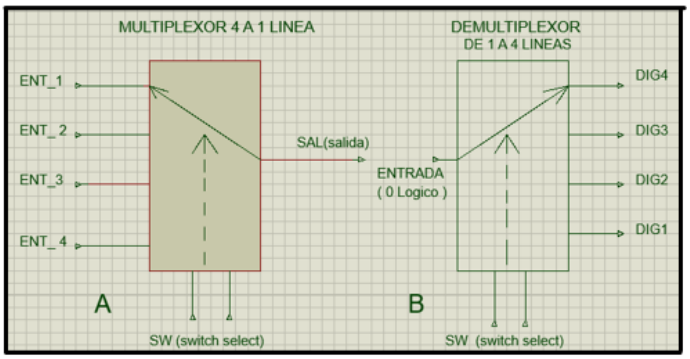


Figura 4-11_Concepto básico de un multiplexor y demultiplexor.

En la figura 4-11 (A) se muestra el concepto básico de un multiplexor de 4 a 1 líneas. El selector de switch entrega cuatro combinaciones posibles y cada una de ellas corresponde a la selección de una de las cuatro entradas que será transferida a la salida. Para apreciar las diferencias entre el multiplexor, en la figura 4-11(B) se visualiza un diagrama correspondiente a un demultiplexor de una a cuatro líneas. El concepto es idéntico, salvo que la entrada (0 lógico) ahora será transferida a una las 4 salidas (DIG1, DIG2, DIG3 Y DIG4). Posteriormente, en futuras prácticas, se realizará la interconexión de estos y otros circuitos en forma de diagramas de bloques (modo arquitectura), con el objeto de enlazar cada uno de los códigos VHDL tratados.

Como una primera practica se muestra a continuación un código VHDL que realiza la tarea de un circuito multiplexor de cuatro líneas de entrada a una salida (fig 4-12). Notar que cada entrada es una línea de un bit.

Las tres primeras líneas de código corresponden a la librería y paquetes a utilizar. La entidad comienza en la línea 4 y finaliza en la línea 13. El nombre de la entidad es DEMUX_4, las entradas de los puertos son desde ENT_1 hasta ENT_4, en modo IN y el tipo de datos es **std_logic** (datos solo de un bit) y la entrada de

modo IN y el tipo de datos es **std_logic** (datos solo de un bit) y la entrada de selector SW, es una palabra de dos bits (tipo de dato **std_logic_vector**). Por otro lado, la única salida SAL, con modo out y de un bit (**std_logic**).

La arquitectura comienza en la línea 14 y finaliza en la línea 21. Para seleccionar alguna de las entradas y que sea transferida a la salida, se utiliza "**with SW select"** en la forma indicada a partir de la línea 16, en consecuencia, la salida **SAL** tendrá el contenido de la ENT_1, cuando SW valga "00". La misma lógica se repite con los 3 canales restantes hasta la línea 20. La figura 14-11(A) ayuda a asociar el hardware con el código explicado anteriormente. En la figura 4-13 se aprecia la interpretación del hardware del código VHDL analizado, a través del diagrama RTL. Hay que recordar que este circuito multiplexor no será ensayado en la placa MAX-10 en forma individual y se agregará con otros archivos a través de diagramas de bloques, para un proyecto de mayor envergadura en los apartados siguientes.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;--LIBRERIAS NECESARIAS
3  USE IEEE.NUMERIC_STD.ALL;
4  ENTITY DEMUX_4 IS
5  PORT(
6  ENT_1: IN STD_LOGIC;         --LAS 4 ENTRADAS A SELECCIONAR DE UN BIT
7  ENT_2: IN STD_LOGIC;
8  ENT_3: IN STD_LOGIC;
9  ENT_4: IN STD_LOGIC;
10 SW:    IN STD_LOGIC_VECTOR(1 DOWNTO 0);--SELECCIONA EL CANAL
11 SAL:  OUT STD_LOGIC                    --LA SALIDA DEL CANAL SELECCIONDOR POR SW
12 );
13 END ENTITY;
14 ARCHITECTURE MUX OF DEMUX_4 IS       --NOMBRE DE ARQUITECTURA Y PROYECTO
15 BEGIN
16 WITH SW SELECT
17 SAL<=ENT_1 WHEN "00",          --LA SALIDA TENDRA LA ENTRADA SELECCIONADA POR sw
18       ENT_2 WHEN "01",
19       ENT_3 WHEN "10",
20       ENT_4 WHEN "11";
21 END ARCHITECTURE;
22
23
```

Figura 4-12_Codigo VHDL de un multiplexor de 4 a 1 línea.

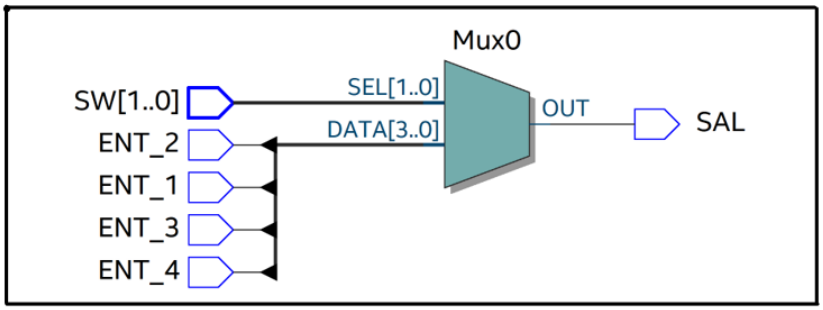


Figura 4-13_Diagrama RTL del multiplexor de 4 a 1 línea.

4.14 MULTIPLEXOR Y DEMULTIPLEXOR DE CUATRO A UNA LINEA Y DE UNA A CUATRO LINEAS RESPECTIVAMENTE.

En la siguiente practica se revisará un código similar al anterior, pero con dos grandes diferencias, la primera diferencia será un código escrito para un multiplexor y demultiplexor en el mismo proyecto (más adelante será un archivo que formará parte de un proyecto mayor) y la segunda diferencia, consiste en realizar un multiplexor con cuatro líneas de entrada, cada una de ellas formada por un array de 4 bits y una salida formada de la misma forma. Para mayor claridad en la figura 4-14 se aprecia el hardware propuesto.

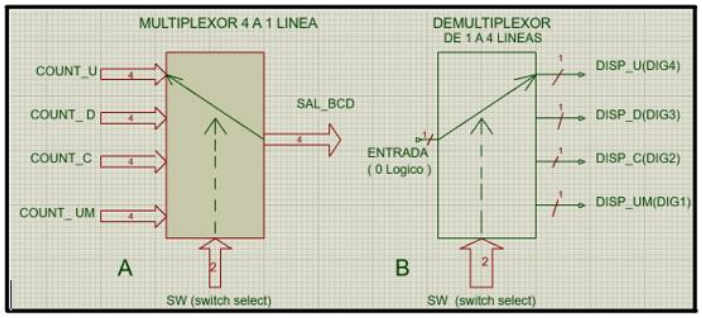


Figura 4-14_ Representación del hardware de un multiplexor y demultiplexor.

La figura 4-15 muestra el código VHDL, asociado a la figura 4-14(A), en el cual se multiplexará los buses de entrada de 4 bits de COUNT_U hasta COUNT_UM, para ser transferido uno de ellos a la salida SAL_BCD, dependiendo del valor de SW. En la figura 4-14(B), se visualiza el demultiplexor en el cual la salida seleccionada por el valor de SW (switch select) obtendrá un 0 lógico.

En las primeras 3 líneas de código, se muestran las librerías y paquetes requeridos. La entidad comienza en la línea 4 y finaliza en la línea 18 y en ella

se visualiza las líneas de entrada y salida del hardware indicado en la figura 4-14 A y 4-14 B, correspondiente al multiplexor y demultiplexor respectivamente.

Desde la línea 7 a 12, dichos puertos se declaran tipo de datos **std_logic_vector**(puertos de 4 bits) y desde la línea 13 a la línea 16 son de un bit(**std_logic**). Los modos en cada uno de ellos son de entrada o salida (IN, OUT O INOUT). Al revisar detalladamente el hardware de las figuras 4-14 se puede asociar fácilmente los tipos de datos y sus modos con el código VHDL. Entre las lineas 19 y 48 esta delimitada la arquitectura de dicho código. Entre la línea 21 y 26 está el código del multiplexor con el **with sw select** correspondiente. El demultiplexor comienza en la línea 27 y finaliza en la línea 47. Es importante reiterar que el propósito del demultiplexor es seleccionar uno de los 4 DISP_X con un 0 lógico en su salida, lo cual lo determina el valor **SW** (switch select).

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  entity MUX4_1 is
5  port(
6
7      SW:      IN std_logic_VECTOR(1 DOWNTO 0); --CONTROL DEL MUX
8      COUNT_U: IN STD_LOGIC_VECTOR(3 DOWNTO 0);--CONTADOR UNIDADES
9      COUNT_D: IN std_logic_VECTOR(3 DOWNTO 0);-- DECENAS
10     COUNT_C: IN STD_LOGIC_VECTOR(3 DOWNTO 0);-- CENTENAS
11     COUNT_UM: IN std_logic_VECTOR(3 DOWNTO 0);-- UNIDAD DE MIL
12     SAL_BCD:INOUT std_logic_VECTOR(3 DOWNTO 0);--SAL HACIA BCD/7SEG
13     DISP_U: OUT STD_LOGIC;                    --SELECCIONA EL DISP DE LAS UNIDADES
14     DISP_D: OUT STD_LOGIC;                    --          "          DECENAS
15     DISP_C: OUT STD_LOGIC;                    --          "          CENTENAS
16     DISP_UM: OUT STD_LOGIC                    --          "       UNIDAD DE MIL
17 );
18 end entity;
19 architecture comp of MUX4_1 is
20 begin
21     WITH SW SELECT                  --AQUI COMIENZA EL MULTIPLEXOR (FIG 4.15A)
22     SAL_BCD<= COUNT_U   WHEN "00",  --CON CERO SELECCIONA UNIDADES
23               COUNT_D   WHEN "01",
24               COUNT_C   WHEN "10",
25               COUNT_UM  WHEN "11";
26 --------------------------------

27     WITH SW SELECT                        -------------------------
28     DISP_U<= '0' WHEN "00",              --SELECCION DISPLAY UNIDADES
29              '1' WHEN "01",
30              '1' WHEN "10",
31              '1' WHEN "11";
32     WITH SW SELECT
33     DISP_D <= '1' WHEN "00",             --SELECCION DECENAS
34               '0' WHEN "01",
35               '1' WHEN "10",
36               '1' WHEN "11";
37           WITH SW SELECT                          --------------------
38     DISP_C<= '1' WHEN "00",              --SELECCION DISPLAY CENTENAS
39              '1' WHEN "01",
40              '0' WHEN "10",
41              '1' WHEN "11";
42     WITH SW SELECT
43     DISP_UM <= '1' WHEN "00",             --SELECCION UNIDAD DE MIL
44                '1' WHEN "01",
45                '1' WHEN "10",
46                '0' WHEN "11";
47
48 END ARCHITECTURE;
```

Figura 4-15_Codigo VHDL correspondiente a un multiplexor y demultplexor.

Es importante tener presente que los DISP_X no seleccionados, se les asigna un 1 lógico. En el primer caso fue seleccionado el DISP_U, con el SW = “00” y para valores distintos a “00” los DISP_X restantes se deshabilitan con un 1 lógico (líneas 29 a 31). Para la selección de los DISP_X restantes se repite el procedimiento con **with sw select**.

4.2 CIRCUITOS SECUENCIALES

Para comenzar a revisar el diseño de circuitos secuenciales con lenguaje VHDL es prioridad revisar el concepto básico de estos. Un circuito secuencial es un circuito digital en el cual la salida depende de las entradas actuales y los valores pasados de dichas entradas, debido a la realimentación de las salidas hacia las entradas, sumado a ello componentes de memoria. Los circuitos secuenciales están formados por entradas y salidas, señales de reloj (CLK) es decir señales periódicas encargadas de sincronizar los cambios de estados, lógica combinacional encargada de determinar la salida según las entradas actuales y las existentes en memoria y finalmente los elementos de almacenamiento o memorias encargados de mantener el estado actual del circuito.

4.21 DIVISOR DE FRECUENCA CON SALIDA DE 1Hz

El presente proyecto consiste en obtener una señal cuadrada de 1 Hz, con el propósito que sea visible un parpadeo de un led a la salida de la placa FPGA MAX-10, a partir del Clock (50 MHZ) interno de la MAX-10 (pin 26) y el pin 93 para energizar el diodo led 1. Para mayor detalle se recomienda consultar el manual

de la placa en el apéndice A. La figura 4-16 entrega una representación del hardware a implementar. Es importante indicar que los códigos VHDL aquí descritos, son portable a otro tipo de placas (tanto FPGA y CPLD) y solo se realiza el cambio de pines en menú **assignments**, **pinplaner** de Quartus II. Para ello es imprescindible tener la información de a lo menos los puertos disponibles, el número de pin del Clock y frecuencia de reloj de la tarjeta a utilizar.

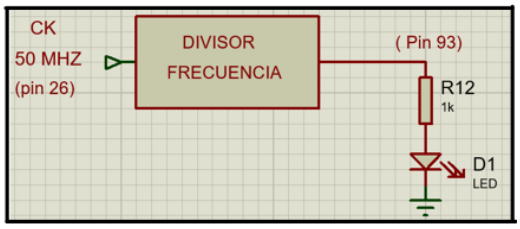


Figura 4-16_ Representación de un divisor de frecuencia.

El presente código es un divisor por un factor de 50.000.000 de veces, debido que la frecuencia de reloj de la placa MAX-10 posee un cristal de 50 Mhz y la frecuencia de requerida a la salida del divisor es de 1 Hz.

Para revisar el presente código se debe tener el conocimiento entregado en el capítulo 2, específicamente los procesos (**process**), el uso de los objetos, **constantes, señales** y el uso del bucle **if-then**.

La estructura del código de la figura 4-18 básicamente es similar a lo revisado hasta ahora, es decir el uso de las librerías, la declaración en la entidad, salvo que se agrega IEEE.NUMERIC_STD.ALL, paquete usado para operaciones numéricas (línea 3). Cambios se observan en la arquitectura, específicamente en la línea 12 donde aparece un objeto llamado CONSTAT que para este caso tiene el nombre COUNT_LIMIT, con un tipo de dato **integer** y con un valor asignado

de 25.000.000. Un segundo objeto es una señal **(signal)** que lleva el nombre de COUNT, con tipo de dato **integer** y un rango de 0 a 50.000.500, lo cual permitirá a COUNT tener valores en ese rango. La señal LED_STATE es algo muy similar a un registro auxiliar, pero sin mapeo físico. En palabras simples el contador COUNT incrementa la cuenta de pulsos de CLK, hasta alcanzar los 25.000.000, para cambiar de estado la salida del divisor (salida_1HZ) y reiniciar el contador COUNT, para comenzar a contar 25.000.000 de pulsos de CLK nuevamente, hasta volver a cambiar la salida al estado opuesto al anterior. De este modo la SALIDA_1HZ, estará cambiando de estado cada 0,5 segundo con un periodo total de 1 segundo, obteniendo como resultado final una salida de 1 Hz con un ciclo de trabajo de un 50%, para mayor claridad la figura 4-17 muestra la idea general.

Regresando al código, la línea 16, inicia el **process** (clock), es decir el proceso revisara cuando el Clock tenga un flaco de subida (rissing Edge) y comienza el primer bucle (línea 18), si la condición es verdadera, es decir existe un flaco de subida, entra a un segundo bucle el cual consultara si COUNT alcanzo la cuenta de 25 millones de pulsos de COUNT_LIMIT, si es afirmativo cambia el estado de la salida a través de la señal LED_STATE y COUNT vuelve a cero (línea 22). El caso opuesto es que COUNT no alcanza la cuenta de 25 millones y el COUNT se incrementa en una unidad (línea 24) y se repiten los bucles. Un detalle importante, para bajar el factor de división y obtener frecuencias mayores a 1 Hz en la salida, se debe reducir el valor de COUNT_LIMIT, es decir para una frecuencia deseada de 10 Hz, COUNT_LIMIT debe tener un valor de 2, 5 millones de pulsos (factor de· división de 5.000.000). De acuerdo a lo analizado anteriormente se determina la siguiente relación:

COUNT_LIMIT=FRECUENCIA DE CK/(FACTOR DIVISION /2)

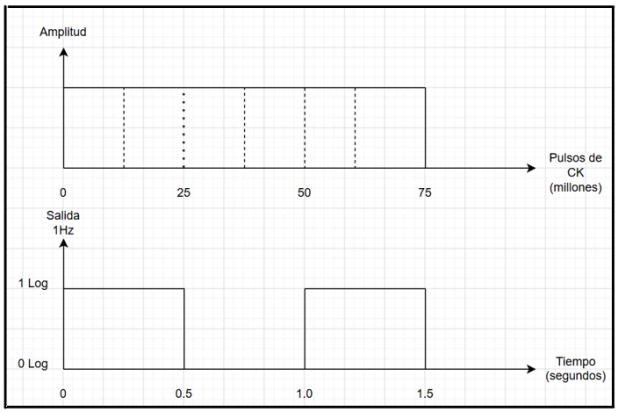


Figura 4-17_Cambio de estado en la Salida 1Hz, cada 25.000.000 pulsos de CLK.

Finalmente, en la figura 4-19, se muestra el reporte o resultado de la compilación del proyecto. Entre la información más importante a observar es el número de pines y el número de elementos lógicos usados en el presente proyecto. A medida que se complejiza un proyecto, mayor es la cantidad de elementos lógicos utilizado y el diseñador debe tener en consideración la capacidad del dispositivo FPGA con el cual trabaja. La Figura 4-20 entrega la información de los pines utilizados en el FPGA.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4  ENTITY DIVISOR_1HZ IS
5  PORT(
6        CLOCK: IN STD_LOGIC;
7
8        SALIDA_1HZ:OUT STD_LOGIC
9        );
10 END ENTITY;
11 ARCHITECTURE FREC_DIV OF DIVISOR_1HZ IS--COMIENZA ARQUITECTURA CON NOMBRE FREC_DIV
12 CONSTANT COUNT_LIMIT: INTEGER:=25000000;--3 OBJETOS 2 SEÑALES Y 1 CONSTANTE
13 SIGNAL COUNT: INTEGER RANGE 0 TO 50000500:=0;--CONSTAN DEFINE EL FACTOR DE DIVISION, RANGO 1 SEG
14 SIGNAL LED_STATE: STD_LOGIC;
15 BEGIN
16 BLINK:PROCESS(CLOCK)      --COMIENZA EL PROCESO CLOCK (PROCESS)
17 BEGIN
18 IF RISING_EDGE(CLOCK) THEN --DETECTA FLANCO DE SUBIDA DEL CLOCK
19 IF(COUNT=COUNT_LIMIT) THEN --HAY FLANCO ENTONCES, REVISA LA CUENTA COUNT SI ALCANZO 25000000
20 LED_STATE<=NOT LED_STATE;  -- CAMBIAR EL ESTADO DE LA SALIDA
21 SALIDA_1HZ<= NOT LED_STATE;--ELCONTENIDO DE LA SEÑAL LA TRANSFIERE A LA SALIDA
22 COUNT<=0;                  --COMO ALCANZO EL VALOR LIMITE, VUELVE A CERO
23 ELSE
24 COUNT<= COUNT+1;           --AUN NOLLEGA A 25000000 PULSOS DE CLOCK
25 END IF;
26 END IF;
27 END PROCESS;
28 END ARCHITECTURE;
```

Figura 4-18_Código divisor de frecuencia, con salida de 1Hz.

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Mar 16 01:27:06 2026
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	DIVISOR_1HZ
Top-level Entity Name	DIVISOR_1HZ
Family	MAX 10
Device	10M08SCE144C8G
Timing Models	Final
Total logic elements	51 / 8,064 (< 1 %)
Total registers	28
Total pins	2 / 101 (2 %)
Total virtual pins	0
Total memory bits	0 / 387,072 (0 %)
Embedded Multiplier 9-bit elements	0 / 48 (0 %)
Total PLLs	0 / 1 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0

Figura 4-19_Resumen del resultado de la compilación del proyecto.

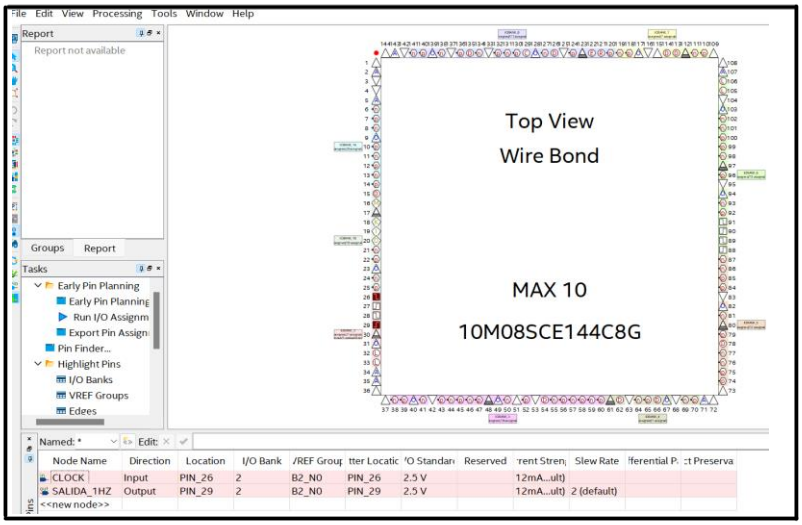
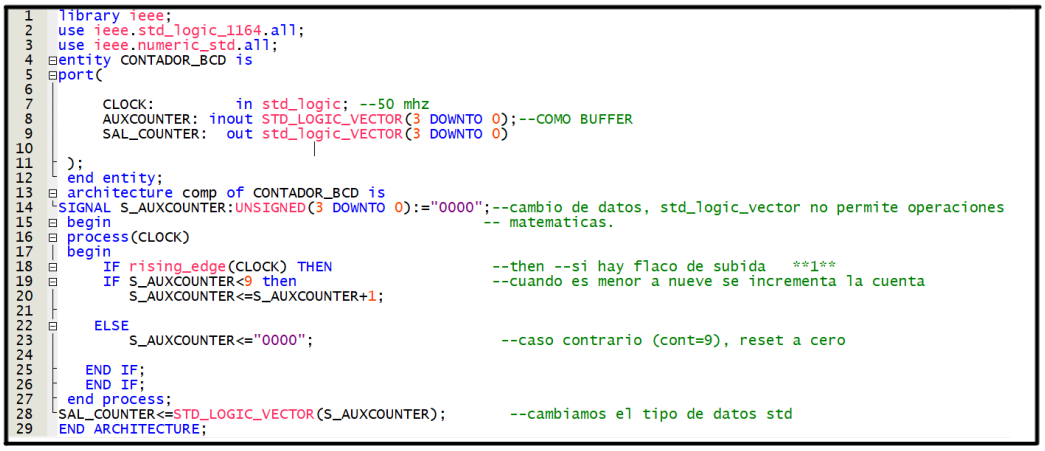


Figura 4-20_Pines utilizados en el FPGA.

4.22 CONTADOR BCD

El presente practico consiste en realizar un ensayo de un contador BCD (de 0 a 9), con pulsos de reloj extraídos directamente de la placa MAX-10, con una frecuencia de 50 Mhz. Una vez alcanzada la cuenta al valor “1001” (9) deberá regresar a cero.

En el código de la figura 4-21, se aprecia la entidad con 3 puertos, donde CLOCK, es la entrada de reloj, SAL_COUNTER la salida del conteo en BCD, con modo de entrada (IN) y salida (OUT) respectivamente y sumado a ello un cuarto elemento AUXCOUNTER, con un modo entrada-salida (INOUT), utilizado como buffer para realimentar la salida. Los tipos de datos son STD_LOGIC_VECTOR (3downto 0), arrays de 4 bits, para ambos registros. El clock como es un solo bit tiene el tipo de datos STD_LOGIC. Dentro de la arquitectura, se incorpora una señal (signal) definida por el programador como S_AUXCOUNTER con tipo de dato unsigned, para poder realizar las operaciones aritméticas. En VHDL los grupos de bits (std_logic_vector) no permite realizar operaciones aritméticas en forma directa, con estos paquetes de librería, por lo que es necesario realizarlas con la señal (**signal**) AUX_COUNTER.



```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  entity CONTADOR_BCD is
5  port(
6
7       CLOCK:          in std_logic; --50 mhz
8       AUXCOUNTER: inout STD_LOGIC_VECTOR(3 DOWNTO 0);--COMO BUFFER
9       SAL_COUNTER:  out std_logic_VECTOR(3 DOWNTO 0)
10
11 );
12 end entity;
13 architecture comp of CONTADOR_BCD is
14 SIGNAL S_AUXCOUNTER:UNSIGNED(3 DOWNTO 0):="0000";--cambio de datos, std_logic_vector no permite operaciones
15 begin                                          -- matematicas.
16 process(CLOCK)
17 begin
18      IF rising_edge(CLOCK) THEN                --then --si hay flaco de subida   **1**
19      IF S_AUXCOUNTER<9 then                    --cuando es menor a nueve se incrementa la cuenta
20         S_AUXCOUNTER<=S_AUXCOUNTER+1;
21
22     ELSE
23         S_AUXCOUNTER<="0000";                  --caso contrario (cont=9), reset a cero
24
25    END IF;
26    END IF;
27 end process;
28 SAL_COUNTER<=STD_LOGIC_VECTOR(S_AUXCOUNTER);       --cambiamos el tipo de datos std
29  END ARCHITECTURE;
```

Figura 4-21_ Código VHDL contador BCD.

Una vez declarada la señal, se procede a escribir el proceso denominado clock(línea 16), debido a que **process** revisara la existencia de flancos de subida (rissing edge) del clock, tal como lo muestra el primer bucle en la línea 18. Una vez ocurrido el flanco, continua un segundo bucle para consultar si S_AUXCOUNTER aun no llega a 9, si es menor a 9, S_AUXCOUNTER se incrementa en 1, caso contrario vuelve a cero. Finalizada la operación aritmética de incremento de S_AUXCOUNTER, en la línea 28 es transferido el valor de S_AUXCOUNTER hacia la salida (SAL_COUNTER), cambiando el tipo de dato a **std_logic_vector**.

En la figura 4-22 se distingue la distribución de pines (pin planer) a utilizar en el dispositivo FPGA. Es importante recordar que antes de programar la placa se debe compilar el proyecto completo.

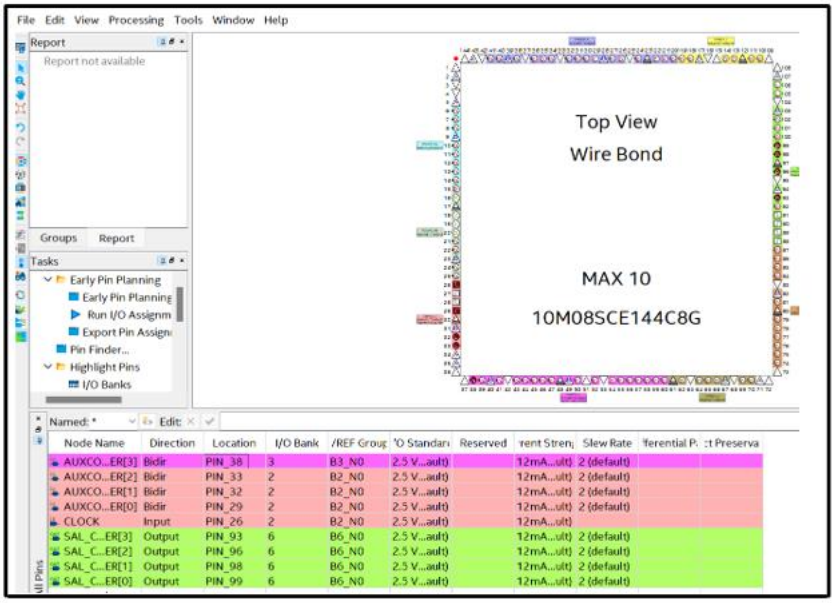


Figura 4-22_Pin planer del contador BCD.

En la figura 4-23 se puede observar el resultado de la compilación del proyecto completo. Es importante tener en consideración la cantidad de elementos lógicos utilizados. Para este caso en particular apenas se utilizaron 5.

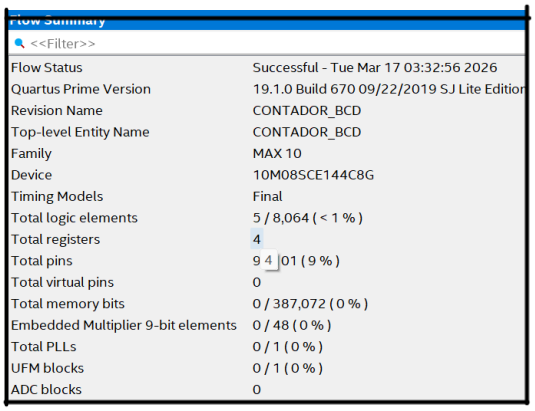


Figura 4-23_Resumen de la compilación del proyecto.

4.23 MODULADOR ANCHO DE PULSO (PWM)

El modulador ancho de pulso es un circuito utilizado en múltiples aplicaciones, básicamente consiste en modificar el ciclo de trabajo de una onda cuadrada de frecuencia constante. En este código desarrollado se visualiza su funcionamiento a través de la placa MAX-10. El control del ancho de pulso se realizará con el Dip switch externo de 8 pines (figura 4-24), por consiguiente, el ancho del pulso tendrá 255 valores posibles.

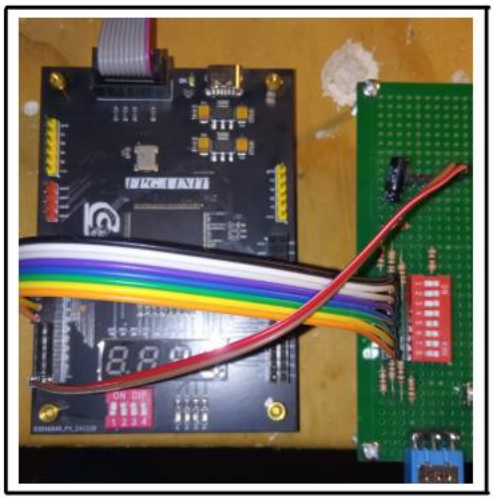


Figura 4-24_Placa MAX-10 conectada a Dip switch de 8 pines externo para realizar el control PWM.

El control PWM se ejecuta sobre una onda cuadrada con periodo de 4 milisegundos, como lo indica la figura 4-26, cuyo ancho del pulso (ciclo de trabajo) lo determina la comparación entre el valor binario entregado por el dip switch de 8 pines y el contenido del contador PWM, tal como se observa en la figura 4-25.

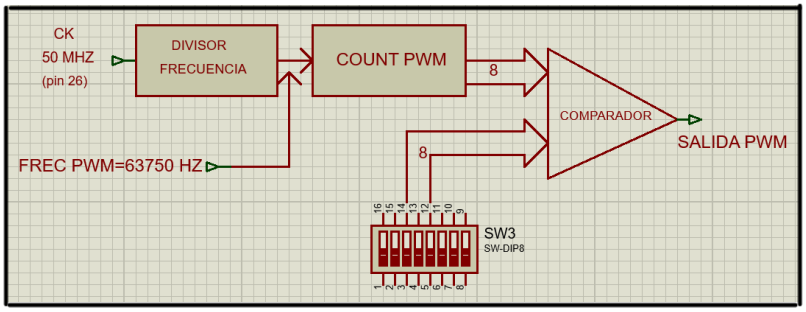


Figura 4-25_Hardware propuesto control PWM.

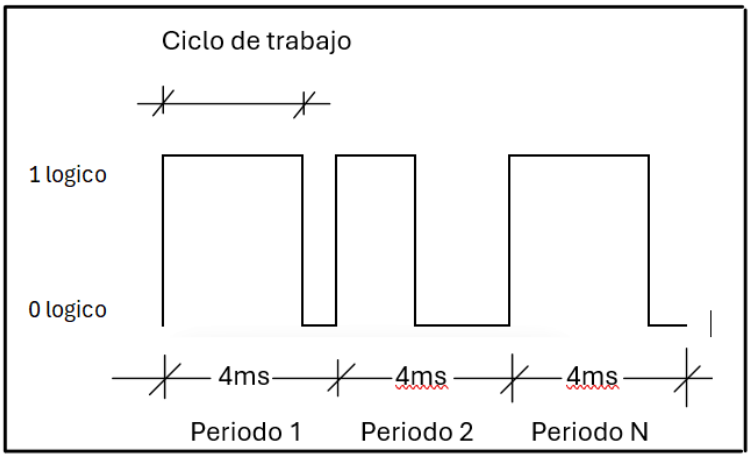


Figura 4-26_Forma de onda de PWM propuesto.

La frecuencia PWM se obtiene a través del clock interno de la placa MAX-10 y el valor de 36.750 Hz se calcula con las siguientes expresiones.

FRECUENCIA PWM= 1/(Tpwm) ó FRECUENCIA PWM = 1/(4 ms/255)

Donde:

Tpwm= 4ms/255.

4 ms: Es el periodo de la onda a modular, valor establecido arbitrariamente para esta práctica.

255: el número de posibles anchos de pulsos que se pueden obtener en la modulación, esto debido a que se trabaja con 8 bits en el contador PWM y el Dip switch. Dicho de otra forma, el contador PWM puede llegar a contar hasta 255 valores en un tiempo de 4 ms, lo que arroja un periodo de 15,7 microsegundos por cada cuenta, en consecuencia, es necesario una frecuencia de reloj de 63.750 Hz a la entrada del contador PWM.

Otro dato muy importante es el factor de división, el divisor de frecuencia y está dado por.

FACTOR DIVISION=5O MHZ/FRECUENCIA PWM

Para este caso en particular es de 784.

En la figura 4-27 se aprecia el código correspondiente al hardware propuesto en la figura 4-25, donde en las primeras líneas se aprecia las librerías utilizadas. En la línea 6 utilizo un paquete muy poco usado para tratamiento numérico, pero en este caso es usado para leer los datos directamente del Dip switch. La entidad comienza en la línea 7 y finaliza en la 15, con 4 puertos. DIP_8 es asignado para el Dip switch externo, con el tipo de datos **std_logic_vector** en modo entrada (IN). La arquitectura comienza en la línea 16, declarando el nombre de la arquitectura y del proyecto como es habitual. En la línea 17 comienza la declaración de 4 objetos, una constante y 3 señales, con diversos tipos de datos. Esta arquitectura es un poco más extensa que las anteriores y se puede subdividir en 3 partes, la primera con la declaración de objetos, la segunda con el código correspondiente al divisor de frecuencia y la tercera parte es el modulador PWM propiamente tal. En la línea 23 comienza el divisor de frecuencia con un **process(clock)** (proceso ya visto con anterioridad), el cual realiza el chequeo de un flanco de subida, entrando a un primer bucle (if-then), una vez ocurrido un flaco de subida, se pasa al segundo

bucle para consultar si el contenido de la señal (**signal**) COUNT_DIV alcanzo el valor COUNT_LIMIT, para hacer el cambio de estado en la salida del divisor (SAL_63KHz). Hay que recordar que, en el divisor de frecuencia en lenguaje VHDL, el valor del COUNT_LIMIT está dado por la siguiente relación:

COUNT_LIMIT=FRECUENCIA DE CK/(FACTOR DIVISION /2)

Donde la frecuencia del clok es de 50 Mhz y el factor de división es de 784, en consecuencia, aplicando la relación, el valor del límite del conteo es de 392 contenido por constante (línea 17). Una vez que COUNT_DIV sea igual a COUNT_LIMIT (392), la salida de 63 Khz cambiara de estado y COUNT_DIV se reinicia cero. Caso contrario COUNT_DIV se incrementará en una unidad y el bucle se repite hasta que exista la igualdad entre ambas señales (COUNT_DIV igual a COUNT_LIMIT). En la línea 34 finaliza el proceso del divisor de frecuencia.

El generador PWM comienza en la línea 37 con un nuevo proceso llamado **process**(salida_63Khz). El bucle **rissing Edge** (salida_63khz) revisa la presencia de un flanco de subida proveniente del divisor de frecuencia que entrega un tren de pulsos con una frecuencia de 63.750 Hz. Una vez que la afirmación sea verdadera, en la línea 41 el bucle compara si el valor contador **pwm** (**COUNT_PWM**) es menor que el valor en el Dip switch (8), si la comparación es verdadera, **COUNT_PWM** se incrementa en 1 y la salida PWM se mantiene en 1 lógico, caso contrario la salida pasa a 0 lógico, pero la cuenta del contador PWM se sigue incrementado hasta llegar al valor de 255, donde el contador PWM pasa a cero y comienza todo el proceso nuevamente. El bucle en la línea 48 realiza la comparación mencionada. Posterior a ello se cierran los bucles y el circuito comienza a repetir el proceso desde la línea 41.

En la figura 4-28 se muestra la distribución de pines asignados al FPGA, es decir los puertos definidos en la entidad. El resumen de la compilación del proyecto completo se puede observar en la figura 4.29, notar la cantidad de elementos lógicos utilizados, un gran aumento con respecto al proyecto contador BCD. Finalmente, en la figura 4-30 se muestra una imagen real de la presente práctica.

```
--CONSISTE EN CONTROLAR EL ANCHO DEL PULSO (CICLO DE TRABAJO), CON UN DIPSWITCH DE 8
--EL PERIODO ESCOGIDO ES DE 4 miliseg, CON 255 CUENTAS, EXTRAIDO DEL CK DE 50 MHZ, P
 LIBRARY IEEE;
 USE IEEE.STD_LOGIC_1164.ALL;
 USE IEEE.NUMERIC_STD.ALL;
 USE IEEE.STD_LOGIC_UNSIGNED.ALL;--REVISAR LA LIBRERIA
ENTITY PWM_DIP IS
PORT(
      CLOCK: IN STD_LOGIC;
      DIP_8: IN STD_LOGIC_VECTOR(7 DOWNTO 0);--DEFINEN EL ANCHO DEL PULSO

      SALIDA_63KHZ:inOUT STD_LOGIC;            --SALIDA DEL DIVISOR, USADA PARA CHEQU
      SAL_PWM:OUT STD_LOGIC              --SALIDA PWM
      );
END ENTITY;
ARCHITECTURE FREC_DIV OF PWM_DIP IS
CONSTANT COUNT_LIMIT: INTEGER:=392;--SE OBTIENE UN CICLO DE 63750 HZ, 4 mSEG
SIGNAL COUNT_DIV: INTEGER RANGE 0 TO 500005000:=0;--CONSTAN DEFINE EL FACTOR DE DIV
SIGNAL LED_STATE: STD_LOGIC;
SIGNAL COUNT_PWM: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000"; --DEFINIMOS RANGO DE LA
BEGIN
--DIVISOR DE FRECUENCIA
BLINK:PROCESS(CLOCK)
BEGIN
IF RISING_EDGE(CLOCK) THEN
IF(COUNT_DIV=COUNT_LIMIT) THEN --CUANDO LLEGA A LA CTA LIMITE CAMBIA EL ESTADO (255
LED_STATE<=NOT LED_STATE;
SALIDA_63KHZ<=LED_STATE;--AGREGAMOS OTRO PIN PARA 2HZ
COUNT_DIV<=0;
ELSE
COUNT_DIV<= COUNT_DIV+1;
END IF;
END IF;
END PROCESS;
--GENERADOR PWM

BLINK2:PROCESS(salida_63Khz)
BEGIN
IF RISING_EDGE(salida_63Khz)THEN
---------------------------------- --REVISA SI HAY PULSO EN SALIDA DE DIVISOR 255
IF(COUNT_PWM< DIP_8) THEN             --MANDA UN 1 SI COUNT_PWM ES MENOR QUE DIP_8
COUNT_PWM<=COUNT_PWM+"00000001";      --INCREMENTA EN 1 COUNT_PWM
SAL_PWM<='1';
ELSE                                  --CASO CONTRARIO COUNT_PWM > DIP_8
SAL_PWM<='0';                         -- MANDA UN 0
COUNT_PWM<=COUNT_PWM+"00000001";      --SIGUE INCREMENTADO COUNT_PWM HASTA 255
-----------------------------------------------------------------------
IF(COUNT_PWM >"11111111") THEN --MAYOR A 255
COUNT_PWM<= "00000000";        --MANDA UN 1 A LA SALIDA
SAL_PWM<='1';
ELSE
COUNT_PWM<=COUNT_PWM+"00000001";
END IF;
END IF;
END IF;
END PROCESS;
END ARCHITECTURE;
```

Figura 4-27_Código VHDL del PWM.

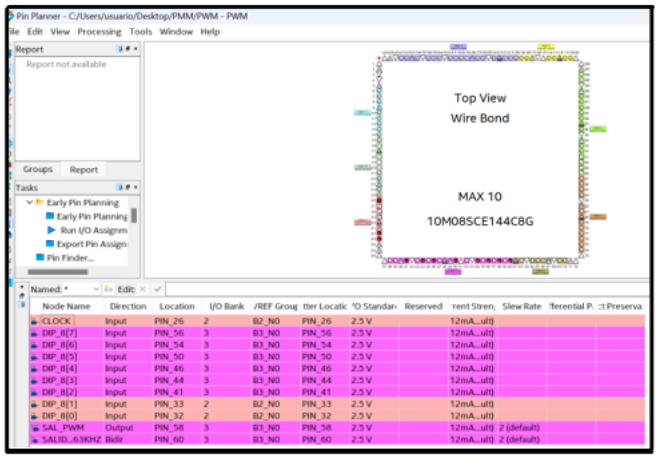


Figura 4-28_Pines asignados al FPGA (pin planer).

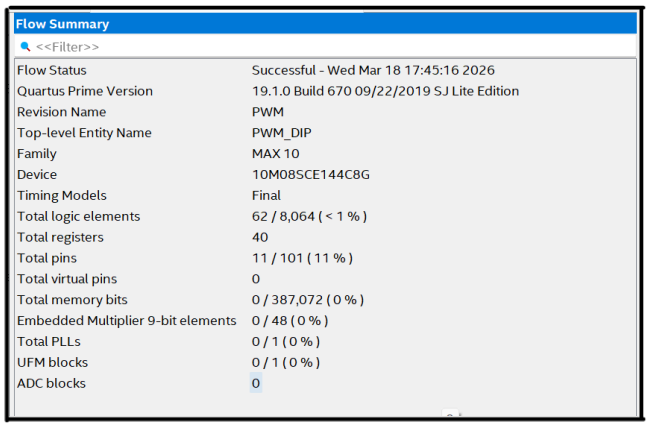


Figura 4-29_Resumen de la compilación del proyecto.

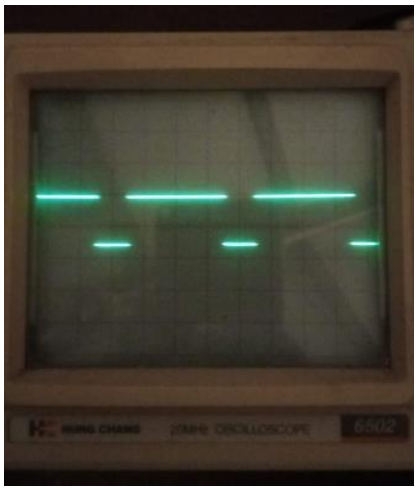


Figura 4-30_Muestra visualizada en osciloscopio de la señal PWM.

4.24 VISUALIZACION DE UN MENSAJE EN DISPLAY LCD 2x16

La presente experiencia consiste en visualizar un breve mensaje en un Display LCD de 2X16 líneas, a través la placa de desarrollo Max-10 mostrada en la figura 4-31, donde se aprecia el conexionado entre la pantalla y los puertos I/O a utilizar en la placa. Antes de analizar el correspondiente código VHDL, es imprescindible tener un conocimiento de cómo se envían los datos y comandos hacia la pantalla de cristal líquido, así como también manejar una breve descripción de los pines y conexionado.

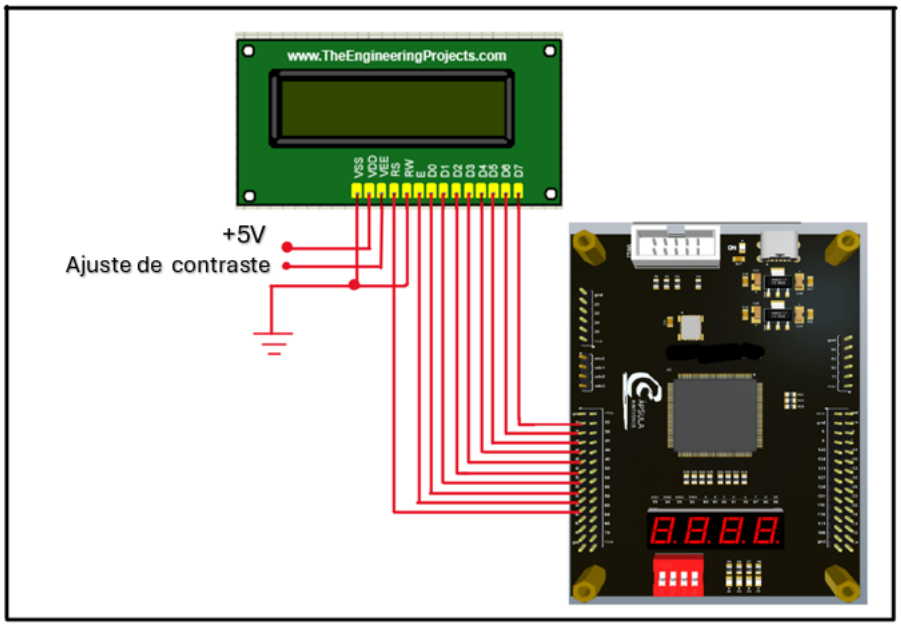


Figura 4-31_Diagrama de conexionado de una pantalla LCD con la placa MAX-10

El control de una pantalla LCD (Liquid Crystal Display) implica trabajar con pines de datos y control (RS, R/W, E), para mostrar texto o gráficos, usando como elemento de control el dispositivo FPGA contenido en la placa MAX-10. Se

utilizan comandos en binario (o hexadecimal) para configurar funciones como borrar pantalla, posicionar el cursor y ajustar el contraste, siendo común el uso de interfaces paralelas de 4 u 8 bits. En este caso en particular, se revisará con interface de 8 bits.
A modo de resumen, es importante tener presente los cuatro conceptos fundamentales, para configurar un LCD.

Pines de Control: Son los encargados de configurar el funcionamiento del módulo LCD tales como el tipo de dato (comando o dato), el modo escritura y lectura y la habilitación o des habilitación del LCD.

Modos de Datos: Se puede usar el bus de datos completo (8 bits: D0-D7) o modo económico (4 bits: D4-D7) para ahorrar pines.

Inicialización: Al energizar el módulo, es crucial enviar una secuencia de comandos específicos con retardos de tiempo para configurar parámetros como el número de líneas y el tamaño de la fuente.

Memoria: Incluye la **DDRAM** (para los caracteres a mostrar) y la **CGRAM** (para definir caracteres personalizados). En el presente trabajo no se utilizaron caracteres personalizados.

En la figura 4.32 se observa la conexión del módulo LCD, conectado a los puertos I/O de la placa Max-10, así como también cada pin del LCD que se describe a continuación:

Vss: Conexión a gnd.

Vdd: Entrada de tensión de alimentación. Para este ensayo de alimenta desde la placa MAX-10.

Vee: Se aplica una tensión ajustable, para regular la intensidad de los caracteres.

Registrer Select (RS): Registro encargado de definir si el LCD trabaja en modo comando o datos. Con valor 0 lógico, trabaja en modo comando, es decir los

datos presentes en el bus de data son palabras de configuración del módulo. Un 1 lógico establece que los bits presentes en el data bus es datos propiamente tal.

Read Writhe (RW): Determina si se lee algún contenido del módulo o se escribe sobre el módulo. Para RW=0, se escribe sobre el módulo y para RD=1, se lee información del módulo. Para el presente ensayo solo se escriben datos (RW=0).

Enable (E): Es encargado de habilitar o deshabilitar el módulo LCD. Para este caso E=0.

Data (D07-D00): Son los datos para enviar y recibir hacia o desde el módulo LCD. Estos pueden ser comando o datos.

4.241 INICIALIZACION Y CONFIGURACION.

Para comenzar con la inicialización del módulo LCD, el hardware propuesto debe estar debidamente armado, en forma similar a la figura 4.32. Como se ha mencionado anteriormente el código VHDL revisado aquí es válido para cualquier FPGA o CPLD incluida en Quartus II, en consecuencia, en otros dispositivos y placas solo cambia la asignación de pines I/O y Clock.

A continuación, se describe en forma conceptual el orden y el procedimiento de la puesta en marcha de un LCD, según lo recomendado por el fabricante, con datos a modo de ejemplo, de manera de facilitar la comprensión del código VHDL. Se adjunta en el apéndice una tabla de comandos con mayores detalles entregados por el fabricante.

El primer paso es la inicialización y consiste en generar un retardo de 40 ms, posterior a la aplicación de la alimentación y como paso a seguir es aplicar **función set (DB5=1)** tal como se muestra en la tabla 4.1

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	E
FUNCION SET	0	0	0	0	1	1	0	0	X	X	1

Tabla 4-2_Funcion set.

RS= 0 Modo comando. **DB5= 1** Función set. **DB2= 0** 5x7 carácter

R/W= 0 Modo escritura. **DB4= 1** 8 Bits de datos **DB1= X** Da lo mismo

E= se debe poner a 1 **DB3= 0** 1 línea **DB0=X** Da lo mismo.

Una vez revisado los datos deseados, el controlador debe esperar un tiempo prudente de 40 ms para poder enviar 1 lógico a **Enable (E)** de unos 50 ms de duración para poder dejar configurado en 8 bits de datos, 1 línea lógica y carácter de 5X7.Paso seguido se debe aplicar el **Comando Show** (**DB3=1)** para apagar Display**,** tal como se muestra en la tabla 4.2.

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	E
COMAND SHOW	0	0	0	0	0	0	1	0	0	0	1

Tabla 4-3_Comando Show.

Una vez cargado el nuevo comando (comand show), con **DB2=0**, se aplica tiempo de espera de 20 ms, para aplicar Enable con un tiempo de 50 ms de duración. Paso seguido se debe limpiar la memoria del LCD, con el comando **Clear Display(DB0=1),** tal como lo muestra la tabla 4.3.

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	E
CLEAR DISPLAY	0	0	0	0	0	0	0	0	0	1	1

Tabla 4-4_Comando clear display.

Para proseguir se recomienda esperar 20 ms para después aplicar Enable por un tiempo de otros 50 ms. Posteriormente se debe aplica el comando **Entry Mode (DB2=1)** consistente en desplazar o no el cursor para la derecha o izquierda y desplazar o no un bloque de caracteres. Para este caso **DB1=1** incrementa el

cursor hacia la derecha, sin desplazamiento de la pantalla completa (**DB0=0**), tal como lo indica la tabla 4.4. Una vez cargados los datos, se repite una espera de 20 ms, para aplicar E=1 por 20 ms de duración.

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	E
ENTRY MODE	0	0	0	0	0	0	0	1	1	0	1

Tabla 4-5_ Comando Entry Mode.

Posteriormente, se vuelve a escribir en **Comand Show**, con el propósito de encender el LCD, con **DB2=1,** aplicando la misma espera de 20 ms, antes aplicar un 1 lógico a Enable por un tiempo de 50 ms. Una vez finalizado este proceso, se procede a enviar datos (o caracteres ASCII) al LCD, para que sean desplegados en pantalla, saliendo del modo comando para cambiar al modo datos, aplicando un 1 lógico al terminal RS (RS=1) y con los mismos tiempos de espera entre el envío de datos o comandos, como lo indica la tabla 4.5.

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	E
DATOS	1	0	0	1	0	0	1	1	1	1	1

Tabla 4-6_Caracter “o”.

Para este caso se envió el carácter “O” en el equivalente ASCII (0100 1111). El diagrama de bloques mostrado en la figura 4.33, muestra la inicialización y configuración de una pantalla LCD.

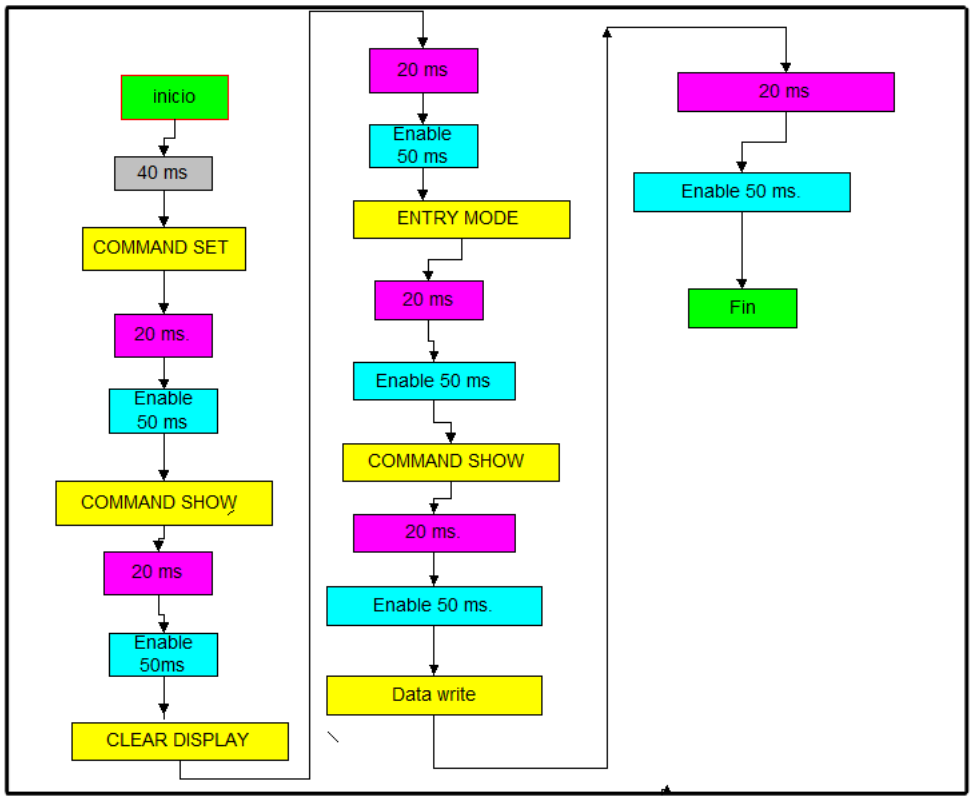


Figura 4-32_Diagrama de flujo de escritura de caracteres en LCD.

Para comenzar a comprender el código VHDL, encargado de "programar" el dispositivo FPGA que enviara los mensajes a un módulo LCD, es necesario mencionar que generalmente se usa un método para escribir dicho código VHDL, llamado máquina de estados finitos.

A continuación, se muestran los conceptos básicos de una máquina de estado finitos, para así poder entender la estructura del código VHDL generado a través de este método.

4.242 MAQUINA DE ESTADOS FINITOS

Una máquina de estados finitos (FSM) es un modelo computacional y lógico que representa un sistema formado por un número limitado de estados, cambiando de uno a otro (transiciones) en respuesta a entradas o eventos externos. Se

utilizan para diseñar lógica de control, interfaces y automatización, en especial en electrónica digital y programación. La figura 4-33 muestra una representación de una máquina de estado del tipo Moore, formada por una lógica de salida, donde el estado de la salida depende solo del estado actual **S**, un registro de estado (FF-D), controlado por la señal de reloj y una lógica del próximo estado, donde el estado futuro **S'** dependerá del estado de la entrada (E) y el estado presente **S**. Notar que K el número de bits y el número de estados posibles esta dado por 2 elevado a K.

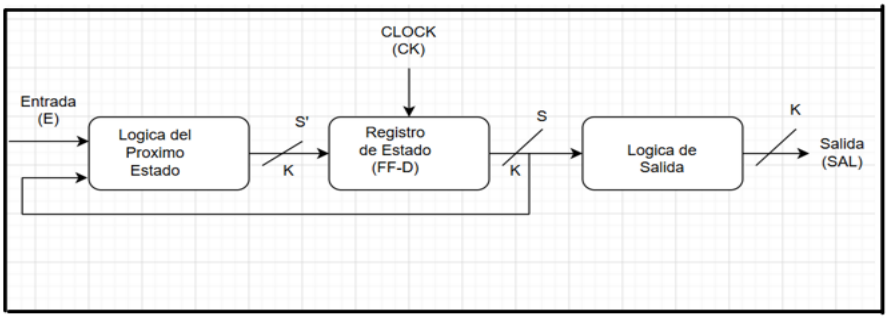


Figura 4-33_Representacion de una Máquina de Estado.

Para entender el concepto anterior y aplicarlo en forma práctica, se utiliza un diagrama de estado, encargado de mostrar todos los estados posibles dentro de un circuito, así como la forma y condiciones para que ocurran las transiciones entre los diferentes estados. El número de estados posibles estará limitado por la cantidad de bits (K) de los estados presente y futuro, tal como lo representa la figura 4-34.

Para comprender como funciona el diagrama de estados, la figura 4-34 muestra un ejemplo práctico con un semáforo de colores; rojo, amarillo y verde. El primer estado encenderá solamente el color rojo cuyo valor es “00”, en el segundo estado encenderá el rojo y amarillo con valor “01”, en el tercer estado encenderá el verde con valor “10” y en el cuarto estado verde amarillo con valor “11”. Notar que el valor de K=2 lo que entrega cuatro estados posibles. La lógica propuesta muestra la entrada E y dependiendo de su valor, es decir si se comienza en el

estado R (“00”) será el estado presente y el estado futuro será RA (“01”) y para exista un cambio de estado de R a RA, la entrada debe valer 1 lógico (E=1), caso contrario permanecerá en R. Una vez cambiado a RA, será el estado presente y el estado futuro será V (“10”). Para cambiar a V la entrada debe cambiar a 0 lógico (E=0), caso contrario permanecerá en RA. Posteriormente una vez situado en V (“10”), será el nuevo estado presente y el nuevo estado futuro será VA (“11”). En este punto para una nueva transición la entrada E, debe cambiar a 1 lógico (E=1), para finalmente ahora el estado presente sea VA y el futuro R (“00”). Para terminar, si E=0, la máquina de estado tendrá como estado presente R (“00”). Para una mayor comprensión de lo anterior se observa la tabla 4-7, con la transición de estados correspondiente.

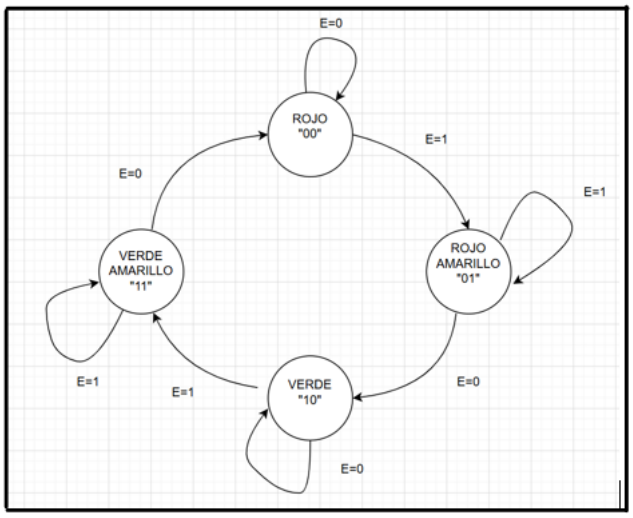


Figura 4-34_Diagrama de estado, con una lógica de semáforo 4 estados.

S	E	S'
R	1	RA
RA	0	V
V	1	VA
VA	0	RA

Tabla 4-7_Tabla de verdad con la transición de estados.

Existe un caso particular de diagrama de estado tal como lo muestra la figura 4-35, en el cual se desprende que cada transición no depende de ninguna entrada (E) y solo estarán sujetas estas transiciones de estado a los cambios de reloj, tal como se verá en el código VHDL de escritura del LCD.

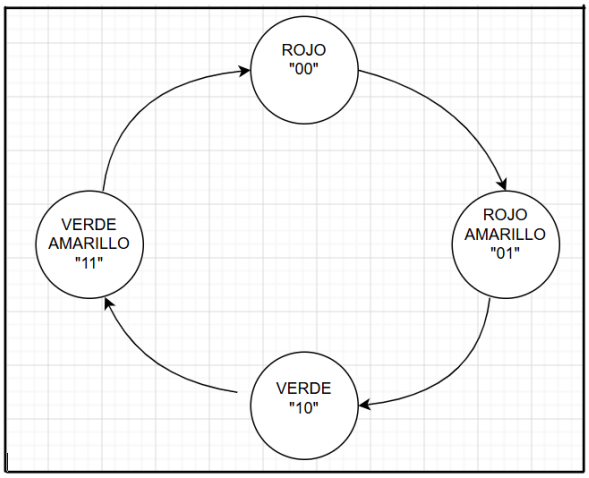


Figura 4-35_Maquina de estado, con cambios de estados dependiente del Clock.

Con los conceptos básicos vistos anteriormente sobre máquina de estados, se retoma la práctica de escritura de un módulo LCD, para aplicarlos a la siguiente secuencia de estados. Una consideración es que la práctica consiste en enviar un mensaje al módulo LCD, sin intervención externa, por lo tanto, las transiciones

de estados las realiza el Clock interno. Otro dato no menor es que para cada comando o dato (instrucción) a enviar, se realiza con dos estados, uno habilitar el módulo con el flaco de subida y el otro para detectar con el flanco de bajada si el dato es comando o dato.

La figura 4-36 muestra la secuencia de estados correspondiente al envió de comandos y caracteres de la práctica del módulo del LCD. Las 3 primeras instrucciones corresponden a la inicialización y puesta en marcha del módulo. El primer envío es la instrucción set, formado por los estados E0 y E1, en el cual se carga el dato 38 HEX, es decir interfaz de 8 bits, carácter de 5x7 pixeles y dos líneas.

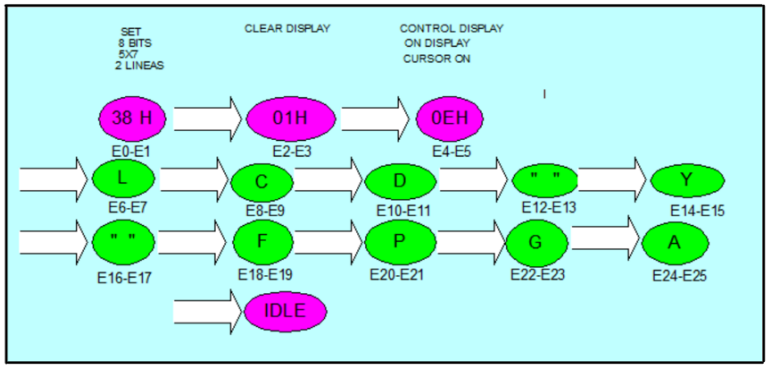


Figura 4-36_Secuencia de estados utilizada para el envío de caracteres al LCD.

La segunda instrucción envía el comando clear, 01 en Hexadecimal y está formado por los estados E2 y E3. La tercera instrucción por enviar es un último comando llamado control Display, formado por los estados E4 y E5, Display on, cursor ON, sin parpadeo, valor 0E en Hexadecimal. Hasta este momento de la secuencia el terminal RS tiene aplicado un 0 lógico (RS=0), debido a que las instrucciones enviadas al LCD son comandos. A partir de la cuarta instrucción en adelante (verde), son caracteres en consecuencia al terminal RS se le aplicara un 1 lógico (RS=1). Como cada caracter es una instrucción, para cada una de

ellas necesario dos estados, es decir para enviar el caracter “L” lo realizaran los estados E6 y E7. El procedimiento se repite hasta el estado E25, para terminar con estado IDLE.

4.243 CODIGO VHDL LCD 2 X 16

En la siguiente imagen, Figura 4-38 se aprecia el código correspondiente al envío de caracteres al módulo LCD, utilizando máquinas de estados. Las primeras tres líneas corresponden a la librerías y paquetes requeridos. Desde la línea 5 hasta la 15 se encuentra la identidad y en ella se aprecia como única entrada al reloj (clock de la placa) y las salidas RS (comando /data), enable (E=1 habilitado E=0 deshabilitado) y los datos (8 bits tanto comandos como data). La arquitectura comienza en la 16 y en la línea 17 se definen los estados finitos (desde E0 a IDLE) de la máquina, a través de TYPE con el nombre ESTADOS (el nombre es optativo). En la línea 21 se declaran dos señales, EDO_PRE Y EDO_FUT (estado presente y estado futuro) y ambas pertenecientes a ESTADOS (E0…E25, IDLE) y se indica que comienzan en E0 (estado cero). En la línea 24 comienza el primer proceso o proceso 1 llamado process(EDO_PRE). Cabe mencionar que este código tiene 2 procesos el primero para hacer transiciones de estados y el segundo para las actualizaciones de los estados. Para comenzar el process(EDO_PRE), comienza con **case** en la línea 27, configurando el LCD con 8 bits, con matriz de 5x7 y dos líneas, enviando el 38 Hexadecimal, con E=1 para habilitar el LCD y RS=0 para indicar que es un comando.

En la línea 33 el estado presente es ahora E1 y en la línea 36 Enable pasa a cero, (para deshabilitar el LCD) manteniendo RS=0 para indicar con el flaco de bajada al módulo que se está enviando un comando y no un carácter (dato). En la línea 41 (estado presente E2) se repite el proceso enviando el comando (RS=0)

Clear Display el valor 0E Hexadecimal habilitando el LCD con E=1. En la línea 47 el estado presente es E3, se deshabilita el módulo con E=0 y con cuyo flanco de bajada se indica que el dato 0E Hexadecimal es el comando de limpieza del LCD. El proceso se repite hasta el último estado IDLE donde se detiene. Notar que desde la línea 66 comienza el envío de caracteres (datos propiamente tal) y el único cambio será que RS=1.

Por otra parte, en la línea 193 del código comienzo el segundo proceso llamado process(clock) y es encargado de hacer las actualizaciones de estados cada 40 milisegundos (puede ser 40 o 50 ms.). Dentro del proceso 2 se creó una variable (en VHDL las variables son locales dentro del proceso) llamada pulso, encargada de contar 2 millones de pulsos de reloj, para cambiar de estado y volver a pulso=0 una vez cumplida la condición y caso contrario se incrementa pulso en una unidad.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY LCD_FPGA IS
6  PORT(
7       CK:IN STD_LOGIC;           --CLOCK REQUERIDO PARA GENRAR UNA BASE DE TIEMPO
8       RS: OUT STD_LOGIC;         --SELECCION DE ENVIO DE COMANDOS O DATOS AL LCD
9       E: OUT STD_LOGIC;          --HABILITACION PARA EL ENVIO DE DATOS (ENABLE)
10      DATOS:  OUT STD_LOGIC_VECTOR(7 DOWNTO 0) --COMANDOS Y DATOS.
11
12
13 );
14
15   END ENTITY;
16    ARCHITECTURE CONTROL OF LCD_FPGA IS
17    TYPE ESTADOS IS (E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,
18                     E12,E13,E14,E15,E16,E17,E18,E19,E20,E21,
19                     E22,E23,E24,E25,IDLE); --TODOS LOS ESTADOS INVOLUCCRADOS EN E
20                                           --DE COMANDOS Y DATOS AL LCD.
21    SIGNAL EDO_PRE,EDO_FUT:ESTADOS:=E0;    --DOS SEÑALES PARA TRABAJAR CON LOS ES
22                                           --ESTADOS PRESENTE Y FUTURO
23    BEGIN
24    PROCESO1: PROCESS(EDO_PRE)
25    BEGIN
26
27    CASE EDO_PRE IS
28   --CONFIGURACION DEL LCD
```

```
29
30     WHEN E0=>
31     DATOS<="00111000";--COMANDO SET, 2 LINEAS, 5X7, 8 BITS DE DATOS
32     E<='1';           --HABILITADOS ENVIO DE DATO
33     RS<='0';           --EL DATO A ENVIAR ES UN COMANDO
34     EDO_FUT<=E1;      --ESTADO SIGUENTE
35
36
37     WHEN E1=>
38     E<='0';           --DESHABILITADO EL ENVIO DE DATOS
39     RS<='0';           --EL DATO A ENVIAR ES UN COMANDO
40     EDO_FUT<=E2;      --ESTADO SIGUENTE
41     --LIMPIAMOS EL LCD
42     WHEN E2=>
43     DATOS<="00000001";--COMANDO CLEAR DISPLAY
44     E<='1';           --HABILITADOS ENVIO DE DATO
45     RS<='0';           --EL DATO A ENVIAR ES UN COMANDO
46     EDO_FUT<=E3;      --ESTADO SIGUENTE
47
48     WHEN E3=>
49     E<='0';           --DESHABILITADO ENVIO DE DATO
50     RS<='0';           --EL DATO A ENVIAR ES UN COMANDO
51     EDO_FUT<=E4;      -- ESTADO SIGUENTE
52
53     --COMANDO SHOW, ACTIVAR DISPLAY
54     WHEN E4=>
55     DATOS<="00001110";--ENCIENDE DISPLAY, CURSOR ON, SIN PARPADEO
56     E<='1';           --HABILITADOS ENVIO DE DATO
57     RS<='0';           --EL DATO A ENVIAR ES UN COMANDO
```

```
58
59      WHEN E5=>
60      E<='0';             --DESHABILITADO ENVIO DE DATO
61      RS<='0';             --EL DATO A ENVIAR ES UN COMANDO
62      EDO_FUT<=E6;        -- ESTADO SIGUENTE
63
64      --CARACTER L
65
66      WHEN E6=>
67      DATOS<="01001100";--CARACTER L
68      E<='1';             --HABILITADOS ENVIO DE DATO
69      RS<='1';             --EL DATO A ENVIAR ES UN CARACTER
70      EDO_FUT<=E7;        --ESTADO SIGUENTE
71
72      WHEN E7=>
73      E<='0';             --DESHABILITADO ENVIO DE DATO
74      RS<='1';             --EL DATO A ENVIAR ES UN COMANDO
75      EDO_FUT<=E8;        --ESTADO SIGUENTE
76
77     --CARACTER C
78
79      WHEN E8=>
80      DATOS<="01000011";--CARACTER C
81      E<='1';             --HABILITADOS ENVIO DE DATO
82      RS<='1';             --EL DATO A ENVIAR ES UN CARACTER
83      EDO_FUT<=E9;        --ESTADO SIGUENTE
84
85      WHEN E9=>
86      E<='0';             --DESHABILITADO ENVIO DE DATO

87      RS<='1';             --EL DATO A ENVIAR ES UN COMANDO
88      EDO_FUT<=E10;
89
90     --CARACTER D
91      WHEN E10=>
92      DATOS<="01000100";--CARACTER D
93      E<='1';             --HABILITADOS ENVIO DE DATO
94      RS<='1';             --EL DATO A ENVIAR ES UN CARACTER
95      EDO_FUT<=E11;        --ESTADO SIGUENTE
96
97      WHEN E11=>
98      E<='0';             --DESHABILITADO ENVIO DE DATO
99      RS<='1';             --EL DATO A ENVIAR ES UN COMANDO
100     EDO_FUT<=E12;
101     --CARACTER " " ESPACIO
102
103     WHEN E12=>
104     DATOS<="00100000";--CARACTER VACIO
105     E<='1';             --HABILITADOS ENVIO DE DATO
106     RS<='1';             --EL DATO A ENVIAR ES UN CARACTER
107     EDO_FUT<=E13;        --ESTADO SIGUENTE
108
109     WHEN E13=>
110     E<='0';             --DESHABILITADO ENVIO DE DATO
111     RS<='1';             --EL DATO A ENVIAR ES UN COMANDO
112     EDO_FUT<=E14;
113     --CARACTER Y
114     WHEN E14=>
115     DATOS<="01011001";--CARACTER Y
```

```
116     E<='1';            --HABILITADOS ENVIO DE DATO
117     RS<='1';            --EL DATO A ENVIAR ES UN CARACTER
118     EDO_FUT<=E15;       --ESTADO SIGUENTE
119
120     WHEN E15=>
121     E<='0';            --DESHABILITADO ENVIO DE DATO
122     RS<='1';            --EL DATO A ENVIAR ES UN COMANDO
123     EDO_FUT<=E16;
124     --CARACTER " " ESPACIO
125
126     WHEN E16=>
127     DATOS<="00100000";--CARACTER ESPACIO
128     E<='1';            --HABILITADOS ENVIO DE DATO
129     RS<='1';            --EL DATO A ENVIAR ES UN CARACTER
130     EDO_FUT<=E17;       --ESTADO SIGUENTE
131
132     WHEN E17=>
133     E<='0';            --DESHABILITADO ENVIO DE DATO
134     RS<='1';            --EL DATO A ENVIAR ES UN COMANDO
135     EDO_FUT<=E18;
136
137     --CARACTER F
138
139     WHEN E18=>
140     DATOS<="01000110";--CARACTER F
141     E<='1';            --HABILITADOS ENVIO DE DATOS
142     RS<='1';            --EL DATO A ENVIAR ES UN CARACTER
143     EDO_FUT<=E19;      --ESTADO SIGUENTE
144
```

```
145     WHEN E19=>
146     E<='0';            --DESHABILITADO ENVIO DE DATO
147     RS<='1';            --EL DATO A ENVIAR ES UN COMANDO
148     EDO_FUT<=E20;
149
150     --CARACTER P
151
152     WHEN E20=>
153     DATOS<="01010000";--CARACTER P
154     E<='1';            --HABILITADOS ENVIO DE DATOS
155     RS<='1';            --EL DATO A ENVIAR ES UN CARACTER
156     EDO_FUT<=E21;      --ESTADO SIGUENTE
157
158     WHEN E21=>
159     E<='0';            --DESHABILITADO ENVIO DE DATO
160     RS<='1';            --EL DATO A ENVIAR ES UN COMANDO
161     EDO_FUT<=E22;
162
163     --CARACTER G
164
165     WHEN E22=>
166     DATOS<="01000111";--CARACTER G
167     E<='1';            --HABILITADOS ENVIO DE DATOS
168     RS<='1';            --EL DATO A ENVIAR ES UN CARACTER
169     EDO_FUT<=E23;      --ESTADO SIGUENTE
170
171     WHEN E23=>
172     E<='0';            --DESHABILITADO ENVIO DE DATO
173     RS<='1';            --EL DATO A ENVIAR ES UN COMANDO
```

```
174     EDO_FUT<=E24;
175
176     --CARACTER A
177     WHEN E24=>
178     DATOS<="01000001";--CARACTER A
179     E<='1';            --HABILITADOS ENVIO DE DATOS
180     RS<='1';            --EL DATO A ENVIAR ES UN CARACTER
181     EDO_FUT<=E25;      --ESTADO SIGUENTE
182
183     WHEN E25=>
184     E<='0';            --DESHABILITADO ENVIO DE DATO
185     RS<='1';            --EL DATO A ENVIAR ES UN CARACTER
186     EDO_FUT<=IDLE;
187   --ESTADO INACTIVO
188     WHEN IDLE=>
189     EDO_FUT<=IDLE;     --AQUI SE DETIENE
190     when others=>
191     END CASE;
192     END PROCESS PROCESO1;
193     PROCESO2:PROCESS(CK)
194     VARIABLE PULSOS: INTEGER RANGE 0 TO 50000000;--VARIABLE ES LOCAL Y VAN :=
195     BEGIN
196     IF RISING_EDGE(CK) THEN
197     IF (PULSOS=2000000) THEN
198     PULSOS:=0;
199     EDO_PRE<=EDO_FUT;
200     ELSE
201     PULSOS:=PULSOS+1;
202     END IF;

203     END IF;
204     END PROCESS PROCESO2;
205     END ARCHITECTURE;
```

Figura 4-37_Código VHDL LCD 2x16

En la Figura 4-38 se visualiza el diagrama de conexionado de la tarjeta MAX-10 o más bien dicho de los pines del dispositivo FPGA, con el módulo LCD. En el apéndice se entrega mayor documentación, como diagramas de tiempo y códigos binarios de los caracteres. La presente practica deja abierta la posibilidad de implementar otros ensayos futuros con entradas o pulsos externos aplicando el mismo concepto de máquinas de estados.

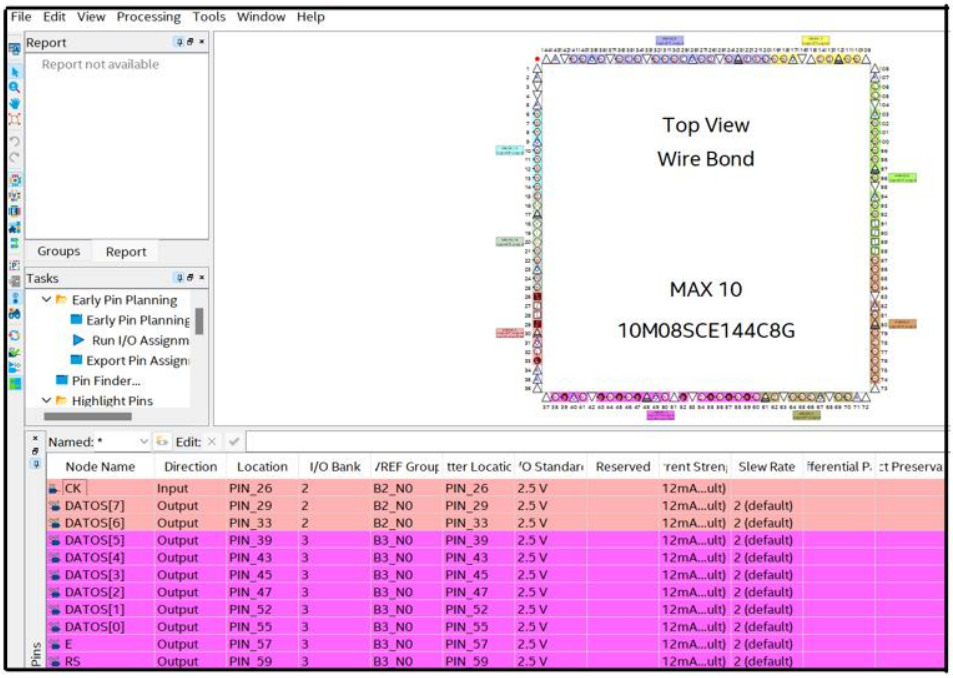


Figura 4-38_Diagramas de pines usados para conectar LCD.

4.3 UNION DE PROYECTOS MEDIANTE BLOQUES

Hasta el momento se han expuesto algunas experiencias prácticas de circuitos combinacionales y secuenciales, pero todos en forma individual, por ejemplo un decodificador BCD a siete segmentos, multiplexor de cuatro a una línea, contador BCD entre otros, sin embargo en esta sección se revisaran algunos proyectos de mayor envergadura en base a los proyectos ensayados anteriormente pero esta vez serán tratados como archivos para poder convertirlos en bloques y posteriormente unirlos entre ellos en una malla desplegada en pantalla. La presente sección tiene como propósito trabajar en el modo arquitecto, como lo denominan algunos autores. El procedimiento para generar los bloques esta descrito en el capítulo 3, en el entorno de desarrollo de Quartus II.

4.31 MODULADOR PWM CONTROLADO POR UN CONTADOR UP DOWN

Anteriormente se revisó una experiencia práctica consistente en un modulador PWM, controlado por un Dip-switch de 8 pines, entregando 255 diferentes valores de ciclos de trabajo de una onda cuadrada con periodo de 4 milisegundos. En esta oportunidad revisará el mismo modulador PWM, pero el control es realizado por dos pulsadores externos normalmente abierto, uno para aumentar el ancho del pulso y el otro para reducir el ancho del pulso. La figura 4.40 muestra el principio básico de la experiencia propuesta.

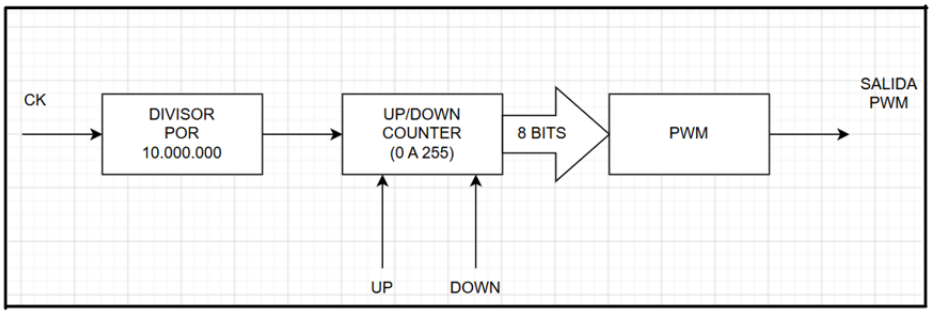


Figura 4-39_Esquema de modulador PWM, contador Up/Down.

Con el esquema anterior es el punto de partida para el diseño planteado, en el cual todas las partes que lo conforman son archivos de proyectos anteriormente vistos. El primer paso consiste en crear un nuevo proyecto, tal como en las experiencias anteriores, siendo hasta ahora un proyecto en blanco. El siguiente paso es agregar archivos de otros proyectos y se agregan tal como lo indica la figura 4-41 y 4-42 en el orden indicado.

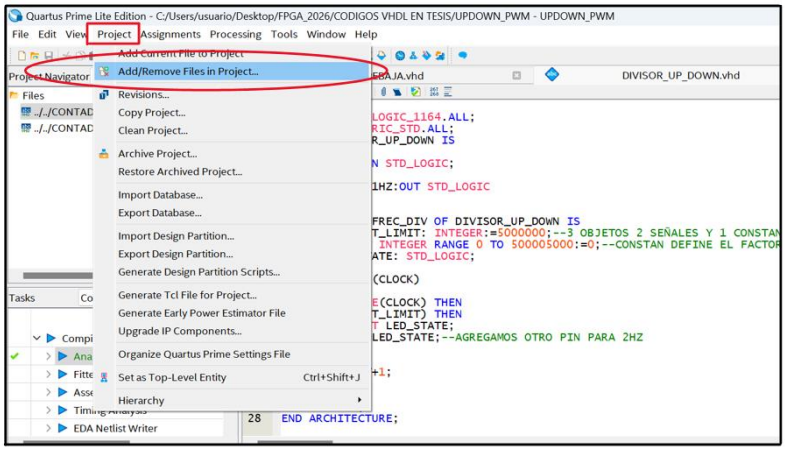


Figura 4-40_La forma de agregar archivos aun proyecto existente.

Es muy importante notar que a medida que se agregan archivos al proyecto, estos comienzan a aparecer una forma inmediata, tal como lo indica la figura 4-42.

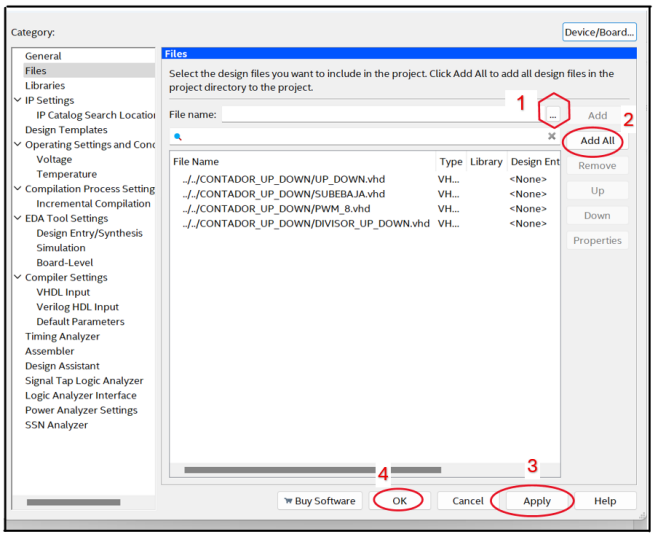


Figura 4-41_Pasos necesarios para agregar archivos a un nuevo proyecto.

En la figura 4-41 se puede apreciar el orden (1 a 4), primero para ir buscar a las diferentes carpetas los archivos requeridos (1). Una vez desplegados los archivos en el listado, se procede a agregarlos (2), para finalmente aplicar y aceptar (3 y 4). Hasta este punto el proyecto tiene los 3 necesarios, es decir, el divisor, el contador binario Up/Down y el modulador PWM.

Es muy importante en este punto, revisar los archivos y sintetizarlos (compilarlos) y para ello, se selecciona cada archivo uno por uno y con botón derecho de coloca cada archivo en alta prioridad (Set as top level-Entity), condición fundamental antes de analizar y sintetizar.

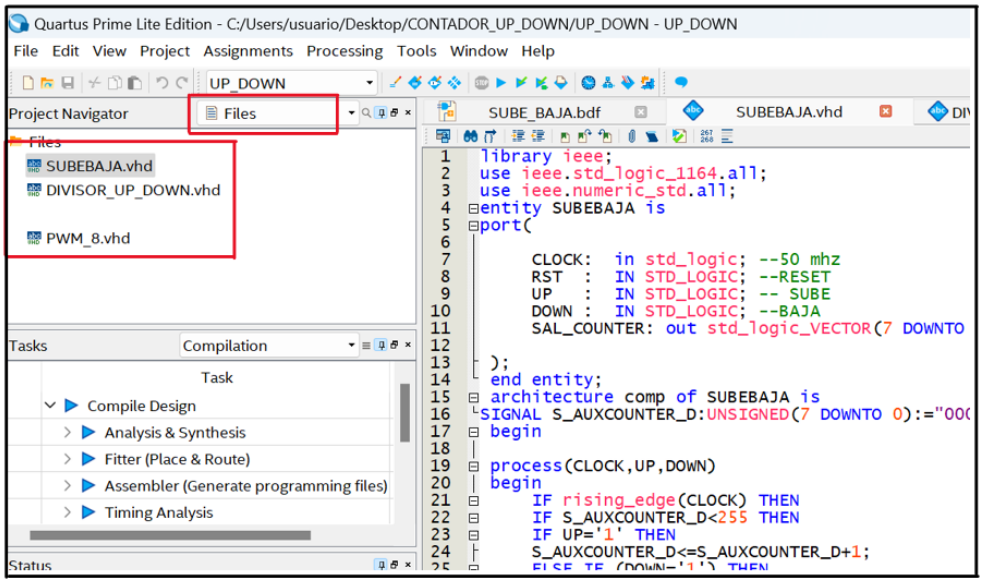


Figura 4-42_Visualizacion de los nuevos archivos en la ventana de navegación.

Una vez revisado cada archivo, se procede a generar la grilla para construir el diagrama de bloques deseado, presionando File, New, **Block/Diagram/Shquematic** file, desplegándose una malla como la indicada en la figura 4-44. Notar que esta malla es un archivo con extensión .dbf.

Posteriormente y tal como se indica detalladamente en el capítulo 3 se procede a general el símbolo por cada archivo, arrastrando cada uno hacia la grilla para comenzar la interconexión de bloques y la creación de pines de cada uno de ellos. Terminada la interconexión de cada bloque, se procede a analizar y sintetizar el bloque (archivo .dbf). Todos los bloques que requieren para su funcionamiento una señal de Clock de 50 Mhz obtenida de la placa MAX-10, se deben interconectar, de manera que el proyecto total queda con una sola entrada de Clock. En el diagrama de bloques podemos observar la interconexión de estos. La señal de reloj es aplicada tanto al divisor de frecuencia como al modulador PWM, para así entregar una base de tiempo de 5 Hz al contador Up/Down y otra de 50 Mhz al PWM.

Los pasos posteriores son de la misma forma que las experiencias anteriores, es decir hay que definir los pines a utilizar (pin planer), así como la compilación del proyecto completo.

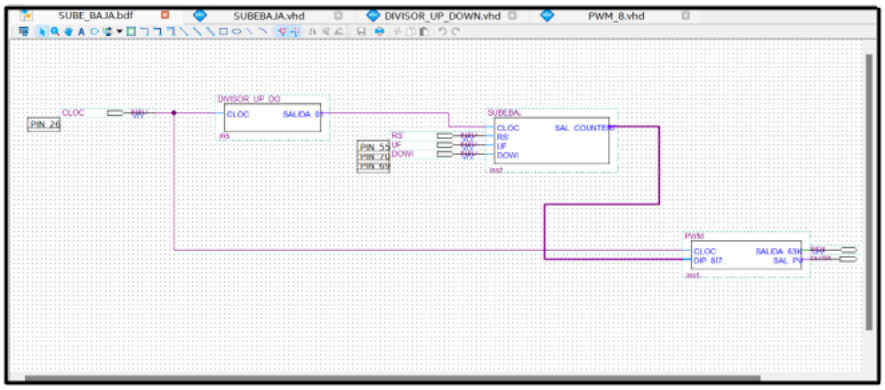


Figura 4-43_Creacion de una nueva malla de trabajo para el diagrama de bloques

4.32 CONTADOR DE 0 A 9999, CADA UN SEGUNDO

La presente y ultima practica a revisar, consiste en un contador de eventos de 0 a 9999, con intervalos de 1 segundo. Una vez alcanzado el valor máximo, la cuenta regresa a cero. El hardware propuesto se aprecia en la figura 4-44. La implementación práctica se realizó con la placa MAX-10, que posee entre otros recursos de hardware, cuatro Display de 7 segmentos, conectados en paralelo y cuya alimentación de Vcc de cada Display se encuentra controlada por un transistor PNP. La iluminación de cada uno de ellos se realiza en forma dinámica a 1.000 Hz, a través del multiplexor de 1 a 4 líneas tal como el mostrado en la figura 4-44. La selección de cada Diplay se realiza gracias a un contador de dos bits (4 valores posibles), llamado CONTADOR MUX, que en forma sincrónica aplica un valor de selección a dicho multiplexor y a un demultiplexor que selecciona el valor de los contadores de las unidades, decenas, centenas y unidad de mil, a convertir a código de 7 segmentos. Cuando la cuenta de contador mux es igual a “00”, el demultiplexor selecciona los datos entregados por el contador de las unidades y simultáneamente energiza el Display correspondiente a las unidades, para mostrar en ese instante el valor en el Display de 7segmentos correspondiente a las unidades. A medida que aumenta la cuenta del contador mux, selecciona los diferentes registros de cada contador y energiza el display correspondiente al digito seleccionado.

La base de tiempo es extraída del Clock de 50 Mhz incorporado en la placa MAX-10, para sacar dos frecuencias, una para aplicar a los contadores (1 Hz) y la otra para aplicar al contador MUX (1000 Hz) y realizar el encendido dinámico de los 4 Display. El multiplexor de 1 a 4 líneas selecciona un 0 lógico para ser aplicado al Display deseado, a través de un transistor pnp similar al mostrado en la figura 4-44.

Una vez definido el concepto del hardware a implementar, comienza la creación de un proyecto en Quartus II, para este caso llamado CONTADOR_999, para poder agregar los archivos necesarios de otros proyectos, incluida la malla (BLOQUE_COUNT.bdf) tal como lo muestra la figura 4-46. Para más detalle, revisar el procedimiento anterior y el capítulo 3 en la creación de bloques. En el apéndice se encuentran todos los archivos involucrados en el proyecto.

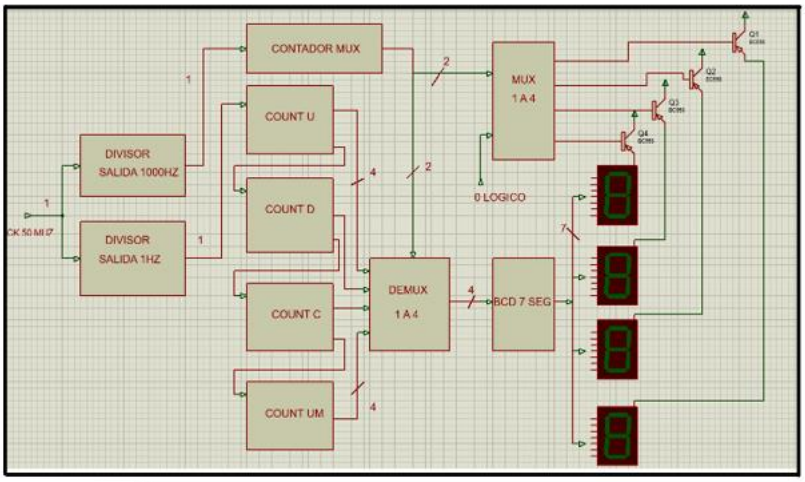


Figura 4-44_Contador de 0 a 9999 a implementar con diagramas de bloques.

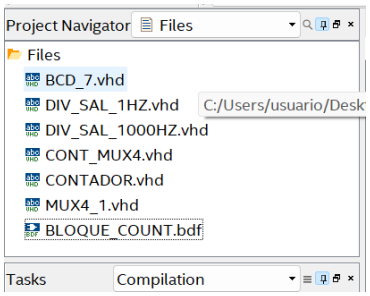


Figura 4-45_Archivos recopilados de otros proyectos existentes.

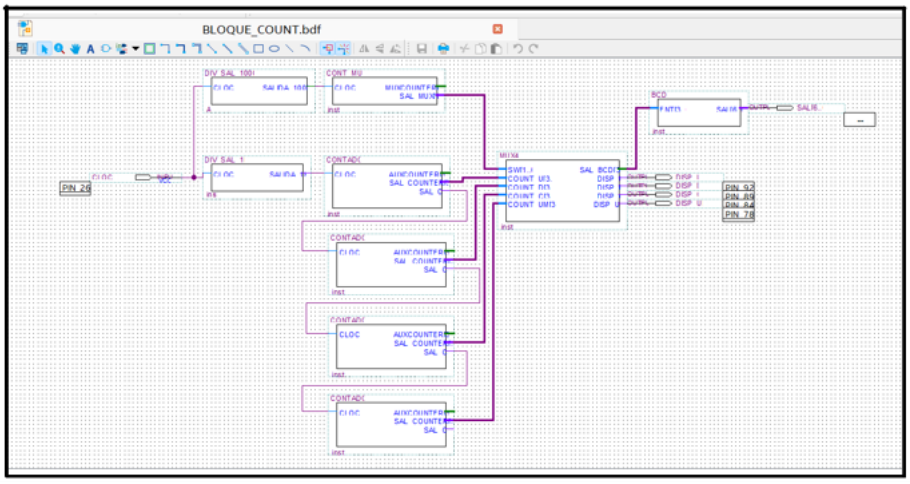


Figura 4-46_Diagrama de bloques del contador de 0 a 9999.

CONCLUSION

El trabajar en lenguaje VHDL entrega una gran portabilidad del código generado, es decir el código se puede cargar en diferentes dispositivos FPGA, con dos condiciones. La primera y la más importante, es que los elementos lógicos utilizados en el proyecto no deben exceder la cantidad de elementos lógicos que posee el dispositivo FPGA especificado por el fabricante. La segunda condición es revisar y modificar la distribución de pines I/O y Clock y configurarlo en la opción **pin planer** en QUARTUS II.

En los ensayos quedo demostrado, que no existen tiempos de latencia en los procesos, así como también la cantidad de proyectos cargados en el dispositivo no incide en agregar tiempos adicionales de posibles retardos.

Los conocimientos aquí adquiridos, sumado a la portabilidad y la no existencia de latencia, son la base para escalar a proyectos de mayor envergadura, tales como realizar proyectos con operaciones matemáticas más complejas, así como también generar códigos VHDL utilizando Matlab. La interacción de Matlab y Quartus II deja la puerta abierta un abanico de posibilidades, en el desarrollo de futuros proyectos tanto en el área del control de procesos y áreas afines con dispositivos FPGA.

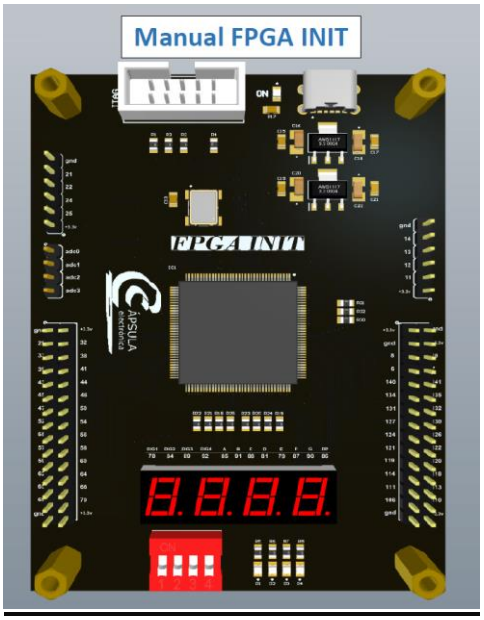
BIBLIOGRAFIA

Maxinez David G (2013). Programación de sistemas digitales con VHDL.

Pong P. CHU (2006). RTL Hardware Desing Using VHDL

APENDICE A

ESPECIFICACIONES TECNICAS DE LA PLACA MAX-10



Características de la FPGA

La placa **Init** está disponible en dos versiones:

- MAX 10 10M08SCE144C8G
- MAX 10 10M25SAE144I7G

Especificaciones técnicas de cada modelo:

10M08:

Elementos lógicos: 8,064

Memoria en bits: 387,072

Memoria flash en bits: 262,144

10M25:

Elementos lógicos: 24,960

Memoria en bits: 691,200

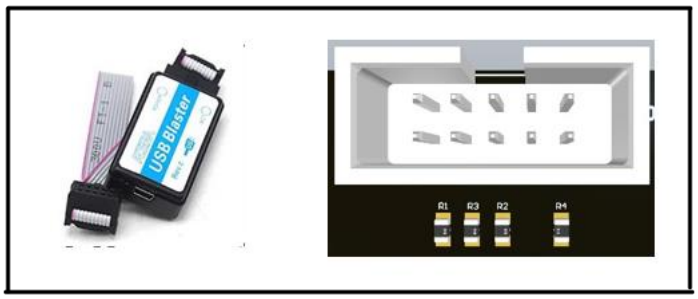
Memoria flash en bits: 6,291,456

▪ Programación mediante JTAG

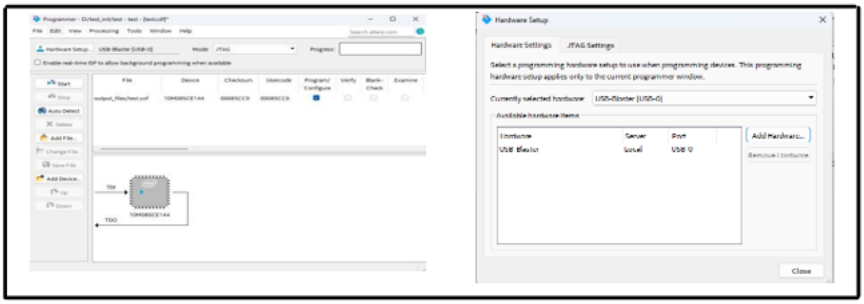
La familia MAX 10 permite la programación mediante dos tipos de archivos:

- POF (Programmer Object File): utilizado para almacenar configuraciones en la memoria flash interna.
- SOF (SRAM Object File): empleado para configuraciones temporales en la RAM del FPGA.

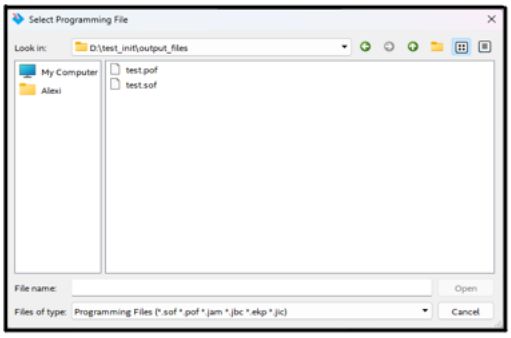
Es fundamental conectar el **USB Blaster** a la interfaz JTAG de la placa **Init** para realizar la programación correctamente.



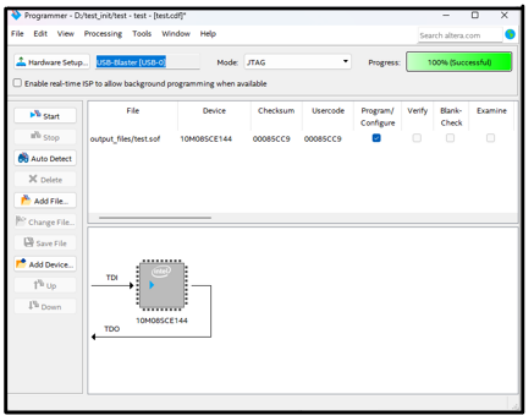
Para que el programa **Quartus** detecte el **USB Blaster**, debe hacer clic en **Hardware Setup** y seleccionar la opción correspondiente a su programador.



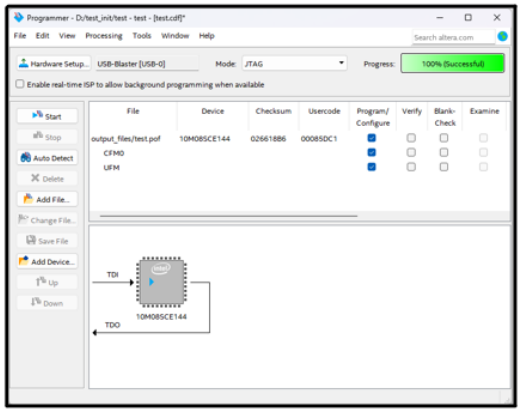
En la opción **Add File**, puede agregar el archivo correspondiente al modo de programación deseado seleccionando el archivo de configuración en formato. .sof o .pof, según sea necesario.



La programación con archivos. sof es temporal. Para cargar este tipo de archivo, debe seleccionar la opción **Program/Configure** en el software.

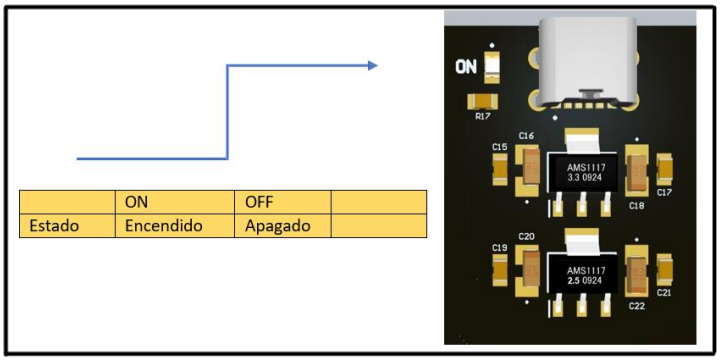


La programación con archivos. p o f es permanente. Para cargar este tipo de archivo, debe seleccionar la opción **Program/Configure** en el software.



Estado de la FPGA

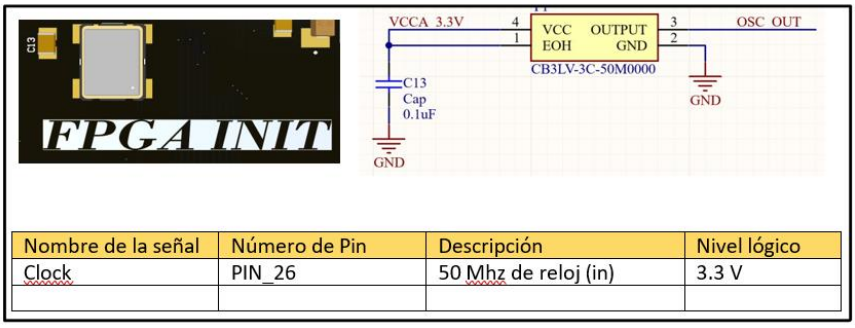
La tarjeta de desarrollo **Init** cuenta con un LED de estado que indica el estado de energía de la placa. Para energizarla, conecte un cable USB tipo C a una fuente de 5 V. Si el LED está encendido, indica que la placa está en estado **'on'**; si el LED está apagado, la placa está en estado **'off'**.



	ON	OFF	
Estado	Encendido	Apagado	

Circuito de reloj

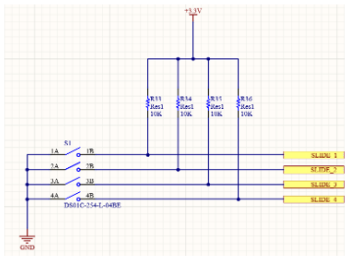
El generador de señal externo se utiliza para distribuir señales de reloj con baja fluctuación. La señal de reloj, de 50 MHz, conectada a la FPGA, se utiliza como fuente de reloj para la lógica del usuario.



Nombre de la señal	Número de Pin	Descripción	Nivel lógico
Clock	PIN_26	50 Mhz de reloj (in)	3.3 V

Slide switch

Hay cuatro interruptores deslizantes conectados a FPGA en la placa. Su conexión es de tipo pull



0 lógico

1 lógico

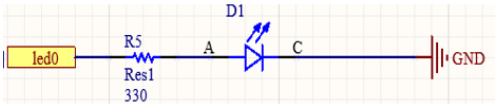


Nombre de la señal	Número de Pin	Descripción	Nivel lógico
SW1	PIN_74	Slide switch (in)	3.3 V
SW2	PIN_75	Slide switch (in)	3.3 V
SW3	PIN_76	Slide switch (in)	3.3 V
SW4	PIN_77	Slide switch (in)	3.3 V

Led’s

La placa cuenta con cuatro LEDs controlables conectados a la FPGA. Cada LED se activa de manera independiente a través de un pin en el MAX10, el cual está asociado a una lógica específica. Un nivel alto enciende el LED, mientras que un nivel bajo lo apaga.

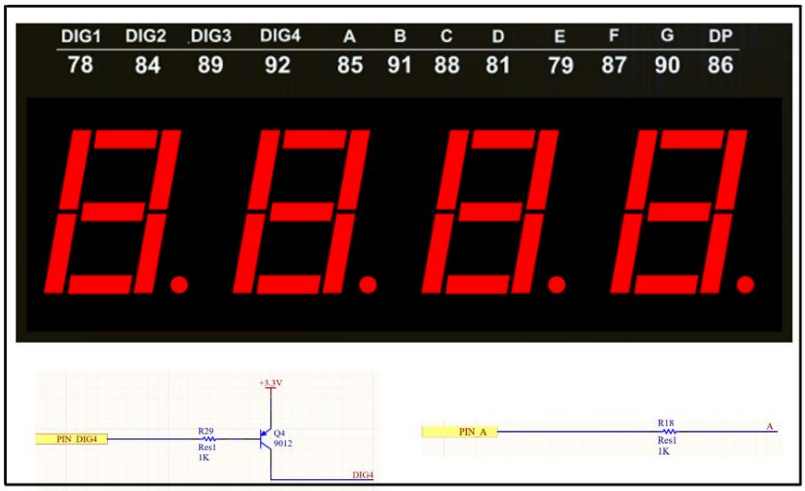




Nombre de la señal	Número de Pin	Descripción	Nivel lógico
D1	PIN_93	LED (out)	3.3 V
D2	PIN_96	LED (out)	3.3 V
D3	PIN_98	LED (out)	3.3 V
D4	PIN_99	LED (out)	3.3 V

Display de 7 segmentos de 4 canales

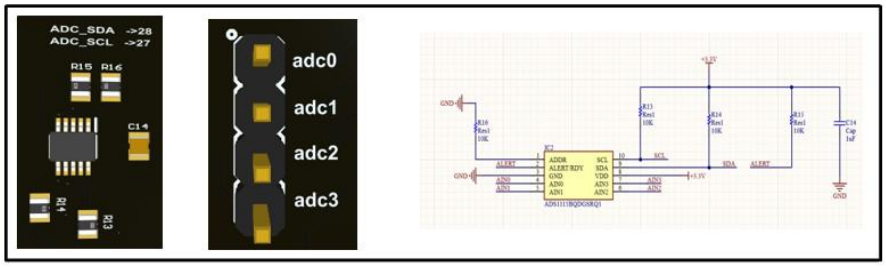
La FPGA **Init** dispone de un display de 7 segmentos de cuatro dígitos con ánodo común. El control de los dígitos se realiza mediante el transistor 9012. Los pines de los segmentos están protegidos con resistencias y se encienden con lógica baja.



Nombre de la señal	Número de Pin	Descripción	Nivel lógico
DIG1	PIN_78	Display (out)	3.3 V
DIG2	PIN_84	Display (outt)	3.3 V
DIG3	PIN_89	Display (out)	3.3 V
DIG4	PIN_92	Display (out)	3.3 V
A	PIN_85	Display (out)	3.3 V
B	PIN_91	Display (out)	3.3 V
C	PIN_88	Display (out)	3.3 V
D	PIN_81	Display (out)	3.3 V
E	PIN_79	Display (out)	3.3 V
F	PIN_87	Display (out)	3.3 V
G	PIN_90	Display (out)	3.3 V
DP	PIN_88	Display (outt)	3.3 V

Convertidor analógico a digital ADS1115

La placa **Init** está equipada con el chip ADS1115, que dispone de cuatro canales para la conversión de señales analógicas a digitales con una resolución de 16 bits. Su dirección es 0x48 y se maneja mediante el protocolo I2C.

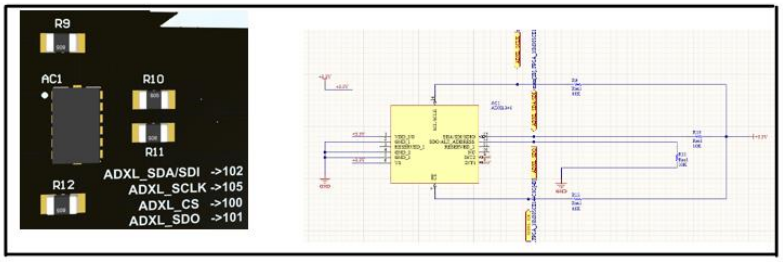


Nombre de la señal	Número de Pin	Descripción	Nivel lógico
ADC_SDA	PIN_28	ADC (inout)	3.3 V
ADC_SCL	PIN_27	ADC (inout)	3.3 V

Nombre de la señal	Número de Pin	Descripción	Nivel lógico
ADC0		ADC	0 V - 5 V
ADC1		ADC	0 V - 5 V
ADC2		ADC	0 V - 5 V
ADC3		ADC	0 V - 5 V

Acelerómetro ADXL345

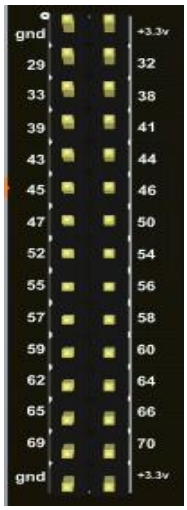
El ADXL345 es un acelerómetro digital de 3 ejes con una resolución de hasta 13 bits, y cuenta con interfaces de manejo SPI e I2C. Su dirección I2C es 0x53.

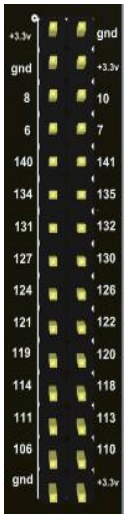


Nombre de la señal	Número de Pin	Descripción	Nivel lógico
ADXL_SDA/SDI	PIN_102	Acelerómetro (inout)	3.3 V
ADXL_SCLK	PIN_105	Acelerómetro (inout)	3.3 V
ADXL_CS	PIN_100	Acelerómetro (out)	3.3 V
ADXL_SDO	PIN_101	Acelerómetro (in)	3.3 V

PIO

La placa **Init** muestra el número de pin mediante serigrafía, y sus niveles lógicos son de 0 V y 3.3 V. Cada pin puede configurarse para funcionar tanto como entrada como salida.





APENDICE B

USB BLASTER

El USB Blaster es un hardware que funciona como un programador y depurador diseñado específicamente para configurar dispositivos de lógica programable, como **FPGAs** y **CPLDs** de la marca Altera. En términos simples es análogo al programador MPLAB-ide, utilizado para programar los microcontroladores Pic.

Funciones Principales

Programación ISP: Permite cargar el diseño de hardware (archivo en VHDL) directamente desde una computadora a la placa de desarrollo a través de un puerto USB.

Interfaz JTAG: Utiliza principalmente el protocolo JTAG para la comunicación, aunque también admite modos de programación serie pasiva y serie activa.

Depuración: Sirve para analizar señales lógicas en tiempo real y corregir errores en el funcionamiento del hardware mediante herramientas como *Signal Tap* de Intel.

Características Técnicas

- **Velocidad**: Es significativamente más rápido (hasta 6 veces) que versiones anteriores como el antiguo *Byte Blaster*.
- **Compatibilidad**: Funciona con diversas familias de chips, incluyendo **Cyclone**, **MAX**, **Stratix** y **Arria**.

- **Voltaje**: Requiere 5.0 V provenientes del puerto USB y se adapta a voltajes de entre 1.5 V y 5.0 V de la placa de destino.

Para utilizarlo, es indispensable tener instalado el software de diseño electrónico Intel Quartus Prime (anteriormente Quartus II), el cual incluye los controladores (*drivers*) necesarios para que la computadora reconozca el dispositivo, sin embargo, para que pueda operar es necesario tener descargado el certificado de los drivers del usb blaster. Para ello dejamos algunos vínculos con la revisar el procedimiento.

DRIVER USB BLASTER ALTERA + CERTIFICADO

RESOLVIENDO PROBLEMAS USB BLASTER

APENDICE C

TABLA DE CARACTERES DEL MODULO HD44780U

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	x	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		-	=	M	]	m	}			ュ	ス	ヘ	ン	ŧ	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	゜	ö	

Note: The user can specify any pattern for character-generator RAM.

HD44780U

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A02)

Lower 4 Bits / Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)															
xxxx0001	(2)															
xxxx0010	(3)															
xxxx0011	(4)															
xxxx0100	(5)															
xxxx0101	(6)															
xxxx0110	(7)															
xxxx0111	(8)															
xxxx1000	(1)															
xxxx1001	(2)															
xxxx1010	(3)															
xxxx1011	(4)															
xxxx1100	(5)															
xxxx1101	(6)															
xxxx1110	(7)															
xxxx1111	(8)															

COMANDOS DE CONFIGURACION DEL MODULO

HD44780U

Normally, instructions that perform data transfer with internal RAM are used the most. However, auto-incrementation by 1 (or auto-decrementation by 1) of internal HD44780U RAM addresses after each data write can lighten the program load of the MPU. Since the display shift instruction (Table 11) can perform concurrently with display data write, the user can minimize system development time with maximum programming efficiency.

When an instruction is being executed for internal operation, no instruction other than the busy flag/address read instruction can be executed.

Because the busy flag is set to 1 while an instruction is being executed, check it to make sure it is 0 before sending another instruction from the MPU.

Note: Be sure the HD44780U is not in the busy state (BF = 0) before sending an instruction from the MPU to the HD44780U. If an instruction is sent without checking the busy flag, the time between the first instruction and next instruction will take much longer than the instruction time itself. Refer to Table 6 for the list of each instruc-tion execution time.

Table 6 Instructions

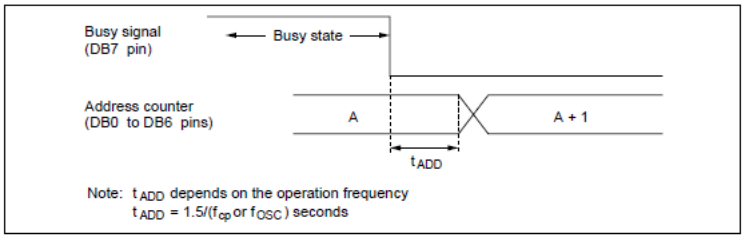
	Code											
Instruction	**RS**	**R/$\overline{W}$**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**	**Description**	**Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)**
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Table 6 Instructions (cont)

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, 37 µs × 270/250 = 40 µs

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.



Note: t_{ADD} depends on the operation frequency
$t_{ADD} = 1.5/(f_{cp}$ or $f_{OSC})$ seconds

Figure 10 Address Counter Update

DIAGRAMA DE TIEMPO DE ESCRITURA

DE COMANDOS

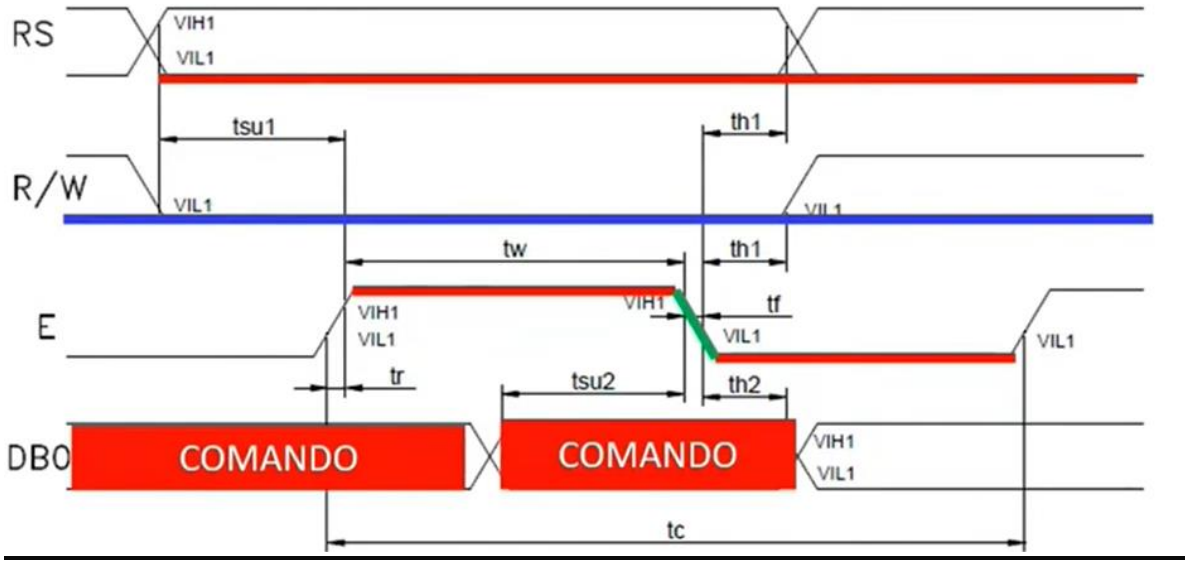
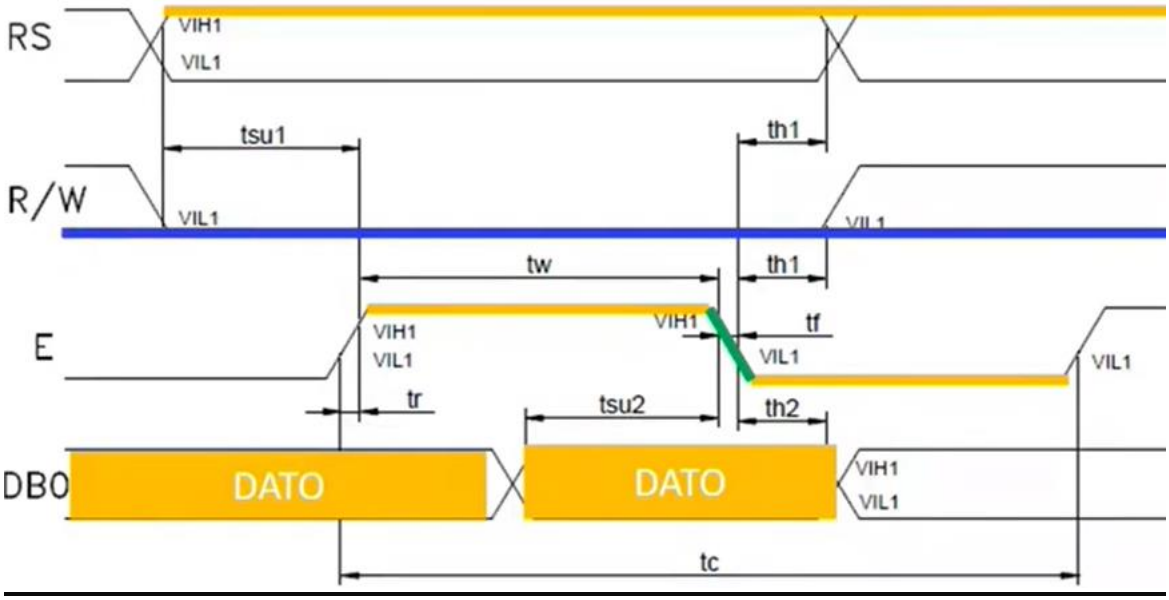


DIAGRAMA DE TIEMPO DE ESCRITURA

DE DATOS (CARACTERES)



APENDICE D

CODIGOS VHDL UTILIZADOS EN LOS PROYECTOS

REALIZADOS EN BLOQUES

DECODIFICADOR BCD A 7 SEGMENTOS

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

entity BCD_7 is

port(
     ENT: in std_logic_vector( 3 downto 0);--PINES DE LA PLACA DE DSWITCH 74,75,76,77

     SAL: out std_logic_vector( 6 downto 0)--Salida 7 seg
    -- CONT: out std_logic_vector(3 downto 0)--control display anodo comun


);
end entity;
architecture comp of BCD_7 is

begin
     with Ent select
SAL<="0000001" when "0000",--0
      "1001111" when "0001",--1
      "0010010" when "0010",--2
      "0000110" when "0011",--3
      "1001100" when "0100",--4
      "0100100" when "0101",--5
      "1100000" when "0110",--6
      "0001111" when "0111",--7
      "0000000" when "1000",--8
      "0001100" when "1001",--9
      "1111111" when "1010",--a
      "1111111" when "1011",--b
      "1111111" when "1100",--c
      "1111111" when "1101",--d
      "1111111" when "1110",--e
      "1111111" when "1111"; --f


end architecture;--todas las operaciones se ejecutan en paralelo
```

DIVISOR DE FRECUENCIA FACTOR 50.000.000 (SALIDA DE 1 HZ)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
ENTITY DIV_SAL_1HZ IS
PORT(
     CLOCK: IN STD_LOGIC;

     SALIDA_1HZ:OUT STD_LOGIC
     );
END ENTITY;
ARCHITECTURE FREC_DIV OF DIV_SAL_1HZ IS
CONSTANT COUNT_LIMIT: INTEGER:=25000000;--3 OBJETOS 2 SEÑALES Y 1 CONSTANTE
SIGNAL COUNT: INTEGER RANGE 0 TO 50000500:=0;--CONSTAN DEFINE EL FACTOR DE DIVISION, RANGO 10 SEG
SIGNAL LED_STATE: STD_LOGIC;
BEGIN
BLINK:PROCESS(CLOCK)
BEGIN
IF RISING_EDGE(CLOCK) THEN
IF(COUNT=COUNT_LIMIT) THEN
LED_STATE<=NOT LED_STATE;
SALIDA_1HZ<=LED_STATE;--AGREGAMOS OTRO PIN PARA 2HZ
COUNT<=0;
ELSE
COUNT<= COUNT+1;
END IF;
END IF;
END PROCESS;
END ARCHITECTURE;
```

DIVISOR DE FRECUENCIA FACTOR 50.000(SALIDA DE 1000 HZ)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
ENTITY DIV_SAL_1000HZ IS
PORT(
     CLOCK: IN STD_LOGIC;

     SALIDA_1000HZ:INOUT STD_LOGIC
     );
END ENTITY;
ARCHITECTURE FREC_DIV OF DIV_SAL_1000HZ IS
CONSTANT COUNT_LIMIT: INTEGER:=25000;--3 OBJETOS, 1KHZ
SIGNAL COUNT: INTEGER RANGE 0 TO 50000500:=0;--CONSTAN DEFINE EL FACTOR DE DIVISION, RANGO 10 SEG
SIGNAL LED_STATE: STD_LOGIC;
BEGIN
BLINK:PROCESS(CLOCK)
BEGIN
IF RISING_EDGE(CLOCK) THEN
IF(COUNT=COUNT_LIMIT) THEN
LED_STATE<=NOT LED_STATE;
SALIDA_1000HZ<=LED_STATE;--AGREGAMOS OTRO PIN PARA 2HZ
COUNT<=0;
ELSE
COUNT<= COUNT+1;
END IF;
END IF;
END PROCESS;
END ARCHITECTURE;
```

CONTADOR DE BINARIO DE “00” A “11”

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity CONT_MUX4 is
port(

     CLOCK:  in std_logic; --50 mhz
     MUXCOUNTER: inout STD_LOGIC_VECTOR(1 DOWNTO 0);--SEÑAL
     SAL_MUX: out std_logic_VECTOR(1 DOWNTO 0)

);
end entity;
architecture comp of CONT_MUX4 is
SIGNAL S_MUXCOUNTER:UNSIGNED(1 DOWNTO 0):="00";--cambio de datos, std_logic_vector no permite op mat.
begin

process(CLOCK)
begin
     IF rising_edge(CLOCK) THEN
     S_MUXCOUNTER<=S_MUXCOUNTER+1;
      END IF;
end process;

SAL_MUX<=STD_LOGIC_VECTOR(S_MUXCOUNTER);--cambiamos el tipo de datos std

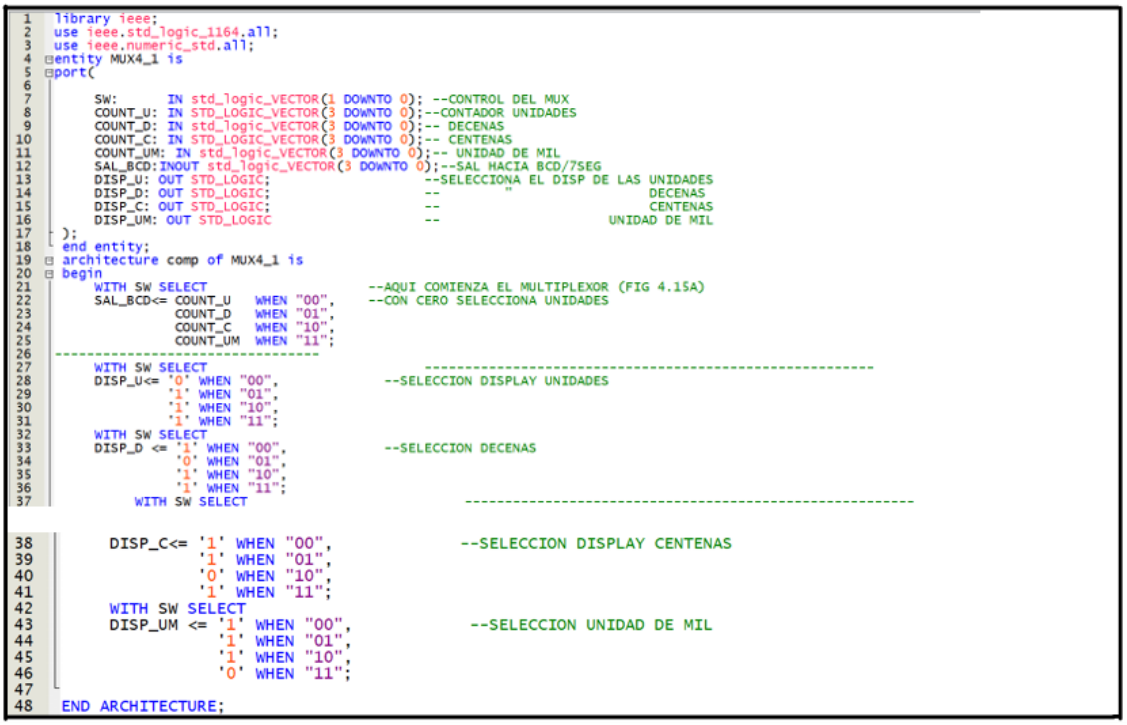
END ARCHITECTURE;
-- Realiza conteo de "00" a "11" para controlar un mux y demux de 4 a 1 linea y de 1 a 4 lineas
-- respectivamente.
```

CONTADOR BCD

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity CONTADOR is
port(

     CLOCK:          in std_logic; --50 mhz
     AUXCOUNTER: inout STD_LOGIC_VECTOR(3 DOWNTO 0);--COMO BUFFER
     SAL_COUNTER:  out std_logic_VECTOR(3 DOWNTO 0);
     SAL_CK:OUT STD_LOGIC                          --SALDIDA PARA EL CLOK para los proximos contadores
);
end entity;
architecture comp of CONTADOR is
SIGNAL S_AUXCOUNTER:UNSIGNED(3 DOWNTO 0):="0000";--cambio de datos, std_logic_vector no permite operaciones
begin                                            -- matematicas.
process(CLOCK)
begin
     IF rising_edge(CLOCK) THEN                  --then --si hay flaco de subida   **1**
     IF S_AUXCOUNTER<9 then                      --cuando es menor a nueve se incrementa la cuenta
        S_AUXCOUNTER<=S_AUXCOUNTER+1;
        SAL_CK<='0';                             --mientras no llegue a 9, no hay pulso ck para los proximos digitos
    ELSE
        S_AUXCOUNTER<="0000";                     --caso contrario (cont=9), reset a cero
        SAL_CK<='1';
   END IF;
   END IF;
end process;
SAL_COUNTER<=STD_LOGIC_VECTOR(S_AUXCOUNTER);       --cambiamos el tipo de datos std
END ARCHITECTURE;
--CONTADOR BCD (0 A 9).Conteo para posteriormente ser decodificado a 7 segmentos.
```

MULTIPLEXOR Y DEMULTIPLEXOR



```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  entity MUX4_1 is
5  port(
6
7       SW:      IN std_logic_VECTOR(1 DOWNTO 0); --CONTROL DEL MUX
8       COUNT_U: IN STD_LOGIC_VECTOR(3 DOWNTO 0);--CONTADOR UNIDADES
9       COUNT_D: IN std_logic_VECTOR(3 DOWNTO 0);-- DECENAS
10      COUNT_C: IN STD_LOGIC_VECTOR(3 DOWNTO 0);-- CENTENAS
11      COUNT_UM: IN std_logic_VECTOR(3 DOWNTO 0);-- UNIDAD DE MIL
12      SAL_BCD:INOUT std_logic_VECTOR(3 DOWNTO 0);--SAL HACIA BCD/7SEG
13      DISP_U: OUT STD_LOGIC;                    --SELECCIONA EL DISP DE LAS UNIDADES
14      DISP_D: OUT STD_LOGIC;                    --                  "       DECENAS
15      DISP_C: OUT STD_LOGIC;                    --                          CENTENAS
16      DISP_UM: OUT STD_LOGIC                    --                      UNIDAD DE MIL
17 );
18 end entity;
19 architecture comp of MUX4_1 is
20 begin
21      WITH SW SELECT                      --AQUI COMIENZA EL MULTIPLEXOR (FIG 4.15A)
22      SAL_BCD<= COUNT_U   WHEN "00",      --CON CERO SELECCIONA UNIDADES
23                COUNT_D   WHEN "01",
24                COUNT_C   WHEN "10";
25                COUNT_UM  WHEN "11";
26 ------------------------------
27      WITH SW SELECT                          ------------------------------------------------------
28      DISP_U<= '0' WHEN "00",             --SELECCION DISPLAY UNIDADES
29               '1' WHEN "01",
30               '1' WHEN "10",
31               '1' WHEN "11";
32      WITH SW SELECT
33      DISP_D <= '1' WHEN "00",            --SELECCION DECENAS
34                '0' WHEN "01",
35                '1' WHEN "10",
36                '1' WHEN "11";
37          WITH SW SELECT                          ------------------------------------------------------

38       DISP_C<= '1' WHEN "00",              --SELECCION DISPLAY CENTENAS
39                '1' WHEN "01",
40                '0' WHEN "10",
41                '1' WHEN "11";
42       WITH SW SELECT
43       DISP_UM <= '1' WHEN "00",            --SELECCION UNIDAD DE MIL
44                  '1' WHEN "01",
45                  '1' WHEN "10",
46                  '0' WHEN "11";
47
48 END ARCHITECTURE;
```

Notar que entre las líneas 21 y 25 corresponde al código del multiplexor, encargado de seleccionar el contador de las unidades, decenas, centenas o unidad de mil. Desde la línea 27 hasta la 47, corresponde al código del demultiplexor de 1 a 4 líneas para seleccionar el display (DIG1, DIG2, DIG3 o DIG4).