

UNIVERSIDAD DE EL SALVADOR  
FACULTAD DE INGENIERÍA Y ARQUITECTURA  
ESCUELA DE INGENIERÍA ELÉCTRICA



# **Principios del FPGA y aplicaciones en el control de procesos industriales.**

PRESENTADO POR:

**BENJAMIN ANTONIO ROBLES RIVAS**

PARA OPTAR AL TÍTULO DE:

**INGENIERO ELECTRICISTA**

CIUDAD UNIVERSITARIA, MARZO DE 2016

**UNIVERSIDAD DE EL SALVADOR**

RECTOR INTERINO :

**LIC. JOSÉ LUIS ARGUETA ANTILLÒN**

SECRETARIA GENERAL :

**DRA. ANA LETICIA ZAVALA DE AMAYA**

**FACULTAD DE INGENIERÍA Y ARQUITECTURA**

DECANO :

**ING. FRANCISCO ANTONIO ALARCÓN SANDOVAL**

SECRETARIO :

**ING. JULIO ALBERTO PORTILLO**

**ESCUELA DE INGENIERÍA ELÉCTRICA**

DIRECTOR :

**ING. ARMANDO MARTÍNEZ CALDERÓN**

UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA DE INGENIERÍA ELÉCTRICA

Trabajo de Graduación previo a la opción al Grado de:

**INGENIERO ELECTRICISTA**

Título :

**Principios del FPGA y aplicaciones en el control de  
procesos industriales.**

Presentado por :

**BENJAMIN ANTONIO ROBLES RIVAS**

Trabajo de Graduación Aprobado por :

Docente Asesor :

**MSC. SALVADOR DE JESÚS GERMAN**

San Salvador, marzo de 2016

Trabajo de Graduación Aprobado por:

Docente Asesor :

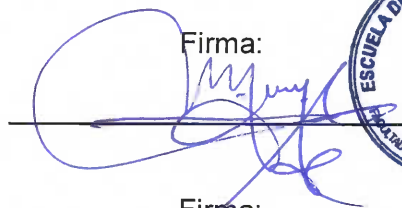
**MSC. SALVADOR DE JESÚS GERMAN**

## ACTA DE CONSTANCIA DE NOTA Y DEFENSA FINAL

En esta fecha, Martes 8 de marzo de 2016, en la Sala de Lectura de la Escuela de Ingeniería Eléctrica, a las 10:00 a.m. horas, en presencia de las siguientes autoridades de la Escuela de Ingeniería Eléctrica de la Universidad de El Salvador:

1. Ing. Armando Martínez Calderón  
Director

Firma:



2. MSc. José Wilber Calderón Urrutia  
Secretario

Firma:

Wilber Calderón

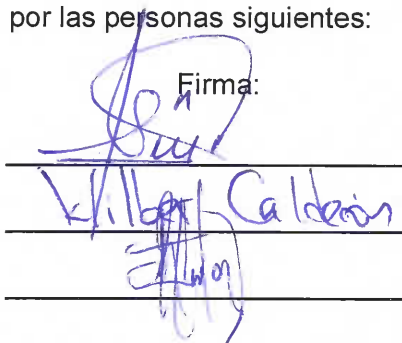
Y, con el Honorable Jurado de Evaluación integrado por las personas siguientes:

1- MSc. Salvador de Jesús German

2- MSc. José Wilber Calderón Urrutia

3- MSc. Ricardo Ernesto Cortez

Firma:



Se efectuó la defensa final reglamentaria del Trabajo de Graduación:

Principios del FPGA y aplicaciones en el control de procesos industriales.

A cargo del Bachiller:

- Robles Rivas Benjamín Antonio

Habiendo obtenido en el presente Trabajo una nota promedio de la defensa final:

7.9

( siete.nove )

## *Agradecimientos*

---

La honra y gloria sea para Dios que me ha permitido llegar hasta este momento, para lo cual me ha dado mucha fortaleza y las virtudes necesarias para finalizar mis estudios, me ha dado la salud para avanzar en este camino y me ha proveído de los medios necesarios para culminar mi carrera profesional.

Agradezco a mi madre María Emilia Rivas que siempre estuvo a mi lado apoyándome paso a paso, dándome el estímulo y los correctivos que eran necesarios para alcanzar esta meta, a mis hermanos Jaime Osmar Velasquez y Maria Jose Robles con los cuales he compartido desde mi infancia y que han significado una parte importante en mi vida ya que a través de la experiencia cotidiana me han dado lecciones invaluable, a mi padre Antonio Cruz Robles el cual ya no está con nosotros quiero dedicarle una mención especial, ya que el siempre quiso lo mejor para mí y se esmero por educarme lo mejor que pudo durante el tiempo que estuvo conmigo.

A mi abuelita María Isabel Rivas que siempre tuvo fe en mí y que con sus consejos y sonrisas me llena de alegría y me motiva a ser una mejor persona, a mis tíos también quiero agradecerles por estar pendientes de mi cada uno a su forma.

A todos mis amigos que han estado conmigo a lo largo de este proceso, a los que conozco de infancia, a aquellos con los cuales conviví en la universidad, a los que he ido encontrando en cada una de las etapas que he vivido solo les resto decirles gracias, porque de cada uno he aprendido algo bueno.

Finalmente agradecer a mi asesor Ing. Salvador German que tuvo fe en este proyecto, y con paciencia me fue apoyando y asesorando en la recta final de mi carrera para alcanzar el objetivo propuesto.

# Índice de contenido

INTRODUCCIÓN	1
OBJETIVOS	2
<i>Generales</i>	2
<i>Específicos</i>	2
<b>CAPITULO 1: TEORÍA GENERAL</b>	<b>3</b>
1.1 PRELIMINARES	3
1.2 ¿POR QUÉ UN FPGA?	4
1.3 FPGA Y SUS APLICACIONES	5
1.4 TECNOLOGÍAS CONSTRUCTIVAS	6
1.4.1 <i>Xilinx</i>	7
1.4.1.1 CLB	7
1.4.1.2 Slices	7
1.4.1.3 Líneas de conexión (interconnect)	8
1.4.1.4 IOB	8
1.4.1.5 Memory RAM blocks	8
1.4.1.6 Complete clock management	9
1.4.2 <i>Altera</i>	9
1.4.2.1 Logic Array Block	9
1.4.2.2 Adaptative Logic Module	10
1.4.2.3 Logic Element	11
1.4.2.4 Look Up Table	11
1.4.2.5 Registros programables	12
1.4.2.6 Cadenas de registro y acarreo	12
1.4.2.7 Señales de control	12
1.4.2.8 Bloques I/O	12
1.4.2.9 Bloques de memoria	13
1.4.2.10 Multiplicadores embebidos	13
1.4.2.11 Transeptores de alta velocidad	13
1.4.2.12 Sincronización	13
1.5 ALTERA DE1 SOC	13
1.5.1 <i>Especificaciones</i>	14
1.5.1.1 FPGA	16
1.5.1.2 Configuración y Depuración	16
1.5.1.3 Memoria	17
1.5.1.4 Comunicaciones	17
1.5.1.5 Conectores	17
1.5.1.6 Video	18
1.5.1.7 Audio	18
1.5.1.8 ADC	18
1.5.1.9 Elementos de lógica digital	18
1.5.1.10 Sensores	19
1.5.1.11 Fuente	19
1.5.1.12 Circuitería de reloj	19
1.5.2 <i>Configuración de FPGA</i>	20
1.5.3 <i>Métodos de programación</i>	20

<b>CAPITULO 2: SOFTWARE</b>	<b>22</b>
2.1 INTRODUCCIÓN	22
2.2 DESCARGA DEL SOFTWARE	22
2.3 INSTALACIÓN DEL SOFTWARE	25
2.4 HERRAMIENTAS	34
2.4.1 Creación de un proyecto	34
2.4.2 Editor de texto	41
2.4.3 Análisis y síntesis	47
2.4.4 Asignación de pines	49
2.4.4.1 Asignación manual	49
2.4.4.2 Asignación por archivo de ajuste (recomendado)	54
2.4.5 Compilación	56
2.4.6 RTL Netlist Viewer y Technolgy Map Viewer	58
2.4.6.1 RTL Viewer.	58
2.4.6.2 Technology Map Viewer	60
2.4.7 Simulación	62
2.5 PROGRAMMER	71
2.5.1 Programar archivos SOF	73
2.5.2 Programar archivo JIC	77
<b>CAPÍTULO 3: LABORATORIOS</b>	<b>84</b>
3.1 INTRODUCCIÓN	84
3.2 LABORATORIO NO. 1: INTERRUPTORES, LEDS Y MULTIPLEXORES	84
3.2.1 Introducción.	84
3.2.2 Objetivos generales	84
3.2.3 Asignación 1	85
3.2.3.1 Objetivo	85
3.2.3.2 Desarrollo	85
3.2.3.3 Procedimiento general	86
3.2.4 Asignación 2	87
3.2.4.1 Objetivo	87
3.2.4.2 Desarrollo	87
3.2.4.3 Procedimiento	89
3.2.5 Asignación 3	89
3.2.5.1 Objetivo	89
3.2.5.2 Desarrollo	89
3.2.5.3 Procedimiento	90
3.2.6 Asignación 4	91
3.2.6.1 Objetivo	91
3.2.6.2 Desarrollo	91
3.2.6.3 Procedimiento	92
3.2.7 Asignación 5	92
3.2.7.1 Objetivo	92
3.2.7.2 Desarrollo	92
3.2.7.3 Procedimiento	94
3.2.8 Asignación 6	95
3.2.8.1 Objetivo	95
3.2.8.2 Desarrollo	95



3.2.8.3 Procedimiento	95
3.3 LABORATORIO NO. 2: NÚMEROS Y DISPLAY	97
3.3.1 <i>Introducción</i>	97
3.3.2 <i>Objetivos Generales</i>	97
3.3.3 <i>Asignación 1</i>	97
3.3.3.1 Objetivo	97
3.3.3.2 Desarrollo	97
3.3.3.3 Procedimiento	99
3.3.4 <i>Asignación 2</i>	99
3.3.4.1 Objetivo	99
3.3.4.2 Procedimiento	99
3.3.4.3 Procedimiento	101
3.3.5 <i>Asignación 3</i>	101
3.3.5.1 Objetivo	101
3.3.5.2 Desarrollo	101
3.3.5.3 Procedimiento	102
3.3.6 <i>Asignación 4</i>	103
3.3.6.1 Objetivos	103
3.3.6.2 Desarrollo	103
3.3.6.3 Procedimiento	103
3.3.7 <i>Asignación 5</i>	104
3.3.7.1 Objetivo	104
3.3.7.2 Desarrollo	104
3.3.7.3 Procedimiento	104
3.3.8 <i>Asignación 6</i>	105
3.3.8.1 Objetivos	105
3.3.8.2 Desarrollo	105
3.3.8.3 Procedimiento	105
3.4 LABORATORIO NO. 3: LATCHES, FLIP FLOPS Y REGISTROS	106
3.4.1 <i>Introducción</i>	106
3.4.2 <i>Objetivos generales</i>	106
3.4.3 <i>Asignación 1</i>	106
3.4.3.1 Objetivo	106
3.4.3.2 Desarrollo	106
3.4.3.3 Procedimiento	108
3.4.4 <i>Asignación 2</i>	109
3.4.4.1 Objetivo	109
3.4.4.2 Desarrollo	109
3.4.4.3 Procedimiento	109
3.4.5 <i>Asignación 3</i>	110
3.4.5.1 Objetivo	110
3.4.5.2 Desarrollo	110
3.4.5.3 Procedimiento	110
3.4.6 <i>Asignación 4</i>	111
3.4.6.1 Objetivos	111
3.4.6.2 Desarrollo	111
3.4.7 <i>Asignación 5</i>	113
3.4.7.1 Objetivos	113
3.4.7.2 Desarrollo	113

3.4.7.3 Procedimiento	113
3.5 LABORATORIO NO. 4: CONTADORES	114
3.5.1 <i>Introducción</i>	114
3.5.2 <i>Objetivos Generales</i>	114
3.5.3 <i>Asignación 1</i>	114
3.5.3.1 Objetivo	114
3.5.3.2 Desarrollo	114
3.5.3.3 Procedimiento	117
3.5.4 <i>Asignación 2</i>	117
3.5.4.1 Objetivo	117
3.5.4.2 Desarrollo	117
3.5.4.3 Procedimiento	117
3.5.5 <i>Asignación 3</i>	118
3.5.5.1 Objetivo	118
3.5.5.2 Desarrollo	118
3.5.5.3 Procedimiento	118
3.5.6 <i>Asignación 4</i>	118
3.5.6.1 Objetivo	118
3.5.6.2 Desarrollo	118
3.5.6.3 Procedimiento	119
3.5.7 <i>Asignación 5</i>	119
3.5.7.1 Objetivo	119
3.5.7.2 Desarrollo	119
3.5.7.3 Procedimiento	120
3.5.8 <i>Asignación 6</i>	120
3.5.8.1 Objetivo	120
3.5.8.2 Desarrollo	120
3.5.8.3 Procedimiento	121
3.6 LABORATORIO NO. 5: TEMPORIZADORES Y RELOJ EN TIEMPO REAL	122
3.6.1 <i>Introducción</i>	122
3.6.2 <i>Objetivos Generales</i>	122
3.6.3 <i>Preliminares</i>	122
3.6.4 <i>Asignación 1</i>	123
3.6.4.1 Objetivo	123
3.6.4.2 Desarrollo	123
3.6.4.3 Procedimiento	124
3.6.5 <i>Asignación 2</i>	125
3.6.5.1 Objetivo	125
3.6.5.2 Desarrollo	125
3.6.5.3 Procedimiento	125
3.6.6 <i>Asignación 3</i>	125
3.6.6.1 Objetivo	125
3.6.6.2 Desarrollo	125
3.6.6.3 Asignación Extra	125
3.7 LABORATORIO NO. 6: SUMADORES, RESTADORES Y MULTIPLICADORES	126
3.7.1 <i>Introducción</i>	126
3.7.2 <i>Objetivos Generales</i>	126
3.7.3 <i>Asignación 1</i>	126
3.7.3.1 Objetivo	126

3.7.3.2 Desarrollo	126
3.7.3.3 Procedimiento	129
3.7.4 <i>Asignación 2</i>	130
3.7.4.1 Objetivo	130
3.7.4.2 Desarrollo	130
3.7.4.3 Procedimiento	130
3.7.5 <i>Asignación 3</i>	130
3.7.5.1 Objetivo	130
3.7.5.2 Desarrollo	130
3.7.5.3 Desarrollo	131
3.7.6 <i>Asignación 4</i>	132
3.7.6.1 Objetivo	132
3.7.6.2 Desarrollo	132
3.7.6.3 Procedimiento	133
3.7.7 <i>Asignación 5</i>	134
3.7.7.1 Objetivo	134
3.7.7.2 Desarrollo	134
3.8 LABORATORIO NO. 7: MÁQUINAS DE ESTADO FINITO	136
3.8.1 <i>Introducción</i>	136
3.8.2 <i>Objetivos Generales</i>	136
3.8.3 <i>Asignación 1</i>	136
3.8.3.1 Objetivo	136
3.8.3.2 Desarrollo	136
3.8.3.3 Procedimiento	139
3.8.4 <i>Asignación 2</i>	140
3.8.4.1 Objetivo	140
3.8.4.2 Desarrollo	140
3.8.4.3 Procedimiento	141
3.8.5 <i>Asignación 3</i>	142
3.8.5.1 Objetivo	142
3.8.5.2 Desarrollo	142
3.8.5.3 Procedimiento	142
3.9 LABORATORIO NO. 8: BLOQUES DE MEMORIA	143
3.9.1 <i>Introducción</i>	143
3.9.2 <i>Objetivos Generales</i>	144
3.9.3 <i>Asignación 1</i>	144
3.9.3.1 Objetivo	144
3.9.3.2 Desarrollo	144
3.9.4 <i>Asignación 2</i>	147
3.9.4.1 Objetivo	147
3.9.4.2 Desarrollo	147
3.9.4.3 Procedimiento	147
3.9.5 <i>Asignación 3</i>	148
3.9.5.1 Objetivo	148
3.9.5.2 Desarrollo	148
3.9.5.3 Procedimiento	148
3.9.6 <i>Asignación 4</i>	148
3.9.6.1 Objetivo	148
3.9.6.2 Desarrollo	148

3.10 LABORATORIO NO. 9: PROCESADOR SIMPLE-----	151
3.10.1 <i>Introducción</i> -----	151
3.10.2 <i>Objetivos generales</i> -----	151
3.10.3 <i>Asignación 1</i> -----	151
3.10.3.1 <i>Objetivo</i> -----	151
3.10.3.2 <i>Desarrollo</i> -----	151
3.10.3.3 <i>Procedimiento</i> -----	153
3.10.4 <i>Asignación 2</i> -----	158
3.10.4.1 <i>Objetivo</i> -----	158
3.10.4.2 <i>Desarrollo</i> -----	159
3.10.4.3 <i>Procedimiento</i> -----	159
3.11 LABORATORIO NO. 10: PROCESADOR AVANZADO-----	162
3.11.1 <i>Introducción</i> -----	162
3.11.2 <i>Objetivos Generales</i> -----	162
3.11.3 <i>Asignación 1</i> -----	162
3.11.3.1 <i>Objetivo</i> -----	162
3.11.3.2 <i>Desarrollo</i> -----	162
3.11.3.3 <i>Procedimiento</i> -----	165
3.11.4 <i>Asignación 2</i> -----	173
3.11.4.1 <i>Objetivo</i> -----	173
3.11.4.2 <i>Desarrollo</i> -----	173
3.11.4.3 <i>Procedimiento</i> -----	173
<b>CAPITULO 4: FPGA PARA CONTROL DE PROCESOS. -----</b>	<b>174</b>
4.1 <i>INTRODUCCIÓN</i> -----	174
4.2 <i>METODOLOGIA PARA LA BÚSQUEDA DE SOLUCIONES</i> -----	174
4.3 <i>BANDA EMPACADORA DE FRUTAS</i> -----	176
4.3.1 <i>Definición del problema</i> -----	176
4.3.2 <i>Solución</i> -----	176
4.3.3 <i>Codificación para FPGA</i> -----	177
4.3.4 <i>Circuitería extra</i> -----	182
4.3.4.1 <i>Circuito de fuerza</i> -----	182
4.3.4.2 <i>Sensores</i> -----	183
<b>CONCLUSIONES -----</b>	<b>185</b>
<b>GLOSARIO-----</b>	<b>186</b>
<b>BIBLIOGRAFÍA-----</b>	<b>187</b>
<b>REFERENCIAS-----</b>	<b>188</b>

## Índice de figuras

<b>FIGURA NO. 1.1: ESTRUCTURA BÁSICA DE UN FPGA XILINX.-----</b>	<b>7</b>
<b>FIGURA NO. 1.2: ESTRUCTURA INTERNA DE UN CLB.-----</b>	<b>8</b>
<b>FIGURA NO. 1.3: ESTRUCTURA DE BANCOS IOB.-----</b>	<b>8</b>

FIGURA NO. 1.4: DIAGRAMA DE BLOQUES PARA UN ALM. <sup>[1]</sup> .....	10
FIGURA NO. 1.5: DIAGRAMA DE BLOQUES PARA UN LE. <sup>[2]</sup> .....	11
FIGURA NO. 1.6: ESTRUCTURA DE UNA LUT. <sup>[3]</sup> .....	12
FIGURA NO. 1.7: ALTERA DE1-SOC. <sup>[4]</sup> .....	14
FIGURA NO. 1.8: MAPA DE ELEMENTOS DE LA TARJETA DE1-SOC. <sup>[5]</sup> .....	15
FIGURA NO. 1.9: DIAGRAMA DE BLOQUES DE LA TARJETA DE1-SOC. <sup>[5]</sup> .....	15
FIGURA NO. 1.10: NOMENCLATURA PARA DISPOSITIVOS DE LA FAMILIA CYCLONE V SE. <sup>[6]</sup> .....	16
FIGURA NO. 1.11: DIAGRAMA DE CONEXIÓN DE PINES GPIO. <sup>[5]</sup> .....	18
FIGURA NO. 1.12: DIAGRAMA DE CONEXIÓN PARA BOTONES PULSADORES. <sup>[5]</sup> .....	19
FIGURA NO. 1.13: DIAGRAMA DE BLOQUES PARA LA CIRCUITERÍA DE RELOJ. <sup>[5]</sup> .....	19
FIGURA NO. 1.14: DIPSWITCH DE CONFIGURACIÓN DE TARJETA DE1-SOC. <sup>[5]</sup> .....	20
FIGURA NO. 1.15: DIAGRAMA DE BLOQUES PARA PROGRAMACIÓN JTAG. <sup>[5]</sup> .....	21
FIGURA NO. 1.16: DIAGRAMA DE BLOQUES PARA PROGRAMACIÓN AS. <sup>[5]</sup> .....	21
FIGURA NO.2.1: SELECTOR DE SOFTWARE POR DISPOSITIVO. ....	23
FIGURA NO. 2.2: SELECCIÓN DE SISTEMA OPERATIVO PARA EL SOFTWARE. ....	24
FIGURA NO. 2.3: ARCHIVOS NECESARIOS PARA LA INSTALACIÓN. ....	24
FIGURA NO. 2.4: PANTALLA INICIAL DE INSTALACIÓN. ....	25
FIGURA NO.2.5: TÉRMINOS Y CONDICIONES. ....	26
FIGURA NO. 2.6: DIRECTORIO DE INSTALACIÓN. ....	26
FIGURA NO. 2.7: COMPONENTES DE QUARTUS II. ....	27
FIGURA NO. 2.8: REQUERIMIENTO EN DISCO DURO. ....	27
FIGURA NO. 2.9: PROCESO DE INSTALACIÓN. ....	28
FIGURA NO. 2.10: FINALIZACIÓN DE INSTALACIÓN. ....	28
FIGURA NO. 2.11: INSTALACIÓN DE DRIVERS USB BLASTER II. ....	29
FIGURA NO. 2.12: INSTALACIÓN CORRECTA DEL USB BLASTER II. ....	29
FIGURA NO. 2.13: PANTALLA INICIAL QUARTUS II. ....	30
FIGURA NO. 2.14: INSTALACIÓN DE QUARTUS II PROGRAMMER. ....	30
FIGURA NO. 2.15: TÉRMINOS Y CONDICIONES DEL SOFTWARE. ....	31
FIGURA NO. 2.16: DIRECTORIO DE INSTALACIÓN. ....	32
FIGURA NO. 2.17: ESPACIO REQUERIDO EN DISCO DURO. ....	32
FIGURA NO. 2.18: PROCESO DE INSTALACIÓN. ....	33
FIGURA NO. 2.19: INTERFAZ PRINCIPAL DEL PROGRAMMER. ....	33

FIGURA NO. 2.20: ICONO DE QUARTUS II. -----	34
FIGURA NO. 2.21: PROCESO ALTERNATIVO PARA INICIAR QUARTUS II. -----	35
FIGURA NO. 2.22: OPCIÓN PRACTICA PARA INICIAR UN NUEVO PROYECTO. -----	35
FIGURA NO. 2.23: OPCIÓN FORMAL PARA INICIAR UN NUEVO PROYECTO. -----	36
FIGURA NO. 2.24: PANTALLA DE INTRODUCCIÓN DEL ASISTENTE DE PROYECTOS. -----	36
FIGURA NO. 2.25: DIRECTORIO DE PROYECTO. -----	37
FIGURA NO. 2.26: TIPO DE PROYECTO. -----	38
FIGURA NO. 2.27: ASISTENTE PARA AÑADIR ARCHIVOS EXTRA AL PROYECTO. -----	38
FIGURA NO. 2.28: ASISTENTE DE SELECCIÓN DE DISPOSITIVOS. -----	39
FIGURA NO. 2.29: HERRAMIENTAS EDA. -----	40
FIGURA NO. 2.30: REPORTE DE CREACIÓN DEL PROYECTO. -----	40
FIGURA NO. 2.31: INVOCACIÓN DE NUEVOS ARCHIVOS. -----	41
FIGURA NO. 2.32: TIPOS DE ARCHIVOS PARA QUARTUS II. -----	41
FIGURA NO. 2.33: INTERFAZ PRINCIPAL PARA EDITOR DE TEXTO. -----	42
FIGURA NO. 2.34: PROCESO PARA GUARDAR UN ARCHIVO. -----	43
FIGURA NO. 2.35: DIRECTORIO DE ARCHIVOS. -----	43
FIGURA NO. 2.36: ARCHIVO DE TEXTO GUARDADO CON ÉXITO. -----	44
FIGURA NO. 2.37. PROCESO PARA ACCEDER A MENÚ DE OPCIONES. -----	44
FIGURA NO. 2.38. MENÚ DE OPCIONES PARA EL EDITOR DE TEXTO. -----	45
FIGURA NO. 2.39: HERRAMIENTA DE AUTORELLENO. -----	46
FIGURA NO. 2.40: PROCESO PARA ACCEDER A LAS PLANTILLAS. -----	46
FIGURA NO. 2.41: MENÚ DE PLANTILLAS. -----	47
FIGURA NO. 2.42: PROCESO DE ANÁLISIS. -----	47
FIGURA NO. 2.43: DEPURADOR. -----	48
FIGURA NO. 2.44: REPORTE PRELIMINAR. -----	49
FIGURA NO. 2.45: PROCESO PARA ASIGNACIÓN DE PINES. -----	49
FIGURA NO. 2.46: EDITOR DE PINES. -----	50
FIGURA NO. 2.47: ASIGNACIÓN DE PINES. -----	50
FIGURA NO. 2.48: NODE FINDER (I). -----	51
FIGURA NO. 2.49: NODE FINDER (II). -----	51
FIGURA NO. 2.50: TABLA CON TODOS LOS PINES. -----	52
FIGURA NO. 2.51: DEFINICIÓN DEL TIPO DE PIN. -----	52

FIGURA NO. 2.52: ASIGNACIÓN DE PINES COMPLETA.-----	53
FIGURA NO. 2.53: PIN PLANNER. -----	54
FIGURA NO. 2.54: DESCARGA DE ARCHIVO QSF.-----	54
FIGURA NO. 2.55: IMPORTE DE AJUSTES. -----	55
FIGURA NO. 2.56: IMPORTE DE ARCHIVO QSF.-----	55
FIGURA NO. 2.57: BÚSQUEDA DE ARCHIVO QSF. -----	55
FIGURA NO. 2.58: ARCHIVO AÑADIDO.-----	56
FIGURA NO. 2.59: PIN PLANNER MÉTODO ALTERNATIVO.-----	56
FIGURA NO. 2.60: PROCESO DE COMPILACIÓN.-----	57
FIGURA NO. 2.61: REPORTE FINAL DE COMPILACIÓN. -----	57
FIGURA NO. 2.62: TIEMPO REQUERIDO PARA LA COMPILACIÓN DEL CÓDIGO TUTORIAL. -----	58
FIGURA NO. 2.63: PROCESO PARA RTL VIEWER. -----	58
FIGURA NO. 2.64: INSTANCIA PRINCIPAL.-----	58
FIGURA NO. 2.65: ELEMENTO COUNTER N2 VISTO A DETALLE. -----	59
FIGURA NO. 2.66: PROCESO PARA EXPORTAR ARCHIVOS RTL. -----	59
FIGURA NO. 2.67: PROCESO PARA CREAR ARCHIVO PDF.-----	60
FIGURA NO. 2.68: PROCESO PARA TECHNOLOGY MAP VIEWER.-----	60
FIGURA NO. 2.69: INSTANCIA PRINCIPAL.-----	61
FIGURA NO. 2.70: ELEMENTO COUNTER N2 VISTO A DETALLE. -----	61
FIGURA NO. 2.71: PROCESO PARA EXPORTAR ARCHIVOS TECHNOLOGY MAP. -----	62
FIGURA NO. 2.72: PROCESO PARA CREAR ARCHIVO PDF.-----	62
FIGURA NO. 2.73: CONTADOR DE 4 BITS BASADO EN FF'S T -----	63
FIGURA NO. 2.74: PROCESO PARA CREAR UN ARCHIVO VWF.-----	63
FIGURA NO. 2.75: SELECCIÓN DE ARCHIVO VWF.-----	64
FIGURA NO. 2.76: SELECCIÓN DE TIEMPO DE SIMULACIÓN. -----	64
FIGURA NO. 2.77: TIEMPO DE SIMULACIÓN EN US.-----	65
FIGURA NO. 2.78: SELECCIÓN DEL TAMAÑO DE LA GRID. -----	65
FIGURA NO. 2.79: TAMAÑO DE LOS INTERVALOS. -----	65
FIGURA NO. 2.80: SELECCIÓN DE NODOS (I).-----	66
FIGURA NO. 2.81: SELECCIÓN DE NODOS (II).-----	66
FIGURA NO. 2.82: SELECCIÓN DE NODOS (III).-----	67
FIGURA NO. 2.83: LISTA DE NODOS DISPONIBLES. -----	67

FIGURA NO. 2.84: LISTA DE NODOS SELECCIONADOS. -----	67
FIGURA NO. 2.85: SELECCIÓN DE NODOS (IV).-----	68
FIGURA NO. 2.86: ASIGNACIÓN DE SEÑALES. -----	68
FIGURA NO. 2.87: ASIGNACIÓN DEL PERIODO DE RELOJ. -----	69
FIGURA NO. 2.88: ASIGNACIÓN PARA SW(0).-----	69
FIGURA NO. 2.89: ASIGNACIÓN PARA SW(1).-----	69
FIGURA NO. 2.90: DIAGRAMA DE TIEMPO CON ASIGNACIONES. -----	70
FIGURA NO. 2.91: GUARDAR ARCHIVO DE SIMULACIÓN. -----	70
FIGURA NO. 2.92: DIRECTORIO DEL ARCHIVO. -----	71
FIGURA NO. 2.93: SIMULACIÓN COMPLETA. -----	71
FIGURA NO. 2.94: MÉTODO I PROGRAMMER TOOL. -----	72
FIGURA NO. 2.95: MÉTODO II PROGRAMMER TOOL. -----	72
FIGURA NO. 2.96: INTERFAZ PRINCIPAL DE PROGRAMMER TOOL.-----	73
FIGURA NO. 2.97: DISPOSITIVO USB BLASTER II CONFIGURADO EXITOSAMENTE. -----	74
FIGURA NO. 2.98: HARDWARE SETUP. -----	74
FIGURA NO. 2.99: AUTO DETECT. -----	75
FIGURA NO. 2.100: SELECCIÓN DE DISPOSITIVO. -----	75
FIGURA NO. 2.101: AGREGAR ARCHIVO PARA SER PROGRAMADO EN FPGA. -----	76
FIGURA NO. 2.102: DIRECTORIO DE ARCHIVO A PROGRAMAR. -----	76
FIGURA NO. 2.103: PROGRAM/CONFIGURE.-----	77
FIGURA NO. 2.104: PROCESO FINALIZADO.-----	77
FIGURA NO. 2.105: PROCEDIMIENTO PARA CONVERSIÓN DE ARCHIVOS. -----	78
FIGURA NO. 2.106: PARÁMETROS DE ARCHIVO JIC. -----	79
FIGURA NO. 2.107: SELECCIÓN DE ARCHIVO SOF. -----	79
FIGURA NO. 2.108: DIRECTORIO ARCHIVO SOF.-----	80
FIGURA NO. 2.109: SELECTOR DE DISPOSITIVOS. -----	80
FIGURA NO: 2.110: VENTANA CON PARÁMETROS COMPLETOS.-----	81
FIGURA NO. 2.111: PROCESO DE GENERACIÓN DE ARCHIVO JIC EXITOSO. -----	81
FIGURA NO. 2.112: SELECCIÓN DE ARCHIVO JIC.-----	82
FIGURA NO. 2.113: ERASE DISPOSITIVO EPCS128.-----	82
FIGURA NO. 2.114: CONFIGURACIÓN DE DISPOSITIVO EPCS128. -----	83
FIGURA NO. 3.1.1A: BLOQUE VHDL PARA ASIGNACIÓN DE ENTRADAS Y SALIDAS. -----	85



FIGURA NO. 3.1.1B: ASIGNACIÓN INDIVIDUAL. -----	85
FIGURA NO. 3.1.2: A) CIRCUITO MULTIPLEXOR 2 A 1. <sup>[7]</sup> -----	87
FIGURA NO. 3.1.2: B) TABLA DE VERDAD, C) SÍMBOLO. <sup>[7]</sup> -----	87
FIGURA NO. 3.1.3: USO DE SIGNAL. -----	88
FIGURA NO. 3.1.4: 4 BITS MULTIPLEXOR DE 2 A 1. <sup>[7]</sup> -----	89
FIGURA NO. 3.1.5: MULTIPLEXOR DE 3 A 1. <sup>[7]</sup> -----	90
FIGURA NO. 3.1.6: DECODIFICADOR DE 7 SEGMENTOS PARA 2 ENTRADAS. <sup>[7]</sup> -----	91
FIGURA NO. 3.1.7: CIRCUITO DE LA ASIGNACIÓN. <sup>[7]</sup> -----	93
FIGURA NO. 3.1.8: ESQUELETO DE CÓDIGO PARA IMPLEMENTAR EL CIRCUITO DE LA FIGURA 3.1.7. -----	94
FIGURA NO. 3.2.1: CÓDIGO PARA ASIGNACIÓN 1. -----	98
FIGURA NO. 3.2.2: DISEÑO PARCIAL PARA CIRCUITO DE ASIGNACIÓN 2. <sup>[8]</sup> -----	100
FIGURA NO. 3.2.3: A) CIRCUITO FULL ADDER B) SIMBOLO FULL ADDER. <sup>[8]</sup> -----	102
FIGURA NO. 3.2.3: C) TABLA DE VERDAD FULL ADDER, D) CIRCUITO RIPPLE-CARRY ADDER. <sup>[8]</sup> -----	102
FIGURA NO. 3.2.4: ALGORITMO PARA UN SUMADOR BCD. <sup>[8]</sup> -----	104
FIGURA NO. 3.3.1: CIRCUITO LATCH RS GATED. <sup>[9]</sup> -----	107
FIGURA NO. 3.3.2: LOGICA DESCRIPTIVA DE UN CIRCUITO LATCH RS GATED -----	107
FIGURA NO. 3.3.3A: DIFERENTES IMPLEMENTACIONES DE LATCH RS (MODELO FORMAL). <sup>[9]</sup> -----	107
FIGURA NO. 3.3.3B: DIFERENTES IMPLEMENTACIONES DE LATCH RS (MODELO DESCRIPTIVO). <sup>[9]</sup> -----	108
FIGURA NO. 3.3.4: SIMULACION DE LATCH RS. <sup>[9]</sup> -----	109
FIGURA NO. 3.3.5: LATCH TIPO D GATED. <sup>[9]</sup> -----	109
FIGURA NO. 3.3.6: FF TIPO D EN CONEXIÓN MAESTRO-ESCLAVO. <sup>[9]</sup> -----	110
FIGURA NO. 3.3.7:A) TIPOS DE LATCHES Y SUS RESPECTIVOS DIAGRAMAS DE TIEMPO (B). <sup>[9]</sup> -----	112
FIGURA NO. 3.3.8: FORMATO DE CÓDIGO QUE DESCRIBE EL COMPORTAMIENTO DE LATCH TIPO D. -----	113
FIGURA NO. 3.4.1: CONTADOR SÍNCRONO DE 4 BITS. <sup>[10]</sup> -----	115
FIGURA NO 3.4.2: CÓDIGO DE ASIGNACIÓN 1 -----	117
FIGURA NO. 3.4.3: DISEÑO PARCIAL PARA UN CONTADOR LENTO. <sup>[10]</sup> -----	119
FIGURA NO. 3.5.1: CODIGO VHDL PARA UN CONTADOR DE N-BITS. <sup>[11]</sup> -----	123
FIGURA NO. 3.5.2: DEFINICIÓN DE UN COMPONENTE COUNTER. <sup>[11]</sup> -----	123
FIGURA NO. 3.5.3: CONTADOR DE MODULO K. -----	124
FIGURA NO. 3.6.1: SUMADOR RIPPLE-CARRY DE CUATRO BITS. <sup>[12]</sup> -----	126
FIGURA NO. 3.6.2: SUMADOR ACUMULADOR. <sup>[12]</sup> -----	127
FIGURA NO. 3.6.3: CÓDIGO CORRESPONDIENTE A LA ASIGNACIÓN 1. -----	129

FIGURA NO. 3.6.4: CIRCUITO ACUMULADOR DE 8 BITS. <sup>[12]</sup> .....	129
FIGURA NO. 3.6.5: MULTIPLICACIÓN DE DOS NÚMEROS A Y B. <sup>[12]</sup> .....	131
FIGURA NO. 3.6.6: CIRCUITO MULTIPLICADOR DE 4 BITS. <sup>[12]</sup> .....	132
FIGURA NO. 3.6.7: CIRCUITO MULTIPLICADOR MATRICIAL IMPLEMENTADO CON SUMADORES DE N-BITS. <sup>[12]</sup> .....	133
FIGURA NO. 3.6.8: CIRCUITO MULTIPLICADOR CON REGISTROS. <sup>[12]</sup> .....	134
FIGURA NO. 3.6.9: ÁRBOL DE SUMAS. <sup>[12]</sup> .....	135
FIGURA NO. 3.7.1: DIAGRAMA DE TIEMPO. <sup>[13]</sup> .....	136
FIGURA NO. 3.7.2: DIAGRAMA DE ESTADOS. <sup>[13]</sup> .....	137
FIGURA NO 3.7.3: CÓDIGO PARA TABLA 3.7.1. ....	139
FIGURA NO 3.7.4: ESQUELETO DE CÓDIGO PARA UNA FSM POR CÓDIGOS BINARIOS. <sup>[13]</sup> .....	141
FIGURA NO. 3.7.5: ASIGNACIÓN DE MÉTODO DE SÍNTESIS EN QUARTUS II. <sup>[13]</sup> .....	142
FIGURA NO. 3.8.1: DIAGRAMAS PARA BLOQUES RAM. <sup>[14]</sup> .....	143
FIGURA NO. 3.8.2: CÓDIGO PARA INICIALIZAR EL MODULO RAM. <sup>[14]</sup> .....	145
FIGURA NO. 3.8.3: A) MENU PRINCIPAL DE IP CATALOG. ....	146
FIGURA NO. 3.8.3: B) CONFIGURACIÓN DE LOS PUERTOS DE ENTRADA Y SALIDA. ....	146
FIGURA NO. 3.8.4: CONFIRMACIÓN DE ADICCIÓN DE DATOS. ....	147
FIGURA NO. 3.8.5: SELECCIÓN DE ARCHIVO MIF. ....	149
FIGURA NO. 3.8.6: FORMATO DE ARCHIVO MIF .....	150
FIGURA NO. 3.9.1: PROCESADOR SIMPLE. <sup>[15]</sup> .....	152
FIGURA NO. 3.9.2: CÓDIGO PARA ASIGNACIÓN 1 .....	157
FIGURA NO. 3.9.3 SIMULACIÓN DEL PROCESADOR SIMPLE. ....	158
FIGURA NO. 3.9.4: PROCESADOR MAS BLOQUE ROM. <sup>[15]</sup> .....	159
FIGURA NO. 3.9.5: CONFIGURACIÓN DEL TAMAÑO DE LA MEMORIA. ....	160
FIGURA NO 3.9.6: ASIGNACIÓN DEL ARCHIVO MIF PARA EL BLOQUE ROM. ....	160
FIGURA NO. 3.10.1: PROCESADOR AVANZADO. <sup>[16]</sup> .....	164
FIGURA NO. 3.10.2: INTERCONEXIÓN DE UN PROCESADOR AVANZADO. <sup>[16]</sup> .....	165
FIGURA NO. 3.10.3: CÓDIGO PARA PROCESADOR AVANZADO. ....	170
FIGURA NO. 3.10.4: REGISTROS. ....	171
FIGURA NO. 3.10.5: PROGRAMA PRINCIPAL. ....	172
FIGURA NO. 3.10.6: FORMATO ARCHIVO MIF. ....	172
FIGURA NO. 4.1: METODOLOGIA PARA EL DESARROLLO DE SOLUCIONES. ....	174
FIGURA NO. 4.2: DIAGRAMA GENERAL. ....	176

FIGURA NO. 4.3: CÓDIGO PRINCIPAL PARA EMPACADORA DE FRUTAS.-----	181
FIGURA NO. 4.4: MODULO DE RELÉS PARA IMPLEMENTACIÓN DE CIRCUITO DE FUERZA -----	182
FIGURA NO. 4.5: PINOUT DE MÓDULO DE RELÉS. -----	182
FIGURA NO. 4.6: CIRCUITO DE FUERZA -----	183
FIGURA NO. 4.7: CIRCUITO PARA BARRERAS LUMÍNICAS. -----	183

## Índice de tablas

TABLA NO. 2.1: PINES FPGA INVOLUCRADOS EN EL EJEMPLO. -----	53
TABLA NO. 3.1.1: TABLA DE VERDAD PARA ASIGNACIÓN 6. -----	95
TABLA NO. 3.2.1: TABLA DE VERDAD PARA ASIGNACIÓN 2. -----	100
TABLA NO. 3.4.1: ROTACIÓN DE UNA PALABRA EN 3 DISPLAY. -----	120
TABLA NO. 3.4.2: ROTACIÓN DE UNA PALABRA EN 6 DISPLAY. -----	120
TABLA NO. 3.7.1: CODIGOS ONE-HOT PARA LA FSM.-----	137
TABLA NO. 3.7.2: CODIGOS ONE-HOT MODIFICADOS PARA LA FSM.-----	140
TABLA NO. 3.7.3: CÓDIGOS BINARIOS PARA LA FSM. -----	140
TABLA NO. 3.9.1: INSTRUCCIONES REALIZADAS POR PROCESADOR. <sup>[15]</sup> -----	152
TABLA NO. 3.9.2: SEÑALES Y CICLOS NECESARIOS PARA COMPLETAR LAS INSTRUCCIONES. <sup>[15]</sup> -----	153
TABLA NO. 3.10.1: NUEVAS INSTRUCCIONES PARA EL PROCESADOR AVANZADO. <sup>[16]</sup> -----	162
TABLA NO. 4.1: VARIABLES DE PROCESO-----	177

## Índice de Ecuaciones

ECUACIÓN NO. 3.1: EXPRESIÓN BOOLEANA PARA MULTIPLEXOR DE 2 A 1-----	87
ECUACIÓN NO. 3.2: EXPRESIÓN EN VHDL PARA MULTIPLEXOR DE 2 A 1 -----	88
ECUACIÓN NO. 3.3: DECLARACIÓN DE UN PUERTO HEX. -----	91
ECUACIÓN NO. 3.4: MODELO PARA UN SUMADOR DE BITS-----	117
ECUACION NO. 4.1: VARIABLE PARA MEMORIA DE PROCESO-----	177
ECUACION NO. 4.2: VARIABLE DE CONTROL PARA BANDA DE CAJAS -----	177
ECUACION NO. 4.3: VARIABLE DE CONTROL PARA BANDA DE FRUTAS -----	177

## Introducción

Los dispositivos de lógica programable se han vuelto una necesidad común en la mayoría de áreas donde se requiere tener un control preciso de las variables que afectan un circuito eléctrico, esta necesidad no es ajena a los avances tecnológicos, es mas, con el enfoque correcto se puede observar que es una consecuencia lógica de los desarrollos que va experimentando la forma en la que se realizan los circuitos eléctricos de control.

Velocidad de respuesta, soporte de altas frecuencias, herramientas de procesamiento de datos, amplias capacidades de elementos lógicos y bajo consumo energético son algunas de las características más buscadas al momento de elegir un dispositivo programable que satisfaga las exigencias de una determinada tarea de control.

El siguiente documento muestra la investigación y resultados obtenidos relacionados al uso de tecnología FPGA para el diseño e implementación de sistemas digitales y de control automático, estos dispositivos son programados mediante lenguajes de modelado de circuitos, la metodología a seguir se fundamento en cuatro capítulos que se explican brevemente a continuación:

- Capitulo 1, Teoría General: en este apartado se aborda todo lo relacionado a los conocimientos generales acerca de la tecnología FPGA, de igual forma se describen algunas ventajas que implica usar este tipo de dispositivos, posterior a eso se hace una descripción de los elementos que componen el equipo a utilizar para realizar la investigación.
- Capitulo 2, Uso de Software: esta sección tiene una función tipo manual de usuario, en la cual se discute brevemente los lenguajes de modelado admitido por el equipo, luego se creó una guía práctica sobre el uso de software a utilizar, dentro de esta guía se contemplan todas las generalidades básicas del programa, así también se muestra el uso correcto de las herramientas que el software pone a disposición para los usuarios.
- Capitulo 3, Laboratorios: La mejor forma de aprender una técnica es practicando, es por eso que se presentan diez prácticas de laboratorio con los cuales es fácil comprender los principios básicos de las funciones de un FPGA, cada práctica está compuesto de 4 a 6 ítems, con relación directa, es decir que los conocimientos practicados serán correlacionados.
- Capitulo 4, Aplicación: Esta sección contienen el proceso seguido para la implementación de una aplicación sobre un proceso de control automático, la aplicación es de carácter didáctica para favorecer una secuencia en el aprendizaje, puesto que la intención final es lograr una cadena de conocimientos que conduzca a poder dar soluciones a problemas reales mediante la tecnología FPGA.

Cabe resaltar que esta tecnología no es nueva pero no es muy utilizada a nivel didáctico, incluso a nivel general son poco utilizados, a pesar de que los lenguajes de programación utilizados son estándares internacionales, de amplio uso y documentación.

El proposito de realizar esta investigación es dejar la base necesaria para comenzar a difundir estos dispositivos dentro del estudiantado y que sea un punto de atención cuando se quiera realizar un circuito programable o incluso al momento de montar un sistema embebido.

## **Objetivos**

### **Generales**

- Fundamentar los conocimientos básicos para el uso de dispositivos FPGA.
- Establecer las bases para el desarrollo e implementación de aplicaciones en el control de procesos industriales y funciones lógicas en general.

### **Específicos**

- Definir las metodologías de diseño disponibles para dispositivos FPGA.
- Describir de forma clara y concisa las diferencias que presentan los dispositivos FPGA según su fabricante.
- Presentar las consideraciones que se deben tomar para la selección de dispositivos FPGA.
- Definir una guía de usuario sobre el software a utilizar.
- Establecer guías prácticas que conduzcan a un mejor manejo de dispositivos FPGA.
- Describir ejemplos base de aplicaciones para el control de procesos e implementación de funciones.

# Capítulo 1: Teoría General

---

## 1.1 Preliminares

Un FPGA (Field Programmable Gate Array) es un dispositivo de silicio semiconductor con la capacidad de ser poder programar una operación deseada, su mapeo interno es similar a una matriz construida con elementos electrónicos interconectados entre si y cuya estructura es propia del fabricante, pueden estar compuestos por CLB (Configurable Logic Block) o por LAB (Logic Array Block), en esencia los dispositivos FPGA siguen una misma lógica no importando el fabricante, lo que puede cambiar son los elementos que componen cada uno de esos bloques.

Surgen como una evolución de los PLD (Programmable Logic Device). En 1984 Ross Freeman y Bernard Vonderschmitt, ambos ingenieros electricistas crearon el primer prototipo de FPGA para la empresa Xilinx de la cual son cofundadores, el primer objetivo de este dispositivo era aumentar significativamente su capacidad en relaciones a sus antecesores, reducir los tiempos de respuesta, bajar su consumo energético y brindar soluciones para problemas con alto nivel de complejidad, desde aquella época hasta el día de hoy la evolución de estos dispositivos ha sido gigantesca y su demanda va creciendo considerablemente en muchos campos donde la electrónica es un factor esencial.

Hoy en día existen muchas empresas fabricantes siendo las más destacadas: ALTERA, XILINX, ATMEL y LATICE, aunque también existen otros fabricantes como: MAZeT, WUNTRONIC, RFEL, GIGOPTIX y Microsemi. Todas estas empresas son fabricantes directos aunque también existen empresas partners que construyen kits de desarrollo basados en dispositivos FPGA.

La ventaja que presenta un FPGA es que puede ser reprogramado al menos unas 100,000 veces, existen FPGA no programables pero son más comunes las FPGA volátiles, este ultimo tipo requiere de dispositivos externos que mantengan su configuración como memorias EEPROM ó FLASH. A nivel de capacidad un FPGA puede ser comparado con un ASIC (Application-Specific Integrated Circuit) que tal como su nombre lo indica son circuitos integrados creados para una aplicación específica. Aunque su capacidad es similar son dos dispositivos con fines muy diferentes, un FPGA puede comportarse como un ASIC pero no lo contrario, si bien es cierto que su consumo energético es mayor, los últimos avances en FPGA van dirigidos a que sean dispositivos de bajo consumo comenzando por sus niveles de voltaje, el nivel máximo de voltaje para un FPGA es de 3.3 V.

Si bien los FPGA actuales ya son de muy bajo consumo energético y sus tiempos de respuesta rondan los nanosegundos, algunas compañías pensaron en llevar este dispositivo a otro nivel, en esencia un FPGA se comporta como un controlador, así que la evolución era agregar un procesador, y es así como hoy en día se pueden adquirir FPGA con sistemas HPS (Hard Processor System), dando origen a la generación SoC (System on Chip), ampliamente utilizada en sistemas embebidos.

En cuanto al método de programación, los dispositivos FPGA utilizan la interfaz JTAG para la prueba y descarga de datos, esta interfaz es de amplio uso entre las compañías de dispositivos electrónicos, desde 1990 es un estándar IEEE con número de referencia 1149.1-1990. En cuanto a los códigos de programación estos pueden ser creados con HDL (Hardware Description Language) siendo el más utilizado VHDL. Para programadores más familiarizados con C, existe la opción de Verilog que es más parecido a este lenguaje, ambos son normas IEEE con número de referencia 1076-1993 y 1364-2001 respectivamente, la suite de compilación y síntesis de los programas varían según el fabricante.

A pesar de que el costo de estos dispositivos a nivel educativo son relativamente altos comparados con otras plataformas para sistemas embebidos, la capacidad de un FPGA supera por mucho las expectativas, estos costos que para un estudiante pueden ser muy altos a las empresas interesadas en adquirir sistemas basados en FPGA les parecen muy bajos, el poder ajustar un circuito a la necesidad específica, poder hacer los diseños con lenguajes de descripción de hardware, darle mantenimiento o reprogramación a un dispositivo de forma remota y rápidamente, es lo que ha lanzado a la escena a estos dispositivos.

## 1.2 ¿Por qué un FPGA?

Esta pregunta es de rigor ya que existen muchas tecnologías que brindan la posibilidad de crear sistemas embebidos manteniendo una lógica programable, pero para dar una respuesta concisa se enumeran las siguientes 5 características que pueden brindar un panorama de las capacidades de un FPGA:

1. Rendimiento: Si bien es cierto que la primera crítica que se hace a los FPGA es que estos demandan un mayor tiempo de respuesta y un consumo mayor en relación a los ASIC, pero estas diferencias de tiempo han sido disminuidas hasta ser casi imperceptibles, ninguna tecnología que brinde soluciones por sistemas embebidos maneja los tiempos de respuesta que actualmente puede brindar un FPGA, BDTI (Berkely Design Technology Inc.) realizó una evaluación en la que se demostraba que un FPGA puede brindar mayor potencia en algunas aplicaciones, que si estas fueran realizadas por una DSP (Digital Signal Processor), esto es con relación al tiempo, ahora si analizamos la gran capacidad de bloques con los que cuenta un FPGA, el número de I/O disponibles para el usuario final, y quizá lo más importante el nivel de procesamiento de datos que puede ser de forma paralela otorgan un gran rendimiento a estos dispositivos, esa cualidad de procesar de forma concurrente los datos, mejora en gran medida la respuesta del dispositivo, agregando que la programación no solo puede ser de forma secuencial.
2. Diseño rápido y confiable: Cuando se experimenta con FPGA se tiene la ventaja de poder crear soluciones y probarlas inmediatamente, de esta forma se puede brindar respuesta a problemas sin tener que esperar el proceso que exige el crear un ASIC, incluso muchas compañías ya distribuyen kit de desarrollo en los cuales se integran periféricos específicos de mucha utilidad logrando que las pruebas de circuitos tarden unos pocos minutos luego de haber creado los códigos necesarios. Sumando a esta característica se puede

mencionar que cualquier cambio o actualización a cualquier sistema no requerirá cambiar el núcleo principal, basta con programar las nuevas funciones y el sistema estará listo, con relación de aprendizaje las herramientas de alto nivel permiten la abstracción de circuitos muy complejos de forma simple, algunas fabricantes incluso poseen una amplia biblioteca de IP Cores (núcleos de propiedad intelectual) los cuales son herramientas que ayudan a crear diseños muy complejos ya que concentran códigos para aplicaciones específicas de forma general dándole mucha ventaja al programador.

3. Precio: De todas las tecnologías para sistemas embebidos se podría decir que el FPGA es la más cara, pero con la dos características antes mencionadas se deja en claro que cada centavo lo vale, si se compara con un ASIC que es sin lugar a dudas su mayor competidor basta con decir que el precio de un ASIC se ve disminuido únicamente por la compra masiva del mismo, si por ejemplo una empresa pequeña desea una aplicación de tipo ASIC la fabricación del mismo será muy costoso, sin mencionar que con el pasar del tiempo puede que requiera modificaciones en un crecimiento eventual teniendo que recurrir a un nuevo dispositivo que cumpla con los requerimientos, en cambio para un FPGA es fácil adaptarse a cambios y esto puede ser aplicado tanto a pequeñas como a grandes empresas, de esta forma la inversión en un FPGA se ve muy justificada.
4. Fiabilidad: Cada diseño de un FPGA atraviesa un proceso de análisis y síntesis en el cual es muy difícil pasar por alto un error, además las herramientas de diseño brindan la oportunidad de visualizar el mapeo del dispositivo de forma que antes de descargarlo se pueden tomar consideraciones que mejoren el desempeño, suele suceder que los microprocesador por la forma secuencial de su programación basada en un núcleo central muchas veces entorpecen sus procesos al verse saturados, el paralelismo que brindan las FPGA permite que se cubran las tareas de forma eficiente.
5. Mantenimiento a largo plazo: Los dispositivos están pensados para que una vez programados e instalados no requieran mayores mantenimientos, y si fuesen necesarios estos se pueden realizar 'in situ' de forma tal que tome un par de minutos la operación, logrando eficiencia en el proceso.

### 1.3 FPGA y sus aplicaciones

Las aplicaciones en las cuales un FPGA puede brindar una solución son amplias, y actúan en muchos campos. Esto se debe al amplio manejo de datos y señales que brindan estos dispositivos. Dentro de las áreas aplicativas se encuentran:

1. Aerodefensa espacial: Radares y sistemas de guía pueden ser implementados, en muchos niveles: comercial, industrial y militar a través de tecnología FPGA.
2. Prototipos para ASIC: Esta tipo de aplicación es la más utilizada ya que a través de un FPGA se pueden crear rápidamente sistemas que cumplan las funciones propias de un ASIC con la ventaja de poder ser probados de forma inmediata lo que brinda un ventaja en la toma de decisiones, de esta forma se puede analizar los pro y contras que tendría la fabricación de un ASIC, abonado a una depuración más eficiente de estos sistemas.



3. Audio y video: el manejo de una amplia gama de señales permite que los FPGA brinden soluciones de bajo costo y con gran confiabilidad.
4. Automotriz: los sistemas de navegación y asistencia al conductor, monitoreo del correcto funcionamiento de un vehículo, sistemas de información y entretenimiento pueden ser creados con estos dispositivos.
5. Informática de alto rendimiento: con la integración de microprocesadores dentro del chip FPGA se puede lograr una aceleración de hardware, lo que conduce a un alto rendimiento en el procesamiento de datos y señales.
6. Centro de datos: diseñado para grandes anchos de banda. con un FPGA se pueden implementar servidores de baja latencia, sistemas de redes informáticas, y aplicaciones de control de datos.
7. Industrial: Con el paralelismo de procesos, la gran capacidad en funciones DSP y la flexibilidad para funcionar como un ASIC hacen que el FPGA se convierta en una solución alternativa para la automatización de procesos como los industriales.
8. Seguridad: ya se hablo del procesamiento de señales de audio y video, asi como la posibilidad de tener un microprocesador dedicado, si a esto se agrega sistemas biométricos y de control de datos, se tienen los elementos fundamentales para la implementación de sistemas de seguridad y vigilancia.
9. Área médica: La gran capacidad de procesamiento, visualización y muestreo de datos permite que se puedan crear aplicaciones para equipos de monitoreo, diagnostico y aplicación de terapias.
10. Broadcast: Se incluye el área de la transmisión y recepción de señales, brindando soluciones que puedan competir con equipos dedicados para estos fines. se vuelve un gran complemento cuando se quiere implementar una estación SDR (Software Defined Radio).
11. Aplicaciones inteligentes: En la búsqueda de la comodidad los sistemas inteligentes han tomado un gran auge, un FPGA puede adaptarse a las exigencias que demandan estos sistemas.
12. Equipos medidores y de pruebas: la gama de equipos que se pueden diseñar es muy amplia, pero como ejemplo se pueden mencionar: Osciloscopios, generadores de señal y analizadores, analizadores lógicos, multímetros, equipos de medicion para industriaautomotriz, probadores analógicos/RF, de señales mixtas, de memoria, etc.

## 1.4 Tecnologías constructivas

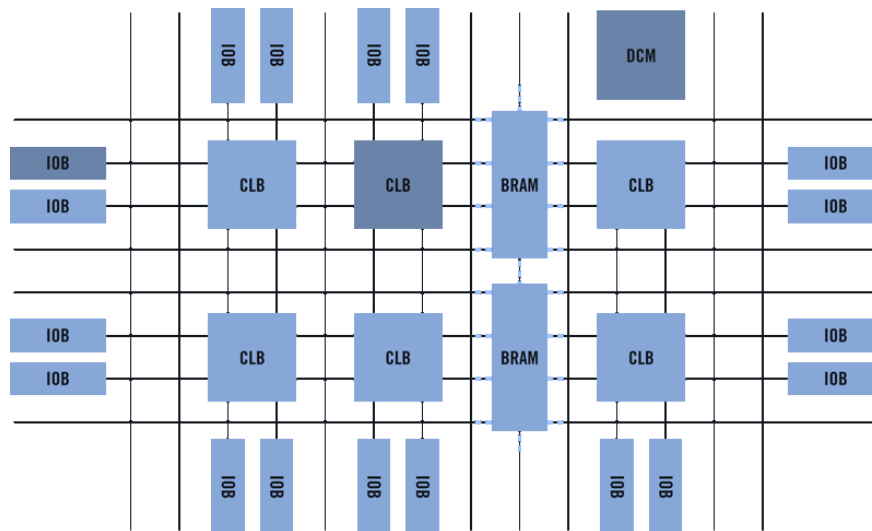
Se mencionó que según el fabricante así puede ser el modelo constructivo del FPGA, todos estos dispositivos mantienen una misma lógica: un arreglo matricial de bloques lógicos interconectados entre sí pero activados mediante programación.

ALTERA y XILINX son fabricantes líderes de FPGA, aunque no son los únicos, se discutirá la tecnología constructiva de los dispositivos de estas empresas ya que son los que disponen de productos enfocados no solo al área empresarial sino también el área educativa.

Por lo general al echar un vistazo a los FPGA por dentro dará la impresión que es parecido a las “matrioskas”, esas muñecas del folklore ruso, que al desglosar sus capas estaban constituidas por piezas cada vez más pequeñas en su interior, de igual forma sucede con los FPGA.

### 1.4.1 Xilinx

Esta empresa pionera de esta tecnología dispone de dispositivos cuya arquitectura está compuesta por 5 elementos. Un ejemplo de una arquitectura básica se muestra en la figura 1.1.



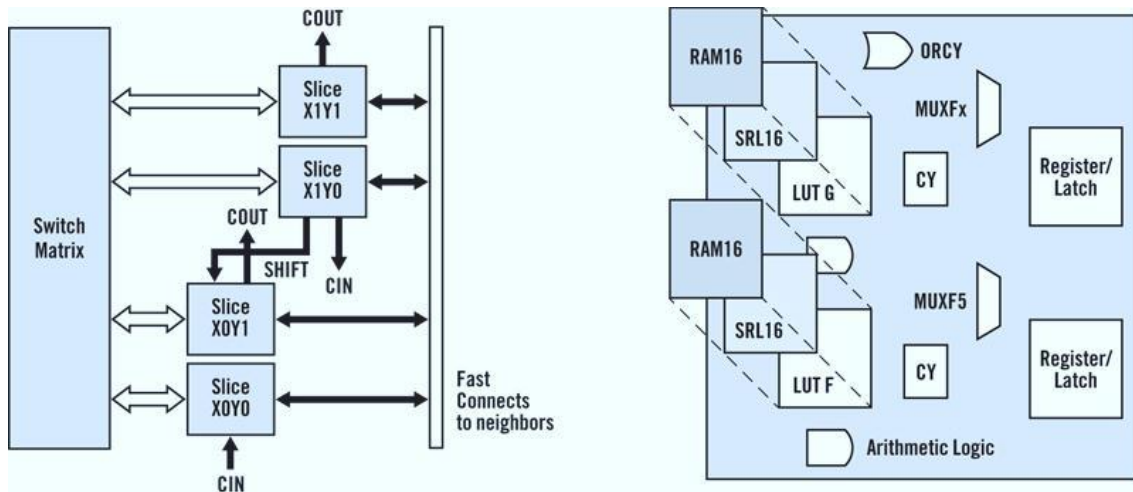
**Figura No. 1.1: Estructura básica de un FPGA Xilinx.**

#### 1.4.1.1 CLB

Un CLB (Configurable Logic Block) es la unidad básica de todo FPGA, el número y características de los mismos varían según la familia del dispositivo, en la figura 1.2 se muestra la estructura interna de un CLB, este elemento está constituido por una matriz de conmutación conectada a 4 sub unidades llamadas *slices*.

#### 1.4.1.2 Slices

Cada slice contiene elementos con la capacidad de manejar lógica combinacional, el elemento a destacar son las LUT (Look Up Table) que pueden ser empleados como Flip Flops, Latches, shift register o elementos de memoria RAM, dependiendo del número de entradas disponibles en la LUT así será el tipo de funciones que podrá desempeñar, dentro de los slices también se pueden encontrar elementos como multiplexores o registros dedicados.



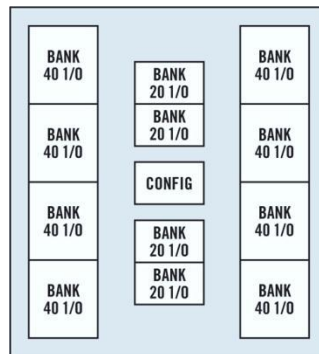
**Figura No. 1.2: Estructura interna de un CLB.**

#### 1.4.1.3 Líneas de conexión (interconnect)

Son las conexiones físicas entre los CLB y las demás unidades básicas, existen varios tipos de enrutamientos, como los que se hacen desde las CLB's hacia las I/O, así como también las de tipo global, para variables internas y de temporización. El enrutamiento está a cargo del software y por lo general el usuario no es capaz de verlo a menos que se indique lo contrario.

#### 1.4.1.4 IOB

Los IOB (Input/Output Blocks) constituyen la interfaz de conexión entre los pines externos y la circuitería interna, estos se encargan de organizar determinado número de pines en grupos independientes, con soporte para 12 tipos de I/O, algunas familias cuentan con un control digital de impedancia, el tamaño de los bloques está determinado por el dispositivo, la figura 1.3 muestra una estructura típica para los bancos I/O.



**Figura No. 1.3: Estructura de bancos IOB.**

#### 1.4.1.5 Memory RAM blocks

Son bloques de memoria con una capacidad de 36 Kbits por bloque, están disponibles en la mayoría de FPGA, según sea la familia así es el tamaño global de memoria RAM disponible en el

chip, los bloques son de doble puerto y disponen de señales propias para controlar la lectura y escritura.

#### **1.4.1.6 Complete clock management**

El uso de un gestor digital de reloj así como la función PLL (phased loop locked) provee una señal de reloj con una gran exactitud, gracias a la corrección de errores que se brinda a través de una etapa de realimentación, lo que brinda confiabilidad en circuitos donde el tiempo es fundamental.

Xilinx dispone de varias familias de dispositivos FPGA que se enlistan a continuación de menor a mayor capacidad:

1. Spartan.
2. Artix (normal scale y ultra scale).
3. Kintex (normal scale y ultra scale).
4. Virtex (normal scale y ultra scale).

Como se mencionó existen empresas asociadas que se encargan de crear kits de desarrollo para estos FPGA, según sea el chip FPGA y el número de periféricos así es el precio, la compañía National Instruments tiene sus propios kits potenciados por chips Xilinx con la variante de que pueden ser programados a través de los programas LABVIEW y SIMULINK.

### **1.4.2 Altera**

Esta compañía tiene su propia arquitectura, que varía en relación a sus familias, existen diferencias entre la gama media-baja y la gama alta, las unidades básicas se describen a continuación:

#### **1.4.2.1 Logic Array Block**

Un LAB (Logic Array Block) es la unidad fundamental de los FPGA Altera. Según la gama del dispositivo así es su modelo constructivo, estos bloques engloban las funciones de procesamiento que puede realizar el FPGA. Debido a que su contenido es variable a continuación se enumeran los elementos que lo componen en base a los dispositivos:

Para dispositivos de gama alta, un LAB contiene:

1. 10 ALM's (Adaptative Logic Module)
2. Cadenas de acarreo (carry Chains).
3. Cadenas aritméticas (arithmetic chains).
4. Cadenas de registros (register chains).
5. Líneas de interconexión (interconnects).
6. Señales de control.

Para dispositivos de gama media-baja un LAB contiene:

1. 16 LE's (Logic Elements).
2. Señales de control.
3. Cadenas de acarreo (carry chains).

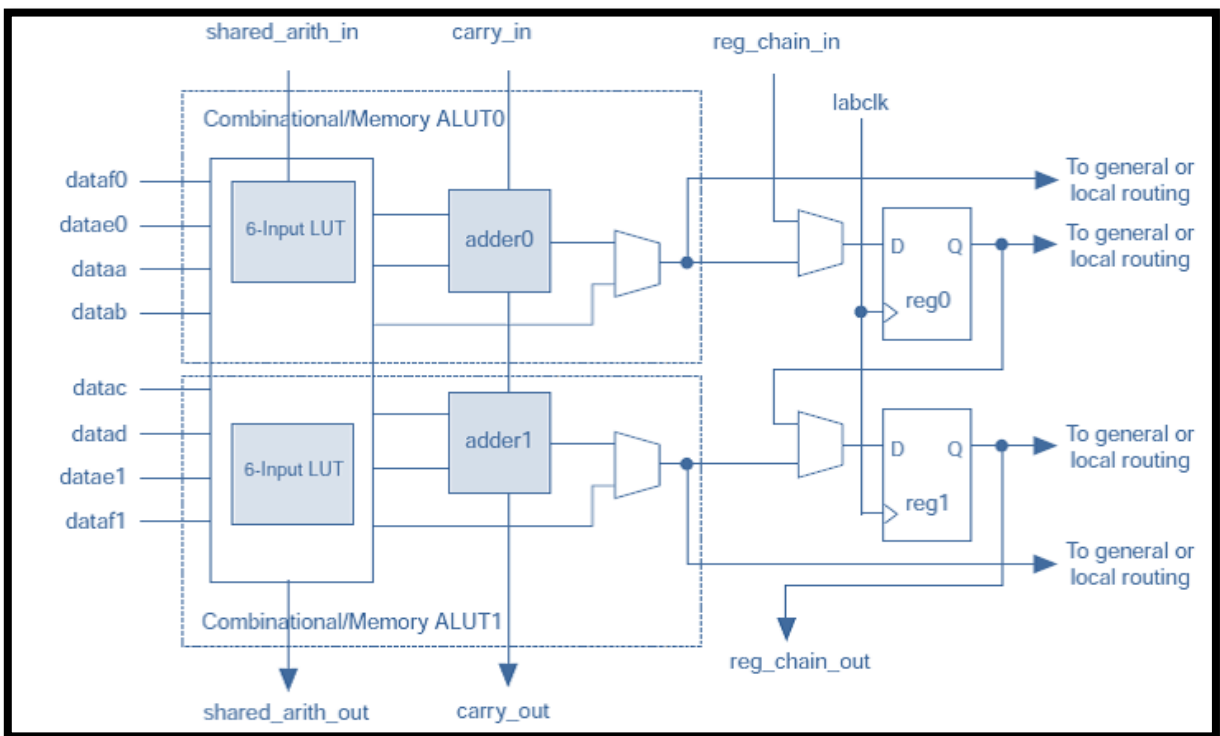
4. Cadenas de registro (register chains).
5. Líneas de interconexión (interconnects).

En los dispositivos de alta gama se pueden encontrar MLAB (Memory Logic Array Block) que añaden LUT's (Look Up Tables) basadas en SRAM. El número de MLAB estará determinado por la cantidad de SRAM con la que cuenta el dispositivo.

Ya que esas listas no brindan un panorama claro se describirá cada uno de los elementos de forma que sea fácil comprender sus funciones.

#### 1.4.2.2 Adaptive Logic Module

En la figura 1.4 se muestra un diagrama de bloques de alto nivel para las ALM:



**Figura No. 1.4: Diagrama de bloques para un ALM.<sup>[1]</sup>**

Los ALM (Adaptive Logic Module) son bloques lógicos encargados de realizar procesos en forma eficiente. Cada ALM contiene básicamente dos ALUT (Adaptive Look Up Table) con un máximo de ocho entrada encargadas de realizar funciones combinacionales, aritméticas y de desplazamiento de datos (shift register), los ALM también cuentan con una entrada aritmética y su respectiva salida, dos Full Adder dedicados, con su respectiva entrada y salida de acarreo, dos registros programables, con su respectiva entrada y salida de registro, y una entrada de señal de reloj. Estos módulos son capaces de realizar dos funciones a la vez, cada ALUT puede ejecutar una función de máximo 6 entradas, son totalmente compatibles para interconexiones con LUT's normales, los ALM son capaces de interconectarse de forma local, por fila y columnas y por vínculo directo.

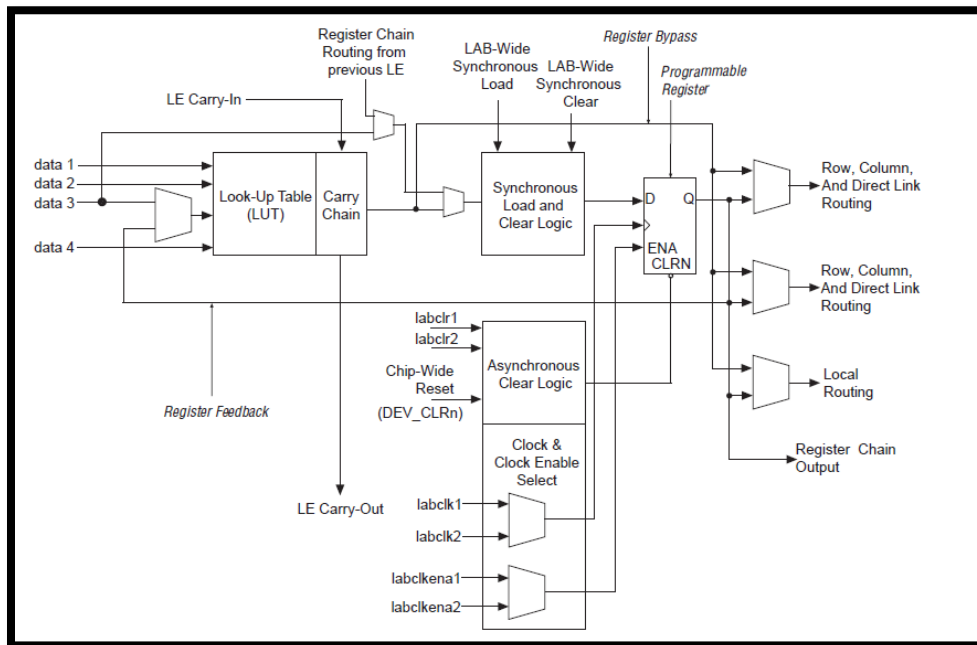
La interconexión de vínculo directo se refiere a la conexión que existe entre dos ALM que no están dispuestas de forma adyacente, esto lo logra el compilador a través de su herramienta Fitter, lograr este tipo de interconexión demanda un poco más al compilador y al dispositivo, pero en muchas ocasiones será algo de rigor y no será algo evitable.

### 1.4.2.3 Logic Element

Un LE (Logic Elements) es la estructura más pequeña dentro de un FPGA, está formado por una LUT de 4 entradas, un registro programable, entrada y salida de acarreo, entrada y salida de registro y entradas de control. Al igual que sucede con los ALM, los LE's se pueden interconectar entre sí de forma local, por filas, columnas o vínculo directo.

Previamente se explica la interconexión de vínculo directo, este tipo de conexión es aplicable a los LE, muchas características de conexión se mantienen desde las unidades más pequeñas hasta las de mayor tamaño.

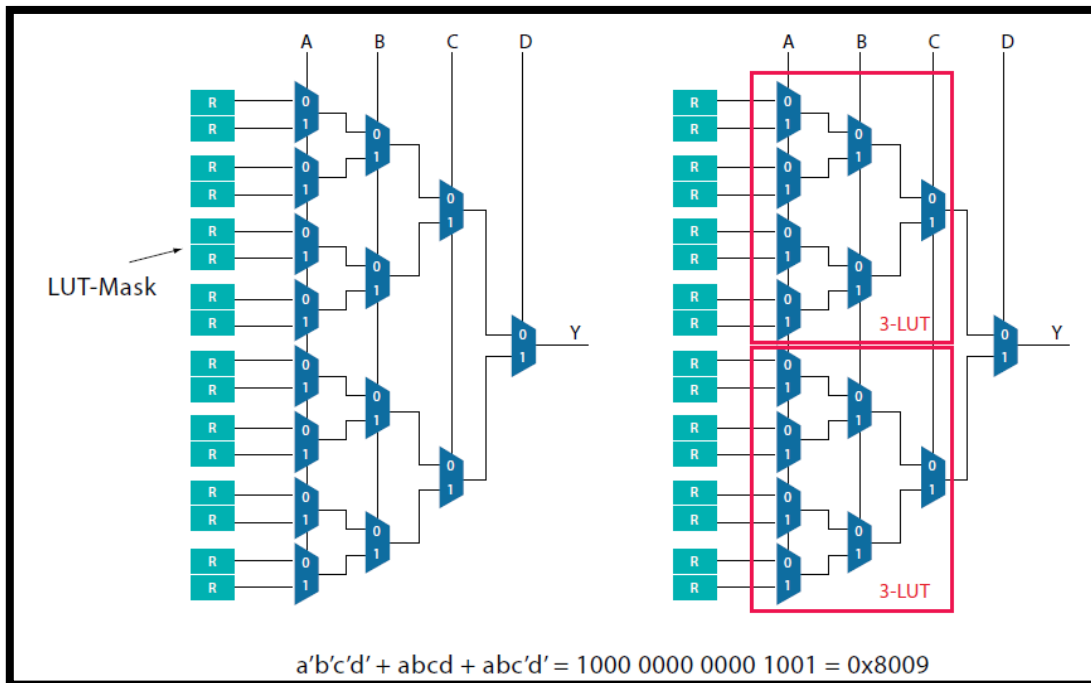
La figura 1.5 muestra un diagrama de bloques que describe un LE:



**Figura No. 1.5: Diagrama de bloques para un LE.[2]**

### 1.4.2.4 Look Up Table

Un Look Up Table es un tipo de construcción basada en vectores de datos, de tal forma que sus entradas se convierten en entradas de selección para un arreglo de multiplexor que a su salida provocan una función combinacional. Las ALUT (Adaptative Look Up Table) van más allá pudiendo realizar funciones aritméticas y desplazamiento de registros, en la figura 1.6 se muestra la estructura básica de una LUT:



**Figura No. 1.6: Estructura de una LUT.<sup>[3]</sup>**

#### 1.4.2.5 Registros programables

Son elementos que pueden ser configurados para que trabajen como un Flip Flop tipo JK, RS, D o T, la señal de reloj es controlada por un reloj global a menos que se indique lo contrario, si se requiere una configuración asíncrona se puede asignar una entrada que controle dicha operación.

#### 1.4.2.6 Cadenas de registro y acarreo

Las cadenas de registro llevan datos entre otros LE's y LAB's de esta forma se pueden lograr conexiones cascada entre registros, así las LUT se pueden encargar del proceso combinacional mientras que en los registros se puede trabajar el desplazamiento de datos.

Las cadenas de acarreo son líneas de interconexión que se encargan de llevar datos entre LE's según los requieran algunos procesos aritméticos.

#### 1.4.2.7 Señales de control

Cada LAB puede manejar alrededor de 10 señales de control que pueden ser: 2-3 relojes según la gama del dispositivo, 2-3 señales enable para relojes, dos reset asíncronos, un reset síncrono y un enable de carga de datos.

#### 1.4.2.8 Bloques I/O

Constituyen la interfaz entre los pines externos y los elementos internos, pueden estar conectados a las filas o columnas de la matriz general, los bloques tienen un número preestablecido de pines, y son capaces de manejar hasta 8 tipos de señales diferentes.

#### 1.4.2.9 Bloques de memoria

Son bloques de controlan memorias tipo RAM y ROM a nivel de puertos o memoria interna del CHIP, con la memoria se pueden realizar funciones tales como: buffers FIFO (First In, First Out), shift register y MLAB (Memory Logic Array Block).

#### 1.4.2.10 Multiplicadores embebidos

Los chips FPGA están en capacidad de poder controlar multiplicadores externos que pueden ser de gran utilidad para funciones DSP, acumuladores o operaciones aritméticas.

#### 1.4.2.11 Transceptores de alta velocidad

Un transceptor es un dispositivo con la capacidad de recibir y transmitir una señal. Dependiendo de la familia este modulo puede estar incluido dentro del chip FPGA.

#### 1.4.2.12 Sincronización

Las señales de reloj se manejan con pines y bloques dedicados para su procesamiento, el hardware cuenta con la estructura necesaria para poder implementar instancias de control como PLL (Phased Loop Locked) y DLL (Delay Loop Locked).

Como se explicaba conforme sea la gama del dispositivo así será los bloques que este contenga. A continuación se enlistan las familias disponibles actualmente para chips FPGA de ALTERA:

1. MAX 10 series.
2. Cyclone Series (normal y SoC).
3. Arria Series (normal y SoC).
4. Stratix Series (normal y SoC).

Las versiones SoC (System on Chip) son FPGA que tienen encapsulado un microprocesador y que funciona de forma independiente o conjunta con el FPGA, están capacitados para soportar muchos sistema operativo, por lo general son procesadores ARM Cortex.

## 1.5 Altera DE1 SoC

El Kit de desarrollo DE1 SoC fabricado por Terasic y potenciado por ALTERA Cyclone V SoC es una plataforma de diseño robusta que combina la tecnología de un procesador de doble nucleo ARM Cortex A9 con la flexibilidad de los dispositivos programables FPGA SoC. Esta herramienta permite tener todas las ventajas que brindan los FPGA sumado a un procesador ARM que trabaja en modo HPS (Hard Processor System), capaz de controlar dispositivos de audio, video, almacenamiento, ethernet, asi como también dispositivos para trabajar todos los aspectos básicos de lógica digital, lo que convierte a esta tarjeta en un modulo ideal para el aprendizaje sobre tecnología FPGA.

Altera University Program provee documentación, tutoriales y laboratorios prácticos para aprender todo lo referente al diseño e implementación de circuitos FPGA a través de esta tarjeta, pero su funcionalidad no se ve limitada al programa de aprendizaje de Altera ya que una vez se tienen los conceptos claros las aplicaciones antes discutidas pueden ser implementadas con esta tarjeta.



La tarjeta trabaja con la suite de programación Quartus II que dispone de una versión web totalmente gratuita y con la cual se pueden completar perfectamente los aspectos básicos de diseño para FPGA, la tarjeta puede ser programa con VHDL, Verilog e IP cores, la suite trabaja perfectamente en sistemas operativos de 64 bits (Windows o Linux) y puede ser descargada desde el sitio web del fabricante.



**Figura No. 1.7: Altera DE1-SoC.<sup>[4]</sup>**

### 1.5.1 Especificaciones

La tarjeta DE1-SoC dispone de los componentes necesarios para realizar todos los laboratorios y tutoriales disponibles en el programa universitario de ALTERA, desde el sitio web de TERIC se puede descargar el CD correspondiente a la tarjeta.

El CD contiene los manuales de funcionamiento, demostraciones y hojas de datos de cada uno de los componentes incluidos.

La disposición final de los elementos incluidos se puede observar en la figura 1.8.

El diagrama de bloques de la figura 1.9 ilustra la conexión interna los elementos, cada uno está conectado directamente al chip FPGA SoC de forma que el usuario tenga la disponibilidad de cada uno de los elementos de forma independiente. El número que reflejan las flechas corresponden a los pines del FPGA que están conectados a ese dispositivo en específico.

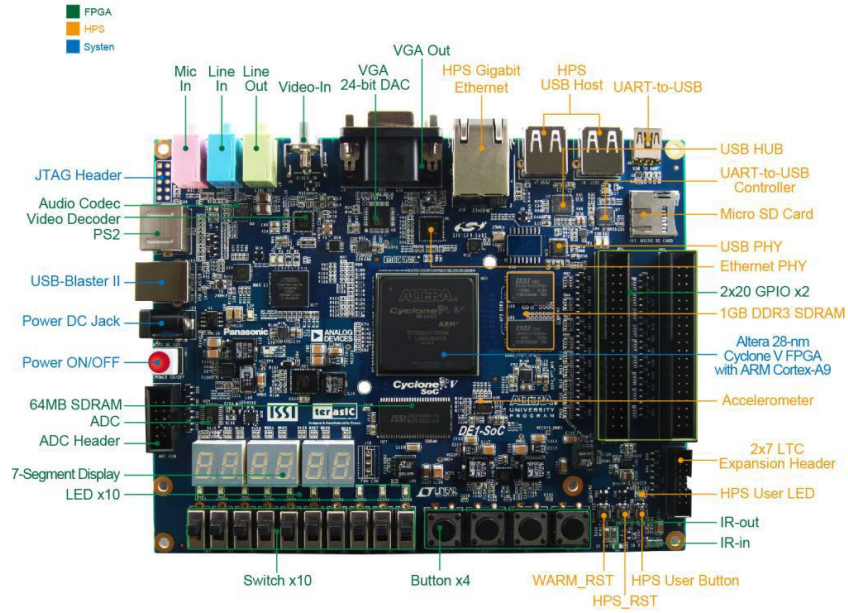


Figura No. 1.8: Mapa de elementos de la tarjeta DE1-SoC.<sup>[5]</sup>

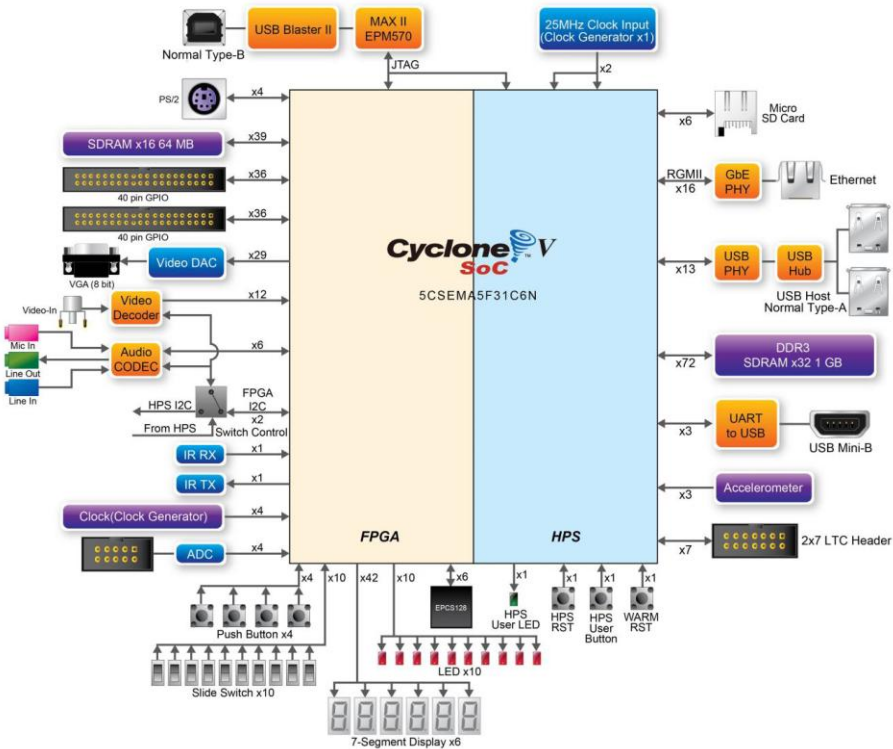
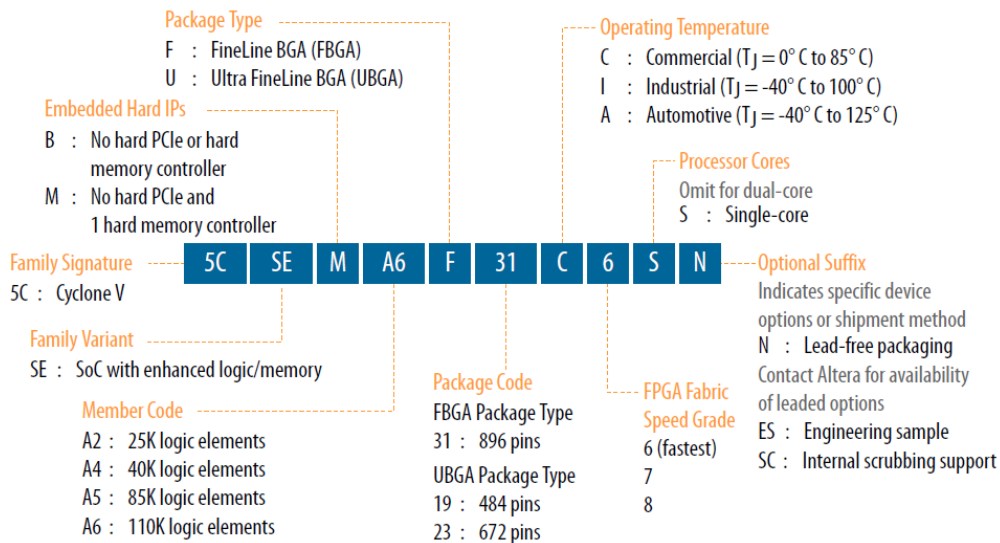


Figura No. 1.9: Diagrama de bloques de la tarjeta DE1-SoC.<sup>[5]</sup>

A continuación se enlistan las especificaciones técnicas de cada uno de los componentes de la tarjeta así como una breve descripción de su función:

### 1.5.1.1 FPGA

- Chip FPGA Cyclone V SoC 5CSEMA5F31C6: es un chip de la familia Cyclone V SE, con modulo SoC, modulo para expansión de memoria, sin modulo PCI, cuenta con 85K LE, tipo de empaquetamiento FBGA, dispone de 896 pines, trabaja en rangos de temperatura comercial (0 °C a 85 °C) y posee el máximo grado de velocidad.
- Procesador Dual Core ARM Cortex A9: 800 MHz, doble nucleo, arquitectura ARMv7-A, lanzado en el año 2008, compatible con OpenCL, funciona en modo HSP.
- 4,450 Kbits de memoria embebida.
- 6 Módulos PLL que cuentan con compensación de retrasos, síntesis de precisión para la señal de reloj y pueden trabajar en modo integral o fraccionario.
- 4 módulos DLL (Delay Locked-Loop).
- 87 bloques de tipo DSP.
- 2 controladores de memoria física.
- Su estructura constructiva está basada en ALM's de 8 entradas.



**Figura No. 1.10: Nomenclatura para dispositivos de la familia Cyclone V SE.<sup>[6]</sup>**

### 1.5.1.2 Configuración y Depuración

- Dispositivos EPCS128: Memoria Flash de 32 Mb, con interfaz de periférico serial (SPI), módulos I/O, capacidad de retención de 20 años, puede trabajar como DDR, buffer de 256 bytes, se utiliza para guardar los programas de configuración ya que el FPGA es volátil, el chip está configurado para extraer los datos de la memoria flash al iniciar, de esta forma cualquier programa guardado dentro de la misma se cargará automáticamente al FPGA al encender la tableta.
- Programador USB-BLASTER II incluido sobre la placa, con conector USB de tipo B.

### 1.5.1.3 Memoria

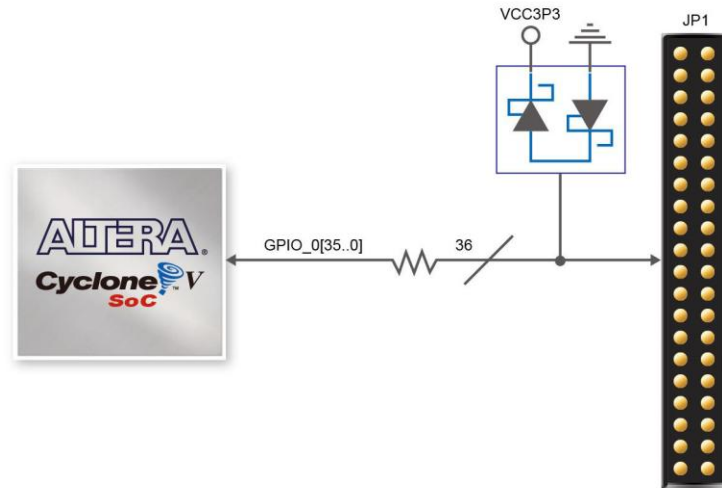
- Memoria SDRAM externa para chip FPGA: modelo IS4245R86400D, capacidad de 64 Mb, con voltaje de operación a 3.3V, el modulo puede ser utilizado para guardar de forma volátil los programas o para las funciones descritas en el sección anterior.
- Memoria DDR3 externa para procesador HPS: modelo IS43/46TR85120A, capacidad de 1 Gb, de uso exclusivo para el microprocesador, funciona de forma similar a una memoria RAM en una PC estándar.
- Modulo de expansión de memoria de almacenamiento SD: se recomienda usar memorias de categoría 10, y un valor minimo de 8 Gb, esta memoria será la encargada de bootear el sistema operativo para que funcione el microprocesador.

### 1.5.1.4 Comunicaciones

- 2 puertos USB 2.0 modelo USB330, interfaz de comunicación ULPI, conector tipo A, diseñados para uso general, cualquier dispositivo conectado a ellos necesitara el driver necesario.
- USB a UART, conector tipo mini B, modelo número FT232R UART IC, los dispositivos UART son aquellos que utilizan un protocolo serial denominado Universal Asynchronous Receiver-Transmitter para la transmisión y recepción de datos.
- Transceptor Ethernet 10/100/1000Mbps que cumple con la norma IEEE 802.3 con soporte RGMII (Reduced Gigabit Media Independent Interface) es cual es un estándar que tiene como propósito reducir el número de pines para establecer la comunicación entre la MAC (Media Access Control) y el PHY (Physical Layer Protocol).
- Puerto PS2 para mouse o teclado estándar.
- Trasmisor\Receptor infrarrojo.
- Multiplexor I<sup>2</sup>C, es un bus de comunicaciones de tipo serial con una velocidad de 100Kbps.

### 1.5.1.5 Conectores

- 2 conectores header de 40 pines, cada pin representa un GPIO (input/output de propósito general), el nivel de voltaje en estos pines es de 3.3 V cuando están activos, en cada conector se encuentran disponibles 36 pines de datos, 2 pines fijos GND, 1 pin +5 V y 1 pin +3.3 V, son pines con entrada protegida por si hay sobre tensión, cada pin puede manejar un máximo de 1.5 mA.
- Los pines de propósito general tienen una capacidad de transferencia de datos de 875 Mbps LVDS para la transmisión y 840 Mbps LVDS para la recepción.
- 1 conector header de 10 pines exclusivo para conectar ADC.
- 1 conector LTC.



**Figura No. 1.11: Diagrama de conexión de pines GPIO.<sup>[5]</sup>**

#### 1.5.1.6 Video

- ADC de 10 bits modelo ADV7123 dedicado a la transmisión de señales de video en formato VGA.
- Video decodificado de 10 bits modelo ADV7180 para la recepción de señales de video en formato NTSC/PAL/SECAM.

#### 1.5.1.7 Audio

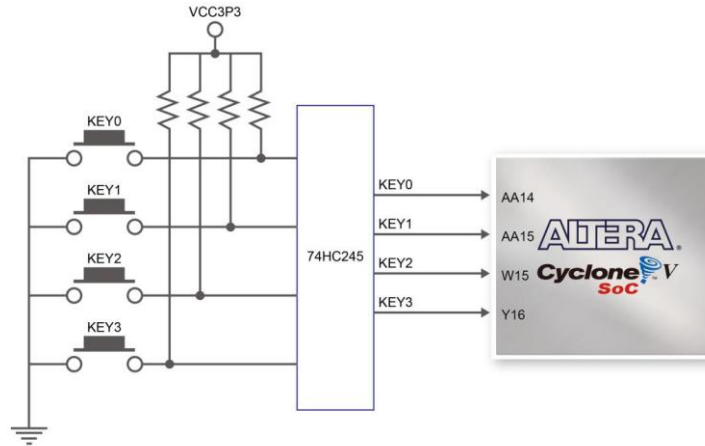
- Decodificador de 24 bits modelo WM8731, con puertos Line in, Line out y headphones jack, utiliza el driver realtek '94 para el controlar el dispositivo.

#### 1.5.1.8 ADC

- 12 bits de resolución, 8 canales de entrada, con un rango de 1 millón de muestras por segundo, los rango de voltaje para la entrada analógica son 0-2.5 V ó 0-5 V, este rango es definido por el usuario mediante un registro.

#### 1.5.1.9 Elementos de lógica digital

- 4 botones pulsadores, en estado de reposo su salida es '1' lógico, mientras que al ser presionado brindan un '0' lógico, su salida está conectada directamente a un dispositivo 74HC245 que actúa como buffer no inversor.
- 6 display de 7 segmentos ánodo común, se necesita un '0' logico para encender cada segmento.
- 10 interruptores de doble estado.
- 11 led (10xFPGA y 1xHPS)
- 2 botones de reset HPS



**Figura No. 1.12: Diagrama de conexión para botones pulsadores.<sup>[5]</sup>**

### 1.5.1.10 Sensores

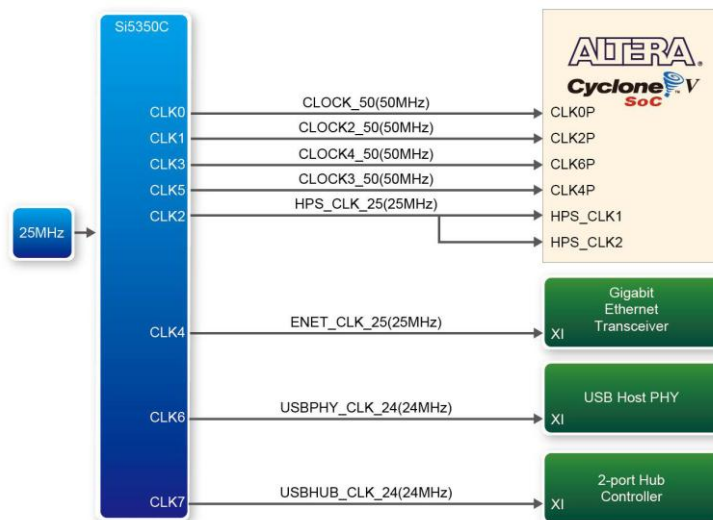
- Acelerómetro de 3 ejes (X,Y,Z) con resolución de 10 bits, únicamente disponible a través de HPS.

### 1.5.1.11 Fuente

- Fuente de voltaje 12V@5A

### 1.5.1.12 Circuitería de reloj

- 4 relojes de 50 MHz de uso exclusivo para FPGA.
- 2 relojes de 25 MHz (1 para HPS y 1 para Ethernet Transceiver).
- 1 reloj de 24 MHz para USB controller (este reloj no está disponible para el usuario).



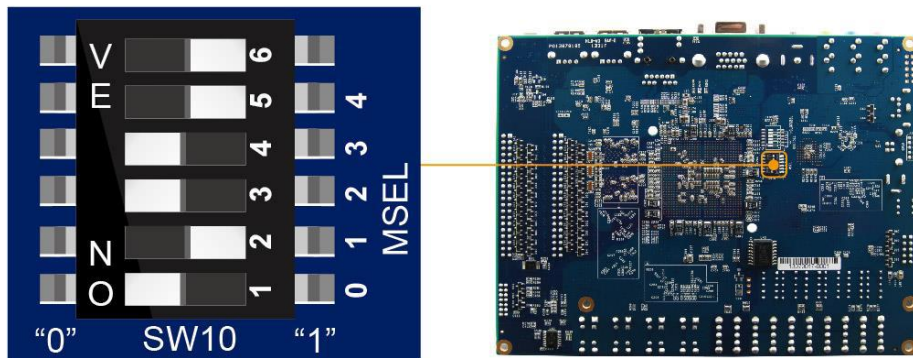
**Figura No. 1.13: Diagrama de bloques para la circuitería de reloj.<sup>[5]</sup>**

### 1.5.2 Configuración de FPGA

La tarjeta DE1-SoC puede ser configurada para modo EPCS o HPS, esto se logra con un DIPSWITCH de 6 bits situado en la parte posterior de la tarjeta, la figura 1.14 muestra la ubicación física dentro de la tarjeta.

La diferencia fundamental entre los modos es la integración de sus componentes ya que el modo EPCS solo utiliza el FPGA, mientras que el modo HPS incluye el procesador ARM al trabajo del FPGA. Algunos periféricos incluidos en la tarjeta solo pueden ser operados en el modo HPS.

Cada modo posee sus ventajas y desventajas propias del desarrollo de circuitos específicos pero en forma general, el modo EPCS permite operar el FPGA de forma volátil y no volátil, la ejecución de programas es más rápida y no está sujeto un sistema operativo, en cambio el modo HPS le provee potencia extra a la tarjeta, y activa todos los periféricos disponibles, con la limitante de estar sujeto a un sistema operativo, las FPGA manejan varios ecosistemas como por ejemplo Linux y android, este sistema se encarga de controlar los procesos internos.



**Figura No. 1.14: DIPSWITCH de configuración de tarjeta DE1-SoC.[5]**

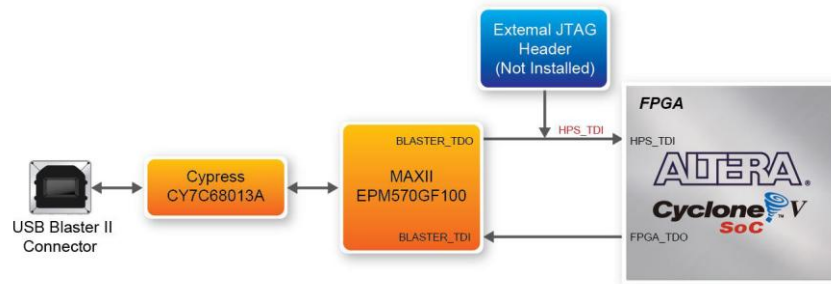
Por defecto la tarjeta viene configurada para funcionar en modo AS (Active Serial) de tal forma que al energizarse los datos contenidos en la memoria EPCS sean descargados al chip FPGA, esta configuración corresponde a la configuración  $MSEL[4:0] = "10010"$ , si el usuario necesita reconfigurar el FPGA para que funcione mediante una herramienta de software (Linux) la configuración debe ser  $MSEL[4:0] = "01010"$ , existe un tercer modo de configuración utilizado para cargar Linux LDXE Desktop desde una memoria SD, para acceder a este, la combinación debe ser  $MSEL[4:0] = "00000"$ .

### 1.5.3 Métodos de programación

El FPGA puede ser programado por dos métodos diferentes:

1. Programación por JTAG: Este método emplea el protocolo IEEE JTAG (Joint Test Action Group), los bits son descargados directamente hacia el chip FPGA, los datos de programación estarán presentes en el chip siempre y cuando este se mantenga

energizado, recordando que el FPGA en sí mismo es un dispositivo volátil, este método de programación es ideal para la prueba de códigos ya que la elaboración de un archivo \*.sof no requiere un proceso extra mas allá de una compilación que ejecuta el mismo software.



**Figura No. 1.15: Diagrama de bloques para programación JTAG.<sup>[5]</sup>**

2. Programación por Active Serial: Este método emplea archivos de extensión .JIC los cuales son configurados en la memoria EPCS256 de forma que la programación sea permanente para el chip FPGA, los archivos requeridos para la configuración no son creados por defecto, requieren un proceso extra que será discutido en el capítulo siguiente.



**Figura No. 1.16: Diagrama de bloques para programación AS.<sup>[5]</sup>**



# Capítulo 2: Software

---

## 2.1 Introducción

El software es la parte complementaria al hardware para los dispositivos programables, el software consta de dos partes básicas: el programa principal con sus herramientas y el lenguaje de programación.

- Programa principal: El fabricante de la tarjeta DE1-SoC pone a disposición el software Quartus II, este programa cuenta con la capacidad de hacer análisis, síntesis, compilación y simulación de circuitos lógicos para dispositivos CPLD y FPGA, el software puede ser descargado desde la página de ALTERA.
- Lenguajes de programación: los lenguajes HDL son por excelencia los encargados para la programación y configuración de dispositivos FPGA, los más populares son Verilog y VHDL, ambos por ser estándares IEEE no serán detalladamente explicados en esta investigación, si se requieren referencias lo recomendable es leer las normas IEEE 1364-2001 (Verilog) y IEEE 1789-1993 (VHDL), además se cuenta con un documento propio de la EIE-UES titulado “Lenguajes de descripción de hardware”, todos estos documentos brindan la información necesaria para programar los circuitos requeridos.

Existen dos tipos de licencia para Quartus II, la versión **suscription edition** tiene un costo, esta edición está destinada para las empresas que realizan proyectos de gran magnitud, también existe la versión **web edition** que es totalmente gratuita y no requiere una licencia especial, en las versiones anteriores a la 11.0 requería la descarga de una licencia a través de un formulario web pero eso fue suprimido para las versiones posteriores.

Por motivos de carácter económico se optó por utilizar la versión web edition con todas las limitantes que presenta por su carácter gratuito. Un punto a resaltar es que esta versión no es un demo, no tiene fecha de vencimiento que obligue a comprar la versión completa, simplemente es una versión con límites de operación, además es totalmente compatible con **ALTERA UNIVERSITY PROGRAM** que es la base de aprendizaje para los dispositivos FPGA académicos.

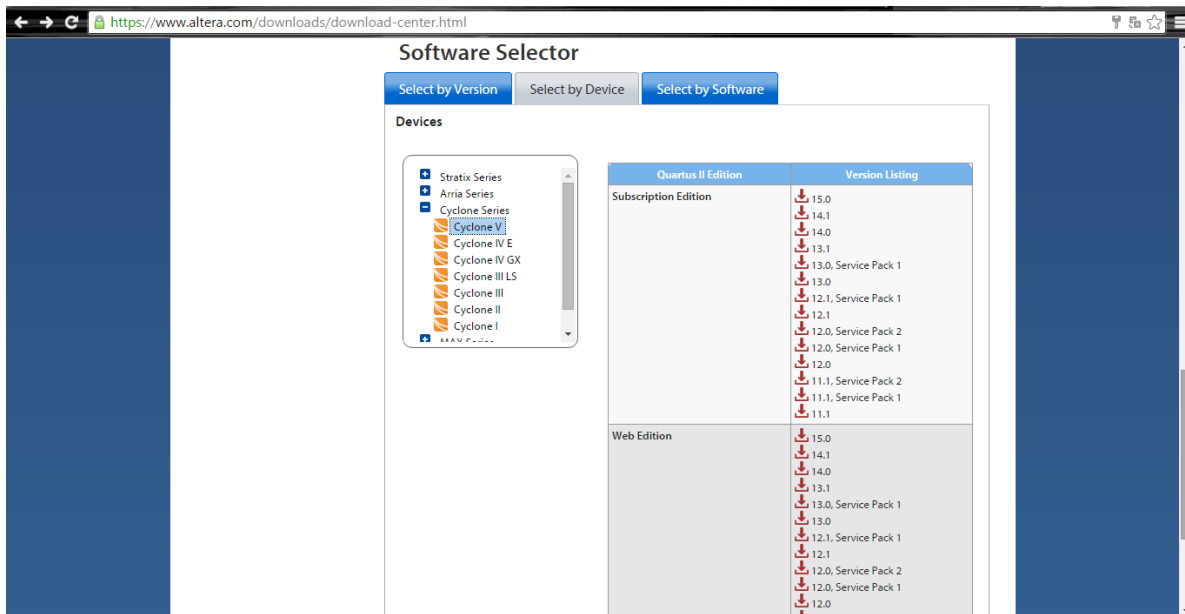
## 2.2 Descarga del software

ALTERA en su página web posee un centro de manejo de descargas en el cual se encuentran las versiones disponibles de su programa además de herramientas alternativas que actúan como componentes de Quartus II, para acceder a este web se debe seguir el link: <https://www.altera.com/downloads/download-center.html>[en línea] [última consulta 24/02/2016].

Para hacer cualquier descarga de software a través del sitio oficial de ALTERA se requiere llenar un formulario de registro en el cual se crea una cuenta de usuario, algo muy recomendable, ya que

esta cuenta también da acceso a los cursos y tutoriales disponibles en el training center. El link de registro es el siguiente: <https://www.altera.com/mal-all/mal-signin.html>[en línea] [ultima consulta 24/02/2016]

Actualmente la última versión de Quartus II es la 15.0. A pesar de que se recomienda estar actualizado, el método más eficaz para saber que versión es conveniente se utiliza el selector por dispositivo que posee la web de descarga. En la figura 2.1 se muestra el selector por dispositivo dentro de la pagina web.

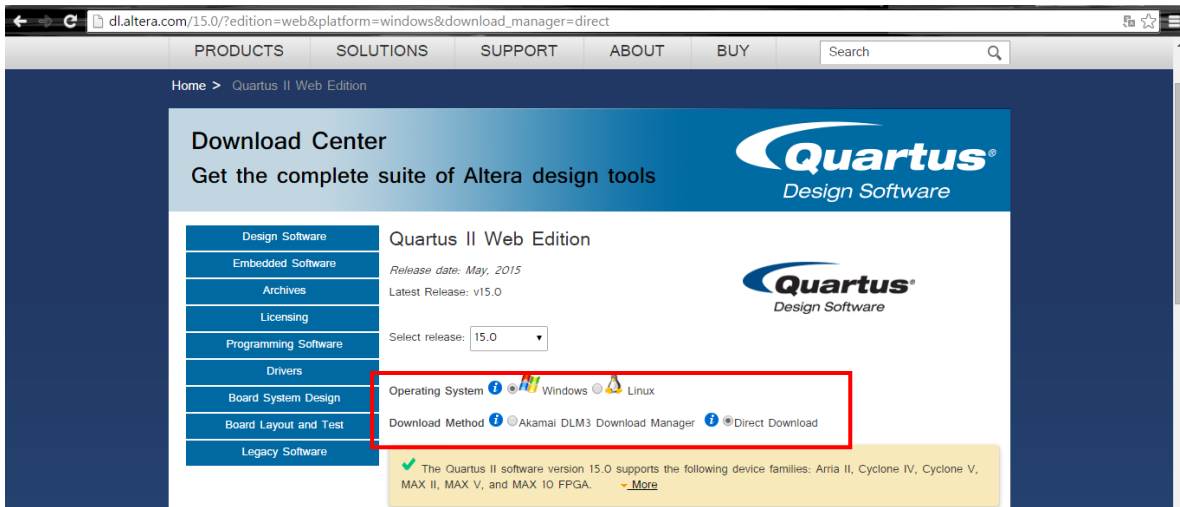


**Figura No.2.1: Selector de software por dispositivo.**

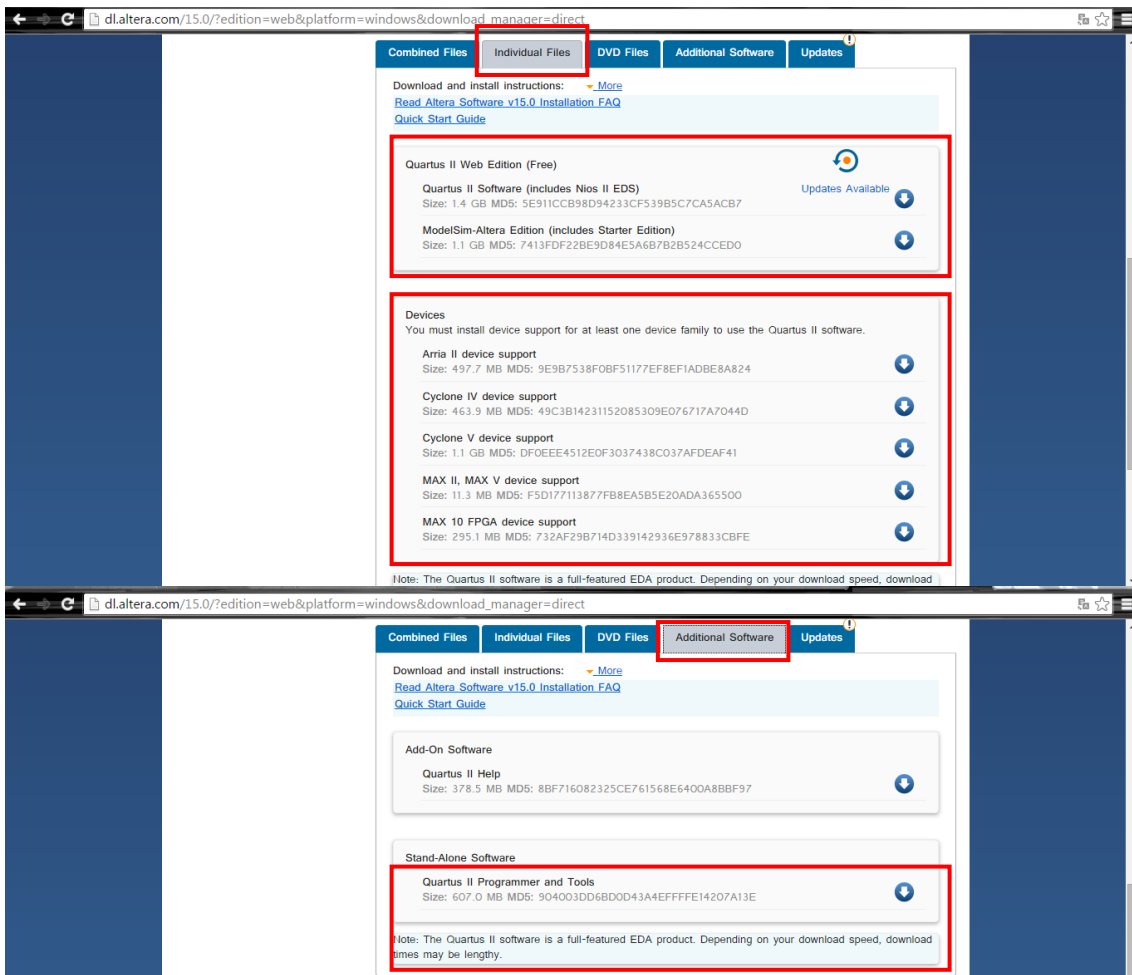
Para el caso de la tarjeta DE1-SoC que posee un FPGA de la familia Cyclone V la versión web disponible más avanzada es la 15.0, así que se procede a descargar el software correspondiente a esa versión.

En la ventana mostrada en la figura 2.2, se debe seleccionar el sistema operativo en el cual se instalará Quartus II web edition 15.0, y se recomienda hacer la descarga por vínculo directo, sin necesidad de terceros programas que administren el proceso.

Siempre en la misma ventana se debe seleccionar si la descarga de archivos será de forma individual o si se realizara por CD de instalación, queda a discreción del usuario como realizar la descarga, si se realiza la descarga de archivos individuales la figura 2.3 muestra los archivos requeridos.



**Figura No. 2.2: Selección de sistema operativo para el software.**



**Figura No. 2.3: Archivos necesarios para la instalación.**

Una vez descargados todos los archivos se procede a la instalación.

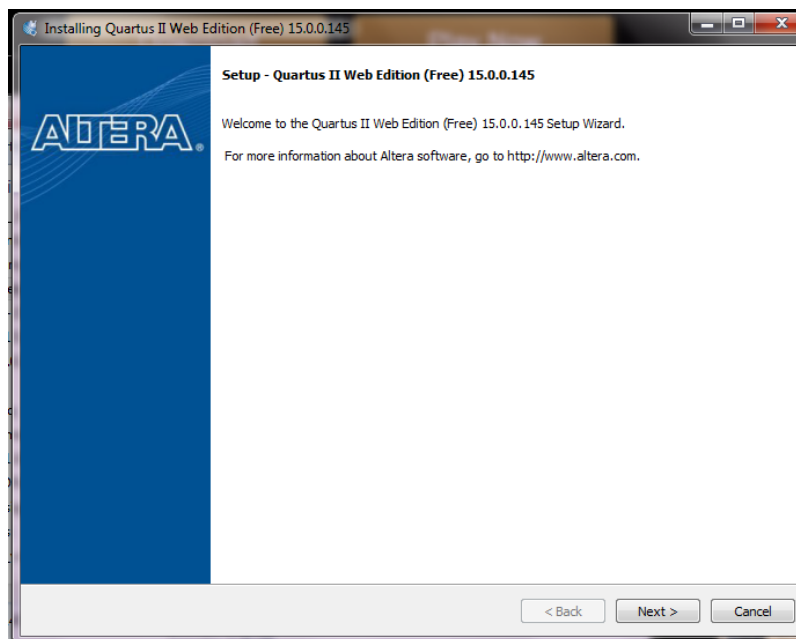
## 2.3 Instalación del software

La página de ALTERA no especifica requerimientos para que el software opere de forma plena en la pc pero basado en la experiencia se puede hacer una estimación de los recursos que debe tener una PC para operar de forma estable y se describen a continuación:

- Windows Vista, 7, 8 (64 bits de preferencia).
- Procesador Dual Core 2.17 GHz (como mínimo).
- 4 Gb de memoria RAM (como mínimo).
- 15 Gb libres de disco duro.
- No requiere especificaciones extra para la tarjeta de video.

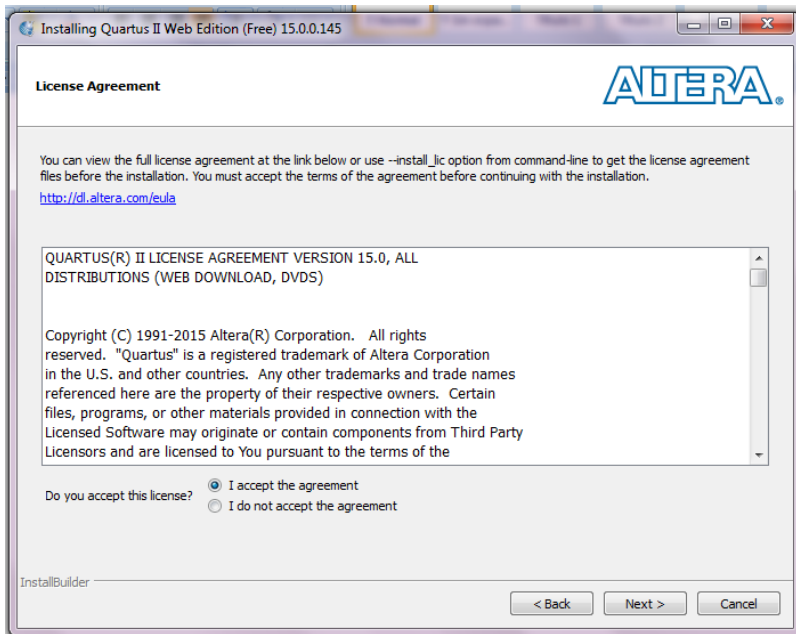
En la compilación y simulación de programas la memoria RAM es altamente demandada por lo cual se recomienda tener un valor alto de este recurso, de esta forma se reduce el tiempo de ejecución y el desgaste de la PC.

El primer paso es ejecutar el archivo “**QuartusSetupWeb-15.0.0.145-windows**”, esto desplegara el asistente de instalación tal como se muestra en la figura 2.4.



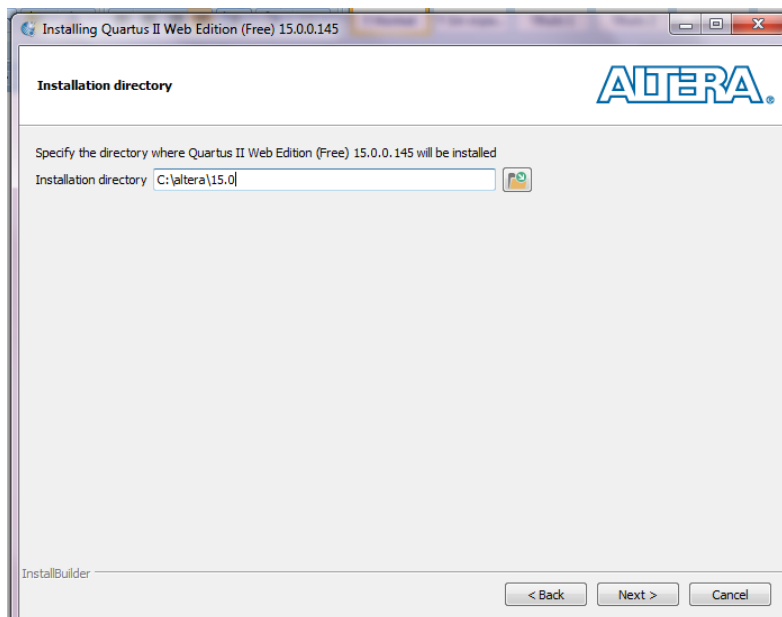
**Figura No. 2.4: Pantalla inicial de instalación.**

La figura 2.5 muestra los términos y condiciones para uso de Quartus II web edition.



**Figura No.2.5: Términos y Condiciones.**

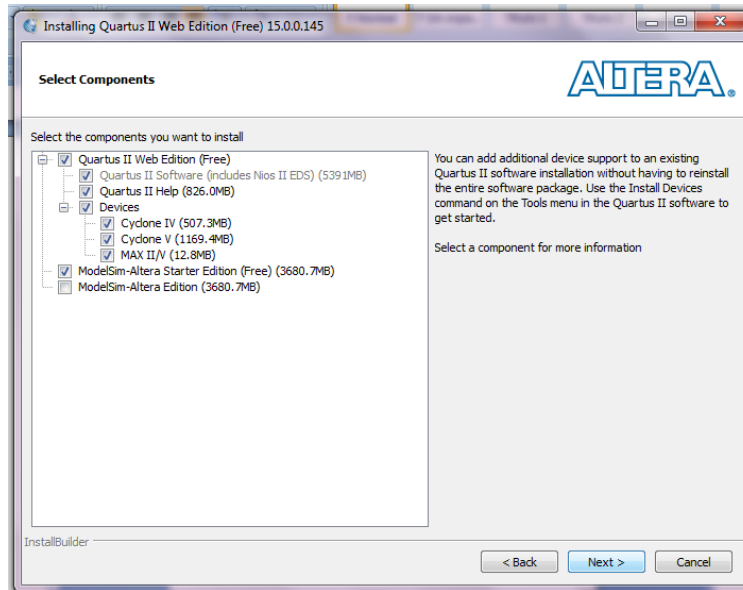
Es importante recordar la dirección de instalación ya que se necesitará mas adelante para la instalación de los drivers del programador USB, el directorio de la instalación se muestra en la figura 2.6.



**Figura No. 2.6: Directorio de instalación.**

Si se descargaron los complementos (dispositivos, ModelSim y programa de ayuda) el instalador se encargará de extraerlos en el mismo proceso de instalación tal como se muestra en la figura 2.7. Para que sean reconocidos los archivos deben estar en la misma carpeta.

El usuario está en la posibilidad de prescindir de algunos componentes, tales como **Quartus Help** ó **ModelSim Altera**, a sabiendas que no podrá disponer de estas herramientas (ayuda del software y simulación) en un futuro. Para usuarios novatos o con poca experiencia es recomendable descargar todos los complementos disponibles, incluso componentes de dispositivos que no se estén utilizando.

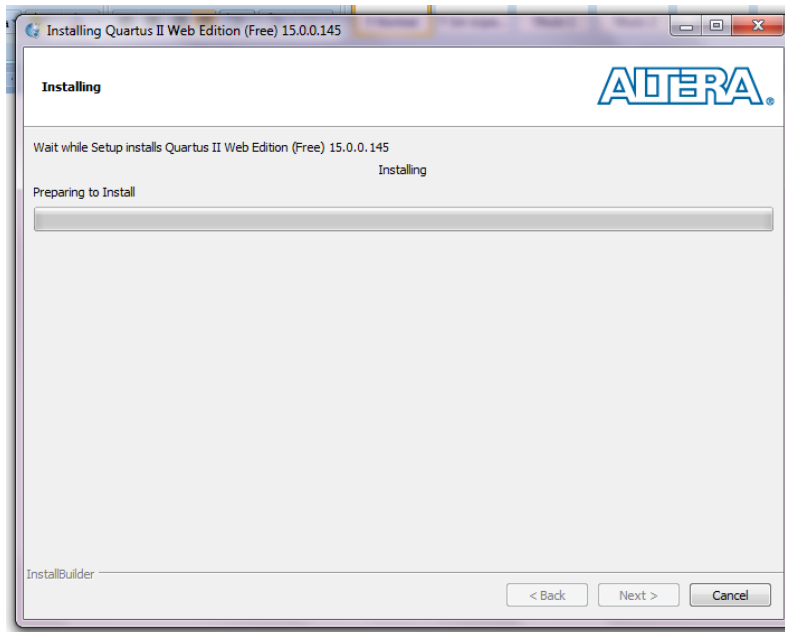


**Figura No. 2.7: Componentes de Quartus II.**

El programa exige inicialmente 11.5 Gb de espacio disponible, se debe considerar que ahí falta la herramienta Programmer, lo cual ascendería a 1 Gb mas, en la figura 2.8 únicamente se muestra los requerimientos para el programa principal.

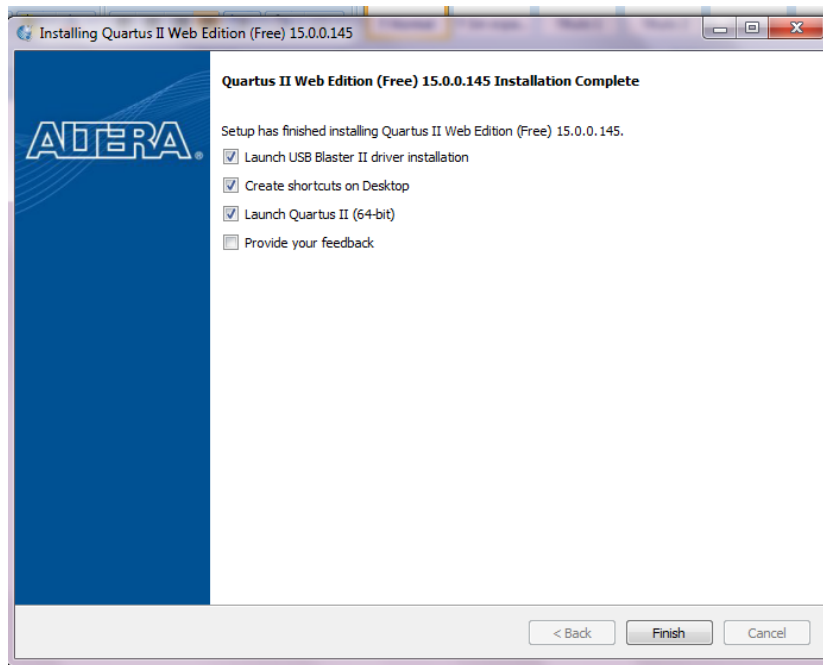


**Figura No. 2.8: Requerimiento en disco duro.**



**Figura No. 2.9: Proceso de instalación.**

El proceso completo toma alrededor de una hora con los recursos antes descritos, el tiempo puede variar en relación a las características de la computadora en donde se instala.

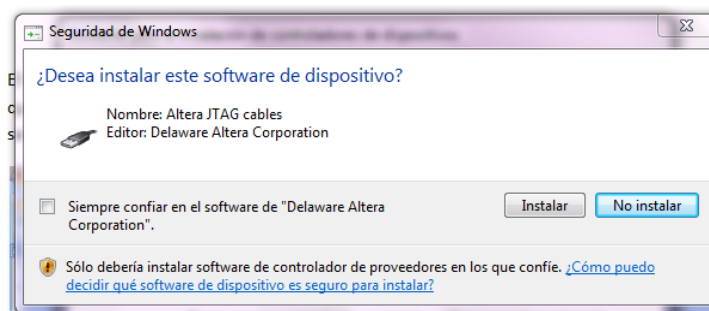


**Figura No. 2.10: Finalización de instalación.**

En la figura 2.10 se muestra la pantalla de finalización del proceso de instalación, una ventaja que tienen la versión 15.0 es que en esta ventana el mismo asistente pregunta si se quiere instalar los drivers del USB Blaster II, el cual es el dispositivo físico incluido en la tarjeta, que tiene como función programar el dispositivo FPGA.

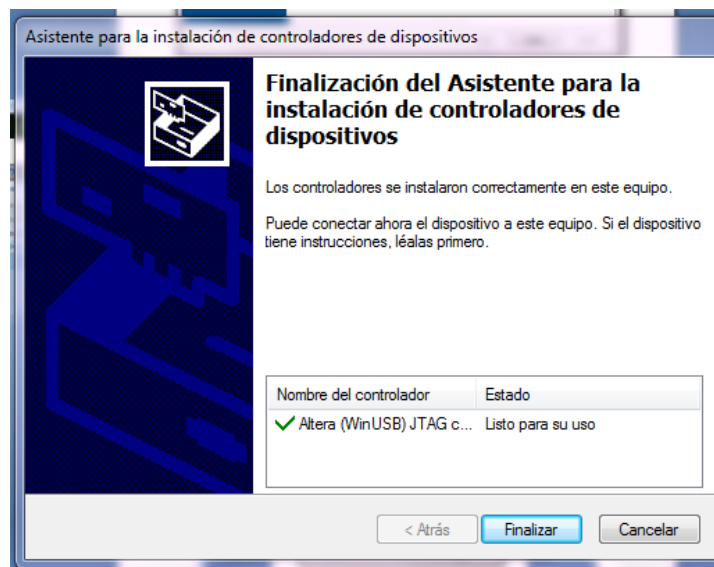
Se recomienda activar la casilla ya que el proceso de instalación será automático y el asistente se encargará de los detalles, si se da el caso que por error u omisión no se marco la casilla, al instalar el Programmer el instalador volverá a preguntar si se quieren instalar los drivers.

Si se marca el cheque de la instalación del USB Blaster II, un nuevo asistente como el que se muestra en la figura 2.11, iniciará el proceso para la instalación de los drivers, para este proceso no es necesario tener conectada la tarjeta.



**Figura No. 2.11: Instalación de drivers USB Blaster II.**

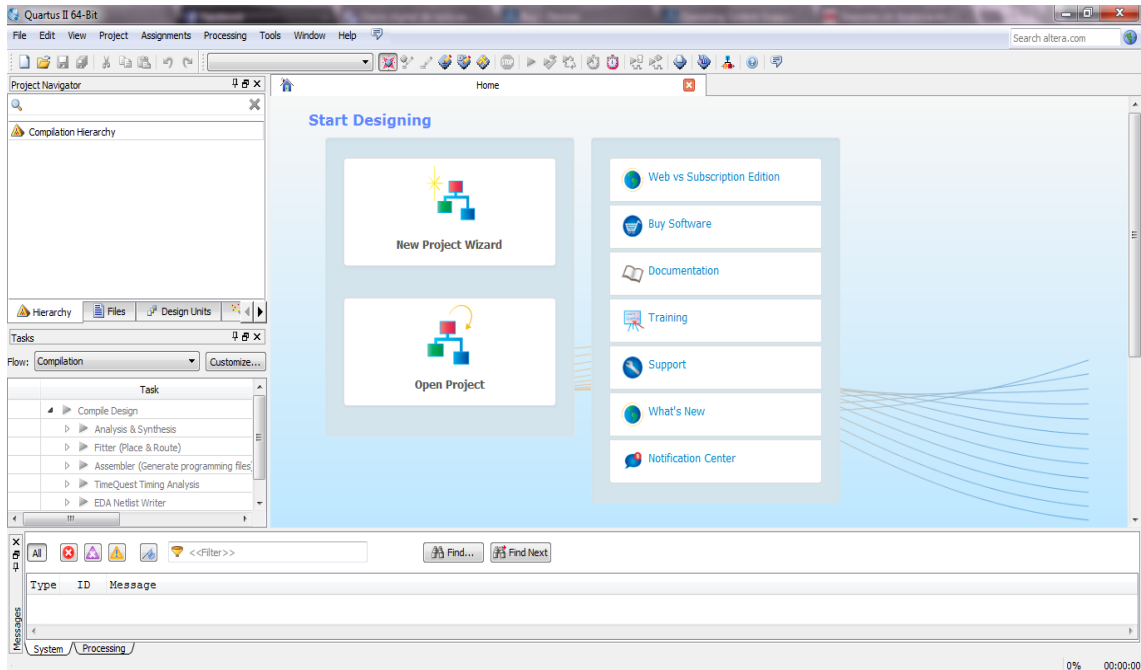
Este proceso tomará un par de minutos y no requiere mayor indicación alguna, cuando el dispositivo este correctamente instalado lanzará una pantalla similar al de la figura 2.12.



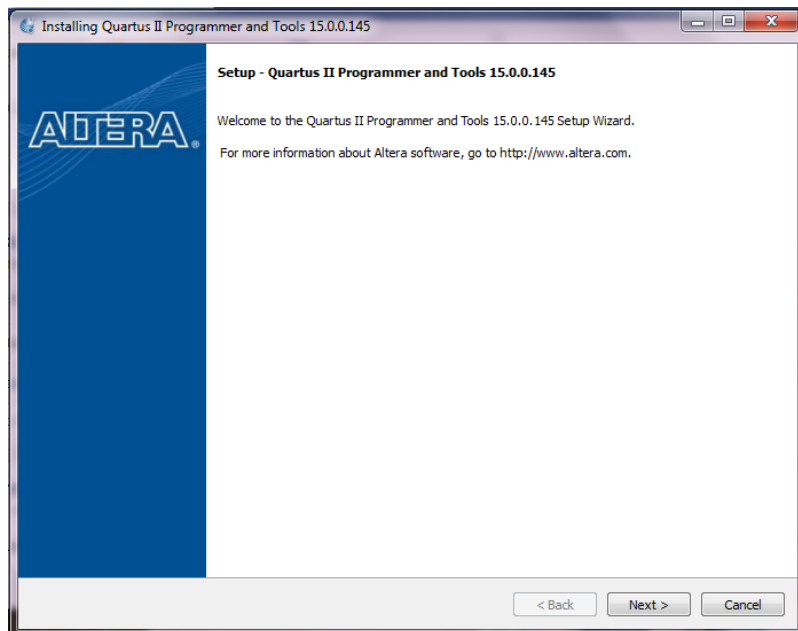
**Figura No. 2.12: Instalación correcta del USB Blaster II.**



Una vez completado este proceso, el software Quartus II se iniciara y mostrara su pantalla inicial (Ver figura 2.13), un dato a destacar es que la interfaz **programmer** es un complemento externo del Quartus II, ya que su funcionamiento e invocación puede ser independiente a la suite principal, por lo tanto el asistente de instalación no tomará en cuenta ese paquete.



**Figura No. 2.13: Pantalla inicial Quartus II.**

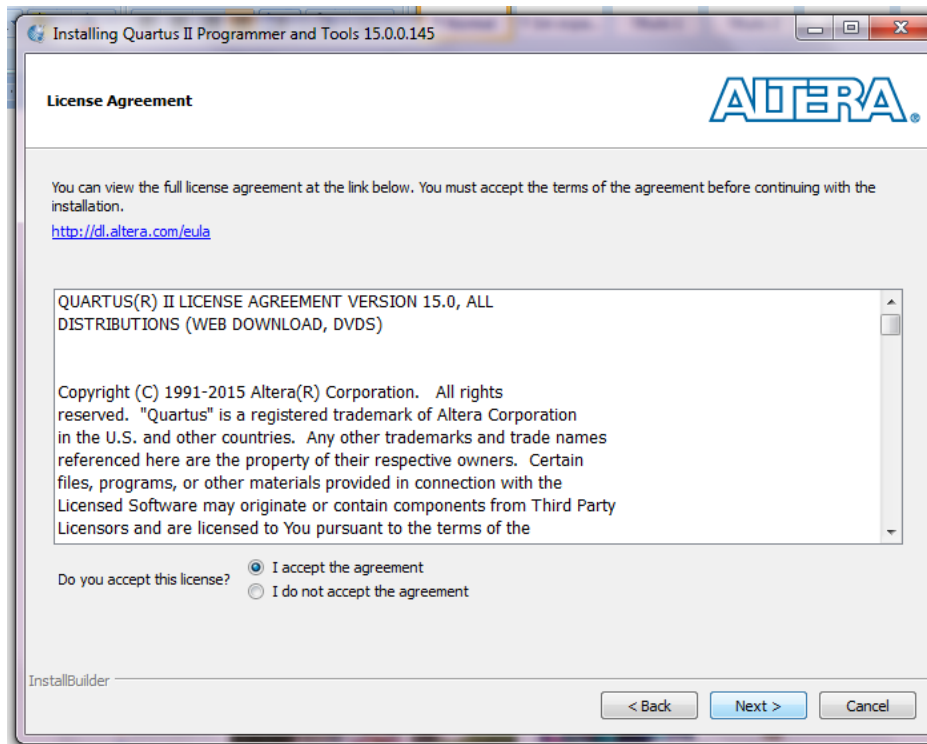


**Figura No. 2.14: instalación de Quartus II programmer.**

Para tener el programa al 100% con todas las herramientas necesarias, es recomendable cerrar la ventana de la figura 2.13 y luego ejecutar el archivo “**QuartusProgrammerSetup-15.0.0.145-windows**” el cual iniciará un asistente encargado de llevar a completar el proceso de instalación de la herramienta **Programmer**, el cual invocará un asistente como el que se muestra en la figura 2.14.

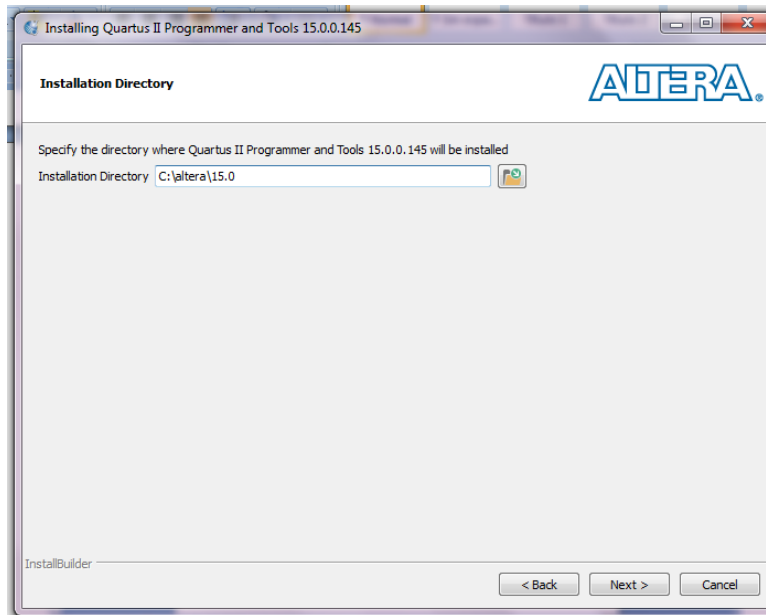
El proceso de instalación es similar al de la consola principal, de igual forma se describe su instalación paso a paso para evitar dudas.

Los términos y condiciones son los mismos que los de la consola principal, parte de estos se muestran en la figura 2.15.



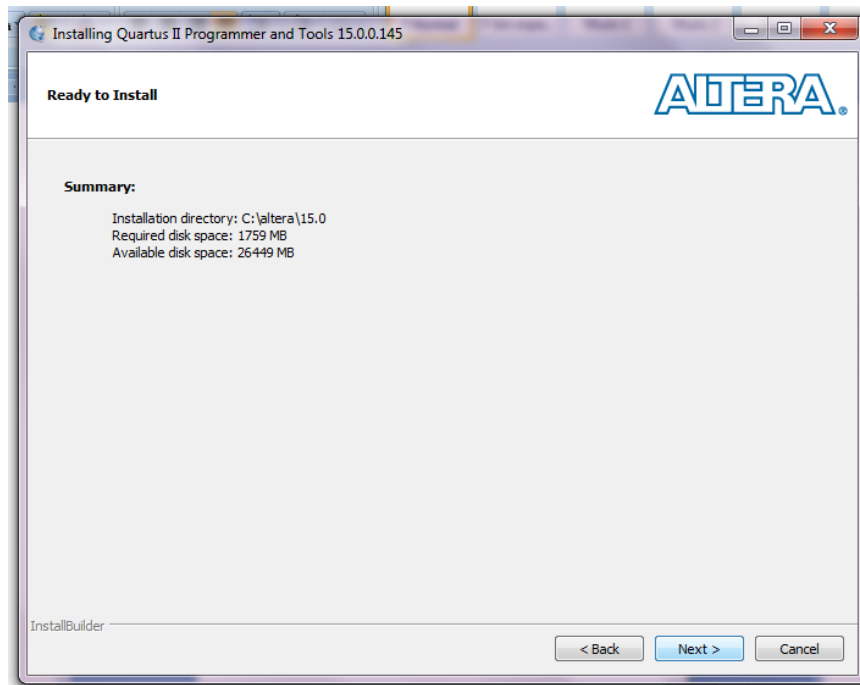
**Figura No. 2.15: Términos y condiciones del software.**

Es muy importante mantener la misma dirección de instalación tal como se muestra en la figura 2.16, para evitar confusiones y mantener un orden coherente en el proceso de instalación.



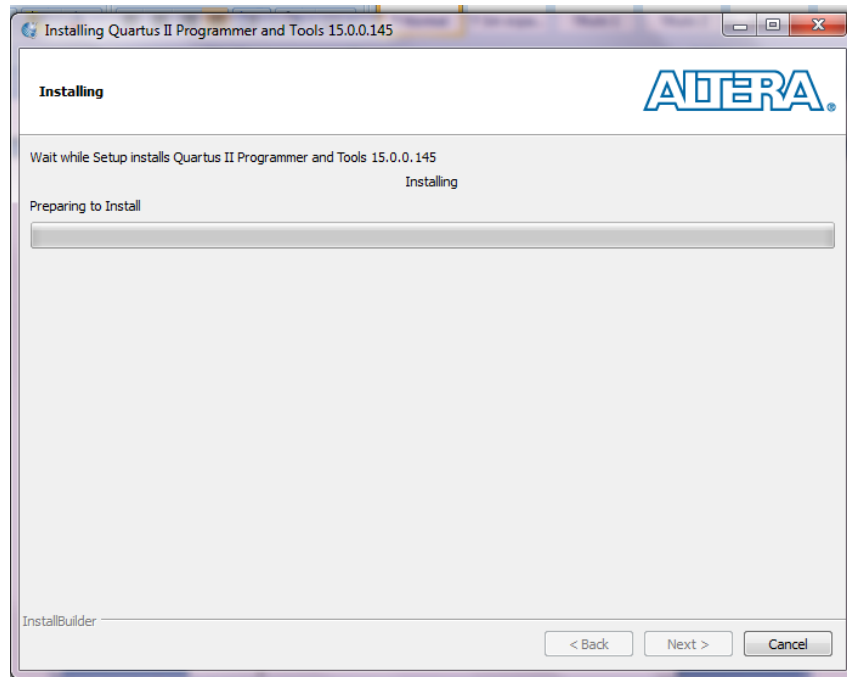
**Figura No. 2.16: Directorio de instalación.**

En total el Programmer requiere aproximadamente unos 2 Gb de disco duro disponible (el monto real se muestra en la figura 2.17), espacio que debe ser considerado desde el inicio de la instalación.

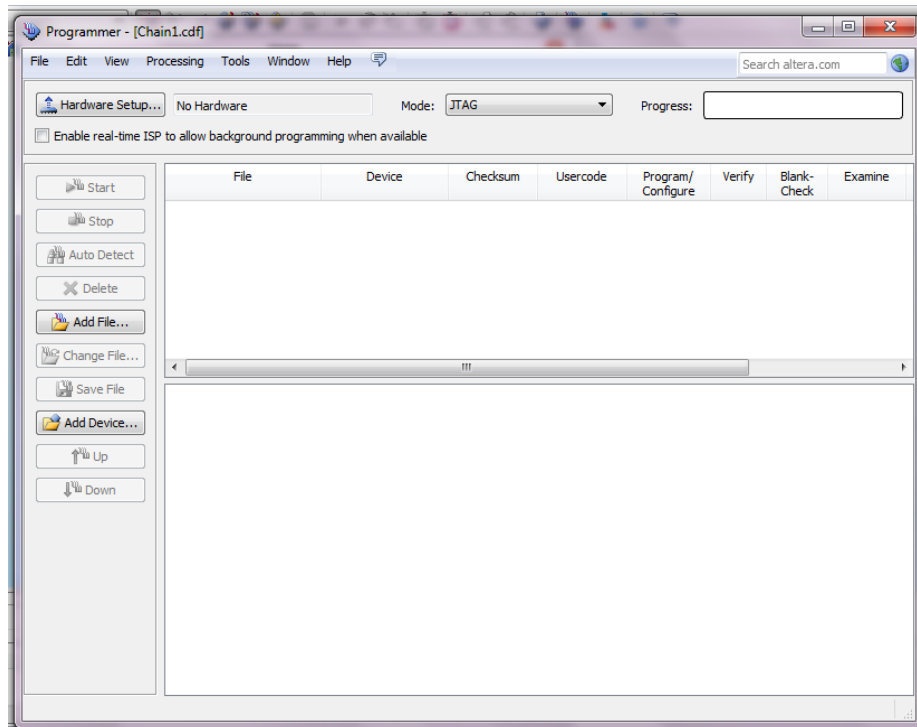


**Figura No. 2.17: Espacio requerido en disco duro.**

Luego de establecidos los parámetros de instalación el asistente se encargará de finalizar el proceso, la figura 2.18 muestra la ventana de progreso de instalación.



**Figura No. 2.18: Proceso de instalación.**



**Figura No. 2.19: Interfaz principal del Programmer.**

Una vez finalizado el proceso, el instalador abrirá el Programmer tal como se muestra en la figura 2.19. Esto comprueba dos circunstancias: la instalación se realizó con éxito y el Programmer es un programa completamente independiente a la consola principal el cual puede ser invocado con o sin ayuda de la consola.

## 2.4 Herramientas

El Quartus II dispone de muchas herramientas útiles en el diseño de circuitos lógicos y aplicativos, por lo general el uso de estas herramientas queda a criterio del usuario y no son inclusivas, ni excluyentes, de tal forma que para un diseño en general no es necesario utilizar todas las herramientas disponibles, de igual forma el uso de una herramienta no afectará el funcionamiento de otras herramientas.

### 2.4.1 Creación de un proyecto

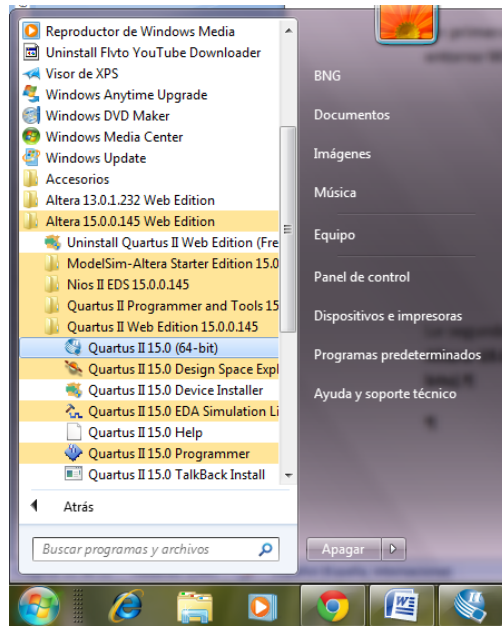
El primer paso para todo diseño es crear un proyecto con la información correspondiente del dispositivo a utilizar, de igual forma se establecen los parámetros básicos para el análisis y la síntesis del mismo.

Lo primero es abrir el software Quartus II, esto se logra de dos formas, si se trabaja en el entorno Windows, la primera opción es buscar el icono de Quartus II en el escritorio, el icono se muestra en la figura 2.20.



*Figura No. 2.20: Icono de Quartus II.*

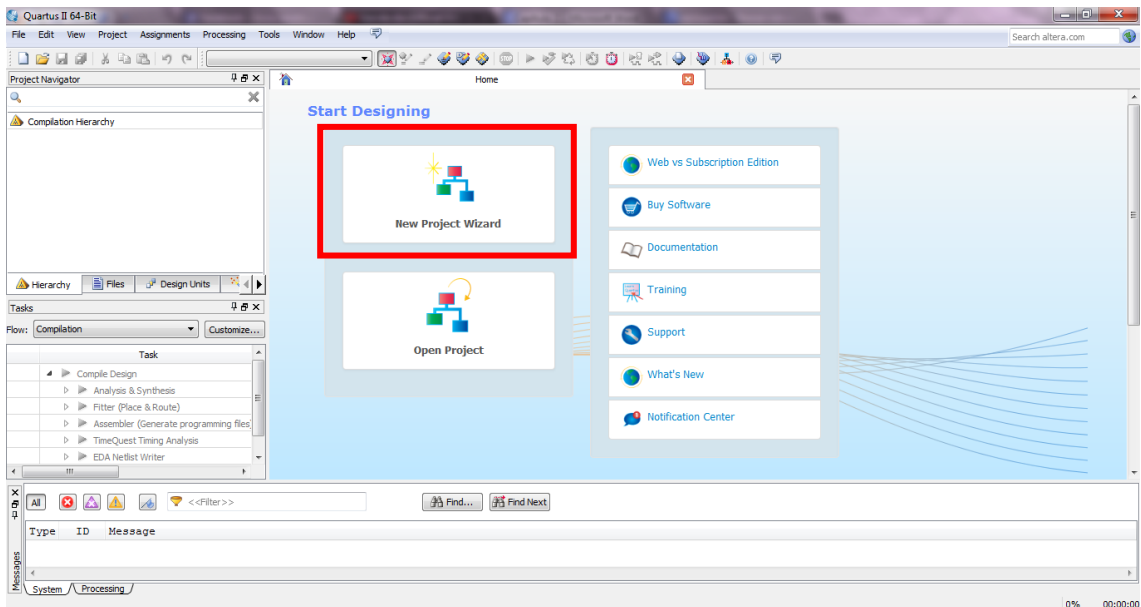
La segunda forma es seguir esta línea de procesos: **Inicio => Todos los programas => Altera 15.0.0.145 Web Edition => Quartus II Web Edition 15.0.0.145 => Quartus II 15.0 (64 bits)**, tal como se muestra en la figura 2.21.



**Figura No. 2.21: Proceso alternativo para iniciar Quartus II.**

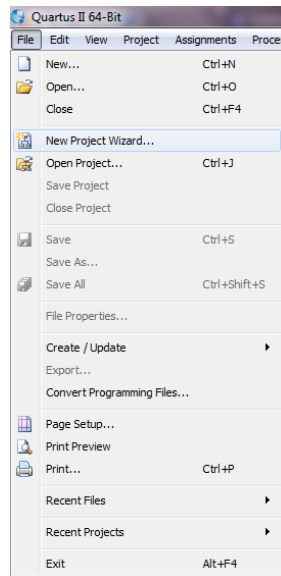
Luego se desplegará la ventana inicial de Quartus II. Para iniciar un nuevo proyecto se cuenta con dos opciones.

La opción práctica es darle clic al icono de **New Project Wizard** que se encuentra en la pantalla inicial, en la figura 2.22 se muestra el icono dentro del recuadro rojo.



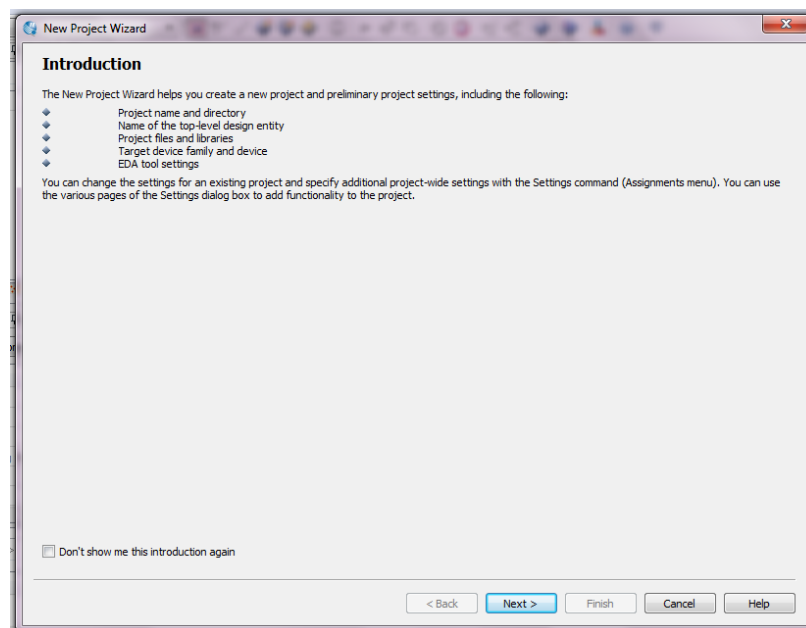
**Figura No. 2.22: Opción practica para iniciar un nuevo proyecto.**

La opción formal es realizar esta línea de procesos: **File => New Project Wizard**, para abrir un nuevo proyecto, el proceso se muestra en la figura 2.23.



**Figura No. 2.23: Opción formal para iniciar un nuevo proyecto.**

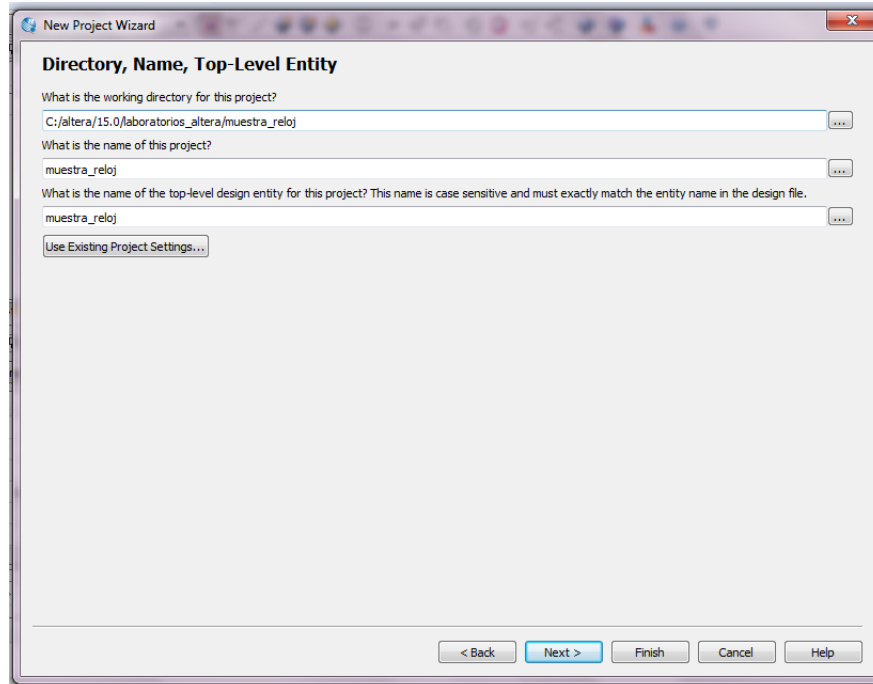
Cualquiera de las dos opciones abrirá el asistente de creación de proyectos y desplegará la ventana de introducción del mismo, tal como se muestra en la figura 2.24.



**Figura No. 2.24: Pantalla de introducción del asistente de proyectos.**

En la ventana mostrada por la figura 2.25 es recomendable declarar una carpeta principal para guardar todos los proyectos realizados por Quartus II y una subcarpeta en la cual se guarde el proyecto específico a realizar, de esta forma para el ejemplo, la carpeta en donde se guardarán todos los proyectos se llama “laboratorios\_altera”, mientras que para este proyecto en específico se creará una subcarpeta llamada “muestra\_reloj”.

Otra recomendación es que el nombre del proyecto tenga el mismo nombre de la subcarpeta que lo contendrá.



**Figura No. 2.25: Directorio de proyecto.**

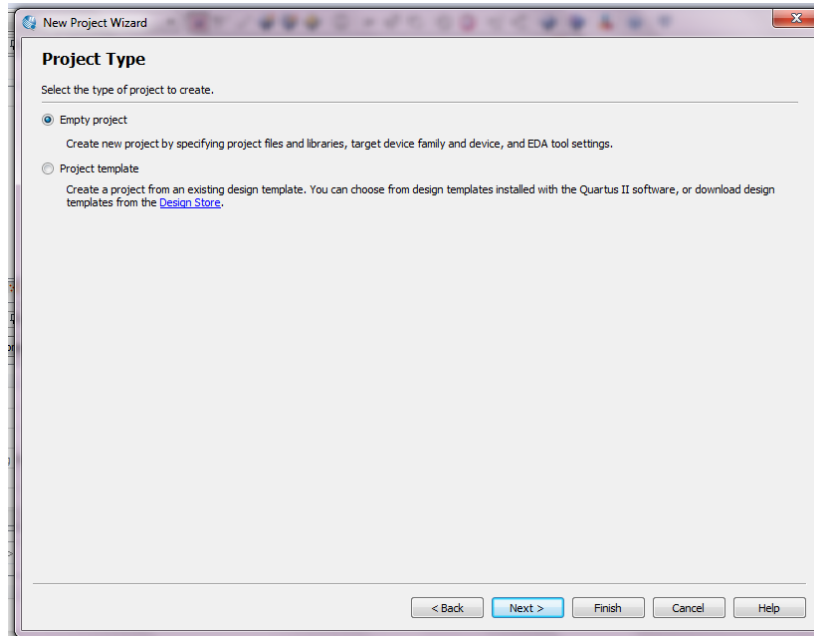
Existen dos tipos de proyectos: vacíos y plantillas.

Los de tipo vacío consideran que el usuario irá definiendo durante la marcha las librerías y paquetes que pretende utilizar, así como también archivos de diseño extra que ayuden al funcionamiento del proyecto principal; las plantillas como su nombre lo indica son proyectos prefabricados con una serie de elementos ya incluidos.

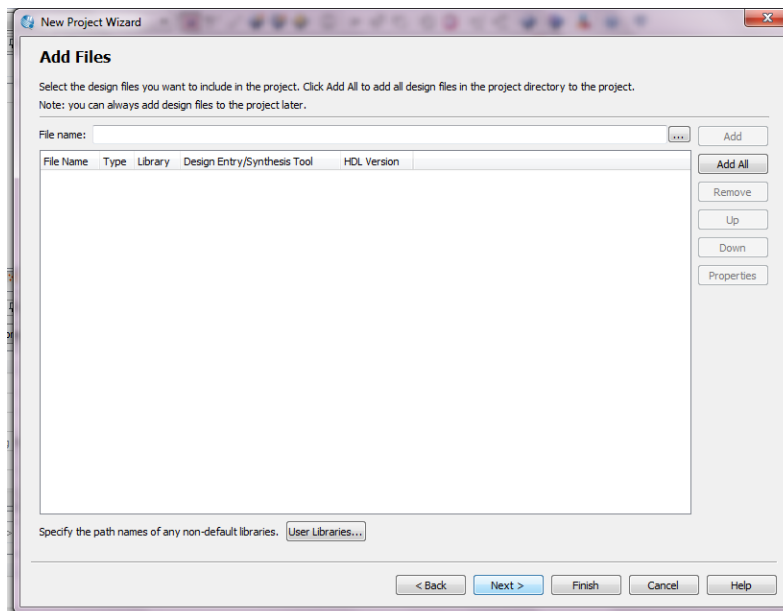
Se recomienda en las primeras etapas del aprendizaje utilizar únicamente proyectos del tipo vacío. La figura 2.26 muestra el selector de tipo de proyecto.

Si fuese necesario añadir un archivo de librería o paquete que no esté pre instalado en el software Quartus II se puede hacer mediante el asistente de adición de archivos como lo muestra la figura 2.27.





**Figura No. 2.26: Tipo de proyecto.**



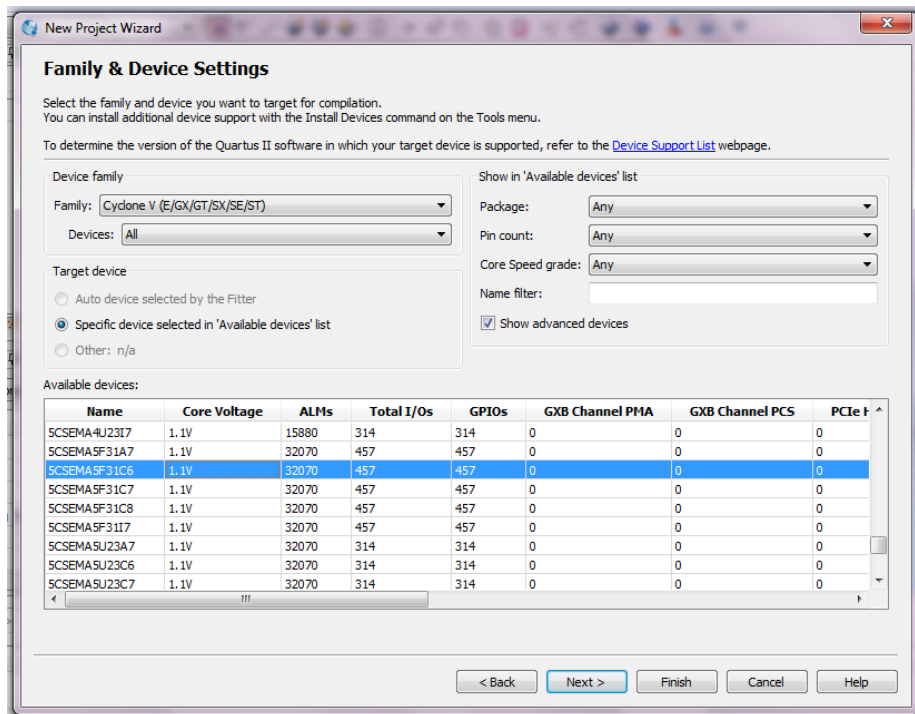
**Figura No. 2.27: Asistente para añadir archivos extra al proyecto.**

La figura 2.28 muestra la ventana de selección de dispositivos. Para saber el número correcto del dispositivo a utilizar conviene leer el manual de usuario de la tarjeta DE1-SoC, ya que en esta sección solo aparecen los números de serie y las familias a las que corresponde el dispositivo integrado en la tarjeta, no aparecerá el número de serie de la tarjeta como tal.

Para el caso de la tarjeta DE1-SoC, el chip integrado es de la familia Cyclone V SE y su número de serie es **5CSEMA5F31C6N**, el prefijo N por ser de carácter opcional rara vez aparece incluido en la lista de dispositivos por tal motivo basta que la coincidencia se dé en los primeros 12 caracteres.

Si el dispositivo de la tarjeta no aparece en la lista, es muy probable que no se hayan descargado todos los archivos de dispositivos o que la versión del software no sea compatible, para evitar que eso suceda se recomienda leer nuevamente la sección Descarga del software, contenida en este documento.

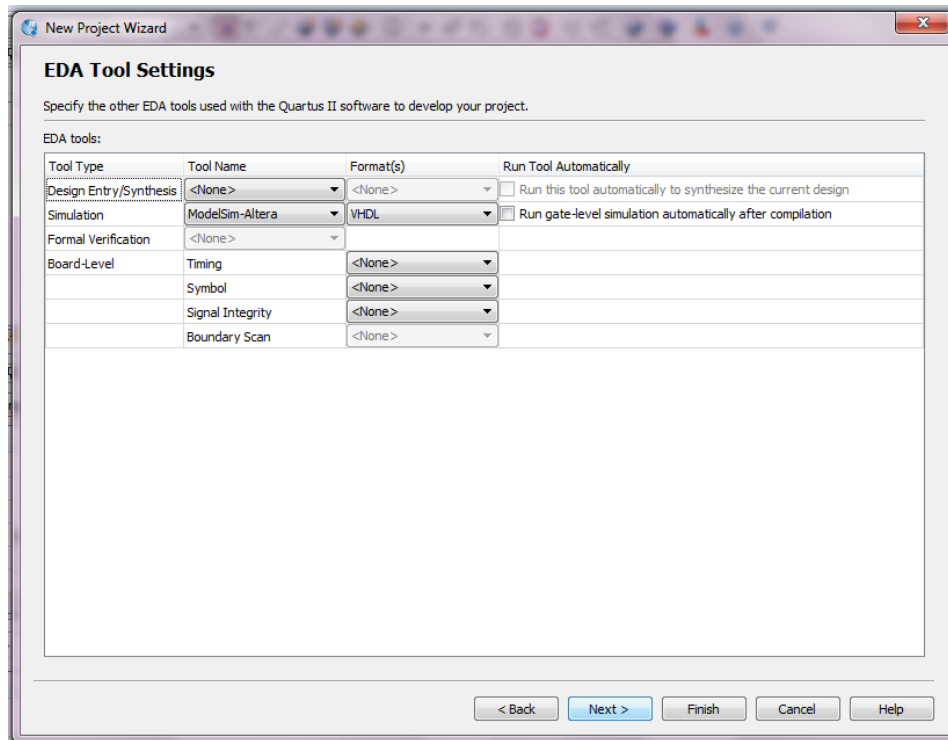
Otro método para añadir un archivo de librería es dar clic en el enlace **Device Support List** que se encuentra en la sección de selección de dispositivos del asistente de creación de proyectos.



**Figura No. 2.28: Asistente de selección de dispositivos.**

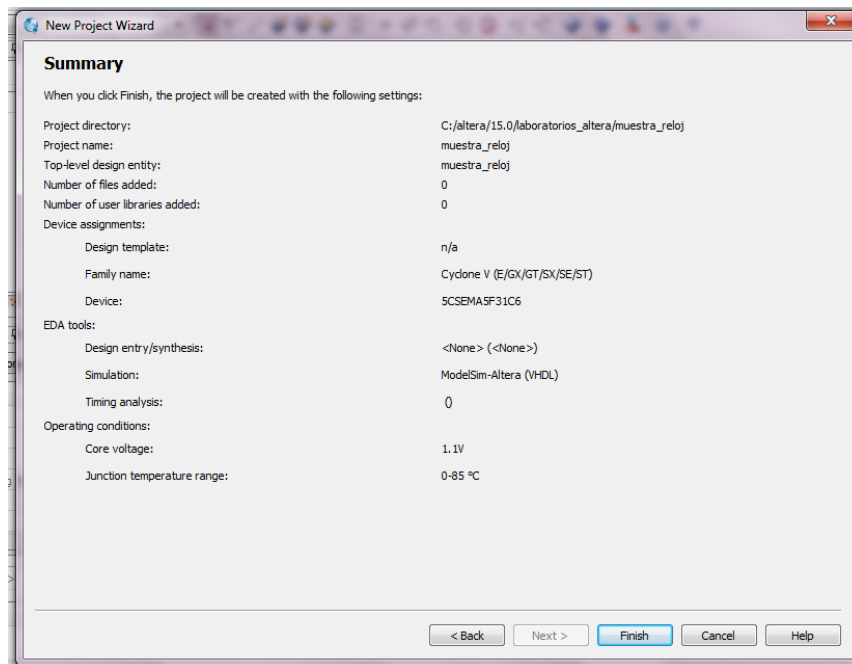
Como último paso se debe configurar las herramientas EDA (Electronics Design Automation) del programa. Estas herramientas son las encargadas del análisis, síntesis, compilación y simulación del circuito.

Si el diseño se trabajará en VHDL o en modo esquemático conviene una configuración como la que se muestra en la figura 2.29, si por el contrario se utilizara Verilog, conviene cambiar ese parámetro.



**Figura No. 2.29: Herramientas EDA.**

La ventana de la figura 2.30 es un reporte de creación del proyecto conteniendo todos los datos de configuración.

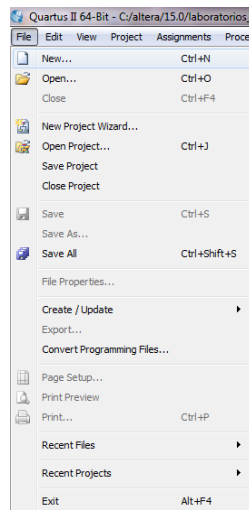


**Figura No. 2.30: Reporte de creación del proyecto.**

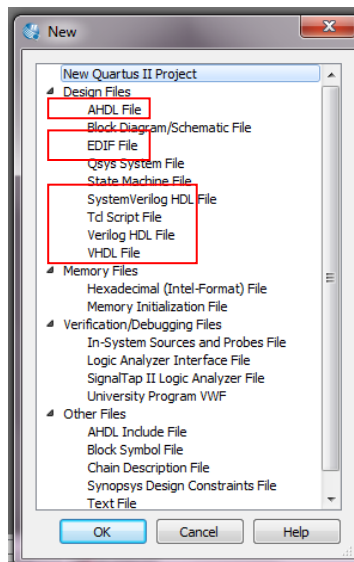
## 2.4.2 Editor de texto

Una herramienta muy utilizada son los editores de texto que provee Quartus II. En general es el mismo editor de texto para todos los archivos, la variante radica en el coloreo de la sintaxis del código, de esta forma si se invoca un archivo VHDL, el mismo editor sabrá que palabras reservadas deberá resaltar y que colores utilizara para esa tarea.

Para acceder al editor de texto se debe seleccionar el lenguaje o tipo de archivo que se quiere utilizar por medio del editor. Para ese propósito se debe seguir esta línea de procesos: **File => New...**, el proceso se muestra en la figura 2.31.



**Figura No. 2.31: Invocación de nuevos archivos.**



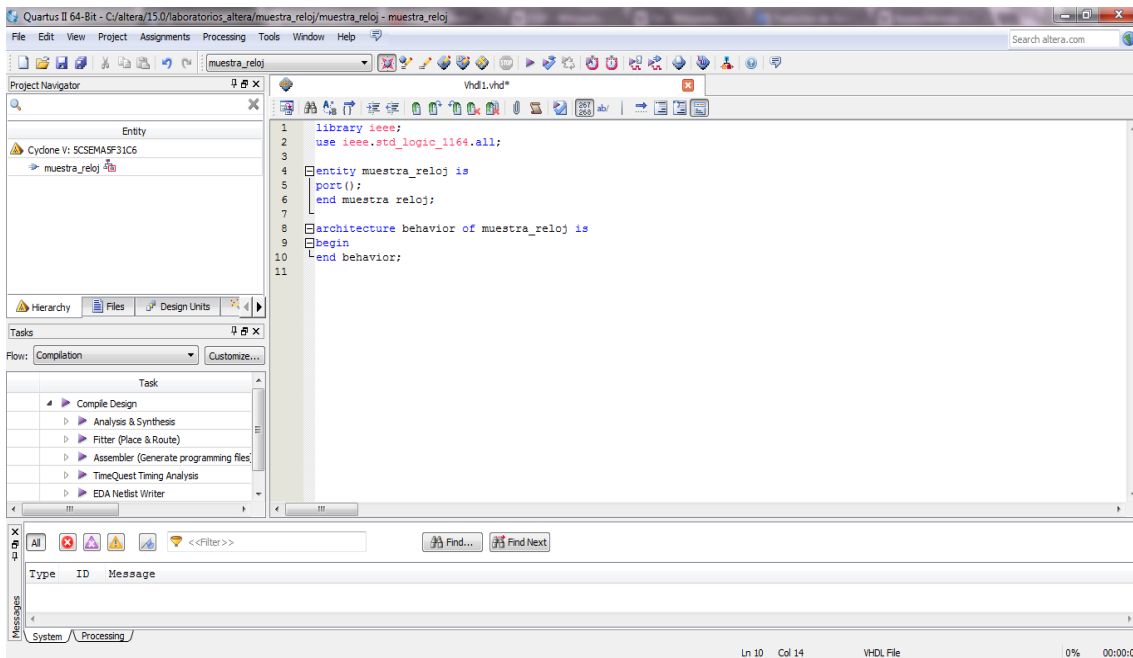
**Figura No. 2.32: Tipos de archivos para Quartus II.**

Luego de esto se activará una ventana con el tipo de archivos disponibles. Los que requieren editor de texto se encuentran señalados por un recuadro rojo dentro de la figura 2.32.

Una breve descripción de los tipos de archivos disponibles a través del editor de texto se presenta a continuación:

- AHDL File: Altera Hardware Description Language (AHDL), es un tipo de archivo específico para este lenguaje, por lo general se utiliza para describir circuitos que serán realizados por dispositivos CPLD.
- EDIF File: Electronic Design Interchange File (EDIF), estos archivos tienen como función específica ser una versión neutral para el intercambio de información concerniente a las netlist o diseños esquemáticos de circuitos.
- System Verilog/ Verilog File: Tipo de archivos de carácter específico para lenguaje Verilog, si son del tipo system se intuye que son para configuración de un dispositivo, mientras que el Verilog estándar es para el modelado de circuitos electrónicos.
- TCL Script File: Tool Command Language (TCL). Este tipo de archivos tiene como propósito describir prototipos, interfaces o bancos de pruebas.
- VHDL File: Archivo destinado a la creación de circuitos por medio de VHDL.

La figura 2.33 muestra la interfaz principal del editor de texto para un archivo VHDL.



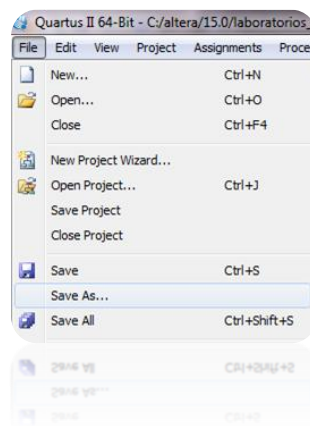
**Figura No. 2.33: Interfaz principal para editor de texto.**

Se selecciona un archivo de tipo VHDL. El editor de texto se despliega en la ventana principal del programa, cada línea está numerada lo que hace más fácil la depuración de archivos, ya que el debugger de Quartus II muestra con número de línea los errores de sintaxis (se discutirá con

mayor amplitud en una sección posterior). El nombre del archivo debe corresponder al proyecto y a la entidad principal, a menos que sea un archivo de tipo librería o paquete, caso en el cual existirán dos tipos de archivos para un solo proyecto, estos pueden ser del mismo o de diferentes tipos.

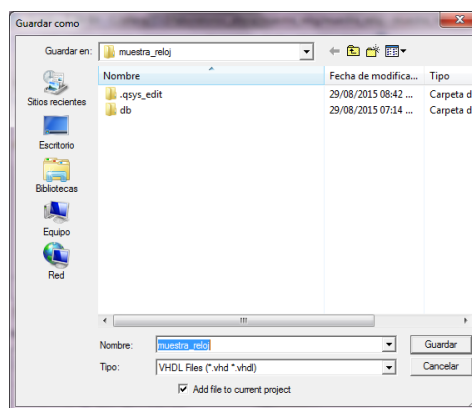
La pestaña superior muestra el nombre del archivo y la extensión del mismo, si este nombre está acompañado de un asterisco es muestra de que el archivo ha sufrido cambios y que estos no han sido guardados.

Para guardar un archivo se siguen los siguientes pasos: **File => Save As...**, este proceso se muestra en la figura 2.34.



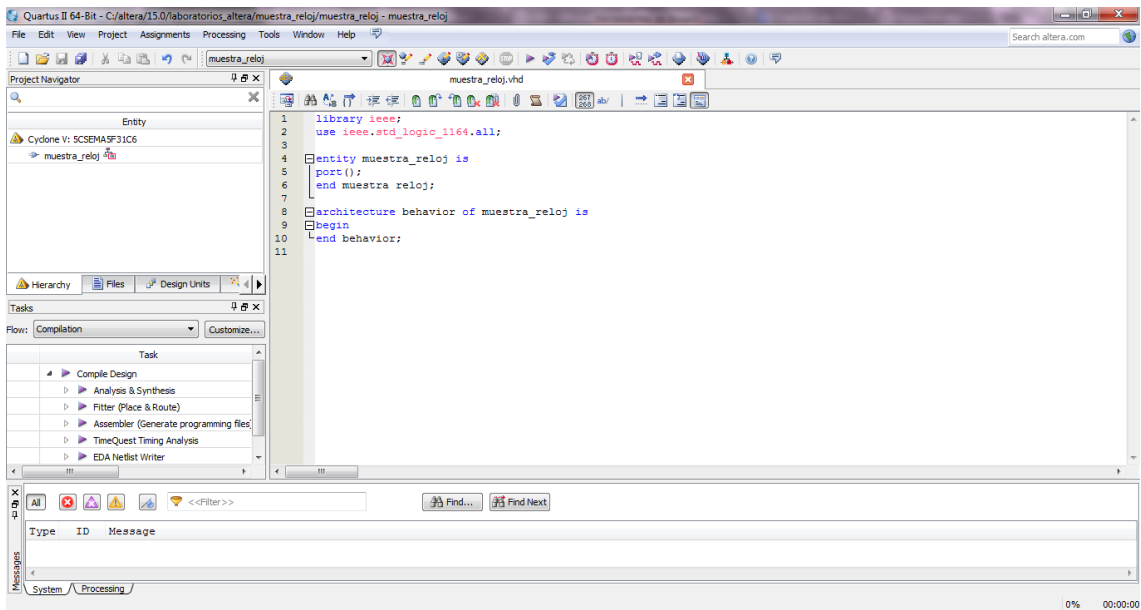
**Figura No. 2.34: Proceso para guardar un archivo.**

El programa por defecto buscará la subcarpeta que se designó para el proyecto y le asignará un nombre al archivo similar al del proyecto, si existiera más de un archivo VHDL en el proyecto, aquel que contenga la entidad o módulo principal es el que llevará el nombre del proyecto por defecto y cada archivo debe ser guardado por separado, la figura 2.35 muestra el buscador de directorios para guardar los archivos de texto.

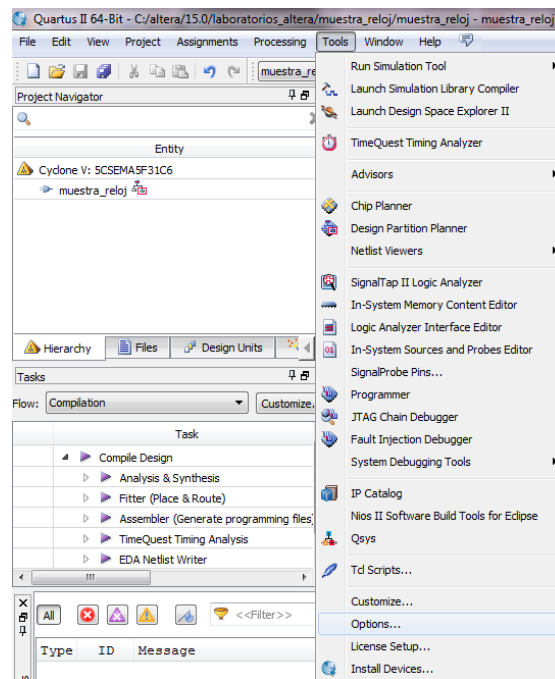


**Figura No. 2.35: Directorio de archivos.**

Al guardar el archivo se verán cambios inmediatos en la pestaña del nombre de archivo, desaparecerá el asterisco y ahora tendrá en nombre asignado, eso puede corroborarse en la figura 2.36.



**Figura No. 2.36: Archivo de texto guardado con éxito.**

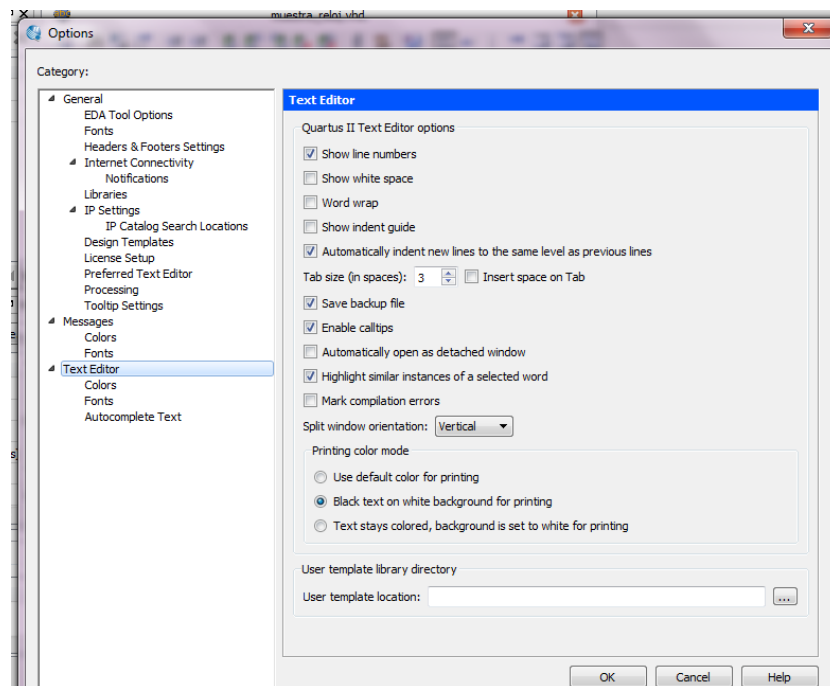


**Figura No. 2.37. Proceso para acceder a menú de opciones.**

El usuario puede hacer las modificaciones pertinentes al editor de texto para mayor comodidad. Estas modificaciones son en su mayoría de carácter gráfico, ya que se pueden modificar el tipo y tamaño de fuente, así como los colores de la misma para las diferentes opciones de resaltado, aunque también hay opciones de comportamiento del editor.

Para acceder a este menú de configuración se deben seguir estos procesos: **Tools => Options...**, este proceso se muestra en la figura 2.37.

Esto activará un submenú mostrado en la figura 2.38, en el cual se debe buscar la opción **Text Editor**.

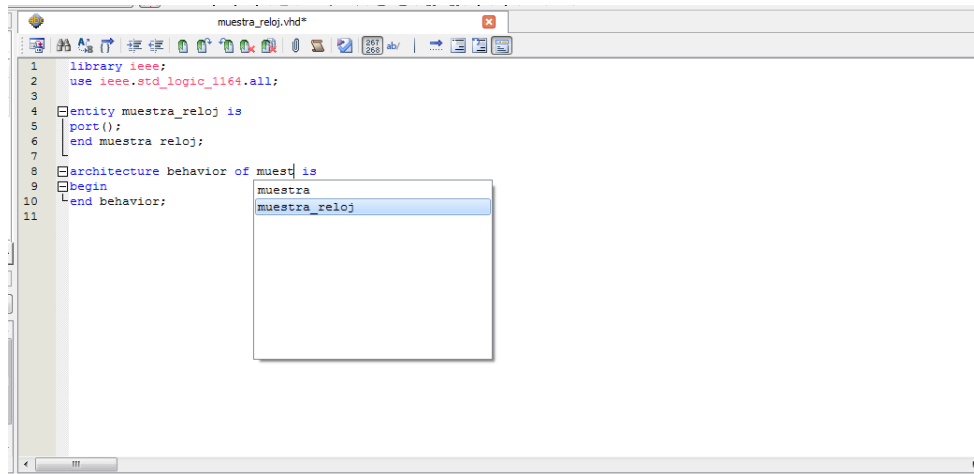


**Figura No. 2.38. Menú de opciones para el editor de texto.**

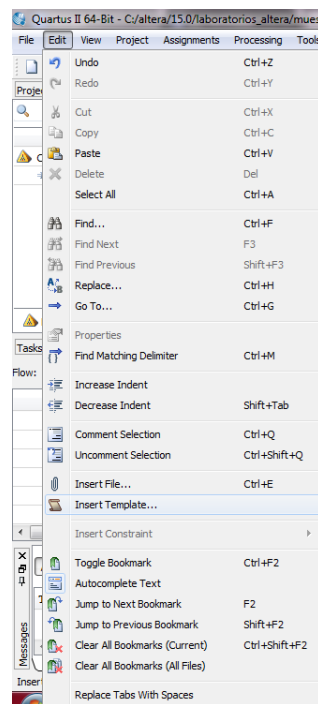
Además de esto el editor de texto cuenta con la opción de autorelleno. Esto permite que el editor recuerde palabras clave del diseño en cuestión, de tal forma que si la palabra se utiliza muchas veces dentro del archivo se encuentre disponible en el autorelleno y baste con darle a la tecla TAB para que esta palabra sea agregada. Lo mismo aplica a las palabras reservadas del lenguaje seleccionado. Todo esto puede ser configurado a través del menú de opciones del editor de texto. La figura 2.39 muestra la herramienta de autorelleno en funciones.

Si en algún momento es necesario agregar una unidad de código de la cual se desconozca la sintaxis, Quartus II provee una serie de plantillas muy útiles en las cuales se muestran la sintaxis de la mayoría de unidades de los HDL admitidos en el editor de texto.





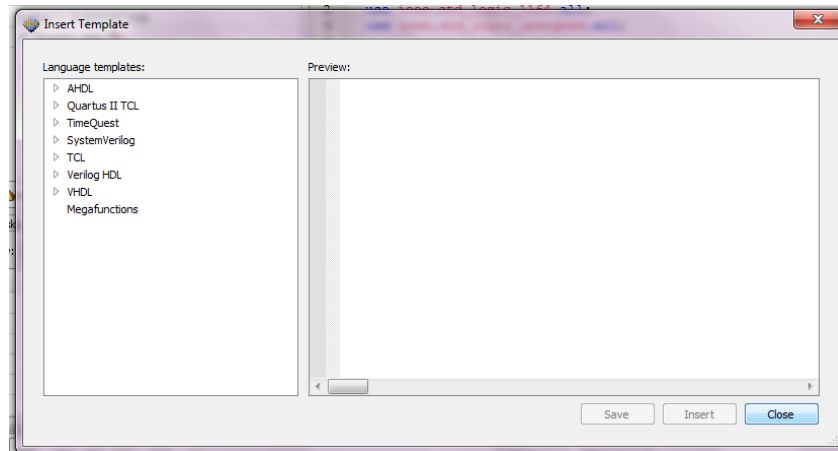
**Figura No. 2.39: Herramienta de autorelleno.**



**Figura No. 2.40: Proceso para acceder a las plantillas.**

Para acceder a esta plantilla se debe seguir este proceso: **Edit => Insert Template...**, tal como se muestra en la figura 2.40.

La figura 2.41 muestra la ventana de selección de plantillas, cada lenguaje tiene sus propias plantillas solo basta seleccionar el tipo de lenguaje y la unidad de la cual se desea saber la sintaxis y luego se da clic en **insert**.



**Figura No. 2.41: menú de plantillas.**

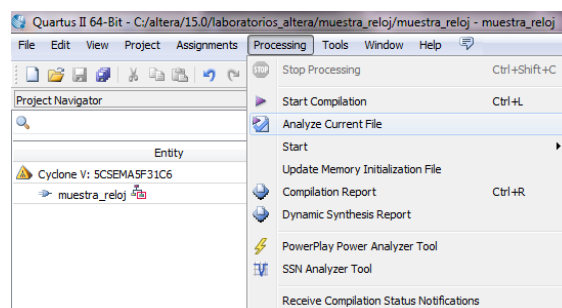
### 2.4.3 Análisis y síntesis

Esta herramienta permite un análisis del código vhdl antes de ser compilado en su totalidad, hacer una sección sobre esta herramienta se vuelve casi obligatorio porque hay aspectos de optimización que vale la pena resaltar.

Con anterioridad se menciona que la compilación y simulación exige muchos recursos de la PC en particular de la memoria RAM, tal es el caso que para una compilación completa con los requerimientos básico el tiempo de la misma puede ascender a los 8 ó 9 minutos, la longitud del código puede agregar un par de segundos al proceso pero no hace una diferencia significativa, algo relativamente exagerado considerando que no se está tratando el procesamiento de imágenes o no se están moviendo bloques de grandes cantidades de bits.

El porqué de ese tiempo está justificado en la cantidad de procesos realizados por el compilador, que demanda una cola de datos y registros bastante amplia, ya que la compilación en paralelo no está disponible para la Web Edition del software.

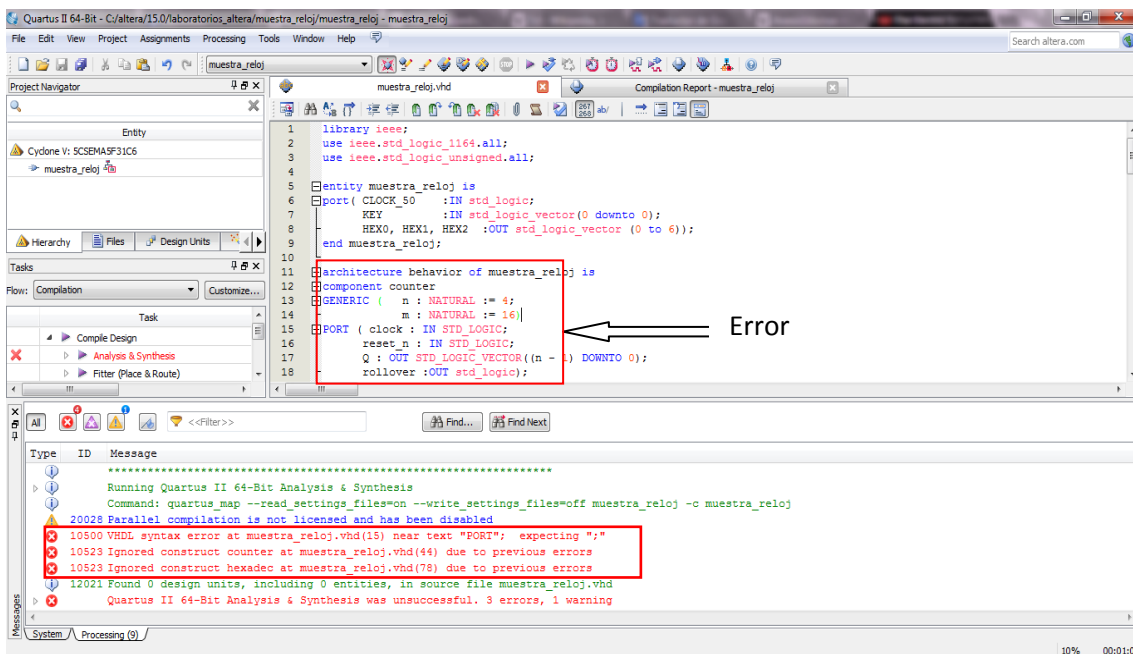
Por lo tanto la herramienta **Analyze Current File** disponible a través de la pestaña **Processing** brinda la opción de desglosar el análisis y síntesis del código, sin necesidad de invocar una compilación completa, este proceso es mostrado en la figura 2.42.



**Figura No. 2.42: Proceso de análisis.**

Lo que hace tan importante esta separación es que el programa no se satura con la invocación de un proceso de compilación completo, y es capaz de realizar el análisis en menor tiempo, lo cual contribuye a la depuración del código.

Si el código atraviesa esta etapa de forma satisfactoria se puede tener la certeza de que la compilación será un éxito, siempre y cuando se respete el pinout del dispositivo (se discutirá en una sección posterior).

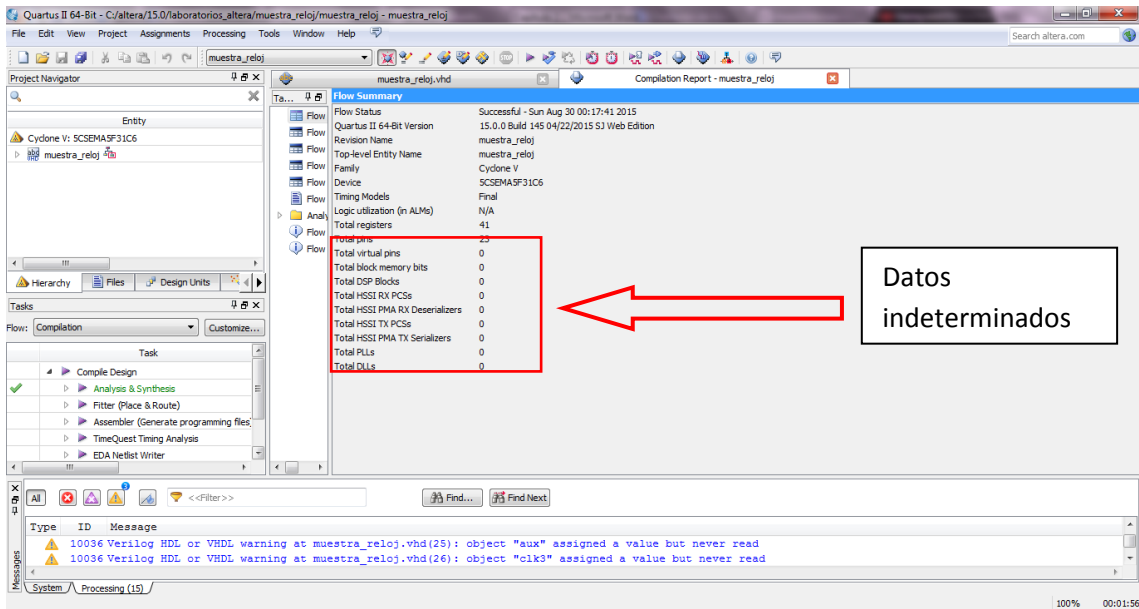


**Figura No. 2.43: Depurador.**

Si el código presenta errores el análisis no se terminará y entrará en efecto el depurador mostrando la línea donde surge el error y una breve explicación del porque está fallando, en el ejemplo de la figura 2.43, de manera intencional se omitió un punto y coma (;) para que el depurador lanzara el error. En la parte posterior de la imagen se puede observar con letras rojas, la causa y la ubicación del error.

Una vez corregido el error se procede a analizar nuevamente el archivo, esta vez de forma satisfactoria arrojando un reporte de compilación, cabe aclarar que este reporte solo engloba una sección de la compilación, por lo tanto muchos datos no serán visibles.

La figura 2.44 muestra un reporte preliminar de compilación.



**Figura No. 2.44: Reporte Preliminar.**

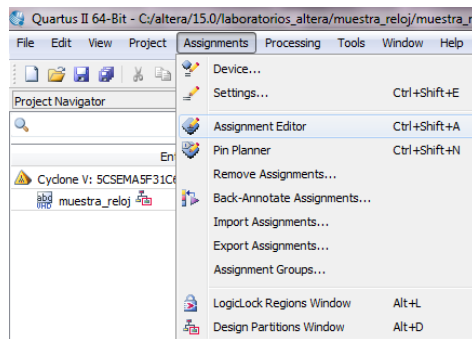
### 2.4.4 Asignación de pines

Es de carácter obligatorio conocer bien el proceso de asignación de pines, ya que los pines del FPGA están preestablecidos en la tarjeta para cada uno de los periféricos y componentes, una mala asignación de pines con seguridad conducirá a daños al equipo.

Para realizar la asignación de pines existen dos métodos: asignación manual y asignación por archivo de ajuste.

#### 2.4.4.1 Asignación manual

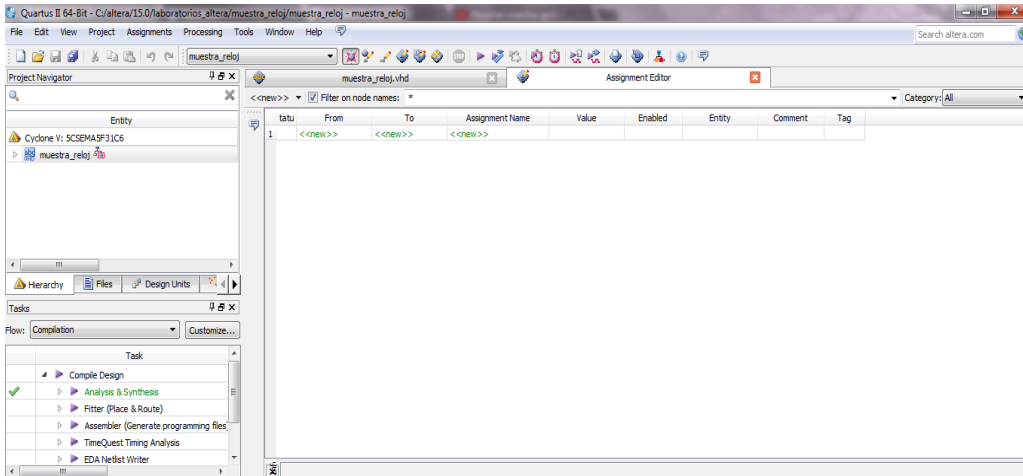
Este proceso requiere paciencia y alta concentración para evitar errores, además se debe tener disponible el manual de la tarjeta.



**Figura No. 2.45: Proceso para asignación de pines.**

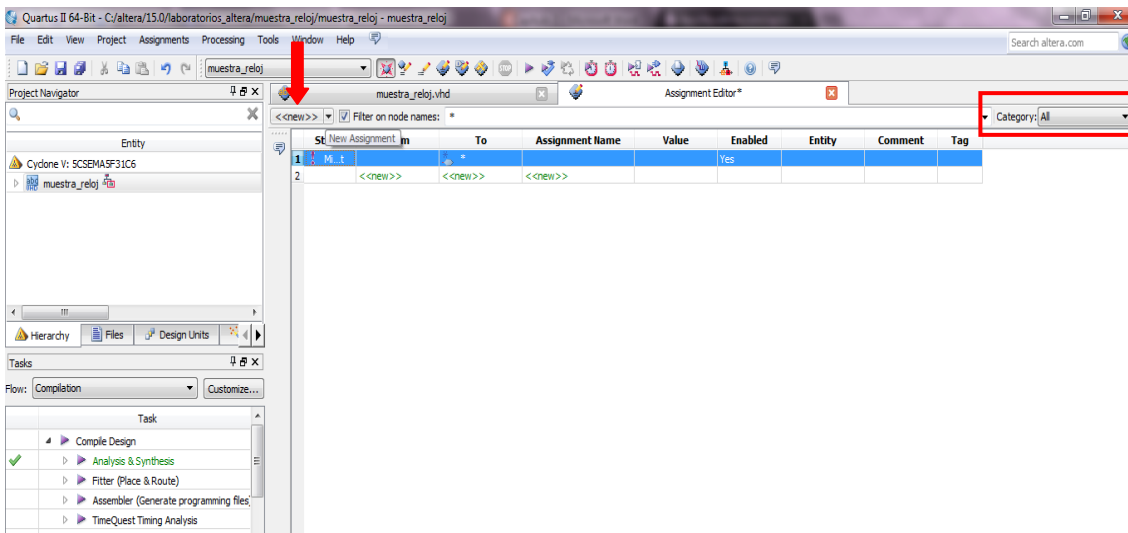
Una vez disponiendo del manual se consideran los puertos a utilizar, tomando como referencia el código anterior en el cual se tenían los puertos CLOCK\_50, KEY[0], HEX0, HEX1 Y HEX2, se procede a invocar la herramienta **Assignment Editor** la cual se encuentra en la pestaña **Assignments**, este procedimiento se muestra en la figura 2.45.

Una vez se abre la herramienta se mostrará la siguiente pestaña (ver figura 2.46):



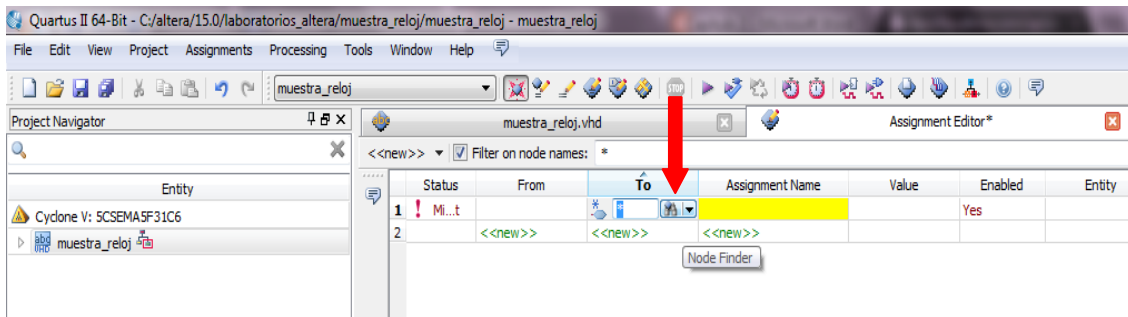
**Figura No. 2.46: Editor de pines.**

En la sección **Category** se debe seleccionar la opción **All**, seguidamente se debe dar clic en el botón **New** ubicado en la esquina superior izquierda de la pestaña, tal como lo indica la flecha de la figura 2.47, esto agregará un nuevo elemento a la lista.



**Figura No. 2.47: Asignación de pines.**

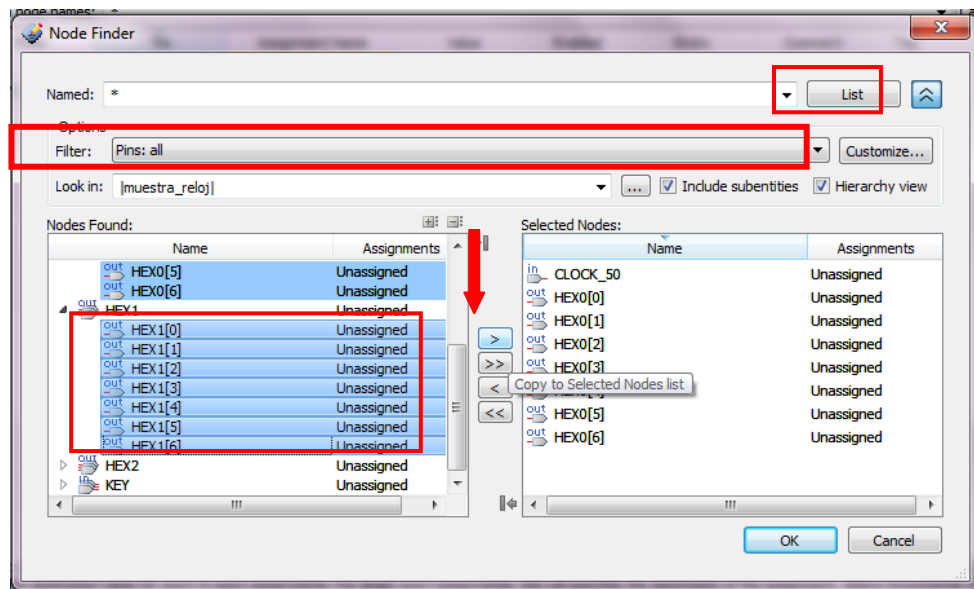
Luego se le debe dar doble clic a la celda correspondiente a la columna **To**, esto activará el botón **Node Finder**, el botón se indica con una flecha en la figura 2.48.



**Figura No. 2.48: Node Finder (I).**

Al dar clic en este botón desplegará el **Node Finder** tal como se muestra en la figura 2.49. Esta herramienta ayuda a buscar y agregar los pines involucrados en el proceso. Para tal motivo en la casilla **Filter** se debe buscar la opción **Pins: Alls**, luego se le debe dar clic al botón **List**. Esto desplegará los pines involucrados en el circuito. La herramienta **Node Finder**, reconoce como pines los elementos incluidos en el Port de la entidad principal.

Al tener los pines en la columna **Node Found** se deben seleccionar los pines de forma individual y luego darle clic al botón **>** esto enviará los pines a la columna **Selected Node**. En el caso de los HEX que son vectores de pines, no se debe seleccionar el grupo, sino que se debe seleccionar los pines individuales. Para finalizar se debe dar clic en **Ok**.



**Figura No. 2.49: Node Finder (II).**

Si todo el proceso de selección se hizo bien, la tabla de asignación se debería ver como en la figura 2.50:

Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	Mi...	in CLOCK_50			Yes			
2	Mi...	out HEX0[0]			Yes			
3	Mi...	out HEX0[1]			Yes			
4	Mi...	out HEX0[2]			Yes			
5	Mi...	out HEX0[3]			Yes			
6	Mi...	out HEX0[4]			Yes			
7	Mi...	out HEX0[5]			Yes			
8	Mi...	out HEX0[6]			Yes			
9	Mi...	out HEX1[0]			Yes			
10	Mi...	out HEX1[1]			Yes			
11	Mi...	out HEX1[2]			Yes		Yes	Status: Missing Assignment Name
12	Mi...	out HEX1[3]			Yes			
13	Mi...	out HEX1[4]			Yes			
14	Mi...	out HEX1[5]			Yes			
15	Mi...	out HEX1[6]			Yes			
16	Mi...	out HEX2[0]			Yes			
17	Mi...	out HEX2[1]			Yes			
18	Mi...	out HEX2[2]			Yes			
19	Mi...	out HEX2[3]			Yes			
20	Mi...	out HEX2[4]			Yes			
21	Mi...	out HEX2[5]			Yes			
22	Mi...	out HEX2[6]			Yes			
23	Mi...	in KEY[0]			Yes			

This cell specifies the assignment name or type.

**Figura No. 2.50: Tabla con todos los pines.**

En las celdas amarillas se debe dar doble clic en cada una de éstas y seleccionar la opción **Location (Accept Wildcards/Groups)**, tal como se muestra en la figura 2.51, esta opción le dice al software que este pin se refiere a una ubicación física dentro del dispositivo FPGA.

Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	Mi...	in CLOCK_50						
2	Mi...	out HEX0[0]	GXB Reserved Transmit Channel (Accepts wildcards/groups)					
3	Mi...	out HEX0[1]	Global Signal (Accepts wildcards/groups)					
4	Mi...	out HEX0[2]	HDL Initial Fan-out Limit (Accepts wildcards/groups)					
5	Mi...	out HEX0[3]	HPS_IO (Accepts wildcards/groups)					
6	Mi...	out HEX0[4]	Half bandwidth mode (Accepts wildcards/groups)					
7	Mi...	out HEX0[5]	I/O Maximum Toggle Rate (Accepts wildcards/groups)					
8	Mi...	out HEX0[6]	I/O Standard (Accepts wildcards/groups)					
9	Mi...	out HEX1[0]	Ignore CARRY Buffers (Accepts wildcards/groups)					
10	Mi...	out HEX1[1]	Ignore CASCADE Buffers (Accepts wildcards/groups)					
11	Mi...	out HEX1[2]	Ignore GLOBAL Buffers (Accepts wildcards/groups)					
12	Mi...	out HEX1[3]	Ignore LCELL Buffers (Accepts wildcards/groups)					
13	Mi...	out HEX1[4]	Ignore Maximum Fan-Out Assignments (Accepts wildcards/groups)					
14	Mi...	out HEX1[5]	Ignore R... GLOBAL Buffers (Accepts wildcards/groups)					
15	Mi...	out HEX1[6]	Ignore S... Buffers (Accepts wildcards/groups)					
16	Mi...	out HEX2[0]	Ignore Ver... initial constructs (Accepts wildcards/groups)					
17	Mi...	out HEX2[1]	Implement Clock Enable					
18	Mi...	out HEX2[2]	Implement Output of Logic Cell					
19	Mi...	out HEX2[3]	Infer RAM from Raw Logic (Accepts wildcards/groups)					
20	Mi...	out HEX2[4]	Input Transition Time (Accepts wildcards/groups)					
			Iteration limit for constant Verilog loops (Accepts wildcards/groups)					
			Iteration limit for non-constant Verilog loops (Accepts wildcards/groups)					
			Keep synchronous clear/preset behavior for DIO INPUT when unmap I/O wysiwyg primitives (Accepts wildcards/groups)					
			LVDS Direct Loopback Mode (Accepts wildcards/groups)					
			Location (Accepts wildcards/groups)					

**Figura No. 2.51: Definición del tipo de pin.**

Una vez completada todas las celdas, en la columna **Value** para cada pin se debe escribir manualmente el nombre de cada pin según se describe en el manual de usuario de la tarjeta, para esto basta con darle doble clic a la celda deseada y luego escribir el nombre del pin, la Tabla 1 muestra los valores de los pines involucrados en el ejemplo y la figura 2.52 muestra la tabla de asignación luego de completado el proceso.

Nombre	Pin FPGA	Nombre	Pin FPGA	Nombre	Pin FPGA
CLOCK_50	PIN_AF14	HEX0[6]	PIN_AH28	HEX2[0]	PIN_AB23
KEY[0]	PIN_AA14	HEX1[0]	PIN_AJ29	HEX2[1]	PIN_AE29
HEX0[0]	PIN_AE26	HEX1[1]	PIN_AH29	HEX2[2]	PIN_AD29
HEX0[1]	PIN_AE27	HEX1[2]	PIN_AH30	HEX2[3]	PIN_AC28
HEX0[2]	PIN_AE28	HEX1[3]	PIN_AG30	HEX2[4]	PIN_AD30
HEX0[3]	PIN_AG27	HEX1[4]	PIN_AF29	HEX2[5]	PIN_AC29
HEX0[4]	PIN_AF28	HEX1[5]	PIN_AF30	HEX2[6]	PIN_AC30
HEX0[5]	PIN_AG28	HEX1[6]	PIN_AD27		

*Tabla No. 2.1: Pines FPGA involucrados en el ejemplo.*

Cuando el número de pin es válido la fila tendrá un cheque verde con la palabra OK, esto significa que el pin existe dentro del dispositivo pero no indica que este bien asignado, la única forma es asegurarse y revisar bien la configuración que se le ha dado al pin.

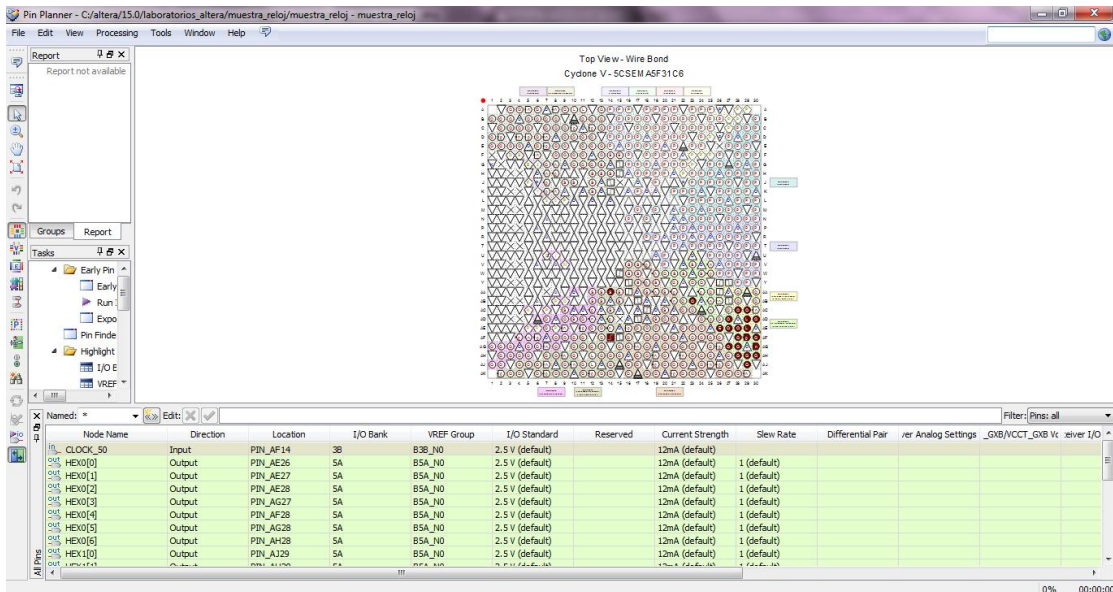
Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	OK	CLOCK_50	Location	PIN_AF14	Yes			
2	OK	HEX0[0]	Location	PIN_AE26	Yes			
3	OK	HEX0[1]	Location	PIN_AE27	Yes			
4	OK	HEX0[2]	Location	PIN_AE28	Yes			
5	OK	HEX0[3]	Location	PIN_AG27	Yes			
6	OK	HEX0[4]	Location	PIN_AF28	Yes			
7	OK	HEX0[5]	Location	PIN_AG28	Yes			
8	OK	HEX0[6]	Location	PIN_AH28	Yes			
9	OK	HEX1[0]	Location	PIN_AJ29	Yes			
10	OK	HEX1[1]	Location	PIN_AH29	Yes			
11	OK	HEX1[2]	Location	PIN_AH30	Yes			
12	OK	HEX1[3]	Location	PIN_AG30	Yes			
13	OK	HEX1[4]	Location	PIN_AF29	Yes			
14	OK	HEX1[5]	Location	PIN_AF30	Yes			
15	OK	HEX1[6]	Location	PIN_AD27	Yes			
16	OK	HEX2[0]	Location	PIN_AE26	Yes			
17	OK	HEX2[1]	Location	PIN_AB23	Yes			
18	OK	HEX2[2]	Location	PIN_AE29	Yes			
19	OK	HEX2[3]	Location	PIN_AD29	Yes			
20	OK	HEX2[4]	Location	PIN_AC28	Yes			
21	OK	HEX2[5]	Location	PIN_AD30	Yes			
22	OK	HEX2[6]	Location	PIN_AC29	Yes			
23	OK	KEY[0]	Location	PIN_AA14	Yes			

*Figura No. 2.52: Asignación de pines completa.*

Para finalizar la asignación de pines se debe seguir el siguiente proceso: **File => Save.**

Luego desde la herramienta **Pin Planner**(ver figura 2.53) que se encuentra dentro del menú **Assignments** se puede observar la locación física de los pines en el dispositivo FPGA.





**Figura No. 2.53: Pin Planner.**

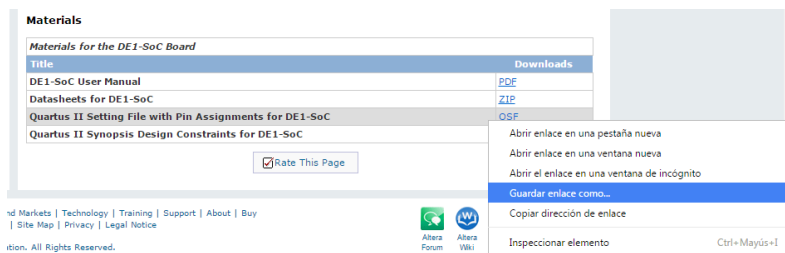
#### 2.4.4.2 Asignación por archivo de ajuste (recomendado)

Esta es la forma que se recomienda para evitar errores en la asignación de pines, ALTERA provee de archivos QSF (Quartus Setting File) para cada una de sus tarjetas, estos archivos contienen la asignación de todos los pines de la tarjeta.

Este archivo está disponible en siguiente enlace:

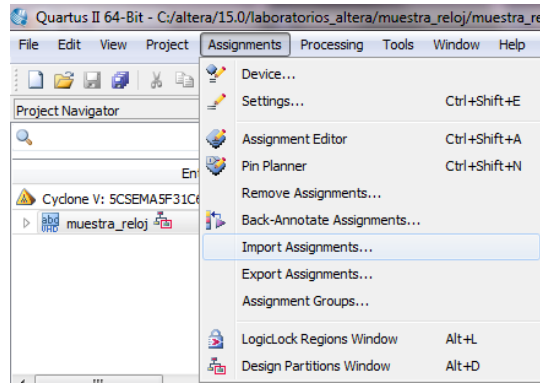
<http://wl.altera.com/education/univ/materials/boards/de1-soc/unv-de1-soc-board.html>[en línea] [ultima consulta 24/02/2016].

En la parte final de la página web se encuentran los archivos disponibles para la tarjeta DE1\_SoC, para descargar el archivo QSF se debe dar clic derecho al enlace QSF y luego seleccionar la opción **Guardar Enlace como...**, tal como se muestra en la figura 2.54.



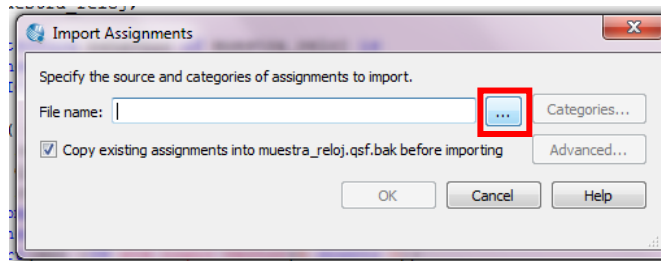
**Figura No. 2.54: Descarga de archivo QSF.**

Una vez se tiene el archivo QSF adecuado para la tarjeta se procede a insertarlo en el proyecto, para esto se debe buscar la herramienta **Import Assignments...** que se ubica en el menú **Assignments**, este procedimiento se muestra en la figura 2.55.



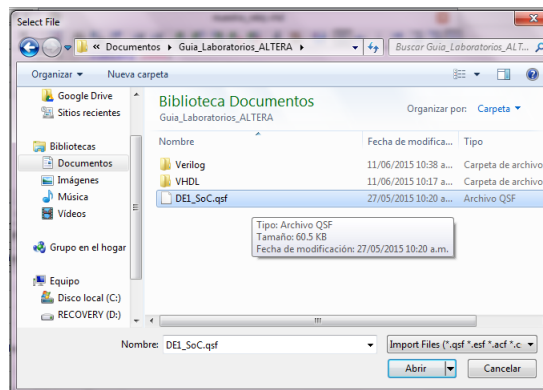
**Figura No. 2.55: Importe de ajustes.**

Se da un clic en el botón (...), ubicado en la ventana mostrada por la figura 2.56.



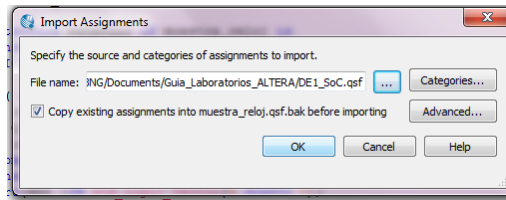
**Figura No. 2.56: importe de archivo QSF.**

Se busca el archivo en el directorio donde se descargó, tal como se muestra en la figura 2.57 y luego OK.



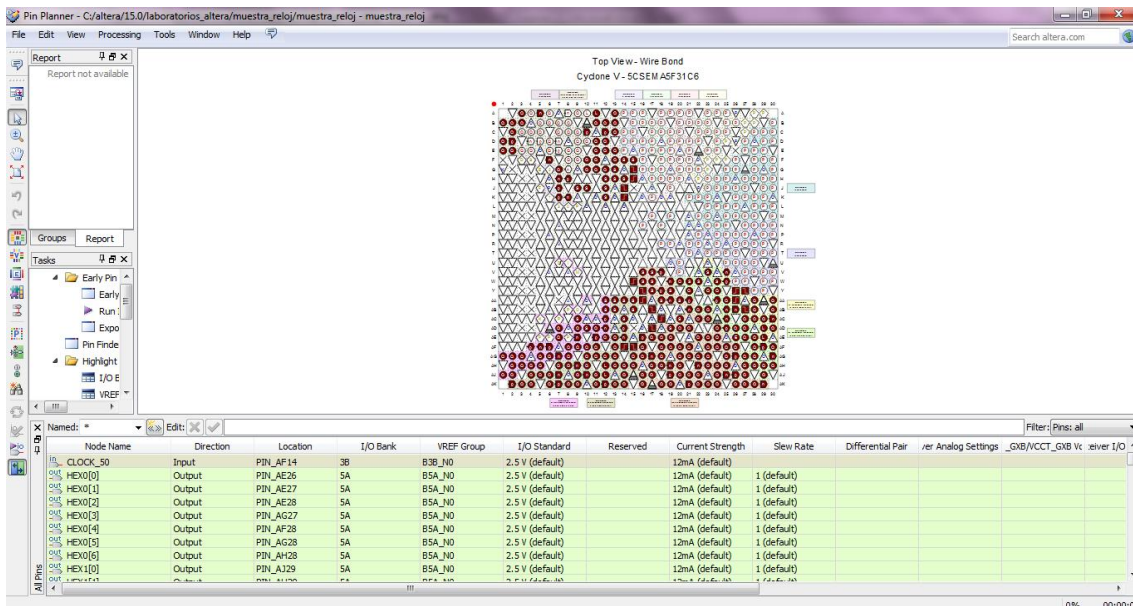
**Figura No. 2.57: Búsqueda de archivo QSF.**

Luego se regresa a la ventana anterior, pero esta vez ya tendrá la dirección del archivo, como se observa en la figura 2.58, dando clic en OK se finaliza el proceso de asignación de pines.



**Figura No. 2.58: Archivo añadido.**

Se puede verificar la asignación de pines desde el **Pin Planner** (ver figura 2.59).



**Figura No. 2.59: Pin planner método alternativo.**

## 2.4.5 Compilación

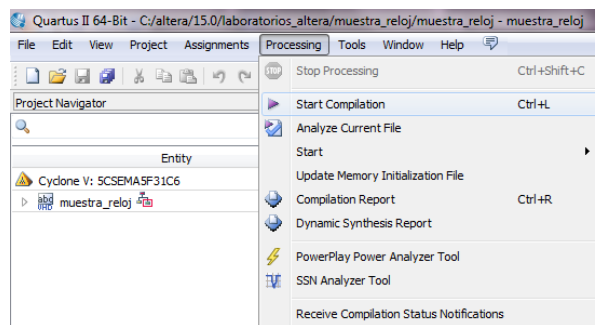
La etapa de compilación comprende 5 etapas fundamentales las cuales se describen a continuación:

- Análisis y síntesis: etapa discutida previamente.
- Fitter (place and route): esta etapa es la encargada de asignar de mejor forma los pines declarados en código con los pines físicos, para el caso de las tarjetas de desarrollo esta etapa se debe hacer de forma manual, también se encarga de designar las ALM a utilizar, así como otros componentes.
- Assembler: El código VHDL y Verilog es un equivalente de assembler de alto nivel, esta etapa se encarga de generar los archivos de salida y de programación, esta herramienta genera el archivo SOF (SRAM Object File) el cual se encarga de contener los parámetros y datos al programador para ser transmitidos al FPGA. Este tipo de archivos solo programa al FPGA, por lo tanto esta programación es de carácter volátil, si se desea una programación fija se necesita un archivo JIC (JTAG Indirect Configuration File). Este último tipo de

archivos requiere un proceso extra para generarlos. Cada uno de los modos de programación se discutirá ampliamente en otra sección de este capítulo.

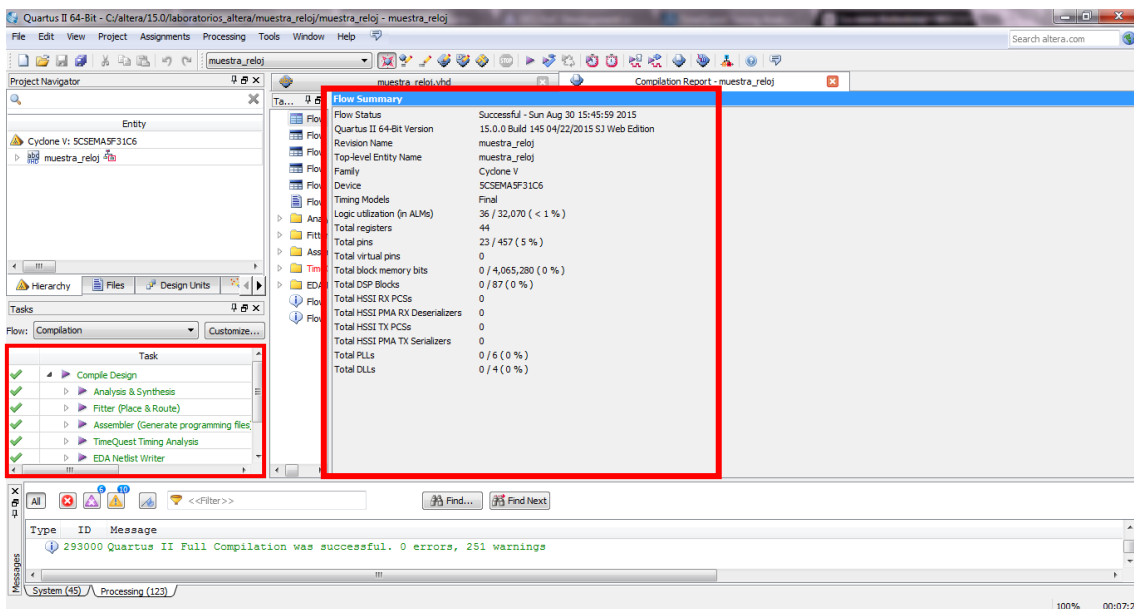
- TimeQuest Time Analysis: Esta herramienta se encarga de verificar que el circuito sea coherente si se han añadido variables que involucren tiempos o retrasos dentro del código.
- EDA Netlist Writer: Las herramientas EDA se encargan de trazar las redes de interconexión, y el mapeo interno del FPGA.

Para invocar una compilación completa es necesario ir al menú **Processing**, luego se selecciona la opción **Start Compilation**, este procedimiento se describe en la figura 2.60.



**Figura No. 2.60: Proceso de compilación.**

El compilador se activará y cada una de las etapas antes descritas actuará de forma secuencial, siempre y cuando se hayan cumplido los pasos descritos en las secciones anteriores el compilador no tendrá ningún problema, mostrando un reporte similar al de la figura 2.61.

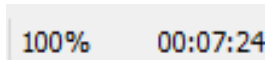


**Figura No. 2.61: Reporte final de compilación.**

Es muy probable que el compilador omita la etapa del TimeQuest ya que rara vez las variables de tiempo serán utilizadas en programas sencillos.

El tiempo de proceso de compilación varía mucho en relación al tamaño y la complejidad del código. También está estrechamente ligado a los recursos que tiene la PC. Por lo general un código de mediana complejidad tarda en 7 a 10 minutos con los recursos básicos descritos.

En la figura 2.62 se muestra un ejemplo del tiempo requerido para completar una compilación.

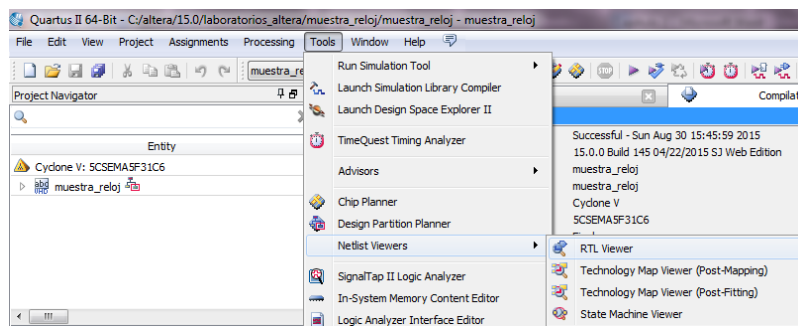


**Figura No. 2.62: Tiempo requerido para la compilación del código tutorial.**

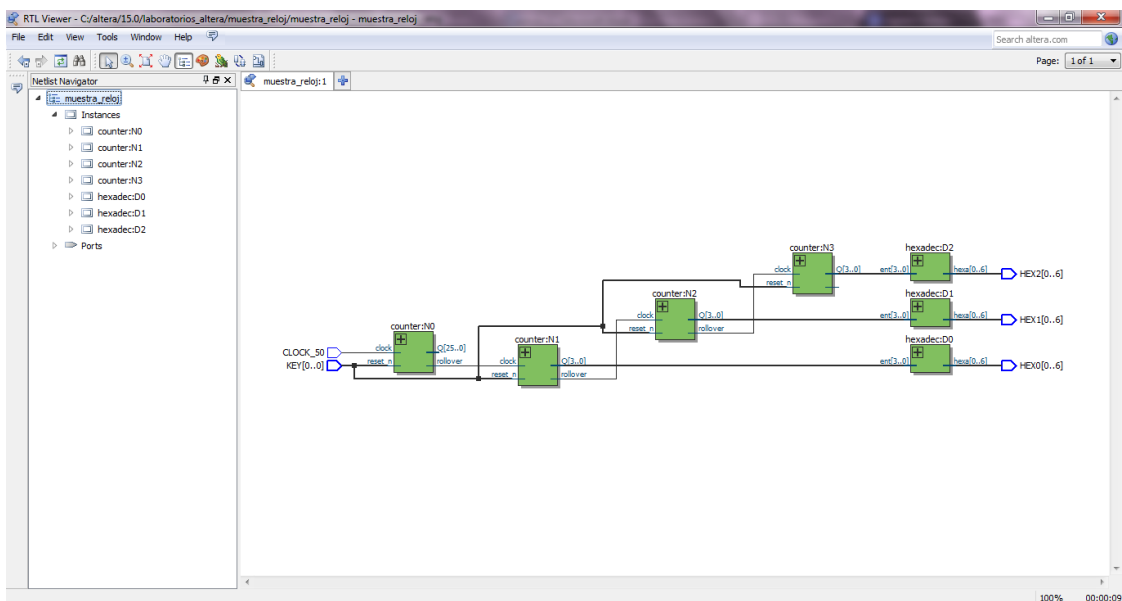
## 2.4.6 RTL Netlist Viewer y Technology Map Viewer

Estas herramientas son de carácter informativo pero tienen una gran utilidad a nivel educativo, también ayudan a realizar un buen monitoreo del FPGA.

### 2.4.6.1 RTL Viewer.



**Figura No. 2.63: Proceso para RTL Viewer.**



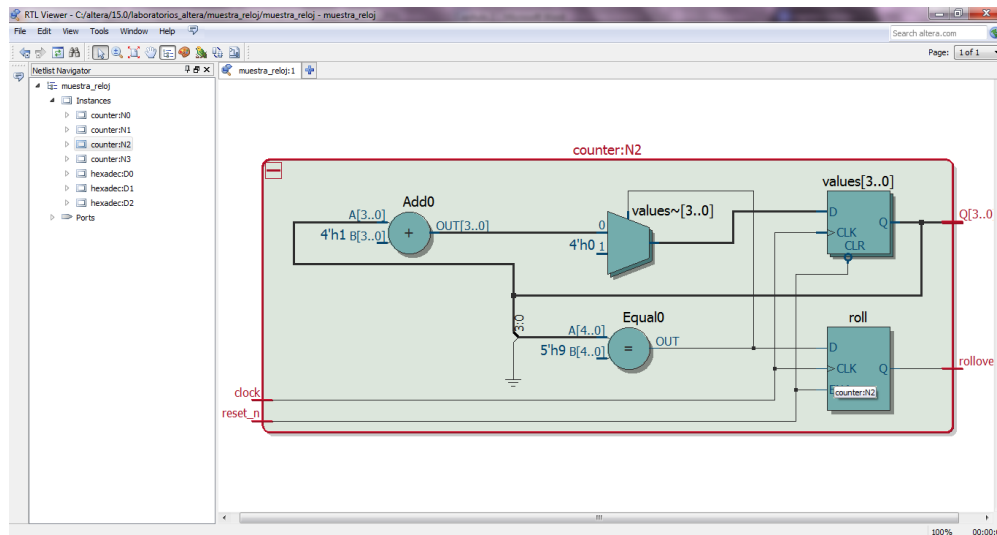
**Figura No. 2.64: Instancia principal.**

Register Transfer Level (RTL) es un tipo de archivo de verificación, en este se observan los elementos creados por las EDA para satisfacer el correcto funcionamiento del circuito.

Para acceder a este archivo se busca el menú **Tools => Netlist Viewer => RTL Viewer**, este proceso está descrito en la figura 2.63.

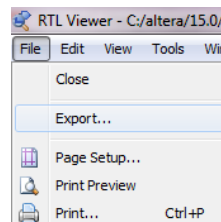
Este archivo solo estará disponible cuando la compilación haya finalizado con éxito, en la figura 2.64 se muestra el archivo generado para el código tutorial.

Cada instancia o componente del código puede ser visto a detalle, en este caso se observará el elemento Counter N2 (ver figura 2.65), para esto se da doble clic sobre el elemento.

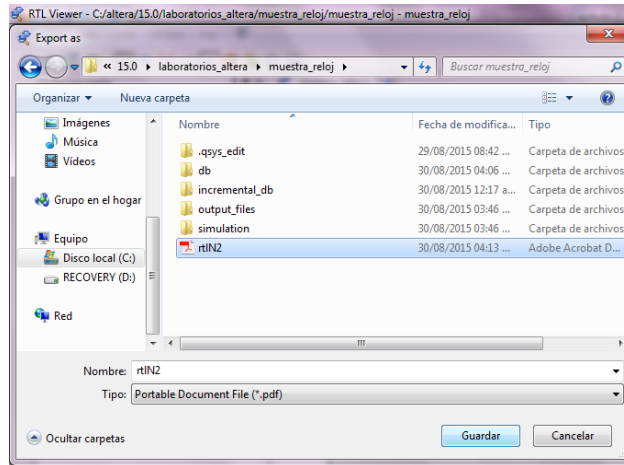


**Figura No. 2.65: Elemento Counter N2 visto a detalle.**

Este tipo de archivos puede ser exportado a formato PDF, esto se logra desde el menú **File => Export...**, este proceso se describe en la figura 2.66, luego en la figura 2.67 basta con asignar un nombre del archivo y darle clic al botón guardar para generar el PDF.



**Figura No. 2.66: Proceso para exportar archivos RTL.**

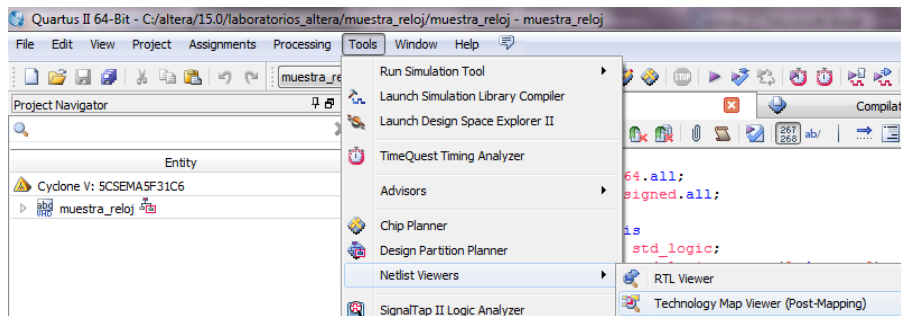


**Figura No. 2.67: Proceso para crear archivo PDF.**

#### 2.4.6.2 Technology Map Viewer

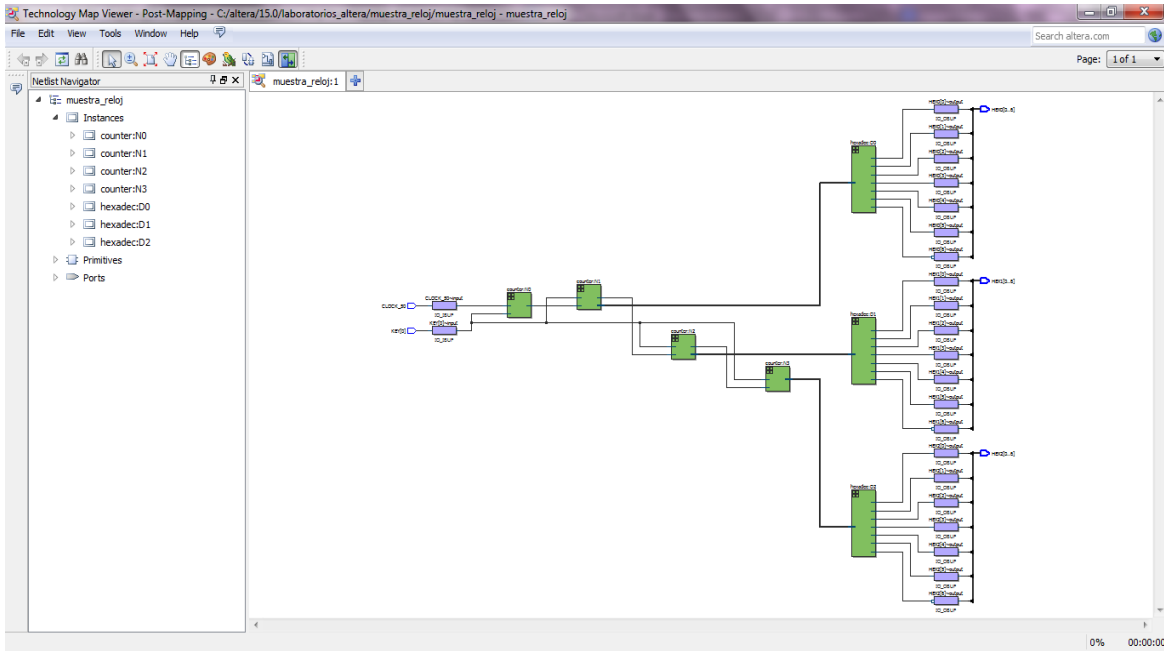
Al igual que el archivo RTL, este es un archivo de verificación que muestra en mapeo interno del FPGA basado en sus unidades básicas o ALM para el caso de la familia Cyclone V SoC.

Para acceder a este archivo se busca el menú **Tools => Netlist Viewer => Technology Map Viewer**, este proceso se describe en la figura 2.68.



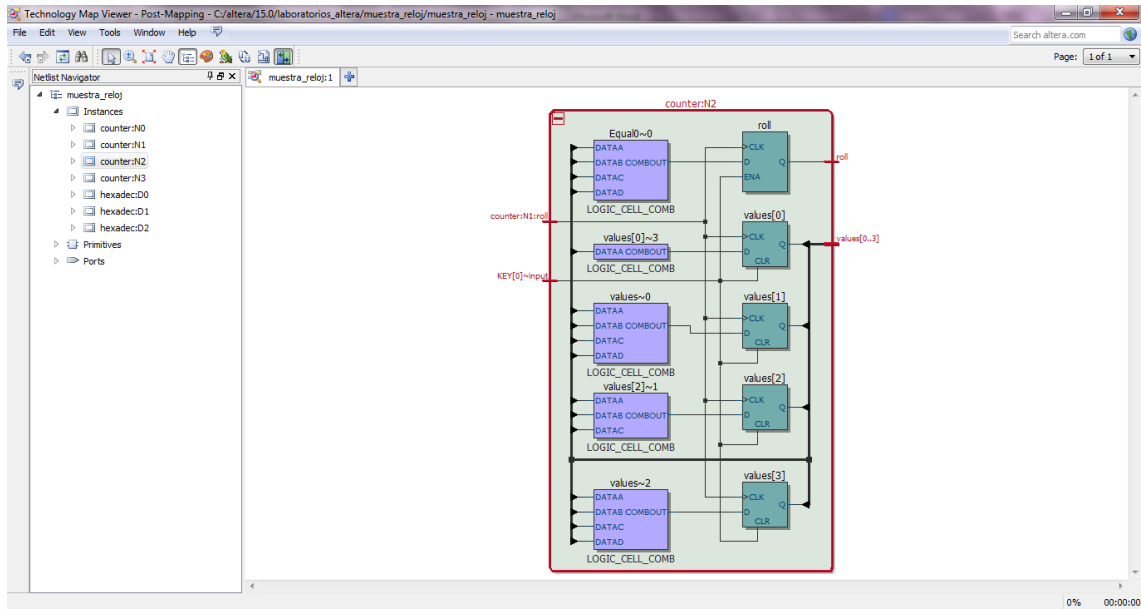
**Figura No. 2.68: Proceso para Technology Map Viewer.**

Este archivo solo estará disponible cuando la compilación haya finalizado con éxito, la figura 2.69 muestra el archivo generado para la instancia principal.



**Figura No. 2.69: Instancia principal.**

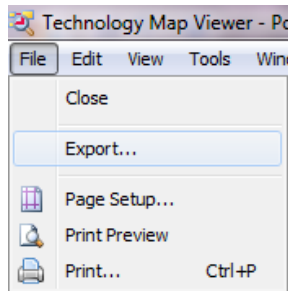
Cada instancia o componente del código puede ser visto a detalle, en este caso se observará el elemento Counter N2 (ver figura 2.70), para esto se da doble clic sobre el elemento.



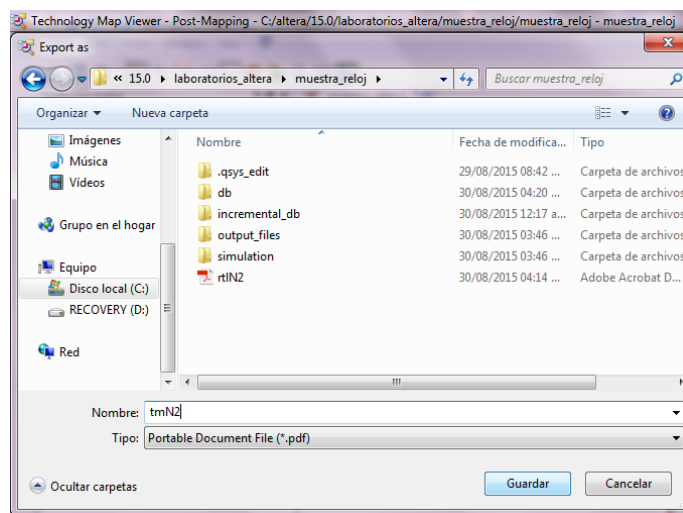
**Figura No. 2.70: Elemento Counter N2 visto a detalle.**



Este tipo de archivos puede ser exportados a formato PDF, esto se logra desde el menú **File => Export...**, este procedimiento se describe en la figura 2.71. En la ventana mostrada por la figura 2.72 basta con asignar un nombre al archivo y dar clic al botón guardar para generar el PDF.



**Figura No. 2.71: Proceso para exportar archivos Technology Map.**



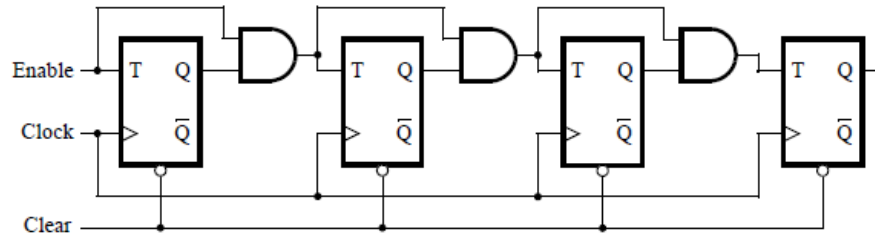
**Figura No. 2.72: Proceso para crear archivo PDF.**

### 2.4.7 Simulación

La simulación que permite Quartus II es a través de diagramas de tiempo, una forma nada despreciable de entender el comportamiento de un circuito lógico.

Este proceso puede ser llevado a cabo por medio del archivo University Program VWF, este archivo se puede generar luego de sintetizar el código.

Para realizar el tutorial de simulación se optó por un código más simple correspondiente a un contador de 8 bits formado por FF's T y con salida a dos display de 7 segmentos.

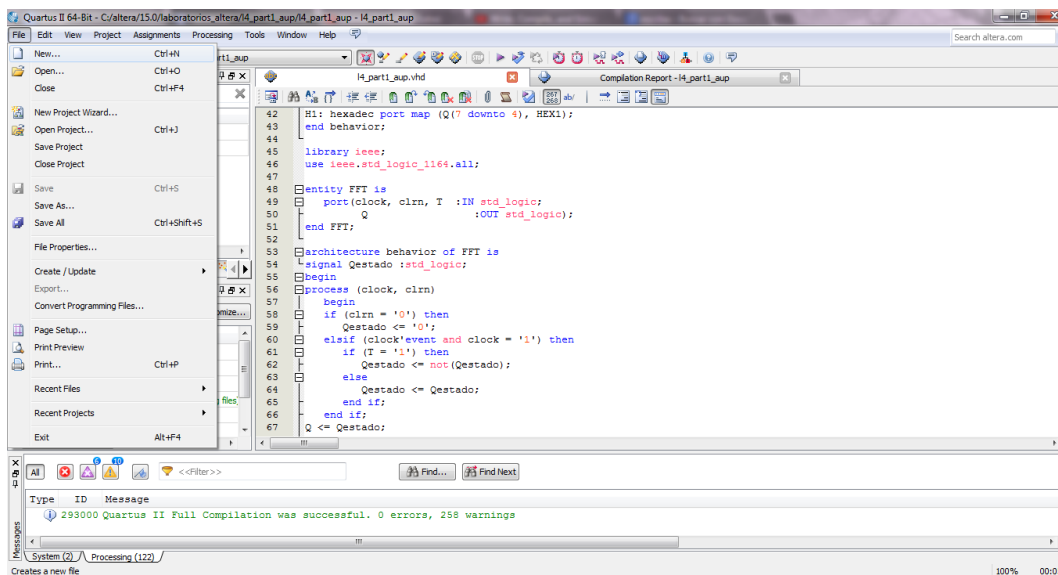


**Figura No. 2.73: Contador de 4 bits basado en FF's T**

La figura 2.73 muestra un contador como el que se requiere en el ejemplo con la variante de que este diseño es para 4 bits.

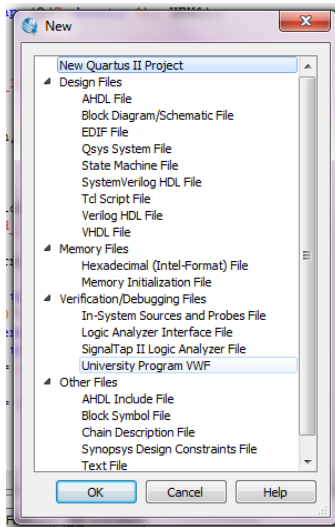
El código y su compilación no son de interés en esta sección de forma que solo se abarca el proceso de simulación.

El primer paso es crear un archivo VWF(Verification Waveform File), para lograrlo se debe acceder al menú **File => New...** este procedimiento se describe en la figura 2.74.



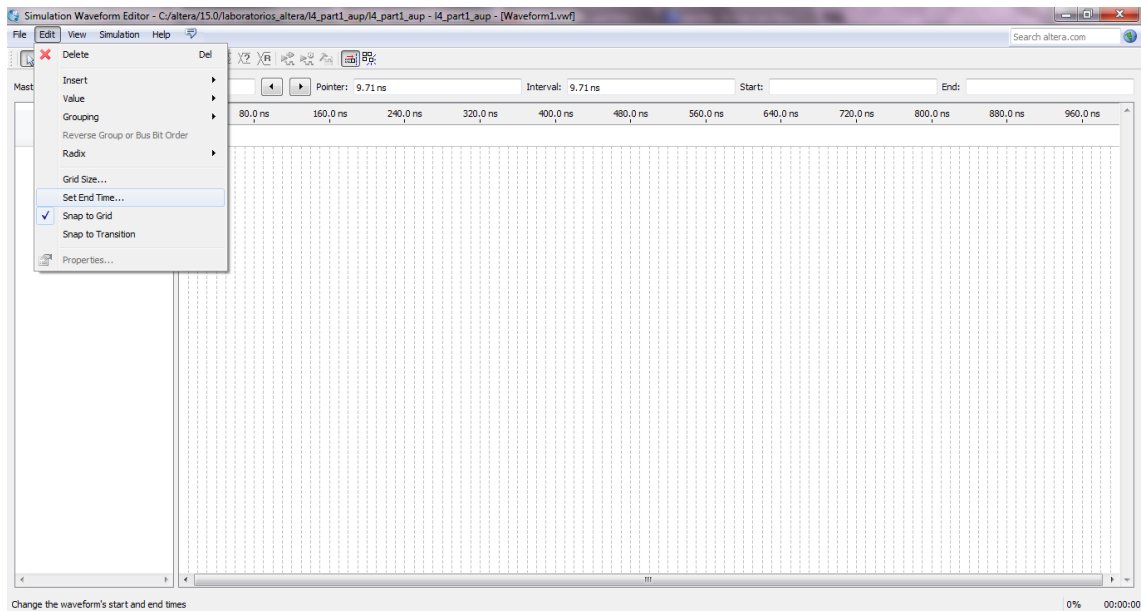
**Figura No. 2.74: Proceso para crear un archivo VWF.**

Luego abrirá el selector de archivos como el que se muestra en la figura 2.75, el tipo VWF se encuentra dentro de la sección **Verification/debugging Files**, aquí se debe seleccionar el tipo **University Program VWF**.



**Figura No. 2.75: Selección de archivo VWF.**

Esto abrirá la interfaz del programa (ver figura 2.76), luego se debe seleccionar un tiempo base de simulación, esto se logra en la opción **Set End Time...** que se encuentra en el menú **Edit**.

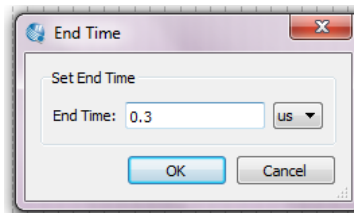


**Figura No. 2.76: Selección de tiempo de simulación.**

Para el ejemplo se consideraron 0.3 us, estos deben ser establecidos como se muestra en la figura 2.77.

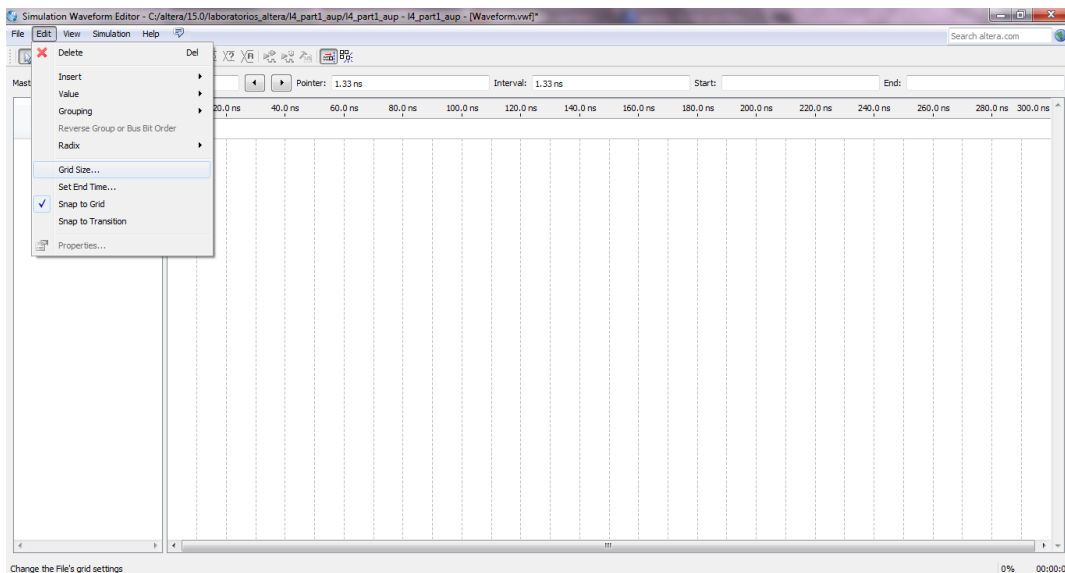
Este número no es arbitrario ya que por defecto las divisiones de tiempo están en 10 ns, considerando ese factor cada división equivaldrá a una combinación posible, en el ejemplo de

muestra el contador tiene 8 salidas lo que provoca 256 combinaciones posibles, cada una de 10 ns, una multiplicación sencilla nos da el resultado de 2560 ns para ver todas las combinaciones posibles.



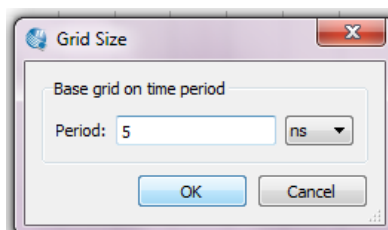
**Figura No. 2.77: Tiempo de simulación en us.**

El programa también permite modificar el tamaño de la separación de los intervalos de tiempo, para modificarlo se debe acceder el menú **Edit** y luego **Grid size...**, tal como se describe en la figura 2.78.



**Figura No. 2.78: Selección del tamaño de la Grid.**

Para el ejemplo se considera un tiempo de 5 ns segundos por intervalo, y debe ser establecido como se muestra en la figura 2.79.

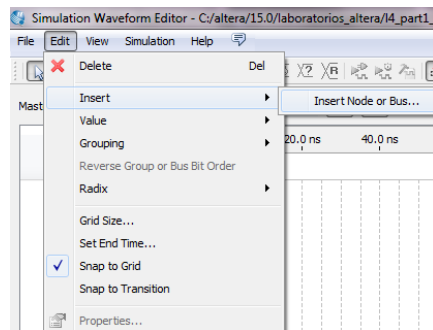


**Figura No. 2.79: Tamaño de los intervalos.**

Este valor de igual forma no es aleatorio, las señales pueden ser modificadas acorde al tamaño del intervalo de tiempo, de esta forma si se requiere que un periodo de reloj sea de 10 ns, el pulso de subida debe ser la mitad de ese tiempo, para que el pulso de bajada constituya la otra mitad, es por eso que el intervalo debe ser 5 ns.

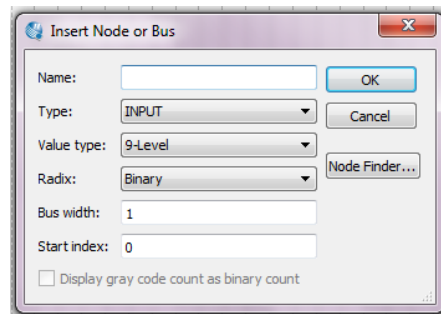
El siguiente paso es insertar los pines correspondientes a la simulación, en este caso se utilizan los elementos SW(0) y SW(1) para representar Reset y Enable respectivamente, la señal de reloj está representada por el elemento KEY(0) y los display HEX0 y HEX1 mostraran las 8 salidas.

Para acceder a los pines se debe buscar en el menú **Edit** la opción **Insert**, en el sub menú desplegado se debe buscar la opción **Insert node o bus...**, tal como se muestra en la figura 2.80.



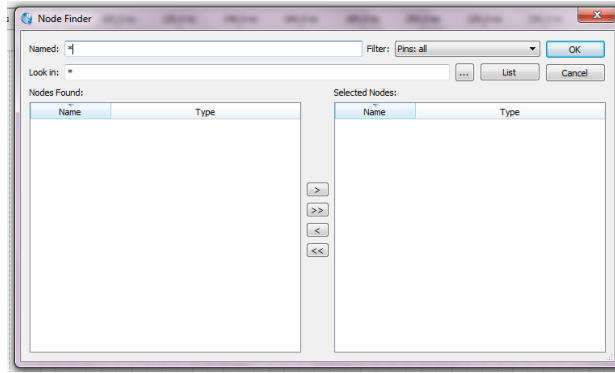
**Figura No. 2.80: Selección de nodos (I).**

La figura 2.81 muestra la ventana en la cual se debe buscar el botón **Node Finder...** y darle clic.



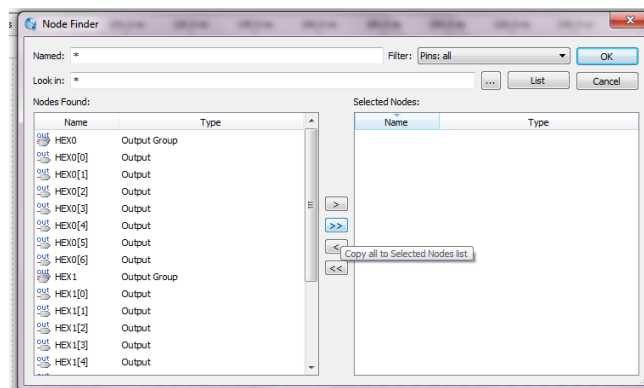
**Figura No. 2.81: Selección de nodos (II).**

Se desplegará una nueva ventana (ver figura 2.82) en la cual se deben considerar dos opciones, en el espacio **Filter** debe estar seleccionada la opción **Pins: All**, luego se debe dar clic en list para desplegar los nodos disponibles.



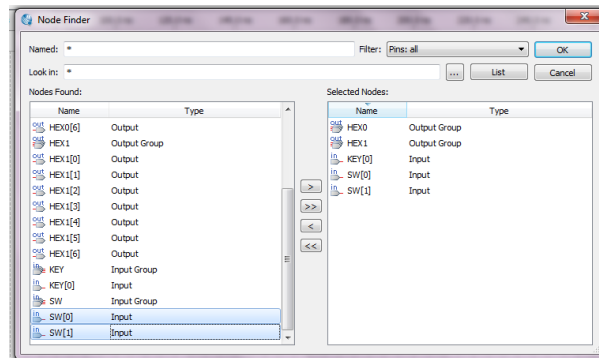
**Figura No. 2.82: Selección de nodos (III).**

Una vez se tiene la lista disponible, como se observa en la figura 2.83, se procede a seleccionar los pines que se desean ver, para esto se debe resaltar el pin o grupo de pines deseados y luego apretar el botón >



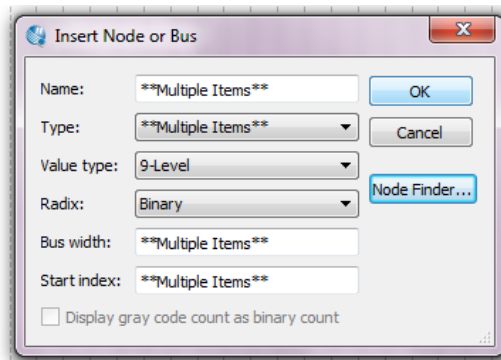
**Figura No. 2.83: Lista de nodos disponibles.**

Para el ejemplo los pines de HEX0 y HEX1 se toman como grupos, los demás nodos se seleccionan de forma individual, debe quedar una lista de nodos seleccionados similar a la que muestra la figura 2.84, luego clic en **OK**.



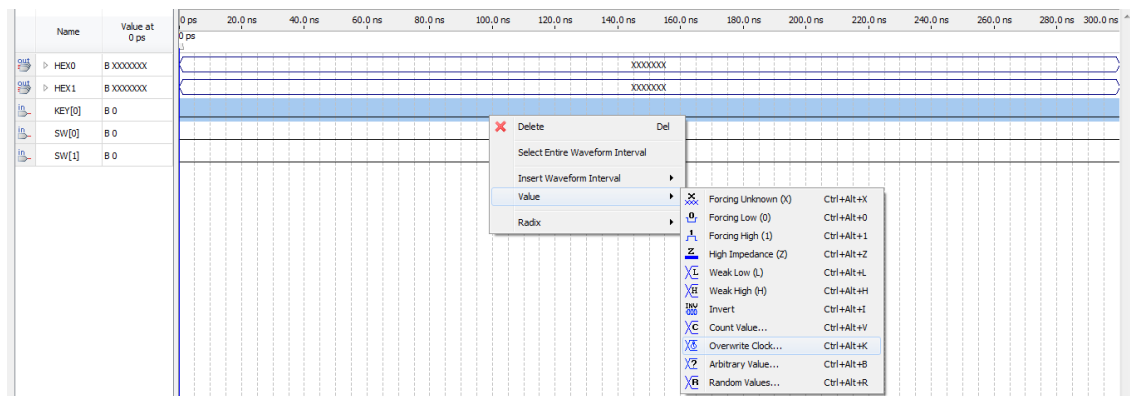
**Figura No. 2.84: Lista de nodos seleccionados.**

Luego de realizado ese proceso debe aparecer la casilla **Name** rellena por la expresión **\*Multiple Items\***, tal como se muestra en la figura 2.85, luego basta un OK para que aparezcan en la interfaz principal.



**Figura No. 2.85: Selección de nodos (IV).**

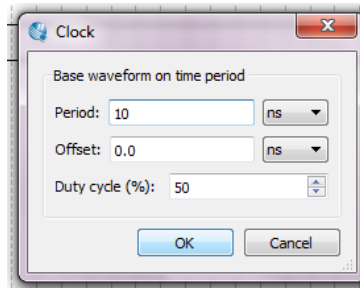
Luego de seleccionados los pines se procede a asignarles valor a las señales, se iniciará con el elemento KEY(0), el cual debe tener una señal de reloj, para lograrlo se debe dar doble clic sobre la señal, esto provocará que se seleccione todo el intervalo disponible y luego con clic derecho sobre la selección se busca la opción **Value => Overwrite Clock**, este proceso se describe en la figura 2.86.



**Figura No. 2.86: asignación de señales.**

Para el ejemplo se debe asignar un valor de 10 ns al periodo, este valor es criterio de usuario, se suele utilizar los 10 ns porque los intervalos de la malla vienen divididos en esa fracción.

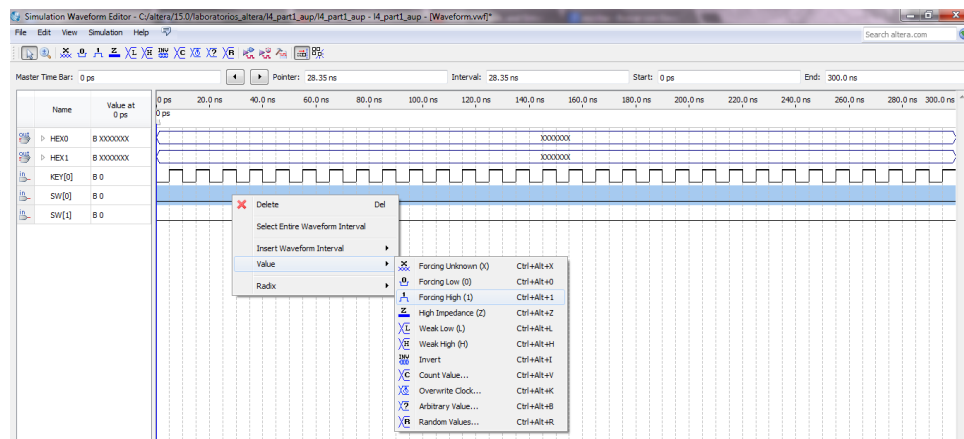
Si este periodo se modifica también deben modificarse el tiempo de muestreo y el intervalo de la malla, para una mejor visualización.



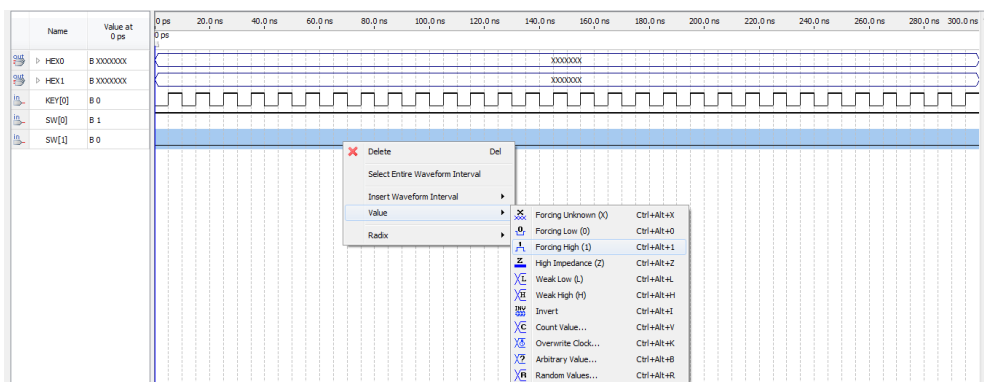
**Figura No. 2.87: Asignación del periodo de reloj.**

Para que el reloj funcione las señales de SW(0) y SW(1) deben ser iguales a 1 por lo tanto se explicará la forma de asignación para una variable, siendo un proceso similar para la otra variable.

Al igual que en el caso anterior se debe dar doble clic sobre la señal para seleccionar el intervalo completo, luego clic derecho sobre la selección y buscar la opción **Value => Forcing High**, este procedimiento se muestra en las figuras 2.88 y 2.89, para las señales SW(0) y SW(1) respectivamente.



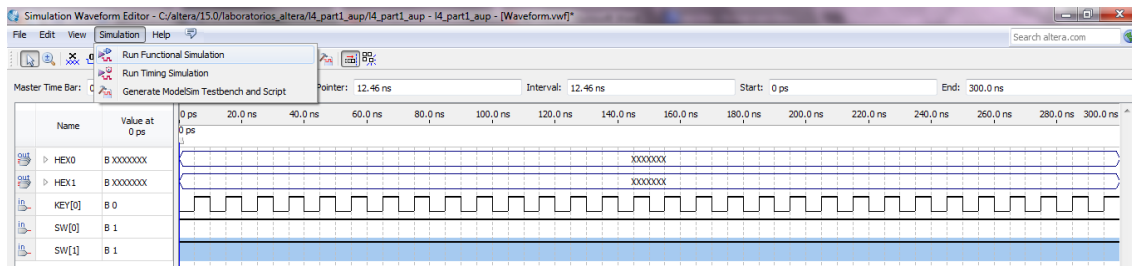
**Figura No. 2.88: Asignación para SW(0).**



**Figura No. 2.89: Asignación para SW(1).**

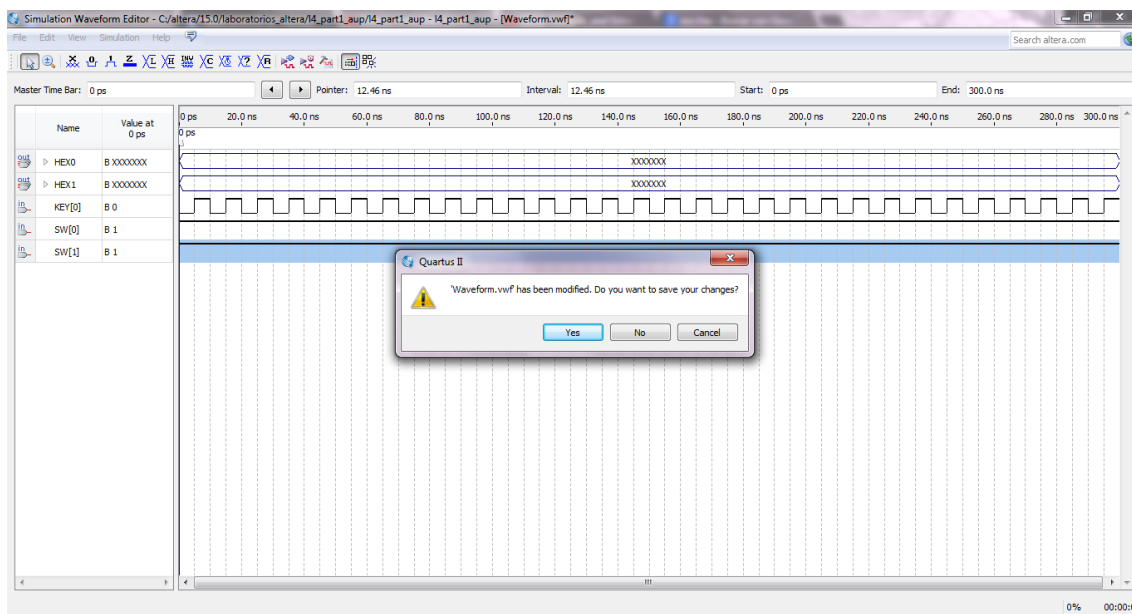


Una vez finalizada la asignación el diagrama de tiempo debe verse como en la figura 89, luego en el menú **Simulation** se debe buscar la opción **Run Functional Simulation** tal como se muestra en la figura 2.90, para iniciar el proceso de simulación.



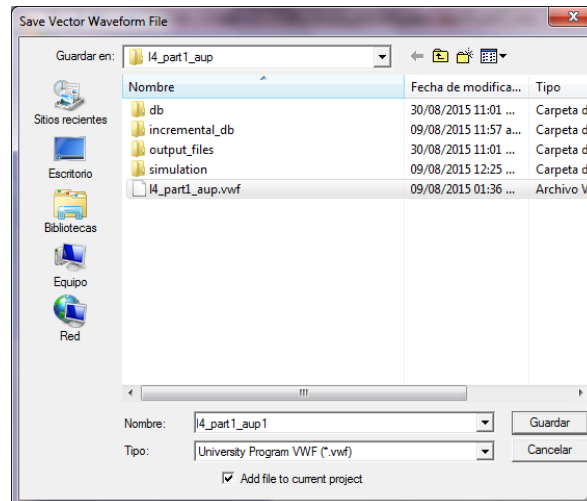
**Figura No. 2.90: Diagrama de tiempo con asignaciones.**

El programa pedirá guardar los cambios efectuados al archivo antes de realizar la simulación, lanzará un mensaje como el de la figura 2.91.



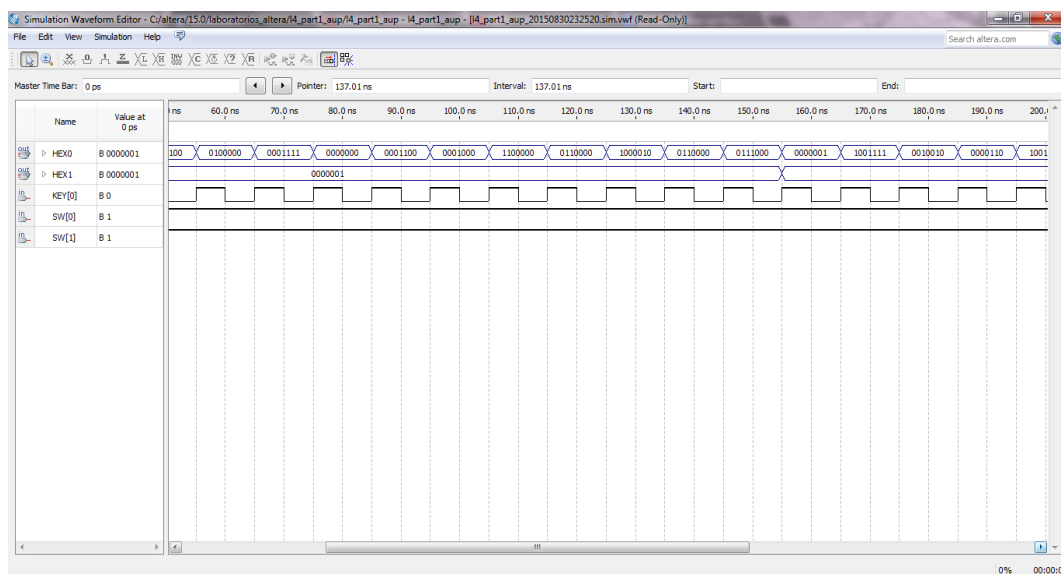
**Figura No. 2.91: Guardar archivo de simulación.**

Se debe buscar un directorio apropiado para guardar el archivo, en la figura 2.92 se observa que para este caso particular se opto por lo conveniente y el archivo se guardo en la misma dirección del proyecto en general.



**Figura No. 2.92: Directorio del archivo.**

Una vez finalizada la simulación el programa lanzará una nueva ventana con el diagrama de tiempo completo tal como se muestra en la figura 2.93.



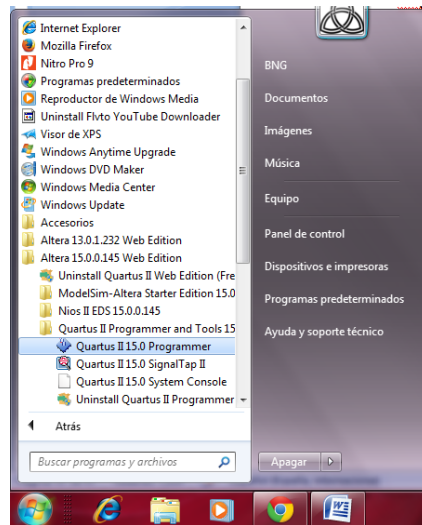
**Figura No. 2.93: Simulación completa.**

## 2.5 Programmer

Esta herramienta tiene peculiaridades que merecen ser discutidas en una sección independiente.

El Programmer es el encargado de crear una interfaz entre los archivos generados y la tarjeta física, esto lo hace a través del dispositivo USB Blaster II, a pesar de considerarse una herramienta del programa principal, este tiene la suficiente independencia como para funcionar bajo su propia cuenta, siempre y cuando los drivers del USB Blaster II estén correctamente instalados.

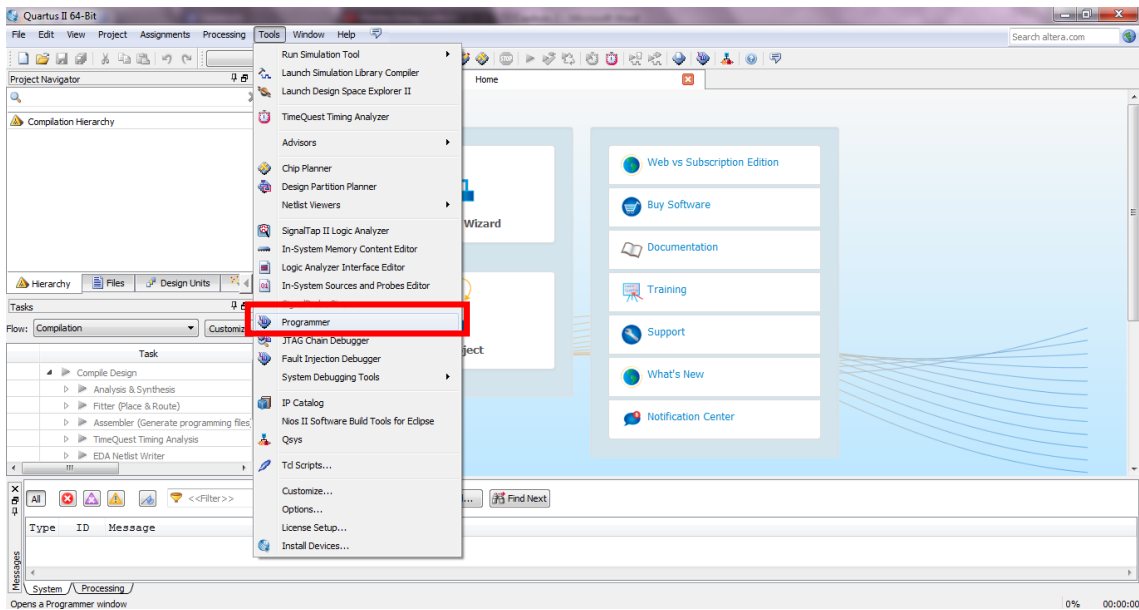
Para invocar al programa se tienen dos opciones, ambas son recomendables.



**Figura No. 2.94: Método I Programmer Tool.**

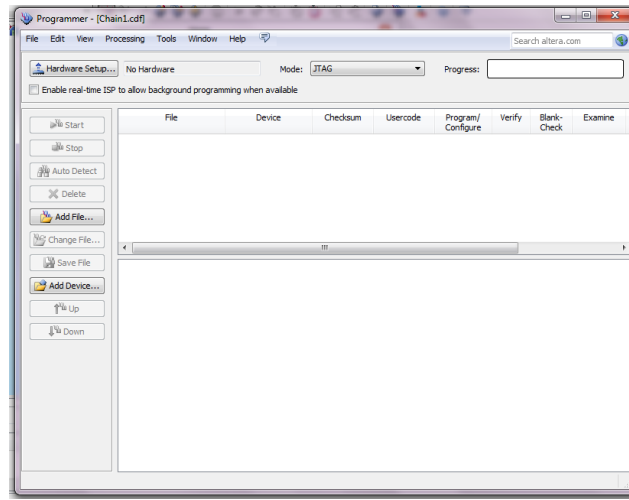
El primer método requiere realizar el siguiente procedimiento: Inicio => Todos los programas => Altera 15.0.0.145 Web Edition => Quartus II Programmer and Tools 15.0.0.145 => Quartus II 15.0 Programmer, este procedimiento se describe en la figura 2.94

El otro método se consigue desde el interior de Quartus II, para acceder a este se debe buscar en el menú **Tools** la opción **Programmer** (ver figura 2.95).



**Figura No. 2.95: Método II Programmer Tool.**

Desde cualquiera de los métodos se accede a la interfaz principal del Programmer (Ver figura 2.96).



**Figura No. 2.96: Interfaz principal de Programmer Tool.**

### 2.5.1 Programar archivos SOF

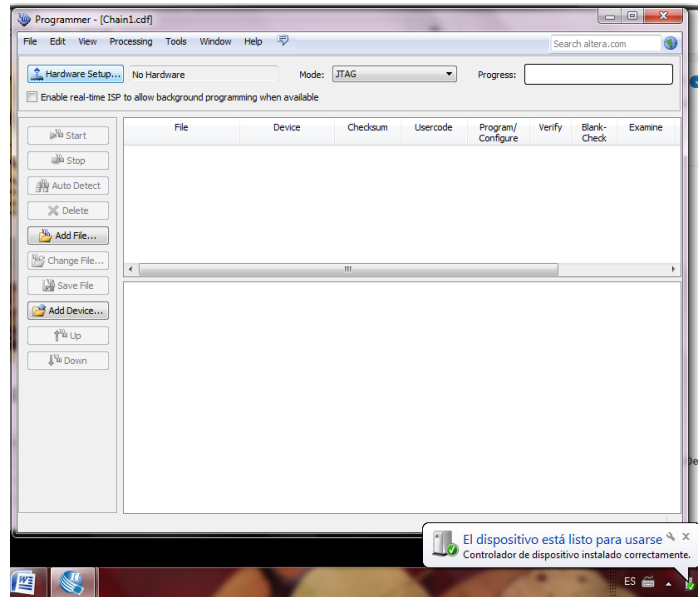
Este tipo de programación es de tipo volátil, por lo tanto al apagar el equipo, toda configuración guardada se borrará.

Los archivos SOF son ideales para la depuración de circuitos ya que permiten observar el comportamiento del código dentro de la tarjeta sin forzar el equipo, sumado a eso, la programación es más rápida.

Es el mismo compilador de Quartus II que genera los archivos SOF en la etapa Assembler, estos archivos están disponibles en la subcarpeta Output\_files, que se encuentra dentro de cada carpeta de proyecto.

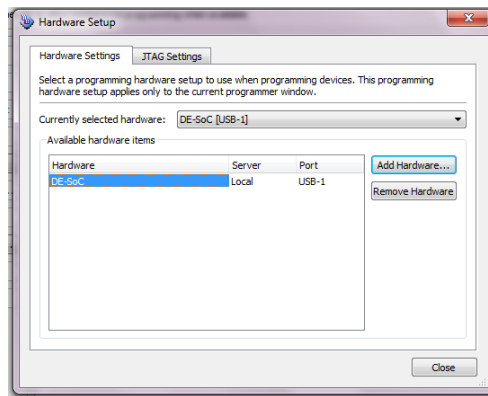
Para transferir archivos SOF a la tarjeta se debe invocar el Programmer, la tarjeta debe estar conectada a la fuente alimentadora y el cable USB type B debe estar conectado en un extremo a la PC y en el otro a la tarjeta DE1-SoC.

Al encender la tarjeta, si todas las conexiones están bien y los drivers están instalados, el administrador de dispositivos lanzará un mensaje indicando que el dispositivo está listo (ver figura 2.97).



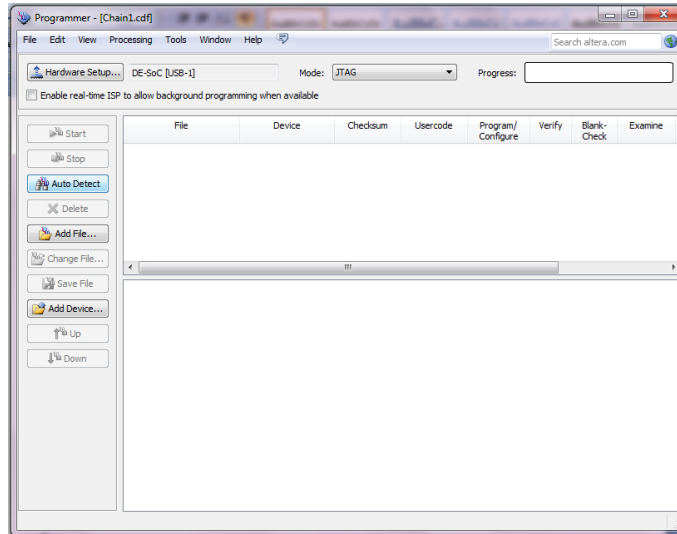
**Figura No. 2.97: Dispositivo USB Blaster II configurado exitosamente.**

Al dar clic en Hardware Setup despliega la ventana mostrada en la figura 2.98, aquí se debe seleccionar el hardware DE1-SoC y asegurarse que la conexión se haga a través de un puerto USB.

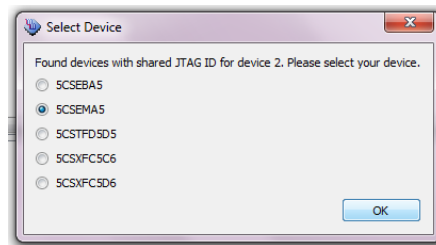


**Figura No. 2.98: Hardware Setup.**

Luego de configurado el dispositivo de transferencia, se debe dar clic en **Auto Detect**, la figura 2.99 muestra el botón resaltado.



**Figura No. 2.99: Auto Detect.**



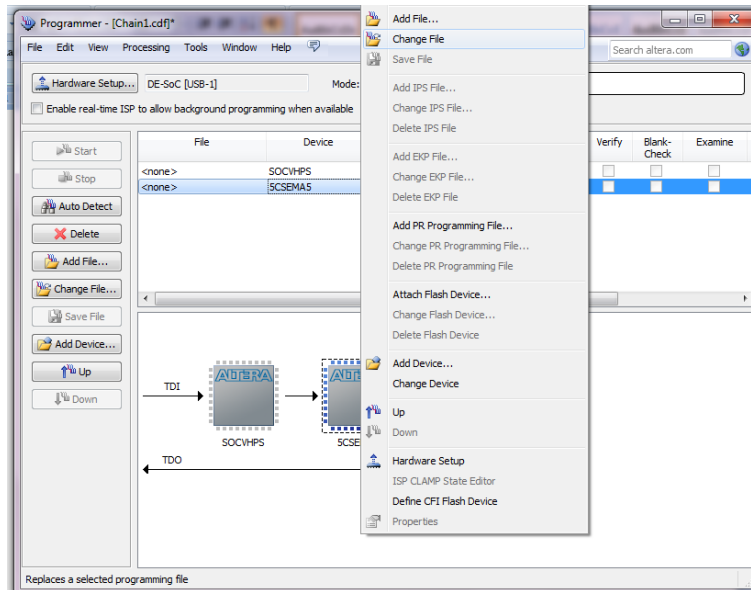
**Figura No. 2.100: Selección de dispositivo.**

La función Auto Detect reconocerá una posible lista de elementos acordes a las características leídas en el chip, ya que los dispositivos de una familia son muy parecidos, el programa pedirá identificar el dispositivo que se está utilizando, como se muestra en la figura 2.100 se debe seleccionar el dispositivo **SCSEMA5**.

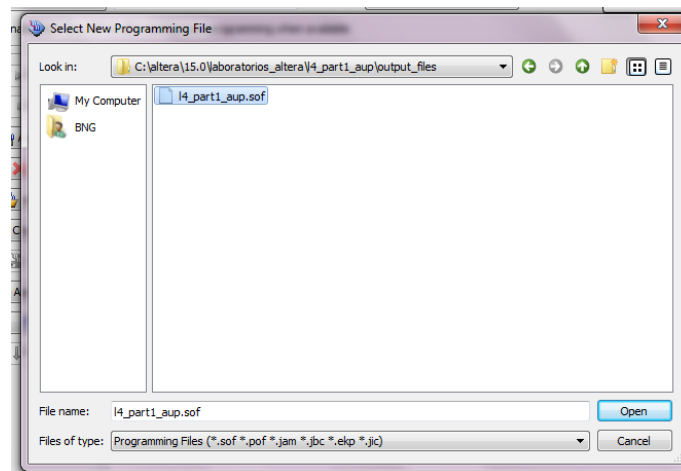
Una vez seleccionado el dispositivo en la interfaz principal del Programmer aparecerán dos dispositivos, uno corresponde al FPGA, mientras que el otro corresponde a la EPCS.

En la figura 2.101 se muestra el proceso para cambiar un archivo de configuración en el FPGA, basta con darle clic derecho sobre el dispositivo FPGA y luego seleccionar la opción **Change File...**

Esto desplegará una ventana de búsqueda de directorios, para el ejemplo se ha tomado un archivo SOF previamente compilado, la ruta de acceso se muestra en la figura 2.102, tal como se había mencionado con anterioridad los archivos SOF se encuentran dentro de la carpeta Output\_files correspondientes a cada proyecto.



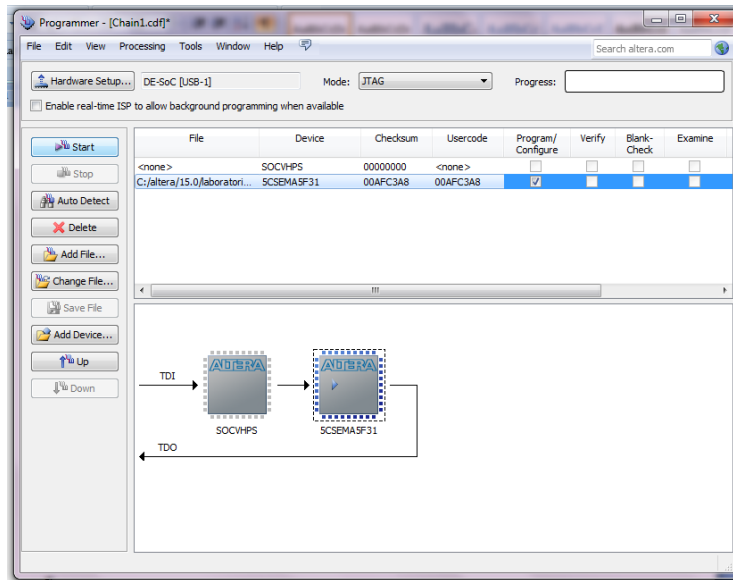
**Figura No. 2.101: Agregar archivo para ser programado en FPGA.**



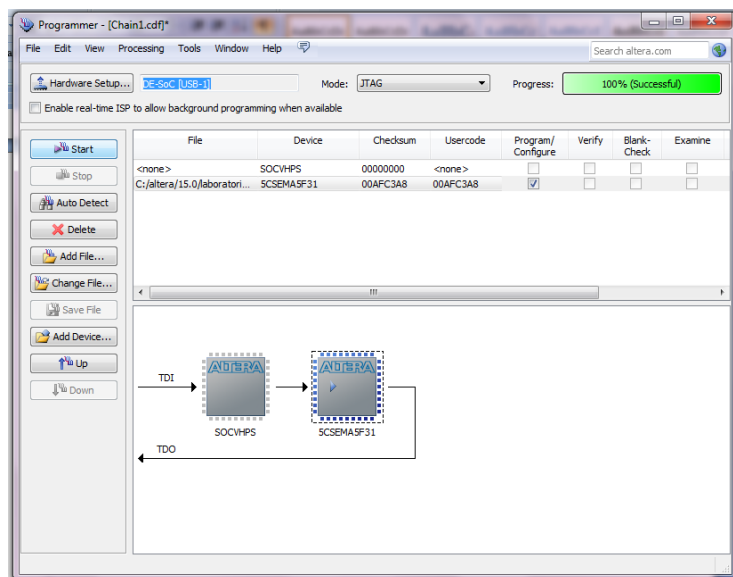
**Figura No. 2.102: Directorio de archivo a programar.**

Una vez seleccionado el archivo se procede a marcar la opción de **Program/Configure** tal como se muestra en la figura 2.103, paso siguiente se da clic en **Start** y el programa se descargará.

En la figura 2.104 se muestra el proceso de programación completo, cuando la barra indique un 100% la tarjeta esta lista para su uso, se debe recordar que este tipo de programación es volátil por lo tanto las pruebas se hacen sin apagar el equipo, de lo contrario se debe comenzar el proceso desde el inicio de esta sección.



**Figura No. 2.103: Program/Configure.**



**Figura No. 2.104: Proceso finalizado.**

## 2.5.2 Programar archivo JIC

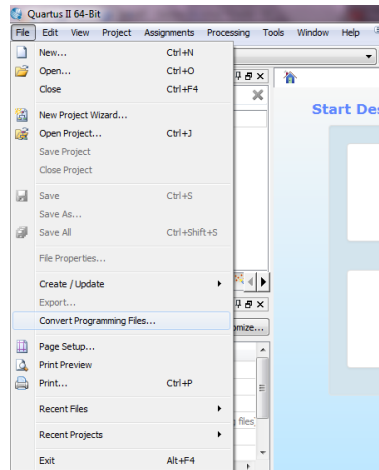
Este tipo de archivos se guarda en el dispositivo EPCS128, que es una memoria de tipo flash de 128 Mb, esta se utiliza como un configurador de tipo serial para el FPGA de forma que el chip al iniciar extrae los datos de la memoria y corre las funciones que estén almacenadas dentro de esta, a menos que se indique lo contrario.



Antes de continuar se debe tener la precaución que el dipswitch de configuración tenga los siguientes parámetros MSEL[4..0] = '10010' de forma que el flash loader sea accesible.

Habiendo revisado ese detalle, se procede a crear un archivo JIC a partir de un archivo SOF.

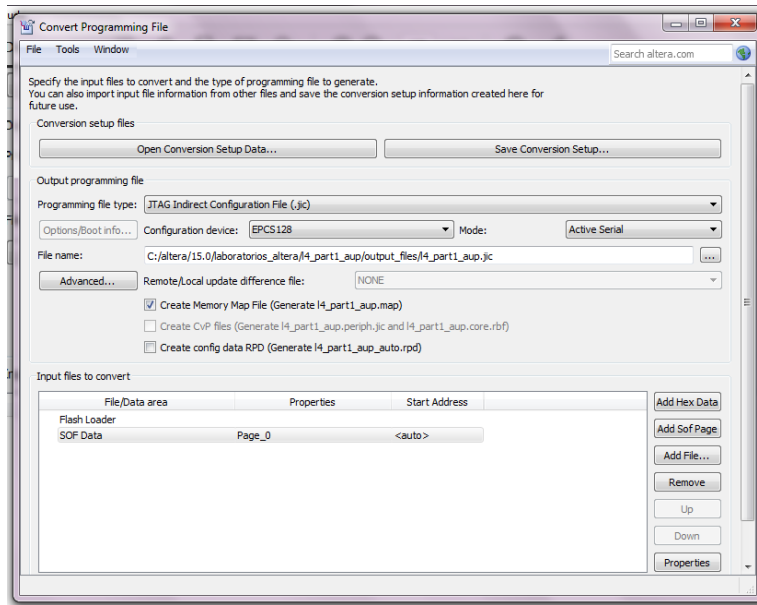
El primer paso es acceder al menú **File** del Quartus II, luego seleccionar la opción **Convert Programming Files...**, este proceso se describe en la figura 2.105.



**Figura No. 2.105: Procedimiento para conversión de archivos.**

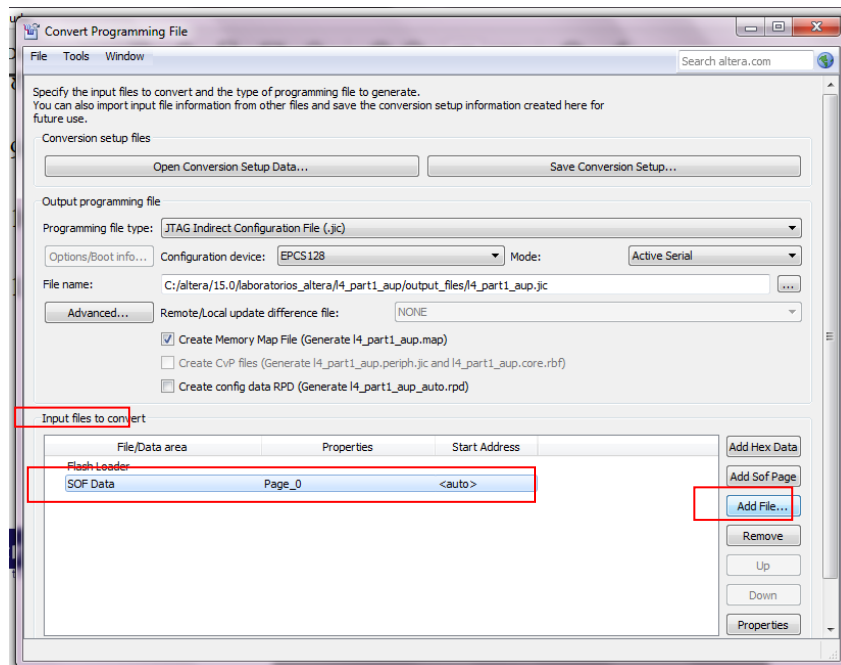
En la figura 2.106 se muestra la ventana principal del convertidor de archivos, en donde se deben establecer los siguientes parámetros para garantizar que el archivo a crear sea el indicado:

- Programming file type: JTAG Indirect Configuration File (.jic)
- Configuration Device: EPCS128
- Mode: Active Serial
- File Name: Nombre y directorio donde se guardara el archive resultante.



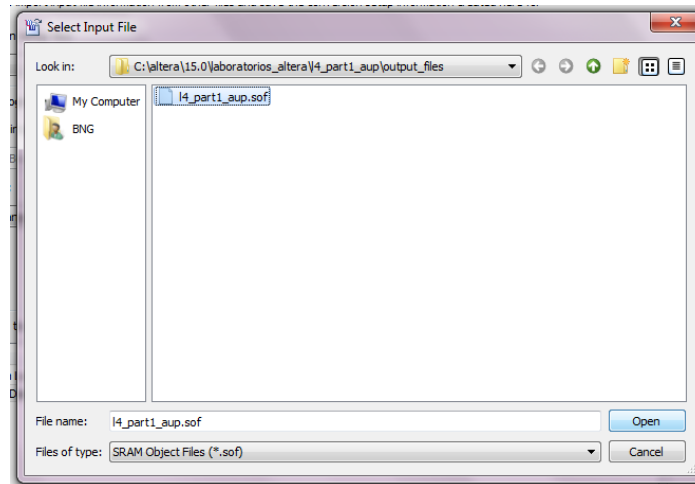
**Figura No. 2.106: Parámetros de archivo JIC.**

Se debe dar clic sobre **SOF Data** ubicado dentro de la columna **Input File to convert**, luego se debe dar clic sobre **Add File**, en la figura 2.107 se enmarcan cada uno de los elementos.



**Figura No. 2.107: Selección de Archivo SOF.**

Se debe buscar el archivo SOF a convertir tal como lo muestra la figura 2.108.



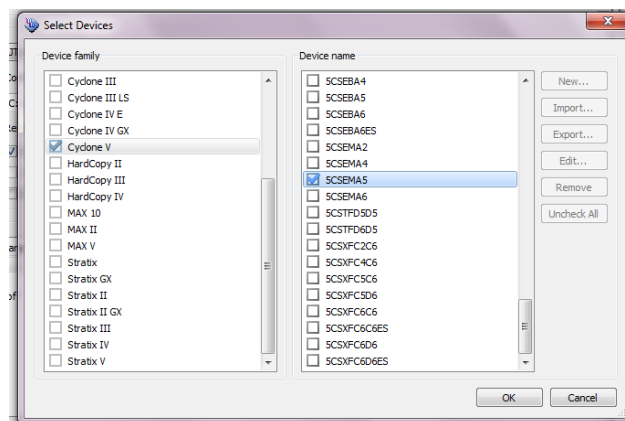
**Figura No. 2.108: Directorio archivo SOF.**

Una vez seleccionado el archivo SOF, se debe seleccionar el dispositivo objetivo, en este caso el FPGA de la familia Cyclone V.

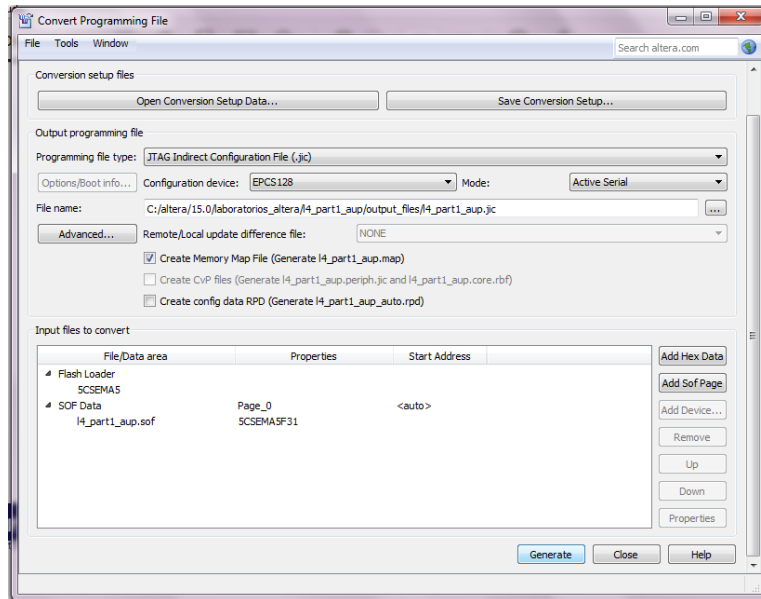
Para realizar este procedimiento se debe dar clic en **Flash Loader** y luego **Add Device...**

Esto invocará una ventana como la que se muestra en la figura 2.109, dentro de esta se debe seleccionar la familia Cyclone V y luego buscar el dispositivo **5CSEMA5**, luego dar clic en OK.

Antes de pulsar el botón **Generate** la ventana principal debe verse como en la figura 2.110, si todos los parámetros están completos se debe dar clic en el botón para generar el archivo JIC.

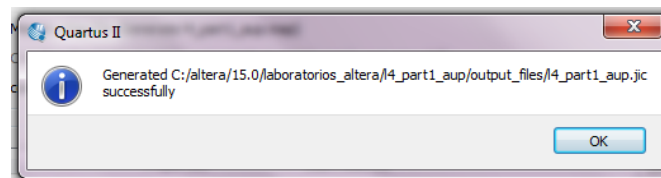


**Figura No. 2.109: Selector de dispositivos.**



**Figura No. 2.110: Ventana con parámetros completos.**

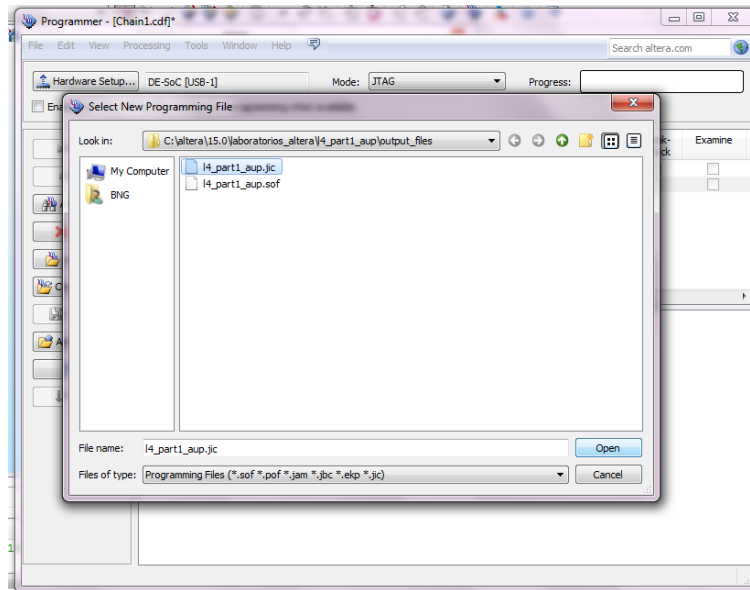
Una vez el programa finalice el proceso enviara un mensaje como el de la figura 2.111, advirtiendo que el archivo JIC ha sido generado.



**Figura No. 2.111: Proceso de generación de archivo JIC exitoso.**

Paso seguido se debe abrir el programador, revisar que la conexión de la tarjeta sea adecuada, seleccionar el tipo de hardware correcto, invocar la auto detección de dispositivo y cerciorarse que el chip destino sea el indicado, este paso es similar al realizado con los archivos SOF.

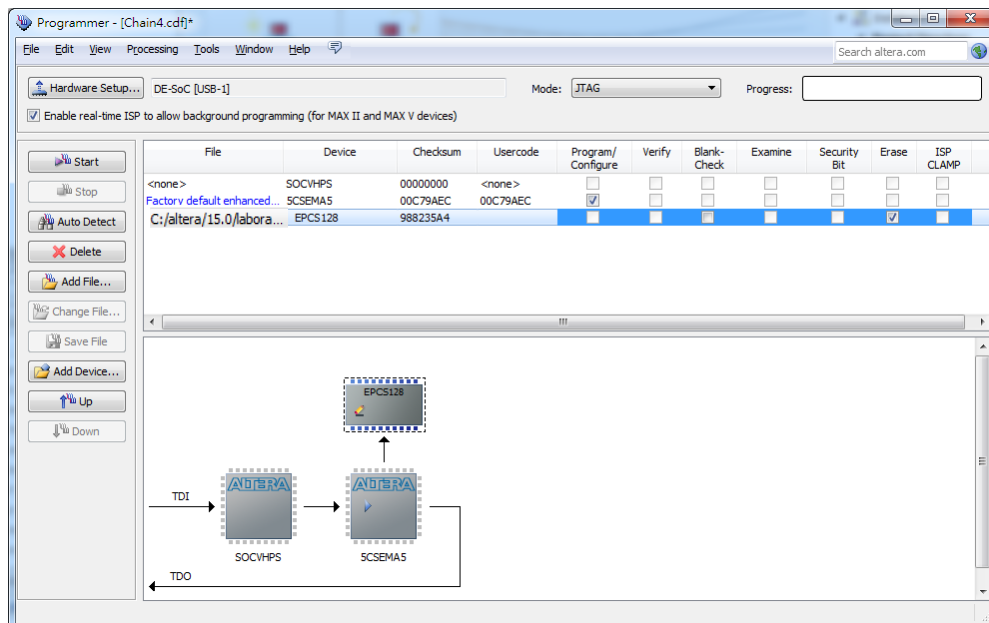
Luego dando doble clic sobre el dispositivo 5CSEMA5 abrirá un buscador de archivos, como el que muestra en la figura 2.112, se debe seleccionar el archivo JIC creado con anterioridad y darle Open.



**Figura No. 2.112: Selección de archivo JIC.**

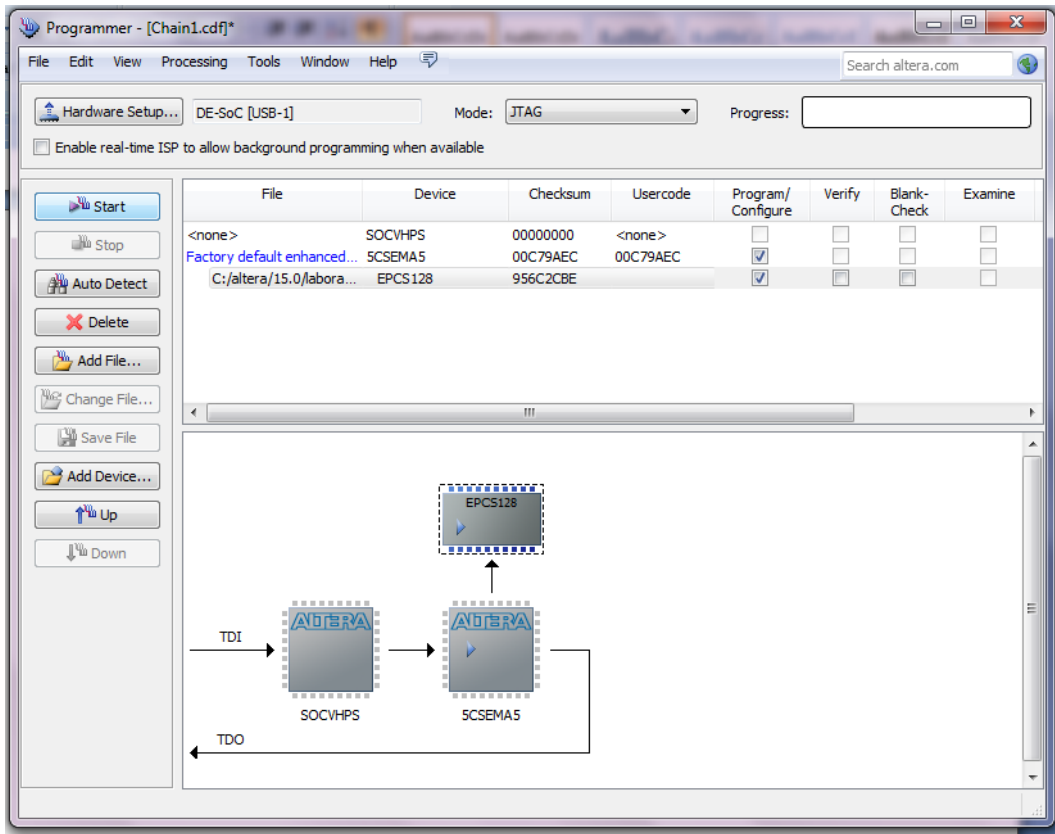
Una vez cargado el archivo JIC aparecerá el dispositivo EPCS128 en el diagrama de conexión, luego debe seleccionarse la opción **Erase** para el dispositivo EPCS128 y para el 5CSEMA se debe marcar la opción **Program/Configure**.

Una vez marcadas estas casillas se debe dar clic en Start, con esto se asegura que la memoria flash esta vacia.



**Figura No. 2.113: Erase dispositivo EPCS128.**

Una vez la memoria este limpia se debe marcar la opción Program/Configure para cada uno de los dispositivos y luego dar Start, de esta forma el archivo JIC será transferido al dispositivo EPCS128 y quedará de forma permanente en la tarjeta.



**Figura No. 2.114: Configuración de dispositivo EPCS128.**

# Capítulo 3: Laboratorios

---

## 3.1 Introducción

Altera provee de la documentación necesaria para la comprensión de las bases de dispositivos FPGA, esta documentación incluye videos tutoriales y laboratorios prácticos, todo esto es accesible a través del **Programa Universitario ALTERA**.

En los laboratorios se da por entendido que el estudiante posee las bases de los lenguajes VHDL y Verilog, también se asume que se conocen todas las técnicas involucradas en el proceso de diseño de sistemas digitales, por tal motivo se recomienda que estos laboratorios se realicen por estudiantes que hayan cursado las materias de Sistemas Digitales I (SDI-115) y Sistemas Digitales II (SDI-215).

Se presentan únicamente los laboratorios, en cada uno de estos la primera parte es de carácter demostrativo, las soluciones quedan en poder del asesor encargado para los usos que disponga conveniente.

## 3.2 Laboratorio No. 1: Interruptores, leds y multiplexores

### 3.2.1 Introducción.

El objetivo general de este laboratorio es aprender a interconectar entradas y salidas de un dispositivo FPGA y como crear circuitos multiplexores, para tal propósito se utilizarán como entradas los interruptores SW(0-9) incluidos en el kit de desarrollo DE1-SoC, para representar las salidas se hará uso de los led LEDR(0-9) y los display de 7 segmentos HEX, también incluidos en el kit, este laboratorio consta de 6 partes, relacionadas entre sí.

Para esta práctica se pide como mínimo saber de ecuaciones lógicas, multiplexores y decodificadores para display de 7 segmentos, cada una de las partes cuenta como asignación, debe resolverse basándose en criterios validos para el diseño de sistemas lógicos digitales.

Todo documento o tutorial necesario para completar el laboratorio será indicado o referenciado para un buen desarrollo, el documeto original puede descargarse en el siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab1\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab1_VHDL.pdf) [en línea] [ultima consulta 24/02/2016]

### 3.2.2 Objetivos generales

- Entender la conexión de entradas y salidas físicas del la tarjeta DE1-SoC.
- Conocer las expresiones concurrentes de VHDL.
- Implementar los códigos en una tarjeta DE1-SoC para ver su funcionamiento en tiempo real.

- Entender la diferencia entre una entidad de alto nivel versus una entidad de bajo nivel, dentro de VHDL.

### 3.2.3 Asignación 1

#### 3.2.3.1 Objetivo

En esta práctica se busca entender el proceso básico para la conexión de señales físicas, así como también el uso de vectores y sentencias concurrentes.

Por medio de sentencias concurrentes individuales se conectarán los interruptores directamente a los leds de la tarjeta, de forma que cada interruptor encienda un led, luego de finalizado el ejercicio, se repetirá pero esta vez la asignación será por medio de vectores.

#### 3.2.3.2 Desarrollo

El kit DE1-SoC cuenta con 10 interruptor de doble estado (0 ó 1), llamados SW(0-9) que pueden ser utilizados como entradas de un circuito, también cuenta con 10 leds de color rojo llamados LEDR(0-9) que pueden ser utilizados como salidas de un circuito, en la figura 3.1.1a se muestra un bloque de código en VHDL para la asignación de entradas y salidas mediante vectores, esta tipo de asignación permite que se mantenga la correlación entre las entradas y las salidas, de esta forma el SW(0) estará ligado al LEDR(0) y así sucesivamente.

```

LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
-- FORMA RAPIDA DE ASIGNAR LOS INTERRUPTOR A LOS LEDS DE FORMA
-- CORRELATIVA.

ENTITY L1_PART1_AUPIS
PORT ( SW : INSTD_LOGIC_VECTOR(9DOWNTO0) ; --INTERRUPTORES DECLARADOS COMO VECTOR
LEDR : OUTSTD_LOGIC_VECTOR(9DOWNTO0) ) ; --LEDS VECTOR
END L1_PART1_AUP;

ARCHITECTURE BEHAVIOR OF L1_PART1_AUPIS
BEGIN
LEDR <= SW;
END BEHAVIOR;

```

**Figura No. 3.1.1a: Bloque VHDL para asignación de entradas y salidas.**

Si se requiere una asignación específica se utilizan los comandos individuales para los elementos del vector, tal como se muestra en la figura 3.1.1b.

```

LEDR (9) <= SW (9) ;
LEDR (8) <= SW (8) ;
LEDR (7) <= SW (7) ;
LEDR (6) <= SW (6) ;
LEDR (5) <= SW (5) ;
LEDR (4) <= SW (4) ;
LEDR (3) <= SW (3) ;
LEDR (2) <= SW (2) ;
LEDR (1) <= SW (1) ;
LEDR (0) <= SW (0) ;

```

**Figura No. 3.1.1b: Asignación individual.**



El Kit DE1-SoC tiene conexiones internas entre el chip FPGA con los interruptores y leds, para usar los SW y los LEDR es necesario hacer una correcta asignación de pines e incluirla en el proyecto de Quartus II sobre el que se está trabajando, el número de pin correspondiente a cada elemento se puede verificar en el documento **DE1-SoC manual user**, por ejemplo el elemento SW(0) esta internamente conectado al pin **AB12** del chip FPGA mientras que el elemento LEDR(0) esta conectado al pin **V16**.

Una forma adecuada de hacer la asignación de pines es agregar el archivo **DE1\_SoC.qsf** el cual se encuentra en el siguiente link: [ftp://ftp.altera.com/up/pub/Altera\\_Material/13.1/Boards/DE1-SoC/DE1\\_SoC.qsf](ftp://ftp.altera.com/up/pub/Altera_Material/13.1/Boards/DE1-SoC/DE1_SoC.qsf)[en línea] [ultima consulta 24/02/2016] este archivo esta dado por ALTERA para el programa universitario, al usar este archivo es importante conocer la nomenclatura de los pines, ya que en base a dicha nomenclatura se deben asignar los nombres de los pines dentro de la entidad, para hacer la asignación de pines de forma manual o por medio de archivo se recomienda leer el documento **Quartus II Introduction using VHDL Design** que se encuentra en el siguiente link:

[ftp://ftp.altera.com/up/pub/Altera\\_Material/12.0/Tutorials/VHDL/Quartus\\_II\\_Introduction.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/12.0/Tutorials/VHDL/Quartus_II_Introduction.pdf)[en línea] [ultima consulta 24/02/2016].

### 3.2.3.3 Procedimiento general

El procedimiento es el mismo para cada una de las secciones, salvo se indique lo contrario, de ser así, se indicará que ítem debe ser sustituido y las instrucciones para cumplirlo.

1. Crear un nuevo proyecto en Quartus II correspondiente al circuito, elegir el chip Cyclone V 5CSEMA5F31C6 como dispositivo objetivo, ya que este es el chip FPGA contenido en el kit DE1-SoC, para mayor información se recomienda leer la sección 2.4.1.
2. Crear un nuevo archivo VHDL para el código mostrado en la figura 3.1.1a e incluirlo dentro del proyecto.
3. Asignar los pines según la discusión previa, y compilar el proyecto, se pueden consultar las secciones 2.4.4 y 2.4.5 para completar este procedimiento, si todo está bien el programa no mostrará errores, es probable que se muestren algunas advertencias.
4. Crear un nuevo archivo **University Program VWF**, esto con el fin de corroborar el circuito a través de un diagrama de tiempo, el uso de archivos de simulación se explica en la sección 2.4.7.
5. Descargar el programa hacia el kit DE1-SoC a través de **Quartus II Programmer tool** cuyo uso se discute en el manual de usuario, así como también en la sección 2.5, se debe tomar en cuenta que el archivo que se debe descargar en la tarjeta es de extensión .sof (SRAM object file), este archivo estará disponible de manera volátil en el chip FPGA, así que se recomienda probar el circuito sin apagar, ni desconectar la tarjeta.
6. Realizar las pruebas necesarias con los interruptores y leds y compruebe que el funcionamiento sea el deseado.

## 3.2.4 Asignación 2

### 3.2.4.1 Objetivo

La practica servirá a para conocer el proceso de creación de circuitos lógicos a través de ecuaciones booleanas descritas por medio de VHDL, para este caso se toma como base un circuito multiplexor.

Para cumplir con la asignación se muestra la ecuación booleana correspondiente a un circuito multiplexor, así como su respectivo circuito y tabla de verdad.

El estudiante deberá crear un archivo VHDL que genere el comportamiento de un multiplexor en la DE1- SoC, su resultado será visible a través de los interruptores y los led incluidos en la tarjeta.

### 3.2.4.2 Desarrollo

La figura 3.1.2a muestra una suma de productos para implementar un multiplexor de 2 entradas a 1 salida (mux2a1) con una variable S de selección, cuando S=0 el valor del bit presente en la entrada X se trasladará a la salida M, por el contrario si la variable S=1 el valor presente en Y será trasladado a M, cada entrada solo puede manejar un bit a la vez, en la figura 3.1.2b se muestra la tabla de verdad del circuito y en la figura 3.1.2c su respectivo símbolo.

La ecuación booleana correspondiente al circuito de la figura 3.2.1a es la siguiente:

$$m = (x \cdot \bar{s}) + (s \cdot y)$$

Ecuación No. 3.1: Expresión booleana para multiplexor de 2 a 1

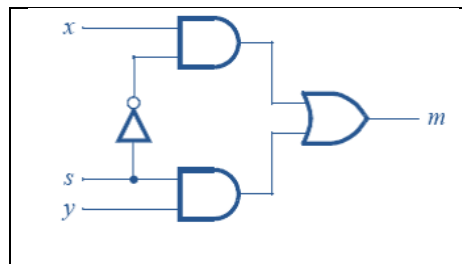


Figura No. 3.1.2: a) circuito multiplexor 2 a 1.<sup>[7]</sup>

<table border="1"><thead><tr><th><u>s</u></th><th><u>m</u></th></tr></thead><tbody><tr><td>0</td><td>x</td></tr><tr><td>1</td><td>y</td></tr></tbody></table> <p>a) Tabla de Verdad.</p>	<u>s</u>	<u>m</u>	0	x	1	y	<p>b) Símbolo.</p>
<u>s</u>	<u>m</u>						
0	x						
1	y						

Figura No. 3.1.2: b) tabla de verdad, c) símbolo.<sup>[7]</sup>

La ecuación 3.2 describe la sintaxis en lenguaje VHDL para que el circuito opere:

```
m <= (NOT (s) AND x) OR (s AND y);
```

Ecuación No. 3.2: Expresión en VHDL para multiplexor de 2 a 1

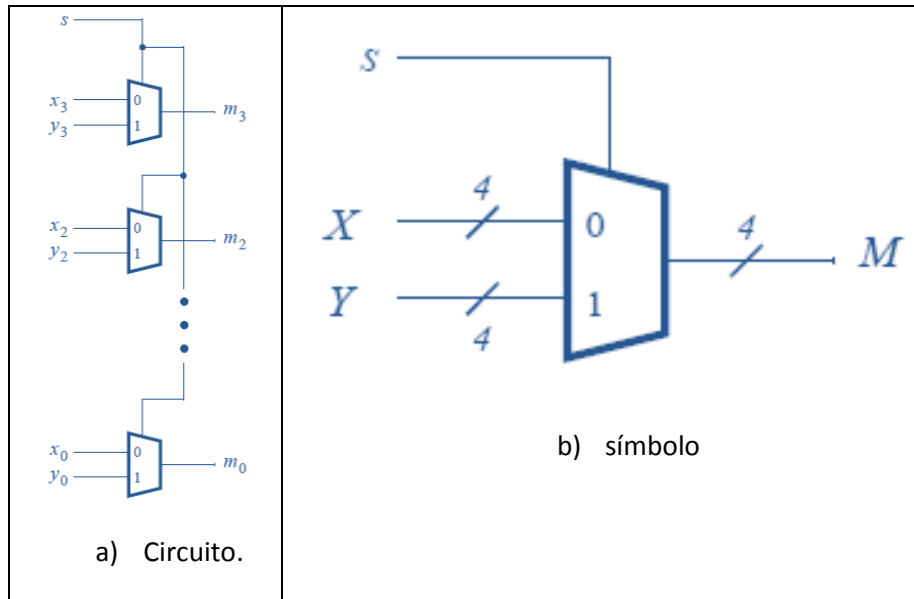
En esta práctica se requiere crear un multiplexor de 2 a 1 pero las entradas deben ser de 4 bits cada una, esto también debe ser aplicada a las salidas, se recomienda crear vectores para las entradas y las salidas, de esta forma las entradas serían X(0 a 4), Y(0 a 4) y la salida sería M(0 a 4), se recomienda el uso de señales internas (signals) dentro del código.

En la figura 3.1.3 se muestra una sección de código que ilustra el uso de señales internas, este tipo de variables representan conexiones físicas internas del FPGA, también se pueden considerar como nodos de conexión, se declaran entre architecture y begin, para mayor información se recomienda leer el documento “Lenguajes de Descripción de Hardware”, el cual se puede solicitar al asesor.

```
ARCHITECTURE BEHAVIOR OF L1 PART2 AUP IS
  SIGNAL M, X1, Y : STD_LOGIC_VECTOR(3DOWNTO0);
  SIGNAL S : STD_LOGIC;
BEGIN
  X1 <= SW(3DOWNTO0);
  Y <= SW(7DOWNTO4);
  S <= SW(9);
  M(0) <= (NOT(S) AND X1(0)) OR (S AND Y(0));
  M(1) <= (NOT(S) AND X1(1)) OR (S AND Y(1));
  M(2) <= (NOT(S) AND X1(2)) OR (S AND Y(2));
  M(3) <= (NOT(S) AND X1(3)) OR (S AND Y(3));
  LEDR(3DOWNTO0) <= M;
  LEDR(9) <= S;
END BEHAVIOR;
```

**Figura No. 3.1.3: Uso de Signal.**

En la figura 3.1.4a muestra de forma general el circuito a implementar, en esta se puede observar la composición del circuito, ya que para aumentar el ancho de bits solo se replicó el circuito 4 veces pero la entrada de selección se mantuvo fija, la figura 3.1.4b muestra su símbolo respectivo.



**Figura No. 3.1.4: 4 bits multiplexor de 2 a 1.[7]**

### 3.2.4.3 Procedimiento

Para implementar el circuito correspondiente a la figura 3.1.4 se debe seguir el procedimiento de la sección 3.2.3.3, con la variante del ítem 2 el cual debe ser sustituido por el siguiente:

2. Crear un nuevo archivo VHDL y desarrollar un código que describa el circuito de la figura 3.1.4a teniendo en cuenta lo siguiente: la variables S se debe asignar a SW(9), la variable X será operada por los interruptores SW(0 a 3), mientras que la variable Y serán los interruptores SW(4 a 7), en cuanto a la salida M esta debe ser visible a través de LEDR(0 a 3) y el estado de la variable S debe ser visible en LEDR(9).

## 3.2.5 Asignación 3

### 3.2.5.1 Objetivo

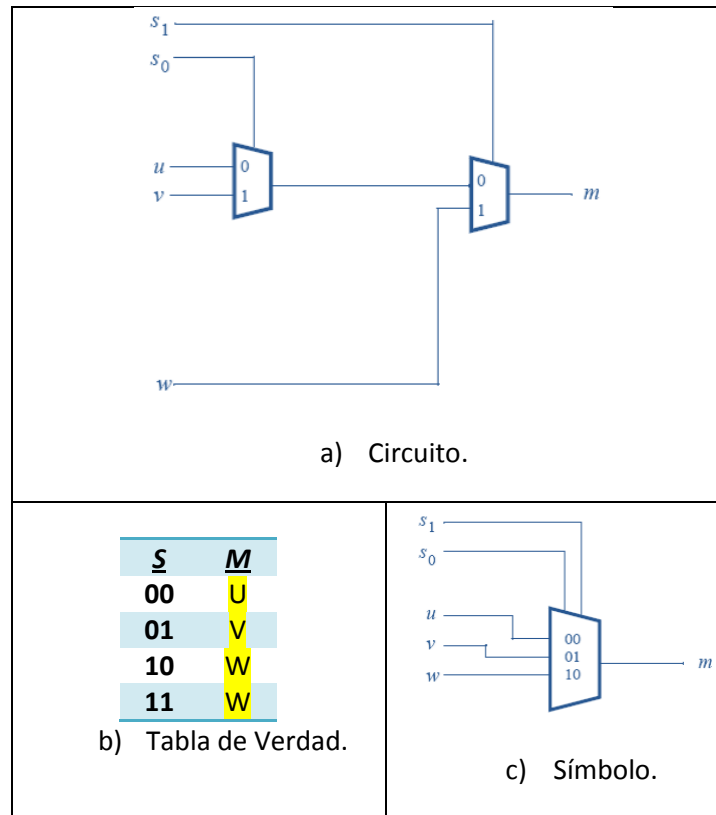
Teniendo en cuenta el circuito de la figura 3.1.4a y la metodología de diseño sobre multiplexores, el paso siguiente es realizar dispositivos en cascada, de esta forma se pueden crear circuitos complejos a partir de unidades simples.

Al final de la práctica el estudiante podrá conectar circuitos en cascada a través de componentes, entenderá el concepto de modularidad y vera el funcionamiento de un multiplexor de 3 entradas y 1 salida.

### 3.2.5.2 Desarrollo

En este ejercicio se pide crear un multiplexor de tres entradas U, V y W, la figura 3.1.5a muestra el circuito a implementar, la figura 3.1.5b muestra la tabla de verdad para el circuito y en la figura 3.1.5c se muestra el respectivo símbolo, no se debe perder de vista que la variable de selección en

esta ocasión será de 2 bits, por lo tanto se generan 4 opciones, ya que solo hay tres entradas, cuando se genere la cuarta opción la entrada a mostrar será W.



**Figura No. 3.1.5: multiplexor de 3 a 1.<sup>[7]</sup>**

Las entradas en esta ocasión tendrán un ancho de dos bits, de forma que U tendrá dos elementos U(0) y U(1), lo mismo sucede para V y W.

Se recomienda el uso de componentes de esta forma el código será más simple, tomando en cuenta que ya se realizó el ejercicio anterior y que su código es reutilizable.

### 3.2.5.3 Procedimiento

Para implementar el circuito correspondiente a la figura 3.1.5 se deben seguir el procedimiento de la sección 3.2.3.3, con la variante del ítem 2 el cual debe ser sustituido por el siguiente:

2. Crear un nuevo archivo VHDL y desarrollar un código que describa el circuito de la figura 3.1.5a teniendo en cuenta lo siguiente: la variables S se debe asignar a SW(8 a 9), la variable U será operada por los interruptores SW(0 a 2), la variable V serán los interruptores SW(3 a 4), mientras que la variables W serán los interruptores SW(5 a 6) en cuanto a la salida M esta debe ser visible a través de LEDR(0 a 1) y el estado de la variable S debe ser visible en LEDR(8 a 9).

## 3.2.6 Asignación 4

### 3.2.6.1 Objetivo

Los decodificadores para display de 7 segmentos son una parte importante en el muestreo de resultados, un decodificador puede tomar muchas variantes y estar ligado a un número bastante grande de entradas, todo recae en la necesidad e inventiva del usuario, y ese es el propósito de esta práctica, crear un decodificador muy diferente al decimal y hexadecimal que son los más utilizados.

Con esta práctica se pretende que el estudiante entienda los conceptos de decodificadores, así como el uso correcto de los display incluidos en la DE1- SoC.

### 3.2.6.2 Desarrollo

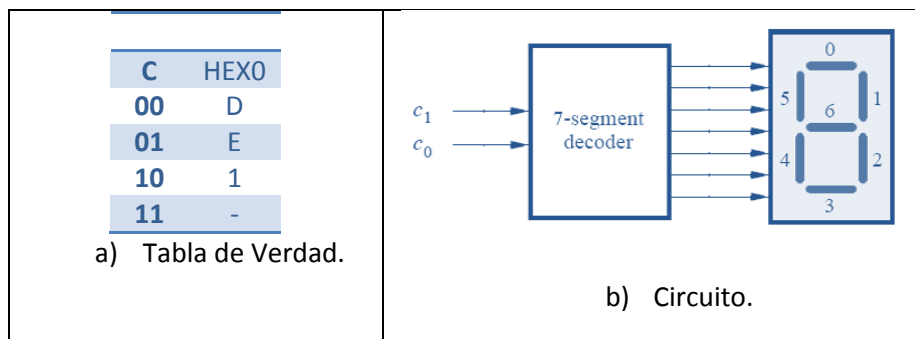
En esta sección se requiere crear un decodificador para un display de 7 segmentos de tal forma que utilizando un vector C de 2 bits se logren los caracteres "d", "E", "1" y "-" para las combinaciones mostradas en la tabla de verdad de la figura 3.1.6a, al igual que sucede con los interruptores y leds, el kit DE1-SoC dispone de 6 display de ánodo común (se necesita un 0 lógico para encender el segmento) interconectados al chip FPGA, es de tomar en cuenta la correcta asignación de pines.

Si se utiliza el archivo DE1\_SoC.qsf la nomenclatura para declarar el puerto de un display es la siguiente:

```
HEX0 :OUTSTD_LOGIC_VECTOR(0 TO 6);
```

Ecuación No. 3.3: Declaración de un Puerto HEX.

Donde el número que acompaña a la palabra HEX denota el número de display a utilizar, ya que se declara como vector cada elemento está ligado a un segmento en específico, esto se describe mejor en la figura 3.1.6b.



**Figura No. 3.1.6: Decodificador de 7 segmentos para 2 entradas.<sup>[7]</sup>**

Se recomienda deducir las ecuaciones booleanas para cada segmento y hacer la asignación siguiendo el criterio de la asignación individual, también es recomendable revisar la asignación de pines.

### 3.2.6.3 Procedimiento

Para implementar el circuito correspondiente a la figura 3.1.6 se deben seguir el procedimiento de la sección 3.2.3.3, con la variante de los ítems 2 y 6, los cuales deben ser sustituidos por los siguientes literales:

2. Crear un nuevo archivo VHDL y desarrollar un código que describa el circuito de la figura 3.1.6b teniendo en cuenta lo siguiente: el vector C debe ser operado por los interruptores SW(0 a 1), como medida de control el estado de C debe ser visible en LEDR(0 a 1), se recomienda usar el display 0 y tener en cuenta la declaración del puerto de 7 bits como se mostró con anterioridad.
6. Realizar las pruebas necesarias con los interruptores y leds y comprobar que el funcionamiento es correcto a través de la respuesta visible en el display.

## 3.2.7 Asignación 5

### 3.2.7.1 Objetivo

Esta práctica tiene como propósito el uso de componentes de diferentes clases para crear unidades complejas, el buen uso de componentes reduce en gran medida el proceso de creación de entidades de alto nivel, también supone un orden en el código y su fácil comprensión.

Para lograr este objetivo se hará uso de los códigos previamente creados en las secciones 3.2.5 y 3.2.6.

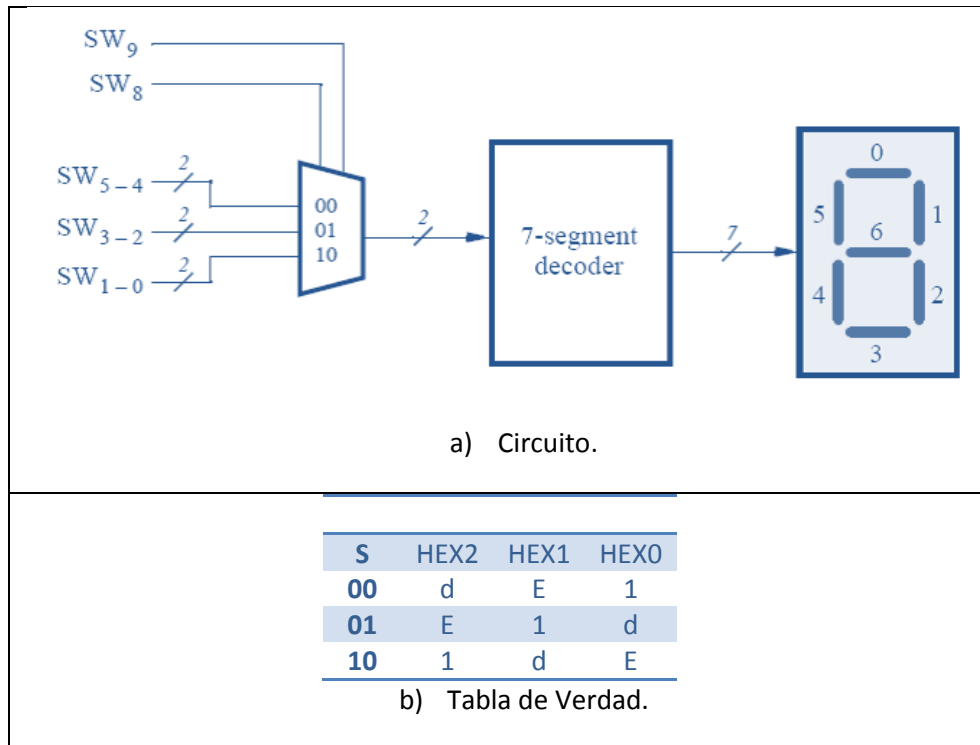
Al final de la práctica se debe tener claro el concepto de modularidad, conexiones entre componentes, uso de interruptores como variables de entrada y uso de leds y display como variables de salida.

### 3.2.7.2 Desarrollo

Considere el circuito descrito en la figura 3.1.7a, este utiliza un multiplexor de 3 a 1, con entradas y salida de 2 bits de ancho para activar los caracteres a mostrar en un display de 7 segmentos. Utilizando el decodificador de la asignación 4 se pueden mostrar 4 caracteres disponibles de acuerdo figura 3.1.5a, dejando fijas las entradas del multiplexor para que cada una muestre un carácter específico, la selección del carácter dependerá de los valores presentes en S.

En la figura 3.1.8 se muestra un esqueleto de código que puede usarse como base para crear el circuito descrito con anterioridad, los circuitos de las asignaciones 3 y 4 son utilizados como subcircuitos en esta estructura, en forma de componentes, en esta práctica se deben utilizar tres display HEX0, HEX1 y HEX2 para mostrar la palabra "dE1", por lo tanto es necesario replicar el circuito de la figura 3.1.6b tres veces, de esta forma se deben tener tres instancias de cada subcircuitos funcionando de forma coordinada, el vector S se utilizara para rotar la palabra hacia la izquierda

tal como se muestra en la figura 3.17b. El vector S será el mismo para todas las instancias y para las variables de entrada se dispondrá únicamente de SW(0 a 5).



**Figura No. 3.1.7: Circuito de la asignación. [7]**

Es de carácter obligatorio realizar esta asignación por medio de componentes, para información adicional sobre el uso de este tipo de estructura, se recomienda leer el documento “Lenguajes de Descripción de Hardware”, además altera provee de un video tutorial sobre VHDL llamado “VHDL Basics”, el cual puede ser encontrado en este sitio web:

<http://wl.altera.com/education/training/courses/OHDL1110>[en línea] [ultima consulta 24/02/2016]

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY PART5 IS
PORT ( SW : INSTD_LOGIC_VECTOR (9DOWNTO0) );
LEDR : OUTSTD_LOGIC_VECTOR (9DOWNTO0) );
HEX0 : OUTSTD_LOGIC_VECTOR (0TO6) );
END PART5;

ARCHITECTURE BEHAVIOR OF PART5 IS
COMPONENT MUX_2BIT_3TO1
PORT ( S, U, V, W : INSTD_LOGIC_VECTOR (1DOWNTO0) );
M : OUTSTD_LOGIC_VECTOR (1DOWNTO0) );
ENDCOMPONENT;
COMPONENT CHAR_7SEG
PORT ( C : INSTD_LOGIC_VECTOR (1DOWNTO0) );
DISPLAY : OUTSTD_LOGIC_VECTOR (0TO6) );

```



```

ENDCOMPONENT;
SIGNAL M0 :STD_LOGIC_VECTOR(1DOWNTO0);
BEGIN
U0: MUX_2BIT_3TO1 PORTMAP (SW(9DOWNTO8), SW(5DOWNTO4), SW(3DOWNTO2),
SW(1DOWNTO0), M0);
H0: CHAR_7SEG PORTMAP (M0, HEX0);
--CODIGO A DESARROLLAR
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
-- IMPLEMENTAR UN MULTIPLEXOR DE 3-A-1 CON 2 BITS DE ANCHO
ENTITY MUX_2BIT_3TO1 IS
PORT( S, U, V, W :INSTD_LOGIC_VECTOR(1DOWNTO0);
M :OUTSTD_LOGIC_VECTOR(1DOWNTO0));
END MUX_2BIT_3TO1;
ARCHITECTURE BEHAVIOR OF MUX_2BIT_3TO1 IS
--CODIGO A DESARROLLAR
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
ENTITY CHAR_7SEG IS
PORT( C :INSTD_LOGIC_VECTOR(1DOWNTO0);
DISPLAY :OUTSTD_LOGIC_VECTOR(0TO6));
END CHAR_7SEG;
ARCHITECTURE BEHAVIOR OF CHAR_7SEG IS
-- CÓDIGO A DESARROLLAR
END BEHAVIOR;

```

**Figura No. 3.1.8: esqueleto de código para implementar el circuito de la figura 3.1.7.**

### 3.2.7.3 Procedimiento

Para implementar el circuito correspondiente a la figura 3.1.7 se deben seguir el procedimiento de la sección 3.2.3.3, con la variante de los ítems 2 y 6, los cuales deben ser sustituidos por los siguientes literales:

2. Crear un nuevo archivo VHDL y desarrollar un código que describa el circuito de la figura 3.1.7a teniendo en cuenta lo siguiente: la variables S se debe asignar a SW(8 a 9), la variable U será operada por los interruptores SW(0 a 2), la variable V serán los interruptores SW(3 a 4), mientras que la variables W serán los interruptores SW(5 a 6), esta asignación de interruptores será similar para los tres multiplexores, se recomienda buscar un orden adecuado para colocarlas en cada instancias, recordando que lo que se puede alterar es el orden de las variables en las combinaciones de cada multiplexor, los interruptores deben ser visibles a través de LEDR(0 a 9), los display a utilizar serán HEX0, HEX1 y HEX2, siempre se debe revisar que los pines estén asignados según el manual de usuario.
6. Realizar las pruebas necesarias con los interruptores y compruebe que el funcionamiento sea el deseado a través de la respuesta de los display.

## 3.2.8 Asignación 6

### 3.2.8.1 Objetivo

Como parte final se deben aplicar todos los conocimientos adquiridos en este laboratorio para crear un circuito de mediana complejidad.

El circuito tiene como propósito mostrar en los 6 display una palabra determinada, luego con los interruptores se rotara la palabra a través de los display, con los led se hará un monitoreo de los estados de los interruptores.

Para completar esta asignación se deben tener los conceptos claros de señales internas, componentes, modularidad y manejo de señales de entrada y salida.

### 3.2.8.2 Desarrollo

En esta práctica se deben extender los circuitos ya realizados de tal forma que utilizando los 6 display se pueda desplazar la palabra dE1 hacia la izquierda rellorando los espacios con el carácter auxiliar, para este propósito se debe crear un multiplexor de 6 a 1, con entradas y salida de 2 bits de ancho, el estudiante debe aplicar los conceptos de multiplexores en cascada teniendo en cuenta que el vector S en esta ocasión será de 3 bits, se debe utilizar los conceptos empleados en la asignación 5 para lograr la rotación, como únicamente se disponen de 10 interruptores y el vector de selección debe ser el mismo para todas las instancias restan únicamente 7 interruptores disponibles para las 6 entradas de los multiplexores, se debe buscar un modelo en donde únicamente sea necesario el uso de los interruptores SW(0 a 5) para lograr el objetivo de la práctica, la Tabla 3.1.1 muestra las combinaciones que satisfacen el circuito.

S	HEX5	HEX4	HEX3	HEX2	HEX1	HEX0
000	-	-	-	d	E	1
001	-	-	d	E	1	-
010	-	d	E	1	-	-
011	D	E	1	-	-	-
100	E	1	-	-	-	D
101	1	-	-	-	d	E

Tabla No. 3.1.1: Tabla de verdad para asignación 6.

### 3.2.8.3 Procedimiento

Para implementar la tabla de verdad 3.1.1 se debe seguir el procedimiento de la sección 3.2.3.3, con la variante de los ítems 2 y 6, los cuales deben ser sustituidos por los siguientes literales:

2. Crear un nuevo archivo VHDL y desarrollar un código que describa el circuito que satisfaga la tabla 3.1.1 teniendo en cuenta lo siguiente: la variables S se debe asignar a SW(7 a 9), la variable U será operada por los interruptores SW(0 a 2), la variable V serán los interruptores SW(3 a 4), mientras que la variables W serán los interruptores SW(5 a 6), en esta asignación se debe crear un variable auxiliar por ejemplo Z la cual sea la encargada de

almacenar el carácter auxiliar, se debe buscar los interruptores adecuados para que esta variable opere de forma idónea, se recomienda leer información sobre el operador & el cual sirve para concatenar elementos, los interruptores deben ser visible a través de LEDR(0 a 9), los display a utilizar serán HEX0, HEX1, HEX2, HEX3, HEX4, HEX5 Y HEX6 siempre se debe revisar que los pines estén asignados según el manual de usuario.

6. Realizar las pruebas necesarias con los interruptores y compruebe que el funcionamiento sea el deseado a través de la respuesta de los display.

## 3.3 Laboratorio No. 2: Números y display

### 3.3.1 Introducción

La representación de símbolos a través de un sistema estructurado de valores ha logrado que la comunicación sea más eficiente, a través de mensajes definidos, es por esto que poder enviar mensajes a través de cualquier sistema se vuelve una necesidad.

Esta práctica se enfocará en crear circuitos capaces de mostrar mensajes a través de los display, en el laboratorio anterior se experimentó con crear decodificadores capaces de mostrar símbolos representables a través de 7 segmentos, en esta oportunidad se crearán decodificadores que muestren valores contenido en los sistemas de numeración más comunes.

Aprovechando la representación numérica que pueden brindar los display se crearán circuitos con la capacidad de realizar operaciones numéricas, tanto los operandos como el resultado serán visibles a través de los display de la tarjeta DE1-SoC.

El archivo original de la práctica puede encontrarse en el siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab2\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab2_VHDL.pdf) [en línea] [ultima consulta 24/02/2016].

### 3.3.2 Objetivos Generales

- Conocer a fondo los diferentes sistemas numéricos representables en display de 7 segmentos.
- Crear circuitos decodificadores para cada uno de los sistemas numéricos.
- Entender la lógica de la operación aritmética suma para los sistemas binarios.
- Desarrollar códigos base para la suma de forma que sirva como módulo para laboratorios posteriores.

### 3.3.3 Asignación 1

#### 3.3.3.1 Objetivo

El código BCD (Binary-Coded Decimal) es un sistema utilizado para representar los números decimales a través de código binario, de forma que se necesitan 4 bits para representar un dígito decimal, como consecuencia, si se necesita representar un número decimal de 3 dígitos, serán necesarios 12 bits para cumplir la tarea.

Ejemplo: 984 (decimal) = 1001 1000 0100 (binario).

El propósito de la práctica es crear un modulo decodificador que reciba en su entrada 4 bits y en su salida provea las condiciones necesarias para mostrar los números decimales a través de los display de 7 segmentos.

#### 3.3.3.2 Desarrollo

Se desea mostrar en los display HEX0 y HEX1 dos números BCD, para establecer los valores de los números se utilizarán los interruptores SW(0 a 7), siendo SW(0 a 3) el primer número que se

mostrará en HEX0, mientras que SW(4 a 7) será el segundo número a mostrar en HEX1, el circuito debe tener la capacidad de mostrar los dígitos del 0 al 9, las combinaciones no utilizadas (1010 a 1111) deben ser tratadas como “don’t care conditions”.

En la figura 3.2.1 se muestra el código a ser implementado, nótese el uso de módulos para la simplificación de la entidad principal.

```

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY LAB2_PART1_AUP IS
PORT ( SW :INSTD_LOGIC_VECTOR (9DOWNTO0) ;
      LEDR :OUTSTD_LOGIC_VECTOR (9DOWNTO0) ;
      HEX0 :OUTSTD_LOGIC_VECTOR (0TO6) ;
      HEX1 :OUTSTD_LOGIC_VECTOR (0TO6) ) ;
END LAB2_PART1_AUP;

ARCHITECTURE BEHAVIOR OF LAB2_PART1_AUP IS
COMPONENT DEC_7SEG
PORT ( A, B, C, D :INSTD_LOGIC;
      DISPLAY :OUTSTD_LOGIC_VECTOR (6DOWNTO0) ) ;
ENDCOMPONENT;
BEGIN
LEDR <= SW;
H0: DEC_7SEG PORTMAP (SW(3), SW(2), SW(1), SW(0), HEX0);
H1: DEC_7SEG PORTMAP (SW(7), SW(6), SW(5), SW(4), HEX1);
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
ENTITY DEC_7SEG IS
PORT ( A, B, C, D :INSTD_LOGIC;
      DISPLAY :OUTSTD_LOGIC_VECTOR (0TO6) ) ;
END DEC_7SEG;

ARCHITECTURE BEHAVIOR OF DEC_7SEG IS
BEGIN
--          0
--      5|__|1
--          6
--      4|__|2
--          3
DISPLAY(0) <= (B AND (NOT C) AND (NOT D)) OR (D AND (NOT A) AND (NOT B) AND (NOT C));
DISPLAY(1) <= (B AND C AND (NOT D)) OR (B AND (NOT C) AND D);
DISPLAY(2) <= (C AND (NOT B) AND (NOT D));
DISPLAY(3) <= (B AND C AND D) OR (B AND (NOT C) AND NOT (D)) OR (D AND (NOT C) AND (NOT B));
DISPLAY(4) <= D OR (B AND (NOT C));
DISPLAY(5) <= (C AND D) OR ((NOT A) AND (NOT B) AND D) OR ((NOT B) AND C);
DISPLAY(6) <= ((NOT A) AND (NOT B) AND (NOT C)) OR (B AND C AND D);
END BEHAVIOR;

```

**Figura No. 3.2.1: Código para asignación 1.**

### 3.3.3.3 Procedimiento

Para implementar el circuito correspondiente se deben seguir los siguientes pasos:

1. Crear un nuevo proyecto en Quartus II correspondiente al circuito, elegir el chip Cyclone V 5CSEMA5F31C6 como dispositivo objetivo, ya que este es el chip FPGA contenido en el kit DE1-SoC.
2. Crear un nuevo archivo VHDL para el código que describa la funcionalidad del circuito e incluirlo dentro del proyecto, se deben utilizar únicamente expresiones simples de lenguaje VHDL, de ser necesario se pueden utilizar componentes, la descripción del decodificador debe ser por medio de ecuaciones booleanas.
3. Asignar los pines de forma adecuada y compilar el proyecto, si todo está bien el programa no mostrará errores, es probable que se muestren algunas advertencias.
4. Crear un nuevo archivo **University Program VWF**, esto con el fin de corroborar el circuito a través de un diagrama de tiempo.
5. Descargar el programa hacia el kit DE1-SoC a través de **Quartus II Programmer tool** cuyo uso se discute en el manual de usuario, se debe tomar en cuenta que el archivo que se debe descargar en la tarjeta es de extensión .sof (SRAM object file), este archivo estará disponible de manera volátil en el chip FPGA, así que se recomienda probar el circuito sin apagar, ni desconectar la tarjeta.
6. Realizar las pruebas necesarias con los interruptores y visualizar los resultados a través de los elementos HEX0 y HEX1.

## 3.3.4 Asignación 2

### 3.3.4.1 Objetivo

En el ejercicio anterior se mostraban dos números independientes teniendo 8 entradas, tal como lo indica el código BCD, en esta práctica se requiere representar los 16 números posibles que forman las combinaciones de 4 entradas, la representación será en decimal, auxiliándose con 2 display.

### 3.3.4.2 Procedimiento

En esta sección se requiere un circuito capaz de convertir un número binario de 4 bits, contenido en un vector V de cuatro bits, para ser mostrado en dos display de 7 segmentos, en el ejercicio anterior ambos números eran independientes, en esta ocasión se deben mostrar las 16 combinaciones correspondientes, en la tabla 3.2.1 se muestra la forma en que el circuito debe trabajar.

V	HEX1	HEX0
0000	0	0
0001	0	1
0010	0	2
0011	0	3
...	...	...
1001	0	9

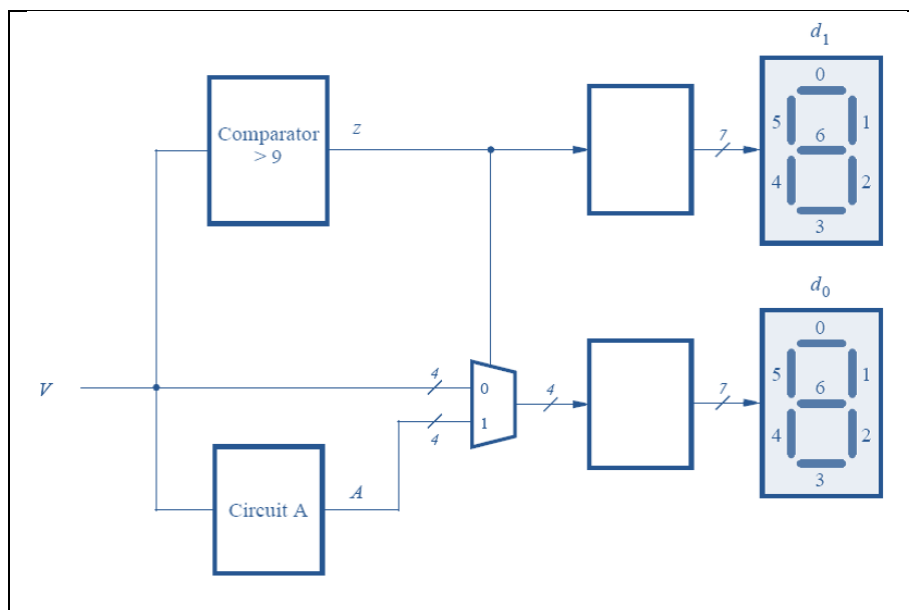
<b>1010</b>	1	0
<b>1011</b>	1	1
<b>1100</b>	1	2
<b>1101</b>	1	3
<b>1110</b>	1	4
<b>1111</b>	1	5

*Tabla No. 3.2.1: Tabla de verdad para asignación 2.*

La figura 3.2.2 muestra un diseño parcial de un circuito capaz de dar solución al problema, este circuito incluye una etapa de comparación en donde se evalúa el vector V, cuando este es mayor que 9 la salida Z=1, se debe diseñar una expresión booleana para la variable Z, se recomienda usar el método de mapas de Karnaugh, el circuito se creará con sentencias simples de VHDL.

La variable Z cumple dos funciones, su primera labor es ser la entrada para un decodificador para HEX1, esto no debe representar mayor problema siempre y cuando se tengan claros los conceptos de concatenación de elementos, la segunda función de Z es ser la variable de selección para un multiplexor de 2 a 1 de 4 bits de ancho.

El multiplexor tendrá a su salida un decodificador similar al utilizado en la parte anterior, la entrada 0 estará conectada al vector V directamente, mientras que la entrada 1 estará conectada a la salida de un circuito auxiliar, este circuito produce una salida A y funciona de tal forma que cuando V="1010", la salida en A="0000", si V="1011" entonces A="0001"... Y así sucesivamente de forma que cuando V="1111" el resultado de A="0101", el circuito auxiliar debe ser diseñado por medio de mapas de Karnaugh.



*Figura No. 3.2.2: Diseño parcial para circuito de asignación 2.<sup>[8]</sup>*

### 3.3.4.3 Procedimiento

Para implementar el circuito correspondiente se deberá seguir el procedimiento descrito en la sección 3.3.3.3, solo se sustituirán los literales detallados a continuación:

2. Crear un nuevo archivo VHDL para el código que describa la funcionalidad del circuito e incluirlo dentro del proyecto, el vector V será controlado por los interruptores SW(0 a 3), se deben utilizar únicamente expresiones simples de lenguaje VHDL, de ser necesario se pueden utilizar componentes, la descripción del decodificador debe ser por medio de ecuaciones booleanas, para esta práctica no se permiten el uso de ninguna estructura de control, tales como IF...ELSE, CASE o sentencias similares.

## 3.3.5 Asignación 3

### 3.3.5.1 Objetivo

Cuando se quieren sumar dos números, el principio de la operación indica que se deben tomar los números menos significativos y sumarlos tal cual, si el resultado de esa operación arroja un número de 2 dígitos, el más significativo se acarrea para ser sumado en la próxima operación, este proceso se repite dígito a dígito, hasta alcanzar los números más significativos.

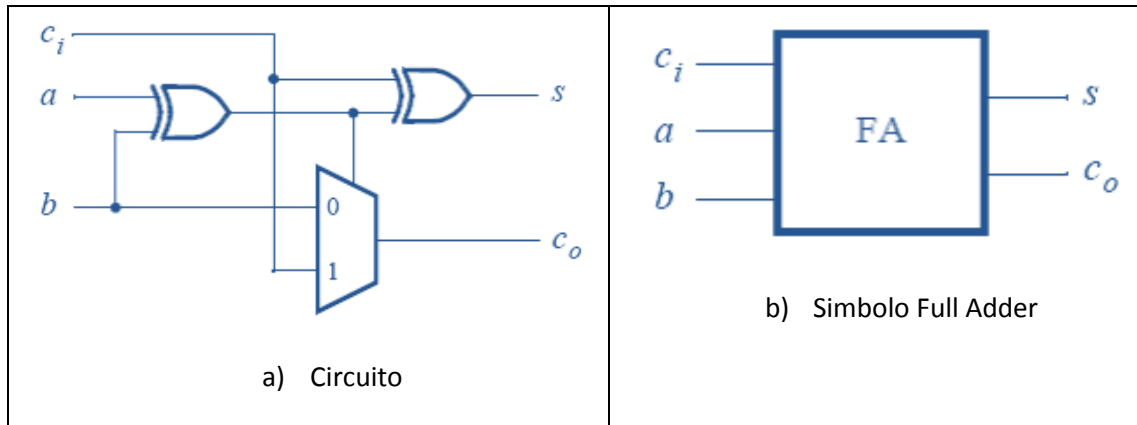
En la operación de bits este principio se aplica de igual forma, la suma bit a bit, y para comprobarlo se realizará un circuito FULL ADDER, el cual toma como base lo antes mencionado.

### 3.3.5.2 Desarrollo

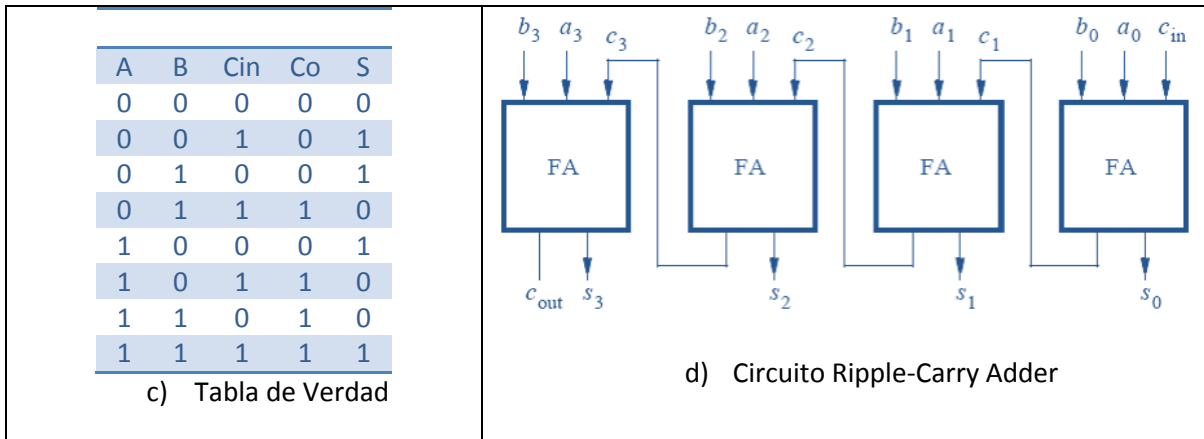
En la figura 3.2.3a se muestra un circuito Full Adder el cual posee tres entradas: a, b y Cin, y cuenta con dos salidas: s y Co, el funcionamiento del circuito se describe a través de su tabla de verdad presentada en la figura 3.2.3d, el circuito hace una suma binaria entre las entradas a y b, luego al resultado le suma el valor de Cin, de esta forma el valor de s puede ser '1' ó '0', Co es la variable que registra si la suma produjo un bit de acarreo, en la figura 3.2.3b se muestra el símbolo correspondiente a este circuito.

La figura 2d muestra un arreglo de Full Adder, este nuevo circuito tiene la capacidad de sumar números de 4 bits, a este arreglo se le llama Ripple-Carry Adder, ya que hace una suma bit a bit, el acarreo resultante en cada operación se convierte en la entrada Cin de la siguiente operación, el orden de operación comienza por el LSB hasta llegar al MSB, si la suma total produce un bit de acarreo este será visible en el último Co.





**Figura No. 3.2.3: a) Circuito Full Adder b) Símbolo Full Adder. [8]**



**Figura No. 3.2.3: c) Tabla de verdad Full adder, d) circuito Ripple-Carry Adder. [8]**

En esta práctica requiere crear un programa en VHDL que describa el funcionamiento del circuito mostrado en la figura 3.2.3d, los vectores A y B serán los interruptores SW(0 a 7) y la variable Cin será el interruptor SW(9), los resultados debe ser visibles a través de los LEDR, se recomienda crear una entidad que cumpla con los requisitos del circuito Full Adder, luego por medio de componentes declare las instancias necesarias para crear el circuito de alto nivel.

### 3.3.5.3 Procedimiento

Para implementar el circuito correspondiente se tomará como base el procedimiento 3.3.3.3, únicamente se reemplazarán los ítems que se detallan a continuación:

2. Crear un nuevo archivo VHDL para el código que describa la funcionalidad del circuito e incluirlo dentro del proyecto, el vector A será controlado por los interruptores SW(0 a 3), mientras que el vector B será controlado por los interruptores SW(4 a 7), la variable Cin será el interruptor SW(9), el vector S deberá ser visible a través de LEDR(5 a 8) y la variable Co debe ser visible a través de LEDR(9) se deben utilizar únicamente expresiones simples

de lenguaje VHDL, de ser necesario se pueden utilizar componentes, la descripción de cada uno de los elementos involucrados debe ser por medio de ecuaciones booleanas, para esta práctica no se permiten el uso de ninguna estructura de control, tales como IF...ELSE, CASE o sentencias similares.

6. Realizar las pruebas necesarias con los interruptores a forma de cubrir todas las combinaciones posibles, verifique los resultados a través de LEDR

### 3.3.6 Asignación 4

#### 3.3.6.1 Objetivos

Una vez comprendido el funcionamiento de los FULL ADDER es válido dar el siguiente paso, suma de números BCD, ya que estos números son de 4 bits, será de gran utilidad el circuito previo, además ya se cuenta con la experiencia suficiente para representar su respuesta a través de display de 7 segmentos.

Esta será la meta de la práctica crear un circuito capaz de hacer sumas BCD y representar su resultado a través de los display incluidos en la tarjeta DE1-SoC.

#### 3.3.6.2 Desarrollo

Usando el circuito creado en la sección anterior se puede crear un sumador BCD para números de 4 bits, asumiendo que se tienen dos números BCD J y K, también se dispone una entrada Cin, se pide diseñar un circuito que sume los dos números y que luego se le sume la entrada Cin, el resultado de esta operación será un número de 5 bits, ya que con 5 bits solo se logran 32 combinaciones fácilmente se observa que el resultado puede ser representado en 2 display HEX0 y HEX1, de esta forma si una suma es igual a 0, los displays serán HEX1HEX0=00, para una suma con resultado diez los valores en los display serán HEX1HEX0=10, considerando que Cin solo puede vale 1 o 0, el valor máximo que supondrá el circuito será:  $J+K+Cin = 9 + 9 + 1 = 19$ .

Una forma fácil de crear este circuito es hacer uso de los códigos diseñados con anterioridad, el primer paso será utilizar el ripple-carry adder para lograr la suma, luego con el circuito de la asignación 2 fácilmente se pueden representar sumas donde  $J+K+Cin \leq 15$ , este circuito puede ser modificado de forma que se puedan mostrar los números restantes.

Se debe agregar un sub-circuito que al detectar un número invalido (mayor que 9) en cualquiera de los dos números encienda LEDR(9).

#### 3.3.6.3 Procedimiento

Para implementar el circuito correspondiente se mantiene como base el procedimiento de la sección 3.3.3.3, únicamente se reemplazaran los ítems a continuación descritos:

2. Crear un nuevo archivo VHDL para el código que describa la funcionalidad del circuito e incluirlo dentro del proyecto, el vector J será controlado por los interruptores SW(0 a 3), mientras que el vector K será controlado por los interruptores SW(4 a 7), la variable Cin será el interruptor SW(9), se utilizarán el display HEX5 para representar a J y HEX3 para representar a K, el resultado de la operación será visible a través de HEX0 y HEX1, LEDR(9)

actuara como piloto para mostrar error en la selección de números, se deben utilizar únicamente expresiones simples de lenguaje VHDL, de ser necesario se pueden utilizar componentes, la descripción de cada uno de los elementos involucrados debe ser por medio de ecuaciones booleanas, para esta práctica no se permiten el uso de ninguna estructura de control, tales como IF...ELSE, CASE o sentencias similares.

6. Realizar las pruebas necesarias con los interruptores a forma de cubrir todas las combinaciones posibles, verifique los resultados a través de HEX0 y HEX1.

### 3.3.7 Asignación 5

#### 3.3.7.1 Objetivo

Un pseudocódigo es un algoritmo de programación de alto nivel, que por lo general muestra las indicaciones que se deben seguir para alcanzar una meta, la meta en esta práctica será convertir un pseudocódigo a un código VHDL capaz de realizar la misma función que la asignación 4 pero con menos instrucciones.

#### 3.3.7.2 Desarrollo

En la figura 3.2.4 se muestra algoritmo que representa un pseudo código que ilustra otro enfoque de entender un sumador BCD.

```
1  T0 = A + B + c0
2  if (T0 >9) then
3      Z0 =10;
4      c1 =1;
5  else
6      Z0 =0;
7      c1 =0;
8  endif
9  S0 = T0 - Z0
10 S1 = c1
```

**Figura No. 3.2.4: algoritmo para un sumador BCD.<sup>[8]</sup>**

No hace falta mencionar que es razonablemente más sencillo que el creado anteriormente, desde la línea 2 hasta 8 se representa una estructura de control if...else, la línea 9 puede ser reemplazada por una función que involucre sumas en lugar de una resta. La intención de este ejercicio es comprobar la fiabilidad de el compilador VHDL para crear circuitos utilizando estructuras de control y operadores VHDL (+, -, >), esto supone que transforme ese pseudo código en un programa VHDL.

Para utilizar los símbolos de resta es necesario incluir dentro del código los componentes de la estructura ieee.std\_unsigned.all, mismos que se encuentran en la librería ieee.

#### 3.3.7.3 Procedimiento

Para implementar la práctica se mantendrá como base el procedimiento de la sección 3.3.3.3, únicamente se reemplazarán los siguientes ítems:

2. Crear un nuevo archivo VHDL para el código que describa la funcionalidad del circuito e incluirlo dentro del proyecto, el vector A será controlado por los interruptores SW(0 a 3), mientras que el vector B será controlado por los interruptores SW(4 a 7), la variable Cin será el interruptor SW(9), se utilizarán el display HEX5 para representar a J y HEX3 para representar a K, el resultado de la operación será visible a través de HEX0 y HEX1.
4. Utilice la herramienta **Quartus II RTL Viewer Tool** para examinar el circuito producido por el compilador VHDL, compare los resultados con el circuito anterior.
6. Realizar las pruebas necesarias con los interruptores a forma de cubrir todas las combinaciones posibles, verifique los resultados a través de HEX0 y HEX1.

### 3.3.8 Asignación 6

#### 3.3.8.1 Objetivos

Los sumadores a menudo se utilizan en conjunto con circuitos correctores para lograr sumas en BCD, la meta de esta práctica es lograr representar un número de 6 bits en formato BCD, utilizando sumadores y correctores de la suma.

#### 3.3.8.2 Desarrollo

Diseñe un circuito combinacional que convierte un número binario de 6 bits en un número decimal de 2 dígitos representados en forma BCD. Utilice los interruptores SW(5-0) para establecer los valores del número y para mostrarlos utilice los display HEX0 y HEX1.

#### 3.3.8.3 Procedimiento

Para implementar la práctica se mantendrá como base el procedimiento de la sección 3.3.3.3, únicamente se reemplazaran los siguientes ítems:

2. Crear un nuevo archivo VHDL para el código que describa la funcionalidad del circuito e incluirlo dentro del proyecto, el vector A será controlado por los interruptores SW(0 a 5) , el resultado de la operación será visible a través de HEX0 y HEX1.
6. Realizar las pruebas necesarias con los interruptores a forma de cubrir todas las combinaciones posibles, verifique los resultados a través de HEX0 y HEX1.

## 3.4 Laboratorio No. 3: Latches, Flip Flops y Registros

### 3.4.1 Introducción

Suele suceder que en un circuito es necesario mantener fijo un pulso, guardar una combinación o simplemente trabajar con señales biestables para determinados propósitos.

Todo esto es gracias a los flip flops, latches y registros, los cuales son circuitos combinatoriales con el propósito de mantener fijas las señales en base a eventos, los cuales por lo general son cambios en la señal de un reloj.

Los tipos de FF y latches varían de acuerdo al arreglo lógico con el que se construyan, los eventos que rigen el cambio en su salida pueden ser de tres tipos: cambio en la señal de reloj (1 o 0), por flanco de bajada y por flanco de subida.

Tomando como base que todo circuito se compone de compuertas lógicas, las primeras asignaciones mostrarán de forma clara cómo crear estos elementos por medio de sistemas combinatoriales.

Los chips FPGA de ALTERA incluyen FF dedicados para la implementación de circuitos, el uso de estos elementos será descrito en la asignación 4 de este laboratorio.

El archivo original de la práctica puede obtenerse a través del siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab3\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab3_VHDL.pdf) [en línea] [Última consulta 24/02/2016].

### 3.4.2 Objetivos generales

- Implementar circuitos biestables a través de lógica combinatorial.
- Aprender el uso de flip flops
- Diferenciar los tipos de eventos que provocan los cambios en los circuitos biestables.
- Reconocer un circuito síncrono y asíncrono.

### 3.4.3 Asignación 1

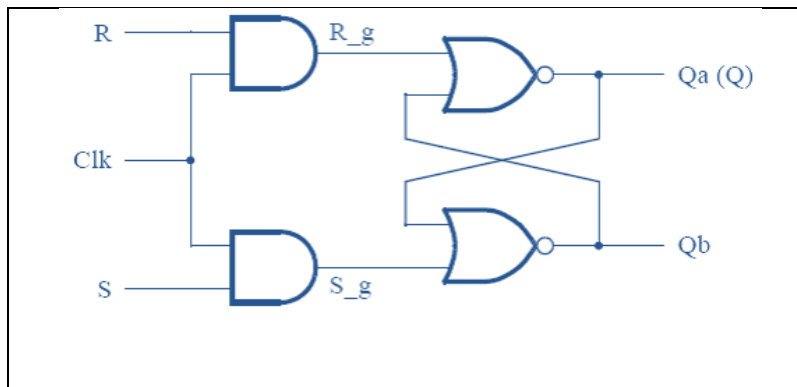
#### 3.4.3.1 Objetivo

El primer circuito a conocer será el Latch (cerrojo), básicamente es un circuito que mantiene una señal fija que cambiará únicamente cuando su activador (trigger) perciba la señal adecuada, en la práctica se muestra cómo implementar un cerrojo activado por una señal de reloj, todo esto usando únicamente compuertas básicas.

#### 3.4.3.2 Desarrollo

La figura 3.3.1 muestra un circuito latch de tipo RS gated, este tipo de circuito se comporta como un latch normal, pero con la variante que necesita un habilitador para que las entradas RS tengan efecto, en el circuito de la figura 3.3.1 este habilitador es una señal tipo clock, en la figura 3.3.2 se muestra una forma de código VHDL para implementar este circuito, el modelo presentado en esta

figura es de carácter descriptivo ya que se guardan los estados de cada una de las variables involucradas, si el chip FPGA tiene LUT's (Look Up Table) de 4 entradas solo se necesitará un LUT para implementarlo, esta forma se conoce como modelo formal y se ilustra en la figura 3.3.3a.



**Figura No. 3.3.1: Circuito Latch RS Gated.<sup>[9]</sup>**

```

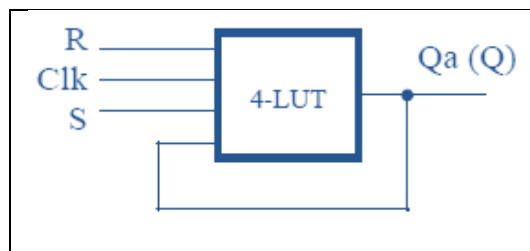
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY PART1 IS
PORT ( CLK, R, S : IN STD_LOGIC;
      Q : OUT STD_LOGIC );
END PART1;

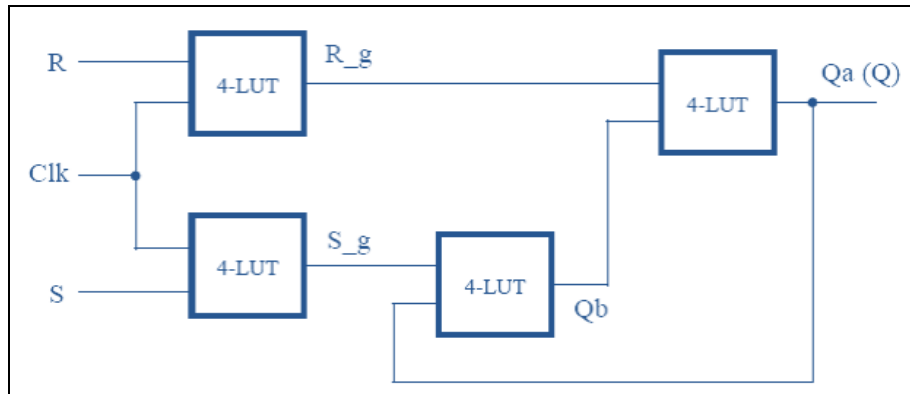
ARCHITECTURE STRUCTURAL OF PART1 IS
SIGNAL R_G, S_G, QA, QB : STD_LOGIC;
ATTRIBUTE KEEP : BOOLEAN;
ATTRIBUTE KEEP OF R_G, S_G, QA, QB : SIGNAL IS TRUE;
BEGIN
R_G <= R AND CLK;
S_G <= S AND CLK;
QA <= NOT (R_G OR QB);
QB <= NOT (S_G OR QA);
Q <= QA;
END STRUCTURAL;

```

**Figura No. 3.3.2: Logica descriptiva de un circuito latch RS gated**



**Figura No. 3.3.3a: Diferentes implementaciones de Latch RS (modelo formal).<sup>[9]</sup>**



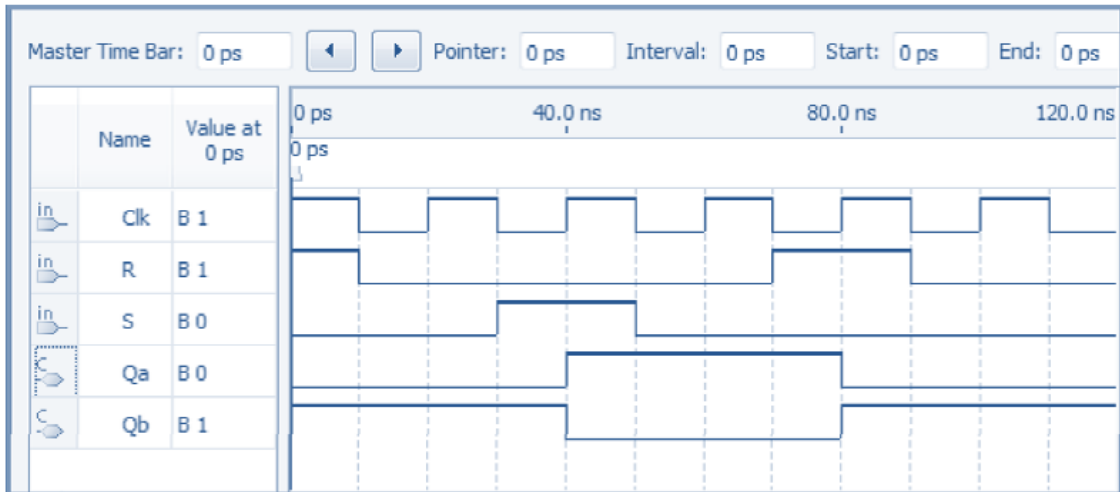
**Figura No. 3.3.3b: Diferentes implementaciones de Latch RS (modelo descriptivo).<sup>[9]</sup>**

A pesar de que la forma correcta de implementar el latch es por medio de una LUT de 4 entradas, esto no permitirá monitorear las variables internas  $R\_g$  y  $S\_g$  porque estas no son salidas que proveen una LUT, para evitar este fenómeno propio del compilador se agrega la directiva KEEP utilizando la herramienta ATTRIBUTE de VHDL, de esta forma esas salidas interna se conservarán, al usar esa directiva Quartus II entenderá que debe separar cada uno de los elementos que se declaren dentro de esa directiva, dicho esto se necesitarán 4 LUT's para implementar un Latch RS gated tal como se muestra en la figura 3.3.3b.

### 3.4.3.3 Procedimiento

Para esta práctica cree un nuevo proyecto Quartus II y realice las siguientes tareas:

1. Crear un nuevo proyecto para el Latch RS. Seleccionar como el chip objetivo del Cyclone V 5CSEMA5F31C6, que es el chip FPGA en el kit DE1-SoC.
2. Generar un archivo VHDL con el código de la figura 3.3.2 e incluirlo en el proyecto.
3. Compile el código. Utilice la herramienta **Quartus II RTL Viewer** para examinar el circuito en nivel de dispositivo producido a partir del código, y utilizar la herramienta **Technology Map Viewer** para verificar que el Latch se implementa como se muestra en la Figura 3b.
4. Simule el comportamiento del código VHDL utilizando las herramientas de simulación que proporciona Quartus II. En primer lugar, se debe crear un archivo **\*.vwf**.
5. A continuación, utilice los comandos disponibles en la herramienta Quartus II Simulation Waveform Editor para ejecutar una simulación del circuito. El procedimiento para usar esta herramienta se encuentra en el documento tutorial **Quartus II Introduction**, sitio web de ALTERA. Un ejemplo de un archivo **.vwf** se muestra en la figura 3.3.4. se inicia configurando  $Clk = 1$  y  $R = 1$ , lo que permite que la herramienta de simulación pueda inicializar todas las señales dentro del latch a valores conocidos, esto evita posibles errores, recuerde que por norma general se sabe que un FF al iniciar puede tener cualquier valor.



**Figura No. 3.3.4: Simulacion de Latch RS.<sup>[9]</sup>**

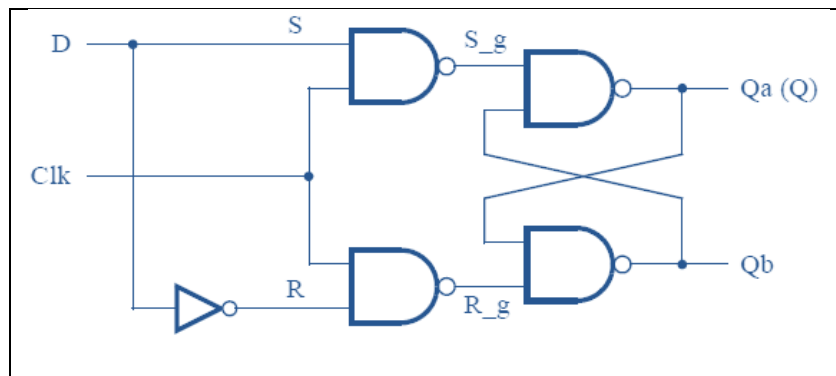
### 3.4.4 Asignación 2

#### 3.4.4.1 Objetivo

En esta práctica se implementará el latch tipo D, siguiendo como base el ejercicio anterior.

#### 3.4.4.2 Desarrollo

La figura 3.3.5 muestra un circuito Latch tipo D gated.



**Figura No. 3.3.5: Latch tipo D gated.<sup>[9]</sup>**

#### 3.4.4.3 Procedimiento

Realice las siguientes tareas para este circuito:

1. Genere un nuevo proyecto Quartus II. cree un archivo VHDL utilizando el formato de código de la figura 2 para el Latch D de la figura 3.3.5. Utilice la directiva KEEP para asegurarse que se podrán separar los elementos lógicos y de esta forma poder monitorear las señales R, S\_E, R\_g, Qa y Qb.
2. Compile el proyecto y luego utilice la herramienta **Technology Map Viewer** para examinar el circuito implementado.



3. Verifique que el circuito funcione correctamente para todas las condiciones de entrada mediante la simulación funcional. Luego examine las características de temporización del circuito mediante la simulación de tiempo.
4. Desarrolle un nuevo proyecto Quartus II que se utilizará para la implementación del Latch D en el kit DE1-SoC. Este proyecto deberá consistir en un módulo de nivel superior. Utilice el interruptor SW(0) para establecer la entrada D del circuito, y utilice SW(1) como la entrada CLK. Conecte la salida Q a LEDR(0).
5. Asigne de forma adecuada los pines como en los ejercicios anteriores.
6. Realice las pruebas necesarias con los interruptores y verifique que el circuito se comporta de forma similar a lo descrito en las simulaciones.

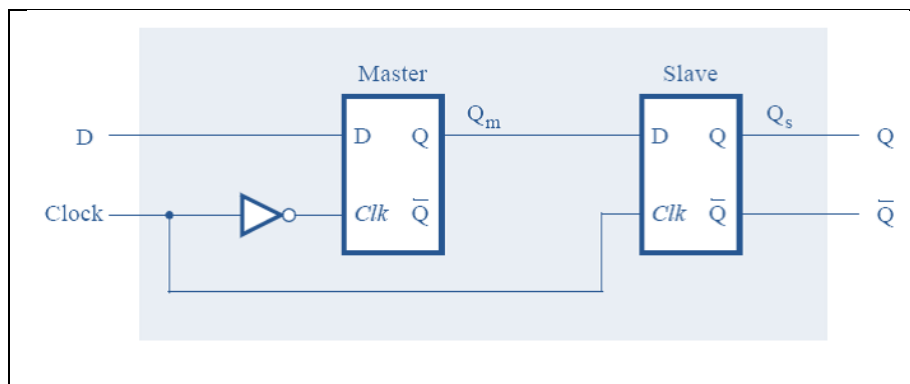
### 3.4.5 Asignación 3

#### 3.4.5.1 Objetivo

El uso de Flip flops en cascada es ampliamente utilizado en circuitos de conteo, para familiarizarse con este proceso en esta práctica se realizará una conexión maestro-esclavo con flip flop tipo D.

#### 3.4.5.2 Desarrollo

La figura 3.3.6 muestra un circuito para un FF tipo D maestro-esclavo, dentro del procedimiento se especificarán las tareas para este diagrama.



**Figura No. 3.3.6: FF tipo D en conexión maestro-esclavo. [9]**

Luego de ver este diagrama, ¿este circuito es síncrono o asíncrono?

#### 3.4.5.3 Procedimiento

Realice las siguientes tareas para este circuito:

1. Genere un nuevo proyecto Quartus II. cree un archivo VHDL utilizando el formato de código de la figura 3.3.2 para crear un componente que genere las dos instancias necesarias para desarrollar el circuito.
2. Compile el proyecto y luego utilice la herramienta **Technology Map Viewer** para examinar el circuito implementado.

3. Verifique que el circuito funcione correctamente para todas las condiciones de entrada mediante la simulación funcional. Luego examine las características de temporización del circuito mediante la simulación de tiempo.
4. Desarrolle un nuevo proyecto Quartus II que se utilizará para la implementación del FF tipo D en el kit DE1-SoC. Este proyecto deberá consistir en un módulo de nivel superior. Utilice el interruptor SW(0) para establecer la entrada D del circuito, y utilice SW(1) como la entrada CLK. Conecte la salida Q a LEDR(0).
5. Asigne de forma adecuada los pines como en los ejercicios anteriores.
6. Realice las pruebas necesarias con los interruptores y verifique que el circuito se comporta de forma similar a lo descrito en las simulaciones.

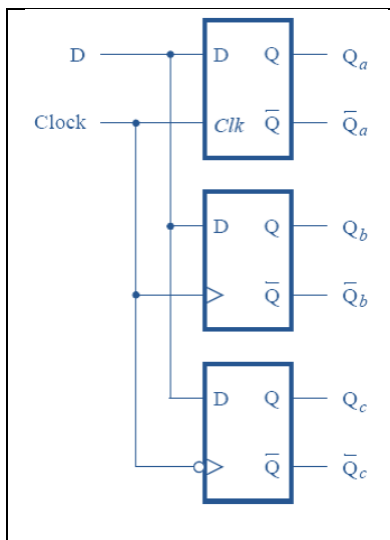
### 3.4.6 Asignación 4

#### 3.4.6.1 Objetivos

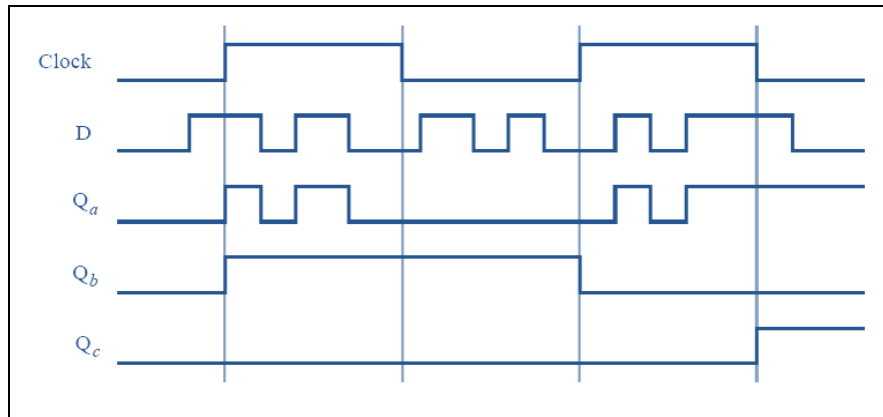
En esta práctica se busca entender los eventos que logran el cambio en las salida de un cerrojo, para tal motivo se crearán tres latches del mismo tipo pero la entrada de activación responderá a tres eventos diferentes, posterior a la creación de los código, la diferencia será notable en el diagrama de tiempo correspondiente al circuito.

#### 3.4.6.2 Desarrollo

La figura 3.3.7 muestra un circuito con tres tipos diferentes de elementos de almacenamiento: un latch tipo D gated, un latch tipo D de flanco de subida y un latch tipo D de flanco de bajada, en la sección 3.4.6.3 se indicarán las instrucciones para completar esta práctica, tomando como referencia este circuito:



a) circuito



b) diagrama de tiempo

**Figura No. 3.3.7:a) Tipos de latches y sus respectivos diagramas de tiempo (b).<sup>[9]</sup>**

Para este circuito realice las siguientes tareas:

1. Cree un nuevo proyecto Quartus II con las especificaciones para ser implementado en el kit DE1 SoC.
2. Genere un archivo VHDL que contenga los tres elementos de almacenamiento. Para esta parte ya no debe usar la directiva KEEP. La Figura 3.3.8 da un formato de código VHDL que especifica el comportamiento del latch tipo D gated descrito en la figura 3.3.5. Como se había mencionado previamente este tipo de elementos puede ser generado en un solo LUT de 4 entradas. Use un formato similar de este código para especificar todos los elementos de la figura 3.3.7a, se recomienda leer sobre el atributo EVENT ya que será muy útil en la implementación de los Latch por flancos.
3. Compile el código y utilice la herramienta **Technology Map Viewer** para examinar el circuito implementado. Verifique que el latch utiliza una LUT y que los FF se implementan utilizando los elementos por defecto que proporciona el chip FPGA.
4. Genere un archivo .vwf, luego especifique las entradas y salidas del circuito. asigne las entradas D y Clk como se indica en la figura 3.3.7. Use la simulación funcional para obtener las tres señales de salida. Observe el comportamiento de los tres elementos de almacenamiento y verifique que su comportamiento sea similar al de la figura 3.3.7.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY LATCH IS
PORT ( D, CLK : IN STD_LOGIC;
      Q : OUT STD_LOGIC );
END LATCH ;

ARCHITECTURE BEHAVIOR OF LATCH IS
BEGIN
PROCESS ( D, CLK )
BEGIN

```

```
IF CLK = '1' THEN
    Q <= D ;
ENDIF;
ENDPROCESS;
END BEHAVIOR ;
```

**Figura No. 3.3.8: Formato de código que describe el comportamiento de latch tipo D.**

## 3.4.7 Asignación 5

### 3.4.7.1 Objetivos

Teniendo los conocimientos sobre registros, en esta práctica se utilizarán para expandir la capacidad base de la tarjeta, de modo que sea capaz de operar números de 8 bits programados a través de los 10 interruptores disponibles.

### 3.4.7.2 Desarrollo

Se desea mostrar el valor hexadecimal de un número de 8 bits A en el dos displays de 7 segmentos HEX3 y HEX2. También se quiere mostrar el valor hexadecimal de un número de 8 bits B en dos displays de 7 segmentos HEX1 y HEX0. Los valores de A y B son entradas de un circuito proporcionadas por medio de los interruptores SW(0 a 7). Para introducir los valores de A y B, primero se establecen los valores deseados de A, luego de esto se deben almacenar estos valores en un registro, con los datos ya almacenados se deben cambiar los interruptores para el valor deseado de B. Por último, utilice un sumador para generar la suma aritmética  $S = a + B$ , y mostrar esta suma en los displays de 7 segmentos HEX5 y HEX4. El acarreo de salida producido por el sumador se debe mostrar en LEDR (0).

### 3.4.7.3 Procedimiento

Para completar esta asignación se deben seguir estos pasos:

1. Genere un nuevo proyecto Quartus II que se utilizará para el circuito, incluya las especificaciones correctas para que puedaimplementar en el kit DE1-SoC.
2. Escriba un código con VHDL que proporcione la funcionalidad requerida. Utilice KEY0 como un reset asíncrono activo-bajo y utilice KEY1 como una entrada de reloj que será la encargada de mandar la orden para que los registros almacenen datos.
3. Asigne de forma adecuada los pines como en los ejercicios anteriores.
4. Realice las pruebas necesarias con los interruptores y pulsadores, luego verifique que el circuito se comporta según la descripción.

## 3.5 Laboratorio No. 4: Contadores

### 3.5.1 Introducción

La necesidad de contar fue lo que derivó en la creación de los sistemas numéricos, de modo que siempre se tiene la necesidad de cuantificar todo evento, y para lograrlo se utilizan contadores que pueden ser programados para situaciones específicas.

El propósito de este ejercicio es aprender a diseñar contadores. Los circuitos se implementarán en un kit DE1-SoC de Altera.

El estudiante debe tener un conocimiento básico sobre contadores y debe estar familiarizado con funciones de VHDL para implementar varios tipos de flip-flops.

El archivo original de la práctica puede descargarse desde el siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab4\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab4_VHDL.pdf) [en línea] [Última consulta 24/02/2016].

### 3.5.2 Objetivos Generales

- Aprender a crear contadores en VHDL a través de flip flops en cascada.
- Utilizar módulos de suma para contar eventos.
- Hacer uso de librerías parametrizadas incluidas en Quartus II para la creación de contadores.
- Entender el uso de un divisor de frecuencia para crear contadores lentos a partir de generadores de señales a alta frecuencia.

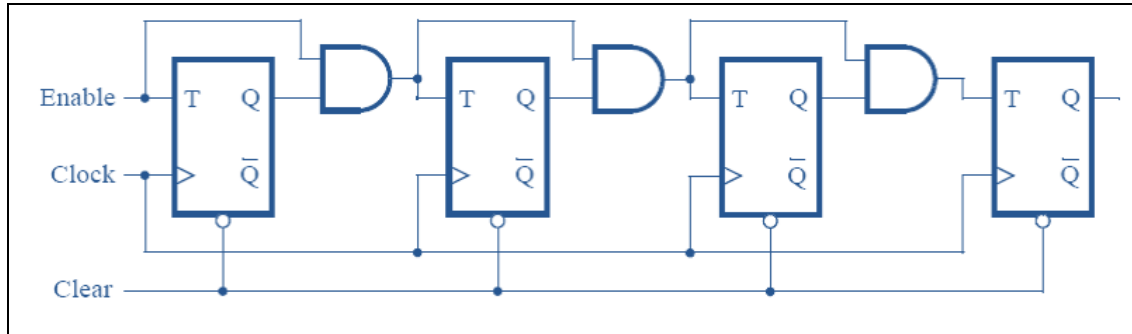
### 3.5.3 Asignación 1

#### 3.5.3.1 Objetivo

Un contador tiene su forma básica en la conexión de flip flops en cascada, cada flip flop representa un bit disponible, en esta práctica se aprenderá como conectar en cascada flip flops tipo T para lograr un contador de 8 bits síncrono.

#### 3.5.3.2 Desarrollo

Considere el circuito en la figura 3.4.1. Es un contador síncrono 4-bit que utiliza cuatro FF tipo T activados por flanco de subida, de tal forma que cuando se registre un cambio de 0 a 1 en la señal Clk se realizará un conteo, siempre y cuando la entrada Enable este en alto. El contador se restablece a 0 cuando la entrada Clear es 0 y llega un nuevo flanco de subida, en esta sección se requiere implementar un contador de 8 bits de este tipo.



**Figura No. 3.4.1: Contador síncrono de 4 bits.<sup>[10]</sup>**

El código base de esta asignación se muestra en la figura 3.4.2:

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY L4_PART1_AUP IS
PORT ( SW      : INSTD_LOGIC_VECTOR(1DOWNTO0) ;
      KEY     : INSTD_LOGIC_VECTOR(0TO0) ;
      HEX1, HEX0 : OUTSTD_LOGIC_VECTOR(0TO6) ) ;
END L4_PART1_AUP;

ARCHITECTURE BEHAVIOR OF L4_PART1_AUP IS
COMPONENT FFT
PORT (CLOCK, CLRN, T : INSTD_LOGIC;
      Q           : OUTSTD_LOGIC);
ENDCOMPONENT;

COMPONENT HEXADEC
PORT (ENT      : INSTD_LOGIC_VECTOR(3DOWNTO0) ;
      HEXA     : OUTSTD_LOGIC_VECTOR(0TO6) ) ;
ENDCOMPONENT;

SIGNAL T, Q : STD_LOGIC_VECTOR(7DOWNTO0);
BEGIN
T(0) <= SW(1);
T(1) <= T(0) AND Q(0);
T(2) <= T(1) AND Q(1);
T(3) <= T(2) AND Q(2);
T(4) <= T(3) AND Q(3);
T(5) <= T(4) AND Q(4);
T(6) <= T(5) AND Q(5);
T(7) <= T(6) AND Q(6);

F0: FFT PORTMAP (KEY(0), SW(0), T(0), Q(0));
F1: FFT PORT MAP (KEY(0), SW(0), T(1), Q(1));
F2: FFT PORTMAP (KEY(0), SW(0), T(2), Q(2));
F3: FFT PORTMAP (KEY(0), SW(0), T(3), Q(3));
F4: FFT PORTMAP (KEY(0), SW(0), T(4), Q(4));
F5: FFT PORTMAP (KEY(0), SW(0), T(5), Q(5));
F6: FFT PORTMAP (KEY(0), SW(0), T(6), Q(6));
F7: FFT PORTMAP (KEY(0), SW(0), T(7), Q(7));

```

```

H0: HEXADEC PORTMAP(Q(3DOWNTO0), HEX0);
H1: HEXADEC PORTMAP(Q(7DOWNTO4), HEX1);
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY FFT IS
PORT (CLOCK, CLRN, T :INSTD_LOGIC;
      Q      :OUTSTD_LOGIC);
END FFT;

ARCHITECTURE BEHAVIOR OF FFT IS
SIGNAL QESTADO :STD_LOGIC;
BEGIN
PROCESS (CLOCK, CLRN)
BEGIN
IF (CLRN = '0') THEN
    QESTADO <= '0';
ELSIF (CLOCK'EVENTAND CLOCK = '1') THEN
IF (T = '1') THEN
    QESTADO <=NOT (QESTADO);
ELSE
    QESTADO <= QESTADO;
END IF;
END IF;
Q <= QESTADO;
ENDPROCESS;
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY HEXADEC IS
PORT (ENT      :INSTD_LOGIC_VECTOR(3DOWNTO0);
      HEXA     :OUTSTD_LOGIC_VECTOR(0TO6));
END HEXADEC;

ARCHITECTURE BEHAVIOR OF HEXADEC IS
BEGIN

PROCESS (ENT)
BEGIN
CASE ENT IS
WHEN"0000"=> HEXA <="0000001";
WHEN"0001"=> HEXA <="1001111";
WHEN"0010"=> HEXA <="0010010";
WHEN"0011"=> HEXA <="0000110";
WHEN"0100"=> HEXA <="1001100";
WHEN"0101"=> HEXA <="0100100";
WHEN"0110"=> HEXA <="0100000";
WHEN"0111"=> HEXA <="0001111";
WHEN"1000"=> HEXA <="0000000";
WHEN"1001"=> HEXA <="0001100";
WHEN"1010"=> HEXA <="0001000";
WHEN"1011"=> HEXA <="1100000";

```

```

WHEN"1100"=> HEXA <="0110001";
WHEN"1101"=> HEXA <="1000010";
WHEN"1110"=> HEXA <="0110000";
WHENOTHERS=> HEXA <="0111000";
ENDCASE;
ENDPROCESS;

END BEHAVIOR;

```

**Figura No 3.4.2: Código de Asignación 1**

### 3.5.3.3 Procedimiento

Para completar esta asignación se deben seguir estos pasos:

1. Cree un proyecto para la DE1-Soc y compruebe que el código VHDL descrito en la figura 3.4.2 define un contador de 8 bits utilizando la estructura representada en la figura 3.4.1.
2. Compile el circuito. ¿Cuántos elementos lógicos (LE) se utilizan para implementar el circuito?
3. Simular el circuito para verificar su exactitud.
4. Asigne dentro del código VHDL un pulsador KEY0 como la entrada Clk, los interruptores SW(0) y SW(1) como Enable y Clear respectivamente, en dos display de 7 segmentos HEX0 y HEX1 se debe mostrar el conteo hexadecimal que realiza el circuito.
5. Descargue el circuito en el chip FPGA y pruebe su funcionalidad.
6. Implemente una versión de 4 bits del mismo circuito y utilice el **Quartus II RTL Viewer** para ver cómo el software Quartus II hace la síntesis del circuito. ¿Cuáles son las diferencias en comparación con la figura 3.4.1?

## 3.5.4 Asignación 2

### 3.5.4.1 Objetivo

En esta sección se creará un contador en base a suma de bits por registros, esto es posible gracias a los componentes de las librerías IEEE.STD\_UNSIGNED, la cual debe ser declarada en la sección de librerías.

### 3.5.4.2 Desarrollo

Como ya se dijo el contador se implementará el uso de un registro y consecutivamente añade 1 a su valor. Esto se puede lograr mediante la siguiente instrucción VHDL:

```
Q <= Q +1;
```

Ecuación No. 3.4: Modelo para un sumador de bits

### 3.5.4.3 Procedimiento

Para completar esta asignación se deben seguir estos pasos:

1. Compile una versión de 16 bits de este contador y determine el número de LE's necesario.
2. Utilice el **Quartus II RTL Viewer** para ver la estructura de esta aplicación y haga los comentarios respectivos sobre las diferencias con el diseño de la asignación 1.



3. Implemente el contador en el kit DE1-SoC, utilice los display HEX0 hasta el HEX3 para mostrar el valor del conteo siempre en valores hexadecimales.

### 3.5.5 Asignación 3

#### 3.5.5.1 Objetivo

Quartus II cuenta con módulos ya definidos, para utilizarlos es necesario saber utilizar las LPM (Library of Parameterized Modules), en esta práctica se aprenderá como simplificar un circuito contador mediante esa herramienta.

#### 3.5.5.2 Desarrollo

Utilice un LPM de la Biblioteca de módulos con parámetros para implementar un contador de 16 bits. Seleccione los elementos correctos del LPM para crear un diseño coherente tal como el anterior, es decir, con Enable y Clear síncrono. ¿Cómo se compara esta versión con los diseños anteriores?

#### 3.5.5.3 Procedimiento

El tutorial *Using the Library of Parameterized Modules (LPM)* explica el uso de los LPM's. Se puede encontrar en el siguiente link:

[ftp://ftp.altera.com/up/pub/Altera\\_Material/11.1/Tutorials/VHDL/Using\\_Library\\_Modules.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Tutorials/VHDL/Using_Library_Modules.pdf) [en línea] [ultima consulta]

Realice los pasos descritos en el tutorial para crear un contador y simule los resultados.

### 3.5.6 Asignación 4

#### 3.5.6.1 Objetivo

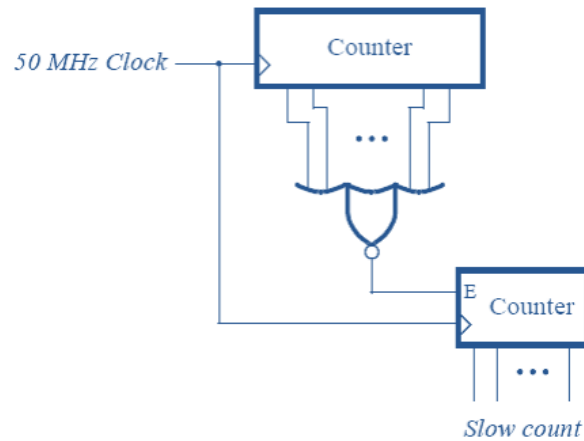
Esta práctica tiene por meta crear un divisor de frecuencia basándose en un contador de gran tamaño, este divisor debe ser capaz de lanzar una señal de habilitación en un intervalo de un segundo aproximadamente, realice los cálculos adecuados considerando que el generador de señal es de 50 MHz.

#### 3.5.6.2 Desarrollo

Atendiendo la explicación previa se debe crear un circuito que muestre sucesivamente dígitos del 0 al 9 en un display de 7 segmentos HEX0. Cada dígito se debe mostrar durante aproximadamente un segundo. Utilice un contador para determinar los intervalos de un segundo. El contador debe ser incrementado por la señal de reloj de 50 MHz proporcionada en el kit DE1-SoC (lea el manual de usuario para encontrar el pin respectivo, así como su nomenclatura). No intente derivar o añadir cualquier otra señal de reloj en el diseño y asegúrese que los FF en el circuito estén conectados de forma sincronizada directamente a la señal de reloj de 50 MHz.

Un diseño parcial del circuito requerido se muestra en la figura 3.4.3, en esta se observa cómo un contador con una gran anchura de bits se puede utilizar para producir una señal de habilitación de

un contador más pequeño. La velocidad a la que se incrementa el contador pequeño puede ser controlada por una elección apropiada de bits en el contador más grande.



**Figura No. 3.4.3: Diseño parcial para un contador lento.<sup>[10]</sup>**

### 3.5.6.3 Procedimiento

Para esta práctica debe cumplir con estas tareas:

1. Cree un proyecto que tenga como objetivo la tarjeta DE1-SoC, luego diseñe un código que satisfaga el circuito 3.4.3.
2. Compile el proyecto.
3. Simule los resultados y compruebe que el circuito cumple los requerimientos.
4. Asigne los pines necesarios para que el código funcione en la tarjeta DE1-SoC.
5. Compile nuevamente y descargue los datos a la tarjeta.
6. Realice las pruebas necesarias.

## 3.5.7 Asignación 5

### 3.5.7.1 Objetivo

Retomando el circuito creado en la sección 3.2.7, en donde se rotaba una palabra en 3 display haciendo uso de 2 interruptores como variables de control; este código se reciclará para esta sección con la variante que el control esta vez será un contador de modulo 2, el cual rotará la palabra cada segundo aproximadamente.

### 3.5.7.2 Desarrollo

Diseñe e implemente un circuito que muestra la palabra dE1 en el tres displays de 7 segmentos HEX2, HEX1 y HEX0. Esta palabra debe desplazarse de derecha a izquierda en intervalos de aproximadamente un segundo. Los patrones que se deben mostrar en intervalos sucesivos se pueden ver en la Tabla 3.4.1. Hay muchas maneras de diseñar el circuito requerido. Una solución es reutilizar el código VHDL diseñado en el ejercicio de Laboratorio 1, la parte 5 para ser más

específicos. Si se usa de ese código, el principal cambio que se debe hacer es reemplazar los dos interruptores que se utilizan para controlar la rotación por un contador de 2 bits que incrementa a intervalos de un segundo.

Ciclo de reloj	HEX2	HEX1	HEX0
0	d	E	1
1	E	1	d
2	1	d	E

Tabla No. 3.4.1: Rotación de una palabra en 3 display.

### 3.5.7.3 Procedimiento

1. Cree un proyecto para la tarjeta DE1-SoC y diseñe el código necesario para satisfacer la asignación.
2. Recuerde que puede reciclar el circuito de la sección 3.2.7, la señal de reloj la debe proveer la tarjeta a través de un generador de 50 MHz, para mostrar los datos se utilizaran los display HEX0, HEX1, HEX2.
3. Asigne los pines necesarios.
4. Compile y simule el circuito.
5. Descargue el archivo .SOF a la tarjeta y compruebe que el funcionamiento sea el adecuado.

## 3.5.8 Asignación 6

### 3.5.8.1 Objetivo

Siguiendo con el reciclaje de códigos esta vez se retomará el circuito de la sección 3.2.8, se mantendrá la variante aplicada a la sección anterior, la rotación de la palabra será por medio de un contador de modulo 3 y los datos se mostrarán en los 6 display disponibles.

### 3.5.8.2 Desarrollo

Mejore el circuito de la sección anterior de modo que pueda rotar la palabra dE1 en seis displays de 7 segmentos desde HEX5 hasta HEX0, los patrones que se deben mostrar se encuentran en la Tabla 3.4.2.

Ciclo de reloj	HEX5	HEX4	HEX3	HEX2	HEX1	HEX0
0	-	-	-	d	E	1
1	-	-	d	E	1	-
2	-	d	E	1	-	-
3	d	E	1	-	-	-
4	E	1	-	-	-	d
5	1	-	-	-	d	E

Tabla No. 3.4.2: Rotación de una palabra en 6 display.

### 3.5.8.3 Procedimiento

1. Cree un proyecto para la tarjeta DE1-SoC y diseñe el código necesario para satisfacer la asignación.
2. Recuerde que puede reciclar el circuito de la sección 3.2.8, la señal de reloj la debe proveer la tarjeta a través de un generador de 50 MHz, para mostrar los datos se utilizaran los display HEX0, HEX1, HEX2, HEX3, HEX4, HEX5.
3. Asigne los pines necesarios.
4. Compile y simule el circuito.
5. Descargue el archivo .SOF a la tarjeta y compruebe que el funcionamiento sea el adecuado.

## 3.6 Laboratorio No. 5: Temporizadores y Reloj en tiempo real

### 3.6.1 Introducción

Lograr señales con tiempos precisos es algo fundamental en los sistemas digitales, en un principio cuando se utilizaba el LM555 para generar señales de onda cuadrada se tenía un ligero error con el Duty Cycle lo que al final iba acumulando errores en segundos.

Para corregir estos detalles en la actualidad se utilizan cristales osciladores con frecuencias fijas, el inconveniente que surge con estos cristales es que son de frecuencias muy altas y es necesario utilizar divisores de frecuencia para llegar a determinadas frecuencias.

En el ejercicio anterior se logró crear un contador de baja frecuencia, este sería el principio de los relojes de tiempo real.

El archivo original de la práctica se puede obtener a través del siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab5\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab5_VHDL.pdf) [en línea] [ultima consulta 24/02/2016].

### 3.6.2 Objetivos Generales

- Implementar divisores de frecuencia para reducir la señal que brindan los cristales internos de 50 MHz.
- Crear contadores de modulo K (genericos) los cuales permitan elegir el limite del contador únicamente con el cambio de una variable.
- Desarrollar un contador que alcance aproximadamente un segundo de periodo.
- Realizar un reloj con tres unidades de tiempo (horas, minutos, segundos) que funcione en tiempo real.

### 3.6.3 Preliminares

En VHDL se puede describir un contador de tamaño variable utilizando una declaración GENERIC, un ejemplo de un contador de n bits se muestra en la Figura 3.5.1.

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY COUNTER IS
GENERIC ( N          : NATURAL := 4 );
PORT ( CLOCK        : IN STD_LOGIC;
      RESET N       : IN STD_LOGIC;
      Q              : OUT STD_LOGIC_VECTOR (N-1 DOWNTO 0) );
END ENTITY;
ARCHITECTURE BEHAVIOR OF COUNTER IS
SIGNAL VALUE : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
BEGIN
PROCESS (CLOCK, RESET N)
BEGIN
IF (RESET N = '0') THEN
VALUE <= (OTHERS => '0');
ELSIF ((CLOCK'EVENT) AND (CLOCK = 1)) THEN
```

```

VALUE<=VALUE+1;
ENDIF
ENDPROCESS
Q <=VALUE;
END BEHAVIOR;

```

**Figura No. 3.5.1: Código VHDL para un contador de n-bits. [11]**

El parámetro n especifica el número de bits en el contador. Un valor particular de este parámetro se define mediante una instrucción GENERIC MAP. Por ejemplo, un contador de 8 bits se puede especificar como:

```

EIGHT_BIT: COUNTER
GENERICMAP ( N =>8)
PORTMAP (CLOCK, RESET N, Q);

```

**Figura No. 3.5.2: Definición de un componente counter.[11]**

Mediante el uso de parámetros se pueden crear instancias de contadores con diferentes tamaños dentro de un circuito lógico, sin tener que crear un nuevo módulo para cada contador.

## 3.6.4 Asignación 1

### 3.6.4.1 Objetivo

Lo primero es aprender a crear un contador modulo K, para esto se provee un código en la figura 3.5.3, queda una aproximación a la sintaxis necesaria para crear el circuito.

### 3.6.4.2 Desarrollo

El contador de módulo-k se creará mediante la modificación del diseño de un contador de 8 bits. El contador debe contar de 0 a (k-1). Cuando el contador alcanza el valor k-1, el siguiente valor del contador debe ser 0. Se Incluye una salida llamada *rollover*, esta salida será '1' durante el ciclo de reloj en el cual el contador tenga el valor de K-1.

```

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY L5_PART1_AUP IS
PORT (KEY :INSTD_LOGIC_VECTOR (1DOWNTO0) ;
      LEDR :OUTSTD_LOGIC_VECTOR (9DOWNTO0) ) ;
END L5_PART1_AUP;

ARCHITECTURE BEHAVIOR OF L5_PART1_AUP IS
COMPONENT COUNTER
GENERIC ( N :NATURAL:=4;
          M :NATURAL:=16) ;
PORT ( CLOCK :INSTD_LOGIC;
      RESET_N :INSTD_LOGIC;
      Q :OUTSTD_LOGIC_VECTOR ((N -1)DOWNTO0) ;
      ROLLOVER :OUTSTD_LOGIC) ;
ENDCOMPONENT;

```

```

BEGIN
C1: COUNTER GENERICMAP (N =>8, M =>200) PORTMAP (KEY (1), KEY (0),
LEDR (7DOWNTO0), LEDR (9));
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY COUNTER IS
GENERIC( N :NATURAL:=4;
M :NATURAL:=16);
PORT( CLOCK :INSTD_LOGIC;
RESET_N :INSTD_LOGIC;
Q :OUTSTD_LOGIC_VECTOR((N -1)DOWNTO0);
ROLLOVER :OUTSTD_LOGIC);
END COUNTER;

ARCHITECTURE BEHAVIOR OF COUNTER IS
SIGNAL VALUES :STD_LOGIC_VECTOR((N -1)DOWNTO0);
BEGIN
PROCESS (CLOCK, RESET_N)
BEGIN
IF (RESET_N = '0') THEN
VALUES <= (OTHERS=> '0');
ELSIF ((CLOCK'EVENT) AND (CLOCK = '1' )) THEN
IF (VALUES = (M-1)) THEN
VALUES <= (OTHERS=> '0');
ROLLOVER <= '1';
ELSE
VALUES <= VALUES +1;
ROLLOVER <= '0';
ENDIF;
ENDIF;
ENDPROCESS;
Q <= VALUES;
END BEHAVIOR;

```

*Figura No. 3.5.3: Contador de modulo K.*

### 3.6.4.3 Procedimiento

Para este circuito realice las siguientes tareas:

1. Desarrolle un nuevo proyecto Quartus II que se utilizará para implementar el circuito en el Kit DE1-SoC.
2. Traslade el código de la figura 3.5.3 a un archivo VHDL, el circuito debe utilizar KEY0 pulsador como un reset asíncrono y KEY1 como una entrada manual reloj. El contenido del contador se debe mostrar en LEDR(7-0), y la señal rollover se deben mostrar en LEDR9.
3. Incluya el archivo VHDL en el proyecto y compile el circuito.
4. Simule el circuito diseñado para verificar su funcionalidad.
5. Asigne los pines de la FPGA para conectarse a los elementos del circuito, recuerde que puede importar el archivo DE1-SoC.qsf.
6. Vuelva a compilar el circuito y descarguelo en el chip FPGA.

7. Compruebe que el circuito funciona correctamente mediante la observación de LEDR.

## 3.6.5 Asignación 2

### 3.6.5.1 Objetivo

Teniendo la experiencia del modulo K, el siguiente paso es crear un contador con periodo de 1 segundo que funcione como subcircuito de control para otras tareas.

En la siguiente práctica la meta será mostrar dígitos BCD en los displays teniendo como variable de control, un reloj de 1 segundo de periodo.

### 3.6.5.2 Desarrollo

Usando el contador modulo de la Parte I como un subcircuito, implemente un contador BCD de 3 dígitos. Muestre el contenido del contador en los displays de 7 segmentos, HEX2, HEX1 y HEX0.

### 3.6.5.3 Procedimiento

Conecte todos los contadores del circuito a la señal de reloj de 50 MHz del kit DE1-SoC, haciendo que el contador BCD incremente a intervalos de un segundo. Utilice el pulsador KEY0 para reiniciar el contador BCD a 0.

## 3.6.6 Asignación 3

### 3.6.6.1 Objetivo

Crear un reloj de tiempo real con tres unidades de tiempo, para que el proceso de prueba sea rápido se utilizarán los minutos, segundos y centésimas de segundo.

### 3.6.6.2 Desarrollo

Diseñar e implementar un circuito en el Kit DE1-SoC que trabaje como un reloj en tiempo real. Debe mostrar los minutos (de 0 a 59) en HEX5 y HEX4, los segundos (de 0 a 59) en HEX3 a HEX2 y las centésimas de segundo (de 0 a 99) en HEX1 y HEX0. Utilice los interruptores SW(7-0) para preestablecer los minutos mostrados por el reloj cuando se pulse KEY1. El reloj se debe detener cada vez que se pulsa KEY0 y debe continuar cuando se libere KEY0.

### 3.6.6.3 Asignación Extra

Teniendo como base el ejercicio anterior, cree un reloj de 24 horas.



## 3.7 Laboratorio No. 6: Sumadores, Restadores y Multiplicadores

### 3.7.1 Introducción

La suma y la resta son la base de operaciones más complejas como la multiplicación y la división, en circuitos lógicos es importante entender como principio la suma bit a bit, luego es bueno entender los números como complemento a 2 ya que estos reflejan un binario negativo.

El archivo original de la práctica puede obtenerse a través del siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab6\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab6_VHDL.pdf) [en línea] [última consulta 24/02/2016].

### 3.7.2 Objetivos Generales

- Crear circuitos acumuladores con la ayuda de registros de bits.
- Entender el proceso de una resta, auxiliándose con la operación suma y los números en complemento a 2.
- Desarrollar un multiplicador utilizando sumadores en cascada.
- Aprender el uso de registros para crear multiplicadores más eficientes.
- Implementar un árbol de sumas.

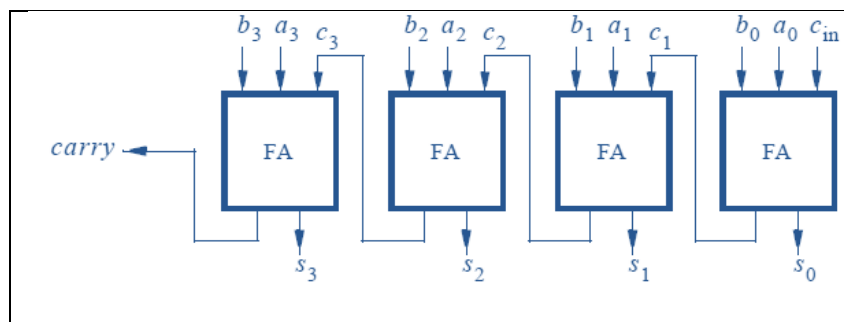
### 3.7.3 Asignación 1

#### 3.7.3.1 Objetivo

La meta de esta práctica es aplicar los conocimientos previos sobre los sumadores para crear un circuito capaz de realizar sumas y mantener en registros los resultados.

#### 3.7.3.2 Desarrollo

Considere nuevamente el circuito sumador ripple-carry de cuatro bits utilizado en el laboratorio 2; su diagrama se describe en la Figura 3.6.1.



**Figura No. 3.6.1: sumador ripple-carry de cuatro bits.** [12]

Este circuito se puede implementar utilizando un signo '+' en VHDL. Por ejemplo, el siguiente fragmento de código suma números de n-bits A y B para producir salidas Sum y Carry:

```
LIBRARY IEEE ;  
USE IEEE.STD_LOGIC_1164.ALL ;  
USE IEEE.STD_LOGIC_ARITH.ALL ;
```

```

USEIEEE.STD_LOGIC_SIGNED.ALL;
...
SIGNAL SUM :STD_LOGIC_VECTOR(N-1DOWNTO0);
...
SUM <= A + B;

```

**Figura No. 3.6.2: Sumador acumulador.** [12]

Utilice esta estructura para implementar el circuito mostrado en la Figura 3.6.4. Este circuito, que a menudo se le llama *acumulador*, se utiliza para sumar el valor de una entrada A, a sí misma de forma repetida. El circuito incluye una salida *Carry* desde el sumador, así como una señal de salida *Overflow*. Si la entrada A es considerada como un número complemento a 2, entonces *overflow* deberá ser ajustado a 1 en el caso que la suma en la salida producida no represente un resultado correcto en formato complemento a 2.

En la figura 3.6.3 se ilustra el código para implementar el circuito de la figura 3.6.4.

```

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.STD_LOGIC_ARITH.ALL;
USEIEEE.STD_LOGIC_SIGNED.ALL;

ENTITY L6_PART1_AUP IS
PORT (SW          :INSTD_LOGIC_VECTOR (7DOWNTO0) ;
      KEY         :INSTD_LOGIC_VECTOR (1DOWNTO0) ;
      LEDR        :OUTSTD_LOGIC_VECTOR (8DOWNTO0) );
END L6_PART1_AUP;

ARCHITECTURE STRUCTURE OF L6_PART1_AUP IS
COMPONENT SUMADOR
GENERIC (N :NATURAL:=8) ;
PORT (AI, AO     :INSTD_LOGIC_VECTOR ((N)DOWNTO0) ;
      SUMA       :OUTSTD_LOGIC_VECTOR ((N)DOWNTO0) );
ENDCOMPONENT;

COMPONENT REGISTROS
GENERIC (N :NATURAL:=8) ;
PORT (AIN :INSTD_LOGIC_VECTOR ((N)DOWNTO0) ;
      CLOCK, RESET :INSTD_LOGIC;
      Q    :OUTSTD_LOGIC_VECTOR ((N)DOWNTO0) );
ENDCOMPONENT;

COMPONENT FFD
PORT (D, CLK :INSTD_LOGIC;
      Q      :OUTSTD_LOGIC);
ENDCOMPONENT;

SIGNAL AR, SR, S :STD_LOGIC_VECTOR (8DOWNTO0) ;
SIGNAL CARRY :STD_LOGIC;
BEGIN
REGISTER1: REGISTROS GENERICMAP (N =>8) PORTMAP ('0' & SW (7DOWNTO0) ,
KEY (1) , KEY (0) , AR) ;
REGISTER2: REGISTROS GENERICMAP (N =>8) PORTMAP (S, KEY (1) , KEY (0) ,SR) ;

```

```

S0: SUMADOR GENERICMAP (N =>8) PORTMAP (AR, SR, S);
REGISTER3: FFD PORTMAP (S(8), KEY(1), CARRY);
LEDR(8) <= CARRY;
LEDR(7DOWNTO0) <= SR(7DOWNTO0);
END STRUCTURE;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.STD_LOGIC_ARITH.ALL;
USEIEEE.STD_LOGIC_SIGNED.ALL;

ENTITY SUMADOR IS
GENERIC (N :NATURAL:=8);
PORT (AI, AO      :INSTD_LOGIC_VECTOR ((N)DOWNTO0);
        SUMA      :OUTSTD_LOGIC_VECTOR ((N)DOWNTO0));
END SUMADOR;

ARCHITECTURE BEHAVIOR OF SUMADOR IS
BEGIN
SUMA <= AI + AO;
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.STD_LOGIC_ARITH.ALL;
USEIEEE.STD_LOGIC_SIGNED.ALL;

ENTITY REGISTROS IS
GENERIC (N :NATURAL:=8);
PORT (AIN :INSTD_LOGIC_VECTOR ((N)DOWNTO0);
        CLOCK, RESET      :INSTD_LOGIC;
        Q      :OUTSTD_LOGIC_VECTOR ((N)DOWNTO0));
END REGISTROS;

ARCHITECTURE BEHAVIOR OF REGISTROS IS
BEGIN
PROCESS (CLOCK, RESET)
BEGIN
IF (RESET = '0') THEN
    Q <= (OTHERS=> '0');
ELSIF (CLOCK'EVENTAND CLOCK = '1') THEN
    Q <= AIN;
ENDIF;
ENDPROCESS;
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY FFD IS
PORT (D, CLK      :INSTD_LOGIC;
        Q      :OUTSTD_LÖGIC);
END FFD;

ARCHITECTURE BEHAVIOR OF FFD IS
BEGIN
PROCESS (CLK)

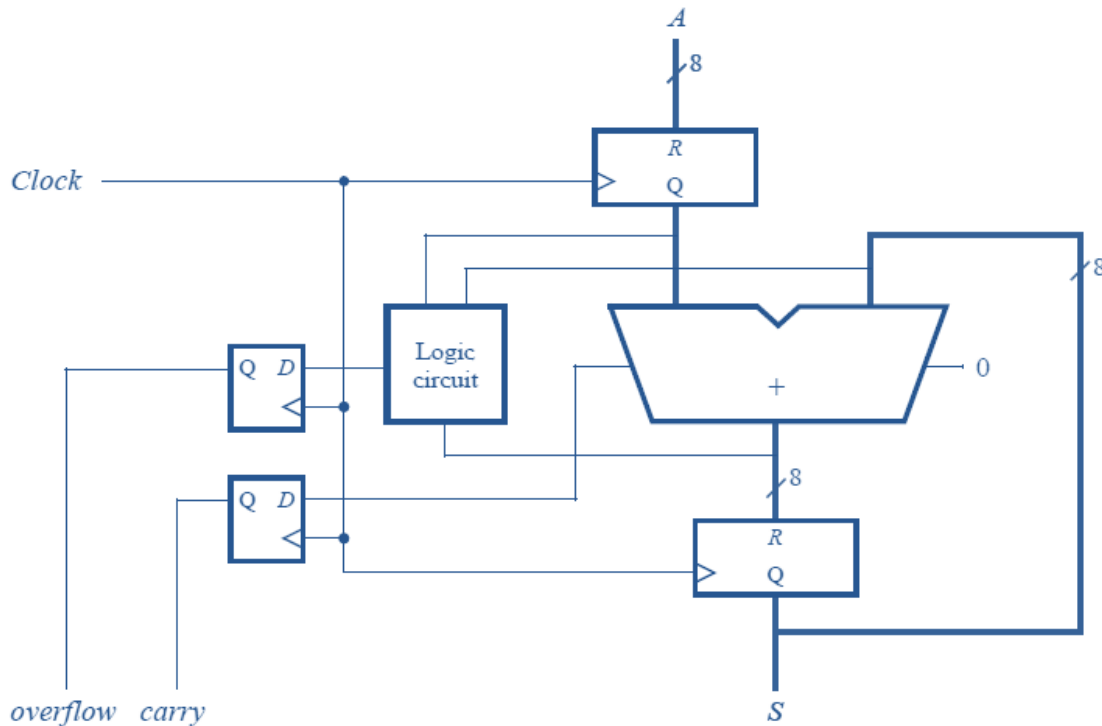
```

```

BEGIN
IF (CLK'EVENTAND CLK = '1') THEN
    Q <= D;
ENDIF;
ENDPROCESS;
END BEHAVIOR;

```

**Figura No. 3.6.3: Código correspondiente a la asignación 1.**



**Figura No. 3.6.4: Circuito acumulador de 8 bits.<sup>[12]</sup>**

### 3.7.3.3 Procedimiento

Para implementar el circuito realice los siguientes pasos:

1. Genere un nuevo proyecto Quartus II.
2. Observe el código de la figura 3.6.3 y verifique que cumple con las siguientes indicaciones: Conecte la entrada de A en los interruptores SW(0-7), utilice KEY0 como un reset asíncrono activo-bajo, y utilice KEY1 como una entrada manual reloj. La respuesta del sumador se debe mostrar en LEDR(0-7), la señal Carry se debe mostrar en LEDR(8), la señal *overflow* se debe mostrar en LEDR(9). Mostrar los valores registrados de A y S como números hexadecimales en la de 7 segmentos muestra HEX(3-2) y HEX(1-0), respectivamente.
3. Traslade el código de la figura a un archivo VHDL y asigne los pines en la FPGA de forma adecuada importando el archivo DE1-SoC.qsf.

4. Compilar el código y utilizar la simulación de tiempo (timing simulation) para verificar el funcionamiento correcto del circuito. Una vez que la simulación funcione correctamente, descargue el circuito en el kit DE1-SoC y comprobar el funcionamiento utilizando diferentes valores de A. Asegúrese de verificar que la salida overflow funciona correctamente.

### 3.7.4 Asignación 2

#### 3.7.4.1 Objetivo

En VHDL es posible declarar las restas mediante el operador (-) siempre y cuando se agregue la los componentes `ieee.std_logic_unsigned.all`, para esta práctica se volverán necesarios ya que se requiere mejorar el circuito anterior.

#### 3.7.4.2 Desarrollo

Mejorar el circuito de la asignación I para ser capaz de sumar y restar números. Para ello, se debe añadir una entrada `add_sub` a su circuito. Cuando `add_sub` es 1, el circuito debe restar A de S, y cuando `add_sub` es 0 el circuito debe sumar A en S como en la Asignación I.

#### 3.7.4.3 Procedimiento

Se debe considerar el procedimiento anterior agregando la variante de la resta

### 3.7.5 Asignación 3

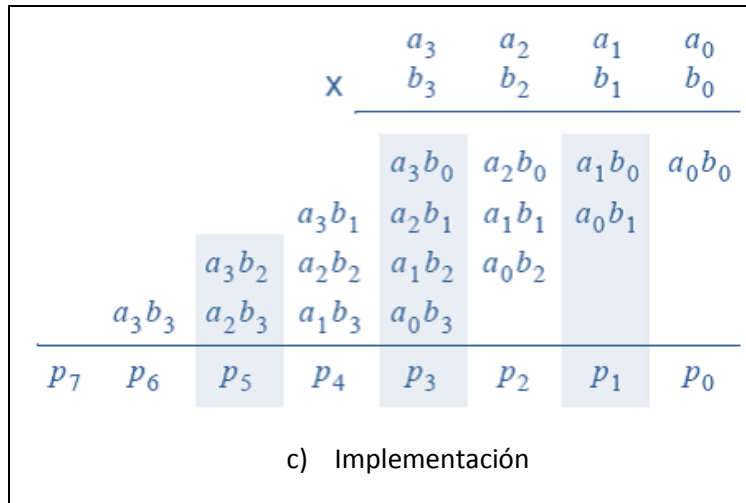
#### 3.7.5.1 Objetivo

La multiplicación también se conoce como sumas sucesivas, teniendo en cuenta este detalle es posible lograr un multiplicador basándose en sumadores conectados en cascada.

#### 3.7.5.2 Desarrollo

La Figura 3.6.5a muestra un ejemplo de la multiplicación en papel y lápiz de  $P = A*B$ , donde  $A = 11$  y  $B = 12$ .

$\begin{array}{r} 11 \\ \times 12 \\ \hline 22 \\ 11 \phantom{0} \\ \hline 132 \end{array}$ <p>a) Decimal</p>	$\begin{array}{r} 1011 \\ \times 1100 \\ \hline 0000 \\ 0000 \\ 1011 \\ 1011 \\ \hline 10000100 \end{array}$ <p>b) Binario</p>
---	--



**Figura No. 3.6.5: Multiplicación de dos números A y B.<sup>[12]</sup>**

Se Calcula  $P = A*B$  como una adición de sumandos. El primer sumando es igual a A veces el dígito B.

El segundo sumando es A veces el dígito de las decenas de B, se desplazó una posición a la izquierda. Se suman los dos términos para formar el producto  $P = 132$ .

La Figura 3.6.5b muestra el mismo ejemplo usando números binarios de cuatro bits. Para calcular  $P = A*B$ , primero se multiplica el primer termino de A por cada digito de B. Puesto que cada dígito de B es 1 o 0, los sumandos pueden ser versiones de A ó 0000. La figura 3.6.5c muestra cómo cada sumando se puede formar mediante el uso del operador AND booleano, para operar A con los bit apropiados de B.

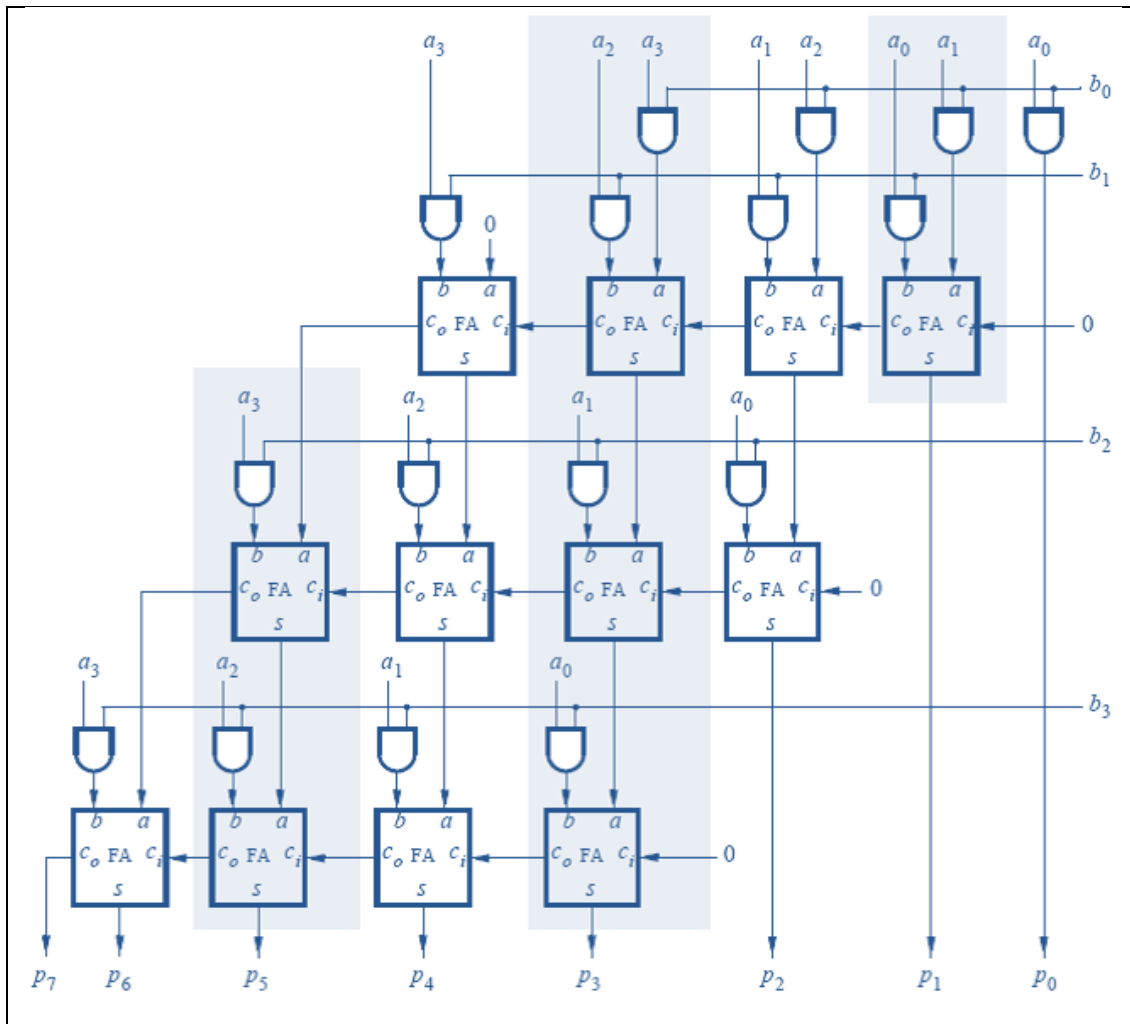
Un circuito de cuatro bits que implementa  $P = A*B$  se ilustra en la Figura 3.6.6. Debido a su estructura regular, este tipo de circuito multiplicador se le conoce como *multiplicador matricial*. Las áreas sombreadas corresponden a las columnas sombreadas en la Figura 3.6.5c. En cada fila del multiplicador se utilizan compuertas AND para producir los sumandos, y se utilizan módulos Full Adder para generar las sumas requeridas.

### 3.7.5.3 Desarrollo

Realice los siguientes pasos para implementar el circuito multiplicador:

1. Crear un nuevo proyecto Quartus II para implementar el circuito deseado en el kit Altera DE1-SoC.
2. Generar el archivo VHDL necesario. Use los interruptores SW(7-4) para representar el número A y SW(3-0) para representar B. Los valores hexadecimales de A y B se van a mostrar en los display de 7 segmentos HEX2 y HEX0, respectivamente. El resultado  $P = A*B$  se mostrará en HEX5 y HEX4.
3. Asignar los pines en la FPGA de forma adecuada.

4. Utilice la simulación para verificar el diseño.
5. Descargue el circuito en el kit DE1-SoC y probar su funcionalidad.



**Figura No. 3.6.6: Circuito multiplicador de 4 bits.<sup>[12]</sup>**

### 3.7.6 Asignación 4

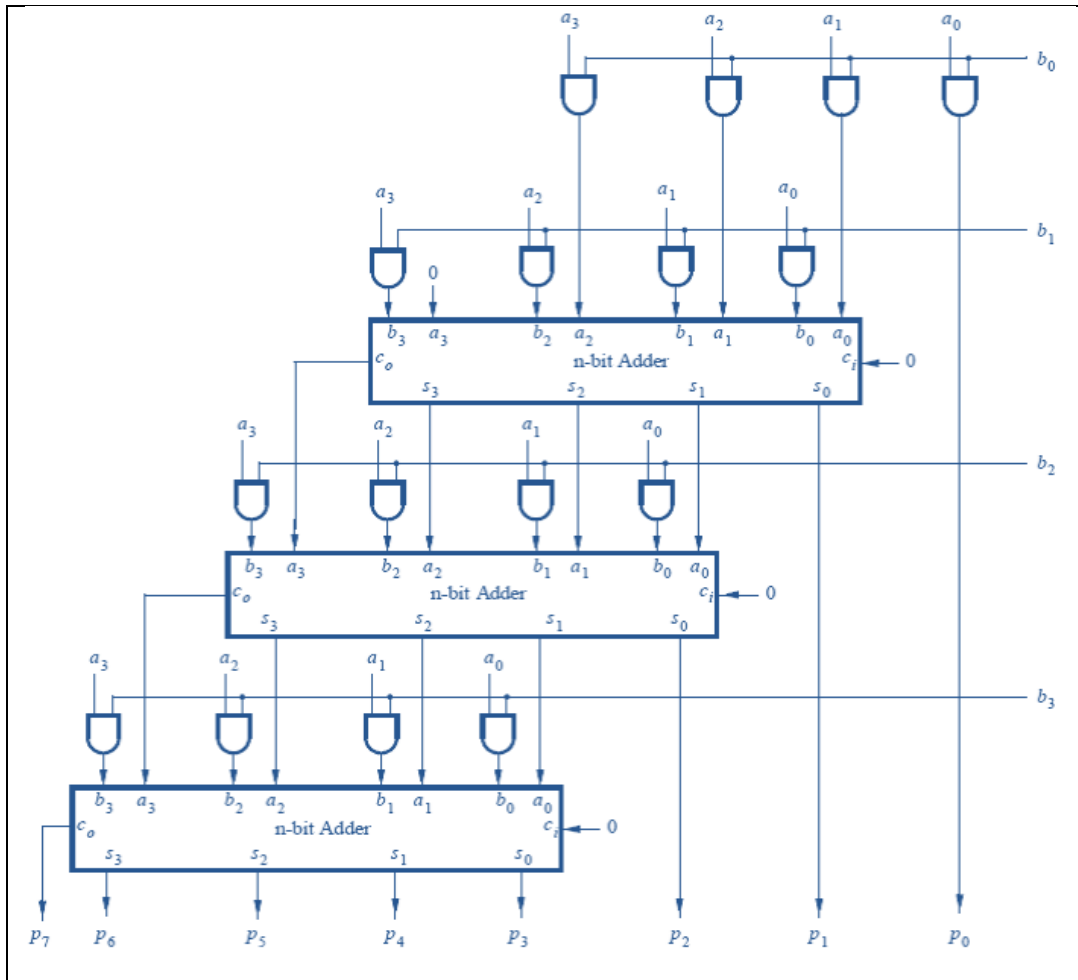
#### 3.7.6.1 Objetivo

Al igual de lo que sucedía con los contadores, muchas veces es necesario tener un circuito multiplicador de modulo k, para lograrlo es necesario que su unidad fundamental, también sea de modulo k, por lo cual para esta práctica se implementaran dichos circuitos manteniendo la lógica que se ha venido trabajando.

#### 3.7.6.2 Desarrollo

En la asignación 3 un multiplicador matricial se implementa mediante módulos Full Adder. En un circuito de alto nivel, una fila de Full Adder pueden ser reemplazados por un sumador de n-bits y el circuito multiplicador matricial se puede representar como se muestra en la Figura 3.6.7.

Cada sumador de n-bits opera una versión desplazada de A para una fila dada con el producto parcial de la fila de arriba. Haciendo una abstracción del circuito multiplicador como una secuencia de adiciones permite construir multiplicadores más grandes. El multiplicador se debe construir con sumadores de n-bits dispuestas en una estructura tal como se muestra en la Figura 3.6.7. Utilice este enfoque para implementar un circuito multiplicador de 8 x 8 con entradas y salidas registradas, como se muestra en la Figura 3.6.8.



**Figura No. 3.6.7: circuito multiplicador matricial implementado con sumadores de n-bits.**

[12]

### 3.7.6.3 Procedimiento

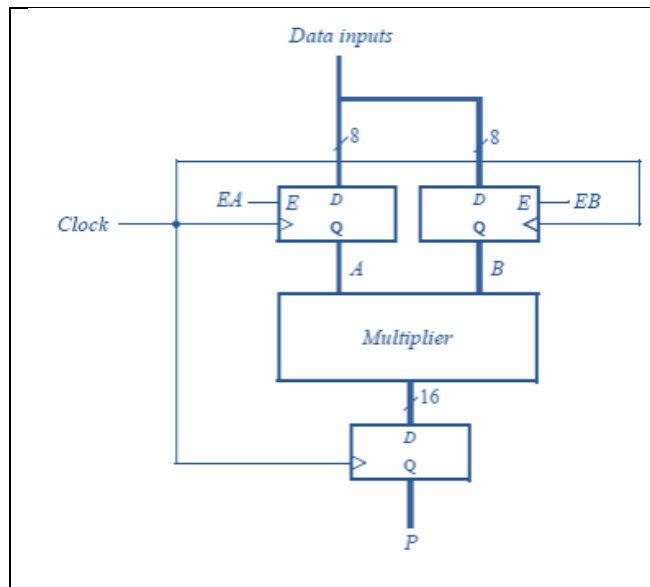
Para este circuito lleve a cabo los siguientes pasos:

1. Genere un nuevo proyecto Quartus II y escriba el archivo VHDL necesario.
2. Use los interruptores SW(7-0) para establecer los datos de entrada al circuito. Utilice SW(9) para la señal de habilitación EA en el registro A, y utilice SW(8) como habilitación para el registro B. Cuando SW9 = 1 muestre el contenido del registro A en LEDR y muestre el contenido del registro B sobre siempre en LEDR cuando SW8 = 1, cabe aclarar que cada registro debe hacerse de forma individual y secuencial. Utilice KEY0 como una entrada de



reset síncrono, y KEY1 como una señal de reloj manual. Muestre el producto  $P = A*B$  como un número hexadecimal en los display de 7 segmentos HEX3, HEX2, HEX1 y HEX0.

3. Incluya las asignaciones de pines necesarias para la FPGA y compile el proyecto.
4. Pruebe la funcionalidad de su diseño mediante la introducción de diversos valores de datos y realice la observación de los productos generados.



**Figura No. 3.6.8: circuito multiplicador con registros.<sup>[12]</sup>**

### 3.7.7 Asignación 5

#### 3.7.7.1 Objetivo

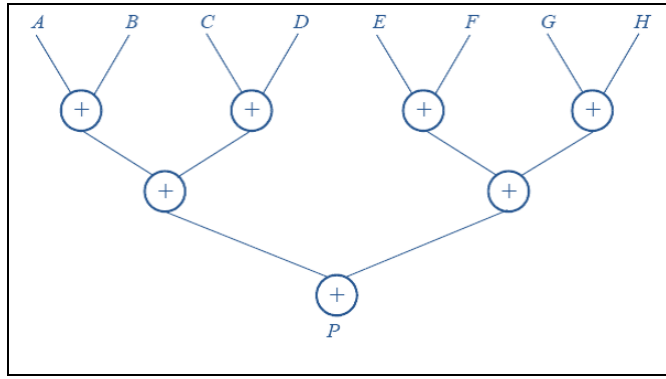
Un árbol de sumas es un modelo alternativo para algunas multiplicaciones, en esta práctica se tiene por meta implementar una versión base de este circuito.

#### 3.7.7.2 Desarrollo

En la asignación 4 se mostró cómo implementar la multiplicación  $A*B$  mediante una secuencia de adiciones. Otra forma de implementar este circuito es mediante un árbol sumador.

Un árbol sumador es un método en el cual se hace la suma de varios números en una forma paralela. Esta idea se ilustra en la figura 3.6.9., los números A, B, C, D, E, F, G, y H se suman juntos en paralelo. La adición de  $A + B$  ocurre simultáneamente con  $C + D$ ,  $E + F$  y  $G + H$ . El resultado de estas operaciones se vuelve a sumar en paralelo otra vez, hasta que se calcula la suma final P.

En esta parte se debe implementar un circuito multiplicador de  $8 \times 8$  utilizando el enfoque árbol sumador. Las entradas A y B, así como la salida P deben ser guardadas en registros como en la asignación 4.



**Figura No. 3.6.9: Árbol de sumas.<sup>[12]</sup>**

## 3.8 Laboratorio No. 7: Máquinas de estado finito

### 3.8.1 Introducción

Las máquinas de estado finitos son circuitos que tienen la capacidad de realizar cálculos automáticos basándose en eventos o variables externas, de forma que son circuitos muy útiles en la automatización de procesos.

Realizar circuitos por medio de máquinas de estado requiere tener bien claro el proceso a seguir, las variables de entrada y salida, el modelo a elegir y tener un método de codificación adecuado.

En esta práctica se abordarán los conceptos generales sobre máquinas de estado, en adelante FSM (Finite State Machine), el archivo original de esta practica puede obtenerse desde el siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab7\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab7_VHDL.pdf) [en línea] [ultima consulta 24/02/2016]

### 3.8.2 Objetivos Generales

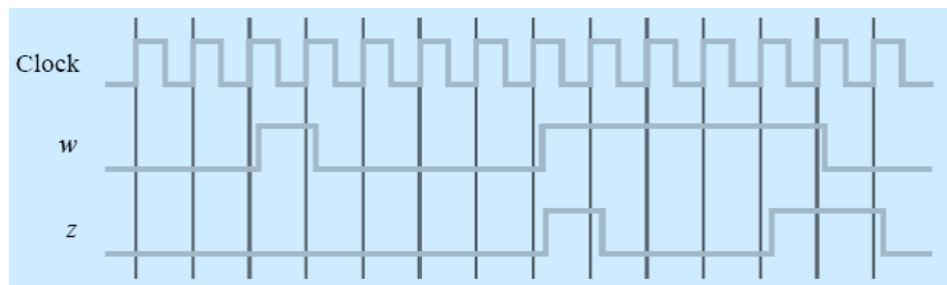
- Entender el funcionamiento de las máquinas de estado en lenguaje VHDL.
- Reconocer los tipos de codificación para las FSM.
- Aprender los modelos de compilación que permite Quartus II
- Indagar sobre la herramienta de diagramas de estado para Quartus II.

### 3.8.3 Asignación 1

#### 3.8.3.1 Objetivo

En esta práctica se trabajará la codificación One Hot para las FSM, se desarrollarán el código para la tabla 3.7.1, de forma que la variante mostrada en la tabla 3.7.2 sea de mayor asimilación para el estudiante en cuanto a la creación del código VHDL correspondiente.

#### 3.8.3.2 Desarrollo

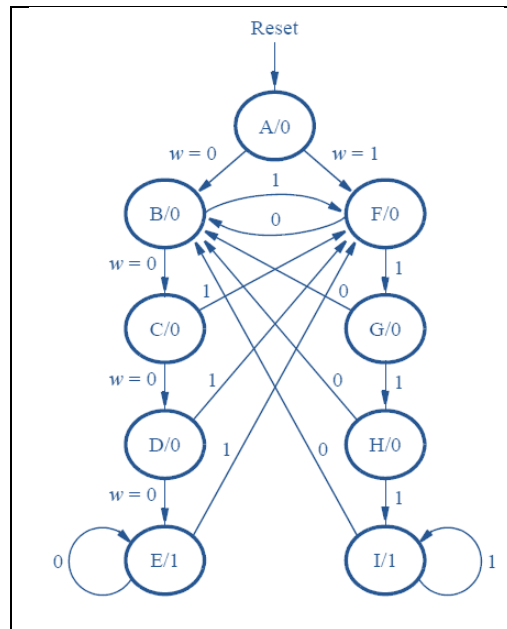


**Figura No. 3.7.1: Diagrama de Tiempo.<sup>[13]</sup>**

Se desea poner en práctica una FSM que reconoce dos secuencias específicas, introducidas por la variable W, cuando el circuito reconoce la secuencia 0000 ó 1111 una salida Z se activa ( $Z=1$ ). Esto sucederá siempre que  $W = 1$  o  $W = 0$  para cuatro pulsos consecutivos de una señal de reloj respectivamente; de lo contrario,  $z = 0$ . Las secuencias traslapadas están permitidas, por lo que si

w = 1 para cinco pulsos de reloj consecutivos la salida Z se activará en el cuarto y quinto pulso. La Figura 3.7.1 ilustra la relación requerida entre W y Z.

Un diagrama de estado para este FSM se muestra en la Figura 3.7.2. Para esta asignación el estudiante debe obtener manualmente un circuito FSM que sea capaz de implementar este diagrama de estados, incluyendo las expresiones lógicas que alimentan cada uno de los FF de estado. Para implementar el FSM se deben utilizar nueve FF de estado llamados  $y_8; \dots, y_0$  y se debe hacer la asignación de un estado One-hot como se indica en la Tabla 3.7.1.



**Figura No. 3.7.2: Diagrama de estados.** [13]

Name	State Codes								
	$y_8$	$y_7$	$y_6$	$y_5$	$y_4$	$y_3$	$y_2$	$y_1$	$y_0$
A	0	0	0	0	0	0	0	0	1
B	0	0	0	0	0	0	0	1	0
C	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	1	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0
H	0	1	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0

**Tabla No. 3.7.1: Codigos One-Hot para la FSM.**

```

LIBRARYIEEE ;
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.STD_LOGIC_ARITH.ALL;

```

```

ENTITY L7_PART1_AUP IS
--SW(0) IS ACTIVE LOW SYNCHRONOUS RESET.--SW(1) IS W.
--KEY(0) IS SIGNAL CLOCK.--LEDR IS SIGNAL OUTPUT.
PORT (SW          :INSTD_LOGIC_VECTOR (1DOWNTO0) ;
      KEY          :INSTD_LOGIC_VECTOR (0DOWNTO0) ;
      LEDR        :OUTSTD_LOGIC_VECTOR (9DOWNTO0)) ;
END L7_PART1_AUP;

ARCHITECTURE BEHAVIOR OF L7_PART1_AUP IS
COMPONENT FLIPFLOP
PORT (D, CLOCK, RESETN, SETN      :INSTD_LOGIC;
      Q      :OUTSTD_LOGIC);
ENDCOMPONENT;
SIGNAL CLOCK, RESETN, W, Z :STD_LOGIC;
SIGNAL Y_D, Y_Q :STD_LOGIC_VECTOR (8DOWNTO0) ;
BEGIN
CLOCK <= KEY (0) ;
RESETN <= SW (0) ;
W <= SW (1) ;

Y_D (0) <= '0' ;
U0: FLIPFLOP PORTMAP (Y_D (0), CLOCK, RESETN, '1', Y_Q (0)) ;
Y_D (1) <= (Y_Q (0) OR Y_Q (5) OR Y_Q (6) OR Y_Q (7) OR Y_Q (8)) AND (NOT (W)) ;
U1: FLIPFLOP PORTMAP (Y_D (1), CLOCK, RESETN, '1', Y_Q (1)) ;
Y_D (2) <= Y_Q (1) AND (NOT (W)) ;
U2: FLIPFLOP PORTMAP (Y_D (2), CLOCK, RESETN, '1', Y_Q (2)) ;
Y_D (3) <= Y_Q (2) AND (NOT (W)) ;
U3: FLIPFLOP PORTMAP (Y_D (3), CLOCK, RESETN, '1', Y_Q (3)) ;
Y_D (4) <= (Y_Q (3) OR Y_Q (4)) AND (NOT (W)) ;
U4: FLIPFLOP PORTMAP (Y_D (4), CLOCK, RESETN, '1', Y_Q (4)) ;
Y_D (5) <= (Y_Q (0) OR Y_Q (1) OR Y_Q (2) OR Y_Q (3) OR Y_Q (4)) AND (W) ;
U5: FLIPFLOP PORTMAP (Y_D (5), CLOCK, RESETN, '1', Y_Q (5)) ;
Y_D (6) <= Y_Q (5) AND W ;
U6: FLIPFLOP PORTMAP (Y_D (6), CLOCK, RESETN, '1', Y_Q (6)) ;
Y_D (7) <= Y_Q (6) AND W ;
U7: FLIPFLOP PORTMAP (Y_D (7), CLOCK, RESETN, '1', Y_Q (7)) ;
Y_D (8) <= (Y_Q (7) OR Y_Q (8)) AND W ;
U8: FLIPFLOP PORTMAP (Y_D (8), CLOCK, RESETN, '1', Y_Q (8)) ;

Z <= Y_Q (4) OR Y_Q (8) ;
LEDR (8DOWNTO0) <= Y_Q ;
LEDR (9) <= Z ;
END BEHAVIOR;

LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY FLIPFLOP IS
PORT (D, CLOCK, RESETN, SETN      :INSTD_LOGIC;
      Q      :OUTSTD_LOGIC);
END FLIPFLOP;
ARCHITECTURE BEHAVIOR OF FLIPFLOP IS
BEGIN
PROCESS (CLOCK)
BEGIN
IF (CLOCK'EVENT AND CLOCK = '1') THEN
IF RESETN = '0' THEN
      Q <= '0' ;

```

```

ELSIF   SETN = '0' THEN
        Q <= '1';
ELSE
        Q <= D;
ENDIF;
ENDIF;
ENDPROCESS;
END BEHAVIOR;

```

**Figura No 3.7.3: Código para Tabla 3.7.1.**

### 3.8.3.3 Procedimiento

Diseñe e implemente el circuito en el kit DE1-SoC de la forma siguiente:

1. Genere un nuevo proyecto Quartus II para el circuito FSM.
2. Traslade a un archivo VHDL el código mostrado en la figura 3.7.3 y compruebe que cumple con los siguientes requisitos: debe poseer un componente necesario para crear las nueve instancias de los FF en el circuito y cada uno debe especificar las expresiones lógicas que alimentan las entradas de los FF. Solo se utilizan instrucciones de asignación simples en el código VHDL para especificar las expresiones lógicas de los flip-flops. Se utiliza el interruptor SW(0) como un reset síncrono activo bajo para el FSM, además el SW1 funciona como la entrada W, y el pulsador KEY0 como la entrada de señal de reloj que se aplicará manualmente. El elemento LEDR(9) como la salida Z, y se asignó las salidas de los FF de estado a LEDR(0 a 8).
3. Incluya el archivo VHDL en el proyecto, y asigne los pines de la FPGA de forma adecuada.
4. Simule el comportamiento del circuito.
5. Una vez que se tenga la certeza de que el circuito funciona correctamente como resultado de la simulación, descargue el circuito en el chip FPGA. Pruebe la funcionalidad del diseño mediante la aplicación de las secuencias de entrada y observe los led's de salida. Asegúrese de que la transición en la FSM sea adecuada entre estados, y verifique que se produce el valor de salida correcto en LEDR(9).
6. La Tabla 3.7.2 muestra una asignación de estados one-hot modificados en el que el estado de reset A (o reposo), tiene todos los FF en estado 0. Esto se logra invirtiendo el valor de la variable y0. Diseñe una versión modificada para este código en VHDL de forma que satisfaga esta asignación de estados.
7. Compile de nuevo el circuito y sométalo a las pruebas respectivas.

Name	State Codes								
	y <sub>8</sub>	y <sub>7</sub>	y <sub>6</sub>	y <sub>5</sub>	y <sub>4</sub>	y <sub>3</sub>	y <sub>2</sub>	y <sub>1</sub>	y <sub>0</sub>
A	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	1	1
C	0	0	0	0	0	0	1	0	1
D	0	0	0	0	0	1	0	0	1
E	0	0	0	0	1	0	0	0	1
F	0	0	0	1	0	0	0	0	1

<b>G</b>	0	0	1	0	0	0	0	0	1
<b>H</b>	0	1	0	0	0	0	0	0	1
<b>I</b>	1	0	0	0	0	0	0	0	1

Tabla No. 3.7.2: Codigos One-Hot modificados para la FSM.

## 3.8.4 Asignación 2

### 3.8.4.1 Objetivo

El tipo de codificación más común es el binario para FSM, en esta práctica se abordará el proceso para crear archivos VHDL que satisfagan este tipo de codificación.

### 3.8.4.2 Desarrollo

Para esta parte se debe escribir un código VHDL con otro formato para la FSM de la figura 3.7.2. En esta versión de código no se deben generar manualmente las expresiones lógicas necesarias para cada estado de los FF. En su lugar, se describe la tabla de estados para la FSM mediante una instrucción CASE VHDL en un bloque PROCESS, se utiliza otro PROCESS para crear las instancias de los FF de estado. Puede utilizar un tercer bloque PROCESS o instrucciones de asignación simples para especificar la salida Z, esa sección queda a criterio del estudiante. Para implementar el FSM, utilice cuatro FF de estados llamados y3, y2, y1 y0, usando códigos binarios, esta lógica es diferente a los códigos one-hot, las combinaciones se muestran en la Tabla 3.7.3.

Name	State Codes			
	y <sub>3</sub>	y <sub>2</sub>	y <sub>1</sub>	y <sub>0</sub>
<b>A</b>	0	0	0	0
<b>B</b>	0	0	0	1
<b>C</b>	0	0	1	0
<b>D</b>	0	0	1	1
<b>E</b>	0	1	0	0
<b>F</b>	0	1	0	1
<b>G</b>	0	1	1	0
<b>H</b>	0	1	1	1
<b>I</b>	1	0	0	0

Tabla No. 3.7.3: Códigos binarios para la FSM.

Un esqueleto de código VHDL se da en la Figura 3.7.4.

```

ARCHITECTURE BEHAVIOR OF PART2 IS
-- DECLARE SIGNALS
TYPE STATE_TYPE IS (A, B, C, D, E, F, G, H, I);
-- ATTRIBUTE TO DECLARE A SPECIFIC ENCODING FOR THE STATES
ATTRIBUTE SYN_ENCODING :STRING;
ATTRIBUTE SYN_ENCODING OF STATE_TYPE :TYPEIS"0000 0001 0010 0011 0100
0101 0110 0111 1000";
SIGNAL Y_Q, Y_D : STATE_TYPE;-- Y_Q IS PRESENT STATE, Y_D ISNEXT STATE
BEGIN-- CODIGO INCOMPLETO
PROCESS (W, Y_Q)--STATE TABLE

```

```

BEGIN
CASE Y_Q IS
WHEN A IF (W = '0') THEN Y_D <= B;
ELSE Y_D <= F;
ENDIF;
-- OTROS ESTADOS
ENDCASE;
ENDPROCESS;-- STATE TABLE
PROCESS (CLOCK)-- STATE FLIP-FLOPS
BEGIN-- CÓDIGO NO MOSTRADO
ENDPROCESS;
-- ASSIGNMENTS FOR OUTPUT Z AND THE LEDS
END BEHAVIOR;

```

**Figura No 3.7.4: Esqueleto de código para una FSM por códigos binarios.<sup>[13]</sup>**

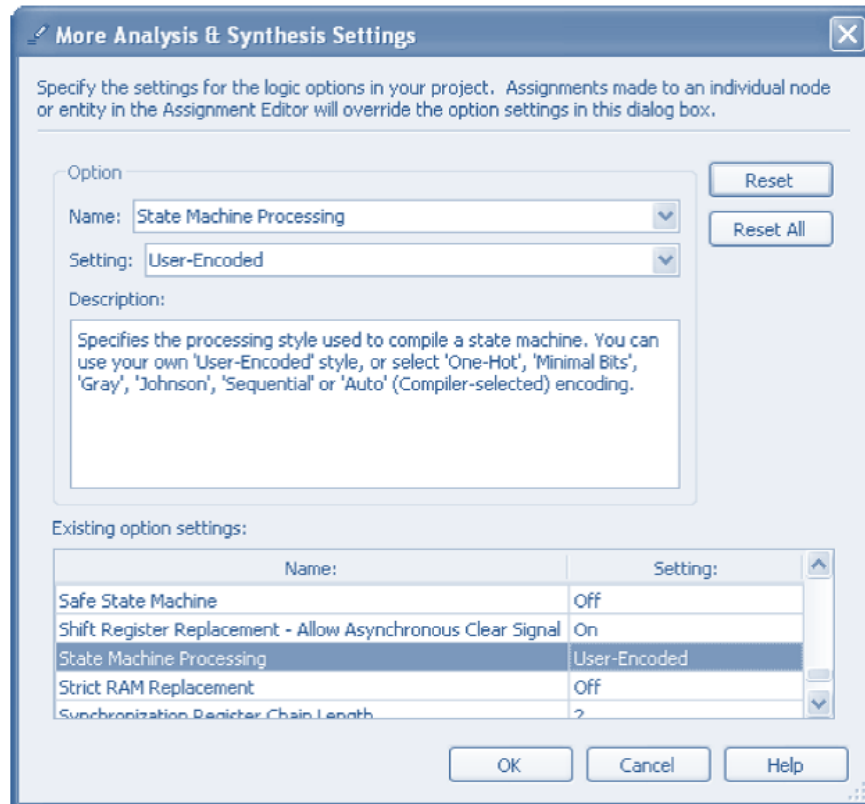
### 3.8.4.3 Procedimiento

Para este circuito es necesario realizar las siguientes tareas:

1. Genere un nuevo proyecto para el FSM.
2. Incluya en el proyecto el archivo VHDL que utiliza el formato de código en la Figura 3.7.4. Utilice los mismos interruptores, pulsadores y led's que se utilizaron en la asignación I.
3. Antes de compilar su código es necesario decirle explícitamente la herramienta de síntesis con la cual trabajará Quartus II la máquina de estados finitos implementada, junto con la asignación de estados especificados en el código VHDL. Si no se especifica explícitamente este ajuste para Quartus II, la herramienta de síntesis utilizará automáticamente una asignación de estados de su propia elección, y no hará caso de los códigos de estado binario especificados en el código VHDL. Para realizar este ajuste, seleccione *Assignments > Settings* en Quartus II, y haga clic en el elemento *Analysis and Synthesis* en el lado izquierdo de la ventana, luego clic en el botón *More Setting*. Como se indica en la Figura 3.7.5, cambiar el parámetro *State Machine Processing* a *User-Encoded*.
4. Compile el proyecto. Para examinar el circuito producido por Quartus II necesitará la herramienta *RTL Viewer*. Haga doble clic en el cuadro en el que se muestra el circuito que representa la máquina de estado finito, y determine si el diagrama de estado que se muestra corresponde de forma correcta a la Figura 3.7.2. Para ver los códigos de estado utilizados para la FSM, abra el informe de compilación, seleccione la sección del informe *Analysis and Synthesis*, y haga clic en *state machines*.
5. Descargue el circuito en el chip FPGA y pruebe la funcionalidad.
6. En el paso 3 se indicó en la herramienta de síntesis del Quartus II que se utilizará la asignación de estados dado en el código VHDL. Para ver el resultado de la eliminación de este ajuste, abra nuevamente la ventana de configuración Quartus II, seleccione *Assignments > Settings* en Quartus II, y haga clic en el elemento *Analysis and Synthesis* en el lado izquierdo de la ventana, luego clic en el botón *More Setting*, luego cambie el parámetro *State Machine Processing* a *One-hot*. Vuelva a compilar el circuito y luego abra el archivo de informe, seleccione la sección del informe *Analysis and Synthesis*, y haga clic



en *state machines*. Compare los códigos de estado que se muestran con los indicados en la Tabla 3.7.2 y discuta cualquier diferencia que observe.



**Figura No. 3.7.5: Asignación de método de síntesis en Quartus II. [13]**

### 3.8.5 Asignación 3

#### 3.8.5.1 Objetivo

El uso de shift register es muy común para el desarrollo de FSM, en esta práctica se tiene como meta crear una FSM basadas en shift register.

#### 3.8.5.2 Desarrollo

Un detector de secuencia puede ser implementado de una manera directa usando registros de desplazamiento, en lugar de utilizar el enfoque más formal descrito anteriormente. Diseñe código VHDL que invoque dos Shift-Register de 4 bits; uno es para el reconocimiento de una secuencia de cuatro 0s, y el otro para cuatro 1s. Incluya las expresiones lógicas apropiadas en el diseño para producir la salida Z.

#### 3.8.5.3 Procedimiento

Genere un proyecto Quartus II para el diseño e implementación del circuito en el kit DE1-SoC. Utilice los interruptores y los leds del kit de una forma similar a la utilizada en las partes I y II, observe el comportamiento de los Shift-Register y la salida Z. Responda a la siguiente pregunta: ¿Se podría utilizar un solo shift-register de 4 bits, en lugar de dos? Explíquela respuesta.

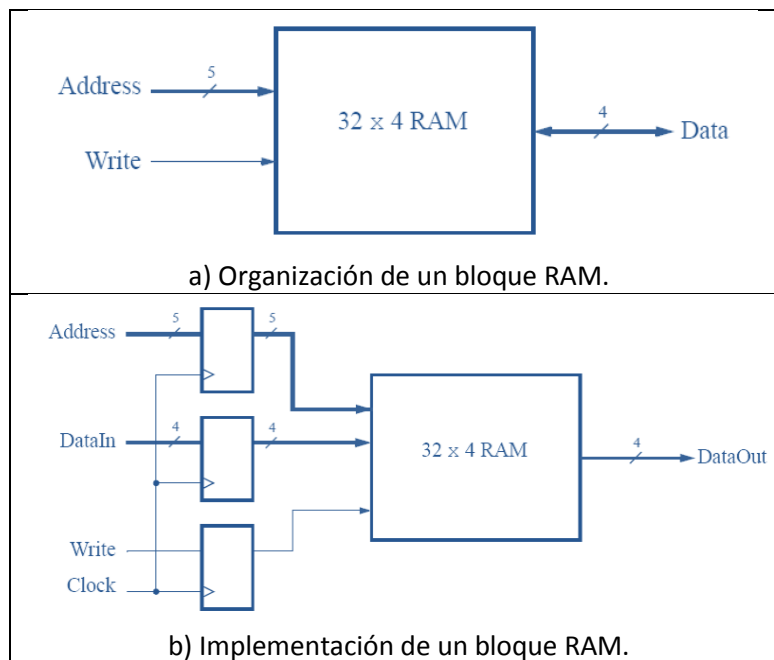
## 3.9 Laboratorio No. 8: Bloques de Memoria

### 3.9.1 Introducción

En los sistemas informáticos es necesario para proporcionar una cantidad sustancial de memoria. Si un sistema se implementa utilizando la tecnología FPGA es posible proporcionar una cierta cantidad de datos mediante el uso de los recursos de memoria que existen en el dispositivo FPGA. En este laboratorio se examinarán los aspectos generales que intervienen en la ejecución de dicha memoria.

Un diagrama del módulo de memoria de acceso aleatorio (RAM) que se implementará se muestra en la Figura 3.8.1a. Este bloque contiene 32 palabras de cuatro bits (filas), las cuales se acceden a través de un puerto de cinco bits de direcciones, un puerto de datos de cuatro bits y una entrada de control de escritura.

El Cyclone V FPGA que se incluye en la tarjeta DE1-SoC ofrece recursos de memoria dedicados llamados bloques M10K. Cada bloque contiene 10240 bits de memoria, que pueden ser separados en diferentes tamaños. Un término común usado para especificar el tamaño de una memoria es su relación de aspecto, la longitud las palabras y la anchura en bits. Algunas relaciones soportadas por los bloques M10K son 8K x 1, 4K x 2, 2K x 4, entre otros.



**Figura No. 3.8.1: Diagramas para bloques RAM. [14]**

En este laboratorio se utilizará el modo 2K x 4, utilizando sólo las primeras 32 palabras en la memoria.

También hay que mencionar que muchos otros modos de operación son compatibles en un bloque M10K, pero no se discutirán aquí.

Hay dos características importantes del bloque M10K que tienen que ser mencionadas:

- En primer lugar, incluye registros que se pueden utilizar para sincronizar todas las señales de entrada y de salida a una entrada de reloj. Los registros en los puertos de entrada siempre se deben utilizar y los registros en los puertos de salida son opcionales.
- En segundo lugar, el bloque M10K tiene puertos separados para los datos que se escriben en la memoria y los datos que se leen de la memoria. Teniendo en cuenta estos requisitos, en todas las asignaciones de este laboratorio se implementará el módulo de 32 x 4 RAM modificado como se muestra en la Figura 3.8.1b. Incluyendo registros para la dirección, la entrada de datos, y la escritura de puertos, se utilizará un puerto de salida de datos sin registrar por separado.

El archivo original de la práctica puede obtenerse a través del siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab8\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab8_VHDL.pdf) [en línea] [última consulta 24/02/2016]

### 3.9.2 Objetivos Generales

- Conocer el proceso para implementar un bloque RAM a partir de los asistentes de diseño incluidos en Quartus II.
- Desarrollar un bloque RAM únicamente con sentencias VHDL.
- Aprender el uso de archivos de inicialización de memoria para bloques RAM.

### 3.9.3 Asignación 1

#### 3.9.3.1 Objetivo

Las estructuras lógicas comunes, tales como sumadores, registros, contadores y memorias, se pueden implementar en un chip FPGA utilizando módulos predefinidos que se proporcionan en las bibliotecas. En esta práctica se utilizará un módulo llamado `altsyncram` para implementar la memoria que se muestra en la Figura 3.8.1b.

#### 3.9.3.2 Desarrollo

1. Crear un nuevo proyecto Quartus II para implementar el módulo de memoria.
2. La versión 15.0 tiene una variante en relación a las LPM, la herramienta **MegaWizard Plugin Manager** fue sustituido por **IP Catalog**, este siempre se puede encontrar en la pestaña **Tools**, del menú principal, al dar clic sobre esta opción no desplegará ninguna ventana pero añadirá un nuevo menú en la parte derecha de la ventana principal tal como se muestra en la figura 3.8.3a, la función que se utilizará es RAM: Port 1, dando doble clic sobre esta opción emergerá una nueva ventana en la cual pedirá el nombre del archivo, la dirección donde se guardará y el lenguaje de programación, la siguiente ventana pedirá los datos del bloque de memoria, aquí se debe seleccionar un bloque de tipo M10K, que contenga 32 palabras de 4 bits cada una, el sistema debe ser síncrono por lo tanto solo se

utilizará una señal de reloj; en la ventana 2 del asistente se debe deseleccionar la opción **'q' output port** que se encuentra en la sección **which ports should be registered?**, tal como se muestra en la figura 3.8.3b, lo ajustes restante deberán permanecer con las opciones por defecto hasta llegar a la opción que permita finalizar el asistente.

- Una vez finalizado el software preguntara si los archivos creados se añaden por defecto al proyecto, esto se muestra en la figura 3.8.4.
- Examine el archivo VHDL ram32x4.vhd que define el siguiente subcircuito:

```

ENTITY RAM32X4 IS
PORT
(
    ADDRESS      :IN STD_LOGIC_VECTOR(4DOWNTO0) ;
    CLOCK        :IN STD_LOGIC:= '1';
    DATA        :IN STD_LOGIC_VECTOR(3DOWNTO0) ;
    WREN         :IN STD_LOGIC;
    Q            :OUT STD_LOGIC_VECTOR(3DOWNTO0)
);
END RAM32X4;

```

- Lo siguiente será crear un archivo VHDL de alto nivel para inicializar este código en forma de módulo, el código de este archivo se muestra en la figura 3.8.2.
- Compile el circuito. Observe en el reporte de compilación que el software utiliza 128 bits en uno de los bloques de memoria M10K para implementar el circuito de memoria RAM.
- Simule el comportamiento del circuito y asegúrese de que puede leer y escribir datos en la memoria.

```

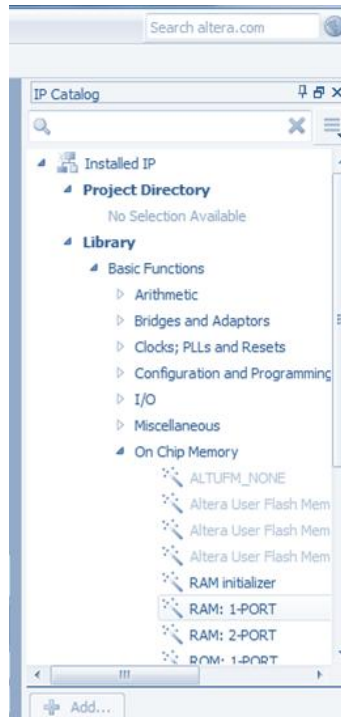
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY L8_PART1_AUP IS
PORT (DIRECCION      :IN STD_LOGIC_VECTOR(4DOWNTO0) ;
      RELOJ          :IN STD_LOGIC;
      DATOS          :IN STD_LOGIC_VECTOR(3DOWNTO0) ;
      ESCRITURA     :IN STD_LOGIC;
      SALIDA         :OUT STD_LOGIC_VECTOR(3DOWNTO0) );
END L8_PART1_AUP;

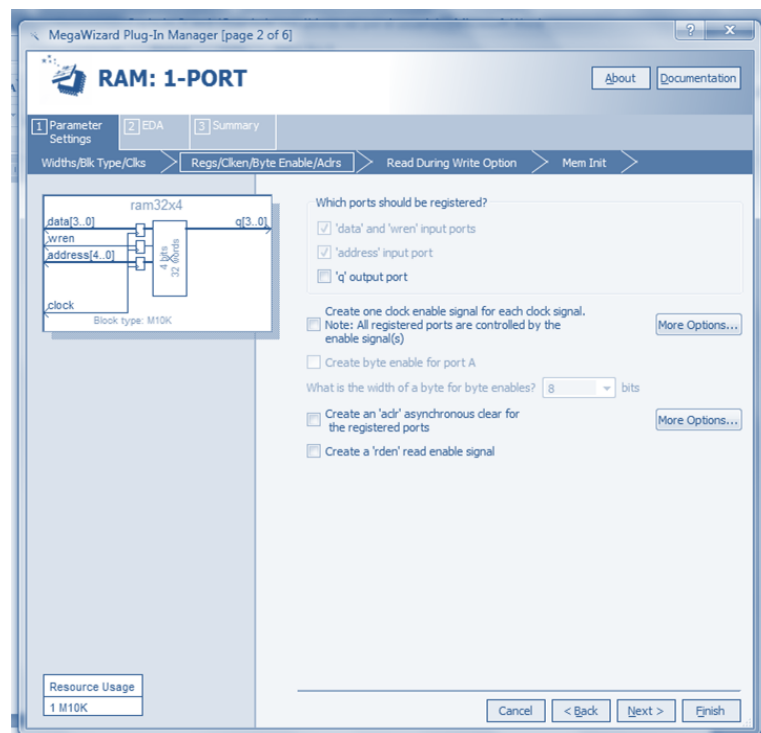
ARCHITECTURE BEHAVIOR OF L8_PART1_AUP IS
COMPONENT RAM32X4
PORT
(
    ADDRESS      :IN STD_LOGIC_VECTOR(4DOWNTO0) ;
    CLOCK        :IN STD_LOGIC:= '1';
    DATA        :IN STD_LOGIC_VECTOR(3DOWNTO0) ;
    WREN         :IN STD_LOGIC;
    Q            :OUT STD_LOGIC_VECTOR(3DOWNTO0)
);
ENDCOMPONENT;
BEGIN
M0: RAM32X4 PORTMAP (DIRECCION, RELOJ, DATOS, ESCRITURA, SALIDA);
END BEHAVIOR;

```

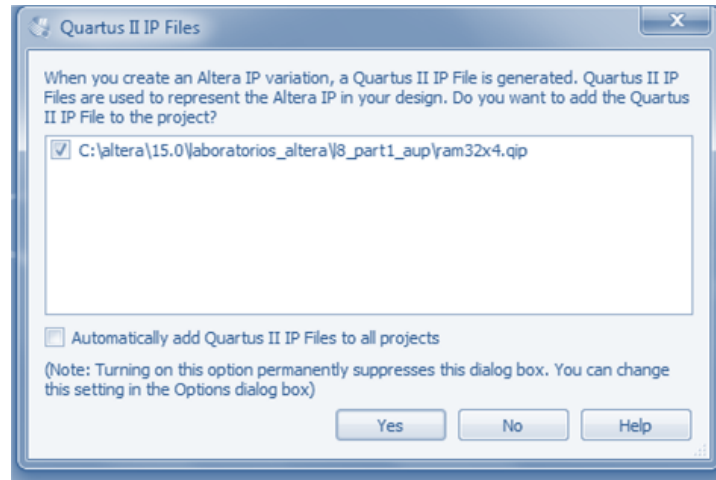
**Figura No. 3.8.2: Código para inicializar el modulo RAM.<sup>[14]</sup>**



**Figura No. 3.8.3: A) Menu Principal de IP Catalog.**



**Figura No. 3.8.3: B) Configuración de los puertos de entrada y salida.**



**Figura No. 3.8.4: Confirmación de adición de datos.**

## 3.9.4 Asignación 2

### 3.9.4.1 Objetivo

Entendido el proceso de creación de bloques RAM y su inicialización el siguiente paso es ver su funcionamiento en la tarjeta DE1-SoC, la meta de esta práctica es asignar los pines necesarios para probar un bloque RAM en la kit de desarrollo.

### 3.9.4.2 Desarrollo

Se requiere implementar el circuito de memoria en la FPGA, se hará uso de interruptores para cargar algunos datos en la memoria creada. También se busca mostrar el contenido de la memoria RAM en los displays de 7 segmentos.

### 3.9.4.3 Procedimiento

Para realizar este circuito realice los siguientes pasos:

1. Cree un nuevo proyecto Quartus II que se utilizará para implementar el circuito en la tarjeta DE1-SoC.
2. Cree otro archivo VHDL que inicialice el módulo ram32x4, incluyendo en este los pines de entrada y salida necesarios. Utilice los interruptores SW(3 a 0) para ajustar los datos de entrada para la RAM, los interruptores SW(8 a 4) para especificar la dirección. Utilice el interruptor SW(9) como señal de escritura y el botón KEY(0) como entrada de reloj manual. Se debe mostrar el valor de la dirección los display HEX5 y HEX4, los datos que se introducen en la memoria se mostrarán en HEX2 y los datos leídos de la memoria se visualizarán en HEX0.
3. Ponga a prueba el circuito y asegúrese de que los datos se pueden almacenar en diferentes direcciones de la memoria.

## 3.9.5 Asignación 3

### 3.9.5.1 Objetivo

Los módulos IP son bastante útiles pero no son la única forma de crear bloques de memoria, de forma general un bloque de memoria es arreglo de registros, así pues, cada palabra de 4 bits representa una fila, en esta práctica se aprenderá el proceso para crear este tipo de arreglos mediante sentencias VHDL.

### 3.9.5.2 Desarrollo

En un diseño VHDL especificado es posible definir la memoria como una matriz multidimensional. Una matriz de 32 x 4, que tiene 32 palabras de 4 bits cada una, la lectura de estos registros únicamente será por filas, este tipo de arreglos pueden ser declarados con la sentencia:

```
TYPE MEM IS ARRAY (0 TO 31) OF STD_LOGIC_VECTOR (3 DOWNTO 0) ;  
SIGNAL MEMORY_ARRAY : MEM;
```

En el Cyclone V FPGA este tipo de arreglos puede ser implementado mediante el uso de los flip-flops que cada elemento lógico contiene o de manera más eficiente, por medio del uso de los bloques de memoria ya incorporados. La Ayuda Quartus II ofrece otros ejemplos de código VHDL que muestran cómo se puede especificar un bloque de memoria ("Inferred memory").

### 3.9.5.3 Procedimiento

Para implementar esta práctica realice los siguientes pasos:

1. Cree un nuevo proyecto que se utilizará para implementar el circuito en la tarjeta DE1-SoC.
2. Escribir un archivo VHDL que proporcione la funcionalidad necesaria, incluyendo la capacidad de cargar la memoria RAM y leer su contenido, como se hizo en la asignación 2.
3. Asigne los pines tal como se hizo en la asignación anterior.
4. Compile el circuito y descárguelo a la tarjeta DE1-SoC.
5. Compruebe el funcionamiento del circuito.

## 3.9.6 Asignación 4

### 3.9.6.1 Objetivo

El bloque SRAM en la figura 3.8.1 tiene un solo puerto que ofrece la dirección de las operaciones de lectura y escritura. En esta asignación se va a crear un tipo diferente de módulo de memoria, en la que se tendrá un puerto para suministrar las direcciones de lectura, y un puerto separado que dará la dirección de escritura.

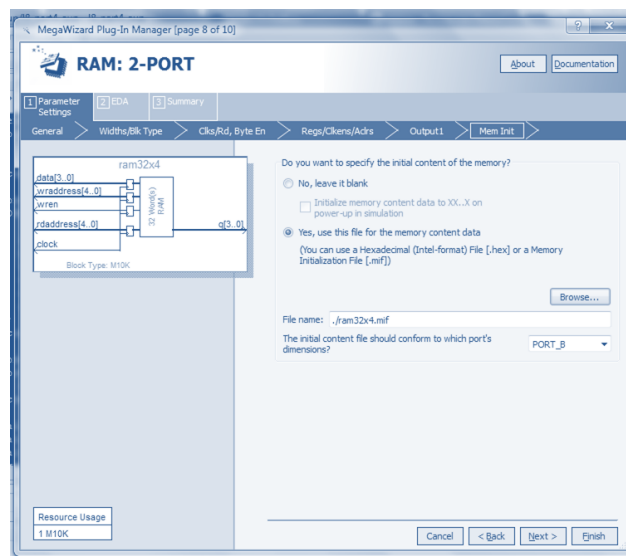
### 3.9.6.2 Desarrollo

Se deben considerar estos pasos para completar la práctica:

1. En la asignación anterior se explicó cómo acceder a los módulos IP que provee Quartus II, para esta práctica será necesario utilizar el módulo **RAM: 2 Port** que se encuentra en la

categoría **On Chip Memory**, los pasos de la primera ventana es similar a lo antes descrito, el archivo se debe llamar **ram32x4**, debe ser para VHDL y se debe guardar en la carpeta general del proyecto, en la ventana siguiente se debe seleccionar la opción **With one read port and one write port** que se encuentra en la categoría **How will you be using the dual port ram?**. La opciones restantes deben mantenerse por defecto, siempre recordando que el tamaño de la memoria será de 32 palabras de 4 bits cada una, el circuito se mantiene síncrono de forma que solo será necesaria una señal para ambos puertos, la salidas no deben estar registradas así que esa opción se debe deseleccionar tal como se hacía en la asignación 2, en la pagina 6 dentro de la categoría **Mixed Port Read-During-Write for Single Input Clock RAM** se debe seleccionar la opción **I do not care (the output will be undefined)**. Esta configuración específica que no importa si la memoria da salida a los nuevos datos que se escriben o los datos almacenados previamente, esto para el caso de que las direcciones de escritura y lectura sean las mismas.

2. En la página 8 la cual se muestra en la figura 3.8.5 se debe seleccionar la opción **Yes, use this file for the memory content data**, luego dando clic en el botón **browse** despliega una ventana con la cual se puede buscar el archivo **ram32x4.mif** (memory initialization file), este archivo permite pre cargar datos en el bloque de memoria creado, un formato de este archivo se muestra en la figura 3.8.6.



**Figura No. 3.8.5: Selección de archivo MIF.**

```
DEPTH =32 ;
WIDTH=4 ;
ADDRESS_RADIX = HEX ;
DATA_RADIX = BIN ;
CONTENT
BEGIN
00:0000 ;
01:0001 ;
```



```
02:0010;  
03:0011;  
...  
1C:1110;  
1D:1111;  
1E:1110;  
1F:1000;  
END;
```

**Figura No. 3.8.6: Formato de archivo MIF**

- Finalice el asistente luego escriba un archivo VHDL que inicialice la memoria de doble puerto. Para ver el contenido de RAM se debe añadir la capacidad de mostrar el contenido de cada palabra de cuatro bits (en formato hexadecimal) en la display HEX0. Utilice un contador como proveedor de las direcciones de lectura, esto también permitirá desplazarse a través de las posiciones de memoria y la visualización de cada palabra debe durar aproximadamente un segundo. La dirección de igual forma debe ser mostrada (en formato hexadecimal) en los display HEX3 y HEX2. Use el reloj de 50 MHz (CLOCK\_50), incluido en la tarjeta DE1-SoC, también se debe utilizar KEY(0) como una entrada de reset. Para la asignar la dirección de escritura y el uso de datos se deben usar SW(8 a 4) y SW(3 a 0) correspondientemente. La dirección de escritura se debe mostrar en HEX5 y HEX4 y los datos de escritura se mostrarán en HEX1. Como se utiliza el reloj de 50 MHz debe asegurarse de sincronizar adecuadamente las entradas de los interruptores a la señal de reloj propuesta.
- Realice las pruebas necesarias al circuito y verifique que los contenidos iniciales de la memoria coincide con el archivo ram32x4.mif. Asegúrese de que se pueden escribir datos de forma independiente a cualquier dirección utilizando los interruptores.

## 3.10 Laboratorio No. 9: Procesador Simple

### 3.10.1 Introducción

Los procesadores son circuitos con la capacidad de efectuar operaciones lógicas con bits basando sus decisiones en respuesta a eventos o por ordenes predeterminadas, por lo general están compuestos por registros, sumadores, contadores y FSM.

Todos estos elementos ya se discutieron en los laboratorios anteriores, lo cual permite tener las herramientas necesarias para crear un procesador simple que cumpla con las operaciones básicas.

El archivo original de la práctica puede ser descargado a través del siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab9\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab9_VHDL.pdf) [en línea] [ultima consulta 24/02/2016]

### 3.10.2 Objetivos generales

- Asimilar los conceptos adquiridos previamente para la creación de un circuito de alta complejidad.
- Desarrollar una unidad aritmética lógica para un procesador.
- Aplicar los conceptos de FSM para crear el centro de decisiones de un procesador.
- Comprender de forma plena el uso de registros y la importancia del traslado de datos en un procesador.

### 3.10.3 Asignación 1

#### 3.10.3.1 Objetivo

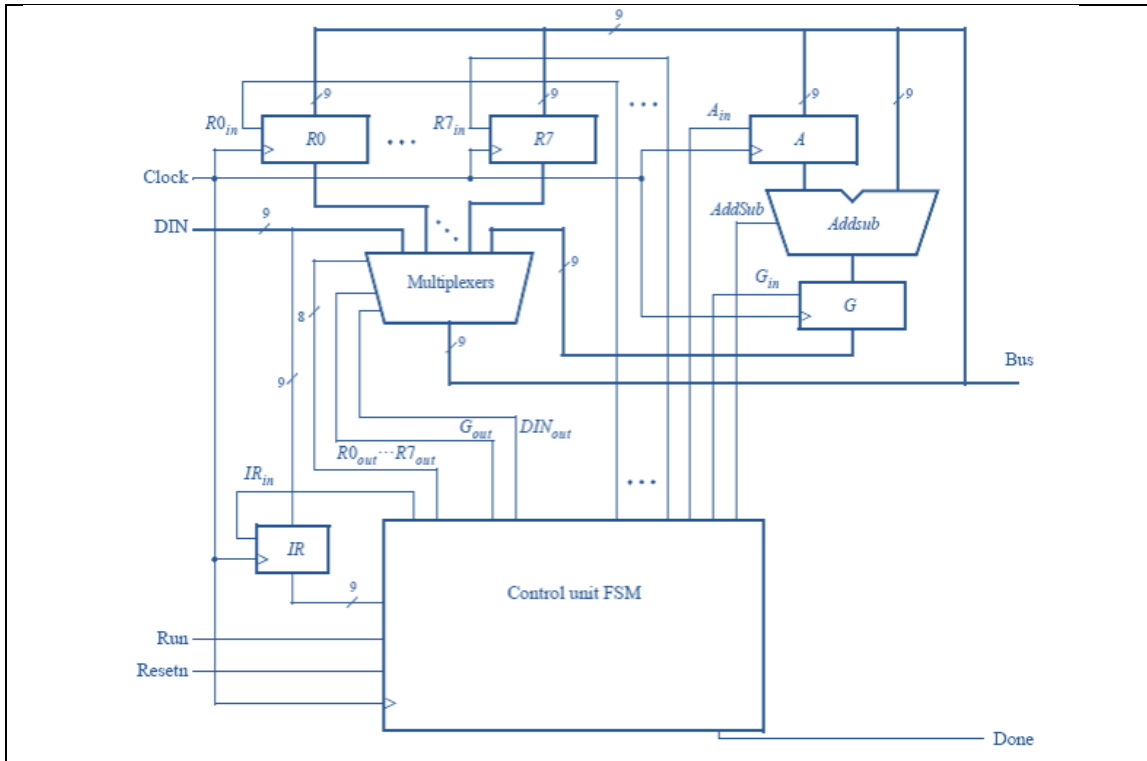
Crear un procesador simple capaz de manejar lotes de datos de 9 bits, con unidad de procesamiento aritmético y una unidad de control basada en FSM.

#### 3.10.3.2 Desarrollo

La figura 3.9.1 muestra un sistema digital que contiene registros de 9 bits, un multiplexor, una unidad sumadora/restadora y una unidad de control (máquina de estados finitos). Los datos se introducen a través de la entrada DIN la cual es de 9 bits. Una vez ingresados estos datos se pueden transportar en el sistema a través de un multiplexor de 9 bits el cual los lleva a los diferentes registros existentes, como lo son R0,..., R7 y A. El multiplexor también permite que los datos sean transferidos de un registro a otro. El cable de salida del multiplexor se le denominará un **bus** en la figura ya que este término se usa a menudo para el cableado que permite que los datos sean transferidos desde una ubicación a otra en un sistema cerrado.

La adición o sustracción se realiza utilizando el multiplexor para cargar el primer número de 9 bits en el bus que será transportado hasta el registro A. Una vez hecho esto, un segundo número de 9-bit se coloca en el bus, la unidad sumadora/restadora realiza la operación solicitada y el resultado se carga en el registro G. Los datos de G a continuación pueden ser transferidos a cualquiera de los otros registros según se requiera.

El sistema puede realizar diferentes operaciones en cada ciclo de reloj, estas operaciones se rigen por la unidad de control. Esta unidad determina cuando los datos se colocarán sobre el bus y controlará cuál de los registros se va a cargar con estos datos. Por ejemplo, si la unidad de control reconoce las señales R0out y Ain, entonces el multiplexor colocará el contenido del registro R0 en el bus y estos datos se cargan en la próxima transición activa de reloj en el registro A.



**Figura No. 3.9.1: Procesador Simple.** [15]

La Tabla 3.9.1 muestra las instrucciones que el procesador tiene que soportar en esta práctica. La columna de la izquierda muestra el nombre de una instrucción y su operando. El significado de la sintaxis  $Rx \leftarrow [Ry]$  es que el contenido del registro Ry se cargan en el registro Rx. La instrucción (mover) mv permite que los datos se copien de un registro a otro. Para la instrucción MVI (mover de inmediato) la expresión  $Rx \leftarrow D$  indica que la constante D se carga en el registro Rx, sin pasar por ningún otro registro.

Operation	Function performed
<b>mv</b> $Rx, Ry$	$Rx \leftarrow [Ry]$
<b>mvi</b> $Rx, \#D$	$Rx \leftarrow D$
<b>add</b> $Rx, Ry$	$Rx \leftarrow [Rx] + [Ry]$
<b>sub</b> $Rx, Ry$	$Rx \leftarrow [Rx] - [Ry]$

*Tabla No. 3.9.1: Instrucciones realizadas por procesador.* [15]

Cada instrucción puede ser codificada y almacenada en el registro IR usando el formato IIIXXXXYY de 9 bits, donde III representa la instrucción, XXX da el registro RX, y YYY da el registro RY. Aunque sólo se necesiten dos bits para codificar las cuatro instrucciones, se están usando tres bits porque otras instrucciones se añadirán al procesador en las asignaciones posteriores de este ejercicio. Por lo tanto IR tiene que ser conectado a los nueve bits de la entrada DIN, como se indica en la figura 3.9.1. Para la instrucción mvi el campo YYY no tiene sentido, ya que los datos #D tienen que ser suministrados por la entrada DIN de 9 bits después la palabra de instrucción mvi se almacenara en IR.

Algunas instrucciones como una adición o sustracción, toman más de un ciclo de reloj para completarse ya que múltiples transferencias tienen que ser realizadas a través del bus. La máquina de estados finitos es la encargada de llevar "pasos a paso" tales instrucciones, esto se realiza mediante la confirmación de señales de control necesarias para cada ciclo de reloj hasta que la instrucción haya finalizado. El procesador comienza a ejecutar la instrucción en la entrada DIN cuando se recibe la señal Run y el procesador envía la salida Done cuando haya finalizado la instrucción. La tabla 3.9.2 indica las señales de control que se deben confirmar en cada ciclo para poner en práctica las instrucciones de la tabla 3.9.1. Tenga en cuenta que la única señal de control que se confirma en la etapa 0 es IRIN, por lo que este ciclo no se muestra en la tabla.

	$T_1$	$T_2$	$T_3$
(mv): $I_0$	$RY_{out}, RX_{in},$ <i>Done</i>		
(mvi): $I_1$	$DIN_{out}, RX_{in},$ <i>Done</i>		
(add): $I_2$	$RX_{out}, A_{in}$	$RY_{out}, G_{in}$	$G_{out}, RX_{in},$ <i>Done</i>
(sub): $I_3$	$RX_{out}, A_{in}$	$RY_{out}, G_{in},$ <i>AddSub</i>	$G_{out}, RX_{in},$ <i>Done</i>

Tabla No. 3.9.2: Señales y ciclos necesarios para completar las instrucciones. <sup>[15]</sup>

### 3.10.3.3 Procedimiento

Diseñe e implemente el procesador mostrado en la figura 3.9.1 utilizando código VHDL siguiendo estos pasos:

1. Cree un nuevo proyecto Quartus II para este ejercicio.
2. Genere el archivo VHDL necesario, inclúyalo en el proyecto, y compile el circuito. Los códigos necesarios para realizar esta práctica se indica en la figura 3.9.2.

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;
```

```

ENTITY L9_PART1_AUP IS
PORT ( DIN :INSTD_LOGIC_VECTOR (8DOWNTO0) ;
        RESETN, CLOCK, RUN :INSTD_LOGIC ;
        DONE :BUFFERSTD_LOGIC ;
        BUSWIRES :BUFFERSTD_LOGIC_VECTOR (8DOWNTO0)) ;
END L9_PART1_AUP ;

ARCHITECTURE BEHAVIOR OF L9_PART1_AUP IS

COMPONENT UPCOUNT
PORT (CLEAR, CLOCK :INSTD_LOGIC ;
        Q :BUFFERSTD_LOGIC_VECTOR (1DOWNTO0)) ;
ENDCOMPONENT ;

COMPONENT DEC3TO8
PORT ( W :INSTD_LOGIC_VECTOR (2DOWNTO0) ;
        EN :INSTD_LOGIC ;
        Y :OUTSTD_LOGIC_VECTOR (0TO7)) ;
ENDCOMPONENT ;

COMPONENT REGN
GENERIC (N :INTEGER:=9) ;
PORT ( R :INSTD_LOGIC_VECTOR (N-1DOWNTO0) ;
        RIN, CLOCK :INSTD_LOGIC ;
        Q :BUFFERSTD_LOGIC_VECTOR (N-1DOWNTO0)) ;
ENDCOMPONENT ;

SIGNAL RIN, ROUT :STD_LOGIC_VECTOR (0TO7) ;
SIGNAL SUM :STD_LOGIC_VECTOR (8DOWNTO0) ;
SIGNAL CLEAR, HIGH, IRIN, DINOUT, AIN, GIN, GOUT, ADDSUB :STD_LOGIC ;
SIGNAL TSTEP_Q :STD_LOGIC_VECTOR (1DOWNTO0) ;
SIGNAL I :STD_LOGIC_VECTOR (2DOWNTO0) ;
SIGNAL XREG, YREG :STD_LOGIC_VECTOR (0TO7) ;
SIGNAL R0, R1, R2, R3, R4, R5, R6, R7, A, G :STD_LOGIC_VECTOR (8DOWNTO0) ;
SIGNAL IR :STD_LOGIC_VECTOR (1TO9) ;
SIGNAL SEL :STD_LOGIC_VECTOR (1TO10) ;

BEGIN
HIGH<= '1' ;
CLEAR <=NOT (RESETN)OR DONE
OR (NOT (RUN)ANDNOT (TSTEP_Q(0))ANDNOT (TSTEP_Q(0))) ;
TSTEP: UPCOUNT PORTMAP (CLEAR, CLOCK, TSTEP_Q) ;
I <= IR (1TO3) ;
DECX: DEC3TO8 PORTMAP (IR (4TO6), HIGH, XREG) ;
DECY: DEC3TO8 PORTMAP (IR (7TO9), HIGH, YREG) ;

CONTROLSIGNALS:PROCESS (TSTEP_Q, I, XREG, YREG)
BEGIN
--VALORES INICIALES
DONE <= '0' ; AIN <= '0' ; GIN <= '0' ; GOUT <= '0' ; ADDSUB <= '0' ;
IRIN <= '0' ; DINOUT <= '0' ; RIN <="00000000" ; ROUT <="00000000" ;
CASE TSTEP_Q IS
WHEN"00"=>-- STORE DIN IN IR AS LONG AS TSTEP_Q = 0
        IRIN <= '1' ;
WHEN"01"=>--DEFINE SIGNALS IN TIME STEP T1
CASE I IS

```

```

WHEN"000"=>
    ROUT    <= YREG;
    RIN <= XREG;
    DONE    <= '1';
WHEN"001"=>
    DINOUT  <= '1';
    RIN    <= XREG;
    DONE    <= '1';
WHEN"010"=>
    ROUT    <= XREG;
    AIN <= '1';
WHENOTHERS=>
    ROUT    <= XREG;
    AIN <= '1';
ENDCASE;
WHEN"10"=>-- DEFINE SIGNALS IN TIME STEP T2
CASE I IS
WHEN"010"=>
    ROUT    <= YREG;
    GIN <= '1';
WHENOTHERS=>
    ROUT    <= YREG;
    GIN <= '1';
    ADDSUB <= '1';
ENDCASE;
WHEN"11"=>-- DEFINE SIGNALS IN TIME STEP T3
CASE I IS
WHEN"010"=>
    GOUT    <= '1';
    RIN <= XREG;
    DONE    <= '1';
WHENOTHERS=>
    GOUT    <= '1';
    RIN <= XREG;
    DONE    <= '1';
ENDCASE;
ENDCASE;
ENDPROCESS;

REG_0: REGN PORTMAP (BUSWIRES, RIN(0), CLOCK, R0);
REG_1: REGN PORTMAP (BUSWIRES, RIN(1), CLOCK, R1);
REG_2: REGN PORTMAP (BUSWIRES, RIN(2), CLOCK, R2);
REG_3: REGN PORTMAP (BUSWIRES, RIN(3), CLOCK, R3);
REG_4: REGN PORTMAP (BUSWIRES, RIN(4), CLOCK, R4);
REG_5: REGN PORTMAP (BUSWIRES, RIN(5), CLOCK, R5);
REG_6: REGN PORTMAP (BUSWIRES, RIN(6), CLOCK, R6);
REG_7: REGN PORTMAP (BUSWIRES, RIN(7), CLOCK, R7);
REG_A: REGN PORTMAP (BUSWIRES, AIN, CLOCK, A);
REG_IR: REGN GENERICMAP (N =>9) PORTMAP (DIN, IRIN, CLOCK, IR);

--ALU
ALU: PROCESS (ADDSUB, A, BUSWIRES)
BEGIN
IF ADDSUB = '0' THEN
    SUM <= A + BUSWIRES;
ELSE
    SUM <= A - BUSWIRES;

```

```

ENDIF;
ENDPROCESS;

REG_G: REGN PORTMAP (SUM, GIN, CLOCK, G);

SEL <= ROUT & GOUT & DINOUT;

BUSMUX:PROCESS (SEL, R0, R1, R2, R3, R4, R5, R6, R7, G, DIN)
BEGIN
IF SEL ="1000000000"THEN
    BUSWIRES <= R0;
ELSIF SEL ="0100000000"THEN
    BUSWIRES <= R1;
ELSIF SEL ="0010000000"THEN
    BUSWIRES <= R2;
ELSIF SEL ="0001000000"THEN
    BUSWIRES <= R3;
ELSIF SEL ="0000100000"THEN
    BUSWIRES <= R4;
ELSIF SEL ="0000010000"THEN
    BUSWIRES <= R5;
ELSIF SEL ="0000001000"THEN
    BUSWIRES <= R6;
ELSIF SEL ="0000000100"THEN
    BUSWIRES <= R7;
ELSIF SEL ="0000000010"THEN
    BUSWIRES <= G;
ELSE
    BUSWIRES <= DIN;
ENDIF;
ENDPROCESS;
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.STD_LOGIC_SIGNED.ALL;

ENTITY UPCOUNT IS
PORT (CLEAR, CLOCK :INSTD_LOGIC;
      Q :BUFFERSTD_LOGIC_VECTOR (1DOWNTO0));
END UPCOUNT;

ARCHITECTURE BEHAVIOR OF UPCOUNT IS
SIGNAL COUNT :STD_LOGIC_VECTOR (1DOWNTO0);
BEGIN
PROCESS (CLOCK)
BEGIN
IF (CLOCK'EVENTAND CLOCK = '1') THEN
IF CLEAR = '1' THEN
    COUNT <="00";
ELSE
    COUNT <= COUNT +1;
ENDIF;
ENDIF;
Q <= COUNT;
ENDPROCESS;
END BEHAVIOR;

```

```

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY DEC3TO8 IS
PORT ( W :INSTD_LOGIC_VECTOR (2DOWNT0) ;
      EN :INSTD_LOGIC ;
      Y :OUTSTD_LOGIC_VECTOR (0TO7) ) ;
END DEC3TO8 ;

ARCHITECTURE BEHAVIOR OF DEC3TO8 IS
BEGIN
PROCESS (W, EN)
BEGIN
IF EN = '1' THEN
CASE W IS
WHEN "000" => Y <= "10000000" ;
WHEN "001" => Y <= "01000000" ;
WHEN "010" => Y <= "00100000" ;
WHEN "011" => Y <= "00010000" ;
WHEN "100" => Y <= "00001000" ;
WHEN "101" => Y <= "00000100" ;
WHEN "110" => Y <= "00000010" ;
WHEN "111" => Y <= "00000001" ;
ENDCASE ;
ELSE
Y <= "00000000" ;
ENDIF ;
ENDPROCESS ;
END BEHAVIOR ;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY REGN IS
GENERIC (N :INTEGER:=9) ;
PORT ( R :INSTD_LOGIC_VECTOR (N-1DOWNT0) ;
      RIN, CLOCK :INSTD_LOGIC ;
      Q :BUFFERSTD_LOGIC_VECTOR (N-1DOWNT0) ) ;
END REGN ;

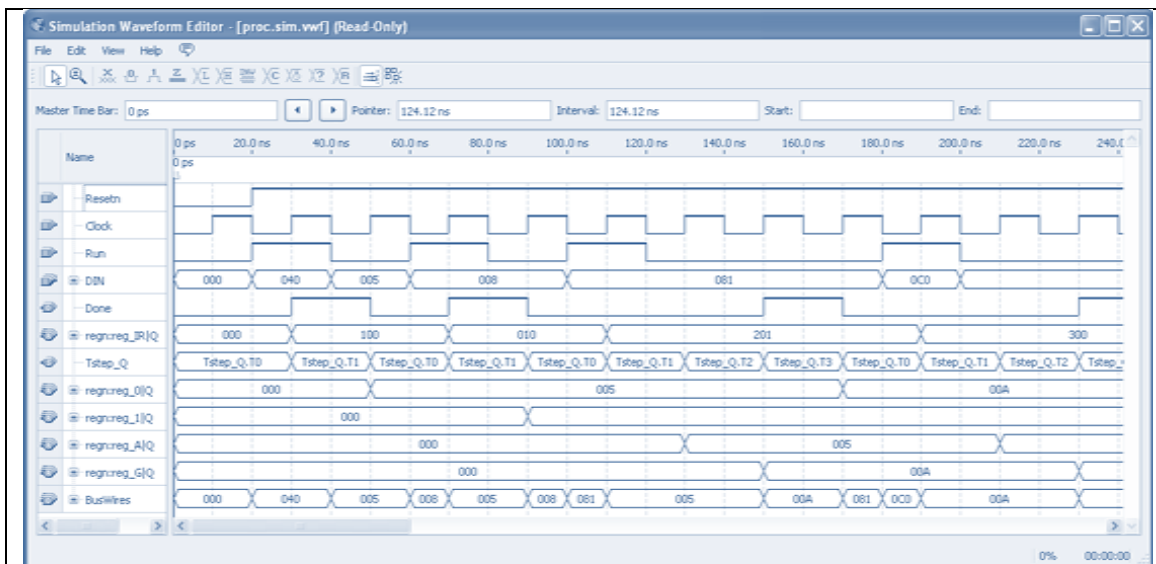
ARCHITECTURE BEHAVIOR OF REGN IS
BEGIN
PROCESS (CLOCK)
BEGIN
IF (CLOCK'EVENTAND CLOCK = '1') THEN
IF RIN = '1' THEN
Q <= R ;
ENDIF ;
ENDIF ;
ENDPROCESS ;
END BEHAVIOR ;

```

**Figura No. 3.9.2: Código para asignación 1**



- Utilice simulación funcional para comprobar que el código es correcto. Un ejemplo de la salida producida por una simulación funcional para un circuito correctamente diseñado se da en la figura 3.9.3. Se está cargando el valor 040 (hexadecimal) en IR a través de DIN en el tiempo 30 ns. Este patrón (los bits más a la izquierda en DIN están conectados a IR) representa la instrucción `mvi R0, # D`, donde el valor `D = 5` se carga en `R0` en el flanco de reloj a los 50 ns. La simulación muestra entonces la instrucción `mv R1 ← R0` a los 90 ns, la suma `R0 ← R1` a 110 ns, y la resta `R0 ← R0` a 190 ns. Tenga en cuenta que la salida de la simulación muestra DIN como un número hexadecimal de 3 dígitos y muestra el contenido de IR como un número octal de 3 dígitos.
- Añada al proyecto la asignación de pines necesarios para la tarjeta DE1-SoC. Compile el circuito y descargarlo en el chip FPGA.
- Pruebe la funcionalidad de su diseño alternando los interruptores y observe el comportamiento del circuito a través de los LEDs.



**Figura No. 3.9.3 Simulación del procesador simple.**

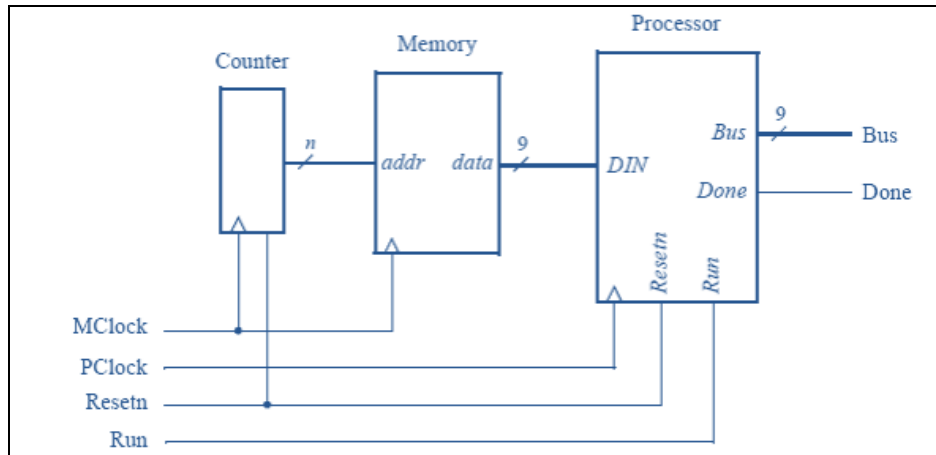
## 3.10.4 Asignación 2

### 3.10.4.1 Objetivo

En esta práctica se debe diseñar el circuito representado en la Figura 3.9.44, en la que un módulo de memoria y un contador están conectados al procesador de la Parte I. Se utiliza el contador para leer el contenido de las direcciones sucesivas en la memoria, y estos datos se trasladan al procesador como un flujo de instrucciones. Para simplificar el diseño e implementación de este circuito se utilizarán señales de reloj separadas, PCKlock y MCKlock, para el procesador y la memoria respectivamente.

### 3.10.4.2 Desarrollo

La figura 3.9.4 muestra un circuito de introducción de instrucciones a través de una memoria ROM (Read Only Memory), este circuito debe ser unido con el de la sección anterior para crear un procesador simple basado en memoria ROM.



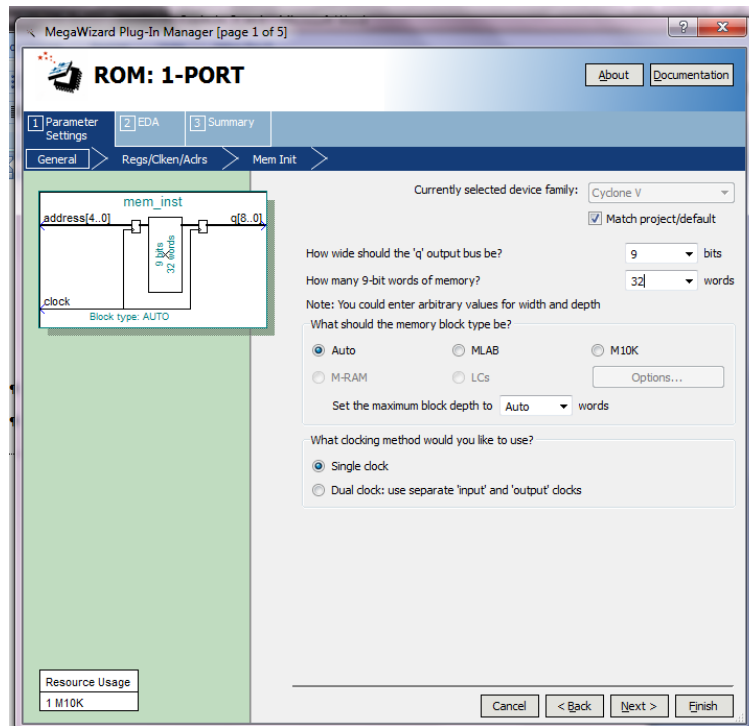
**Figura No. 3.9.4: Procesador mas bloque ROM.[15]**

### 3.10.4.3 Procedimiento

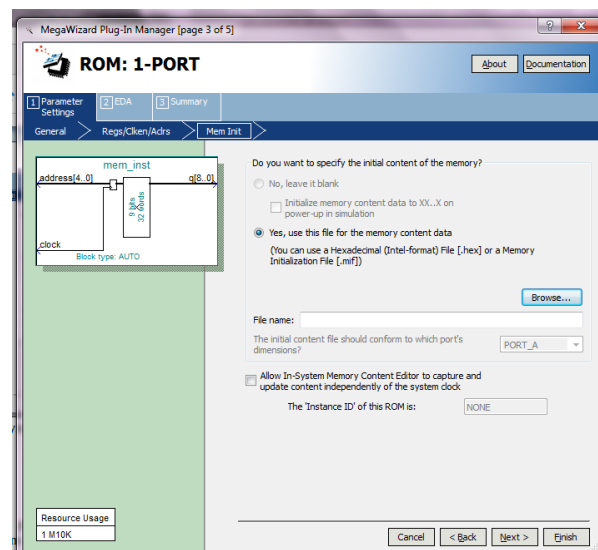
Para realizar esta práctica se deben seguir los siguientes pasos:

1. Cree un nuevo proyecto Quartus II que se utilizará para probar el circuito.
2. Genere un archivo VHDL de alto nivel que inicialice el procesador, la memoria, y el contador. El módulo de memoria ROM se creará utilizando módulos IP siguiendo el procedimiento que se proporciona en el Laboratorio Ejercicio 8. El módulo correcto se llama ROM: 1-PORT. Siga las instrucciones proporcionadas por el asistente para crear una memoria que debe tener 32 palabras de 9 bits cada una, esta configuración se ajusta en la página 3 del asistente, la cual se muestra en la Figura 3.9.5. Avance a la página siguiente y desmarque el ajuste llamado '**q**' output port en la categoría **Which ports should be registered?**, dado que esta memoria sólo tiene un puerto de lectura, y ningún puerto de escritura, se le conoce como memoria de lectura síncrona (**synchronous ROM**). Tenga en cuenta que la memoria incluye un registro de direcciones para cargar de forma síncrona. Este registro es necesario debido al diseño de los recursos de memoria en la FPGA Cyclone V, como se discutió en Laboratorio 8.
3. Para colocar instrucciones del procesador a la memoria, es necesario especificar los valores iniciales que se deben almacenar en la memoria una vez que su circuito ha sido programado en el chip FPGA. Esto se logra mediante un archivo MIF, el cual se puede elegir en la pagina 3 y que se ilustra en la figura 3.9.6. Se ha especificado un archivo llamado inst\_mem.mif, que luego tiene que ser añadido al directorio de archivos que contiene el proyecto Quartus II. Un ejemplo de un archivo de inicialización de memoria se proporciona en Laboratorio Ejercicio 8. También puede utilizar Quartus II Help en línea para aprender sobre el formato del archivo MIF.

4. Establezca el contenido de su archivo MIF tal que proporcione suficientes instrucciones del procesador para probar el circuito.



**Figura No. 3.9.5: Configuración del tamaño de la memoria.**



**Figura No 3.9.6: Asignación del archivo MIF para el bloque ROM.**

5. Utilice la simulación funcional para probar el circuito. Asegúrese de que los datos se lean correctamente fuera de la ROM y sean ejecutados por el procesador.
6. Asegúrese de que el proyecto incluye los nombres de los pines necesarios para implementar el circuito en la tarjeta DE1-SoC. Utilice el interruptor SW(9) para activar la entrada Run del procesador, use KEY0 para la función Reset, así mismo asigne KEY1 para MClock y utilice KEY2 para PCLock. Conecte los datos del bus del procesador a LEDR(8 a 0) y conecte la señal DONE a LEDR(9).
7. Compile el circuito y descargarlo en el chip FPGA.
8. Pruebe la funcionalidad del diseño alternando los interruptores y observe los resultados en los LEDs.

## 3.11 Laboratorio No. 10: Procesador avanzado

### 3.11.1 Introducción

En el laboratorio 9 se implementó un procesador simple. En la primera parte de ese ejercicio el procesador fue diseñado y en la segunda asignación el procesador estaba conectado a un contador externo y una unidad de memoria. Para este laboratorio se describen las asignaciones posteriores del diseño del procesador, de forma que se logre un procesador avanzado.

El archivo original de la práctica puede obtenerse a través del siguiente enlace: [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-Soc/vhdl/lab10\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-Soc/vhdl/lab10_VHDL.pdf) [en línea] [última consulta 24/02/2016].

### 3.11.2 Objetivos Generales

- Ampliar la cantidad de instrucciones que puede manejar el procesador ya creado.
- Implementar un procesador independiente de un contador y memoria fija.
- Añadir un módulo de escritura para bloques de memoria externos.

### 3.11.3 Asignación 1

#### 3.11.3.1 Objetivo

En esta práctica se podrá ampliar la capacidad del procesador de manera que ya no es necesario el contador externo, y el procesador deberá tener la capacidad de realizar operaciones de lectura y escritura utilizando la memoria u otros dispositivos.

#### 3.11.3.2 Desarrollo

Se añadirán tres nuevas instrucciones para el procesador, como se muestra en la tabla 3.10.1. La instrucción LD (Load) carga datos en el registro RX de la dirección de memoria externa especificados en el registro RY. La instrucción ST (Store) almacena los datos contenidos en el registro RX en la dirección de memoria almacenada en el registro RY. Por último, la instrucción mvnz (move if not zero) permite una operación mv para ser ejecutada únicamente cuando el contenido actual del registro G sea diferente de 0.

Operation	Function performed
ld $R_x, [R_y]$	$R_x \leftarrow [[R_y]]$
st $R_x, [R_y]$	$[R_y] \leftarrow [R_x]$
mvnz $R_x, R_y$	if $G \neq 0, R_x \leftarrow [R_y]$

Tabla No. 3.10.1: Nuevas instrucciones para el procesador avanzado.<sup>[16]</sup>

Un diagrama esquemático del procesador avanzado se da en la figura 3.10.1. Los registros R0 a R6 son los mismos que en la figura 3.9.1 de Laboratorio 9, pero el registro R7 se ha cambiado a un

contador. Este contador se utiliza para proporcionar las direcciones de la memoria desde la cual se leen las instrucciones del procesador; en el ejercicio del laboratorio anterior, se utilizó un contador externo al procesador para este propósito. En adelante R7 se conocerá como contador de programa del procesador (PC), porque esta terminología es común para los procesadores reales disponibles en la industria. Cuando el procesador se restablece, PC se establece en la dirección 0. Al comienzo de cada instrucción (en la etapa 0) el contenido de PC se utiliza como una dirección para leer una instrucción de la memoria. La instrucción se almacena en IR y PC se incrementa automáticamente para que apunte a la siguiente instrucción (en el caso de mvi el PC proporciona la dirección de los datos inmediatos y luego se incrementa de nuevo).

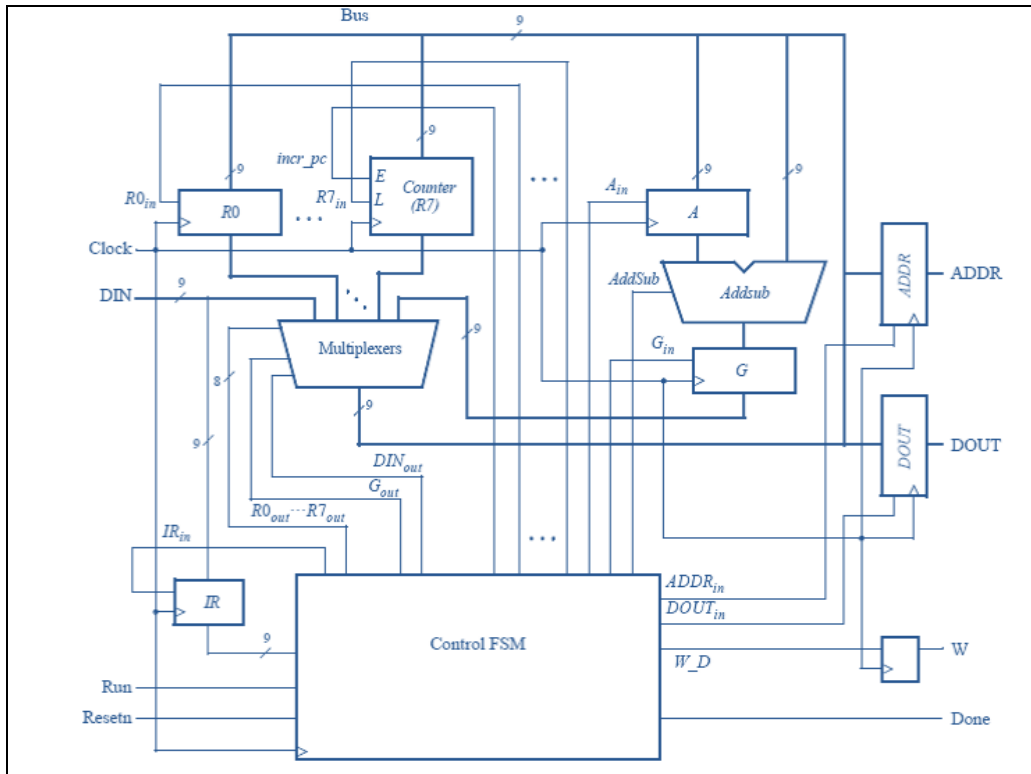
La unidad de control del procesador incrementa PC mediante el uso de la señal incr\_PC, que es sólo un habilitador para este contador. También es posible cargar directamente una dirección en PC (R7) haciendo que el procesador ejecute una instrucción mv o mvi en el que se especifica el registro de destino como R7. En este caso la unidad de control utiliza la señal R7in para realizar una carga en paralelo del contador. De esta manera, el procesador puede ejecutar instrucciones en cualquier dirección en la memoria, en lugar de sólo ser capaz de ejecutar instrucciones que se almacenan en direcciones sucesivas. Del mismo modo, el contenido actual del PC se puede copiar en otro registro mediante el uso de una instrucción mv. Un ejemplo de código que utiliza el registro PC para implementar un bucle se muestra a continuación, donde el texto después del símbolo % en cada línea es sólo un comentario. La instrucción mv R5 ← R7 coloca en R5 la dirección de memoria de la instrucción sub R4 ← R2. Entonces, la instrucción mvnz R7 ← R5 hace que la instrucción sub sea ejecutada varias veces hasta que R4 se convierta en 0. Este tipo de bucle podría ser utilizado en un programa más complejo como alternativa para crear un delay.

```

mvi R2,#1
mvi R4,#10000000 % binary delay value
mv R5,R7 % save address of next instruction
sub R4,R2 % decrement delay count
mvnz R7,R5 % continue subtracting until delay count gets to 0

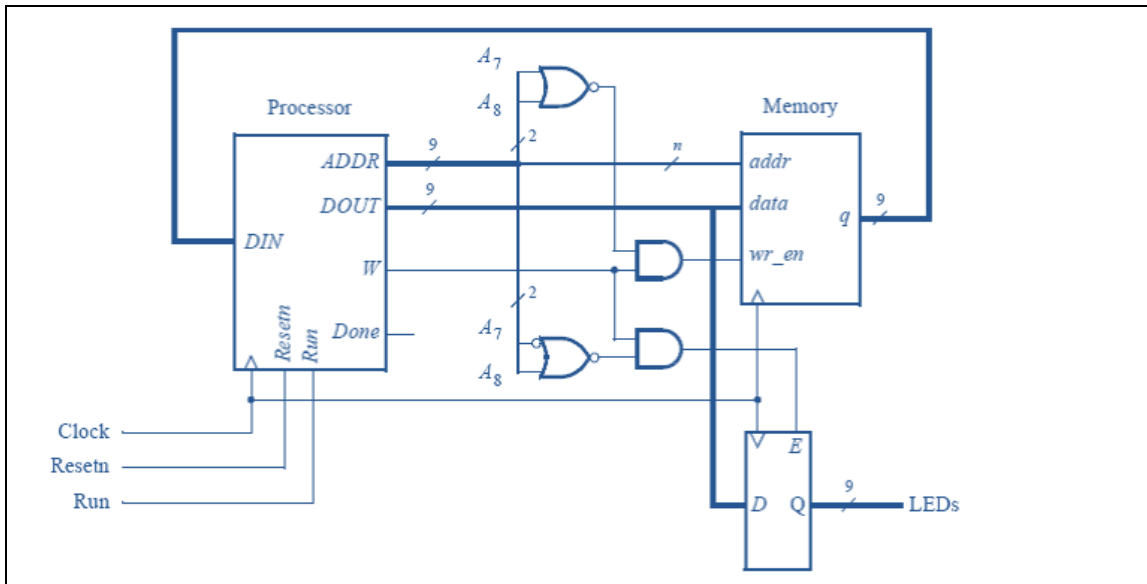
```

La figura 3.10.1 muestra dos registros en el procesador que se utilizan para las transferencias de datos. El registro ADDR se utiliza para enviar direcciones a un dispositivo externo, tal como un módulo de memoria, y el registro DOUT es utilizado por el procesador para proporcionar datos que se pueden almacenar fuera del procesador. Uno de los usos del registro ADDR es para leer o buscar las instrucciones de la memoria; cuando el procesador quiere buscar una instrucción el contenido de PC (R7) se transfiere a través del bus y se carga en ADDR. Esta dirección se proporciona a la memoria. Además de ir a buscar instrucciones, el procesador puede leer datos en cualquier dirección utilizando el registro ADDR. Tanto los datos y las instrucciones se leen en el puerto de entrada DIN. El procesador puede escribir datos para el almacenamiento en una dirección externa mediante la colocación de esta dirección en el registro ADDR, la colocación de los datos a ser almacenados en el registro DOUT y ajustando la salida del flip flop W (escritura) a 1.



**Figura No. 3.10.1: Procesador Avanzado. [16]**

La figura 3.10.2 ilustra cómo el procesador avanzado está conectado a una memoria y otros dispositivos. La unidad de memoria en la figura soporta operaciones de lectura y de escritura y por lo tanto tiene entradas de direcciones y datos, así como un habilitador de escritura. La memoria también tiene una entrada de reloj, porque la dirección, los datos y los habilitadores deben ser cargados en la memoria con una transición de memoria positiva. Este tipo de unidad generalmente se denomina como memoria estática de acceso aleatorio síncrono (SRAM synchronous). La figura 3.10.2 también incluye un registro de 9 bits que se puede utilizar para almacenar datos procedentes del procesador; este registro puede estar conectado a un conjunto de LED para permitir la visualización de los datos en la tarjeta DE1-SoC. Para permitir que el procesador pueda seleccionar la unidad de memoria o se registren los datos cuando se realiza una operación de escritura, el circuito incluye algunas puertas lógicas que realizan la decodificación de direcciones. La Figura 3.10.2 muestra  $n$  líneas de dirección inferiores conectados a la memoria; en este ejercicio una memoria con 128 palabras es probablemente suficiente, lo que implica que  $n = 7$  y el puerto de dirección de memoria es impulsado por A6 hasta A0.



**Figura No. 3.10.2: Interconexión de un procesador avanzado.<sup>[16]</sup>**

### 3.11.3.3 Procedimiento

Para completar esta práctica se deben realizar los siguientes pasos:

1. Cree un nuevo proyecto Quartus II para la versión mejorada del procesador.
2. Traslade el código de la figura 3.10.3 hacia un archivo VHDL el cual define el procesador avanzado y pruebe el circuito utilizando simulación funcional, en la simulación agregue las instrucciones al que involucren el puerto DIN y observe las señales internas del procesador y como estas ejecutan las instrucción. Preste especial atención a la sincronización de las señales entre el procesador y la memoria externa.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;

ENTITY PROC IS
PORT( DIN :IN STD_LOGIC_VECTOR(8DOWNTO0);
      RESETN, CLOCK, RUN :IN STD_LOGIC;
      DOUT :OUT STD_LOGIC_VECTOR(8DOWNTO0);
      ADDR :OUT STD_LOGIC_VECTOR(8DOWNTO0);
      W :OUT STD_LOGIC;
      DONE :BUFFER STD_LOGIC);
END PROC;

ARCHITECTURE BEHAVIOR OF PROC IS

COMPONENT UPCOUNT
PORT(CLEAR, CLOCK :IN STD_LOGIC;
      Q :BUFFER STD_LOGIC_VECTOR(2DOWNTO0));
ENDCOMPONENT;

COMPONENT PC_COUNT
PORT(R :IN STD_LOGIC_VECTOR(8DOWNTO0);
      RESETN, CLOCK, E, L :IN STD_LOGIC;

```



```

        Q          :OUT STD_LOGIC_VECTOR(8DOWNTO0));
END COMPONENT;

COMPONENT DEC3TO8
PORT ( W :IN STD_LOGIC_VECTOR(2DOWNTO0);
      EN :IN STD_LOGIC;
      Y :OUTSTD_LOGIC_VECTOR(0TO7));
END COMPONENT;

COMPONENT REGN
GENERIC(N :INTEGER:=9);
PORT ( R :IN STD_LOGIC_VECTOR((N-1)DOWNTO0);
      RIN, CLOCK :IN STD_LOGIC;
      Q :BUFFER STD_LOGIC_VECTOR((N-1) DOWNTO0));
END COMPONENT;

COMPONENT FLIPFLOP
PORT(D, RESETN, CLOCK :IN STD_LOGIC;
      Q :OUT STD_LOGIC);
END COMPONENT;

SIGNAL RIN, ROUT      :STD_LOGIC_VECTOR(0TO7);
SIGNAL BUSWIRES, SUM  :STD_LOGIC_VECTOR(8DOWNTO0);
SIGNAL CLEAR, IRIN, ADDRIN, DINOUT, DOUTIN, AIN, GIN, GOUT, ADDSUB :STD_LOGIC;
SIGNAL TSTEP_Q        :STD_LOGIC_VECTOR(2DOWNTO0);
SIGNAL I               :STD_LOGIC_VECTOR(2DOWNTO0);
SIGNAL XREG, YREG      :STD_LOGIC_VECTOR(0TO7);
SIGNAL R0, R1, R2, R3, R4, R5, R6, R7, A, G :STD_LOGIC_VECTOR(8DOWNTO0);
SIGNAL IR              :STD_LOGIC_VECTOR(1TO9);
SIGNAL SEL             :STD_LOGIC_VECTOR(1TO10);
SIGNAL PC_INC, W_D, Z, Z_D :STD_LOGIC;

BEGIN
--HIGH <= '1';
CLEAR <=NOT(RESETN)OR DONE OR(NOT(RUN)ANDNOT(TSTEP_Q(0))ANDNOT(TSTEP_Q(0)));
TSTEP: UPCOUNT PORT MAP(CLEAR, CLOCK, TSTEP_Q);
I <= IR(1TO3);
DECX: DEC3TO8 PORT MAP(IR(4TO6), '1', XREG);
DECY: DEC3TO8 PORT MAP(IR(7TO9), '1', YREG);

CONTROLSIGNALS:PROCESS(TSTEP_Q, I, XREG, YREG, Z, RUN)
BEGIN
--VALORES INICIALES
DONE <= '0'; AIN <= '0'; GIN <= '0'; GOUT <= '0'; ADDSUB <= '0';
IRIN <= '0'; DINOUT <= '0'; DOUTIN <= '0'; W_D <= '0'; ADDRIN <= '0';
RIN <="00000000"; ROUT <="00000000";
CASE TSTEP_Q IS
WHEN"000"=>-- STORE DIN IN IR AS LONG AS TSTEP_Q = 0
    ROUT <="00000001";
    ADDRIN <= '1';
    PC_INC <= RUN;
WHEN"001"=>-- STORE DIN IN IR AS LONG AS TSTEP_Q = 0
    ROUT <="00000001";
    ADDRIN <= '1';
WHEN"010"=>-- STORE DIN IN IR AS LONG AS TSTEP_Q = 0
-- ROUT <= "00000001";
    IRIN <= '1';
WHEN"011"=>--DEFINE SIGNALS IN TIME STEP T1
CASE I IS
WHEN"000"=>
    ROUT <= YREG;
    RIN <= XREG;
    DONE <= '1';

```

```

WHEN "001"=>
    DINOUT <= '1';
    RIN <= XREG;
    PC_INC <= '1';
    DONE <= '1';

WHEN "010"=>
    ROUT <= XREG;
    AIN <= '1';

WHEN "011"=>
    ROUT <= XREG;
    AIN <= '1';

WHEN "100"=>
    ROUT <= YREG;
    ADDRIN <= '1';

WHEN OTHERS=>
    IF Z = '0' THEN
        ROUT <= YREG;
        RIN <= XREG;
    ELSE
        RIN <="00000000";
        ROUT <="00000000";
    END IF;
    DONE <= '1';
END CASE;
WHEN "100"=>-- DEFINE SIGNALS IN TIME STEP T2
CASE I IS
WHEN "010"=>
    ROUT <= YREG;
    GIN <= '1';

WHEN "011"=>
    ROUT <= YREG;
    GIN <= '1';
    ADDSUB <= '1';

WHEN "100"=>
    W_D <= '0';

WHEN OTHERS=>
    ROUT <= XREG;
    DOUTIN <= '1';
    W_D <= '1';
END CASE;
WHEN OTHERS=>-- DEFINE SIGNALS IN TIME STEP T3
CASE I IS
WHEN "010"=>
    GOUT <= '1';
    RIN <= XREG;
    DONE <= '1';

WHEN "011"=>
    GOUT <= '1';
    RIN <= XREG;
    DONE <= '1';

WHEN "100"=>
    DINOUT <= '1';
    RIN <= XREG;
    DONE <= '1';

WHEN OTHERS=>
    DONE <= '1';
END CASE;
END CASE;
END PROCESS;

REG_0: REGN PORT MAP(BUSWIRES, RIN(0), CLOCK, R0);
REG_1: REGN PORT MAP(BUSWIRES, RIN(1), CLOCK, R1);
REG_2: REGN PORT MAP(BUSWIRES, RIN(2), CLOCK, R2);

```

```

REG_3: REGN PORT MAP(BUSWIRES, RIN(3), CLOCK, R3);
REG_4: REGN PORT MAP(BUSWIRES, RIN(4), CLOCK, R4);
REG_5: REGN PORT MAP(BUSWIRES, RIN(5), CLOCK, R5);
REG_6: REGN PORT MAP(BUSWIRES, RIN(6), CLOCK, R6);
PC: PC_COUNT PORT MAP(BUSWIRES, RESETN, CLOCK, PC_INC, RIN(7), R7);

REG_A: REGN PORT MAP(BUSWIRES, AIN, CLOCK, A);
REG_DOUT: REGN PORT MAP(BUSWIRES, DOUTIN, CLOCK, DOUT);
REG_ADDR: REGN PORT MAP(BUSWIRES, ADDRIN, CLOCK, ADDR);
REG_IR: REGN GENERIC MAP(N =>9) PORTMAP(DIN, IRIN, CLOCK, IR);
REG_W: FLIPFLOP PORT MAP(W_D, RESETN, CLOCK, W);
--ALU
ALU: PROCESS(ADDSUB, A, BUSWIRES)
BEGIN
IF ADDSUB = '0' THEN
    SUM <= A + BUSWIRES;
ELSE
    SUM <= A - BUSWIRES;
END IF;
END PROCESS;

REG_G: REGN PORT MAP(SUM, GIN, CLOCK, G);

Z_D <= '1' WHEN(G = 0) ELSE '0';

REG_Z: FLIPFLOP PORT MAP(Z_D, RESETN, CLOCK, Z);

SEL <= ROUT & GOUT & DINOUT;

BUSMUX: PROCESS(SEL, R0, R1, R2, R3, R4, R5, R6, R7, G, DIN)
BEGIN
IF SEL ="1000000000" THEN
    BUSWIRES <= R0;
ELSIF SEL ="0100000000" THEN
    BUSWIRES <= R1;
ELSIF SEL ="0010000000" THEN
    BUSWIRES <= R2;
ELSIF SEL ="0001000000" THEN
    BUSWIRES <= R3;
ELSIF SEL ="0000100000" THEN
    BUSWIRES <= R4;
ELSIF SEL ="0000010000" THEN
    BUSWIRES <= R5;
ELSIF SEL ="0000001000" THEN
    BUSWIRES <= R6;
ELSIF SEL ="0000000100" THEN
    BUSWIRES <= R7;
ELSIF SEL ="0000000010" THEN
    BUSWIRES <= G;
ELSE
    BUSWIRES <= DIN;
END IF;
END PROCESS;
END BEHAVIOR;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;

ENTITY PC_COUNT IS
PORT(R
    :IN STD_LOGIC_VECTOR(8DOWNTO0);
    RESETN, CLOCK, E, L :IN STD_LOGIC;
    Q
    :OUT STD_LOGIC_VECTOR(8DOWNTO0));

```

```

END PC_COUNT;

ARCHITECTURE BEHAVIOR OF PC_COUNT IS
SIGNAL COUNT :STD_LOGIC_VECTOR(8DOWNTO0);
BEGIN
PROCESS (CLOCK)
BEGIN
IF (CLOCK'EVENT AND CLOCK = '1') THEN
IF RESETN = '0' THEN
COUNT <= (OTHERS=> '0');
ELSIF (L = '1') THEN
COUNT <= R;
ELSIF (E = '1') THEN
COUNT <= COUNT +1;
END IF;
END IF;
Q <= COUNT;
END PROCESS;
END BEHAVIOR;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;

ENTITY UPCOUNT IS
PORT (CLEAR, CLOCK :IN STD_LOGIC;
Q :BUFFER STD_LOGIC_VECTOR(2DOWNTO0));
END UPCOUNT;

ARCHITECTURE BEHAVIOR OF UPCOUNT IS
SIGNAL COUNT :STD_LOGIC_VECTOR(2DOWNTO0);
BEGIN
PROCESS (CLOCK)
BEGIN
IF (CLOCK'EVENT AND CLOCK = '1') THEN
IF CLEAR = '1' THEN
COUNT <= "000";
ELSE
COUNT <= COUNT +1;
END IF;
END IF;
Q <= COUNT;
END PROCESS;
END BEHAVIOR;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY DEC3TO8 IS
PORT ( W :IN STD_LOGIC_VECTOR(2DOWNTO0);
EN :IN STD_LOGIC;
Y :OUT STD_LOGIC_VECTOR(0TO7));
END DEC3TO8;

ARCHITECTURE BEHAVIOR OF DEC3TO8 IS
BEGIN
PROCESS (W, EN)
BEGIN
IF EN = '1' THEN
CASE W IS
WHEN "000" => Y <= "10000000";
WHEN "001" => Y <= "01000000";
WHEN "010" => Y <= "00100000";

```

```

WHEN "011" => Y <= "00010000";
WHEN "100" => Y <= "00001000";
WHEN "101" => Y <= "00000100";
WHEN "110" => Y <= "00000010";
WHEN "111" => Y <= "00000001";
ENDCASE;
ELSE
    Y <= "00000000";
ENDIF;
ENDPROCESS;
END BEHAVIOR;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY REGN IS
    GENERIC (N : INTEGER := 9);
    PORT ( R : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
          RIN, CLOCK : IN STD_LOGIC;
          Q : BUFFER STD_LOGIC_VECTOR (N-1 DOWNTO 0));
END REGN;

ARCHITECTURE BEHAVIOR OF REGN IS
    BEGIN
        PROCESS (CLOCK)
            BEGIN
                IF (CLOCK'EVENT AND CLOCK = '1') THEN
                    IF RIN = '1' THEN
                        Q <= R;
                    END IF;
                END IF;
            END PROCESS;
        END BEHAVIOR;

```

**Figura No. 3.10.3: Código para procesador avanzado.**

3. Cree otro proyecto Quartus II que inicialice el procesador, memoria y registros, los códigos se muestran en la figura 3.10.4 y 3.10.5, en el caso de la memoria lo recomendable es crear un modulo IP de una memoria RAM: 1-port, esta memoria debe ser de 128 palabras de 9 bits. Utilice un archivo MIF para almacenar las instrucciones que van a ser ejecutadas por el procesador. Un programa de ejemplo en forma de archivo MIF se muestra en la Figura 3.10.6.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FLIPFLOP IS
    PORT (D, RESETN, CLOCK : IN STD_LOGIC;
          Q : OUT STD_LOGIC);
END FLIPFLOP;

ARCHITECTURE BEHAVIOR OF FLIPFLOP IS
    BEGIN
        PROCESS (CLOCK)
            BEGIN
                IF (CLOCK'EVENT AND CLOCK = '1') THEN
                    IF RESETN = '0' THEN
                        Q <= '0';
                    END IF;
                END IF;
            END PROCESS;
        END BEHAVIOR;

```

```

ELSE
    Q <= D;
ENDIF;
ENDIF;
ENDPROCESS;
END BEHAVIOR;

```

**Figura No. 3.10.4: Registros.**

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY L10_PART1_AUP IS
PORT (KEY :IN STD_LOGIC_VECTOR (0DOWNTO0) ;
      SW :IN STD_LOGIC_VECTOR (9DOWNTO0) ;
      CLOCK_50 :IN STD_LOGIC ;
      LEDR :OUT STD_LOGIC_VECTOR (9DOWNTO0) ;
      DOUT, ADDR :BUFFER STD_LOGIC_VECTOR (8DOWNTO0) ;
      W, DONE :BUFFER STD_LOGIC ;
      HEX0, HEX1, HEX2, HEX3, HEX4, HEX5 :OUT STD_LOGIC_VECTOR (0TO6) );
END L10_PART1_AUP;

ARCHITECTURE BEHAVIOR OF L10_PART1_AUP IS
COMPONENT PROC
PORT ( DIN :IN STD_LOGIC_VECTOR (8DOWNTO0) ;
      RESETN, CLOCK, RUN :IN STD_LOGIC ;
      DOUT :OUT STD_LOGIC_VECTOR (8DOWNTO0) ;
      ADDR :OUT STD_LOGIC_VECTOR (8DOWNTO0) ;
      W :OUT STD_LOGIC ;
      DONE :BUFFER STD_LOGIC );
END COMPONENT;

COMPONENT MEM_INST
PORT
(
    ADDRESS :IN STD_LOGIC_VECTOR (6DOWNTO0) ;
    CLOCK :IN STD_LOGIC:= '1';
    DATA :IN STD_LOGIC_VECTOR (8DOWNTO0) ;
    WREN :IN STD_LOGIC ;
    Q :OUT STD_LOGIC_VECTOR (8DOWNTO0)
);
END COMPONENT;

COMPONENT FLIPFLOP
PORT (D, RESETN, CLOCK :IN STD_LOGIC ;
      Q :OUT STD_LOGIC );
END COMPONENT;

COMPONENT REGN
GENERIC (N :INTEGER:=9) ;
PORT ( R :IN STD_LOGIC_VECTOR (N-1DOWNTO0) ;
      RIN, CLOCK :IN STD_LOGIC ;
      Q :BUFFER STD_LOGIC_VECTOR (N-1DOWNTO0) );
END COMPONENT;

SIGNAL SYNC, RUN, INST_MEM_ES, LEDR_REG_ES :STD_LOGIC ;
SIGNAL DIN, LEDR_REG :STD_LOGIC_VECTOR (8DOWNTO0) ;

BEGIN
HEX0 <="0000000";

```

```

HEX1 <="00000000";
HEX2 <="00000000";
HEX3 <="00000000";
HEX4 <="00000000";
HEX5 <="00000000";

U1: FLIPFLOP PORT MAP(SW(9), KEY(0), CLOCK_50, SYNC);
U2: FLIPFLOP PORT MAP(SYNC, KEY(0), CLOCK_50, RUN);

U3: PROC PORT MAP(DIN, KEY(0), CLOCK_50, RUN, DOUT, ADDR, W, DONE);

INST_MEM_ES <= '1' WHEN(ADDR(8DOWNTO7)="00")ELSE '0';
U4: MEM_INST PORT MAP(ADDR(6DOWNTO0), CLOCK_50, DOUT, (INST_MEM_ES AND W), DIN);

LEDR_REG_ES <= '1' WHEN(ADDR(8DOWNTO7)="01")ELSE '0';
U5: REGN PORT MAP(DOUT, (LEDR_REG_ES AND W), CLOCK_50, LEDR_REG);

LEDR(8DOWNTO0)<= LEDR_REG(8DOWNTO0);
END BEHAVIOR;

```

**Figura No. 3.10.5: Programa principal.**

```

DEPTH =128;
WIDTH=9;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN
% THIS CODE DISPLAYS A COUNT (INREGISTER R2)ON THE RED LEDS.%
00:001001000;% MVI R1,#1// INITIALIZE R %
01:000000001;
02:001010000;% MVI R2,#0// COUNTER TO DISPLAY ON LEDS %
03:000000000;
04:001011000;%LOOP MVI R3,#010000000// R3 = ADDRESS OF LEDS REGISTER%
05:010000000;
06:101010011;% ST R2,R3 //WRITETO LEDS %
07:010010001;% ADD R2,R1 // INCREMENT COUNTER FOR LEDS %
08:001011000;% MVI R3,#111111111// DELAY VALUE%
09:111111111;
0A:000101111;% MV R5,R7 // SAVE ADDRESS OFNEXT INST.%
0B:001100000;% OUTER MVI R4,#111111111// NESTED DELAY LOOP%
0C:111111111;
0D:000000111;% MV R0,R7 // SAVE ADDRESS OFNEXT INST.%
0E:011100001;% INNER SUB R4,R1 // DECREMENT LOOP DELAY VARIABLE%
0F:110111000;% MVNZ R7,R0 // CONTINUE INNER LOOPIF R4 !=0%
10:011011001;% SUB R3,R1 // DECREMENT OUTER LOOP DELAY %
11:110111101;% MVNZ R7,R5 // CONTINUE OUTER LOOPIF R3 !=0%
12:001111000;% MVI R7,#LOOP// EXECUTE AGAIN %
13:000000100;
END;

```

**Figura No. 3.10.6: Formato Archivo MIF.**

4. Utilice la simulación funcional para probar el circuito. Asegúrese de que los datos se lean correctamente de la memoria y sean ejecutados por el procesador.
5. Incluya en el proyecto la asignación de pines necesarios para poner en práctica el circuito en la tarjeta DE1-SoC. Utilice el interruptor SW(9) para controlar la entrada Run del

procesador, use KEY0 para Resetn, y utilice la señal de reloj de 50 MHz de la tarjeta como la entrada de reloj. Dado que el circuito necesita para funcionar correctamente a 50 MHz, asegúrese de que una restricción de tiempo se encuentra en Quartus II para restringir el reloj del circuito para esta frecuencia. Lea el informe producido por el analizador de tiempos Quartus II para asegurar que su circuito funciona a esta velocidad; si no, utilice las herramientas Quartus II para analizar el circuito y modificar el código VHDL para hacer un diseño más eficiente que cumpla con el requisito de velocidad de 50 MHz. También tenga en cuenta que la entrada Run es asíncrona a la señal de reloj, así que asegúrese de sincronizar esta entrada utilizando flip-flops.

6. Compile el circuito, descargue en el chip FPGA, y asegúrese de que el programa se ejecuta correctamente.

## 3.11.4 Asignación 2

### 3.11.4.1 Objetivo

En esta práctica se debe conectar un módulo de E/S adicional al circuito de la asignación 1 y escribir código que sea ejecutado por el procesador.

### 3.11.4.2 Desarrollo

Se debe añadir un módulo llamado seg7\_scroll al circuito el cual debe contener un registro para cada display de 7 segmentos de la tarjeta DE1-SoC. Cada registro debe operar directamente los segmentos del display, de modo que el procesador puede escribir caracteres sobre estos. Cree la decodificación de dirección necesaria para permitir que el procesador pueda escribir a los registros en el módulo seg7\_scroll.

### 3.11.4.3 Procedimiento

Para realizar esta práctica se deben seguir estos pasos:

1. Cree un proyecto Quartus II para el circuito y escriba un código VHDL que incluya el circuito de la figura 3.10.3, además de su módulo seg7\_scroll.
2. Utilice simulación funcional para probar el circuito.
3. Añada las limitaciones de tiempo apropiadas y la asignación de pines para el proyecto, y escriba un archivo MIF que permita al procesador escribir caracteres en los displays de 7 segmentos.
4. Pruebe la funcionalidad de su diseño mediante la ejecución de código de la memoria y la observación de los displays de 7 segmentos.



# Capítulo 4: FPGA para control de procesos.

---

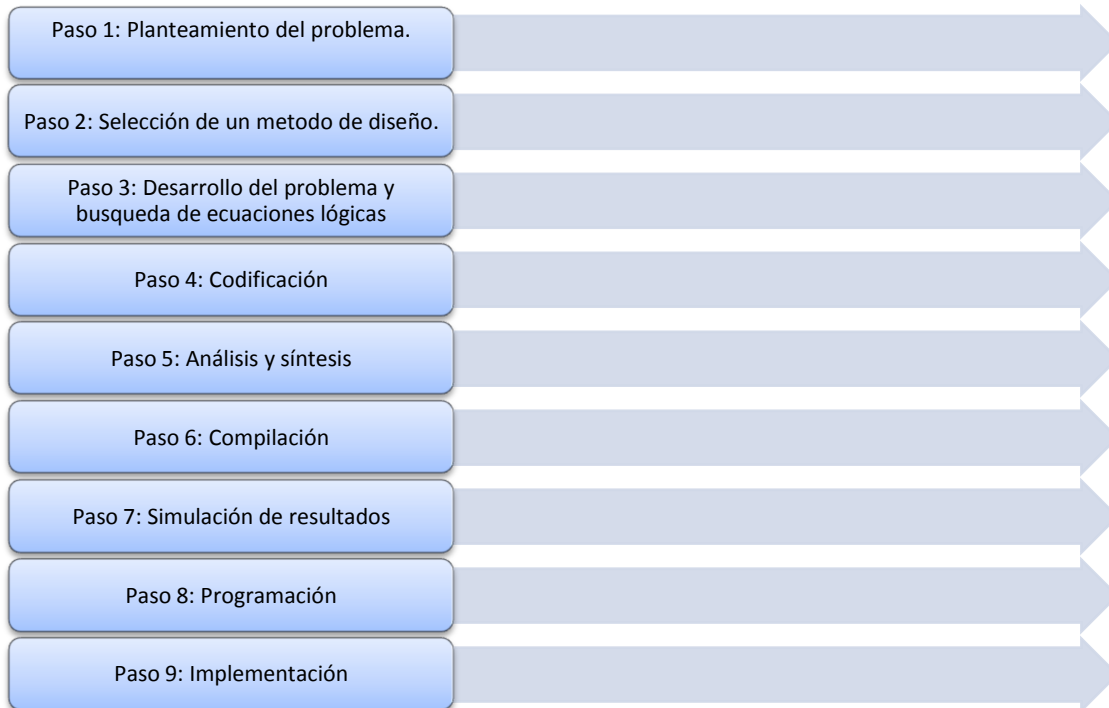
## 4.1 Introducción

Dadas las capacidades y ventajas que ofrece el FPGA es válido considerar esta tecnología para el control y/o automatización de procesos industriales, la programación concurrente es un factor determinante para esta consideración, ya que la mayoría de procesos automáticos requieren una simultaneidad de ejecución de eventos.

Utilizando lo aprendido con los laboratorios de la sección anterior se aplicarán diferentes métodos de diseño con los cuales se realizarán ejemplos didácticos de procesos de automatización que pueden ser ejecutados en la tarjeta DE1-SoC, también se introducirá el uso de los pines de propósito general para controlar un modulo externo de relés.

## 4.2 Metodología para la búsqueda de soluciones

Para generalizar la metodología a seguir se estimó un diagrama de flujo en donde se sugiere una serie de pasos que pueden facilitar la solución de un problema, este muestra en la figura 4.1:



**Figura No. 4.1: Metodología para el desarrollo de soluciones.**

- Planteamiento del problema:

Una buena redacción del problema, conocer todas las variables involucradas y delimitar los alcances facilitará una solución más efectiva.

- Selección de un método de diseño:

En relación a la cantidad de variables involucradas y la complejidad del sistema se debe seleccionar un método de diseño adecuado.

Mapas K, Quine McCluskey, FSM, Método Razonado, GRAFCET, son algunas alternativas disponibles para el usuario

- Desarrollo del problema y búsqueda de ecuaciones lógicas:

A través del método de diseño se deben obtener las ecuaciones lógicas que satisfagan la solución para el problema propuesto

- Codificación:

Una vez obtenidas las ecuaciones lógicas se procede a su codificación en VHDL o Verilog, además se agregan los módulos extra como controles de tiempo o de selección (si es que se necesitaran).

- Análisis y síntesis:

Esta sección está a cargo del software, únicamente será superable cuando el código no contenga errores, ni inconsistencias, en caso de errores se debe volver al paso 4.

- Compilación:

En este paso se crean los archivos de salida, estos archivos son creados automáticamente por el software luego que supera el paso 5.

- Simulación de resultados:

A través de diagramas de tiempo es posible ver el comportamiento del circuito creado, los archivos de creación son altamente fiables, son necesarios para la comprobación de resultados, en caso de tener respuestas incorrectas dentro del circuito se debe volver al paso 3.

- Programación

En esta etapa se descargan los archivos SOF al FPGA, de ser necesario la programación permanente hay que preparar el archivo JIC de la forma descrita en la sección 2.5.2.

- Implementación

Es la etapa final en donde se prueba el funcionamiento del FPGA junto al equipo requerido que satisfaga el sistema.

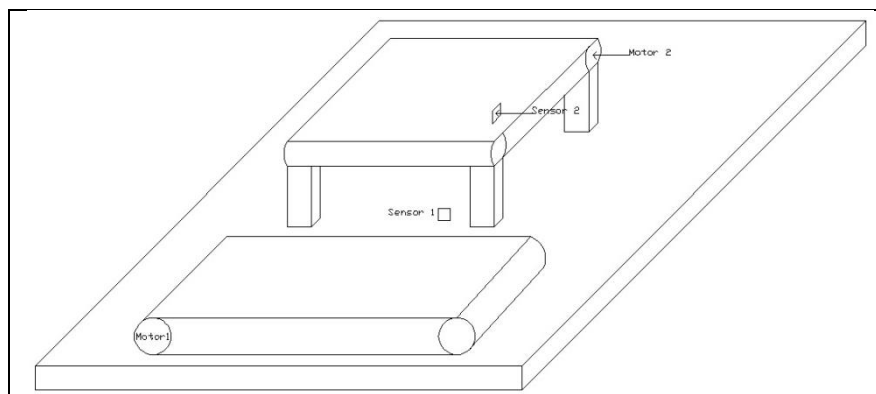
## 4.3 Banda empacadora de frutas

### 4.3.1 Definición del problema

Se necesita crear un sistema automático que sea capaz de llenar cajas con un determinado número de frutas, para este fin se disponen dos bandas transportadoras, la primera banda se activará al pulsar un botón de marcha, la banda desplazará una caja vacía hasta alcanzar un sensor fotoeléctrico el cual detendrá la caja en la posición de llenado, luego se activará la banda que transporta las frutas, antes de que estas caigan dentro de la caja activan el segundo sensor, cada caja debe ser llenada con 12 piezas de fruta, cuando esto sucede la banda que transporta las frutas se detiene y activa la cinta que transporta las cajas para retirar la caja llena y poder situar una nueva caja vacía en la posición de llenado.

El sistema cuenta con un botón de paro el cual detiene completamente el sistema, también se debe programar un temporizador de 1 segundo que le permita a la cinta dejar caer la fruta 12.

La figura 4.1 muestra un diagrama general del sistema:



**Figura No. 4.2: Diagrama general.**

### 4.3.2 Solución

El problema se resolverá por método razonado, tomando en cuenta las siguientes consideraciones:

- El contador módulo 12 será programado de forma interna en el FPGA, la señal de reloj será alimentada por el sensor de frutas, a su vez debe proveer una salida activa alta, que servirá como enable para el temporizador cuyo sistema también será programado dentro de la FPGA.
- Los pulsadores KEY0 y KEY1 se utilizarán como botón de marcha y de paro, respectivamente, estos elementos están incluido en la tarjeta DE1-SoC.
- Para la lectura de las señales se utilizarán los puertos GPIO de la tarjeta, de igual manera las señales de mando serán enviadas a través de estos puerto y acopladas por medio de un módulo de relés.

- Para mantener el ciclo de trabajo será necesario añadir una variable X que funcione como memoria del sistema, esta variable no constará como una señal de mando.
- Es importante mencionar que las salidas de los pines de propósito general por defecto es de 3.3 voltios, por lo cual se deben hacer las consideraciones pertinentes.

Continuando con el método razonado, se procede a considerar las variables de entrada y salida así como las condiciones creadoras y anuladoras del sistema:

Entradas	M(marcha), S1(Sensor cajas), S2(sensor frutas)				
Salidas	X	B1 (motor banda 1)	B2 (Motor banda 2)	T1 (temporizador)	Co (contador)
Creadoras	$m\bar{p}, X$	$X\bar{B}_2, B_1$	$S_1, B_2$	$C_0$	$B_2$
Anuladoras	P	$S_1, P$	$T_1, P$	$B_1, P$	$T_1, P$

Tabla No. 4.1: Variables de proceso

Resultado de las ecuaciones:

$$X = \bar{P}(m + x)$$

Ecuacion No. 4.1: Variable para memoria de proceso

$$B1 = \bar{S}_1 * \bar{P} (X\bar{B}_2 + B_1)$$

Ecuacion No. 4.2: Variable de control para banda de cajas

$$B2 = \bar{T}_1 * \bar{P} (S_1 + B_2)$$

Ecuacion No. 4.3: Variable de control para banda de frutas

Las señales creadoras del temporizador y contador constituyen la señal de habilitación del circuito, mientras que las señales anuladoras corresponden al reset del modulo.

### 4.3.3 Codificación para FPGA

Una vez se tienen las ecuaciones se procede a codificar en VHDL o Verilog, eso queda a discreción del usuario.

Para programar el núcleo del sistema se tiene el código mostrado en la figura 4.2:

```

LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY PROBLEMA1 IS
PORT (CLOCK_50 : IN STD_LOGIC;
      LEDR      : OUT STD_LOGIC_VECTOR (9 DOWNTO 0) ;
      KEY       : IN STD_LOGIC_VECTOR (1 DOWNTO 0) ;
      SW        : IN STD_LOGIC_VECTOR (9 DOWNTO 8) ;
      HEX0, HEX1 : OUT STD_LOGIC_VECTOR (0 TO 6) ;
      GPIO_0    : OUT STD_LOGIC_VECTOR (3 DOWNTO 2) ;

```

```

        GPIO_0 :OUTSTD_LOGIC_VECTOR(1DOWNTO0));
END PROBLEMA1;

ARCHITECTURE BEHAVIOR OF PROBLEMA1 IS
COMPONENT COUNTER
GENERIC ( N :NATURAL:=4;
          M :NATURAL:=16);
PORT ( CLOCK :INSTD_LOGIC;
       RESET_N, ENABLE :INSTD_LOGIC;
       Q :OUTSTD_LOGIC_VECTOR((N -1)DOWNTO0);
       ROLLOVER :OUTSTD_LOGIC);
ENDCOMPONENT;

COMPONENT HEXADEC
PORT (ENT :INSTD_LOGIC_VECTOR(3DOWNTO0);
      HEXA :OUTSTD_LOGIC_VECTOR(0TO6));
ENDCOMPONENT;

COMPONENT HEXAUNI
PORT (ENT :INSTD_LOGIC_VECTOR(3DOWNTO0);
      HEXA :OUTSTD_LOGIC_VECTOR(0TO6));
ENDCOMPONENT;

COMPONENT TEMPO
GENERIC ( N :NATURAL:=4;
          M :NATURAL:=16);
PORT ( CLOCK :INSTD_LOGIC;
       RESET_N, ENABLE :INSTD_LOGIC;
       ROLLOVER :OUTSTD_LOGIC);
ENDCOMPONENT;

COMPONENT EDGEN
PORT (SENSOR, BANDA :INSTD_LOGIC;
      SALIDA :OUTSTD_LOGIC);
ENDCOMPONENT;

SIGNAL MA, J, T1, S1, S2, B1, B2, CO, F1, BANDA1, BANDA2:STD_LOGIC:= '0';
SIGNAL P :STD_LOGIC;
SIGNAL SQ :STD_LOGIC_VECTOR(3DOWNTO0);
BEGIN
MA <=NOT(KEY(0));
S1 <= GPIO_0(2);
S2 <= GPIO_0(3);
P <= KEY(1);

GPIO_0(0)<=NOT(B1);
GPIO_0(1)<=NOT(B2);

LEDR(9)<=NOT(B1);
LEDR(8)<=NOT(B2);
LEDR(7)<= CO;
LEDR(6)<= T1;
LEDR(5)<= F1;
LEDR(4)<= J;

SEN: EDGEN PORTMAP(S1, B2, F1);
J <=(NOT(P))AND(MA OR J);

PROCESS ( J, B1, F1, P, B2)
BEGIN
IF(J = '0') THEN
    B1 <= '0';
ELSE

```

```

    B1 <= ((NOT(F1))AND(NOT(P)))AND((J AND(NOT(B2)))OR B1);
ENDIF;
ENDPROCESS;

PROCESS ( J, B1, F1, P, B2, T1)
BEGIN
IF(J = '0')THEN
    B2 <= '0';
ELSE
    B2 <= ((NOT(T1))AND(NOT(P)))AND(F1 OR B2);
ENDIF;
ENDPROCESS;

C1: COUNTER GENERICMAP(N =>4, M =>12) PORTMAP(S2, (T1 OR P), B2, SQ, CO);
N0: TEMPO GENERICMAP(N =>26, M =>50000000) PORTMAP(CLOCK_50, (B1 OR P), CO, T1);
DISP1: HEXAUNI PORTMAP(SQ, HEX0);
DISP2: HEXADEC PORTMAP(SQ, HEX1);

LEDR(3DOWNTO0)<="0000";
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.STD_LOGIC_ARITH.ALL;
USEIEEE.STD_LOGIC_SIGNED.ALL;

ENTITY COUNTER IS
GENERIC (
    N :NATURAL:=4;
    M :NATURAL:=16);
PORT (
    CLOCK :INSTD_LOGIC;
    RESET_N, ENABLE :INSTD_LOGIC;
    Q :OUTSTD_LOGIC_VECTOR((N -1)DOWNTO0);
    ROLLOVER :OUTSTD_LOGIC);
END COUNTER;

ARCHITECTURE BEHAVIOR OF COUNTER IS
SIGNAL VALUES :STD_LOGIC_VECTOR((N -1)DOWNTO0);
SIGNAL ROLL :STD_LOGIC;
BEGIN
PROCESS(CLOCK, RESET_N, ENABLE)
BEGIN
IF(RESET_N = '1')THEN
    VALUES <=(OTHERS=> '0');
    ROLL <= '0';
ELSIF((CLOCK'EVENT)AND(CLOCK = '1')AND(ENABLE = '1'))THEN
IF(VALUES =(M -1))THEN
    VALUES <=(OTHERS=> '0');
    ROLL <= '1';
ELSE
    VALUES <= VALUES +1;
    ROLL <= '0';
ENDIF;
ENDIF;
ENDPROCESS;
ROLLOVER <= ROLL;
Q <= VALUES;
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.STD_LOGIC_ARITH.ALL;
USEIEEE.STD_LOGIC_SIGNED.ALL;

```

```

ENTITY TEMPO IS
GENERIC ( N :NATURAL:=4;
          M :NATURAL:=16);
PORT( CLOCK :INSTD_LOGIC;
      RESET_N, ENABLE :INSTD_LOGIC;
      ROLLOVER :OUTSTD_LOGIC);
END TEMPO;

ARCHITECTURE BEHAVIOR OF TEMPO IS
SIGNAL VALUES :STD_LOGIC_VECTOR((N -1)DOWNT00);
SIGNAL ROLL :STD_LOGIC;
BEGIN
PROCESS(CLOCK, RESET_N, ENABLE)
BEGIN
IF(RESET_N = '1')THEN
VALUES <=(OTHERS=> '0');
ROLL <= '0';
ELSIF((CLOCK'EVENT)AND(CLOCK = '1' )AND(ENABLE = '1'))THEN
IF(VALUES =(M -1))THEN
VALUES <=(OTHERS=> '0');
ROLL <= '1';
ELSE
VALUES <= VALUES +1;
ROLL <= '0';
ENDIF;
ENDIF;
ENDPROCESS;
ROLLOVER <= ROLL;
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY EDGEN IS
PORT(SENSOR, BANDA :INSTD_LOGIC;
      SALIDA :OUTSTD_LOGIC);
END EDGEN;

ARCHITECTURE BEHAVIOR OF EDGEN IS
SIGNAL SO :STD_LOGIC;
BEGIN
PROCESS(SENSOR, BANDA)
BEGIN
IF(BANDA = '1')THEN
SO <= '0';
ELSIF((SENSOR'EVENT)AND(SENSOR = '0'))THEN
SO <= '1';
ENDIF;
ENDPROCESS;
SALIDA <= SO;
END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY HEXAUNI IS
PORT(ENT :INSTD_LOGIC_VECTOR(3DOWNT00);
      HEXA :OUTSTD_LOGIC_VECTOR(0TO6));
END HEXAUNI;

ARCHITECTURE BEHAVIOR OF HEXAUNI IS
BEGIN

```

```

PROCESS (ENT)
BEGIN
CASE ENT IS
WHEN"0000"=> HEXA <="0000001";
WHEN"0001"=> HEXA <="1001111";
WHEN"0010"=> HEXA <="0010010";
WHEN"0011"=> HEXA <="0000110";
WHEN"0100"=> HEXA <="1001100";
WHEN"0101"=> HEXA <="0100100";
WHEN"0110"=> HEXA <="0100000";
WHEN"0111"=> HEXA <="0001111";
WHEN"1000"=> HEXA <="0000000";
WHEN"1001"=> HEXA <="0001100";
WHEN"1010"=> HEXA <="0000001";
WHEN"1011"=> HEXA <="1001111";
WHEN"1100"=> HEXA <="0010010";
WHEN"1101"=> HEXA <="0000110";
WHEN"1110"=> HEXA <="1001100";
WHENOTHERS=> HEXA <="0100100";
ENDCASE;
ENDPROCESS;

END BEHAVIOR;

LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY HEXADEC IS
PORT (ENT :INSTD_LOGIC_VECTOR(3DOWNTO0);
      HEXA :OUTSTD_LOGIC_VECTOR(0TO6));
END HEXADEC;

ARCHITECTURE BEHAVIOR OF HEXADEC IS
BEGIN

PROCESS (ENT)
BEGIN
CASE ENT IS
WHEN"0000"=> HEXA <="0000001";
WHEN"0001"=> HEXA <="0000001";
WHEN"0010"=> HEXA <="0000001";
WHEN"0011"=> HEXA <="0000001";
WHEN"0100"=> HEXA <="0000001";
WHEN"0101"=> HEXA <="0000001";
WHEN"0110"=> HEXA <="0000001";
WHEN"0111"=> HEXA <="0000001";
WHEN"1000"=> HEXA <="0000001";
WHEN"1001"=> HEXA <="0000001";
WHEN"1010"=> HEXA <="1001111";
WHEN"1011"=> HEXA <="1001111";
WHEN"1100"=> HEXA <="1001111";
WHEN"1101"=> HEXA <="1001111";
WHEN"1110"=> HEXA <="1001111";
WHENOTHERS=> HEXA <="1001111";
ENDCASE;
ENDPROCESS;
END BEHAVIOR;

```

**Figura No. 4.3: Código principal para empacadora de frutas.**

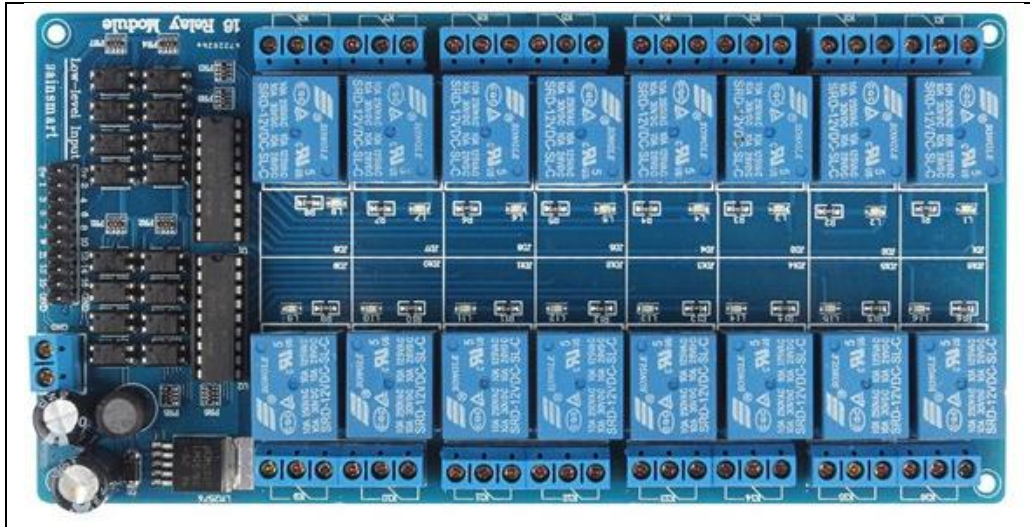


### 4.3.4 Circuitería extra

Para resolver el problema será necesario crear dos circuitos extra, uno que se encargue del control de motores y las barreras lumínicas, a continuación se muestran los diagramas correspondientes para cada uno de los circuitos.

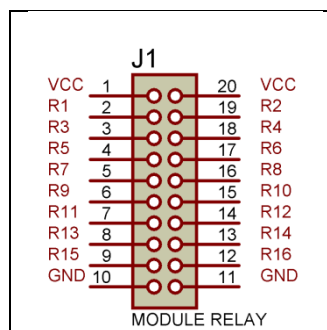
#### 4.3.4.1 Circuito de fuerza

Este circuito consta de dos motores controlado a través de un modulo de relés, la figura 4.3 muestra el dispositivo utilizado.



**Figura No. 4.4: Modulo de relés para implementación de circuito de fuerza**

Este dispositivo se alimenta con 5 voltios que pueden ser suplidos por la tarjeta FPGA, cuenta con opto acopladores que permiten tener la carga totalmente separada de la tarjeta, dando seguridad al equipo, cada uno de los relés es activo bajo, su activación es independiente, en la figura 4.4 se muestra un pinout general de este dispositivo.

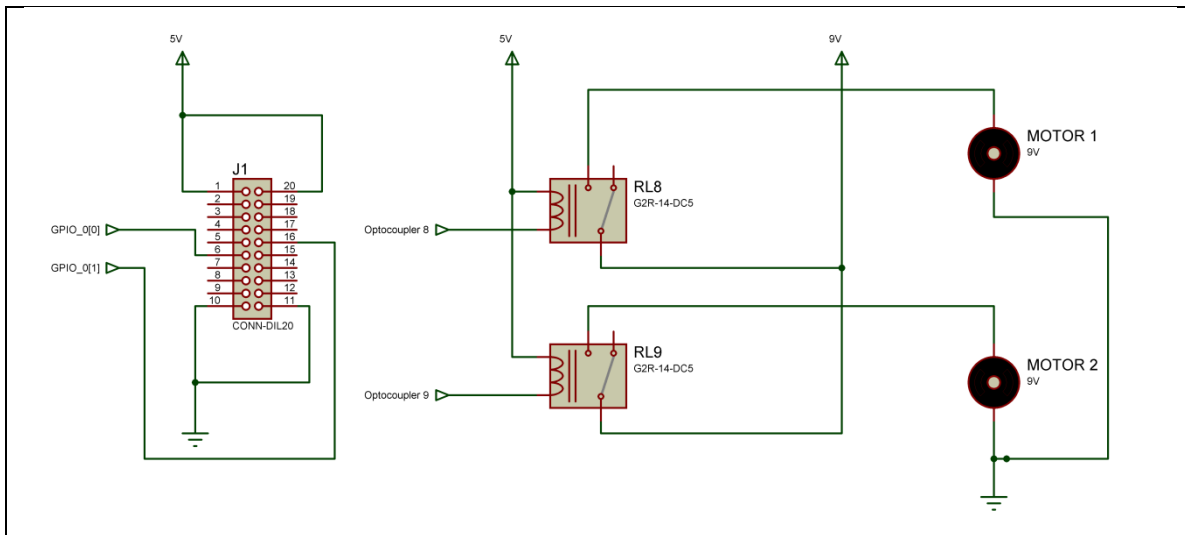


**Figura No. 4.5: Pinout de módulo de relés.**

Cada relé cuenta con salidas independientes, de forma que cada uno posee un pin común, contacto normalmente abierto y contacto normalmente cerrado.

Una consideración a tomar es que un pulso de 3.3 V proveniente de la FPGA será asimilado en forma de 0 por el módulo, lo que conlleva a poner un convertidor de voltaje entre los pines de pulso y las entradas a las bobinas.

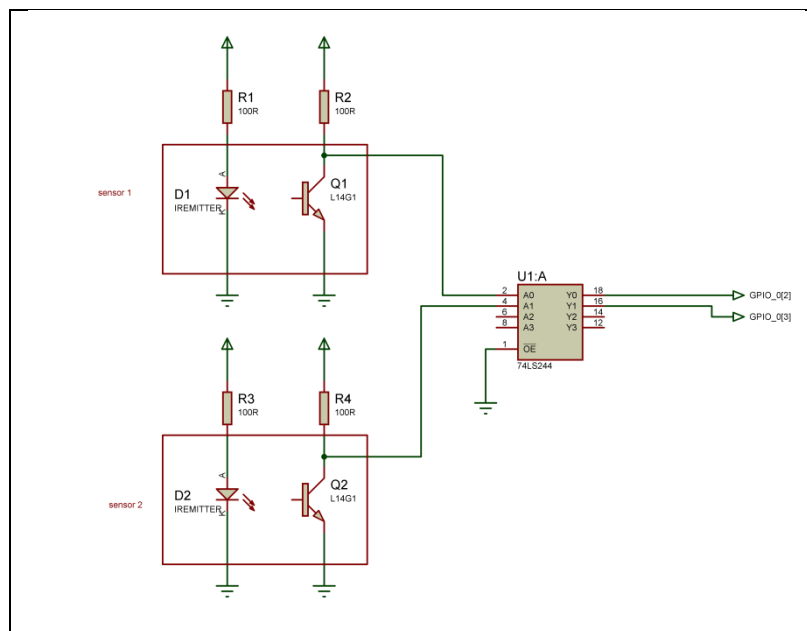
La figura 4.5 muestra de forma simplificada el circuito de fuerza:



**Figura No. 4.6: Circuito de fuerza**

#### 4.3.4.2 Sensores

En la figura 4.6 se muestra el circuito simplificado de los sensores lumínicos:



**Figura No. 4.7: Circuito para barreras lumínicas.**

Este circuito consta de una barrera lumínica formada por un led y un foto transistor L14G1, en las primeras pruebas se detectó que los diodos infrarrojos brindaban poca distancia, lo que provocaba que el circuito funcionara mal, así que los infrarrojos fueron sustituidos por punteros laser los cuales brindaban una distancia de hasta 5 metros.

Su funcionamiento es básico, cuando el haz de luz es interrumpido el transistor enviará un pulso equivalente a un "1" lógico, el chip 74LS244 fue agregado para minimizar los rebotes o los pulsos indeterminados, es una parte importante ya que uno de estos sensores se utiliza como señal de reloj para un contador, tener señales parasitas provocaría el mal funcionamiento del sistema.

# Conclusiones

---

- Los dispositivos FPGA han demostrado tener una capacidad muy amplia para cumplir funciones de alto nivel convirtiéndolos en dispositivos muy atractivos para implementar proyectos relacionados con la electrónica programable.
- La versatilidad que poseen de poder probar un diseño antes de ser grabado de forma permanente, permite al usuario experimentar diferentes soluciones o alternativas para un proyecto.
- A nivel didáctico resultan dispositivos amigables ya que los lenguajes de programación tienen una estructura definida, el banco de palabras reservadas para estos lenguajes es totalmente funcional y satisface todas las necesidades que el usuario pueda tener.
- El hecho de poder incluir programación secuencial junto a la programación concurrente hace que las FPGA sean dispositivos aplicables a los procesos de control y automatismo.
- En general para el ejercicio aplicativo solo se utilizaron técnicas y palabras reservadas que se habían descrito con anterioridad en los laboratorios.
- La compatibilidad con múltiples voltajes (5v, 3.3v y 2.5V) convierte al FPGA en un dispositivo versátil y adaptable a cualquier módulo programable, incluso a aquellos originalmente destinados a otras tarjetas o plataformas.
- El utilizar lenguajes de programación de uso libre provee al usuario la libertad de migrar a otra compañía fabricante de FPGA, si bien el software es privativo con una versión gratuita es importante mantener la independencia en la cadena de conocimientos.

# Glosario

---

- HDL: Hardware Description Language
- FPGA: Field Programmable Gate Array
- ASIC: Application Specific Integrated Circuit
- CLB: Configurable Logic Block
- LAB: Logic Array Block
- PAL: Programmable Array Logic
- PLD: Programmable Logic Device
- HPS: Hard Processor System
- SoC: System on Chip
- DSP: Digital Signal Processor
- VHDL: Very High Speed Hardware Description Language
- SDR: Software Defined Radio
- IOB: Input Output Block
- ALM: Adaptative Logic Module
- LE: Logic Element
- LUT: Look Up Table
- EDA: Electronic Design Automation
- RTL: Register Transfer Level
- RAM: Random Access Memory
- ALTERA: Compañia fabricante de FPGA.
- Xilinx: Compañía fabricante de FPGA.
- Quartus II: Software dedicado para el diseño y programación de circuitos en FPGA.

# Bibliografía

---

- [https://es.wikipedia.org/wiki/Field\\_Programmable\\_Gate\\_Array](https://es.wikipedia.org/wiki/Field_Programmable_Gate_Array) [en línea] [última consulta 24/02/2016]
- <http://www.ni.com/fpga/esa/> [en línea] [última consulta 24/02/2016]
- <http://www.ni.com/white-paper/6984/es/> [en línea] [última consulta 24/02/2016]
- <http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm> [en línea] [última consulta 24/02/2016]
- <http://www.xilinx.com/training/free-video-courses.htm#ASIC> [en línea] [última consulta 24/02/2016]
- <https://www.altera.com/products/fpga/cyclone-series/cyclone-v/overview.html> [en línea] [última consulta 24/02/2016]
- <https://www.altera.com/support/literature/wp/lit-cyclone-v-white-papers.html> [en línea] [última consulta 24/02/2016]
- [https://en.wikipedia.org/wiki/Logic\\_block](https://en.wikipedia.org/wiki/Logic_block)
- [http://zone.ni.com/reference/enXX/help/371599G01/lvfpgaconcepts/fpga\\_basic\\_chip\\_terms/](http://zone.ni.com/reference/enXX/help/371599G01/lvfpgaconcepts/fpga_basic_chip_terms/)
- ALTERA, Logic Array Blocks and Adaptive Logic Modules in Stratix IV Devices, Febrero 2011, [en línea] [última consulta 24/02/2016 ] [https://www.altera.com/content/dam/altera-www/global/en\\_US/pdfs/literature/hb/stratix-iv/stx4\\_siv51002.pdf](https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stratix-iv/stx4_siv51002.pdf)[1]
- [https://es.wikipedia.org/wiki/Procesador\\_digital\\_de\\_se%C3%B1ales](https://es.wikipedia.org/wiki/Procesador_digital_de_se%C3%B1ales)[en línea] [última consulta 24/02/2016].
- [https://es.wikipedia.org/wiki/Circuito\\_integrado\\_de\\_aplicaci%C3%B3n\\_espec%C3%ADfica](https://es.wikipedia.org/wiki/Circuito_integrado_de_aplicaci%C3%B3n_espec%C3%ADfica) [en línea] [última consulta 24/02/2016].
- <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836>[en línea] [última consulta 24/02/2016].
- [https://es.wikipedia.org/wiki/Programmable\\_Array\\_Logic](https://es.wikipedia.org/wiki/Programmable_Array_Logic)[en línea] [última consulta 24/02/2016].
- [n.s], Dispositivos Lógicos Programables [http://www.uhu.es/rafael.lopezahumada/Cursos\\_anteriores/fund97\\_98/plds.pdf](http://www.uhu.es/rafael.lopezahumada/Cursos_anteriores/fund97_98/plds.pdf) [en línea] [última consulta 24/02/2016].
- ALTERA, Using Library Modules in VHDL Designs, Agosto 2011, [ftp://ftp.altera.com/up/pub/Altera\\_Material/11.1/Tutorials/VHDL/Using\\_Library\\_Modules.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Tutorials/VHDL/Using_Library_Modules.pdf) [en línea] [última consulta 24/02/2016].
- ALTERA, Quartus II Introduction Using VHDL Designs, Mayo 2012 [ftp://ftp.altera.com/up/pub/Altera\\_Material/12.0/Tutorials/VHDL/Quartus\\_II\\_Introduction.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/12.0/Tutorials/VHDL/Quartus_II_Introduction.pdf) [en línea] [última consulta 24/02/2016]
- [ftp://ftp.altera.com/up/pub/Altera\\_Material/13.1/Boards/DE1-SoC/DE1\\_SoC.qsf](ftp://ftp.altera.com/up/pub/Altera_Material/13.1/Boards/DE1-SoC/DE1_SoC.qsf)[en línea] [última consulta 24/02/2016].

- <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=836&PartNo=4>[en línea] [última consulta 24/02/2016].
- <https://www.altera.com/downloads/download-center.html>[en línea] [última consulta 24/02/2016].
- <http://wl.altera.com/education/univ/materials/boards/de1-soc/unv-de1-soc-board.html> [en línea] [última consulta 24/02/2016].
- <http://wl.altera.com/education/training/courses/OHDL1110>[en línea] [última consulta 24/02/2016].
- \*\*Todos los laboratorios fueron traducidos e interpretados a partir de los documentos originales que provee el sitio de ALTERA University Program, los cuales son una propuesta de aprendizaje para el uso de los kit de desarrollo, las imágenes de los laboratorios originales han sido utilizadas en la traducción, el tamaño y la disposición de las misma a cambiado a favor de mantener el formato establecido para este documento, los documentos originales pueden ser consultados en el sitio:  
<https://www.altera.com/support/training/university/materials-lab-exercises.html>

## Referencias

---

- ALTERA, Logic Array Blocks and Adaptive Logic Modules in Stratix IV Devices, Febrero 2011, [https://www.altera.com/content/dam/altera-www/global/en\\_US/pdfs/literature/hb/stratix-iv/stx4\\_siv51002.pdf](https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stratix-iv/stx4_siv51002.pdf)[en línea] [última consulta 24/02/2016 ] [1]
- ALTERA, Logic Elements and Logic Array Blocks in Cyclone IV Devices, Noviembre 2009. [https://www.altera.com/en\\_US/pdfs/literature/hb/cyclone-iv/cyiv-51002.pdf](https://www.altera.com/en_US/pdfs/literature/hb/cyclone-iv/cyiv-51002.pdf) [en línea] [última consulta 24/02/2016] [2].
- ALTERA, FPGA Architecture, White Paper, Julio 2006. [https://www.altera.com/en\\_US/pdfs/literature/wp/wp-01003.pdf](https://www.altera.com/en_US/pdfs/literature/wp/wp-01003.pdf) [en línea] [última consulta 24/02/2016] [3].
- <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=836&PartNo=2>[en línea] [última consulta 24/02/2016]. [4]
- Altera, DE1-SoC User Manual, 2013 [5]
- ALTERA, Cyclone V Device Overview, Diciembre 2015. [https://www.altera.com/en\\_US/pdfs/literature/hb/cyclone-v/cv\\_51001.pdf](https://www.altera.com/en_US/pdfs/literature/hb/cyclone-v/cv_51001.pdf) [en línea] [última consulta 24/02/2016] [6]
- ALTERA, Laboratory Exercise 1 Switches, Lights, and Multiplexers [ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab1\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab1_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [7]

- ALTERA, Laboratory Exercise 2 Numbers and Displays  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab2\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab2_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [8]
- ALTERA, Laboratory Exercise 3 Latches, Flip-flops, and Registers  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab3\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab3_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [9]
- ALTERA, Laboratory Exercise 4 Counters  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab4\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab4_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [10]
- ALTERA, Laboratory Exercise 5 Timers and Real-time Clock  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab5\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab5_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [11]
- ALTERA, Laboratory Exercise 6 Adders, Subtractors, and Multipliers  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab6\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab6_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [12]
- ALTERA, Laboratory Exercise 7 Finite State Machines  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab7\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab7_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [13]
- ALTERA, Laboratory Exercise 8 Memory Blocks  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab8\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab8_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [14]
- ALTERA, Laboratory Exercise 9 A Simple Processor  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab9\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab9_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [15]
- ALTERA, Laboratory Exercise 10 An Enhanced Processor  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/Laboratory\\_Exercises/Digital\\_Logic/DE1-SoC/vhdl/lab10\\_VHDL.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab10_VHDL.pdf) [en línea] [ultima consulta 24/02/2016] [16]