DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE Y EL LENGUAJE DE DESCRIPCIÓN DE HARDWARE VHDL. ASIGNATURA: SISTEMAS DIGITALES.

HÉCTOR ALFREDO MEJÍA RONDANO
MIGUEL FERNANDO MUÑOZ PINILLA

Universidad Industrial de Santander
Facultad de Ingenierías Físico-Mecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2012

DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE Y EL LENGUAJE DE DESCRIPCIÓN DE HARDWARE VHDL. ASIGNATURA: SISTEMAS DIGITALES.

HÉCTOR ALFREDO MEJÍA RONDANO
MIGUEL FERNANDO MUÑOZ PINILLA

Trabajo de grado presentado como requerimiento parcial para optar al título de:

Ingeniero Electrónico

Tesis desarrollada con el grupo de investigación CPS

Director:
MSc. Jorge H. Ramón Suarez

Co-Director:
MSc. William A. Salamanca B.
Ing. Carlos A. Angulo J.
MSc. Carlos A. Fajardo A.
MSc. Sergio A. Abreo C.

Universidad Industrial de Santander
Facultad de Ingenierías Físico-Mecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2012

Agradecimientos

A nuestro director de proyecto por darnos la oportunidad de trabajar y aportar nuestros conocimientos en aspectos tan fundamentales como la formación de futuros profesionales,

A nuestros codirectores de proyecto por brindarnos el soporte, la experiencia y el apoyo que necesitamos durante la realización del presente proyecto,

A los estudiantes y compañeros de proyecto que voluntariamente colaboraron en la validación necesaria para el desarrollo del presente,

A la compañia Texas Instruments por facilitarnos los dispositivos necesarios,

A la Universidad Industrial de Santander por brindarnos las herramientas necesarias para tener una formación íntegra que nos permita colaborar en la sociedad que vivimos.

En primer medida quiero dar gracias a Dios por tener la salud y la energía para poder realizar éste trabajo, además por brindarme la oportunidad de tener una formación universitaria.

En segundo lugar quiero agradecer a mi familia que ha estado presente a lo largo de mi vida, prestando atención en mis dificultades e invirtiendo recursos en mi formación no sólo académica en los quiero resaltar A mis padres y a mis hermanas, sin ellos este trabajo nunca lo hubiese podido llevar a cabo.

Como tercera medida quiero agradecer a mis amigos y compañeros de la universidad, ya que gracias a un apoyo conjunto, he escalado hasta llegar al trabajo final de mi carrera.

Siendo lo último pero no lo menos importante, quiero agradecer a Jennifer Julieth Rada Rodriguez por apoyarme y compartir incondicionalmente con aspectos no que han llenado mi vida de felicidad.

Héctor Alfredo Mejía Rondano

Primero que todo agradezco a Dios por ser la guía en mi camino y darme la salud necesaria que me permite cumplir con mis objetivos.

Agradezco al profesor Gabriel Vargas por ayudarme a superar dificultades en un momento importante de esta etapa.

Agradezco a mi familia por brindarme todo su apoyo y ser incondicional en todo momento, en especial a mi hermana Francy Liliana Muñoz, a mi tía Claudia Fabiola Pinilla y a mi tía Alba Luz Pinilla quién siempre me brindó sus sabias palabras animándome a continuar adelante y culminar esta etapa de mi vida.

Agradezco a mi novia Silvia Natalia Serrano por apoyarme y acompañarme en cada momento que la necesité.

Agradezco a mi padre Juan Fernando Muñoz por sus consejos y palabras de aliento en momentos difíciles de la carrera.

A mi madre Esperanza Pinilla además de darle las gracias, la felicito por que éste logro le pertenece en su totalidad a ella más que a ninguna otra persona, ella siempre estuvo a mi lado apoyándome y es quien me da la fuerza necesaria para seguir progresando en mi vida.

Miguel Fernando Muñoz Pinilla

Tabla de Contenido

Lista de Figuras

Lista de Anexos

Resumen

TÍTULO: DESARROLLO DE MATERIAL DIDÁCTICO PARA EL ÁREA DE SISTEMAS DIGITALES BASADOS EN LA PLATAFORMA SIE Y EL LENGUAJE DE DESCRIPCIÓN DE HARDWARE VHDL ASIGNATURA: SISTEMAS DIGITALES.*

AUTORES: HÉCTOR ALFREDO MEJÍA RONDANO, MIGUEL FERNANDO MUÑOZ PINILLA **

PALABRAS CLAVE: Plataforma SIE, GPIO, FPGA.

El presente proyecto va dirigido principalmente a estudiantes que deseen iniciarse en el mundo de los sistemas digitales, proporcionando a los mismos un material didáctico que permita aplicar los conocimientos adquiridos en las clases teóricas.
Así pues, cabe mencionar que el material desarrollado consta de un considerable número de guías que fortalecen los conceptos aprendidos en clase, cubriendo la temática ofrecida en la asignatura de sistemas digitales, además de una tarjeta de expansión de pines que se adapta perfectamente a la tarjeta SIE, brindando la posibilidad de desarrollar cada una de las guías de laboratorio propuestas. De igual forma se facilita un manual de usuario de la tarjeta SIE, con el fin de brindar al alumno los conocimientos necesarios para el manejo y utilización de la misma.
Adicionalmente, con el objetivo de aplicar los conocimientos adquiridos en el transcurso de la asignatura, se plantea la realización de dos proyectos en donde los alumnos tienen la oportunidad de resolver problemas de la vida real, brindando las herramientas necesarias para proponer una solución que requiere conceptos tanto de sistemas digitales, como de circuitos electrónicos básicos, con la firme intención de construir un conocimiento integral, donde el alumno reconozca su formación profesional como instrumento de gran valor para servir a una sociedad que lo necesita. Finalmente se hace una validación del material desarrollado con el fin de realizar las mejoras correspondientes para hacer cómodo y útil a la comunidad estudiantil de la asignatura de sistemas digitales ó a quien esté interesado.

*Trabajo de grado

**Facultad: Facultad de Ingenierías Físico-Mecánicas. Escuela: Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones.Grupo de Investigación: CPS. Director: MSc. Jorge H. Ramón Suarez. Codirectores: MSc. William A. Salamanca B., Ing. Carlos A. Angulo J., MSc. Carlos A. Fajardo A., MSc. Sergio A. Abreo C.

Abstract

TITLE: DEVELOPMENT OF A DIDACTIC MATERIAL FOR THE DIGITAL SYSTEMS AREA BASED ON THE SIE PLATFORM AND VHDL. SUBJECT: DIGITAL SYSTEMS .*

AUTHORS: HÉCTOR ALFREDO MEJÍA RONDANO, MIGUEL FERNANDO MUÑOZ PINILLA **

KEY WORDS: SIE Board, GPIO, FPGA.

This project is mainly addressed to students who wish to begin exploring the digital systems world by providing them didactic material in which they will be able to apply the knowledge acquired through theoretical classes. Hence, it is important to point out that the present document consists of a considerable number of guides which strengthen concept covering the themes discussed in the digital systems course; furthermore, a pin expansion card, that perfectly fits to the SIE board, offering the opportunity of developing each proposed laboratory guide. Likewise, a user manual of the SIE board is provided with the purpose of offering to students the required knowledge in order to operate it.

In addition, for achieving the objective of applying the knowledge learned during the course, it is suggested to carry out two projects in which students have the chance of solving real life problems, providing tools for proposing a solution requiring digital systems and circuit electronic concepts; with the clear purpose of developing an integral understanding, students will be able to recognize their professional education as an instrument of a great value to serve the society that really need it. Finally, a validation of the proposed material is made in order to carry out the required improvements in such a way that the digital systems course can be comfortable and useful to the students.

*Degree project

**Faculty: Physico-mechanical Engineering Faculty. School: School of Electrical, Electronics and Telecommunications Engineering. Research group: CPS. Director: MSc. Jorge H. Ramón Suarez. Codirectors: MSc. William A. Salamanca B., Ing. Carlos A. Angulo J., MSc. Carlos A. Fajardo A., MSc. Sergio A. Abreo C.

INTRODUCCIÓN

El concepto de computadora digital se remonta a Charles Babbage, quien desarrolló un basto dispositivo de computación mecánico hacia 1830. La primera computadora digital funcional fue construida en 1944 en la Universidad de Harvard, pero era electromecánica, no electrónica. La electrónica digital moderna comenzó en 1946 con una computadora digital electrónica llamada ENIAC, que fue fabricada con válvulas de vacío. Aunque ocupaba una habitación entera, ENIAC no tenía ni siquiera la potencia que puede tener hoy en día una calculadora de bolsillo. [9]

Actualmente el mundo se encuentra colmado de sistemas digitales por doquier, los cuales han suplido innumerables necesidades mejorando cada vez la calidad de vida humana, encontrando grandes ventajas que se ven reflejadas en las comunicaciones, instrumentación, navegación, procesos industriales, aeronáutica, deportes, seguridad, investigación y electrónica de consumo entre muchos otros campos.

Es así como los centros educativos de formación profesional en su afán de ir de la mano con el acelerado cambio del mundo y con la intensión de hacer de los profesionales personas capaces de servir mejor a la sociedad, se interesa por brindar las herramientas necesarias para desarrollar en los estudiantes la capacidad de resolver problemas y suplir las necesidades que se presentan a diario con el pasar de los días y la evolución del mundo.

Así pues, en la Universidad Industrial de Santander se presenta la tarjeta SIE como un proyecto creado para satisfacer las necesidades de los desarrolladores de hardware permitiendo la creación de aplicaciones bajo la licencia Creative Commons BY - SA [3], que permite la distribución y modificación del diseño (incluso para aplicaciones comerciales), con el único requisito de que los productos derivados deben tener la misma licencia y deben dar crédito al autor del trabajo original. Lo que constituye la base de los productos *hardware copyleft*.[2]

Es así como en la asignatura de sistemas digitales, ha sido posible acoplar la tarjeta SIE, brindando a los alumnos una herramienta muy útil que ayuda de sobremanera en el proceso de aprendizaje de los mismos, desarrollando un material didáctico que les permita entender el funcionamiento de la tarjeta para una correcta manipulación, aplicar conocimientos teóricos a través de una serie de prácticas de laboratorio revisadas minuciosamente con el objetivo de brindar un material entendible, cómodo y eficiente que establezca bases lo suficientemente fuertes en el primer curso de la rama de sistemas digitales, para que todo aquel que desee seguir por la misma línea tenga los conocimientos necesarios para tomar cursos con un nivel más avanzado en el tema, además se tiene la posibilidad de complementar el proceso con la realización de dos proyectos finales que pongan a prueba las habilidades y los conocimientos adquiridos en el trascurso de la asignatura, dando solución a un problema de la vida real, con el fin de generar consciencia en los alumnos de la herramienta útil, importante y poderosa que tienen al aplicar los conceptos de sistemas digitales para el crecimiento y desarrollo de la sociedad que los rodea.

Finalmente, cabe mencionar que del resultado de este trabajo han surgido, una tarjeta de expansión de pines que permite perfectamente utilizar la tarjeta SIE en el desarrollo de aplicaciones que requieran la utilización del FPGA, además de diez prácticas de laboratorio basadas en la descripción de circuitos por medio del lenguaje VHDL, que complementan y aplican los conocimientos teóricos adquiridos en clase, y adicionalmente, dos proyectos finales de la asignatura de sistemas digitales, que concentran las experiencias adquiridas tanto en las clases teóricas como prácticas, donde los alumnos pueden aplicar sus conocimientos dando solución a un problema de la vida real, aplicando conceptos básicos de circuitos electrónicos y de trabajo ingenieril, con el ánimo de construir un conocimiento integral, que fortalezca las habilidades de los futuros profesionales en ingeniería.

1. ESPECIFICACIONES DEL PROYECTO

1.1 Objetivo general

Desarrollar material didáctico para el área de sistemas digitales basado en la plataforma SIE utilizando el lenguaje de descripción de hardware VHDL.

1.2 Objetivos especificos

- Elaborar un manual básico de manejo de la plataforma SIE orientando su aplicación a la asignatura Sistemas Digitales.
- Diseñar, implementar y validar siete prácticas de laboratorio para la asignatura Sistemas Digitales utilizando el lenguaje de descripción de hardware VHDL
- Diseñar, implementar y validar dos proyectos finales para la asignatura Sistemas Digitales utilizando el lenguaje de descripción de hardware VHDL

1.3 Alcances

Los alcances de este proyecto fueron:

- Explicar los conceptos generales y el buen uso de la tarjeta SIE, que se relaciona con el contenido de la asignatura de sistemas digitales, a través de la realización de una guía de fácil entendimiento.
- Proponer y realizar diversas prácticas donde se apliquen los conocimientos adquiridos en las clases teóricas de la asignatura de sistemas digitales.
- Proponer y desarrollar dos prácticas que relacionen y apliquen los conceptos teórico-prácticos adquiridos a lo largo de la asignatura de sistemas digitales como proyectos finales.

2. GENERALIDADES

2.1 Estado del arte

Para empezar a realizar este trabajo primero se realizó un estudio en las universidades más relevantes del mundo, investigación que fue de gran utilidad ya que debido a esta se logró establecer las temáticas en las cuales las universidades más influyentes centran la asignatura de sistemas digitales, realizando una comparación con la temática trabajada actualmente en la universidad Industrial de Santander en el área de sistemas digitales, encontrando que efectivamente se trabajan los conceptos fundamentales de la asignatura.

2.2 Plataforma SIE.

El proyecto SIE [10] fue creado para satisfacer las necesidades de los desarrolladores de hardware permitiendo la creación de aplicaciones comerciales bajo la licencia Creative Commons BY -SA [3] la que permite la distribución y modificación del diseño (incluso para aplicaciones comerciales), con el único requisito de que los productos derivados deben tener la misma licencia y deben dar crédito al autor del trabajo original. [2]
Así pues, la plataforma SIE es concebida por el profesor Carlos Camargo de la Universidad Nacional de Colombia y es una adaptación de Ben NanoNote a la cual se le ha añadido un FPGA y puertos de entrada salida, igualmente se ha extraído un teclado el cual libera un gran número de pines que pueden ser utilizados en muchas más aplicaciones. [8]

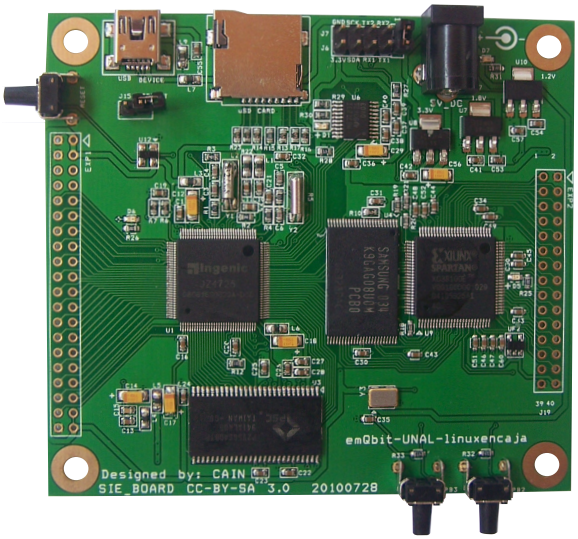


Figura 2.1: SIE

FUENTE: Los autores.

En la figura 2.1 se nota la plataforma SIE con los diversos componentes que la conforman, dichos componentes se mencionan a continuación: [8]

- Un procesador Ingenic JZ4725.
- 64MB de memoria SDRAM.
- Una memoria NAND de 2GB.
- FPGA XC3S100E_VQ100 que proporciona 25 puertos de entrada salida de propósito general con señales digitales (rango 0-3.3V).
- El Puerto USB puede ser usado como Ethernet, o dispositivo de consola serial.
- Puerto Micro SD.
- Líneas de entrada / salida de audio Estéreo.
- Señal de entrada de micrófono.
- Puerto I2C.
- Puerto RS-232 Serial UART.

Así mismo, en la figura 2.2 se muestra el respectivo diagrama de bloques característico de la plataforma SIE, notando en el mismo los componentes mencionados anteriormente, resaltando el FPGA y los periféricos externos debido a que son los principales componentes que se manejan en la asignatura de sistemas digitales.

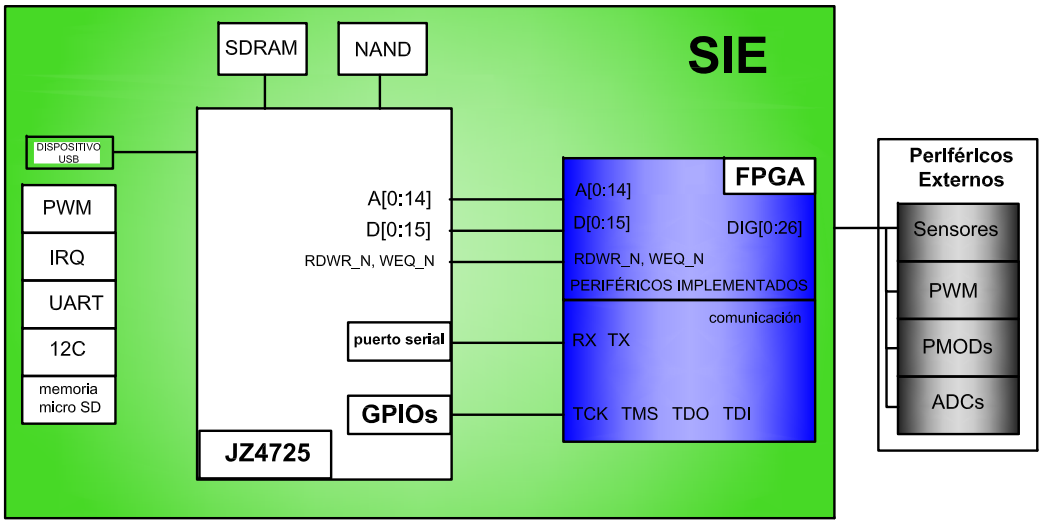


Figura 2.2: Diagrama de bloques de la SIE.

FUENTE: Los autores.

2.3 Linux

Linux es un sistema operativo gratuito y de libre distribución inspirado en el sistema Unix, escrito por Linus Torvalds con la ayuda de miles de programadores en internet.[1] Unix es un sistema operativo desarrollado en 1970, una de las mayores ventajas se debe a que es fácilmente portable a diferentes tipos de ordenadores, por lo que existen versiones de Unix para casi todos los tipos de ordenadores, desde PC y Mac hasta estaciones de trabajo y superordenadores. Al contrario que otros sistemas operativos, como por ejemplo MacOS (Sistema operativo de los Apple Macintosh), Unix no está pensado para ser fácil de emplear, sino para ser sumamente flexible. Por lo tanto Linux no es en general tan sencillo de emplear como otros sistemas operativos, aunque se están realizando grandes esfuerzos para facilitar su uso. Pese a toda la enorme flexibilidad de Linux y su gran estabilidad (y el bajo coste) han hecho de este sistema operativo

una opción muy interesante para aquellos usuarios que se dediquen a trabajar a través de redes, naveguen por internet, o se dediquen a la programación. Además el futuro de Linux es brillante y cada vez más gente y más empresas (entre otras IBM, Intel, Corel) están apoyando este proyecto, con lo que el sistema será cada vez más sencillo de emplear y los programas serán cada vez mejores. [1].

2.3.1. Acerca de la shell

Para la interacción entre un usuario y el sistema operativo es necesario algún elemento que interprete los mandos del usuario y se los comunique al sistema operativo, esperando una respuesta del sistema para mostrársela en pantalla nuevamente al usuario y terminar el proceso. Así pues, el intérprete de comandos se define como la interfaz entre un usuario y el sistema operativo, razón por la cual se le da el nombre de Shell que en inglés tiene el significado de caparazón, teniendo en su interior el sistema.
Entonces, la *shell* se ve como un intermediario entre el sistema operativo y el usuario gracias a las líneas de comando que introduce el usuario. El proceso que hace la shell consiste en:

- Interpretar los comandos
- Transmitir los comandos al sistema
- Arrojar un resultado

En la Figura 2.3, es posible notar el proceso interno que realiza la *shell* desde que se le ingresa por medio del teclado una orden, hasta que responde con una acción o resultado mostrado en la pantalla del ordenador donde se esté trabajando.
Por último cabe mencionar un concepto que puede ser de utilidad, el *prompt* de *Linux* hace referencia a la pantalla o terminal en donde la *shell* y el usuario leen los comandos que este último digita.

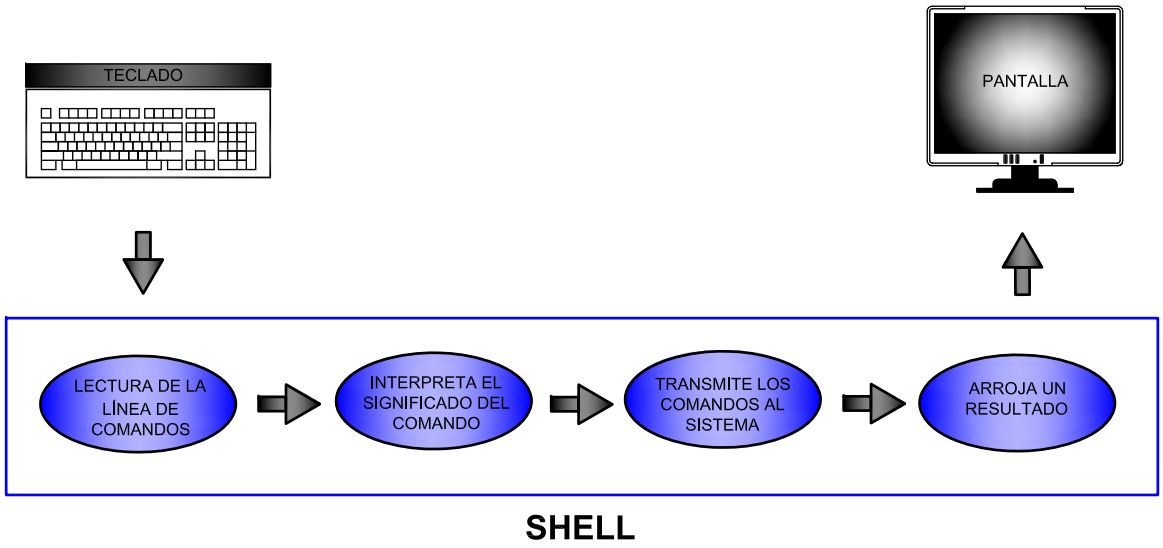


Figura 2.3: Proceso interno de la shell

FUENTE: Los autores.

2.4 Proceso realizado con la FPGA

La asignatura de sistemas digitales muestra aplicaciones que basan su interacción con la tarjeta SIE, en la implementación de diversos circuitos descritos por medio del lenguaje VHDL en la FPGA, los cuales pueden a su vez interactuar de acuerdo con su comportamiento por medio de los periféricos externos que se le conecten a la plataforma SIE.

La plataforma SIE proporciona un canal de comunicación entre el procesador y el puerto JTAG de la FPGA, este canal es utilizado para la configuración de la misma, enviando un archivo con la función de cada uno de los pines de la FPGA.

Para llevar a cabo el proceso de implementación de un circuito en la FPGA de la tarjeta SIE; lo primero es realizar la descripción del hardware a implementar, para el caso se utiliza VHDL; seguido de un proceso de síntesis realizado con la herramienta *ISE* de *Xilinx*; el resultado que se obtiene es un archivo de extensión *.bit*.

Posteriormente se envía este archivo a la SIE por medio del protocolo de comunicaciones *ssh*, asignando dirección IP tanto a la SIE como al PC que envía la información.

Cuando el archivo de configuración *(.bit)* ha sido transmitido, se ingresa al procesador

de la SIE y se utiliza la aplicación *xc3sprog* para configurar la FPGA.

La figura 2.4 muestra un esquema de los archivos necesarios para implementar un circuito en la FPGA, donde se muestran los correspondientes puertos de comunicación entre dispositivos que tienen como finalidad generar un archivo que contiene una información que debe llegar a la FPGA y así culminar el proceso de implementación, este proceso se explica detalladamente en las guías de laboratorio propuestas que se encuentran anexas al presente documento.

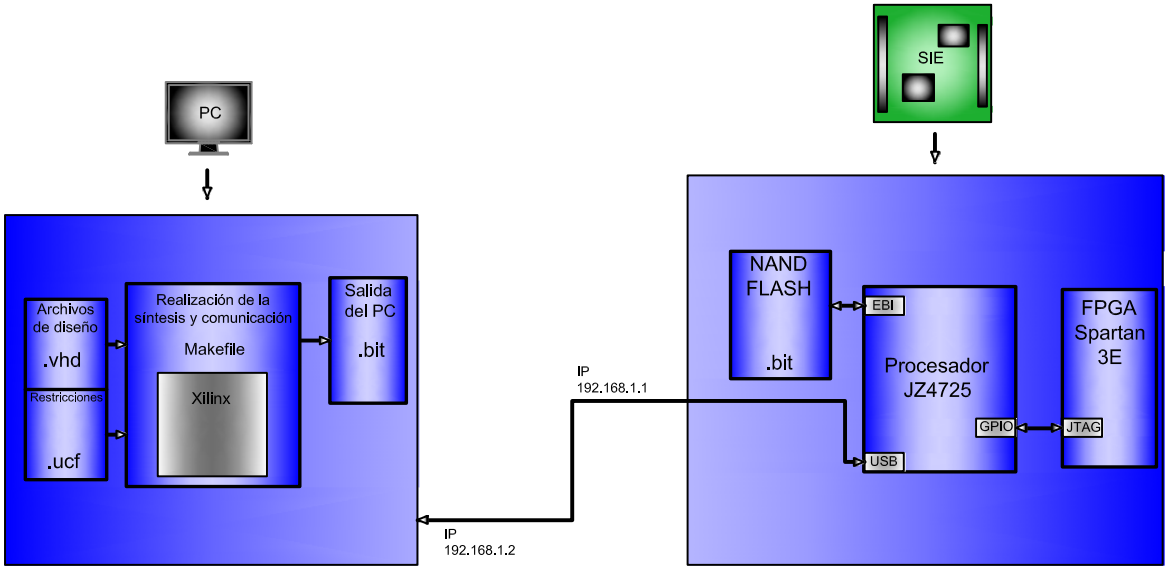


Figura 2.4: Esquema del proceso de configuración de la FPGA de la tarjeta SIE para implementar un circuito digital

FUENTE: Los autores.

3. METODOLOGÍA

3.1 Metodología del proyecto

Para llevar a cabo el cumplimiento de los objetivos del proyecto planteados, se trabaja siguiendo un plan que hace referencia a una serie de fases que describen el desarrollo del proyecto.

- FASE I: Estudio de Linux

  Etapa 1: Instalación de Linux

  Etapa 2: Aprender a utilizar los comandos relevantes de Linux para el proyecto

  Etapa 3: Aplicación de Linux a la plataforma SIE

- FASE II: Conocimiento de la SIE

  Etapa 1: Introducción a la tarjeta SIE

  Etapa 2: Identificación de sus partes

  Etapa 3: Simulaciones con la tarjeta SIE

- FASE III: Recopilación de posibles prácticas

  Etapa 1: Revisión del estado del arte

  Etapa 2: Revisión de material utilizado por la Universidad Industrial de Santander

  Etapa 3: Revisión de los objetivos de aprendizaje de la asignatura de Sistemas Digitales

- FASE IV: Elaboración del hardware

  Etapa 1: Selección de los puertos de salida de la tarjeta SIE a utilizar

  Etapa 2: Utilización de software para el diseño

Etapa 3: Impresión de la PCB

- FASE V: Realización de las guías de cada uno de los laboratorios

  Etapa 1: Selección de las prácticas a realizar

  Etapa 2: Implementación de las prácticas seleccionadas

  Etapa 3: Documentación de las prácticas seleccionadas

- FASE VI: Validación de las guías de laboratorio

  Etapa 1: Entrega de las guías al comité asesor y a estudiantes

  Etapa 2: Aplicación de mejoras a las guías propuestas

- FASE VII: Realización de las guías de los proyectos finales

  Etapa 1: Selección de los proyectos finales

  Etapa 2: Implementación de los proyectos finales

  Etapa 3: Documentación de los proyectos finales

- FASE VIII: Validación de los proyectos finales

  Etapa 1: Entrega de la guía al comité asesor y a estudiantes

  Etapa 2: Aplicación de mejoras a la guía propuesta

- FASE IX: Elaboración del documento final

  Etapa 1: Recopilación de los documentos con sus respectivas mejoras

  Etapa 2: Unificación del material didáctico

3.2 Metodología para el diseño de las guías

3.2.1. Estado del arte

Es necesaria una investigación de la temática que se trabaja en las universidades más relevantes del mundo, con el fin de complementar la utilizada en la Universidad Industrial de Santander referente a la asignatura de sistemas digitales.

3.2.2. Estudio de sugerencias por parte de los alumnos

Debido a que el material didáctico se dirige directamente a los estudiantes, se indaga en las sugerencias que tienen los alumnos en la realización de prácticas en el laboratorio de sistemas digitales, igualmente se reciben propuestas de nuevas actividades con el fin de acomodar las guías a los intereses por parte del estudiante.

3.2.3. Selección de los temas de cada una de las guías

Con el estado del arte de los sistemas digitales y su comparación con los temas trabajados en la Universidad Industrial de Santander se hace una selección de las temáticas que deben ser cubiertas por las guías que se deben desarrollar en el laboratorio.

3.2.4. Definir una estructura para las guías

Es necesario establecer una estructura de las guías, organizando la información para que sea de fácil entendimiento por parte de los estudiantes y cumpla con los objetivos propuestos en cada una de las mismas.

3.2.5. Investigación minuciosa de cada uno de los temas

Para saber con exactitud los conceptos que deben quedar claros para un estudiante de sistemas digitales, es preciso realizar una investigación de cada uno de los temas que se trabajan en las guías propuestas.

3.2.6. Diseño de las guías

Una vez se entiende la temática a trabajar, se procede a diseñar cada una de las guías teniendo presente en cada línea, los objetivos que debe cumplir.

3.2.7. Primera etapa de revisión

Una vez diseñada cada una de las guías, se procede a realizar repetidas revisiones por parte de los autores, haciendo las correcciones pertinentes.

3.2.8. Segunda etapa de revisión

En esta etapa los codirectores de proyecto hacen una revisión de toda la guía, haciendo las pertinentes sugerencias y recomendaciones para que sea un material entendible y productivo para los alumnos de sistemas digitales.

3.2.9. Primera etapa de mejoras

Despúes de recibir las sugerencias por parte de los codirectores se procede a realizar las diversas mejoras teniendo un material mucho más calificado para ser desarrollado por los estudiantes.

3.2.10. Validación por parte de alumnos

Es necesaria una validación por parte de los alumnos con el fin de confirmar que la estructura de la guía y los conceptos planteados sean entendibles y captados por parte de los estudiantes, de igual forma se reciben sugerencias por parte de los directamente beneficiados con el desarrollo de este proyecto.

3.2.11. Segunda etapa de mejoras

Se realiza una segunda fase teniendo en cuenta las sugerencias expuestas por parte de los alumnos, con el fin de mejorar aún más el material.

3.3 Estructura de las guías

La estructura que posee cada una de las guías tiene la siguiente forma:

3.3.1. Introducción

En esta parte se resalta la importancia del tema que se trabaja en la guía con el fin de despertar el interés de aprendizaje por parte del estudiante.

3.3.2. Objetivos

Se describen los logros que se deben alcanzar cuando el alumno desarrolle la totalidad de la guía.

3.3.3. Marco teórico

En esta parte se exponen los conceptos necesarios para desarrollar cada una de las guías y se hace un resumen de la temática trabajada en las clases teóricas, no con el fin de reemplazarlas, sino con la intención de tener a la mano los conceptos fundamentales para que se pueda realizar con éxito la práctica.

3.3.4. Procedimiento general

En esta sección se expone un problema que debe ser solucionado por parte de los estudiantes, proponiendo un proceso que se puede llevar a cabo para la solución del mismo.

3.3.5. Procedimiento paso a paso

Se propone seguir un procedimiento paso a paso para cumplir con lo planteado en el procedimiento general, haciendo una descripción muy detallada de forma tal que se acompañe al estudiante en el desarrollo de todo el proceso.

3.3.6. Resumen

Propone recordar los conceptos más relevantes que se han presentado en las guías, de tal forma que cuando el estudiante lea esta sección, recuerde rápidamente los temas

que fueron asistidos.

3.3.7. Preguntas de prueba

Se dejan unos interrogantes al alumno que deben ser resueltos, con esto se busca que los estudiantes sean partícipes de su aprendizaje y generen conocimientos.

3.3.8. Ejercicio propuesto

Crean una situación, la cual obliga al estudiante a utilizar los conceptos adquiridos en el desarrollo de la guía, con el fin de verificar si los objetivos propuestos fueron o no alcanzados.

4. GUÍAS Y PROYECTOS DE SISTEMAS DIGITALES

4.1 Manual básico de Linux

Para utilizar la tarjeta SIE, es necesario tener conceptos básicos acerca del sistema operativo Linux, pues es sobre éste donde se hacen las diversas configuraciones de la tarjeta y por medio del cual la tarjeta SIE lleva a cabo cada una de las diversas implementaciones que se desarrollarán en el transcurso de la asignatura de sistemas digitales. Así pues, debido a que actualmente este sistema no es lo suficientemente conocido y manejado por los alumnos, es preciso hacer énfasis en los conceptos básicos del mismo, su funcionamiento y principales características, los comandos que se van a utilizar en el transcurso del curso y unos ejercicios con el objetivo de que el alumno desarrolle las destrezas necesarias para manipular de forma exitosa la tarjeta SIE.

4.2 Implementación de una compuerta AND en la tarjeta SIE

Esta guía es de vital importancia. Fue desarrollada con sumo cuidado debido a que es la primera vez que los estudiantes van a interactuar con la tarjeta SIE, luego es importante que perciban que tienen a la mano una herramienta amigable con la que van a desarrollar sus capacidades en lo concerniente a sistemas digitales a lo largo del curso. Luego esta guía fue validada más de cinco veces, por parte de alumnos de las asignaturas de sistemas embebidos y sistemas digitales, haciendo las mejoras pertinentes en repetidas ocasiones.

Esta guía muestra el proceso completo que se debe llevar a cabo cuando se requiere la implementación de un código *VHDL* en la tarjeta SIE. Inicialmente se muestra el comportamiento lógico de una compuerta *AND*, para luego realizar el proceso que se va a repetir en cada una de las prácticas futuras. Luego se brinda una explicación paso a paso, resaltando los archivos necesarios para lograr una implementación y cuidando el

no omitir detalle alguno que deje dudas en el estudiante.

4.3 Visualizador de siete segmentos implementado con ecuaciones estandar

Con este material se inicia la aplicación de los conceptos adquiridos en las clases teóricas. Se realiza una guía donde los alumnos deben repetir el proceso de la guia inmediatamente anterior referente a la implementación de una compuerta *AND*, para describir un circuito combinacional con ecuaciones lógicas, identificando las partes principales de una descripción en *VHDL*, ya que por medio de este lenguaje se van a realizar todas las descripciones en el transcurso de la asignatura, que al ser un lenguaje desconocido para la mayoría de los alumnos, es preciso hacer énfasis en la identificación de sus partes principales. Con esta guía también se pretende que los estudiantes adquieran habilidades en el manejo del lenguaje *VHDL* y adquieran destrezas en el manejo de la plataforma *SIE*.

4.4 Visualizador de siete segmentos implementado con ecuaciones simplificadas

En esta guía se aplican los conocimientos adquiridos referentes a los métodos de simplificación de ecuaciones booleanas utilizando los mapas de *Karnaugh*, implementando el mismo circuito referente a un visualizador de siete segmentos. Se plasman las diferencias de las dos maneras de describir el mismo circuito, dando a conocer las grandes ventajas que se tienen al realizar simplificaciones de las ecuaciones booleanas. El desarrollo de la misma guía plantea la versatilidad del lenguaje VHDL a la hora de implementar un circuito, plasmando que no es necesario ligarse con algún método de descripción; por el contrario el lenguaje *VHDL* representa flexibilidad para la descripción de circuitos.

4.5 Concurrencia

La aplicación de sentencias concurrentes muestra una vez más la versatilidad que tiene el lenguaje *VHDL* para describir circuitos digitales, realizando una implementación del visualizador de siete segmentos, con el objetivo de mostrar las diversas maneras en que

se puede lograr esto, además de aclarar la diferencia entre un circuito secuencial y uno concurrente, ya que generalmente existe una confusión en dicho concepto.
En esta guía se exponen las sentencias concurrentes más utilizadas y hasta este punto se considera que el estudiante ha adquirido las destrezas suficientes para reconocer los archivos principales necesarios para implementar un circuito en la *FPGA* de la tarjeta *SIE*, logrando diferenciar las partes de las que se compone una descripción por medio del lenguaje *VHDL*.

4.6 Descripción estructural

En esta guía se continúa con las temáticas vistas en las clases teóricas, exponiendo paso a paso el proceso que se requiere para realizar una descripción estructural por medio del lenguaje VHDL, complementando y aplicando los conocimientos adquiridos previamente en la asignatura. Se presenta el comportamiento de un sumador completo, realizando su descripción estructural por medio de módulos semisumadores. En esta guía se plantea la importancia que tienen las descripciones estructurales y los diagramas RTL, acompañando al alumno en la descripción estructural de un sumador de dos bits, por medio de los módulos sumador completo y semisumadores.

4.7 ALU

En esta guía se plantea describir uno de los circuitos más básicos e importantes de los cuales se debe tener claridad dentro de las temáticas de sistemas digitales, la ALU (Unidad Aritmético Lógica). Así pues, se propone describir el funcionamiento básico de la ALU por medio del lenguaje VHDL, a través de la descripción de un sumador y un restador binario, representando el resultado mediante el sistema signo-magnitud dando como resultado una ALU restador/sumador en la tarjeta SIE.
Esta guía es de gran importancia debido a que reúne todos los conceptos vistos en las anteriores, mostrando al alumno una aplicación que seguramente han utilizado muchas veces y mostrando la herramienta que están aprendiendo al describir circuitos por medio del lenguaje VHDL.

4.8 FLIP-FLOPS

Continuando con la temática expuesta en clase, en esta guía se presentan los conceptos fundamentales referentes a los circuitos secuenciales, presentando las características, los comportamientos y tipos de *flip-flops* más comunes, haciendo recomendaciones de gran utilidad para llevar a cabo una buena descripción de estos circuitos por medio del lenguaje VHDL, principalmente por medio de la sentencia *process*.
En esta guía se describen circuitos de importancia como el reductor de frecuencia, el cual es de utilidad en la mayoría de las descripciones realizadas para diversos proyectos en el área de sistemas digitales.

4.9 Registros y Contadores

En esta guía se presenta otros de los circuitos más importantes en la asignatura, los registros y contadores. Inicialmente se exponen los conceptos básicos de registro, mencionando los tipos que generalmente se implementan, además de las diferencias entre contadores síncronos y asíncronos, describiendo el proceso de implementación de uno de los registros y uno de los contadores expuestos.

4.10 Máquinas de Estado

En este material se presenta el proceso completo y paso a paso que se debe tener en cuenta para implementar una máquina de estados, por medio de un ejemplo que muestra la forma en que este tipo de circuitos puede tener lugar en la descripción de circuitos por medio del lenguaje VHDL. En la descripción del circuito implementado se sigue mostrando la versatilidad del lenguaje VHDL, realizando el proceso de dos maneras distintas para seguir ilustrando las diversas soluciones que pueden darse en la descripción de circuitos.
En esta guía se exponen las máquinas de estado de Moore y Mealy, al igual que la diferencia que existe entre ellas, logrando descripciones por medio de la sentencia *type*.

4.11 Proyectos

La idea principal del planteamiento de estos dos proyectos, se basa en dar las herramientas básicas de circuitos electrónicos que sean útiles para realizar mejoras de los mismos ó utilizarlos en futuros proyectos, de modo que los estudiantes puedan capturar cualquier tipo de señal, así como manejar cualquier tipo de circuito que exceda la capacidad de potencia de la SIE.

4.11.1. Proyecto 1: Edificio Inteligente

Uno de los proyectos propuestos al final de la asignatura, consiste en la adquisición de una señal analógica la cual proviene de un sensor y debe ser adaptada a un ADC de comunicación serial, para luego interactuar con un teclado *PS2* y responder a unas especificaciones y requisitos que se deben cumplir para el desarrollo del proyecto.
Lo que se busca con esta propuesta es ampliar el campo en la realización de proyectos de la asignatura de sistemas digitales, ya que debido a conocimientos requeridos en circuitos electrónicos, en muchas ocasiones se ven frustradas muchas de las ideas presentadas por parte de los estudiantes para su realización, luego se busca construir un conocimiento integral donde se complementen tanto los conocimientos básicos de sistemas digitales como los de circuitos electrónicos, además se pide construir el modelo a pequeña escala, debido a que un trabajo ingenieril debe tener en cuenta aspectos diversos que pueden llegar a ser determinantes a la hora de dar solución a un problema de la vida real.
Así pues, se plantea la construcción y el diseño de un edificio inteligente el cual cumpla con las innumerables aplicaciones que tiene una edificación de su condición, este proyecto está planteado para muchos años, de forma tal que los estudiantes que tomen el curso en futuros semestres, hagan sus aportes o mejoren sobremanera proyectos planteados donde a elección de los docentes, es posible requerir una aplicación específica, o aceptar nuevas ideas por parte de los estudiantes, o también tomar una de las mejoras que se proponen al final del proyecto, de forma tal que sea un proyecto en evolución continua.
Ahora bien, en ese orden de ideas para un semestre inicial se plantea un problema de la

vida real, con el ánimo de despertar el interés en los alumnos al aplicar los conocimientos adquiridos tanto en las clases teóricas como en las experiencias vividas en el laboratorio en la solución de una necesidad que requiera un buen trabajo de ingeniería. Por lo tanto se requiere de un sistema de seguridad para tomar acciones preventivas contra los movimientos de la tierra que muchas veces desencadenan tragedias que pueden llegar a disminuirse con una adecuada reacción en esos momentos críticos. Así pues, se necesita un sistema que registre continuamente los niveles de intensidad de un temblor, estratificándolos en tres rangos: uno que identifique los movimientos normales de la tierra, otro que indique precaución para estar atento y un tercer nivel en el que se active una alarma sonora para poder evacuar el edificio. Adicionalmente se propone para realizar campañas de prevención, una clave secreta que al ser digitada a través de un teclado *PS2*, ponga en marcha una simulación progresiva en los indicadores de temblor con el fin de estudiar las reacciones de los residentes y realizar tareas de simulacro.

4.11.2. Proyecto 2: Automóvil

Se plantea un segundo proyecto en donde a cambio de adquirir una señal analógica, se requiere manejar circuitos de alta potencia que la tarjeta SIE no puede manejar directamente, por lo tanto requiere de un circuito electrónico para poder tener control sobre sistemas que excedan su capacidad de entrega de potencia, adicionalmente se propone la forma de adaptar circuitos a las necesidades que se presenten.

Así pues, se plantea la idea de trabajar en un proyecto que pueda ser mejorado semestre tras semestre, ya sea realizando mejoras o presentando ideas innovadoras que sean iniciativas por parte de los estudiantes, además de reunir conceptos integrando los sistemas digitales, los circuitos electrónicos y llevando a cabo un trabajo de ingeniería para resolver un problema de la vida real. Por lo tanto se presenta un automóvil como otro de los sistemas que tiene múltiples aplicaciones y al cual se le pueden adaptar gran cantidad de ideas siendo esta una aplicación que despierta el interés de los estudiantes. Entonces, en un proyecto inicial se plantea dar movimiento al automóvil hacia adelante y hacia atrás, controlando su velocidad por medio de un teclado matricial, simulando

los cambios de un automóvil real, además de asignarle una bocina para evitar peatones y automóviles imprudentes que se atraviesen por su camino, además cabe recordar que los proyectos deben ser aplicados a pequeña escala.

5. VALIDACIÓN

Uno de los aspectos más importantes a considerar, es lo que respecta al proceso de validación de todo el material didáctico diseñado en este proyecto, pues a partir de este, se puede tener una medida de la calidad del mismo, evaluando si el material resulta claro, entendible, legible y sobre todo que verdaderamente complemente los conocimientos adquiridos en cada una de las clases teóricas de la asignatura de sistemas digitales, apuntando hacia la construcción de conocimiento por medio de experiencias en el laboratorio de sistemas digitales.

En ese orden de ideas, se realizaron varias fases de validación donde estudiantes de niveles superiores, profesores de sistemas digitales y estudiantes de niveles inferiores colaboraron sobremanera en este proceso, realizando las sugerencias pertinentes de forma tal que permitieran a los autores realizar las mejoras correspondientes para tener un material con un mayor nivel de comprensión por parte de los alumnos.

Así, después de diseñar cada una de las guías se procede a realizar una revisión exhaustiva por parte de los autores, con el objetivo de revisar ortografía, redacción, nivel de comprensión y organización, preparándola así para una segunda fase de revisión.

La segunda etapa de validación es realizada por los codirectores del proyecto, que con su experiencia en el tema y en lo referente a la enseñanza, realizaron sugerencias constructivas permitiendo así realizar las mejoras respectivas.

La tercera etapa consiste en el desarrollo de la guía por parte de estudiantes de niveles superiores del programa de Ingeniería Electrónica, para lo cual hubo gran colaboración por parte de los alumnos del curso Diseño con Microprocesador y Microcontroladores, donde los autores acompañaban a los alumnos en el desarrollo de la guía, haciendo anotación de las diversas sugerencias y comentarios que aparecían en el transcurso del mismo. Cabe resaltar que algunas de las guías diseñadas se sometieron a una cuarta fase de validación, donde un alumno de primer nivel del programa de Ingeniería Electrónica

aportó las sugerencias respectivas en el desarrollo de la misma.

La enseñanza en el área de sistemas digitales exige una gran responsabilidad y hace referencia a que es la primera vez que los alumnos van a entrar en este campo, luego el mayor proceso de validación se realizó con la primera guía concerniente a la implementación de un circuito en la tarjeta SIE. Por tal motivo los autores asistieron a los cursos de sistemas digitales que se dictan en el primer periodo académico del 2012, acompañando a los alumnos en el desarrollo de la guía, realizando mejoras y evaluando los resultados a través de otro grupo de alumnos, donde los resultados fueron satisfactorios.

Un buen proceso de validación es fundamental cuando se quiere diseñar un material de enseñanza para estudiantes en cualquier campo, luego se dispuso de mucha atención y tiempo para recibir todo tipo de sugerencias, comentarios, críticas y observaciones que pudieran hacer del resultado de este proyecto una verdadera guía para los compañeros que están comenzando su proceso de formación profesional, encaminando el trabajo realizado a mejoras continuas.

6. CONCLUSIONES Y RECOMENDACIONES PARA TRABAJOS FUTUROS

Una de las formas de aportar al desarrollo industrial de Colombia, es a través de la formación de profesionales capaces de asimilar, adaptar, innovar y aplicar tecnología proveniente de un país con mayor desarrollo en el área, con el fin de fomentar iniciativas de evolución tecnológica que aporten al crecimiento del país.

Con la utilización de la tarjeta SIE en las universidades, se busca contribuir a minimizar el margen de atraso tecnológico que presenta Colombia respecto a países más industrializados, fomentando desde los primeros cursos de sistemas digitales la consciencia de aplicar los conocimientos adquiridos en pro del desarrollo a nivel local.

Siendo una conocida problemática la falta de recursos económicos para la educación en Colombia, la tarjeta SIE ofrece una disminución considerable de costos en la adquisición de herramientas de software necesarias para la implementación de circuitos, logrando así sacar mejor provecho de los recursos económicos con que cuenta la universidad.

Una FPGA se acomoda a las necesidades de aprendizaje que presentan los alumnos de sistemas digitales, pues al ser un arreglo de compuertas programable, les permite implementar diversos circuitos y repetidas veces, lo cual aporta en gran manera al entendimiento de conceptos básicos de la asignatura, ostentando como principal ventaja la posibilidad de ser reconfigurada.

El uso de la plataforma SIE promueve difundir los conocimientos e investigaciones que se llevan a cabo dentro de la universidad, exponiendo los avances generados en un ámbito de manejo público, donde cualquier otra universidad del país pueda tener acceso a la información, para que la absorción de una nueva tecnología sea de uso común y así generar por parte de las industrias una credibilidad en los profesionales de Colombia, iniciando un proceso encaminado a la inversión en los productos locales, pudiendo distribuir y modificar cualquier diseño concebido en la plataforma, con el

compromiso de documentar todo tipo de idea que se genere a partir de esta.

La utilización de la plataforma SIE se encamina hacia la unión de la industria con la academia, mostrando al alumno de sistemas digitales que puede resolver problemas de la vida real, incentivándolo a pensar en las necesidades internas que presenta Colombia y en busca de soluciones, mostrando las ventajas que puede acarrear esto, como generación de empleo, ganancias económicas y sobre todo cumplir con la labor ingenieril para la cual es formado que es la de servir mejor a la sociedad.

Como un aporte del presente trabajo, se considera el hecho de promover en los estudiantes de sistemas digitales la capacidad de aplicar la ingeniería de manera íntegra a la solución de diversos problemas de la vida real, donde tomando principalmente los conceptos adquiridos en sistemas digitales se puedan complementar con conocimientos básicos de circuitos electrónicos, proponiendo al estudiante realizar modelos a pequeña escala de su trabajo, donde considere aspectos fundamentales a la hora de dar solución a determinados problemas prácticos, llevando a cabo el proceso de diseño completo como selección de componentes, interpretación de hoja de datos, montajes físicos, diseño de circuitos impresos, entre otros.

Toda experiencia que se adquiere en un laboratorio, enriquece sobremanera al estudiante, dando el complemento perfecto a los conocimientos teóricos adquiridos en el transcurso de las asignaturas, pudiendo así cumplir con un proceso de aprendizaje completo.

La validación de todo material que vaya dirigido a estudiantes es de vital importancia, pues es la forma de tener una medida de la calidad del mismo, luego existe un compromiso grande en el desarrollo de dicho material, teniendo cuidados en su estructura y en la forma de presentar al alumno las diversas competencias a desarrollar.

Para futuros trabajos se sugiere la realización de proyectos conjuntos, donde no sólo alumnos de sistemas digitales puedan llevar a cabo el desarrollo de algún proyecto independiente, sino con la colaboración de estudiantes de asignaturas del siguiente nivel relacionadas con los sistemas digitales, que puedan concebir proyectos mucho más interesantes para una problemática común. También se plantea el estudio de necesidades

en programas como medicina, geología, ingenierías en general y en cualquier otro campo donde se requiera resolver problemas, contribuyendo cada vez al desarrollo socio económico del país.

La continuación de este trabajo puede contemplar visitas a las diversas industrias de Bucaramanga, donde se haga un estudio de una necesidad particular, la cual pueda ser suplida por los alumnos de las diversas asignaturas referentes a sistemas digitales, dando una adecuada organización al proyecto donde se requiera aplicar los conocimientos adquiridos.

La necesidad de transferencia tecnológica sugiere el estudio de plataformas que se puedan adaptar a las condiciones socio-económicas del país y sobre todo a las necesidades que tiene la academia en un proceso continuo de enseñanza y formación, sugiriendo así la investigación de tecnologías desarrolladas en países industrializados, de forma tal que se pueda tener iniciativa propia por parte de la Universidad Industrial de Santander.

Bibliografía

[1] Alberto Mora Javier García de Jalón, Iker Aguinaga. Aprenda linux como si estuviera en primero. http://www.digilentinc.com, 2000.

[2] Camargo, Carlos Ivan. SIE: Plataforma Hardware copyleft para la ensenanza de sistemas digitales. Universidad Nacional de Colombia. 2010. 6 p.

[3] Creative Commons. Licencias Creative Commons. http://creativecommons.org/licenses.,2004.

[4] FAIRCHILD SEMICONDUCTOR GENERAL PURPOSE 6-PIN PHOTOTRANSISTOR OPTOCOUPLERS. http://www.mbari.org/mars/images/4N37.pdf,

[5] In Electronics. Generación de señales PWM con el microcontrolador PIC16F84 http://usuarios.multimania.es/ccencho/2005Abril.pdf, 2004.

[6] MIGUEL GRASSI PuenteH.pdf. http://www.miguelgrassi.com.ar/mecatronica/PuenteH.pdf, 2006.

[7] Proyectos Electronicos http://proyectoselectronics.blogspot.com/2008/09/optoacoplador-que-es-y-como-funcionan.html,

[8] Qi Hardware http://en.qi-hardware.com/wiki/SIE.

[9] Thomas L. Floyd. Digital Fundamentals (7th Edicion). Prentice-Hall, Inc, Madrid, 2000.

[10] W.Spraul, C. Camargo, and A. Wrang. Proyecto SACK. http://en.qi-hardware.com/wiki/SAKC.

ANEXO A. GUÍAS DE LABORATORIO SISTEMAS DIGITALES

Índice

Lista de Figuras

Lista de Tablas

LABORATORIO 1

Manual Básico de Linux.

1.1 Manual Básico de Linux

1.1.1 Objetivos

- Hacer una introducción acerca del sistema operativo Linux.
- Dar a conocer los comandos básicos utilizados en Linux para interactuar con la plataforma SIE.

1.1.2 ¿Qué es Linux?

Linux es un sistema operativo gratuito y de libre distribución inspirado en el sistema Unix, escrito por Linus Torvalds con la ayuda de miles de programadores en internet. Unix es un sistema operativo desarrollado en 1970, una de las mayores ventajas se debe a que es fácilmente portable a diferentes tipos de ordenadores, por lo que existen versiones de Unix para casi todos los tipos de ordenadores, desde PC y Mac hasta estaciones de trabajo y superordenadores. Al contrario de otros sistemas operativos, como por ejemplo MacOS (Sistema operativo de los Apple Macintosh), Unix no está pensado para ser fácil de emplear, sino para ser sumamente flexible. Por lo tanto Linux no es en general tan sencillo de emplear como otros sistemas operativos, aunque se están realizando grandes esfuerzos para facilitar su uso. Pese a toda la enorme flexibilidad de Linux y su gran estabilidad (y el bajo coste) han hecho de este sistema operativo una opción muy interesante para aquellos usuarios que se dediquen a trabajar a través de redes, naveguen por internet, o se dediquen a la programación. Además el futuro de Linux es brillante y cada vez más gente y más empresas (entre otras IBM, Intel, Corel) están apoyando este proyecto, con lo que el sistema será cada vez más sencillo de emplear y los programas serán cada vez mejores. [25].

1.1.3 Acerca de la shell

Para la interacción entre un usuario y el sistema operativo es necesario algún elemento que interprete los deseos del usuario y se los comunique al sistema operativo, esperando una respuesta del sistema para mostrársela en pantalla

nuevamente al usuario y terminar el proceso. Así pues, el intérprete de comandos se define como la interfaz entre un usuario y el sistema operativo, razón por la cual se le da el nombre de Shell que en inglés tiene el significado de caparazón, teniendo en su interior el sistema.

1.1.4 ¿Qué hace la Shell?

Actúa como un intermediario entre el sistema operativo y el usuario gracias a las líneas de comando que introduce el usuario. El proceso que hace la Shell consiste en:

- Interpretar los comandos
- Transmitir los comandos al sistema
- Arrojar un resultado

En la figura 1.1, es posible notar el proceso interno que realiza la shell desde que se le ingresa por medio del teclado una orden, hasta que responde con una acción o resultado mostrado en la pantalla del ordenador donde se esté trabajando.

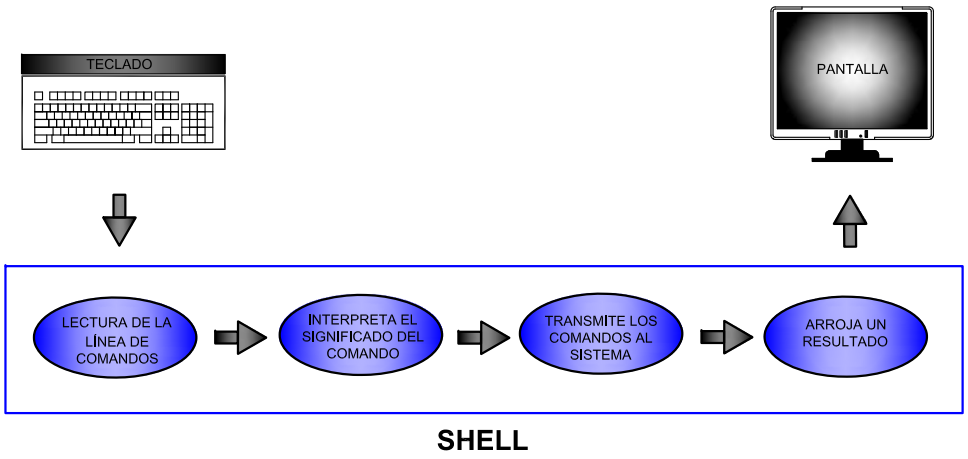


Figura 1.1: Proceso interno de la shell

FUENTE:Los autores

Cabe mencionar que cada usuario tiene una Shell predeterminada y que existen varios tipos de Shell:

- sh llamada Bourne Shell
- bash llamada Bourne again Shell
- csh llamada C Shell
- Tcsh llamada Tonex C shell

- Ksh llamada Korn Shell
- Zsh llamada Zero Shell

Para saber qué tipo de *Shell* se posee, basta con leer la última línea del archivo `passwd`, el cual se encuentra generalmente en la dirección `/etc/passwd`.

Algunos usuarios se sienten más a gusto usando una *Shell* determinada, cada una de estas *Shells* tiene sus particularidades y su forma de interpretar los comandos, que aunque tienen la misma base *sh*, en algunos aspectos unas *Shells* son más favorables que otras, cada usuario se inclina en determinado punto por alguna *Shell* especifica dependiendo de sus necesidades. Inicialmente se recomienda empezar el proceso con la *Shell bash*, la cual es una evolución de *sh*, en la que se pueden hacer todas las funciones de *Bourne Shell*, pero se le agregan unas nuevas por lo que su nombre varia a *Bourne Again Shell*, es decir que se muestra como una evolución de la *sh*.

1.1.5 ¿Qué es el prompt?

El *prompt* hace referencia a la pantalla o terminal en donde la *Shell* y el usuario leen los comandos que este último digita. Para la mayoría de las *shells*, el indicador consiste en el nombre del equipo seguido de dos puntos (:), el directorio actual y un caracter que indica el tipo de usuario que está trabajando, donde el signo `$` especifica un usuario normal, y el signo # especifica el usuario administrador. Así pues, en el *Script* 1.1 se muestra más claro lo anteriormente mencionado.

```
1 Equipo: /directorio/actual$  en este espacio se ingresan las peticiones del usuario o comandos
```

Script 1.1: Prompt de Linux-Ubuntu 10.10

1.1.6 Línea de comandos

Una línea de comandos es una cadena de caracteres formada por un comando que corresponde a un archivo ejecutable del sistema. También se interpreta como un comando de *Shell* con algunos argumentos y opciones como bien se muestra en la figura 1.2.

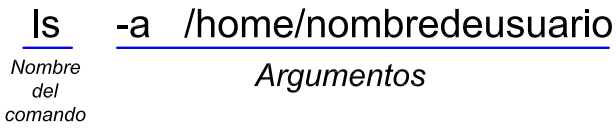


Figura 1.2: Línea de comandos

FUENTE:Los autores

Los argumentos que empiezan con un guión "-", en realidad son opciones del comando. Para cada comando hay una cierta cantidad de opciones, por ejemplo el comando **ls** cuenta con algunas opciones como **ls -a, ls -l, ls -lh**, entre otras.

Si se desea averiguar cuales son las opciones con las que cuenta determinado comando basta con digitar la palabra *man* seguida del comando, como se muestra en el Script 1.2, y el cual desplegará todas las opciones que se derivan del comando requerido.

```
1 Equipo: /directorio/actual$      man ls
```

Script 1.2: Prompt de Linux-Ubuntu 10.10

1.1.7 COMANDOS FUNDAMENTALES DE LINUX

Para interactuar con el *prompt* de Linux, es necesario tener un manejo básico de los comandos más relevantes que se utilizan. Como primera medida será necesario configurar el teclado de forma tal que coincida lo digitado con lo que aparece en pantalla. Cabe resaltar que si no se logra la configuración deseada por medio del entorno gráfico que muestra Linux, se puede proceder a digitar en la terminal lo mostrado en el Script 1.3.

```
1 $       setxkbmap -layout es
```

Script 1.3: Prompt de Linux-Ubuntu 10.10

Seguido de esto se exponen los comandos básicos necesarios para un buen manejo de Linux que se ha planteado.

1.1.7.1 pwd

Muestra la dirección del directorio actual.

1.1.7.2 cd

Cambia la dirección del directorio actual a home/usuario.

El símbolo "/" hace referencia a la raíz principal y el símbolo $\sim$ hace referencia al usuario en el home.

1.1.7.3 cd /

Cambia la dirección actual a la raíz principal.

1.1.7.4 cd $\sim$

Cambia la dirección actual a home/usuario.

1.1.7.5 cd ..

Cambia al directorio padre inmediato, es decir al que contiene la carpeta actual.

1.1.7.6 cd .

Continua en el directorio actual.

1.1.7.7 cd -

Cambia la dirección actual al directorio en el que se encontraba anteriormente.

La tecla `Tab`, es de gran utilidad al momento de completar directorios, pues tan solo con poner 2 o 3 letras del directorio que se desee y seguido la tecla `Tab`, se autocompletará el nombre del directorio evitando que se coloquen directorios que no existan.

1.1.7.8 ls

Muestra el contenido de la carpeta actual excepto los archivos ocultos

1.1.7.9 ls -a

Muestra todos los archivos de la carpeta actual incluidos los archivos ocultos

1.1.7.10 ls -l

Muestra todos los archivos con información como nombre, permiso, usuario y grupo al que pertenece, tamaño, fecha y hora de creación.

1.1.7.11 ls -l -h

Cumple la misma función que el comando ls -l, tan solo que el tamaño del archivo lo muestra con el respectivo prefijo del sistema internacional de unidades.

1.1.7.12 EJERCICIO 1

```
1  $ cd /                  ### Se requiere estar en la raíz
2  $ pwd                   ### Debe mostrar la dirección actual es decir la raíz ó /
3  $ ls                    ### Muestra el contenido de la raíz
4  $ cd home               ### Se requiere entrar al directorio home
5  $ pwd                   ### Muestra la dirección actual /home
6  $ ls home               ### Muestra el contenido del home
7  $ cd usuario            ### Entra al directorio de cada usuario
8  $ ls usuario            ### Muestra el contenido de /home/usuario
9  $ cd Downloads          ### Entra al directorio de descargas
10 $ pwd                   ### verifica la dirección del directorio actual
11 $ clear                 ### Limpia la pantalla
```

Script 1.4: Ejercicio 1

1.1.7.13 mkdir

Permite crear una carpeta en la dirección que se desee.

1.1.7.14 mkdir -p

Permite crear una carpeta dentro de otra en la dirección actual.

1.1.7.15 rmdir

Se usa para eliminar una carpeta, y funciona siempre que la carpeta a remover se encuentre vacia.

1.1.7.16 rmdir -p

Permite borrar una cadena de carpetas siempre que la carpeta del final de la cadena se encuentre vacia.

1.1.7.17 EJERCICIO 2

```
1 $ mkdir  /home/usuario/Downloads Prueba     ### Crea una carpeta llamada Prueba en Downloads.
2 $ cd /home/usuario/Downloads                ### Ingresar a la carpeta Downloads
3 $ ls Downloads                              ### Muestra el contenido de Downoads donde debe
4                                             ### encontrarse la carpeta Prueba creada.
5 $ mkdir-p CarpetaA/CarpetaB                 ### crea una CarpetaB dentro de otra CarpetaA en el
6                                             ### directorio actual.
7 $ rmdir Prueba                              ### Remueve la carpeta Prueba
8 $ rmdir-p CarpetaA/CarpetaB                 ### Remueve las carpetas A y B
```

Script 1.5: Ejercicio 2

1.1.7.18 file

Describe el tipo de archivo.

1.1.7.19 touch

Crea un archivo.

1.1.7.20 touch -t

Crea un archive con el año, mes, hora y nombre de creación que se desee.

1.1.7.21 rm

Se usa para remover un archivo.

1.1.7.22 rm -i

Se usa para remover un archivo, con la diferencia que pregunta antes de borrarlo si el usuario esta seguro de eliminarlo. Es por seguridad.

1.1.7.23 rm -rf

Remueve cualquier archivo o carpeta sin importar si está o no vacía. Se debe tener precaución pues equivocadamente se puede eliminar algún archivo importante del sistema.

1.1.7.24 EJERCICIO 3

```
1 $ cd /home/usuario/Downloads          ### Ingresar a la carpeta Downloads
2 $ touch documento1                    ### Crea un archivo llamado documento1
3 $ ls                                  ### verifica que documento1 fue creado en Downloads
4 $ touch -t 149210121200 America       ### Crea un archivo con fecha 12 de octubre de 1492
5 $ file America                        ### Describe el tipo de archivo America
6 $ rm America                          ### Remueve America
```

Script 1.6: Ejercicio 3

1.1.7.25 cp

Se utiliza para copiar un archivo, digitando la fuente y el destino.

1.1.7.26 cp archivoA archivoB

Copia el archivo A como B en el directorio actual

1.1.7.27 cp ArchivoA carpeta

Copia el archivo A dentro de la carpeta

1.1.7.28 cp -r

Copia una carpeta dentro de otra carpeta

1.1.7.29 EJERCICIO 4

Es posible copiar varios archivos dentro de una carpeta. Así pues en el ejercicio 4 se demuestra como copiar los archivos 1,2,3 y 4 en la carpeta de destino que se seleccione.

```
1 $ cp archivo1 archivo2 carpeta1/archivo3 carpeta2/archivo4 CARPETADESTINO
```

Script 1.7: Ejercicio 4

1.1.7.30 cp -i

Previene que sobrescribamos sobre un archivo existente con el mismo nombre. En caso que el archivo exista, preguntará antes si se desea sobrescribir en este.

1.1.7.31 cp -p

Hace una copia incluyendo la hora y fecha en que fue creado, guardado con el nombre que se elija.

1.1.7.32 ll

Muestra todos los archivos y carpetas

1.1.7.33 ll *

Muestra todos los archivos y carpetas, incluyendo el contenido de las segundas.

1.1.7.34 mv

Cambia el nombre de un archivo o lo mueve a otra carpeta

1.1.7.35 EJERCICIO 5

```
1  $ cd ~                                ### Ingresar al home/usuario
2  $ cd Downloads                        ### Ingresa a la carpeta Downloads
3  $ touch archivouno                    ### Crea archivouno dentro de Downloads
4  $ touch archivodos                    ### Crea archivodos dentro de la carpeta Downloads
5  $ mkdir Carpeta1                      ### Crea el directorio Carpeta1
6  $ mkdir Carpeta2                      ### Crea el directorio Carpeta2
7  $ cp  archivouno archivounocop        ### Copia el archivouno con el nombre archivounocop
8  $ cp archivodos Carpeta2              ### Copia el archivodos dentro de la Carpeta2
9  $ cp -r Carpeta1 Carpeta2             ### copia la Carpeta1 dentro de la Carpeta2
10 $ cd /home/usuario/Downloads          ### Ingresar a Downloads
11 $ ll *                                ### Muestra todo el contenido de Downloads
```

Script 1.8: Ejercicio 5

1.1.7.36 head

Muestra las primeras 10 lineas de un archivo.

1.1.7.37 head -4

Muestra las primeras cuatro líneas de un archivo.

1.1.7.38 tail

Muestra las ultimas diez líneas de un archivo.

1.1.7.39 cat

Permite ver el contenido de un archivo de texto.

1.1.7.40 cat >

Crea un archivo de texto. Al terminar el contenido del archivo se finaliza con Ctrl + z.

1.1.7.41 cat -b

Muestra el contenido de un archivo con las líneas enumeradas excluyendo la numeración de los espacios.

1.1.7.42 cat -n

Muestra la numeración de todas las líneas incluyendo las de espacios.

1.1.7.43 tac

Muestra el contenido de un archivo en orden desde la última línea hasta la primera.

Cuando el contenido a visualizar es muy largo, el comando *more* lo mostrará página por página usando la barra espaciadora y volverá al estado normal de la terminal tecleando la letra `q`.

1.1.7.44 cat > archivo << stop

Crea un archivo de texto que cuando se digite la palabra stop, será finalizado y guardado.

1.1.7.45 Cat archivo > nuevonombre

Copia el contenido de archivo en uno nuevo de nombre nuevonombre.

1.1.7.46 EJERCICIO 6

```
1  $ cd /home/usuario/Downloads     ### Ingresar a la carpeta Downloads
2  $ cat > nuevoarchivo << stop     ### Crea un archivo de texto llamado nuevoarchivo
3  > uno
4  > dos
5  > tres
6  > cuatro
7  > cinco
8  > stop
9  $ head -4 nuevoarchivo           ### Muestra las primeras cuatro líneas de nuevoarchivo
10 $ tail -3 nuevoarchivo           ### Muestra las tres ultimas líneas de nuevoarchivo
11 $ cat -n nuevoarchivo            ### Muestra todas las líneas de nuevoarchivo numeradas
12 $ tac nuevoarchivo               ### Muestra nuevoarchovo desde la ultima hasta la primer
13                                  ### línea
```

Script 1.9: Ejercicio 6

1.1.7.47 which

Muestra la ubicación de los comandos externos.

1.1.7.48 alias

Sirve para crear un alias que el usuario recuerde fácilmente, tambien es usado para mostrar los alias que son usados actualmente.

1.1.7.49 unalias

Deshace un alias creado.

1.1.7.50 echo

Repite en pantalla el mensaje que se desee.

1.1.7.51 EJERCICIO 7

```
1 $ which cp ls mv rm cd mkdir pwd        ### Muestra la ubicación de los comandos
2 $ alias                                 ### Muestra los alias existentes
3 $ alias c=clear                         ### crea un alias para clear denominado c
4 $ c                                     ### Limpia la pantalla
5 $ alias                                 ### Muestra los alias existentes
6 $ echo VOY POR BUEN CAMINO              ### repite el mensaje en la pantalla
```

Script 1.10: Ejercicio 7

1.1.8 OPERADORES ÚTILES DE CONTROL

Dentro de los operadores de control más útiles se encuentran los siguientes: ";", "&&", "||" y el "#". Donde el punto y coma ";" sirve para ejecutar varios comandos a la vez; el par de Ampersand "&&" se interpretan como una compuerta AND lógica, donde el segundo comando solo se ejecuta si el primero se ejecutó con éxito. Ahora bien siguiendo con la descripción de operadores de control, el tercero de ellos "||" representa la compuerta lógica OR, donde el segundo comando es ejecutado solo cuando el primero falla, de lo contrario no se ejecuta, y por último cabe mencionar que cuando se tiene el signo numeral "#", todo lo que se escribe después de este es ignorado por la Shell. Generalmente es usado para escribir comentarios.

Así pues, se considera que con la información expuesta en este pequeño manual de Linux, y con los ejercicios que se proponen en el mismo, se pretende preparar a un usuario en el manejo básico de los comandos más importantes, que se requieren a la hora de interactuar con la terminal de Linux, así como exponer conceptos básicos que sirvan de introducción al mundo de la tarjeta SIE, y poder interactuar con la misma, a través de una serie de prácticas enfocadas a los sistemas digitales.

1.1.8.1 EJERCICIO 8

```
1 $ echo Aprender                  ### Repite en la pantalla la palabra Aprender
2 $ echo Linux                     ### Repite la palabra Linux
3 $ echo Aprender ; echo Linux     ### Repite las palabras del par de comandos, Aprender y Linux.
4 $ techo Aprender && echo Linux                  ### No repite ninguna palabra
5 $ echo Aprender && echo Linux                   ### repite Aprender y Linux
6 $ echo primero || echo segundo ; echo tercero   ### repite primero y tercero
```

Script 1.11: Ejercicio 8

LABORATORIO **2**

Implementación de una compuerta AND en la tarjeta SIE.

En medio de todas las dificultades está la oportunidad.

EINSTEIN

2.1 INTRODUCCIÓN

A medida que pasa el tiempo, los sistemas digitales tienen una participación más activa en el mundo que nos rodea y se hace fundamental su entendimiento y desarrollo a través de diversas herramientas que contribuyen con su investigación. Ahora bien, como una de estas herramientas, se presenta la plataforma SIE, la cual ha tenido gran acogida por parte de estudiantes y docentes de diversas áreas, contribuyendo en gran manera con el aprendizaje desde usuarios que se inician en el mundo digital, hasta usuarios con conocimientos avanzados en el tema.

Así pues, resulta de gran importancia, entender todo el proceso que se debe realizar al momento de implementar un circuito digital en la SIE, realizando descripciones por medio del lenguaje VHDL.

2.2 OBJETIVOS

- Establecer los pasos necesarios para la implementación de un código en VHDL en la tarjeta SIE.
- Reconocer los ficheros necesarios para la implementación de un diseño digital dentro del FPGA.

2.3 MARCO TEÓRICO

La puerta AND es una de las más utilizadas a la hora de construir arreglos de funciones lógicas, ésta puede tener dos o más entradas y realiza la operación que se conoce como multiplicación lógica. El término puerta se utiliza para describir un circuito que realiza una operación lógica básica; la puerta AND puede llegar a tener dos o más entradas pero una única salida, como lo indican los símbolos lógicos estándar mostrados a continuación en la figura 2.1.

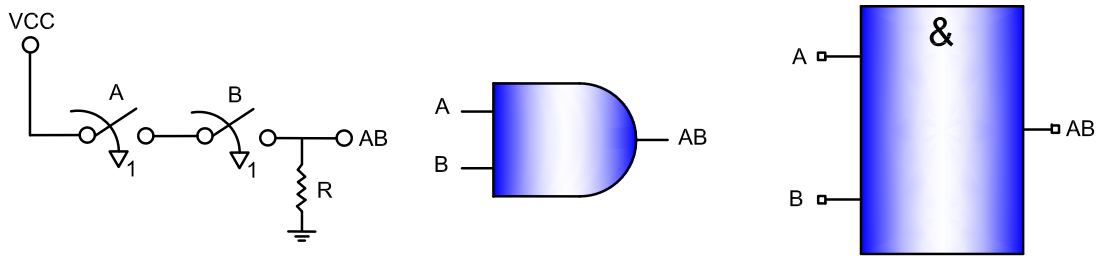


Figura 2.1: Símbolos lógicos estandar de una compuerta AND.

FUENTE:Los autores

En la primera representación de la figura 2.1 es posible notar la manera de implementar una multiplicación lógica por medio de *switches*, así pues, las variables representadas por las letras A y B sólo pueden tomar dos posibles valores: 1 ó 0. Si el *switch* se encuentra abierto, representa un valor lógico de 0 ó bajo, por el contrario si el *switch* se encuentra cerrado representa un valor lógico de 1 ó alto. Seguido a esto se encuentran las representaciones más comunes encontradas en libros y diversos circuitos. Las representaciones anteriormente mostradas, presentan una compuerta AND con dos entradas, pero puede llegar a tener cualquier número de entradas superior a este.

2.3.1 OPERACIÓN LÓGICA DE UNA COMPUERTA AND

La puerta AND genera una salida de nivel alto sólo cuando todas las entradas están a nivel alto. Cuando cualquiera de las entradas está a nivel bajo, la salida se pone a nivel bajo. Por lo tanto el propósito básico de una puerta AND es determinar cuándo ciertas condiciones de entrada son simultáneamente verdaderas, como lo indican todas sus entradas estando a nivel alto y producir una salida a nivel alto, para indicar que esas condiciones son verdaderas. Las entradas de la puerta AND de dos entradas se designan mediante A y B, la salida con AB, representando así la multiplicación lógica entre las correspondientes entradas. Por lo tanto es posible establecer que el funcionamiento de la puerta es el siguiente: En una puerta AND de dos entradas, la salida AB es un nivel alto si A y B están a nivel alto; y AB es un nivel bajo si A es un nivel bajo, B es un nivel bajo, o A y B están ambos a nivel bajo.

Así pues, se puede resumir y analizar el comportamiento de la puerta AND en la tabla 2.1 en la que se muestran todas las combinaciones posibles de la entrada con las correspondientes salidas, donde se puede notar el comportamiento descrito de la siguiente manera: cuando A=0 y B=0 entonces la salida AB = 0, cuando A=0 y B=1 entonces la salida AB = 0, cuando A=1 y B=0 entonces la salida AB = 0 y cuando A=1 y B=1 entonces la salida AB = 1.

ENTRADA A	ENTRADA B	SALIDA AB
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2.1: Tabla de verdad de una compuerta AND

FUENTE:Los autores

2.4 PROCEDIMIENTO GENERAL

Implementar el comportamiento de una compuerta AND en el FPGA de la tarjeta SIE siguiendo los pasos que se muestran a continuación:

1. Crear un directorio que contenga los ficheros a implementar en la tarjeta
2. Crear el archivo de extensión .vhd
3. Descripción del código en el lenguaje VHDL
4. Copiar el archivo Makefile en la carpeta de trabajo
5. Asignación de pines creando el archivo de extensión .ucf
6. Comprobar la existencia de los archivos necesarios para sintetizar e implementar
7. Realizar la síntesis de la descripción
8. Verificar posibles errores
9. Conectar la tarjeta SIE
10. Configuración del puerto USB
11. Enviar información al procesador
12. Acceder al procesador de la tarjeta
13. Enviar información al FPGA

Para llevar a cabo el proceso en el cual se implementa un circuito en el FPGA de la tarjeta SIE, es necesario primero que todo, tener el diseño en un lenguaje de descripción de hardware VHDL, lo siguiente a la descripción del circuito, es realizar el proceso de síntesis, lo cual se logra con la utilización de un programa proporcionado por *Xilinx*, con la ayuda del archivo *Makefile*, de lo cual se obtiene como resultado un archivo de extensión *.bit.* Posteriormente se envía este archivo a la SIE por medio del protocolo de comunicaciones *ssh*, asignando dirección

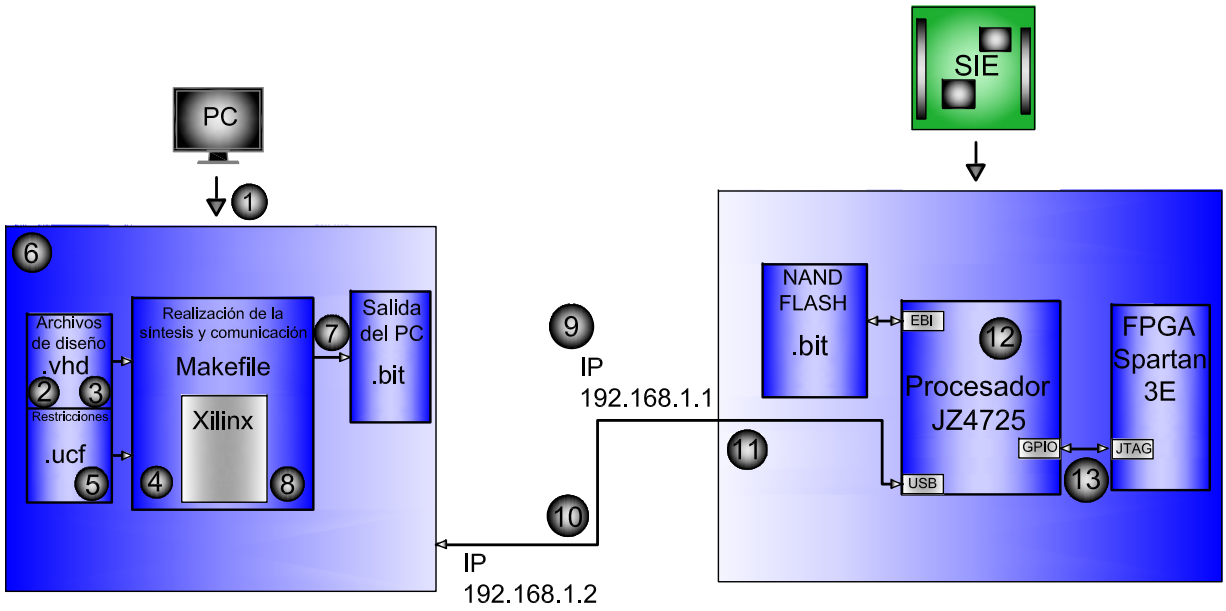


Figura 2.2: Esquema del proceso de configuración del FPGA de la tarjeta SIE para implementar un circuito digital

FUENTE:Los autores

IP tanto a la SIE como al PC que envia la información. Cuando el archivo de configuración *(.bit)* ha sido transmitido, se ingresa al procesador de la SIE y se utiliza la aplicación *xc3sprog* para configurar el FPGA. El esquema general del proceso a realizar se muestra en la figura 2.2, donde los números al interior de los círculos, representan los pasos a realizar en el transcurso de la práctica.

2.4.1 PROCEDIMIENTO PASO A PASO

A continuación se muestra el proceso detallado paso a paso necesario para la implementación de una compuerta AND en la tarjeta SIE. Es necesario seguir los pasos que se recomiendan para garantizar el éxito de la práctica.

2.4.1.1 Paso 1: Crear un directorio que contenga los ficheros a implementar en la tarjeta

Como sugerencia inicial se propone crear una carpeta en la cual se deben guardar todos los archivos que se necesiten para el respectivo desarrollo de la práctica, dicha carpeta debe estar contenida a su vez en una carpeta creada previamente por el ususario (Mi_carpeta). Para ello se puede utilizar el comando `mkdir` que permite crear un directorio o carpeta, en este caso se crearán en `home`, por lo tanto se procede a digitar en la terminal de linux:

```
1 $       mkdir ~/Mi_carpeta
2 $       cd ~/Mi_carpeta
3 $       mkdir and_dos
4 $       cd and_dos
```

Script 2.1: Código para crear un directorio.

El comando *cd*, permite acceder a un directorio previamente creado. Este directorio no necesariamente tiene que ser creado en `home`.

2.4.1.2 Paso 2: Crear el archivo de extensión .vhd

Dentro del directorio de trabajo creado en el paso 1 (and_dos), se procede a crear el archivo de nombre `and_dos.vhd`, digitando la línea mostrada en el *Script* 2.2.

```
$       gedit and_dos.vhd
```

Script 2.2: Código para crear el archivo.

Al comando del *Script* 2.2, se abre el editor de texto gedit, en esa interfaz se procede a digitar el archivo del paso 3. El archivo creado anteriormente es necesario para que el software ISE reconozca el código descrito en el lenguaje VHDL.

2.4.1.3 Paso 3: Descripción del código en el lenguaje VHDL

En el archivo `and_dos.vhd`, se procede a digitar el código que se muestra en el *Script* 2.3. (Por favor digitar, no copiar y pegar)

```
--Los comentarios en VHDL van seguidos mínimo de dos guiones--
-----------------------------------
--SE DECLARAN LAS LIBRERIAS---------
-----------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
-----------------------------------
--SE DECLARAN ENTRADAS Y SALIDAS-----
-----------------------------------
ENTITY and_dos IS
PORT ( a : IN  STD_LOGIC;
       b : IN  STD_LOGIC;
       q: OUT STD_LOGIC);
END and_dos;
-----------------------------------
--SE DESCRIBE EL COMPORTAMIENTO------
-----------------------------------
ARCHITECTURE arreglo OF and_dos IS
BEGIN
    q <= (a and b);
END arreglo;
-----------------------------------
```

Script 2.3: Código de una descripción VHDL de una compuerta AND.

Seguido de esto se procede a guardar los cambios y cerrar el editor de texto (gedit), verificando que el archivo creado se encuentre ubicado en la carpeta de trabajo creada en el paso 1 (and_dos).

2.4.1.4 Paso 4: Copiar el archivo Makefile en la carpeta de trabajo

Este archivo será suministrado por el profesor de la clase, ya que se tienen que cumplir ciertas reglas para editar este fichero. Este archivo debe ser pegado en el directorio de trabajo `and_dos` previamente creado en el paso 1. La única modificación necesaria para este archivo se hace en el segundo renglón, el cual comienza con la palabra `DESIGN`, tal como se muestra en el *Script* 2.5, en el que a su vez se coloca el nombre del *fichero.vhd* a implementar (`and_dos`) (Nota: En este punto no se recomienda realizar ningún otro cambio al Makefile.). Para editar dicha segunda línea del `Makefile` se procede a digitar la línea del *Script* 2.4.

Cabe mencionar que para el `Makefile` mostrado en el *Script* 2.5 se encuentra la segunda línea editada con la palabra `MI_DISEÑO`, luego es necesario reemplazarla con el mismo nombre que lleva el diseño a implementar, para este caso and_dos. Nuevamente guardar cambios y cerrar el editor de texto gedit.

```
1 gedit Makefile
```

Script 2.4: Línea para editar el Makefile.

Este Makefile puede funcionar con otras tarjetas haciendo un breve cambio, haciendo una modificación en la línea del Makefile que comienzan con la palabra `DEVICE`. El Makefile es utilizado para facilitar el proceso de síntesis con ISE Xilinx. Es preciso mencionar que dentro de un Makefile, todo texto que esté después del signo **#** es tomado como un comentario.

```
1  # Definiciones del proyecto: Nombres, tarjeta, pines, etc..
2  DESIGN          = and_dos# Nombre del circuito a implementar
3  PINS            = $(DESIGN).ucf# Nombre del Archivo para la asignación de pines
4  DEVICE          = xc3s100e-VQ100-4 # Referencia del FPGA
5  BGFLAGS         = -g TdoPin:PULLNONE -g DonePin:PULLUP \
6                    -g CRC:enable -g StartUpClk:CCLK
7
8  SIE_IP         = 192.168.1.1
9
10 # Definiciones de las fuentes .vhd
11 SRC_HDL         = $(DESIGN).vhd #Fuentes vhd
12 # Objetivos del Makefile
13 all:            bits
14
15 remake:         clean-build all
16
17 clean:
18         rm -f *~ */*~ a.out *.log *.key *.edf *.ps trace.dat
19
20 clean-build: clean
21         rm -rf build
22
23 cleanall:   clean
24         rm -rf build $(DESIGN).bit download.cmd
25
26 bits:           $(DESIGN).bit
27
28 #
```

```
29 # Synthesis
30 #
31 build/project.src:
32         @[ -d build ] || mkdir build
33         @rm -f $@
34         for i in $(SRC); do echo verilog work ../$$i >> $@; done
35         for i in $(SRC_HDL); do echo VHDL work ../$$i >> $@; done
36
37 build/project.xst: build/project.src
38         echo "run" > $@
39         echo "-top $(DESIGN) " >> $@
40         echo "-p $(DEVICE)" >> $@
41         echo "-opt_mode Area" >> $@
42         echo "-opt_level 1" >> $@
43         echo "-ifn project.src" >> $@
44         echo "-ifmt mixed" >> $@
45         echo "-ofn project.ngc" >> $@
46         echo "-ofmt NGC" >> $@
47         echo "-rtlview yes" >> $@
48 #Realiza el proceso de sintesis
49 build/project.ngc: build/project.xst $(SRC_HDL)
50         cd build && xst -ifn project.xst -ofn project.log
51 #Inicia la implementacion del diseno
52 #Realiza el proceso de translate
53 build/project.ngd: build/project.ngc $(PINS)
54         cd build && ngdbuild -p $(DEVICE) project.ngc -uc ../$(PINS)
55
56 #Realiza el proceso de Map
57 build/project.ncd: build/project.ngd
58         cd build && map -pr b -p $(DEVICE) project
59
60 #Realiza el proceso de Place & route
61 build/project_r.ncd: build/project.ncd
62         cd build && par -w project project_r.ncd
63
64 build/project_r.twr: build/project_r.ncd
65         cd build && trce -v 25 project_r.ncd project.pcf
66 # Finaliza proceso de implementacion
67
68 #Realiza el proceso de Generate programing file
69 $(DESIGN).bit:  build/project_r.ncd build/project_r.twr
70         cd build && bitgen project_r.ncd -l -w $(BGFLAGS)
71         @mv -f build/project_r.bit $@
72
73 download.cmd: $(DESIGN).bit
74         echo "setMode -bscan" > $@
75         echo "setCable -p auto" >> $@
76         echo "identify" >> $@
77         if [ $(DEVICE) = "xc3s500E-FG320-4" ] ;
78         then echo "assignfile -p 1 -file $(DESIGN).bit" >> $@ ;
79         else echo "assignfile -p 3 -file $(DESIGNE).bit" >> $@ ; fi
80         if [ $(DEVICE) = "xc3s500E-FG320-4" ] ; then echo "program -p 1" >> $@ ;
81         else echo "program -p 3" >> $@ ; fi
```

```
82          echo "quit" >> $@
83
84 #Realiza el proceso de configuracion de la tarjeta
85 download: $(DESIGN).bit download.cmd
86          impact -batch download.cmd
87
88 #Realiza el proceso de envio del archivo de configuracion a la tarjeta SAKC
89 upload: $(DESIGN).bit
90          scp $(DESIGN).bit root@$(SIE_IP):
```

Script 2.5: Código del Makefile.

Para correr exitosamente el `Makefile` se debe verificar que el nombre asignado al archivo es **"Makefile"**.

2.4.1.5 Paso 5: Asignación de pines creando el archivo de extensión .ucf

Para este paso, es preciso crear un archivo de extension `.ucf`, donde se hará la respectiva asignación de pines. Para ello se crea un archivo con el mismo nombre del circuito a implementar con extensión `ucf`, para este caso `and_dos.ucf`, para lo cual se procede a digitar la línea mostrada en el *Script* 2.6.

```
1 $         gedit and_dos.ucf
```

Script 2.6: Creando el archivo de asignación de pines.

Ahora se procede a digitar las líneas que aparecen en el *Script* 2.7, asignando los respectivos pines a las dos entradas y a la única salida que se tiene en la descripción de la compuerta AND realizada.

```
1 net a loc="P13"; # pulsador 1
2 net b loc="P30"; # pulsador 2
3 net q loc="P44"; # led D5
```

Script 2.7: Archivo ucf de la compuerta AND.

Digitado lo anterior se procede a guardar cambios y cerrar el editor de texto gedit.

2.4.1.6 Paso 6: Comprobar la existencia de los archivos necesarios para sintetizar e implementar

Con el objetivo de verificar que se tienen los archivos necesarios para poder sintetizar e implementar el código descrito en VHDL, se procede a digitar la línea mostrada en el *Script* 2.8,

```
1 $         ls
```

Script 2.8: Comprobando existencia de archivos.

Digitado este comando deben aparecer los tres archivos necesarios para una implementación exitosa, tal como se muestra en el *Script* 2.9.

```
1           and_dos.ucf  and_dos.vhd  Makefile
```

Script 2.9: Comprobando existencia de archivos.

> Nota : Si no aparece alguno de los tres archivos, es necesario hacer una revisión de los pasos anteriores.

2.4.1.7 Paso 7: Realizar la síntesis de la descripción

Ahora bien, es necesario llevar a cabo el proceso de síntesis de la descripción del código de la compuerta AND realizada, para lo cual se requiere digitar una serie de comandos algo complejos, razón por la cual se hace uso del Makefile donde internamente se encuentran dichos comandos necesarios para la respectiva síntesis.

Para empezar a ejecutar el Makefile, es necesario permanecer en la carpeta donde se encuentra el archivo Makefile. Cumpliendo con lo dicho anteriormente se procede a digitar la línea mostrada en el *Script* 2.10.

```
1 $       make
```

Script 2.10: Ejecutando el Makefile.

La anterior línea tendrá como consecuencia llevar a cabo la síntesis del código que está escrito en el archivo `and_dos.vhd`, además de crear un archivo dentro de la carpeta de trabajo de extensión `.bit`, que en este caso hace referencia al archivo `and_dos.bit`. Si no se produce ningún error en la ejecución del Makefile, al final aparece un mensaje **"Bitstream generation is complete"**. Seguido de esto se sugiere hacer uso del comando de linux `ls`, para verificar que efectivamente se creó el archivo mencionado anteriormente (`and_dos.bit`). Si se produce algún error, mirar el reporte para realizar su respectiva corrección y continuar con el paso 8. Si no se generó ningún error, omitir el paso 8 y continuar con el paso 9.

2.4.1.8 Paso 8: Problemas de síntesis [opcional]

En algunas oportunidades el proceso de síntesis presentará algunos errores, dado este caso, es preciso corregir los respectivos errores y sintetizar nuevamente.

Antes de realizar el proceso de síntesis por segunda vez, se recomienda borrar todos los ficheros que fueron creados en el primer proceso de síntesis que presento errores, para lo cual se digita la línea que aparece en el *Script* 2.11.

```
1 $       make cleanall
```

Script 2.11: Borrar proceso de síntesis.

Luego de esto, ya corregidos los errores se continua con el paso 7.

2.4.1.9 Paso 9: Conectar la tarjeta SIE

Para conectar la tarjeta SIE se hace uso del cable USB que viene con la misma, el cual se conecta en cualquiera de los puertos USB de la máquina en la cual se esté trabajando, como también se utiliza el adaptador para evitar daños en la tarjeta. Para saber si la tarjeta se encuentra en óptimas condiciones de trabajo, verificar que al momento de realizar

la conexión, se encienda el led marcado con la referencia D7 de la tarjeta SIE. Si esto no sucede, comunicarlo al profesor de la clase. Para verificar si el computador reconoció la tarjeta SIE, se abré una nueva terminal de linux y se digita el comando *ifconfig*, para lo cual debe aparecer el resultado que se muestra en el *Script* 2.12.

```
1 $       ifconfig
2
3 usb0    link encap:Ethernet Hwaddr XX:XX:XX:XX:XX:XX
4         inet6 addr: ffff::ffff:ffff:fff:ff22/64 Scope:Link
5         UP BRODCAST MULTICAST MUT:1500 Metric:1
6         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
7         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
8         collisions:0 txqueuelen:1000
9         RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Script 2.12: Verificando conexión.

2.4.1.10 Paso 10: Configuración del puerto USB

La configuración del puerto USB es necesaria para que dicho puerto sirva de interfaz con la tarjeta SIE. Para hacer la correspondiente configuración se procede a abrir una nueva terminal de linux, de modo que ahora se tienen dos terminales de trabajo, la primera, en donde se ha venido trabajando, sirve de interfaz para tener acceso al procesador de la SIE y configurar el FPGA; la segunda sirve para configurar la comunicación de la tarjeta con el PC.

Entonces, en la segunda terminal, donde aparece la conexión usb0 del paso 9, se procede a digitar las líneas que se muestran en el *Script* 2.13, una vez hecho esto se procede a verificar que realmente existe conexión ó alguna comunicación con la tarjeta SIE, digitando en la misma terminal, la instrucción mostrada en el *Script* 2.14.

```
1 $       dmesg
2 $       sudo ifconfig usb0 192.168.1.2 up
```

Script 2.13: Configuración del puerto USB.

```
1 $       ping 192.168.1.1
```

Script 2.14: Envío y recepción de datos de prueba

Es posible saber si existe conexión entre el PC y la tarjeta, ya que estos dos se envían datos que se van registrando en la terminal una vez se digita la línea del *Script* 2.14. Seguido de esto se cancela el envío de datos de prueba de conexión, presionando las teclas **ctrl + c**.

Para saber si todavía existe comunicación exitosa entre el PC y la tarjeta, basta con digitar en cualquiera de las dos terminales de linux el comando del *Script* 2.15,

```
1 $       ifconfig
2
3 usb0    link encap:Ethernet Hwaddr XX:XX:XX:XX:XX:XX
4         inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
5         inet6 addr: ffff::ffff:ffff:fff:ff22/64 Scope:Link
6         UP BRODCAST MULTICAST MUT:1500 Metric:1
```

```
7           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
8           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
9           collisions:0 txqueuelen:1000
10          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Script 2.15: Verificando conexión y configuración de IP.

Nótese que en la cuarta linea, debe aparecer la dirección IP que le fue asignada en la configuración del puerto usb, si no aparece no existe conexión entre el PC y la tarjeta SIE.

2.4.1.11 Paso 11: Enviar información al procesador

Después de un proceso de síntesis exitoso el cual no presentó errores y una correcta configuración del puerto usb, es preciso enviar el archivo creado en el paso 7 (and_dos.bit) al procesador de la SIE, el cual se encuentra en el directorio de trabajo.

Desde la primera terminal, la cual tiene como dirección actual la carpeta de trabajo, se digita el comando necesario para enviar el archivo de configuración (and_dos.bit) a la memoria NAND de la SIE. tal como se muestra en el *Script* 2.16.

```
1 $         make upload
```

Script 2.16: Enviando información al procesador.

2.4.1.12 Paso 12: Acceder al procesador de la tarjeta

Hasta este punto se ha enviado la información al procesador de la tarjeta, ahora es preciso entrar a dicho procesador y enviar el archivo `and_dos.bit` al FPGA.

Así pues, garantizando la respectiva conexión con la tarjeta se procede a digitar en la primera terminal, la línea mostrada en el *Script* 2.17.

```
1 $         ssh root@192.168.1.1
```

Script 2.17: Accediendo al procesador.

Después de digitar la línea anterior, la tarjeta pide una clave de acceso la cual hace referencia al password de la tarjeta SIE, y que a su vez ha sido establecida con la tecla `enter`. Así pues, después de presionar la tecla `enter` se ha ingresado al procesador de la tarjeta SIE.

2.4.1.13 Paso 13: Enviar información al FPGA

Ya estando en el procesador de la tarjeta SIE, se procede a enviar el archivo que lleva la información, es decir `and_dos.bit` al FPGA, para lo cual es necesario digitar en la terminal de trabajo, la línea mostrada en el *Script* 2.18.

```
1 $       xc3sprog and_dos.bit
```

Script 2.18: Enviando información al FPGA.

Ya enviando la información al FPGA, se procede a la manipulación de los *switches* asignados y la verificación de la correspondiente salida visualizando el LED que se determino como salida (LED D5 de la SIE).

2.5 RESUMEN

En la presente práctica se hace un planteamiento detallado del proceso que se debe llevar a cabo a la hora de implementar una descripción en VHDL sobre la tarjeta SIE, resaltando tres procesos principales:

- Archivos necesarios para la implementación sobre la plataforma SIE
- Comunicación con la tarjeta SIE a través del protocolo ssh
- Interacción con la tarjeta SIE

2.6 PREGUNTAS DE PRUEBA

- ¿Cuáles son los archivos necesarios para realizar una implementación sobre la tarjeta SIE?
- ¿Por medio de qué protocolo se hace la comunicación con la tarjeta SIE?
- ¿Finalmente a que parte de la plataforma SIE debe llegar la descripción en VHDL?
- ¿Porqué sin oprimir ningún pulsador el led se encuentra prendido inicialmente?

2.7 EJERCICIO PROPUESTO

En la figura 2.3 se muestra la representación de la compuerta OR y en la figura 2.4 su respectiva tabla de verdad. La puerta OR es otra de las compuertas básicas con las que se construyen todas las funciones lógicas. Una puerta OR puede tener dos o más entradas y realiza la operación que se conoce como suma lógica.[4]

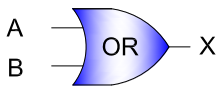


Figura 2.3: Símbolo distintivo de una puerta OR.

FUENTE:Los autores

ENTRADA A	ENTRADA B	SALIDA X
0	0	0
0	1	1
1	0	1
1	1	1

Figura 2.4: Tabla de verdad de una compuerta OR

FUENTE:Los autores

Realizar los pasos necesarios para implementar una puerta OR en el FPGA de la tarjeta SIE.

LABORATORIO **3**

Display de siete segmentos implementado con ecuaciones estandar.

Aprender es el primer paso, vivir no es más que el segundo.

VICTOR HUGO

3.1 INTRODUCCIÓN

VHDL es un lenguaje de descripción de circuitos electrónicos digitales que utiliza distintos niveles de abstracción. El significado de las siglas VHDL es VHSIC(Very high Speed Integrated Circuits) + HDL (Hardware Description Languaje). Esto significa que VHDL permite acelerar el proceso de diseño [2]. Es importante tener claro que VHDL no es un lenguaje de programación sino un lenguaje de descripción de hardware, con el cual es posible describir circuitos síncronos y asíncronos. Para poder realizar una descripción de hardware es de vital importancia saber los componentes principales de la misma, así como las diversas formas que existen de realizarla, estando dentro de este grupo una de las mas básicas que hace referencia a la descripción por medio de funciones lógicas. Así pues, es de gran importancia tener presente el proceso a seguir, al momento de implementar un circuito combinacional por medio de ecuaciones lógicas, utilizando VHDL.

3.2 OBJETIVOS

- Identificar las partes principales de una descripción en VHDL.
- Describir un circuito combinacional mediante ecuaciones lógicas.

3.3 MARCO TEÓRICO

El display de 7 segmentos es actualmente usado en gran variedad de aplicaciones de la industria electrónica debido a la simplicidad de su manejo, utilizado principalmente para la representación de números en muchos dispositivos.

3.3.1 DISPLAY DE SIETE SEGMENTOS

El modo de funcionamiento de un display consiste en la aplicación de un voltaje en un diodo para que este emita una señal de luz, generalmente de color rojo debido a su fácil visualización, así reorganizando todos los diodos en un arreglo como el de la figura 3.1, se puede mostrar la representación de un sistema numérico, para este caso el decimal, así eligiendo que los diodos energizados sean todos, es decir a,b,c,d,e,f y g, se obtendrá la representación del número 8, siendo los diodos energizados el b y el c, se representara el número 1, siendo los diodos energizados el a,b,d,e y g, se obtendrá la representación del número 2.

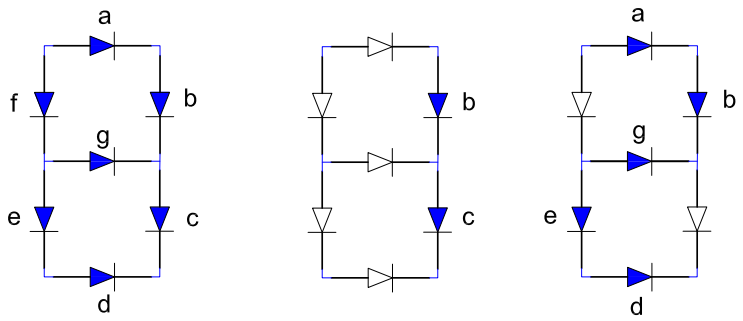


Figura 3.1: Reorganización de diodos.

FUENTE:Los autores

En el mercado se pueden encontrar dos tipos de display, el display 7 segmentos de ánodo común y el display 7 segmentos de cátodo común. El primero mostrado en la figura 3.2 se configura conectando el ánodo de cada uno de los diodos a una fuente de alimentación positiva.

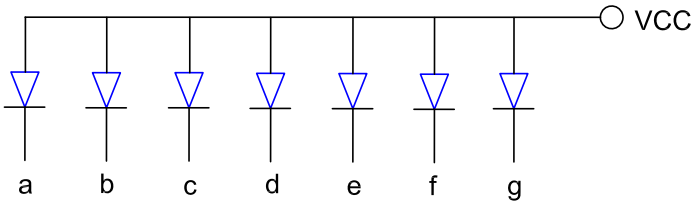


Figura 3.2: Display de ánodo común.

FUENTE:Los autores

Para el display de 7 segmentos de ánodo común se activa cualquiera de los diodos conectando el respectivo cátodo a tierra por medio de una resistencia con el fin de limitar la corriente que pueda pasar por el diodo y evitar

daños en el mismo [3].

Para el display de 7 segmentos de cátodo común, mostrado en la figura 3.3, es necesario conectar todos los cátodos a tierra, teniendo como consecuencia la activación de cada uno de los diodos a través de la conexión del respectivo ánodo a la fuente de alimentación por medio de una resistencia que cumplirá la misma función de limitar la corriente para proteger el diodo [3].

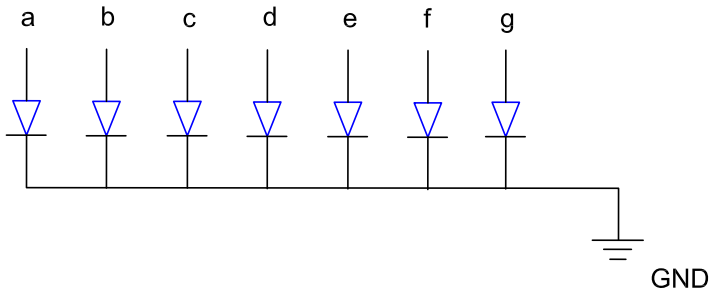


Figura 3.3: Display de cátodo común.

FUENTE:Los autores

3.3.2 DECODIFICADORES

La función básica de un decodificador es detectar la presencia de una determinada combinación de bits (código) en sus entradas y señalar la presencia de este código mediante un cierto nivel de salida. En su forma más general, un decodificador posee líneas de entrada para gestionar n bits, y en una de las 2^n líneas de salida indica la presencia de una o más combinaciones de n bits [4].

3.4 PROCEDIMIENTO GENERAL

Se propone la representación de los números binarios en decimales, para esto se requiere hacer un decodificador que tiene como dato de entrada un código BCD y proporciona salidas capaces de alimentar un display de 7 segmentos para indicar un dígito decimal. Para dicho procedimiento se recomienda seguir los siguientes pasos:

1. Identificación de entradas y salidas
2. Tabla de verdad
3. Expresar por Suma de Productos
4. Descripción del código en VHDL
5. Simulación
6. Identificación de dispositivos a utilizar

7. Asignación de pines creando el archivo de extensión .ucf

8. Copiar el archivo Makefile en la carpeta de trabajo

9. Implementación en la tarjeta SIE

3.4.1 PROCEDIMIENTO PASO A PASO

Teniendo una idea general de las especificaciones del problema y de los pasos a seguir, se procede a realizar un análisis detallado del mismo.

3.4.1.1 Paso 1: Identificación de entradas y salidas

En la caja representativa de la figura 3.4, se tienen 4 entradas A0,A1,A2 y A3; tomando como bit menos significativo A0 y como bit más significativo A3, los cuales corresponden al código BCD, y la combinación de estos bits, darán como resultado una salida indicada por los 7 bits o 7 segmentos denotados por las letras a,b,c,d,e,f y g, que corresponden a los 7 segmentos del display.

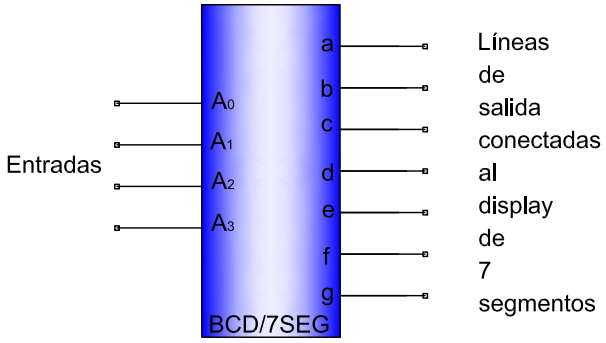


Figura 3.4: Caja representativa de un decodificador BCD/7SEG.

FUENTE:Los autores

3.4.1.2 Paso 2: Tabla de verdad

Ahora bien, sabiendo con exactitud las entradas y correspondientes salidas que manejará el circuito, es preciso organizar dicha información que se tiene hasta el momento en una tabla de verdad.

Asi pues, en la tabla 3.1, se puede notar detalladamente el comportamiento lógico de la salida del decodificador representado por cada uno de los siete segmentos denotados por las letras a, b, c, d, e, f y g, salida que depende del valor lógico asignado a cada una de las cuatro entradas que se denotan como A_3, A_2, A_1 y A_0.

	ENTRADAS				SALIDAS						
DECIMAL	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Tabla 3.1: Tabla de verdad del decodificador BCD/7SEG

3.4.1.3 Paso 3: Expresar por Suma de Productos.

Cuando se tiene una tabla de verdad, es posible expresar la salida de un circuito, que en este caso representa la salida del decodificador en función de las entradas, luego es preciso expresar cada uno de los segmentos en función de las entradas A_3, A_2, A_1 y A_0.

Una forma de llevar a cabo este proceso es verificando la tabla de verdad y respecto a esta hacer la correspondiente suma de productos de cada uno de los siete segmentos, el resultado de dicho proceso se muestra en las ecuaciones 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, y 3.7.

$$\begin{aligned} a &= (\overline{A_3} * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (\overline{A_3} * \overline{A_2} * A_1 * \overline{A_0}) + (\overline{A_3} * \overline{A_2} * A_1 * A_0) + (\overline{A_3} * A_2 * \overline{A_1} * A_0) \\ &+ (\overline{A_3} * A_2 * A_1 * \overline{A_0}) + (\overline{A_3} * A_2 * A_1 * A_0) + (A_3 * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (A_3 * \overline{A_2} * \overline{A_1} * A_0) \end{aligned} \quad (3.1)$$

$$\begin{aligned} b &= (\overline{A_3} * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (\overline{A_3} * \overline{A_2} * \overline{A_1} * A_0) + (\overline{A_3} * \overline{A_2} * A_1 * \overline{A_0}) + (\overline{A_3} * \overline{A_2} * A_1 * A_0) \\ &+ (\overline{A_3} * A_2 * \overline{A_1} * \overline{A_0}) + (\overline{A_3} * A_2 * A_1 * A_0) + (A_3 * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (A_3 * \overline{A_2} * \overline{A_1} * A_0) \end{aligned} \quad (3.2)$$

$$\begin{aligned} c &= (\overline{A_3} * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (\overline{A_3} * \overline{A_2} * \overline{A_1} * A_0) + (\overline{A_3} * \overline{A_2} * A_1 * A_0) + (\overline{A_3} * A_2 * \overline{A_1} * \overline{A_0}) \\ &+ (\overline{A_3} * A_2 * \overline{A_1} * A_0) + (\overline{A_3} * A_2 * A_1 * \overline{A_0}) + (\overline{A_3} * A_2 * A_1 * A_0) + (A_3 * \overline{A_2} * \overline{A_1} * \overline{A_0}) \\ &+ (A_3 * \overline{A_2} * \overline{A_1} * A_0) \end{aligned} \quad (3.3)$$

$$\begin{aligned} d &= (\overline{A_3} * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (\overline{A_3} * \overline{A_2} * A_1 * \overline{A_0}) + (\overline{A_3} * \overline{A_2} * A_1 * A_0) + (\overline{A_3} * A_2 * \overline{A_1} * A_0) \\ &+ (\overline{A_3} * A_2 * A_1 * \overline{A_0}) + (A_3 * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (A_3 * \overline{A_2} * \overline{A_1} * A_0) \end{aligned} \quad (3.4)$$

$$e = (\overline{A_3} * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (\overline{A_3} * \overline{A_2} * A_1 * \overline{A_0}) + (\overline{A_3} * A_2 * A_1 * \overline{A_0}) + (A_3 * \overline{A_2} * \overline{A_1} * \overline{A_0}) \quad (3.5)$$

$$\begin{aligned} f &= (\overline{A_3} * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (\overline{A_3} * A_2 * \overline{A_1} * \overline{A_0}) + (\overline{A_3} * A_2 * \overline{A_1} * A_0) + (\overline{A_3} * A_2 * A_1 * \overline{A_0}) \\ &+ (A_3 * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (A_3 * \overline{A_2} * \overline{A_1} * A_0) \end{aligned} \quad (3.6)$$

$$\begin{aligned} g &= (\overline{A_3} * \overline{A_2} * A_1 * \overline{A_0}) + (\overline{A_3} * \overline{A_2} * A_1 * A_0) + (\overline{A_3} * A_2 * \overline{A_1} * \overline{A_0}) + (\overline{A_3} * A_2 * \overline{A_1} * A_0) \\ &+ (\overline{A_3} * A_2 * A_1 * \overline{A_0}) + (A_3 * \overline{A_2} * \overline{A_1} * \overline{A_0}) + (A_3 * \overline{A_2} * \overline{A_1} * A_0) \end{aligned} \quad (3.7)$$

3.4.1.4 Paso 4: Descripción del código en VHDL

A partir de la Suma de Productos es preciso llevar a cabo la descripción del código en VHDL, pues hasta este punto se ha identificado el comportamiento de las salidas del circuito en función de las entradas. Así pues, la descripción del código debe llevar tres procesos principales: declaración de las librerias a utilizar, declaración de entradas y salidas, y la descripción de cada una de las salidas en función de las correspondientes entradas. Es así como en el Script 3.1 se lleva a cabo la respectiva dsescripción.

```
1  ----------SE DECLARAN LAS LIBRERIAS----------
2  ------------------------------------------------------------------------------
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use IEEE.STD_LOGIC_ARITH.ALL;
6  use IEEE.STD_LOGIC_UNSIGNED.ALL;
7  ----------SE DECLARAN ENTRADAS Y SALIDAS----------
8  ------------------------------------------------------------------------------
9  entity bcd_a_7_seg_largo is
10
11 Port ( A3 : in  STD_LOGIC;
12           A2 : in  STD_LOGIC;
13           A1 : in  STD_LOGIC;
14           A0 : in  STD_LOGIC;
15           a : out  STD_LOGIC;
16           b : out  STD_LOGIC;
17           c : out  STD_LOGIC;
18           d : out  STD_LOGIC;
19           e : out  STD_LOGIC;
20           f : out  STD_LOGIC;
21           g : out  STD_LOGIC);
22
23 end bcd_a_7_seg_largo;
24
25 ----------SE DESCRIBE CADA UNA DE LAS ECUACIONES EN FUNCION DE LAS COMPUERTAS----------
26 ------------------------------------------------------------------------------
```

```
27
28 architecture Behavioral of bcd_a_7_seg_largo is
29
30 begin
31
32 a<=((not A3)and(not A2)and(not A1)and(not A0))or((not A3)and(not A2)and(A1)and(not A0))
33     or((not A3)and(not A2)and(A1)and(A0))or((not A3)and(A2)and(not A1)and(A0))
34     or((not A3)and(A2)and(A1)and(not A0))or((not A3)and(A2)and(A1)and(A0))
35     or((A3)and(not A2)and(not A1)and(not A0))or((A3)and(not A2)and(not A1)and(A0));
36
37 b<=((not A3)and(not A2)and(not A1)and(not A0))or((not A3)and(not A2)and(not A1)and(A0))
38     or((not A3)and(not A2)and(A1)and(not A0))or((not A3)and(not A2)and(A1)and(A0))
39     or((not A3)and(A2)and(not A1)and(not A0))or((not A3)and(A2)and(A1)and(A0))
40     or((A3)and(not A2)and(not A1)and(not A0))or((A3)and(not A2)and(not A1)and(A0));
41
42 c<=((not A3)and(not A2)and(not A1)and(not A0))or((not A3)and(not A2)and(not A1)and(A0))
43     or((not A3)and(not A2)and(A1)and(A0))or((not A3)and(A2)and(not A1)and(not A0))
44     or((not A3)and(A2)and(not A1)and(A0))or((not A3)and(A2)and(A1)and(not A0))
45     or((not A3)and(A2)and(A1)and(A0))or((A3)and(not A2)and(not A1)and(not A0))
46     or((A3)and(not A2)and(not A1)and(A0));
47
48 d<=((not A3)and(not A2)and(not A1)and(not A0))or((not A3)and(not A2)and(A1)and(not A0))
49     or((not A3)and(not A2)and(A1)and(A0))or((not A3)and(A2)and(not A1)and(A0))
50     or((not A3)and(A2)and(A1)and(not A0))or((A3)and(not A2)and(not A1)and(not A0))
51     or((A3)and(not A2)and(not A1)and(A0));
52
53 e<=((not A3)and(not A2)and(not A1)and(not A0))or((not A3)and(not A2)and(A1)and(not A0))
54     or((not A3)and(A2)and(A1)and(not A0))or((A3)and(not A2)and(not A1)and(not A0));
55
56 f<=((not A3)and(not A2)and(not A1)and(not A0))or((not A3)and(A2)and(not A1)and(not A0))
57     or((not A3)and(A2)and(not A1)and(A0))or((not A3)and(A2)and(A1)and(not A0))
58     or((A3)and(not A2)and(not A1)and(not A0))or((A3)and(not A2)and(not A1)and(A0));
59
60 g<=((not A3)and(not A2)and(A1)and(not A0))or((not A3)and(not A2)and(A1)and(A0))
61     or((not A3)and(A2)and(not A1)and(not A0))or((not A3)and(A2)and(not A1)and(A0))
62     or((not A3)and(A2)and(A1)and(not A0))or((A3)and(not A2)and(not A1)and(not A0))
63     or((A3)and(not A2)and(not A1)and(A0));
64
65 end Behavioral;
```

Script 3.1: Código de una descripción VHDL de la compuerta AND.

3.4.1.5 Paso 5: Simulación

Antes de realizar la implementación de cualquier descripción, es preciso asegurarse de los resultado esperados por medio de una simulación. Así pues en la figura 3.5 se muestra la respectiva simulación del decodificador a implementar.

En la simulación mostrada en la figura 3.5, se nota el marcador en aproximadamente 250 ns, donde se muestra que a una entrada $a3 = 0, a2 = 0, a1 = 0, a0 = 1$ correspondiente al número decimal 1, el circuito responde

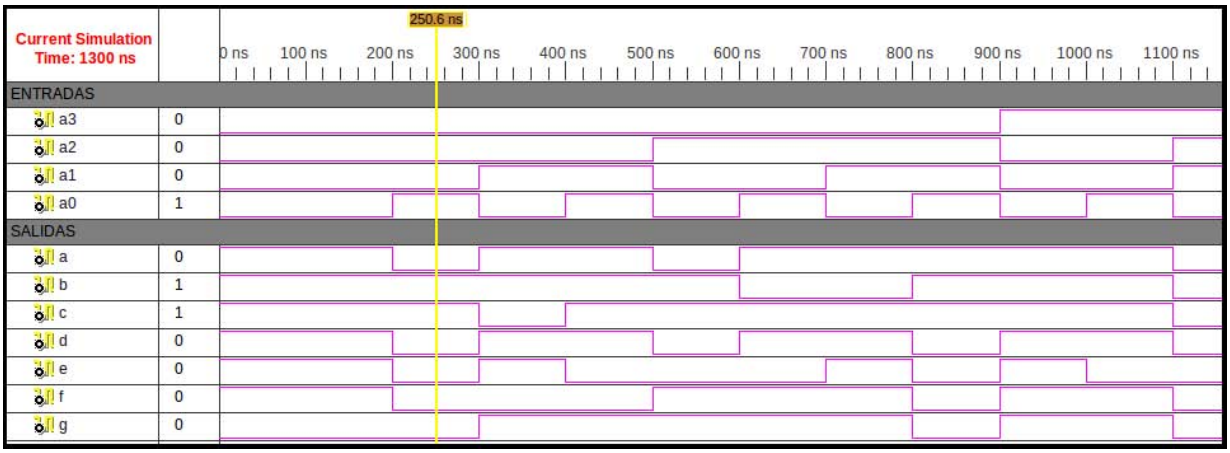


Figura 3.5: Simulación del decodificador descrito con ecuaciones lógicas

FUENTE:Herramienta de simulación ISE

con una salida $a = 0, b = 1, c = 1, d = 0, e = 0, f = 0, g = 0$, correspondiente a los segmentos del display y mostrando el comportamiento esperado, luego ahora sí es posible pasar al siguiente paso.

3.4.1.6 Paso 6: Identificación de dispositivos a utilizar

A continuación se debe hacer un análisis del número de entradas y salidas que requiere el circuito, y de acuerdo a esto, listar los dispositivos que se requieren para llevar a cabo la respectiva implementación. Para el presente circuito es necesario disponer de 4 entradas, por lo que se hace necesaria la presencia de un módulo que tenga a disposición cuatro botones o switches manipulables. Ahora bien, para mostrar la salida es necesario hacer uso de un display de 7 segmentos. Asi pues los dispositivos necesarios a utilizar son:

- Digilent PmodSWT Switch Module Board [8]
- Digilent PmodSSD Peripheral Module Board [8]

Es necesario referirse a la hoja de datos de cada uno de estos dispositivos para entender su funcionamiento y poder hacer una óptima elección de pines para llevar a cabo un correcto proceso de implementación del mismo.

3.4.1.7 Paso 7: Asignación de pines creando el archivo de extensión .ucf

Ya habiendo estudiado las hojas de datos de cada uno de los dispositivos a utilizar, es preciso crear el archivo necesario para llevar a cabo la asignación de los pines del circuito, este archivo se muestra en el Script 3.2.

```
1 net A3 loc="P54"; # pulsador switch 1
2 net A2 loc="P58"; # pulsador switch 2
3 net A1 loc="P61"; # pulsador switch 3
```

```
4  net A0 loc="P63"; # pulsador switch 4
5  net a loc="P33"; # segmento a del display
6  net b loc="P27"; # segmento b del display
7  net c loc="P24"; # segmento c del display
8  net d loc="P22"; # segmento d del display
9  net e loc="P49"; # segmento e del display
10 net f loc="P47"; # segmento f del display
11 net g loc="P40"; # segmento g del display
```

Script 3.2: Archivo ucf de ejemplo.

Cabe mencionar que la localización de cada uno de los pines se hace en base al diseño de una tarjeta de expansión de pines creada con el propósito de adecuar la tarjeta SIE a ciertas prácticas de Sistemas Digitales, por lo cual es preciso referirse al Manual de la tarjeta SIE al final del presente documento.

3.4.1.8 Paso 8: Copiar el archivo Makefile en la carpeta de trabajo

Para llevar a cabo el procedimiento necesario en el presente paso, es preciso referirse a la guia de laboratorio 1, en donde se expone de forma detallada todo lo referente al `Makefile`, el cual es un archivo proporcionado por el profesor de la clase y al cual se le debe editar haciendo una pequeña modificación en la segunda línea, donde debe agregarse el nombre del fichero que se desea implementar.

3.4.1.9 Paso 9: Implementación en la tarjeta SIE

Ya teniendo los archivos necesarios para la respectiva implementación, es preciso llevar a cabo el proceso necesario para un óptimo desarrollo, para lo cual es necesario realizar cada uno de los items mostrado a continuación:

1. Conectar la tarjeta SIE
2. Configuración del puerto USB
3. Realizar la sintesis de la descripción
4. Enviar información al procesador
5. Acceder al procesador de la tarjeta
6. Enviar información al FPGA

Es preciso mencionar que si se tiene alguna duda para llevar a cabo el proceso mencionado anteriormente, es necesario referirse a la guía de laboratorio 1 titulada *Implementación de una compuerta AND en la tarjeta SIE*, en donde se explica detalladamente cada uno de los anteriores.

3.5 RESUMEN

A través de la presente práctica se ha dado a conocer el proceso necesario para la implementación de un decodificador que tiene como entrada un dato en código BCD, y sus salidas muestran un dígito decimal en un display de siete segmentos que ha sido adaptado a la tarjeta SIE para su respectiva implementación. Se presentan como temas principales los siguientes:

- Partes principales de una descripción en VHDL
- Descripción en VHDL por medio de funciones lógicas
- Implementación en la tarjeta SIE

3.6 PREGUNTAS DE PRUEBA

- ¿Cuáles son las partes principales de una descripción en VHDL?
- ¿Qué sucede cuando se pone en la entrada el código 1100 y porqué pasa esto?

3.7 EJERCICIO PROPUESTO

Modificar el archivo que contiene la decripción del decodificador en VHDL, para que en el display de 7 segmentos, se representen los primeros quince números en codigo hexadecimal con ecuaciones estándar. La tabla 3.2 puede ser de ayuda para la respectiva representación.

DECIMAL		HEXADECIMAL	DECIMAL		HEXADECIMAL
0	$\rightarrow$	0	8	$\rightarrow$	8
1	$\rightarrow$	1	9	$\rightarrow$	9
2	$\rightarrow$	2	10	$\rightarrow$	A
3	$\rightarrow$	3	11	$\rightarrow$	B
4	$\rightarrow$	4	12	$\rightarrow$	C
5	$\rightarrow$	5	13	$\rightarrow$	D
6	$\rightarrow$	6	14	$\rightarrow$	E
7	$\rightarrow$	7	15	$\rightarrow$	F

Tabla 3.2: Tabla para representar números decimales en hexadecimal

LABORATORIO **4**

Display de siete segmentos implementado con ecuaciones simplificadas.

Haz de los obstáculos escalones para aquello
que quieres alcanzar.

CHARLES CHAPLIN

4.1 INTRODUCCIÓN

En el diseño de cualquier sistema, se busca satisfacer las necesidades de quién lo demande, donde la parte interesada buscará un sistema que cumpla con los requisitos establecidos de manera óptima, entendiéndose como parámetro de optimización, la menor utilización de recursos posible.

Así pues, el mapa de Karnaugh se utiliza para reducir expresiones booleanas a su mínima expresión. Una expresión suma de productos minimizada está formada por el mínimo número de términos producto posibles con el mínimo número de variables por término. Generalmente, una expresión suma de productos minimizada puede ser implementada mediante un número de puertas menor que su expresión estándar, lo cual constituye la finalidad del proceso de simplificación.[4]

4.2 OBJETIVOS

- Realizar simplificaciones de expresiones booleanas utilizando mapas de Karnaugh.

4.3 MARCO TEÓRICO

4.3.1 DECODIFICADORES

La función básica de un decodificador es detectar la presencia de una determinada combinación de bits (código) en sus entradas y señalar la presencia de este código mediante un cierto nivel de salida. En su forma mas general, un decodificador posee, líneas de entrada para gestionar n bits, y en una de las 2^n líneas de salida indica la presencia de una o mas combinaciones de n bits.[4]

4.3.2 MAPAS DE KARNAUGH

Un mapa de karnaugh proporciona un método sistemático de simplificación de expresiones booleanas y, si se aplica adecuadamente, genera las expresiones suma de productos y producto de sumas mas simples posibles. La efectividad de la simplificación algebráica depende de la familiaridad que se tenga con las leyes, reglas y teoremas del álgebra booleana y de la habilidad que se tenga a la hora de aplicarlas. Es preciso resaltar que el mapa de Karnaugh es considerado una "receta" para la simplificación.

Un mapa de karnaugh es similar a una tabla de verdad, ya que muestra todos los posibles valores de las variables de entrada y la salida resultante para cada valor, tan solo que en vez de estar organizada en filas y columnas como una tabla de verdad, el mapa de karnaugh es una secuencia de celdas en las que cada celda representa un valor binario de las variables de entrada. Las celdas se disponen de manera que la simplificación de una determinada expresión consiste en agrupar adecuadamente las celdas. Los mapas de Karnaugh son usualmente utilizados en expresiones de dos, tres, cuatro y cinco variables.

El número de celdas de un mapa de karnaugh es igual al número total de posibles combinaciones de las variables de entrada, al igual que el número de filas de una tabla de verdad. Para tres variables, el número de celdas necesarias es de $2^3 = 8$. Para cuatro variables, el numero de celdas es de $2^4 = 16$.[4]

4.4 PROCEDIMIENTO GENERAL

Se requiere la representación de los números binarios en decimales, para esto se requiere hacer un decodificador que tiene como dato de entrada un código BCD y proporciona salidas capaces de excitar un display de 7 segmentos para indicar un dígito decimal. Es preciso indicar las diversas salidas de la forma mas simplificada posible, con el firme propósito de utilizar el menor número de compuertas posibles, utilizando el método de simplificación conocido como Mapas de Karnaugh. Para dicho procedimiento se recomienda seguir los siguientes pasos:

- ☞ Identificación de entradas y salidas
- ☞ Tabla de verdad
- ☞ Organizar información en Mapas de karnaugh

- ☞ Expresar en Suma de Productos
- ☞ Descripción del código en VHDL
- ☞ Simulación
- ☞ Identificación de dispositivos a utilizar
- ☞ Asignación de pines
- ☞ Copiar el archivo Makefile en la carpeta de trabajo
- ☞ Implementación en la tarjeta SIE

4.4.1 PROCEDIMIENTO PASO A PASO

Teniendo una idea general de las especificaciones del problema y de los pasos a seguir, se procede a realizar un análisis detallado del mismo.

4.4.1.1 Paso 1: Identificación de entradas y salidas

En la caja representativa de la figura 4.1, se tienen 4 entradas A_0,A_1,A_2 y A_3 tomando como bit menos significativo A_0 y como bit más significativo A_3, los cuales corresponden al código BCD, y la combinación de estos bits, darán como resultado una salida indicada por los 7 bits o 7 segmentos denotados por las letras a,b,c,d,e,f y g, que corresponden a los 7 segmentos del display.

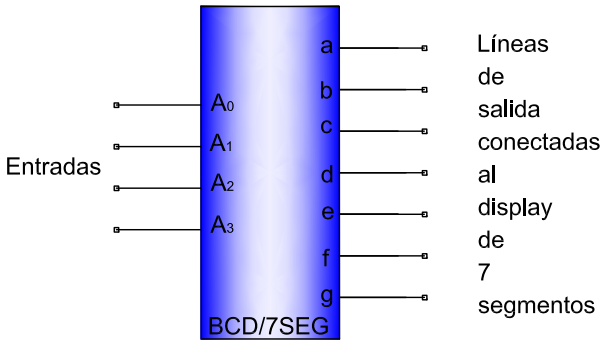


Figura 4.1: Caja representativa de un decodificador BCD/7SEG.

FUENTE:Los autores

4.4.1.2 Paso 2: Tabla de verdad

Ahora bien, sabiendo con exactitud las entradas y correspondientes salidas que manejará el circuito, es preciso organizar dicha información que se tiene hasta el momento en una tabla de verdad.

	ENTRADAS				SALIDAS						
DECIMAL	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Tabla 4.1: Tabla de verdad del decodificador BCD/7SEG

Así pues, en la tabla 4.1, se puede notar detalladamente el comportamiento lógico de la salida del decodificador representado por cada uno de los siete segmentos denotados por las letras a, b, c, d, e, f y g, salida que depende del valor lógico asignado a cada una de las cuatro entradas que se denotan como A_3, A_2, A_1 y A_0.

Si se quisiera realizar la Suma de Productos directamente de la tabla de verdad se obtendría un extenso arreglo de compuertas, luego es preciso buscar un método de simplificación que proporcione una solución más sencilla con el objetivo de reducir el número total de compuertas, el método a utilizar en la presente hace referencia a los mapas de karnaugh.

4.4.1.3 Paso 3: Organizar información en Mapas de karnaugh

Una vez se tiene la información de entradas y salidas organizadas en una tabla de verdad, es preciso pasar dicha información a un mapa de Karnaugh, dicho procedimiento se debe hacer para cada una de las salidas, es decir para cada uno de los segmentos del display que se desea implementar. Es por eso que en las figuras 4.2, 4.3, 4.4, 4.5, 4.6, 4.7 y 4.8 se encuentra cada uno de los Mapas de Karnaugh necesarios para poder expresar cada uno de los siete segmentos en función de las cuatro entradas, y de la forma mas simplificada posible.

Después de organizar la información en los mapas de Karnaugh, y realizar las diversas agrupaciones de `1s` ó `0s`, se procede a realizar una Suma de Productos ó un Producto de Sumas según se requiera, para este caso particular se agruparon los `1s` en cada uno de los mapas, luego se procede a expresarlo en Suma de Productos.

4.4.1.4 Paso 4: Expresar en Suma de Productos

Ahora bien, una vez ubicada la información en los diversos Mapas de Karnaugh, y habiendo hecho las diversas agrupaciones de `1s` necesarias para expresar cada uno de los segmentos como un Producto de Sumas de la manera más simplificada posible, se tiene como resultado las ecuaciones 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 y 4.7.

$$a = (\overline{A_2} * \overline{A_0}) + (A_2 * A_0) + A_1 + A_3 \tag{4.1}$$

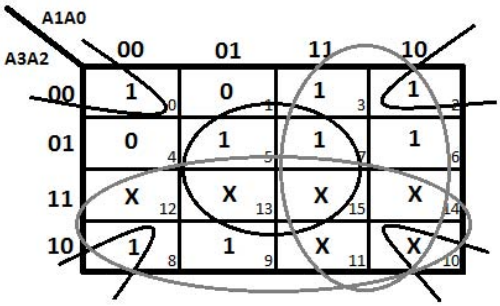


Figura 4.2: Mapa de Karnaugh para el segmento a

FUENTE:Los autores

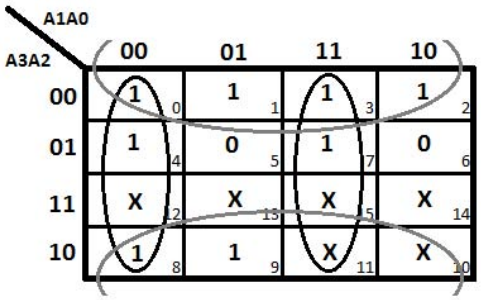


Figura 4.3: Mapa de Karnaugh para el segmento b

FUENTE:Los autores

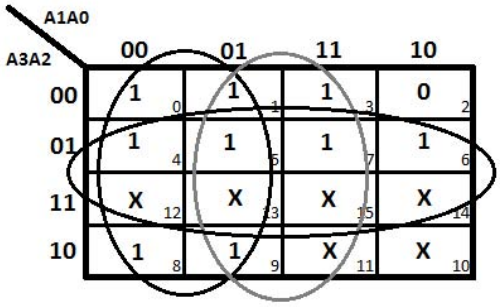


Figura 4.4: Mapa de Karnaugh para el segmento c

FUENTE:Los autores

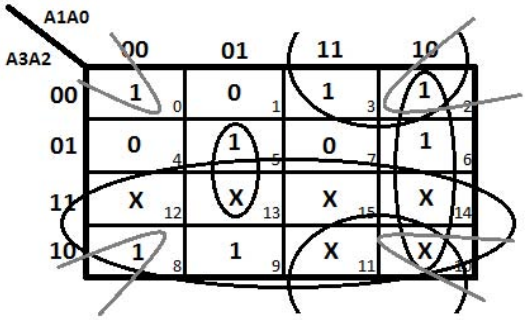


Figura 4.5: Mapa de Karnaugh para el segmento d

FUENTE:Los autores

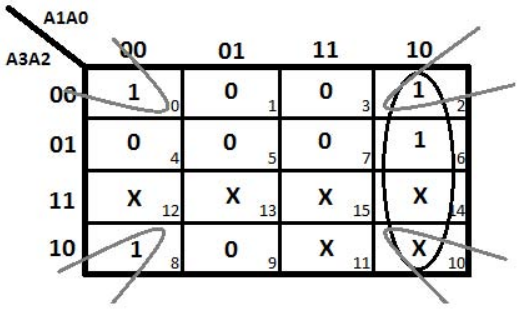


Figura 4.6: Mapa de Karnaugh para el segmento e

FUENTE:Los autores

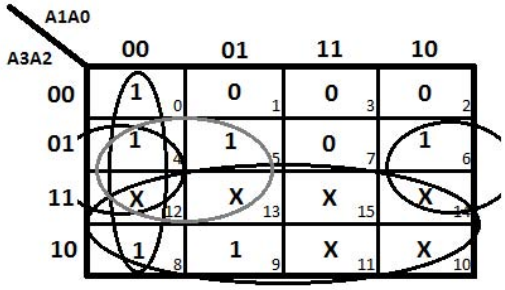


Figura 4.7: Mapa de Karnaugh para el segmento f

FUENTE:Los autores

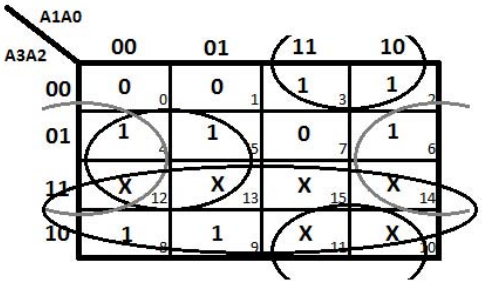


Figura 4.8: Mapa de Karnaugh para el segmento g

FUENTE:Los autores

$$b = (\overline{A_1} * \overline{A_0}) + (A_1 * A_0) + \overline{A_2} \tag{4.2}$$

$$c = \overline{A_1} + A_0 + A_2 \tag{4.3}$$

$$d = (A_2 * \overline{A_1} * A_0) + (\overline{A_2} * \overline{A_0}) + (\overline{A_2} * A_1) + (A_1 * \overline{A_0}) + A_3 \tag{4.4}$$

$$e = (\overline{A_2} * \overline{A_0}) + (A_1 * \overline{A_0}) \tag{4.5}$$

$$f = (\overline{A_1} * \overline{A_0}) + (A_2 * \overline{A_1}) + (A_2 * \overline{A_0}) + A_3 \tag{4.6}$$

$$g = (\overline{A_2} * A_1) + (A_2 * \overline{A_1}) + (A_2 * \overline{A_0}) + A_3 \tag{4.7}$$

4.4.1.5 Paso 5: Descripción del código en VHDL

A partir de la Suma de Productos es preciso llevar a cabo la descripción del código en VHDL, pues hasta este punto se ha identificado el comportamiento de las salidas del circuito en función de las entradas. Es así como en el Script 4.1 se lleva a cabo la respectiva descripción.

```
1  ----------DECLARACION DE LAS LIBRERIAS NECESARIAS----------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7  ----------DECLARACION DE ENTRADAS Y SALIDAS----------
8  entity bcd_a_7_seg_mapak is
9
10 Port ( A3 : in  STD_LOGIC;
```

```
11              A2 : in  STD_LOGIC;
12              A1 : in  STD_LOGIC;
13              A0 : in  STD_LOGIC;
14              a : out  STD_LOGIC;
15              b : out  STD_LOGIC;
16              c : out  STD_LOGIC;
17              d : out  STD_LOGIC;
18              e : out  STD_LOGIC;
19              f : out  STD_LOGIC;
20              g : out  STD_LOGIC);
21  end bcd_a_7_seg_mapak;
22
23
24  ----------------------------INICIO DE LA ARQUITECTURA-----------------------------
25  architecture Behavioral of bcd_a_7_seg_mapak is
26
27  begin
28
29  a<=((not A2)and(not A0))or((A2)and(A0))or(A1)or(A3);
30  b<=((not A1)and(not A0))or((A1)and(A0))or(not A2);
31  c<=(not A1)or(A0)or(A2);
32  d<=((A2)and(not A1)and(A0))or((not A2)and(not A0))or((not A2)and(A1))or((A1)and(not A0))or(A3);
33  e<=((not A2)and(not A0))or((A1)and(not A0));
34  f<=((not A1)and(not A0))or((A2)and(not A1))or((A2)and(not A0))or(A3);
35  g<=((not A2)and(A1))or((A2)and(not A1))or((A2)and(not A0))or(A3);
36
37  end Behavioral;
38  ----------------------------------------------------FIN-----------------------------------------------------
```

Script 4.1: Código de la descripción VHDL del decodificador BCD/7SEG

4.4.1.6 Paso 6: Simulación

Antes de realizar la implementación de cualquier descripción, es preciso asegurarse de los resultado esperados por medio de una simulación. Así pues en la figura 4.9 se muestra la respectiva simulación del decodificador a implementar.

En la simulación mostrada en la figura 4.9, se nota el marcador en aproximadamente 350 `ns`, donde se muestra que a una entrada $a3 = 0, a2 = 0, a1 = 1, a0 = 0$ correspondiente al número decimal 2, el circuito responde con una salida $a = 1, b = 1, c = 0, d = 1, e = 1, f = 0, g = 1$ correspondiente a los segmentos del display y mostrando el comportamiento esperado, luego ahora sí es posible pasar al siguiente paso.

4.4.1.7 Paso 7: Identificación de dispositivos a utilizar

Los módulos necesarios para la implementación de la práctica son:

- Digilent PmodSWT Switch Module Board. [8]
- Digilent PmodSSD Peripheral Module Board. [8]

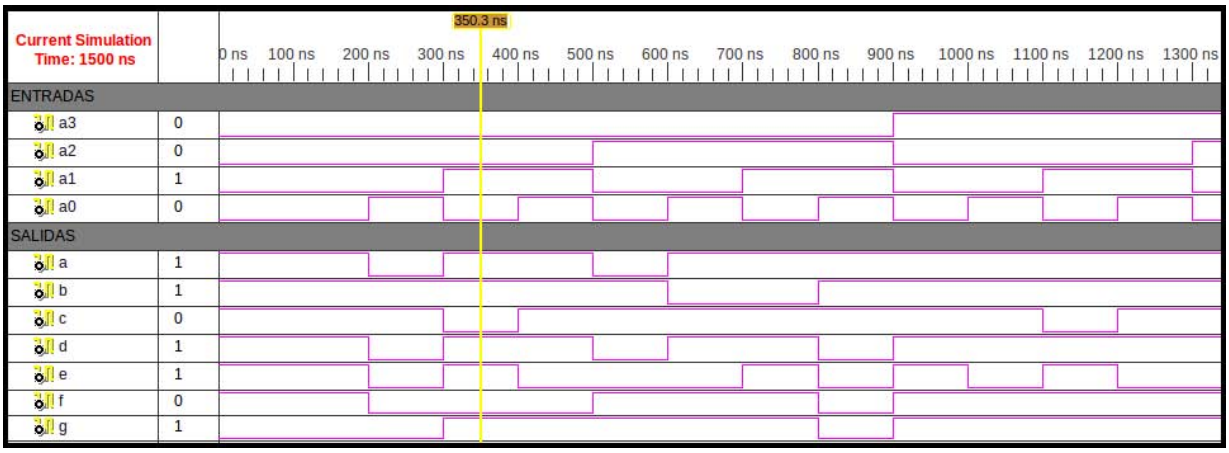


Figura 4.9: Simulación del decodificador con ecuaciones simplificadas

FUENTE:Herramienta de simulación ISE

4.4.1.8 Paso 8: Asignación de pines

La construcción del archivo `.ucf` se muestra en el Script 4.2.

```
net A3 loc="P54"; # pulsador switch 1
net A2 loc="P58"; # pulsador switch 2
net A1 loc="P61"; # pulsador switch 3
net A0 loc="P63"; # pulsador switch 4
net a loc="P33"; # segmento a del display
net b loc="P27"; # segmento b del display
net c loc="P24"; # segmento c del display
net d loc="P22"; # segmento d del display
net e loc="P49"; # segmento e del display
net f loc="P47"; # segmento f del display
net g loc="P40"; # segmento g del display
```

Script 4.2: Archivo ucf

4.4.1.9 Paso 9: Copiar el archivo Makefile en la carpeta de trabajo

Para llevar a cabo el procedimiento necesario en el presente paso, es preciso referirse a la guía de laboratorio 1, en donde se expone de forma detallada lo referente al Makefile, el cual es un archivo proporcionado por la persona encargada de guiar la clase, y al cual se le debe editar haciendo una pequeña modificación en la segunda linea, donde debe agregarse el nombre del fichero que se desea implementar.

4.4.1.10 Paso 10: Implementación en la tarjeta SIE

Ya teniendo los archivos necesarios para la respectiva implementación, es preciso llevar a cabo el proceso necesario para una óptima implementación, para lo cual es necesario realizar el proceso mostrado a continuación:

- Conectar la tarjeta SIE
- Configuración del puerto USB
- Realizar la síntesis de la descripción
- Enviar información al procesador
- Acceder al procesador de la tarjeta
- Enviar información al FPGA

Es preciso mencionar que si se tiene alguna duda para llevar a cabo el proceso mencionado anteriormente, es necesario referirse a la guía de laboratorio 1, en donde se explica detalladamente cada uno de los anteriores.

4.5 RESUMEN

A través de la presente práctica se ha dado a conocer el proceso necesario para la implementación de un decodificador que tiene como entrada un dato en código BCD, y sus salidas muestran un dígito decimal en un display de siete segmentos, estas últimas han tenido un proceso de simplificación el cual ha sido llevado a cabo por medio del método de minimización por Mapas de Karnaugh. Además el decodificador ha sido adaptado a la tarjeta SIE para su respectiva implementación. Asi pues, se han presentado los siguientes temas:

- Descripción en VHDL de un decodificador BCD/7SEG
- Minimización por medio de los Mapas de Karnaugh
- Implementación en la tarjeta SIE

4.6 PREGUNTAS DE PRUEBA

- ¿La descripción en VHDL se ha hecho por medio de funciones lógicas?
- ¿Indique con sus palabras para que se utilizan los mapas de Karnaugh?
- ¿Se obtendrían los mismos resultados sin utilizar los Mapas de Karnaugh?

4.7 EJERCICIO PROPUESTO

Modificar el archivo que contiene la decripción del decodificador en VHDL, para que en el display de 7 segmentos, se representen los primeros quince números en codigo hexadecimal con ecuaciones simplificadas. La tabla 4.2 puede ser de ayuda para la respectiva representación.

DECIMAL		HEXADECIMAL	DECIMAL		HEXADECIMAL
0	$\rightarrow$	0	8	$\rightarrow$	8
1	$\rightarrow$	1	9	$\rightarrow$	9
2	$\rightarrow$	2	10	$\rightarrow$	A
3	$\rightarrow$	3	11	$\rightarrow$	B
4	$\rightarrow$	4	12	$\rightarrow$	C
5	$\rightarrow$	5	13	$\rightarrow$	D
6	$\rightarrow$	6	14	$\rightarrow$	E
7	$\rightarrow$	7	15	$\rightarrow$	F

Tabla 4.2: Tabla para representar números decimales en hexadecimal

LABORATORIO **5**

Concurrencia.

Mi grandeza no reside en no haber caído nunca, sino en haberme levantado siempre.

NAPOLEÓN BONAPARTE

5.1 INTRODUCCIÓN

La descripción de un circuito a través de un lenguaje como `VHDL` se encuentra abierta a muchas posibilidades, las cuales pueden describir un mismo circuito desde perspectivas diferentes, lo que conlleva a no estar ligado con un estilo único a la hora de hacer una descripción.

Ahora bien, en una descripción es posible encontrar sentencias secuenciales y concurrentes, las primeras hacen referencia a establecer una prioridad y un orden; para las segundas no se necesita establecer un orden específico para llevar a cabo una óptima descripción de un circuito.

5.2 OBJETIVOS

- Realizar una descripción en `VHDL` mediante la aplicación de sentencias concurrentes.

5.3 MARCO TEÓRICO

5.3.1 Construcciones concurrentes

En VHDL existe un conjunto de sentencias concurrentes que permiten definir de forma clara y precisa los circuitos combinacionales, entendiendo por concurrencia la capacidad de procesar un conjunto de instrucciones de forma paralela.[9]

Las sentencias que generalmente son utilizadas en una descripción se pueden clasificar como se muestra a continuación:[10]

- SENTENCIAS CONCURRENTES
  - Procesos
  - Bloques
  - Sentencia when-else
  - Sentencia whit-select
- SENTENCIAS SECUENCIALES
  - Sentencia wait
  - Sentencias Condicionales
    - If-then-elsif-else
    - Case-when
    - Bucles
      - For-loop
      - While
      - Loop
      - Exit
      - Next

De la clasificación mostrada anteriormente es preciso mencionar que en una descripción por medio de VHDL, es posible encontrar tanto concurrencia como secuencialidad, aclarando que el comportamiento concurrente del circuito se despliega a través de toda la descripción de la arquitectura como se puede notar en el Script 5.1.

```
1 architecture Behavioral of entidad is
2 begin
3
4         --Aqui van las sentencias concurrentes
5
6 end Behavioral;
7 ----------------------------------------------------------------------
```

Script 5.1: Descripción VHDL de un multiplexor When-Else

Igualmente haciendo referencia al Script 5.2 se puede ver como una sentencia concurrente (`process`) puede ser descrita por medio de sentencias secuenciales, lo cuál no anula el comportamiento concurrente del circuito[11].

```
1  architecture Behavioral of entidad is
2
3  begin
4          process()
5            begin
6                sentencias secuenciales
7          end process;
8
9  end Behavioral;
10 ----------------------------------------------------------------------------
```

Script 5.2: Descripción VHDL de un multiplexor When-Else

Ahora bien, en una descripción es posible encontrar el comportamiento secuencial en varios procesos, pero la arquitectura en donde se hace la unión de todos estos procesos tendrá un comportamiento concurrente. Las sentencias concurrentes son sentencias condicionales que tienen al menos un valor por defecto para cuando no se cumple ninguna de las condiciones. Aunque podría utilizarse una sentencia común como un `if` con obligación de `else`, generalmente se utilizan dos sentencias particulares, `When-else` y `With-Select` [2].

5.3.1.1 Sentencia When-Else

La sentencia `When-Else` permite la descripción de circuitos combinacionales cuyas salidas sean funciones diversas de un conjunto de entradas, tal como es el caso de los multiplexores.

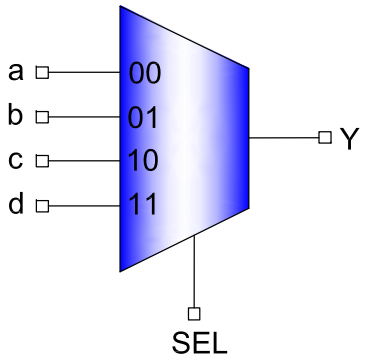


Figura 5.1: Multiplexor con una entrada de control.

FUENTE:Los autores

Por ejemplo en la figura 5.1 tenemos un multiplexor de una entrada de control de dos bits, donde dependiendo de la misma, la salida toma el valor de `a`,`b`,`c` ó `d`, que son las entradas de datos, el cual puede ser descrito en VHDL como se muestra en el Script 5.3.

```
1  ----------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
```

```
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------
7  entity mux_4a1 is
8
9  port (          a,b,c,d:in std_logic;
10                 sel:in std_logic_vector(1 downto 0);
11                 y: out std_logic);
12 end mux_4a1;
13 ----------------------------------------------------------------------------
14 architecture Behavioral of mux_4a1 is
15   begin
16
17         y<=     a when sel="00" else
18                 b when sel="01" else
19                 c when sel="10" else
20                 d;
21
22 end Behavioral;
23 ----------------------------------------------------------------------------
```

Script 5.3: Descripción VHDL de un multiplexor When-Else

En la descripción mostrada en el Script 5.3 es posible notar que la implementación del circuito combinacional que se ha realizado hace referencia al comportamiento del circuito, omitiendo la descripción clásica utilizando tablas de verdad, describiendo el circuito a nivel comportamental como se menciono anteriormente. Una herramienta de descripción de hardware facilita de gran manera el desarrollo de circuitos digitales como se ha verificado por medio de la sentencia when-else.

5.3.1.2 Sentencia Whit-Select

La sentencia `With-Select` permite enumerar todas las posibles elecciones que se puedan hacer de las entradas. En el momento en que se enumeren todas las entradas y exista un conjunto de entradas cuyas salidas son comunes o redundantes se puede utilizar la instrucción `when others` agrupando dichas entradas.

Para aclarar un poco la aplicación de la sentencia `With-Select`, se retomará el ejemplo del multiplexor mostrado en la figura 5.1, describiéndolo mediante VHDL como se muestra en el Script 5.4.

```
1  ----------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------
7  entity mux_4a1 is
8  port(           a,b,c,d:in std_logic;
9                  sel:in std_logic_vector(1 downto 0);
10                 y:out std_logic);
11 end mux_4a1;
12 ----------------------------------------------------------------------------
```

```
13  architecture Behavioral of mux_4a1 is
14  begin
15          WITH     sel select
16
17                   y<=      a when "00",
18                            b when "01",
19                            c when "10",
20                            d when others;
21
22  end Behavioral;
23  ------------------------------------------------------------------------
```

Script 5.4: Descripción VHDL de un multiplexor With-Select

Hasta este punto se han expuesto dos sentencias distintas que se pueden utilizar a la hora de describir un circuito digital, dependerá de la aplicación que lo requiera la sentencia a utilizar, pues aunque el circuito tiene el mismo comportamiento independientemente de la que se utilice para su descripción, existe una diferencia en la forma de implementación que puede dar la pauta para elegir entre una u otra. Así pues, en las figuras 5.2 y 5.3 se muestran los diagramas RTL del multiplexor descrito con la sentencia `when-else` y `with-select` respectivamente.

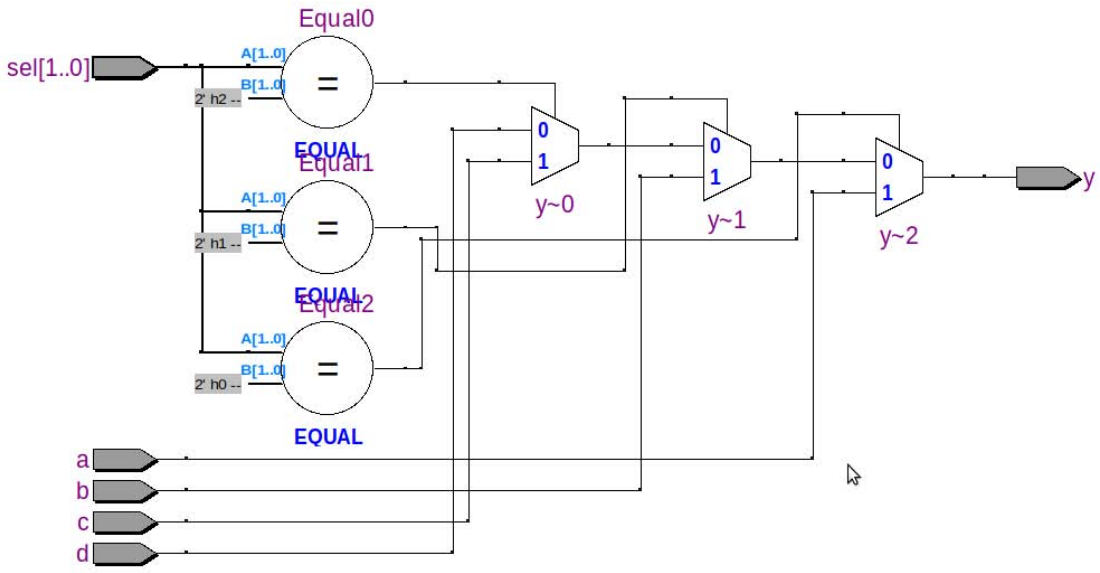


Figura 5.2: RTL de un mux 4 a 1 descrito con when-else.

FUENTE:Herramienta de simulación Quartus II

Cabe resaltar que todo circuito tiene un tiempo de respuesta que por mínimo que sea, en muchas aplicaciones es necesario tenerlo en cuenta. Haciendo un seguimiento a los datos de entrada del circuito descrito con la sentencia `when-else`, es preciso notar en la figura 5.2 que para una entrada `sel=10` el dato `c` debe pasar por tres multiplexores, mientras que para una entrada `sel=00`, el dato `a` solo debe pasar por un multiplexor para verse reflejado en la salida del circuito, claro está que son tiempos sumamente pequeños que son notorios en aplicaciones particulares.

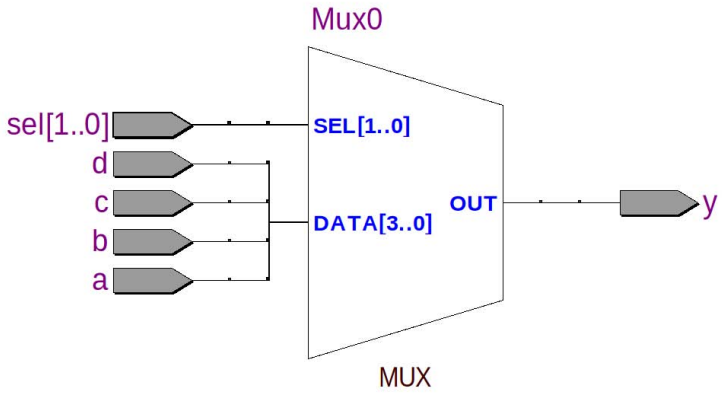


Figura 5.3: RTL de un mux 4 a 1 descrito con with-select.

FUENTE:Herramienta de simulación Quartus II

Ahora bien, el diagrama RTL del circuito descrito por medio de la sentencia `with-select`, muestra que el proceso que debe seguir cualquiera de los datos de entrada para ser reflejado a la salida del circuito, tarda el mismo tiempo independientemente del dato de entrada seleccionado.

Así pues, la diferencia entre una y otra se puede notar en las conexiones que realiza cada una, la sentencia `when-else` muestra multiplexores conectados en cascada, mientras que la sentencia `with-select` muestra conexiones de multiplexores en paralelo.

5.3.2 Decodificadores

La función básica de un decodificador es detectar la presencia de una determinada combinación de bits (código) en sus entradas y señalar la presencia de este código mediante un cierto nivel de salida. En su forma más general, un decodificador posee, líneas de entrada para gestionar n bits, y en una de las 2^n líneas de salida indica la presencia de una o más combinaciones de n bits [4].

5.4 PROCEDIMIENTO GENERAL

Se requiere la representación de los números binarios en decimales, para esto se propone hacer un decodificador que tiene como dato de entrada un código BCD y proporciona salidas capaces de excitar un display de 7 segmentos para indicar un digito decimal, describiendo dicho circuito bajo el uso de las instrucciones `when-else` y `with-select`. Para dicho procedimiento se recomienda seguir los siguientes pasos:

- ☞ Identificación de entradas y salidas
- ☞ Tabla de verdad
- ☞ Descripción del código en VHDL

- ☞ Simulación
- ☞ Identificación de dispositivos a utilizar
- ☞ Asignación de pines creando el archivo de extensión .ucf
- ☞ Copiar el archivo Makefile en la carpeta de trabajo
- ☞ Implementación en la tarjeta SIE

5.4.1 PROCEDIMIENTO PASO A PASO

Teniendo una idea general de las especificaciones del problema y de los pasos a seguir, se procede a realizar un análisis detallado del mismo.

5.4.1.1 Paso 1: Identificación de entradas y salidas

En la caja representativa de la figura 5.4, se tienen 4 entradas A_0,A_1,A_2 y A_3 tomando como bit menos significativo A_0 y como bit más significativo A_3, los cuales corresponden al código BCD, y la combinación de estos bits, darán como resultado una salida indicada por los 7 bits o 7 segmentos denotados por las letras a,b,c,d,e,f y g, que corresponden a los 7 segmentos del display.

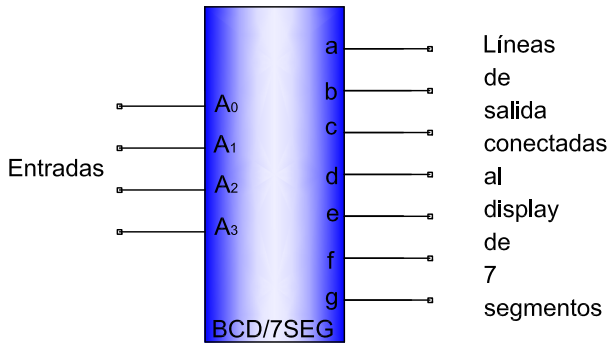


Figura 5.4: Caja representativa de un decodificador BCD/7SEG.

FUENTE:Los autores

Cabe mencionar que las entradas denotadas por A_0, A_1, A_2 y A_3 se pueden reemplazar por un vector de 4 bits, al igual que las salidas a,b,c,d,e,f y g por un vector de 7 bits, y se obtendran los mismos resultados.

5.4.1.2 Paso 2: Tabla de verdad

Ahora bien, sabiendo con exactitud las entradas y correspondientes salidas que manejará el circuito, es preciso organizar dicha información que se tiene hasta el momento en una tabla de verdad, tal como se muestra en la tabla 5.1.

	ENTRADAS				SALIDAS						
DECIMAL	A_3	A_2	A_1	A_0	P_6	P_5	P_4	P_3	P_2	P_1	P_0
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Tabla 5.1: Tabla de verdad del decodificador BCD/7SEG

5.4.1.3 Paso 3: Descripción del código en VHDL

En el Script 5.5 se muestra la descripción del decodificador utilizando la instrucción `When-else`.

```
1  ----------------------------------------------------------------------------------
2
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use IEEE.STD_LOGIC_ARITH.ALL;
6  use IEEE.STD_LOGIC_UNSIGNED.ALL;
7
8  ----------------------------------------------------------------------------------
9
10 entity deco_7seg_WE is
11   Port (          A : in  STD_LOGIC_VECTOR (3 downto 0);
12                   P : out  STD_LOGIC_VECTOR (6 downto 0));
13 end deco_7seg_WE;
14
15 ----------------------------------------------------------------------------------
16
17 architecture Behavioral of deco_7seg_WE is
18
19 begin
20
21 P<="1111110" when A="0000" else
22                 "0110000" when A="0001" else
23                 "1101101" when A="0010" else
24                 "1111001" when A="0011" else
25                 "0110011" when A="0100" else
26                 "1011011" when A="0101" else
27                 "1011111" when A="0110" else
28                 "1110000" when A="0111" else
29                 "1111111" when A="1000" else
30                 "1111011" when A="1001" else
31                 "0000000";
32
```

```
33 end Behavioral;
34 ----------------------------------------------------------------------------------
```

Script 5.5: Descripción en VHDL de un decodificador BCD/7SEG con la sentencia When-else

En el Script 5.6 se muestra la descripción del decodificador utilizando la instrucción `Whit-Select`.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7
8  entity deco_7seg_WS is
9    Port (         A : in  STD_LOGIC_VECTOR (3 downto 0);
10                  P : out  STD_LOGIC_VECTOR (6 downto 0));
11 end deco_7seg_WS;
12
13 ----------------------------------------------------------------------------------
14
15 architecture Behavioral of deco_7seg_WS is
16
17 begin
18
19 with A select
20
21         p<= "1111110" when "0000",
22                 "0110000" when "0001",
23                 "1101101" when "0010",
24                 "1111001" when "0011",
25                 "0110011" when "0100",
26                 "1011011" when "0101",
27                 "1011111" when "0110",
28                 "1110000" when "0111",
29                 "1111111" when "1000",
30                 "1111011" when "1001",
31                 "0000000" when others;
32
33 end Behavioral;
34 ----------------------------------------------------------------------------------
```

Script 5.6: Descripción en VHDL de un decodificador BCD/7SEG con la sentencia Whit-Select

5.4.1.4 Paso 4: Simulación

En la figura 5.5 se muestra la simulación correspondiente al decodificador descrito por medio de la sentencia concurrente when-else, donde se nota el marcador en aproximadamente 550 `ns`, mostrando que a una entrada correspondiente al número decimal 4 expresado en BCD, el circuito responde con una salida $p[6] = 0, p[5] = 1, p[4] =$

$1, p[3] = 0, p[2] = 0, p[1] = 1, p[0] = 1$ correspondiente a los segmentos del display y mostrando el comportamiento esperado.

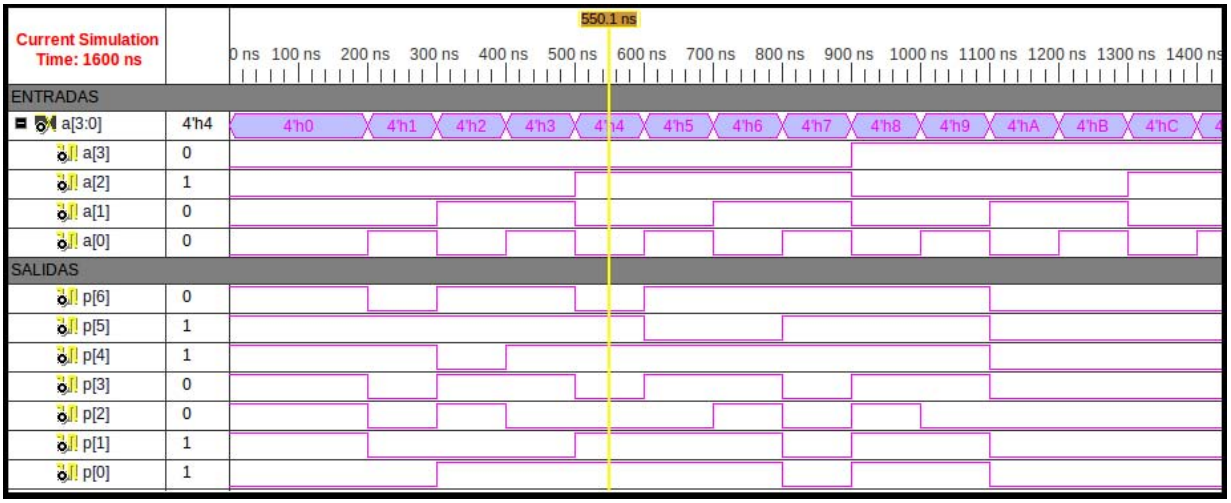


Figura 5.5: Simulación del decodificador descrito con la sentencia when-else

FUENTE:Herramienta de simulación ISE

De igual forma, en la figura 5.6 se muestra la simulación correspondiente al decodificador descrito por medio de la sentencia concurrente whit-select, donde se nota el marcador en aproximadamente 650 `ns`, mostrando que a una entrada correspondiente al número decimal 5 expresado en BCD, el circuito responde con una salida $p[6] = 1, p[5] = 0, p[4] = 1, p[3] = 1, p[2] = 0, p[1] = 1, p[0] = 1$ correspondiente a los segmentos del display y mostrando el comportamiento esperado.

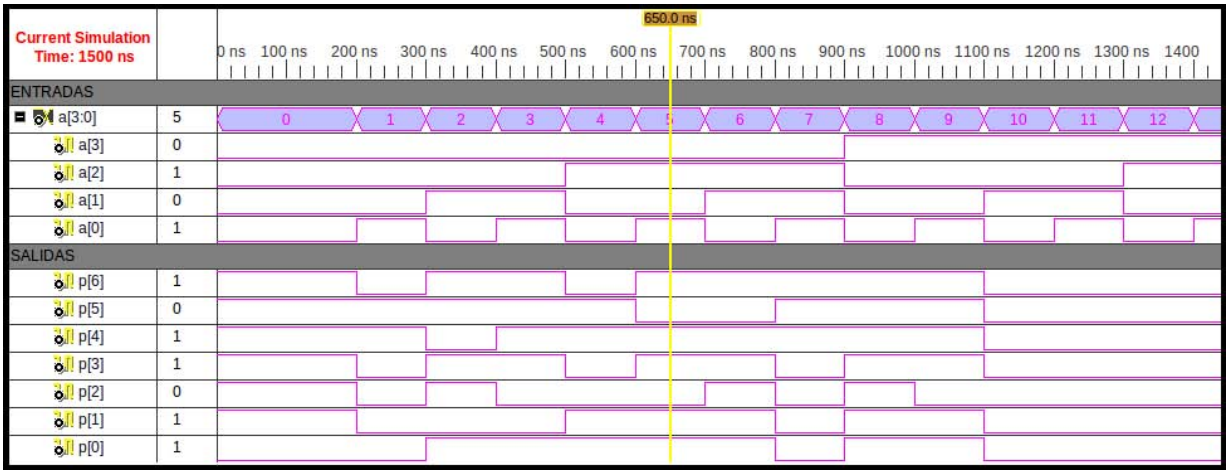


Figura 5.6: Simulación del decodificador descrito con la sentencia whit-select

FUENTE:Herramienta de simulación ISE

5.4.1.5 Paso 5: Identificación de dispositivos a utilizar

Los módulos necesarios para la implementación de la práctica son:

- Digilent PmodSWT Switch Module Board. [8]
- Digilent PmodSSD Peripheral Module Board. [8]

5.4.1.6 Paso 6: Asignación de pines creando el archivo de extensión .ucf

La construcción del archivo `.ucf` se muestra en el Script 5.7. Los pines `P60,P62,P65` y `P70` se conecta el PmodSWT Switch Module Board, y en los pines `P33,P27,P24,P22,P49.P47` y `P40` se conecta el Digilent PmodSSD Peripheral Module Board.

```
1  net A(3) loc="P54"; # pulsador switch 1
2  net A(2) loc="P58"; # pulsador switch 2
3  net A(1) loc="P61"; # pulsador switch 3
4  net A(0) loc="P63"; # pulsador switch 4
5  net P(6) loc="P33"; # segmento a del display
6  net P(5) loc="P27"; # segmento b del display
7  net P(4) loc="P24"; # segmento c del display
8  net P(3) loc="P22"; # segmento d del display
9  net P(2) loc="P49"; # segmento e del display
10 net P(1) loc="P47"; # segmento f del display
11 net P(0) loc="P40"; # segmento g del display
```

Script 5.7: Archivo ucf

5.4.1.7 Paso 7: Copiar el archivo Makefile en la carpeta de trabajo

Para llevar a cabo el procedimiento necesario en el presente paso, es preciso referirse a la guía de laboratorio 1, en donde se expone de forma detallada lo referente al Makefile, el cual es un archivo proporcionado por la persona encargada de guiar la clase, y se le debe editar haciendo una pequeña modificación en la segunda linea, donde debe agregarse el nombre del fichero que se desea implementar.

5.4.1.8 Paso 8: Implementación en la tarjeta SIE

Ya teniendo los archivos necesarios para la respectiva implementación, es preciso llevar a cabo el proceso necesario para una óptima implementación, para lo cual es necesario realizar el proceso mostrado a continuación:

- Conectar la tarjeta SIE
- Configuración del puerto USB

- Realizar la síntesis de la descripción
- Enviar información al procesador
- Acceder al procesador de la tarjeta
- Enviar información al FPGA

Es preciso mencionar que si se tiene alguna duda para llevar a cabo el proceso mencionado anteriormente, es necesario referirse a la guía de laboratorio 1, en donde se explica detalladamente cada uno de los anteriores.

5.5 RESUMEN

En el laboratorio realizado se dio a conocer el concepto de concurrencia, y se mostraron dos de las sentencias utilizadas para la descripción de circuitos con VHDL, `When-else` y `Whit-Select`, presentando los siguientes temas:

- Descripción en VHDL de un decodificador BCD/7SEG utilizando la sentencia `When-Else`.
- Descripción en VHDL de un decodificador BCD/7SEG utilizando la sentencia `Whit-Select`.
- Implementación en la tarjeta SIE.

5.6 PREGUNTAS DE PRUEBA

- ¿Defina con sus palabras el concepto de concurrencia?
- ¿Cómo saber cuál sentencia utilizar al momento de describir un circuito?
- ¿El resultado final de la implementación depende de la sentencia que se utilice para describir un circuito?
- ¿Nombre cuatro formas diferentes de describir un decodificador?

5.7 EJERCICIO PROPUESTO

Modificar el archivo que contiene la decripción del decodificador en VHDL, para que en el display de 7 segmentos, se representen los primeros quince números en codigo hexadecimal, utilizando las sentencias concurrentes descritas en la presente guía. La tabla 5.2 puede ser de ayuda para la respectiva representación.

DECIMAL		HEXADECIMAL	DECIMAL		HEXADECIMAL
0	$\rightarrow$	0	8	$\rightarrow$	8
1	$\rightarrow$	1	9	$\rightarrow$	9
2	$\rightarrow$	2	10	$\rightarrow$	A
3	$\rightarrow$	3	11	$\rightarrow$	B
4	$\rightarrow$	4	12	$\rightarrow$	C
5	$\rightarrow$	5	13	$\rightarrow$	D
6	$\rightarrow$	6	14	$\rightarrow$	E
7	$\rightarrow$	7	15	$\rightarrow$	F

Tabla 5.2: Tabla para representar números decimales en hexadecimal

FUENTE:Los autores

LABORATORIO **6**

Descripción estructural.

Ríe y el mundo reirá contigo; llora y el mundo, dándote la espalda, te dejará llorar.

CHARLES CHAPLIN

6.1 INTRODUCCIÓN

En las descripciones de circuitos por medio de un lenguaje como VHDL es posible encontrar gran variedad de aplicaciones, dentro de las cuales pueden haber desde circuitos tan simples como una compuerta AND hasta circuitos mas complejos como un sistema de control para un determinado proceso, el cual necesita de una considerable cantidad de compuertas lógicas para ser descrito, además en muchas ocasiones se ha tenido que recurrir a la reutilización de módulos ya existentes que pueden ser de gran utilidad, lo cual disminuye gran parte del tiempo de trabajo, lo que hace que la descripción de hardware sea una herramienta tan recomendada.

Así pues, para descripciones de hardware algo extensas se ha venido trabajando de manera modular, es decir dividiendo toda la descripción en bloques mas sencillos, lo cual ha traído grandes ventajas como la pronta detección y solución de errores, la reutilización de módulos ya existentes en una nueva descripción, la organización y entendimiento de la descripción, la sencilla metodología para describir hardware entre otras.

6.2 OBJETIVOS

- Implementar una descripción estructural por medio de VHDL.
- Describir el funcionamiento de un semisumador.
- Describir el funcionamiento de un sumador completo.

6.3 MARCO TEÓRICO

6.3.1 Descripción estructural

Realizar descripciones estructurales tiene grandes ventajas en cuanto a organización, tiempo y reutilización de descripciones ya existentes, por lo tanto es de gran importancia saber el proceso a seguir a la hora de hacer una interconexión entre diversos módulos en el lenguaje VHDL, por tal motivo a continuación se muestra un ejemplo que sirva como base para descripciones posteriores más complejas, el cual consiste en describir una compuerta `and` de cuatro entradas por medio de tres compuertas `and`, cada una de dos entradas. Para ello se propone realizar el proceso mostrado a continuación.

6.3.1.1 Hacer un diagrama de lo que se quiere hacer ó RTL

La descripción a realizar corresponde a una compuerta `and` de cuatro entradas por medio de compuertas `and` de dos entradas, luego es preciso notar en la figura 6.1 una serie de esquemas representativos; inicialmente haciendo un bosquejo por medio de compuertas `and` de dos entradas, luego viendo cada una de estas compuertas como un pequeño bloque cada uno con dos entradas y una salida, y finalmente la compuerta `and` de cuatro entradas (`AND_4`) como el bloque principal, con cuatro señales de entrada y una de salida como es de esperarse.

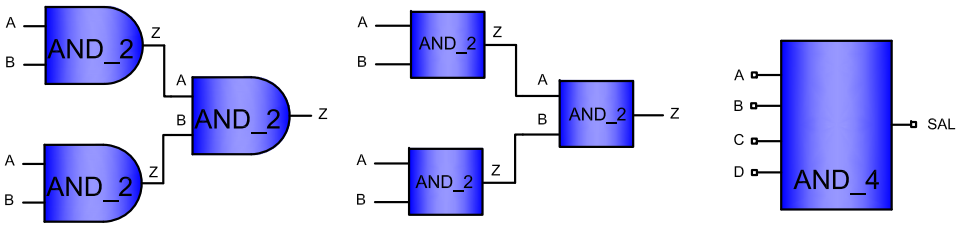


Figura 6.1: Representaciones de una compuerta AND de cuatro entradas.

FUENTE:Los autores

Ya teniendo el modelo estructural de la compuerta `and` de cuatro entradas, se recomienda poner nombre a cada una de las señales que hacen conexiones internas entre cada uno de los módulos, omitiendo poner señales a las que van conectadas directamente a entradas ó salidas del módulo principal, tal como lo muestra la figura 6.2.

6.3.1.2 Identificar y realizar la descripción de cada uno de los módulos internos

Para el ejemplo, los bloques internos que componen la compuerta `and` de cuatro entradas, son tres módulos `and` de dos entradas y una salida cada uno, los cuales a su vez tienen internamente una compuerta `and` tal como lo muestra la figura 6.2. Así pues, es preciso en primera instancia, hacer que cada uno de los módulos internos funcione de manera correcta, por lo que se procede a describir la compuerta `and` de dos entradas como se muestra en el Script 6.1.

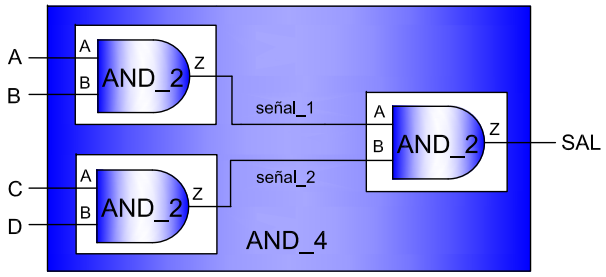


Figura 6.2: Descripción Estructural de una compuerta AND de cuatro entradas

FUENTE:Los autores

```
1  ----------------------------------
2  LIBRARY ieee ;
3  USE ieee.std_logic_1164.all ;
4  USE ieee.std_logic_arith.all;
5  USE ieee.std_logic_unsigned.all;
6  ----------------------------------
7  ENTITY and_dos IS
8  PORT ( A: in std_logic;
9         B: in std_logic;
10        Z: out std_logic);
11 END and_dos ;
12 ----------------------------------
13 ARCHITECTURE Behavioral OF and_dos IS
14 BEGIN
15     Z <= (A and B);
16 END Behavioral;
17 ----------------------------------
```

Script 6.1: Código de una descripción VHDL de una compuerta AND.

Después de esto se debe realizar la respectiva simulación e implementación de la compuerta `AND` de dos puertos, y asegurarse de que funciona correctamente. Cabe resaltar que es posible utilizar tantos componentes como se necesiten, pero para el caso, el único módulo interno que compone el circuito a implementar, es la compuerta `and` de dos puertos.

6.3.1.3 Descripción del circuito principal

Una vez seguros del exitoso funcionamiento de cada uno de los módulos que componen el circuito principal, se procede a realizar la descripción tal como se muestra en el Script 6.2.

```
1  ------------------------------------------------------------------------------
2  LIBRARY ieee ;
3  USE ieee.std_logic_1164.all ;
4  USE ieee.std_logic_arith.all;
5  USE ieee.std_logic_unsigned.all;
```

```
6  ----------ENTRADAS Y SALIDAS DEL CIRCUITO PRINCIPAL----------
7  ENTITY and_cuatro IS
8  PORT ( A : IN  STD_LOGIC ;
9         B : IN  STD_LOGIC;
10        C : IN  STD_LOGIC;
11        D : IN  STD_LOGIC;
12        SAL: OUT STD_LOGIC) ;
13 END and_cuatro ;
14 ------------------------------------------------------------
15 ARCHITECTURE behavioral OF and_cuatro IS
16 ------------------------------------------------------------
17 --Se declara cada uno de los componentes a utilizar describiendo sus entradas y salidas-----
18
19 ----------Componente compuerta and_dos----------
20 component and_dos
21         port  (A: in std_logic;
22                B: in std_logic;
23                Z: out std_logic);
24 end component;
25 ------------------------------------------------------------
26 ----------SEÑALES PARA LAS INTERCONEXIONES----------
27 signal senal_1:std_logic;
28 signal senal_2:std_logic;
29 ------------------------------------------------------------
30 BEGIN
31 ----------INSTANCIAS----------
32 ----Para las instancias, se procede a realizar la conexión de cada uno de los puertos-----
33 --- de cada uno de los módulos que componen el circuito principal, se debe seguir el RTL----
34
35 ----------COMPUERTA AND_DOS # 1----------
36 instancia_and_dos_1:and_dos
37
38         port map (A=A>,
39                   B=B>,
40                   Z=>senal_1);
41 ----------COMPUERTA AND_DOS # 2----------
42 instancia_and_dos_2:and_dos
43
44         port map (A=>C,
45                   B=>D,
46                   Z=>senal_2);
47 ----------COMPUERTA AND_DOS # 3----------
48 instancia_and_dos_3:and_dos
49
50         port map (A=>senal_1,
51                   B=>senal_2,
52                   Z=>SAL);
53 ----------COMPUERTA AND_DOS # 1----------
54 END behavioral;
```

Script 6.2: Código de una descripción VHDL de una compuerta AND.

6.3.1.4 Agregar las fuentes al Makefile

Uno de los aspectos que se deben tener en cuenta cuando se trabaja una descripción estructural, consiste en agregar al archivo `Makefile`, las fuentes (archivos `.vhd`) de los módulos internos que constituyen la descripción principal, lo cual se debe hacer en el renglón que inicia con la palabra `SRC_HDL` y dejando un espacio con la fuente principal de la descripción que está denominada como `(DESIGN).vhd` y que para este caso hace referencia al seudónimo `and_cuatro`.

```
1  # Definiciones del proyecto: Nombres, tarjeta, pines, etc..
2  DESIGN          = and_cuatro
3  PINS            = $(DESIGN).ucf
4  DEVICE          = xc3s100e-VQ100-4 # SIE
5  .
6  .
7  .
8  SIMGEN_OPTIONS  = -p $(FPGA_ARCH) -lang $(LANGUAGE)
9  SAKC_IP         = 192.168.1.1
10
11 # Definiciones de las fuentes .vhd y .v
12 #SRC              = $(DESIGN).v ADC_peripheral.v # Fuentes verilog
13
14 SRC_HDL         = $(DESIGN).vhd          and_dos.vhd     #Fuentes vhd
15 # Objetivos del Makefile
16 all:            bits
17 .
18 .
19 .
20 upload: $(DESIGN).bit
21         scp $(DESIGN).bit root@$(SAKC_IP):
```

Script 6.3: Código del Makefile.

6.3.2 Semisumador

Las reglas básicas de la suma binaria son:

$$\begin{boxed} 0+0=0 \\ 0+1=1 \\ 1+0=1 \\ 1+1=10 \end{boxed}$$

Un semisumador admite dos dígitos binarios en sus entradas y genera dos dígitos binarios en sus salidas: un bit de suma y un bit de acarreo. Los semisumadores son generalmente representados mediante el símbolo y diagrama lógico mostrado en la figura 6.3, presentando el comportamiento que se puede ver en la tabla 6.1.[4]

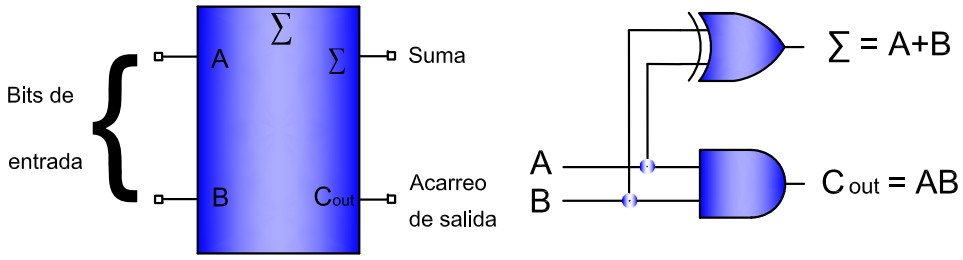


Figura 6.3: Símbolo y diagrama lógico de un semisumador

FUENTE:Los autores

A	B	C_{out}	$\sum$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 6.1: Tabla de verdad de un semisumador.

6.3.3 Sumador completo

Un sumador completo acepta dos bits de entrada y un acarreo de entrada, y genera una salida de suma y un acarreo de salida. Un sumador completo es generalmente representado mediante el símbolo lógico mostrado en la figura 6.4 y presenta el comportamiento que se muestra en la tabla 6.2.[4]

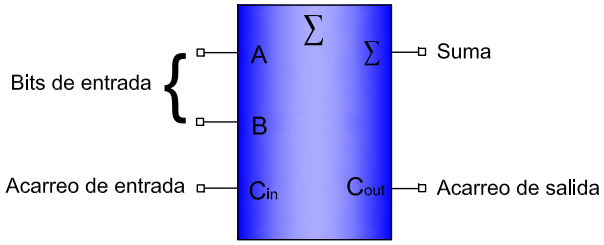


Figura 6.4: Símbolo lógico de un sumador completo

FUENTE:Los autores

Es preciso resaltar que un sumador completo puede ser descrito utilizando dos semisumadores, para lo cuál se requiere de una descripción estructural, donde la figura 6.5 muestra un esquema de las interconexiones que se deben hacer (RTL).

A	B	C_{in}	C_{out}	$\sum$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabla 6.2: Tabla de verdad de un sumador completo.

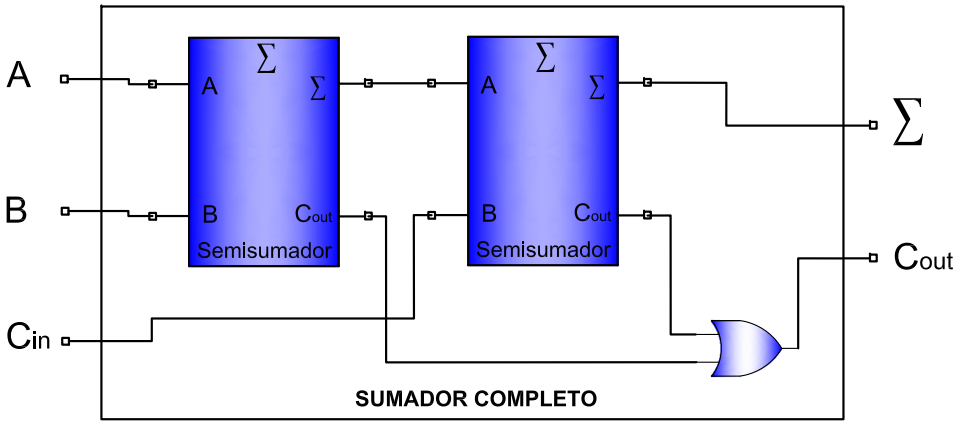


Figura 6.5: Diagrama RTL del sumador completo

FUENTE:Los autores

6.4 PROCEDIMIENTO GENERAL

Se propone realizar un sumador que tenga como entradas, dos números binarios cada uno de dos bits, y como salida la suma de estos números. Para dicho procedimiento se recomienda seguir los siguientes pasos:

- ☞ Identificación de entradas y salidas
- ☞ Realizar el esquema RTL
- ☞ Describir en VHDL los módulos internos
- ☞ Hacer la descripción estructural del circuito a implementar
- ☞ Simulación
- ☞ Identificación de dispositivos requeridos para la implementación

- ☞ Asignación de pines creando el archivo de extensión .ucf
- ☞ Agregar las fuentes necesarias al archivo Makefile
- ☞ Implementación

6.4.1 PROCEDIMIENTO PASO A PASO

Teniendo una idea general de las especificaciones del problema y de los pasos a seguir, se procede a realizar un análisis detallado del mismo.

6.4.1.1 Paso 1: Identificación de entradas y salidas

En la caja representativa de la figura 6.6 es preciso notar dos entradas cada uno de dos bits, que hacen referencia a los dos números binarios que van a sumarse, de igual manera aparece también una salida de tres bits, correspondiente al resultado de la suma.

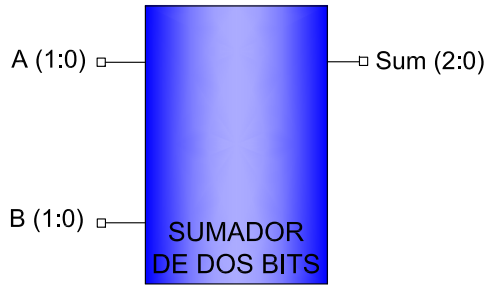


Figura 6.6: Sumador de dos bits

FUENTE:Los autores

6.4.1.2 Paso 2: Realizar el esquema RTL

En la figura 6.7 se presenta el esquema RTL, donde se muestra el sumador de dos bits como circuito principal y cada uno de los módulos que lo componen, dentro de los que aparecen un semisumador, un sumador completo, y las respectivas conexiones a realizar.

6.4.1.3 Paso 3: Describir en VHDL cada uno de los módulos internos

La descripción del semisumador se muestra en el Script 6.4.

```
1 ----------------------------------------------------------------------------------
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_ARITH.ALL;
5 use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

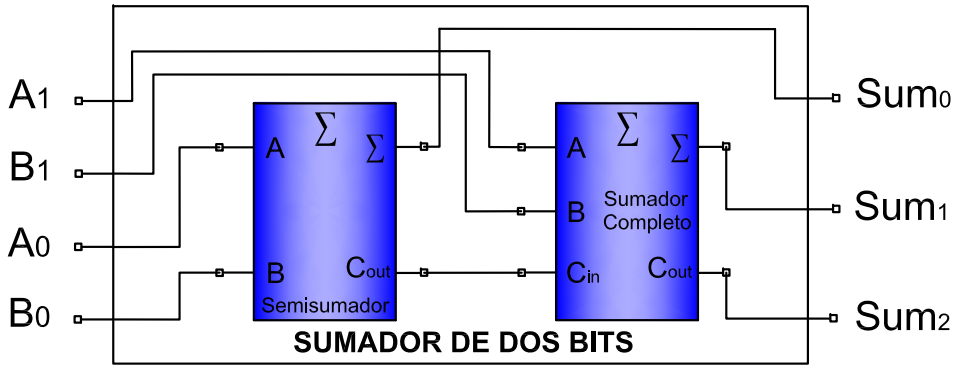


Figura 6.7: Diagrama Estructural del Sumador

FUENTE:Los autores

```
6  ----------------------------------------------------------------------------------
7  entity semisumador is
8  Port ( A : in STD_LOGIC;
9                         B : in STD_LOGIC;
10                        Suma : out STD_LOGIC;
11                        Cout : out STD_LOGIC);
12 end semisumador;
13 ----------------------------------------------------------------------------------
14 architecture Behavioral of semisumador is
15 begin
16 Suma<=A xor B;
17 Cout<=A and B;
18 end Behavioral;
19 ----------------------------------------------------------------------------------
```

Script 6.4: Descripción en VHDL del semisumador

El sumador completo se ha descrito de forma estructural utilizando dos módulos semisumadores y su descripción se muestra en el Script 6.5.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity sumadorcompleto is
8
9  Port ( A : in STD_LOGIC;
10        B : in STD_LOGIC;
11        Cin : in STD_LOGIC;
12        SUMA : out STD_LOGIC;
13        Cout : out STD_LOGIC);
14
15 end sumadorcompleto;
```

```
16 ----------------------------------------------------------------------------------
17 architecture Behavioral of sumadorcompleto is
18 ----------------------------------------COMPONENTES-----------------------------------------
19 component semisumador
20     Port (       A : in  STD_LOGIC;
21                  B : in  STD_LOGIC;
22                  Suma : out  STD_LOGIC;
23                  Cout : out  STD_LOGIC);
24 end component;
25 ----------------------------------DECLARACION DE SEÑALES-----------------------------------
26 signal signal_suma:std_logic;
27 signal signal_1_or:std_logic;
28 signal signal_2_or:std_logic;
29 ----------------------------------------------------------------------------------
30 begin
31 -------------------------------------INSTANCIAS-------------------------------------------
32 inst_semisumador_1:semisumador
33         port map (       A=>A,
34                          B=>B,
35                          Suma=>signal_suma,
36                          Cout=>signal_1_or);
37
38 inst_semisumador_2:semisumador
39         port map (       A=>signal_suma,
40                          B=>Cin,
41                          Suma=>SUMA,
42                          Cout=>signal_2_or);
43 ----------------------------------------------------------------------------------
44 Cout<=(signal_1_or)or(signal_2_or);
45 end Behavioral;
46 ----------------------------------------------------------------------------------
```

Script 6.5: Descripción en VHDL del sumador completo

6.4.1.4 Paso 4: Hacer la descripción estructural del circuito a implementar

El sumador de dos bits se puede describir como se muestra en el Script 6.6, donde se pueden notar los módulos semisumador y sumadorcompleto, los cuales componen el sumador de dos bits, tal como se ilustró en el diagrama RTL de la la figura 6.7.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity sumador_dos_bits is
8  Port (  A : in  STD_LOGIC_vector(1 downto 0);
9          B : in  STD_LOGIC_vector(1 downto 0);
10         Sum  : out  STD_LOGIC_vector(2 downto 0));
11 end sumador_dos_bits;
```

```
12 ----------------------------------------------------------------------------------
13 architecture Behavioral of sumador_dos_bits is
14 ----------------------------------COMPONENTES-------------------------------------
15 component semisumador
16     Port (       A : in  STD_LOGIC;
17                  B : in  STD_LOGIC;
18                  Suma : out  STD_LOGIC;
19                  Cout : out  STD_LOGIC);
20 end component;
21
22 component sumadorcompleto
23     Port ( A : in  STD_LOGIC;
24            B : in  STD_LOGIC;
25            Cin : in  STD_LOGIC;
26            SUMA : out  STD_LOGIC;
27            Cout : out  STD_LOGIC);
28 end component;
29 ----------------------------------DECLARACION DE SEÑALES--------------------------
30 signal signal_acarreo:std_logic;
31 ----------------------------------------------------------------------------------
32 begin
33 ------------------------------------INSTANCIAS------------------------------------
34 inst_semisumador:semisumador
35         port map (       A=>A(0),
36                          B=>B(0),
37                          Suma=>Sum(0),
38                          Cout=>signal_acarreo);
39
40 inst_sumadorcompleto:sumadorcompleto
41         port map (       A=>A(1),
42                          B=>B(1),
43                          Cin=>signal_acarreo,
44                          SUMA=>Sum(1),
45                          Cout=>Sum(2));
46
47 end Behavioral;
```

Script 6.6: Descripción en VHDL de un sumador de dos bits

6.4.1.5 Paso 5: Simulación

En la figura 6.8 se muestra la simulación correspondiente al semisumador, donde se nota el marcador en aproximadamente 250 ns, mostrando que a una entrada correspondiente de $1 + 1$, el circuito responde con una salida 10, mostrando así el comportamiento esperado.

En la figura 6.9 se muestra la simulación correspondiente al sumador completo, donde se nota el marcador en aproximadamente 850 ns, mostrando que a una entrada correspondiente de $1 + 0$, el circuito responde con una salida 01, mostrando así el comportamiento esperado.

Finalmente, en la figura 6.10 se muestra la simulación correspondiente al sumador de dos bits, donde se nota el

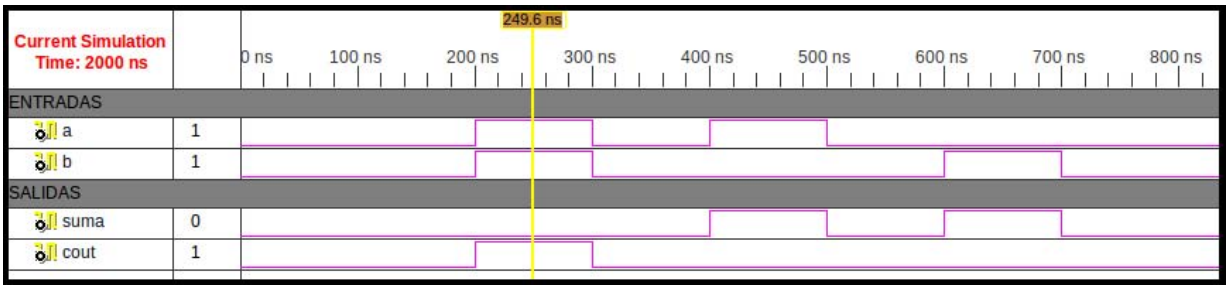


Figura 6.8: Simulación del semisumador

FUENTE:Herramienta de simulación ISE

Figura 6.9: Simulación del sumador completo

FUENTE:Herramienta de simulación ISE

marcador en aproximadamente 1030 `ns`, mostrando que a una entrada correspondiente a $2+1$, el circuito responde con una salida igual a 3, mostrando así el comportamiento esperado.

En la figura 6.10 se muestra la simulación del sumador completo.

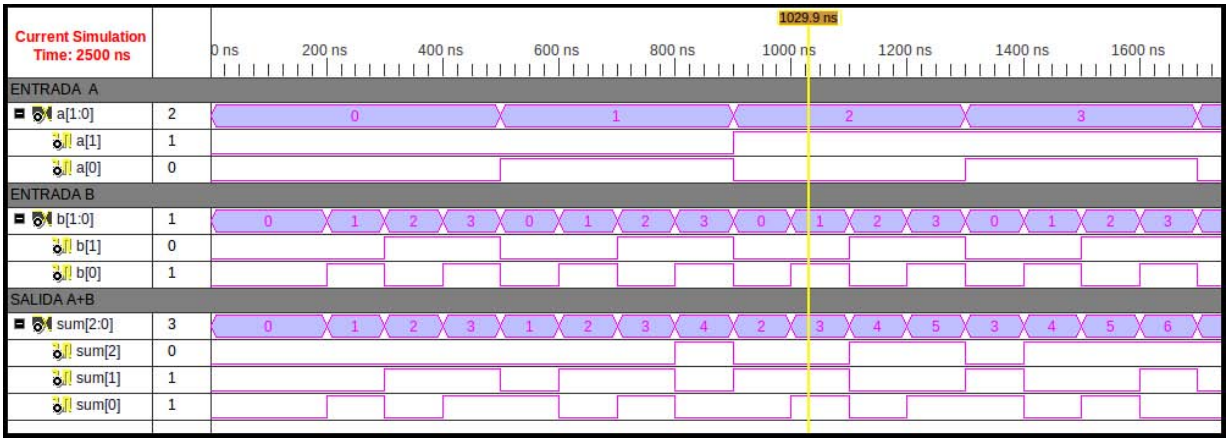


Figura 6.10: Sumador

FUENTE:Herramienta de simulación ISE

6.4.1.6 Paso 6: Identificación de dispositivos requeridos para la implementación

Los módulos necesarios para la implementación de la práctica son:

- Digilent PmodSWT Switch Module Board. [8]
- 3 Leds de la tarjeta de expansión de pines.

6.4.1.7 Paso 7: Asignación de pines creando el archivo de extensión .ucf

La construcción del archivo `.ucf` se muestra en el Script 6.7.

```
1 net A(1) loc="P54"; # pulsador switch 1
2 net A(0) loc="P58"; # pulsador switch 2
3 net B(1) loc="P61"; # pulsador switch 3
4 net B(0) loc="P63"; # pulsador switch 4
5 net Sum(2) loc="P41"; # Led Sum2
6 net Sum(1) loc="P36"; # Led Sum1
7 net Sum(0) loc="P34"; # Led Sum0
```

Script 6.7: Archivo ucf

6.4.1.8 Paso 8: Agregar las fuentes necesarias al archivo Makefile

Antes de copiar el archivo `Makefile` a la carpeta de trabajo, es necesario editarlo en la segunda línea, donde debe agregarse el nombre del fichero que se desea implementar, además de esta modificación, en una descripción estructural es necesario agregar todas las fuentes por las que está compuesto el archivo principal, es decir: `semisumador.vhd` y `sumadorcompleto.vhd`.

6.4.1.9 Paso 9: Implementación

Ya teniendo los archivos necesarios para la respectiva implementación, es preciso llevar a cabo dicho proceso, para lo cual es necesario realizar los pasos mostrados a continuación:

- Comprobar la existencia de los archivos necesarios para sintetizar e implementar
  - `Makefile`
  - `semisumador.vhd`
  - `sumadorcompleto.vhd`
  - `sumador_dos_bits.vhd`
  - `sumador_dos_bits.ucf`

- Conectar la tarjeta SIE
- Configuración del puerto USB
- Realizar la síntesis de la descripción
- Enviar información al procesador
- Acceder al procesador de la tarjeta
- Enviar información al FPGA

Es preciso mencionar que si se tiene alguna duda para llevar a cabo el proceso mencionado anteriormente, es necesario referirse a la guía de laboratorio 1, en donde se explica detalladamente cada uno de los anteriores.

6.5 RESUMEN

En el laboratorio realizado se implementó paso a paso una descripción estructural de un sumador de dos bits, compuesto por un `semisumador`, un `sumador completo` y una compuerta `or`.

6.6 PREGUNTAS DE PRUEBA

- ¿Cuál es el objetivo de hacer descripciones estructurales?
- ¿Qué es un diagrama RTL?
- ¿Puede haber una descripción estructural dentro de otra descripción estructural?

6.7 EJERCICIO PROPUESTO

Agregar a la descripción estructural, un módulo que permita visualizar el resultado de la operación en un display 7 segmentos con la respectiva representación decimal.

LABORATORIO **7**

ALU.

Hacer es la mejor forma de decir.

JOSÉ MARTÍ

7.1 INTRODUCCIÓN

Para entender el funcionamiento de los sistemas digitales, es imprescindible tener conocimientos acerca de las operaciones elementales de la aritmética binaria, las cuales se refieren a la suma, la resta, la multiplicación y la división. Dentro de la suma y la resta binaria, cabe mencionar que para la primera es posible trabajar con números naturales; pero para restar aparece la necesidad de manipular números negativos, para lo cual se puede usar la aritmética en complemento a 2.

Así pues, una computadora debe manipular tanto números positivos como negativos, y es de gran importancia aprender los conceptos básicos y entender el proceso que se lleva a cabo a la hora de realizar operaciones tan elementales como la suma y la resta.

7.2 OBJETIVOS

- Describir el funcionamiento básico de una ALU por medio del lenguaje VHDL.
- Describir el funcionamiento de un sumador y un restador binario utilizando VHDL.
- Representar un resultado de una operación aritmética mediante un signo y una magnitud.
- Implementar una ALU sumador/restador en la tarjeta SIE.

7.3 MARCO TEÓRICO

7.3.1 Unidad aritmético lógica

La ALU es el elemento clave de procesamiento del microprocesador. Está gobernada por la unidad de control para realizar operaciones aritméticas como la suma y la sustracción, así como operaciones lógicas como NOT, AND, OR, y OR-exclusiva. [4]

7.3.2 Suma

En la suma, los dos números a sumar se denominan `sumandos`, y al resultado se le llama `suma`. Para realizar sumas de números con más de un bit, se conectan varios sumadores completos en paralelo, dependiendo de la cantidad de bits de los sumandos.

7.3.2.1 Sumadores binarios en paralelo

Para formar un sumador binario en paralelo se conectan dos o mas sumadores completos, se necesita un sumador completo por cada bit que tengan los números que se quieren sumar. Así, para números de dos bits se necesitan dos sumadores; para números de cuatro bits hacen falta cuatro sumadores, y así sucesivamente [4].

En la figura 7.1 se puede notar la conexión que se debe hacer entre sumadores completos, para realizar una suma de dos números, cada uno de dos bits.

$$\begin{array}{cccc} & & A_1 & A_0 \\ + & & B_1 & B_0 \\ \hline \sum_2 & \sum_1 & \sum_0 \end{array}$$

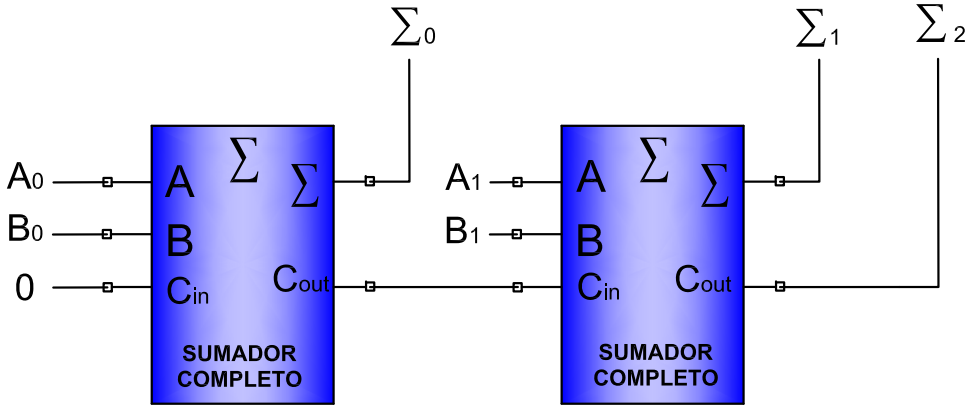


Figura 7.1: Diagrama de bloques de un sumador paralelo de 2 bits.

FUENTE:Los autores

7.3.3 RESTA

La resta es una operación aritmética a la que se asocian 3 términos: minuendo, sustraendo y diferencia tal como se muestra en la tabla 7.1.

	A_1	A_0	Minuendo
-	B_1	B_0	Sustraendo
$\sum_2$	$\sum_1$	$\sum_0$	Diferencia

Tabla 7.1: Partes de la resta.

La resta es considerada un caso especial de la suma, pues simplemente se hace una suma después de cambiar de signo al sustraendo. Así pues, para cambiar de signo un número, es preciso tener la representación negativa del mismo, para lo cual se trabaja comúnmente con el complemento a 1 y 2 de los números binarios.

7.3.3.1 Complemento a 1

El complemento a 1 de un número binario se obtiene cambiando todos los `1s` por `0s`, y todos los `0s` por `1s` [4]. La forma más sencilla de obtener el complemento a 1 de un número binario mediante un circuito digital, es utilizar inversores para cada uno de los bits que conforman el número, tal como se muestra en la figura 7.2 para un número de 8 bits.

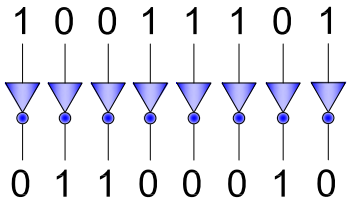


Figura 7.2: Inversores para obtener el complemento a 1 de un número.

FUENTE:Los autores

7.3.3.2 Complemento a 2

Una de las maneras para obtener el complemento a 2 de un número binario es sumando 1 al LSB del complemento a 1. El proceso a llevar a cabo se puede ver en la tabla 7.2.

7.3.4 Casos de la suma y la resta

Dentro de las operaciones aritméticas básicas se pueden mencionar la suma, la resta, la multiplicación y la división; haciendo énfasis en las dos primeras, cabe resaltar que la suma es conmutativa, pues $A + B = B + A$, mientras que

	1 0 1 1 0 0 1 0	Número Binario
	0 1 0 0 1 1 0 1	Complemento a 1
+	1	Se suma 1
	0 1 0 0 1 1 1 0	Complemento a 2

Tabla 7.2: Proceso para hallar el complemento a 2.

la resta no posee dicha propiedad, debido a que $A - B \neq B - A$. Así pues, para llevar a cabo los procesos básicos de una ALU, es preciso evaluar 3 casos, uno referente a la suma y dos referentes a la resta:

- $A + B$
- $A - B$ cuando $A > B$
- $A - B$ cuando $A < B$

Ahora bien, es preciso profundizar en cada caso para saber con exactitud las condiciones y parámetros a tener en cuenta, para ello se muestran algunos ejemplos con números de 3 bits.

7.3.4.1 Caso A+B

Para este caso, tanto A como B son números positivos, por lo tanto la suma es positiva, donde debido a que son números de 3 bits, el máximo resultado que se puede obtener es 14, y el mínimo 0.

0 1 1	3	1 1 1	7
+ 1 0 1	+ 5	+ 1 0 1	+ 5
1 0 0 0	8	1 1 0 0	1 2

Tabla 7.3: Ejemplos de suma binaria para números de tres bits.

7.3.4.2 Caso A-B cuando A es mayor que B

En este caso, el minuendo es mayor que el sustraendo, luego la diferencia es positiva. Al realizar el proceso completo de la resta, se obtiene un acarreo el cual no se tiene en cuenta. El proceso a seguir para la operación "$7 - 5 = 2$" es el que se muestra en la tabla 7.4.

7 →	1 1 1 →	1 1 1	
- 5 →	- 1 0 1 →	+ 0 1 1	
		1 0 1 0 →	0 1 0 → 2

Tabla 7.4: Pasos a seguir para restar A-B cuando A es mayor que B.

7.3.4.3 Caso A-B cuando A es menor que B

En este caso, el minuendo es menor que el sustraendo, luego el resultado de la diferencia es de signo negativo. Al finalizar el proceso de la resta, el resultado se obtiene en complemento a 2, luego para saber su magnitud es preciso obtener nuevamente el complemento a 2 de la diferencia.

1	→	0	0	1	→			0	0	1				
- 6	→	-	1	1	0	→	+	0	1	0				
								0	1	1	→	1	0	1 → 5

Tabla 7.5: Pasos a seguir para restar A-B cuando A es menor que B.

En la tabla 7.5 se muestra el proceso a llevar a cabo cuando se presenta este caso, cabe resaltar que la diferencia se obtiene en complemento a 2 debido a que es un número negativo, si se desea saber la magnitud de la misma, simplemente se aplica nuevamente el complemento a 2 a la diferencia, y se obtiene la magnitud deseada. Ahora bien, para expresar la magnitud que se obtuvo en la tabla 7.5, es preciso anteceder dicho resultado con un signo menos, pues el número no debe perder sus propiedades al ser representado de una u otra forma, una de las formas de hacer esto es a través del sistema signo-magnitud.

7.3.5 Sistema signo-magnitud

Cuando un número binario con signo se representa en formato signo-magnitud, el bit más a la izquierda es el bit de signo y los bits restantes son los bits de magnitud. Los bits de magnitud son el número binario real (no complementado) tanto para los números positivos como para los negativos [4].

Ahora bien, para representar un número de 3 bits por medio del sistema signo-magnitud, es preciso agregar un cuarto bit el cual tendrá el valor `0` cuando el número sea positivo y `1` en caso contrario.

En la tabla 7.6 se muestran algunas representaciones para números de 4 bits.

0000	→	+0
0010	→	+2
0100	→	+4
1100	→	-4
1111	→	-7

Tabla 7.6: Representación en el sistema signo-magnitud.

7.4 PROCEDIMIENTO GENERAL

Se propone realizar la descripción de una ALU y su respectiva implementación para que cumpla con las siguientes condiciones:

- ✔ Debe tener como entradas dos números de tres bits cada uno, representando dos magnitudes.
- ✔ Debe tener una entrada que indique la operación a realizar: suma ó resta.
- ✔ Debe tener una salida de cinco bits, uno para expresar el signo y cuatro para representar la magnitud del resultado.
- ✔ La magnitud del resultado debe verse en un display de 7/SEG expresada en hexadecimal.

Para llevar a cabo el proceso planteado, se recomienda seguir los pasos mostrados a continuación:

- ☞ Identificación de entradas y salidas
- ☞ Plantear el RTL de la suma A+B
- ☞ Plantear RTL del complemento a 2
- ☞ Plantear el RTL de la resta A-B para cuando $A > B$
- ☞ Plantear el RTL de la resta A-B para cuando $A < B$
- ☞ Realizar el esquema RTL de la ALU
- ☞ Describir en VHDL los módulos internos a utilizar
- ☞ Hacer la descripción estructural del circuito a implementar
- ☞ Simulación
- ☞ Identificación de dispositivos requeridos para la implementación
- ☞ Asignación de pines creando el archivo de extensión .ucf
- ☞ Agregar las fuentes necesarias al archivo Makefile
- ☞ Implementación

7.4.1 PROCEDIMIENTO PASO A PASO

Teniendo una idea general de las especificaciones del problema y de los pasos a seguir, se procede a realizar un análisis detallado del mismo.

7.4.1.1 Paso 1: Identificación de entradas y salidas

En la caja representativa de la figura 7.3 es preciso notar tres entradas, dos de tres bits haciendo referencia a las dos magnitudes, y una de un bit indicando la operación a realizar. De igual manera aparecen dos salidas, una de cinco bits representando el bit de signo mas los cuatro bits de magnitud, y una salida de 7 bits, la cual representa cada uno de los segmentos del display 7/SEG.

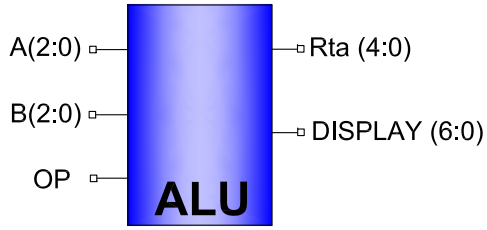


Figura 7.3: Entradas y salidas de una ALU.

FUENTE:Los autores

7.4.1.2 Paso 2: Plantear el RTL de la suma A+B

Para llevar a cabo la suma de dos números cada uno de tres bits, se plantea la conexión de tres sumadores completos en paralelo tal como se muestra en la figura 7.4.

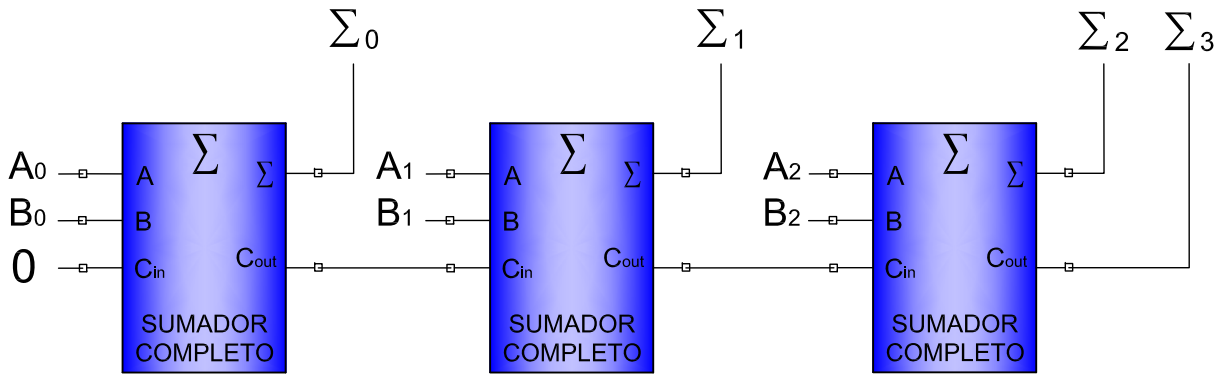


Figura 7.4: RTL de un sumador de 3 bits.

FUENTE:Los autores

7.4.1.3 Paso 3: Plantear RTL del complemento a 2

Para obtener el complemento a 2 de un número se plantea el esquema mostrado en la figura 7.5.

7.4.1.4 Paso 4: Plantear el RTL de la resta A-B para cuando A es mayor que B

Para llevar a cabo la resta, específicamente cuando el minuendo es mayor que el sustraendo, se plantea el RTL mostrado en la figura 7.6.

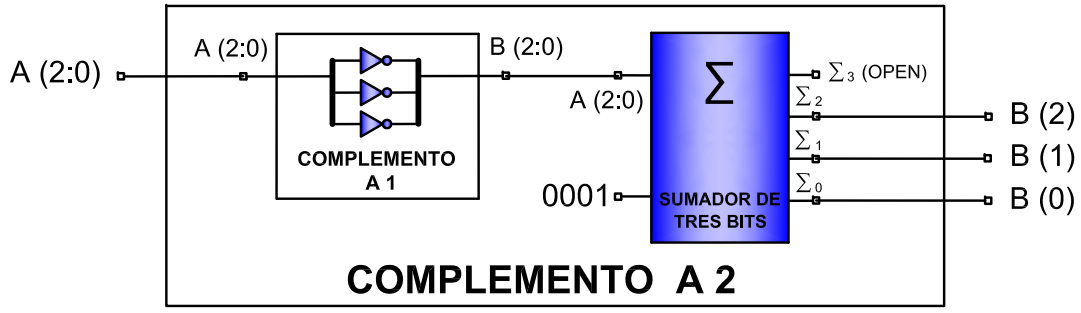


Figura 7.5: RTL del complemento a 2.

FUENTE:Los autores

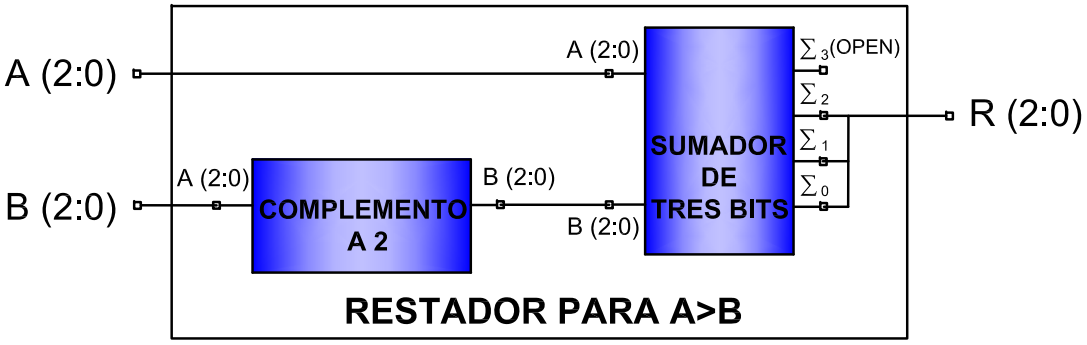


Figura 7.6: RTL de la resta cuando el minuendo es mayor que el sustraendo.

FUENTE:Los autores

7.4.1.5 Paso 5: Plantear el RTL de la resta A-B para cuando A es menor que B

Para llevar a cabo el proceso de la resta en el caso en el que el minuendo es menor que el sustraendo se plantea el RTL mostrado en la figura 7.7.

7.4.1.6 Paso 6: Realizar el esquema RTL de la ALU

Para la descripción de la ALU, se plantea el esquema RTL mostrado en la figura 7.8. En este módulo se agregan las 3 operaciones a realizar por el mismo, y un bloque en el que por medio de sentencias condicionales se procede a elegir el signo y el resultado de la operación que se debe ver reflejada en la salida de acuerdo a la entrada `OP` y las magnitudes de `A` y de `B`.

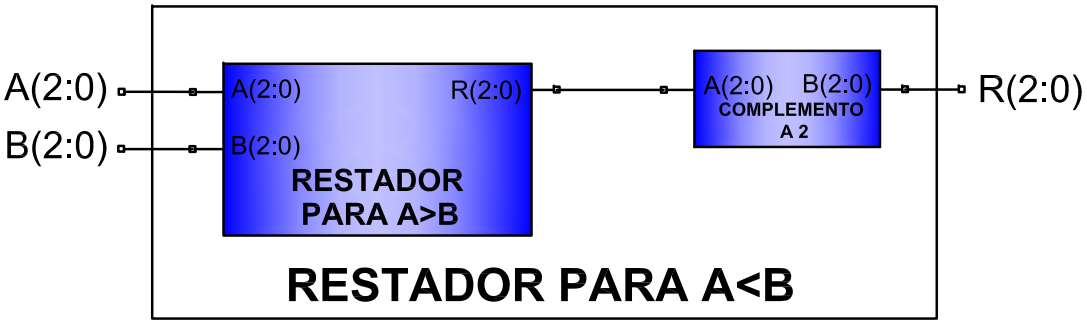


Figura 7.7: RTL de la resta cuando el minuendo es menor que el sustraendo.

FUENTE:Los autores

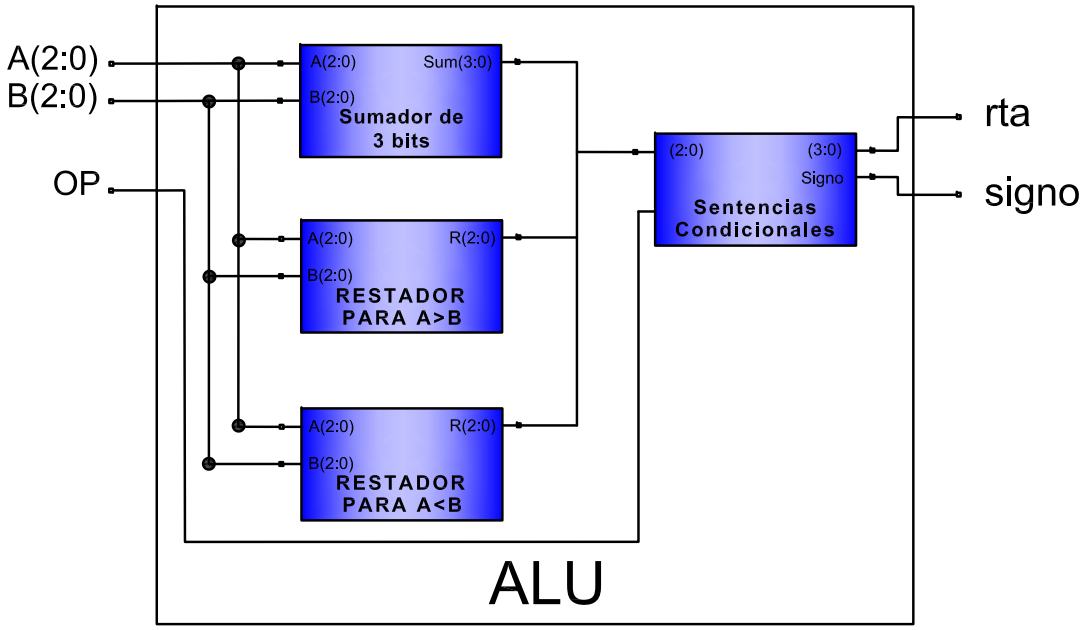


Figura 7.8: RTL de la ALU

FUENTE:Los autores

7.4.1.7 Paso 7: Describir en VHDL los módulos internos a utilizar

Inicialmente en el Script 7.1 se muestra la descripción del semisumador, necesario para la descripción del sumador completo.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity semisumador is
8  Port ( A : in STD_LOGIC;
9                         B : in STD_LOGIC;
10                        Suma : out STD_LOGIC;
11                        Cout : out STD_LOGIC);
12 end semisumador;
13 ----------------------------------------------------------------------------------
14 architecture Behavioral of semisumador is
15 begin
16 Suma<=A xor B;
17 Cout<=A and B;
18 end Behavioral;
19 ----------------------------------------------------------------------------------
```

Script 7.1: Descripción en VHDL del semisumador.

En el Script 7.2 se muestra la descripción del sumador completo, para lo cual fue necesario hacer uso del componente `semisumador`. El sumador completo es necesario para la descripción del sumador de 3 bits.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity sumadorcompleto is
8
9  Port ( A : in STD_LOGIC;
10        B : in STD_LOGIC;
11        Cin : in STD_LOGIC;
12        SUMA : out STD_LOGIC;
13        Cout : out STD_LOGIC);
14
15 end sumadorcompleto;
16 ----------------------------------------------------------------------------------
17 architecture Behavioral of sumadorcompleto is
18 ------------------------------------COMPONENTES-----------------------------------
19 component semisumador
20     Port (      A : in STD_LOGIC;
21                 B : in STD_LOGIC;
22                 Suma : out STD_LOGIC;
23                 Cout : out STD_LOGIC);
```

```
24 end component;
25 ------------------------------------DECLARACION DE SEÑALES-------------------------------------
26 signal signal_suma:std_logic;
27 signal signal_1_or:std_logic;
28 signal signal_2_or:std_logic;
29 -----------------------------------------------------------------------------------------------
30 begin
31 ------------------------------------INSTANCIAS-------------------------------------------------
32 inst_semisumador_1:semisumador
33         port map (      A=>A,
34                         B=>B,
35                         Suma=>signal_suma,
36                         Cout=>signal_1_or);
37
38 inst_semisumador_2:semisumador
39         port map (      A=>signal_suma,
40                         B=>Cin,
41                         Suma=>SUMA,
42                         Cout=>signal_2_or);
43 -----------------------------------------------------------------------------------------------
44 Cout<=(signal_1_or)or(signal_2_or);
45 end Behavioral;
46 -----------------------------------------------------------------------------------------------
```

Script 7.2: Descripción en VHDL del sumador completo.

En el Script 7.3 se muestra la descripción del sumador de 3 bits, compuesto de 3 sumadores completos conectados en paralelo. Este circuito es necesario para la descripción del complemento a 2, del restador y de la ALU. Para mayor claridad ver el RTL de la figura 7.4.

```
1  -----------------------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  -----------------------------------------------------------------------------------------------
7  entity sumador3bits is
8
9  Port (  A : in  STD_LOGIC_vector(2 downto 0);
10         B : in  STD_LOGIC_vector(2 downto 0);
11         Sum  : out  STD_LOGIC_vector(3 downto 0));
12
13 end sumador3bits;
14
15
16 architecture Behavioral of sumador3bits is
17
18 ------------------------------------COMPONENTES------------------------------------------------
19 component sumadorcompleto
20     Port (      A : in  STD_LOGIC;
21                 B : in  STD_LOGIC;
22                 Cin : in  STD_LOGIC;
23                 SUMA : out  STD_LOGIC;
```

```
24                  Cout : out  STD_LOGIC);
25 end component;
26 ----------------------------------DECLARACION DE SEÑALES-----------------------------------
27 signal signal_acarreo:std_logic_vector(1 downto 0);
28 -------------------------------------------------------------------------------------------
29
30 begin
31
32 ------------------------------------------INSTANCIAS-------------------------------------
33
34 inst1_sumadorcompleto:sumadorcompleto
35         port map (      A=>A(0),
36                         B=>B(0),
37                         Cin=>'0',
38                         SUMA=>Sum(0),
39                         Cout=>signal_acarreo(0));
40
41
42 inst2_sumadorcompleto:sumadorcompleto
43         port map (      A=>A(1),
44                         B=>B(1),
45                         Cin=>signal_acarreo(0),
46                         SUMA=>Sum(1),
47                         Cout=>signal_acarreo(1));
48
49
50 inst3_sumadorcompleto:sumadorcompleto
51         port map (      A=>A(2),
52                         B=>B(2),
53                         Cin=>signal_acarreo(1),
54                         SUMA=>Sum(2),
55                         Cout=>Sum(3));
56
57
58 end Behavioral;
```

Script 7.3: Descripción en VHDL del sumador de 3 bits.

En el Script 7.4 se muestra la descripción del complemento a 1, la cual se hace simplemente negando la entrada. Este circuito es necesario para la descripción del complemento a 2.

```
1  -------------------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  -------------------------------------------------------------------------------------------
7  entity complementoa1de3bits is
8
9  Port (  A : in  STD_LOGIC_VECTOR(2 downto 0);
10         B : out STD_LOGIC_VECTOR(2 downto 0));
11
12 end complementoa1de3bits;
```

```
13 ----------------------------------------------------------------------------------
14 architecture Behavioral of complementoa1de3bits is
15 ----------------------------------------------------------------------------------
16 begin
17   B<=not A;
18 end Behavioral;
```

Script 7.4: Descripción en VHDL del complemento a 1.

En el Script 7.5 se muestra la descripción del complemento a 2, el cual tiene dentro de sus componentes el complemento a 1 y el sumador de 3 bits. Este circuito es utilizado para llevar a cabo la descripción del restador, tanto para el caso en el que el minuendo es mayor que el sustraendo como para en el que el minuendo es menor que el sustraendo. Para mayor claridad ver el RTL de la figura 7.5.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity complementoa2de3bits is
8
9  Port (  A : in  STD_LOGIC_vector(2 downto 0);
10         B : out STD_LOGIC_vector(2 downto 0));
11
12 end complementoa2de3bits;
13
14 architecture Behavioral of complementoa2de3bits is
15
16 ------------------------------COMPONENTES-----------------------------------------
17 component complementoa1de3bits
18     Port (      A : in  STD_LOGIC_VECTOR(2 downto 0);
19                 B : out  STD_LOGIC_VECTOR(2 downto 0));
20 end component;
21
22 component sumador3bits
23     Port (      A : in  STD_LOGIC_vector(2 downto 0);
24                 B : in  STD_LOGIC_vector(2 downto 0);
25                 Sum  : out  STD_LOGIC_vector(3 downto 0));
26 end component;
27 ------------------------------DECLARACION DE SEÑALES------------------------------
28 signal signal_complemento:std_logic_vector(2 downto 0);
29 signal signal_open:std_logic;
30 ----------------------------------------------------------------------------------
31 begin
32 ----------------------------------------------------------------------------------
33
34 inst_complementoa1de3bits:complementoa1de3bits
35         port map (      A=>A,
36                         B=>signal_complemento);
37
38 inst_sumador3bits:sumador3bits
```

```
39          port map (      A=>signal_complemento,
40                          B=>"001",
41                          Sum(3)=>signal_open,
42                          Sum(2)=>B(2),
43                          Sum(1)=>B(1),
44                          Sum(0)=>B(0));
45
46  end Behavioral;
```

Script 7.5: Descripción en VHDL del complemento a 2.

En el Script 7.6 se muestra la descripción del restador para cuando el minuendo es mayor que el sustraendo, el cual está compuesto por el complemento a 2 y el sumador de 3 bits. Este circuito es necesario para la descripción del restador para el caso en el que el minuendo es mayor que el sustraendo, y para la descripción de la ALU. Para mayor claridad ver el RTL de la figura 7.6.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity restamayorb is
8
9  Port ( A : in  STD_LOGIC_vector(2 downto 0);
10        B : in  STD_LOGIC_vector(2 downto 0);
11        resta:out STD_LOGIC_VECTOR(2 downto 0));
12
13 end restamayorb;
14
15 ----------------------------------------------------------------------------------
16 architecture Behavioral of restamayorb is
17 ----------------------------------COMPONENTES-------------------------------------
18 component complementoa2de3bits
19                Port ( A : in  STD_LOGIC_vector(2 downto 0);
20                       B : out STD_LOGIC_vector(2 downto 0));
21 end component;
22
23 component sumador3bits
24                Port ( A : in  STD_LOGIC_vector(2 downto 0);
25                       B : in  STD_LOGIC_vector(2 downto 0);
26                       Sum  : out  STD_LOGIC_vector(3 downto 0));
27 end component;
28 ------------------------------DECLARACION DE SEÑALES-----------------------------
29 signal signal_complementoa2:std_logic_vector(2 downto 0);
30 signal signal_open:std_logic;
31 ----------------------------------------------------------------------------------
32 begin
33 ---------------------------------INSTANCIAS---------------------------------------
34 inst_complementoa2de3bits:complementoa2de3bits
35          port map (      A=>B,
36                          B=>signal_complementoa2);
```

```
37
38 inst_sumador3bits:sumador3bits
39         port map (       A=>A,
40                          B=>signal_complementoa2,
41                          Sum(3)=>signal_open,
42                          Sum(2)=>resta(2),
43                          Sum(1)=>resta(1),
44                          Sum(0)=>resta(0));
45
46 ----------------------------------------------------------------------------------
47 end Behavioral;
```

Script 7.6: Descripción en VHDL del restador para A>B.

En el Script 7.7 se muestra la descripción del circuito restador para el caso en el que el minuendo es menor que el sustraendo, está compuesto por el restador para el caso en el que el minuendo es mayor que el sustraendo y el módulo de complemento a 2. Este circuito es utilizado en la descripción de la ALU. Para mayor claridad ver el RTL de la figura 7.7.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity restamenorb is
8
9  Port ( A : in  STD_LOGIC_vector(2 downto 0);
10        B : in  STD_LOGIC_vector(2 downto 0);
11        resta:out STD_LOGIC_VECTOR(2 downto 0));
12
13 end restamenorb;
14
15 ----------------------------------------------------------------------------------
16 architecture Behavioral of restamenorb is
17 ------------------------------COMPONENTES-----------------------------------------
18 component restamayorb
19                Port ( A : in  STD_LOGIC_vector(2 downto 0);
20                       B : in  STD_LOGIC_vector(2 downto 0);
21                       resta:out STD_LOGIC_VECTOR(2 downto 0));
22 end component;
23
24 component complementoa2de3bits
25                Port ( A : in  STD_LOGIC_vector(2 downto 0);
26                       B : out STD_LOGIC_vector(2 downto 0));
27 end component;
28 ---------------------------DECLARACION DE SEÑALES---------------------------------
29 signal signal_resta:std_logic_vector(2 downto 0);
30 ----------------------------------------------------------------------------------
31 begin
32 ------------------------------INSTANCIAS------------------------------------------
33 inst_restamayorb:restamayorb
```

```
34          port map (      A=>A,
35                          B=>B,
36                          resta=>signal_resta);
37
38  inst_complementoa2de3bits:complementoa2de3bits
39          port map (      A=>signal_resta,
40                          B=>resta);
41
42  ----------------------------------------------------------------------------
43  end Behavioral;
```

Script 7.7: Descripción en VHDL del restador para A<B.

7.4.1.8 Paso 8: Hacer la descripción estructural del circuito a implementar

En el Script 7.8 se muestra la descripción de la ALU, la cual está compuesta por tres módulos principalmente, el sumador, el restador para cuando el minuendo A es mayor que el sustraendo B, y el restador para cuando el minuendo A es menor que el sustraendo B. Además, tiene un bloque en el que través de sentencias condicionales dentro de un `process`, se eligen las salidas a mostrar dependiendo de las respectivas entradas del circuito. Para mayor claridad ver el RTL de la figura 7.8.

```
1  ----------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------
7  entity ALU is
8
9  Port (  A : in  STD_LOGIC_vector(2 downto 0);
10         B : in  STD_LOGIC_vector(2 downto 0);
11         OP: in  STD_LOGIC;
12         signo:out std_logic;
13         resp:out STD_LOGIC_VECTOR(3 downto 0));
14 end ALU;
15
16 ----------------------------------------------------------------------------
17 architecture Behavioral of ALU is
18 ----------------------------COMPONENTES-------------------------------------
19 component sumador3bits
20                 Port (  A : in  STD_LOGIC_vector(2 downto 0);
21                         B : in  STD_LOGIC_vector(2 downto 0);
22                         Sum  : out  STD_LOGIC_vector(3 downto 0));
23 end component;
24
25 component restamayorb
26                 Port (  A : in  STD_LOGIC_vector(2 downto 0);
27                         B : in  STD_LOGIC_vector(2 downto 0);
28                         resta:out STD_LOGIC_VECTOR(2 downto 0));
29 end component;
```

```
30
31  component restamenorb
32                  Port (  A : in  STD_LOGIC_vector(2 downto 0);
33                          B : in  STD_LOGIC_vector(2 downto 0);
34                          resta:out STD_LOGIC_VECTOR(2 downto 0));
35  end component;
36  ------------------------------------DECLARACION DE SEÑALES-------------------------------------
37  signal signal_suma:std_logic_vector(3 downto 0);
38  signal signal_amayorb:std_logic_vector(2 downto 0);
39  signal signal_amenorb:std_logic_vector(2 downto 0);
40  -----------------------------------------------------------------------------------------------
41  begin
42  ------------------------------------------INSTANCIAS-------------------------------------------
43  inst_sumador3bits:sumador3bits
44          port map (      A=>A,
45                          B=>B,
46                          Sum=>signal_suma);
47
48  inst_restamayorb:restamayorb
49          port map (      A=>A,
50                          B=>B,
51                          resta=>signal_amayorb);
52
53  inst_restamenorb:restamenorb
54          port map (      A=>A,
55                          B=>B,
56                          resta=>signal_amenorb);
57  -----------------------------------------------------------------------------------------------
58  process (A,B,signal_suma,signal_amayorb,signal_amenorb,OP)
59  begin
60
61          if (OP='0') then
62                  resp<=signal_suma;
63                  signo<='0';
64          else
65
66                  if (A>=B) then
67                          resp(3)<='0';
68                          resp(2)<=signal_amayorb(2);
69                          resp(1)<=signal_amayorb(1);
70                          resp(0)<=signal_amayorb(0);
71                          signo<='0';
72
73                  else
74
75                          resp(3)<='0';
76                          resp(2)<=signal_amenorb(2);
77                          resp(1)<=signal_amenorb(1);
78                          resp(0)<=signal_amenorb(0);
79                          signo<='1';
80                  end if;
81          end if;
82  end process;
```

```
83 ----------------------------------------------------------------------------------
84 end Behavioral;
```

Script 7.8: Descripción en VHDL de la ALU.

En el Script 7.9 se muestra la descripción de la ALU, conectando un decodificador a la salida para así poder ver el resultado en el display de 7 segmentos, tal como lo requiere el planteamiento del problema.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity ALUyDECO is
8
9  Port (  A : in  STD_LOGIC_vector(2 downto 0);
10         B : in  STD_LOGIC_vector(2 downto 0);
11         OP: in  STD_LOGIC;
12         signo:out std_logic;
13         resp:out STD_LOGIC_VECTOR(3 downto 0);
14         display:out STD_LOGIC_VECTOR(6 downto 0));
15 end ALUyDECO;
16
17 ----------------------------------------------------------------------------------
18 architecture Behavioral of ALUyDECO is
19 ------------------------------------COMPONENTES-----------------------------------
20 component ALU
21                 Port (  A : in  STD_LOGIC_vector(2 downto 0);
22                         B : in  STD_LOGIC_vector(2 downto 0);
23                         OP: in  STD_LOGIC;
24                         signo:out std_logic;
25                         resp:out STD_LOGIC_VECTOR(3 downto 0));
26 end component;
27
28
29 component deco_7seg_WS
30                 Port (  A : in  STD_LOGIC_VECTOR (3 downto 0);
31                         P : out  STD_LOGIC_VECTOR (6 downto 0));
32 end component;
33 ------------------------------DECLARACION DE SEÑALES------------------------------
34 signal signal_alu:std_logic_vector(3 downto 0);
35 ----------------------------------------------------------------------------------
36 begin
37 ---------------------------------------INSTANCIAS---------------------------------
38 inst_ALU:ALU
39         port map (      A=>A,
40                         B=>B,
41                         OP=>OP,
42                         signo=>signo,
43                         resp=>signal_alu);
44
45 inst_deco_7seg_WS:deco_7seg_WS
```

```
46          port map (      A=>signal_alu,
47                          P=>display);
48 ----------------------------------------------------------------------------------
49 resp<=signal_alu;
50 ----------------------------------------------------------------------------------
51 end Behavioral;
```

Script 7.9: Descripción en VHDL de la ALU con display.

7.4.1.9 Paso 9: Simulación

En la figura 7.9 se muestra la simulación correspondiente al complemento a 1, donde se nota el marcador en aproximadamente 325 ns, mostrando que a una entrada correspondiente de 010, el circuito responde con una salida 101, mostrando así el comportamiento esperado.

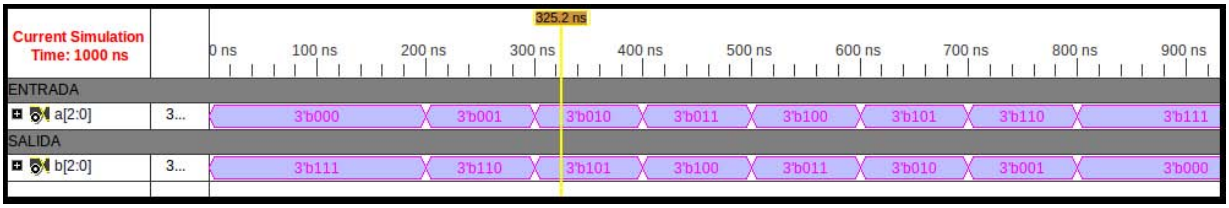


Figura 7.9: Simulación del complemento a 1.

FUENTE:Herramienta de simulación ISE

En la figura 7.10 se muestra la simulación correspondiente al complemento a 2, donde se nota el marcador en aproximadamente 425 ns, mostrando que a una entrada correspondiente de 011, el circuito responde con una salida 101, mostrando así el complemento a 2 de la entrada como era de esperarse.

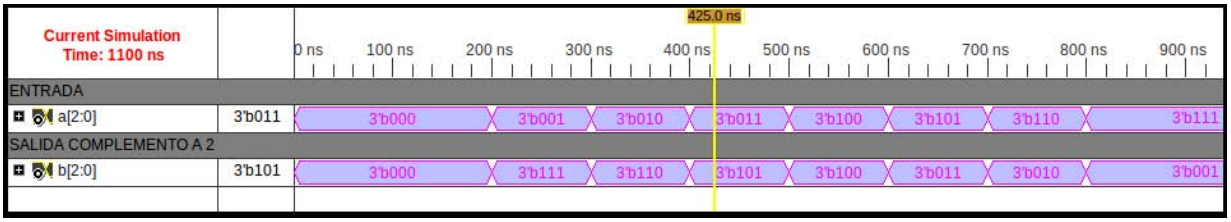


Figura 7.10: Simulación del complemento a 2.

FUENTE:Herramienta de simulación ISE

En la figura 7.11 se muestra la simulación correspondiente al sumador de 3 bits, donde se nota el marcador en aproximadamente 825 ns, mostrando que a una entrada correspondiente de $4 + 7$, el circuito responde con una salida de 11, mostrando así la suma entre los dos números y el comportamiento esperado.

En la figura 7.12 se muestra la simulación correspondiente al restador para el caso en el que el minuendo es mayor que el sustraendo, donde se nota el marcador en aproximadamente 425 ns, mostrando que a una entrada

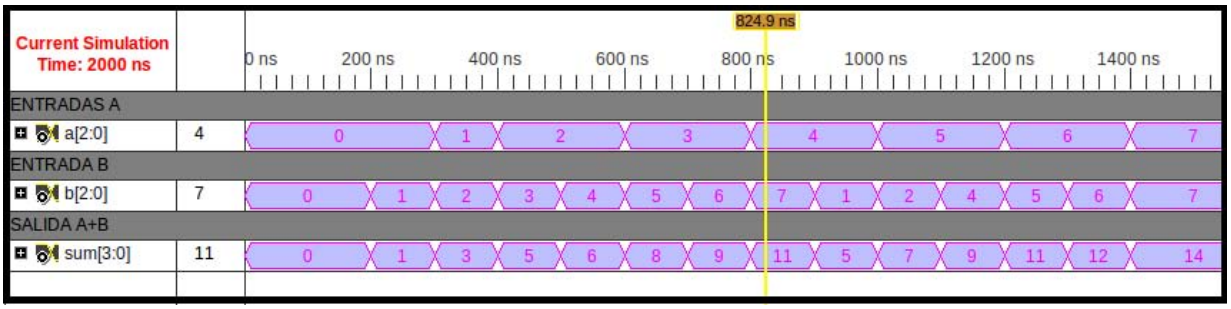


Figura 7.11: Simulación del sumador de 3 bits.

FUENTE:Herramienta de simulación ISE

correspondiente a $3 - 1$, el circuito responde con una salida igual a 2, mostrando así la diferencia de los dos números y el comportamiento esperado.

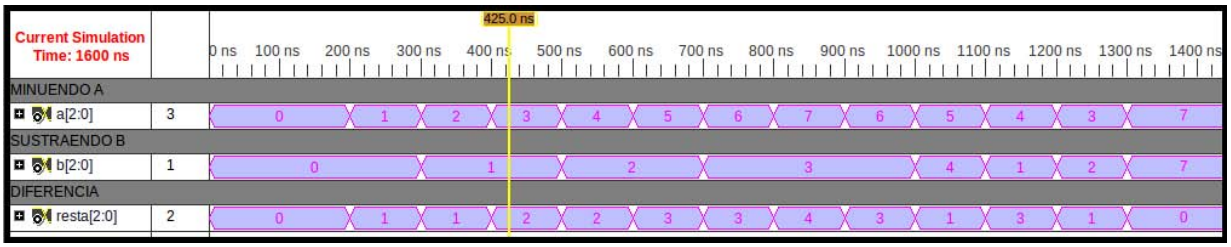


Figura 7.12: Simulación del restador para $A > B$.

FUENTE:Herramienta de simulación ISE

En la figura 7.13 se muestra la simulación correspondiente al restador para cuando el minuendo es menor que el sustraendo, donde se nota el marcador en aproximadamente 725 `ns`, mostrando que a una entrada correspondiente a $2 - 6$, el circuito responde con una salida igual a 4, mostrando así la magnitud de la salida esperada.

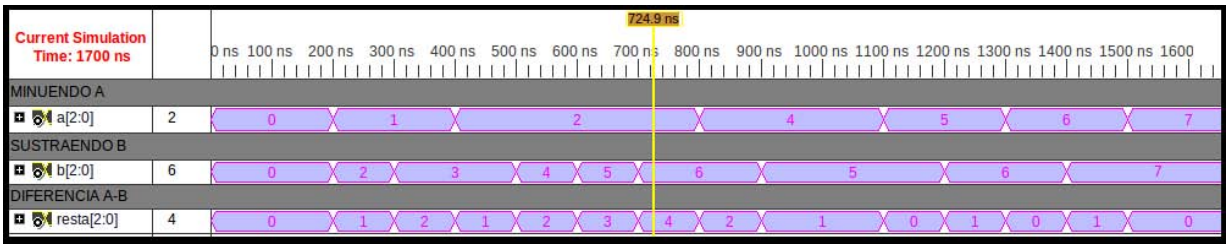


Figura 7.13: Simulación del restador para $A < B$.

FUENTE:Herramienta de simulación ISE

Finalmente en la figura 7.14 se muestra la simulación correspondiente a la ALU, donde se nota el marcador en aproximadamente 1025 `ns`, mostrando que a una entrada correspondiente de $1 - 7$, el circuito responde con una salida igual a -6, mostrando así el comportamiento esperado.

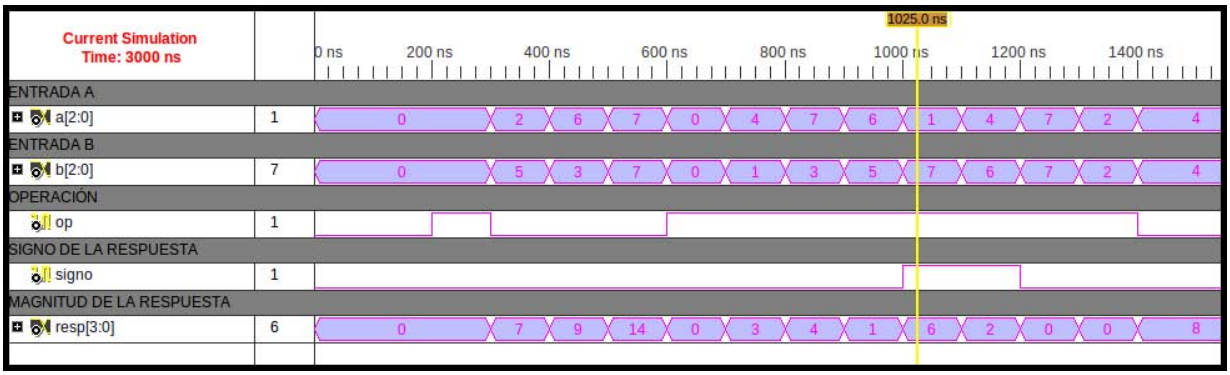


Figura 7.14: Simulación de la ALU.

FUENTE:Herramienta de simulación ISE

7.4.1.10 Paso 10: Identificación de dispositivos requeridos para la implementación

Los módulos necesarios para la implementación de la práctica son:

- 2 Digilent PmodSWT Switch Module Board. [8]
- 4 Leds de la tarjeta de expansión de pines.
- 1 Digilent PmodSSD Peripheral Module Board. [8]

7.4.1.11 Paso 11: Asignación de pines creando el archivo de extensión .ucf

La construcción del archivo `.ucf` se muestra en el Script 7.10. Los pines `P60,P62,P65` y `P70` se conecta a uno de los PmodSWT Switch Module Board, en los pines `P54,P58,P61` y `P63` se conecta el segundo PmodSWT Switch Module Board, los pines `P34,P32,P26,P23` y `P18` están asociados a 5 leds de la tarjeta de expansión de pines, y finalmente en los pines `P33,P27,P24,P22,P49.P47` y `P40` se conecta el Digilent PmodSSD Peripheral Module Board, referente al display de 7 segmentos.

```
1  net A(2) loc="P62"; # pulsador switch 1
2  net A(1) loc="P65"; # pulsador switch 2
3  net A(0) loc="P70"; # pulsador switch 3
4
5  net B(2) loc="P58"; # pulsador switch 1
6  net B(1) loc="P61"; # pulsador switch 2
7  net B(0) loc="P63"; # pulsador switch 3
8
9  net OP loc="P60"; # pulsador switch
10
11 net signo loc="P34"; # LED
12
13 net resp(3) loc="P32"; # Led
```

```
14 net resp(2) loc="P26"; # Led
15 net resp(1) loc="P23"; # Led
16 net resp(0) loc="P18"; # Led
17
18
19 net display(6) loc="P33"; # segmento a del display
20 net display(5) loc="P27"; # segmento b del display
21 net display(4) loc="P24"; # segmento c del display
22 net display(3) loc="P22"; # segmento d del display
23 net display(2) loc="P49"; # segmento e del display
24 net display(1) loc="P47"; # segmento f del display
25 net display(0) loc="P40"; # segmento g del display
```

Script 7.10: Archivo ucf

7.4.1.12 Paso 12: Agregar las fuentes necesarias al archivo Makefile

Antes de copiar el archivo `Makefile` a la carpeta de trabajo, es necesario editarlo en la segunda línea, donde debe agregarse el nombre del fichero que se desea implementar en este caso `ALUYDECO`, además de esta modificación, en una descripción estructural es necesario agregar todas las fuentes por las que está compuesto el archivo principal, es decir: `semisumador.vhd` y `sumadorcompleto.vhd`, `sumador3bits.vhd`, `complementoa1de3bits.vhd`, `complementoa2de3bits.vhd`, `restamayorb.vhd`, `restamenorb.vhd`, `ALU.vhd` y `deco_7seg_WS.vhd`.

7.4.1.13 Paso 13: Implementación

Ya teniendo los archivos necesarios para la respectiva implementación, es preciso llevar a cabo dicho proceso, para lo cual es necesario realizar los pasos mostrados a continuación:

- Comprobar la existencia de los archivos necesarios para sintetizar e implementar
  - `Makefile`
  - `semisumador.vhd`
  - `sumadorcompleto.vhd`
  - `sumador3bits.vhd`
  - `complementoa1de3bits.vhd`
  - `complementoa2de3bits.vhd`
  - `restamayorb.vhd`
  - `restamenorb.vhd`
  - `ALU.vhd`
  - `deco_7seg_WS.vhd`

  - `ALUYDECO.vhd`
  - `ALUYDECO.ucf`

- Conectar la tarjeta SIE
- Configuración del puerto USB
- Realizar la síntesis de la descripción
- Enviar información al procesador
- Acceder al procesador de la tarjeta
- Enviar información al FPGA

Es preciso mencionar que si se tiene alguna duda para llevar a cabo el proceso mencionado anteriormente, es necearía referirse a la guía de laboratorio 1, en donde se explica detalladamente cada uno de los anteriores.

7.5 RESUMEN

En la guía realizada se plantea la descripción en VHDL e implementación de una Unidad aritmético lógica (ALU), se presenta un proceso paso a paso describiendo cada uno de los componentes necesarios y las interconexiones necesarias entre los mismos, para llevar a cabo una exitosa implementación en la tarjeta SIE. En la guía se presentan los conceptos mostrados a continuación.

- Sumadores en paralelo
- Complemento a 1
- Complemento a 2
- Restador básico
- Representación de números con signo
- Unidad aritmético lógica (ALU)

7.6 PREGUNTAS DE PRUEBA

- ¿Cómo se puede implementar un sumador de 8 bits?
- ¿Cuál es la diferencia entre la representación de números en complemento a 1 y en complemento a 2?
- ¿Qué sistema utiliza una computadora para manipular números negativos?

7.7 EJERCICIO PROPUESTO

Añadir una nueva función a la ALU, la cual multiplica un número de tres bits por 3. Describirlo, agregarlo a la ALU e implementarlo en la tarjeta SIE.

Añadir un módulo a la ALU descrita en esta guía, que ito que multiple un número por 3, adicionarlo a la ALU realizada e implementarlo en la tarjeta SIE.

LABORATORIO **8**

FLIP-FLOPS.

La libertad no es poder elegir entre unas pocas opciones impuestas, sino tener el control de tu propia vida. La libertad no es elegir quien será tu amo, es no tener amo.

RICHARD MATTHEW STALLMAN

8.1 INTRODUCCIÓN

A diferencia de los circuitos combinacionales, los circuitos secuenciales necesitan de cierta lógica capaz de retener la historia del circuito. Por ello, la descripción de circuitos secuenciales usa elementos de memoria, que a través de bucles cerrados internos, permiten el almacenamiento de datos. A los elementos de memoria se les conoce como biestables, estos se dividen en *latches* y *flip-flops*.

La característica principal de un biestable (celda binaria) es mantener o almacenar un bit de manera indefinida hasta que a través de un pulso o una señal cambie de estado. Los tipos de *flip-flops* más conocidos son los SR, D, T y JK.

8.2 OBJETIVOS

- Implementar los tipos de *flip-flop* (SR, D, T y JK).
- Describir los tipos más comunes de *flip-flops* en *VHDL*.

8.3 MARCO TEÓRICO

Para empezar a desarrollar una buena descripción de un *flip-flop*, fundamentalmente se deben tener en cuenta dos factores: su comportamiento y el instante en que responde. A continuación se reforzarán estos dos conceptos.

8.3.1 Características de los *flip-flops*

8.3.1.1 Transiciones de los *flip-flops*

Si el *flip-flop* realiza la transición cuando la señal de sincronía (llamada clk) cambia de 0 a 1 se dice que éste es de **transición con pendiente positiva(TTP)**; cuando la señal de sincronía cambia de 1 a 0 se dice que este *flip-flop* es de **transición con pendiente negativa (TTN)**.

La Figura 8.1 muestra una simulación realizada a un *flip-flop* tipo D (TTP), aquí se encuentran resaltados los tiempos en los cuales puede ocurrir la transición de las salidas **q** y **nq**. Otro detalle que cabe resaltar, es que en 300 *ns* y 700 *ns* este circuito puede realizar un cambio en sus señales de salida, mas aún no lo hace, por el comportamiento del *flip-flop* tipo D.

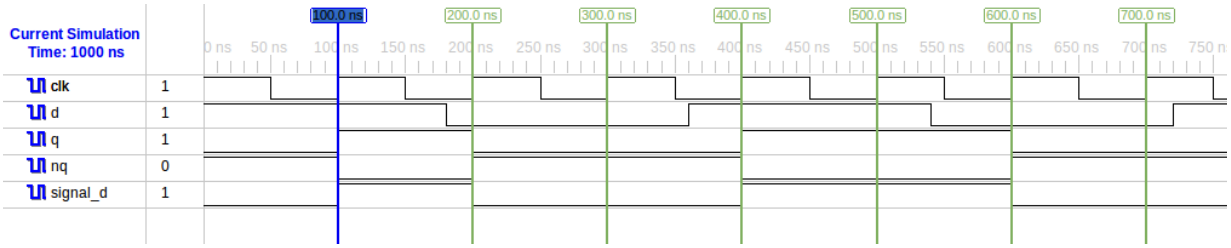


Figura 8.1: Transiciones de un *flip-flop* tipo D.

FUENTE:Herramienta de simulación ISE

No sólo un *flip-flop* hace la transición cuando hay un cambio de la señal de sincronía (clk), éste también puede realizar una transición cuando tiene entradas asíncronas que operan independientemente de las síncronas.
Las entradas asíncronas se pueden emplear para fijar la salida de un *flip-flop* en el valor de 1 o 0 *en cualquier instante, sin importar las condiciones presentes en las otras entradas.* Estas entradas asíncronas se designan como $\overline{PRESTABLECER}$ y $\overline{RESTABLECER}$, la línea en la parte superior quiere decir que estas entradas se activan por BAJO, como también se indica en la figura 8.2, con las burbujas en las entradas asíncronas del *flip-flop*. Para la nomenclatura de las entradas asíncronas se ha llegado a un acuerdo, las denominaciones más comunes son PRE (abreviatura de PRESET o PREESTABLECER) y CLR (abreviatura para CLEAR o RESTABLECER), cabe mencionar que PRE obliga a que la salida del *flip-flop* en el instante que se activa, sea 1, hasta el momento que se desactiva la señal asíncrona (PRE). En ese instante la salida puede tomar cualquier valor, dependiendo del circuito y del estado en que se encuentre; inversamente CLR obliga a que la salida en ese instante que se activa sea 0.

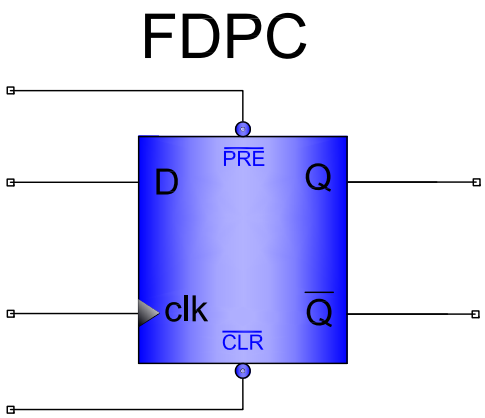


Figura 8.2: Representación del *flip-flop* tipo D, con entradas asíncronas

FUENTE:Los autores

8.3.1.2 Comportamiento de los *flip-flops*

En este punto se debe conocer el comportamiento del *flip-flop*, cuándo sus señales de entrada o estado actual tiene un valor asignado. En esta guía el comportamiento de los *flip-flops* se representará por medio de una tabla de comportamiento a medida que se van presentando cada uno de los *flip-flops*.

8.3.2 Sentencia *process*

VHDL presenta una estructura particular denominada *process*, describe un circuito que actúa, si y sólo si alguna de las señales de su lista de sensibilidad se ha modificado. Un *process* se encuentra dentro de la arquitectura, tal como se muestra en el siguiente código:

```
1
2  architecture nombre_arq of nombre_ent is
3  -- declaraciones de la arquitectura:
4  -- tipos
5  -- señales
6  -- componentes
7  begin
8  -- código de descripción
9  -- instrucciones concurrentes
10 -- ecuaciones booleanas
11 -- componentes
12 process (Lista_de_sensibilidad)
13 -- asignacion de variables
14 -- opcional no recomendable
15 begin
16 -- Sentencias condicionales
17 -- Asignaciones de valores
18 end process;
19 end nombre_arq  ;
```

Script 8.1: Entorno de un *process*.

La sentencia *process* es una de las más utilizadas en descripción de hardware con *VHDL*, ya que tanto los circuitos combinacionales realizados por las sentencias condicionales, como la descripción de hardware secuencial, se pueden llevar a cabo dentro de esta sentencia. Para aquellos que se acercan por primera vez a la síntesis con *VHDL*, este es uno de los principales problemas para realizar un correcto diseño, por esto a continuación se van a nombrar una serie de recomendaciones para el uso de la sentencia *process*. Será de necesario cumplimiento para que el código generado pueda ser sintetizado e implementado de manera correcta.

8.3.2.1 Recomendaciones para realizar una buena descripción de la sentencia *process* en *VHDL*

Siempre que se describa un circuito en *VHDL*, se debe tener presente realizar una representación en diagramas de bloques del circuito.

- En la lista de sensibilidad de los circuitos combinacionales han de incluirse al menos todas las señales que se lean dentro del *process* (señal_escritura <= señal_lectura).
  El circuito descrito en una estructura *process*, sólo modificara sus señales de salida, cuando varíe alguna de las señales de su lista de sensibilidad.

- En los circuitos combinacionales, las señales de salida de un *process* deben tener definidos todos los casos posibles, para evitar que la señal de salida tome un valor indeseado como se presenta en el *script* 8.3.

```
1  ------------------------
2  -- condicional completo
3  ------------------------
4  process (a,b)
5  begin
6    if a = b then
7      c <= a or b;
8      elsif a<b then
9        c <= b;
10     else
11       c <= '0';
12   end if;
13 end process;
14 ------------------------
```

Script 8.2: Condicional completo.

Tabla de verdad del condicional completo		
a	**b**	**c**
0	0	0
0	1	1
1	0	0
1	1	1

Tabla 8.2: condicional completo

```
1  ------------------------------
2  -- condicional incompleto
3  ------------------------------
4  process (a,b)
5  begin
6    if a = b then
7      c <= a or b;
8      elsif a<b then
9        c <= b;
10   end if;
11 end process;
12 ------------------------------
```

Script 8.3: Condicional incompleto.

Tabla de verdad del condicional incompleto		
a	**b**	**c**
0	0	0
0	1	1
1	0	-
1	1	1

Tabla 8.3: condicional incompleto

Siempre que se escriba una sentencia condicional es obligatorio asignar el valor que deben tener todas las señales en cada rama del árbol condicional. Toda condición debe tener su rama *else*.

- Los *process* son concurrentes entre si; sin embargo, si dentro del *process* se asigna valor a una señal en dos sitios diferentes el resultado será aquél de la última asignación, exactamente igual que en los lenguajes de programación comunes, por lo tanto es preciso comprobar que no se asigne el valor a una señal en dos sitios diferentes del *process*. (si puede hacerse en dos ramas diferentes del mismo condicional como ocurre en el *script* 8.2).

- Existirá con bastante probabilidad un código con dos o más *process*, en esos casos se ejecutan en paralelo como ocurre con el resto de las sentencias. Si dos *process* se están ejecutando en paralelo, no se puede modificar la misma señal en ambos *process*, porque en ese caso no se podría saber cuál es el valor real de la señal (el obtenido en el *process* 1 o el obtenido en el *process* 2). Por lo tanto no se puede sintetizar el circuito.

8.3.2.2 *Flip-flop* tipo SR

El primer *flip-flop* presentado en esta guía es el SR, el cual cumple con la tabla de característica 8.1, en ella está plasmada el comportamiento de éste. En la tabla de característica 8.1, así como en posteriores tablas se representa el flanco (transición de la señal de sincronía CLK) como una flecha, si se encuentra hacia arriba se dice que esta señal es TTP ó flanco positivo (FP), además se especifica la operación que realiza dependiendo de sus entradas.

Nota: cuando la señal de sincronía no presenta cambios (clk), el *flip-flop* no modifica sus salidas.

La representación del flip-flops SR se presenta en la Figura 8.3, las entradas en la parte izquierda y sus salidas al lado derecho.

Entadas			Salidas		Operación
Reloj (clk)	S	R	Q	$\overline{Q}$	
↑	0	0	Qo	$\overline{Qo}$	Mantener
↑	1	0	1	0	Set (establecer)
↑	0	1	0	1	Reset (reiniciar)
↑	1	1	1	1	Ambiguo (no se usa)

Tabla 8.1: Tabla de comportamiento del *flip-flop* tipo SR

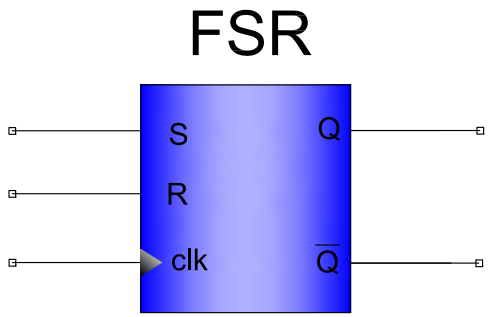


Figura 8.3: Representación del *flip-flop* tipo SR

FUENTE:Los autores

8.3.2.3 *Flip-flop* tipo D

Los *flip-flops* también puede presentar entradas asíncronas, un ejemplo de esto es el *flip-flop* tipo D con entradas $\overline{PRE}$ y $\overline{CLR}$. La tabla 8.2 muestra la característica del *flip-flop* tipo D, brinda información sobre su comportamiento.

Entadas				Salidas		Operación
$\overline{PRE}$	$\overline{CLR}$	Reloj (clk)	D	Q	$\overline{Q}$	
0	0	X	X	-	-	Ambiguo (no se usa)
0	1	X	X	1	0	Prestablecer
1	0	X	X	0	1	Restablecer
1	1	X	X	Qo	$\overline{Qo}$	Desactivar entradas asíncronas
1	1	↑	0	0	1	Reset(establecer)
1	1	↑	1	1	0	Set (reiniciar)

Tabla 8.2: Tabla de comportamiento del *flip-flop* tipo D

La representación del *flip-flop* tipo D se presenta en la figura 8.4, como se puede observar las entradas asíncronas están ubicadas en la parte inferior y superior, las entradas síncronas en la parte izquierda y la salida en la parte derecha.

> Nota: el símbolo en la parte superior e inferior en las entradas asíncronas parecido a unas burbujas señalan que éstas se activan por lógica de nivel bajo.

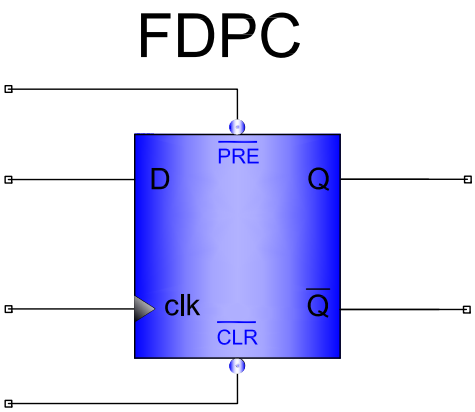


Figura 8.4: Representación del *flip-flop* tipo D, con entradas asíncronas

FUENTE:Los autores

8.3.2.4 *Flip-flop* tipo T

No sólo los *flip-flops* realizan la transición cuando hay TTP, en el *flip-flop*, realiza la transición cuando la señal de sincronía cambia de 1 a 0, (TTN) ó flanco negativo (FN). La tabla de verdad 8.3, expone el comportamiento de las salidas frente a los cambios que puede realizar las entradas.

Entadas				Salidas		Operación
$\overline{PRE}$	$\overline{CLR}$	Reloj (clk)	T	Q	$\overline{Q}$	
0	0	X	X	-	-	Ambiguo (no se usa)
0	1	X	X	1	0	Prestablecer
1	0	X	X	0	1	Restablecer
1	1	X	X	Qo	$\overline{Qo}$	Desactivar entradas asíncronas
1	1	$\downarrow$	0	Qo	$\overline{Qo}$	Mantener
1	1	$\downarrow$	1	$\overline{Qo}$	Qo	Cambiar

Tabla 8.3: Tabla de comportamiento del *flip-flop* tipo T, con entradas asíncronas

El cambio que se resalta en la Figura 8.5 es que en la entrada de la señal de sincronía se encuentra una burbuja, indicando que este flip-flop es TTN.

8.3.2.5 *Flip-flop* tipo JK

Se presenta un *flip-flop* parecido al SR, el cual cuenta con las entradas llamadas J y K, sin olvidar las entradas asíncronas y clk. En la tabla de característica 8.1 y 8.4 se ve la diferencia entre ellos, el *flip-flop* JK tiene definido el estado cuando sus entradas tienen el valor de 1 (J=1,K=1) y la operación asociada a la configuración mencionada es cambiar, ésta realiza el cambio de las salidas Q y $\overline{Qo}$.

La representación del *flip-flop* JK se encuentra en la Figura 8.6

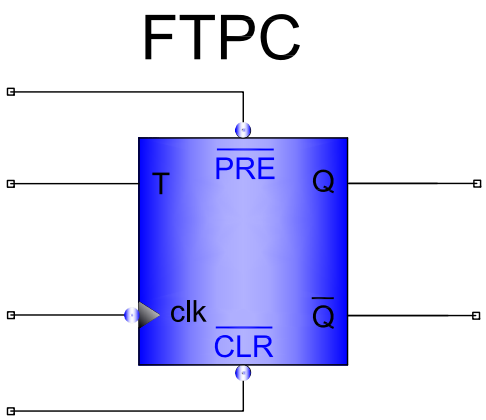


Figura 8.5: Representación del *flip-flop* tipo T, con entradas asíncronas

FUENTE:Los autores

Entadas					Salidas		Operación
$\overline{PRE}$	$\overline{CLR}$	Reloj (clk)	J	K	Q	$\overline{Q}$	
0	0	X	X	X	-	-	Ambiguo (no se usa)
0	1	X	X	X	1	0	Prestablecer
1	0	X	X	X	0	1	Restablecer
1	1	X	X	X	Qo	$\overline{Qo}$	Desactivar entradas asíncronas
1	1	↑	0	0	Qo	$\overline{Qo}$	Mantener
1	1	↑	1	0	1	0	Set (establecer)
1	1	↑	0	1	0	1	Reset (reiniciar)
1	1	↑	1	1	$\overline{Qo}$	Qo	Cambiar

Tabla 8.4: Tabla de comportamiento del *flip-flop* tipo JK, con entradas asíncronas

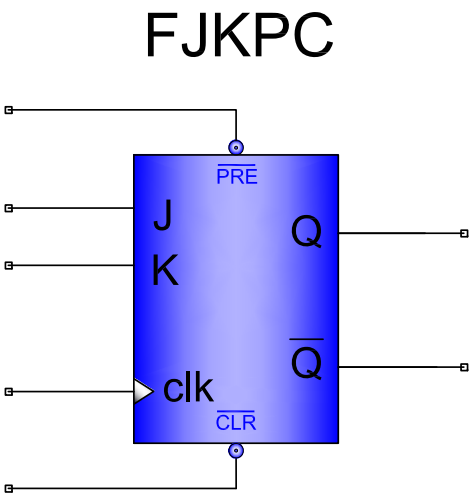


Figura 8.6: Representación del *flip-flop* tipo JK, con entradas asíncronas

FUENTE:Los autores

8.4 PROCEDIMIENTO GENERAL

Para el desarrollo de esta laboratorio se plantea realizar la descripción de los tipos de *flip-flops* en lenguaje *VHDL*(SR,D,T y JK), mostrando su correcto funcionamiento por medio de una simulación. seguido a esto se propone elegir uno de los cuatro e implementarlo en la tarjeta SIE, mostrando la señal de sincronía a través de un *LED* para notar los instantes de tiempos en los cuales el *flip-flop* puede almacenar los datos.

Los incisos que son esenciales para el procedimiento paso a paso son los siguientes:

- Plantear la tabla de característica de los *flip-flops* a describir.
- Realizar la descripción en *VHDL* de los tipos de *flip-flops*.
- Simulación de la descripción de los diferentes tipos de *flip-flops*.
- Identificación de dispositivos a utilizar.
- Asignación de pines.
- Utilización del *Makefile*.
- Implementación en la tarjeta *SIE*.

8.4.1 PROCEDIMIENTO PASO A PASO

8.4.2 Paso 1: Plantear la tabla de característica de los *flip-flops* a describir

A continuación se presentan los comportamientos de los circuitos a implementar, haciendo pequeñas variaciones en cada uno de los *Flip-flop*.

8.4.2.1 *Flip-flop* tipo SR

El la tabla 8.5 se muestra el comportamiento que presenta un *flip-flop* tipo SR.

Entadas			Salidas		Operación
Reloj (clk)	S	R	Q	$\overline{Q}$	
↑	0	0	Qo	$\overline{Qo}$	Mantener
↑	1	0	1	0	Set (establecer)
↑	0	1	0	1	Reset (reiniciar)
↑	1	1	1	1	Ambiguo (no se usa)

Tabla 8.5: Tabla de comportamiento del *flip-flop* tipo SR

Nota: cuando la señal de sincronía no presenta cambios (clk), el *flip-flop* no modifica sus salidas.

8.4.2.2 *Flip-flop* tipo D

El la tabla 8.6 se muestra el comportamiento que presenta un *flip-flop* tipo D.

Entadas				Salidas		Operación
$\overline{PRE}$	$\overline{CLR}$	Reloj (clk)	D	Q	$\overline{Q}$	
0	0	X	X	-	-	Ambiguo (no se usa)
0	1	X	X	1	0	Prestablecer
1	0	X	X	0	1	Restablecer
1	1	X	X	Qo	$\overline{Qo}$	Desactivar entradas asíncronas
1	1	$\uparrow$	0	0	1	Reset(establecer)
1	1	$\uparrow$	1	1	0	Set (reiniciar)

Tabla 8.6: Tabla de comportamiento del *flip-flop* tipo D

8.4.2.3 *Flip-flop* tipo T

El la tabla 8.7 se muestra el comportamiento que presenta un *flip-flop* tipo T.

Entadas				Salidas		Operación
$\overline{PRE}$	$\overline{CLR}$	Reloj (clk)	T	Q	$\overline{Q}$	
0	0	X	X	-	-	Ambiguo (no se usa)
0	1	X	X	1	0	Prestablecer
1	0	X	X	0	1	Restablecer
1	1	X	X	Qo	$\overline{Qo}$	Desactivar entradas asíncronas
1	1	$\downarrow$	0	Qo	$\overline{Qo}$	Mantener
1	1	$\downarrow$	1	$\overline{Qo}$	Qo	Cambiar

Tabla 8.7: Tabla de comportamiento del *flip-flop* tipo T, con entradas asíncronas

8.4.2.4 *Flip-flop* tipo JK

Finalmente en la tabla 8.8 se muestra el comportamiento que presenta un *flip-flop* tipo T.

8.4.3 Paso 2: Realizar la descripción en *VHDL* de los tipos de *flip-flops*

A continuación se va presentar una manera de realizar la descripción de hardware de cada uno de los *flip-flops*.

En la tarjeta SIE se tiene un reloj de 50 *MHz*, del cual se obtiene la señal de sincronía que necesitan los *flip-flops*, pero para poder percibir los momentos en que cambia la salida del *flip-flop* es necesario disminuirla. Así pues, seleccionando un tiempo de 2 segundos para entre cada flanco de reloj, se disminuye la frecuencia del reloj de 50 *MHz* a 0.5 *Hz* tiempo suficiente para notar los cambios que se producen a la salida del *flip-flop*.

Entadas					Salidas		Operación
$\overline{PRE}$	$\overline{CLR}$	Reloj (clk)	J	K	Q	$\overline{Q}$	
0	0	X	X	X	-	-	Ambiguo (no se usa)
0	1	X	X	X	1	0	Prestablecer
1	0	X	X	X	0	1	Restablecer
1	1	X	X	X	Qo	$\overline{Qo}$	Desactivar entradas asíncronas
1	1	↑	0	0	Qo	$\overline{Qo}$	Mantener
1	1	↑	1	0	1	0	Set (establecer)
1	1	↑	0	1	0	1	Reset (reiniciar)
1	1	↑	1	1	$\overline{Qo}$	Qo	Cambiar

Tabla 8.8: Tabla de comportamiento del *flip-flop* tipo JK, con entradas asíncronas

Posteriormente se crea un circuito que contenga, el *flip-flop* a implementar así como un módulo reductor de frecuencia. Un diagrama RTL se presenta en la figura 8.7,donde se ilustran las conexiones que se requieren para su funcionamiento.

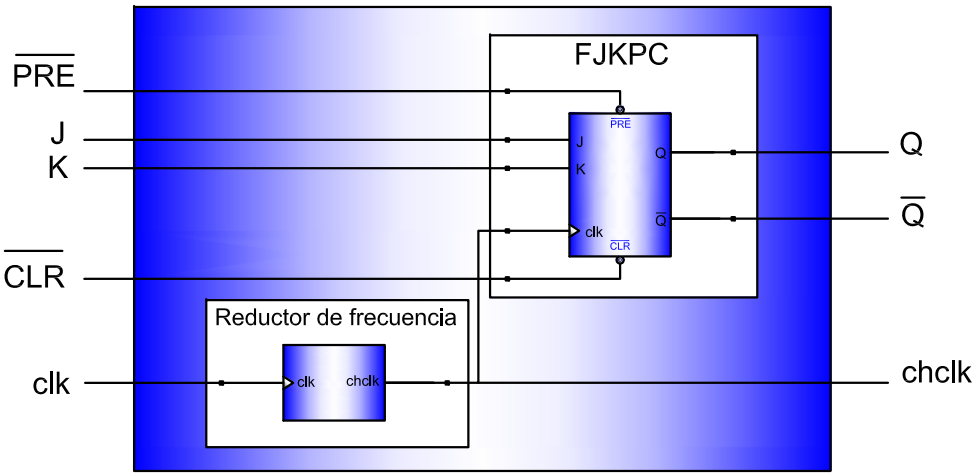


Figura 8.7: Diagrama RTL del circuito que contiene los dos componentes.

FUENTE:Los autores

De acuerdo a lo planteado anteriormente, adicionalmente a las descripciones de los *flip-flops* se presentan dos módulos: el reductor de frecuencia y un circuito en el que se hacen las instancias de uno del reductor de frecuencia y el tipo de *flip-flop*. haciendo referencia a la figura 8.7.

8.4.3.1 Descripción en *VHDL* del *flip-flop* tipo SR

Como todas las descripciones en *VHDL* primero declaramos la librería, a continuación declaramos los puertos de entrada y los de salida, finalmente se realiza la descripción de *hardware* ó descripción del comportamiento.

En todos los *flip-flops* se usa la sintaxis *process* para el lenguaje de *VHDL*. A continuación se presenta la descripción en *VHDL* del *flip-flop* SR en el *script* 8.4

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4
5  --Esta parte del código es la que define las entradas y las salidas del circuito lógico a
6  --implementar, asi pues se toma R,S como entradas asíncronas, clk esta representado como una
7  --entrada síncrona y q, nq como las salidas.
8
9  Entity fftSR is
10         Port ( R, S, clk        : in  std_logic;
11                q, nq            : out  std_logic);
12 end fftSR;
13
14 --Esta última parte del código es la que define el comportamiento, conexión o flujo de datos
15 --del circuito lógico.
16
17 architecture Comportamiento of fftSR is
18 signal signal_q : std_logic:= '0';
19 signal signal_nq: std_logic:= '1';
20 begin
21
22  process(clk,R,S,signal_q,signal_nq) -- inicio del process
23  begin
24         if (clk'event and clk='1') then
25          if (S='0' and R='1') then
26                 signal_q <='0';
27                 signal_nq <='1';
28           elsif (S='1' and R='0') then
29                 signal_q <='1';
30                 signal_nq <='0';
31           elsif (R='0' and S='0') then
32                 signal_q <=signal_q;
33                 signal_nq <=signal_nq;
34           else
35                 signal_q <='1';
36                 signal_nq <='1';
37          end if;
38         end if;
39 q <= signal_q;
40 nq <= signal_nq;
41  end process; -- finalización del process
42 end Comportamiento;
```

Script 8.4: Descripción en *VHDL* del *flip-flop* tipo SR.

8.4.3.2 Descripción en *VHDL* del *flip-flop* tipo D

A continuación se presenta la descripción en *VHDL* del *flip-flop* tipo D en el *script* 8.5

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4
5  --Esta parte del código es la que define las entradas y las salidas del circuito lógico a
6  --implementar, asi pues se toma  a D yclk como entradas sincornas. set y reset son entradas
7  --asíncronas, q y nq como sus salidas.
8
9  Entity fftD is
10         Port ( D, clk    : in  std_logic;
11                nPRE, nCLR : in std_logic;
12                q, nq    : out  std_logic);
13 end fftD;
14
15 --Esta última parte del código es la que define el comportamiento, conexión o flujo de datos
16 --del circuito lógico.
17
18 architecture Comportamiento of fftD is
19 begin
20  process(clk,D,nPRE,nCLR)
21  begin
22         if nCLR='0' then
23                q<='0';
24                nq<='1';
25          elsif nPRE='0' then
26                q<='1';
27                nq<='0';
28           elsif rising_edge(clk) then
29         q<=D;
30         nq<=not(D);
31         end if;
32  end process;
33 end Comportamiento;
```

Script 8.5: *flip-flop* tipo D.

> Nota : en la descripción del circuito presentado se cambio la sentencia que detecta el cambio del reloj por *rising_edge (clk)*.

8.4.3.3 Descripción en *VHDL* del *flip-flop* T

A continuación se presenta la descripción en *VHDL* del *flip-flop* tipo T en el *script* 8.6

```
1
2  Library ieee;
3  Use ieee.std_logic_1164.all;
4
5  Entity fftT is
6    Port ( clk, T: in std_logic;
7         nPRE,nCLR : in std_logic;
```

```
8         q, nq: out std_logic);
9   end fftT;
10
11 Architecture comportamiento of fftT is
12 signal estado :std_logic:= '0';
13 Begin
14   Process (clk, nCLR, nPRE, estado)
15   Begin
16         if nCLR='0'
17         then
18                 estado <='0';
19         elsif nPRE='0'
20         then
21                 estado <='1';
22         elsif clk'event and clk='0' then
23       estado <= T XOR estado;
24     end if;
25     q <= estado;
26     nq <= not(estado);
27   End process;
28 End comportamiento;
```

Script 8.6: *flip-flop* tipo T.

Nota : se presenta una manera de describir un *flip-flop* con flanco negativo.

8.4.3.4 Descripción en *VHDL* del flip-flop JK

A continuación se presenta la descripción en *VHDL* del *flip-flop* tipo JK en el *script* 8.7

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4
5  --Esta última parte del código es la que define el comportamiento, conexión o flujo de datos
6  --del circuito lógico.
7
8  architecture Comportamiento of fftJKrs is
9  signal signal_q: std_logic:='0';
10 begin
11  process(clk, nPRE, nCLR) -- inicio del process
12  begin
13         if nCLR='0' then
14             signal_q <='0'; -- el Restablcer es negado como se presenta en la tabla
15         elsif nPRE='0' then
16             signal_q <='1'; -- el Prestablecer en negado como es presentado en la tabla
17         elsif clk'event and clk='1' then -- esta linea es la encargada de detectar el flanco de
18                                          -- reloj.
19             if (J='0' and K='1') then
20                 signal_q <='0';
21           elsif (J='1' and K='0') then
```

```
22                 signal_q <='1';
23           elsif (J='0' and K='0') then
24                 signal_q <=signal_q;
25           else
26                 signal_q <=not(signal_q);
27          end if;
28          end if;
29  end process; -- finalización del process
30
31         q <= signal_q; --esta parte define las saldas del flip-flop descrito
32         nq <= not(signal_q);
33
34  end Comportamiento;
```

Script 8.7: *flip-flop* tipo JK.

> Nota : en el circuito descrito anteriormente se observa que la entada nCLR, es descrita primero en el código, por lo tanto ésta tiene mayor prioridad que las entradas descritas después, por ejemplo nPRE.

En el *script* 8.8 se presenta la descripción en *VHDL* del circuito reductor de frecuencia.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  ------------------------------------------------------------------------------
5  entity clock is
6  port (
7          clk      : in std_logic;
8          clkch    : out std_logic);
9  end clock;
10 ------------------------------------------------------------------------------
11 architecture comportamiento of clock is
12 signal contador : std_logic_vector (26 downto 0) :="000000000000000000000000000";
13 signal clk1: std_logic := '0';
14 begin
15  process (clk , clk1, contador )
16  begin
17         if clk'event and clk = '1' then
18                        if contador = "10111110101111000001111111" then
19                                contador <= ( others => '0');
20                                clk1 <= not(clk1);
21                                else
22                                contador <= std_logic_vector ( unsigned ( contador )+1);
23                        end if;
24         end if;
25  end process;
26 clkch <= clk1;
27 ------------------------------------------------------------------------------
28 end comportamiento;
```

Script 8.8: Descripción en VHDL del circuito reductor de frecuencia.

del circuito en el cual se realizan las instancias (*script* 8.9).

En el *script* 8.9 se presenta la descripción del circuito que contiene el módulo reductor de frecuencia y el *flip-flop* a implementar (JK).

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity Imfftjkrs is
7          Port ( J, K, clk, nPRE, nCLR     : in  std_logic;
8                 q, nq, clk1               : out  std_logic);
9  end Imfftjkrs;
10
11 architecture Behavioral of Imfftjkrs is
12 ----------------------------COMPONENTES-----------------------------
13 component clock
14
15 port (
16         clk      : in std_logic;
17         clkch    : out std_logic);
18 end component;
19
20 component fftjkrs
21         Port ( J, K, clk          : in  std_logic;
22                nPRE, nCLR         : in std_logic;
23                q, nq              : out  std_logic);
24 end component;
25
26 ----------------------------SEÑALES A USAR-------------------------
27 signal signal_clkch:std_logic;
28 -------------------------------------------------------------------
29 begin
30
31 inst_clock:clock
32
33         port map(        clk=>clk,
34                          clkch=>signal_clkch);
35
36
37 inst_fftSR:fftjkrs
38
39         port map(        J=> not J,
40                          K=> not K,
41                          nPRE=> nPRE,
42                          nCLR=> nCLR,
43                          clk=>signal_clkch,
44                          q=>q,
45                          nq=>nq);
```

```
46  clk1 <= signal_clkch;
47  end Behavioral;
```

Script 8.9: Descripción del circuito de un *flip-flop* y el reductor de frecuencia.

8.4.4 Paso 3: Simulación de la descripción de los diferentes tipos de *flip-flops*

A continuación se muestran las gráficas de las simulaciones realizadas a los *flip-flops* descritos anteriormente.

En la figura 8.8 se presenta el marcador a los 100 *ns*, tiempo en el cual se puede almacenar los datos.

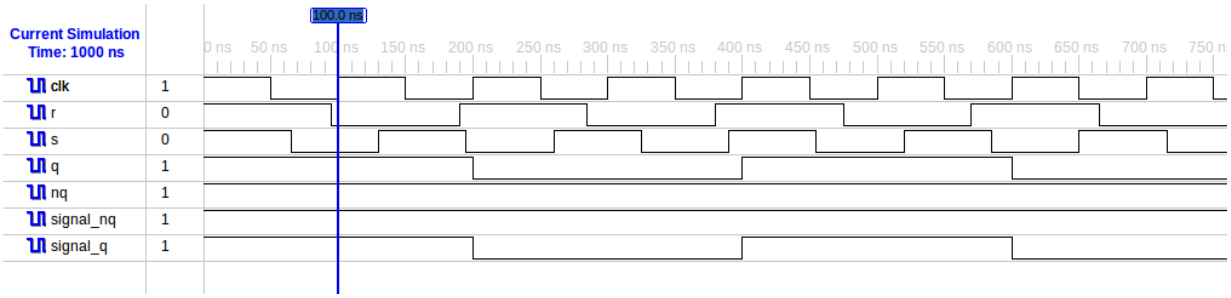


Figura 8.8: Simulación de un *flip-flop* tipo SR.

FUENTE:Herramienta de simulación ISE

En la figura 8.9 se presenta el marcador a los 50 *ns*, recuerde que este *flip-flop* se activa con flanco negativo.

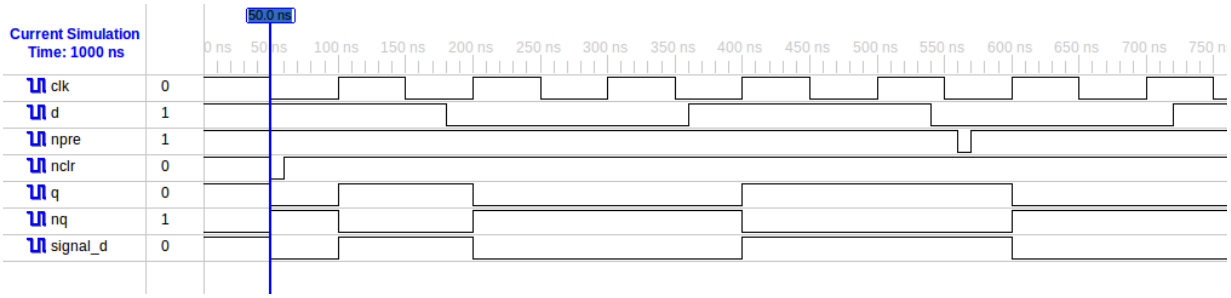


Figura 8.9: Simulación de un *flip-flop* tipo D.

FUENTE:Herramienta de simulación ISE

En la figura 8.10 se presenta el marcador a los 20 *ns*, tiempo en el cual se activa la señal **npre**.

En la figura 8.11 se presenta el marcador a los 10 *ns*, tiempo en el cual se activa la señal **nclr**.

8.4.4.1 Paso 4: Identificación de dispositivos a utilizar

Los Elementos necesarios para la implementación del *flip-flop* JK son:

- La tarjeta *SIE*.

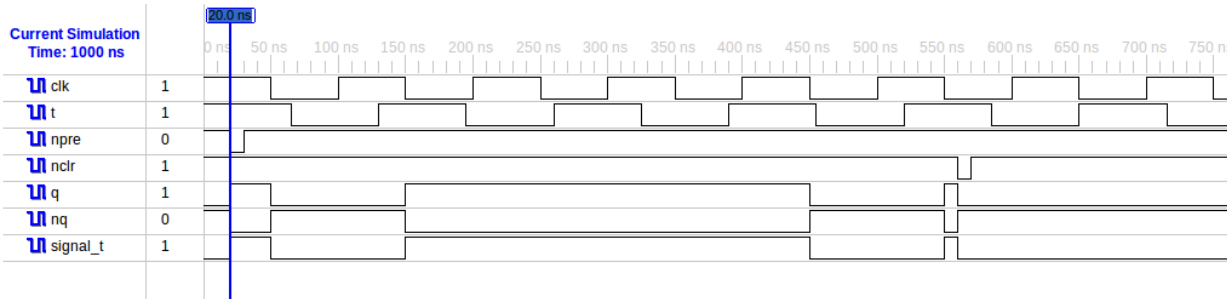


Figura 8.10: Simulación de un *flip-flop* tipo T.

FUENTE:Herramienta de simulación ISE

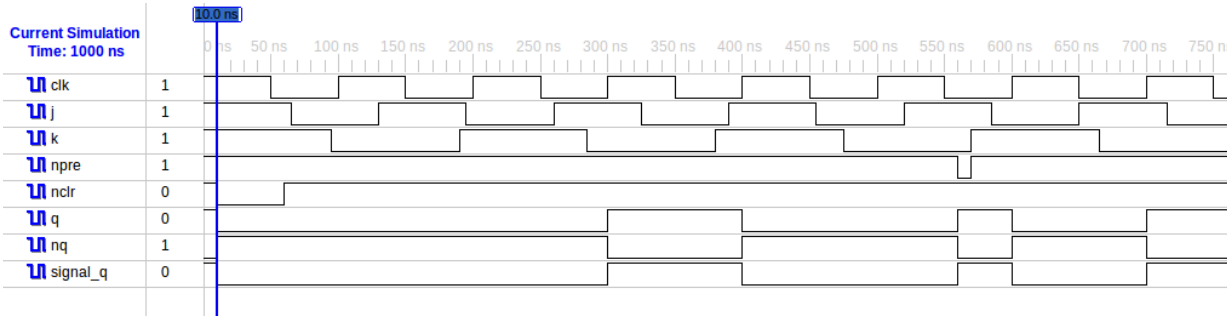


Figura 8.11: Simulación de un *flip-flop* tipo JK.

FUENTE:Herramienta de simulación ISE

- La tarjeta de expansión de pines.

8.4.4.2 Paso 5: Asignación de pines

Para realizar la prueba de laboratorio se entregan circuitos reductores de frecuencia (contador de 2 segundos), el cual funciona con el reloj de 50MHz que tiene la plataforma *SIE*, para ello es necesario crear un circuitos de contenga al reloj como al *flip-flop* aprobar.

El *script* 8.10 nos presenta un ejemplo de la asignación de pines realizada al circuito que contiene a un *flip-flop* tipo JK y al reductor de frecuencia con *reset*.

```
1 net clk loc= "P38"; #Entrada del reloj de la FPGA 50MHz
2 net clk1 loc= "P48"; #led marca de 2 segundos
3 net J loc= "P30"; #pulsador de la SIE
4 net K loc= "P13"; #pulsador de la SIE
5 net nCLR loc= "P57"; #pulsador de la SIE
6 net nPRE loc= "P53"; #pulsador de la SIE
7 net q loc= "P34"; #led salida del flip-flop
```

```
8  net nq loc= "P36"; #led salida negada del flip-flop
```

Script 8.10: Archivo *ucf* del circuito que tiene las instancias

Al circuito que contiene a los otros, se le debe modificar las entradas para el caso que se desee implementar (El circuito esta descrito para funcionar con el *flip-flop* JK).

Obsérvese, la lógica de la tarjeta fue cambiada para que cuando presione un pulsador la salida sea un nivel alto (1). la señal **clk1** nos presenta el reloj de entrada al *flip-flop*, se busca que este lo vea el usuario final, para la comprobar el efecto que causa éste en el *flip-flop* implementado.

8.4.4.3 Paso 6: Utilización del *Makefile*

Para llevar a cabo el procedimiento necesario en el presente paso, es preciso referirse a la guía de laboratorio 1, en donde se expone de forma detallada lo referente al *Makefile*, el cual es un archivo proporcionado por la persona encargada de guiar la clase, y al cual se le debe editar haciendo una pequeña modificación en la segunda linea, donde debe agregarse el nombre del fichero que se desea implementar.

Nota : No olvidar agregar las fuentes en el *Makefile* necesarias para realizar las instancias.

8.4.4.4 Paso 7: Implementación en la tarjeta *SIE*

Ya teniendo los archivos necesarios para la respectiva implementación, es preciso llevar a cabo el proceso necesario para una óptima implementación, para lo cual es necesario seguir los incisos mostrados a continuación:

- Conectar la tarjeta *SIE*
- Configuración del puerto *USB*
- Realizar la síntesis de la descripción
- Enviar información al procesador
- Acceder al procesador de la tarjeta
- Enviar información al *FPGA*

Es preciso mencionar que si se tiene alguna duda para llevar a cabo el proceso mencionado anteriormente, es necesario referirse a la guía de laboratorio 1, en donde se explica detalladamente cada uno de los anteriores.

8.5 RESUMEN

Lo que se busca con esta práctica es mostrar formas de implementar los *flip-flops* en una *FPGA*, utilizando un lenguaje de descripción de *hardware* (*VHDL*). En primer lugar se presentan las características y la representación de cada uno de los *flip-flops*, siguiendo con el procedimiento se plantea una forma de describir los circuitos con *VHDL*, para proceder a la simulación. Finalmente se siguen los últimos pasos para la implementación. En la implementación se necesita crear un circuito que reduzca la frecuencia del reloj que tiene la plataforma SIE. Se entrega el circuito de reloj y se presenta el circuito principal, que contiene tanto al módulo reductor de frecuencia, como al *flip-flop* a implementar.

8.6 PREGUNTAS DE PRUEBA

- Si se tiene en hardware un flip-flop tipo JK, ¿qué debo adicionar a éste para convertirlo en un flip-flop tipo D?

  Nota: compare las tablas de característica 8.2 y 8.4.

- Seleccionar un flip-flop previamente presentado y suponer que la salida y la señal de sincronía se encuentran representadas en la figura 8.12.

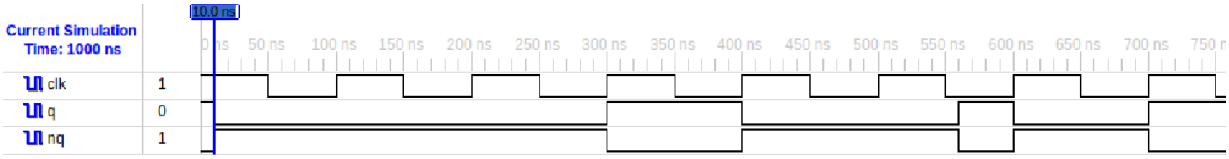


Figura 8.12: Salida y señal de sincronía de un *flip-flop*.

FUENTE:Herramienta de simulación ISE

Realizar un diagrama de tiempo que presente las entradas que pudo tener dicho *flip-flop*. (tiempos en los cuales se activan o desactivan las señales de entrada).

8.7 EJERCICIO PROPUESTO

Implementar en la tarjeta SIE un flip flop tipo D y uno tipo T.

LABORATORIO **9**

Registros y Contadores.

Leer demasiados libros es peligroso.

MAO TSE TUNG

9.1 INTRODUCCIÓN

Por mucho, el uso más común de los *flip-flops* es para el almacenamiento de datos o información. Los datos pueden representar valores numéricos (por ejemplo, números binarios, decimales codificados en BCD). Estos datos generalmente se almacenan en grupos de *flip-flops* llamados registros. La operación que se realiza con más frecuencia sobre los datos almacenados en un *flip-flop* o registro es la **transferencia**. Esta operación comprende la transferencia de datos de un *flip-flop* o registro a otro[5]. Los registros más usados son los de entrada y salida en paralelo, que almacenan un dato de n bits. No obstante en sistemas de comunicaciones son muy útiles los denominados registros de desplazamiento, los cuales permiten enviar una palabra de n bits por un solo cable, enviando los bits en serie de uno en uno.

Por otra parte los circuitos lógicos secuenciales se dividen básicamente en dos grupos: Los circuitos asíncronos y los circuitos síncronos. Los primeros pueden cambiar los estados de sus salidas como resultado del cambio de los estados de las entradas, mientras que los circuitos sincrónicos pueden cambiar el estado de sus salidas en instantes de tiempo discretos bajo el control de una señal de reloj.

9.2 OBJETIVOS

- Realizar la descripción en VHDL de un registro.
- Realizar la descripción en VHDL de un contador.
- Implementar los circuitos descritos en VHDL.

9.3 MARCO TEÓRICO

9.3.1 Registro con entrada serie/salida serie.

La figura 9.1 muestra dos registros de corrimientos de 3 bits conectados de modo que el contenido del registro **X** será transferido en serie al registro **Y**. Utilizando *flip-flops* tipo D por cada registro de corrimiento, ya que esto requiere de menos conexiones que los flip-flops tipo J-K. Note la forma en que X_0 (último *flip-flop* del registro **X**), está conectado a la entrada de Y_2 (primer *flip-flop* del registro **Y**). De esta manera, cuando se aplican pulsos de corrimiento, la transferencia de la información se lleva a cabo como sigue: $X_2 \rightarrow X_1 \rightarrow X_0 \rightarrow Y_2 \rightarrow Y_1 \rightarrow Y_0$. El *flip-flop* X_2 pasará a estados determinados por su entrada D. [5]

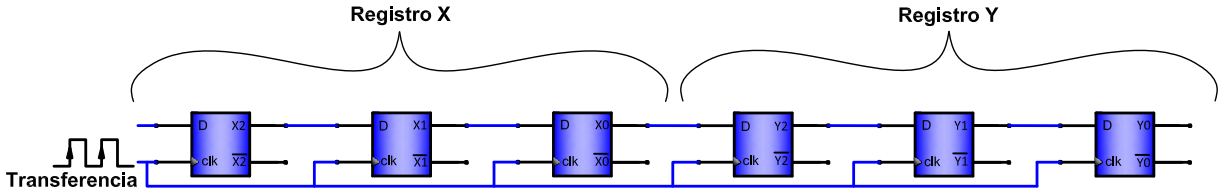


Figura 9.1: Diagrama RTL de un registro entrada serie/salida serie.

FUENTE:Los autores

9.3.2 Registro con entrada serie/salida paralelo

En este tipo de registro los bits de datos se introducen en serie (empezando por el bit situado a la derecha), del mismo modo que se ha visto en la sección 9.3.1. La diferencia está en la forma en que dichos bits se extraen del registro; en un registro con salida paralelo, se dispone de la salida de cada etapa. Una vez que los datos se han almacenado, cada bit se presenta en su respectiva línea de salida, estando disponibles todos los bits simultáneamente, en lugar de bit a bit como en el caso de la salida serie. [4] La figura 9.2, muestra un registro de desplazamiento con entrada serie/salida paralelo.

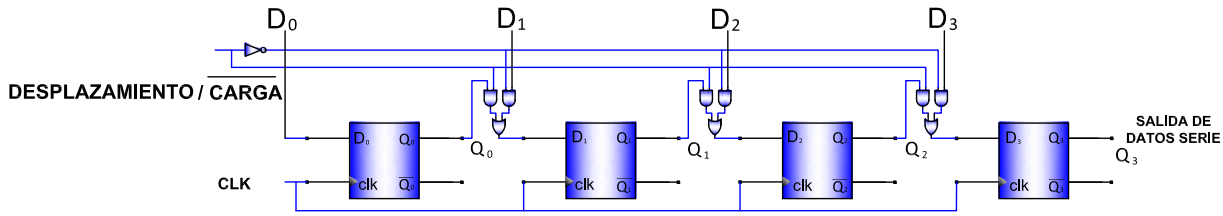


Figura 9.2: Diagrama RTL de un registro de entrada serie/salida paralelo.

FUENTE:Los autores

9.3.3 Registro con entrada paralelo/salida serie

En un registro con entradas de datos paralelo, los bits se introducen simultáneamente en sus respectivas etapas a través de líneas paralelo, en lugar de bit a bit a través de una única línea como ocurre con las entradas de datos serie. La salida serie se hace del mismo modo que se ha descrito en la sección 9.3.1, una vez que todos los datos están almacenados en el registro. [4]

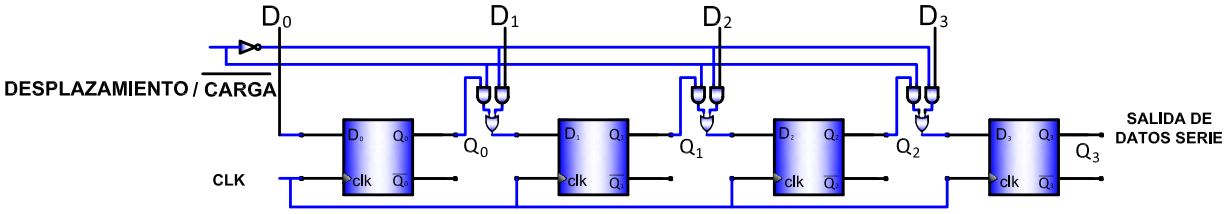


Figura 9.3: Diagrama RTL de un registro de entrada paralelo/salida serie.

FUENTE:Los autores

9.3.4 Registro con entrada paralelo/salida paralelo

La figura 9.4 muestra la transferencia de datos de un registro a otro mediante el uso de *flip-flops* tipo D. El registro **X** consta de los flip-flops X_1, X_2, X_3; el registro **Y** costa de los *flip-flops* Y_1, Y_2, Y_3. Cuando se aplica un pulso de transferencia, el nivel almacenado X_1 será transferido a Y_1, X_2 a Y_2 y X_3 a Y_3. La transferencia del contenido del registro **X** al registro **Y** es una transferencia síncrona. También se le conoce como transferencia paralela.

9.3.5 Contadores asíncronos

Un contador binario asíncrono consiste en una conexión en serie de flip-flops complementarios (tipo T ó JK), con la salida de cada flip-flop conectado a la entrada de sincronía del siguiente flip-flop de mayor orden. El flip-flop que almacena el bit menos significativo recibe los pulsos de reloj (clk). El diagrama de un contador asíncrono de 4 bits se muestra en la figura 9.5. Todas las entradas J y K son iguales a un 1 (nivel alto)[6].

9.3.6 Contadores síncronos

Los contadores síncronos se distinguen de los contadores asíncronos en que los pulsos de reloj se aplican a las entradas o terminales de sincronía de todos los flip-flops. El pulso común dispara todos los flip-flops simultáneamente en vez de uno a la vez en cadencia como en un contador asíncrono. La decisión de cuándo se debe o no complementar un flip-flop se determina de los valores de las entradas J y K en el momento del pulso. Si J=K=0, el flip-flop permanece sin cambio. Si J=K=1 el flip-flop se complementa[6]. La figura 9.6 presenta un circuito contador síncrono módulo 10.

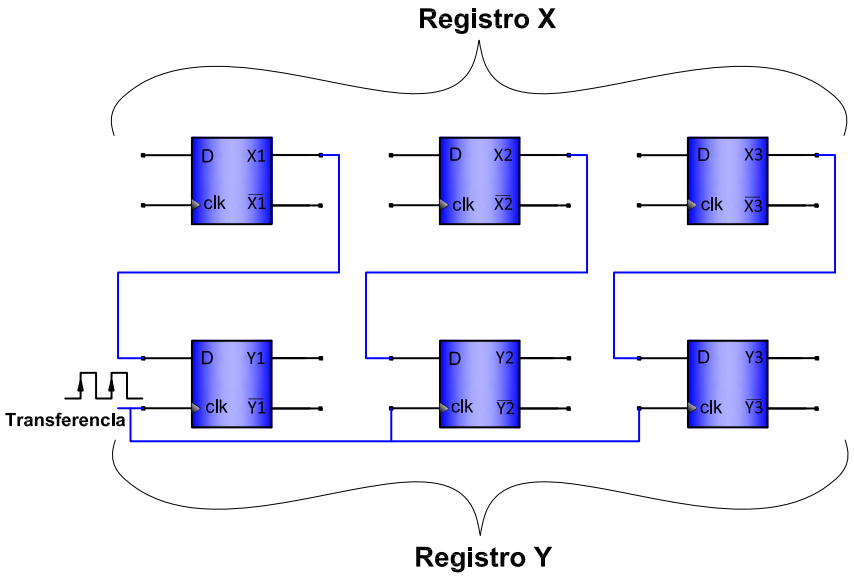


Figura 9.4: Diagrama RTL de un registro con entrada paralelo/salida paralelo.

FUENTE:Los autores

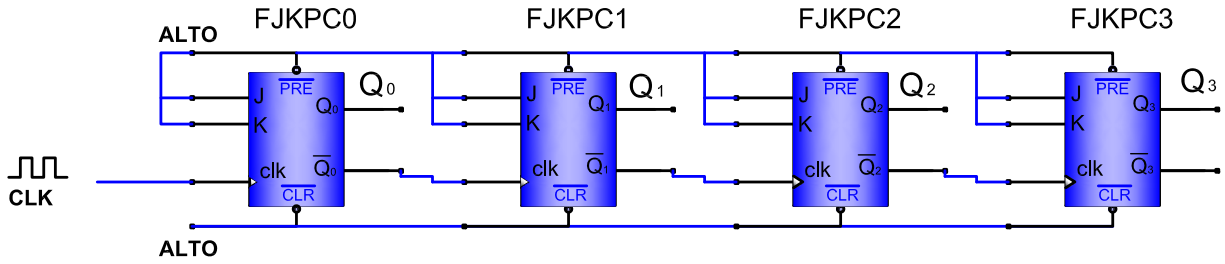


Figura 9.5: Diagrama RTL de un contador asíncrono.

FUENTE:Los autores

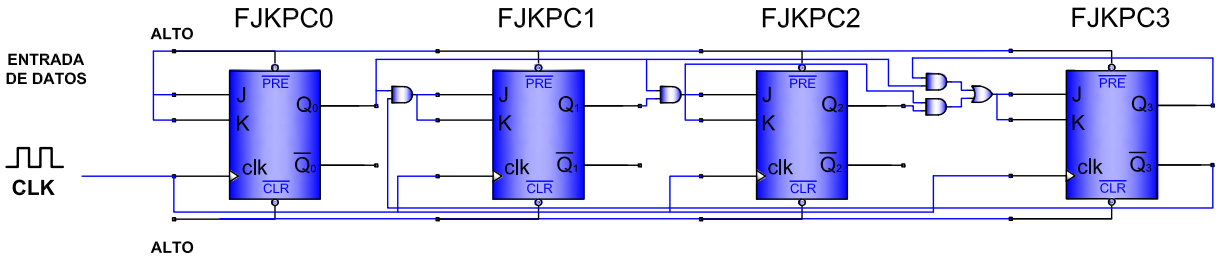


Figura 9.6: Diagrama RTL de un contador síncrono módulo 10.

FUENTE:Los autores

9.4 PROCEDIMIENTO GENERAL

Esta práctica consta de la realización de dos circuitos: En el primero se desea guardar la información de una entrada en un registro, la información de entrada se desea observar en los *LEDs* y con un juego de *switches* se introducen los datos a la plataforma. En el segundo circuito se desea implementar un contador módulo 10 y su salida se puede observar en un *display* de 7 segmentos, como entrada tiene un pulsador para reiniciar la cuenta no importando el estado en que se encuentre.

Para llevar a cabo la realización de la guía se propone la siguientes recomendaciones:

- Identificación de entradas y salidas
- Plantear los circuitos a utilizar
- Descripción de los módulos internos a utilizar
- Observación de los RTLs de los circuitos descritos
- Simulación
- Elementos necesarios para la implementación
- Asignación de pines creando el archivo .ucf
- Realizar la descripción estructural
- Implementación

9.4.1 PROCEDIMIENTO PASO A PASO

9.4.1.1 Paso 1: Identificación de entradas y salidas

Para el primer circuito a implementar tenemos las entradas, las cuales son: una entrada de 4 bits la cual es la información que se desea guardar en el registro, una entrada $overline{CLR}$ la cual cambia la salida de forma asíncrona a un estado "0000", una entrada que indica cuando los datos van a ser almacenados o transmitidos (c_d) y la entrada de reloj **clk**; por otra parte se tienen las salidas las cuales son: la salida de un bit del registro, la representación de la entrada de 4 bits y una salida de un bit que cambia cuando la señal de sincronía realiza una transición (chclk).Lo dicho anteriormente se representa en el esquema de la figura 9.7.

El segundo circuito es un contador síncrono módulo 10, este consta de dos entradas de un bit, las cuales son clk y $overline{CLR}$; el circuito tiene una salida de 4 bits representando el conteo y una de un bit representando la transición del reloj (clk). Un esquema del módulo contador se ilustra en la figura 9.8

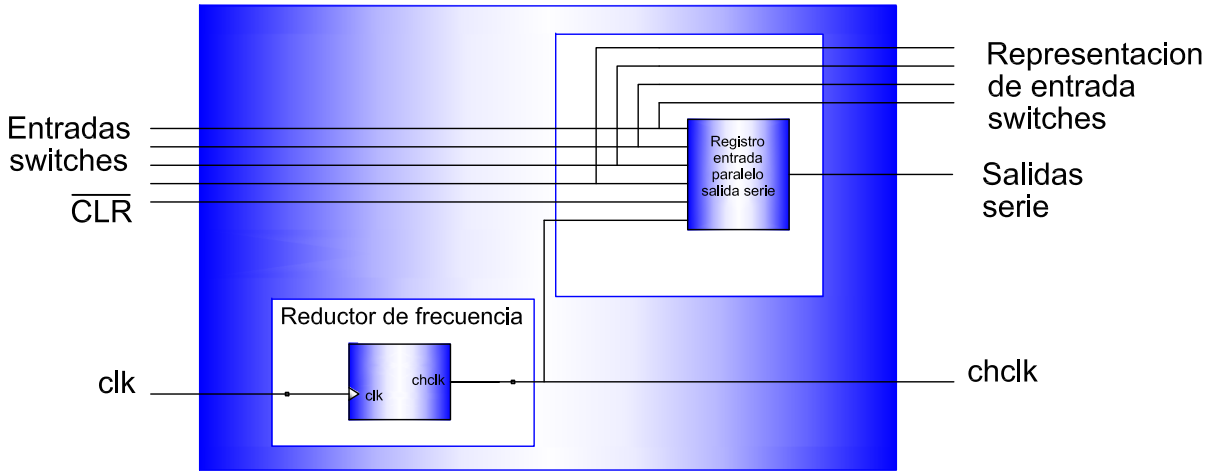


Figura 9.7: Diagrama de bloques del registro entrada paralelo/salida serie.

FUENTE:Los autores

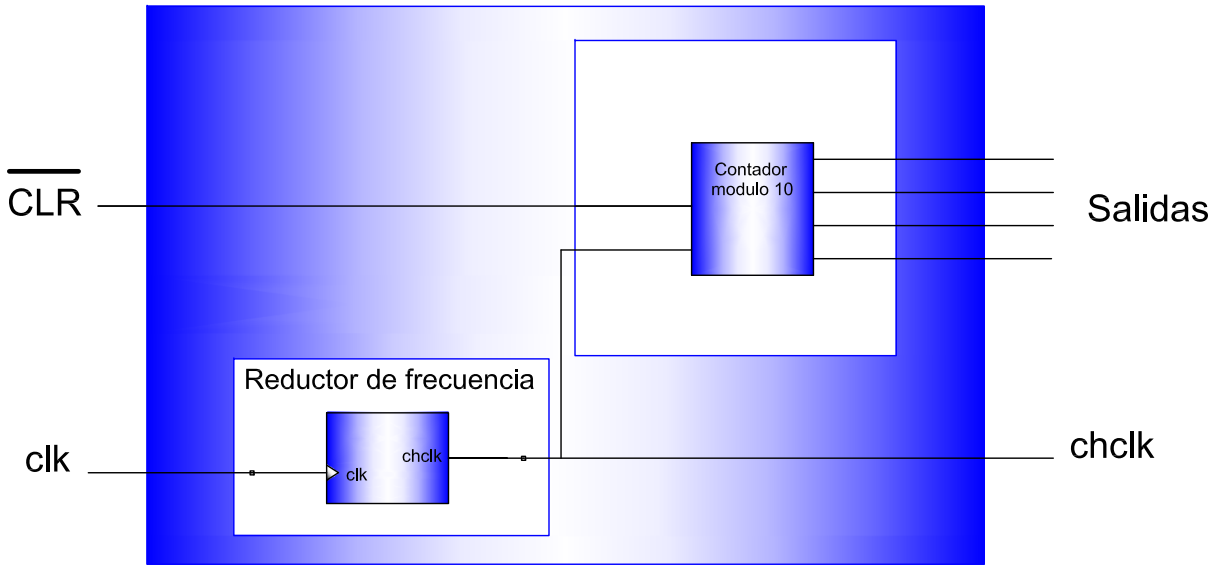


Figura 9.8: Diagrama de bloques del contador asíncrono módulo 10.

FUENTE:Los autores

9.4.1.2 Paso 2: Plantear los circuitos a utilizar

Para realizar el primer circuito vamos a utilizar el registro paralelo-serie, este permite tomar los datos de la entrada en el mismo instante de tiempo y enviarlos de forma serie. Es necesario tener una señal que permita saber cuándo hay que cargar los datos y en que momento los datos se deben transmitir, en la figura 9.9 vemos que la señal se llama desplazamiento/$\overline{carga}$.

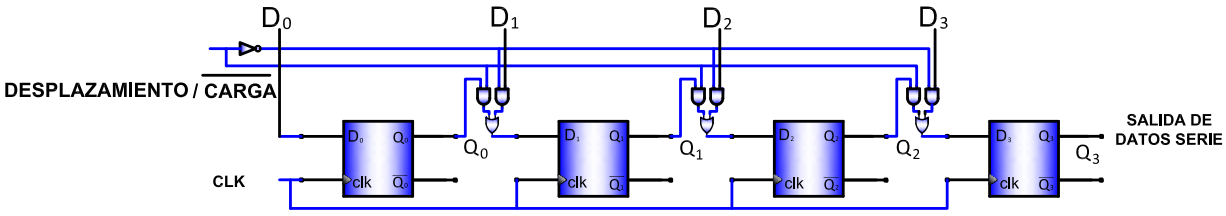


Figura 9.9: Registro de entrada paralelo/salida serie.

FUENTE:Los autores

Se plantea utilizar un circuito reductor de frecuencia, ya que el reloj de la plataforma tiene una frecuencia de 50 *MHz*, por ello se crea un circuito que modifica la frecuencia de 50 *MHz* a una frecuencia de 0.5 *Hz*, es decir un periodo de 2 segundos.

En el segundo circuito tenemos un contador síncrono módulo m, para comprender el funcionamiento de un contador módulo m, se parte de utilizar *flip-flops* tipo T o JK, ya que estos ofrecen dos operaciones (cambiar y mantener), estas operaciones son las que permiten realizar el conteo. Lo primero que se debe determinar es el número de estados necesario para implementar el contador, para el ejemplo se utiliza un contador módulo 10.
Como son necesarios 10 estados basta con 4 *flip-flops* ya que $2^4 = 16$. El segundo paso es realizar un diagrama de tiempos como lo muestra la figura 9.10. Para entender el funcionamiento de este tipo de contador debe examinarse detenidamente su secuencia de estados, la cual se muestra en la tabla 9.1.

Figura 9.10: Diagrama de tiempo del contador síncrono módulo 10.

FUENTE:Herramienta de simulación ISE

analizando la tabla 9.1 se observa que Q_0 cambia en cada impulso de reloj a medida que el contador avanza desde su estado original hasta su estado final, para luego iniciar un nuevo ciclo a partir de su estado original. Para conseguir este funcionamiento, el *flip-flop* FJKPC0 mostrado en la figura 9.11 debe que realizar la operación cambiar, aplicando constantemente niveles altos en sus entradas J_0 y K_0. Téngase en cuenta que Q_1 pasa al estado contrario cada vez que Q_0 está en 1 y Q_3 está en 0. Este cambio se produce en el pulso de reloj 2, 4, 6, 8. El pulso

Pulso de reloj	Q_3	Q_2	Q_1	Q_0
Inicialización	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (nuevo ciclo)	0	0	0	0

Tabla 9.1: Estados del contador de décadas BCD.

de reloj 10 hace que el contador inicie un nuevo ciclo. Para conseguir este modo de operación, se conecta Q_0 and $\overline{Q_3}$ a las entradas de J_1 y K_1 de FJKPC1. cuando Q_0 y $\overline{Q_3}$ están a 1 y se produce un impulso de reloj, FJKPC1 se encuentra realizando la operación cambiar y, por tanto, cambia de estado. El resto de las veces el *flip-flop* FJKPC1 está en la modo mantener, reteniendo su estado actual. A continuación, se presenta la forma de conseguir que el *flip-flop* FJKPC2 cambie de estado en los instantes adecuados de acuerdo a la secuencia binaria. Obsérvese que las dos veces que Q_2 cambia de estado, debe cumplirse la única condición de que tanto Q_0 como Q_1 estén a nivel alto. Esta condición se detecta mediante la compuerta *and*, cuya salida se aplica a las entradas J_2 y K_2 del *flip-flop* FJKPC2 logrando un nivel alto en sus entradas, y de éste modo hacer que su salida cambie en el siguiente ciclo de reloj. El resto de las veces, las entradas J_2 y K_2 de FJKPC2 se mantienen a nivel bajo, al igual que la puerta *and*, y FJKPC2 no cambia de estado. Finalmente, el *flip-flop* FJKPC3 (Q_3) cambia de estado en el siguiente impulso de reloj cada vez que Q_0=1, Q_1=1, Q_2=1 (estado 7), o cuando Q_0=1 y Q_3=1 (estado 9), y se puede deducir del mapa de *karnaugh* mostrado en la figura 9.11.

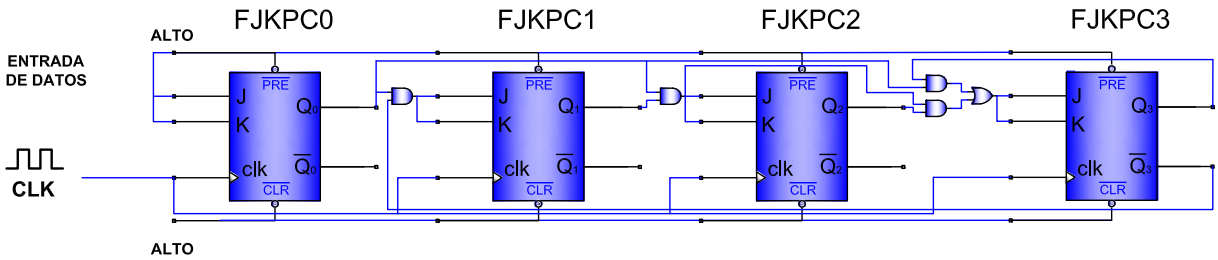


Figura 9.11: Diagrama RTL de un contador síncrono módulo 10.

FUENTE:Los autores

La ecuación obtenida del mapa de *karnaugh* que rige el comportamiento de FJKPC3 es la siguiente:

$$J_3 = K_3 = Q_0 * Q_1 * Q_2 + Q_0 * Q_3 \tag{9.1}$$

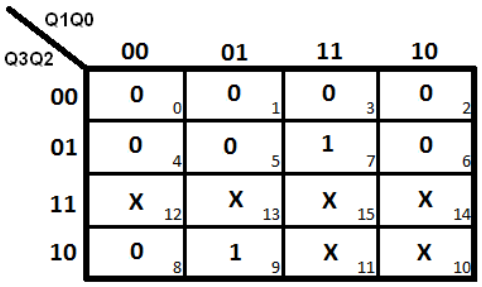


Figura 9.12: Mapa de karnaugh para el *flip-flop* FJKPC3.

FUENTE:Los autores

9.4.1.3 Paso 3 Descripción en VHDL de los módulos internos a utilizar

Para realizar la práctica es necesario describir en VHDL los circuitos mencionados anteriormente. El *script* 9.1 presenta la descripción en VHDL de un registro de entrada paralelo y salida serie.

```
1  library ieee;
2  use ieee. std_logic_1164 .all;
3  ------------------------------------------------------------------------------
4  entity RegParSer is
5          port (
6                  clk : in std_logic;
7                  reset : in std_logic;
8                  c_d : in std_logic; -- Carga (0) o desplaza (1)
9                  e_p : in std_logic_vector (3 downto 0); -- Entrada
10                 s_s : out std_logic ); -- Salida serie
11         end RegParSer;
12 ------------------------------------------------------------------------------
13 architecture comportamiento of RegParSer is
14 signal registro : std_logic_vector (3 downto 0):="0000";
15 begin
16
17  Reg: process (clk, reset, c_d, e_p )
18  begin
19         if reset = '1' then
20                 registro <= (others => '0');
21                  elsif clk 'event and clk = '1' then
22                         if c_d = '0' then -- Carga
23                                 registro <= e_p;
24                         else -- Desplaza
25                                 registro (3) <= '0'; -- Se introducen ceros.
26                                 registro (2 downto 0) <= registro (3 downto 1);
27                         end if;
28                 end if;
29 end process Reg;
30
31 -- Se copia el bit menos significativo a la salida serie.
32 s_s <= registro (0);
33 ------------------------------------------------------------------------------
```

```
34 end comportamiento;
```

Script 9.1: Descripción en VHDL de un registro entrada paralelo-salida serie.

En el *script* 9.2 se presenta la descripción contador síncrono módulo 10.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  ----------------------------------INSTANCIAS-----------------------------------------------
4  Entity fftJKrs is
5          Port (  J, K, clk       : in  std_logic;
6                  nPRE, nCLR      : in std_logic;
7                  q, nq           : out  std_logic);
8  end fftJKrs;
9  ---------------------------------------------------------------------------------------------
10
11 architecture Comportamiento of fftJKrs is
12 signal signal_q: std_logic:='0';
13 begin
14  process(clk, nPRE, nCLR) -- inicio del process
15  begin
16         if nCLR='0' then
17                 signal_q <='0'; -- el reset es negado como se presenta en la tabla
18          elsif nPRE='0' then
19                 signal_q <='1'; -- el set en negado como es presentado en la tabla
20           elsif clk'event and clk='1' then -- esta linea es la encargada de detectar el
21                                                  --flanco de reloj en este caso subiendo
22                         if(J='0' and K='1') then
23                                 signal_q <='0';
24                         elsif (J='1' and K='0') then
25                                 signal_q <='1';
26                         elsif (J='0' and K='0') then
27                                 signal_q <= signal_q;
28                         else
29                                 signal_q <=not(signal_q);
30                         end if;
31          end if;
32  end process; -- finalización del process
33         q <= signal_q; --esta parte define las saldas del flip-flop descrito
34         nq <= not(signal_q);
35 end Comportamiento;
36 ---------------------------------------------------------------------------------------------
37 ----------APARTIR DE ESTE PUNTO SE INICIA LA DESCRIPCION DEL CIRCUITO PRINCIPAL-------------
38 library ieee;
39 use ieee.std_logic_1164.all;
40 ---------------------------------------------------------------------------------------------
41 entity contadorm10 is
42 port (  clk1 : in std_logic;
43         n_pre : in std_logic;
44         n_clr : in std_logic;
45         h , i : out std_logic_vector(3 downto 0));
46 end contadorm10;
47 ---------------------------------------------------------------------------------------------
48 architecture estructural of contadorm10 is
```

```
49 ------------------------------------------COMPONENTES------------------------------------------
50 component fftJKrs
51         Port (  J, K, clk        : in  std_logic;
52                 nPRE, nCLR       : in std_logic;
53                 q, nq            : out  std_logic);
54 end component fftJKrs;
55 ------------------------------------------SEÑALES----------------------------------------------
56 signal l: std_logic_vector(3 downto 0):="0000";
57 signal m: std_logic_vector(3 downto 0):="1111";
58 signal JK: std_logic_vector(3 downto 0):="0000";
59 -----------------------------------------------------------------------------------------------
60 begin
61 ------------------------------------------INSTANCIAS-------------------------------------------
62     inst_fftJKrs0:fftJKrs
63       port map(J=>JK(0), K=>JK(0), clk=>clk1, npre=>n_pre, nclr=>n_clr, q=>l(0), nq=>m(0));
64     inst_fftJKrs1:fftJKrs
65       port map(J=>JK(1), K=>JK(1), clk=>clk1, npre=>n_pre, nclr=>n_clr, q=>l(1), nq=>m(1));
66     inst_fftJKrs2:fftJKrs
67       port map(J=>JK(2), K=>JK(2), clk=>clk1, npre=>n_pre, nclr=>n_clr, q=>l(2), nq=>m(2));
68     inst_fftJKrs3:fftJKrs
69       port map(J=>JK(3), K=>JK(3), clk=>clk1, npre=>n_pre, nclr=>n_clr, q=>l(3), nq=>m(3));
70
71     JK(0)<='1';
72     JK(1)<=l(0) and m(3);
73     JK(2)<=(l(0) and l(1));
74     JK(3)<=(l(2) and JK(2)) or (l(0) and l(3));
75 ------------------------------------------SALIDAS----------------------------------------------
76 h <= l;
77 i <= m;
78 -----------------------------------------------------------------------------------------------
79 end estructural;
```

Script 9.2: Descripción en VHDL de un contador síncrono módulo 10.

El *script* 9.3 presenta la descripción en *VHDL* del circuito reductor de frecuencia, además se adiciona en el *script* 9.4 el código del *display* 7 segmentos que ya fue utilizado en las prácticas iniciales.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  ------------------------------------------------------------------------------------------
5  entity clock is
6  port (
7          clk      : in std_logic;
8          clkch    : out std_logic);
9  end clock;
10 ------------------------------------------------------------------------------------------
11 architecture comportamiento of clock is
12 signal contador : std_logic_vector (26 downto 0) :="00000000000000000000000000";
13 signal clk1: std_logic := '0';
14 begin
15  process (clk , clk1, contador )
16  begin
```

```
17          if clk'event and clk = '1' then
18                          if contador = "10111110101111000001111111" then
19                                  contador <= ( others => '0');
20                                  clk1 <= not(clk1);
21                                  else
22                                  contador <= std_logic_vector ( unsigned ( contador )+1);
23                          end if;
24          end if;
25   end process;
26  clkch <= clk1;
27  ------------------------------------------------------------------------------
28  end comportamiento;
```

Script 9.3: Descripción en VHDL del circuito reductor de frecuencia.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7   entity deco_7seg_WS is
8
9   Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
10         P : out  STD_LOGIC_VECTOR (6 downto 0));
11
12  end deco_7seg_WS;
13  ------------------------------------------------------------------------------
14  architecture Behavioral of deco_7seg_WS is
15
16  begin
17
18  with A select
19
20          p<=     "1111110" when "0000",
21                  "0110000" when "0001",
22                  "1101101" when "0010",
23                  "1111001" when "0011",
24                  "0110011" when "0100",
25                  "1011011" when "0101",
26                  "1011111" when "0110",
27                  "1110000" when "0111",
28                  "1111111" when "1000",
29                  "1111011" when "1001",
30                  "0000000" when others;
31  ------------------------------------------------------------------------------
32  end Behavioral;
```

Script 9.4: Código de utilizado para realizar la instancia(*display* 7 segmentos).

Finalmente se presentan los circuitos que realizan las instancias tanto como de el registro de entrada paralelo y salida serie, como el contador asíncrono módulo 10 en los *scripts* 9.5 y 9.6 respectivamente.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7  entity instanciaregpase is
8
9  Port (                  clk : in  std_logic;
10                         nCLR : in  std_logic;
11                         c_d: in std_logic;
12                         e_p: in std_logic_vector(3 downto 0);
13                         chclk: out std_logic;
14                         s_s:out std_logic;
15                         led:out std_logic_vector(3 downto 0));
16
17 end instanciaregpase;
18
19 architecture Behavioral of instanciaregpase is
20
21 ----------------------------------COMPONENTES-----------------------------------------
22
23 component RegParSer
24
25         port (
26                 clk : in std_logic;
27                 reset : in std_logic;
28                 c_d : in std_logic; -- Carga (0) o desplaza (1)
29                 e_p : in std_logic_vector (3 downto 0); -- Entrada
30                 s_s : out std_logic ); -- Salida serie
31 end component;
32
33
34
35
36 component clock
37                 port (
38                         clk     : in std_logic;
39                         clkch   : out std_logic);
40 end component;
41
42
43 ----------------------------------DECLARACION DE SEÑALES-----------------------------
44 signal signal_clkch: std_logic;
45 -------------------------------------------------------------------------------------
46 begin
47 ----------------------------------------INSTANCIAS-----------------------------------
48 inst_clock: clock
49         port map (                      clk=>clk,
50                                         clkch=>signal_clkch);
51
52
```

```
53 inst_RegParSer: RegParSer
54         port map (                    clk=>signal_clkch,
55                                       reset=> not(nCLR),
56                                       c_d=>c_d,
57                                       e_p=>e_p,
58                                       s_s=>s_s);
59 ------------------------------------------------------------------------------
60 chclk<=signal_clkch;
61 led<=e_p;
62 ------------------------------------------------------------------------------
63 end Behavioral;
```

Script 9.5: Código utilizado para realizar la instancia del registro entrada paralelo/salida serie.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity instanciacontador is
7
8  Port (                  clk : in  std_logic;
9                          nCLR : in  std_logic;
10                         chclk: out std_logic;
11                         display:out std_logic_vector(6 downto 0));
12
13 end instanciacontador;
14
15 architecture Behavioral of instanciacontador is
16
17 ------------------------------COMPONENTES-----------------------------------
18
19 component contadorm10
20                 port (  clk1 : in std_logic;
21                         n_pre : in std_logic;
22                         n_clr : in std_logic;
23                         h , i : out std_logic_vector(3 downto 0));
24 end component;
25
26
27 component deco_7seg_WS
28                 Port (  A : in  std_logic_vector (3 downto 0);
29                         P : out  std_logic_vector (6 downto 0));
30 end component;
31
32
33 component clock
34                 port (
35                         clk     : in std_logic;
36                         clkch   : out std_logic);
37 end component;
38
39
```

```
40 ------------------------------------DECLARACION DE SEÑALES-------------------------------------
41 signal signal_h:std_logic_vector(3 downto 0);
42 signal signal_clkch: std_logic;
43 -----------------------------------------------------------------------------------------------
44 begin
45 ------------------------------------------INSTANCIAS-------------------------------------------
46 inst_clock: clock
47         port map (                    clk=>clk,
48                                       clkch=>signal_clkch);
49
50
51 inst_contadorm10: contadorm10
52         port map (                    clk1=>signal_clkch,
53                                       n_pre=> '1',
54                                       n_clr=> nCLR,
55                                       h=>signal_h,
56                                       i=>open);
57
58 inst_deco_7seg_WS:deco_7seg_WS
59         port map (                    A=>signal_h,
60                                       P=>display);
61 -----------------------------------------------------------------------------------------------
62 chclk<=signal_clkch;
63 -----------------------------------------------------------------------------------------------
64 end Behavioral;
```

Script 9.6: Código utilizado para realizar la instancia del contador síncrono módulo 10.

9.4.1.4 Paso 4: Observación de los RTLs de los circuitos descritos

A continuación se presentan los esquemas RTLs de las descripciones de los circuitos realizadas en los *scripts* 9.1 y 9.2 (registro entrada paralelo/salida serie y contador síncrono módulo 10). Los códigos se sintetizaron utilizado un programa llamado *Quartus II*.

En la figura 9.13 se presenta el RTL del registro entrada paralelo/salida serie. El RTL presenta un MUX en el cual la señal de control es c_d, también presenta las entradas que son manejadas por la señal de control (la entrada paralelo al registro o el corrimiento que se produce en el registro que almacena los datos).

En la figura 9.14 se presenta el RTL del contador síncrono módulo 10, donde aparece algo no tan deseado en los circuitos, pues si se nota con detalle se ve que la compuerta JK[3] depende implícitamente de la salida del primer *flip-flop* ya que existe un tiempo de propagación entre la salida del primer *flip-flop* y las entadas de la compuerta JK[3], lo cual no afecta mucho en este circuito pero cuando se tienen circuitos de alta velocidad esto puede ocasionar inconvenientes ya que la señal de entrada al *flip-flop* fftJKrs:inst_fftJKrs3 no sería estable.

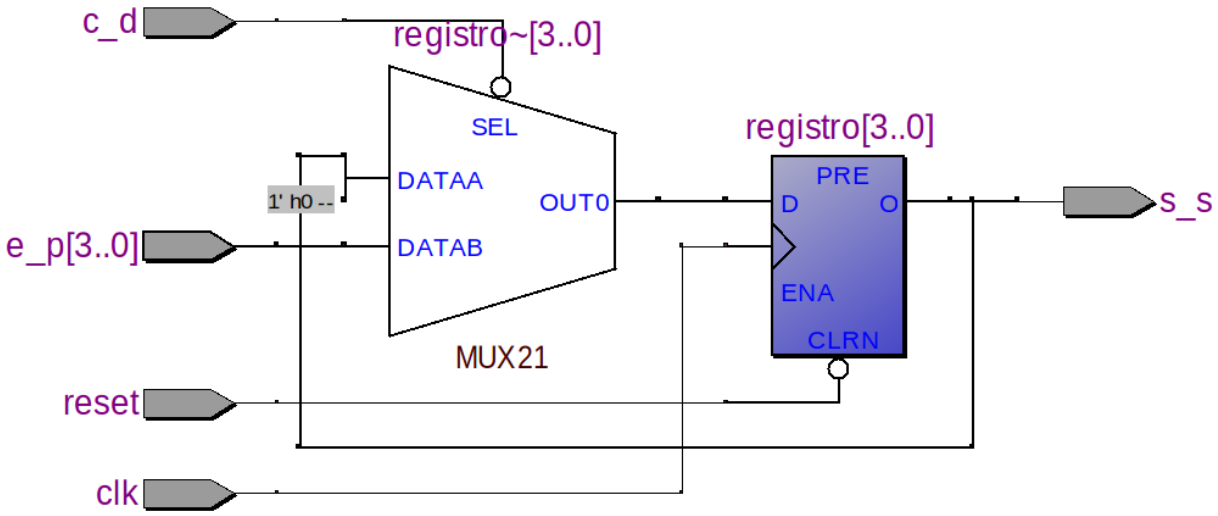


Figura 9.13: RTL de un registro con entrada paralelo/salida serie

FUENTE: Herramienta de simulación Quartus II

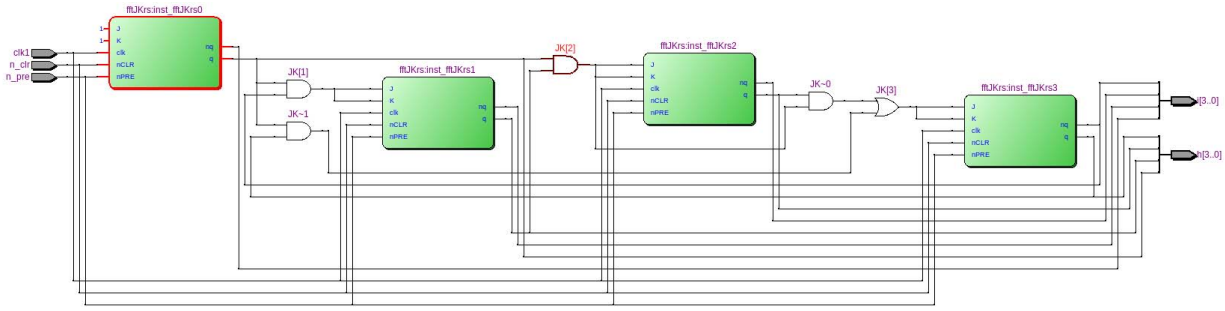


Figura 9.14: Contador síncrono módulo 10.

FUENTE: Herramienta de simulación Quartus II

9.4.1.5 Paso 5: Simulación de los circuitos a implementar

La figura 9.15 presenta la simulación del registro paralelo. Obsérvese que el primer dato a la salida (s_s), se presenta en 100 *ns* cuando la señal c_d (carga o desplazamiento) tiene el valor de 0. En 500 *ns*, ya se ha realizado la transmisión de los datos pero sigue manteniendo en su salida cero ya que así se diseño el circuito, en 600 *ns* se presentan las condiciones para que el circuito tome otro valor.

En la figura 9.16 se presenta la simulación del circuito módulo 10, nótese que a los 100 *ns* se inicia la cuenta y a los 1000 *ns* el contador vuelve a su estado inicial.

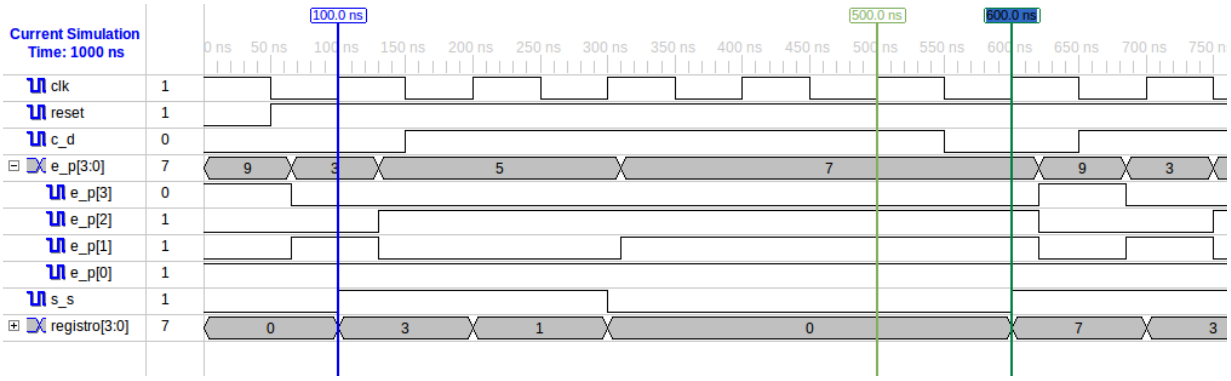


Figura 9.15: Simulación de un registro con entrada paralelo/salida serie

FUENTE: Herramienta de simulación ISE

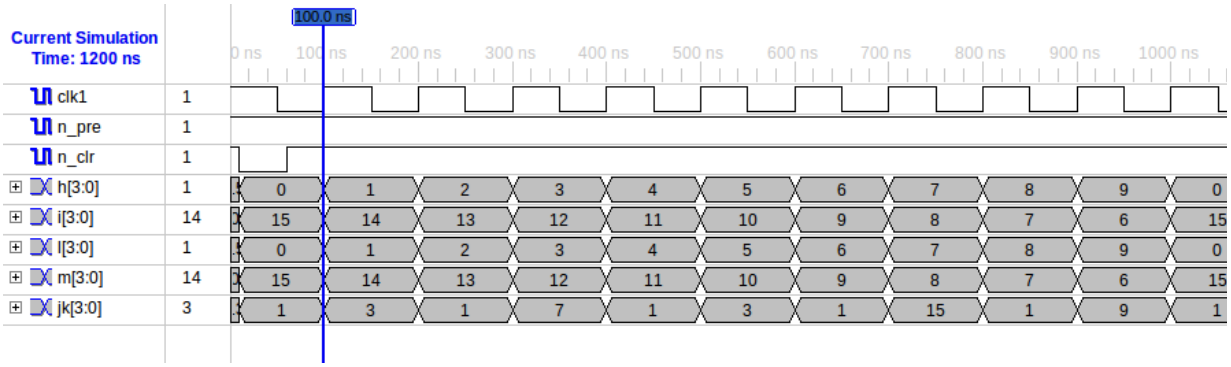


Figura 9.16: Simulación de un registro con entrada paralelo/salida serie

FUENTE: Herramienta de simulación ISE

9.4.1.6 Paso 6: Elementos necesarios para la implementación

Los módulos necesarios para la implementación de la práctica son:

- Digilent PmodSWT Switch Module Board. [8]
- 5 *LEDs* de la tarjeta de expansión de pines.
- Digilent PmodSSD Peripheral Module Board [8]

9.4.1.7 Paso 7: Asignación de pines

La asignación de pines para el registro de entrada serie/salida paralelo se presenta en el *script* 9.7

```
1 net clk loc= "P38"; #Entrada del reloj de la FPGA 50MHz
2 net nCLR loc= "P57"; #pulsador de la SIE
3 net c_d loc="P53";
4 net e_p<0> loc= "P33";
```

```
net e_p<1> loc= "P27";
net e_p<2> loc= "P24";
net e_p<3> loc= "P22";
net chclk loc= "P18"; #Reloj con periodo de 2 segundos
net s_s loc= "P32"; #salida del registro
net led<3> loc= "P48";
net led<2> loc= "P41";
net led<1> loc= "P36";
net led<0> loc= "P34";
```

Script 9.7: Archivo *ucf* del circuito que tiene las instancias

La asignación de pines para el contador síncrono módulo 10 se muestra en el *script* 9.8

```
net clk loc= "P38"; #Entrada del reloj de la FPGA 50MHz
net nCLR loc= "P53"; #pulsador de la SIE
net chclk loc= "P48"; #Reloj con periodo de 2 segundos
#net display<0> loc="P33"; # segmento a del display
#net display<1> loc="P27"; # segmento b del display
#net display<2> loc="P24"; # segmento c del display
#net display<3> loc="P22"; # segmento d del display
#net display<4> loc="P49"; # segmento e del display
#net display<5> loc="P47"; # segmento f del display
#net display<6> loc="P40"; # segmento g del display

net display<0> loc="P40"; # segmento a del display
net display<1> loc="P47"; # segmento b del display
net display<2> loc="P49"; # segmento c del display
net display<3> loc="P22"; # segmento d del display
net display<4> loc="P24"; # segmento e del display
net display<5> loc="P27"; # segmento f del display
net display<6> loc="P33"; # segmento g del displayplay
```

Script 9.8: Archivo *ucf* del circuito que tiene las instancias

9.4.1.8 Paso 8: Creación del archivo *Makefile*

Para llevar a cabo el procedimiento necesario en el presente paso, es preciso referirse a la guía de laboratorio 1, en donde se expone de forma detallada lo referente al *Makefile*, el cual es un archivo proporcionado por la persona encargada de guiar la clase, y al cual se le debe editar haciendo una pequeña modificación en la segunda linea, donde debe agregarse el nombre del fichero que se desea implementar.

Nota : No olvidar agregar las fuentes en el *Makefile* necesarias para realizar las instancias.

9.4.1.9 Paso 9: Implementación en la tarjeta *SIE*

Ya teniendo los archivos necesarios para la respectiva implementación, es preciso llevar a cabo el proceso necesario para una óptima implementación, para lo cual es necesario seguir los incisos mostrados a continuación:

- Conectar la tarjeta *SIE*
- Configuración del puerto *USB*
- Realizar la síntesis de la descripción
- Enviar información al procesador
- Acceder al procesador de la tarjeta
- Enviar información al *FPGA*

Es preciso mencionar que si se tiene alguna duda para llevar a cabo el proceso mencionado anteriormente, es necesario referirse a la guía de laboratorio 1, en donde se explica detalladamente cada uno de los anteriores.

9.5 RESUMEN

La finalidad de la práctica es implementar en la tarjeta SIE, un registro de entrada paralelo/salida serie y un contador módulo 10, para ello se retoman los siguientes conceptos:

- Registro con entrada serie/salida serie
- Registro con entrada serie/salida paralelo
- Registro con entrada paralelo/salida serie
- Registro con entrada paralelo/salida paralelo
- Contadores asíncronos
- Contadores síncronos

9.6 PREGUNTAS DE PRUEBA

- ¿Cuántos *flip-flops* son necesarios para implementar un contador módulo 5812 ?
- Hacer un esquema de un contador módulo 5.

9.7 EJERCICIO PROPUESTO

Un contador de anillo es un circuito secuencial, muy similar a registro de corrimiento. La salida en serie del registro de corrimiento, se retroalimenta al pin de entrada serie del registro. En el circuito resultante circula un patrón de bits alrededor del registro.

Sea n el número de flip-flops y, por tanto, el número de estados del contador. Los flip-flops del registro de corrimiento se rotulan X_0, X_1,...,X_n. El funcionamiento del contador se inicia con la entada de un 1 lógico . Esto hace que X_0 sea alta y X_1, X_2,...,X_n-_2, X_n-_1 sean bajas. En este momento, sólo hay un 1 lógico en el flip-flop X_0. En la siguiente transición de la señal de sincronia, el 1 lógico se transfiere del flip-flop X_0 al flip-flop X_1.

El proceso continúa hasta que el 1 lógico llega al final del registro de corrimiento, el flip-flop X_n-_1. En la transición del siguiente pulso de reloj, el 1 lógico se transfiere mediante la línea de realimentación al primer flip-flop en el registro de corrimiento, X_0. Después, el proceso se repite. En otras palabras, el 1 lógico recorre un ciclo a través del registro de corrimiento cada n pulsos de reloj. Así, el contador de anillo tiene un único estado para cada flip-flop. En la tabla 9.2 Se presenta la secuencia de estados.[30]

Estados	$\mathbf{X}_0$	$\mathbf{X}_1$	$\mathbf{X}_{\cdot\cdots i}$	$\mathbf{X}_n-_1$
Inicialización	1	0	0	0
Estado 1	0	1	0	0
Estado i	0	0	1	0
Estado $n-1$	0	0	0	1
Estado 0 (nuevo ciclo)	1	0	0	0

Tabla 9.2: Secuencia de estados de un contador de anillos.

Donde i varia desde 2 hasta $n-2$.

Presentados los conocimientos acerca del funcionamiento de un contador de anillo, proceder a realizar la implementación de éste.

LABORATORIO **10**

Máquinas de Estado.

El mundo entero se aparta cuando ve pasar a
un hombre que sabe a dónde va.

ANTOINE DE SAINT-EXUPÉRY

10.1 INTRODUCCIÓN

De la mano con los avances tecnológicos, los sistemas digitales presentan cada vez más exigencias en la realización de sus diseños, y por ende el hardware a describir es cada vez más complejo. Así pues, se presenta el diseño por medio de máquinas de estado como una herramienta que facilita de sobremanera la descripción de circuitos secuenciales de gran exigencia, que en gran parte son utilizados como circuitos de control debido a que una máquina de estados responde en función del estado actual del circuito y el valor de sus entradas.

10.2 OBJETIVOS

- Desarrollar un diagrama de estados para una secuencia determinada.
- Diseñar e implementar un circuito síncrono y secuencial en la tarjeta SIE.

10.3 MARCO TEÓRICO

La estructura de los sistemas secuenciales síncronos basa su funcionamiento en los elementos de memoria conocidos como *flip-flops*. La palabra sincronía se refiere a que cada uno de estos elementos de memoria que interactúan en un sistema se encuentran conectados a la misma señal de reloj, de forma tal que sólo se producirá un cambio de estado en el sistema cuando ocurra un flanco de disparo o un pulso en la señal de reloj [7].

Los circuitos lógicos secuenciales se pueden clasificar dentro de la categoría de circuitos conocidos como máquinas de estado, de las que existen dos tipos [4]: la máquina de estados de Moore y la máquina de estados de Mealy.

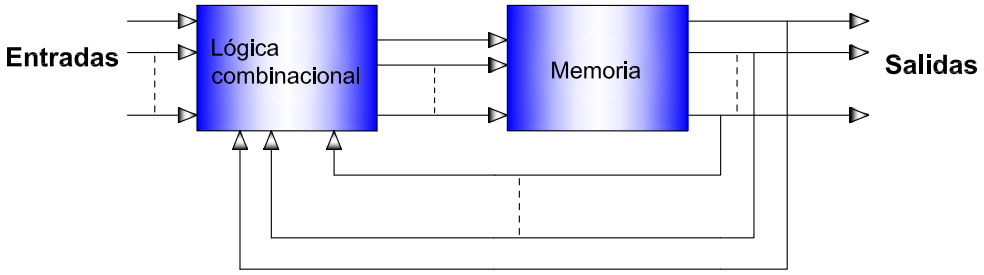


Figura 10.1: Máquina de estados Moore.

FUENTE: Los autores

En la figura 10.1 aparece la máquina de estados de Moore, donde la señal de salida sólo depende del estado en el cual se encuentre. Por otra parte, en la figura 10.2 se muestra un esquema de la máquina de estados de Mealy, donde la señal de salida depende tanto del estado en que se encuentre el sistema, como de la entrada que se aplica en determinado momento. [7].

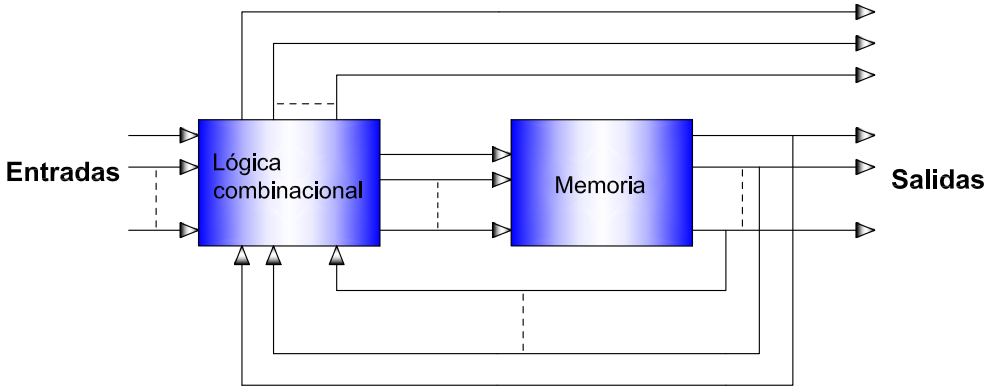


Figura 10.2: Máquina de estados Mealy.

FUENTE: Los autores

Para diseñar una máquina de estados es preciso obtener la función de salida y la función de transición entre los estados teniendo en cuenta las especificaciones del circuito, de acuerdo a esto es preciso seguir un proceso el cual se resume en los siguientes pasos:

1. Diagrama de estados

2. Tabla del estado siguiente

3. Tabla de transiciones de los *flip-flops*
4. Mapas de *Karnaugh*
5. Expresiones lógicas para las entradas de los *flip-flops*
6. Implementación
    - Esquema RTL
    - Descripción en VHDL

Para ilustrar el proceso a seguir para diseñar una máquina de estados se plantea realizar un contador de 0 a 2 y de 3 a 5 que haga el conteo de acuerdo a una entrada de control.

10.3.1 Diagrama de estados

En primer lugar, se describe el contador mediante un diagrama de estados, el cual muestra la progresión de estados por los que el contador avanza cuando se aplica una señal de reloj [4]. En la figura 10.3 se muestra el respectivo diagrama de estados del contador que se quiere implementar.

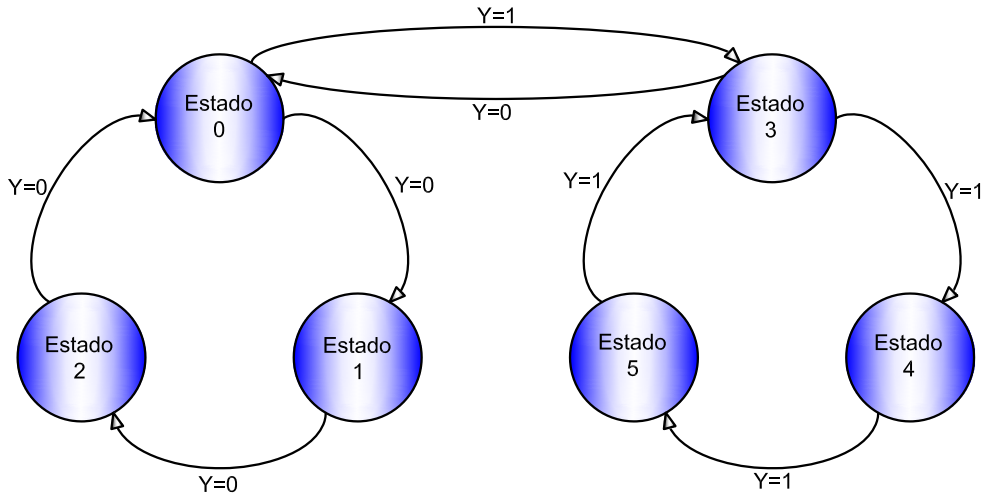


Figura 10.3: Diagrama de estados del contador.

FUENTE: Los autores

10.3.2 Tabla del estado siguiente

Una vez que se define el circuito secuencial mediante un diagrama de estados, el segundo paso consiste en obtener una tabla del estado siguiente, que enumera cada estado del contador (estado actual) junto con el correspondiente

estado siguiente. El estado siguiente es el estado al que el contador pasa desde su estado actual, al aplicar un impulso de reloj [4].

En la tabla 10.1 es posible notar la tabla del estado siguiente, la cual debe abarcar todas las posibles transiciones y cambios de estado que debe tener la máquina.

VAR	ESTADO ACTUAL			ESTADO SIGUIENTE		
Y	$\mathbf{Q_2}$	$\mathbf{Q_1}$	$\mathbf{Q_0}$	$\mathbf{Q_2}$	$\mathbf{Q_1}$	$\mathbf{Q_0}$
0	0	0	0	0	0	1
X	0	0	1	0	1	0
X	0	1	0	0	0	0
1	0	0	0	0	1	1
1	0	1	1	1	0	0
X	1	0	0	1	0	1
X	1	0	1	0	1	1
0	0	1	1	0	0	0

Tabla 10.1: Tabla del estado siguiente para el contador.

10.3.3 Tabla de transiciones de los *flip-flops*

El paso a seguir es definir el tipo de *flip-flops* que se van a utilizar y tener presente la tabla de transiciones del mismo, para el ejemplo se implementarán *flip-flops* tipo JK y la tabla 10.2 muestra sus respectivas transiciones.

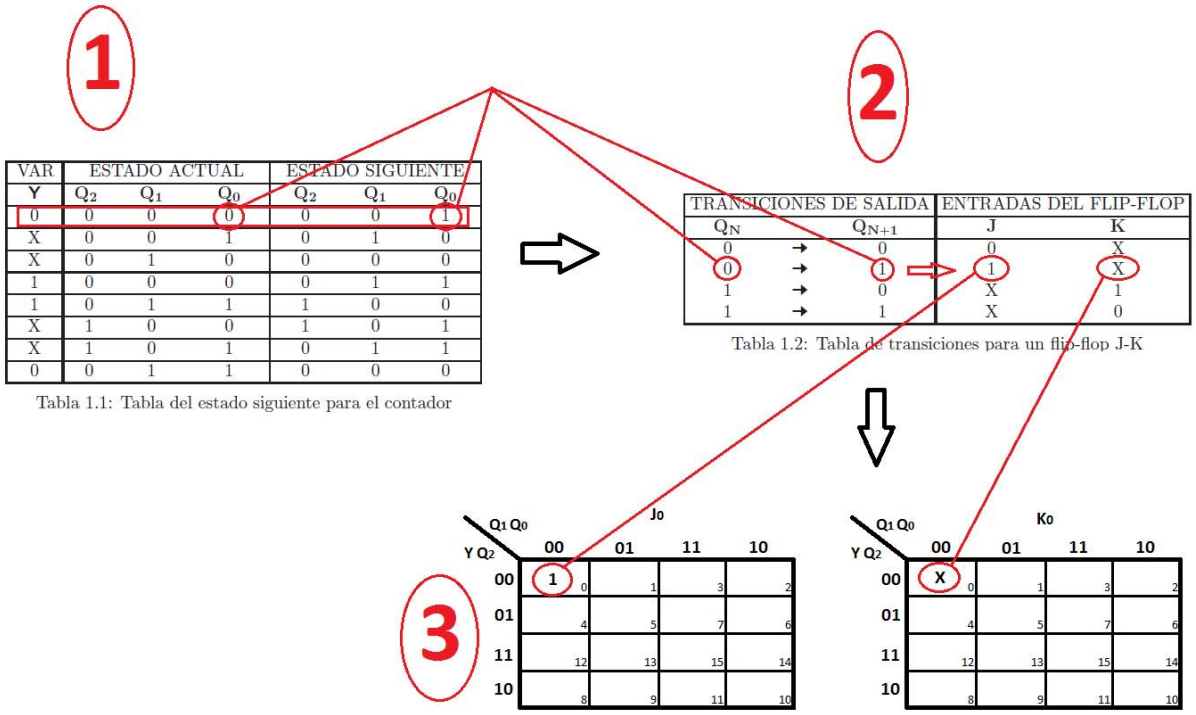
TRANSICIONES DE SALIDA			ENTRADAS DEL *FLIP-FLOP*	
$\mathbf{Q_N}$		$\mathbf{Q_{N+1}}$	**J**	**K**
0	→	0	0	X
0	→	1	1	X
1	→	0	X	1
1	→	1	X	0

Tabla 10.2: Tabla de transiciones para un *flip-flop* tipo JK.

10.3.4 Mapas de *Karnaugh*

Los mapas de *Karnaugh* se utilizan para determinar la lógica requerida para las entradas J y K de cada *flip-flop* del contador. Se debe utilizar un mapa de *Karnaugh* para la entrada J y otro para la entrada K de cada *flip-flop*. En este procedimiento de diseño, cada celda del mapa de *Karnaugh* representa uno de los estados actuales de la secuencia del contador enumerados en la tabla 10.1[4].

A partir de los estados J y K de la tabla de transiciones (Tabla 10.2) se introduce un 1, un 0 o una X en cada celda de la tabla correspondiente al estado actual, dependiendo de la transición de la salida Q del *flip-flop* en particular [4]. En la figura 10.4 se muestra un ejemplo de lo mencionado anteriormente pasando de los valores que se tienen en las tablas, a los mapas de *Karnaugh*.



VAR	ESTADO ACTUAL			ESTADO SIGUIENTE		
Y	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	1
X	0	0	1	0	1	0
X	0	1	0	0	0	0
1	0	0	0	0	1	1
1	0	1	1	1	0	0
X	1	0	0	1	0	1
X	1	0	1	0	1	1
0	0	1	1	0	0	0

Tabla 1.1: Tabla del estado siguiente para el contador

TRANSICIONES DE SALIDA		ENTRADAS DEL FLIP-FLOP	
Q_N	Q_{N+1}	J	K
0 →	0	0	X
0 →	1	1	X
1 →	0	X	1
1 →	1	X	0

Tabla 1.2: Tabla de transiciones para un flip-flop J-K

Figura 10.4: Procedimiento de utilización de mapas de *Karnaugh*.

FUENTE: Los autores

Ahora bien, en las figuras 10.5, 10.6 y 10.7 se muestran los mapas de *Karnaugh* completos para cada una de las entradas J y K de los tres *flip-flops* referentes a J_0, K_0, J_1, K_1, J_2 y K_2 respectivamente.

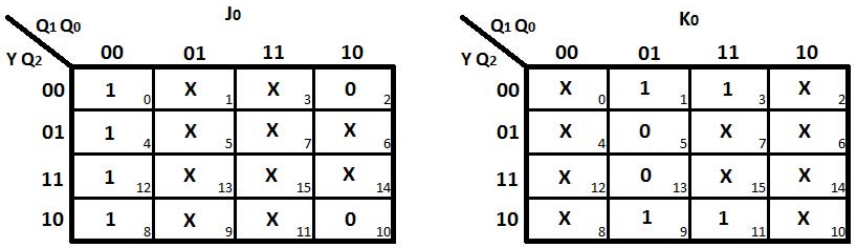


Figura 10.5: Mapas de *Karnaugh* para las entradas J0 y K0.

FUENTE: Los autores

10.3.5 Expresiones lógicas para las entradas de los *flip-flops*

Ahora, es posible a partir de los mapas de *Karnaugh*, obtener las expresiones para las entradas J y K de cada uno de los *flip-flops*.

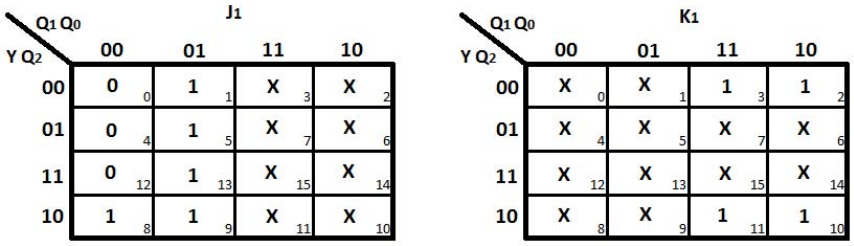


Figura 10.6: Mapas de *Karnaugh* para las entradas J1 y K1.

FUENTE: Los autores

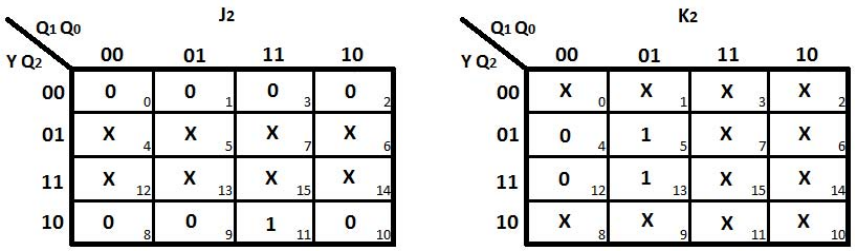


Figura 10.7: Mapas de *Karnaugh* para las entradas J2 y K2.

FUENTE: Los autores

$$J_0 = \overline{Q_1} \tag{10.1}$$

$$K_0 = \overline{Q_2} \tag{10.2}$$

$$J_1 = Q_0 + (Y * \overline{Q_2}) \tag{10.3}$$

$$K_1 = 1 \tag{10.4}$$

$$J_2 = (Y * Q_1 * Q_0) \tag{10.5}$$

$$K_2 = \overline{Q_0} \tag{10.6}$$

10.3.6 Implementación

Para la implementación es preciso realizar un esquema RTL con toda la información que se tiene hasta el momento.

10.3.6.1 Esquema RTL

En este punto lo único que resta es realizar un esquema de los tres *flip-flops* junto con las compuertas necesarias para obtener las respectivas entradas J y K de cada uno de estos. Así pues, en la figura 10.8 se muestra un esquema del resultado del procceso.

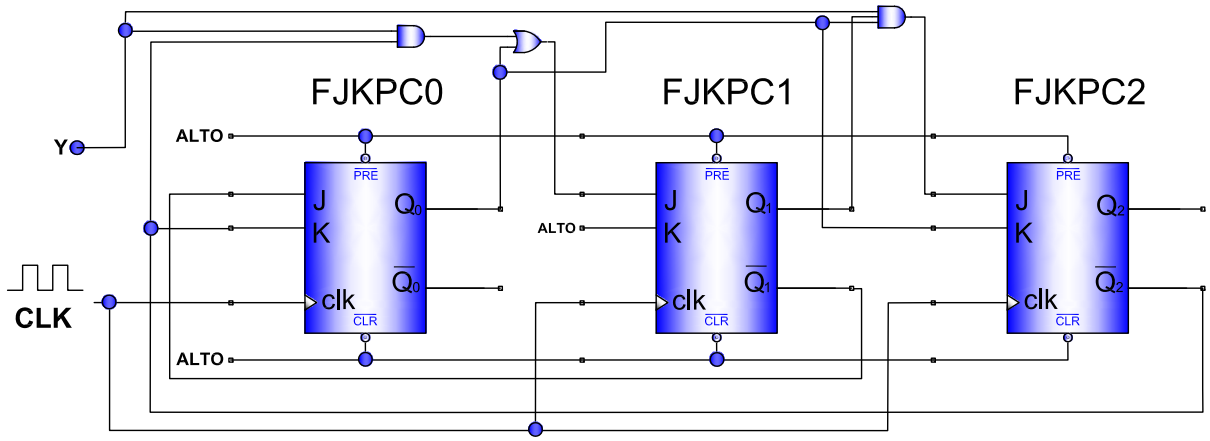


Figura 10.8: Planteamiento del esquema RTL del contador.

FUENTE: Los autores

10.3.6.2 Descripción en VHDL

Para la descripción en VHDL del circuito planteado que se muestra en la figura 10.8, es necesario un módulo interno que describa el *flip-flop* JK, y adicionalmente una descripción donde se realice la instancia del *flip-flop* y se describa el circuito principal. Así pues, en el *script* 10.1, se muestra la descripción del *flip-flop* tipo JK.

```
1  ----------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------
7  entity flipflopJK is
8
9       Port (     J, K, clk        : in  std_logic;
10                 nPRE, nCLR       : in std_logic;
11                 q, nq            : out  std_logic);
12 end flipflopJK;
13 ----------------------------------------------------------------------------
14 architecture Behavioral of flipflopJK is
15
16 signal signal_q: std_logic:='0';
17
18 begin
```

```
19 ----------------------------------------------------------------------------------
20 process(clk, nPRE, nCLR) -- inicio del process
21  begin
22           if nCLR='0' then
23                 signal_q <='0'; -- Reset negado
24           elsif nPRE='0' then
25                 signal_q <='1'; -- Set negado
26           elsif clk'event and clk='1' then -- Detecta el flanco del reloj
27
28                          if(J='0' and K='1') then
29                                  signal_q <='0';
30                          elsif (J='1' and K='0') then
31                                  signal_q <='1';
32                          elsif (J='0' and K='0') then
33                                  signal_q <= signal_q;
34                          else
35                                  signal_q <=not(signal_q);
36                          end if;
37          end if;
38 end process;
39 ------------------------------------SALIDAS---------------------------------------
40 q <= signal_q;
41 nq <= not(signal_q);
42 end Behavioral;
43 ----------------------------------------------------------------------------------
```

Script 10.1: Descripción en VHDL del *flip-flop* JK.

Con la descripción del *flip-flop* JK, se procede a describir el circuito principal, tal como se muestra en el *script* 10.2

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity contador is
8
9  Port (  Y: in  STD_LOGIC;
10         clk,nPRE,nCLR : in  STD_LOGIC;
11         Conteo : out  STD_LOGIC_VECTOR(2 downto 0));
12
13
14 end contador;
15
16 architecture Behavioral of contador is
17 ------------------------------------COMPONENTES-----------------------------------
18 component flipflopJK
19                 Port (  J, K, clk       : in  std_logic;
20                         nPRE, nCLR      : in std_logic;
21                         q, nq           : out  std_logic);
22 end component;
```

```
23 ----------------------------------DECLARACION DE SEÑALES-----------------------------------
24 signal signal_J:std_logic_vector(2 downto 0);
25 signal signal_K:std_logic_vector(2 downto 0);
26 signal signal_Q:std_logic_vector(2 downto 0);
27 signal signal_nQ:std_logic_vector(2 downto 0);
28 -------------------------------------------------------------------------------------------
29 begin
30 ----------------------------------INSTANCIAS-----------------------------------------------
31 inst_flipflopJK_1:flipflopJK
32         port map (      J=>signal_J(0),
33                         K=>signal_K(0),
34                         clk=>clk,
35                         nPRE=>nPRE,
36                         nCLR=>nCLR,
37                         q=>signal_Q(0),
38                         nq=>signal_nQ(0));
39
40 inst_flipflopJK_2:flipflopJK
41         port map (      J=>signal_J(1),
42                         K=>signal_K(1),
43                         clk=>clk,
44                         nPRE=>nPRE,
45                         nCLR=>nCLR,
46                         q=>signal_Q(1),
47                         nq=>signal_nQ(1));
48
49 inst_flipflopJK_3:flipflopJK
50         port map (      J=>signal_J(2),
51                         K=>signal_K(2),
52                         clk=>clk,
53                         nPRE=>nPRE,
54                         nCLR=>nCLR,
55                         q=>signal_Q(2),
56                         nq=>signal_nQ(2));
57 ----------------------------------CONEXIONES-----------------------------------------------
58 signal_J(0)<=signal_nQ(1);
59 signal_K(0)<=signal_nQ(2);
60 signal_J(1)<=((Y)and(signal_nQ(2)))or(signal_Q(0));
61 signal_K(1)<='1';
62 signal_J(2)<=((Y)and(signal_Q(1))and(signal_Q(0)));
63 signal_K(2)<=signal_Q(0);
64 ----------------------------------SALIDAS--------------------------------------------------
65 Conteo(0)<=signal_Q(0);
66 Conteo(1)<=signal_Q(1);
67 Conteo(2)<=signal_Q(2);
68
69 end Behavioral;
70 -------------------------------------------------------------------------------------------
```

Script 10.2: Descripción en VHDL del contador por medio de flip flops JK.

10.3.6.3 Simulación del circuito descrito por medio de instancias del *flip-flop* tipo JK

En la figura 10.9 se muestra la simulación correspondiente a la descripción realizada en el *script* 10.2, donde se nota el marcador en 1800 `ns`, mostrando la transición del estado 4 al estado 5, a la detección de un flanco de subida del reloj, y con la entrada de control Y=1, mostrando el comportamiento esperado.

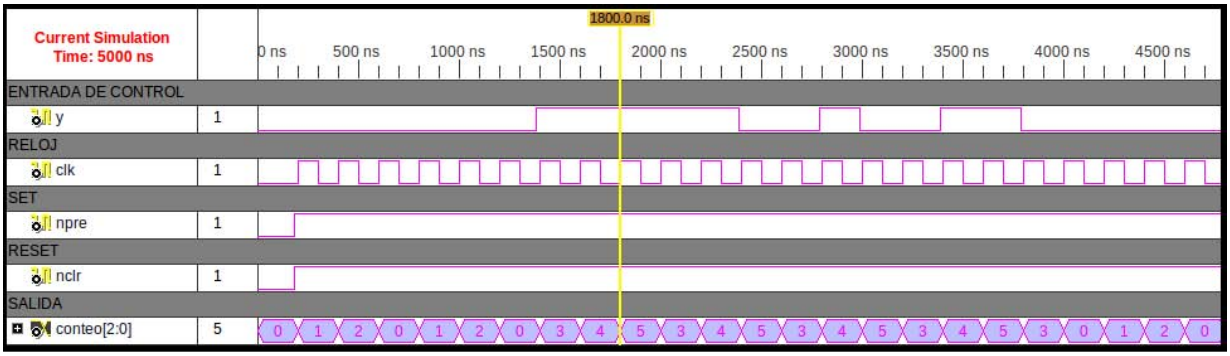


Figura 10.9: Simulación del contador descrito por medio de *flip-flops* tipo JK.

FUENTE: Herramienta de simulación ISE

10.3.6.4 RTL generado del circuito descrito por medio de instancias del *flip-flop* tipo JK

Como se muestra en la figura 10.10, el diagrama RTL mostrado por una herramienta llamada `quartusII`, es muy similar al planteado inicialmente en la figura 10.1 antes de empezar a describir el circuito.

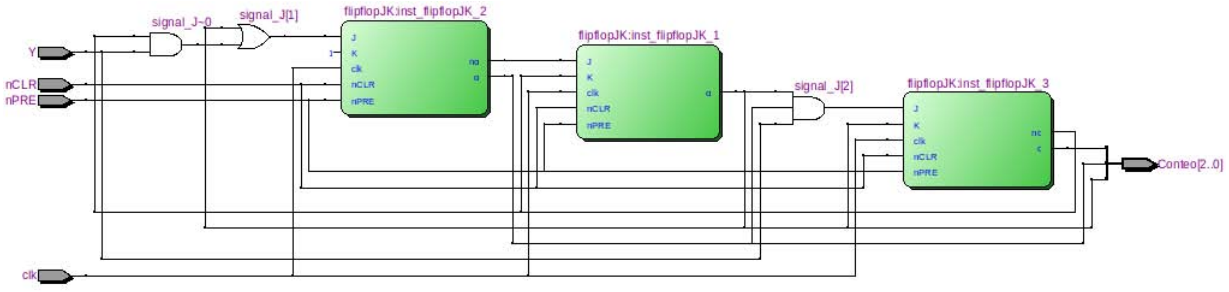


Figura 10.10: RTL del contador descrito por medio de *flip-flops* tipo JK.

FUENTE: Herramienta de simulación Quartus II

10.3.7 Descripción de una máquina de estados con VHDL

Una máquina de estados puede ser codificada con facilidad mediante una descripción de alto nivel en VHDL. Esta descripción supone el uso de declaraciones `case-when` las cuales determinan, en un caso particular, el valor que tomará el siguiente estado. Por otro lado, la transición entre estados se realiza por medio de senten-

cias `if-then-else`, de tal forma que éstas se encargan de establecer la lógica que seguirá en la descripción para realizar la asignación del estado [7].

Ahora bien, para declarar los estados es necesario utilizar la declaración `type`. teniendo en cuenta el nombre representativo de cada uno de los estados, tal como se muestra en el *script* 10.3, así como las señales utilizadas para el estado actual y el estado siguiente.

```
type estados is (estado1, estado2, estado3, estado4);
signal estado_presente, estado_futuro:estados;
```

Script 10.3: Declaración de estados en VHDL.

Seguidamente, se proceden a describir tres procesos: el registro de estados, la lógica del estado siguiente y la lógica de salida. En el registro de estados se coloca la respuesta del circuito a las señales de reset y al detectar un flanco del reloj, lo cual indica que a una entrada de `reset=1` el circuito pasa al estado CERO, como estado inicial del proceso, y que cada vez que detecte un flanco de reloj, el circuito pase del estado presente al estado futuro; dicha descripción se muestra en el *script* 10.4.

```
process (clk,reset)
        begin
                if (reset='1') then presente<=CERO;
                elsif(clk='1' and clk'event) then presente<=futuro;
                end if;
end process;
```

Script 10.4: Descripción de un registro de estados en VHDL.

En la descripción de la lógica del estado siguiente se procede a describir, todas las posibles transiciones que puedan suceder entre los estados, ya sea dependiendo de alguna entrada de control(**Y**), o del estado actual en el que se encuentre la máquina. En el *script* 10.5 se muestra la descripción para la máquina de estados de la figura 10.3.

```
process(presente,Y)
        begin
                case presente is

                        when CERO=> if (Y='0') then futuro<=UNO;
                                    else futuro<=TRES;
                                    end if;

                        when UNO=> futuro<=DOS;

                        when DOS=> futuro<=CERO;

                        when TRES=> if (Y='0') then futuro<=CERO;
                                    else futuro<=CUATRO;
                                    end if;

                        when CUATRO=> futuro<=CINCO;

                        when CINCO=> futuro<=TRES;
```

```
20                 end case;
21         end process;
```

Script 10.5: Descripción de la lógica del estado siguiente en VHDL.

En la lógica de salida, se describe la salida del circuito en cada uno de los estados tal como lo muestra el *script* 10.6 para el diagrama de estados de la figura 10.3.

```
1  process(presente)
2          begin
3                  case presente is
4
5                          when CERO=> Conteo<="000";
6                          when UNO=> Conteo<="001";
7                          when DOS=> Conteo<="010";
8                          when TRES=> Conteo<="011";
9                          when CUATRO=> Conteo<="100";
10                         when CINCO=> Conteo<="101";
11
12                 end case;
13         end process;
```

Script 10.6: Descripción la lógica de salida para el diagrama de la figura 10.3.

Finalmente, en el *script* 10.7 se muestra la descripción completa del circuito contador por medio de una máquina de estados.

```
1  ----------------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  ----------------------------------------------------------------------------------
7  entity maquinaTYPE is
8
9          Port ( clk : in STD_LOGIC;
10                                reset : in STD_LOGIC;
11                                Y : in STD_LOGIC;
12                                Conteo : out STD_LOGIC_vector(2 downto 0));
13
14 end maquinaTYPE;
15 ----------------------------------------------------------------------------------
16 architecture Behavioral of maquinaTYPE is
17 ----------------------------------------------------------------------------------
18 type estados is (CERO,UNO,DOS,TRES,CUATRO,CINCO);
19 signal presente,futuro:estados;
20
21 begin
22 ------------------------------REGISTRO DE ESTADOS-------------------------------
23 process (clk,reset)
24         begin
25                         if (reset='1') then presente<=CERO;
26                         elsif(clk='1' and clk'event) then presente<=futuro;
```

```
27                          end if;
28          end process;
29  ----------------------------LOGICA DEL ESTADO SIGUIENTE-----------------------------
30  process(presente,Y)
31          begin
32                  case presente is
33
34                          when CERO=> if (Y='0') then futuro<=UNO;
35                                                      else futuro<=TRES;
36                                                      end if;
37
38                          when UNO=> futuro<=DOS;
39
40                          when DOS=> futuro<=CERO;
41
42                          when TRES=> if (Y='0') then futuro<=CERO;
43                                                      else futuro<=CUATRO;
44                                                      end if;
45
46                          when CUATRO=> futuro<=CINCO;
47
48                          when CINCO=> futuro<=TRES;
49                  end case;
50          end process;
51  ----------------------------------LOGICA DE SALIDA----------------------------------
52  process(presente)
53          begin
54                  case presente is
55
56                          when CERO=> Conteo<="000";
57                          when UNO=> Conteo<="001";
58                          when DOS=> Conteo<="010";
59                          when TRES=> Conteo<="011";
60                          when CUATRO=> Conteo<="100";
61                          when CINCO=> Conteo<="101";
62
63                  end case;
64          end process;
65  ------------------------------------------------------------------------------------
66  end Behavioral;
67  ------------------------------------------------------------------------------------
```

Script 10.7: Descripción la lógica del estado siguiente en VHDL.

10.3.7.1 Simulación del circuito descrito utilizando la sentencia *type*

En la figura 10.11 se muestra la simulación correspondiente a la descripción realizada en el *script* 10.7, donde se nota el marcador en 1800 `ns`, mostrando la transición del estado 0 al estado 1, a la detección de un flanco de subida del reloj, y con la entrada de control Y=0, mostrando el comportamiento esperado.

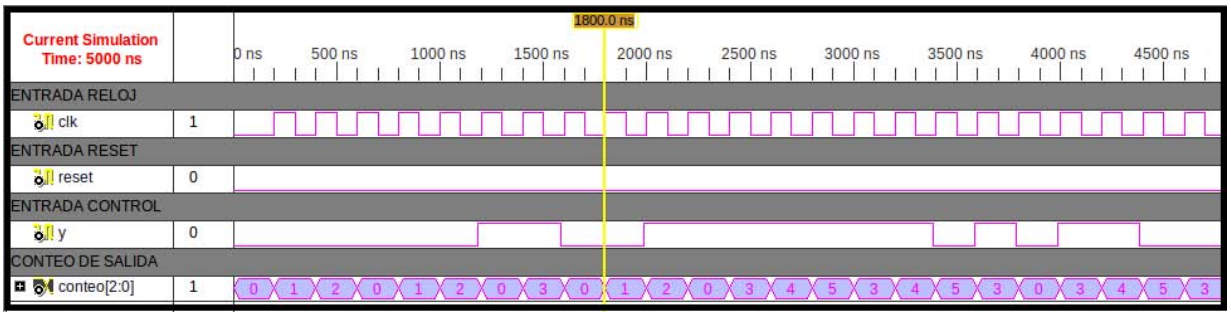


Figura 10.11: Simulación del contador descrito por medio de la declaración *type* en VHDL.

FUENTE: Herramienta de simulación ISE

10.3.7.2 RTL generado del circuito descrito utilizando la sentencia *type*

Ahora bien, el diagrama RTL generado a través de la herramienta *quartus II*, se muestra en la figura 10.12, donde es preciso resaltar la no aparición de *flip-flops* directamente, en su lugar una representación por medio de un bloque (presente), y unas cuantas compuertas lógicas apuntando a la salida.

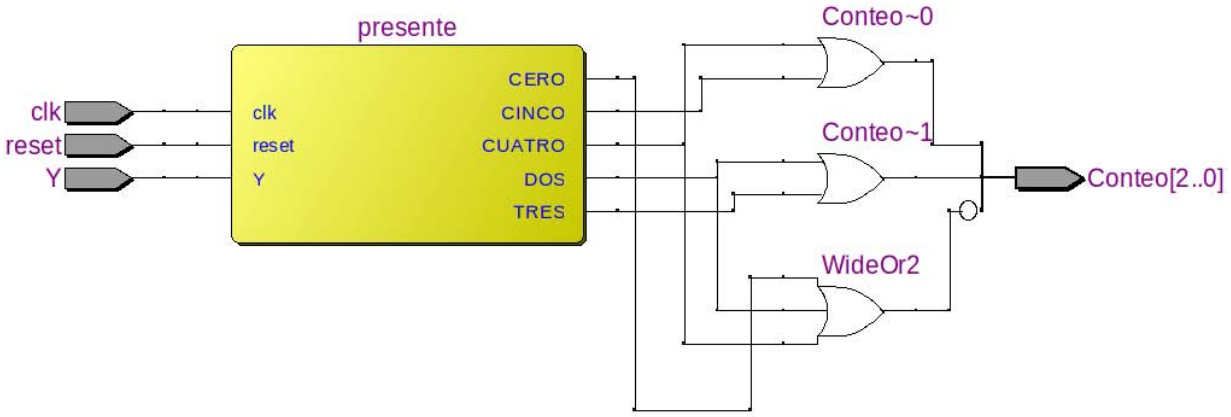


Figura 10.12: RTL del contador descrito por medio de la declaración *type* en VHDL.

FUENTE: Herramienta de simulación Quartus II

Dentro del bloque `presente`, es posible notar la máquina de estados de la figura 10.13, donde se plasma un diagrama de estados muy similar al planteado en el inicio del ejemplo, en la figura 10.3.

10.4 PROCEDIMIENTO GENERAL

Se plantea describir en VHDL e implementar en la tarjeta SIE, un circuito que cumpla con las cuatro secuencias especificadas en la figura 10.4.1.2. El circuito debe reunir las siguientes especificaciones:

- ✔ Debe tener una entrada clk.

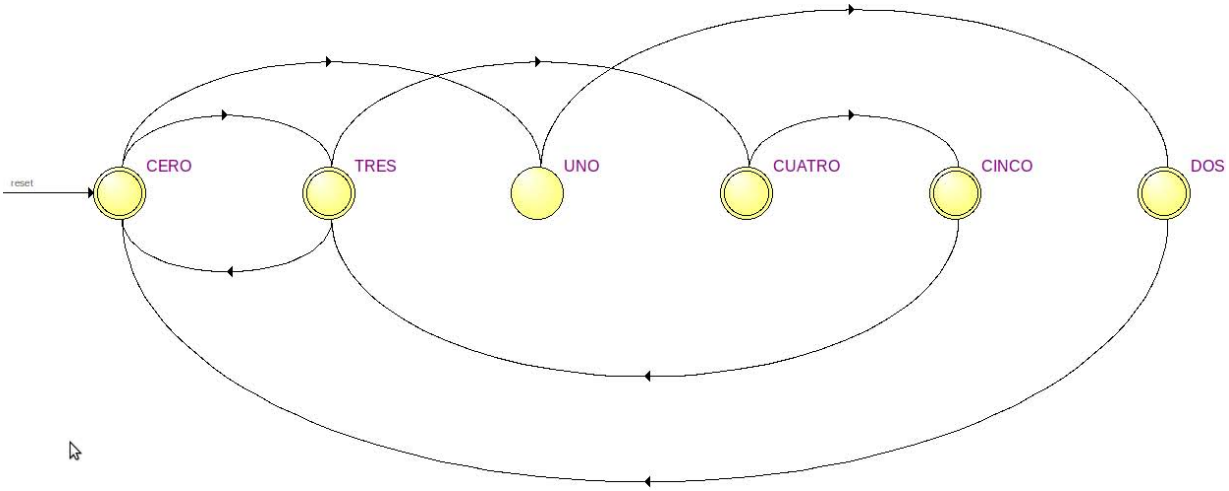


Figura 10.13: Diagrama de estados del RTL del contador descrito por medio de la declaración *type* en VHDL.

FUENTE: Herramienta de simulación Quartus II

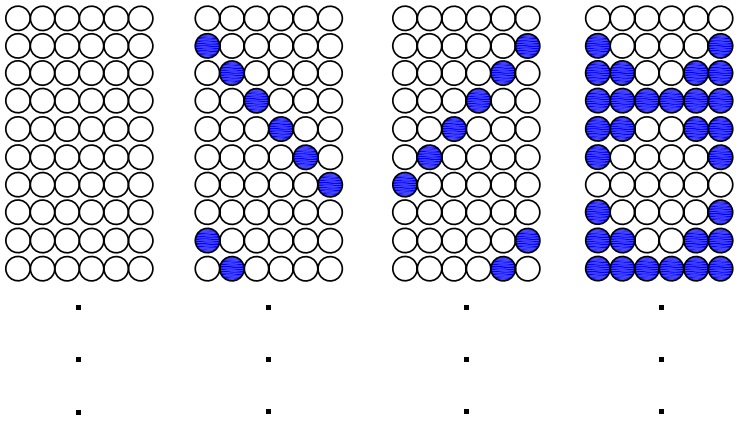


Figura 10.14: Representación de la secuencia de encendido y apagado de los LEDs, para el ejercicio planteado.

FUENTE: Los autores

- ✔ Debe tener una entrada para reiniciar el circuito en cualquier momento.
- ✔ Se desea implementar con máximo 2 entradas de control.
- ✔ La salida del circuito debe estar representada por medio de seis LEDs.

Para llevar a cabo la respectiva implementación del circuito planteado anteriormente, se recomienda seguir los siguientes pasos:

- Diagrama de estados
- Identificación de entradas y salidas
- Tabla del estado siguiente
- Descripción en VHDL
- Simulación
- Identificación de dispositivos requeridos para la implementación
- Asignación de pines creando el archivo de extensión .ucf
- Implementación

10.4.1 PROCEDIMIENTO PASO A PASO

Teniendo una idea general de las especificaciones del problema y de los pasos a seguir, se procede a realizar un análisis detallado del mismo.

10.4.1.1 Paso 1: Diagrama de estados

En la figura 10.15 se presenta el diagrama de estados que cumple con las especificaciones del problema.

10.4.1.2 Paso 2: Identificación de entradas y salidas

En la caja representativa de la figura 10.16 se observan tres entradas: una entrada de control de dos bits (**Y**), una entrada de un bit indicando el reinicio del circuito (reset) y una entrada de reloj (clk). El módulo también muestra una salida de seis bits, en la cual se mostrará la secuencia de luces planteada en la figura .

10.4.1.3 Paso 3: Tabla del estado siguiente

En la tabla 10.3 se plasma todas las posibles transiciones que se pueden presentar entre el estado actual y el estado siguiente para el circuito de la secuencia de luces.

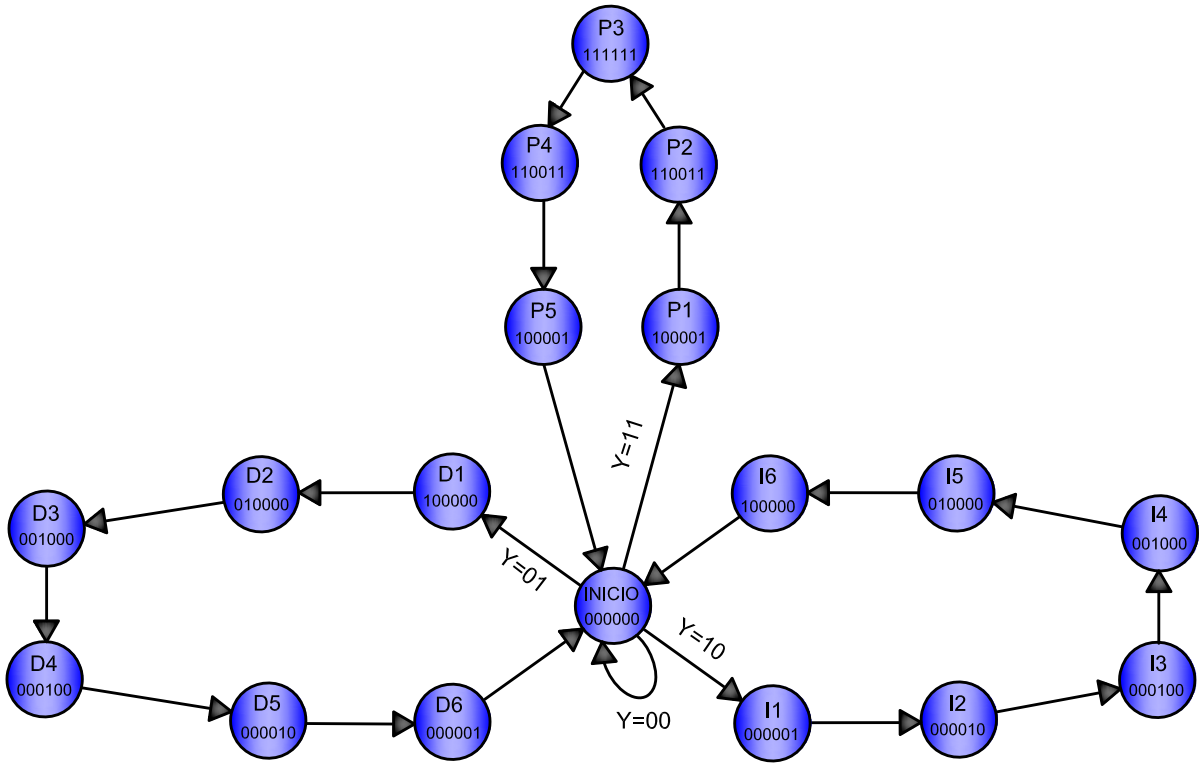


Figura 10.15: Diagrama de estados de la secuencia de luces.

FUENTE: Los autores

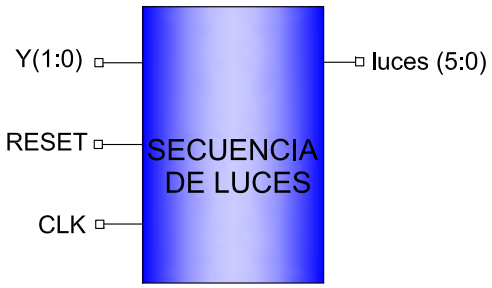


Figura 10.16: Módulo representativo del circuito de la secuencia de luces.

FUENTE: Los autores

VAR	ESTADO ACTUAL						ESTADO SIGUIENTE					
Y	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0
00	0	0	0	0	0	0	0	0	0	0	0	0
01	0	0	0	0	0	0	1	0	0	0	0	0
X	1	0	0	0	0	0	0	1	0	0	0	0
X	0	1	0	0	0	0	0	0	1	0	0	0
X	0	0	1	0	0	0	0	0	0	1	0	0
X	0	0	0	1	0	0	0	0	0	0	1	0
X	0	0	0	0	1	0	0	0	0	0	0	1
X	0	0	0	0	0	1	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	1
X	0	0	0	0	0	1	0	0	0	0	1	0
X	0	0	0	0	1	0	0	0	0	1	0	0
X	0	0	0	1	0	0	0	0	1	0	0	0
X	0	0	1	0	0	0	0	1	0	0	0	0
X	0	1	0	0	0	0	1	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	1	0	0	0	0	1
X	1	0	0	0	0	1	1	1	0	0	1	1
X	1	1	0	0	1	1	1	1	1	1	1	1
X	1	1	1	1	1	1	1	1	0	0	1	1
X	1	1	0	0	1	1	1	0	0	0	0	1
X	1	0	0	0	0	1	0	0	0	0	0	0

Tabla 10.3: Tabla del estado siguiente de la secuencia de luces.

10.4.1.4 Paso 4: Descripción en VHDL

Ya teniendo la certeza de todas las transiciones por las que puede pasar la máquina de estados, en el *script* 10.8 se muestra la descripción en VHDL del circuito que cumple con los requisitos planteados en el problema.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  ----------------------------------------------------------------------------
6  entity secuencia_de_luces is
7
8  Port ( clk : in STD_LOGIC;
9         reset : in STD_LOGIC;
10        Y : in STD_LOGIC_VECTOR(1 downto 0);
11        luces : out STD_LOGIC_vector(5 downto 0));
12 end secuencia_de_luces;
13 ----------------------------------------------------------------------------
14 architecture Behavioral of secuencia_de_luces is
15 ----------------------------------------------------------------------------
16 type estados is (INICIO,I1,I2,I3,I4,I5,I6,P1,P2,P3,P4,P5,D1,D2,D3,D4,D5,D6);
17 signal presente,futuro:estados;
18 ----------------------------------------------------------------------------
```

```
19  begin
20  ----------------------------------REGISTRO DE ESTADOS----------------------------------
21  process (clk,reset)
22          begin
23                          if (reset='1') then presente<=INICIO;
24                          elsif(clk='1' and clk'event) then presente<=futuro;
25                          end if;
26          end process;
27  ------------------------------LOGICA DEL ESTADO SIGUIENTE------------------------------
28  process(presente,Y)
29          begin
30                  case presente is
31
32                          when INICIO=> if (Y="01") then futuro<=D1;
33                                        elsif (Y="10") then futuro<=I1;
34                                        elsif (Y="11") then futuro<=P1;
35                                        else     futuro<=INICIO;
36                                        end if;
37
38                          when I1=> futuro<=I2;
39                          when I2=> futuro<=I3;
40                          when I3=> futuro<=I4;
41                          when I4=> futuro<=I5;
42                          when I5=> futuro<=I6;
43                          when I6=> futuro<=INICIO;
44
45                          when P1=> futuro<=P2;
46                          when P2=> futuro<=P3;
47                          when P3=> futuro<=P4;
48                          when P4=> futuro<=P5;
49                          when P5=> futuro<=INICIO;
50
51                          when D1=> futuro<=D2;
52                          when D2=> futuro<=D3;
53                          when D3=> futuro<=D4;
54                          when D4=> futuro<=D5;
55                          when D5=> futuro<=D6;
56                          when D6=> futuro<=INICIO;
57
58                  end case;
59          end process;
60  ----------------------------------LOGICA DE SALIDA----------------------------------
61  process(presente)
62          begin
63                  case presente is
64
65                          when INICIO=> luces<="000000";
66                          when D1=> luces<="100000";
67                          when D2=> luces<="010000";
68                          when D3=> luces<="001000";
69                          when D4=> luces<="000100";
70                          when D5=> luces<="000010";
71                          when D6=> luces<="000001";
```

```
72
73                          when P1=> luces <="100001";
74                          when P2=> luces <="110011";
75                          when P3=> luces <="111111";
76                          when P4=> luces <="110011";
77                          when P5=> luces <="100001";
78
79                          when I1=> luces <="000001";
80                          when I2=> luces <="000010";
81                          when I3=> luces <="000100";
82                          when I4=> luces <="001000";
83                          when I5=> luces <="010000";
84                          when I6=> luces <="100000";
85
86                  end case;
87          end process;
88  end Behavioral;
```

Script 10.8: Descripción en VHDL del circuito de la secuencia de luces.

10.4.1.5 Paso 5: Simulación

En la figura 10.17 se muestra la simulación del circuito de secuencia de luces implementados con máquina de estados, donde se nota el marcador en 2600 `ns` mostrando que a una entrada de control de Y=11,el circuito cambia las señales de salida de 100001 a 110011, es decir realiza una transición del estado P1 al estado P2 como era de esperarse.

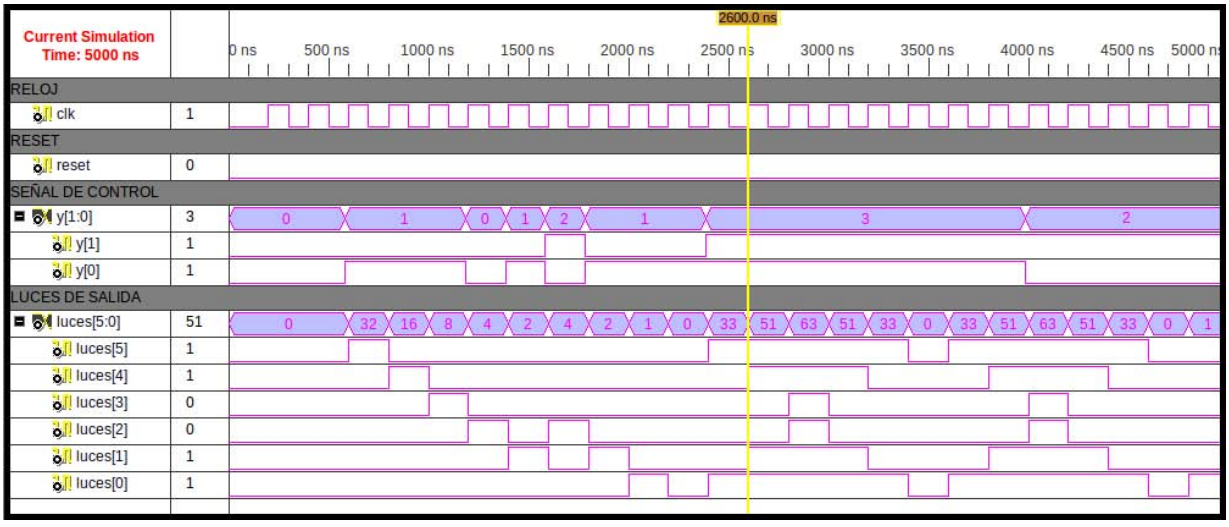


Figura 10.17: Simulación del circuito de secuencia de luces.

FUENTE: Herramienta de simulación ISE

10.4.1.6 Paso 6: Identificación de dispositivos requeridos para la implementación

Los módulos necesarios para la implementación de la práctica son:

- 1 pulsador que cumpla la función de resetear el circuito en cualquier momento.
- 2 Switches de la tarjeta de expansión de pines ó 2 Switches del Digilent PmodSWT Switch Module Board. [8]
- 6 Leds de la tarjeta de expansión de pines.

10.4.1.7 Paso 7: Asignación de pines creando el archivo de extensión .ucf

La construcción del archivo `.ucf` se muestra en el *script* 10.9. Los pines asignados pertenecen en su totalidad a la tarjeta de expansión de pines, cabe resaltar que también es posible utilizar módulos PMOD conectados a la misma tarjeta.

```
1  net clk loc="P38"; # Reloj de la SIE.
2
3  net reset loc="P53"; # pulsador para el reset
4
5  net Y(0) loc="P65"; # switch para señal de control
6  net Y(1) loc="P70"; # swich para señal de control
7
8
9  net luces(0) loc="P23"; # LED
10 net luces(1) loc="P26"; # LED
11 net luces(2) loc="P32"; # LED
12 net luces(3) loc="P34"; # LED
13 net luces(4) loc="P36"; # LED
14 net luces(5) loc="P41"; # LED
```

Script 10.9: Archivo ucf

10.4.1.8 Paso 8: Implementación

Ya teniendo los archivos necesarios para la respectiva implementación, es preciso llevar a cabo dicho proceso, para lo cual es necesario realizar los pasos mostrados a continuación:

- Comprobar la existencia de los archivos necesarios para sintetizar e implementar
  - `Makefile`
  - `secuencia_de_luces.vhd`
  - `secuencia_de_luces.ucf`

- Conectar la tarjeta SIE
- Configuración del puerto USB
- Realizar la síntesis de la descripción
- Enviar información al procesador
- Acceder al procesador de la tarjeta
- Enviar información al FPGA

Es preciso mencionar que si se tiene alguna duda para llevar a cabo el proceso mencionado anteriormente, es necesario referirse a la guía de laboratorio 1, en donde se explica detalladamente cada uno de los anteriores.

10.5 RESUMEN

En la presente guía se plantea la descripción de un circuito por medio de máquinas de estados comparando dos formas distintas de hacerlo: en la primera se realizan instancias de *flip-flop* tipo JK y la segunda por medio de la sentencia *type*, para lo cual se tomaron en cuenta los siguientes conceptos:

- Maquina Moore
- Maquina de mealy
- Diagrama de estados

10.6 PREGUNTAS DE PRUEBA

- ¿Qué tipo de máquina se implementó en la práctica, Moore o Mealy?
- ¿Qué ventajas o desventajas tiene utilizar la sentencia *type* sobre la implementación por medio de instancias de *flip-flops*?
- Realizar un diagrama de estados de un contador de los números pares de 0 a 10

10.7 EJERCICIO PROPUESTO

Realizar la descripción de un circuito que haga un conteo de 1 hasta 10, el cual presente la opción de contar de uno en uno, de dos en dos, de tres en tres y de cuatro en cuatro.

ANEXO B. PROYECTOS DE LABORATORIO SISTEMAS DIGITALES

LABORATORIO **11**

Proyecto 1: EDIFICIO INTELIGENTE.

Luchar para vivir la vida, para sufrirla y para gozarla...La vida es maravillosa si no se le tiene miedo.

CHARLES CHAPLIN

11.1 INTRODUCCIÓN

La academia debe ir de la mano con el cambio del mundo, pues día a día aparecen necesidades nuevas que requieren una solución la cual puede ser determinada con la aplicación de los sistemas digitales. Así pues, no cabe duda que un profesional capacitado en el área, tiene a la mano una herramienta muy poderosa para dar solución a innumerables necesidades que se presentan a diario alrededor del mundo.

Actualmente, los profesionales más capacitados para servir a la sociedad son aquellos que no solo tienen conocimientos suficientes de un tema, sino que de algún modo han tenido la oportunidad de aplicarlos con el fin de afianzar los mismos.

Ahora bien, es de vital importancia realizar la mayor cantidad de prácticas posibles durante la formación y capacitación profesional, pues es en donde aparecen aspectos de gran importancia que pueden ser determinantes, la aplicación de los sistemas digitales pueden llegar a resolver necesidades y problemas que se presentan a diario alrededor del mundo, pero se deben tener en cuenta muchos aspectos que son determinantes a la hora de realizar cualquier proyecto, y que sólo se aprenden con la práctica.

11.2 OBJETIVOS

- Realizar el proceso necesario para adquirir y procesar una señal analógica en la tarjeta SIE.
- Describir un circuito en el FPGA de la tarjeta SIE, que permita guardar los datos provenientes de un ADC.

- Describir un circuito en VHDL que adquiera los datos provenientes de un teclado PS2.

- Resolver un problema mediante la aplicación de los de sistemas digitales.

11.3 CONTEXTO

El mundo cada vez es más inteligente, puertas del supermercado que se abren cuando se acerca una persona, las luces que se encienden respondiendo a un movimiento, cancelar el costo del parqueadero a una máquina etc. Así pues, es preciso decir que cualquier sistema inteligente, es un campo en el que los sistemas digitales tienen innumerables funciones y participaciones.

Se desea construir un edificio inteligente, el cual tenga gran cantidad de funciones que permitan la comodidad, seguridad y confortabilidad de las personas que lo deseen habitar.

En cuanto al tema de seguridad y prevención de riesgos, se ha pensado en las catástrofes que pasan a diario en el mundo, para lo cual es preciso realizar diversas campañas de prevención, no con el objetivo de evitar un fenómeno natural que puede ocurrir en cualquier momento, sino con la firme intención de saber cómo actuar ante determinada situación, a través de alarmas indicadoras de riesgo.

En cuanto al tema de comodidades, se ha pensado en oficinas o apartamentos inteligentes que mejoren la calidad de vida de sus residentes, para lo cual la compañia constructora esta abierta a recibir cualquier tipo de idea por parte de una firma de ingenieros que les ayude a cumplir con su cometido.

Uno de los requerimientos de la constructora es manejar cualquier tipo de proyecto por medio de un teclado *PS2*.

11.4 Marco Teórico

Para la realización del proyecto se deben tener ciertos conocimientos básicos, los cuales son fundamentales para un entendimiento integral del trabajo que se va a realizar, entre los que se destacan: el funcionamiento del hardware requerido, el estudio de las hojas de datos de los dispositivos a utilizar, los circuitos electrónicos a utilizar, y los dispositivos a seleccionar etc.

11.4.1 Funcionamiento del teclado PS2

El puerto *PS2* es utilizado para realizar la conexión eléctrica del teclado *PS2*, la cual se hace por medio de cuatro hilos, de los cuales dos son de alimentación y dos son de la comunicación y transferencia de datos.

Físicamente el conector *PS2* es tal como se muestra en la figura 11.1.

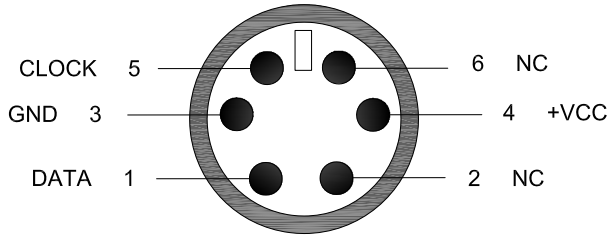


Figura 11.1: Puerto PS2

FUENTE: Los autores

11.4.2 Protocolo de comunicación del teclado PS2

El teclado *PS2* se comunica mediante un *Protocolo Serie Síncrono*, donde la señal *Reloj* indica el momento en el que están disponibles los *bits* de la señal *Datos*, el flanco de bajada del reloj es el que controla el flujo de datos, cabe mencionar que el periodo de la señal del reloj está entre 60μs y 100μs, lo cual quiere decir que la frecuencia del reloj está entre 10 kHz y 16.7 kHz.

Al presionar una tecla, el teclado responde con una trama de 11 *bits*, iniciando con un *bit* de *Start*, seguidamente envía 8 *bits* que corresponden al *make code* ó código de la tecla pulsada, a continuación transmite un *bit* de paridad y finalmente un *bit* de parada. Cabe mencionar que la señal *Dato* es estable por lo menos 5μs antes y después del flanco da bajada del reloj.

En la figura 11.2 se puede observar un esquema que aclara la transmisión de la trama de 11 bits mencionada anteriormente.

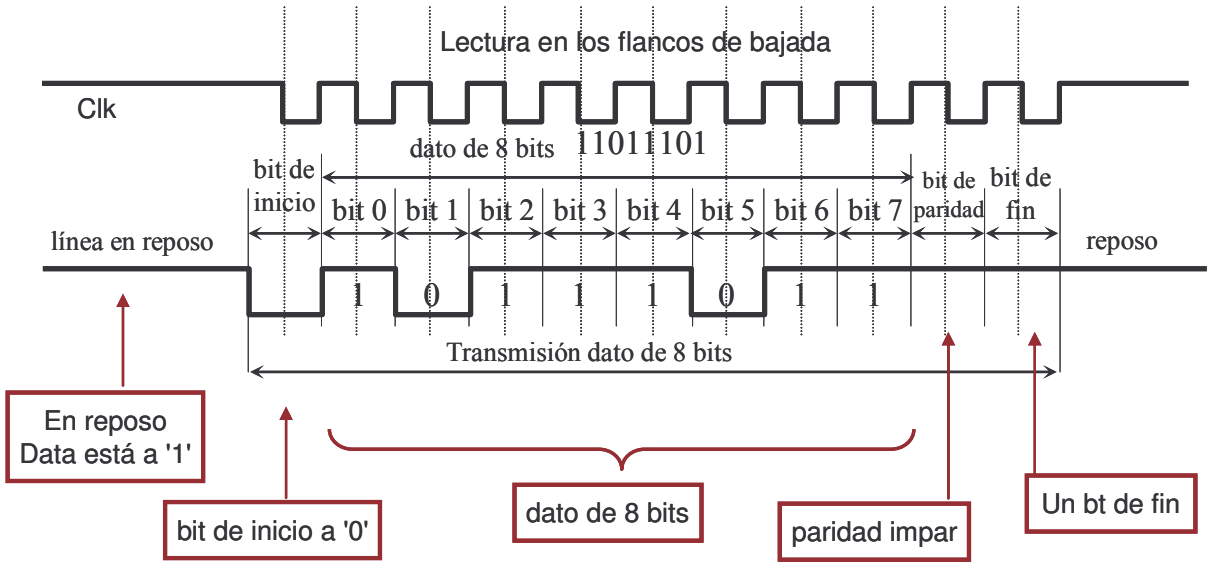


Figura 11.2: Trama de los 11 bits transmitidos

Cada una de las teclas tiene un código que la diferencia de las otras y al cual se le da el nombre de *make code*. Además el teclado transmite un código siempre que se deja de presionar una tecla, a este código se le llama *break code* y es el mismo para todas las teclas (11110000).

Ahora bien, al momento de tener presionada una tecla continuamente, se sigue transmitiendo el *make code* repetidamente. Por defecto el teclado *PS2* transmite el *make code* alrededor de 100 ms despues de que una tecla se mantiene presionada por 0.5 segundos. Si una tecla es liberada ó se deja de presionar, el teclado transmite un *break code*, y seguido el *make code* de la tecla que se dejó de presionar. En la figura 11.3 se puede observar el esquema del teclado caracterizando cada una de sus teclas con un código representado en hexadecimal.

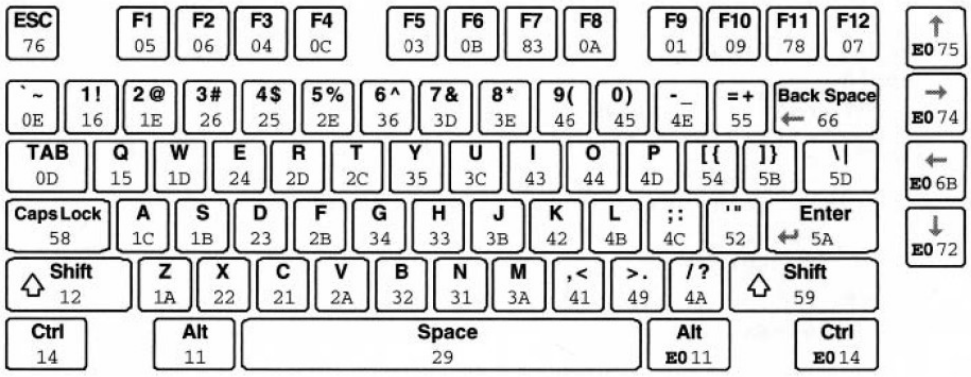


Figura 11.3: Makecode del teclado

Es preciso mencionar que este protocolo es bidireccional, es decir que el teclado admite también comandos enviados desde el PC, pero para llevar a cabo la realización de este proyecto no será necesario su estudio [13] [14].

11.4.3 Adquisición de una señal analógica

En la figura 11.4 se muestra una de las formas de adquirir una señal analógica, para que a su vez pueda ser procesada por el FPGA.

Figura 11.4: Adquisición de señal analógica

FUENTE: Los autores

El proceso muestra inicialente un bloque que hace referencia a una señal analógica, ya sea proveniente de un generador de señales, de un sensor, de algún dispositivo etc.

El bloque de la mitad, hace referencia a un circuito necesario para adecuar la señal analógica, pues esta señal debe ser debidamente acoplada para conectarse a la entrada del ADC según especificaciones de este último. Además muchos sensores necesitan de un circuito para poder trabajar eficientemente.

El tercer bloque es el ADC, el cual tiene como entrada la señal analógica adecuada para ser codificada y transmitir los datos digitales.

11.5 Actividad Propuesta

Para esta ocasión, la constructora requiere proyectos que ayuden a la prevención de catástrofes y accidentes que puedan suceder en el edificio.

Así pues, un terremoto es considerado como una de las catástrofes que pueden llegar a ocurrir en cualquier momento, por lo tanto es importante en cualquier construcción tener una alarma que nos indique los niveles de intensidad de los movimientos de la tierra con el fin de responder a una señal de alarma, con acciones que reduzcan el peligro buscando lugares de resguardo seguro y tomando las acciones recomendadas por las autoridades competentes.

Así pues, se ha abierto una licitación para la realización de un sistema que mida los niveles de intensidad de un temblor que cumpla con los siguientes requisitos:

- Se deben identificar 3 niveles de intensidad
- El nivel 1 hace referencia a los movimientos normales de la tierra.
- El nivel 2 hace referencia a un nivel que sobrepasa los niveles normales de la tierra, y se requiere para que una persona esté preparada para tomar las acciones preventivas correspondientes.
- El nivel 3 hace referencia a un nivel en el que las personas empiecen a tomar acciones de prevención.
- Debe tener un sistema que se active con una clave secreta y que simule los niveles del temblor progresivamente, con el fin de realizar diversos simulacros para realizar campañas de prevención.
- La clave secreta debe ser ingresada mediante un teclado *PS2*.
- Se desea representar cada uno de los niveles del temblor en tres luces.
- En el nivel 1 debe encender una de las luces, en el nivel 2, la luz del medio, y en el nivel 3 del temblor, deben encender y apagar repetidamente las tres luces cada 2 segundos, junto con una alarma sonora.
- Por motivos internos de la compañia, se exige trabajar con un ADC de comunicación serial, preferiblemente el de referencia *TLV0831*.

11.6 Dispositivos a utilizar

Según las necesidades del problema, se debe tener un sensor que responda a vibraciones o movimientos, para así medir el nivel del temblor que active una alarma cuando la situación se ponga crítica. En este punto cabe resaltar que existen muchas soluciones, dependiendo de muchos factores como: el punto de vista del ingeniero, el presupuesto que haya para el proyecto, la facilidad de conseguir materiales y dispositivos que se requieran etc, también deben ser seleccionados los elementos que se requieran para adecuar la señal analógica al ADC, y lo mas importante, se debe seleccionar un ADC que cumpla con especificaciones para conectarse a la tarjeta SIE, pues no todos son ideales para una determinada especificación.

Así pues, los dispositivos y circuitos que se deben seleccionar para este proyecto son:

- Un sensor.
- Circuito de adecuación de la señal analógica.
- Un ADC.

11.6.1 Selección del sensor

Después de realizar una investigación de los sensores que responden a vibraciones, se seleccionó el Sensor de Vibración *MiniSense 100 Vibration Sensor*, el cual es un sensor de polímero piezoeléctrico de vibración, que se usa para detección de golpes, rupturas de cristales etc. En la figura 11.5, se presenta una imagen del sensor utilizado, que presenta como características principales:

Figura 11.5: Sensor de vibración MiniSense

FUENTE: Dynamo Electronics

- Alta sensibilidad (1V/g).
- Resonancia sobre 5 V/g
- Montaje vertical u horizontal.
- Bajo costo

- Linealidad $< 1\%$

Para mas detalles, se puede consultar las características en su respectiva hoja de datos. [15] [16]

11.6.2 Selección del ADC

Para la selección del ADC se tuvo en cuenta principalmente los voltajes de los niveles lógicos de la tarjeta SIE, es decir, la SIE tiene un rango de voltajes para los cuales considera un 0 lógico (de 0 V a 0.8 V), y otro rango de voltajes para los cuales considera un 1 lógico (de 2 V a 3.3 V). Además con la intensión de proporcionar el voltaje de alimentación con la tarjeta SIE, también se tuvo en cuenta que se pudiera alimentar el ADC con aproximadamente 3.3 V.

Así pues, se procede a seleccionar el ADC con las especificaciones que siguen:

- Voltaje de salida de nivel alto desde 2 hasta 3.3
- Voltaje de salida de nivel bajo desde 0 hasta 0.8
- Voltaje de alimentación de 3 V.

El dispositivo seleccionado que cumple con las especificaciones anteriores es el ADC de referencia *TLV0831*, el cual presenta las siguientes características principales:

- 8-Bits de resolución
- Voltaje de alimentación Vcc de 2.7 V hasta 3.6 V
- 1 Canal
- Rango de entrada de 0 V a Vcc
- Voltaje de salida mínimo a nivel alto 2.4 V
- Voltaje de salida máximo a nivel bajo 0.34 V

En la figura 11.6 se describe su funcionamiento principal, mostrando la secuencia de operación. Para mas especificaciones es preciso referirse a la respectiva hoja de datos del dispositivo *TLV0831*. [17]

11.6.3 Selección de dispositivos para adecuar la señal

Para la adecuación de una señal, existen diversas posibilidades, dependiendo de las necesidades y de las aplicaciones, por ejemplo para aplicaciones donde el circuito electrónico no puede ser situado cerca del sensor, se

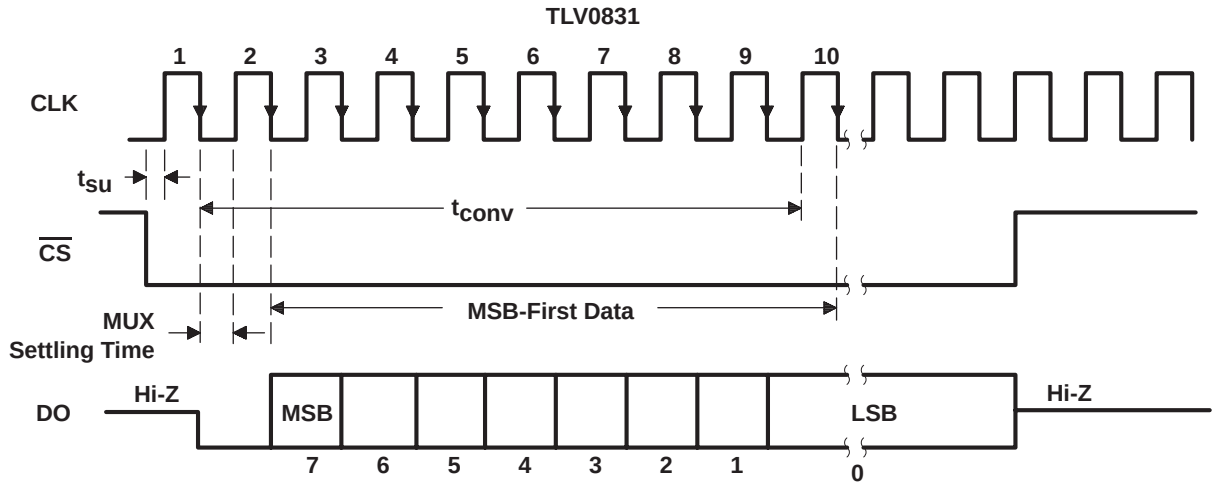


Figura 11.6: Secuencia de operación del ADC

FUENTE: Texas Instruments

recomienda un circuito buffer, cuando se considera la temperatura del ambiente se recomienda otro tipo de configuración y así existen diversas configuraciones dependiendo principalmente del tipo de sensor y de la aplicación requerida.

Para este tipo de sensores que hacen referencia a los sensores piezo film, se recomienda un circuito buffer de ganancia unitaria, el cual se muestra en la figura 11.7. [16]

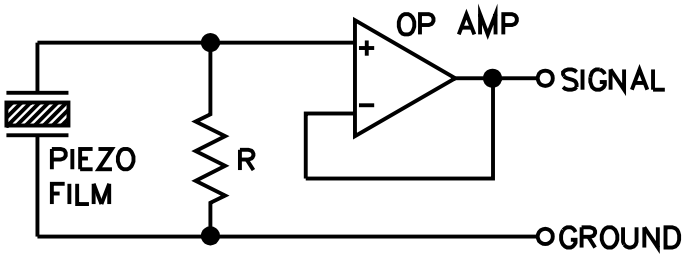


Figura 11.7: Circuito para la adecuación de señal

Para llevar a cabo la implementación del circuito de la figura **??**, se recomienda buscar el amplificador adecuado, para lo cual se tiene en cuenta que funcione con un voltaje de alimentación de aproximadamente 3.3 V con el objetivo de ser alimentado con la misma fuente que el ADC.

El amplificador seleccionado es el *OPA2336*, el cual se utiliza como buffer. Para entrar en detalle con las especificaciones, es preciso referirse a la hoja de datos correspondiente al dispositivo. [17]

11.7 Conexiones de los dispositivos

La figura 11.8, muestra las conexiones que se deben realizar entre los diversos dispositivos, para llevar a cabo la implementación del proyecto.

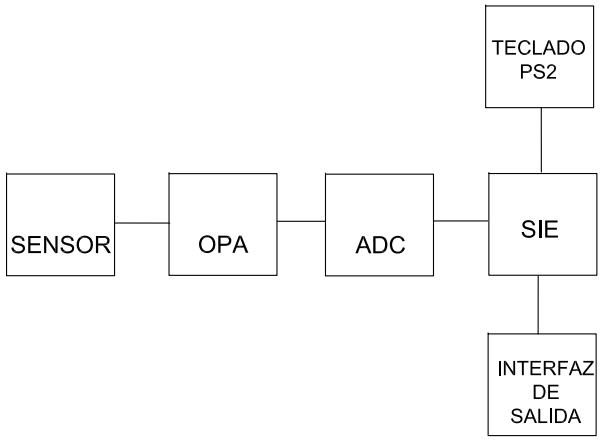


Figura 11.8: Conexiones de todos los dispositivos

FUENTE: Los autores

11.8 Esquemático general del proyecto

El esquema RTL de todo el proyecto se plasma en la figura 11.9, donde se muestran las entradas y salidas del proyecto, las entradas hacen referencia a las dos señales provenientes del teclado *PS2*, a una señal proveniente del *ADC*, y a las señales de reloj y reset que actuan sobre cada uno de los módulos. De igual forma, las salidas del proyecto hacen referencia a una señal de 4 *bits*, de los cuales 3 indican los niveles uno, dos y tres de intensidad del temblor, y el cuarto bit indica la señal sonora de alarma, además existen tres salidas adicionales que indican en que momento se puede ingresar un dato de la clave secreta por medio del teclado *PS2*, otro que indica si la clave marcada es errada, y un último que indica si la clave marcada es correcta.

Así pues, es notorio que el proyecto se puede dividir en tres módulos principales: uno que capture los datos provenientes del teclado *PS2*, otro que tenga una clave guardada y decida si la clave ingresada por el ususario es o no correcta, y un tercer módulo que capture los datos provenientes del *ADC* y responda con la señal de intensidad del temblor.

11.8.1 Esquemático del Teclado PS2

El RTL del circuito que guarda los datos provenientes del teclado *PS2* se muestra en la figura 11.10.

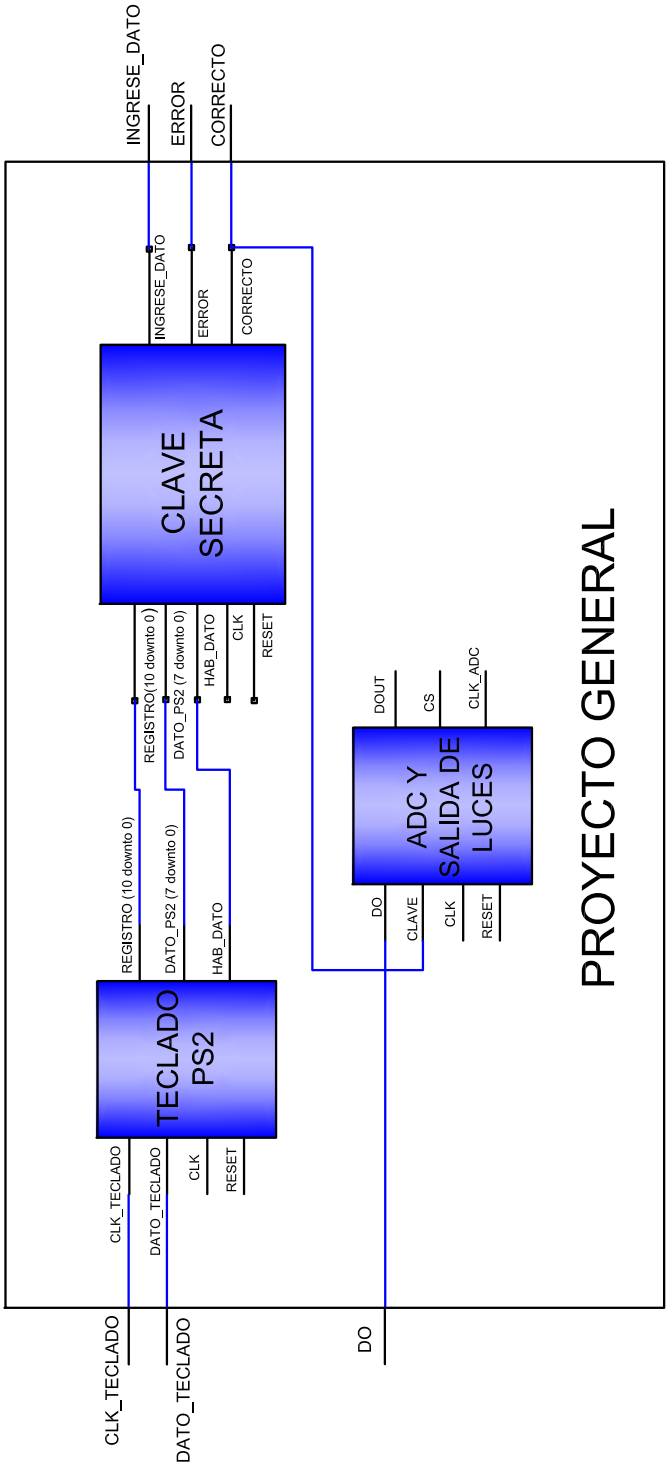


Figura 11.9: Esquema RTL del todo el proyecto

FUENTE: Los autores

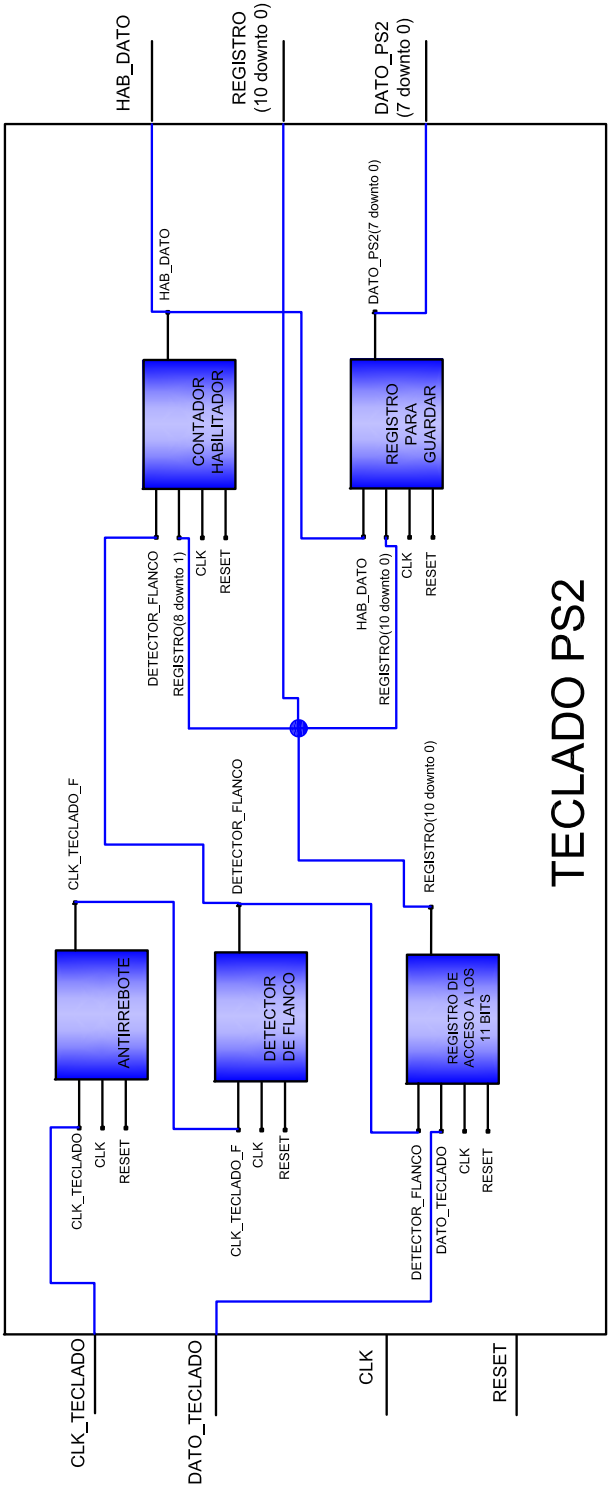


Figura 11.10: Esquema RTL del circuito para el teclado PS2

FUENTE: Los autores

Las entradas de este circuito son las dos señales dedicadas a la comunicación provenientes del teclado *PS2*, el reset y el *clk* de sincronismo de todos los circuitos, y las salidas son el dato proveniente del teclado guardado.

11.8.1.1 Circuito detector de flanco de bajada

El circuito detector de flanco tiene como entrada la señal de un reloj y como salida, un pulso que se activa al detectar el flanco de bajada del reloj.

11.8.1.2 Circuito Antirrebote

El circuito antirrebote tiene como entrada una señal que tarda unos instantes en estabilizarse y tiene como salida, la misma señal de entrada pero sin los rebotes, es decir sirve para filtrar una señal.

11.8.1.3 Registro para tener acceso a los datos

Este módulo se encarga de recibir los datos provenientes del teclado a través de un registro, el cual presenta como entrada la señal *Dato* proveniente del teclado y una señal que detecta el flanco de bajada del reloj proveniente del teclado *PS2*, de igual forma presenta como salida, una señal de 11 *bits* que va registrando los datos provenientes del teclado.

11.8.1.4 Contador para habilitar un dato

Este módulo se revfiere a un circuito que inicia un conteo tras una señal, y al finalizar el conteo, pone en nivel alto una determinada señal, con el objetivo de indicar la finalización del conteo, el circuito presenta en su entrada, la señal proveniente del registro y la señal que detecta el flanco de bajada del reloj del teclado *PS2*, de igual forma presenta a su salida una señal que sólo se activa cada vez que finaliza el conteo.

11.8.1.5 Registro para guardar el dato

Este circuito es el encargado de capturar y guardar el dato que se encuentra en el registro que permite tener acceso al dato proveniente del teclado, este módulo presenta a su entrada la señal que indica que el contador ha pasado un ciclo y la señal proveniente del registro que permite tener acceso a los datos del teclado. Así mismo a su salida presenta una señal que contiene el dato capturado y guardado.

11.8.2 Esquemático de la clave

En este módulo se describe el circuito que controla el ingreso de datos por parte del usuario mediante el teclado *PS2*, además debe tener una clave secreta que a petición del usuario sea marcada y el circuito indique si la clave es o no correcta.

En la figura 11.11 se presentan las señales de entrada y de salida del correspondiente módulo.

Figura 11.11: Módulo de la clave secreta

FUENTE: Los autores

11.8.3 Esquemático del ADC

En la figura 11.12 se presenta el esquema RTL del circuito que tras recibir los datos provenientes del *ADC*, indica cual es el ivel de intensidad de un temblor.

11.8.3.1 Reloj que impone la frecuencia de trabajo del ADC

El ADC necesita un reloj con cierta frecuencia para funcionar, luego este circuito impone dicha frecuencia dependiendo de las especificaciones del ADC, el módulo presenta como entrada la señal *clk* de la SIE y como salida una señal de reloj con una frecuencia deseada, que para el caso se recomienda de 250 kHz.

11.8.3.2 Circuito detector de flanco de bajada

El presente módulo sirve para detectar el flanco de bajada de la señal de reloj que se impone al ADC para que trabaje a condiciones normales óptimas.

11.8.3.3 Gernerador del CS

Este circuito genera la señal *CS*, que es requerida por el *ADC*, la cual debe ir conectada directamente al mismo.

11.8.3.4 Registro para guardar los datos del ADC

En este módulo es necesario hacer un registro con el fin de tener acceso a los datos provenientes directamente del ADC.

11.8.3.5 Registro para capturar el código de una tecla

El presente circuito es el encargado de capturar y guardar el dato proveniente del registro que permite tener acceso a los datos provenientes del *ADC*, tras una señal de aviso, es decir que se le debe indicar en que momento capture una señal.

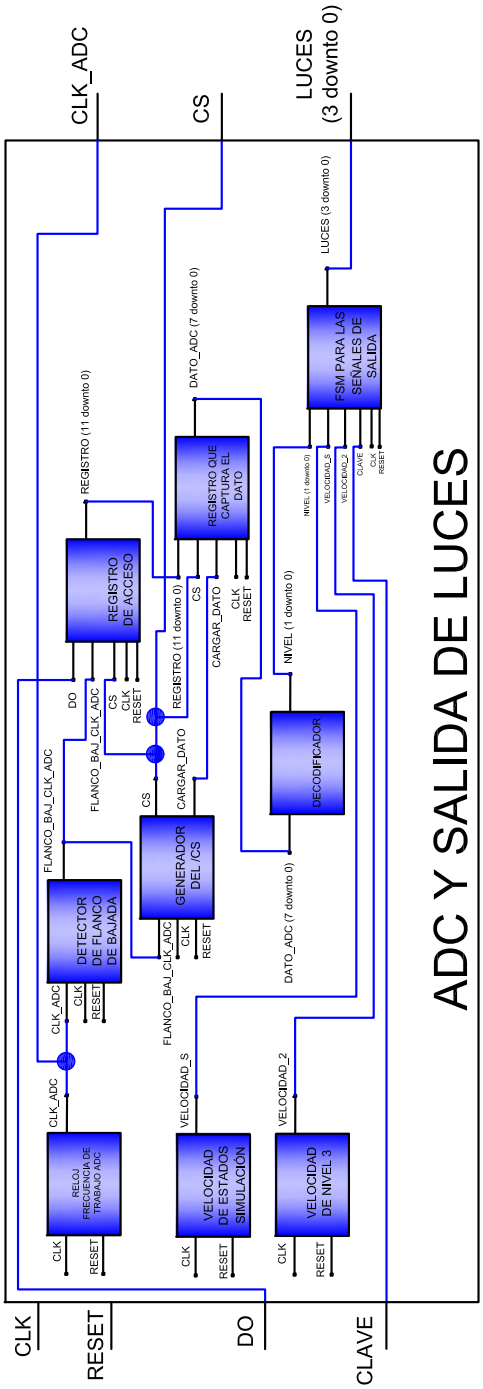


Figura 11.12: ADC y salida de luces

FUENTE: Los autores

11.8.3.6 Decodificador para los niveles de intensidad

Este módulo es utilizado para determinar el rango de valores que son asignados a los diferentes datos provenientes del ADC, que son estratificados en tres niveles distintos.

11.8.3.7 Contador para controlar los cambios de estado de la simulación

El presente módulo describe el circuito que regula los tiempos de cambio de estado en una FSM, es decir que simplemente hace referencia a una señal que se active cada cierta cantidad de tiempo si es requerida en cualquier otro módulo.

11.8.3.8 Reloj para controlar el encendido y apagado de los leds en el nivel mas alto

Al igual que el circuito anterior, el presente módulo regula el tiempo de un cambio de estado, que para el caso hace referencia al encendido y apagado de los leds en el nivel mas alto del temblor como lo presentaban los requisitos.

11.8.3.9 Circuito que describe las salidas dependiendo del nivel del temblor

Este módulo se utiliza para describir la lógica de las señales de salida dependiendo el nivel de intensidad del temblor, es decir es el que indica cuando debe estar o no activa una señal de salida.

11.9 Mejoras y contribución al proyecto

Este proyecto es una iniciativa en la construcción de innumerables sistemas que pueden ser adaptados al proyecto base del edificio inteligente, al cual pueden realizarse mejoras y en el cual puede ser adaptada cualquier idea que contribuya con el nivel de inteligencia del mismo.

Dentro de las mejoras y nuevas ideas a trabajar se proponen:

- Interfaz para programar una nueva clave secreta
- Alimentación independiente para la tarjeta SIE
- Controlar el flujo de personas que entran y salen del edificio
- Abrir puerta principal tras detectar que se acerca una persona
- Escaleras eléctricas
- Encender la luz del parqueadero al detectar oscuridad
- Ascensores del edificio
- Alarma contra incendios

- Alarma contra inundaciones
- Alarma contra robo
- Boveda de seguridad con clave secreta
- Encendido de luces tras un sonido particular
- Abrir la puerta del parqueadero tras pulsar una clave
- Tener acceso al edificio con una tarjeta personal
- Sistema de seguridad que ante la activación de una alarma llame al número respectivo de emergencia según sea la necesidad
- Pantalla LCD para múltiples funciones

LABORATORIO **12**

Proyecto 2: AUTOMOVIL.

Nunca vayas por el camino trazado, porque conduce hacia donde otros han ido ya.

ALEXANDRE GRAHAM BELL

12.1 INTRODUCCIÓN

Los proyectos finales buscan aplicar todos los conocimientos que han sido adquiridos, en estos proyectos se plantea un problema muy particular, el cual tras su análisis se reduce a pequeños módulos que se estudiaron a través del curso. Estos proyectos a su vez, intenta mostrar que los sistemas digitales están muy ligados a la parte de *hardware*, porque es la que le permite interactuar con su entorno, para así poder satisfacer alguna necesidad planteada.

Los proyectos finales son muy importantes porque el estudiante puede experimentar, que al tratar de modificar variables existe un grado de dificultad mayor ya que en un medio existen muchos factores que nos se tienen en cuenta cuando no se crean modelos a pequeña escala por ejemplo fuerzas de rozamientos, inercias de las masas, constantes de tiempos, y muchos tipos de fenómenos que aparecen al tratar de dar solución a un problema. Por lo tanto los estudiantes al finalizar este tipo de proyectos debe estar en la capacidad de diseñar una PCB, leer algunos parámetros importantes en una hoja de datos y aún mas importante realizar un modelo tangible de una solución al problema planteado.

12.2 OBJETIVOS

- Expandir el campo de aplicación de la SIE manejando circuitos de mayor potencia
- Mostrar los conocimientos adquiridos a lo largo del curso.
- Aplicar la información mostrada por las hojas de datos.

12.3 CONTEXTO

A lo largo de el tiempo la humanidad ha desarrollado objetos, aparatos, maquinas, cada vez más complejas, estos grandes desarrollos se han dado por la necesidad de solucionar problemas en una sociedad. Como bien se conoce, las instituciones educativas, más que todo las universidades han sido concebidas para dar soluciones a problemas generados por la comunidad, cuando la universidad quiere dar una solución a un problema, primero debe acudir a la comunidad para tomar las especificaciones y así cumplir con los requerimientos para el mismo, lo siguiente es seccionar el problema general en partes especificas y finalmente se crean grupos de trabajo que resuelven detalles específicos del problema general.

Un grupo calificado ha dividido un problema de un automóvil, en el cual se le a asignado la tarea a los estudiantes de sistemas digitales de realizar el movimiento de un automóvil hacia adelante y hacia atrás, también se ha mencionado que dicho automóvil pueda variar la velocidad y además que pueda producir el sonido de una bocina.

Algunos estudiantes se reunieron por aparte y realizaron una proposición entre ellas estaban:

- Teclado

  Como el carro no va a tener muchas funciones, no es muy practico utilizar un teclado PS2, es mejor utilizar un teclado matricial.

- Control de velocidad

  Controlar la velocidad del motor es posible por medio de PWM.

- movimiento hacia adelante y atrás

  Los giros del motor se pueden manejar con un circuito puente H.

- Manejo de corriente

  Después de detallar un momento el problema se dieron cuenta que los motores consumen mucha más corriente de las que pueden manejar los circuitos de control prestados en la universidad y deciden aislar estos circuitos con optoacopladores.

Cada grupo puede decidir como resolver el problema, el apoyo que le brinda la universidad es el préstamo de la tarjeta SIE junto con un laboratorio en horarios disponibles y como conocimiento la asignatura por la que cruzan, aparte de la ayuda de los profesores.

12.4 Marco Teórico

Para tener más claridad sobre la información que los estudiantes nos han mencionado se presenta una pequeña explicación.

12.4.1 Teclado Matricial

Un teclado matricial es un simple arreglo de botones conectados en filas y columnas, de modo que se pueden leer varios botones con el mínimo número de entradas requeridas. Un teclado matricial 4x4 solamente ocupa 4 líneas de un puerto para las filas y otras 4 líneas para las columnas, de este modo se pueden leer 16 teclas utilizando solamente 8 entradas (En un teclado no matricial cada tecla necesita una línea de entrada, con lo cual representa una cantidad mayor de líneas de I/O de un FPGA).

En la figura 12.1 se presenta la organización estándar de un teclado matricial

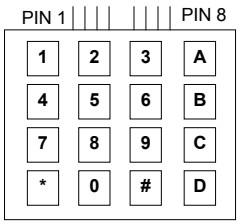


Figura 12.1: Representación del teclado matricial

Para controlar el teclado, los puertos del FPGA correspondientes a las filas se describen como salidas y las columnas se definen como entradas. De tal forma que el objetivo principal del circuito para decodificar el teclado consiste en determinar la fila y columna que corresponde a la tecla que se presionó.

Lo anterior se logra rotando un valor lógico (ya sea 1 ó 0) en cada una de las líneas configuradas como salidas (filas en este caso) e inmediatamente después leer el setado lógico de las líneas conectadas como entradas (columnas).

Cuando el valor lógico que rota es 1 al algoritmo se le denomina **walking ones** y **walking zeros** cuando se trata de un 0. Este dispositivo, contiene varias teclas, las cuales sirven para introducir datos, por ejemplo a un microcontrolador, que realiza la función de calculadora, marcador telefónico, cerradura electrónica y una infinidad de aplicaciones.

12.4.2 PWM

PWM es el acrónimo de "Pulse Width Modulation", expresión que designa un modo concreto de modulación, la llamada "Modulación de pulsos en anchura". Inicialmente, esta técnica se utilizaba casi exclusivamente para el control de potencia y velocidad de motores de corriente continua, pero con el tiempo se ha ido ampliando el campo de aplicación, por ejemplo, en las fuentes de alimentación conmutadas, onduladores c.c.-c.a., etc.; con ella se ha conseguido realizar dispositivos mucho más eficientes, más compactos y más ligeros.

12.4.2.1 ¿Qué es la modulación PWM?

Antes de la puesta a punto de la técnica PWM, la velocidad de un motor de corriente continua se regulaba mediante un potenciómetro o reóstato en serie. Este "poco fino" método comportaba un gran gasto de energía, con mucha generación de calor. Cuanta más potencia, mayor desperdicio de energía. Se opto por aplicar toda la potencia disponible, pero no continuadamente sino en forma de impulsos más o menos breves.

Puede hacerse un sencillo experimento con una pila y una bobillo de filamento, como las de las linternas de bolsillo. Soldar un polo de la pila a la bombillo con un tramo de hilo conductor y cerrar el circuito aproximando a mano el otro polo, a través de otro tramo de hilo. Al hacer un contacto intermitente, juntando y separando rápidamente el terminal, se observará que, gracias a la inercia de la resistencia y a la persistencia del ojo humano, la bombillo lucirá de forma casi estable, aunque con una intensidad luminosa inferior. Se está aplicando íntegramente la tensión de la pila pero sólo a intervalos. Dos parámetros tienen suma importancia: el tiempo que la bombillo permanece alimentada (Ton) y el tiempo entre dos impulsos de tensión (Toff). Dentro de un cierto periodo de tiempo, cuanto menor sea la suma de tiempos en que la bombillo queda sin alimentar, mayor es la intensidad luminosa emitida. De aquí sale la definición de "ciclo de trabajo" (duty cycle): es el porcentaje que indica el tiempo durante el cual la señal permanece a nivel alto, durante un periodo dado. El ejemplo típico de un ciclo de trabajo del 50 % es una señal perfectamente cuadrada, como las que entregan los generadores de señal. Una tensión de cero voltios corresponde a un ciclo de trabajo del 0 %. En la figura 12.2,

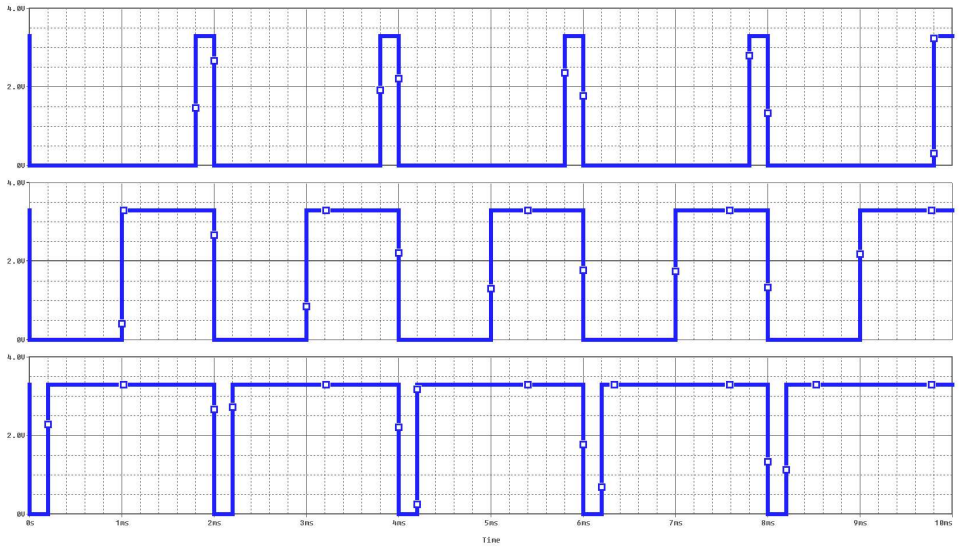


Figura 12.2: variación de ancho de pulsos por medio de PWM

FUENTE: Herramienta de simulación OrCAD

Se presenta una señal Ttot=2ms, esta señal se le presentan variaciones del ciclo de trabajo de 10 %, 50 %, 90 % respectivamente, en donde en Ton representa el periodo durante el cual la bombillo recibe la tensión máxima (3.3

V) y Toff es el tiempo en que no recibe tensión alguna (0 V). Modificando la duración relativa de estos tiempos se consigue modificar la tensión media que llega a los contactos de la lámpara y, por consiguiente, se regula su luminosidad. En la práctica, se actúa sobre Ton y se mantiene constante la duración del ciclo (Ttot=Ton + Toff). O sea que sólo se modifica el porcentaje relativo de Ton y Toff respecto al diclo total. Así, la señal PWM puede definirse como una señal de tensión rectangular en la cual se ha previsto una determinada distribución temporal entre un pulso alto y un pulso bajo. Admitiendo que Ttot no varía, al alargar la duración de Ton, la de Toff se acorta necesariamente. Aplicando una PWM a la alimentación de un motor, si Ton representa el 10 % del ciclo, la tensión apare ce en bornes del motor a intervalos muy cortos y el ciclo de trabajo es bajo. Pero si Ton representa el 90 % del ciclo, el motor recibe tensión a intervalos proporcionalmente muy largos y el ciclo de trabajo es elevado. [26]

12.4.3 Optoacopladores

Los optoacopladores son también conocidos como optoaisladores o dispositivos de acoplamiento óptico, basan su funcionamiento en el empleo de un haz de radiación luminosa para pasar señales de un circuito a otro sin conexión eléctrica. Estos son muy útiles cuando se requiere proteger un circuito, por ejemplo cuando se trabajan circuitos de muy alta potencia que la tarjeta SIE no logra suplir pero necesita manejar, el optoacoplador permite hacer una conexión entre la SIE y el circuito de alta potencia, evitando daños en la tarjeta tras alguna falla o corto en el circuito de alta potencia.

En la figura 12.3 se presenta una representación de los pines de un optoacoplador

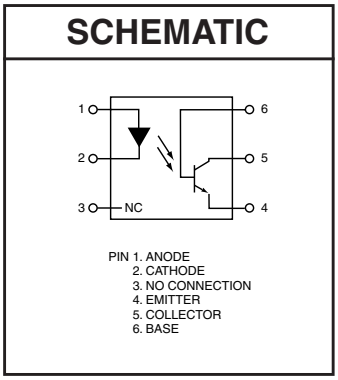


Figura 12.3: Esquema de un optoaclopador [28]

FUENTE: Fairchild

La gran ventaja de un optoacoplador reside en el aislamiento eléctrico que puede establecerse entre los circuitos de entrada y salida. Fundamentalmente este dispositivo está formado por una fuente emisora de luz, y un fotosensor de silicio, que se adapta a la sensibilidad espectral del emisor luminoso, todos estos elementos se encuentran dentro de un encapsulado que por lo general es del tipo DIP.[29]

12.5 Actividad Propuesta

Realizando un resumen sobre la situación planteada, lo que se requiere es un automóvil que pueda variar su rapidez, éste vehículo se debe desplazar con cinco velocidades diferentes hacia adelante y una velocidad hacia atrás, como parte adicional se requiere que al presionar una tecla el automóvil pite, la entrada a los movimientos del carro debe ser matricial ya que existen otras funciones para el carro que no se les ha asignado y el teclado matricial permite adicionar otro tipo de entradas.

Lo que se debe entregar en el proyecto es:

- Un modelo del automóvil en funcionamiento con las condiciones descritas anteriormente
- Todos los circuitos descritos en VHDL necesarios para llegar a cabo el funcionamiento del carro
- Parámetros de selección de los elementos que utilizó

12.6 Dispositivos a utilizar

Según las necesidades del problema, se debe tener ciertos elementos que permitan ayudar con la tarea. Así pues, los dispositivos y circuitos que se deben seleccionar para este proyecto son:

- Dos optoacopladores
- Tarjeta de expansión para la SIE
- dos motores

12.6.1 Selección de los optoacopladores

Después de revisar los optoacopladores se nota que un parámetro de importancia en ellos es el tiempo de respuesta, pues ya que lo deseado es realizar una conmutación.

- Costo del optoacoplador
- Tiempo que se demora la llega del producto del extranjero
- Tiempo de respuesta de optoacoplador
- Rango de entrada y salida
- Corriente máxima que soporta el dispositivo

Para más detalles, se puede consultar las características en su respectiva hoja de datos.
http://doc.chipfind.ru/fairchild/4n25.htm

12.6.2 Señal de control de velocidad

Para el control de velocidad de motor se recomienda el uso de PWM, cabe recordar que se debe tener en cuenta la frecuencia a la que se desea trabajar con PWM, los parámetros a tener en cuenta son:

- Especificaciones de la SIE
- Frecuencia del PWM
- Mínimo ciclo de trabajo para que exista movimiento en el vehículo

12.6.3 Selección de los Motores

A continuación se procede ha realizar los parámetros de los motores, se debe tener en cuenta que estos puedan desplazar el peso de ellos y del modelo del vehículo, además se deben tener en cuenta los siguientes parámetros:

- Costo del los motores
- Tiempo que se demora la llega del producto del extranjero
- Torque de los motores
- Rango de entrada
- Corriente máxima que soporta el dispositivo

Para más detalles, se puede consultar las características de los motores mabuchi
http://www.kysanelectronics.com/Products/Detail.php?recordID=1539 http://www.wellgainelectronics.com/mabuchieg-530yd-2bh12vdcmotor.aspx http://audiolabga.com/mal_cart/mal_cart.php?nte=MOTOR

12.6.4 Cuidados necesarios para la tarjeta SIE

Como ya es conocido se desea trabajar con la FPGA para esto es necesario conocer sus limitaciones. Se debe analizar detalladamente en la hoja de datos y se puede encontrar en:
http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf
Las características eléctricas se encuentran a partir de la pagina 117.

12.7 Esquemático general del proyecto

Como ayuda para los estudiantes se les presenta un diagrama RTL en la figura 12.4, de una posible solución del problema la cual en su parte interna esta dividida por tres bloques:

En éste diagrama se puede observar que existen tres módulos mencionados a continuación:

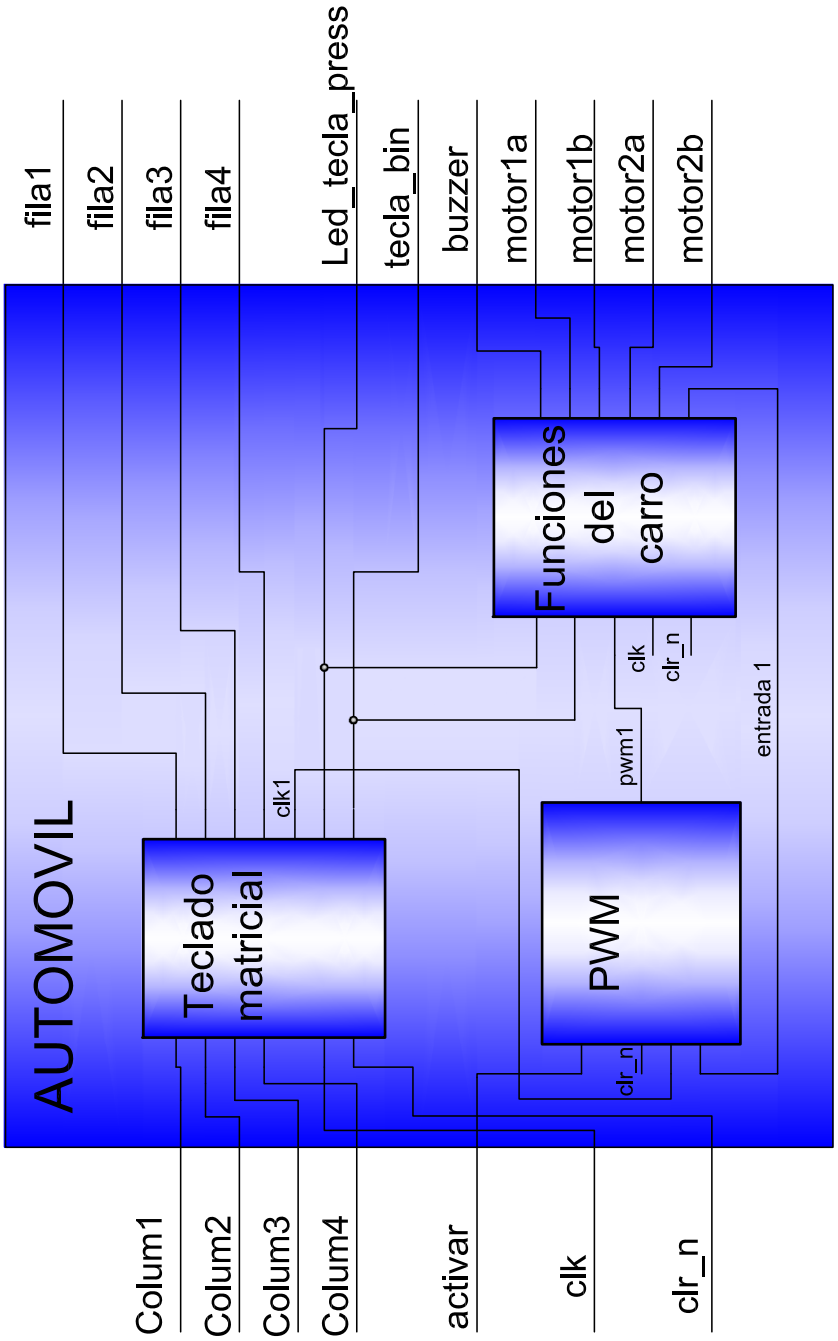


Figura 12.4: Esquema RTL de proyecto del automóvil

FUENTE: Los autores

- Módulo del teclado matricial
- Módulo del PWM
- Modulo de las funciones del carro.

Para su mayor comprensión estos módulos se ha dividido en pequeñas partes.

12.7.1 Esquemático del Teclado matricial

para el módulo del teclado matricial puede dividir en bloques que se han estudiado a lo largo del curso como lo muestra la figura 12.5

A continuación se pretende dar una breve idea de lo que debe hacer cada módulo de este diagrama presentado anteriormente.

12.7.1.1 Registro columna

Registro para almacenar las entradas de las columnas, el registro cuenta con 4 bits de almacenamiento.

12.7.1.2 Detector de tecla presionada

Es un circuito que detecta si se ha presionado una tecla, como entradas puede tener filas o columnas dependiendo del circuito y como salida un bit que diferencie la presión de una tecla.

12.7.1.3 Concatenado

Es un circuito que almacena las filas y las columnas para su posterior decodificación.

12.7.1.4 Decodificador binario

Es un registro que guarda los datos codificados del módulo decodificador binario.

12.7.1.5 Contador de anillo

Es el circuito que se encarga de escanear las filas o las columnas.

12.7.1.6 Divisor de frecuencia

Es un circuito que disminuye la frecuencia dando el tiempo necesario para que la señal emitida por los pulsadores sea estable.

12.7.2 Esquemático del PWM

En la figura 12.6 se presenta el diagrama RTL de el módulo PWM. se detalla sobre sus módulos internos.

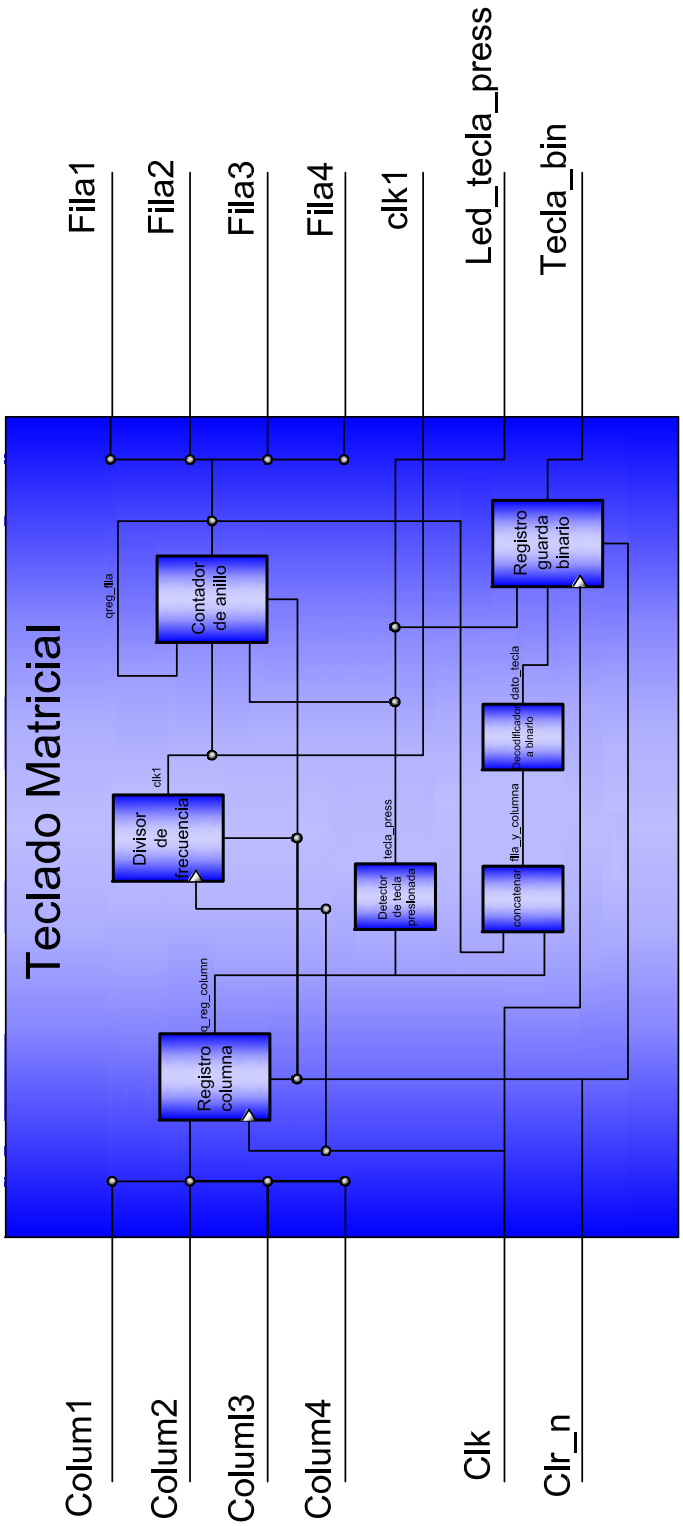


Figura 12.5: Digarama RTL del teclado matricial

FUENTE: Los autores

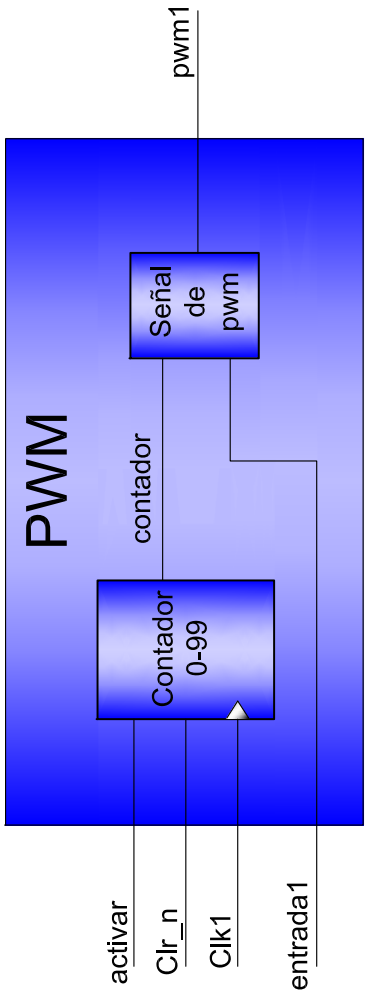


Figura 12.6: Digarama RTL módulo PWM

FUENTE: Los autores

12.7.2.1 Contador módulo 100

Es un contador que realiza una cuenta hasta 100 para determinar el ciclo de trabajo del PWM.

12.7.2.2 Señal PWM

Este módulo presenta la descripción de un circuito, que genera la señal PWM y dependiendo de la señal entrada1 determina el ciclo de trabajo.

12.7.3 Esquemático de funciones del carro

Sobre este esquemático no se brinda información, se deja abierto a las soluciones dadas por los estudiantes.

12.8 Mejoras y contribución al proyecto

- Giro de las llantas delanteras para dar la dirección del auto
- Se deben instalar las luces que debe tener todo auto, luces altas, bajas, traseras, delanteras etc.
- Agregar una LCD que indique el cambio del carro.
- Alarma antirrobo con clave programable
- Sistema de seguridad que tras una señal se empiece a desocupar el tanque de gasolina del auto poco a poco para que los ladrones no lleguen muy lejos.
- Un indicador de puerta abierta
- Registro continuo de la velocidad del motor
- Grabar un recorrido en el carro para que lo realice automaticamente tras una orden
- Guardar el recorrido que realice durante unos instantes de tiempo.

Bibliografía

[1] Carlos Iván Camargo. Sie: Plataforma hardware copyleft para la enseñanza de sistemas digitales.

[2] Marcos Sanchez-Elez. Introduccion a la programacion en vhdl. http://www.dacya.ucm.es/marcos/intvhdl.pdf.

[3] Display de 7 segmentos. http://www.unicrom.com/Tut_display-7-segmentos.asp.

[4] Thomas L. Floyd. *Digital Fundamentals (7th Edicion)*. Prentice-Hall, Inc, Madrid, 2000.

[5] Ronald J. Tocci. *Sistemas Digitales principios y aplicaciones (6th Edicion)*. Pearson Educación, México, 2003.

[6] Mano M. Morris. *Arquitectura de computadoras (3th Edicion)*. Prentice-Hall, Inc, Madrid, 2000.

[7] David G. Maxinez , Jessica Alcalá Jara. *El arte de programar sistemas digitales (1ra Edicion)*. Compañia editorial continental, Inc, México, 2002.

[8] Digilent. http://www.digilentinc.com.

[9] Concurrencia. http://www.cannic.uab.es/docencia/DSD/Apunts/SintesisConcurrent.pdf.

[10] S.Fernández S.A. Pérez, E. Soto. Diseño de sistemas digitales con vhdl. http://www.dte.uvigo.es/vhdl/c5.html.

[11] Departamento de automática. Apuntes vhdl 00. http://atc2.aut.uah.es/~rico/docencia/asignaturas/informatica/lab_org_comp/archivos/Documentacion/VHDL/Apuntes%20VHDL%2000.pdf.

[12] Alberto Mora Javier García de Jalón, Iker Aguinaga. Aprenda linux como si estuviera en primero. http://www.digilentinc.com, 2000.

[13] Pong P. Chu. *Fpga Prototyping By Vhdl Examples,xilinx Spartan -3 Version*. Editorial:John Wiley Sons Inc.

[14] Dto. de Tecnología Electrónica. U. Rey Juan Carlos Puerto PS/2 http://laimbio08.escet.urjc.es/assets/files/docencia/DCSE/dcse_p8_ps2.pdf.

[15] DYNAMO electronics Puerto PS/2 http://www.dynamoelectronics.com/.

[16] measurement SPECIALTIES TM Interfacing Piezo Film to Electronics http://www.meas-spec.com/.

[17] TEXAS INSTRUMENTS 8-Bit 49 kSPS ADC Serial Out, Differential input, Configurable as SE input, 1 Ch. Single-Supply, MicroPower CMOS Operational Amplifiers MicroAmplifier Series http://www.ti.com/sitesearch/docs/universalsearch.tsp?searchTerm=TLV0831&linkId=1.

[18] USB-DRIVERS full suport for parallel ports on ISE 12 (2010-05-24) http://git.zerfleddert.de/cgi-bin/gitweb.cgi/usb-driver.

[19] XILINX ALL PROGRAMMABLE INSTALL-DRIVERS https://secure.xilinx.com/webreg/clickthrough.do?cid=103670.

[20] THE GEEK STUFF Execution sequence for .bash_profile, .bashrc, .bash_login, .profile and .bash_logout http://www.thegeekstuff.com/2008/10/execution-sequence-for-bash_profile-bashrc-bash_login-profile-and-bash_logout/.

[21] Creative Commons. *Licencias Creative Commons*. http://creativecommons.org/licenses., 2004.

[22] Camargo, Carlos Ivan. *SIE: Plataforma Hardware copyleft para la ensenanza de sistemas digitales*. Universidad Nacional de Colombia. 2010. 6 p.

[23] Qi Hardware http://en.qi-hardware.com/wiki/SIE.

[24] W.Spraul, C. Camargo, and A. Wrang. *Proyecto SACK*. http://en.qi-hardware.com/wiki/SAKC.

[25] Alberto Mora Javier García de Jalón, Iker Aguinaga. Aprenda linux como si estuviera en primero. http://www.digilentinc.com, 2000.

[26] In Electronics. Generación de señales PWM con el microcontrolador PIC16F84 http://usuarios.multimania.es/ccencho/2005Abril.pdf, 2004.

[27] MIGUEL GRASSI PuenteH.pdf. http://www.miguelgrassi.com.ar/mecatronica/PuenteH.pdf, 2006.

[28] FAIRCHILD SEMICONDUCTOR GENERAL PURPOSE 6-PIN PHOTOTRANSISTOR OPTOCOUPLERS. http://www.mbari.org/mars/images/4N37.pdf,

[29] Proyecto Electronicos http://proyectoselectronics.blogspot.com/2008/09/optoacoplador-que-es-y-como-funcionan.html,

[30] Heriberto Vargas Radillo CONTADOR ANILLO Y JOHNSON http://proton.ucting.udg.mx/dpto/maestros/hvargas/sd09/SD09.html,

ANEXO C. MANUAL DE USUARIO DE LA TARJETA SIE LABORATORIO SISTEMAS DIGITALES

APÉNDICE **A**

Manuales de la tarjeta SIE.

A.1 Manipulación de la tarjeta SIE

El proyecto SIE [24] fué creado para satisfacer las necesidades de los desarrolladores de hardware permitiendo la creación de aplicaciones comerciales bajo la licencia Creative Commons BY -SA [21] la que permite la distribución y modificación del diseño (incluso para aplicaciones comerciales), con el único requisito de que los productos derivados deben tener la misma licencia y deben dar crédito al autor del trabajo original. [22]

Así pues, la plataforma SIE es concebida por el profesor Carlos Camargo de la Universidad Nacional de Colombia y es una adaptación de Ben NanoNote a la cual se le ha añadido un FPGA y puertos de entrada salida, igualmente se ha extraído un teclado el cual libera un gran número de pines que pueden ser utilizados en muchas más aplicaciones. [23]

En la figura A.1 se nota la plataforma SIE con los diversos componentes que la conforman, dichos componentes se mencionan a continuación: [23]

- Un procesador Ingenic JZ4725.
- 64MB de memoria SDRAM.
- Una memoria NAND de 2GB.
- FPGA XC3S100E_VQ100 que proporciona 25 puertos de entrada salida de propósito general con señales digitales (rango 0-3.3V).
- El Puerto USB puede ser usado como Ethernet, o dispositivo de consola serial.
- Puerto Micro SD.
- Líneas de entrada / salida de audio Estéreo.
- Señal de entrada de micrófono.

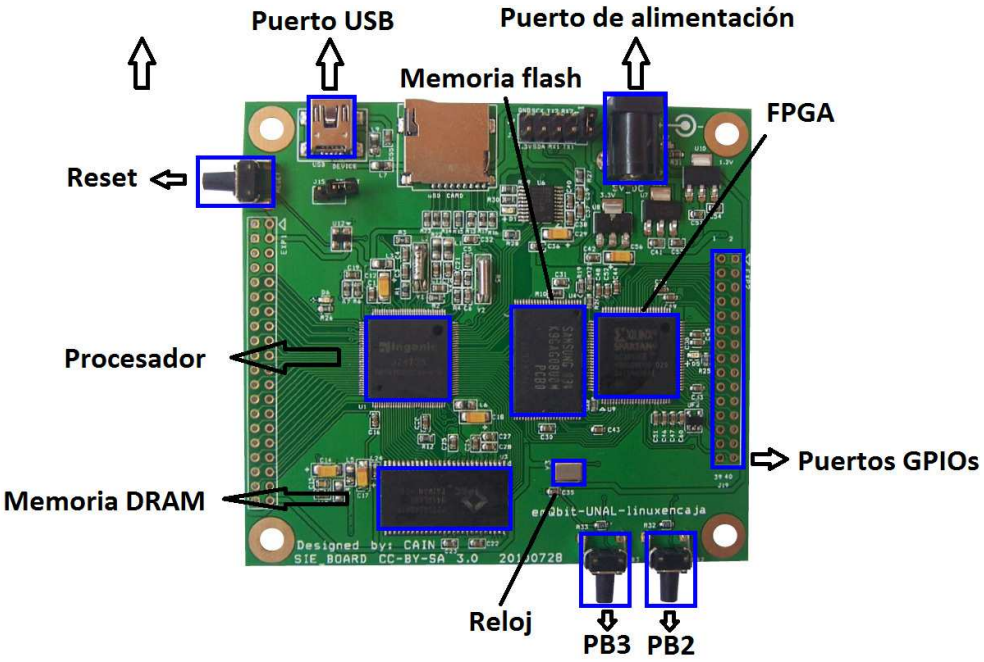


Figura A.1: Plataforma SIE

FUENTE: Los autores

- Puerto I2C.
- Puerto RS-232 Serial UART.

Así mismo, en la figura A.2 se muestra el respectivo diagrama de bloques que caracteriza la plataforma SIE, notando en el mismo los componentes mencionados anteriormente, resaltando el FPGA y los periféricos externos debido a que son los principales componentes que se manejan en la asignatura de sistemas digitales.

Los componentes que se utilizan en la asignatura de sistemas digitales son los siguientes:

- FPGA
- Memoria flash
- Memoria DRAM
- Conexión de la SIE

A.1.1 FPGA

Una FPGA (*Field Programmable Gate Array*) es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada mediante un lenguaje de descripción especializado. La lógica

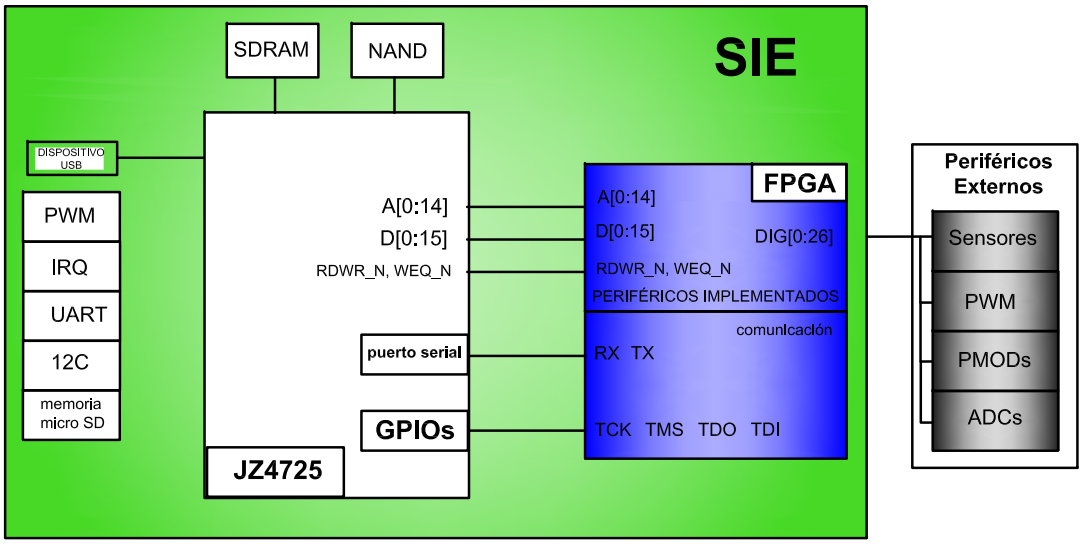


Figura A.2: Diagrama de bloques de la SIE.

FUENTE: Los autores

programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip. Los bloques que existen en la FPGA *Spartan-3E* son los siguientes 5:

- Bloques lógicos configurables (*CLBs*) : contiene *Look-Up Tables* (LUTs) que pueden implementar lógica además de elementos de almacenamientos tales como *flip-flops* o *laches*. Los CLBs realizan gran variedad de funciones lógicas, así como el almacenamiento de datos.
- Bloques de entrada/salida (*IOBs*) : éstos controlan el flujo de datos entre los pines de I/O y la lógica interna del dispositivo. Cada bloque soporta el flujo bidireccional además de controlar 3 estados.
- Bloques de RAM: proporciona almacenamiento de datos en forma de bloques doble puerto.
- Bloques multiplicadores: acepta dos números binarios como entrada para calcular el producto.
- Bloques DCM: Bloques que permiten la auto-calibración, soluciones digitales para multiplicar, dividir, realizar cambios de fase a una señal de reloj.

En la figura A.3 se presenta la organización de los bloques mencionados anteriormente.

La FPGA consta de máximo 66 pines disponibles para entradas/salidas,la mayoría de estos pines se disponen en 3 buses mencionados a continuación:

- Bus A[0:14]:Es un bus que realiza la comunicación entre la FPGA y la memoria DRAM.

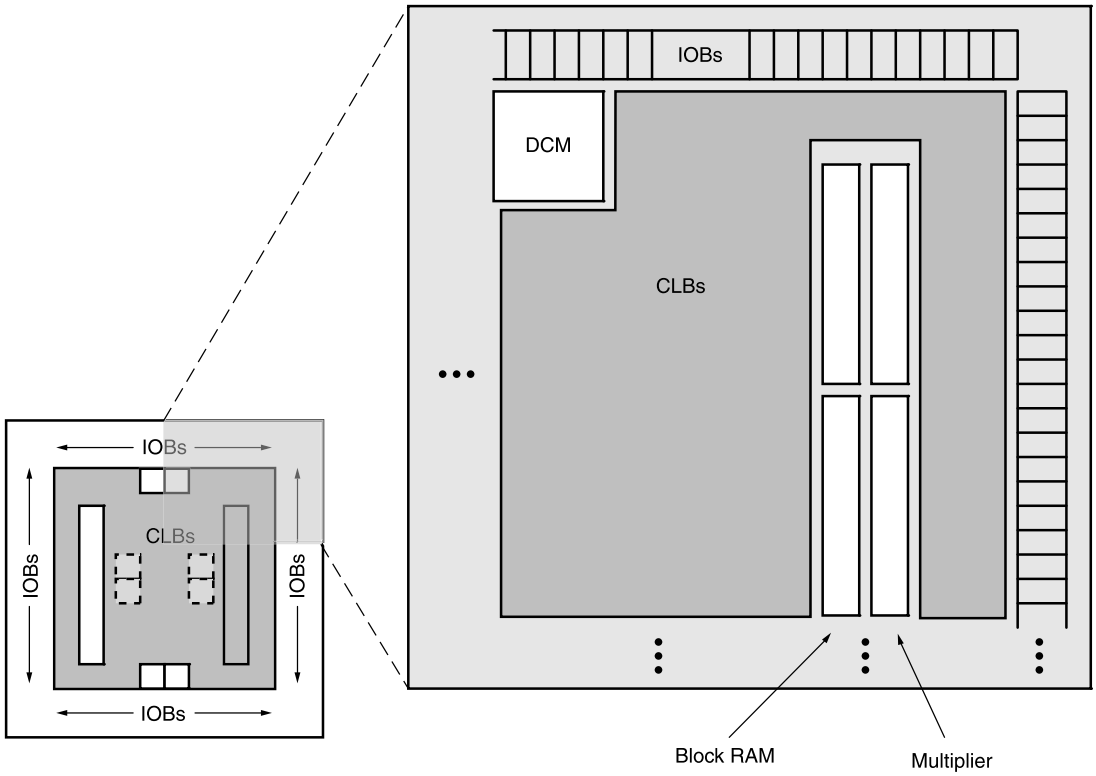


Figura A.3: Arquitectura de la familia *spartan-3E*.
FUENTE: Xilinx

- Bus D[0:15]: Es un bus que realiza la comunicación entre la FPGA La memoria NAND y la memoria DRAM.
- Bus DIG[0:26]: Es El conjunto de pines de propósito general, que puede configurar el usuario de la asignatura de sistemas digitales.

Otros pines son utilizados en la FPGA tales como DIGIN1, DIGIN2, OSC_XM, LED D5 los cuales son:

- DIGIN1: Es conocido en la plataforma como PB3.
- DIGIN2: Es conocido en la plataforma SIE como PB2.
- OSC_XM: Es la entrada al reloj de 50MHz de la SIE.
- LED D5: Es un LED que se presenta como salida en la FPGA.

Los pines usados en la asignatura de sistemas digitales, son los del bus DIG[0:26] entre otros, la tabla A.1 presenta el nombre que se le debe asignar en las restricciones de usuarios.

En la figura A.4, se asocian los nombres mostrados en la tabla con el conector GPIOs de tiene la tarjeta SIE.

DIG0	→	"P53"	DIG8	→	"P65"	DIG16	→	"P36"	DIG24	→	"P23"
DIG1	→	"P54"	DIG9	→	"P66"	DIG17	→	"P35"	DIG25	→	"P22"
DIG2	→	"P57"	DIG10	→	"P70"	DIG18	→	"P34"	DIG26	→	"P18"
DIG3	→	"P58"	DIG011	→	"P49"	DIG19	→	"P33"	DIGIN1	→	"P13"
DIG4	→	"P60"	DIG012	→	"P48"	DIG20	→	"P32"	DIGIN2	→	"P30"
DIG5	→	"P61"	DIG013	→	"P47"	DIG21	→	"P27"	OSC_XM	→	"P38"
DIG6	→	"P62"	DIG014	→	"P41"	DIG22	→	"P26"	LED D5	→	"P44"
DIG7	→	"P63"	DIG015	→	"P40"	DIG23	→	"P24"			

Tabla A.1: Tabla que asocia los nombres de los pines, con mencionados en archivos .ucf.

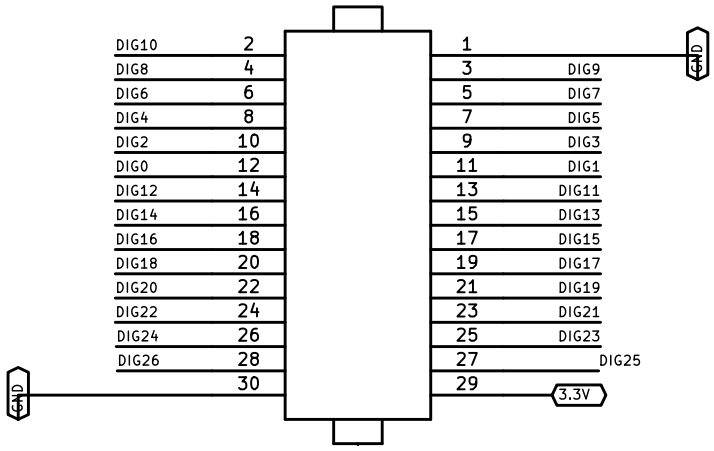


Figura A.4: Ubicación física de los pines GPIOs.

FUENTE: Linux en caja

A.1.2 Memoria flash

Es una memoria flash NAND de 2G, esta se encarga de almacenar los archivos .bit de configuración, que son enviados desde el PC. Otra propiedad importante de ésta memoria es su comportamiento no volátil.

A.1.3 Memoria DRAM

Es un tipo de memoria dinámica de acceso aleatorio. Se denomina dinámica, ya que para mantener almacenado un dato, se requiere revisar el mismo y recargarlo, cada cierto período, en un ciclo de refresco. Es una memoria volátil, es decir cuando no hay alimentación eléctrica, la memoria no guarda la información. En la figura **??** se presenta la plataforma SIE, mostrando los componentes más relevantes de esta.

A.1.4 Conexión de la tarjeta SIE

La conexión de la tarjeta SIE, se puede dividir en dos, como primera medida tenemos la comunicación, esta se da por el puerto USB y es por éste tipo de conexión que se configura la FPGA. El segundo tipo de conexión, que se

realiza en la SIE es la de alimentación, esta conexión se realiza para mantener más estable la alimentación de la tarjeta. Se procede a realizar la conexión de alimentación después de que se ha establecido la comunicación con la SIE.

A.2 Tarjeta de expansión de pines

La tarjeta de expansión de pines fue desarrollada para adaptar la plataforma SIE a las prácticas de los laboratorios de la asignatura de sistemas digitales, este desarrollo se da debido a la necesidad de utilizar los puertos GPIOs de la FPGA. En el desarrollo de la tarjeta se hizo el intento de hacerla muy fácil de manejar para los estudiantes, agregándole elementos muy utilizados en el área de sistema digitales, tales como pulsadores, interruptores, LEDs, *Pmods* y un zumbador.

Otro aspecto que se dejo abierto hace referencia a los módulos *Pmods* que permiten la adaptación de cualquier otro periférico a la tarjeta. Para realizar la tarjeta más configurable se le adicionó *jumpers* que permiten la multiplexación de algunos puertos de la FPGA, los pines de la tarjeta de expansión se encuentra nombrados, de tal manera que no se debe buscar ninguna información para realizar los archivos de restricciones del usuario (.ucf)

En la figura A.5 se presenta la tarjeta de expansión mencionada anteriormente.

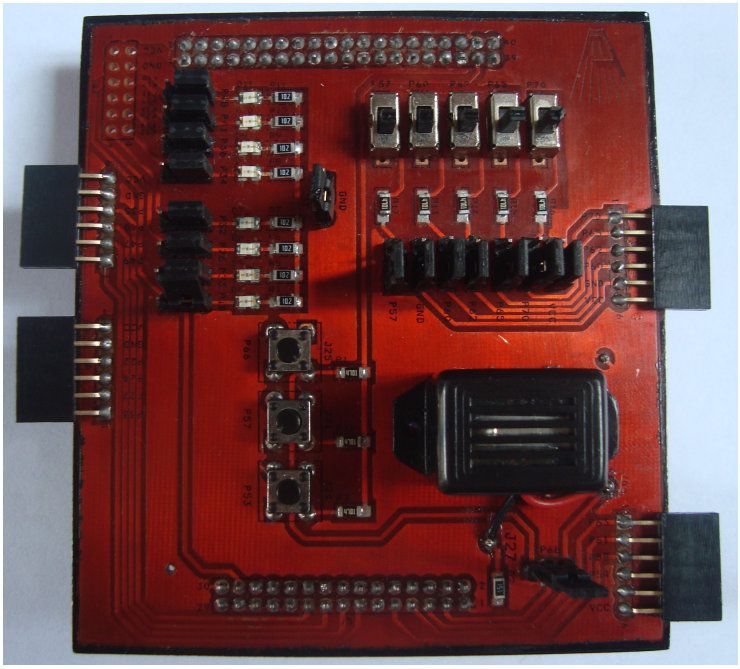


Figura A.5: Tarjeta de expansión de pines

FUENTE: Los autores

Como se puede apreciar esta tarjeta consta de los siguiente elementos:

- 5 interruptores
- 3 pulsadores
- 4 módulos *Pmods*
- 1 zumbador (buzzer)

Como se mencionó anteriormente algunos puertos se encuentran multiplexados, cuando se utilizan esos pines se debe tener cuidado de ubicar los *jumpers* de manera adecuada para que funcione la tarjeta de expansión.

Bibliografía

[1] Carlos Iván Camargo. Sie: Plataforma hardware copyleft para la enseñanza de sistemas digitales.

[2] Marcos Sanchez-Elez. Introduccion a la programacion en vhdl. http://www.dacya.ucm.es/marcos/intvhdl.pdf.

[3] Display de 7 segmentos. http://www.unicrom.com/Tut_display-7-segmentos.asp.

[4] Thomas L. Floyd. *Digital Fundamentals (7th Edicion)*. Prentice-Hall, Inc, Madrid, 2000.

[5] Ronald J. Tocci. *Sistemas Digitales principios y aplicaciones (6th Edicion)*. Pearson Educación, México, 2003.

[6] Mano M. Morris. *Arquitectura de computadoras (3th Edicion)*. Prentice-Hall, Inc, Madrid, 2000.

[7] David G. Maxinez , Jessica Alcalá Jara. *El arte de programar sistemas digitales (1ra Edicion)*. Compañia editorial continental, Inc, México, 2002.

[8] Digilent. http://www.digilentinc.com.

[9] Concurrencia. http://www.cannic.uab.es/docencia/DSD/Apunts/SintesisConcurrent.pdf.

[10] S.Fernández S.A. Pérez, E. Soto. Diseño de sistemas digitales con vhdl. http://www.dte.uvigo.es/vhdl/c5.html.

[11] Departamento de automática. Apuntes vhdl 00. http://atc2.aut.uah.es/~rico/docencia/asignaturas/informatica/lab_org_comp/archivos/Documentacion/VHDL/Apuntes%20VHDL%2000.pdf.

[12] Alberto Mora Javier García de Jalón, Iker Aguinaga. Aprenda linux como si estuviera en primero. http://www.digilentinc.com, 2000.

[13] Pong P. Chu. *Fpga Prototyping By Vhdl Examples,xilinx Spartan -3 Version*. Editorial:John Wiley Sons Inc.

[14] Dto. de Tecnología Electrónica. U. Rey Juan Carlos Puerto PS/2 http://laimbio08.escet.urjc.es/assets/files/docencia/DCSE/dcse_p8_ps2.pdf.

[15] DYNAMO electronics Puerto PS/2 http://www.dynamoelectronics.com/.

[16] measurement SPECIALTIES TM Interfacing Piezo Film to Electronics http://www.meas-spec.com/.

[17] TEXAS INSTRUMENTS 8-Bit 49 kSPS ADC Serial Out, Differential input, Configurable as SE input, 1 Ch. Single-Supply, MicroPower CMOS Operational Amplifiers MicroAmplifier Series http://www.ti.com/sitesearch/docs/universalsearch.tsp?searchTerm=TLV0831&linkId=1.

[18] USB-DRIVERS full suport for parallel ports on ISE 12 (2010-05-24) http://git.zerfleddert.de/cgi-bin/gitweb.cgi/usb-driver.

[19] XILINX ALL PROGRAMMABLE INSTALL-DRIVERS https://secure.xilinx.com/webreg/clickthrough.do?cid=103670.

[20] THE GEEK STUFF Execution sequence for .bash_profile, .bashrc, .bash_login, .profile and .bash_logout http://www.thegeekstuff.com/2008/10/execution-sequence-for-bash_profile-bashrc-bash_login-profile-and-bash_logout/.

[21] Creative Commons. *Licencias Creative Commons*. http://creativecommons.org/licenses., 2004.

[22] Camargo, Carlos Ivan. *SIE: Plataforma Hardware copyleft para la ensenanza de sistemas digitales*. Universidad Nacional de Colombia. 2010. 6 p.

[23] Qi Hardware http://en.qi-hardware.com/wiki/SIE.

[24] W.Spraul, C. Camargo, and A. Wrang. *Proyecto SACK*. http://en.qi-hardware.com/wiki/SAKC.

[25] Alberto Mora Javier García de Jalón, Iker Aguinaga. Aprenda linux como si estuviera en primero. http://www.digilentinc.com, 2000.

[26] In Electronics. Generación de señales PWM con el microcontrolador PIC16F84 http://usuarios.multimania.es/ccencho/2005Abril.pdf, 2004.

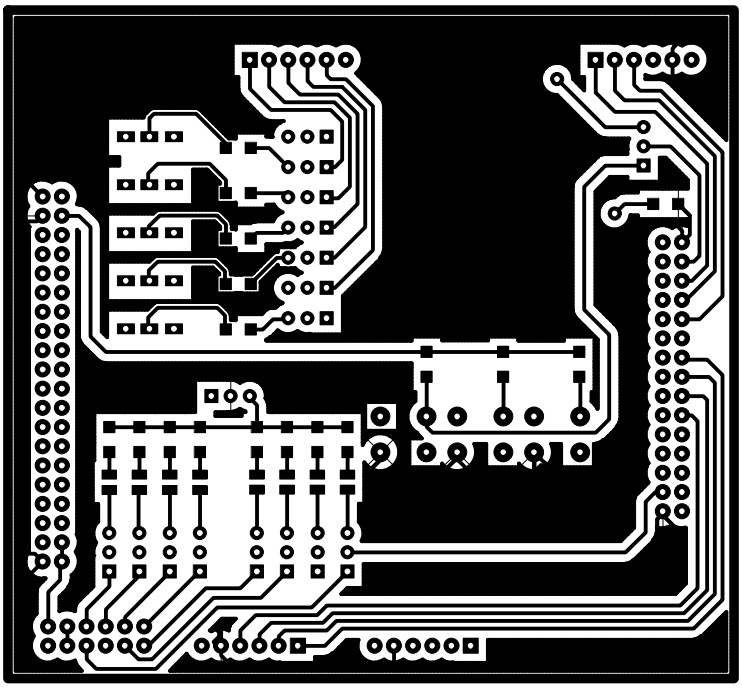
[27] MIGUEL GRASSI PuenteH.pdf. http://www.miguelgrassi.com.ar/mecatronica/PuenteH.pdf, 2006.

[28] FAIRCHILD SEMICONDUCTOR GENERAL PURPOSE 6-PIN PHOTOTRANSISTOR OPTOCOUPLERS. http://www.mbari.org/mars/images/4N37.pdf,

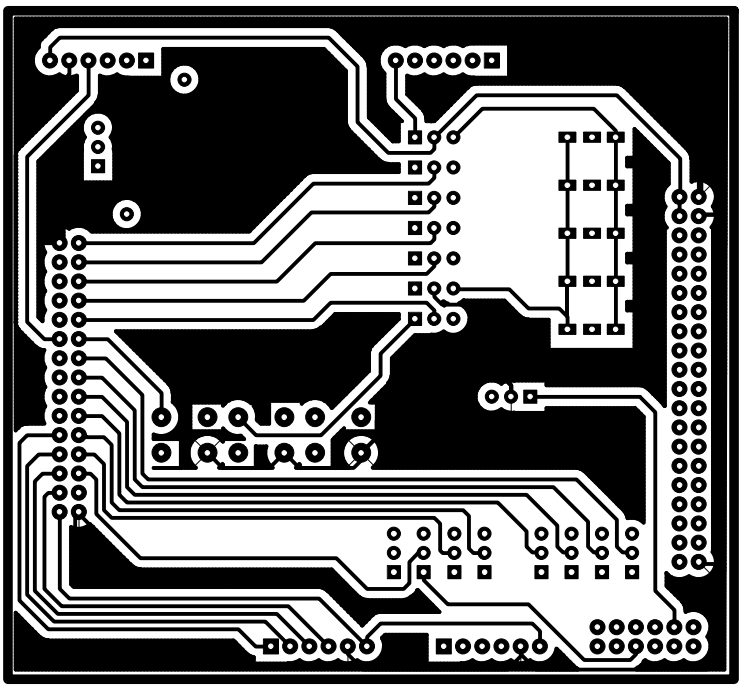
[29] Proyecto Electronicos http://proyectoselectronics.blogspot.com/2008/09/optoacoplador-que-es-y-como-funcionan.html,

[30] Heriberto Vargas Radillo CONTADOR ANILLO Y JOHNSON http://proton.ucting.udg.mx/dpto/maestros/hvargas/sd09/SD09.html,

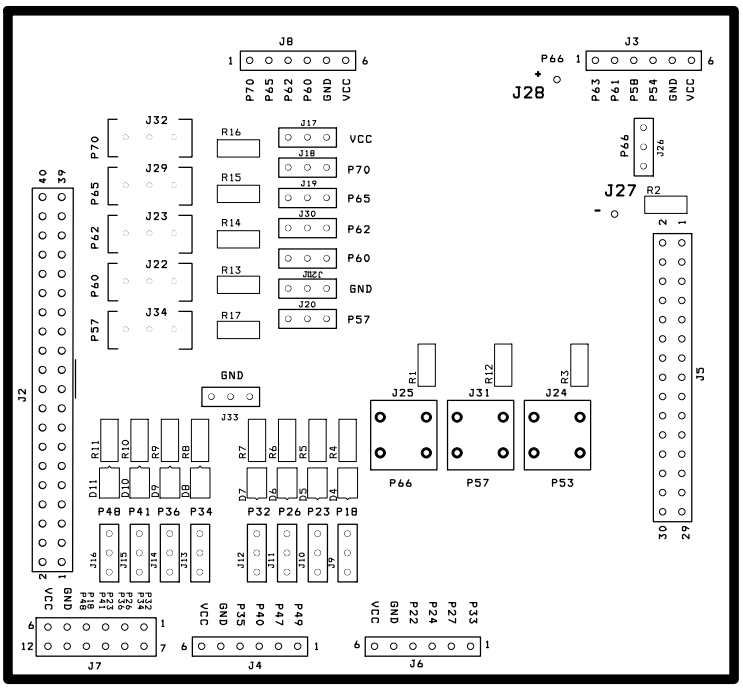
ANEXO D. PLACAS DE CIRCUITO IMPRESO PARA EL LABORATORIO DE SISTEMAS DIGITALES



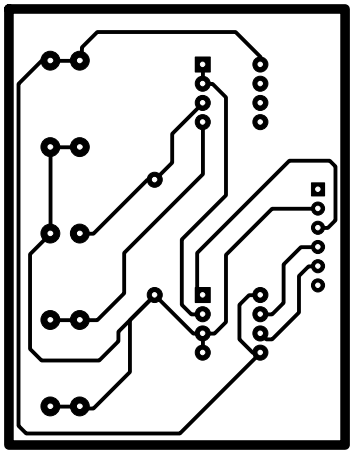
Tarjeta de expansión de pines SIE. Capa de arriba.



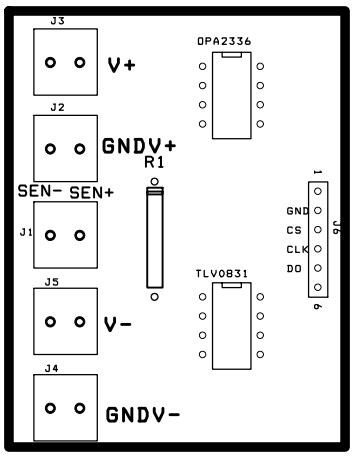
Tarjeta de expansión de pines SIE. Capa de abajo.



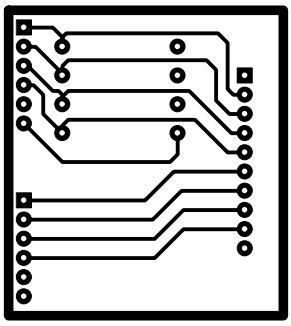
Tarjeta de expansión de pines SIE. Capa de letras



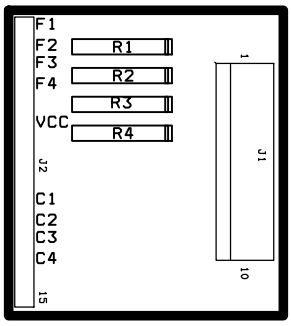
Proyecto edificio. Capa de arriba.



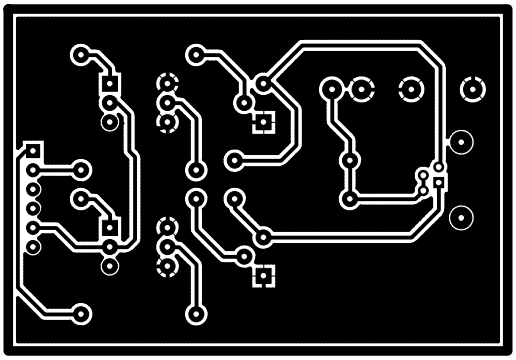
Proyecto edificio. Capa de letras.



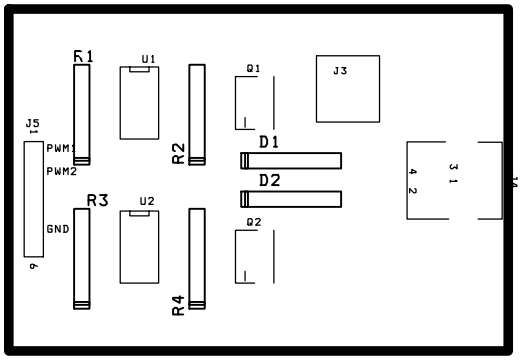
Proyecto carro: teclado. Capa de arriba



Proyecto carro: teclado. Capa de letras.



Proyecto carro: motores. Capa de arriba.



Proyecto carro: motores. Capa de letras.