UNIVERSIDAD AUTÓNOMA DE CIUDAD JUÁREZ

IIT

Instituto de Ingeniería y Tecnología

Manual de prácticas para la materia de Circuitos Digitales

Dr. Gabriel Bravo Martínez

Dr. Francisco Javier Enríquez Aguilera

Dr. Juan de Dios Cota Ruiz

Primera edición 2022

Universidad Autónoma de Ciudad Juárez

Instituto de Ingeniería y Tecnología

Departamento de Ingeniería Eléctrica y Computación

Colaboradores:

Dr. Gabriel Bravo Martínez

Dr. Francisco Javier Enríquez Aguilera

Dr. Juan de Dios Cota Ruiz

Cd. Juárez, Chihuahua, México 2022

Instituto de Ingeniería y Tecnología

Departamento de Eléctrica y Computación

Ing. en Sistemas Digitales y Comunicaciones

Ficha bibliográfica

Bravo M., Gabriel; Enríquez A., Francisco Javier; Cota R., Juan de Dios

Manual de prácticas para la materia de Circuitos Digitales

Primera revisión

Ciudad Juárez, Chih., México

Universidad Autónoma de Ciudad Juárez, 2022

Páginas: 147

Prefacio

El presente manual está hecho pensando en los conocimientos que el alumno debe de adquirir en la asignatura de Circuitos Digitales perteneciente a la Coordinación de Circuitos Digitales y Comunicaciones del Departamento de Eléctrica y Computación. El contenido del manual consta de 16 prácticas, iniciando por una práctica cognoscitiva del software de simulación Multisim, lo cual se puede aplicar en el primer día de laboratorio, el resto de las prácticas estará dado una por semana.

Contenido y organización

La primera práctica dará pie al alumno a conocer el software de simulación de circuitos Multisim y dispositivos básicos en la simulación. En la práctica 2 y 3 se comenzará el contacto con los circuitos integrados lógicos en simulación y en físico respectivamente. En la práctica 4 se continuará con el conocimiento de los circuitos integrados y la elaboración de una función lógica Tanto en simulación como en físico. En la práctica 5 se realiza una introducción a la creación de un proyecto y simulación en Vivado considerando Como hardware de desarrollo a la placa de desarrollo Basys 3. En las prácticas 6 y 7 Se introduce al alumno al uso de los aparatos de medición digital a través de una función lógica en simulación y físicamente respectivamente. La práctica 8 introduce la programación Y simulación en esquemático para Vivado, así como la programación de la placa Basys 3. La práctica 9 realizada decodificación para el display de 7 segmentos en la Basys 3. La práctica 10 propone un divisor de frecuencia y la práctica 11 introduce una segunda opción para el divisor de frecuencia junto con un contador. La práctica número 12 introduce al estudiante a la implementación de componentes en VHDL. La práctica 13 es la implementación de un flip-flop en VHDL y es utilizado para realizar un contador asíncrono. Las prácticas 14 y 15 implementan respectivamente un “contador síncrono” y un “contador síncrono módulo 10” utilizando el clip lo implementado en la práctica 13. Finalmente, la práctica 16, es la realización de una máquina de estado finito con lo que culmina el curso, dejando la puerta abierta al proyecto final De la materia.

Práctica # 1 Implementación y simulación de entrada y salida digital en Multisim

Objetivos de la práctica # 1

1. Conocer y utilizar el Multisim para realizar la simulación del circuito digital propuesto.
2. Conocer y emplear los componentes básicos dentro de Multisim para determinar una entrada y una salida de un sistema digital con el fin de realizar la simulación del circuito digital propuesto.
3. Conocer y emplear los cálculos para la resistencia limitadora de un LED.

Material y equipo

Enseguida se muestra en la tabla 1.1 los componentes y equipo a utilizar en la presente práctica, tener en cuenta que hay que hacer los cálculos para una de las resistencias en el transcurso de la misma práctica.

Tabla 1.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con software Multisim

Introducción

De los componentes

Los circuitos digitales se encuentran presentes en nuestra vida cotidiana, representados por computadoras, relojes, aparatos domésticos entre otros. Los circuitos digitales utilizan niveles de voltaje, el que es proporcionado por la fuente de alimentación (V_{cc}), para comunicarse entre sí, de tal forma que se presentan dos posibles estados, ausencia de voltaje (0 Volts) y la presencia de voltaje (Lo más común en microprocesadores y microcontroladores son 5, 3.3 y 1.8 Volts); a la representación de estos estados se le denomina un 0 lógico y un 1 lógico respectivamente. Los componentes que utilizamos para generar la entrada lógica al circuito son el interruptor de un polo y un tiro (SPST del inglés *Single Pole Single Throw*), una resistencia y el estado generado se representa con un visualizador de forma ilustrativa. Los 5 elementos mencionados anteriormente se muestran en la figura 1.1, donde a) es el símbolo de una resistencia de 1 kΩ, b) es el símbolo de un

interruptor SPST abierto, c) el visualizador del estado generado y d) símbolo de fuente de alimentación en 5 V.

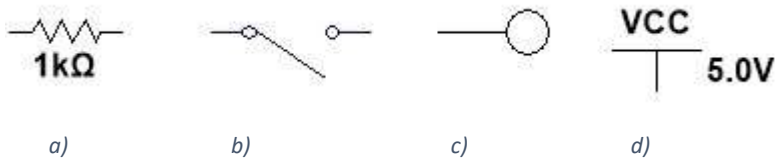


Figura 1.1 Símbolos a) Resistencia, b) interruptor, c) visualizador y d) símbolo de fuente de alimentación.

Para introducir manualmente un valor en un circuito lógico, utilizamos interruptores para generar el estado lógico deseado, de tal forma que, si el interruptor está abierto, obtendremos un 0 lógico, y al estar cerrado obtendremos un 1 lógico. Un primer intento de conectar un interruptor para generar los estados lógicos (y que es una conexión incorrecta) se presenta en la figura 1.2, con el interruptor abierto en a) y con el interruptor cerrado en b), sin embargo se observa que el voltaje de salida es igual a cero; Hay que evitar utilizar un interruptor de esta manera, pues al intentar general el 1 lógico, lo que se está generando al cerrar el interruptor es un cortocircuito (es decir, una conexión directa del voltaje de alimentación hacia tierra), por lo que muy probablemente se pueda dañar la fuente de alimentación si no está protegida. Hay que evitar este tipo de conexión.

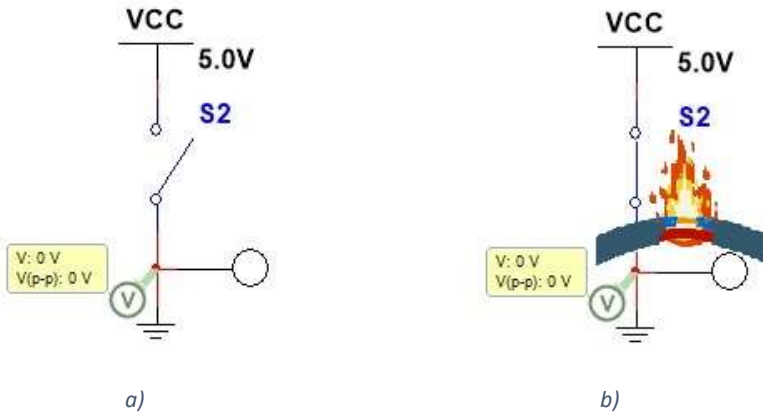
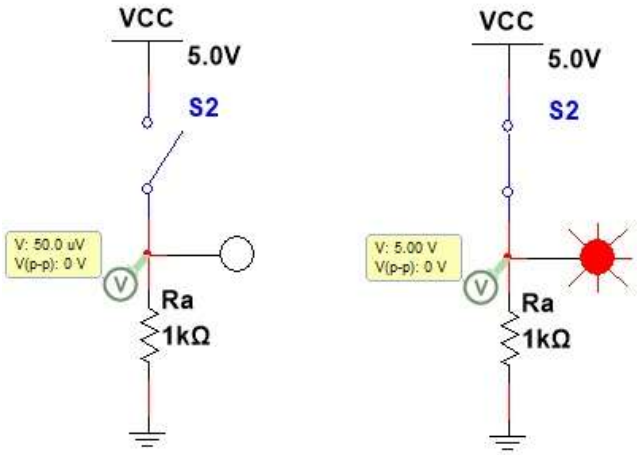


Figura 1.2 Forma incorrecta de conectar un interruptor.

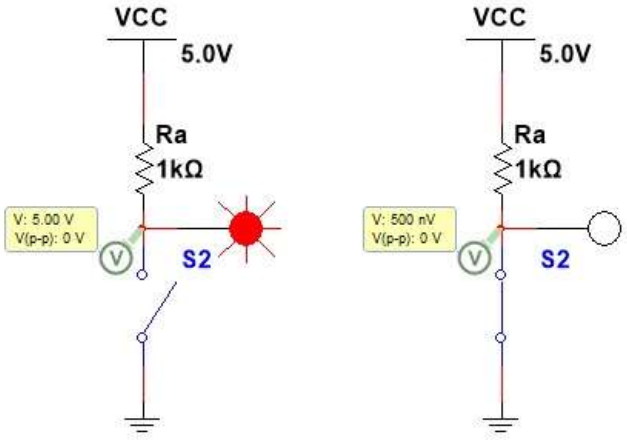
La conexión correcta del interruptor se presenta en la figura 1.3, en conjunto con una resistencia que llamaremos "Resistencia Pull Down" y la salida digital proporcionada. En el inciso a) de la figura 1.3 se muestra el interruptor abierto, generando un estado de ausencia de voltaje a la salida digital (0 lógico) y en el inciso b) de la figura 1.3 se muestra el interruptor cerrado, generando un estado de presencia de voltaje a la salida digital (1 lógico).



a) *b)*

Figura 1.3 Conexión de interruptor para generar una entrada digital manual.

En la figura 1.3 a), cuando el interruptor está abierto, se muestra que el voltaje es de 50 uV (micro volts), lo que se considera prácticamente 0 Volts. Es precisamente donde se muestra la punta de prueba donde se considera la entrada lógica al circuito, por lo que a partir de ese punto se hará la conexión hacia el primer elemento del circuito digital, el cual puede ser una compuerta lógica generalmente. Una forma alternativa para la conexión del interruptor es la mostrada en la figura 1.4, en la que se puede observar en el inciso a) que, para generar un 0 lógico, el interruptor permanece cerrado y haciendo uso de una resistencia que denominaremos "Resistencia Pull Up", y para generar un 1 lógico, el interruptor permanece abierto en la figura 1.4 b).



a) *b)*

Figura 1.4 Conexión alternativa del interruptor.

La salida de un circuito digital, cuando se trata de un estado individual, se representa por un LED (*light-emitting diode*), los cuales se encuentran en el mercado de diferentes tamaños

y colores siendo el más común el color rojo de 5 mm. En la figura 1.5 se muestra la polaridad y el símbolo de un LED, especificando el ánodo y el cátodo, pues es un dispositivo polarizado y su conexión a la alimentación debe de ser la adecuada, la alimentación positiva en el ánodo y la alimentación negativa en el cátodo.

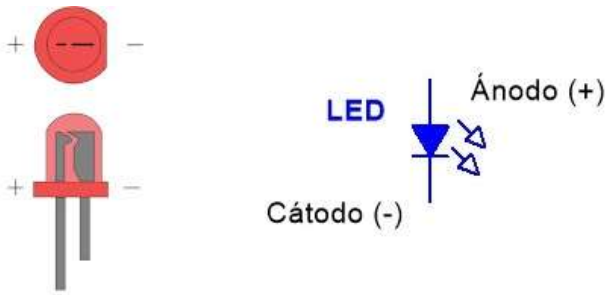
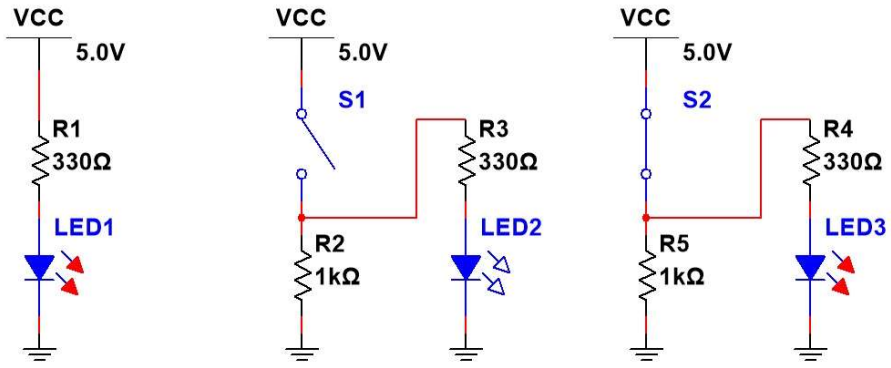


Figura 1.5 Polaridad y símbolo del LED.

Los LED's Tienen definido un voltaje de polarización, en el cual depende del tamaño y color del LED. Para alimentar el LED con su voltaje adecuado, se utiliza una resistencia limitadora, la que se conecta en serie con el LED. En la figura 1.6 se muestra la polarización de un LED encendido de forma directa (a), apagado a través de un interruptor abierto (b) y encendido a través de un interruptor cerrado (c).



a) *b)* *c)*

Figura 1.6 Polarización de un LED, a) directa, b) y c) con interruptor.

Para determinar la resistencia limitadora de corriente, hacemos uso de la ley de Kirchhoff de voltaje para una trayectoria cerrada, la cual dice, "la sumatoria de los voltajes en una trayectoria cerrada es igual a cero". considerando la trayectoria de la fuente de alimentación a través de la resistencia y el LED, resulta la siguiente ecuación:

$$\sum V_i = 0 = V_{cc} - V_R - V_{LED} \qquad 1.1$$

Por lo que si se conoce el voltaje de alimentación, el voltaje y corriente del LED y sustituimos el voltaje de la resistencia aplicando Ley de Ohm, es decir, VR = IR, donde la corriente de la resistencia es la misma corriente del LED, tenemos:

$$V_{cc} = V_R + V_{LED} = I_{LED}R + V_{LED} \qquad 1.2$$

Como la única incógnita es la resistencia, despejamos:

$$R = \frac{V_{cc} - V_{LED}}{I_{LED}} \qquad 1.3$$

Del software y el equipo

Multisim es un software de National Instruments utilizado como estándar en diferentes industrias para el diseño y simulación de circuitos digitales y es una herramienta que nos permite simular los circuitos antes de proceder a su implementación física, con lo cual, evitamos correr riesgos de dañar algún circuito y poder realizar cambios necesarios en la lógica del mismo circuito. La interfaz del Multisim es bastante intuitiva y los principales elementos los podemos ver en la figura 1.7.

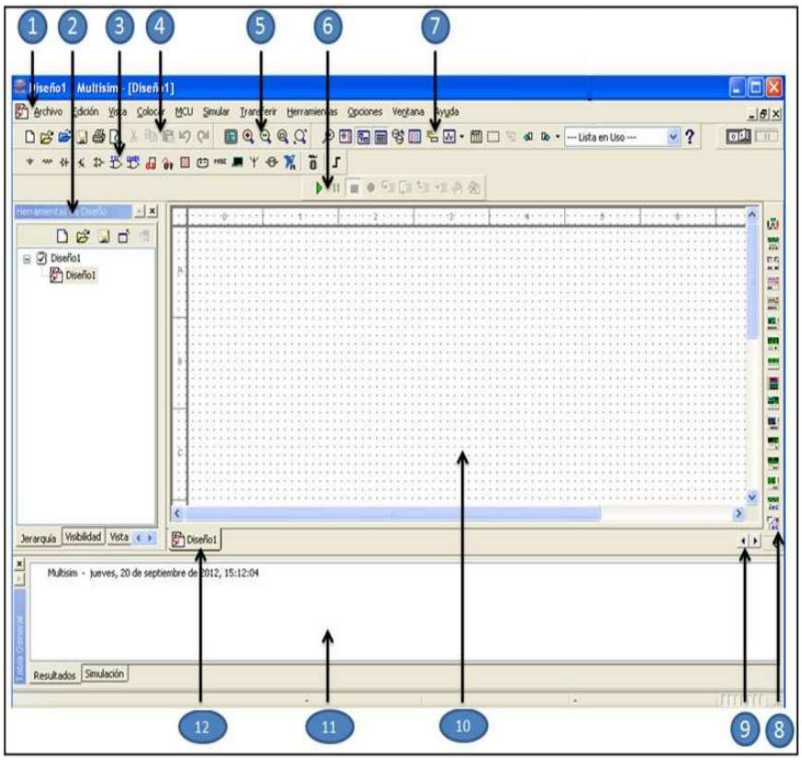


Figura 1.7 Interfaz de Multisim.

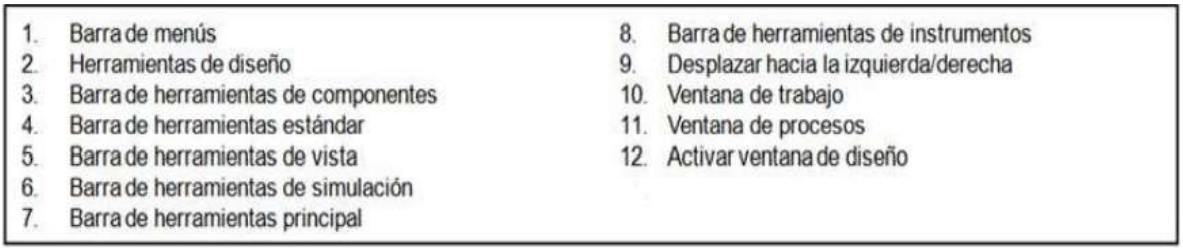


Figura 1.7 Interfaz de Multisim (continuación).

Para el uso de Multsim es muy conveniente emplear los atajos de teclado los cuales, por grupo, los más comunes son:

1. Comandos de edición (*Edit*):

90 grados en el sentido de las agujas del reloj Ctrl+R

90 grados en contra del sentido de las agujas del reloj Ctrl+Shift+R

Cancelar Esc

Copiar Ctrl+C

Cortar Ctrl+X

Eliminar Delete

Buscar Ctrl+F

Voltear horizontalmente Alt+X

Voltear verticalmente Alt+Y

Pegar Ctrl+V

Rehacer Ctrl+Y

Seleccionar todo Ctrl+A

Deshacer Ctrl+Z

2. Comandos de colocación (*Place*)

Arc Ctrl+Shift+A

Bus Ctrl+U

Ellipse Ctrl+Shift+E

HB/SC Connector Ctrl+I

Hierarchial Block from File... Ctrl+H

Junction Ctrl+J

Line Ctrl+Shift+L

New Subcircuit Ctrl+B

Más utilizado → Place Component Ctrl+W (colocar componentes en la ventana de trabajo)

Place Wire Ctrl+Q

Polygon Ctrl+Shift+G

Replace by Hierarchical Block Ctrl+Shift+H

Replace by Subcircuit Ctrl+Shift+B

Text Ctrl+T

El comando más frecuente al iniciar un circuito en Multisim es "*Place Component*", Con el cual se realiza la selección del componente a insertar en la ventana de trabajo. En la figura 1.8 se muestra la ventana de selección de componente.

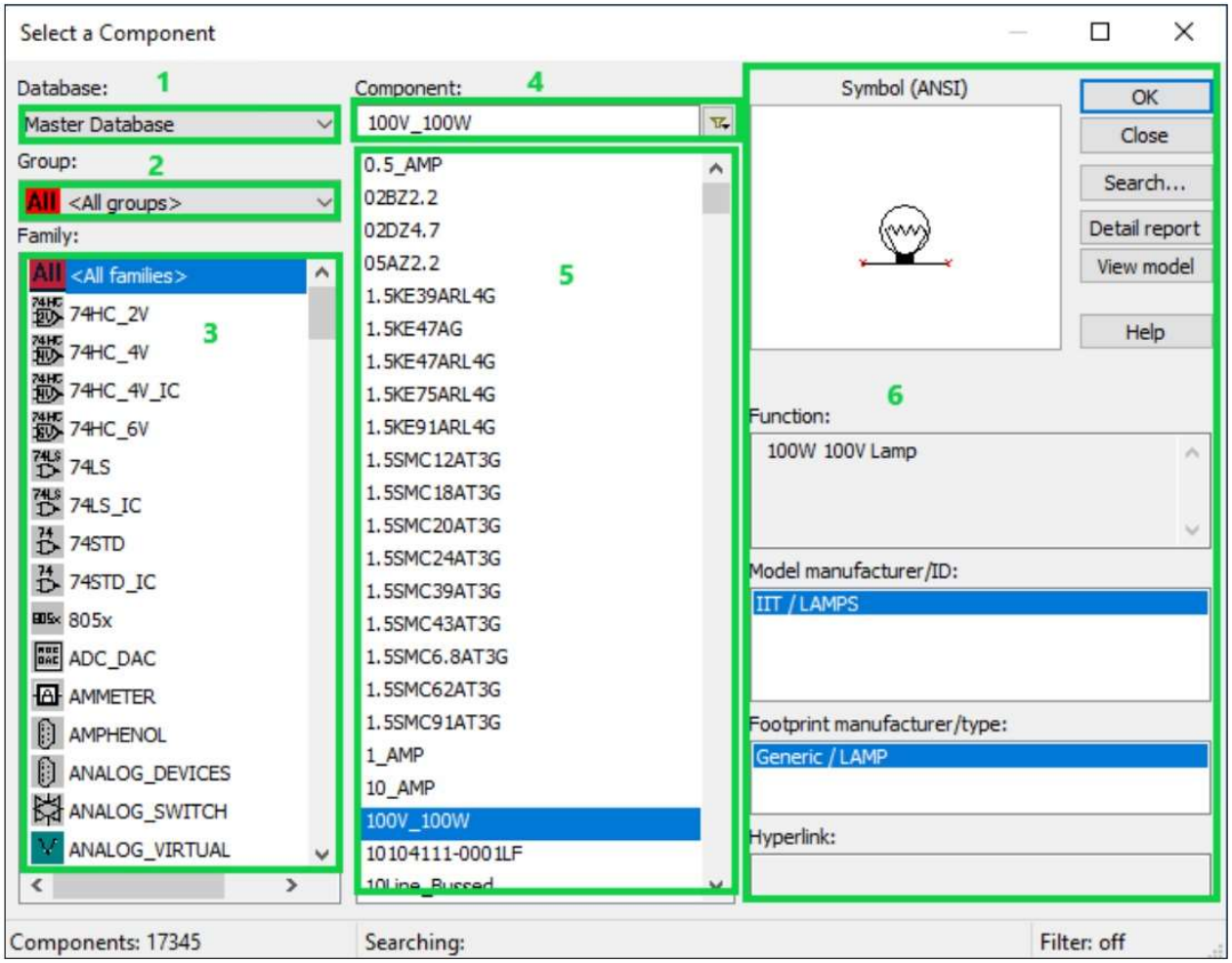


Figura 1.8 Ventana de selección de componente.

Enseguida se enumeran las opciones de la ventana de Selección de componente:

1. **Database**: bases de datos de Multisim, propias del usuario o importadas.

2. **Group**: grupos en los que se divide la base de datos.

3. **Family**: familias en las que se engloban los componentes de cada grupo.

4. **Component**: en este campo se puede filtrar directamente por el nombre del componente.

5. Listado de componentes: se rellena tras seleccionar una familia.

6. Propiedades del componente: cuando se selecciona un componente aquí se muestran todas sus características junto con su símbolo en el simulador.

En el análisis de los circuitos digitales se requiere de diferentes equipos, tanto para la alimentación del circuito como para la medición de los parámetros de este. El equipo necesario para suministrar la potencia al circuito se denomina fuente de alimentación. Por lo general vamos a trabajar con 5 V. En la figura 1.9 se muestra la carátula de una fuente de alimentación de triple canal programable, la cual puede programar tanto el voltaje de alimentación máximo, así como la corriente máxima.

Figura 1.9 Carátula de una fuente de alimentación de triple canal programable.

Para programar un canal de esta fuente, primero seleccionas el canal deseado a utilizar (CH1, CH2 o CH3) seguido del botón de programación de voltaje o de corriente según sea el caso (V-Set o I-Set), selecciona en el teclado numérico la magnitud deseada de la variable seleccionada (observa en el visualizador (*display*) los valores que se introducen) y finalizas presionando el botón denominado "*Enter*".

Una vez que se ha generado el voltaje este es necesario medirse en los diferentes puntos del circuito. El aparato que hace la medición de voltaje corriente y otras variables eléctricas recibe el nombre de multímetro. El multímetro 34401A Está disponible en el Laboratorio de Digitales actualmente y su carátula se muestra en la figura 1.10.

Figura 1.10 Multímetro 34401A.

Este mismo multímetro es el que puedes acceder virtualmente a través de Multisim, como se indica en la figura 1.11, accediendo a la barra de "*Instruments*" y seleccionando el botón del instrumento llamado "*Agilent multimeter*"; seguido aparecerá el icono del multímetro y al darle doble clic aparecerá la carátula del multímetro.

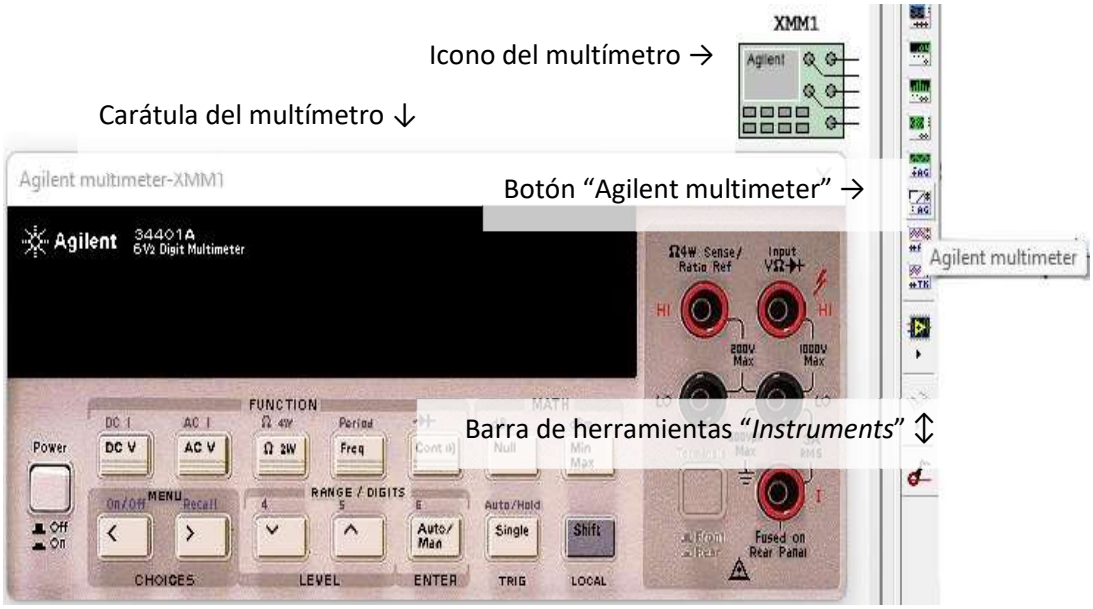


Figura 1.11 Agilent multimeter.

Análisis previo

Para realizar el cálculo de la resistencia limitadora del circuito de la figura 1.6 (b y c), empleamos la fórmula 3, sabiendo que vamos a alimentar al circuito con 5 V y que la corriente conveniente para un LED es de 10 mA, podemos determinar el voltaje requerido para un LED de color rojo simulando el circuito con una resistencia relativamente muy baja, en este caso de **100 Ω** de tal forma que **se garantice el encendido del LED**, como se indica en la figura 1.12. **Nota 1.1**: No armar en físico este circuito con la resistencia de 100 Ω porque al permitir pasar más corriente de la que tolera el LED, ocasionará que se queme el LED.

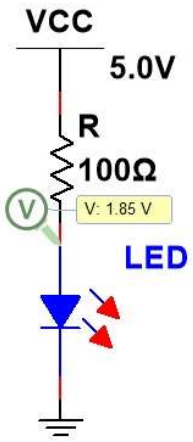


Figura 1.12 Simulación del circuito con una resistencia muy baja.

De acuerdo con la simulación de la figura 1.12, tenemos que el voltaje requerido para un led de color rojo es de 1.8 V, así, la fórmula 3 quedará de la siguiente forma:

$$R = \frac{V_{cc} - V_{LED}}{I_{LED}} = \frac{5 - 1.8}{10x10^{-3}} = 320\ \Omega \qquad 1.4$$

Para adquirir la resistencia de la ecuación cuatro, hay que acudir a un comercio, donde los valores de las resistencias a la venta son bien definidos. El valor comercial más cercano a 320 Ω es el de 330 Ω, valor que seleccionaremos para la simulación del circuito, así como para el circuito físico. **Nota 1.2**: Este cálculo se repetirá para cada color de LED.

Desarrollo

Después de realizar los cálculos necesarios para cada una de las resistencias que se utilizaran con cada uno de los leds que se van a analizar, se realizará la simulación del circuito, para ello se realizarán los siguientes pasos:

1. Ejecute el programa Multisim
2. Agregue componentes a la ventana de trabajo, utilizando el atajo de teclado **Ctrl-W**.
3. Una vez que está abierta la ventana de selección de componente, agregue los siguientes componentes, escribiendo en la caja de texto "C*omponent*": "VCC" para la alimentación de 5 V, "GROUND" para la referencia de tierra, "SPST" para el interruptor y "LED_red" para el LED rojo (y consecutivamente ir eligiendo el color para el resto de las simulaciones con su respectiva resistencia calculada). Para agregar las resistencias, en el grupo "*Family*", seleccione la familia "*RESISTOR*" y seleccione los valores de 1 kΩ y de 330 Ω (y consecutivamente ir eligiendo el valor calculado para cada color de LED) como se muestra en la figura 1.13.

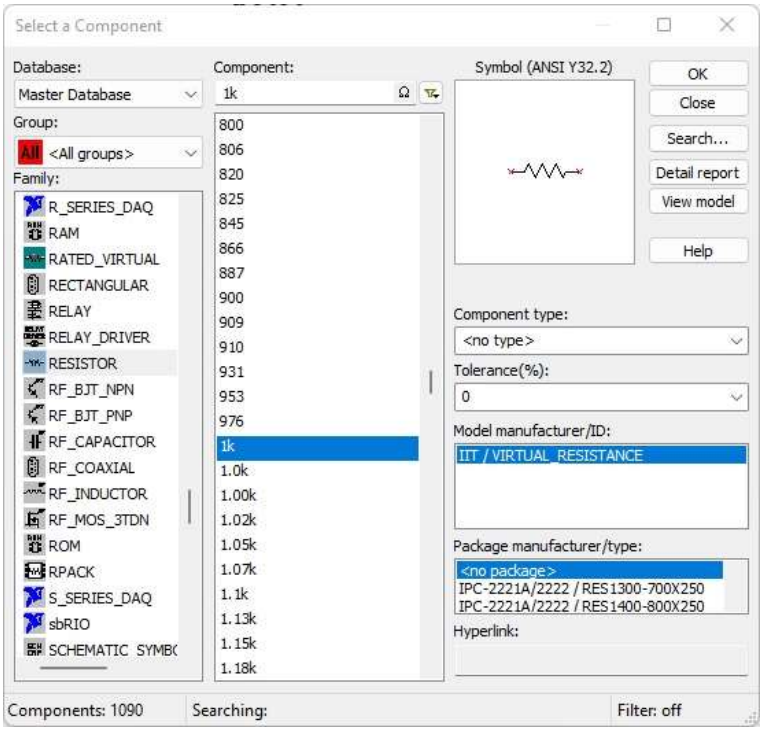


Figura 1.13 Selección de las resistencias.

El circuito simulado Similar al mostrado en la figura 1.14

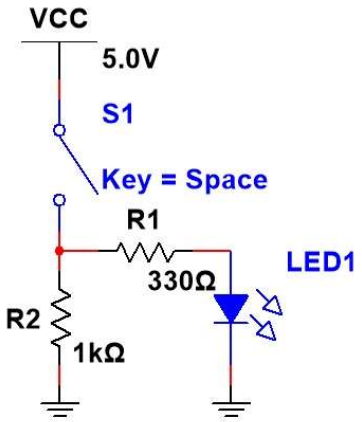


Figura 1.14 Circuito para el encendido del LED rojo.

Con el circuito armado en el simulador, se ejecutará la simulación dando clic en el icono de ejecución que se encuentra en la barra "*Simulation*"  . Enseguida cierre y abra el interruptor, ya sea con el puntero del ratón o con la barra espaciadora y observen el encendido y apagado del LED.

Una vez que haya comprobado que el led enciende y apaga, detenga la simulación dando clic en el botón "*Stop*" en la barra "*Simulation*"  . Agregue el instrumento llamado "*Agilent multimeter*" el cual se encuentra en la barra de "*Instruments*". Conecte la terminal

de referencia a la tierra del circuito y la terminal de voltaje al ánodo del LED, de doble clic en el icono del multímetro para que aparezca la carátula, como se muestra en la figura 1.15.

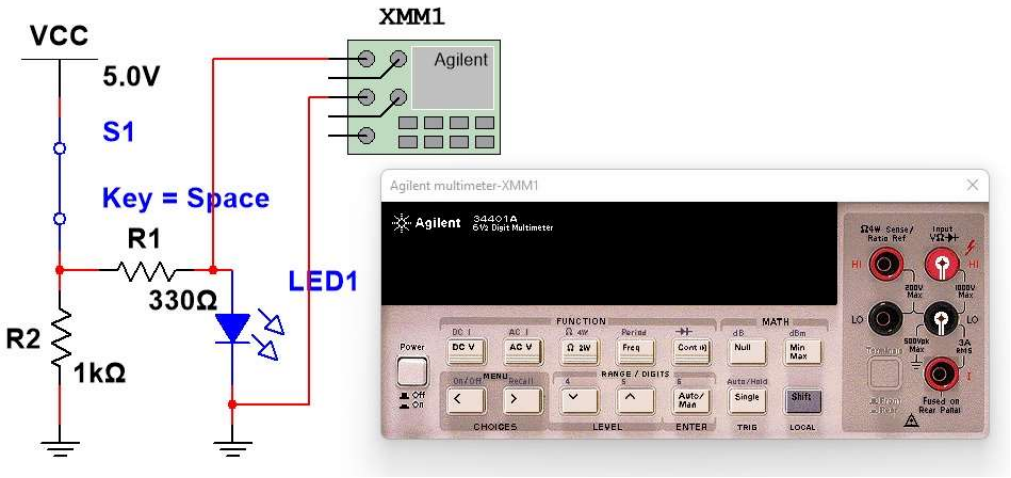


Figura 1.15 Circuito conectado al multímetro virtual "Agilent multimeter".

Encienda el multímetro virtual dando clic con el ratón en el botón "*Power*" 

, seleccione el tipo de voltaje que se va a medir, dando clic en el botón "*DC V*" 

. Ejecute la simulación dando clic en el icono de ejecución que se encuentra en la barra "*Simulation*"  y vuelva a encender y apagar el led observando y tomando nota los voltajes medidos con el multímetro virtual anotándolos en la tabla 1.2.

Vuelva a tomar nota de los voltajes medidos en el circuito después de intercambiar el LED color rojo por los indicados en la tabla 1.2, utilizando el menú contextual del componente LED1 y seleccionando la opción "*Replace components…*".

Tabla 1.2 Valores correspondientes a diferentes colores de LED.

Componente	Valor voltaje 1 lógico (LED encendido)	Valor voltaje 0 lógico (LED Apagado)	Valor resistencia limitadora
LED rojo			
LED verde			
LED amarillo			
LED naranja			
LED azul			

Incluir en el reporte de práctica los cálculos realizados, imágenes de las diferentes simulaciones y la tabla de valores medidos. **Nota** 1.3: La introducción dada en este manual

de prácticas no es la misma introducción que usted dará en su reporte, pues debe realizar una investigación de lo realizado en la presente práctica para utilizarlo como introducción.

Preguntas

1. ¿Cuál es el atajo para agregar un componente a la ventana del trabajo?
2. ¿En qué grupo y familia se encuentra el componente **Resistor**?
3. ¿En qué grupo y familia se encuentra el componente interruptor **SPST**?
4. ¿En qué grupo y familia se encuentra el componente **Vcc** y **Ground**?
5. ¿Qué botones se encuentran en la barra "*Simulation*" y cuál es su función?

Referencias

1. Bradshaw, Jaime Martín; "Documentación/tutorial multimedia del Simulador Multisim (módulos digitales)", trabajo fin de grado, MADRID 2018
2. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.
3. Londoño Ocampo, Oscar David; Restrepo Grisales, John Wilmar; "Manual para la simulación de circuitos de corriente alterna Senoidal haciendo uso del software multisim 11.0", Universidad Tecnológica de Pereira, Facultad de Tecnología, Pereira 2013
4. NI Circuit Design Suite Getting Started Guide.

Práctica # 2 Simulación con compuertas lógicas And, Or, Xor y Not

Objetivos de la práctica # 2

1. Conocer y utilizar los símbolos y las tablas de verdad para las compuertas lógicas Y (And), O (Or) , Or Exclusiva (Xor) , Nor Exclusiva (XNor) y NO (Not).
2. Conocer y simular los circuitos integrados de las compuertas Y (And), O (Or), O Exclusiva (Xor), Nor Exclusiva (XNor) y NO (Not).
3. Conocer y simular el uso de las compuertas en funciones que resuelvan un ejercicio sencillo de circuitos digitales.

Material y equipo

Enseguida se muestra en la tabla 2.1 los componentes y equipo a utilizar en la presente práctica, que será únicamente simulación.

Tabla 2.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con software Multisim

Introducción

Las compuertas lógicas son los elementos básicos de la solución a los problemas en el área de los circuitos digitales y las compuertas básicas son Y (And), O (Or) y No (Not). La compuerta O Exclusiva (Xor) se deriva de las anteriores, pero su uso es muy frecuente por lo que es conveniente comenzar a utilizarla. Las compuertas lógicas utilizan algebra booleana tanto para su entrada como para su salida.

La lógica booleana emplea sólo dos valores: falso y verdadero, los cuales son representados por 0 y por 1 respectivamente; estos valores son representados físicamente por los niveles lógicos de las compuertas TTL (lógica transistor-transistor por sus siglas en inglés) que son 0 V para el cero lógico y 5 V para el 1 lógico. En diferentes sistemas se utilizan otros valores, por ejemplo, en la comunicación serial se utiliza -15 volts para el 0 lógico y 15 V para el 1 lógico, mientras que en los microprocesadores actuales se manejan el 0 V y 3.3 V mientras que otros utilizan hasta 0-2.8 y 0-1.8 V. El empaquetado de estos dispositivos recibe el nombre de circuitos integrados C.I. En la figura 2.1 se muestra el empaquetado en plástico

de un circuito TTL 74LS86N en la que se indica una muesca que permite la identificación d la numeración de las terminales comenzando con la terminal número 1, este circuito pertenece a la familia LS (*low power schottky*).

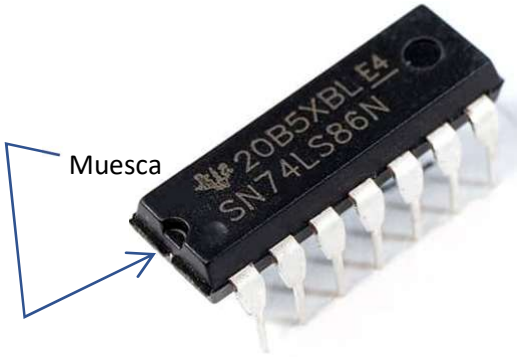


Figura 2.1 Circuito integrado TTL 74LS86N.

La información necesaria del C.I se puede encontrar en la hoja de datos la cual puedes encontrar realizando la búsqueda "74ls86n *datasheet*" y de la cual se muestra una porción de la información en la figura 2.2, en la cual indica que la *N* corresponde a un empaquetado de plástico y la numeración de las terminales iniciadas con la muesca que se encuentra en la parte superior izquierda de la imagen del circuito integrado. Finalmente, la asignación de las terminales, así como su diagrama lógico se muestran en la figura 2.3.

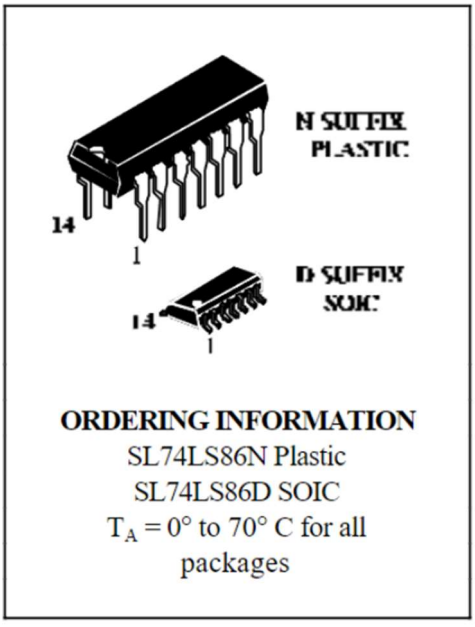


*Figura 2.2 Información obtenida de la hoja de datos "**datasheet**".*

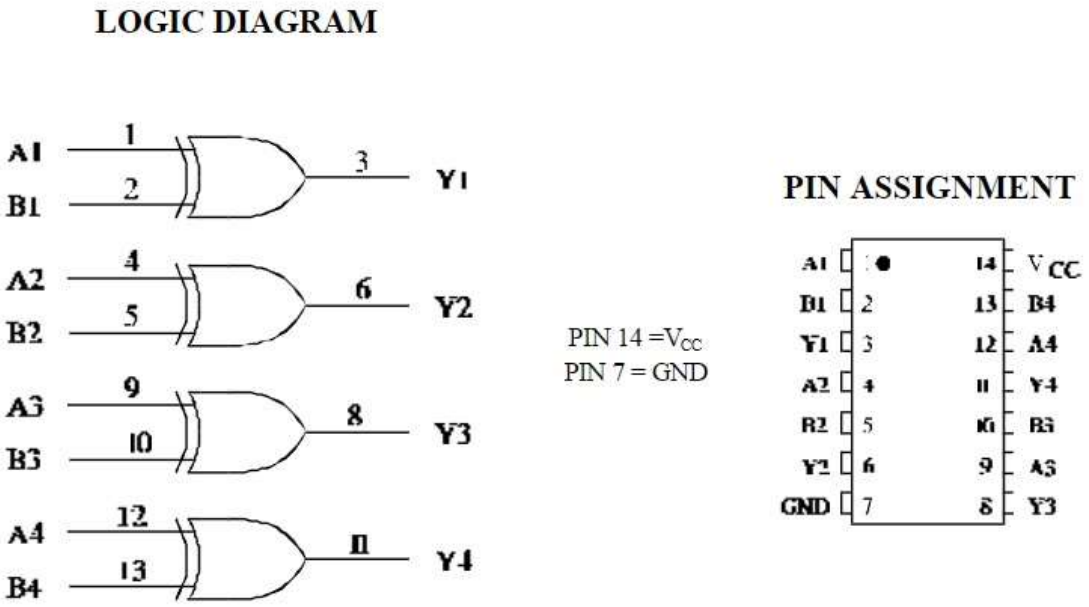


Figura 2.3 diagrama lógico y asignación de terminales del TTL 74LS86N.

Otro elemento importante de la información es la tabla de verdad, la cual proporciona el comportamiento de la salida de la compuerta debido a los valores lógicos de las entradas. en la figura 2.4 se muestra la tabla de verdad de la compuerta Xor, indicando como nivel lógico bajo "L" y como un nivel lógico alto "H".

FUNCTION TABLE

Inputs		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

Figura 2.4 Tabla de verdad de la compuerta Xor obtenida de la hoja de datos.

Análisis previo

Para tener una idea del funcionamiento de las compuertas lógicas, Investigar la tabla de verdad de las compuertas Y (And), O (Or) y No (Not) y explicar las en el reporte de la práctica.

Desarrollo

El desarrollo consta de 2 partes, la primera parte consiste en la comprobación de las compuertas Y (And), O (Or), O Exclusiva (Xor) y No (Not); la segunda parte consiste en analizar un ejercicio aplicable a circuitos digitales y obtener su solución utilizando las compuertas estudiadas en esta práctica.

Comprobación de las compuertas

La primera parte consiste en la simulación de las compuertas Y (And), O (Or), O Exclusiva (Xor) y No (Not). Se dará el procedimiento de la compuerta No (Not) y se deberá de seguir el mismo procedimiento para las restantes compuertas (que a diferencia de la compuerta no que solo tiene una entrada digital, las restantes tienen 2 entradas digitales) e incluirlo en el reporte de práctica.

1. Ejecute el programa Multisim
2. Agregue componentes a la ventana de trabajo, utilizando el atajo de teclado **Ctrl-W**.
3. Una vez que está abierta la ventana de selección de componente, agregue los siguientes componentes, escribiendo en la caja de texto "C*omponent*": "VCC" para la alimentación de 5 V, "GROUND" para la referencia de tierra, "SPST" para el interruptor y "LED_red" para el LED rojo. Para agregar las resistencias, en el grupo "*Family*", seleccione la familia "*RESISTOR*" y seleccione los valores de 1 kΩ y de 330 Ω como se muestra en la figura 2.5 y la parte del circuito realizado hasta el momento se muestra en la figura 2.6.
4. Modifica la tecla de activación del interruptor, el cual tiene asignada la tecla de espacio "Space", y asignarle la tecla "1".
5. Agregue la compuerta No (Not) a la ventana de trabajo, utilizando el atajo de teclado **Ctrl-W**.

Una vez que está abierta la ventana de selección de componente, en el selector de Grupo "*Group*" seleccione el grupo TTL 

seguido de la selección de la Familia "*Family*" 74LS 

y busque el componente 74LS04N y conéctelo entre la resistencia de 1 kΩ y la resistencia de 330 Ω. En la figura 2.7 se tiene el circuito completo para la simulación de la compuerta No (Not), en la cual se indica como la tecla de activación para el interruptor ha sido actualizado para la tecla "1".

6. Inicie la simulación del circuito  , e intercambie el estado del interruptor por medio de la tecla "1" y anote sus resultados en la tabla 2.2 (Observe que la compuerta

solo tiene una entrada, por lo que la tabla tendrá que ser modificada aumentandole una columna para el resto de las compuertas que tienen 2 entradas).

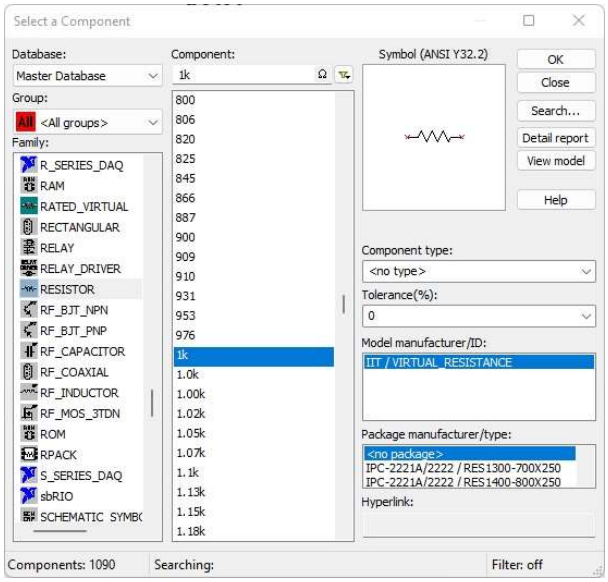


Figura 2.5 Selección de las resistencias.

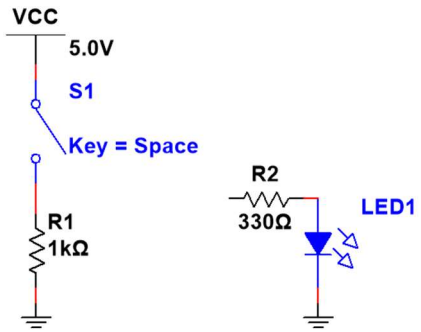


Figura 2.6 Avance del circuito para comprobar la compuerta No (NOT).

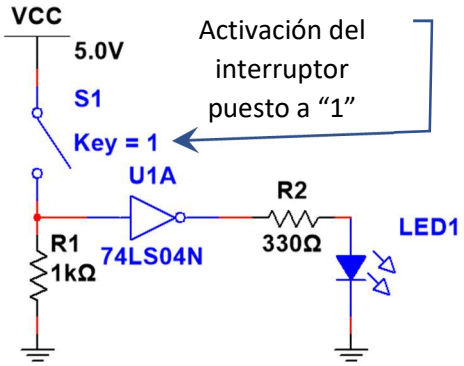


Figura 2.7 Circuito para la simulación de la compuerta No (Not).

Tabla 2.2 Aquí se capturan los resultados de la simulación de la compuerta No (Not).

Componente 74LS04	
Entrada (A)	Salida (Y)
0	
1	

Nota 2.1: Adicionalmente a las compuertas mencionadas anteriormente, investigue la tabla de verdad y haga la simulación de la compuerta XNor (74LS266); la compuerta 74LS266 tiene una salida que se denomina colector abierto (Open collector output), por lo que requiere de una configuración diferente; investiga la configuración requerida para una salida de colector abierto y realiza la simulación.

Aplicación de circuitos digitales

Enseguida se da el enunciado de un ejercicio que se puede resolver con circuitos digitales.

"Un sistema de cómputo tiene 2 discos, en los cuales se realiza el guardado de la información en redundancia (la redundancia en datos es la duplicación o re-escritura de información con la intención de aumentar la confiabilidad del sistema, generalmente en forma de respaldo de almacenamiento o prueba de fallas). Los discos utilizados se conectan a un circuito digital que los supervisa y cada disco le proporciona una señal digital al circuito, que cuando está trabajando correctamente el disco, entrega un estado lógico "1", en caso contrario arroja un estado lógico "0". El circuito digital de supervisión tiene 3 LEDs, un LED de color verde para indicar que los 2 discos están trabajando correctamente, un LED de color amarillo con el que indica que uno de los 2 discos está fallando, y finalmente un LED de color rojo para indicar que los 2 discos están fallando."

Realizar la simulación que resuelva la situación del enunciado anterior, donde los 2 discos se simularán por un interruptor, y las salidas serán los respectivos LEDs con el color indicado, los cuales deben de estar dirigidos por una o dos compuertas de las que sean comprendido en la presente práctica.

Preguntas

1. ¿Cuál es la diferencia entre las familias De los circuitos integrados TTL "L", "S" y "LS"?
2. ¿Cuántas compuertas lógicas contiene cada uno de los C.I. vistos en la presente práctica?

3. ¿Qué es salida de colector abierto?
4. ¿Cuál es la ventaja de una compuerta de colector abierto?
5. ¿Qué es la hoja de datos "*datasheet*"?

Referencias

1. 74LS86 Datasheet (PDF) - System Logic Semiconductor, https://pdf1.alldatasheet.com/datasheet-pdf/view/46213/SLS/74LS86.html, agosto 2022.
2. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.
3. NI Circuit Design Suite Getting Started Guide.

Práctica # 3 Uso de la placa de pruebas (*Protoboard*) y compuertas lógicas en CI

Objetivos de la práctica # 3

1. Conocer la configuración de una placa de pruebas.
2. Utilizar la placa de prototipos en la comprobación de las compuertas lógicas en CI's.
3. Conocer y aplicar el uso de las compuertas lógicas en CI's para realizar funciones que resuelvan un ejercicio sencillo de circuitos digitales.

Material y equipo

Enseguida se muestra en la tabla 3.1 los componentes y equipo a utilizar en la presente práctica, que será únicamente simulación.

Tabla 3.1 Material y equipo.

Cantidad	Material/equipo
1	Fuente de voltaje
1	Cables para fuente
1	Placa de prototipos
1	CI 74LS04
1	CI 74LS08
1	CI 74LS32
1	CI 74LS86
1	Interruptor tipo DIP (*DIP switch*)
4	LED (preferentemente diferente color) *
4	Resistencias de 330 Ω
3	Resistencias de 1 kΩ
--	Alambres para interconectar en la placa de pruebas

* Nota: primero realizar la comprobación de las compuertas y **presentar al docente**; después de su revisión, continuar con la solución del ejercicio.

Introducción

La placa de pruebas (*Protoboard*) Es una placa de plástico duro con perforaciones en las cuáles se pueden insertar los componentes electrónicos con el fin de realizar y comprobar prototipos de circuitos tanto eléctricos como electrónicos y digitales. La configuración de

una placa de pruebas, para permitir el contacto eléctrico entre los diferentes dispositivos que se inserten en sus perforaciones, es de 2 líneas (bus de alimentación) con 25 nodos o perforaciones en los extremos largos (generalmente marcados con una línea roja y una línea azul) en donde se conecta la alimentación de voltaje (V_{in} a la línea roja) y referencia o tierra (GND a la línea azul) y 35 líneas perpendiculares, cada una con 5 nodos o perforaciones referenciadas de **a** a **e** y de **f** a **j**. En la figura 3.1 se muestra una placa de pruebas translúcida lo cual permite visualizar las conexiones internas.

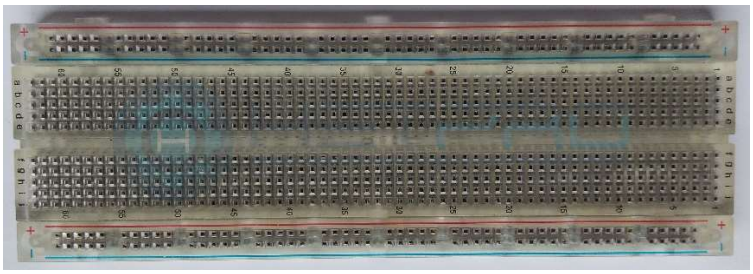


Figura 3.1 Imagen de una placa de pruebas.

En el centro de la placa se encuentra un canal central y a sus lados se derivan las 35 líneas mencionadas, cada una con 5 nodos o perforaciones de conexión y referenciadas de **a** a **e** y de **f** a **j**, perpendicularmente al canal. En la figura 3.2 se observa una imagen de una placa de pruebas, En la cual se dibujan 2 líneas verdes para definir los nodos comunes que se encuentran en los extremos de la placa, tanto en la parte superior como en la parte inferior de la vista en la imagen; de color morado se muestran los nodos interconectados las líneas perpendiculares al canal central.

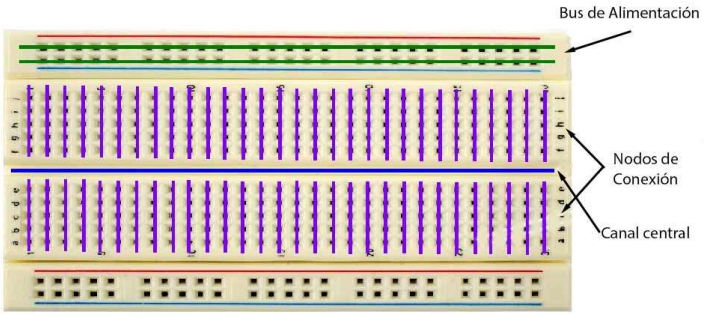


Figura 3.2 placa de pruebas con líneas que indican la continuidad de los diferentes nodos o perforaciones.

Un ejemplo de la conexión de dispositivos sobre la placa de pruebas se muestra una figura 3.3 donde se indican las siguientes conexiones:

a) Conexiones de alimentación positiva y tierra que se muestran con la anotación (1) y (2) respectivamente.
b) Conexión sobre el canal central de un interruptor DIP (*Dual In-line Package*) de cuatro interruptores, partiendo de la línea 2 a la línea 5 de la placa de pruebas, del cual se

utiliza el interruptor 1 que se conecta al voltaje de entrada por medio del nodo 2j en la parte superior de la figura 3.3 con la anotación (3), y se aterriza por medio de una resistencia de 1 kΩ en el nodo b2 con la anotación (4).

c) Conexión del circuito integrado 74LS04 sobre el canal central partiendo de la línea 7 hasta la línea 13, correspondiendo la terminal 1 del CI en el nodo en el nodo e7 indicado con la anotación (5); el CI se alimenta con 5 V conectando el nodo j7 por medio de un cable rojo que va al bus de alimentación superior positivo, y se conecta a tierra de un cable negro qué parte del nodo b13 hacia el bus de alimentación inferior negativo, ambos con la anotación (6) y (7) respectivamente. La terminal del interruptor 1 del empaquetado DIP se conecta a la entrada de la compuerta NOT de la terminal 1 del CI a través de un cable verde qué parte del nodo d2 al nodo d7, indicado con la anotación (8). Finalmente, la salida de la compuerta NOT se conecta al LED rojo por medio de un cable azul qué parte del nodo d8 y llega al nodo d20, que se indica con la anotación (9).

d) El LED conecta el ánodo en el nodo e20 y el cátodo en el nodo e19 como indica la anotación (10). El cátodo del LED se aterriza por medio de una resistencia de 1 kΩ qué parte del nodo b19 hacia tierra, indicado con la anotación (11).

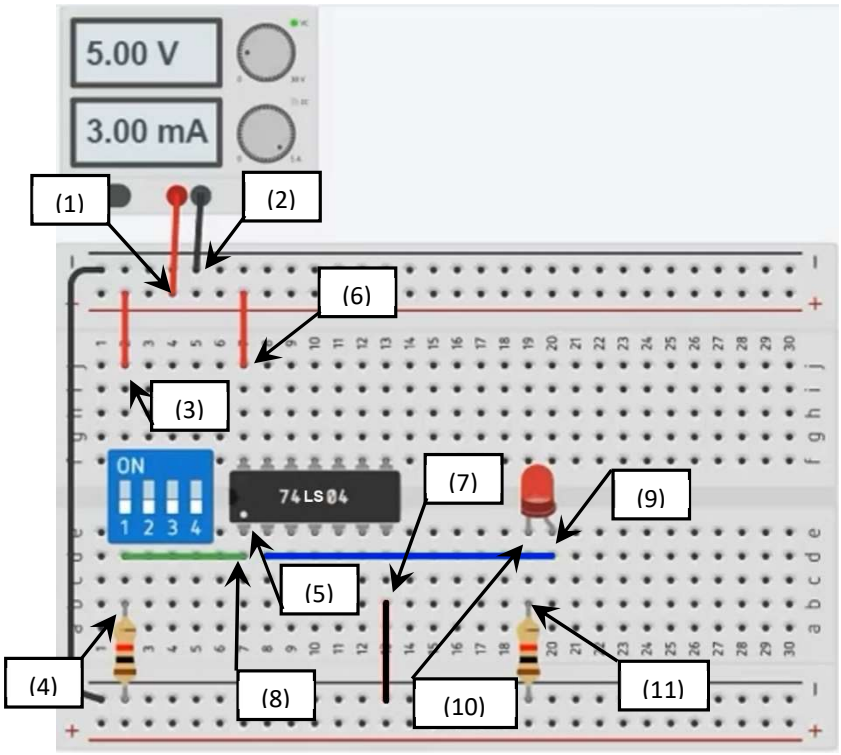


Figura 3.3 ejemplo de conexiones sobre la placa de pruebas.

Las compuertas lógicas son los elementos básicos de la solución a los problemas en el área de los circuitos digitales y las compuertas básicas son Y (And), O (Or) y No (Not). La

compuerta O Exclusiva (Xor) se deriva de las anteriores, pero su uso es muy frecuente por lo que es conveniente comenzar a utilizarla. Las compuertas lógicas utilizan algebra booleana tanto para su entrada como para su salida.

La lógica booleana emplea sólo dos valores: falso y verdadero, los cuales son representados por 0 y por 1 respectivamente; estos valores son representados físicamente por los niveles lógicos de las compuertas TTL (lógica transistor-transistor por sus siglas en inglés) que son 0 V para el cero lógico y 5 V para el 1 lógico. En diferentes sistemas se utilizan otros valores, por ejemplo, en la comunicación serial se utiliza -15 volts para el 0 lógico y 15 V para el 1 lógico, mientras que en los microprocesadores actuales se manejan el 0 V y 3.3 V mientras que otros utilizan hasta 0-2.8 y 0-1.8 V. El empaquetado de estos dispositivos recibe el nombre de circuitos integrados C.I. En la figura 2.1 se muestra el empaquetado en plástico de un circuito TTL 74LS86N en la que se indica una muesca que permite la identificación d la numeración de las terminales comenzando con la terminal número 1, este circuito pertenece a la familia LS (*low power schottky*).

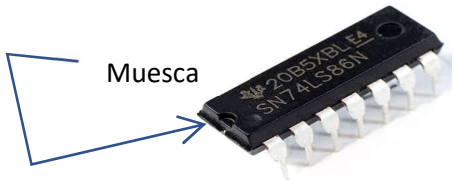


Figura 3.4 Circuito integrado TTL 74LS86N.

La información necesaria del C.I se puede encontrar en la hoja de datos la cual puedes encontrar realizando la búsqueda "74ls86n *datasheet*" y de la cual se muestra una porción de la información en la figura 2.2, en la cual indica que la *N* corresponde a un empaquetado de plástico y la numeración de las terminales iniciadas con la muesca que se encuentra en la parte superior izquierda de la imagen del circuito integrado. Finalmente, la asignación de las terminales, así como su diagrama lógico se muestran en la figura 2.3.

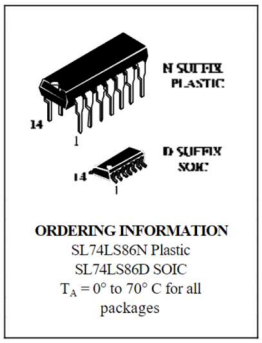


*Figura 3.5 Información obtenida de la hoja de datos "**datasheet**".*

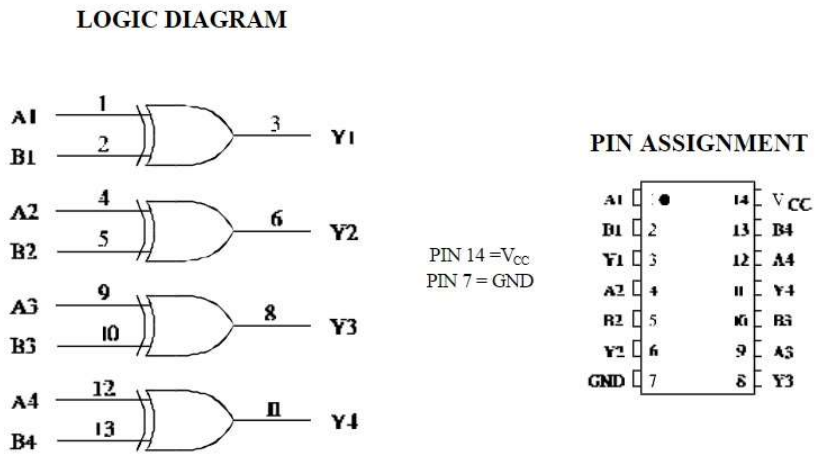


Figura 3.6 diagrama lógico y asignación de terminales del TTL 74LS86N.

Otro elemento importante de la información es la tabla de verdad, la cual proporciona el comportamiento de la salida de la compuerta debido a los valores lógicos de las entradas. en la figura 2.4 se muestra la tabla de verdad de la compuerta Xor, indicando como nivel lógico bajo "L" y como un nivel lógico alto "H".

FUNCTION TABLE

Inputs		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

Figura 3.7 Tabla de verdad de la compuerta Xor obtenida de la hoja de datos.

Análisis previo

Realizar una lista de las conexiones que se realizarán sobre la placa de pruebas, por ejemplo, para la compuerta Not sería la siguiente lista (incluir las listas de conexiones para cada compuerta en el reporte de práctica).

1. Conectar el interruptor sobre el canal central a partir de la línea uno dejando la primera terminal en e1.
2. Conectar el nodo j1 a Vcc (voltaje de alimentación) por medio de un alambre.
3. Conectar resistencia de 1 kΩ del nodo b1 a tierra.
4. Conectar el CI 74LS04 sobre el Canal central a partir del nodo e7.

5. Alimentar el CI conectando un alambre del nodo j7 al voltaje del bus de alimentación (Vcc) y otro alambre del nodo a13 al voltaje de referencia en el bus de alimentación (GND).
6. Conectar un alambre desde el interruptor (nodo d1) hasta la primera compuerta NOT del CI (nodo d7).
7. Conectar el ánodo y cátodo del LED en los nodos e15 y e16 respectivamente.
8. Conectar Pon un alambre la salida de la primera compuerta Not (nodo d8) hasta el ánodo del LED (nodo d15)
9. Conectar una resistencia al cátodo del LED (nodo b16) hacia voltaje de referencia (GND) en el bus de alimentación.
10. Conectar los cables de la fuente la placa de pruebas al bus de alimentación.

Desarrollo

El desarrollo consta de 2 partes, la primera parte consiste en la comprobación de las compuertas No (Not), Y (And), O (Or) y O Exclusiva (Xor); la segunda parte consiste en implementar el circuito para el ejercicio aplicable a circuitos digitales y obtener su solución utilizando las compuertas estudiadas en esta práctica.

Comprobación de las compuertas

La primera parte consiste en la comprobación de las compuertas No (Not), Y (And), O (Or) y O Exclusiva (Xor)). Se dará el procedimiento de la compuerta No (Not) y se deberá de seguir el mismo procedimiento para las restantes compuertas (que a diferencia de la compuerta no que solo tiene una entrada digital, las restantes tienen 2 entradas digitales) e incluirlo en el reporte de práctica.

1. Asegúrese de tener los componentes necesarios para la comprobación de la compuerta No (Not).
2. Realice las conexiones de acuerdo con la lista de conexiones de la sección “Análisis previo”.
3. Prenda la fuente de voltaje previamente configurada a 5 V y prenda y apague el interruptor 1 del interruptor tipo DIP.
4. Registrar en la tabla 3.2 sus observaciones del circuito.

Tabla 3.2 Aquí se capturan los resultados de la simulación de la compuerta No (Not).

Componente 74LS04	
Entrada (A)	Salida (Y)
0	
1	

Realizar lo anterior para el resto de los circuitos integrados enlistados en la tabla 3.1; No olvidar que el resto de las compuertas tienen 2 entradas las cuales puedes nombrar A y B, esto debe de quedar registrado en la tabla 3.2 al modificarla de forma conveniente.

Aplicación de circuitos digitales

Enseguida se da el enunciado de un ejercicio que se puede resolver con circuitos digitales.

"Un sistema de cómputo tiene 2 discos, en los cuales se realiza el guardado de la información en redundancia (la redundancia en datos es la duplicación o re-escritura de información con la intención de aumentar la confiabilidad del sistema, generalmente en forma de respaldo de almacenamiento o prueba de fallas). Los discos utilizados se conectan a un circuito digital que los supervisa y cada disco le proporciona una señal digital al circuito, que cuando está trabajando correctamente el disco, entrega un estado lógico "1", en caso contrario arroja un estado lógico "0". El circuito digital de supervisión tiene 3 LEDs, un LED de color verde para indicar que los 2 discos están trabajando correctamente, un LED de color amarillo con el que indica que uno de los 2 discos está fallando, y finalmente un LED de color rojo para indicar que los 2 discos están fallando."

Realizar el circuito, en la placa de pruebas, que resuelva la situación del enunciado anterior, donde los 2 discos se simularán por un interruptor, y las salidas serán los respectivos LEDs con el color indicado, los cuales deben de estar dirigidos por una o dos compuertas de las que sean comprendido en la presente práctica.

Preguntas

1. ¿Cómo puedes determinar la terminal número uno de un circuito integrado de TTL?
2. ¿Cuántas líneas tiene cada bus de alimentación?

3. ¿Cuántas compuertas tiene el circuito integrado 74LS04?
4. ¿Cuántas compuertas tienen el resto de los circuitos que utilizamos en esta práctica?
5. ¿Cuáles son las terminales de alimentación (Vcc y GND) de los circuitos TTL utilizados en esta práctica?

Referencias

1. 74LS86 Datasheet (PDF) - System Logic Semiconductor, https://pdf1.alldatasheet.com/datasheet-pdf/view/46213/SLS/74LS86.html, agosto 2022.
2. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.
3. NI Circuit Design Suite Getting Started Guide.

Práctica # 4 Circuito lógico combinacional

Objetivos de la práctica # 4

1. Conocer e identificar un circuito lógico combinacional.
2. Analizar un circuito lógico combinacional para obtener su expresión booleana.
3. Analizar un circuito lógico combinacional para obtener su tabla de verdad.
4. Comprobar el análisis de un circuito lógico combinacional en la placa de pruebas (protoboard).

Material y equipo

Enseguida se muestra en la tabla 4.1 los componentes y equipo a utilizar en la presente práctica, que será únicamente simulación.

Tabla 4.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Multisim
1	Fuente de voltaje
1	Cables para fuente
1	Placa de prototipos
1	CI 74LS04
1	CI 74LS08
1	CI 74LS32
1	Interruptor tipo DIP (*DIP switch*) con al menos **4** interruptores
1	LED
1	Resistencia de 330 Ω
4	Resistencias de 1 kΩ
--	Alambres para interconectar en la placa de pruebas

Introducción

A pesar de que la electrónica digital solo funciona con dos estados denominados como ceros y unos, existen diferentes formas en las que un sistema se puede comportar, ya sea que solo se dependa de las entradas para actualizar sus salidas, o que se requiera de una retroalimentación de los estados anteriores y/o el tiempo para que se actualice. A estos

diferentes tipos de respuestas se les conoce como circuitos lógico combinacional y circuitos lógico secuencial. Un circuito lógico combinacional se encarga de procesar (transformar) la información digital, la cual se presenta en forma de señales binarias; es aquel en donde la salida depende exclusivamente de sus entradas y no almacenan ningún tipo de información.

Para comprender el funcionamiento de un circuito lógico combinacional, lo podemos hacer a partir de lo que es su expresión booleana y de su tabla de verdad. Con la expresión booleana de un circuito se puede evaluar e incluso, utilizando álgebra booleana, minimizar la expresión. Con la tabla de verdad de un circuito lógico combinacional tenemos tubularmente la salida del circuito para cualquier combinación de valores en las entradas.

Análisis previo

La expresión booleana de un circuito lógico combinacional se obtiene escribiendo en la salida de cada compuerta del circuito la combinación lógica que se genera de sus entradas; esto se hace consecutivamente hasta llegar a la salida del circuito lógico combinacional. En la figura 4.1 se muestra un circuito lógico combinacional sencillo, el cual consta de 3 entradas (A, B y C).

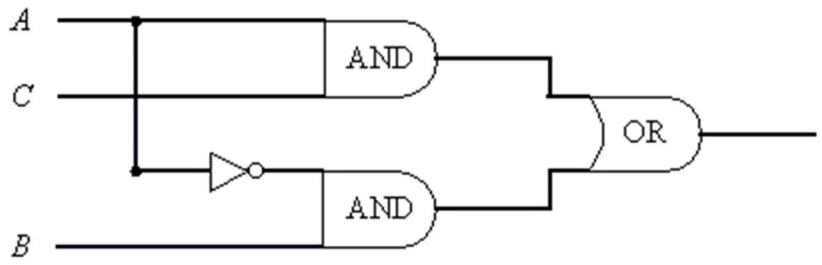
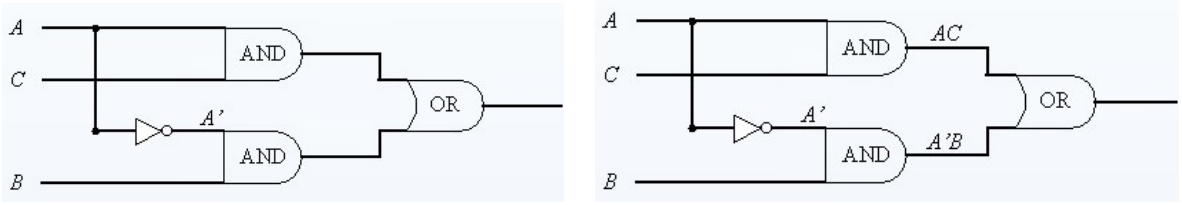


Figura 4.1 Circuito lógico combinacional para análisis de expresión booleana.

En la figura 4.2 a) se puede ver la notación de la salida de la compuerta No (Not) con A negada (A'), que es la consecuencia detener a la entrada la variable A. Continuando con la figura 4.2 b), se han agregado a las salidas de las compuertas Y (And) las combinaciones respectivas a las entradas de cada compuerta, siendo AC para la compuerta Y (And) superior y A'B para la compuerta Y (And) inferior. Finalmente, en la figura 4.3, que es el análisis de la compuerta final del circuito lógico combinacional, tenemos la combinación O (Or) de las combinaciones presentes a la entrada de la compuerta, siendo éstas las salidas de las compuertas Y (And) anteriores y finalizando la expresión booleana del circuito lógico combinacional.



a) b)

Figura 4.2 Análisis para la obtención de la expresión booleana.

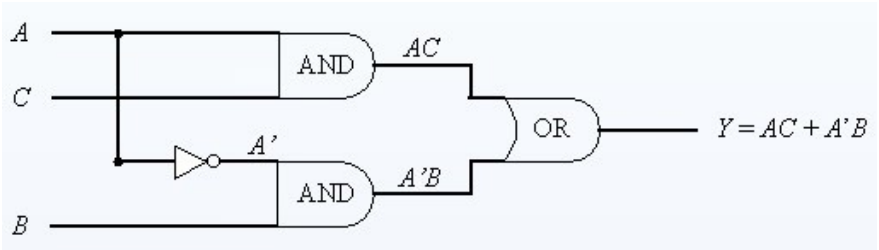


Figura 4.3 Resultado del análisis para obtener la expresión booleana.

La tabla de verdad de un circuito lógico combinacional la podemos obtener de la expresión booleana del mismo. Primeramente, realiza la tabla de verdad poniendo en el encabezado las variables involucradas en la expresión booleana seguida de todas las combinaciones binarias; seguido, observa la expresión booleana y sepárala en expresiones que estén asociadas con O (OR) o con Y (And). En nuestro ejemplo sencillo observamos que tenemos 2 expresiones (AC y A'B) unidas con una O (OR), analizamos cada una por separado. Sabemos que como están unidas con una expresión O (OR), entonces colocaremos un uno (1) cuando la expresión analizada sea verdadera. En el caso de la expresión AC, sabemos que es verdadera cuando tanto A y C lo son, Lo cual nos lleva a poner un uno en la tabla cuando a A y C valen uno independiente demente del valor de B; Esta parte de la tabla se muestra en la tabla 4.2.

Tabla 4.2 Análisis del término AC.

A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	**1**
1	1	0	
1	1	1	**1**

Continuando con el término A'B, sabemos que es verdadero cuando A es falso y B es verdadero, por lo que agregamos un uno a la tabla de verdad cuando se cumplen esta condición; la tabla 4.3 muestra el resultado de este análisis.

Tabla 4.3 Análisis del término A'B.

A	B	C	Y
0	0	0	
0	0	1	
0	1	0	**1**
0	1	1	**1**
1	0	0	
1	0	1	1
1	1	0	
1	1	1	1

después de revisar las expresiones del circuito, y como resultado de que fue derivado de una asociación O (Or), las salidas restantes se rellenan con cero. La tabla 4.4 muestra el resultado final. Es importante mencionar que cuando se está buscando el resultado de una relación Y (And), se buscan los valores (de forma reciproca) que lo hacen cero a los términos involucrados, términos que por lo general son O (Or).

Tabla 4.4 Resultado del análisis de la función booleana.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Desarrollo

Dado el circuito lógico combinacional de la figura 4.4, Realizar los siguientes pasos.

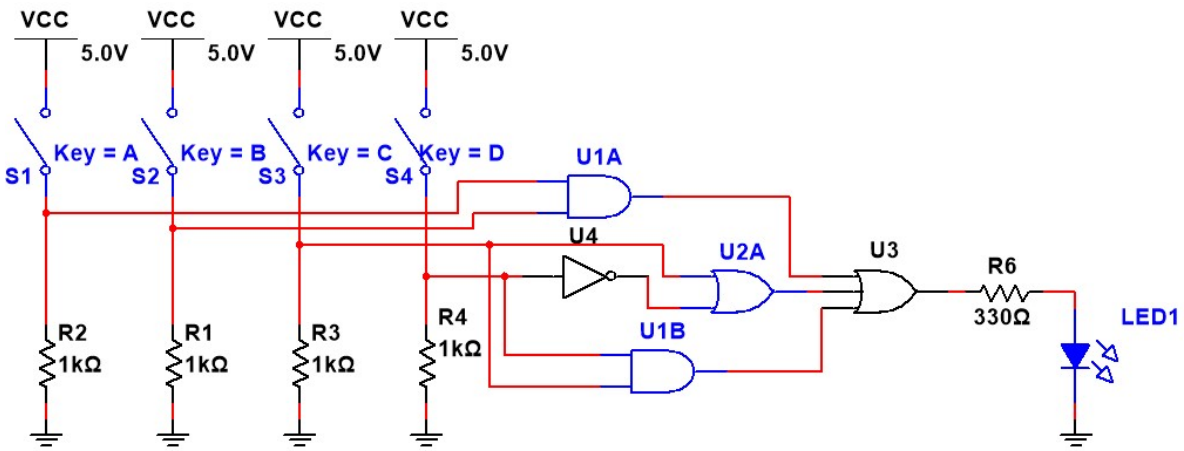


Figura 4.4 Circuito logico combinacional para su análisis.

1. Realizar el análisis del circuito para obtener la expresión booleana. Registrar y reportar al docente los pasos tomados para este análisis el día de la práctica.
2. Realizar el análisis de la expresión booleana. Registrar y reportar al docente los pasos tomados para este análisis el día de la práctica.
3. Realizar la simulación del circuito en multicines y obtenga la tabla de verdad del circuito. Compare la tabla de verdad obtenida en este pasó con la tabla de verdad obtenida en el paso anterior y de sus observaciones.
4. Arme el circuito en la placa de pruebas (protoboard) y obtenga la tabla de verdad.
5. Compare la tabla de verdad obtenida a partir del circuito armado en la placa de pruebas con las 2 tablas de verdad obtenidas anteriormente y de sus observaciones.

* Registrar toda la información de los pasos anteriores para adjuntarlo al reporte de la práctica.

Preguntas

1. Dibuje el diagrama de la implementación de la compuerta O (Or) de 3 entradas que realizó en la placa de pruebas (protoboard)
2. De las 4 compuertas Y (And) que trae el integrado 74LS08 (A, B, C y D), ¿Cuáles son las que utilizo para implementar el circuito en la placa de pruebas?
3. De las 4 compuertas O (Or) que trae el integrado 74LS32 (A, B, C y D), ¿Cuáles son las que utilizo para implementar la compuerta de 2 entradas y cuál es para la compuerta de 3 entradas del circuito en la placa de pruebas?

4. ¿Cuáles son las terminales de alimentación (Vcc y GND) de los circuitos TTL utilizados en esta práctica?
5. ¿Cuál es el significado de las palabras "de forma reciproca" en el contexto de la obtención de la tabla de verdad en la que se refiere al análisis de la compuerta O (Or) y de la compuerta Y (And)?

Referencias

1. Álgebra booleana y circuitos lógicos, https://www.monografias.com/trabajos104/algebra-booleana-y-circuitos-logicos/algebra-booleana-y-circuitos-logicos, septiembre 2022.
2. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.
3. NI Circuit Design Suite Getting Started Guide.

Práctica # 5 Creación de proyecto y simulación en Vivado

Objetivos de la práctica # 5

1. Conocer, comprender y aplicar las bases de VHDL en la creación y simulación de un proyecto en Vivado.
2. Conocer y utilizar la interfaz del juego de herramientas de la industria para el desarrollo de System on Chip (SoC) Vivado.
3. Conocer y aplicar la configuración de la tarjeta de desarrollo fpga "Basys 3" para un proyecto de un circuito lógico combinacional.
4. Conocer y comprender los pasos necesarios para simular un proyecto en Vivado.

Material y equipo

Enseguida se muestra en la tabla 5.1 los componentes y equipo a utilizar en la presente práctica, que será únicamente simulación.

Tabla 5.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado

Introducción

VIVADO DESIGN SUITE es el primer juego de herramientas de la industria para el desarrollo de System on Chip (SoC). Vivado ofrece un entorno de desarrollo para: SoC-strength, IP-centric y System-centric, con herramientas de última generación que se han diseñado desde cero para hacer frente a las dificultades de la productividad de sistemas digitales a nivel de integración e implementación. VIVADO DESIGNER está en la delantera de la productividad global, facilita el uso y capacidades de integración a nivel de sistema. VIVADO es compatible con los dispositivos de las siguientes familias: UltraScale, Virtex-7, Kintex-7, Artix-7 y Zynq-700.

Basys3 de Digilent es una placa FPGA de nivel de entrada diseñada exclusivamente para la Vivado Design Suite (figura 5.1 y tabla 5.2: Referencias en la imagen y descripción de componentes), que cuenta con la arquitectura FPGA Xilinx Artix-7. Basys3 es la nueva incorporación a la popular línea de placas FPGA de inicio de Basys. Basys3 incluye las

características estándar que ofrecen todas las placas Basys: hardware completo listo para usar, una gran colección de dispositivos de E/S en placa, todos los circuitos de apoyo FPGA y una versión libre de herramientas de desarrollo, todo a un precio rentable para los estudiantes. Debido a la migración desde la familia Spartan 3E a la clase de dispositivo Artix-7, el Basys3 ofrece un aumento sustancial de las capacidades de hardware. Con la nueva FPGA Artix se pueden encontrar un número 15 veces mayor de celdas lógicas (desde 2,160 a 33,280) y la actualización desde multiplicadores hasta porciones DSP verdaderas. También agrega más de 26 veces la cantidad de RAM.

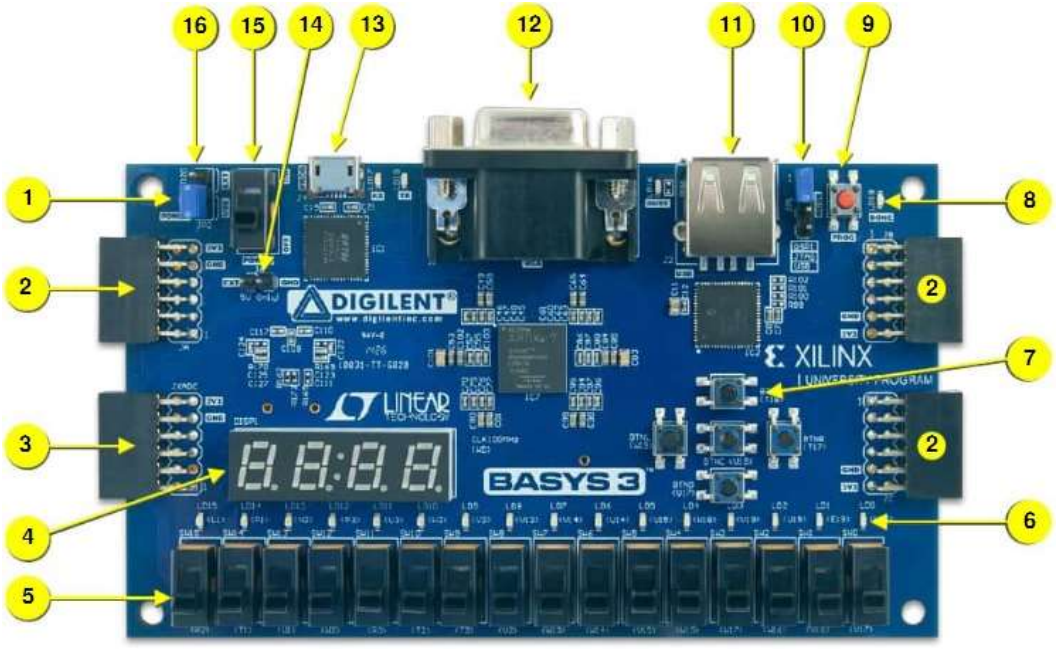


Figura 5.1 Tarjeta Basys 3 con referencias (ver tabla 5.2)

Tabla 5.2 Descripción de componentes Basys 3.

Referencia	Descripción	Referencia	Descripción
1	LED Prendido/correcto	9	Reinicia configuración FPGA
2	Conectores tipo *Pmod*	10	Selector modo de programar
3	Conector *Pmod* señal análoga	11	Conector USB host
4	Display 4 dígitos / 7 segmentos	12	Conector VGA
5	Interruptores deslizables (16)	13	Puerto USB UART/JTAG
6	LEDs (16)	14	Entrada alimentación externa
7	Interruptor de botón (5)	15	Interruptor de encendido
8	Termina programación FPGA	16	Selector de alimentación

Características Basys 3:

- 33,280 celdas lógicas en 5200 porciones (cada porción contiene cuatro LUT de 6 entradas y 8 *flip-flops*)

- 1,800 Kbits de RAM de bloqueo rápido
- Cinco CMT, cada uno con una fase de bucle cerrado (PLL)
- 90 porciones DSP
- Velocidades de reloj interno superiores a 450 MHz
- Convertidor de analógico a digital (XADC) en chip
- 16 interruptores de usuario
- 16 LEDs de usuario
- 5 pulsadores de usuario
- Pantalla de 4 dígitos y 7 segmentos
- 4 conectores Pmod
    - 3 Pmod estándar de 12 pines
    - 1 Pmod de uso doble para señal XADC o estándar
- Salida VGA de 12 bits
- Puente de USB a UART
- Flash serial
- Puerto USB a JTAG Digilent para programación y comunicación FPGA
- Host USB HID para ratón, teclados y memoria extraíble

VHDL es un lenguaje de especificación definido por el IEEE (Institute of Electrical and Electronics Engineers) (ANSI/IEEE 1076-1993) utilizado para describir circuitos digitales y para la automatización de diseño electrónico, a estos lenguajes se les suele llama lenguajes de descripción de hardware. VHDL es acrónimo proveniente de la combinación de dos acrónimos: VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language). Aunque puede ser usado de forma general para describir cualquier circuito digital se usa principalmente para programar PLD (Programable Logic Device - Dispositivo Lógico Programable), FPGA (Field Programmable Gate Array), ASIC y similares. Originalmente, el lenguaje VHDL fue desarrollado por el departamento de defensa de los Estados Unidos a inicios de los años 80 basado en el lenguaje de programación ADA con el fin de simular circuitos eléctricos digitales. Posteriormente se desarrollaron herramientas de síntesis e implementación en hardware a partir de los archivos VHD.

Análisis previo

La introducción del circuito en el archivo de diseño VHDL consta de 2 partes principalmente, la entidad del circuito y la arquitectura del circuito. La entidad del circuito describe las entradas y salidas, así como el tipo de datos que son. La arquitectura del circuito describe el comportamiento lógico asociado a la entidad del circuito, tomando en cuenta las entradas y salidas de esta. Para la realización de la entidad y la arquitectura del circuito

tomaremos como referencia la función booleana obtenida en la práctica anterior y mostrada en la figura 5.2 y le asignaremos el nombre de "*MiCircuito*" a la entidad.

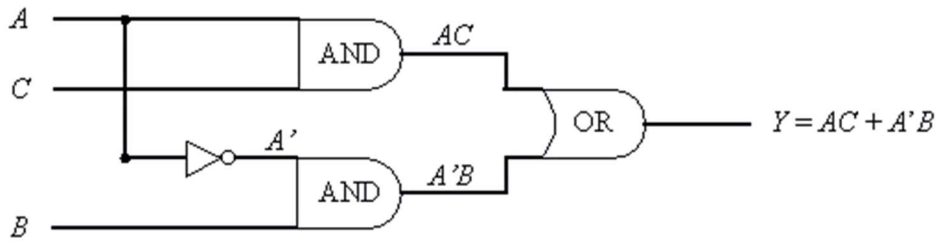


Figura 5.2 Circuito de referencia y su función booleana.

Partiendo de la función booleana, se determinan 3 entradas y una salida que son A, B, C y Y respectivamente, y que se pueden representar por un bit cada una de ellas, siendo en VHDL del tipo std_logic, por lo que la entidad del circuito quedaría

```
entity MiCircuito is
 Port (
       A: in std_logic;
       B: in std_logic;
       C: in std_logic;
       Y: out std_logic);
end MiCircuito;
```

Es importante saber que VHDL no es sensible a las mayúsculas, por lo que puedes utilizar mayúsculas y minúsculas de forma indiferente. La arquitectura debe de asociarse con la entidad por medio del nombre que se le asignó la entidad ("*MiCircuito*"), y la función se debe de expresar utilizando los nombres de los operadores lógicos en vez de la simbología mostrada en la figura 5.2, quedando de la siguiente forma la conversión:

```
Y <= (A and C) or ((not A) and B);
```

Como se puede observar, la sintaxis para la asignación de la operación lógica es utilizando el símbolo "<=" y terminando la expresión con punto y coma (;). Es recomendable utilizar paréntesis para asegurar la asociación de los operadores lógicos de acuerdo con la función lógica original. De esta forma, la arquitectura del circuito será

```
architecture Behavioral of MiCircuito is
begin
        Y <= (A and C) or ((not A) and B);
end Behavioral;
```

Una vez terminado el archivo de diseño VHDL, procedemos con la simulación del circuito, para lo que creamos un archivo de simulación o *test bench*. En del archivo de simulación, dentro de la arquitectura y antes de la sentencia "`begin`", habrá que definir o "agregar" (1) el contenido de la entidad que se realizó para el archivo de diseño y (2) las señales inicializadas a cero (`:= '0'`); en el componente cambiar el identificador de "`entity`" por "`component`" y el final de la entidad con "`end component;`"; estos cambios se muestran, remarcados con comentarios, enseguida tomando como referencia la entidad anterior:

```
component MiCircuito is -- El original era: entity MiCircuito is
 Port (
       A: in std_logic;
       B: in std_logic;
       C: in std_logic;
       Y: out std_logic);
end component; -- El original era: end MiCircuito
```

```
signal A_s: std_logic := '0'; --Señal agregada e inicializada a cero
signal B_s: std_logic := '0'; --Señal agregada e inicializada a cero
signal C_s: std_logic := '0'; --Señal agregada e inicializada a cero
signal Y_s: std_logic := '0'; --Señal agregada e inicializada a cero
```

Y en la sección del código de la arquitectura, después de la sentencia "`begin`", agregamos el código que dará la pauta a los cambios en las entradas para la simulación en 2 etapas, la primera etapa es la instanciación (o creación) del componente definido previamente (antecedido por una etiqueta `MiComponente:`), asociando las señales del componente con las señales definidas en el paso anterior (la razón de la definición de las señales es poder manipular las entradas y salidas del componente, el cual funciona como una unidad que ha

sido definido en otro archivo y se agrega en el presente archivo), quedando el código qué se agregará después de la sentencia "begin" de la siguiente forma:

```
MiComponente: principal port map(
        A => A_S,
        B => B_s,
        C => C_s,
        Y => Y_s);
```

Continuando en la arquitectura y para agregar los cambios en las variables de entrada en forma temporizada, lo que definimos como un proceso etiquetado y le damos la instrucción de iniciar (*label: process begin*), que, para nuestro caso, la etiqueta se nombrará entrada_A y se sigue de dos puntos (todas las etiquetas se siguen de ":"). Dentro del proceso, iniciamos con un retardo de 10 ns (*wait for 10 ns;*), generando cambios en la entrada A de cero a uno y de uno a cero cada 10 ns con el código "*A_s <= not A_s;*".

```
entrada_A: process
           begin
                wait for 10 ns;
                A_s <= not A_s;
           end process;
```

Aplicamos el mismo proceso a las demás entradas (B y C), cambiando los tiempos de espera para generar cambios coordinados en las entradas de tal forma que generen las diferentes combinaciones posibles de las 3 variables. El código completo (A, B y C) de la simulación resulta:

```
entrada_A: process
           begin
                wait for 10 ns;
                A_s <= not A_s;
           end process;
entrada_B: process
           begin
                wait for 20 ns; --El tiempo se duplica para generar
--los dos cambios de la variable anterior (de 0 a 1 y de 1 a 0)
                B_s <= not B_s;
           end process;
```

```
entrada_C: process
           begin
                wait for 40 ns; --El tiempo se duplica para generar
--los dos cambios de la variable anterior (de 0 a 1 y de 1 a 0)
                C_s <= not C_s;
           end process;
```

Para introducir comentarios en el código anterior, se emplea doble guion (--). La forma de onda generada se muestra en la figura 5.3, donde la primera forma de onda corresponde a la variable A, la segunda corresponde a la variable B y la tercera forma de onda corresponde a la variable C.

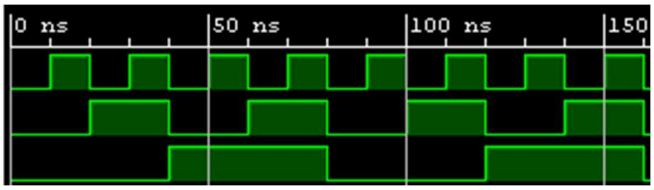


Figura 5.3 Formas de onda de (señales A, B y C) generadas para la simulación.

Desarrollo

Para el proyecto y la simulación en Vivado utilizaremos el circuito lógico combinacional mostrado en la figura 5.4, en él se indica el nombre de las variables de entrada (A, B, C, D y E). Los pasos por seguir para la realización del proyecto se dan enseguida.

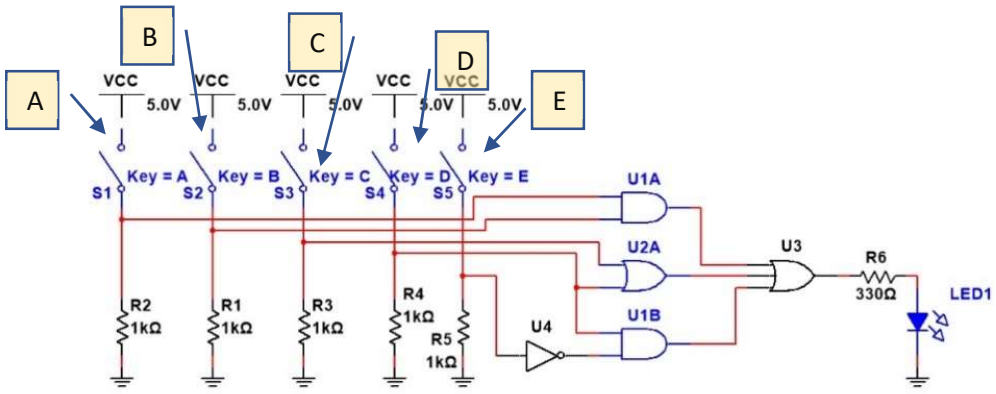


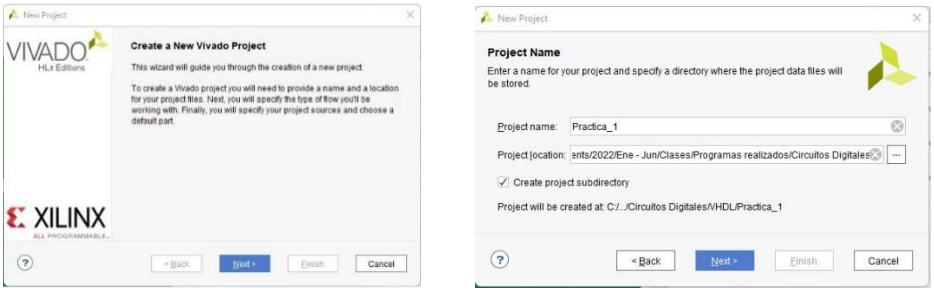
Figura 5.4 Circuito logico combinacional para su análisis.

1. Abrir el programa Vivado, que lo podrás distinguir por su icono: 

2. Una vez iniciado el programa, selecciona en el menú rápido crear proyecto (figura 5.5).

Figura 5.5 Ventana de inicio de Vivado y la opción "Crear proyecto".

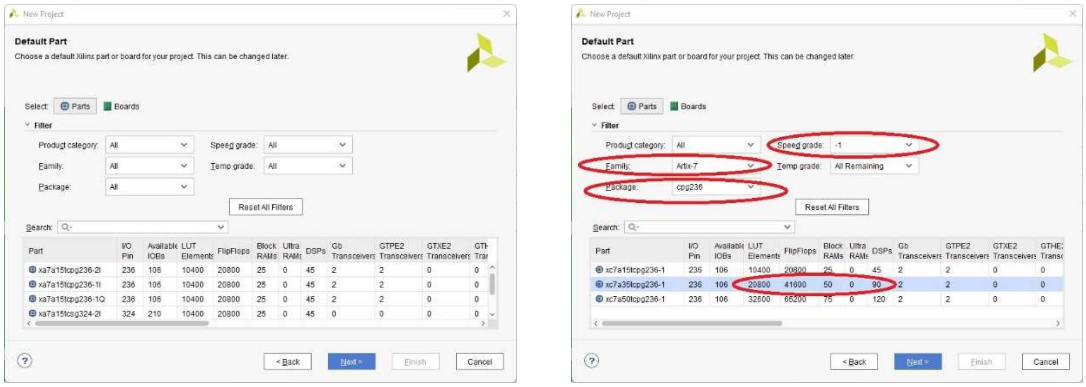
3. Se abrirá una ventana en la que se transforma que se creará un proyecto (figura 5.6 a), dar clic en continuar y aparecerá una ventana donde hay que colocar el nombre del proyecto, en este caso *Practica_1* (figura 5.6 b), y dar clic en continuar.



a) *b)*

Figura 5.6 Ventana de creación de proyecto y de asignación de nombre al proyecto.

4. La siguiente ventana es para definir el dispositivo de la tarjeta (figura 5.7 a), el cual corresponde a la familia *Artix-7*, de empaquetado *cpg236* con grado de velocidad *-1*. Esto dará 3 opciones en la lista de partes, selecciona la opción de 20800 LUTs (figura 5.7 b) y dar clic en siguiente con lo que aparecerá una ventana de resumen (imagen no mostrada) a la cual hay que darle clic en finalizar.



a) *b)*

Figura 5.7 Ventana de Selección del dispositivo.

5. Terminados los pasos anteriores, se abre la interfaz de ambiente integrado (IDE) de Vivado (figura 5.8 anotación 1), en la que hay que darle clic en agregar fuentes (Add sources) lo que abrirá la ventana correspondiente (figura 5.8 anotación 2). Asegurarse que la opción de agregar o crear fuentes de diseño está seleccionada y dar clic en siguiente.

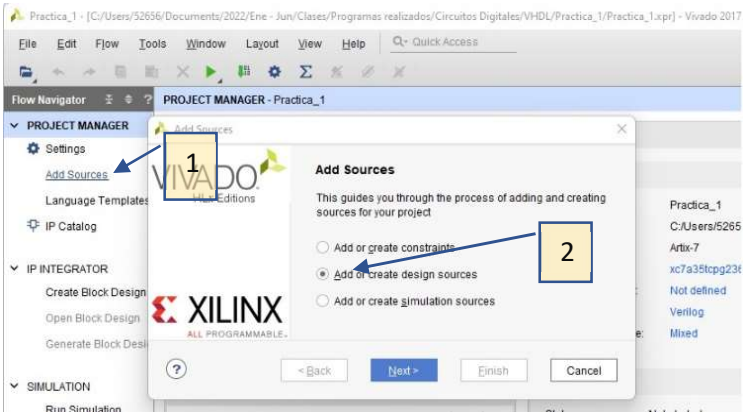


Figura 5.8 Ventana para agregar fuentes de diseño.

6. En la siguiente ventana, seleccionar la opción crear archivo (figura 5.9 anotación 1)

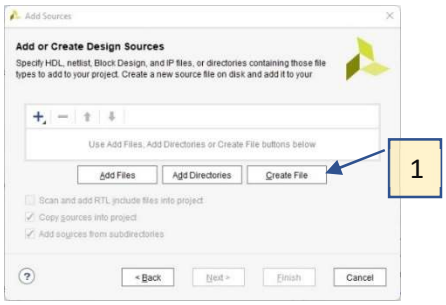


Figura 5.9 Ventana para agregar archivos fuente.

7. Se abrirá la ventana de crear archivo fuente, asegurar que el tipo de archivo es VHDL y dale el nombre de Practica_1 como se muestra en la figura 5.10. Dar clic en "OK".

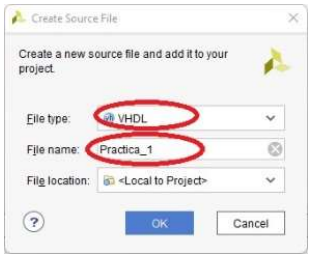
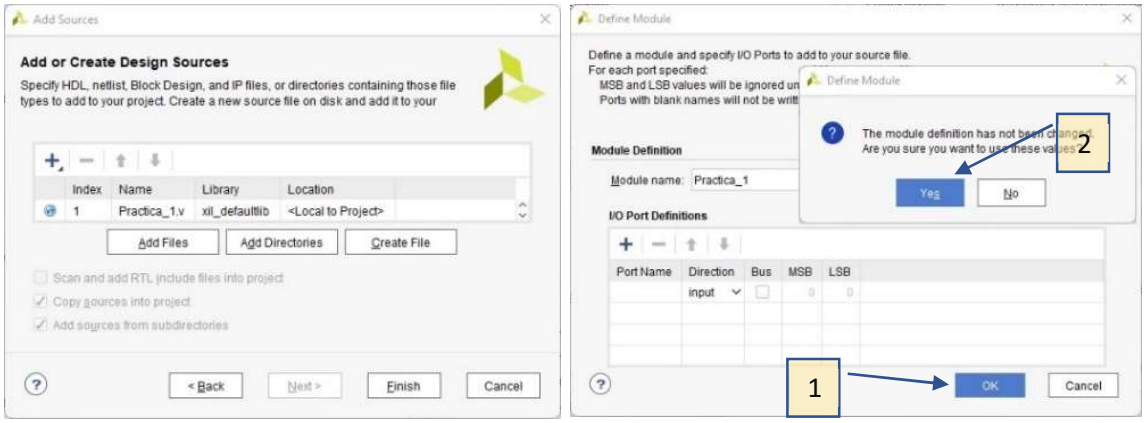


Figura 5.10 Creación del archivo fuente.

8. Al cerrarse la ventana anterior, regresamos a la ventana de agregar fuentes, obsérvese que ya tiene agregado el archivo Practica_1 (figura 5.11 a). Dar clic en el botón "OK" (figura 5.11 anotación 1), abrirá una ventana informando que hubo cambios, presionar el botón "Yes" (figura 5.11 anotación 2).



a) *b)*

Figura 5.11 Finalización de agregar archivo fuente.

9. De regreso a la IDE de Vivado, se observa que en la ventana fuentes aparece el archivo recién agregado *Practica_1*, como lo indica la figura 5.12.

10. Para iniciar la edición del archivo de diseño, dar 2 clics sobre la referencia al archivo *Practica_1*, dentro de la ventana fuentes, como indica la figura 5.12 y se abrirá el archivo para edición como se muestra en la figura 5.13.

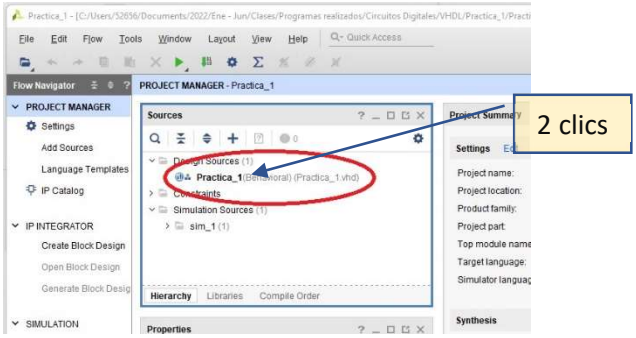


Figura 5.12 Archivo Practica_1 agregado en ventana fuentes.

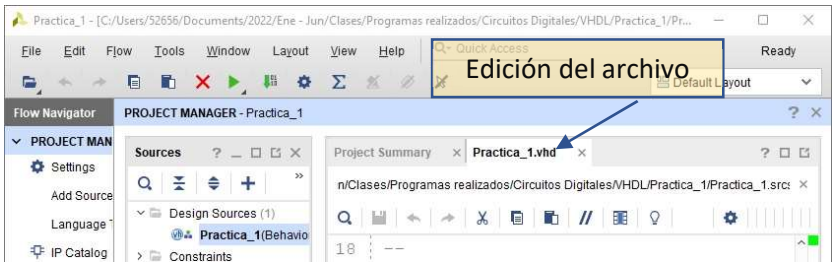
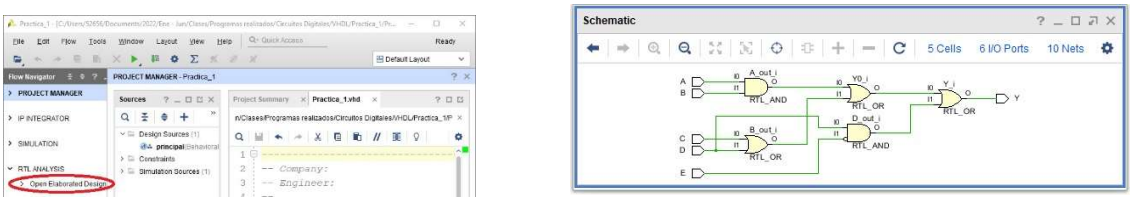


Figura 5.13 Ventana de edición del archivo de diseño.

11. Realiza la entidad del circuito mostrado en la figura 5.4 conforme se indicó en el apartado de "Análisis previo" y agrégala en la entidad denominada *Practica_1* del archivo de diseño que se encuentra aproximadamente en la línea 34.

12. Realiza la arquitectura del circuito mostrado en la figura 5.4 conforme se indicó en el apartado de "Análisis previo" y agrégala en la arquitectura *Behavoiral* de *Practica_1* del archivo de diseño que se encuentra aproximadamente (después de haber modificado la entidad denominada *Practica_1*) en la línea 44. Se recomienda utilizar señales como lo indica el vídeo https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos.

13. Para visualizar el circuito, da clic en Lenguaje de transferencia de registro RTL Abrir el diseño elaborado "*RTL Open Elaborated Circuit*" (figura 5.14 a) y esto abrirá una ventana con el esquemático del diseño introducido en la arquitectura asociada a la entidad del circuito codificado, y con él, podrás verificar que tu diseño coincide con el circuito esperado (figura 5.14 b). Asegúrate de que el diseño es una copia fiel de lo esperado (y parte, entre otras cosas, del reporte de práctica) para continuar en el siguiente paso.



a) *b)*

Figura 5.14 Indicaciones para abrir el esquemático del diseño y la ventana del esquemático.

14. Los siguientes pasos corresponden a la parte de la simulación del proyecto diseñado. Comenzamos por agregar un archivo de simulación dando clic en agregar fuentes como se indica en la figura 5.15 nota 1, lo cual abre una ventana con tres opciones, asegurarse de que se escoge la opción de agregar una fuente de simulación como se indica en la figura 5.15 nota 2 dando lugar a la ventana de agregar fuente.

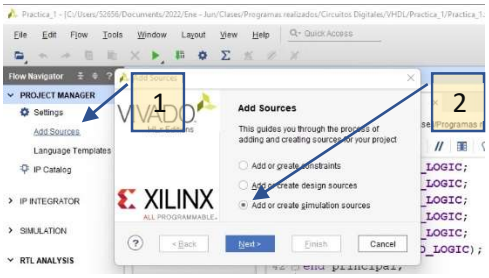
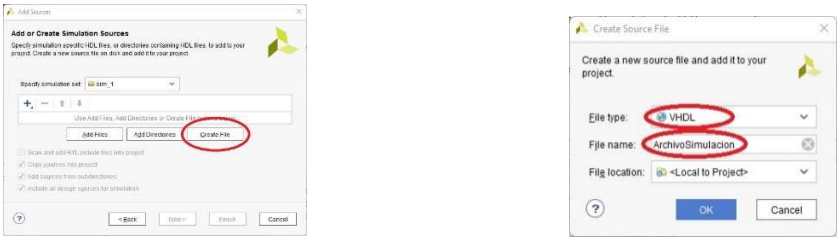


Figura 5.15 Pasos para agregar archivo de simulación.

15. En la ventana agregar fuente seleccionar crear archivo como se indica en la figura 5.16 a) con lo que se abrirá la ventana de crear archivo fuente y deberá especificarse el tipo del

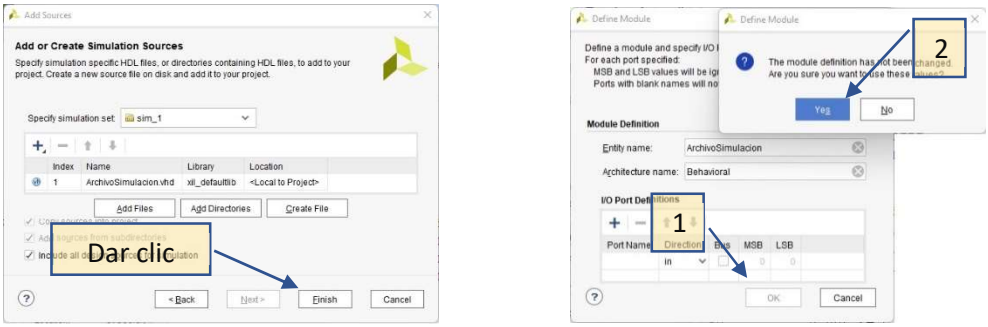
archivo VHDL y como nombre sugerido *ArchivoSimulacion* como se indica en la figura 5.16 b); dar clic en el botón "*OK*".



a) *b)*

Figura 5.16 Ventanas para agregar archivo de simulación.

16. Cerrar la ventana de agregar fuentes dando clic en el botón finalizar como se indica en la figura 5.17 a) se abrirá una ventana para definir módulos, dar clic en "*OK*" y en la siguiente ventana darle clic en "*Yes*" como se muestra en la figura 5.17 b).



a) *b)*

Figura 5.17 Clics en ventanas para finalizar agregar fuente.

17. Volviendo a la IDE de Vivado, para realizar la edición del archivo de simulación, dar clic en el nombre del archivo "*ArchivoSimulacion*" que se encuentra en la ventana fuentes y observar que en la ventana de edición estará el archivo como lo muestra la figura 5.18.

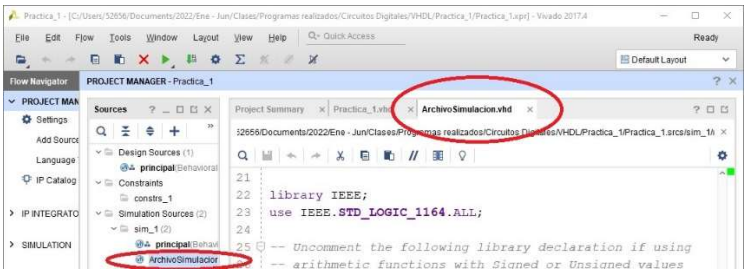


Figura 5.18 Apertura de la edición del archivo de simulación.

18. Convierte la entidad del paso (11.) a componente como se indica en la sección "Análisis Previo" y agrégalo a la edición del archivo "*ArchivoSimulación*" aproximadamente en el renglón 40, dentro de la arquitectura de "*ArchivoSimulación*" antes de la sentencia "`begin`".

19. En la misma sección de arquitectura del archivo "*ArchivoSimulación*", después de la sentencia "`begin`", agregar la instanciación (o creación) del módulo asociando sus entradas y salidas con las señales definidas anteriormente como se indica en la sección "Análisis Previo" y seguido agregar los cambios en las variables de entradas en forma temporizada, también he explicado en la sección "Análisis Previo". Observa como el componente ha sido instanciado abajo del archivo de simulación en la figura 5. 19.

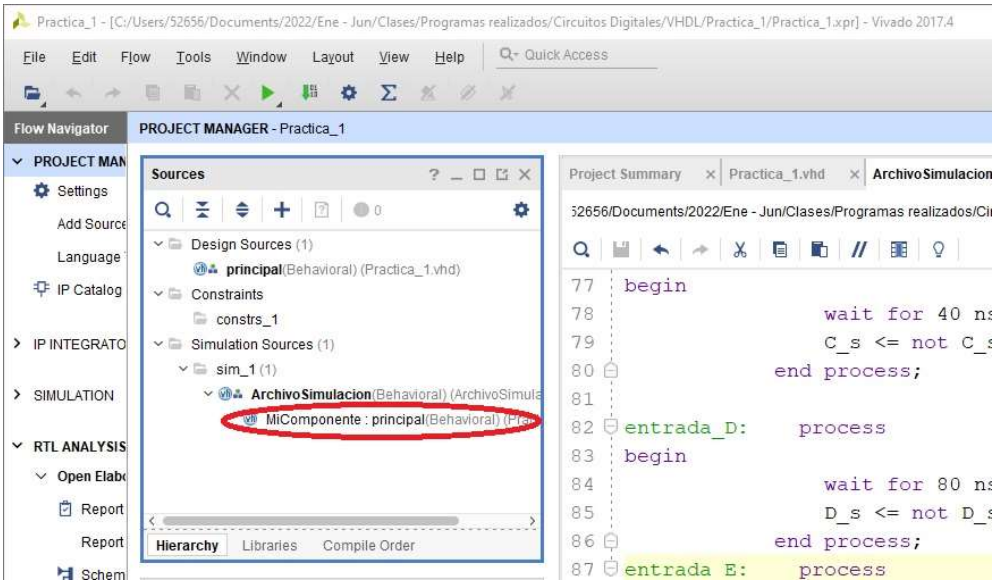


Figura 5.19 Archivo de simulación y la instanciación del componente.

20. Para realizar la simulación, dar clic al submenú "Correr simulación" del menú simulación como se muestra en la figura 5.20 y seleccionar en el menú desplegable "Correr simulación del comportamiento".

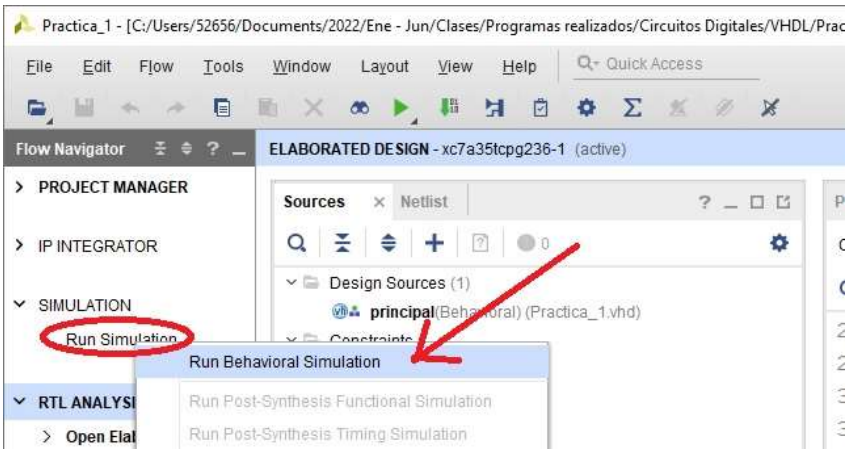


Figura 5.20 Submenú para realizar la simulación.

21. Al terminar la corrida de simulación se abren dos ventanas como se muestra en la figura 5. 21. El diagrama de tiempo nos aprecia por lo que hay que cambiar el intervalo de tiempo.

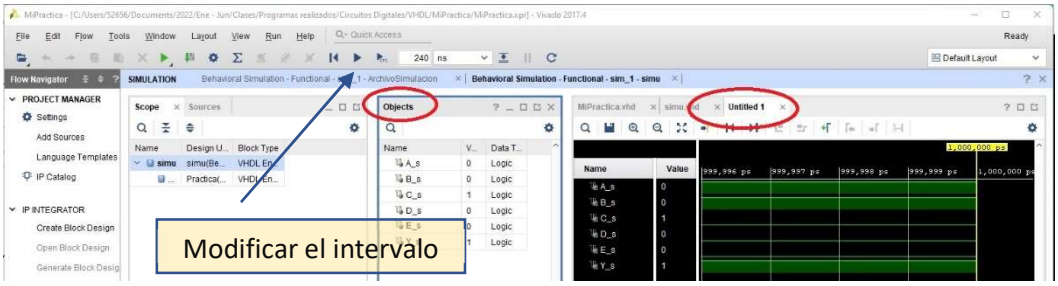


Figura 5.21 Ventanas de la simulación e intervalo de tiempo.

22. Cambiar el intervalo del tiempo, que para nuestra práctica será de 320 ns, modificando en la Barra de configuración de tiempo el valor tiempo  y damos clic en retroceder () seguido de el botón correr período () y por último presionamos el botón "Ajustar zoom" () este último botón se encuentra en la ventana del diagrama de tiempo y se indica en la figura 5.22 junto con el diagrama de tiempo del circuito lógico combinacional diseñado.

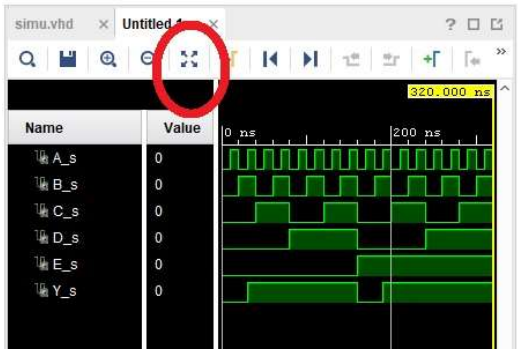


Figura 5.22 Ventana de diagrama de tiempo y el botón "Ajustar zoom".

23. En la ventana objetos podemos ver los valores que toman las variables al ir cambiando la posición del cursor (que se representa por una línea amarilla con una cabecera indicando el tiempo) al dar clic sobre el diagrama de tiempo. En la figura 5.23 se muestra el cursor en el tiempo 103.489 ns y en la ventana objetos nos muestran los valores de las variables de entrada y de salida.

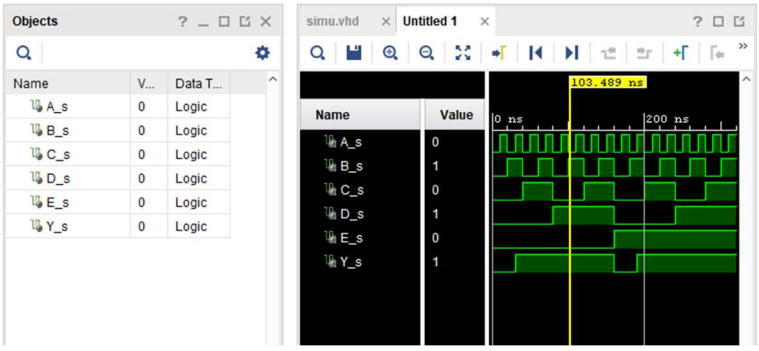


Figura 5.23 Selección de tiempo con el cursor en el diagrama de tiempos.

Preguntas

1. ¿Qué es una IDE?
2. ¿Qué es un componente en VHDL?
3. ¿Cuáles son los parámetros de configuración de Vivado para realizar un diseño enfocado a Basys 3?
4. ¿Qué significa RTL y que tipo de tecnología es?
5. ¿Qué significa SoC y que tipo de tecnología es?

Referencias

1. Llanos, Juan; "Vivado Tutorial 1 Crear un proyecto", https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; "Vivado Tutorial 2 Simulación", https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.

Práctica # 6 Manejo de equipo para señales digitales

Objetivos de la práctica # 6

1. Conocer y configurar un generador de funciones.
2. Conocer y operar un osciloscopio en la medición de señales digitales.
3. Conocer y realizar la medición de amplitud tiempo y frecuencia de señales digitales.

Material y equipo

Enseguida se muestra en la tabla 6.1 los componentes y equipo a utilizar en la presente práctica, que será únicamente simulación.

Tabla 6.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Multisim

Introducción

La señal digital es un tipo de señal en que cada signo que codifica el contenido de esta puede ser analizado en término de algunas magnitudes que representan valores discretos, en lugar de valores dentro de un cierto rango. Ejemplo, el interruptor de la luz solo puede tomar dos valores o estados: abierto o cerrado, o la misma lámpara: encendida o apagada (véase circuito de conmutación). Esto no significa que la señal físicamente sea discreta ya que los campos electromagnéticos suelen ser continuos, sino que en general existe una forma de discretizarla unívocamente. La señal digital es un tipo de señal en que cada signo que codifica el contenido de esta puede ser analizado en término de algunas magnitudes que representan valores discretos, en lugar de valores dentro de un cierto rango.

Los sistemas digitales, como por ejemplo la computadora, usan la lógica de dos estados representados por dos niveles de tensión eléctrica, uno alto, H y otro bajo, L (de *High* y *Low*, respectivamente, en inglés). Por abstracción, dichos estados se sustituyen por ceros y unos, lo que facilita la aplicación de la lógica y la aritmética binaria; la representación del nivel alto por 1 y del nivel bajo por 0 se muestra en la figura 6.1. Si el nivel alto se representa por 1 y el bajo por 0, se habla de lógica positiva y en caso contrario de lógica negativa.

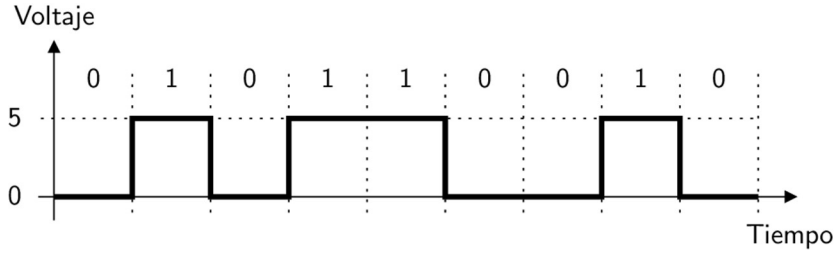


Figura 6.1 Señal digital (0 a 5 V) cambiando en el tiempo.

Cabe mencionar que, además de los niveles, en una señal digital están las transiciones de alto a bajo y de bajo a alto, denominadas flanco de bajada y de subida, respectivamente. En la figura 6.2 se muestra una señal digital donde se identifican los niveles y los flancos.

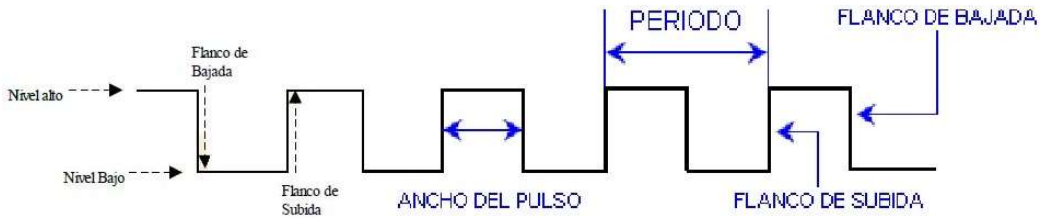


Figura 6.2 Señal digital en la que se indican varios parámetros.

Otros parámetros importantes de la señal digital son el ancho de pulso y el periodo; el periodo es el tiempo en que una señal periódica repite su forma de onda, a partir de un punto de referencia de la señal, la frecuencia de la señal es igual al inverso del período y está dado en Hertz (Hz). es muy común ver la forma de onda en una forma ideal, es decir, completamente cuadrada, sin embargo, la forma de onda en la realidad dista de ser idealmente cuadrada. En la figura 6.3 a) observamos una forma de onda en la que los flancos de subida y bajada tienen cierta inclinación (3, 4) y en la figura 6.3 b) como es una forma de onda digital con ruido, ambos casos asemejándose a la realidad.

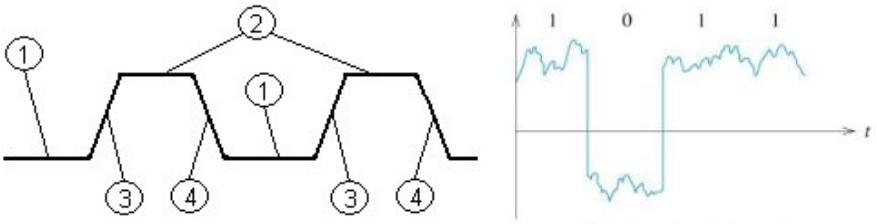


Figura 6.3 Formas de onda acercadas a la realidad.

Cabe recalcar que las ondas digitales pueden definirse como periódicas o no periódicas, una onda periódica es aquella que se repite cada cierto tiempo T (T es el tiempo del periodo, TH es tiempo en alto y TL es tiempo en bajo, así T = TH + TL) y una onda no periódica carece del patrón de repetición. En la figura 6.4 se muestran dos formas de onda, periódica y no

periódica, así como la definición del tiempo en alto TH, el tiempo en bajo TL, y su suma, el tiempo T del período.

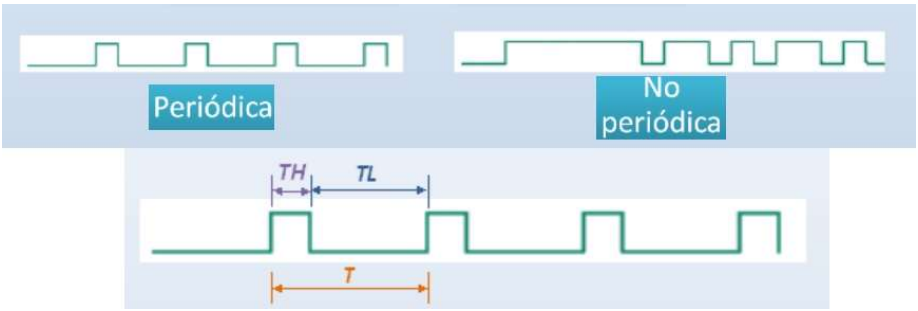


Figura 6.4 Forma digital periódica y no periódica.

Como se había mencionado anteriormente, la frecuencia es el inverso del período y viceversa.

$$f = \frac{1}{T}\,\text{Hz}, \quad T = \frac{1}{f}\,\text{segundos}$$

En los sistemas digitales la información binaria se representa por medio de secuencias de bits. Cada bit dentro de una secuencia tiene un intervalo de tiempo definido, denominado periodo de bit. En la mayoría de los sistemas digitales, las señales se sincronizan por medio de una señal de temporización básica denominada reloj. El reloj es una señal periódica en la que cada intervalo entre impulsos (el periodo) equivale a la duración del bit. El cambio de nivel de una señal se puede generar durante el flanco de subida, flanco de bajada, o al nivel alto o bajo, de la señal del reloj. En la figura 6.5 se muestra un diagrama de tiempo, en el que se observa una señal digital de entrada (que lo general son señales no periódicas), que para ser leída dentro del sistema digital se sincroniza mediante una señal de reloj (las señales de reloj son señales periódicas) que activan la lectura del sistema en el flanco de subida. La tercera señal muestra la lectura de la señal digital de entrada en sus respectivos valores de ceros y unos.

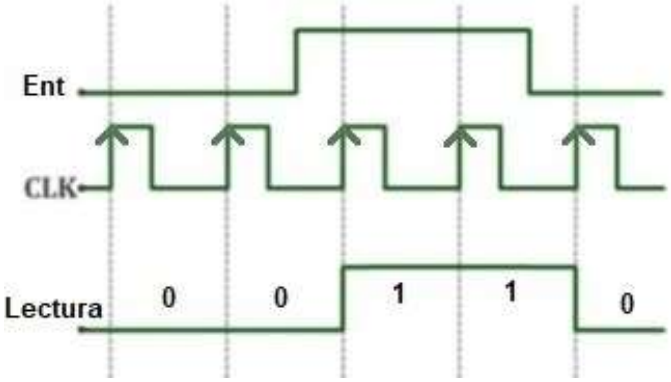


Figura 6.5 Lectura de una señal digital (no periódica) que entra a un sistema digital, coordinado por una señal periódica de reloj CLK.

Un diagrama de tiempo es una gráfica de formas de onda digitales que muestra la relación temporal entre varias señales, y cómo varía cada señal en relación con las demás. Un diagrama de tiempo puede contener cualquier número de señales relacionadas entre sí. Examinando un diagrama de tiempos, se puede determinar los estados, nivel alto o nivel bajo, de cada una de las señales en cualquier instante de tiempo especificado, y el instante exacto en que cualquiera de las señales cambia de estado con respecto a las restantes. El propósito primario del diagrama de tiempo es mostrar los cambios en el estado o la condición de una línea de una señal digital a lo largo del tiempo lineal.

Cuando se están estudiando las señales digitales, podemos utilizar un generador de funciones para proporcionar señales bien controladas, como lo es un tren de pulsos o señales rectangulares o sinusoidales, con las cuales dar entrada a los sistemas digitales que se estén estudiando. Por el contrario, si se desean conocer las señales asociadas a un circuito digital, el osciloscopio es el instrumento con el cual podemos visualizar tales señales, tanto en forma como en tiempo. Estos dos instrumentos (el osciloscopio y el generador de funciones) están disponibles de forma virtual en Multisim, comenzaremos viendo el generador de funciones. Para allá agregar el generador de Funciones al área de trabajo de Multisim, damos clic al ícono del generador de funciones que se encuentra en la Barra de instrumentos  . En la figura 6.6 se observa la ubicación del icono del generador de funciones, el instrumento en el área de trabajo y la carátula de este.

Figura 6.6 Ícono, caratula y botón del generador de funciones en Multisim.

Para encender el generador de funciones, el botón de encendido está del lado izquierdo de la carátula; una vez encendido el generador de funciones, podemos escoger el tipo de señal a generar por medio de los botones de selección de tipo de señal, agrupados en el conjunto "Function", que se muestran en la figura 6.7 en conjunto con las señales que puede generar, que en orden de los botones mostrados son: funciones senoidales, cuadrada triangular diente de sierra ruido o arbitraria y respectivamente con la numeración 1 a 5.

Figura 6.7 Botones para la selección del tipo de señal y numeración 1 a 5.

Para seleccionar la frecuencia amplitud y desplazamiento de la señal, contamos con el grupo "Modify" mostrado en la figura 6.8, el cual también ofrece la numeración 6 a 8.

Figura 6.8 Grupo "Modify" y numeración 6 a 8.

En la figura 6.9 se muestran los botones para la selección de disparo, el manejo de los Datos hacia la memoria y el botón "Enter Number", con el que se indica teclear directamente el valor numérico de la amplitud, frecuencia o desplazamiento.

Figura 6.9 Botones para selección de disparo, memoria e introducción numérica.

Finalmente, con el generador de señales, se muestran los botones "*Enter*" (introduce el valor seleccionado) y "*Shift*" (selecciona el valor superior del botón) y un conjunto de flechas que dan desplazamiento en el display, todos ellos mostrados en la figura 6.10.

Figura 6.10 Botones para el manejo del menú.

Un osciloscopio puede tener dos o cuatro canales de entrada, por lo que se pueden visualizar dos o cuatro señales simultáneamente, dependiendo de las características del instrumento. En la figura 6.11 se muestra la carátula del osciloscopio Tecktronix TDS2024 de cuatro canales, el cual está disponible en Multisim, asimismo se muestra el botón para insertar el multímetro Tektronix TDS 2024 (el botón tachado en rojo corresponde al osciloscopio Agilent, evitar seleccionarlo) y el icono del osciloscopio en el área de trabajo.

Figura 6.11 Osciloscopio Tektronix TDS 2024 insertado en el área de trabajo Multisim.

Al encender el osciloscopio, la pantalla mostrará una serie de divisiones horizontales y verticales, las cuales corresponden a las divisiones del tiempo y de voltaje respectivamente. De igual forma se muestra el tiempo y el voltaje por cada división. En la figura 6.12 se muestran los atributos mencionados anteriormente.

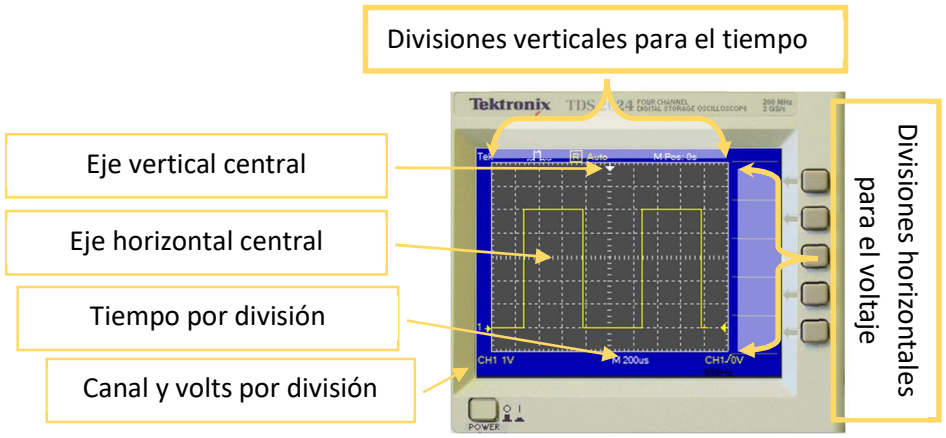


Figura 6.12 Descripción de elementos de la pantalla del osciloscopio.

En la figura 6.12 se muestra únicamente la señal del canal 1 (una señal cuadrada), con las divisiones verticales para el tiempo en la que se pueden apreciar 10 espacios temporales y como lo indica el tiempo por división, cada uno son de 200 milisegundos (200ms), da un total de 2 segundos de la señal mostrados en la pantalla. Siguiendo con la figura 6.12, las divisiones horizontales nos permiten apreciar 8 espacios de magnitud en voltaje, siendo cada división de un volt, por lo que podemos llegar a apreciar una magnitud total de hasta 8 V; en este caso la señal cuadrada tiene una magnitud de 5 V al abarcar 5 cuadros verticalmente (los cuadros verticales se dan por las divisiones horizontales).

Los parámetros del osciloscopio se pueden modificar gracias a los controles que se dividen en los siguientes grupos:

Grupo VERTICAL. Este grupo se muestra en la figura 6.13 con los controles de los cuatro canales de los que dispone el osciloscopio y en él encontramos las perillas de VOLTS/DIV y las perillas de POSITION Para modificar respectivamente la cantidad de volts por división y la posición vertical de la señal correspondiente a cada canal; también en este grupo se muestra el botón en diferentes colores y con la leyenda "CH X" para seleccionar la aparición del canal X en pantalla, además de una serie de menús relacionados con los mismos canales y con operaciones matemáticas entre las señales del canal 1 y el canal 2. En la parte inferior figura 6.13 se muestran los conectores de entrada de las señales y en la esquina inferior izquierda se puede observar las terminales que proveen de una señal de prueba de 5 V a 1 kHz (señal que se muestra en la figura 16.12).

Figura 6.13 Grupo VERTICAL y sus respectivas perillas de los cuatro canales, además de los conectores de entrada en la parte inferior.

Grupo HORIZONTAL. En él (figura 6.14) se puede modificar el tiempo por división con la perilla SEC/DIV, así como la posición/desplazamiento horizontal de las señales mostradas en pantalla.

Figura 6.14 Grupo HORIZONTAL para controlar el tiempo por división y desplazamiento de las señales en pantalla.

Grupo TRIGGER. En la figura 6.15 se muestra el control de disparo, el que permite la sincronización de la señal de barrido horizontal y la señal de entrada, permitiendo la presentación en pantalla de la señal de entrada estática, por lo que no se visualizarán

desviaciones en pantalla. El nivel de disparo se ajusta hasta que la señal en la pantalla se estabiliza.

Figura 6.15 Grupo TRIGGER para sincronización.

Finalmente, la figura 6.16 muestra la parte superior de los controles, Donde encontramos diversas funcionalidades del osciloscopio; una de ellas es el AUTO SET, que de forma automática ajusta las escalas para visualizar la señal. La función MEASURE muestra el menú de mediciones automatizadas. En el mando de canal vertical se localizan los botones para seleccionar la escala vertical de cada canal, así como los botones de posición vertical. El botón CURSOR muestra el Menú de Cursores. Los cursores se mantienen visibles después de que se deja el menú, pero ya no se pueden ajustar, a menos que se elija la opción OFF. Finalmente, el botón del extremo derecho (RUN/STOP) detiene o realiza la lectura continua de las señales para ser mostradas en la pantalla; si la señal se está mostrando en forma continua en la pantalla y se presiona este botón, la señal aparecerá en forma estática y requerirá oprimir una vez más el botón para que se realice la lectura continua de la señal.

Figura 6.16 Botones de menús y control de desplegado en pantalla de las señales.

Análisis previo

Investigar el efecto de girar las perillas VOLTS/DIV, POSITION, SEC/DIV, así como el efecto de presionar los botones CH 1, CH 2, AUTO SET, RUN/STOP cuando se tiene conectada la señal de prueba de 5 V a 1 kHz en los conectores de entrada CH 1 y CH 2 como se indica en la figura 6.17.

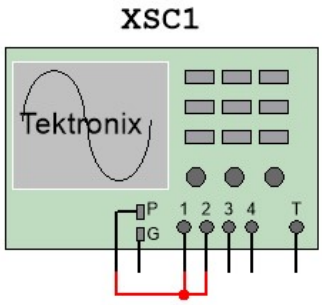


Figura 6.17 Conexión de la señal de prueba a los canales CH 1 y CH 2.

Desarrollo

Primera parte. Uso de generador y osciloscopio en el Multisim. (Tomado del Manual de Prácticas, "Práctica #5", de la Dra. Maribel Gómez Franco).

1. Calibración.

1.1. En la hoja de trabajo coloque el osciloscopio de Tektronix, en el menú **Simulate/Instrumens/**.

1.2. Conecte el canal 1 a la terminal P, como se indica en la figura 6.18.

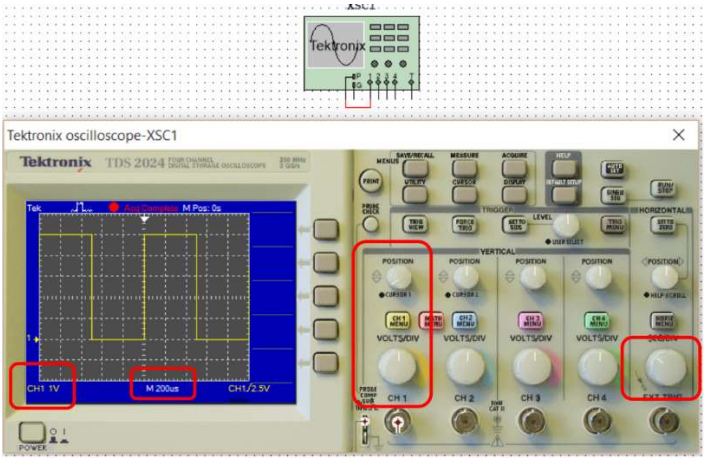


Figura 6.18 Calibración del osciloscopio en Multisim.

1.3. Ejecute el programa, **Run**.

1.4. Encienda el osciloscopio presionando el botón.

1.5. Si la señal cuadrada no se observa completa como en la figura 13, mueva la señal hacia abajo hasta que la señal se vea completa, utilice el botón de posición del control vertical, con el botón del ratón mueva el indicador (línea blanca) de la perilla.

1.6. Si la señal cuadrada se ve muy pequeña, mueva la perilla VOLTS/DIV del canal 1, hasta que la escala quede en 1V.

1.7. Si la señal cuadrada no se observa como en la figura 6.18, con dos pulsos en nivel alto y dos en nivel bajo, utilice el botón de control horizontal SEC/DIV, hasta que la escala sea de 200µs.

1.8. Cuente el número de cuadros en el eje vertical y multiplíquelo por la escala de voltaje, registre este valor de voltaje en la sección de resultados y conclusiones.

1.9. Cuente el número de cuadros de un ciclo en el eje horizontal y multiplíquelo por la escala de tiempo, registre este valor de tiempo en la sección de resultados y conclusiones.

1.10. Calcule la frecuencia y registre esta información en la sección de resultados y conclusiones.

2. Configuración del generador de funciones.

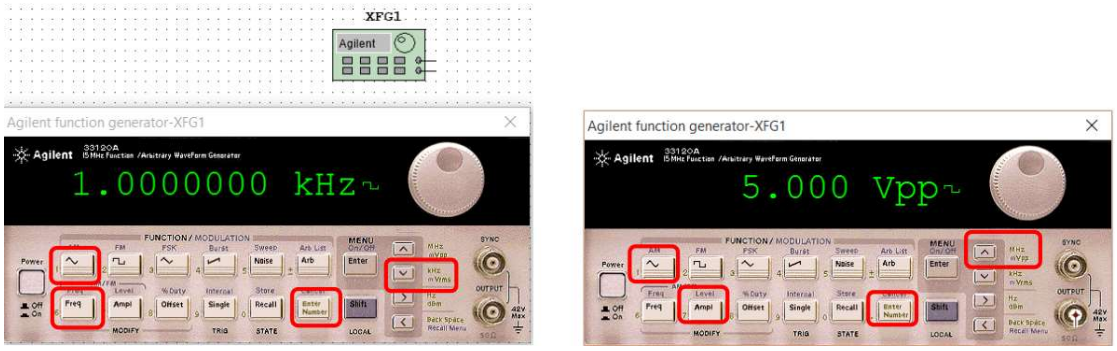
2.1 En la hoja de trabajo coloque el generador de funciones de Agilent.

2.2 Encienda el generador presionando el botón POWER.

2.3 Presione el botón  para seleccionar una señal cuadrada.

2.4 Presione los botones **Freq**, **Enter Number**, **1** (**Senoidal**),  kHz (Para Hz presionar  Hz), en ese orden para seleccionar una frecuencia de 1 kHz como se indica en la figura 6.19 (a).

2.5 Presione los botones **Ampl**, **Enter Number**, **5** (**Noise**),  (Vpp), en ese orden para seleccionar un voltaje de 5Vpp como se indica en la figura 6.19 (b).



a) *b)*

Figura 6.19 Configuración del generador de funciones en Multisim.

3. Ajuste de desplazamiento de la señal.

3.1 Conecte la salida inferior (OUTPUT) del generador de funciones al canal 1 del osciloscopio como se muestra en la figura 6.20.

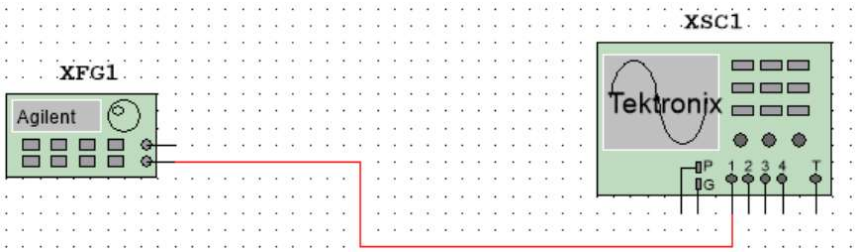
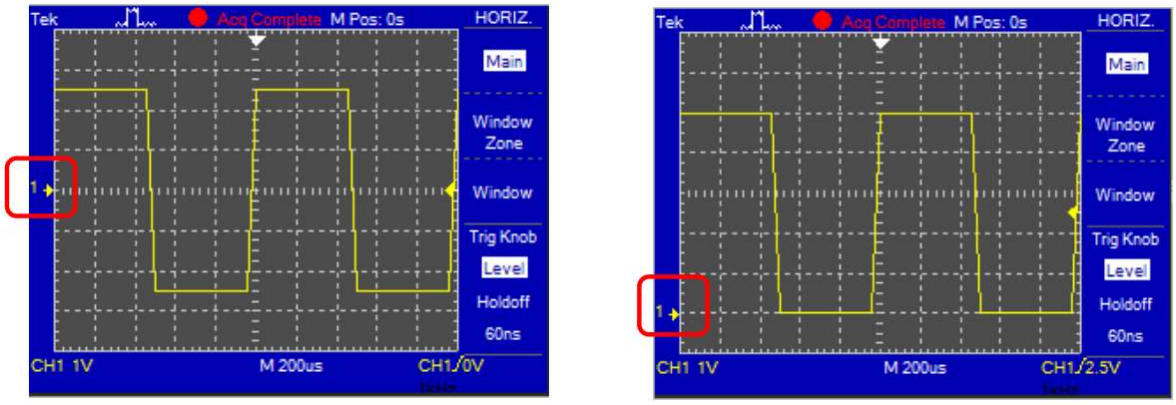


Figura 6.20 Conexión del generador de funciones y osciloscopio en Multisim.

3.2 En la pantalla del osciloscopio debe observar una señal cuadrada, ajuste la escala de voltaje y tiempo para observar la señal como se muestra en la figura 6.21.



a) *b)*

Figura 6.21 Ajuste de desplazamiento.

3.3 La señal generada tiene una amplitud de +/- 5 Vpp, esto significa que con respecto a la referencia de tierra hay +2.5 Vpp y -2.5 Vpp. En el osciloscopio se puede observar esta referencia de tierra a la derecha de la pantalla como **1→**, que está justo en el centro de la señal, ver figura 6.21 (a). Los circuitos digitales funcionan con voltajes positivos de 0 V y 5 V. Por lo tanto, es necesario desplazar esta señal hacia arriba de tal manera que la referencia quede en el nivel de voltaje inferior.

3.4 En el generador de funciones presione los botones **Offset**, ver figura 6.22, mueva la perilla del generador y al mismo tiempo observe cómo el nivel de referencia de la señal del osciloscopio se va modificando, ajuste el nivel de desplazamiento hasta que la referencia quede en el nivel bajo, ver la figura 6.21 (b).

Figura 6.22 Ajuste de desplazamiento de la señal en el generador de funciones en Multisim.

4. Diagrama de tiempos del inversor.

4.1 Conecte un inversor como se muestra en la figura 6.23.

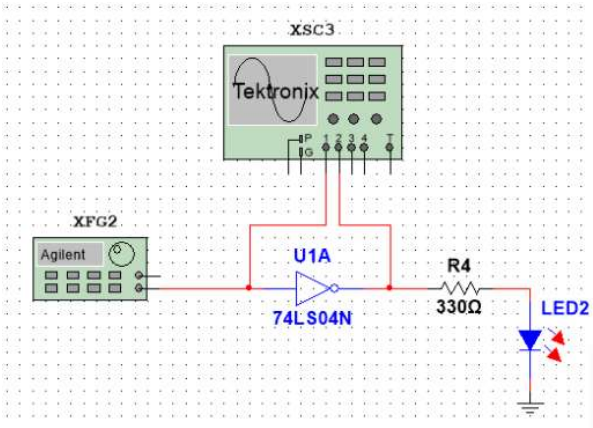


Figura 6.23 Conexión del inversor e instrumentos.

4.2 La salida del generador de funciones se conecta a la entrada del inversor.

4.3 La salida del inversor se conecta a un extremo de la resistencia de 330 Ω.

4.4 El otro extremo de la resistencia se conecta al ánodo del LED y el cátodo del LED a tierra.

4.5 El canal 1 del osciloscopio se conecta a la entrada del inversor y el canal 2 del osciloscopio se conecta a la salida del inversor.

4.6 En la pantalla del osciloscopio debe observar una señal cuadrada en el canal 1 (amarillo), y en el canal 2 (azul), debe observar la señal invertida, como se muestra en la figura 6.24.

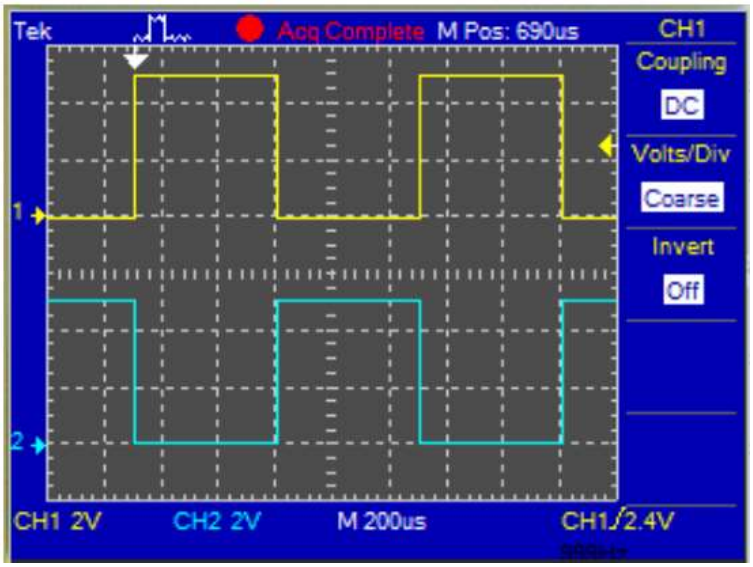


Figura 6.24 Pantalla del osciloscopio.

5. Dibuje la señal que aparece en la pantalla del osciloscopio en la sección de resultados del Reporte de Práctica, asegúrese de incluir el indicador de referencia de nivel de tierra o cero volts.

Segunda parte. Análisis de señales de un circuito.

1. Conecte el circuito que se muestra en la figura 6.25. Es el mismo circuito que analizó en la práctica 4.

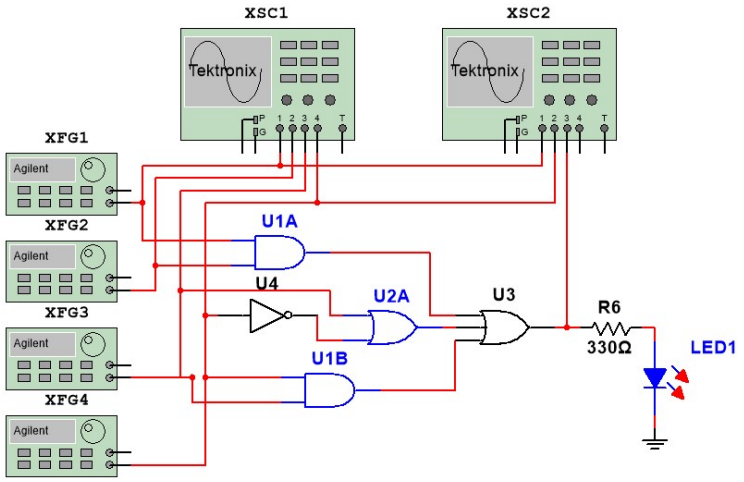


Figura 6.25 Circuito digital para ser analizado.

2. Configure los generadores de funciones de la siguiente manera:

XFG1: 125 Hz, 5 Vpp, desplazamiento (offset) 2.5 V.

XFG2: 250 Hz, 5 Vpp, desplazamiento (offset) 2.5 V.

XFG3: 500 Hz, 5 Vpp, desplazamiento (offset) 2.5 V.

XFG4: 1 kHz, 5 Vpp, desplazamiento (offset) 2.5 V.

3. Configure todos los canales de los osciloscopios a: 5 V, 1 ms.

4. En el osciloscopio XSC1 conecte cada uno de los canales a cada una de las entradas. Organice los canales para que las señales se muestren como en la figura 6.26.

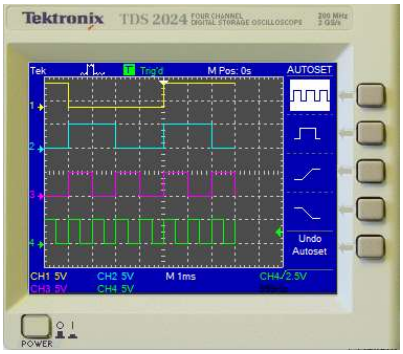


Figura 6.26 Señales de entrada de los generadores de funciones.

5. En el osciloscopio XSC2 conecte las entradas A, D y la salida del circuito. Las entradas A y D son con el propósito de sincronizar la salida con las entradas para que pueda determinar el nivel lógico de la salida.

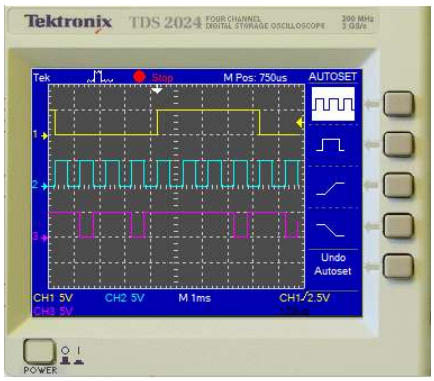


Figura 6.27 Señales A y D de entrada y la salida.

6. Dibuje las señales de entrada y salida que aparece en las pantallas de los osciloscopios en la sección de resultados y conclusiones del Reporte de Práctica. Solo dibuje las 4 señales de entrada y la señal de salida.

Sin sección de preguntas

Referencias

1. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.
2. Stephen Brown y Zvonko Vranesic. "Fundamentals of Digital Logic with VHDL Design". Second Edition. Ed. McGrawHill USA, 2005.

3. Tocci, Ronald. “Sistemas Digitales Principios y Aplicaciones”. Sexta edición. Ed. Prentice Hall, México, 1996.
4. Charles Roth Jr. Fundamentos de diseño lógico. Ed. Thomson. Quinta edición. México, 2005.
5. Bignell James W. y Robert L. Donovan. “Electrónica Digital”. Ed. CECSA. México, segunda reimpresión 1999.
6. Floyd, T.L. “Fundamentos de Sistemas Digitales”. Ed. Prentice Hall. Sexta edición, Madrid, 1997.
7. Osciloscopios de la serie HP5600B. Guía del usuario y de mantenimiento. Hewlett Packard.
8. AGF2021 Quick Start User Manual. Tectronix.
9. TBS100B User Manual. Tectronix.

Práctica # 7 Diseño e implementación de un circuito lógico digital.

Objetivos de la práctica # 7

1. Diseñar un circuito lógico digital con un funcionamiento requerido.
2. Utilizar álgebra de Boole en el diseño de un circuito lógico digital.
3. Implementar en una placa de pruebas (*Protoboard*) un circuito lógico digital y comprobar su funcionamiento respecto a los requisitos solicitados.

Material y equipo

Enseguida se muestra en la tabla 7.1 los componentes y equipo a utilizar en la presente práctica.

Tabla 7.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Multisim
1	Fuente de voltaje
1	Cables para fuente
1	Placa de prototipos
Las necesarias	Compuertas lógicas TTL determinadas en el diseño del circuito
1	Interruptor tipo DIP (*DIP switch*) con al menos **4** interruptores
1	LED
1	Resistencia de 330 Ω
4	Resistencias de 1 kΩ
--	Alambres para interconectar en la placa de pruebas

Introducción

La realización de un circuito lógico digital tiene como fin resolver una problemática o dar una solución a una situación en la cual la entrada está definida por los estados lógicos de *n* cantidad de variables. Así, para la realización del circuito lógico digital, se requiere de diseñarlo tomando en cuenta todas las combinaciones lógicas de las entradas involucradas. Sí en las entradas lógicas involucradas hubiese algunas combinaciones que no se generarán nunca, dichas combinaciones se denominarán "no importa" (*Don't care*), las cuales ayudan

a una mejor minimización en el diseño del circuito resultante. Los medios que se utilizan para el diseño de los circuitos lógicos digitales son a) La tabla de verdad, en la cual se representan todas las combinaciones lógicas de la entrada y los respectivos valores de la salida o salidas si hubiese más de una. b) El mapa de Karnaugh, con el cual se procede a la minimización de la función de cada salida, si hay más de una, siguiendo la técnica de eliminar las variables encontradas en las celdas adyacentes por estar asociadas en su forma normal y complementada. c) Optimización por factorización o uso de las compuertas Xor, Xnor, Nand o Nor, pues frecuentemente, el utilizar la factorización o las compuertas mencionadas ayudan en la minimización otorgada por el mapa de Karnaugh.

Análisis previo

El diseño del circuito lógico se inicia con la elaboración de la tabla de verdad, la cual puede ser proporcionada directamente o por medio de un enunciado, una operación matemática o una definición numérica. Para la presente práctica utilizaremos una definición numérica denominada **palíndromo numérico** o **capicúa**. Tomado de https://es.wikipedia.org, "*En matemáticas, la palabra capicúa (del catalán cap i cua, 'cabeza y cola') se refiere a cualquier número que se lee igual de izquierda a derecha que de derecha a izquierda. Ejemplos: 161, 2992, 3003, 91019, 5005, 292, 2882, 2442, 9102019.*"

En el caso de la representación de los números binarios, 101, 11, 11011, 1001001 son palíndromos. 100, 110, 1011, etc., no son palíndromos. (Tomado de https://www.techiedelight.com/es/check-if-binary-representation-number-palindrome/).
Como caso particular de la presente práctica, consideraremos al cero y al uno como palíndromos, y los ceros que se encuentren a la izquierda del valor numérico no serán considerados, dando como palíndromo, por ejemplo, "0011", "0101" y a otras combinaciones que empiezan con cero.

Desarrollo

Enseguida se da el enunciado en el que se describe la problemática a resolver:

Diseñar un circuito lógico digital de cuatro entradas y una salida; una combinación binaria del 0_{10} al 15_{10} será representada en las cuatro entradas binarias y la salida permanecerá el LED apagado, el cual, al encender, indicará que el valor representado en la entrada binaria es un palíndromo, de acuerdo con lo considerado en el análisis previo.

Los pasos que se deberán de registrar para el reporte de la práctica son:

1. Obtención de la tabla de verdad de acuerdo con lo indicado en el enunciado de la problemática a resolver.

2. De la tabla de verdad, obtenga una expresión lógica que brinda la solución a la problemática expuesta en el enunciado, sumando los mintérminos correspondientes a un verdadero en la salida de la misma tabla de verdad.

3. Emplee álgebra de Boole para simplificar la expresión obtenida de la tabla de verdad.

4. Verifique si su diseño se puede optimizar utilizando alguna de las compuertas mencionadas en el "Análisis previo".

5. Utilice el mapa de Karnaugh para obtener una expresión minimizada de la tabla de verdad obtenida en el paso 1.

6. Verifique si la minimización obtenida del mapa de Karnaugh puede ser optimizada utilizando alguna de las compuertas mencionadas en el "Análisis previo".

7. Compare las expresiones obtenidas en el paso 4 y en el paso 6 y de sus conclusiones.

8. Implemente el diseño del circuito lógico en Multisim y verifique que cumplen con lo solicitado en el enunciado de la problemática a resolver.

9. Implemente el diseño del circuito lógico en la placa de pruebas y registre sus resultados en una tabla que incluya los resultados del circuito lógico en Multisim y anéxela al reporte de la práctica.

Preguntas

1. Describa los pasos para realizar un diseño lógico digital.
2. Indica cuáles son las compuertas que pueden optimizar el resultado de un mapa de Karnaugh.
3. Explique que son los términos "No importa" dentro del mapa de Karnaugh.
4. Explique que es un palíndromo binario.

Referencias

1. Tocci, Ronald. "Sistemas Digitales Principios y Aplicaciones". Sexta edición. Ed. Prentice Hall, México, 1996.

2. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.
3. NI Circuit Design Suite Getting Started Guide.
4. "Comprobar si la representación binaria de un número es palíndromo o no", https://www.techiedelight.com/es/check-if-binary-representation-number-palindrome/
5. "Números palíndromos", http://progra.usm.cl/apunte/ejercicios/1/es-numero-palindromo.html
6. "Compruebe si la representación binaria de un número es palíndromo", https://es.acervolima.com/compruebe-si-la-representacion-binaria-de-un-numero-es-palindromo/
7. "Capicúa", https://es.wikipedia.org/wiki/Capic%C3%BAa

Práctica # 8 Proyecto en esquemático para Vivado

Objetivos de la práctica # 8

1. Conocer, comprender y aplicar las bases de la creación basado en esquemático en Vivado.
2. Conocer y utilizar la interfaz de desarrollo basado en esquemático para Vivado.
3. Programar la tarjeta de desarrollo fpga "Basys 3" para un proyecto desarrollado en esquemático.
4. Conocer y comprender los pasos necesarios para simular un proyecto en Vivado utilizando el lenguaje Verilog.

Material y equipo

Enseguida se muestra en la tabla 8.1 los componentes y equipo a utilizar en la presente práctica, que será únicamente simulación.

Tabla 8.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado
1	Tarjeta Basys 3
1	Cable USB para Basys 3
1	Biblioteca XUP.zip para Vivado IP Integrator (Archivo en C. V.)

Introducción

El proceso de diseño consiste en (a) Entrada del Diseño, (b) Implementación del Diseño y (c) verificación del diseño. Los diseños se pueden realizar de varias formas: Usando un Editor de Esquemáticos para dibujar un diagrama lógico, usando una descripción textual orientada al comportamiento del circuito (ABEL o VHDL), o una combinación de ambos. En esta práctica se le guiará a través del flujo de diseño usando el software Xilinx Vivado para crear un circuito digital simple usando Vivado IP Integrator (IPI) la que es una biblioteca para el diseño de circuitos basado en esquemático. Un flujo de diseño típico por esquemático, consiste en crear un proyecto de Vivado, configurar opcionalmente una configuración de biblioteca de IP definida por el usuario, crear un diseño

de bloque usando varias IP (para el esquemático), crear un envoltorio HDL (qué es el archivo que introduce al proyecto el esquemático en cuestión y está en lenguaje Verilog), crear y/o agregar archivos de restricciones de usuario, opcionalmente ejecutar una simulación de comportamiento (opcionalmente se puede realizar en VHDL o Verilog, en la práctica 5 se realizó con VHDL, en esta práctica se realizará en lenguaje Verilog), sintetizando el diseño, implementando el diseño, generando el flujo de bits y finalmente verificando la funcionalidad en el hardware descargando el archivo de flujo de bits generado.

Verilog HDL es uno de los dos lenguajes de descripción de hardware (HDL, del inglés *Hardware Description Language*) más comunes que utilizan los diseñadores de circuitos integrados (IC), con una sintaxis derivada del lenguaje de programación C. Se corresponde inicialmente con el IEEE standard 1364-1995, extendido posteriormente en IEEE standard 1364-2001 y SystemVerilog.

Un HDL permite simular el diseño en etapas tempranas del ciclo de diseño para corregir errores o experimentar con diferentes arquitecturas, llevar a cabo la síntesis lógica del mismo y facilitar la documentación de este. Los diseños descritos en HDL son independientes de la tecnología, fáciles de diseñar y depurar, y suelen ser más legibles que los esquemas, especialmente para circuitos grandes y complejos, lo que permite además un prototipado rápido y acelerar el tiempo de salida del producto final (TTM, del inglés *Time-To-Market*).

Verilog se puede utilizar para describir diseños en cuatro niveles de abstracción:

-Nivel **comportamental**: descripción algorítmica o arquitectural (similar al código C con instrucciones if, case y loop).

-Nivel de **transferencia de registros** (RTL, del inglés Register Transfer Level): utilizando registros conectados por ecuaciones booleanas.

-Nivel de **puertas**: interconexión de puertas lógicas AND, NOR, etc.

-Nivel de **conmutación**: interconexión de los transistores MOS que forman las puertas.

Los últimos tres niveles se corresponden con una descripción también denominada estructural, si bien el lenguaje permite llevar a cabo descripciones mixtas que combinan parte comportamental y estructural, así como las construcciones que pueden utilizarse para controlar la entrada y salida de la simulación.

Es común actualmente utilizar descripciones Verilog como entrada para herramientas de síntesis lógica que generan descripciones a nivel de puerta o netlist del circuito, si bien algunas construcciones de Verilog no son sintetizables y es importante diferenciar ambos tipos. También la manera de escribir el código afectará en gran medida al tamaño y la velocidad del circuito sintetizado. Como en la mayoría de los casos se pretenderá sintetizar

los circuitos, las construcciones no sintetizables deben usarse sólo para bancos de pruebas (testbenches). Se trata de módulos de programa utilizados para generar las E/S necesarias para simular el resto del diseño normalmente bajo el modelo estímulo/respuesta.

Como en la mayoría de HDLs se distinguen dos tipos/estilos de codificación/ modelado/descripción que tienen correspondencia con determinados niveles de abstracción señalados:

Estructural, como la descripción del esquemático de un circuito sin almacenamiento, p.ej.:

```
assign a = b & c | d;
assign d = e & (~c);
```

Procedural, usado para circuitos con almacenamiento, o como una descripción dependiente de condiciones o estados, p.ej.:

```
always @(posedge clk)
count <= count+1;
```

El estilo procedural es preferido por los diseñadores por la facilidad de codificación y semejanza con los lenguajes de programación de alto nivel (HLL, del inglés High Level Language), si bien es importante tener en cuenta que la mayoría de construcciones procedurales con asignación implicarán la generación de almacenamiento en síntesis y esto puede no ser deseable en algunos casos donde se desea evitar lógica superflua atendiendo a restricciones de área o de temporización, por lo que se recomienda hacer uso del estilo estructural en circuitos puramente combinacionales.

Análisis previo

Creación del archivo esquemático utilizando compuertas básicas.

Hay dos bibliotecas digitales para Vivado IP Integrator que incluyen bloques lógicos básicos para la entrada de esquemas y la enseñanza del diseño lógico básico; Modelos XUP_LIB (en su mayoría compuertas básicas: AND, OR, NOT, XOR, etc.) y 74LSXX. Estas bibliotecas incluyen bloques IP destinados a la enseñanza en el aula y se pueden usar con el entorno gráfico del integrador IP Vivado para diseñar, simular y crear diseños para Xilinx FPGA utilizando esquemas. Para realizar la programación en esquemático en Vivado, es necesario instalar la biblioteca; cree un nuevo proyecto de Vivado y, en la configuración del proyecto (en Vivado, Herramientas > Configuración del proyecto), seleccione la pestaña IP y, en la pestaña Administrador de repositorio, agregue el directorio de las carpetas XUP_LIB. Luego podrá agregar estos bloques de IP a sus diseños en el integrador de IP. En

la figura 8.1 se muestra cómo acceder directamente desde el flujo de navegador a la ventana de configuración, en la cual se selecciona el repositorio con el menú que está abajo de la opción IP; para definir el repositorio se presiona la cruz abajo de la tabla (IP Repositories).

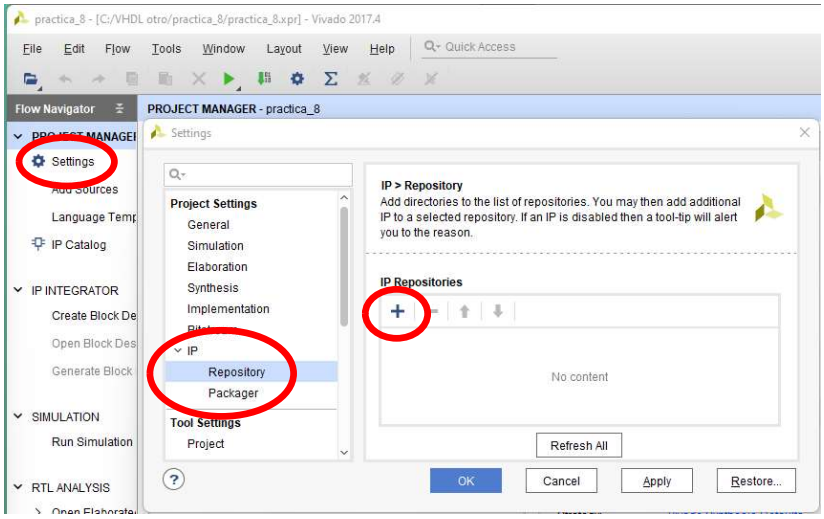


Figura 8.1 Pasos para agregar la biblioteca de esquemáticos.

Al presionar la cruz, aparecerá la ventana mostrada en la figura 8.2, en donde habrá de buscar y seleccionar el directorio donde se descomprimió el archivo XUP.zip, el cual se encuentra en el Campus Virtual.

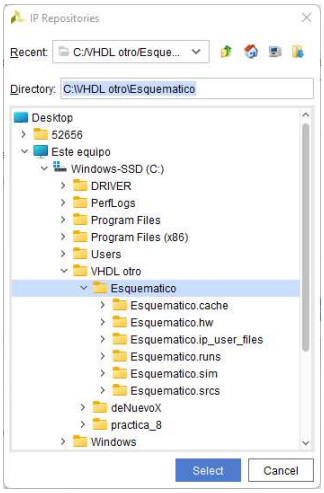


Figura 8.2 Explorador de Vivado para agregar repositorios.

Una vez seleccionado el archivo del repositorio, dar seleccionar y en la siguiente ventana se observará el repositorio referenciado, como se muestra en la figura 8.3. finalmente cierre la ventana dando clic en el botón "Ok".

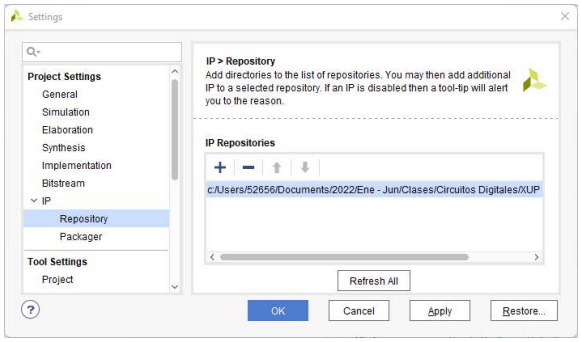


Figura 8.3 Repositorio agregado al proyecto.

Seguido creamos un bloque de diseño correspondiente al IP Integrator como se muestra la figura 8.4, le asignamos un nombre al diseño y cerramos la ventana.

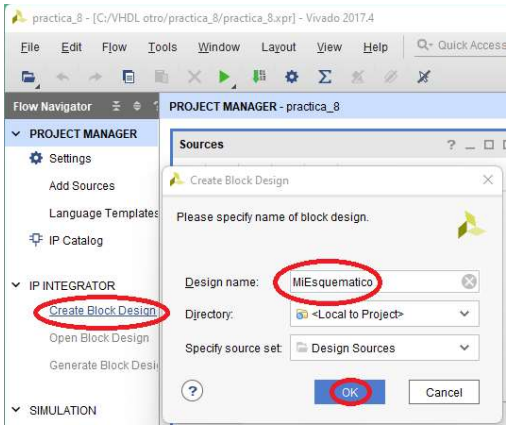


Figura 8.4 Creación de un bloque de diseño.

Se generará una pestaña llamada Diseño al lado de la pestaña Fuentes; dentro de la pestaña Diseño se encuentra el diagrama que se acaba de generar, así como la ventana del diseño para el diagrama, la cual contiene una cruz, que al presionarla se podrán agregar los bloques para el diagrama; lo anterior se muestra en la figura 8.5.

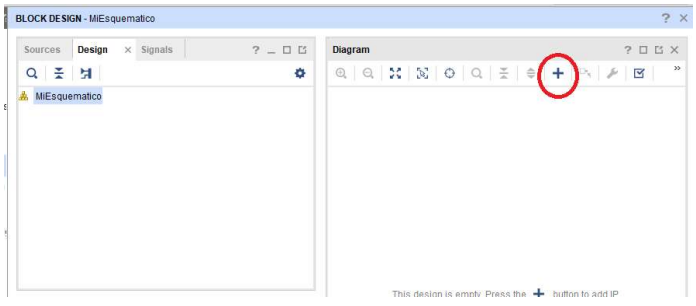
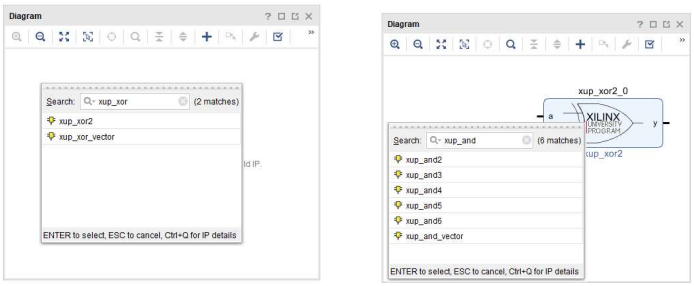


Figura 8.5 Diagrama esquemático agregado y ventana para su diseño.

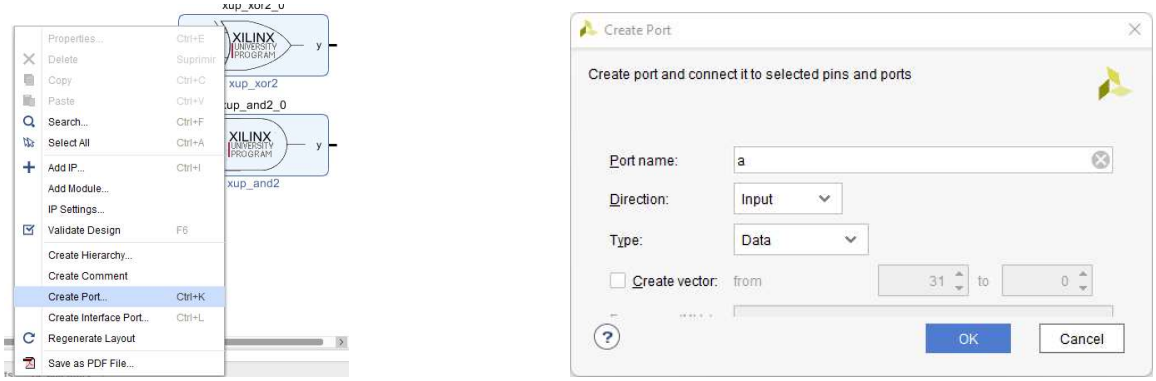
Para el presente análisis, se desarrollará un circuito medio sumador. El primer elemento qué se agregará es una compuerta XOR, seguido de una compuerta AND. Estos pasos se muestran en la figura 8.6 a) y b).



a) *b)*

Figura 8.6 Modificación del diagrama, agregando compuertas a) XOR y b) AND.

Enseguida se agrega los puertos de entrada (bits a sumar *a* y *b*), dando clic en el área del diagrama, y seleccionando en el menú contextual la opción "Create Port…", creando un puerto para la entrada *a* y otro para la entrada *b* según como se muestra en la figura 8.7 a). En la figura 8.7 b) se muestra la configuración del puerto *a*, con una dirección de entrada y del tipo Dato.



a) *b)*

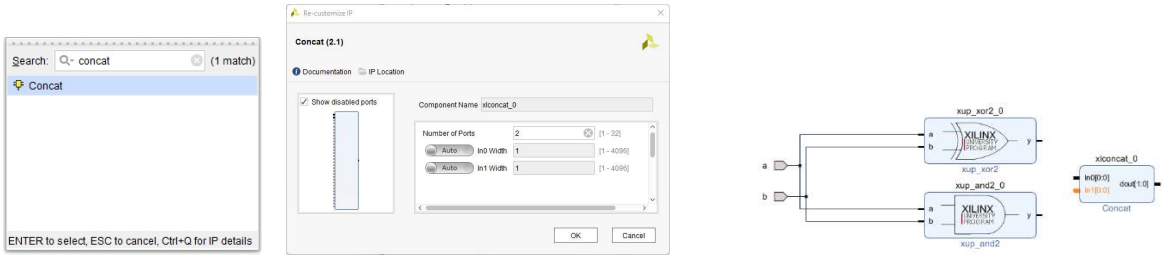
Figura 8.7 Creación del puerto a) 'a' y b) configuración

Para el puerto *b* se siguen exactamente los mismos pasos de la figura 8.7. En la figura 8.8 a) se muestra el avance del diagrama y en la figura 8.8 b) se indica el cableado, el cual se consigue al acercar el ratón en las terminales del puerto o en las terminales de las compuertas y dando clic y jalando hacia la dirección deseada.

a) *b)*

Figura 8.8 Avance del diagrama en a) y en b) las conexiones realizadas.

Para la salida concatenaremos los 2 bits en una sola palabra de longitud 2, el cual encontraremos en la biblioteca con el nombre de "Concat"; este módulo se configurará dándole 2 clics al módulo donde indicaremos 2 puertos de entrada (2 bits) y 2 salidas de un bit. El módulo agregado y la configuración se muestran en la figura 8.9 a) y b). Una vez agregados los módulos de concatenación, uniremos las salidas de las compuertas a través de cableado como lo indica la figura 8.9 c).



a) *b)* *c)*

Figura 8.9 a) Selección del módulo Concat, b) configuración y c) resultado.

Si se desea tener un bus de entrada y dividirlo, ya sea en conjunto de bits o en bits sencillos, para asignarlo a diferentes entradas, el módulo que se utiliza se denomina "Slice". Para finalizar el diseño, se conectan las salidas de las compuertas al concatenador y la salida del concatenado se conecta a un puerto bus de 2 bits, cuya configuración se muestra en la figura 8.10.

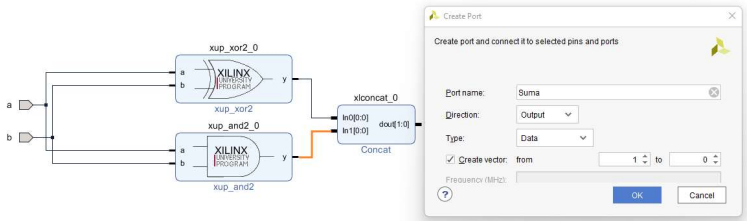


Figura 8.10 Avance de diseño y configuración del puerto de salida.

El diagrama del esquemático terminado se muestra en la figura 8.11.

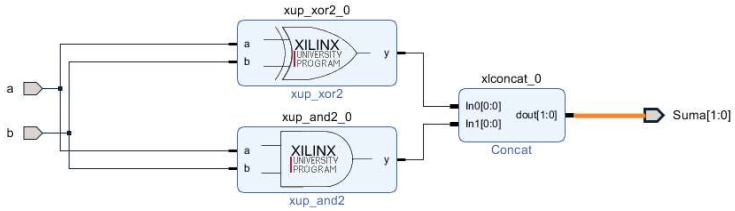


Figura 8.11 Diagrama del esquemático finalizado.

Una vez terminado el diagrama, seleccionamos el archivo del esquemático de la pestaña Fuentes, le damos clic con el botón derecho para abrir el menú contextual y seleccionamos la opción "Generar Productos de Salida" y seleccionamos generar en la ventana que abre, estos pasos se muestran en la figura 8.12.

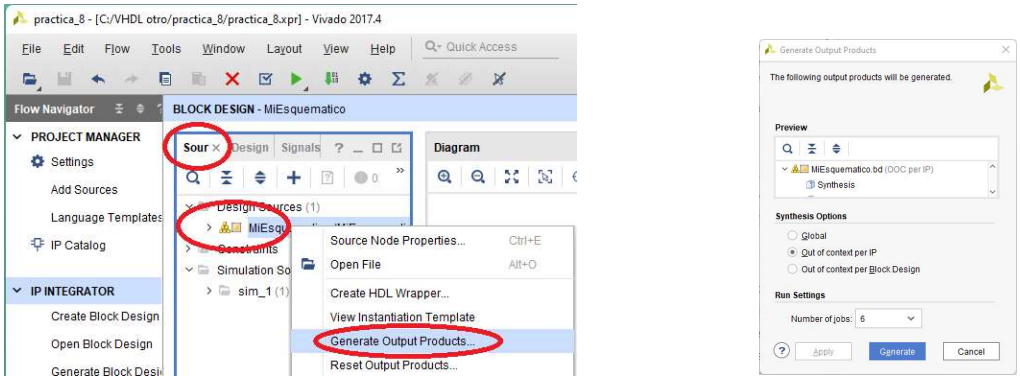
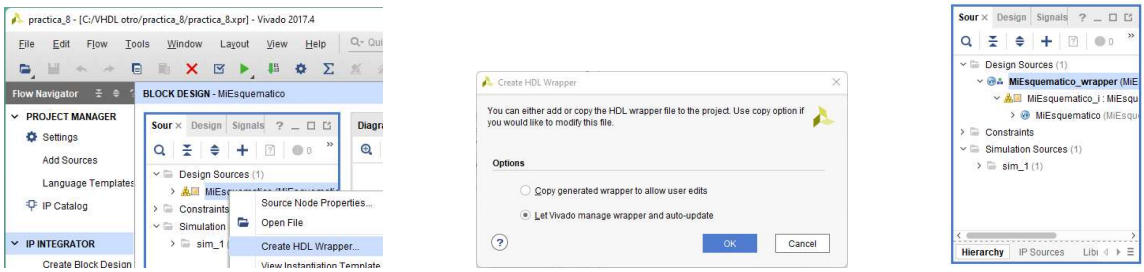


Figura 8.12 Pasos para la generación de los productos de salida del esquemático.

Para finalizar la parte del esquemático, de nuevo damos clic con el botón derecho sobre el archivo del esquemático y seleccionamos la opción "Crear Envolvente HDL…", lo que nos abrirá una ventana en la que daremos clic en "Ok", lo que generará una jerarquía como la que se muestra en la figura 8.13 a), b) y c) respectivamente.



a) *b)* *c)*

Figura 8.13 a) Creación de envolvente HDL, b) Ventana resultante de crear envolvente HDL, c) Jerarquía resultante.

Creación del archivo de simulación Verilog.

Los siguientes pasos corresponden a la parte de la simulación del proyecto diseñado. Comenzamos por agregar un archivo de simulación dando clic en agregar fuentes como se indica en la figura 8.14 nota 1, lo cual abre una ventana con tres opciones, asegurarse de que se escoge la opción de agregar una fuente de simulación como se indica en la figura 8.14 nota 2 dando lugar a la ventana de agregar fuente.

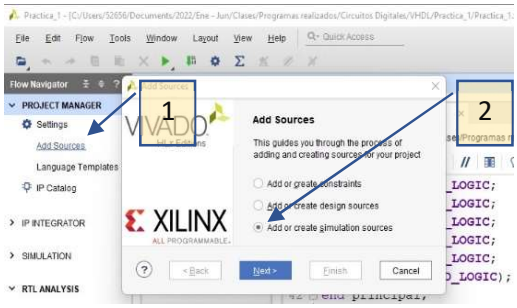
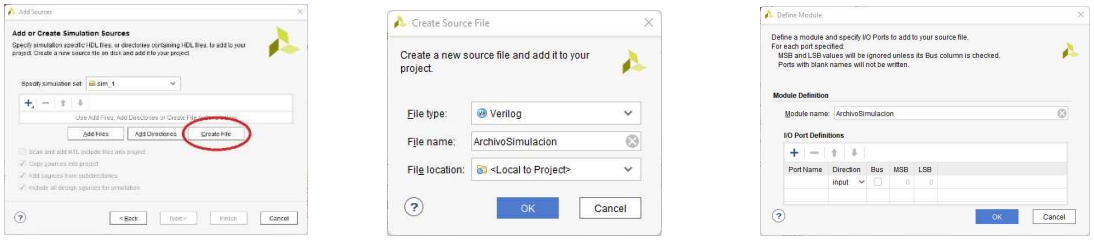


Figura 8.14 Pasos para agregar archivo de simulación.

En la ventana agregar fuente seleccionar crear archivo como se indica en la figura 8.15 a) con lo que se abrirá la ventana de crear archivo fuente y deberá especificarse el tipo del archivo, en este caso Verilog y como nombre sugerido *ArchivoSimulacion*, como se indica en la figura 8.15 b); dar clic en el botón "*OK*" y finalizamos la agregación de archivos fuente. Cerrramos la última ventana sin definir módulo figura 8.15 c).



a) *b)* *c)*

Figura 8.15 Ventanas para agregar archivo de simulación.

Abrimos el archivo de simulación que se ubicará en la ventana de Fuentes con la ruta: Sources -> Simulation Sources -> sim_1 -> ArchivoSimulacion:

Y modificamos el archivo con el siguiente código:

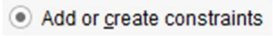
```
module ArchivoSimulacion();
    reg a_en;
    // Alambrado para la simulación entrada a, entrada es reg
    reg b_en;
    // Alambrado para la simulación entrada b, entrada es reg
    wire [1:0] Suma_sal;// Alambrado salida Suma, salida wire
MiEsquematico_wrapper MiModulo (.a(a_en),.b(b_en),.Suma(Suma_sal));
// Los nombres del módulo original se anteceden por punto y se
//alambran utilizando paréntesis
initial begin
    a_en = 0; b_en = 0;//Valores asignados
    #50//Espera 50 unidades de tiempo (50 ns)
    a_en = 1; b_en = 0;//Valores asignados
    #50//Espera 50 unidades de tiempo (50 ns)
    a_en = 1; b_en = 1;//Valores asignados
end
endmodule
```

Seleccionamos ejecutar la simulación como se indica en la figura 8.16 a) y posteriormente analizamos el diagrama de tiempos como aparece en la figura 8.17 b).

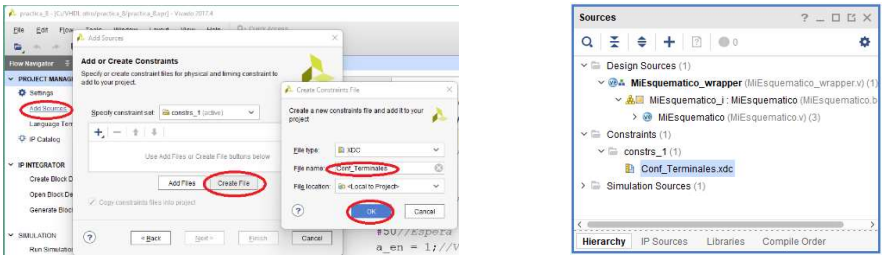
a) *b)*

Figura 8.16 a) Seleccion de ejecutar simulación y b) resultado de la simulación.

Creación del archivo de restricciones (Asignación de terminales).

Para agregar un archivo de restricciones para la asignación de terminales del proyecto, los pasos se muestran en la figura 8.17 a) que consiste en dar clic en agregar fuentes, esto abre una ventana (NO mostrada) de agregar fuentes donde hay que indicar que se desea agregar las restricciones 

, abriendo otra ventana donde hay que indicar que se desea crear el archivo y en la siguiente ventana le indicamos el nombre dando aceptar para todas las ventanas y terminar con el archivo agregado en Fuentes -> Restricciones -> constrs_1 como se muestra una figura 8.17.



a) *b)*

Figura 8.17 a) Pasos para agregar archivo de restricciones y b) Archivo añadido.

Las líneas para asignar a las variables del diseño las terminales físicas del FPGA:

```
set_property PACKAGE_PIN V17 [get_ports a]
set_property IOSTANDARD LVCMOS33 [get_ports a]
set_property PACKAGE_PIN V16 [get_ports b]
set_property IOSTANDARD LVCMOS33 [get_ports b]
set_property PACKAGE_PIN U16 [get_ports {Suma[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Suma[0]}]
set_property PACKAGE_PIN E19 [get_ports {Suma[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Suma[1]}]
```

La configuración de las terminales en la Basys 3 se muestra en la figura 8.18 y la asignación se hace por medio del comando "`set_property PACKAGE_PIN`"; para asignar el voltaje de trabajo de la terminal se utiliza "`LVCMOS33`", que indica que trabajará con 3.3 V dicha terminal.

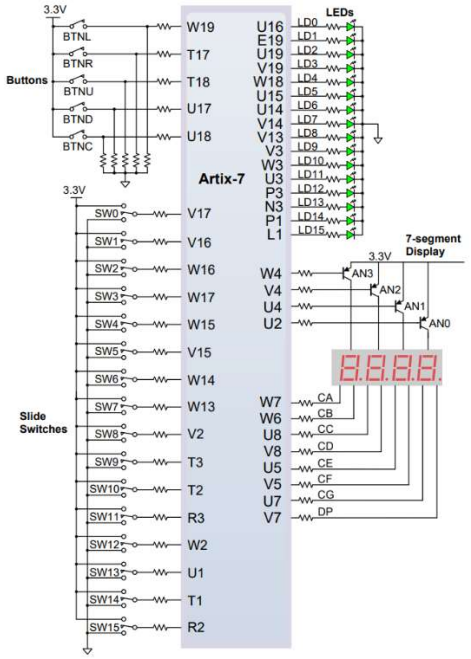
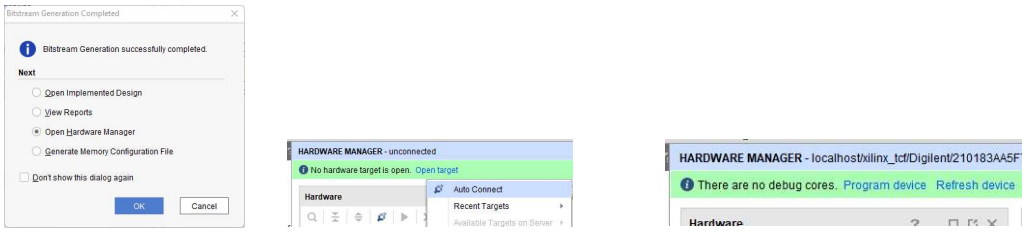


Figura 8.18 Configuración de terminales en Basys 3.

Programación de la Basys 3.

Seleccionamos la opción de generación del archivo Bitstream Generate Bitstream que se encuentra bajo la opción de Programación y Depuración PROGRAM AND DEBUGaceptando los diálogos que se presentan. Es recomendable tener ya conectado a la tarjeta Basys 3 para que sea reconocida por el programa. Al terminar la generación del archivo Bitstream, se abrirá una ventana donde hay que escoger "Abrir Manager de Hardware" como se muestra en la figura 8.19 a), una vez abierta la ventana, seleccionar abrir destino e indicar auto conectar (figura 8.19 b). Finalmente seleccionar programar dispositivo (figura 18.19 c).



a) *b)* *c)*

Figura 8.19 a) Seleccionar Manejador de Hardware, b) Conectar a destino, c) Programar dispositivo.

Al terminar los pasos de la figura 8.19, aparecerá la ventana para programar el dispositivo, dándole clic en el botón programar como se indica en la figura 8.20.

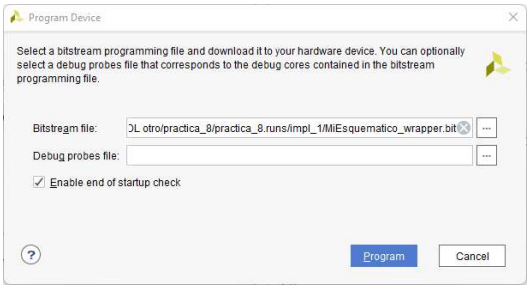


Figura 8.20 Ventana para realizar programación del dispositivo.

Desarrollo

Realizar un proyecto utilizando esquemático para un comparador de 2 palabras de 3 bits, llamadas **A** y **B**, y cuya salida será las relaciones lógicas **A** = **B**, **A** > **B** y **A** < **B**, incluyendo la simulación y la programación de la Basys 3.

Preguntas

1. ¿Qué es un BUS de datos?
2. ¿Cuáles son las terminales para los interruptores en la Basys 3?
3. ¿Cuáles son los productos generados con "Generar Productos de Salida"?
4. ¿Cuál es la finalidad del módulo esquemático "Slice"?
5. ¿Cuáles son los pasos para agregar un repositorio en Vivado?

Referencias

1. Llanos, Juan; "Vivado Tutorial 1 Crear un proyecto", https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; "Vivado Tutorial 2 Simulación", https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.

Práctica # 9 Decodificador de hexadecimal a 7 segmentos para Basys 3

Objetivos de la práctica # 9

1. Conocer y utilizar la configuración de los display's de 7 segmentos de la Basys 3.
2. Emplear la programación VHDL en el uso de los display's de 7 segmentos.
3. Conocer y comprender los pasos necesarios para simular y programación de un proyecto en Vivado para la Basys 3.

Material y equipo

Enseguida se muestra en la tabla 9.1 los componentes y equipo a utilizar en la presente práctica, que comprenderá la programación en VHDL junto con el grabado de la tarjeta Basys 3 con el programa generado.

Tabla 9.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado
1	Tarjeta Basys 3
1	Cable USB para Basys 3

Introducción

Los display de 7 segmentos son muy comunes en los aparatos electrónicos para mostrar cantidades, generalmente decimales, pero también se pueden mostrar valores hexadecimales de forma conveniente. La tarjeta Basys 3 integra un módulo con cuatro display's de 7 segmentos ánodo común de forma multiplexada, por lo que, al mostrar información en las terminales de los 7 segmentos, habrá que habilitarse el display en el que se desea mostrar el dígito. En la figura 9.1 se muestra la tarjeta Basys 3 Indicando con el número cuatro en la posición del módulo de cuatro display de 7 segmentos. El módulo de display activara cada segmento con lógica negativa, Ya que cada segmento disponible el cátodo, que es la parte negativa del LED.

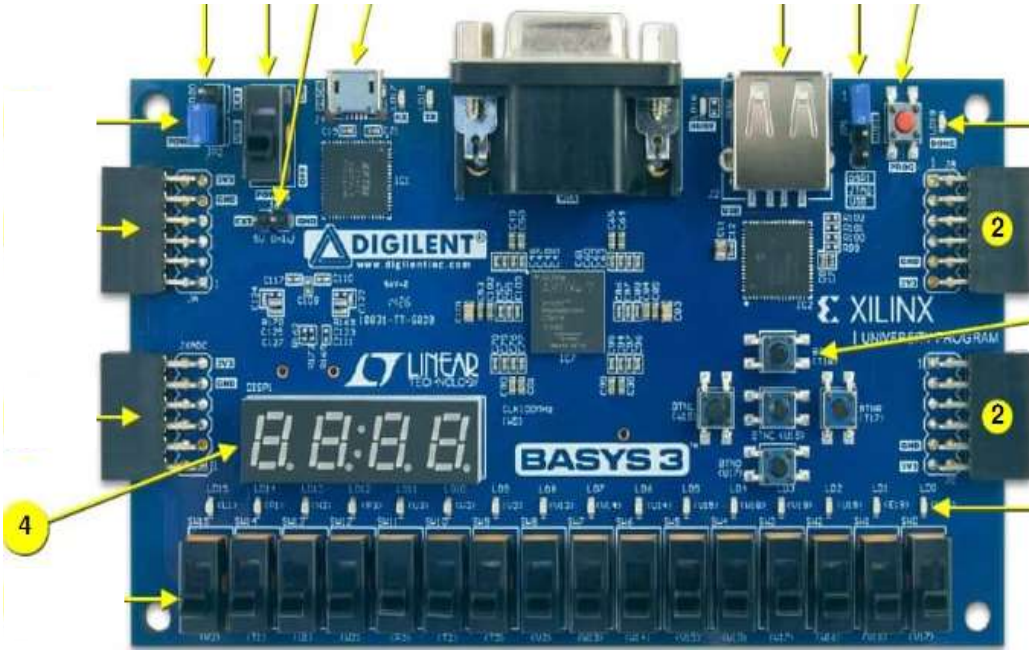


Figura 9.1 Tarjeta Basys 3, Indicando con el número cuatro el módulo de cuatro display's de 7 segmentos.

En la figura 9.2 se muestra la configuración de los leds para el display de 7 segmentos ánodo común y cátodo común, aquí se observa claramente que la lógica para encender los leds en un ánodo común es lógica negada.

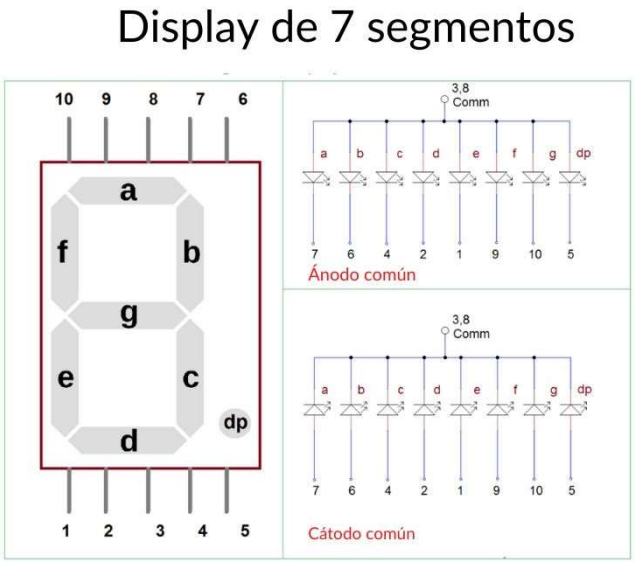


Figura 9.2 Configuración del display de 7 segmentos en ánodo común y cátodo común.

La tarjeta Basys 3 asigna una serie de salidas del FPGA al módulo de 4 display's, así como a las líneas que seleccionan a cada uno de los display's, además de los puntos asociados al display. la figura 9.3 muestra esta configuración y es importante tenerla a la mano en el

momento que se realiza la asignación de terminales a las variables de la programación. Como se puede observar en la figura 9.3, la selección de los display's se realiza con lógica negativa, ya que podrá recordar que los transistores de tipo **PNP** están impulsados por una corriente negativa polarizada en la base para controlar el flujo del Emisor al Colector.

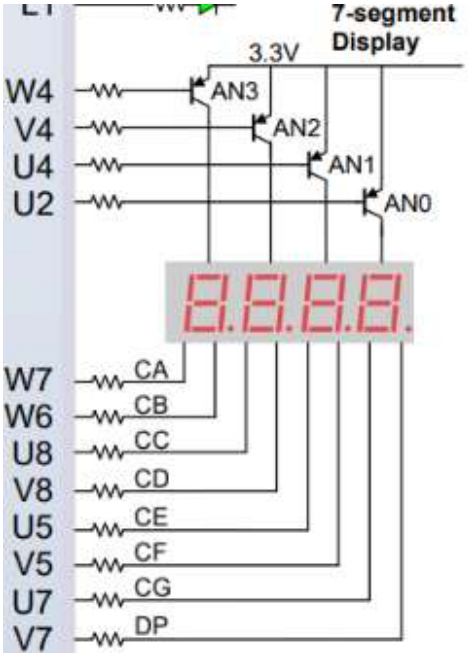


Figura 9.3 Disposición de terminales para controlar el módulo del display en Basys 3.

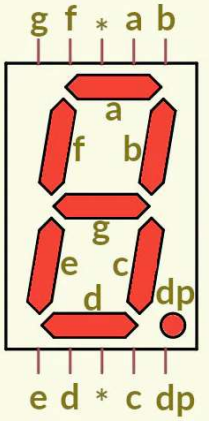
Análisis previo

Para realizar el diseño del decodificador de hexadecimal a 7 segmentos, primeramente, utilizamos la tabla de verdad para cada uno de los segmentos. En esta sección del manual, únicamente realizaremos el segmento **a**, dejando el resto de los segmentos para el desarrollo de práctica por parte de los estudiantes. La tabla de verdad para todos los segmentos se muestra en la figura 9.2 considerando que el display tiene una configuración de ánodo común y el dígito hexadecimal consta de cuatro bits.

Obtención de la función para el segmento a del display de 7 segmentos.

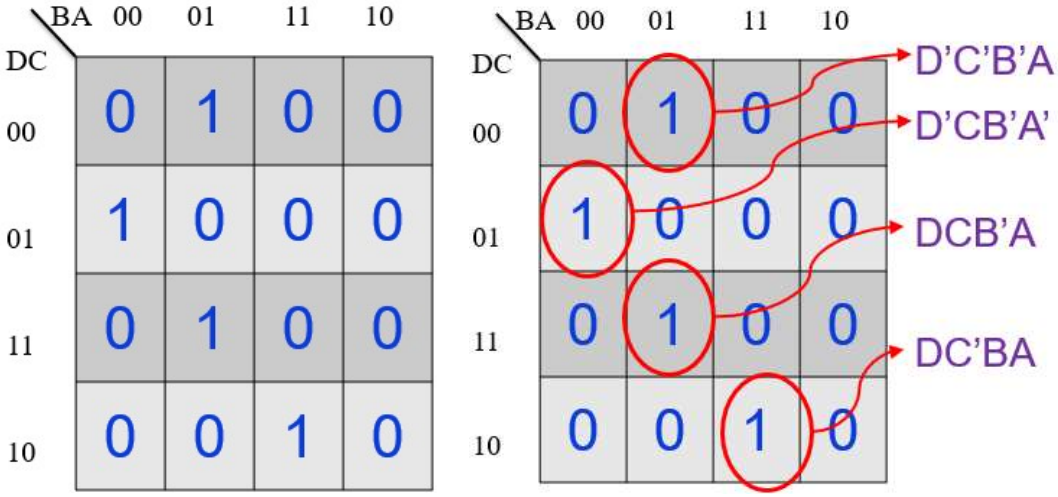
En la figura 9.3 a) se presenta el mapa de Karnaugh para el segmento **a** tomado de los datos de la figura 9.2, y en la figura 9.3 b) se tiene la minimización del mapa de Karnaugh. Es importante que note la disposición de los bits, teniendo como más significativo el bit D y como menos significativo el bit A.

Display 7 segmentos ánodo común



Ánodo Común								
Común *	Número	g	f	e	d	c	b	a
VCC	0	1	0	0	0	0	0	0
VCC	1	1	1	1	1	0	0	1
VCC	2	0	1	0	0	1	0	0
VCC	3	0	1	1	0	0	0	0
VCC	4	0	0	1	1	0	0	1
VCC	5	0	0	1	0	0	1	0
VCC	6	0	0	0	0	0	1	0
VCC	7	1	1	1	1	0	0	0
VCC	8	0	0	0	0	0	0	0
VCC	9	0	0	1	0	0	0	0
VCC	A	0	0	0	1	0	0	0
VCC	b	0	0	0	0	0	1	1
VCC	C	1	0	0	0	1	1	0
VCC	d	0	1	0	0	0	0	1
VCC	E	0	0	0	0	1	1	0
VCC	F	0	0	0	1	1	1	0

Figura 9.4 Tabla de verdad para los 7 segmentos del display ánodo común con salida hexadecimal.



a) *b)*

*Figura 9.5 a) Mapa de Karnaugh para segmento **a** y b) minimización.*

Una vez que la función se a minimizado por medio de los mapas de Karnaugh, la función resultante es:

$$f_{a(DCBA)} = \bar{D}\bar{C}\bar{B}A + \bar{D}C\bar{B}\bar{A} + DC\bar{B}A + D\bar{C}BA$$

Intentamos optimizar la función ya sea por factorización o si es posible utilizar compuertas tal como la Xor; empezamos por factorizar el primer término y el tercer término:

$$\bar{D}\bar{C}\bar{B}A + DC\bar{B}A$$

Resultando:

$$(\bar{D}\bar{C} + DC)\bar{B}A = (D \odot C)\bar{B}A$$

Finalmente, la función queda de la siguiente forma:

$$f_{a(DCBA)} = (D \odot C)\bar{B}A + \bar{D}C\bar{B}\bar{A} + D\bar{C}BA$$

Creación del proyecto en Vivado

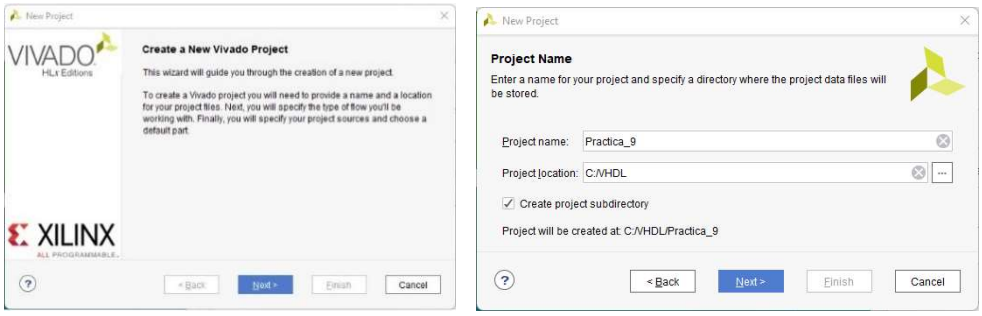
Siguiendo las instrucciones de la práctica 5, los pasos a seguir son:

1. Abrir el programa Vivado, que lo podrás distinguir por su icono: 

2. Una vez iniciado el programa, selecciona en el menú rápido crear proyecto (figura 9.6).

Figura 9.6 Ventana de inicio de Vivado y la opción "Crear proyecto".

3. Se abrirá una ventana en la que se transforma que se creará un proyecto (figura 9.7 a), dar clic en continuar y aparecerá una ventana donde hay que colocar el nombre del proyecto, en este caso *Practica_9* (figura 9.7 b), y dar clic en continuar.



a) *b)*

Figura 9.7 Ventana de creación de proyecto y de asignación de nombre al proyecto.

4. Ahora toca definir el tipo de proyecto, seleccionamos RTL Project en la ventana que continuo del paso anterior y seleccionamos la opción de no especificar los archivos fuentes en este momento como se muestra en la figura 9.8.

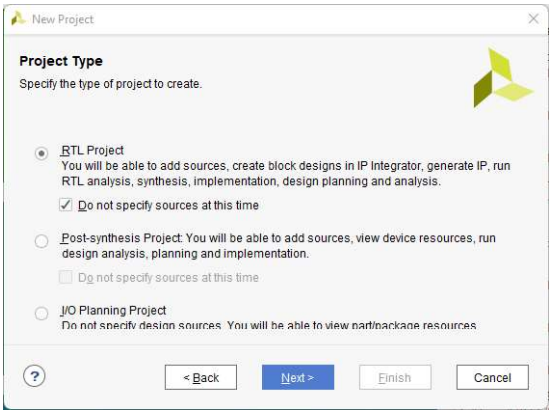
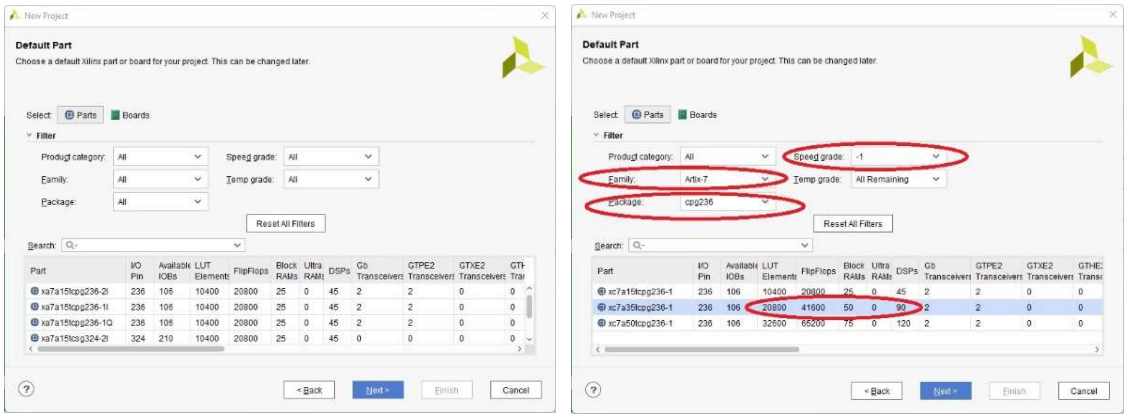


Figura 9.8 Ventana para definir el tipo de proyecto.

5. La siguiente ventana es para definir el dispositivo de la tarjeta (figura 9.9 a), el cual corresponde a la familia *Artix-7*, de empaquetado *cpg236* con grado de velocidad *-1*. Esto dará 3 opciones en la lista de partes, selecciona la opción de 20800 LUTs (figura 9.9 b) y dar clic en siguiente con lo que aparecerá una ventana de resumen (**imagen no mostrada**) a la cual hay que darle clic en finalizar.



a) *b)*

Figura 9.9 Ventana de Selección del dispositivo.

6. Terminados los pasos anteriores, se abre la interfaz de ambiente integrado (IDE) de Vivado (figura 9.10 anotación 1), en la que hay que darle clic en agregar fuentes (Add sources) lo que abrirá la ventana correspondiente (figura 9.10 anotación 2). Asegurarse que la opción de agregar o crear fuentes de diseño está seleccionada y dar clic en siguiente.

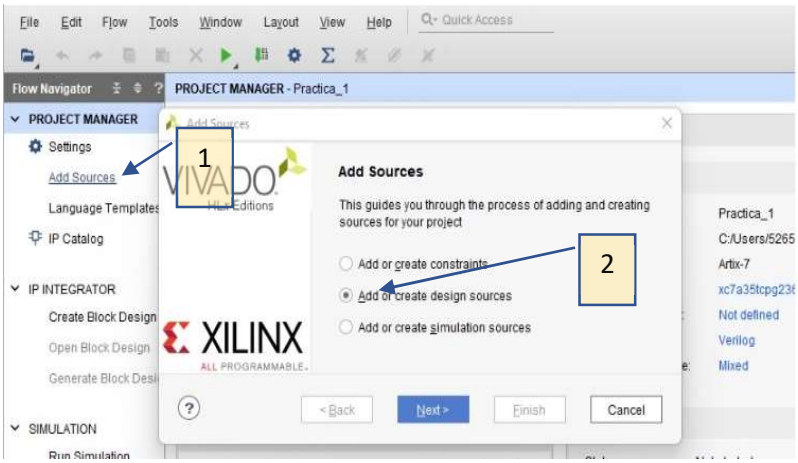


Figura 9.10 Ventana para agregar fuentes de diseño.

7. En la siguiente ventana, seleccionar la opción crear archivo (figura 9.11 anotación 1)

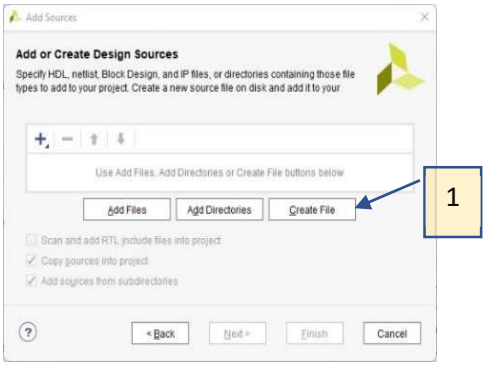


Figura 9.11 Ventana para agregar archivos fuente.

8. Se abrirá la ventana de crear archivo fuente, asegurar que el tipo de archivo es VHDL y dale el nombre de Practica_9 como se muestra en la figura 9.12. Dar clic en "OK".

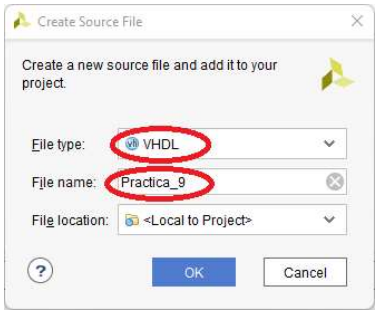
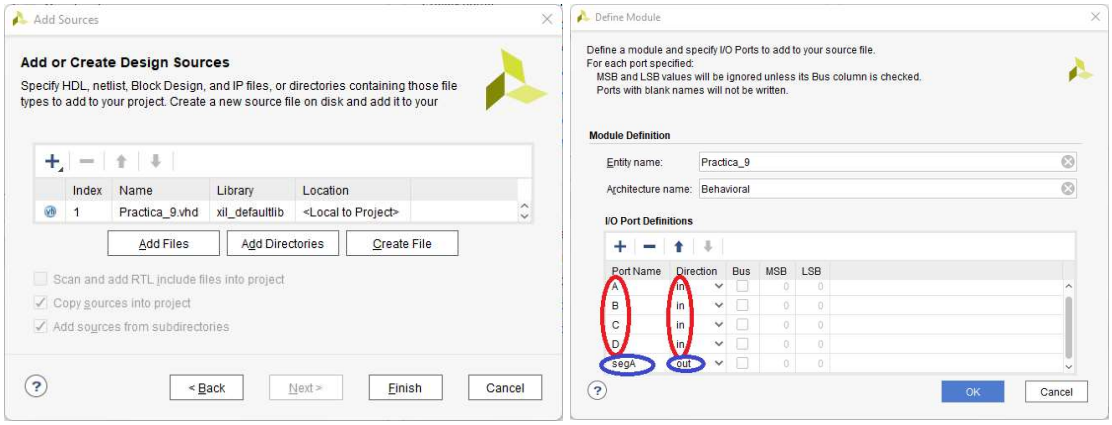


Figura 9.12 Creación del archivo fuente.

9. Al cerrarse la ventana anterior, regresamos a la ventana de agregar fuentes, obsérvese que ya tiene agregado el archivo Practica_9 (figura 9.13 a); le damos clic en el botón Terminar para que nos abra la ventana de Definir Módulo, en la cual agregamos la interfaz mostrada en la figura 9.13 b), que consiste en tres entradas llamadas A, B, C y D (Óvalos de color rojo)

y de una salida, para el segmento **a** del display de 7 segmentos, llamada segA (Óvalos de color azul). Dar clic en el botón "OK" para terminar.



a) *b)*

Figura 9.13 Finalización de agregar archivo fuente.

10. De regreso a la IDE de Vivado, se observa que en la ventana fuentes aparece el archivo recién agregado *Practica_9*, como lo indica la figura 9.14.

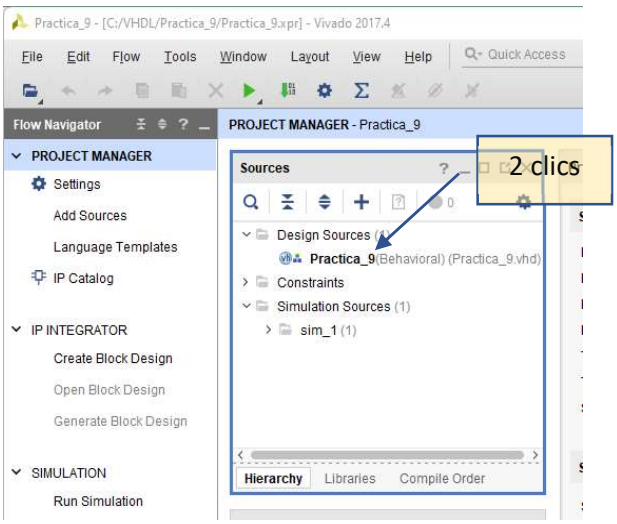


Figura 9.14 Archivo Practica_9 agregado en ventana fuentes.

11. Para iniciar la edición del archivo de diseño, dar 2 clics sobre la referencia al archivo *Practica_9*, dentro de la ventana fuentes, como indica la figura 9.14 y se abrirá el archivo para edición como se muestra en la figura 9.15.

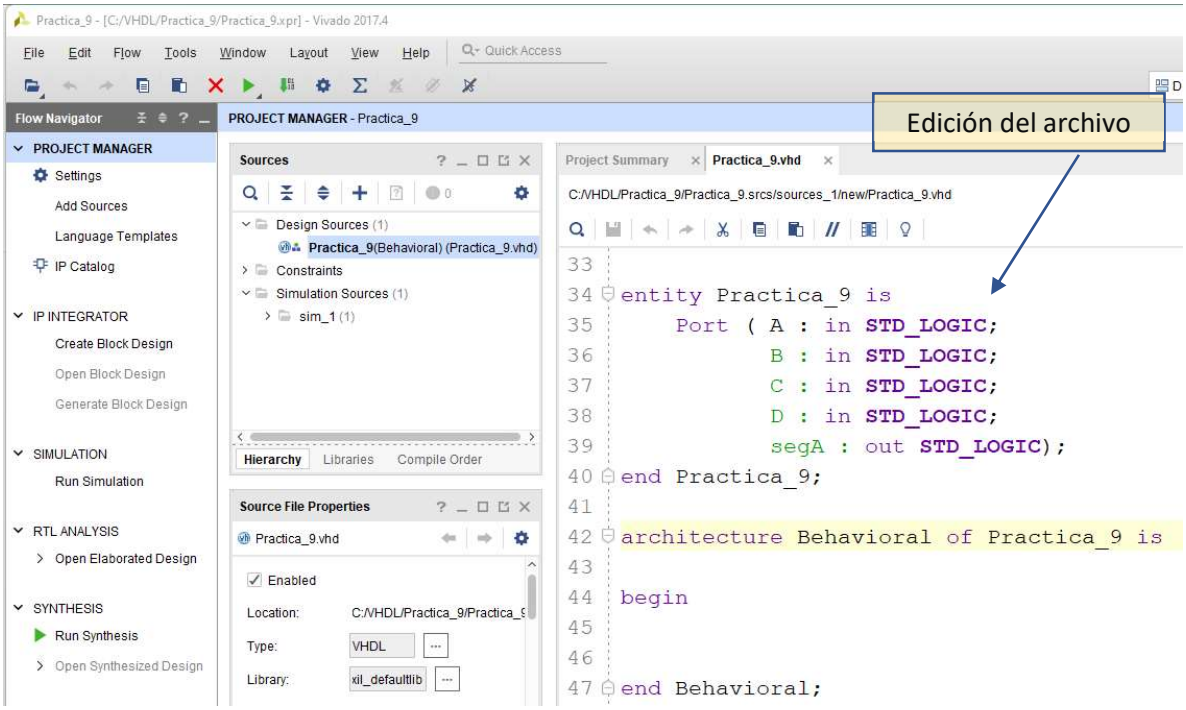


Figura 9.15 Ventana de edición del archivo de diseño.

12. Agregamos a la arquitectura del proyecto la función del segmento **a**, quedando el código como se muestra en la figura 9.16. Guardar los cambios realizados al proyecto (Ctrl+s).

```
entity Practica_9 is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           D : in STD_LOGIC;
           segA : out STD_LOGIC);
end Practica_9;

architecture Behavioral of Practica_9 is

begin
    segA <= ((D xnor C) and not B and A) or
    (not D and C and not B and not A) or
    (D and not C and B and A);
end Behavioral;
```

Figura 9.16 Función del segmento a codificada en la arquitectura del proyecto.

Simulación del proyecto

Los siguientes pasos indican como agregar el archivo de simulación y ejecutarlo para visualizar el diagrama de tiempo resultante:

1. Da clic en agregar fuentes como se indica en la figura 5.15 nota 1, lo cual abre una ventana con tres opciones, asegurarse de que se escoge la opción de agregar una fuente de simulación como se indica en la figura 5.15 nota 2 dando lugar a la ventana de agregar fuente.

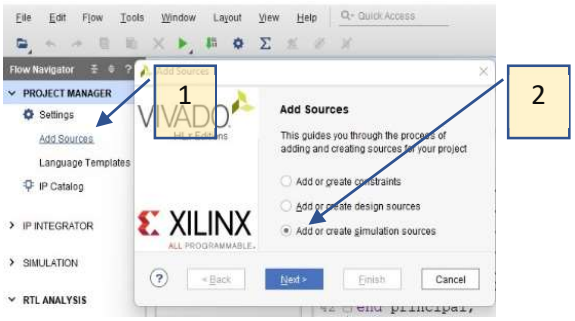
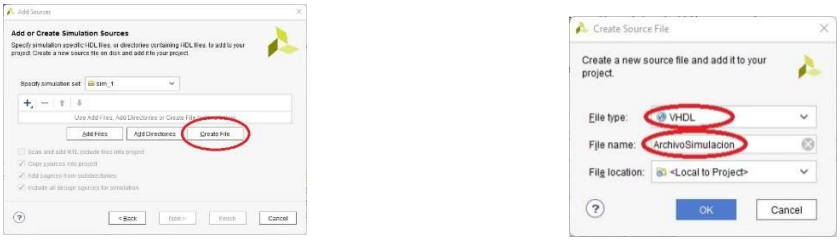


Figura 9.17Pasos para agregar archivo de simulación.

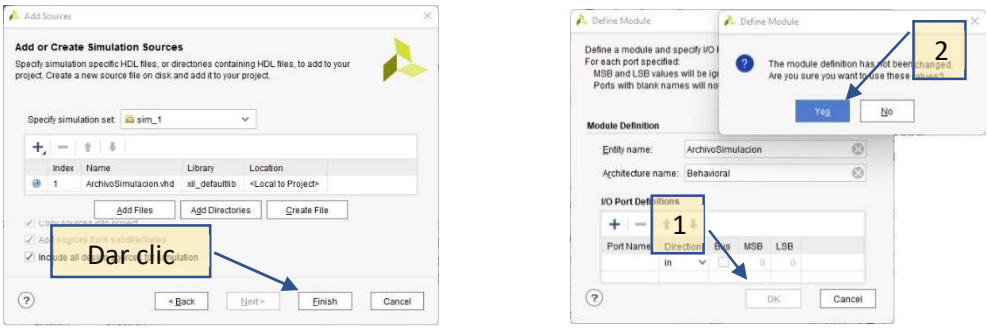
2. En la ventana agregar fuente seleccionar crear archivo como se indica en la figura 9.18 a) con lo que se abrirá la ventana de crear archivo fuente y deberá especificarse el tipo del archivo VHDL y como nombre sugerido *ArchivoSimulacion* como se indica en la figura 9.18 b); dar clic en el botón "*OK*".



a) *b)*

Figura 9.18 Ventanas para agregar archivo de simulación.

3. Cerrar la ventana de agregar fuentes dando clic en el botón finalizar como se indica en la figura 9.19 a) se abrirá una ventana para definir módulos, dar clic en "*OK*" y en la siguiente ventana darle clic en "*Yes*" como se muestra en la figura 9.19 b).



a) *b)*

Figura 9.19 Clics en ventanas para finalizar agregar fuente.

4. Volviendo a la IDE de Vivado, para realizar la edición del archivo de simulación, dar clic en el nombre del archivo "*ArchivoSimulacion*" que se encuentra en la ventana fuentes y observar que en la ventana de edición estará el archivo como lo muestra la figura 9.20.

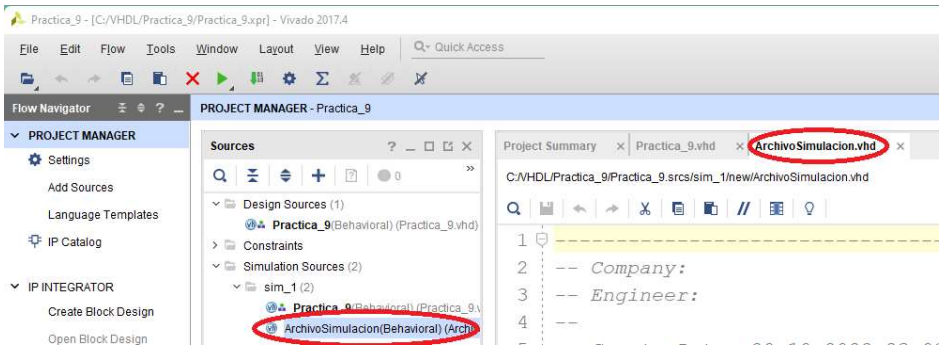


Figura 9.20 Apertura de la edición del archivo de simulación.

5. Siga las instrucciones de la práctica 5 para agregar el componente en la arquitectura del archivo de simulación, así como las variables, la asignación de las variables al componente y los cambios de las variables en forma temporizada.

6. Para realizar la simulación, dar clic al submenú "Correr simulación" del menú simulación como se muestra en la figura 9.21 y seleccionar en el menú desplegable "Correr simulación del comportamiento".

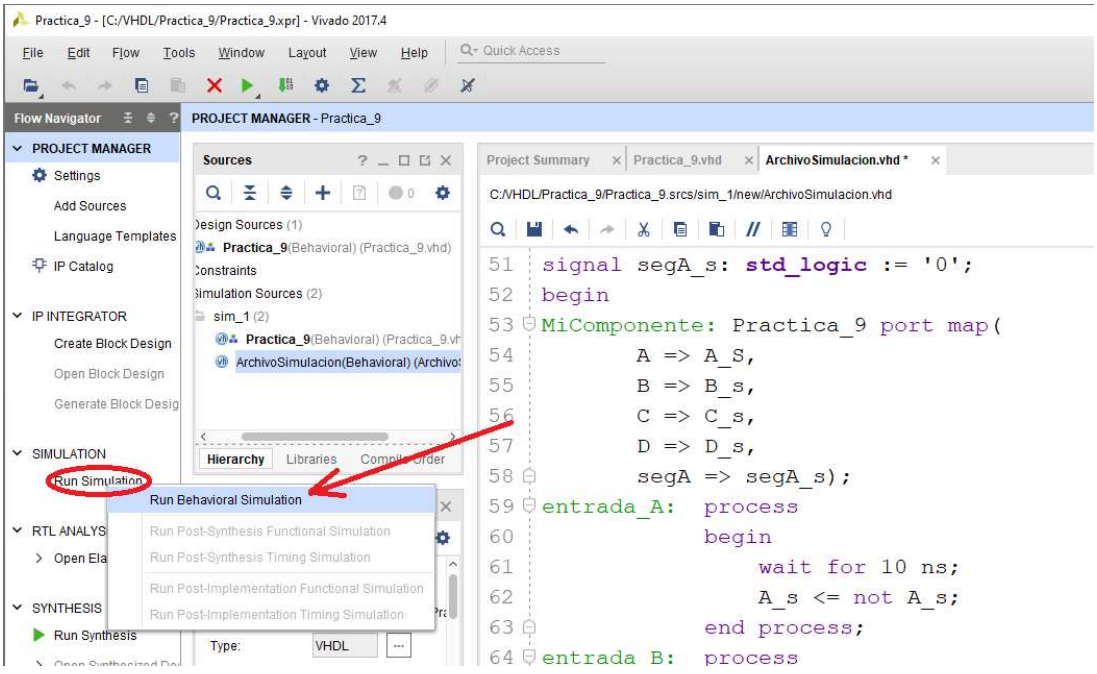


Figura 9.21 Submenú para realizar la simulación.

21. Seleccione las entradas en el simulador y cree un Bus Virtual, dándole clic con el botón derecho y seleccionando del menú contextual la opción 

. El diagrama de tiempo deberá de parecerse a lo mostrado en la figura 9.22.

Figura 9.22 Ventanas de la simulación e intervalo de tiempo.

Compare la salida del diagrama de tiempo con el segmento **a** de la tabla de verdad, y asegúrese de que los valores dados para el segmento **a** en ceros y unos en la tabla de verdad son los mismos del diagrama de tiempo.

Programación de la tarjeta de Basys 3

Para la programación de la tarjeta de Basys 3, baje del Campus Virtual y copie la información que se encuentra en el archivo "Archivo configuración (XDC) Basys 3" y pegue la información en el archivo de restricciones que agregue al proyecto siguiendo las instrucciones de la práctica 8; remueva los comentarios a las líneas que vaya a utilizar y modifique el nombre de las variables para que coincida con los nombres de las variables del proyecto. Una vez listo el archivo de restricciones en el proyecto, realizar la grabación de la tarjeta Basys 3.

Desarrollo

Para el proyecto, simulación en Vivado y programación de la tarjeta Basys 3, agregue los segmentos restantes siguiendo el procedimiento de minimización con mapa de Karnauh, optimización de la función, programación en VHDL de las funciones de acuerdo con lo presentado en la sección Análisis previo.

Preguntas

1. ¿Cuál es la diferencia entre un display de cátodo común y un display de ánodo común?
2. ¿Por qué se dice que los cuatro display del módulo de la tarjeta Basys 3 están multiplicados?

3. Si se quisiera tener el control de que display se prende y que display se apaga por medio de interruptores de la placa Basis 3, ¿Cuáles serían las terminales en el archivo de restricciones que se tendrían que asignar?
4. ¿Qué tipo de lógica utilizan las terminales que controlan que display se enciende o se apaga?
5. ¿Cuál sería el cambio que se debe de realizar en la presente práctica si en vez de querer utilizar una entrada hexadecimal se manejará una entrada decimal?, explica ampliamente.

Referencias

1. Llanos, Juan; "Vivado Tutorial 1 Crear un proyecto", https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; "Vivado Tutorial 2 Simulación", https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.

Práctica # 10 Divisor de frecuencia para Basys 3 (Ejem. A)

Objetivos de la práctica # 10

1. Conocer y programar en VHDL un divisor de frecuencia.
2. Conocer y utilizar la base de tiempo que proporciona la Basys 3.
3. Utilizar la base de tiempo de la Basys 3 para generar una frecuencia de 1 Hz.

Material y equipo

Enseguida se muestra en la tabla 10.1 los componentes y equipo a utilizar en la presente práctica, que comprenderá la programación en VHDL junto con el grabado de la tarjeta Basys 3 con el programa generado.

Tabla 10.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado
1	Tarjeta Basys 3
1	Cable USB para Basys 3

Introducción

Los dispositivos de división de frecuencia se utilizan cuando una señal de alta frecuencia, como un reloj, necesita reducirse a una frecuencia más baja para ser utilizada en otros dispositivos. Los divisores de frecuencia se pueden diseñar para reducir esta frecuencia mediante cantidades variables, pero debe ser a través de algún divisor íntegro y no se pueden utilizar con ningún otro valor. Cuando se realiza esta conversión, las señales de salida se pueden utilizar para conducir otros componentes, que no pueden operar en la misma alta frecuencia registrada inicialmente. Por lo general, los divisores de frecuencia se utilizan para generar un reloj de referencia para los bucles enganchados en fase que operan en un circuito.

Estos dispositivos se diseñan utilizando una variedad de técnicas que implican compuertas lógicas y registros. Utilizan diferentes combinaciones de estos dispositivos que permiten modificaciones de altas frecuencias. Cuando la señal modificada de un divisor de frecuencia se alimenta a un contador o temporizador, actúa como una nueva frecuencia del reloj para

estos dispositivos. Un circuito puede tener muchos de estos dispositivos reduciendo un reloj mundial a diferentes frecuencias, dependiendo de las necesidades del dispositivo.

Análisis previo

Se dice divisor de frecuencia un circuito que recibe en entrada una señal de una frecuencia determinada *f* y da una señal de salida de frecuencia *f/n* donde *n* es un número entero. La necesidad de un divisor de frecuencia es, ya que tiene tanto la frecuencia dividida y la misma señal de reloj, es que debe conducir circuitos en diferentes frecuencias, y porque es más fácil para estabilizar por medio de un circuito en el cuarzo un circuito dado a una tasa superior y luego obtener una frecuencia más baja, que también se a estabilizado, aunque no es un cristal de cuarzo a la frecuencia deseada.

Conectando en cascada múltiples *flip-flops* de tipo *T* se puede obtener divisores de frecuencia múltiplos de 2 de acuerdo con la siguiente fórmula:

$$fn = \frac{f}{2^n}$$

donde *n* es un número entero. Deseando obtener un divisor de 2 o de 4, podemos utilizar el esquema de la figura 10.1 a) y b).

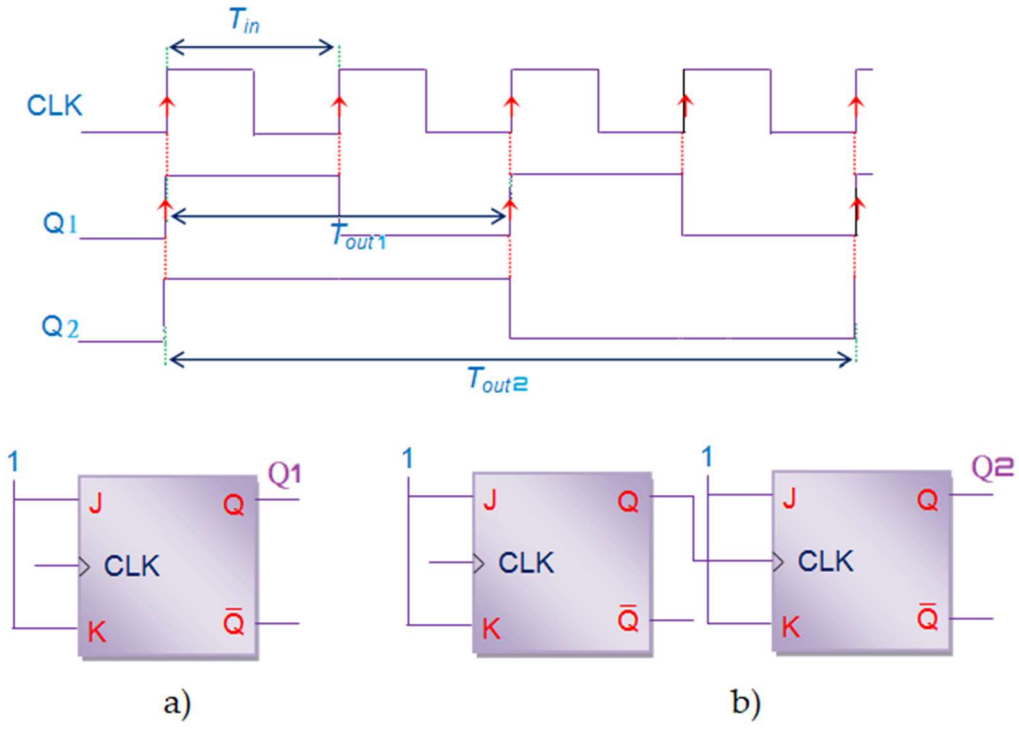


Figura 10.1 Divisor de frecuencia con a) un FF y b) 2 FF.

Si se desea, en su lugar, obtener un divisor que no sea de potencias de 2, debemos de contar los impulsos, cuando se ha alcanzado el número deseado, como se indica en la tabla 10. 2.

Tabla 10.2 Valores de referencia para obtener la división en valores independientes de potencias de 2.

División	Q2	Q1	Q0
N/A	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0
9	0	0	1

Si se desea una división de frecuencia de 5_{10}, entonces el divisor deberá de reiniciar en la posición 5, es decir, $(Q_2\ Q_1\ Q_0) = 101_2$. Recordando que el divisor de frecuencia es un contador, por lo cual, para reiniciar en la cuenta número 5, se utilizará una compuerta A**nd** o **Nand**, dependiendo si las terminales de reinicio de los FF son *no negadas* o *negadas*.

Para la programación de la tarjeta Basys 3, utilizaremos el último método descrito, para lo cual agregaremos una señal del tipo entero que estaremos incrementando en cada flanco ascendente de la base del tiempo de la tarjeta. Para la interfaz del divisor de frecuencia se considerarán las siguientes señales, **clk** para la señal base de tiempo proporcionada por la tarjeta, **reset** para reiniciar y mantener en cero la división de frecuencia, y **clk_out** como la salida del divisor de frecuencia; es importante hacer notar que hay que agregar la biblioteca (`use) IEEE.numeric_std.ALL;` con el fin de realizar operaciones aritméticas con y sin signo y los tipos numéricos. La figura 10.2 muestra la entidad del divisor de frecuencia.

```
entity Divider is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        clk_out : out STD_LOGIC);
end Divider;
```

Figura 10.2 Entidad del divisor de frecuencia.

En la figura 10.3 se muestra la arquitectura del divisor, donde se puede observar que la variable **count** es de tipo entero por lo que su rango permitirá la división de la base de tiempo proporcionada la tarjeta, la cual es de 50 Mhz.

```
architecture Behavioral of Divider is
    signal count: integer:=1;
    signal tmp : std_logic := '0';
begin
    process(clk,reset)
    begin
        if(reset='1') then
            count<=1;
            tmp<='0';
        --elsif rising_edge(clk) then
        elsif (clk'event and clk='1') then
            count <=count+1;
            if (count = 5) then
                tmp <= NOT tmp;
                count <= 1;
            end if;
        end if;
    end process;
    clk_out <= tmp;
end Behavioral;
```

Figura 10.3 Arquitectura del divisor de frecuencia.

Como podrás observar en la figura 10.3, la división de frecuencia se está realizando en una cuenta de 5, por lo que hay que determinar cuál es la división de frecuencia real de acuerdo con la condición que incrementa la variable de conteo. En la figura 10.4 se muestra la porción superior de la arquitectura utilizada para el banco de pruebas del divisor de frecuencia. Está incluye el componente del divisor de frecuencia, así como las señales necesarias para su instanciación.

```
architecture Behavioral of prueba is
    COMPONENT Divider
    PORT(
        clk : IN std_logic;
        reset : IN std_logic;
        clk_out : OUT std_logic
    );
    END COMPONENT;

    --Entradas
    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    -- Salida
    signal clk_out : std_logic;
    -- Definicion del periodo del reloj
    constant clk_period : time := 20 ns;
```

Figura 10.4 Declaración del componente del divisor y las señales necesarias para la instantación del componente.

El código de la codificación correspondiente a la arquitectura de la simulación se da en la figura 10.5, comenzando con la instrucción **begin**; se puede observar, como primera porción del código, la instanciación del componente.

```
BEGIN
-- Instanciacion del componente
uut: Divider PORT MAP (
    clk => clk,
    reset => reset,
    clk_out => clk_out
);
```

Figura 10.5 Porción inicial de la codificación de la arquitectura.

Finalmente, en la figura 10.6 se muestran los procesos correspondientes a la creación del tren de pulsos que fungirá como la base de tiempo y la activación de los estímulos, correspondiendo en este caso a la señal de reinicio.

```
clk_process :process
begin
    clk <= '1';
    wait for clk_period/2;
    clk <= '0';
    wait for clk_period/2;
end process;
-- Estimulos
process
begin
    wait for 200 ns;
    reset <= '0';
    wait for 104 ns;
    reset <= '1';
    wait for 100 ns;
    reset <= '0';
    wait;
end process;
end Behavioral;
```

Figura 10.6 Porción final de la codificación de la arquitectura para la simulación del divisor.

Desarrollo

Realizar y reportar las siguientes actividades:

1. Realizar un proyecto en Vivado para la tarjeta Basys 3, que corresponda al código mostrado en la sección de Análisis Previo; Realiza la simulación del divisor de frecuencia y explica las señales resultantes en el diagrama de tiempos, tanto con los cambios de la señal de reinicio, así como el de la frecuencia resultante del divisor.

2. Modifica la división de la frecuencia para que al programar la tarjeta Basys 3, prenda y apague un led con una frecuencia de 1 Hz.

Preguntas

1. ¿Cuál es el valor que hay que utilizar en el divisor de frecuencia para obtener una frecuencia de salida de 1 Hz, sabiendo que la base de tiempo de la tarjeta Basys 3 es de 50 Mhz?
2. Explica por qué no se utilizó el valor de 50 000 000 en el divisor de frecuencia para obtener la frecuencia de 1 Hz.
3. Escribe 2 formas de detectar el flanco de transición de subida en la codificación.
4. Indica si la señal de reinicio es síncrona o asíncrona, y qué habría que hacer para cambiar su condición.
5. Describe la funcionalidad del paquete de biblioteca `IEEE.numeric_std.ALL`.

Referencias

1. Llanos, Juan; “Vivado Tutorial 1 Crear un proyecto”, https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; “Vivado Tutorial 2 Simulación”, https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; “Manual de Prácticas de Circuitos Digitales”, Universidad Autónoma de Ciudad Juárez, diciembre 2020.

Práctica # 11 Contador basado en divisor de frecuencia (Ejem. B)

Objetivos de la práctica # 11

1. Conocer y diversificar el uso de la base de tiempo de la tarjeta Basys 3.
2. Conocer y modificar el código para un decodificador en VHDL.
3. Conocer y modificar el código para un multiplexor en VHDL.

Material y equipo

Enseguida se muestra en la tabla 11.1 los componentes y equipo a utilizar en la presente práctica, que comprenderá la programación en VHDL junto con el grabado de la tarjeta Basys 3 con el programa generado.

Tabla 11.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado
1	Tarjeta Basys 3
1	Cable USB para Basys 3

Introducción

Un contador es un circuito secuencial construido a partir de biestables y puertas lógicas capaces de almacenar y contar los impulsos (a menudo relacionados con una señal de reloj), que recibe en la entrada destinada a tal efecto, así mismo también actúa como divisor de frecuencia. Normalmente, el cómputo se realiza en código binario, que con frecuencia será el binario natural o el BCD natural (contador de decenas). Ejemplo, un contador de módulo 4 pasa por 4 estados, y contaría del 0 al 3. Si necesitamos un contador con un módulo distinto de 2^n, lo que haremos es añadir un circuito combinacional. La finalidad de un contador es incrementar y/o decrementar la cantidad de dicho número por cada pulso de reloj que percibe. Los códigos de salida pueden ser de tipo binario generalmente, aunque también pueden existir con salidas hexadecimales, octales, entre otros.

Los contadores se pueden clasificar por diferentes características, pero una de las más importantes es su tipo de funcionamiento. Los contadores pueden ser de tipo síncrono y de tipo asíncrono.

Contadores de tipo asíncrono.

Consisten en circuitos formados por *flip-flops* tipo J-K puenteados (Realizando un puente entre las entradas J y K) organizados en cascada, es decir, la salida Q de un *flip-flops* va a la entrada J-K del siguiente *flip-flops*. Cada *flip-flops* representa un bit o espacio para el código. Siendo el número de *flip-flops* igual a N y utilizando las potencias base dos, la magnitud del contador en binario será igual la potencia base 2 elevada al número N de *flip-flops* que constituyan al contador. La desventaja de este tipo de contadores es que solo puede realizar secuencias en serie de tipo ascendentes y/o de tipo descendente.

Contadores de tipo síncrono.

Están conformados principalmente por *flip-flops* que poseen señal de reloj, las cuales se activan de manera simultánea ejecutando cada *flip-flops* al tiempo del pulso cuadrado o de reloj. También se puede decir que las operaciones de los *flip-flops* son ejecutadas en forma paralela, a diferencia de los contadores de tipo asíncrono que son de manera serial. Este tipo de contadores es mucho más flexible que los contadores asíncronos.

Otro tipo de clasificación es según su función, los cuales pueden ser Contadores Up/Down, que pueden contar de manera ascendente y descendente; contadores de décadas, poseen una salida de tipo BCD(Código Binario Decimal, por sus siglas en inglés), por lo cual tienen una salida "directamente" decimal al ser usadas con display de este tipo (Por ejemplo, display 7-SEG).

En la figura 11.1 se puede apreciar la configuración para un contador asíncrono mientras que en la figura 11.2 se muestra la configuración para un contador síncrono.

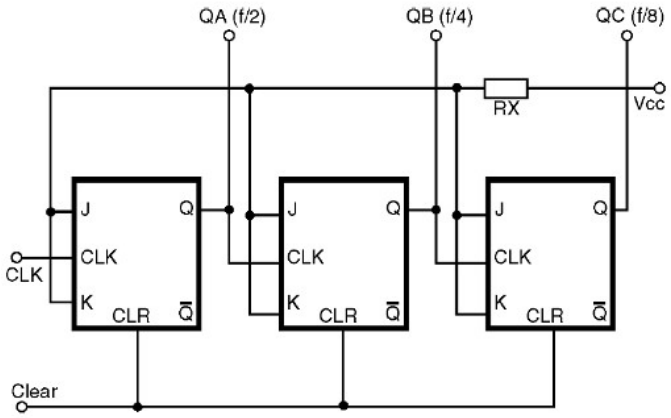


Figura 11.1Configuración para un contador asíncrono de tres etapas.

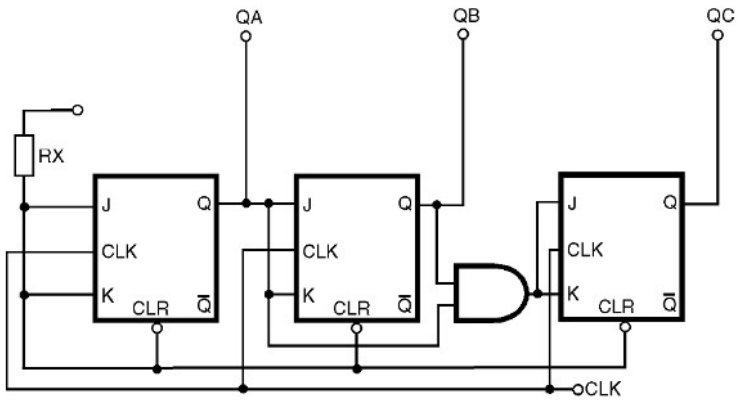


Figura 11.2 Configuración para un contador síncrono de tres etapas.

Análisis previo

Creamos un proyecto en Vivado y le agregamos un archivo de diseño VHDL llamado "practica_11" y un archivo de restricciones para la asignación de terminales "Conf_term". El resultado en la ventana de fuentes será la mostrada en la figura 11.3.

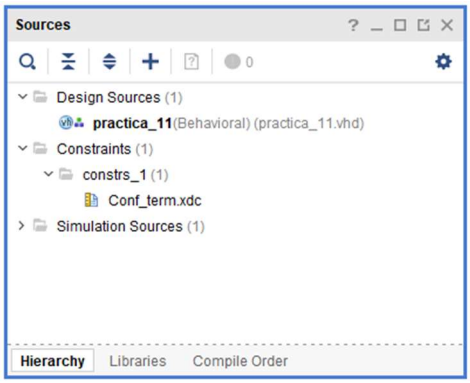


Figura 11.3 Ventana de archivos fuente con el archivo de diseño y el de restricciones.

Dado que el contador se realizará con una variable del tipo **integer** que se estará incrementando por medio de la operación suma, se requiere de utilizar la biblioteca **numeric_std**, lo cual se logra agregando la siguiente línea al código.

```
Use IEEE.numeric_std.ALL;
```

La entidad del proyecto tendrá una entrada para la base de tiempo de la tarjeta Basys 3 y una salida de 2 bits, donde se tendrá el conteo de cero a 3 en binario, como se indica en la figura 11.4, donde también se muestra la inserción de las bibliotecas necesarias para el funcionamiento del proyecto.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity practica_11 is
    Port ( Clk : in STD_LOGIC;
           Salida : out STD_LOGIC_VECTOR (1 downto 0));
end practica 11;
```

Figura 11.4 Código correspondiente a las bibliotecas utilizadas y la entidad del proyecto.

El código de la arquitectura del proyecto se muestra en la figura 11.5.

```
architecture Behavioral of practica_11 is
    signal Divisor : integer := 0;
    signal Contador : STD_LOGIC_VECTOR (1 downto 0) := "00";
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            Divisor <= Divisor + 1;
        end if;
        if(Divisor = 9000000) then
            Divisor <= 0;
            if(Contador = "01") then
                Contador <= "10";
            elsif(Contador = "10") then
                Contador <= "11";
            elsif(Contador = "11") then
                Contador <= "00";
            else
                Contador <= "01";
            end if;
        end if;
    end process;
    Salida <= Contador;
end Behavioral;
```

Figura 11.5 Código de la arquitectura del proyecto.

La codificación de la arquitectura del proyecto requiere de una señal (denominada Divisor en el código) para realizar la actividad de división de frecuencia; el proceso donde se realiza este incremento debe de ser sensible a los cambios de la señal **clk**, por lo que la señal es agregada a la lista de variables sensitivas por medio de los paréntesis del comando **process**. El conteo se lleva a cabo en la señal Contador, que se asigna por medio de una estructura **if-elsif**. El contenido del archivo para la simulación es mostrado en la figura 11.6 y el contenido del archivo de restricciones se muestra en la figura 11.7.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity x is
--  Port ( );
end x;
architecture Behavioral of x is
component practica_11
    Port ( Clk : in STD_LOGIC;
           Salida : out STD_LOGIC_VECTOR (1 downto 0));
end component;
    signal Clk : STD_LOGIC:= '0';
    signal Salida : STD_LOGIC_VECTOR (1 downto 0):="00";
    constant clk_period : time := 20 ns;
begin
-- Instanciacion del componente
uut: practica_11 PORT MAP (clk => clk, Salida => Salida );
clk_process :process
    begin
        clk <= '1';
        wait for clk_period/2;
        clk <= '0';
        wait for clk_period/2;
    end process;
end Behavioral;
```

Figura 11.6 Contenido del archivo para la simulación.

```
## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {Salida[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {Salida[1]}]

## Señal de reloj
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports Clk]
```

Figura 11.7 Contenido del archivo de restricciones.

Desarrollo

Realizar y reportar las siguientes actividades:

1. Cambiar el contador de 2 bits a 3 bits en el proyecto mostrado.

2. Realizar un proyecto en Vivado para la tarjeta Basys 3, que corresponda al código mostrado en la sección de Análisis Previo. Realiza la simulación del divisor de frecuencia y explica las señales resultantes en el diagrama de tiempos, tanto con los cambios de la señal de reinicio, así como el de la frecuencia resultante del divisor. Nota: no olvides bajar la frecuencia a un valor muy bajo para que lo puedas ver en el simulador.

3. Modifica la división de la frecuencia para que al programar la tarjeta Basys 3, prenda y apague un led con una frecuencia de 100 Hz.

Referencias

1. Llanos, Juan; "Vivado Tutorial 1 Crear un proyecto", https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; "Vivado Tutorial 2 Simulación", https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.

Práctica # 12 Implementación de componentes en VHDL

Objetivos de la práctica # 12

1. Conocer e implementar los componentes en un proyecto VHDL.
2. Conocer y modificar el código para un decodificador en VHDL.
3. Conocer y modificar el código para un multiplexor en VHDL.

Material y equipo

Enseguida se muestra en la tabla 12.1 los componentes y equipo a utilizar en la presente práctica, que comprenderá la programación en VHDL junto con el grabado de la tarjeta Basys 3 con el programa generado.

Tabla 12.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado
1	Tarjeta Basys 3
1	Cable USB para Basys 3

Introducción

Los módulos creados en VHDL pueden utilizarse en diferentes diseños, lo que permite la reutilización del código. Además, la misma descripción puede utilizarse para diferentes tecnologías sin tener que rediseñar todo el circuito. Esta descripción utiliza para la creación de la arquitectura de la entidad entidades descritas y compiladas previamente, de esta manera en VHDL podemos aprovechar diseños ya realizados, o realizar diseños sabiendo que se utilizarán en otros más complicados. Así se ahorra trabajo al diseñador-programador. Se declaran los componentes que se van a utilizar y después, mediante los nombres de los nodos, se realizan las conexiones entre los puertos. Las descripciones estructurales son útiles cuando se trata de diseños jerárquicos *botton-up*. Se pueden utilizar tantos componentes como necesite el diseño, estos componentes son entidades anteriormente declaradas o compiladas, en el ejemplo, subcircuito debe ser el nombre de una entidad que se ha declarado anteriormente en el código o que se ha compilado como parte de otro fichero VHDL. Se puede utilizar un componente tantas veces como sea

necesario (podemos asignar chip_i y chip_j al mismo componente, pero con distintas señales de entrada y salida). Podemos realizar un diseño estructural puro, mediante la unión de componentes que describen las distintas funcionalidades del circuito, en el que necesitaremos definir las señales intermedias entre las que se encontrará siempre las señales que interconectan las salidas de los módulos con las salidas de la *entity*.

Un decodificador tiene como función detectar la presencia de una determinada combinación de bits en sus entradas y señalar la presencia de este código mediante un cierto nivel de salida. Un decodificador posee N líneas de entrada para gestionar N bits y en una de las 2N líneas de salida indica la presencia de una o más combinaciones de n bits. Para cualquier código dado en las entradas solo se activa una de las N posibles salidas. El decodificador Hexadecimal convierte cada código Hexadecimal en uno de los 16 posibles dígitos Hexadecimales. El decodificador BCD a 7 segmentos es un circuito combinacional que permite un código BCD en sus entradas y en sus salidas activa un display de 7 segmentos para indicar un dígito decimal. El display está formado por un conjunto de 7 leds conectados en un punto común en su salida, bien en ánodo común o ánodo común. Un multiplexor o también conocido como MUX o MPX es un dispositivo que sirve para transmitir datos de diferentes entradas a una sola salida, es decir, todos los datos que entran al circuito salen por el mismo lugar, dependiendo del tipo que se utilice, pueden manejar señales analógicas o digitales. El funcionamiento de los multiplexores se basa en circuitos de compuertas lógicas, en donde se conectan de tal forma que todas las entradas salen por la misma salida, con la única condición de que se debe de seleccionar la entrada que mandara los datos hacia la salida, es decir, que el circuito no puede leer todas las entradas al mismo tiempo, si no una por vez. En palabras sencillas, este circuito funciona como un conmutador que solo permite seleccionar la lectura de una de las entradas, pero debido a la gran velocidad de conmutación que tiene el circuito, se puede llegar a creer que puede leer todas las entradas al mismo tiempo. En la figura 12.1 se muestra el diagrama de un multiplexor de cuatro entradas (**A** a **D**) con 2 líneas de selección (con 2 líneas de selección se generan cuatro combinaciones, como cuatro son las entradas en este caso), y al generar la combinación correspondiente a una de las entradas, dicha entrada es colocada a la salida llamada **M**.

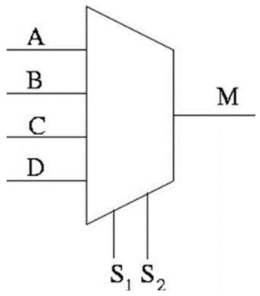


Figura 12.1 Diagrama de un multiplexor de 4 entradas a una salida.

Análisis previo

Enseguida se realizará un proyecto utilizando módulos en VHDL, el proyecto consiste en recibir un valor binario de cuatro bits, y este valor será representado en forma decimal en el cuarto display de 7 segmentos de la tarjeta Basys 3 y la cantidad de unos que contenga el valor será mostrado en el primer display de 7 segmentos. En la figura 12.2 se muestra el diagrama a bloques del proyecto.

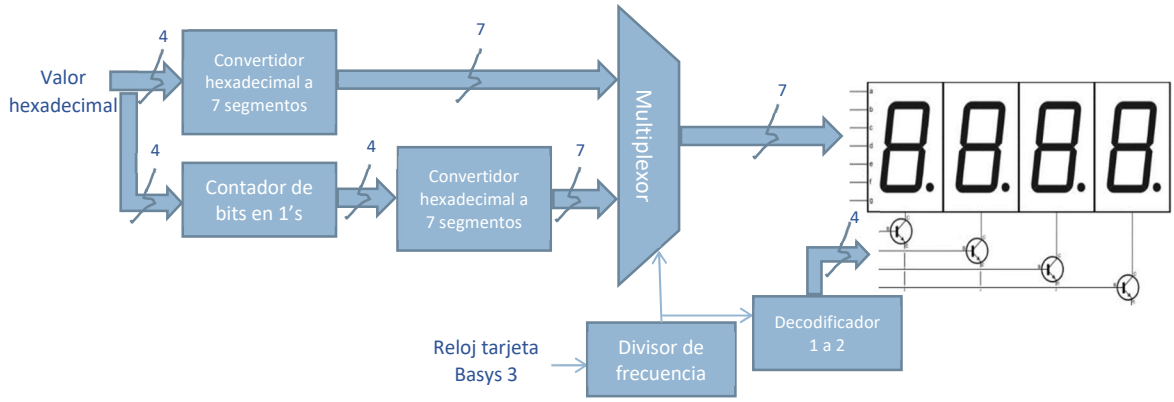


Figura 12.2 Diagrama a bloques del proyecto.

De acuerdo con el diagrama bloques del proyecto, como se muestra en la figura 12.2, se tendrán que definir los siguientes módulos:

a) Módulo Hexadecimal a 7 segmentos; recibirá un valor hexadecimal de cuatro bits y regresará su codificación a 7 segmentos. Como se puede observar en el diagrama, se realizarán 2 instancias de este módulo, uno para el valor hexadecimal entrante y otro para la cuenta de bits en 1's.

b) Módulo Contador de bits en 1's; recibirá un valor hexadecimal de cuatro bits y regresará la cuenta de los bits que se encuentren con valor '1'.

c) Módulo Divisor de frecuencia; recibirá la señal de reloj de la tarjeta Basys 3 y lo dividirá a 100 Hz.

d) Módulo Decodificador 1 a 2; recibe la señal de reloj, decodificando el valor binario a 2 salidas; sin embargo, la salida está compuesta de cuatro bits, 2 de ellos permanecen fijos en alto para apagar los display de 7 segmentos centrales (1110 y 0111).

e) Módulo Multiplexor 2 a 1 con longitud de palabra de 7 bits, con el que se multiplexara los 2 valores codificados a 7 segmentos a la entrada del módulo de 4 display's.

En la figura 12.3 se muestra la entidad del módulo CodHex7Seg, la codificación de la arquitectura es la realizada en la práctica #9.

```
entity CodHex7Seg is
    Port (
       Hexa : in STD_LOGIC_VECTOR (3 downto 0);
       seg7 : out STD_LOGIC_VECTOR (6 downto 0));
end CodHex7Seg;
```

Figura 12.3 Entidad de la codificación a 7 segmentos.

La figura 12.4 muestra la entidad del módulo Divisor, la codificación de la arquitectura es la realizada en la práctica # 10, omitiendo el puerto **reset**.

```
entity Divisor is
    Port (
        clk : in STD_LOGIC;
        clk_out : out STD_LOGIC);
end Divisor;
```

Figura 12.4 Entidad del divisor de frecuencia.

El módulo que realiza la cuenta de bits hace Uso de la biblioteca STD_LOGIC_UNSIGNED para realizar las sumas en el vector de tipo STD_LOGIC_VECTOR. el código completo se muestra en la figura 12.5.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CuentaBits is
    Port ( Bits : in STD_LOGIC_VECTOR (3 downto 0);
           Cuenta : out STD_LOGIC_VECTOR (3 downto 0));
end CuentaBits;
architecture Behavioral of CuentaBits is

begin
    Cuenta <= "0000" + Bits(0) + Bits(1) + Bits(2) + Bits(3);
end Behavioral;
```

Figura 12.5 Código del módulo para realizar la cuenta de bits en 1's.

El decodificador de 1 a 2 líneas se muestra en la figura 12.6. Se hace notar que tiene cuatro líneas de salida (2 líneas fijas en '1') para conectarse al módulo de cuatro display's de la tarjeta Basys 3.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Decodifica is
    Port ( Bin : in STD_LOGIC;
           Linea : out STD_LOGIC_VECTOR (3 downto 0));
end Decodifica;
architecture Behavioral of Decodifica is
begin
    process(Bin)
    begin
        if Bin = '0' then
            Linea <= "1110";
        else
            Linea <= "0111";
        end if;
    end process;
end Behavioral;
```

Figura 12.6 Código del módulo decodificador de 1 a 2.

El multiplexor de 2 a 1 con palabra de 7 bits se muestra en la figura 12.7.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Multiplexor is
    Port ( Ent1 : in STD_LOGIC_VECTOR (6 downto 0);
           Ent2 : in STD_LOGIC_VECTOR (6 downto 0);
           Selector : in STD_LOGIC;
           Salida : out STD_LOGIC_VECTOR (6 downto 0));
end Multiplexor;
architecture Behavioral of Multiplexor is
begin
    process(Selector)
    begin
        case Selector is
            when '0' => Salida <= Ent1;
            when '1' => Salida <= Ent2;
        end case;
    end process;
end Behavioral;
```

Figura 12.7 Código del multiplexor 2 a 1 con palabra de 7 bits.

Hasta este momento, la ventana fuentes mostrará la información que se indica en la figura 12.8.

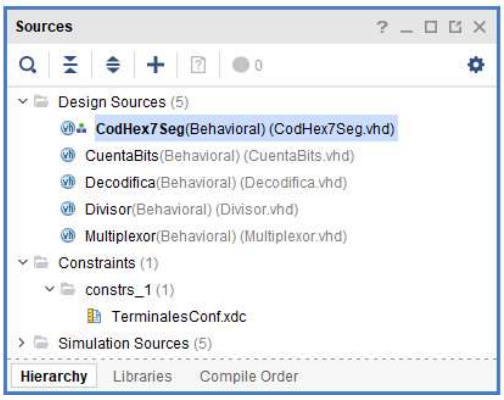


Figura 12.8 Avance del proyecto en ventana fuentes.

Se agrega otro archivo fuente del tipo VHDL y con nombre "practica_12", en el cual será el contenedor del proyecto y en él se enlazarán todos los módulos para terminar el proyecto. La de ventana de fuentes se presenta en la figura 12.9.

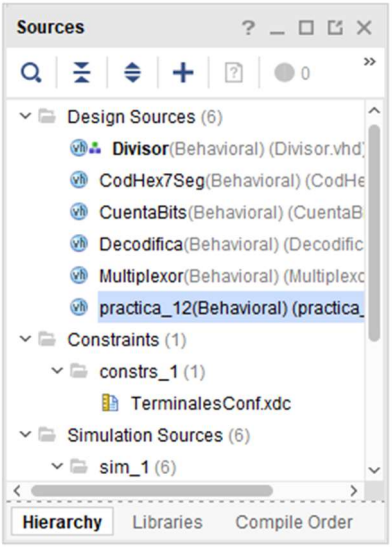


Figura 12.9 El archivo contenedor del proyecto "practica_12" agregado al proyecto.

Los puertos del módulo son cuatro interruptores que se representan por la variable **Sw**, la base de tiempo de la tarjeta Basys 3, representado por la variable **clk_in**, las 7 líneas que irán conectadas al módulo de display´s de 7 segmentos del Basys 3, dado por la variable **Display**, y una salida de cuatro bits para la selección del display en el que se exhibirá el dato del multiplexor.

La entidad descrita anteriormente se muestra en la figura 12.10.

```
entity practica_12 is
    Port ( Sw : in STD_LOGIC_VECTOR (3 downto 0);
           clk_in : in STD_LOGIC;
           Display : out STD_LOGIC_VECTOR (6 downto 0);
           Selector : out STD_LOGIC_VECTOR (3 downto 0));
end practica_12;
```

Figura 12.10 Entidad del módulo contenedor del proyecto.

La siguiente porción de código, dado en la figura 12.11, muestra la porción inicial de la arquitectura del módulo contenedor, en el cual se encuentra la declaración de los módulos creados anteriormente.

```
architecture Behavioral of practica_12 is
    component CodHex7Seg
    Port (
          Hexa : in STD_LOGIC_VECTOR (3 downto 0);
          seg7 : out STD_LOGIC_VECTOR (6 downto 0));
    end component;
    component CuentaBits
        Port ( Bits : in STD_LOGIC_VECTOR (3 downto 0);
               Cuenta : out STD_LOGIC_VECTOR (3 downto 0));
    end component;
    component Decodifica
        Port ( Bin : in STD_LOGIC;
               Linea : out STD_LOGIC_VECTOR (3 downto 0));
    end component;
    component Divisor
        Port (
            clk : in STD_LOGIC;
            clk_out : out STD_LOGIC);
    end component;
    component Multiplexor
        Port (
               Ent1 : in STD_LOGIC_VECTOR (6 downto 0);
               Ent2 : in STD_LOGIC_VECTOR (6 downto 0);
               Selector : in STD_LOGIC;
               Salida : out STD_LOGIC_VECTOR (6 downto 0));
    end component;
```

Figura 12.11 Parte inicial de la arquitectura del módulo contenedor.

En la figura 12.12 se muestra la declaración de las señales necesarias para la interconexión de los módulos, al igual que la interconexión de los mismos módulos.

```
    signal reloj: std_logic := '0';
    signal CantUnos: std_logic_vector (3 downto 0);
    signal Con7_1: std_logic_vector (6 downto 0);
    signal Con7_2: std_logic_vector (6 downto 0);
begin
Instancia_divisor: Divisor PORT MAP (clk=>clk_in,clk_out=>reloj);
Instancia_Decodif: Decodifica PORT MAP (Bin=>reloj,Linea=>Selector);
Instancia_ContBit: CuentaBits PORT MAP (Bits=>Sw,Cuenta=>CantUnos);
Instancia_CodH_71: CodHex7Seg PORT MAP (Hexa=>Sw,seg7=>Con7_1);
Instancia_CodH_72: CodHex7Seg PORT MAP (Hexa=>CantUnos,seg7=>Con7_2);
Instancia_Multipl: Multiplexor PORT MAP (Ent1=>Con7_1,Ent2=>Con7_2,
                                         Selector=>reloj,Salida=>Display);
end Behavioral;
```

Figura 12.12 Señales e interconexión de los módulos.

En la figura 12.13 tenemos la nueva jerarquía de los archivos en la ventana fuente, donde se observa que la jerarquía del archivo fuente "practica_12" se encuentra en el nivel superior, y en consecuencia, el resto de los archivos fuente quedan abajo de esa jerarquía.

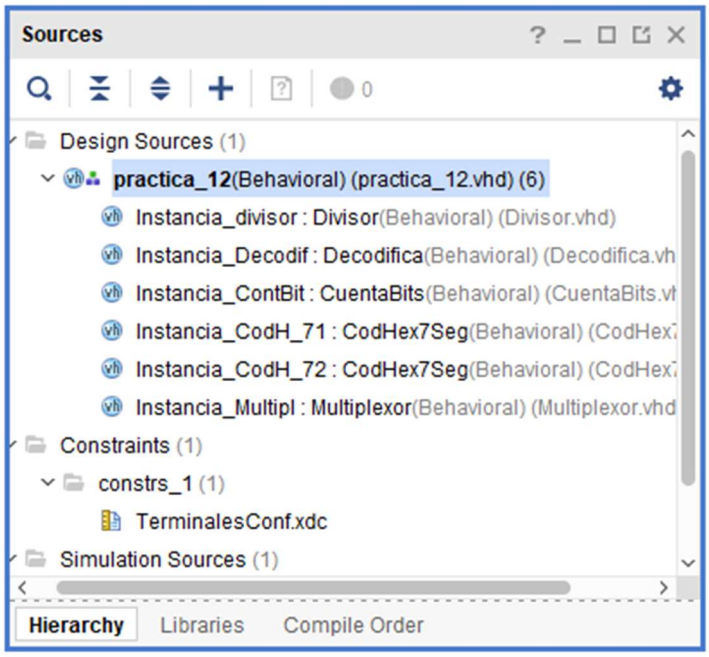


Figura 12.13 Nueva jerarquía de archivos fuente.

Finalmente, la figura 12.13 muestra el contenido del archivo de restricciones, correspondiente a la configuración de las terminales de software con las físicas.

```
## Interruptores
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {Sw[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {Sw[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {Sw[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {Sw[3]}]

## Display 7 Segmentos
set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports {Display[0]}]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports {Display[1]}]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports {Display[2]}]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports {Display[3]}]
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports {Display[4]}]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports {Display[5]}]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports {Display[6]}]

## Anodos de display
set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports {Selector[0]}]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports {Selector[1]}]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports {Selector[2]}]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports {Selector[3]}]


## Señal de reloj
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk_in]
```

Figura 12.14 Contenido del archivo de restricciones.

En la figura 12.15 se observa el programa del proyecto corriendo en la tarjeta Basys 3, en la que se ha introducido una **F** (1111), y se muestra valor introducido del lado derecho del módulo de display's, y en el lado izquierdo se muestra la cantidad de 1's que conforman al valor introducido.

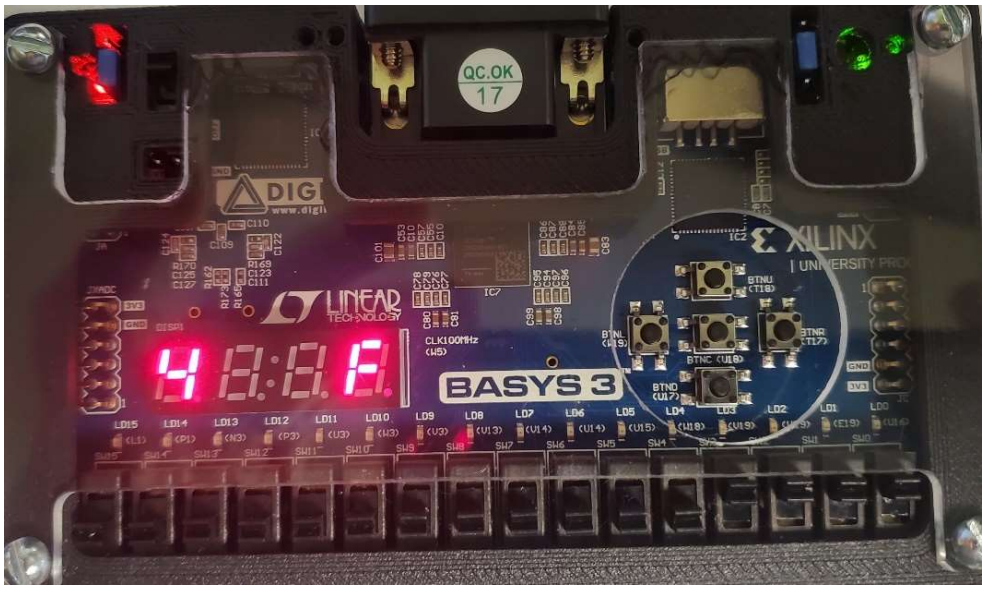


Figura 12.15 Proyecto ejecutado en la tarjeta Basys 3.

Desarrollo

Modifica el proyecto visto en el Análisis Previo para obtener un sumador de 2 valores de cuatro bits (valor hexadecimal), para lo cual hay que realizar los siguientes pasos:

1. Modificar la entidad del contenedor del proyecto para que tenga 2 puertos de entrada, cada uno de cuatro bits, que serán las entradas dadas por los interruptores (8, 4 por puerto) que representarán a los 2 valores de entrada. La figura 12.16 muestra una sugerencia del cambio.

```
entity practica_12 is
    Port (
            SwA : in STD_LOGIC_VECTOR (3 downto 0);
            SwB : in STD_LOGIC_VECTOR (3 downto 0);
            clk_in : in STD_LOGIC;
            Display : out STD_LOGIC_VECTOR (6 downto 0);
            Selector : out STD_LOGIC_VECTOR (3 downto 0));
end practica_12;
```

Figura 12.16 Entidad del contenedor del proyecto sugerida.

2. Modificar el divisor de frecuencia para que a su salida tenga un contador de 2 bits (cuenta de 0 a 3), la entidad para el divisor de frecuencia sugerida se muestra en la figura 12.17; no olvides modificar el código de la arquitectura de acuerdo con lo propuesto.
3.

```
entity Divisor is
    Port ( clk : in STD_LOGIC;
           clk_out : out STD_LOGIC_VECTOR (1 downto 0));
end Divisor;
```

Figura 12.17 Entidad del divisor de frecuencia sugerida.

4. Modificar el decodificador 2 a 1 a un decodificador de 2 a 4, en la figura 12.18 se muestra la sugerencia de la entidad; no olvides modificar el código de la arquitectura de acuerdo con lo propuesto.

```
entity Decodifica is
    Port ( Bin : in STD_LOGIC_VECTOR (1 downto 0);
           Linea : out STD_LOGIC_VECTOR (3 downto 0));
end Decodifica;
```

Figura 12.18 Entidad del decodificador sugerida.

5. Modificar el de multiplexor de 2 a 1 con palabra de 7 bits a un multiplexor de 4 a 1, como se sugiere en la figura 12.19; no olvides modificar el código de la arquitectura de acuerdo con lo propuesto.
6.

```
entity Multiplexor is
    Port ( Ent1 : in STD_LOGIC_VECTOR (6 downto 0);
           Ent2 : in STD_LOGIC_VECTOR (6 downto 0);
           Ent3 : in STD_LOGIC_VECTOR (6 downto 0);
           Ent4 : in STD_LOGIC_VECTOR (6 downto 0);
           Selector : in STD_LOGIC_VECTOR (1 downto 0);
           Salida : out STD_LOGIC_VECTOR (6 downto 0));
end Multiplexor;
```

Figura 12.19 Entidad del multiplexor sugerida.

7. Modifica el módulo **CuentaBits** para que realice la sumatoria de los 2 valores introducidos y regrese 2 vectores de cuatro bits cada uno, que representarán las unidades y las decenas de la suma. No olvides modificar el código de la arquitectura de acuerdo con lo propuesto. La figura 12.20 es una sugerencia de la entidad resultante de la modificación.
8.

```
entity CuentaBits is
    Port ( BitsA : in STD_LOGIC_VECTOR (3 downto 0);
           BitsB : in STD_LOGIC_VECTOR (3 downto 0);
           CuentaA : out STD_LOGIC_VECTOR (3 downto 0);
           CuentaB : out STD_LOGIC_VECTOR (3 downto 0));
end CuentaBits;
```

*Figura 12.20 Entidad del módulo **CuentaBits** sugerida.*

9. Modificar todas las declaraciones de los componentes modificados en la arquitectura del contenedor. La figura 12.21 muestra las modificaciones de acuerdo con las sugerencias dadas.

```
architecture Behavioral of practica_12 is
    component CodHex7Seg
    Port (
           Hexa : in STD_LOGIC_VECTOR (3 downto 0);
           seg7 : out STD_LOGIC_VECTOR (6 downto 0));
    end component;
    component CuentaBits
        Port ( BitsA : in STD_LOGIC_VECTOR (3 downto 0);
               BitsB : in STD_LOGIC_VECTOR (3 downto 0);
               CuentaA : out STD_LOGIC_VECTOR (3 downto 0);
               CuentaB : out STD_LOGIC_VECTOR (3 downto 0));
    end component;
    component Decodifica
        Port ( Bin : in STD_LOGIC_VECTOR (1 downto 0);
               Linea : out STD_LOGIC_VECTOR (3 downto 0));
    end component;
    component Divisor
        Port (
               clk : in STD_LOGIC;
               clk_out : out STD_LOGIC_VECTOR (1 downto 0));
    end component;
    component Multiplexor
        Port ( Ent1 : in STD_LOGIC_VECTOR (6 downto 0);
               Ent2 : in STD_LOGIC_VECTOR (6 downto 0);
               Ent3 : in STD_LOGIC_VECTOR (6 downto 0);
               Ent4 : in STD_LOGIC_VECTOR (6 downto 0);
               Selector : in STD_LOGIC_VECTOR (1 downto 0);
               Salida : out STD_LOGIC_VECTOR (6 downto 0));
    end component;
```

Figura 12.21 Modificación de las declaraciones de los módulos en la arquitectura del contenedor.

10. Al cambiar la cantidad y longitud de los vectores, las señales deberán de ser acordes. la figura 12.22 muestra esta modificación necesaria.

```
signal reloj: std_logic_vector (1 downto 0);
signal Suma1: std_logic_vector (3 downto 0);
signal Suma2: std_logic_vector (3 downto 0);
signal Con7_1: std_logic_vector (6 downto 0);
signal Con7_2: std_logic_vector (6 downto 0);
signal Con7_3: std_logic_vector (6 downto 0);
signal Con7_4: std_logic_vector (6 downto 0);
```

Figura 12.22 Actualización de las señales de la arquitectura del contenedor.

11. Siguiendo con los cambios sobre la arquitectura del proyecto, en la figura 12.23 se muestra la actualización del alambrado requerido por los cambios mencionados.

```
Instancia_divisor: Divisor PORT MAP (clk=>clk_in,clk_out=>reloj);
Instancia_Decodif: Decodifica PORT MAP (Bin=>reloj,Linea=>Selector);
Instancia_ContBit: CuentaBits PORT MAP (BitsA=>SwA,BitsB=>SwB,
                                        CuentaA=>Suma1,CuentaB=>Suma2);
Instancia_CodH_71: CodHex7Seg PORT MAP (Hexa=>SwA,seg7=>Con7_1);
Instancia_CodH_72: CodHex7Seg PORT MAP (Hexa=>SwB,seg7=>Con7_2);
Instancia_CodH_73: CodHex7Seg PORT MAP (Hexa=>Suma1,seg7=>Con7_3);
Instancia_CodH_74: CodHex7Seg PORT MAP (Hexa=>Suma2,seg7=>Con7_4);
Instancia_Multipl: Multiplexor PORT MAP (Ent1=>Con7_1,Ent2=>Con7_2,
                                         Ent3=>Con7_3,Ent4=>Con7_4,
                                         Selector=>reloj,Salida=>Display);
end Behavioral;
```

Figura 12.23 Cambios en el alambrado de los componentes instanciados.

12. Para programar la tarjeta Basys 3, implementar el archivo de restricciones como se muestra en la figura 12.24.

```
## Interruptores
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {SwA[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {SwA[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {SwA[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {SwA[3]}]

set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {SwB[0]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {SwB[1]}]
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {SwB[2]}]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports {SwB[3]}]

## Display 7 Segmentos
set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports {Display[0]}]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports {Display[1]}]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports {Display[2]}]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports {Display[3]}]
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports {Display[4]}]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports {Display[5]}]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports {Display[6]}]

## Anodos de display
set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports {Selector[0]}]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports {Selector[1]}]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports {Selector[2]}]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports {Selector[3]}]


## Señal de reloj
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk_in]
```

Figura 12.24 Archivo de restricciones actualizado.

Documente los cambios realizados en su proyecto, así como los resultados para incluirlo en el reporte de la práctica. Importante: **incluir el diagrama a bloques del proyecto actualizado**.

Referencias

1. Llanos, Juan; "Vivado Tutorial 1 Crear un proyecto", https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; "Vivado Tutorial 2 Simulación", https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.

Práctica # 13 Implementación de contador asíncrono utilizando FF como componentes

Objetivos de la práctica # 13

1. Diseñar e implementar los componentes en un proyecto VHDL.
2. Diseñar dispositivos basados en componentes en VHDL.
3. Conocer, comprender e implementar varias instancias de un mismo componente en VHDL.

Material y equipo

Enseguida se muestra en la tabla 13.1 los componentes y equipo a utilizar en la presente práctica, que comprenderá la programación en VHDL junto con el grabado de la tarjeta Basys 3 con el programa generado.

Tabla 13.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado
1	Tarjeta Basys 3
1	Cable USB para Basys 3

Introducción

Los módulos creados en VHDL pueden utilizarse en diferentes diseños, lo que permite la reutilización del código. Además, la misma descripción puede utilizarse para diferentes tecnologías sin tener que rediseñar todo el circuito. Esta descripción utiliza para la creación de la arquitectura de la entidad entidades descritas y compiladas previamente, de esta manera en VHDL podemos aprovechar diseños ya realizados, o realizar diseños sabiendo que se utilizarán en otros más complicados. Así se ahorra trabajo al diseñador-programador. Se declaran los componentes que se van a utilizar y después, mediante los nombres de los nodos, se realizan las conexiones entre los puertos. Las descripciones estructurales son útiles cuando se trata de diseños jerárquicos *botton-up*. Se pueden utilizar tantos componentes como necesite el diseño, estos componentes son entidades anteriormente declaradas o compiladas, en el ejemplo, subcircuito debe ser el nombre de una entidad que se ha declarado anteriormente en el código o que se ha compilado como parte de otro fichero VHDL. Se puede utilizar un componente tantas veces como sea

necesario (podemos asignar chip_i y chip_j al mismo componente, pero con distintas señales de entrada y salida). Podemos realizar un diseño estructural puro, mediante la unión de componentes que describen las distintas funcionalidades del circuito, en el que necesitaremos definir las señales intermedias entre las que se encontrará siempre las señales que interconectan las salidas de los módulos con las salidas de la *entity*.

Un *flip-flops* (FF), también conocido en español como dispositivo biestable, es un circuito de tipo multivibrador y secuencial que puede adquirir dos estados de manera indefinida, a menos que se perturbe de alguna manera dicho circuito. Es un dispositivo ampliamente usado en el almacenaje de datos e información en artículos digitales y electrónicos. Los biestables se utilizan para el almacenamiento de pequeñas cantidades de datos, llegando a poder almacenar un bit. Es por este motivo que se usan en cantidad para contener los datos a través de un código binario de todo tipo de dispositivos digitales y electrónicos, tales como contadores, máquinas de estado finitas, relojería, memorias de computadoras y calculadoras, por mencionar algunos.

En el mercado, los tipos más conocidos de biestables son los siguientes:

- FF R-S: El biestable R-S adquiere su nombre por sus entradas Reset y Set, para reiniciar y poner a uno, la información ingresada o almacenada en el dispositivo, respectivamente.

- FF T: En este tipo de FF el cambio de estado de produce mediante un pulso, el cual se constituye como un ciclo de cero a uno de manera completa. Este modelo de biestable puede utilizarse como un complemento de reloj para el modelo R-S.

- FF J-K: Este dispositivo es una combinación de los dos anteriores, pero se diferencia del RS en su comportamiento al activarse ambas entradas a la vez: Este biestable hace que su salida tenga el estado contrario al que poseía antes de abrirse las dos entradas simultáneamente.

En la figura 13.1 se muestran los cuatro diferentes tipos de FF's.

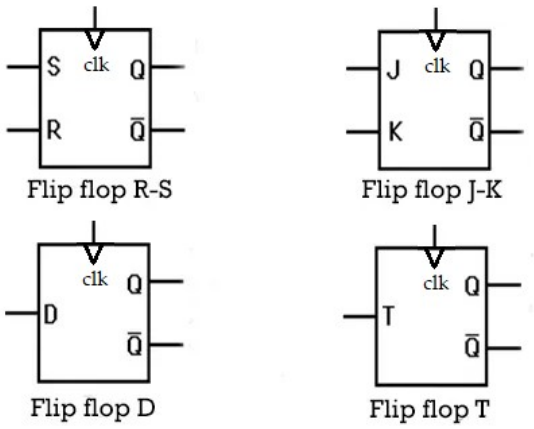


Figura 13.1 Los cuatro tipos de FF's.

Contadores digitales: Es todo circuito o dispositivo que genera una serie de combinaciones a sus salidas sincronizadas por una señal de reloj externa.

Clasificaciones, según el comportamiento con la señal de reloj: Contadores asíncronos y síncronos. En los sistemas asíncronos los FF no están conectados al mismo reloj, por lo que no cambian simultáneamente. La señal de reloj sólo ataca al FF que representa al bit menos significativo. Los otros FF se conectan en cascada sirviendo su salida de reloj para el siguiente, hasta llegar al bit más significativo. Si se utilizan FF T activos en el flanco de bajada, la estructura de un contador ascendente binario puede ser como el mostrado a continuación en la figura 13.2.

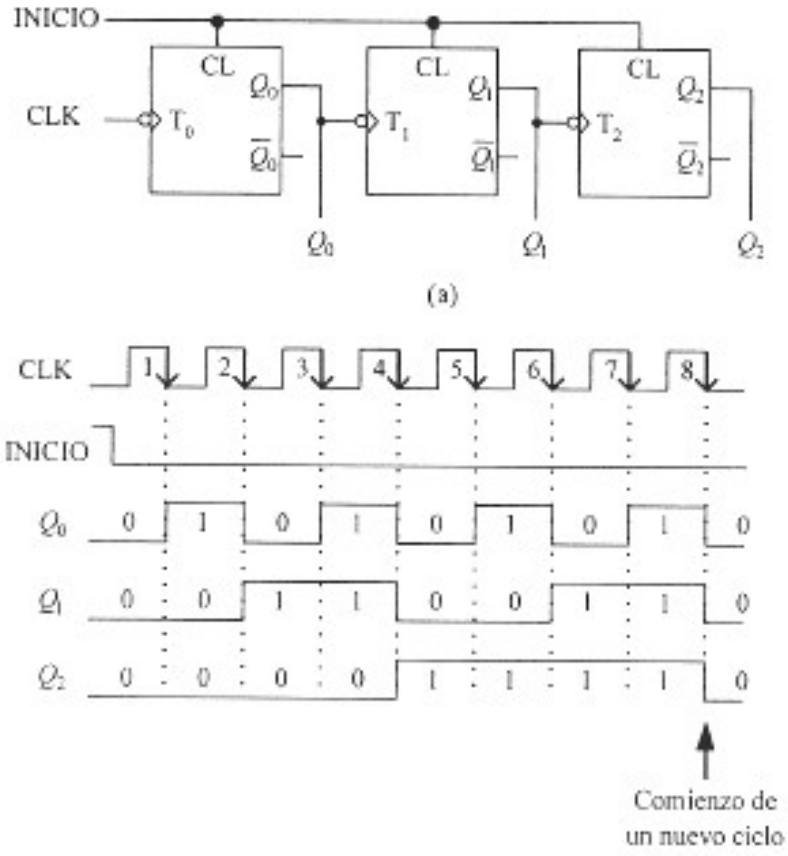


Figura 13.2 Configuración y cuenta de un contador asíncrono descendente módulo 8.

Observamos que la diferencia entre este contador asíncrono binario ascendente y el contador asíncrono binario descendente estriba únicamente en que los FF son activos en los flancos de subida y que este simple cambio da lugar a un cambio de comportamiento del contador.

Análisis previo

En la presente práctica, habrá que generar un proyecto en Vivado, al cual se le debe de agregar archivos fuente con el fin de generar los módulos para el funcionamiento de un FF JK, de un divisor de frecuencia, un contenedor del FF JK con la frecuencia proporcionada por el divisor de frecuencia (para una visualización amigable al usuario) y finalmente, un cuarto archivo fuente, en el que se desarrollará el contador asíncrono módulo 8 utilizando tres instancias del FF realizado. Revisar la práctica # 5 para repasar los conocimientos para la creación de un proyecto en Vivado (enfocarse en la creación del proyecto, cómo agregar archivos fuente y archivos de simulación, al igual que el proceso de simulación) y la práctica # 12 para repasar el manejo de componentes y la instanciación de los componentes en Vivado.

Desarrollo

La práctica se desarrollará en 3 etapas las cuales se Irán indicando enseguida.

Primera etapa

1. Crea un nuevo proyecto en Vivado y llámale “practica13”; agrégale un archivo fuente, en el cual habrá de programarse un FF que sea acorde a la figura 13.3, que incluye las entradas J, K, Clk (para la señal de reloj) y borrar (pone en cero al FF de forma asíncrona), y la salida Q. Programe su comportamiento utilizando sentencias if-then-elsif, enseguida se sugiere inicio del proceso para el FF:

```
process(clk, borrar)
begin
  if(borrar = '1') then
```

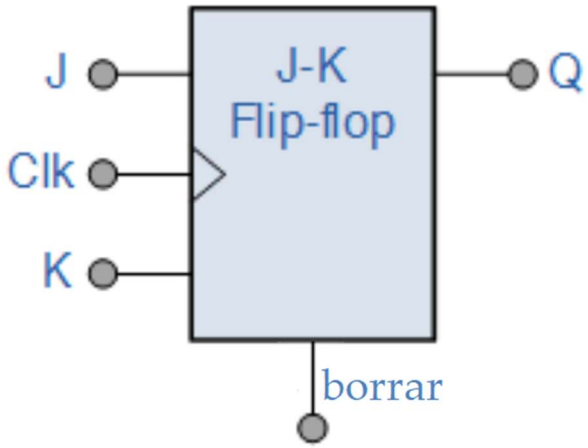


Figura 13.3 Diagrama del FF JK a programar en VHDL.

2. Agregue un archivo de simulación en el cual agregará el componente del recién hecho FF y lo instanciará. Agregue las señales necesarias para hacer la conexión de la instancia del FF con el proceso de simulación. En la figura 13.4 se muestra el código correspondiente a las señales utilizadas en el proceso de "alambrado" de la instancia del FF y la instancia del FF y en la figura 13.5 se observan los procesos para la simulación.

```
    signal j: std_logic := '0';
    signal k: std_logic := '0';
    signal borrar: std_logic := '0';
    signal clk: std_logic := '0';
    signal Q: std_logic := '0';

    constant reloj_periodo : time := 20 ns;
begin

instanciaFF:    flip_flop port map(
                j=>j,k=>k,borrar=>borrar,clk=>clk, Q=>Q);
```

Figura 13.4 Código de las señales utilizadas para el alambrado de la instancia del FF y la instancia del FF.

```
process
begin
    clk <= '0';
    wait for reloj_periodo/2;
    clk <= '1';
    wait for reloj_periodo/2;
end process;
process
begin
    borrar <= '0';
    j <= '0';
    k <= '0';
    wait for 60 ns;
    borrar <= '0';
    j <= '1';
    k <= '0';
    wait for 60 ns;
    borrar <= '1';
    j <= '1';
    k <= '0';
    wait for 60 ns;
```

Figura 13.5 Código para la simulación.

```
        borrar <= '0';
        j <= '1';
        k <= '0';
        wait for 60 ns;
        j <= '0';
        k <= '1';
        wait for 60 ns;
        j <= '1';
        k <= '1';
        wait for 60 ns;
        wait;
    end process;
end Behavioral;
```

Figura 13.6 Código para la simulación (Cont.)

3. Agregue un archivo de restricciones para la generación del archivo Bitstream y cárguelo a la tarjeta Basys 3.

Mostrar simulación e implementación física al profesor.

Segunda etapa

1. Agrégale un **segundo** archivo fuente al proyecto para programar un divisor de frecuencia; el código recomendado para el divisor de frecuencia se muestra en la figura 13.7, por tener mejor desempeño que el visto en la práctica # 10.

```
architecture Behavioral of divisor is
    signal count: std_logic_vector(25 downto 0):="00000000000000000000000000";
    signal tmp : std_logic := '0';
begin
    process(clk, count, tmp)
    begin
        if (clk'event and clk='1') then
            count <=count+1;

        end if;
        if (count(25) = '1' and count(23) = '1' and count(22) = '1' and
            count(21) = '1' and count(20) = '1' and count(19) = '1' and
            count(17) = '1') then
            tmp <= NOT tmp;
            count <= "00000000000000000000000000";
        end if;
    end process;
    clk_out <= tmp;
end Behavioral;
```

Figura 13.7 Código recomendado para el divisor de frecuencia.

No olvide agregar las bibliotecas

```
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;
```

para su correcto funcionamiento.

2. Agregue un **tercer** archivo fuente, en el que agregar a un componente para el FF y otro componente para el divisor de frecuencia; agregue las señales necesarias para el "alambrado" de las instancias de los 2 componentes mencionados.

3. Genera el archivo Bitstream, utilizando el archivo de restricciones de la etapa anterior y cárguelo a la tarjeta Basys 3.

Mostrar implementación física al profesor.

Tercera etapa

1. Al **tercer** archivo fuente agregado en la etapa anterior (el archivo donde instanciaste el FF y el divisor de frecuencia), desde la ventana archivos fuente, darle clic con el botón derecho del ratón y seleccionar deshabilitar como se muestra en la figura 13.8, de esta manera podrás trabajar con un nuevo archivo fuente para implementar el contador asíncrono sin borrar el archivo anterior.

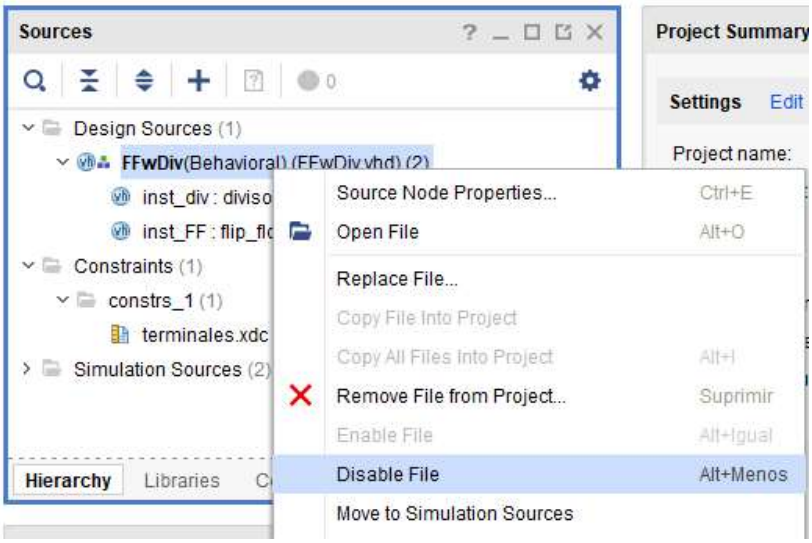


Figura 13.8 Inhabilitación del archivo del conjunto FF y divisor de frecuencia.

2. Agrega un **cuarto** archivo fuente, en el que agregarás un componente para el divisor de frecuencia y un componente para el FF.

3. Agrega las señales necesarias para realizar el "alambrado" de la instancia del divisor de frecuencia y de 3 instancias para el componente del FF. El alambrado de las instancias de los FF deberás de ser de acuerdo con un contador asíncrono de módulo 8.

4. Genera el archivo Bitstream, utilizando el archivo de restricciones de la primera etapa y cárguelo a la tarjeta Basys 3.

Mostrar implementación física al profesor.

Referencias

1. Llanos, Juan; "Vivado Tutorial 1 Crear un proyecto", https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; "Vivado Tutorial 2 Simulación", https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.

Práctica # 14 Implementación de contador síncrono utilizando FF como componentes

Objetivos de la práctica # 14

1. Diseñar e implementar los componentes en un proyecto VHDL.
2. Diseñar dispositivos basados en componentes en VHDL.
3. Conocer, comprender e implementar varias instancias de un mismo componente en VHDL.

Material y equipo

Enseguida se muestra en la tabla 14.1 los componentes y equipo a utilizar en la presente práctica, que comprenderá la programación en VHDL junto con el grabado de la tarjeta Basys 3 con el programa generado.

Tabla 14.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado
1	Tarjeta Basys 3
1	Cable USB para Basys 3

Introducción

Un *flip-flops* (FF), también conocido en español como dispositivo biestable, es un circuito de tipo multivibrador y secuencial que puede adquirir dos estados de manera indefinida, a menos que se perturbe de alguna manera dicho circuito. Es un dispositivo ampliamente usado en el almacenaje de datos e información en artículos digitales y electrónicos. Los biestables se utilizan para el almacenamiento de pequeñas cantidades de datos, llegando a poder almacenar un bit. Es por este motivo que se usan en cantidad para contener los datos a través de un código binario de todo tipo de dispositivos digitales y electrónicos, tales como contadores, máquinas de estado finitas, relojería, memorias de computadoras y calculadoras, por mencionar algunos.

En el mercado, los tipos más conocidos de biestables son los siguientes:

- FF R-S: El biestable R-S adquiere su nombre por sus entradas Reset y Set, para reiniciar y poner a uno, la información ingresada o almacenada en el dispositivo, respectivamente.

- FF T: En este tipo de FF el cambio de estado de produce mediante un pulso, el cual se constituye como un ciclo de cero a uno de manera completa. Este modelo de biestable puede utilizarse como un complemento de reloj para el modelo R-S.

- FF J-K: Este dispositivo es una combinación de los dos anteriores, pero se diferencia del RS en su comportamiento al activarse ambas entradas a la vez: Este biestable hace que su salida tenga el estado contrario al que poseía antes de abrirse las dos entradas simultáneamente.

En la figura 14.1 se muestran los cuatro diferentes tipos de FF's.

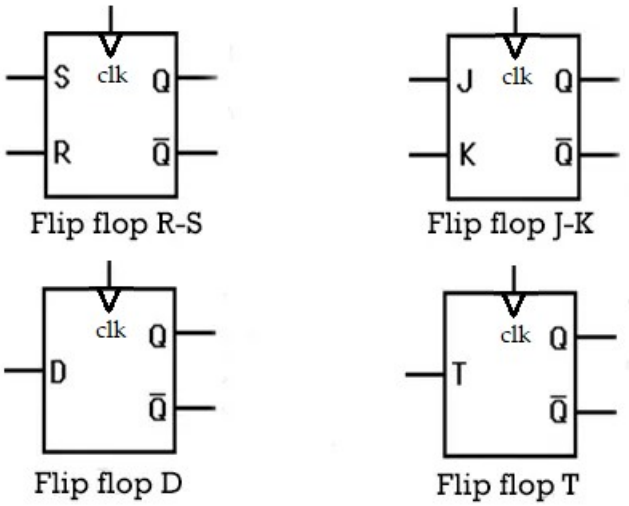


Figura 14.1 Los cuatro tipos de FF's.

Contadores digitales: Es todo circuito o dispositivo que genera una serie de combinaciones a sus salidas sincronizadas por una señal de reloj externa.

Clasificaciones, según el comportamiento con la señal de reloj: Contadores asíncronos y síncronos. En los sistemas síncronos todos los FF están conectados al mismo reloj, por lo que cambian simultáneamente. La estructura de un contador síncrono ascendente binario puede ser como el mostrado a continuación en la figura 14.2.

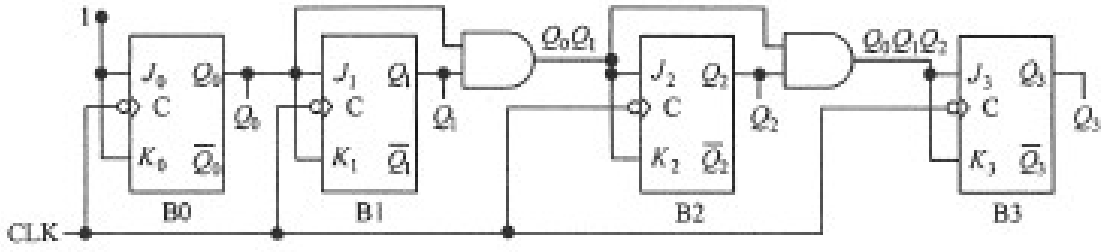


Figura 14.2 Configuración de contador síncrono módulo 16.

El diagrama de tiempo del contador síncrono módulo 16 mostrado en la figura 14.2 se muestra en la figura 14.3, nótese como se indica los cambios dado por la combinación AND de los estados, con lo cual se activa el siguiente FF en su entrada T.

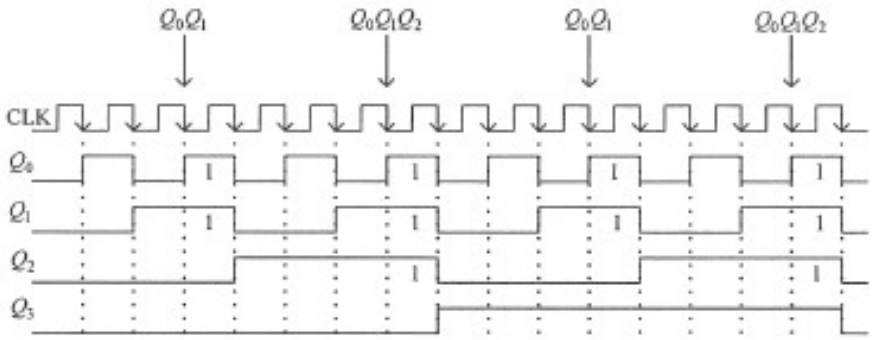


Figura 14.3 Diagrama de tiempo del contador síncrono módulo 16.

Análisis previo

Para la realización de la práctica # 14, habrá que realizarse por completo la práctica # 13 y realizar las modificaciones sugeridas en la sección de Desarrollo de la presente práctica.

Desarrollo

La práctica que se desarrollará consiste en modificar el cuarto archivo fuente agregado al proyecto de Vivado correspondiente a la práctica # 13, de tal forma que la configuración de los FF del contador asíncrono módulo 16 se modifique a una configuración de FF para un contador síncrono módulo 16, como se muestra en la figura 14.2.

Mostrar implementación física al profesor.

Referencias

1. Llanos, Juan; "Vivado Tutorial 1 Crear un proyecto", https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; "Vivado Tutorial 2 Simulación", https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; "Manual de Prácticas de Circuitos Digitales", Universidad Autónoma de Ciudad Juárez, diciembre 2020.

Práctica # 15 Contador síncrono módulo 10.

Objetivos de la práctica # 15

1. Diseñar e implementar los componentes en un proyecto VHDL.
2. Diseñar dispositivos basados en componentes en VHDL.
3. Conocer, comprender e implementar varias instancias de un mismo componente en VHDL.

Material y equipo

Enseguida se muestra en la tabla 15.1 los componentes y equipo a utilizar en la presente práctica, que comprenderá la programación en VHDL junto con el grabado de la tarjeta Basys 3 con el programa generado.

Tabla 15.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado
1	Tarjeta Basys 3
1	Cable USB para Basys 3

Introducción

Se denomina módulo a la cantidad de estados diferentes en los que se va a encontrar un contador, es decir, sí tenemos un contador que inicia en cero y su cuenta termina en 7 para reiniciar una vez más en cero, entonces tenemos 8 estados para dicho contador, por lo que su módulo será 8. El desarrollo de contadores de diferentes módulos es útil porque permite acoplar una cuenta a cierta necesidad particular, por ejemplo, la cuenta de los minutos y de los segundos es sexagesimal, por lo que se podrá implementar convenientemente con un contador de unidades módulo 10 y un contador de decenas módulo 6.

En la figura 15.1 se muestra una configuración de un contador síncrono módulo 8, pues su cuenta iniciará en cero y terminará en 7 y volviendo a reiniciar en cero. Sí de esta configuración se requiere un módulo menor, habrá que agregar circuitería para cuando se llegue a la cuenta final, la circuitería mencionada reinicie la cuenta a cero o incluso a un valor diferente de cero.

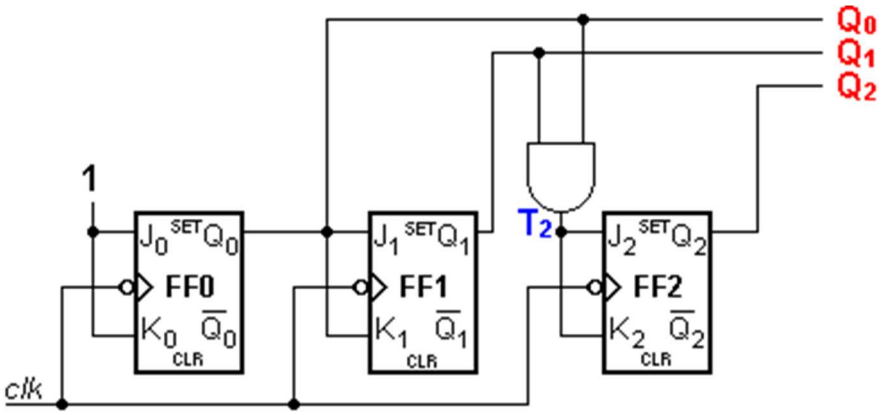


Figura 15.1 Contador síncrono módulo 8

Análisis previo

Para el diseño del contador módulo 10, al menos se requieren 4 *flip-flops*, como el contador realizado en la práctica # 14, y agregarle la lógica de reinicio al iniciar la cuenta 10 (o al terminar la cuenta 10).

Desarrollo

Partiendo del resultado de la práctica # 14, en donde se tiene un contador síncrono módulo 16, seguir los siguientes pasos.

1. Realizar la tabla de verdad, donde las entradas estarán dadas por los estados de los cuatro *flip-flops* y la salida será la conexión a la entrada BORRAR de los clips. Hay que considerar que la cuenta será de 0 a 9 y el resto de las cuentas del tipo Don'tcare. Recuerda que la señal de entrada BORRAR es positiva.

2. Minimizar la función por medio de mapas de Karnaugh.

3. Optimiza la función de ser posible, utilizando asociaciones o uso de compuertas XOR, XNOR, etc.

4. Agregar el código necesario para la función de reinicio al contador realizado en la práctica # 14.

Mostrar implementación física al profesor.

Referencias

1. Llanos, Juan; “Vivado Tutorial 1 Crear un proyecto”, https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; “Vivado Tutorial 2 Simulación”, https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; “Manual de Prácticas de Circuitos Digitales”, Universidad Autónoma de Ciudad Juárez, diciembre 2020.

Práctica # 16 Máquina de Estado Finito

Objetivos de la práctica # 16

1. Conocer, abstraer e implementar una máquina de estado finito en VHDL.
2. Desarrollar las diferentes etapas de una máquina de estado finito en VHDL.
3. Realizar e implementar una máquina de estado finito en la tarjeta Basys 3.

Material y equipo

Enseguida se muestra en la tabla 16.1 los componentes y equipo a utilizar en la presente práctica, que comprenderá la programación en VHDL junto con el grabado de la tarjeta Basys 3 con el programa generado.

Tabla 16.1 Material y equipo.

Cantidad	Material/equipo
1	Computadora con Vivado
1	Tarjeta Basys 3
1	Cable USB para Basys 3

Introducción

Las máquinas de estados finitos se utilizan ampliamente en aplicaciones en ciencias de la computación y redes de datos. Por ejemplo, las máquinas de estados finitos son la base para los programas de corrección ortográfica, la comprobación de la gramática, la indexación o la búsqueda de grandes volúmenes de texto, reconocimiento de voz, la transformación de texto utilizando lenguajes de marcado como XML y HTML, y los protocolos de red que especifican cómo las computadoras se comunican.

Una Máquina de Estado Finito (Finite State Machine), llamada también Autómata Finito es una abstracción computacional que describe el comportamiento de un sistema reactivo mediante un número determinado de Estados y un número determinado de Transiciones entre dicho Estados. Las Transiciones de un estado a otro se generan en respuesta a eventos de entrada externos e internos; a su vez estas transiciones y/o subsecuentes estados pueden generar otros eventos de salida. Esta dependencia de las acciones (respuesta) del sistema a

los eventos de entrada hace que las Máquinas de Estado Finito (MEF) sean una herramienta adecuada para el diseño de Sistemas Reactivos y la Programación Conducida por Eventos (Event Driven Programming), cual es el caso de la mayoría de los sistemas embebidos basados en microcontroladores o microprocesadores. Las MEF se describen gráficamente mediante los llamados Diagramas de Estado Finito (DEF), llamados también Diagramas de Transición de Estados. Un Sistema Reactivo es aquel que interactúa constantemente con su medio ambiente, tiene la característica de ser conducido por eventos (event driven), la respuesta de tiempo es crítica y una vez que el sistema se activa permanece en ese estado de manera indefinida. En estos sistemas los eventos llegan en tiempos impredecibles y el sistema debe tener la capacidad de responder de manera inmediata, en el orden de los milisegundos o microsegundos, sobre todo en sistemas donde la seguridad es crítica (ejemplo: un piloto automático en un avión o una máquina para soporte de vida en un hospital). La gran mayoría de los sistemas embebidos (en base a microcontroladores o microprocesadores) corresponden a esta categoría, debido a que estos sistemas están típicamente conectados a varios tipos de sensores y transductores de entrada encargados de captar los estímulos del medio ambiente (temperatura, presión, luz, magnetismo, fuerza / peso, etc.), procesar dicha información y generar una respuesta del sistema hacia el medio ambiente a través de transductores de salida y actuadores. A diferencia de los Sistemas Reactivos un Sistema Transformacional es aquel que recibe cierta información de entrada, realiza una cierta cantidad de cómputo, produce cierta información de salida y luego termina. No muchos sistemas embebidos caen en esta categoría; ejemplos más típicos son las aplicaciones para PC, como, por ejemplo: Un procesador de texto.

Un Diagrama de Estado Finito es un gráfico que representa los diferentes estados de una MEF y todas las transiciones posibles entre los estados. En la imagen 16.1 se muestra un diagrama de una máquina de estado finito de 2 estados.

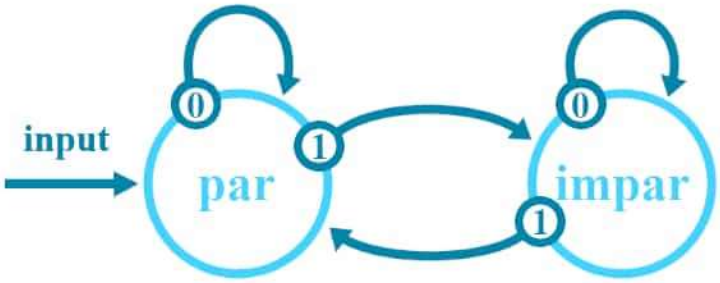


Figura 0.1 Diagrama de estados de una máquina de estado finito.

Las máquinas de estado finito se caracterizan porque adoptan un estado y solo uno en cada momento t_i. Dichos estados son configuraciones que la máquina puede adoptar y que

determinan el comportamiento de la máquina en el paso siguiente t_{i+1}. Un semáforo por ejemplo tiene tres estados: rojo, verde y ámbar. Si el semáforo está en rojo en el estado t, significa que estará verde en el estado t_{i+1}. El número de estados disponibles para el autómata será finito, es por eso por lo que se conocen como autómatas de estado finito.

Análisis previo

Se revisarán previamente los pasos para generar Una máquina de estado finito o autómata que dará pie a un juego de adivinar un valor generado aleatoriamente con anterioridad. el enunciado se menciona enseguida:

Realizar una máquina de estado finito (MEF) que tendrá el fin de generar un número aleatorio de un dígito hexadecimal, y dicho valor deberá de ser adivinado por el usuario en 5 oportunidades, utilizando interruptores de tipo botón en la forma indicada. El valor hexadecimal que el usuario introducirá para adivinar el valor generado aleatoriamente por la MEF, será por medio de cuatro interruptores deslizables. El juego tendrá 3 entradas de control:

*a) La primera entrada denominada 'Reiniciar', al ser presionada en cualquier estado, nos llevará al estado inicial, en el cual, se mostrarán las letras "*Inic*", indicativo de que el juego está listo para inicializarse. Durante este primer estado, un contador de cuatro bits se estará incrementando y terminará su cuenta en el momento que pase al siguiente estado, generando de esta forma un número aleatorio hexadecimal.*

b) La segunda entrada denominada 'Juega' nos llevará al segundo estado (estando en el primer estado o en el tercer estado), el que dará paso a introducir un nuevo valor, por medio de los interruptores deslizables, el cual el usuario dará con el fin de adivinar el valor generado en forma aleatoria. Esta entrada se podrá utilizar para salir del estado inicial y comenzar el juego, así como para continuar el juego después de recibir las indicaciones de modificar el valor introducido para acertar al valor aleatorio; un último uso es, al finalizar el juego, dará el puntaje conseguido en el juego.

*c) La tercera entrada denominada "Verifica" nos llevará al tercer estado, en el que la MEF tomará el valor introducido y lo comparará con el valor generado aleatoriamente, si el valor introducido es menor que el valor generado, aparecerá la leyenda "*SubE*" en los display de la tarjeta; si el valor introducido es mayor que el valor generado, entonces la leyenda "*BAJA*" aparecerá en los displays; si el valor introducido es igual al valor generado, en los display aparecerá la leyenda "*gAnA*", mientras que si se acabaron las 5 oportunidades para atinarle al valor generado, la leyenda que aparecerá en los displays es "*UPSS*". Cuando en el tercer estado se muestran los mensajes "*SubE*" o "*BAJA*", El siguiente estado será el segundo, donde se introduce un nuevo valor, pero cuando los mensajes "*gAnA*" y "*UPSS*" aparecen, el siguiente estado será el cuarto, mostrando el puntaje obtenido y quedándose pausado en él, hasta que se presione la entrada 'Reiniciar'.*

En la figura 16.2 se muestra el diagrama de estados de la MEF descrita anteriormente con comentarios y nombres dados en los estados por letras del alfabeto.

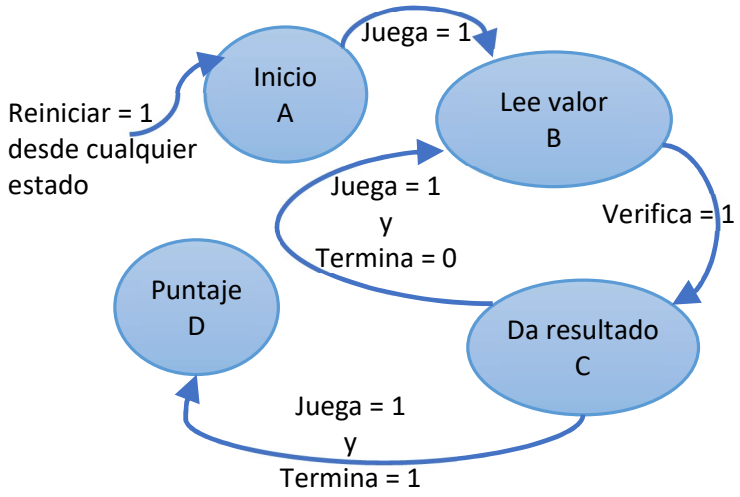


Figura 0.2 Diagrama de estados de la MEF propuesta. Obsérvese que solo se toman acciones cuando las entradas tienen valor de '1'. 'Termina' es una memoria interna.

En la figura 16.2 se menciona que 'Termina' es una memoria interna, con valor inicial de '0' y es puesta en '1' cuando el usuario adivina el valor generado o cuando se terminan las 5 oportunidades de adivinar el valor general.

Se realiza un proyecto en el cual se van a incluir los componentes utilizados anteriormente para divisor de frecuencia, codificador a 7 segmentos y el componente multiplexor selector, los cuales se utilizaron previamente en la práctica del contador asíncrono, práctica del contador síncrono y práctica del contador síncrono módulo 10. Las bibliotecas utilizadas en este proyecto junto con una sugerencia para la entidad se muestran en la figura 16.3.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Juego is
    Port (
              CLK : in  STD_LOGIC;       --Entrada del reloj
              Reiniciar : in  STD_LOGIC;         --Se~al de reiniciar
              Juega : in  STD_LOGIC;         --Se~al de introducir valor
              Verifica : in  STD_LOGIC;      --Se~al de verificar intento
              Dat_Ent : in  STD_LOGIC_VECTOR(3 downto 0);        --Datos de los interruptores
              control: out  STD_LOGIC_VECTOR (3 downto 0);       --Control de los displays
              Sal7Seg: out  STD_LOGIC_VECTOR (6 downto 0));      --Segmentos displays
end Juego;
```

Figura 0.3 Entidad sugerida para el proyecto MEF.

En la sección de la arquitectura del proyecto, agrega los componentes mencionados en el párrafo anterior y en seguida, agrega las definiciones de señales con las que se configurarán las instancias de los componentes, como se indican en la figura 16.4.

```
TYPE MisEstados is (A, B, C, D);
signal EdoActual, EstadoSig : MisEstados;
signal CLKout: STD_LOGIC;
signal Termina: STD_LOGIC := '0';
signal Valor_Secreto_Orig, Valor_Secreto:STD_LOGIC_VECTOR (3 downto 0);
signal Puntaje_B, Puntaje_C, Dat_Ent_Copia: STD_LOGIC_VECTOR(3 downto 0);
signal Dat_Ent7, Puntaje_C7:STD_LOGIC_VECTOR (6 downto 0);
signal Disp0, Disp1, Disp2, Disp3:STD_LOGIC_VECTOR (6 downto 0);
```

Figura 0.4 Señales sugeridas para ser utilizadas en el proyecto para la configuración de las instancias de los componentes.

Observa en la figura 16.4 que se está utilizando el identificador TYPE para generar una enumeración denominada "MisEstados" y está definida para los valores A, B, C y D, que vienen siendo los cuatro estados necesarios para la MEF del proyecto; las variables del tipo "MisEstados" son 2, "EdoActual" y "EstadoSig", que serán precisamente en donde se guardará el estado en el que se está actualmente y el estado al que habrá de cambiarse de acuerdo con los cálculos realizados.

La instanciación de los componentes necesarios para el proyecto se muestra en la figura 16.5, en los que se incluye una instancia para el divisor, una instancia para el multiplexor selector y 2 instancias para el codificador de 7 segmentos.

```
Divisor_Freq: divisor port map (clk=>CLK, clk_out=>CLKout);
SieteSeg1: cod7seg port map(Dato=>Dat_Ent,Sal7Seg=>Dat_Ent7);
SieteSeg2: cod7seg port map(Dato=>Puntaje_C,Sal7Seg=>Puntaje_C7);
FourDisplay: MultiplexorSelector port map(D1=>Disp0,D2=>Disp1,D3=>Disp2,D4=>Disp3,
                    Clock=>CLKout, control=>control, Display=>Sal7Seg);
```

Figura 0.5 Instanciación de los componentes utilizados en el proyecto.

En la figura 16.6 se muestra la lógica de estado para la MEF, y la correspondiente lógica de memoria en la figura 16.7. Es importante recalcar que la única variable de memoria que se utiliza en la lógica de estado es la variable denominada 'Termina', la cual se puede observar que en la lógica de memoria es inicializada a '0' cuando la MEF se encuentra en el estado A, y es cambiado a '1' en el estado C cuando el dato introducido por el usuario acierta al valor generado de forma aleatoria o cuando se terminan las oportunidades de introducir valores. otro detalle importante es que se agregó una variable para guardar los valores de entrada y poderlos utilizar en el estado C, pues sí se maneja directamente el valor de la entrada para realizar las comparaciones, si el usuario realiza cambios en los interruptores deslizables, estos cambios se ven reflejados automáticamente en el estado C, permitiendo acertar al valor generado de forma tramposa.

```
Logica_Estado: Process (EdoActual, Juega, Verifica, Termina)
begin
    case (EdoActual) is
        when A => if (Juega ='1') then EstadoSig <= B;
            else EstadoSig <= A; end if;
        when B => if (Verifica ='1') then EstadoSig <= C;
            else EstadoSig <= B; end if;
        when C => if (Juega ='1') then
                    if(Termina = '0') then EstadoSig <= B;
                    else EstadoSig <= D; end if;
            else EstadoSig <= C;
            end if;
        when D => EstadoSig <= D;
    end case;
end process;
```

Figura 0.6 Lógica de estados, obsérvese el uso de una variable de memoria (Termina) que ayuda a determinar el fin del juego.

```
Logica_Memoria: Process (CLKout, EstadoSig, Termina)
begin
    if  (CLKout = '1' and CLKout'event) then
        Valor_Secreto_Orig <= Valor_Secreto_Orig + 1;
        if EdoActual = A then -- Codigo para guardar datos a memoria.
            Valor_Secreto <= Valor_Secreto_Orig;--Guarda valor aleatorio
            Termina <= '0';--Inicializa variable a cero (Esta iniciando)
            Puntaje_C <= "0110";-- 5 (+ 1 por secuencia) oportunidades
        elsif EdoActual = B then
            Dat_Ent_Copia <= Dat_Ent;
            Puntaje_B <= Puntaje_C - 1;-- Decrementa puntaje
        elsif EdoActual = C then
            Puntaje_C <= Puntaje_B;-- Resguarda puntaje
            if Dat_Ent_Copia = Valor_Secreto or Puntaje_C = 1 then
                Termina <= '1';
            end if;
        elsif EdoActual = D then
           if Puntaje_C = 1 and Dat_Ent_Copia /= Valor_Secreto then
                Puntaje_C <= "0000";
            end if;
        end if;
        if Reiniciar = '1' then
            EdoActual <= A; --Reiniciar en A
        else
            EdoActual<=EstadoSig;
        end if;
    end if;
end process;
```

Figura 0.7 Logica de memoria, donde se toman las desiciones de que se guarda y como.

Debido a que el proyecto refiere a un juego, gran parte del esfuerzo del desarrollo esta dado en las salidas, ya que se trata de hacer atractivo visualmente. Para facilitar la programación de la lógica de salida, se realiza una tabla de los valores que se mostrará en cada display´s dependiendo del estado en que se encuentre la MEF. Fracción del desarrollo de la tabla para la programación de la salida es mostrada en la tabla 16.2.

Tabla 16.2 Valores de las salidas de acuerdo con el estado (incompleta).

	Valores de los display's			
Estado	Disp0	Disp1	Disp2	Disp3
A	"1111001" Letra (I)	"0101011" Letra (n)	"1111011" Letra (i)	"0100111" Letra (c)
B	"1111111" apagado	"1111111" apagado	"1111111" apagado	Dat_Ent7 Valor de los interruptores
C	**Completar**	**Completar**	**Completar**	**Completar**
D	**Completar**	**Completar**	**Completar**	**Completar**

El código correspondiente a la asignación de los valores para los display se muestra en la figura 16.8

```
Disp3 <= "0100111" when EdoActual = A else -- Letra c
          Dat_Ent7 when EdoActual = B else -- Valor de entrada
         "0010010" when EdoActual = C and Dat_Ent_Copia /= Valor_Secreto and Termina = '1' else -- Letra S
         "0000110" when EdoActual = C and Dat_Ent_Copia < Valor_Secreto else -- Letra E
         "0001000" when EdoActual = C and Dat_Ent_Copia > Valor_Secreto else -- Letra A
         "0001000" when EdoActual = C and Dat_Ent_Copia = Valor_Secreto else -- Letra A
         Puntaje_C7; -- Puntaje
Disp2 <= "1111011" when EdoActual = A else -- Letra i
         "1111111" when EdoActual = B else -- Apagado
         "0010010" when EdoActual = C and Dat_Ent_Copia /= Valor_Secreto and Termina = '1' else -- Letra S
         "0000011" when EdoActual = C and Dat_Ent_Copia < Valor_Secreto else -- Letra b
         "1110001" when EdoActual = C and Dat_Ent_Copia > Valor_Secreto else -- Letra j
         "0101011" when EdoActual = C and Dat_Ent_Copia = Valor_Secreto else -- Letra n
         "0010010"; -- Letra S
Disp1 <= "0101011" when EdoActual = A else -- Letra n
         "1111111" when EdoActual = B else -- Apagado
         "0001100" when EdoActual = C and Dat_Ent_Copia /= Valor_Secreto and Termina = '1' else -- Letra P
         "1100011" when EdoActual = C and Dat_Ent_Copia < Valor_Secreto else -- Letra u
         "0001000" when EdoActual = C and Dat_Ent_Copia > Valor_Secreto else -- Letra A
         "0001000" when EdoActual = C and Dat_Ent_Copia = Valor_Secreto else -- Letra A
         "0000111"; -- Letra t
Disp0 <= "1111001" when EdoActual = A else -- Letra I
         "1111111" when EdoActual = B else  -- Apagado
         "1000001" when EdoActual = C and Dat_Ent_Copia /= Valor_Secreto and Termina = '1' else -- Letra U
         "0010010" when EdoActual = C and Dat_Ent_Copia < Valor_Secreto else -- Letra S
         "0000000" when EdoActual = C and Dat_Ent_Copia > Valor_Secreto else -- Letra B
         "0010000" when EdoActual = C and Dat_Ent_Copia = Valor_Secreto else -- Letra g
         "0001100"; -- Letra P
```

Figura 0.8 Programación de los display's.

Finalmente, el código para la asignación de las terminales en la placa Basys 3 se escribe enseguida:

```
## Interruptores

set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 }
[get_ports {Dat_Ent[0]}]

set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 }
[get_ports {Dat_Ent[1]}]

set_property -dict { PACKAGE_PIN W16 IOSTANDARD LVCMOS33 }
[get_ports {Dat_Ent[2]}]

set_property -dict { PACKAGE_PIN W17 IOSTANDARD LVCMOS33 }
[get_ports {Dat_Ent[3]}]


## Interruptor tipo boton

set_property -dict { PACKAGE_PIN W19 IOSTANDARD LVCMOS33 }
[get_ports Juega]

#set_property -dict { PACKAGE_PIN T17 IOSTANDARD LVCMOS33 }
[get_ports Juega2]

set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 }
[get_ports Verifica]

set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 }
[get_ports Reiniciar]


## Señal de reloj

set_property -dict { PACKAGE_PIN W5 IOSTANDARD LVCMOS33 }
[get_ports CLK]


## Anodos de display

set_property -dict { PACKAGE_PIN U2 IOSTANDARD LVCMOS33 }
[get_ports {control[0]}]

set_property -dict { PACKAGE_PIN U4 IOSTANDARD LVCMOS33 }
[get_ports {control[1]}]

set_property -dict { PACKAGE_PIN V4 IOSTANDARD LVCMOS33 }
[get_ports {control[2]}]

set_property -dict { PACKAGE_PIN W4 IOSTANDARD LVCMOS33 }
[get_ports {control[3]}]


## Display 7 Segmentos

set_property -dict { PACKAGE_PIN W7 IOSTANDARD LVCMOS33 }
[get_ports {Sal7Seg[0]}]
```

```
set_property -dict { PACKAGE_PIN W6 IOSTANDARD LVCMOS33 }
[get_ports {Sal7Seg[1]}]

set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS33 }
[get_ports {Sal7Seg[2]}]

set_property -dict { PACKAGE_PIN V8 IOSTANDARD LVCMOS33 }
[get_ports {Sal7Seg[3]}]

set_property -dict { PACKAGE_PIN U5 IOSTANDARD LVCMOS33 }
[get_ports {Sal7Seg[4]}]

set_property -dict { PACKAGE_PIN V5 IOSTANDARD LVCMOS33 }
[get_ports {Sal7Seg[5]}]

set_property -dict { PACKAGE_PIN U7 IOSTANDARD LVCMOS33 }
[get_ports {Sal7Seg[6]}]
```

Desarrollo

Crea un nuevo proyecto llamado “practica16” y agrégale los archivos para el divisor de frecuencia, el codificador de 7 segmentos y el multiplexor selector ya utilizados anteriormente. Agrega un archivo y llámale “Juego”, en él programaras los componentes de los archivos agregados previamente y realizarás las instanciaciones necesarias de acuerdo con lo que se indica en Análisis previo. Continúa con agregar el código, pero el resto del funcionamiento del juego.

Para el reporte deberás de agregar el diagrama de estados completo, la tabla donde se señalan las entradas con sus respectivas tareas, la tabla de los estados con sus respectivas tareas, la tabla de los estados actuales y siguientes y las condiciones que los llevan a tales nuevos estados, un diagrama general de la máquina de estado finito, y la tabla de los valores de la salida de acuerdo con el estado actual completa (tabla 16.2). Como referencia de cada elemento solicitado en este párrafo, recurrir a la presentación de PowerPoint localizada en Campus Virtual llamada “Circuitos Digitales Semana 15 y 16.pptx”.

Referencias

1. Llanos, Juan; “Vivado Tutorial 1 Crear un proyecto”, https://www.youtube.com/watch?v=YH9PR8ra384&ab_channel=JuanLlanos, septiembre 2022.
2. Llanos, Juan; “Vivado Tutorial 2 Simulación”, https://www.youtube.com/watch?v=sb6AcERnjx0&ab_channel=JuanLlanos, septiembre 2022.
3. Gómez Franco, Maribel; “Manual de Prácticas de Circuitos Digitales”, Universidad Autónoma de Ciudad Juárez, diciembre 2020.