Universidad de Costa Rica

Tarea 2

IE-0523 Circuitos Digitales II

Prof. Enrique Coen Alfaro

Amy Herrera Mora
B53473

5 de mayo de 2024

Índice

1. Resumen **3**

2. Descripción Arquitectónica **3**
2.1. Diagrama de bloques 3
2.2. Diagrama de estados, codificación e implementación de la Máquina de Estados 4

3. Plan de Pruebas **5**

4. Instrucciones de utilización de la simulación **5**

5. Resultados y Análisis **6**
5.1. Síntesis genérica RTLIL 6
5.2. Síntesis mapeada a la biblioteca CMOS 7
5.3. Síntesis con Retardos 8

6. Conclusiones y Recomendaciones **10**
6.1. Conclusiones 10
6.2. Recomendaciones 10

1. Resumen

El objetivo principal de la presente tarea era realizar un proceso de síntesis y verificación para un controlador de estacionamiento utilizando el programa de síntesis Yosys y una biblioteca de elementos lógicos proporcionada en clase. En primer lugar, se realizó una síntesis de alto nivel del diseño conductual de la Tarea 1, lo que generó una descripción estructural genérica independiente de la tecnología utilizada empleando los componentes de la biblioteca interna de Yosys. Luego, se verificó que esta descripción funcionara correctamente mediante pruebas diseñadas previamente. Posteriormente, se seleccionó la biblioteca cmos_cells.lib para realizar el mapeo tecnológico del diseño, creando una descripción estructural utilizando los componentes disponibles en esta biblioteca específica. Se revisó que esta nueva descripción también pasara las pruebas de verificación establecidas.Seguidamente, se modificó el archivo de la biblioteca cmos_cells.v para introducir retardos en la simulación, asegurando que el diseño sintetizado con la biblioteca modificada funcione correctamente.

Finalmente, se evaluó el diseño considerando el número de componentes utilizados, como NAND, NOR, NOT y Flip-Flops, así como el retardo de propagación desde las entradas hasta las salidas, en función de los requisitos y especificaciones del proyecto. Dicho proceso garantizó la correcta síntesis, verificación y evaluación del controlador de estacionamiento, utilizando una combinación de herramientas de síntesis y bibliotecas específicas. El sistema demostró su eficacia y viabilidad al obtener de forma exitosa una descripción estructural de lo que fue la descripción conductual de la Tarea 1. Como principales conclusiones se tiene que el sistema funcionó según lo previsto y demostró su viabilidad con éxito tanto para la descripción conductual como para la descripción estructural. Realizar una síntesis paso a paso y realizar simulaciones en cada parte permitió un buen desarrollo de la Tarea 2. Esto revela que el controlador efectivamente puede convertirse en un circuito físico.

2. Descripción Arquitectónica

2.1. Diagrama de bloques

Haciendo uso de la primera Tarea y algunas modificaciones para que la descripción conductual fuera sintetizable, se hizo uso del diagrama de bloques expuesto en la figura (1). Aquí se observan las entradas en la parte izquierda que se le ingresaran al controlador, y las salidas que se desean de acuerdos a dichas entradas en la parte derecha pero ahora para su descripción estructural.

Figura 1: Diagrama de bloques para el controlador de la Tarea 2

2.2. Diagrama de estados, codificación e implementación de la Máquina de Estados

Una vez construido el diagrama de bloques, se obtuvieron 4 estados para implementar este controlador y se detallan a continuación de acuerdo a su codificación: $IDLE = 000$, $WAIT\_PASSWORD = 001$, $RIGHT\_PASS = 010$ y $LOCKED = 011$. Con la figura (2) se explica el diagrama de estados y las transiciones que se quiere que esta Máquina de Estados cumpla con el fin de obtener un correcto funcionamiento.

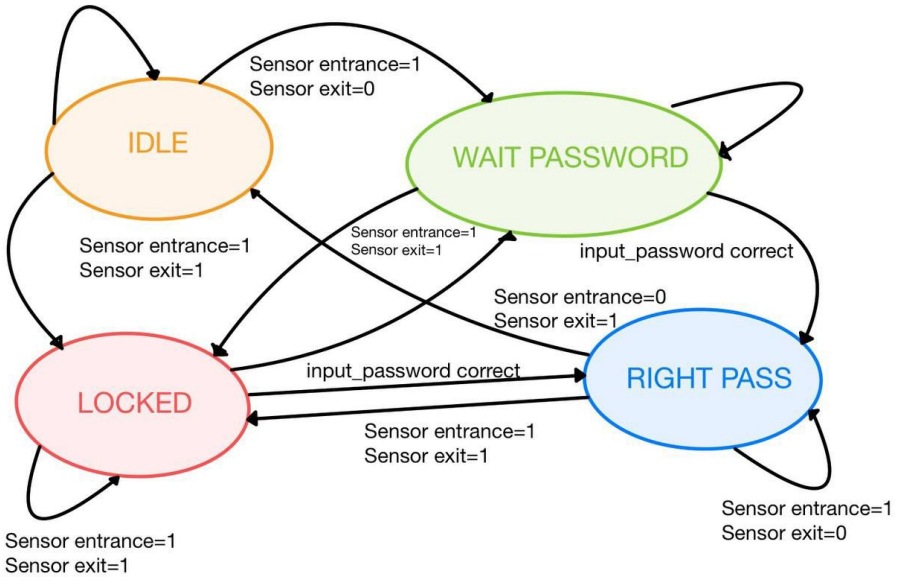

Figura 2: Diagrama de estados para el controlador de la Tarea 2

En esta Máquina de Estados implementada con un bloque $always@(*)$ en Verilog, se inició con un bloque case que depende del estado actual del sistema y se procedió a describir la transición siguiendo el diagrama de estados anteriormente mencionado. Dicha transición se enumerará acontinuación:

1. **IDLE (Inactivo):** En este estado, el sistema está en espera. Si el sensor de entrada está activado, pasa al estado de espera de contraseña. Si ambos sensores están activados, el sistema pasa al estado bloqueado.

2. **WAIT_PASSWORD (Espera de contraseña):** En este estado, el sistema espera que se ingrese una contraseña. Si la contraseña ingresada es correcta, pasa al estado de poder pasar. Si la contraseña es incorrecta pero el contador no ha llegado al valor de máximo de intentos, se incrementa el contador y vuelve a este mismo estado. Pero si la contraseña ingresada es incorrecta y ya se llegó al tope del contador, encendemos alarma de pin incorrecto, reiniciamos el contador y seguimos pidiendo contraseña hasta que sea correcta.

3. **RIGHT_PASS (Contraseña correcta):** Solo en este estado se abre la compuerta. Caso contrario la compuerta siempre permanece cerrada. Aquí reiniciamos el contador, y si obtenemos actividad del sensor de salida porque el vehículo ya termino de ingresar, nos vamos al estado inactivo para esperar mas llegada de autos.

4. **LOCKED (Bloqueado):** Si el sistema está bloqueado y se ingresa la contraseña correcta, vuelve al estado de contraseña correcta para abrir la compuerta. En este estado también se implementa el contador para darle seguimiento a la alarma de pin incorrecto.

5. **default (por defecto):** Si ninguna de las condiciones anteriores se cumple, el sistema vuelve al estado inactivo por defecto que en este caso sería IDLE.

Para la parte del contador y la transición de estados se implementó un bloque *always* @(*posedge clk*, por lo tanto se hace una revisión cada flanco de reloj positivo para que el sistema funcione de manera correcta. De acuerdo a los valores que se inicializan en las entradas del diagrama de bloques, obtendremos valores en las salidas cada flanco positivo de reloj al estar trabajando con Flip Flops. Con esto en mente se procedió a sintetizar la descripción conductual de la Tarea 1 para obtener la descripción estructural en la Tarea 2. Se explicará detalladamente en la sección 5.

3. Plan de Pruebas

En la tabla (1) se puede observan el Plan de Pruebas mínimo para garantizar el corrector funcionamiento ahora de la descripción estructural de este controlador. En ella, se hace referencia al número de prueba, si falló o no, y la descripción de cada una en orden. Dichas pruebas fueron implementadas con un único archivo tester.v tanto para la síntesis genérica RTLIL y la mapeada a la biblioteca CMOS. Esto se hizo en un bloque de initial donde se cambiaban los valores de las entradas para verificar el valor de las salidas con la herramienta gtkwave.

Prueba	**Falló**	**Descripción**
1	No	Prueba de funcionamiento normal básico. Llegada de un vehículo, ingreso del pin correcto y apertura de puerta, sensor de fin de entrada y cierre de compuerta.
2	No	Prueba de ingreso de pin incorrecto menos de 3 veces. Llegada de un vehículo, ingreso de pin incorrecto (una o dos veces), puerta permanece cerrada. Ingreso de pin correcto, funcionamiento normal básico. Revisión de contador de intentos incorrectos.
3	No	Prueba de ingreso de pin incorrecto 3 o más veces. Revisión de alarma de pin incorrecto. Revisión de contador de intentos incorrectos. Ingreso de pin correcto, funcionamiento normal básico. Revisión de limpieza de contadores y alarmas.
4	No	Prueba de alarma de bloqueo. Ambos sensores encienden al mismo tiempo, encendido de alarma de bloqueo, ingreso de clave incorrecta, bloqueo permanece. Ingreso de clave correcta, desbloqueo. Funcionamiento normal básico.

Tabla 1: Descripción de las pruebas realizadas.

4. Instrucciones de utilización de la simulación

Al extraer el archivo B53473.zip se encontrarán 4 directorios. En primer lugar se encuentra *conductual_version* que contiene la descripción conductual del controlador **controller.v**, el **testbench.v** y el **tester.v**; y para su debida compilación y simulación se debe escribir en la terminal

de linux desde el dir B53473 **cd conductual_version** seguido de un **make**. En segundo lugar, se tiene el directorio *rtlil_version* para la versión sintetizada con la biblioteca genérica de Yosys. Dentro de este se encuentran el **controller.v**, el **testbench.v** y el **tester.v**, además del archivo **.ys** y el **.gtkw** para simular de forma mas rápida el correcto funcionamiento. Dentro del dir B53473 se debe escribir en terminal **cd rtlil_version** seguido de un **make** para que se realice la síntesis, compilación y simulación de esta parte al agregar el archivo **.ys** dentro del make para todos los casos de esta Tarea 2. De igual manera se utiliza para el directorio que realiza la síntesis mapeada a la biblioteca CMOS *cmos_version*. Dicho directorio contiene los archivos pertinentes para la síntesis completa del controlador y la simulación de las pruebas al escribir en la terminal desde el dir B53473 **cd cmos_version** seguido de un **make**. Finalmente, para el directorio retardos, se utilizan los mismo archivos del directorio *cmos_version*, con la diferencia de que el archivo **cmos_cells.v** tiene retardos en las salidas para simular un funcionamiento mas real. Para este caso basta digitar en la terminal de linux desde el dir B53473 **cd retardos** seguido de un **make** para observar el funcionamiento.

5. Resultados y Análisis

En la presente tarea se realizaron dos síntesis. En primer lugar, se tiene la síntesis genérica RTLIL para esta primera descripción estructural con la biblioteca por defecto de la herramienta Yosys. El objetivo principal era implementar los comandos de la tabla (2) para ir sintetizando paso a paso según el tipo de síntesis, y luego poder simular el comportamiento del controlador para verificar si funcionaba igual que la descripción conductual de la Tarea 1. Por último, se debía modificar la biblioteca CMOS y agregarle algunos retardos a las salidas de las compuertas para verificar su funcionamiento y observar algunos cambios que se pueden generar en la vida real.

Comando en Yosys	Funcionalidad
read_verilog name_file.v	Lee el archivo que contiene la descripción conductual en Verilog. Convierte los if y los else en árbol de decisión.
hierarchy -check -top -controller	Se utiliza cuando existen varios módulos y necesito establecer una jerarquía
proc	Convierte los procesos a netlists. Depura el árbol de decisión y construye muxes. Sabe la cantidad de componentes que necesitará.
opt	Se utiliza después de cada comando de síntesis para realizar una optimización. Quita celdas vacías e idénticas para simplificar.
fsm	Comando utilizado para detectar una máquina de estados desde la descripción conductual.
memory	Se utiliza por si existen memorias
techmap	Este comando convierte a la biblioteca genérica de Yosys. Se utiliza para mapear a una tecnología. Genera la versión RTLIL de la síntesis.
dfflibmap -liberty ./cmos_cells.lib	Este comando mapea los flip flops al asociar al archivo liberty para que sepa el tipo de FlipFlop, pues este contiene las características físicas de cada celda.
abc -liberty ./cmos_cells.lib	Con el comando de abc se convierte la lógica booleana a la biblioteca de componentes que se quiere usar donde se proporcionan las características físicas de las celdas con las que se quiere trabajar específicamente.
show	Este comando se puede utilizar cada vez que se mapea y se va realizando la síntesis paso a paso para visualizarlo en forma de cables y compuertas.
write_verilog synth_name_file.v	Este comando se utiliza para escribir en forma de código la síntesis. Se puede obtener la síntesis tanto para el mapeo de la biblioteca genérica de Yosys como para la biblioteca cmos que queremos usar.
clean	Utilizado para realizar una limpieza.

Tabla 2: Tabla de comandos para realizar la síntesis con la herramienta Yosys.

5.1. Síntesis genérica RTLIL

El objetivo principal para la síntesis RTLIL era implementar los comandos: *read_verilog controller.v*, *proc*, *opt*, *fsm*, *opt*, *memory*, *opt*, *techmap*, *opt*, *show* y *write_verilog controller_rtlil.v* para observar cada paso según la tabla (2) y así obtener una síntesis genérica antes de utilizar una biblioteca específica. Al realizar un show se pudo observar que ahora el código en verilog está en términos de

algunas compuertas conocidas y desconocidas, además de muxes, constantes y muchos cables hasta este punto.

En la figura (3) se puede apreciar el correcto funcionamiento para las cuatro pruebas que anteriormente se habían implementado en la descripción conductual de este controlador. Se observa la entrada de cada vehículo, el ingreso de las contraseñas correctas e incorrectas, la transición de estados, el contador de pines con su respectiva alarma, la abertura y cierre de la compuerta según el estado en el que se encuentra, y finalmente el estado de bloqueo al encenderse ambos sensores y la activación de la alarma.

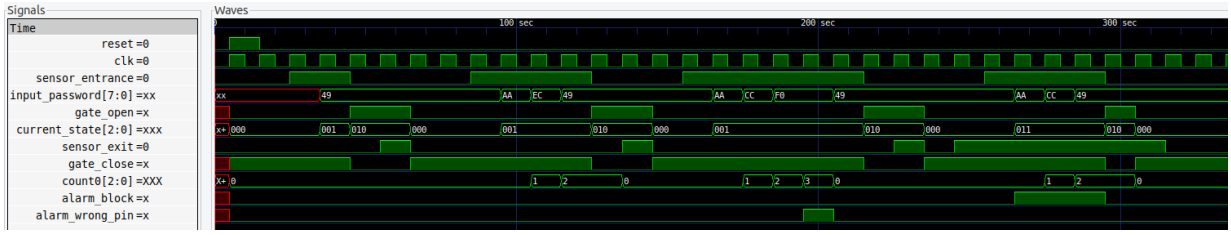

Figura 3: Simulación del correcto funcionamiento de las 4 pruebas con la síntesis genérica RTLIL.

5.2. Síntesis mapeada a la biblioteca CMOS

Seguidamente, se procedió a realizar la misma síntesis, pero utilizando los comandos que mapean a una biblioteca CMOS llamada *cmos_cells.lib* que consta de la caracterización de un buffer, un inversor, una compuerta NAND, una compuerta NOR y un FlipFlop con su área y sus respectivos pines. El objetivo era básicamente implementar los mismos comandos de la tabla (2), pero ahora agregando *dfflibmap −liberty ./cmos_cells.lib*, el cual es el comando para mapear los FlipFlops utilizados para el contador y la transición de estados, y *abc −liberty ./cmos_cells.lib* para mapear la demás lógica combinacional. En las figuras (4) y (5) se puede ver la información desplegada en Yosys desde la consola sobre esta síntesis. Al realizar un resumen en la (3) se presentan la cantidad de compuertas obtenidas para poder realizar este controlador, así como la cantidad de FlipFlops necesarios.

```
11.1. Executing DFFLEGALIZE pass (convert FFs to types supported by the target).
Mapping DFF cells in module `\controller':
  mapped 6 $_DFF_P_ cells to \DFF cells.
```

Figura 4: Cantidad de FlipFlops creados desde la herramienta Yosys para la síntesis CMOS.

```
12.1.2. Re-integrating ABC results.
ABC RESULTS:               NOT cells:       18
ABC RESULTS:              NAND cells:       19
ABC RESULTS:               NOR cells:       33
ABC RESULTS:            _const0_ cells:      1
ABC RESULTS:        internal signals:      131
ABC RESULTS:          input signals:        17
ABC RESULTS:         output signals:        10
Removing temp directory.
```

Figura 5: Cantidad de compuertas creadas desde la herramienta Yosys para la síntesis CMOS.

Cantidad de compuertas obtenidas en la síntesis	
NOT cells	18
NAND cells	19
NOR cells	33
Flip-Flops D	6

Tabla 3: Tabla de componentes

En cuanto a la simulación para ver el correcto funcionamiento de las 4 pruebas de la tabla (1), se tiene la figura (6). Al realizar la compilación desde el archivo **testbench.v** se tuvo que hacer el include de *cmos_cells.v*, pues esta implementa como módulos cada compuerta para que Verilog entienda el mapeo y se pueda simular. Se observa la entrada de cada vehículo, el ingreso de las contraseñas correctas e incorrectas, la transición de estados, el contador de pines con su respectiva alarma, la abertura y cierre de la compuerta según el estado en el que se encuentra, y finalmente el estado de bloqueo al encenderse ambos sensores y la activación de la alarma. Al igual que en la versión RTLIL, se puede ver que el controlador funciona también pero ahora con su descripción estructural utilizando solamente compuertas NOT, NAND, NOR y Flip-Flops, ya que si observamos el archivo generado con show en la síntesis se podrá ver que el circuito ahora es más simplificado que la versión RTLIL. Este resultado es de suma importancia, pues en la industria parte de los aspectos más relevantes a evaluar en un circuito es su área y el consumo de potencia; y cuando se implementan controladores interesa que realicen su funcionamiento, pero que también se puedan construir en la vida real, y esto sólo se garantiza con la síntesis. Además, el hecho de que se haya logrado una síntesis con compuertas específicas dadas logra la minimización buscada por un proceso de síntesis donde existen limitaciones para poder realizar una descripción estructural con lo que se tenga disponible.

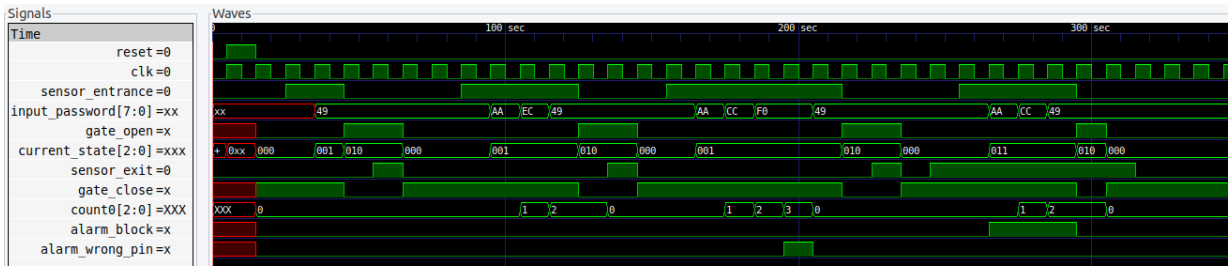

Figura 6: Simulación del correcto funcionamiento de las 4 pruebas con la síntesis mapeada a la biblioteca CMOS.

5.3. Síntesis con Retardos

Para esta parte se utilizó la síntesis realizada anteriormente mapeada a la biblioteca $cmos_cells$ para simular el controlador y revisar las 4 pruebas del tester.v. En este caso, los retardos consistían en agregar retrasos en las salidas de las compuertas de la biblioteca para adecuar la simulación a lo que sucede en la vida real, pues los retardos son ese tiempo que dura una salida en cambiar cuando la entrada cambia, y que toda lógica secuencial y combinacional posee. En la tabla (4) se pueden observar los retardos agregados en el file cmos_cells.v para esta parte.

Retardos de forma arbitraria	Unidades de tiempo
Buffer cells	1
NOT cells	2
NAND cells	2
NOR cells	2
Flip-Flops D	5

Tabla 4: Tabla de retardos

Al retrasar el funcionamiento tanto de la lógica combinacional como secuencial, se tuvo que modificar el tester para aumentar el periodo del reloj. Asimismo se tuvieron que duplicar las unidades de tiempo en las que ocurrían los cambios en las entradas, pues ahora hay que tomar en cuenta que el funcionamiento no es ideal en cada flanco de reloj como en los dos casos anteriores, y que hay que darle tiempo a las compuertas de que hagan su trabajo. En la figura (7) podemos observar para las 4 pruebas de la tabla (1) que efectivamente las transiciones y cambios en las salidas del sistema ocurren unidades de tiempo después de cada flanco de reloj positivo.

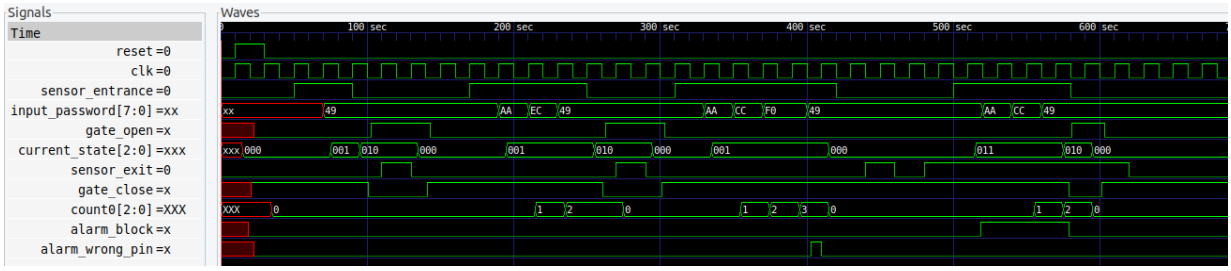

Figura 7: Simulación de la 4 pruebas con los retardos implementados.

De forma detallada, para la prueba 2 de la tabla (1) se puede apreciar para el contador y la transición de estados que ocurren 5 unidades luego de cada flanco positivo correspondiente debido a que se sintetizaron con FlipFlops. De igual manera, la respuesta que se obtiene desde la lógica combinacional también se retrasa, y por esta razón se dura más abriendo o cerrando la compuerta, y activando las alarmas de pin incorrecto y de estado de bloqueo al activarse ambos sensores. Este resultado es de suma relevancia a la hora de realizar síntesis, pues realmente se están construyendo circuitos cuya velocidad es indispensable. Es importante que un circuito lógico funcione con un reloj apropiado cuando hay retardos porque este ayuda a sincronizar las operaciones dentro del circuito. Cuando hay retardos, es crucial que las señales de entrada y salida se coordinen adecuadamente para evitar errores y garantizar el funcionamiento correcto del circuito.

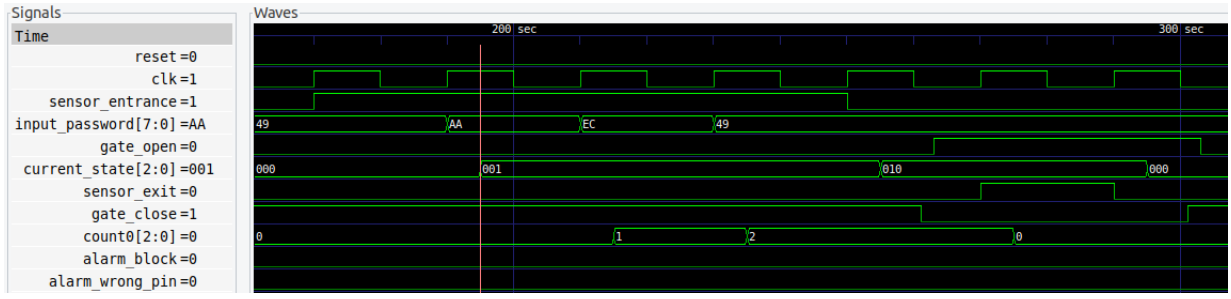

Figura 8: Prueba 2 con retardos en las compuertas.

6. Conclusiones y Recomendaciones

6.1. Conclusiones

1. El sistema funcionó según lo previsto y demostró su viabilidad con éxito tanto para la descripción conductual como para la descripción estructural. Realizar una síntesis paso a paso y realizar simulaciones en cada parte permitió un buen desarrollo de la Tarea 2.

2. Las simulaciones realizadas en el archivo tester.v fueron todas exitosas y consistentes para cada una de las síntesis realizadas. Esto revela que el controlador efectivamente puede convertirse en un circuito físico.

3. Los problemas encontrados durante el proceso, como las partes de la descripción conductual que no se podían sintetizar, fueron superados con recursos disponibles. La documentación, los archivos de clase sobre máquinas de estado en Verilog y la práctica fueron fundamentales para el éxito alcanzado en esta tarea.

4. Se identificó un desafío particular en el manejo de la inicialización tanto de la máquina de estados al utilizar un reset como de las señales de entrada, pues es primordial que los cambios sean detectados en flancos positivos de reloj al estar utilizando Flip-Flops para evitar valores desconocidos en las simulaciones.

6.2. Recomendaciones

1. Priorizar la fase de diseño inicial para la descripción conductual, con la creación detallada de diagramas y la definición clara de requisitos y funcionalidades del sistema, es fundamental, pues no todo código es sintetizable, y en ocasiones se deben realizar modificaciones para obtener una buena descripción estructural.

2. Realizar pruebas exhaustivas del sistema mediante simulaciones en diferentes escenarios y condiciones para garantizar su robustez y fiabilidad. Esto permite observar el funcionamiento de nuestro trabajo para poder identificar mejor las fallas, y así poder corregir cualquier problema que se presente.

3. Pedir ayuda externa, ya sea en horas consulta, es indispensable, pues existen muchos errores de funcionalidad que no pueden ser detectados por una única persona que diseña circuitos digitales.

4. Familiarizarse profundamente con el lenguaje de programación Verilog antes de iniciar el desarrollo. Esto incluye entender los conceptos básicos de la sintaxis, la estructura del código y las mejores prácticas de diseño.

5. Continuar aprendiendo y practicando el uso de Verilog y otras herramientas de diseño de hardware para mejorar constantemente las habilidades y la eficiencia en futuros proyectos similares.