Universidad de Costa Rica

Tarea 1

IE-0523 Circuitos Digitales II

Prof. Enrique Coen Alfaro

Amy Herrera Mora
B53473

16 de abril de 2024

Índice

1. Resumen

La tarea inicial consistió en crear un sistema automatizado para controlar el acceso de vehículos a un estacionamiento, siguiendo un conjunto de directrices específicas. Al ingresar al estacionamiento, un sensor detecta la llegada de un vehículo, solicitando al conductor una clave binaria de 8 bits. Si la clave ingresada es correcta, se emite una señal para abrir la compuerta y permitir el acceso; de lo contrario, la compuerta permanece cerrada. Tras tres intentos fallidos, se activa una alarma por clave incorrecta.

Una vez dentro, otro sensor indica la presencia total del vehículo, activando la señal para cerrar la compuerta. Si ambos sensores se activan simultáneamente, se bloquea la compuerta y se activa una alarma de bloqueo, que solo se desactiva ingresando la clave correcta.

La implementación se realizó mediante una máquina de 4 estados en Verilog, dividida en tres archivos principales: controller.v para la lógica secuencial y combinacional, tester.v para las pruebas e inicialización de señales y testbench.v como punto de conexión entre los módulos para instanciarlos. Se realizaron 4 pruebas que verificaban el correcto funcionamiento de las especificaciones dadas anteriormente mencionadas y todas funcionaron. Las alarmas se encienden donde deben ponerse en alto y la activación de cerrado o abierto de puerta no coinciden nunca.

El sistema demostró su eficacia y viabilidad, gracias a la precisión y control que brindó la máquina de estados. La planificación detallada, desde el diseño inicial hasta los diagramas de transición de estados, facilitó la implementación en Verilog. Se recomienda una comprensión profunda de Verilog antes de iniciar el desarrollo, incluyendo su sintaxis, estructura y mejores prácticas de diseño.

2. Descripción Arquitectónica

2.1. Diagrama de bloques

Para la primera Tarea, en donde se debió describir conductualmente un controlador automatizado para la entrada de un estacionamiento, se diseñó el 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.

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

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

Una vez construido el diagrama de bloques, se procedió a pensar en cuáles estados se querían para la máquina de estados. Para este caso, 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.

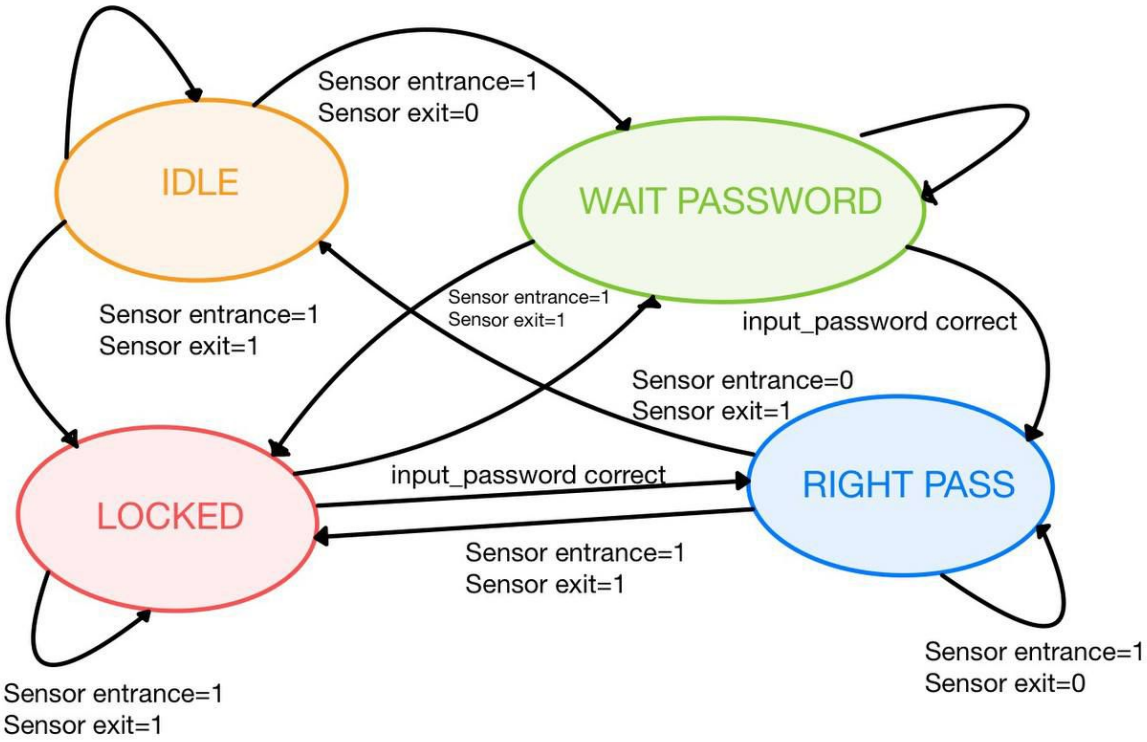

Figura 2: Diagrama de estados para el controlador de la Tarea 1

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 y el sensor de salida está desactivado, pasa al estado de espera de contraseña. Si ambos sensores están activados, el sistema pasa al estado bloqueado. De lo contrario, permanece en el estado inactivo.

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 contraseña correcta. Si ambos sensores están activados, el sistema pasa al estado bloqueado. De lo contrario, permanece en el estado de espera de contraseña.

3. **RIGHT_PASS (Contraseña correcta):** Si la contraseña ingresada es correcta, el sistema verifica la actividad de los sensores de entrada y salida para determinar el próximo estado. Si ambos sensores están activados, pasa al estado bloqueado. Si solo el sensor de entrada está activado, permanece en el estado de contraseña correcta. Si solo el sensor de salida está activado, regresa al estado inactivo.

4. **LOCKED (Bloqueado):** Si el sistema está bloqueado y se ingresa la contraseña correcta, vuelve al estado de contraseña correcta. Si ambos sensores se activa, se mantiene en este estado. De lo contrario, vuelve al estado de espera de contraseña.

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 cuenta de las contraseñas incorrectas ingresadas se implementó un bloque *always* @(*posedge clk or negedge* !*reset_n*), 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:

1. `gate_open`: Se activa cuando el estado actual es `RIGHT_PASS` y la contraseña ingresada es correcta.

2. `gate_close`: Se activa cuando el estado actual es `LOCKED`, `IDLE`, `WAIT_PASSWORD` o `ALARM_PIN`.

3. `alarm_wrong_pin`: Se activa cuando el contador de intentos de contraseña incorrecta supera el máximo permitido.

4. `alarm_block`: Se activa cuando el sistema está en estado `LOCKED` o estamos en `WAIT_PASSWORD` con ambos sensores activados.

3. Plan de Pruebas

En la tabla (1) se puede observan el Plan de Pruebas mínimo para garantizar el corrector funcionamiento de este diseño de 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 en el archivo tester.v 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	Fallo	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

El primer archivo, **controller.v**, contiene la lógica secuencial y combinacional necesaria para un funcionamiento correcto. El segundo archivo, **tester.v**, está diseñado para inicializar las entradas del sistema y obtener los valores de salida en cada flanco de reloj positivo y generar las simulaciones. Finalmente, el tercer archivo, **testbench.v**, sirve como punto de conexión entre los módulos **controller.v** y **tester.v**. En este archivo se instancian ambos módulos y se declaran los wires necesarios para conectar las salidas del controlador con las entradas del tester. Por esta razón basta con entrar al directorio donde están los tres archivos y digitar en la consola de linux **iverilog testbench.v** para generar el ejecutable llamado a.out. Una vez que se genera, se corre el comando **vvp a.out** para poder ver el funcionamiento en consola y generar el archivo resultados.vcd. Finalmente, se escribe en la terminal de linux **gtkwave resultados.vcd** para poder observar las simulaciones. Estos pasos se resumen en un Makefile adjunto donde sólo basta digitar el comando **make** para poder mostrar el correcto funcionamiento.

5. Resultados y Análisis

5.1. Prueba 1: Funcionamiento básico

En esta prueba se debía verificar el funcionamiento normal básico del controlador. Dicha prueba constaba de la llegada de un vehículo, el ingreso del pin correcto y apertura de puerta, verificación del sensor de fin de entrada y cierre de compuerta. En el siguiente código se aprecia la inicialización de las señales de entrada en cero, y luego cómo se van cambiando para producir valores deseados en las salidas.

```
clk = 0;
reset_n = 0;
sensor_entrance = 0;
```

```
sensor_exit = 0;
#10 reset_n = 1;

//Prueba #1: Funcionamiento normal bsico
$display("Prueba␣#1:␣Funcionamiento␣normal␣bsico");
#5 sensor_entrance = 1; // Llegada de vehculo
#5 input_password = 8'b01001001; // Ingresa contrasea correcta: 01001001
//#10; // Retardo de 100 unidades de tiempo
$display("Esperando␣que␣la␣compuerta␣se␣abra...");
#5 sensor_entrance = 0; // Sensor en bajo ya que est terminando de ingresar un vehculo
#5 sensor_exit = 1; // Sensor de que ya el vehculo ingres
$display("Esperando␣que␣la␣compuerta␣se␣cierre...");
#5 sensor_exit = 0;
#10; //Delay
```

En la figura (3) se aprecia que llega un vehículo, se ingresa el pin correcto, y una vez que el pin es correcto la *gate_open* se pone en 1 para abrirse en el flanco positivo del reloj. Luego, se verifica con *sensor_exit* que el carro ya haya ingresado para que *gate_close* cierre la puerta y espere un segundo vehículo.

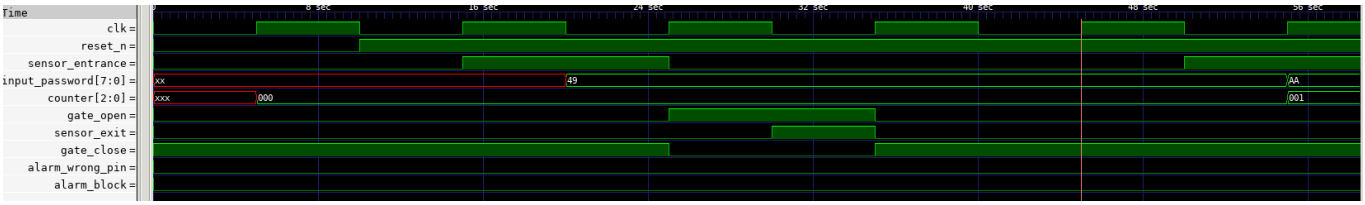

Figura 3: Prueba 1: Funcionamiento básico

5.2. Prueba 2: Ingreso de pin incorrecto menos de 3 veces

En la segunda prueba se debía verificar la llegada de un vehículo, luego el ingreso de una o dos contraseñas incorrectas para demostrar que la puerta permanece cerrada. Seguidamente, al ingresar el pin correcto, se procedía al funcionamiento normal básico. Se debía revisar el contador de intentos incorrectos.

```
// Prueba #2: Ingreso de PIN incorrecto menos de 3 veces
$display("Prueba␣#2:␣Ingreso␣de␣PIN␣incorrecto␣menos␣de␣3␣veces");
#5 sensor_entrance = 1; // Llegada de vehculo
$display("Esperando␣contrasea...");
#5 input_password = 8'b10101010; // Ingreso de primer pin incorrecto
#5;
$display("Esperando␣contrasea...");
#5 input_password = 8'b11101100; // Ingreso de segundo pin incorrecto
$display("Esperando␣contrasea...");
#5;
#5 input_password = 8'b01001001; // Ingreso de pin correcto a la tercera vez
// Esperar a que la compuerta se abra 01001001
$display("Esperando␣que␣la␣compuerta␣se␣abra...");
#5 sensor_entrance = 0; // Sensor en bajo ya que est terminando de ingresar un vehculo
#5 sensor_exit =1;
```

```
// Esperar a que la compuerta se cierre
$display("Esperando␣que␣la␣compuerta␣se␣cierre...");
#5 sensor_exit = 0;
```

En la figura (4) se aprecia que llega un vehículo aproximadamente a los 50 s que ingresó el carro de la Prueba 1. Luego, se ingresan dos pines incorrectos, y se muestra el valor que lleva el contador binario ya que sólo cuenta las veces que se ingresó mal el pin. Una vez que el pin es correcto la *gate_open* se pone en 1 para abrirse y el contador se reinicia. Luego, se verifica con *sensor_exit* que el carro ya haya ingresado para que *gate_close* cierre la puerta y espere un tercer auto.

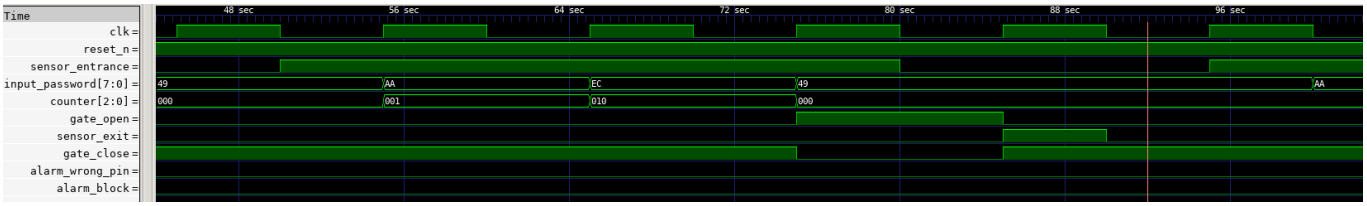

Figura 4: Prueba 2: Ingreso de pin incorrecto menos de 3 veces

5.3. Prueba 3: Ingreso de pin incorrecto 3 o más veces

Esta tercera prueba consistía en la revisión de la alarma de pin incorrecto si el contador es mayor o igual a 3 intentos. Luego se revisaba el valor de dicho contador de intentos incorrectos. Seguidamente, se ingresa el pin correcto para tener un funcionamiento normal básico. Finalmente, se hace la revisión de limpieza de contadores y la alarma de pin incorrecto.

```
// Prueba #3: Ingreso de pin incorrecto 3 veces
$display("Prueba␣#3:␣Ingreso␣de␣pin␣incorrecto␣3␣veces");
#5 sensor_entrance = 1; // Llegada de vehculo
#5 input_password = 8'b10101010; // Ingreso de primer pin incorrecto (revisar alarma
    de pin incorrecto)
#5;
#5 input_password = 8'b11001100; // Ingreso de segundo pin incorrecto
#5;
#5 input_password = 8'b11110000; // Ingreso de tercer pin incorrecto
#5;
#5 input_password = 8'b01001001; // Ingresa contrasea correcta: 01001001
$display("Esperando␣que␣la␣compuerta␣se␣abra...");
#5 sensor_entrance = 0; // Sensor en bajo ya que est terminando de ingresar un vehculo
#5 sensor_exit = 1; // Sensor de que ya el vehculo ingres
$display("Esperando␣que␣la␣compuerta␣se␣cierre...");
#5 sensor_exit = 0;
```

En la figura (5) se aprecia que llega un tercer vehículo como a los 90 s, se realiza el ingreso de tres contraseñas incorrectas, con cada ingreso de contraseña incorrecta se aprecia el valor del contador en tiempo real. Cuando se llega a 3 o más el valor del contador, se aprecia que a los 125 s la alarma de pin incorrecto se pone en alto. Una vez que el pin es correcto la *gate_open* se pone en 1 para abrirse, se reinicia el contador y la alarma de pin incorrecto se pone en bajo. Luego, se verifica con *sensor_exit* que el carro ya haya ingresado para que *gate_close* cierre la puerta y espere un cuarto auto.

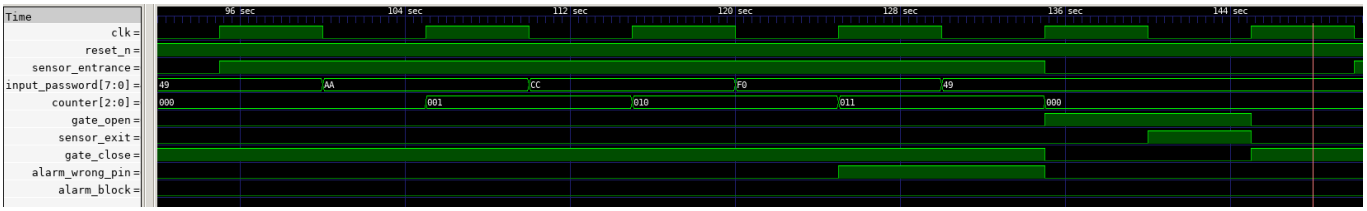

Figura 5: Prueba 3: Ingreso de pin incorrecto 3 o más veces

5.4. Prueba 4: Alarma de bloqueo

En esta cuarta prueba, ambos sensores se encienden al mismo tiempo para verificar el encendido de alarma de bloqueo. Seguidamente se ingresa la clave incorrecta para que el bloqueo permanezca. Finalmente, se realiza el ingreso de la clave correcta para desbloquear el sistema y que haya un funcionamiento normal básico.

```
//Prueba #4
 $display("Prueba␣#4:␣alarma␣de␣bloqueo");
 #5 sensor_entrance = 1; // Llegada de vehculo
 #5 sensor_exit = 1; // Sensor que ya el vehculo entr (ambos sensores activados al
     mismo tiempo)
 #5 input_password = 8'b10101010; // Ingreso de pin incorrecto (alarma de bloqueo debe
     seguir en alto)
 #5;
 #5 input_password = 8'b11001100; // Ingreso de pin incorrecto (sigue en alto alarma de
      bloqueo)
 #5;
 #5 input_password = 8'b01001001; // Ingresa contrasea correcta: 10010010
 #5;
 $display("Esperando␣que␣la␣compuerta␣se␣abra...");
 #5 sensor_entrance = 0; // Sensor en bajo ya que est terminando de ingresar un vehculo
 #5 sensor_exit = 1; // Sensor de que ya el vehculo ingres
 $display("Esperando␣que␣la␣compuerta␣se␣cierre...");
 #5 sensor_exit = 0;
 #400 $finish; // End simulation
```

En las figuras (6) se observa que en aproximadamente 150 s llega un último vehículo, sin embargo, al estar ambos sensores en alto, se hace transición al estado LOCKED. Luego, se ingresa un total de dos pines incorrectos como se aprecia en los valores del contador y el sistema sigue bloqueado. Una vez que el pin es correcto la *gate_open* se pone en 1 para abrirse, el contador se reinicia y el sistema se desbloquea para funcionar normalmente. Luego, se verifica con *sensor_exit* que el carro ya haya ingresado para que *gate_close* cierre la puerta y espere un siguiente auto si así se desea.

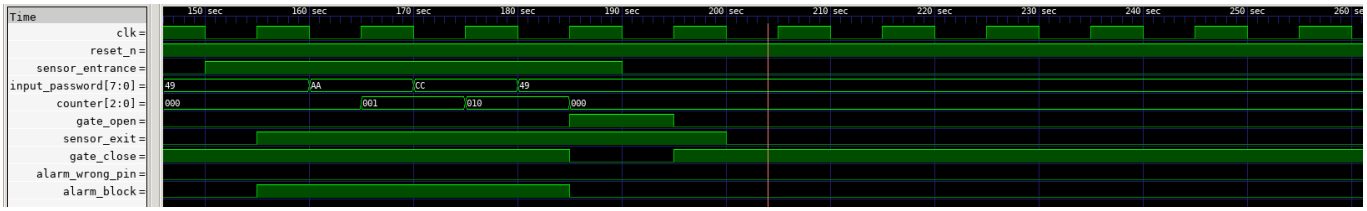

Figura 6: Prueba 4: Alarma de bloqueo

Cabe recalcar que las 4 simulaciones se realizaron en un mismo tester.v variando las unidades de tiempo donde cambiaban las entradas. Se aprecia también que *gate_close* comienza cerrada y no cambia a bajo si no es porque *gate_open* se ha abierto.

6. Conclusiones y Recomendaciones

6.1. Conclusiones

1. El sistema funcionó según lo previsto y demostró su viabilidad con éxito. La utilización de una máquina de 4 estados permitió un control efectivo y preciso de las operaciones de entrada, clave de acceso y gestión de la compuerta.

2. Las simulaciones realizadas en el archivo tester.v fueron todas exitosas y consistentes, validando el funcionamiento correcto del sistema en diferentes condiciones.

3. Se destaca la importancia de realizar un diseño detallado como paso inicial. El uso de papel, diagramas de bloques y diagramas de transición de estados facilitó enormemente la implementación posterior del sistema en Verilog, asegurando una estructura clara y coherente.

4. Los problemas encontrados durante el proceso, como la falta de familiaridad inicial con el lenguaje Verilog, fueron superados con recursos disponibles, como blogs y tutoriales. La documentación y la práctica fueron fundamentales para comprender la sintaxis y la estructura del código.

5. Se identificó un desafío particular en el manejo de la inicialización de las señales, especialmente en el caso de la contraseña. Se concluyó que dejar la contraseña en valores desconocidos al principio evita posibles interferencias con el contador de intentos incorrectos, mejorando así la robustez del sistema.

6.2. Recomendaciones

1. Priorizar la fase de diseño inicial, incluyendo la creación detallada de diagramas y la definición clara de requisitos y funcionalidades del sistema.

2. 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.

3. Utilizar recursos disponibles, como blogs, tutoriales y documentación oficial, para superar cualquier obstáculo relacionado con la implementación técnica.

4. Realizar pruebas exhaustivas del sistema mediante simulaciones en diferentes escenarios y condiciones para garantizar su robustez y fiabilidad antes de una posible implementación física.

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.