Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0523 Circuitos Digitales II
I ciclo 2024

Tarea #2 Descripción estructural del controlador de estacionamiento

Oscar Porras Silesky C16042
Grupo 001

Profesor: Ing. Enrique Coen

4 de mayo de 2023

Índice

1. Resumen.

Este proyecto presenta la descripción estructural del controlador de estacionamiento. Por medio de sensores y actuadores, el sistema gestiona la entrada de vehículos de forma segura y eficiente, validando códigos de acceso binarios de 8 bits.

El éxito del proyecto se evidencia en la capacidad del controlador para manejar correctamente todas las situaciones previstas, desde la detección de la llegada de un vehículo hasta la activación de una alarma tras múltiples intentos de acceso incorrectos.

Las pruebas realizadas demuestran la fiabilidad del controlador en diversas circunstancias, incluyendo el manejo adecuado del cierre de compuerta tras el ingreso del vehículo y la activación de alarmas.

Lo que distingue a este diseño es, que a diferencia del anterior, se implementó el clock para su sintetización con Flip Flops, y compuertas NAND, NOR, y NOT.

2. Descripción Arquitectónica.

2.1. Diagrama de Bloques del controlador.

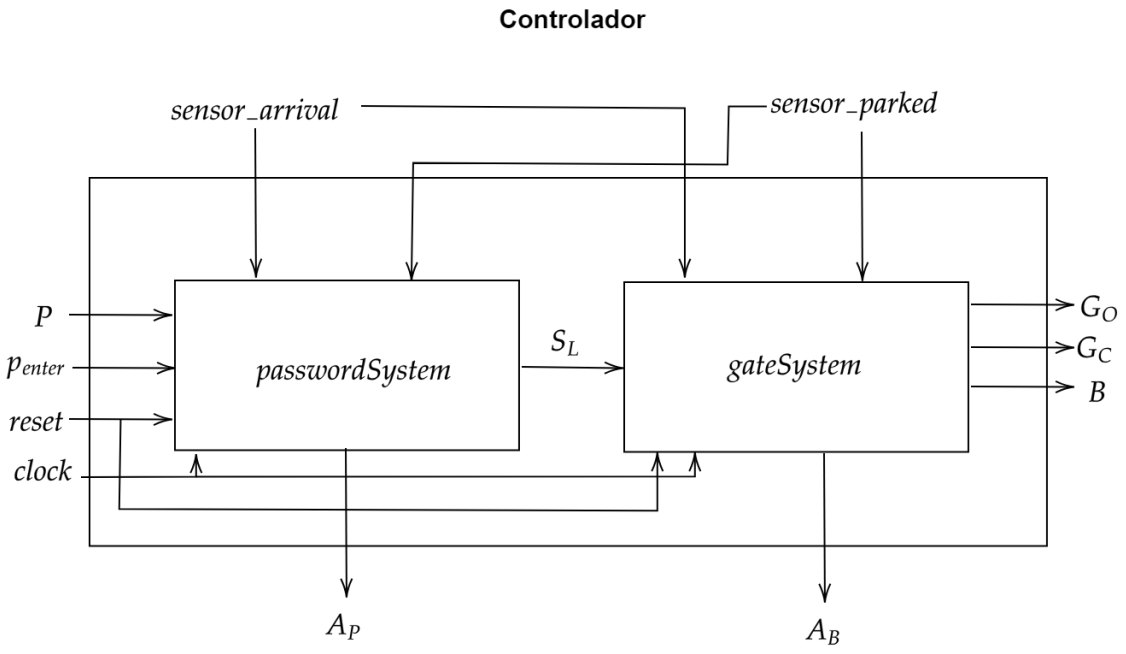

Figura 1: Archivos oculto y no oculto con comando ls y ls -a.

En la figura del controlador, se muestran las interacciones entre las señales del sistema y los módulos `passwordSystem` y `gateSystem`. Las señales y su propósito son los siguientes:

- `P`: Contraseña proporcionada por el usuario, representada como un valor binario de 8 bits.
- `p_enter`: Señal de confirmación de entrada de la contraseña. Cuando se activa, el sistema procede a verificar la contraseña ingresada.
- `reset`: Señal utilizada para reiniciar el sistema y restablecer alarmas y contadores de intentos fallidos. Esta señal solo fue utilizada al principio, para un inicio y limpio.

- `sensor_arrival`: Sensor que detecta la llegada de un vehículo en la entrada del estacionamiento.
- `sensor_parked`: Sensor que detecta si un vehículo está estacionado y por lo tanto, si la compuerta debe permanecer cerrada.
- `S_L` (*Señal de Luz*): Señal de salida del módulo `passwordSystem` que indica si la contraseña proporcionada es correcta.
- `A_P` (*Alarma de Password*): Alarma que se activa cuando se ingresan tres contraseñas incorrectas consecutivas.
- `G_O`: Controla la apertura compuerta basándose en la validez de la contraseña y la señal del sensor de vehículo estacionado.
- `G_C`: Controla el cierre de la compuerta basándose en la validez de la contraseña y la señal del sensor de vehículo estacionado.
- `B`: Señal de bloqueo del sistema que se activa si se presenta una condición de error.
- `A_B` (*Alarma de Bloqueo*): Alarma que se activa cuando el sistema se encuentra en estado de bloqueo.
- `clock` (*Clock*): Clock utilizado para el funcionamiento del sistema.

2.2. Diagrama de Bloques del funcionamiento de los archivos.

Es importante destacar que `pruebas.v` es el archivo con los módulos en lógica secuencial, `both_RTLIL.v` es el archivo con los módulos en RTLIL y `both_synth.v` es el archivo con los módulos sintetizados. Donde `pruebas.v` se puede intercambiar con alguno de los otros dos. En el caso de `testbench.v` también existe `testbench_retardo.v`, para el retardo en la sintetización final.

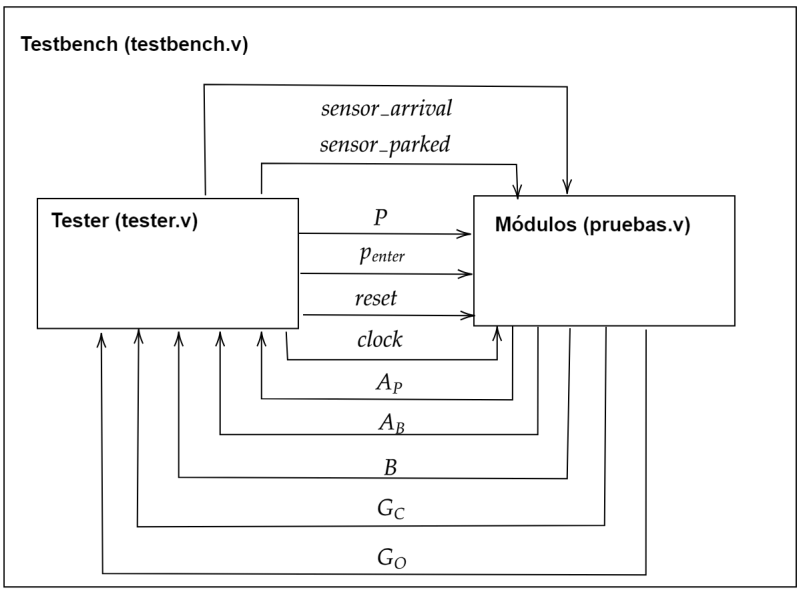

Figura 2: Diagrama de bloques de cómo los archivos se comunican.

2.3. Diagrama de estados.

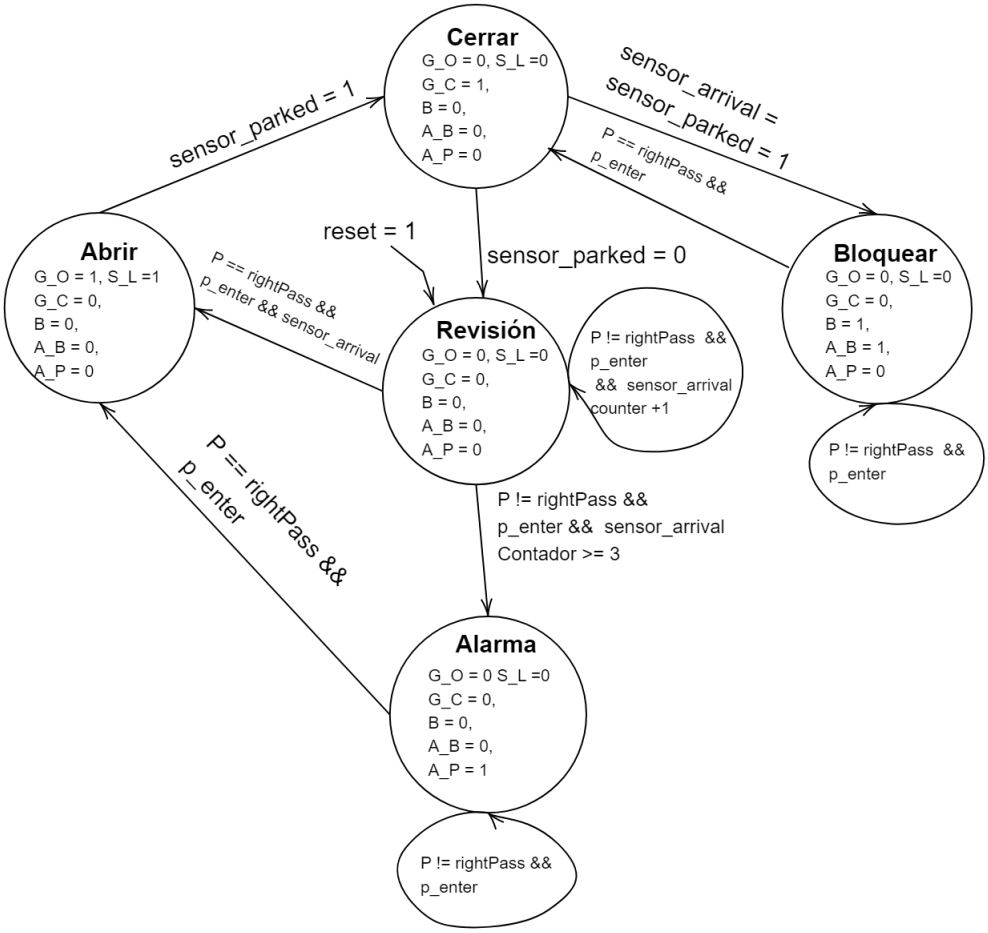

Figura 3: Diagrama de estados del sistema entero.

3. Plan de Pruebas.

El plan de pruebas se diseñó para validar el funcionamiento del controlador de acceso a estacionamiento automatizado. Se enumeran a continuación las pruebas realizadas, describiendo cada una con su propósito específico, los pasos ejecutados, y los resultados esperados. Se destaca que no se utilizó un *clock* para sincronizar las pruebas, aprovechando la respuesta inmediata que permite la lógica conductual del diseño frente a las señales de los sensores.

1. **Prueba #1: Funcionamiento normal básico.**
   - *Descripción:* Verificación de la funcionalidad básica del sistema con un vehículo que llega y proporciona una clave correcta.
   - *Pasos:*
     i. Envío de pulso de reset.
     ii. Simulación de la llegada de un vehículo.
     iii. Ingreso de la clave correcta.

iv. Actuación de la puerta para permitir el acceso al vehículo.
v. Verificación del cierre de la puerta una vez estacionado el vehículo.

- *Resultado esperado:* El sistema abre y luego cierra la compuerta sin errores.

2. **Prueba #2: Ingreso de pin incorrecto menos de 3 veces.**

- *Descripción:* Respuesta del sistema a intentos de acceso con clave incorrecta seguidos por el ingreso correcto de la misma.
- *Pasos:*
  i. Reset del sistema.
  ii. Simulación de llegada de otro vehículo.
  iii. Dos intentos de acceso con claves incorrectas.
  iv. Ingreso de la clave correcta.
- *Resultado esperado:* La compuerta permanece cerrada durante intentos incorrectos y se abre con el ingreso correcto.

3. **Prueba #3: Ingreso de pin incorrecto 3 o más veces.**

- *Descripción:* Prueba de la activación de la alarma tras tres ingresos incorrectos consecutivos.
- *Pasos:*
  i. Envío de pulso de reset.
  ii. Simulación de llegada de un nuevo vehículo.
  iii. Tres intentos consecutivos con claves incorrectas.
  iv. Ingreso de clave correcta para resetear la alarma.
- *Resultado esperado:* Alarma activada tras tres intentos incorrectos y desactivada con el ingreso correcto.

4. **Prueba #4: Alarma de bloqueo.**

- *Descripción:* Esta prueba evalúa la lógica de bloqueo del sistema cuando se detecta simultáneamente la llegada de un vehículo y un vehículo estacionado, una condición que no debería ocurrir bajo operación normal y que, por lo tanto, debe activar una alarma de bloqueo.
- *Pasos:*
  i. Se envía un pulso de reset para inicializar el sistema.
  ii. Se simula la llegada de un vehículo mientras ningún vehículo está estacionado.
  iii. Se activan ambos sensores para simular una situación de bloqueo.
  iv. Se introduce una contraseña incorrecta manteniendo el estado de bloqueo.
  v. Luego se ingresa la contraseña correcta para verificar que el sistema se desbloquee correctamente.
  vi. Se desactiva el sensor de llegada y se espera para simular que el vehículo ya no está llegando.
  vii. Finalmente, se simula la salida del vehículo estacionado para restablecer el sistema y prepararlo para la siguiente entrada.
  viii. Se simula la prueba número 1 y debe funcionar con normalidad.

- *Resultado esperado:* El sistema debe reconocer una condición anormal y activar la alarma de bloqueo. Después de ingresar la contraseña correcta, el sistema debe desbloquearse, lo cual se verifica al permitir la entrada del siguiente vehículo sin activar la alarma de bloqueo.

Cada prueba fue ejecutada de manera aislada para verificar las reacciones específicas del controlador y asegurar cobertura total de todos los escenarios posibles. Las simulaciones se realizaron utilizando herramientas estándar como Icarus Verilog para observar el comportamiento del sistema ante cada conjunto de entradas.

4. Instrucciones de utilización de la simulación.

4.1. Makefile.

Para facilitar el proceso de compilación y simulación, se ha proporcionado un `Makefile`. Para utilizar el archivo `Makefile`, este debe estar en la misma carpeta que el resto de archivos, al igual que los archivos `cmos_cells.v`, `cmos_cells.lib`, `synthesis.ys` y `C16042gtkwaveTarea2.gtkw`. **No olvidar colocar los barra baja en los nombres a la hora de correr los comandos.**

4.1.1. Para solamente compilar y simular el caso conductual:

Se debe correr el comando `make view_conductual`

4.1.2. Para solamente compilar y simular el caso RTLIL:

Se debe correr el comando `make view_RTLIL`

4.1.3. Para solamente compilar y simular el caso sintetizado:

Se debe correr el comando `make view_synth`

4.1.4. Para solamente compilar y simular el caso sintetizado con retardo:

Se debe correr el comando `make view_retardo`

4.1.5. Para compilar y simular todos los casos:

Para ejecutar todos los comandos automáticamente, se debe correr el comando `make` en la terminal. Este comando compilará el testbench y los módulos de diseño, ejecutará la simulación para generar el archivo de ondas y abrirá GTKWave con la configuración predefinida para visualizar los resultados. Es importante recalcar que este comando correrá las 4 simulaciones, solo se le debe dar a la X de cerrar ventana del visor de ondas de GTKWave y correrá la siguiente simulación.

4.1.6. Para limpiar todo:

Luego de revisar las ondas y cuando se desee limpiar el directorio de trabajo, se usa el comando `make clean` para eliminar todos los archivos generados durante la compilación y la simulación.

5. Ejemplos de los resultados.

Los resultados que se van a presentar son las 4 simulaciones en su totalidad, y después se analizarán ciertos detalles de las versiones sintetizada y sintetizada con retardo. Esto debido a que el análisis de las respuestas a cada prueba ya se realizó en la Tarea 1. La prueba número 1 corresponde desde los 0 segundos hasta antes de los 200 segundos. La prueba 2 corresponde desde después de los 200 segundos hasta antes de los 500 segundos. La prueba 3 corresponde desde después de los 500 segundos hasta antes de los 900 segundos. La prueba 4 corresponde desde después de los 900 segundos hasta el final.

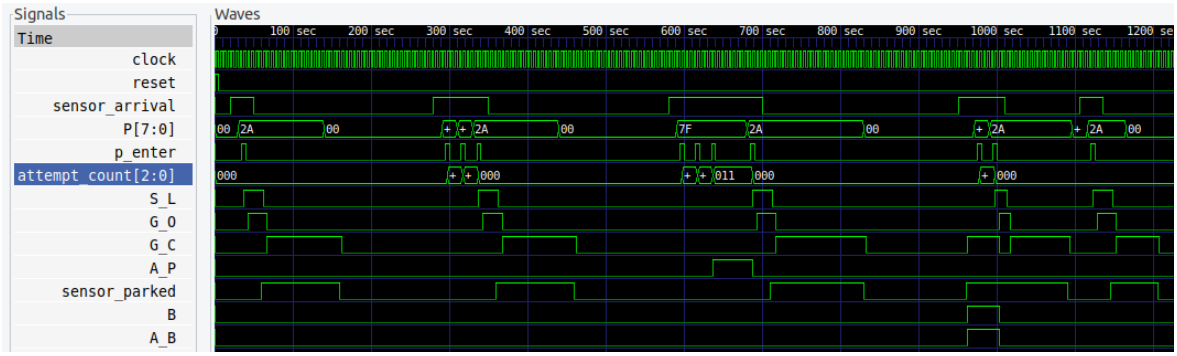

Figura 4: Simulación conductual obtenida con el archivo pruebas.v.

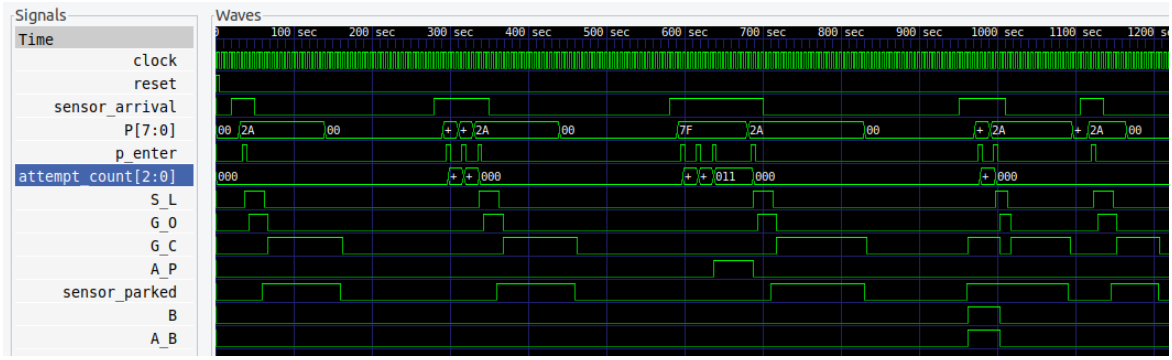

Figura 5: Simulación RTLIL obtenida con el archivo both_RTLIL.v.

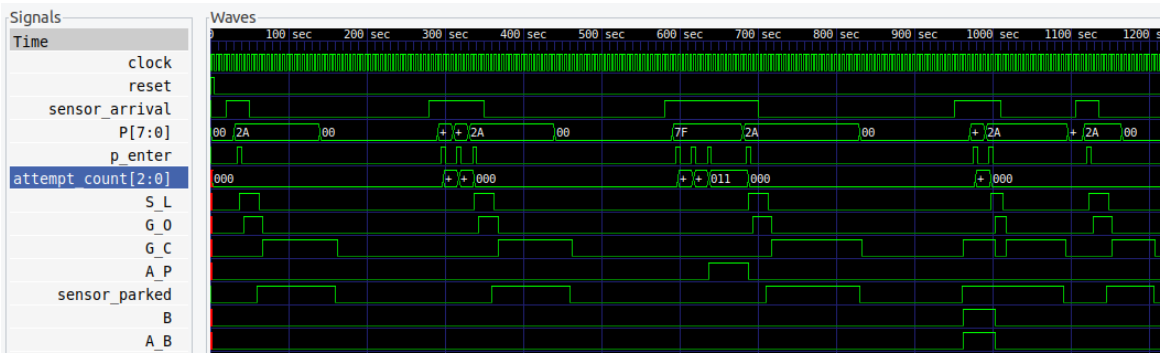

Figura 6: Simulación sintetizada obtenida con el archivo both_synth.v.

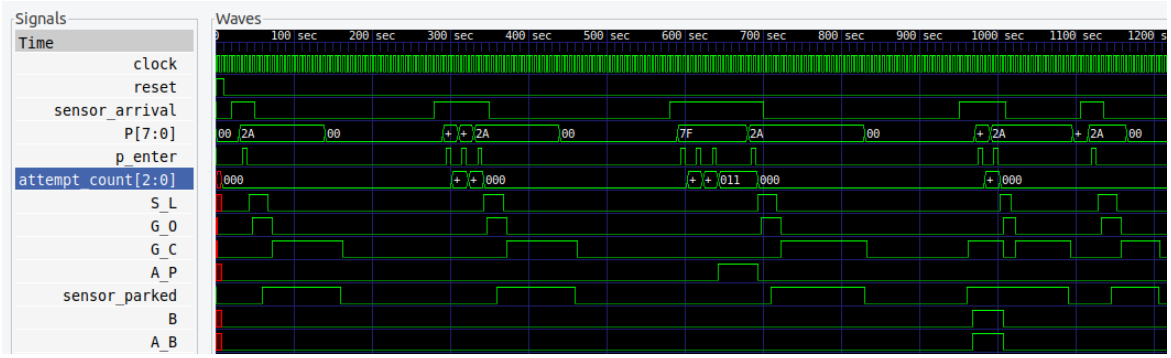

Figura 7: Simulación sintetizada con retardo obtenida con el archivo both_synth.v.

5.1. Comparación y análisis del retardo

Para el diseño del sistema, se estableció un retardo uniforme de 1 segundo para todas las compuertas lógicas. Es importante notar que el reloj utilizado tiene un periodo de 5 segundos, dividido en flancos positivos y negativos de 2.5 segundos cada uno. Este diseño permite observar claramente el impacto del retardo en la sincronización de los eventos respecto al reloj.

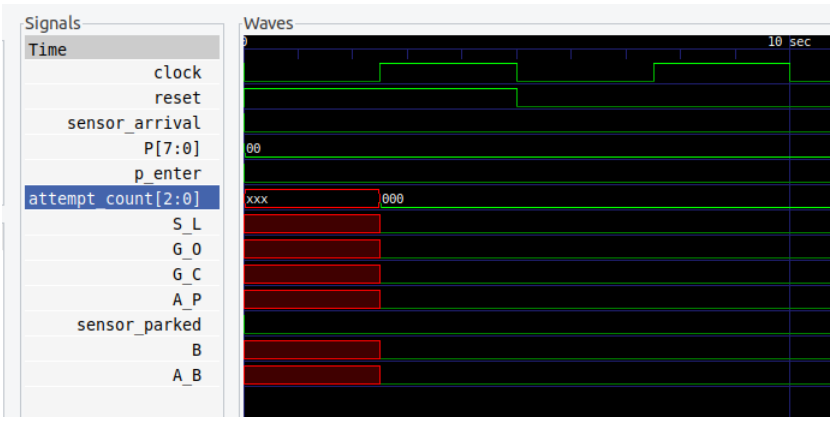

Figura 8: Aplicación de reset inicial, sin retardo.

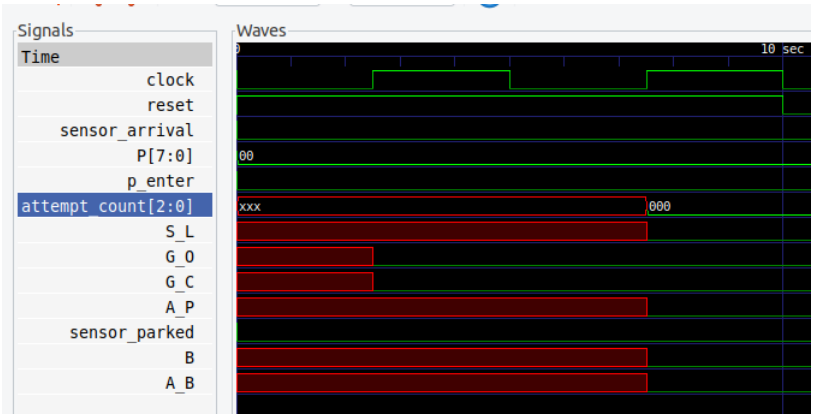

Figura 9: Aplicación de reset inicial, con retardo.

En las Figuras 8 y 9, se puede observar cómo la implementación del retardo afecta la actualización de los estados en el sistema. En la Figura 8, la respuesta a la señal de `reset` es inmediata, con los estados lógicos de las señales cambiando en el mismo ciclo de reloj en que el

`reset` es activado, lo que indica la ausencia de retardo adicional en la propagación de esta señal a través del sistema. Por otro lado, la Figura 9 muestra un contraste, donde tras la activación del `reset`, las señales como `attempt_count` retienen su estado previo hasta el siguiente flanco positivo del reloj. Esto demuestra que el sistema mantiene su estado anterior durante un ciclo completo del reloj antes de proceder con la actualización, reflejando el impacto del retardo implementado.

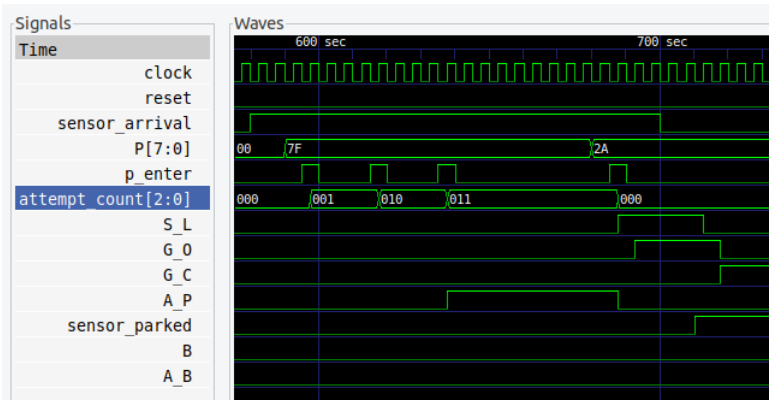

Figura 10: Actualización de intentos de contraseña, sin retardo.

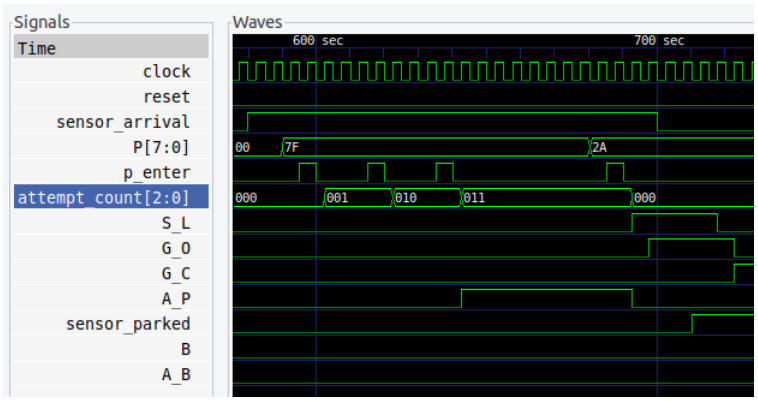

Figura 11: Actualización de intentos de contraseña, con retardo.

En las Figuras 10 y 11, se observa claramente el efecto del retardo en la actualización del contador de intentos de contraseña. Sin retardo, como se muestra en la Figura 10, la actualización del contador de intentos ocurre inmediatamente después de cada entrada, reflejando cambios instantáneos en respuesta a las acciones del usuario. Por contraste, en la Figura 11, el retardo implementado hace que el contador mantenga su valor previo hasta el próximo flanco positivo del reloj, retrasando así la actualización. Este comportamiento, aunque introduce una demora en la respuesta del sistema, es beneficioso ya que asegura la estabilidad del estado durante los flancos de reloj, evitando transiciones no deseadas que pueden ser causadas por ruidos o fluctuaciones momentáneas en las señales. Aunque el sistema responde con un ciclo de retraso, la integridad funcional se preserva.

6. Detalles, retos y complicaciones durante la solución

Lo que hace relevante a esta Tarea es que se puede comparar con la Tarea 1 para ver las diferencias en el código de los módulos para poder obtener una sintetización deseada.

El principal reto durante el desarrollo del diseño fue evitar la creación inadvertida de latches, que se producían al intentar sintetizar el código. Esta situación surgió por dos razones principales: la ausencia de un manejo adecuado del reloj en ciertas partes del código y la falta de cláusulas 'else' en algunas estructuras condicionales 'if', lo que propiciaba la formación de latches.

Un problema principal que se tuvo fue que al intentar correr la versión sintetizada, saltaba un error que decía que no se conocían los módulos NOT, NOR y NAND. Esto se produjo porque en el testbench no se realizó la inclusión del archivo 'cmos_cells.v'.

7. Evaluación de diseño obtenido

Tipo de Celda	**Cantidad**	**Retardo Aplicado (s)**
NAND	20	1
NOR	28	1
NOT	17	1
DFF	9	1

Tabla 1: Cantidad de celdas lógicas y retardo aplicado en el diseño.

El diseño muestra que, debido a los retardos implementados, las respuestas del sistema generalmente toman un ciclo más para activarse en comparación con una configuración sin retardos. Esta característica, aunque aumenta la latencia general del sistema, mejora la robustez del diseño al reducir los errores de sincronización y las transiciones lógicas erróneas durante los cambios de estado del reloj.

8. Conclusiones y recomendaciones.

La evaluación del retardo en el diseño del sistema digital demuestra su impacto en la sincronización de las transiciones de estado y la estabilidad general del sistema. La implementación de un retardo uniforme en las compuertas lógicas, si bien incrementa la latencia total del sistema, el sistema responde correctamente a lo que se esperaba. Se concluye que la gestión adecuada del retardo no solo mejora la fiabilidad del sistema, sino que también asegura la integridad operativa en entornos prácticos.

Las recomendaciones para futuros desarrollos o mejoras incluyen:

- Implementar un sistema de log que registre los intentos de acceso, tanto exitosos como fallidos, para un seguimiento detallado de la seguridad.
- Considerar la incorporación de un temporizador que limite el tiempo de respuesta tras la activación de `sensor_arrival`, para mejorar la eficiencia del sistema.
- Evaluar la integración de un mecanismo de respaldo que permita el acceso en casos de emergencia, aun cuando el sistema principal falle.
- Realizar análisis adicionales para explorar el ajuste óptimo de retardos en función de los requisitos específicos de cada módulo del sistema, con el objetivo de optimizar tanto el rendimiento como la seguridad.