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

Tarea #5 Diseño de un generador y un receptor de transacciones I^2C

Oscar Porras Silesky C16042
Grupo 001

Profesor: Ing. Enrique Coen

17 de junio de 2023

Índice

1. Resumen

Este proyecto aborda el diseño e implementación de un sistema de comunicación I2C (Inter-Integrated Circuit) utilizando Verilog. El sistema consta de módulos maestro y esclavo que interactúan para realizar transacciones de datos de manera coordinada y segura.

El rendimiento del sistema I2C se demostró eficazmente en la gestión de las señales y en la coordinación de las transacciones de datos entre el maestro y el esclavo. Se implementaron mecanismos de manejo de errores y sincronización de reloj para asegurar la integridad y la confiabilidad de la transmisión de datos.

Se llevaron a cabo pruebas para verificar la funcionalidad del sistema bajo varios escenarios, enfocándose en la correcta inicialización, transmisión de datos y respuesta a errores en tiempo de ejecución. Estas pruebas validaron que el diseño del sistema I2C es robusto y capaz de manejar distintas configuraciones de comunicación y errores de transmisión.

Una característica destacada de este diseño es la capacidad para manejar las condiciones de inicio y parada del protocolo I2C, así como la correcta arbitraje del bus y la detección de condiciones de reconocimiento (ACK/NACK). La eficiencia del sistema se mejoró significativamente mediante la optimización del control de reloj y el manejo eficiente de las líneas SDA y SCL, asegurando así la sincronización perfecta entre los dispositivos.

Este documento detalla cada parte del diseño, la funcionalidad del testbench y la integración de los módulos maestro y esclavo, ilustrando cómo interactúan dentro del entorno de prueba I2C.

2. Descripción Arquitectónica

2.1. Diagrama de Bloques del controlador I2C

El generador de transacciones I^2C es un módulo de hardware digital que se comunica con una CPU y un receptor de transacciones I^2C. Este módulo se conecta a la CPU mediante diferentes interfaces, incluyendo señales de control como CLK, RESET, START_STB, RNW, I2C_ADDR, WR_DATA y RD_DATA, del mismo modo las líneas de bus SDA_OE, SCL, SDA_OUT y SDA_IN.

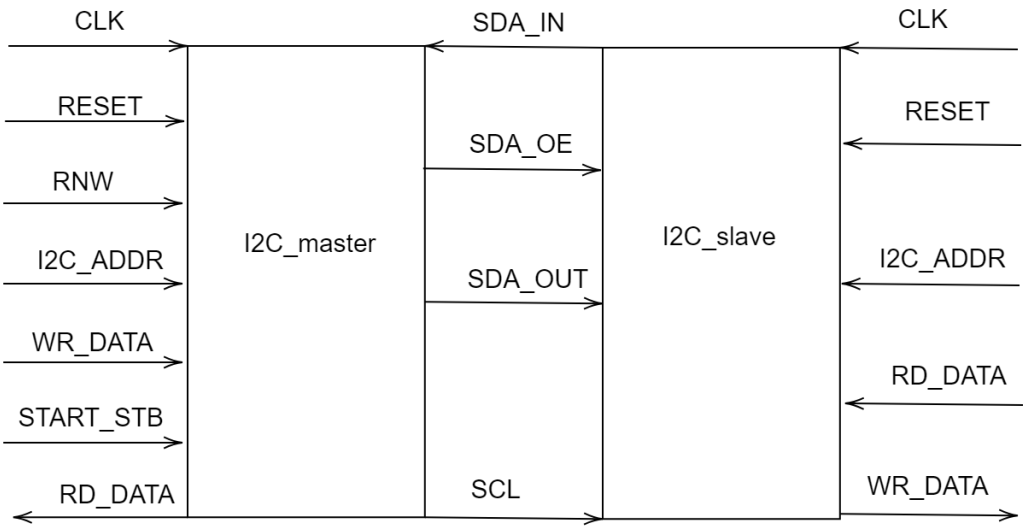

Figura 1: Diagrama de bloques del controlador I2C.

En la figura del controlador I^2C, se muestran las interacciones entre las señales del sistema

y el módulo `I2C_master` y `I2C_slave`. Las señales y su propósito son los siguientes.

- `CLK`:
  - En el caso del master: Es una entrada que llega al generador de transacciones desde el CPU con una frecuencia determinada. El flanco activo de `CLK` es el flanco creciente.
  - En el caso del slave: Entrada que llega al receptor de transacciones desde el CPU con una frecuencia determinada. El flanco activo de `CLK` es el flanco creciente.
- `RESET`:
  - En el caso del master: Entrada de reinicio del generador. Si `RESET` es igual a 1, el generador funciona normalmente. En caso contrario, el generador vuelve a su estado inicial y todas las salidas toman el valor de cero.
  - En el caso del slave: Entrada de reinicio del generador. Si `RESET` es igual a 1, el generador funciona normalmente. En caso contrario, el generador vuelve a su estado inicial y todas las salidas toman el valor de cero.
- `START_STB`: En el caso del master: Strobe (pulso de un ciclo de reloj). Indica al generador que el CPU desea iniciar una transacción de I2C.
- `RNW`: En el caso del master: Read not write. Esta señal indica la dirección de la transacción que desea ejecutar el CPU. Cuando `RNW` es igual a 1, se trata de una transacción de lectura, y cuando `RNW` es igual a 0, se trata de una escritura.
- `I2C_ADDR[6:0]`:
  - En el caso del master: Entrada proveniente de un registro del CPU que contiene la dirección del receptor de transacciones con el que el generador se quiere comunicar.
  - En el caso del slave: Entrada proveniente de un registro del CPU que contiene la dirección del receptor de transacciones. Cuando se recibe una transacción, debe cerciorarse de que es una transacción destinada a sí mismo, comparando el valor recibido en `SDA_IN` con el valor almacenado en este registro.
- `SCL`:
  - En el caso del master: Salida de reloj para el I2C. El flanco activo de la señal `SCL` es el flanco creciente. `SCL` es una salida del generador que debe tener una frecuencia del 25 % de la frecuencia de la entrada `CLK`. El generador debe generar `SCL` con la frecuencia correcta para cualquier posible valor de la frecuencia de entrada `CLK`.
  - En el caso del slave: Entrada de reloj para el I2C. El flanco activo de la señal `SCL` es el flanco creciente. `SCL` es una entrada del receptor que debe provenir del generador de transacciones o de un probador que emule su comportamiento.
- `SDA_OUT`:
  - En el caso del master: Salida serial. Debe tener el comportamiento especificado por el protocolo I2C para las condiciones de `START`, `STOP` y transacciones de escritura y lectura.
  - En el caso del slave: Entrada serial. La dirección de esta señal es inversa a la del generador de transacciones. Debe tener el comportamiento especificado por el protocolo I2C para las condiciones de `START`, `STOP` y transacciones de escritura y lectura.

- `SDA_OE`:
  - En el caso del master: Habilitación de `SDA_OUT`. Debe ponerse en 1 para aquellas porciones de la transacción donde el generador tiene control del bus I2C y ponerse en 0 para las porciones de la transacción donde el receptor tiene control del bus, de acuerdo con las especificaciones del protocolo.
  - En el caso del slave: Habilitación de `SDA_OUT`. Entrada serial. La dirección de esta señal es inversa a la del generador de transacciones. Debe ponerse en 1 para aquellas porciones de la transacción donde el generador tiene control del bus I2C y ponerse en 0 para las porciones de la transacción donde el receptor tiene control del bus, de acuerdo con las especificaciones del protocolo.
- `SDA_IN`:
  - En el caso del master: Entrada serie proveniente del probador. Debe tener el comportamiento especificado por el protocolo I2C para producir la indicación de `ACK`, así como proporcionar los datos de entrada en transacciones de lectura.
  - En el caso del slave: Salida serial enviada desde el receptor hacia el generador de transacciones. Debe tener el comportamiento especificado por el protocolo I2C para producir la indicación de `ACK`, así como proporcionar los datos de entrada en transacciones de lectura.
- `WR_DATA[15:0]`:
  - En el caso del master: Entrada paralela. Contiene los 16 bits que deben enviarse por la interfaz I2C durante una transacción de escritura de acuerdo a este enunciado.
  - En el caso del slave: Salida paralela. La dirección de esta señal es inversa a la del generador de transacciones. Contiene los 16 bits que se reciben por la interfaz I2C durante una transacción de escritura de acuerdo a este enunciado.
- `RD_DATA[15:0]`:
  - En el caso del master: Salida que debe producir los 16 bits que se reciben desde el receptor de transacciones I2C durante una transacción de lectura recibida en `SDA_IN`. Una vez que se han recibido los 16 bits completos, el generador de transacciones debe enviar la condición de `STOP` de acuerdo al protocolo I2C.
  - En el caso del slave: La dirección de esta señal es inversa a la del generador de transacciones. Esta entrada debe producir los 16 bits que se envían desde el receptor de transacciones I2C durante una transacción de lectura recibida en `SDA_OUT`. Los bits de `RD_DATA` se deben enviar a través de `SDA_IN` de acuerdo con el protocolo.

2.2. Funcionamiento de los Archivos

En el contexto del sistema SPI, cada archivo tiene un propósito específico dentro del proceso de simulación y verificación:

2.3. Funcionamiento de los Archivos

En el contexto del sistema I2C, cada archivo tiene un propósito específico dentro del proceso de simulación y verificación:

- `Testbench.v`: Este archivo actúa como el testbench principal que genera la simulación.

- `I2C_master.v`: Contiene la implementación del módulo maestro del I2C. Se encarga de iniciar las transacciones, generar el reloj I2C y manejar la comunicación mediante las señales SDA y SCL.
- `I2C_slave.v`: Contiene la implementación del módulo esclavo del I2C. Responde a las señales del maestro, recibiendo datos a través de SDA y enviando respuestas por la misma línea según el protocolo I2C.
- `Tester.v`: Proporciona el entorno de test y genera las señales de entrada para el módulo maestro y el módulo esclavo.

2.4. Módulo master_i2c

El módulo `master_i2c` implementa la lógica del maestro en un bus I2C. A continuación, se describen brevemente los estados del módulo:

- **Start_IDLE**: Estado inicial del maestro. Espera a que la señal `START_STB` se active para comenzar una transacción. En este estado, el maestro prepara las líneas `SDA_OUT` y `SDA_OE` para iniciar la comunicación.
- **Confirm_address**: En este estado, el maestro envía la dirección del esclavo al bus I2C. Se asegura de que el esclavo correcto responda antes de proceder con la transacción de lectura o escritura, dependiendo de la señal `RNW`.
- **Waiting_received_writing**: El maestro espera la confirmación del esclavo de que está listo para recibir datos. Si la confirmación es recibida (`SDA_IN` en bajo), el maestro pasa al estado de `Escritura`.
- **Waiting_received_reading**: Similar al estado anterior, pero para operaciones de lectura. El maestro espera la confirmación del esclavo antes de comenzar a leer datos.
- **WRITE**: En este estado, el maestro envía los datos al esclavo. La comunicación continúa hasta que todos los bits han sido transmitidos. Si el esclavo reconoce la recepción de los datos, el maestro vuelve al estado `Start_IDLE`.
- **READ**: El maestro recibe datos del esclavo. Los datos recibidos se almacenan en `Rd_data_master`. Una vez que todos los bits han sido recibidos, el maestro envía la condición de `STOP` y regresa al estado `Start_IDLE`.

2.5. Módulo slave_i2c

El módulo `slave_i2c` implementa la lógica del esclavo en un bus I2C. A continuación se describen brevemente los estados del módulo:

- **START**: Estado inicial del esclavo. Espera la condición de inicio (SDA_OUT en bajo mientras SCL está en alto) para comenzar a recibir la dirección del maestro.
- **COMPARING_ADDRES**: En este estado, el esclavo compara la dirección recibida con su propia dirección. Si coinciden, el esclavo determina si la transacción es de lectura o escritura y pasa al estado correspondiente.
- **WRITE**: En este estado, el esclavo recibe los datos del maestro. Los datos recibidos se almacenan en `Wr_data_slave`. Una vez completada la recepción de todos los bits, el esclavo vuelve al estado **START**.

- **READ**: En este estado, el esclavo envía datos al maestro. Los datos se envían desde `Rd_data_slave` bit a bit. Una vez completada la transmisión de todos los bits, el esclavo vuelve al estado **START**.

3. Plan de Pruebas

El sistema realiza operaciones de lectura y escritura de datos. Se ejecutarán pruebas de transacciones de dos bytes, tanto de lectura como de escritura.

El diseño se comunica con un dispositivo receptor cuya dirección se define con los dos últimos dígitos del número de carné universitario, en este caso C16042. La dirección es 42 en decimal, equivalente a 7'b0101010 en binario.

Las pruebas asegurarán el correcto envío y recepción de datos, así como el manejo adecuado de condiciones de error, como operaciones en direcciones incorrectas.

A continuación se enumeran las pruebas a realizar:

1. **Prueba 1: Test de Lectura Completa**
   - **Nombre/Número de prueba**: Lectura Completa
   - **Descripción de la prueba**: Verificación de la operación de lectura de dos bytes de datos desde la dirección del esclavo (7'b0101010).
   - **Resultado esperado**: Recepción correcta de los dos bytes de datos.
   - **Pasa/Falla**: Pasó
2. **Prueba 2: Test de Escritura Completa**
   - **Nombre/Número de prueba**: Escritura Completa
   - **Descripción de la prueba**: Verificación de la operación de escritura de dos bytes de datos a la dirección del esclavo (7'b0101010).
   - **Resultado esperado**: Escritura correcta de los dos bytes de datos en la dirección del esclavo.
   - **Pasa/Falla**: Pasó
3. **Prueba 3: Test de Dirección Incorrecta**
   - **Nombre/Número de prueba**: Dirección Incorrecta
   - **Descripción de la prueba**: Confirmación de que no se completa una escritura cuando la dirección del esclavo es incorrecta (7'b1111111).
   - **Resultado esperado**: No se permite la escritura debido a la dirección incorrecta.
   - **Pasa/Falla**: Pasó

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 el archivo `gtkwaveconfig.gtkw`.

4.1.1. Para solamente compilar:

Se debe correr el comando `make compile`

4.1.2. Para solamente ejecutar la simulación:

Se debe correr el comando `make run`

4.1.3. Para solamente abrir el GTKWave:

Se debe correr el comando `make gtkwave`

4.1.4. Para compilar, simular y abrir el GTKWave:

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.

4.1.5. 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 y análisis

En los ejemplos se ilustran tres partes de la simulación con resultados distintos.

En la primera parte de la simulación, se muestra una transacción de escritura que comienza con SDA bajando mientras SCL permanece alta, indicando la condición de inicio. Luego se envía la dirección junto con el bit que indica si es una operación de lectura o escritura. El acuse de recibo se manifiesta cuando SDA baja mientras SDA_oe se mantiene en bajo, marcando el comienzo de la transacción. Durante esta fase, se transmiten 8 bits y se recibe la confirmación del esclavo con SDA_in en bajo y SDA_oe aún en bajo. Posteriormente, se transmiten otros 8 bits, concluyendo así la transacción, tal como se muestra en la Figura 2.

En la segunda parte de la simulación, se muestra una transacción de lectura que inicia de manera similar, con la condición de inicio. El proceso sigue con la verificación de la dirección, donde tras recibir la confirmación (SDA_in en bajo), empieza la transacción dicha. Se leen 8 bits enviados por el esclavo al maestro, y el acuse de recibo se indica con SDA_out en bajo y SDA_oe en alto. Luego se procede con la transmisión de un segundo grupo de 8 bits, y con SDA_out en bajo junto con SDA_oe en alto, se finaliza la transacción, como se muestra en la Figura 3.

En la tercera parte de la simulación, se muestra el comportamiento cuando la dirección es incorrecta. En este caso, el esclavo regresa a su estado inicial a la espera de una nueva condición de inicio, mientras el maestro aguarda una respuesta de algún esclavo. Ante la ausencia de respuesta, después de un intervalo de tiempo, el maestro retorna al estado inicial sin haber completado ninguna transacción, lo cual se refleja en la Figura 4.

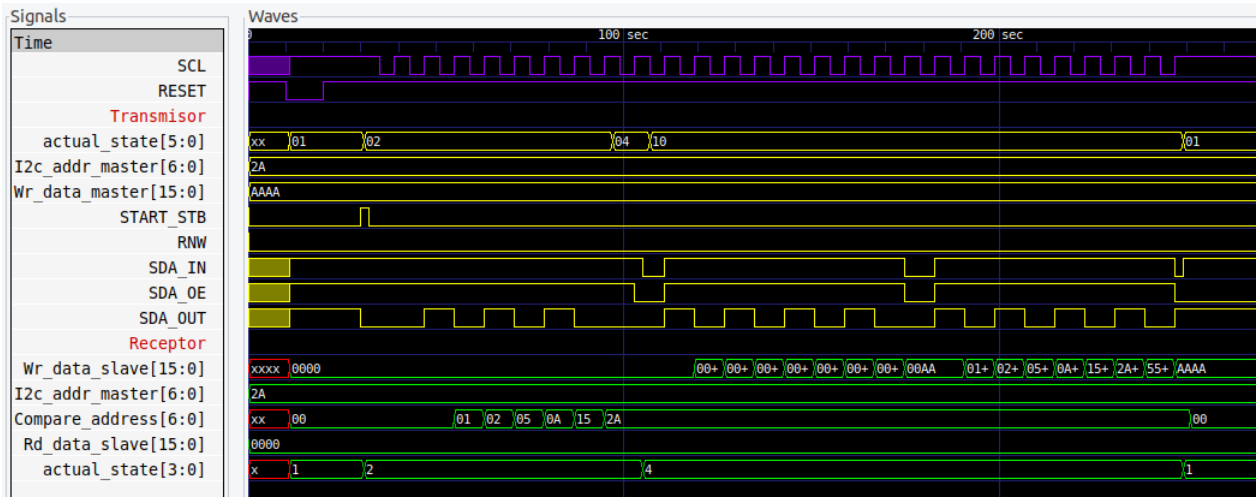

Figura 2: Transacción de escritura.

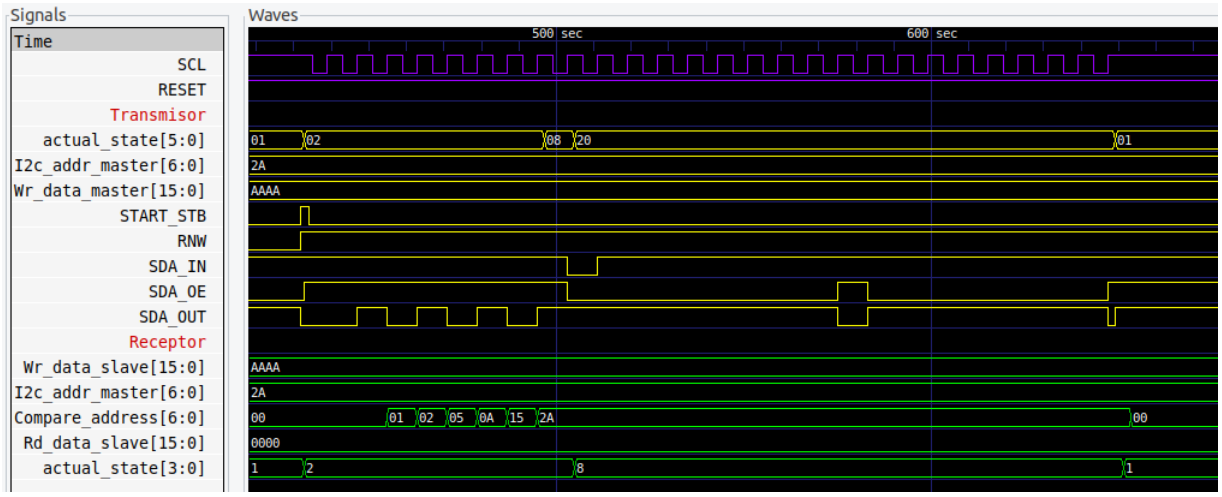

Figura 3: Transacción de lectura.

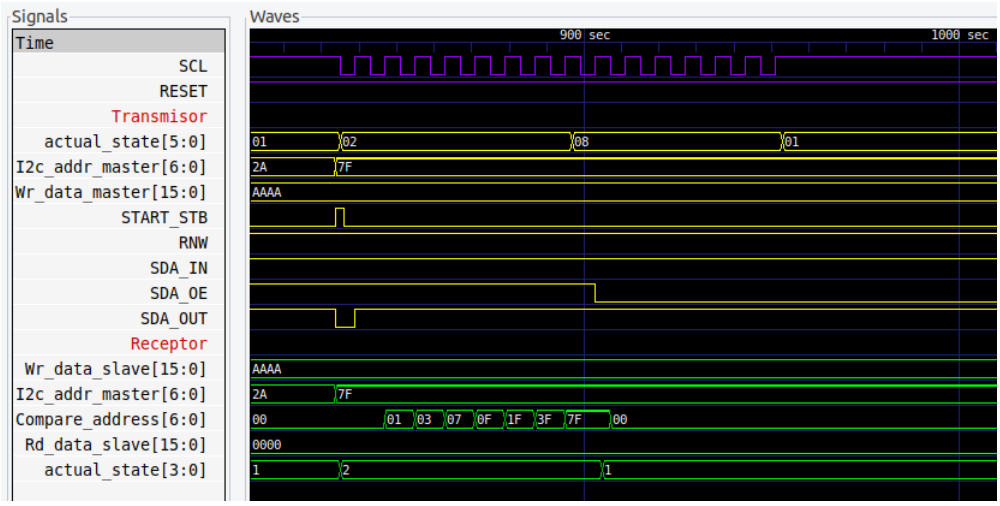

Figura 4: Condición de dirección incorrecta.

6. Detalles, retos y complicaciones durante la solución

El principal reto durante el desarrollo del diseño fue el control de la señal SCL para que funcionara conforme al protocolo I2C. Se presentaron problemas para sincronizar correctamente el reloj, lo que llevó a crear una señal intermedia para manejar adecuadamente la sincronización. Otro reto fue que los bits registrados en la señal `Wr_data_slave` se atrasaban por uno después de activarse la señal `SDA_OUT` por primera vez. Esto se solucionó modificando la lógica de estados, agregando un `next_state` para evitar atrasos y permitir que las variables `Prox_wr_data_slave` y `Wr_data_slave` se actualizaran correctamente.

7. Conclusiones y Recomendaciones

En conclusión, el diseño implementado del módulo I2C Master y Slave ha demostrado su funcionalidad y eficacia bajo todas las condiciones de operación probadas. La capacidad del sistema para manejar diferentes transacciones de lectura y escritura sin errores valida su robustez y adaptabilidad. Además, la prueba en condiciones variadas confirmó la integridad de los datos, resaltando la fiabilidad del diseño.

Las recomendaciones para futuros desarrollos o mejoras son las siguientes:

- **Planificación Detallada:** Implementar una fase de planificación inicial detallada antes de proceder con la estructuración del código. Esto debería incluir la definición de requisitos, la selección de metodologías de diseño y el establecimiento de un marco para pruebas preliminares.
- **Pruebas de Síntesis:** Realizar pruebas de síntesis para confirmar la viabilidad de construcción física del circuito propuesto. Esta aproximación proporciona una verificación temprana de que el diseño es no solo funcional en teoría, sino también práctico y realizable, además de evaluar el comportamiento del circuito con retardos para probarlo en un entorno más realista.