Trabajo Práctico Final
Sistemas Digitales 1

Diseño de un receptor y transmisor de comunicación I2C en un FPGA

Matteo Martinez, Franco de Vargas

Facultad de Ingeniería UNA, Ing. Mecatrónica

Palabras clave

I2C, VHDL, FPGA, Spartan 6, DS3231, Master, Slave.

Resumen

Resumen : En este proyecto se tratara de diseñar un receptor y transmisor de comunicación I2C en VHDL, sintetizado con Xilinx ISE Design Suite 14.7, con su implementacion en un FPGA XC6SLX9.

1. Introducción

En el mundo de la comunicación de datos en serie A.P.Godse (2008), existen protocolos como RS-232, RS-422, RS-485, SPI (interfaz de periféricos en serie), Microwire para interconectar periféricos de alta y baja velocidad. Estos protocolos requieren más conexiones de pines en el IC (circuito integrado) para que se realice la comunicación de datos en serie, ya que el tamaño físico del IC ha disminuido con los años, requerimos menos cantidad de conexiones de pines para la transferencia de datos en serie. USB/SPI/Microwire y principalmente UARTS son simplemente sistemas de bus de transferencia de datos de “un punto a un punto”. Utilizan multiplexación de la ruta de datos y reenvío de mensajes para dar servicio a múltiples dispositivos. Para superar este problema, Phillips introdujo el protocolo I 2C Wilmshurst (2007), que requiere solo dos líneas para la comunicación con dos o más chips y puede controlar una red de chips de dispositivos con solo dos pines de E/S de propósito general, mientras que otros buses Los protocolos requieren más pines y señales para conectar dispositivos.

En este proyecto, estamos implementando el protocolo de bus I2C para interconectar dispositivos periféricos de baja velocidad en FPGA Singh (2012). También es el mejor bus para las aplicaciones de control, donde es posible que sea necesario agregar o quitar dispositivos del sistema. El protocolo I2C también se puede utilizar para la comunicación entre múltiples placas de circuito en equipos con o sin cable blindado, dependiendo de la distancia y la velocidad de transferencia de datos. El bus I2C es un medio de comunicación donde el controlador maestro Himpe (2011) se utiliza para enviar y recibir datos hacia y desde el esclavo DS3231. El periférico de baja velocidad, DS3231, está interconectado con el bus maestro I2C y sintetizado en Spartan 6. La Figura 1 muestra el sistema de bus I2C con el controlador maestro I2C implementado en una FPGA y el dispositivo de reloj en tiempo real actuando como esclavo. La sinopsis del artículo es la siguiente: En la sección 2, analizamos el protocolo I2C de nuestro diseño propuesto, que también presenta la descripción del módulo para nuestro sistema propuesto. En la sección 3, presentamos la implementación del software junto con el algoritmo y el diagrama de flujo. En la sección 4, se encuentra la descripción detallada de la implementación de hardware del bus maestro I2C. controller in Spartan 6 FPGA Design kit using Xilinx software. Finalmente en la seccioon 5 encontramos la conclusion.

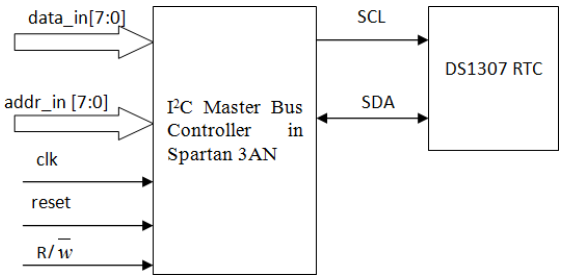

Fig. 1: Diagrama de I2C Master Controller conectado a DS3231 RTC dispositivo Slave

2. Trabajo Propuesto

2.1. *Protocolo I2C*

I2C es un bus serie bidireccional de dos cables que proporciona una comunicación de datos efectiva entre dos dispositivos. El bus I2C admite muchos dispositivos y cada dispositivo se reconoce por su dirección única. En la figura 1 data in y addr in son la dirección de 8 bits proporcionada como entrada. Clk y reset son las líneas de entrada utilizadas para iniciar el proceso del controlador de bus. La señal R/w se proporciona como entrada para indicar si el maestro o el esclavo actúa como transmisor en la transmisión de datos. El bus físico I2C consta de sólo dos cables, llamados SCL y SDA. SCL es la línea del reloj. Se utiliza para sincronizar todas las transferencias de datos a través del bus I2C. SDA es la línea de datos. Las líneas SCL y SDA están conectadas a todos

los dispositivos en el bus I2C. Como las líneas SCL y SDA son controladores de ”drenaje abierto”, se activan mediante resistencias de activación 1 (2000). Se dice que el bus I2C está inactivo cuando tanto SCL como SDA están en el nivel lógico 1. Cuando el maestro (controlador) desea transmitir datos a un esclavo (DS3231), comienza emitiendo una secuencia de inicio en el bus I2C, que es una transición de alto a bajo en la línea SDA mientras la línea SCL está en alto, como se muestra en la Fig 2. Se considera que el autobús está ocupado después de la condición de START.

Fig. 2. Secuencia START y STOP

Después de la condición de START, la dirección del esclavo es enviada por el maestro. El dispositivo esclavo cuya dirección coincida con la dirección enviada por el maestro responderá con un bit de reconocimiento en la línea SDA bajando la línea SDA. Los datos se transfieren en secuencias de 8 bits. Los bits se colocan en la línea SDA comenzando con el MSB (Bit más significativo). Por cada 8 bits transferidos, el dispositivo esclavo que recibe los datos envía un bit de reconocimiento, por lo que en realidad hay 9 pulsos de reloj SCL para transferir cada byte de datos de 8 bits, como se muestra en la Figura 3. Si el dispositivo receptor devuelve un bit ACK bajo, entonces ha recibido los datos y está listo para aceptar otro byte. Si devuelve un nivel alto, indica que no puede aceptar más datos y el maestro debe finalizar la transferencia enviando una secuencia de PARADA. En la Figura 2, que muestra la secuencia de STOP, donde la línea SDA está baja mientras que la línea SCL está alta. Esto señala el final de la transacción con el dispositivo esclavo.

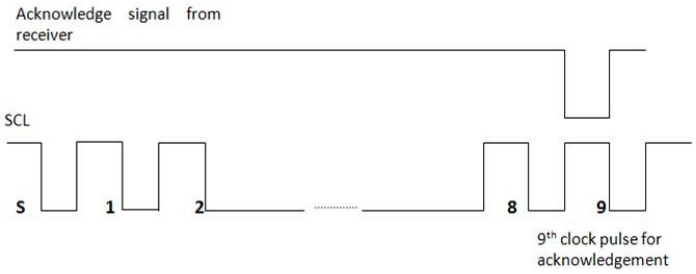

Fig. 3. Señal ACK del receptor

2.2. *Serial data communication*

El bus I2C tiene dos modos de funcionamiento: transmisor maestro y receptor maestro. El bus maestro I2C inicia la transferencia de datos y puede controlar líneas SDA y SCL.kamal (2008) El dispositivo esclavo (DS3231) es direccionado por el maestro. Sólo puede emitir datos en la línea SDA. En el modo de transmisión maestro, después del inicio de la secuencia de INICIO, el maestro envía una dirección de esclavo. El byte de dirección contiene la dirección DS3231 de 7 bits, que es 1101000, seguida de el bit de dirección (R/w). Después de recibir y decodificar el byte de dirección, el dispositivo genera reconocer en la línea SDA. Después de que el DS3231 reconoce la dirección del esclavo + bit de escritura, el maestro transmite una dirección de registro al DS3231, esto configurará el puntero de registro en el DS3231. Luego, el maestro comenzará a transmitir cada byte de datos y el DS3231 acusará recibo de cada byte recibido. El maestro generará una condición de parada para finalizar la escritura de datos. En el modo de receptor maestro, el primer byte se recibe y se maneja como en el modo de transmisión maestro. Sin embargo, en este modo, el bit de dirección indicará que la dirección de transferencia está invertida. Los datos en serie se transmiten en SDA mediante el DS3231 mientras que el reloj en serie se ingresa en SCL. Las condiciones de INICIO y PARADA se reconocen como el comienzo y el final de una transferencia en serie (Fig-5). El byte de dirección es el primer byte recibido después de que el maestro genera la condición de inicio. El byte de dirección contiene la dirección DS3231 de 7 bits, que es 1101000, seguido del bit de dirección (R/w). Después de recibir y decodificar el byte de dirección, las entradas del dispositivo confirman en la línea SDA. El DS3231 luego comienza a transmitir datos comenzando con la dirección de registro señalada por el puntero de registro. Si el puntero de registro no se escribe antes del inicio de un modo de lectura, la primera dirección que se lee es la última almacenada en el puntero de registro. El DS3231 debe recibir un ”no reconocido”para finalizar una lectura.

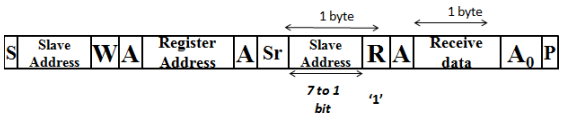

Fig. 4. Master en modo transmisión

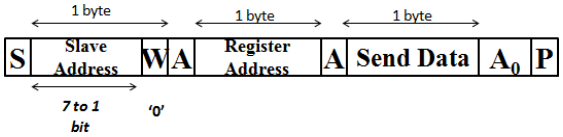

Fig. 5. Master en modo receptor

2.3. *DS3231*

El DS3231 admite un protocolo de transmisión de datos y bus bidireccional de 2 cables. La asignación de pines del DS3231 se muestra en la Fig-6. El DS3231 funciona como esclavo en el bus de 2 cables muestra la conexión de la interfaz del bus I2C en Spartan 6 con el chip DS3231 RTC.

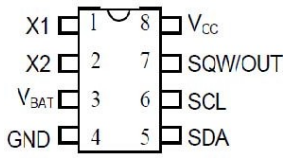

Fig. 6. Asignación de pines DS3231

3. Implementación del Software

El controlador maestro I2C está diseñado utilizando VHDL Sutherland (2001) basado en Finite State Machine (FSM) Vahid (2010). FSM es un circuito secuencial que utiliza un número finito de estados para realizar un seguimiento de su historial de operaciones y, según el historial de operación y la entrada

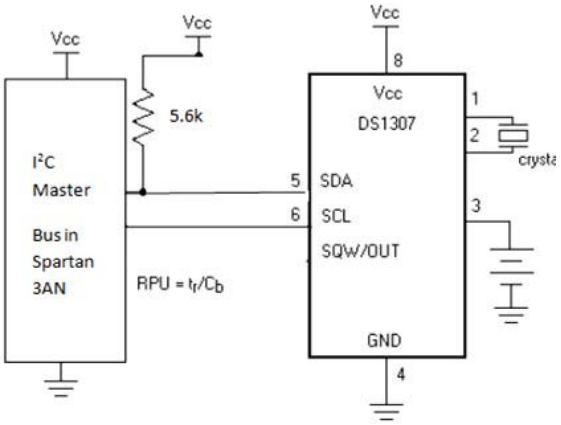

Fig. 7. DS3231 conectado bus serial

actual, determina el siguiente estado. Hay varios estados para obtener el resultado.

Hay dos capas en este diseño:

- El propio controlador de bajo nivel.
- La interfaz entre el controlador y el microprocesador, es decir. los registros presentados Ambos pueden considerarse de forma relativamente aislada, pero, por supuesto, existe una superposición.

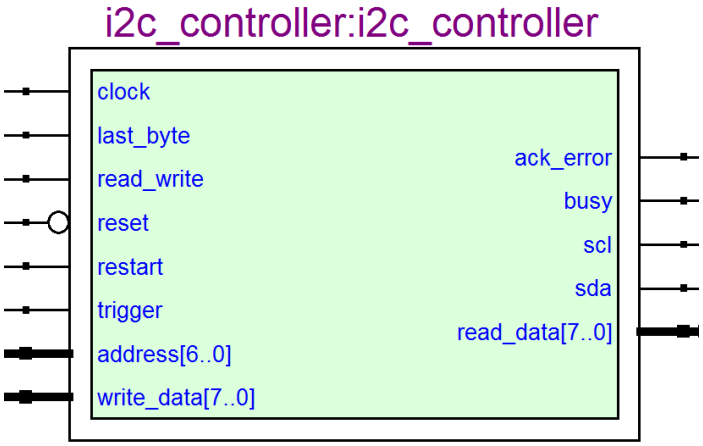

Fig. 8. Entidad del controlador I2C

El problema inicial fue generar un c. Reloj de 100 KHz para el flujo de datos en serie I2C. Hay un ángulo interesante en este reloj: no puede ser simplemente, digamos, el reloj del procesador dividido. La razón de esto es que los datos en serie deben entrar o salir tan pronto como el procesador lo solicite, y no en el siguiente límite del reloj de 100 KHz, ya que esto retrasará el procesador innecesariamente.

Para lograr esto, se crearon dos procesos sincrónicos, ambos sincronizados con el reloj del procesador. El primer proceso registra un contador, cuyo bit superior genera el reloj I2C de aproximadamente 100 KHz. Este contador comienza a contar solo después de un evento desencadenante (por ejemplo, cuando el procesador escribe en el registro de dirección esclavo), que está activo durante exactamente un ciclo de reloj del procesador.

La máquina de estado del controlador I2C real se encuentra en el segundo proceso. Como se mencionó, este proceso también se sincroniza con el reloj del procesador, pero solo en el flanco ascendente del bit superior del contador descrito anteriormente realiza alguna acción. De lo contrario no hace nada.

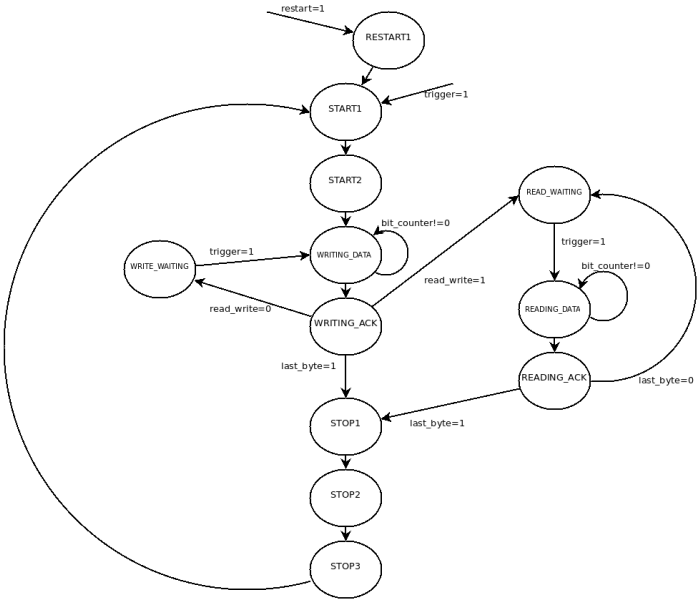

Fig. 9. Diagrama de Bloques del controlador I2C

Los fragmentos de estado se dividen de la siguiente manera:

- START1 y START2: se genera la forma de onda de inicio, es decir, el pin SDA baja mientras el SCL permanece alto.
- WRITING DATA: Los 8 bits de datos están sincronizados. Se utiliza el mismo estado para enviar la dirección esclava que para enviar datos útiles. Se requieren 16 relojes de entrada, ya que la salida SCL debe realizar un ciclo por cada bit escrito.
- WRITING ACK: el controlador bloquea el ACK o NAK del esclavo. Además, el controlador cambia al estado de espera de escritura o lectura según el modo solicitado. Si este es el último byte que debe escribirse (last byte = 1), el controlador ingresa al primer estado STOP.
- WRITE WAITING: aquí la máquina de estado simplemente se queda esperando el siguiente evento desencadenante, tras el cual bloqueará la entrada write data para el siguiente byte.
- READING DATA: Se registran 8 bits de datos. Se requieren 16 relojes de estado".
- READING ACK: El controlador genera un ACK (es decir, baja el SDA) si este no es el último byte que necesita lectura; de lo contrario, lo pone alto para un NAK. Si este es el último byte a leer, entonces el controlador ingresa al primer estado STOP; de lo contrario, pasa a READ WAITING.
- READ WAITING: el controlador espera otro disparador antes de ingresar READING DATA.
- STOP1 a STOP3: El generador de secuencia de parada. El pin SDA se eleva mientras que SCL permanece alto.

- RESTART1: Esto se usa para bajar los pines SDA y SCL en preparación para enviar una nueva secuencia de START, y algunos IC esclavos lo usan para eliminar el requisito de crear una nueva transacción pasando por una secuencia de parada/inicio.

La simulación se llevo a cabo en ISim integrado en Xilinx ISE Design Suite. El testbench inicia con la carga del slave address '1101000' y se procede a la carga un byte para escritura '0f' y luego se vuelve a cargar otro 'aa' sobre la linea write data, se observa la correcta carga en la linea sda y se procede a la lectura, se cambia a modo lectura read write y se manda el restart. Tenga en cuenta que el testbench no imita al dispositivo slave , por lo que read data no contiene datos después de las operaciones de lectura.

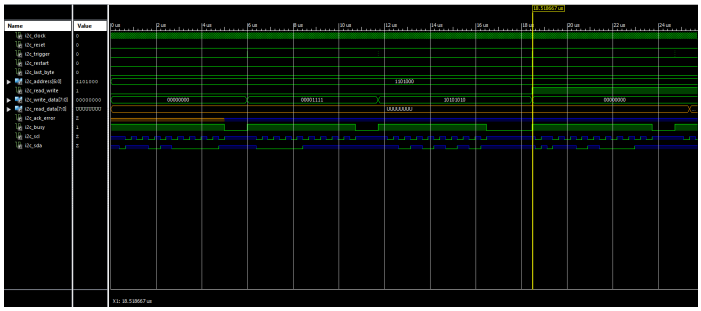

Fig. 10. Testbench en modo escritura

Se concluye el éxito del testbench y el correcto funcionamiento del controlador I2C.

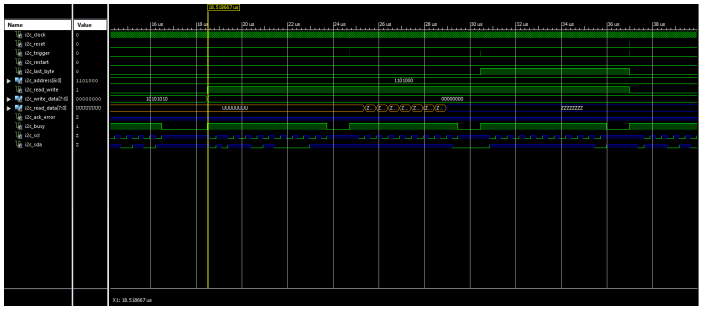

Fig. 11. Testbench en modo lectura

4. Implementación del Hardware

La implementación del hardware se realiza conectando el DS3231 con el maestro I2C presente en el kit Spartan 6. Es un DIP de 8 pines. El esclavo recibe una fuente de alimentación de 5 V. El pin 4 está conectado a tierra (GND) y el pin 8 está conectado al VCC. Los pines 5 y 6 están conectados a SDA y SCL del bus maestro I2C en Spartan 6. Tanto SDA como SCL son líneas abiertas con drenaje. Cuando se conectan varios maestros, es necesario conectar tanto SDA como SCL con una resistencia de 6.8 k ohm al VCC. Dado que sólo hay un maestro conectado, no es necesario activar SCL. El cable rojo indica VCC, el cable negro indica tierra (GND), el cable amarillo indica línea SCL y el cable verde indica línea SDA conectada con el kit Spartan 6 FPGA. Después de completar la codificación en VHDL, se descarga al kit Spartan 6 utilizando el software Xilinx. Y la entrada correspondiente se proporciona según el protocolo I2C.

Operación de Escritura

- Pulse reset, si esta es la primera transacción.
- Establezca last byte, restart y trigger a 0 y read write a 0, y el 7 bit slave address
- Pulse trigger
- Espere a busy hasta que vuelva a 0
- Chequear ack error y reaccionar
- Para cada byte a enviar que no es el ultimo:
  1. Pon el byte a enviar en write data
  2. Pulse trigger
  3. Espere que busy vuelva a 0
  4. Chequee ack error
- Para el byte final:
  1. Establezca last byte a 1
  2. Repita los pasos anteriores

Operación de Lectura

- Pulse reset, si esta es la primera transacción.
- Establezca last byte, restart y trigger a 0 y read write a 1, y el 7 bit slave address
- Pulse trigger
- Espere a busy hasta que vuelva a 0
- Chequear ack error y reaccionar
- Para cada byte a enviar que no es el ultimo:
  1. Pulse trigger
  2. Espere que busy vuelva a 0
  3. Latch read data
- Para el byte final:
  1. Establezca last byte a 1
  2. Repita los pasos anteriores

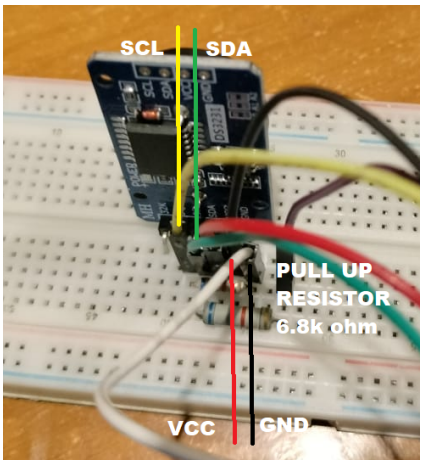

Fig. 12. Conexiones del Slave DS3231

En el montaje final del proyecto se puede observar en la Fig-12 y Fig-13, donde slave DS3231 esta conectado a las

lineas SDA y SCL del Master FPGA Spartan 6. Finalmente la distribución del pinado en el FPGA. El Header P6 se utiliza para el address de entrada, el P7 para el write data de entrada, el P8 para read data de salida, en el P9 los pines del 1 al 4 son las lineas SDA, SCL, busy y ACK, estas ultimas dos son salidas, los dip switches del 4 al 8 se utilizan para entradas son respectivamente write read, last byte, restart, trigger y reset.

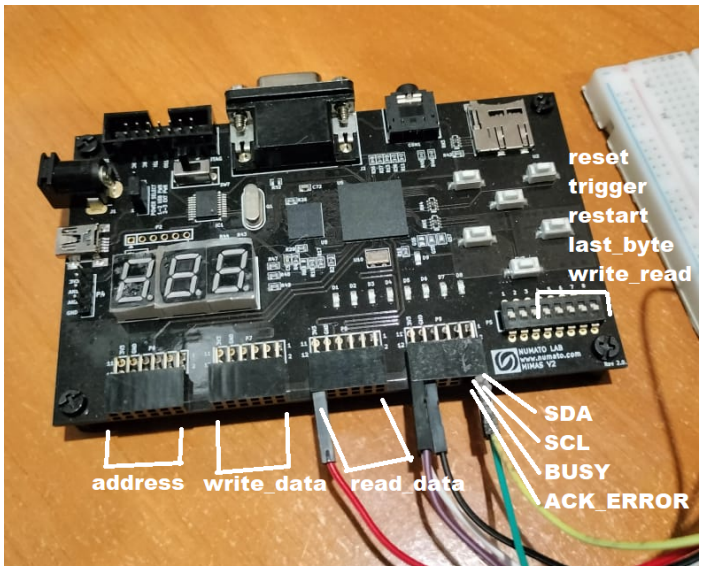

Fig. 13. Distribución de puertos en el FPGA Spartan 6

5. Conclusión

Este proyecto demuestra cómo el controlador maestro I2C (maestro) transmite y recibe datos hacia y desde el DS3231 (esclavo). Para que cualquier dispositivo periférico de baja velocidad pueda conectarse utilizando el protocolo de bus I2C como maestro. En el futuro, esto se podrá implementar como reloj en tiempo real en redes que contengan múltiples maestros y múltiples esclavos para coordinar todo el sistema mediante técnicas de sincronización de reloj.

6. Referencias

(2000). *Philips Semiconductor "I2C Bus Specification"*.
A.P.Godse (2008). *"Microprocessor, Microcontroller Applications"*.
Himpe, V. (2011). *"Mastering the I2C bus" Elektor Verlag publications*.
kamal, R. (2008). *"Embedded system: Architecture programming and Design"*.
Singh, P. J. K. (2012). *"Design and Implementation of I2c master controller on FPGA using VHDL,"*.
Sutherland, S. (2001). *"Verilog® HDL Quick ReferenceGuide"*.
Vahid, F. (2010). *"Digital Design with RTL Design, Verilog and VHDL"VP and Executive publisher*.
Wilmshurst, T. (2007). *"Designing Embedded Systems with PIC Microcontrollers: Principles and Applications"*.