Diseño de un Modulo de Propiedad Intelectual Basado en FPGA para el manejo del Bus I2C

Renato Manzano, Mónica Pérez , Ronald Ponguillo
Facultad de Ingeniería de Electricidad y Computación
Escuela Superior Politécnica del Litoral
Km 30.5 Vía Perimetral, 09015863, Guayaquil, Ecuador
rmanzano@fiec.espol.edu.ec, jperez@fiec.espol.edu.ec, rponguil@espol.edu.ec

Resumen

El presente trabajo es el desarrollo de un IPCORE I2C sencillo aplicado para el control y monitoreo de un dispositivo ESCLAVO RTC (Real Time Clock) y un dispositivo MAESTRO alojado en un FPGA de Altera.

Nuestra aplicación está desarrollada en dos tarjetas, una en la que está nuestro dispositivo ESCLAVO I2C (RTC) y la otra tarjeta será la DE2 de ALTERA en la que estará nuestro MAESTRO; conectadas a través de los puertos de entrada/salida de propósito general. En esta aplicación el usuario podrá setear todos los parámetros del reloj calendario como segundos, minutos, horas, día, fecha, mes y año a través de botoneras montadas en la tarjeta DE2 y visualizadas en un Display también ubicadas en la misma tarjeta.

Palabras Claves: *IPCORE, RTC, FPGA, DE2, ALTERA.*

Abstract

This work is the development of a simple I2C IPCORE applied to control and monitors a slave device RTC (Real Time Clock) and a master device inside of an Altera FPGA.

Our application is developed on two boards, connected through input / output general purpose pins. The first is an I2C slave device (RTC) and the other board is ALTERA DE2, where we have the I2C master. In this application the user can to set all parameters of the RTC as seconds, minutes, hours, day, date, month and year through keypads mounted on the DE2 board and also showed on a display located on the same card.

Keywords: *IPCORE, RTC, FPGA, DE2, ALTERA.*

1. Introducción

Existen varios protocolos para la comunicación entre dispositivos, siendo SPI e I2C los más conocidos, si nos referimos a comunicar dispositivos relativamente cercanos.

Las ventajas de SPI son varias frente a I2C, entre ellos la velocidad, a pesar de eso I2C lo encontramos en muchos dispositivos hoy en día, televisores, DVD, juegos de video, teléfonos móviles, agendas personales, etc. Una diferencia de I2C contra SPI es que solo trabaja con 2 hilos para la comunicación, esto puede ser una ventaja o desventaja dependiendo donde lo vayamos a aplicar; por ejemplo: si no estamos limitados de espacio físico para desarrollar nuestra tarjeta y/o necesitamos comunicación de gran velocidad, se recomendaría SPI, pero si estamos limitados en espacio físico, lo mejor en este sentido es I2C, este protocolo permite optimizar recursos entre todos los dispositivos que se quieran comunicar, ya que solo ocuparíamos 2 pines de cada dispositivo y el resto de pines podrimos utilizarlo para otro propósito.

En nuestra aplicación desarrollamos un IPCORE I2C sencillo, fácil de entender para una comprensión rápida del protocolo, y así desarrollar futuros proyectos ya sea de propósito general o de sistemas embebidos, donde podamos reducir al máximo el tamaño de nuestra tarjeta. Este proyecto se lo desarrollo con fines educativos por su relativa sencillez; mas no con fines comerciales.

2. Diseño

Nuestro diseño consta de 4 partes: IPCORE I2C MAESTRO, Módulo RTC ESCLAVO, Módulo LCD y Aplicación Ejemplo.

IPCORE I2C MAESTRO (CORE).- Es el encargado de generar las señales SCL y SDA para producir la comunicación mediante el Protocolo I2C.

MÓDULO RTC ESCLAVO.- Describe los pasos para llevar a cabo la comunicación con un Real Time Clock DS1307 (RTC) siguiendo las descripciones de la hoja de datos.

MÓDULO LCD.- Controla la escritura de los caracteres en una LCD.

La Aplicación Ejemplo (reloj - calendario).- Controla la lectura y escritura en el RTC, la escritura

en la LCD y el procesamiento de los datos de entrada y salida (conversiones y validaciones).

2.1. Diseño del IPCORE MAESTRO

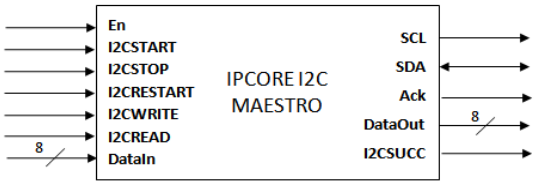

Figura 1. Ipcore I2C MAESTRO

Descripción de señales de entrada y salida del IPcore I2C MAESTRO

En.- Señal de entrada que habilita el CORE si es igual a '1' lógico o lo deshabilita si es igual a '0' lógico.

I2CSTART.- Señal de entrada que habilita el proceso que debe generarse en el comando de START para iniciar la comunicación I2C.

I2CSTOP.- Señal de entrada que habilita el proceso que debe generarse en el comando de STOP para finalizar la comunicación I2C.

I2CRESTART.- Señal de entrada que habilita el proceso que debe generarse en el comando de RESTART para indicar que se va a proceder a leer.

I2CWRITE.- Señal de entrada que habilita el proceso de escritura MAESTRO-ESCLAVO.

I2CREAD.- Señal de entrada que habilita el proceso de lectura MAESTRO-ESCLAVO.

Ack.- Señal de salida obtenida de la línea SDA luego de enviar un dato MAESTRO-ESCLAVO,

I2CSUCC.- Señal de salida, si I2CSUCC es igual a '1' lógico, indica finalización del proceso que ha sido habilitado por alguno de los comando de entradas y si I2CSUCC es igual a '0' lógico, indica que no se ha finalizado.

SCL.- Señal que genera un tren de pulsos de reloj.

SDA.- Señal que transporta de forma serial los datos desde el MAESTRO al ESCLAVO y viceversa.

Descripción de datos de entrada y salida del Ipcore I2C MAESTRO

DataIn.- Vector de 8 bits de entrada, posee el dato a ser escrito en el ESCLAVO.

DataOut.- Vector de 8 bits de salida, posee el dato leído desde el ESCLAVO.

En la Figura 2 observamos el Diagrama ASM del Ipcore I2C MAESTRO, dividido en 5 bloques uno por cada comando.

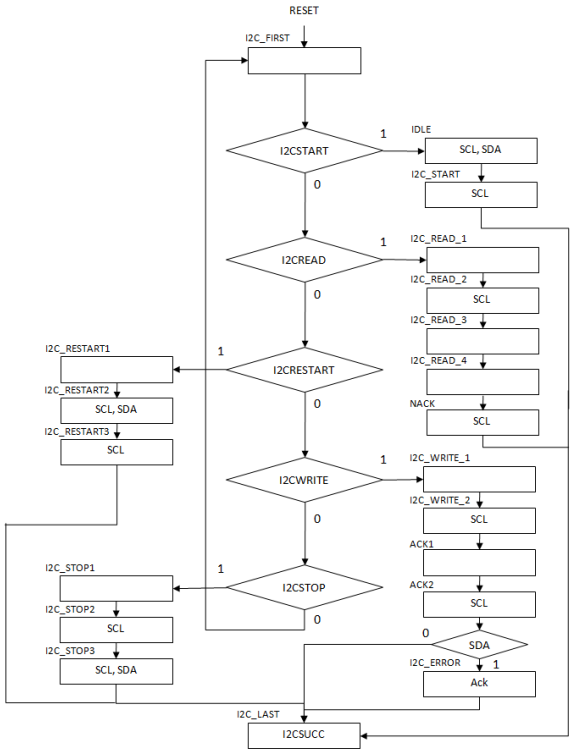

Figura 2. Diagrama ASM del Ipcore I2C MAESTRO

Activado el CORE se inicia en el estado I2C_FIRST que evalúa las señales de entrada para la ejecución de los diferentes procesos.

Enviar Condición START

La ejecución del comando START es necesaria para iniciar la comunicación.

Esto alertará a los dispositivos ESCLAVOS a la recepción y evaluación de la dirección de dispositivo enviada a continuación para corroborar si desean comunicarse con ellos.

Cuando se solicita el envió del comando START debe estar activa la señal I2CSTART.

El proceso que debe ejecutarse para generar la condición START es el descrito en los estados IDLE e I2C_START; se genera la transición de 'alto a bajo' en SDA mientras SCL permanece en 'alto'.

Enviar condición STOP

La ejecución del comando STOP es necesaria para finalizar la comunicación.

Cuando se solicita el envió del comando STOP debe estar activa la señal I2CSTOP.

El proceso que debe ejecutarse para generar la condición STOP es el descrito en los estados I2C_STOP_1, I2C_STOP_2 y I2C_STOP_3, en el estado I2C_STOP_1 e I2C_STOP_2; se genera la transición 'bajo a alto' en SDA mientras SCL permanece en 'alto'.

Enviar condición RESTART

Cuando se solicita el envió del comando RESTART debe estar activa la señal I2CRESTART.

El proceso que debe ejecutarse para generar la condición RESTART es el descrito en los estados I2C_RESTART1, I2C_RESTART2 y I2C_RESTART3, en el estado I2C_RESTART1 e I2C_RESTART_2; se genera la transición 'alto a bajo' en SDA mientras SCL permanece en 'alto'.

Escritura en ESCLAVO

Cuando se solicita la escritura MAESTRO-ESCLAVO debe estar activa la señal I2CWRITE.

El proceso que debe ejecutarse para realizar la escritura es el descrito en los estado I2C_WRITE_1, I2C_WRITE_2, ACK1 y ACK2, estos envían uno a uno los 8-bits del dato, empezando por el MSB y lee el ACK. Ver Figura 3.

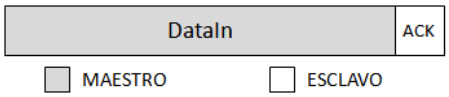

Figura 3. Transferencia de dato

Lectura de ESCLAVO

Cuando se solicita la lectura MAESTRO-ESCLAVO debe estar activa la señal I2CREAD.

El proceso que debe ejecutarse para realizar la lectura es el descrito en los estados I2C_READ_1, I2C_READ_2 e I2C_READ_3, estos receptan uno a uno los 8 bits del dato, empezando por el MSB y envía el NACK. Ver Figura 4.

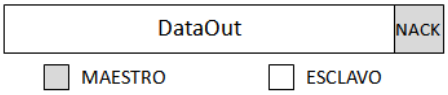

Figura 4. Recepción de dato

2.2. Diseño del módulo RTC ESCLAVO

Diagrama de Bloques del módulo RTC ESCLAVO

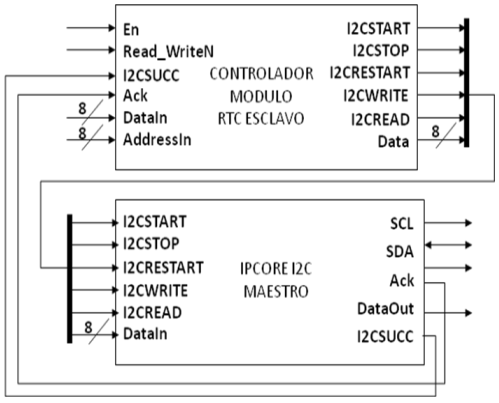

Figura 5. Diagrama de bloque del módulo RTC ESCLAVO

Descripción de señales de entrada y salida del circuito controlador del módulo RTC ESCLAVO

En.- Señal de entrada que habilita el módulo RTC ESCLAVO si es igual a '1' lógico o lo deshabilita si es igual a '0' lógico.

Read_WriteN.- Señal de entrada que indica si se va a realizar una escritura o una lectura; si Read_WriteN es igual a '1', indica lectura y si Read_WriteN es igual a '0', indica escritura.

I2CSUCC.- Señal de entrada proveniente del CORE que indica que el comando se ha ejecutado con éxito si I2CSUCC es igual a '1' lógico y que no ha llegado a su ejecución si I2CSUCC es igual a '0' lógico.

Ack.- Señal de entrada proveniente del CORE que indica si el dato enviado al dispositivo fue leído, si Ack es igual a '0' lógico, el dato fue leído y si Ack es igual a '1' lógico, el dato no fue leído.

I2CSTART.- Señal de salida enviada al CORE para que ejecute el comando START.

I2CSTOP.- Señal de salida enviada al CORE para que ejecute el comando STOP.

I2CRESTART.- Señal de salida enviada al CORE para que ejecute el comando RESTART.

I2CWRITE.- Señal de salida enviada al CORE para que ejecute el comando WRITE.

I2CREAD.- Señal de salida enviada al CORE para que ejecute el comando READ.

Descripción de datos de entrada y salida del circuito controlador del MÓDULO RTC ESCLAVO

DataIn.- Vector de 8 bits de entrada, posee el dato ingresado por el usuario (segundos, minutos, hora, día, fecha, mes o año).

AddressIn.- Vector de 8 bits de entrada, posee la dirección de registro.

Data.- Vector de 8 bits de salida, posee el dato enviado al CORE.

Data (salida) cambia dependiendo del segmento de la trama, en el siguiente orden: SlaveAddress_Write, AddressIn y DataIn. Ver Figura 7.

Data se trasmite desde el bit más significativo (MSB).

SlaveAddress_Write del RTC DS1307 es 1101000 + Read/Write o bit de dirección en cero (0) indicando escritura.

AddressIn es la dirección del registro (segundos, minutos, hora, día, fecha, mes o año) que va a ser seteado por el usuario.

DataIn posee el dato obtenido de un banco de switch permitiendo al usuario configurar la fecha y hora actual.

Seguido de cada byte trasmitido desde el MAESTRO al ESCLAVO el ESCLAVO retorna un Ack indicando su recepción, la ausencia de Ack nos indica que el byte no fue recibido.

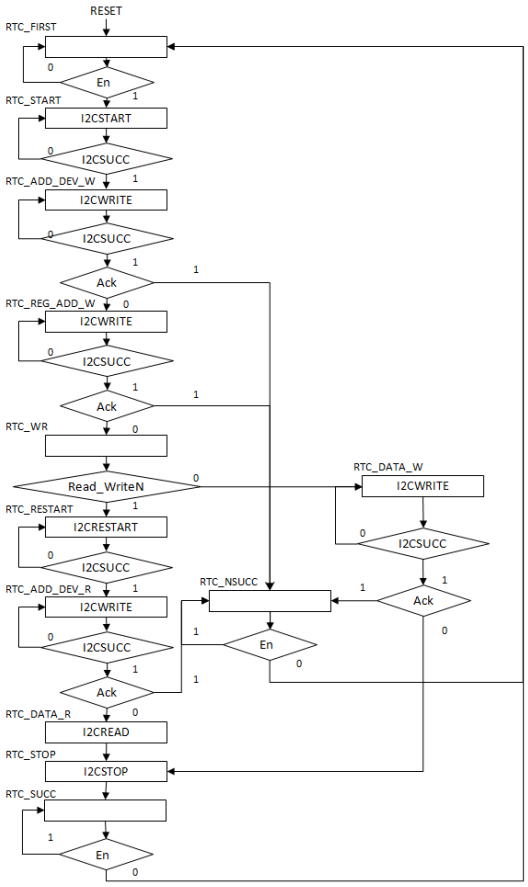

Figura 6. Diagrama ASM circuito controlador del módulo RTC esclavo

Trama de escritura al RTC DS1307

Escribir un dato en el RTC implica generar la siguiente trama.

Enviar condición START.

Escribir dirección del RTC con el bit RW en '0' lógico.

Leer ACK enviado por el RTC.

Escribir dirección de registro según la Tabla 1.

Leer ACK enviado por el RTC.

Escribir dato al RTC.

Leer ACK enviado por el RTC.

Enviar condición STOP. Ver Figura 7.

Tabla 1. Mapa de direcciones

00H	*SEGUNDOS*
.	*MINUTOS*
.	*HORAS*
.	*DIA*
.	*FECHA*
.	*MES*
.	*AÑIO*
7H	*CONTROL*
8H	*RAM 56x8*
3FH	

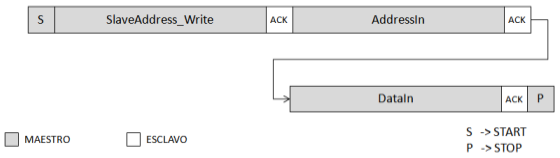

Figura 7. Trama de escritura al RTC DS1307

El proceso que debe ejecutarse para generar la trama de escritura es el descrito en los estados RTC_START, RTC_ADD_DEV_W, RTC_REG_ADD_W, RTC_WR, RTC_DATA_W y RTC_STOP.

Trama de lectura al RTC DS1307

Leer un dato del RTC implica generar la siguiente trama.

Enviar condición START.

Escribir dirección del RTC con el bit RW en '0' lógico.

Leer ACK enviado por el RTC.

Escribir dirección de registro según la TABLA 5.

Leer ACK enviado por el RTC.

Enviar condición de RESTART.

Escribir dirección del RTC con el bit WR en '1'.

Leer ACK enviado por el RTC.

Leer dato enviado por el RTC.

Enviar NACK.

Enviar condición STOP. Ver Figura 8.

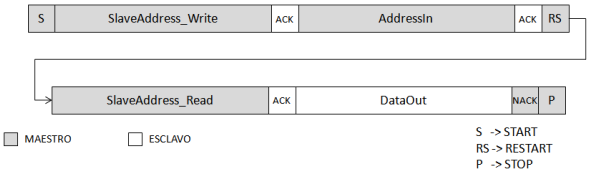

Figura 8. Trama de lectura al RTC DS1307

También puede ver la trama de lectura en el diagrama de tiempos en la Figura 19.

El proceso que debe ejecutarse para generar la trama de lectura es el descrito en los estados RTC_START, RTC_ADD_DEV_W, RTC_REG_ADD_W, RTC_WR, RTC_RESTART, RTC_ADD_DEV_R, RTC_DATA_R y RTC_STOP.

2.3. Diseño del modulo LCD

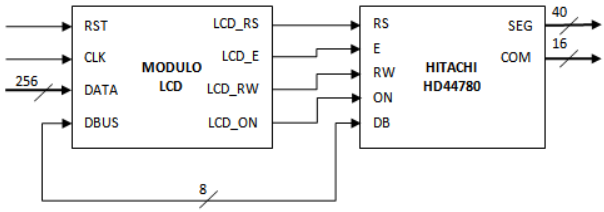

Figura 9. Diagrama de bloque de la interfaz LCD hitachi

Descripción de señales de entrada y salida del módulo LCD

RST- Resetea todas las señales del MÓDULO LCD.

CLK.- Señal de entrada de reloj.

LCD_RS.- Señal de salida, que habilita el envió de un dato o comando.

LCD_E.- Señal de salida, que habilita la LCD.

LCD_RW.- Señal de salida, que habilita la lectura o escritura a la LCD.

LCD_ON.- Señal de salida, que enciende la LCD.

Descripción de datos de entrada y salida del MÓDULO LCD

DATA.- Vector de 256 bits de entrada, posee fecha y hora leído del RTC.

DBUS.- Vector de 8 bits de entrada/salida, posee los caracteres de ingreso a la LCD.

2.4. Diseño de la aplicación ejemplo

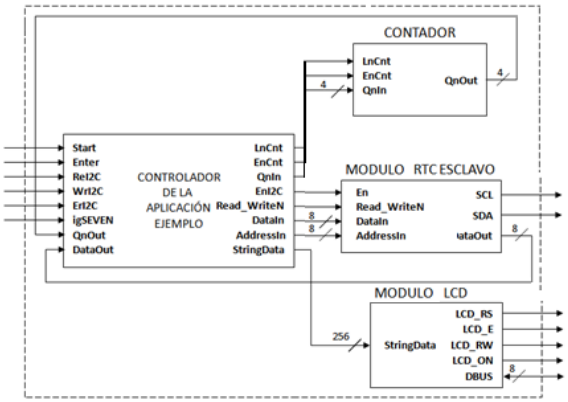

Figura10. Diagrama de bloque de la aplicación ejemplo

Descripción de señales de entrada y salida del circuito controlador de la Aplicación Ejemplo

Start.- Señal de entrada que es introducida por el usuario solicitando el arranque de la aplicación.

Enter.- Señal de entrada que habilita el paso de datos que se encuentra en el banco de switches al RTC.

ReI2C.- Señal de entrada que se genera cuando el MÓDULO RTC finaliza la lectura del dispositivo.

WrI2C.- Señal de entrada que se genera cuando el MÓDULO RTC finaliza la escritura en el dispositivo.

ErI2C.- Señal de entrada que se genera cuando en el MÓDULO RTC se produce un error.

igSEVEN.- Señal de entrada que se genera al comparar la salida del contador con.

LnCnt.- Señal de salida que setea el registro de entrada al contador.

EnCnt.- Señal de salida que habilita el contador.

EnI2C.- Señal de salida que habilita el módulo RTC.

Read_WriteN.- Señal de salida que indica si se va a escribir o leer.

Descripción de datos de entrada y salida del circuito controlador de la Aplicación Ejemplo

QnOut.- Vector de 4 bits de entrada generada por el CONTADOR.

DataOut.- Vector de 8 bits de entrada, posee el dato leído del RTC.

QnIn.- Vector de 4 bits de salida, igual a ‘0000’.

DataIn.- Vector de 8 bits de salida, posee el dato a escribirse en el RTC.

AddressIn.- Vector de 8 bits de salida, posee la dirección de registro a acceder deel RTC.

StringData.- Vector de 256 bits de salida, posee fecha y hora a escribirse en la LCD.

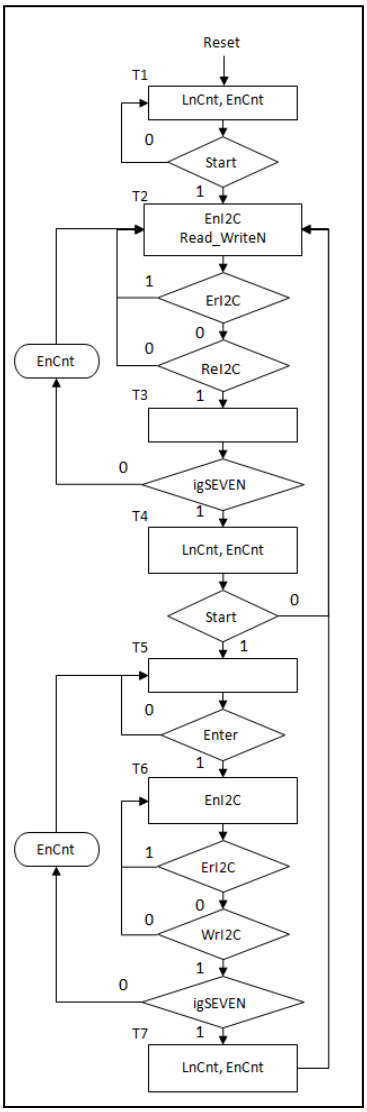

Figura 11. Diagrama ASM del circuito controlador de la aplicación ejemplo

3. Implementación

3.1 IPcore I2C MAESTRO

Para un correcto funcionamiento del módulo se debe resetear las señales: SCL, SDA e I2CSUCC;

setear un ‘1’ lógico en las señales SCL y SDA indicando que no hay comunicación, setear un ‘0’ lógico en la señal I2CSUCC indicando que no se ha ejecutado comando alguno; conociendo como comandos del protocolo I2C a: START, STOP, RESTART, WRITE y READ.

Enviar Condición START

El CORE setea un ‘1’ lógico seguido de un ‘0’ lógico en la línea SDA generando la transición de ‘alto a bajo’. Ver Figura 12.

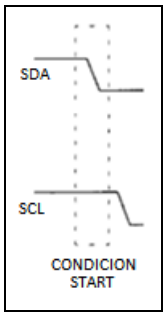

Figura 12. Transición de condición START

A continuación mostramos la implementación en código

```
WHEN IDLE =>
        SCL <= '1';
        SDA <= '1';
        I2C_STATE <= I2C_START;
WHEN I2C_START =>
        SCL <= '1';
        SDA <= '0';
        SUCCESSFUL <= '1';
```

Enviar condición STOP

El CORE setea un ‘0’ lógico seguido de un ‘1’ lógico en la línea SDA generando la transición de ‘bajo a alto’. Ver Figura 13.

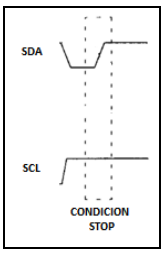

Figura 13. Condición STOP

A continuación mostramos la implementación en código

```
WHEN I2C_STOP1 =>
        SCL <= '0';
        SDA <= '0';
        I2C_STATE <= I2C_STOP2;
WHEN I2C_STOP2 =>
        SCL <= '1';
        SDA <= '0';
        I2C_STATE <= I2C_STOP3;
WHEN I2C_STOP3 =>
        SCL <= '1';
        SDA <= '1';
        SUCCESSFUL <= '1';
```

Enviar condición RESTART

El CORE setea un ‘1’ lógico seguido de un ‘0’ lógico en la línea SDA generando la transición de ‘alto a bajo’ ver Figura 14.

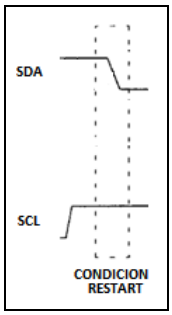

Figura 14. Condición RESTART

A continuación mostramos la implementación en código

```
WHEN I2C_RESTART1 =>
        SCL <= '0';
        SDA01 <= '1';
        IF SDA = '1' THEN
                I2C_STATE <= ERROR;
        ELSE
                I2C_STATE <= I2C_RESTART2;
        END IF;
WHEN I2C_RESTART2 =>
        SCL <= '1';
        SDA01 <= '1';
        I2C_STATE <= I2C_RESTART3;
WHEN I2C_RESTART3 =>
        SCL <= '1';
        SDA01 <= '0';
        SUCCESSFUL <= '1';
```

Escritura en ESCLAVO

El CORE envia los 8 bits contenidos en DataIn al ESCLAVO y lee el bit Ack. Ver Figura 15.

El estado I2C_WRITE_1 setea un ‘0’ lógico en SCL y envía el MSB de DataIn, el estado I2C_WRITE_2 setea un ‘1’ lógico en SCL e incrementa el índice que se utiliza para obtener un bit del DataIn (DataIn(bitcount)). Este cambio en SCL genera un tren de pulso mientras se va enviando el dato bit a bit empezando en el MSB.

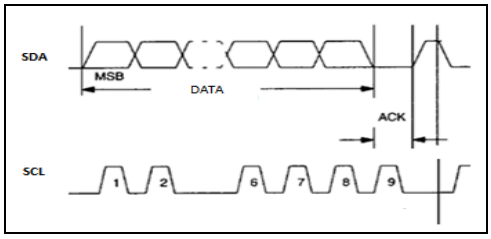

Figura 15. Escritura en esclavo

A continuación mostramos la implementación en código

```
WHEN I2C_WRITE_1 =>
        SCL <= '0';
        SDA <= DataIn(bitcount);
        I2C_STATE <= I2C_WRITE_2;
WHEN I2C_WRITE_2 =>
        SCL <= '1';
        IF (bitcount - 1) >= 0 THEN
                bitcount <= bitcount - 1;
                I2C_STATE <= I2C_WRITE_1;
        ELSE
                bitcount <= 7;
                I2C_STATE <= ACK1;
        END IF;
WHEN ACK1 =>
        SCL <= '0';
        SDA <= '1';
        I2C_STATE <= ACK2;
WHEN ACK2 =>
        SCL <= '1';
        Ack <= SDA;
        IF Ack = '1' THEN
                I2C_STATE <= ERROR;
        ELSE
                SUCCESSFUL <= '1';
        END IF;
```

Si Ack es igual a ‘0’ lógico, el dato ha sido recibido por el ESCLAVO y si Ack es igual a ‘1’ lógico, el dato no ha sido recibido por el ESCLAVO.

Lectura de ESCLAVO

El CORE recepta los 8 bits de dato enviado por el ESCLAVO y envía en el bit NACK. Ver Figura 16.

El estado I2C_READ_1 setea un ‘0’ lógico en SCL, el estado I2C_READ_2 setea un ‘1’ lógico en SCL e incrementa el índice. Este cambio en SCL genera un tren de pulso mientras recepta el dato bit a bit empezando por el MSB.

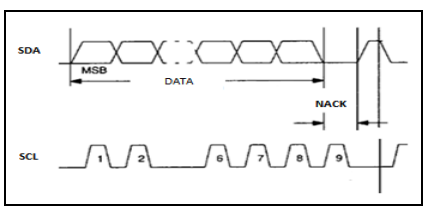

Figura 16. Lectura de esclavo

A continuación mostramos la implementación en código

```
WHEN I2C_READ_1 =>
        SCL <= '0';
        SDA <= '1';
        I2C_STATE <= I2C_READ_2;
WHEN I2C_READ_2 =>
        SCL <= '1';
        Data(bitcount) <= SDA;
        IF (bitcount - 1) >= 0 THEN
                bitcount <= bitcount - 1;
                I2C_STATE <= I2C_READ_1;
        ELSE
                bitcount <= 7;
                I2C_STATE <= I2C_READ_3;
        END IF;
WHEN I2C_READ_3 =>
        SCL <= '0';
        SDA <= '1';
        I2C_STATE <= NACK;    -- detiene lectura
        DataOut <= Data;
WHEN NACK =>
        SCL <= '1';
        SUCCESSFUL <= '1';
```

NOTA: los dispositivos conectados al bus I2C deben encargarse de crear la trama de escritura y lectura, como veremos en la implementación del MÓDULO RTC ESCLAVO.

3.2. Módulo RTC esclavo

El módulo RTC ESCLAVO es encargado de crear la trama de escritura y lectura activando los comandos para habilitar los procesos que crean la trama.

Trama de lectura al RTC DS1307

Los estados RTC_START, RTC_RESTART y RTC_STOP envían los comandos START, RESTART y STOP respectivamente; RTC_ADD_DEV_W escribe en el RTC la dirección del dispositivo (SlaveAddress_Write) b“1100100” + RW b‘0’ y verifica ACK; RTC_REG_ADD_W escribe en el RTC la dirección de registro (AddressIn) x“00”, x“01”, x“02”, x“03”, x“04”, x“05” ó x“06” dependiendo del dato a ser leído y verifica ACK; RTC_WR verifica si Read_WriteN es igual a ‘1’ lógico; RTC_ADD_DEV_R escribe en el RTC la dirección del dispositivo (SlaveAddress_Write) b”1100100” + RW b‘1’ y verifica ACK igual a ‘0’ lógico; RTC_DATA_R lee el dato del RTC y envía NACK.

Trama de escritura al RTC DS1307

Los estados RTC_START, RTC_RESTART y RTC_STOP envían los comandos START, RESTART y STOP respectivamente; RTC_ADD_DEV_W escribe en el RTC la dirección del dispositivo (SlaveAddress_Write) b“1100100” + RW b‘0’ y

verifica ACK; RTC_REG_ADD_W escribe en el RTC la dirección de registro (AddressIn) x“00”, x“01”, x“02”, x“03”, x“04”, x“05” ó x“06” dependiendo del dato a ser escrito y verifica ACK; RTC_WR verifica si Read_WriteN es igual a ‘0’ lógico; RTC_DATA_W escribe el dato en el RTC y verifica ACK.

ACK emitido por el RTC confirma recepción si es igual a ‘0’ lógico.

NACK emitido por el CORE indica fin de lectura del dato si es igual a ‘1’ lógico.

A continuación mostramos la implementación en código

```
SIGNAL RTC_STATE: RTC_STATES;
RTC_O: PROCESS (RTC_STATE)
BEGIN
CASE RTC_STATE IS
        WHEN RTC_START => I2CSTART <= '1';
        WHEN RTC_ADD_DEV_W => I2CWRITE
<= '1'; Data <= SlaveAddress_Write;
        WHEN RTC_REG_ADD_W => I2CWRITE
<= '1'; Data <= AddressIn;
        WHEN RTC_DATA_W => I2CWRITE <=
'1'; Data <= DataIn;
        WHEN RTC_RESTART => I2CRESTART
<= '1';
        WHEN RTC_ADD_DEV_R => I2CWRITE
<= '1'; Data <= SlaveAddress_Read;
        WHEN RTC_DATA_R => I2CREAD <= '1';
        WHEN RTC_STOP => I2CSTOP <= '1';
        WHEN RTC_NSUCC => ACK <= '1';
        WHEN RTC_SUCC => ACK <= '0';
END CASE;
END PROCESS RTC_O;
```

3.4 Implementación de la aplicación ejemplo

La Aplicación Ejemplo es un reloj-calendario que presenta fecha y hora en la LCD de la tarjeta DE2. Esta fecha y hora los lee del RTC mediante el protocolo de comunicación I2C, el usuario puede cambiar estos valores. En la Figura 17 se observa la salida en la LCD.

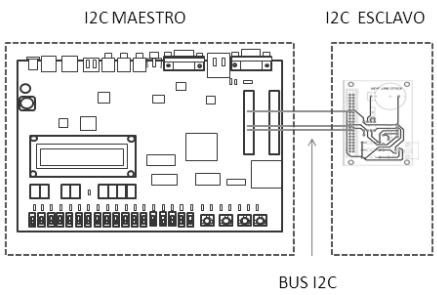

Figura 17. Implementación de la aplicación ejemplo

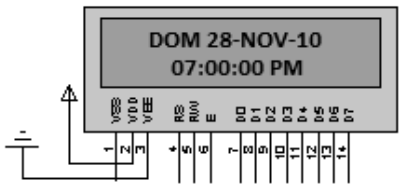

Figura 18. Fecha y hora visualizada en LCD

Escritura de fecha y hora

Los estados que intervienen en la lectura son: T5 y T6.

T5 activa el módulo RTC ESCLAVO en modo escritura seteando un ‘1’ lógico en EnI2C y un ‘0’ lógico en Read_WriteN, prepara de igual forma los datos a enviar al módulo RTC ESCLAVO.

Para obtener la dirección de registro se toma en consideración los datos generados por el contador; los datos se ingresan en un orden específico al menos en la fecha por las validaciones. Pasamos al siguiente estado solo cuando se active WrI2C que indica que el dato ha sido escrito con éxito; si se produjera un error se activaría ErI2C y permanecería en el estado T5 solicitando la lectura nuevamente.

Lectura de fecha y hora

La lectura debe ser a continuación de la hora y la fecha del RTC, para visualizar los datos en tiempo real. Los estados que intervienen en la lectura son: T2, T3 y T4.

T2 activa el MÓDULO RTC ESCLAVO en modo lectura seteando un ‘1’ lógico en EnI2C y en Read_WriteN, prepara de igual forma los datos a enviar al módulo RTC ESCLAVO. Ver Figura 19.

Un contador contabiliza los siete datos del RTC, el mismo que es utilizado para obtener la dirección de registro: x“00” para segundos, x“01” para minutos, x“02” para hora, x“03” para día, x“04” para fecha, x“05” para mes y x“06” para año. Esta se obtiene concatenando el dato de salida del contador con b“0000” como describe el código a continuación:

AddressIn <= "0000" & Cnt;

Pasamos al siguiente estado solo cuando se active ReI2C que indica que el dato ha sido leído con éxito; si se produjera un error se activaría ErI2C y permanecería en el estado T2 solicitando la lectura nuevamente. Ver Figura 19.

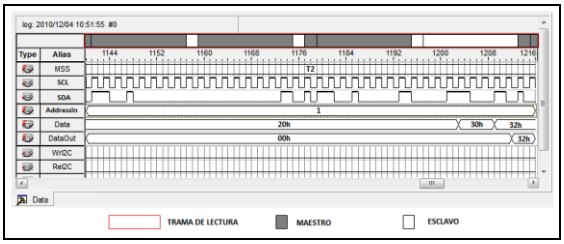

Figura 19. Trama de lectura

En este estado debe almacenarse los datos recibidos por el RTC, en las variables que correspondan.

A continuación mostramos la implementación en código

```
CASE Cnt IS
    WHEN "0000" => oSEG <= DataOUT;
    WHEN "0001" => oMIN <= DataOUT;
    WHEN "0010" => oHOR <= DataOUT;
    WHEN "0011" => oDIA <= DataOUT;
    WHEN "0100" => oFEC <= DataOUT;
    WHEN "0101" => oMES <= DataOUT;
    WHEN "0110" => oANI <= DataOUT;
END CASE;
```

T3 es un estado de transición, en este estado se pregunta si igSEVEN es igual a '1'; si no lo es igual se activa la salida condicionas EnCnt incrementando el contador; si fuera igual se pasa al siguiente estado.

T4 es un estado que inicializa el contador y pregunta si Start ha sido presionando, solicitando cambiar los datos en el RTC.

Ingreso

La interfaz de ingreso es sencilla, proporciona al usuario los siguientes botones:

Reset
Start
Enter

Y un banco de switches donde puede setear el dato en formato binario. Ej:

b"0000-0000" si estuviera ingresando segundos representaría 0 seg.

b"0000-0001" si estuviera ingresando mes representaría enero.

b"1000-0000" si estuviera ingresando año seria 2010.

Como se puede observar los bits más significativo representan las decenas y los menos significativos las unidades.

Salida

Enviar a publicar en la LCD, se toma los datos de la hora y la fecha independientemente que están posee los en formato binario, se los manda a la función convert_fecha_srtLCD o convert_hora_strLCD quien devuelve todos los valore listos para pasar a la LCD.

A continuación mostramos la implementación en código

```
strLCD_fecha <= convert_fecha_strLCD (oDIA, oFEC, oMES, oANI);
strLCD_hora <= convert_hora_strLCD (oHOR, oMIN, oSEG);
```

4. Conclusiones

[1] La utilization de VHDL estándar permite que la descripción de hardware pueda ser sintetizada para FPGAs de otros fabricantes.

[2] Este bus es ideal en sistemas donde no se requiere mucha velocidad en la transferencia de datos entre varios dispositivos, además requiere muy pocas líneas de circuito impreso (para SDA y para SCL).

[3] La implementación del protocolo I2C no resulto muy complicado, se lo pudo desarrollar en pocas líneas de código, permitiendo ahorrar recursos en nuestro FPGA y dando la posibilidad de cargarlo en cualquier FPGA.

[4] El hecho de trabajar solo con dos líneas para la comunicación, hace más sencillo la depuración de errores tanto en hardware como en software.

[5] Para trabajar con la interface I2C, la solución propuesta por nuestro proyecto resulta más fácil que trabajar con microncontroladores, ya que no tenemos necesidad de leer la hoja de datos del fabricante (Microchip, Atmel, etc) para conocer su arquitectura y poder configurar bien los registros como, SSPCON, SSPSAT y SSPBUF.

5. Consideraciones Futuras

[1] Soporte multimaestros.
[2] Soporte control de flujo.
[3] Soporte de wishbone y/o avalon.

6. Referencias

[4] NXP, The I2C-bus specification, disponible en: http://www.i2c-bus.org, consultado el: 5 de Noviembre del 2010.

[5] UNIVERSITY ILLINOIS, DE2 Development and Education Board, disponible en: http://courses.engr.illinois.edu/ece385/documents/DE2_UserManual.pdf, consultado el: 15 de Noviembre del 2010.

[6] WIKIPEDIA, QUARTUS II, disponible en: http://es.wikipedia.org/wiki/Quartus_II, consultado el: 7 de Diciembre del 2010.

[7] ALTERA, QUARTUS II Key Features, disponible en:http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html, consultado el: 7 de Diciembre del 2010.

[8] DALLAS SEMICONDUCTOR, DS1307 64 x 8 Serial Real-Time Clock, disponible en: http://www.datasheetcatalog.org/datasheet/maxim/DS1307.pdf, consultado el: 7 de Diciembre del 2010.

[9] WIKIPEDIA, FPGA, disponible en: http://es.wikipedia.org/wiki/FPGA, consultado el: 7 de Diciembre del 2010.

[10] DALLAS SEMICONDUCTOR, Crystal Considerations with Dallas Real-Time Clocks – RTCs. disponible en: http://www.maxim-ic.com/app-notes/index.mvp/id/58, consultado el: 8 de Diciembre del 2010.