ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO DE UN MODULO DE PROPIEDAD INTELECTUAL BASADO EN FPGA PARA EL MANEJO DEL BUS I2C"

TESINA DE SEMINARIO

PREVIO A LA OBTENCIÓN DEL TÍTULO DE:

INGENIERO EN COMPUTACION ESPECIALIZACIÓN SISTEMAS TECNOLOGICOS

INGENIERO EN CIENCIAS COMPUTACIONALES ESPECIALIZACIÓN SISTEMAS TECNOLOGICOS

Presentado por:

Renato Javier Manzano Araujo
Mónica Janina Pérez Avilés

Guayaquil – Ecuador

2010

AGRADECIMIENTO

Agradecemos a Dios por todas las bendiciones recibidas y el permitirnos realizar este trabajo. A nuestros padres, hermanos y amigos que han caminado junto a nosotros en los momentos difíciles.

DEDICATORIA

A Dios, a mis padres, hermanos, familiares y amigos por el respaldo permanente que me brindaron.

A Norma Avilés, madre y compañera fiel. A su apoyo incondicional y sus cuidados. A su disposición de mostrarme el horizonte las veces que fueran necesarias. A sus oraciones y bendiciones que me cubren día a día.

TRIBUNAL DE SUSTENTACIÓN

_____

Ing. Ronald Ponguillo

PROFESOR DEL SEMINARIO DE GRADUACIÓN

_____

Ing. Sara Ríos

PROFESOR DELEGADO DEL DECANO

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este Trabajo de Graduación, nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".

(Reglamento de Graduación de la ESPOL).

Renato Javier Manzano Araujo

Mónica Janina Pérez Avilés

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; conectadas a través de los puertos de entrada/salida de propósito general. 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. 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.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

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, permitiéndonos optimizar recursos de todos los dispositivos involucrados, ya que tendríamos más pines libres para otros propósitos.

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

Capítulo 1

1. Equipos y tecnología a usarse

HARDWARE

1.1. Protocolo I2C

1.1.1. Introducción

I2C fue desarrollado por PHILIPS, publicando su primera versión 1.0 en 1992 y su última publicación en el año 2000 con la versión 2.1.

El bus I2C soporta muchos procesos de fabricación de IC (nmos, cmos, bipolar). Tiene dos hilos, uno el dato serial (SDA) y el reloj serial (SCL), lleva la información entre los dispositivos conectados al bus. Cada dispositivo es reconocido por una dirección única (este puede ser un micro-controlador, driver LCD, memoria o interface de teclado) y puede operar como trasmisor o receptor, dependiendo la función del dispositivo. Obviamente un controlador de LCD solamente recibe, mientras que una memoria puede ser ambos, recibir y transmitir datos. Además de transmitir y recibir, los

dispositivos pueden también ser considerado como MAESTRO o ESCLAVOS cuando se está realizando la transferencia de datos. Un MAESTRO es un dispositivo el cual inicia la transferencia de datos en el bus y genera la señal de reloj para permitir esta transferencia. En este tiempo, cualquier otro dispositivo es considerado un ESCLAVO.

TERMINO	DESCRIPCION
Trasmisor	El dispositivo que envía datos al bus.
Receptor	El dispositivo que recibe datos del bus.
Maestro	El dispositivo que inicia la transferencia, genera la señal de reloj y termina la transferencia.
Esclavo	El dispositivo direccionado por un MAESTRO.
Multi-maestro	Más de un MAESTRO puede controlar el bus a la vez, sin corromper el mensaje.
Arbitraje	Proceso que asegura que, si más de un MAESTRO simultáneamente intenta controlar el bus, solamente a uno le es permitido para no corromper el mensaje.
Sincronización	Procedimiento para temporizar con la misma señal de reloj dos o más dispositivos.

Tabla 1 Terminologías del bus I2C

1.1.2. Protocolo I2C

El protocolo solo trabaja con 2 lineas para la comunicación, y ambas líneas son bidireccionales SDA y SCL, estas líneas son conectadas a un voltaje de fuente positivo a través de una fuente de corriente con resistencia PULL-UP, como se muestra en la figura1. Cuando el bus está libre, ambas líneas están en alto. Las etapas de salida de los dispositivos conectados al bus deben tener un drenador abierto o colector abierto para realizar el cableado y funciones. El dato en el bus I2C se puede transferir en velocidades de hasta 100 kbit /s en el modo normal o estándar, 400 kbit/s en el modo fast-mode, o hasta 3,4 mbit/s en el modo high-speed. El número de interfaces conectados al bus depende únicamente del límite de capacitancia de 400pf.

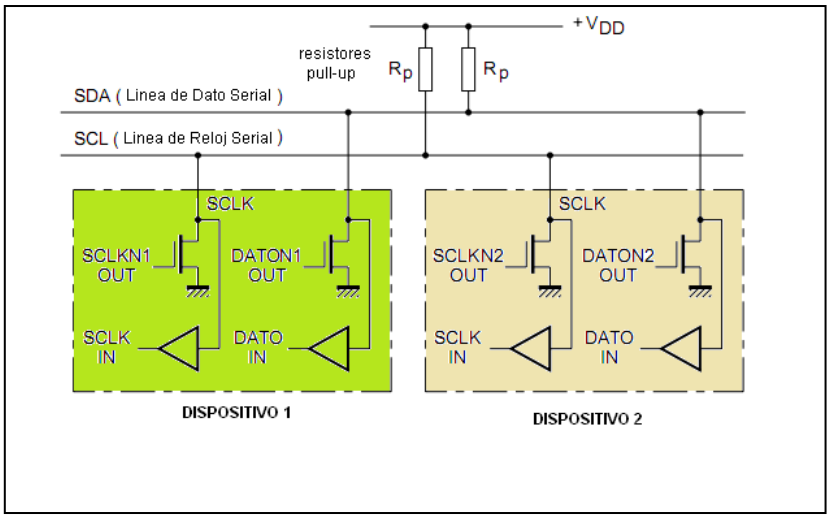

Figura 1 Conexión de dispositivos en el bus I2C

1.1.3. Bit de Transferencia

Debido a la variedad de dispositivos de tecnología diferentes (cmos, nmos, bipolar) que pueden ser conectado a al bus I2C, los niveles de lógica "0" (bajo) y "1" (high) no son fijos y dependen de los niveles asociados de VDD. Un pulso de reloj es generado para cada bit de datos transferido (figura 2).

Los datos de la línea SDA debe ser estable durante el período alto del reloj. El estado alto o bajo de la línea de datos solo puede cambiar cuando la señal de reloj en la línea SCL es baja.

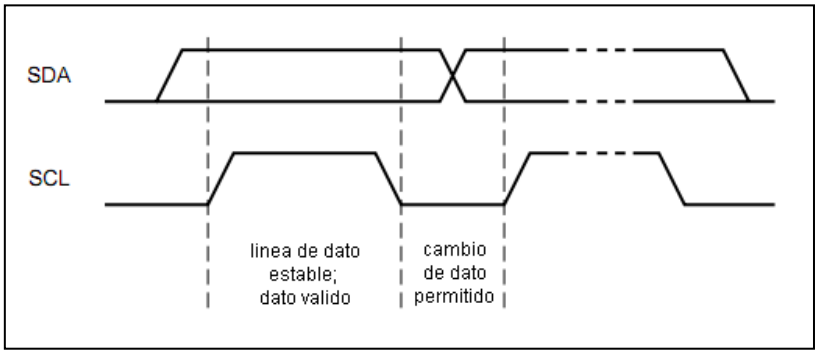

Figura 2 Bit de transferencia en el bus I2C

Condiciones de START y STOP

Dentro del proceso de transferencia de datos en el bus I2C hay dos situaciones básicas que son el inicio y fin. Estas son:

Inicio (START). – una transición de “1” a “0” (caída) en la línea de datos (SDA) mientras la línea (SCL) está en “1”.

Fin (STOP). – una transición de “0” a “1” (ascenso) en la línea de datos (SDA) mientras la línea de reloj (SCL) está en “1”.

Dicha situación se presenta en la figura 3.

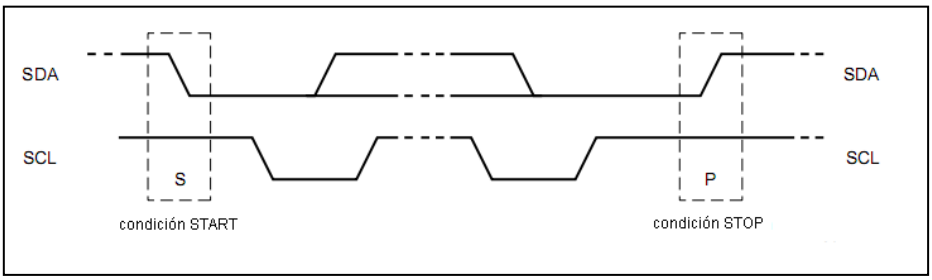

Figura 3 Condición de inicio y parada

Las condiciones de START y STOP son siempre generadas por el maestro. El bus i2c se considera ocupado después de la condición de START y desocupado de nuevo después de un cierto tiempo tras la condición de STOP.

Es decir, al pulso “1” de la línea SCL le puede corresponder un pulso “0” o “1” de la línea SDA en función de la información del byte que se envíe, recordemos que a cada bit de SDA le corresponde un bit de SCL, pero nunca (salvo en la condición de inicio) a un bit de SCL le corresponde una situación de “1” a “0” o sea, pasa por dos estados la línea SDA. Ocurre la condición contraria en la condición de stop, que el maestro envía un bit a

SCL mientras cambia SDA de “0” a ”1” durante el tiempo que está enviando la señal de “1” a SCL.

El bus continuará ocupado si se genera una repetición de la condición de START (SR) en lugar de la condición de STOP (P). En este aspecto, la condición START y la de repetición de START (SR) son funcionalmente idénticas.

La detección de condiciones de START y STOP por los dispositivos conectados al bus es fácil si se incorpora el hardware necesario. No obstante, micro-controladores que no disponen de esta interface tienen que muestrear la línea SDA por lo menos dos veces por ciclo de reloj para detectar la transferencia.

1.1.4. Transferencia de datos

Formato Byte

Cada byte que se pone en la línea SDA debe ser de 8-bits de longitud. El número de bytes que se pueden transmitir no tiene restricción. Cada byte tiene que estar seguido de un bit de ACK enviado por el receptor (vea la figura 4). Los datos se transfieren con el bit más significativo primero (MSB). Si un dispositivo esclavo no puede recibir o transmitir un byte de datos completo (puede estar haciendo otra cosa, por ejemplo atendiendo una interrupción), puede mantener la línea de reloj SCL a “0” lo que forza al MAESTRO a permanecer un estado de espera. Los datos continúan transfiriéndose cuando el dispositivo esclavo está listo para otro byte de datos y libere la línea de reloj SCL.

En algunos casos, es permitido usar un formato diferente del formato del bus I2C (CBUS para dispositivos compatibles por ejemplo). Un mensaje que comienza con una dirección puede ser terminada por la generación de una condición de STOP, incluso durante la transmisión de un byte. En este caso, un NACK es generado.

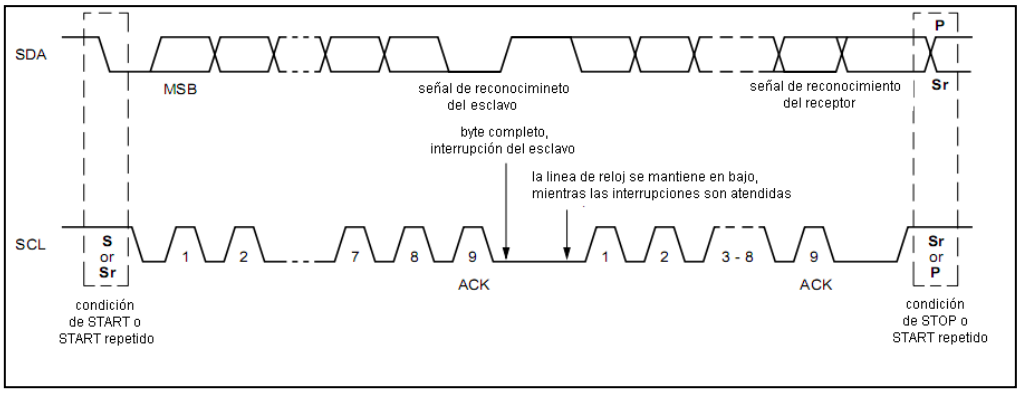

Figura 4 Transferencia de datos del bus

ACK

La transferencia de datos con ACK es obligatoria. El pulso de reloj ACK correspondiente es generado por el maestro. El transmisor libera la línea SDA (“1”) durante el pulso de reloj de ACK.

El receptor debe poner a nivel bajo la línea SDA durante el pulso de reloj ACK para que siga siendo “0” durante el tiempo que el maestro genera el pulso “1” de ACK. Se puede observar en la figura 5.

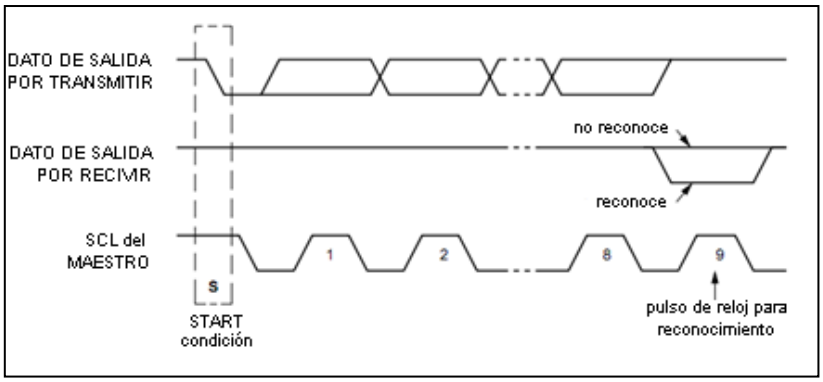

Figura 5 Ack en el bus I2C

Generalmente, un receptor que ha sido direccionado es obligado a generar un ACK después de cada byte que se ha recibido, excepto cuando el mensaje empiece con una dirección CBUS.

Cuando un ESCLAVO no genera el ACK (porque está haciendo otra cosa y no puedo atender el bus), debe mantener la línea SDA en alto durante el bit ACK. El MAESTRO puede entonces generar una condición de PARADA abortando la transferencia de datos, o repetir la condición de INCIO para empezar una nueva transferencia de datos.

Si un ESCLAVO-RECEPTOR que está direccionando no desea recibir más bytes, el MAESTRO debe detectar la situación y no enviar más byte. Esto se indica porque el ESCLAVO no genera el bit ACK en el primer byte. El ESCLAVO pone la línea SDA a “1” lo que es detectado por el MAESTRO el cual genera la condición de STOP o repite la condición de START.

Si un MAESTRO-RECEPTOR está recibiendo datos de un ESCLAVO-TRANSMISOR debe generar un bit ACK después de cada byte recibido del ESCLAVO, para finalizar la transferencia de datos no debe generar el ACK después del último byte enviado por el ESCLAVO. El ESCLAVO-

TRANSMISOR debe permitir desbloquear la línea de datos para permitir que el MAESTRO genere un STOP o una condición repetida de START.

1.1.5. Generación de reloj y arbitraje

Sincronización

Todos los MAESTROS generan su propio reloj en la línea SCL para transferir datos sobre el bus I2C. El dato solo es válido durante el ciclo de reloj a nivel alto. Por tanto, necesitaríamos un clock definido para el arbitraje bit a bit.

La sincronización del reloj se realiza utilizando la conexión AND de todos los dispositivos del bus a la línea SCL. Esto significa que una transición de un maestro de “1” a “0” en la línea SCL hace que la línea pase a “0”, esto mantiene la línea SCL en este estado. Sin embargo la transición de “0” a “1” no cambia el estado de la línea SCL si otro reloj está todavía en su periodo de “0”. Por lo tanto la línea SCL permanecerá a “0” tanto como el periodo más largo de cualquier dispositivo cuyo nivel sea “0”. Los dispositivos que tiene un periodo más corto de reloj “0” entran en un periodo de espera.

Cuando todos los dispositivos conectados al bus han terminado con su periodo “0”, la línea del reloj se desbloquea y pasa a nivel “1”. Por lo que hay que diferenciar entre los estados de reloj de los dispositivos y los estados de la línea SCL. El primer dispositivo que completa su nivel “1” pone nuevamente la línea SCL a “0”.

Resumiendo, la línea SCL permanecerá en “0” durante el periodo más largo en BAJO (nivel “0”) de todos los dispositivos maestros, y permanecerá en

“1” durante el periodo más corto en ALTO (nivel “1”) de todos los maestros conectados. Se observa en la figura 6.

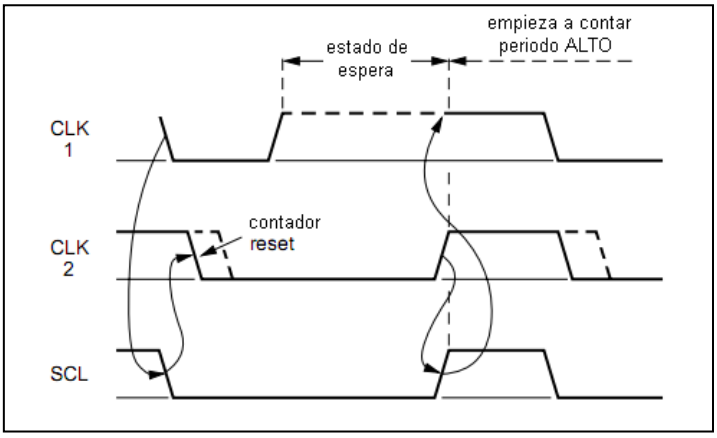

Figura 6 Sincronización de la señal de reloj

Arbitraje

Un MAESTRO puede iniciar una transmisión solo si el bus esta libre. Dos o más maestros pueden generar una condición de START en el bus lo que da como resultado una condición de START general. Cada MAESTRO debe comprobar si el bit de datos que transmite junto a su pulso de reloj, coincide con el nivel lógico en la línea SDA. El sistema de arbitraje actúa sobre la línea de datos SDA, mientras la línea SCL está en nivel “1”, de una manera tal que el maestro que transmite un nivel “1”, pierde el arbitraje sobre otro MAESTRO que envía un nivel ”0” a la línea de datos SDA. Esta situación continua hasta que se detecte la condición de STOP generada por el maestro que se hizo cargo del bus.

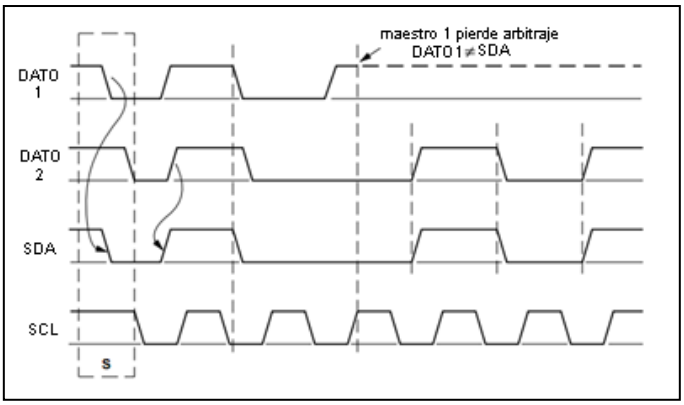

Figura 7 Arbitraje entre 2 maestros

En la figura 7 se ve el arbitraje entre dos MAESTROS, aunque pueden estar involucrados más dependiendo de cuantos dispositivos hay conectados en el bus. En el momento en que hay una diferencia entre el nivel interno de la línea de datos del maestro DATA 1 y el actual nivel de la línea de datos SDA, su salida de datos es interrumpida, lo cual significa que un nivel “1” esta dominando el bus. Esto no afecta los datos transferidos inicialmente por el MAESTRO que toma el bus.

El arbitraje puede continuar varios bits hasta que se da la circunstancia de control del bus por uno de los MAESTRO. Tras el arbitraje, los MAESTROS perdedores se deben poner inmediatamente en modo MAESTRO-RECEPTOR y ESCLAVO pues los datos que envíe el MAESTRO dominante puede ser para uno de ellos.

Un MAESTRO que pierde el arbitraje puede generar pulsos de reloj hasta el fin de byte en el cual el pierde el arbitraje. En el momento que un MAESTRO toma el control solo este MAESTRO toma las decisiones y genera los códigos de dirección, no existen MAESTROS centrales, ni existen órdenes prioritarias en el bus.

Se debe poner especial atención si durante una transferencia de datos el procedimiento de arbitraje está todavía en proceso justo en el momento en el que se envía al bus una condición de STOP. Es posible que esta situación pueda ocurrir, en este caso el maestro afectado debe mandar condiciones de START o STOP.

1.1.6. Ventajas del Protocolo I2C

- Mucho de los semiconductores manufacturados son de bajo costo y tiene la compatibilidad del bus I2C, memorias EEPROMs, Relojes de Tiempo real, ADCs, DACs, Controladores de motores PWM, potenciómetros digitales, sensores digitales de temperatura, etc.
- Mucho de estos circuitos integrados son de 8 pines, lo que hace más pequeño el circuito.
- Se pueden conectar muchos dispositivos esclavos solamente usando 2 pines del dispositivo (microcontrolador, fpga, etc), lo cual es muy eficiente.
- El diseño del bus es muy simple, usa 2 líneas y 2 resistencias.
- Reducción de costos al fabricar, montar y probar dispositivos I2C.
- Permite control de flujo.
- Detección de errores.
- Administración del bus entre varios dispositivos (multimaestros).

1.1.7. Desventajas del Protocolo I2C

- En si el protocolo de comunicaciones del bus I2C puede resultar un poco complicado.

- Cada circuito integrado esclavo tiene parámetros únicos como por ejemplo la dirección (slave adress), la forma de enviar y recibir datos, por lo tanto tendrás que tener la hoja de datos del fabricante a la mano.
- Baja velocidad comparada con otros protocolos de comunicación como el SPI (creado por motorola).
- Solo permite distancias cortas de acuerdo con la hoja del fabricante (limitado a 400pf incluyendo el bus y todos los dispositivos conectados).

1.1.8. Aplicaciones del Protocolo I2C

A pesar de su lentitud en comparación con otros protocolos, se lo sigue utilizando en: selectores de tv, video juegos, memorias EEPROM, sensores de temperatura, sensores de humedad, medidores de distancia, relojes calendario, dispositivos de seguridad electrónica.

1.2. FPGA

1.2.1. Descripción

El FPGA (Field Programmable Gate Array) es un dispositivo semiconductor que contiene bloques de elementos lógicos cuya interconexión y funcionalidad puede ser configurada 'in situ' mediante un lenguaje de programación especializado. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip.

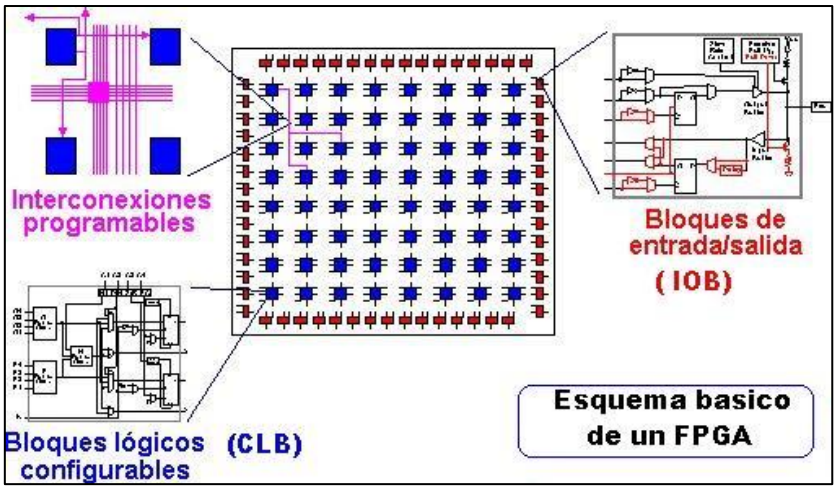

Figura 8 Esquema básico de un FPGA

1.2.2. Características

Una jerarquía de interconexiones programables permite a los bloques lógicos de un FPGA ser interconectados según la necesidad del diseñador del sistema, algo parecido a un breadboard(es una placa de uso genérico reutilizable o semi permanente) programable. Estos bloques lógicos e interconexiones pueden ser programados después del proceso de manufactura por el usuario (diseñador), así que el FPGA puede desempeñar cualquier función lógica necesaria.

Una tendencia reciente ha sido combinar los bloques lógicos e interconexiones de los FPGA con microprocesadores y periféricos relacionados para formar un **"**Sistema programable en un chip". Ejemplo de tales tecnologías híbridas pueden ser encontradas en los dispositivos Virtex-II PRO y Virtex-4 de Xilinx, los cuales incluyen uno o más procesadores PowerPC embebidos junto con la lógica del FPGA. El FPSLIC de Atmel es otro dispositivo similar, el cual usa un procesador AVR en combinación con la arquitectura lógica programable de Atmel. Otra alternativa es hacer uso de

núcleos de procesadores implementados haciendo uso de la lógica del FPGA. Esos núcleos incluyen los procesadores MicroBlaze y PicoBlaze de Xlinx, Nios y Nios II de Altera, y los procesadores de código abierto LatticeMicro32 y LatticeMicro8.

Muchos FPGA modernos soportan la reconfiguración parcial del sistema, permitiendo que una parte del diseño sea reprogramada, mientras las demás partes siguen funcionando. Este es el principio de la idea de la “computación reconfigurable”, o los “sistemas reconfigurables”.

1.2.3. Programación

La tarea del programador es definir la función lógica que realizará cada uno de los CLB (bloques lógicos programable), seleccionar el modo de trabajo de cada IOB (bloques de entrada y salida) e interconectarlos.

El diseñador cuenta con la ayuda de entornos de desarrollo especializados en el diseño de sistemas a implementarse en un FPGA. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de un lenguaje de programación especial. Estos lenguajes de programación especiales son conocidos como HDL o Hardware Description Language (lenguajes de descripción de hardware). Los HDLs más utilizados son:

- VHDL
- Verilog
- ABEL
- SistemC

1.2.4. Tecnologías de la memoria de programación

Las FPGAs también se pueden diferenciar por utilizar diferentes tecnologías de memoria:

- Volátiles: Basadas en RAM. Su programación se pierde al quitar la alimentación. Requieren una memoria externa no volátil para configurarlas al arrancar (antes o durante el reset).
- No Volátiles: Basadas en ROM. Hay de dos tipos, las reprogramables y las no reprogramables.

1.2.5. Ventajas del FPGA

- Paralelismo en la ejecución de instrucciones.
- Es reprogramable (en la mayoría de los modelos).
- Permite encriptación del código grabado en el dispositivo (LOCK)**.**
- Permite el desarrollo de microprocesadores en Software.
- Escalabilidad en cuanto al procesamiento de datos (cloud FPGA)
- Permite optimizar recursos (celdas logicas), de acuerdo a nuestras necesidades.

1.2.6. Desventajas del FPGA

- Costo muy elevado del FPGA.
- Mientras el FPGA incrementa las capacidades, los fabricantes lo encapsulan en BGA o miro-BGA, dificultando el montaje con la tecnología que se encuentra en nuestro país.

1.2.7. Aplicaciones del FPGA

Cualquier circuito de aplicación específica puede ser implementado en un FPGA, siempre y cuando esta disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan los FPGA incluyen a los DSP (procesamiento digital de señales), radio definido por software, sistemas aeroespaciales y de defensa, prototipos de ASICs, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, entre otras. Cabe notar que su uso en otras áreas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo.

1.3. Tarjeta DE2 de ALTERA

1.3.1. Descripción

DE2 es un kit de desarrollo para proyectos que involucran un Cyclone II 2C35 (FPGA de altera), Esta tarjeta ya viene con hardware montado como son:

- Socket para tarjeta SD.
- 4 botoneras.
- 18 switches.
- 18 LEDs rojos.
- 9 LEDs verdes.
- Oscilador de 50MHz y 27-MHz para reloj.

- Codec de audio de 24-bit de alta calidad con linea de entrada, línea de salida, y micrófono.
- Convertidor Análogo digital VGA con un conector de salida VGA.
- Decodificador de TV (NTSC/PAL) y conector de entrada de TV.
- Controlador Ethernet 10/100 con un conector.
- Controlador USB Host/esclavo con conector tipo USB-A y USB-B.
- Transceiver RS-232 y conector de 9 pines.
- Conector PS/2 mouse/teclado.
- Transceiver IrDA.
- Dos conectores de expansión de 40 pines (IDE) con diodos de protección.

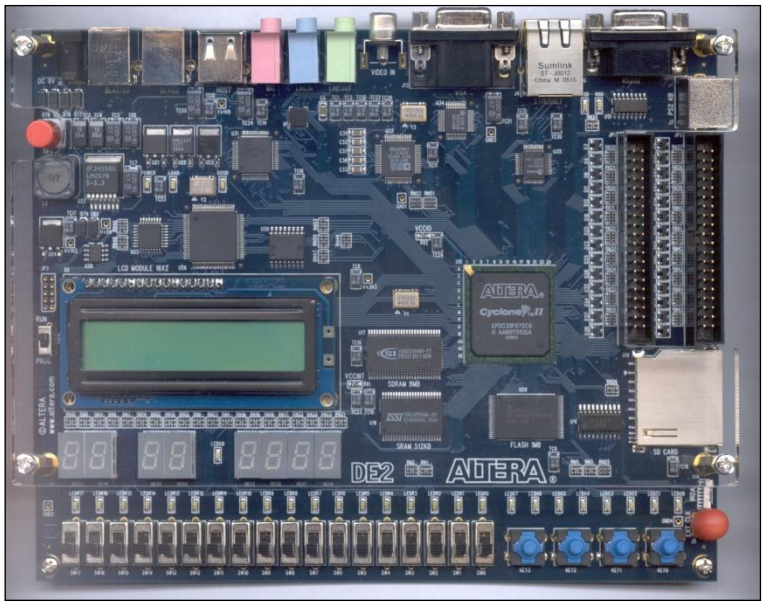

Figura 9 Tarjeta DE2 de ALTERA

1.3.2. Características

Cyclone II 2C35 FPGA

- 33,216 Elementos lógicos.

- 105 M4K Bloques de RAM.
- 483,840 bits de RAM.
- 35 multiplicadores en hardware.
- 4 PLLs.
- 475 pines de Entrada/Salida.
- Empaquetado FineLine BGA de 672 pines.

Dispositivo de configuración general y circuito USB Blaster

- Dispositivo de configuración Serial en EPCS16 de Altera.
- Programación USB Blaster en el circuito.
- JTAG y programación en modo AS son soportados.

SRAM

- Chip de memoria RAM Estática de 512-Kbyte.
- Organizado en 256K x 16 bits.
- Accesible para el procesador Nios II y por el Panel de Control DE2.

SDRAM

- Chip de memoria RAM Dinámica de 8-Mbyte.
- Organizado en 4 bancos de 1M x 16 bits.
- Accesible para el procesador Nios II y por el Panel de Control DE2.

Memoria Flash

- Memoria Flash de 4Mbyte tipo NOR (1 Mbyte en el algunas tarjetas).
- Bus de datos 8-bit.
- Accessible por el procesador Nios II y el Panel de Control DE2.

Socket de tarjeta SD

- Acceso a la tarjeta SD por modo SPI.
- Accesibilidad a la memoria por el procesador Nios II con el controlador de la tarjeta SD del DE2.

Botoneras

- 4 botones.
- Circuito disparador sin rebote.
- Normalmente en ALTO; genera un pulso BAJO cuando el switch es presionado.

Banco de switches

- 18 switches para el usuario.
- Un switch causa 0 lógico cuando está en la posición BAJA y causa un 1 lógico 1 cuando está en la posición ALTA.

Entradas de Reloj

- Oscilador de 50MHz.
- Oscilador de 27MHz.
- Entrada externa de reloj SMA.

CODEC de Audio

- CODEC de audio Wolfson WM8731 de 24-bit sigma-delta.
- Entrada de línea-nivel, Salida de línea-nivel, y entrada de micrófono.
- Probador de frecuencia: 8 a 96 KHz.

- Aplicaciones para grabadores/ reproductores MP3 y grabadores, PDAs, teléfonos inteligentes, grabadores de voz, etc.

Salida VGA

- Usa un DAC de Video ADV7123 de 240MHz de 10bit.
- Con 15 pines y un conector D-sub.
- Soporta hasta 1600x1200 con una frecuencia de refresco de 100Hz.
- Puede ser usado con Cyclone II FPGA implementando ALTO-DESEMPEÑO en el TV Encoder.

Circuito decodificador de TV NTSC/PAL

- Usa Decoder de Video ADV7181B Multi-formato SDTV
- Soporta NTSC(M,J,4.43), PAL(B/D/G/H/I/M/N), SECAM
- Integra 3 ADCs de 54MHz de 9bit.
- Oscilador simple de 27MHz.
- Soporta Video Compuesto (CVBS) conector de entrada RCA.
- Soporta formato de salida digital (8-bit/16-bit): ITU-R BT.656 YCrCb 4:2:2 salida + HS, VS, y FIELD
- Aplicaciones: Grabadores DVD , LCD TV, cajas Set-top, TV Digital, Dispositivos de video portables.

Controlador 10/100 Ethernet

- MAC integrada y PHY con un procesador general de interface.
- Soporta aplicaciones 100Base-T y 10Base-T.
- Soporta full-duplex en 10 Mb/s y 100 Mb/s, con auto-MDIX.

- Completamente compatible con las especificaciones de la IEEE 802.3u.
- Soporta generación de checksum IP/TCP/UDP y checking.

Controlador USB Host/Esclavo

- Compatibilidad completa con la especificación de la Universal Serial Bus Rev. 2.0
- Soporta transferencia de datos en alta-velocidad y baja-velocidad
- Soporta USB host y USB esclavo.
- Dos puertos USB (uno tipo A para host y otro tipo B para dispositivos).
- Provee una velocidad alta en interface paralelo que deberá ser habilitado por un procesador; soporta Nios II con un controlador Terasic.
- Soporta Entrada/Salida Programada I/O (PIO) y Acceso de Memoria Directo (DMA).

Puerto Serial

- Un puerto RS-232.
- Un puerto PS/2.
- Conector DB-9 serial para el puerto RS-232.
- Conector PS/2 para conectar un mouse PS2 o teclado en la tarjeta DE2.

Transceiver IrDA

- Contiene un transceiver infrarojo de 115.2-kb/s.
- Corriente del controlador de LED de 32 mA.
- Entrada de detección de flanco

Dos conectores de Expansión (IDE) de 40 pines

- 72 pines de Entrada/Salida del Cyclone II, y 8 pines de alimentación de energía y líneas de tierra.
- Conectores de expansión.
- Los conectores de 40 pines con diseñados y aceptado por el estándar del cable usado para los controladores de disco duro IDE.
- Provee una protección de Diodos y resistores.

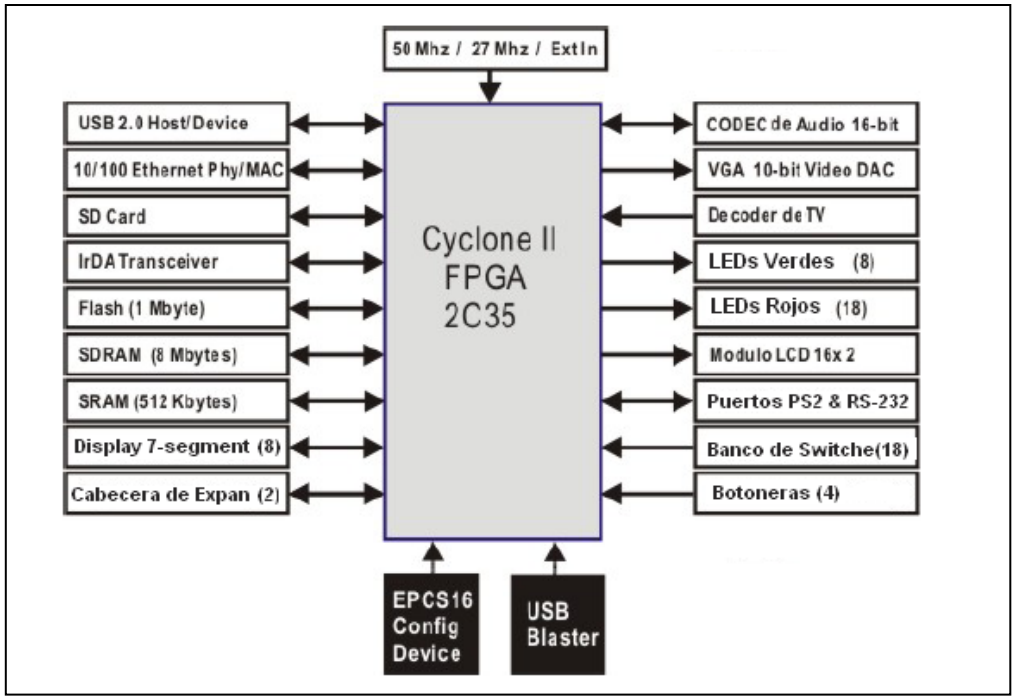

Figura 10 Diagrama de bloque de la tarjeta DE2

1.3.3. Ventajas de la Tarjeta DE2

- Es una buena opción para las personas que recién empiezan a desarrollar proyecto basados en FPGA, ya que tiene un gran número de componentes(hardware) para probar nuestros diseños
- Rapidez en el desarrollo de proyectos de pequeña y mediana escala.
- Fácil programación a través del USB BLASTER
- El tamaño de la memoria flash tiene lo necesario para proyectos de pequeña y mediana escala como también para efectos de depuración de código.

1.3.4. Desventajas de la Tarjeta DE2

- Costo de la tarjeta de desarrollo es un poco elevado
- El tamaño de la memoria flash puede resultar pequeña para el análisis de señales a través de SignalTapII, por ejemplo
- Las botoneras se desgastan al poco tiempo
- Si se está desarrollando proyecto medianos, necesita de licencia para la compilación incremental

1.3.5. Módulo LCD

1.3.6. Descripción

LCD basado en HITACHI, configurado a 32 caracteres, 2 líneas, 5X8 puntos por carácter y un reloj de 400Hz.

1.3.7. Operación

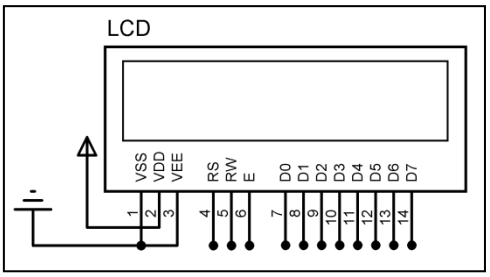

Figura 11 Esquema del módulo LCD

El LCD trabaja con lógica TTL . Se lo puede encender con intensidad retro-iluminada constante o variable, haciendo uso de un potenciómetro entre las patas de VSS y VDD controlándolo con VEE. Este LCD tiene para trabajar en modo de 4 bit a 8 bit de datos.

COMANDO	RS	R/W	DATAB								DESCRIPCIÓN
			7	6	5	4	3	2	1	0	
LIMPIAR PANTALLA	0	0	0	0	0	0	0	0	0	1	Limpia la entrada del display y setea la dirección o de la DDRAM en el contador de direcciones.
RETORNAR A INICIO	0	0	0	0	0	0	0	0	1	-	Setea la dirección de la DDRAM en el contador de direcciones. También regresa el display a su posición original. El contenido de la DDRAM no es cambiado.
MODO DE ENTRADA	0	0	0	0	0	0	0	1	I/D	S	Setea la dirección de movimiento del cursor y especifica el desplazamiento en pantalla.
CONTROL DE ENCENDIDO Y APAGADO	0	0	0	0	0	0	1	D	C	B	Setea la entrada del display para encender/apagar (D), encender/apagar cursor (C), parpadeo del cursor (B).
DESPLAZAMIENTO DEL CURSOR O DISPLAY	0	0	0	0	0	1	S/C	R/L	-	-	Mueve el curso y desplaza el display sin cambiar el contenido de la DDRAM.
SETEO DE FUNCIÓN	0	0	0	0	1	DL	N	F	-	-	Setea la longitud de datos de la interfaz (DL), Número de líneas en el display (N), y la fuente (texto) (F).

Tabla 2 Instrucciones del LCD

SEÑAL	DESCRIPCIÓN
I/D	1: Incrementa. 0: Decremento.
S	1:Pantalla se mueve
	0:Pantalla no se mueve
S/C	1: Mueve la Pantalla 0: Mueve el Cursor.
R/L	1: Desplaza a la derecha. 0: Desplaza a la izquierda.
DL	1: 8 bits. 0: 4 bits.
N	1: 2 líneas. 0: 1 línea.
B	1: cursor con parpadeo
	0:cursor sin parpadeo
F	1: 5 × 10 puntos. 0: 5 × 8 puntos.
BF	1: Controlador ocupado. 0: Controlador Libre.

Tabla 3 Terminología de Instrucciones del LCD

1.3.8. Descripción de Instrucciones

Limpiar Pantalla

Limpia la pantalla escribiendo el código ASCII 20H que corresponde al espacio en blanco, dentro de todas las direcciones de la DDRAM (Display Data RAM), provocando que todo el mensaje escrito desaparezca. A continuación setea la dirección 0 de DDRAM dentro del contador de direcciones, de esta manera coloca el cursor en la posición 1,1 de la pantalla. En otras palabras, la pantalla es borrada y el cursor retorna a la posición más izquierda de la primera línea (en el caso de tener más de una línea). Además en el Modo de entrada cambia I/D a 1 (modo incremento), sin cambiar el valor de S (Shift).

Retornar a inicio

Pone la dirección 0 dentro del contador de dirección de DDRAM, y devuelve al display su estado original en el caso de que hayan sido desplazados. El contenido de DDRAM no cambia.

El cursor se va al extremo izquierdo de la primera línea de la pantalla (en la el caso de tener más de una línea).

Modo de entrada

I/D: Incrementa (I/D = 1) o decrementa (I/D = 0) la dirección DDRAM por 1 cuando un código de caracter es escrito o leído dentro de la DDRAM.

El cursor se mueve a la derecha cuando se incrementa en uno y a la izquierda cuando se decrementa en 1.

Lo mismo se aplica cuando se escribe o se lee de la CGRAM (Character Generator RAM).

S: Si S es 1, desplaza la pantalla a la derecha (I/D = 0) o a la izquierda (I/D = 1). Si S=0 la pantalla no se desplaza, se queda “quieto”.

Cuando S= 1, aparecerá como si el cursor no se mueve en la pantalla (normalmente se utiliza I/D=1 y S=0 por lo que el comando típico es 0x06). La pantalla no cambia cuando se está leyendo de la DDRAM. Además la escritura dentro o la lectura fuera de la CGRAM no cambia el desplazamiento de la pantalla.

Control de encendido/apagado

D: La pantalla se enciende cuando D es 1 y se apaga cuando D es 0. Cuando está apagado, la visualización de los datos permanecen en la DDRAM, pero puede ser presentado instantáneamente cuando se setea D a 1.

C: El cursor se muestra cuando C es 1 y no se muestra cuando C es 0. Incuso si el cursor desaparece, la función de I/D u otras especificaciones no cambiaran durante la visualización de los datos a escribir. El cursor se muestra usando 5 puntos en la 8va línea para 5X8 puntos por caracter de la selección de fuente y dentro de la 11ava línea para 5X10 puntos por caracter de selección.

B: Cuando B es 1, el caracter indicado por el curso parpadeara. El parpadeo se muestra como el cambio entre todos los puntos blancos y los caracteres presentados a una velocidad de intervalos de 409.6-ms cuando fcp o fOSC es 250 kHz. El cursor parpadea y puede ser seteado para presentarlo simultáneamente. (El parpadeo de frecuencia cambia de acuerdo a fOSC o el reciproco fcp. Por ejemplo, cuando fcp is 270 kHz, 409.6 × 250/270 =379.2 ms.)HD44780U194

Desplazamiento de la pantalla o cursor

Desplaza a la derecha o izquierda, la posición del cursor o pantalla, sin necesidad de escribir o leer los datos en la pantalla (con S/C=1 se mueve la pantalla, con S/C=0 el cursor). Esta función se emplea normalmente para modificar solo algunos datos en la pantalla, evitando la necesidad de

limpiarla (borrarla) y escribir nuevamente todo su contenido. Cuando pasa el 40 avo digito de la primera línea de la pantalla, el cursor se mueve automáticamente a la segunda línea de la pantalla, pero de la segunda línea no regresara a la primera.

El contenido del contador de dirección (AC) no cambiara si la única acción que se realiza es un desplazamiento de pantalla.

Seteo de función

DL: Setea la longitud de datos de la interface. El Dato es enviado o recibido en longitud de 8-bit (DB7 a DB0) cuando DL es 1, y en longitud de 4-bit (DB7 a DB4) cuando DL es 0. Cuando la longitud de 4-bit es seleccionado, es preciso efectuar dos operaciones en vez de una. En una primera instancia viajan los bits más significativos y, en una última instancia, viajan los bits menos significativos.

N: Setea el número de líneas en el display.

F: Setea la fuente de texto.

Nota: Realizar la función en la cabecera del programa antes de ejecutar alguna instrucción (excepto para la bandera de ocupado para leer y de la dirección de la instrucción). Desde este punto, la instrucción de seteo de la función no puede ser ejecutada a menos que la longitud de datos de la interface se cambie.

1.4. RTC (Real time clock)

1.4.1. Descripción

El DS1307 Serial Real-Time Clock es de bajo consumo, completamente en BCD (Binary-Coded Decimal), viene con una NV SRAM de 56 bytes. La dirección y los datos son transferidos vía serial en dos hilos, bus bi-direccional.

El reloj/calendario provee información de segundos minutos, horas, días, fecha, mes y año. La fecha del mes es ajustada automáticamente con los meses que tienen menos de 31 días, incluyendo correcciones de los años bisiestos. El reloj opera en formato de 24-horas o 12-horas con un indicador AM/PM. El DS1307 ha sido construido para que sea muy sensible al suministro de energía, detectando la falla de energía y cambiando de manera automática a la batería de respaldo.

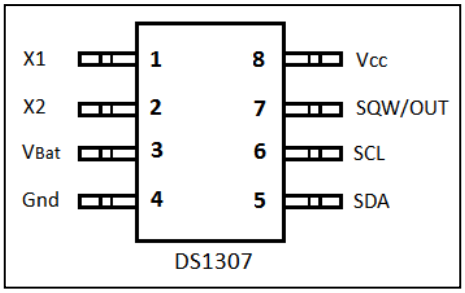

Figura 12 RTC DS1307

1.4.2. Características

- Reloj de tiempo-Real (RTC) cuenta segundos, minutos, horas, fechas del mes, día de la semana y año. Compensación válido hasta 2100
- 56-byte, batería-respaldo, RAM no volátil (NV) para almacenar los datos.

- 2 hilos de interface serial.
- Señal de salida programable para onda cuadrada.
- Detección automática de Falla de Energía.
- Consume menos de 500mA en la batería de respaldo con oscilador funcionando.
- Opcional Rango de temperatura para uso industrial:-40°C a +85°C.
- Disponible en 8-pin DIP o SOIC.

SEÑAL	DESCRIPCIÒN	ENTRADA	SALIDA
X1, X2	Cristal de 32.768Khz.	X	
VBAT	+3V batería.	X	
GND	Tierra.	X	
SDA	Dato Serial.		X
SCL	Reloj Serial.	X	
SQW/OUT	Onda Cuadrada/Controlador de salida.		X
VCC	Encendido del RTC.	X	

Tabla 4 Descripción de pines del RTC DS1307

1.4.3. Operación

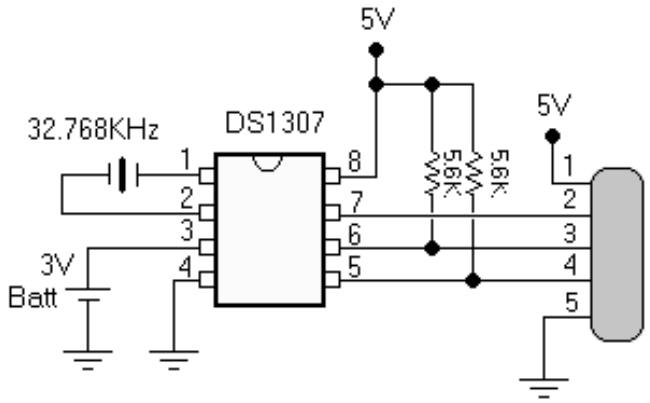

Figura 13 Esquemático del RTC

El DS1307 opera como dispositivo esclavo en el bus serial. El acceso es obtenido por la implementación de una condición START y proveyendo un código de identificación del dispositivo seguido por la dirección de registro.

Subsecuentemente los registros pueden ser accedidos de forma secuencial hasta que una condición de STOP es ejecutada. Cuando VCC cae por debajo 1.25 x VBAT el dispositivo en curso termina el acceso y restablece el contador de direcciones del dispositivo. En este momento las entradas al dispositivo no serán reconocidas para evitar que se escriban datos erróneos en el dispositivo por fuera de la tolerancia del sistema. Cuando VCC cae por debajo de VBAT el dispositivo internamente conmuta a “bajo consumo” de batería (modo de respaldo o back-up). Si al encender, el VCC del dispositivo es más grande que VBAT + 0.2V, el dispositivo conmuta de BATERIA a VCC, también reconoce si VCC es más grande que 1.25 x VBAT.

1.4.4. Señales

VCC, GND – DC

La alimentación de energía es provista al dispositivo por estos pines. VCC es una entrada de +5V. Cuando se aplica 5V con los límites normales, el dispositivo es completamente accesible y los datos pueden ser escritos y leídos. Cuando se tiene conectado una batería de 3V al dispositivo y VCC es por debajo de 1.25 x VBAT, la lectura y la escritura son deshabilitados. Sin embargo, la función de la hora normal no se ve afectada por la bajada de tensión en la entrada. Cuando VCC cae por debajo de VBAT, la alimentación de energía cambia de VCC a VBAT (nominal 3.0V DC).

VBAT

Entrada de la batería estándar de 3V, celda de litio u otra fuente de energía. El voltaje de la batería debe estar entre 2.0V y 3.5V para una operación adecuada. Una batería de litio con 48 mAhr o más grande, respaldara al DS1307 por más de 10 años en el abastecimiento de energía a 25ºC.

SCL

(Entrada del reloj serial) – SCL es usado para sincronizar los movimientos de datos de la interface serial.

SDA

(Entrada/Salida de datos) – SDA es el pin input/output de los dos alambres de la interface serial. El pin SDA es de drenador abierto el cual requiere un resistor PULL-UP.

SQW/OUT

(Square Wave/Output Driver) – Cuando está habilitado, el bit SQWE esta seteado en 1, el pin de salida SQW/OUT tendrá 4 ondas cuadradas de diferentes frecuencias (1Hz, 4kHz, 8kHz, 32kHz). El pin SQW/OUT es de drenador abierto y requiere una resistencia externa PULL-UP. SQW/OUT operara con un voltaje aplicado de VCC o Vbat.

X1, X2

Conexiones para un cristal de cuarzo estándar de 32.768kHz. El circuito del oscilador interno es diseñado para operar con un cristal teniendo una carga específica de capacitancia (CL) de 12.5pF.

Para más información de la selección del cristal y las consideraciones de diseño del cristal, por favor consulte la nota de Aplicación 58, “Crystal Considerations with Dallas Real-Time Clocks.” El DS1307 puede ser también manejado por un oscilador externo de 32.768Khz. En esta configuración, el pin X1 es conectado al oscilador externo y el pin X2 es flotante.

1.4.5. RTC Y RAM mapa de direcciones

El mapa de direcciones para los registros RTC y RAM del DS1307 se podrá ver en la TABLA 5. El registro RTC es localizado en la direcciones 00h a 07h. El registro de la RAM es localizado en las direcciones 08h a 3Fh.

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

Tabla 5 Mapa de direcciones

1.4.6. Reloj y Calendario

El tiempo y la información del calendario son obtenidas leyendo los byte del registro correspondiente. Los registros RTC son ilustrados en la TABLA 6.

El tiempo y el calendario son seteados o inicializados por escribir los bytes de registro correspondiente. El contenido de los registros del tiempo y calendario están en formato BDC. El registro del día de la semana se incrementa en la media noche. Los valores que corresponden a los días de la semana son definidos por el usuario, pero debe ser secuencial (es decir, si 1 es igual a domingo, entonces 2 es igual a lunes, y así sucesivamente). Entradas de tiempo y fecha ilógicos causará una operación indeterminada. El Bit 7 del registro 0 es para detener el reloj (Clock Halt - CH) bit. Cuando este bit se setea a 1, el oscilador esta deshabilitado. Cuando se setea a 0, el oscilador está habilitado.

Tenga en cuenta que el estado inicial (power-on) todos los registros no están definidos. Además, es importante habilitar el oscilador (CH bit = 0) durante la configuración inicial.

El DS1307 puede ser corrido en 12horas o 24horas. En el Registro de la Hora (02H) el Bit 6 define el modo de trabajo (12H o 24H). Cuando el Bit6=1, el modo de 12horas es seleccionado. Y con el Bit5 podremos configurar si es AM o PM, si el Bit 5 =1 es PM. En modo de 24horas, el bit 5 es el segundo bit de la decena, y el Bit4 es el primer bit de la decena; por ejemplo si queremos setear 21H00(modo 24H), debemos ingresar, 0010-0001. Si queremos setear 11H00-AM, ingresaríamos 0101-0001.

En el tiempo actual es transferido un segundo registro. La información del tiempo es leída de ese registro secundario, mientras el reloj continua corriendo. Esto elimina la necesidad de re-leer los registros en caso de actualización del registro principal durante una lectura.

Registro	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Rango
		DECENAS			**UNIDADES**				
00H	CH	10’ SEGUNDOS			SEGUNDOS				00-59
01H	0	10’ MINUTOS			MINUTOS				00-59
02H	0	24 12	10’HR AM/PM	10’HR	HORAS				1-12 AM/PM 00-23
.	0	0	0	0	0	DIA			01-07
.	0	0	10’ FECHA		FECHA				01-31
.	0	0	0	10’ MES	MES				01-12
.	10’ AÑO				AÑO				00-99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	
10’ = decenas									

Tabla 6 Registros del cronometro

1.4.7. REGISTRO DE CONTROL

El registro de control del DS1307 (registro 07H) es usado para el control de operación del pin SQW/OUT.

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
OUT	0	0	SQWE	0	0	RS1	RS0

Tabla 7 Control de registro RTC

OUT (Control de salida)

Este bit (bit7 del registro 07H) controla el nivel de salida del pin SQW/OUT (pin7 del DS1307) siempre y cuando la onda cuadrada de salida es deshabilitada (SQWE = 0). SQW/OUT = 1 si OUT = 1 y es 0 si OUT = 0.

SQWE (Onda cuadrada habilitada)

Este bit, cuando se setea a 1, habilitará la salida del oscilador. La frecuencia de la onda cuadrada de salida dependerá de los valores de los bits RS0 y RS1. Con la onda cuadrada de salida de 1 Hz, los registros del reloj son actualizados en el flanco de bajada de la onda cuadrada.

RS (seleccionar rango)

Estos bit de control son los que dan la frecuencia a la salida de la onda cuadrada cuando SQW=1 (habilitada). La lista de las frecuencias de la onda y pueden ser seleccionas con los bit RS como se ver TABLA 8.

RS1	RS0	SQW OUTPUT FRECUENCIA
0	0	1Hz
0	1	4.096Khz
1	0	8.192Khz
1	1	32.768Khz

Tabla 8 Frecuencia de salida SQW

SOFTWARE

1.5. Quartus II

1.5.1. Descripción

Es una herramienta de software producida por Altera para el análisis y la síntesis de diseños realizados en plataformas de dispositivos lógicos programables.

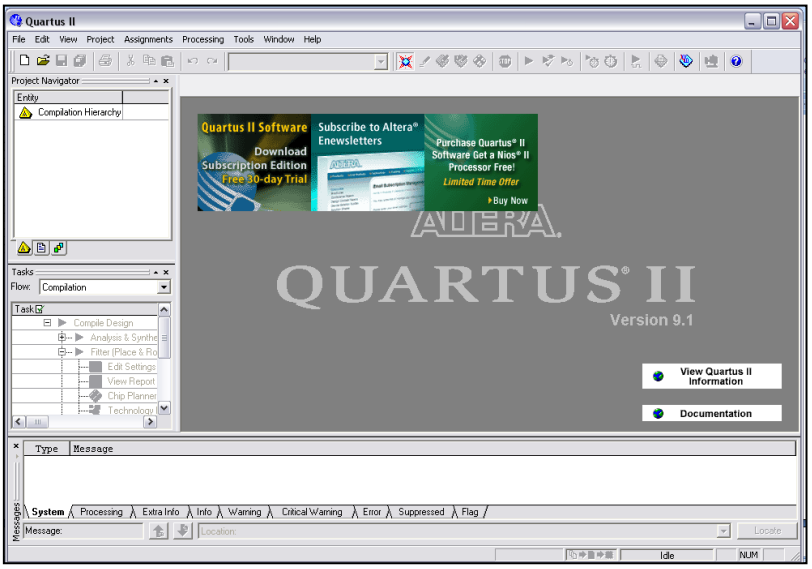

Figura 14 Quartus II

Quartus II permite al desarrollar compilar sus diseños, realizar análisis de tiempo, examinar diagramas rtl y configurar el dispositivo lógico programable con el programador.

1.5.2. Características principales

- Diseño de flujo.
- Diseño de entidades.
- Síntesis.
- Optimización de diseño y análisis de tiempo.

- Programación y configuración rápida en el dispositivo.
- Soporta herramientas EDA.
- Permite la depuración de nuestro diseño.

1.5.3. Ventajas

- Posee editor de lenguaje simbólico.
- Posee un editor de formas de onda que puede ser utilizado para describir un sistema digital, como para determinar las entradas en una simulación.
- Posee un compilador que es capaz de recibir como entrada tanto archivos con la descripción del sistema (VHDL, VERILOG), como archivos con circuitos esquemáticos. Puede ejecutar simulaciones, usando como archivo de entrada uno creado por el editor de formas de onda.
- Puede programar un dispositivo CPLD o FPGA (primero es necesario especificar el modelo y los pines de entrada y salida que se usaran).

1.5.4. Desventajas

- Costo de licencias.
- Algunas librerías importantes tienen costo, no se las puede utilizar sin pagar.
- El simulador integrado para la depuración de código no es muy bueno, se debe optar por escoger otro software especialista en esa línea.

- Las máquinas de estado generadas, solo se presentan como máquinas de estado finita, para una mejor comprensión a veces se necesita de una maquina de estado algorítmica.
- No se puede compilar código de otro fabricante, que no sea altera.

1.6. Signaltap II

1.6.1. Descripción

Es un analizador lógico que se encuentra en Quartu II, que se utilizó para evaluar el estado de las señales en el dispositivo, lo que nos permite encontrar la causa de defectos en el diseño del sistema.

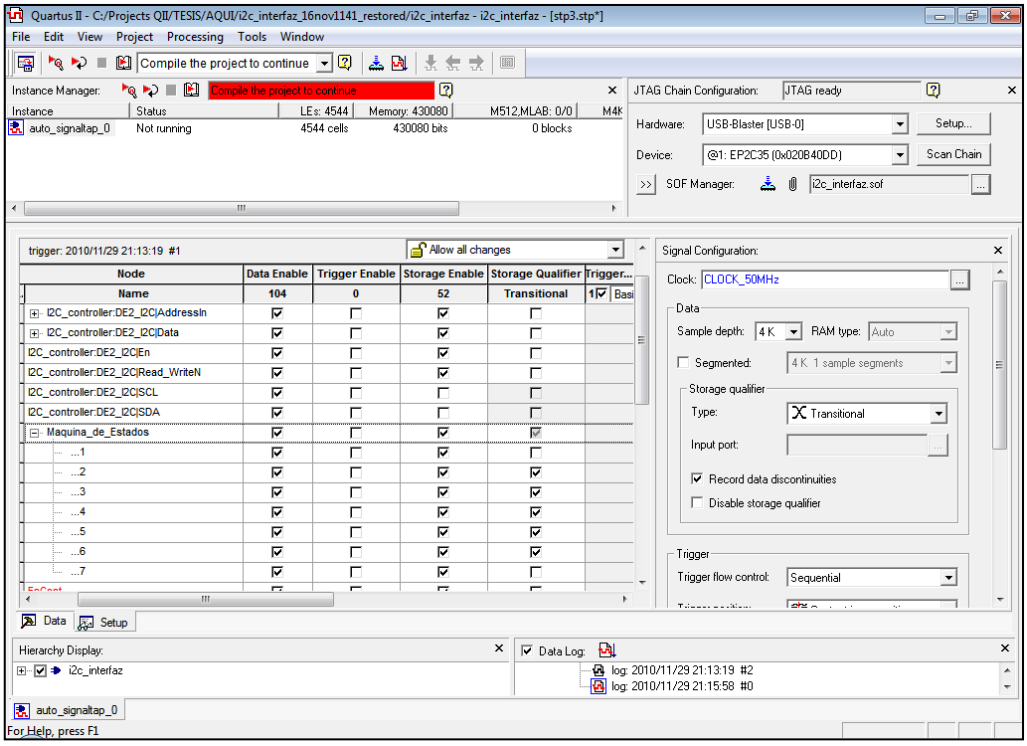

Figura 15 Signaltap II

1.6.2. Ventajas

- Permite encontrar la causa de defectos en el diseño del sistema.

- Permite depurar nuestro código rápidamente, revisando el estado de la señal que estamos analizando, en cualquier instante de tiempo que queramos.
- Permite crear condiciones complejas (triggers) a nuestras señales, para análisis más complejos
- Permite crear nemónicos a nuestras maquinas de estado para mejor visualización y depuración de nuestro proyecto.

1.6.3. Desventajas

- Tiene ligeros retardos en la representación de las señales.
- A medida que el análisis va creciendo, se necesita de mayor recurso y tiempo para la compilación de nuestro proyecto y mayor memoria flash en nuestro circuito.

1.7. Altium

1.7.1. Descripción

Altium Designer summer 08: Es una herramienta utilizada para el diseño electrónico en todas sus etapas, captura de esquemáticos, simulación, diseño de circuitos impresos, implementación de FPGA.

Figura 16 Altium Designer

1.7.2. Características

Diseño de PCB

Altium Designer ha unificado el diseño de la plataforma física con el diseño de tarjeta de circuito impreso, con soporte para lógica programable. Esto proporciona un sistema de desarrollo totalmente unificado que puede desplegarse a través de todos los elementos del proceso de diseño de productos electrónicos.

Gestión de librerías

Elegir un componente obsoleto o fuera de stock puede dar lugar a la producción de largos retrasos y sobrecostos. Altium Designer ofrece una amplia gama de herramientas de gestión de datos y recursos de información que le permiten mantener el control sobre el uso de partes.

Diseño para fabricación

Altium Designer ayuda a reducir la brecha entre el diseño y fabricación, y le permite administrar de forma activa la generación y

verificación de todos los datos de fabricación, ahorrando tiempo y minimizando los costosos errores durante el flujo de diseño.

Dispositivos programables

Altium Designer es la única herramienta que permite explotar plenamente el potencial que ofrecen hoy los dispositivos programables, al unificar el diseño de FPGAs, el desarrollo de software y el diseño de PCBs, en una sola plataforma.

Diseño Unificado de FPGA/PCB

Los dispositivos programables son cada vez más usados en desarrollo electrónico. Este tipo de dispositivos abren nuevas posibilidades de diseño y aportan beneficios significativos para el proceso de diseño, lo que permite que la complejidad funcional se traslade de dispositivos discretos cableados al ámbito de dispositivos programables**.**

Gestión de todo el proceso de desarrollo

Altium Designer unifica todo el proceso de diseño y permite gestionar todos los aspectos del desarrollo dentro de un único entorno de diseño, y ofrece una infraestructura de administración de proyectos y documentos**.**

1.7.3. Ventajas

- Permite el ruteo automático y manual de pistas.
- Posee un simulador para el análisis de nuestro circuito.
- Soporta VHDL.
- Es compatible con ALTERA

- Tiene un buen visualizador 3D para mostrarnos como va a quedar nuestro circuito impreso y armado.
- Soporta librerías de otros fabricantes para los diferentes componentes en nuestro PCB.

1.7.4. Desventajas

- No posee análisis térmico para el PCB.
- No posee análisis electromagnético para el PCB.
- No posee análisis mecánico para el PCB.

Capítulo 2

2. Diseño

2.1. Introducción

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.

2.2. Diseño de IPCORE I2C MAESTRO

Diagrama de Bloque del IPCORE I2C MAESTRO

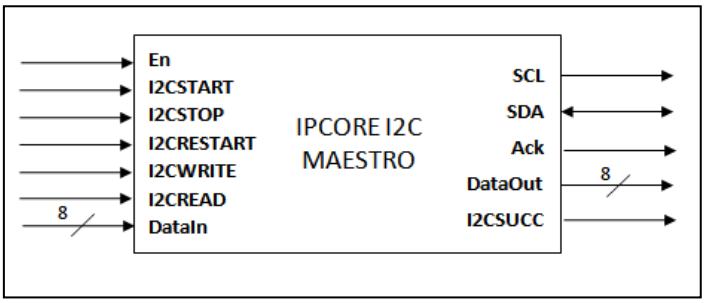

Figura 17 Diagrama del bloque de ipcore I2C maestro

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 entrada 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.

Diagrama ASM del IPCORE I2C MAESTRO

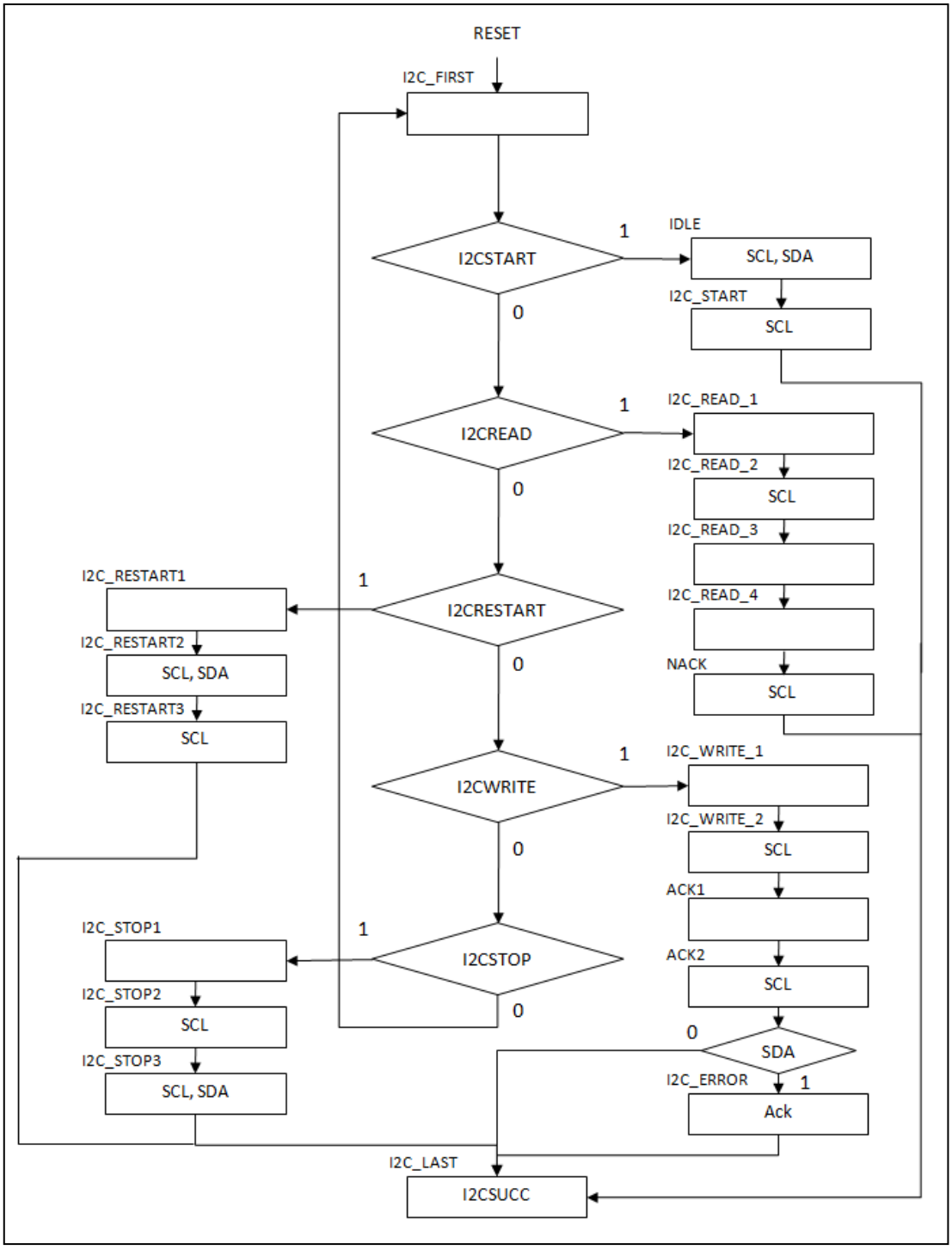

Figura 18 Diagrama asm del ipcore I2C maestro

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 19.

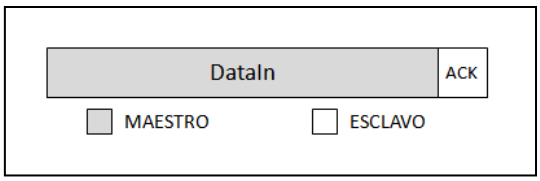

Figura 19 Transferencia de dato

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 20.

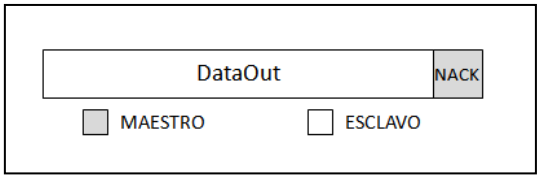

Figura 20 Recepción de dato

2.3. Diseño del MÓDULO RTC ESCLAVO

Diagrama de Bloques del MÓDULO RTC ESCLAVO

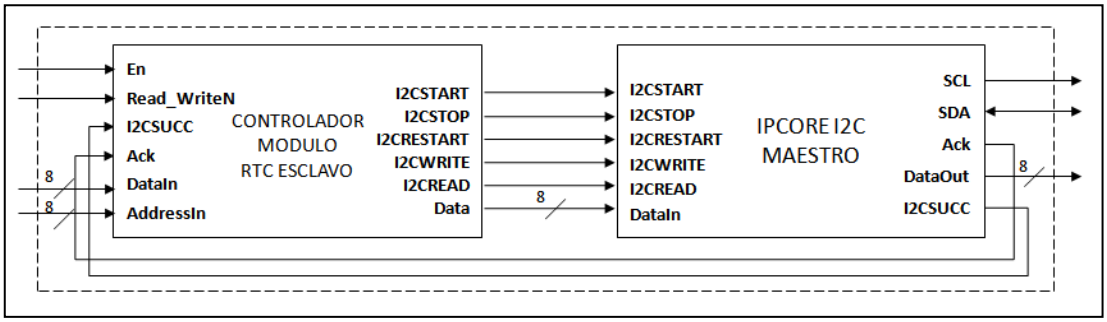

Figura 21 Diagrama de bloque del módulo RTC esclavo

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.

Diagrama ASM del circuito controlador del MÓDULO RTC ESCLAVO

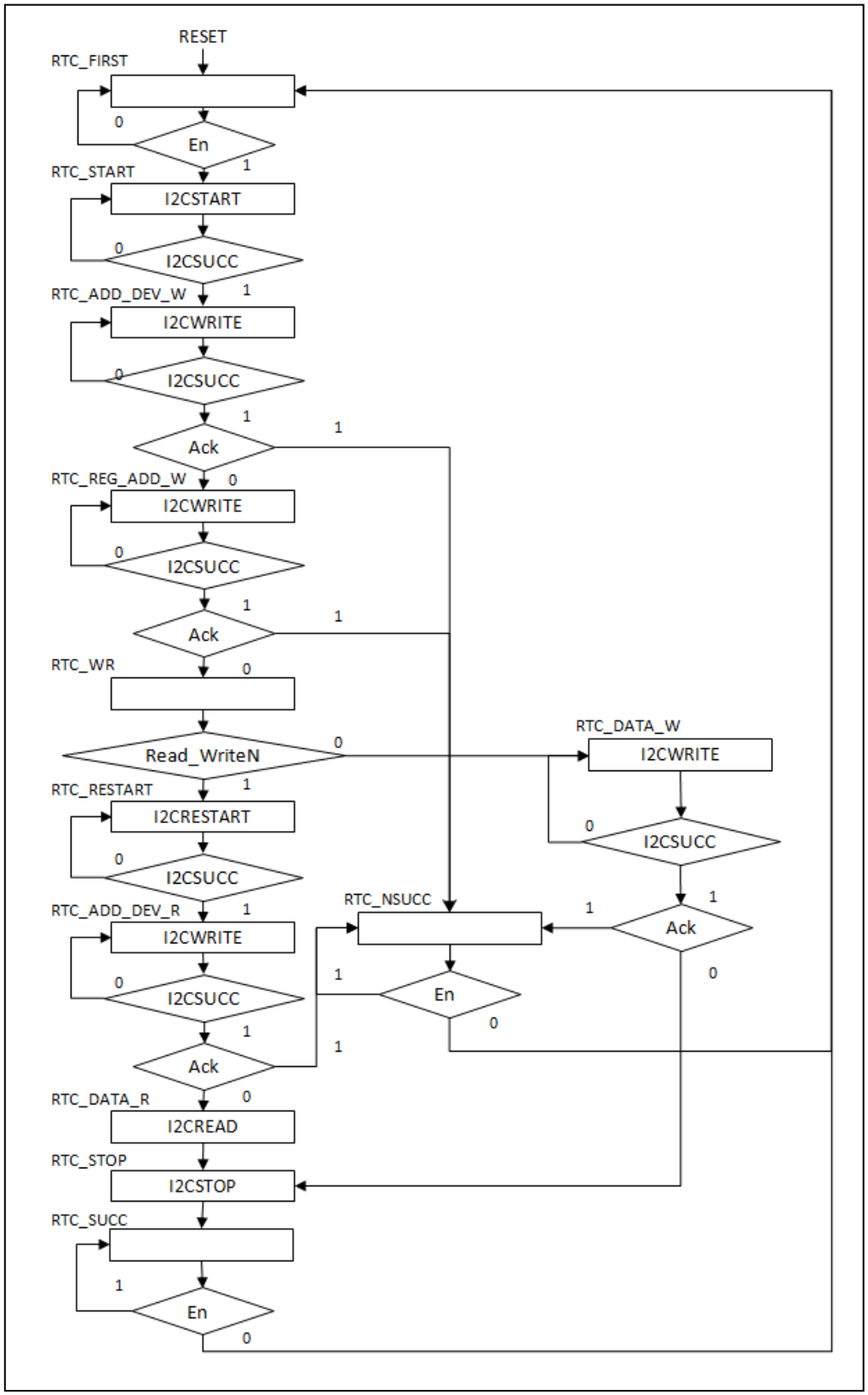

Figura 22 Diagrama asm circuito controlador del módulo RTC esclavo

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 5.
- Leer ACK enviado por el RTC.
- Escribir dato al RTC.
- Leer ACK enviado por el RTC.
- Enviar condición STOP. Ver Figura 23.

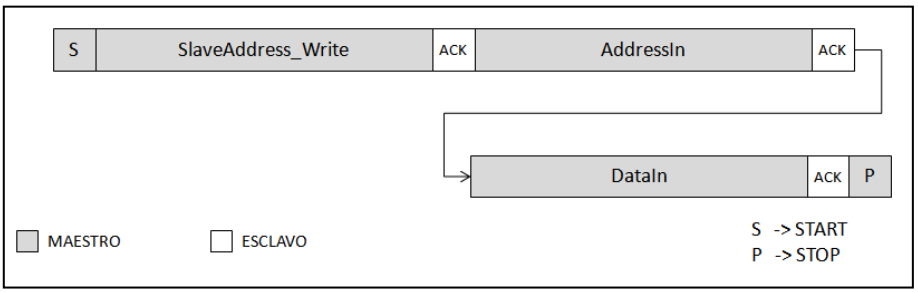

Figura 23 Trama de escritura al RTC DS1307

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 24.

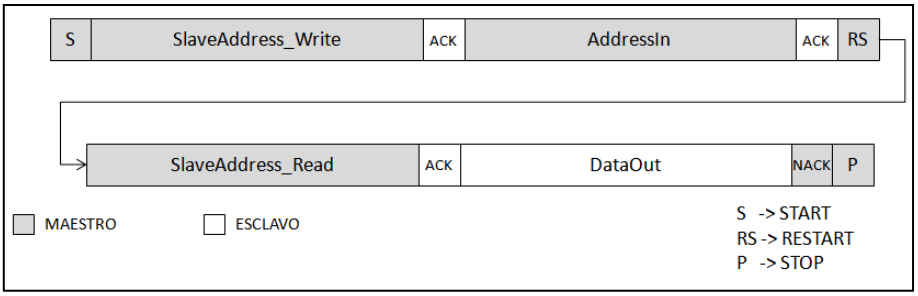

Figura 24 Trama de lectura al RTC DS1307

También puede ver la trama de lectura en el diagrama de tiempos en la Figura 48.

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.

2.4. Diseño del MÓDULO LCD

Diagrama de Bloque de la Interfaz LCD HITACHI

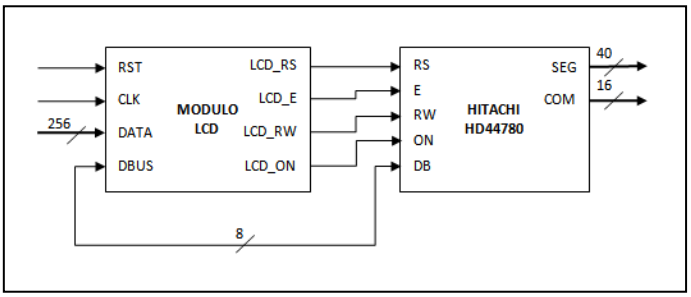

Figura 25 Diagrama de bloque de la interfaz LCD HITACHI

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.

2.5. Diseño de la Aplicación Ejemplo

Diagrama de Bloque de la Aplicación Ejemplo

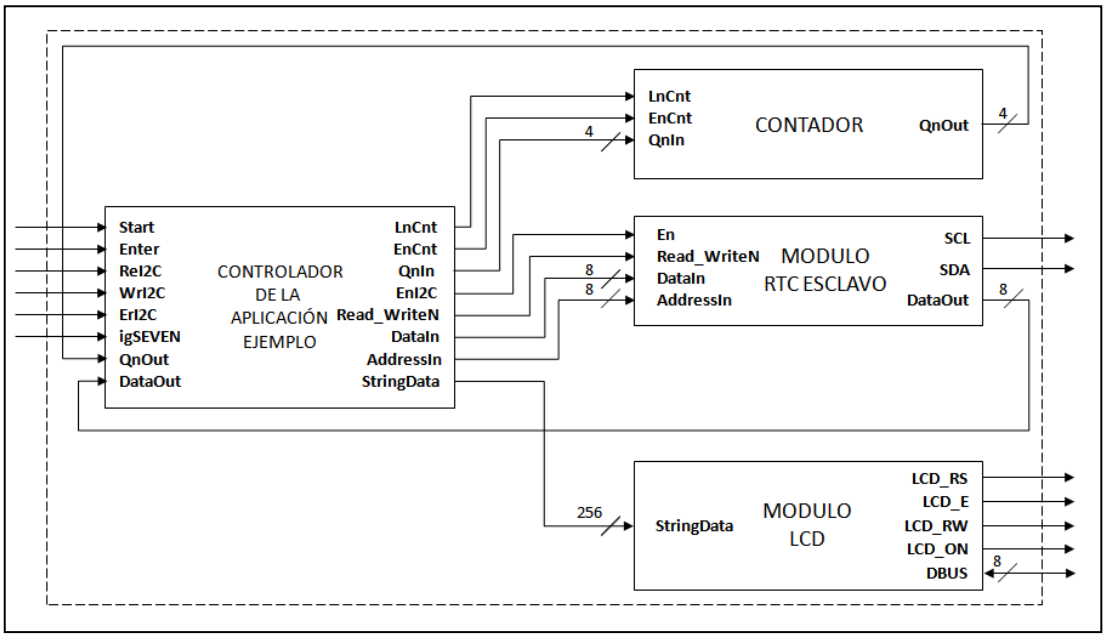

Figura 26 Diagrama de bloque de la aplicación ejemplo

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.

Diagrama ASM del circuito controlador de la Aplicación Ejemplo

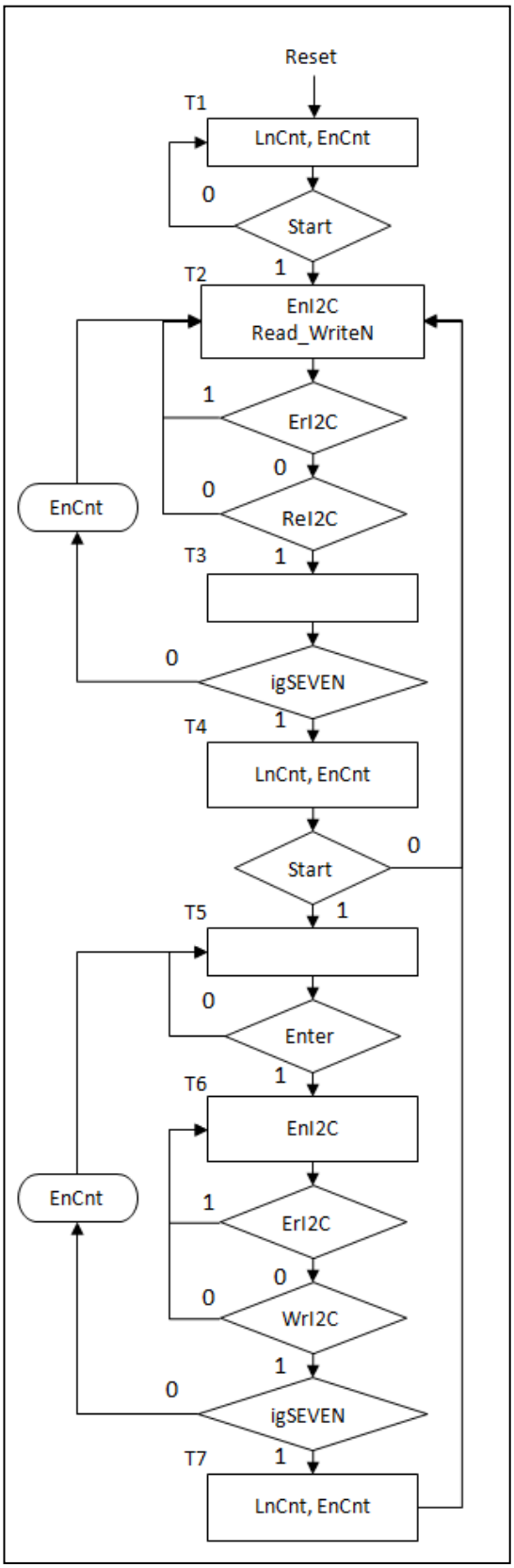

Figura 27 Diagrama asm del circuito controlador de la aplicación ejemplo

Capítulo 3

3. Implementación

3.1. Introducción

Para una mejor comprensión de nuestro proyecto, presentamos una analogía con el modelo OSI

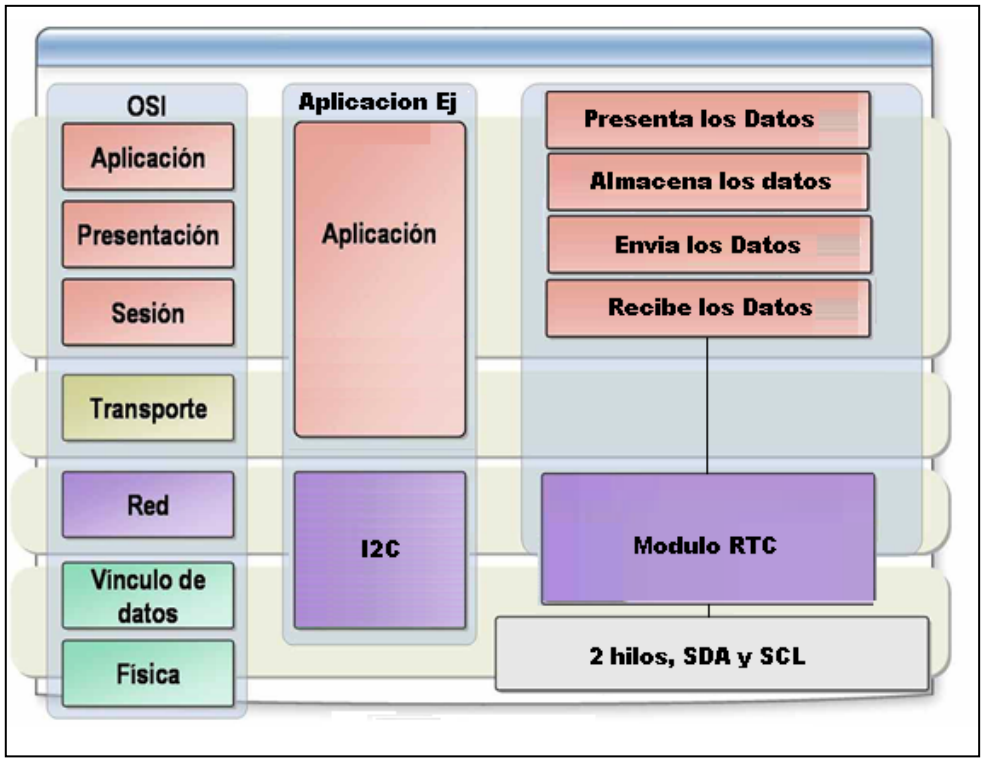

Figura 28 Capas de la aplicación ejemplo

El modelo de 7 capas de OSI, lo resumimos en solo dos capas.

La capa de aplicación

Esta capa se encarga de presentar los datos en el display, almacena los datos que el usuario ingresa, envía los datos al módulo RTC y recibe datos

del módulo RTC. Además esta capa se encarga de filtrar datos correctos al módulo RTC.

LA CAPA I2C

Esta es la capa que se encarga de manejar todo el protocolo de comunicación con el mundo exterior. En la aplicación Ejemplo esta tarea lo realiza el Modulo RTC, que se conecta con el IPCORE MAESTRO (I2C).

El Modulo RTC es el encargado de recibir el dato y dirección de registro (la dirección del dispositivo esclavo ya esta seteada en este módulo).

La capa Física

La capa física está hecha solo con dos hilos: uno para el clock (SCL) y otro para el dato (SDA), estas dos líneas están conectadas con resistencias pull-up.

3.2. Implementación de IPCORE I2C MAESTRO

Para un correcto funcionamiento del módulo se debe resetear las señales; 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 29.

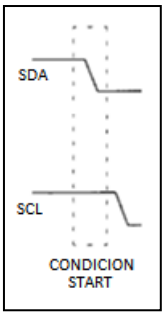

Figura 29 Transición de condición START

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 30.

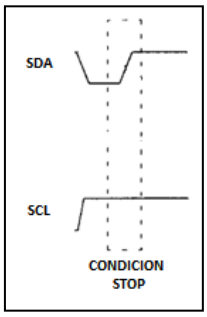

Figura 30 Condición STOP

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 31.

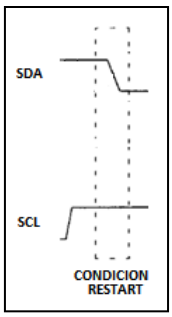

Figura 31 Condición RESTART

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 32.

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.

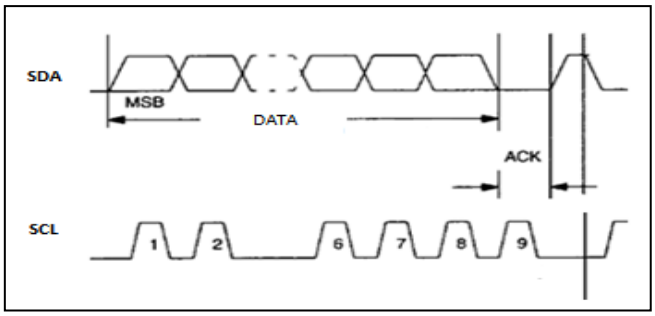

Figura 32 Escritura en esclavo

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 al ESCLAVO y envía en el bit NACK. Ver Figura 33.

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.

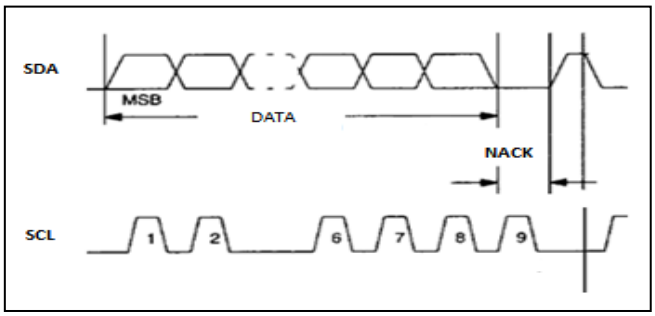

Figura 33 Lectura de esclavo

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.

3.3. Implementación del 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 y coloca en el CORE el correspondiente dato de entrada a la sección de la trama. Ver Figura 21.

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 del MÓDULO LCD

Seteo de función

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Display
SETEO DE FUNCION										
0	0	0	0	1	1	1	1	*	*	

Figura 34 Seteo de función

A continuación mostramos la implementación en código

```
WHEN FUNC_SET =>
        LCD_E <= '1';
        LCD_RS <= '0';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= X"3C";
        state <= DISPLAY_OFF;
```

Apagado de Pantalla

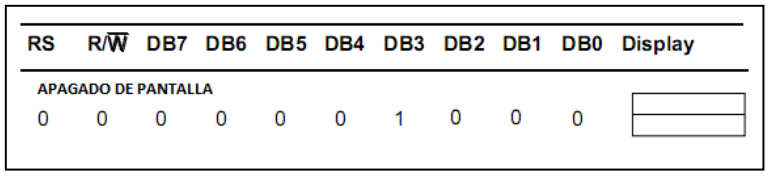

Figura 35 Apagado de pantalla

A continuación mostramos la implementación en código

```
WHEN DISPLAY_OFF =>
        LCD_E <= '1';
        LCD_RS <= '0';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= X"08";
        state <= DISPLAY_CLEAR;
```

Limpiar Pantalla

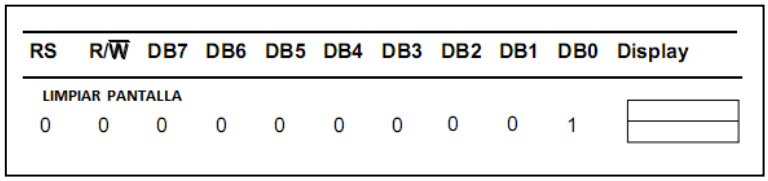

Figura 36 Limpiar pantalla

A continuación mostramos la implementación en código

```
WHEN DISPLAY_CLEAR =>
        LCD_E <= '1';
        LCD_RS <= '0';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= X"01";
        state <= DISPLAY_ON;
```

Encender Pantalla

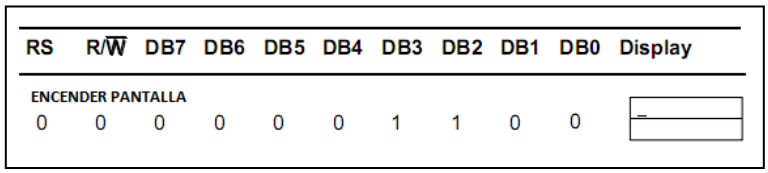

Figura 37 Encender pantalla

A continuación mostramos la implementación en código

```
WHEN DISPLAY_ON =>
        LCD_E <= '1';
        LCD_RS <= '0';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= X"0C";
        state <= MODE_SET;
```

Modo de Entrada

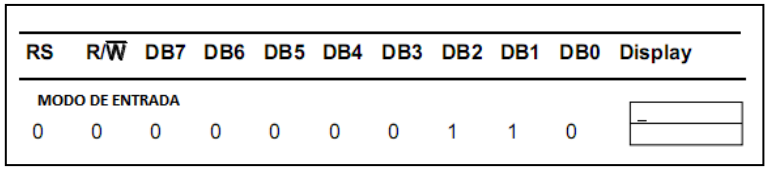

Figura 38 Modo de entrada

A continuación mostramos la implementación en código

```
WHEN MODE_SET =>
        LCD_E <= '1';
        LCD_RS <= '0';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= X"06";
        state <= WRITE_CHARL10;
        bytecount <= 255;
```

Escribir caracter en la primera línea

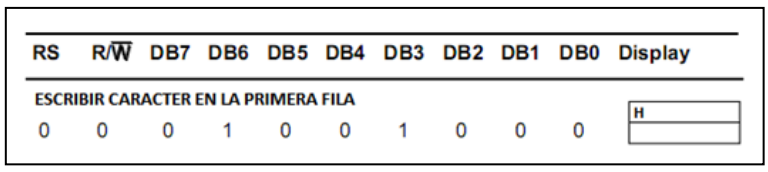

Figura 39 Escribir caracter en la primera línea

A continuación mostramos la implementación en código

```
WHEN WRITE_CHARL10 =>
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= stringdata(bytecount DOWNTO bytecount-7);
        IF (bytecount - 8) >= 128 THEN
                bytecount <= bytecount - 8;
                state  <= WRITE_CHARL10;
        ELSE
                bytecount <= 127;
                state  <= NEXT_LINE;
        END IF;
```

Cambia de línea

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Display
CAMBIO DE LINEA										
0	0	0	0	0	0	0	0	1	1	H

Figura 40 Cambio de línea

A continuación mostramos la implementación en código

```
WHEN NEXT_LINE =>
        LCD_E <= '1';
        LCD_RS <= '0';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= X"03";
        state <= ADDR_LINE_TWO;
```

Colocar cursor en la parte inferior izquierda de la pantalla

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Display
COLOCAR CURSO EN PARTE INFERIOR IZQUIERDA DE LA PANTALLA										
0	0	1	1	0	0	0	0	0	0	H _

Figura 41 Colocar cursor en la parte inferior izquierda de la pantalla

A continuación mostramos la implementación en código

```
WHEN ADDR_LINE_TWO =>
        LCD_E <= '1';
        LCD_RS <= '0';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= X"C0";
```

```
state <= WRITE_CHARL20;
bytecount <= 127;
```

Escribir caracter en la segunda línea

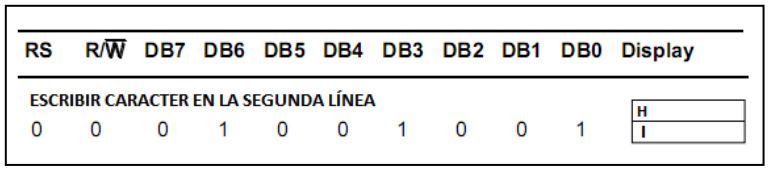

Figura 42 Escribir caracter en la segunda línea

A continuación mostramos la implementación en código

```
WHEN WRITE_CHARL20 => -- WRITE_LINE2;
        LCD_E <= '1';
        LCD_RS <= '1';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= stringdata(bytecount DOWNTO bytecount-7);
        IF (bytecount - 8) >= 0 THEN
                bytecount <= bytecount - 8;
                state <= WRITE_CHARL20;
        ELSE
                bytecount <= 255;
                state <= RETURN_HOME;
        END IF;
```

Retorna a inicio

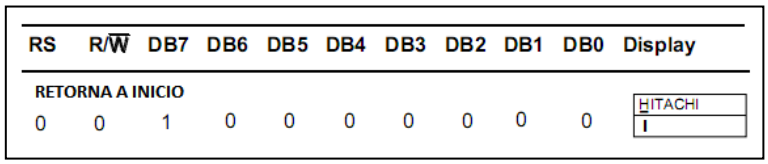

Figura 43 Retorna a inicio

A continuación mostramos la implementación en código

```
WHEN RETURN_HOME =>
        LCD_E <= '1';
        LCD_RS <= '0';
        LCD_RW <= '0';
        DATA_BUS_VALUE <= X"80";
        state <= WRITE_CHARL10;
```

3.5. 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 4445 se observa la salida en la LCD.

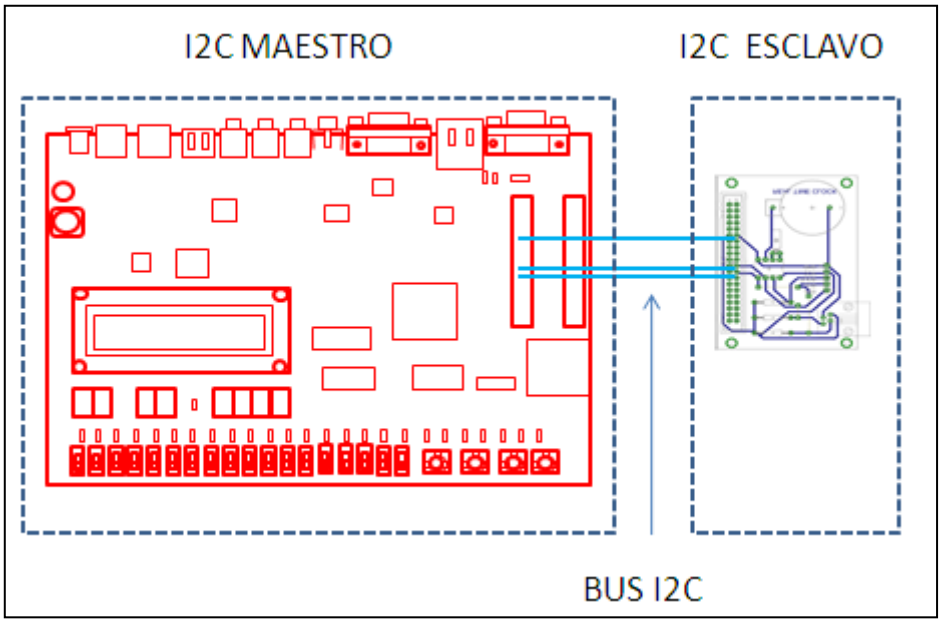

Figura 44 Implementación de la aplicación ejemplo

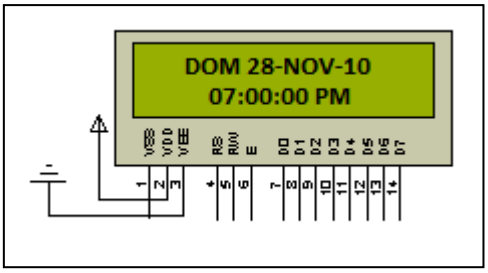

Figura 45 Fecha y hora visualizada en LCD

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 continúa 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 48, pag 74.

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 49, pag74.

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 por 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);
```

3.6. Asignación de Pines

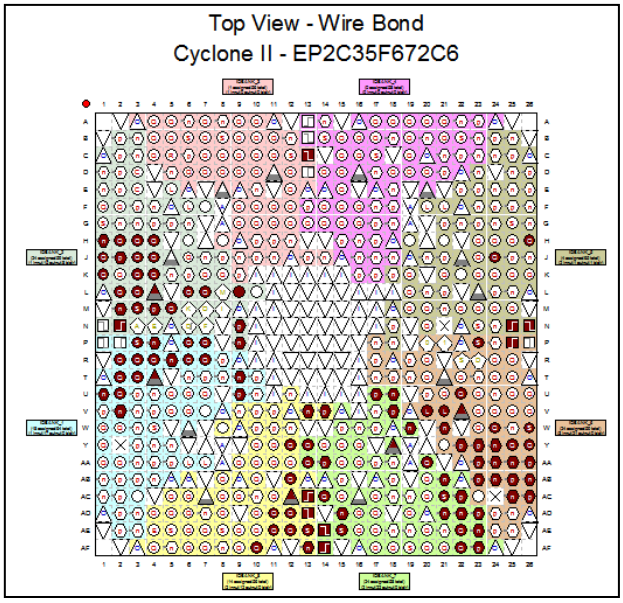

Figura 46 Mapa de pines del Cyclone II EP2C35F672C6

SEÑAL	FPGA Pin No.	DESCRIPCION
CLOCK_50MHz	PIN_N2	Reloj general del sistema
Start	PIN_W26	Botonera para START
Reset	PIN_V2	Botonera para RESETEO
Enter	PIN_P23	Botonera para ENTER
SDA	PIN_J24	Bus serial de datos – I2C
SCL	PIN_H26	Bus serial de clock –I2C
DATO[7]	PIN_C13	Datos de entrada MSB
DATO[6]	PIN_AC13	Datos de entrada
DATO[5]	PIN_AD13	Datos de entrada
DATO[4]	PIN_AF14	Datos de entrada
DATO[3]	PIN_AE14	Datos de entrada
DATO[2]	PIN_P25	Datos de entrada
DATO[1]	PIN_N26	Datos de entrada
DATO[0]	PIN_N25	Datos de entrada LSB
L_E	PIN_K3	Habilitador para R/W – LCD
L_ON	PIN_L4	Encendido de LCD – LCD
L_RS	PIN_K1	Selector de registro - LCD
L_RW	PIN_K4	Lectura/ escritura – LCD
DATA_BUSP[7]	PIN_H3	Bus de datos a la LCD MSB
DATA_BUSP[6]	PIN_H4	Bus de datos a la LCD
DATA_BUSP[5]	PIN_J3	Bus de datos a la LCD
DATA_BUSP[4]	PIN_J4	Bus de datos a la LCD
DATA_BUSP[3]	PIN_H2	Bus de datos a la LCD
DATA_BUSP[2]	PIN_H1	Bus de datos a la LCD
DATA_BUSP[1]	PIN_J2	Bus de datos a la LCD
DATA_BUSP[0]	PIN_J1	Bus de datos a la LCD LSB

Tabla 9 Asignación de pines del Cyclone II Ep2c35f672c6

Síntesis

Flow Status	
Flow Status	Successful - Mon Nov 29 03:08:52 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Web Edition
Revision Name	i2c_interfaz
Top-level Entity Name	i2c_interfaz
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	No
Total logic elements	2,200 / 33,216 (7 %)
Total combinational functions	1,483 / 33,216 (4 %)
Dedicated logic registers	1,356 / 33,216 (4 %)
Total registers	1356
Total pins	110 / 475 (23 %)
Total virtual pins	0
Total memory bits	430,080 / 483,840 (89 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Tabla 10 Resumen de síntesis

3.7. Diagramas de Tiempo de la Aplicación Ejemplo

En la Figura 47 mostramos la recepción de datos desde RTC.

En la Figura 48 se muestra un zoom la caja blanca de la Figura 47 con la trama de lectura de los minutos.

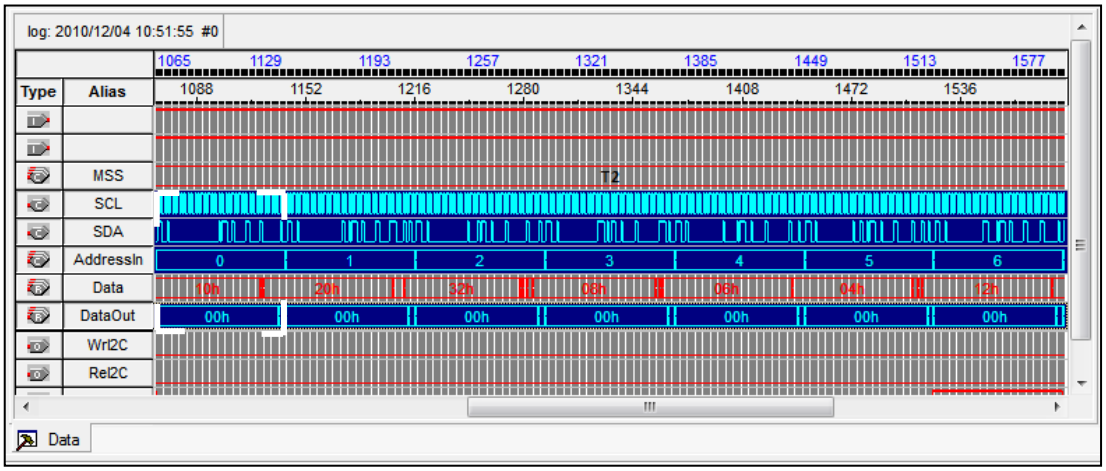

Figura 47 Lectura de fecha y hora

En la Figura 48 vemos con detalle todo el proceso de captura de datos por parte el MAESTRO. En la primera parte el MAESTRO inicia la comunicación con un START, luego envía la dirección del RTC con el último bit en ‘0’; b“1101-000[0]”; inmediatamente el RTC responde con un ACK = 0, indicando que si escucho el llamado; luego el MAESTRO envía la dirección del registro que desee obtener del esclavo; “0000-0000” (registro = x“01”, corresponde a minutos); nuevamente el esclavo responde con un ACK = 0, indicando que si escucho el llamado.

En la segunda parte, el MAESTRO manda un RESTART y ahora, envía la dirección del dispositivo del RTC con el último bit en ‘1’; b“1101-000[1]”; inmediatamente el RTC responde con un ACK = 0, posterior a esto el esclavo envía los datos solicitado; b“0011-0011” (50 MINUTOS); y el

MAESTRO finalmente manda un NACK y termina la comunicación con un STOP.

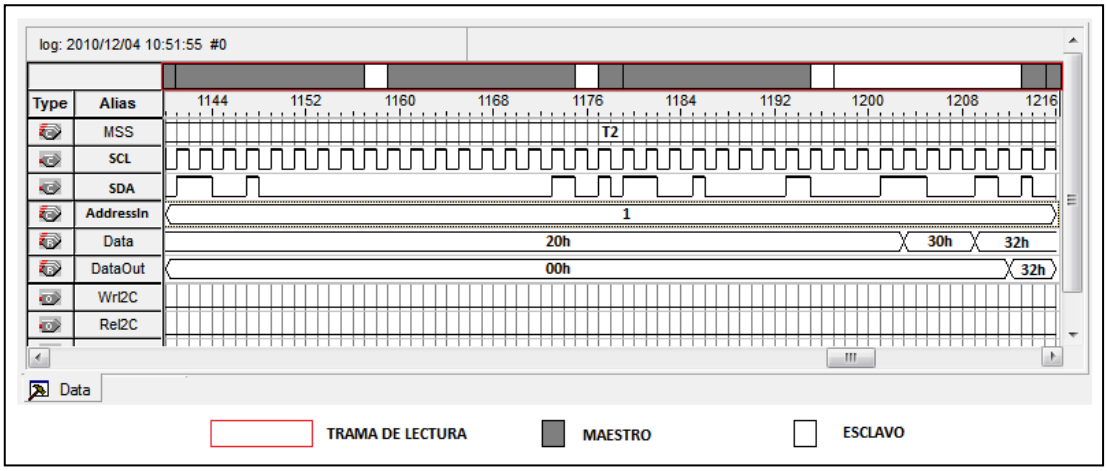

Figura 48 Trama de lectura

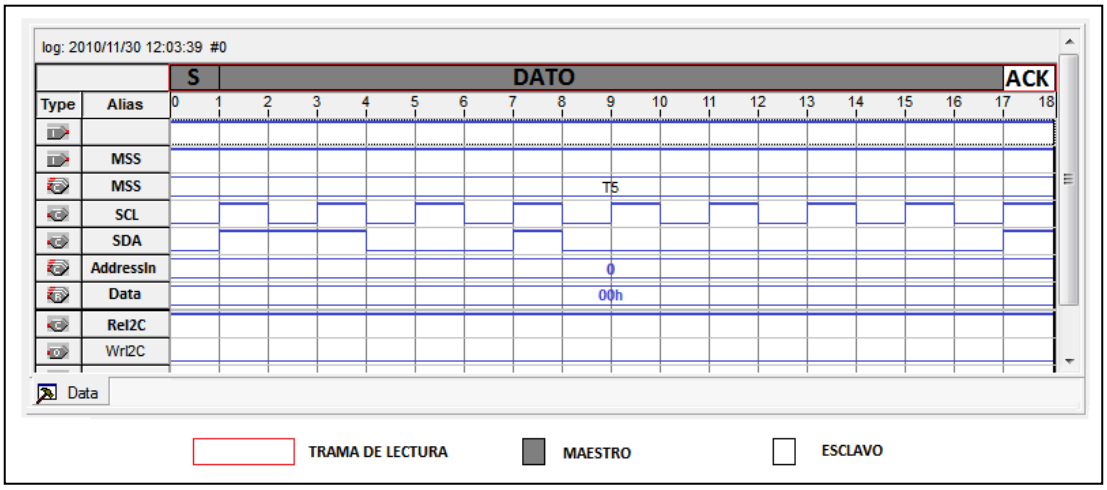

Figura 49 Ack = '1'

3.8. PCB en Altium Designer Summer 08

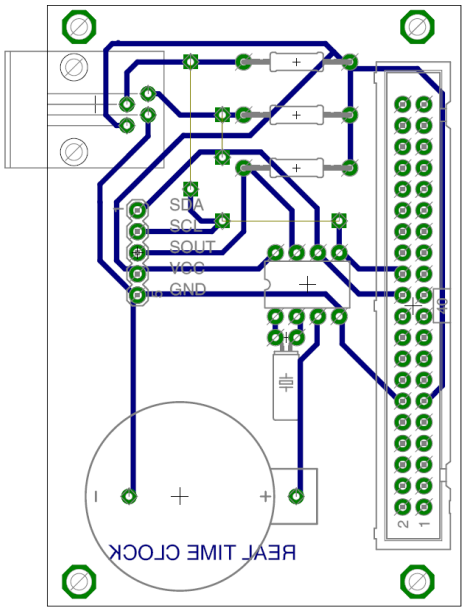

Figura 50 Pcb RTC (real time clock)

3.9. Observaciones

- En el diagrama de tiempos obtenido con SIGNALTAP II (nuestro analizador) pudimos comprobar que las graficas que nos presenta, no son en TIEMPO REAL; las señales sufren un retraso y/o adelanto. A continuación detallamos lo ocurrido con las condiciones de START, STOP y RESTART; en el caso de un START y de una STOP continuos lo que hace SIGNALTAP, es representarlo en un solo ciclo, como lo vemos en la figura 32; para la condición de RESTART ocurre algo similar SIGNALTAP II muestra de forma “comprimida” en un ciclo lo que debería ser ciclo y medio como se observa en la figura 12.
- A continuación del byte de dato MAESTRO-ESCLAVO debe enviarse ACK de respuesta del ESCLAVO indicando que recibió el dato; SIGNALTAP II visualiza en el diagrama de tiempos en el noveno ciclo

de reloj del SCL lo que se esperaba observar debido a que solo le toma un ciclo entero y no fraccionario.

- A continuación del byte de dato ESCLAVO-MAESTRO debe enviarse NACK por el MAESTRO para indicar fin de la LECTURA; SIGNALTAP II visualiza en el diagrama de tiempos en el noveno ciclo de reloj del SCL lo que se esperaba observar debido a que solo le toma un ciclo entero y no fraccionario.
- En la LECTURA de datos MAESTRO-ESCLAVO el comportamiento es como lo esperábamos de acuerdo al estándar de PHILLIPS.
- En la ESCRITURA de datos MAESTRO-ESCLAVO el comportamiento es como lo esperábamos de acuerdo al estándar de PHILLIPS.
- El dispositivo usado en la demostración de la funcionalidad de nuestro IPCORE I2C fue el Real Time Clock DS1307, el cual es de uso sencillo fácil de configurar pero tiene un comportamiento anómalo que no se encuentra detallado en el datasheet; permitiendo setear valores fuera de los rangos reales para segundos, meses, etc. Se probo ingresando como dato 60 en el registro 00 que es de segundos y comprobamos que el RTC almacena el valor y continua contando los segundos de la siguiente forma: 60, 61, 62, ..., 80, 81, 82, ..., 40; normalizándose después de el ultimo valor descrito.

Conclusiones y Recomendaciones

Las conclusiones son:

1. La utilización 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.

Las recomendaciones son:

1. Fijarse que la frecuencia de reloj sean las mismas de todos los dispositivos que se encuentren conectados al bus i2c.
2. Verificar que todos los dispositivos conectados al bus no sobrepasen los 400pf, eso podría causar perdida de datos o que los dispositivos no respondan.
3. Consultar el anexo en caso de atrasos, adelantos, paradas en el RELOJ (RTC).
4. Consultar el manual del dispositivo esclavo que se va a conectar para saber los pasos que debe seguir para abrir la comunicación y así desarrollar su controlador i2c para ese dispositivo.
5. Verificar que las líneas SDA y SCL no estén sueltas en la placa, para evitar fallas en la comunicación.
6. Desarrollar una configuración MULTI-MAESTRO, para más de un MAESTRO en el bus.
7. Desarrollar una plataforma para que opere también como esclavo.
8. Desarrollar un pre escalador para la configuración de relojes de distinta velocidad, dependiendo del dispositivo esclavo que se vaya a conectar.
9. Desarrollar una compatibilidad Wishbone, para interconectarlo con otros componentes, como por ejemplo un micropocesador.
10. Desarrollar detección de errores.
11. Desarrollar control de flujo.

Anexos

Solución de problemas del RTC

Esta sección es un resumen de las causas más frecuentes de la inexactitud del RTC. Esta sección se ha dividido en tres partes.

La primera parte se consideran los factores que causan que un RTC corra muy rápido y la segunda parte se considerara los factores que causan que un RTC corra muy lento. En la tercera parte se abordara las causas por las que un RTC se queda inhibido.

Relojes rápidos

Los siguientes argumentos, son los más comunes que causan que un RTC basado en un cristal corra rápido.

1. El acoplamiento de ruido en el cristal por señales adyacentes, usualmente causa que el RTC sea inexacto.

2. Cristal incorrecto. Un RTC normalmente correrá rápido, si se utiliza un cristal con una capacidad de carga específica (CL) más grande que la especificada por el RTC (12,5 pF). La gravedad de la falta de precisión depende del valor de CL.

Por ejemplo, si utilizamos un cristal con un CL de 12 pF en un RTC diseñado para 6pF, causara que el RTC corra unos 3-4 minutos por mes rápido.

Relojes lentos

Los siguientes son los escenarios más comunes que causan que un RTC basado en un cristal corra lento.

1. Sobreimpulso (overshoot) en los pines de entrada del RTC. Una posible causa es deteniendo periódicamente el oscilador causando el acoplamiento de ruido en la señal de entrada del RTC.

Otra situación que es común para el sobreimpulso, es tener el suministro de energía del RTC en 5 voltios cuando el RTC está en el modo de respaldo de batería. Esto causa que algunos RTC se apaguen automáticamente. Es muy importante asegurarse de que la alimentación de energía del RTC no sea mayor que el voltaje de batería cuando el dispositivo está en el modo de respaldo de batería.

2. Cristal incorrecto: En un RTC, normalmente correrá lento, si es usado un cristal con una capacidad de carga especifica (CL) menor que la que se especifica en el RTC. La gravedad de la inexactitud depende del valor de CL.

3. Capacitancia parásita: La capacitancia parásita entre los pines de cristal y/o tierra, puede disminuir la marcha de un RTC. Por lo tanto hay que tener cuidado cuando se está diseñando el PCB para asegurar que la capacitancia parásita se mantenga a un mínimo.

4. Temperatura: La temperatura adicional a la establecida por el cristal, hace que el cristal oscile más despacio.

Reloj no corre

Los siguientes son los argumentos más comunes que causan que un RTC basado en un cristal no corra.

1. El problema más común cuando un RTC no corre, es que el bit CH (Clock Halt - detener reloj) o EOSC (enable oscillator - habilitar oscilador) no haya sido establecido o borrado, según sea necesario. Muchos RTC de Dallas Semiconductor incluyen un circuito que mantiene corriendo el oscilador cuando se lo energiza por primera vez. Esto permite a un sistema esperar hasta que el usuario configure los parámetros necesarios, sin entrar a modo de respaldo de batería. Cuándo el sistema es alimentado por primera vez, el programa/firmware debe habilitar el oscilador y pedir al usuario la hora y fecha correctas.

2. Cargas parásitas causadas por la condensación o residuos de flux de soldar en el PCB, también impiden que el oscilador corra.

3. Pines de entrada flotante. Cualquier entrada no utilizada, como Vbat, debe ser conectado a tierra. Si un pin de entrada queda flotando, la comunicación con el RTC puede que no funcione.

4. Asegúrese que el cristal sea conectado en los pines correctos (pin X1 y X2). Note que el circuito oscilador de los RTCs de Dallas son de bajo consumo y la señal en los pines del oscilador de entrada puede ser de sólo unos pocos cientos de milivoltios pico a pico, haciéndolo muy susceptible al ruido.

Tabla de estados del CORE

El CORE genera un código cuando lee, escribe y si es que hay algún error en la comunicación. Ver Tabla 11.

Estado	Código
Leer	32
Escribir	17
Error	33

Tabla 11 Estados del core I2c

Esto códigos denotan los estados por donde va pasando para generar la trama solicitada. Son enviados al MÓDULO con el fin de controlar cual fue el último estado antes de salir del mismo.

Esquemático de la Aplicación Ejemplo

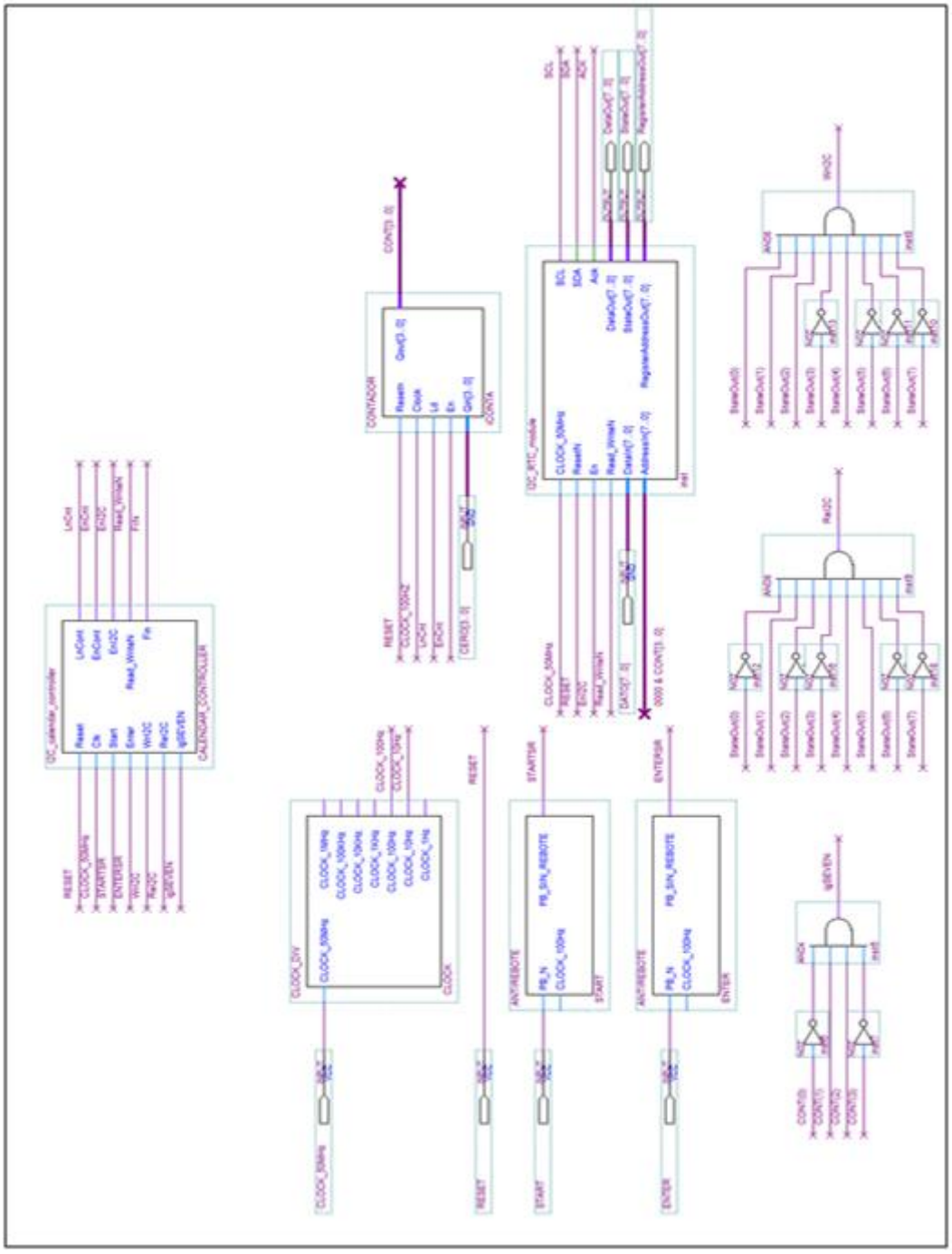

Figura 51 Esquemático de la aplicación ejemplo

Bibliografía

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

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.

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

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.

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.

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

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.