Tecnológico de Costa Rica

Ingeniería en Computadores

Introducción a los Sistemas Embebidos

Diseño de SoC mínimo para reloj despertador

Estudiantes:
Malcolm Davis
David Monestel
Fabian Solano

Profesor:
Ing. Jeferson González

24 de octubre de 2018

Índice

1. Descripción del Sistema

Este documento contiene el proceso detallado del diseño y desarrollo de un reloj despertador"que, se logró con el uso de un sistema a la medida implementado en la placa de desarrollo de Altera DE1-SoC.
Para proponer el prototipo del sistema desarrollo en proyecto se utilizó la metodología modular, se decidió de este modo para así facilitar el diseño individual de cada uno de los componentes o soluciones de subproblemas con menor complejidad que al final se unen para componer el sistema completo [1]. La subdivisión de estos problemas se hace en subsistemas individuales que cumplen con los criterios de desarrollo y pueden ser utilizados para crear el sistema completo, más adelante se explica cada uno de ellos.

1.1. Descripción de los Métodos

A continuación se describen los métodos utilizados para el funcionamiento del programa que ejecuta el procesador NIOS II para el reloj despertador.

- **displaySevenSegment(unsigned char* pPointer, int pNumber)** Método que se utiliza para el correcto funcionamiento de los displays de siete segmentos, actualiza el puntero poniendo en uno o cero el los bits requeridos según el número ingresado.
- **updateTimeDisplay()** Método que actualiza el tiempo que muestra el reloj llamando la función descrita anteriormente con los registros que corresponden a cada display.
- **updateAlarmDisplay()** Similar al anterior, actualiza el el tiempo de la alarma que se muestra en el display.
- **handlerTimerInterrupt(void* context)** Método que maneja la interrupción del tiempo, cada vez que el temporizador dispara una interrupción se actualiza el tiempo en segundos, minutos o horas según corresponda.
- **handleAction(int pIDButton)** Método que maneja las entradas del usuario que son ingresadas por medio de los botones, ya sea cambiar la hora, la alarma y otras funciones.

- **handleKeyInterrupt(void* context)** Método detecta las interrupciones disparadas por los botones de usuario.
- setupKeys(void) Método que inicializa las entradas asignando el valor de memoria correspondiente.
- **setupTimer(void)** Método que inicializa el temporizador asignando la dirección de memoria correspondiente, registrando las interrupciones y empezando el temporizador.

1.2. Descripción de las Bibliotecas

A continuación se describen las bibliotecas utilizadas:

- **unistd** Biblioteca que provee las operaciones POSIX.
- **alt_stdio** Biblioteca que provee las operaciones de entrada y salida.
- **alt_irq** Biblioteca que provee operaciones para el manejo de interrupciones.

1.3. Requisitos del Sistema

Es necesario la construcción de un SoC mínimo para el manejo de un reloj despertador con temporizador incluido mediante el uso de interrupciones. Algunos de los requisitos recolectados para el diseño de este sistema son:

- **R0** El sistema debe controlar y mostrar la hora, por medio de display de 7 segmentos.
- **R1** El sistema deberá permitir la configuración de la hora actual por medio de los botones de la tarjeta DE-SoC1.
- **R2** El sistema deberá permitir la configuración de la alarma con los botones de la tarjeta, así como mostrar la misma en el display 7 segmentos.
- **R3** El sistema deberá contar con un módulo temporizador (timer).
- **R4** Todo el control del sistema deberá realizarse por medio del CPU NIOS II.

Cuadro 1: Cumplimiento de los Requerimientos del Proyecto

Código	Estado
R0	Cumple
R1	Cumple
R2	Cumple
R3	Cumple
R4	Cumple
R5	Cumple
R6	Cumple
R7	Cumple

- **R5**El sistema deberá contar con una memoria en chip para el programa y los datos. La selección de 1 o 2 memorias deberá ser justificada.
- **R6**Todo componente dentro del sistema en chip deberá ser usado eficientemente y no se permitirán componentes que no tengan una funcionalidad justificada.
- **R7**La alarma deberá mostrarse de manera diferenciada en el hardware (combinación de leds, por ejemplo).

2. Metodología de diseño

A continuación se encuentra una descripción de la metodología de diseño utilizada para el proyecto, la cual involucra el análisis del problema, la investigación respectiva, y las propuestas de diseño.
La metodología de diseño modular como se mencionó anteriormente, y de igual forma se describió. Consiste en la división en módulos(compilación, ejecución e interfaz con el usuario), dónde el comportamiento del sistema deseado se aprecia al unir los módulos[1].

2.1. Análisis del Problema

El principal objetivo de la computación ubicua es la creación de productos inteligentes conectados que tengan una alta disponibilidad, y que hagan

la comunicación más fácil[9]. Estos dispositivos son utilizados para integrar elementos del entorno del ser humano facilitando su quehacer diario.
En los últimos años, el desarrollo de los dispositivos embebidos ha tenido un gran incremento gracias a su bajo consumo de energía y espacio. La tecnología más reciente pretende crear cada vez más dispositivos con más funciones y que utilicen la menor cantidad de recursos energéticos o de espacio. Para lograr esto se debe utilizar menos componentes y/o componentes más modestos.
Este proyecto está enfocado en desarrollar el **S**ystem **o**n **C**hip que utilizaría un reloj despertador. Por los puntos descritos anteriormente, y con el objetivo de consumir la menor cantidad de recursos y mantener la simplicidad del sistema el SoC se diseña para contener los componentes mínimos que permitan crear un reloj con alarma programable.
Para desarrollar el prototipo del sistema desarrollado en proyecto se utilizó la metodología modular, se decidió de este modo para así facilitar el diseño individual de cada uno de los componentes o soluciones de subproblemas con menor complejidad que al final se unen para componer el sistema completo [1]. La subdivisión de estos problemas se hace en subsistemas individuales que cumplen con los criterios de desarrollo y pueden ser utilizados para crear el sistema, en este documento se explica cada uno de ellos.
Tomando en cuenta que la forma más sencilla de hacer que el procesador consuma menos energía es mediante el uso de interrupciones, que permiten al procesador realizar únicamente los cálculos requeridos cuando es necesario y evita comprobaciones innecesarias.Por eso, esta forma de comunicación entre los dispositivos y el procesador reduce la energía consumida lo cuál favorece la implementación de un **SoC**[5]. Así, este manejo de interrupciones sería parte de los requisitos que el sistema debe de cumplir.
Otro de los retos propuestos con este proyecto, es que se debe de proveer una interfaz al usuario para poder modificar los valores del sistema, realizando acciones como cambiar la hora y configurar la alarma. Esta interfaz debe de ser intuitiva ya que ya existen productos de este tipo, además como con el resto del proyecto, se debe de implementar en un entorno con pocos recursos. Tanto el reloj como la alarma, y las configuraciones deben de ser mostradas utilizando los LEDs de los displays de 7 segmentos provistos en la placa de desarrollo. Para mantener el estándar impuesto por la industria, se debe mostrar la hora en un formato de HH:MM:SS de 24 horas.
Al igual que los relojes antiguos, se utilizará como método de entrada botones pulsadores que representan cada uno una funcionalidad especial. Con

esto se reduce la cantidad de recursos de entrada, especialmente utilizando los pulsadores con anti rebote.
Por lo expuesto anteriormente, se deja en claro que el proyecto busca simular el comportamiento de un reloj con alarma despertadora antiguo pero en un dispositivo moderno con la menor cantidad de recursos posible. Además, según lo expuesto esto debería de ser posible con las herramientas disponibles.

2.2. Investigación Respectiva

La investigación del proyecto se enfocó principalmente en el uso de las interrupciones y el dispositivo temporizador, esto porque el resto de requisitos involucran módulos y herramientas ya utilizadas previamente, en cursos de la carrera como Arquitectura de Computadores II.
El SoC a diseñar debe tener un control de interrupciones por eso se investigó para poder entender completamente cada uno de los dispositivos requeridos y como manejar su interrupción.
Para el temporizador, al ser un dispositivo nuevo se investigó completamente sus características y su estructura interna[7]. Para logra entender su comportamiento, se analizó los ejemplos provistos y la forma de manejar las interrupciones. Se desarrollaron pruebas para verificar que la información adquirida anteriormente fuera verídica, y se concluyó que se tenía la información necesaria para realizar el proyecto.
Luego de entender al temporizador, y con las pruebas realizadas, se obtuvo la experiencia necesaria para manejar las interrupciones de los otros dispositivos. Así, se investigó el funcionamiento de las interrupciones de los botones pulsadores[8]. Aunque ya se conocía el funcionamiento de los botones, aún no se había utilizado la característica que les permite generar interrupciones, al igual que con el temporizador se analizó la estructura de los PIO y sus interrupciones.
De igual forma, para el manejo de los leds, y los GPIO se referenció el manual de usuario oficial de la placa de desarrollo, esto para datos puntuales sobre el manejo de cada dispositivo aunque ya se conocía la funcionalidad general de los mismos[8].
Con las pruebas realizadas, incluyendo las de las interrupciones, se diseñó la maquina de estados del sistema, que es una tarea conocida. Y por último la unión del código no fue una tarea compleja ya que se conocía el proceso.

2.3. Propuestas de Diseño

Seguidamente, se muestran las propuestas de diseño planteadas para el sistema a desarrollar, cada una difiere de la otra según la forma en que describe el comportamiento que el sistema debería tener para cumplir con las funciones requeridas. Se planearon 3 propuestas, una que utiliza sondeo y otras dos que utilizan los temporizadores de diferentes formas.
Propuesta 1: Para la primera propuesta, como se mencionó anteriormente, se plantea utilizar el sondeo para la verificación de los botones, controlar los siete segmentos y al finalizar utilizar una operación del procesador para hacer un "sleep"durante un tiempo determinado. La única interrupción que se manejaría para esta propuesta sería la del temporizador que aumenta el valor de las variables que llevan el tiempo del sistema.
Propuesta 2: La segunda, consiste en utilizar temporizadores para cada parte del sistema que necesita llevar el tiempo, i.e. reloj, alarma y temporizador. Cada uno de esos temporizadores actualizaría de manera independiente de los demás, esto con el objetivo de reducir el tamaño del código a lo mínimo, estructurando el código de tal forma que el código correspondiente al manejo de interrupciones sea la mayor parte, evitando el código de control, lo cuál permite utilizar una memora de instrucciones de menor tamaño. Los botones activan interrupciones igual que los temporizadores, éstas contienen el código de respuesta de salida y el programa principal sólo debería inicializar datos.
Propuesta 3: La tercera propuesta, es utilizar sólo un temporizador que lleve el control de los segundos transcurridos del sistema, la interrupción de cada segundo actualiza la información de las variables de tiempo y alarma. Con esta propuesta a diferencia de la anterior, se busca reducir la cantidad de componentes de hardware sacrificando un mayor uso de memoria de instrucciones al tener que utilizar espacio para las instrucciones de control y cálculos requeridos para el funcionamiento correcto del sistema. Las interrupciones de los botones se encargarían de actualizar datos para que la maquina de estados se mueva entre los estados, pero no todas las operaciones recaen en el programa principal.

2.4. Comparación de las propuestas

En esta sección se profundiza en las razones por las cuales se eligió la propuesta número 3, señalando los puntos fuertes y débiles de cada una.
La **primer propuesta**, es muy sencilla de implementar, pero requiere el

procesamiento de muchas instrucciones innecesarias mientras espera alguna orden del usuario, consumiendo energía que podría ser ahorrada. Y esto va en contra de los objetivos del proyecto ya que se desea desarrollar un SoC que consuma una baja cantidad de energía. Además, esta propuesta tiene un tiempo de retraso al tener que esperar a que la instrucción que sondea el botón adecuado se active, por lo tanto algunas acciones tardan en activarse; y este retraso se ve aumentado según la cantidad de operaciones que se quieran realizar en simultaneo. Por lo tanto no terminaría cumpliendo todos los requisitos del sistema(ver cuadro **??**).
La **segunda propuesta** plantea un manejo de interrupciones que cumple con las funcionalidades solicitadas, reduce el tamaño de la memoria de instrucciones que se utilizan. Pero al requerir de 3 temporizadores, y esto contraria el objetivo de mantener el consumo de energía tan bajo como sea posible, además de la mínima cantidad de dispositivos posibles.
Por último, la **tercera propuesta** cumple con utilizar la mínima cantidad de dispositivos posibles, y al usar las interrupciones para cada uno de esos dispositivos evita la ejecución de instrucciones de comparación innecesarias. Pero de igual forma que las otras, tiene un punto negativo y este es que s necesita una mayor cantidad de instrucciones de control para el manejo de las interrupciones, esto aumenta el tamaño del programa lo cuál aumenta el tamaño de la memoria que debe ser sintetizada.
Tomando en cuanta lo anterior, descartamos la primera porque aunque se puede cumplir con las funcionalidades requeridas no se hace de la mejor forma y la calidad resultante es dudosa. En cuanto a la segunda, si tomamos en cuenta que no cumple con ser un sistema mínimo entonces se descarta. Y por último, aunque la tercer propuesta requiera de mayor cantidad de memoria (aproximadamente 7000Kb) decidimos que era una cantidad de memoria despreciable en comparación con los beneficios que tiene esta propuesta con respecto a las otras dos. Que en pocas palabras representan un menor consumo energético y de espacio y precio con respecto a la primera y la segunda respectivamente.

2.5. Mapa de Memoria

La figura 1 contiene la tabla con las direcciones de memoria de cada componente. Este mapeo se realizó con la herramienta Qsys para poder controlar los componentes.

	cpu.data_master	cpu.instruction_master
GPIO.s1	0xc050 - 0xc05f	
S0.s1	0xc060 - 0xc06f	
S1.s1	0xc070 - 0xc07f	
S2.s1	0xc080 - 0xc08f	
S3.s1	0xc090 - 0xc09f	
S4.s1	0xc0a0 - 0xc0af	
S5.s1	0xc0b0 - 0xc0bf	
cpu.debug_mem_slave	0x0800 - 0x0fff	0x0800 - 0x0fff
jtag_uart.avalon_jtag_slave	0xc000 - 0xc007	0xc000 - 0xc007
key.s1	0xc040 - 0xc04f	
led.s1	0xc010 - 0xc01f	
onchip_memory.s1	0x8000 - 0xbfff	0x8000 - 0xbfff
timer.s1	0xc020 - 0xc03f	

Figura 1: Mapa de Memoria

3. Herramientas de Ingeniería

Seguidamente, se muestran las principales herramientas de ingeniería utilizadas para el diseño e implementación del SoC del reloj despertador.

3.1. Plataform Designer

La primera herramienta a mencionar es la herramienta de integración de sistemas Plataform Designer(QSys), esta ahorra mucho tiempo y esfuerzo en el proceso de diseño de FPGA, esto mediante la generación automática de la lógica de interconexión para conectar las funciones de propiedad intelectual y subsistemas (IP)[3]. Que fue de mucha utilidad para diseñar los módulos utilizados en el sistema.

3.2. Quartus

El programa Quartus incluye las herramientas necesarias para diseñar para las FPGAs, SoCs y CPLDs. Herramientas de diseño, la síntesis de optimización, verificación y simulación. Este software permitió la integración de los módulos para crear el sistema. Por último se sintetiza el sistema y se programa la FPGA. [2]

3.3. Altera DE1-SoC

La placa de desarrollo DE1-SoC presenta una plataforma de diseño de Hardware robusta diseñada por Altera alrededor de sus FPGA SoC. Combina un procesador de doble núcleo Cortex-A9 con lógica programable de nivel

industrial para dar la mejor flexibilidad [4]. Esta herramienta fue especialmente útil para materializar los modelos diseñados y el código sintetizado.

Referencias

[1] J. Leiva, *"Diseño de Algoritmos"*, Lcc.uma.es. [Online]. Available: http://www.lcc.uma.es/ jlleivao/algoritmos/t2.pdf. [Accessed: 21- Sep- 2018].

[2] *"FPGA Design Software - Intel® Quartus® Prime"*, , Intel. [Online]. Available: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html. [Accessed: 18- Oct- 2018].

[3] *"Platform Designer - Intel's System Integration Tool"* , Intel.com. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/products/design-software/fpga-design/quartus-prime/features/qts-platform-designer.html. [Accessed: 18- Oct- 2018].

[4] T. Technologies, *"Terasic - SoC Platform - Cyclone - DE1-SoC Board"*, Terasic.com.tw. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836. [Accessed: 18- Oct- 2018].

[5] J. Leiva, *"What is a CPU Interrupt Code (CIC)? - Definition from Techopedia"*, Techopedia.com. [Online]. Available: https://www.techopedia.com/definition/5653/cpu-interrupt-code-cic. [Accessed: 18- Oct- 2018].

[6] *."Exception Handling, Nios II Software Developer's Handbook"*, Intel.com, 2018. [Online]. Available: https://www.intel.com/content/dam/altera-www/global/ja_JP/pdfs/literature/hb/nios2/n2sw_nii52006.pdf. [Accessed: 18- Oct- 2018].

[7] *"Timer Core, Quartus II 9.1 Handbook, Volume"*, Intel.com, 2018. [Online]. Available: https://www.intel.co.jp/content/dam/altera-www/global/ja_JP/pdfs/literature/hb/nios2/n2cpu_nii51008.pdf [Accessed: 18- Oct- 2018].

[8] *"DE1 SoC User Manual"*, Courses.cs.washington.edu, 2018. [Online]. Available: https://courses.cs.washington.edu/courses/cse467/15wi/docs/DE1_SoC_User_Manual.pdf. [Accessed: 18- Oct- 2018].

[9] *"What is Ubiquitous Computing? - Definition from Techopedia"*, Techopedia.com, 2018. [Online]. Available: https://www.techopedia.com/definition/22702/ubiquitous-computing. [Accessed: 21 - Sep- 2018].