Proyecto 3 - Sistema Empotrado simple en QSys

Mauricio Caamaño, Marco Espinoza, Tomás González

***Abstract*—Este informe describe la ejecución de un proyecto de implementación de un sistema en QSys de Altera en el cual se incluyó en la lógica programable un softcore del procesador Nios II, en conjunto con otros bloques IP para memoria y comunicación serial. Se muestra el proceso de ejecución mediante las herramientas para el manejo de un temporizador que permitió realizar estimaciones de tiempo de ejecución de aplicaciones. Además, se discute acerca la generación de códigos de prueba y simulación de un módulo SPI esclavo que interactúa con el sistema.**

***Index Terms*—Qsys, Lógica programable, Nios II, SPI.**

I. Introducción

Los sistemas empotrados basados en FPGAs actuales contienen una gran variedad de componentes de hardware que han permitido aumentar la escalabilidad y desempeño. Los nuevos Sistemas en Chip cuentan con procesadores, memorias, interfaces de entrada/salida, protocolos de comunicación, temporizadores y más, accesibles desde el ámbito de software como núcleos IP del fabricante o terceros, que agilizan significativamente el proceso de diseño e implementación de un proyecto. Los nuevos sistemas heterogéneos se caracterizan por dar al desarrollador una arquitectura dividida en un sistema de procesamiento (PS) y otra de lógica programable (PL), con una robusta interfaz de comunicación entre ambas partes, como por ejemplo AMBA AXI de ARM, utilizado en familias de Altera y Xilinx.

Los fabricantes ponen a disposición del desarrolador herramientas de software para la integración de estos sistemas, que se encargan de interconectar automáticamente la lógica, manejar bloques de propiedad intelectual (IP) y otros subsistemas. Para el caso de este proyecto, el software Qsys de Altera incluye el soft core del procesador embebido Nios II. Este procesador se implementa en el chip FPGA y permite ejecutar aplicaciones de alto nivel que controlan la lógica implementada en el resto del chip.

El documento se estructura de la siguiente manera: la sección II describe el procedimiento ejecutado para la elaboración del diseño y la integración del sistema completo mediante las diferentes herramientas de software. La sección III muestra los resultados obtenidos y las simulaciones de las diferentes transferencias y ejecución de los bloques. Además, se discute brevemente en la sección IV acerca del uso de un *watchdog* y sus implicaciones. Por último, se describen las conclusiones en la sección V.

II. Proceso de diseño

A. Diseño de los bloques del sistema

El diseño del sistema se basó a partir de la descripción del proyecto, en la cual se especifica cada uno de los módulos que deben de ser usados para la elaboración del sistema a desarrollar. La siguiente imagen muestra un diagrama de bloques del sistema requerido.

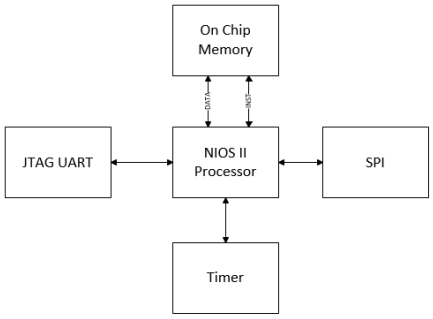


Figure 1. Diagrama de bloques del diseño implementado

Básicamente, se tiene un procesador NIOS II conectado a una memoria RAM de 64KB, además el procesador recibirá datos a través del JTAG UART y también a través del SPI, adicionalmente se tiene un timer que se encargará de medir el tiempo total de ejecución del sistema. A continuación se especifica el funcionamiento de cada uno de los bloques:

1) Memoria RAM:: Este bloque fue generado utilizando la librería de IP Core que posee altera, la misma es una memoria RAM de 64 KB que tiene como entradas un clock, un reset, un byte enable, una señal de read y write, un address y un bus bidireccional de datos. Un ejemplo de una escritura se muestra a continuación: En este caso se puede observar el address de

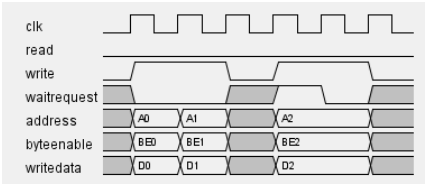


Figure 2. Escritura de un dato en la memoria RAM implementada

la memoria en la que se desea escribir, el byte enable y el dato a escribir, además de que la señal de write se encuentra en 0x1.

Finalmente, la siguiente figura muestra el bloque generado por el IP core para la memoria RAM:

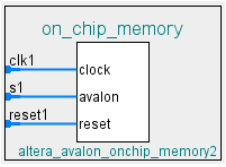


Figure 3. Bloque de memoria RAM generado con IP core

2) Procesador NIOS II:: Para el uso del procesador NIOS II se utilizó de igual manera la biblioteca de IP Core, y se escogió el procesador NIOS II/e ya que el mismo cumple con los requerimientos del sistema a diseñar y por lo tanto no utilizamos recursos innecesarios para el mismo. La figura a continuación muestra el bloque generado:

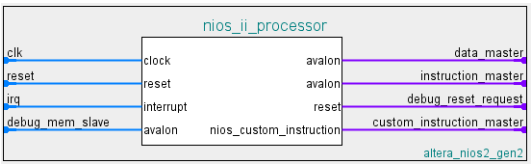


Figure 4. Bloque del procesador NIOS II generado con el IP core

3) Timer:: Se utilizará un timer para determinar el consumo de CPU utilizado por una aplicación, por lo que se debe de generar el mismo utilziando de igual manera la biblioteca de IP Core que proporciona Altera. El mismo es un timer full featured de 32 bits. La figura siguiente muestra el bloque generado:

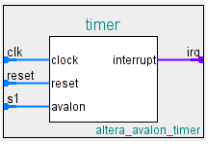


Figure 5. Bloque del timer generado por el IP core de Altera

4) JTAG UART:: El UART se utilizara para la comunicación de datos entre el procesador y el sistema. Al igual que en los casos anteriores, se utiliza la biblioteca IP core para la generación del mismo. La figura siguiente muestra el resultado de dicho bloque:

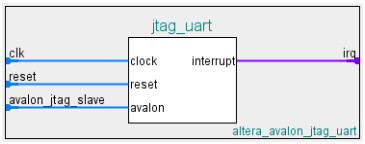


Figure 6. Bloque para el JTAG UART generado por la biblioteca de IP Core

5) SPI:: Por último, en la segunda parte del proyecto se requiere utilizar un SPI para realizar una comunicación serial de datos entre una aplicación de C y el sistema desarrollado. Por ello, se requiere de un bloque SPI que es igualmente generado con la biblioteca IP Core. El bloque elaborado se muestra en la figura a continuación:

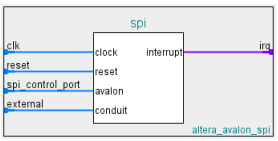


Figure 7. Bloque del SPI generado por el IP core de Altera

B. Implementación del diseño para la utilización del timer

En esta sección se utilizó el timer explicado en la sección anterior para poder medir el tiempo de ejecución de una aplicación generada en el lenguaje de programación C. Para la utilización del timer se debió agregar en la aplicación el código necesario para poder hacer la lectura del mismo. Para ello se definió un puntero apuntando a la base definida para el timer. La siguiente imagen muestra la definición de la base en el qsys y la asignación del puntero en la aplicación de C: Seguidamente se configurará en la aplicación de C el timer

(a) Definicion del timer en qsys

```
#define TIMER_BASE (unsigned int*)0x11020
#define LIST_SIZE 35
```

(b) Codigo de C utilizado para definir la base del timer

Figure 8. Definicion del timer implementado

para que el mismo se detenga mientras se configuran los valores iniciales del mismo. La siguiente figura muestra la especificación del registro de timer, para poder ser utilizado adecuadamente. Por lo que, en la aplicación, antes de ejecutar

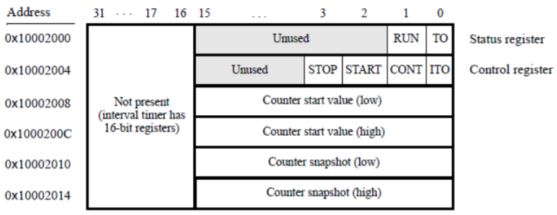


Figure 9. Especificacion del registro utilizado para configurar el timer

el programa, se debe de escribir el bit 3 del registro de control para poder detener el contador, una vez detenido se escriben los valores iniciales del mismo. Una vez que los valores iniciales se han escrito, se inicia la ejecución del programa. La figura siguiente muestra el código implementado que realiza esta configuración:

Además, una vez que el programa se ha ejecutado, se debe de hacer un snapshot para poder determinar el valor final del contador. Para ello se deben leer tanto la parte alta como la parte baja del counter snapshot, y una vez leído el número de ciclos leído es multiplicado por la frecuencia del reloj, de esta manera es que se obtiene el tiempo total de ejecución del programa. En la figura siguiente se muestra el

```
// Parte 2 - Determinar tiempo de ejecucion de una aplicacion
unsigned int* timer = TIMER_BASE;
unsigned int timer_data[2];
*(timer + 1) = 0x0008; //Stop timer
*(timer + 2) = 0xffff; //Set periodl register to max value
*(timer + 3) = 0xffff; //Set periodh register to max value
*(timer + 1) = 0x0004; //Start timer
test_program(); //Run test program
*(timer + 4) = 0x0000; //Indicate timer to take a snapshot
timer_data[0] = *(timer + 4); //Get snapl register value
timer_data[1] = *(timer + 5); //Get snaph register value
timer_data[1] = timer_data[1] << 16;
unsigned int timer_data_joined = timer_data[1] | timer_data[0];
int cycles_elapsed = 0xffffffff - timer_data_joined; //Cycles elapsed during the test program
int time_elapsed = cycles_elapsed * CLK_PERIOD; //Time elapsed during the test program
printf("Time elapsed = %d ns\n", time_elapsed);
```

Figure 10. Codigo implementado para configurar los valores iniciales del timer

código implementado para poder hacer la lectura y el cálculo explicado:

```
test_program(); //Run test program
*(timer + 4) = 0x0000; //Indicate timer to take a snapshot
timer_data[0] = *(timer + 4); //Get snapl register value
timer_data[1] = *(timer + 5); //Get snaph register value
timer_data[1] = timer_data[1] << 16;
unsigned int timer_data_joined = timer_data[1] | timer_data[0];
int cycles_elapsed = 0xffffffff - timer_data_joined; //Cycles elapsed
int time_elapsed = cycles_elapsed * CLK_PERIOD; //Time elapsed during
printf("Time elapsed = %d ns\n", time_elapsed);
```

Figure 11. Codigo implementado para leer el tiempo total de ejecucion

C. *Implementación del diseño para la utilización del SPI*

Debido a que SPI es uno de los estándares de comunicación serial más utilizados. En esta sección se generará una aplicación de C para poder comunicarse con el puerto serial y además es posible escribir y leer datos desde la interfaz. Sin embargo, a diferencia del caso donde se utilizó el timer, que los métodos a los registros de configuración del timer se hicieron manualmente, en este caso es posible utilizar los drivers provistos por altera en el BSP. Para ello, se utilizó el método int alt_avalon_spi_command, el cual se explica a continuación:

```
Función: int alt_avalon_spi_command(alt_u32 base, alt_u32 slave,
                                    alt_u32 write_length, const alt_u8 * write_data,
                                    alt_u32 read_length, alt_u8 * read_data,
                                    alt_u32 flags);
```

Figure 12. Método utilizado para la escritura y lectura de datos a través del SPI

- Slave: Es el indicador del esclavo a hablar, en nuestro caso solo tenemos uno entonces dejar este valor en 0.
- X_length: la longitud de datos a leerescribir en bytes.
- X_data: los punteros del buffer de lectura/escritura.
- Flags: una de dos:
  - ALT_AVALON_SPI_COMMAND_MERGE
  - ALT_AVALON_SPI_COMMAND_TOGGLE_SS_N

Por lo tanto, hacienda uso de esta función, se generó el siguiente código para acceder a la interfaz a través del SPI:

Como se observa en la figura, se genera un array de dos números que se utilizará para escribir a la interfaz a través del SPI, y se genera otro array que se utilizará para leer los datos desde la interfaz a través del SPI.
Seguidamente, se utiliza la función explicada anteriormente

```
// Parte 5 - Utilizar el SPI para escribir y leer datos
unsigned char write_data[2] = {0xaa, 0x81};
unsigned char read_data[2];
printf("write_data[0] = %x\n", write_data[0]);
printf("write_data[1] = %x\n", write_data[1]);
alt_avalon_spi_command(SPI_BASE, 0, 2, write_data, 2, read_data, ALT_AVALON_SPI_COMMAND_MERGE);
printf("read_data[0] = %x\n", read_data[0]);
printf("read_data[1] = %x\n", read_data[1]);
```

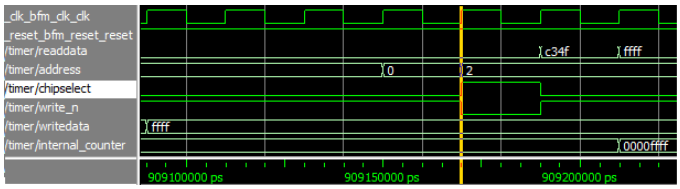
Figure 13. Codigo C utilizado para la implementación del SPI en la interfaz

explicada para setear la base donde se encuentra el SPI en la interfaz, además se escriben los dos datos contenidos en el array, y además se especifica el arreglo donde se guardaran los datos leídos a través del SPI una vez que se hacen las escrituras.
Una vez finalizado esto, se imprime a través del JTAG los valores leídos desde el SPI.

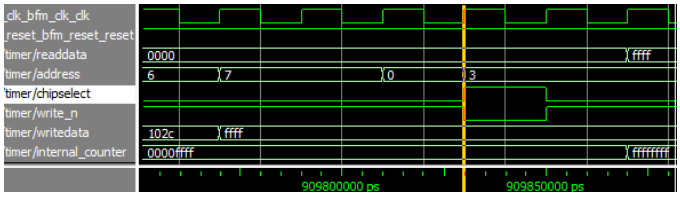
III. Resultados obtenidos

A. *Parte I. Uso de temporizador*

Para la primera parte del proyecto se utilizó el temporizador del sistema para medir el tiempo de ejecución de una aplicación ejecutada por el CPU. Mediante la aplicación en C ejecutada por el CPU, se realizó el procedimiento para configurar el temporizador, siguiendo la secuencia de comandos requerida. Primero se detiene el temporizador, enviando el comando $0x0008$ a través del registro de control para poner en alto el bit de *STOP*; éste es leído en el instante en que se ativa la señal de $chipselect$. Seguidamente se configura el valor del temporizador en el valor máximo, es decir $0xffffffff$. Para ello, primero se envía la parte baja de la palabra de 32 biys, y luego la parte alta. Los diagramas de la figura 14 muestra la configuración de los registros.



(a) Parte baja



(b) Parte alta

Figure 14. Escritura del valor inicial del temporizador

Para iniciar el contador, se procedió a enviar el comando $0x0004$ al registro de control, poniendo en alto el bit de *START*, lo cual inicia la cuenta regresiva. Inmediatamente se ejecuta la rutina de prueba *test_program()* y, al finalizar, se toma una captura del valor del temporizador, escribiendo a través del registro *snapl*. La figura 15 muestra cómo dos ciclos después del momento en que se escribe en el registro, se

observa en $readdata$ el valor $0xED5B$. dos ciclos después que se escribe el valor.

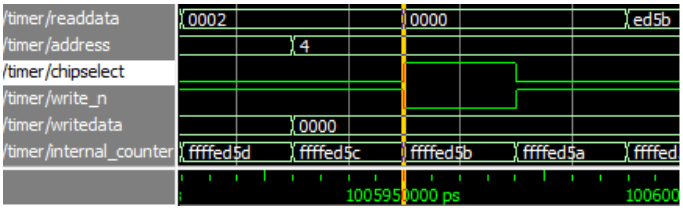


Figure 15. Captura del valor del registro de conteo del temporizador

Para el cálculo del tiempo de ejecución, se utiliza la siguiente ecuación, donde $time_{max}$ denota el valor inicial de conteo, es decir $0xffffffff$, y $time_{cnt}$ contiene el valor obtenido durante la captura del conteo.

$$time\_elapsed = (timer_{max} - timer_{cnt}) \cdot clk\_period$$

El resultado obtenido para la ejecución de la rutina *test_program()* fue de:

$$time\_elapsed = 4722 \cdot 20ns = 95440ns$$

Este valor también fue calculado y desplegado mediante la interfaz JTAG UART desde la aplicación en C ejecutada por el CPU. La salida de la simulación completa, incluyendo la parte II, se puede observar en la figura 19.

B. Parte II. Comunicación con esclavo SPI

En esta parte se integró al sistema un módulo de interfaz de comunicación SPI maestro y se simuló un dispositivo esclavo capaz de enviar y recibir datos del sistema completo. Para transmitir datos desde el sistema maestro, se escribió uilizó el comando $alt_avalon_spi_command$ para enviar dos bytes: $0xaa$ y $0x81$ y luego recibir dos bytes del esclavo. Esta sección del código se muestra en la figura 16.

```
printf("write_data[0] = %x\n", write_data[0]);
printf("write_data[1] = %x\n", write_data[1]);
alt_avalon_spi_command(SPI_BASE, 0, 2, write_data,
    2, read_data, ALT_AVALON_SPI_COMMAND_MERGE);
printf("read_data[0] = %x\n", read_data[0]);
printf("read_data[1] = %x\n", read_data[1]);
```

Figure 16. Código para envío y recepción de datos por interfaz SPI maestro

La figura 17 muestra el envío de los dos bytes a través de la señal de $MOSI$. Los bytes recibidos por el esclavo se observan correctamente a través de $rxdata$.

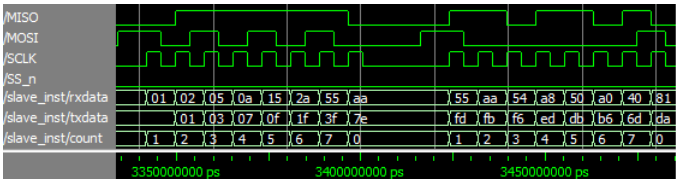


Figure 17. Diagrama de transmisión SPI maestro a esclavo

Para la transmisión por parte del esclavo, se utilizó en el mismo *testbench* un generador de bytes aleatorio. El ejemplo del diagrama de la figura 18 muestra la recepción de los bytes $0x5a$ y $0x2f$. Estos valores son deplegados a través de la interfaz UART del sistema principal en las dos últimas líneas de la salida mostrada en la figura 19.

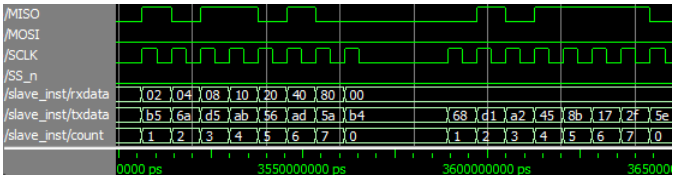


Figure 18. Diagrama de recepción SPI maestro a esclavo

```
# Time elapsed = 95440 ns
# write_data[0] = aa
# write_data[1] = 81
# @ 3341770: rxdata = 00 (00000000)
# @ 3396650: txdata = 3f (00111111)
# @ 3423870: rxdata = aa (10101010)
# @ 3478750: txdata = 6d (01101101)
# @ 3506430: rxdata = 81 (10000001)
# @ 3561310: txdata = 5a (01011010)
# @ 3589230: rxdata = 00 (00000000)
# @ 3644110: txdata = 2f (00101111)
# read_data[0] = 5a
# read_data[1] = 2f
```

Figure 19. Mensajes de salida de la simulación partes I y II

IV. Usos de un timer como WatchDog

El watchdog consiste en la utilización de un timer que enviará una interrupción después de un tiempo determinado. Esto puede ser utilizado entre otras cosas para evitar que una aplicación se quede utilizando los recursos del procesador durante un tiempo determinado, de manera que si este tiempo se excede, el watchdog enviará una interrupción para que la aplicación termine su ejecución. Algunos de los beneficios que puede presentar un watchdog son los siguientes:

- Evitar que una aplicación que se ha quedado en un loop infinito, se quede utilizando los recursos del sistema .
- Definir un tiempo máximo permitido para cada una de las aplicaciones, para así evitar que una aplicación no permita a las demás poder ser ejecutadas.

Sin embargo, se debe tener cuidado del tiempo definido para el watchdog, debido a que el mismo podría sacar de operación una aplicación que aún está ejecutando un proceso, de manera que el mismo no podría ser terminado adecuadamente.

V. Conclusiones

El desarrollo del proyecto permitió comprobar los beneficios en términos de productividad y eficiencia del uso de herramientas de diseño de sistema como Qsys. Siguiendo el procedimiento de diseño se logró implementar e integrar rápidamente un sistema completo con procesador, memoria e interfaces de comunicación.