Diseño e Implementación de un servidor de base de datos utilizando la FPGA Terasic DE10-Standard

Lino Ontano
Telemática
Guayaquil - Ecuador
lontano@espol.edu.ec

Jocelyn Miranda
Telemática
Guayaquil - Ecuador
jocammir@espol.edu.ec

***Resumen*—Diseño e implementación de un sistema que realiza el monitoreo a una base de datos implementada en el sistema operativo Linux del Hard Processor System (HPS) de la FPGA Terasic DE10-Standard y es mostrada por pantalla utilizando protocolo VGA.**

***Index Terms*—FPGA, HPS, VGA**

I. Introducción

El presente proyecto se basa en el diseño y la implementación de un sistema embebido que realice monitoreo a una base de datos. El servidor de la base de datos es implementado en el Hard Processor System (HPS) de la FPGA Terasic DE-10 Standard, la cuál es manejada vía Putty SSH conectada a una red local. La FPGA es la encargada de realizar las consultas por medio de sus periféricos de entrada al HPS el cual ejecuta scripts que devuelven valores y escribe esos resultados en memoria para que la FPGA lea y muestre el resultado por pantalla por medio de su puerto VGA. El proyecto consta de tres etapas:

1. Montar el servidor de la base de datos en el HPS, habilitar el acceso vía SSH y configurar las interfaces de red para el acceso en la red local. La imagen del sistema operativo es proporcionada por la tarjeta desde el sitio web de Terasic.
2. Diseñar e implementar la arquitectura de la FPGA donde el procesador Nios II enviará datos para ser mostrados por medio del VGA. Además de proporcionar acceso a los periféricos de entrada, para nuestro caso, habilitar acceso a los *switches* que puedan ser leídos por el procesador y acceso a la *On-Chip Memory* de la FPGA, para leer los valores que serán enviados por el HPS en la siguiente etapa.
3. Finalmente, la arquitectura desarrollada deberá ser levantada por el HPS, para ello, se utilizará el proyecto de demostración Control Panel QT[1] y se modificará dicha arquitectura y se agregará los bloques que requerimos. Se utiliza ese proyecto ya que contiene lo necesario para el correcto funcionamiento del HPS junto a la FPGA.

II. Metodología

II-A. Hardware

[1]El proyecto Control Panel QT se lo puede encontrar en el CD proporcionado por la tarjeta; cuenta con licencia en ciertos bloques, los cuales no son utilizados y lo demás es utilizado exclusivamente con fines educativos.

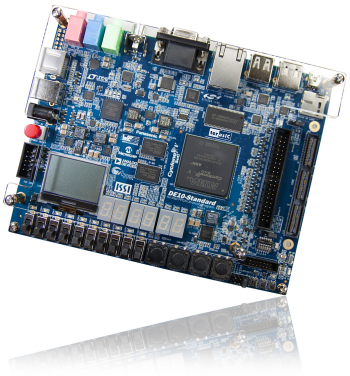


Figura 1: Tarjeta DE10-Standard.

Tarjeta de desarrollo DE10-Standard: El kit de desarrollo DE10-Standard presenta una robusta plataforma de diseño de hardware construida alrededor del Intel System-on-Chip (SoC) FPGA, que combina los últimos núcleos incorporados de doble núcleo Cortex-A9 con una lógica programable líder en la industria para una máxima flexibilidad de diseño. Los usuarios ahora pueden aprovechar la potencia de una reconfigurabilidad tremenda combinada con un sistema de procesador de bajo rendimiento y alto rendimiento. El SoC de Altera integra un sistema de procesador duro (HPS) basado en ARM que consiste en un procesador, periféricos e interfaces de memoria vinculados sin problemas con el tejido FPGA utilizando una red troncal de interconexión de gran ancho de banda.
La placa DE10-Standard tiene muchas características que permiten a los usuarios implementar una amplia gama de circuitos diseñados, desde circuitos simples hasta varios proyectos multimedia.
FPGA

- Intel Cyclone V SX SoC—5CSXFC6D6F31C6N
- Dispositivo de configuración serial - EPCS128
- USB-Blaster II onboard for programming; JTAG Mode
- 64MB SDRAM (16-bit data bus)
- 4 botoneras
- 10 switches
- 10 LEDs
- 6 pantallas de 7 segmentos
- Cuatro fuentes de reloj de 50MHz del generador de reloj

- 24-bit CD-quality audio CODEC con línea de entrada, línea de salida y jack de entrada de micrófono.
- VGA DAC con conector VGA de salida
- Decoder de TV y conector de entrada de TV
- PS/2 mouse/conector de teclado
- IR receptor y emisor.
- Un HSMC con configuración I/O standard 1.5/1.8/2.5/2.3
- Una expansión de 40 pines con protección de diodos.
- A/D convertidor, 4 pines SPI interfaz con FPGA

ARM-BASED HARD PROCESSOR SYSTEM **(HPS)**

- 925 MHz Dual-core ARM Cortex-A9 MPCore processor
- 1GB DDR3 SDRAM (32-bit data bus)
- 1 Gigabit Ethernet PHY con conector RJ45
- 2 puertos USB Host, conector USB normal tipo-A
- Socket para tarjeta Micro SD
- Acelerómetro (interfaz 12C + interrupt)
- UART a USB, USB mini-B conector
- botón reset *warm* y botón reset *cold*
- Un botón de usuario y un LED de usuario
- LTC 2x7 cabecera de expansión
- 128x64 puntos de módulo LCD con blacklight

Device Tree Blob: Para el correcto funcionamiento del hardware de la FPGA con el sistema operativo de la HPS requerimos una estructura que permita acceder y usar esos componentes. Un *Device Blob* es una estructura de datos que describe los componentes de hardware de un dispositivo en particular para que así el *kernel* del sistema operativo pueda usarlo y manejar esos componentes. Se requerirá generar el archivo .dtb para que exista el puente entre el HPS y la FPGA.

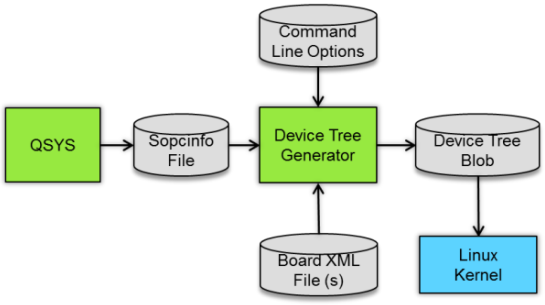


Figura 2: Generador del Device Tree.

II-B. Arquitectura

En la FPGA se requiere tener acceso al periférico de los *switches*, tener un *On-Chip Memory* y un procesador Nios II como arquitectura básica. Para poder mostrar las consultas de la base, requerimos mostrar texto por medio del VGA. Para lograr aquello, utilizamos un bloque denominado *Character Buffer*, que consta con librerías para poder ser manipulado desde el procesador y mostrar las líneas que queramos. Ese Character Buffer requiere transformar su salida a una frecuencia manejable para el VGA *Controller*, por tanto se utiliza un *Dual Clock Buffer*, que baja la frecuencia de salida del *Character Buffer* a 25MHz. El VGA *Controller* que se conecta al VGA DAC para mostrar por pantalla. Para el correcto funcionamiento del HPS se añade a la arquitectura de la FPGA un procesador ARM, y varios componentes que utiliza el HPS obtenido del proyecto de demostración antes mencionado.
En el HPS debe estar montado la base de datos en la microSD que contiene la imagen del sistema operativo, además de los archivos de la arquitectura de la FPGA antes creada para acceder, desde el kernel, a los periféricos FPGA por medio del *bridge*. Todo lo antes mencionado está resumido en la figura 3.
Para lograr la arquitectura, el Qsys[2] requiere utilizar los siguientes componentes:

- Arria V/Cyclone V Hard Processor System
- Nios II Processor
- Clock Source
- Avalon-MM Pipeline Bridge
- JTAG UART
- System ID Peripheral
- Interval Timer
- On-Chip Memory (RAM or ROM)
- PIO (Parallel I/O) x3(Leds, Switches y Keys)
- SPI (3 Wire Serial)
- Altera PLL
- Character Buffer for VGA display
- Dual-Clock FIFO
- VGA Controller
- Contador (IP propio)
- Divisor Clock (IP propio)

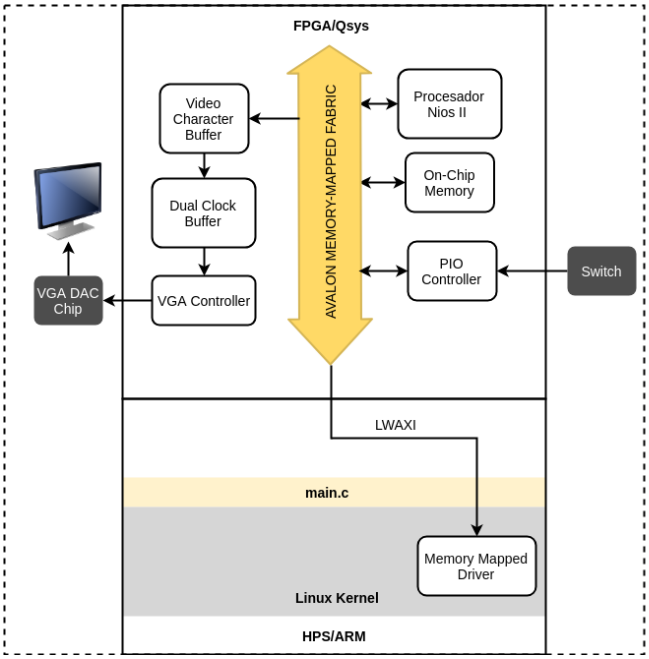


Figura 3: *Diagrama de bloques de la arquitectura*

[2]Qsys es una herramienta de integración del sistema en el software Quartus® Prime que ahorra tiempo y esfuerzo en el proceso de diseño de FPGA.

Existen dos componentes IP propias que muestran el tiempo de ejecución de las consultas por medio de un reloj de 1Hz. Los demás bloques se pueden obtener directamente del catálogo IP del Qsys.

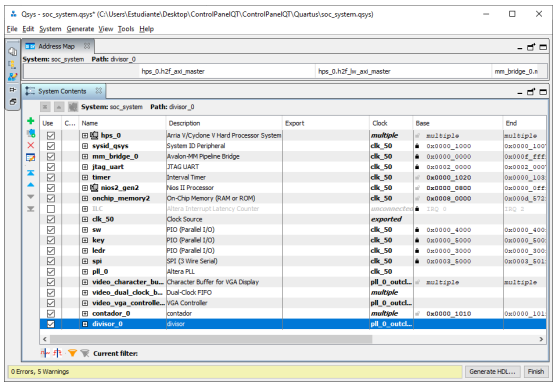
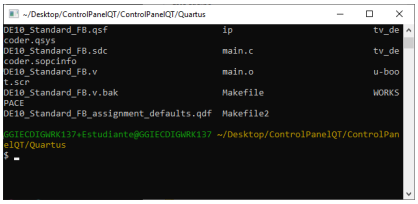


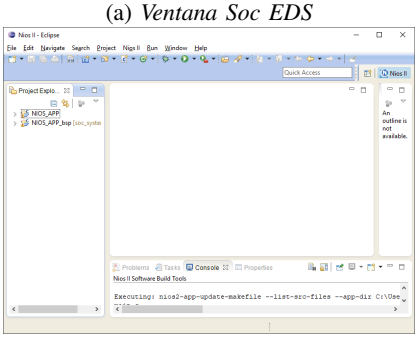
Figura 4: *Ventana Qsys con arquitectura*

II-C. Software

Existen tres programas en C que se ejecutan en nuestro proyecto:



(a) *Ventana Soc EDS*



(b) *Ventana Eclipse*

Figura 5: *Herramientas utilizadas*

- Un programa que se ejecuta en la **FPGA** por medio del Nios II, que se encarga de leer en memoria y mostrar por pantalla lo que nos mande el HPS, interactúa con los *switches* para decidir cuando mostrar la consulta por pantalla.
- Dos programa que se ejecuta en el **HPS**. Uno realiza la consulta a la base de datos y el resultado lo guarda en un archivo de texto, y el segundo se encarga de leer esos archivos y guardarlos en la memoria *On-Chip* de la FPGA, además interactúa con los *switches* para decidir en que momento realizar una determinada consulta y almacenarla.

El programa de la FPGA se carga en el Nios II utilizando la herramienta *Nios II Software Build Tools for Eclipse*, mientras que en el HPS se lo realiza localmente con la herramienta *SoC EDS Command Shell* y se pasa el ejecutable por protocolo SSH.

Programa FPGA: El programa en C que se realizó en la FPGA utiliza la siguiente librería:

```
#include "altera_up_avalon_video_character_buffer_with_dma.
    h"
```

que permite comunicarse con el *Character Buffer* para imprimir datos a la pantalla.
Para cada consulta, la FPGA si se activa el *switch* SW9, mostrará por pantalla utilizando los siguientes métodos:

```
if(SW_value&512){
 alt_up_char_buffer_clear(dev1);
 consulta_val = IORD(memoria,0x000a0000);
 sprintf(read, "%d", consulta_val);
 alt_up_char_buffer_string(dev1, "Mostrando consulta:",
   0,1);
 alt_up_char_buffer_string(dev1, "TABLA: sensor_temp1:",
   0,3);
 alt_up_char_buffer_string(dev1, "Temperatura (ultimo
   registro):", 0,4);
 alt_up_char_buffer_string(dev1, read, 50,4);
 alt_up_char_buffer_string(dev1, "En el segundo: ", 0,47)
   ;
 sprintf(time, "%d", *(contador+2));
 alt_up_char_buffer_string(dev1, time, 17,47);
 band = 1;
}
```

Se observa que lee la información de la *On-Chip* por medio de la función `IORD`. Y muestra las diferentes líneas con el método `alt_up_char_buffer_string`, que se le manda la cadena de caracteres a mostrar con su ubicación por medio de unas coordenadas en x,y.

Programa HPS: El programa en C que se realizó en el HPS es generado por un makefile que importa las librerías requeridas para el proyecto:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "hwlib.h"
#include "socal/socal.h"
#include "socal/hps.h"
#include "socal/alt_gpio.h"
#include "hps_0.h"
```

El script lee los valores de los switches para determinar que combinación ha sido accionada, de acuerdo a eso dependerá del valor que guarde de la consulta, que es generada por un script de bash que ejecuta otro C que contiene el acceso a la base y la sentencia a consultar, y se lo lee por medio de una función que accede al archivo de texto generado con el resultado de la consulta.

```
while(1){
sw_value=*(uint32_t *)sw_addr;
usleep(1000000);
printf("%u\n",sw_value);
```

```
if(sw_value==1){
 consulta = 1;
 system("./mostrar.sh");
 valor = leerConsulta(consulta);
 //printf("%p\n",&on_chip);
 printf("%d\n",valor);
 if (valor == -1){
  valor = 0; //Valido error de archivo
 }
 *(uint32_t *)on_chip=valor;
```

III. Análisis de Resultados

El programa en HPS consiguió acceder a los periféricos de la FPGA, mostrando sus valores y los cambios cuando eran accionados. El programa en la FPGA, imprime los valores establecidos de acuerdo al accionar del *switch* SW9 y la lectura de la memoria y su impresión en pantalla por medio del protocolo VGA.
El script que hace la consulta a la base, devolvían los valores mostrados en la tabla y son guardados en un archivo de texto. El programa en HPS es capaz de leer los valores del archivo resultado de la consulta, y almacenado en memoria para ser mostrado por pantalla por la FPGA.
Existen inconvenientes al querer acceder o almacenar a más

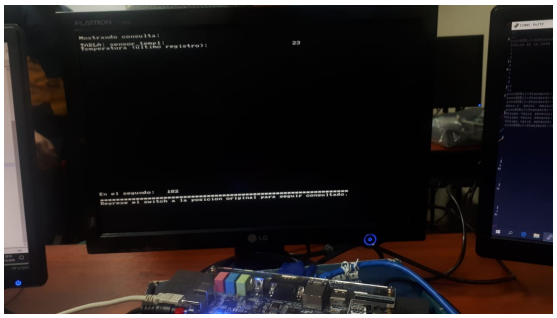


Figura 6: *Salida por pantalla por parte de la FPGA*

valores de la memoria compartida, ya que las direcciones no coinciden entre el HPS y la FPGA en su totalidad. Se debe a que la arquitectura difiere un poco del archivo *header* del script del HPS; por lo que para lograr varias consultas de varios datos a la vez, se necesita corregir y generar un *header* de acuerdo a la nueva arquitectura. De esta forma tenemos más datos para enviarle a la FPGA por parte del HPS y de esa manera volver más dinámica la consulta.

IV. Conclusiones

- Se implementó un sistema embebido que accede a base de datos por medio de un *bridge* entre una arquitectura FPGA y HPS.
- Se requirió generar un *Device Tree* para realizar el puente por medio de comandos no disponibles en el manual.
- Las consultas a la base de datos requiere más manejo de memoria para poder realizar múltiples consultas a la vez.
- Es importante entender como se guarda la información en la memoria *On-Chip* para poder acceder a ella y no generar mal funcionamiento de la FPGA.

Referencias

[1] Roberto Fernandez Molanes, Filipe Salgado, Jose Farina, Juan Rodriguez, “Characterization of FPGA-Master ARM Communication Delays in Cyclone V Devices ”, 2015.
[2] Terasic, “DE10’Standard QT Control Panel Manual”, 2017.
[3] Terasic, “DE10’Standard User Manual”, 2017.
[4] GitHub: robertofem, “Angstrom 2013.12”, https://github.com/robertofem/CycloneVSoC-examples/tree/master/SD-operating-system/Angstrom-v2013.12 2013.
[5] Altera Corporation, “Cyclone V Device Handbook”, 2014.