Lab Assignment 1

Digital Interfacing

Luke Jaffe
jaffe.lu@husky.neu.edu

Brian Crafton
crafton.b@husky.neu.edu

Submit Date: 02/02/2016
Due Date: 02/02/2016

Instructor: Prof. Gunar Schirner
TA: Jinghan Zhang

Abstract

The purpose of this lab is to setup the Zynq-7000 board for future use, refresh C skills, and learn about common concepts and peripherals in microprocessors. In this lab, concepts used include user I/O, GPIO pins, LEDs, and switches. Each of the assignments builds off of the previous assignment, using what has been established to do more complicated things.

Introduction

Any time that one begins using a new environment or tool, it is important to start with a basic project so that a starting point for future projects is established. In this lab, several starting points are established. First off, the Xilinx SDK is setup in Eclipse IDE. This process is straight-forward and it is described in a document provided. After downloading the workspace and setting Eclipse's default directory to workspace Xilinx SDK is ready to use in Eclipse. The next thing introduced was functions belonging to the FreeRTOS api. Functions' prototypes and their descriptions are listed below.

```
BaseType_t xTaskCreate(
      TaskFunction_t pvTaskCode,
      const char * const pcName,
       unsigned short usStackDepth,
      void *pvParameters,
      UBaseType_t uxPriority,
      TaskHandle_t *pvCreatedTask );
```

Although it takes several parameters the only one that we changed from the template code was 'TaskFunction_t pvTaskCode'. This parameter is a pointer to a task entry function. When calling 'xTaskCreate' a pointer to a task we had implemented would be passed so that it could be run. The next function:

```
void vTaskStartScheduler( void );
```

This function takes no parameters. It starts the RTOS scheduler and starts to schedule tasks that have been created. This function is called after all tasks are created.

One more concept that was covered was writing to the memory mapped GPIO. Because each GPIO is mapped to a memory address, the programmer can simply write a value to the GPIO as if it were a memory address. This makes reading and writing the GPIO pins easy and straight-forward. After looking up the address in documentation, one can simply make a constant pointer to the GPIOs address and reference it as if it were a memory location. The lab assumes that the student has done the prelab and understands how to read and write GPIO pins, follow the protocol described in the Xilinx SDK setup documentation, and understand a few functions used from the FreeRTOS api.

Lab Materials

- Hardware
  - Xilinx Zynq®-7000 All Programmable SoC
- Software
  - FreeRTOS
- Environment
  - Eclipse IDE
  - Xilinx SDK

Results and Analysis

I. Temperature Conversion

For terminal interaction, it was necessary to configure port COM4 to terminal I/O at 115,200 baud.

To get a temperature as user input, we used the fgets() function as follows:

```
      char tempf[10];
      fgets(tempf,10,stdin);
```

To convert the user input temperature from string format to float format, we used the sscanf() function:

```
      float f = 0;
      sscanf(tempf, "%f", &f);
```

Finally, converting temperature from celsius to Fahrenheit was done with the following simple code:

```
      float c = (f - 32.0)*(5.0/9.0);
```

II. GPIO PWM

First, we found the memory addresses for data, output enable, and direction mode of GPIO bank 2. We dereferenced the addresses of output enable and direction mode registers and set the LED bits high for output mode:

```
      *gpio_oen_ptr = 0xFF;
      *gpio_dirm_ptr = 0xFF;
```

Then, we toggled the LEDs in an infinite loop by switching them on and off with some simple logic:

```
      if (led_state) //led_state is unsigned int
            *gpio_data_ptr = (0xFF);
      else
            *gpio_data_ptr = (0x00);
      led_state = led_state ? 0 : 1;
```

Finally, to create a delay in the loop after each toggling, we used the task delay function:

```
      const TickType_t xDelay = 500 / portTICK_PERIOD_MS;
      vTaskDelay(xDelay); //delays for 500ms
```

III. GPIO Counter

We acquired user input with the same method as in part I. In this case, the “number to count to” was requested from the user. To treat the LEDs as a binary bit string representing the current counter value, we simply dereferenced the GPIO bank 2 data register, setting it to this value:

```
      int i;
      for(i=0; i<=user_input;i++)
      {
            *gpio_data_ptr = i; //this is the pointer to the data register
            vTaskDelay(xDelay);
      }
```

IV. GPIO Push Button

For this problem, we needed to initialize the LEDs to output, and the switches to input:

```
	*gpio_oen_ptr = 0x000000FF;
	*gpio_dirm_ptr = 0x000000FF;
```

Setting the LEDs to match the value of the corresponding switches requires just one line of code. We just take the switch values in the data register and shift them 8 bits right, then set the LED values in the data register to this value:

```
	(*gpio_data_ptr) = (((*gpio_data_ptr)>>8) & 0xFF);
```

Printing output to console every time an LED toggles requires a loop. We check each of the led bits against each of the switch bits. If they are equal, then no change has happened in this polling round. If they are different, then we check if the switch bit is high or low, and print that the LED has switched on or off respectively:

```
	int i, j;
	for (i = 0; i < 8; i++)
	{
		j = i + 8;
		int led_bit = (*gpio_data_ptr) & (0x01<<i);
		int switch_bit = (*gpio_data_ptr) & (0x01<<j);
		switch_bit = switch_bit >> 8;
		if (led_bit ^ switch_bit)
		{
			if (switch_bit) printf("LED%d is now ON\n", i);
			else printf("LED%d is now OFF\n", i);
		}
	}
```

Post-Lab Questions

1. The LEDs appear less bright when there is no delay because they are instantaneously cycling between the on and off states (as fast as the processor, hardware can manage). If we assume that it takes equal time to turn them on and off, then they are on approximately 50% of the time, creating an illusion of reduced brightness.

2. In the Zynq manual there is a reference to a profiling tool (OProfile). This makes it possible to measure the execution time of functions, percent of time spent in a function, and other useful features. Another way of profiling a function would be to connect a GPIO pin to an oscilloscope. At the start of the function set the pin high and at the end of the function set it low. On the oscilloscope it is possible to see the time between the logical 1 and logical 0 and that would be the execution time of the function.

Conclusion

In this lab common concepts in microprocessors were implemented using GPIO pins, LEDs, and switches. The lab went as expected, after implementing user I/O and GPIO each part of the lab was straightforward and used basic concepts in programming to execute the assignment. The work done and concepts learned in Lab 1 will be used in future labs. This will allow for more complex and useful assignments.

References

[1] Xilinx, Zynq-7000 All Programmable SoC Software Developers Guide, v12.0 09/30/2015
http://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf

[2] Xilinx, Zynq-7000 All Programmable SoC Technical Reference Manual, v1.10, 02/23/2015
http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf