Lab 6 report – FreeRTOS on Raspberry Pi

Student name: Vu Nguyen (71141)

Date: 17.03.2015

Assumptions on knowledge of the reader

It is assumed that a reader should know some basic programming skills in C/C++ in order to understand the source code. A basic knowledge of hardware such as processors, LEDs, pin ports, kernel booting process, and basic real-time kernel should be known. Also, basic usage of Linux command lines is required in order to replicate the lab.

Description of the task

Lab 6 consists of two tasks. Task 1 serves the purpose of board bring-up. A demo application available in the FreeRTOS port for Raspberry Pi will be built and installed for the Raspberry Pi. The Demo program starts two tasks. One task turns on an LED while the other task turns of the LED.

Task 2 of the lab consists of two subtasks: tasks 2a and 2b. Task 2a requires to create a system based on the demo application where pressing a button will turn on an LED. Task 2b requires to create a system that counts button presses since the Raspberry Pi booted, and shows the counter by repeating flashing the LED the number of times the button has been pressed.

Description of the equipment used

Equipment used in the lab consists of the following

Raspberry Pi (x1)
SD card (x1)
Power supply (x1)
Daughterboard with 3 buttons, 3 LEDs, TTL serial port – USB (x1)
SD card reader (x1)
Linux computer (x1)

Description of the performed work

Task 1 - Board bring-up

Task 1 was followed from the instructions shown in James Walmsley's github page [2]. First the arm-none-eabi- toolchain [3] and the FreeRTOS port for Raspberry Pi are downloaded [2]. In order to build the demo application, a Makefile is needed. This Makefile needs to be modified so that library locations can be found. The path to the library location needs to be modified depending on the library location installed in the user's machine.

```
kernel.elf: LDFLAGS += -L"c:/yagarto/lib/gcc/arm-none-eabi/4.7.1/" -lgcc
```

```
kernel.elf: LDFLAGS += -L"c:/yagarto/arm-none-eabi/lib/" -lc
```

Next, make command is typed in Linux to compile the demo application. The result of the make is a kernel.img file. After that the kernel.img file in SD card is replaced with the kernel.img file from the FreeRTOS build system. Once that process is done, the SD card is plugged into the SD card slot of the Raspberry Pi, and the power supply is given to the Raspberry Pi.

Task 2a – LED blinking when push button pressed

In task 2a, the demo application is modified so that when a button pressed, the LED on the expansion board starts blinking. The steps of compiling the source files and copying the compiled kernel.img file to the SD card are done in the same way as in task 1. The modified code is as follows:

```
#include <FreeRTOS.h>
#include <task.h>

#include "Drivers/interrupts.h"
#include "Drivers/gpio.h"

//LEDs on the Raspberry Pi's daughterboard
#define LED_RED 22
#define LED_YELLOW 21
#define LED_GREEN 17

//Buttons on the Raspberry Pi's daughterboard
#define BUTTON_MIDDLE 9           //This pin is configured as pull-up
#define BUTTON_MIDDLE_PRESSED 0
#define BUTTON_MIDDLE_RELEASED 1

//Global variables
int button_flag_g = 0;

void task1(void *pParam) {
  while(1) {
    if (ReadGpio(BUTTON_MIDDLE) == BUTTON_MIDDLE_PRESSED) {
      button_flag_g = 1;
    } else {
      button_flag_g = 0;
    }
  }
}

void task2(void *pParam) {
  while(1) {
    if (button_flag_g) {
      //button flag is set, turn on led
      SetGpio(LED_RED, 1);
    } else {
      SetGpio(LED_RED, 0);
    }
  }
}
```

```
void main(void) {
	DisableInterrupts();
	InitInterruptController();

	//Set red LED pin (GPIO pin 16) an output pin
	SetGpioDirection(LED_RED, GPIO_OUT);

	//Set the middle button an input pin
	SetGpioDirection(BUTTON_MIDDLE, GPIO_IN);

	xTaskCreate(task1, "button", 128, NULL, 0, NULL);
	xTaskCreate(task2, "led", 128, NULL, 0, NULL);

	vTaskStartScheduler();

	/*
	 *	We should never get here, but just in case something goes wrong,
	 *	we'll place the CPU into a safe loop.
	 */
	while(1) {
		;
	}
}
```

Listing 1. Implementation of application for task 2b

The application has two tasks. Task1 is responsible for checking the status of the middle button on the expansion board. If this button is pressed, *button_flag_g* will be set. If not, it will be cleared. Task 2 turns on the red LED on the expansion board depending on the *button_flag_g* flag.

Task 2 b – LED blinking with the number of button presses

In this task, the number of led blinking is determined based on the number of button presses. The demo application is again modified. There are two tasks in the application. Task 1 counts the number of button presses. Every time a button is pressed and released, the button_press_count variable will be incremented by 1. In order to avoid button bouncing problem, a delay of 100 ticks is used. The code snippet for task 1 is shown in Listing 2.

```
//Global variables
unsigned int button_press_count = 0;

void task1(void *pParam) {
  unsigned int buttonState = BUTTON_MIDDLE_RELEASED;
  unsigned int buttonLastState = BUTTON_MIDDLE_RELEASED;

  while(1) {
    buttonState = ReadGpio(BUTTON_MIDDLE);
    if (buttonState != buttonLastState) {
      button_press_count++;
      buttonLastState = buttonState;
      vTaskDelay(100);
    }
```

```
  }
}
```

Listing 2. Code snippet for task 1 of task 2b

Task 2 in the application blink the red LED on the expansion board with a frequency of 1s. To distinguish between the on/off blinking of the LED and the number of button pressed. A delay of 1000 ticks is used. For example, if the button has been pressed three times from the beginning, the LED will blink with a frequency of 1s for three times. After that it stops blinking for 1000 ticks and starts repeating the pattern over and over again. Listing 3 shows the code snippet for task 2

```
void task2(void *pParam) {
  unsigned int i = 0;

  while(1) {

    for (i = 0; i < button_press_count; i++) {
      SetGpio(LED_RED, 1);
      vTaskDelay(500);
      SetGpio(LED_RED, 0);
      vTaskDelay(500);
    }

    vTaskDelay(1000);
  }
}
```

Listing 3. Code snipped for task 2 of task 2b

Achieved results

As for task 1, the board bring-up is successful. The red LED on the expansion blinks. Regarding task 2a, it works perfectly. When the middle button on the expansion board is pressed, the red LED starts blinking. When the button is not pressed, the red LED stops blinking. Task 2b does not work properly since there are problems with button bouncing. When a button is pressed once, the counter is incremented by 1 for several times. This results in the led blinking with incorrect number of times. One possible solution to this problem is to use a Schmitt trigger circuit for debouncing the button.

References

[1] https://github.com/jameswalmsley/RaspberryPi-FreeRTOS/blob/master/Demo/main.c
Accessed 17 March 2015

[2] https://github.com/jameswalmsley/RaspberryPi-FreeRTOS
Accessed 17 March 2015

[3] https://launchpad.net/gcc-arm-embedded
Accessed 17 March 2015