Greg Paczkowski
EE4341
Thomas Posbergh
Due:3/20/20

LAB REPORT 3: Real Time Operating System

Abstract

Using FreeRTOS and the library for the SPI accelerometer files, The buttons on the PIC32 were programmed to display the X axis value, Y axis value and Button 3 corresponding to the different buttons. This was accomplished by setting up the PIC32 with its initialization functions for the SPI serial communication. An initial task is created that handles the setup and getting reading from the accelerometer. The readings are sent to the different tasks to be displayed if scheduled by the kernel. Three tasks, app1, app2, app3 display X axis readings, Y axis reading, and whether button 3 has been pressed respective to their buttons and tasks.

Introduction

In this lab, FreeRTOS was introduced as the real time operating system that is going to be developed upon. FreeRTOS is an open source kernel and libraries under the MIT open source license. The OS supports multiple architectures under one code base. MPLAB Harmony is used in addition to FreeRTOS to add to the abstraction and help create a modular framework for ease of use. This framework adds app files that are used as tasks that are used in the MPLAB Harmony application. These files hold logic for the application's state machine and can call drivers and other services as needed.

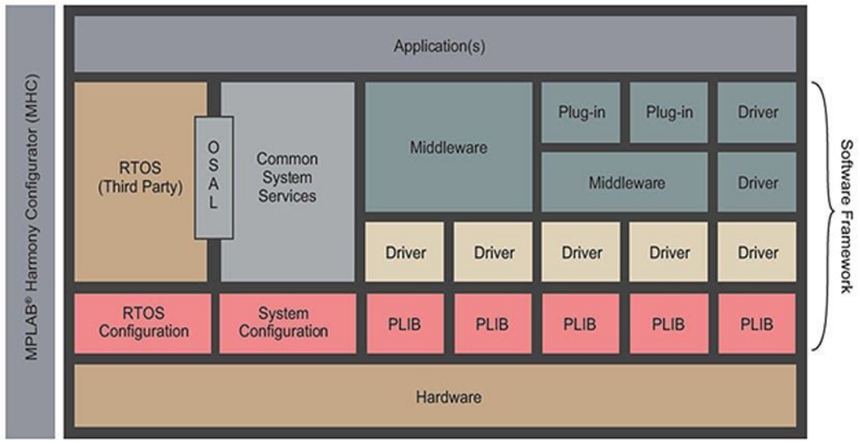

Image taken from Microchip Harmony V2.[1]

The image above illustrates the resources provided by the framework. The RTOS(Third Party) block is where FreeRTOS kernel and other libraries reside. Four app files are used to run separate tasks that are controlled by the kernel and run at the schedulers discretion. It is the schedulers job to make sure all tasks are run in a timely manner. FreeRTOS uses a priority scheduler with Round Robin approach for those of the same priority.[2] The goal of the two weeks of the lab were to have different tasks run serial peripheral interface communication with the accelerometer, LIS3DH which has been used in the lab. This also uses UART and the RS232 connector to display the reading on the computer through PuTTY.

Procedure

The first week was spent setting up the Harmony software and creating the app files to send simple values to the Terminal in PuTTY. FreeRTOS was installed with the installation of Harmony. The app files were already preconfigured to an extent but needed values replaced with variable declarations and initializations. A queue was created to send values. Two variables were put in the Queue and sent to the other tasks to print using the xQueueSend function. A vTaskDelay function is also used for the tasks to wait for the other task to retrieve the value from the queue. The other tasks, app1 and app2, had to be initialized with UART to print the values to PuTTY. The initialization functions were taken from code previously written because the values and ports used are identical. Initialization only had to occur in one file for it to be initialized for every task. App2 file was also identical to App1 file but without the UART initialization and with the added vTaskDelay function to have it properly wait for app1 to retrieve the data it needs to print. App3 was not used in week one but as a heartbeat debugging LED. If there was a major bug, app3 would not run thus the LED would not blink. After the code for sending variables which were the character ‘a’ and ‘b’ were printed and written, week one was concluded.

The second and final week was to display reading from the accelerometer. This required the use of a library previously made for the SPI accelerometer. The first set was setting up the app file which is used to get accelerometer data and send it to other tasks through the queue to the UART RS232 connector and displayed on PuTTY. This file called required setup functions and created queues:

```
void APP_Initialize ( void ) {
   xQueue1 = xQueueCreate( 10, sizeof( unsigned long ) );
   xQueue2 = xQueueCreate( 10, sizeof( unsigned long ) );
   io_setup();
   uart1_setup();
   spi2_setup();
   accel_setup();
}
```

Two queues were used to send the separate X and Y axis values. Two other functions were created to get the SPI reading.These called the registers from the accelerometer. The value sent was set inside the function. The function to read the X axis value is below:

```
void RTOS_SPI2_READ_X(void) {
   int16_t X_H; int16_t X_L; int16_t X;
   X_H = spi2_read_register(0x29);
   X_L = spi2_read_register(0x28);
   X = X_H << 8; // Combine data from both registers
   X = X | X_L; // See Lab2 manual for instructions
   ulValueToSend1 = (float)(0.000061f * X);
}
```

In the main APP_Tasks function the functions were called for accelerometer data and values were sent through the queue and a task delay was called. The app1 and app2 files only required the value to be printed and turn on the LED. App1 example after getting value from queue:

```
if (!BUTTON1) {
        LED1 = 1;
        printf("B1 x-axis: %f\r\n", ulReceivedValue);
} else {
        LED1 = 0;
}
```

The variables LED1 and BUTTON1 were declared in the io_setup.h file and u1ReceivedValue was created to get the value out of the queue. App3 was the simplest file only requiring a string to be sent to PuTTY. The code required for app3 to display BUTTON3 in APP3_Tasks function shown below:

```
if (!BUTTON3) {
        LED3 = 1;
        printf("BUTTON3 \r\n");
} else {
        LED3 = 0;
}
```

To get these files to work, io_setup.h, io_setup.c and x500_accelerometer_spi.h were copied into the project. These were necessary for variable calls and function calls. After debugging and correct output was verified, week two concluded.

Conclusion/Summary

This lab introduced the real time operating system, FreeRTOS. The purpose of the lab was to get exposure to the use of the operating system as well as its quirks. We were required to send values through SPI and UART and display reading from an accelerometer. There weren't many mistakes made or errors besides minor debugging errors. Two queues were found to be easier to send values instead of one queue because the values are constantly being updated by app.c and the APP_Tasks function. To get values, axis was sent through one queue and the other axis was sent through the second queue and to their respective tasks.

References:

[1] https://www.microchip.com/mplab/mplab-harmony/mplab-harmony-v2

[2] https://freertos.org/FAQSched.html