Lab 4 Report

ECSE 426 – Microprocessor Systems

Group 7

Gregoire Martin – 260364773

Patrick White – 260353491

November 13th 2013

# Table of Contents

[1. Abstract 3](#_Toc369952480)

[2. Problem Statement 3](#_Toc369952481)

[3. Theory and Hypothesis 4](#_Toc369952482)

[4. Implementation 6](#_Toc369952483)

[5. Testing and Observations 10](#_Toc369952484)

[6. Conclusion 12](#_Toc369952485)

**No table of figures entries found.**

# 1. Abstract

The primary goal of this lab is to implement a multithreaded system that tracks the temperature of the STM32F4 board, tracks the tilt angles of the board, and performs two different pulse width modulation (PWM) algorithms and to use the board’s LEDs to provide a simple graphical output to the user for any given mode. The four modes are grouped into 2 main modes, one equivalent to Lab 2 and one equivalent to Lab 3. Finally, the user must be able to switch between modes of operation by either pressing a button or by tapping the board. In order to achieve these functionalities, this lab involves the use of a Real-Time Operating System (RTOS) and its associated characteristics such as multithreading and synchronization.

# 2. Problem Statement

The main goal of this experiment is to provide two main modes of operation, each with two sub modes, using the STM32F4 Discovery Board and also using a RTOS approach. The problem can be broken down into six parts.



* Using the LEDs to display the board’s temperature trend
  + Temperature data must be collected from the temperature sensor
  + Data must be filtered to reduce noise in the signal
  + LEDs must be turned on and off in a clockwise/counterclockwise sequence depending on if the board’s temperature is rising or falling
* Using the LEDs to display the board’s tilt angles
  + The LEDs must be brighter when the tilt angle of the board is higher
  + Must be done for both pitch and roll angles
* Displaying two different hardware based PWM algorithms using the board’s LEDs
  + Must flash the LEDs progressively brighter periodically
  + Have one PWM linked to accelerometer function, and the other to the temperature tracking function to provide the other respective sub modes of the system
* Providing the user with a way of switching between the two main modes by tapping on the board
  + Must switch to the other main mode whenever a tap is detected
  + Switches to accelerometer mode or temperature tracking mode depending on the current mode
* Providing the user with a way of switching between the two submodes by pressing a button on the board
  + Must be able to switch between PWM and either accelerometer mode or temperature tracking mode depending on the current main mode that the system is in
* Implementing the above functionalities using an RTOS approach
  + Must be able to guarantee safety of system by regulating access to shared resources and variables
  + Must ensure correct sampling rates

The system must be able to follow the behaviour described in Table 1.

Table 1: Next mode as a function of current mode and inputs

|  |  |  |  |
| --- | --- | --- | --- |
| Current Mode | Tap Detected | Button Press Detected | Next Mode |
| Accelerometer | N | N | Accelerometer |
| Accelerometer | N | Y | PWM 1 |
| Accelerometer | Y | N | Temperature |
| Accelerometer | Y | Y | PWM 1 |
| PWM 1 | N | N | PWM 1 |
| PWM 1 | N | Y | Accelerometer |
| PWM 1 | Y | N | Temperature |
| PWM 1 | Y | Y | Accelerometer |
| Temperature | N | N | Temperature |
| Temperature | N | Y | PWM 2 |
| Temperature | Y | N | Accelerometer |
| Temperature | Y | Y | PWM 2 |
| PWM 2 | N | N | PWM 2 |
| PWM 2 | N | Y | Temperature |
| PWM 2 | Y | N | Accelerometer |
| PWM 2 | Y | Y | Temperature |

# 3. Theory and Hypothesis

The most basic unit of the RTOS is the thread. Threads are independent sequences of execution. Each thread executes a certain set of instructions independently from the instructions of all the other threads. The OS has its own algorithm for scheduling the order of execution for threads. Threads allow multiple sets of instructions to be interleaved together to make it appear such that they’re all occurring at the same time. This is very useful for the case where multiple sets of computation need to occur all at the same time.

A problem arises when threads need to access or modify data that is common to multiple threads. Because many instructions are not immutable, it means that data corruption can occur if different threads are attempting to access the shared resources at the same time. The solution to this is to ensure that only one thread can access a shared resource at any given time. To do this, a mutex is employed. A mutex is an OS service that can force threads to wait while another thread holds a variable that another thread needs. The set of instructions where a thread needs a shared variable is called the critical section. To ensure proper execution, each thread waits for a mutex to be free before entering their critical section, and releases the mutex when they have completed their critical section. If a mutex is available when the thread attemps to enter the critical section, the thread will acquire the mutex and execute the critical section. Finally, it will release the mutex after execution of the critical section is complete. If the mutex is not available when the thread attempts to enter the critical section, the thread will sleep until the mutex is released.



To ensure proper sampling rates and mode switches, timers and tap detection their interrupts must be used. An Interrupt service routine (ISR) occurs whenever an interrupt occurs. It is a set of instructions that is executed whenever a given interrupt occurs. The purpose of the ISR is to handle the interrupt and provide the rest of the system with the information it needs from the interrupt. Some threads need to suspend execution while they wait for an interrupt, and to continue executing after the interrupt has occurred. Threads need to be able to know precisely when the interrupt has occurred. The solution to this problem is to use an OS signal. The signal can be sent from a specific ISR to a specific thread to indicate that the proper interrupt has occurred and that the thread can continue executing. In order to do this, the In this case, the ISR can set a signal to indicate to the required threads that the required interrupt has occurred in order to inform the other threads that they may continue execution.



Talk very quickly about accelerometer/temperature/PWM but put reference to other lab reports for their theory

# 4. Implementation



# 5. Testing and Observations

# 6. Conclusion

# Appendix