Experiment #4

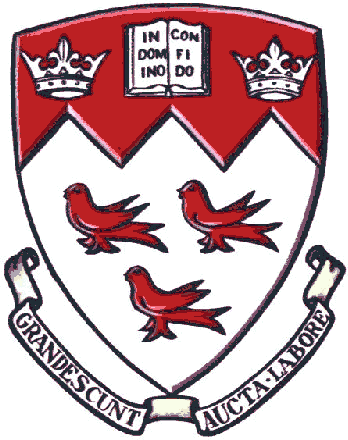
Multithreaded, interrupt-driven sensor reading and peripheral control

Maxim Goukhshtein ID: 260429739

Olivier Laforest ID:260469066

Department of Electrical and Computer Engineering

McGill University, Montreal



March 23rd, 2015

Group 3

ECSE 426 Microprocessor Systems

# Abstract

The goal of the experiment presented in this report was to design and implement a multithreaded system using the real time operating system (RTOS) CMSIS-RTOS, capable of sensing the STM32F407 Discovery board's processor temperature and pitch angle and provide a visual display of those readings to the user. The report will show how a 4x4 alphanumeric keypad was use to provide the user a mean to select the desired mode of operation and how a 4 digits 7-segments display in combination with the on-board LEDs of the board allowed a way to provide a visual feedback to the user. To that end, the work of experiment 2 and 3 were slightly modified and successfully combined using threads in an RTOS environment on the discovery board. This report will show how the thread implementation was done such that the concurrent multithread computation operates safely.

# Problem Statement

The goal of this experiment is to design and implement a system which can concurrently sense both the current processor's temperature and either the pitch or roll angle of the STM32F407 Discovery board. To that end, the on chip temperature sensor of the discovery board's processor is to be used in order to measure the processor's temperature and the off chip tri-axial MEMS Accelerometer sensor LIS3DSH is to be used to measure the gravitational acceleration in order to compute the board's pitch or roll angle. The system needs to provide the user with a mean to input the desired mode of operation through the use of a 4x4 alphanumeric keypad. The system should also provide visual feedback using the 4 digits 7-segments display as well as the on-board LEDs of the discovery board. There are two possible modes of operation in which the system may find itself in, temperature mode or accelerometer mode. In temperature mode, the system needs to display the real time current temperature of the board's processor in degrees Celsius on the 7-segments display. If the temperature exceeds an overheating threshold, the 7-segments display should flash on and off repeatedly as long as the processor's temperature remains above the threshold to denote danger level. In accelerometer mode, the system needs to display the real time current pitch or roll angle of the board on the 7-segment display and will allow the user to select one of the four user LEDS on the board and adjust the LED brightness using pulse width modulation (PWM) according to the board's angle. The LED should be completely off when the board's angle is and gradually get brighter as the board's angle gets wider all the way up to where it should be the brightest. The alphanumeric keypad's keys 1 to 4 should allow the user to select which one of the four user LEDs is to be lit at the current time. The user also needs to be able to switch from temperature mode to accelerometer mode and the other way around by pressing preselected key(s) on the keypad. Note that while in accelerometer mode, if the board's processor temperature exceeds the overheating threshold, the 7-segments display should also be flashing on and off to denote danger level. To allow for concurrent measurement of the processor's temperature and the board's tilt angle, the system should be a multithreaded system that uses CMSIS-RTOS. The work from experiments 2 and 3 should be modified and incorporated into threads in order to achieve the desired system and therefore, the requirements regarding sampling rates of the sensors, calibration and filtering are the same as those found in those respective experiments.

# Theory and Hypothesis

This experiment combines the work done in lab 2 and 3 into a multithreaded sensing system which concurrently performs measurements of the discovery board's temperature and tilt angle, displays the appropriate information through a four digits 7-segments display and on board LEDs, and records user inputs through a 4x4 alphanumeric keypad. The theory for the operation of the temperature sensor, ADC conversion and operation, MEMS sensor usage and calibration, Kalman filters, etc, has been explained in the theory sections of experiments 1 to 3 lab reports and will therefore not be covered here.

## Real-Time Operating Systems (RTOS)

Real-time operating systems (RTOS) make embedded programming similar to desktop programming and they usually include a large amount of pre-made services such as process management, file management, memory management, date and time, user management, networking, etc [1]. They allow real-time operation which means that they process information coming from sensors connected to it in real-time, or with minimum delay. RTOS can be separated into two categories, soft and hard time RTOS. In soft time RTOS, the operating system generally meets a deadline and in hard time RTOS, the system deterministically meet the deadline [2]. RTOS allow multi-tasking to take place through the use of threads. They provide tools to synchronize those threads and utilities to share resources among them. The CMSIS-RTOS is an API which provides a generic interface all ARM Cortex-M processor devices [3]. The CMSIS-RTOS API interfaces with an existing real-time Kernel and provides several attributes and functionalities which are listed below [3]. Each thread may be in one of the following four states.

### Threads

Threads are different processes or tasks which run concurrently within a system/application and may have interdependencies with each other. Systems which runs on single core processor may only run one thread at a time. Such systems need a scheduler, which is a mechanism to decide which thread should currently be running. Each thread possess its own stack and state buffer and each is assigned a priority to allow proper scheduling [2].

* Running: Only one thread at a given time can be in that state on single core systems. The thread in the running state is the tasks which is currently executing.
* Ready: Threads which are ready to be executed are in that state. Once the thread which in the running state terminates, the thread which is ready and has the highest priority will switch to the running state and start execution.
* Waiting: A thread in the waiting state is a thread which is waiting for an event to occur (e.g. signal flag to be set, semaphore or mutex to be released, message or mail to be received, etc.)
* Inactive: Threads which have not been created or terminated are inactive and therefore do not use any system resources [4].

The following figure shows the state transition diagram for threads in CMSIS RTOS.

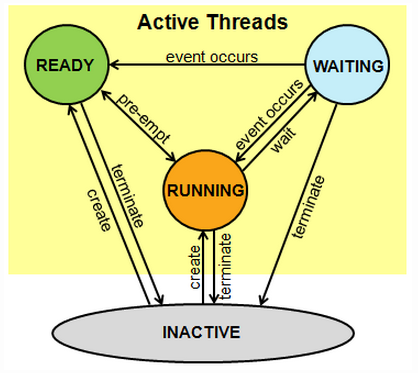


Figure 1: State transition diagram for threads in CMSIS-RTOS

### Scheduler

The CMSIS-RTOS (RTX version) uses a pre-emptive scheduler which reschedule which thread should be running when an event happens. The different possible events can be a resource release, the tick of a timer, arrival of a message or mail, status change of a signal flag, etc [2] [3]. The thread with the highest priority is the one which will be in the running state [2]. If two or more threads have the same priority, the scheduler will have each of them run alternatively following a round-robin scheme. The equal priority tasks will alternate at tick events which are generated by the SysTick which is consequently no longer available to the programmer. The frequency of the tick event can be configured. A low frequency leads to less frequent change in the threads which is currently running and therefore reduces the overhead computational cost of context switching. A high frequency allows each equal priority threads to run more frequently and therefore lead to a more even distribution of computational resources [2]. Since the thread priority dictates which thread is going to be running and when it is going to be running, the assignment of priorities by the designer is very important to ensure the proper operation of the system.

### Communication between threads

Communication between threads can be done via signals, message queues and/or mail queues.

* Signals: Signal functions allow a thread to communicate with other threads by setting, clearing or waiting for signal flags. Signal flags are assigned to each threads [4].
* Message queues: Message queue functions allow to define a memory pool associated with a message queue, send, receive and/or wait for messages. Integers or pointer value may be sent through as messages to a thread or an interrupt service routine [4].
* Mail queues: Mail queue are very similar to message queues except that they are used to send a block of memory instead of an integer or a pointer value [4].

### Synchronization

To allow synchronization between the execution of threads sharing a resource, CMSIS-RTOS provides mutex and semaphore management functions.

**Mutex**

A mutex ensures that a resources which is used by more than one thread can only be accessed by one of those threads at a time. When a thread acquires the resource, this resource become unavailable to other threads until it is released by the thread which is currently using it. The other threads needing the shared resource need to wait in the mean time, which is why threads requiring a shared resource should minimize their critical section (i.e. code requiring the shared resource) as much as possible in order to avoid starving other threads [4]. The following figure illustrates the interaction between the threads and the mutex.

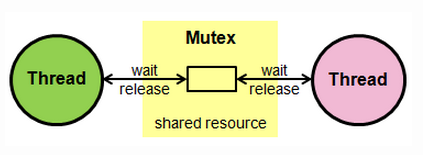


Figure 2: Interaction between threads and a mutex [4]

**Semaphore**

A semaphore works in a very similar way to a mutex, except that they protect the access to shared resources which has several identical instances of the resource (e.g. ports). Therefore, a semaphore has a fix number associated to it which corresponds to the number of instances of the resource available. When a thread gets access to one instance of the resource, it decreases the semaphore. When a thread is done using the shared resource, it releases it and increases the semaphore. When the semaphore reaches 0, no more instances of the resource are available and subsequent threads requiring the shared resource need to wait for one of the instances to be released [4]. The following figure illustrates the interaction between several threads and a semaphore.

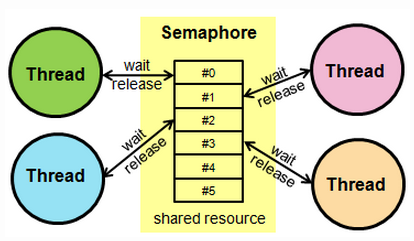


Figure 3: Interaction between several threads and a semaphore [4]

### Timer

The CMSIS-RTOS include several timer management function which allows creating, starting, stopping, restarting, etc, operating system timers. CMSIS-RTOS allows both on-shot and periodic timers to be created [4]. Timers are assigned the highest priority by default, however they can still be interrupted by hardware interrupts and are therefore less accurate than hardware timers. Operating system timers are usually a multiple of the SysTick frequency for the Round-Robin scheduler [2].

### Interrupt Service Routines

The CMSIS-RTOS can use interrupt service routines, however, interrupt handler cannot wait for anything [2]. Consequently, mutexes cannot be used, wait are not permitted in the interrupt service routine and all timeouts in method parameters need to be 0 [4].

# Implementation

# Testing and Observations

# Conclusion

# References

|  |  |
| --- | --- |
| [1] | Z. Zilic, *ECSE 426 System Services and Final Project,* Montreal, QC, Canada: McGill University, 2015. |
| [2] | B. Nahill, A. Suyyagh, *Real-Time Operating Systems,* Montreal, QC, Canada: ECSE 426-Microprocessors Systems Tutorial 4, 2014. |
| [3] | http://www.keil.com/pack/doc/CMSIS/RTOS/html/index.html, "CMCIC-RTOS," ARM Ltd., 20 March 2015. [Online]. [Accessed 23 March 2015]. |
| [4] | "ARMmbed CMSIS RTOS," ARM, [Online]. Available: http://developer.mbed.org/handbook/CMSIS-RTOS. [Accessed 23 March 2015]. |
| [5] | "UM1472 User manual Discovery kit for STM32F407/417 lines," STMicroelectronics, 2014. [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user\_manual/DM00039084.pdf?s\_searchtype=keyword. [Accessed 21 February 2015]. |
| [6] | "RM0090 Reference manual STM32F405xx, STM32F407xx, STM32F415xx, and STM32F417xx advanced ARM-based 32-bit MCUs," STMicroelectronics, 2014. [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference\_manual/DM00031020.pdf. [Accessed 21 February 2015]. |
| [7] | "STM32F405xx/STM32F407xx Datasheet," STMicroelectronics, 2013. [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00037051.pdf. [Accessed 21 February 2015]. |

# Appendix