Lab 4 – Report

Multi-threaded system design for concurrent measurements of temperature from STMF32 Board and accelerometer from LSM9DS1 chipset

Group 7
Hoai Phuoc Truong – 260526454
Ravi Chaganti – 260469339

ECSE 426
Department of Electrical and Computer Engineering
McGill University

6th November, 2015

Table of Contents

1. Abstract

The main objective of this experiment is to implement a multi-threaded RTOS-system based design which will utilize both the temperature and accelerometer sensors of the microprocessor to streamline the incoming measurements and simultaneously display them on the 4-digit seven segment display clock. Concepts of multi-threading and access protection tools for shared resources will be used to optimize performance of the microprocessor.

2. Problem Statement

In our experiment, our primary focus will be to identify key processes which need to be executed concurrently using threads. Proper implementation of thread division must take place in terms of using CMSIS-RTOS primitives including timers, interrupt services requests and relevant OS services.

Previous components and functions which were used in experiments 2 and 3 need to be modified for this lab. Temperature sensor must be configured again using proper ISR implementation (TIM based interrupts). The alarm which was previously based on flashing (circulating) the LEDs must be replaced with a blink-on/off type model that will be displayed directly on the clock display to highlight overheating of the microprocessor. Also, the accelerometer configuration must calculate both roll and pitch after calibrating and filtering the measurements received from the accelerometer sensor (using LSM9DS1).

The keypad must be designed in such a way that the user should be able to choose any of the following readings that must be displayed on the display clock:

1. Roll (measured from the accelerometer sensor)
2. Pitch (measured from the accelerometer sensor)
3. Temperature (measured from the temperature sensor)

The blinking mode which will highlight the overheating of the processor must be enabled for both modes of accelerometer's angles and temperature displays, irrespective of what is being displayed on the clock display.

3. Theory and Hypothesis

3.1 Multi-threading

Multi-threads enable the OS to handle its operations in an efficient way by running programs and tasks concurrently and managing shared resources in an efficient way to enhance performance of the processor in terms of time complexity. Each thread defines a certain task that will be computed by the microprocessor. Since these threads are independent, if an exception occurs in one thread, it does not affect the performance of other threads in any way (Reference 1 and 2 in Appendix).

3.2 Mutex vs Semaphores

Mutex is used to define an access protection for a particular code or program that cannot be executed by more than 1 thread, i.e. it only allows one thread execution at a time and keeping other threads in wait mode until the current thread completes its task and releases the mutex (Reference 3 in Appendix).

Semaphores are used for access protection for multiple threads and multiple shared resources. There are 2 types of semaphores: count (for multiple access protection for multiple shared resources) and binary (blocking and unblocking).

3.3 RTOS

Real time operating system (RTOS) are used for special applications which require real-time predictable response with much lower density of tasks compared to operating systems (OS). Multi-tasking using RTOS (which uses SysTick to run) can be achieved by either one of the following ways: time sharing design or event driven design. RTOS's scheduler, which decides on when to run which thread of program, allows the user to set the priority level for each thread of execution (Reference 4 in Appendix).

3.4 Timer's period equation

To calculate TIM's parameters of prescaler and period, the following equation1 will be used (Reference 5 in Appendix):

$$PWM\_frequency = \frac{\frac{default\_frequency}{prescaler + 1}}{TIM\_Period\ + 1}$$

Equation1: prescaler and period equation

3.5 Serial Peripheral Interface (SPI) protocol

SPI protocol is used to read and write the registers from the external device with the help of 4 wires:

1. Chip Select (CS): low at the start of the transmission signal and high at the end, connected to PB9 on the microprocessor
2. Serial Port Clock (SPC): controlled by SPI master, no transmission when CS is high, connected to PA5 on the microprocessor
3. Serial Port Data Input (SPDI): connected to PA7 on the microprocessor
4. Serial Port Data Output (SPDO): connected to PA6 on the microprocessor

Data is written to the device when bit_0 is 0 (chip will enable SPDI) and data is read from the device when bit_0 is 1 (chip will enable SPDO). It will take 16 clock cycles for the read and write register commands to be completely executed. These connections will be used for LSM9DS1 chipset implementation (Reference 6 in Appendix).

3.6 Other OS Services

Other OS services for threading communication include (Reference 9 in Appendix):

1. Signal event: to generate signals for threads by using osSignalSet
2. Messages: used for queuing purposes (integer or pointer)
3. Mailbox: used for larger blocks of memory

The following are the hypotheses that have been established for this experiment:

1. Measurements from the temperature and accelerometer sensors will be processed concurrently using threads to improve the performance of the processor.
2. SysTick, which was implemented in configuring the temperature sensor, cannot be used as it is exclusively reserved for RTOS.
3. Binary semaphores will be implemented to handle access protection for multi-threads requesting shared resources.

4. Implementation

4.1 Temperature Sensor Re-Configuration

Since we cannot use SysTick, we had to implement the temperature sensor configuration again by making a few modifications. The following Table1 summarizes the modified configurations used to initialize the temperature sensor:

Configuration	Key Parameters	Explanation
adc_init_s	RCC_APB2PeriphClockCmd ADC_Channel_16	Used to enable ADC peripheral and power for bus APB2 that is connected to ADC1 peripheral. Channel_16 used for temperature sensor initialization for ADC
TIM_TimeBaseStructure	TIM_Period TIM_Prescaler	Defined time base configuration TIM4 to generate hardware based time delay. Set TIM_Period as 99, TIM_Prescaler as 8049Hz, enabled ITConfig and counter for TIM4

NVIC_InitStructure	NVIC_IRQChannel NVIC_IRQChannelPreemptionPriority	To add IRQ to the NVIC as global external interrupt to MCU, NVIC_IRQChannel set as TIM4_IRQn with preemptionPriority as 0x02 (third highest priority)

Table1: Temperature sensor initialization

Table2 summarizes a list of methods that were re-implemented (most of them modified) to calculate temperature readings:

Method	**Key Parameters**	**Explanation**
temperature_sensor_set_semaphore	data_semaphore	Method used to set data_semaphore (semaphore ID) as semaphore (implementing a binary semaphore)
temperature_sensor_read_temperature_raw	result raw_reading	ADC's digital signal value stored in result, which is converted to temperature format and stored in raw_reading
temperature_sensor_read	temperature	Final temperature value stored in temperature variable after filtering data using moving average filter
temperature_sensor_alarm	temperature SEVEN_SEGMENT_DISPLAY_MODE_BLINK SEVEN_SEGMENT_DISPLAY_MODE_NORMAL	Method will check if temperature is over threshold. If yes, then blink mode display on and if no, then normal display mode on.
TIM4_IRQHandler	TIM4 semaphore	External interrupt handler, which will verify if TIM4 is reset and then reset semaphore to NULL by releasing once the operation/task is completed by releasing semaphore

Table2: Temperature sensor methods

These implementations can be seen in temperature_sensor_interface.c and temperature_sensor_sm.c.

4.2 Accelerometer Sensor's Modifications for LSM9DS1 chipset

Since we used the LSM9DS1 sensor package, which is a collection of all sensors including temperature, accelerometer, gyroscope and magnetometer, we had to recalibrate given this additional chipset.

Table3 summarizes the initialization of LSM9DS1 with some key configurations:

Configuration	Key Parameters	Explanation
GPIO_InitStructure	SPI -> SCK, MOSI, MISO, CS pin configurations INT1, 2 (interrupts)	SCK, MOSI, MISO, CS connected to pins 5 (GPIOA), 7 (GPIOA), 6 (GPIOA), 9 (GPIOB) INT1,2 connected to GPIOE pins 0 and 1 to detect interrupts
SPI_InitStructure	RCC_APB2PeriphClockCmd RCC_AHB1PeriphClockCmd SPI_I2S_DeInit	APB2 used to initialize SPI peripheral, AHB1 used to enable SCK (clock), MOSI (input), and MISO (output) GPIO clocks, CS (chip select), INT1 and INT2 (interrupts). SPI1 enabled.

Table3: LSM9DS1 initialization

To enable interrupts, read and write data and other configurations of LSM9DS1, we defined certain methods which have been briefly described in Table4 that can be found in lsm9ds.c:

Method	Explanation
LSM9DS1_filterConfig	Used to set internal high pass filter MEMS configuration in terms of high pass filter cut-off level, interrupt and data selection bits
LSM9DS1_InterruptConfig	Used to configure the latch interrupt request for the chipset
LSM9DS1_lowPowerCmd	Used to change the lowpower mode for LSM9DS1 to new state of either power down mode or active mode
LSM9DS1_DataRateCmd	Used to set the transfer of output data at rates of either 100Hz or 400 Hz
LSM9DS1_RebootCmd	Used to reboot memory content of LSM9DS1
LSM9DS1_Write	Used to write bytes to LSM9DS1 which will set the chip select high at the end of data transmission

LSM9DS1_Read	Used to read a block of data from LSM9DS1 which will set the chip select high at the end of transmission
LSM9DS1_ReadACC	Used to read LSM9DS1 output register to calculate the acceleration
LSM9DS1_SendByte	Used to send a byte through the SPI interface and return the byte received from the SPI bus

Table4: LSM9DS1 configuration methods

We also defined EXTI0_Handler for the accelerometer sensor on the microprocessor which will check the status of the external interrupt generated if it's not RESET to execute the task and then eventually set the semaphore to NULL by releasing semaphore (implemented in accelerometer_interface.c).

4.3 Multi-Thread Design

We split the tasks for the processor into 4 threads with osPriority as normal for all:

1. LEDs (thread1 with stack size 100)
2. Temperature measurement (thread2 with default stack size)
3. Keypad (thread3 with stack size 100)
4. Accelerometer measurement (thread4 with default stack size)

The following Figure1 summarizes our overall implementation including threads, semaphores, interrupts, display modes and contents:

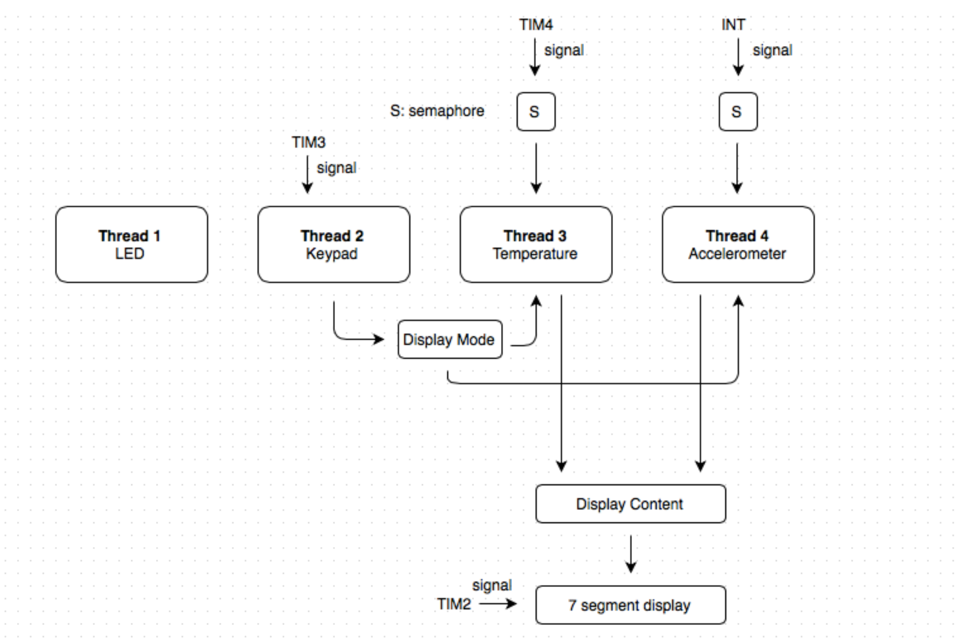


Figure1: Multi-thread implementation diagram for all peripherals

The following Table5 summarizes the implementation of all the methods in lab4_threads.c:

Method	Key Parameters	Explanation
Lab4_os_init	lock_display_module lock_accelerator_display_mode lock_temperature_data lock_accelerometer_data	Method will create and initialize all the semaphore objects and set semaphores for temperature and accelerometer sensors
thread_leds	led_rotation_rotate_leds() osDelay(20)	Will simply rotate leds (independent of any other threads) and will go in sleep mode for 20 ms
thread_temperature	lock_temperature_data	Will wait till lock_temperature_data semaphore is ready, then read temperature and set the display mode to temperature and call the sensor alarm function to detect overheating
thread_keypad	KEYPAD_DISPLAY_MODULE_SWITCH = 4 KEYPAD_ACCELEROMETER_ROLL = 1 KEYPAD_ACCELEROMETER_PITCH = 2	Will check what number is pressed on the keypad (if 1, then display roll, if 2, then display pitch, if 4 then switch display from either accelerometer or temperature) and will go to sleep for 20 ms
thread_accelerometer	display_module lock_accelerometer_data	Will first wait for lock_accelerometer_data semaphore to be ready, and then read the raw angles and then filter the measurements using moving average filter. If display_module is roll, then display roll value and if pitch, then display pitch value

Table5: Threads and semaphores implementation

Finally, all the initializations and executions with respect to sub-systems, threads, semaphores and kernel in the main method in main.c have been summarized in the following Figure2:

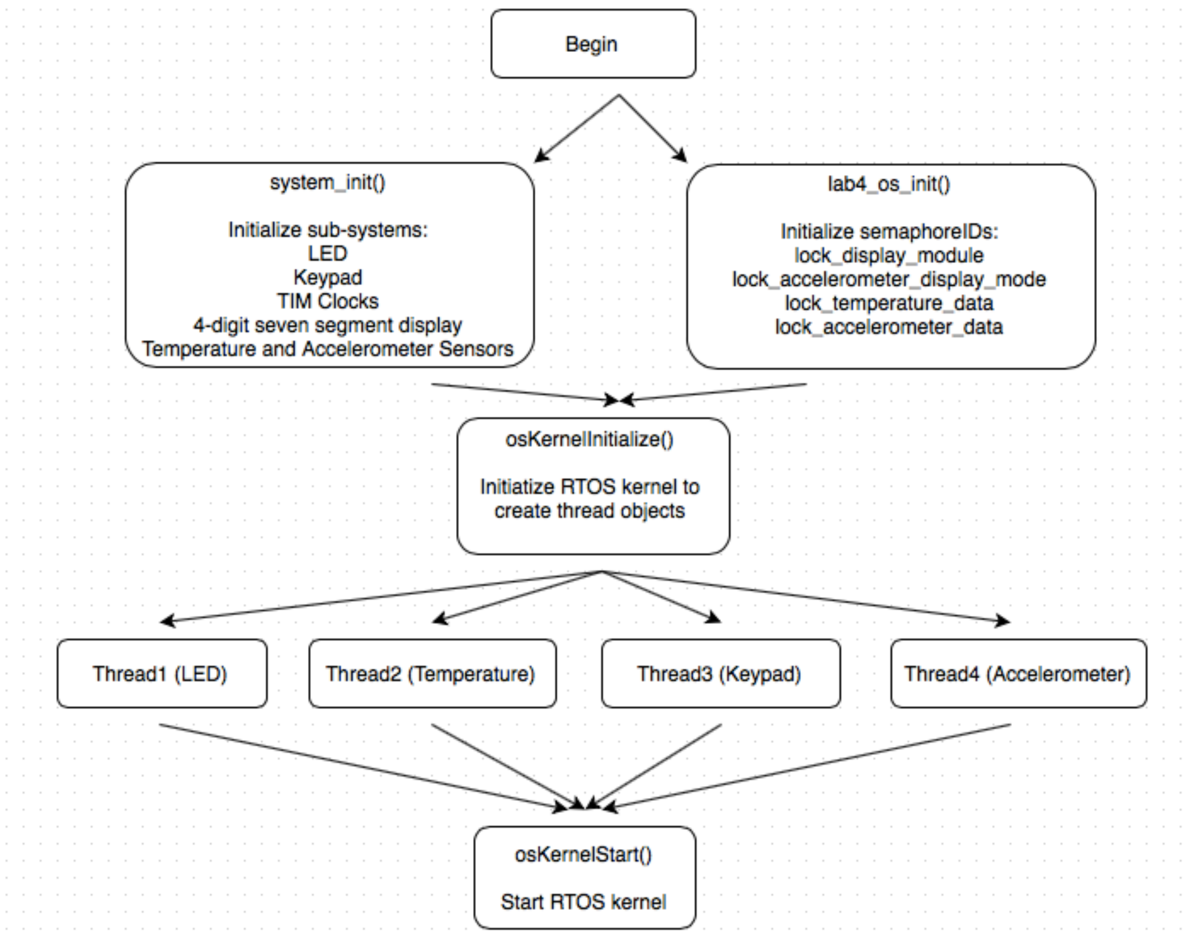


Figure2: Main method in main.c (initialization and execution of tasks)

The final setup can be see in Figure6 in Appendix.

5. Observation

Since we were using LSM9DS1 chipset, we had to recalibrate the setup. Using equations 2 and 3 from Appendix, we obtained the following normalization matrix:

{1.01025618, 0.01006384, 0.02401493}
{-0.00904249, 1.00515987, -0.01296093}
{0.03992654, 0.00910658, 1.0367582}

The constant values for x, y, and z obtained are -28.44255082, -217.47119435, -1548.73592871.

With these values, we observed that the calibration was quite accurate with an uncertainty of 0.1 degrees and worked well while placing the LSM9DS1 chipset flat and perpendicular on the table. We also retained the same depth for the moving average filter (14) from the previous experiments which can be seen in Figure5 (also see equation4) in Appendix.

While executing the test, we ran the Event Viewer to see the thread usage to observe and calculate the instantaneous frequency of each thread which can be seen in Figure3.

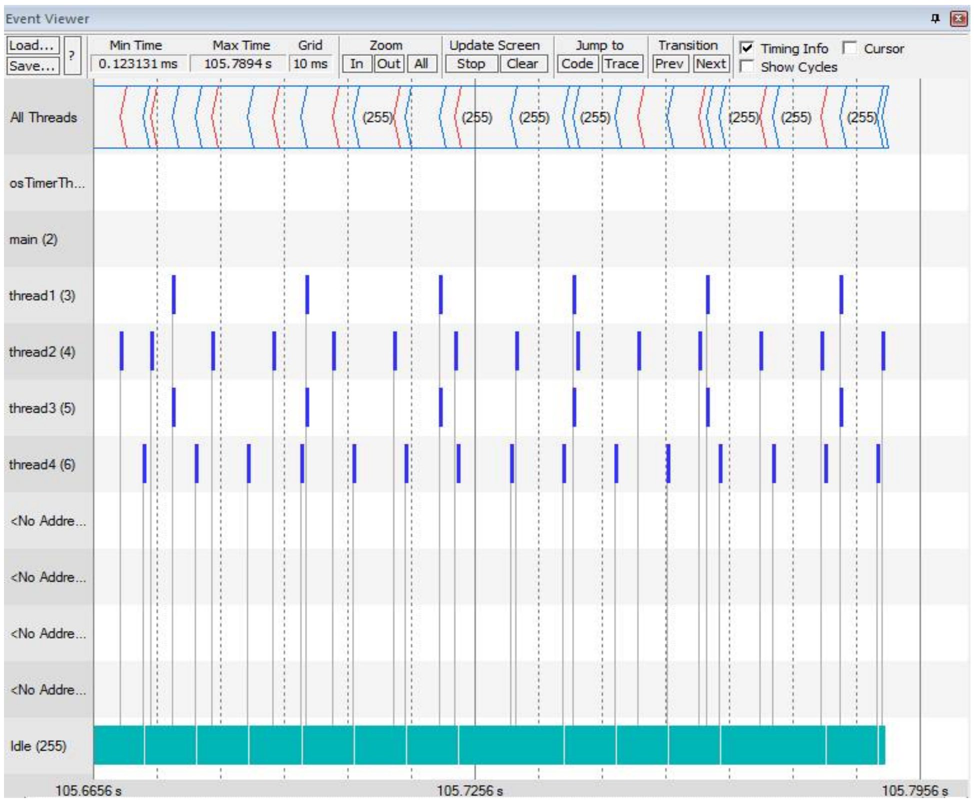


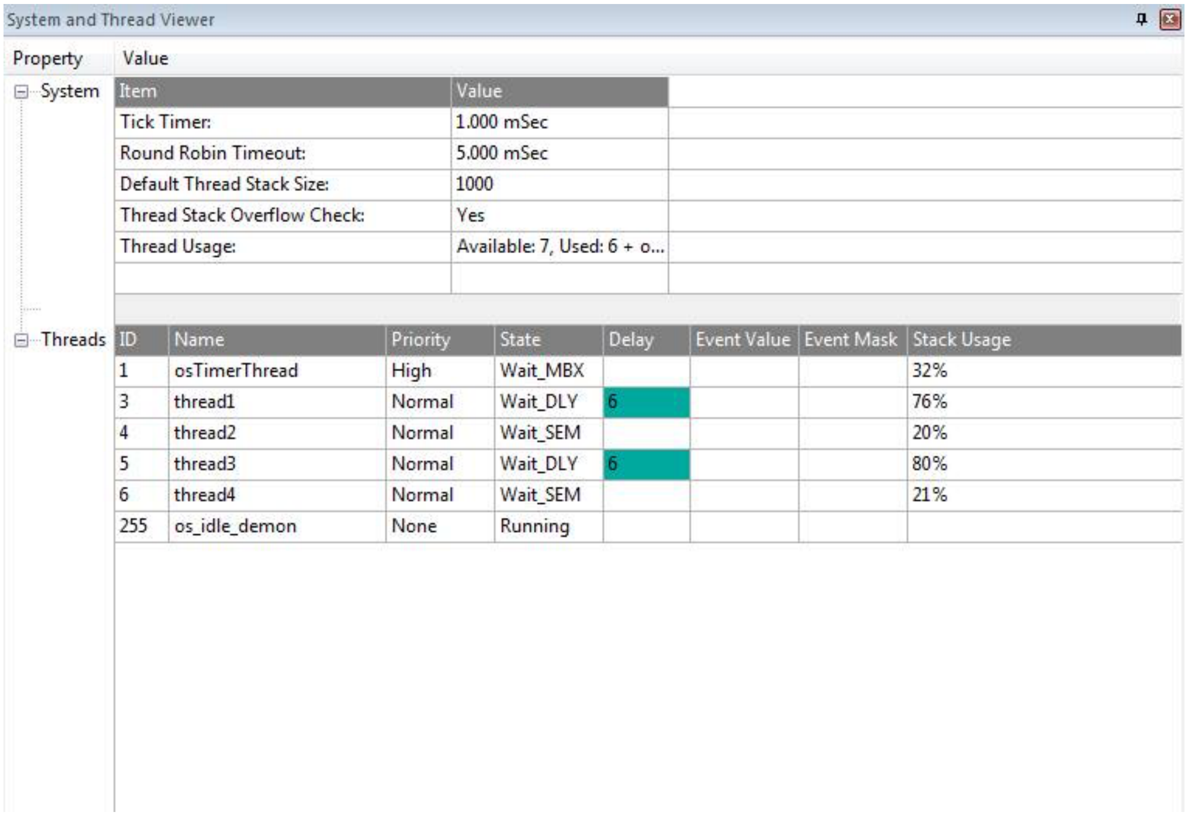
Figure3: Runtime execution of all threads

From this figure, we notice that threads 2 and 4, which are for temperature and accelerometer sensors, are running concurrently at different times because of semaphores. The instantaneous frequency operation of tasks for all threads can be seen in the following table6 (using equation5 in Appendix):

Thread	Task	Start Second	Stop Second	Instantaneous Frequency
1	LED	142.2839s	142.3049s	47.61Hz
2	Temperature	142.2921s	142.3019s	102.04Hz
3	Keypad	142.3051s	142.3261s	47.61Hz
4	Accelerometer	142.3061s	142.3158	103.09Hz

Table6: Runtime instantaneous frequency for all threads

Figure4 shows the efficient stack usage of all the threads:



System and Thread Viewer

Property: System

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	1000
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 6 + o...

Property: Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				32%
3	thread1	Normal	Wait_DLY	6			76%
4	thread2	Normal	Wait_SEM				20%
5	thread3	Normal	Wait_DLY	6			80%
6	thread4	Normal	Wait_SEM				21%
255	os_idle_demon	None	Running				

Figure4: stack usage of all threads

Since we defined the stack size for thread1 and thread3 as 100, their stack usage is visibly more (76% and 80%) than for threads 3 and 4 with default stack size 1000 (20% and 21%).

6. Conclusion

Overall, the peripherals of keypad, clock display, temperature and accelerometer sensors were in synchronization with respect to the threads created while using LSM9DS1 chipset. As seen in figure3, the temperature and accelerometer sensor threads are operating at different times due to the implementation of semaphores and that the instantaneous frequencies are almost the same to avoid potential conflict of access of shared resources over a long period of time.

7. Appendix

References

1. What is multithreading? - Definition from WhatIs.com. (n.d.). Retrieved November 9, 2015, from http://whatis.techtarget.com/definition/multithreading
2. Multithreading. (n.d.). Retrieved November 9, 2015, from http://ocw.mit.edu/courses/civil-and-environmental-engineering/1-124j-foundations-of-software-engineering-fall-2000/lecture-notes/multithreading/
3. Mutex vs. Semaphore, what is the difference? (n.d.). Retrieved November 9, 2015, from http://koti.mbnet.fi/niclasw/MutexSemaphore.html
4. What is An RTOS? (n.d.). Retrieved November 9, 2015, from http://www.freertos.org/about-RTOS.html
5. Majerle, T. (2014, May 10). STM32F4 PWM tutorial with TIMERs - STM32F4 Discovery. Retrieved November 3, 2015, from http://stm32f4-discovery.com/2014/05/stm32f4-stm32f429-discovery-pwm-tutorial/
6. Application Note - LIS302DL. (n.d.). Retrieved November 2, 2015, pp. 13,14, from http://www.st.com/web/en/resource/technical/document/application_note/CD00098549.pdf
7. Application Note. (n.d.). Retrieved October 29, 2015, pp. 14,15,16, from http://www.st.com/web/en/resource/technical/document/application_note/CD00268887.pdf
8. Smith, S. (1997). Moving Average Filters. In The scientist and engineer's guide to digitalsignal processing (pp. 277-280). San Diego, Calif.: California Technical Pub.
9. Liu, J. (n.d.). The Definitive Guide to ARM Cortex-M3 Second Edition. Retrieved November 10, 2015, from http://www.eecs.umich.edu/courses/eecs373/labsW14/refs/M3 Guide.pdf

Figures

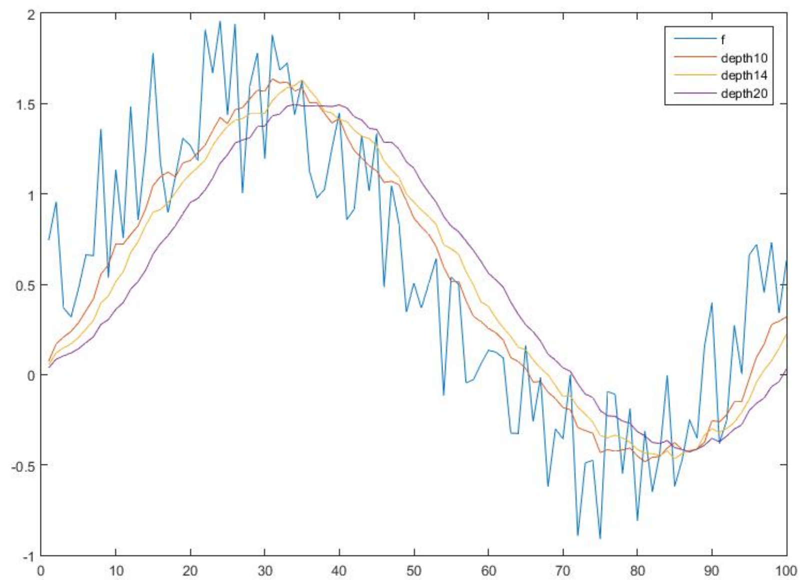


Figure5: Plots for different depths for moving average filter

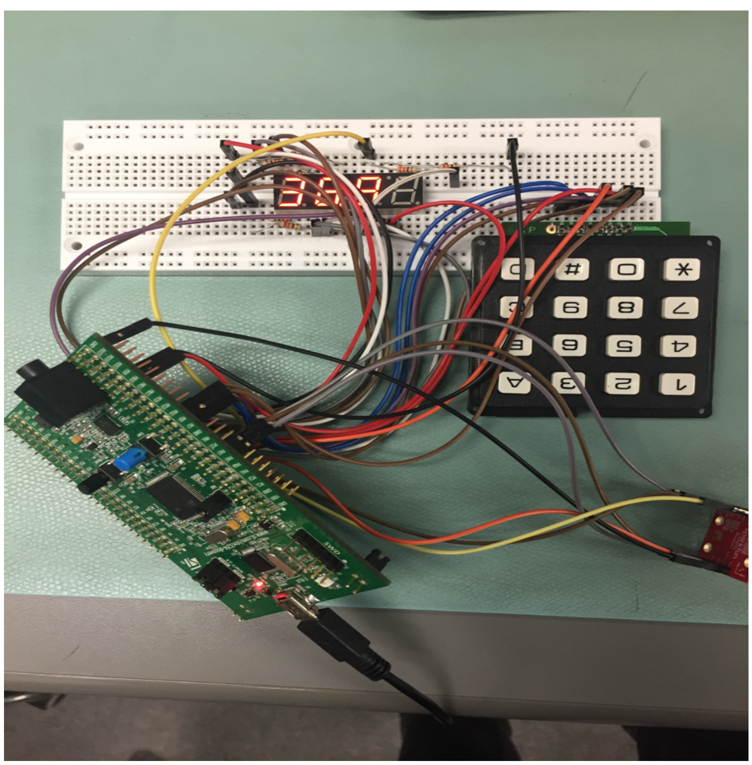


Figure6: Experiment 4 setup with LSM9DS1 chipset

Equations

$$X = [w^T . w]^{-1} . w^T . Y$$

Equation 2: Least square approximation to calculate X (Reference 7 in Appendix)

$$Y = w.X$$

Equation 3: Generic least square approximation (Reference 7 in Appendix)

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j]$$

Equation 4: Moving average filter equation (Reference 8 in Appendix)

$$frequency\ of\ thread = \frac{1}{stop\ second - start\ second}$$

Equation 5: Runtime frequency for each thread