Final Project

3D Printing Machine

Maxim Goukhshtein, 260429739

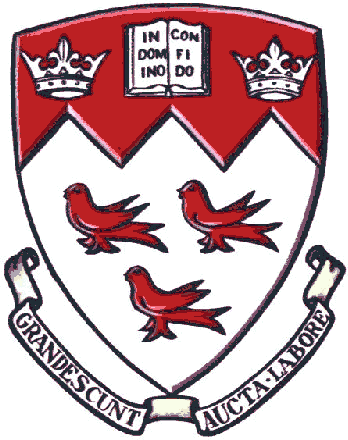
Olivier Laforest, 260469066

Nuri Ege Kozan, 260359680

Geneviève Nantel, 260481768

Department of Electrical and Computer Engineering

McGill University, Montreal



April 21st, 2015

Group 4

ECSE 426 Microprocessor Systems

Presented to Professor

Zeljko Zilic

# Abstract

The goal of the final project is to create a 3D printing machine which is controlled wirelessly by a command board. Using the keypad, a user selects the desired shape of his/her choice and the graphical LCD board will display the chosen shape on the screen. Subsequently, the plotting/printing commands are sent wirelessly to the controller board which controls the motors in order to draw the desired shapes on the whiteboard. The x-y-z coordinates representing the shapes are sent to the 3D printing controller board and go through various calculations in order to determine the motor angle rotation. The whiteboard marker is held by two plastic arms that were 3D printed and is controlled by two stepper-motors to operate the arms in the specific x-y directions. Another stepper-motor is used to lift the structure containing the two motors up and down. Three simple pre-defined shapes such as a square, a rectangle and a triangle can be printed on the whiteboard as well as “on-the-fly” shapes. The “on-the-fly” settings are preconfigured in the keypad and map to certain directions the user wants to draw. Finally, in order to make this project successful, the various tasks are implemented across multiple threads using the RTX RTOS, helping to initialize, synchronize and maintain to proper functioning of the different components.

# Problem Statement

In order to make this project successful, many different challenges needed to be tackled. This section outlines in detail the specifics of the problem at hand.

## Mechanical Construction of the 3D Printing Device

We were provided with 6 plastic 3D-printed pieces. These pieces needed to be assembled together in order to construct a stable system. We were given the following items:

* One base for the motor doing the lifting.
* One base for the two motors controlling the angles for the marker.
* Two arms attached to the motors and pivot.
* Two arms attached from the pivot and holding the marker.

## Timer and PWM Configuration for the Motors

In order to use the PWM module, specific pins were chosen in order to use the hardware PWM module.

## Calculations for Motor Control

Depending on the desired shape chosen by the user, the x, y, z values are sent to the receiver board and then converted to angles. These angles are used to control the rotation of the motors. However, calculations must be done in order to get those angle values.

# Theory

In order to move the motors to the appropriate x and y values sent by the transmitter board, calculations need to be done in order for the motors to move to the appropriate angles. The calculations below convert x and y values to angle values that will be sent to change the motor pulse. Our system can be represented by a pentagon as shown on the figure below:

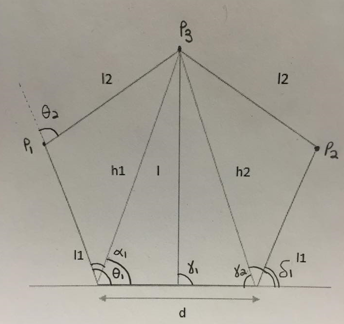
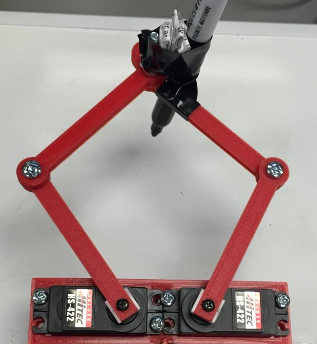
 

Figure : Pentagon Theoretical Representation Figure : Pentagon Hardware Representation

The following cosine and sine mathematical laws were used in order to do the calculations for properly converting the x-y values to motor angles. The first step is to calculate the length which is the length from the marker to the position between the motors:

|  |  |
| --- | --- |
|  |  |

With this, and can be calculated using the cosine law stated below:

|  |  |
| --- | --- |
|  |  |

Similarly,

|  |  |
| --- | --- |
|  |  |

Using the sine law mentioned below, can be calculated:

|  |  |
| --- | --- |
|  |  |
|  |  |

Similarly, can be calculated using the sine law:

|  |  |
| --- | --- |
|  |  |
|  |  |

Using the cosine law again, and can be calculated:

|  |  |
| --- | --- |
|  |  |
|  |  |

And finally,

|  |  |
| --- | --- |
|  |  |
|  |  |

and are the values that are being used in order to control the duty-cycle for the PWM of the motors.

# Implementation

## Motor Configuration

The motors are controlled using a hardware pulse-width modulation (PWM) feature. To do so, a hardware timer that supports PWM must be chosen as well as the channels on the timer. TIM3 was used as it supports PWM. The table below shows all possible pins for the TIM3 timer and each channel. Each output channel can only support a maximum of 3 pins.

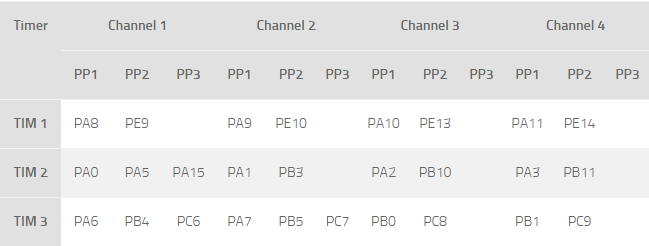


Figure : TIM3 PWM Channels and Pins

The prescaler and period were then set. The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit/32-bit register (in the TIMx\_PSC register). Basically, it takes a basic timer clock frequency and divides it by some value. It allows the timer to be clocked according to user’s desire. The desired rate is defined as follows:

|  |  |
| --- | --- |
|  |  |

Furthermore, the Output Compare (OC) mode and PWM were then configured. OC is used to measure duty cycle, period, frequency and PWM is used to produce a square wave with a particular duty cycle, period, and frequency. In other words, OC is input and PWM is output. The OC function is used to control an output waveform or indicating when a period of time has elapsed. When a match is found between the capture/compare register and the counter, the output compare functions. The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or ‘111 (PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register. PWM2 clear on compare was the one being used. Finally, each motor was associated to one pin and each pin was associated to an OC channel where the pulse was adjusted in accordance to the incoming angle.

## Keypad Controller



Figure : Keypad used in the experiment

The keypad was used to control the behavior of the overall system. It acted as the user interface between the user and the transmitter board. The instructions received from the keypad in the transmitter board are then translated into simple drawing commands, which would be wirelessly sent to the receiver board. The initialization and the structure of the keypad was kept the same as in the previous labs, with the addition of extra de-bounce features that would only read the pressed button once, even if the button was kept down. The specific commands used for the instructions were:

* Buttons A, B and C were used to switch between different operation modes. Mode A was used for the predefined shapes, Mode B was used for the on-the-fly and MEMS mode, and finally mode C was used to draw numbers.
* Pound button (#) was used to reset the position of the drawing marker to the initial position at (0, 0).
* Button D was used as the "enter" button, which is required to be pressed every time after the user selects their desired instruction to take effect. When this button was pressed, the desired command would be wirelessly transmitted to the receiver as well.
* All of the other remaining buttons had different commands for each mode;
  + In the predefined shape mode, buttons 1, 2 and 3 were used to scroll through the square, rectangle and triangle respectively.
  + In on-the-fly mode, all of the number buttons except for 5 were used to navigate the cursor vertically, horizontally or at an angle. The button 5 was used to activate or deactivate MEMS mode.

In numbers mode, each number on the keypad represented that number to be drawn.

## LCD Display

The LCD display is used as a feedback device to the user to indicate what state the system is in and to keep track of what inputs were given to the system. Simple methods given in the LCD library of the STM32F429 were used to draw on the display. Using a coordinate system, lines and shapes can be drawn on the LCD by giving the proper coordinates needed for the chosen line or shape. Zero point on the y-axis on the LCD was defined to be the very top point of the display. We reversed the y-axis on the LCD so that it matches our (0, 0) point on the whiteboard. For example to draw a line on the display, we gave it two locations on the coordinate system, and a straight line would be drawn between those points. Here are a few screenshots of the LCD display:

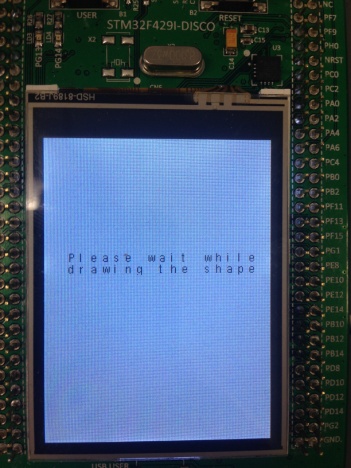
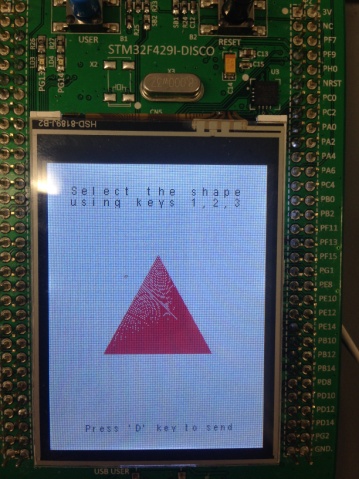
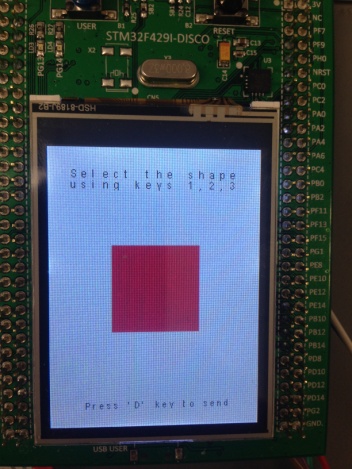
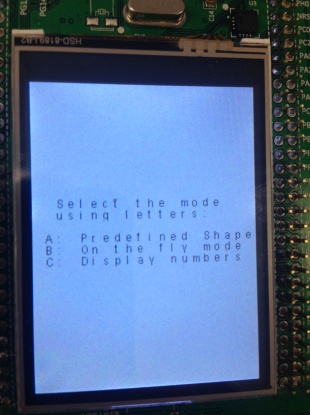


Figure : LCD Screenshots

To implement the blinking effect on drawing the next line, we used a simple technique of re-drawing the same line on top of the previous line with the background color. This gives the visual effect of the line appearing for a little and then disappearing. Although technically a new line is being on top with a different color. Using this blinking effect, the user would be able to see where the next line will be drawn. We can see the results of how accurate the LCD and whiteboard drawings were in figure 6 below:

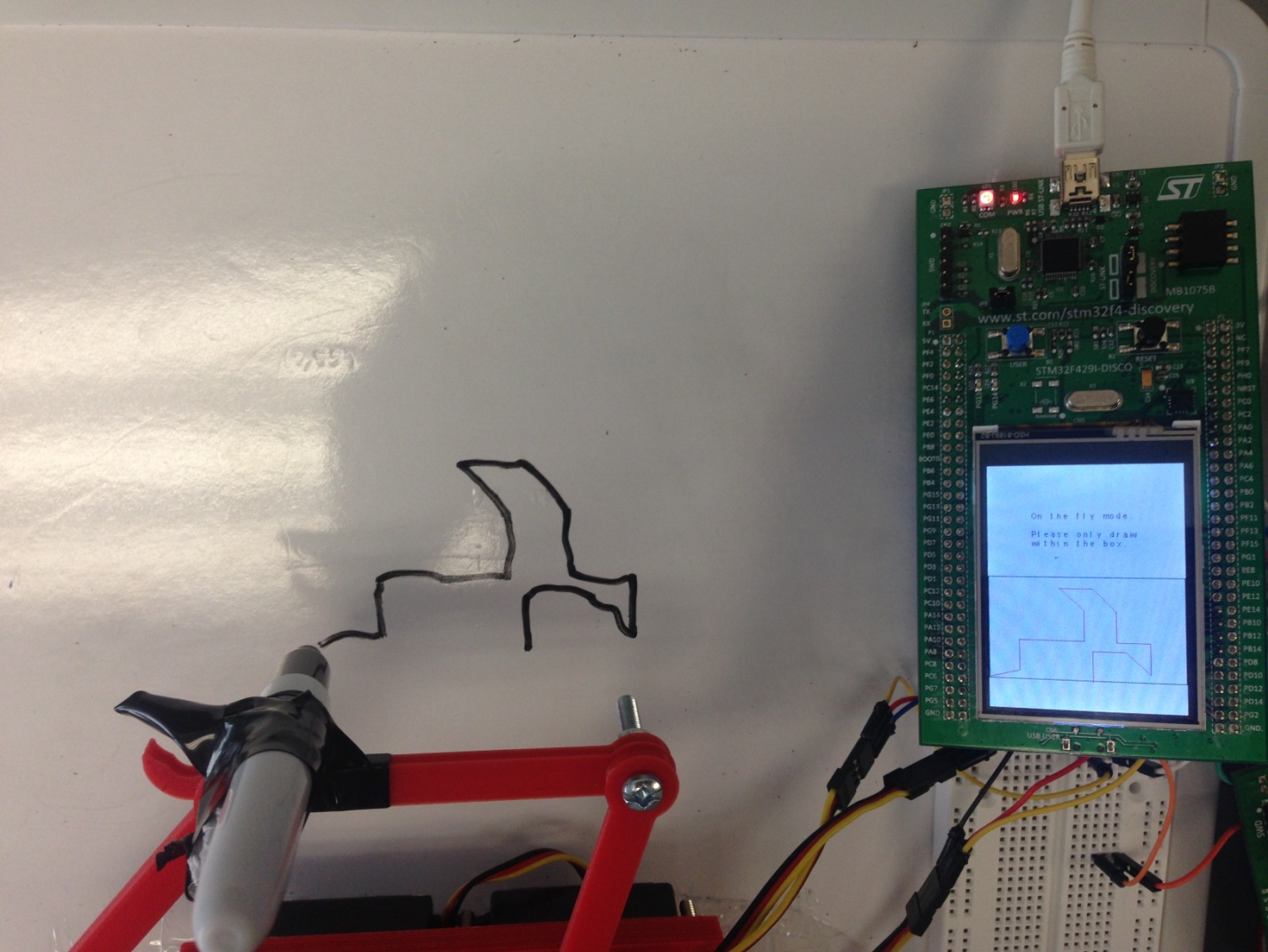


Figure : Whiteboard versus LCD drawing comparison

## Wireless Communication

In order to allow for communication between the transmitter (command) and receiver (3D printing) boards, we were required to use the CC2500 RF transceiver chips (found on the TI MSP430 chipset), in order to establish a means for wireless communication. The wired communication between the STM microcontrollers and the wireless chips was to be done using an SPI interface. To that end, we implemented the CC2500 driver which provides an abstraction for the lower-level SPI drivers. The CC2500 driver draws heavily from the MEMS driver which had been previously used in labs 3 and 4. At the lowest level, the wireless driver provides facilities for writing to and reading from the CC2500 registers. At a higher level, functions for changing the chip’s mode (e.g. receiver, transmitter, idle modes), checking the size of the receiver/transmitter buffers, flushing these buffers and reading/writing data from/to the receiver/transmitter FIFOs were implemented. In order to ensure the proper functioning of the wireless communication, it was essential to assign the configuration registers with the appropriate values. Most of these values were provided to us (with the others being set to the default values). The 2.4 GHz wireless channel was configured to 32, as per the requirements (i.e. group number (4) \* 8). The driver also includes a routine for low-level initialization, which is used to set the various GPIOs and SPI structure and enable hardware clocks. At the highest level, we implemented modules which read and write data to/from the FIFOs once such data is available (e.g. once a user selected a shape to draw, once a packet is received).

## Transmitter OS Threads High Level Design

We had three different threads running at the same time for the transmitter board. These threads were the keypad thread, LCD control thread and finally the data send thread. The priorities were set so that the data send thread had the highest priority, LCD control thread had normal priority and the keypad thread had the lowest priority. The reason for that was because both the LCD and data send threads could only be executed by the keypad thread, whenever there was a user input. When there was an input, the response from the system would be more important than listening to the next instruction. So whenever there was a need to send data, that would happen first, and then any information would be indicated on the LCD, and finally then we could listen to the keypad again. All of the threads were initialized with 600 kB default stack size to guarantee safe thread operations.

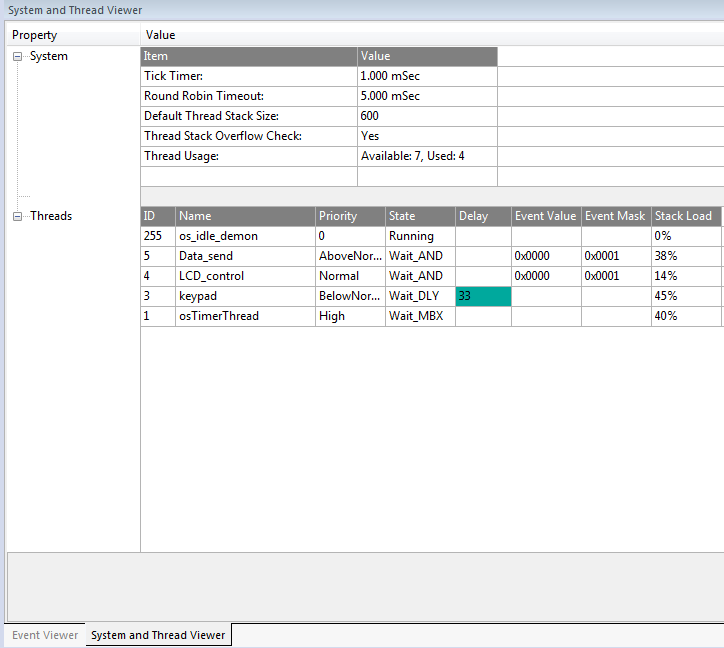


Figure : Thread priorities and stack sizes

### Keypad Thread

The keypad thread acted as the "master" thread that controls the other threads. In fact, the only independent thread in our system was the keypad thread, meaning that it was the only thread to be executed periodically by a timer instead of waiting for a signal. There only needs to be a response from the system if there is an instruction from the user, and this is why the keypad is used to send signals to the LCD and data send threads. The keypad thread would periodically check the buttons to see if there are any inputs, and then if there were, it would modify the global variables affected by the input, and then send a signal to the proper thread.

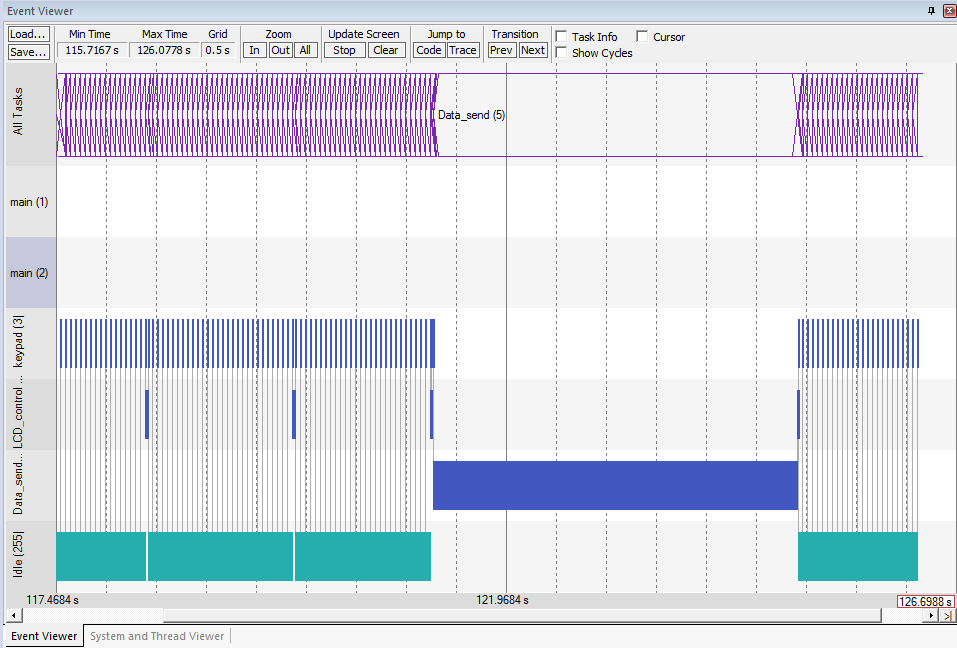


Figure : Mode A (predefined shapes) event viewer

As we can see above in figure 8, the keypad thread executes periodically. Whenever a button is pressed on the keypad, the LCD thread executes for a little bit to print new information on the display. Finally when something needs to be drawn on the whiteboard, data send thread takes over and sends the data wirelessly to the receiver.

### LCD Control Thread

The LCD control thread controls what is being shown on the display of the transmitter board. It waits for a signal from the keypad thread, and when the signal comes, it checks the appropriate global variables to decide what should be shown on the display. First it checks which mode we are in and which mode we were in the previous state. If the mode has changed, it clears the display and prints the current mode's home screen.

In mode A (i.e. the predefined shapes mode), the only time needed to refresh the display was when the user was scrolling through the shapes. In mode B however (i.e.the on-the-fly mode), a blinking effect had to be implemented to indicate which position the next line would be drawn on if the enter button was pressed. To do so, a signal had to be sent to the LCD control thread periodically to show the effect.

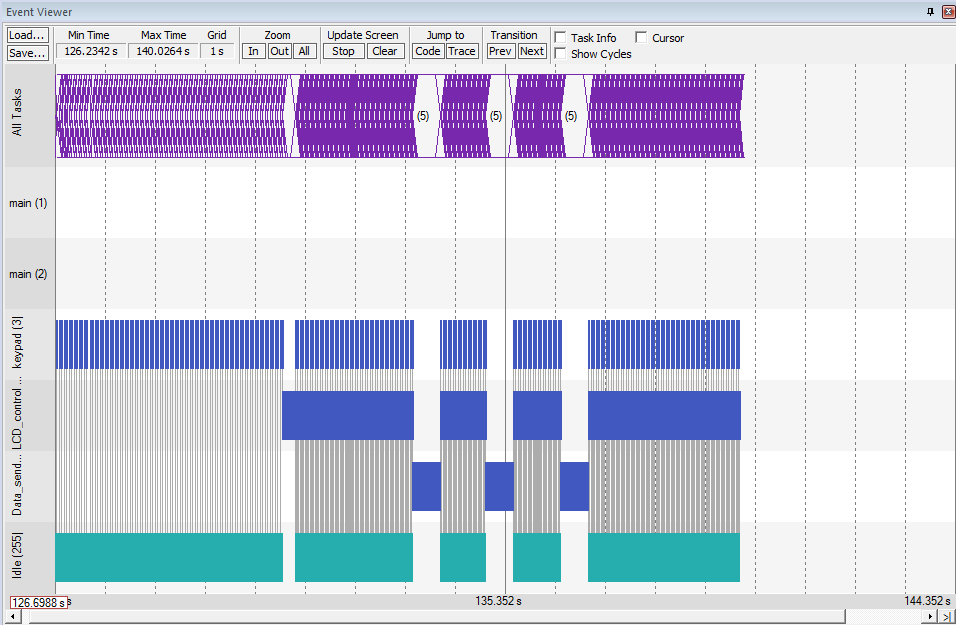


Figure : Mode B (on the fly) event viewer

As we can see in figure 9, when mode B is selected, in order to have the blinking effect the LCD control thread starts executing a lot more often compared to mode A shown in figure 8.

### Data send Thread

The data send thread is responsible for executing only when data needs to be sent. It waits until a signal is received from the keypad "enter" button, and when it is received, it checks the global array which holds the variables needed to be sent and sends it to the receiver.

Although sending the data takes very little time, the data send thread holds control for a lot longer than that. This is because we do not want the user to give more instructions while there is a drawing in progress. For example, as is evident from figure 8, takes a long time to fully draw one shape. After a "please wait" message is displayed on the LCD, the data send thread locks the system to prevent the user from giving any other instructions. Any inputs given during this time will be ignored by the system. Comparing figure 8 and figure 9, we see that on-the-fly mode holds the lock for a lesser amount of time because of the fact that drawing a single line takes less time to complete than drawing a full shape.

## Receiver OS Threads High Level Design

On the receiver side, 3 threads were implemented, namely the data receiving thread, the servomotor thread and the MEMS thread. Communication (and synchronization) of the threads was done using message queues which share their source of memory from a single memory pool.

### Data receiving Thread

This thread is responsible for periodically checking whether a data packed was received. The system was designed such that a data packet always contains 4 bytes, with the first byte being a preamble (also used to signal that the MEMS mode was to be used), followed by 3 bytes, one per coordinate, the x-coordinate, the y-coordinate and the z-coordinate (in this order). Once it is determined that a packet was received (i.e. the RX FIFO contains at least 4 bytes), the bytes are read and the required action is taken. If the preamble byte is found to be equal to 254, the mems mode flag is set, allowing for the MEMS thread to know whether it should be functional. Otherwise, if the preamble is equal to 255, the data which was read from the FIFO is then sent to the servomotor thread via a message queue. To implement the periodic polling, a TIM3 clock, running at 20 Hz, was used. When the clock expires, it signals this thread, allowing it to become active and proceed with its actions.

### MEMS Thread

This thread is used in order to implement the MEMS mode, which can be chosen by the user using the procedure outlined above. The functioning of this thread is similar to the previous labs. An external interrupt on a line connected to the MEMS was setup. Once an interrupt occurs and if the MEMS mode flag is set, the normalized accelerations are read and used to calculate the board’s pitch and roll angles, as was done in previous labs. The calculated values were then filtered using 2 Kalman filters (one for each angle), using the procedure used and discussed in all 4 of the previous labs. The normalization of the accelerations was done using an (offline) calibration procedure which was outlined in . Once the filtered values are retrieved, they are converted to x,y positions as follows:

|  |  |
| --- | --- |
|  |  |
|  |  |

The idea was to map the possible angle’s range of 0 ̊ to 180 ̊, to the range of possible x,y positions reachable by the drawing arms. Note also that in the case of the y-coordinate (i.e. roll angle), angles below 45 ̊ were set to 45 ̊ and angles above 135 ̊ were set to 135 ̊, in order to make the user experience more pleasant (i.e. the user doesn’t need to tilt the board too much, which can be difficult with all the wires, in order to reach the full range of y coordinates).

Finally, once the positions are determined, they are packed into a message queue and sent to the servomotor thread.

### Servomotor Thread

This thread is responsible for issuing the commands needed for moving the drawing arms to the correct position. First this thread initialized the motors and timer/PWM structures required for the proper functioning of the motor. In the main body of the thread, it waits for a signal to be set. This signal is set be either the data receiving thread or the MEMS thread. Once the signal is set, it checks on which message queue data was received (i.e. this queue should match the thread which most recently set the signal). The message, containing the new position to which the arms must be moved, is then read and converted to angles (using the method described in the theory section), which are then used to set the PWM duty cycle for the motors.

# Testing and Observations

As the system is composed of numerous components, testing of each component was done separately. Testing for the motors, MEMS, keypad and LCD was done in a straight-forward way. Once each of these modules was completed, they were tested individually to ensure that their functionality matches our expectations. For example, the keypad was tested to ensure that the keys are properly mapped and that the debouncing is working properly. The LCD was tested to ensure that the messages and drawings shown on the screen are in line with the user’s input. Similarly, the motors were tested by programming them to move to various predefined positions and making sure that they reach these positions with an acceptable degree of accuracy.

The testing of the wireless communication component was done in a number of steps. First, once the basic SPI read/write functionality was implemented, we wrote various values into the configuration registers and then read them back, to ensure that these 2 core functions work properly. Once this was validated, we tested to make sure that the receiver board was able to read the beacons transmitted by the board which was setup in the lab by the TAs. Finally, we then wrote the transmitter module, modified the required wireless channel to 32 (from the default 0) and made sure that our test messages, sent by the transmitter board, were properly received and read by the receiving board.

Once all the individual components were shown to work properly, we integrated all of them into a single system. We then followed by performing extensive testing of the entire system, making sure that all the modes and their functionalities work according to the specifications.

## Breakdown of Work Between Team Members

|  |  |
| --- | --- |
| Name | Work Description |
| Olivier Laforest | * Wireless * Motor calculations |
| Genevieve Nantel | * Hardware * Motor configuration and calculations |
| Nuri Ege Kozan | * LCD, keypad and transmitter board logic * Combining all components |
| Maxim Goukhshtein | * Wireless, MEMS and receiver board logic * Combining all components |

# 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]. |