Colin Young

The future of engineering is here and it's bendable. The flex sensor, the latest and greatest new addition to our engineering laboratories, allows for variable resistance based on the degree of bend in the sensor. The implementation of this technology has countless uses and will no doubt change the way we live our day to day lives but before we get carried away we first need to figure out how to make this analog input translate into digital output. The goal of this project is to work with ADC input in the form of a 25k flat flex sensor and filter the data through a 16 point moving average filter. In order to keep our code organized, we will keep the processing of the ADC input in a separate source file called EE340.

Requirements:

1. Set up ADC and UART Parameters
2. Create circuit for the flex sensor
3. Receive input from the flex sensor
4. Compute a 16 point moving average of the incoming data
5. Transmit the filtered average to the D8 pin sending the data to tera term
6. All coding tasks must be done outside of the main.c file

Verification:

1. In order to utilize the ADC components on the NUCLEO-L552E-Q board we will be setting pins PC0, PC1, and PC2 to be inputs for ADC1. We also need to make sure the low power UART connection on PG7(TX) and PG8(RX) is activated as we will be transmitting our processed data on those pins. The UART transmission will be using a baud rate of 115200, have an 8 bit word length with no parity and one stop bit.

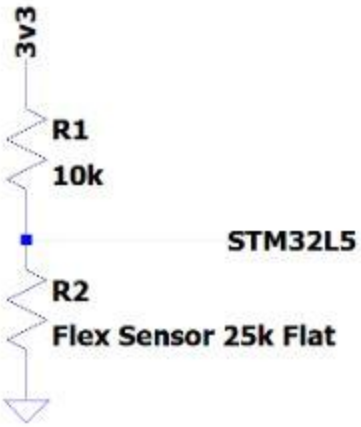
2. In this project we will be using a flex sensor connected to our nucleo board via the C2 pin to read in data. To get the circuit working correctly we must attach the 3v3 pin to a 10k resistor, which is in turn connected in parallel to the stm32l5 _pin, and the flex sensor. The other pin on the sensor is connected to ground. See tech details picture 1 and 2 for the circuit diagram and a picture of the board.

3. In order to receive data from the ADC we must implement a few HAL commands. The first command is HAL_ADC_START(), which enables the ADC, HAL_ADC_PollForConversion(), which sets a flag to be triggered once the conversion is completed, and HAL_ADC_GetValue(), which returns a value from the register.

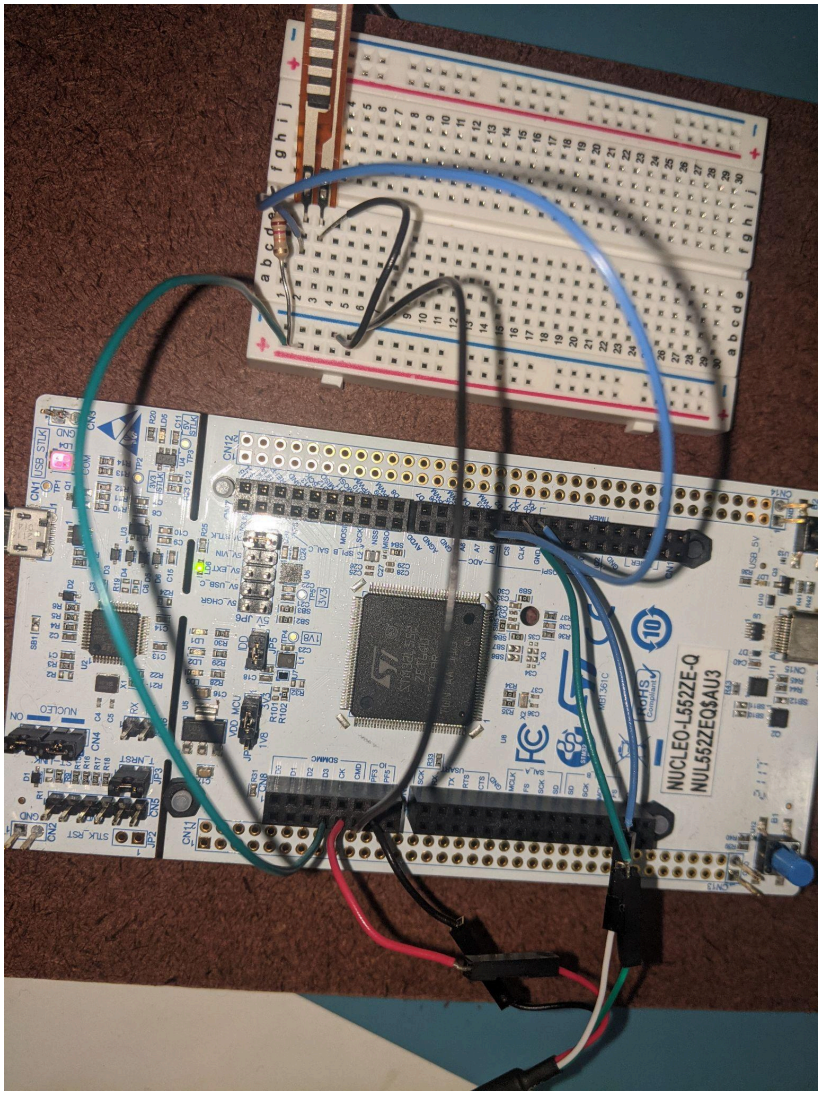
4. In order to implement the moving filter we will be setting up and using a circular queue. To implement the data structure we need to create a few functions: isEmpty, which checks to see if the queue is empty, isFull, which checks to see if the queue is full, enqueue, which adds an element to the back of the queue, and dequeue, which removes the first element from the queue. The size of the filter can be easily changed by adjusting the defined value at the top of the source file but for our lab we will be using a size of 16. As we read in data, we store each value into the queue. To make things easier on ourselves, we implement the queue in such a way that it keeps track of its own total sum, adding or subtracting from the total on each de/enqueue. We keep storing data into the queue until it reaches our desired filter size. Once the queue is full and we read in a new data value, then transmit the sum divided by the filter size to get our average value. We then simply dequeue the first value entered, and enqueue the new value to prep our queue for the next average. For more information reference the class diagram and flow charts in the tech detail section of the report.

Tech Details:

Picture 1 (circuit diagram):



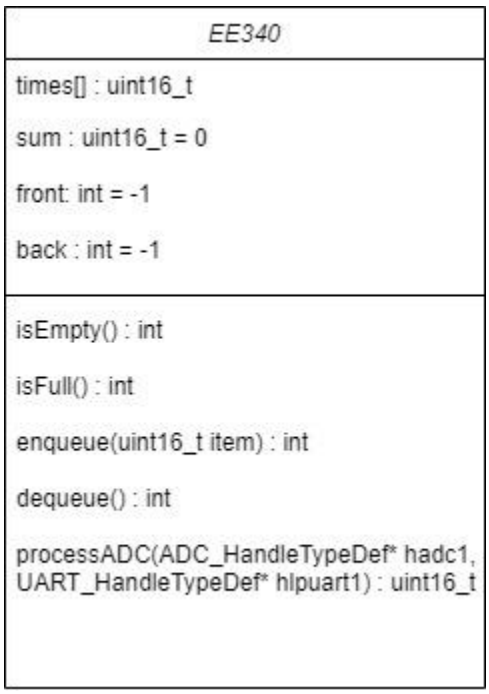
Picture 2 (board layout) :



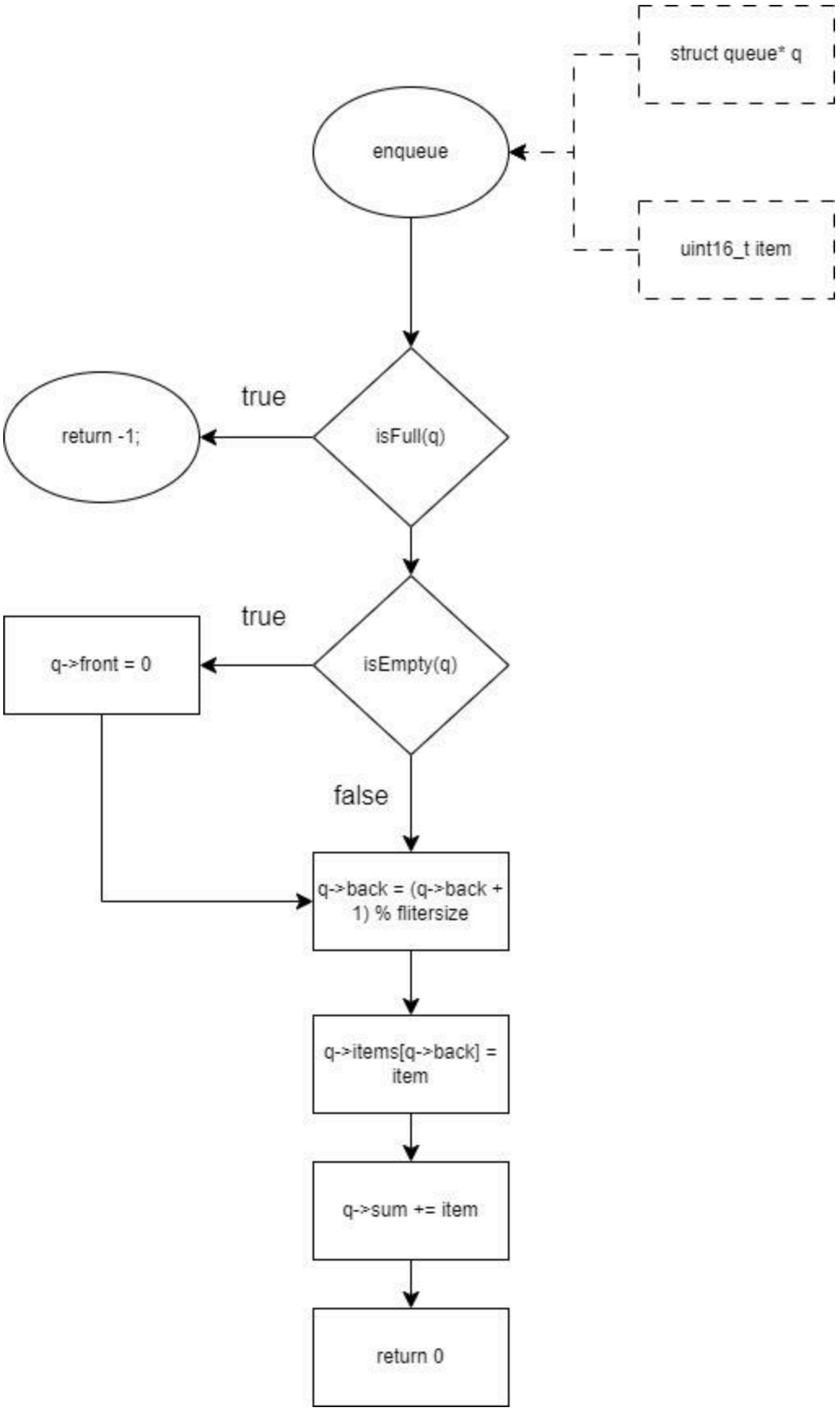
Picture 3 (main.c) :
The image below is the main while loop in main.c. all of the processing for the lab is done in the EE340.c file so all main does is call the function processADC from that file.

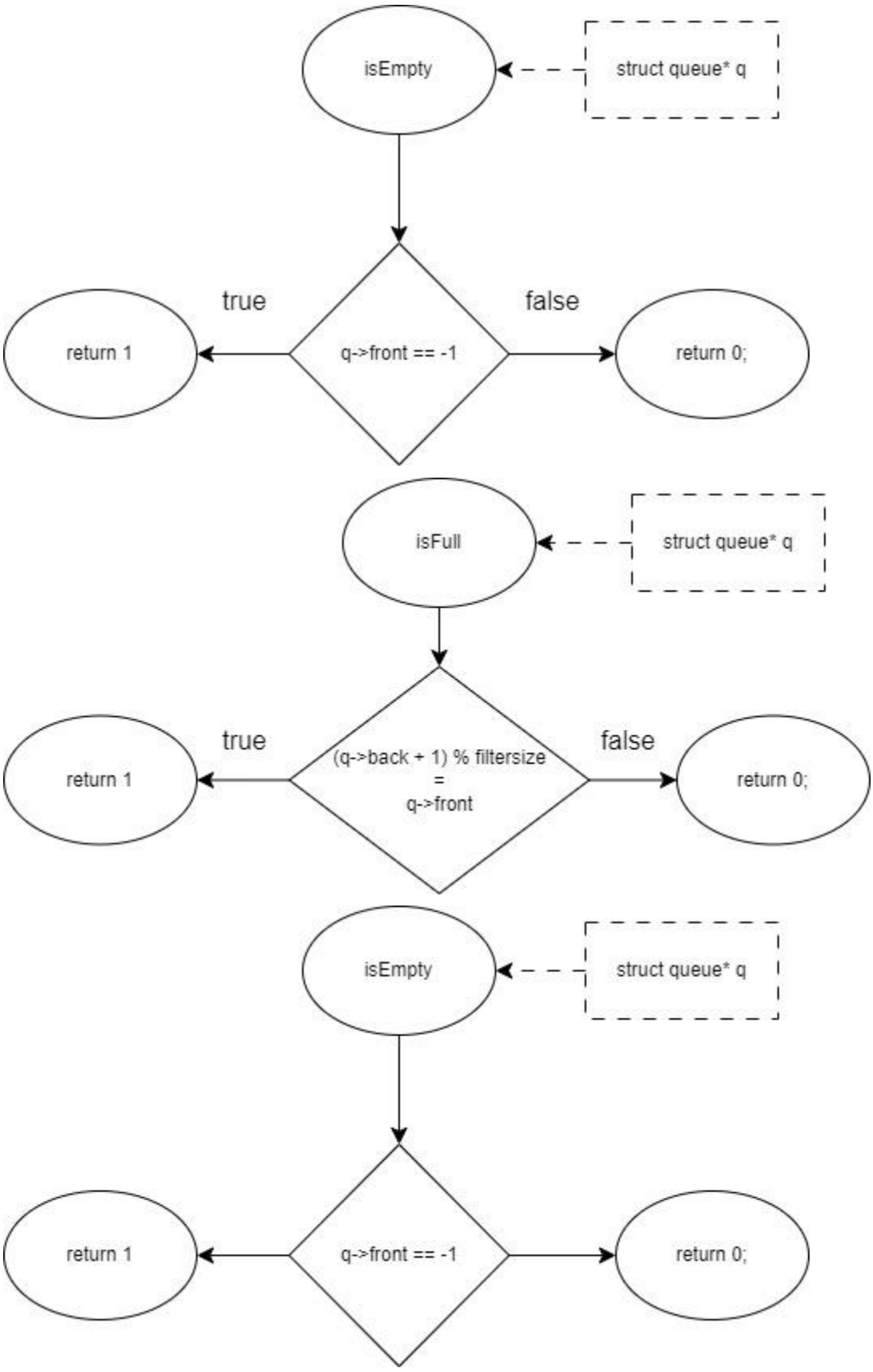
```
117   /* Infinite loop */
118   /* USER CODE BEGIN WHILE */
119   while (1)
120   {
121     /* USER CODE END WHILE */
122
123       processADC(&hadc1, &hlpuart1);
124
125     /* USER CODE BEGIN 3 */
126   }
127   /* USER CODE END 3 */
128 }
```

EE340 class diagram:



The figures below show the flowcharts of the EE340 functions.


struct queue* q
enqueue
uint16_t item
true
return -1;
isFull(q)
true
q->front = 0
isEmpty(q)
false
q->back = (q->back + 1) % flitersize
q->items[q->back] = item
q->sum += item
return 0


isEmpty
struct queue* q
true
false
return 1
q->front == -1
return 0;
isFull
struct queue* q
true
false
return 1
(q->back + 1) % filtersize
=
q->front
return 0;
isEmpty
struct queue* q
return 1
q->front == -1
return 0;

Conclusion:

The main difficulties in this project mostly stemmed from setting up the board connections and making sure things were actually being read into the ADC. Not having access to all of the required materials slowed down the process a bit as we could only fully test in the lab, but we eventually got everything set up and running properly.