Project 2:

Heart Monitor

Noha Mohamed

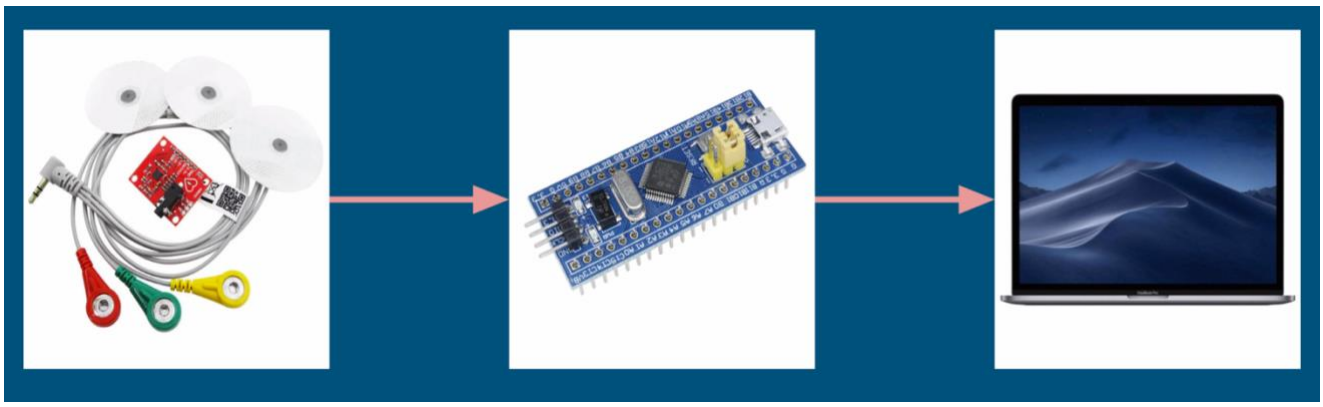
900163073

In this project we were tasked with developing an embedded application that the uses an STM32 board and the ECG sensor to read a heartbeat signal that is measured by the ECG sensor for 1 minute, calculate the BPM, and plot that signal.

I used the STM32F103C8 blue pill and the ECG module AD8232 along with the USB-to-TTL module converter to read the heartbeat signal and display it as a plot on my PC over a USB link.

The ADC output is an analog signal. Therefore, in order to be able to process the signal on my PC I need to convert it into a digital signal, for that I need to use an ADC. I'm using my board's on-chip ADC to do the conversion. First I need to connect the output pin from the ECG sensor to pin A0 on my board which is IN0 for the first ADC on my board. Then I connect my board to the USB-to-TTL module which is connected to my PC. I used UART to transmit the ADC output from the board to the PC. I then read the signal from the com port and I plot the values received through UART.

The Diagram below demonstrates the flow of data:



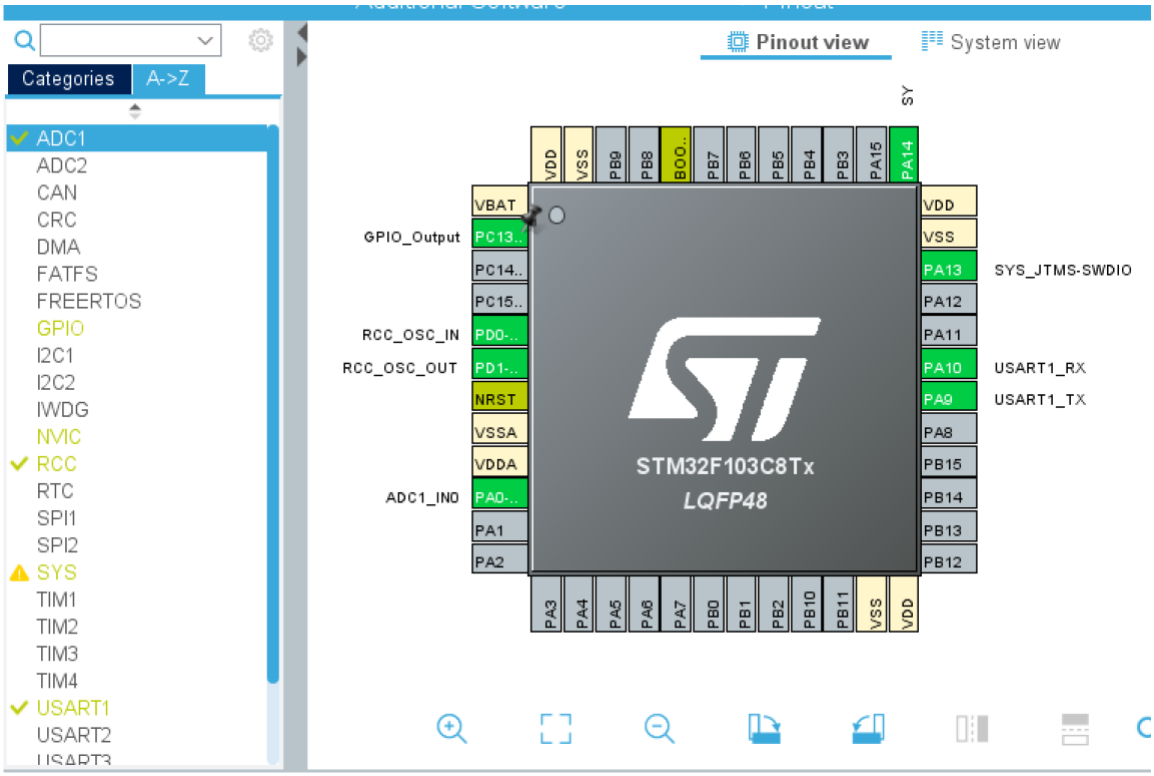
The project was developed on many fronts. First we needed to configure all the necessary elements on the board through CubeMX in order to be able to use the board properly. Then I needed to right the code that I was going to load into my board on Keil uvision, build that code and load onto the STM32F103C8 board. At this point the role of Keil is done and I used python to start my conversion, read the data from the serial port and plot it through commands. In this report I will be discussing my work on all three fronts.

STM32CubeMX:

First I created a project with my board type. I needed to enable the UART to be able to send and receive data from and to my board. Therefore, I enabled USART1 in Asynchronous mode. In the UART's parameter settings I set the baud rate to 115200, and in the NVIC settings I enabled USART1 global interrupt. The USART1_RX was shown to be PA10 and the USART1_TX was shown to be PA9.

Other than the UART, I configures PC13 which is the board's LED as a GPIO_Output in order to be able to use the LED (this is not necessary but was kind of helpful for debugging purposes). Last thing I needed to enable was the ADC. I needed the ADC in order to convert the analog signal coming from the ECG to a digital signal. I chose to send data over IN0 which is the first input channel to the ADC, therefore I enabled it. These were the important configurations that needed to be don through STM32CubeMX in order to be able to use the STM32F103C8 board to read data from the ECG, convert it to a

digital output and transmit over the UART. Below is a photo of the final pinout from STM32CubeMX:



Keil uVision5:

After I finished my STM32CubeMX configurations, I generated the code and opened the project on Keil uVision5. Here is where I wrote the code that I programed into my board.

The only thing I wrote in the main was the following:

```
HAL_UART_Receive_IT(&huart1,(uint8_t *)s, sizeof(s));
```

I called this function in order to receive through the UART data that starts the ADC conversion. This function triggers the UART handler in the stm32f1xx_it.c, in there I read the input which is the sampling rate and the period for which I will read from the ADC, and set them to variables that are

global in the main and external in the stm32f1xx_it.c file. Below is the code from the handler:

```
int x = 0;
if(strcmp(s,"\r") == 0){
  counterTime++;
  if(counterTime == 2){
    sscanf(second, "%d", &x);
    Sample = 1;
    period = x;
    counterTime = 0;
    HAL_ADC_Start(&hadc1);
    SystemCoreClockUpdate();
    SysTick_Config(SystemCoreClock/samplingRate);
    s[0] = '\0';
    memset(second, 0, sizeof(second));
  }else{
    sscanf(second, "%d", &x);
    samplingRate = x;
    memset(second, 0, sizeof(second));
  }
}else{
  strcat(second,s);
}
```

I check to see if I have received a \r indicating that enter was pressed and then I read the written data. Since I'm sending to variables over the UART, I made a counter so that when the first \r comes I can save the sampling rate. And when the second \r comes I can save the period (the input should be in seconds).

When I receive the period, I set it an I configure the systick timer as follows:

Systick_Config(SystemCoreClock/samplingRate)

This means that the Systick handler will be called x times in 1 second where x is the sampling rate. This enables me to read from the ADC based on the provided sampling rate. The configuration of the systick triggers the

HAL_IncTick() function which will be called on intervals based on the sampling rate, I also start the ADC in order to begin reading from it. I made sure that HAL_IncTick() does not get called unless the UART data is sent by calling HAL_SuspendTick() which suspends the systick interrupt. I call this function as long as the Sample flag is 0 and I only set the sample flag to 1 in the UART handler when both the sampling rate and the period are received.

In order to read values from the ADC, I created the following function:

```
void startADC(){
  pulse = HAL_ADC_GetValue(&hadc1);
  BPM(pulse);
  numOfSamples++;
  sprintf(temp, "%d\r\n", pulse);
  HAL_UART_Transmit(&huart1,(uint8_t *)temp, strlen(temp), 10);
}
```

Pulse is of type uint32_t because the ADC is 12 bits so the output can range from 0 to 4096. I transmit the values that I read from the ADC over the UART. I also call BPM which is a function that calculates the bpm (beats per minute). The code for BPM is shown below:

```
void BPM(uint32_t val){
  if(newBeat == 1 && (val > threshold))
  {
    bpm++;
    newBeat = 0;
  }else{
    if (val < threshold){
      newBeat = 1;
    }
  }
}
```

I increment the global variable BPM when I get a values that is above a certain threshold and the newBeat flag is 1. The threshold is 3000 (ADC output is between 0 and 4096), and when I increment the bpm I set the newBeat flag to zero and I only set it to one again if the input signal is below the threshold. In other words, if I get a signal above the threshold and the previous signal was below the threshold, then this counts as a beat and I increment the bpm.

All of the function mentioned above are called periodically inside the HAL_IncTick() function as shown below:

```
void HAL_IncTick()
{
  if(Sample == 1){
    flag = 1;
    tick++;
    if((tick >= period*samplingRate) && (flag == 1)){
      Sample = 0;
      tick = 0;
      flag = 0;
      samplingRate = 0;
      samplingTime = 0;
      period = 0;
      HAL_ADC_Stop(&hadc1);
      HAL_SuspendTick();
      sprintf(done, "BPM = %d\r\n", bpm);
      HAL_UART_Transmit(&huart1,(uint8_t *)done, strlen(done), 10);
      numOfSamples = 0;
      bpm = 0;
    }

    if(flag == 1){
      startADC();
    }
  }
}
```

if Sample flag is high (which means that the sampling rate and the period are both received), I start my logic. I have a variable named tick that I increment on each iteration to indicate how much time has passed since I first triggered the

timer. I need to run the code for a certain amount of time so I check to see if the period*sampling rate is less than tick, this means that the specified amount of time has passed, and in that case I stop the ADC, suspend the systick, transmit the BPM, and reset all my flags so that I could begin another conversion through software without having to manually reset my board. If the specified time has not yet passed, then I call the startADC() function which takes a sample from the ADC. This describes all the code on the Keil uVision5 front.

Python:

After programming my board, I developed a simple UI on the python front. I used PySerial in order to establish a connection with the serial port, and matplotlib in order to plot the data that I read through PySerial.

```
ser = serial.Serial()
counter = 0
port = input("Enter the port number: ")
baudrate = input("Enter the baud rate: ")
ser.port = port
ser.baudrate = baudrate
if ser.is_open is not True:
    ser.open()
```

In the code above, I take as input the port number and the baud rate. I then assign those values to my serial variable. I check to see if the port is not open and if it's not I open it.

After that, in an infinite loop, I take as input the commands I would like to execute. There 2 commands I can input, the start command which is followed by setting the sampling rate and the period and starts live plotting the data that comes from the serial port, and the bpm command which displays the BPM value. If the command I type is start, I set the plot figure and I call matplotlib's animation function so that I could plot the signal live as I receive the data.

```
ani = animation.FuncAnimation(fig, animate, init_func=inti_func, repeat=True, fargs=(ys, samplingRate), interval=100, save_count=200)
```

The Initialization function calls an initialization function which sets the title for the plot and the axes:

```
def inti_func():
    plt.title('Heart Rate Monitor')
    plt.xlabel('Samples')
    plt.ylabel('ECG Output')
```

And the animate function which plots the received data:

```
def animate(i,ys,samplingRate):
    ecgList2, stopFlag2 = readData(samplingRate)
    if(stopFlag2 == True):
        ys.clear()
        ys = [0] * x_len
        ani.event_source.stop()
    for x in ecgList2:
        ys.append(x)
    ys = ys[-x_len:]

    line2.set_ydata(ys)

    return line2,
```

At first, I was plotting the data point by point, but that slowed down the plotting process and created a lag between the transmitting of a signal and the graph being updated with that signal. Therefore, I created a function called readData that takes as input the sampling rate and returns a list of data that was read from the serial port, then I plot that list in animate. So instead of plotting one point at a time, I'm plotting one frame of data at a time, the code for readData is shown below:

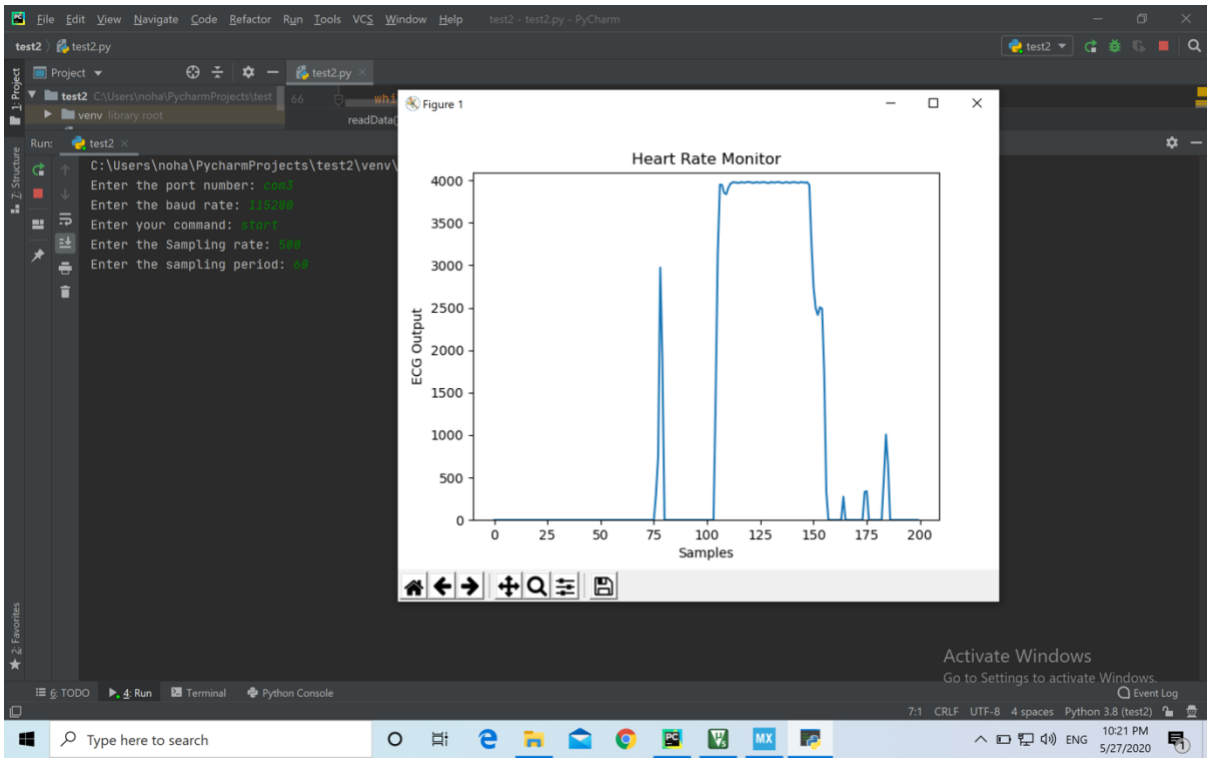
```
def readData(samplingRate):
    global bpmFinal
    ecgList = []
    stopFlag = False
    samplingRate = int(samplingRate)
    looper = int(0.25*samplingRate)
    for i in range(looper):
        line = ser.readline().decode('utf-8')
        line = line.strip("\n")
        line = line.strip("\r")
        #print("val = ", str(line))
        if(line.find('B') != -1):
            stopFlag = True
            bpmFinal = line
            return ecgList, stopFlag
        if ((len(line) > 0) and (len(line) < 5) and (line.find('\r') == -1)):
            line = float(line)
            ecgList.append(line)

    return ecgList, stopFlag
```

The amount of data that I store for plotting is a function of the sampling rate. In other words, if the sampling rate is x, then I plot 25% of x at a time. I have a stopping condition for the animate function so that I could stop plotting once I receive the BPM values, therefore when the input has B, I set a flag to true. Which is why the readData function returns both the flag and the list of data.

Results:

The following are pictures that describe the input process and show the output of running the application.




```
Enter your command: bpm
BPM = 99
```