HEART RATE MONITOR

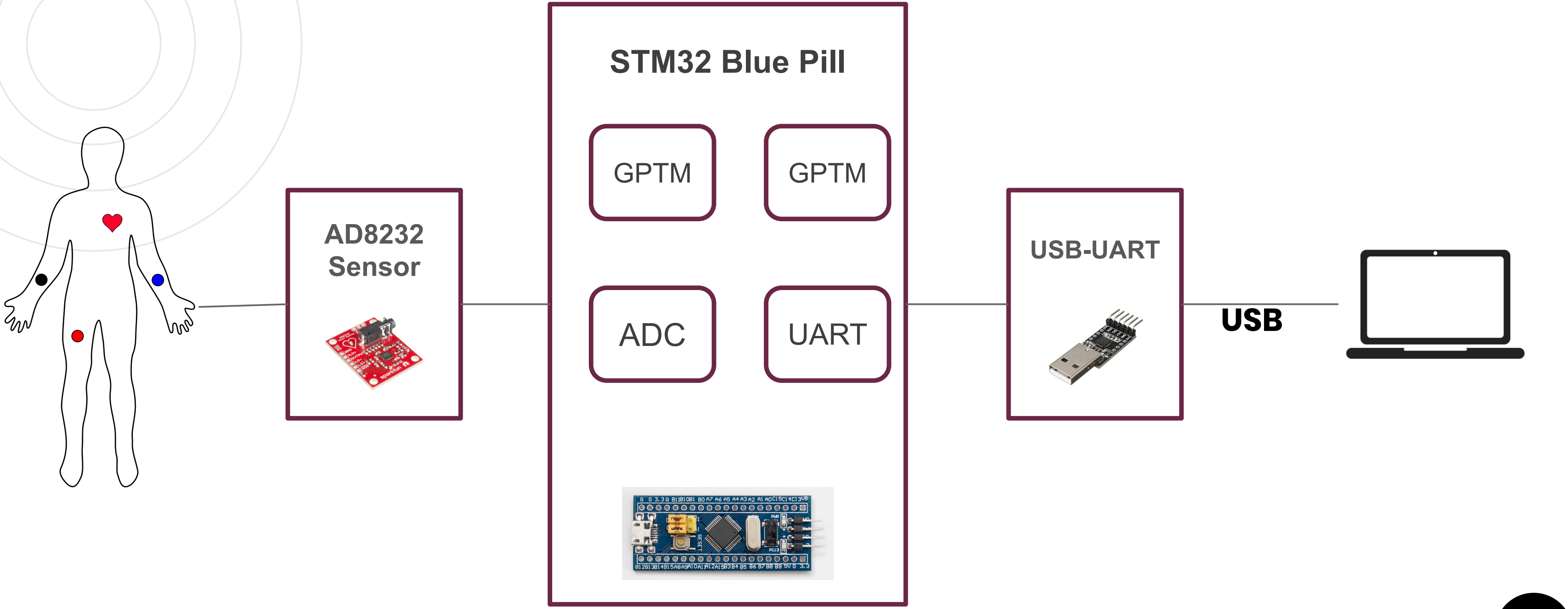
Manar Abdelatty

https://github.com/Manarabdelaty/HeartSesnor

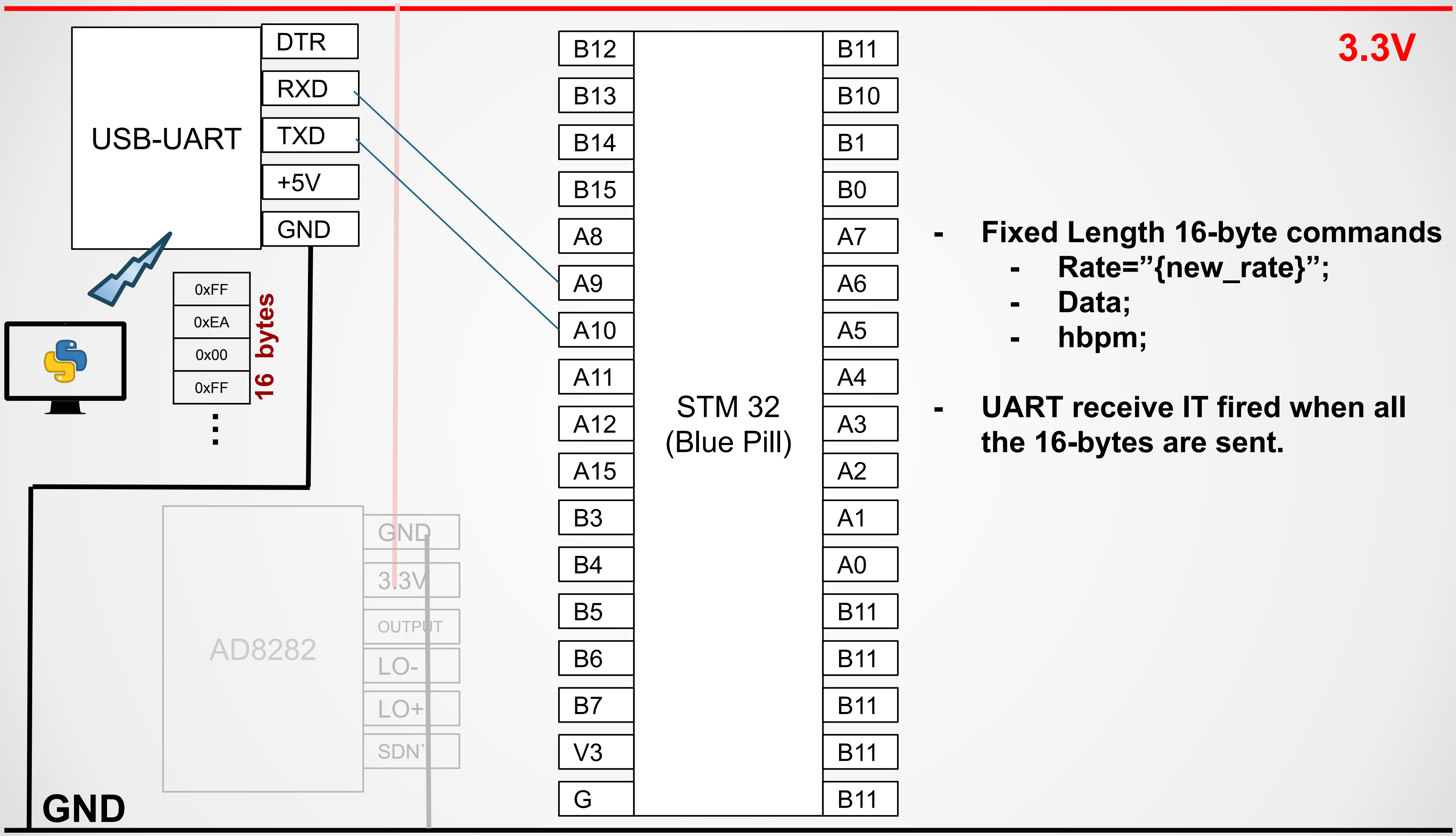
Requirements

- Set ADC sample rate [150 SPS -- 1000 SPS]
- Collect ADC data for one minute
- Send ADC data over UART
- UART Baud rate at least (40 KHz)
- Display ECG signal real-time
- Calculate heart beats per minute

Block Diagram



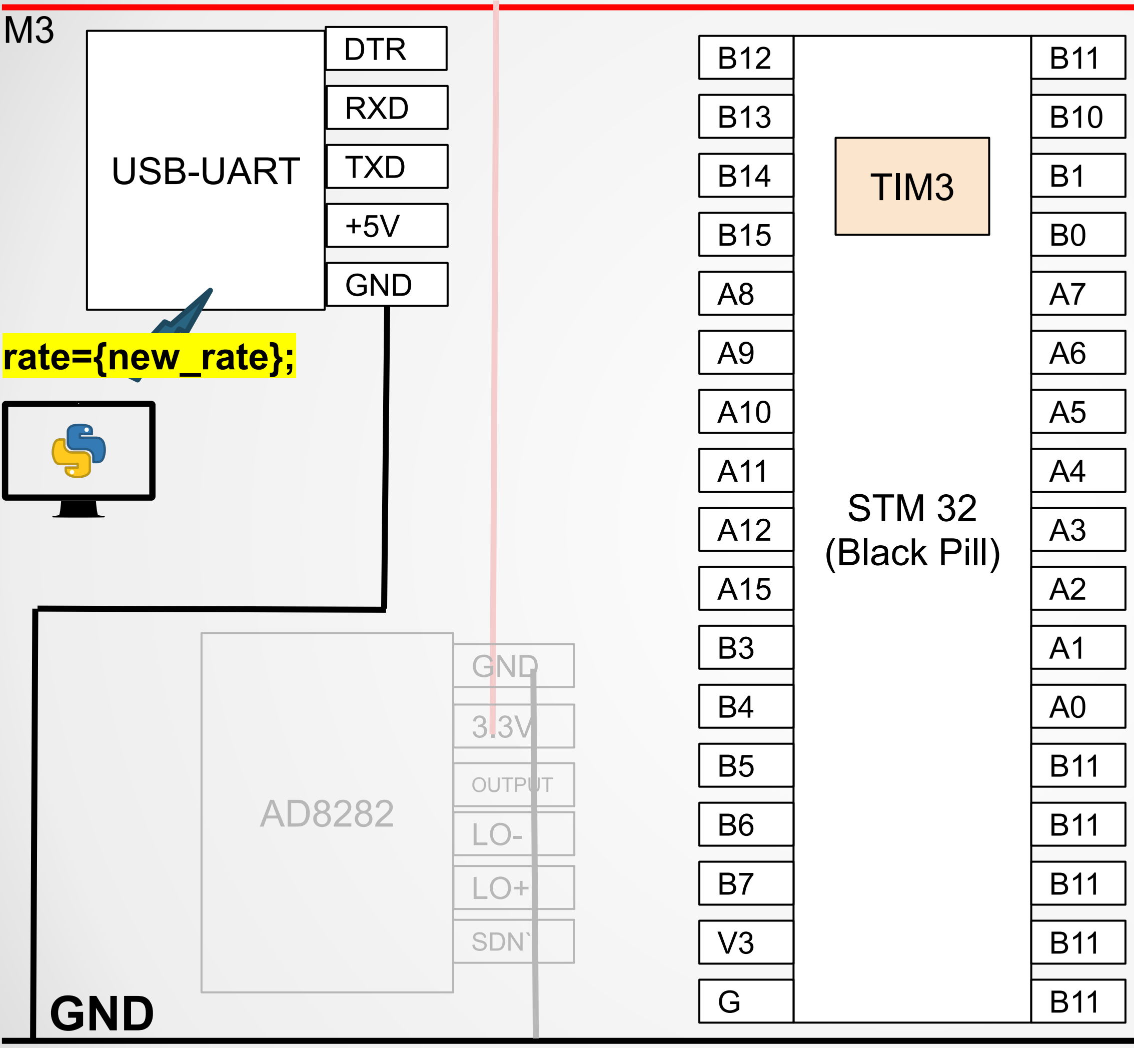
Design



- Fixed Length 16-byte commands
    - Rate="{new_rate}";
    - Data;
    - hbpm;

- UART receive IT fired when all the 16-bytes are sent.

TIM3



1) **Sample Rate Command rate={new_rate};**

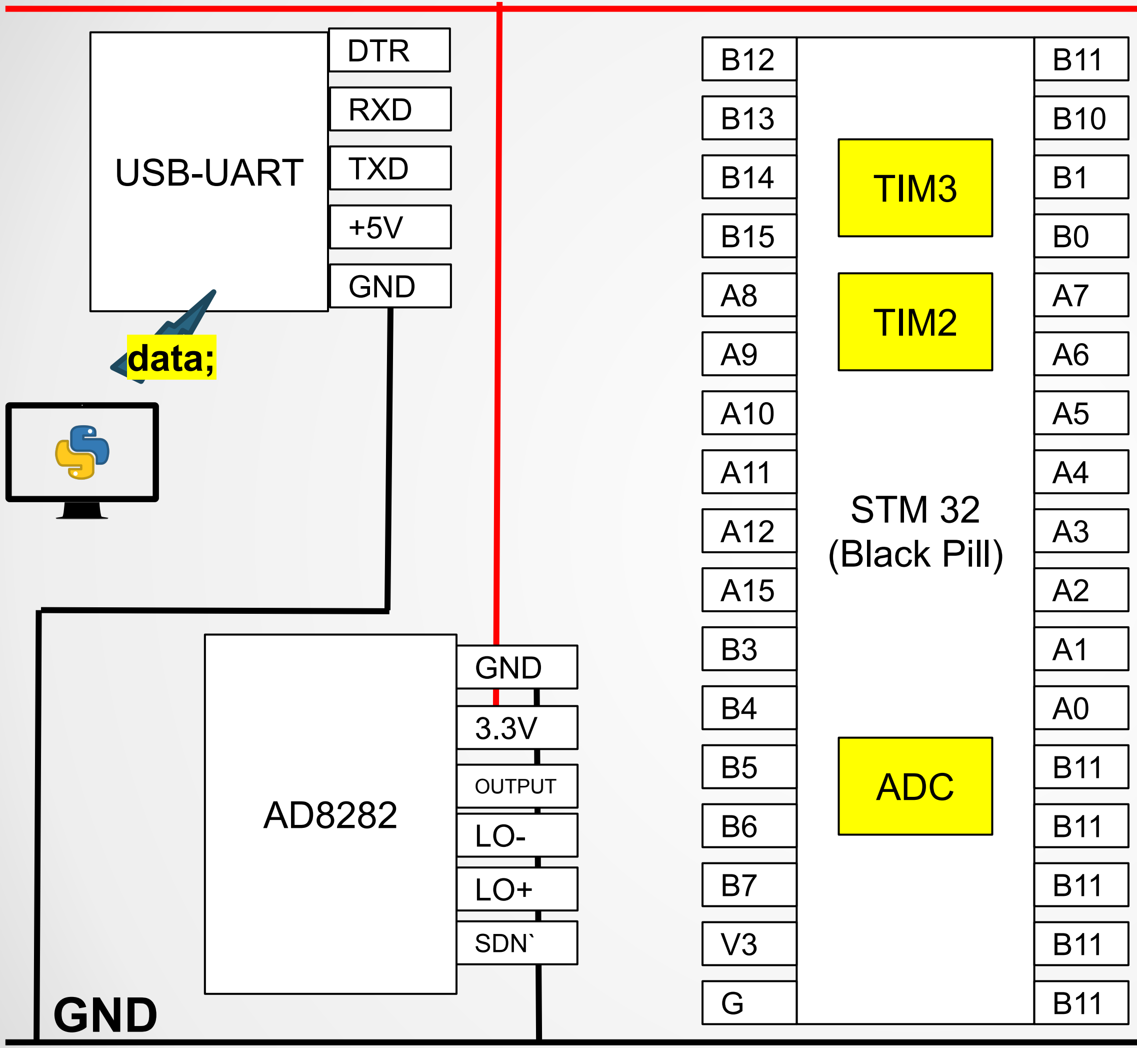
ARR = Sysclk / pre-scaler / new_rate

Prescaler
3999

Sysclk
8MHz

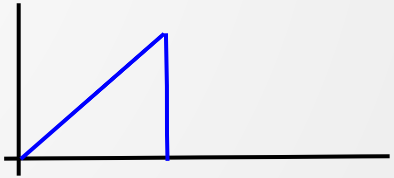
ARR
??

TIM3

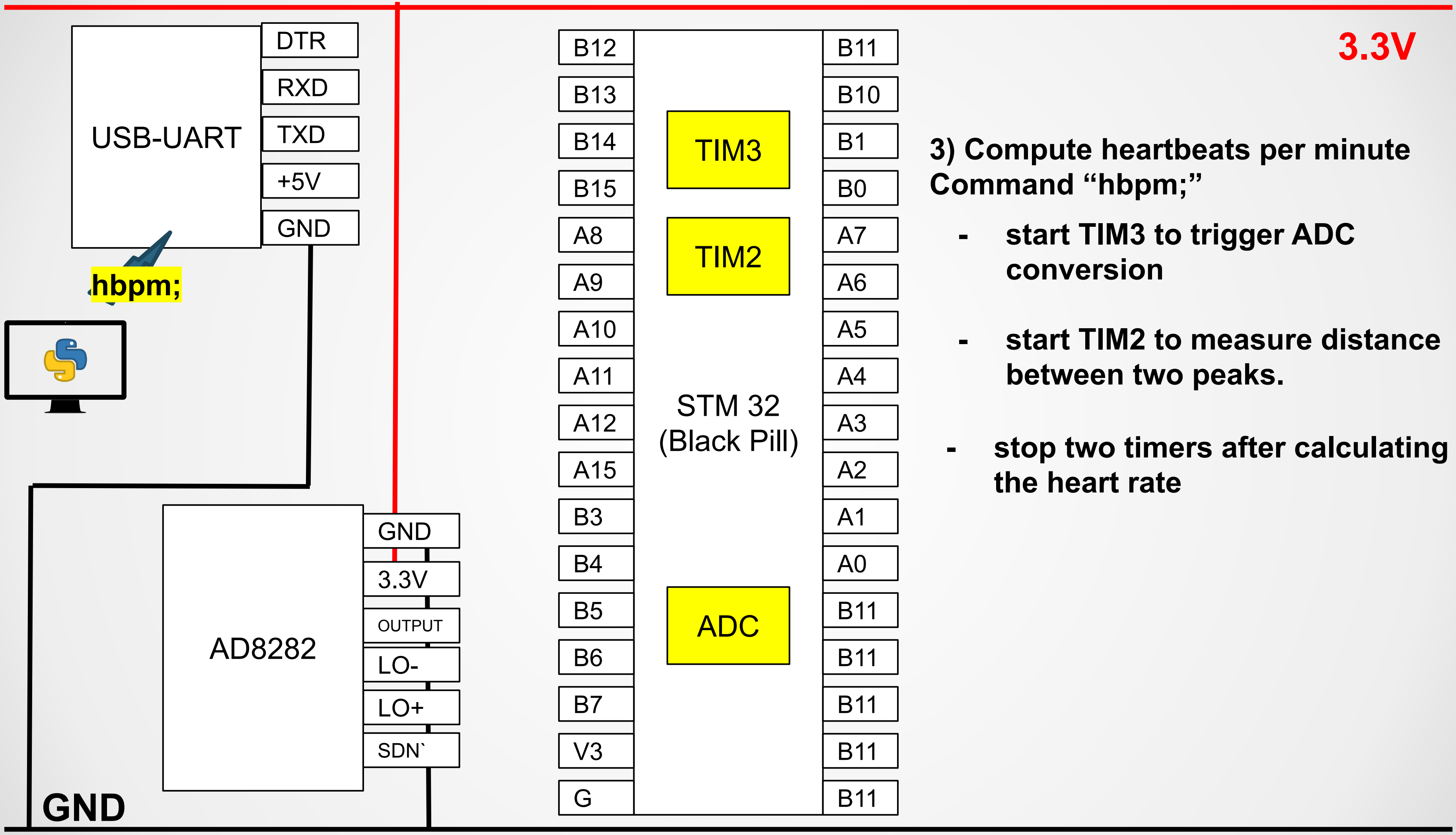


2) Collect Data Command “data;”

- start TIM3 to trigger ADC conversion
- start TIM2 to count for one minute
- stop two timers after one minute elapses



TIM2 Counter Period

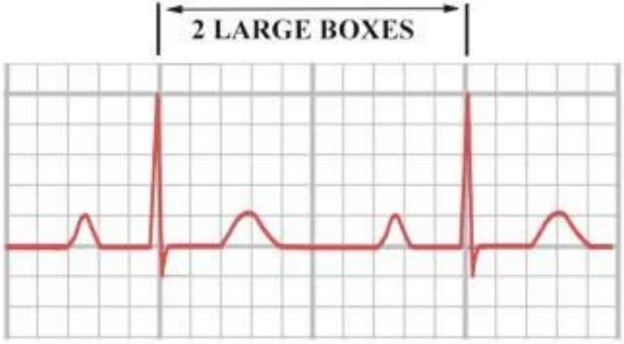


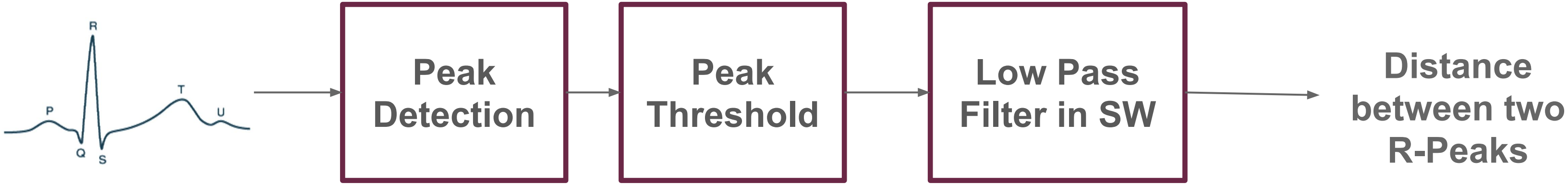
3) Compute heartbeats per minute
Command "hbpm;"

- **start TIM3 to trigger ADC conversion**
- **start TIM2 to measure distance between two peaks.**
- **stop two timers after calculating the heart rate**

Heart Rate Calculation

```
Bpm = 60 / peak_distance
```





Peak Detection

Peak Threshold

Low Pass Filter in SW

Distance between two R-Peaks

3

SW Archietecutre

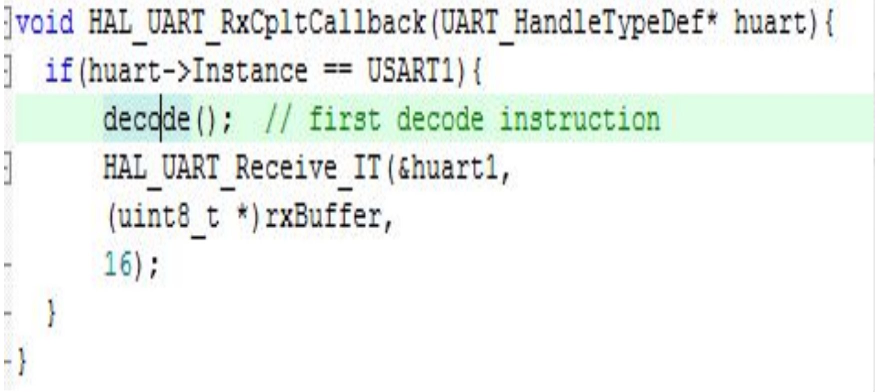
Round Robin with Interrupts

- **UART_Receive_Interrupt**
  - Receives commands from the Python application.

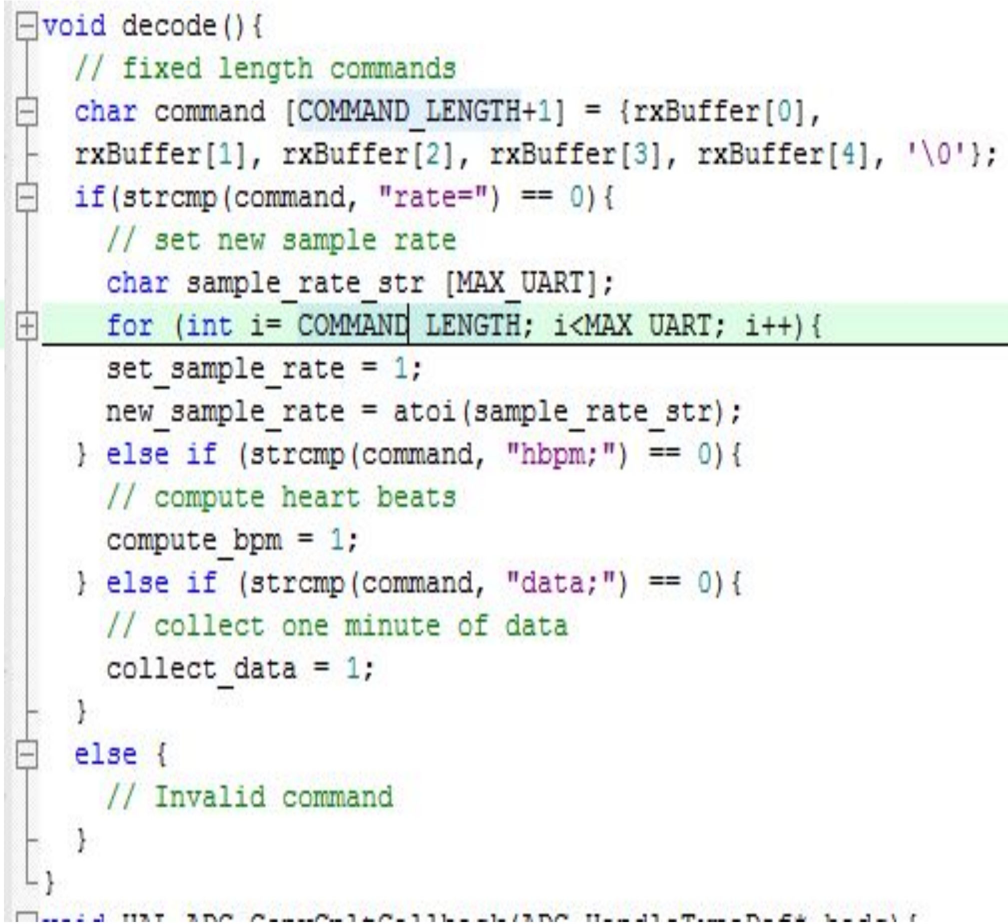
- **TIM2_Period_Elapsed_Interrupt**
  - Generated when one minute passes

- **ADC_Conversion_Interrupt**
  - Generated on one complete ADC conversion

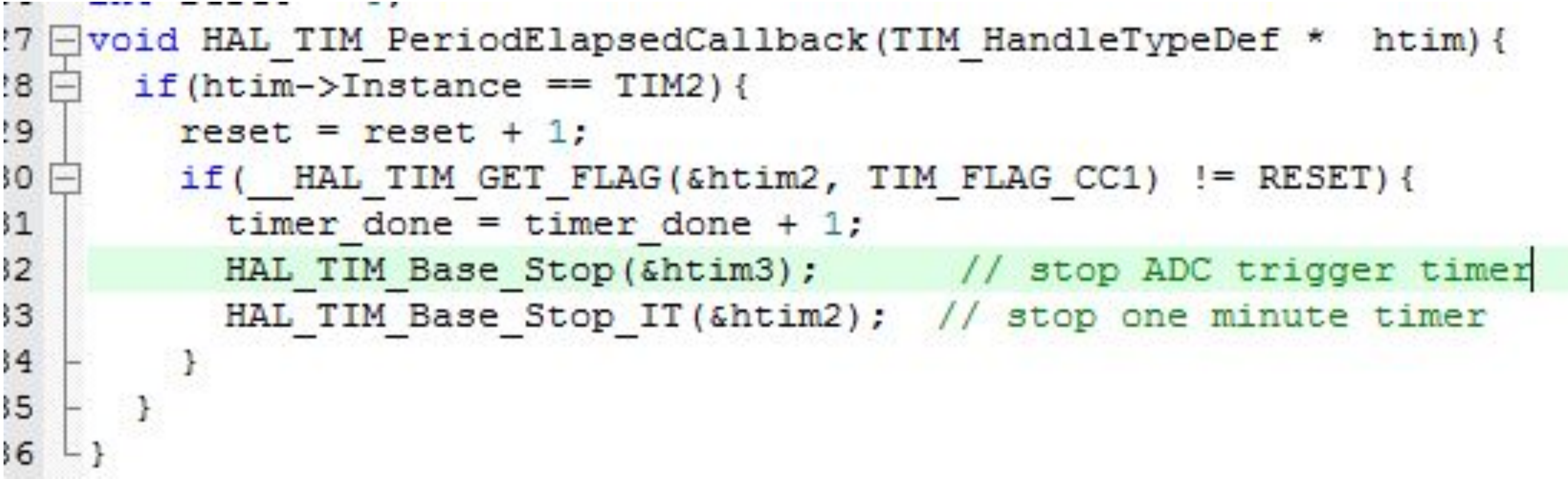
UART_Receive_Interrupt



```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart){
  if(huart->Instance == USART1){
      decode();  // first decode instruction
      HAL_UART_Receive_IT(&huart1,
      (uint8_t *)rxBuffer,
      16);
  }
}
```

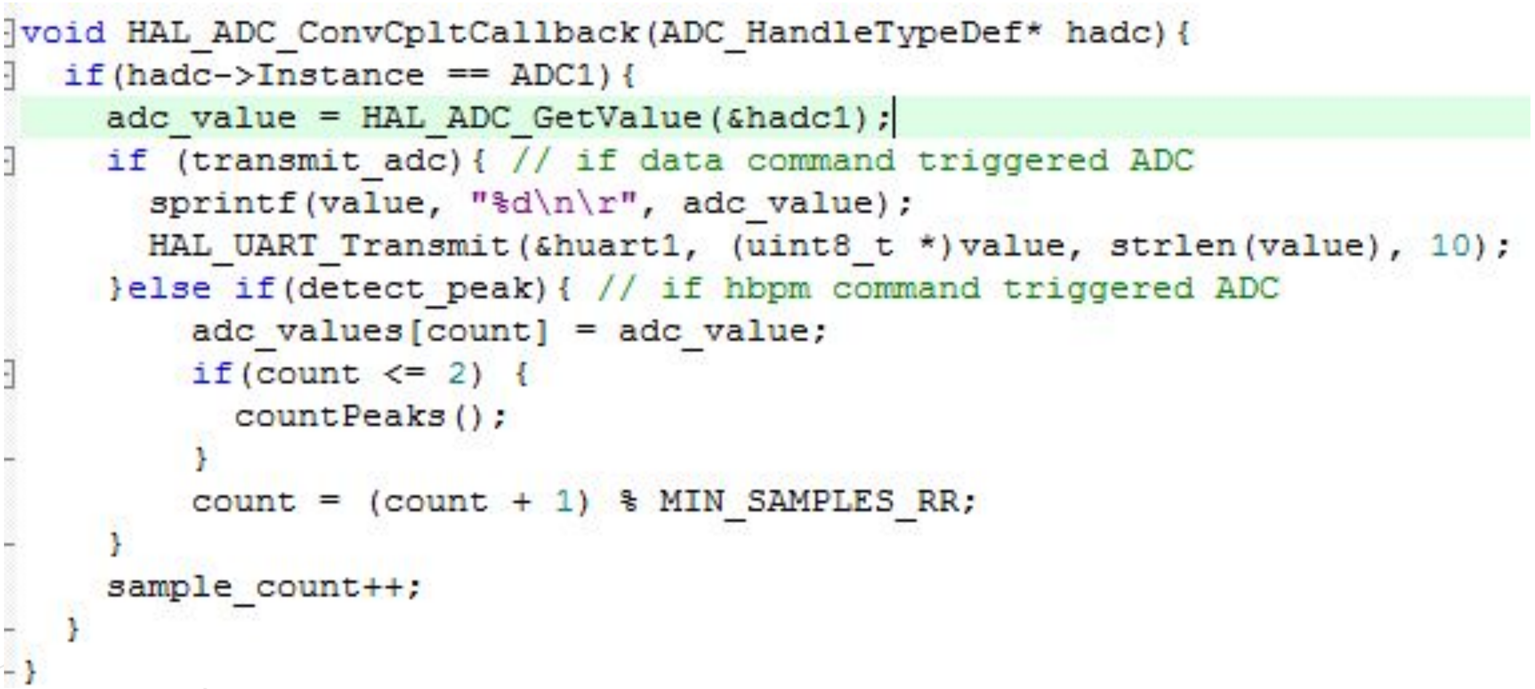


```
void decode(){
  // fixed length commands
  char command [COMMAND_LENGTH+1] = {rxBuffer[0],
  rxBuffer[1], rxBuffer[2], rxBuffer[3], rxBuffer[4], '\0'};
  if(strcmp(command, "rate=") == 0){
    // set new sample rate
    char sample_rate_str [MAX_UART];
    for (int i= COMMAND_LENGTH; i<MAX_UART; i++){
    set_sample_rate = 1;
    new_sample_rate = atoi(sample_rate_str);
  } else if (strcmp(command, "hbpm;") == 0){
    // compute heart beats
    compute_bpm = 1;
  } else if (strcmp(command, "data;") == 0){
    // collect one minute of data
    collect_data = 1;
  }
  else {
    // Invalid command
  }
}
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){
```



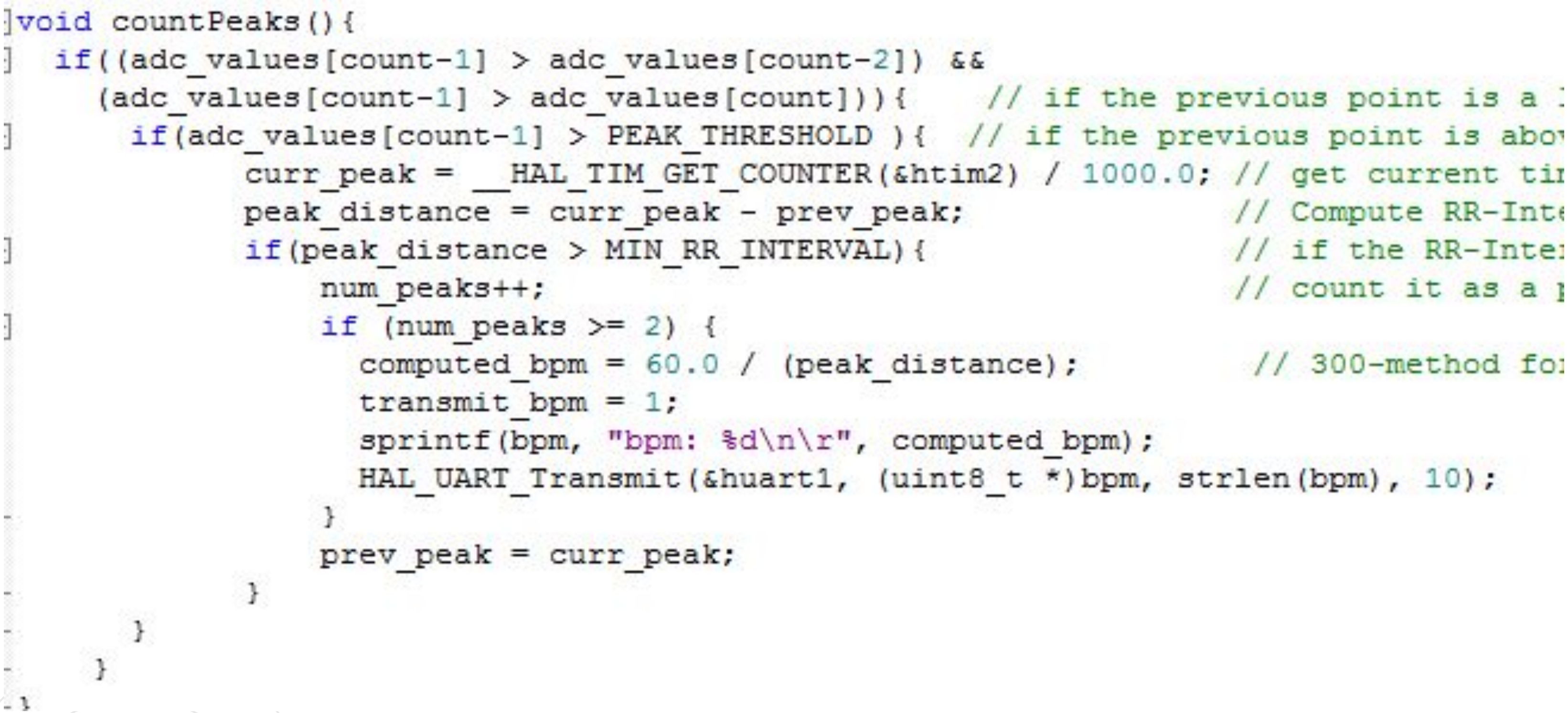
```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *  htim){
  if(htim->Instance == TIM2){
    reset = reset + 1;
    if(__HAL_TIM_GET_FLAG(&htim2, TIM_FLAG_CC1) != RESET){
      timer_done = timer_done + 1;
      HAL_TIM_Base_Stop(&htim3);      // stop ADC trigger timer
      HAL_TIM_Base_Stop_IT(&htim2);  // stop one minute timer
    }
  }
}
```

TIM2_Period_Elapsed_Interrupt



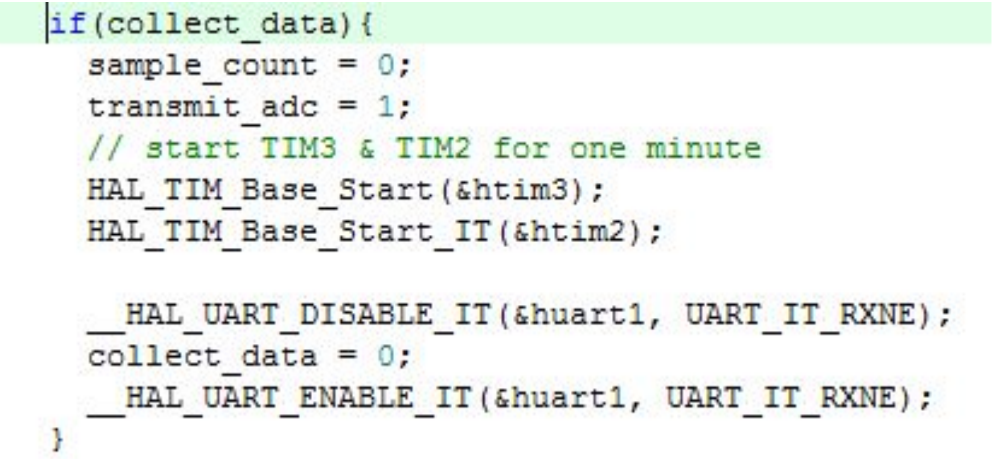
```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){
 if(hadc->Instance == ADC1){
  adc_value = HAL_ADC_GetValue(&hadc1);
  if (transmit_adc){ // if data command triggered ADC
   sprintf(value, "%d\n\r", adc_value);
   HAL_UART_Transmit(&huart1, (uint8_t *)value, strlen(value), 10);
  }else if(detect_peak){ // if hbpm command triggered ADC
    adc_values[count] = adc_value;
    if(count <= 2) {
     countPeaks();
    }
    count = (count + 1) % MIN_SAMPLES_RR;
  }
  sample_count++;
 }
}
```

ADC_Conversion_Complete_Interrupt



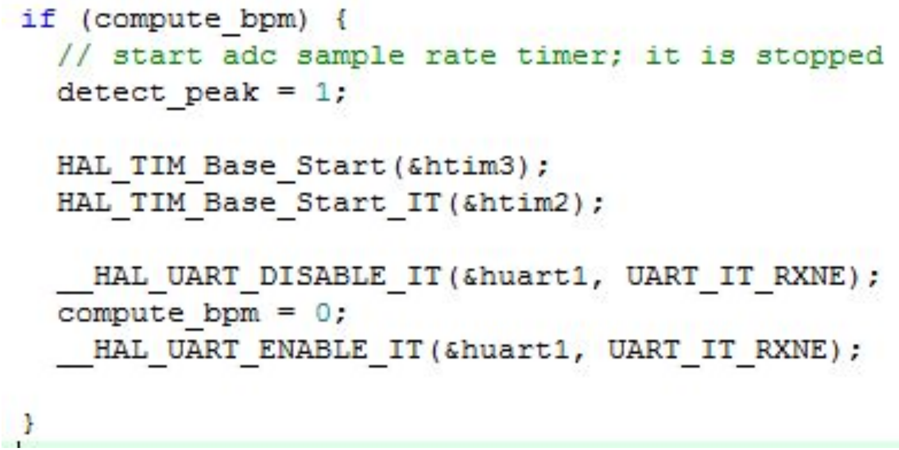
```
void countPeaks(){
  if((adc_values[count-1] > adc_values[count-2]) &&
    (adc_values[count-1] > adc_values[count])){    // if the previous point is a
      if(adc_values[count-1] > PEAK_THRESHOLD ){  // if the previous point is abov
            curr_peak = __HAL_TIM_GET_COUNTER(&htim2) / 1000.0; // get current ti
            peak_distance = curr_peak - prev_peak;              // Compute RR-Inte
            if(peak_distance > MIN_RR_INTERVAL){                // if the RR-Inte
                num_peaks++;                                    // count it as a
                if (num_peaks >= 2) {
                  computed_bpm = 60.0 / (peak_distance);         // 300-method fo
                  transmit_bpm = 1;
                  sprintf(bpm, "bpm: %d\n\r", computed_bpm);
                  HAL_UART_Transmit(&huart1, (uint8_t *)bpm, strlen(bpm), 10);
                }
                prev_peak = curr_peak;
            }
      }
    }
}
```

Count Peaks



```
if(collect_data){
  sample_count = 0;
  transmit_adc = 1;
  // start TIM3 & TIM2 for one minute
  HAL_TIM_Base_Start(&htim3);
  HAL_TIM_Base_Start_IT(&htim2);

  __HAL_UART_DISABLE_IT(&huart1, UART_IT_RXNE);
  collect_data = 0;
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);
}
```

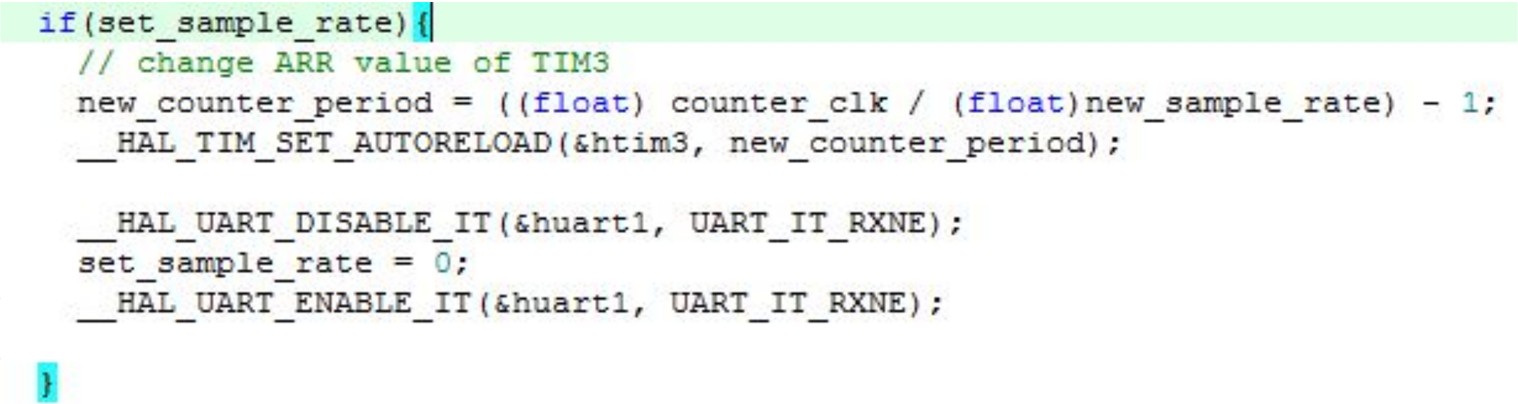


```
if (compute_bpm) {
  // start adc sample rate timer; it is stopped
  detect_peak = 1;

  HAL_TIM_Base_Start(&htim3);
  HAL_TIM_Base_Start_IT(&htim2);

  __HAL_UART_DISABLE_IT(&huart1, UART_IT_RXNE);
  compute_bpm = 0;
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);

}
```



```
if(set_sample_rate){
  // change ARR value of TIM3
  new_counter_period = ((float) counter_clk / (float)new_sample_rate) - 1;
  __HAL_TIM_SET_AUTORELOAD(&htim3, new_counter_period);

  __HAL_UART_DISABLE_IT(&huart1, UART_IT_RXNE);
  set_sample_rate = 0;
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);

}
```

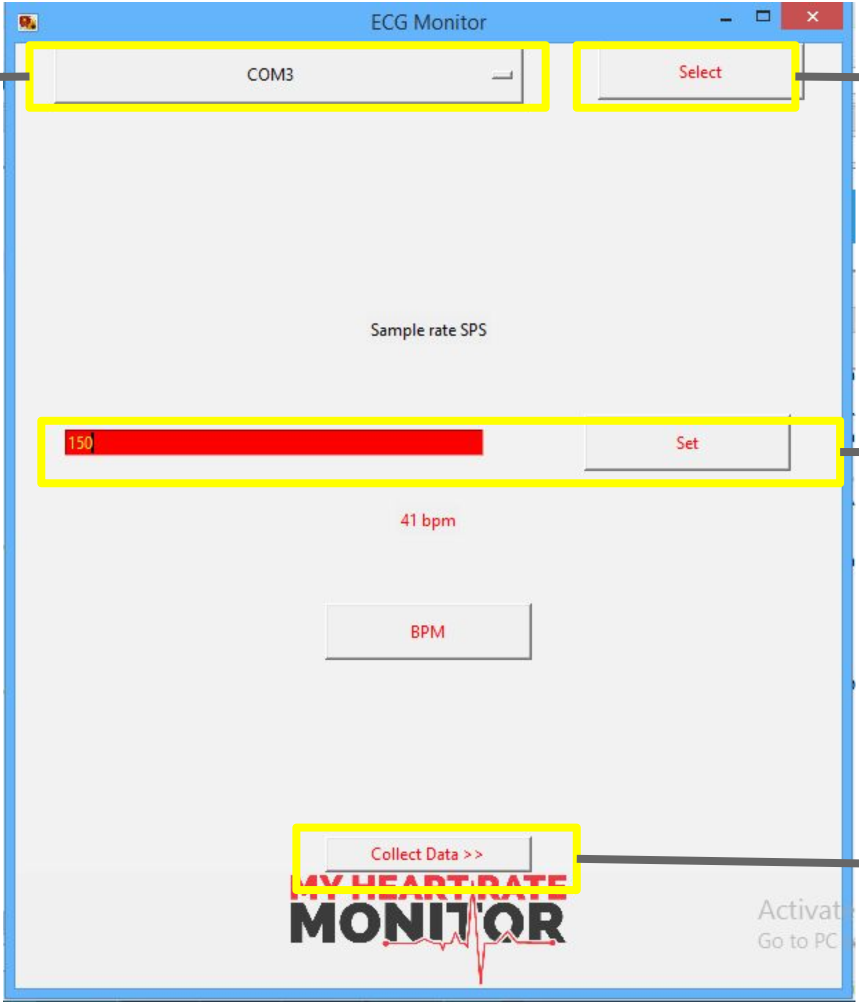
Main Loop

Python Application

- **Pyserial**
  - UART communication

- **Matplotlib Animation**
  - Real-time plotting

- **tkinter**
  - GUI



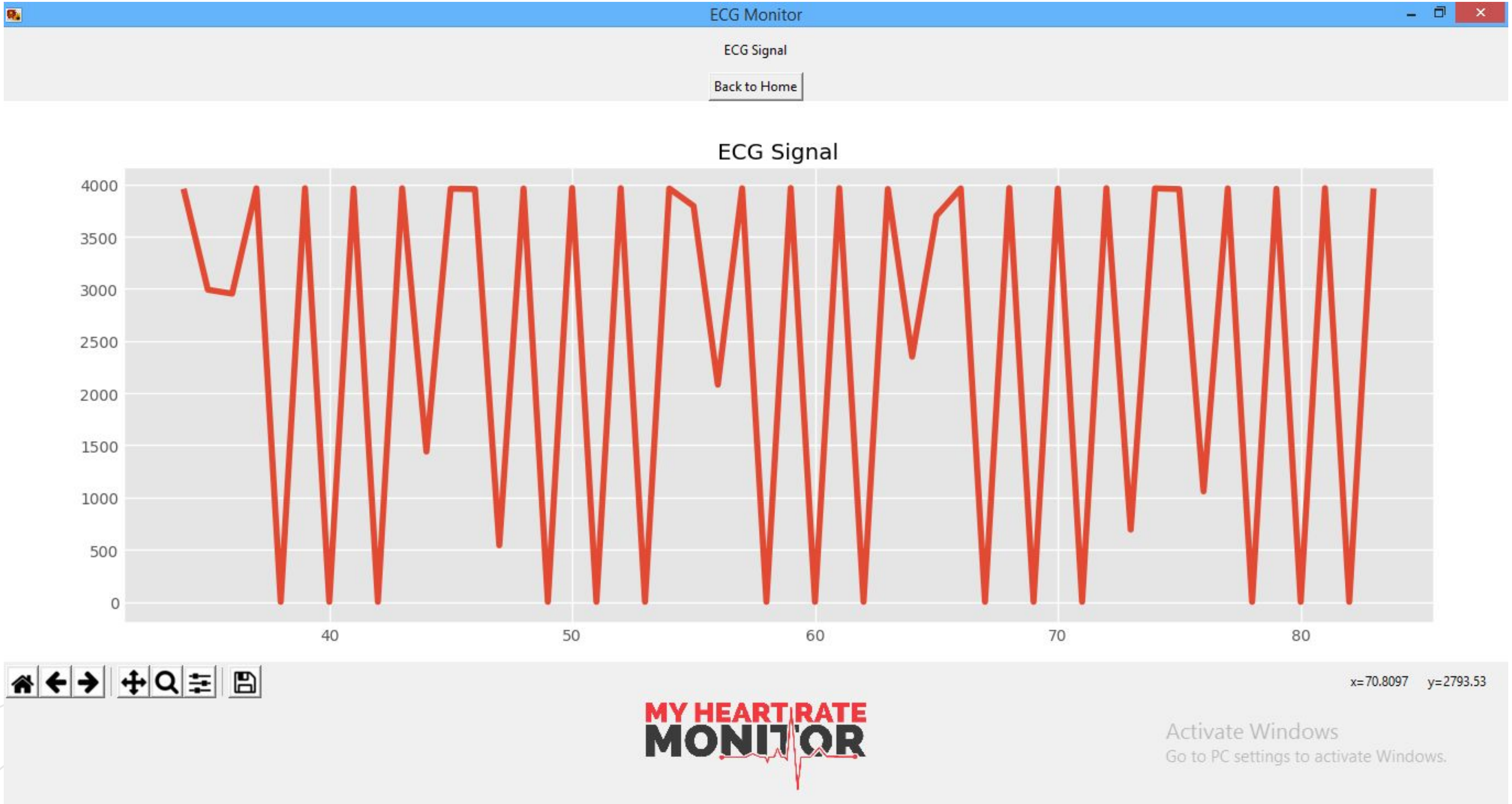
Available ports list

Opens the selected port.

Sends the rate command

Opens a new page to collect one minute of data

Main Page



Real-Time Display

DEMO