PULSE OXIMETER DESIGN

ARDA ÜNAL

1. INTRODUCTION

Pulse oximeters are noninvasive medical devices which are used for measuring blood oxygen saturation. Peripheral oxygen saturation(SpO_2) readings are typically within 2% accuracy of the more accurate than the reading of the arterial oxygen saturation(SaO_2) from arterial blood gas analysis. (REF 2)

The working principle of the pulse oximeter is based on different light absorption characteristics of oxygenated and deoxygenated blood. Pulse oximetry is based on the principle that oxyhemoglobin (O_2Hb) and deoxyhemoglobin (HHb) have significant differences in absorption at red and near-IR light because these 2 wavelengths penetrate tissues well whereas blue, green, yellow, and far-IR light are significantly absorbed by nonvascular tissues and water. O_2Hb absorbs greater amounts of IR light and lower amounts of red light than does HHb. On the other hand, HHb absorbs more red light andispears less red, which are illustrated in Figure 1. Exploiting this difference in light absorption properties between O_2Hb and HHb, pulse oximeters emit two wavelengths of light, red at 660 nm and near-IR at 940 nm from a pair of LEDs located in one arm of the finger probe. The emitted lights are detected by a photodiode on the opposite arm of the probe with a finger between 2 probes of the device.(REF 2)

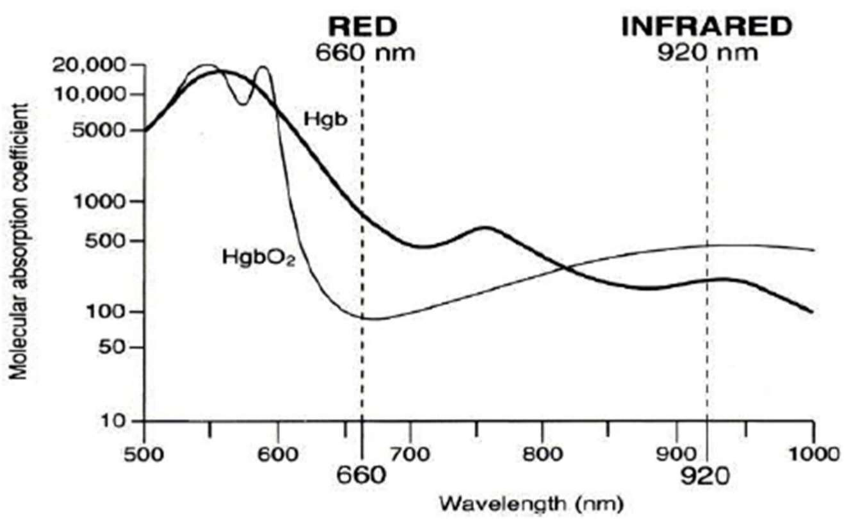


Figure 1. Absorption curves for both types of hemoglobin

Pulse oximetry is useful in many situations such as for assessment of any patient's oxygenation and determining the need for supplemental oxygen, pilots that are in unpressurized aircraft, mountain climbers, athletes whose oxygen levels may decrease during exercise and so on. Additionally, there are different types of pulse oximeter devices available such as portable ones which are generally supplied by 2xAAA batteries. They use a simple LCD screen to show the current oxygen level of blood. In addition to that wireless pulse meters are used in hospitals to not create crowded cabling.(REF 1)

2. THEORETICAL PROCEDURE(REF)

The ability of pulse oximetry to detect SpO_2 of only arterial blood is based on the principle that the amount of red and IR light absorbed fluctuates with the cardiac cycle, as the arterial blood volüme increases during systole and decreases during diastole. A portion of the light passes through tissues without being absorbed strikes the probe's photodetector and, accordingly, crates signals with a relatively stable and non-pulsatile(DC) and a pulsatile(AC) component as show Figure 2.

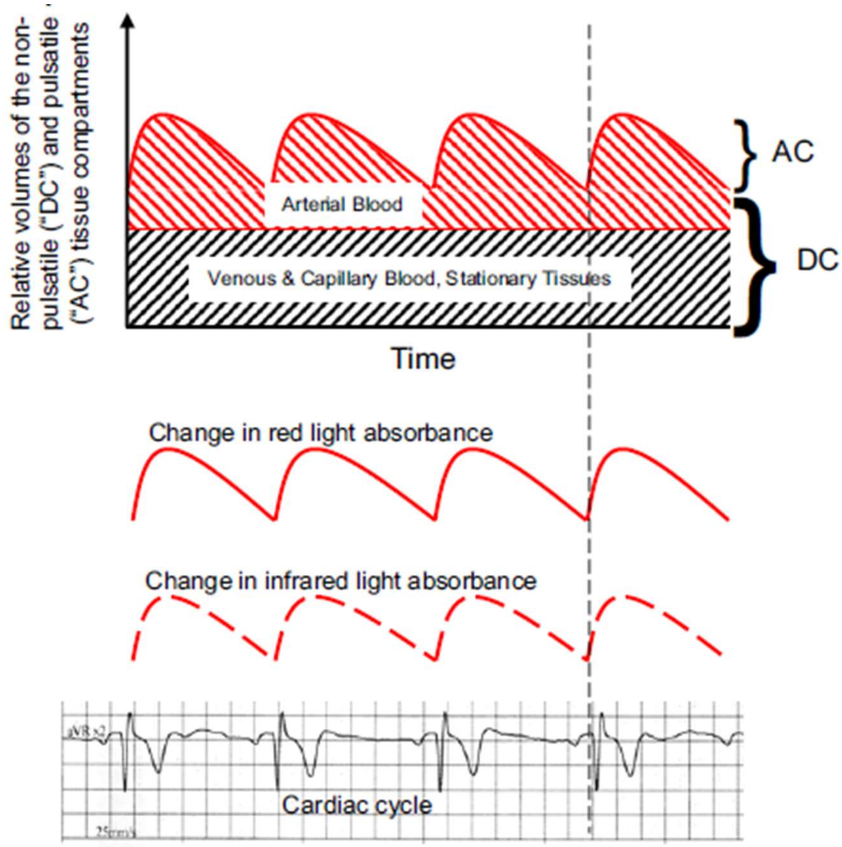


Figure 2. AC and DC components of red and IR LED pulses

The Pulse oximeters use amplitüde of the absorbances to calculate Red:IR Modulation Ratio(R) $R = (\frac{A_{red,AC}}{A_{red,DC}})/(\frac{A_{IR,AC}}{A_{IR,DC}})$ where $A = absorbances$. In other words, R is a double-ratio of the pulsatile and non-pulsatile components of red light absorption to IR light absorption. At low arterial oxygen saturations, the relative change in amplitude of the red light absorbance due to the pulse is greater than the IR absorbance. It means that $A_{red,AC} > A_{IR,AC}$ resulting in a higher R value. Conversely, at higher oxygen saturation $A_{IR,AC} > A_{red,AC}$ and the R value is lower as show below in Figure 3.

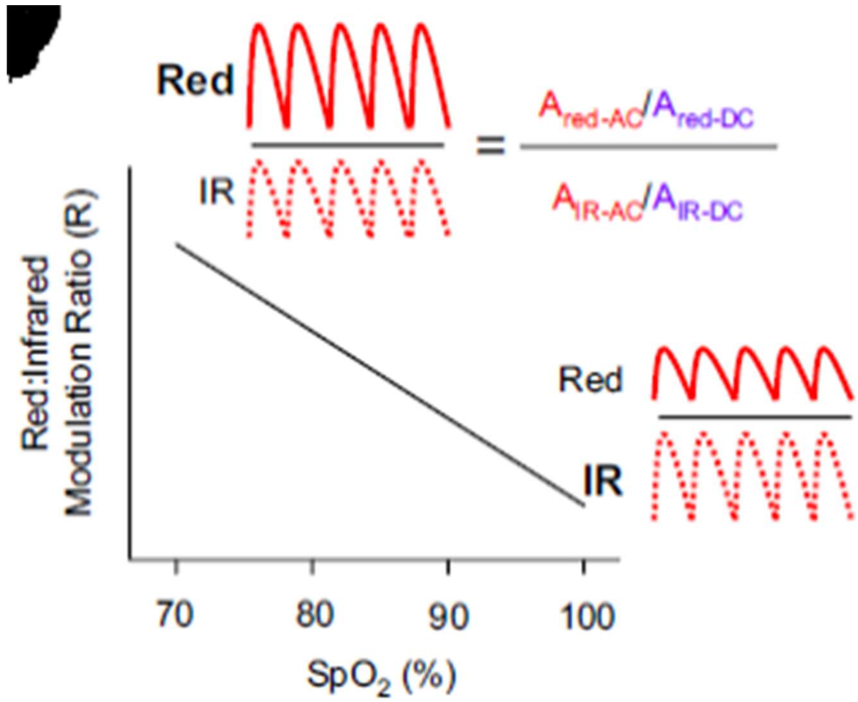


Figure 3 A diagram of a calibration curve of the Red:IR Modulation Ration in relation to the SpO_2

A microprocessor in pulse oximeters uses this ratio to determine the SpO_2 based on a calibration curve that was generated empirically bu measuring R in healthy volunteers whose saturations were altered from 100% to approximately 70%. Thus SpO_2 readings below 70% should not be considered quantitatively reliable although it is unlikely any clinical decisions would be altered based on any differences in SpO_2 measured below 70%.

3. TECHNICAL PROCEDURE

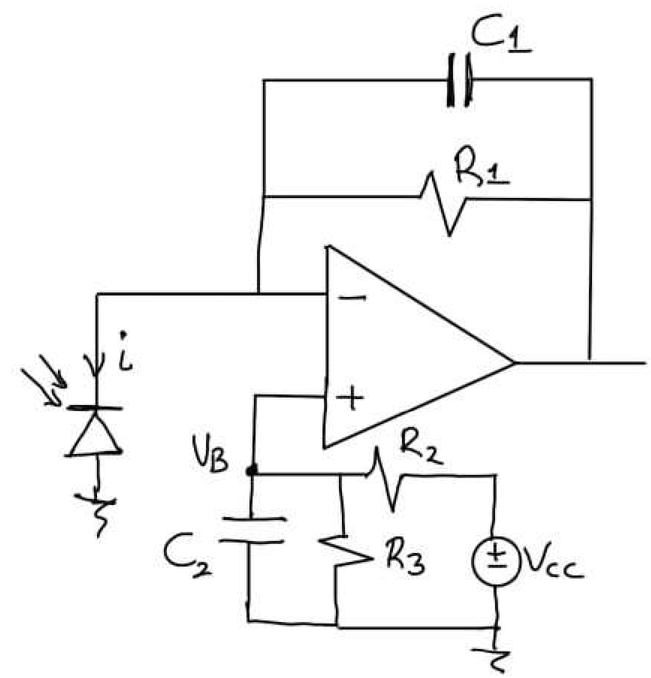
4.

3.1 ELECTRONIC CIRCUIT DESIGN PROCESS

3.1.1 HAND CALCULATIONS

First of all, from an electronic component website, I analyzed some circuit elements such as photodiode, OPAMP, IR and red LED. After I chose these essential components according to their prices, performance values and datasheets. I mainly used a reference(REF 1) which was a master thesis on pulse oximetry. In this thesis, the current coming from the photodiode is mentioned as a few milliamps(REF 1), therefore after I chose OPAMP, I made some theoretical calculations shown below:

Transimpedance Amplifier



$$V_{out} = iR_1 + V_B = iR_1 + V_{cc}\frac{R_3}{R_3+R_2}$$

Photodiode parameters:

$$V_R = 0V \Rightarrow C_j = 70pF$$

$$E_e = 1mW/cm^2,\quad V_R = 5 \Rightarrow I_r = 10\mu A$$

Gain Calculation:

$$\frac{V_{out}(max) - V_{out}(min)}{I_{IN}(max)} = R_1 = \frac{3.3 - \frac{3.3}{31k}\cdot 1k}{10^{-4}}$$

$$= 320k\Omega$$

Feedback Capacitor Calculation:

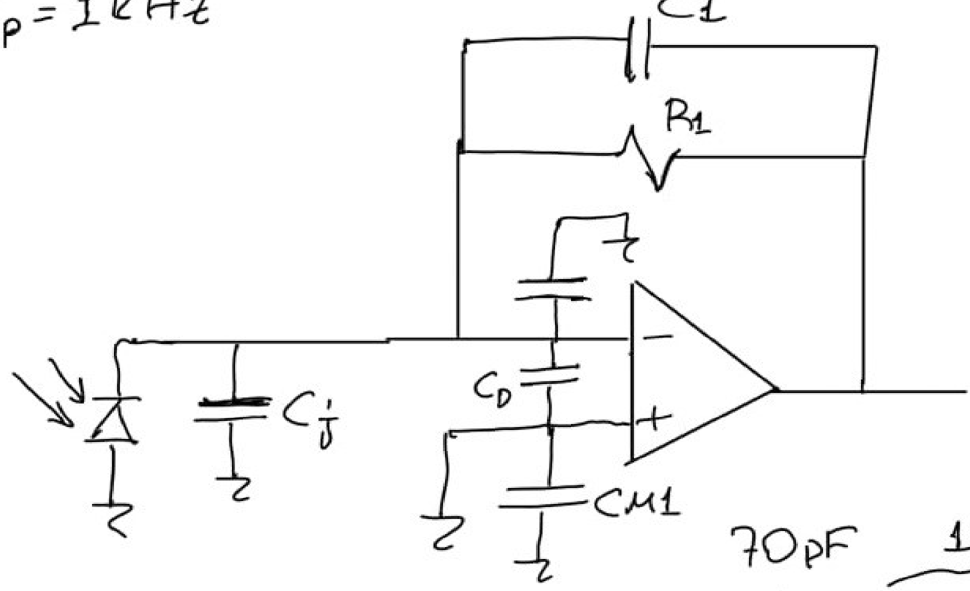
$$f_p = \frac{1}{2\pi C_1 R_1} \Rightarrow C_1 = \frac{1}{2\pi . 320k . 1k} = 497\ pF$$

$$\cong 500pF$$

Amplifier Gain BW Calculation:

→ Small signal model of the OPAMP

$f_p = 1\,kHz$



→ At inverting input $C_T = C_j + \overbrace{C_D}^{70pF} + \overbrace{C_{CM2}}^{15\,pF}$

→ The capacitance at the inverting input of the OP-AMP produces a zero at:

$$f_z = \frac{1}{2\pi (C_1 + C_T) . R_1} = 5.85k\,Hz$$

→ In the region above the pole, the freq. of intersection is:

$$f_I = \frac{C_1}{C_T + C_1} f_{GBW} \cong 176.5kHz$$

, where f_{GBW} is the unity bandwidth of the OP-AMP. ($f_{GBW} = 1\,MHz$)

→ Thus, in order to ensure stability, the unity BW of the OP-AMP must obey the rule:

$$f_I > f_P$$ → Condition is satisfied!

Bias Network:

$$V_B = V_{CC} \frac{R_3}{R_2 + R_3}$$ → To prevent the amp. output from saturating at the negative voltages.

→ The cap. C_2 is used to reduce PSU noise:

$$f_p = \frac{1}{2\pi C_2 (R_2 // R_3)}$$

→ $C_2 = 1\mu F$ would be sufficient.

→ The full power bandwidth of the OPAMP is

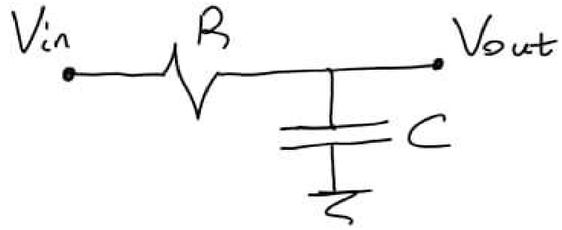
$$f_{FP} = \frac{SR}{2\pi A}$$

where SR is the slew rate of the OPAMP and A is the amplitude of the sinusoid.

→ Above that freq. the amplifier will produce a distorted output for full-scale signals.

$V_B \cong 0.1V$, $V_{CC} = 3.3V$ ⇒ $\frac{R_2}{R_3} = 32$ → $30k\Omega$ is found

Digital Filter Design



$$V_{out} + RC\frac{dV_{out}}{dt} = V_{in}$$

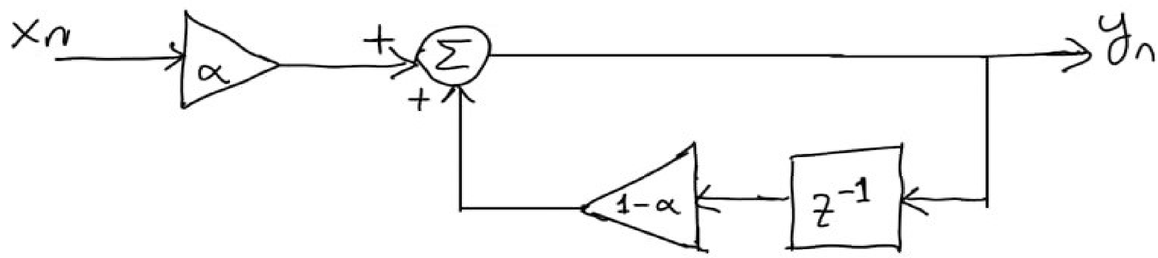
$$\frac{dV}{dt} = \frac{V[n] - V[n-1]}{T}$$ → Backward Euler Difference

$$V_{out}[n] + RC\frac{V_{out}[n] - V_{out}[n-1]}{T} = V_{in}[n]$$

$$\Rightarrow V_{out}[n] = \frac{T}{T+RC}V_{in}[n] + \frac{RC}{T+RC}V_{out}[n-1]$$

$$f_{-3dB} = \frac{1}{2\pi RC} = 1\,kHz \Rightarrow RC = 1.5915 \times 10^{-4}$$

EMA FILTER — LOWPASS



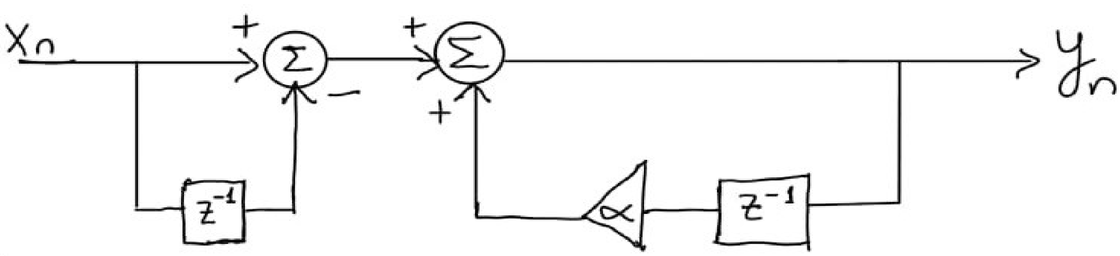
$$y_n = \alpha x_n + (1-\alpha) y_{n-1} , \quad 0 \leq \alpha \leq 1$$

for $\alpha = 0.1$, the cut-off freq. is close to 1 kHz.

HIGH PASS IIR FILTER

$$H(z) = \frac{1 - z^{-1}}{1 - \alpha z^{-1}} \Rightarrow (1 - \alpha z^{-1}) Y = X (1 - z^{-1})$$

$$\Rightarrow y_n = \alpha y_{n-1} + x_n - x_{n-1} , \quad \alpha = 0.992$$



↳ This digital filter is embeded in the algorithm!

3.1.2 LTSpice SIMULATIONS

After calculations, I simulated the circuit and played with values shown below in Figure 4.

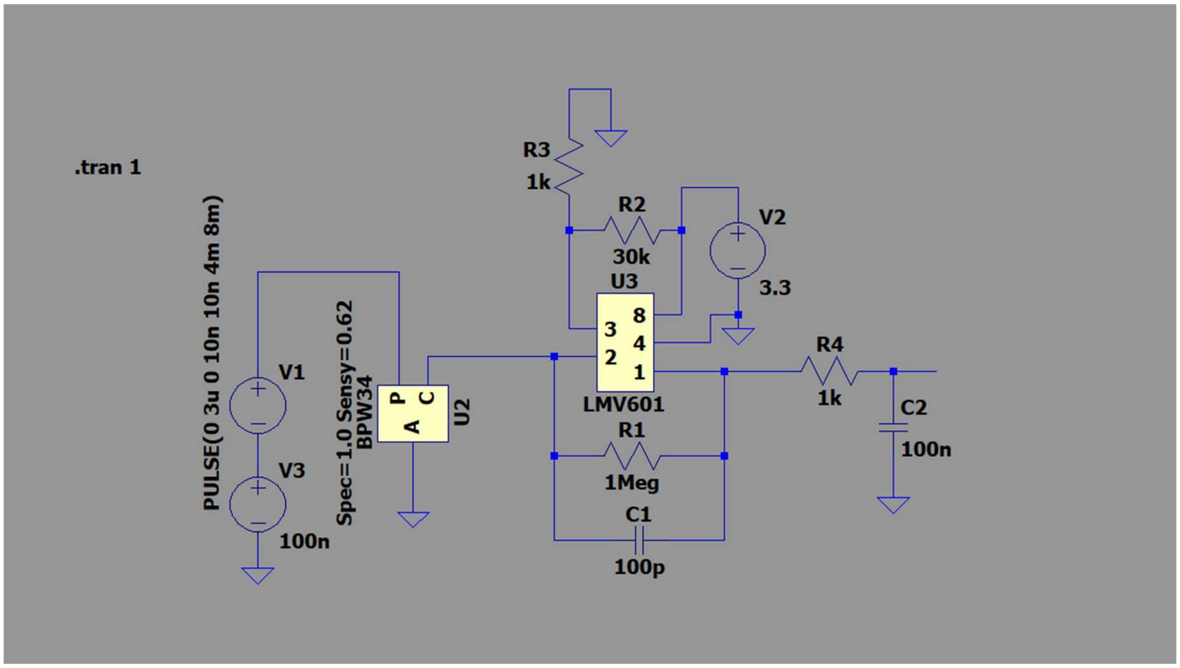


Figure 4. LTspice circuit simulation

The results of the simulation for the given waveform and DC offset input are shown in Figure 5 and Figure 6.

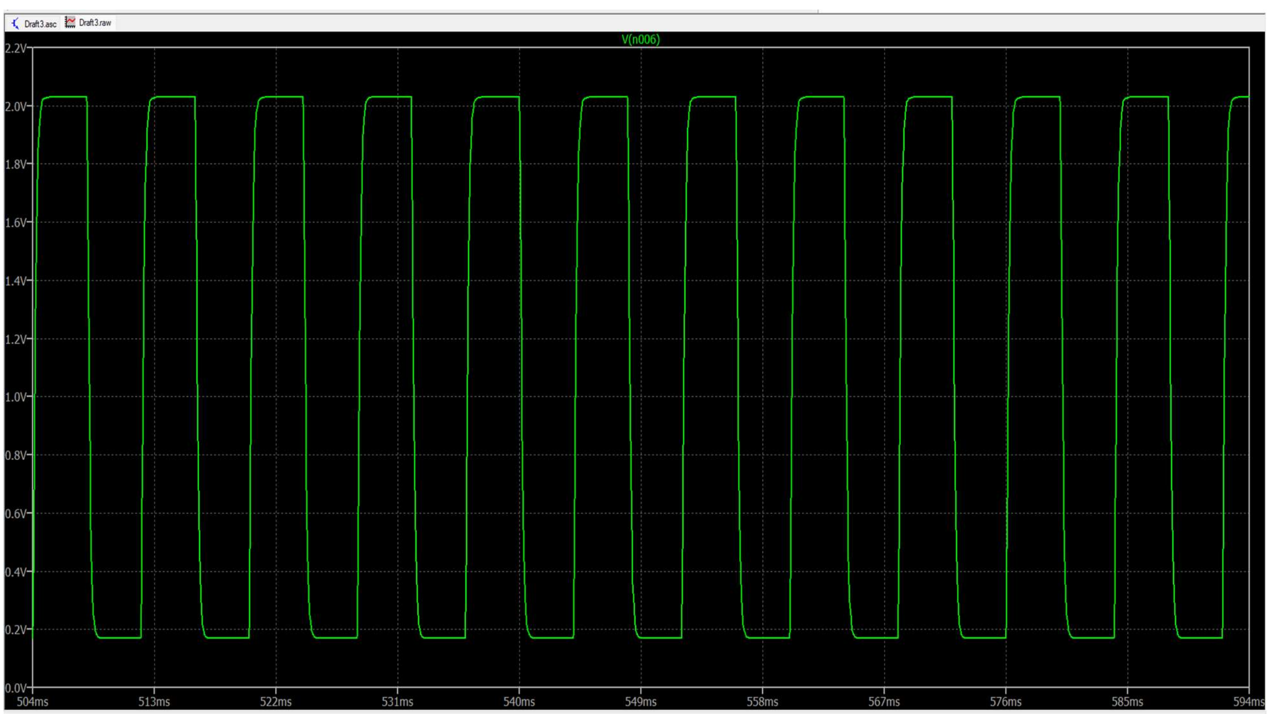


Figure 5. Output waveform of the Transimpedance Amplifier

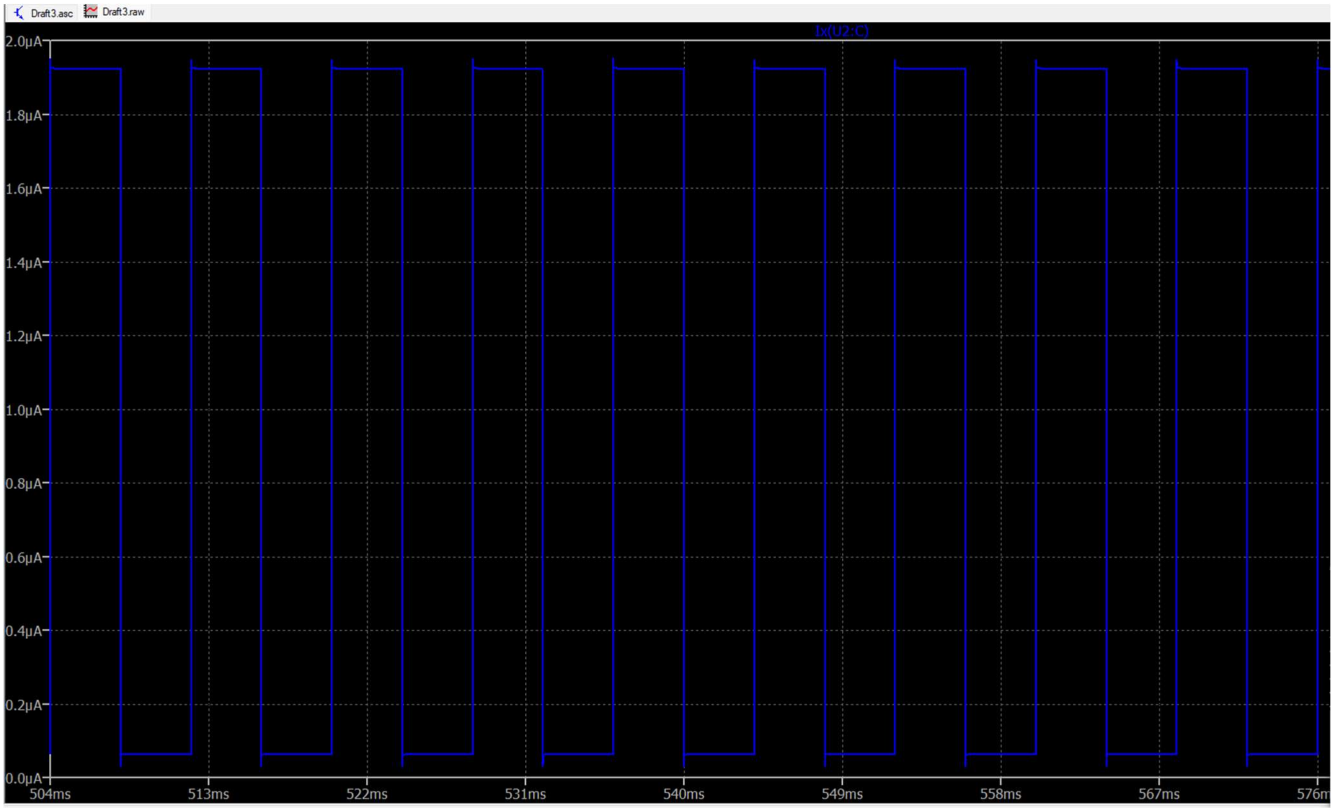


Figure 6. Photodiode current produced by pulsed signal

As a side note, I found a photodiode simulation file on the internet which also affected my decision to go through with this photodiode. I chose OPAMP such that it works between 2.7V and 5V and thus it is suitable for portable applications. It spends power only when it is needed via the Active low enable input pin, but for this pulsed application, I did not use this property. More details can be found in the datasheet(REF 6). I chose the Bluepill development board from ST. It has very less components on it therefore it occupies a small space. Moreover, as far as I know, ST MCUs are more reliable than Arduino MCUs, although I faced many problems which are mentioned in the software section.

3.1.3 PCB DESIGN

After validation of the circuit via simulation, I designed the PCB by using the KiCad open-source PCB design program. PCB is designed such that it is suitable for the handmade PCB process. In Figure 7 and Figure 8, schematics and PCB layout are shown respectively. I used a simple printer to print the PCB layout on oil paper and iron, I copied ink into the copper plate. After cleaning the copper plate with water until all paper is removed from the not ink places. After that using HCl and water with oxygen, I removed unnecessary coppers from the PCB. Then by using emery, I removed the ink. After that, I opened the holes for header pins and for Bluepill development board and solder all the components. I chose 0805 packages for resistors and capacitors because they are easy to solder.

As you can see from schematics, I used 3 different bypass capacitors on the power lines although I fed the circuit from the STM board. STM board has already a regulator and additional bypass capacitors, therefore I did not take care of the 50 Hz signal too much.

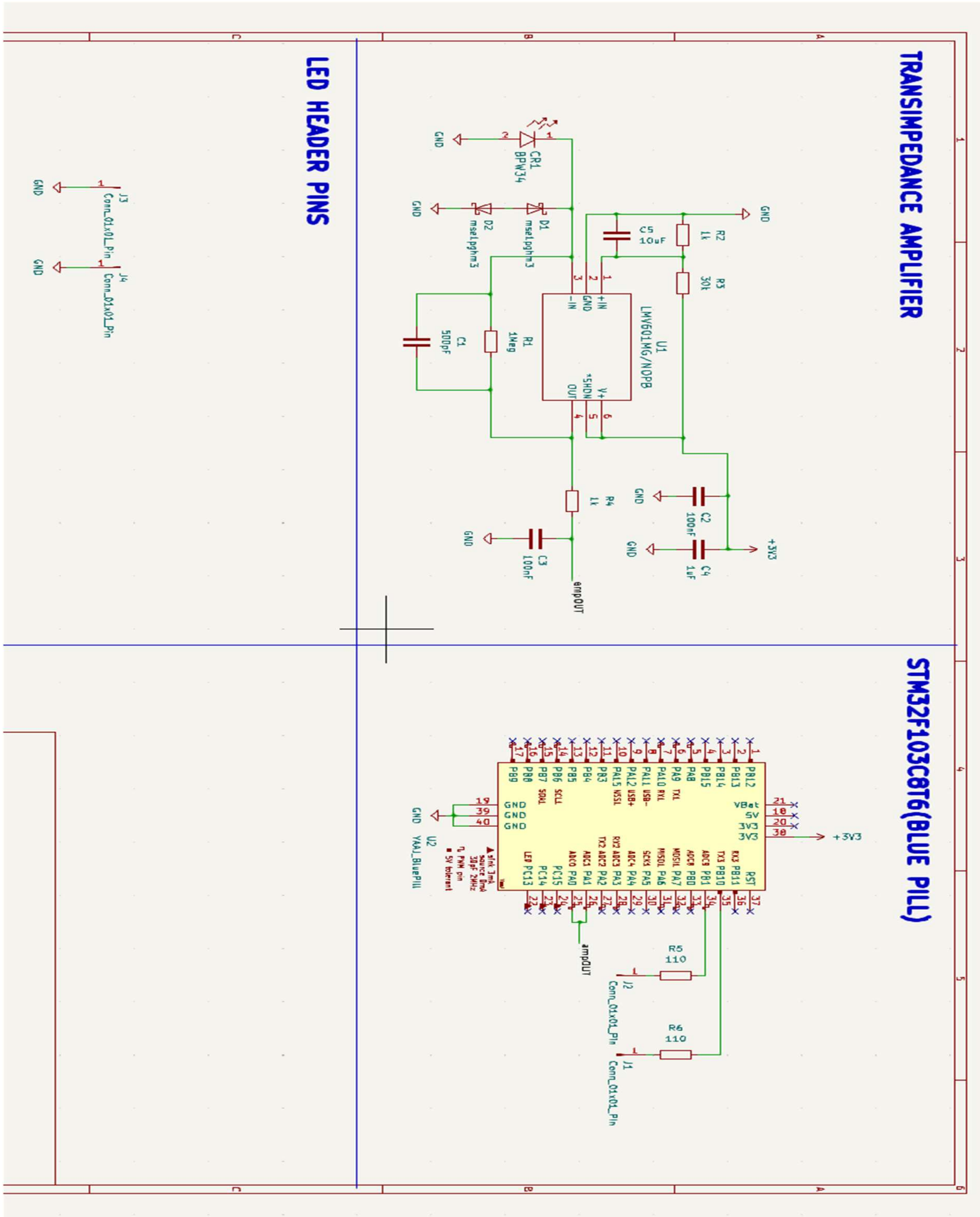


Figure 7. PCB Schematics

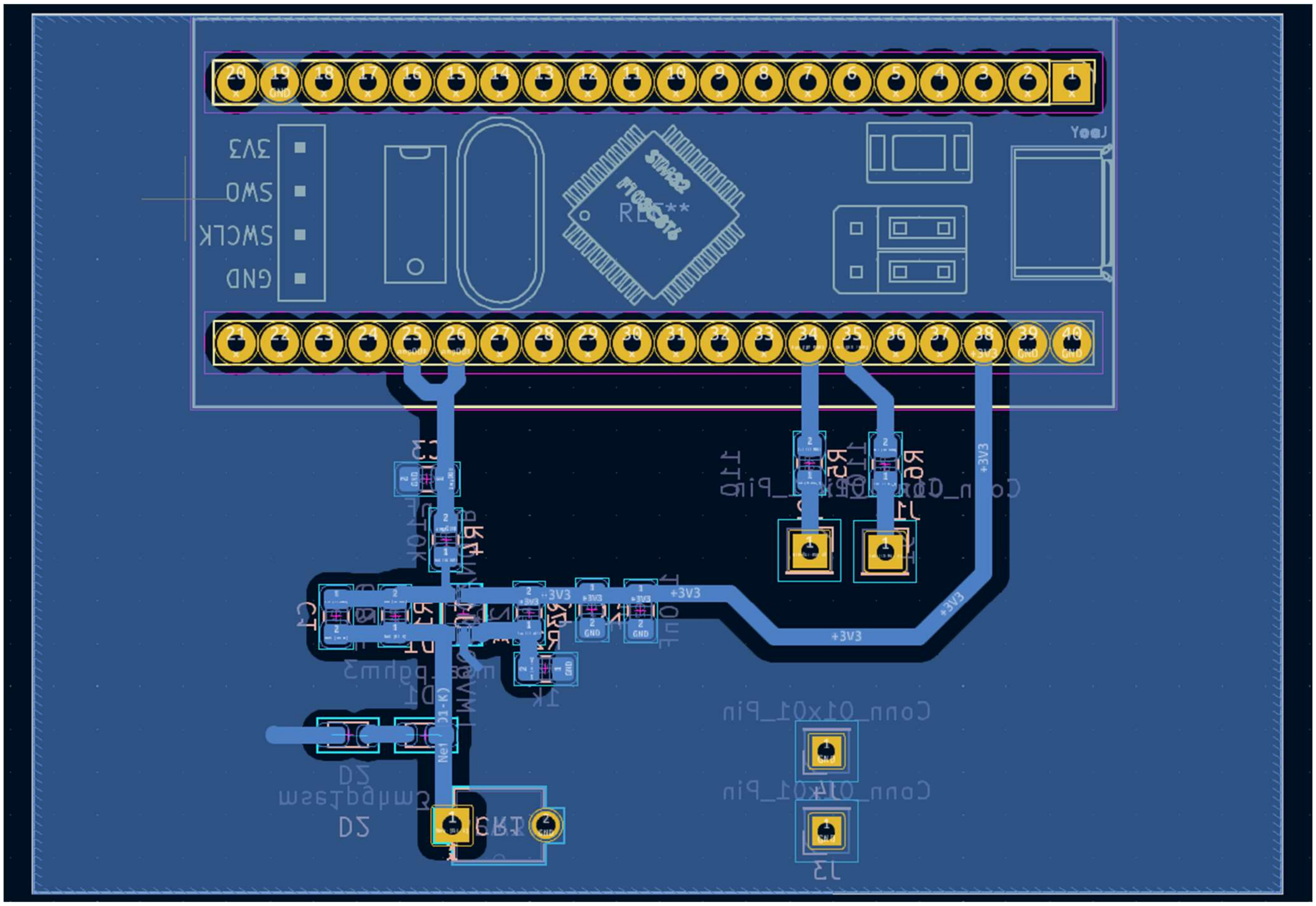


Figure 8. PCB Board

After successfully printing the PCB, I started to work on software and during tests, I changed some components for example feedback resistor and capacitance, led resistors, filter resistors and so on.

3.1.4 BOM LIST

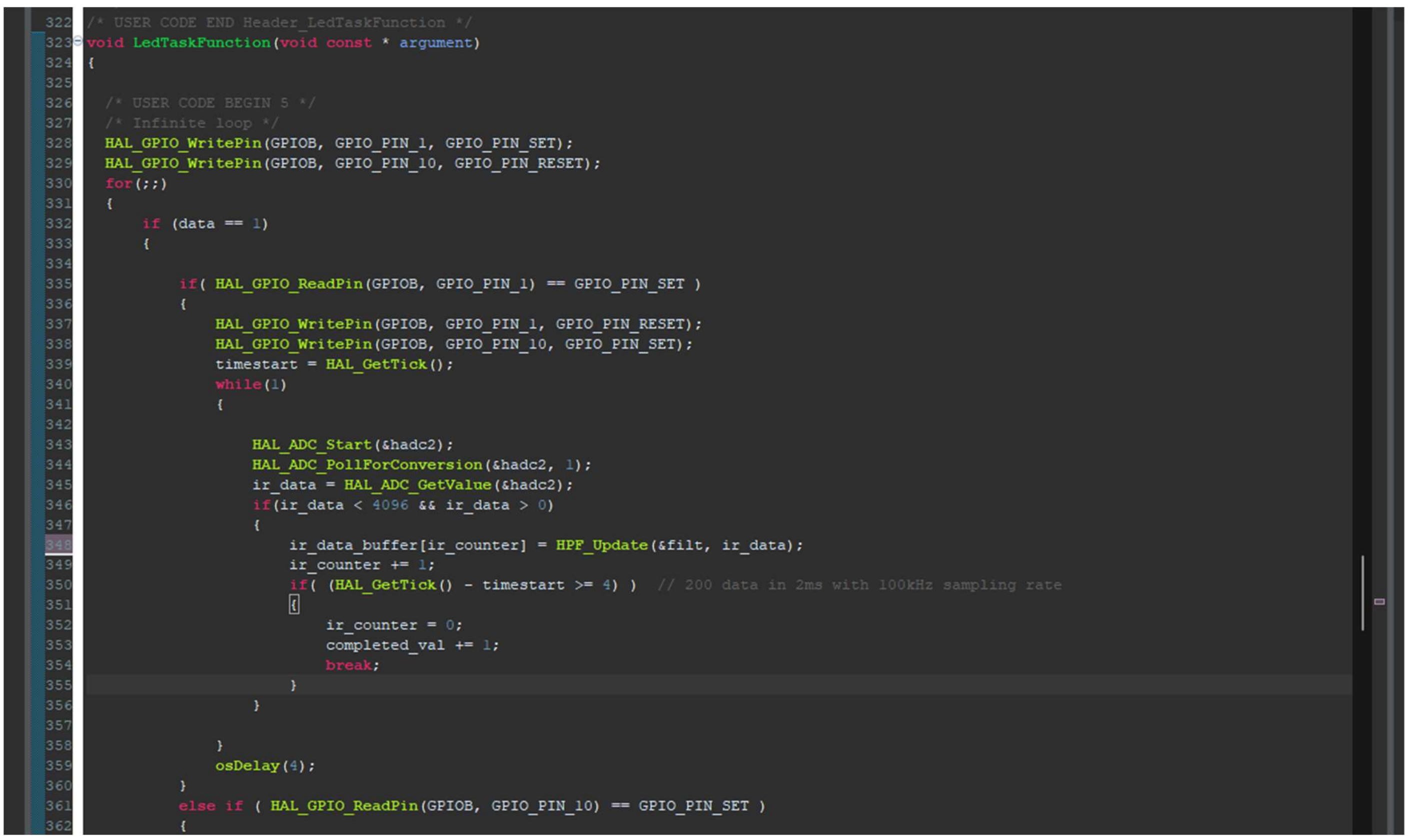
Product Description	Unit	Price(Unit)
LMV601MGX/NOPB - OPAMP	1	6.66883 ₺ + KDV
BPW34 - PHOTODIODE	1	20.28262 ₺ + KDV
3004R1D-ALS - RED LED	1	0.38013 ₺ + KDV
LTE-4206 – IR LED	1	0.33293 ₺ + KDV
PASSIVE COMPONENTS	10	0.001 ₺ + KDV
STM32F103C8T6 Development Board	1	77.58 ₺ + KDV
TOTAL		105.25398 ₺ + KDV

3.2 SOFTWARE DEVELOPMENT PROCESS

3.2.1 FREERTOS BASED PROGRAMMING

Since I need milliseconds resolution for led toggling, I decided to use FreeRTOS system because there is a system tick which ticks every 1 ms and we can set tasks with proirities such that we can arrange a healthy system. For this project I defined 3 tasks. 2 tasks, which are LedTaskFunction and ADCTaskFunction have above normal priorities and CommTaskFunc has normal priority. High-priority levels give priority to these functions such that if they are not completed or they want to begin, they begin without completing the low priority task.

Below in Figure 9, Figure 10, Figure 11 and Figure 12, I showed the last version of the functions because as I said this method did not work properly. Therefore I will shortly describe the function duties without going through too deep and I will tell what is wrong with these codes.



```
/* USER CODE END Header_LedTaskFunction */
void LedTaskFunction(void const * argument)
{

  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
  for(;;)
  {
      if (data == 1)
      {

          if( HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1) == GPIO_PIN_SET )
          {
              HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
              timestart = HAL_GetTick();
              while(1)
              {

                  HAL_ADC_Start(&hadc2);
                  HAL_ADC_PollForConversion(&hadc2, 1);
                  ir_data = HAL_ADC_GetValue(&hadc2);
                  if(ir_data < 4096 && ir_data > 0)
                  {
                      ir_data_buffer[ir_counter] = HPF_Update(&filt, ir_data);
                      ir_counter += 1;
                      if( (HAL_GetTick() - timestart >= 4) )  // 200 data in 2ms with 100kHz sampling rate
                      {
                          ir_counter = 0;
                          completed_val += 1;
                          break;
                      }
                  }

              }
              osDelay(4);
          }
          else if ( HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_10) == GPIO_PIN_SET )
          {
```

Figure 9. LedTaskFunction part 1

```
361            else if ( HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_10) == GPIO_PIN_SET )
362            {
363                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
364                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
365                timestart = HAL_GetTick();
366                while(1)
367                {
368
369                        HAL_ADC_Start(&hadc2);
370                        HAL_ADC_PollForConversion(&hadc2, 1);
371                        red_data = HAL_ADC_GetValue(&hadc2);
372                        if(red_data < 4096 && red_data > 0)
373                        {
374                            //hpf_red_out =  HPF_Update(&filt, red_data);
375                            red_data_buffer[red_counter] = red_data;
376                            red_counter += 1;
377                            if((HAL_GetTick() - timestart >= 4) )
378                            {
379                                red_counter = 0;
380                                completed_val += 1;
381                                break;
382                            }
383                        }
384                }
385                osDelay(4);
386            }
387        }
388        else
389        {
390            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
391            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
392            osDelay(4);
393        }
394    }
395    /* USER CODE END 5 */
```

Figure 10. LedTaskFunction part 2

In the LedTaskFunction function, I toggle the LEDs every 4 ms and I read ADC in the while function. In the beginning, it sets one pin and goes through the task infinite loop. In this loop, first, it checks if the start flag, which is defined as data here, is true or false. If the user sends a start message to the stm32 via computer then data becomes one DAQ starts. If the red LED pin out in the MCU is activated then it enters first if and it starts to acquire ADC values in the while loop. In the while loop, at line 346, it checks if the read ADC value is acceptable or not. If it is acceptable, it applies a high pass filter function. Then at the second if it checks time such that if the time is bigger and/or equal than then 4ms, then it breaks to loop and it does the same thing in the else if part for the IR led.

In the ADCTaskFunc function, first, I check if both red_data_buffer and ir_data_buffer are filled or not by using a flag. Then if both of them are filled I calculated the R value although the function name is spo2_calc. Every time both red and ir led buffers are filled up, I calculate the R-value and increase a counter such that if it becomes 20, then it takes the average of the 20 R-value and sends via CDC_Transmit_FS function to the computer and clean used variables and buffers.

```c
void ADCTaskFunc(void const * argument)
{
  /* USER CODE BEGIN ADCTaskFunc */
  /* Infinite loop */
  for(;;)
  {
      if(  completed_val == 2 )
      {

          spo2_buffer[spo2_counter] = spo2_calc(red_data_buffer, 130, ir_data_buffer, 130);
          spo2_counter += 1;
          completed_val = 0;

          if(spo2_counter >= 20)
          {
              for(int i = 0; i < 20; i++)
              {
                  spo2_final_val += spo2_buffer[i];
              }
              spo2_final_val = spo2_final_val/20;
              sprintf(msg, "%d\n", (int)spo2_final_val);
              CDC_Transmit_FS( (uint8_t *) msg, strlen(msg) );
              msg[0] = '\0';
              remove_element(spo2_buffer, 20);
              remove_element(red_data_buffer, 130);
              remove_element(ir_data_buffer, 130);
              spo2_counter = 0;
              spo2_final_val = 0.0;
          }
      }
   osDelay(10);
   }
  /* USER CODE END ADCTaskFunc */
}
```

Figure 11. ADCTaskFunc

The last task, which has low priority, in the main is CommTaskFunc which checks every 15ms if the user sends a message to start the whole process. It is shown in Figure 12.

```c
*/
/* USER CODE END Header_CommTaskFunc */
void CommTaskFunc(void const * argument)
{
  /* USER CODE BEGIN CommTaskFunc */
  /* Infinite loop */
  for(;;)
  {
      for(int i = 0; i < 64; i++)
      {
          if (received_data_before[i] != received_data[i])
          {
              buffer_check++;
          }

          if (buffer_check != 0)
          {
              if (data == 1)
                  data = 0;
              else
                  data = 1;

              buffer_check = 0;
          }
      }
    osDelay(15);
  }
  /* USER CODE END CommTaskFunc */
}
```

Figure 12. CommTaskFunc

In the different file, which is called Spo2Calculation.c, I created some functions for R calculation. In the spo2_calc function, I used the equation which is given in the (REF). It is another way to calculate SpO_2 saturation level of the blood. The code is given in Figure 13.Some of the basic functions, which are min, max and average calculation functions are given in Figure 14.

```c

float spo2_calc(float red_array[], int red_array_len, float ir_array[], int ir_array_len)
{
    float max_red, max_ir, min_red, min_ir, avg_red, avg_ir;
    float R_val = 0;

    max_red = max_calc(red_array, red_array_len) + 4096;
    min_red = min_calc(red_array, red_array_len) + 4096;
    max_ir = max_calc(ir_array, ir_array_len) + 4096;
    min_ir = min_calc(ir_array, ir_array_len) + 4096;
    //avg_red = avg_calc(red_array, red_array_len);
    //avg_ir = avg_calc(ir_array, ir_array_len);

    R_val = (log(max_red)/ log(min_red)) / (log(max_ir)/ log(min_ir));

    return R_val;
}

void remove_element(float array[], int array_length)
{
   for(int i = 0; i < array_length ; i++) array[i] = 0;
}

```

Figure 13. R calculation function

*main.c HighPassFilter.h HighPassFilter.c *Spo2Calculation.c

```c
#include "Spo2Calculation.h"

float max_calc(float array[], int array_len)
{
    float max = array[0];

    for(int i = 0; i < array_len; i++)
    {
        if(array[i] > max)
        {
            max = array[i];
        }
    }

    return max;
}

float min_calc(float array[], int array_len)
{
    float min = array[0];

    for(int i = 0; i < array_len; i++)
    {

        if(array[i] < min)
        {
            min = array[i];
        }
    }

    return min;
}

float avg_calc(float array[], int array_len)
{
    float sum = 0;

    for(int i = 0; i < array_len; i++)
    {

        sum += array[i];
    }

    return sum/(array_len);
}
```

Figure 14. Basic functions

```c
#ifndef HighPassFilter_H
#define HighPassFilter_H

typedef struct{

    float alpha;
    float inp;
    float out;
}HPF;

void HPF_Init(HPF *filt, float alpha);
void HPF_SetAlpha(HPF *filt, float alpha);
float HPF_Update(HPF *filt, float inp);

#endif
```

Figure 15. Header file of high-pass filter

```c
#include "HighPassFilter.h"

void HPF_Init(HPF *filt, float alpha)
{
    filt->out = 0.0f;
}

void HPF_SetAlpha(HPF *filt, float alpha)
{
    if (alpha > 1.0f)
    {
        alpha = 1.0f;
    }else if (alpha < 0.0f)
    {
        alpha = 0.0f;
    }

    filt->alpha = alpha;
}

float HPF_Update(HPF *filt, float inp)
{
    filt->out = filt->out * filt->alpha + inp - filt->inp;
    filt->inp = inp;
    return filt->out;
}
```

Figure 16. High-pass IIR Filter code implementation

Last but not least , in Figure 15, I shared the high pass IIR filter header file and in Figure 16 its implementation in the C code. It has 3 functions. First one initialize the TypeDef object. The second one can be used to set the Alpha value which determines the cut-off frequency of the filter. The last one simply the takes current output and multiply by alpha, then adds the current input value and subtracts before the input value. In the second line of the HPF_Update function. It equates before input to current input such that new input comes it becomes old input and then it returns new output value. This discrete function was shown in the calculation section for the electronic circuit design section.

In this program, since the calculations are very slow, given 4ms calculations are not finished and since there are a lot of calculations and check values the ADC sampling rate is unstable such Nyquist sampling criteria are disobeyed. Therefore data is not reliable and I could not optimize it further.

3.2.2 ADC DMA PROGRAMMING

Since I want always the same sampling rate when I am acquiring data. I decided to use DMA with interrupt. I set the ADC interrupt function such that its priority is high so whenever data should be acquired, it interrupts what the program does, and starts the acquisition until the ADC buffer is filled. In this program, the clock of ADC was set such that every 4ms it acquires 200 samples which corresponds to 100 kHz sampling frequency. Therefore whatever happens Nyquist's criterion is obeyed.

But since I use buffers and these buffers change frequently, I defined a flag such that ADC interruption does not start before necessary buffers are used although when it starts, its sampling frequency does not change. Our first import function which depends on the system tick time, that is equal to 1ms, is called toggleLed () function. This function is shown below in Figure 17.

```
/* USER CODE BEGIN 4 */
void toggleLed()
{
    if(adc_flag == 0 && toggle_flag >= 4)
    {
        toggle_flag = 0;
        if( HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1) == GPIO_PIN_SET )
        {
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        }else
        {
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
        }
        HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adcBuffer, ADC_BUFFER_SIZE);

    }

    toggle_flag += 1;

}
```

Figure 17. Inside of the toggleLed Function

As you can see this function has 2 conditions. First one is if MCU is busy with other calculations, I set adc_flag true such that, ADC does not interrupt the calculations and meanwhile led pins are not toggled. Another condition is toggle_flag which counts up to 4 ms since this function is called SystemTick every 1ms. If 4 ms or more is passed then we can toggle the LEDs and start the ADC function.

In Figure 18, one can see after ADC is done function. In this function, whenever ADC acquires all the data which is needed, ADC interrupts call this function.

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{

    if ( HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1) == GPIO_PIN_SET )
    {

        for(int i = 0; i < ADC_BUFFER_SIZE; i++)
        {
            //red_buffer[i] = HPF_Update(&filt, adcBuffer[i]);
            red_buffer[i] = adcBuffer[i];
        }

        red_flag = 1;
    }else
    {

        for(int i = 0; i < ADC_BUFFER_SIZE; i++)
        {
            //ir_buffer[i] = HPF_Update(&filt, adcBuffer[i]);
            ir_buffer[i] = adcBuffer[i];
        }
        ir_flag = 1;
    }

}
/* USER CODE END 4 */
```

Figure 18. Inside of the toggleLed Function

In this function red_buffer is equated to ADC buffer if the red led pin is set, otherwise, ir_buffer is equated and their flags are equated 1 which we use in an infinite loop which is given below in Figure 19.

In this loop, if both red_buffer and ir_buffer are filled then we calculate the R value and put it into another buffer which is called spo2_buffer. This buffer holds calculated R values up to its length which is shown in Figure 20. Whenever it is filled again we calculate its average, but since the sprintf function only sends integer value and spo2_final_val is generally between 0 and 1, to not lose data, I do some additional calculation to see meaningful float value. This is why sometimes R values were negative during the project demo session. In this code, as you requested I did not use a high pass filter.

Finally, in Figure 21, you can see the spo2_calc function which relates to the formula in the (REF).

```c
/* USER CODE BEGIN WHILE */
while (1)
{
    if(red_flag == 1 && ir_flag == 1)
    {

        spo2_buffer[spo2_counter] = spo2_calc(red_buffer, ADC_BUFFER_SIZE, ir_buffer, ADC_BUFFER_SIZE);

        adc_flag = 0;
        spo2_counter += 1;
        red_flag = 0;
        ir_flag = 0;
    }

    if(spo2_counter == SPO2_BUFFER_SIZE)
        {
          for(int i = 0; i < SPO2_BUFFER_SIZE; i++)
          {
              spo2_final_val += spo2_buffer[i];
          }

          spo2_final_val = ((spo2_final_val/SPO2_BUFFER_SIZE)*100 - 90)*100;
          sprintf(msg, "%d\n", (int)spo2_final_val);
          CDC_Transmit_FS( (uint8_t *) msg, strlen(msg) );  // Every 1000 * 1ms = 100ms, we send spo2 value
          msg[0] = '\0';
          remove_element(spo2_buffer, SPO2_BUFFER_SIZE);
          spo2_counter = 0;
          spo2_final_val = 0;
        }
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Figure 19. Infinite loop in the main function

```c
/* Private define -------------------------------------
/* USER CODE BEGIN PD */
#define ADC_BUFFER_SIZE 200
#define SPO2_BUFFER_SIZE 50
#define MSG_BUFFER_SIZE 64
/* USER CODE END PD */
```

Figure 20. Buffer size definitions

```c
#include "Spo2Calculation.h"
#include <stdint.h>
#include <math.h>

extern volatile uint8_t adc_flag;

float spo2_calc(uint16_t red_array[], uint8_t red_array_len, uint16_t ir_array[], uint8_t ir_array_len)
{
    adc_flag = 1;

    int16_t max_red, max_ir, min_red, min_ir;
    float avg_red, avg_ir, R_val;
    max_red = max_calc(red_array, red_array_len);
    min_red = min_calc(red_array, red_array_len);
    max_ir = max_calc(ir_array, ir_array_len);
    min_ir = min_calc(ir_array, ir_array_len);
    avg_red = avg_calc(red_array, red_array_len);
    avg_ir = avg_calc(ir_array, ir_array_len);

    R_val = ((max_red - min_red)/ avg_red) / ((max_ir - min_ir)/ avg_ir);

    return R_val;
}
```

Figure 20. spo2_calc function with adc_flag

As you can see, before these calculations do not finish, ADC does not start DAQ because otherwise values in the buffers will change and we lose data.

3.2.3 DATA ACQUISITION PROGRAMS

3.2.3.1 PYTHON GUI

Before I used this GUI at the beginning of the project to observe live data of RED and IR ADC data. Unfortunately Python minimum thread is max 15ms, this program is useless and I really forgot the show that during the project demo session. As you can see, by using this GUI, you can set the baud rate and choose the COM port whatever the device port. It has a click function such that whenever you click if the COM port list is changed it shows new COM ports. When you click starts, it sends a start string to MCU to start its process and it acquires data and plots it, but it is useful only for small acquisition processes. You can see the program in Figure 21.

3.2.3.2 C++ Script

After observing that how Python is lazy, I decided to acquire data with C++. I first tried to few libraries for live data plots and eventually, I ended up with Matplotlib anyway. I somehow added the Matplotlib library to my C++ code, but unfortunately, I saw that the problem was the plotting itself. It is not useful for such high frequencies. Therefore eventually every calculation was made on MCU and I just used C++ code to show low frequency data stream in the demo session. You can find the code in the GitHub repository(REF 7).

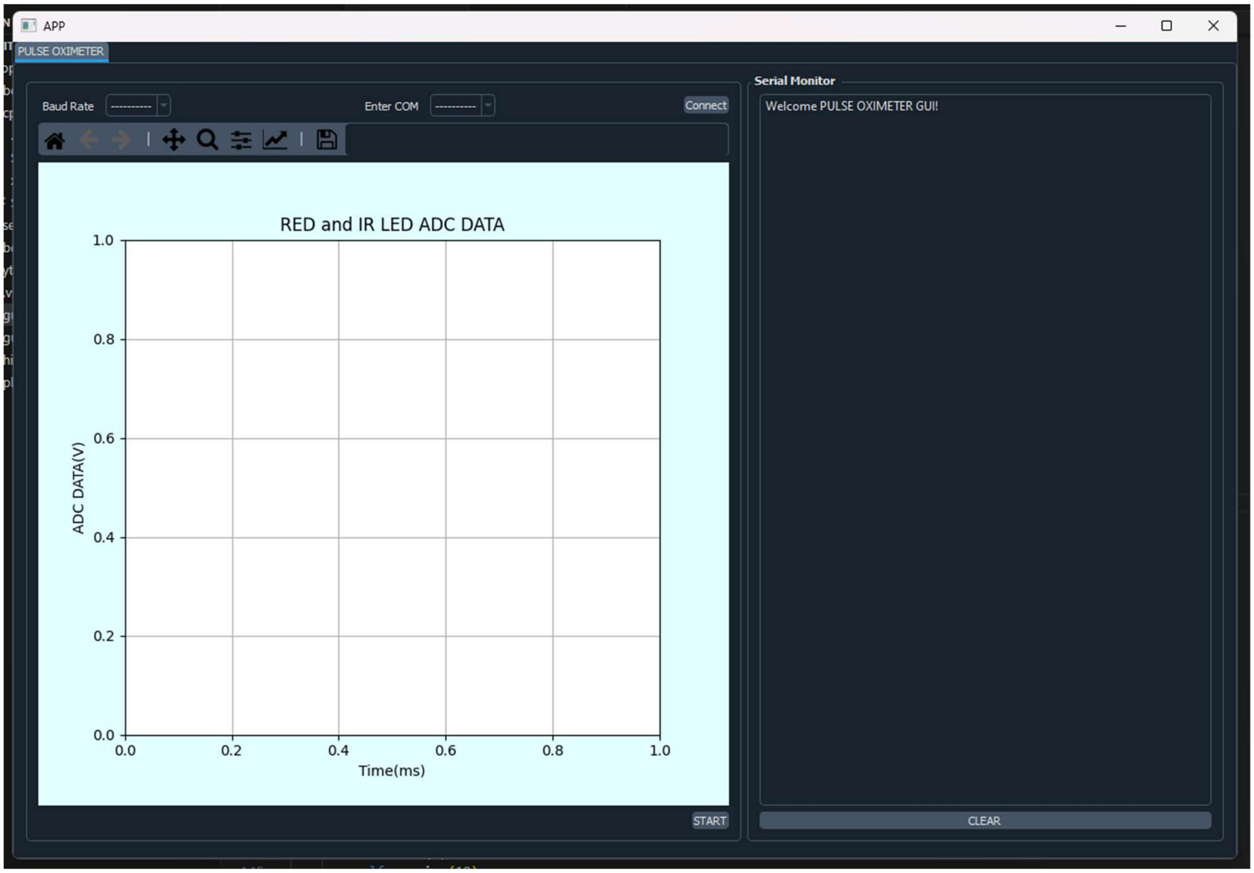


Figure 21. Python GUI for real-time DAQ

5. CONCLUSION

My actual aim was to acquire data and process it in the computer without doing any calculations in the MCU because of the problems that I saw on FreeRTOS, but it seems that I was stunned by the programming side of the project much more than the hardware side. Although as we observed at the last acquisition, there is a very small trend in the data, it is a sign that the hardware of the project can be made reliable by using a bigger gain resistor. Unfortunately, I did not have any SMD component with me so that I can change the gain resistor to show you, but I believe you will see that from the calculations and simulations sides, the hardware side of the project is also reliable.

6. References

1) https://avesis.deu.edu.tr/dosya?id=a1c2646b-6c6a-4bbd-865f-fa5fafb34729
2) https://www.sciencedirect.com/science/article/pii/S095461111300053X
3) https://diposit.ub.edu/dspace/bitstream/2445/200772/1/ALDANA%20LONDO%C3%91O%20KATERIN_7934146.pdf

4) https://www.researchgate.net/publication/272325643_The_theory_and_application_of_pulse_oximetry
5) https://dl.acm.org/doi/10.1145/3592307.3592319
6) https://www.ti.com/lit/ds/symlink/lmv601.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1704301246235&ref_url=https%253A%252F%252Fwww.ti.com%252Fgeneral%252Fdocs%252Fsuppproductinfo.tsp%253FdistId%253D10%2526gotoUrl%253Dhttps%253A%252F%252Fwww.ti.com%252Flit%252Fgpn%252Flmv601
7) https://github.com/ardaunal4/Pulse-Oximeter