24Ghz-RADAR for Movement Detection

Group D3 Project Documentation

Lab
Instructors: Mr. Holger Wech
Mr. Manuel Leidel

This Document describes the Movement Detection using 24GHz CW-RADAR sensor which was performed in the System Driven Hardware Design Lab as a lab group.

Author: Ananya Harish Communications 1119248
Drashti Shah Embedded Systems 1119295
Girish Tabaraddi Embedded Systems 1119236

Status: released

Revision	Date	Editor	Reason
v1.0	02/July/2023	Group D3	Basic content and formatting

1 KiCad Schematic

The necessary schematic of Movement Detection using IPM-165 radar module is designed using KiCad 7.0 open-source free software.

1.1 Overall Schematic

- Clearly separated designs with Global Labels/Flags.
- Arduino shield was added for mounting of the amplifier on the FreeSoC board.
- 2 Stage Amplifier Design.
- Extra test points for easy debugging.
- LEDs to indicate the status of State Machine.

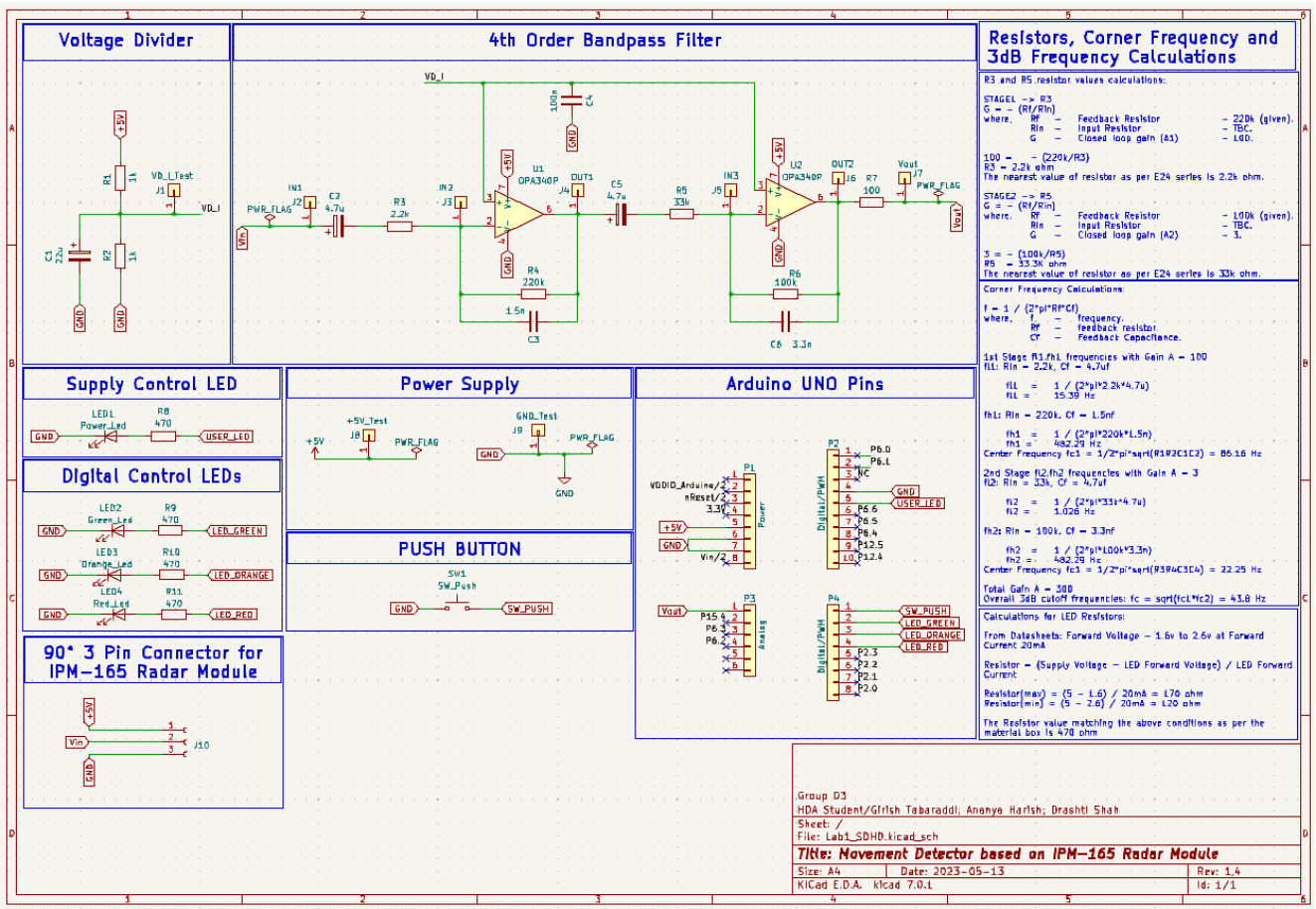


Fig.1 KiCad Schematic

1.2 Analog Circuit

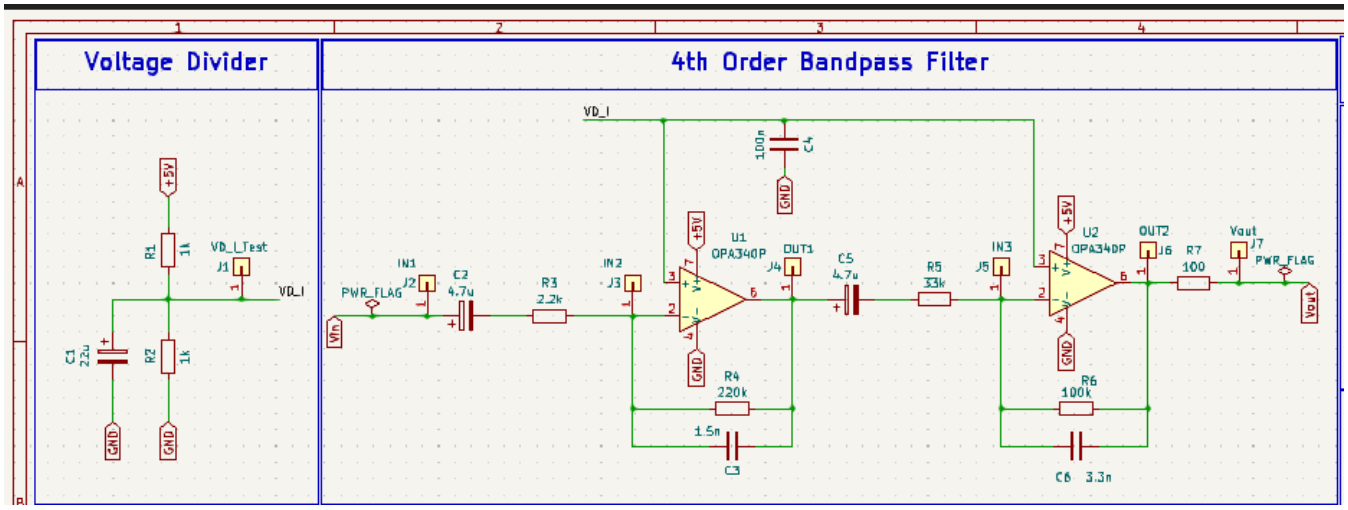


Fig 2. Voltage Divider and Bandpass Filter

1.2.1 Voltage Divider Circuit

Voltage Divider circuit is used to cut down the 5V given from the FreeSoC board to the non-inverting terminals of both the op-amps U1 & U2. The voltage is reduced to 2.5V, because the resistors selected have equal value (1kΩ each).

1.2.2 4th Order Bandpass Filter

V_{in} (output of the radar module) is fed as the input to inverting terminal of the OPA-340 U1. The voltage range of the radar sensor is -300mV to 300mV. The circuit is designed to have a total gain of 300, which is divided into gain of 100 for the first stage amplifier and a gain of 3 for the second stage of the amplifier. Each of the circuit stages has their own test points for easy debugging and testing. The output of the amplifier circuit is passed on to the 16-bit ADC of PSOC board.

```
R3 and R5 resistor values calculations:

STAGE1 -> R3
G = - (Rf/Rin)
where,    Rf   -    Feedback Resistor          - 220k (given).
          Rin  -    Input Resistor             - TBC.
          G    -    Closed loop gain (A1)      - 100.

100 =    - (220k/R3)
R3 = 2.2k ohm
The nearest value of resistor as per E24 series is 2.2k ohm.

STAGE2 -> R5
G = - (Rf/Rin)
where,    Rf   -    Feedback Resistor          - 100k (given).
          Rin  -    Input Resistor             - TBC.
          G    -    Closed loop gain (A2)      - 3.

3 = - (100k/R5)
R5  = 33.3K ohm
The nearest value of resistor as per E24 series is 33k ohm.
```

Fig 3. R3 and R5 resistor calculations.

```
Corner Frequency Calculations:

f = 1 / (2*pi*Rf*Cf)
where,    f    –    frequency.
          Rf   –    feedback resistor.
          Cf   –    Feedback Capacitance.

1st Stage fl1,fh1 frequencies with Gain A = 100
fl1: Rin = 2.2k, Cf = 4.7uf

     fl1  =    1 / (2*pi*2.2k*4.7u)
     fl1 =     15.39 Hz

fh1: Rin = 220k, Cf = 1.5nf

     fh1  =    1 / (2*pi*220k*1.5n)
     fh1 =     482.29 Hz
Center Frequency fc1 = 1/2*pi*sqrt(R1R2C1C2) = 86.16 Hz

2nd Stage fl2,fh2 frequencies with Gain A = 3
fl2: Rin = 33k, Cf = 4.7uf

     fl2  =    1 / (2*pi*33k*4.7u)
     fl2 =     1.026 Hz

fh2: Rin = 100k, Cf = 3.3nf

     fh2  =    1 / (2*pi*100k*3.3n)
     fh2 =     482.29 Hz
Center Frequency fc1 = 1/2*pi*sqrt(R3R4C3C4) = 22.25 Hz

Total Gain A = 300
Overall 3dB cutoff frequencies: fc = sqrt(fc1*fc2) = 43.8 Hz
```

Fig 4. Corner Frequency Calculations.

1.3 Digital Circuit

Push Button is used to change from initial state to sampling state in software implementation, which is explained later in the following sections.

Similarly, the LEDs are added to visually notice the change of states. The Supply control LED is added in the design to ensure that the PCB is powered up.

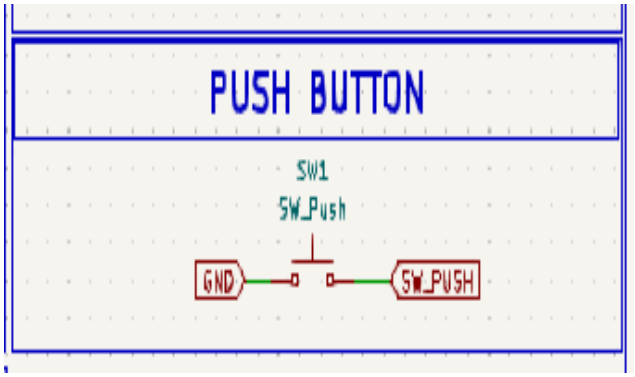


Fig 5. Push Button.

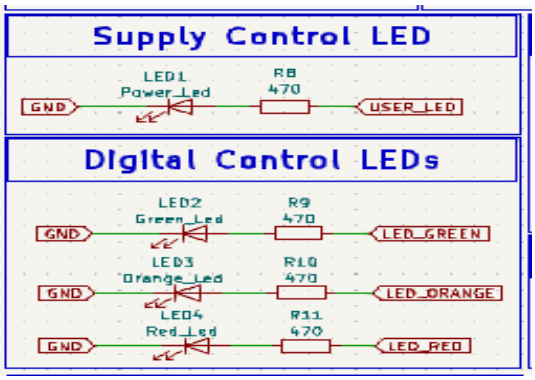


Fig 6. Digital LEDs for State-machine.

Calculations for LED Resistors:

From Datasheets: Forward Voltage – 1.6v to 2.6v at Forward Current 20mA

Resistor = (Supply Voltage – LED Forward Voltage) / LED Forward Current

Resistor(max) = (5 – 1.6) / 20mA = 170 ohm
Resistor(min) = (5 – 2.6) / 20mA = 120 ohm

The Resistor value matching the above conditions as per the material box is 470 ohm

Fig 7. LED Resistor Calculations.

1.4 Power Supply

Power flags and global naming flags ared added for convenience. Sufficient test points are added in the circuit for easy debugging and testing.

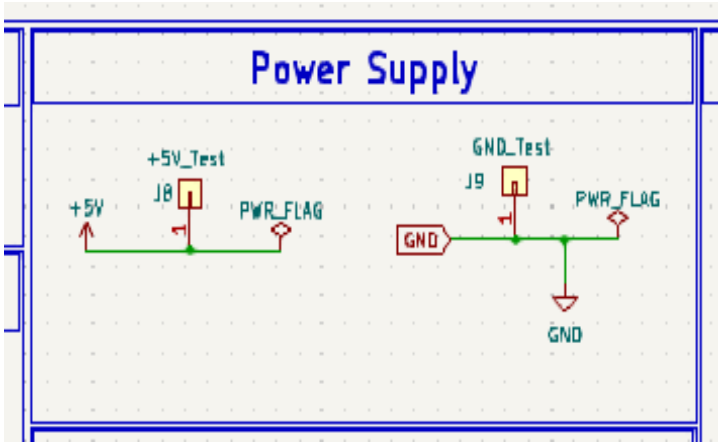


Fig 8. Power Supply Connection.

1.5 IPM-165 Radar Module

The IPM-165 is a 24GHz doppler module with an asymmetrical wide beam for detection of moving objects. Pin 1 of the sensor is used to power up the radar module, Pin 2 is used to give input to the bandpass filter and Pin 3 is connected to the ground.

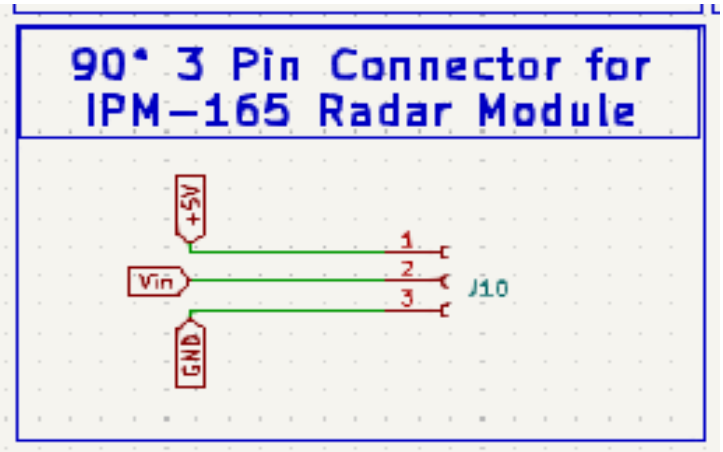


Fig 9. IPM-165 Radar Module

1.6 Interface to FreeSoC Board

To integrate our circuit to the FreeSoC board, we need the Arduino UNO shield which matches the FreeSoC pins. So, we imported the Arduino UNO shield from KiCad inbuilt library and the necessary pin mapping is provided in the following figure.

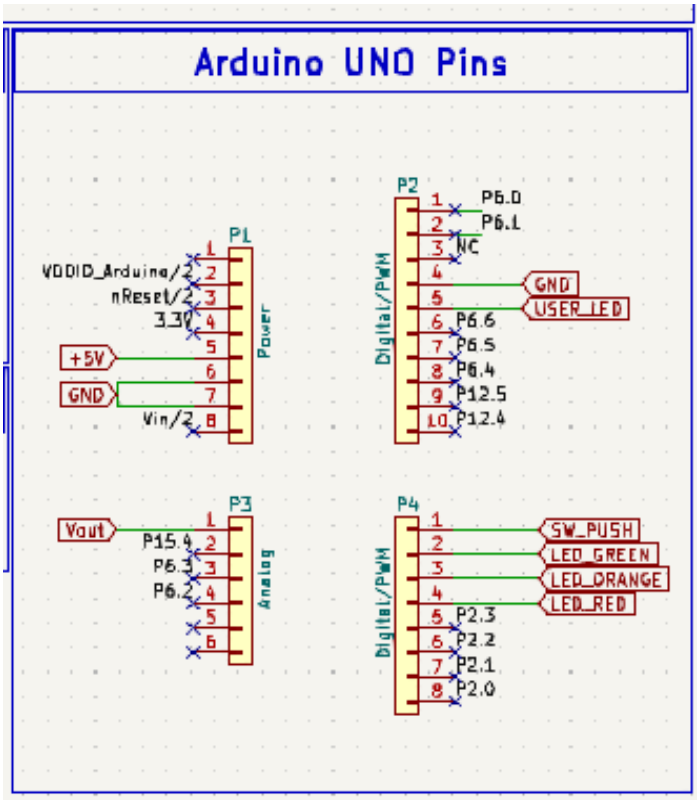


Fig 10. Arduino UNO Shield pin mapping.

2 PCB Design

2.1 About PCB

Our PCB has a thickness of 1.6mm and consists of four layers. Four layers PCB means there are 4 layers to rout electrical signals: Top Layer, Inner Layer 1, Inner Layer 2 and Bottom Layer. TOP and bottom layers are the outer layers where we are placing components and routing. Inner layer 1 and inner layer 2 are in the core, and normally used as power planes or used for signal routing. Therefore, 4-layer PCB = 3 Signal layer + GND layer or 4-layer PCB = 2 signal layers + a VCC layer + a GND layer [Reference: https://www.allpcb.com/4_layer_pcb.html].

2.2 Layer 1

On this layer all signals and the components are placed. On the top and bottom layer, ground is poured to provide insulation. Vias are added to connect the ground layers on top and bottom together.

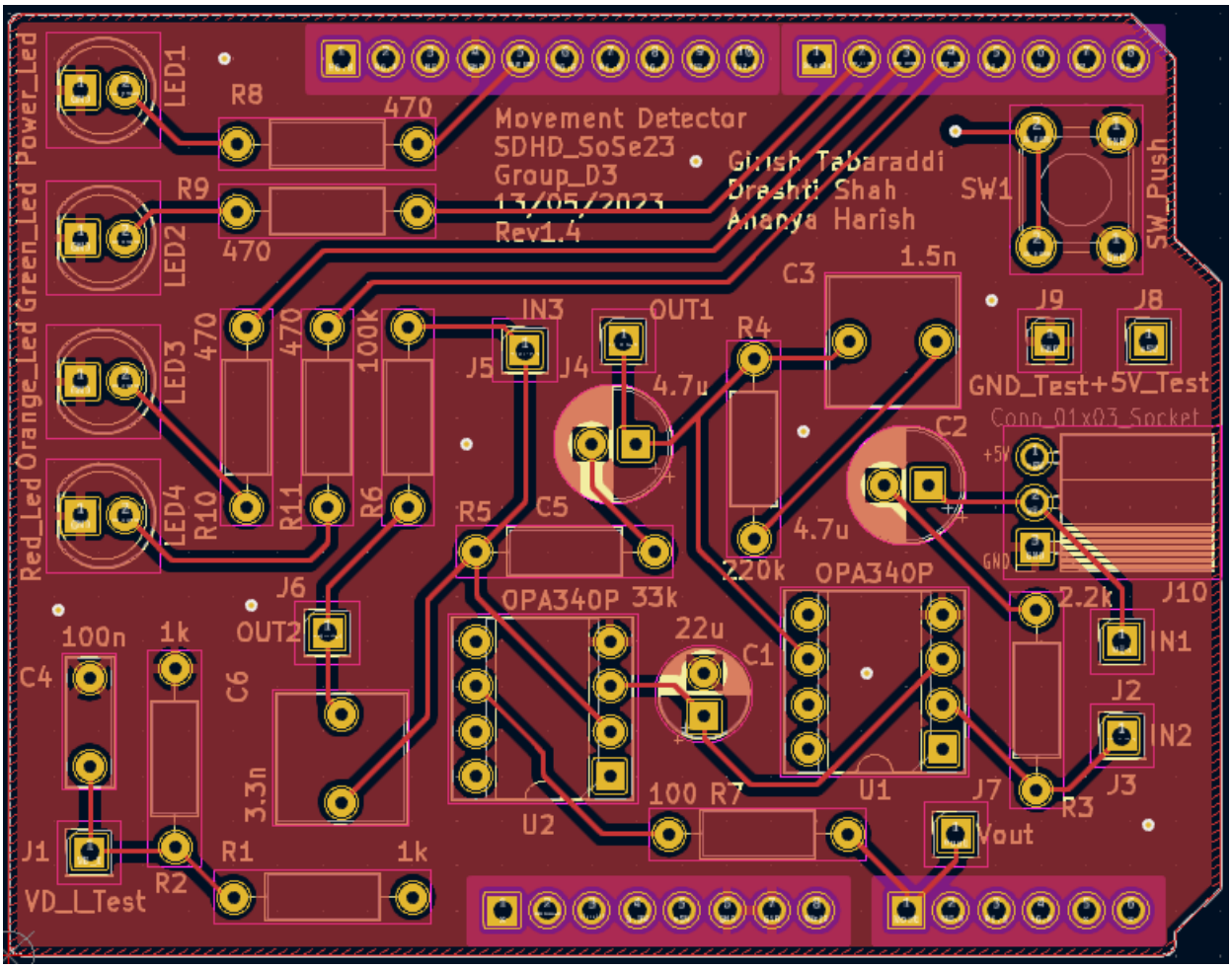


Fig 11. Front Signal Layer with routing

2.3 Layer 2

Ground is poured here.

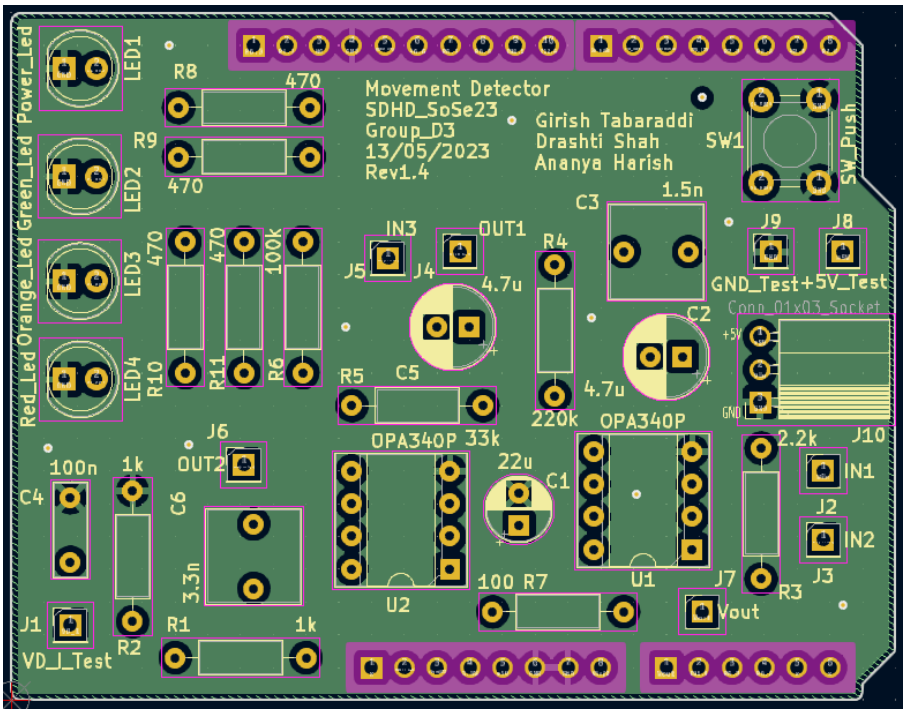


Fig 12. Ground Layer

2.4 Layer 3

Here the VCC is poured over the third layer.

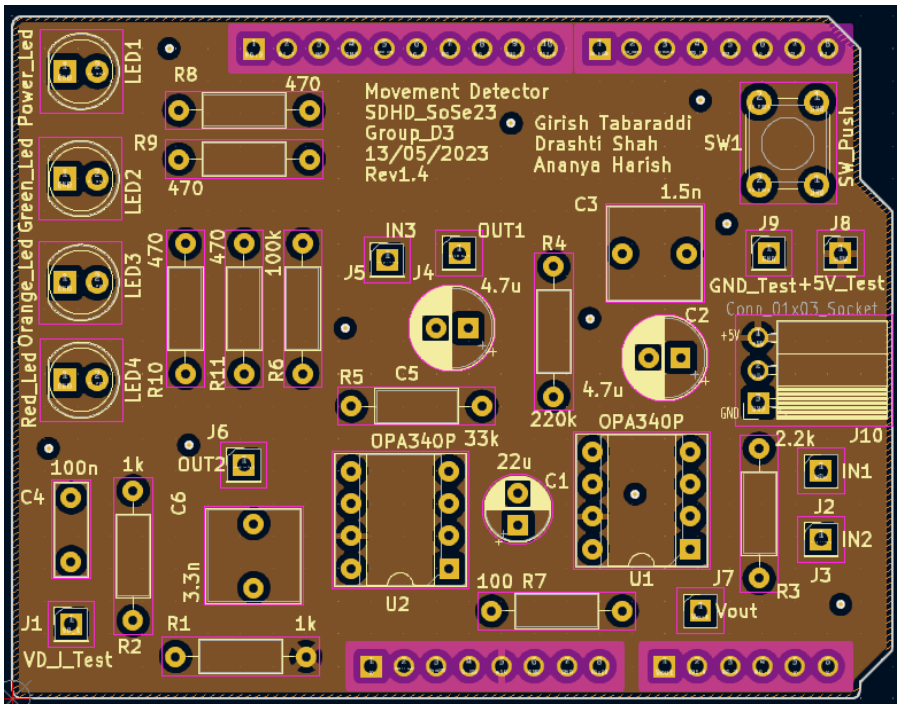


Fig 13. Power Layer

2.5 Layer 4

In order to avoid crossings of routing in first layer, the crossings are connected in the fourth layer. Vias are added to connect first and fourth layer.

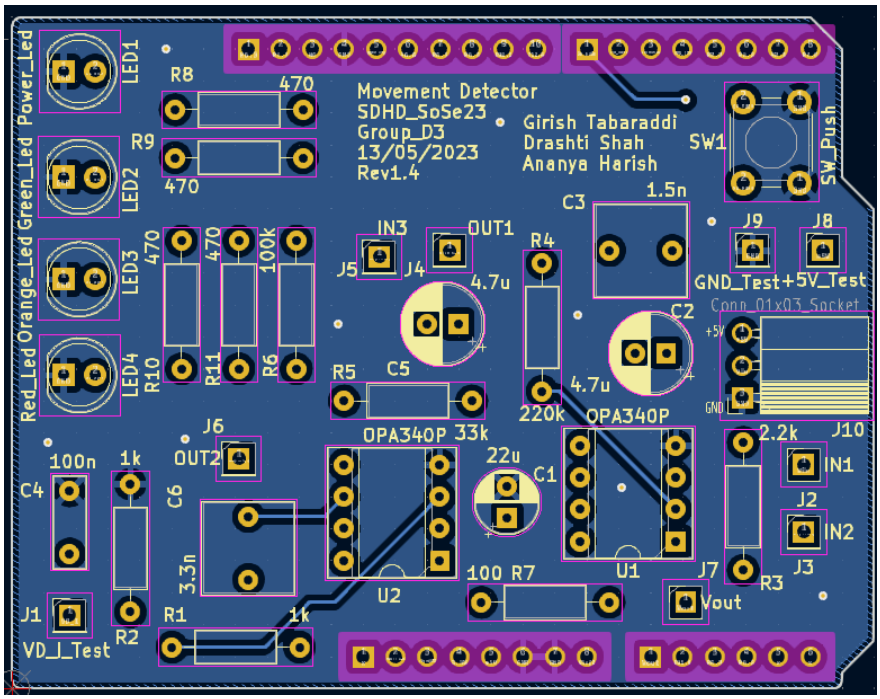


Fig 14. Bottom Signal Layer with routing

2.6 3D view of PCB

The 3D view of the designed PCB is shown in the below figure. The minimum tract length between the input of the Radar and to the input of the IC is worked out.

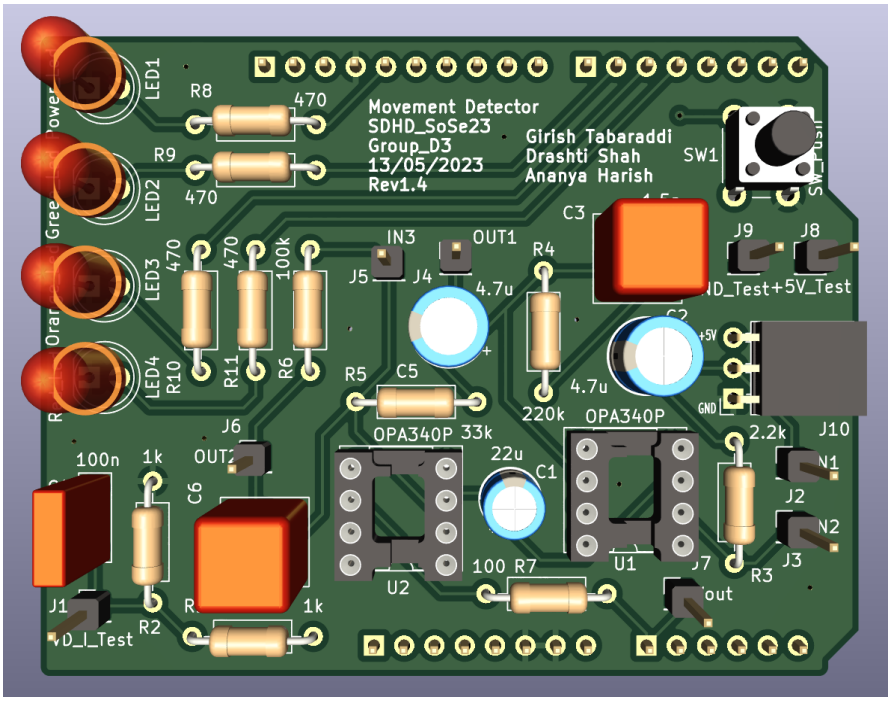


Fig 15. 3D View

3 Hardware Connections

3.1 Stage 1 circuit connection

The connections of the first stage from the PCB are verified at test point J4(OUT1). An additional voltage divider circuit is connected to the PCB externally in order to reduce the input voltage such that the output voltage remains in the given range. For testing, we considered V_{in} of 200mV which was cut down to 8mV using resistors of 2.2kΩ and 100Ω. The gain of the first stage designed is 100 and the same was verified by obtaining an output voltage of 0.8V. The connection and output are as shown below.

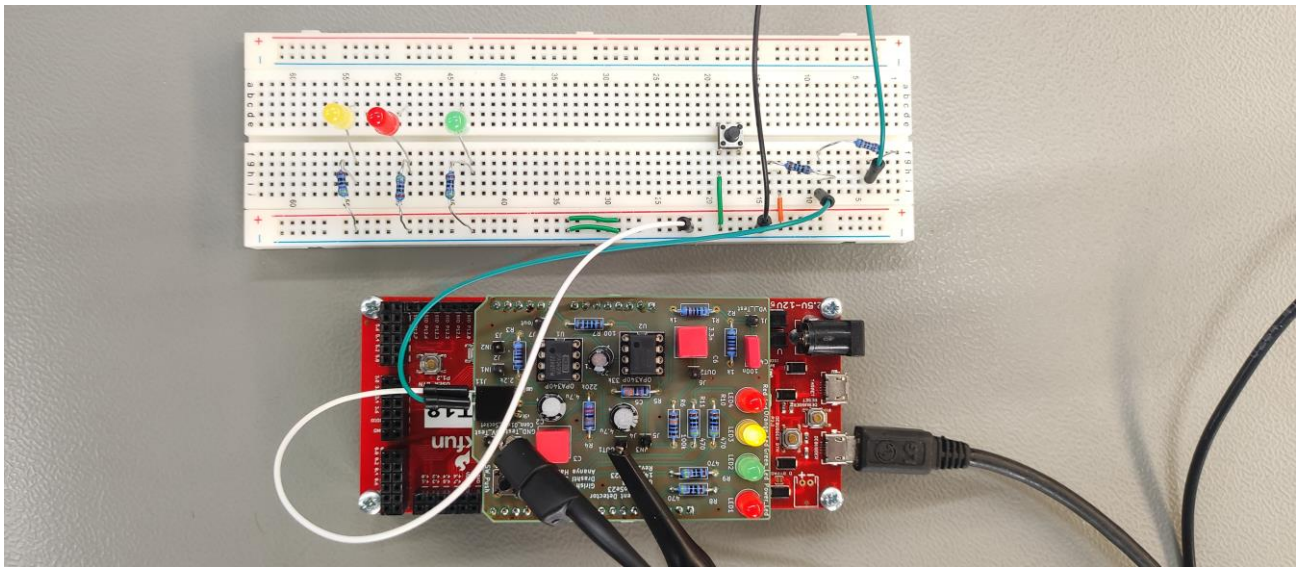


Fig 16. First stage connection.

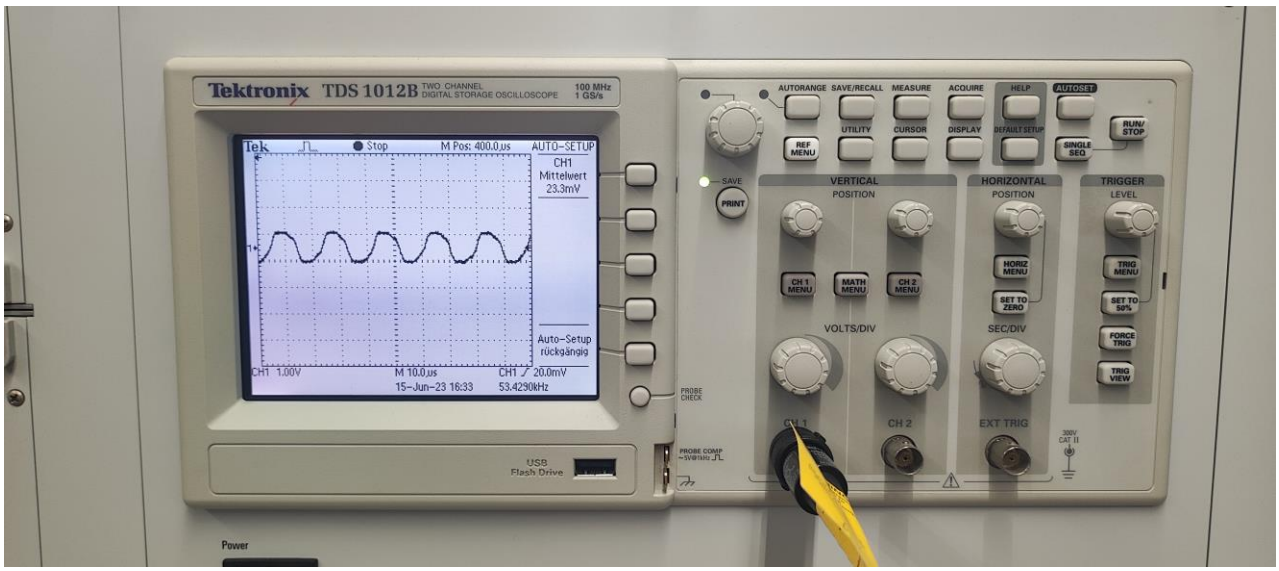


Fig 17. First stage output.

3.2 2 Stage circuit connection

The connections of the second stage from the PCB are verified at test point J7(V_{OUT}). For the V_{in} of 200mV which was cut down to 8mV using resistors of 2.2kΩ and 100Ω at first stage. For the second stage with design of gain 3, the output was verified by obtaining 2.4V output voltage. The connection and output are as shown below.

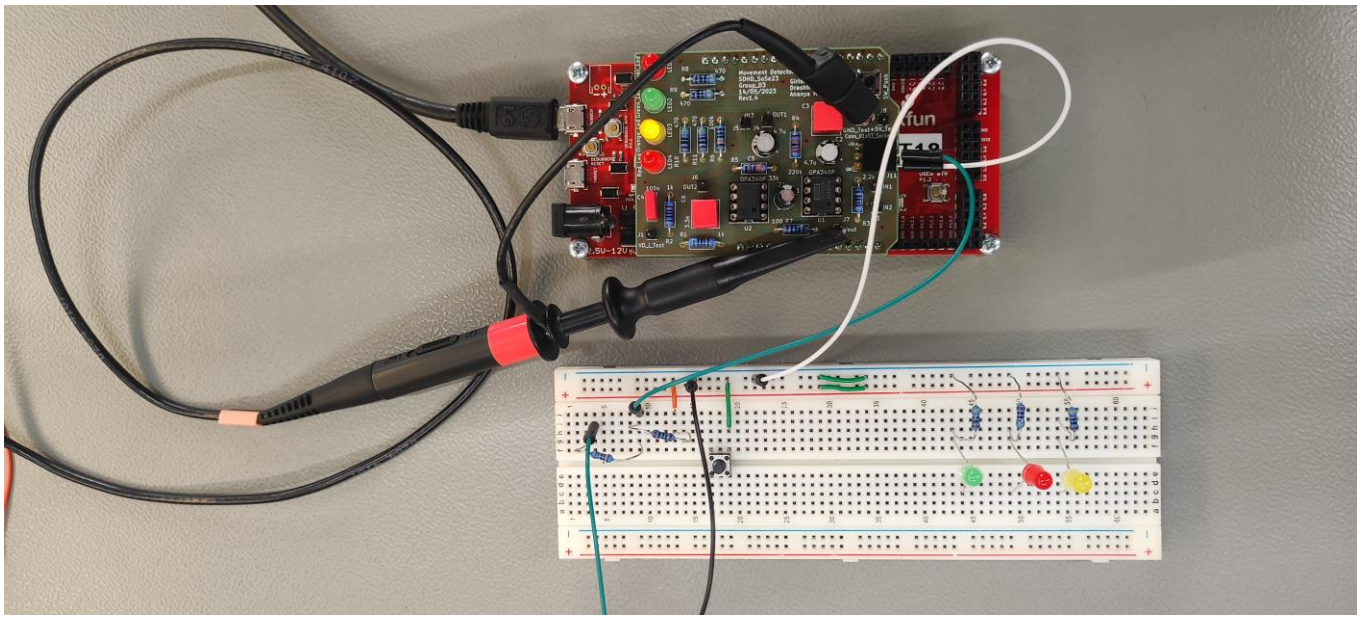


Fig 18. Second stage connection.

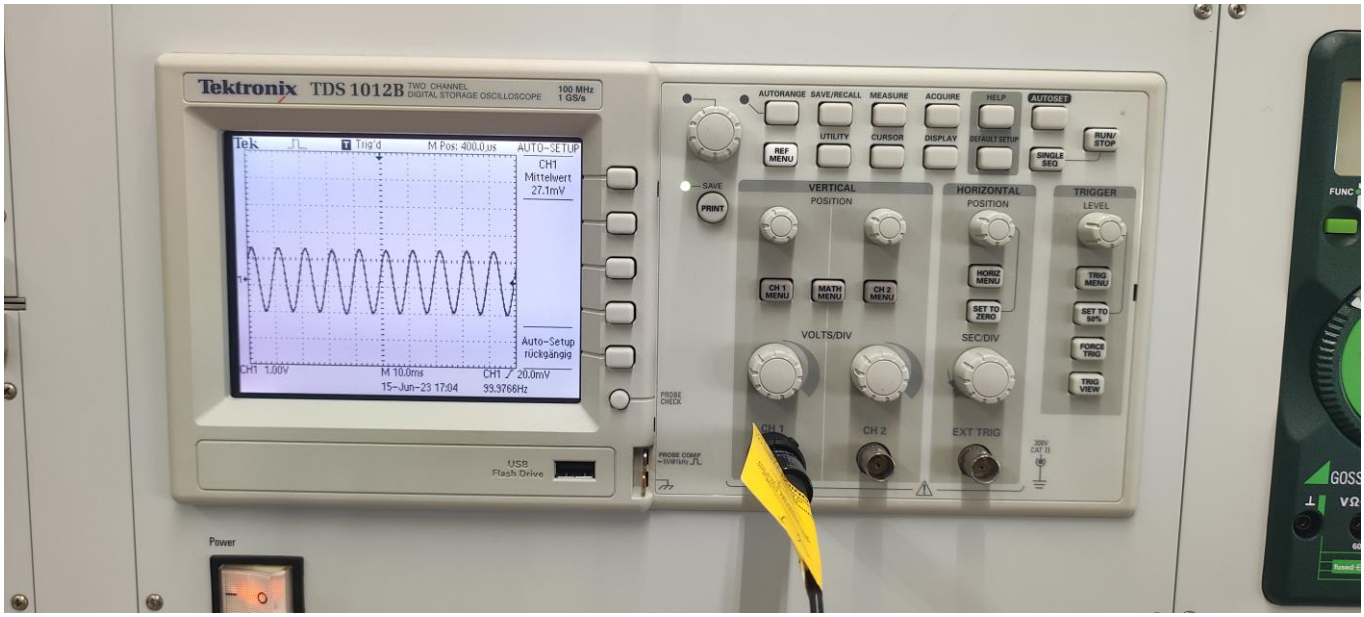


Fig 19. Second stage output.

3.3 PCB with IPM-165 Radar Module

The IPM-165 Radar module is connected to the working PCB in order to detect the movement of a moving object. From the output it can be observed that the movement is detected by change of frequency as well as amplitude of the signal.

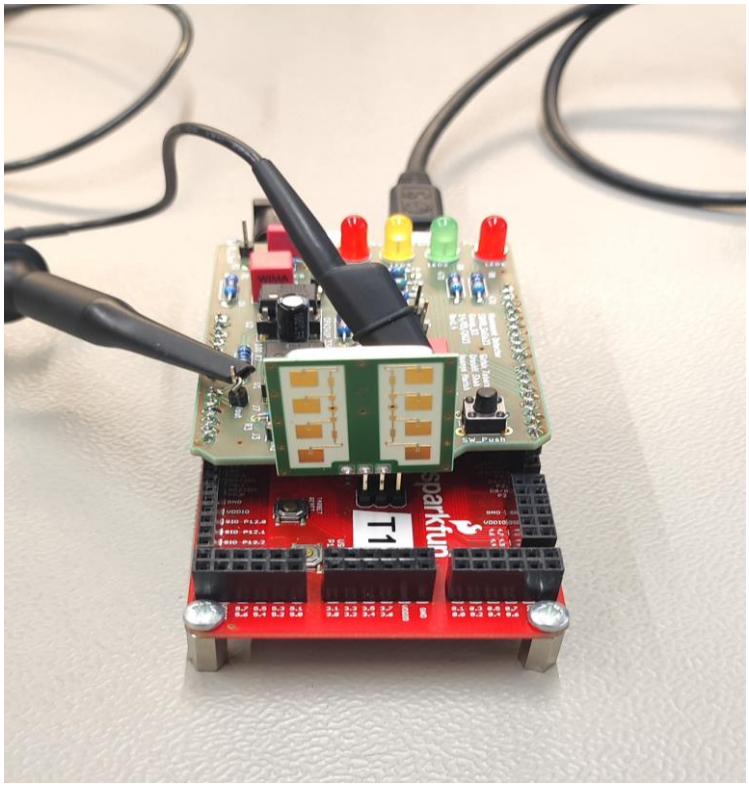


Fig 20. PCB with radar module

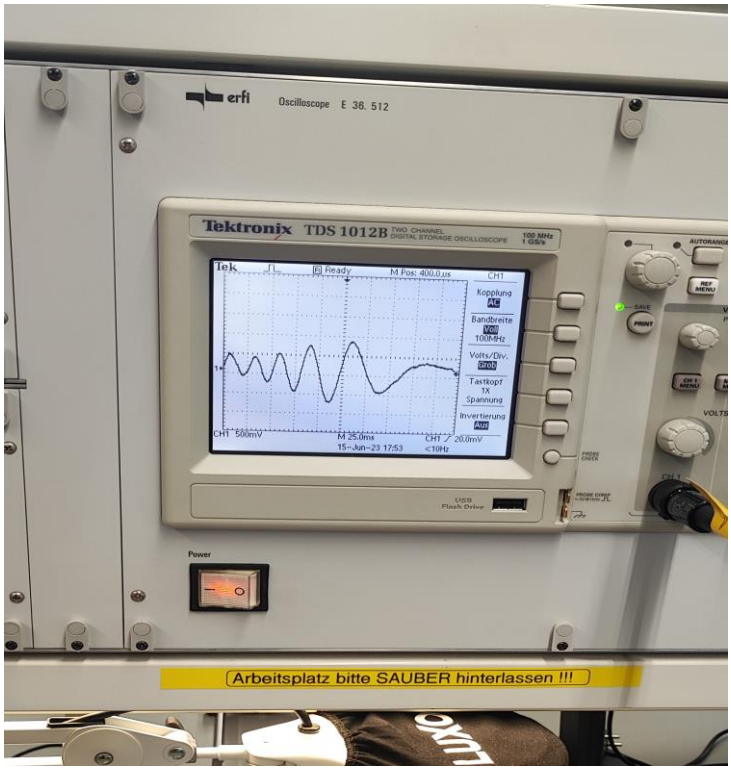


Fig 21. Radar module output

4 Software

4.1 Top Design for State-machine

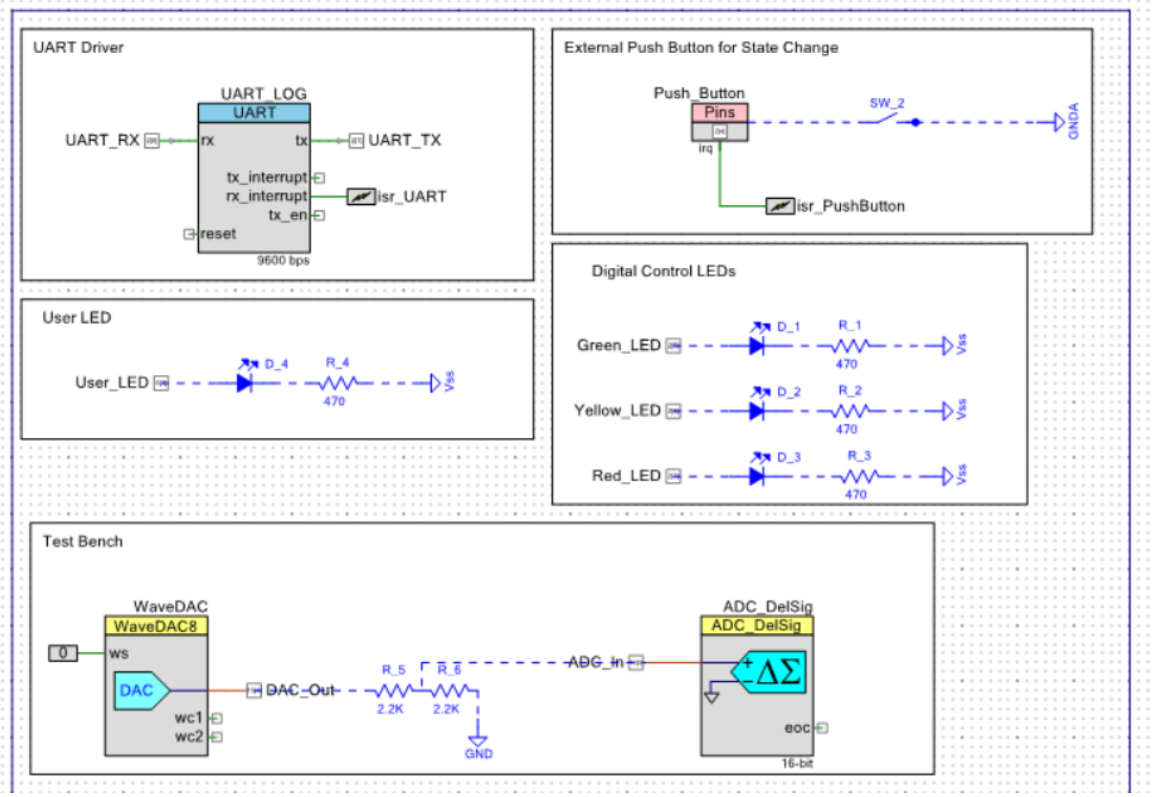
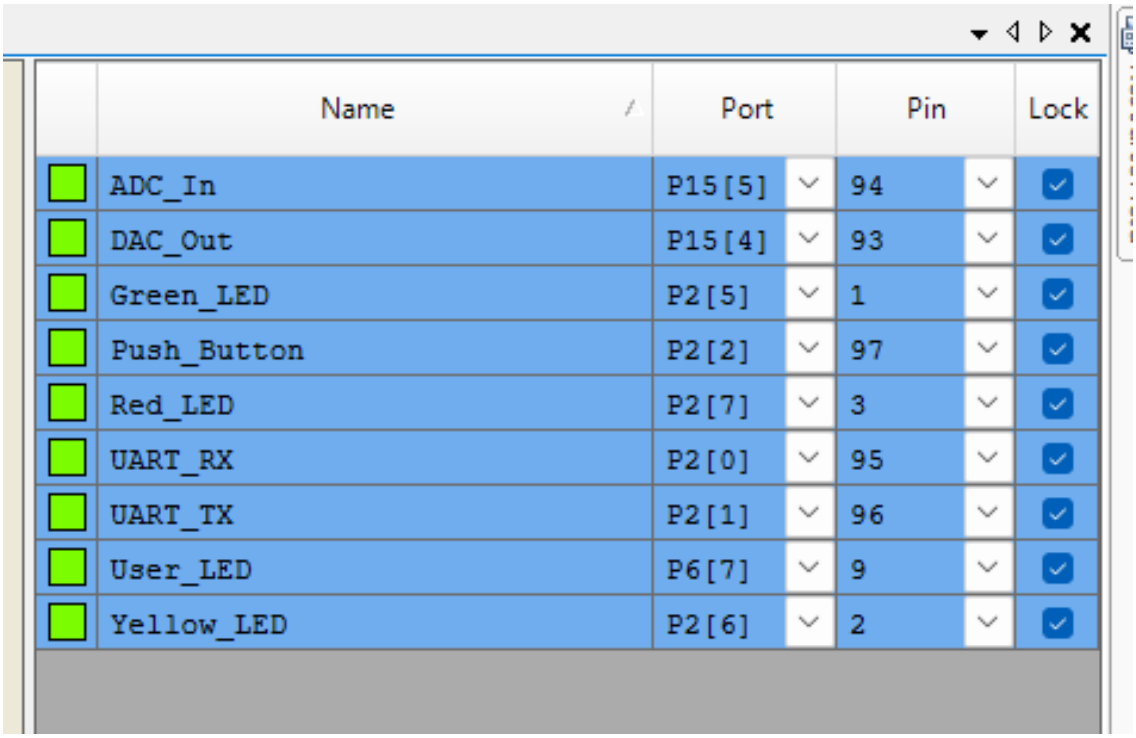


Fig 22. Top Design



Name	Port	Pin	Lock
ADC_In	P15[5]	94	✓
DAC_Out	P15[4]	93	✓
Green_LED	P2[5]	1	✓
Push_Button	P2[2]	97	✓
Red_LED	P2[7]	3	✓
UART_RX	P2[0]	95	✓
UART_TX	P2[1]	96	✓
User_LED	P6[7]	9	✓
Yellow_LED	P2[6]	2	✓

Fig 23. Pin mapping

4.1.1 ADC

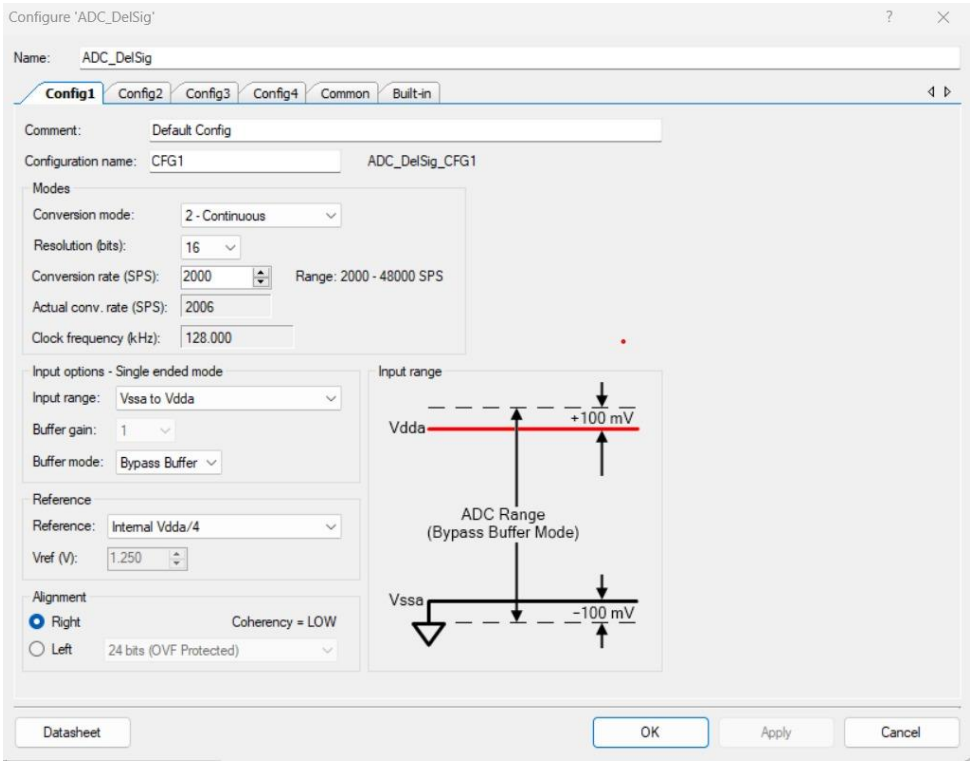


Fig 24. ADC Configuration

4.1.2 DAC

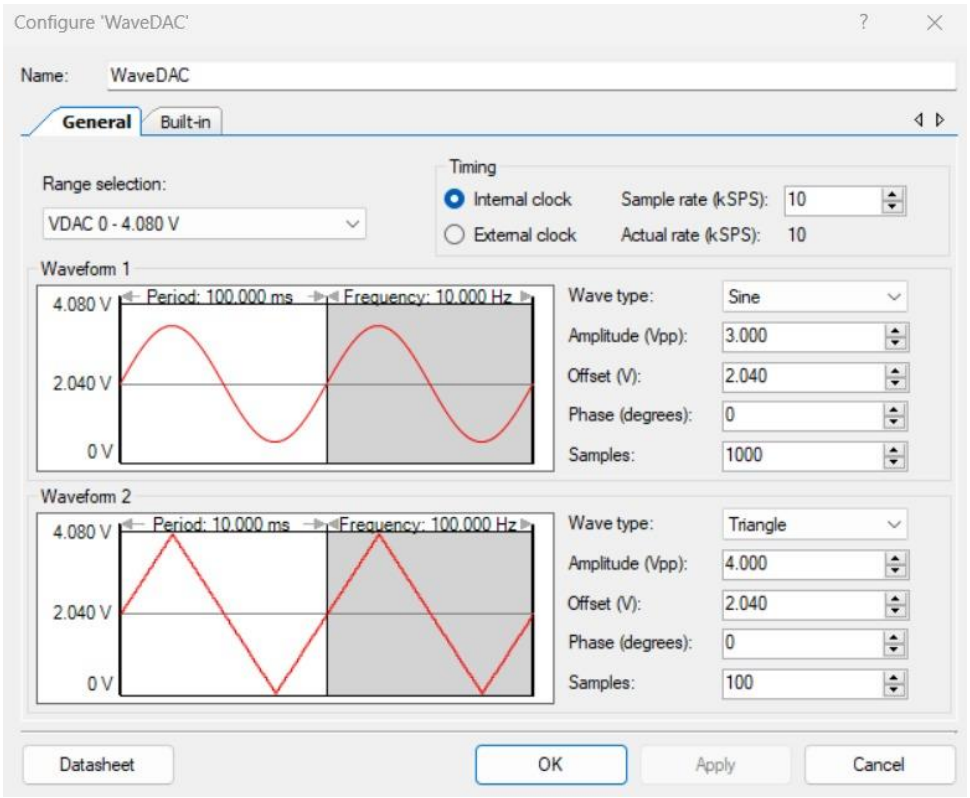


Fig 25. DAC Configuration

4.1.3 UART

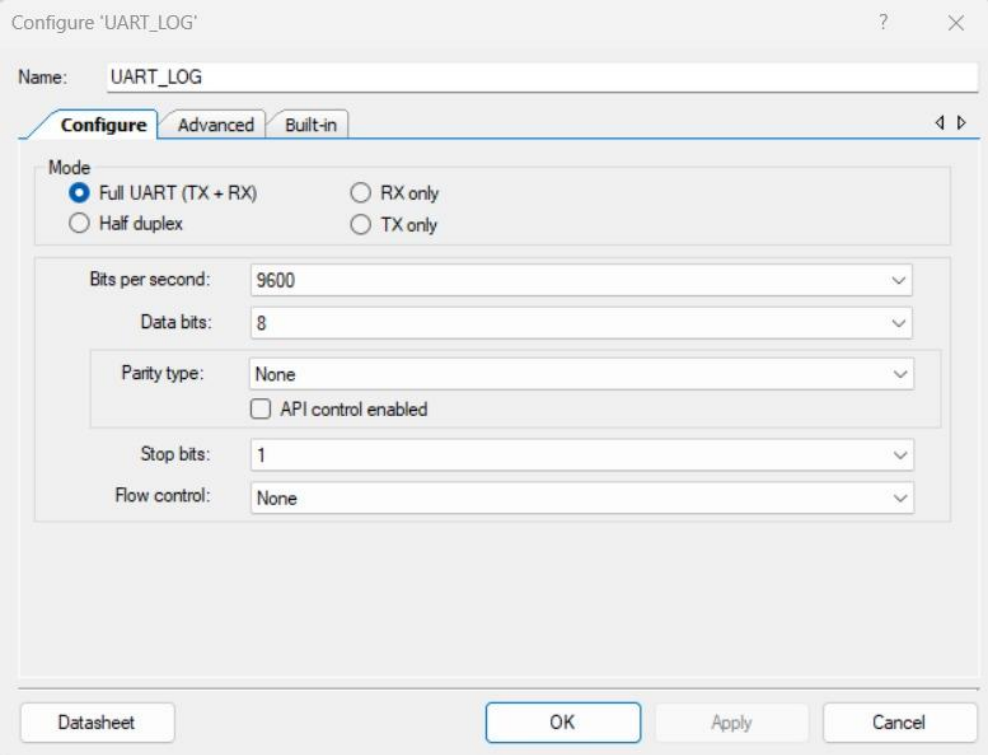


Fig 26. UART Configuration

4.2 State Diagram

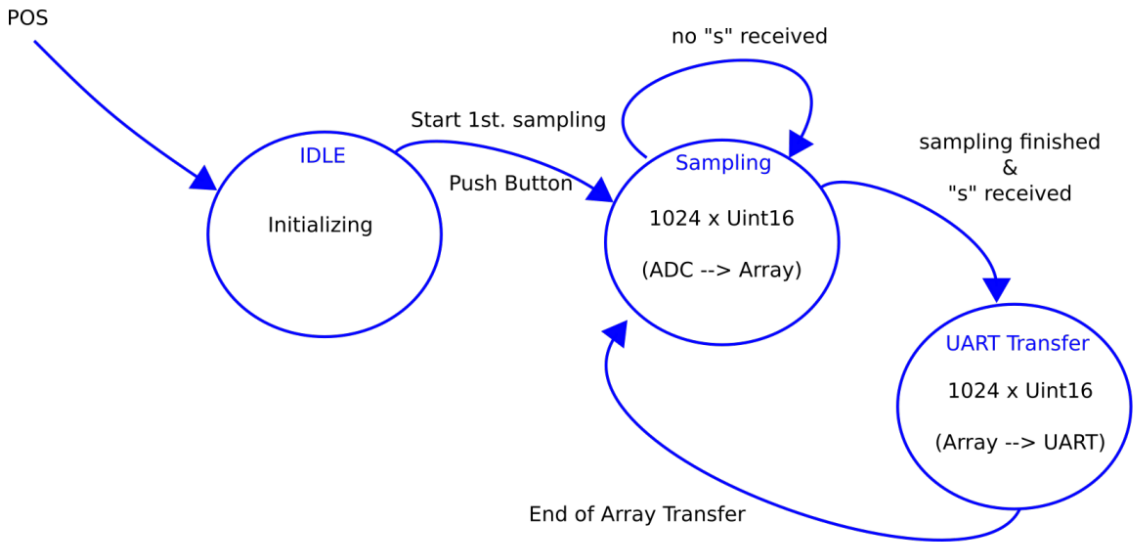


Fig 27. State Diagram

4.3 State-machine Code

4.3.1 Hardware Initialization

```
/**
 * Function that initializes all the Hardware used in the top design.
 * @param: None
 * @return: None
 */
void Hardware_Init()
{
    UART_LOG_Start();
    WaveDAC_Start();
    ADC_DelSig_Start();
    ADC_DelSig_StartConvert();

    isr_PushButton_StartEx(isr_PushButton_Enable_Interrupt);
    isr_UART_StartEx(isr_UART);
}
```

Fig 28. Hardware Init Function

4.3.2 ISR functions

```
/**
 * An interrupt function that stores 1 in push button toggle variable if PushButton is pressed.
 */
CY_ISR(isr_PushButton_Enable_Interrupt)
{
    Push_Button_ClearInterrupt();

    if(push_button_toggle == 0)
    {
        push_button_toggle = 1;
    }
}


/**
 * An interrupt function from UART that stores 's' or 'o' in their respective flag variables,
 * when the characters are sent from Matlab or Termite.
 */
CY_ISR(isr_UART)
{
    rxDataCh = UART_LOG_GetChar();

    if(rxDataCh == 's')
    {
       //UART_LOG_PutString("S is sent \n");        //Debug statement added while testing UART with Termite.
        sChRx = 1;
    }
    if(rxDataCh == 'o')
    {
        oChRx = 1;
    }
}
```

Fig 29. ISR Function definiton

4.3.3 Idle State

```
case IDLE_STATE:

    Yellow_LED_Write(1);              // Added as a debug LED, turns on when state is in IDLE.
    Green_LED_Write(0);
    Red_LED_Write(0);

    //Initializing count
    count =  0;

    if(push_button_toggle == 1)
    {
        // If push button is pressed then change the state to SAMPLING state.
        State = SAMPLING_STATE;
        push_button_toggle = 0;
    }
break;
```

Fig 30. Idle State logic

4.3.4 Sampling State

```
case SAMPLING_STATE:

    Yellow_LED_Write(0);          //Added as a debug LED; turns off when state changes from IDLE to SAMPLING.
    Green_LED_Write(1);
    Red_LED_Write(0);

    for(uint16_t idx = 0; idx < 1024; idx++)
    {
        //Sample the voltage divider output and store it in an Array.
        ADC_DelSig_IsEndConversion(ADC_DelSig_WAIT_FOR_RESULT);
        SamplingArray[idx] = (uint16_t)ADC_DelSig_GetResult32();
    }

    //If 's' is recieved from MATLAB, then change the state to UART TRANSFER state.
    if(sChRx)
    {
        // UART_LOG_PutString("I received S \n");       // Debug statement added when testing the statemachine through termite.
        User_LED_Write(1);                             // Added as a debug pin, turns on when 's' is received from Matlab or termite.
        State = UART_TRANSFER_STATE;
        sChRx = 0;
    }

break;
```

Fig 31. Sampling State logic

4.3.5 UART Transfer State

```
case UART_TRANSFER_STATE:

    Green_LED_Write(0);            //Added as a debug LED; turns off when state changes from IDLE to SAMPLING.
    Red_LED_Write(1);
    Yellow_LED_Write(0);

    //char buffer[100];


    for(uint16_t idx = 0; idx < 1024; idx++)
    {
        //Send the ADC Sampled data to MATLAB
        UART_LOG_PutChar(SamplingArray[idx]);
        UART_LOG_PutChar(SamplingArray[idx]>>8);
    }

    fft_app(SamplingArray,FFT_SamplesArray,1024);

    for(int32 idx = 0; idx < 2048; idx++)
    {
        //Send the ADC Sampled data to MATLAB
        UART_LOG_PutChar(FFT_SamplesArray[idx]);
        UART_LOG_PutChar(FFT_SamplesArray[idx]>>8);
        UART_LOG_PutChar(FFT_SamplesArray[idx]>>16);
        UART_LOG_PutChar(FFT_SamplesArray[idx]>>24);
    }


    count++;                       // Increment the count to repeat the above process for 10 times.

    if((count < 10) && oChRx)
    {
        User_LED_Write(0);         // Added as a debug pin, turns off when 'o' is received from Matlab or termite.
        State = SAMPLING_STATE;
        oChRx = 0;
    }
    else if((count == 10) && oChRx)
    {
        State = IDLE_STATE;
        oChRx = 0;
    }
break;
```

Fig 32. UART Transfer State logic

4.4 Matlab Code & Output

```
flg_data_avai = 0;
fwrite(PSoC,'s','uchar') % means send, I am ready to receive
while(flg_data_avai == 0)

        if PSoC.BytesAvailable >= ADC_bytes + FFT_Bytes
              fwrite(PSoC,'o','uchar') % means I received all expected data
              rx_data_adc = fread(PSoC,1024,'uint16');
              rx_data_fft = fread(PSoC,2048,'int32');
              fprintf(" Transfer %i DONE \n",count);

              fft_real = rx_data_fft(1:2:end);
              fft_imag = rx_data_fft(2:2:end);

              fft_mag = sqrt(fft_real.^2 + fft_imag.^2);

              % Plotting the received data

              figure(1)
              subplot(2,1,1)
              plot([0:(length(rx_data_adc)-1)],rx_data_adc(1:(length(rx_data_adc))));
              title(['Received Time Domain Data No.:',num2str(count)]);
              subplot(2,1,2)
              plot([0:1023],1/length(rx_data_adc)*20*log10(abs(fft(rx_data_adc))));
              title('FFT - Array - Matlab');

              figure(2)
              subplot(2,1,1)
              plot([0:(length(rx_data_adc)-1)],rx_data_adc(1:(length(rx_data_adc))));
              title(['Received Time Domain Data No.:',num2str(count)]);
              subplot(2,1,2)
              plot([0:1023],1/length(rx_data_adc)*20*log10(fft_mag));
              title('FFT - Array - PSoC');


              % Save the received data
              save(strcat('CW_rx_data_adc_',int2str(count),'.mat'),'rx_data_adc');
              count=count+1;
        end

        if count == 11
            break;
        end

        fwrite(PSoC,'s','uchar') % means send, I am ready to receive
end
```

Fig 33. MATLAB Script

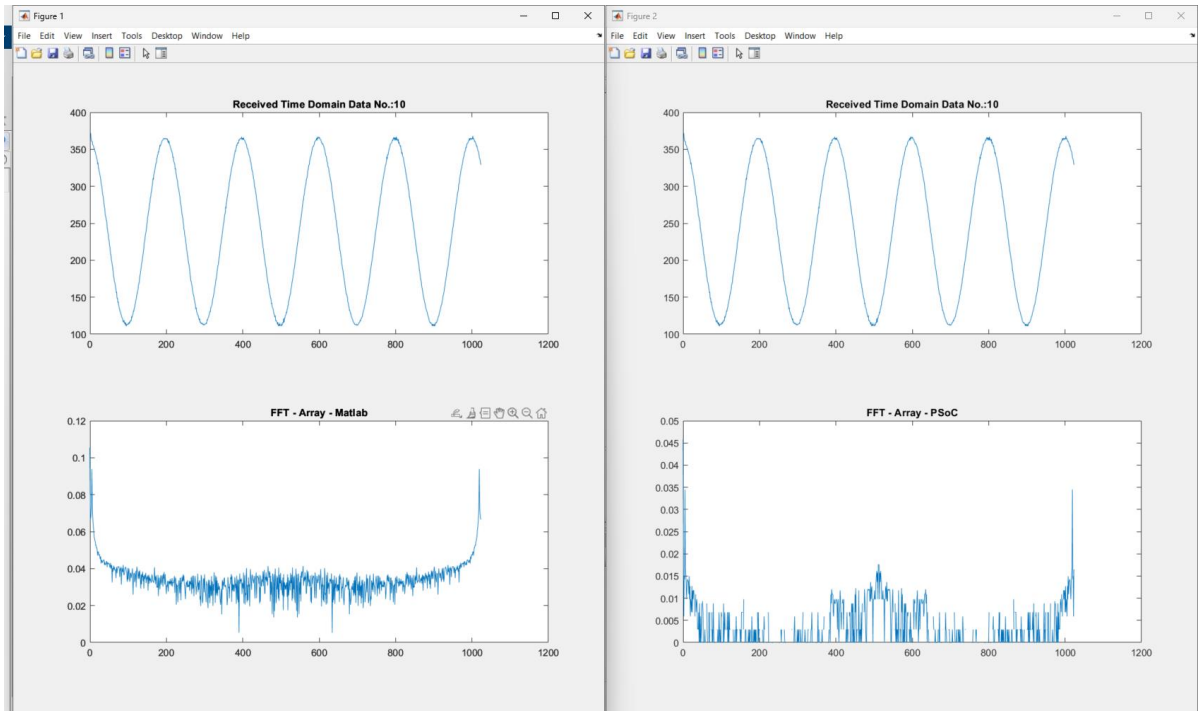


Fig 34. MATLAB FFT Output

5 CFAR Algorithm

5.1 Target detection

A threshold is calculated and set in order to detect a signal and differentiate it with respect to noise. This in turn is used to reduce the probability of false alarm detection.

The following probabilities are used:

P_D: A target is declared when a target is in fact present – Detection.
P_M: A target is not declared when a target is in fact present – Missed.
P_{FA}: A target is declared when a target is in fact not present – False alarm.

Formula to calculate threshold:

$$T = \sqrt{2N\sigma_w^2}\,\mathrm{erf}^{-1}(1 - 2\,P_{FA})$$

The probability of detection is calculated using:

$$P_D = \frac{1}{2}\,\mathrm{erfc}\left(\mathrm{erfc}^{-1}(2\,P_{FA}) - \sqrt{\frac{SNR}{2}}\right)$$

5.2 CFAR Code

```
% CFAR Detection
detection = zeros(1, N);

for i = NG + NR + 1 : N - (NG + NR)
    referenceWindow = xnoise(i - NR - NG : i + NR + NG);
    guardWindow = xnoise(i - NG : i + NG);
    Navg = 1 + sum(referenceWindow) + sum(guardWindow);

    SD=(1/Navg)*sum(xnoise);
    alpha=Navg*(PFA^(-1/Navg)-1);
    threshold  = alpha*SD;

    if xnoise(i) > threshold
        detection(i) = 1;
        %disp('Detection:', detection(i));
    end
end
```

Fig 35. CFAR Algorithm Script

5.3 CFAR Output

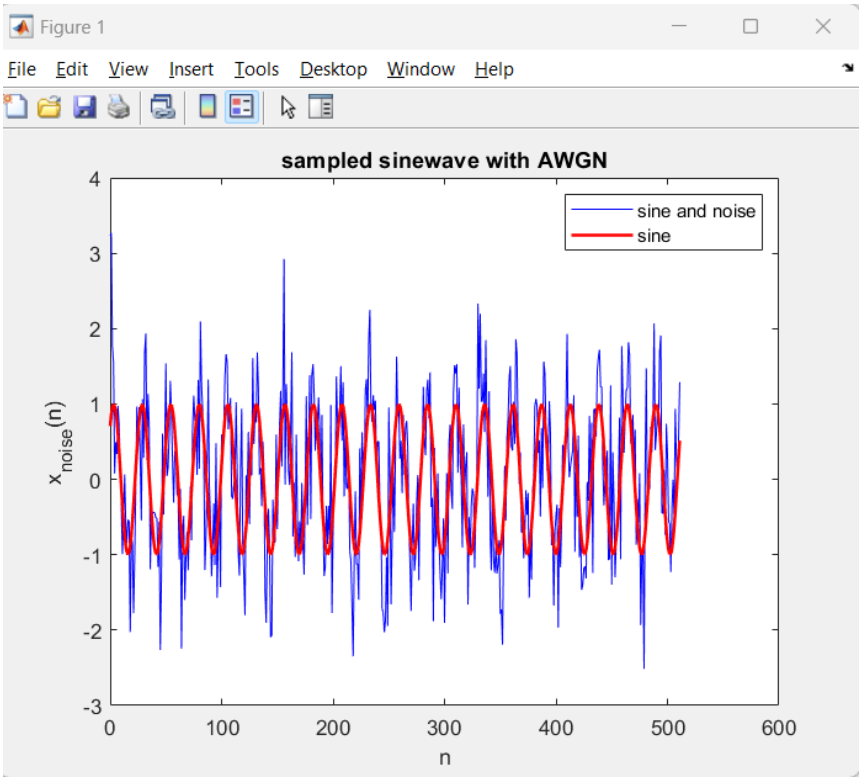


Fig 36. Signal + Noise Waveform

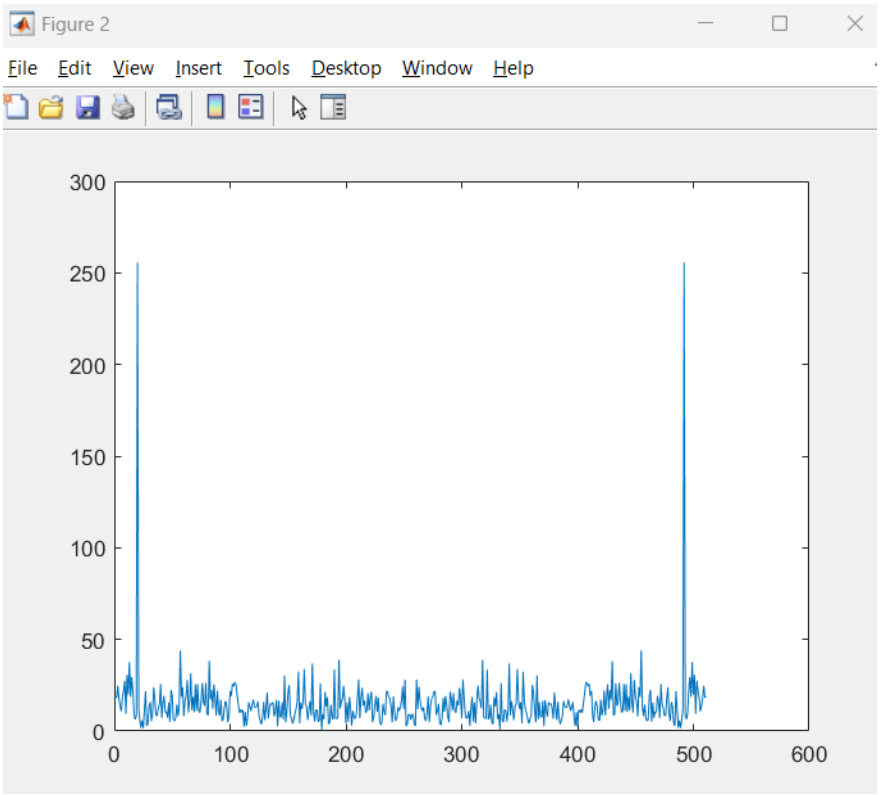


Fig 37. FFT of Signal + Noise

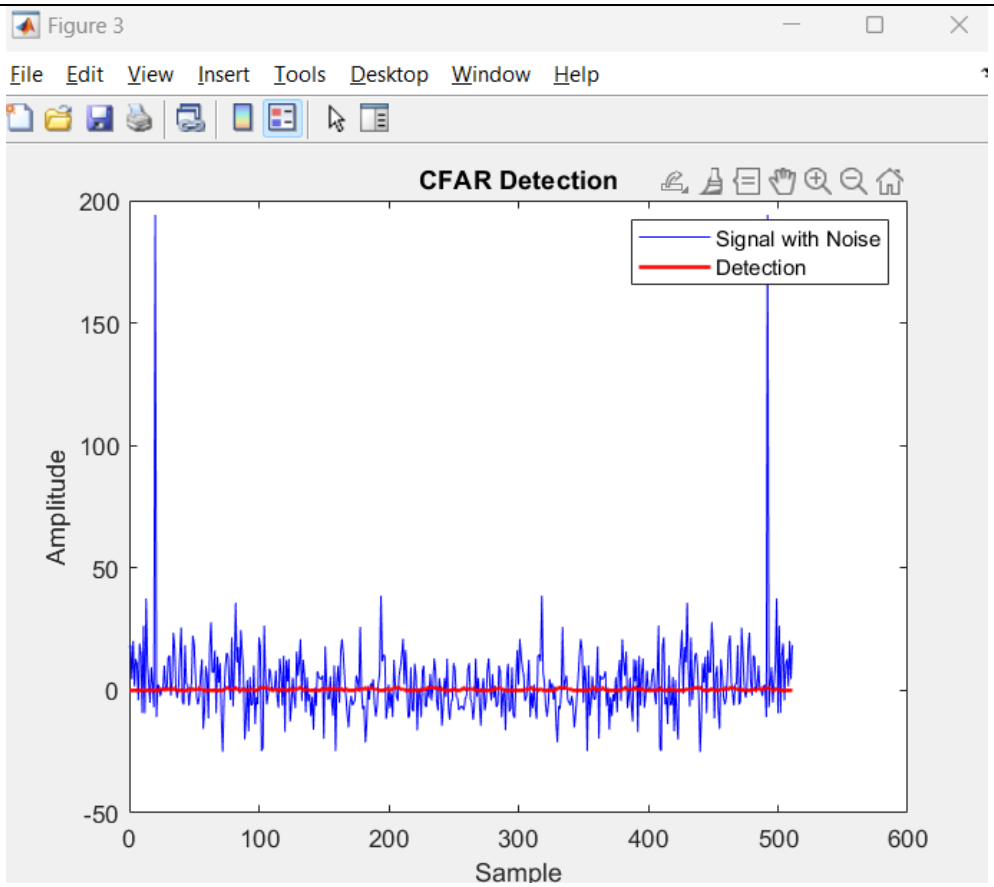


Fig 38. CFAR Detection

6 Manufactured & Soldered PCB

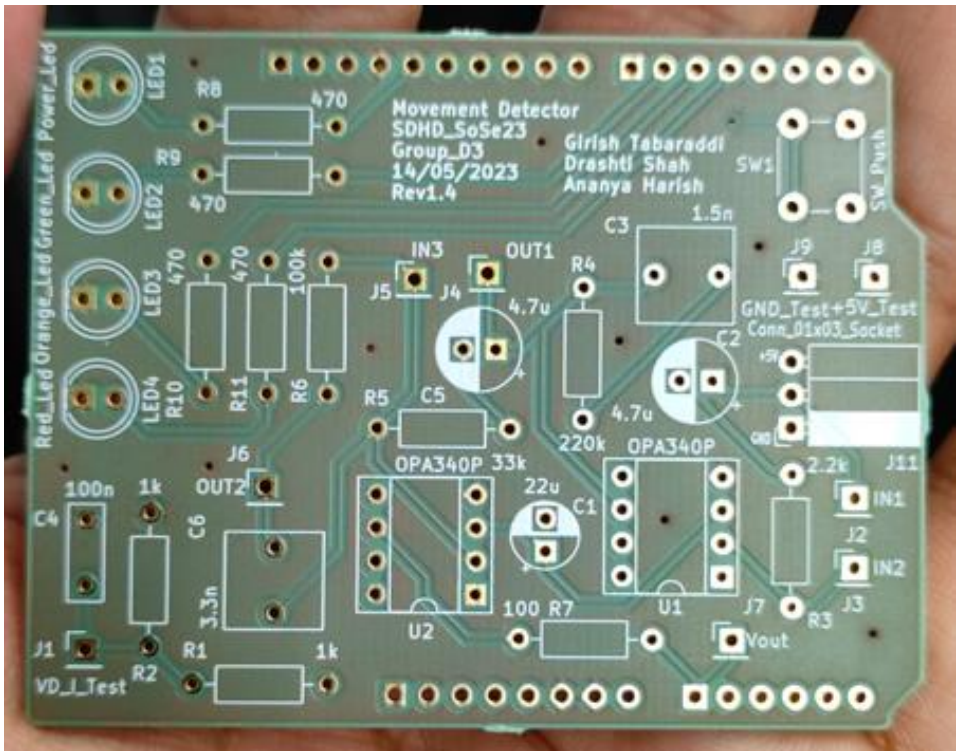


Fig 39. Unsoldered PCB

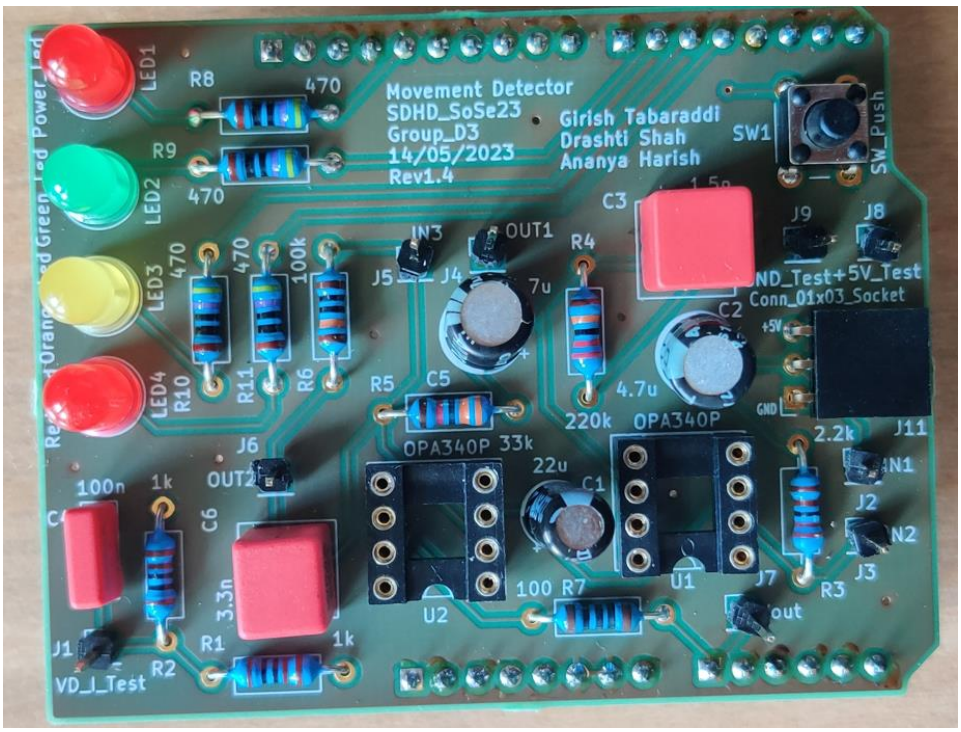


Fig 40. Soldered PCB Top View

7 References:

1. SDHD Lecture Slides
2. www.mathworks.com
3. https://mimir.h-da.io/public/index.htm
4. PSoC Getting Started by Thomas Barth
5. Embedded Architectures and Operating Systems Workbook (v2020a) by Prof. Dr.-Ing. Peter Fromm
6. https://www.sparkfun.com/products/13714
7. https://www.infineon.com/dgdl/Infineon-KitProg_User_Guide-UserManual-v01_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0eee99837dd9