SYSTEM DRIVEN HARDWARE DESIGN

Final Presentation

Development of a 24 GHz RADAR Sensor for movement detection

Group : A6

Prajwal Mangaluru (1134277)

Dheeraj Swaroop Saligrama Mahesh (1134298)

h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Agenda

- Introduction
- Block Diagram
- Schematic and PCB design
- 3D View and Soldered PCB
- State Machine
- Software Top Design
- Testbench
- Hardware Testing
- RADAR Testing
- CFAR Algorithm

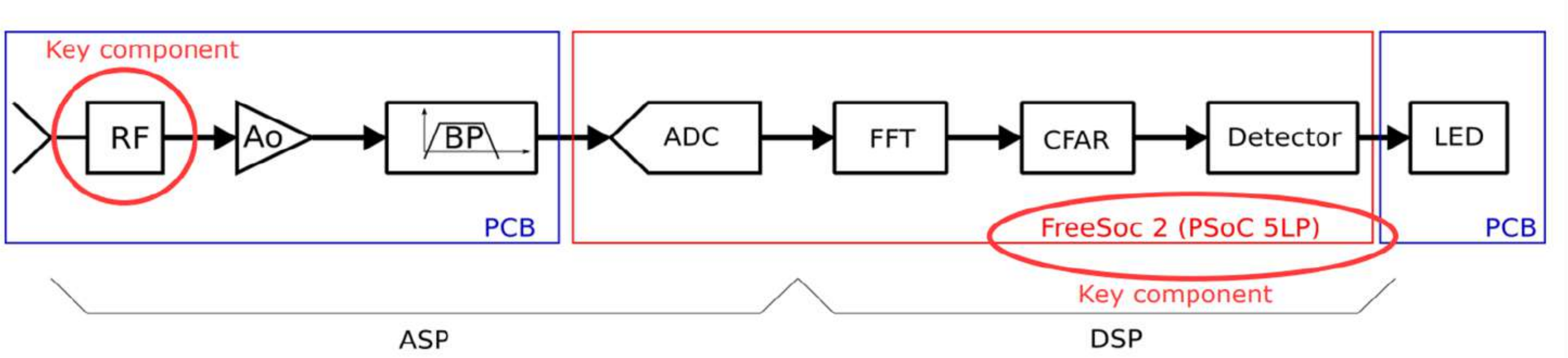
Introduction

- A 24 GHz radar sensor detects movements and send Doppler frequency to bandpass filter if the movement is detected.

- The 4th order Bandpass filter gives Baseband signal as the output which in-turn is send to the ADC.

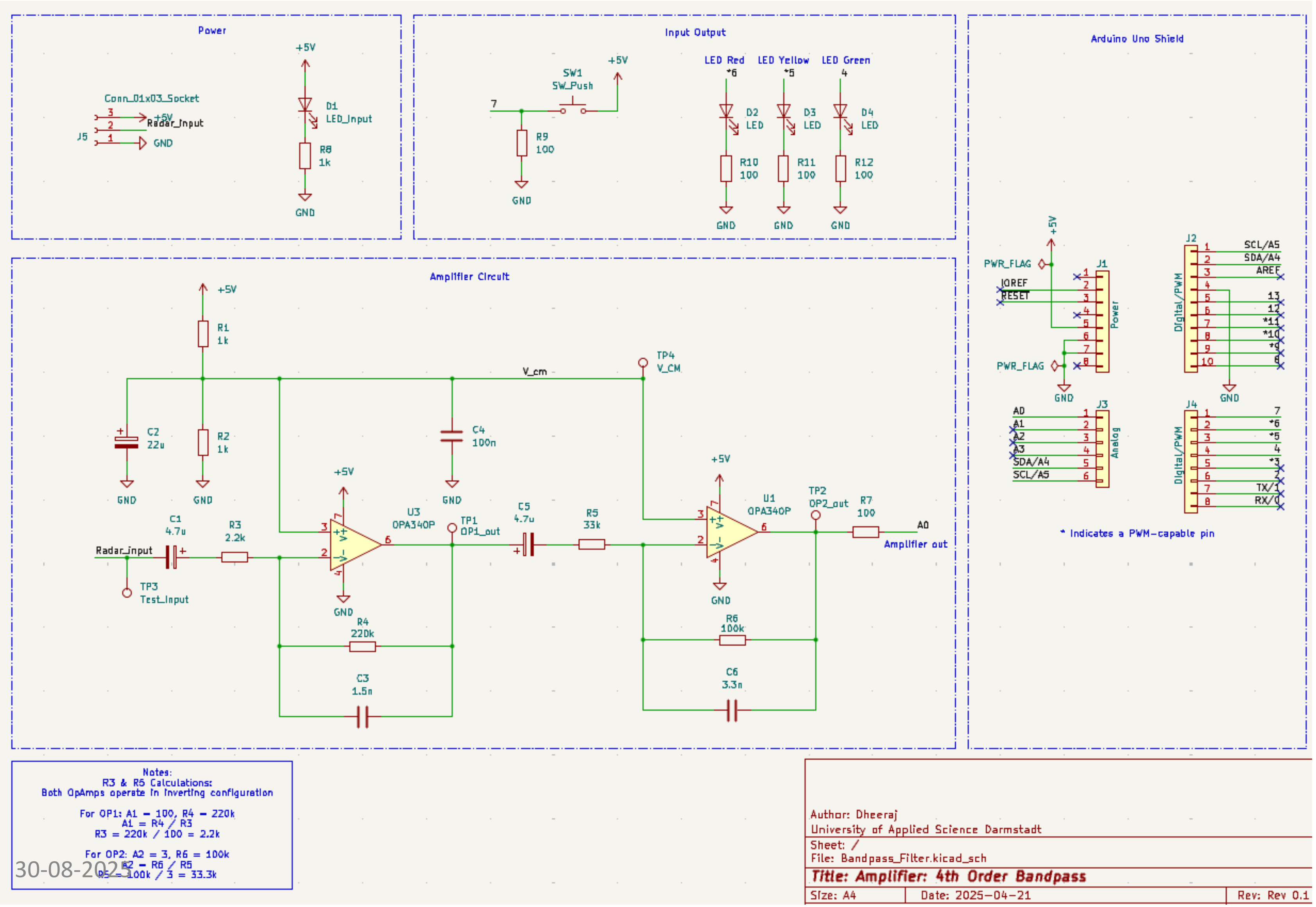
- Algorithms like FFT and CFAR, implemented in FreeSoC, enable precise target detection by identifying variations in the reflected signals.

Block Diagram



- The radar captures the RF signal.
- The signal is amplified and filtered to reduce noise.
- An ADC converts the cleaned analog signal into digital form.
- FFT analyzes the signal’s frequency components.
- CFAR filters out background clutter, highlighting potential targets.
- The system confirms if a real object is present.
- An LED lights up to indicate a detected target.

KiCad Schematic



Corner Frequency Calculations:

1. First High-Pass:

$$f_{HP1} = \frac{1}{2\pi \cdot R_3 \cdot C_1} = 15.4Hz$$

2. First Low-Pass:

$$f_{LP1} = \frac{1}{2\pi \cdot R_4 \cdot C_3} = 483Hz$$

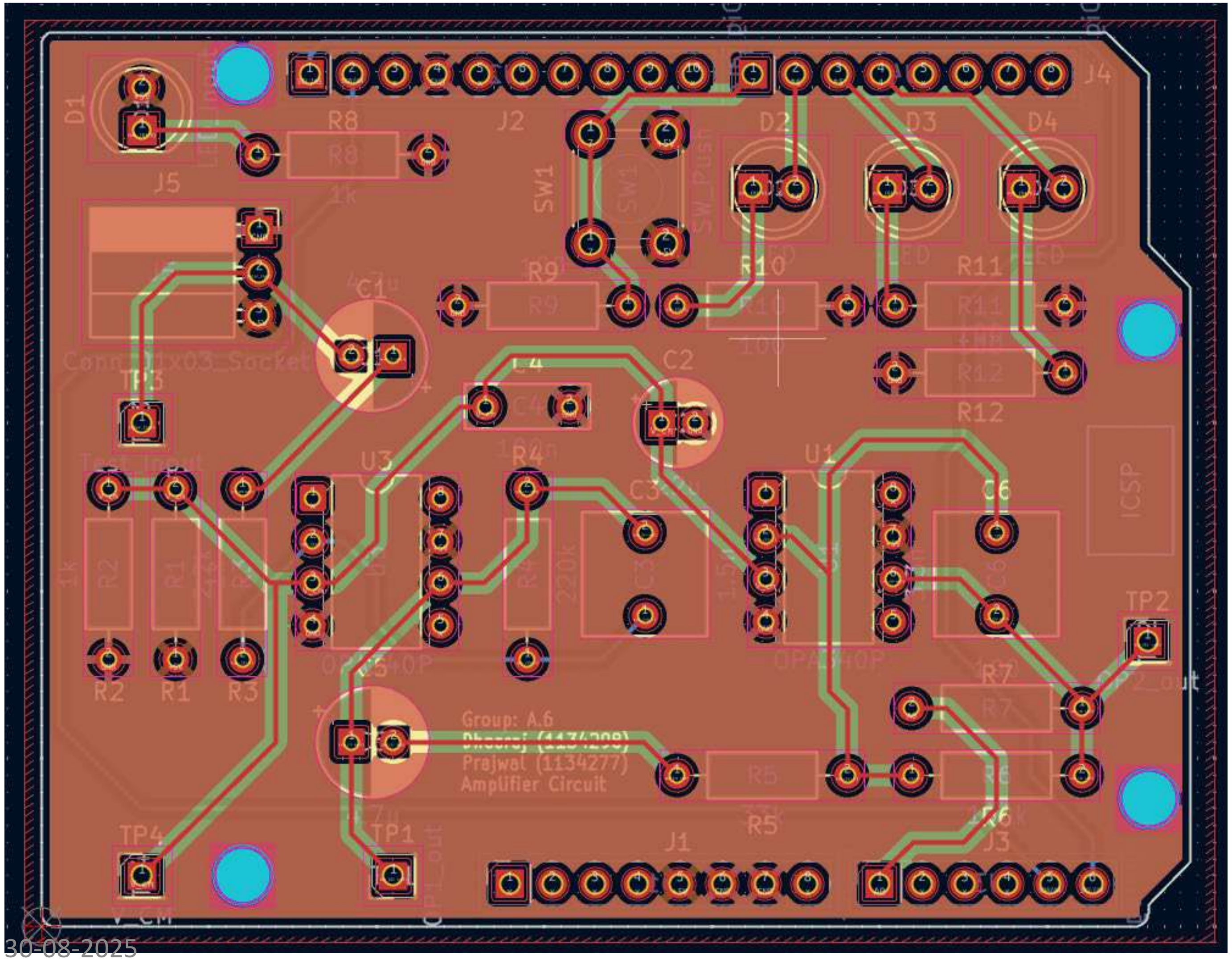
3. Second High-Pass:

$$f_{HP2} = \frac{1}{2\pi \cdot R_5 \cdot C_5} = 1.02Hz$$

4. Second Low-Pass:

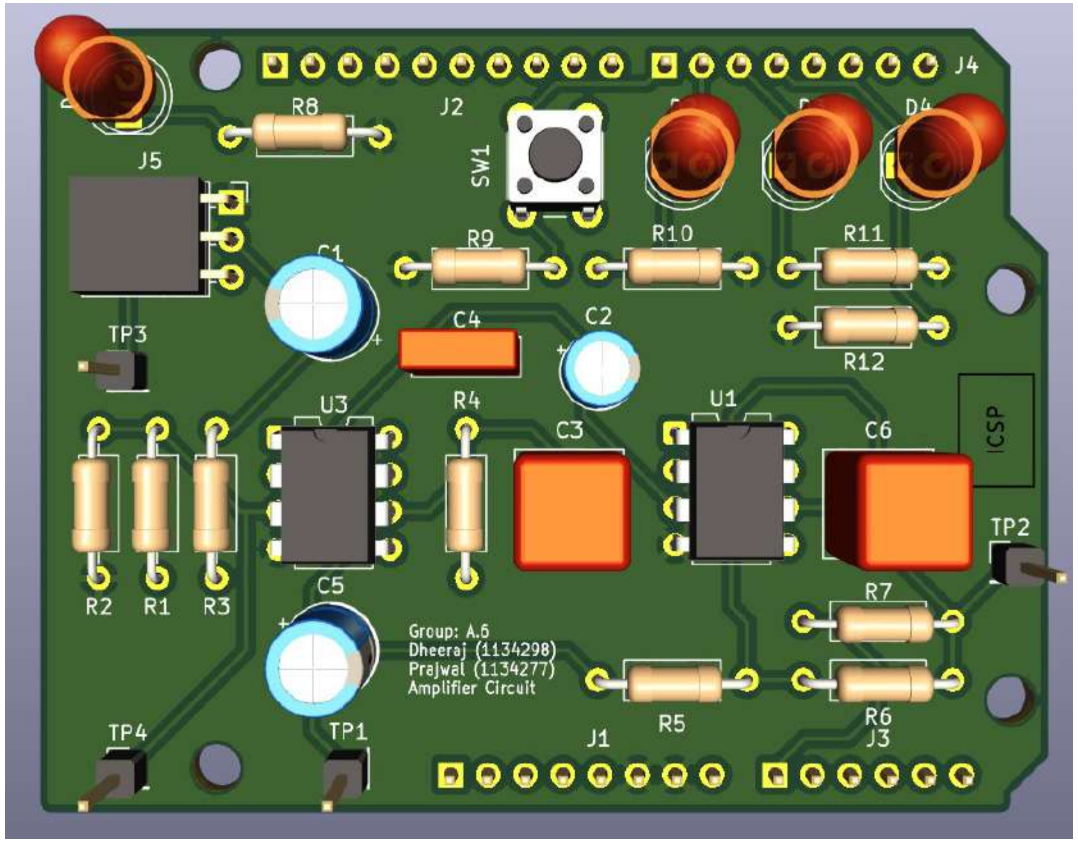
$$f_{LP2} = \frac{1}{2\pi \cdot R_6 \cdot C_6} = 483Hz$$

KiCad PCB Layout Design

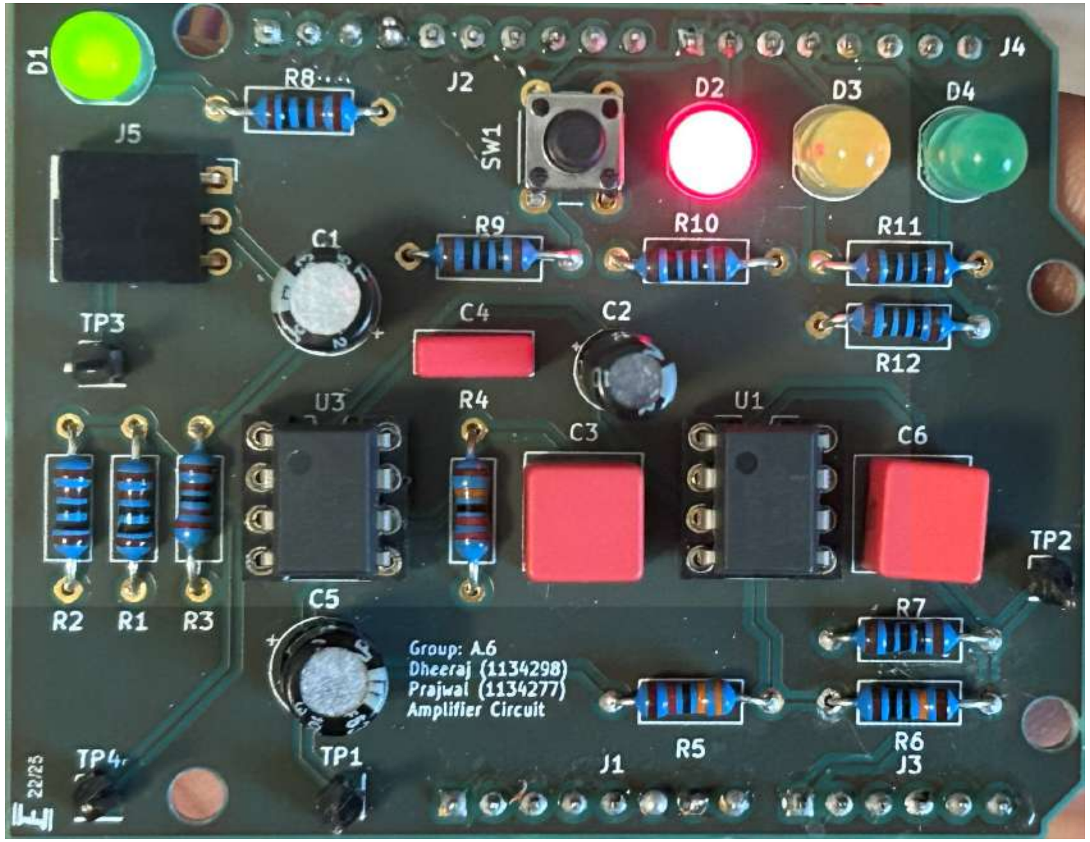


- Built using Through-Hole Technology (THT) for component mounting on the PCB.
- Four PCB layers:
  1. Front Copper
  2. Ground
  3. VCC
  4. Back Copper
- Digital and analog grounds are isolated to avoid interference.
- Powered by the FreeSoC board.

PCB 3D View and Soldered PCB



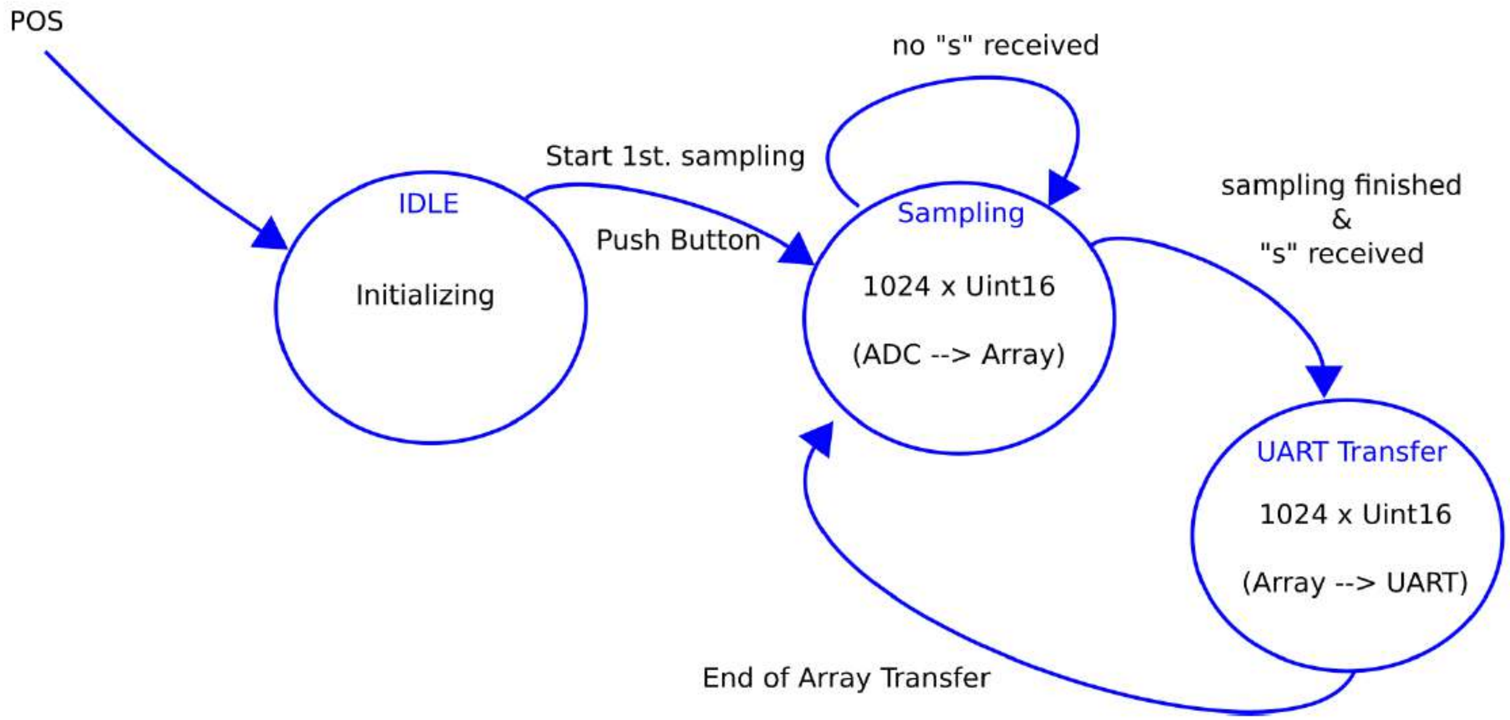
PCB 3D View



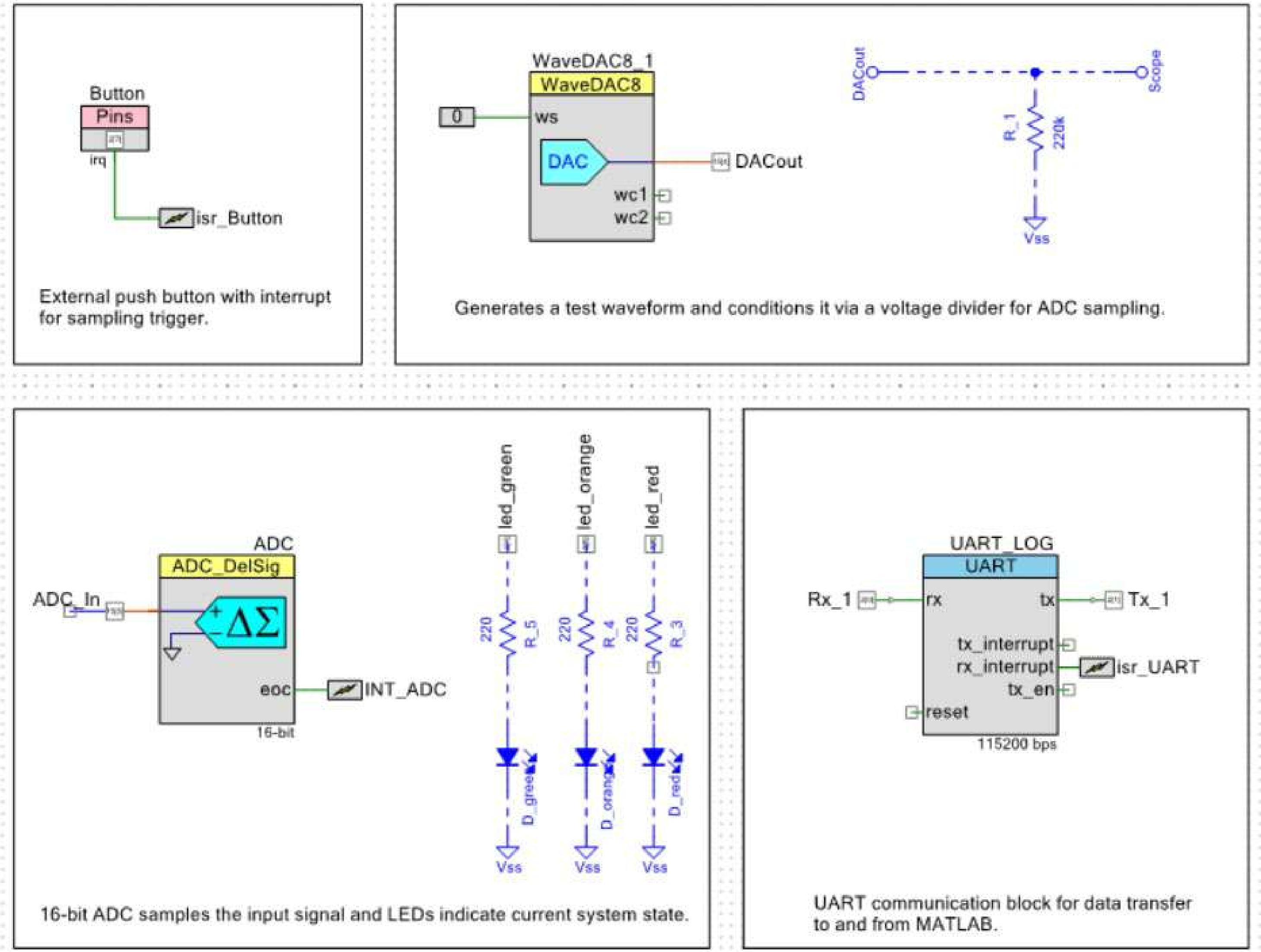
PCB Top View

State Machine

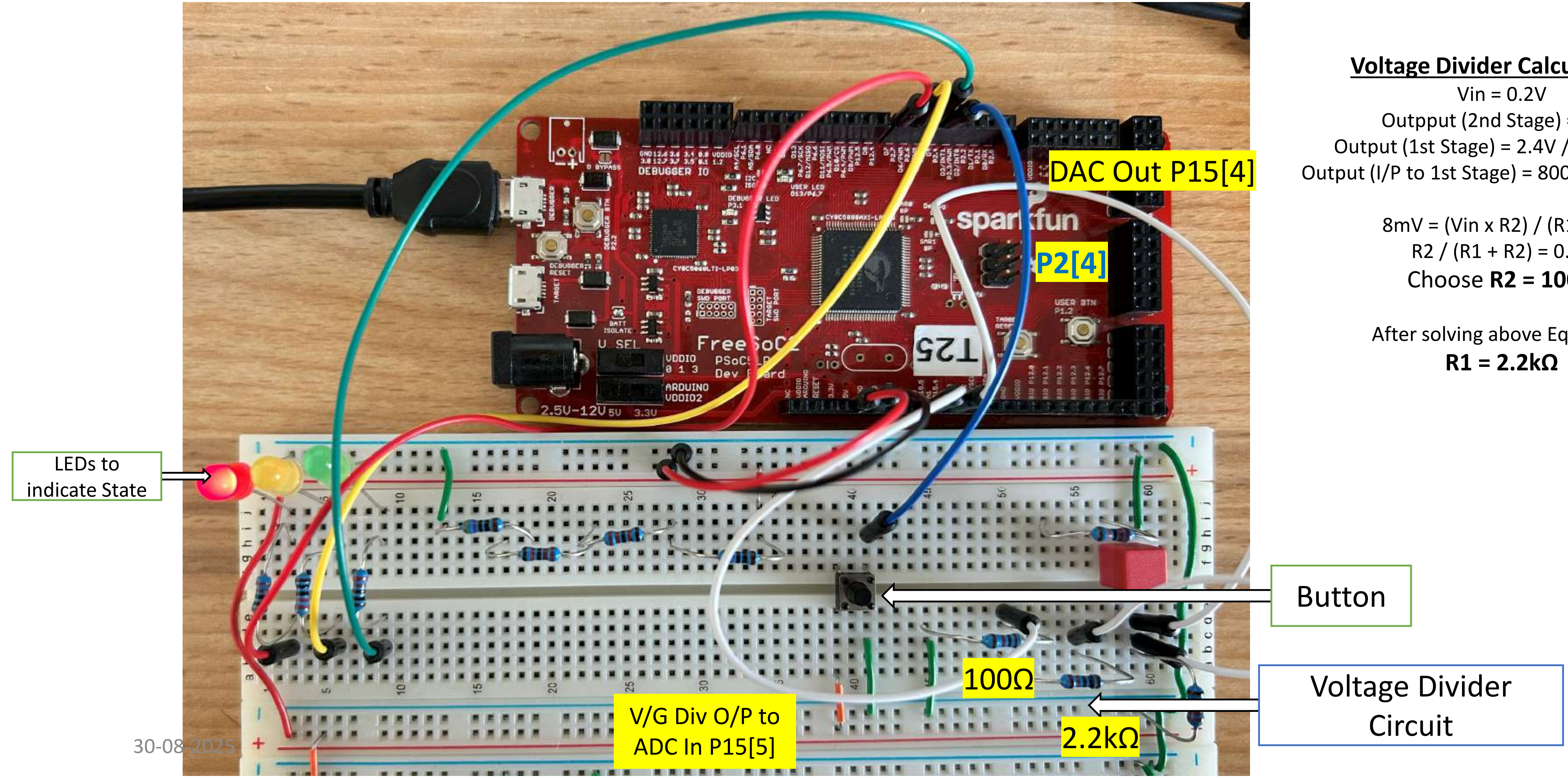
- This project utilizes the FreeSoC2 to sample analog signals through its ADC and send the digital data to MATLAB for processing.
- It was later extended to include FFT for frequency analysis and a CFAR algorithm to adaptively control the false alarm rate.



Software Top Design



Testbench



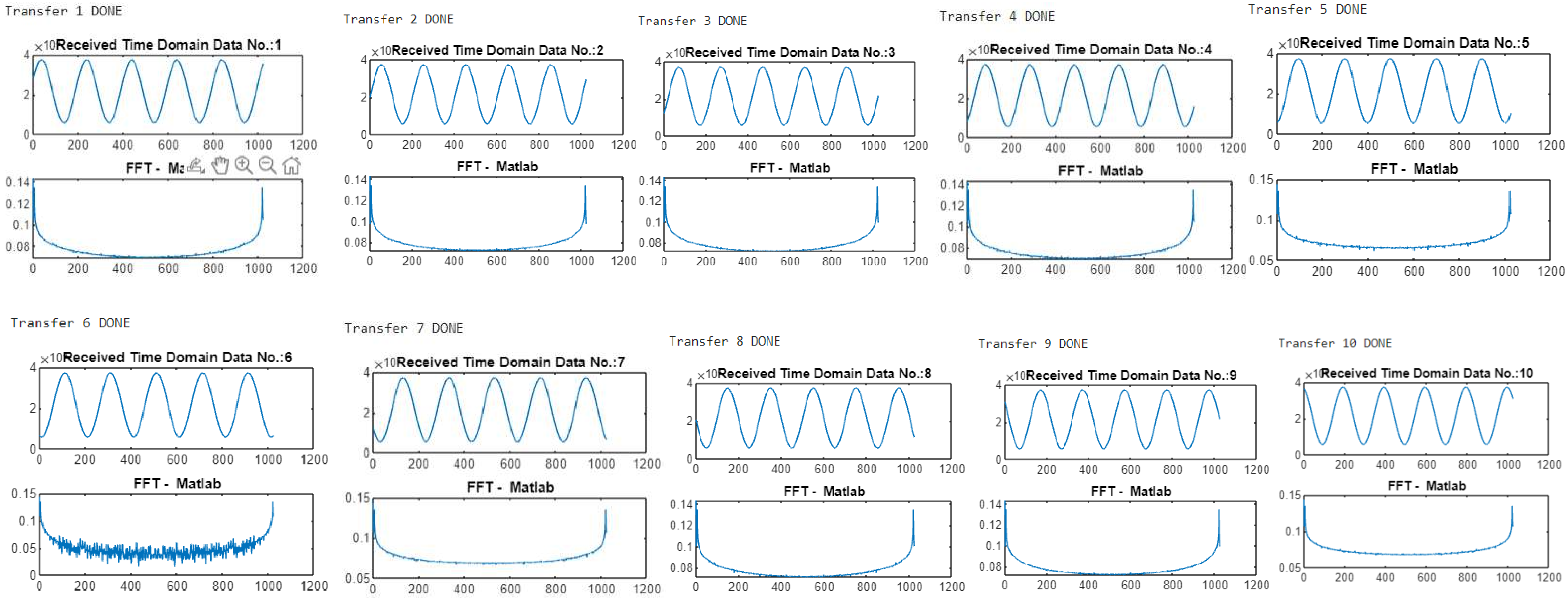
<u>Voltage Divider Calculations:</u>

Vin = 0.2V
Outpput (2nd Stage) = 2.4V
Output (1st Stage) = 2.4V / 3 = 800mV
Output (I/P to 1st Stage) = 800m / 100 = 8mV

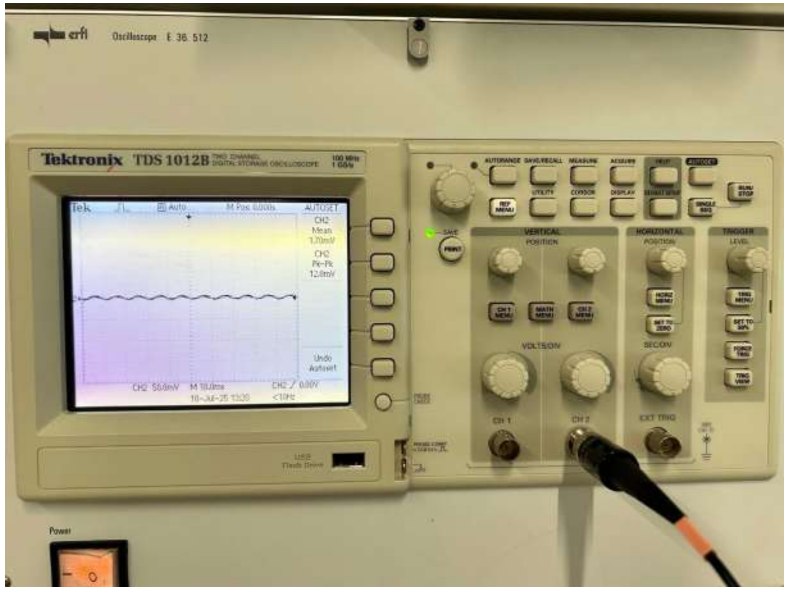
8mV = (Vin x R2) / (R1 + R2)
R2 / (R1 + R2) = 0.04
Choose **R2 = 100Ω**

After solving above Equation:
R1 = 2.2kΩ

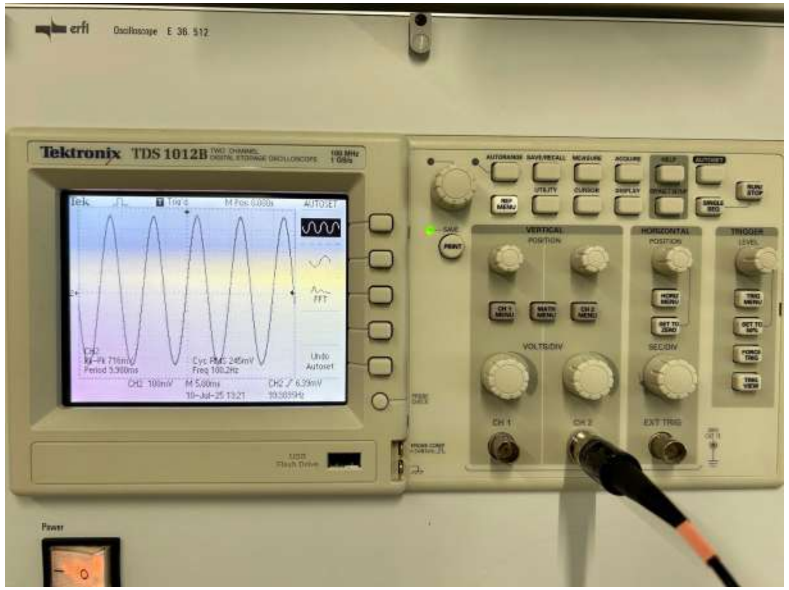
MATLAB Output



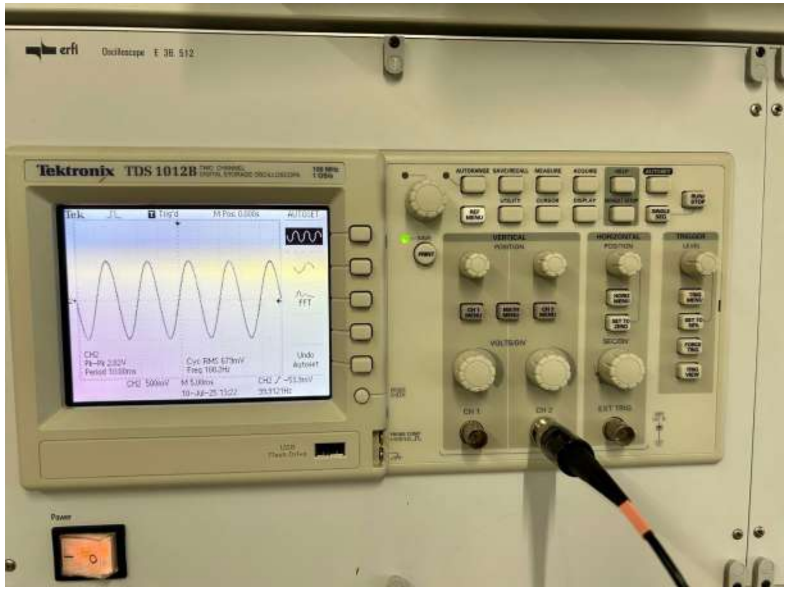
Hardware Testing



Voltage Divider Output
Expected : 8mV
Actual : 12mV



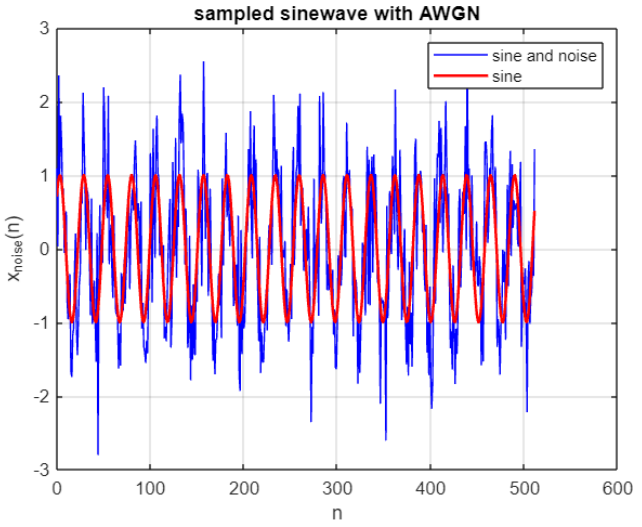
Amplifier 1st Stage Output
Expected : 800mV
Actual : 716mV

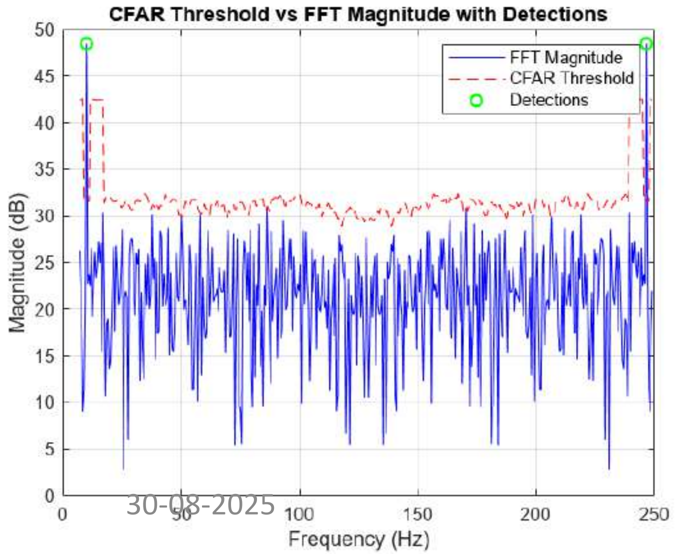


Amplifier 2nd Stage Output
Expected : 2.4V
Actual : 2.02V

RADAR Testing

CFAR Algorithm Implementation – MATLAB (Test Input)






```
% === CFAR Processing Loop ===
for i = (NR + NG + 1):(N - (NR + NG))
    noise_sum = 0;

    % Leading reference cells
    for j = (i - NR - NG):(i - NG - 1)
        re = fft_buffer(2*j - 1);
        im = fft_buffer(2*j);
        noise_sum = noise_sum + re^2 + im^2;
    end

    % Lagging reference cells
    for j = (i + NG + 1):(i + NG + NR)
        re = fft_buffer(2*j - 1);
        im = fft_buffer(2*j);
        noise_sum = noise_sum + re^2 + im^2;
    end

    noise_avg = noise_sum / (2*NR);
    threshold = alpha * noise_avg;
    thresholds(i) = threshold;

    % CUT (Cell Under Test)
    re = fft_buffer(2*i - 1);
    im = fft_buffer(2*i);
    mag = re^2 + im^2;
    magnitude(i) = mag;

    if mag > threshold && mag > MIN_MAG
        cfar_output(i) = 1;
    end
end
```

Noise estimate in the reference cells

$$\widehat{\sigma_w^2} = \frac{1}{N}\sum_{k=1}^{N} x_{ref}$$

$$x_{ref}(k) = |DFT(k)|^2$$

Required Threshold

$$\hat{T} = \alpha\,\widehat{\sigma_w^2}$$

For α we obtain:

$$\alpha = N\left(\bar{P}_{FA}^{-1/N} - 1\right)$$

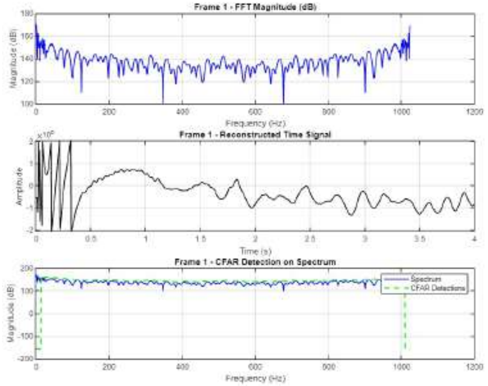
with:

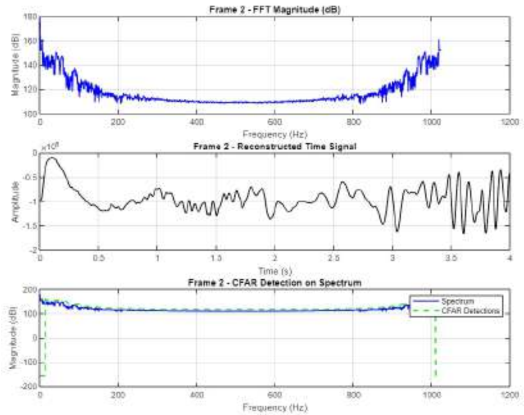
N number of averaged cells

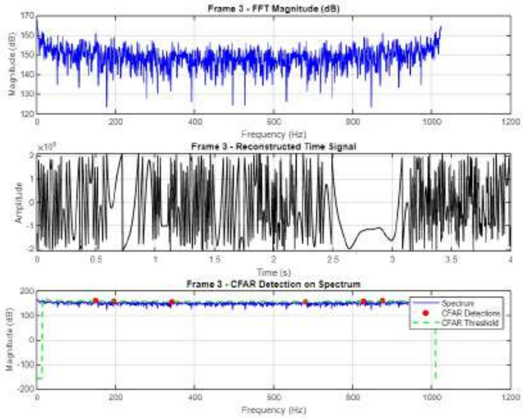
P_{FA} target false alarm probabilty

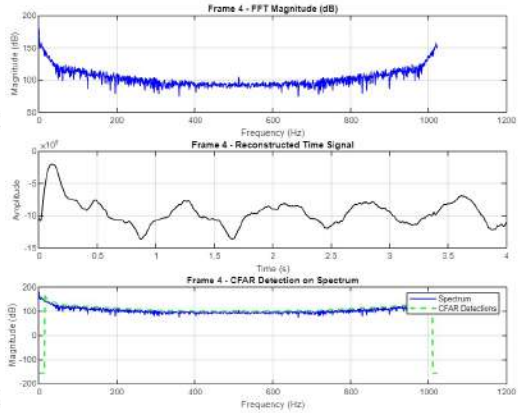
CFAR Algorithm Implementation - PSoC

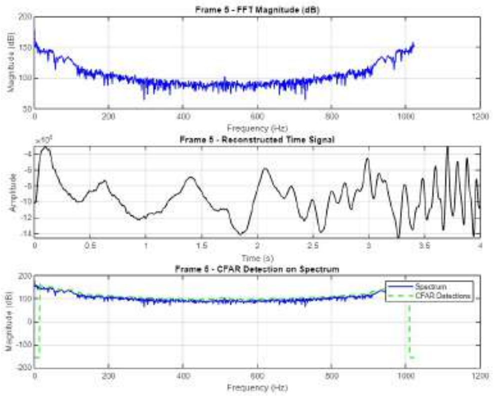
```c
void run_cfar(const int32_t *fft_buffer, uint16_t num_samples)
{
    // CFAR configuration parameters
    const int NG = 2;                           // Number of Guard Cells on one side of the CUT (Cell Under Test)
    const int NR = 12;                          // Number of Reference Cells on one side of the CUT
    const float PFA = 1e-2f;                    // Desired Probability of False Alarm
    const float MIN_MAG = 1e-4f;                // Minimum magnitude threshold to suppress low-level noise
    const float alpha = NR * (powf(PFA, -1.0f / NR) - 1.0f);  // CFAR scaling factor based on PFA and NR

    // Step 1: Initialize output array to zero (no detections)
    for (int i = 0; i < num_samples; i++)
        cfar_output[i] = 0;

    // Step 2: Slide the CFAR window over each FFT bin, skipping edges
    //         (Start and end bins cannot form full reference and guard regions)
    for (int i = NR + NG; i < num_samples - (NR + NG); i++)
    {
        float noise_sum = 0.0f;  // Accumulator for noise power in reference cells

        // --- Leading Reference Cells (before the guard cells) ---
        // For each reference cell before the CUT, calculate magnitude^2 and accumulate
        for (int j = i - NR - NG; j < i - NG; j++)
        {
            // Convert Q31 to float [-1.0, 1.0]
            float re = fft_buffer[2 * j]     / 2147483648.0f;
            float im = fft_buffer[2 * j + 1] / 2147483648.0f;
            noise_sum += re * re + im * im;  // Power = real^2 + imag^2
        }

        // --- Lagging Reference Cells (after the guard cells) ---
        // Same as above, for reference cells after the CUT
        for (int j = i + NG + 1; j <= i + NG + NR; j++)
        {
            float re = fft_buffer[2 * j]     / 2147483648.0f;
            float im = fft_buffer[2 * j + 1] / 2147483648.0f;
            noise_sum += re * re + im * im;
        }

        // Step 3: Calculate average noise power and threshold
        float noise_avg = noise_sum / (2.0f * NR);  // Average over all reference cells
        float threshold = alpha * noise_avg;        // Scale it to compute the detection threshold

        // Step 4: Calculate CUT magnitude
        float re = fft_buffer[2 * i]     / 2147483648.0f;
        float im = fft_buffer[2 * i + 1] / 2147483648.0f;
        float magnitude = re * re + im * im;

        // Step 5: Detection logic
        // If CUT magnitude > threshold and above minimum magnitude, mark as detection
        if (magnitude > threshold && magnitude > MIN_MAG)
        {
            cfar_output[i] = 1;         // Mark detection
            led_red_Write(1);           // Turn on RED LED as an indicator (application-specific)
        }
    }
}
```

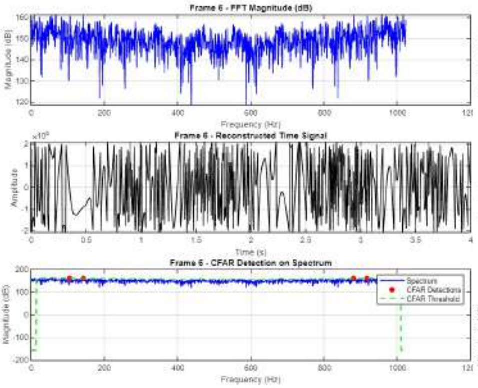

Frame 1 - FFT Magnitude (dB)
Frame 1 - Reconstructed Time Signal
Frame 1 - CFAR Detection on Spectrum
Spectrum
CFAR Detections

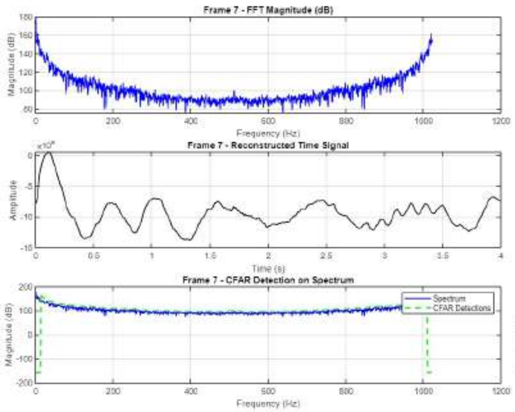

Frame 2 - FFT Magnitude (dB)
Frame 2 - Reconstructed Time Signal
Frame 2 - CFAR Detection on Spectrum
Spectrum
CFAR Detections

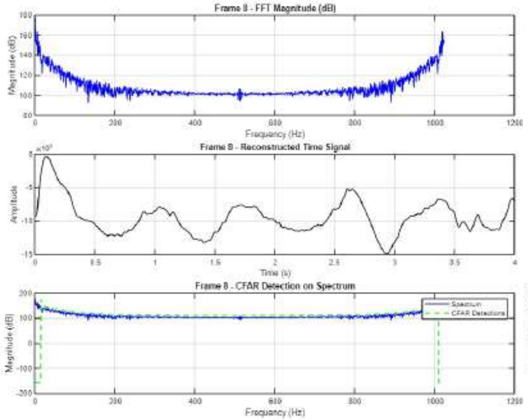

Frame 3 - FFT Magnitude (dB)
Frame 3 - Reconstructed Time Signal
Frame 3 - CFAR Detection on Spectrum
Spectrum
CFAR Detections
CFAR Threshold

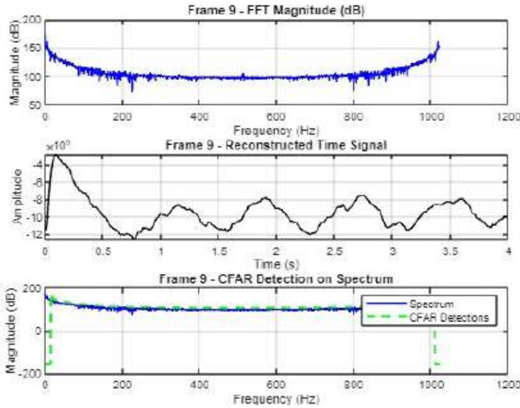

Frame 4 - FFT Magnitude (dB)
Frame 4 - Reconstructed Time Signal
Frame 4 - CFAR Detection on Spectrum
Spectrum
CFAR Detections

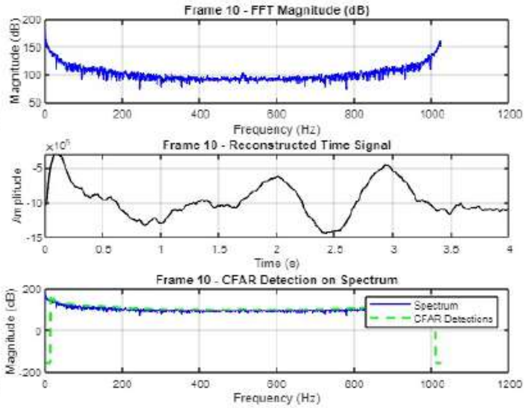

Frame 5 - FFT Magnitude (dB)
Frame 5 - Reconstructed Time Signal
Frame 5 - CFAR Detection on Spectrum
Spectrum
CFAR Detections


Frame 6 - FFT Magnitude (dB)
Frame 6 - Reconstructed Time Signal
Frame 6 - CFAR Detection on Spectrum
Spectrum
CFAR Detections
CFAR Threshold


Frame 7 - FFT Magnitude (dB)
Frame 7 - Reconstructed Time Signal
Frame 7 - CFAR Detection on Spectrum
Spectrum
CFAR Detections


Frame 8 - FFT Magnitude (dB)
Frame 8 - Reconstructed Time Signal
Frame 8 - CFAR Detection on Spectrum
Spectrum
CFAR Detections


Frame 9 - FFT Magnitude (dB)
Magnitude (dB)
Frequency (Hz)
Frame 9 - Reconstructed Time Signal
Amplitude
Time (s)
Frame 9 - CFAR Detection on Spectrum
Spectrum
CFAR Detections


Frame 10 - FFT Magnitude (dB)
Magnitude (dB)
Frequency (Hz)
Frame 10 - Reconstructed Time Signal
Amplitude
Time (s)
Frame 10 - CFAR Detection on Spectrum
Spectrum
CFAR Detections