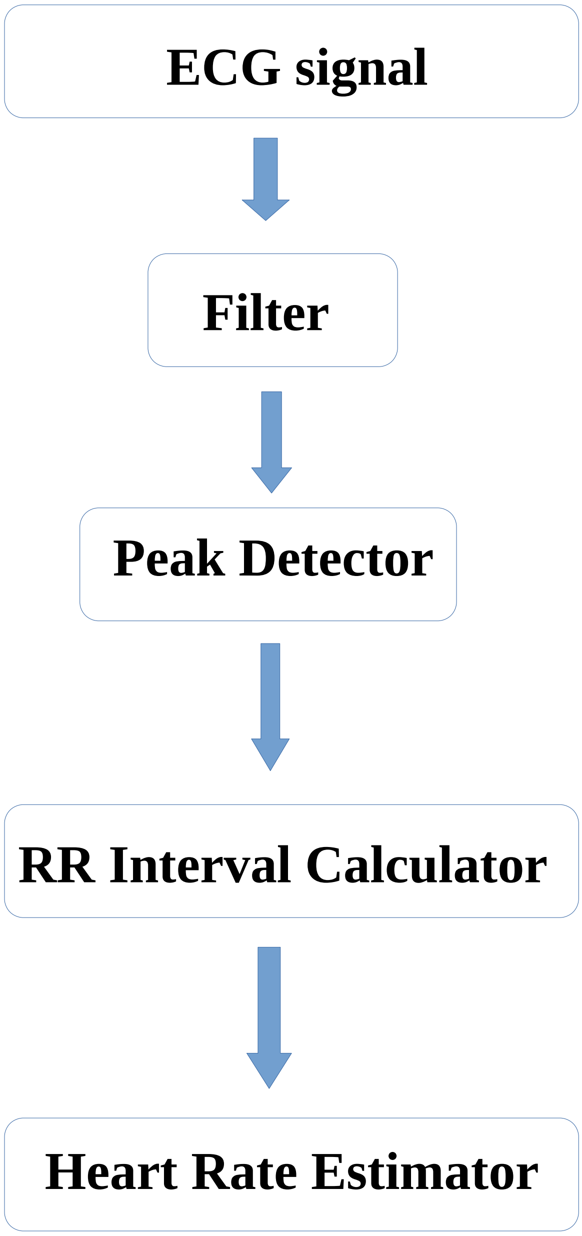
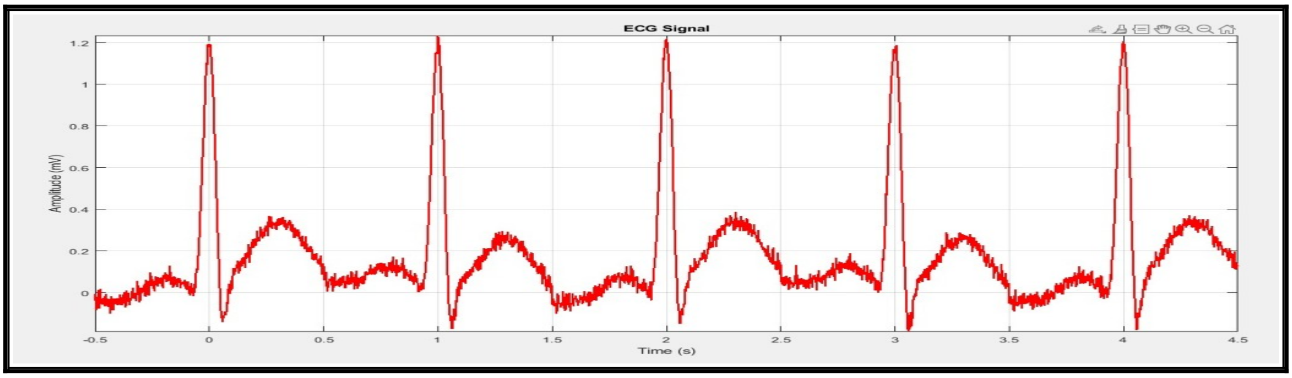
HEART RATE ESTIMATOR



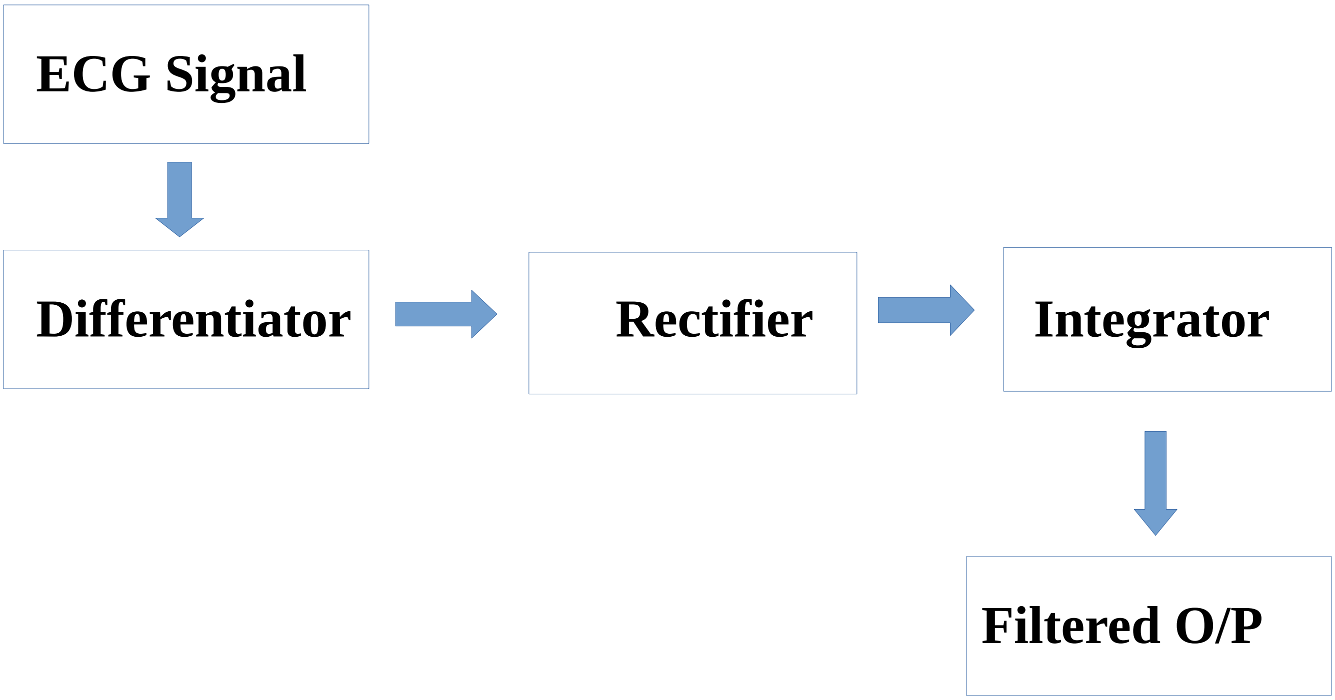
ECG Signal



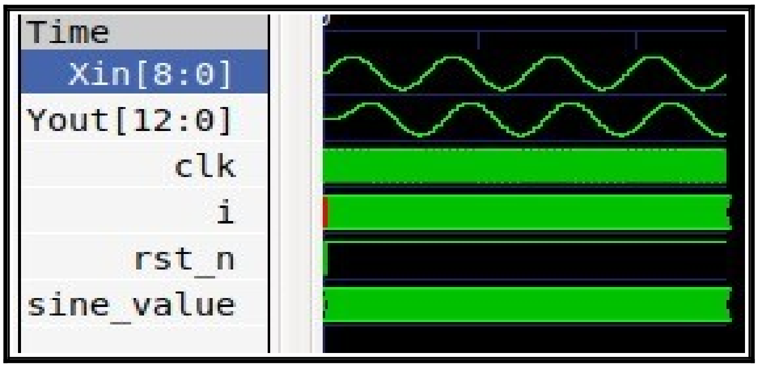

```
clc;
clear;
close all;

% Sampling frequency
fs = 500;  % Standard ECG sampling rate (Hz)
t_total = 5; % Total duration (seconds) for 5 beats
t = 0:1/fs:t_total; % Time vector

% Heart rate (bpm)
HR = 60;  % 60 BPM (1 beat per second)
f_HR = HR / 60; % Convert BPM to Hz
ecg_period = 1 / f_HR; % One heartbeat duration

% ECG waveform parameters (P-QRS-T complex)
A_p = 0.1;  % P wave amplitude
A_q = -0.15; % Q wave amplitude
A_r = 1.2;  % R wave amplitude (main peak)
A_r_neg = -0.5; % Occasional Negative R peak
A_s = -0.4; % S wave amplitude
A_t = 0.3;  % T wave amplitude

% Wave timing (based on ECG characteristics)
T_p = -0.2 * ecg_period;  % P wave occurs before QRS
T_q = -0.05 * ecg_period; % Q wave slightly before R peak
T_r = 0;                  % R wave at center
T_r_neg = 0.02 * ecg_period; % Negative R peak (occasional)
T_s = 0.05 * ecg_period;  % S wave after R
T_t = 0.3 * ecg_period;   % T wave occurs later
```

```
% Wave durations (Gaussian standard deviations)
sigma_p = 0.08 * ecg_period;  % P wave width
sigma_q = 0.02 * ecg_period;  % Q wave width
sigma_r = 0.03 * ecg_period;  % R wave width
sigma_r_neg = 0.03 * ecg_period;  % Negative R wave width
sigma_s = 0.02 * ecg_period;  % S wave width
sigma_t = 0.12 * ecg_period;  % T wave width
% Generate ECG for one heartbeat
t_single = -ecg_period/2:1/fs:ecg_period/2;
ecg_heartbeat = zeros(size(t_single));
for i = 1:length(t_single)
  ecg_heartbeat(i) = ...
    A_p * exp(-((t_single(i) - T_p) .^ 2) / (2 * sigma_p^2)) + ... % P wave
    A_q * exp(-((t_single(i) - T_q) .^ 2) / (2 * sigma_q^2)) + ... % Q wave
    A_r * exp(-((t_single(i) - T_r) .^ 2) / (2 * sigma_r^2)) + ... % R wave
    (mod(i, round(fs * ecg_period * 2)) == 0) * A_r_neg * exp(-((t_single(i) - T_r_neg)
.^ 2) / (2 * sigma_r_neg^2)) + ... % Occasional Negative R peak
    A_s * exp(-((t_single(i) - T_s) .^ 2) / (2 * sigma_s^2)) + ... % S wave
    A_t * exp(-((t_single(i) - T_t) .^ 2) / (2 * sigma_t^2));      % T wave
end
% Generate a full ECG signal with 5 beats
num_beats = 5;  % Exactly 5 R-peaks
final_ecg = [];
final_t = [];
for i = 1:num_beats
  final_ecg = [final_ecg, ecg_heartbeat];  % Append each heartbeat
  final_t = [final_t, t_single + (i-1) * ecg_period];  % Adjust time axis
end
% Add Moderate Noise Components
baseline_wander = 0.05 * sin(2 * pi * 0.5 * final_t);  % Mild baseline drift (0.5Hz)
muscle_noise = 0.02 * randn(size(final_t));          % Smaller high-frequency noise
powerline_noise = 0.01 * sin(2 * pi * 50 * final_t);  % Weaker 50Hz powerline
interference
% Final Noisy ECG Signal
ecg_noisy = final_ecg + baseline_wander + muscle_noise + powerline_noise;
% Save ECG data for further processing
fileID = fopen('ecg_noisy.txt', 'w');
fprintf(fileID, '%f\n', ecg_noisy);
fclose(fileID);
plot(final_t, ecg_noisy, 'r', 'LineWidth', 1.5);
title('ECG Signal');
xlabel('Time (s)');
ylabel('Amplitude (mV)');
grid on;
axis tight;
```

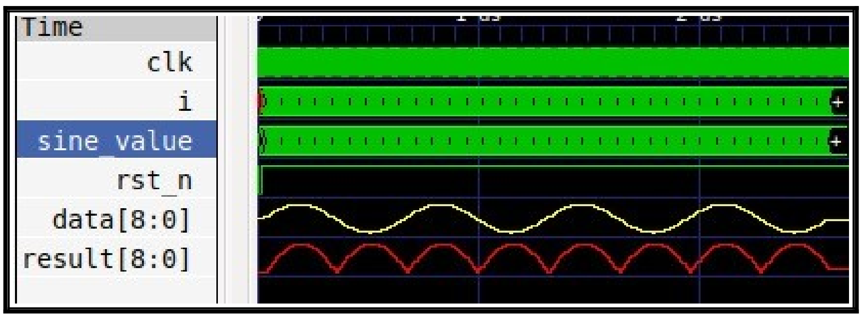
Filter



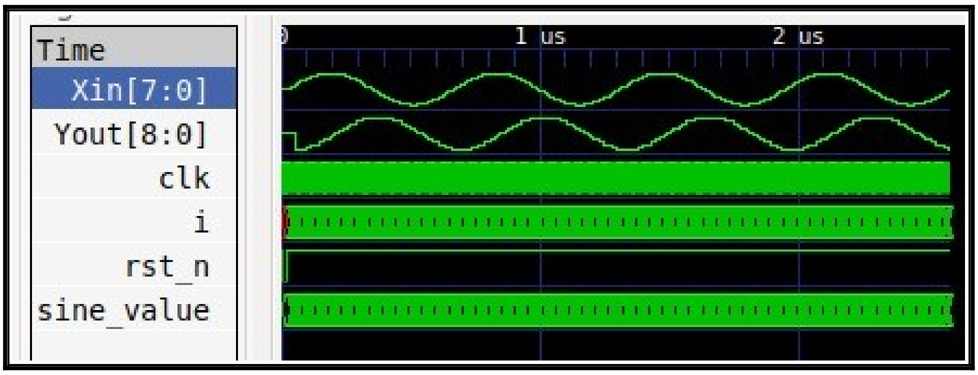
Differentiator



Rectifier



Integrator



Verilog Code

```
module heart_rate_pipeline (
    input clk,
    input rst_n,
    input signed [7:0] Xin,
    output signed [12:0] Yout
);
    // Stage 1: Differentiation (Xin - Xin_prev)
    reg signed [7:0] Xin_Reg [1:0];
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            Xin_Reg[0] <= 8'd0;
            Xin_Reg[1] <= 8'd0;
        end else begin
            Xin_Reg[1] <= Xin_Reg[0];
            Xin_Reg[0] <= Xin;
        end
    end
    wire signed [8:0] diff_out = Xin_Reg[0] - Xin_Reg[1];

    // Stage 2: Rectification (Absolute Value)
    reg signed [8:0] rect_out;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            rect_out <= 9'd0;
        else
            rect_out <= (diff_out[8] == 1'b0) ? diff_out : (~diff_out + 1'b1);
    end
```

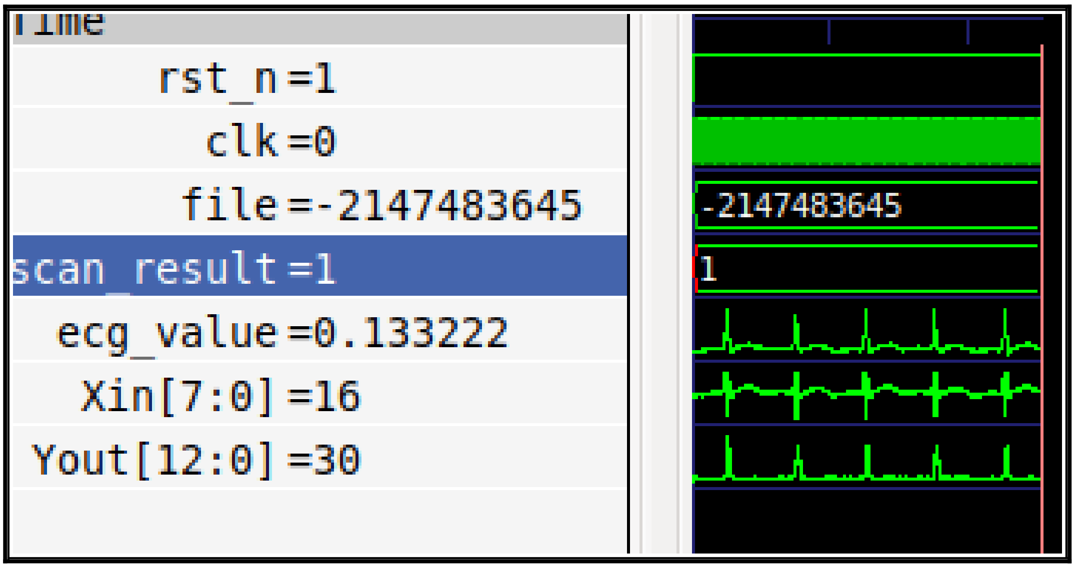
```
    // Stage 3: Integration (16-element shift register + 4-level adder tree)
    reg signed [8:0] Xin_Reg_Inte [15:0];
    integer i; // Declare integer outside the always block
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            for (i = 0; i < 16; i = i + 1)
                Xin_Reg_Inte[i] <= 9'd0;
        end else begin
            for (i = 15; i > 0; i = i - 1)
                Xin_Reg_Inte[i] <= Xin_Reg_Inte[i - 1];
            Xin_Reg_Inte[0] <= rect_out;
        end
    end

    // 4-level adder tree for accumulation
    wire signed [12:0] sum_lvl_2;
    assign sum_lvl_2 = Xin_Reg_Inte[0] + Xin_Reg_Inte[1] + Xin_Reg_Inte[2] +
Xin_Reg_Inte[3] +
                      Xin_Reg_Inte[4] + Xin_Reg_Inte[5] + Xin_Reg_Inte[6] + Xin_Reg_Inte[7] +
                      Xin_Reg_Inte[8] + Xin_Reg_Inte[9] + Xin_Reg_Inte[10] + Xin_Reg_Inte[11] +
                      Xin_Reg_Inte[12] + Xin_Reg_Inte[13] + Xin_Reg_Inte[14] + Xin_Reg_Inte[15];
    assign Yout = sum_lvl_2;
endmodule
```

Testbench for Filter :

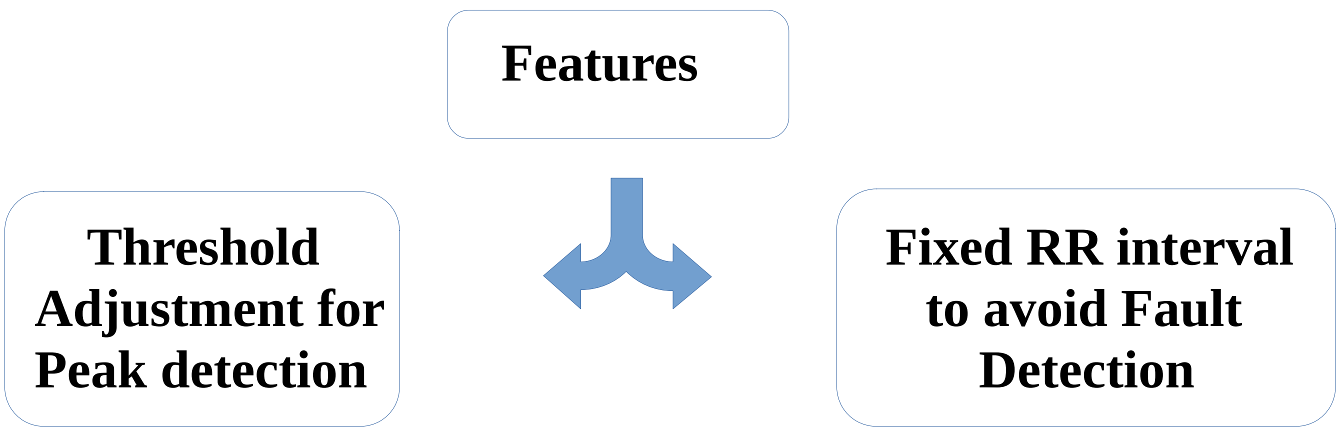
```
 `timescale 1ns/1ps
module heart_rate_pipeline_tb;
    // Inputs
    reg clk;
    reg rst_n;
    reg signed [7:0] Xin;
    // Outputs
    wire signed [12:0] Yout;
    // Instantiate the Unit Under Test (UUT)
    heart_rate_pipeline uut (
        .clk(clk),
        .rst_n(rst_n),
        .Xin(Xin),
        .Yout(Yout)
    );
```

```
  initial begin
        clk = 0;
        forever #5 clk = ~clk; // 10ns period clock
    end
    initial begin
        rst_n = 0;
        #20; // Hold reset for 20ns
        rst_n = 1;
    end
    // Read ECG data from file and apply to Xin
    integer file;
    integer scan_result;
    real ecg_value;
    initial begin
        file = $fopen("ecg_signal.txt", "r");
        if (file == 0) begin
            $display("Error: Could not open file.");
            $finish;
        end
// Initialize Xin
        Xin = 0;
        #30;
        while (!$feof(file)) begin
            scan_result = $fscanf(file, "%f\n", ecg_value); // Read a floating-point value from the file
            if (scan_result != 1) begin
                $display("Error: Failed to read ECG value from file.");
                $finish;
            end
            Xin = $rtoi(ecg_value *127); // Convert to 8-bit fixed-point
            #10;
        end
//Close the file
        $fclose(file);

        // Finish the simulation
        #100;
        $finish;
    end

    // Monitor the outputs
    initial begin
        $monitor("Time: %0t | Xin: %d | Yout: %d", $time, Xin, Yout);
            $dumpfile("heart_rate_pipeline");
            $dumpvars(0, heart_rate_pipeline_tb);
    end

endmodule
```

Filtered Output :



Peak Detector

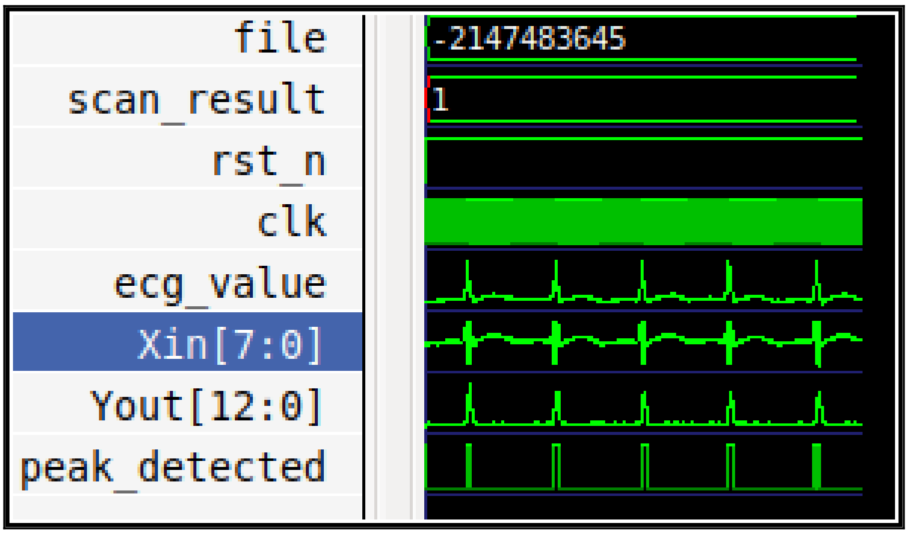


Code :

```
module self_adaptive_threshold (
    input clk,
    input rst_n,
    input signed [12:0] transformed_signal,
    output reg peak_detected);
    reg signed [12:0] signal_buffer [0:255];
    reg [7:0] index;
    integer j;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            index <= 8'd0;
        else if (index < 8'd255)
            index <= index + 8'd1;
        else
            index <= 8'd0;
    end
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            for (j = 0; j < 256; j = j + 1)
                signal_buffer[j] <= 13'd0;
        end else
            signal_buffer[index] <= transformed_signal;
    end
```

```
  // Calculate the maximum value in the buffer
    reg signed [12:0] max_value, temp_max_value;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            max_value <= 13'd0;
        else begin
            temp_max_value = 13'd0;
            for (j = 0; j < 256; j = j + 1)
                if (signal_buffer[j] > temp_max_value)
                    temp_max_value = signal_buffer[j];
            max_value <= temp_max_value;
        end
    end
// Set the dynamic threshold
    reg signed [12:0] threshold;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            threshold <= 13'd0;
        else
            threshold <= (max_value >> 1) + 13'd50; // Use arithmetic right shift
    end
// Detect peaks
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            peak_detected <= 1'b0;
        else
            peak_detected <= (transformed_signal > threshold);
    end
endmodule
```

Output :



RR Interval Calculator :

Average of N intervals is considered as the RR interval of the ECG to enhance accuracy

Code :

```
module RR_Interval_Calculator (
    input clk,
    input rst_n,
    input peak_detected,
    output reg [15:0] avg_interval
  );
    parameter NUM_PEAKS = 4;
    reg output_valid;
    reg [15:0] current_time;
    reg [15:0] sum_intervals;
    reg [3:0] index;
    reg [15:0] current_interval;
    reg [15:0] prev_time;
    reg calibration_done;
    reg prev_peak_detected;
    wire peak_rising_edge;
    assign peak_rising_edge = peak_detected && !prev_peak_detected;
    integer i;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            prev_time = 16'd0;
            current_time = 16'd0;
            index = 4'd0;
            sum_intervals = 16'd0;
            avg_interval = 16'd0;
            calibration_done = 1'b0;
            output_valid = 1'b0;
            prev_peak_detected = 1'b0;
        end
```

```
else begin
            prev_peak_detected = peak_detected;
            if (peak_rising_edge) begin
                if (prev_time == 16'd0) begin
                    prev_time = current_time;
                end
                else begin
                            current_interval = current_time - prev_time;
                    sum_intervals = sum_intervals + (current_time - prev_time);
                    prev_time = current_time;
                    if (!calibration_done) begin
                        if (index == NUM_PEAKS-1) begin
                            avg_interval = sum_intervals / (NUM_PEAKS);
                            calibration_done = 1'b1;
                            output_valid = 1'b1;
                        end else begin
                            index = index + 1;
                        end
                    end
                end
            end
            else begin
                current_time = current_time + 1;
            end
        end
    end
endmodule
```

Testbench :

```
`timescale 1ns/1ps
module RR_interval_calculator_tb;
    reg clk;
    reg rst_n;
    reg signed [7:0] Xin;
    wire signed [12:0] Yout;
    wire peak_detected;
    wire [15:0] avg_RR_interval;
```

```
heart_rate_pipeline uut_pipeline (
        .clk(clk),
        .rst_n(rst_n),
        .Xin(Xin),
        .Yout(Yout)
    );
    self_adaptive_threshold uut_threshold (
        .clk(clk),
        .rst_n(rst_n),
        .transformed_signal(Yout),
        .peak_detected(peak_detected)
    );
    RR_Interval_Calculator uut_rr_calculator (
        .clk(clk),
        .rst_n(rst_n),
        .peak_detected(peak_detected),
        .avg_interval(avg_RR_interval)
    );
    initial begin
        clk = 0;
    end
    always #10 clk = ~clk;
    initial begin
        rst_n = 0;
        #20;
        rst_n = 1;
    end
    integer file;
    integer scan_result;
    real ecg_value;
 initial begin
        file = $fopen("ecg_signal.txt", "r");
        if (file == 0) begin
            $display("Error: Could not open file.");
            $finish;
        end
        Xin = 0;
        #30;
        while (!$feof(file)) begin
            scan_result = $fscanf(file, "%f\n", ecg_value);
            if (scan_result != 1) begin
                $display("Error: Failed to read ECG value from file.");
                $finish;
            end
            Xin = $rtoi(ecg_value * 128);
            #10;
            end
        $fclose(file);
        #100;
        $finish;
    end
```

```
initial begin
        $monitor("Time: %0t | Xin: %d | Yout: %d | Peak Detected: %b | Avg RR Interval: %d",
$time, Xin, Yout, peak_detected, avg_RR_interval);
            $dumpfile("RR_interval_calculator.vcd");
            $dumpvars(0, RR_interval_calculator_tb);
    end

endmodule
```

Output :