Senior Design Project [SDP]

Title : Hear Rate Calculation Based on PPG Signal

GUIDE:

PROF.GDVSANTOSHKUMAR

TEAM

Sikhatapu HariBabu	**21BEC7264**
Sapparapu Devi Sri Prasad	**21BEC7245**

Implementing PPG algorithm in Verilog using the manual coding method

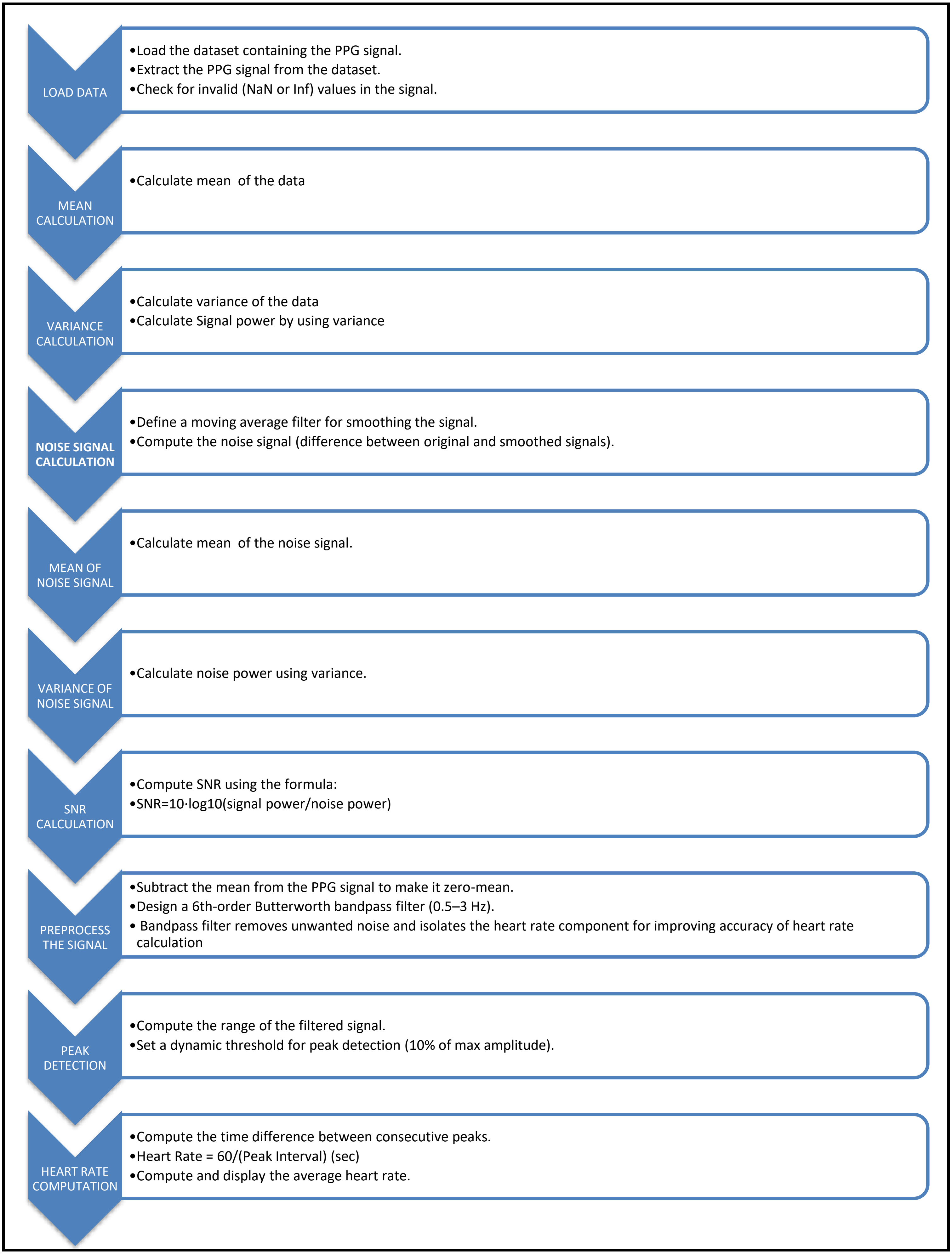
Introduction:

In this project, we aim to translate the Photoplethysmography (PPG) algorithm from its high-level MATLAB implementation to Verilog, a hardware description language suitable for FPGA and ASIC designs. The algorithm has been systematically divided into distinct modules to ensure modularity, ease of debugging, and efficient hardware realization.

Methodology:

Following the divide-and-conquer approach, the MATLAB code has been segmented into different modules.

Block Diagram :


LOAD DATA
•Load the dataset containing the PPG signal.
•Extract the PPG signal from the dataset.
•Check for invalid (NaN or Inf) values in the signal.
MEAN CALCULATION
•Calculate mean of the data
VARIANCE CALCULATION
•Calculate variance of the data
•Calculate Signal power by using variance
NOISE SIGNAL CALCULATION
•Define a moving average filter for smoothing the signal.
•Compute the noise signal (difference between original and smoothed signals).
MEAN OF NOISE SIGNAL
•Calculate mean of the noise signal.
VARIANCE OF NOISE SIGNAL
•Calculate noise power using variance.
SNR CALCULATION
•Compute SNR using the formula:
•SNR=10·log10(signal power/noise power)
PREPROCESS THE SIGNAL
•Subtract the mean from the PPG signal to make it zero-mean.
•Design a 6th-order Butterworth bandpass filter (0.5–3 Hz).
• Bandpass filter removes unwanted noise and isolates the heart rate component for improving accuracy of heart rate calculation
PEAK DETECTION
•Compute the range of the filtered signal.
•Set a dynamic threshold for peak detection (10% of max amplitude).
HEART RATE COMPUTATION
•Compute the time difference between consecutive peaks.
•Heart Rate = 60/(Peak Interval) (sec)
•Compute and display the average heart rate.

MATLAB CODE FOR HEART RATE CALCULATION :

```
clc;
clear all;

% Load the dataset
data = load("E:\CAPSTONE\27132483\sujeto39_PPG_INFO.mat"); % Update with your actual file path

% Extract the PPG signal (replace 'datos_sujeto.senal_PPG' with the correct field name if different)
if isfield(data.datos_sujeto, 'senal_PPG')
   signal = data.datos_sujeto.senal_PPG; % Extract the PPG signal
else
   error('The specified signal variable does not exist in the dataset.');
end

% Check if the signal contains NaN or Inf values (invalid data)
if any(isnan(signal)) || any(isinf(signal))
   error('Signal contains invalid (NaN or Inf) values. Please preprocess your data.');
end

% Step 1: Calculate Signal Power
signal_mean = mean(signal); % Mean of the signal
signal_variance = var(signal); % Variance of the signal
signal_power = signal_variance; % Power of the signal (based on variance)

% Step 2: Calculate Noise Signal
window_size = 5; % Define a moving average window size for smoothing
smoothed_signal = movmean(signal, window_size); % Smooth the signal
noise_signal = signal - smoothed_signal; % Noise is the difference between the original and smoothed signals

% Step 3: Calculate Noise Power
mean_noise = mean(noise_signal); % Mean of noise (not used in SNR calculation)
variance_noise = var(noise_signal); % Variance of noise
noise_power = variance_noise; % Power of noise is the variance of noise

% Step 4: Calculate SNR
snr = 10 * log10(signal_power / noise_power); % SNR formula in dB

% Display results for SNR calculation
disp(['Signal Mean: ', num2str(signal_mean)]);
disp(['Signal Variance: ', num2str(signal_variance)]);
disp(['Signal Power: ', num2str(signal_power)]);
disp(['Mean of Noise: ', num2str(mean_noise)]);
disp(['Variance of Noise: ', num2str(variance_noise)]);
disp(['Noise Power: ', num2str(noise_power)]);
disp(['SNR: ', num2str(snr), ' dB']);
```

```
% Step 5: Check if SNR is in the acceptable range
if snr < 20
    disp('SNR is too low. The signal quality is not acceptable for further processing.');
    return;
else
    disp('SNR is in the acceptable range. Proceeding to Heart Rate calculation...');
end

% Step 6: Heart Rate (BPM) Calculation

% Check for the sampling frequency in the data or set a default
if isfield(data, 'fs')  % If sampling frequency is provided in the data
    fs = data.fs;  % Replace 'fs' with the actual field name if necessary
else
    fs = 50;  % Default to 50 Hz if not found
end

% Create a time vector based on the sampling frequency
t = (0:length(signal)-1)/fs;

% Subtract the mean from the signal to make it zero-mean
ppg_signal_zero_mean = signal - mean(signal);

% Dynamically adjust window size based on signal length or noise level
window_size = max(5, round(length(signal) / 100));  % Use at least a 5-sample window

% Apply a simple moving average filter to smooth the signal
smoothed_ppg_signal = movmean(ppg_signal_zero_mean, window_size);

% Design a Butterworth bandpass filter (6th order)
order = 6;
low_cutoff = 0.5 / (fs / 2);  % Normalize by Nyquist frequency
high_cutoff = 3 / (fs / 2);  % Set higher cutoff to 3 Hz to accommodate for higher heart rate variability
[b, a] = butter(order, [low_cutoff high_cutoff], 'bandpass');

% Apply the filter to the zero-mean signal
filtered_ppg_signal = filtfilt(b, a, ppg_signal_zero_mean);  % Zero-phase filtering

% Check if the signal has enough variation to detect peaks
signal_range = max(filtered_ppg_signal) - min(filtered_ppg_signal);
disp(['Signal range: ', num2str(signal_range)]);

% Dynamically adjust peak detection parameters based on signal characteristics
signal_max = max(filtered_ppg_signal);
min_peak_height = 0.1 * signal_max;  % Set minimum peak height to 10% of the max signal amplitude
```

```
% Remove negative values if they exist in the signal (helps with peak detection)
filtered_ppg_signal(filtered_ppg_signal < 0) = 0;  % Set all negative values to 0

% Detect peaks (local maxima) in the filtered PPG signal
[peaks, locs] = findpeaks(filtered_ppg_signal, 'MinPeakHeight', min_peak_height, 'MinPeakDistance', 0.3 * fs);

% Check if any peaks were detected
if isempty(peaks)
   disp('No peaks detected.');
   return;  % Skip further processing if no peaks are found
end

% Calculate the time intervals between peaks (in seconds)
peak_intervals = diff(locs) / fs;

% Calculate the heart rate in BPM
heart_rate_bpm = 60 ./ peak_intervals;

% Display the average heart rate
average_heart_rate = mean(heart_rate_bpm);
disp(['Average Heart Rate: ', num2str(average_heart_rate), ' BPM']);

% Plot the Original Signal, Smoothed Signal, and Noise
figure;
subplot(3, 1, 1);
plot(t, signal);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3, 1, 2);
plot(t, smoothed_signal);
title('Smoothed Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3, 1, 3);
plot(t, noise_signal);
title('Noise Signal');
xlabel('Time (s)');
ylabel('Amplitude');

% Plot the filtered signal with detected peaks
figure;
plot(t, filtered_ppg_signal);
```

```
hold on;
plot(locs / fs, peaks, 'ro');  % Mark detected peaks with red circles
title('Filtered PPG Signal with Detected Peaks');
xlabel('Time (s)');
ylabel('Amplitude');

% Plot heart rate over time
figure;
plot(locs(2:end) / fs, heart_rate_bpm, '-o');
title('Heart Rate Over Time');
xlabel('Time (s)');
ylabel('Heart Rate (BPM)');
```

DESCRIPTION :

This MATLAB script performs comprehensive processing of a Photoplethysmogram (PPG) signal to evaluate signal quality (via SNR) and calculate heart rate in beats per minute (BPM). The processing is divided into several key stages:

---

1. Data Loading and Validation

- Loads a `.mat` file containing PPG signal data.
- Validates that the signal exists and does not contain invalid values like `NaN` or `Inf`.

---

2. Signal and Noise Power Calculation

- Computes the **mean** and **variance** of the original signal.
- Applies a **moving average filter** to obtain a smoothed version of the signal.
- Calculates the **noise signal** by subtracting the smoothed signal from the original.
- Computes the **mean** and **variance** of the noise signal.
- Calculates **SNR (Signal-to-Noise Ratio)** in decibels using:

  SNR (dB)=10·log10(signal variance/noise variance)

- Displays these metrics and checks whether the SNR exceeds a threshold (20 dB) for acceptable signal quality.

---

3. Preprocessing for Heart Rate Estimation

- Sets a default **sampling frequency** (`fs = 50 Hz`) if not found in the dataset.
- Removes the DC component from the signal by subtracting the mean.
- Applies a **dynamic moving average filter** for initial smoothing.
- Designs and applies a **6th-order Butterworth bandpass filter** to isolate frequencies relevant to heart rate (0.5–3 Hz).

---

4. Peak Detection

- Ensures the signal has sufficient variation for peak detection.
- Uses `findpeaks()` to identify heartbeats by detecting peaks above 10% of the maximum amplitude.
- Eliminates negative values before peak detection to avoid false peaks.

---

5. Heart Rate Calculation

- Calculates **time intervals** between consecutive peaks.
- Computes the **instantaneous heart rate** as:

  Heart Rate (BPM) = 60/Time Interval Between Peaks (s)

- Displays the **average heart rate**.

---

6. Visualization

- Plots:
  - Original, smoothed, and noise signals.
  - Filtered signal with detected peaks marked.
  - Heart rate trend over time.

INPUT DATA FILE CONVERSION :

.mat TO .txt AND FLOATING TO FIXED POINT CONVERSION :

```
% MATLAB Script to Convert .mat File Data to Fixed-Point Integers and Save as .txt



% Clear workspace
clc;
clear;


% Specify the .mat file path
matFilePath = 'signal_data.txt'; % Replace with your .mat file path
txtFilePath = 'output_file.txt'; % Output .txt file path


% Load the .mat file
data = load(matFilePath);


% Extract the PPG signal (replace 'data_field_name' with the actual variable name in your .mat file)
if isfield(data, 'data_field_name') % Replace 'data_field_name' with the variable name in the .mat file
    ppg_signal = data.data_field_name;
else
    error('The specified signal variable does not exist in the .mat file.');
end
```

```
% Check if the signal contains valid data
if any(isnan(ppg_signal)) || any(isinf(ppg_signal))
    error('The PPG signal contains invalid (NaN or Inf) values. Please preprocess your data.');
end


% Convert the floating-point values to fixed-point integers
scaling_factor = 1024; % Define the scaling factor (2^10)
ppg_fixed_point = round(ppg_signal * scaling_factor);


% Save the fixed-point data to a .txt file
writematrix(ppg_fixed_point, txtFilePath);


% Display success message
disp(['Fixed-point data successfully saved to: ', txtFilePath]);
```

DECIMAL TO HEXADECIMAL CONVERSION :

```
% Specify the input and output file names

input_file = 'output_file.txt';  % Your input file with decimal values
output_file = 'output_file_hex.txt';  % The output file where hex values will be stored


% Open the input file for reading
fid_input = fopen(input_file, 'r');
if fid_input == -1
    error('Could not open input file');
end


% Open the output file for writing
fid_output = fopen(output_file, 'w');
if fid_output == -1
    fclose(fid_input);
    error('Could not open output file');
end


% Read each line of the input file, convert to hexadecimal, and write to the output file
tline = fgets(fid_input);  % Read the first line
while ischar(tline)
    % Remove any leading/trailing whitespaces or newline characters
    tline = strtrim(tline);

    % Convert the decimal value to a hexadecimal value (16-bit wide)
    decimal_value = str2double(tline);  % Convert the line to a number
    hex_value = sprintf('%04X', round(decimal_value));  % Convert to 4-digit hexadecimal

    % Write the hex value to the output file
    fprintf(fid_output, '%s\n', hex_value);

    % Read the next line
    tline = fgets(fid_input);
end


% Close the files
fclose(fid_input);
fclose(fid_output);
```

```
disp('Conversion complete. Hexadecimal data written to output file.');
```

SNR CALCULATION

DATA LOADER MODULE :

Module Name: `load_data`

Functionality:

The `load_data` module is designed to load and store PPG (Photoplethysmogram) signal data from a memory file into internal memory and provide access to it based on an address input. It acts as a ROM-like data loader for the PPG IP Core.

Key Features:

- Loads hexadecimal PPG signal data from a file into internal memory at the start of simulation.
- Allows data retrieval through a specified address input.
- Outputs a flag indicating when the data loading process is complete.

CODE :

```
module load_data #(parameter DATA_WIDTH = 16, MEMORY_DEPTH = 5968) (
  input clk,                // Clock signal
  input reset,              // Reset signal
  input [12:0] read_address,     // Address to read from memory (13 bits for 5968 locations)
  output reg [DATA_WIDTH-1:0] data_out, // Data output for the given address
  output reg loaded           // Flag indicating data has been loaded
);
  // Internal memory array
  reg [DATA_WIDTH-1:0] memory [0:MEMORY_DEPTH-1];
  // Load the data during initialization
  initial begin
    loaded = 0;
    $readmemh("C:/Xilinx/output_file_hex.txt", memory); // Load data from the file (ensure it's in hexadecimal format)
    loaded = 1; // Indicate data has been successfully loaded
  end
  // Provide data based on read address
  always @(posedge clk or posedge reset) begin
  if (reset) begin
    data_out <= 0; // Clear data_out on reset
  end else if (loaded) begin
    if (read_address < MEMORY_DEPTH) begin
      data_out <= memory[read_address]; // Read valid data
    end else begin
```

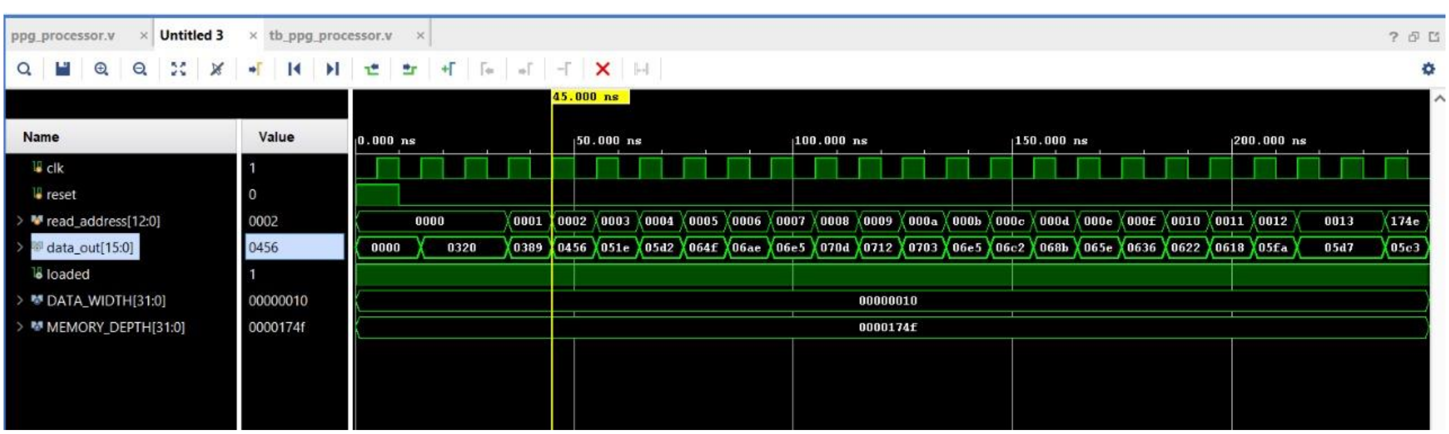
```
      data_out <= 0; // Handle out-of-bounds address
      $display("Error: Read address %d is out of bounds!", read_address);
    end
  end else begin
    $display("Memory not loaded yet!");
  end
end
endmodule
```

TESTBENCH FOR LOAD DATA :

```
module tb_load_data;
  parameter DATA_WIDTH = 16;
  parameter MEMORY_DEPTH = 5968; // Match the file size

  reg clk, reset;
  reg [12:0] read_address;       // Address to access memory (13 bits for 5968 values)
  wire [DATA_WIDTH-1:0] data_out; // Data output
  wire loaded;                   // Data load status

  // Instantiate the load_data module
  load_data #(DATA_WIDTH, MEMORY_DEPTH) uut (
    .clk(clk),
    .reset(reset),
    .read_address(read_address),
    .data_out(data_out),
    .loaded(loaded)
  );

  // Generate clock signal
  initial begin
    clk = 0;
    forever #5 clk = ~clk; // Clock period = 10 time units
  end

  // Test procedure
  initial begin
    // Monitor signals for debugging
    $monitor("Time: %0t | clk: %b | reset: %b | read_address: %d | data_out: %h | loaded: %b",
         $time, clk, reset, read_address, data_out, loaded);

    // Initialize testbench
    reset = 1;
    read_address = 0;

    #10 reset = 0; // Deassert reset

    // Wait until data is loaded
    @(posedge clk);
```

```
        wait(loaded == 1);
        $display("Data loaded successfully!");

        #10; // Wait for memory to load

        // Read and display the first 20 values for verification
        for (integer i = 0; i < 20; i = i + 1) begin
            read_address = i; // Set read address
            #10; // Wait for data_out to stabilize
            if (data_out === 16'hxxxx) begin
                $display("Error: Uninitialized memory read at address %0d!", i);
            end else begin
                $display("Memory[%0d] = %h", i, data_out); // Display in hexadecimal
            end
        end

        // Wait for a brief time
        #10;

        // Read and display the last value in memory
        read_address = MEMORY_DEPTH - 1; // Last address
        #10; // Wait for data_out to stabilize
        if (data_out === 16'hxxxx) begin
            $display("Error: Uninitialized memory read at last address!");
        end else begin
            $display("Last Memory Value [%0d]: %h", MEMORY_DEPTH-1, data_out); // Display in hexadecimal
        end

        // End simulation
        $display("Testbench completed successfully.");
        $finish;
    end
endmodule
```

WAVEFORM :



CALCULATING MEAN OF THE DATA :

Module Name: `calculate_mean`

Functionality:

The `calculate_mean` module calculates the arithmetic mean (average) of a set of PPG signal data samples stored in memory. It leverages the `load_data` module to fetch the input data and processes it sequentially to compute the mean.

Key Features:

- Automatically reads a fixed number of samples (`MEMORY_DEPTH`) from memory.
- Accumulates the total sum of all samples and computes the mean.
- Signals when the mean calculation is complete via the `done` flag.

CODE :

```
module calculate_mean #(parameter DATA_WIDTH = 16, MEMORY_DEPTH = 5968) (
   input clk,                    // Clock signal
   input reset                   // Reset signal
);
   // Internal Signals
   wire [DATA_WIDTH-1:0] data_out; // Data output from the load_data module
   wire loaded;                  // Indicates when data is loaded
   reg [12:0] read_address;      // Address counter (13 bits for 5967 locations)
   reg [DATA_WIDTH+12:0] sum;    // Accumulator for summing the data (DATA_WIDTH + log2(MEMORY_DEPTH))
   reg [12:0] sample_count;      // Tracks the number of samples processed
   reg done;                     // Flag to indicate the mean calculation is complete
   reg [DATA_WIDTH-1:0] mean;    // Mean value to display

   // Instantiate the `load_data` module
   load_data #(
      .DATA_WIDTH(DATA_WIDTH),
      .MEMORY_DEPTH(MEMORY_DEPTH)
   ) memory_module (
      .clk(clk),
      .reset(reset),
      .read_address(read_address),
      .data_out(data_out),
      .loaded(loaded)
   );

   // Mean Calculation Logic
   always @(posedge clk or posedge reset) begin
      if (reset) beginS
         sum <= 0;
```

```
      read_address <= 0;
      sample_count <= 0;
      done <= 0;
      mean <= 0;
    end else if (loaded && !done) begin
      if (read_address < MEMORY_DEPTH) begin
        sum <= sum + data_out;        // Accumulate data
        read_address <= read_address + 1; // Increment address
        sample_count <= sample_count + 1; // Increment sample count
      end else if (read_address == MEMORY_DEPTH) begin
        mean <= sum / sample_count;   // Calculate mean after processing all addresses
        done <= 1;                    // Indicate processing is complete
        //$display("Mean Value: %d", mean); // Display the mean value
      end
    end
  end
endmodule
```

TESTBENCH CODE :

```
module tb_calculate_mean();
  reg clk;
  reg reset;

  // Instantiate the calculate_mean module
  calculate_mean uut (
    .clk(clk),
    .reset(reset)
  );

  // Clock Generation
  always #5 clk = ~clk;

  initial begin
    // Initialize signals
    clk = 0;
    reset = 1;

    #10 reset = 0; // Release reset

    // Wait for mean calculation to complete
    wait (uut.done); // Wait for the 'done' signal in the calculate_mean module
    $display("Calculation Complete. Mean Value: %d", uut.mean);
    $stop; // End simulation
  end
```

```
// Monitor values during simulation
initial begin
  $monitor("Time: %0t | Address: %d | Data Out: %d | Sum: %d | Mean: %d | Done: %b",
        $time, uut.read_address, uut.data_out, uut.sum, uut.mean, uut.done);
end
```

RESULT :

```
Time: 59465000 | Address: 5946 | Data Out:  905 | Sum:   7825525 | Mean:     0 | Done: 0
Time: 59475000 | Address: 5947 | Data Out:  880 | Sum:   7826430 | Mean:     0 | Done: 0
Time: 59485000 | Address: 5948 | Data Out:  910 | Sum:   7827310 | Mean:     0 | Done: 0
Time: 59495000 | Address: 5949 | Data Out: 1015 | Sum:   7828220 | Mean:     0 | Done: 0
Time: 59505000 | Address: 5950 | Data Out: 1195 | Sum:   7829235 | Mean:     0 | Done: 0
Time: 59515000 | Address: 5951 | Data Out: 1360 | Sum:   7830430 | Mean:     0 | Done: 0
Time: 59525000 | Address: 5952 | Data Out: 1505 | Sum:   7831790 | Mean:     0 | Done: 0
Time: 59535000 | Address: 5953 | Data Out: 1605 | Sum:   7833295 | Mean:     0 | Done: 0
Time: 59545000 | Address: 5954 | Data Out: 1690 | Sum:   7834900 | Mean:     0 | Done: 0
Time: 59555000 | Address: 5955 | Data Out: 1740 | Sum:   7836590 | Mean:     0 | Done: 0
Time: 59565000 | Address: 5956 | Data Out: 1755 | Sum:   7838330 | Mean:     0 | Done: 0
Time: 59575000 | Address: 5957 | Data Out: 1765 | Sum:   7840085 | Mean:     0 | Done: 0
Time: 59585000 | Address: 5958 | Data Out: 1765 | Sum:   7841850 | Mean:     0 | Done: 0
Time: 59595000 | Address: 5959 | Data Out: 1750 | Sum:   7843615 | Mean:     0 | Done: 0
Time: 59605000 | Address: 5960 | Data Out: 1710 | Sum:   7845365 | Mean:     0 | Done: 0
Time: 59615000 | Address: 5961 | Data Out: 1665 | Sum:   7847075 | Mean:     0 | Done: 0
Time: 59625000 | Address: 5962 | Data Out: 1615 | Sum:   7848740 | Mean:     0 | Done: 0
Time: 59635000 | Address: 5963 | Data Out: 1580 | Sum:   7850355 | Mean:     0 | Done: 0
Time: 59645000 | Address: 5964 | Data Out: 1555 | Sum:   7851935 | Mean:     0 | Done: 0
Time: 59655000 | Address: 5965 | Data Out: 1530 | Sum:   7853490 | Mean:     0 | Done: 0
Time: 59665000 | Address: 5966 | Data Out: 1500 | Sum:   7855020 | Mean:     0 | Done: 0
Time: 59675000 | Address: 5967 | Data Out: 1475 | Sum:   7856520 | Mean:     0 | Done: 0
Time: 59685000 | Address: 5968 | Data Out:    x | Sum:   7857995 | Mean:     0 | Done: 0
Error: Read address 5968 is out of bounds!
Calculation Complete. Mean Value:  1316
```

CALCULATING VARIANCE :

Module Name: `calculate_variance`

Functionality:

The `calculate_variance` module computes the **statistical variance** of PPG signal data stored in memory. It does so by:

1. Calculating the mean of the dataset using the `calculate_mean` module.
2. Computing the squared deviations from the mean and accumulating them.
3. Dividing the result by `(N-1)` to obtain the variance.

Key Features:

- Fully automated pipeline: loads data, calculates mean, and then variance.
- Uses modular design by instantiating `calculate_mean` and `load_data`.
- Outputs both variance and a completion signal.

CODE :

```
module calculate_variance #(parameter DATA_WIDTH = 16, MEMORY_DEPTH = 5968) (
   input clk,
   input reset
);
   // Internal signals
   wire [DATA_WIDTH-1:0] mean;        // Mean value from calculate_mean
   wire [DATA_WIDTH-1:0] data_out;    // Data output from load_data
   wire loaded;                       // Indicates data is loaded
   wire done_mean;                    // Indicates mean calculation is complete
   reg [12:0] read_address;           // Address counter
   reg [DATA_WIDTH+24:0] squared_sum;  // Sum of squared differences
   reg [DATA_WIDTH+12:0] variance;     // Final variance result
   reg [12:0] sample_count;           // Number of samples
   reg done_variance;                 // Indicates variance calculation is complete

   // Instantiate calculate_mean module
   calculate_mean #(
      .DATA_WIDTH(DATA_WIDTH),
      .MEMORY_DEPTH(MEMORY_DEPTH)
   ) mean_module (
      .clk(clk),
      .reset(reset),
      .mean(mean),                 // Output: Mean value
      .done(done_mean)             // Output: Completion flag
   );
```

```
    // Instantiate load_data module
    load_data #(
        .DATA_WIDTH(DATA_WIDTH),
        .MEMORY_DEPTH(MEMORY_DEPTH)
    ) data_module (
        .clk(clk),
        .reset(reset),
        .read_address(read_address),    // Input: Address to read
        .data_out(data_out),            // Output: Data at the address
        .loaded(loaded)                 // Output: Data load status
    );

    // Variance Calculation Logic
    always @(posedge clk or posedge reset) begin
    if (reset) begin
        squared_sum <= 0;
        read_address <= 0;
        sample_count <= 0;
        done_variance <= 0;
        variance <= 0;
    end else if (loaded && done_mean && !done_variance) begin
        if (read_address < MEMORY_DEPTH) begin
            squared_sum <= squared_sum + ((data_out - mean) * (data_out - mean));
            read_address <= read_address + 1;
        end else if (read_address == MEMORY_DEPTH) begin
            variance <= (MEMORY_DEPTH > 1) ? (squared_sum / (MEMORY_DEPTH-1)) : 0;
            done_variance <= 1;
        end
    end
end
endmodule
```

TESTBENCH CODE :

```
`timescale 1ns / 1ps
module tb_calculate_variance();
    // Inputs
    reg clk;
    reg reset;

    // Instantiate the calculate_variance module
    calculate_variance uut (
        .clk(clk),
        .reset(reset)
    );
    // Clock Generation
    always #5 clk = ~clk;
```

```
  initial begin
    // Initialize signals
    clk = 0;
    reset = 1;
    #5 reset = 0; // Release reset
    // Wait for variance calculation to complete
    wait (uut.done_variance); // Wait for the 'done_variance' signal in the calculate_variance module
    $display("Calculation Complete. Variance Value: %d", uut.variance);
    $stop; // End simulation
  end

  // Monitor values during simulation
  initial begin
    $monitor("Time: %0t | Address: %d | Data Out: %d | Mean: %d | Squared Sum: %d | Variance: %d | Done: %b",
         $time, uut.read_address, uut.data_out, uut.mean, uut.squared_sum, uut.variance, uut.done_variance);
  end
endmodule
```

RESULT :

```
Time: 119195000 | Address: 5951 | Data Out:  1360 | Mean:  1316 | Squared Sum:     523442446 | Variance:          0 | Done: 0
Time: 119205000 | Address: 5952 | Data Out:  1505 | Mean:  1316 | Squared Sum:     523444382 | Variance:          0 | Done: 0
Time: 119215000 | Address: 5953 | Data Out:  1605 | Mean:  1316 | Squared Sum:     523480103 | Variance:          0 | Done: 0
Time: 119225000 | Address: 5954 | Data Out:  1690 | Mean:  1316 | Squared Sum:     523563624 | Variance:          0 | Done: 0
Time: 119235000 | Address: 5955 | Data Out:  1740 | Mean:  1316 | Squared Sum:     523703500 | Variance:          0 | Done: 0
Time: 119245000 | Address: 5956 | Data Out:  1755 | Mean:  1316 | Squared Sum:     523883276 | Variance:          0 | Done: 0
Time: 119255000 | Address: 5957 | Data Out:  1765 | Mean:  1316 | Squared Sum:     524075997 | Variance:          0 | Done: 0
Time: 119265000 | Address: 5958 | Data Out:  1765 | Mean:  1316 | Squared Sum:     524277598 | Variance:          0 | Done: 0
Time: 119275000 | Address: 5959 | Data Out:  1750 | Mean:  1316 | Squared Sum:     524479199 | Variance:          0 | Done: 0
Time: 119285000 | Address: 5960 | Data Out:  1710 | Mean:  1316 | Squared Sum:     524667555 | Variance:          0 | Done: 0
Time: 119295000 | Address: 5961 | Data Out:  1665 | Mean:  1316 | Squared Sum:     524822791 | Variance:          0 | Done: 0
Time: 119305000 | Address: 5962 | Data Out:  1615 | Mean:  1316 | Squared Sum:     524944592 | Variance:          0 | Done: 0
Time: 119315000 | Address: 5963 | Data Out:  1580 | Mean:  1316 | Squared Sum:     525033993 | Variance:          0 | Done: 0
Time: 119325000 | Address: 5964 | Data Out:  1555 | Mean:  1316 | Squared Sum:     525103689 | Variance:          0 | Done: 0
Time: 119335000 | Address: 5965 | Data Out:  1530 | Mean:  1316 | Squared Sum:     525160810 | Variance:          0 | Done: 0
Time: 119345000 | Address: 5966 | Data Out:  1500 | Mean:  1316 | Squared Sum:     525206606 | Variance:          0 | Done: 0
Time: 119355000 | Address: 5967 | Data Out:  1475 | Mean:  1316 | Squared Sum:     525240462 | Variance:          0 | Done: 0
Time: 119365000 | Address: 5968 | Data Out:     x | Mean:  1316 | Squared Sum:     525265743 | Variance:          0 | Done: 0
Calculation Complete. Variance Value:      88028
```

MOVING AVERAGE FILTER :

Module Name: `moving_average_filter`

Functionality:

The `moving_average_filter` module implements a real-time **moving average filter** to smooth out high-frequency noise in the input signal. It uses a fixed-size sliding window buffer to compute the average of the most recent samples, effectively acting as a **low-pass filter**.

Key Features:

- Performs real-time signal smoothing with configurable window size.

- Dynamically updates buffer and sum for efficient computation.
- Handles startup phase when the buffer isn't full.
- Produces a smoothed output signal based on current and past inputs.

CODE :

```
module moving_average_filter #(parameter DATA_WIDTH = 16, WINDOW_SIZE = 5) (
   input wire clk,
   input wire reset,
   input wire [DATA_WIDTH-1:0] data_out,  // Input signal
   output reg [DATA_WIDTH-1:0] smoothed_signal
);

   reg [DATA_WIDTH-1:0] buffer [0:WINDOW_SIZE-1];
   reg [DATA_WIDTH+12:0] sum;
   reg [2:0] index;
   reg [2:0] count;
   integer i;

   // Declare old_value properly here
   reg [DATA_WIDTH-1:0] old_value;

   always @(posedge clk or posedge reset) begin
      if (reset) begin
         sum <= 0;
         index <= 0;
         count <= 0;
         smoothed_signal <= 0;
         old_value <= 0;
         for (i = 0; i < WINDOW_SIZE; i = i + 1) begin
            buffer[i] <= 0;
         end
      end else begin
         // Store old value BEFORE overwriting
         old_value <= buffer[index];

         // Update buffer
         buffer[index] <= data_out;

         // Update sum properly
         if (count >= WINDOW_SIZE) begin
            sum <= sum - old_value + data_out;
         end else begin
            sum <= sum + data_out;
            count <= count + 1;
         end
```

```
      // Increment index
      index <= (index == WINDOW_SIZE-1) ? 0 : index + 1;

      // Calculate smoothed output
      if (count >= WINDOW_SIZE) begin
        smoothed_signal <= (sum + (WINDOW_SIZE/2)) / WINDOW_SIZE;
      end else begin
        smoothed_signal <= (sum + (count/2)) / count;
      end
    end
  end
endmodule
```

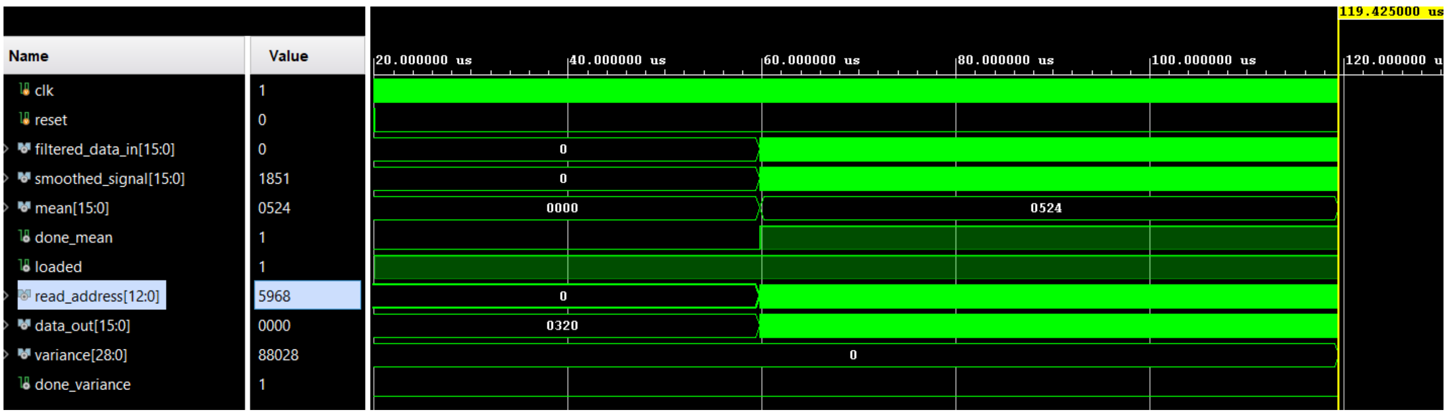
TESTBENCH CODE :

```
`timescale 1ns / 1ps

module tb_moving_average_filter();

  // Inputs
  reg clk;
  reg reset;

  // Outputs
  wire [15:0] smoothed_signal;
  wire [15:0] mean;
  wire done_mean;
  wire loaded;
  wire [12:0] read_address;
  wire [15:0] data_out;
  wire [28:0] variance;
  wire done_variance;

  // Instantiate moving_average_filter module
  moving_average_filter uut (
    .clk(clk),
    .reset(reset),
    .data_out(data_out), // Corrected input
    .smoothed_signal(smoothed_signal)
  );

  // Instantiate calculate_variance module
  calculate_variance var_inst (
    .clk(clk),
    .reset(reset),
    .variance(variance),
    .done_variance(done_variance),
```

```
    .read_address(read_address),
    .data_out(data_out)   // Corrected - removed invalid ports
);

// Instantiate calculate_mean module
calculate_mean mean_inst (
    .clk(clk),
    .reset(reset),
    .mean(mean),
    .done(done_mean)
);

// Instantiate load_data module
load_data data_loader (
    .clk(clk),
    .reset(reset),
    .read_address(read_address),
    .data_out(data_out),
    .loaded(loaded)
);

// Clock Generation
always #5 clk = ~clk;

initial begin
    // Initialize signals
    clk = 0;
    reset = 1;

    #50 reset = 0; // Hold reset longer to allow proper initialization

    // Wait for the data to load
    wait (loaded);
    $display("Data loaded successfully!");

    // Wait for mean calculation
    wait (done_mean);
    $display("Mean calculation completed: %d", mean);

    // Wait for variance calculation to complete
    wait (done_variance);
    $display("Variance calculation completed: %d", variance);

    $stop; // End simulation
end
```

```
    // Monitor values during simulation
    initial begin
        $monitor("Time: %0t | Smoothed: %d | Mean: %d | Variance: %d | Done_Mean: %b | Done_Variance: %b |
Read_Address: %d | Data_Out: %d",
                 $time, smoothed_signal, mean, variance, done_mean, done_variance, read_address, data_out);
    end

endmodule
```

RESULT :




```
Time: 119215000 | Smoothed:   943 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5948 | Data_Out:   910
Time: 119225000 | Smoothed:   924 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5949 | Data_Out:  1015
Time: 119235000 | Smoothed:   914 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5950 | Data_Out:  1195
Time: 119245000 | Smoothed:   927 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5951 | Data_Out:  1360
Time: 119255000 | Smoothed:   981 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5952 | Data_Out:  1505
Time: 119265000 | Smoothed:  1072 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5953 | Data_Out:  1605
Time: 119275000 | Smoothed:  1197 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5954 | Data_Out:  1690
Time: 119285000 | Smoothed:  1336 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5955 | Data_Out:  1740
Time: 119295000 | Smoothed:  1471 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5956 | Data_Out:  1755
Time: 119305000 | Smoothed:  1580 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5957 | Data_Out:  1765
Time: 119315000 | Smoothed:  1659 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5958 | Data_Out:  1765
Time: 119325000 | Smoothed:  1711 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5959 | Data_Out:  1750
Time: 119335000 | Smoothed:  1743 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5960 | Data_Out:  1710
Time: 119345000 | Smoothed:  1755 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5961 | Data_Out:  1665
Time: 119355000 | Smoothed:  1749 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5962 | Data_Out:  1615
Time: 119365000 | Smoothed:  1731 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5963 | Data_Out:  1580
Time: 119375000 | Smoothed:  1701 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5964 | Data_Out:  1555
Time: 119385000 | Smoothed:  1664 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5965 | Data_Out:  1530
Time: 119395000 | Smoothed:  1625 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5966 | Data_Out:  1500
Time: 119405000 | Smoothed:  1589 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5967 | Data_Out:  1475
Time: 119415000 | Smoothed:  1556 | Mean:  1316 | Variance:          0 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5968 | Data_Out:     x
Variance calculation completed:      88028
```

NOISE SIGNAL :

Module Name: `noise_signal`

Functionality:

The `noise_signal` module calculates the **noise component** of an input signal by subtracting its smoothed (filtered) version from the original. This is particularly useful for analyzing the high-frequency content (noise) removed by the filtering process.

Key Features:

- Real-time noise extraction from an input signal.
- Operates synchronously with the system clock.
- Designed to work directly with the output of a moving average or any smoothing filter.

CODE :

```
module noise_signal #(parameter DATA_WIDTH = 16) (
   input wire clk,
   input wire reset,
   input wire [DATA_WIDTH-1:0] data_out,        // Original input signal
   input wire [DATA_WIDTH-1:0] smoothed_signal,  // Filtered output from moving_average_filter
   output reg [DATA_WIDTH-1:0] noise_signal      // Noise component
);

   always @(posedge clk or posedge reset) begin
      if (reset) begin
         noise_signal <= 0;
      end else begin
         noise_signal <= data_out - smoothed_signal; // Compute noise component
      end
   end
endmodule
```

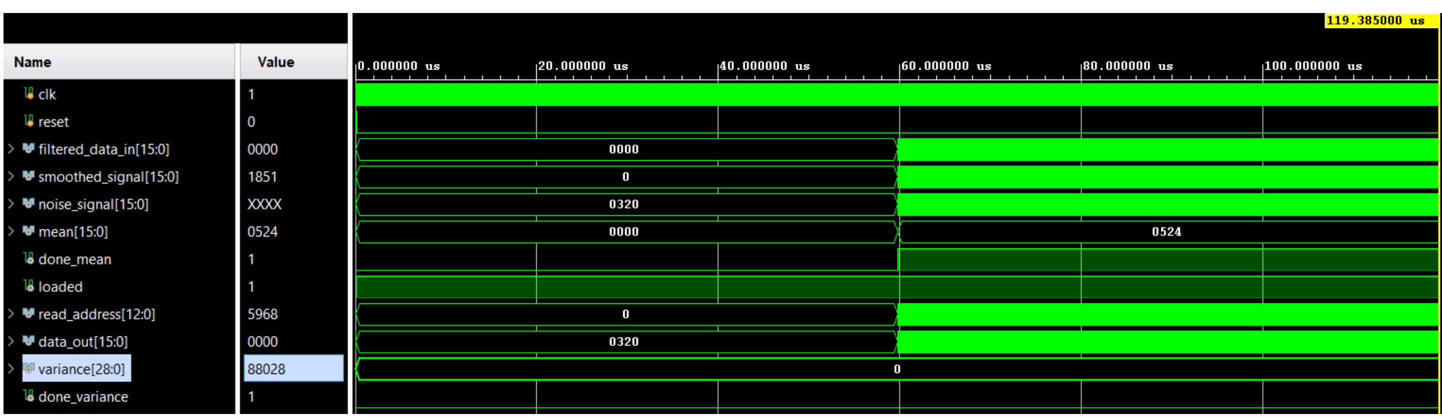
TESTBENCH :

```
`timescale 1ns / 1ps

module tb_noise_signal();

   // Inputs
   reg clk;
   reg reset;

   // Outputs
   wire [15:0] smoothed_signal;
   wire [15:0] noise_signal;
   wire [15:0] mean;
   wire done_mean;
   wire loaded;
   wire [12:0] read_address;
   wire [15:0] data_out;
   wire [28:0] variance;
   wire done_variance;

   // Instantiate load_data module (First module to provide input data)
```

```
load_data data_loader (
    .clk(clk),
    .reset(reset),
    .read_address(read_address),
    .data_out(data_out),
    .loaded(loaded)
);

// Instantiate moving_average_filter module
moving_average_filter uut_filter (
    .clk(clk),
    .reset(reset),
    .data_out(data_out),        // Now correctly taking data_out from load_data
    .smoothed_signal(smoothed_signal)
);

// Instantiate noise_signal module
noise_signal uut_noise (
    .clk(clk),
    .reset(reset),
    .data_out(data_out),        // Corrected input from load_data
    .smoothed_signal(smoothed_signal),
    .noise_signal(noise_signal)
);

// Instantiate calculate_variance module
calculate_variance var_inst (
    .clk(clk),
    .reset(reset),
    .variance(variance),
    .done_variance(done_variance),
    .read_address(read_address),
    .data_out(data_out)
);

// Instantiate calculate_mean module
calculate_mean mean_inst (
    .clk(clk),
    .reset(reset),
    .mean(mean),
    .done(done_mean)
);

// Clock Generation
always #5 clk = ~clk;
```

```
  initial begin
    // Initialize signals
    clk = 0;
    reset = 1;

    #10 reset = 0; // Release reset

    // Wait for data to load
    wait(loaded);
    $display("Data loaded successfully!");

    // Wait for variance calculation to complete
    wait(done_variance);
    $display("done_variance = %d", done_variance);
    $display("Calculation Complete. Variance Value: %d", variance);

    $stop; // End simulation
  end

  // Monitor values during simulation
  initial begin
    $monitor("Time: %0t | Data_Out: %d | Smoothed: %d | Noise: %d | Mean: %d | Variance: %d | Done_Mean: %b | Done_Variance: %b | Read_Address: %d",
             $time, data_out, smoothed_signal, noise_signal, mean, variance, done_mean, done_variance, read_address);
  end

endmodule
```

RESULT :



```
Time: 119215000 | Data_Out:  1505 | filtered_data:  1505 | Smoothed:  1166 | Noise:    243 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5952
Time: 119225000 | Data_Out:  1605 | filtered_data:  1605 | Smoothed:  1253 | Noise:    339 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5953
Time: 119235000 | Data_Out:  1690 | filtered_data:  1690 | Smoothed:  1373 | Noise:    352 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5954
Time: 119245000 | Data_Out:  1740 | filtered_data:  1740 | Smoothed:  1518 | Noise:    317 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5955
Time: 119255000 | Data_Out:  1755 | filtered_data:  1755 | Smoothed:  1674 | Noise:    222 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5956
Time: 119265000 | Data_Out:  1765 | filtered_data:  1765 | Smoothed:  1819 | Noise:     81 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5957
Time: 119275000 | Data_Out:  1765 | filtered_data:  1765 | Smoothed:  1931 | Noise:    -54 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5958
Time: 119285000 | Data_Out:  1750 | filtered_data:  1750 | Smoothed:  2012 | Noise:   -166 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5959
Time: 119295000 | Data_Out:  1710 | filtered_data:  1710 | Smoothed:  2064 | Noise:   -262 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5960
Time: 119305000 | Data_Out:  1665 | filtered_data:  1665 | Smoothed:  2093 | Noise:   -354 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5961
Time: 119315000 | Data_Out:  1615 | filtered_data:  1615 | Smoothed:  2097 | Noise:   -428 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5962
Time: 119325000 | Data_Out:  1580 | filtered_data:  1580 | Smoothed:  2082 | Noise:   -482 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5963
Time: 119335000 | Data_Out:  1555 | filtered_data:  1555 | Smoothed:  2054 | Noise:   -502 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5964
Time: 119345000 | Data_Out:  1530 | filtered_data:  1530 | Smoothed:  2017 | Noise:   -499 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5965
Time: 119355000 | Data_Out:  1500 | filtered_data:  1500 | Smoothed:  1975 | Noise:   -487 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5966
Time: 119365000 | Data_Out:  1475 | filtered_data:  1475 | Smoothed:  1931 | Noise:   -475 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5967
Time: 119375000 | Data_Out:     x | filtered_data:     x | Smoothed:  1889 | Noise:   -456 | Done_Mean: 1 | Done_Variance: 0 | Read_Address: 5968
done_variance = 1
Calculation Complete. Variance Value:     88028
```

NOISE MEAN :

Module Name: `calculate_noise_mean`

Functionality:

The `calculate_noise_mean` module computes the **mean (average) value of a noise signal** over a fixed number of samples. This is a critical step in analyzing the statistical characteristics of noise in biomedical or signal processing systems, particularly in **SNR (Signal-to-Noise Ratio)** computation.

Key Features:

- Accumulates signed noise signal values over `MEMORY_DEPTH` samples.
- Computes a **signed mean** value after the full dataset is processed.
- Controlled by a `valid_noise` enable signal, allowing synchronized noise sampling.

CODE :

```
module calculate_noise_mean #(parameter DATA_WIDTH = 16, MEMORY_DEPTH = 5968)(
   input wire clk,
   input wire reset,
   input wire valid_noise,
   input wire signed [DATA_WIDTH-1:0] noise_signal,
   output reg signed [DATA_WIDTH+13:0] noise_sum,  // Signed accumulator
   output reg signed [DATA_WIDTH-1:0] noise_mean,  // Final signed mean
   output reg done_noise_mean
);

   reg [12:0] sample_count;

   always @(posedge clk or posedge reset) begin
      if (reset) begin
         noise_sum <= 0;
         noise_mean <= 0;
```

```
      sample_count <= 0;
      done_noise_mean <= 0;
    end else if (!done_noise_mean && valid_noise) begin
      if (sample_count < MEMORY_DEPTH) begin
        noise_sum <= noise_sum + noise_signal;
        sample_count <= sample_count + 1;
      end else if (sample_count == MEMORY_DEPTH) begin
        noise_mean <= noise_sum / MEMORY_DEPTH;
        done_noise_mean <= 1;
      end
    end
  end

endmodule
```

TESTBENCH :

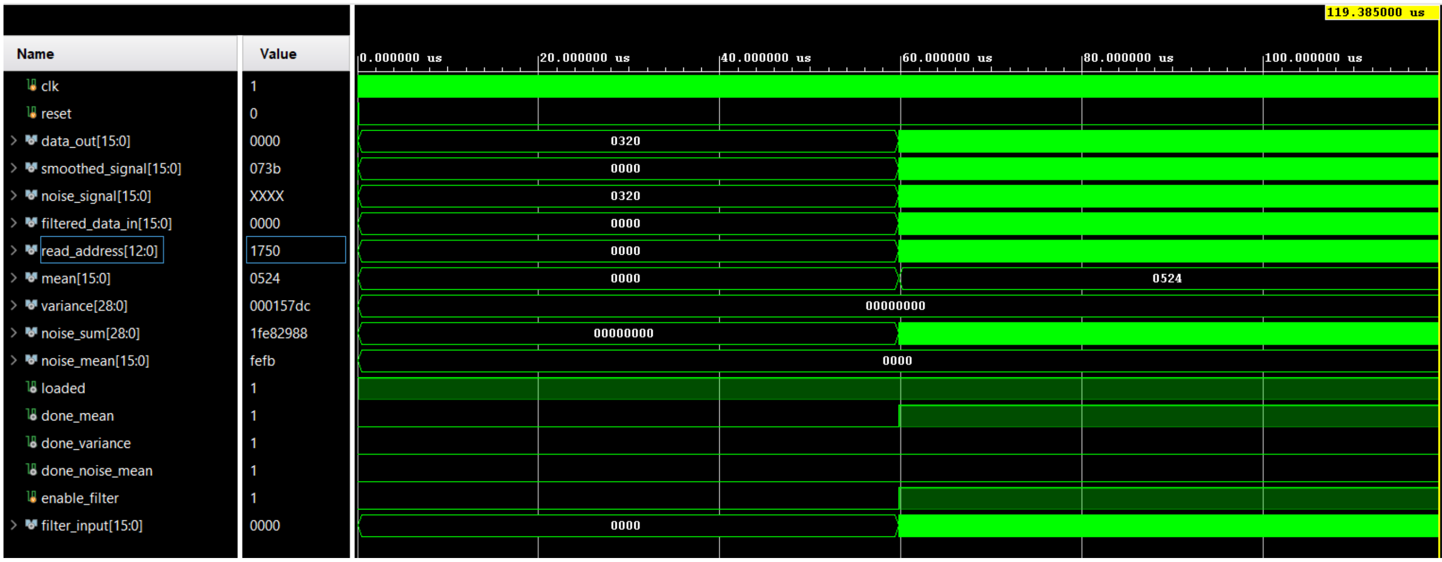
```
`timescale 1ns / 1ps

module tb_mean_noise();

  // Clock and Reset
  reg clk;
  reg reset;

  // Internal signals
  wire [15:0] data_out;
  wire [15:0] smoothed_signal;
  wire signed [15:0] noise_signal;
  wire [15:0] filtered_data_in;
  wire [12:0] read_address;
  wire [15:0] mean;
  wire [28:0] variance;
  wire signed [28:0] noise_sum;
  wire signed [15:0] noise_mean;
  wire loaded;
  wire done_mean;
  wire done_variance;
  wire done_noise_mean;

  // Control signal to enable feeding data to filter
  reg enable_filter;
  wire [15:0] filter_input = enable_filter ? data_out : 16'd0;

  // Clock generation
  always #5 clk = ~clk;
```

```
// Module instantiations

// Load data module
load_data data_loader (
    .clk(clk),
    .reset(reset),
    .read_address(read_address),
    .data_out(data_out),
    .loaded(loaded)
);

assign filtered_data_in = done_mean ? data_out : 16'd0;
// Moving Average Filter (feed data only after done_mean)
moving_average_filter filter_inst (
    .clk(clk),
    .reset(reset),
    .data_out(filter_input),
    .smoothed_signal(smoothed_signal)
);

// Noise Signal
noise_signal noise_inst (
    .clk(clk),
    .reset(reset),
    .data_out(data_out),
    .smoothed_signal(smoothed_signal),
    .noise_signal(noise_signal)
);

// Mean calculation for original signal
calculate_mean mean_inst (
    .clk(clk),
    .reset(reset),
    .mean(mean),
    .done(done_mean)
);

// Variance calculation for original signal
calculate_variance var_inst (
    .clk(clk),
    .reset(reset),
    .variance(variance),
    .done_variance(done_variance),
    .read_address(read_address),
    .data_out(data_out)
);
```

```
// Mean of noise signal
calculate_noise_mean noise_mean_inst (
    .clk(clk),
    .reset(reset),
    .valid_noise(done_mean), // start accumulating noise after done_mean is high
    .noise_signal(noise_signal),
    .noise_sum(noise_sum),
    .noise_mean(noise_mean),
    .done_noise_mean(done_noise_mean)
);

// Initialization
initial begin
    clk = 0;
    reset = 1;
    enable_filter = 0;

    #15 reset = 0;

    // Wait for data loading
    wait(loaded);
    $display("Data loaded successfully.");

    // Wait for mean of signal
    wait(done_mean);
    $display("Mean of signal done: %d", mean);
    enable_filter = 1;

    // Wait for noise mean
    wait(done_noise_mean);
    $display("Noise mean calculation complete. Noise Mean: %d", noise_mean);

    $stop;
end

// Monitor key values
//  initial begin
   //  $monitor("Time=%0t | DataOut=%d | Smoothed=%d | Noise=%d | NoiseSum=%d | NoiseMean=%d |
DoneMean=%b | DoneNoiseMean=%b | ReadAddr=%d",
   //      $time, data_out, smoothed_signal, noise_signal, $signed(noise_sum), noise_mean, done_mean,
done_noise_mean, read_address);
// end

initial begin
```

```
        $monitor("Time: %0t | Read_Address: %d | Data_Out: %d | Smoothed: %d | Noise: %d | NoiseSum=%d | Mean: %d | Variance: %d | Noise_Mean: %d | Done_Mean: %b | Done_Variance: %b | Done_Noise_Mean: %b",
                 $time, read_address, data_out, smoothed_signal, $signed(noise_signal), $signed(noise_sum), mean, variance, $signed(noise_mean), done_mean, done_variance, done_noise_mean);
    end

endmodule
```

RESULT :




```
Time: 119255000 | Read_Address: 5956 | Data_Out:  1755 | Smoothed:  1674 | Noise:    222 | NoiseSum=  -1558826 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119265000 | Read_Address: 5957 | Data_Out:  1765 | Smoothed:  1819 | Noise:     81 | NoiseSum=  -1558604 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119275000 | Read_Address: 5958 | Data_Out:  1765 | Smoothed:  1931 | Noise:    -54 | NoiseSum=  -1558523 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119285000 | Read_Address: 5959 | Data_Out:  1750 | Smoothed:  2012 | Noise:   -166 | NoiseSum=  -1558577 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119295000 | Read_Address: 5960 | Data_Out:  1710 | Smoothed:  2064 | Noise:   -262 | NoiseSum=  -1558743 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119305000 | Read_Address: 5961 | Data_Out:  1665 | Smoothed:  2093 | Noise:   -354 | NoiseSum=  -1559005 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119315000 | Read_Address: 5962 | Data_Out:  1615 | Smoothed:  2097 | Noise:   -428 | NoiseSum=  -1559359 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119325000 | Read_Address: 5963 | Data_Out:  1580 | Smoothed:  2082 | Noise:   -482 | NoiseSum=  -1559787 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119335000 | Read_Address: 5964 | Data_Out:  1555 | Smoothed:  2054 | Noise:   -502 | NoiseSum=  -1560269 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119345000 | Read_Address: 5965 | Data_Out:  1530 | Smoothed:  2017 | Noise:   -499 | NoiseSum=  -1560771 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119355000 | Read_Address: 5966 | Data_Out:  1500 | Smoothed:  1975 | Noise:   -487 | NoiseSum=  -1561270 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119365000 | Read_Address: 5967 | Data_Out:  1475 | Smoothed:  1931 | Noise:   -475 | NoiseSum=  -1561757 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Time: 119375000 | Read_Address: 5968 | Data_Out:     x | Smoothed:  1889 | Noise:   -456 | NoiseSum=  -1562232 | Mean:  1316 | Variance:          0 | Noise_Mean:      0 | Done_Mean: 1 | Done_Variance: 0
Noise mean calculation complete. Noise Mean:   -261
```

NOISE VARIANCE :

Module Name: `calculate_noise_variance`

Functionality:

The `calculate_noise_variance` module computes the **statistical variance** of a given noise signal using a pre-computed noise mean. This is critical in **SNR (Signal-to-Noise Ratio)** calculations and evaluating noise energy in signal processing pipelines.

Key Features:

- Calculates variance from signed noise samples.
- Handles signed arithmetic to accommodate both positive and negative noise deviations.
- Outputs intermediate debugging values (`diff_out`, `squared_sum_out`).
- Controlled by a `valid_noise` signal to synchronize with data availability.

CODE :

```
module calculate_noise_variance #(
   parameter DATA_WIDTH = 16,
   parameter MEMORY_DEPTH = 5968
)(
   input clk,
   input reset,
   input valid_noise,
   input signed [DATA_WIDTH-1:0] noise_signal,
   input signed [DATA_WIDTH-1:0] noise_mean,
   output reg [DATA_WIDTH+12:0] noise_variance,
   output reg done_noise_variance,
   output reg signed [2*DATA_WIDTH-1:0] diff_out,
   output reg [2*DATA_WIDTH+15:0] squared_sum_out
);

   reg signed [2*DATA_WIDTH-1:0] diff;
   reg [2*DATA_WIDTH+15:0] squared_sum;

   // Control signal to stop accumulation
   reg [12:0] sample_count;

   always @(posedge clk or posedge reset) begin
      if (reset) begin
         diff <= 0;
         squared_sum <= 0;
         noise_variance <= 0;
         done_noise_variance <= 0;
         sample_count <= 0;
      end else if (valid_noise && !done_noise_variance) begin
         diff <= noise_signal - noise_mean;
         squared_sum <= squared_sum + (noise_signal - noise_mean) * (noise_signal - noise_mean);
         sample_count <= sample_count + 1;

        if (^noise_signal === 1'bx || sample_count == MEMORY_DEPTH - 1) begin
          noise_variance <= squared_sum / (MEMORY_DEPTH - 1);
          done_noise_variance <= 1;
        end
      end
   end

   // Output latching block
   always @(posedge clk or posedge reset) begin
      if (reset) begin
         diff_out <= 0;
```

```
        squared_sum_out <= 0;
      end else begin
        diff_out <= diff;
        squared_sum_out <= squared_sum;
      end
    end

endmodule
```

TESTBENCH :

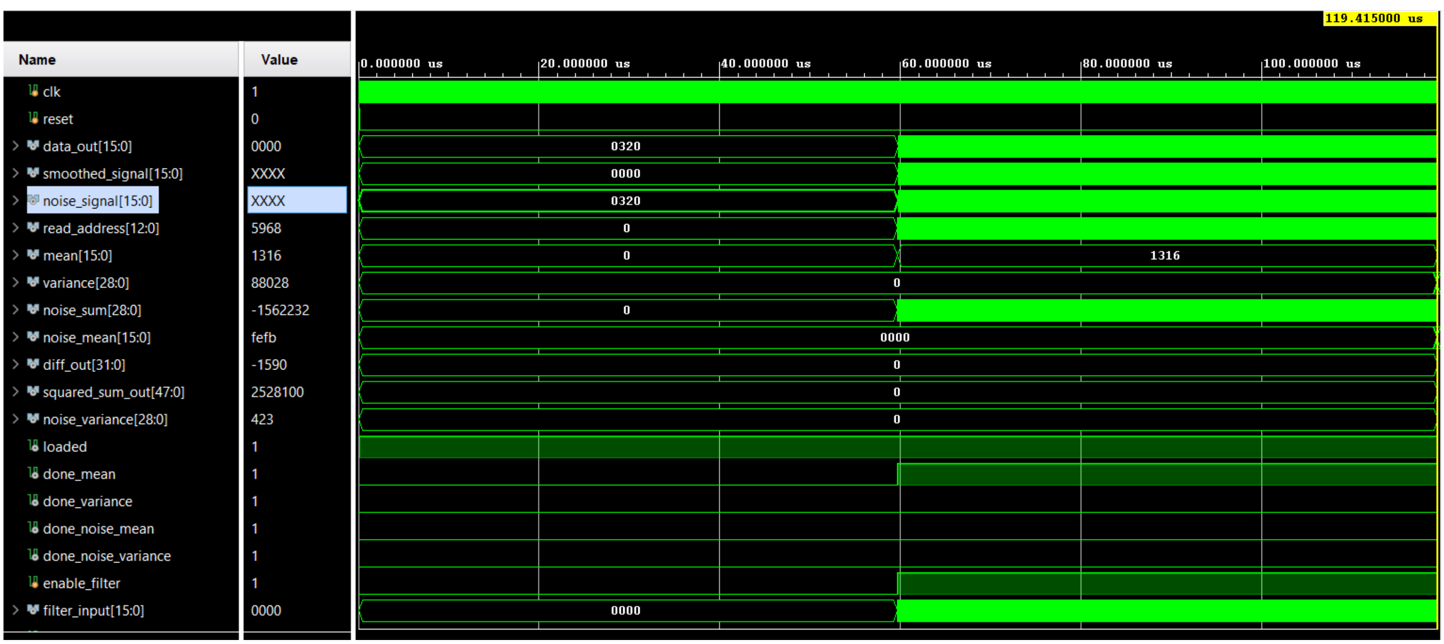
```
`timescale 1ns / 1ps

module tb_noise_variance();

    reg clk;
    reg reset;

    wire [15:0] data_out;
    wire [15:0] smoothed_signal;
    wire signed [15:0] noise_signal;
    wire [12:0] read_address;
    wire [15:0] mean;
    wire [28:0] variance;
    wire signed [28:0] noise_sum;
    wire signed [15:0] noise_mean;

    wire signed [31:0] diff_out;
    wire [47:0] squared_sum_out;
    //wire [12:0] sample_count_out;

    wire [28:0] noise_variance;
    wire loaded;
    wire done_mean;
    wire done_variance;
    wire done_noise_mean;
    wire done_noise_variance;

    reg enable_filter;
    wire [15:0] filter_input = enable_filter ? data_out : 16'd0;
    reg valid_noise;
always @(posedge clk or posedge reset) begin
    if (reset)
      valid_noise <= 0;
    else if (done_noise_mean && !done_noise_variance)
      valid_noise <= 1;
    else if (done_noise_variance)
```

```
    valid_noise <= 0;
end

  always #5 clk = ~clk;

  // Load data
  load_data data_loader (
    .clk(clk),
    .reset(reset),
    .read_address(read_address),
    .data_out(data_out),
    .loaded(loaded)
  );

  // Filter
  moving_average_filter filter_inst (
    .clk(clk),
    .reset(reset),
    .data_out(filter_input),
    .smoothed_signal(smoothed_signal)
  );

  // Noise
  noise_signal noise_inst (
    .clk(clk),
    .reset(reset),
    .data_out(data_out),
    .smoothed_signal(smoothed_signal),
    .noise_signal(noise_signal)
  );

  // Mean
  calculate_mean mean_inst (
    .clk(clk),
    .reset(reset),
    .mean(mean),
    .done(done_mean)
  );

  // Variance
  calculate_variance var_inst (
    .clk(clk),
    .reset(reset),
    .variance(variance),
    .done_variance(done_variance),
    .read_address(read_address),
```

```
        .data_out(data_out)
    );

    // Noise Mean
    calculate_noise_mean noise_mean_inst (
        .clk(clk),
        .reset(reset),
        .valid_noise(done_mean),
        .noise_signal(noise_signal),
        .noise_sum(noise_sum),
        .noise_mean(noise_mean),
        .done_noise_mean(done_noise_mean)
    );

    // Noise Variance
calculate_noise_variance #(
    .DATA_WIDTH(16),
    .MEMORY_DEPTH(5968)
) uut (
    .clk(clk),
    .reset(reset),
    .valid_noise(valid_noise),
    .noise_signal(noise_signal),
    .noise_mean(noise_mean),
    .noise_variance(noise_variance),
    .done_noise_variance(done_noise_variance),

    // Debug outputs
    .diff_out(diff_out),
    .squared_sum_out(squared_sum_out)
     // .sample_count_out(sample_count_out)
);


    initial begin
        clk = 0;
        reset = 1;
        enable_filter = 0;

        #15 reset = 0;

        wait(loaded);
        $display("Data loaded.");

        wait(done_mean);
        $display("Signal mean done.");
```

```
    enable_filter = 1;

//      wait(done_variance);
//      $display("Variance calculation Done.");

    wait(done_noise_mean);
    $display("Noise mean done: %d", noise_mean);

    wait(done_noise_variance);
    $display("Noise variance done: %d", noise_variance);

    $stop;
  end

  initial begin
    $monitor("diff_out = %d | squared_sum_out = %d | Time=%0t | Addr=%d | Data=%d | Smooth=%d | Noise=%d |
Mean=%d | Var=%d | NoiseMean=%d | NoiseVar=%d | DoneMean=%b | DoneVar=%b | DoneNoiseMean=%b |
DoneNoiseVar=%b",
             diff_out,squared_sum_out,$time, read_address, data_out, smoothed_signal, noise_signal, mean, variance,
noise_mean, noise_variance,
             done_mean, done_variance, done_noise_mean, done_noise_variance);
  end

endmodule
```

RESULT :



```
diff_out =          0 | squared_sum_out =                0 | Time=119305000 | Addr=5961 | Data= 1665 | Smooth= 2093 | Noise=  -354 | Mean= 1316 | Var=        0 | NoiseM
diff_out =          0 | squared_sum_out =                0 | Time=119315000 | Addr=5962 | Data= 1615 | Smooth= 2097 | Noise=  -428 | Mean= 1316 | Var=        0 | NoiseM
diff_out =          0 | squared_sum_out =                0 | Time=119325000 | Addr=5963 | Data= 1580 | Smooth= 2082 | Noise=  -482 | Mean= 1316 | Var=        0 | NoiseM
diff_out =          0 | squared_sum_out =                0 | Time=119335000 | Addr=5964 | Data= 1555 | Smooth= 2054 | Noise=  -502 | Mean= 1316 | Var=        0 | NoiseM
diff_out =          0 | squared_sum_out =                0 | Time=119345000 | Addr=5965 | Data= 1530 | Smooth= 2017 | Noise=  -499 | Mean= 1316 | Var=        0 | NoiseM
diff_out =          0 | squared_sum_out =                0 | Time=119355000 | Addr=5966 | Data= 1500 | Smooth= 1975 | Noise=  -487 | Mean= 1316 | Var=        0 | NoiseM
diff_out =          0 | squared_sum_out =                0 | Time=119365000 | Addr=5967 | Data= 1475 | Smooth= 1931 | Noise=  -475 | Mean= 1316 | Var=        0 | NoiseM
diff_out =          0 | squared_sum_out =                0 | Time=119375000 | Addr=5968 | Data=    x | Smooth= 1889 | Noise=  -456 | Mean= 1316 | Var=        0 | NoiseM
Noise mean done:   -261
diff_out =          0 | squared_sum_out =                0 | Time=119385000 | Addr=5968 | Data=    0 | Smooth= 1851 | Noise=     x | Mean= 1316 | Var=    88028 | NoiseM
diff_out =          0 | squared_sum_out =                0 | Time=119395000 | Addr=5968 | Data=    0 | Smooth=    x | Noise= -1851 | Mean= 1316 | Var=    88028 | NoiseM
diff_out =          0 | squared_sum_out =                0 | Time=119405000 | Addr=5968 | Data=    0 | Smooth=    x | Noise=     x | Mean= 1316 | Var=    88028 | NoiseM
Noise variance done:        423
```

SNR_LINEAR :

Module Name: `calculate_snr_linear`

Functionality:

The `calculate_snr_linear` module computes the **linear Signal-to-Noise Ratio (SNR)** by dividing the signal variance by the noise variance. This ratio is vital in quantifying how much the signal dominates over noise, particularly in biomedical, communication, and sensor signal analysis.

Key Features:

- Performs SNR calculation in linear scale:

  SNR linear= (Variance/ Noise Variance)

- Handles divide-by-zero edge case gracefully.
- Controlled with a `start_snr` signal for synchronization.
- Outputs a `done_snr_linear` flag when calculation completes.

CODE :

```
module calculate_snr_linear #(
   parameter DATA_WIDTH = 29,
   parameter OUTPUT_WIDTH = 32
)(
   input clk,
   input reset,
   input start_snr,
   input [DATA_WIDTH-1:0] variance,
   input [DATA_WIDTH-1:0] noise_variance,
   output reg [OUTPUT_WIDTH-1:0] snr_linear,
   output reg done_snr_linear
);

   always @(posedge clk or posedge reset) begin
      if (reset) begin
         snr_linear <= 0;
         done_snr_linear <= 0;
      end else if (start_snr && !done_snr_linear) begin
```

```
        if (noise_variance != 0) begin
          snr_linear <= variance / noise_variance;
        end else begin
          snr_linear <= 0;
        end
        done_snr_linear <= 1;
      end
    end

endmodule
```

TESTBENCH :

```
`timescale 1ns / 1ps

module tb_snr_linear();

    reg clk;
    reg reset;

    // Clock generator
    always #5 clk = ~clk;

    // Internal wires and regs
    wire [15:0] data_out;
    wire [15:0] smoothed_signal;
    wire signed [15:0] noise_signal;
    wire [12:0] read_address;
    wire [15:0] mean;
    wire [28:0] variance;
    wire signed [28:0] noise_sum;
    wire signed [15:0] noise_mean;

    wire signed [31:0] diff_out;
    wire [47:0] squared_sum_out;

    wire [28:0] noise_variance;
    wire loaded;
    wire done_mean;
    wire done_variance;
    wire done_noise_mean;
    wire done_noise_variance;

    wire [31:0] snr_linear;
    wire done_snr_linear;
```

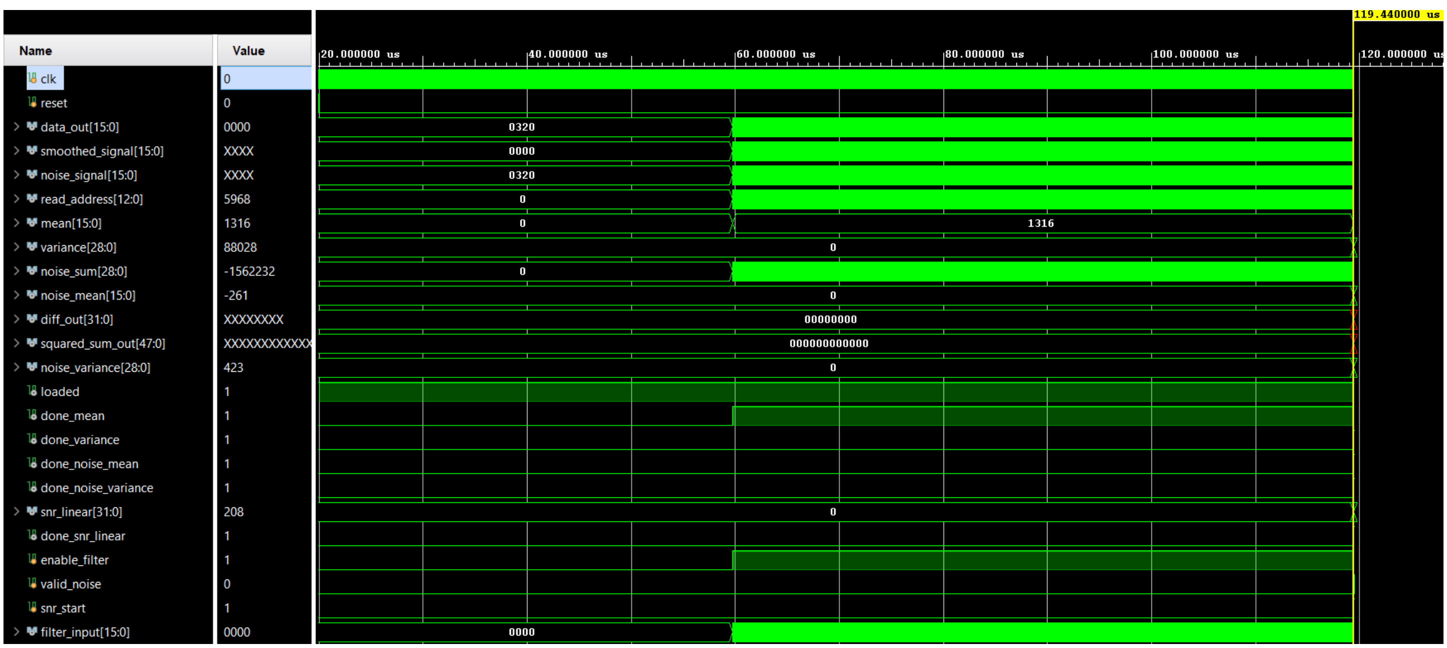
```
reg enable_filter;
reg valid_noise;
reg snr_start;

wire [15:0] filter_input = enable_filter ? data_out : 16'd0;

// Valid_noise controller
always @(posedge clk or posedge reset) begin
   if (reset)
      valid_noise <= 0;
   else if (done_noise_mean && !done_noise_variance)
      valid_noise <= 1;
   else if (done_noise_variance)
      valid_noise <= 0;
end

// SNR start trigger
always @(posedge clk or posedge reset) begin
   if (reset)
      snr_start <= 0;
   else if (done_variance && done_noise_variance)
      snr_start <= 1;
   else if (done_snr_linear)
      snr_start <= 0;
end

// Load data
load_data data_loader (
   .clk(clk),
   .reset(reset),
   .read_address(read_address),
   .data_out(data_out),
   .loaded(loaded)
);

// Filter
moving_average_filter filter_inst (
   .clk(clk),
   .reset(reset),
   .data_out(filter_input),
   .smoothed_signal(smoothed_signal)
);

// Noise extraction
noise_signal noise_inst (
   .clk(clk),
```

```
    .reset(reset),
    .data_out(data_out),
    .smoothed_signal(smoothed_signal),
    .noise_signal(noise_signal)
);

// Signal Mean
calculate_mean mean_inst (
    .clk(clk),
    .reset(reset),
    .mean(mean),
    .done(done_mean)
);

// Signal Variance
calculate_variance var_inst (
    .clk(clk),
    .reset(reset),
    .variance(variance),
    .done_variance(done_variance),
    .read_address(read_address),
    .data_out(data_out)
);

// Noise Mean
calculate_noise_mean noise_mean_inst (
    .clk(clk),
    .reset(reset),
    .valid_noise(done_mean),
    .noise_signal(noise_signal),
    .noise_sum(noise_sum),
    .noise_mean(noise_mean),
    .done_noise_mean(done_noise_mean)
);

// Noise Variance
calculate_noise_variance #(
    .DATA_WIDTH(16),
    .MEMORY_DEPTH(5968)
) noise_var_inst (
    .clk(clk),
    .reset(reset),
    .valid_noise(valid_noise),
    .noise_signal(noise_signal),
    .noise_mean(noise_mean),
    .noise_variance(noise_variance),
```

```
        .done_noise_variance(done_noise_variance),
        .diff_out(diff_out),
        .squared_sum_out(squared_sum_out)
    );

    // SNR Linear
    calculate_snr_linear snr_inst (
    .clk(clk),
    .reset(reset),
    .start_snr(snr_start),
    .variance(variance),
    .noise_variance(noise_variance),
    .snr_linear(snr_linear),
    .done_snr_linear(done_snr_linear)
);

    // Initial block
    initial begin
        clk = 0;
        reset = 1;
        enable_filter = 0;
        snr_start = 0;

        #15 reset = 0;

        wait(loaded);
        $display("Data loaded.");

        #10;
        wait(done_mean);
        $display("Signal mean done.");
        enable_filter = 1;

        #10;
        wait(done_noise_mean);
        $display("Noise mean done: %d", noise_mean);

        #10;
        wait(done_variance);
        $display("Signal variance done: %d", variance);

        #10;
        wait(done_noise_variance);
        $display("Noise variance done: %d", noise_variance);

        #1;
```

```
    wait(done_snr_linear);
    $display("SNR Linear Ratio (Q16.16): %d", snr_linear);
    #5;

    $stop;
  end

  // Monitor block
  initial begin
    $monitor("Time=%0t | Addr=%d | Data=%d | Smooth=%d | Noise=%d | Mean=%d | Var=%d | NoiseMean=%d | 
NoiseVar=%d | SNR=%d | DoneMean=%b | DoneVar=%b | DoneNoiseMean=%b | DoneNoiseVar=%b | DoneSNR=%b",
             $time, read_address, data_out, smoothed_signal, noise_signal, mean, variance,
             noise_mean, noise_variance, snr_linear,
             done_mean, done_variance, done_noise_mean, done_noise_variance, done_snr_linear);
  end

endmodule
```

RESULT :




```
Time=119345000 | Addr=5965 | Data= 1530 | Smooth= 2017 | Noise=  -499 | Mean= 1316 | Var=        0 | NoiseMean=     0 | NoiseVar=        0 | SNR=         0 | DoneMean=1 |
Time=119355000 | Addr=5966 | Data= 1500 | Smooth= 1975 | Noise=  -487 | Mean= 1316 | Var=        0 | NoiseMean=     0 | NoiseVar=        0 | SNR=         0 | DoneMean=1 |
Time=119365000 | Addr=5967 | Data= 1475 | Smooth= 1931 | Noise=  -475 | Mean= 1316 | Var=        0 | NoiseMean=     0 | NoiseVar=        0 | SNR=         0 | DoneMean=1 |
Time=119375000 | Addr=5968 | Data=    x | Smooth= 1889 | Noise=  -456 | Mean= 1316 | Var=        0 | NoiseMean=     0 | NoiseVar=        0 | SNR=         0 | DoneMean=1 |
Noise mean done:   -261
Time=119385000 | Addr=5968 | Data=    0 | Smooth= 1851 | Noise=     x | Mean= 1316 | Var=    88028 | NoiseMean=  -261 | NoiseVar=        0 | SNR=         0 | DoneMean=1 |
Signal variance done:     88028
Time=119395000 | Addr=5968 | Data=    0 | Smooth=    x | Noise= -1851 | Mean= 1316 | Var=    88028 | NoiseMean=  -261 | NoiseVar=        0 | SNR=         0 | DoneMean=1 |
Time=119405000 | Addr=5968 | Data=    0 | Smooth=    x | Noise=     x | Mean= 1316 | Var=    88028 | NoiseMean=  -261 | NoiseVar=        0 | SNR=         0 | DoneMean=1 |
Noise variance done:       423
SNR Linear Ratio (Q16.16):         208
Time=119415000 | Addr=5968 | Data=    0 | Smooth=    x | Noise=     x | Mean= 1316 | Var=    88028 | NoiseMean=  -261 | NoiseVar=      423 | SNR=         0 | DoneMean=1 |
SNR Linear Ratio (Q16.16):         208
Time=119435000 | Addr=5968 | Data=    0 | Smooth=    x | Noise=     x | Mean= 1316 | Var=    88028 | NoiseMean=  -261 | NoiseVar=      423 | SNR=       208 | DoneMean=1 |
```

FINAL ANSWERS :

MATLAB :

```
Signal Mean: 1.286
Signal Variance: 0.083921
Signal Power: 0.083921
Mean of Noise: -1.5425e-05
Variance of Noise: 0.00039776
Noise Power: 0.00039776
SNR: 23.2425 dB
```

```
SNR is in the acceptable range. Proceeding to Heart Rate calculation...
```

VIVADO :

Signal Mean = 1316

Signal Variance (or) Signal power = 88028

Noise Mean = -261

Noise Variance (or) Noise power = 423

SNR_linear (signal_variance/noise_variance) = 208

SNR (In Decibels) = 10 x log(208)
= 10 x 2.31806
= 23.1806