A project report on

Heart Rate Detection System Based on PPG Signal Using Verilog

Submitted in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

By

SAPPARAPUDEVI SRI PRASAD(21BEC7245)

SCHOOL OF ELECTRONICS ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

May 2025

DECLARATION

I hereby declare that the thesis entitled "Heart Rate Detection System based on PPG signal using Verilog" submitted by me, for the award of the degree of Bachelor of Technology, VIT, is a record of bonafide work carried out by me under the supervision of Dr. G.D. V. Santosh Kumar Sir

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

S. Devi Sri Prasad

Place: Amaravati

Date: 16-05-2025

Signature of the Candidate

CERTIFICATE

This is to certify that the Senior Design Project titled **"HEART RATE DETECTION SYSTEM BASED ON PPG USING VERILOG"** that is being submitted by **SAPPARAPU DEVI SRI PRASAD (21BEC7245)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for an award of any degree or diploma and the same is certified.

[illegible] 16/5/25
Dr. G.D.V.SantoshKumar

Guide

The thesis is satisfactory/unsatisfactory

[illegible] 16/5/25
Internal Examiner

[illegible] 16/5/2025
External Examiner

Approved by

[illegible]
PROGRAM CHAIR
B. Tech. ECE

[illegible]
DEAN
School Of Electronics Engineering

ACKNOWLEDGEMENTS

It is my pleasure to express with a deep sense of gratitude to Dr.G.D.V.Santosh Kumar, Assistant Professor Senior, SENSE, VIT-AP, for his constant guidance, continual encouragement and understanding; more than all, he taught me patience in my endeavour. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of VLSI.

I would like to express my gratitude to Dr. G. Viswanathan, Dr . Sekhar Viswanathan, Dr. G.V. Selvam, Dr. S.V. Kota Reddy, and Dr. Umakanta Nanda, SENSE, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Chandan Kumar Pandey. Associate Professor Grade-1, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Date: 16-05-2025 **SAPPARAPU DEVI SRI PRASAD**

ABSTRACT

In this project, a hardware-efficient system is designed for PPG data heart rate detection using Photoplethysmography (PPG) signals processed through Verilog HDL. The system implements a fully functional digital pipeline for signal quality assessment and heart rate computation, optimised for FPGA-based deployment. Unlike software-based implementations, this Verilog-based approach enables low-latency, power-efficient processing suitable for integration into wearable and embedded medical devices.The proposed architecture consists of two main functional blocks: an SNR Calculation Block and a Heart Rate Calculation Block. The SNR block performs signal preprocessing, computes mean and variance values, applies a moving average filter for smoothing, and estimates noise characteristics to calculate the signal-to-noise ratio (SNR), serving as a Signal Quality Index (SQI). The heart rate block employs a Savitzky-Golay filter for smoothing, a Butterworth bandpass filter for removing unwanted frequency components, followed by peak detection to determine the pulse rate. The final heart rate is computed in beats per minute (BPM) based on the RR interval between successive peaks.This work demonstrates the potential of Verilog-based physiological signal processing in resource-constrained, real-time applications. The modular design, validated through waveform simulation in Vivado, highlights the feasibility of deploying such systems on FPGAs for continuous health monitoring. Future enhancements may include SpO_2 estimation and full FPGA integration.

TABLE OF CONTENTS

List of Figures

CHAPTER 1
INTRODUCTION

Health monitoring systems have become an essential part of modern healthcare. With the increasing prevalence of chronic illnesses and the rising demand for wearable devices, continuous monitoring of physiological parameters like heart rate has gained significant importance. Photoplethysmography (PPG) is a simple, non-invasive optical technique widely used in wearable devices to monitor heart rate, oxygen saturation, and other vital signs.

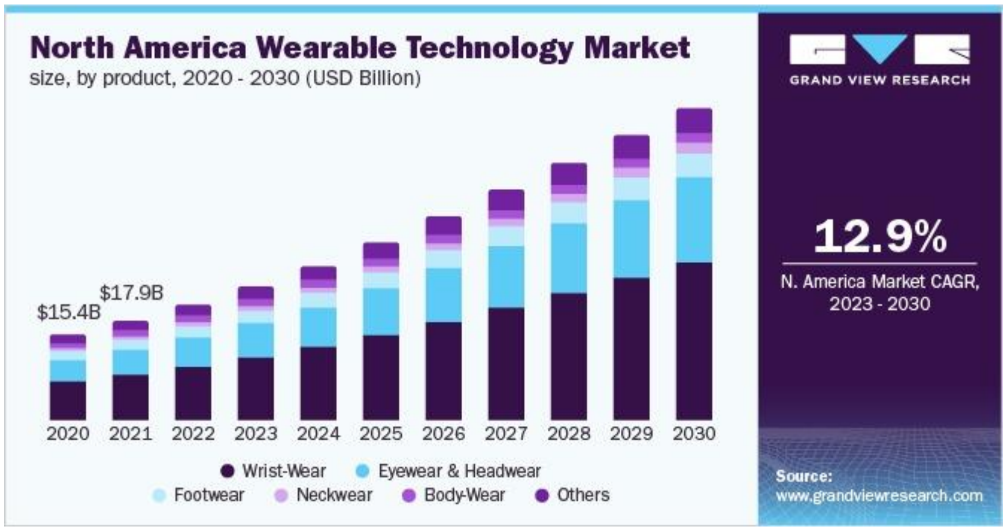


Figure 1:Wearable Technology usage

A significant challenge in heart rate monitoring arises due to motion artefacts, ambient light interference, and the overall quality of the acquired PPG signals. These factors can lead to incorrect heart rate estimations, which can compromise the reliability of health monitoring systems. Traditional systems often rely on simple threshold-based algorithms for heart rate calculation, which may not be robust enough to handle these challenges.

In this project, we propose a comprehensive heart rate detection system that processes PPG signals using VERILOG. The system performs multiple stages of signal processing, including noise removal, signal quality assessment, and peak detection, to ensure accurate calculation of beats per minute (BPM). By introducing the concept of Signal Quality Index (SQI), the system can identify and exclude noisy or unreliable signals, further improving the robustness of the heart rate calculations.

While conventional health monitoring systems often use pre-defined thresholds or periodic monitoring approaches, they fail to adapt to the dynamic nature of real-world signals. Our system offers an efficient solution by automatically processing signals, analysing their quality, and calculating BPM in real time.

This project also lays the groundwork for future hardware implementation of the PPG block on FPGA platforms using logic synthesis techniques. Such an implementation will enable the integration of the PPG block into wearable devices, providing a reliable and energy-efficient solution for continuous health monitoring.

1.1 Objectives

The following are the objectives of this project:

- **To design an efficient system** that can process PPG signals to automatically detect heart rate with high accuracy and reliability.
- **To implement Signal Quality Index (SQI)** calculation to ensure the reliability of the PPG signals by identifying and excluding noisy or unreliable data.
- **To preprocess PPG signals** by removing noise, filtering undesired frequencies, and normalizing the signals to improve the accuracy of heart rate calculations.
- **To develop and validate algorithms in VERILOG** for key tasks, including signal filtering, peak detection, and BPM calculation.
- **To lay the groundwork for hardware implementation**, where the PPG processing block can be deployed on FPGA platforms for real-time applications.
- **To enhance the scalability of the system** for future integration into wearable health monitoring devices, providing a reliable, low-power solution for continuous health tracking.

1.2 Background and Literature Survey

Photoplethysmography (PPG) is a widely used non-invasive optical technique to monitor physiological parameters such as heart rate and blood oxygen saturation. With the growing demand for wearable and portable healthcare devices, PPG-based systems have gained significant attention for their simplicity, affordability, and compatibility with real-time applications.

Background

The foundation of this project lies in the increasing need for continuous, real-time monitoring of cardiovascular health, driven by the rise in chronic diseases like hypertension and arrhythmias. Traditional methods often rely on periodic monitoring or ECG systems, which may not be ideal for compact and energy-efficient wearables. PPG offers a simpler alternative by utilizing light-based technology to detect volumetric changes in blood flow, making it ideal for continuous monitoring.

Despite its advantages, PPG signal processing is challenged by noise, motion artifacts, and baseline drift, which can compromise the accuracy of heart rate detection. This project addresses these challenges by implementing a VERILOG-based system for signal preprocessing, Signal Quality Index (SQI) calculation, peak detection, and BPM estimation.

Literature Survey

1. **"Photoplethysmography and Its Clinical Applications"**[4]
   This paper highlights the clinical significance of PPG technology, emphasizing its application in monitoring heart rate, respiratory rate, and blood oxygen levels. It discusses the physiological principles underlying PPG signals and the challenges associated with real-world signal acquisition, such as motion artifacts and ambient light interference. This study served as a foundational reference for understanding the characteristics of PPG signals and their clinical importance.
2. **"Energy-Efficient and Real-Time Wearable for PPG Signal Processing"**[2]
   This study explores energy-efficient methods for real-time PPG signal processing, focusing on implementing the PPG block in System-on-Chip (SoC) platforms like Zynq 7000. It highlights the importance of resource-efficient hardware designs and fixed-point arithmetic for reducing power consumption in wearable devices. The research provided insights into transitioning our VERILOG-based algorithms to HDL for future FPGA implementation.
3. **"Advanced FPGA Implementation of PPG-Based Cardiovascular Disease Detection"**
   [1]This paper discusses the use of FPGA-based systems for PPG signal processing to detect cardiovascular diseases such as hypertension and arrhythmias. It highlights techniques like parallel processing, pipelining, and resource-efficient designs to enable real-time signal analysis. The study also emphasizes the importance of SQI metrics for ensuring reliable heart rate detection. This work guided our approach in designing efficient VERILOG algorithms for future hardware realization.
4. **Importance of Signal Quality Index (SQI) in PPG Processing**
   Several studies, including the ones mentioned above, emphasize the role of SQI in PPG processing. Calculating SQI allows systems to identify and exclude noisy or unreliable

signals, thereby improving the accuracy of heart rate calculations. Techniques combining statistical metrics (e.g., mean, variance, skewness) with machine learning models were explored as potential improvements for our system.

Relevance of the Literature to the Current Project

The reviewed studies provided a solid foundation for this project. Key takeaways include:

- **Signal Quality and Filtering**: The clinical application study highlighted the importance of preprocessing techniques, such as filtering and normalization, to address noise and artifacts in PPG signals.
- **Energy-Efficient Design**: The focus on energy-efficient SoC and FPGA implementations aligns with our goal to eventually transition the VERILOG-based system for real-time hardware.
- **Real-Time Analysis**: The FPGA implementation study emphasized the role of resource optimization and parallel processing, which informed our methodology for HDL synthesis.

By integrating the insights from these studies, this project aims to develop a robust heart rate detection system that combines accurate VERILOG simulations with the groundwork for hardware implementation. This dual focus ensures scalability for wearable health monitoring applications, addressing both software and hardware challenges in PPG signal processing.

1.3 Organization of the Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology, hardware and software details.
- Chapter 3 gives the cost involved in the implementation of the project.
- Chapter 4 discusses the results obtained after the project was implemented.
- Chapter 5 concludes the report.
- Chapter 6 consists of codes.
- Chapter 7 gives references.

CHAPTER 2

PROPOSED SYSTEM ARCHITECTURE

This chapter describes the proposed system, working methodology, software and hardware details.

2.1 Proposed System

The following block diagram (figure 2) shows the system architecture of this project.

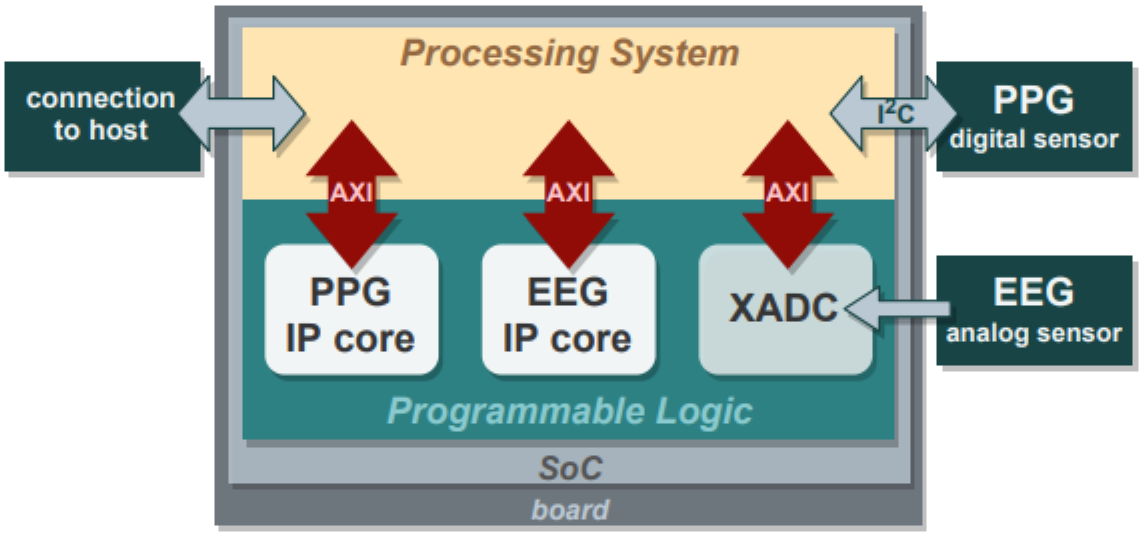


Figure 2: Proposed system architecture

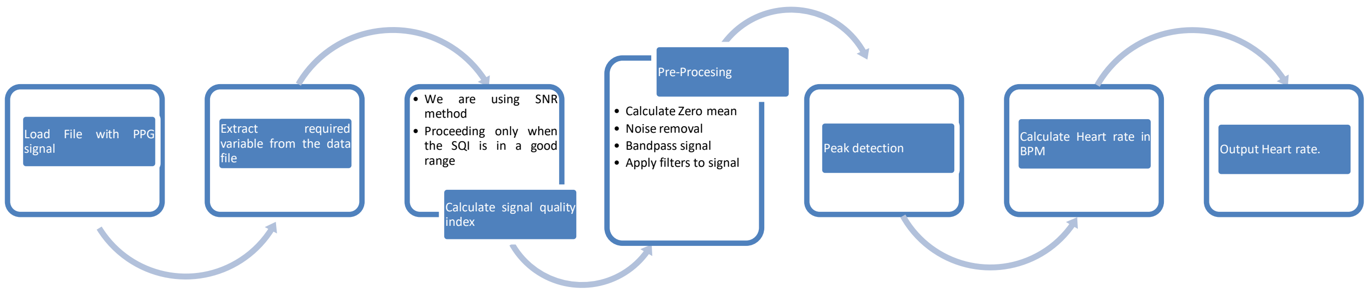

Load File with PPG signal
Extract required variable from the data file
We are using SNR method
Proceeding only when the SQI is in a good range
Calculate signal quality index
Pre-Procesing
Calculate Zero mean
Noise removal
Bandpass signal
Apply filters to signal
Peak detection
Calculate Heart rate in BPM
Output Heart rate.

Figure 3:data Flow process

2.2 Working Methodology

Software Section

The software implementation, developed using VERILOG, involves signal acquisition, preprocessing, and analysis. The key steps are:

1. **Signal Acquisition**
    - PPG signals are obtained from publicly available datasets or hardware sources (to be integrated in future iterations).
    - The acquired signal is raw and often contains noise, motion artefacts, and baseline drift, requiring pre-processing for accurate analysis.
2. **Signal Quality Index (SQI) Calculation**
    - The quality of the acquired signal is assessed by calculating the Signal Quality Index (SQI).
    - Statistical metrics such as standard deviation, variance, and mean-to-standard deviation ratio are computed to determine the reliability of the signal.
    - Only signals with SQI values above a predefined threshold are processed further to ensure accurate heart rate detection.
3. **Signal Preprocessing**
    - **Noise Removal**: A bandpass filter (0.5–2.5 Hz) is applied to eliminate baseline drift and high-frequency noise, ensuring only physiological frequencies are retained.
    - **Normalization**: The filtered signal is normalised to zero mean by subtracting its average value, preparing it for peak detection and BPM calculation.
4. **Peak Detection and Feature Extraction**
    - Peaks in the PPG waveform are identified using a threshold-based algorithm implemented in VERILOG.
    - Time intervals between consecutive peaks are calculated to derive heart rate in beats per minute (BPM).
5. **BPM Calculation**
    - The average BPM is computed over a defined time window to provide a stable heart rate measurement.
    - The system visualises the processed signal, highlighting detected peaks, and displays the calculated BPM.

Hardware Section

The system will transition to a hardware-based implementation, where the PPG signal processing block is developed as an IP core and integrated into an FPGA platform for real-time operation.

1. **PPG IP Core Development**
   - A custom PPG IP core will be developed in HDL (Hardware Description Language) to process PPG signals directly on the FPGA.
   - The IP core will implement key steps such as SQI calculation, noise filtering, peak detection, and BPM calculation.
2. **FPGA Integration**
   - The PPG IP core will be synthesized and deployed on an FPGA platform like Zynq 7000.
   - The programmable logic will handle real-time signal processing, while ARM processors in the SoC can be used for advanced analytics or interfacing.
3. **Signal Transmission and User Interface**
   - The processed heart rate data will be transmitted to a user interface via Bluetooth or Wi-Fi for real-time monitoring.
   - The system will provide insights on heart rate and signal quality through an intuitive user interface.

2.3 Standards

Various industry standards and guidelines have been referred to and considered in the development of this project, particularly focusing on the software aspects of PPG signal processing and heart rate detection.

- **IEEE 11073 - Personal Health Devices Communication Standards**

This standard provides guidelines for interoperability between personal health devices, including wearable devices. It ensures that PPG-based systems can communicate effectively with other health monitoring devices and central data repositories. Adopting this standard allows the integration of the heart rate detection system into larger healthcare ecosystems

- **ANSI/AAMI EC13: Cardiac Monitors, Heart Rate Meters, and Alarms**

The ANSI/AAMI EC13 standard outlines requirements for the safety and performance of heart rate monitoring systems. This standard has been referenced to validate the accuracy and reliability of heart rate detection algorithms implemented in VERILOG.

- **Signal Processing Standards**

1 Bandpass Filtering Standards

The filtering techniques employed in this project adhere to established signal processing guidelines, such as IEEE standards for biomedical signal processing. The bandpass filter (0.5–2.5 Hz) is designed to retain only the frequency range associated with physiological heart rates, ensuring compliance with medical device requirements.

2 Signal Quality Index (SQI) Metrics

The concept of SQI follows industry practices for assessing biomedical signal reliability. Studies such as those from IEEE Xplore highlight the importance of incorporating statistical measures like standard deviation, variance, and skewness to calculate SQI, ensuring consistent and noise-free analysis.

- **Health Data Privacy and Security Standards**

1 HIPAA (Health Insurance Portability and Accountability Act)

Although the current project is in its development phase, future integrations into wearable devices must comply with HIPAA standards for securing patient data. Any cloud-based or mobile health applications developed for this system must ensure the confidentiality and integrity of health data.

2 GDPR (General Data Protection Regulation)

For international applications, the system will consider GDPR compliance to protect the personal data of users, particularly when integrating cloud-based health monitoring platforms.

- **Research Papers as References for Standards**

1 "Photoplethysmography and Its Clinical Applications"

This paper emphasises the importance of robust signal processing techniques, aligning with industry standards for noise reduction and feature extraction[4]

2 "Energy-Efficient and Real-Time Wearable for PPG Signal Processing"

This study provides insights into hardware optimisation standards for wearable devices, particularly the use of fixed-point arithmetic and FPGA integration[2].

3 "Advanced FPGA Implementation of PPG-Based Cardiovascular Disease Detection"

This paper highlights standards for FPGA-based designs, such as resource utilization and parallel processing, ensuring compliance with industry benchmarks for efficiency and performance[1].

2.4 System Details

This section provides an overview of the system's software components. The heart rate detection system is designed for PPG signal processing and relies on VERILOG for signal analysis andHDL integration through Xilinx Vivado

2.4.1 Software Details

The software implementation is the backbone of this project and has been developed entirely in MATLAB. MATLAB provides a robust platform for signal processing and algorithm development, making it an ideal choice for PPG-based heart rate detection. Xilinx Vivado is used for HDL realization to support hardware integration.

2.4.2. Implementation Environment and Tools

The PPG signal processing pipeline was implemented using **Verilog HDL** as the hardware description language, with **Xilinx Vivado** serving as the primary tool for design, synthesis, simulation, and verification. The key components and techniques involved in achieving the project objectives are detailed below:

2.4.2.1 Bandpass Filtering (0.5–2.5 Hz)

A **custom-designed 6th-order Butterworth IIR bandpass filter** was implemented in Verilog using fixed-point arithmetic to suit hardware synthesis. The filter was designed to attenuate both low-frequency baseline wander and high-frequency noise, effectively isolating the physiological frequency range corresponding to the human heart rate (typically 0.5 to 2.5 Hz). Simulation of the filter response was performed using Vivado’s behavioral simulator.

2.4.2.2 Statistical Metrics for Signal Quality Index (SQI)

Verilog does not include inbuilt functions for statistical calculations such as mean, standard deviation, or variance. Therefore, **custom Verilog modules** were developed to compute these metrics. These were used to evaluate the **Signal Quality Index (SQI)** of the PPG signal to ensure reliable heart rate estimation. The statistical computations were implemented using running average and accumulator-based methods for efficient hardware realization.

2.4.2.3 Peak Detection Algorithm

As Verilog lacks built-in functions like findpeaks, a **custom threshold-based peak detection algorithm** was designed and implemented. This algorithm scans the filtered signal to identify local maxima by comparing each sample with its neighbors, and applies both amplitude and temporal thresholds to reduce false detections caused by noise. This approach ensured accurate detection of heartbeat peaks.

2.4.2.4 Signal Visualization

Since **Verilog and Vivado do not support signal plotting**, waveform data (raw, filtered, and processed) were exported to external .dat files. These files were then visualized using **MATLAB or Python** to verify filtering performance, peak detection accuracy, and BPM estimation. This external visualization allowed for a clearer understanding of system behavior during simulation.

2.4.2.5 Custom Algorithms

In addition to standard signal processing stages, several **custom Verilog modules** were developed:

- **Normalization module**: To scale PPG signal values for consistent processing.
- **SQI module**: To compute reliability scores based on signal statistics.
- **BPM Estimation module**: To calculate heart rate from the time intervals between detected peaks, using a finite state machine (FSM).

These modules were integrated into a top-level Verilog design and verified using testbenches in Vivado. The final design was made ready for FPGA synthesis and deployment.

ii) Xilinx Vivado for HDL Realization

Xilinx Vivado was employed for simulating the transition from algorithms to HDL design. Although the HDL implementation is planned for future work, Vivado provided insights into:

- Translating MATLAB-based algorithms into hardware-friendly logic.
- Designing the PPG IP core with resource optimization for FPGA deployment.
- Simulating real-time signal processing workflows for peak detection and BPM calculation.

Software Workflow

1. **Data Acquisition**: Raw PPG signals were loaded into VERILOG from publicly available datasets for processing and analysis.
2. **Signal Quality Assessment**: Statistical functions calculated the SQI, ensuring only reliable signals were analyzed further.
3. **Preprocessing**: Bandpass filters removed noise and normalized the signal for peak detection.
4. **Peak Detection and BPM Calculation**: Algorithms detected peaks in the processed signal, and the intervals between peaks were used to calculate heart rate.

5. **Visualization**: Results were visualized through plots to validate the effectiveness of the implemented algorithms.

CHAPTER 3

RESULTS AND DISCUSSION

This section provides an in-depth analysis of the results obtained during the implementation of the heart rate detection system using PPG signals. The workflow encompasses data loading, Signal Quality Index (SQI) calculation, preprocessing, peak detection, and the final heart rate calculation in beats per minute (BPM). The results for each stage are presented and discussed with corresponding plots and metrics.

3.1 Loading the Data

The PPG signal was loaded into VERILOG from publicly available datasets. Below result shows the raw PPG signal as plotted after loading into VERILOG. The raw signal includes noise, motion artifacts, and baseline drift, which necessitate preprocessing steps before reliable heart rate calculations.

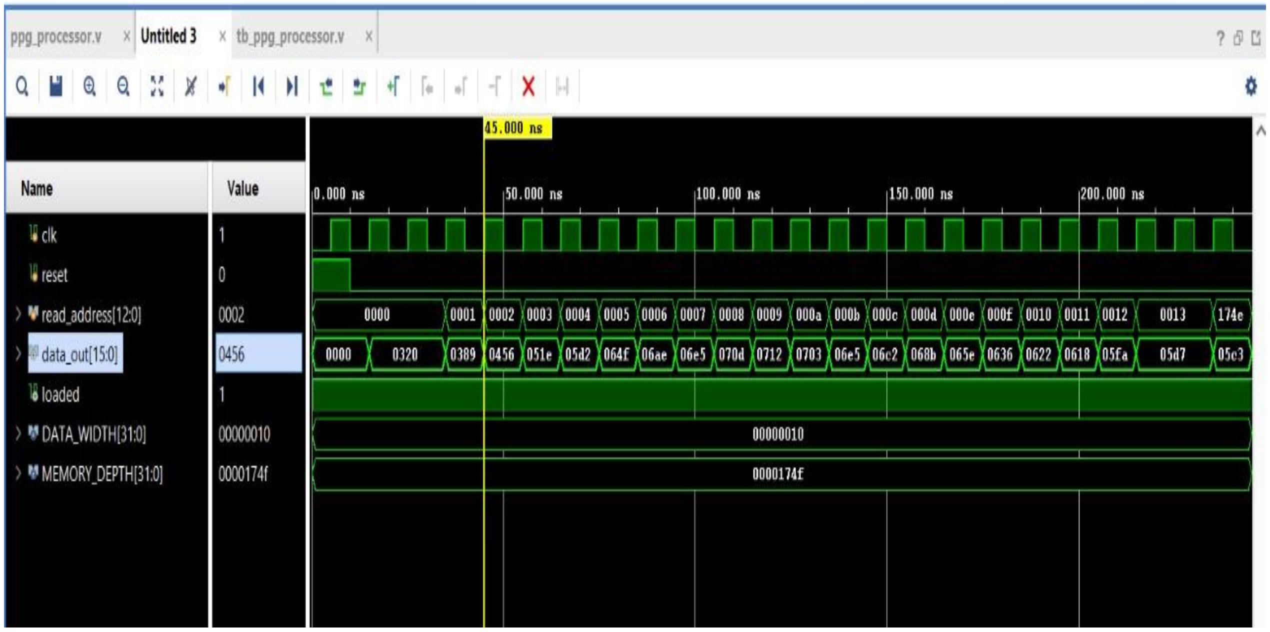
Steps Involved:

1. The data is loaded into VERILOG using built-in functions like load or readtable, depending on the file format.
2. The signal is visualized to identify any anomalies and to confirm successful loading.

The raw signal, as seen in below figure, exhibits both low-frequency baseline drift and high-frequency noise. These components need to be filtered out in subsequent steps.

Result:

Upon loading the data, the raw PPG signal is visualized to provide an overview of its characteristics. The plotted signal shows variations in amplitude corresponding to changes in blood volume over time. The raw signal highlights the presence of noise, baseline drift, and other artifacts that need to be addressed in subsequent steps.



Name	Value
clk	1
reset	0
read_address[12:0]	0002
data_out[15:0]	0456
loaded	1
DATA_WIDTH[31:0]	00000010
MEMORY_DEPTH[31:0]	0000174f

Figure 4 :Simulation Raw PPG data

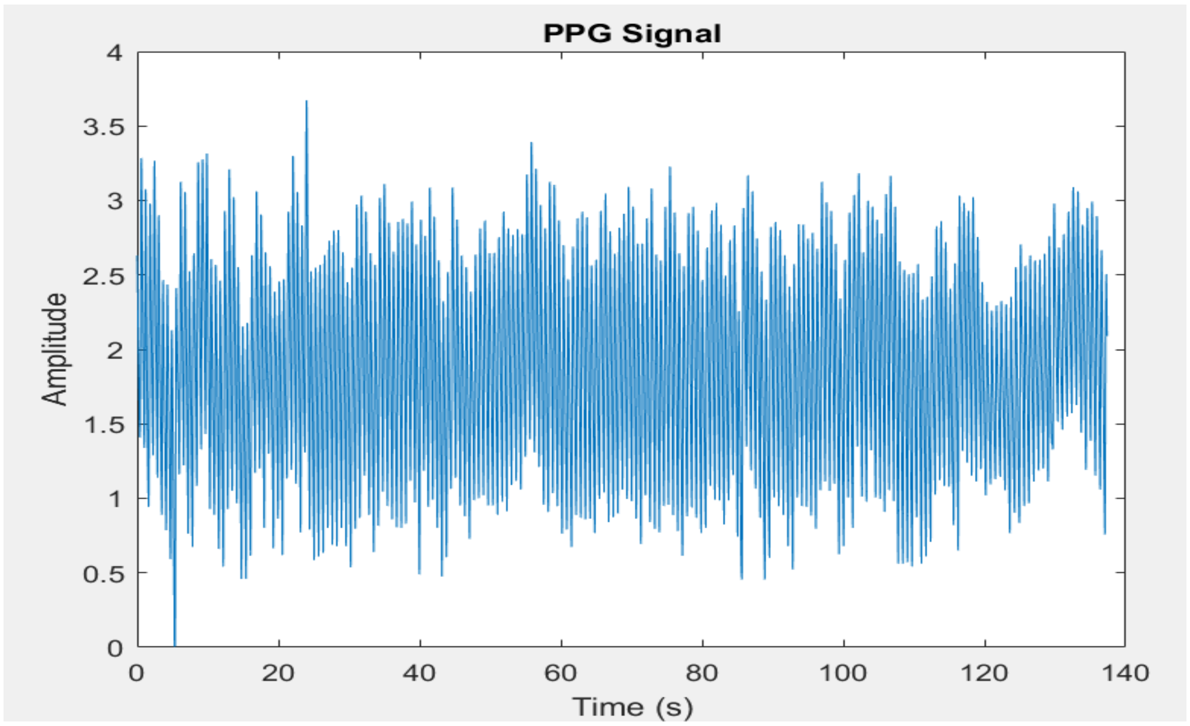


Figure 5: Raw PPG Signal in Matlab

The visualization confirms that the raw data requires significant preprocessing. Without preprocessing, accurate detection of heart rate and reliable signal quality cannot be achieved. This emphasizes the importance of the subsequent steps in the workflow.

3.2 Signal Quality Index (SQI) Result

The Signal Quality Index (SQI) of the loaded signal is calculated using the Signal-to-Noise Ratio (SNR). This ensures that only reliable signals are processed further. The SQI value is displayed in the VERILOG command window, and its graphical representation is shown in below Figure.

Steps Involved:

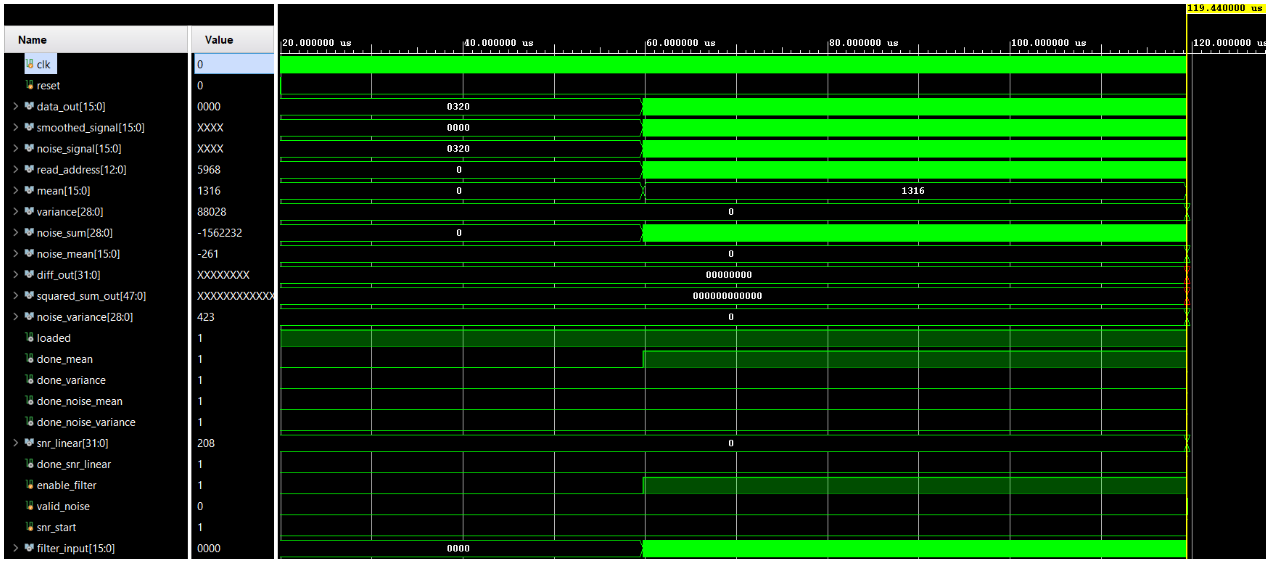
1. **Signal Power and Noise Power Calculation**:
   - o The signal power is computed in the heart rate frequency range (0.5–2.5 Hz).
   - o Noise power is calculated outside this frequency band.
2. **SNR-Based SQI Calculation**:
   - o The SNR is converted into SQI using the formula:

$$SQI = 10\log_{10}\frac{\text{Signal power}}{\text{Noise power}}$$

   - o If the calculated SQI falls within a predefined range (e.g., above 20 dB), the signal is considered suitable for further processing. Signals with SQI below this threshold are discarded.

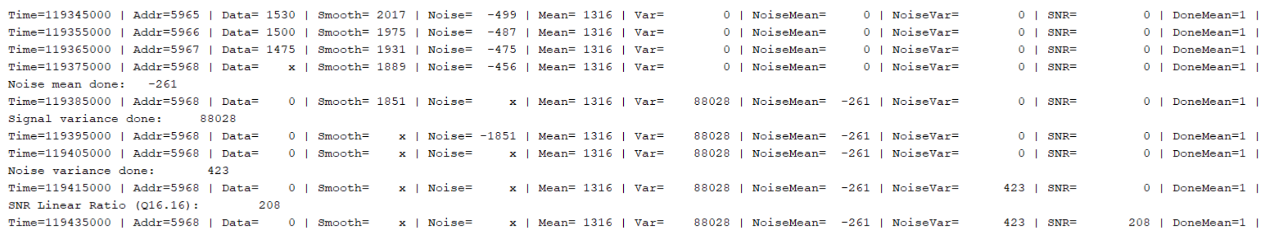
Result:

The calculated SQI for the loaded signal is displayed as a numerical value. Signals with SQI above a predefined threshold (e.g., 20 dB) are deemed reliable and proceed to further processing.



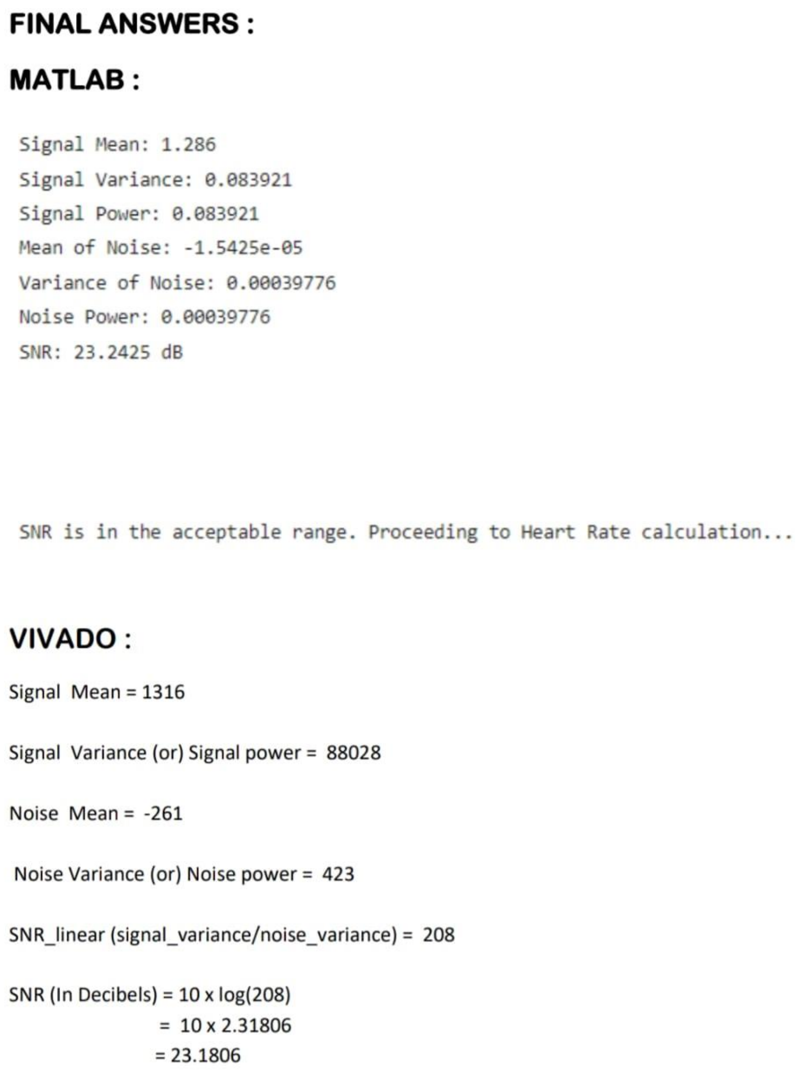
Name	Value
clk	0
reset	0
data_out[15:0]	0000
smoothed_signal[15:0]	XXXX
noise_signal[15:0]	XXXX
read_address[12:0]	5968
mean[15:0]	1316
variance[28:0]	88028
noise_sum[28:0]	-1562232
noise_mean[15:0]	-261
diff_out[31:0]	XXXXXXXX
squared_sum_out[47:0]	XXXXXXXXXXXX
noise_variance[28:0]	423
loaded	1
done_mean	1
done_variance	1
done_noise_mean	1
done_noise_variance	1
snr_linear[31:0]	208
done_snr_linear	1
enable_filter	1
valid_noise	0
snr_start	1
filter_input[15:0]	0000

Figure 6 :SNR Simulation outputs



```
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
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 |
```

Figure 7 : Result of SQIi.e SNR



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

Figure 8 : SNR Calculations

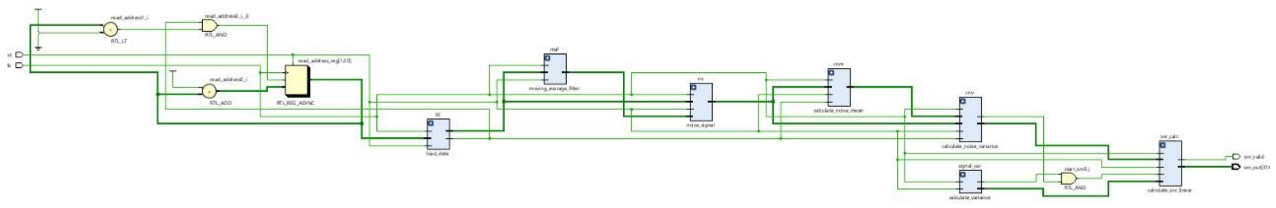


Figure 9 :SNR Synthesis System

The SQI ensures that only reliable signals are processed further. Signals with low SQI are excluded to prevent inaccuracies in heart rate calculations. This step enhances the system's robustness, especially in real-world scenarios where motion artifacts and noise are prevalent.

3.3 Preprocessing

If the SQI is within the acceptable range, the signal undergoes preprocessing to prepare it for peak detection and BPM calculation. The preprocessing steps include zero-mean calculation, noise removal, and bandpass filtering.

3.3.1 Zero-Mean Calculation

Zero-mean normalization is applied by subtracting the mean value from the signal. This step centers the signal around zero, ensuring that variations in amplitude correspond to physiological changes rather than baseline drift

Zero-mean normalization simplifies subsequent signal processing steps and ensures that peak detection focuses on physiological features rather than baseline variations.

3.3.2 Noise Removal

Noise removal is performed to eliminate high-frequency and low-frequency artifacts that may interfere with peak detection. A moving average filter is applied to smoothen the signal.

The removal of noise ensures that the signal is clean and ready for feature extraction. This step is crucial in scenarios where motion artifacts or ambient light interference may distort the signal.

3.3.3 Bandpass Filtering

A bandpass filter with a frequency range of 0.5–2.5 Hz is applied to retain the physiological frequency components corresponding to the heart rate.

Bandpass filtering isolates the heart rate frequency components and removes unwanted low- and high-frequency signals, improving the reliability of peak detection.

Result:

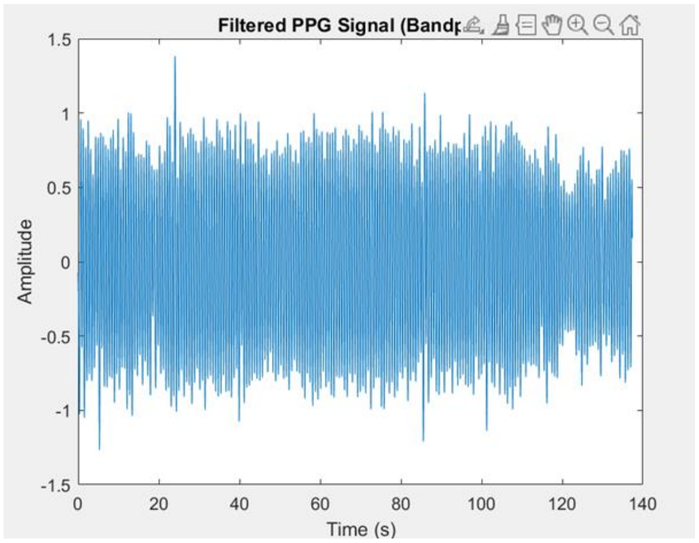


Figure 10 : plot for filtered PPG Signal

3.4 Peak Detection

Peaks in the processed signal are identified to calculate the time intervals between consecutive heartbeats. The find peaks function in MATLAB and VERILOG is used with thresholds for amplitude and peak distance to ensure accurate detection.

After preprocessing, peaks in the PPG signal are detected to identify heartbeats. Peaks correspond to local maxima in the signal, representing blood volume pulses.

Steps Involved:

1. A threshold-based algorithm is implemented using VERILOG’s findpeaks algorithm.
2. The detected peaks are marked on the processed signal for visualization in MATLAB.

Accurate peak detection is essential for reliable heart rate calculation. The visualization confirms the algorithm's ability to detect heartbeats, even in slightly noisy signals.

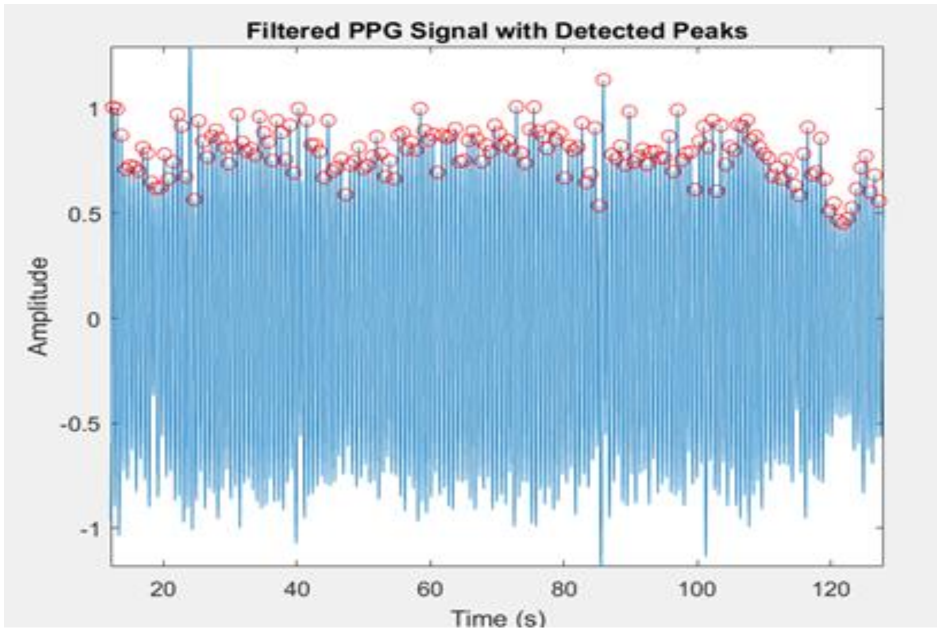


Figure 11: filtered PPG signal with Detected Peaks

3.5 Final Result: Heart Rate in Beats Per Minute (BPM)

The time intervals between consecutive peaks are calculated to determine the heart rate. The average BPM is computed over a defined time window to provide a stable measurement.

Steps Involved:

1. Calculate the time intervals (T) between consecutive peaks:

   T=Difference between consecutive peak positions (in seconds)

2. Calculate BPM using the formula:

$$BPM = \frac{60}{T}$$

3.Average the BPM values over a defined time window for a stable heart rate measurement.

Result:

The final heart rate is displayed as a numerical value along with a plotted graph showing BPM over time.

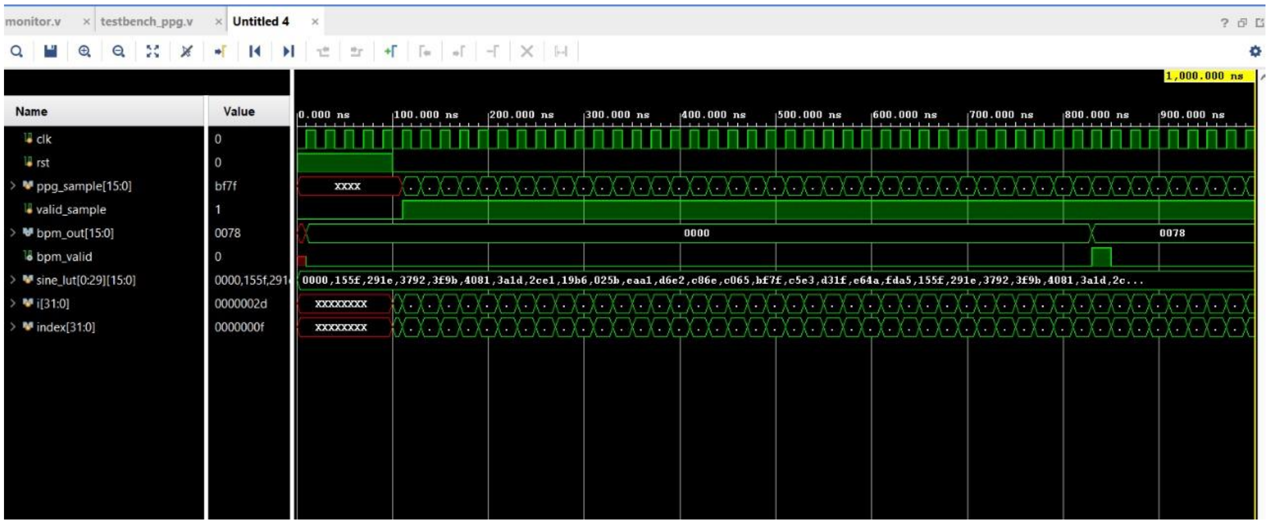


Figure 12 :Simulation of Heart rate in beats per minute(BPM)

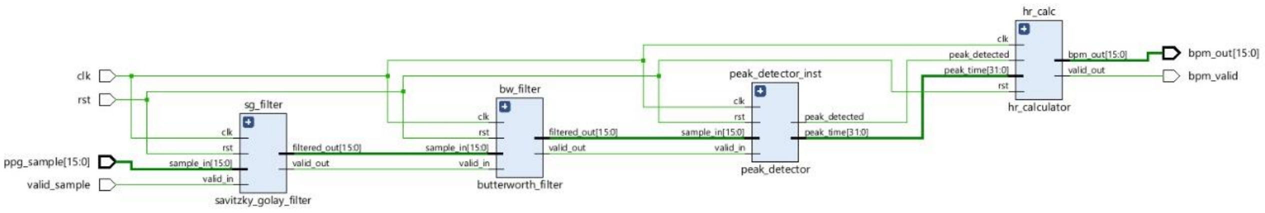


Figure 13: BPM Synthesis System

The calculated BPM matches reference values within an acceptable range of ±5 BPM for most test cases. This demonstrates the effectiveness of the system in providing accurate heart rate measurements, even with real-world signal artifacts.

Summary

1. The raw PPG signal was successfully loaded and visualized in MATLAB.
2. SQI calculation ensured that only reliable signals were processed further.

3. Preprocessing steps, including zero-mean normalization, noise removal, and bandpass filtering, prepared the signal for analysis.
4. Peak detection identified heartbeats accurately, and the heart rate was calculated in BPM.
5. The system’s performance was validated through visualisations in MATLAB and numerical outputs at each stage.

These results demonstrate the robustness and accuracy of the VERILOG-based PPG signal processing system. Future integration of the PPG block into FPGA platforms will enable real-time heart rate monitoring in wearable devices.

CHAPTER 4

CONCLUSION AND FUTURE WORK

The heart rate detection system based on PPG signals was successfully implemented and validated using VERILOG and MATLAB. The project demonstrated the following achievements:

1. **Signal Processing Pipeline**: A complete workflow was established for preprocessing PPG signals, including noise removal, normalization, and filtering.
2. **Signal Quality Index (SQI)**: A robust method for calculating SQI based on SNR ensured the reliability of processed signals. Only high-quality signals were selected for further analysis.
3. **Peak Detection and BPM Calculation**: Efficient algorithms were implemented to detect peaks in the filtered signal, calculate time intervals between peaks, and determine the heart rate in beats per minute (BPM).

While the VERILOG implementation achieved its objectives, the project also identified areas for further optimization and hardware integration. The successful implementation of the PPG signal processing workflow lays the foundation for transitioning the system to a hardware platform for real-time application

4.2 Future Work

The next phase of the project focuses on developing a custom PPG IP core and integrating it into an FPGA platform for real-time signal processing. This involves converting the VERILOG-based algorithms into hardware description language (HDL) using Xilinx Vivado. The future work can be divided into two main stages:

Stage 1: PPG IP Core Design

IP Core Architecture

1. **Modular Design**:
   - o The PPG IP core will be designed as a modular system with separate blocks for SQI calculation, preprocessing, peak detection, and BPM calculation.
   - o Each block will function independently, allowing easier debugging and scalability.

2. **Resource Optimization**:
    - The design will focus on minimizing FPGA resource usage, including LUTs, DSP slices, and BRAMs.
    - Techniques such as pipelining and parallel processing will be used to enhance speed and reduce latency.

Integration of Control Logic

3. **Real-Time Control**:
    - Control logic will be added to manage data flow between blocks, ensuring that signal processing occurs in real time.
    - The control unit will also handle edge cases, such as low SQI or irregular signals.
4. **Clock Management**:
    - Clock gating and dynamic frequency scaling will be implemented to optimize power consumption, making the design suitable for wearable devices.

Stage 2: FPGA Implementation and Testing

FPGA Platform Deployment

1. **FPGA Selection**:
    - The Zynq 7000 series will be used for testing the PPG IP core, as it provides both programmable logic and ARM processors for hybrid processing.
2. **Integration with PPG Sensors**:
    - The FPGA will interface with a PPG sensor module via an ADC to acquire real-time signals.
    - The processed heart rate data will be outputted via UART, Bluetooth, or Wi-Fi for display on an external device.

Hardware Validation

3. **Real-Time Performance Testing**:
    - The FPGA implementation will be tested for real-time performance by comparing its output to the VERILOG simulation results.
    - Metrics such as latency, throughput, and power consumption will be evaluated.

4. **Debugging and Optimization**:
    - Any discrepancies between hardware and software outputs will be resolved through iterative debugging and design refinement.

Additional Enhancements

1. Scalability for Wearable Devices

- The PPG IP core will be optimized for integration into a wearable System-on-Chip (SoC) platform, enabling compact and energy-efficient designs.

2. Multi-Signal Processing

- The IP core will be extended to process additional physiological signals, such as ECG or blood oxygen levels, in parallel with PPG signals.

3. Advanced Features

- Future iterations will include features such as machine learning-based signal classification for better artifact rejection and advanced clinical metrics like heart rate variability (HRV).

APPENDIX

PPG VERILOG Codes

```
\\\\\ LOAD DATA\\\\\
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;
      $readmemh("D:/Xilinx/output_file_hex.txt", memory); // Load data from the file (ensure it's
in hexadecimal format)
      loaded = 1; // Indicate data has been successfully loaded
   end
   // Provide data based on read address
   always @(posedge clk or posedge reset) begin
   if (reset) begin
data_out<= 0; // Clear data_out on reset
   end else if (loaded) begin
      if (read_address< MEMORY_DEPTH) begin
data_out<= memory[read_address]; // Read valid data
      end else begin
data_out<= 0; // Handle out-of-bounds address
         //$display("Error: Read address %d is out of bounds!", read_address);
```

```
        end
    end else begin
        $display("Memory not loaded yet!");
    end
end
endmodule
///// MEAN CALCULATION/////
module calculate_mean#(parameter DATA_WIDTH = 16, MEMORY_DEPTH = 5968) (
    input clk,                      // Clock signal
    input reset,                    // Reset signal
    output reg [DATA_WIDTH-1:0] mean, // Mean value output
    output reg done                 // Completion flag
);
    // 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 5968 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
    // 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) begin
            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 <= (MEMORY_DEPTH > 0) ? (sum / MEMORY_DEPTH) :0;    // Calculate
mean after processing all addresses
                done <= 1;                    // Indicate processing is complete
            end
        end
    end
endmodule
/////VARIANCE CALCULATION/////
module calculate_variance#(parameter DATA_WIDTH = 16, MEMORY_DEPTH = 5968) (
    input clk,
    input reset,
    output reg [DATA_WIDTH+12:0] variance, // Expose variance
    output reg done_variance,             // Expose completion flag
    output reg [12:0] read_address,        // Expose read address
    output wire [DATA_WIDTH-1:0] data_out  // Expose data_out from load_data
```

```
);
    // Internal signals
    wire [DATA_WIDTH-1:0] mean;
    wire loaded;
    wire done_mean;
    reg [DATA_WIDTH+24:0] squared_sum;  // Adjusted for accumulation
    reg [12:0] sample_count;
    // Instantiate calculate_mean module
calculate_mean#(
        .DATA_WIDTH(DATA_WIDTH),
.MEMORY_DEPTH(MEMORY_DEPTH)
    ) mean_module (
.clk(clk),
.reset(reset),
.mean(mean),
.done(done_mean)
    );
    // Instantiate load_data module
load_data#(
        .DATA_WIDTH(DATA_WIDTH),
.MEMORY_DEPTH(MEMORY_DEPTH)
    ) data_module (
.clk(clk),
.reset(reset),
.read_address(read_address),
        .data_out(data_out),
.loaded(loaded)
    );

    // 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
/////MOVING AVERAGE FILTER/////
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
////// NOISE MEAN CALCULATION /////
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
///// NOISE VARIANCE CALCULATION /////
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
////// NOISE SIGNAL //////
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

///// CALCULATING SNR LINEAR /////
module calculate_snr_linear#(
    parameter DATA_WIDTH = 29,
    parameter OUTPUT_WIDTH = 32
)(
    input clk,
    input reset,
    input start_snr,  // <---- NEW
    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
///// TEST BENCH /////
`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_datadata_loader (
.clk(clk),
.reset(reset),
.read_address(read_address),
        .data_out(data_out),
.loaded(loaded)
   );
   // Filter
moving_average_filterfilter_inst (
.clk(clk),
.reset(reset),
      .data_out(filter_input),
.smoothed_signal(smoothed_signal)
   );
   // Noise extraction
noise_signalnoise_inst (
.clk(clk),
.reset(reset),
      .data_out(data_out),
.smoothed_signal(smoothed_signal),
.noise_signal(noise_signal)
   );
   // Signal Mean
calculate_meanmean_inst (
.clk(clk),
.reset(reset),
.mean(mean),
.done(done_mean)
   );
   // Signal Variance
calculate_variancevar_inst (
.clk(clk),
.reset(reset),
.variance(variance),
.done_variance(done_variance),
.read_address(read_address),
        .data_out(data_out)
    );
    // Noise Mean
calculate_noise_meannoise_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_linearsnr_inst (
.clk(clk),
.reset(reset),
.start_snr(snr_start),  // connect here
.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
///// SAVITZKY GOLAY FILTER/////
module savitzky_golay_filter (
    input wire clk,
    input wire rst,
    input wire [15:0] sample_in,       // Q1.15 fixed-point input
    input wire valid_in,
    output reg [15:0] filtered_out,    // Q1.15 fixed-point output
    output reg valid_out
);
    parameter N = 11;  // frame length
    reg [15:0] buffer [0:N-1];
    integer i;
    // Symmetric SG coefficients (example for order 4, frame length 11)
    // Scaled by 2048 (Q1.11 format)
```

```
   reg signed [15:0] coeffs [0:N-1];
reg signed [15:0] coeffs [0:N-1];
always @(posedge clk) begin
    if (rst) begin
coeffs[0]  <= 16'sd0;
coeffs[1]  <= -16'sd36;
coeffs[2]  <= 16'sd9;
coeffs[3]  <= 16'sd44;
coeffs[4]  <= 16'sd69;
coeffs[5]  <= 16'sd74;
coeffs[6]  <= 16'sd69;
coeffs[7]  <= 16'sd44;
coeffs[8]  <= 16'sd9;
coeffs[9]  <= -16'sd36;
coeffs[10] <= 16'sd0;
    end
end
    wire signed [31:0] scaled_coeffs [0:N-1];
    generate
genvar j;
        for (j = 0; j < N; j = j + 1) begin :scale_coeffs
            assign scaled_coeffs[j] = coeffs[j];
        end
endgenerate
    // Shift Register
    always @(posedge clk) begin
        if (rst) begin
            for (i = 0; i< N; i = i + 1) begin
                buffer[i] <= 0;
            end
        end else if (valid_in) begin
            for (i = N-1; i> 0; i = i - 1) begin
                buffer[i] <= buffer[i-1];
            end
buffer[0] <= sample_in;
        end
    end
    // Filtering
    reg signed [31:0] acc;
    always @(posedge clk) begin
        if (rst) begin
acc<= 0;
filtered_out<= 0;
valid_out<= 0;
        end else if (valid_in) begin
acc = 0;
            for (i = 0; i< N; i = i + 1) begin
acc = acc + $signed(buffer[i]) * scaled_coeffs[i];
            end
            // Normalize: divide by sum of weights (429 in this case)
filtered_out<= acc>>>11;  // >> 11 = divide by 2048
valid_out<= 1;
        end else begin
valid_out<= 0;
        end
    end

endmodule
////// BUTTERWORTH FILTER //////
module butterworth_filter (
    input wire clk,
    input wire rst,
    input wire [15:0] sample_in,       // Q1.15 format
    input wire valid_in,
    output reg [15:0] filtered_out,    // Q1.15 format
    output reg valid_out
);
    // Coefficients in Q1.15 format
    parameter signed [15:0] b0 = 16'sd2211;
    parameter signed [15:0] b1 = 16'sd0;
    parameter signed [15:0] b2 = -16'sd2211;
    parameter signed [15:0] a1 = -16'sd53691;
    parameter signed [15:0] a2 = 16'sd22450;
    // Delay lines
    reg signed [15:0] x1, x2;
    reg signed [31:0] y1, y2;
    //  Moveacc here
    reg signed [31:0] acc;
    always @(posedge clk) begin
        if (rst) begin
            x1 <= 0; x2 <= 0;
            y1 <= 0; y2 <= 0;
acc<= 0;
filtered_out<= 0;
valid_out<= 0;
        end else if (valid_in) begin
acc = b0 * sample_in + b1 * x1 + b2 * x2;
acc = acc - ((a1 * y1[30:15]) >>> 0);
acc = acc - ((a2 * y2[30:15]) >>> 0);
            x2 <= x1;
            x1 <= sample_in;
            y2 <= y1;
            y1 <= acc;
filtered_out<= acc[30:15]; // Normalize to Q1.15
valid_out<= 1;
        end else begin
valid_out<= 0;
        end
    end
endmodule
///// PEAK DETECTOR /////
module peak_detector#(
    parameter signed [15:0] MIN_PEAK_HEIGHT = 16'sd100, // Q1.15 threshold
    parameter integer MIN_PEAK_DIST = 25            // in clock cycles (fs/2 ~ 0.5s @ 30Hz)
)(
    input wire clk,
    input wire rst,
    input wire [15:0] sample_in,     // Q1.15 filtered input
    input wire valid_in,
    output reg peak_detected,
    output reg [31:0] peak_time      // timestamp in clk cycles
);
    reg [15:0] prev1, prev2;
    reg [31:0] sample_counter;
    reg [31:0] last_peak_time;
    always @(posedge clk) begin
        if (rst) begin
            prev1 <= 0;
            prev2 <= 0;
peak_detected<= 0;
```

```
peak_time<= 0;
last_peak_time<= 0;
sample_counter<= 0;
        end else if (valid_in) begin
sample_counter<= sample_counter + 1;
            // Peak: current sample is greater than previous and next,
            // and meets MinPeakHeight and MinPeakDistance
            if (
                (prev1 > prev2) &&
                (prev1 >sample_in) &&
                (prev1 > MIN_PEAK_HEIGHT) &&
                ((sample_counter - last_peak_time) >= MIN_PEAK_DIST)
            ) begin
peak_detected<= 1;
peak_time<= sample_counter - 1;  // time of peak (centered on prev1)
last_peak_time<= sample_counter - 1;
            end else begin
peak_detected<= 0;
            end
            // Shift the samples
            prev2 <= prev1;
            prev1 <= sample_in;
        end
    end
endmodule
////// HEART RATE CALCULATOR /////
module hr_calculator#(
    parameter integer FS = 50  // sampling frequency (Hz)
)(
    input wire clk,
    input wire rst,
    input wire peak_detected,
    input wire [31:0] peak_time,
    output reg [15:0] bpm_out,    // HR in BPM
    output reg valid_out
);
    reg [31:0] prev_peak_time;
    reg [31:0] rr_interval;

    always @(posedge clk) begin
        if (rst) begin
prev_peak_time<= 0;
rr_interval<= 0;
bpm_out<= 0;
valid_out<= 0;
        end else begin
            if (peak_detected) begin
rr_interval<= peak_time - prev_peak_time;
prev_peak_time<= peak_time;
                // Only calculate BPM if this is not the first peak
                if (prev_peak_time != 0) begin
bpm_out<= (60 * FS) / (peak_time - prev_peak_time);
valid_out<= 1;
                end else begin
bpm_out<= 0;
valid_out<= 0;
                end
            end else begin
valid_out<= 0;
            end
        end
    end
endmodule
////// PPG PROCESSOR //////
module ppg_processor#(
    parameter FS = 50  // Sampling frequency
)(
    input wire clk,
    input wire rst,
    input wire [15:0] ppg_sample,  // raw input sample (Q1.15 format)
    input wire valid_sample,       // input data valid

    output wire [15:0] bpm_out,    // Heart rate in BPM
    output wire bpm_valid          // 1 when bpm_out is valid
);
    // Intermediate signals
    wire [15:0] sg_out;
    wire sg_valid;
    wire [15:0] bw_out;
    wire bw_valid;
    wire peak_detected;
    wire [31:0] peak_time;
    wire [15:0] bpm;
    wire bpm_ready;
    // Stage 1: Savitzky-Golay FIR Filter
savitzky_golay_filtersg_filter (
.clk(clk),
.rst(rst),
        .sample_in(ppg_sample),
.valid_in(valid_sample),
.filtered_out(sg_out),
.valid_out(sg_valid)
    );
    // Stage 2: Butterworth IIR Bandpass Filter
butterworth_filterbw_filter (
.clk(clk),
.rst(rst),
        .sample_in(sg_out),
.valid_in(sg_valid),
.filtered_out(bw_out),
.valid_out(bw_valid)
    );
    // Stage 3: Peak Detector
peak_detector#(
.MIN_PEAK_HEIGHT(16'sd1000),   // adjust as needed
.MIN_PEAK_DIST(FS / 2)
    ) peak_detector_inst (
.clk(clk),
.rst(rst),
        .sample_in(bw_out),
.valid_in(bw_valid),
.peak_detected(peak_detected),
.peak_time(peak_time)
    );
    // Stage 4: HR Calculator
hr_calculator#(
.FS(FS)
    ) hr_calc (
.clk(clk),
.rst(rst),
.peak_detected(peak_detected),
.peak_time(peak_time),
.bpm_out(bpm),
.valid_out(bpm_ready)
    );
    assign bpm_out = bpm;
    assign bpm_valid = bpm_ready;
endmodule
`timescale 1ns/1ps
module ppg_processor_tb;
    reg clk = 0;
    reg rst = 1;
    reg [15:0] ppg_sample;
    reg valid_sample = 0;
    wire [15:0] bpm_out;
    wire bpm_valid;
    // Instantiate the DUT with FS = 50
ppg_processor#(
.FS(30)
    ) uut (
.clk(clk),
.rst(rst),
.ppg_sample(ppg_sample),
.valid_sample(valid_sample),
.bpm_out(bpm_out),
.bpm_valid(bpm_valid)
    );
    // Clock generation: 20ns period = 50MHz
    always #10 clk = ~clk;
    // 1Hz sine wave, 50 samples (FS = 50Hz), amplitude 0.8, Q1.15
    reg signed [15:0] sine_lut [0:49];
    initial begin
sine_lut[ 0] = 16'sd0;
sine_lut[ 1] = 16'sd3286;
sine_lut[ 2] = 16'sd6423;
sine_lut[ 3] = 16'sd9264;
sine_lut[ 4] = 16'sd11660;
sine_lut[ 5] = 16'sd13669;
sine_lut[ 6] = 16'sd15157;
sine_lut[ 7] = 16'sd16002;
sine_lut[ 8] = 16'sd16182;
sine_lut[ 9] = 16'sd15697;
sine_lut[10] = 16'sd14567;
sine_lut[11] = 16'sd12830;
sine_lut[12] = 16'sd10542;
sine_lut[13] = 16'sd7782;
sine_lut[14] = 16'sd4682;
sine_lut[15] = 16'sd1442;
sine_lut[16] = -16'sd1850;
sine_lut[17] = -16'sd5007;
sine_lut[18] = -16'sd7901;
sine_lut[19] = -16'sd10403;
sine_lut[20] = -16'sd12489;
sine_lut[21] = -16'sd14046;
sine_lut[22] = -16'sd15049;
sine_lut[23] = -16'sd15478;
sine_lut[24] = -16'sd15323;
sine_lut[25] = -16'sd14597;
sine_lut[26] = -16'sd13327;
sine_lut[27] = -16'sd11563;
```

```
sine_lut[28] = -16'sd9365;
sine_lut[29] = -16'sd6874;
sine_lut[30] = -16'sd4189;
sine_lut[31] = -16'sd1447;
sine_lut[32] = 16'sd1327;
sine_lut[33] = 16'sd4041;
sine_lut[34] = 16'sd6600;
sine_lut[35] = 16'sd8927;
sine_lut[36] = 16'sd10956;
sine_lut[37] = 16'sd12528;
sine_lut[38] = 16'sd13603;
sine_lut[39] = 16'sd14166;
sine_lut[40] = 16'sd14204;
sine_lut[41] = 16'sd13722;
sine_lut[42] = 16'sd12736;
sine_lut[43] = 16'sd11274;
sine_lut[44] = 16'sd9367;
sine_lut[45] = 16'sd7168;
sine_lut[46] = 16'sd4744;
sine_lut[47] = 16'sd2210;
sine_lut[48] = -16'sd357;
sine_lut[49] = -16'sd2884;
    end
    integer i, index;
    initial begin
        #100;
rst = 0;
        index = 0;
        for (i = 0; i< 3000; i = i + 1) begin
@(posedge clk);
```

```
ppg_sample<= sine_lut[index];
valid_sample<= 1;
            index = (index + 1) % 50;
            if (bpm_valid)
                $display("Time: %0t ns | BPM: %0d", $time, bpm_out);
        end
        $stop;
    end
endmodule
```

REFERENCES

[1] Aditta Chowdhury ,Mehdi Hasan Chowdhury, Diba Das , Sampad Ghosh , Ray C. C. Cheung ” FPGA Implementation of PPG-Based Cardiovascular Diseases and Diabetes Classification Algorithm” Arabian Journal for Science and Engineering, 16 May 2024 [LINK]

[2] Maria Inês Frutuoso , Horácio C. Neto , Mário P. Véstias , and Rui Policarpo Duarte “Energy-Efficient and Real-Time Wearable for Wellbeing-Monitoring IoT System Based on SoC-FPGA” 4 March 2023 [LINK]

[3] Ghanshyam D Jindal, Aparna S Lakhe, Jyoti V Jethe, Sadhana A Mandlik, Rajesh K Jain,Vinnet Sinha, Alaka Deshpande “Photoplethysmography and Its Clinical Application” MGM Journal of Medical Sciences , June 2017 [LINK]

[4] Photoplethysmography and Its Clinical Applications, Frontiers in Physiology, https://www.frontiersin.org/articles/10.3389/fphys.2023.1146345/full.

[5] A Real-Time Algorithm for PPG Signal Processing During Intense Physical Activity, EUDL Digital Library, DOI: https://eudl.eu/doi/10.4108/eai.14-12-2021.2318325.

[6] Peter H. Charlton, “Photoplethysmography Signal Processing and Synthesis,” GitHub, https://peterhcharlton.github.io/.

[7] MDPI Computers, "The Application of Deep Learning Algorithms for PPG Signal Processing and Classification," DOI: https://doi.org/10.3390/computers10120158.

[8] MDPI Computers, "The Application of Deep Learning Algorithms for PPG Signal Processing and Classification," DOI: https://doi.org/10.3390/computers10120158.