DSP BASED QRS DETECTION USING PAN-TOMKINS ALGORITHM

MINI PROJECT REPORT

Submitted by

ABDUL FALAH MAJEED (TKM22EC001)

ABHIJITH U (TKM22EC003)

BIBIN BABY (TKM22EC038)

SALMANUL FARIS PV (TKM22EC116)

to

APJ Abdul Kalam Technological University

In partial fulfillment of the requirements for the award of the Degree of

Bachelor of Technology in Electronics & Communication

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TKM COLLEGE OF ENGINEERING, KOLLAM 691005

APRIL 2025

DECLARATION

We, hereby declare that the mini-project report "**DSP BASED QRS DETECTION USING PAN-TOMKINS ALGORITHM**" submitted for partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Electronics and Communication Engineering of APJ Abdul Kalam Technological University, Kerala, is a bonafide work done by us. This submission represents our ideas in our own words, and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained.

Place: KOLLAM
Date: 00-00-25

ABDUL FALAH MAJEED (TKM22EC001)
ABHIJITH U (TKM22EC003)
BIBIN BABY (TKM22EC038)
SALMANUL FARIS PV (TKM22EC116)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TKM COLLEGE OF ENGINEERING

KOLLAM 691005

CERTIFICATE

This is to certify that the mini project report entitled **"DSP BASED QRS DETECTION USING PAN-TOMKINS ALGORITHM**" is a bonafide record of the work presented by **ABDUL FALAH MAJEED (TKM22EC001) , ABHIJITH U (TKM22EC003), BIBIN BABY (TKM22EC038), SALMANUL FARIS PV (TKM19EC103)** to APJ Abdul Kalam Technological University in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in Electronics and Communication Engineering during the academic year 2024-2025 under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Dr. Nissan Kunju	**Dr. Nissan Kunju**	**Dr. Nishanth N.**
Mini-Project Coordinator	Mini-Project Guide	Head of the Department

ACKNOWLEDGEMENT

We take this opportunity to convey our deep sense of gratitude to all those who have been kind enough to offer their advice and assistance when needed which has led to the successful completion of this project. First of all, we thank God Almighty for all His blessings throughout this endeavour without which it would not have been possible.

It is our privilege and pleasure to express our profound sense of respect, gratitude and indebtedness to Dr. Nishanth N, Head of Department of Electronics and Communication Engineering for guiding and providing facilities for the successful completion of the preliminary phase of our project work.

We sincerely thank Dr. Nissan Kunju, Project coordinator, Associate Professor, Department of Electronics and Communication Engineering for his guidance, valuable support and constant encouragement given to us during this work.

We take this opportunity to express our sincere gratitude to our guide, Dr. Nissan Kunju, Associate Professor, Department of Electronics and Communication Engineering, for her advice, guidance, and reference materials provided during the preparation of the preliminary phase of our project work.

We are also thankful to all the faculty members of the Department of Electronics and Communication Engineering for their guidance and wholehearted support. Their effort and sincerity will always be remembered in our heart.

Last, but not the least, we wish to acknowledge our parents and friends for giving us moral strength and encouragement throughout this period.

ABDUL FALAH MAJEED (TKM22EC001)

ABHIJITH U (TKM22EC003)

BIBIN BABY (TKM22EC038)

SALMANUL FARIS PV (TKM22EC116)

ABSTRACT

Our project focuses on detecting QRS complexes in ECG signals using the Pan-Tompkins algorithm implemented on the TMS320C6748 DSP processor. QRS detection is a key step in cardiac analysis, enabling heart rate monitoring and arrhythmia detection. The Pan-Tompkins algorithm processes raw ECG signals through a series of filters, differentiation, squaring, and thresholding stages to identify R-peaks. Initially, the algorithm was verified using MATLAB to understand signal behavior and performance. After successful simulation, we translated the algorithm into embedded C code and implemented it on the DSP using Code Composer Studio. The ECG input was fed to the processor, and output was validated against MATLAB results.

To evaluate performance, we tested our implementation with the MIT-BIH Arrhythmia Database and with custom noisy ECG recordings. The results were analysed and compared with the MATLAB output to ensure accuracy. The DSP-based implementation offers real-time performance, low power consumption, and is ideal for wearable healthcare applications. Compared to software simulations, our hardware solution demonstrates improved efficiency for real-time environments. The TMS320C6748's processing speed and hardware-level control played a crucial role in handling biomedical signals effectively. This project highlights how embedded signal processing can be applied in portable ECG monitoring systems.

Keywords – ECG Signal, QRS Detection, Pan-Tompkins Algorithm, TMS320C6748, Code Composer Studio, Biomedical Signal Processing

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

1. **ADC** - Analog-to-Digital Converter
2. **BPM** - Beats Per Minute
3. **CCS** - Code Composer Studio
4. **DDR** - Double Data Rate
5. **DMA** - Direct Memory Access
6. **DSP** - Digital Signal Processor
7. **ECG** - Electrocardiogram
8. **EDMA** - Enhanced Direct Memory Access
9. **FFT** - Fast Fourier Transform
10. **FIR** - Finite Impulse Response
11. **FPGA** - Field-Programmable Gate Array
12. **GPIO** - General-Purpose Input/Output
13. **HDL** - Hardware Description Language
14. **HPF** - High-Pass Filter
15. **IDE** - Integrated Development Environment
16. **IIR** - Infinite Impulse Response
17. **IoT** - Internet of Things
18. **JTAG** - Joint Test Action Group
19. **LPF** - Low-Pass Filter
20. **MIT-BIH** - Massachusetts Institute of Technology-Beth Israel Hospital (Arrhythmia Database)
21. **NPKI** - Noise Peak Index
22. **QRS** - Q, R, and S waves in an ECG complex
23. **RAM** - Random Access Memory
24. **ROM** - Read-Only Memory
25. **RR Interval** - Time between successive R-peaks in an ECG
26. **SPI** - Serial Peripheral Interface
27. **SPKI** - Signal Peak Index
28. **SRAM** - Static Random Access Memory

29. **THD** - Total Harmonic Distortion
30. **TI** - Texas Instruments
31. **UART** - Universal Asynchronous Receiver-Transmitter
32. **VLIW** - Very Long Instruction Word

CHAPTER OF ORGANIZATION

Chapter 1: Introduction – Introduces the problem of accurate ECG analysis and the significance of QRS detection in cardiac monitoring. Outlines the goal of implementing the Pan-Tompkins algorithm on a DSP platform for offline ECG signal processing.

Chapter 2: Literature Review – Reviews prior work on ECG signal analysis, QRS detection algorithms, and previous implementations on digital signal processors and embedded platforms.

Chapter 3: Methodology – Describes the use of the MIT-BIH Arrhythmia Database and custom noisy ECG recordings. Covers MATLAB simulation of the Pan-Tompkins algorithm and the step-by-step signal processing pipeline: filtering, differentiation, squaring, and thresholding.

Chapter 4: The Pan-Tompkins Algorithm - Explains the complete working of the Pan-Tompkins algorithm for QRS detection in ECG signals, including bandpass filtering, derivative operation, squaring, moving window integration, adaptive thresholding, T-wave discrimination, and search-back mechanism.

Chapter 5: TMS320C6748 Embedded DSP - Describes the architecture, core features, and technical specifications of the TMS320C6748 processor. Highlights its suitability for biomedical signal processing applications due to its floating-point support, low power consumption, and efficient parallel processing capabilities.

Chapter 6: Algorithm and Code Flow – Presents detailed explanation of each processing block, C code structure, and how the ECG signal is handled internally. Includes debugging and verification steps on the DSP.

Chapter 7: Results and Discussion – Analyses output from the DSP and compares it with MATLAB results. Highlights detection accuracy for MIT-BIH and custom ECG inputs under varying noise conditions.

Chapter 8: Tools Used – Lists tools like MATLAB, Code Composer Studio, CCS emulator, and the TMS320C6748 DSP development board.

Chapter 9: Conclusion and Future Scope – Summarizes the success of DSP-based ECG QRS detection and suggests enhancements such as onboard storage, automated noise handling, or integration into portable diagnostic devices.

CHAPTER 1
INTRODUCTION

1.1 Background

Electrocardiography (ECG) is a widely used non-invasive technique for monitoring cardiac activity, providing crucial information for the diagnosis and treatment of various heart conditions. The accurate detection of QRS complexes, particularly R-peaks, in ECG signals is fundamental to heart rate calculation and arrhythmia detection. However, ECG signals are often contaminated with various forms of noise, including powerline interference, muscle artifacts, baseline wander, and motion artifacts, which complicate the accurate detection of these cardiac events.

The Pan-Tompkins algorithm, introduced in 1985, remains one of the most robust and widely-implemented approaches for real-time QRS detection in ECG signals. This algorithm employs a series of digital filtering and decision rule techniques to enhance the QRS complex while suppressing noise components, making it particularly suitable for implementation on digital signal processors (DSPs). The algorithm processes ECG signals through several stages: bandpass filtering, differentiation, squaring, moving window integration, and adaptive thresholding for QRS detection.

Traditional software implementations of the Pan-Tompkins algorithm on general-purpose computers are resource-intensive and may not be suitable for portable or wearable ECG monitoring devices where power consumption, processing speed, and memory usage are critical constraints. Dedicated hardware implementations using digital signal processors (DSPs) offer significant advantages in terms of power efficiency, processing speed, and potential for real-time applications.

1.2 Problem Statement

Despite the effectiveness of the Pan-Tompkins algorithm, its implementation on DSP platforms presents several challenges. These include

optimizing the algorithm for constrained hardware resources, maintaining detection accuracy across diverse ECG morphologies, and ensuring reliable performance in the presence of various noise artifacts. Additionally, there is a need to validate the DSP implementation using standard databases such as the MIT-BIH Arrhythmia Database as well as custom recordings to ensure robust performance in real-world scenarios.

Most existing implementations focus either on software simulations or fully-customized hardware designs, with limited literature on optimized implementations for specific DSP platforms like the TMS320C6748. Furthermore, the transition from algorithm design to practical hardware deployment requires addressing issues related to signal acquisition, data format conversion, computational efficiency, and result verification that are often overlooked in theoretical studies.

This project addresses these gaps by implementing the complete Pan-Tompkins algorithm on the TMS320C6748 DSP platform, evaluating its performance with both standard MIT-BIH database signals and custom ECG recordings captured using an AD8232 sensor module.

1.3 Objectives

The main objectives of this project are:

1. To implement the Pan-Tompkins QRS detection algorithm on the TMS320C6748 DSP platform for offline ECG signal processing.
2. To develop a methodology for preprocessing and standardizing ECG signals from both the MIT-BIH Arrhythmia Database and custom recordings obtained using the AD8232 sensor module.
3. To evaluate the performance of the DSP implementation in terms of detection accuracy, processing efficiency, and robustness to noise, using both standard and custom ECG recordings.

4. To compare the results obtained from MATLAB simulations with those from the DSP implementation to validate the correctness of the hardware-based approach.
5. To document the complete design process, implementation challenges, and optimization techniques for reference in future biomedical signal processing applications on DSP platforms.

1.4 Motivation

Cardiovascular diseases remain the leading cause of mortality worldwide, making effective cardiac monitoring technologies crucial for early detection and intervention. Portable and wearable ECG monitoring devices powered by efficient DSP implementations can significantly enhance the accessibility of cardiac health monitoring, particularly in resource-constrained environments or remote settings. The implementation of the Pan-Tompkins algorithm on DSP platforms aligns with the growing trend toward edge computing in healthcare, where signal processing and preliminary diagnosis occur directly on the acquisition device rather than requiring transmission to centralized servers. This approach reduces latency, enhances privacy, and enables operation in areas with limited connectivity.

Moreover, mastering the implementation of biomedical signal processing algorithms on DSP platforms provides valuable insights into the practical challenges and optimization techniques required for real-world biomedical engineering applications. The knowledge gained from this project contributes to bridging the gap between theoretical algorithm development and practical deployment in healthcare devices.

Finally, by validating the DSP implementation with both standard databases and custom recordings, this project addresses the often-overlooked challenge of transferring algorithms from controlled research environments to real-world applications where signal quality and characteristics may vary significantly.

CHAPTER 2

LITERATURE REVIEW

The detection of QRS complexes from ECG signals is a crucial step in the analysis and diagnosis of cardiac conditions. Over the years, many algorithms have been proposed to ensure accurate and real-time QRS detection, each with different trade-offs in terms of computational complexity, noise resilience, and implementation feasibility. This chapter provides a detailed review of three significant studies that have directly influenced the development and implementation approach of this project.

[1] Pan, J., & Tompkins, W. J. (1985). ***A Real-Time QRS Detection Algorithm*****. IEEE Transactions on Biomedical Engineering.**

This groundbreaking paper presented one of the first real-time QRS detection algorithms suitable for computer implementation. The Pan-Tompkins algorithm processes ECG signals through a sequence of stages:

- Bandpass Filtering: Reduces both high-frequency noise and baseline wander, effectively isolating the QRS frequency range (5–15 Hz).
- Differentiation: Highlights the rapid changes in the QRS complex slope, making it easier to identify.
- Squaring Function: Makes all data points positive and amplifies the higher-frequency components related to the QRS complex.
- Moving Window Integration: Aggregates energy over a fixed interval, helping identify the full width of the QRS complex.
- Adaptive Thresholding and Peak Detection: Dynamically adjusts the decision threshold based on the signal's characteristics, enhancing robustness against noise and baseline shifts.

The algorithm achieved over 99% accuracy on the MIT-BIH Arrhythmia Database, which was a significant breakthrough at the time. Importantly, its low computational cost made it feasible for real-time implementation, which is why it remains a standard for QRS detection in both software and hardware applications today.

[2] Lourenço, A., Silva, H., Leite, P., Lourenço, R., & Fred, A. (2019). ***Review and Comparison of QRS Detection Algorithms for Arrhythmia Diagnosis*****. IEEE Conference Publication.**

This survey paper offers a comprehensive overview of modern QRS detection algorithms, comparing both traditional signal-processing-based and emerging machine-learning-based techniques. The review highlights:

- Evaluation Metrics: Sensitivity, specificity, detection error rate, and computational time were analyzed across various datasets, including MIT-BIH, European ST-T, and real-world noisy recordings.

- Comparison of Techniques: Pan-Tompkins, Wavelet Transforms, Hilbert Transform, and Support Vector Machines (SVM)-based detection algorithms were compared.
- Suitability for Embedded Systems: While machine learning methods (e.g., deep learning models) often outperform in terms of accuracy, they tend to require substantial computational resources and are less suited for real-time embedded applications.
- Conclusion: The paper concludes that traditional algorithms like Pan-Tompkins still offer the best balance between accuracy and resource usage for practical implementations, especially on microcontrollers and DSPs.

This review reinforced the choice of Pan-Tompkins for this project, as it remains the most reliable and implementable algorithm in constrained environments, particularly on embedded DSP platforms like the TMS320C6748.

[3] Pavlatos, C., Dimopoulos, A., Manis, G., & Papakonstantinou, G. (2005). *Hardware Implementation of Pan & Tompkins QRS Detection Algorithm*. National Technical University of Athens

This study directly addresses the practical challenges of implementing the Pan-Tompkins algorithm in hardware. The authors designed a system on FPGA and DSP platforms and provided insight into:

- Fixed-Point Arithmetic Optimization: To reduce memory usage and computation time while preserving signal fidelity.
- Pipeline Architecture Design: Each stage of the algorithm was designed to work in a pipeline structure for faster processing.
- Threshold Adaptation in Hardware: A major challenge addressed was the real-time computation of dynamic thresholds in limited-resource environments, which the authors tackled using efficient memory lookup tables and conditional logic blocks.
- Timing and Latency Analysis: The total delay introduced by hardware stages was kept minimal to maintain real-time performance.

The findings from this research are highly relevant to the DSP-based simulation performed in this project. The strategies discussed for optimizing memory, execution time, and arithmetic precision were applied during the C-based implementation and testing on the TMS320C6748 processor.

CHAPTER 3

METHODOLOGY

This chapter presents a detailed methodology followed for implementing the Pan-Tompkins algorithm for QRS detection on the TMS320C6748 DSP. It includes dataset collection, preprocessing, MATLAB simulation, DSP implementation, and testing using both standard and custom ECG signals.

3.1 Methodology Flow Chart

The methodology flow chart shown in the figure below outlines the step-by-step procedure adopted for offline ECG signal processing. It begins with data collection, followed by preprocessing, algorithm simulation in MATLAB, and final deployment on the DSP hardware for verification.

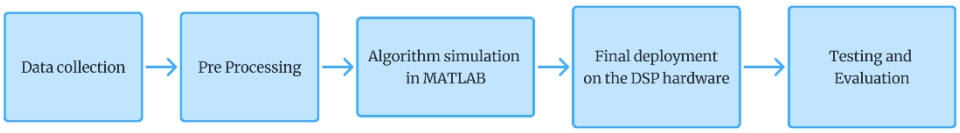


Figure 3.1 Methodology Flow Chart

3.2 Dataset Collection

Two types of ECG datasets were used in this project. The first was the MIT-BIH Arrhythmia Database, a clinically annotated and widely accepted benchmark dataset for QRS detection algorithms. ECG recordings from this dataset were downloaded and converted into .txt format for further processing. The second dataset was a custom ECG recording obtained using an AD8232 analog front-end ECG sensor module. This sensor was connected to a microcontroller, and raw ECG signals were transmitted to a computer using serial communication. Python scripts with the PySerial library were used to capture and store these signals in .txt files.

3.3 Signal Characteristics

The MIT-BIH signals had a sampling rate of 360 Hz. To maintain uniformity, the custom ECG recordings were resampled to match this sampling rate. The MIT-BIH dataset covered a variety of arrhythmias, while the custom dataset included signals with real-world noise such as motion artifacts, baseline drift, and powerline interference.

3.4 Preprocessing

In MATLAB, preprocessing involved applying a bandpass filter to remove baseline wander and high-frequency noise. The filter typically passed frequencies in the range of 5 to 15 Hz. The filtered signal was then processed through the next stages of the Pan-

Tompkins algorithm. Custom ECG signals, being noisier, underwent an additional visual inspection to ensure that prominent features like the QRS complex were retained after filtering.

3.5 MATLAB Simulation

The Pan-Tompkins algorithm was fully implemented in MATLAB. The steps included bandpass filtering, derivative filtering to highlight QRS slopes, squaring to enhance signal energy, and moving window integration to smooth the waveform. Finally, thresholding was applied to detect R-peaks. The algorithm was validated by plotting outputs from each stage and comparing the detected R-peaks with annotations provided in the MIT-BIH dataset. For custom signals, the outputs were manually verified.

3.6 DSP Implementation

After successful simulation in MATLAB, the algorithm was converted into embedded C and implemented in Code Composer Studio. The program was first tested using the functional simulator for the TMS320C6748 DSP. Graphs for filtered signal, integrated signal, and detected R-peaks were visualized using CCS's graph tool, while heart rate and R-peak indices were printed to the console. After simulation verification, the code was deployed to the actual DSP hardware. Both MIT-BIH and custom ECG recordings were loaded offline into the DSP. The DSP processed the signals and output R-peak locations and heart rate successfully, even for noisy custom recordings.

CHAPTER 4

PAN-TOMKINS ALGORITHM

4.1 Overview

QRS detection is difficult, not only because of the physio logical variability of the QRS complexes, but also because of the various types of noise that can be present in the ECG signal. Noise sources include muscle noise, artifacts due to electrode motion, power-line interference, baseline wander, and T waves with high-frequency characteristics similar to QRS complexes. In our approach, digital filters reduce the influence of these noise sources, and there by improve the signal-to-noise ratio. Of the many QRS detectors proposed in the literature, few give serious enough attention to noise reduction.

The algorithm uses a dual-threshold technique to find missed beats, and thereby reduce false negatives. There are two separate threshold levels in each of the two sets of thresholds. One level is half of the other. The thresholds continuously adapt to the characteristics of the signal since they are based upon the most-recent signal and noise peaks that are detected in the ongoing processed signals. If the program does not find a QRS complex in the time interval corresponding to 166 percent of the current average RR interval, the maximal peak detected in that time interval that lies between the two thresholds is considered to be a possible QRS complex, and the lower of the two thresholds is applied. In this way, we avoid requiring a long memory buffer for storing the past history of the ECG, and thus require minimal computing time to accomplish the search-back procedure to look for a missing QRS complex.

Once a valid QRS complex is recognized, there is a 200ms refractory period before the next one can be detected since QRS complexes cannot occur more closely than this physiologically. This refractory period eliminates the possibility of a false detection such as multiple triggering on the same QRS complex during this time interval. When a QRS detection occurs following the end of the refractory period but within 360ms of the previous complex, we must determine if it is a valid QRS complex or a T wave. In this case, we judge the waveform with the largest slope to be the QRS complex. To be reliable, a QRS detection algorithm must adapt each of its parameters with time so as to be able to operate properly for ECG's of different patients as well as for ECG morphology changes in a single patient. In our algorithm, each threshold automatically adapts periodically based upon peak values of signal and noise. When a QRS must be found using second (lower) thresholds, threshold readjustment occurs twice as fast as usual. In the dual-threshold technique, the RR-interval average must be updated for each heartbeat.

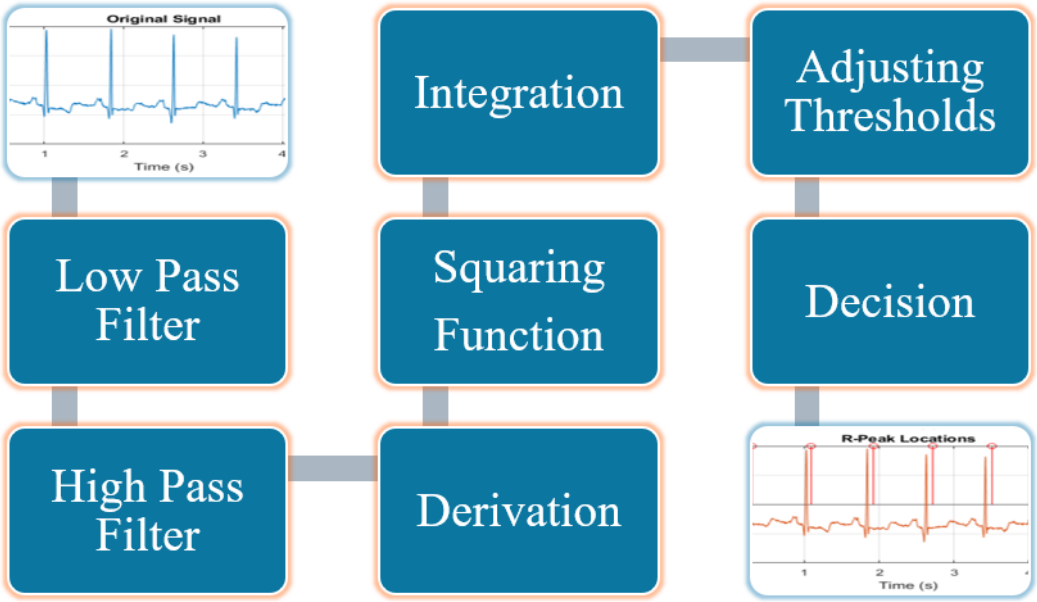


Figure 4.1 Block Diagram of the Pan-Tompkins Algorithm

4.2 Bandpass Filtering

The bandpass filter reduces the influence of muscle noise, 60Hz interference, baseline wander, and T-wave interference. The desirable passband to maximize the QRS energy is approximately 5-15Hz. Our filter is a fast, real-time recursive filter in which poles are located to cancel zeros on the unit circle of the z-plane.

The low pass filter has the recursive equation:

$$y(nT) = 2y(nT - T) - y(nT - 2T) + x(nT) - 2x(nT - 6T) + x(nT - 12T)$$

The high pass filter has the recursive equation:

$$y(nT) = 32x(nT - 16T) - y(nT - T) - x(nT) + x(nT - 32T)$$

4.3 Derivative Filter

The derivative of the input signal is taken to obtain the information of the slope of the signal. Thus, the rate of change of input is obtain in this step of the algorithm.

The derivative filter has the equation:

$$y(nT) = [-x(nT - 2T) - 2x(nT - T) + 2x(nT + T) + x(nT + 2T)]/(8T)$$

4.4 Squaring

The squaring process is used to intensify the slope of the frequency response curve obtained in the derivative step. This step helps in restricting false positives which may be caused by T waves in the input signal.

The squaring filter has the equation:

$$y(nT) = [x(nT)]^2$$

4.5 Moving Window Integration

The moving window integration process is done to obtain information about both the slope and width of the QRS complex. A window size of 0.15*(sample frequency) is used for more accurate results.

The moving window integration has the recursive equation:

$y(nT) = [y(nT - (N-1)T) + x(nT - (N-2)T) + ... + x(nT)]/N$

where N is the number of samples in the width of integration window.

4.6 Peak Detection

- **Fiducial Mark:** An approximate location of the QRS complex can be obtained in the initial phase of detection by sensing the rising edge of the integration waveform. Since, a peak is determined by the change in slope of the curve, the differentiated signal is used to determine the fiducial marks.
- **Adjusting Thresholds:** Since the signal to noise ratio is improved by the bandpass filter, two sets thresholds are maintained to account for low threshold values. The higher thresholds of each set are used to detect peaks in the first analysis and the lower thresholds in the search back process. The thresholds are adjusted accordingly to account for the detected signal peaks and noise values.
- **Adjusting RR Interval and Limits:** To keep track of the time between two successive R peaks, two RR intervals are maintained. The first RR interval keeps track of the eight most recent beats while the second RR interval keeps track of the eight most recent beats having RR intervals that fall within the rate limits. Two averages pertaining to these RR intervals are calculated. These averages are then used to update the rate limits for the RR intervals. If a QRS complex is not found within the calculated limits a search back process is initiated to find the maximal peak value within the two calculated thresholds and this peak is taken to be a QRS candidate.
- **T Wave Identification:** If the calculated RR interval is less than 360ms, which in this case is the sample frequency of the input ECG signal, then the maximal slope of this waveform is calculated. This is done to ensure that the interval in consideration is actually a QRS complex or a T wave. If the calculated maximal slope of the considered interval is less than half of the slope of the last QRS complex detected, then the current interval is considered to be a T wave.

After the successful detection of the R peaks, the heart rate of an individual can be calculated by considering the time difference between successive R peaks. The heart rate can be calculated as:

Heart Rate = 60/RR Interval (in seconds)

CHAPTER 5

TMS320C6748 EMBEDDED DSP

This chapter provides an overview of the Texas Instruments TMS320C6748 digital signal processor, which was used for implementing the Pan-Tompkins algorithm in this project. The TMS320C6748 is a low-power, high-performance processor that supports both fixed-point and floating-point operations, making it suitable for real-time and offline biomedical signal processing applications such as ECG analysis.

5.1 Overview of TMS320C6748

The TMS320C6748 is part of the C6000 DSP family and is designed for applications that require high-speed data processing with low power consumption. It features a 32-bit DSP core capable of executing up to 3600 MIPS and supports floating-point arithmetic, which simplifies algorithm development in applications involving biomedical signals. It is often used in medical, industrial, and audio signal processing domains.

5.2 Key Features and Specifications

Core: C674x floating-/fixed-point DSP

Clock Speed: Up to 456 MHz

Architecture: Very Long Instruction Word (VLIW)

Instruction Width: 8 parallel instructions per cycle

On-chip RAM: 256 KB L2 RAM

External Memory Interfaces: DDR2/DDR3, NAND, NOR

Peripheral Support: McBSP, McASP, UART, SPI, I2C, GPIO

Timers and EDMA: Supports multiple timers and enhanced DMA controller

Power Management: Supports low power modes for energy-efficient operation

Development Tools: Supported by Code Composer Studio (CCS) IDE

5.3 Architecture Highlights

The C6748 DSP uses a VLIW architecture that allows multiple instructions to be executed simultaneously, increasing throughput and computational efficiency. The DSP supports both 16-bit fixed-point and 32-bit floating-point operations, which is crucial for signal processing tasks that require precision and speed.

The TMS320C674X DSP CPU consists of eight functional units, two register files, and two data paths as shown in Figure. The two general-purpose register files (A and B) each contain 32 32- bit registers for a total of 64 registers. The general-purpose registers can be used for data or can be data address pointers. The data types supported include

packed 8-bit data, packed 16-bit data, 32-bit data, 40- bit data, and 64-bit data. Values larger than 32 bits, such as 40-bit-long or 64-bit-long values are stored in register pairs, with the 32 LSBs of data placed in an even register and the remaining 8 or 32 MSBs in the next upper register (which is always an odd-numbered register). The eight functional units (.M1, .L1, .D1, .S1, .M2, .L2, .D2, and .S2) are each capable of executing one instruction every clock cycle. The .M functional units perform all multiply operations. The .S and .L units perform a general set of arithmetic, logical, and branch functions. The .D units primarily load data from memory to the register file and store results from the register file into memory.

The processor is divided into multiple functional units that allow parallel execution of operations such as multiplication, addition, logic, and shifting. This makes it highly optimized for algorithms like filtering, differentiation, and integration — key steps in the Pan-Tompkins pipeline.

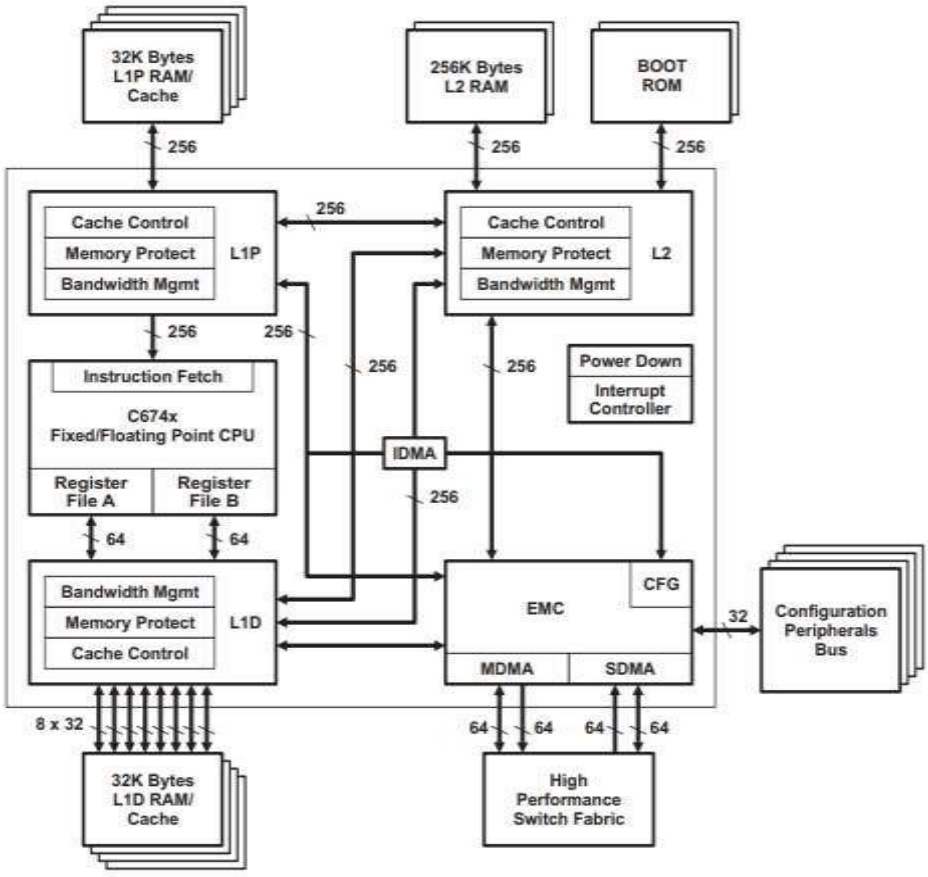


Figure 5.1 Block Diagram of TMS320C6748 DSP Core

CHAPTER 6

SOFTWARE DESCRIPTION

This chapter provides an overview of the software used for the implementation of the Pan-Tompkins-based QRS detection algorithm on the TMS320C6748 DSP. It covers the algorithm flow, code logic, deployment steps, and the verification of results.

6.1 Algorithm Flow (Code Logic)

The Pan-Tompkins QRS detection algorithm consists of multiple stages that are executed in sequence to detect R-peaks from the ECG signal. The stages are as follows:

Low-pass Filter: Removes high-frequency noise from the ECG signal.

High-pass Filter: Removes baseline wander from the ECG signal.

Derivative: Highlights the QRS complex.

Squaring: Enhances the R-peaks for easier detection.

Integration: Smooths the squared signal to prepare for peak detection.

Peak Detection: The algorithm identifies the R-peaks based on a thresholding technique, using SPKI (Signal Peak Index) and NPKI (Noise Peak Index).

The algorithm also calculates the heart rate based on the RR intervals between consecutive R-peaks.

6.2 MATLAB Code Implementation

The MATLAB code was initially used for simulating and validating the Pan-Tompkins algorithm. The following MATLAB code snippet demonstrates the key operations of the algorithm.

matlab

```
% PAN TOMPKINS ALGORITHM

clc;

close all;

clear all;

% FUNCTION DEFINITIONS



% FILTERS

% LOW PASS FILTER

function y = low_pass_filter(x)

   N = length(x); % Length of the input sequence

   y = zeros(size(x)); % Initialize output

   for n = 13:N

      y(n) = 2 * y(n-1) - y(n-2) + x(n) - 2 * x(n-6) + x(n-12);

   end

end
```

```
% HIGH PASS FILTER
function y = high_pass_filter(x)
    N = length(x); % Length of the input sequence
    y = zeros(size(x)); % Initialize output
    for n = 33:N
        y(n) = y(n-1) - (1/32) * x(n) + x(n-16) - x(n-17) + (1/32) * x(n-32);
    end
end

% DERIVATIVE
function y = derivative(x)
    N = length(x); % Length of the input sequence
    y = zeros(size(x)); % Initialize output
    for n = 5:N  % Start at n=5 because of the x(n-4) term
        y(n) = (1/8) * (2*x(n) + x(n-1) - x(n-3) - 2*x(n-4));
    end
end

% SQUARING
function y = square(x)
```

```
    N = length(x); % Length of the input sequence
    y = zeros(size(x)); % Initialize output
    for n = 1:N
        y(n) = x(n)^2; % Square each sample
    end
end


% SLIDING WINDOW INTEGRATION
function y = integration(x)
    % Formula: y(n) = (1/30) * sum(x(n-29), ..., x(n))
    % Initialize variables
    N = length(x);
    y = zeros(1, N); % Output signal
    % Define the window size (30 samples as per the equation)
    window_size = 30;
    % Apply the integration filter
    for n = window_size:N
        % Sum the last 30 samples and divide by 30 to get the average
        y(n) = sum(x(n - window_size + 1:n)) / window_size;
    end
```

```
    % For the first few samples (less than 30), the filter cannot apply fully,
    % so we just set the output to zero
    y(1:window_size-1) = 0;
end


% PEAK DETECTION
function [r_peaks, heart_rate] = peak_detection(x, fs)
    % PEAK_DETECTION_HW - Implements hardware-compatible R-peak detection with adaptive thresholds.
    % Inputs:
    %   x  - Integrated ECG signal after preprocessing
    %   fs - Sampling frequency (Hz)
    % Outputs:
    %   r_peaks - Detected R-peak indices
    %   heart_rate - Estimated heart rate (bpm)

    N = length(x);
    r_peaks = []; % Store detected R-peak locations
    rr_intervals = zeros(1, 8); % Store last 8 RR intervals
```

```
% Initial threshold values

SPKI = max(x) * 0.25;  % Initial signal peak estimate

NPKI = max(x) * 0.125; % Initial noise peak estimate



THRESHOLD1 = NPKI + 0.25 * (SPKI - NPKI);

THRESHOLD2 = 0.5 * THRESHOLD1; % Secondary threshold for search-back



last_peak = 0;



for n = 2:N-1

   % Check if current sample is a local maximum (peak detection)

   if x(n) > x(n-1) && x(n) > x(n+1)

      PEAKI = x(n); % Detected peak



      if PEAKI > THRESHOLD1

         % Ensure minimum RR interval (to reject noise & T-waves)

         if isempty(r_peaks) || (n - last_peak) > round(0.6 * fs)

            r_peaks = [r_peaks, n]; % Store R-peak index

            last_peak = n;
```

```
            % Update SPKI (signal peak estimate)
            SPKI = 0.125 * PEAKI + 0.875 * SPKI;
        end
    else
        % Update NPKI (noise peak estimate)
        NPKI = 0.125 * PEAKI + 0.875 * NPKI;
    end

    % Update Thresholds
    THRESHOLD1 = NPKI + 0.25 * (SPKI - NPKI);
    THRESHOLD2 = 0.5 * THRESHOLD1;
  end
end

% Compute Heart Rate
if length(r_peaks) > 1
  rr_intervals = diff(r_peaks) / fs;
  heart_rate = 60 ./ rr_intervals; % Convert RR intervals to BPM
else
  heart_rate = 0;
```

```
    end
end


% MAIN PROGRAM
[tm,signal,fs,labels] = rdmat('100m');
ch1 = signal(:,1);
ch2 = signal(:,2);
t = tm(1:1800);
s1 = ch1(1:1800);
s2 = ch2(1:1800);


x = s1;
figure;
subplot(3,3,1);
plot(t, x);
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Signal');
grid on;
```

```
% Apply Low-Pass Filter
y = low_pass_filter(x);
subplot(3,3,2);
plot(t, y);
xlabel('Time (s)');
ylabel('Amplitude');
title('Filtered Signal (Low-Pass)');
grid on;

% Apply High-Pass Filter
y = high_pass_filter(y);
subplot(3,3,3);
plot(t, y);
xlabel('Time (s)');
ylabel('Amplitude');
title('Filtered Signal (High-Pass)');
grid on;

% Apply Derivative
y = derivative(y);
```

```
subplot(3,3,4);

plot(t, y);

xlabel('Time (s)');

ylabel('Amplitude');

title('Derivative');

grid on;



% Squaring

y = square(y);

subplot(3,3,5);

plot(t, y);

xlabel('Time (s)');

ylabel('Amplitude');

title('Squared');

grid on;



% Integration

y = integration(y);

subplot(3,3,6);

plot(t, y);
```

```
xlabel('Time (s)');

ylabel('Amplitude');

title('Integration');

grid on;


% Peak Detection

[r_peaks, heart_rate] = peak_detection(y, fs);  % Fixed variable y5 -> y


% Plot Integrated ECG Signal and Detected Peaks in subplot (not a new figure)

subplot(3,3,7);

plot(t, y, 'b'); % Plot integrated ECG signal

hold on;

plot(t(r_peaks), y(r_peaks), 'ro', 'MarkerFaceColor', 'r', 'MarkerSize', 8); % Plot detected peaks

xlabel('Time (s)');

ylabel('Amplitude');

title('Peak Detection on Integrated ECG Signal');

legend('Integrated Signal', 'Detected R-Peaks', 'Location', 'southeast');

grid on;

hold off;
```

```
% Final plot: Peaks visualization
subplot(3,3,8);
z = ones(size(y)); % Ensure z matches the signal length
stem(t(r_peaks), z(r_peaks), 'r');
hold on;
plot(t, x);
xlabel('Time (s)');
ylabel('Amplitude');
title('R-Peak Locations');
grid on;
hold off;
```

This MATLAB code demonstrates the filtering, derivative, squaring, integration, and peak detection stages of the Pan-Tompkins algorithm. The output is displayed in the form of plots that visualize each stage of the process and the detected R-peaks.

6.3 Embedded C Code (DSP)

The embedded C code was implemented on the TMS320C6748 DSP to detect R-peaks from the ECG signal in real time. The following are key functions implemented in the DSP code:

Low-pass and High-pass Filters: Perform filtering operations on the input ECG signal.

Derivative, Squaring, and Integration: Process the filtered signal to enhance the QRS complex for peak detection.

Peak Detection: Identifies R-peaks using a thresholding method.

The following is a snippet of the DSP C code:

c

```
// final working code rev a | gave full path | removed endless while loop



/*

got out put like this :

Processing complete.

Heart Rate: 73.79 BPM

R peaks detected at following indices:

108 392 694

*/



#include <math.h>

#include <string.h>

#include <stdio.h>



#define FS 360.0f          // Sampling frequency in Hz
```

```
#define WINDOW_SIZE 30     // Integration window size
#define BUFFER_SIZE 900    // Number of ECG samples
#define MAX_PEAKS 100      // Maximum number of peaks


// Buffers
float ecg_signal[BUFFER_SIZE];
float y_lp[BUFFER_SIZE];
float y_hp[BUFFER_SIZE];
float y_der[BUFFER_SIZE];
float y_sq[BUFFER_SIZE];
float y_int[BUFFER_SIZE];
int r_peak_binary[BUFFER_SIZE];
int r_peaks[MAX_PEAKS];
float heart_rate = 0.0f;
int num_peaks = 0;


// Filters and stages
void low_pass_filter(float *x, float *y, int N) {
    int n;
    memset(y, 0, N * sizeof(float));
```

```
    for (n = 12; n < N; n++) {
        y[n] = 2 * y[n - 1] - y[n - 2] + x[n] - 2 * x[n - 6] + x[n - 12];
    }
}

void high_pass_filter(float *x, float *y, int N) {
    int n;
    memset(y, 0, N * sizeof(float));
    for (n = 32; n < N; n++) {
        y[n] = y[n - 1] - (1.0f / 32.0f) * x[n] + x[n - 16] - x[n - 17] + (1.0f / 32.0f) * x[n -
32];
    }
}

void derivative(float *x, float *y, int N) {
    int n;
    memset(y, 0, N * sizeof(float));
    for (n = 4; n < N; n++) {
        y[n] = (1.0f / 8.0f) * (2 * x[n] + x[n - 1] - x[n - 3] - 2 * x[n - 4]);
    }
```

```
}


void square(float *x, float *y, int N) {
    int n;
    for (n = 0; n < N; n++) {
        y[n] = x[n] * x[n];
    }
}


void integration(float *x, float *y, int N) {
    int n, i;
    memset(y, 0, N * sizeof(float));
    for (n = WINDOW_SIZE - 1; n < N; n++) {
        float sum = 0.0f;
        for (i = 0; i < WINDOW_SIZE; i++) {
            sum += x[n - i];
        }
        y[n] = sum / WINDOW_SIZE;
    }
}
```

```
void peak_detection(float *x, int N, float fs, int *r_peaks, int *r_peak_binary, int *num_peaks, float *heart_rate) {
    float SPKI = 0.0f, NPKI = 0.0f;
    float THRESHOLD1, THRESHOLD2;
    int last_peak = 0;
    *num_peaks = 0;
    int n, i;

    memset(r_peak_binary, 0, N * sizeof(int));

    float max_val = x[0];
    for (i = 1; i < N; i++) {
        if (x[i] > max_val) max_val = x[i];
    }

    SPKI = 0.25f * max_val;
    NPKI = 0.125f * max_val;
    THRESHOLD1 = NPKI + 0.25f * (SPKI - NPKI);
    THRESHOLD2 = 0.5f * THRESHOLD1;
```

```
for (n = 1; n < N - 1; n++) {
    if (x[n] > x[n - 1] && x[n] > x[n + 1]) {
        float PEAKI = x[n];
        if (PEAKI > THRESHOLD1) {
            if (*num_peaks == 0 || (n - last_peak) > (int)(0.6f * fs)) {
                if (*num_peaks < MAX_PEAKS) {
                    r_peaks[*num_peaks] = n;
                    r_peak_binary[n] = 1;
                    (*num_peaks)++;
                    last_peak = n;
                    SPKI = 0.125f * PEAKI + 0.875f * SPKI;
                }
            }
        } else {
            NPKI = 0.125f * PEAKI + 0.875f * NPKI;
        }
        THRESHOLD1 = NPKI + 0.25f * (SPKI - NPKI);
        THRESHOLD2 = 0.5f * THRESHOLD1;
    }
```

```
    }


    if (*num_peaks > 1) {
        float rr_sum = 0.0f;
        for (i = 1; i < *num_peaks; i++) {
            float rr = (float)(r_peaks[i] - r_peaks[i - 1]) / fs;
            rr_sum += 60.0f / rr;
        }
        *heart_rate = rr_sum / (*num_peaks - 1);
    } else {
        *heart_rate = 0.0f;
    }
}


int main() {
    FILE *fp = fopen("C:/Users/imsal/Desktop/pat/PanTomSim/ecg_signal.txt", "r");
    int i;
    if (fp == NULL) {
        printf("Error opening ecg_data.txt\n");
        return 1;
```

```
    }

    for (i = 0; i < BUFFER_SIZE; i++) {
        if (fscanf(fp, "%f", &ecg_signal[i]) != 1) {
            printf("Error reading sample %d\n", i);
            break;
        }
    }
    fclose(fp);

    low_pass_filter(ecg_signal, y_lp, BUFFER_SIZE);
    high_pass_filter(y_lp, y_hp, BUFFER_SIZE);
    derivative(y_hp, y_der, BUFFER_SIZE);
    square(y_der, y_sq, BUFFER_SIZE);
    integration(y_sq, y_int, BUFFER_SIZE);
    peak_detection(y_int, BUFFER_SIZE, FS, r_peaks, r_peak_binary, &num_peaks,
&heart_rate);

    printf("Processing complete.\n");
    printf("Heart Rate: %.2f BPM\n", heart_rate);
```

```
    printf("R peaks detected at following indices:\n");
    for (i = 0; i < num_peaks; i++) {
        printf("%d ", r_peaks[i]);
    }

    //while (1); // Stay here for debugging
    return 0;
}
```

This C code implements the same algorithm stages as in the MATLAB code, but is optimized for real-time processing on the TMS320C6748 DSP.

6.4 Toolchain & Deployment

For deploying the Pan-Tompkins-based QRS detection algorithm onto the TMS320C6748 DSP, Code Composer Studio (CCS) served as the primary integrated development environment. The complete toolchain involved the following stages:

6.4.1. Compiler Usage

The TI C6000 compiler was used to convert the C code implementation of the algorithm into optimized assembly instructions for the TMS320C6748 DSP. Special compiler flags were configured to enhance real-time performance and reduce latency, essential for ECG signal processing tasks. The compiler also handled intrinsic DSP functions that utilize the VLIW architecture of the C6748 processor.

6.4.2. Linker Role

The TI linker generated an executable .out file by combining object files and linking them with the necessary runtime libraries and memory configuration scripts (.cmd files). The linker ensured correct memory mapping to match the TMS320C6748 memory architecture, placing code, data, and stack in their respective memory regions (L2 SRAM, DDR2, etc.).

6.4.3. CCS Integration

Code Composer Studio (CCS) was used for:

- Editing and compiling the C code into object files.
- Project configuration, specifying the target device, compiler options, and build settings.
- Flashing the binary into the DSP memory and initiating real-time execution.
- Debugging, using breakpoints, watch variables, and real-time graphs to monitor signal data (like R-peak detection) on the fly.
- Profiling and performance analysis, ensuring the algorithm met real-time constraints.

6.4.4. Deployment Process

- The final .out file was loaded onto the TMS320C6748 board using the JTAG interface.
- The real-time ECG data was fed into the DSP via onboard ADC or preloaded datasets.
- CCS's debugger was used to validate and analyze the output, such as QRS complex detection and heart rate calculation, confirming algorithm accuracy.

This toolchain ensured a smooth transition from algorithm prototyping in MATLAB to real-time embedded implementation on the DSP, demonstrating both functional correctness and computational efficiency.

.

6.5 Output Verification

The MATLAB analysis was conducted initially to simulate and validate the R-peak detection and heart rate calculation algorithms. Afterward, the same algorithm was implemented on the TMS320C6748 DSP kit from Texas Instruments, where the R-peaks were successfully detected in real-time. This was accomplished with the help of Code Composer Studio for simulation and deployment. The results from the DSP implementation aligned with the MATLAB analysis, confirming the accuracy and effectiveness of the system in detecting R-peaks and calculating the heart rate within a real-time embedded environment.

CHAPTER 7

RESULTS

This chapter summarizes the outcomes of the Pan-Tompkins QRS detection system. The algorithm was initially validated using MATLAB simulations based on standard ECG datasets. Following successful simulation, the same algorithm was ported and implemented on the TMS320C6748 DSP platform using Code Composer Studio (CCS) to enable real-time QRS detection. The results from both software and hardware implementations are presented and compared through signal plots and peak detections.

7.1 MATLAB Simulation Results

The Pan-Tompkins algorithm was implemented in MATLAB and tested with MIT-BIH ECG datasets. The signal underwent several stages of preprocessing, including low-pass filtering, high-pass filtering, differentiation, squaring, and moving-window integration. The adaptive threshold-based peak detection method was then applied.

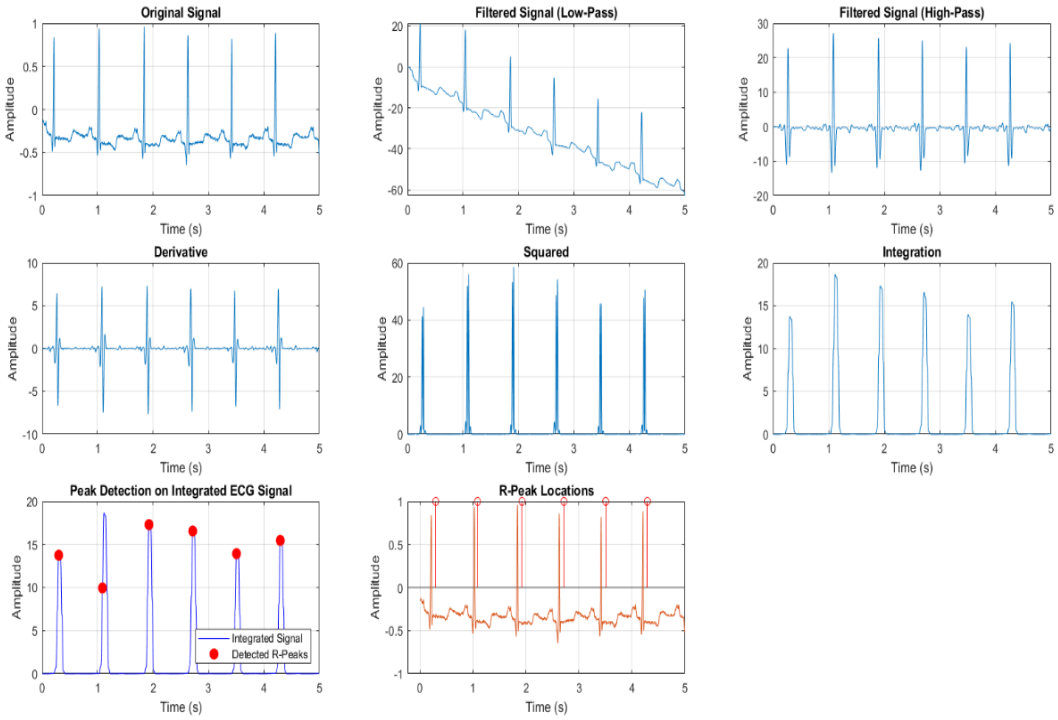


Figure 7.1 Simulations using MATLAB

7.2 DSP Hardware Implementation

The real-time implementation was carried out on the Texas Instruments TMS320C6748 DSP using Code Composer Studio. The preprocessed ECG signal was fed to the DSP board via UART interface, and the R-peaks were detected using the same algorithm logic written in C.

The detected R-peaks were transmitted back to the PC for plotting and validation. The real-time plots show that the DSP is capable of detecting R-peaks accurately with minimal latency, validating the algorithm's suitability for embedded healthcare applications.

Figure 7.2 TMS320C6748

Figure 7.3 Simulations using Dsp kit

7.3 Comparison of MIT-BIH and Real-Time Noisy ECG Data

To assess the performance and robustness of the Pan-Tompkins QRS detection algorithm, a comparative study was conducted using two types of ECG data:

- MIT-BIH Arrhythmia Database – a widely used standard dataset with clean ECG signals.
- Real-Time ECG Data from AD8232 Sensor – practical ECG signals with inherent noise and baseline wander.

The objective was to evaluate how effectively the algorithm detects R-peaks under both ideal and noisy conditions.

7.3.1 R-Peak Detection on MIT-BIH Data

The DSP simulation using the MIT-BIH dataset shows clear ECG morphology with sharp R-peaks. The algorithm processes the signal through various stages (filtering, differentiation, squaring, and integration), and adaptive thresholding helps accurately identify the R-peaks.

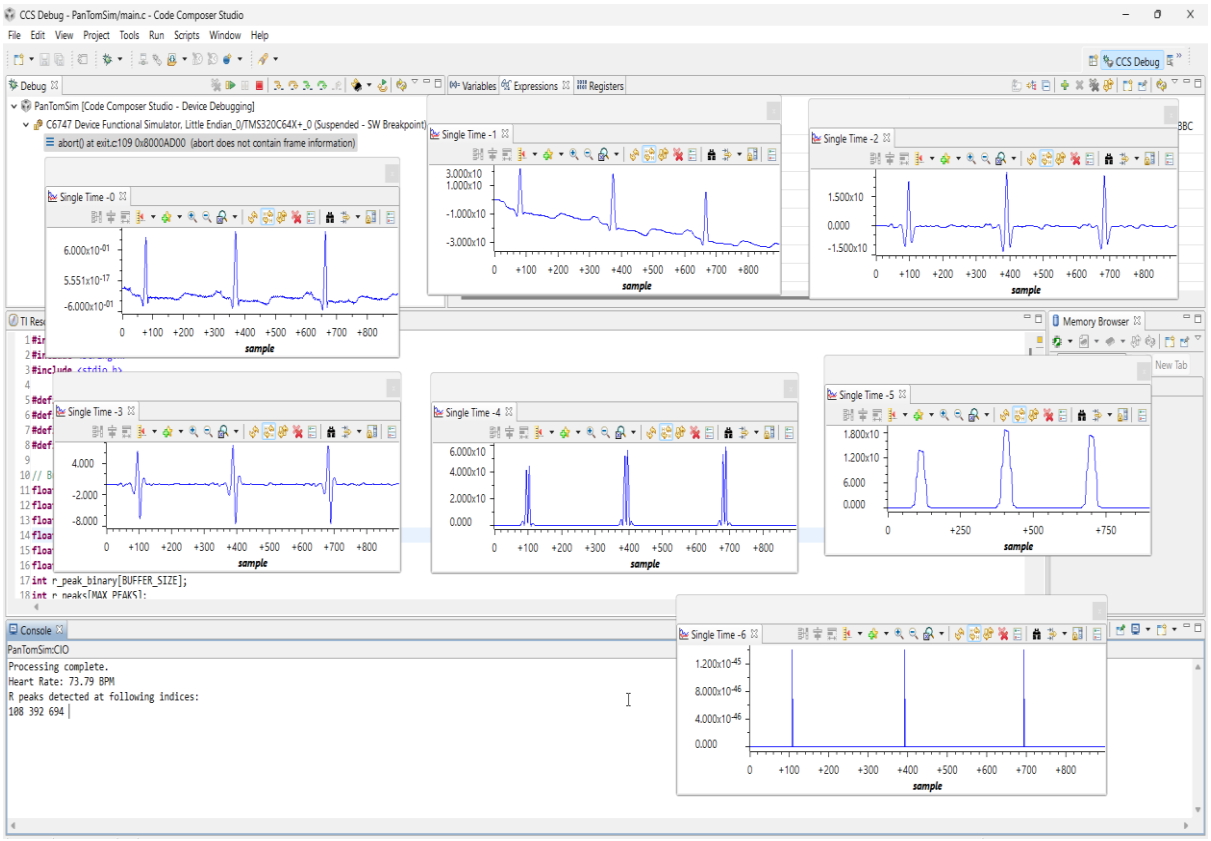


Figure 7.4 R-Peak Detection from MIT-BIH Dataset using DSP Kit

7.3.2 R-Peak Detection on Noisy Real-Time ECG (AD8232)

The real-time ECG signal obtained from the AD8232 module contains noise due to muscle artifacts and ambient interference. Despite the degraded signal quality, the same algorithm implemented on the DSP successfully detects the R-peaks with reasonable accuracy. This demonstrates the algorithm's ability to adapt to variable signal quality in real-world conditions.

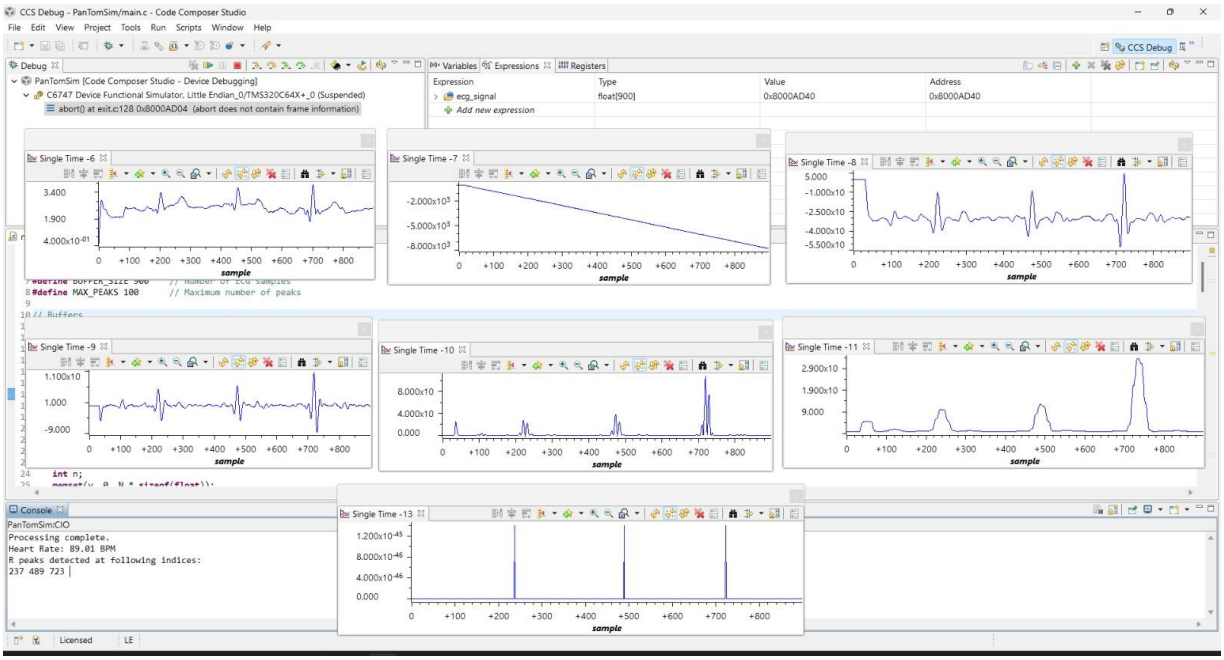


Figure 7.5 R-Peak Detection from Noisy AD8232 ECG using DSP kit

7.3.3 Comparative Analysis

The Pan-Tompkins algorithm was applied to both clean (MIT-BIH) and noisy (AD8232) ECG signals for R-peak detection. The comparison of the results shows how the algorithm performs under ideal conditions (clean ECG) versus in the presence of noise(noisyECG).

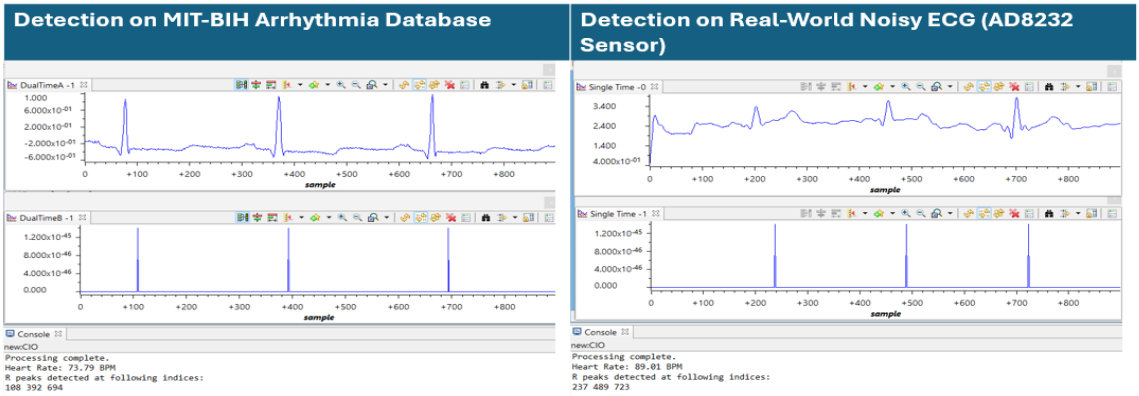
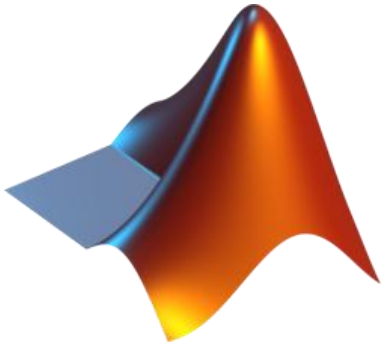


Figure 7.6 R-Peak Detection Performance – Clean vs. Noisy ECG Signal

CHAPTER 8

TOOLS USED

8.1 MATLAB



MATLAB is a high-level programming environment used for algorithm development, data analysis, visualization, and numerical computation. It is widely adopted in academic and industrial settings, especially in fields like signal processing and biomedical engineering.

In this project, MATLAB was used for:
- Designing and testing the Pan-Tompkins algorithm using sample ECG signals.
- Simulating the complete signal processing pipeline, including filtering, differentiation, squaring, integration, and thresholding.
- Visualizing intermediate results to evaluate the performance of each stage before hardware implementation.
- Validating the detection of QRS complexes and ensuring the accuracy of the algorithm under varying signal conditions.

8.2 CODE COMPOSER STUDIO (CCS)

Code Composer Studio is an integrated development environment (IDE) provided by Texas Instruments for embedded system development on TI processors. It offers a complete suite of tools for coding, debugging, compiling, and hardware interaction.

In this project, CCS was used for:
- Writing and compiling the Pan-Tompkins algorithm in C for the TMS320C6748 DSP processor.
- Emulating the algorithm using the onboard JTAG emulator for preliminary testing before hardware deployment.
- Deploying the code onto the actual DSP hardware via JTAG interface.
- Performing real-time debugging, stepping through code, monitoring variables, and evaluating processing performance on-chip.

The use of CCS enabled seamless integration of algorithm logic into the DSP environment, essential for real-time ECG signal processing.

8.3 PYTHON

Python is a versatile programming language widely used for data acquisition, preprocessing, and automation. With a rich ecosystem of libraries, it enables efficient interaction with sensors and hardware.

In this project, Python was used for:
- Acquiring ECG signals from the AD8232 analog front-end sensor module connected to a microcontroller or serial interface.
- Using libraries such as `pyserial` to read real-time signal data via serial communication.
- Ensuring that the captured ECG waveform was suitable for QRS detection and further algorithm testing.

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

9.1 Conclusion

In this project, the Pan-Tompkins QRS detection algorithm was successfully implemented on the TMS320C6748 digital signal processing (DSP) platform. The implementation was thoroughly tested using both benchmark ECG signals from the MIT-BIH Arrhythmia Database and real-time noisy ECG recordings acquired via the AD8232 sensor. The system demonstrated robust and accurate detection of R-peaks, even in the presence of significant signal noise and variability. These results confirm the practicality and reliability of the algorithm for real-world biomedical signal processing tasks. The deployment on a dedicated DSP highlights the suitability of this hardware for efficient, real-time ECG processing in embedded health-monitoring applications.

9.2 Future Scope

The successful implementation of the Pan-Tompkins algorithm opens the door to several enhancements and future developments. Some of the potential directions include:

- Real-time, wearable ECG monitoring: The algorithm's robustness in handling noisy signals makes it highly suitable for integration into wearable ECG monitoring systems. These could be used for continuous health tracking in home care and ambulatory settings.

- Wireless and IoT integration: By incorporating wireless communication modules, the DSP platform can transmit detected R-peak data to cloud or mobile applications, enabling remote cardiac monitoring and real-time health alerts.

- Multichannel ECG processing: Future work could expand the system to handle multiple ECG leads, improving diagnostic capabilities for more complex arrhythmias and cardiac conditions.

- Adaptive thresholding and machine learning: Enhancing the algorithm with adaptive or AI-assisted decision-making can improve accuracy under dynamic physiological conditions and further reduce false positives in noisy environments.

- Miniaturized hardware deployment: The current implementation can be miniaturized for integration into portable or wearable medical devices, allowing for low-power, high-efficiency on-device signal processing.

REFERENCES

1. Pan, J., & Tompkins, W. J., *A real-time QRS detection algorithm*, IEEE Transactions on Biomedical Engineering, Vol. BME-32, No. 3, pp. 230–236, 1985.
2. Lourenço et al., *Review and comparison of QRS detection algorithms for arrhythmia diagnosis*, IEEE Conference Publication, 2019.
3. Pavlatos et al., *Hardware implementation of Pan & Tompkins QRS detection algorithm*, National Technical University of Athens, 2005.

PO MAPPING

TITLE	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12
Design and Simulation …..	3	3	3	3	3	1	0	3	3	3	1	3

PO1 – Engineering Knowledge: The implementation of the Pan-Tompkins algorithm required applying foundational knowledge of digital signal processing, embedded systems, and biomedical instrumentation, specifically in ECG signal acquisition and filtering techniques.

PO2 – Problem Analysis: The project involved identifying the challenges of QRS detection in noisy ECG signals. Effective analysis and filtering techniques were applied to enhance accuracy and reduce false detections.

PO3 – Design/Development of Solutions: A real-time QRS detection system was designed and implemented on the TMS320C6748 DSP. The algorithm was optimized for embedded deployment, demonstrating improved accuracy on both clean and noisy ECG signals.

PO4 – Conduct Investigations of Complex Problems: Investigations were conducted using standard datasets (MIT-BIH) and real-time signals from the AD8232 sensor. Comparative testing on both sources validated the robustness of the system under diverse signal conditions.

PO5 – Modern Tool Usage: Modern tools such as MATLAB for algorithm simulation, Python for signal acquisition, and Code Composer Studio for DSP deployment were used effectively throughout the development process.

PO6 – The Engineer and Society: This project contributes to the healthcare domain by enabling low-cost, real-time cardiac monitoring solutions, especially useful in remote and resource-constrained settings.

PO7 – Environment and Sustainability: By utilizing software-based solutions and embedded hardware, the project promotes sustainable, low-power medical devices suitable for long-term monitoring without significant environmental impact.

PO8 – Ethics: Proper citations were provided for any referenced datasets or algorithms. Ethical practices were maintained in both implementation and documentation.

PO9 – Individual and Team Work: The project was collaboratively executed, demonstrating strong teamwork, task delegation, and integration of diverse skills in hardware, software, and biomedical signal processing.

PO10 – Communication: The objectives, methods, and outcomes of the project were clearly communicated through technical documentation, reports, and presentations, ensuring clear understanding among stakeholders.

PO11 – Project Management and Finance: Project milestones were efficiently planned and executed using a timeline. All tools used, including the licensed version of MATLAB, were managed within available resources.

PO12 – Lifelong Learning: The project required self-learning of DSP architecture, biomedical algorithms, and real-time processing techniques, fostering independent learning and adaptability to evolving technologies.