HARDWARE IMPLEMENTATION OF REAL-TIME BEAT DETECTION AND CLASSIFICATION ALGORITHM FOR AUTOMATED ECG ANALYSIS

by

Ria Ghosh


APPROVED BY SUPERVISORY COMMITTEE:

_____

Dr. Lakshman S. Tamil, Chair

_____

Dr. Mehrdad Nourani

_____

Dr. Diana Cogan

To my Mom, Dad and Sister,

Thank you for always being there for me.

# HARDWARE IMPLEMENTATION OF REAL-TIME BEAT DETECTION AND CLASSIFICATION ALGORITHM FOR AUTOMATED ECG ANALYSIS

by

RIA GHOSH, BE

THESIS

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN

COMPUTER ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS

August 2018

# ACKNOWLEDGMENTS

I would like to express my special gratitude to my advisor, Dr. Lakshman S. Tamil, for his constant guidance that helped me successfully complete this research and grow as a researcher. His advice on both research and my career have been invaluable.

I would like to thank Cheng Si and Vignesh Kalidas, Dr. Tamil's PhD students, who helped me to be on course whenever I faced problems during the entire time of my work.

I would also like to thank my committee members, Professor Mehdrad Nourani and Dr. Diana Cogan, for serving on my committee and for their brilliant comments and suggestions.

Above all, I would like to thank my family and all my friends for being supportive, understanding and showing faith in me always.

April 2018

HARDWARE IMPLEMENTATION OF REAL-TIME BEAT DETECTION AND CLASSIFICATION ALGORITHM FOR AUTOMATED ECG ANALYSIS

Ria Ghosh, MS
The University of Texas at Dallas, 2018

Supervising Professor: Dr. Lakshman S. Tamil

The epidemics of diabetes and obesity, along with unhealthy and stressful lifestyles, have highly contributed to the increased number of patients with heart failure in recent times. As the saying goes, “Prevention is better than cure”, detecting heart abnormalities accurately in initial stages can save patients from severe consequences and expensive surgeries. Hence, in the past few years there has been extensive research in beat detection and real-time cardiac monitoring to determine algorithms that can detect heart beat location and analyze whether the distance between two beats are normal or not. Such a regular check on the health of the heart using a device that could give real-time cardiac monitoring outside the hospital helps to ensure early diagnosis of any kind of abnormality that the cardiac system of an individual might be facing or is prone to face in the near future.

Various QRS complex detecting algorithms have been implemented into smart watches and fitness trackers, which has led to the commercialization of various wearable heart beat monitoring devices that have been effective to quite an extent. However, various factors like unwanted noise and inconsistency in differentiating beat locations, may reduce the accuracy of such devices. Hence, it

is necessary to ensure that any algorithm maintains accurate precision during both software and hardware testing.

Therefore, this thesis aims towards analyzing and confirming the accuracy of the hardware implementation of a Real-time QRS complex detector and Heart Beat classifier using an algorithm based on the modified Pan Tompkins algorithm, which sets a threshold for detecting the peak locations and then classifies them as normal or ventricular. The algorithm, which is a single-lead, first derivative based heart-beat detector and classifier, has been coded in MATLAB. Then using MATLAB's HDL Coder and System Generator applications, it was converted to VHDL. VHDL is the hardware descriptive language that can communicate with our FPGA board in Xilinx ISE 14.7. All analysis and conclusions have been verified using the SPARTAN-6 FPGA board specifications.

TABLE OF CONTENTS

ACKNOWLEDGMENTS ..... v

ABSTRACT ..... vi

LIST OF FIGURES ..... x

LIST OF TABLES ..... xi

LIST OF ABBREVIATIONS ..... xii

CHAPTER 1. INTRODUCTION ..... 1

1.1 ANALYZING THE ELECTROCARDIOGRAPH ..... 2

1.2 MOTIVATION ..... 5

1.3 CONTRIBUTION AND THESIS ORGANIZATION ..... 6

CHAPTER 2. BEAT DETECTION AND CLASSIFICATION ALGORITHM ..... 8

2.1 DATA CLEANING AND DENOISING OF THE SIGNAL ..... 10

2.2 BEAT DETECTION LOGIC ..... 14

2.2.1 Determining the threshold for the ECG signal ..... 16

2.2.2 Reducing the False positive and False negative beat detections ..... 18

2.3 CLASSIFYING THE DETECTED BEATS ..... 19

CHAPTER 3. FLOATING POINT TO FIXED POINT CONVERSION IN MATLAB ..... 21

3.1 ERROR IN FLOATING POINT TO FIXED POINT CONVERSION ..... 22

3.2 FIXED-POINT DESIGNER ..... 23

CHAPTER 4. HARDWARE IMPLEMENTATION ..... 25

4.1 HDL CODER AND SYSTEM GENERATOR ..... 26

4.2 STEP-WISE DESIGN APPROACH AND IMPLEMENTATION ..... 27

4.2.1 Data Cleaning Stage- Butterworth Filtering ..... 28

4.2.2 Low-Pass Filtering ..... 29

4.2.3 High-Pass Filtering ..... 29

4.2.4 Squaring, Smoothening and Padding zeroes to the ECG signal ..... 30

4.3 MEMORY AND SLIDING WINDOW BLOCK ..... 31
4.3.1 Memory Allocation Stage ..... 32
4.3.2 Sliding Window Block ..... 33

CHAPTER 5. SIMULATION RESULTS AND COMPARISON ..... 35
5.1 RTL SCHEMATIC OF OUR ECG BEAT DETECTING ALGORITHM ..... 35
5.2 ANALYZING RESULTS TO CHECK ACCURACY RATES ..... 37
5.2.1 Analysis of Record 100 of MIT database using our Beat Detection Algorithm ..... 38
5.2.2 Analysis of Record 105 of MIT Arrhythmia Database ..... 40
5.2.3 Analysis of Record 119 from MIT-BIH Arrhythmia Database ..... 42
5.3 DEVICE RESOURCE UTILIZATION ..... 46
5.3.1 Synthesis Report ..... 47
5.3.2 Timing Constraints Report ..... 48
5.3.3 Power Report ..... 49

CHAPTER 6. CONCLUSION AND SUGGESTED FURTHER IMPROVEMENT ..... 50
6.1 FUTURE WORK AND ADVANCEMENT ..... 51

APPENDIX ..... 52

REFERENCES ..... 66

BIOGRAPHICAL SKETCH ..... 69

CURRICULUM VITAE

# LIST OF FIGURES


1.1 A Typical two-cycle ECG tracing ..... 3

2.1 Generalized Backbone of a QRS Analysis Algorithm ..... 9

2.2 (a) Unfiltered ECG signal with noisy peaks in Record 105 of MIT arrhythmia database. (b) Baseline wander noise in the same record. (c) Ventricular Arrhythmic beats which look similar to noisy peaks ..... 11

2.3 (a) Original signal of Record 100 from MIT-BIH Database (b) The signal cleaned after passing through a bi-directional Butterworth filter. ..... 12

2.4 QRS complexes from the frontal leads, in Lead II for R-peak detection ..... 14

2.5 Lead II electrode placement for ECG analysis ..... 15

2.6 Ventricular beat detected in the record 105 using our algorithm ..... 19

3.1 Floating to Fixed point conversion with error ..... 23

4.1 Block system calling the Fixed-point generated MATLAB code in Simulink ..... 26

4.2 Detailed Hardware System for the beat-detection Model ..... 27

4.3 Original ECG input above and below the signal is cleaned by the Butterworth Filter ..... 28

4.4 Low-Pass filter output of our ECG signal ..... 29

4.5 High-Pass filter results on our processed signal ..... 30

4.6 Results of our squared and smoothened ECG signal ..... 31

4.7 Memory block for our ECG System designed in Xilinx ..... 32

4.8 Block diagram with input and output pins of the Sliding window block ..... 33

5.1 The complete Beat Detection System RTL Schematic ..... 35

5.2 Beat-detection Block of our algorithm with the I/O pins ..... 36

5.3 Design summary after implementation and synthesis of our design ..... 37

5.4 Normal beats of Record 100 detected in Green and Ventricular in Red ..... 38

5.5 Premature Ventricular beat detected in Record 100 of the MIT-database ..... 39

5.6 Beats classified for the 30-minute signal of Record 105 of MIT database ..... 40

5.7 Baseline-wander noise cleaned, and peaks detected by our algorithm ..... 41

5.8 Noise in the ECG frequency range, leads to error in results for Record 105 ..... 42

5.9 Normal and Ventricular beat locations for Record 119 ..... 43

5.10 Ventricular beat pattern clearly visible in Record 119 ..... 44

# LIST OF TABLES

5.1 Comparing our accuracy rates with MIT database results ..... 45
5.2 Design Summary of our Beat-Detection Algorithm ..... 47
5.3 Timing Details and Constraints for our Design ..... 48
5.4 Power Supply and Consumption Summary of our Design ..... 49
A.1 Elaborate Resource Utilization of the SPARTAN-6 FPGA ..... 52

# LIST OF ABBREVIATIONS

AVC: Atrial Premature Contraction

DWT: Discrete Wavelet Transform

ECG: Electrocardiogram

FFT: Fast Fourier Transform

FIR: Finite Impulse Response

FN: False Negative

FP: False Positive

FPGA: Field Programmable Gate Array

HDL: Hardware Descriptive Language

I/O: Input/ Output

IOBs: Input Output Buffers

ISE: Integrated Synthesis Environment

MIT-BIH: Massachusetts Institute of Technology-Beth Israel Hospital

MUXCY: Carry Multiplexer

PCM: Phase Change memory

PVC: Premature Ventricular Complex

VHDL: VHSIC Hardware Description Language

# CHAPTER 1

# INTRODUCTION

In today's digitalized fast paced world, keeping track of one's health is a challenge in itself. Hence, when manual entries of cardiac activity got digitalized to automated monitoring in the late 20$^{th}$ century, heart-rate monitors began to be used widely by performers of various types of physical exercise [1,2]. Slowly it got commercialized into wearable devices that monitor and record a person's fitness activities on a regular basis. Now considering this as a foundation, there has been advancement towards bringing the Electrocardiography (ECG) technology out of the hospitals and emergency care centers in the form of non-invasive wearable ECG sensors, that can give real time reports of any abnormality detected in the heart [3]. If these reports are accurate then, can save numerous people from sudden cardiac arrests, arrhythmias or heart failures. Another point that is to be noted is that as individuals wear the ECG sensors on a continual basis while performing daily activities, there is a high probability that external noise might corrupt the signals obtained from these sensors, resulting in the beat detection process to be error prone and difficult to analyze. External noise here can be defined as, human-made unwanted sound due to movements during an individual's routine activity, wireless signal transmission interferences, muscle movements, baseline wander, etc., resulting in reduction of the quality of the vital ECG information. Hence the denoising and data-cleaning parts of the beat detecting algorithm must be fool-proof, taking into consideration all worst-case scenarios and ensuring that there are minimal false detections. Also, to ensure real-time monitoring, the algorithm should be capable of adapting to varying heart rates exhibited by various cardiac arrhythmias when present. The algorithm should also be capable of distinguishing an arrhythmic heart beat from noise, as arrhythmia being an abnormal sequence of

heart beats can be mistaken as noise [3,4,5,6]. In this research, the algorithm is already coded and checked for its correctness in software using either MATLAB and LabVIEW [3,4], after which the code was converted into VHDL for testing the algorithm on hardware. This is because software processes ideally run on a virtual environment, which is often easy to deal with. Hence to ensure real-time effective analysis, one of the most reliable hardware interfaces to ensure that every part of the algorithm works the same way as in software, is the Xilinx ISE [7]. In the present study, the specifications of the very powerful FPGA- Spartan6 were used to confirm the accuracy and implementation of the algorithm on hardware. The algorithm was tested on data from the MIT-BIH Arrhythmia database.

### 1.1 Analyzing the Electrocardiograph

Electrocardiograph measures the electrical activity of the heart. To make sure that the heart is working fine it is necessary to understand the various abnormalities which differ from a normal heart beat [2]. It is known that the electrical conduction through the heart follows a set pathway under normal conditions. The Electrocardiogram signal depicted below in Fig. 1.1 shows the pattern that a normally functioning heart should have a particular pattern. Even slight disturbance in the electric conduction will alter the pathway of depolarization and change the time of the electrical events. Such alteration in the ECG signal might mean that the heart is suffering from a major problem. According to previous research every interval and duration of the signal must be in a specific range to be considered as a normal heart beat [8,9], which plays an important role when writing algorithms for accurate beat detection and classification. Hence, to ensure that the

threshold values analyzed by a beat detection algorithm are precise, the following major calculations regarding the intervals and durations should be strictly taken into consideration:

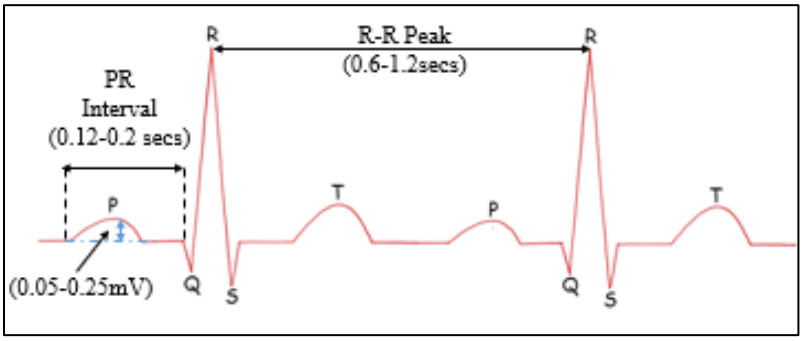


Fig. 1.1: A normal two-cycle ECG tracing

- **PR Interval:** The PR interval is measured from the beginning of the P wave to the beginning of the QRS complex. The electrical vector spreading from the right atrium to the left atrium gives the P wave during atrial depolarization. This interval corresponds to the atrioventricular (AV) node delay. This delay is responsible for activating the AV node to control the heart rate. This interval provides initial symptoms of certain cardiac arrhythmias. The normal range of this interval is supposed to be between 0.12 to 0.20 seconds.

- **QRS Duration-** The QRS duration gives the information of time taken for Ventricular Depolarization. As the ventricles possess more muscles mass than the atria, the electric signal has more conduction velocity which results in high spiked R peaks. These R peaks are usually the most common criteria for detecting a heartbeat in beat detecting algorithms

as it has a very steep peak and can be distinguished easily from noise. The normal length of the QRS complex should be maintained between 0.06 to 0.10 seconds (60 to 100ms).

- **ST Segment –** While QRS complex denotes the start of ventricular depolarization, the ST segment which follows the QRS complex is the time at which both ventricles are completely depolarized. This segment is important in the diagnosis of ventricular ischemia or hypoxia because under those conditions, the ST segment can become either depressed or elevated. Total time of ventricular depolarization is up to 0.43 seconds.

- **QT Interval-** The QT Interval reflects the total duration of depolarization and repolarization which means it denotes the time the heart takes to contract and then refill with blood before beginning the next contraction. This interval should not be more than 440ms which means 0.44 seconds.

- **T wave –** The ventricular repolarization which in general terms is stated as ventricular recovery is represented by the T- wave. The shape of the T-wave is sharply or bluntly rounded with amplitude of less than 5mm. The duration of the T wave is usually supposed to be between 0.10- 0.25 seconds. Abnormalities of the T-wave can be associated with life threatening diseases like hyperkalemia and hence perfect denoising of the ECG signal becomes very important when it comes to dealing with peaks with such low amplitudes so that they do not get missed during diagnosing an ECG through the wearable ECG sensors.

- **R-R Interval-** It is the time between QRS complexes. This interval plays an important role in determining instantaneous heart rate in modern wearable ECG sensors. The R-R interval

should be in the range of 0.6 to 1 second. This is referred to the fact that a heart rate is considered normal if it is in the range of 60 to 100 beats per minute.

The R-R interval checks for the ventricular rhythm while the P-P interval checks for the atrial rhythm. Ensuring that these intervals remain the same throughout the ECG is important, else unrhythmic beats gives early symptoms of heart problems [10].

## 1.2 Motivation

Analyzing the ECG signals at hospitals and health care centers have been a usual necessity when a person falls sick or feels pain in the left chest area. But the human body like any other machine or engine needs regular maintenance and to prevent fatal conditions like sudden cardiac attacks [11] from arising it is important, that medical check-ups are done at regular intervals. Today, we have a wide range of wearable fitness trackers, using different algorithms and some different technologies altogether. ECG sensors have got fitted to devices as small as finger rings. Watches and wristbands monitor heart rates without using the 12-Lead system use the Photoplethysmogram technology [12] which illuminates the skin and measures the change in light absorption. Now the main need and challenge are that algorithms should be capable of removing unwanted noise altogether very accurately ensuring that beat detection and classification are precise, as it's the matter of life or death for an individual. Also, when it comes to hardware implementation, the algorithm should not be using too many resources.

Beat detection and classification algorithms today are based on various theories, which was started as an idea using an algorithm based on first and second derivatives originally developed by Balda et.al [13] which was then constructively modified by Pan and Tompkins [4,5] who designed an

algorithm based on digital filtering and Fourier Transform which had an accuracy of 99.14% when tested on the MIT-BIH database. But this particular algorithm used up more than 72% of the hardware resources [14] which leads to excessive power dissipation [15] along with memory usage, is definitely not a good choice. Next, an algorithm was proposed by Li et al. [16] based on the concept of Wavelet Transform, which was a possible improvement as wavelet can be localized both in time and frequency whereas Fourier transform gives values only in the time domain. That means, more data for analysis, hence more accuracy. As the popularity of Machine Learning increased, Armato et al. in the research work [17] proposed a QRS features extraction using unsupervised learning of neural networks. But again, implementing machine learning algorithms on hardware calls for high end parallel computing which leads to model-training performed on graphic cards (rather than only on the CPU) [18]. More effective advancements would include using well designed filters with the wavelet transform concept, which would lead to more effective hardware implementation, which is the basic concept of our research.

### 1.3 Contribution and Thesis Organization

We propose the SPARTAN-6 FPGA hardware implementation of an algorithm which is based on detecting the threshold during the de-noising of pre-processing of the incoming ECG signal [3]. This threshold determination from the de-noised ECG signal helps in reducing the number of comparisons of a fiducial mark for being a QRS complex or not. Also, it saves memory space as the algorithm does not require a secondary threshold detection as in [4,5]. The MATLAB code for the algorithm is converted to VHDL using System Generator and HDL Coder, which is a MATLAB application.

We also propose to use an additional bi-directional Butterworth filter for the denoising and data cleaning stage to prevent cases like baseline wander and to effectively differentiate noise from ventricular arrhythmic beats.

Finally, we provide the simulation and hardware resource usage results after implementing the VHDL code on the SPARTAN-6 using Xilinx ISE. We also compare our accuracy with the previously established algorithms and power consumption characteristics to analyze the performance of our algorithm.

# CHAPTER 2

# BEAT DETECTION AND CLASSIFICATION ALGORITHM

The approaches for beat detection and classification of ECG signals require an expanded knowledge of signal processing, physics, biology and complex algebra. By linking all these fields together, mixing and matching various theories from them, over the past 40 years QRS detection algorithms have varied from genetic algorithms, digital filtering [4,5,6], wavelet transform [6,16] to heuristic methods of supervised or unsupervised neural networks [17,18] and non-linear transformation [19].

Even though the logic behind every algorithm varies and generates different levels of accuracy, the basic framework of the beat detection process has almost remained the same. The entire process can be simplified and broken down into two broad parts which then have sub-parts. The advantage here in having such stages is that each stage has a specific purpose and it does not matter how that purpose is solved, but what matters is at the end of that specific stage, the outcome should match the already established results, that are universally accepted, both biologically and scientifically [20].

Here we have relied on data from the MIT-BIH arrhythmia database which has ECG signal recordings of 10 seconds to 1-hour long. It also contains data related to all kinds of arrhythmias and provides the specification of each record, e.g. if Record-100 is chosen from the MIT database as an input to the algorithm, then the result that is generated as an output of the beat detection algorithm should match with the already existing results that the MIT database states about that particular record. If the database states that this particular record has a total of 2273 beats, then the

result of the algorithm should match that beat number. If the output of the algorithm is more or less than the already stated value, the difference will become the error and determine the accuracy of the algorithm.

The following block diagram shows the basic outline around which algorithms for QRS analysis are usually based upon. The two main stages that every algorithm must have is the pre-processing stage and the decision-making stage [14]. As our algorithm also classifies the beat a normal and ventricular, we also have a stage after peak detection as the beat classifier.

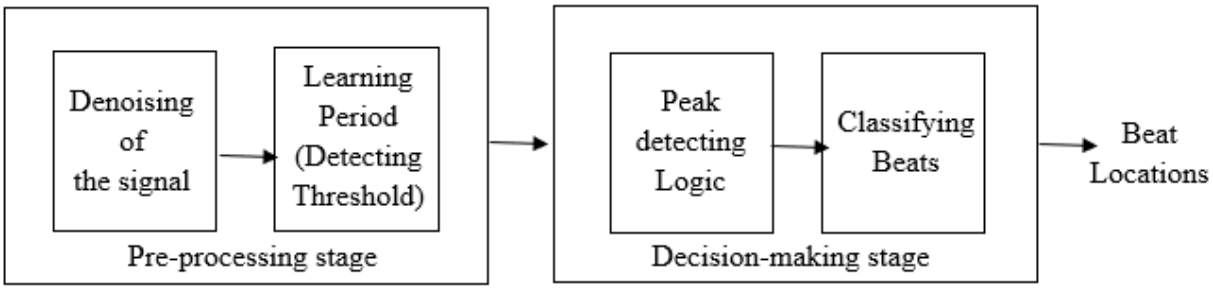


Fig. 2.1: Generalized backbone of a QRS analysis algorithm

Most of the algorithms have different strategies while generating the logic for the pre-processing stage. An incoming ECG signal is prone to noise and other vulnerabilities, hence every beat detecting algorithm is written with taking worst scenarios in to consideration. This is where the concepts of Digital filtering, Wavelet Transform and machine learning play their role and contribute towards improving the accuracy of the entire system. We in this section will be elaborating on the algorithm that we designed and highlight the differences our algorithm has from the other popular algorithms for ECG beat detection. We have divided our algorithm in to 4 parts, namely- Data Cleaning, Pre-Processing, Peak-detection and Beat Classification.

In our algorithm, pre-processing is considered the main function which then calls the remaining three functions of data-cleaning, peak-detection and ventricular-beat detection.

## 2.1 Data Cleaning and De-noising of the Signal

As it was pointed out earlier unwanted noise attributed to artifacts due to continuous unrhythmic movement, improper placement of the device or interferences can hamper the output result even if the logic of the peak- detecting algorithm is correct. Hence it is necessary that the technique used for data-cleaning algorithm is robust enough to differentiate a heart-beat from noise. That said, most of the arrhythmias have extreme unrhythmic beat patterns which might be confused with noise also. This is when the actual challenge comes. Below in Fig. 2.2, we have provided the comparison between a noisy ECG signal and a few Arrhythmias, so as to show how close these cases actually are and how such cases can determine the accuracy and robustness of an algorithm.

The foundation of any algorithm is its logic, and if the following worst-case scenarios are analyzed from Fig. 2.2 (a), (b) and (c), unless an ECG signal is appropriately filtered it becomes next to impossible to give correct results. All the three signals are part of the un-cleaned MIT database record 105 and can be labelled as irregular ECG signals but narrowing the classification in to noise and arrhythmic signals is what the actual challenge is. We cannot take the risk to labelling a noisy peak just because it has a high amplitude as a heart-beat nor can we let a ventricular arrhythmic beat be unnoticed considering it as noise. For example, in Fig. 2.2 (b) is a case of Baseline wander [20] which can be categorized as noise due to respiration, motion of the individual or even the instrument. If we ignore all the peaks during this low frequency activity of the ECG signal and label them as noise, the entire algorithm will flop. That's the reason why, while

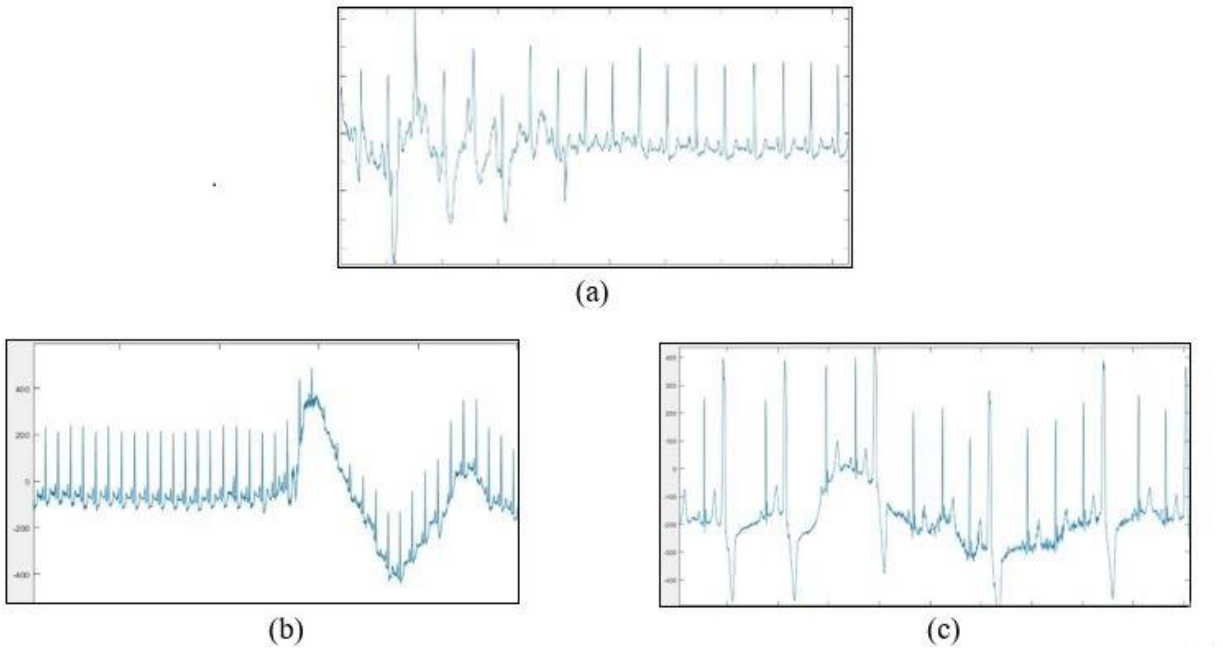


Fig. 2.2: (a) Unfiltered ECG signal with noisy peaks in Record 105 of MIT arrhythmia database. (b) Baseline wander noise in the same record. (c)Ventricular Arrhythmic beats which look similar to noisy peaks.

planning the de-noising algorithm, it is necessary to have sound knowledge of signal processing as that helps to understand which filters would best suit the processing of such noisy ECG signals.

For our algorithm for the initial stage of data cleaning we used the Bi-directional Butterworth filter [21] as it works like the wavelet transform concept [16]. Just as wavelet transform separates out frequency bands along time, the bi-directional Butterworth filter can be used to extracts out the ECG signal which usually ranges between 0.05 to 40 Hz [20] and filters out the remaining signals as noise. Also, Butterworth filters have better roll offs, which means that it provides better attenuation in the “reject band” [21]. Moreover, for hardware implementation, this bi-directional filter consumes lesser hardware resources than wavelet transform. So, all in all, choosing the Butterworth filter for our initial stage of data cleaning makes a good choice. Although

the signal now is not completely noise free, omitting out all the other signal frequencies apart from the actual range of the ECG signal, narrows down the chances of error. The difference of the Butterworth filtered signal from the initial input ECG signal can be seen as follows:

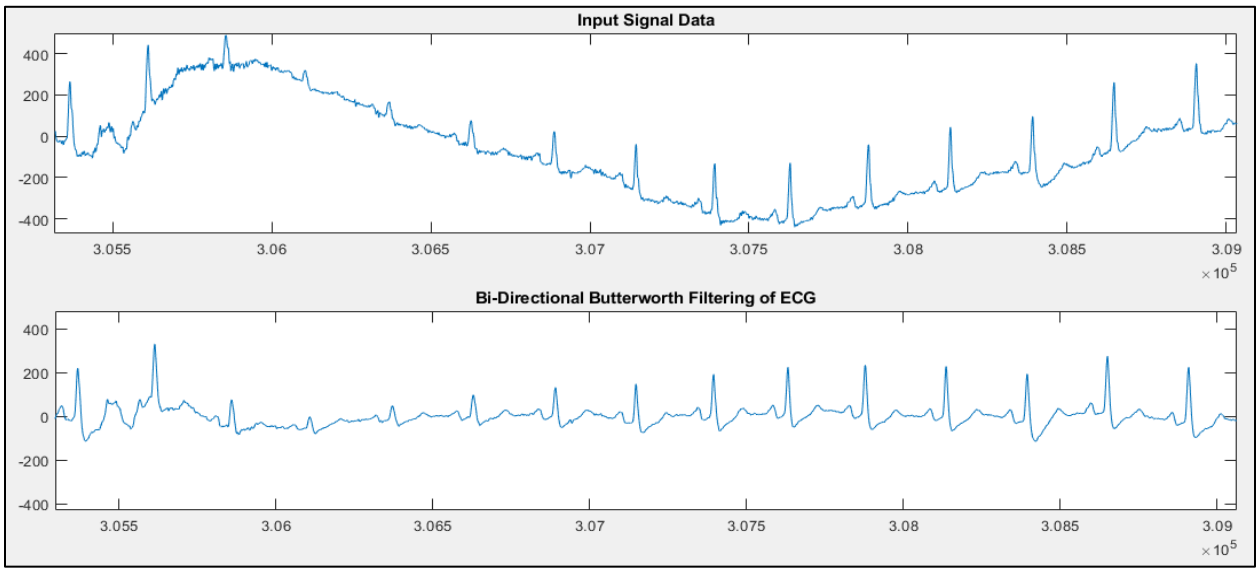


Fig. 2.3: Original signal of Record 100 from MIT-BIH Arrhythmia Database cleaned after passing through Bi-directional Butterworth filter

The Fig. 2.3 shows our results after passing the Record 100 data from the MIT arrhythmia database through the first stage of data smoothing and cleaning using the bi-directional Butterworth filter. This smoothened signal will then be passed through further filters to nullify the presence of any kind of noise which would ensure the accuracy of our algorithm.

To further enhance the preciseness, a low-pass filter to cut out any remaining high frequency noise is placed after the bi-directional filter. Additional prevention from low frequency noise requires a high pass filter which we used along with the low pass filter. Both the filters are of order 1 and are cutting off the frequencies above 30 Hz and 0.05 Hz respectively. Also, we

padded our signal with about 32 values of zeros just to increase the length of our signal and get more data, which in turn means more accurate results [3]. Finally, our de-noised signal was squared to convert if any negative values to positive and as we are considering just the R-R peaks here, squaring the peaks lesser than the amplitude of 1.0 will reduce their heights further and make the R-R peaks more prominent for beat detection.

If we actually have to make a logic, to confirm that denoising is done right, it can be inferred that if in a clear signal when the distance between the R-R peaks are less than 225 milliseconds [4,5] or the amplitude of the T or P wave is outstandingly high as discussed in the introduction, then those beats are surely arrhythmic beats and immediate action must be taken in such cases. Even though a ventricular beat having a high peak might be very close to an actual R-peak, it is usually a negative high peak, which should be easily distinguishable if noise is removed efficiently. Hence, accurate de-noising of the real-time acquired ECG signal can save an individual from suffering for any kind of severe cardiac arrests.

## 2.2 Beat Detection Logic

It has often been seen that an ECG signal might have inverted waves or intervals. This does not mean that there is some problem in the heart, but it's just that the lead might be negatively oriented. Hence while analyzing a particular feature of the heart's electrical activity, it is important that we understand the standard 12- Lead ECG design and which Lead to choose for reference while collecting results. Lead refers to an imaginary line between two ECG electrodes [22]. The 12-lead ECG provides the heart's electrical activity by recording information through 12 different perspectives on 12 separate graphs on ECG paper.

As mentioned earlier, our algorithm is single-lead first derivative based and as per previous work [10], Lead II is a good option to count and characterize the distinct R-R peaks. The Fig. 2.4 below shows the ECG output of the hearts' electrical activity in the frontal plane.

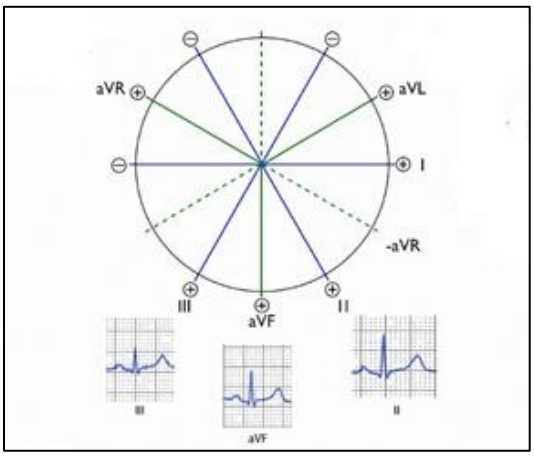


Fig. 2.4: QRS complexes from the frontal leads, which shows that Lead II is most suitable for R-peak detection

In Lead II, most of the heart's electrical current flows towards its positive axis and this lead can justify the pattern of an ECG in the best possible way [22]. Lead II requires electrodes to be placed on the Right arm, left leg and left arm. This placement also provides the bi-polarity required for ECG monitoring. Choosing the limbs as electrode locations is a safe option as there is no muscle there to generate action potentials that obscure the hearts activity [22]. The Fig. 2.5 shows that Lead II connects the left leg as positive to the right arm's negative, while the right leg is used a ground. Lead II gives a clear view of events moving up or down which means, the action and coordination between the atrium and the ventricle.

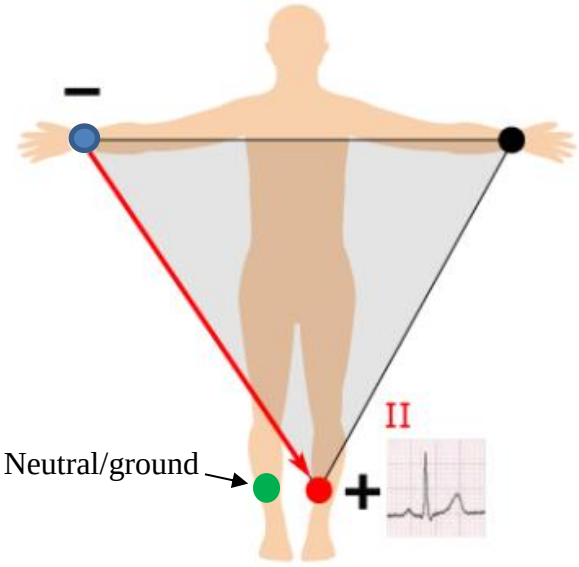


Fig. 2.5: Lead II electrode placement for ECG analysis

For real-time generation of input signals to our hardware, we would need to place the electrodes similar to the Fig. 2.5. The number of R peaks gives the total count of beats, whereas the distance between those peaks determines whether the beat is a normal or ventricular beat.

### 2.2.1 Determining the threshold for the ECG signal

For a peak in the ECG to be labelled an R-peak, it should abide by certain specifications so that the particular QRS complex which contains the R-peak can be considered as a beat. Thus, the beat-detection algorithm needs to begin with a learning period that studies the incoming ECG signal, then analyzes the highest and lowest peaks after which it generates a logic which is the basis on a threshold amplitude value based on which the algorithm is going to decide the further R peaks in the signal. As our QRS detector works real-time, having a good learning period becomes really important for accurately detecting the beats.

For our algorithm the input data is a 30 minutes long signal, which we are taking from the MIT-BIH Arrhythmia database [11]. We then divided this 30-minute-long input data into chunks which are 10 seconds long. This means that there are 180 chunks in total each chunk being of 10 seconds. Our algorithm has a sliding window which de-noises and reads 2 chunks at a time. During each window, our algorithm scans through all the possible high peak values and decides on an average value above which all peaks determined in the further chunks would be labelled as an R peak. Although the threshold value completely depends on the incoming signal, on an average high amplitude peak can be labelled as an R peak if its amplitude is more than 75% greater than the peaks on the either sides [25].

MATLAB has a built-in function called 'findpeaks()' which is used as a basis for the beat-detection logic. The function 'findpeaks(data)' [23] returns a vector with the local maxima (peaks) of the input signal. As explained above this function compares the two neighboring samples, and checks whether it is greater than them, if it is, then the function checks if the peak is almost equal or around the previous point which was labeled a peak and returns back the peak locations. Hence with regular comparison, the findpeaks() function helps in determining the R-peaks locations in our algorithm. This function is implemented on the clean signal and is called in the peak_detection() function. But the threshold determination takes place during the pre-processing stage itself. The peak_detection() is called by the pre-processing() function. The peak_detection() function calculates and sends back the threshold value back to the pre-processing() function and after the pre-processing of the signal is done in the first 20 seconds, this threshold value is stored in a variable, and then is used for future analysis of the R-peaks. The concept of R-peak threshold determination and adaption are taken from [3] and modified in MATLAB. Hence, this summarizes

the pre-processing() function- calling the de-noising function which has all the linear and non-linear filters to give out a clean signal and determining the threshold of the beats in the ECG signal for further peak detections.

The peaks detection() function has a sliding window of 10 seconds which detects the peaks based on the threshold value and then moves to the next chunk to determine the peaks. This loop is continued till all the peaks in the 180 chunks of the signal are labelled. But also, here as we are dealing with real-time biological signals where a very minute difference in the time-length of a peak or interval can mean, the electric signals are not being transmitted as they should, which might again mean, something serious. That is why any algorithm that is used for ECG beat detection must be adaptive in nature in order to follow the variations in the pattern of the waveform. After the peaks are decided, it is very necessary that before defining them as beats, the distance between the peak before and after it is measured. It is to be noted that the distance between the beats must not be less than 225 milliseconds and any peak detected within that 225-millisecond distance must be discarded as it is highly improbable to have two QRS complexes so close to each other [14]. Added to that if the distance between two peaks is too large, may be more than 1500 milliseconds which is 1.5 seconds, the algorithm should have the capability to understand that a beat is missing. Adaptive search-back logic has also been incorporated in our algorithm based on the algorithm elaborated in [3,14].

### 2.2.2 Reducing False Positives and False Negatives Beat Detections

A false positive beat detection would mean that a peak has been wrongly labelled as a QRS complex. The peak that was detected as an R-peak, might either be a noisy peak or some high

amplitude T-wave. A noisy peak detection can be avoided by proper de-noising of the signal, which we effectively do in our algorithm. Also, detecting a high amplitude T-wave as an R-peak is avoided in our algorithm by placing the restriction that the distance between any two consecutive peaks cannot be less than 225 milliseconds. Hence, even if the peak-detection() function determines an high amplitude peak wrongly, the back-up logic of the minimum distance between two peaks being 225 milliseconds will surely prevent false detections of the QRS complex. Also, a high amplitude T-wave might mean a severe case of arrhythmia [2], here our main focus is on beat detection and classification, so any detection of irregular peaks has been considered as noise.

Talking about false negatives, a false negative beat detection means that the QRS detecting algorithm missed detecting a R- peak or the QRS complex. Using the Adaptive Search-back logic, our algorithm makes sure that if a high threshold is acquired by the threshold adapting algorithm depending on the previous beats, then sudden appearance of a low peak beat should not be missed assuming it a noisy peak [1].

False positive and False negative beat detection can hamper the accuracy of a beat detecting algorithm to massive extents [9,19]. Hence, taking into consideration every possible worst-case scenario is very important. As in our QRS complex detecting algorithm, we made sure the conditions that lead up to such cases are removed altogether by narrowing down the specifications for heart-beat detection.

### 2.3 Classifying the Detected Beats

Our algorithm follows a logic [3] which classifies the detected beats as normal or pre-mature ventricular, as based on the information provided in the MIT-BIT Arrhythmia database [11]. As

per Mayo Clinic, the cases for Ventricular pre-mature (PVC) beats are very common, with almost 3 million US cases per year [10]. Such beats are termed as extra beats which are initiated by either of the lower ventricles and disrupt the normal electrical activity of the heart. At times, this might result in a skipped beat or feeling a flutter in the chest as shown in Fig. 2.6 [2].

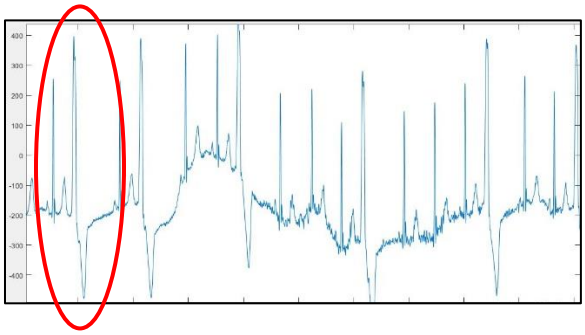

Fig. 2.6: Ventricular beat detected in the record 105 using our algorithm

The premature discharges might be caused by electrolyte imbalance, high blood pressure, lack of oxygen or a minor heart attack and immediately after this pre-mature contraction, the electric system of the heart almost resets but it still can be felt at times. The symptoms of PVCs are pounding, loss of breath or increased awareness of the heart-beat [2]. But quite often a person who might not be aware might leave it unnoticed. PVC due to lack of oxygen or blood pressure might not be dangerous initially and can be ignored, but for case like PVC after a heart attack with conditions like Bigeminy (Every other beat is a PVC) or trigeminy (Three consecutive beats are PVC) [3] proper diagnosis is essential. Such cases come under the definition of Ventricular Tachycardia, which if prolonged may lead to Ventricular Fibrillation, which is a fatal condition and is almost incurable [20]. Hence using our algorithm when implemented as a wearable device, a regular monitoring over the heart's health can help in an early detection of heart problems and

result in saving a person’s life. After all, having proper medication and taking needed precautions are much worthier than a painful death.

# CHAPTER 3

## FLOATING POINT TO FIXED POINT CONVERSION IN MATLAB

Our algorithm which is software-coded in MATLAB has functions based on floating point data types, which means such data types can take in numbers that have fractional parts too. But as we had to implement this algorithm on an FPGA and test it for hardware accuracy, we needed to convert the floating-point based functions to fixed point. This is because embedded hardware systems with the tight cost constraints and more focus on increasing throughput rates [25] prefer not having a floating-point hardware take in Embedded hardware system like the Field Programmable Gate Array.

Hence it became important to determine the fixed-point data type of each variable in the function, namely the word length, truncation mode and overflow mode [26]. For this the entire MATLAB code had to be scanned and changes had to be made so that the conversion of the code could be successfully done to VHDL using the HDL Coder [27] and Simulink.

As MATLAB does not need the declaration of the various variables used in the code, while going through each stage of the HDL Coder application of MATALB, which helps in automatically converting MATLAB code to VHDL, we faced multiple errors, where we had to go back to the locations where the variables were either not declared or the fixed-point conversion got missed. The basic functions for the Low-pass and High-pass filters were automatically converted by MATLAB to fixed point but the functions that we created using our own logic were to be converted from floating point data-type to fixed point manually. MATLAB in one of its documentations also informs us that it has a specific set of fixed-point run-time library functions from the Fixed-Point

Designer [24] and that we must have all our functions from the list that is already in that documentation. This restriction narrowed down our MATLAB to VHDL code conversion. We had to make sure that we maintained the logic but also at the same time abide the rules of the HDL coder. Analyzing that HDL Coder being a pretty recent application offered by MATLAB, might not be a good choice while converting the complex algorithm with multiple functions to VHDL.

Also, when it comes to manual fixed-point programming, we had to be very careful, making sure overflows were avoided and the integer words must be scaled such that there is almost null loss of precision [26]. Added to that determining the number of shifts is a difficult task and is time consuming.

### 3.1 Error in Floating point to Fixed point conversion

Errors in any form may lead to unwanted results. And if after positive software testing of the algorithm, we get errors, while preparing for the hardware testing where the error is not in the logic of the algorithm but in just a basic step of fixed point conversion, then the results will surely be disastrous. Fixed point conversion has always been quite prone to errors, as it can be seen from the Fig. 3.1 below, where, an IIR filter as a floating-point data type is converted to fixed point, and the error during the conversion is also shown. This is just for one filter, if every linear or non-linear filter that we use, gives such a high error rate, our algorithm would have failed right at the De-noising stage itself.

With a lot of research in this particular field of Floating point to fixed point conversion (FFC), today there are strong and stable algorithms which have improved the conversion accuracy.

As FFC is a very important feature of embedded system design, various software have been made available to reduce the manual workload and problems faced while implementing a fixed-point algorithms on hardware like FPGA.

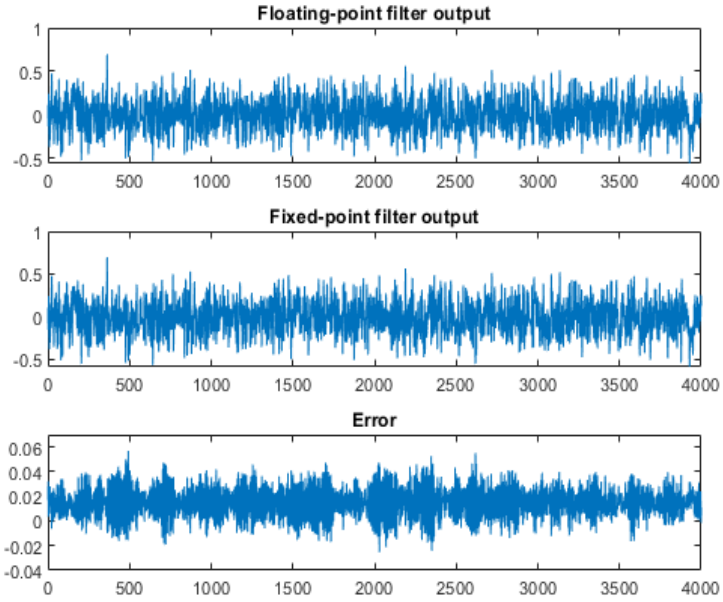


Fig. 3. 1: Floating to Fixed point conversion with error

### 4.2 Fixed-Point Designer

To make our floating to fixed point problem a bit easier, we relied on the Fixed-Point Designer which is again an application provided by MATLAB itself. We also separated our algorithm in parts and worked upon each function separately [24]. Fixed-Point Designer then helped us log the minimum and maximum values of all the named and intermediate variables. Based on these logged values the fixed-point code is generated where it takes into consideration the required overflow and scaling.

Then using Code Analyzer, a report is generated that identifies the calls to the functions and checks for the data types which got converted and which are not supported by Fixed-Point Designer. In our case it was just one function based which was the Wavelet Transform function, which we had initially used for the De-noising section. As this function did not support Fixed-point conversion using MATLAB, we planned to replace the Wavelet Transform function with the Butterworth bi-directional filter, which easily got converted to a Fixed-point code. Hence using automatic as well as manual methods we got our MATLAB code prepared for VHDL code conversion. Testing on the SPARTAN-6, which does not support floating-point hardware, might have taken us through the struggle of converting the floating-point data-types to fixed-point. But with improved technology and the system on chip getting smaller, it will be soon when FPGAs too get equipped with such peripherals.

Moving further in the following chapter we will be explaining, how we implemented this modified code in a Simulink block module so that Xilinx can interact with Simulink through System Generator to give the final outcome of the VHDL code for our beat detection algorithm whose functionality is the exact same as the MATLAB code.

# CHAPTER 4

# HARDWARE IMPLEMENTATION

For this research we implemented the modified version of the Pan-Tomkins beat-detection algorithm [3,4] that we have explained in the previous chapters, on a SPARTAN-6 FPGA board to ensure that the algorithm works accurately on hardware and also to analyze the performance of our algorithm in practical environment. As our beat-detection algorithm is mainly based on digital filtering where most of the filters have integer coefficients, the accuracy of our algorithm on hardware is assured to quite an extent already. Hence our focus is more on analyzing the various parameters on which efficiency of any hardware is determined upon like power consumption, heat dissipation, time constraints and memory usage. It is also very important to ensure that an algorithm is designed such that it does not use more than 50% of the hardware resources [13] as it can help in multi-tasking, adding new features to the algorithm in future and contribute in speeding up of the system. First, we will be discussing the software that we used to communicate with the hardware and then discuss the block diagram of our design.

## 4.1 Simulink and System Generator

The MATLAB 2017b [27] update has certain new updates and libraries which helps to generate portable, and synthesizable Verilog and VHDL code from MATALB functions. The HDL Coder and the latest version of System Generator helped us to customize our design block-wise with low-power consuming filter designs and storage units. As our algorithm is complicated with a top-module and more than 5 sub-modules, having an application like HDL coder to support and accelerate the MATLAB to VHDL conversion reduces the chances of missing out on important

parts of the logic and the block design using Xilinx System Generator makes sure that the circuitry is properly connected.

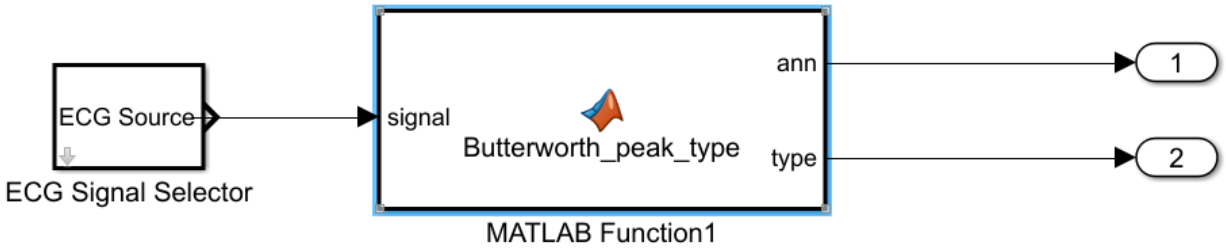


Fig. 4.1: Block system calling the Fixed-point generated MATLAB code in Simulink

Simulink is a platform for designing and simulating electronic systems which contain a main model with various sub-systems, then after testing the system in hardware environment, it automatically helps in generating VHDL code using the Xilinx block set via system generator. In the above Fig. 4.1, we created a Simulink environment in which we are giving input from a ECG signal selector [28] which is a set of pre-recorded and simulated ECG signals where all the signals are having a sampling rate of 360 Hz. This input is given to a function block which calls the fixed-point QRS detector code. Then System Generator assigns Xilinx Blocks to each of the filters which are then generated as sub-modules under the top-module of the design which gives out the vector location of the peaks and classified category of either ventricular or normal beats.

## 4.2 Step-wise Design Approach and Implementation

As our design is divided into two parts- the pre-processing and the decision-making modules (see Fig. 4.2), we started with modelling our Denoising module. The Bi-directional Butterworth Filter takes in the initial noisy ECG signal and gives the first round of cleaning of the data by separating

the generalized ECG bandwidth of 0.05 to 40 Hz. Next the signal is fed to a Low Pass and High Pass filter respectively to further narrow down the possibilities of noise effecting the signal. Then to increase the signal length, it is padded with a series of zeros, 32 in our case. The last stage of denoising is squaring and smoothing of the signal. This cleaned data signal is then given to the peak-detector and classifier algorithm that we discussed in the previous chapter. When it comes to hardware, storing the desired output in correct locations is very important. Hence, we reserved

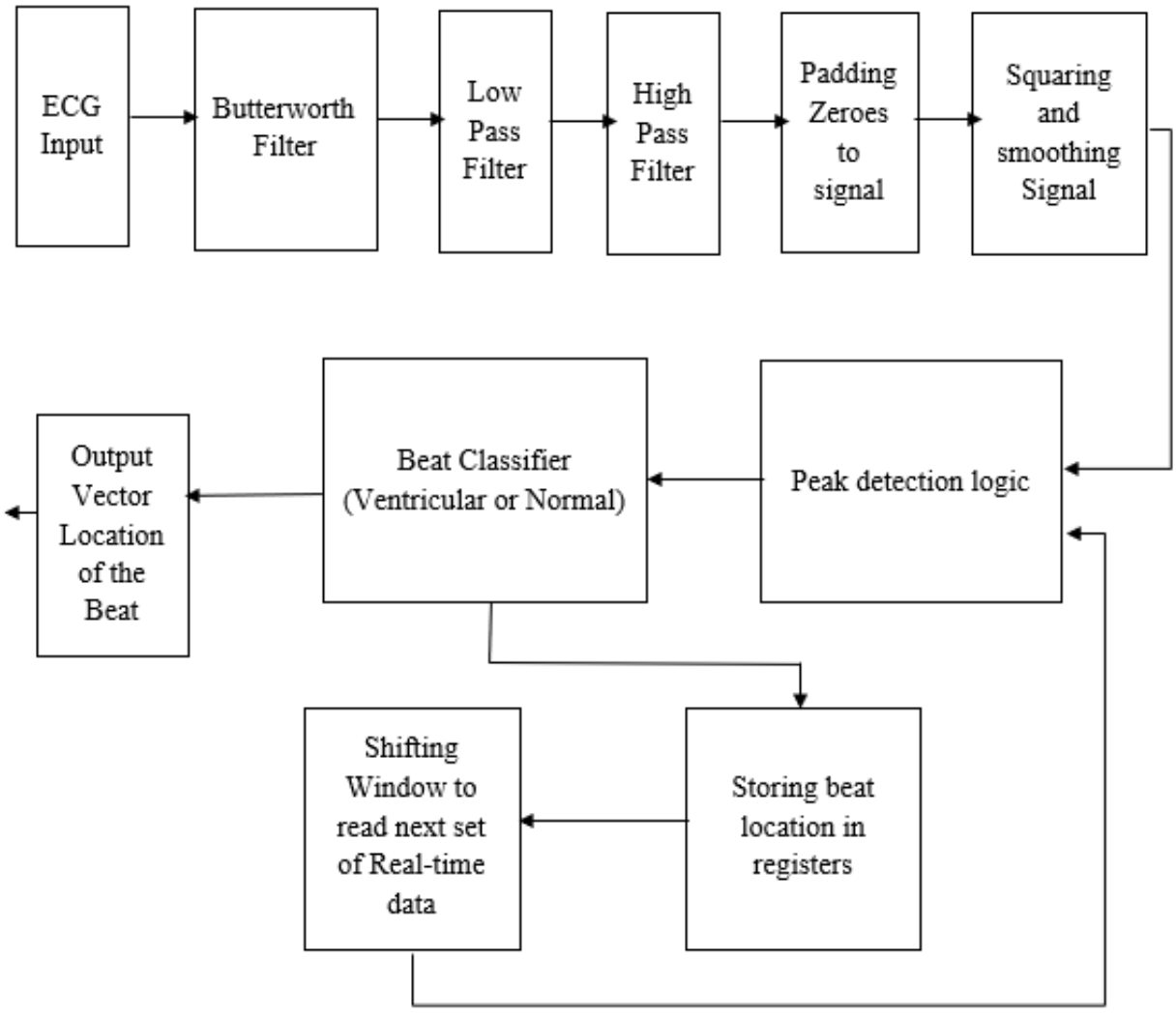


Fig. 4.2: Detailed Hardware System for the Beat Detection Model

enough registers, to make sure we do not run out of memory locations while storing the beat vector location values. Also, we have a separate module for our sliding window, which reads the first 20 seconds for threshold detection and there after reads 10 secs of real-time data at a time, and then slides to read the next set of the signal. We will be explaining each of the block diagrams in the further section.

### 4.2.1 Data Cleaning Stage- Butterworth Filtering

As discussed before, the incoming ECG signal is first cleaned by passing it through a Butterworth filter. It is the part of the pre-processing stage, which functions similar to the Wavelet Transform function but uses less hardware resources. As the Fig. 4.3 below shows that the smaller and irregular peaks have been removed by the Butterworth filter to reduce errors while peak-detection.

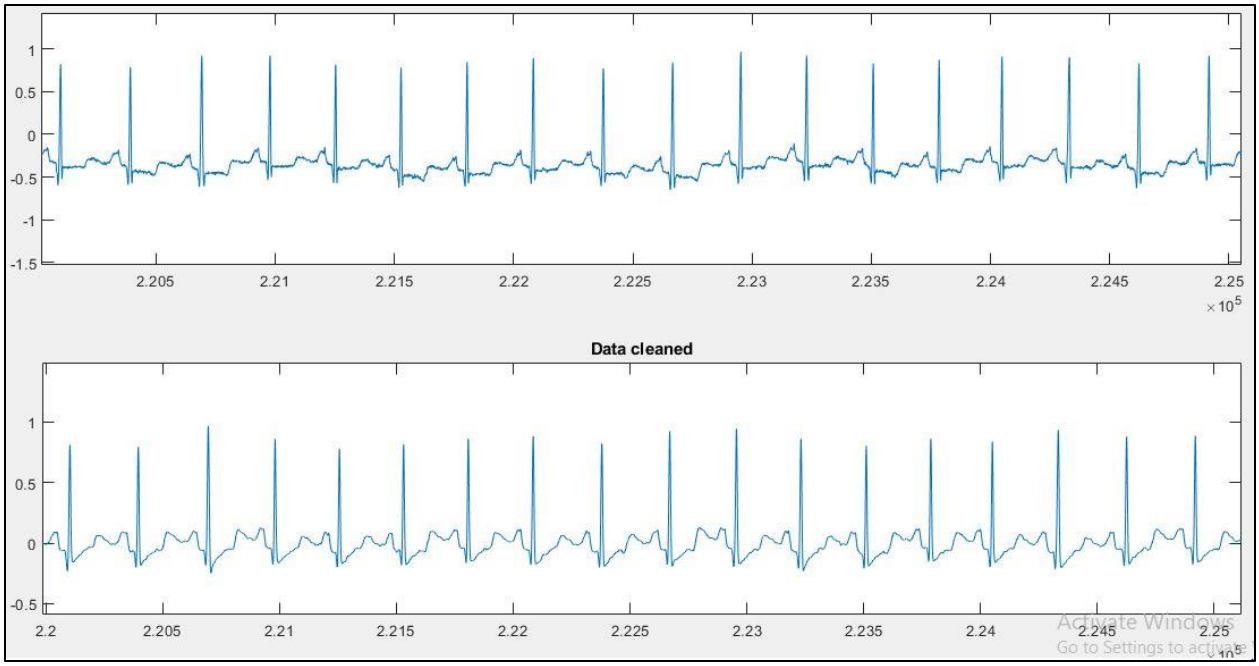


Fig. 4.3: Original ECG input (above) and the signal cleaned by the Butterworth Filter (below)

### 4.2.2 Low-Pass Filtering

We used a first order low-pass filter [14] in our design that had a transfer function of –

$$H(s) = \frac{\omega_c}{s + \omega_c} = \frac{1}{1 + s/\omega_c}$$

This simple design of our Low-pass filter helps in ensuring that the output cuts off all frequencies above 30 Hz (see Fig. 4.4), as the usual range of an ECG signal is from 0.05 Hz to 30 Hz. Hence the Low-pass filter removes all the high-frequency noise from the ECG signal.

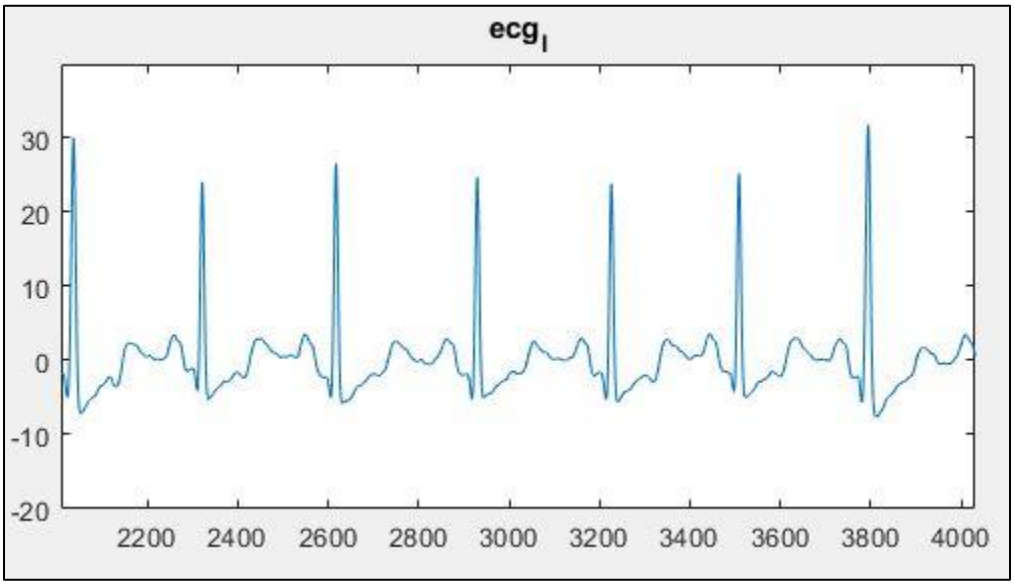


Fig. 4.4: Low-Pass filter output of our ECG signal

### 4.2.3 High Pass Filtering

Real-time ECG signal does not always remain at the same D.C. level, which is why conditions like Baseline wander happen. Hence placing the high-pass filter after the low-pass filter ensures the

removal of D.C. offset signals and lower frequencies and sets it to zero level as in Fig. 4.5 [20]. For our design we used a first order high-pass filter with a cut-off frequency of 0.05 Hz. This filtering magnifies the R-peaks and flattens down all other peaks, which rules out any possibilities of choosing wrong peak as beats. The blocks for the first-order Low-pass filter and high-pass filter have been taken from MATLAB's existing Simulink block functions and included in our ECG signal analysis system.

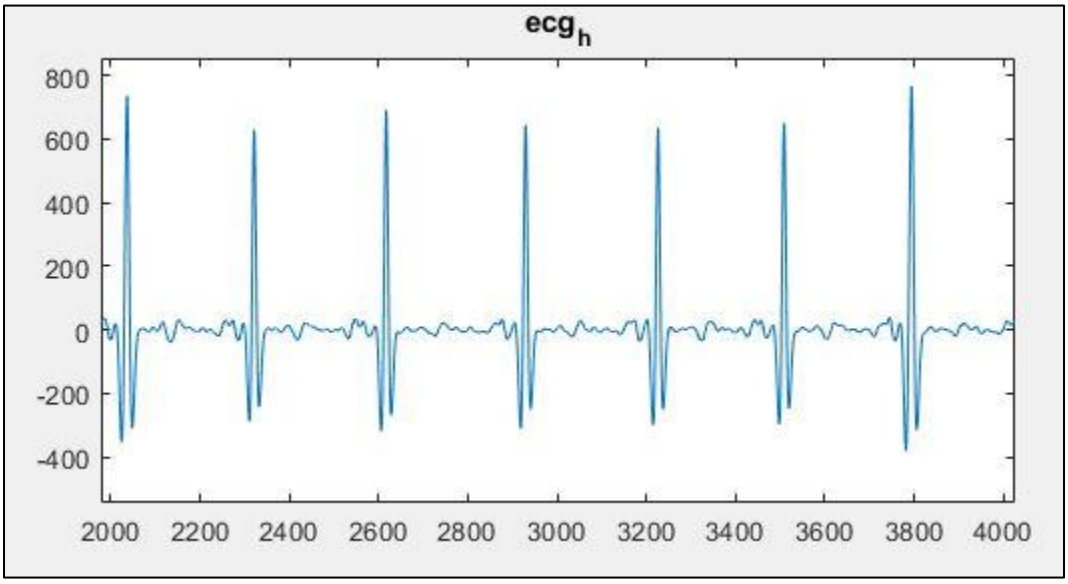


Fig. 4.5: High-pass filter results on our processed signal.

### 4.2.4 Squaring, Smoothing and Padding zeros to the ECG signal

The main purpose of padding zeroes to any signals is to produce longer result vectors, as a larger set of data contributes to lesser error possibilities. The squaring multiplier block, is used to produce a 40-bit output as a square-value of the input. Here in our design, the squaring logic has three main purposes. The first is to give the negative parts of the QRS complex positive values, which again

contributes to the assurance of noise-removal and gives a uniform pattern to the QRS complex that in turn reduces the possibilities of confusion in labeling the peaks. The second purpose of squaring is to further double and increase the amplitude of the already existing R-peaks to stand them out from the smaller peaks, ensuring proper beat detection. The third need for squaring was to further reduce the amplitude of the peak which are below 1mV, which will again ensure in giving better results. Also, as the Fig. 4.6 shows, smoothening down a signal gives a clearer analyzing criteria on determining the peak values, contributing to the accuracy of beat-detection. The output of this stage is the output of the pre-processing stage.

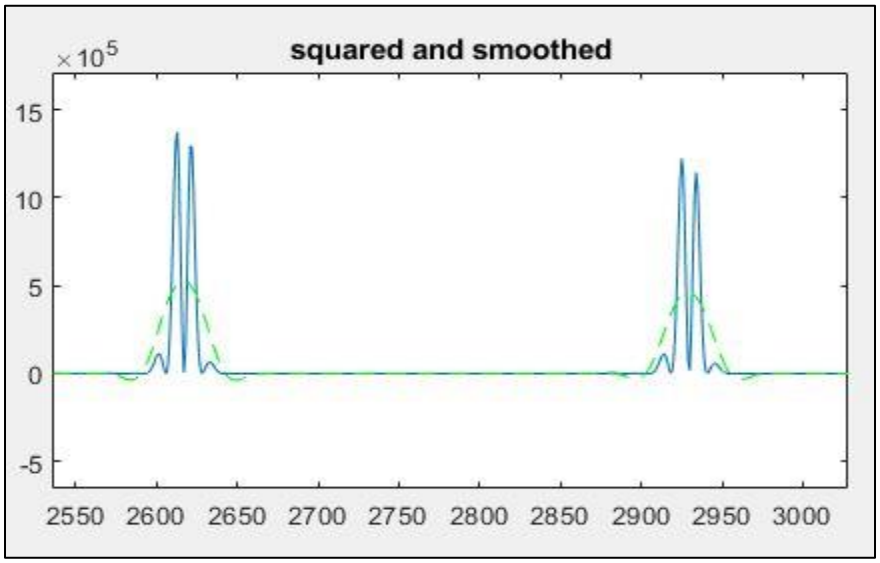


Fig. 4.6: Result of our Squared and Smoothened ECG signal

### 4.3 Xilinx SPARTAN-6 Memory and Shifting Window block implementation

Field Programmable Gate Arrays which were invented by Xilinx, are today a very powerful device which provide strong hardware support for complex circuit designs and also provide a possibility to update any system without the need to add or remove parts of the hardware. The FPGA board

on which we decided to test our algorithm is SPARTAN-6 which is built on the proven and improved 45 nm technology with industry-level leading connectivity features [26] and advanced I/O protocols.

### 4.3.1 Memory Allocation Stage

The Memory allocation module is based around the resources that the SPARTAN-6 board provided to us. From previous research [14], it can be understood that there can be a delay of around 60 samples between the signal at peak detection and the same signal at band-pass stage, hence taking in to consideration the worst cases, our memory module has a size of 60x40 bits which stores the oldest value in the $0^{th}$ position and the last value is stored in the $59^{th}$ location. Also, this memory module handles all our clock specification and the overall slack and delay calculation of our module.

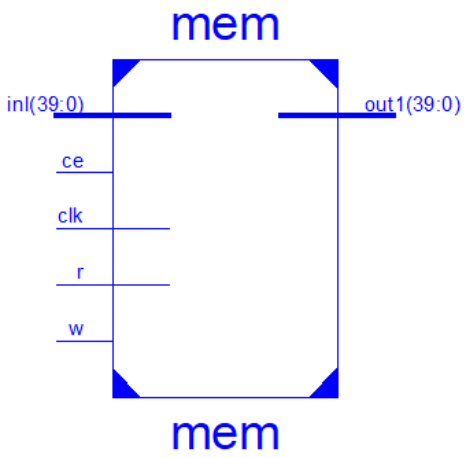


Fig. 4.7: Memory block for our ECG system designed in Xilinx

As Fig. 4.7 shows we are storing 40-bit 60 output peak location vectors at a time in the RAM. From the RAM the older values get stored safely for long term in storage register so that

newer values can be loaded to the RAM and the older vector peak locations detectors can be moved to the less used memory register locations. The ‘w’ is the write enable and ‘r’ is the read enable which provide permission to access the stored data accordingly.

### 4.3.2 Sliding Window Block

As discussed before the input data from the MIT arrhythmia database is usually 30-minute long. Hence for real-time analysis our algorithm is designed such that the first 20 seconds of input data is read and processed giving a certain threshold value, then a sliding window moves to the next 10 seconds chunk and compares the threshold value of the current chunk to that of the previous, to make sure beats are not skipped or to prevent sudden noise interference. ECG is biological signal, because of which measurements of such signals vary primarily due to environmental conditions like presence of glands and blood vessels, different tissue fat levels or electrode displacements, which is why keeping a check on the threshold value for peak detection of the ECG signal is necessary every 10 seconds of data (Fig. 4.8).

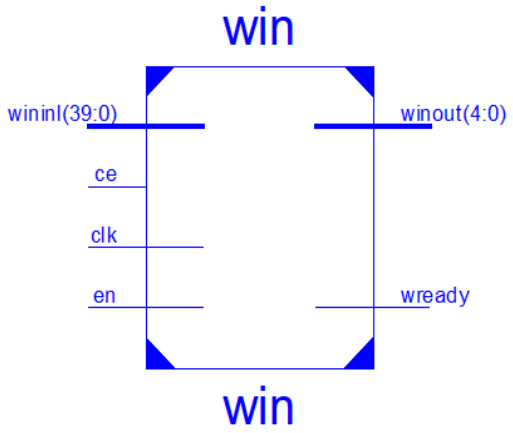


Fig. 4.8: Block diagram with input and output pins of the Sliding window block

The blocks described above have their own HDL code elaborated in several modules which were converted from MATLAB to VHDL using Xilinx system generator and HDL Coder. In the following chapter we will be discussing the performance of the system as a whole and compare its accuracy with the software implemented design. The hardware algorithm was simulated in Xilinx's own simulation environment and implemented in the SPARTAN 6 environment specifying on timing constraints, memory usage, speed and power consumption.

# CHAPTER 5

# SIMULATION RESULTS AND COMPARISONS

Once each module of our algorithm was converted to VHDL, our main aim was to test the system as a whole. Xilinx ISE 14.7 gave us positive simulation results in complete SPARTAN-6 environment with all the design utilities along with the RTL schematic, pre and post mapping, placement and routing of the design on the FPGA. The RTL schematic of the entire system consists of multiple small blocks combined together. The complex schematic with all input, intermediate and output blocks has been shown below in Fig.5.1.

## 5.1 RTL Schematic of our ECG beat Detecting algorithm

The Register-Transfer Level design abstraction gives the flow of the digital signal between the hardware registers and logical operations performed by the signals at each stage.

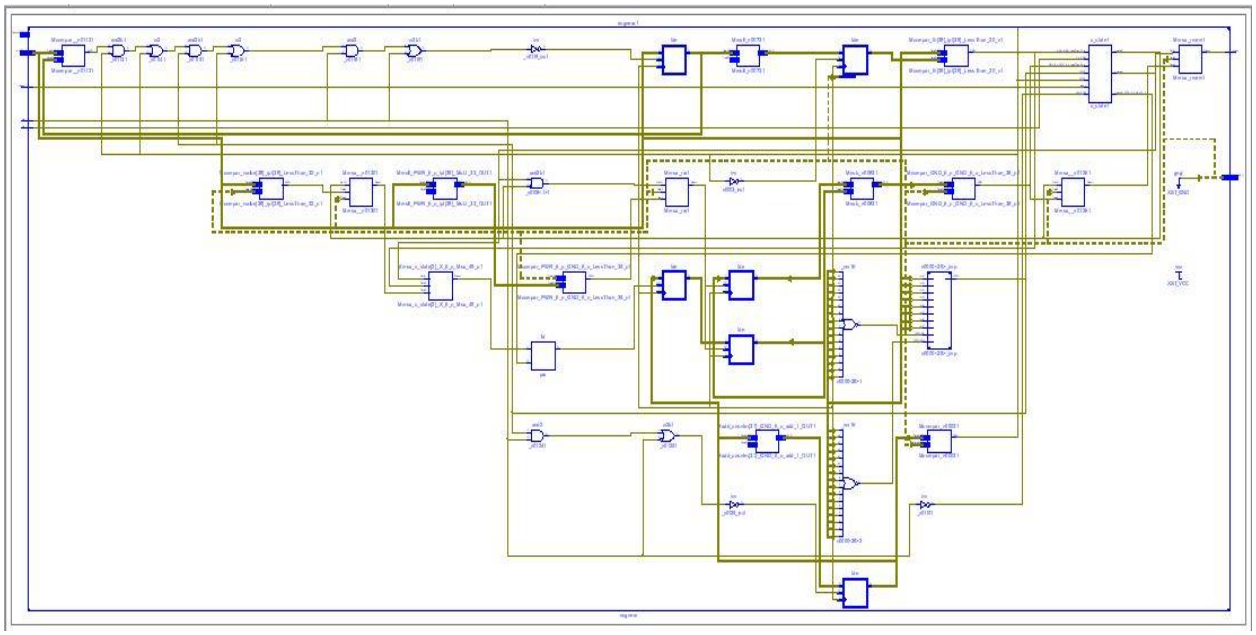

Fig. 5.1: The complete Beat Detection System RTL Schematic

The above Fig. 5.1 elaborates on the block connections of our design. It has all the pre-processing, peak-detection logic, memory allocation and sliding window modules embedded and connected in it. After Placement and routing of the signal, input and output pins are assigned according to the pin and port location of the SPARTAN-6 board. Fig. 5.2 gives the peak-detection block of our algorithm which we named ‘ecgnew’ as we kept modifying and improving our code.

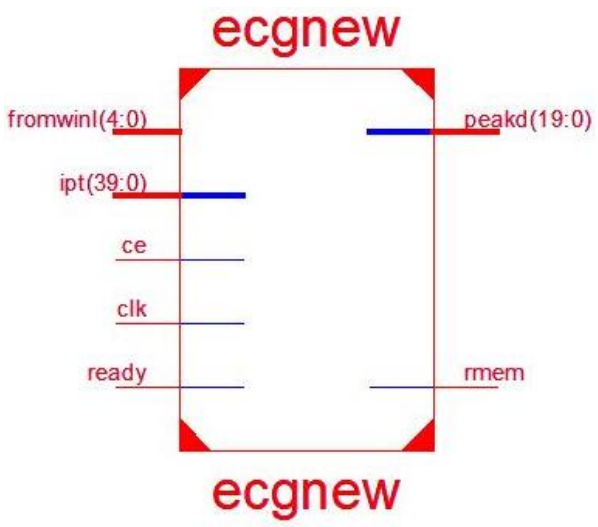


Fig. 5.2: Beat-detection Block of our algorithm with the I/O pins

The main two input pins of our beat detection block are the 40-bit input data and the sliding window that selects a specific chunk of 10 seconds of data, processes it at the positive edges of a synchronous clock, enabled with the ‘ce’ pin then gives the 20-bit long vector peak locations labelled as beats as output and ‘rmem’ gives the memory location registers in which the vector values are stored so that we can directly read from any particular memory location and match our results with the software implementation.

For the complete block diagram, we analyzed the Floor-Plan design of our algorithm. This is the stage of assigning pin and ensuring that no pin is over-lapped or remains undefined. We also generated Mapped, Translation and Power Reports which is shown in Fig. 5.3. The Mapped and

Translation Reports give the details of the percentage of Hardware resources used on the SPARTAN-6 FPGA. The Power report was generated by X-Power Analyzer which gives power consumption and heat dissipation results for our algorithm after being burnt on the FPGA. We will be discussing in detail and comparing the various reports after ensuring that our algorithm is accurate and is giving precise results.

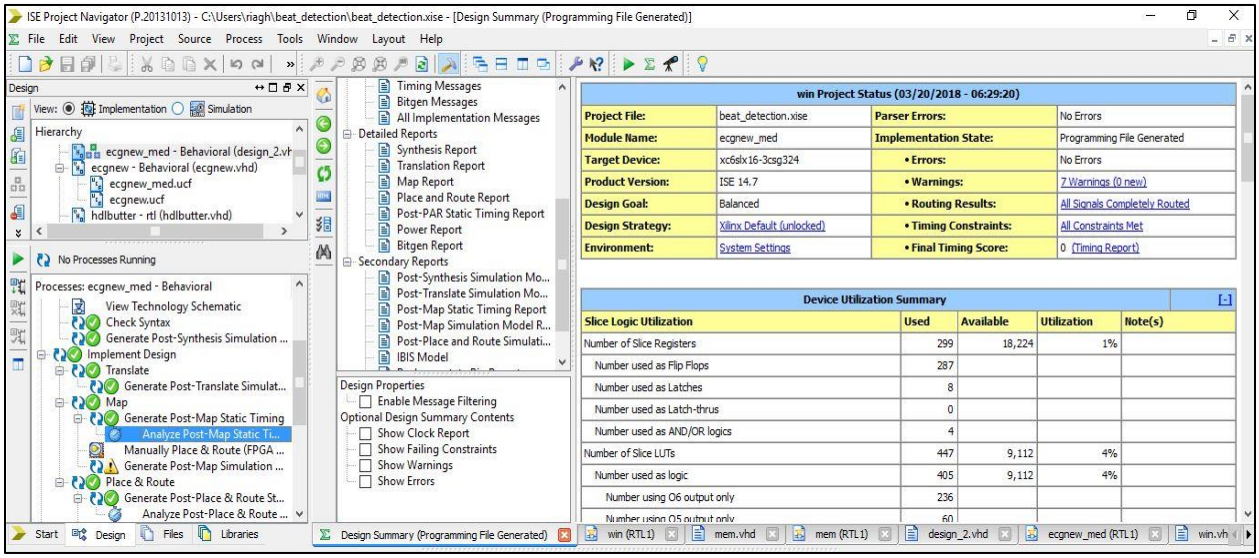


Fig. 5.3: Design summary after Implementation and synthesis of our Design

## 5.2 Analyzing results to check Accuracy Rate

To analyze and confirm the accuracy of our results, we created a test-bench with sample data from the following records of the MIT-BIH Arrhythmia database [11]- 100, 101, 105,117, 119 and 200. The MIT-Database along with input ECG signal data provides the characteristics of all its signals i.e. it also contains information regarding the total number of beats, the number of beats which have Pre-Ventricular Contraction and whether or not the signal is affected by noise. We programmed our test-bench to provide a .txt file of all the vector point locations that have been

determined as heart-beats by our hardware system and the result generated gave the same count and same vector values for the location as in our software implementation, which confirms that we were capable of maintaining the accuracy of our design in hardware too. Also, to visualize the classification oh the beats as normal and ventricular, we added in our test bench the logic to mark the normal beats in green and the ventricular beats in red. We also added the code to count the number of beats marked in green or red and both together. Then we compared our results with the MIT database to analyze our accuracy rate. We generated the beat classification waveform using Modelsim software [13] which is also a HDL simulation environment but is more flexible in generating varied signal outputs rather than Xilinx ISE. We will be discussing our Modelsim simulation results in the following section.

### 5.2.1 Analysis of Record 100 of MIT database using our beat-detection algorithm

In Record 100, the upper signal for ECG is generated through a modified limb lead II (MLII), which is obtained by placing the electrodes on the chest. The lower signal is V5 of a male aged 69 [11]. The following output was generated Modelsim for our algorithm.

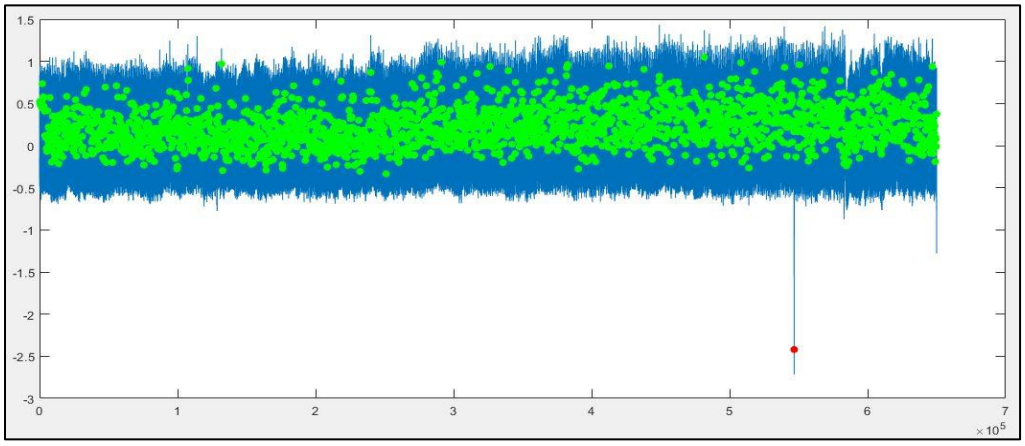


Fig. 5.4: Normal beats of Record 100 detected in Green and Ventricular in Red

Record 100 is a pretty clean signal with not much noise contamination. It is to be noted that we classify only normal and ventricular beats, any other arrhythmia type like Atrial Premature Complexes (APC) would get classified as normal beat, as we did not code for pattern detection of an APC in ECG. The MIT database classifies beats as normal, VPC and APC, where as we classify beats only as normal and ventricular.

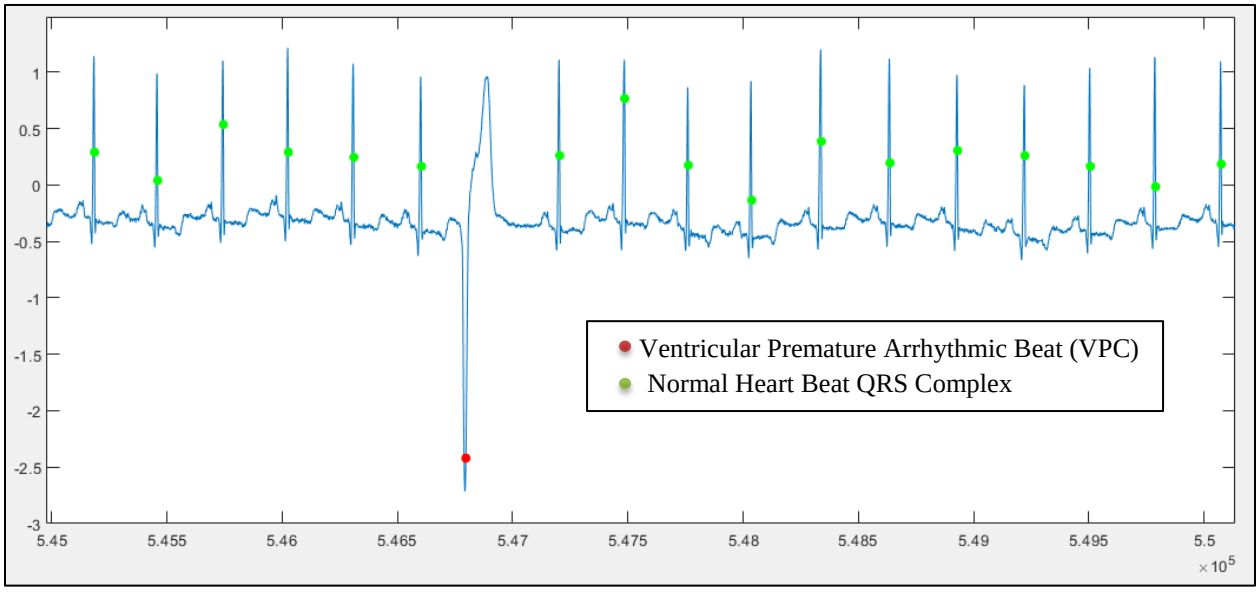


Fig. 5.5: Premature Ventricular beat detected in Record 100 of the MIT-database

Our algorithm detected the total heart beat count in Record 100 as 2273, which matches exactly with values given in the MIT-database website. It classifies 1 beat in the signal as ventricular and so as ours. The ventricular beat shown above in our result is the only beat that our algorithm also detected as a ventricular beat. The remaining beats have been detected as normal. If a person has such a situation having just 1 or two Premature Ventricular beats in their ECG, then the heart is still considered normal and this one VPC detection can be ignored.

Comparing our results with the values of this record in the MIT-database our algorithm can be tagged as accurate. But the main purpose of determining the accuracy of an algorithm is that how robust it can be and how much precise results are obtained in worst case scenarios. In our case worst case situations would be a signal which is highly noise contaminated or there are too many ventricular beats, because at certain points the algorithm might confuse normal beats with Ventricular beats.

### 5.2.2 Analysis of Record 105 of MIT Arrhythmia Database

Record 105 has quite a bit of noise contamination and that too, the noise frequency in this case lies quite a lot in the 10-20Hz range, which made it next to impossible for us to get rid of the noise in the range. There were about 40 segments of 10 second chunks which were reported to be noisy and difficult to remove.

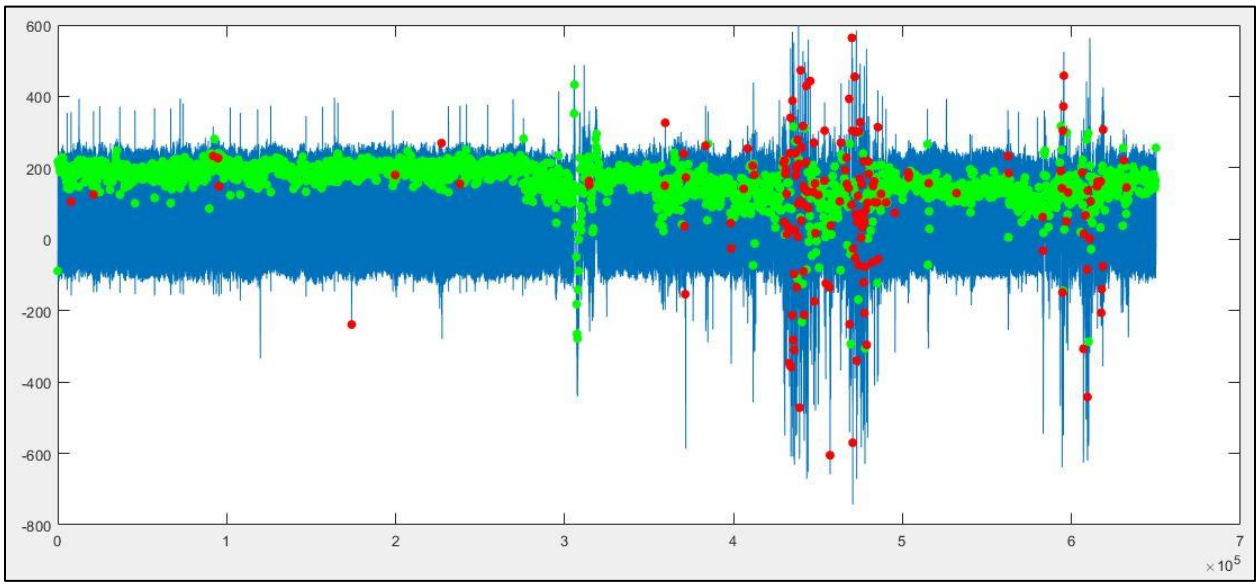


Fig. 5.6: Beats classified for the 30-minute signal of Record 105 of MIT database

Starting with Baseline-wander noise, the Record 105 reports a couple of segments with baseline wander noise. Our algorithm cleaned up the signal enough to create distinguishable peaks and label them as beats. But varying frequency noise which lies in the signal range gave us a tough time detecting proper peaks and classifying them as normal or ventricular.

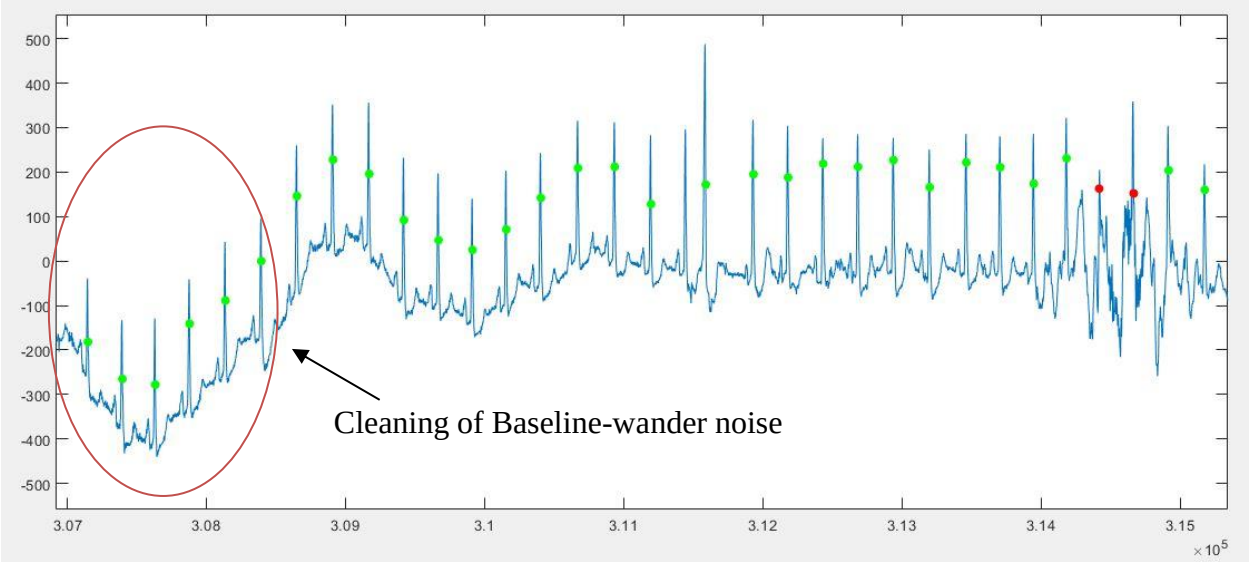


Fig. 5.7: Baseline-wander noise cleaned, and peaks detected by our algorithm

As the baseline wander noise was still manageable, other parts of the signal had some serious noise interference which has also been mentioned in the MIT-BIH arrhythmia database. MIT database states that the pre-dominant feature of this tape of record 105 is high grade noise and artifact [11]. It states that for record 105 out of total 2572 beats detected, 5 are unclassified, 2526 of them being normal and 41 ventricular beats. If these values are to be considered accurate, our algorithm gives errors while classifying the beats. For record 105 we got a total count of 2568 beats out of which our logic labelled 2400 beats as normal while 168 of the beats were classified as Ventricular beats. The leads to about 4% decrease in accuracy. Noise that lies in the most

sensitive ECG frequencies become a big hurdle to overcome. But compared to the high percentage of noise present in the signal, our algorithm performed pretty well giving 95.01% accuracy. The extent of noise invading the ECG signal is presented in Fig. 5.8 below.

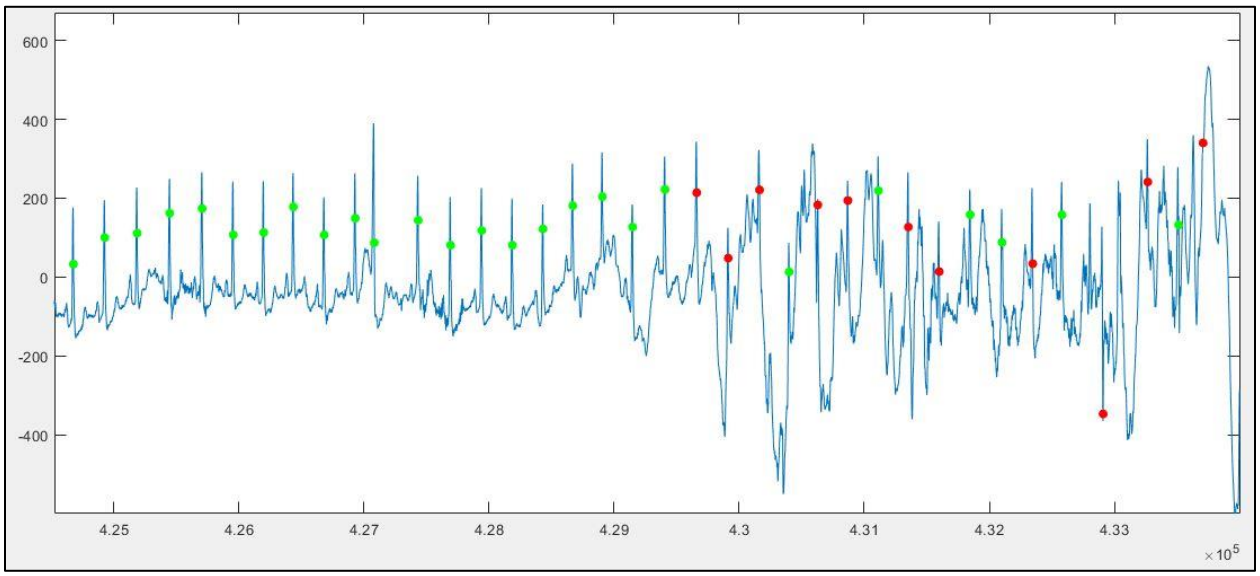


Fig. 5.8: Noise interference in the ECG frequency range, leads to error in results for Record 105

It seems like quite a few noisy peaks got detected as ventricular by our algorithm, which in ideal conditions should have been neglected. Record 105 comes in the league of the worst ECG data sets and for such an input set, a 95.01% accuracy seems affordable but further research will surely be continued for improvements.

### 5.2.3 Analysis of Record 119 from MIT-BIH Arrhythmia Database

Record 119 of the MIT database can be defined as a ECG of a person suffering from acute Ventricular Bigeminy or even in some instances Ventricular Trigeminy [12]. Such a condition of an ectopic beat every second or third beat may arise due to drugs like caffeine or anxiety, but

usually it is mainly the patient suffered previously from hypokalemia, hypomagnesemia and ischemia [8] being an important cause. The symptoms of Ventricular Bigeminy or Trigeminy initially start with palpitations and dizziness which then moves on to more severe symptoms, hence it is advisable to approach medical help once such symptoms show up on regular basis.

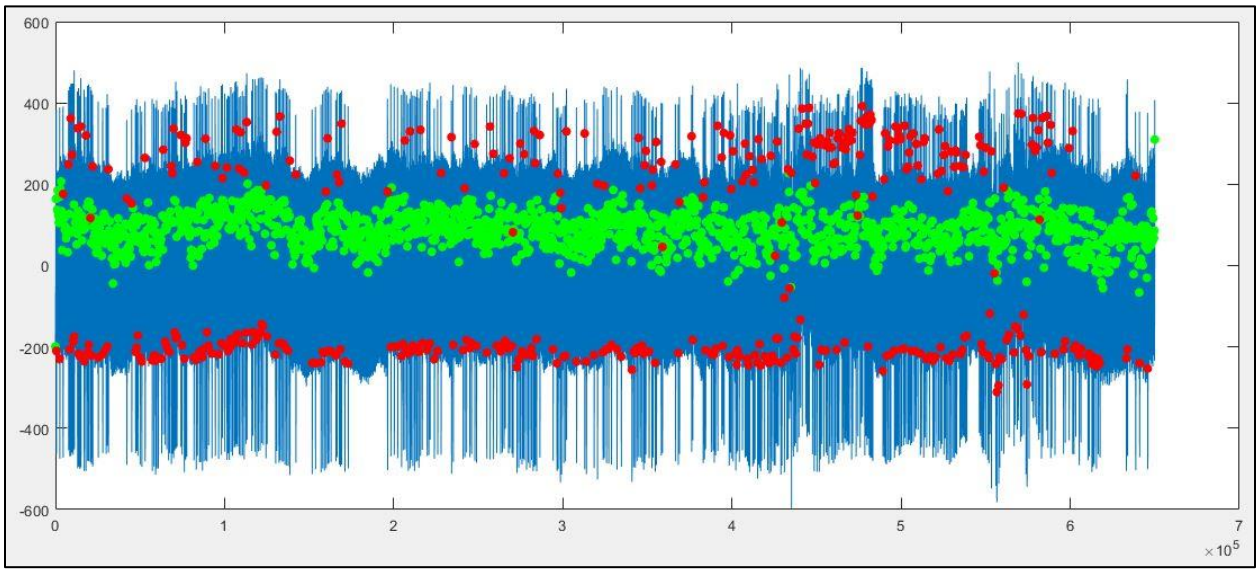


Fig. 5.9: Normal and Ventricular beat locations for Record 119

This record is a clean data set with almost no noise interference. It is purely infected with Premature Ventricular Complexes which our algorithm accurately identified by obtaining 100% accuracy in counting the number of PVCs when compared with the values on the MIT database. The MIT database for record 119 states that out of the total 1987 beats detected, 1543 have been tagged normal whereas 444 of the beats have been labelled as ventricular which mean almost every other beat is a premature ventricular beat, which causes non-rhythmic beating of the heart and also irregular pumping of blood to the body. Our algorithm could identify all the 444 ventricular beats accurately but detected 1989 beats in total, which means that our algorithm detected a couple of

false positives, which gives an error of 0.01% for this particular record. The difference of normal peaks from ventricular peaks can be easily viewed through this record.

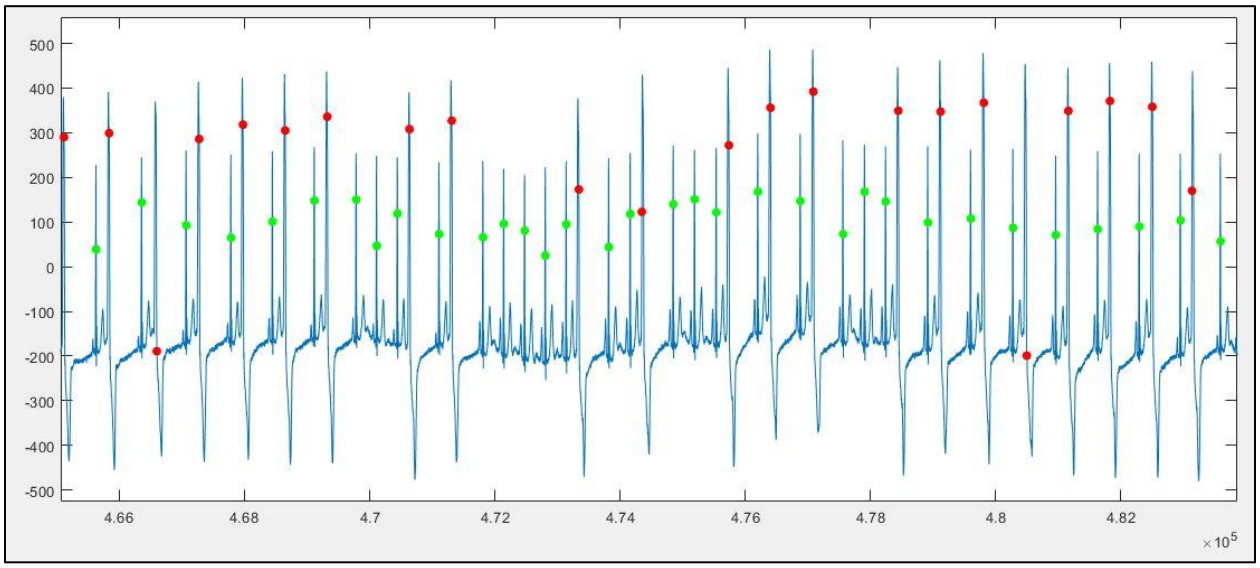


Fig. 5.10: Ventricular beat pattern clearly visible in Record 119

As we see that our algorithm provides almost accurate results when it comes to detecting a heartbeat and analyzing whether it is a Premature Ventricular beat. There are a few cases of false detection, but getting ideal results becomes next to impossible in the real-world scenario where ECG signal are very much prone to noise contamination and other related factors. For a summarizing analysis we did a table-based comparison of our results with that of the MIT database and also calculate the total accuracy and error rate of our algorithm.

After the accuracy of the algorithm the main factor of concern that comes next is how fast is the algorithm going to give results? Also, the hardware resources used, and power consumed or dissipated play an important role while aiming for a good, stable long-term hardware. Our Timing-

constraints report, Mapping, Place and Route, Power and Pin Layout report will elaborate on how effective our design is in the next section.

Table 5.1: Comparing our accuracy rates with MIT database results

| Our Results | | | Noise | MIT Arrhythmia Database Results | | Failure |
|---|---|---|---|---|---|---|
| Record # | Total Beats Detected | Ventricular Beats | Presence of Noise | Total Beats Detected | Ventricular Beats | % Failed Detection |
| 100 | 2273 | 1 | No | 2273 | 1 | 0 |
| 101 | 1866 | 0 | No | 1865 | 0 | 0 |
| 105 | 2568 | 68 | Yes | 2572 | 41 | 0.49 |
| 117 | 1536 | 0 | No | 1535 | 0 | 0.01 |
| 119 | 1989 | 444 | No | 1987 | 444 | 0.01 |
| 200 | 2601 | 816 | Yes | 2601 | 826 | 0.21 |
| Total Beats | 12830 | 1329 | | 12833 | 1312 | 0.72 |
| Failure Rate= $\frac{\text{No. of False Detections}}{\text{Total Beats}} = \frac{45}{12830} \times 100 = 0.0032\%$ | | | | | | |

Table 5.1 presents our algorithm's failure rate to be 0.0032% which means the accuracy of our algorithm is 99.68%. As our algorithm is a modification over the Pan Tompkins algorithm, we compared our accuracy rate with the previously established algorithm which has an accuracy rate of 99.14% [3, 14]. Hence our algorithm is an improvement over the Pan Tomkins algorithm for beat-detection in terms of accuracy by 0.54%. We chose to compare the records present in the table as they have distinct and different characteristics from each other. Record 100 and 117 are pretty clean tapes and helped us understand a pattern in the ECG signal. Record 101 has atrial premature beats, which we wanted to test whether our algorithm detects or not. Record 105 as discussed above has fairly high amount of noise contamination along with a couple of regions with uniform

PVCs. Record 119 and 200 on the other hand are very high on PVCs, and also had noise contamination in the lower channels. We also checked our failure rates for other records, but as we are focusing on hardware implementation, efficient hardware usage is also an important goal. Now we are going to discuss our algorithm's improvement in terms hardware resource utilization and speed.

### 5.3 Device Resource Utilization

As stated previously we chose the SPARTAN-6 family for our research out of which results were performed on the XC6SLX16 device clubbed with the CS324 package and as we did not wish to compromise on performance, we chose the speed grade to be -3[29]. After carefully going through the datasheet for the SPARTAN-6, we made sure that we do not exceed the resource utilization specifications provided by the datasheet. We burned our VHDL code on the NEXYS 3 manufactured by Digilent Inc. The FPGA board has the Xilinx Spartan6 XC6LX16-CS324 embedded in the center with high quality peripherals such as- 16Mbyte Micron Cellular RAM, and 16Mbyte Micron Parallel PCM (Phase Change Memory).

Table 5.2 shows the summary of the total resources occupied by our design on the Spartan6 kit. It can be seen that overall resource utilization is less than 30%. Only the number of bonded Input/ Output buffers seem to be above 50% and it is expected that having fewer IOBs increases the speed of the hardware, but in our case as we have quite a lot of variables, the design automatically allotted IOBs to the intermediate signal variables too, although we have only 27 exterior pins or bonded IOBs, the remaining are not connected to the exterior pins and just have been declared. This would be an advantage to us in future when we modify the circuit.

Table 5.2: Design Summary of our Beat-Detection Algorithm

| Slice Logic Utilization | Used | Available | Utilization |
|---|---|---|---|
| Number of Slice Registers | 299 | 18,224 | 1% |
| Number of Slice LUTs | 501 | 9,112 | 5% |
| Number of fully used LUT-FF pairs | 207 | 499 | 41% |
| Number of MUXCYs used | 300 | 4,556 | 6% |
| Number of Bonded IOBs used | 152 | 232 | 65% |
| Number of BUFG used | 1 | 16 | 6% |
| Number of Occupied Slices | 141 | 2,278 | 6% |
| Average Fanout | 3.29 | | |

When we compared the resource utilization of our algorithm with the Pan Tompkins algorithm, we found that our algorithm uses around 20% less hardware resources as compared to the latter algorithm which uses almost 55% of the FPGA board resources [10]. The idea behind such improvement would be the fact that we used in-built software designs for all our components which helped us to get a compact design due to the automation and also, as we used the latest kit from the Spartan family, we could take advantage of the improved model of the FPGA board.

### 5.3.1 Synthesis Report

Synthesis of our design gave zero errors but came with thirteen warnings which were, basic precautions taken to ensure proper connections throughout the design after it is burnt on the FPGA. This report gives the Macro-statistics and advanced HDL synthesis reports which contain the type of RAM being used, that in our case is 8x1 single-port Read Only RAM, then the detailed use of number of adders, subtractors and multiplexors. After Macro Processing and Low-Level Synthesis,

the final Register report of Macro-Statistics came out to be that 289 registers and flip-flops were used. The REAL time to simulation completion took 9 seconds while the CPU time to read and reach simulation completion for our design took 8.48 seconds. The total memory usage by our design is 344600 kilobytes. Further, detailed table of the synthesis report is provided in the appendix.

### 5.3.2 Timing Constraints Report

No timing constraints were found, and our timing constraints report just presents default enumeration. All our setup and hold times are positive and above 2ns which gives the data path enough time to travel from the external FPGA pin to the internal register that captures the data. Negative values of timing constraints mean that the design cannot be implemented, and the synthesis would be aborted. The minimum period of data travel for our design was- 4.103ns with a maximum frequency of 243.731MHz. Each block in our design has its own gate delay and clock specifications mainly based on the levels of logic which is provided with complete details in the appendix. The peak memory usage to analyze the static timing was 278 MB. A few overall timing details are present in the Table 5.3 below-

Table 5.3: Timing Details and Constraints for our Design

| **Timing Parameters** | **Values** |
|---|---|
| Minimum input arrival time before clock | 6.414ns |
| Maximum output required time after clock | 7.206ns |
| Maximum combinational path delay | 7.781ns |
| Worst case Slack | 0.409ns |
| Best case Achievable | 4.738ns |

### 5.3.4 Power Reports

Power Consumption and dissipation are very important factors while determining whether a hardware is functioning efficiently or not. Power dissipation in our design is determined by the thermal dissipation in the form of heat.

$$\text{Thermal Dissipation} = \frac{\text{Junction Temperature- Ambient Temperature}}{\text{Thermal Resistance}} \quad [15]$$

The Thermal resistance for our design is 84.4 C/W. The Junction temperature provided in our design summary is 27.8 C and the ambient temperature is 25.6 C. If all these values are placed in, then the power dissipation comes out to be 0.026W or 260mW. Previous research has records that power dissipation while using Pan Tomkins algorithm is 566.88mW, which is almost 50% more dissipation as compared to our design as shown in Table 5.4.

Table 5.4: Power Supply and Consumption Summary of our Design

| Supply Power Consumed (mW) | Total | Dynamic Power | Static Power |
|---|---|---|---|
| | 20.38 | 0.48 | 19.90 |

Hence, analysis based on power also gives positive results for our design, making it an acceptable design in all forms. The on-chip power summary is given in details in the Appendix.

While previous software simulation through MATLAB took about 3 minutes for complete analysis, the hardware simulation took 2 minutes 35 seconds to generate results. This simulation time is good enough to generate real-time heart beat analysis. Considering parameters of accuracy, speed and resource utilization, our algorithm passed all the tests for being labelled as an effective algorithm for heart-beat detection.

# CHAPTER 6

## CONCLUSION AND SUGGESTED FURTHER IMPROVEMENT

The main goal of this research was to implement, test and analyze our beat-detection algorithm on hardware, so we can ensure that when our algorithm is implemented on wearable devices, the accuracy and consistency is maintained with results similar to that of the software accuracy. We successfully attained hardware accuracy of 99.68% which matched with the software accuracy rate. Further, we designed the hardware for our algorithm such that it used less than 40% of hardware resources. Consequently, the design consumes less power and acquires good sufficient speed to be practical.

For the de-noising preprocessing stage of our algorithm, we initially had planned to use Wavelet Transform [6], but the function for wavelet transform showed errors during fixed point conversion, which let us to use the bi-directional Butterworth filter instead. Comparing the output results of the bi-Butterworth filter with that of the wavelet transform results, we found that both gave the same waveform as an output and moreover the bi-directional Butterworth filter has a more compact design structure as compared to that of the wavelet transform algorithm hardware design, which led us to finally choose bi-directional filter for the data-cleaning stage of pre-processing.

The code conversion to VHDL was assisted by Simulink, System Generator and HDL Coder which had around 6 sub-modules as our design is based on many functions, that were created as blocks in Simulink, then called by System Generator which provided the environment for our design. The updated versions of these software packages helped us make accurate conversions and produce exact same results as that in software. Our results matched with almost all the results

published in the MIT database and the design is quick enough to give real-time analysis within 2 minutes.

## 6.1 Future Work and Improvements

In order to make our algorithm resistant to noise, we had linear and non-linear filters placed in the pre-processing stage, which aim to extract the ECG signal band from noise interference, but when the noise frequency becomes same as the ECG signal frequency as in record 105, our algorithm mis-interprets a few noisy beats as Premature Ventricular Complexes. We are still working on improving our algorithm and finding ways to get rid of the noise without hampering the ECG signal in the 5-20Hz range. Adding more adaptive techniques to understand and predict the beat rate changes in the ECG signal, like introducing machine learning or deep learning to our algorithm could make our algorithm more efficient.

This research work is highly important in today's world, where the age range of having heart problems is limited not just to the senior citizens of our community. Eating disorders, pollution or even a heathy eating style but a large amount of stress could lead to fatal conditions. That's where daily monitoring of the heart comes into action and the way today's technology has reduced the size of health monitoring devices, it has become obligatory that we provide even more advanced and accurate algorithms to the world. This algorithm could be made versatile by adding logic to find and inform individuals about the other kinds and forms of arrhythmia that could help them prevent life-threatening situations from happening rather than finding a cure for it later.

## APPENDIX

### Synthesis and Implementation Reports

This Appendix includes the synthesized and implemented design reports in detail.

### *A.1 Design Utilization Summary*

| Device Utilization Summary | | | | |
|---|---|---|---|---|
| **Slice Logic Utilization** | **Used** | **Available** | **Utilization** | **Note(s)** |
| Number of Slice Registers | 299 | 18,224 | 1% | |
| Number used as Flip Flops | 287 | | | |
| Number used as Latches | 8 | | | |
| Number used as Latch-thrus | 0 | | | |
| Number used as AND/OR logics | 4 | | | |
| Number of Slice LUTs | 447 | 9,112 | 4% | |
| Number used as logic | 405 | 9,112 | 4% | |
| Number using O6 output only | 236 | | | |
| Number using O5 output only | 60 | | | |
| Number using O5 and O6 | 109 | | | |
| Number used as ROM | 0 | | | |
| Number used as Memory | 0 | 2,176 | 0% | |
| Number used exclusively as route-thrus | 42 | | | |
| Number with same-slice register load | 40 | | | |
| Number with same-slice carry load | 2 | | | |
| Number with other load | 0 | | | |
| Number of occupied Slices | 141 | 2,278 | 6% | |
| Number of MUXCYs used | 300 | 4,556 | 6% | |
| Number of LUT Flip Flop pairs used | 499 | | | |
| Number with an unused Flip Flop | 240 | 499 | 48% | |
| Number with an unused LUT | 52 | 499 | 10% | |
| Number of fully used LUT-FF pairs | 207 | 499 | 41% | |
| Number of unique control sets | 10 | | | |

| | | | | |
|---|---|---|---|---|
| Number of slice register sites lost to control set restrictions | 17 | 18,224 | 1% | |
| Number of bonded IOBs | 152 | 232 | 65% | |
| IOB Flip Flops | 2 | | | |
| IOB Latches | 1 | | | |
| Number of BUFIO2FB/BUFIO2FB_2CLKs | 0 | 32 | 0% | |
| Number of BUFG/BUFGMUXs | 1 | 16 | 6% | |
| Number used as BUFGs | 1 | | | |
| Number used as BUFGMUX | 0 | | | |
| Number of DCM/DCM_CLKGENs | 0 | 4 | 0% | |
| Number of ILOGIC2/ISERDES2s | 0 | 248 | 0% | |
| Number of IODELAY2/IODRP2/IODRP2_MCBs | 0 | 248 | 0% | |
| Number of OLOGIC2/OSERDES2s | 3 | 248 | 1% | |
| Number used as OLOGIC2s | 3 | | | |
| Number used as OSERDES2s | 0 | | | |
| Average Fanout of Non-Clock Nets | 3.29 | | | |

### *A.2 Synthesis Reports*

```
Release 14.7 - xst P.20131013 (nt64)
Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.13 secs
--> Parameter xsthdpdir set to xst
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.13 secs
--> Reading design: ecg_beat_detection.prj
TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Parsing
  3) HDL Elaboration
  4) HDL Synthesis
     4.1) HDL Synthesis Report
  5) Advanced HDL Synthesis
     5.1) Advanced HDL Synthesis Report
  6) Low Level Synthesis
  7) Partition Report
  8) Design Summary
```

8.1) Primitive and Black Box Usage
8.2) Device utilization summary
8.3) Partition Resource Summary
8.4) Timing Report
8.4.1) Clock Information
8.4.2) Asynchronous Control Signals Information
8.4.3) Timing Summary
8.4.4) Timing Details
8.4.5) Cross Clock Domains Report

```
=========================================================================
*                    Synthesis Options Summary                          *
=========================================================================
---- Source Parameters
Input File Name                    : "ecg_beat_detection.prj"
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "ecg_beat_detection"
Output Format                      : NGC
Target Device                      : xc6slx16-3-csg324

---- Source Options
Top Module Name                    : ecg_beat_detection
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : LUT
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Shift Register Extraction          : YES
ROM Style                          : Auto
Resource Sharing                   : YES

Shift Register Minimum Size        : 2
Use DSP Block                      : Auto


---- Target Options
LUT Combining                      : Auto
Reduce Control Sets                : Auto
Add IO Buffers                     : YES
Global Maximum Fanout              : 100000

Register Duplication               : YES
Optimize Instantiated Primitives   : NO
Use Clock Enable                   : Auto
```

```
Use Synchronous Set                : Auto
Use Synchronous Reset              : Auto
Pack IO Registers into IOBs        : Auto
Equivalent register Removal        : YES

---- General Options
Optimization Goal                  : Speed
Optimization Effort                : 1
Power Reduction                    : NO
Keep Hierarchy                     : No
Netlist Hierarchy                  : As_Optimized
RTL Output                         : Yes
Global Optimization                : AllClockNets
Read Cores                         : YES
Write Timing Constraints           : NO
Cross Clock Analysis               : NO
Hierarchy Separator                : /
Bus Delimiter                      : <>
Case Specifier                     : Maintain
Slice Utilization Ratio            : 100
BRAM Utilization Ratio             : 100
DSP48 Utilization Ratio            : 100
Auto BRAM Packing                  : NO
Slice Utilization Ratio Delta      : 5

=========================================================================


=========================================================================
*                     HDL Parsing                       *
=========================================================================
Parsing VHDL file "C:\Users\riagh\Test_code_ise\ecg_beat_detection.vhd" into library work
Parsing entity <ecg_beat_detection>.
Parsing architecture <Behavioral> of entity <ecg_beat_detection>.


=========================================================================
*                     HDL Synthesis                     *
=========================================================================

Synthesizing Unit <ecg_beat_detection>.
    Related source file is "C:\Users\riagh\Test_code_ise\ecg_beat_detection.vhd".
        width39 = 39
        width19 = 19
        back = 76
    Register <med_en> equivalent to <w_med> has been removed
    Found 40-bit register for signal <mx>.
    Found 3-bit register for signal <c_state>.
```

Found 40-bit register for signal <th>.
Found 40-bit register for signal <ivalue>.
Found 20-bit register for signal <peakv>.
Found 20-bit register for signal <reg2<0>>.
Found 20-bit register for signal <reg2<1>>.
Found 1-bit register for signal <w_med>.
Found 40-bit register for signal <to_med>.
Found 32-bit register for signal <count2>.
Found 32-bit register for signal <counter>.
Found 21-bit subtractor for signal <n0130> created at line 249.
Found 32-bit adder for signal <counter[31]_GND_6_o_add_1_OUT> created at line 82.
Found 20-bit adder for signal <counter[19]_GND_6_o_add_52_OUT> created at line 269.
Found 32-bit adder for signal <count2[31]_GND_6_o_add_91_OUT> created at line 308.
Found 20-bit subtractor for signal <GND_6_o_GND_6_o_sub_54_OUT<19:0>> created at line 269.
Found 40-bit adder for signal <n0113> created at line 106.
Found 40-bit adder for signal <n0118> created at line 118.
Found 8x1-bit Read Only RAM for signal <c_state[2]_GND_7_o_Mux_70_o>
Found 1-bit 6-to-1 multiplexer for signal <c_state[2]_n_state[1]_Mux_87_o> created at line 183.
Found 1-bit 6-to-1 multiplexer for signal <c_state[2]_n_state[0]_Mux_89_o> created at line 183.
Found 32-bit comparator greater for signal <n0003> created at line 84
Found 40-bit comparator lessequal for signal <mx[39]_ipt[39]_LessThan_4_o> created at line 85
Found 32-bit comparator greater for signal <GND_6_o_counter[31]_LessThan_17_o> created at line 104
Found 40-bit comparator greater for signal <th[39]_ipt[39]_LessThan_44_o> created at line 208
Found 40-bit comparator greater for signal <ivalue[39]_ipt[39]_LessThan_46_o> created at line 227
Found 41-bit comparator greater for signal <ipt[39]_GND_6_o_LessThan_48_o> created at line 233
Found 20-bit comparator greater for signal <GND_6_o_GND_6_o_LessThan_51_o> created at line 249
Found 32-bit comparator greater for signal <count2[31]_GND_6_o_LessThan_57_o> created at line 286
Summary:
inferred 1 RAM(s).
inferred 7 Adder/Subtractor(s).
inferred 288 D-type flip-flop(s).
inferred 9 Latch(s).
inferred 8 Comparator(s).
inferred 19 Multiplexer(s).
Unit <ecg_beat_detection> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# RAMs : 1
8x1-bit single-port Read Only RAM : 1
# Adders/Subtractors : 7
20-bit adder : 1
20-bit subtractor : 1

```
 21-bit subtractor                                 : 1
 32-bit adder                                      : 2
 40-bit adder                                      : 2
# Registers                                        : 11
 1-bit register                                    : 1
 20-bit register                                   : 3
 3-bit register                                    : 1
 32-bit register                                   : 2
 40-bit register                                   : 4
# Latches                                          : 9
 1-bit latch                                       : 9
# Comparators                                      : 8
 20-bit comparator greater                         : 1
 32-bit comparator greater                         : 3
 40-bit comparator greater                         : 2
 40-bit comparator lessequal                       : 1
 41-bit comparator greate                          : 1
# Multiplexers                                     : 19
 1-bit 2-to-1 multiplexer                          : 13
 1-bit 6-to-1 multiplexer                          : 2
 20-bit 2-to-1 multiplexer                         : 2
 40-bit 2-to-1 multiplexer                         : 2

=========================================================================

=========================================================================
*                  Advanced HDL Synthesis                  *
=========================================================================

Synthesizing (advanced) Unit <ecg_beat_detection>.
The following registers are absorbed into counter <count2>: 1 register on signal <count2>.
The following registers are absorbed into counter <counter>: 1 register on signal <counter>.
INFO:Xst:3212 - HDL ADVISOR - Asynchronous or synchronous initialization of the register <c_state>
prevents it from being combined with the RAM <Mram_c_state[2]_GND_7_o_Mux_70_o> for
implementation as read-only block RAM.
    ----------------------------------------------------------------------
    | ram_type          | Distributed                   |        |
    ----------------------------------------------------------------------
    | Port A
    |     aspect ratio   | 8-word x 1-bit                |        |
    |     weA            | connected to signal <GN       | high   |
    |     addrA          | connected to signal <c_state  |        |
    |     diA            | connected to signal <GND>     |        |
    |     doA            | connected to internal node    |        |
    ----------------------------------------------------------------------
Unit <ecg_beat_detection> synthesized (advanced).

=========================================================================
```

Advanced HDL Synthesis Report

Macro Statistics
# RAMs : 1
 8x1-bit single-port distributed Read Only RAM : 1
# Adders/Subtractors : 5
 20-bit adder : 1
 20-bit subtractor : 2
 40-bit adder : 2
# Counters : 2
 32-bit up counter : 2
# Registers : 224
 Flip-Flops : 224
# Comparators : 8
 20-bit comparator greater : 1
 32-bit comparator greater : 3
 40-bit comparator greater : 2
 40-bit comparator lessequal : 1
 41-bit comparator greater : 1
# Multiplexers : 19
 1-bit 2-to-1 multiplexer : 13
 1-bit 6-to-1 multiplexer : 2
 20-bit 2-to-1 multiplexer : 2
 40-bit 2-to-1 multiplexer : 2

=========================================================================

=========================================================================
*  Low Level Synthesis  *
=========================================================================

Optimizing unit <ecg_beat_detection> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block ecg_beat_detection, actual ratio is 6.
FlipFlop w_med has been replicated 1 time(s) to handle iob=true attribute.

Final Macro Processing ...

=========================================================================
Final Register Report

Macro Statistics
# Registers : 289
 Flip-Flops : 289
=========================================================================

```
=========================================================================
*                      Partition Report                       *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                      Design Summary                         *
=========================================================================

Top Level Output File Name         : ecg_beat_detection.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                      : 958
#      GND                 : 1
#      INV                 : 6
#      LUT1                : 63
#      LUT2                : 120
#      LUT3                : 9
#      LUT4                : 168
#      LUT5                : 122
#      LUT6                : 13
#      MUXCY               : 280
#      VCC                 : 1
#      XORCY               : 175
# FlipFlops/Latches        : 298
#      FD                  : 4
#      FDE                 : 242
#      FDR                 : 43
#      LD                  : 9
# Clock Buffers            : 1
#      BUFGP               : 1
# IO Buffers               : 151
#      IBUF                : 88

#      OBUF                : 63
=========================================================================
```

### *A.3 Timing Report*

Timing Summary:
---

Speed Grade: -3

  Minimum period: 4.103ns (Maximum Frequency: 243.731MHz)
  Minimum input arrival time before clock: 6.414ns
  Maximum output required time after clock: 7.206ns
  Maximum combinational path delay: 7.781ns

Timing Details:
---
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
 Clock period: 4.103ns (frequency: 243.731MHz)
 Total number of paths / destination ports: 14638 / 325
-------------------------------------------------------------------------
Delay:              4.103ns (Levels of Logic = 22)
 Source:           mx_1 (FF)
 Destination:      mx_0 (FF)
 Source Clock:     clk rising
 Destination Clock: clk rising

 Data Path: mx_1 to mx_0

  ----------------------------------------
  Total   Delay            4.103ns (1.736ns logic, 2.367ns route)
                           (42.3% logic, 57.7% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
 Total number of paths / destination ports: 6395 / 198
-------------------------------------------------------------------------
Offset:             6.414ns (Levels of Logic = 9)
 Source:           ipt<27> (PAD)
 Destination:      counter_0 (FF)
 Destination Clock: clk rising

 Data Path: ipt<27> to counter_0

  ----------------------------------------
  Total Delay              6.414ns (2.413ns logic, 4.001ns route)
                           (37.6% logic, 62.4% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'Mram_c_state[2]_GND_7_o_Mux_70_o'
 Total number of paths / destination ports: 553 / 9
-------------------------------------------------------------------------
Offset:             5.430ns (Levels of Logic = 23)

```
  Source:          ipt<0> (PAD)
  Destination:     rw (LATCH)
  Destination Clock: Mram_c_state[2]_GND_7_o_Mux_70_o falling

  Data Path: ipt<0> to rw

    ----------------------------------------
    Total Delay               5.430ns (2.580ns logic, 2.850ns route)
                              (47.5% logic, 52.5% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 648 / 62
-------------------------------------------------------------------------
Offset:             7.206ns (Levels of Logic = 23)
  Source:           counter_0 (FF)
  Destination:      peakd<19> (PAD)
  Source Clock:     clk rising

  Data Path: counter_0 to peakd<19>
    ----------------------------------------
    Total Delay               3.648ns (3.069ns logic, 0.579ns route)
                              (84.1% logic, 15.9% route)
=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 275 / 20
-------------------------------------------------------------------------
  Delay:            7.781ns (Levels of Logic = 24)
  Source:           fromwin1<0> (PAD)
  Destination:      peakd<19> (PAD)

  Data Path: fromwin1<0> to peakd<19>
    ----------------------------------------
    Total Delay               7.781ns (5.081ns logic, 2.700ns route)
                              (65.3% logic, 34.7% route)
=========================================================================
Total REAL time to Xst completion: 9.00 secs
Total CPU time to Xst completion: 8.48 secs
-->
Total memory usage is 344600 kilobytes
Number of errors   :    0 (   0 filtered)
Number of warnings :   13 (   0 filtered)
Number of infos    :    2 (   0 filtered)
=========================================================================
```

### *A.4 Translation Report*

Release 14.7 ngdbuild P.20131013 (nt64)

Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.
Command Line: C:\xilinxnew\14.7\ISE_DS\ISE\bin\nt64\unwrapped\ngdbuild.exe
-intstyle ise -dd _ngo -nt timestamp -i -p xc6slx16-csg324-3 ecg_beat_detection.ngc
ecg_beat_detection.ngd

Reading NGO file "C:/Users/riagh/beat_detection_please/ecg_beat_detection.ngc" ...
Gathering constraint information from source properties...
Done.

Resolving constraint associations...
Checking Constraint Associations...
Done...
Checking expanded design ...

Partition Implementation Status
-------------------------------
  No Partitions were found in this design.
-------------------------------
NGDBUILD Design Results Summary:
  Number of errors:     0
  Number of warnings:   0

Total memory usage is 224920 kilobytes

Writing NGD file "ecg_beat_detection.ngd" ...
Total REAL time to NGDBUILD completion:  4 sec
Total CPU time to NGDBUILD completion:   3 sec

Writing NGDBUILD log file "ecg_beat_detection.bld"...
================================================================================

### *A.5 Mapping Report*

Release 14.7 Map P.20131013 (nt64)
Xilinx Mapping Report File for Design 'ecg_beat_detection'

Design Information
------------------
Command Line   : map -intstyle ise -p xc6slx16-csg324-3 -w -logic_opt off -ol
high -t 1 -xt 0 -register_duplication off -r 4 -global_opt off -mt off -ir off
-pr off -lc off -power off -o ecg_beat_detection_map.ncd ecg_beat_detection.ngd ecg_beat_detection.pcf
Target Device  : xc6slx16
Target Package : csg324
Target Speed   : -3
Mapper Version : spartan6 -- $Revision: 1.55 $
Mapped Date    : Tue Mar 20 23:56:55 2018

Peak Memory Usage:  461 MB
Total REAL time to MAP completion:  26 secs
Total CPU time to MAP completion:  23 secs

==========

### *A.6 Power Report*

```
--------------------------------------------------------------------------------------------------------
|                                                Xilinx XPower Analyzer
--------------------------------------------------------------------------------------------------------
| Release              | 14.7 - P.20131013 (nt64)
|
| Command Line         | C:\xilinxnew\14.7\ISE_DS\ISE\bin\nt64\unwrapped\xpwr.exe -intstyle ise -ol std
ecg_beat_detection.ncd ecg_beat_detection.pcf -o ecg_beat_detection.pwr  |
--------------------------------------------------------------------------------------------------------
-------------------------------
|     Table of Contents       |
-------------------------------
| 1.  Settings                |
| 1.1.  Project               |
| 1.2.  Device                |
| 1.3.  Environment           |
| 1.4.  Default Activity Rates |
| 2.  Summary                 |
| 2.1.  On-Chip Power Summary  |
| 2.2.  Thermal Summary       |
| 2.3.  Power Supply Summary   |
| 2.4.  Confidence Level      |
| 3.  Detailed Reports        |
| 3.1.  By Hierarchy          |
| 4.  Warnings                |
-------------------------------

1.  Settings
---------------------------------------------
1.1                 Device                  |
---------------------------------------------
| Family          | Spartan6                |
| Part            | xc6slx16                |
| Package         | csg324                  |
| Temp Grade      | C-Grade                 |
| Process         | Typical                 |
```

| Speed Grade | -3 |
| Characterization | Production,v1.3,2011-05-04 |

1.2. Environment

| Environment | |
|---|---|
| Ambient Temp (C) | 25.0 |
| Use custom TJA? | No |
| Custom TJA (C/W) | NA |
| Airflow (LFM) | 0 |
| Heat Sink | None |
| Custom TSA (C/W) | NA |

1.3. Default Activity Rates

| Default Activity Rates | |
|---|---|
| FF Toggle Rate (%) | 12.5 |
| I/O Toggle Rate (%) | 12.5 |
| Output Load (pF) | 5.0 |
| I/O Enable Rate (%) | 100.0 |
| BRAM Write Rate (%) | 50.0 |
| BRAM Enable Rate (%) | 50.0 |
| DSP Toggle Rate (%) | 12.5 |

2. Summary

2.1. On-Chip Power Summary

| On-Chip | Power (mW) | Used | Available | Utilization (%) |
|---|---|---|---|---|
| Clocks | 0.48 | 3 | --- | --- |
| Logic | 0.00 | 447 | 9112 | 5 |
| Signals | 0.00 | 606 | --- | --- |
| IOs | 0.00 | 152 | 232 | 66 |
| Static Power | 19.90 | | | |
| Total | 20.38 | | | |

2.2. Thermal Summary

| | |
|---|---|
| Effective TJA (C/W) | 27.8 |
| Max Ambient (C) | 84.4 |
| Junction Temp (C) | 25.6 |

2.3. Power Supply Summary

| | Total | Dynamic | Static Power |
|---|---|---|---|
| Supply Power (mW) | 20.38 | 0.48 | 19.90 |

| Power Supply Currents | | | | |
|---|---|---|---|---|
| Supply Source | Supply Voltage | Total Current (mA) | Dynamic Current (mA) | Quiescent Current (mA) |
| Vccint | 1.200 | 6.51 | 0.40 | 6.11 |
| Vccaux | 2.500 | 3.03 | 0.00 | 3.03 |
| Vcco25 | 2.500 | 2.00 | 0.00 | 2.00 |

## REFERENCES


[1] S. R. Steinhubl, E. J. Topol, “Moving from Digitalization to Digitization in Cardiovascular Care: Why Is it Important, and What Could it Mean for Patients and Providers?”, *Journal of the American College of Cardiology*, Volume 66, Issue 13, 2015, Pages 1489-1496, ISSN 0735-1097.

[2] B. Surawicz, “Brief History of Cardiac Arrhythmias Since the End of the Nineteenth Century”, *Journal of Cardiovascular Electrophysiology.*, (2004) *15*(1), 101-111.

[3] T. Tiwar, “An Efficient Algorithm for ECG Denoising and Beat Detection”, *Master’s Thesis, UMI ProQuest Digital Dissertations,2011,* UMI Number: 1461380.

[4] J. Pan and W. J. Tompkins, “A Real-Time QRS Detection Algorithm”, *IEEE Transactions on Biomedical Engineering, 1985, Volume: BME-32 , Issue: 3 , page(s): 230 - 236 ,* doi: 10.1109/TBME.1985.325532.

[5] P. S. Hamilton and W. J. Tompkins, “Quantitative Investigation of QRS Detection Rules using MIT/BIH Arrhythmia Database”, *IEEE Transaction on Biomedical Engineering,* Vol.BME-33, No.12, December 1986.

[6] V. Kalidas and L. Tamil, "Real-time QRS detector using Stationary Wavelet Transform for Automated ECG Analysis," *2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE)*, Washington, DC, 2017, pp. 457-461. doi: 10.1109/BIBE.2017.00-12

[7] Xilinx Spartan-6 Starter Kit User Guide, UG380(v2.10), March 31, 2017.

[8] American Heart Association website,“http://www.americanheart.org”.

[9] Vital Statistics of the U.S., Data Warehouse, NCHS; “http://www.cdc.gov/nc-hs/datawh.htm, 2004”, ICD10 codes I20I25

[10] P. Hamilton, "Open source ECG analysis," *Computers in Cardiology*, 2002, pp. 101-104. doi: 10.1109/CIC.2002.1166717

[11] MIT- BIH arrhythmia Database, “http://www.physionet.org/physiobank/database/mitdb/”, *Beth Israel Hospital*, Biomedical Engineering Division Room.

[12] S. Sudin *et al*., "Wearable heart rate monitor using photoplethysmography for motion," *2014 IEEE Conference on Biomedical Engineering and Sciences (IECBES)*, Kuala Lumpur, 2014, pp. 1015-1018. doi: 10.1109/IECBES.2014.704762

[13] A. Gupta and U. M. Chaskar, "Hardware implementation and reduction of artifacts from ECG signal," *2015 International Conference on Information Processing (ICIP)*, Pune, 2015, pp. 155-159. doi:

10.1109/INFOP.2015.7489369.

[14] A. Shukla, “Hardware Implementation of Real Time ECG Analysis Algorithms”, *Master’s Thesis, 2008. Retrieved from*-https://scholarspace.manoa.hawaii.edu/-bitstream/10125/.../MS_Q111.H3_4289_r.pdf

[15] J. Kovačević *et al*., "FPGA low-power implementation of QRS detectors," *2014 3rd Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 2014, pp. 98-101. doi: 10.1109/MECO.2014.6862667

[16] C. Li, C. Zheng and C. F. Tai, “Detection of ECG characteristic points using wavelet transforms”, *IEEE Transactions on Biomedical Engineering, 1995;42:2129*, doi: 10.1109/10.362922.

[17] A. Armato *et al*., "An FPGA Based Arrhythmia Recognition System for Wearable Applications," *2009 Ninth International Conference on Intelligent Systems Design and Applications*, Pisa, 2009, pp.660-664. doi: 10.1109/ISDA.2009.246.

[18] T. Debnath, M. M. Hasan and T. Biswas, "Analysis of ECG signal and classification of heart abnormalities using Artificial Neural Network ," *2016 9th International Conference on Electrical and Computer Engineering (ICECE)*, Dhaka, 2016, pp. 353-356. doi: 10.1109/ICECE.2016.7853929

[19] H. Zairi, M. Kedir -Talha, S. Benouar and A. Ait -Amer, "Intelligent system for detecting cardiac arrhythmia on FPGA, " *2014 5th International Conference on Information and Communication Systems (ICICS)*, Irbid, 2014, pp. 1-5. doi: 10.1109/IACS.2014.6841953.

[20] B. U. Kohler, C. Hennig and R. Orglmeister, "The principles of software QRS detection," in *IEEE Engineering in Medicine and Biology Magazine*, vol. 21, no. 1, pp. 42-57, Jan.-Feb. 2002. doi: 10.1109/51.993193

[21] H. A. Kestler, M. Hoher, F. Schwenker and V. Hombach, "Filtering beat-to-beat recordings of the high resolution electrocardiogram," *Computers in Cardiology 1996*, Indianapolis, IN, USA, 1996, pp. 461-464. doi: 10.1109/CIC.1996.542573

[22] 12-Lead ECG Placement Guide- https://www.cablesandsensors.com/pages/12-lead-ecg-placement-guide-with-illustrations

[23] MATLAB Functions- https://www.mathworks.com/help/signal/ref/findpeaks.html

[24] P. Banerjee, D. Bagchi, M. Haldar, A. Nayak, V. Kim and R. Uribe, "Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design," *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, 2003, pp. 263-264

[25] A. Gupta and U. M. Chaskar, "Hardware implementation and reduction of artifacts from ECG

signal," *2015 International Conference on Information Processing (ICIP)*, Pune, 2015, pp. 155-159. doi: 10.1109/INFOP.2015.7489369

[26] S. Roy and P. Banerjee, "An algorithm for converting floating-point computations to fixed-point in MATLAB based FPGA design," *Proceedings. 41st Design Automation Conference, 2004.*, San Diego, CA, USA, 2004, pp. 484-487. doi: 10.1145/996566.996701

[27] Simulink and HDL Coder for FPGA- https://www.mathworks.com/solutions/fpga-design/simulink-with-xilinx-system-generator-for-dsp.html

[28] ECG Input Block for Simulink- https://www.mathworks.com/help/dsp/examples/real-time-ecg-qrs-detection.html

[29] Xilinx Spartan-6 Starter Kit User Guide, UG380(v2.10), March 31, 2017

## BIOGRAPHICAL SKETCH

Ria Ghosh was born in West Bengal India as the eldest daughter of Dr. Ratan Chandra Ghosh and Mrs. Bandita Ghosh. She completed her early schooling from Surrey, London after which she moved back to India and completed her high school from Delhi Public School, Risali. She holds a Bachelor of Engineering degree in Electronics and Telecommunication Engineering from Bhilai Institute of Technology (BIT), Durg from where she graduated in 2016. BIT, Durg is the topmost Engineering and Management Institute in Central India. Along with a good academic standing she was a regular organizer of cultural events at her university and has won many debate competitions for her college. During her undergraduate degree she interned at Bharat Sanchar Nigam Limited, which is the leading Telecomm company in India.

She was admitted to the Master's program in Computer Engineering at The University of Texas at Dallas in August, 2016. During her study at the School of Engineering, she also served as Teaching Assistant and helped in re-designing the course EE/CE1202-Introduction to Electrical Engineering II. She is a research student at the Quality of Life Technology (QoLT) Lab in the Department of Electrical and Computer Engineering at The University of Texas at Dallas.

# CURRICULUM VITAE

# RIA GHOSH

Department of Electrical & Computer Engineering
The University of Texas at Dallas
800 W. Campbell Rd. Richardson, TX 75080
Ria.Ghosh@utdallas.edu | (469)-602-3977

## Education

**August 2016- Present** — **Master of Science with Thesis in Computer Engineering**
**University of Texas at Dallas, USA**

- Current GPA- 3.523
- Courses include Computer Architecture, Design and Analysis of Computer Algorithms, Advanced Operating Systems, VLSI Design, Special Topics in Computer Engineering- Hardware Security, Database Design, Application Specific Integrated Circuit Design.
- Master's Thesis- 'Implementation and Testing of Heart Disease Detection Algorithm on a Field Programmable Gate Array.' Advisor- Dr. Lakshman S. Tamil.

**August 2012- June,2016** — **Bachelor of Engineering in Electronics and Telecommunication**
**Bhilai Institute of Technology, Durg, India**

- GPA –9.00/10
- Course completed- Microwave Communication & Engineering, Network Analysis & Synthesis, Digital Signal Processing, Wireless Communication, Antenna & Wave Propagation, Data Structures & Programming in C.
- Bachelor's Thesis- 'Generating a Novel Random Sequence using a Power Optimized Linear Feedback Shift Register on a FPGA. Advisor- Dr. Manisha Sharma.
- Best Speaker at University Level, 2013, Best Essay Writing, State Level, 2015.

**May 2012** — **High School Degree (Pure Science, STEM)**
**Delhi Public School, Risali (CBSE Board)**

- Graduated with of 88.4% with Physics, Chemistry, Biology and Mathematics.
- Awarded Scholar Blazer for being school scholar for consecutive 9 years.
- Best Script Writer, 2011, Best Debater, Best Painter, 2010.

## Research Interests

Bio-Mechanics, Bio-Informatics, Machine Learning, Optimization of VLSI Circuits, Automated Control Systems, Power Electronics

## Publications

1. Ghosh, Ria, Gowtami, P. and Mishra, S.D. (2016) Design and Analysis of a maximum length 5-Bit Parallel Linear Feedback Shift Register using VHDL Structural Modeling. **International Journal of Innovative Research in Computer and Communication Engineering. 4(4),6750-6757.**
2. Ghosh, Ria (2015), Organic Photosynthetic Cells over Silicon Solar Cells, Ezine Articles (http://EzineArticles.com/expert/Ria_Ghosh/2202212 )

## Academic Projects

**Implementation and Testing of Heart Disease Detection Algorithm on FPGA** Sept,2017- Present

- Aiming towards a Master's with thesis degree, this research work revolves around designing an efficient MATLAB code from a planned Deep Learning based algorithm, then convert to VHDL for burning and Test on an FPGA.

**Design of a Mini Stereo Digital Audio Processor** Nov-Dec,2017

- Currently working on Designing and implementing MSDAP in Verilog, defining the specifications in detail, writing the behavioral model for the MSDAP, a testbench to simulate all cases of the specification and performing functional verification to validate the specification.

**Layout and Verification of a 4x4 RAM Design** Sept- Oct,2017

- Laid out the 4x4 RAM design with 3000 cells using separately generated cells from my cell library to ensure Automatic Placement and Routing of my design using Cadence's Encounter, running Design Rule Check, Layout versus Schematic and running PrimeTime on final layout to ascertain the worst-case delay.

**Instruction Level Parallelism of DES Algorithm to improve the Design Architecture** May 2017

- Coded the DES algorithm in VHDL and C, then modified it to enhance performance, reduce Delay and power using techniques like- Loop Unrolling, Pipelining combined with Loop unrolling, use of register variable and look up tables instead of algorithms.

**Design and Analysis of Performance of Arbiter & Butterfly PUFs on FPGA** Jan-Feb 2016

- To ensure advanced hardware security designed the least resource utilizing PUF architecture and analyzed to what extent the similarly designed PUFs are unclonable by giving a series of challenges and monitoring them on FPGA.

**Wireless Power Transfer Circuit Design** Nov- Dec 2017

- Designed a two-part design with basic electrical components, of which the Transmitter circuit was for wireless voltage generation and the receiver circuit consisted of a receiver coil, rectifier, and regulator was simulated in SPICE circuit simulator useful to charge device like pacemakers.

## Core Competence and Skills

VLSI Design, Computer Architecture, Circuit Design, Hardware and Software Integration, FPGA/CPLD Implementation, PCB Design, Hardware Security, Robotics Design, Research, Coding and Scripting, Debugging and Troubleshooting, Database Design, Testing and Documentation, Computer Algorithms, Data Analysis, Advanced Operating Systems. Languages- C, C++, Python, Verilog, VHDL, MATLAB, System Verilog, Java, SQL. Software: XILINX ISE, MODELSIM, VIVADO, EAD, ECLIPSE, Microsoft Office (Advanced Excel), ELK, LabVIEW.

## Professional Experience

**GRADUATE RESEARCH ASSISTANT** Dallas, Texas

QUALITY OF LIFE TECHNOLOGY LABORTORY (Jan- May 2018)

- Design MATLAB based algorithm for Heart-beat Detection and classification of Arrhythmias.
- Run simulation results and worst-case input test data to analyze accuracy.
- Convert the MATLAB code to VHDL and implement the design on SPARTAN-6 FPGA.
- Design system for Real-time beat classification and analyze hardware resource utilization.

**TECH EDVENTURES (CURRICULUM PRACTICAL TRAINING)** Dallas, Texas

INTERN-ENGINEERING DESIGN AND IMPLEMENTATION (Sept-Dec 2017)

- Implementing Pilot Drone Design, Embedded Controller Hardware Design, Robotic Car Design using Arduino, C++ and Verilog on OEM boards and Genuine MKR1000.
- Ensuring prevention and correction of Technical issues, Testing machines and measurement systems.

**VARDHAMAN EDUTECH PVT. LTD. (INTERNSHIP)** Bhilai, India

HARDWARE DESIGN SUMMER INTERN (May 2015- August 2015)

- Research mechanisms to reduce Power Dissipation of Linear Feedback Shift Register.
- Verilog/ VHDL coding for Power-Optimization of Circuits, burn on FPGA kit to compare Dynamic Power use. Testing, Debugging, Troubleshooting and Documentation.

## Workshops and Extra-Curriculars

1. Data Science with Python Workshop 10th June to 6th August 2017, University of Texas at Dallas.
2. Big Data Club Workshop, 25th June to 7th August,2017, University of Texas at Dallas.
3. Microsoft Workshop on IOT Design, Data Lake creation using Live data via Arduino controlled IC, April,2017.