AN ANALOG NEUROMORPHIC CLASSIFIER CHIP FOR ECG ARRHYTHMIA DETECTION

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Murat Alp Güngen

September 2019

An Analog Neuromorphic Classifier Chip for ECG Arrhythmia Detection
By Murat Alp Güngen
September 2019

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Abdullah Atalar(Advisor)

Ömer Morgül

Itır Köymen

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

Dedicated to the memory of
İsmail Can Özersin
(1994 - 2016)

ABSTRACT

AN ANALOG NEUROMORPHIC CLASSIFIER CHIP FOR ECG ARRHYTHMIA DETECTION

Murat Alp Güngen
M.S. in Electrical and Electronics Engineering
Advisor: Abdullah Atalar
September 2019

Following Moore's Law, the increase in the availability of more processing power alongside the development of algorithms that can use this power, electrocardiogram (ECG) systems are now becoming a part of our daily lives. The analytical detection of irregularities within the ECG scan, arrhythmias, is tricky due to the variations in the signals that differ from people to people due to physiological reasons. In order to overcome this problem, a two stage machine-learning based time-domain algorithm is first developed and tested on MatLab using datasets from the MIT - BIH Arrhythmia Database. The algorithm begins with the *preprocessing* stage where seven features are extracted from the input ECG waveform. These features are then moved onto the second *classification* stage where a perceptron classifies the features as **arrhythmic** or **normal**. The algorithm was then converted into an analog CMOS circuit using the XFAB XC06M3 fabrication process on Cadence Virtuoso. Most of the operations in the preprocessing stage were completed using operational transconductance amplifiers (OTAs). For the classifier, the circuit uses analog floating gate metal oxide semiconductor transistors (FGMOS) to store the weights of the perceptron and a winner-take-all current comparator for the activation function. Simulation results show that the circuit works as intended with a power consumption of 290 μW.

Keywords: Neuromorphics, ECG, arrhythmia, arrhythmia detection.

ÖZET

EKG'DE ARİTMİ TESPİTİ İÇİN BİR ANALOG NÖROMORFİK TANIMLAYICI ÇİP

Murat Alp Güngen
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Abdullah Atalar
Eylül 2019

Moore yasasını takiben veri işleme gücündeki donanımsal ve algoritmik artış sayesinde eskiden hastanelerle sınırlı olan elektrokardiyografi (EKG) sistemleri yavaşça gündelik hayatımızın bir parçası olmaktadır. Farklı kişiler arasındaki fizyolojik farklardan ötürü bir EKG taramasındaki anormallikleri, aritmileri, analitik bir şekilde tespit etmek kolay değildir. Bu sorunun üstesinden gelmek için öncelikle, iki aşamalı makine öğrenme tabanlı zaman bölgesi bir algoritma tasarlanır ve MIT - BIH veri tabanından alınan verileri kullanarak MatLab üzerinde denenir. Algoritma *önişleme* aşaması ile başlar. Bu ilk aşamada sisteme verilen EKG sinyalinden yedi özellik çıkarılır. Bu özellikler daha sonra ikinci *tanımlama* aşamasına aktarılır. Bu aşamada bir algılayıcı özelliklere dayanarak o an işlenen EKG sinyalini **aritmik** veya **normal** olarak sınıflandırır. Algoritma daha sonra XFAB XC06M3 üretim sürecini kullanarak Cadence Virtuoso'da bir analog CMOS devreye dönüştürülür. Önişleme aşamasındaki işlemlerin çoğu işlemsel iletkenlik yükseltici devreleri kullanarak yapılır. Devre tanımlayıcıdaki ağırlıkları depolamak için analog yüzen geçit metal oksit yarı iletken transistörler kullanır. Aktivasyon fonksiyonu içinse bir kazanan-hepsini-alır akım karşılaştırıcısı kullanılır. Simülasyon sonuçları devrenin istenen şekilde çalıştığını gösterir. Devrenin toplam güç tüketimi 290 μW'dır.

Anahtar sözcükler: Nöomorfik, EKG, aritmi, aritmi tespiti.

Acknowledgement

I would like to begin by thanking Prof. Dr. Abdullah Atalar and Asst. Prof. Dr. Hakan Töreyin for their support, patience, theoretical and technical assistance, and hindsight. Without their light, the dark path of this thesis could not be illuminated to completion. I would also like to extend my gratitude to Prof. Dr. Ömer Morgül and Dr. Itır Köymen for being in my thesis committee and helping improve this thesis.

I would also like to thank my fellow (current and past) group mates: Cem Bülbül and Zülkarneyn Şişman for all of their technical assistance, and my fellow graduate students: Çelik Boğa, Buğra Alp Çevikgibi, Muhammed Said Aldemir, Ali Alper Özaslan, and Giray İlhan for their moral support.

From my first days at the department seven years ago to the submission of this thesis, it has been a journey with both ups and downs. I would especially like to thank the Bilkent University Administration, Mürüvet Parlakay, the previous head of department Prof. Dr. Orhan Arıkan and the dean of engineering Prof. Dr. Ezhan Karaşan for all of their support and help in this journey.

Finally, I would like to thank my family, especially my brother, for all of their support every step of the way.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Background

With the steady increase in processing power over the years, incorporating health and environmental monitoring systems into smaller, mobile devices is slowly picking up pace. Medical monitoring systems that in the past could only be implemented in hospitals using bulky, complicated equipment can now fit into the palm of ones hand.

One of the best examples to illustrate this evolution is the Electrocardiogram (ECG). The first device that measured electrocardiographic activity entered the medical stage in 1901. The *Eintoven (String) Galvanometer* [5] [3] was the size of a small car and required the patient to submerse both his/her hands and one foot in separate containers full of a salt solution [3]. The ECG waveform, was printed onto a piece of paper. As the years progressed, the components required to produce an ECG scan were miniaturised. By the 1950's, the device was small enough to fit into a suitcase and move around [4]. Similar to the Einthoven Galvanometer, this version of the ECG scanner also printed the ECG waveform onto a piece of paper but, unlike Einthoven's system, was portable (the entire system fit into a briefcase). Today, technology has progressed to a point where the entire system can fit into a single wrist-mounted device. The 2018 model of the Apple Watch has the ability to take an ECG scan of the wearer with the push of a single button, and can send the result to the users target device of choice.

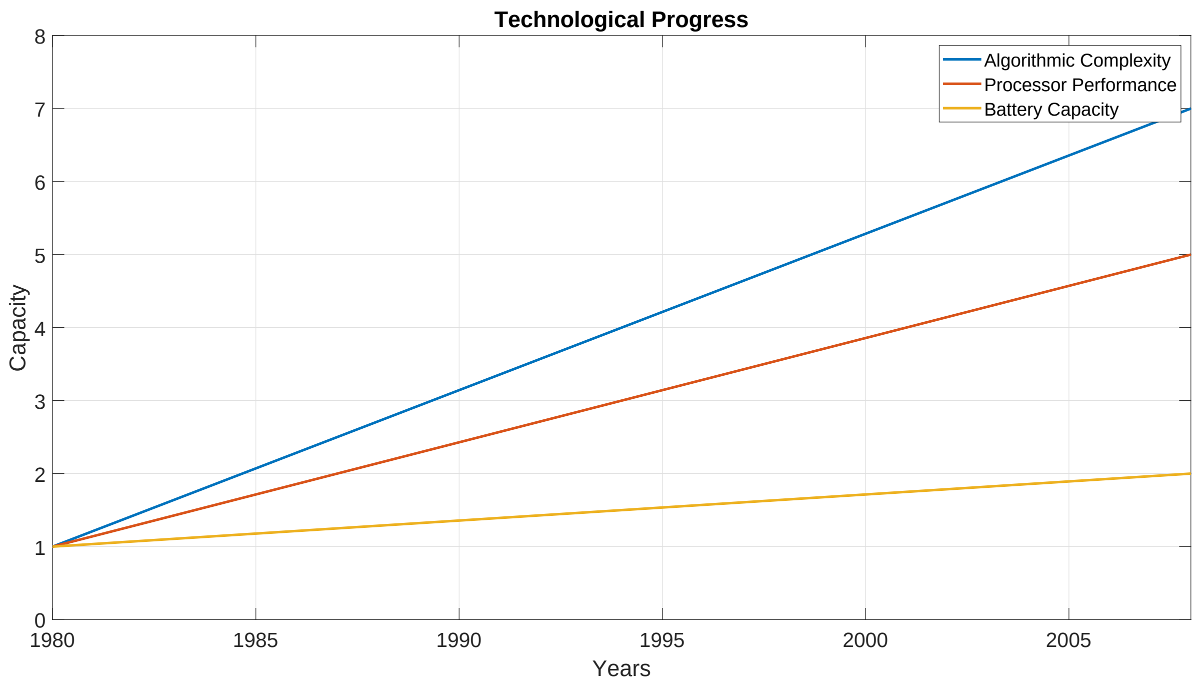


Figure 1.1: The increase in battery capacity has lagged greatly behind the increase in algorithmic complexity and processor performance. The data for this figure was obtained from [1]

While the hardware and the algorithms running on it are becoming more and more complex, energy storage research-and-development has lagged behind (figure 1.1). This has led to devices that have high capabilities but low number of operations when only powered by batteries. In order to overcome this issue, rather than sit idly by waiting for energy-storage technologies to catch up, research is being conducted to decrease the power consumption of algorithms and the hardware they run on without losing precision.

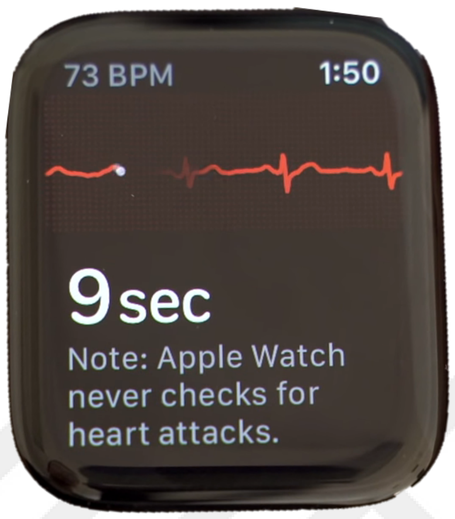


Figure 1.2: Screen-shot of the Apple Watch Series IV midway through taking the ECG of the Author. The Apple Watch is currently the smallest readily and commercially available ECG scanner.

On the algorithmic side, research on machine learning is picking pace thanks to the continuous emergence of more data-sets. Machine learning algorithms, depending on various factors, enable the emulation of certain algorithms at a fraction of the computational cost. This is achieved by finding non-linear correlations between input and target data (provided that the volume of data needed for successful emulation is sufficient). For certain tasks like image recognition and classification, machine learning algorithms can outperform analytical ones. More details on machine learning is available in section 2.1.4.

On the hardware side, analog signal processing is also becoming more widely used. The analog implementation of algorithms can be done with a smaller number of components, leading to smaller and more power efficient implementations (less number of parts requires draws less current). In terms of power saving, analog implementations offer a 20 year leap with respect to their digital counterparts (figure 2.4).

The marriage of machine learning algorithms with analog hardware is a subset of an area known as *neuromorphic systems* (section 2.2). Since its emergence in the 1980's, the area holds promise in enabling the creation of low-power alternatives to various current digital systems.

1.2 Research Objectives

The aim of the work presented in this thesis is to design and implement an analog neuromorphic application specific integrated circuit (ASIC) for real-time ECG arrhythmia detection. The circuit will work in two stages. In the first stage, *preprocessing*, the input ECG waveform (taken from an online database) will be processed in various steps. At the end, various time-domain features will be extracted and fed (as inputs) to the second stage. The second stage, *classification*, uses an analog perceptron to classify the preprocessed ECG waveform as arrhythmic or not. The output will be a simple binary signal (high for arrhythmic).

The chip is intended for usage in mobile devices as a low-power alternative to digital signal processing algorithms used to obtain the same outcome.

The algorithms for each stage will first be implemented in MatLab then converted to analog hardware on Cadence Virtuoso. The **XFAB XC06M3** process will be used as to implement the hardware. All of the required hardware components will be created on Cadence from scratch. Once schematic level simulations are completed, the layout of the circuit will be drawn and post-layout simulations will be performed. The results of these simulations will be compared to the schematic and MatLab simulations.

Overall, the designed system can be used as a proof of concept for the potential benefits of analog neuromorphic chips. The stages will also be designed to show a degree reconfigurability in order to introduce some flexibility for usage in other applications.

1.3 Thesis Outline

Chapter 2 provides an overview of ECG signals, arrhythmias, machine learning and its advantage in ECG arrhythmia detection, and neuromorphics. The first half of chapter 3 begins with the design of the algorithm on MatLab. The second half focuses on the implementation of the components needed for the hardware implementation then proceeds with the hardware implementation itself. Chapter 4 evaluates the simulation results to draw conclusions and offer future improvements.

Chapter 2

Overview

2.1 ECG

2.1.1 What is an ECG?

The Electrocardiogram (ECG or EKG) is a biophysical time-domain signal that shows the activity of the heart. The signal originates from electrical activity in different regions of the heart during a single heart-beat and is measured as a voltage vs. time signal collected from various non-invasive electrodes placed on the human body. It is used by doctors extensively to assess the physiology of the heart under various conditions (rest, physical activity, during surgery, etc.). Starting with the Eindhoven String Galvanometer at the turn of the 20^{th} century [5], ECGs have been around for a long time. Today, the technology has been miniaturised to the point where it can be fitted into a watch and is slowly becoming a part of out daily lives. An example of an ECG waveform is given below in figure 2.1.

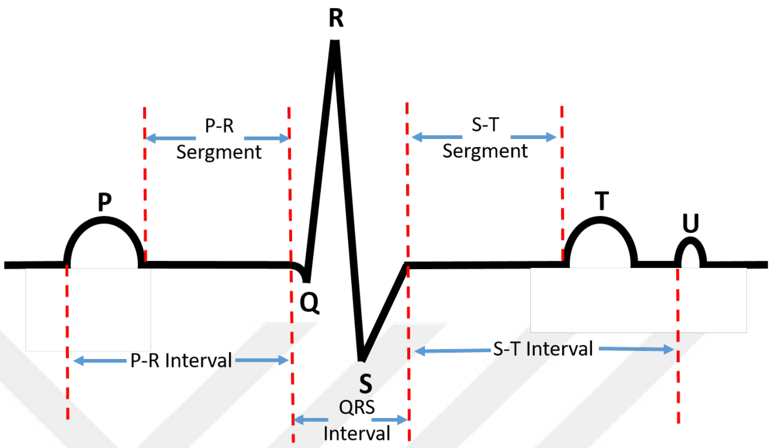


Figure 2.1: Example ECG Waveform. Each region of the waveform corresponds to activity in different areas of the heart. The origin of the components of the waveform are as follows: The **P** wave is and the **QRS** interval (also known as the QRS complex) is generated by the *depolarization* (contraction) of the atrium and ventricles respectively. The **T** and **U** waves are generated by *repolarization* (returning to the resting state) of the ventricles and papillary muscle respectively.

Each segment of the waveform corresponds to various expansion/contractions in different regions of the heart. The ECG signal can be considered as the sum of all of these individual activities.

2.1.2 Arrhythmias

Arrhythmias are irregularities in the operation of the heart with respect to the timing of the beats. They can also be defined as fluctuations in the normal rate or rhythm. The irregularities can arise from a multitude of reasons ranging from physical trauma to genetics.

Whatever the cause, the irregularity arises when one or multiple regions of the heart perform their respective tasks earlier or later than they are supposed to, altering their respective components within the ECG waveform. As can be expected, this causes the ECG waveform to differ from its standard shape, leading to heart problems.

Four common types of arrhythmias are listed below:

- Premature Atrial Contraction (PAC),
- Supraventricular tachycardia (SVT),
- Atrial fibrillation (Afib),
- Premature ventricular contraction (PVC).

Figure 2.2 shows an example ECG with five normal and one arrhythmic heart beat. As can be seen from the figure, the arrhythmic activity can easily be distinguished from the other regular waveforms. The shape can be likened to a malformed vertical reflection of a normal ECG waveform.

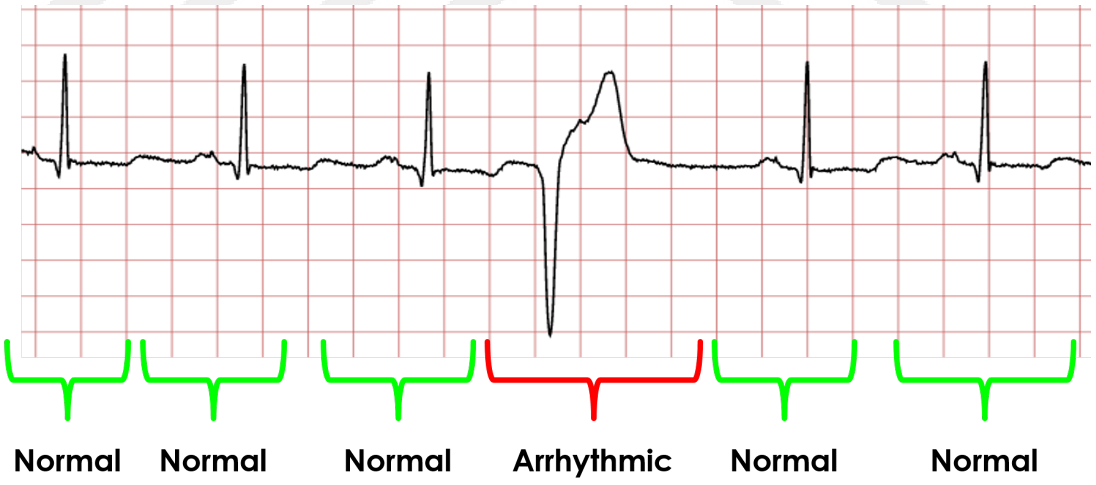


Figure 2.2: Example labelled ECG scan with a single arrhythmic heartbeat.

2.1.3 ECG Arrhythmia Detection and Classification

As mentioned previously, arrhythmias cause time-domain variations in the shape of the ECG waveform. Hence, using various forms of signal processing techniques (both in the time and frequency domains), it is possible to identify an arrhythmic heart beat. However, due to the problems listed below, it is very hard to create an algorithm that uses traditional signal processing techniques for efficient detection and classification of a given waveform as arrhythmic, that works efficiently across different devices and patients. Machine learning algorithms are proposed as a potential solution to this problem thanks to their ability to adapt to variations in the data that can be hard to model.

Issues with ECG classification [6]:

- Lack of standardisation of ECG features,
- Variability amongst ECG features due to physiological differences,
- Variability in the waveforms themselves between patients,
- Noise factors
- variations in the quality of the ECG signal due to the calibration of the machine itself.

2.1.4 Machine Learning

Machine learning, a form of statistical signal processing shown to work exceptionally in non-linear classification problems, offers a solution to the problems that prevent proper arrhythmia classification with only signal processing techniques, mentioned in the previous subsection.

Machine learning algorithms, specifically artificial neural networks, can be used to find the common target features within a given data sample that may be impossible to determine analytically. When a device running these algorithms is presented with a sufficiently large dataset and a specific task (recognition, classification, etc.), it will begin with a training phase. During training, over multiple iterations, the algorithm will adapt itself repeatedly using the data until it achieves the desired task as optimally as possible.

The overall performance of the algorithm depends on multiple factors ranging from the size and quality of the dataset to the way the desired algorithm is implemented. Usually, in order to improve the performance of the algorithm, the data is first pre-processed where noise (if present) is reduced and the relevant features are extracted from the data. The algorithm is then run on these features rather than the entire dataset itself which improves its overall performance.

Features are the parts of the data within the dataset that are of relevant importance to the algorithm (the rest can be considered as noise). Compared to raw data, using extracted features can be more efficient as the algorithm won't have to extract features from the data itself (which, depending on the nature of the feature, may be very difficult or even impossible).

2.1.4.1 Artificial Neural Networks

An artificial neural network is a method of machine learning that draws its inspiration from the biological nervous system. The network consists of individual units called “neurons” that (similar to a biological neuron) multiply their inputs with weights, sum them, then pass the result through an activation function.

The network can be “trained” to respond as desired to different sets of inputs by changing the values of its weights. This enables the neuron to find the relevant features within the input that are of importance to the desired output.

Within the network, individual neurons are arranged in layers (called the input, hidden, and output layers). The weights of each layer learn different patterns from their respective inputs. The neurons then sum their respected weighted inputs, pass the result through a non-linear function known as an ”activation function”, and, unless they are at the output layer, feed the output of their activation functions into the inputs of the neurons on the next layer. The output(s) of the neuron(s) in the output layer becomes the output of the network.

During training, the network utilizes algorithms that update the weights in the direction of the desired output. The output is continuously compared to a target value. The difference between the networks output and the target value is taken, known as the error, and used to update the weights. Weight update algorithms like backpropagation are used to determine the amount with which each individual weight will be updated.

In a neural network that uses backpropagation, each weight is individually summed with an update value consisting of the product of the partial derivative of the error with respective to that weight and other variable to control the magnitude of the update value. The aim is to minimise the output of the error function as much as possible. In other words, to update the weights until minimum error is achieved. The process is shown below in appendix A.4.

2.1.4.2 The Usage of Artificial Neural Networks in ECG Arrhythmia Detection

The availability of the MIT BIH ECG arrhythmia dataset has helped many researchers tackle the ECG classification problem. Over the past 2 decades, there have been multiple academic publications on the usage of artificial neural networks for the detection and classification of various types of ECG arrhythmias.[6]

These publications show that artificial neural networks coupled with properly extracted time or frequency domain features can overcome the problems standard signal processing algorithms face.

Some of the popular features utilised by these publications are listed below:

- **RR Interval:** The time duration between two R waves,
- **R Peak:** The amplitude of the R wave,
- **QRS Duration:** The time duration of the QRS wave,
- **QT Duration:** The time duration from the starts of the Q segment to the end of the T wave,
- **PR Interval:** The time duration from the start of the P wave to the end of the R wave,
- **U Peak:** The amplitude of the U wave (if available),
- **Frequency:** For extracting heart rate from the ECG scan.

2.2 Neuromorphics

2.2.1 What are Neuromorphics?

Neuromorphics or *Neuromorphic Systems* are defined as systems (mainly consisting of analog or digital electronic circuits) that mimic various properties of biological neurons and neural systems. The definition also encompasses analog and digital implementation of artificial neural networks. Different types of systems in which neuromorphic algorithms were implemented in can be seen in figure 2.3.

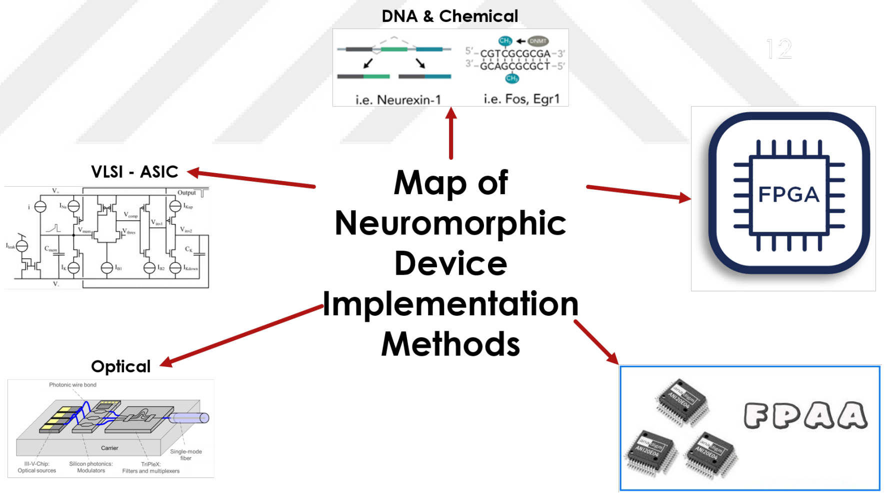


Figure 2.3: Example labelled ECG Waveform with a single PAC heartbeat.

The area began in the late 1980s with the publication of Carver Mead's *Analog VLSI and Neural Systems* [7]. Since then, multiple innovations have been made in the area ranging from new sensor and information processing technologies to new types of hardware architectures. [8]

2.2.2 Analog Neuromorphic Systems

The greatest advantage neuromorphic systems present is power. Neuromorphic systems mimic the biological nervous system which itself is the most power-efficient information processing system in nature. The human brain has the information processing capacity far greater than a supercomputer yet consumes orders of magnitude less power. Gene's law (similar to Moore's law), illustrated in figure 2.4, shows that analog circuits have a potential 20 year leap with respect to their digital counterparts in terms of power consumption.

Compared to their digital counterparts, an analog implementation of a neuromorphic system (e.x. a neural network) can achieve similar results with a smaller number of transistors consuming less current. Hence, an analog neuromorphic circuit will consume less power.

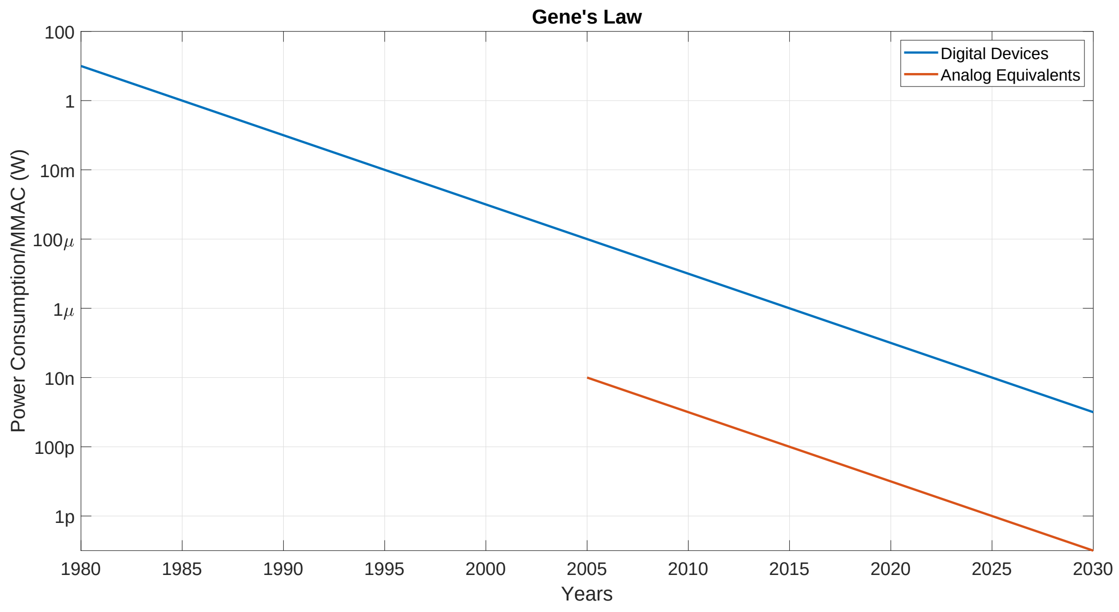


Figure 2.4: Gene's Law shows that analog signal processing circuits have a 20 year leap with respect to their digital counterparts in terms of power consumption (Power/(Million Multiply Accumulate Cycles per second)). Data for this plot was obtained from [2]

Analog neuromorphic circuits generally utilise cutting-edge/experimental memory technologies in order to store weight values. Examples of these technologies are:

- Analog Floating Gate MOSFET transistors,
- Memristors,
- Phase Change Materials.

The above listed technologies are all non-volatile and low power memory storage components. For neuromorphic applications, these components are generally used to store weights which can be preprogrammed or updated by the system itself as it is being used.

Research in this area has also led to the commercialization of certain products. *Audience* and *Synaptics* are examples of two companies, founded by researchers who have worked on the area of neuromorphics, that produce biologically inspired circuits for audio and tactile sensing circuits respectively. Some of the customers of the chips produced by these companies include Apple, Google, and HP.

Chapter 3

Proposed Algorithm for ECG Arrhythmia Detection and Simulations

3.1 Sample ECG Waveform

For simplicity the waveform shown in figure 3.1 will be used to illustrate the stages of the algorithm. Simulation results with additional waveforms can be found in figure 3.43. Information on the ECG dataset used can be found in appendix A.1.

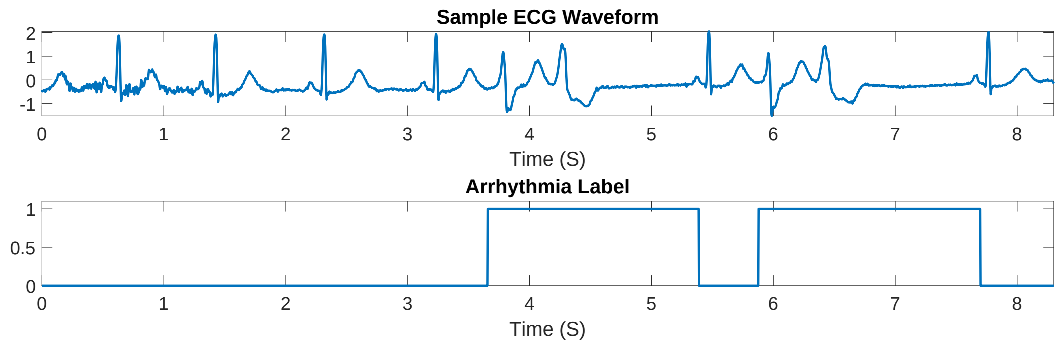


Figure 3.1: The sample ECG waveform alongside the labels highlighting the arrhythmic beats.

3.2 The Algorithm

The algorithm proposed for ECG arrhythmia detection takes a raw ECG waveform as its input and outputs a binary value, indicating whether the waveform is arrhythmic or not (**High** = *Arrhythmic*, **Low** = *Normal*). It consists of two stages: *preprocessing* and *classification*. The preprocessing stage itself uses various techniques to extract seven features from the input waveform. These seven features are then fed into the input of the second, classifier, stage where a single artificial neural network (also known as a perceptron) is used to classify the ECG waveform as arrhythmic or not. The algorithm was first implemented on MatLab and later converted to an analog ASIC. Due to the analog nature of the ASIC, the algorithm is limited to time-domain methods (as frequency domain methods require the FFT (Fast Fourier Transform) of the waveform to be taken which cannot be implemented with analog components.

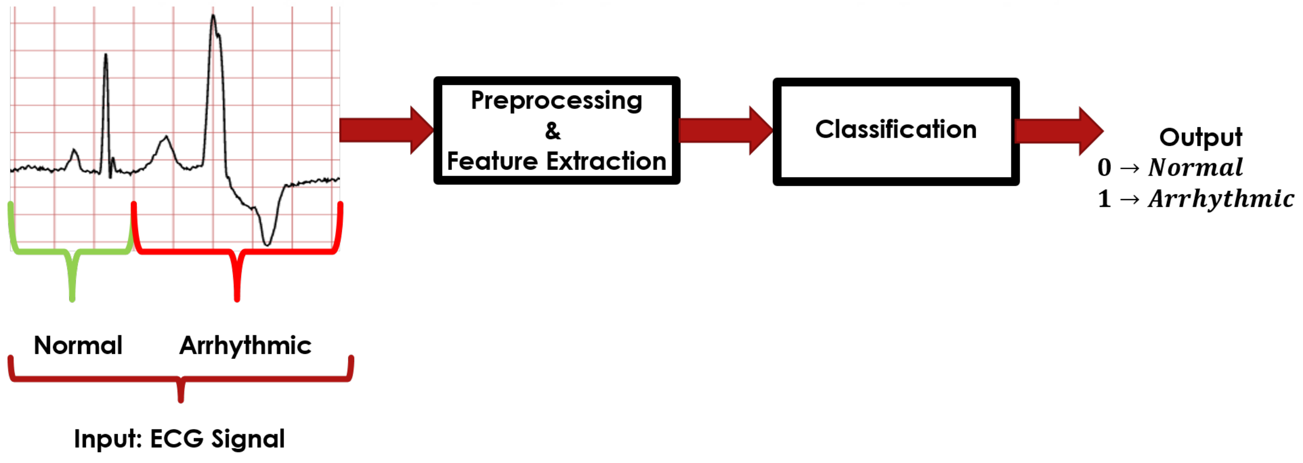
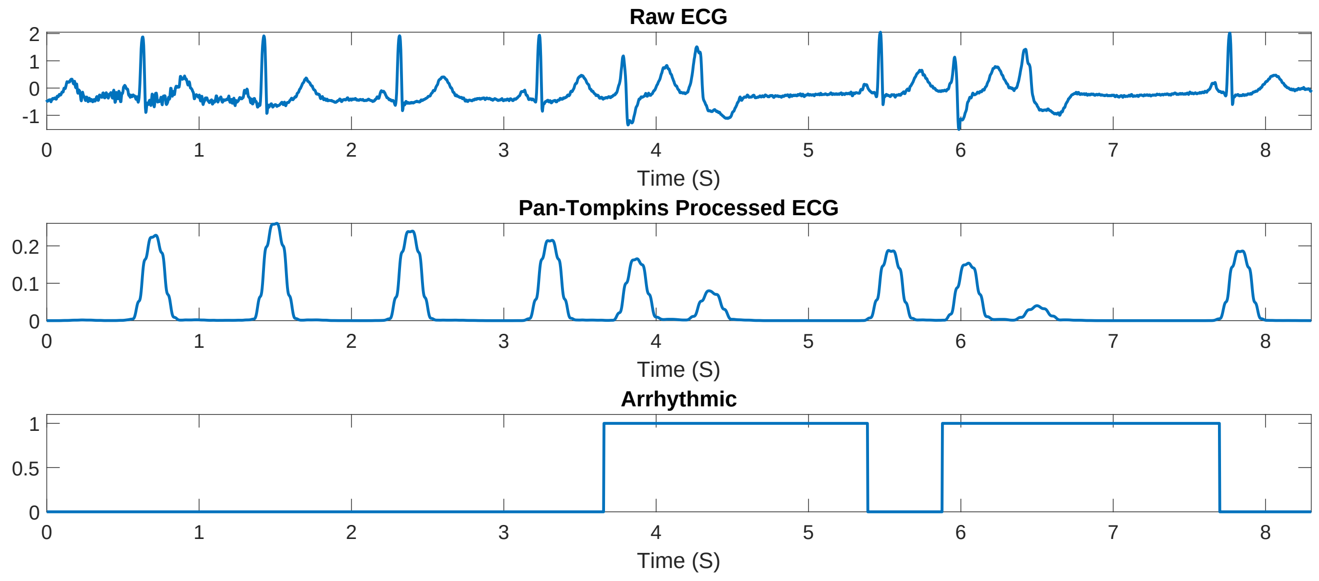


Figure 3.2: The stages of the algorithm.

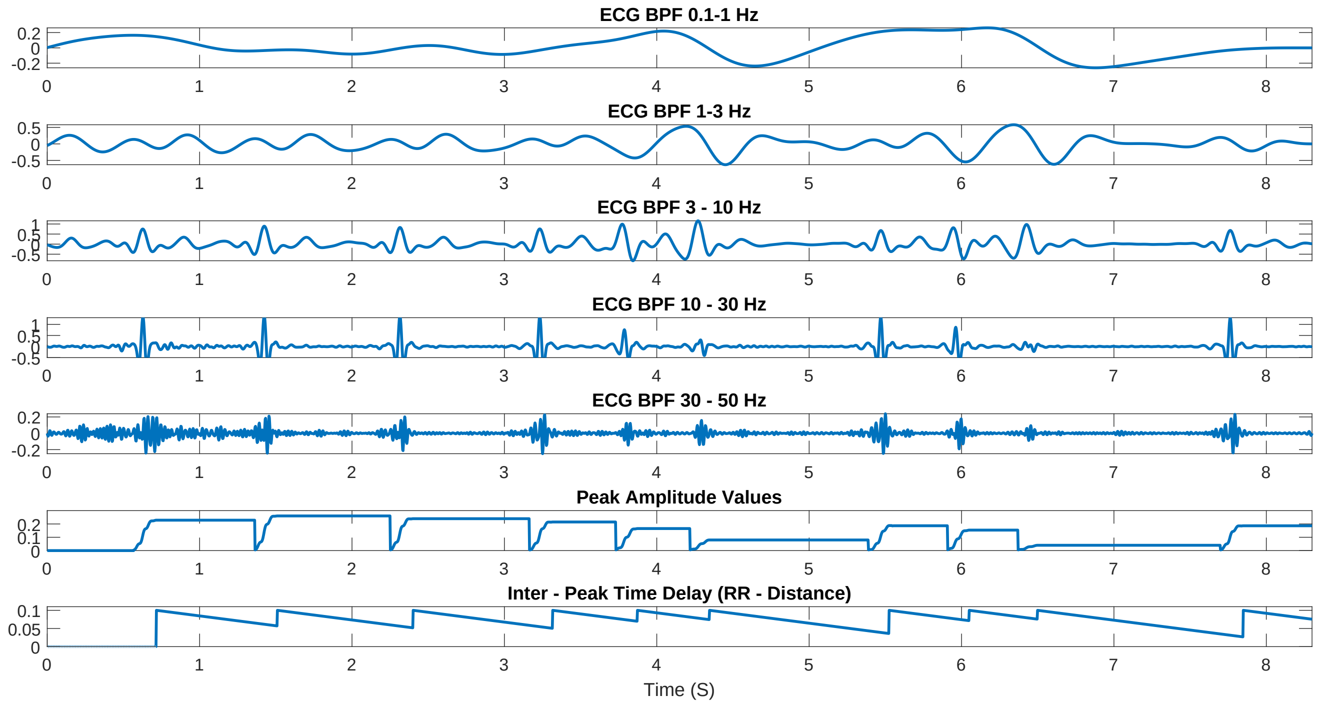
3.2.1 The Preprocessing Stage

As mentioned above, the preprocessing stage of the algorithm extracts seven features from the raw ECG waveform. plots of these features are given below on figure 3.3b. These features are:

1. The input ECG waveform bandpass filtered at the 0.1-1Hz interval,
2. The input ECG waveform bandpass filtered at the 1-3Hz interval,
3. The input ECG waveform bandpass filtered at the 3-10Hz interval,
4. The input ECG waveform bandpass filtered at the 10-30Hz interval,
5. The input ECG waveform bandpass filtered at the 30-50Hz interval,
6. Pan-Tomkins algorithm (figure 3.3a) processed inter-peak time duration,
7. Pan-Tompkins algorithm (figure 3.3a) processed peak amplitude value.



(a) The input ECG waveform, its Pan-Tompkins algorithm output, and arrhythmia labels.



(b) The seven extracted features from the sample ECG waveform.

Figure 3.3: The sample waveform and extracted features.

3.2.2 Differences Between Normal and Arrhythmic ECG Waveforms at Different Frequency Bands

As can be seen from figure 3.3b, normal and arrhythmic waves differ greatly at different frequencies. When filtered at different frequency bands, the amplitudes of the regions of the filtered signal corresponding to the arrhythmic waves are either smaller or greater with respect to the normal waves.

In the 0.1 - 1 Hz, 1 - 3 Hz and 3 - 10 Hz frequency bands, the waveforms corresponding to the arrhythmic beats have larger peak-to-peak amplitude differences. The opposite is observed on the for the 10 - 30 Hz and the 30 - 50 Hz bands.

These differences highlight the abnormal nature of arrhythmia. When closely inspected, normal ECG waves at different bands appear periodic, their behaviour easy to deduce whereas the anomalies introduced by arrhythmias are more random.

3.2.3 The Pan-Tompkins Algorithm

The most important part of the preprocessing stage is the Pan-Tompkins algorithm.[9] Developed in 1985, the purpose of this algorithm is to ease the real-time detection of the QRS complex (also referred to as *QRS interval*, figure 2.1) of an ECG waveform which can be problematic due to noise or medical conditions. The stages of the full algorithm are listed below.

1. **Filtering:** The waveform is filtered with a bandpass filter at the 5-15 Hz interval.
2. **Differentiation:** The derivative of the filtered signal is taken, with respect to time.
3. **Squaring:** In order to remove the negative values of the signal, the square of the differentiated signal is taken.
4. **Integration:** A moving window integrator is used to collect the individual small peaks of the same QRS wave generated from the previous steps into a single peak.
5. **Fiducial Mark Detection:** The rising edge of the integrator stage output is used to mark the start of the QRS complex.
6. **Adaptive Thresholding:** Converts the integrated signals into pulse streams where each *pulse* corresponds to the temporal location of a single QRS complex.

Only the first four of the stages of the algorithm is used for the preprocessing stage as the inclusion of the following stages actually deletes some of the features of the ECG waveform that can be beneficial for classification (e.x. ECG peak amplitude). Graphical results of each stage of the Pan-Tompkins algorithm applied to the sample ECG waveform can be seen on figure 3.4.

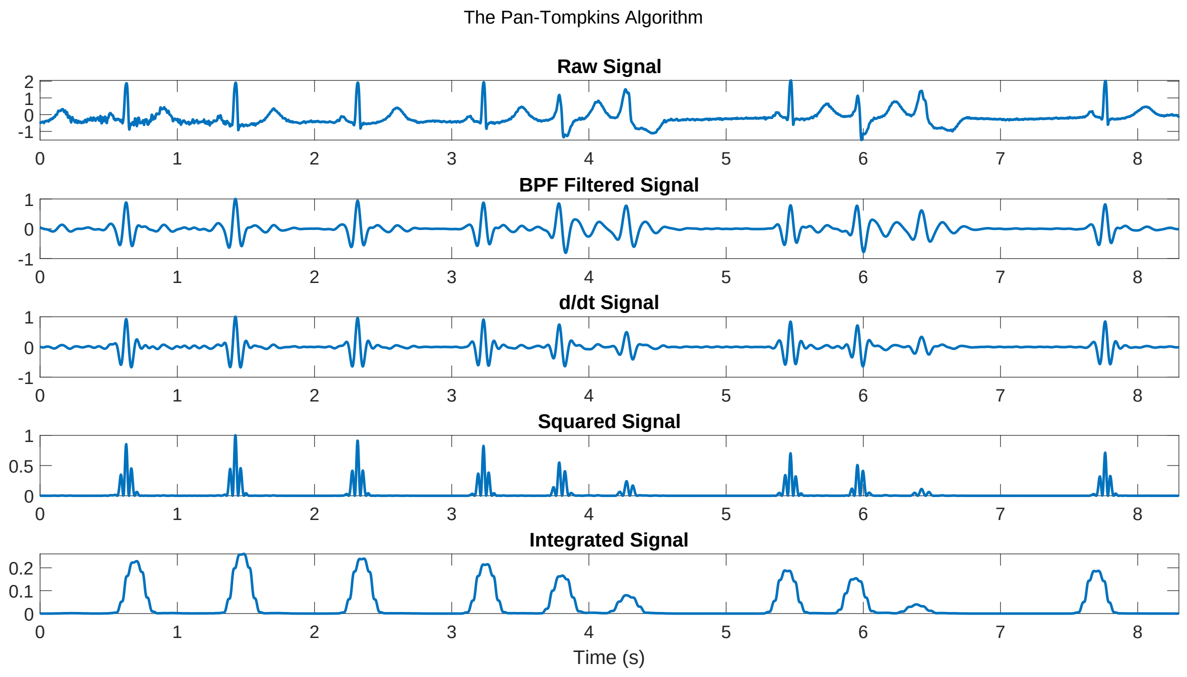


Figure 3.4: The relevant stages Pan-Tompkins algorithm applied to the sample arrhythmic ECG signal.

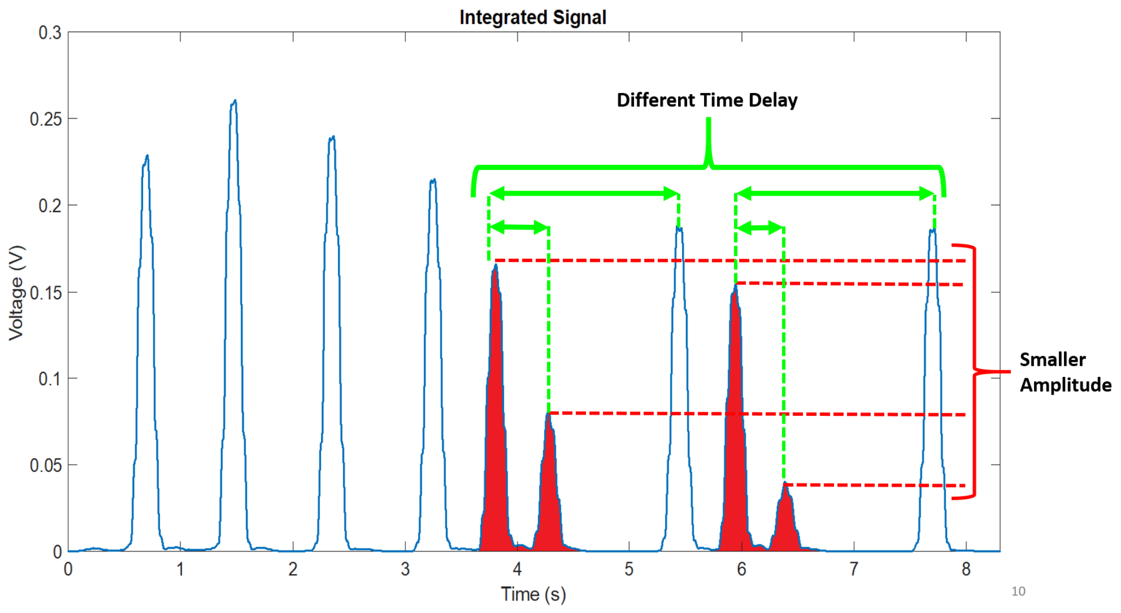


Figure 3.5: Closeup of the final stage from figure 3.4 with the arrhythmic waveforms (highlighted in read) differences between normal and arrhythmic waveforms indicated.

3.2.3.1 Differences Between Normal and Arrhythmic ECG Waveforms Following the Implementation of the Pan-Tompkins Algorithm

Figure 3.5, on the previous page, shows a close-up of the final segment of figure 3.4 with the two arrhythmic waveforms highlighted in red. As can be seen from this figure, the arrhythmic waves, with respect to their normal counterparts, have smaller peak amplitudes and varying time delays. The time delay within the peaks of the same arrhythmic wave is much smaller that the one between two normal ECG waves. The time delay following the end of the arrhythmic wave, on the other hand is much longer. Biologically, this longer time delay corresponds to the a short rest period in the heart following arrhythmic beats. These two critical differences also

3.2.4 The Classification Stage

The second stage of the algorithm focuses on the classification of arrhythmic waves based on the outputs of the preprocessing stage. The selected classifier for this stage is the perceptron.

The perceptron, a single neuron from an artificial neural network (section 2.1.4.1), works by multiplying its inputs with a set of weights, summing them up along with a bias value and passing the result through an activation function. Mathematically, the following equation is used to describe the operation of a perceptron:

$$O = f\left(\sum_{i=1}^{n} \omega_i x_i\right) \rightarrow O = \begin{cases} 1, \ \sum \omega_i x_i \geq \theta \\ 0, \ \sum \omega_i x_i < \theta \end{cases} \tag{3.1}$$

Where O is the output, *f()* is the activation function, θ is the threshold value used by the activation function, and ω_i and x_i $(i \epsilon n)$ are the respective products of the ith input and weight of the perceptron. O is a binary output, meaning that it is either equal to *1* or *0*.

Mathematically, the perceptron could also be considered as a comparator where the activation function does the comparison. If the sum of the weighted inputs is greater than a threshold value, the perceptron outputs 1. Otherwise it will output 0.

3.2.5 Positive Weighted Perceptron

In order to ease the complexity of the hardware design, negative currents should be removed from the system. On the input, this is easily achieved by adding an offset that makes the entire input ECG waveform positive. The squaring function (and respective circuit) in the Pan-Tompkins algorithm deals with the negative outputs of the differentiator, eliminating the problems with negative currents in the preprocessing stage.

As artificial neural networks usually include both positive and negative valued weights, eliminating the negative currents that arise from the product of the inputs (all positive) with these weights can't be done in a straightforward manner such as adding an offset.

Instead, using the comparator feature of the perceptron is used with the following procedure: First the weighted inputs are separated into the positive and negative values and summed separately. Next, θ, depending on its sign is added to its corresponding sum. Finally, the negative sum consisting of the negatively weighted signals is moved to the other side of the comparator function, turning it positive. The resulting equation implements the same function as the perceptron without any negatively valued signals. Mathematically, the described operations are shown below:

$$\sum \omega_i x_i \geq \theta = \sum_{\omega>0} \omega_i^+ x_i + \sum_{\omega<0} \omega_i^- x_i \geq \theta$$

Where:

$$\sum_{\omega<0} \omega_i^- x_i = -\sum_{\omega<0} |\omega_i^- x_i|$$

Hence:

$$\sum \omega_i x_i \geq \theta = \sum_{\omega>0} \omega_i^+ x_i - \sum_{\omega<0} \left|\omega_i^- x_i\right| \geq \theta$$

Depending on the sign of θ, the new comparator equations become one of the following:

$$\begin{aligned} \theta \geq 0 &\rightarrow \sum \omega_i x_i \geq \theta \Rightarrow \sum_{\omega>0} \omega_i^+ x_i \geq \sum_{\omega<0} \left|\omega_i^- x_i\right| + \theta \\ \theta < 0 &\rightarrow \sum \omega_i x_i \geq \theta \Rightarrow \sum_{\omega>0} \omega_i^+ x_i + \theta \geq \sum_{\omega<0} \left|\omega_i^- x_i\right| \end{aligned} \tag{3.2}$$

This positively weighted comparator can easily be implemented using a class of circuits known as *Winner-Takes-All* (WTA) comparators.

3.3 MatLab Implementation of the Algorithm

The algorithm was implemented on MatLab. For the Pan-Tompkins algorithm, the relevant parts of the code from [10] were extracted. The remaining parts of the code were written from scratch. At the end of the preprocessing stage on MatLab, the data is transformed into the form shown in figure 3.3b. The algorithm was used on the entire dataset. Afterwards, the processed data was split into separate *training* and *testing* sets with a 70:30 ratio (training and testing respectively).

For the classifier stage, a single layer artificial neural network with one output (similar to a perceptron) was created on MatLab's *neural network/data manager* toolbox and the training set was used to train the network. A *TANSIG* activation function was chosen as it is the one that most resembles a comparator. Running the test set on the network showed an overall 85% precision. A closer examination of the results is given in the next subsection.

3.3.1 MatLab Simulation Results

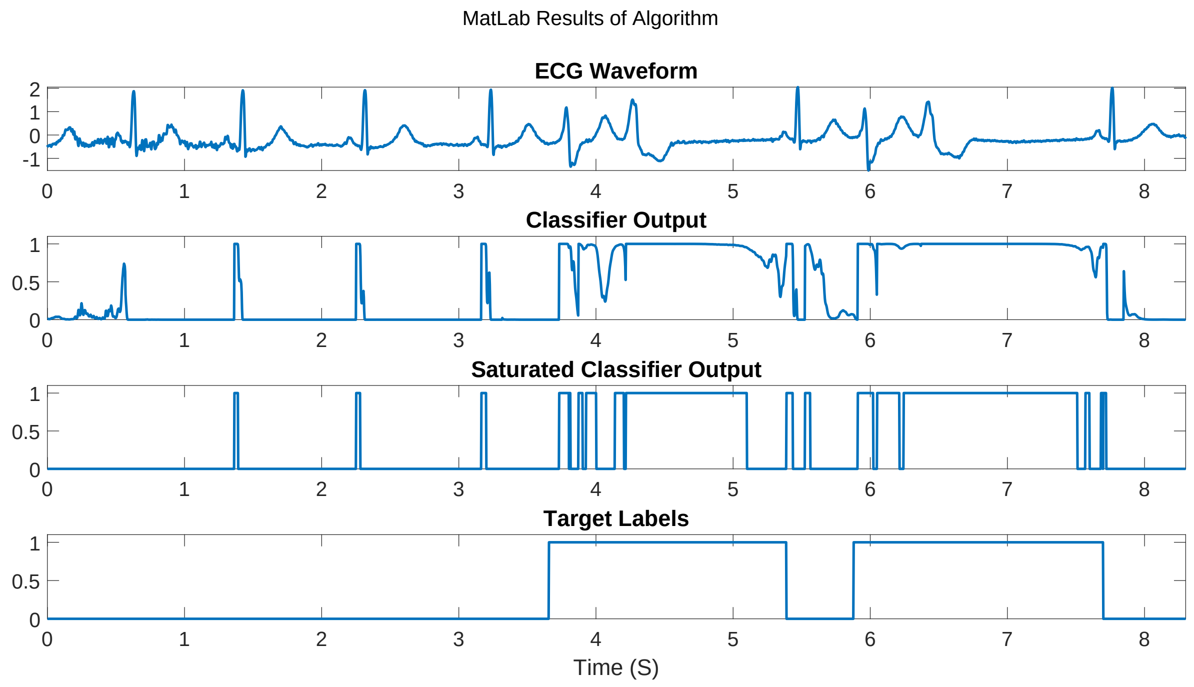


Figure 3.6: The ECG sample waveform, MatLab processed outputs, and the target labels.

Figure 3.6 above shows the output of the MatLab implementation of the algorithm on the sample ECG waveform. The first plot shows the waveform itself, the second plot shows raw output of the classifier. In order to improve the results, thresholding (set at 0.95) was applied, the results of which are shown in the third plot. The fourth and final plot shows the target labels.

As can be seen from the results above, the algorithm is successful in classifying the two arrhythmic beats present in the sequence. However, there are multiple false negatives and positives present in the output that should be addressed.

False positives indicate the existence of an arrhythmic beat when there isn't one. The false positives outputted by the algorithm could be seen to occur with the R portion of the ECG waveform. Their duration is also consistent with that of the R segment. As the duration of the arrhythmia is much greater than the duration of the R segment (and hence the false positives) the false positives can be removed with the application of a low-pass filter.

False negatives indicate the absence of an arrhythmic beat when there actually is one. The mainly occur around the start and end of the arrhythmic beat and occur at a higher frequency (based on their short intervals, similar to that of the false positives). They can be removed from the data with the inclusion of a low-pass filter at the output.

In analog circuits, an integrator behaves like a low-pass filter. A basic moving window integrator was applied to the output of the saturated algorithm output yielding the results below in figure 3.7. As can be seen from these new results, the false positives and negatives have disappeared. While the duration and position of the processed outputs may differ from the target labels, this should not be too much of a problem as the aim of this work is the detection/classification of the presence of arrhythmic ECG waveforms within the scan.

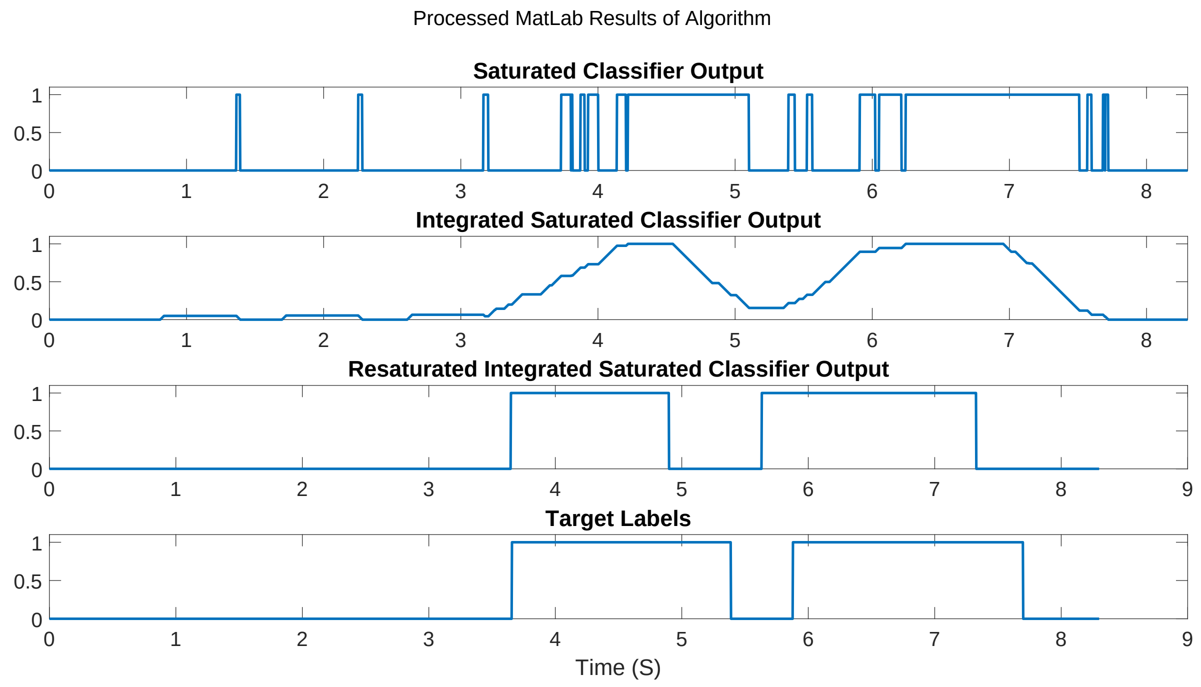


Figure 3.7: Processing the saturated algorithm outputs with an integrator followed by a second saturation removes the high-frequency false positive and negatives.

3.4 Hardware Implementation of the Algorithm

As the algorithm consists of time domain methods, mapping it onto an analog IC only requires utilizing/designing the relevant components of each stage of the algorithm and connecting them to each other. All of the components were created from scratch using transistors and capacitors from XFAB's XC06M3 process.

3.4.1 Hardware System Components Overview

The main component used in most throughout this chip is the operational transconductance amplifier. It is used to implement most of the stages of the Pan-Tomkins algorithm. The names of the circuits and their corresponding functions in the algorithm are listed below:

1 **The Operational Transconductance Amplifier** - Used in most components.

2 C^4 **Bandpass Filter** - Used to implement the filter bank and the first stage of the Pan-Tompkins algorithm.

3 **Differentiator** - Used in the second stage of the stage of the Pan-Tompkins algorithm.

4 **Current Squarer** - Used in the third stage of the Pan-Tompkins algorithm.

5 **Leaky Integrator** - Used in the fourth stage of the Pan-Tompkins algorithm, and at the end of the classifier.

6 **Peak Detector** - Used to extract the first part of the pre-processing stage of the algorithm.

7 **RR Distance Finder** - Used to extract the second part of the pre-processing stage of the algorithm.

8 **FGMOS Transistors** - Used to store the weights of the classifier.

9 **Vector Matrix Multiplier** - Used to implement the matrix multiplication stage of the classifier.

10 **Winner-Take-All Comparator** - Used to implement the transfer function of the classifier.

3.4.2 Circuits Designs and Layouts

All of the plots shown in the subsections below, unless otherwise stated, are created using data from Cadence simulations of actual corresponding circuits processed on MatLab.

3.4.2.1 The Operational Transconductance Amplifier

The operational transconductance amplifier (OTA, figure 3.8) is a type of amplifier which generates an output current based on its differential input voltage (difference between the '+' and '-' terminals). It could also be considered as a differential voltage controlled current source.

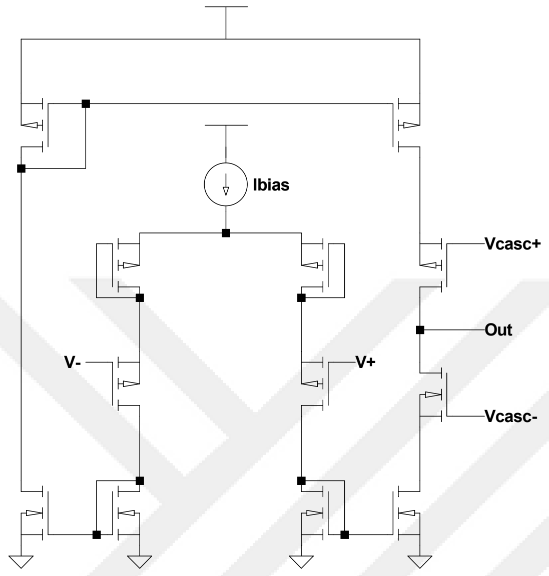
The *transconductance* (denoted as $\mathbf{G_m}$) of the amplifier deduces the output current/input differential voltage relationship. Ideally, this relationship expressed mathematically as shown in equation 3.3 below. The transconductance value itself, depends on the magnitude of the bias current (implemented using current mirrors connected to a reference bias current).

$$I_{Out} = G_m(\mathbf{V}_+ \; - \; \mathbf{V}_-) \tag{3.3}$$

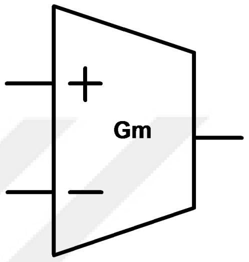
Equation 3.3 is for ideal cases only. In reality, the transfer function of the OTA is non-linear (figure 3.9a) with only a small linear region around the point where the differential input value is close to zero. In reality, equation 3.4 is more commonly used to show the transfer function of the OTA.

$$I_{Out} = I_b \tanh\left(\frac{k(\mathbf{V}_+ \; - \; \mathbf{V}_-}{2})\right) \tag{3.4}$$

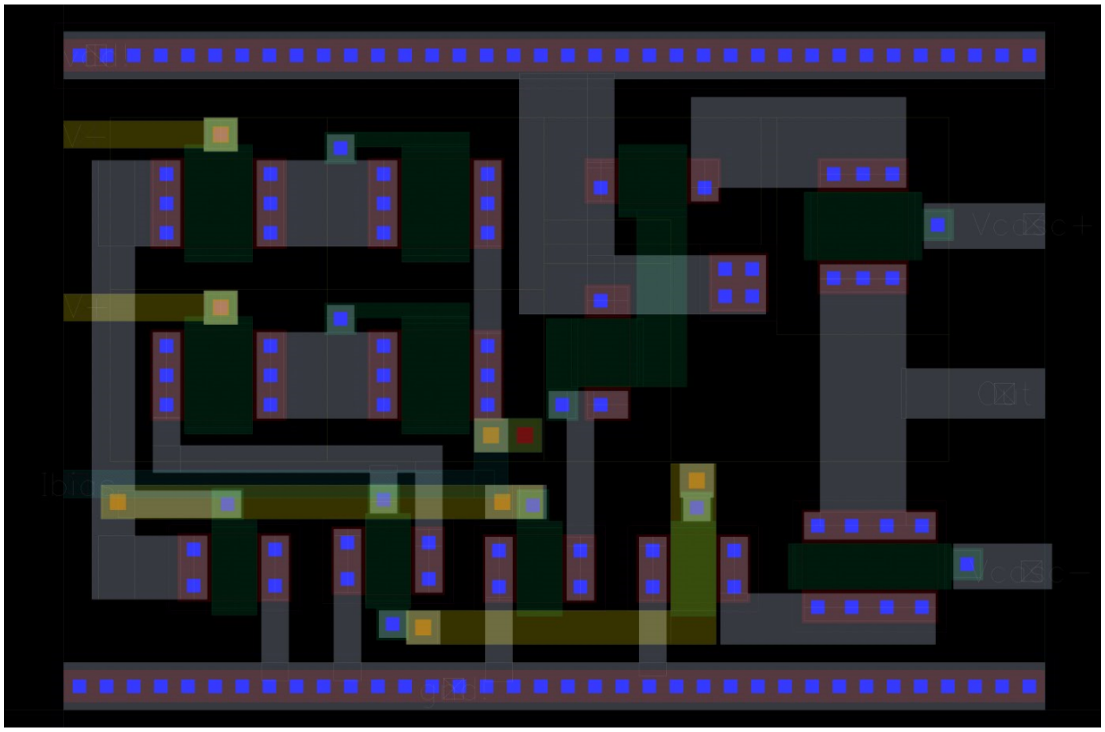
The value of the transconductance of the OTA itself depends on the magnitude of the bias current. The bias current/transconductance relationship is shown on figure 3.9b (for derivation, see appendix C.1).



(a) The OTA circuit.

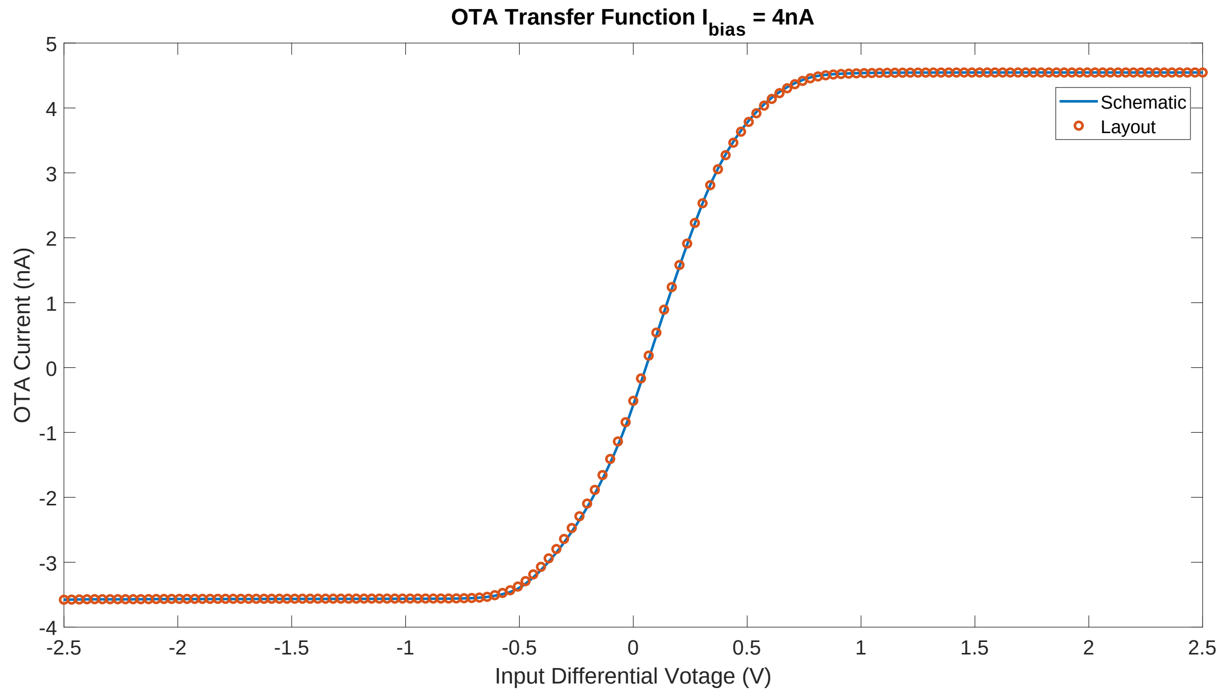


(b) The OTA symbol.

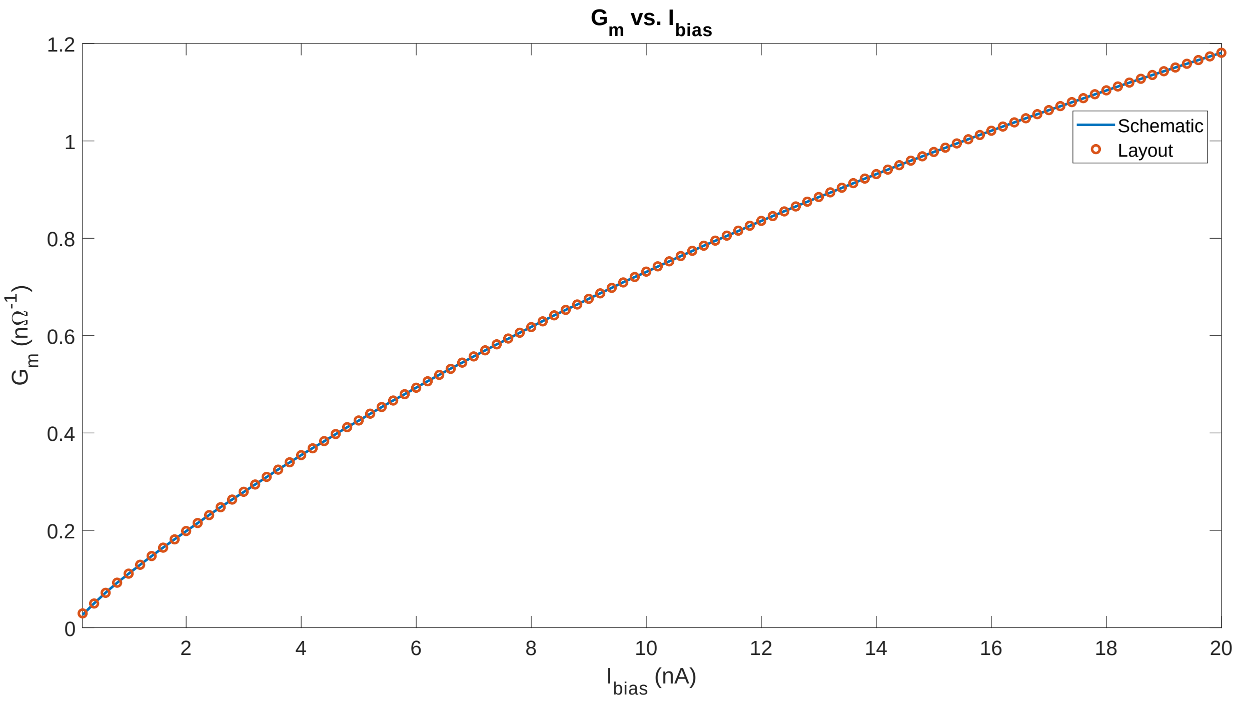


(c) The OTA Layout.

Figure 3.8: The operational transconductor amplifier circuit and symbol.



(a) The OTA transfer function. The linear region lies between the saturated current values, around the -0.5 - 0.5 Input Differential Voltage interval. For test circuit, see B.1.



(b) The gm-I_{bias} relationship.

Figure 3.9: The operational transconductor amplifier simulation results.

3.4.2.2 C^4 Bandpass Filter

The *Capacitively Coupled Current Conveyer* (C^4) bandpass filter is a 2nd order OTA based programmable bandpass filter (figure 3.10a)[11]. The transfer function of the filter is shown below, in equation 3.5 (for derivation, see appendix C.2). The programmability of the filter comes from the effect the transconductance value of each OTA has on the transfer function.

$$\begin{aligned} \frac{V_{out}}{V_{in}} &= \frac{\tau_b S \frac{\tau}{Q}(S\tau_a - 1)}{S^2\tau^2 + \frac{\tau}{Q} + 1} \\ \tau_a &= \frac{C_{fb}}{G_{m2}} \\ \tau_b &= \frac{C_{in}G_{m2}}{C_L G_{m1} + C_{fb}G_{m2}} \\ \tau &= \sqrt{\frac{C_{in}C_L + C_{fb}C_L C_{fb}C_{in}}{G_{m1}G_{m2}}} \\ Q &= \sqrt{\frac{(C_{in}C_L + C_{fb}C_L C_{fb}C_{in}G_{m1}G_{m2})}{(G_{m1}C_L + C_{fb}G_{m2})^2}} \end{aligned} \tag{3.5}$$

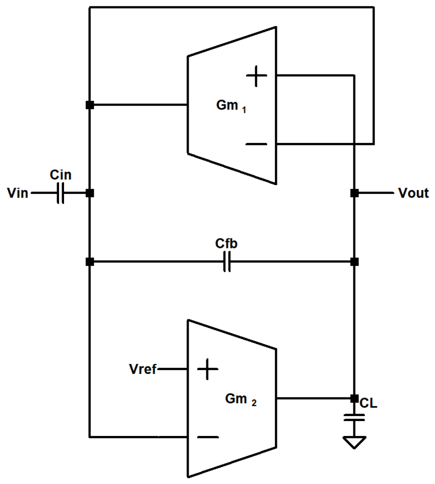
The center frequency of the filter can be found from the following equation:

$$f_c = \sqrt{\frac{G_{m1}G_{m2}}{C_{in}C_L + C_{fb}C_L C_{fb}C_{in}}} \tag{3.6}$$

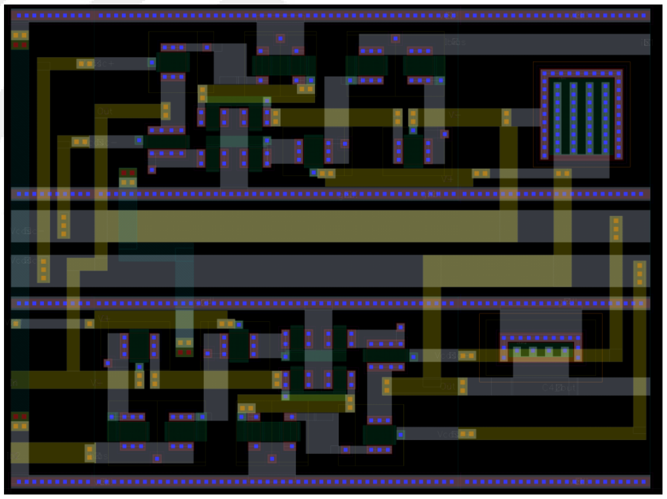
Solving equations 3.5 and 3.6 simultaneously according to the desired center frequency (f_c) and quality (Q) values. Figure 3.11 shows the variation of f_c and Q with respect to different bias currents on the schematic (figure 3.11a) and post layout (figure 3.11b) simulations.

Overall, the C^4 filter is used six times for the following tasks:

- **Feature extraction:** f_c = 0.55Hz, Q = 0.9,
- **Feature extraction:** f_c = 2.0Hz, Q = 2.0,
- **Feature extraction:** f_c = 6.5Hz, Q = 3.5,
- **Feature extraction:** f_c = 20Hz, Q = 20,
- **Feature extraction:** f_c = 40Hz, Q = 20,
- **Pan Tompkins first stage:** f_c = 10Hz, Q = 10.

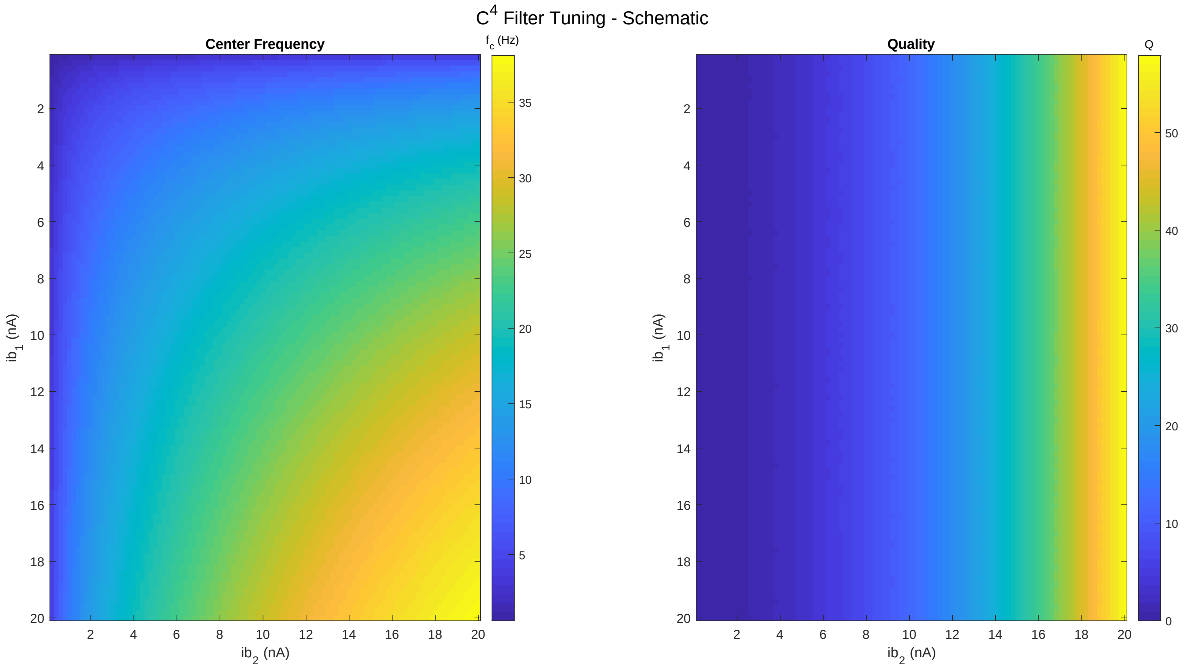


(a) Schematic of the C^4 filter.

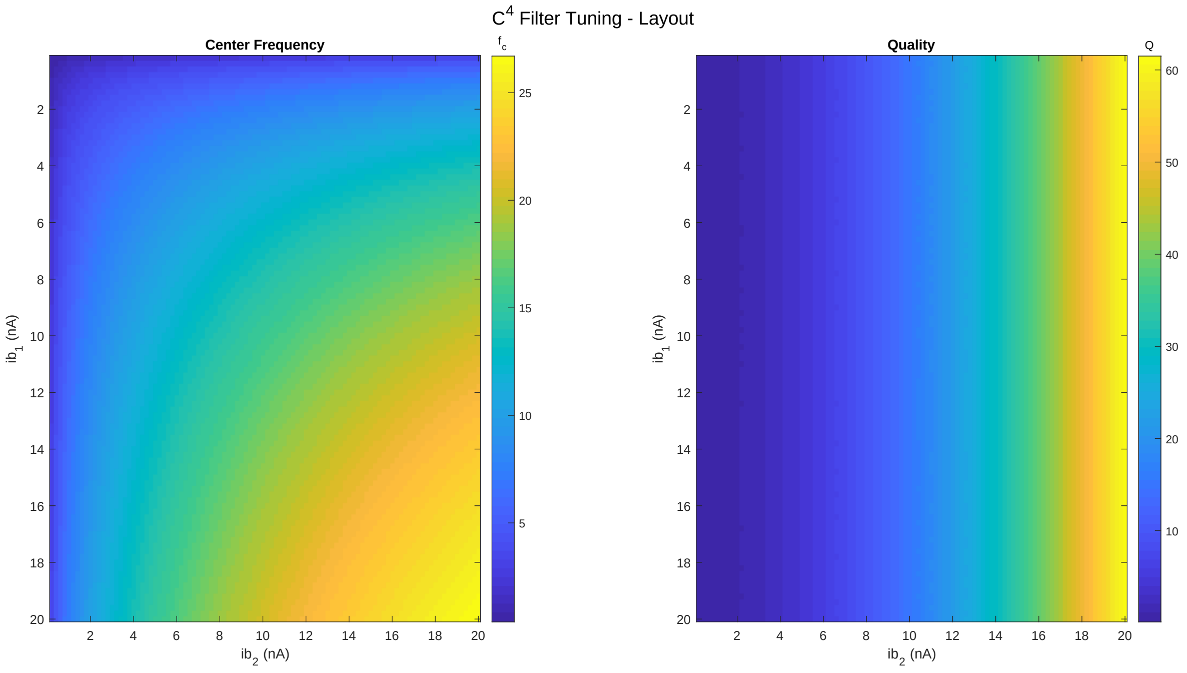


(b) Layout of the C^4 filter.

Figure 3.10: C^4 schematic and layout.



(a) Schematic simulation results for the center frequency and quality of the C^4 filter with respect to ib_1 and ib_2.

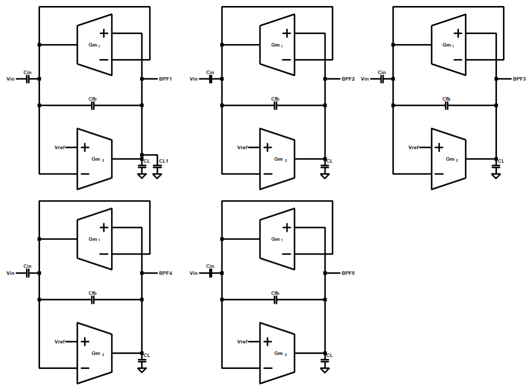


(b) Post layout simulation results for the center frequency and quality of the C^4 filter with respect to ib_1 and ib_2.

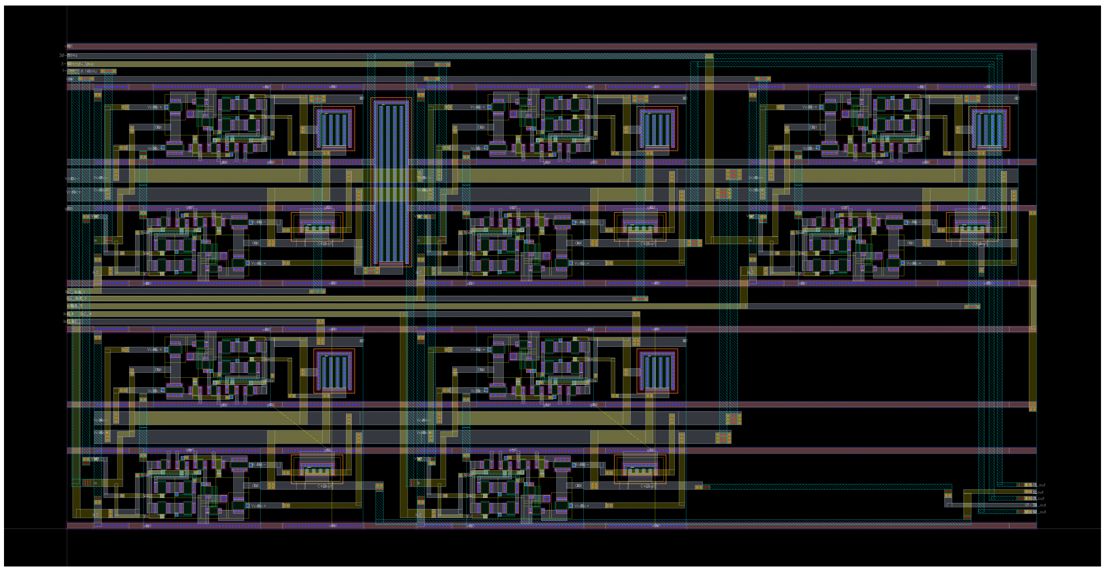
Figure 3.11: Center frequency and quality variation with respect to different bias currents.

3.4.2.3 The C^4 Filter Bank

As mentioned in section 3.2.1, the first five of the seven required features consist of the ECG input waveform filtered at different frequency intervals. In order to achieve this, five instances of the C^4 filter have been created, tuned as required, and implemented as a filter bank on Cadence. The schematic and layout of this filter bank can be seen below on figure 3.12.



(a) Schematic of the C^4 filter bank.



(b) Layout of the C^4 filter bank.

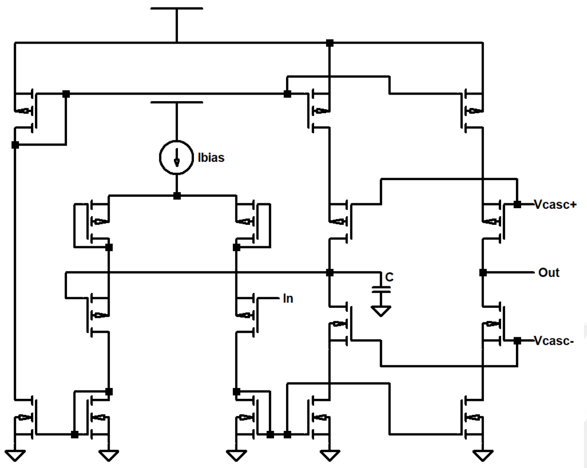
Figure 3.12: C^4 filter bank schematic and layout.

3.4.2.4 Differentiator

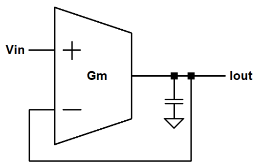
The next component on the chain of devices used to implement the Pan-Tompkins algorithm is the differentiator (figure 3.13. The base of the circuit is created the same as the operational transconductance amplifier mentioned in section 3.4.2.1. The input signal is given as a voltage to the positive (V+) input terminal of the OTA and the output is connected to the negative (V-) input terminal (as can be seen below in figure 3.13a). A capacitor connected to the output of the OTA creates the differential output of the input signal. This is caused by the differential voltage-current relationship of the capacitor [12].

In order to convert the differential output voltage to a current, as well as prevent the alteration of the differential effect of the capacitor by introducing other devices, the output of the OTA is replicated by connecting 4 identical transistors to the output. The gates of these transistors is connected to the gates of their respective counterparts in order to replicate the same currents flowing through them. The output of these transistors can then be given as a current to the next device linked on the Pan-Tomkins chain.

The input output relationship of the differentiator for an input triangle wave is plotted in figure 3.14. As can be seen from the figure, the output (for both schematic and layout implementations on Cadence) is a positive square wave (corresponding to the rising edge of the triangle) followed by a negative square wave (corresponding to the falling edge of the triangle). The output square waves of the differentiator not ideal (as the corners are not sharp) but the circuit still works as intended. It should also be noted that the AV extracted results show some instability in the form of minor oscillations. However, this didn't cause any problems in the Pan-Tompkins circuit.



(a) The differentiator circuit.



(b) The differentiator symbol.

(c) The differentiator Layout.

Figure 3.13: The differentiator.

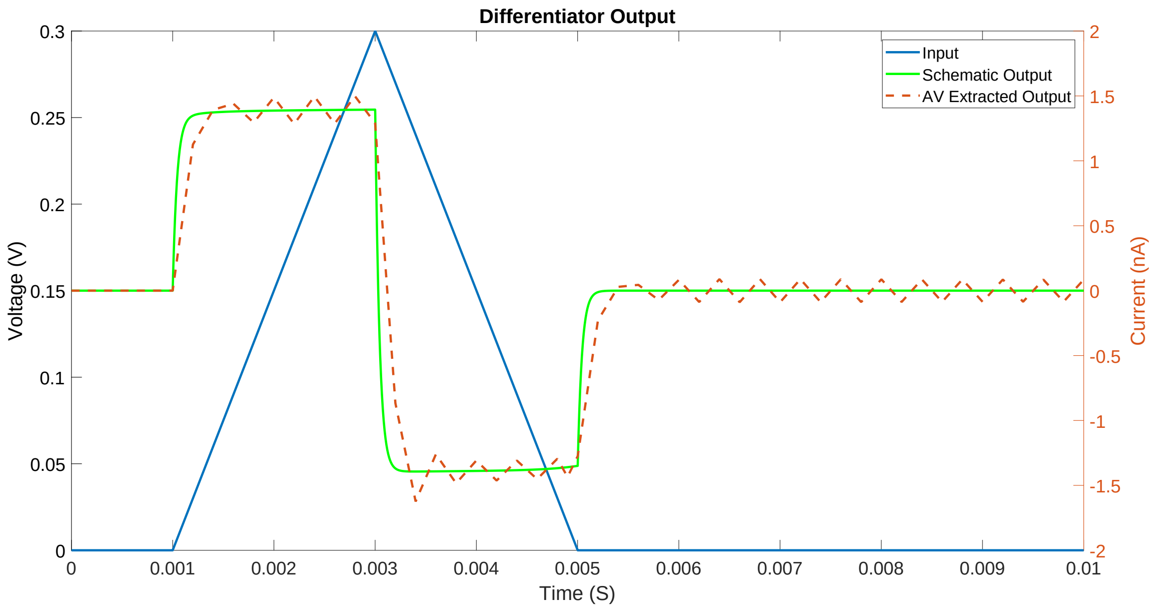
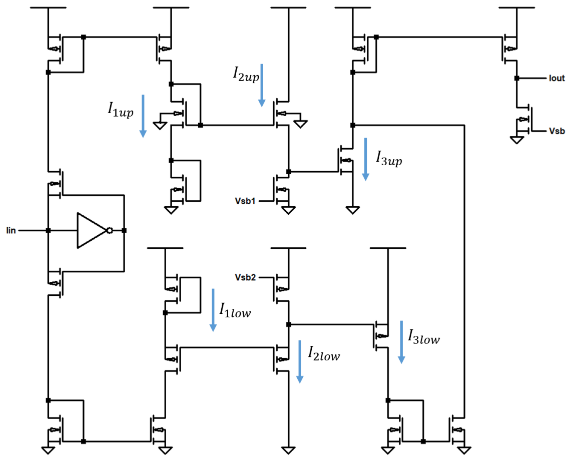


Figure 3.14: The differentiators response to a triangle input. The test circuit could be found in appendix B.2

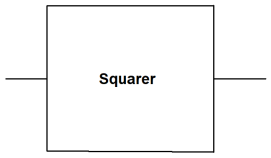
3.4.2.5 Current Squarer

The next step of the Pan-Tomkins algorithm is squaring. The circuit chosen to achieve this is shown below in figure 3.15. It is a two-stage circuit (based on [13] and [12]) that takes the square of the negative and positive parts of the input separately (similar to a class AB amplifier). Depending on the nature of the input current (whether it flows into the circuit's input terminal (positive) or out of the input terminal (negative)) the inverter will either output *Vdd* or *gnd* which will switch between the upper and lower parts respectively. The outputs of both stages are then combined

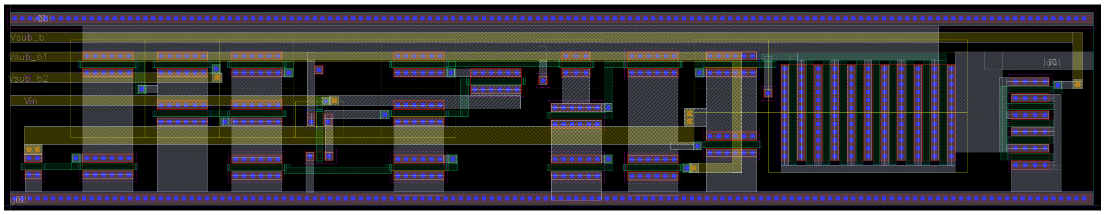
Both upper and lower stages use a process known as *translinear squaring [13]* for the squaring procedure. Circuit simulations can be seen below on figure 3.16.



(a) The squarer circuit.



(b) The squarer symbol.



(c) The squarer Layout.

Figure 3.15: The squarer circuit.

The relationship between the I_1, I_2, and I_3 currents for both upper and lower sections of the circuit is expressed with equation 3.7 given below. For the derivation of this equation, refer to appendix C.3.

$$I_3 = \frac{(I_1)^2}{I_2} \tag{3.7}$$

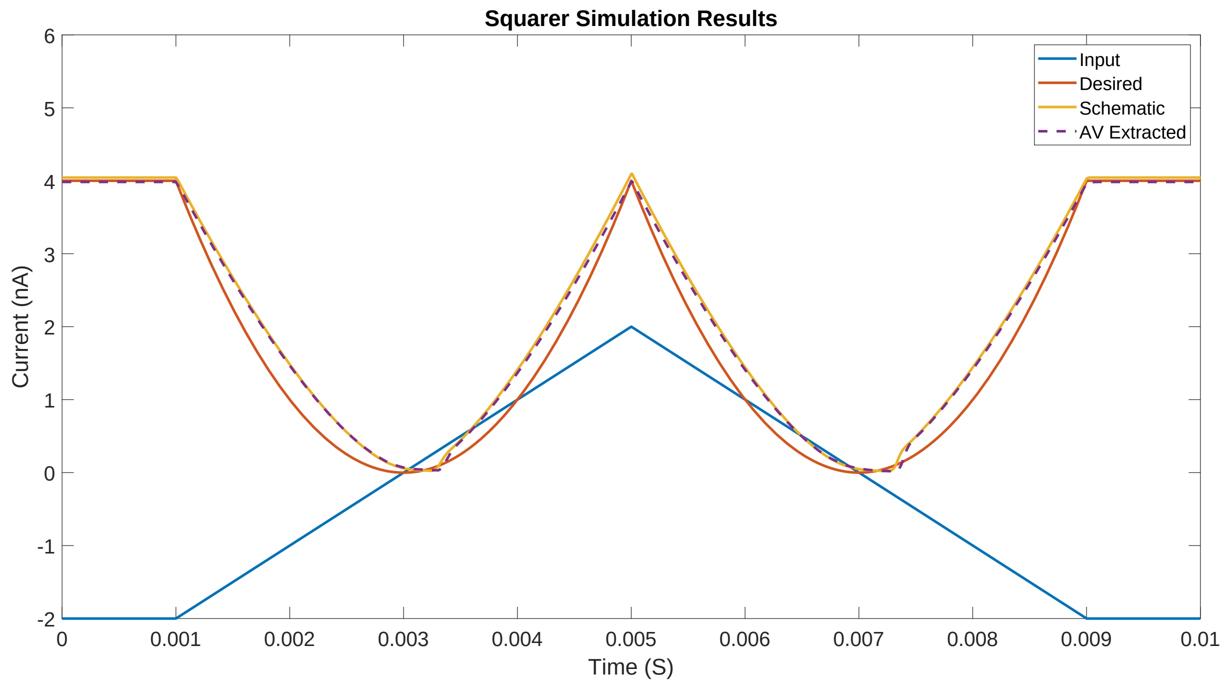


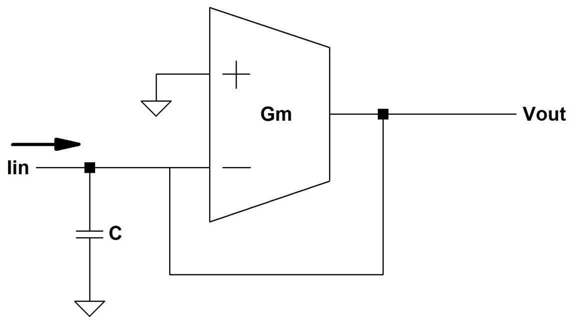
Figure 3.16: Simulation results for both schematic and post layout of the squarer circuit. The test circuit could be found in appendix B.3

In the simulation, an 8 ms triangle wave with a peak-to-peak current of 4 nA (-2 nA to +2 nA) is given to the circuit as an input. Both the schematic and AV extracted results are identical with each other. Magnitude-wise, the circuit parameters extracted from the layout (post-layout parameters i.e. *AV extracted*) results are a little less than the schematic results. When compared to the desired signal, both the schematic and AV Extracted show deviations with errors upto 10%. These errors are greatest when the input currents absolute magnitude is less than 1 nA. Another issue, close to 0 nA, the shape gets distorted. These distortions will not be too problematic as the classifier can be trained to classify preprocessed inputs with the effects of these distortions.

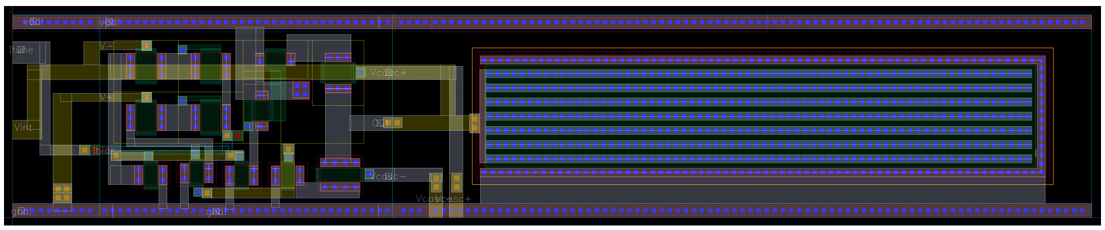
3.4.2.6 The Leaky Integrator

The final component of the Pan-Tompkins circuit is the leaky integrator, which is an integrator that gradually loses the value (charge) it holds over a fixed rate. The circuit consists of a capacitor, whose differential current-voltage relationship is used to obtain the integral of the input current, and an OTA based pseudo-resistor. In order to create the pseudo-resistor, the positive terminal of the OTA is grounded and the negative terminal (also the input of the circuit) is connected to the output.

The integrator receives the output current of the squarer (figure 3.15) as its input, converts it to a voltage via the capacitor. The OTA then outputs the integral of the input signal. The transfer function of the integrator is given below (equation 3.8) The full derivation of this equation can be found in the appendix (C.4).



(a) The leaky integrator schematic.



(b) The leaky integrator Layout.

Figure 3.17: The leaky integrator circuit, schematic and layout.

$$\frac{V_{out}}{I_{in}} = \frac{1}{1 + \frac{sC}{G_m}} \tag{3.8}$$

As Pan-Tompkins algorithm requires a leaky integrator [9], the circuit was tuned to have a leakage time of around 80 ms. Simulation results for both schematic and layout extracted circuits are shown below on figure 3.18.

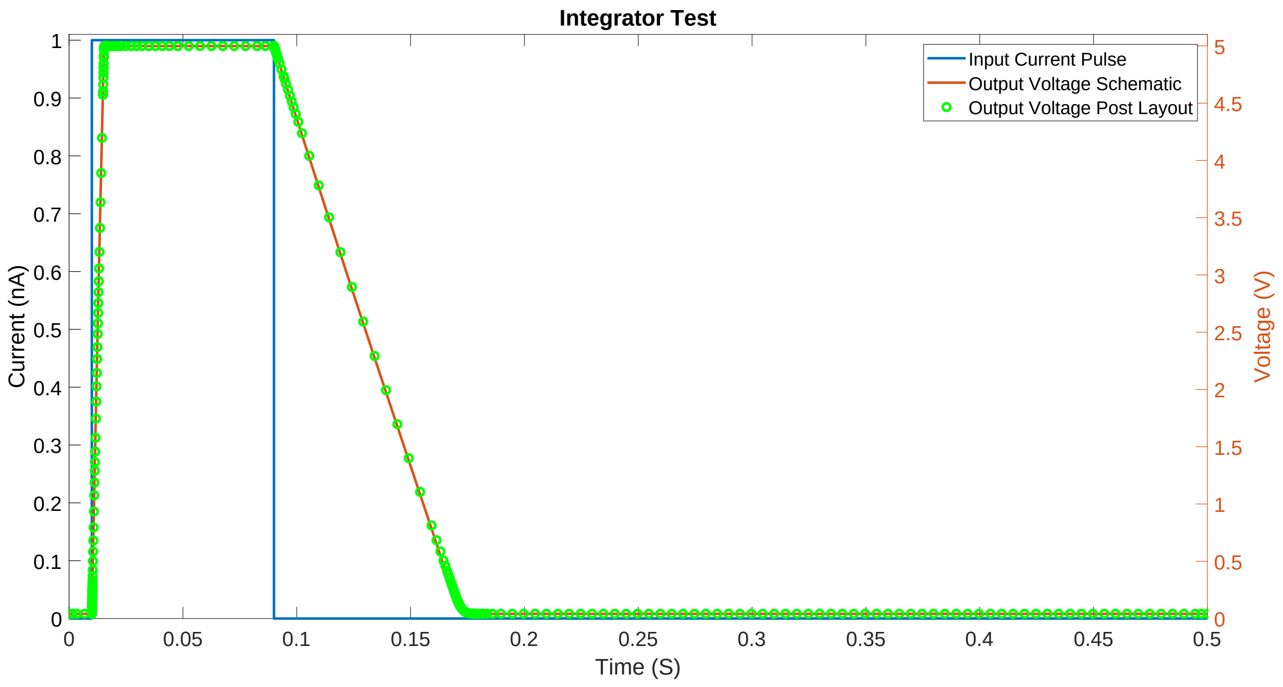


Figure 3.18: Simulation results for both schematic and post layout of the leaky integrator circuit. The test setup schematic used to obtain the results of this simulation can be found in appendix B.4.

In addition to the leaky integrator shown above, a second type of leaky integrator was also created. Shown below in figure 3.19, this integrator was created by adding a second OTA to the current input of the integrator in figure 3.17a. This OTA converts its input voltage into a current which is then integrated by the circuit discussed previously. This leaky integrator is used to integrate voltages. This integrator is used at the output of the classifier as a low-pass filter, the intended (algorithmic) impact of this component on the output of the classifier is shown in figure 3.7.

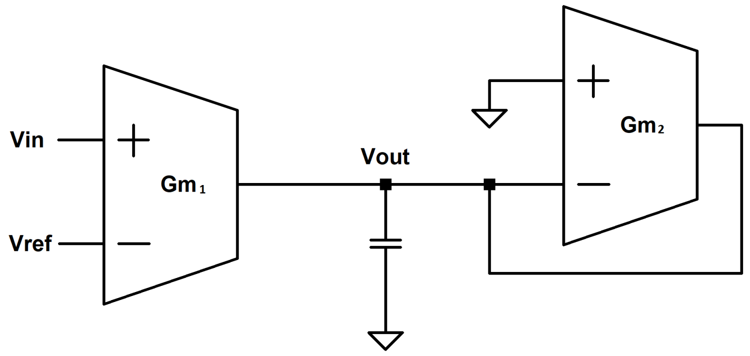


Figure 3.19: The leaky voltage integrator implemented with a pseudo-resistance mode OTA.

The transfer function of this leaky integrator is given below in equation 3.9. The derivation of this equation can also be found in appendix C.4. Simulation results of this integrator at different leakage rates is given below in figure 3.20.

$$\frac{V_{out}}{V_{in}} = \frac{G_{m1}}{1 + \frac{sC}{G_{m2}}} \tag{3.9}$$

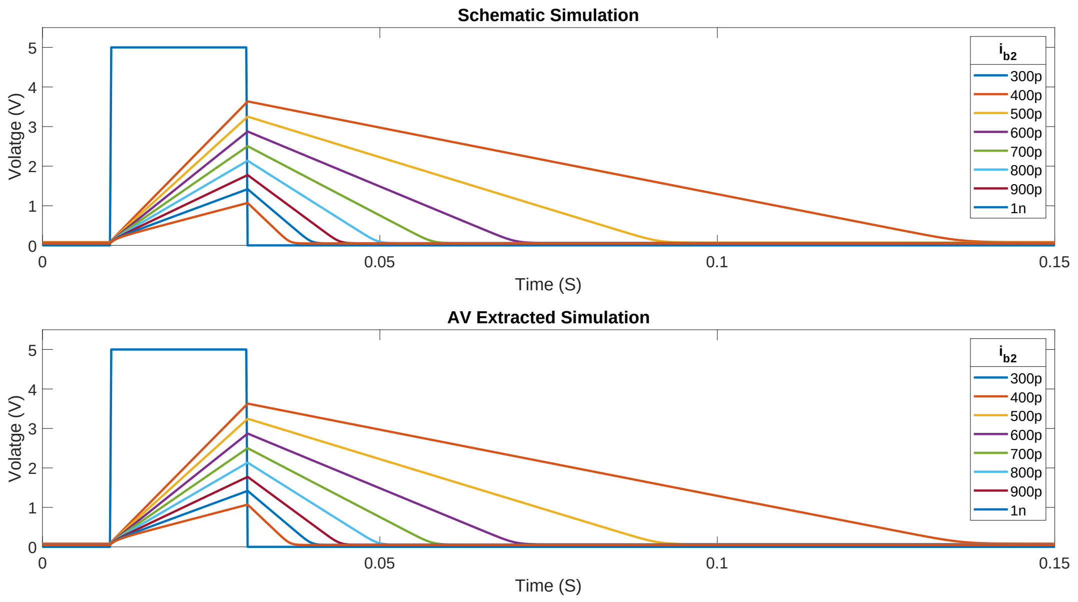
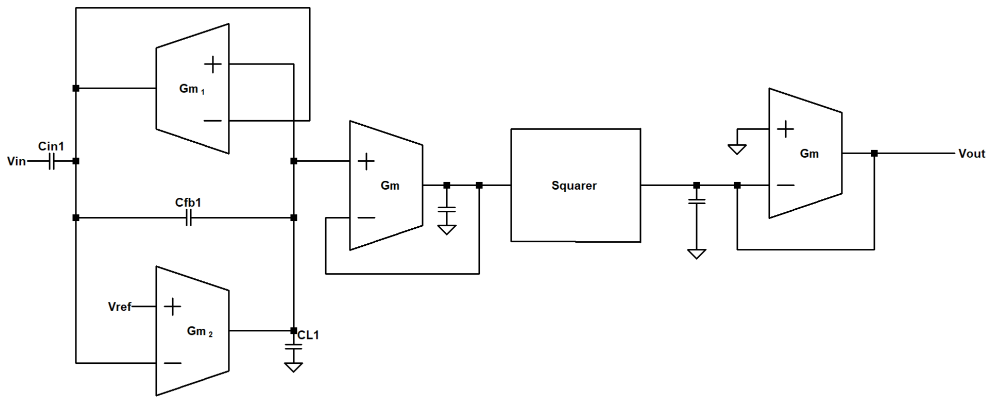


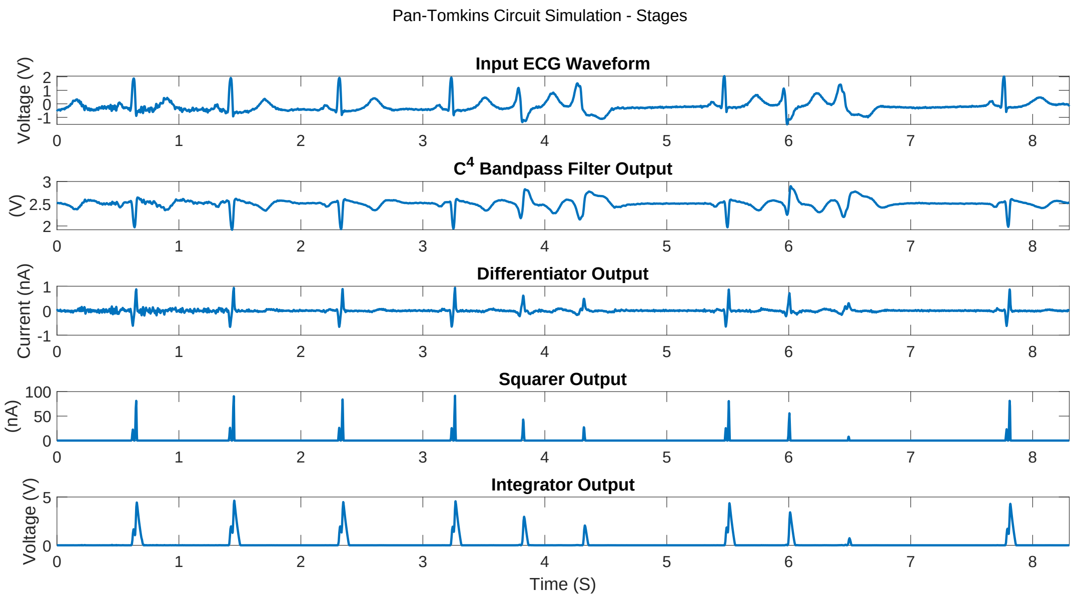
Figure 3.20: The leaky voltage integrator simulation results.

3.4.3 The Pan-Tompkins Circuit

When the devices shown in the previous subsections (3.4.2.2, 3.4.2.4, 3.4.2.5 and 3.4.2.6) are chained together in the order they appear here (shown below in figure 3.21a), they implement Pan-Tomkins algorithm in the time domain [12]. Figure 3.21 shows the output of each circuit. As can be seen from the results, the circuit stages output similar waveforms to those seen in figure 3.5.



(a) The Pan-Tompkins circuit.



(b) Cadence schematic simulation results for the Pan-Tompkins circuit.

Figure 3.21: The Pan Tompkins Circuit and Simulation results.

However, there are also some differences especially evident in the differentiation and integration stages. These mainly arise from the difference between the analog and digital means used to implement those methods. It could be argued that as digital differentiation and integration is less precise than their analog counterparts, the circuit outputs are closer to the desired ideal forms.

3.4.3.1 Peak Detector

The role of the peak detector circuit is to find the maximum level of the output of each integrator peak and hold onto that value until the next peak arrives. This is achieved using two identical *OTA-C* (operational transconductance amplifier - capacitor) circuits. A single instance of the OTA-C peak detector is shown in figure 3.22.

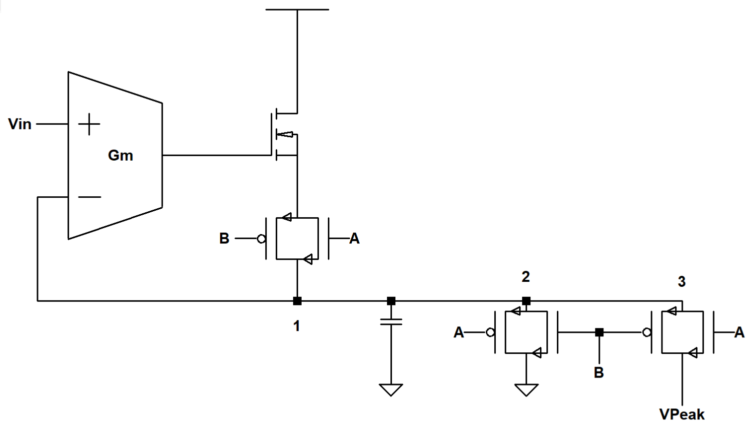
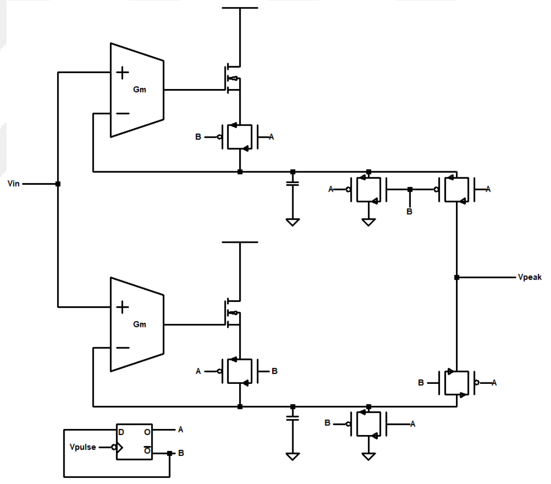


Figure 3.22: A single instance of the peak detector circuit.

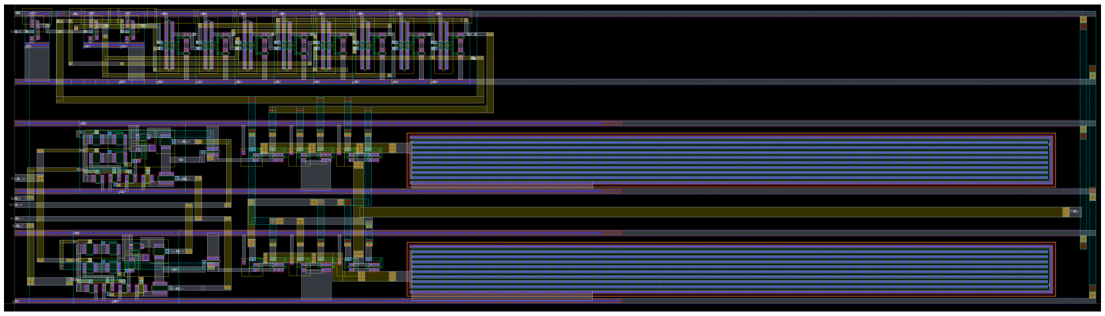
The aim of this circuit is to capture and hold onto the max value of the Pan-Tompkins peaks. Hence, unlike a traditional peak detector, its output should not decrease over time. In order to achieve this, a series of transmission gates (labelled as 1,2, and 3). The transmission gates 1 and 3 are connected to the same complementary control signals A and B ($B = \overline{A}$). Transmission gate 2 is connected to the inversion of these control signals. gates

When gates 1 and 3 are active, and the input voltage of the OTA is greater than the voltage stored on the capacitor, the OTA starts charging the capacitor via the NMOS connected to its output. Gate 3 then allows the capacitor voltage to to pass be outputted. When the complimentary control signal is given, gates 1 and 3 turn off (reach a very high resistance state) and gate 2 switches on. As gate 2 directly connects the capacitor to the ground, it being on discharges the capacitor, bringing its charge to 0 and resetting the circuit for a new peak.

In order to create the peak value detection circuit, the circuit was replicated and the orientation of the transmission gates was switched so that the replicated circuit would have the control signals connected to the dual terminals (illustrated in figure 3.23a). This enables only one of the two circuits to be switched on at a time, allowing the other one to discharge. In other words, the two individual detectors are multiplexed to the same output. The control signals are generated using the normal and inverted outputs of a 1-bit digital counter. The counter circuit is created using a D flip-flop with its inverted output connected back to its D input. The clock of the D flip-flop receives its inputs from the digital pulses generated by the RR Distance Finder circuit (*Vpulse* in 3.26a).



(a) The peak detector circuit, complete schematic.



(b) The peak detector circuit layout.

Figure 3.23: The peak detector schematic and layout.

When the layout of the peak detector (figure 3.23b) is examined closer, an unused region can be seen in the top right third of the layout. This section was intentionally left blank in order to fit the RR distance finder circuit (figure 3.26b) and shorter the connections between its pulse signal to the clock of the D flip-flop (top left third of the peak detector layout). The final version can be seen in figure 3.27b.

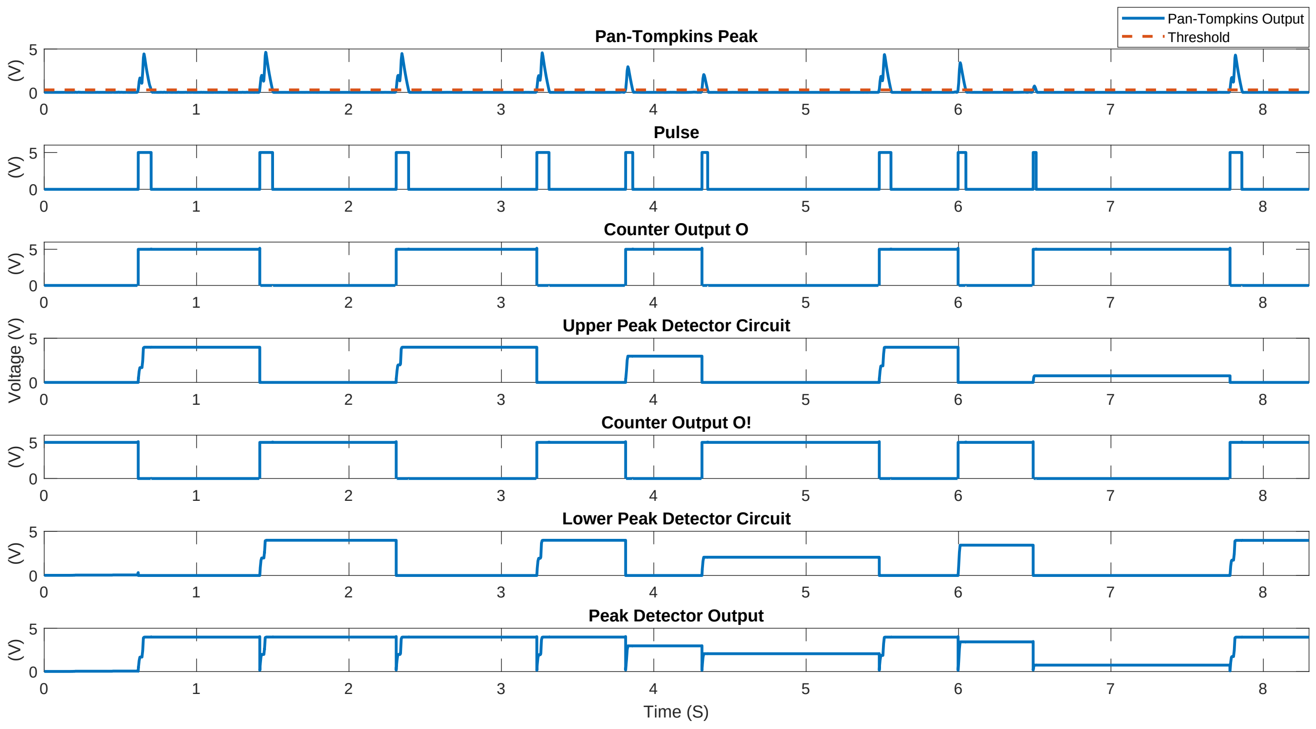


Figure 3.24: Waveforms of the peak detector circuit.

Figure 3.24 above shows the waveforms for different instances of the Peak detector with the output of the Pan-Tompkins circuit (figure 3.21b) fed as an input. The uppermost plot shows the Pan-Tompkins peaks along with the threshold value that generates the pulses. The second plot (*Pulse*) shows the corresponding pulses that arise when the Pan-Tompkins signal is above the threshold. The third and fifth plots (*Counter Output O* and *Counter Output O!* respectively) shows the two outputs of the counter. The fourth and sixth plots (*Upper Peak Detector Circuit* and *Lower Peak Detector Circuit* respectively) show the voltages of the two peak detector. As can be seen from these plots, the peak detectors show activity when their corresponding counter output is high. The final plot (*Peak Detector Output*) shows the overall output of the circuit which multiplexes the outputs of the upper and lower circuits based on their control signals.

3.4.3.2 RR Distance Finder

The distance between two R peaks (*RR distance*) is the final feature required by the classification algorithm. As the important aspect here is the time delay between the two peaks (figure 3.4), the shape of the waveform itself, as well as its peak value, is not important. The time delay between the two consecutive R peaks is shown as a voltage that decays from a thresholded digital pulse. Figure 3.25 below illustrates the operation described above.

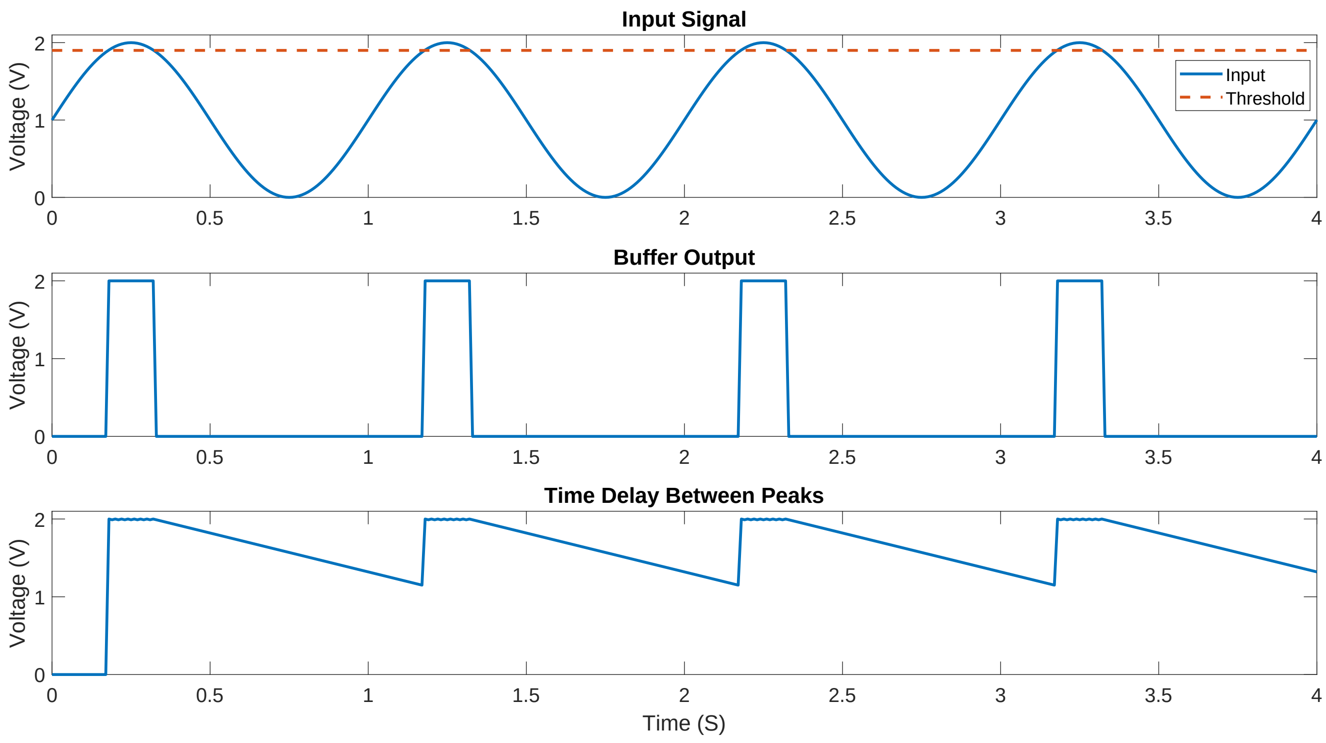
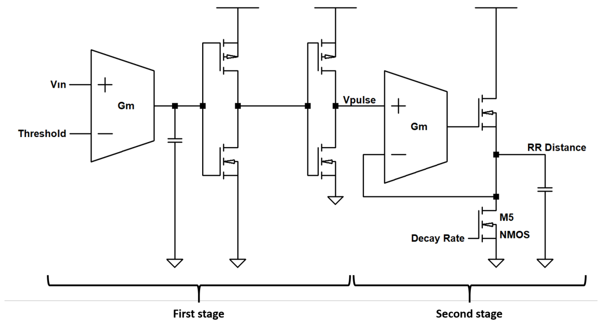


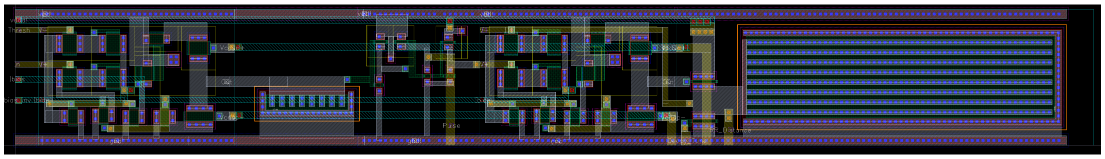
Figure 3.25: A MatLab based example to illustrate the order of operations conducted by the RR distance circuit: As the input signal crosses a certain threshold value (top plot), a digital pulse is generated (middle plot). The duration of the pulse lasts as long as the input is above the threshold. Once the pulse is gone, the value of the pulse starts to decay. The greater the decay, the greater the time delay between the two pulses.

The circuit designed to find the distance between the two R waves consists of two stages. In the first stage, thresholding occurs via an OTA-C circuit connected to a digital voltage buffer. The OTA is tuned to detect a threshold voltage of 100 mV, once detected, it quickly fills a capacitor whose rise in voltage then triggers the buffer. An OTA-C circuit was chosen for this task as the OTA's low dynamic range allows it to almost act like a digital switch. Also, it should be noted that thresholded inverters (where the W/L ratio between the PMOS and the NMOS transistors is adjusted to enable thresholding) do not reach the target sensitivity without taking up a huge amount of area.

The pulse generated by the buffer is then fed into the second stage which consists of a conventional OTA based peak detector. Here, the output of the OTA drives the amount of current flowing through the upper NMOS transistor. This transistor fills up the capacitor which stores the voltage representing the time delay between the two input peaks. Simulation results of this circuit on the sample ECG sequence is given in figure 3.28b.



(a) The RR Distance Finder schematic.

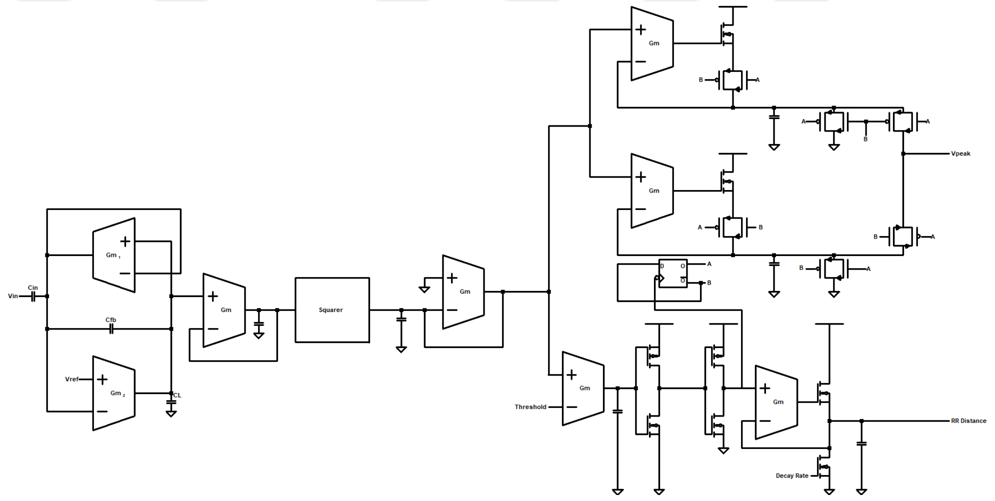


(b) The RR Distance Finder Layout.

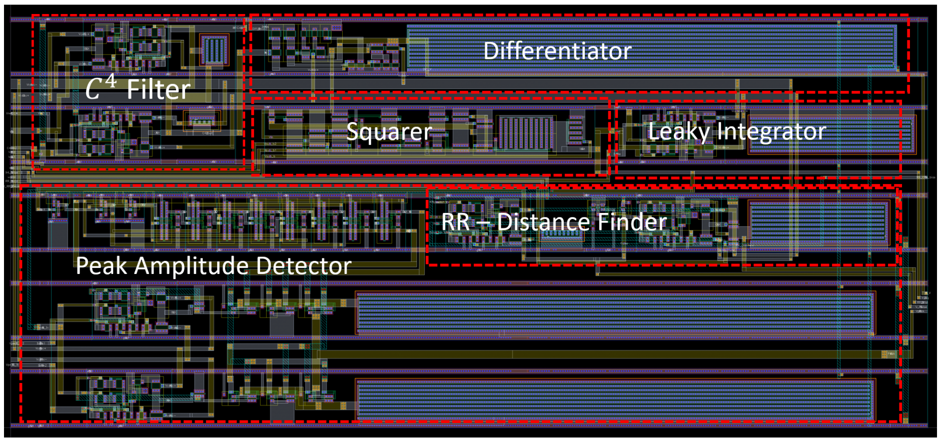
Figure 3.26: The RR Distance Finder.

3.4.3.3 Feature Extraction Stage

Figure 3.27a shows the placement of each component on the layout of the Pan-Tompkins + Peak Detector and RR Distance Finder circuits. Figure 3.27b shows the overall layout. The components were placed and routed in a fashion that would be the most area efficient. Simulation results (schematic and post layout) of this circuit are shown in figure 3.28 below.

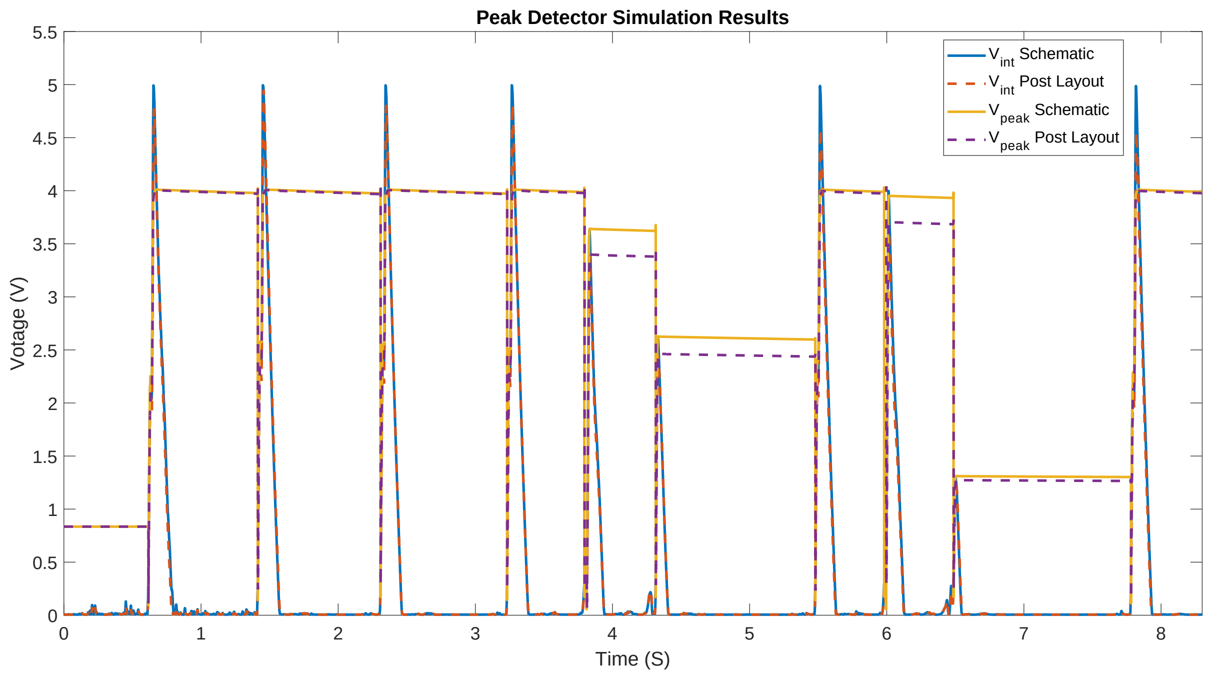


(a) The combination of each of the above components for the preprocessing circuit.

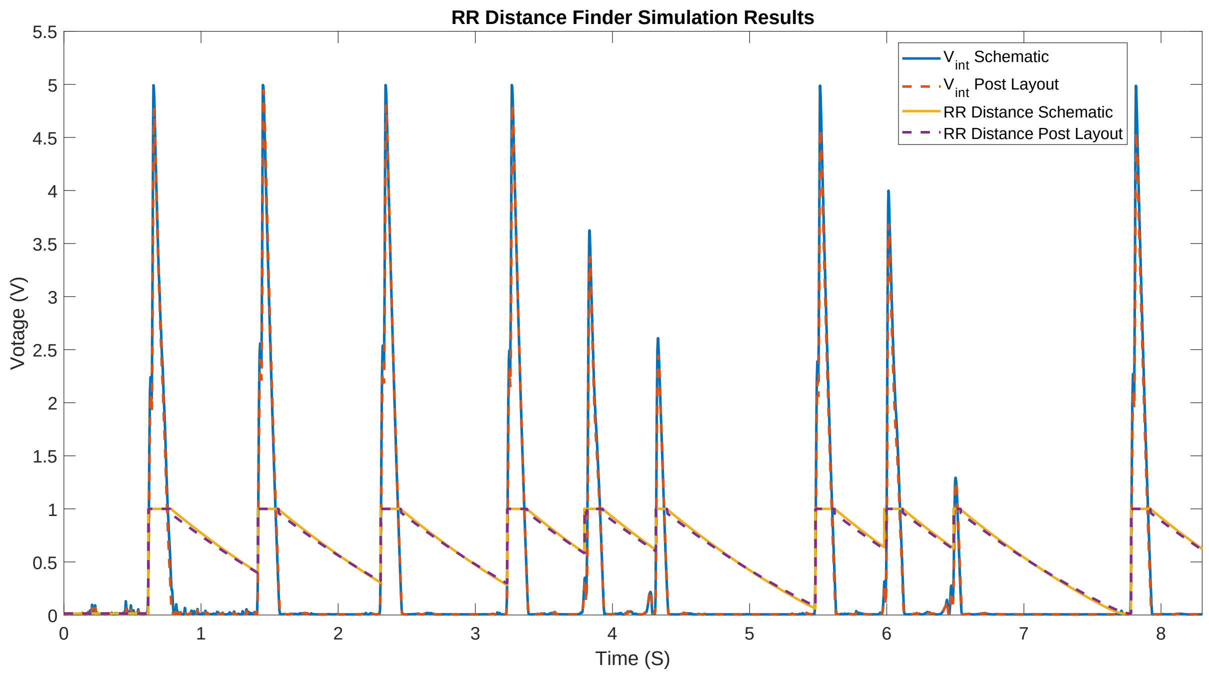


(b) The complete layout of the preprocessing circuit with components labelled.

Figure 3.27: The preprocessing circuit.



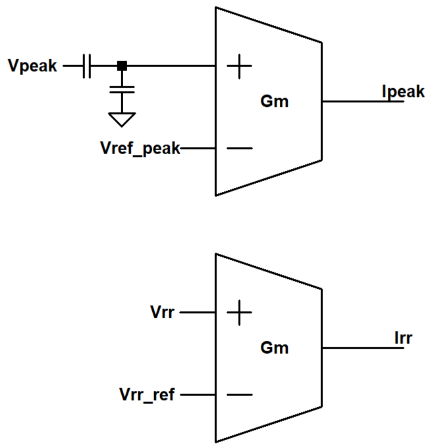
(a) The peak detector circuit simulation results.



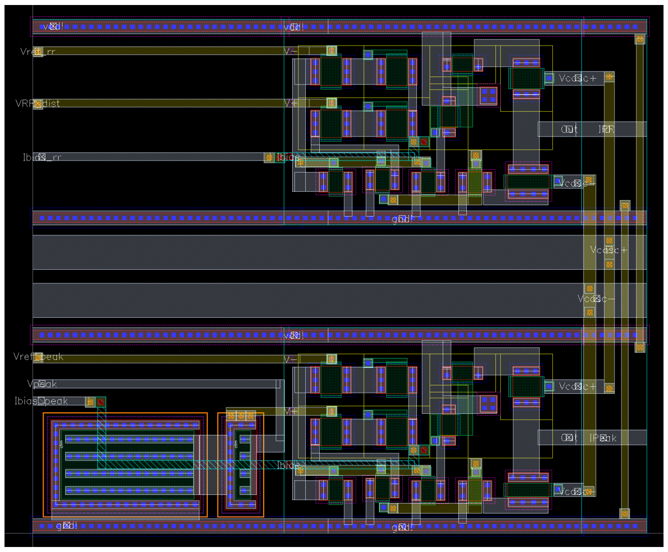
(b) The RR Distance Finder simulation results.

Figure 3.28: The feature extraction circuit simulation results.

As will be explained in the next subsection, the classifier receives currents as input whereas feature extractor outputs voltages. Hence, the following converter circuits (figure 3.29) are added to the output of the peak detector and the RR distance finder in order to convert their outputs into a current. For the case of the peak converter, the a capacitive voltage divider is used so that the peak values fit into the linear region of the OTAs transfer function (figure 3.9a).



(a) The converter circuit schematic.



(b) The converter circuit layout.

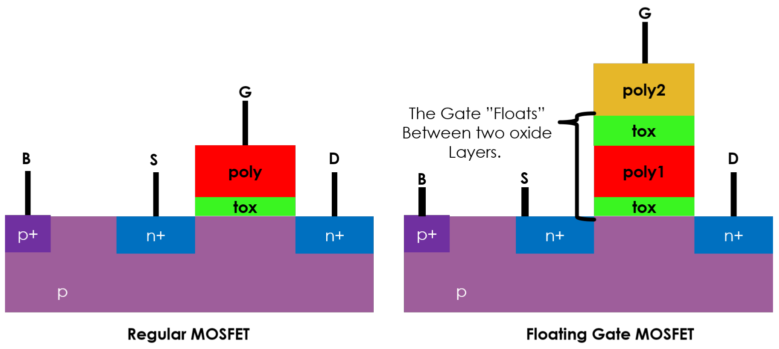
Figure 3.29: The converter circuit for the Pan-Tompkins peak and RR distance values.

3.4.3.4 Classifier

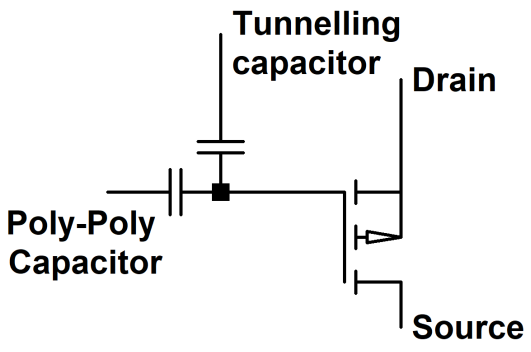
Moving onto the classifier stage of the algorithm, the positive weighted perceptron (see 3.2.5) is implemented in two stages. The first stage implements the weight multiplication and summing using a FGMOS based vector-matrix-multiplier. The second stage utilizes a winner-take-all current comparator to emulate the activation function of the perceptron.

3.4.3.5 Analog FGMOS Transistors for Weight Storage

The *floating gate metal oxide semiconductor transistor* (FGMOS) is a type of non-volatile memory storage device. Its main difference from a regular CMOS transistor is its gate, which consists of 2 poly layers with the first one sandwiched between two oxide layers, isolating it from the surroundings (figure 3.30a). The other poly layer is at the top, above the oxide. The high electrical resistance of the oxide traps the charge stored at the gate. In total, 20 FGMOS's are used in this work (the *vector matrix multiplier*, section 3.4.4).



(a) Cross-section comparisons of a regular MOSFET (left) and an FGMOSFET (right). The poly layer above the gate in the FGMOSFET (poly1) is sandwiched between two oxides, trapping the electrical charge within.

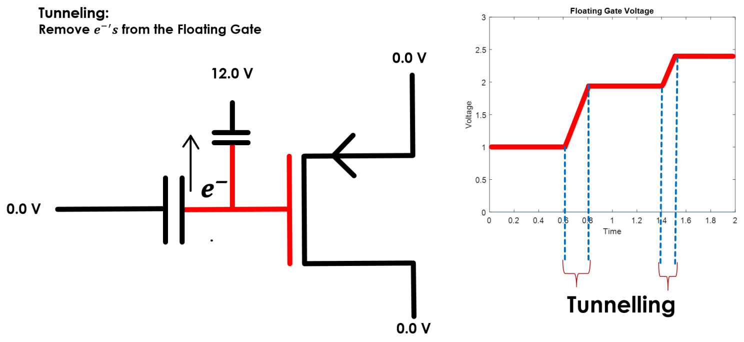


(b) The pins of an FGMOSFET. The Poly-poly capacitor is used to increase the total amount of possible storable charge by increasing the amount of poly1.

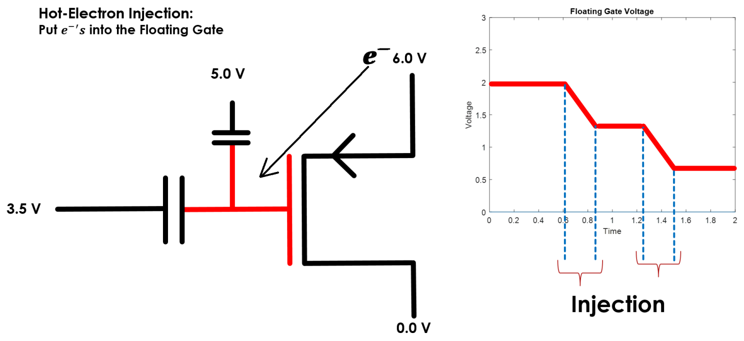
Figure 3.30: FGMOSFET transistors.

The trapped charge remains in the first poly layer and unless altered, will remain constant for a long time (on the order of years). Two different processes that can be used to alter the charge trapped in the poly layer are:

- **Tunnelling:** (figure 3.31a) A high voltage is used to extract electrons from the poly layer. The removal of electrons (which tunnel through the oxide) increases the net positive charge of the poly layer, increasing the voltage at the gate.
- **Hot electron injection:** (figure 3.31b) Electrons are injected into the poly layer which increases the net negative charge, decreasing the voltage at the gate.



(a) During **tunnelling**, the application of a high voltage through the tunnelling transistor extracts electrons from the *floating* poly layer, increasing the gate voltage.



(b) During **hot electron injection**, the application of a high voltage on the drain of the transistor injects electrons into the *floating* poly layer, increasing the gate voltage.

Figure 3.31: Different FGMOS programming modes. The voltages are applied as pulses with 0.2s durations, leading to the changes in the plots.

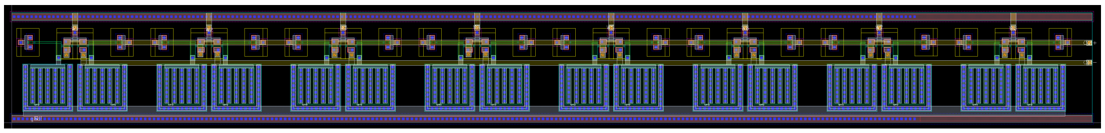


Figure 3.32: Layout of the FGMOSFETs used for weight storage in the VMM.

The voltages required for different stages of programming and operation are supplied externally. In order to decrease the number of required pads, a 4-pin analog multiplexer was created. The multiplexer uses transmission gates connected to different logic signals to multiplex between various states. The same logic signals are used to control transmission gates at all of the terminals. The complete layout of the FGMOS's is shown below in figure 3.33.

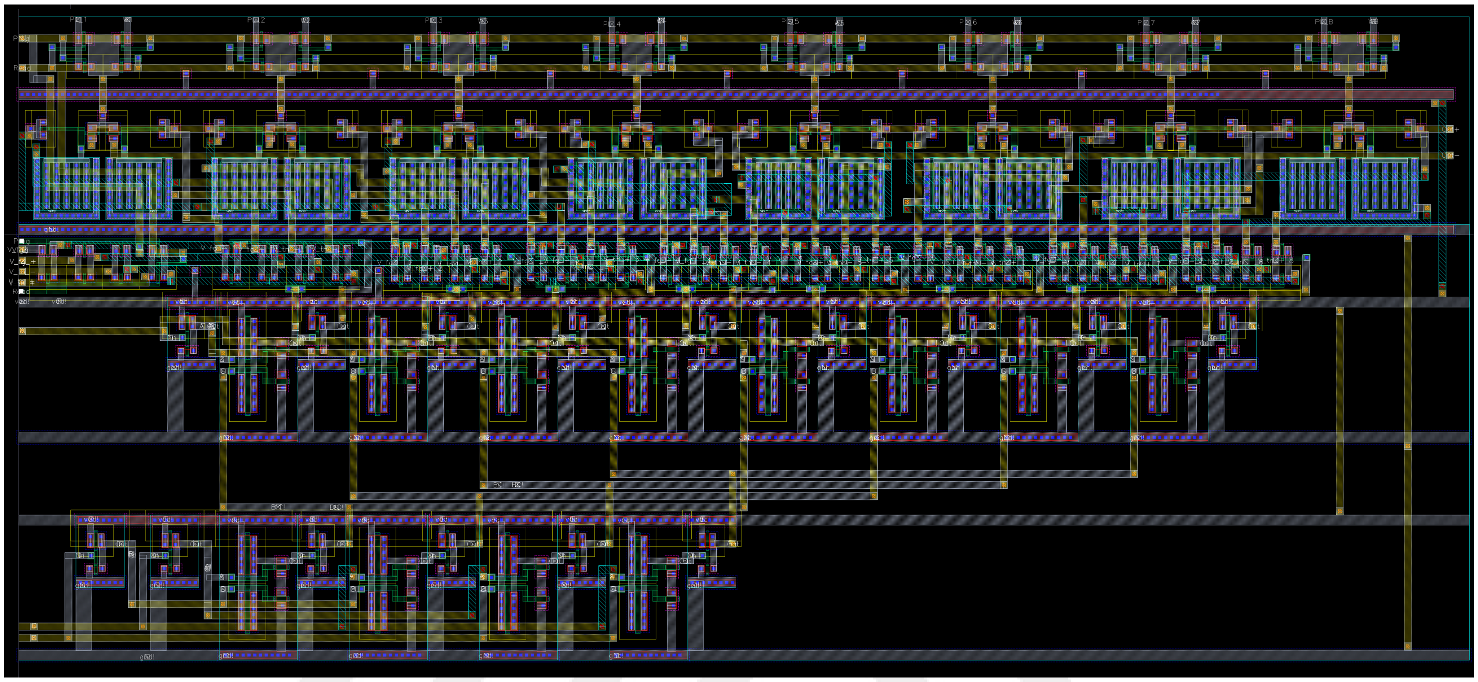
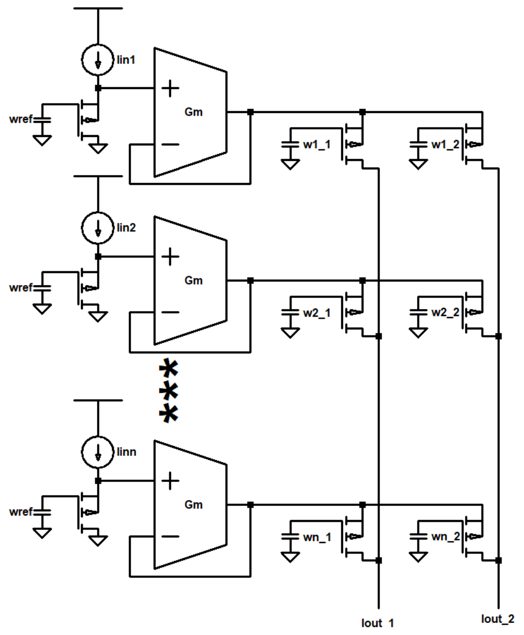


Figure 3.33: Layout of the FGMOSFETs with the analog multiplexer circuit.

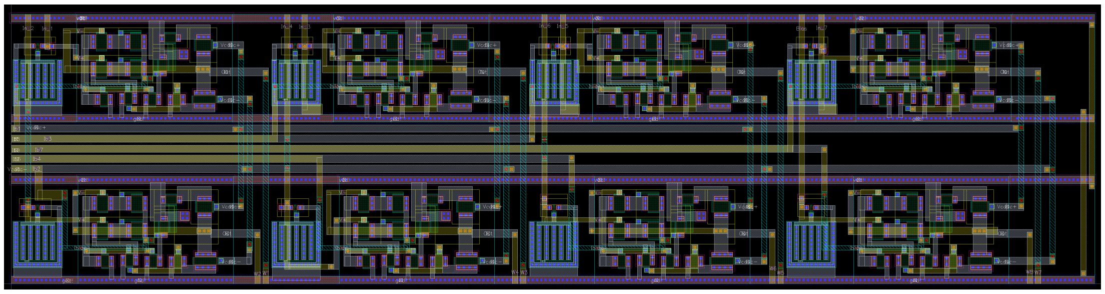
The control signals for the multiplexer consist of three select bits (each corresponding to a different row) and one enable bit.

3.4.4 The Vector Matrix Multiplier

The vector matrix multiplier (VMM) circuit is an analog circuit used to implement the multiplication of a vector with a matrix. As can be seen on figure 3.34a below.



(a) The VMM circuit schematic.



(b) The VMM circuit layout.

Figure 3.34: The vector matrix multiplier.

Each row of the circuit correlates to a row in the matrix. On the left-hand-side, the input is fed through the reference FGMOS with charge *wref* loaded at the gate. The input (value of the vector that will be multiplied with the row of the matrix) is given as a current through the source of the reference FGMOS. The voltage at the source of the reference FGMOS is then replicated to the sources of the FGMOS'on the right hand side via the voltage follower OTA. The ratio of the charge on the right-hand-side FGMOS's with respect to the reference charge, determines the magnitude of the reference current that will flow through that transistor. In other words, the current that flows through each right-hand-side FGMOS is equivalent to the reference current multiplied by the ratio between of the charges on the right-hand-side transistors and the reference charge. Mathematically, this is expressed below:

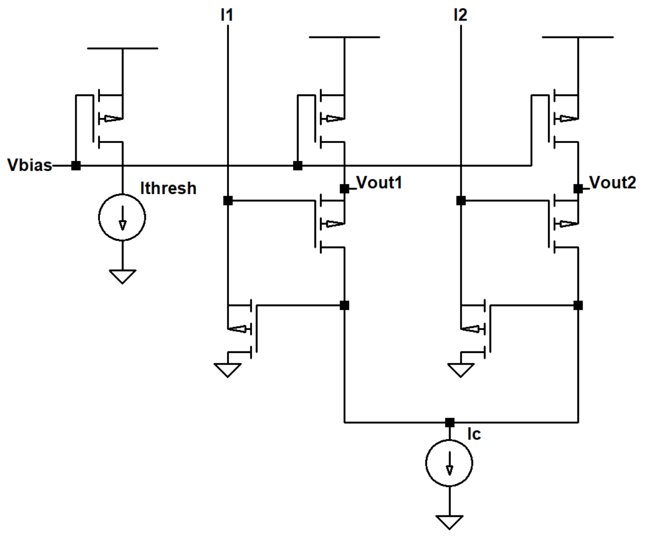
$$I_{out} = \frac{\omega_{FG}}{\omega_{ref}} I_{in} \tag{3.10}$$

The drains of the right-hand-side FGMOS's in each column are connected to each other. This enables their currents to be summed. The net current on each column corresponds to a single row of matrix multiplication.

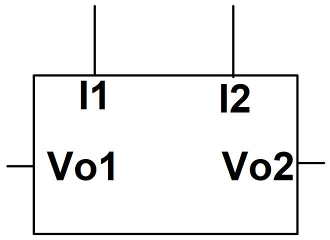
It should be noted that in the layout of the VMM provided in figure 3.34b, only the left-hand side of the circuit is present. The FGMOS's for the right-hand-side were created separately and can be viewed in figure 3.32.

3.4.4.1 The Winner-Take-All Comparator

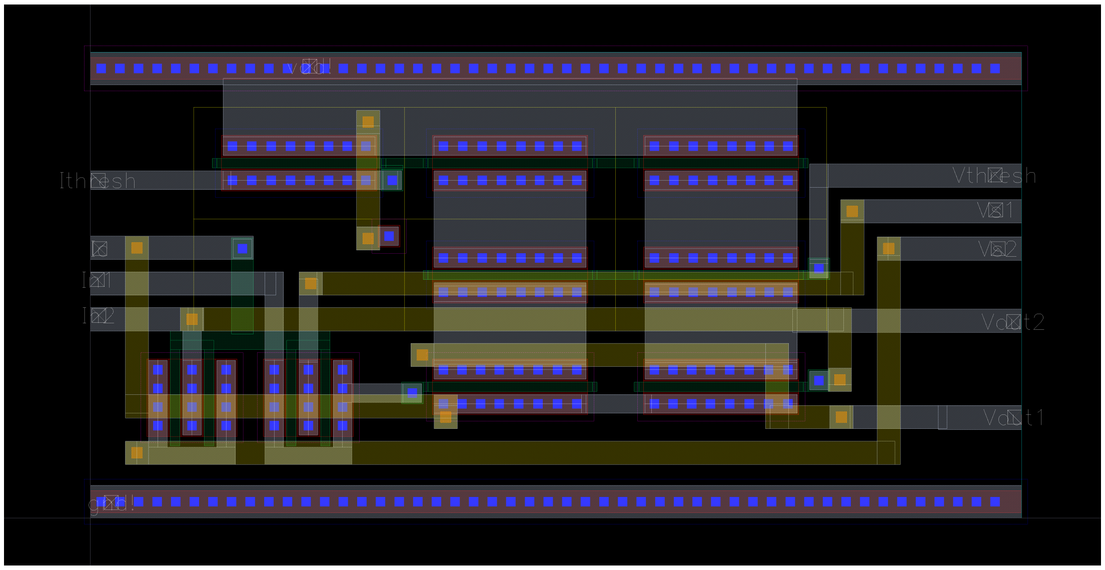
The final component is the winner-take-all (WTA) current comparator. This circuit receives two currents as its input and outputs two voltages whose values are proportional to the sizes of the currents.



(a) The WTA circuit schematic.



(b) The WTA circuit symbol.



(c) The WTA circuit layout.

Figure 3.35: The winner-take-all current comparator.

3.4.5 Classifier

The schematic of the classifier circuit is given below in figure 3.36. As previously discussed in subsection 3.3.1, a voltage integrator is added to the output of the WTA comparator in order to improve the results and behave like a low-pass filter. Unlike the placement discussed in subsection 3.3.1, here the voltage integrator is added directly to the output of the comparator followed by the saturating inverters in order to reduce the overall number of components. In the simulation stage (subsection 3.3.1), the integrator was added to the output of the saturated classifier output which would require two more inverters.

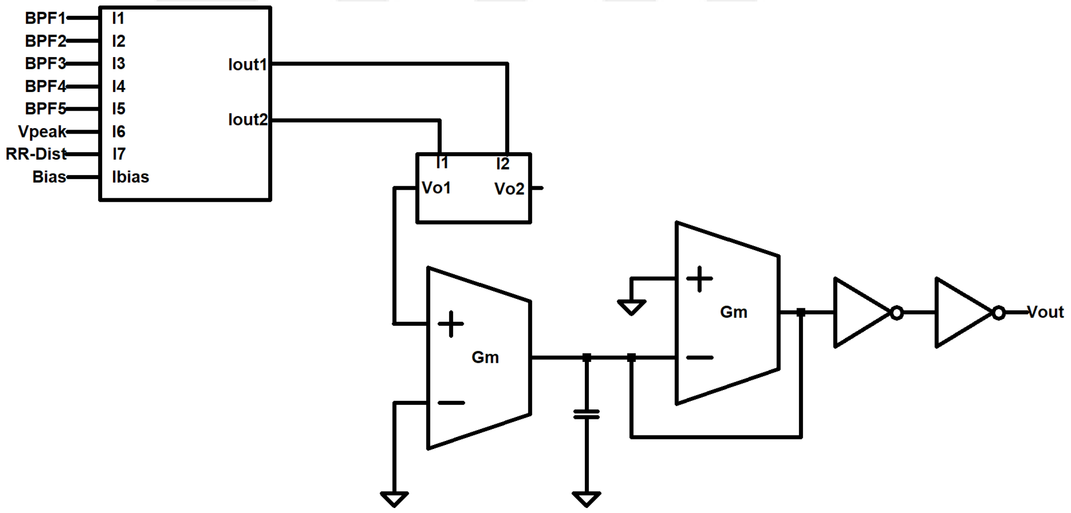


Figure 3.36: The classifier.

3.4.6 Full Circuit

The symbolic representation of the classifier connected to the feature extraction circuits is given below in figure 3.37. Figure 3.38 shows the complete layout of the chip. Figure 3.39 shows the layout with the components labelled in a similar fashion to figure 3.27b. Finally, figure 3.40 shows the chip connected to its pads.

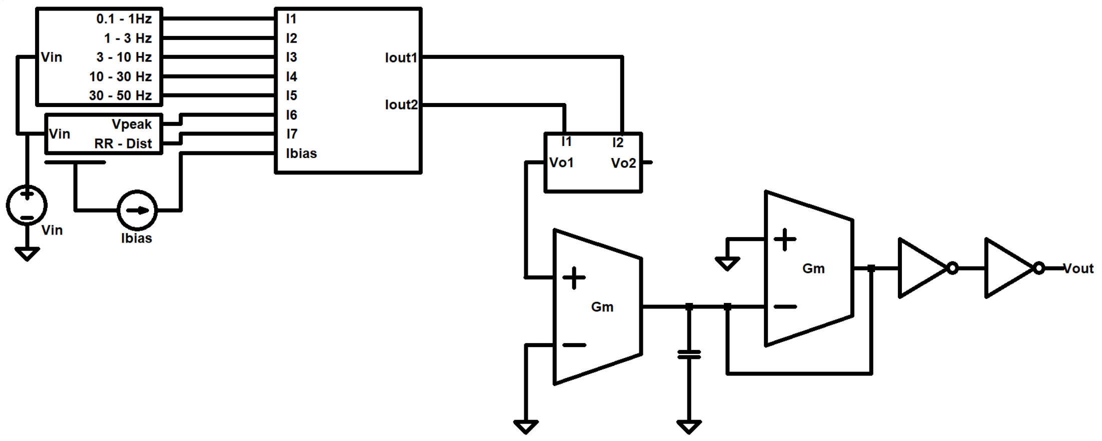


Figure 3.37: In the classifier, the outputs of the VMM are fed into the WTA. The output of the WTA corresponding to the existence of an arrhythmic beat is then filtered with a leaky OTA integrator before being thresholded and converted to a digital signal by the inverters.

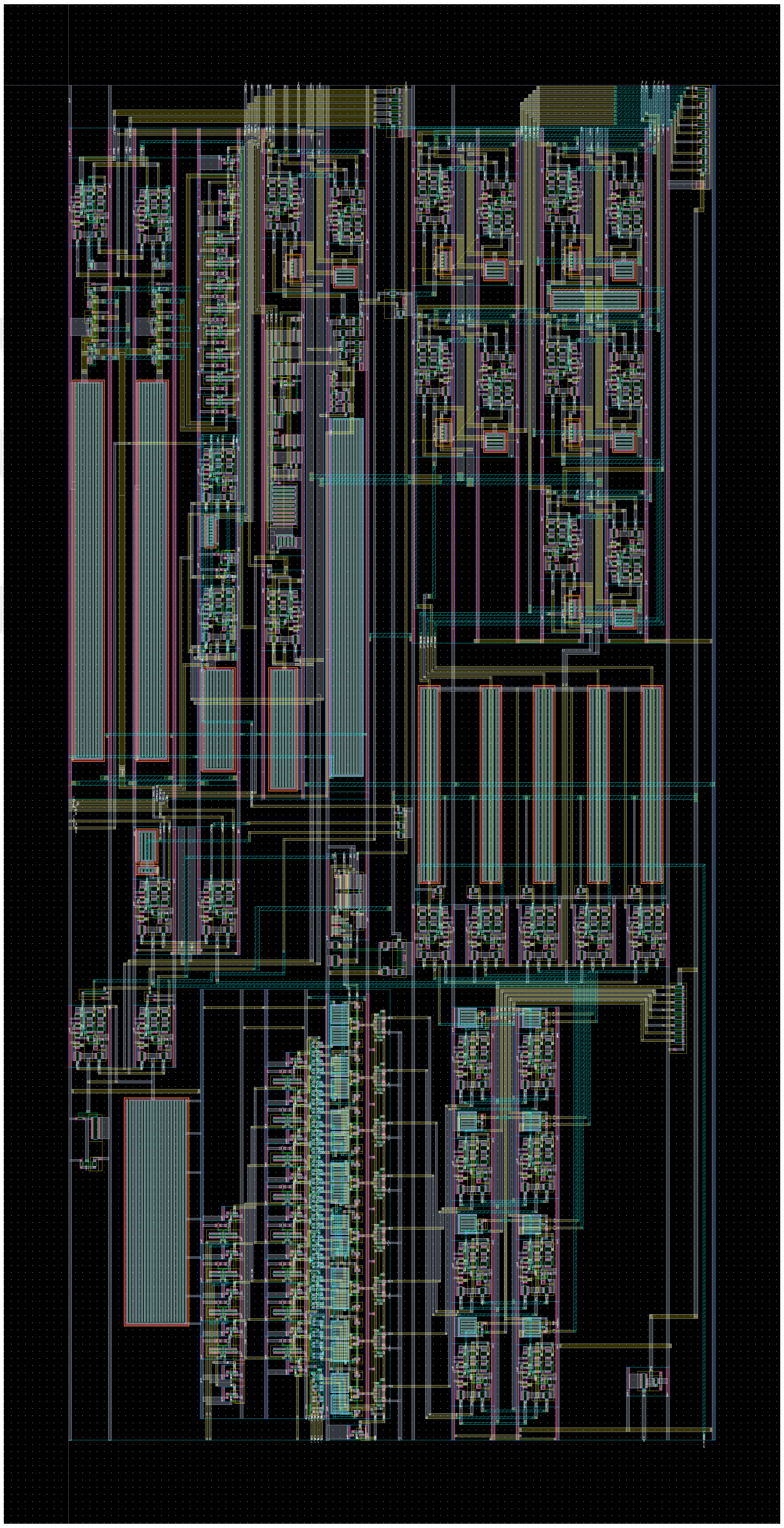


Figure 3.38: The classifier chip.

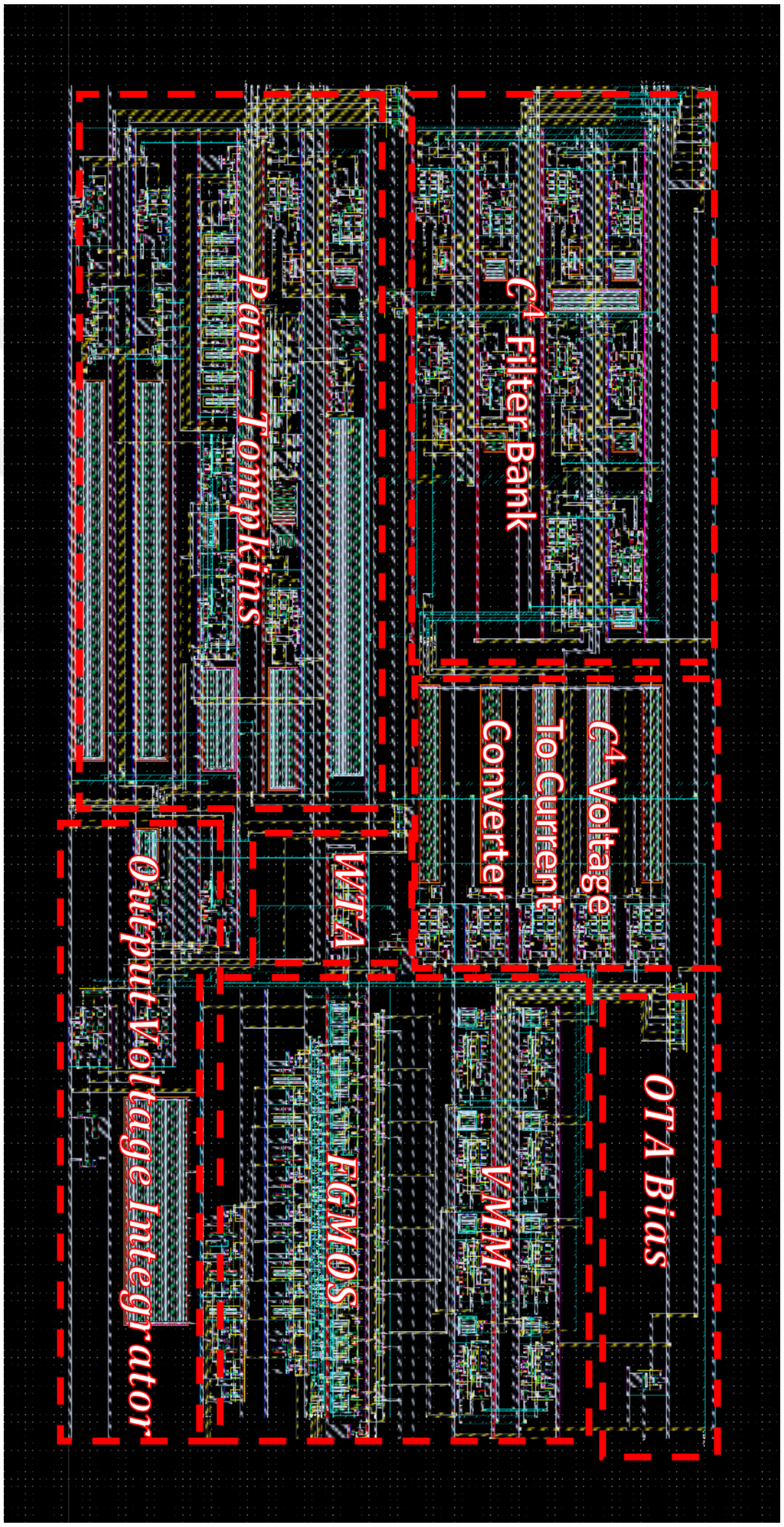


Figure 3.39: The classifier chip with components labelled.

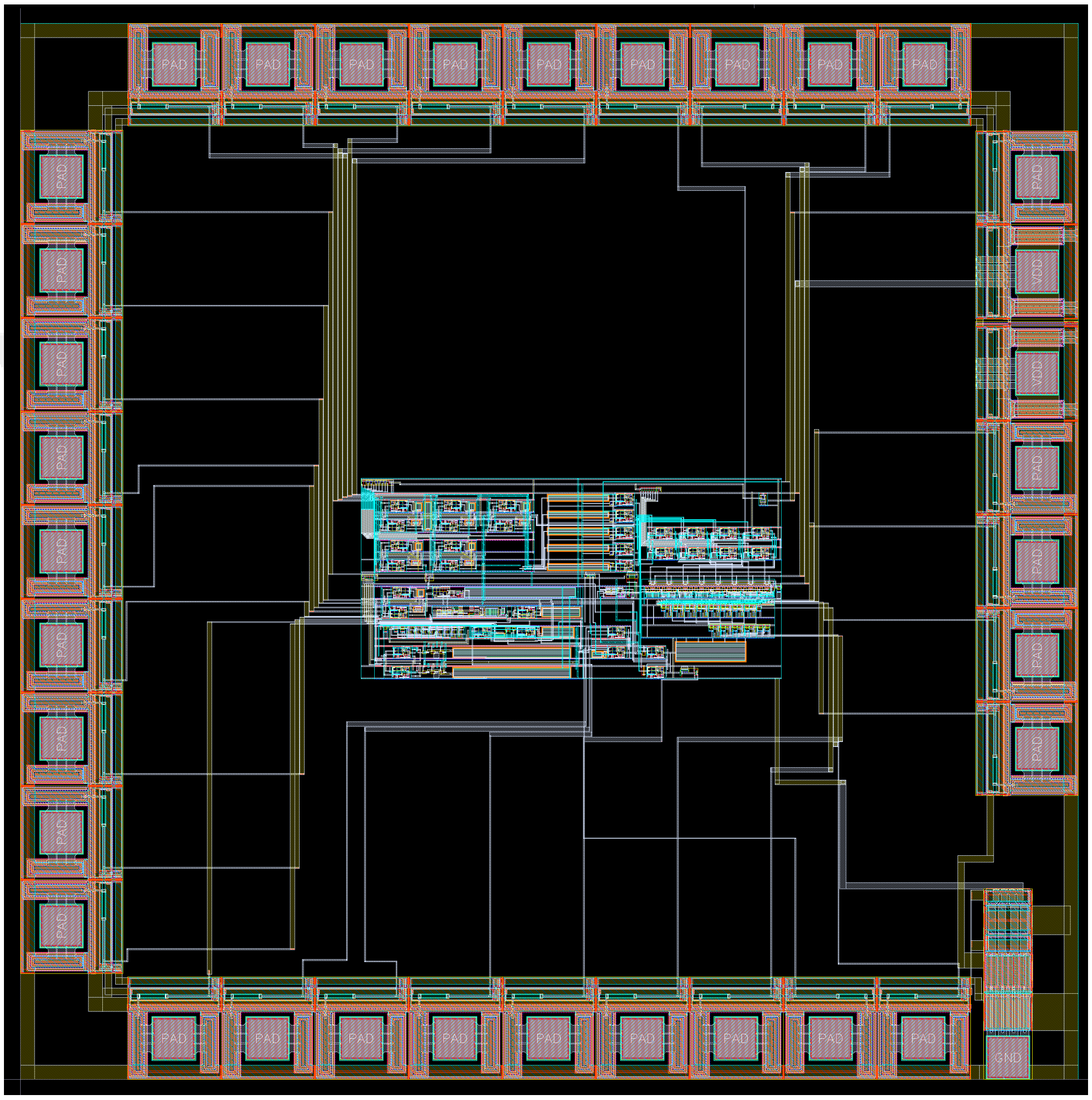


Figure 3.40: The classifier chip surrounded by pads.

3.4.7 Classifier Results

The simulation results of the classifier are given below, on figure 3.41. As can be seen from the results below, both the schematic and layout simuations show success in recognising parts of the arrhythmic waveform. As can be seen from the results, there are also some misclassifications however, compared to the correct ones, they are short in duration and can be removed with the inclusion of a low-pass filter.

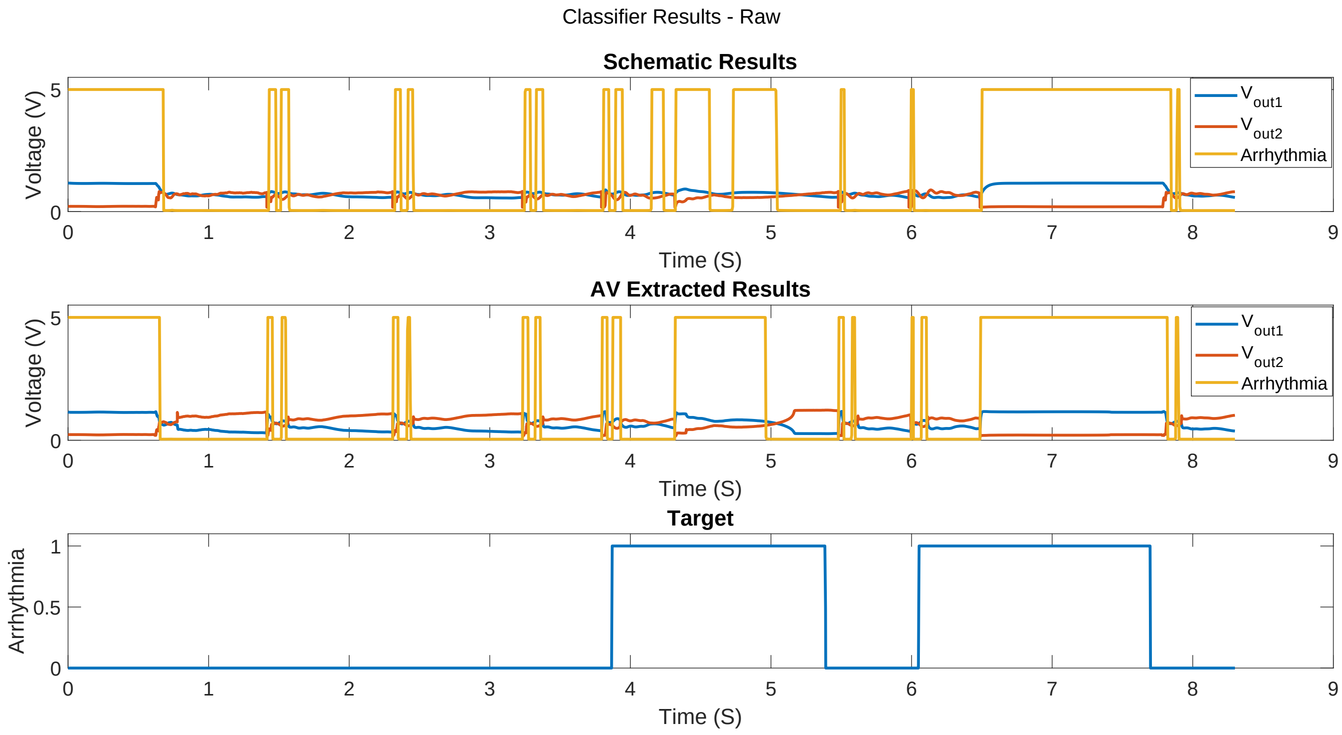


Figure 3.41: The raw classifier output. The output of the WTA prior to low-pass filtering. Both false positive and negative outputs can be seen.

The results with the inclusion of the voltage integrator at the output of the classifier are given below in figure 3.42.

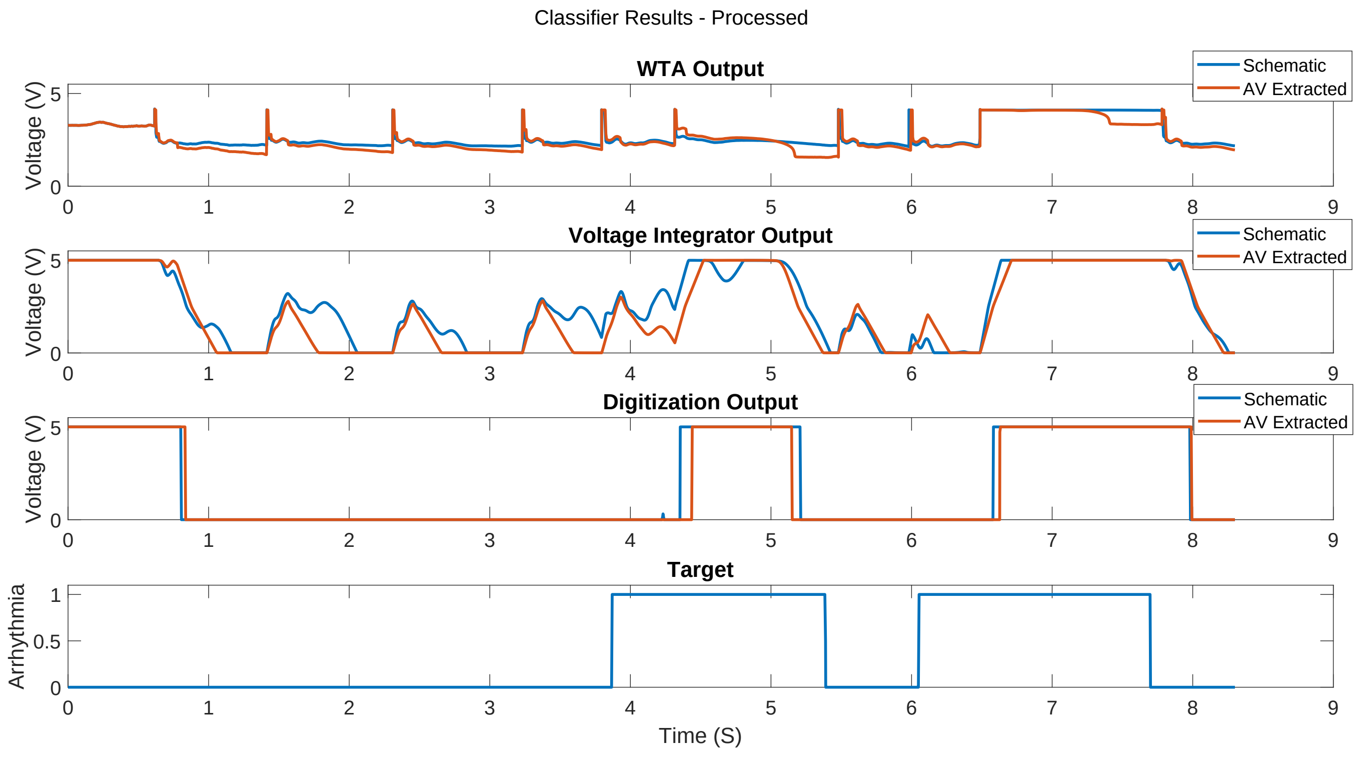


Figure 3.42: The filtered output of the WTA showing the presence of arrhythmic waveforms as desired.

As can be seen from these results, the inclusion of the voltage integrator has removed the false positives and negatives. While the duration of the classifier outputs are shorter that length of the labels, they are still long enough to cover most of the arrhythmic waveform, indicating its presence.

Figure 3.43 below shows the output of the chip on a 50 S interval of the test set containing normal and arrhythmic waveforms. Similar to the results in figure 3.42, the chip correctly classifies the arrhythmic waveforms.

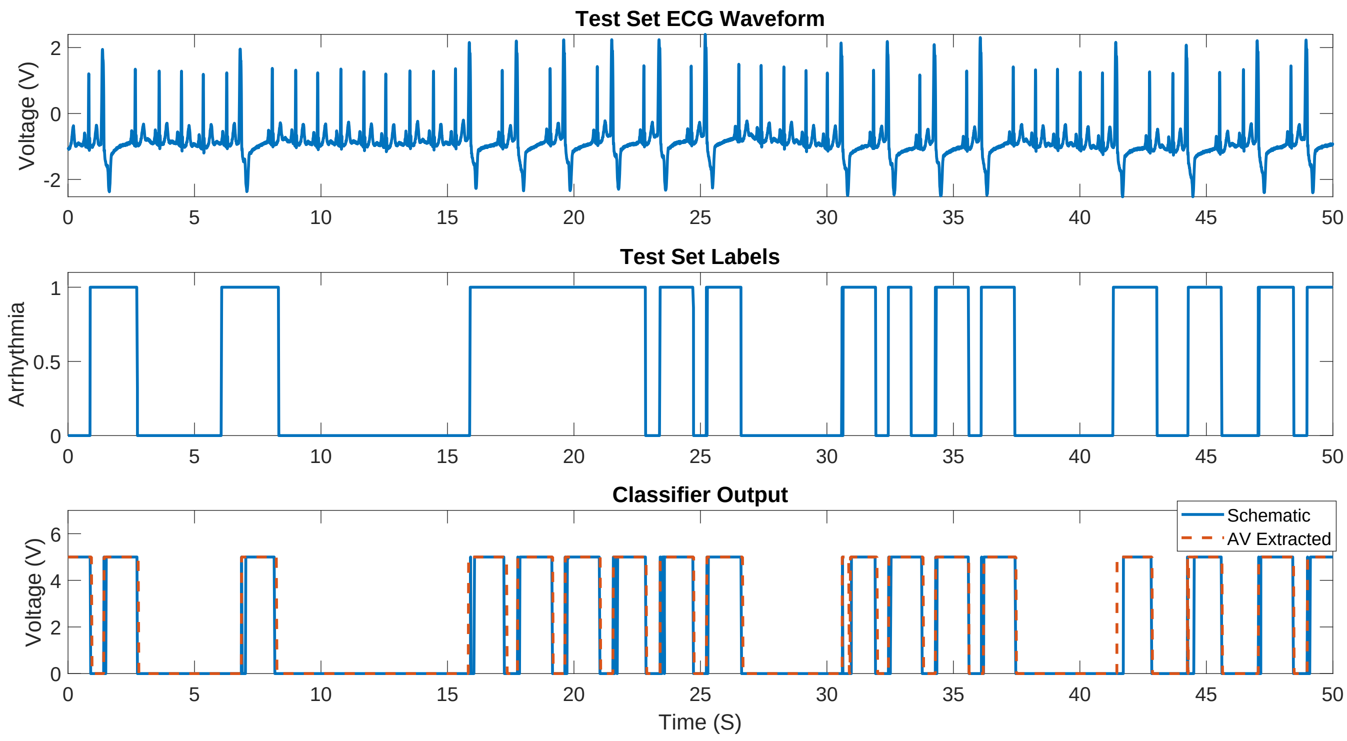


Figure 3.43: The output of the classifier with a 50 second slice from the test set. The classifier performs as intended, showing the presence of arrhythmias that align with the target labels.

3.4.7.1 Power Consumption

Simulations showed that the average current drawn by the circuit is 58 μA which, when multiplied by the Vdd voltage, shows that the total power consumption of the circuit is 290 μW.

Detailed examination of the currents drawn in the simulations show that while the upper limit of the current drawn by the analog components is set by their biasing currents (all nA), the digital components (the inverters and the D-Flip-Flop counter) drawn on the order of μA during their operation (the transitions from high-to-low and vice versa).

This problem could be resolved by replacing the digital components with low-power digital components from a more power-efficient process (with smaller transistor gate lengths). Another possible solution, at the cost of more silicon real-estate, is replacing the voltage buffers (two digital inverters connected in series) with a custom voltage inverter with a small integrating capacitor or large biasing currents. With a low threshold voltage (set by biasing the negative input terminal which may require an extra pin for the reference voltage) the capacitor can be charged and discharged to/from zero to Vdd as fast as a digital voltage buffer.

Chapter 4

Conclusion

4.1 Conclusion

In this work, the aim was to create a neuromorphic ECG arrhythmia classifier. Starting with the algorithm, followed by the hardware design and simulations, the aim was achieved. The algorithm starts by extracting seven features from the ECG waveform. These features are then fed into a perceptron that is pre-trained to classify arrhythmic waveforms.

Initial MatLab simulations show that the algorithm is successful at detecting arrhythmic waveforms. Following the success of the MatLab results, the algorithm was converted to an analog circuit on Cadence Virtuoso using the XFAB XC06M3 fabrication process. First an OTA was created which was then used to implement most of the other functions within the algorithm. A VMM, with FGMOS transistors to hold the weight values, connected to a WTA current comparator was created to implement the classifier.

The simulation results show that the classifier has succeeded in detecting the arrhythmic waveforms in the test dataset (a 50 second long slice (figure 3.43)).

Hence, the goal of the work has been successfully achieved. Potential improvements and future goals are discussed below.

4.2 Future Goals

4.2.1 Fabrication

While the circuit works successfully in the simulations, in order to verify its performance, it has to be fabricated and tested using real signals. Due to the high cost of ASIC fabrication coupled with the fabrication schedule of the XFAB foundary, the circuit could not be fabricated for experimental validation. A possible future goal is the fabrication and experimental testing of the circuit for verification.

4.2.2 Potential Improvements

As discussed before, in order to further reduce the power consumption of this circuit, the digital components can be replaced with those of a more efficient process or replaced by custom analog components that perform similarly.

Another potential improvement is training and implementing more perceptrons that are sensitive to different types of arrhythmias. This way, the circuit can identify the type of arrhythmia it encounters. The perceptrons can all receive the same inputs, negating the need to replicate the pre-processing circuit.

Bibliography

[1] Campi, F. and Toma, M. and Lodi, A. and Cappelli, A. and Canegallo, R. and Guerrieri, R., “A VLIW processor with reconfigurable instruction set for embedded applications,” *2003 IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC.*, 2003.

[2] C. R. Schlottmann and J. Hasler, *A Coordinated Approach to Reconfigurable Analog Signal Processing*. PhD thesis, Georgia Institute of Technology, 2012.

[3] Rivera-Ruiz, Moises and Cajavilca, Christian and Varon, Joseph, “Einthoven’s String Galvanometer The First Electrocardiograph,” *Texas Heart Institute Journal*, vol. 35, no. 2, p. 174–178, 2008. url:`https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2435435/`.

[4] Baird, Erika, “Artifact of the Week: The Sanborn Viso-Cardiette, model 51,” *Aurora Historical Society – Hillary House National Historic Site – Koffler Museum of Medicine*, 2017.

[5] M. Alghatrif and J. Lindsay, “A brief review: history to understand fundamentals of electrocardiography,” *Journal of Community Hospital Internal Medicine Perspectives*, vol. 2, 2012. doi: `10.3402/jchimp.v2i1.14383`.

[6] Shweta H. Jambukia, Vipul K. Dabhi, Harshadkumar B. Prajapati, “Classification of ECG signals using machine learning techniques: A survey,” in *International Conference on Advances in Computer Engineering and Applications*, 2015.

[7] C. Mead, *Analog VLSI and neural systems.* Addison-Wesley, 1989.

[8] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, “A Survey of Neuromorphic Computing and Neural Networks in Hardware,” *arXiv*, 2017.

[9] Jiapu Pan, Willis J. Tompkins, “A real-time *QRS* decetion algorithm,” *IEEE Transactions of Biomedical Engineering*, vol. 32, no. 3, pp. 230 – 236, 1985.

[10] Sedghamiz, Hooman, “Matlab Implementation of Pan Tompkins ECG QRS detector.,” *Mathworks*, 03 2014. doi: `10.13140/RG.2.2.14202.59841`.

[11] D. W. Graham, P. E. Hasler, R. Chawla, and P. D. Smith, “A low-power programmable bandpass filter section for higher order filter applications,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 6, p. 1165–1176, 2007.

[12] Hakan Töreyin, Cihan Berk Güngör, “A 0.5 nw analog ECG processor for real time R-wave detection based on Pan-Tompkins algorithm,” *IEEE EMBS International Conference on Biomedical and Health Informatics (BHI)*, pp. 1–4, 05 2019. doi: `10.1109/BHI.2019.8834508`.

[13] B. A. Minch, “Analysis and synthesis of static translinear circuits,” tech. rep., Cornell University, 2000.

[14] G. Moody and R. Mark, “The impact of the MIT-BIH Arrhythmia Database,” *IEEE Engineering in Medicine and Biology Magazine*, vol. 20, no. 3, p. 45–50, 2001. doi: `10.1109/51.932724`.

[15] Oldberger, A. L. and Amaral, L. A. N. and Glass, L. and Hausdorff, J. M. and Ivanov, P. Ch. and Mark, R. G. and Mietus, J. E. and Moody, G. B. and Peng, C.-K. and Stanley, H. E., “PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals,” *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000 (June 13). doi: `10.1161/01.CIR.101.23.e215"`.

[16] MATLAB, *9.5.0.1049112 (R2018b) Update 3.* Natick, Massachusetts: The MathWorks Inc., 2018.

[17] Cadence Design Systems, “Virtuoso Design Environment IC6.1.6-64b.500.8,” 1991.

[18] C. Das and J. Mclean, “From prototyping to small volume-the EUROPRACTICE global ASIC solution,” *Proceedings 1999 IEEE International Conference on Microelectronic Systems Education (MSE99) Systems Education in the 21st Century (Cat. No.99-63794)*, Jul 1999.

Appendix A

Extra Info

Information about the following is provided here:

- A.1: The Dataset,
- A.2: MatLab,
- A.3: Cadence,
- A.4: Artificial Neural Network Training.

A.1 Dataset

All of the ECG waveforms were obtained from the MIT - BIH Arrhythmia database [14]. Published in the 1980's, the dataset contains forty-eight half-hour recordings from forty-seven patients, which were recorded in the 1970's.

The entire database was first downloaded from *PhysioNet* [15] (where it is housed today and is freely available) then processed on MatLab in order to create the dataset.

The dataset was created by first selecting thirty patients from the database with a sufficient number of arrhythmias within their recordings. The MLII recordings were selected as they have the waveforms of interest.

In total, the generated dataset has:

- 59365 Normal waveforms,
- 6394 Arrhythmic waveforms.
  - 5734 PVC waveforms,
  - 660 APC waveforms.

The labels provided by the database are timestamped with the relevant label placed at the center of the waveform. In order to convert this to suit the needs of the algorithm, the labels were processed on MatLab so that they encompassed the entire waveform. As shown in the second subplot of figure 3.43 this process can sometimes combine consequent arrhythmias into a single long label. This did not turn out to be problematic as both the algorithm and the circuit successfully isolated each arrhythmic beat (within a combined label) individually.

A.2 MatLab

All MatLab related operations were performed on the R2018b version of Matlab. [16]

A.3 Cadence

All circuit and layout designs and simulations were performed on the Virtuoso Design Environment from Cadence Design Systems [17]. The XFAB XC06M3 fabrication process was used as the basis of the design. Licences for the XFAB XC06M3 process was obtained thanks to Europractice [18].

Figure A.1 below shows the graphical user interface of Virtuoso during the design of the layout of the classifier chip.

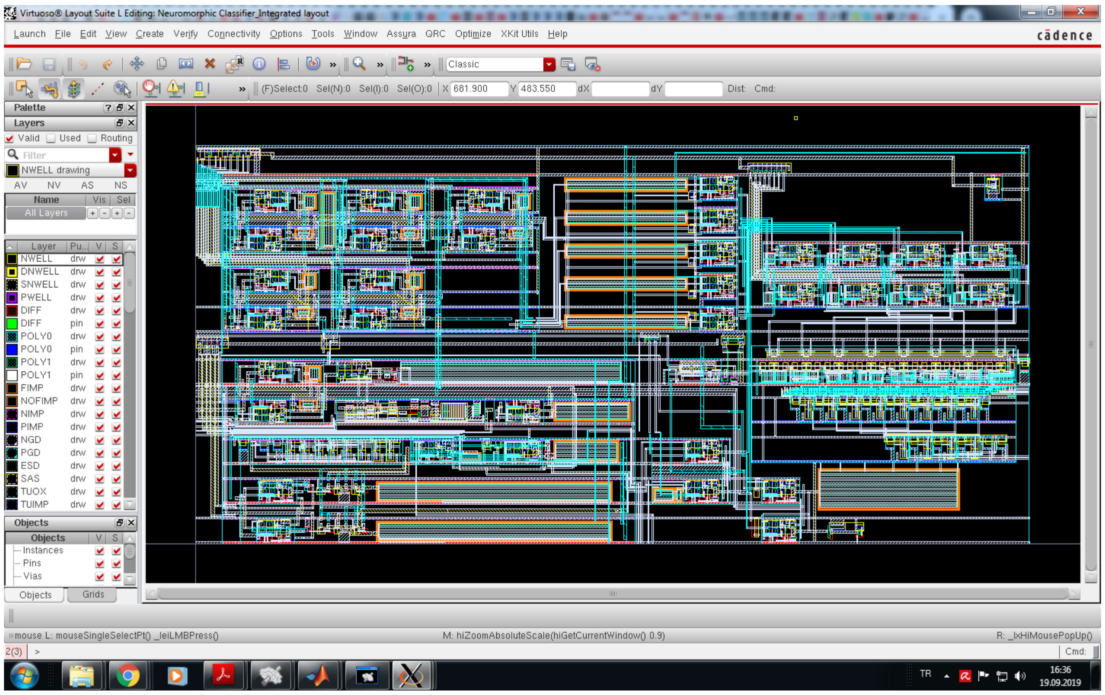


Figure A.1: A screen shot showing the graphical user interface of the Virtuoso Design Environment during the layout design process of the classifier chip.

A.4 Artificial Neural Network Training

Figure A.4 below shows the training algorithm for a single perceptron (similar to the one used in the classifier). The perceptron receives a 1xn vector as an input, multiplies it with 1xn weights (ω), sums the products of the input and weights, and passes the result through the activation function f. Next, the output is compared to the desired value and the error is calculated. If this error is equal to that found in the previous r iterations (in 1 iteration, all of the training data is used to update the weights) then that means that the network cannot learn better. If not, it means that the weights can be improved. Hence, they are updated. The update value depends on the type of update function used.

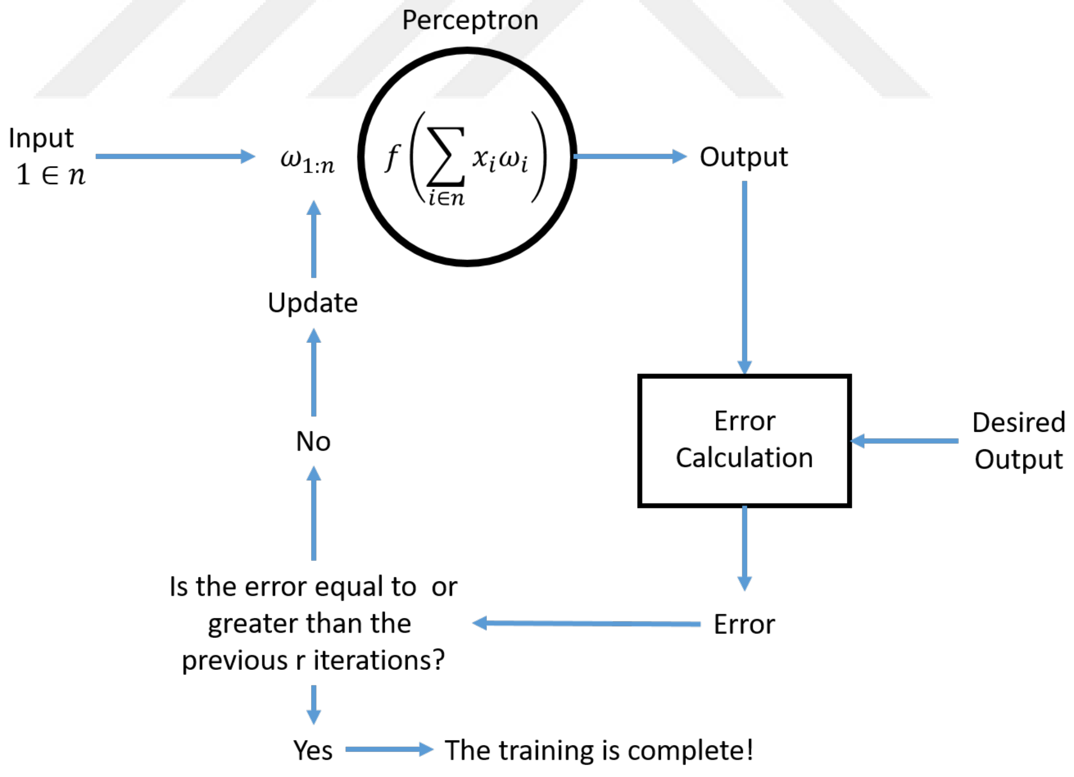

Perceptron
Input
$1 \in n$
$\omega_{1:n}$
$f\left(\sum_{i \in n} x_i \omega_i\right)$
Output
Update
Error Calculation
Desired Output
No
Is the error equal to or greater than the previous r iterations?
Error
Yes
The training is complete!

Figure A.2: ANN training

Appendix B

Test Circuits

The following test circuits are given below:

- B.1: Operational Transconductance Amplifier,
- B.2: Differentiator,
- B.3: Squarer,
- B.4: Leaky Integrator.

B.1 OTA

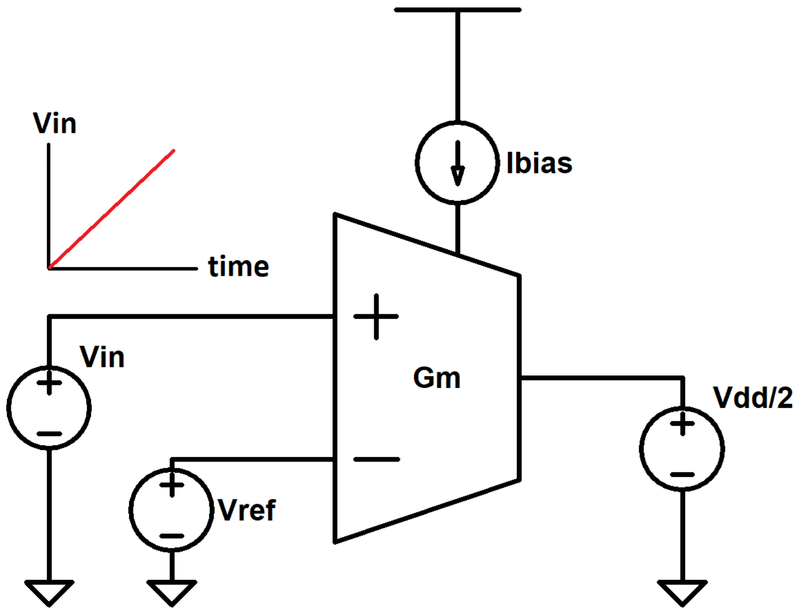


Figure B.1: The test circuit for the OTA. Results shown in figure 3.9.

B.2 Differentiator

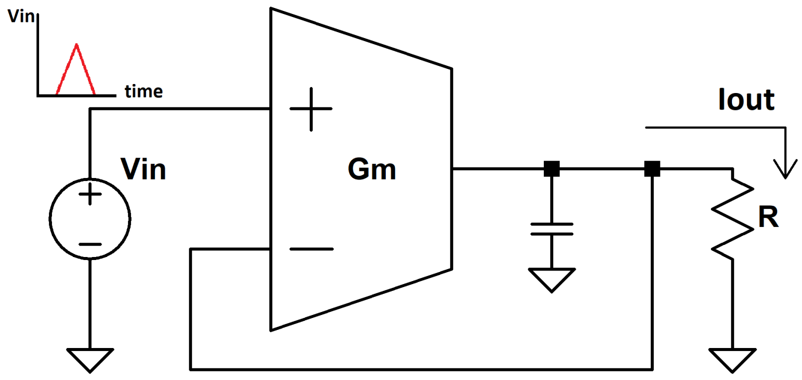


Figure B.2: The test circuit for the differentiator. Results shown in figure 3.14.

B.3 Squarer

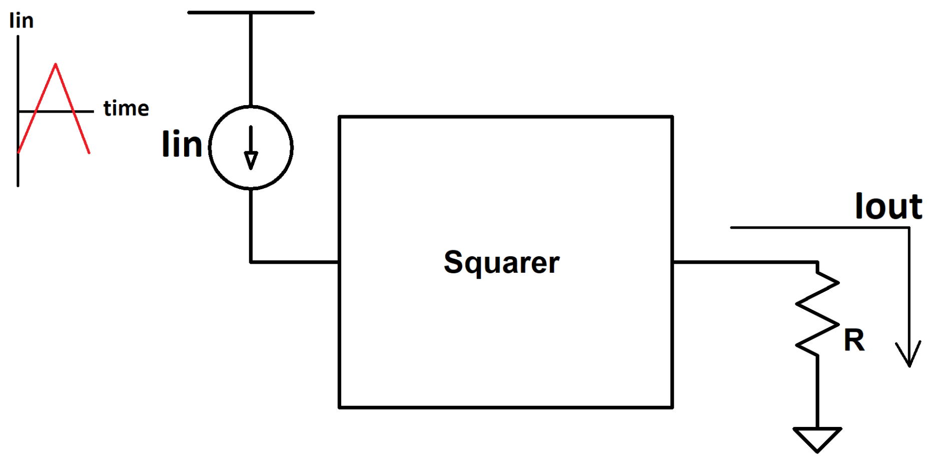


Figure B.3: The test circuit for the squarer. Results shown in figure 3.16.

B.4 Leaky Integrator

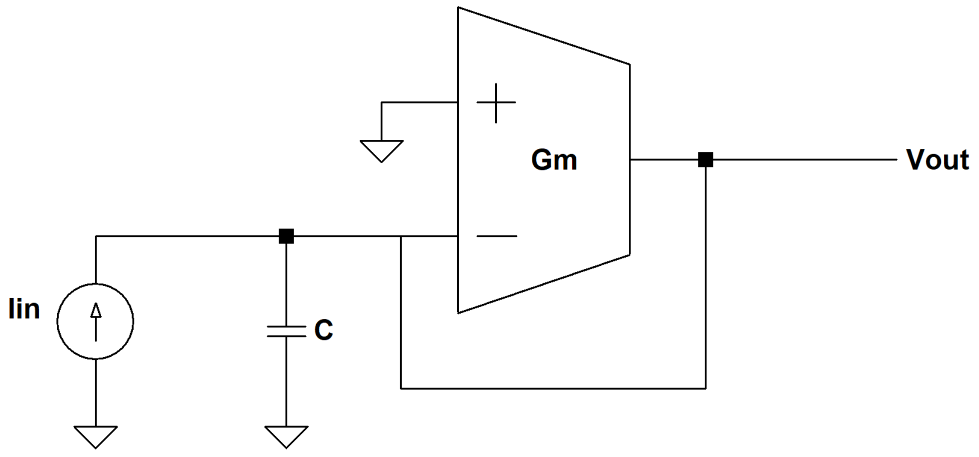


Figure B.4: The test circuit for the leaky integrator. Results shown in figure 3.18.

Appendix C

Derivations

The following derivations are given below:

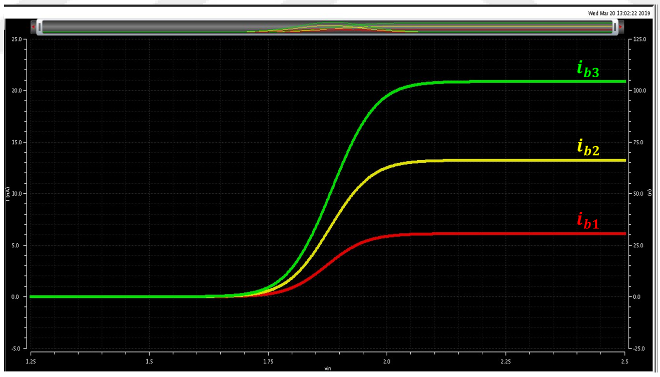
- C.1: The OTA G_m - I_{BIAS} Relationship
- C.2: Derivation of the C^4 Transfer Function
- C.3: Derivation of the Squarer Transfer Function
- C.4: Derivation of the Integrator Transfer Function

C.1 The OTA G_m - I_{BIAS} Relationship

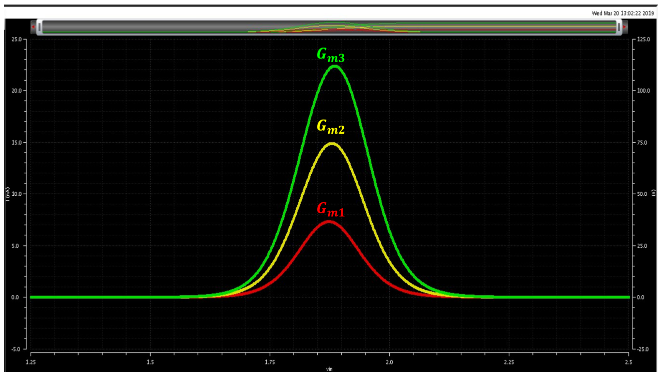
Using the test circuit in B.1, the OTA transfer function was generated for different bias current values. The transconductance (G_m) value is obtained using the following equation is used:

$$G_m = max\left(\frac{\mathrm{d}i_{out}}{\mathrm{d}\Delta V}\right) \tag{C.1}$$

The figures below (generated in *Cadence Virtuoso*) show the application of equation C.1 to three different bias current based OTA transfer functions (figure C.1). As can be seen from figure C.1b, larger bias currents lead to larger transconductance values.



(a) The OTA transfer function. At three different bias currents.



(b) The gm-I_{bias} relationship.

Figure C.1: The transconductance is equal to the maximum value of the derivative of the transfer function.

C.2 Derivation of the C^4 Transfer Function

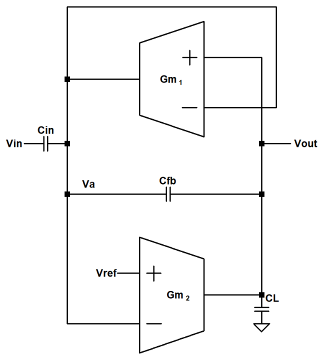


Figure C.2: The C^4 filter.

Writing nodal equations around the nodes V_{out} and V_a (figure C.2) yields:

$$\frac{V_{out}}{\frac{1}{j\omega C_L}} + \frac{V_{out} - V_a}{\frac{1}{j\omega C_{fb}}} + G_{m2}(V_a - V_{REF}) = 0 \rightarrow V_a = V_{out}\frac{j\omega(C_L + C_{fb})}{j\omega C_{fb} - G_{m2}} \tag{C.2}$$

$$\begin{gathered}\frac{\frac{V_a - V_{in}}{1}}{j\omega C_{in}} + \frac{\frac{V_a - V_{out}}{1}}{j\omega C_{fb}} + G_{m1}(V_{out} - V_a) = 0 \\ V_a(j\omega(C_{in} + C_{fb}) + G_{m1}) - V_{out}(G_{m1} + j\omega C_{fb}) = V_{in}(j\omega C_{fb} - G_{m2})\end{gathered} \tag{C.3}$$

We can simplify equations C.2 and C.3 with the following substitutions:

$C_{in} + C_{fb} = C_{IN}$

$C_L + C_{fb} = C_{OUT}$

$$V_a = V_{out}\frac{j\omega(C_{OUT})}{j\omega C_{fb} - G_{m2}} \tag{C.4}$$

$$V_a(j\omega(C_{IN}) + G_{m1}) - V_{out}(G_{m1} + j\omega C_{fb}) = V_{in}(j\omega C_{fb} - G_{m2}) \tag{C.5}$$

Combining equations C.4 and C.5 yields:

$$V_{out}\left((j\omega C_{IN}+G_{m1})\tfrac{j\omega C_{OUT}}{j\omega C_{fb}-G_{m2}}-G_{m1}-j\omega C_{fb}\right)=V_{in}(j\omega C_{in})$$

The inside of the bracket on the left hand side can be rewritten as:

$$\tfrac{-\omega^2 C_{OUT}C_{IN}+j\omega C_{OUT}G_{m1}}{-G_{m2}+j\omega C_{fb}}-G_{m1}-j\omega C_{fb}=\tfrac{-\omega^2 C_{OUT}C_{IN}+G_{m1}G_{m2}+\omega^2 C_{fb}^2+j\omega(C_{OUT}G_{m1}+G_{m2}C_{fb}-G_{m1}C_{fb})}{-G_{m2}+j\omega C_{fb}}$$

Hence,

$$\frac{V_{out}}{V_{in}}=\frac{(j\omega C_{in})-G_{m2}+j\omega C_{fb}}{-\omega^2 C_{OUT}C_{IN}+G_{m1}G_{m2}+\omega^2 C_{fb}^2+j\omega(C_{OUT}G_{m1}+G_{m2}C_{fb}-G_{m1}C_{fb})} \tag{C.6}$$

The following simplifications can be made to equation C.6:

$$\tfrac{V_{out}}{V_{in}}=\tfrac{j^2\omega^2 C_{in}C_{fb}-j\omega C_{in}G_{m2}}{G_{m1}G_{m2}+j\omega(C_L G_{m1}+C_{fb}G_{m2})+j^2\omega^2(C_{in}C_L+C_{fb}C_L+C_{in}C_{fb})}$$

$$\tfrac{V_{out}}{V_{in}}=\tfrac{\frac{1}{G_{m1}G_{m2}}}{\frac{1}{G_{m1}G_{m2}}}\tfrac{j^2\omega^2 C_{in}C_{fb}-j\omega C_{in}G_{m2}}{G_{m1}G_{m2}+j\omega(C_L G_{m1}+C_{fb}G_{m2})+j^2\omega^2(C_{in}C_L+C_{fb}C_L+C_{in}C_{fb})}$$

$$\frac{V_{out}}{V_{in}}=\frac{\frac{1}{G_{m1}G_{m2}}(j^2\omega^2 C_{in}C_{fb}-j\omega C_{in}G_{m2})}{1+j\omega\left(\frac{C_L}{G_{m2}}+\frac{C_{fb}}{G_{m1}}\right)+\frac{j^2\omega^2}{G_{m1}G_{m2}}(C_{in}C_L+C_{fb}C_L+C_{in}C_{fb})} \tag{C.7}$$

In order to transform equation C.7 into the form shown in equation 3.5 the following steps will be taken:

$$j^2\omega^2\tau^2=j^2\omega^2\tfrac{C_{in}C_L+C_{fb}C_L C_{fb}C_{in}}{G_{m1}G_{m2}}\rightarrow\tau=\sqrt{\tfrac{C_{in}C_L+C_{fb}C_L C_{fb}C_{in}}{G_{m1}G_{m2}}}$$

$$\tfrac{\tau}{Q}=\tfrac{C_L}{G_{m2}+\frac{C_{fb}}{G_{m1}}}\rightarrow Q=\tfrac{\sqrt{\frac{C_{in}C_L+C_{fb}C_L C_{fb}C_{in}}{G_{m1}G_{m2}}}}{\frac{C_L}{G_{m2}+\frac{C_{fb}}{G_{m1}}}}$$

Moving on:

$$\tfrac{V_{out}}{V_{in}}=\tfrac{\frac{1}{G_{m1}G_{m2}}(j^2\omega^2 C_{in}C_{fb}-j\omega C_{in}G_{m2})}{1+j\omega\frac{\tau}{Q}+j^2\omega^2\tau^2}\ \tfrac{\frac{j^2\omega^2 C_{in}C_{fb}-j\omega C_{in}G_{m2}}{G_{m1}G_{m2}}}{\frac{\tau}{Q}}=\tfrac{j^2\omega^2 C_{in}C_{fb}-j\omega C_{in}G_{m2}}{G_{m1}G_{m2}}\tfrac{G_{m1}G_{m2}}{C_L G_{m1}+C_{fb}G_{m2}}=\tfrac{j^2\omega^2 C_{in}C_{fb}-j\omega C_{in}G_{m2}}{C_L G_{m1}+C_2 G_{m2}}$$

$$\frac{j^2\omega^2 C_{in}C_{fb} - j\omega C_{in}G_{m2}}{C_L G_{m1} + C_2 G_{m2}} = \frac{C_{in}G_{m2}}{C_L G_{m1} + C_{fb}G_{m2}} j\omega \left(j\omega \frac{C_{fb}}{G_{m2}} - 1 \right)$$

$$\frac{V_{out}}{V_{in}} = \frac{\frac{\tau}{Q} \frac{C_{in}G_{m2}}{C_L G_{m1} + C_{fb}G_{m2}} \left(j\omega \frac{C_{fb}}{G_{m2}} - 1 \right)}{1 + j\omega \left(\frac{C_L}{G_{m2}} + \frac{C_{fb}}{G_{m1}} \right) + \frac{j^2\omega^2}{G_{m1}G_{m2}} (C_{in}C_L + C_{fb}C_L + C_{in}C_{fb})}$$
$$\tau_a = \frac{C_{fb}}{G_{m2}}$$
$$\tau_b = \frac{C_{inG_{m2}}}{C_L G_{m1} + C_{fb}G_{m2}} \tag{C.8}$$

Converting equation C.8 to the S domain:

$$\frac{V_{out}}{V_{in}} = \frac{s\tau_b \frac{\tau}{Q}(s\tau_a - 1)}{1 + s\frac{\tau}{Q} + s^2\tau^2} \tag{C.9}$$

Which is the equation found in 3.6.

C.3 Derivation of the Squarer Transfer Function

The main principle behind this circuit is *translinearity* and translinear circuits [13]. The term is defined as:

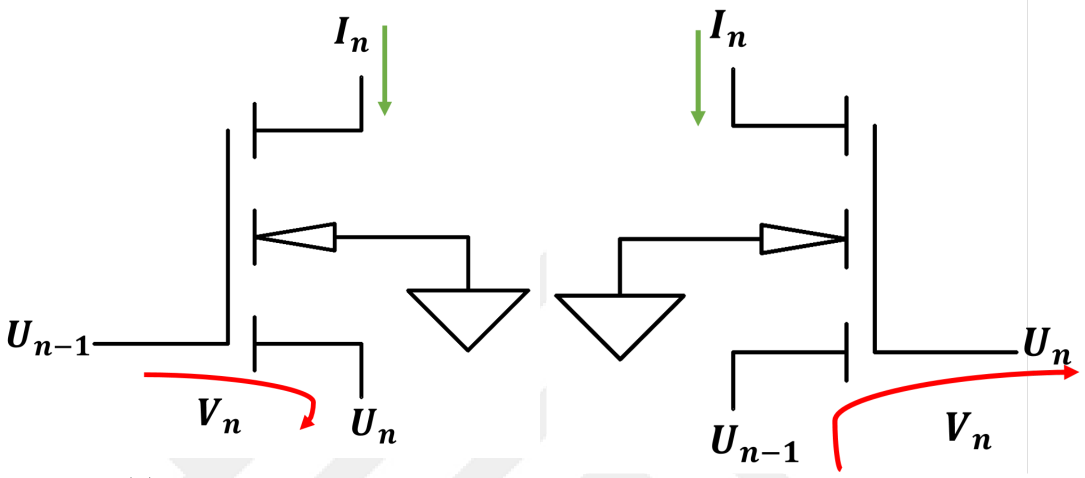
”*...a class of circuits whose large-signal behaviour hinges on the extraordinary precise exponential current-voltage characteristic of the bipolar transistor and the intimate thermal contact and close matching of monolithically integrated devices.*” [13]

Translinear circuits have the ability to perform analog mathematical operations with currents (such as multiplication and squaring) that are not possible with linear circuits. The exponential current-voltage relationship (mentioned above) of an nMOS transistor can be described with the following equation:

$I = \frac{W}{L} I_0 e^{\kappa V_g / U_T} \left(e^{-V_s/U_T} - e^{-V_d/U_T} \right)$

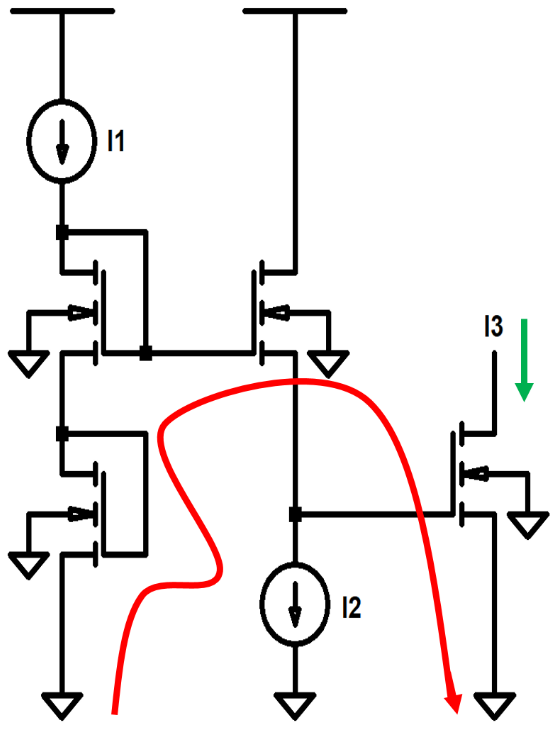
If V_{DS} (the drain-to-source voltage) across the transistor is greater than $4U_T$, then the equation above could be simplified to:

$$I = \frac{W}{L} I_0 e^{(\kappa V_g - V_s)/U_T} \tag{C.10}$$



(a) Clockwise element.

(b) Counterclockwise element.



(c) A simplified version of the squaring circuit in 3.15a.

Figure C.3: The red arrows indicate the orientation of the transistor in the translinear loop while the green arrows indicate the direction of the current.

According to the translinearity principle, the product of subthreshold currents flowing through the clockwise (figure C.3a) and counterclockwise (figure C.3b) elements within a translinear loop (figure C.3c) must be equal. As shown by equation C.11 below. For the full derivation of this equation refer to [13].

$$\prod_{n \epsilon CW} I_n = \prod_{n \epsilon CCW} I_n \tag{C.11}$$

The circuit in figure C.3c is a simplified version of the upper potion of the squaring circuit in figure 3.15a. Following the loop (the red arrow) within this circuit, it can be seen that the current I_1 flows through the two counterclockwise transistors while the currents I_2 and I_3 flow through the two clockwise transistors. Applying equation C.11 to this circuit yields:

$$I_1 * I_1 = I_2 * I_3 \rightarrow (I_1)^2 = I_2 * I_3 \tag{C.12}$$

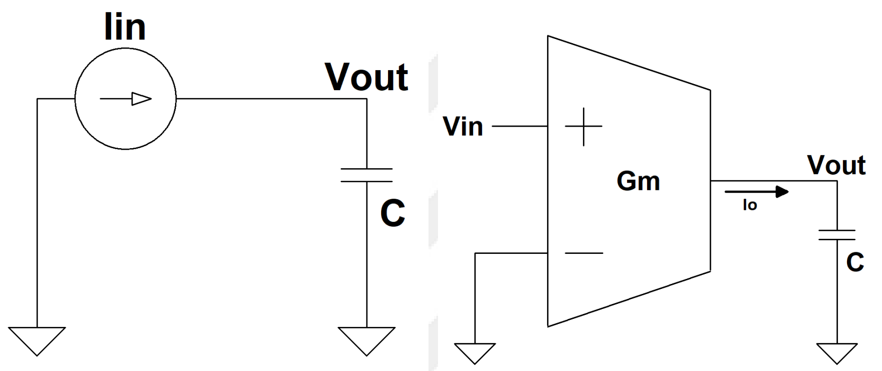
Rearranging the terms within C.12 leads to:

$$I_3 = \frac{(I_1)^2}{I_2} \tag{C.13}$$

Which is the same as equation 3.7.

C.4 Derivation of the Integrator Transfer Function

First, let's begin with the ideal current voltage relationship of a capacitor:



(a) The output voltage is equal to the integration of the current flowing into the capacitor. (b) The circuit implemented with an OTA.

Figure C.4: The integrator.

Where (assuming that the initial voltage across the capacitor is equal to 0) the voltage across the capacitor (V_{out} in figure C.4a) is equal to the product of the integration of the voltage over time and the inverse of the capacitance of the capacitor:

$$V_{out} = \frac{1}{C}\int_0^T i_{in} dt \tag{C.14}$$

Which can be rewritten in the Laplace (S-) domain as:

$$V_{out}(S) = \frac{1}{sC} I_{in}(S) \tag{C.15}$$

The circuit in figure C.4a can be implemented using an OTA, as shown in figure C.4b. Here, rewriting equation C.15 becomes:

$$V_{out}(S) = \frac{1}{sC} G_m V_{in}(S) \tag{C.16}$$

Under ideal conditions, the capacitor will hold onto the charge it accumulates forever. However, in real life capacitors have an internal resistance that leaks the accumulated charge. An external resistance may also be introduced which discharges the capacitor on purpose.

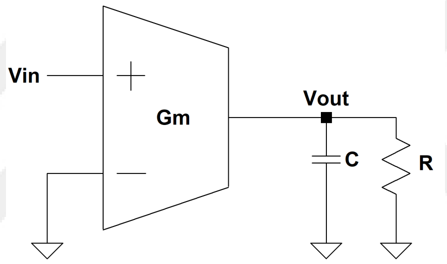


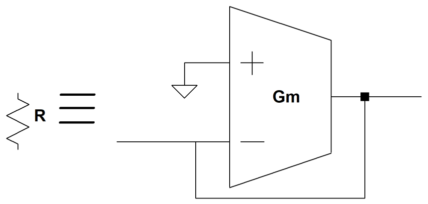
Figure C.5: The inclusion of a resistance causes the capacitor to leak.

Adding a resistor to the output of figure C.4b as shown in figure C.5 alters the transfer function as follows:

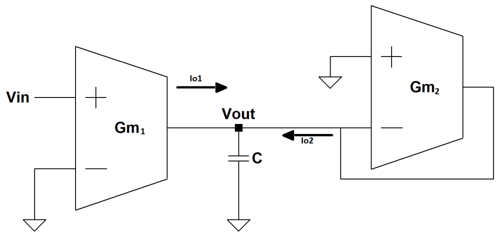
$$V_{out}(S) = \frac{G_m R}{1 + sCR} V_{in}(S) \tag{C.17}$$

Where $G_m R$ determines the overall gain and the product of R and C determines the time constant of the leakage.

The circuit in figure C.5 can be created using an OTA based pseudo-resistor (figure C.6a, which is created by grounding the positive input terminal and connecting the negative input terminal to the output) resulting in figure C.6b. Depending on the direction of the current, the current will flow through the upper PMOS (positive) or lower NMOS (negative) output transistors of the OTA.



(a) An OTA based pseudo-resistor with tunable resistance. The resistance is equal to $\frac{1}{G_m}$.



(b) The leaky integrator circuit implemented with an OTA based pseudo-resistor.

Figure C.6: The leaky integrator implemented with a pseudo-resistance.

Where the output currents of the two OTAs are:

$$I_{o1} = G_{m1}V_{in}$$
$$I_{o2} = -G_{m2}V_{out}$$

The output voltage is, hence, equal to the total current flowing through the capacitor:

$$V_{out} = \frac{I_{o1}+I_{o2}}{sC} = \frac{G_{m1}V_{in}-G_{m2}V_{out}}{sC}$$

The overall transfer function can be rewritten as:

$$\frac{V_{out}}{V_{in}} = \frac{G_{m1}}{1+\frac{sC}{G_{m2}}} \tag{C.18}$$

Where, similar to RC in equation C.17, $\frac{C}{G_{m2}}$ is the leakage time constant.

Removing the first OTA (the one that converts the input voltage into a current) so that the input is a current (as shown in figure 3.17a, section 3.4.2.6) alters the transfer function as 3.8.

$$\frac{V_{out}}{I_{in}} = \frac{1}{1+\frac{sC}{G_m}} \tag{C.19}$$

Appendix D

Code

The MatLab code used for the preprocessing stage is given below. The parts of the code where the Pan-Tompkins algorithm [9] is used are from [10].

```
clear all
close all
clc

load ECG_dataset.mat
ECG_dataset_signals = ECG_dataset_signals(1,:);
ECG_dataset_tm = ECG_dataset_tm(1,:);
ECG_dataset_Fs = 360;
%%
% First Stage: BPF
ECG_dataset_signals_bpf = [];

f1=5;                              % cuttoff low frequency to
    get rid of baseline wander
f2=15;                             % cuttoff frequency to
    discard high frequency noise
N = 3;                             % order of 3 less
    processing
```

```

for i = 1:length(ECG_dataset_Fs)
     fs = ECG_dataset_Fs(i);
    Wn=[f1 f2]*2/fs;               % cutt off based on fs
     [a,b] = butter(N,Wn);         % bandpass filtering

     ecg_h = filtfilt(a,b,ECG_dataset_signals(i,:));
     ecg_h = ecg_h/ max( abs(ecg_h));

     ECG_dataset_signals_bpf = [ECG_dataset_signals_bpf;
         ecg_h];
end
clear a b ecg_h i N Wn fs f1 f2
%%
% Second Stage: Derivative
% ------ H(z) = (1/8T)(-z^(-2) - 2z^(-1) + 2z + z^(2))
    ---------- %
ECG_dataset_signals_bpf_ddt = [];
for i = 1:length(ECG_dataset_Fs)
     fs = ECG_dataset_Fs(i);
     b = [1 2 0 -2 -1].*(1/8)*fs;
     ecg_d = filtfilt(b,1,ECG_dataset_signals_bpf(i,:));
     ecg_d = ecg_d/max(ecg_d);
     ECG_dataset_signals_bpf_ddt = [
         ECG_dataset_signals_bpf_ddt; ecg_d];
end
clear b i ecg_d fs
%%
% Third Stage: Squareing
ECG_dataset_signals_bpf_ddt_sq =
    ECG_dataset_signals_bpf_ddt.^2;

%%
```

```
% Fourth Stage: Integrator
%————Y(nt) = (1/N)[x(nT−(N − 1)T)+ x(nT − (N − 2)T)
    +...+x(nT)]———————%
ECG_PT = [];
for i = 1:length(ECG_dataset_Fs)
     fs = ECG_dataset_Fs(i);
     ecg_m = conv(ECG_dataset_signals_bpf_ddt_sq(i,:) ,...
          ones(1 ,round(0.150*fs))/round(0.150*fs));
     ECG_PT = [ECG_PT; ecg_m];
end
clear i ecg_m fs
%%
plot_PanTompson = 1;
if(plot_PanTompson == 1)
     ECG_sample_id = 1;
     Time_sample = [260000, 265000];

     aa_Raw_signal = ...
          ECG_dataset_signals(ECG_sample_id,[Time_sample(1):
              Time_sample(2)]);
     time = ECG_dataset_tm(ECG_sample_id,[Time_sample(1):
         Time_sample(2)]);
     ac_BPF_signal = ...
          ECG_dataset_signals_bpf(ECG_sample_id,[Time_sample
              (1):Time_sample(2)]);
     ad_ddt_signal = ECG_dataset_signals_bpf_ddt(
         ECG_sample_id,...
          [Time_sample(1):Time_sample(2)]);
     ae_squared_signal = ECG_dataset_signals_bpf_ddt_sq(
         ECG_sample_id,...
          [Time_sample(1):Time_sample(2)]);
     ab_integrated_signal = ECG_PT(ECG_sample_id,[
         Time_sample(1):Time_sample(2)]);

     figure(2)
     sgtitle('Pan Tompkins Algorithm')
     subplot(5,1,1)
     plot(time,aa_Raw_signal);
     title('Raw Signal')
     xlim([time(1), time(end)])

     subplot(5,1,2)
     plot(time,ac_BPF_signal);
     title('BPF Filtered Signal')
     xlim([time(1), time(end)])

     subplot(5,1,3)
     plot(time,ad_ddt_signal);
     title('d/dt Signal')
     xlim([time(1), time(end)])

     subplot(5,1,4)
     plot(time,ae_squared_signal);
     title('d/dt Signal')
     xlim([time(1), time(end)])

     subplot(5,1,5)
     plot(time,ab_integrated_signal);
     title('Integrated Signal')
     xlim([time(1), time(end)])
     xlabel('Time (s)')

%     figure(2)
%     subplot(2,1,1)
%     plot(time,ab_integrated_signal);
%     [pks, lcs] = ...
%           findpeaks(ab_integrated_signal,'MinPeakHeight
    ',0.05,'MinPeakDistance',200)
%       subplot(2,1,2)
%       plot(time,smooth(ab_integrated_signal));
%       findpeaks(smooth(ab_integrated_signal),...
%           'MinPeakHeight',0.05,'MinPeakDistance',200)
    clear aa_Raw_signal ac_BPF_signal ad_ddt_signal
        ae_squared_signal
    clear time ECG_sample_id Time_sample
        ab_integrated_signal
end

plot_PanTompson = 1;
if(plot_PanTompson == 1)
    ECG_sample_id = 1;
    Time_sample = [260000, 265000];

    aa_Raw_signal = ...
        ECG_dataset_signals(ECG_sample_id,[Time_sample(1):
            Time_sample(2)]);
    time = ECG_dataset_tm(ECG_sample_id,[Time_sample(1):
        Time_sample(2)]);
    ac_BPF_signal = ...
        ECG_dataset_signals_bpf(ECG_sample_id,[Time_sample
            (1):Time_sample(2)]);
    ad_ddt_signal = ECG_dataset_signals_bpf_ddt(
        ECG_sample_id,...
        [Time_sample(1):Time_sample(2)]);
    ae_squared_signal = ECG_dataset_signals_bpf_ddt_sq(
        ECG_sample_id,...
        [Time_sample(1):Time_sample(2)]);
    ab_integrated_signal = ECG_PT(ECG_sample_id,[
        Time_sample(1):Time_sample(2)]);

    figure(1)
    sgtitle('Pan Tompkins Algorithm')
    subplot(5,1,1)
    plot(time,aa_Raw_signal);
    title('Raw Signal')
    xlim([time(1), time(end)])

    subplot(5,1,2)
    plot(time,ac_BPF_signal);
    title('BPF Filtered Signal')
    xlim([time(1), time(end)])

    subplot(5,1,3)
    plot(time,ad_ddt_signal);
    title('d/dt Signal')
    xlim([time(1), time(end)])

    subplot(5,1,4)
    plot(time,ae_squared_signal);
    title('d/dt Signal')
    xlim([time(1), time(end)])

    subplot(5,1,5)
    plot(time,ab_integrated_signal);
    title('Integrated Signal')
    xlim([time(1), time(end)])
    xlabel('Time (s)')

%     figure(2)
%     subplot(2,1,1)
%     plot(time,ab_integrated_signal);
%     [pks, lcs] = ...
%           findpeaks(ab_integrated_signal,'MinPeakHeight
    ',0.05,'MinPeakDistance',200)
%       subplot(2,1,2)
%       plot(time,smooth(ab_integrated_signal));
%       findpeaks(smooth(ab_integrated_signal),...
%           'MinPeakHeight',0.05,'MinPeakDistance',200)
    clear aa_Raw_signal ac_BPF_signal ad_ddt_signal
        ae_squared_signal
    clear time ECG_sample_id Time_sample
        ab_integrated_signal plot_PanTompson
end
%%
% Feature Extraction
RR_Amplitudes = {};
R_Intervals = {};
for i = 1:length(ECG_dataset_Fs)
    Signal = ECG_PT(i,:);
    time = ECG_dataset_tm(i,:);
    [Peaks, Locs] = findpeaks(Signal,'MinPeakHeight',0.05,
        'MinPeakDistance',200);
    RR_Amplitudes = [RR_Amplitudes;{Peaks}];
    R_Intervals = [R_Intervals; {time(Locs(2:end))-time(
        Locs(1:end-1))}];
end
clear Signal time i Peaks Locs
% BPF Features
N = 3;
ECG_dataset_bpf_a = [];
ECG_dataset_bpf_b = [];
ECG_dataset_bpf_c = [];
ECG_dataset_bpf_d = [];
ECG_dataset_bpf_e = [];

```

```
for i = 1: length(ECG_dataset_Fs)
     fs = ECG_dataset_Fs(i);
    % a. 0.1-1 Hz
    Wn_a=[0.1 1]*2/fs;
     [bpf_a_a,bpf_a_b] = butter(N,Wn_a);
     ECG_dataset_bpf_a = [ECG_dataset_bpf_a;filtfilt(
        bpf_a_a,bpf_a_b,ECG_dataset_signals(i,:))];
    % b. 1 Hz-3 Hz
    Wn_b=[1 3]*2/fs;
     [bpf_b_a,bpf_b_b] = butter(N,Wn_b);
     ECG_dataset_bpf_b = [ECG_dataset_bpf_b;filtfilt(
        bpf_b_a,bpf_b_b,ECG_dataset_signals(i,:))];
    % c. 3 Hz- 10 Hz
    Wn_c=[3 10]*2/fs;
     [bpf_c_a,bpf_c_b] = butter(N,Wn_c);
     ECG_dataset_bpf_c = [ECG_dataset_bpf_c;filtfilt(
        bpf_c_a,bpf_c_b,ECG_dataset_signals(i,:))];
    % d. 10 Hz - 30 Hz
    Wn_d=[10 30]*2/fs;
     [bpf_d_a,bpf_d_b] = butter(N,Wn_d);
     ECG_dataset_bpf_d = [ECG_dataset_bpf_b;filtfilt(
        bpf_d_a,bpf_d_b,ECG_dataset_signals(i,:))];
    % e. 30 Hz - 50 Hz
    Wn_e=[30 50]*2/fs;
     [bpf_e_a,bpf_e_b] = butter(N,Wn_e);
     ECG_dataset_bpf_e = [ECG_dataset_bpf_e;filtfilt(
        bpf_e_a,bpf_e_b,ECG_dataset_signals(i,:))];
end
Time_sample = [260000, 265000];
figure()
          ECG_dataset_signals=ECG_dataset_signals([
              Time_sample(1):Time_sample(2)]);
        time = ECG_dataset_tm([Time_sample(1):Time_sample(2)])
            ;
subplot(6,1,1)
plot(time,ECG_dataset_signals)
title('Raw ECG Signal')
xlim([time(1),time(end)])
                ECG_dataset_signals=ECG_dataset_bpf_a([Time_sample
                    (1):Time_sample(2)]);
        time = ECG_dataset_tm([Time_sample(1):Time_sample(2)])
            ;
subplot(6,1,2)
plot(time,ECG_dataset_signals)
title('0.1 - 1HZ BPF ECG Signal')
xlim([time(1),time(end)])
                ECG_dataset_signals=ECG_dataset_bpf_b([Time_sample
                    (1):Time_sample(2)]);
        time = ECG_dataset_tm([Time_sample(1):Time_sample(2)])
            ;
subplot(6,1,3)
plot(time,ECG_dataset_signals)
title('1 - 3HZ BPF ECG Signal')
xlim([time(1),time(end)])
                ECG_dataset_signals=ECG_dataset_bpf_c([Time_sample
                    (1):Time_sample(2)]);
        time = ECG_dataset_tm([Time_sample(1):Time_sample(2)])
            ;
subplot(6,1,4)
plot(time,ECG_dataset_signals)
title('3 - 10HZ BPF ECG Signal')
xlim([time(1),time(end)])
                ECG_dataset_signals=ECG_dataset_bpf_d([Time_sample
                    (1):Time_sample(2)]);
```

```
      time = ECG_dataset_tm([Time_sample(1):Time_sample(2)])
         ;
subplot(6,1,5)
plot(time,ECG_dataset_signals)
title('10 - 30HZ BPF ECG Signal')
xlim([time(1),time(end)])
          ECG_dataset_signals=ECG_dataset_bpf_e([Time_sample
              (1):Time_sample(2)]);
      time = ECG_dataset_tm([Time_sample(1):Time_sample(2)])
         ;
subplot(6,1,6)
plot(time,ECG_dataset_signals)
title('30 - 50HZ BPF ECG Signal')
xlabel('Time')
xlim([time(1),time(end)])
clear fs i N Wn_a Wn_b Wn_c Wn_d Wn_e bpf_a_a bpf_b_a
    bpf_c_a bpf_d_a bpf_e_a
clear bpf_a_b bpf_b_b bpf_c_b bpf_d_b bpf_e_b
    ECG_dataset_signals_bpf_ddt
clear ECG_dataset_signals_bpf_ddt_sq
    ECG_dataset_signals_bpf
```