Florida State University Libraries

Electronic Theses, Treatises and Dissertations

The Graduate School

2016

Real-Time Fetal ECG System Design Using Embedded Microprocessors

Harikrishna Muddu

FLORIDA STATE UNIVERSITY
FAMU-FSU COLLEGE OF ENGINEERING

REAL-TIME FETAL ECG SYSTEM DESIGN USING EMBEDDED MICROPROCESSORS

By
HARIKRISHNA MUDDU

A Thesis submitted to the
Department of Electrical & Computer Engineering
in partial fulfillment of the
requirements for the degree of
Master of Science

2016

Harikrishna Muddu defended this thesis on November 22, 2016.

The members of the supervisory committee were:

Uwe H. Meyer-Baese

Professor Directing Thesis

Simon Y. Foo

Committee member

Shonda Bernadin

Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the thesis has been approved in accordance with university requirements.

To my family members,
Vani, Govind, Vamshi, Raghava
And would-be
Swetha

ACKNOWLEDGMENTS

I would like to sincerely thank my academic advisor, Dr. Uwe Meyer-Baese for providing wonderful opportunity to work with him and also a wealth of resources, knowledge and support. My extended thanks go to Dr. Simon Foo and Dr. Shonda Bernadin for their interest to be member of my supervisor committee and valuable suggestions.

Additionally, I thank Dr. John Burkardt, Faculty in Scientific Computing Department, FSU for providing the useful resources online under GNU public license and also Altera corporation and Xilinx Inc. for providing hardware, software and resources under University Programs.

I convey my special thanks to FAMU-FSU college of Engineering and Electrical and Computer Engineering Department faculty and administration, Dr. Simon Foo the chairman of the Department, Dr. Petru Andrei the Graduate program director, Dr. Reginald Perry Associate Dean of college, for providing a chance to work under him as a GA, Mrs. Melissa Jackson for her constant support since very first day, and Mr. Eric Saponneti for helping me in all the paperwork.

I thank my parents, brothers and would-be for their immense support and encouragement throughout my journey, last certainly not the least my friends Sindhuja Sudharanjan, Balaji Srikant and many others, who provided the external help.

TABLE OF CONTENTS

List of Tables	vii
List of Figures	viii
ABSTRACT	ix
CHAPTER 1: INTRODUCTION	1
1.1 Motivation for finding fetal heart rate (FHR)	1
1.2 Brief Overview of ECG	2
1.3 Literature review of studying FECG.....	3
1.3.1 Commonly used clinical practices	3
1.3.2 Signal processing techniques	5
1.3.3 Blind source separation methods	7
1.3.4 Artificial intelligence methods	8
1.3.5 Hardware models	10
1.3.6 Overview of all methods	11
CHAPTER 2: PROPOSED METHODOLOGY	12
2.1 Calculating the covariance matrix	13
2.2 Applying Principle Component Analysis (PCA)	14
2.3 Calculating reconstruction matrix and apply reconstruction.....	15
2.4 Calculating BPM and SNR	18
CHAPTER 3: FPGA BASED MICROPROCESSORS	21
3.1 Embedded systems overview	21
3.2 Introduction to FPGA.....	21
3.3 System on chip on FPGA	24
3.4 Different FPGA processors selected	27
CHAPTER 4: RESULTS AND CONCLUSIONS	31
4.1 Performance data of basic operation	31
4.2 Performance data of ECG algorithm	33

CHAPTER 5: CONCLUSIONS AND FUTURE WORK.....	37
5.1 Conclusion.....	37
5.2 Future work	37
APPENDICES	39
A. EIGENVALUE ALGORITHMS	38
B. C IMPLEMENTATION OF THE ALGORITHM	44
REFERENCES	51
Biographical Sketch	55

LIST OF TABLES

1	Mean values for 5 abdominal lead signals.....	14
2	BPM scores and eigenvalues for the 5 channels configuration according to the rankings	16
3	BPM scores and SNR for 5 channels	20
4	Key features of FPGA prototyping boards used in this project	27
5	FP unit specification comparison.....	29
6	FPGA prototyping board used in the project with different FP options	29
7	Timing data for Nios II for conversion operations with and without FP HW	32
8	Time taken for ECG algorithm for individual steps in case of Nios II/s	33
9	Point frames per second capability of the embedded configuration	35

LIST OF FIGURES

1	The electrical Conduction System of heart (left) ECG pulse (right) [3].....	2
2	ANFIS network [34]	9
3	Overall FPGA System architecture proposed for extraction of FECG by DCT method [6] ...	11
4	Leads taken form abdomen and thorax (above). Ch.1-5 are AECG and Ch.6-8 are ECG waves (below)	13
5	Eigenvalues and eigenvectors of covariance matrix	16
6	ECG plot after reconstruction for 5 channel system for 1000 samples	17
7	ECG of each after performing reconstruction, squaring and smoothening by taking moving average with window 5	19
8	CLB interconnection in FPGA Source: Xilinx Handbook on FPGAs.....	23
9	Xilinx and Altera DE2 FPGA boards respectively	24
10	Nios II SoC Design for the ECG system using bottom up approach.....	25
11	MicroBlaze SoC Design for the ECG system using bottom up approach	26
12	(a) Quartus summary of resource utilization of Nios II for ECG algorithm without FP unit (b) for without FP unit (top). (c) is Slow Model Fmax summary for no FP unit included and (d) is in case of FP unit includes(bottom).....	30
13	Performance data of Floating point operations (FLOPs) using FPGA based microprocessor system using C/C++ GCC compilers with different compiler optimization levels	32
14	Runtime of ECG algorithm for the FPGA-based microprocessor system compiled with C/C++ GNU compiler with optimization –O3 for ARM/MicroBlaze and –O1 for Nios II/s. Eigenvector Jacobi computation by JB	34
15	LCD display of mother and Fetus BPM using Altera DE2 FPGA monitor.....	36

ABSTRACT

Monitoring the fetal ECG (FECG) gives us important information about the fetal wellbeing. FECG is a complex waveform where each of the P through T complexes provides wealth of information. The objective of this project is to develop the best state-of-art real-time FECG monitoring system using embedded microprocessors. Many researchers from various fields like signal processing, artificial intelligence, and advanced statistics, have applied different techniques to extract FECG from the mixture of MECG and other noises and calculate the FHR, with accuracy as an objective. Most of them are calculation intensive and not real-time. The proposed approach focuses mainly on real time processing, robustness and portability of the system. The work discussed here will provide a novel algorithm to extract FECG from abdominal ECG (AECG) which is mixture of FECG, MECG, and noise, and finding Fetal FHR with less number of dimensions (measurements) with the best signal-to-noise ratio. This approach is tested on different soft-core processors and results are compared with other commercial of-the-shelf (COTS) hardcore solutions, in terms of power, cost, size and speed. In the end FECG was successfully extracted and identified on the basis of BPM and SNR values calculated using this method. It was found that hard-core processor (ARM Cortex A9) has achieved the best real-time performance among all.

Keywords: ECG, FECG, AECG, MECG, Eigenvalues, PCA, Nios II, MicroBlaze, ARM, FPGA, Embedded Systems, System on Chip

CHAPTER 1

INTRODUCTION

1.1 Motivation for finding fetal heart rate (FHR)

During the last decades, Fetal Heart Rate (FHR) monitoring has been extensively used for antepartum and intrapartum monitoring to know fetal health condition. Monitoring of fetal heart rate is very important during the pregnancy and labor. One can obtain important information about the fetus growth and also if it is undergoing any distress or injury. By only measuring the FHR, arrhythmia and changes in fetal heart rate can be detected. Cardiac waveform itself is source of valuable diagnostic information. Combination of fetal heart rate and polarity of P wave of Fetal Electrocardiogram (FECG) reveals information about types of supraventricular tachycardia. It is a condition where $FHR > 200$ bpm and narrow QRS complexes [4]. By early monitoring of fetus health conditions, irreversible neurological damages or even fetal death during labor can be prevented. FECG monitoring is important even during labor to understand the dynamics of labor process by knowing cervical dilations and pressure developed due to uterus contractions, also to avoid false-positive tests hence unnecessary surgical intervention.

Typically, the signal strength of the FECG system is in the order of microvolts. The Maternal Electrocardiogram (MECG) signal is at least 10 times stronger. Also the measured signal at the abdomen may be a complex mixture of FECG, MECG, muscle contractions and noise. These aspects make the design of FECG system more challenging. In standard ECG 10-12 measurements are taken across various parts of the body such as 3 bipolar and 3 unipolar between arms and legs, and 6 around the heart. But in the case of pregnant woman, signals taken from thorax may have insignificant contribution towards FECG [2]. The thorax signals may be valuable, in the case of adaptive filtering technique where MECG is taken as reference signal. As Data Identification System (DaISy) demonstrates, even the signals measured from arms and legs also show less contribution towards FECG [5]. Length of the signal measurements in DaISy dataset are 10 seconds and sampled at 250Hz. This means 2500 samples are taken. The data set available online consists of 8 channels: first 5 abdominal lead signals and finally the 3 thorax leads.

1.2 Brief Overview of Electrocardiogram (ECG)

The Electrocardiogram (ECG or EKG) is a skin-surface measurement of the electrical vector generated by the heart muscles with each heartbeat. It is an important clinical routine in diagnose of any cardiac ailment. ECG is simply a representation conduction of ions through heart muscle for every heartbeat. The action potentials generated by the different cells in the myocardium sum up to generate the ECG waveform that can be measured at the skin surface. Our body is considered as a voluminous conductor. Dipole model allows us to view the net charge flowing through myocardium as a vector originating at the center of a uniform sphere. Figure 1 describes the corresponding waves of ECG, as the heart goes through sequence of depolarization and repolarization activities and at the end, all the waves are superimposed to show how ECG wave form is formed [3].

Figure 1. The electrical conduction system of heart (left) ECG pulse (right) [3]

Direction of depolarization is normally towards left due to the leftward orientation of heart in the chest and also greater muscle mass. A fundamental principle of ECG recording is that when the wave of depolarization travels towards a recording lead it results in positive or upward direction, when it travels away from a recording lead this results in a negative or downward deflection. Generally, 10-12 leads are used to record an ECG. Four leads record at limbs and six leads around the chest. Electrical axis of depolarization is $+30^{\circ}$ to -90° hence at least 10 leads are required to cover entire electric field [7]. In a typical ECG waveform electrical activity is

measured in voltages (mV) along y-axis against time (ms) along x-axis. Frequency of occurrence of the QRS complex, gives us useful information of Beats per Minute (BPM).

ECG signal of a healthy human would consist of 70-100 beats per minute. For all medical purposes the bandwidth of ECG is taken from 0.5Hz to 50Hz [3]. However, for FECG we should expect 110-130 bpm and 1.3-3.5 Hz in normal case. As shown in the Figure 1, each ECG wave consists of Pre wave P (80ms), QRS Complex (80-120ms), and post wave T (160ms). Typical altitudes are $P \leq 0.2$ mV; $Q = R/4$; $R = 0.6-2.6$ mV; $S < 0.6$ mV, and $T = R/7$ for the MECG. And in case of FECG, we can consider 0.1 to 0.2 times of the MECG amplitudes.

So we can clearly say that, MECG will have strong interference while measuring FECG. So the task is to extract FECG out of this complex mixture and measured FHR, which should lie in the desired range of 110-130 bpm.

1.3 Literature review of extracting FECG

ECG is a major field of study in biomedical application. In recent years it grabbed the attention of researchers in fields of signal processing, artificial intelligence, and advance statistical mathematics to develop elegant algorithms to accurately filter out FECG from the AECG and measure FHR. Importance of fetal ECG information also became important in recent decades. It is a challenging task to extract FECG as it is entangled in a complex mixture, as above mentioned. An exhaustive study of state-of-art literatures, journals, and papers has been done before embarking with the new idea. A journal published by M. A. Hasan, M. B. I. Reaz, M. I. Ibrahimy, M. S. Hussain and J. Uddin [6] provided an extensive knowledge on various methods and theories to extract FECG from AECG.

1.3.1 Commonly used clinical practices

Most widely used fetal monitoring techniques are categorized into two types. They are *invasive* and *non-invasive* methods. Invasive method of monitoring is done by placing an electrode on scalp of the fetus and directly measuring ECG. This may generate accurate signals. But it has two major drawbacks. 1) This is possible only during labor, hence not much feasible 2) placing electrode probes may tamper the scalp of the fetus. Hence mostly non-invasive methods are used by clinicians. Non-invasive techniques are also known as *Auscultation*. It is a method of

listening to the fetal heartbeat from mother's abdomen. Major techniques are *Doppler ultrasound* and *Electronic monitoring*.

1. Doppler Ultrasound device is small transducer, which operates on the principle of Doppler Effect. This device is pressed against mother's abdomen. These ultrasound waves (2-18 MHz) are reflected by fetus heart. Frequency of reflected waves will vary in accordance with the fetus heartbeat. Changes in the received ultrasound waves are then converted into images.

2. Another method is to use fetoscope, a device similar to stethoscope. Its open end is pressed against mother's abdomen and we can hear heartbeats of fetus, similar to stethoscope

Extended use of ultrasound waves may cause damage to the soft tissues because, some of the ultrasound frequencies are readily absorbed by the body. Hence electronic methods are better options for FECG monitoring. Electronic fetal monitoring is done through special equipment to observe fetal response to the uterus contractions. There are two types of monitoring methods *Internal* and *External* monitoring. These methods can offer to choose comfortable physical position while examining.

1. *Internal Monitoring*. It is an invasive method. Small probes are inserted inside the womb and placed on the scalp of the fetus. As there will be no interference of MECG, this method gives us good quality of image as compared to other methods. However, this may tamper the scalp and lead to bleeding. Though these injuries on the fetus are self-healing in nature, we cannot use this method regularly.

2. *External Monitoring*. In this non-invasive method, probes are placed on the abdomen. This method does not cause any damage to fetus upon prolonged usage. Probes are adhered to abdomen using glue like substance, to make perfect conduction of charges through the skin and electrode interface. It detects the fetus response to uterus contractions. But this measurement is a mixture of muscle contractions, electrical noise, and MECG. And also this system is not suitable for ambulatory purposes.

Measuring Abdominal ECG (AECG) is another approach for monitoring FECG and calculating FHR. These are external non-invasive methods of finding antepartum FHR. Principle behind measuring AECG is similar to the normal ECG, i.e. measuring the body potential of *fetus*

at the surface of maternal abdomen [46]. However, it is immensely difficult to measure the body potential of fetus from outside. It is advantageous to use AECG while extracting FECG, as it can provide much useful information [18] than Doppler Ultrasound method does. In this method, multiple leads are attached to abdomen to measure body potentials. Sometimes thorax ECG is also taken as a reference signal to eliminate MECG to filter out FECG. But this system has two major drawbacks. It has poor Signal-to-Noise ratio. FECG is so weak as compared to MECG that it is often misinterpreted as noise and eliminated or goes undetected. Another issue with this approach is that it is not suitable for mobility.

Therefore, many researchers have been trying to extract FECG using other reliable techniques. Various methods have been carried out to extract FECG and calculate FHR such as subtraction of MECG from AECG, matched filtering, adaptive filtering, orthogonal basic functions, temporal structures, polynomial networks, wavelet transforms and real-time signal processing.

Traditional methods are computationally heavy and do not produce appreciable SNR. Finding the reliable FHR with less complex methods has been a topic of research in many fields such as Artificial Intelligence (AI), advanced mathematic models in Wavelet Transform (WT), Fourier transforms, time-frequency approaches, statistical signal analysis are few examples. AI methods include Artificial Neural Networks, Fuzzy set theory, self-organizing map (SOM) neural network approaches. Some of the methods are described below. These approaches are not purely based on ideas of one particular field. Most of them are multi-disciplinary in nature.

1.3.2 Signal processing techniques

One of the basic techniques is to take MECG as a reference signal and subtracting it from AECG. But this method results in very poor SNR and has serious issues like baseline shift of MECG. As both MECG and FECG are prone to change over the time, some important short duration contractions of fetus may be misinterpreted as noise and they tend to be suppressed. Two methods can be used for finding FHR. They are peak picking and using spectrum analysis after transformation.

In case of peak picking, it is done by scanning through a segment sampled data of FECG and detecting R peaks. Then the average duration of occurrence of R-R waves is calculated. Resultant average will give average FHR. Mainly, the result of the search depends on the

algorithm used and on the local SNR in the above specified data segment. As AECG can vary significantly over the time, local SNR value fluctuates about the SNR value of the entire FECG signal and sometimes might not produce reliable extraction. Therefore, we can expect some peaks may go undetected. Even in the case of spectrum analysis method, pattern may not be repeated at regular intervals. Some of the short time impulses are nullified while using transfer functions. However, transform method still gives an appreciable value of SNR. In order to avoid such unwanted perturbations in SNR, strong signal processing techniques are needed. Some of the signal processing methods for extraction and separation of FECG are discussed here.

It is a pellucid fact that each wave in FECG complex, from P wave through T wave has its own significance in determining fetal health condition. For example, Position of T wave is important to determine the level of oxygen supply to fetus. Hence we have to keep three things mind while extracting FECG. They are Amplitude of different waves, Duration of the wave and FHR. However, because of the noninvasive nature of measurement technique, direct recording of accurate FECG may not be possible. Often P and T waves are suppressed or hidden. Various filtering techniques such as linear filter, average filter are applied to analyze the spectrum of AECG. But these techniques are not efficient enough to separate seemingly alike three spectral components i.e. MECG, FECG and noise. Some efficient signal processing techniques are discussed below to extract FECG.

Wavelet Transform. Wavelet transform (WT) is similar to the Fourier transform (or much more to the windowed Fourier transform) with a completely different merit function. This transform uses functions that are localized in both real and Fourier space. It is an efficient mathematical tool to analyze fast varying signals. The WT technique is one of the ways to extract FECG from AECG.

In 1996, J.C Echeverria et al. developed a procedure using WT and pattern matching techniques, to eliminate MQRS and retain FECG from AECG. Mathematically this is represented as [19]

$$s(t) = r(t)[f(t)+m(t)+n(t)] \quad (1)$$

where $f(t)$ represents fetus signal, $m(t)$ represent mother signal and $n(t)$ is the Gaussian noise and $r(t)$ [20] is the modulation factor. This method involves two stages: first stage is to suppress the

unwanted frequencies (low and high) frequencies (by squaring and using appropriate filtering). Second stage is to detect and eliminate maternal QRS from the complex and extract FECG. Maternal QRS complexes are recognized using pattern matching method. In order to preserve the homogeneity of maternal and fetal ECG signals, unwanted detail signals [21] are filtered out from AECG using wavelet multiresolution decomposition [22]. Pattern matching has more robustness due to its specificity of local patterns and preserving the local SNR unaffected. Another approach is proposed by Ye Datian and Ouyang Xuemei [23] to detect and eliminate distorted MECG from AECG and extract FECG. Yet it was challenging to accurately detect FECG because it is enhanced only under certain scale factor of wavelet base function.

Another mathematical model is to monitor FECG by applying Lipschitz exponent [24] analysis. Its main drawback was this system cannot detect FECG if it is hidden behind or strongly interfered with MECG. This method has failed to detect 3 out of every 10 fetal QRS signals, which unacceptable. F. Mochimaru and Y. Fujimoto [25] used Wavelet transform based method to remove baseline shifting and suppress the noise. They used Multiresolution Analysis (MRA) on the raw ECG data, with 12 levels of decomposition and Daubechies20 wavelet function.

1.3.3 Blind source separation methods

It is an efficient method to separate FECG from AECG by eliminating MECG. This method is applied when the source is concealed in the complex mixture of noise and other desired signal. Extracting an FECG can be readily stated as a Blind Source separation (BSS) problem where source is AECG and reference signal is MECG. Two advanced mathematical models used in this method are Principle Component Analysis (PCA) and Independent Component Analysis (ICA) [26]. In conjunction with other methods like Linear adaptive filtering, Singular Value Decomposition (SVD), BSS works very efficiently. Resultant SNR is considerably high as compared to the others. As both the methods are dealing with the dataset, where all record taken by abdominal leads are statistically independent from each other. Hence sudden variation in the AECG signal may still be accounted. Lieven De Lathauwer et al. [48] proposed an innovative method to extract FECG, based on ICA technique. Multi lead potential recordings are taken from abdomen and applied ICA on the statistically independent values of multi lead records. This technique extracts the most relevant features the multidimensional measurements.

D.E. Marossero et al. [28] had a theory for separating FECG based on ICA technique. They used an entropy function called Renyi's Mutual Information (Mermaid) in information theory, to perform ICA. Simulations of this method show that this method is effective particularly when long data sets are used (with higher dimensional matrixes). But computationally it is very complex [29]. Ping Gao et al. [48] applied a combined method of singular value decomposition (SVD) and ICA to separate FECG from the mixture. They applied BSS using an SVD of the spectrogram and repetitive use of ICA on both spectral and temporal representation of AECG. SVD contributed towards the separation of each component and ICA contributed towards independence of components from mixture.

In 2004, M. Burghoff and P. Van Leeuwen had a theory to identify even the twin pregnancy using ICA. Recently Farshid Soheili Najafabadi [31] proposed a theory for isolating FECG and MECG from abdominal signal using ICA. They stated that, it can work well even if AECG has $\text{SNR} = -200\text{dB}$. However, they used simulated data for the study and accuracy of the method depended on quantization noise.

Principle Component Analysis is another BSS technique to identify and separate FECG from the mixture. Sdaya Redif, Umut Fahrioglu developed a novel approach to extract FECG using PCA technique [30]. They used applied PCA through Eigenvalue decomposition method (EVD) to polynomial matrices. Isolation is achieved by projecting the sampled data on to the orthogonal fetal space thereby suppressing maternal and noise signals. M.P.S. Chawla proposed a theory by combining ICA-PCA to extract FECG. Its efficiency is verified on 6 channel and 12 channel ECG data [42]. This technique concluded that, based on the combinations of leads chosen, better feature selection can be achieved using ICA, and better denoising can be achieved through PCA. Hence noise can be removed from AECG using PCA and followed by extraction of independent components (IC) using ICA method.

1.3.4 Artificial intelligence methods

Artificial Intelligence (AI) techniques may also be used for the extraction of FECG. During recent years AI approaches such as neural networks, deep learning, and fuzzy set theory combined with other tools like signal processing and linear adaptive filtering are widely used in biomedical applications. FECG is one such important area it can apply on.

In 1994, John Liszka-Hackzell [32] developed a back propagation network and SOM network to classify FHR. Results obtained with this method were in agreement with the manual classification. K. Assaleh [33] proposed a method combining both ANN and Fuzzy set theory. He named it as Adaptive Neuro-Fuzzy Inference Systems (ANFIS). It had 4 layers of neural network (Figure 2) in which 2 of them were adaptive nodes and 2 nodes are stationary. The multilayer network is trained with the desired values of ECG using back propagation method and minimizing the errors using recursive least square error method. Fuzzy based rule-set is deployed for classification at the end. This system was shown to be working well even in the case where maternal-to-fetal SNR is below -20 dB.

Figure 2. ANFIS network [34]

R. Srinivasan and S. Selvan [34] used adaptive noise cancellation and adaptive signal enhancer filters to process FECG by using neural network. Network is initially trained using real-time recurrent learning algorithm, until the mean square error converged to a lower value. This network was suitable for real-time training data set. The proposed system performed better than noise canceller alone or cascaded combination of both noise canceller and signal enhancer. Mamun Bin Ibne and Lee Sze Wei et al. [35] came up with a theory to suppress MECG from AECG to extract FECG by subtraction method. They used adaptive linear neural network for detecting MECG pattern. This network was initially trained to emulate MECG as closely as possible. Thus it can detect MECG out of AECG and subtract from AECG. The proposed method is better than direct noise cancellation filters because subtracted signal closely mimics MECG.

1.3.5 Hardware models

Because of the importance of FECG extraction in medical field, there are many Application Specific Integrated Circuits for biomedical applications were designed over the years to expedite real-time processing of streaming data while maintaining a balance between accuracy and complexity. Over the years Microprocessors have been improving their clock speeds and bus widths, which will help in quicker monitoring of FHR. As early in 1977 Wen C.Len et al. [35] designed a microprocessor based system that continuously acquires signal from external fetal recording setup and stored into a buffer and simultaneously values are processed in the processor to evaluate cervical dilation, maternal intrauterine pressure and FHR. This system was designed with an intention to help clinicians to gain better understanding of the ongoing dynamics of labor through monitoring intrauterine pressures.

However, this system took significantly longer to design and implement. Also hardware resource limitations hinder the performance. So in recent decades, Field Programmable Gate Arrays (FPGA) became more attractive platform for rapid prototyping of digital systems to implement almost any type of above mentioned algorithms i.e. wavelet transforms, ANN, signal processing techniques etc., on FPGAs. B.S Pimentel et al. [36] implemented ECG signals digital compression on FPGAs, based on Discrete Cosine Transform (DCT). The proposed system focused on real-time compression of ECG signal. To achieve the real-time sensing of FECG, it required two entry buffers, as shown in Figure 3: One buffer for storing the incoming data and another for performing arithmetic operations. It is obvious that, second buffer should process the values, before the first one becomes full.

To keep the design simple, incoming data is always stored in RAM module, thus it always has the latest data. DCT component (second buffer) is stored in ROM module. Later multipliers and accumulators are used to multiply and add the results. At the end results are stored in another RAM module. These are the transformed signal values.

Charayaphan Charoensak and Farook Sattar [43] attempted to design a real-time FECG monitoring system using BSS method. They chose FPGA as preferred platform for rapid prototyping and reduced cost. FPGAs offered much flexibility for designing the hardware system in terms of ample gate count and minimal clock speed. M.A. Hassan, M.I. Ibrahimy, Mamun B.I. Reaz, D. J. Uddin and Md S. Hussain [38] used FPGA platform to implemented Neural

Networks (NN) on Altera's Stratix II EP2S15F484C3 to extract FECG. It also aimed to perform the extraction in real-time.

Figure 3. Overall FPGA System architecture proposed for extraction of FECG by DCT method [6]

1.3.6 Overview of all methods

Extensive study of these proposed all methodologies reveal the history of fetal heart rate monitoring, routed back to decades. Majority of the proposed systems are non-invasive methods. All of the above mentioned theories and techniques focused on some important aspects such as reliability i.e. producing higher signal-to-noise ratio, reducing computational complexity, and best eliminate MECG. But only few algorithms emphasized on the necessity of real-time monitoring. It is also important to make the system ambulatory and power efficient to satisfy today's requirement of portability. The proposed hardware system in this research aims to fill the gaps between drawbacks in previous systems and current requirements.

CHAPTER 2

PROPOSED METHODOLOGY

It is very challenging to measure MECG and FECG separately from AECG as they are superimposed. Hence we have to apply a technique to isolate individual components to exactly measure FECG. Many techniques evolved over last few decades to suppress MECG and noise. But as the signal strength of FECG is low as compared MECG, which is almost in the order of noise. Sometimes when we attempt to remove noise, there may be a chance of losing valuable information contained in FECG as well.

The proposed system uses Blind Source Separation (BSS) method to isolate desired FECG signal from the total mixture. Principle Component Analysis (PCA) is one of the BSS techniques. Accuracy of the method is based on the score obtained by individual component. The steps below describe the whole algorithm;

1. Calculate the DC of input
2. Remove DC part
3. Calculate the Covariance matrix
4. Apply PCA on the covariance matrix
5. Calculate reconstruction matrix and apply reconstruction
6. Calculate BPM and SNR

Matlab code for the proposed methodology is

```
s = sum(x);          %%%% 1) Calculate DC of input
x = x-s/length(x);  %%%% 2) remove DC part
Cxx = cov(x', 1);    %%%% 3) Calculate the covariance matrix
[E, D] = eig(Cxx);   %%%% 4) Eigenvector and Eigenvalues
V=D^-.5 * E';        %%%% 5) Compute reconstruction matrix and
z=V*x;               %%%% apply reconstruction
bpm(z)               %%%% 6) Calculate the BPM and S/N
```

2.1 Calculating the covariance matrix

To make the system realistic, 5 channel data of DaISy data set is taken, with the duration of 1.5s signal at 250Hz sampling frequency (i.e. total of 375 samples). Analysis shows that 1.5s signal duration is sufficient to achieve reliable results of FHR. This data is stored in the ROM of FPGA. Computation of covariance matrix requires DC free input data. To make the input DC free, average of the sample range is taken for all channels. Later the average is subtracted from each original sample value, for all channels. Mean values of all the channels are shown in the below Table 1.

Figure 4. Leads taken form abdomen and thorax (above). Ch.1-5 are AECEG and Ch.6-8 are ECG waves (below)

Table 1. Mean values for 5 abdominal lead signals

Channel	1	2	3	4	5
Mean	0.128	-0.571	0.379	0.700	0.442

First step towards calculation of covariance matrix is reading the data from buffer. Depending upon the hardware configuration, data is read from the buffer through Direct Memory Access (DMA) or burst read from a FIFO, via periodic polling of peripherals or HW interrupt.

Now covariance matrix is computed using the inner product of the DC free input data of all the channels.

After the calculations Covariance matrix C_{xx} is found out to be

$$C_{xx} = E \{(\mathbf{x}-\boldsymbol{\eta})^T(\mathbf{x}-\boldsymbol{\eta})\} = \begin{bmatrix} 91.127 & -70.962 & 75.963 & -21.56 & 104.376 \\ -70.962 & 328.179 & -195.476 & -46.503 & -259.359 \\ 75.963 & -195.476 & 140.807 & 5.961 & 178.274 \\ -21.56 & -46.503 & 5.961 & 40.147 & 17.779 \\ 104.376 & -259.359 & 178.274 & 17.779 & 241.856 \end{bmatrix} \dots\dots\dots (2)$$

Where $\mathbf{x}$ is the input value and $\boldsymbol{\eta}$ is the mean. As it is the result of inner product of a data set, it is a symmetric matrix.

Covariance matrix describes the variation of one variable along the direction of other. All the diagonal elements of covariance matrix are the variance of one variable with itself whereas the off diagonals are variance of one variable with others. To reduce the noise in the system, it is highly desirable to minimize the covariance of an element in other dimensions. This is achieved by diagonalizing the covariance matrix.

2.2 Applying Principle Component Analysis (PCA)

Principle Component Analysis also known as Karhunen-Loeve transform, is a statistical process to analyze the complex data which involves higher number of measurement types and each of type has large number of records. The measurements taken may not always be meaningful and best represent the dynamics of the system. So to extract the better meaning out of data samples or in other words, to understand the relation between different measurement

types, we have to eliminate the redundancies and irrelevant data (noise). PCA is one of the most efficient black box techniques in this regard [44].

Mathematically speaking, PCA will reduce the number of dimensions from given data, and gives meaningful representation of it. We can also say that the change of the basis for the input data C_{xx} , in the dimensions where its measurement types are strongly correlated.

Let $\mathbf{X}$ be the data matrix of order $\mathbf{m} \times \mathbf{n}$ and $\mathbf{Y}$ ($\mathbf{n} \times \mathbf{n}$) is the desired meaningful matrix where $\mathbf{m}$ is the number of measurements and $\mathbf{n}$ is the number of records. The change of basis is done using a transformation matrix $\mathbf{P}$.

$$\mathbf{PX} = \mathbf{Y}$$

The transformation matrix is so chosen that the correlation of data variables is maximized after the transformation in the dimensions $\mathbf{r}$, where $\mathbf{r} \leq \mathbf{m}$ is the rank of the matrix $\mathbf{X}$. This theorem proves that the eigenvectors of $\mathbf{X}$ are the best choice to do the desired job. Hence the $\mathbf{P}$ matrix is of the order $\mathbf{n} \times \mathbf{m}$ where each column of $\mathbf{P}$ is the eigenvector of covariance matrix of $\mathbf{X}$.

According to the theory of Karhunen-Loeve transform, best choice of transformation matrix $\mathbf{P}$ would be the eigenvectors of covariance matrix of input data $\mathbf{X}$. As the covariance matrix is symmetric, its eigenvalues are real. There exist several algorithms to calculate the eigenvalues and eigenvectors. Jacobi method of finding eigenvalues and eigenvectors is used here, with the modification suggested by John Burkart [38]. It is based on two modifications for the algorithm suggested by Rutishauser [39], which is basic for C/C++ implementation of `Jacobi()` in the Numerical Recipes Collection [40]. Detailed description is mentioned in the Appendix A. A screenshot of calculation result of eigenvalues and eigenvectors are shown in Figure 5. Eigenvalues are arranged in the ascending order.

2.3 Calculating reconstruction matrix and apply reconstruction

After the calculation of eigenvalues and eigenvectors, the most significant eigenvectors are taken on the basis of ratio of successive eigenvalues. If the ratio is appreciable, then corresponding eigenvectors can be considered towards reconstruction. Table 2 shows the eigenvalues and their respective Beats Per Minute (BPM) arranged in ascending order.

Figure 5. Eigenvalues and eigenvectors of covariance matrix

Table 2. BPM scores and eigenvalues for the 5 channels configuration according to the rankings

Channel	1	2	3	4	5
BPM	81.96	82.4	129.3	131.57	187.5
Eigenvalues	713.305	103.137	18.73	4.131	2.812

It can be clearly seen that only first 3 values have appreciable ratio with their successive eigenvector. They are the principle components of this data set, which contains most useful information, mother and fetal heart rates in our case. Hence they are only considered towards further process. By reducing the number of eigenvectors used for reconstruction, we have achieved the objective of dimension reduction in the original data set. Though eliminating some channels while reconstruction may cause loss some of the data contained in those respective dimensions. But they do not contribute much towards fetal heart rate. Hence it is affordable to loose such data. But here all 5 channels are considered for reconstruction as extracting FECG is the main motive here.

Figure 6. ECG plot after reconstruction for 5 channel system

Reconstruction is done by pre multiplying the Feature matrix, with the data adjust matrix. Feature matrix is constructed by taking the matrix with eigenvectors (of the corresponding eigenvalues selected towards reconstruction) written in columns and taking its transpose. Data adjust matrix is the data set constructed by subtracting the mean from the original data set, which is of the order 5×375 . Resultant output matrix is of the dimension 5×375 . Figure 6 shows the ECG signal for each PC for 5 channel system after reconstruction. Below equations explain the reconstruction step.

```
V=D^-.5 * E';          %%% 5) Compute reconstruction matrix and
z=V*x;                  %%% apply reconstruction
```

Mathematically speaking, all the eigenvectors are orthogonal to each other. By performing the principle component analysis transform, the axes of the original data set are rotated towards the axes of eigenvectors which are the principle components. By projecting data on rotated axes in the direction of the principle components, the variance along that direction is maximized. And the axes which have less number of data projections have been omitted at the beginning of reconstruction. Choice of number of principle components taken towards dimension reduction is

mostly based on the experience. The product of Feature matrix and data adjust matrix is called reconstruction matrix, which is further used for calculating Beats per Minute.

2.4 Calculating BPM and SNR

Beats per minute (BPM) is the rate of appearance of QRS complex in a given time. After the calculation of data adjust matrix as mentioned in the above section, all the values are plotted on the new rotated axis and count for the QRS complex shapes. This peak picking process may seem to be easy. But computationally it is very tricky for three reasons.

1. Pre wave P and post wave T complexes (local maxima) may be mistakenly interpreted as QRS complex
2. Any cross channel talk due to imperfect separation
3. Reconstructed ECG signal may produce zigzag QRS complex and may be missed in peak picking

To identify the exact peaks, we have to mitigate the noise component and define the threshold on the basis of global maxima. Proposed steps describe the robust procedure to measure BPM and SNR in this algorithm, we have used 50% of global maximum to be the threshold value to identify the QRS peaks. Figure 7 shows the ECG signal of each PC after reconstruction, squaring and taking moving average with window 5. Below steps describe the procedure to calculate BPM and SNR.

1. Square each PC ECG signal to remove polarity of waves and minimize the noise component
2. Moving average for 20 ms or 5 samples is taken to remove multiple maxima and irregularities of waveform
3. Find maxim in the window and define the threshold of 50% of global maxima $\max_{50\%}$
4. Search through data and find first maximum by looking for first value $v_1 \geq \max_{50\%}$ at time t_1 with positive slope
5. Inspect data further and find second maxima (t_2, v_2) with $v_2 \geq \max_{50\%}$
6. Compute BPM for each PC by using formula $60 \cdot F_s / (t_2 - t_1)$
7. Calculate SNR: Signal power is assumed to be spanned for 10 samples (40 ms). Hence accumulating samples in windows $[t_1, t_1 + 40 \text{ ms}]$ and $[t_2 - 40, t_2]$ will give signal strength

8. (S) and samples in the remaining part of spectrum represents noise (N). $10.0 \cdot \log_{10}(S/N)$ gives SNR

Figure 7. ECG of each after performing reconstruction, squaring and smoothening by taking moving average with window 5.

Signal to noise ratio (SNR) is a good measure of robustness the system. Higher SNR indicates that the system is more reliable and robust and vice a versa. It is the measure of invariance of signal strength of the system to the noise in the system. Generally, noise may be due to thermal variations of the system, electrical noise, and external. In this particular case of finding the fetus heart rate, mother ECG, noise may be due to probes when they are in contact with the skin, muscle contractions of the mother, and other electrical noises present in the leads. Variation of temperature does not much affect ECG signals. Added feature of this proposed system is that it

gives better SNR while balancing the reduced computational complexity, as compared to the other methods. Table 3 shows SNR for each channel.

Table 3. BPM scores and SNR for 5 channels

PC	1	2	3	4	5
BPM	81.96	82.4	129.3	131.57	187.5
SNR	14.51	6.38	11.95	-3.16	-3.71
Confidence Level	high	medium	high	low	low

The C code for whole algorithm is described in Appendix B. BPM information from Table 3 for PC 1 and 3 indicate they are in the range of mother ECG while PC 3 and 4 are in the range of fetus. And from SNR information first 3 channels have good confidence level. Based on both observations, it can be concluded that PC 3 is representing the closest features of FECG after extraction. Hence FECG extraction is achieved. However, these results are specific to a particular positioning of electrodes. SNR may vary little but BPM would still be in the same range.

This algorithm is tested by taking different chunks of the data (e.g. 376-750, 751-1125, 1001-1375 of each 375 samples and 1001-1500 and 2001-2500 of each 500 samples) and changing the order of channel numbers (i.e. 23451, 34512 and 54321 instead of 12345). Results show that algorithm extracted the FECG within the similar ranges of BPM and confidence scores (8-12 dB) as above. With these results it can be stated that the algorithm is robust and independent of attributes like segment of dataset chosen and different combinations of channel numbers.

CHAPTER 3

FPGA BASED MICROPROCESSORS

3.1 Embedded systems overview

Embedded Systems (ES) are microprocessor based systems that not necessarily look like a computer from the outside, i.e. may not have a keyboard, monitor etc. as usual computer. They usually perform more specialized tasks. Application of embedded system spread over wide range of areas from domestic appliances such as toasters, microwave ovens to heavy machineries such as airplanes and defense systems [8]. Most popular form of embedded systems is microprocessor. Firmware engineering is an emerging area of interest. It is about the interfacing software with hardware to make the peripherals work according the desired functionality. Embedded systems became a platform to realize this achievement. As there are various kinds of processors available to serve different purposes, the choice of embedded system really depends upon the application. Typical requirements of any ideal embedded system should have

- Number of I/O ports/devices which are compatible with wide range of sensors, Displays, storage devices, clock and interrupts etc.
- Computing platforms such as COTS microprocessor boards, FPGA boards, or FPGA microprocessors on chip
- Software tools used to build programs real-time non real-time operating system environments
- IP cores and Board Support Packages to perform the wide range of tasks on System on Chip design configuration
- Debug and I/O options i.e., board support switches, buttons, I2C, JTAG, CAN, USB, GTX receiver, and 10/100/1000 Ethernet.
- memory and data management via EEPROM, DDR3 SDRAM, or QSPI flash

3.2 Introduction to FPGA

Field Programmable Gate Arrays (FPGA) are collection of field configurable logic blocks called Logical Elements (LEs), which can be programmed to perform any desired functionality.

The term *Field Programmable* depicts the fact that these devices can be configured even after installing in the field [8] and the term *Gate Arrays* signifies that logical elements are arranged in a two dimensional array which are capable of performing primitive Boolean operations. Logical elements are also called Configurable Logic Blocks (CLBs). These CLBs are interconnected to each other by switch matrix as shown in the Figure 8. Modern FPGAs contain more than 35,000 of LEs and some of these are dedicated for particular operations like embedded multipliers. These CLBs are composed of Look Up Tables (LUTs), multiplexers and flip-flops.

For a specific application, ASICs and ASSPs can offer better speed than FPGAs with lower power consumption because of their compact design. However, they lack the flexibility to modify the design and functionality after deploying them in the field. This very feature of *flexibility* makes FPGAs more powerful and promising for rapid prototyping, to meet the ever changing nature of the requirements. Choosing FPGAs can offer the designer more selectivity and less risk to the development scheduling as there is no need to re-implement the whole system again to make fewer design changes. Modern FPGAs are comprised of IP (Intellectual Properties) cores. IP cores are the commonly used design blocks. These IP cores provide relaxation for digital designers from the overhead of implementing the same blocks otherwise which would need to configure thousands of LEs to perform the same task, every time. Most commonly used IP cores are clock, SDRAM, interrupts, flash memory, switches, hardware accelerators etc.

There is a trade-off while choosing only an FPGA, over ASIC or microcontroller. FPGA does not have any driver ecosystem that microprocessor architectures and OSs do [9]. This limitation is due to the volatile memory of EEPROMs. LEs inside FPGA do not do anything exclusively, unless we program them to do specific task.

This problem has been addressed by integrating the FPGA device with a microprocessor, making it a heterogeneous architecture even more flexible to connect with I/O peripherals. In recent decades, strong software development environments have emerged and strongly coupled with the hardware. These software tools can compile and generate bit stream files, which can then be loaded in FPGA devices and can convert the graphical blocks into digital hardware circuitry. Another important criterion while choosing an FPGA is, compatibility of third party IP

cores with the system and good integration of the subsystems like System on Chip implementation capability and extensive documentation of all features.

Modern FPGAs are good starting point for rapid prototyping of embedded systems. They are replacing *Application Specific Integrated Circuits* (ASICs) and *Application Specific Standard Products* (ASSPs) for designing, implementing and verifying the digital designs. FPGAs offer more flexibility and reduced risk, decreased cost of ownership of the product and reduced time to market, as compared to ASICs and ASSPs. Functionality of the ASICs can be verified only after the deployment of the chip in the field and there is no scope for improvement after the end product has been developed. Whereas improvements in the design can be reflected immediately using FPGA, without changing much of the hardware.

Figure 8. CLB interconnection in FPGA
Source: Xilinx Handbook on FPGAs

Nowadays FPGAs are extensively used in the areas of real-time signal processing applications and high speed systems. FPGAs enjoy many attributes of ASICs such as security against illegal copies and better interfacing with peripherals [8]. Modern FPGA platforms can support many interfaces such as HDMI (video in, video out), USB-UART, Trimode

Ethernet(10/100/1000), Audio Codec, LCD, VGA, Debounce logic, switches, DDR2 SDRAM, and serial port connector etc.

Figure 9. Xilinx and Altera DE2 FPGA boards respectively

3.3 System on chip (SoC) on FPGA

FPGAs are generally, a part of whole system implementation. They are growing enormously in recent years, architectures have evolved, so that the whole digital system can be built on an FPGA, making it a System on Chip configuration (SoC). SoC may contain standard

IP cores and third party IP core interfacing with I/O peripherals and strongly integrated with CPUs. SoCs became emerging trend in many academic and industrial fields such as autonomous driving, high speed data communications, 3D image processing, and high performance computing. They offer

- Reduction in the system power, cost, and board space by integrating two chips the processor and FPGA into one,
- Improving system performance via high-throughput datapaths between the Hard Processor System (HPS) and the FPGA,
- Mainly differentiating the end product by customizing in both hardware and software [10].

There are two kinds of approaches for building the SoC. They are *Bottom up* and *Top down* approaches. The bottom up approach is basically building the entire system from scratch. Initially the system requirements and functionalities are defined. Then based on the functionality and availability of IPs, it is only needed to write HDL code for any blocks that are not already available as IP [8].

System Contents									
Address Map Clock Settings Project Settings Instance Parameters System Inspector HDL Example Generation									
	Use	C...	Name	Description	Export	Clock	Base	End	IRQ
	☒		clk_0	Clock Source					
	☒		nios2_qsys_0_mhs	Nios II Processor		clocks_sys...	0x0000_1800	0x0000_1fff	
	☒		Floating_point_Copro...	Floating Point Hardware			Opcode 252	Opcode 255	
	☒		onchip_memory2_0	On-Chip Memory (RAM or ROM)		clocks_sys...	0x0000_0000	0x0000_0fff	
	☒		switches_mhs	PIO (Parallel IO)		clocks_sys...	0x0000_2020	0x0000_202f	
	☒		LEDs_mhs	PIO (Parallel IO)		clocks_sys...	0x0000_2030	0x0000_203f	
	☒		jtag_uart_0_mhs	JTAG UART		clocks_sys...	0x0000_2040	0x0000_2047	
	☒		sdram	SDRAM Controller		clocks_sys...	0x0100_0000	0x017f_ffff	
	☒		clocks	Clock Signals for DE-series Board Peripherals		clk_0			
	☒		character_lcd_0	16x2 Character Display		clocks_sys...	0x0000_2048	0x0000_2049	
	☒		timer_0	Interval Timer		clocks_sys...	0x0000_2000	0x0000_201f	
	☒		up_clocks_0	Clock Signals for DE-series Board Peripherals		clocks_sys...			

Figure 10. Nios II SoC Design for the ECG system using bottom up approach

Figure 10 shown above depicts the System on Chip implementation of fetal heart rate system on Nios II SoC, built through bottom up approach using Qsys Tool. In the bottom up method generally we start with selecting microprocessor and adding the components that are

needed. This requires good experience of designing systems with various bus widths, clocks, interrupts, and memory modules etc.

Relatively *easier* approach as compared to the earlier one is Top down approach. This is easier in the sense that, initially all the modules available in the system are listed down and we need to simply select and remove the items which we don't need. Top down approach also gives us an idea of what all resources are provided by the vendor. This approach is also relatively less prone to the designing errors hence less time can be devoted towards debugging which reduces design time. This approach may also mitigate the errors like incompatible bus widths, and improper connection among the components when multiple clock sources are used. It is the designer's choice to use any particular method. Figure 11 shown below is the example of Top down approach of building SoC on MicroBlaze processor using Xilinx EDK tool.

Figure 11. MicroBlaze SoC Design for the ECG system using bottom up approach

3.4 Different FPGA processors selected

As it is mentioned earlier FPGA boards are good starting point for rapid prototyping of embedded systems. Here fetal ECG system is designed using 3 different FPGA development boards. They are mentioned in the below table.

Table 4. Key features of FPGA prototyping boards used in this project

Board Name	TERASIC DE2	DIGILENT ATLYS	ZYBO
FPGA Device	Cyclone II 2C35	Spartan-6 LX45	Artix-7 Z7010
Architecture type /Instruction set	ISA/RISC	ISA/RISC	ARMv7/RISC
Microprocessor	Nios II e/s/f	MicroBlaze	ARM Dual core Cortex A9
DMIPS	50	68-125	4000
Switches/Buttons/LEDs	18/4/27	8/6/8	4/8/5
Display options	LCD/VGA/TV Decoder	VGA/HDMI	VGA/HDMI
A/D options	24bit 2Ch. CODEC	2Ch. CODEC	CODEC and 6 Pmod

TERASIC DE2 board is uses Nios II embedded processor. As shown in the above Table, it can be configured in 3 processor cores i.e. e/s/f which stands for economy, standard and fast processor cores respectively [45].

- Nios II/e – The noise II/e economy core is designed to achieve smaller possible core size with the limited features
- Nios II/s – This core offers smaller design sizes while maintaining the performance
- Nios II/f – This core offers the fastest performance among all and provides all available feature set

As it can be seen in the SoC design of Nios II, it has a flexibility to couple or decouple the hardware accelerator or *Floating Point Unit* which can handle the IEEE 754 format single

precision floating point operations. Terasic DE2 FPGA board offers only basic arithmetic operations such as addition, subtraction, multiplication and division. Terasic DE2-115 board offers additional operations such as square root, FP to Integer and integer to FP conversions, and double precision floating point arithmetic operations.

MicroBlaze also has the facility to include hardware accelerator unit. It has a different configuration as compared to Nios II. MicroBlaze FP unit has 3 modes of configuration they are; ‘None’ mode with no FP support, ‘Basic’ mode with standard FP operations, ‘Extended’ mode which supports conversion and square root operations. Digilent ZYBO board contains ARM Cortex-A9 processor tightly integrated with Xilinx 7-series FPGA device [11]. This FPGA device shares many features with Artix-7 family. ZYNQ AP (Application Processor) SoC devices are designed around the Cortex A9 dual core. Programmable I/O and Programmable Logic acts as slaves which are configured with JTAG [12]. Programmable System (PS) contains dual core Cortex A9 processors, tightly coupled with 64-bit FP units and NEON media processing units. Total Application Processing Unit is available as a hard core in AP SoC design [13]. FP unit in ARM processor is capable of single precision and double precision (64-bit) arithmetic operations and conversion from integer to floating point and comparison. SoC design using ZYBO would be similar to that of ATLYS, as the GUI for both ZYNQ Z7010 and ATLYS Spartan-6 is same i.e. ISE WebPack Xilinx EDK tool.

All the above mentioned processors share a common feature which is FP units in all microprocessors are of 32 bits, except ARM Cortex A9 which has 64-bits. Eigenvalue and Eigenvectors are so computationally intensive such that regular 8 and 16 bit processors will slow down the whole system, and cannot serve *Real-Time* operations. In spite of using 32 bit FP units, some 64-bit operations such as `sqrtf()` and `log10()` may delay the calculations. The real metric for the performance of microprocessor for any commercial application would be, time taken to complete a given task. In order to measure the times elapsed for the whole algorithm as well as for individual steps are measured using timers. These timers are available as IP cores in both Altera and Xilinx platforms, mentioned as `timer_0` in DE2 SoC design and `AXI_TIMER` in ATLYS and ZYBO designs respectively. Table explains the integrity and operations supported by FP units with their respective microprocessors.

Table 5. FP unit specification comparison

Board Name	DE2	DE2-115	ATLYS	ZYBO
Microprocessor	Nios II	Nios II	MicroBlaze	ARM Cortex A9
FPU width (bits)	32	32	32	64
Integration	Loosely	Loosely	Loosely	Tightly
Arithmetic Ops	basic	Basic, Extended	None, Basic, extended	All

Improvement in the performance is achieved at the cost of using additional resources as FP units are included in the design. Addition of FP HW may increase the silicon footprint: however, the overall energy consumption of the system will be reduced. Table 6 shows some of the extra resources associated with inclusion of FP unit with a permissible increase of 13-21% of silicon for enhanced speed.

Table 6. FPGA prototyping board used in the project with different FP options

Microprocessor	Nios II/s	Nios II/s	MicroBlaze	MicroBlaze	MicroBlaze
FP support	No	Yes	No	Yes	Yes
Speed/ (MHz)	76.90	87.38	102.523	101.792	102.955
Embedded Multipliers	4	11	3	8	8
Les/LUTs no FP	3688	9670	5503	6259	6693
BRAM memory(KB)	23	38	18	18	18
I/D Cache	4KB/None	4KB/None	8KB/8KB	8KB/8KB	8KB/8KB

Figure 12 shown below is the Quartus II 13.0 simulation summary of SoC design using Nios II/s processor. Figure 12 (a) is the summary when there is no FP unit added (b) is summary of resource utilization when FP unit is added (c) Slow max Fmax summary in case of no FP unit added and the (d) is when FP unit is included. Clearly there is an improvement in Fmax but at the cost of using extra resources of FPGA.

Flow Summary	
Flow Status	Successful - Sat Mar 05 16:17:59 2016
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	DE2_Media_Computer
Top-level Entity Name	DE2_Media_Computer
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	4,413 / 33,216 (13 %)
Total combinational functions	3,952 / 33,216 (12 %)
Dedicated logic registers	2,863 / 33,216 (9 %)
Total registers	2952
Total pins	69 / 475 (15 %)
Total virtual pins	0
Total memory bits	97,664 / 483,840 (20 %)
Embedded Multiplier 9-bit elements	4 / 70 (6 %)
Total PLLs	1 / 4 (25 %)

(a)

Flow Summary	
Flow Status	Successful - Sat Mar 05 17:12:36 2016
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	DE2_Media_Computer
Top-level Entity Name	DE2_Media_Computer
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	10,459 / 33,216 (31 %)
Total combinational functions	9,206 / 33,216 (28 %)
Dedicated logic registers	6,729 / 33,216 (20 %)
Total registers	6818
Total pins	69 / 475 (15 %)
Total virtual pins	0
Total memory bits	98,745 / 483,840 (20 %)
Embedded Multiplier 9-bit elements	11 / 70 (16 %)
Total PLLs	1 / 4 (25 %)

(b)

Slow Model Fmax Summary					
	Fmax	Restricted Fmax	Clock Name	Note	
1	76.96 MHz	76.96 MHz	CLOCK_50		

(c)

Slow Model Fmax Summary					
	Fmax	Restricted Fmax	Clock Name	Note	
1	87.25 MHz	87.25 MHz	CLOCK_50		

(d)

Figure 12. (a) Quartus summary of resource utilization of Nios II for ECG algorithm without FP unit (b) for without FP unit (c) is Slow Model Fmax summary for no FP unit included and (d) is in case of FP unit includes(bottom)

As ZYBO is a hardcore processor, its synthesis results show that it does not need any logic resources for microprocessor exclusively whereas fast memory access requires reasonable logic resources. Total resources used in ZYBO design are 5443 out of 17600 LUTs, 6 out of 60 BRAM blocks and no embedded multipliers.

CHAPTER 4

RESULTS AND CONCLUSIONS

4.1 Performance data of basic operation

Performances of basic FP operations, which are needed for the implementation, were analyzed using different processors. Figure 13 shows the run time plot of individual floating point arithmetic operations. Compiler optimization levels of `-O3` for MicroBlaze and ARM, and `-O1` are adopted for Nios II respectively. As it can be seen from the below Figure 13, ARM processor is predominantly leading the race among all. FP units are boosting the performance for both Nios II and MicroBlaze at least by 25 times. In case of the ARM processor, its superiority is explained by the fact that it has 64-bit FP unit which is tightly coupled to the core, and built in support for all FP operations. In the case of Nios II, inclusion of FP unit does not improve anything upon conversion operations, `sqrt()` and `sqrtf()` operations. It is clear that those operations are 64-bit wide, hence the delay. However, we can expect some improvement in FLOPs for MicroBlaze upon the inclusion of FPU.

C/C++ GCC compilers have a capability of performing 64-bit operations at software level. But there is an immense need of finding HW which can perform in real time environment. Above data can provide us an idea on some of the critical aspects of processors while choosing the appropriate one for computationally intensive application. Compilers play very smart role nowadays. They reduce the operating system's burden of optimizing the code, through prioritizing the tasks, dynamically predicting the branches, and organizing the code to minimize the interdependencies among instructions. For example, they can help minimizing the time of conversion, for frequently used data types like INT to FP or FP to INT, based on the previous history and can exactly predict the result if the same request is made again. This phenomenon clearly explains the fact that, Nios II has taken significantly less time to do conversions like INT to FP and FP to INT in one single program. Enormous amount time has been elapsed while doing same conversions separately in different C/C++ codes.

Figure 13. Performance data of Floating point operations (FLOPs) using FPGA based microprocessor system using C/C++ GCC compilers with different compiler optimization levels

Table 7. Timing data for Nios II for conversion operations with and without FP HW

Without FP HW					
No of Operations (millions)	No of operations				
	10 M	20M	30 M	40 M	50M
FP to INT (s)	102.533	205.625	309.892	413.821	517.568
INT to FP and FP to INT(s)	62.084	124.622002	188.113007	251.514008	314.867004
INT to FP (s)	102.470916	205.500378	309.703887	413.569486	517.253133
With FP HW					
No of Operations (millions)	No of operations				
	10 M	20M	30 M	40 M	50M
FP to INT (s)	102.531	205.623	309.889	413.820	517.568
INT to FP and FP to INT(s)	62.074	424.622002	188.113007	251.514008	314.867004
INT to FP (s)	102.468916	205.498.378	309.700.887	413.568486	517.253133

4.2 Performance data of ECG algorithm

The proposed algorithm is performed with variety of processors. As mentioned in the section 2.1 and 2.2 Eigenvalues are calculated for the covariance matrix C_{xx} . Figure 14 shows the run times of all steps of the algorithm, for all processors. Nios II/s and MicroBlaze are tested under with and without inclusion of FP units. ARM and MicroBlaze have the compiler optimization options as $-O3$ and Nios II/s has $-O1$.

Table 8 shows the time taken to perform each step in the algorithm by Nios II processor.

Table 8. Time taken for ECG algorithm for individual steps in case of Nios II/s

Without FP HW							
ECG Algorithm	Steps						
	DC calc	Sub DC	Get C_{xx}	EV	Reconstruct	BMP+S/ N	Total Time
Time Taken (ms)	33	29	252	29	105	89	537
With FP HW							
ECG Algorithm	Steps						
	DC calc	Sub DC	Get C_{xx}	EV	Reconstruct	BMP+S/N	Total Time
Time Taken (ms)	5	3	14	30	10	49	111

From below figures we can conclude that ARM Cortex A9 has outperformed among all. It has a real time capability of performing double precision operations but 32-bit processor is sufficient for most part of the algorithm. And also its higher clock speed (666 MHz as compared to 100/50 MHz) and tight coupling of FP unit makes the processor more powerful. There are some interesting observations based on performance of basic arithmetic operations by different processors.

- Measured performance is substantially lower than the data from the manual [14-17], since no assembler code was used
- GCC compiler optimization from $-O0$ through $-O3$ will take insignificant longer compilation time, however there is improvement in run time by a factor of 2.3 for MicroBlaze+FP2 to 15.9 for ARM

Figure 14. Runtime of ECG algorithm for the FPGA-based microprocessor system compiled with C/C++ GNU compiler with optimization –O3 for ARM/MicroBlaze and –O1 for Nios II/s. Eigenvector Jacobi computation by JB

- Function available in math.h library such as `sin()`, `sqrt()`, `log10()` are defined for augmented double. But soft-core processors provide HW support for 32-bit FP not for 64-bit
- MicroBlaze and Nios II support floating point not double precision types using custom instructions. A factor of 3 slower performance was measured for the double `sqrt()` compared with C99 standard `sqrtf()` function
- Performance of Nios II shows overall improvement of a factor 20 when using the FP unit for 4 basic operations. Conversions from INT to FP and FP to INT and `sqrt()`, `sqrtf()` are not affected

- In case of MicroBlaze performance of basic operations are improved by factor of ca. 30 when used FP basic option. Extended FP option improved runtime for conversion functions (factor 7) and 32 bit `sqrtf ()` by a factor of 17. Double precision `sqrt ()` is not effected.

Figure 14 plots the runtime for all individual steps performed on all processors. We can observe following things from the plot.

- Only ARM Cortex A9 has the real-time performing capability among all.
- Runtime for standard computations such as DC calculations and `Cxx` was substantially longer (factor of 4.8 for ARM and 2.4 for MicroBlaze+FP2) than more complex eigenvector computation
- Execution time for Eigenvalue calculation did not improve for Nios II by adding FP because of the involvement of `sqrt ()` function used inside
- Execution time of DC calc and `Cxx` have improved by factor of 6.6 and 10 respectively overall time of execution by factor of 4.8.
- It can be seen from table 5 that total number of LUTs used have been increased by factor of 2.62 for Nios II which is much higher as compared to 1.21 for MicroBlaze+FP2
- Maximum operating frequency has no impact due to addition of FP unit in any processor

Overall run time is simply summation of times elapsed for individual steps. It might be useful to find the Frames per Second (FPS) a processor can process. FPS is simply reciprocal of the total runtime. It gives a sense of how many 375 samples can be computed per second. Table 9 shows FPS data of all processors.

Table 9. 375 point frames per second capability of the embedded configuration

μP	Nios II/s	Nios II/s	MicroBlaze	MicroBlaze	MicroBlaze	ARM Cortex A9
FP support	No	Yes	No	Basic	Sqrt & conv	Yes
FPS	1.2	5.7	6.6	97.3	102.4	2182.7

This comparison of 32-bit vs 64-bit FP unit support emphasizes their importance to perform in real-time and highlights the fact that, even small amounts of double precision operations can affect the overall execution time.

Figure 15. LCD display of mother and Fetus BPM using Altera DE2 FPGA monitor

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This study of real-time fetal ECG system design and experiments has some important conclusions.

- ARM Cortex A9 is the superior performer among three. If metric of FPS is relaxed, and the processor needs to evaluate only each incoming frame every time, then the Nios II/s with FP unit can also do the desired extraction and peak-picking job.
- Using more channels in a PCA system does not produce more reliable FECG. Hence Non-abdominal leads need not to be used.
- Algorithm is robust and independent of attributes such as changing the order of the channels or taking different data segments
- Peak picking method was developed is more robust than other FECG methods and it provides 3 level confidence score.
- Sophisticated algorithms such as Eigenvectors via Jacobi method require over (30 sparse) matrix multiplications and the use of FP arithmetic is recommended, since conversion of fixed-point to floating point takes too long.
- While using C/C++ algorithms, we have to avoid using functions which perform 64-bit double precision operations such as ANSI C87 function `sqrt ()` or `log10 ()` and replace these with C99 32-bit float functions. Over 100% run-time deduction was achieved by using C99 functions.
- A soft-core microprocessor allows an optimization to find power/size/speed optimum, but hard-core microprocessor provides substantially more power when FP operations are needed.

5.2 Future work

Over all system is fast, robust and compact. True power measurements can be performed for critical operations, which can help in determining battery life. This algorithm can be tried using different locations of leads on abdomen to see if there is any variation in the results. The

availability of FPGA resources, which can be included into a processor environment, opens opportunity to further improve performance and also power efficiency. We can assign some parts of the algorithm to be implemented by specialized hardware components, which can augment soft-core processors to become competitive or even superior to a hardcore processor. At the end it has a huge scope for commercialization.

APPENDIX A

EIGENVALUE ALGORITHMS

Eigenvalue (EV) calculation is one of the most important calculations in almost all fields of mathematics, engineering and technology. There are various algorithms developed to find the EVs. But iterative algorithms are most popular among computer engineers, because of the ease of coding and greatly reduce complex calculations. They are chosen by editors at Computing in Science and Engineering as the 10 most influential algorithms of the 20th century.

1. Calculation of EVs

Let $\mathbf{A}$ be an $N \times N$ matrix. If there exists any scalar value of λ , which can satisfy

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

Then the scalar value λ is called Eigenvalue and $\mathbf{x}$ is the Eigenvector vector matrix associated with λ .

Eigenvalues of any given $N \times N$ matrix can be calculated by its characteristic equation.

$$\det(\mathbf{A} - \lambda \mathbf{I}) = 0 = (\lambda_1 - \lambda)(\lambda_2 - \lambda)(\lambda_3 - \lambda)(\lambda_4 - \lambda) \dots (\lambda_n - \lambda) \quad (1)$$

The values $\lambda_1, \lambda_2, \lambda_3, \lambda_4 \dots \lambda_n$ are the Eigenvalues of $\mathbf{A}$ where $\mathbf{I}$ is the identity matrix. Equation (1) is called the characteristic polynomial of $\mathbf{A}$. For any $N \times N$ matrix, there will be N eigenvalues. But they need not be unique. In other words, EVs represent the dependency of linear combination of rows of $\mathbf{A}$. Each eigenvalue has its own corresponding set of eigenvectors.

But finding eigenvalues by hand calculation method is laborious for the matrices larger than 3×3 . So iterative methods are used to calculate. Even these methods involve the multiplications and additions are of the order n^3 . For example, matrix multiplication of two 4×4 matrices involve, 64 multiplications and 48 add operations.

So as mentioned as mentioned above, iterative methods are very easy to implement in computer programming. Most useful iterative methods are

- Jacobi iteration

- Power method
- QR factorization
- Scalar value decomposition (SVD)
- Power method

2. Jacobi Iteration method

This project implements Jacobi iteration method. The matrix is assumed to be symmetric. All the iteration method works by diagonalizing the matrix. i.e. making the upper and lower triangular elements converge as close as possible to zeroes. So that, the diagonal elements are the eigenvalues. In this particular case of Jacobi iteration, algorithm starts with making the lower diagonal elements as zeroes. This implementation is shown as below

$$\mathbf{A}_{k+1} = \mathbf{J}_k^T \mathbf{A}_k \mathbf{J}_k \text{ where } k=1,2,3,\dots$$

$\mathbf{J}_k$ and $\mathbf{J}_k^T$ are the Givens matrix and its transpose for k th iteration. Iterations terminate when $\mathbf{A}_{k+1}$ becomes only diagonal matrix. Then the diagonal elements are eigenvalues. This method also guarantees to preserve Eigenvectors.

First set of Givens matrices *and* Givens rotation are shown below. $c^2+s^2=1$ is the idea behind this method.

$$\mathbf{J}_1 = \begin{bmatrix} c & s & 0 & 0 & 0 \\ -s & c & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{J}_2 = \begin{bmatrix} c & 0 & s & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -s & 0 & c & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{J}_1^T = \begin{bmatrix} c & -s & 0 & 0 & 0 \\ s & c & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{J}_2^T = \begin{bmatrix} c & 0 & -s & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ s & 0 & c & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Where c and s are cosine and sine functions. They are calculated as shown below.

$$c = \frac{1}{\sqrt{1+t^2}} \text{ and } s = c*t, \text{ and } t \text{ is given by } t = \frac{\text{sign}(\tau)}{\tau + \sqrt{1+\tau^2}}$$

where $\tau = \frac{(a_{qq} - a_{pp})}{2a_{pq}}$ (a_{pq} is the off diagonal elements and a_{qq} , a_{pp} are diagonal elements).

$$A_2 = J_1^T * A * J_1 \text{ (1st rotation of 1st iteration)}$$

$$A_3 = J_2^T * A * J_2 \text{ (2nd rotation of 1st iteration)}$$

Every iteration involves $n(n-1)/2$ rotations, which converges the lower triangular matrix to zero, element by element. These are called sweeps. For example, above iterations will alter the values of 5th and 10th elements of A (index starts from 0 for the 1st element).

3. Modified Jacobian iteration

As shown above, rotations and iterations will continue until all the off diagonal elements are converged to zeroes. We can reduce the number of iterations by setting a threshold value, for each modified matrix value.

C code implementation

Threshold value is defined as

```
for ( j = 0; j < n; j++ )
{
    for ( i = 0; i < j; i++ )
    {
        thresh = thresh + a[i+j*n] * a[i+j*n];
    }
}
thresh = sqrt (thresh) / ( double ) ( 4 * n );
```

Threshold is calculated after every sweep.

$\text{thresh} = \frac{\sqrt{\sum A(p,q)^2}}{4n}$ is called Frobenius [2] norm. A gap function is defined as

“ $b = b + 10 * \text{eps}$ ”. This C implementation of Jacobi iteration used here followed online available routine developed by one of our FSU faculty John Burkardt, from the Department of scientific

computing. That algorithm modified gap function as “ $b=b+100*eps$ ” for faster convergence of off diagonal elements.

C code implementation of finding the Givens matrix is shown below.

```

/* Annihilate tiny offdiagonal elements.*/
if ( 4 < *it_num &&
    term_p == fabs ( d[p] ) &&
    term_q == fabs ( d[q] ) )
{
    a[p+q*n] = 0.0;
}
/*
    Otherwise, apply a rotation.
*/
else if ( thresh <= fabs ( a[p+q*n] ) )
{
    h = d[q] - d[p];
    term = fabs ( h ) + gapq;

    if ( term == fabs ( h ) )
    {
        t = a[p+q*n] / h;
    }
    else
    {
        theta = 0.5 * h / a[p+q*n];
        t = 1.0 / ( fabs ( theta ) + sqrt ( 1.0 + theta * theta ) );
        if ( theta < 0.0 )
        {
            t = - t;
        }
    }
    c = 1.0 / sqrt ( 1.0 + t * t );
    s = t * c;
    tau = s / ( 1.0 + c );
    h = t * a[p+q*n];
    and its rotations are given as below.
    /* Rotate, using information from the upper triangle of A only.*/
    for ( j = 0; j < p; j++ )
    {
        g = a[j+p*n];
        h = a[j+q*n];
        a[j+p*n] = g - s * ( h + g * tau );
        a[j+q*n] = h + s * ( g - h * tau );
    }
    for ( j = p + 1; j < q; j++ )
    {
        g = a[p+j*n];
        h = a[j+q*n];
        a[p+j*n] = g - s * ( h + g * tau );
        a[j+q*n] = h + s * ( g - h * tau );
    }
}

```

```

for ( j = q + 1; j < n; j++ )
{
    g = a[p+j*n];
    h = a[q+j*n];
    a[p+j*n] = g - s * ( h + g * tau );
    a[q+j*n] = h + s * ( g - h * tau );
}

```

Meanwhile diagonal elements are also modified in every sweep. It is implemented in C code as shown below.

```

/* Accumulate corrections to diagonal elements. */
    zw[p] = zw[p] - h;
    zw[q] = zw[q] + h;
    d[p] = d[p] - h;
    d[q] = d[q] + h;

    a[p+q*n] = 0.0; where h = d[q] - d[p];

```

4. Configuration of FP unit in Nios II.

In this particular project, calculation of EV is carried out to measure the floating point unit performance in terms of time and resource utilization. A header file called `sys\alt_alarm.h`, which generated by SOCP builder is used to perform the time calculations. Simulation is run in both presence and absence of FP unit. An argument to the GCC compiler can be passed to access FP unit i.e. including FP unit using software trigger. We can also include FP unit manually using the IP core available under Altera University program license.

```

# pragma nios2-elf-gcc -mcustom-fpu-cfg=60-2 (includes FP division)
# pragma nios2-elf-gcc -mcustom-fpu-cfg=60-1 (without FP division)

```

`-mcustom-fpu-cfg` is a package of add, sub and multiply operations. We can individually

```

-mcustom-fmul=252
-mcustom-fadd=253
-mcustom-fsub=254
-mcustom-fdiv=255

```

Alternatively, we can make the compiler not to use the floating point unit by the following macro definitions

```

#pragma no_custom_fadd -forces software implementation of floating-point add
#pragma no_custom_fsub -forces software implementation of floating-point subtract
#pragma no_custom_fmul -forces software implementation of floating-point multiply
#pragma no_custom_fdiv -forces software implementation of floating-point division

```

APPENDIX B

C IMPLEMENTATION OF THE ALGORITHM

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include "system.h"
#include "sys/alt_alarm.h"
#include "sys/matrix.h"
#include "jacobi_eigenvalue.h"

// Specify the device ID that is generated in xparameters.h
#define TIMER_LOAD_VALUE 0x7FFFFFFF
#define CLOCK_MHz 50
#define sample_window 375
#define channels 5
#define N 5
#define DEBUG 0
#define ATYPE 3
#define RSTART 2
#define REND 4
#define RINC 2

float ymean[channels][sample_window];
float send_array[channels][sample_window];
volatile float VctrMean[channels];
volatile float MtrxOrig[channels][sample_window];
volatile float MtrxMean[channels][sample_window];
float MtrxCxx[channels][channels];
float VctrEivals[channels];
float MtrxEivec[channels][channels];

volatile float MtrxOut[channels][sample_window];
char chararray[sizeof(float)] = {0};
float samplesdiv = 1/(float)sample_window;

int seconds = 0;
int status;

int main()
{
    int i,j, k, u;
    int rot_num, it_num;
    int Value1=0;
    int Value2=0;
    int Value21=0;
    int seconds=0;
    int first,second,x1,x2,dx, HR, BPM;
    float Vmax, v1, v2, ssum, nsum, Vmax50;
    float **aa;
    float **vv;
```

```

float *dd;
float *ee;
float **zz;
if (ATYPE==2){
dd = vector(0,N);
vv = matrix(0,N-1,0,N-1);
aa = matrix(0,N-1,0,N-1);
}
if (ATYPE ==3) {
dd = vector(1,N);
zz = matrix(1,N,1,N);
ee = vector(1,N);
}
printf("*****\n\nr");
printf("***** Hello World from ECG run *****\n\nr");
printf("*****\n\nr");

if (DEBUG) {
printf("--- First %dx%d Input values * 1000 (int) ...\n\nr",channels,channels);
for (i = 0 ; i < channels ; i++) {
for (j=0; j<5; j++) {
Value1 = decg[i][j] * 1000;
printf("%d , ",Value1);
}
printf("\n\nr");
}}
Value1 = alt_nticks();
for(i=0 ; i <channels;i++)
{
VctrMean[i] = 0;
for(j=0; j< sample_window;j++)
{ //Write Values in the Original (Untouched) Input Matrix
MtrxOrig[i][j]=decg[i][j];
//Write the Values in the Matrix that is later used as the Matrix with
the subtracted Mean Values
//MtrxMean[i][j]=decg[i][j];
MtrxMean[i][j] = MtrxOrig[i][j];
//Add the Row Values to a sum, later used for Mean Value Calculation
// When writing the values into the matrix, calculate the mean value
totals
// Saving two for loops

VctrMean[i] += MtrxMean[i][j];
}
//Calculate the Mean Value
VctrMean[i] *= samplesdiv;
}
//get timer value and compute elepased time
Value2 = alt_nticks();
Value21 = Value2 - Value1;
printf("Copy data cycles: %d\n\nr", Value21);
printf("Copy data time: %d/1000 ms\n\nr", (int) Value21/333);

if (DEBUG){
printf("--- The mean values are * 1000(int)...\n\nr");
}

```

```

for (i = 0 ; i < channels ; i++) {
    Value1 = VctrMean[i] *1000;
    Value2 = VctrMean[i];
    printf("Mean[%d]= %d /1000 = %d\r\n",i,Value1,Value2);
}

printf("*** Subtract mean values from matrix...\n\r");
Value1 = alt_nticks();
for (i = 0 ; i < channels ; i++)
{
    for (j = 0 ; j < sample_window ; j++)
    {
        MtrxMean[i][j] -= VctrMean[i];
    };
};

//get timer value and compute elapsed time
Value2 = alt_nticks();
Value21 = Value1 - Value2;
printf("Sub mean cycles: %d\r\n", Value21);
printf("Sub mean   time: %d/1000 ms\r\n", (int) Value21/333);

printf("*** Calculate the Covariance-Variance Matrix...\n\r");
Value1 = alt_nticks();
for (i = 0; i< channels ; i++) {
    for (k = 0; k< channels ; k++) {
        MtrxCxx[i][k] = 0;
        for (j = 0; j< sample_window ; j++) {
            //Matrix Multiplication: MtrxMean*MtrxMean'
            //However, by changing the access coefficients, the transpose can
be simulated
            // Saving a buffer matrix and creating the transposed matrix
            MtrxCxx[i][k] += (MtrxMean[i][j] * MtrxMean[k][j]);
        }
        MtrxCxx[i][k] *= samplesdiv;
    }
}

//get timer value and compute elepassed time
Value2 = alt_nticks();
Value21 = Value1 - Value2;
printf("Get Cxx cycles: %d\r\n", Value21);
printf("Get Cxx   time: %d/1000 ms\r\n", (int) Value21/333);
if (DEBUG) {
    printf("--- The %dx%d Cxx matrix is * 1000 (int) ...\n\r",channels,channels);
    for (i = 0 ; i < channels ; i++) {
        for (j=0; j<channels; j++) {
            Value1 = MtrxCxx[i][j] * 1000;
            printf("%d , ",Value1);
        }
        printf("\n\r");
    }
}

printf("*** Calculate the %d Eigenvectors and Eigenvalues...\n\r",channels);

if ( ATYPE == 1) {

```



```

        printf("*** Method is Jacobi/Rutishauser by src: John
Burkhardt...\n\r");
        Value1 = alt_nticks();//Start counter
        jacobi_eigenvalue_float ( channels, MtrxCxx, 100, MtrxEivec, VctrEivals,
&it_num, &rot_num );
        Value2 = alt_nticks();// Stop counter
    }

    // jacobi ( aa, n, dd, vv, &it_num, &rot_num ); from
    if ( ATYPE == 2) {
        printf("*** Method is Jacobi/Rutishauser by src: Numerical
Receips...\n\r");
        for (i=0;i<N;i++) for (j=0;j<N;j++) { aa[i][j]=MtrxCxx[i][j]; }

        Value1 = alt_nticks();// Start counter
        jacobi(aa, channels,dd,vv,&it_num, &rot_num); // from numerical receipts
        Value2 = alt_nticks();// Stop counter

        for (i=0;i<N;i++) for (j=0;j<N;j++) { MtrxEivec[i][j]=vv[i][j]; }
        for (i=0;i<N;i++) { VctrEivals[i]=dd[i]; }
    }

    if ( ATYPE == 3) {
        printf("*** Method is QL after Wilkison et al. (1968) by src: Numerical
Receips...\n\r");

        for (i=0;i<N;i++) for (j=0;j<N;j++) { zz[i+1][j+1]=MtrxCxx[i][j]; }
        Value1 = alt_nticks();// Start counter

        tred2 ( zz, N, dd, ee);// from numerical receipts
        tqli ( dd, ee, N, zz);
        Value2 = alt_nticks();// Stop counter

        for (i=0;i<N;i++) for (j=0;j<N;j++) { MtrxEivec[i][j]=zz[i+1][j+1]; }
        for (i=0;i<N;i++) { VctrEivals[i]=dd[i+1]; }
    }

    // compute elapsed time
    Value21 = Value1 - Value2;
    printf("Jacobi eig cycles: %d\r\n", Value21);
    printf("Jacobi eig time: %d/1000 ms\r\n", (int) Value21/333);
    if (DEBUG) {
        printf("--- Jacobi: Iterations=%d Rotations=%d\r\n",it_num,rot_num);
        printf("--- The %dx%d Eigenvector matrix is * 1000 (int)
...\n\r",channels,channels);
        for (i = 0 ; i < channels ; i++) {
            for (j=0; j<channels; j++) {
                Value1 = MtrxCxx[i][j] * 1000;
                printf("%d , ",Value1);
            }
            printf("\n\r");
        }

        if (DEBUG){
            printf("--- The %d Eigenvalues are...\n\r",channels);
            for (i = 0 ; i < channels ; i++) {
                Value1 = VctrEivals[i] *1000;
                printf("EV[%d]= %d /1000 = %d\r\n",i,Value1,Value1/1000);
            }
        }
    }

```

```

}}
//Multiply Eigenvalues with Eigenvectors
//Eigenvalues Matrix has only diagonal elements
//No general matrix multiply required
//MtrxCxx is reused and used as buffer matrix

printf("*** Calculate the Reconstruction...\n\n");
Value1 = alt_nticks();
for (i = 0 ; i < channels ; i++) {
    for (j = 0 ; j < channels ; j++) {
        MtrxCxx[i][j] = MtrxEivec[i][j] / sqrt(VctrEivals[i]);
    }
}

//Compute Output values
for(i = 0 ; i < 2 ; i++) {
    for (j = 0 ; j < sample_window ; j++) {
        MtrxOut[i][j]=0;
    }
}

//Compute all output values or the required 2+4
for (i = RSTART; i<= REND ; i+=RINC)
{
    for (k = 0; k< sample_window ; k++)
    {
        MtrxOut[i][k]=0;
        send_array[i][k]=0;
        for (j = 0; j< channels ; j++)
        {
            //MtrxOut[i][k] += (MtrxCxx[i][j] * MtrxOrig[j][k]);
            send_array[i][k] += (MtrxCxx[i][j] * MtrxOrig[j][k]);
        }
    }
}

//get timer value and compute elapsed time
Value2 = alt_nticks();
Value21 = Value1 - Value2;
printf("*** Reconstruct for %d channels...\n\n", (REND-
RSTART)/RINC+1);
printf("Reconstruct cycles: %d\n\n", Value21);
printf("Reconstruct time: %d/1000 ms\n\n", (int) Value21/333);
printf("*** Calculate BPM for mother and fetus...\n\n");
Value1 = alt_nticks();
if (DEBUG) {
    printf("**** 1) square first to reduce noise\n\n");
    printf("**** 2) Moving average of 5 sample \n\n");
    printf("**** 3) Find maximum in window\n\n");
    printf("**** 4) Inspected data and find first maximum (x1,v1)\n\n");
    printf("**** 5) Inspected data and find second maximum (x2,v2)\n\n");
    printf("**** 6) Compute BPM via 250/(x2-x1)*60\n\n");
    printf("**** 7) Determine S/N (reliability) via S range=>x1+10;x2-10; N=
x2+10...x2-10\n\n");
}
for (k=RSTART;k<=REND;k+=RINC) { // Do mother and fetus PC only: use K02 k+=2

```

```

for (j = 0; j< sample_window ; j++) { ///// Square
    send_array[k][j]*=send_array[k][j];
    ymean[k][j]=0.0;
}
for (j = 0; j< sample_window ; j++) { ///// Mean
    if (j>5) {
        for (i=0;i<5;i++) {
            ymean[k][j] += send_array[k][j-i];
        }
    } else {
        ymean[k][j] = send_array[k][j];
    }
}
Vmax=0.0;
for (j = 0; j< sample_window ; j++) { ///// Get max
    if (ymean[k][j]>Vmax) Vmax=ymean[k][j];
}

////////// Step 4+5: first the first and second maximum
Vmax50=Vmax*0.5; j=Vmax;
if (DEBUG) printf(" Search Ch. %d: max is %d\n\r",k,j);
first=1;second=0;v1=0;x1=0;v2=0;x2=0;
for (i=0; i<= sample_window; i++) {
    if ((first==1) && (ymean[k][i] > v1) && ( ymean[k][i] > Vmax50)) {
        v1 = ymean[k][i]; x1=i; j= v1;
        if (DEBUG) printf("First: new max %d at %d\n\r",j,i);
    }
    if (((i-x1) > 30) && (first==1) && (v1>Vmax50) && (second==0)) {
        second=1;first=first+1;
        if (DEBUG) printf("Look for second: at  %d \n\r",i);
    }
    if ((second==1) && (ymean[k][i] > v2) && (ymean[k][i] > Vmax50)) {
        v2 = ymean[k][i]; x2=i; j=v2;
        if (DEBUG) printf("Second: new max %d at %d\n\r",j,i);
    }
    if (((i-x2) > 30) && (second==1) && (v2>Vmax50)) {
        second=second+1;
        if (DEBUG) printf("Done with second: at %d\n\r",i);
    }
}

////////// Step 6: Compute BPM
dx=x2-x1; HR=250.0/dx*1000; BPM=250.0*60/dx;
if (DEBUG) printf(" --> Heart rate Ch %d sample=%d => %d /1000 Hz => %d
BPM\n\r",k,dx,HR,BPM);

////////// Step 7: Compute S/N
nsum=0.0;
for (i=x1+10;i<=x2-10;i++) { /// Add the noise
    nsum = nsum + ymean[k][i] ;
}
ssum=0.0;
for (i=0;i<10;i++) { /// Add the signal
    ssum=ssum+ ymean[k][x1+i];
    ssum=ssum+ ymean[k][x2-i];
}

```

```

    }
    i=ssum; j=nsum; u=10.0*log10(ssum/nsum)*1000;
    if (DEBUG) printf(" --> Ch %d S/N %d/%d = %d /1000 dB\n\r",k,i,j,u);
}

//get timer value and compute elepased time
Value2 = alt_nticks();
Value21 = Value1 - Value2;
printf("BMP + S/N cycles: %d\r\n", Value21);
printf("BMP + S/N   time: %d/1000 ms\r\n", (int) Value21/333);

    printf("Second counter starts\n\r");
    Value1=alt_nticks();
    while (seconds<5) {
        Value2 = alt_nticks();
        Value21 = Value1 - Value2;
        if (Value21 > 333500000){ // second count is 667/2MHz cycles per second
            Value1 = Value2;
            printf("seconds=%d   ",seconds);
            seconds++;
        }
    }

    printf("\n\rSuccessfully ran ECG on ARM\r\n");
    return 0;
}

```

REFERENCES

- [1] Meyer-Baese. U., [Digital Signal Processing with Field Programmable Gate Arrays], 4th ed., Springer-Verlag, New York (2014).
- [2] U. Meyer-Baese, H. Muddu, S. Schinhaerl, M. Kumm, P. Zipf; “Real-time Fetal ECG System Design using Embedded Microprocessors” Proc. SPIE Commercial + Scientific Sensing and Imaging. April 2016, Vol. 9871, pp. 987106-1-14, doi: 10.1117/12.2224256
- [3] ECG Overview 2.75/6.525/2.750/6.025 Design of Medical Devices; September 12, 2013, <http://web.mit.edu/2.75/lab/ECG%20Overview.pdf>
- [4] Callaerts D., Vanderwalle, J. Sansen W., Janssesns J. and Vantrappen G. “Acquisition and processing of the antepartum FECG“ in A Critical Appraisal of Fetal Surveillance, Elsevier Science B.V., Ch. 43, 371-380 (1994)
- [5] B. De Moor, Database for the identification of systems (DaISy), online <http://homes.esat.kuleuven.be/~smc/daisy/daisydata.html>
- [6] M.A. Hasan, M.B.I. Reaz, M.I. Ibrahimy, M.S. Hussain, J. Uddin., Detection and Processing Techniques of FECG Signal for Fetal Monitoring, Biol Proced Online. 2009; 11: 263–295. Published online 2009 Mar 27. doi: 10.1007/s12575-009-9006-z
- [7] <http://www.southsudanmedicaljournal.com/archive/may-2010/how-to-read-an-electrocardiogram-ecg.-part-one-basic-principles-of-the-ecg.-the-normal-ecg.html>
- [8] Andrew Moore, FPGAs Handbook, ALTERA Special Edition, John Wiley & Sons, Inc, (2014), Hobolken, NJ.
- [9] http://sine.ni.com/np/app/main/p/ap/imc/lang/en/pg/1/sn/n17:imc,n19:FPGA_target/fmid/2170/
- [10] Altera’s User-Customizable ARM-Based SoC https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/br/br-soc-fpga.pdf, last accessed 10-20-2016.
- [11] Zybo Reference Manual: DOC# 502-279, Revised February 14, 2014. Last accessed 11-07-2016.
- [12] A Zynq Accelerator for Floating Point Matrix Multiplication Designed with Vivado HLS, XAPP1170 (v2.0) January 21, 2016, last accessed 10-20-2016.
- [13] Cortex -A9 Floating Point Unit, Revision: r4p1, Technical reference manual, last accessed 11-7-2016.

- [14] Altera, online http://www.altera.com/literature/tt/tt_floating_point_custom_instructions.pdf, last accessed: 09-25-2016
- [15] ARM, Cortex-A9 Floating-Point Unit: Technical Reference Manual, 2012 online: <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.cortexa.a9/index.html>, last accessed 10-3-2016
- [16] U. Meyer-Baese, G. Botella, S. Mookherjee, E. Castillo, A. García “Energy optimization of Application-Specific Instruction-Set Processors by using hardware accelerators in semicustom ICs technology” Original Research Article, *Microprocessors and Microsystems*, 36(2), 127-137 (2012)
- [17] Xilinx, “MicroBlaze Processor Reference Guide EDK: FPU software support” Ver. 14.4, p. 78, last accessed 11-07-2016
- [18] Solum T., Ingemarsson I., Nygren A. (1980) The accuracy of abdominal ECG for fetal electronic monitoring. *J Perinat Med* 8:142–149.
- [19] Echeverria, J. C., Ramirez, N., Pimentel, A.B., et al. (1996) Fetal QRS extraction based on wavelet analysis and pattern matching. *Engineering in medicine and biology society. Bridging disciplines for biomedicine. Proceedings of the 18th Annual International Conference of the IEEE. Amsterdam. Netherlands; 31 October–3 November 1996. vol. 4. p. 1656–1657.*
- [20] Bergveld P., Arjan J. K., Peuscher J. H. J. (1986) Real-time fetal ECG Recording. *IEEE Trans Biomed Eng BME-33:505–509.*
- [21] Mallat S. G. (1989) A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Trans Pattern Anal Mach Intell* 1:674–693.
- [22] Abboud S., Sadeh D. (1989) Spectral analysis of the fetal electrocardiogram. *ComputBiol Med* 19:409–415.
- [23] Datian, Y., Xuemei, O. (1996) Application of wavelet analysis in detection of fetal ECG. *Engineering in Medicine and Biology Society. Bridging Disciplines for Biomedicine. Proceedings of the 18th Annual International Conference of the IEEE. Amsterdam. Netherlands; 31 October–3 November 1996. vol. 3. p. 1043–1044.*
- [24] Mallat S., Hwang W. L. (1992) Singularity detection and processing with wavelets. *IEEE Trans Inf Theory* 38:617–643.
- [25] Mochimaru F., Fujimoto Y. (2002) Detecting the fetal electrocardiogram by wavelet theory-based methods. *Prog Biomed es* 7:185–193.
- [26] Zarzoso V., Nandi A. K. (2001) Noninvasive fetal electrocardiogram extraction: blind separation versus adaptive noise cancellation. *IEEE Trans Biomed Eng* 48:12–18.

- [27] de Araujo, D., Barros, A. K., Baffa, O., et al. (2003) Fetal magnetocardiographic source separation using the poles of the autocorrelation function. Proceedings of the 4th international symposium on independent component analysis and blind signal separation, Nara, Japan, 1–4 April 2003, pp. 833–836.
- [28] Marossero, D. E., Erdogmus, D., Euliano, N., et al. (2003) Independent components analysis for fetal electrocardiogram extraction: a case for the data efficient mermaid algorithm. Neural Networks for Signal Processing (NNSP'03). IEEE 13th Workshop. Toulouse, France; 17–19 September 2003. p. 399–408.
- [29] Hild K. E. II, Erdogmus D. E., Principe J. (2001) Blind source separation using Renyi's mutual information. IEEE Signal Process Lett 8:174–176.
- [30] S. Redif., U. Fahroglu., et al (2010) Foetal ECG Extraction Using Broadband Signal Subspace Decomposition. Published in Microwave Symposium (MMS), 2010 Mediterranean conference 25-27 August 2010.
- [31] Najafabadi, F. S., Zahedi, E., Ali, M. A. M. (2005) A novel model for abdominal electrocardiography of a pregnant woman. Sensors and the international conference on new techniques in pharmaceutical and biomedical research 64–68. Kuala Lumpur, Malaysia.
- [32] Liszka-Hackzell, J. (1994) Categorization of fetal heart rate patterns using neural networks. Comput Cardiol 97–100. Bethesda, Maryland, USA.
- [33] Assaleh K (2007) Extraction of Fetal Electrocardiogram Using Adaptive Neuro-Fuzzy Inference System. IEEE Trans Biomed Eng 54: 59-68.
- [34] Selvan, S., Srinivasan, R. (2000) A novel adaptive filtering technique for the processing of abdominal fetal electrocardiogram using neural network. Adaptive systems for signal processing, communications and control symposium 2000. Louise, Alta, Canada; p. 289–292.
- [35] Lin W. C., Feng C. H., Neuman M. R. (1977) A microprocessor-based data acquisition and processing system for studying the kinematics of labor. Proc IEEE 65:722–729 New York, US.
- [36] Pimentel, B. S., de Avila Valgas Filho, J. H., Campos, R. L., et al. (2001) A FPGA implementation of a DCT-based digital electrocardiographic signal compression device. 14th symposium integrated circuits and systems design. Pirenopolis. Brazil; 10–15 September 2001. p. 44–49.
- [37] M.A. Hasan, M.B.I. Reaz, M.I. Ibrahimy, M.S. Hussain, J. Uddin., VHDL Modeling of FECG Extraction from the Compisite Abdominal ECG using Artificial Intelligence, Measurement Science Review. Volume 12, Issue 2, Pages 52–55, ISSN (Online) 1335-8871, DOI: 10.2478/v10048-012-0007-8, April 2012.
- [38] https://people.sc.fsu.edu/~jburkardt/c_src/jacobi_eigenvalue/jacobi_eigenvalue.html

- [39] Rutishauser, H. “The Jacobi Method for Real Symmetric Matrices” *Numerische Mathematik* 9, 1-10 (1966).
- [40] William H., Teukolsky, S., Vetterling, W. and Flannery, B. *Numerical Recipes: The Art of Scientific Computing* 3rd edition, New York, USA, Cambridge University Press (2007).
- [41] S. Hunter and S Robson, “Adaption of the maternal heart in pregnancy” *Br Heart J.* 1992 Dec; 68(6):540-543.
- [42] Chawla, M.P.S. *Neural Comput & Applic* (2009) 18: 539. doi:10.1007/s00521-008-0195-1 <http://link.springer.com/article/10.1007/s00521-008-0195-1>
- [43] Charoensak, C., Sattar, F. (2005) Design of low-cost FPGA hardware for real-time ICA-based blind source separation algorithm. *EURASIP Journal on Applied Signal Processing* 3076–3086.
- [44] Shelens, J., A Tutorial on Principle Component Analysis, December 10, 2005: version 2
- [45] ALTERA Media Compute System for the Altea DE2 Board, Available under University Program license.
- [46] Solum T., Ingemarsson I., Nygren A. (1980) The accuracy of abdominal ECG for fetal electronic monitoring. *J Perinat Med* 8:142–149.
- [47] Maria P., John C., Jean-Francois P. et al. (2001) Monitoring the fetal heart noninvasively: a review of methods. *J Perinat Med* 29:408–416.

BIOGRAPHICAL SKETCH

Harikrishna Muddu was born in Achalapuram, India. He obtained his bachelor's degree in Electronics and Communication Engineering from NIT Hamirpur (HP) India in 2012. Later he worked for Atos India Private Limited for 3 years as an Application Developer. In 2015 he joined in Master of Science program in the Department of Electrical and Computer Engineering at FAMU-FSU College of Engineering. His areas of interests include Embedded Microprocessor System Design, FPGAs, Computer Architecture and Real-time Operating Systems. He published a paper on "Real-Time Fetal ECG system Design Using Embedded Microprocessors", in May 2016, in SPIE Conference, Baltimore, MD.