Investigation into a Light-Weight Reconfigurable VLSI Architecture for Biomedical Signal Processing Applications

by

NUPUR JAIN
201221008

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

March, 2019

Declaration

I hereby declare that

i) the thesis comprises of my original work towards the degree of Doctor of Philosophy at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) due acknowledgment has been made in the text to all the reference material used.

Nupur Jain

Certificate

This is to certify that the thesis work entitled INVESTIGATION INTO A LIGHTWEIGHT RECONFIGURABLE VLSI ARCHITECTURE FOR BIOMEDICAL SIGNAL PROCESSING APPLICATIONS has been carried out by NUPUR JAIN for the degree of Doctor of Philosophy at *Dhirubhai Ambani Institute of Information and Communication Technology* under my supervision.

Prof. Biswajit Mishra
Thesis Supervisor

Acknowledgments

Undertaking this PhD has been a truly life-changing experience for me and it would not have been possible to do without the support and guidance that I received from many people.

I would like to first say a very big thank you to my supervisor Dr. Biswajit Mishra for all the support and encouragement he gave me, during the time I spent at DA-IICT and for providing me post-doctoral opportunity later on. Without his guidance and constant feedback this PhD would not have been achievable.

I greatly appreciate the support received from dearies Palas Parmar and Mansi Singh during mapping DCT and DWT algortihms. They were patient in understanding my work and all the more dedicated in using it in their respective M.Tech Theses. Thank you both.

I am indebted to all my friends and family who were always so helpful in numerous ways. Special thanks to Archana, Kamal, Miral, Nidhi, Parth, Sarita, Sujata and Vandana for their friendship and the warmth they extended during my doctoral studies. My deep appreciation goes out to all the present (Bittu, Dhrumil, Neha and Yash) and past lab members for keeping the work environment real fun as well as for the technical help, discussion and motivation I needed from time to time.

A very special thank you both Dr. Amit Bhatt and Dr. Ritu Parekh for their valuable advice and feedback on my research and for always being so supportive of my work. And of course a big thank you to Purvi P., for being a wonderful first (lab) mate, motivating me during various phases of my thesis and helping me enormously, especially with the mammoth task of submission of this thesis.

I would also like to say a heartfelt thank you to my Mummy, Daddy, and

Ankur bhaiya for always believing in me and encouraging me to follow my dreams. And Mummyji, Papaji and Shreya for helping in whatever way they could during this challenging period. And finally to Kesher, who has been by my side throughout this Ph.D, living every single minute of it, and without whom, I would not have had the courage to embark on this journey in the first place. And to darling son, Keyansh, for being such a good little baby that past one year, and making it possible for me to complete what I started.

Contents

Abstract **ix**

List of Tables **x**

List of Figures **xiii**

List of Publications **xix**

1 Introduction **1**

1.1 Monitoring Systems 3

1.1.1 Patient Monitoring Systems 3

1.1.2 Biotelemetry and Telemedicine 4

1.1.3 Ambulatory Monitoring 4

1.2 Thesis Contributions 8

1.2.1 Thesis Organisation 10

2 Literature Review **11**

2.1 General introduction to Wireless Sensor Networks (WSNs) 11

2.2 Wireless Body Area Networks (WBANs) 12

2.2.1 Constraints of Wireless Body Sensor Nodes 13

2.3 System Specifications 14

2.3.1 Power/Energy Profile 14

2.3.2 Performance 16

2.3.3 Functionality 17

2.4 Various Body Sensor Node Systems 19

3 Reconfigurable Shift-Accumulate (SAC) Architecture **27**
3.1 Underlying Mathematics . . . 28
3.1.1 Optimized Interpretation . . . 29
3.2 Functional Units . . . 30
3.2.1 Register Unit . . . 30
3.2.2 Computation Unit . . . 31
3.2.3 Architecture Operation . . . 35
3.2.4 Configurable Datapath . . . 36
3.2.5 Configurable Datapath Gate Count . . . 39
3.3 Architecture Topologies . . . 40
3.3.1 Systolic Array . . . 40
3.3.2 Four Tile . . . 41
3.3.3 Systolic Tile . . . 41
3.4 Other Domain-Specific Reconfigurable Architectures . . . 42
3.5 Conclusion . . . 44

4 Mapping of Processing Functions **46**
4.1 Variable Coefficient Functions . . . 47
4.1.1 Multiplication . . . 47
4.1.2 Multiply-Accumulate (MAC) . . . 50
4.1.3 FIR filtering . . . 55
4.2 Fixed Coefficient Functions . . . 62
4.2.1 COrdinate Rotation DIgital Computer (CORDIC) . . . 63
4.2.2 Compression Algorithms . . . 76
4.2.3 Discrete Cosine Transform . . . 92
4.3 Conclusion . . . 101

5 System Verification on an FPGA **102**
5.1 Number System . . . 102
5.2 Control Word for Various functions and Topologies . . . 104
5.3 Configuration of the Architecture . . . 106
5.4 Simulation Results . . . 108

5.4.1 FIR Filtering . . . 108
5.4.2 CORDIC Algorithm . . . 110
5.4.3 Discrete Cosine Transform (2-D DCT) . . . 113
5.5 Hardware Results . . . 115
5.6 Overview of Field Programmable Gate Array (FPGA) Board . . . 117
5.7 Design of Peripheral Interface Controllers . . . 117
5.7.1 RS-232 Communication Interface . . . 123
5.8 Discrete Wavelet Transform (2-D DWT) . . . 127
5.8.1 Block ROM . . . 128
5.8.2 Digital clock manager (DCM) . . . 129
5.8.3 Reconstructed image . . . 130
5.9 Multiple Function realization . . . 133
5.10 Clock Profiling . . . 133
5.11 Gate Profiling . . . 134
5.12 Comparison with Similar Works . . . 135
5.13 Conclusion . . . 136

6 On-the-fly Application Reconfigurability 137
6.1 Morphology of ECG signal . . . 139
6.2 Pan-Tompkins Algorithm (PTA) for QRS detection . . . 139
6.2.1 Pan-Tompkins Algorithm: Analysis and Discussion . . . 140
6.3 The Shift-Accumulate (SAC) Architecture . . . 142
6.3.1 Control Word and Configurable Datapath . . . 143
6.4 On-the-fly Reconfigurability . . . 144
6.4.1 Memory Management and Interpretation Methodology . . . 145
6.4.2 State Restoring Methodology . . . 146
6.4.3 Configuration Methodology . . . 147
6.5 Results and Discussion . . . 149
6.5.1 System overview . . . 149
6.5.2 QRS Detection . . . 150
6.5.3 Related work . . . 157
6.6 Circuit Implementation . . . 159

6.7 Conclusion 164

7 Conclusion and Future Work **166**

7.1 Summary of Contributions 167

7.2 Modularity of the SAC Architecture 169

7.3 Future Work 170

References **172**

Appendix A VGA and UART Controllers basics **196**

A.1 VGA Interface 196

A.1.1 Basic operation of a Display Panel: 197

A.2 RS-232 Communication Interface 199

Appendix B ECG Concepts **203**

B.1 The ElectroCardioGram (ECG) 203

B.1.1 Standard 12-lead ECG 204

Appendix C Mapping 8×8 DCT on 3×3 SAC Architecture **208**

C.1 Mapping scheme for 8×8 DCT on (3×3) SAC architecture 208

Abstract

The Body Sensor Network systems consist of signal acquisition and processing blocks along with Power Management Unit and radio transmission capabilities. The high power consumption of the radio transmission is often eliminated by adopting the on-node processing through signal processing platform with increased computation ability. Dedicated hardware accelerators optimized for operations predominantly seen in biomedical signal processing algorithms are often used in tandem with a microprocessor for this purpose. However, they do not support further algorithm improvements and optimizations owing to their dedicated nature. The benefits of configurability can be found in reconfigurable architectures at the cost of reconfiguration overheads. The shift-accumulate architecture developed in this thesis leverage the regularity in dominant functions in biomedical signal processing and thereby yields gate count advantages. The configurable datapath of the architecture renders multiple DSP operation emulation by means of mapping methodologies developed for efficient realization in terms of hardware utilization and memory accesses. The architecture exhibits various topologies which further supports efficient function realization.

The configuration scheme of the architecture is developed which effectively consist of control word and tightly coupled data memory. The architecture is realized on a Filed Programmable Gate Array (FPGA) platform demonstrating the target function emulation and hardware results are compared with ideal outcomes. The Video Graphics Array (VGA) and Universal Asynchronous Receiver Transmitter (UART) interface controllers are developed in this work for error quantification and analysis. The architecture contains a 6×6 array of functional units having shift-accumulate as its underlying operation and has gate count of $\approx$25k

and 46.9 M*Hz* operating frequency while emulating 36-tap FIR, CORDIC, DCT, DWT, moving average, squaring and differentiation functions.

Generally, biomedical signal processing functions include multiple stages consisting of noise removal, feature detection and extraction *etc.* The on-the-fly reconfigurability is incorporated into the architecture that leverage the low input datarates of biosignals. The architecture reconfigures dynamically while realizing different functions of the signal chain. The memory adapts to the incoming target function and supports 7 functions in its present structure. However, the architecture and memory remains scalable. Pan-Tompkins Algorithm based QRS detection realization is demonstrated on the architecture using the reconfigurability. This work offers $\approx 4\times$ reduced area and $2.3\times$ increase in performance with respect to the existing contemporary literatures.

List of Tables

2.1 Performance specification of biomedical signal processing applications. . . . 15
2.2 Typical data rates and bandwidth of human biopotentials and biophysical signals [1] . . . 17
2.3 Digital signal processing functions in ambulatory biomedical applications. . . . 18

3.1 The comparison of Methods I, II and III suited to implement multiply-accumulate equation. . . . 29
3.2 Number of gates in RU . . . 32
3.3 Number of gates in CU . . . 34
3.4 Gate count of multiplexers in Configurable Datapath . . . 39

4.1 The value of $arctan(2^{-i})$ with 2-bit precision. . . . 64
4.2 The hyperbolic variable (m), the mode of operation, arctan variable (α) along with intial guess and post-processing for the supported functions. . . . 66
4.3 The RU coefficients for the CORDIC variables computation . . . 72
4.4 Filter coefficients of the targetted wavelets . . . 84
4.5 The active RUs, coefficients loaded and result computed in various states of 2-D DWT state machine. . . . 88
4.6 DCT multiplexer coefficients in decimal and hexadecimal. . . . 100

5.1 Control word for target functions and topologies. . . . 105
5.2 The resolution bits are respective output sections selected for feedback. . . . 106

5.3 Range and Resolution of CORDIC on developed Architecture . . . 109
5.4 % relative accuracy for CORDIC functions. 113
5.5 Comparison of various mask size with respect to computation, memory and MSE metrics. 114
5.6 Relation between write address of BRAM_2 with read address of BRAM_1 122
5.7 Comparison of the actual and reconstructed Lena images 128
5.8 The comparison of various 2-D DWT block sizes. 128
5.9 Comparison of the original and reconstructed checkerboard images. 132
5.10 Comparison of the original and reconstructed Lena images. 132
5.11 Comparison with state of the art. 135

6.1 ECG characteristic features specifications for normal ECG signal. . 139
6.2 Transfer function of PTA blocks [2]. 140
6.3 Data selection by Configuration and Bypass Multiplexers 144
6.4 Range and Resolution of functions in PTA signal chain 150
6.5 Resolution of PTA functions and their representation in the control word 150
6.6 Control word of the PTA based QRS detection signal chain 153
6.7 Clock cycle consumed in Configuration and Computation of the Architecture 155
6.8 Gate Count of the Architecture 156
6.9 Comparison with State-of-the-art 158
6.10 Basic gates characterized in terms of delay, power and energy at three different (1.8V, 0.7V and 0.5V) V_{DD} 159
6.11 The Control Word and Energy Consumption (0.5V, 1MHz) of target functions. 160
6.12 The cycle counts of target functions. 161
6.13 Gate count breakup of the architecture along with its interfaces and supporting state machine. 162
6.14 Comparison with state-of-the-art. 164

A.1 VGA Resolution Timing Parameters. 199
A.2 Time required by UART to transfer image of different sizes 202

List of Figures

1.1 Block Diagram of Man-Instrument System. 2
1.2 The Block Diagram of Biotemeletry Transmitter and Receiver. . . . 3
1.3 A Conceptual Generic Biomedical System. 5
1.4 The power consumption profile for a WATS system based on off-the-shelf blocks for an ECG application [3]. 6
1.5 Proportion and kind of signal processing functions in (a) ECG Arrhythmia Classification [4] (b) Heart Sound processing [5] (c) Pulse Oximetry [6]. 7

2.1 Various fields of Application of WSN (Adapted from OECD). . . . 12

3.1 The Shift-Accumulate Architecture. 30
3.2 The Register Unit. 31
3.3 The Computation Unit hierarchy. 33
3.4 MSB duplication in RCA of tree adder with example. 34
3.5 MSB rotation in coefficient register. 35
3.6 The Configuration and Bypass multiplexers in SAC Architecture. . 38
3.7 Different Architecture Topologies. 40

4.1 Mapping scheme for multiplication along with circular and coefficient memories. 49
4.2 The direct hardware implementation of multiply-accumulate operation. 50
4.3 Active RUs and datapath for mapping MAC. 51
4.4 The circular and coefficient memory connection for MAC mapping. 54
4.5 The direct-form realization of FIR filter. 57

4.6 Lattice filter implementation of FIR filter. 57
4.7 Active RUs and datapath for FIR mapping. 60
4.8 The circular and coefficient memory connections for FIR mapping. 61
4.9 (a) Given's Rotation (b) Elementary angles of rotation. 63
4.10 Implementation of one iteration. 65
4.11 Word-serial implementation. 67
4.12 Active RUs and datapath for CORDIC mapping. 69
4.13 Flow chart of CORDIC state machine. 74
4.14 The circular memory and coefficient multiplexer connections for CORDIC mapping. 75
4.15 Decomposition and reconstruction of an image using 2D-DWT . . . 78
4.16 Convolution-based architecture using parallel filters. F_H and F_G are the length of low-pass and high-pass filters, respectively. 79
4.17 Coefficients of bior2.2 after zero padding 82
4.18 Pictorial representation of 2-D DWT algorithm. 83
4.19 Active RUs and datapath for 2-D DWT mapping. 85
4.20 Flow chart of 2-D DWT state machine. 87
4.21 The DWT mask manipulation. 89
4.22 Data stored in interleaved manner in the circular memory with the write and read pointers 90
4.23 The coefficient multiplexers for 2-D DWT. 91
4.24 Computation of 2-D DCT using separability property 94
4.25 The 2-D DCT algorithm steps. 96
4.26 Flow chart of 2-D DCT state machine. 98
4.27 The coefficient multiplexer for 2-D DCT mapping. 99

5.1 The notional representation of binary point in fixed-point number system. 103
5.2 Architecture input bus. The zoomed section denotes the multiplication results and 8-b serialized output. 107
5.3 The Pan-Tompkins Algorithm signal chain. 109
5.4 Hardware and MATLAB results for LPF in PTA. 110

5.5 Hardware simulation results of CORDIC in (1) Rotation non-hyperbolic computing sine and cosine (2) Rotation Hyperbolic computing hyperbolic sine and cosine and (3) Vectoring Hyperbolic modes computing Ln(x). 112
5.6 The hardware and MATLAB results for (a) Sine and Cosine (b) Hyperbolic Sine and Cosine (c) exponential (d) natural logarithm functions. 113
5.7 Convergence point for DCT matrix of size 8x8. 115
5.8 DCT (a) Hardware setup block diagram (b) Hardware computed (c) MATLAB computed results. 116
5.9 (a)System level interaction of SAC Architecture with peripherals (b) Actual hardware setup. 116
5.10 XUP Virtex-II Pro Development System Board Photo 118
5.11 Block Diagram of Image Processing Module 119
5.12 Relation of sync signals with the pixels of the screen 120
5.13 Logic for Timing Generation module 121
5.14 Hardware Setup for image transformations using VGA interface. . 123
5.15 Results for (a) Identity (b) Inversion (c) Vertical Flip (d) InvertFlip transforms on the "240 x 160" loons image. 123
5.16 State Machine of UART Controller 126
5.17 Block diagram of UART Transmitter Controller. 127
5.18 Screenshot of Coolterm Terminal showing data received in hexadecimal format 127
5.19 The MSE and # of computations plotted against different 2-D DWT block sizes. 129
5.20 UART output 130
5.21 Reconstruction of image I_R from the coefficient matrix C 131
5.22 DWT Results on 16×16 checkerboard image (a) Actual (b) MATLAB simulated (c) Hardware computed. 131
5.23 DWT Results on 128×128 Lena intensity image (a) Actual (b) MATLAB simulated (c) Hardware computed. 132

5.24 Hardware results for simultaneous mapping of two functions on SAC Architecture. 133

6.1 ECG fudicial points. 138
6.2 (a) Reduced PTA signal chain. (b) Magnitude response of LPF-HPF and BPF in PTA. 141
6.3 Noise attenuation block. (a) The LPF-HPF (PTA) and modified BPF operated ECG signal. (b) Absolute error between them. 141
6.4 The (3×3) SAC architecture along with its 18-b control word. 142
6.5 Timing Diagram of QRS signal chain in respect to the input sample frequency. 144
6.6 (a) A general circular memory structure (b) Memory subsections for on-the-fly reconfigurability (c) The state restore methodology in each memory subsection (d) Data wrap around on every 9^{th} traversal of signal chain. 146
6.7 The function execution state machine. 147
6.8 Timing diagram for QRS configuration. 148
6.9 The MATLAB computed results of each PTA block in double and 8-b precision. (a,b) BPF. (c,d) Differentiation. (e,f) Differentiation zoomed about zero line. (g,h) Squaring. (i,j) Squaring zoomed about zero line. (k,l) Moving average. 152
6.10 Hardware computed PTA results (a) BPF. (b) Differentiation. (c) Squaring (d) Moving Average. 154
6.11 The comparison of MATLAB and hardware computed results with and without bit truncation along with the ECG signal. 155
6.12 The state-wise energy dissipation for target functions. 161
6.13 Percentage gate breakup of major components of the architecture. . 162
6.14 Percentage energy breakup of the architecture averaged over four reconfigurations. 163

A.1 Timing diagram of a horizontal scan. 197
A.2 (a) Pin diagram of VGA connector (DE15 port). (b) VGA cable . . . 198

A.3 (a) Pin diagram of RS-232 port. (b) RS-232 cable. 200
A.4 State of serial transmission line of RS-232 port when 0x55 is sent as data. 201

B.1 Depolarization and repolarization of single myocardial cell during the cardiac cycle along with potential variation and representative ECG. 204
B.2 Einthoven's triangle and its position in relation to heart's schematic. 205
B.3 Frontal plane limb leads. 206
B.4 Front view of the transverse leads and its position in relation to heart's schematic. 206

C.1 A conceptual diagram of four (3×3) SAC units connected together. 209

List of Publications

[1] **N. Jain** and B. Mishra, “A light-weight configurable architecture for QRS detection," in *IET Computers and Digital Techniques*, vol. 13, no. 2, pp. 70-77, March 2019.

[2] **N. Jain**, M. Singh, B, Mishra, “Image Compression using 2D-Discrete Wavelet Transform on a Light Weight Reconfigurable Hardware," in *International Conference on VLSI Design (VLSID)*, pp. 61-66, January 2018.

[3] **N. Jai**n and B. Mishra, “DCT and CORDIC on a Novel Configurable Hardware," in *Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PRIME-ASIA)*, pp. 51-56, November 2015. (Best Paper Award)

[4] **N. Jain** and B. Mishra, “CORDIC on a Configurable Serial Architecture for Biomedical Signal Processing Applications," in *International Symposium on VLSI Design and Test (VDAT)*, pp. 1-6, June 2015. (Women-In Engineering Fellowship)

[5] **N. Jain** and B. Mishra, “QRS detection algorithm on a novel reconfigurable multiplierless architecture," in *International Conference on Industrial Instrumentation and Control (ICIC)*, pp. 469-474, December 2014.

[6] V. Sharma, **N. Jain** and B. Mishra, “Fully-digital Time based ADC/TDC in 0.18Îijm CMOS," in *International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA)*, pp. 1-6, August 2016.

[7] S. Tripathi, **N. Jain** and B. Mishra, “Ultra low power 128 byte memory design based on D-Latch in 0.18 Îijm process," in *IEEE Asia Pacific Conference on*

Postgraduate Research in Microelectronics and Electronics (PrimeAsia), pp. 89-93, November 2015.

[8] B. Mishra, S. Thakkar and **N. Jain**, "Ultra Low Power Digital Front End for Single Lead ECG Acquisition Integrated with a Time to Digital Converter," in *IET Computers and Digital Techniques*. (Under Review)

[1]-[5] Publications out of this thesis.

[6]-[8] Publications out of other collaborative work.

CHAPTER 1

Introduction

Physiology is the science related to functioning of various systems of the human body that are often termed as ***physiological systems***. The cardiovascular, respiratory, nervous and digestive systems constitutes the major physiological functional systems of the human body [7]. Deterioration or failure in the performance of one or more of these systems have serious implications on one's life causing permanent damage to organs or fatality in long run. The biomedical signals often serve as primary indicators for functional integrity of physiological systems not accessible for direct observation and investigation as they are internal to the body. Biomedical signals are interpreted to extract information on various biological systems. The process of extracting information can be as simple as feeling the pulse on the wrist of a person or as complex as analysing the structure of internal soft tissues by an ultrasound scanner. Biomedical signals originate from different sources and processes occurring within the body. Signals generated by nerve and muscle cells are termed as ***bioelectric signals*** and represent the action potential caused by excitation of cell membrane potential under certain conditions. ElectroCardioGram (ECG) and ElectoEncephaloGram (EEG) signals are most common example of bioelectric signals. Bioacoustic, Biomechanical, Biochemical signals constitute other biomedical signals arising out of sound/flow, motion and concentration of ions, respectively, in various sections of body.

Medical instrumentation is used to measure or determine the presence of physical quantity that assist the medical practitioner to make better assessment and treatment decisions. This is done by measuring, recording and monitoring various biomedical signals. The majority of instruments used are electrical or elec-

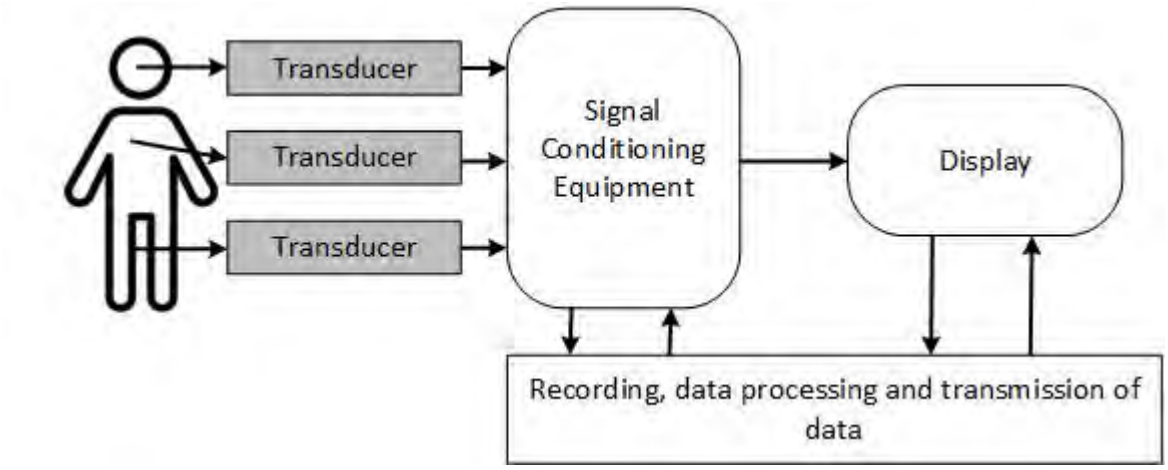


Figure 1.1: Block Diagram of Man-Instrument System.

tronic systems although mechanical systems like ventilators or spirometers are also employed. A generic medical instrumentation with its major functional blocks is shown in Fig. 1.1. The physical quantity under measurement is termed as ***measurand*** and is generated within the body. The primary use of the ***transducer*** in medical instrumentation is to generate proportional electrical equivalents of non-electrical phenomenons. For instance, Plethysmographs designed for constant pressure or volume measurement uses pressure or displacement transducer that responds to pressure changes within the chamber and provides a signal calibrated to represent the volume of the limb. ***Signal Conditioner*** Equipment acts on the electrical output of the transducer and amplifying, modifies or changes the signal making it suitable for operating the recording or display system that follows. The electrical output of the signal-processing equipment is converted to a form that can be perceived by physicians and consequently convey the information obtained by the measurement in a meaningful way using a ***Display System***. A graphic pen recorder generating a permanent record of data, for example, is a display system for ECG recorder. ***Recording, Data-Processing, and Transmission Equipment*** equipment ensures recording the measured information for storage or transmission from one location to another often desirable in medical systems. This equipment also performs the necessary data processing when a computer is a part of the instrumentation system.

Sensor networks and personal digital assistant based monitoring systems are

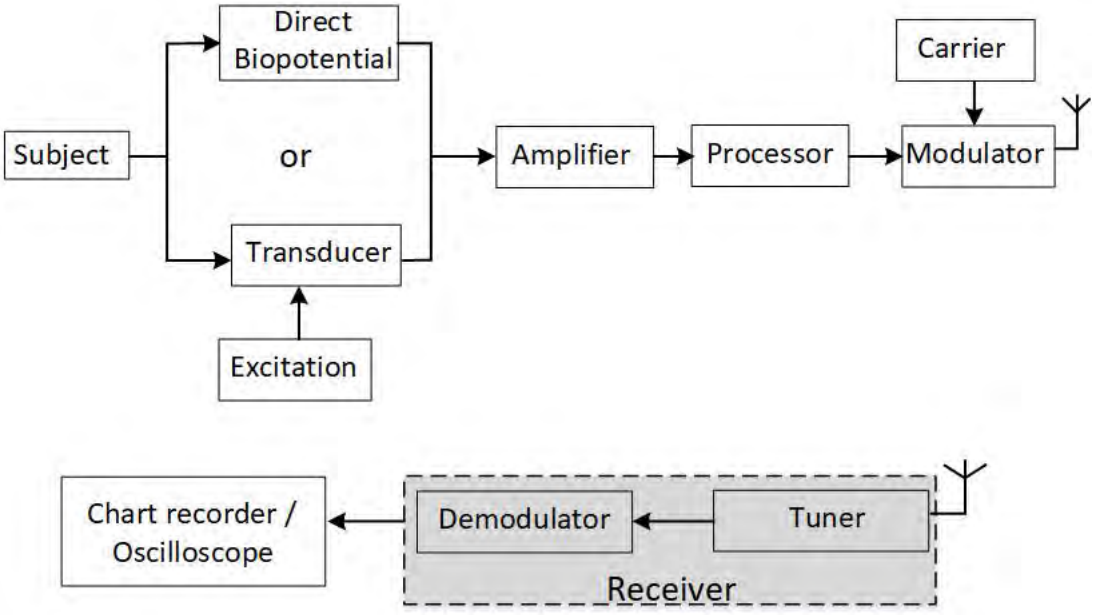


Figure 1.2: The Block Diagram of Biotemeletry Transmitter and Receiver.

increasingly becoming popular as data acquisition system and data manipulation system, respectively. Wearable sensors offer great patient mobility within and outside hospitals and are a cost-efficient solution.

1.1 Monitoring Systems

1.1.1 Patient Monitoring Systems

Patient monitoring systems use electronic equipments that record vital characteristics and parameters of the critically ill by means of continuous monitoring facilitating early detection with remedies taken before problems translate into severe damage. Cardiac-monitoring units are often employed in coronary care units as they can reliably measure the associated parameters. Other than critical care units, monitoring systems are also employed in operation theatres, catheterization laboratories *etc*.

1.1.2 Biotelemetry and Telemedicine

Biotelemetry System includes remote measurement and transmission of biological parameters. Various means can be adopted for the purpose of transmitting data from point of generation to the point of reception. One of the earliest evidences of such transmission is seen in 1903, when Einthoven used telephone lines to transmit electrocardiogram from hospital to his laboratory many miles away. However, technological improvements and innovations in telemetry over the years led to elimination of wires in modern telemetry systems predominantly based on radio transmissions. The basic functional blocks of a biotelemetry system is shown in Fig. 1.2. In the transmitter side, the physiological signal obtained from the transducer is passed through amplification and processing circuits and a modulation stage. The receiver selects the transmission frequency using a tuner and separates the signal from the carrier in the demodulation block. The obtained signal can be displayed on a chart recorder/ oscilloscope.

1.1.3 Ambulatory Monitoring

In recent times, ambulatory monitoring outweighs the conventional clinical or home monitoring systems due to its increased accuracy, reliability and diagnostic capabilities based on continuous monitoring of physiological signals throughout the day [8–10]. Studies in [11, 12] demonstrates patients exhibit an increased acceptance to ambulatory systems over home or clinical setup due to psychological ease and physical comfort in former the settings. Predominantly, the blood pressure and electrical activity of the heart, brain and muscles indicated by ECG, EEG and Electromyogram (EMG) potentials, respectively, along with PhotoPlethysmograph (PPG), blood saturation, heart sound, constitute signals analysed by ambulatory monitoring systems. The ambulatory monitors detect conditions ranging from critical cardiovascular diseases (CVD) such as cardiac arrhythmia, hypertension *etc.*, onset of epileptic seizures to evaluating effectiveness of preventive (and curative) measures like ongoing medication, pacemaker function. The first use of these monitors is reported in 1960s [13] and the real-time analysis of ambulatory

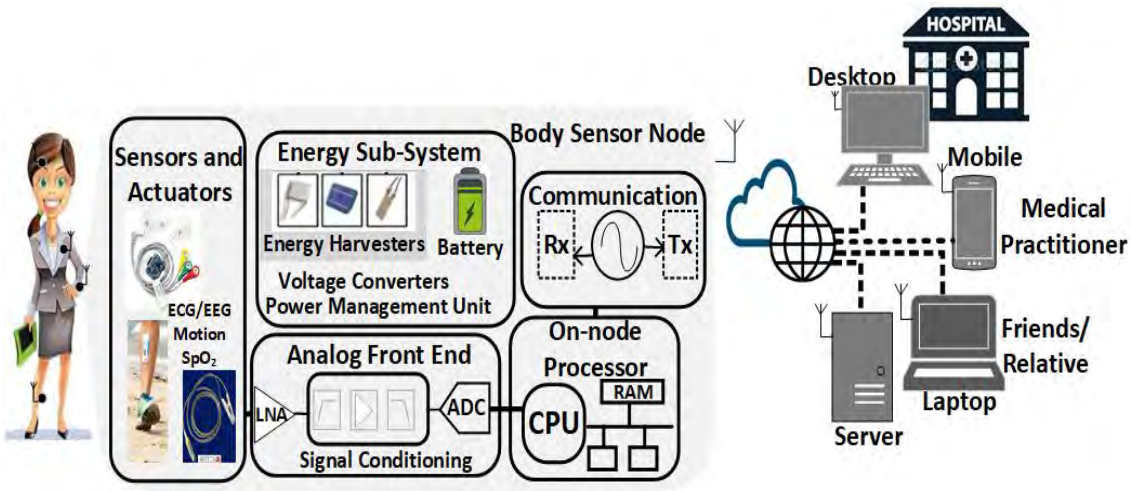


Figure 1.3: A Conceptual Generic Biomedical System.

monitoring system is reported in [14] twenty years later. Advances in integrated circuit technologies in recent times have also made miniaturization possible in these devices favouring patient convenience as a result. This promoted the use of ambulatory or biomedical devices in general for everyday monitoring as a part of wireless or body sensor network [15, 16] that engages data exchanges among sensor nodes over wireless interface.

Wearable Biomedical Systems

A generic biomedical device typically consist of sensors and actuators for signal acquisition, signal processing block, energy management unit and communication subsystems (Fig. 1.3). The biomedical signals interfaces with the physical environment by means of sensors and are processed on the signal processing block. These signals undergo data conversion, storage, feature extraction *etc.* as part of processing for further sending them to medical practitioners or database servers after necessary data conversions and amplification. The biomedical devices work with tight power budgets that directly translates into battery lifetimes and thus require efficient energy subsystems consisting of energy harvesters [17, 18] and energy management units [19, 20]. Apart from the low energy operation, these devices must exhibit small aspect ratio for the ease of wearability when deployed as wearable sensor nodes.

A wireless autonomous transducer system (WATS) proposed in [3] presents

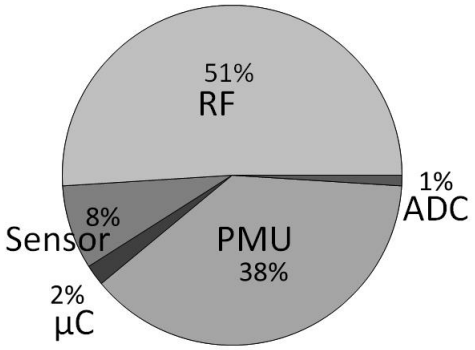


Figure 1.4: The power consumption profile for a WATS system based on off-the-shelf blocks for an ECG application [3].

custom models of RF harvester, analog to digital converter and radio transmissions. A study of ECG application using off-the-shelf WATS components establishes that radio and power management are the major power dissipating blocks consuming >85% of the total power (See Fig. 1.4). Limiting the radio transmissions to transmitting alerts and alarms favours the low power budget requirement of biomedical devices. The physiological signals must undergo processing in the signal processing block on the device that generates alarms and alerts indicating deviation from normal operation. This requires the signal processing block to perform computation intensive functions focussed on feature extraction in addition to noise removal, storage, data compression *etc*. The proportion and type of signal processing functions used in a few ambulatory biomedical applications is shown in Fig. 1.5 . Clearly, certain functions, say, Finite Impulse Response (FIR), Discrete Wavelet Transform (DWT), trigonometric operations, Fast Fourier Transform (FFT) *etc.* are the dominant signal processing functions for these applications. Dedicated ASIC (Application Specific Integrated Circuit) and hardware accelerators for these functions provide a cycle and energy efficient solution [21–23]. However, these platforms do not support further algorithm enhancement due to their dedicated nature. Reconfigurable Architectures (RAs) support architectural flexibility but the interconnect overhead amounts to high gate count. Biomedical signal processing domain specific RA is relatively unexplored and the general purpose RAs fail to leverage the inherent regularity in the biomedical signal processing algorithms.

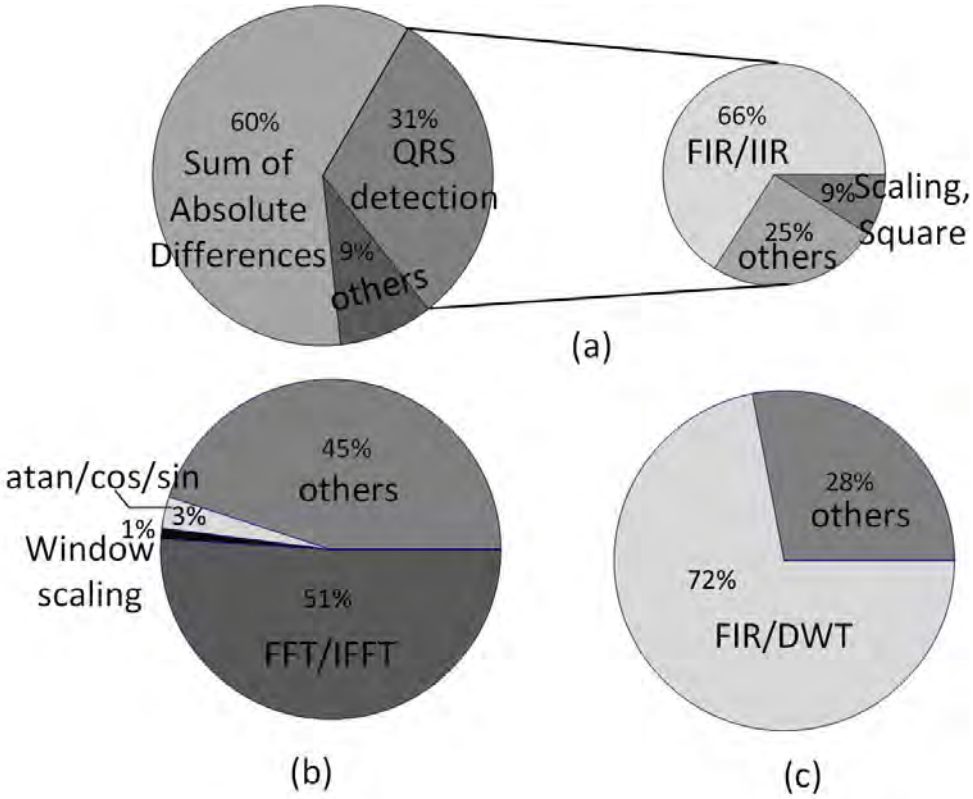


Figure 1.5: Proportion and kind of signal processing functions in (a) ECG Arrhythmia Classification [4] (b) Heart Sound processing [5] (c) Pulse Oximetry [6].

A light-weight and reconfigurable architecture is developed in this thesis targeting the biomedical signal processing application domain. The fundamental operation of the developed architecture is a shift-accumulation which results in a low gate count or light-weight platform primarily because of elimination of multipliers. The architecture has the capability to perform various digital signal processing functions by means of configurable datapath. Additionally, the architecture offers flexibility (coarse-grained) and various topologies for realizing functions and applications unaccounted for during design time. The mapping methodologies for target functions are developed considering optimized resource and cycle utilization thereby complying to the light-weight nature of the architecture. Due to the innate low performance nature of biosignal datarates, the on-the-fly reconfigurability aspect strikes an optimal balance between energy and throughput. The hardware realization of the architecture with various target functions mapped confirms functional integrity. As a proof of concept, the architecture is

realized on the Field Programmable Gate Array (FPGA) with energy and cycle quantifications in the circuit level simulations. The architecture is demonstrated with widely used biomedical application, QRS Detection in ECG signal, utilizing the on-the-fly reconfigurability feature. In the process of developing the configurable platform, several supporting interfaces are also implemented.

1.2 Thesis Contributions

The **key contributions of the thesis** are listed below:

- **Architecture Development:** The underlying operation of the developed architecture is shift-accumulation and hence is named as Shift-Accumulate (SAC) architecture. The architecture consists of systolic array of functional units (called Register Units or RUs) computing multiply-accumulate operation of thiry-six 8-b$\times$8-b binary numbers in a serial manner. The accumulation is carried out in Computation Unit (or CU) by means of an adder tree with the latency of 8 clock cycles. The architecture is made flexible by adding configuration elements in the datapath which makes mapping of multiple and diverse functions possible. The configuration scheme for the architecture is developed along with the input/output memories. The input memory serves as the data memory and has a circular structure which exhibits different interpretations while serving multiple function data requirements. The SAC architecture is ported on an FPGA as proof-of-concept and functionality for all the target functions is verified. The architecture configuration is achieved using a time multiplexed 9-b input bus. The control word for target functions and different architecture topologies is developed. The fields of control word can be configured independently which gives rise to numerous architecture topologies and thus enables efficient realization of a diverse set of functions. Two interfaces, VGA and RS-232, developed for visual perception and output communication between the FPGA and computer. It is observed, that the SAC architecture realization reports minimum gate count among dedicated or software-hardware implementations target-

ing similar functions.

- **Function Mapping Methodologies:** A set of functions often encountered in biomedical signal processing are targeted on the architecture and mapping methodologies are developed for them. The mapping methodology ensures optimum functional unit implementation by altering the datapath as well as increases the hardware resource utilization by storing intermediate results within the architecture eliminating excess memory and its access time requirements. In case of algorithm-driven functions, coefficient multiplexers are employed to provide fixed coefficients stored internal to the architecture. The regularity in the algorithm is exploited to map such functions with smaller state machines and cycle counts.

- **On-the-fly Reconfigurability and Application Demonstration:** The SAC architecture could support on-the-fly reconfigurability because of its ability to emulate individual functions at performance sufficiently large for reconfiguration while maintaining pace with the input datarates of biomedical signal processing applications. Furthermore, on-the-fly reconfigurability is incorporated within the architecture while keeping its interfaces unchanged by introducing a memory management methodology. The results for functions appearing in the midst of the function-chain are stored inside the memory thereby abiding to the light-weight nature of the architecture. The state machine controlling the execution of functions in the on-the-fly reconfigurable manner contains configuration, restore, compute, store states. The status of the function is captured and later restored (context switching) in the restore state beneficial particularly in windowing functions. A modified QRS detection algorithm is mapped on the architecture realized on FPGA as proof-of-concept. Additionally, the architecture is characterized for energy, gate count and cycle count in the circuit level implementation. The architecture is compared on the basis of gate count and energy with other hardware implementations realizing QRS detection. The architecture offers comparable energy and ≈ 10 times reduced gate count in comparison with other literature.

1.2.1 Thesis Organisation

A large number of studies focussed on low power signal processing module across ASIC, FPGA and micro-processor platforms (μP) with computational abilities at par with on-node processing required in biomedical devices are discussed in Chapter 2. Additionally, system requirements of a biomedical device is analyzed.

The SAC architecture is presented in Chapter 3 along with its organisation. The functional units of the architecture are briefly discussed in section 3.2. The working of the architecture is explained next along with its latency. The architecture is later analyzed for configurability (Section 3.2.4) on examining DSP functions commonly encountered in biomedical signal processing applications. Following the configurability, the various topologies of the architecture is discussed in section 3.3.

In Chapter 4, the target DSP functions are analyzed on mathematical as well as hardware implementation aspects and mapping methodologies are discussed for these functions. The functional units involved in mapping target functions are identified and their state machines are presented. The hardware implementation of the architecture is discussed in Chapter 5. The functional verification of the hardware results is presented along with the clock and gate count profile.

A common biomedical signal processing application focussed on ECG analysis is mapped on architecture in Chapter 6. The QRS detection algorithm is analyzed and modified for mapping on 3×3 variant of the architecture. In the process, the concept of on-the-fly reconfigurability is explored and applied on the architecture. The clock and gate count of the architecture are estimated and compared with literature dedicated to ECG processing. The circuit level simulations of the 3×3 SAC architecture are carried out with clock and energy quantifications of individual functions and application realization. Chapter 7 includes conclusion and future work.

CHAPTER 2

Literature Review

2.1 General introduction to Wireless Sensor Networks (WSNs)

The origin of WSNs can be attributed to the Sound Surveillance System developed by the United States Military in 1950s to detect and track enemy submarines. The United States Defence Advanced Research Projects Agency (DARPA) initiated the Distributed Sensor Network (DSN) program in 1980 exploring the challenges and constraints in implementing distributed sensor networks. Since then, fuelled by the Government and Universities efforts, WSNs found home in academia and civilian scientific research.

Various fields of applications of WSNs are shown in Fig. 2.1. Smart Dust [24,25] platform is increasingly became popular for majority of WSN applications. Smart Dust motes along with its supplementary TinyOS (Tiny Microthreading Operating System) software operating system can be deployed easily as well as work well under the constraints of power, size and cost. Few of the pilot projects of precision agriculture and animal tracking are Zebranet [26], Great Duck Island [27], Minerva [28, 29] and railway tracking [30] to avoid animal fatality due to accidents, understand their interaction patterns through activity tracking and monitoring. [31] presents the applicability of WSNs in industrial scenario in the form of controlling various processes, early detection of equipment failure *etc.* by means of monitoring various physical and environmental parameters. The WISE-WAI [32] project by University of Padova aims to monitor and control traf-

fic, pollution, habitat and environment as well as provide real-time assistance to fireman and rescue squads, open places surveillance *etc.* The real-time operation and response of WSNs are gaining popularity because of its suitability for interactive monitoring of systems leading to efficient control of operating conditions. Authors in [33] present various application scenarios of WSNs operating in real-time.

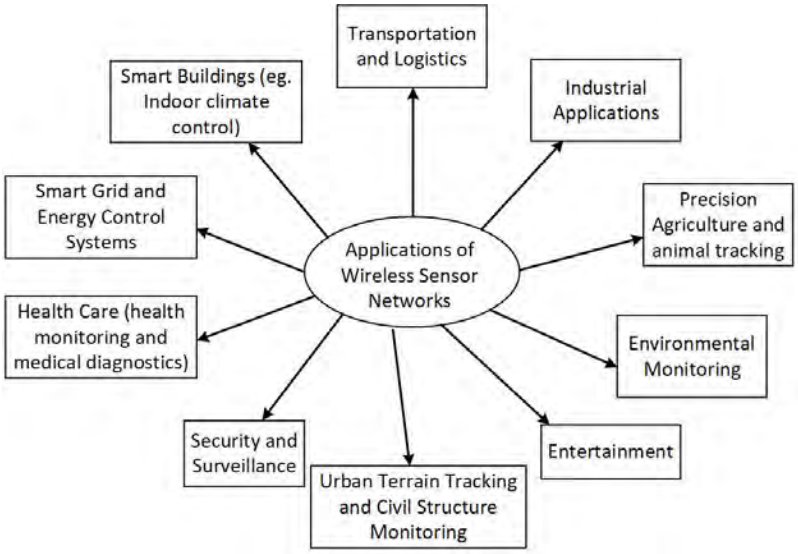


Figure 2.1: Various fields of Application of WSN (Adapted from OECD).

2.2 Wireless Body Area Networks (WBANs)

Use of WSNs in healthcare has brought revolutionary effects in the form of improved (practically continuous) patient's access to the care givers approaching the necessary framework for pervasive health care. Human body monitoring, also termed as Body Sensor Network (BSN), includes a system that responds to and interacts with its external surroundings in a self-contained manner using a network of wireless sensors attached inside/outside the body. Owing to the critical physiological processes and the associated crisis involved, BSN is expected to necessarily exhibit greater robustness, accuracy, energy-efficiency, bio-compatibility, real-time processing and early event detection capabilities.

A BSN contains a number of portable, miniaturized, and autonomous sensor nodes in , on and around the human body that monitor the body function for

sporting, health, entertainment, and emergency applications. The BSN applications can be categorized into medical and non-medical domains. The medical domain includes wearable and implantable BSNs which assess, monitor, detect and provide feedback making the patient aware of probable organ damage, episodic events, allergic agents aiding better rehabilitation with timely analysis and treatment. Additionally, it includes network of remote controlled medical devices building the framework for self-conducted care of patients by administering various conditions of recovery process, assisted living, ambulatory monitoring *etc.* The non-medical applications of BSN includes gaming, social networking, emotion detection, video/audio streaming, emergency detection (fire/poisonous gas detection), security authentication *etc.*

2.2.1 Constraints of Wireless Body Sensor Nodes

We now discuss the general requirements [34] of a typical sensor node in BSN platform. The sensors employed in body networks are generally mobile and consequently are battery powered. The target lifetime must be upto 5 years for implantable sensors and couple of weeks for wearable sensors. The target battery lifetime can be increased by adopting various energy scavenging techniques [35] (*e.g.* solar, thermal, motion and vibration), energy-efficient MAC, routing and data dissemination protocols or wireless charging technologies *etc.* Alternatively, on-node processing replacing the raw data transmission indicates a significant reduction on the energy consumption of the sensor node. BSN operates in a multiple sensor nodes scenario sensing a range of data on a subject as well as dealing with multiple subjects simultaneously. This requires the sensor nodes to co-exist with legacy devices in a physiological environment while maintaining the security and privacy of data. Other highly desirable features include re-programmability, reconfiguration, customization, bio-compatibility and ergonomic ease. The aforementioned requirements gives rise to a number of crucial constraints that requires due consideration while designing a body sensor node. On the system level, the sensor node must exhibit on-node processing with re-programmability capabilities in an energy-aware manner while being light-weight. Furthermore owing

to the advances and innovations in Integrated Circuit (IC) technologies, an integrated chip with sensor, processor unit and antenna in close proximity yields a favourable form factor, cost-efficient and setup and procedure time-compliant implementation, in general.

2.3 System Specifications

The body sensor node typically consists of sensors, data conversion components (ADC/DAC, low noise amplifiers *etc.*), energy subsystem (primarily DC-DC converters, power sources and energy harvesters), data processing block (processor, memory, interfaces) and communication module. The data processing unit plays an important role in the overall performance and power budgeting of the body sensor node system. Over the years various signal processing platforms are considered and adopted catering to data processing requirements of BSN, which, generally speaking, are limited to light signal conditioning, feature detection, extraction *etc*. The reason for limited sensor node processing capabilities is primarily the tight power budget of the device which allows room only for low complexity function computations. This has motivated researchers to develop architectures that can perform light signal processing activities in a power-aware manner in recent years. Before moving on to the various signal processing solutions, a brief overview of power, performance and target function domain for the biomedical applications is presented, with special focus on body area (or ambulatory) applications.

2.3.1 Power/Energy Profile

The power and energy budget of the biomedical application has unprecedented affect on its battery lifetime. The performance profile of certain biomedical applications is presented in [36]. Among them pacemaker [37], hearing aid and cochlear processor [38] represent the traditional applications in the biomedical domain where associated techniques and methods are well matured. The primary concern for pacemaker devices is the power consumption and the battery is ex-

pected to yield years of operation. The typical power consumption of pacemaker lies in tens of μW. The batteries of hearing aids and cochlear processor are relatively easier to access and provide a few weeks of uninterrupted operation. Body area monitoring [39] is an emerging application in battery powered domain with $\approx$140μW power consumption. Table 2.1 lists a few recent BSN systems emphasizing the low power requirements of the devices. As can be seen, the applications mostly require 8 to 12-bit processing with frequency of operation ranging from kHz to few MHz. Furthermore, the power consumption lies within tens of mW to hundreds of μW.

Table 2.1: Performance specification of biomedical signal processing applications.

Application	Power	ADC	Processor
Beat Detection [40]	71μ	12-bit	DSP, 1MHz
Arrhythmia Detection (frequency domain) [41]	72μ	8-bit	μC*, 10kHz
R-R Detection [41]	92μ		
Congestive Heart Failure (PPG SoC) [42]	336-1554μ	9-bit	low power μC*
Capsule Endoscopy [43]	6.2m	8-bit	ASIC, 40MHz
Universal Sensing Module [44]	7.4m	8-bit	low power μC*, 32.768kHz
Physical activity Monitoring [45]	-	12-bit	MCU*, 38.4kHz
Anesthesia Depth Monitoring [46]	25.2m	6-bit	DSP, 50 MHz

* represents commercially available devices

The sensor node communication interface is the heart of the system as it is responsible for secure and reliable data exchanges. However, commercial low power radios available now a days are not able to meet the stringent BSN requirements [47,48]. A commercial radio based BSN system discussed in [49] establishes that the current consumption during the radio transmission in active mode translates to $\approx$90% of the total system power. Another study in [3] presents power profile of a wireless system while recording and evaluating the ECG signal. The radio consumes 51% (= 558.96μW) of the total power when the system is modelled using off-the shelf components for raw data transmission. This propelled the research and innovation in two directions. First, exploring techniques and designs for custom low power radios that support ad-hoc ultra low power wire-

less networking. Second, reducing the active mode fraction of radio by limiting the transmission data leading to rigorous duty cycling. The simplistic approach to achieve the latter is on-node signal processing. Additionally, combining the aforementioned two approaches furnish increased energy efficiency at multiple levels provided the power consumed in on-node processing does not exceed the power savings from the radio duty-cycling.

The said power trade-off is analyzed by means of a few studies reported in literature focussed on evaluating the net effect on power in the two scenarios, namely on-node processing and slackened duty-cycling of radio. Authors in [40] analyze three scenarios for wireless transmission modes with the primary objective of estimating the power saving as a result of on-node signal processing. The first mode represents streaming raw ECG signal at 256 *Hz*. The filtered ECG signal is transmitted in the second mode at the same sampling frequency. The third mode includes involved signal processing algorithms aimed at accurate R-peak search and heart-beat detection transmitting only the heart rate at 1 *Hz*. The overall system power consumption adopting the third mode of transmission in 1.04 mW reporting an improvement of 19$\times$ and 12$\times$ over first and second transmission modes. This translates to upto one month additional lifetime with a standard 400 mAh battery thereby increases the autonomy of the system. Another study in [50] reports 95.6% energy reduction while sending heart-rate only as compared to raw data streaming. Although custom low power radios architectures are reported in [51–54], their inclusion in the system requires precise synchronization and necessary overhead which adds to the power cost.

2.3.2 Performance

Sensor nodes in wearable applications usually monitor biological functions and environmental conditions. The data requirements of majority of such sensors are modest in general, since both the resolution as well as the update rates are low. Table 2.2 summarizes the data rates and frequency range of few examples. It is evident that the signal data rates are quite low, with a maximum of $\approx$ 300kbits/s. This further implies moderate to low clock rates for the process-

ing platform performing data manipulations. The same is also evident from table 2.1, where the processor operating frequency attains a maxima of tens of MHz for biomedical applications. However, it can be observed that further lower clock (tens of kHz) frequencies are reported for applications specific to ambulatory monitoring [40, 41, 44, 45]. This is further substantiated by the low sampling rate of standard databases for biopotentials. The various ECG databases: MIT BIH arrhythmia [55, 56], Long-Term ST segment and ANSI/AAMI EC13 test waveforms/databases haves sampling rates 360 Hz, 250 Hz and 720 Hz, respectively. The EEG database presented in [4] is sampled at 256 Hz. The MIMIC database [57] for PPG exhibits the sampling rate of 125 Hz. In general, low performance nature of ambulatory applications renders the associated electronics to be operated at sub/near threshold region resulting in low power operation, much desired in the BSN domain.

Table 2.2: Typical data rates and bandwidth of human biopotentials and biophysical signals [1]

	Data rate (bit/s)	Bandwidth (Hz)
ECG (12 leads)	288k	0.1-1000
Electromyogram	320k	5-10000
EEG (12 leads)	50k	0.1-100
Movement	35k	0-500
Temperature	120	0-1
Blood Saturation	16	0-1
Blood Pressure	16	0-1

2.3.3 Functionality

Various signal processing functions are used in biomedical applications for signal conditioning, feature extraction, classification *etc*. However, because of the desired limited signal processing capability of a body sensor node, a subset of DSP functions suffice for the task. Table 2.3 presents the survey of DSP functions used in biomedical applications performed in ambulatory setting. Majority of the tabulated applications report the use of Finite Impulse Response or Infinite Impluse Response (FIR or IIR) filtering adopted primarily for noise removal and occasionally in feature extraction as well. The differentiation function is highly used to

track the variation pattern of signal by means of slope information. Moving average, median and maxima functions are used to extract the peak information of a signal. The transform domain functions, Fast Fourier Transform (FFT) and Discrete Wavelet Transform (DWT) are used at large for analysis of signal in the frequency domain. The classical approach of estimating blood saturation involves the Discrete Cosine Transform (DCT) computations. Additionally, the DWT and DCT algorithms are reported to be used for the signal compression purpose to reduce the transmission share of the radio. Furthermore, complex trigonometric function computations is reported alongside the fundamental operations of multiplication, addition, scaling, division *etc*.

Table 2.3: Digital signal processing functions in ambulatory biomedical applications.

Signal	Application	DSP functions
ECG	Onset and Duration of QRS [58]	Infinite Impulse Response (IIR), Magnitude of vector ($\sqrt{\cdot}$, $(\cdot)^2$, addition), division, median
	QRS Detection [2]	IIR, Finite Impulse Response (FIR), $\frac{d}{dx}$, $(\cdot)^2$, moving average
	QRS Detection [59]	FIR, $\frac{d}{dx}$, $(\cdot)^2$, weighted moving average
	QRS Detection [60]	running Slope, multiplication, scaling, average, maxima
	ECG fudicial points [61]	Discrete Wavelet Transform (DWT), local maxima modulus
EEG	Epileptic Seizure onset [4]	FIR
	Seizure Detection [62]	DWT
	Automatic recognition of alertness [63]	DWT
PPG	Pulse Oximetry [6]	FIR, log
	Blood Saturation [64]	Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT)
Heart Sound	Auscultation Aid [5]	FFT, tan, sin

In general, μP does not readily support $\sqrt{(\cdot)}$ and log functions and thus require extensive software emulation. The CORDIC algorithm supports these functions along with trigonometric, FFT, division computations utilizing simple shift-add operations. Clearly, it can be observed that FIR or IIR, CORDIC, DWT, DCT,

differentiation and moving average form the dominant operations encountered in ambulatory biomedical signal processing. As a consequence, these operations constitute the target domain functions for a signal processing platform and confirms a variety of application realization on platform supporting these functionalities.

In conclusion, the BSN system requirements include 8 to 12-bit processing capability with the operating frequency ranging from k*Hz* to few M*Hz*. The signal processing algorithms used in the BSN systems exhibit regularity in terms of the kind of DSP operations. These operations include FIR, CORDIC, DWT, Moving Average, FFT *etc.* The BSN device signal processing platform is required to support these functions with the said performance in an energy aware manner. In the following section, various signal processing platforms used in biomedical signal processing are discussed and evaluated upon aforementioned requirements.

2.4 Various Body Sensor Node Systems

The characteristic features of flexibility, energy-efficiency and performance play pivotal role in choosing the signal processing platform among Application Specific Integrated Circuit (ASIC), micro-controller/micro-processor (μC/μP) and Reconfigurable processors/Field Programmable Gate Array (FPGA). On one end of the horizon lies the ASIC, that exhibits greater energy-efficiency as compared to the other two alternatives owing to the direct matching between the target application and hardware. Additionally, it also provides the highest performance owing to minimum cycle and gate overheads. However, in general, ASIC implementation is not cost-efficient unless manufactured at large scale because of the custom development, sophisticated fabrication and longer design time. On the other end lies the micro-controllers that are highly flexible and cost-efficient but lacks the energy-efficient operation (three orders less as compared to ASIC) due to generic instruction set architecture (ISA) and increased execution time. The middle ground is owned by the reconfigurable processors and FPGAs, that provide the combination of performance benefits of hardware implementation and

flexibility of software realizations at the cost of increased overhead and routing. The energy-efficiency of the reconfigurable processors is not as high as ASIC because overhead and routing are equally dominant as the logic realizing blocks but it can be as high as two orders of magnitude as compared to embedded processors. Intermediate approaches include Application Specific Instruction Processors (ASIP) and Digital Signal Processors (DSPs) which are energy-optimized for certain classes of applications by using a mix of dedicated hardware and instruction set.

Literature reports extensive use of all the three aforementioned platforms as signal processing solutions in biomedical applications. [21, 65–70] are some of the ASIC implementations of the ECG based processor, primarily computing ECG fudicial points followed by detecting and classifying arrhythmias. The former two uses novel algorithms based on wavelet transform and level crossing techniques for QRS detection in an ECG signal. The derivative based Pan-Tompkins algorithm is adopted in [21] wherein each of the signal processing step is implemented as a dedicated module. The implemented ASICs in these works are dedicated hardware power optimized exclusively for the purpose they are designed for, QRS detection in this case. These hardware do not support any architectural alterations arising out of new algorithm/technique development/optimization due to their application specific nature and thus do not present a lucrative picture alongside the ever advancing research and development efforts. Nevertheless, all of them report low power operation for the target application of QRS detection because of low performance operation (few hundreds of Hz) and hardware matched realization of computation intensive functions which otherwise would require increased clock cycles (consequently, power) on an alternate implementation, for example μP based. However, this was achieved at the cost of flexibility. Other studies focussed on ambulatory processing using ASIC based platforms include [23,71]. The application under consideration for these studies is EEG processing along with feature classification and tend to lie on slightly higher side of the performance scale *i.e.* operating at tens of MHz. The fundamental signal processing blocks in these chips include FIR filtering, multiplication, addition,

matrix multiplication, CORDIC engine (square root, magnitude and angle computations), division *etc.*

Often simultaneous monitoring and combined analysis of multiple parameters aids in better diagnosis and treatment. For instance, joint analysis of EEG with heart rate variablility (HRV) is preferred for seizure detection [72], sleep apnea [73] *etc.* The joint monitoring is an attractive feature in the context of WBANs as the functional domain of the node now encompasses multiple physiological parameters. This has attracted the attention of researchers who in [74] and [75] present fully custom multi-processor SoCs for bedside monitoring and brain-heart monitoring, respectively. The SoC in [74] implements dedicated engines for monitoring brain, heart and basic physiological signals. In addition to these, the later incorporates a dedicated compression engine to reduce the transmitted data bits. Though these solutions target a wider set of biomedical applications, they use dedicated engine for every added functionality which requires sophisticated control and synchronisation among various signals, typically having different sampling rates. Albeit, the target application domain of these solutions is wider than the single application platforms, they remain inflexible and application specific.

Various μP based biomedical signal processors are reported in [76–79] that exhibit increased flexibility as compared to the ASIC implementations and low power operation. Other body area network pilot projects based on commercial motes (For example, Telos, MicaZ, Pluto, BSN node, TmoteSky *etc.*) consisting of low power μPs like TI MSP430F149, Atmel Atmega 128L *etc.* include Code Blue (Harward University) [80], MIThril 2003 (MIT) [81], BSN node (Imperial College), disaster aid network (John Hopkins University) [82]. They use power aware and lightweight operating systems (OS) (for example, TinyOS) that readily support hardware emulation. The use of neural network for anomaly detection in ECG on a custom processor is discussed in [83]. Alternatively, use of personnel digital assistant (PDA) is reported in [81] to manage and process sensor data. Low power operation is extracted by means of multiple power domain operation of the μPs that can be software programmed. For instance, CoolFlux BSP adopted

in [76] has 15 power domains. Additionally, the μPs employ techniques of clock gating, dynamic voltage and frequency scaling and duty cycling to reduce power consumption at multiple abstraction levels. Though, flexibility offered by μPs support realizing multiple application realization, easy sensor interfaces, in-situ programmability, they are less energy-efficient as compared to hardware implementation while computing computation intensive tasks, in particular. For such computations, they tend to consume large number of clock cycles which adversely affects the energy-efficiency as they remain active is over an increased spread of time. The traditionally existing approaches of hardware-software co-design can be beneficial which combines the advantages of both paradigms. The control related activities and decision making processes are designated to the software whereas computationally intensive functions are dealt by the hardware that, in general, provides optimized implementation at multiple facets, namely, hardware resources, energy and cycle count.

A hardware-software co-design based ECG processor presented in [40, 50] bifurcates the processing into three pipelined stages, namely pre-processing, classification and post-processing. The classification stage is implemented in 16-b RISC whereas dedicated hardware is used for the other two stages. The flexibility in the classification stage permits varied conditions to be evaluated which can be fine tuned according to patient requirements. Additionally, it allows analysis of multiple parameters simultaneously (for example, atrial and ventricular arrhythmias) which provides a holistic view necessary for better assessment. The dedicated hardware for pre/post processing stages ensure fast computation of complex operations involved in feature extraction, encryption *etc*. Another study in [84] presents a custom biological signal processing unit which functions alongside a general controller, as a co-processor, enhancing its signal processing and compression capabilities. The co-processor provides time-domain and transform-domain functions and contains a configurable DWT engine. The configurable DWT engine enables targeting multiple mother wavelet functions which can be used to extract different features of the signal under interest.

Another family of systems alongside the processors are the hardware acceler-

ators that are used to improve the computation power of the processor. The use of accelerators in biomedical signal processing domain has largely captivated the attention of researchers in the last decade. The low performance nature of biomedical application encourage aggressive voltage scaling, which results in energy and cycle efficient solution out of dedicated accelerator implementations. Authors in [22, 41, 50, 85, 86] reported use of various accelerators yielding energy benefits of as high as 50% as compared to a non-accelerator implementation. However, the choice of accelerator is highly determined by the application under consideration. General practice followed on this aspect is, the operation identified with more computation time and contribution in the function profile is most likely to be realized as an accelerator. For instance, compression accelerator [50, 85] are used in applications with large data transmission requirement, FFT accelerator [41, 86] employed for applications dominated by transform domain analysis, noise dominated application use FIR filtering accelerator [85] *etc.* This provides an energy-efficient realization for the target application, however this solution becomes inflexible as the accelerators are dedicated and optimized only for target application parameters. Additionally, accelerators only optimizes certain part of the application and the remaining application still continues to runs on the μP in an inefficient manner.

Researchers in [22] addresses this issue by identifying and developing accelerators for common signal processing functions predominantly observed in biomedical signal processing applications. These accelerators could be connected in any sequence to realize multiple varied algorithms. This presented a configurable energy-efficient solution with enough flexibility to map a large set of biomedical applications while allowing algorithm advancements to some extent. The role of the μP was limited to control and configuration tasks among various accelerators and peripheral interfaces. However, the accelerators individually may be accessed for little or no time in an application and remain idle during majority of run-time. Additionally, multiple accelerators increase the overall gate count of the system which adversely affects the power in the low voltage (sub-threshold) regime [87]. Furthermore, increased gate count increases the

area which adversely affects the form factor as well as cost of the device. Authors in [87] analyzed the effect of gate count on power and critical path of an architecture in the sub(near)-threshold region and reported the energy benefits of low gate count design over its high gate count equivalent. Additionally, the performance/throughput advantage of light-weight architecture is presented in [88] which enables real-time processing of large and complex computations.

The average access time of the accelerator can be increased by adding the feature of hardware reconfigurability. This combines the benefits of hardware implementation with the software flexibility to some extent. In this approach, single accelerator (re)configures itself to perform multiple functions of an application in a time-multiplexed manner. Furthermore, reconfiguration ensures a wide functional domain for the accelerator with the ability to target varied algorithms. However, (re)configuration is obtained at the expense of overhead, which in fine-grained architectures proves daunting for applications working in a power-frugal environment. In addition to this, the pitfalls of low/medium performance and poor area realization become pronounced when fine-grained architectures are used to implement word-level data processing. Coarse-grained reconfigurable architectures (CGRAs) are more suited for such applications which supports word-level operations on optimally designed processing elements (PEs) and special-purpose interconnections retaining enough flexibility for mapping varied applications. Due to their coarse-grained granularity higher performance, reduced reconfiguration overhead and lower power consumption than the fine-grained architectures is achieved [89]. When dealing with domain specific processing, the coarse-grained architectures typically consume less energy as compared to low power Digital Signal Processors (DSPs) owing to faster processing *i.e.* computations over a short time interval.

CGRAs have been previously proposed for multimedia, communication, embedded and DSP applications. In these works, the researchers efforts were focussed primarily on attaining enhanced performance rather than on reducing energy consumption. Use of CGRA as a processing platform in the wearable and implantable biomedical devices has received little attention over the years. The

applicability of certain existing CGRAs in the biomedical signal processing domain is now presented. Single Instruction Multiple Data (SIMD) based CGRAs, MorphoSys [90], ADRES [91], CGRA Express [92], render efficient operation for vectorized algorithms but are ineffective for signal chain or concatenated operations. SmartCell [93] architecture support algorithms that have irregular access pattern, like Fast Fourier Transform (FFT), but is power and area hungry owing to dense interconnections. REMARC [94], AMBER [95] perform floating-point operations and report large functional unit area, thereby are less suited for biomedical devices from the form factor and power perspectives. Authors in [96] demonstrate the seizure detection application on the SYSCORE architecture [97]. It accepts four inputs in every functional unit and support logic operations like MULT-ADD, MULT-SUB, compare because of their significance in feature extraction process. The architecture attains low power operation by voltage scaling and collectively reducing the intermediate memory accesses and logic switching. [98] discusses the CGRA features necessary for real-time processing of biological signals and demonstrates the same through use-cases. Another architecture, Ultra Low Power Samsung Reconfigurable Processor (ULP-SRP) with a CGRA component reported in [99] is explored for ExG applications in [100] with the objective of increasing the computational power at high performance in an energy-aware manner suitable for involved on-node biomedical signal processing. The architecture adopts multiple power domains, power gating, unified memory techniques for reduced power consumption. The mapping on both these architectures require elaborate data flow graphs and dedicated Software Development Kit (SDK). This binds the efficacy of these platforms to the effectiveness of the compiler, which partitions the target application to extract maximum parallelization, performance mode selection, kernel mapping, memory accesses *etc*. Another architecture in [101] consist of 8 RISC processors with a sophisticated synchronizer programmed by means of software ISA. The SYSCORE and ULP-SRP contains $\approx$2500k and 500k NAND2 equivalent gates, respectively, which contributes to tremendous leakage power when coupled with aggressive voltage scaling.

From the functionality analysis of BSN systems, it is clear that the biomedi-

cal application algorithms possess an inherent regularity in terms of the nature of DSP functions encountered in them. This aspect is predominantly neglected in biomedical domain-specific CGRAs reported in literature [97,99,101] that primarily focus on increased flexibility necessary to emulate varied applications achieved at the cost of increased gate count. The regularity of biomedical signal processing algorithms is exploited in this work which enables restricting the extent of architectural flexibility to encompass frequently occurring DSP functions in algorithms of interest. The limited flexibility results in reduced overhead and consequently, a light-weight architecture is obtained.

Within this purview, in this thesis a light-weight architecture with configurable datapath is developed which renders gate count benefits. The architecture is based on shift-accumulate operation and follows serial execution. The mapping schemes for DSP functions found dominant in biomedical signal processing are developed. The mapping schemes ensures increased hardware utilization, reusability and reduced memory accesses. Additionally, the developed architecture has simplistic configuration scheme having little configuration time with tightly coupled memory. The architecture supports on-the-fly reconfigurability which renders signal chain emulation with provision for context switching. Furthermore, the architecture is based on the dominant application kernel exhibiting regularity, repetitive execution, context-aware switching ability at performance sufficient for biomedical signal processing domain.

CHAPTER 3

Reconfigurable Shift-Accumulate (SAC) Architecture

A novel configurable architecture, called the shift-accumulate (SAC) architecture, as shift-accumulate operation constitutes its underlying principle is presented in this chapter. The developed architecture performs multiply-accumulate operation in a serial multiplier-less manner yielding a light-weight platform with gate count advantages. The configurable datapath of the architecture enables multiple function realization on the same platform as opposed to multiple dedicated hardware unit implementation. Conceptually, this further boosts the gate savings advantage. However, the number of components used to achieve configurability is crucial in maintaining the light-weight nature. The judicious placement and use of these components has resulted in optimized datapath and resource utilization of the architecture. The gate count of building blocks of architecture and components aiding configurability are reported individually, bringing out the effect of configurability on an otherwise dedicated hardware implementation. The discussion related to various fields of control word used to configure the architecture is presented. Additionally, the mathematical framework leading to architecture development is presented along with various topologies it exhibits.

3.1 Underlying Mathematics

The mathematical equation of $y[n]$, output of N-tap FIR filtering function with inputs $x[n]$ and coefficients b_i is given by Eqn 3.1

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + \cdots + b_{N-1} \cdot x[n-(N-1)] \tag{3.1}$$

Assuming the standard 8-b binary processing, the coefficient b_i in Eqn 3.1 can be expressed as bit value, b_{ik} ('0' or '1'), and bit weights (2^k) where k varies from 0 to 7 resulting in Eqn 3.2.

$$y[n] = \sum_{i=0}^{N-1} b_i \cdot x[n-i] = \sum_{i=0}^{N-1} \left[\left(\sum_{k=0}^{7} b_{ik} \cdot 2^k \right) x[n-i] \right] \tag{3.2}$$

The direct hardware implementation of Eqn 3.1 (Method I) requires N 8-b×8-b multipliers and (N-1) 16-b adders. Alternatively, a multiplier-less realization would utilize 64 AND gates and 7 16-b adders in place of one multiplier. Eqn 3.2 (Method II) offers hardware optimization over Eqn 3.1 by replacing each multiplication operation by 8-b×1-b AND gate and 16-b shift-accumulation. The generated multiplication results are accumulated in adder tree producing the final output.

$$y[n] = \sum_{k=0}^{7} \left[\left(\sum_{i=0}^{N-1} b_{ik} \cdot x[n-i] \right) 2^k \right] \tag{3.3}$$

A simple rearrangement of Eqn 3.2 (Method III) results in Eqn 3.3. Here, AND matrix is used to generate the partial products. The common bit weight partial products are accumulated and shift operation is used when partial product sum of different bit weights is added together, generating the FIR output. As opposed to N shift-accumulators used in Eqn 3.2, only one shift-accumulator is required to implement Eqn 3.3. Additionally, in Eqn 3.3 a smaller (in terms of number of bits) adder tree is adequate to accumulate N 8-b partial products as compared to 16-b adder tree of Eqn 3.2. The comparison of three methods and resultant gate savings are also tabulated in Table 3.1 along with discussion in next subsection.

3.1.1 Optimized Interpretation

The hardware implementation of Method I translates into N multipliers and (N-1) adders and results in an entirely parallel realization with no latency but low hardware utilization. The unoptimized 8-b$\times$8-b array multiplier contains 64 AND gates and 7 16-b adders or 1808 equivalent NAND gates. Here, XOR, OR and AND gates are accounted as 4, 3 and 2 NAND gates, respectively. Furthermore, (N-1) 16-b adder contributes 240 equivalent NAND gates each, resulting in 2048N-240 total gates. Method II is a mixed architecture where the partial products of multiplication are generated and accumulated in a serial manner but the subsequent accumulation of multiplication results is done parallelly. Serial computation of multiplication reduces and reuses the hardware increasing its utilization factor but has 8 cycles latency. In this case, 8 AND gates and 16-b shift-accumulator (SA) *i.e.* 16-b adder and 16-b register constitute a multiplier. The multiplication results are accumulated using (N-1) 16-b adders yielding NAND equivalent gate count of 496N-240 with 16N additional FFs for Method II implementation. Method III realization is serial with latency of 8 cycles. The partial product generation is done in a similar way as Method II, however the common weight partial products are accumulated using 9-b adder tree containing N adders. The final accumulation uses one 16-b SA. This method uses 143N+240 gates with 16 FFs for its realization and provides 13.43$\times$ and 3.47$\times$ saving over Method I and II, respectively.

Table 3.1: The comparison of Methods I, II and III suited to implement multiply-accumulate equation.

	Method I	**Method II**	**Method III**
Nature of Execution	Parallel	Mixed	Serial
Partial Product Generation	64 AND	8 AND	8 AND
Partial Product Accumulation	7 16-b adders	SA	9-b adder tree
Multiplication result Accumulation	(N-1) 16-b adders	(N-1) 16-b adders	SA
Equivalent NAND gates	2048N-240	496N-240	143N+240
Gates Savings	13.43$\times$	3.47$\times$	1

3.2 Functional Units

The proposed architecture, called the Shift-Accumulate (or SAC) architecture, consist of Register Units, Computation Unit and Control unit. A systolic array of functional units (called the Register Units (RU)) based on Eqn 3.3 is developed and is shown in Fig. 3.1. The array consist of thirty-six RUs arranged as a 6×6 matrix. The partial products generated in RUs are forwarded to the Computation Unit (CU). The output is obtained from the CU after partial products accumulation and shift adjustments. The architecture has configurable datapath determined by the control word of target functions discussed later in 3.2.4 subsection. In addition to controlling the function execution, the control unit generates necessary synchronising signals for RUs and CU.

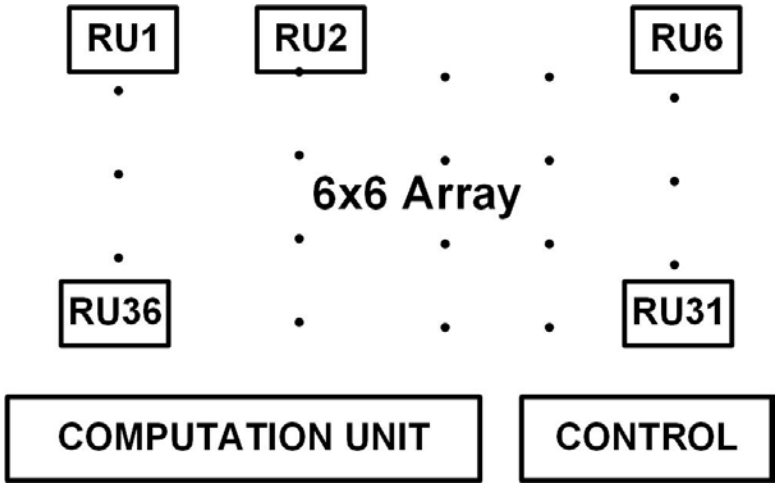


Figure 3.1: The Shift-Accumulate Architecture.

3.2.1 Register Unit

The Register Unit (RU) shown in Fig. 3.2 consists of registers, 8×1 AND matrix, 8×1 XOR matrix and supporting logic for partial product generation. The 9-b (sign + 8-bit magnitude) inputs $x[n]$ (input data) and b_i (coefficients) are loaded into data $\{\pm, D_7–D_0\}$ and coefficient $\{\pm, C_7–C_0\}$ registers, respectively. In general, these registers hold the multiplicand and multiplier for the multiplication operation. The coefficient is further loaded to shifter block where the multiplier is rotated one place left at every clock cycle. The MSB of the shifter block is fed to the

AND matrix and D_7–D_0 is the second input. The unsigned partial product thus generated is further converted to signed partial product (1's complement form) by XORing it with the sign bit. The 9-b 2:1 multiplexer makes the selection between registered and unregistered incoming data based on the signal from the control word. The latter is of importance in case of functions with feedback where the delay incurred during registering the input upsets the computation synchronisation. The data register forwards its data to the next RU after eight clock cycles. This is beneficial for windowing functions where the previous data are also required for computations. The gate count break-up for RU is presented in Table 3.2 and 6×6 array has 8712 total gates. Here, AND, OR, XOR, NOT are considered basic gates and each flip-flop is considered as 7 equivalent gates.

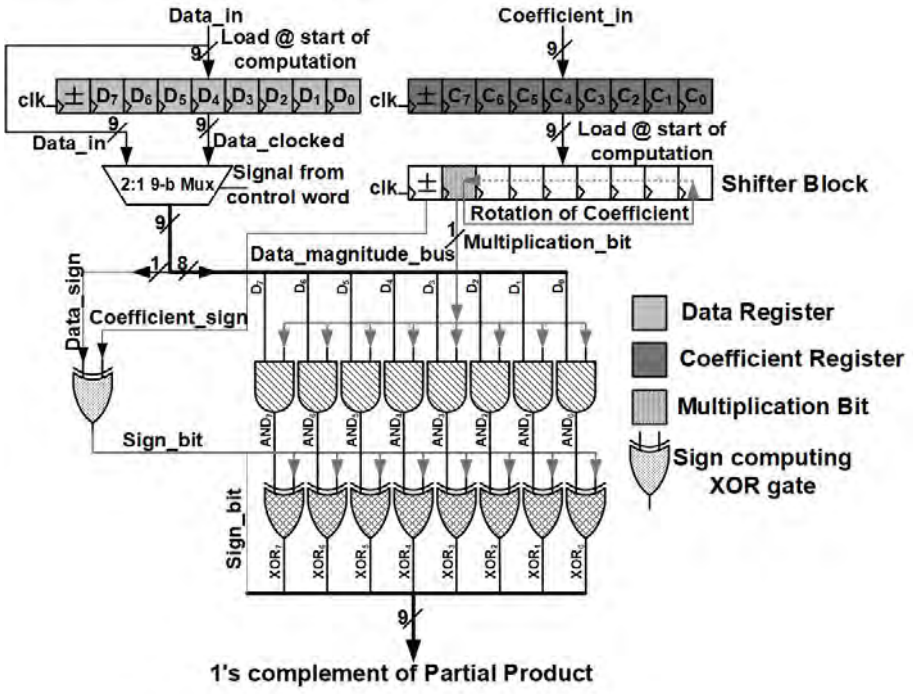


Figure 3.2: The Register Unit.

3.2.2 Computation Unit

The Computation Unit (CU) is divided into four identical sub-blocks (called subsections) and each sub-block acts on nine partial products as shown in Fig. 3.3. The outputs of subsections are accumulated by 21-b and 22-b adders. Each CU subsection (see Fig. 3.3) contains adder tree, shift-accumulator and output reg-

Table 3.2: Number of gates in RU

Components	Type of Gate	Gate Count
AND Gate matrix	AND	8
XOR Gate matrix + sign computing XOR	XOR	9
27 Reg (1 Reg = 7 gates)		189
2:1 9-b mux	AND	18
	OR	9
	NOT	9
Total Gates in RU = 242		
RU array (36 * RU) = 8712		

ister. The adder tree has ripple carry adders (RCA) that adds nine 9-b partial products. The partial product sum is added with the previous partial product accumulated sum using the 21-b adder and shift-accumulate register. The 23-b output register is loaded with the final result once the serially computed multiplication concludes. Earlier research in adders [102, 103] indicates compact adder circuit architectures profits in area, delay and power with scaling technology. Additionally, authors in [87] elucidates that RCA exhibits minimum energy in near threshold voltages. For the above metric and ease of implementation, RCA is adopted in this thesis.

The tree adder adds the signed partial products generated in 1's complement form as opposed to 2's complement representation because the later requires an additional adder. The MSB of full-adder is duplicated while realizing n-bit adder in the tree adder as shown in Fig. 3.4. This is done to address the potential problem arising out of negative number addition that generates a carry because of bit overflow. An an illustration, when -255 and -1 are added, the expected result is -256 that generates a carry-out because of overflow in 8-b magnitude representation. In conventional 1's complement addition, this carry-out is added into the magnitude which gives erroneous result +255. On the other hand, with MSB (or sign bit) duplication, the carry (n^{th} bit) and sign bit ($(n+1)^{th}$ bit) are clearly demarcated resulting in correct computation of results. The component wise gate count of the computation unit is presented in Table 3.3.

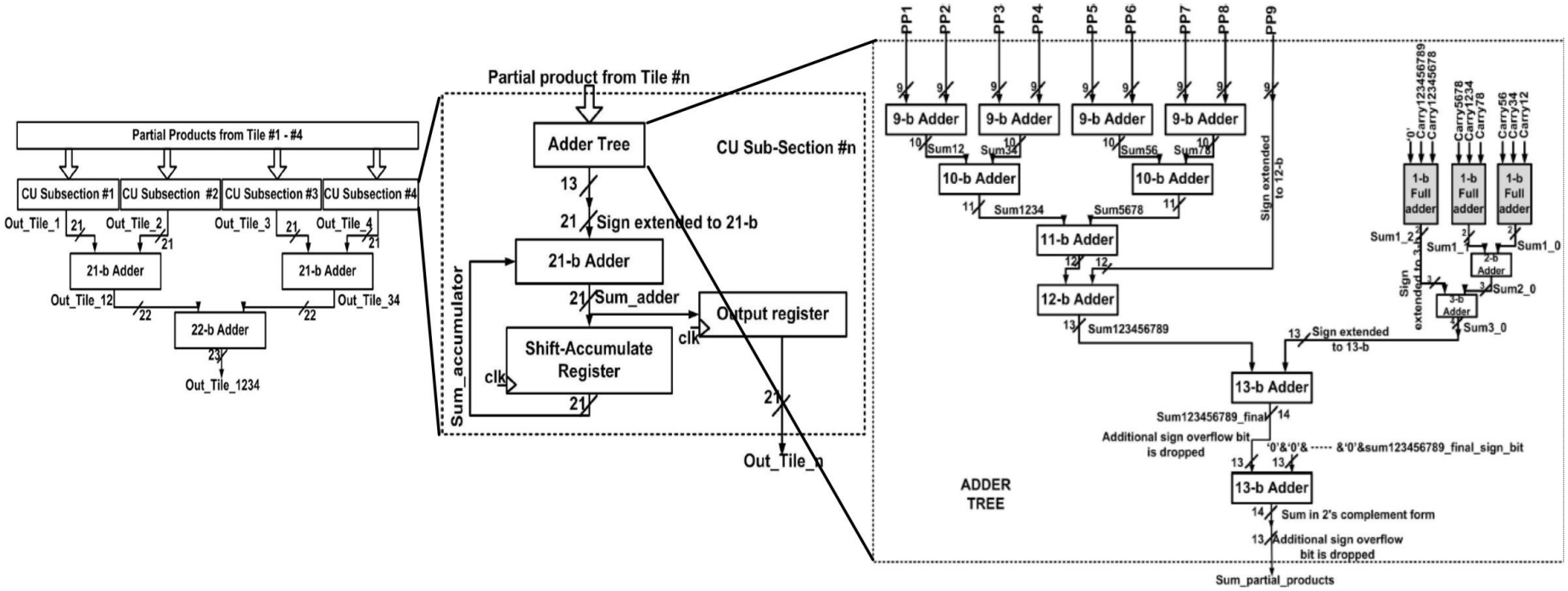


Figure 3.3: The Computation Unit hierarchy.

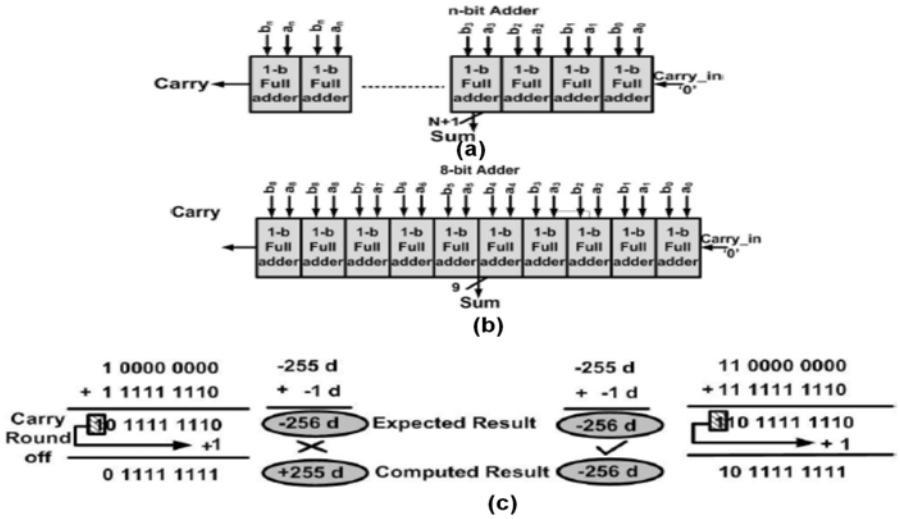


Figure 3.4: MSB duplication in RCA of tree adder with example.

Table 3.3: Number of gates in CU

Adder Tree			
Component Name	**# of Components**	**# of 1-b full adders**	**# of Gates**
9-b Adder	4	$10 \times 4 = 40$	200
10-b Adder	2	$11 \times 2 = 22$	110
11-b Adder	1	$12 \times 1 = 12$	60
12-b Adder	1	$13 \times 1 = 13$	65
13-b Adder	2	$14 \times 2 = 28$	140
1-b Adder	3	$1 \times 3 = 3$	15
2-b Adder	1	$2 \times 1 = 2$	10
3-b Adder	1	$3 \times 1 = 3$	15
Total Gates = 615			
CU Sub-Section			
Component Name		**# of 1-b full adders**	**# of Gates**
Adder Tree		–	615
21-b Adder		$21 \times 1 = 21$	105
21-b Shift Accumulator Reg		–	$21 \times 7 = 147$
21-b Output Reg		–	$21 \times 7 = 147$
Total Gates = 1014			
CU			
Component Name	**# of Components**	**# of 1-b full adders**	**# of Gates**
CU Sub-Section	4	–	$4 \times 1014 = 4056$
21-b Adder	2	$21 \times 2 = 42$	210
22-b Adder	1	$22 \times 1 = 22$	110
Total Gates = 4376			

1-b full adder = 5 gates (2 AND, 2 XOR and 1 OR), 1 Reg = 7 gates

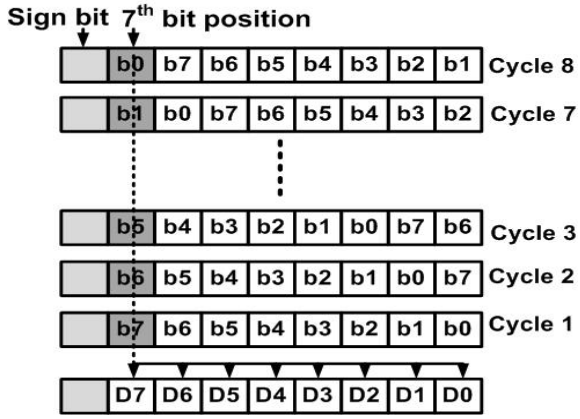


Figure 3.5: MSB rotation in coefficient register.

3.2.3 Architecture Operation

The coefficient register is loaded with multiplier in one clock followed by multiplicand loading in the data register and multiplier forwarded to shifter block in the subsequent clock (Clock #1). The multiplicand is ANDed with 7^{th} bit or MSB of multiplier generating the unsigned partial product. The partial product is a copy of multiplicand when the MSB is '1' and is 0 otherwise. The sign bits of multiplier and multiplicand are XORed to generate the sign of multiplication result that is positive if both inputs are of same polarity and is negative otherwise. The resultant sign bit is XORed with the unsigned partial product that remains unaltered when the resultant sign is '0'. In case the resultant sign bit is '1', the unsigned partial product is changed to its 1's complement representation because XOR operation generates the complement of its input when one of the inputs is held at '1'. The signed partial products are added in the adder tree in the CU. This partial product sum is added with the 0 *i.e.* reset value of the shift-accumulator in 21-b adder and is stored in the shift-accumulator.

In the next clock (Clock #2), shifter block has rotated the multiplier left by one place and 6^{th} bit is in MSB now as illustrated in Fig. 3.5. The signed partial products for 6^{th} bit are generated and are added in adder tree. The shift-accumulator block has 7^{th} bit partial product sum (PP_7) that is shifts left by one place *i.e.* multiplied by 2^1. The 6^{th} partial product sum (PP_6) is added with the shifted PP_7 in the 21-b adder. The shift-accumulator register is now loaded with $PP_7 \cdot 2^1 + PP_6$. In

the following clock (Clock #3), due to another left shift PP_7 is multiplied by 2^2 and PP_6 is multiplied by 2^1. This is added with 5^{th} bit partial product sum results in $PP_7 \cdot 2^2 + PP_6 \cdot 2^1 + PP_5$. The shift-accumulator register holds the final output *i.e.* $PP_7 \cdot 2^7 + PP_6 \cdot 2^6 + PP_5 \cdot 2^5 + PP_4 \cdot 2^4 + PP_3 \cdot 2^3 + PP_2 \cdot 2^2 + PP_1 \cdot 2^1 + PP_0$ after 8 clock cycles which represent the architecture latency. This result is loaded into the output register and is available for further processing.

3.2.4 Configurable Datapath

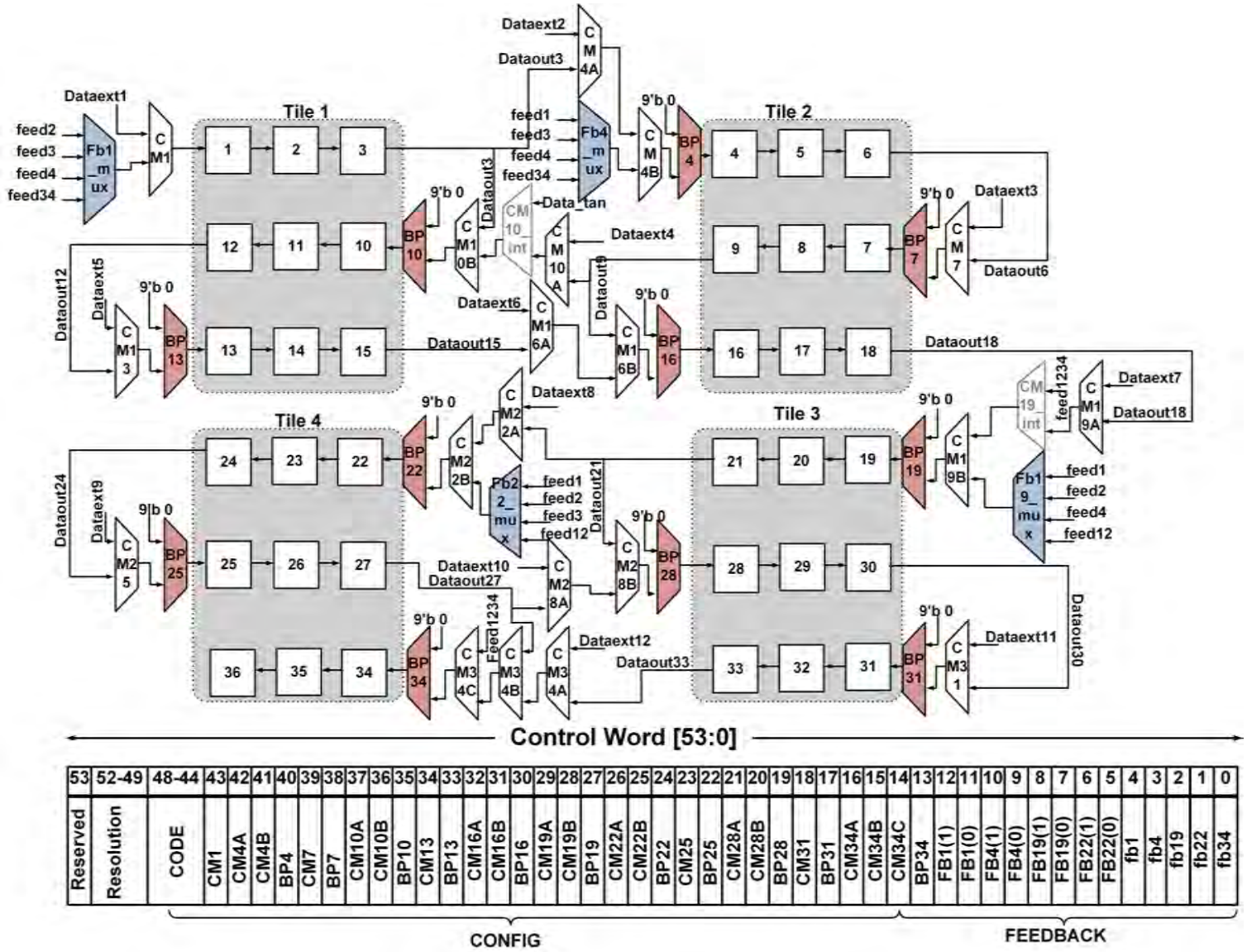
The SAC architecture implements the multiply-accumulate by means of simple AND-shift-accumulate operation. However, keeping the entire architecture in active state while analysing simple operations like multiply, add, shift *etc.* results in low utilization factor as well as adversely affects the power of the system. This condition is addressed by making the datapath of the architecture configurable, making it possible to disconnect certain sections of the architecture. Additionally, the configurable datapath also ensures simultaneous realization of multiple functions, increasing the throughput as well as hardware utilization.

The architecture is configured using a 54-b control word. The control word has *Code*, *Config*, *Feedback* and *Resolution* fields. The 5-bit *Code* field is used to identify the function realized on the architecture broadly between two categories - fixed or external coefficient functions. The 31-b *Config* field denotes the select line of the multiplexers placed in the datapath. Two sets of multiplexers, called the Configuration and Bypass, are used in the datapath and allow different connection topologies between RUs as well as disconnect RUs from the active datapath. For instance, group of nine RUs is called a RU Tile, shown in gray boxes in Fig. 3.6. The configuration multiplexer provides input to the RU data register (multiplier) and makes the selection between the incoming data from previous RU or external data (marked Dataextn in Fig. 3.6). Additional configuration multiplexers are placed between the tiles to derive tile configuration and provide feedback. This is discussed further in section 3.3. The bypass multiplexer either forwards the configuration multiplexer output or feeds 0 to the RU disconnecting it from the datapath as 0 data does not contribute to the computation irrespective of the mul-

tiplicand. The bypass multiplexers are situated after every three RUs, thus RUs can be bypassed in a set of three. This set of three RUs is termed as RU triplet in the rest of the thesis.

Least significant thirteen bits in the control word, labelled FBx and fbx, represent the select lines of feedback multiplexers (marked in blue in Fig. 3.6) and constitute the *feedback* field of control word. The individual or combined output of tiles can be fed to RU using the feedback multiplexer. Bits 0-4 of the control word are the select lines of the multiplexer inside RU that forwards registered or unregistered multiplicand. These bits are dedicated for RUs that can receive feedback data *i.e.* RU #1,4,19,22 and 34.

The leading four bits of the control word (*Resolution* field) provide the resolution of output. While doing calculations with numbers involving decimal points the resolution of output varies with function mapped and different data sets. However, based on the resolution of input data and coefficients the resolution of the output can be determined. The 23-b output is truncated to 9-b in accordance with the resolution bits that helps in locating the bits having the relevant output to be fed back to the architecture as input in case of feedback or multiple function operation described later in subsection 3.3.3. The remaining bits of the control word are reserved for future modifications.



53	52-49	48-44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	Resolution	CODE	CM1	CM4A	CM4B	BP4	CM7	BP7	CM10A	CM10B	BP10	CM13	BP13	CM16A	CM16B	BP16	CM19A	CM19B	BP19	CM22A	CM22B	BP22	CM25	BP25	CM28A	CM28B	BP28	CM31	BP31	CM34A	CM34B	CM34C	BP34	FB1(1)	FB1(0)	FB4(1)	FB4(0)	FB19(1)	FB19(0)	FB22(1)	FB22(0)	fb1	fb4	fb19	fb22	fb34

Figure 3.6: The Configuration and Bypass multiplexers in SAC Architecture.

Table 3.4: Gate count of multiplexers in Configurable Datapath

Datapath Configuration (1620 Gates)				
	Dimension	Units	Equi 9-b 2:1 Muxes	Total Gates
Configuration Mux	9-b 2:1	31	31	$31 \times 1 \times 36 = 1116$
Feedback Mux	9-b 4:1	4	3	$4 \times 3 \times 36 = 432$
CORDIC Internal Mux	9-b 2:1	2	1	$2 \times 1 \times 36 = 72$
Coefficient Configuration (5368 Gates)				
	Dimension	Units	Equi n-b 2:1 Muxes	Total Gates
Coefficient Mux *n*=9	9-b 5:1	3	4	$3 \times 4 \times 36 = 756$
	9-b 4:1	17	3	$17 \times 3 \times 36 = 432$
	9-b 3:1	1	2	$1 \times 2 \times 36 = 72$
DCT Mux *n*=8,1	9-b 7:1	19	6(8-b) + 1(1-b)	$19 \times (6 \times 32 + 1 \times 4) = 3724$
CORDIC Mux *n*=8	8-b 7:1	2	6	$2 \times 6 \times 32 = 384$
Total Gates in Configuration = 6988				

Configuration Muxes have 20 CMx and 11 BPx.

3.2.5 Configurable Datapath Gate Count

Datapath multiplexers employed are 9-b wide and are either 2:1 or 4:1 contributing 1620 gates as shown in Table 3.4. This overhead amount to ≈12.5% of the combined gate count of RU and CU. The SAC architecture inherently supports linear functions, however its functional domain is extended to include certain non-linear functions by adding coefficient multiplexers. Chapter 4 contains a function-wise discussion on coefficient multiplexers highlighting its utility in context with the target functions of the architecture. These multiplexers are connected to the coefficient register of RU and provide necessary coefficients while computing functions that follow an algorithm. The coefficient multiplexers contribute 5368 gates to the architecture amounting to 41% of the RU-CU gates. In addition to this, internal storage of coefficients require 156 flip-flops. This increases the configuration overhead to ≈50% of the gate count dedicated for computation indicating a trade-off between the configurability overhead and employing dedicated hardware for every function. However, if the developed implementation methodology of the target functions comprehends them as a set of common operations by means of minor modifications, the use of configurability hardware may turn out gate effi-

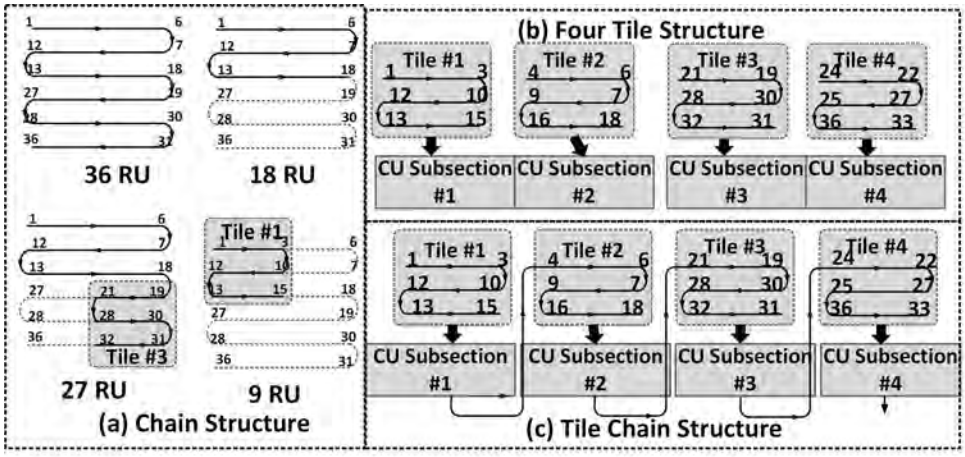


Figure 3.7: Different Architecture Topologies.

cient as opposed to dedicated hardware realization which would require separate input/output memory modules, interfaces, interconnects and interrupt logic for every function.

3.3 Architecture Topologies

The configurable datapath of the SAC architecture exhibits various topologies (shown in Fig. 3.7) that are discussed now.

3.3.1 Systolic Array

The RUs can be arranged in a continuous chain of nine, eighteen, twenty-seven or thirty-six RUs (see Fig. 3.7 (a)). The most straight forward structure of thirty-six RUs forming a chain is formed with RUs connected one after the other. The bypass multiplexer select are set and configuration multiplexers passes the previous RU output. The eighteen RU chain can also be realized in the same manner with BP19 forced to '0'. However, a chain of nine RU is obtained in Tile #1 and it consists of RU # 1,2,3,10,11,12,13,14,15 by forcing '0' and '1' on CM10B and CM13. Additionally, select line of all bypass multiplexers is '0' except for BP10 and BP13. The twenty seven RU chain is formed by combining the eighteen RU chain and Tile #3 while bypassing Tile #4.

3.3.2 Four Tile

The four tile structure (see Fig. 3.7 (b)) is obtained on dividing the RUs into four groups of nine RU blocks. This configuration enables parallel execution of functions as each tile forwards its partial products to their respective CU subsection that computes the results irrespective of the ongoing computation in other tile or CU subsection. The configuration multiplexers CMxB select line are forced to '0' thereby passing forward the data from the RU above it. For instance, Dataout #3, 9, 21 and 27 are fed to RU #10, 16, 28 and 34, respectively. The bypass multiplexers are set when all the four tiles are in use. However, when less than four tiles are required for function realization the bypass multiplexer of RU leading the tile is used to disconnect the tile.

3.3.3 Systolic Tile

The tiles when connected one after the other forms the systolic tile structure (see Fig. 3.7 (c)). This structure is beneficial when a function chain is mapped on the architecture. As an example, function 1 can be realized on Tile #1, function 2 on Tile #2 and so on. On doing this, the architecture is handling multiple functions that have interdependencies *i.e.* function 2 is acting on output of function 1. Here, not only the computation of different functions is independent of each other but also the functions outputs are calculated almost simultaneously thereby saving on the architecture reconfiguration latency. Furthermore, no additional memory is required to store the function 1 outputs as they get stored within the architecture in Tile #2.

The feedback multiplexer placed in front of the tile feeds individual or combined outputs of other tiles by means of dedicated select lines (FBx) in the control word. The combined output of tiles is represented as *feed12*, *feed23 etc.* The CM and BP select lines are set according to the four tile structure.

Several alternate realizations of the aforementioned topologies are possible. The eighteen-RU chain can be formed by combining RU #19-36 or nine-RU chain realized in Tile #3 are few of the many possibilities. Additionally, the architecture

also supports mixed topologies. For instance, one of the possible combination of systolic tile and array structures can result in datapath from Tile1 –> Tile2 –> RU #19-36. These topologies can be obtained by setting the necessary CMx and BPx bits in the control word.

3.4 Other Domain-Specific Reconfigurable Architectures

Flexible reconfigurable architectures intended for DSP and multi-media applications are presented in [104] and [105], respectively. The former leverage the similarity in popular DSP algorithms and their architectures to develop the architecture consisting of adder and multiplier in its processing element (PE) with programmable datapath leading to various combinations of these two operations. Furthermore, the fine-grained interconnect architecture in [104], inspired by the traditional FPGA interconnect design, is configurable and consists of local interconnect, switch box and connection box enabling communication among various PEs. In case, little or no encoding is used on the interconnect bit stream, the configuration bits grow considerably with the adopted hierarchical interconnection scheme. However, owing to the coarse grained PE design in [104], $\approx 7\times$ area savings is achieved as compared to a fine grained FPGA implementation.

The MORA architecture in [105] performs high performance, computation intensive, real time data processing using a 2-D array of identical reconfigurable cells having a tightly coupled memory. The processing element contains two 8×4 multipliers and adders along with supporting logic for high data rates processing. The interconnection network is flexible but does not support run-time configuration. Further, [105] reports that MORA outperforms DSP processors and fine-grained FPGA implementations while computing 8×8 2-D DCT operation in performance as well as area. However, the large memory (256×8-bit SRAM) coupled with the processing element might affect the power consumption in limited energy systems, typical of wearables, mobile devices *etc.*

An Ultra Low Power Samsung Reconfigurable Processor (ULP-SRP) [99] ded-

icated for biomedical signal processing exhibits a Coarse Grained Reconfigurable Architecture (CGRA) consisting of nine functional units (FUs). It supports high performance data processing (order of 100 M*Hz*) with a shortened execution time, thereby reducing the total energy consumption. It has three performance modes chosen dynamically according to the application during run-time. The performance modes require certain sections of the processor to be active/inactive during execution. This is achieved using a Power Management Unit (PMU) having fourteen power domains. The change over between performance modes requires a series of steps to be performed which includes writing special registers, executing special instructions, selective powering of FUs *etc.* This results in frequent stalling of the processor adversely affecting its execution time. Furthermore, the data interdependencies in certain applications may limit computations in high performance mode missing out on the reduced execution time advantage. Additionally, determining the switching sequence of performance modes requires a sophisticated compiler making critical decisions based on the incoming application rendering increased complexity and reduced user control over the system.

Another biomedical processing platform, a multi-core reconfigurable architecture in [101] uses multiple software cores along with a shared CGRA platform deployed for application acceleration. A scheduler is used to partition the application between software cores and CGRA providing better energy efficiency by means of efficient mapping and consequently increased idle time. The reconfigurable cell (RC) embeds an ALU with tightly coupled memory to reduce data access and storage time. The synchronizer also takes care of the flow of data among different cores and CGRA as well as provides the conventional interconnect network.

The SYSCORE platform in [97] has a energy efficient CGRA architecture targeting biosignal processing. It has coarse grained interconnect network and compact processing element (called reconfigurable function unit) that reduces the reconfiguration overhead and logic switching leading to reduced energy consumption. The systolic structure of the architecture favours reduced RAM access which further improves the energy efficiency.

3.5 Conclusion

The SAC architecture is an inherently light weight design owing to various adopted measures. The processing element of SAC architecture, Register Unit (RU), is coarse grained and ensures logic switching at every clock through serial generation of partial products. The elimination of multiplier from the design yields gate savings. The custom interconnect network, designed to support various biomedical signal processing functions, is a coarse grained network yet ensures varied kernels realization along with optimized resource utilization using selective activation and deactivation of RUs. Furthermore, the coarse grained interconnections provide a scope of efficient function mapping through increased user control over the architecture which is seldom observed in compiler based systems. The systolic structure and tightly coupled data memory, both ensures considerably reduced memory accesses. The fetch and decode stage of the traditional processor is coupled into a single configuration stage. Additionally, the reduced gate count of the architecture ensures reduced overall leakage and consequently a potential increase in energy efficiency when voltage scaling scheme is adopted. Furthermore, the increased switching activity due to the serial nature of architecture, results in reduced leakage power at the cost of increased dynamic power [87].

A 6×6 shift-accumulate configurable architecture presented in this chapter, has multiply-accumulate as its underlying operation that furnish reduced gate count as compared to parallel or mixed form MAC based architectures. The developed architecture is designed to operate on signed partial products and has a latency of 8 clock cycles. The Register Unit uses AND gates to generate partial products, accumulated later in Computation Unit. The datapath of architecture contains multiplexers that enables connecting RUs in various topologies and efficient hardware utilization by means of disconnecting unused RUs from datapath. The architecture has simple configuration scheme and control word predominantly consists of datapath multiplexers select lines. The gate level structure and gate count of RUs, CU and configuration components are presented. The gate configuration overhead constitutes ≈12.5% of the gates dedicated for computa-

tion which further increases to ≈50% when coefficient multiplexers are included. However, the configuration overhead may be acceptable over multiple dedicated hardware solution that includes separate memory modules, input/output interfaces and associated interrupt logic. Additionally, the architecture can exhibit various topologies including systolic array, four tile, systolic tile *etc.* These topologies are obtained owing to disjoint partial products accumulation with each tile acting as an individual processing unit. This indicates that modularity and scalability features are readily supported by the developed architecture.

CHAPTER 4

Mapping of Processing Functions

Predominantly, ambulatory monitoring includes recording and analysis of physiological signals like ElectroEncephaloGraph (EEG), ElectroCardioGraph (ECG/ EKG), ElectroMyoGraph (EMG) and PhotoPlethysmoGraph (PPG) signals necessary for estimating normal neural, cardiac, muscle and blood oxygenation activity, respectively. In some cases, artifacts derived from these signals serve as reliable indicators of organ damage in renal and retinal systems. The acquired physiological signals undergo multiple processing steps including denoising, feature detection, extraction, classification *etc.* to identify abnormal events. The configurability of the developed SAC architecture can be leveraged to realize multiple biomedical signal processing steps on it. This was possible, because majority of the digital signal processing functions include multiplication and addition as their fundamental operations which is also the underlying principle of the SAC architecture. Various biomedical signal processing algorithms are proposed in literature that detect and extract useful features from physiological signals vital in ambulatory monitoring. These algorithms were further analyzed for the number of signal processing functions present in them in previous chapter (section 2.3.3 and functions frequently used in biomedical signal processing were identified. Mapping methodologies for the predominant signal processing functions that are optimized for cycle, gate count and computations are presented in this chapter. The target functions are classified broadly into two categories: *Variable* and *Fixed* coefficient functions. The coefficients of the former depends on characteristic parameters of these functions. For example, bandwidth, gain and number of taps determine the coefficients of FIR filter. On the other hand, the fixed coefficient functions have constant coeffi-

cients as the name suggests. These functions undergo a series of steps following a fixed sequence/algorithm resulting in constant coefficients. Therefore, it is redundant for the user to explicitly provide constant coefficients to the architecture and it is eliminated by means of designing small state machines for these functions.

4.1 Variable Coefficient Functions

This category includes functions that accept user-defined coefficients. The specifications of functions plays an important role in determining the coefficients. These coefficients can be derived using analytical equations or MATLAB tool sets. In the following sections, the underlying mathematical equations and usability in biomedical applications of the target functions are discussed followed by mapping and interpretation on SAC architecture.

4.1.1 Multiplication

Multiplication is one of the most fundamental operations of digital signal processing and accepts two inputs (multiplicand and multiplier) to generate one output. Classic method of computing the product A×B includes generation of partial products followed by their accumulation in accordance with the weight associated with them. In binary number system, multiplication of two inputs, A and B, having N_1 and N_2 bits, respectively results in $(N_1 + N_2)$ bits output, P. The binary representation for inputs A and B is shown in eqn 4.1

$$A = \sum_{i=0}^{N_1-1} A_i \cdot 2^i; \quad B = \sum_{j=0}^{N_2-1} B_j \cdot 2^j \quad with \quad A_i, B_j \quad \epsilon \quad \{0,1\} \tag{4.1}$$

The multiplication operation is defined as follows:

$$\begin{aligned} P &= A \times B = \sum_{k=0}^{M+N-1} P_k \cdot 2^k \\ &= \left(\sum_{i=0}^{N_1-1} A_i \cdot 2^i\right) \cdot \left(\sum_{j=0}^{N_2-1} B_j \cdot 2^j\right) = \sum_{i=0}^{N_1-1} \left(\sum_{j=0}^{N_2-1} A_i \cdot B_j \cdot 2^{i+j}\right) \end{aligned} \tag{4.2}$$

The array multiplier is the straightforward hardware implementation of classic multiplication method and has one-to-one topological correspondence with the manual multiplication. However, it is not optimized for area, critical path delay and efficient layout. Literature reports various multiplier architectures that are optimized for one or more of these short-comings. Hierarchical, pipelined, vedic, modified booth are some of the multiplier architectures [106–109] with improved area, energy and (or) performance along with support for floating point computations in [109].

Biomedical signal processing algorithms make use of multiplication in Finite/ Infinite Impulse Response (FIR/IIR) filtering, matrix multiplications, wavelet transform *etc.* The squaring operation can be considered as a special case of multiplication where a number is multiplied with itself *i.e.* same number acts as multiplicand and multiplier ($A = B$). This operation is often employed in signal analysis and error analysis. For example, mean square error estimation uses square operation to limit the signal deviations to positive direction. Additionally, multiplication by (1/n) implements division functions used in computing averages, relative errors, normalized data *etc.*

A paradigm shift is observed in multiplier designs and multiplier-less architectures for computing products are gaining popularity in recent times for biomedical [110] and DSP applications [88], in general. This is achieved by serial computation of multiplication as opposed to a parallel mode seen in case of array multiplier earlier. The serial mode of computation increases the hardware utilization at the expense of increased latency. The simplest realization of serial multiplication is repetitive addition where the multiplicand is added with itself multiplier number of times. Such an implementation requires an adder along with a subtractor and comparator logic wherein the later decrements the multiplier till it becomes zero and addition operation is terminated at this point. An alternate method of serial multiplication discussed in Chapter 3, generates the partial products serially using AND gates matrix followed by shift-accumulate operation. We now discuss the mapping of multiplication operation on the SAC architecture.

Mapping Methodology of Multiplication

The multiplication of multiplicand and multiplier terms can be realized on SAC architecture by loading them into the data and coefficient registers of RU #1, respectively. The SAC architecture supports 8-b$\times$8-b signed multiplication in fixed point arithmetic. The data and coefficient registers are loaded with same number to map square operation. To emulate squaring, the data bus is connected to coefficient bus through a coefficient multiplexer which ensures data forwarded to the coefficient. This is advantageous for systems with separate coefficient and data memory as it is not necessary to provide squaring input to both the memories explicitly. The division by N operation, where $N \epsilon \mathbb{R}$ limited to 8-b resolution, is realized by loading the coefficient with $1/N$. The subsequent multiplication causes the multiplier to be divided by N.

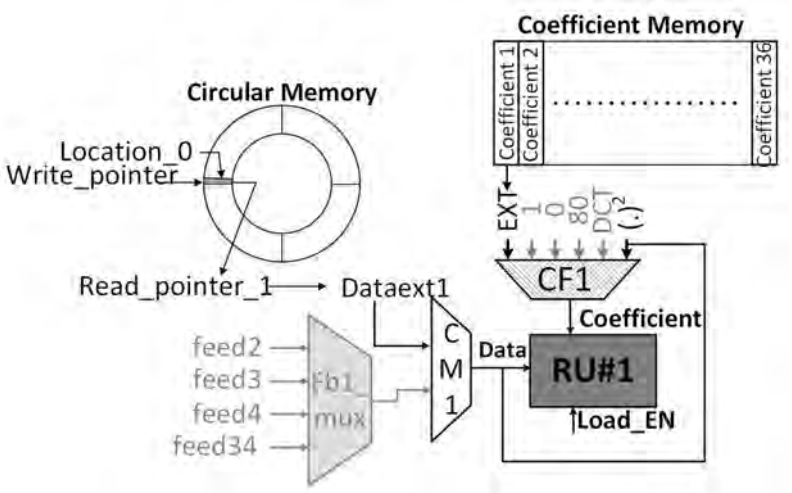


Figure 4.1: Mapping scheme for multiplication along with circular and coefficient memories.

The multiplier and multiplicand terms are provided to the architecture through circular and coefficient memories, respectively. Multiplier is written into the location_0 (address 0) of the circular memory and further writing is ceased. This data is read by read_pointer_1 which supplies the multiplier to *Dataext1* input of RU #1. The multiplicand is loaded into RU #1 coefficient register from coefficient memory by activating the respective coefficient register through load enable signal. The *EXT* input of coefficient multiplexer is attached to coefficient register input and gets loaded once register is enabled. These enabling signals are gen-

erated using the coefficient state machine. Additionally, a coefficient multiplexer connects the data to the coefficient register input as required by the square operation. Figure 4.1 illustrates the data and coefficient registers connections with associated memories and multiplexers. The gray colored text and arrows indicate inputs and connections unused at the moment. However, as we progress with other functions mapping, the purpose of these inputs and connections will be addressed.

4.1.2 Multiply-Accumulate (MAC)

In most general sense, the multiply-accumulate (MAC) operation decomposes into a combination of multiplication and addition. A typical MAC unit performs $A + B \times C$, where numbers B and C are multiplied together followed by addition with the accumulator data A. The accumulator data is updated to represent the MAC result after every such operation leading to serial accumulation of product terms. The direct hardware implementation of MAC requires a multiplier, adder and accumulator register as shown in Fig. 4.2.

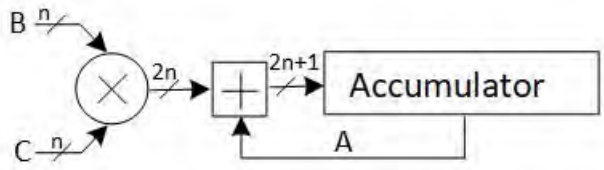


Figure 4.2: The direct hardware implementation of multiply-accumulate operation.

Alternatively, N-term multiply-accumulate can be obtained by generating N product terms using N multipliers and subsequent accumulation by $(N-1)$ adders. This constitutes the parallel or dot product approach to compute MAC which leverage cycle advantage when a large number of products are to be accumulated and is particularly beneficial for real-time processing [88]. This form of MAC can be readily realized on the SAC architecture because of its direct topological equivalence. Further, such a unit is capable of performing add (or subtract) only by setting $B = \pm 1$ converting MAC into $A \pm C$.

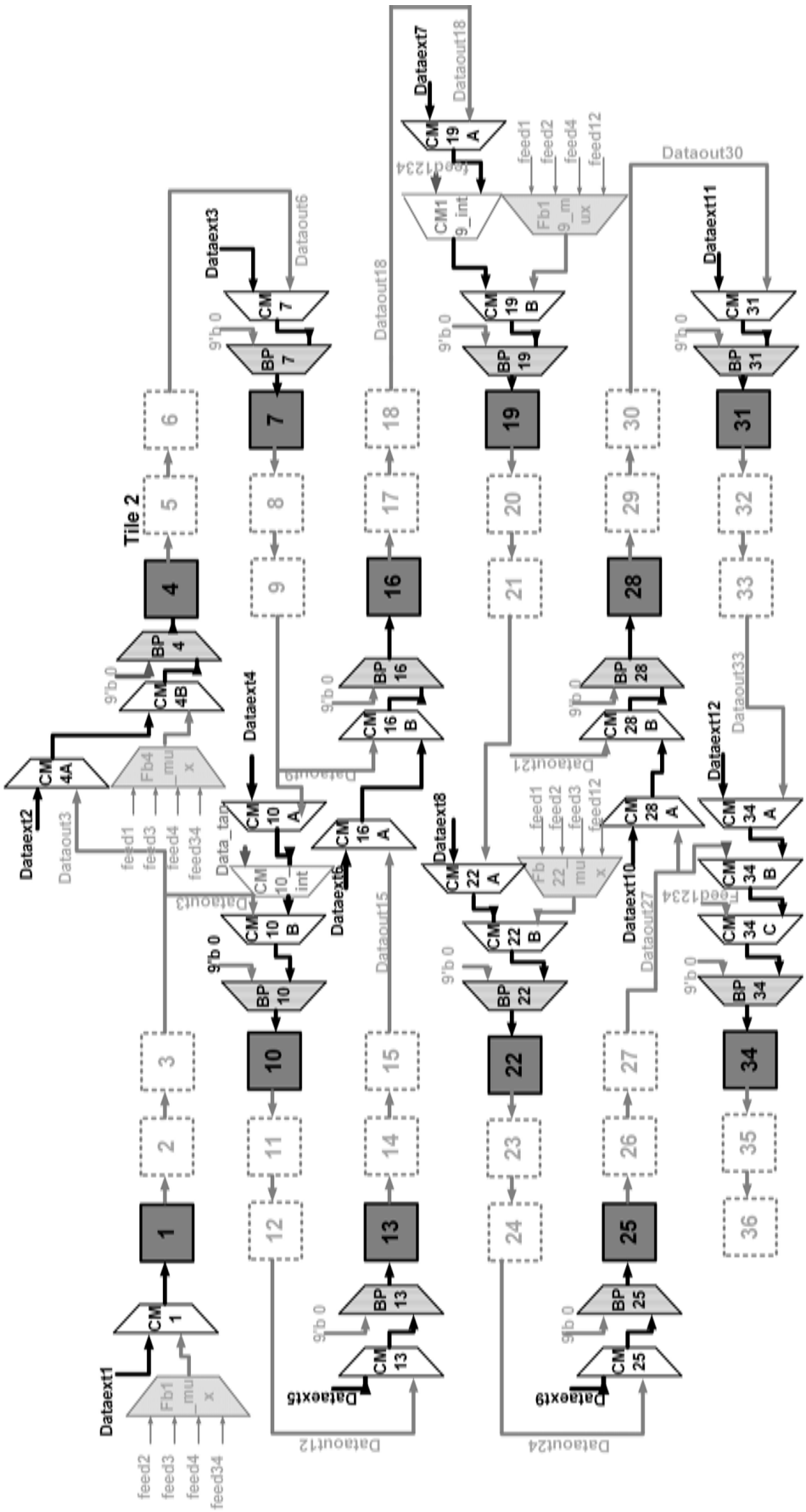


Figure 4.3: Active RUs and datapath for mapping MAC.

Mapping Methodology of MAC

The SAC architecture supports MAC for $\leq$12 terms implementing the general equation $\sum_{i=0}^{11} a_i \cdot b_i$. When $i < 12$, the excess (12-i) a_is are set to zero which translates to clearing coefficient register of RUs used to implement MAC. RU #1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 30 and 33 accept external data input (*Dataext n*, $1 \leq n \leq 7$) (through corresponding configuration multiplexers) and thus become obvious sites for MAC computation. The active multiplexer inputs and RUs are shown in Fig. 4.3. The select line of configuration multiplexer CM1 is clear, connecting *Dataext1* to RU #1 data register. For the remaining RUs, this connection is made through the bypass multiplexer which is set forwarding the configuration multiplexer data to corresponding RU data register. An alternate way to realize MAC with < 12 terms is by clearing the bypass multiplexer select line which results in disconnecting a RU section by feeding 0 in RU's data register. The active RUs and multiplexer setting for addition operation is same as MAC mapping, with the exception that coefficients of active RUs are loaded with 1 using coefficient multiplexers.

For MAC (or addition) operation with $\leq$ 36 terms alternate mapping scheme can be employed where the data is propagated to the RUs next to active RUs indicated in Fig. 4.3. MAC operation upto 24 (or 36) terms can be realized when data propagates to one (or two) RU down the line. The latency increases by 8 clock cycles with every depth level increase. The active RUs represent depth level 0 and generate the MAC (or addition) result in 8 clock cycles *i.e.* the minimum latency of the architecture. The MAC realization with depth level 1 *i.e.* active and its adjacent RU involved in the operation takes 16 clock cycles to produce result. Similarly, results produced in depth level 2 realization takes 24 clock cycles.

The b_i terms of the MAC equation is loaded in data registers and a_i's in coefficient registers. For implementing depth level 0 MAC, n data where $n \leq 12$ is written into the circular memory that are later read in parallel by multiple read pointers. This data is provided to architecture through the *Dataext n* inputs. The read pointers are anchored at various memory addresses and Fig. 4.4 shows the location of 12 read pointers. The coefficient a_i's are taken from the coefficient

memory through the *EXT* input of coefficient multiplexer.

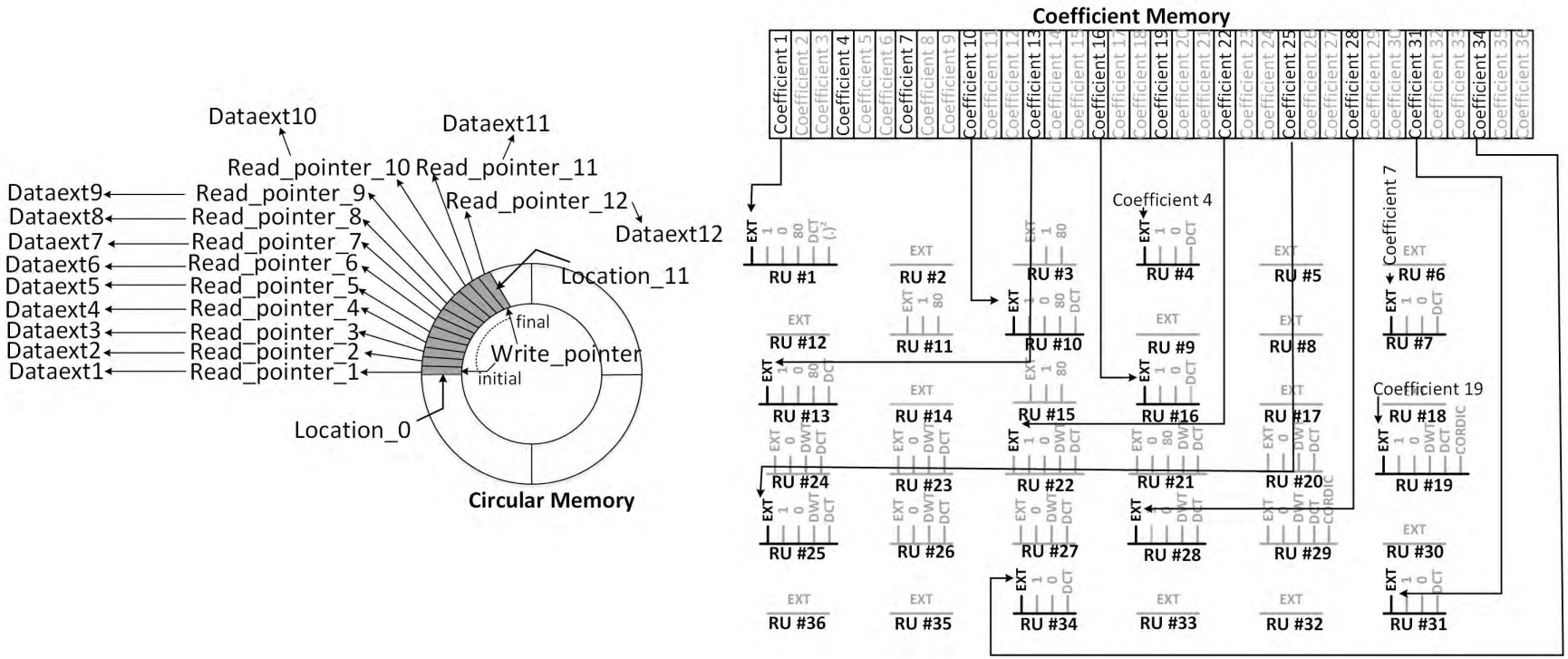


Figure 4.4: The circular and coefficient memory connection for MAC mapping.

4.1.3 FIR filtering

The term *filter* is used to describe a linear time-invariant system used to perform spectral shaping or frequency selective filtering. It modifies the input signal spectrum $X(w)$ according to its frequency response of the filter $H(w)$ to yield output signal with spectrum $y(w) = H(w)X(w)$. The nature of this filtering action is determined by the frequency response characteristics $H(w)$, which in turn depends on the choice of system parameters (*e.g.* passband (and stopband) ripple, gain, passband (and stopband) edge frequency). These system parameters determine the coefficients, a_k and b_k in the difference equation, required for frequency selective filter design that pass signals with frequency components in some bands while attenuate signals containing frequency components in other frequency bands.

Z-transform serves as a discrete version of laplace transform and is particularly important in modelling transfer function and characterizing the nature of system based on pole-zero analysis. We estimate system function, $H(z)$, of a linear time-invariant system described by a linear constant-coefficient difference equation represented as eqn 4.3 by computing the z-transform of both sides of the equation.

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k) \tag{4.3}$$

By applying the time-shifting property of z-transform, we obtain eqn 4.4 which takes the form of eqn 4.5 on rearrangement

$$Y(z) = -\sum_{k=1}^{N} a_k Y(z) z^{-k} + \sum_{k=0}^{M} b_k X(z) z^{-k} \tag{4.4}$$

$$Y(z)\left(1 + \sum_{k=1}^{N} a_k Y(z) z^{-k}\right) = X(z)\left(\sum_{k=0}^{M} b_k X z^{-k}\right)$$

$$\frac{Y(z)}{X(z)} = H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} \tag{4.5}$$

Eqn 4.5 represents the general form for the system function of a system described by linear constant-coefficient difference equation. This form contains both poles and zeroes, and hence the corresponding system is called a *pole-zero system*, with N poles and M zeroes. Due to the presence of poles, the impulse response of such a system is infinite in duration, and hence it is an Infine Impulse Response (IIR) system.

From the general form of system function, we obtain an important special form by setting $a_k = 0$ for $1 \leq k \leq N$. This reduces the system function to

$$H(z) = \sum_{k=0}^{M} b_k z^{-k} = \frac{1}{z^M} b_k X(z) z^{M-k} \tag{4.6}$$

In this case, $H(z)$ contains M zeroes, whose values are determined by the system parameters b_k, and an M^{th}-order pole at origin $z = 0$. Since the system contains only trivial poles (at $z = 0$) and non-trivial zeroes, it is called an *all-zero system*. Clearly, as such a system has finite-duration impulse response (FIR) it is called an FIR system or a Moving Average (MA) system.

FIR Architectures and Applications

Few methods for implementing FIR system are now presented. Note that equivalent implementation for IIR systems can be found in [111]. The direct-form structure realization follows immediately from the non-recursive difference equation given by eqn 4.3 with $a_k = 0$. The structure is illustrated in Fig. 4.5 where the unit delay system is represented by its system function z^{-1}. Delays of more than one sample can be denoted with a system function of z^{-M}, where M is the number of samples of delay. The actual implementation of M samples of delay would generally be done by cascading M unit delays. We observe that the structure requires $(M-1)$ memory locations for storing the $(M-1)$ previous inputs, and has a computation complexity of M multiplications and $(M-1)$ additions per output point. Since the output consists of the weighted linear combination of $(M-1)$ past values of the input and present value of input, the structure resembles a tapped delay line or a transversal system and thus is often called as tapped-delay-line or

transversal filter.

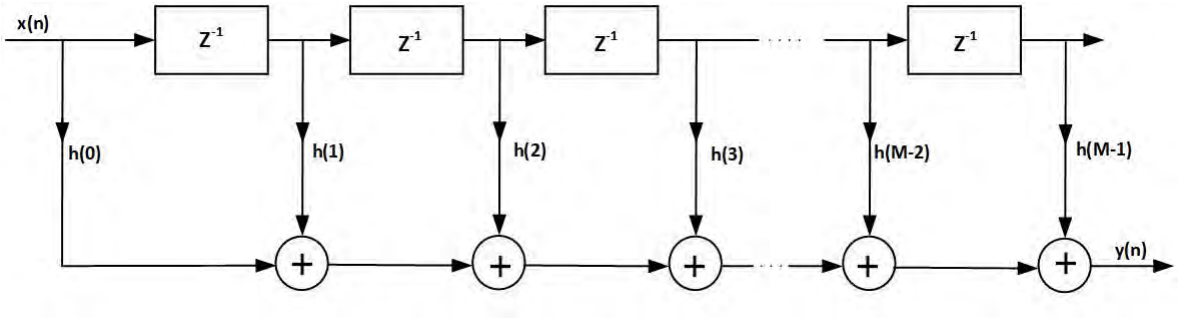


Figure 4.5: The direct-form realization of FIR filter.

Another structure used extensively in digital speech processing and adaptive filter implementations is lattice filters shown in Fig. 4.6. A direct equivalence between mth-order direct-form FIR and an m-stage lattice filter is discussed in [111]. The lattice form provides a compact representation of the class of m FIR filters as they are described by a set of recursive equations.

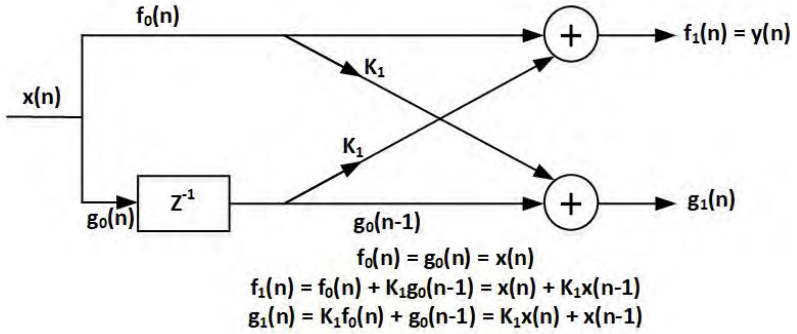


Figure 4.6: Lattice filter implementation of FIR filter.

Difference equations are often used to calculate numerical differentiation and integration that forms the heart of calculus and are exploited in many applications. Derivative of a function, $f(x)$ at $x = a$, is defined as

$$\left(\frac{df}{dx}\right)_{x=a} = lim\ \frac{f(a+\triangle x) - f(a)}{\triangle x}, as\ \triangle x \to 0. \tag{4.7}$$

This is called a forward-difference approximation. An equivalent backward-difference approximation can be obtained by approaching the limit from the opposite direction. A digital system processing discrete data calculates the approximation to the derivative by using a very small value for $\triangle x$ while keeping the

round-off and truncation errors balanced. Higher order differentiation techniques include weighted accumulation of past, present and future samples also known as the three-point formulae.

By inspection, it can be observed that the differentiation computation, in general, includes accumulation (or signed addition) of adjacent samples and scaling by factor $\triangle x$, thus leverage itself for equivalent MAC implementations. For the ease of implementation, the future sample in eqn 4.7, $f(a + \triangle x)$, can be treated as the present sample resulting in time-shifted computation of result.

$$y[n] = K(x[n+1] - x[n]) \textit{ is equivalent to} \tag{4.8}$$

$$y[n-1] = K(x[n] - x[n-1]) \tag{4.9}$$

Eqn 4.9 shows time-shifted output computation using present and past sample that is equivalent to eqn 4.8 which in turn is directly analogous to eqn 4.7. In VLSI systems, such shifted outputs are often encountered and are acceptable as they do not disrupt the signal integrity.

Digital signal processing applications use filtering to improve signal to noise ratio by removing the noise in the signal added during acquisition, data conversion *etc*. The effect of noise is pronounced in case of biomedical signals where the signal peak amplitude lies in hundreds of μV range. The peak signal amplitude for EEG and EMG are in 50-500 μV [112] whereas peak amplitude picked up by the ECG electrodes is slightly higher in comparison and is in the range on 1 mV [113]. For instance, an ECG signal is prone to artefacts arising from the interference from the power line (50 Hz noise), baseline wandering and muscle tremor [112]. The movement of patient or electrodes is primarily responsible for shifting of the baseline causing delay in the electrochemical equilibrium at the electrode-skin interface and muscle tremor is generally observed in older patients. The biomedical signals are passed through filter stages removing undesirable noise components along with occasional amplification of the signal.

Mapping Methodology of FIR

In the light of important role of filters in biomedical signal processing, and digital signal processing in general, the FIR filters mapping scheme on the SAC architecture is developed and is discussed now. The SAC architecture readily supports FIR filter mapping as both entities share the same mathematical foundation *i.e.* the multiple-accumulate operation. The developed architecture supports thirty-six taps and 9-b signed arithmetic for filtering functions. The filter coefficients are applied to the coefficient registers of RUs and input data ($x[n]$) is applied to RU #1 through the $Dataext1$ input. The RUs can be connected back to back by means of configuration and bypass multiplexers forming a systolic array of thirty-six functional units. For such a configuration, all RUs are active and the data from the previous RU is connected to next RU data register by forcing appropriate select signal on the configuration and bypass multiplexers as illustrated in Fig. 4.7. For instance, RU #4 accepts data from RU #3 through configuration multiplexers $CM4A$ and $CM4B$ attached to '1' and '0', respectively. The bypass multiplexer, $BP4$ is set for the data to move forward to RU #4 data register. For RU chains $<$36, the additional RUs can be disconnected from the active datapath by clearing the bypass multiplexer select line. The architecture emulates 9, 18 or 27-tap FIR filtering by using one, combination of two or combination of three tiles, respectively. These topologies are previously discussed in detail in Chapter 3 section 3.3. Several other topologies for FIR filter chains is possible and one such topology is discussed now as an example. A scheme for 11-tap FIR filter mapping is presented that needs 11 RUs for SAC architecture implementation. Among numerous ways of choosing these RUs, two schemes are presented. First combination constitutes choosing a tile of 9 RUs and a section of three RUs situated next to the tile. For instance, Tile #1 and RUs 16, 17 and 18. In this selection, RU #18 must be loaded with 0 to circumvent its effect in the result. It must be noted that the RUs can be chosen or bypassed in a group of three because of the chosen arrangement of bypass multiplexers in the architecture. The second combination can be formed by joining RU #1-12 and loading 0 in RU #12 coefficient. Both these schemes uses 12 RUs in the datapath and the remaining 24 RUs are bypassed.

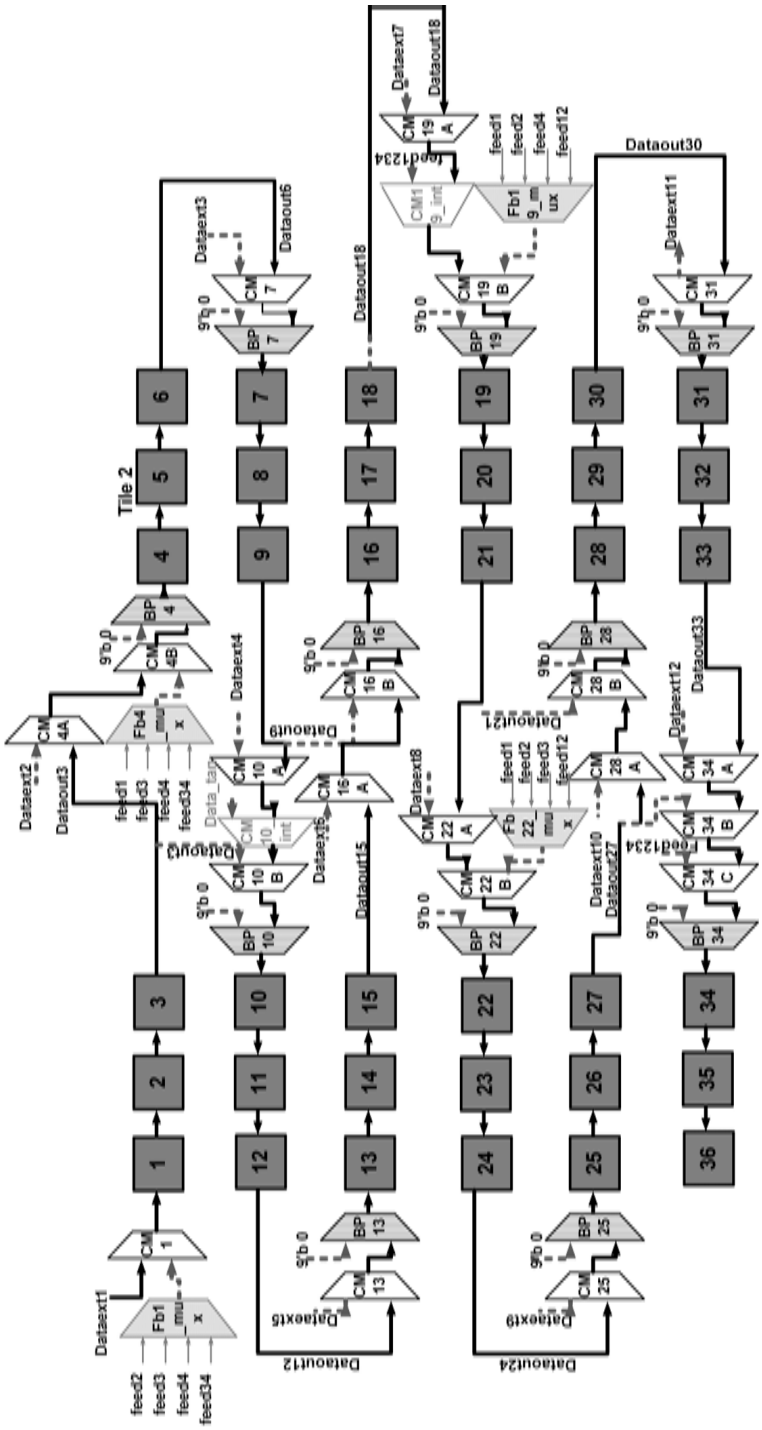


Figure 4.7: Active RUs and datapath for FIR mapping.

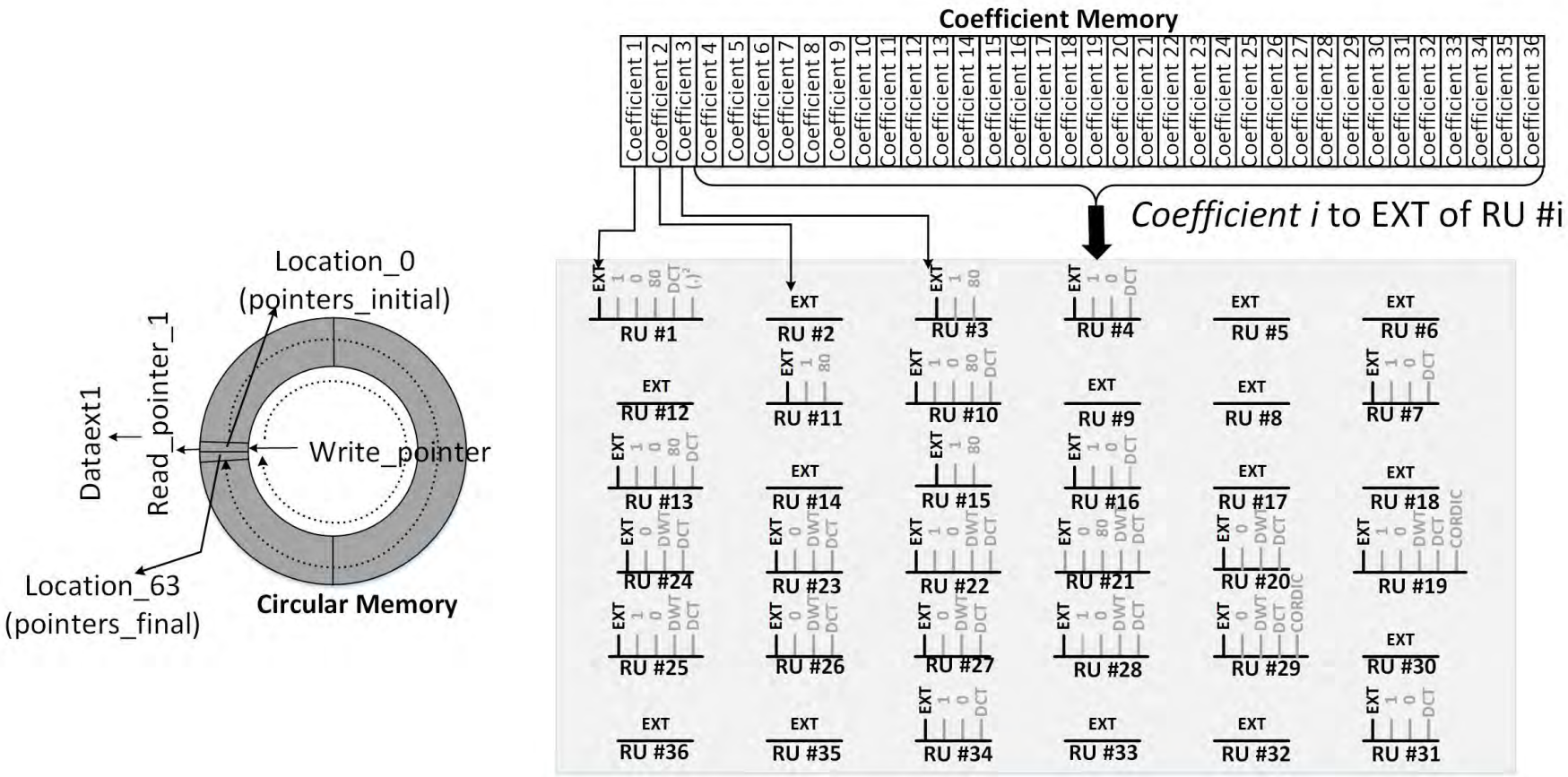


Figure 4.8: The circular and coefficient memory connections for FIR mapping.

The data and coefficient for FIR filtering is stored in circular and coefficient memories, respectively. The write pointer writes data till the circular memory is completely full. Following this, the read pointer reads data and the cycle repeats. The data is read through one read pointer which in turn is connected to *Dataext1* input of architecture. The coefficient memory is connected to *EXT* input of coefficient multiplexers and coefficient registers receive filter coefficients from it. The architecture connections with circular and coefficient memories are illustrated in Fig. 4.8.

Often in signal processing applications, noise cancellation is done in multiple steps by using different filters handling different types of noise. This would require the output of one filter to be fed back as input to another filter. For instance, a notch filter is used to remove the 50 Hz noise followed by the low pass filter cancelling the high frequency artefacts. The SAC architecture supports multiple filter realization following the four tile structure topology discussed in Chapter 3 section 3.3. This is possible due to the modular nature of the computation unit (CU) of the architecture that accumulates products received from tiles independent of one another. The output of a tile (or combination of tiles) can be connected to input of other tile using the feedback multiplexers of the architecture. This also facilitates IIR filter realizations that involves weighted sum of past outputs ($y[n-i]$) along with the inputs for computing the output.

4.2 Fixed Coefficient Functions

Functions grouped in this category perform computations following their underlying algorithm. This includes performing a fixed set of operations on the input.For instance, an algorithmic step may include multiplying a number by 2. In analogy to the architecture, this operations breaks down to multiplying input data by fixed coefficient, number 2 in this case. This step can be hard-wired as shift operation in a state machine when implementing it in hardware. Following the similar approach, function whose operation is based on a algorithm is mapped on a architecture using state machine. The state machine breaks the algorithm into a

series of simple steps performed in an orderly manner. This includes modifying the input by forcing a fixed sequence of coefficients as directed by the algorithm. As the coefficients for the function is fixed, they are supplied from within the state machine wherever needed.

4.2.1 COordinate Rotation DIgital Computer (CORDIC)

CORDIC algorithm proposed by Volder in [114, 115], is derived from the Givens rotation transform (shown in Fig. 4.9) that serves as the conventional method of computing 2-D vector rotation. A vector $P_0 = (x_0, y_0)$ when rotated counter-clockwise in the Cartesian plane by an angle ϕ results in vector $P_n = (x_n, y_n)$ represented by eqn 4.10, further simplified as eqn 4.11 [116] :

$$x_n = x_0 \cos\phi - y_0 \sin\phi; \quad y_n = x_0 \sin\phi + y_0 \cos\phi \tag{4.10}$$

$$\Rightarrow x_n = \cos\phi[x_0 - y_0 \tan\phi]; \quad y_n = \cos\phi[y_0 + x_0 \tan\phi] \tag{4.11}$$

The hardware realization of these equations require four multiplications, two signed additions and accessing the table with trigonometric coefficients. However, according to the algorithm, arbitrary angles of rotation (ϕ) can be expressed as a series of successively smaller elementary rotations δ_i *i.e.* $\phi = \delta_1 + \delta_2 + \delta_3 + \cdots + \delta_n$.

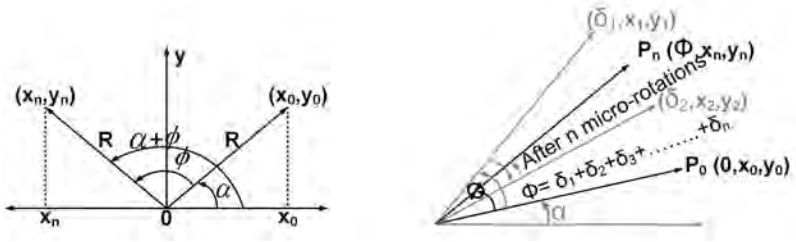


Figure 4.9: (a) Given's Rotation (b) Elementary angles of rotation.

The elementary rotations when restricted such that $\tan\delta_i$ equals $\pm 2^{-i}$ results in the iterative rotation given by Eqn 4.12. The selection of δ_i's translates the $tan\phi$ multiplication to simple shift operation reducing the computation operation to only shift and add operations.

$$x_{i+1} = K_i[x_i - y_i \cdot 2^{-i} \cdot d_i]$$

$$y_{i+1} = K_i[y_i + x_i \cdot 2^{-i} \cdot d_i] \tag{4.12}$$

$$z_{i+1} = z_i - arctan(2^{-i}) \cdot d_i \tag{4.13}$$

where, K_i and $d_i = \pm 1$ denotes the gain and direction of rotation i, respectively.

Eqn 4.13 represents decision variable that minimizes ϕ (angle offset) and determines the subsequent direction of rotation. The terms $arctan(2^{-i})$ are precomputed and stored internally in a memory array. The values of $arctan(2^{-i})$ expressed in decimal and binary is presented in Table 4.1.

Table 4.1: The value of $arctan(2^{-i})$ with 2-bit precision.

i	2^{-i}	$arctan(2^{-i})$	
		Dec	Binary
0	1	45°	101101.00
1	0.5	26.56°	011010.10
2	0.25	14.03°	001110.00
3	0.125	7.125 °	000111.00
4	0.0625	3.576°	000011.10
5	0.03125	1.789°	000001.11
6	0.015625	0.895°	000000.11
7	0.0078125	0.447°	000000.01

After n CORDIC iterations, the rotation equation becomes:

$$\lim_{n\to\infty} \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} = K \times \begin{pmatrix} x_0 \cos z_0 - y_0 \sin z_0 \\ x_0 \sin z_0 + y_0 \cos z_0 \\ 0 \end{pmatrix} \tag{4.14}$$

where the $scalefactorK$ is equal to $\prod_{n=0}^{\infty} \sqrt{1 + 2^{-2n}} \approx 1.646760$.

An extended and generalized version of the algorithm, given by Eqn 4.15 [117, 118], computes hyperbolic and non-linear functions (logarithm, square root *etc.*) in addition to the standard trigonometric functions. Figure 4.10 represents implementation of one iteration of generalized CORDIC equation given by eqn 4.15. These equations are applicable in *Circular*, *Linear* and *Hyperbolic* coordinate

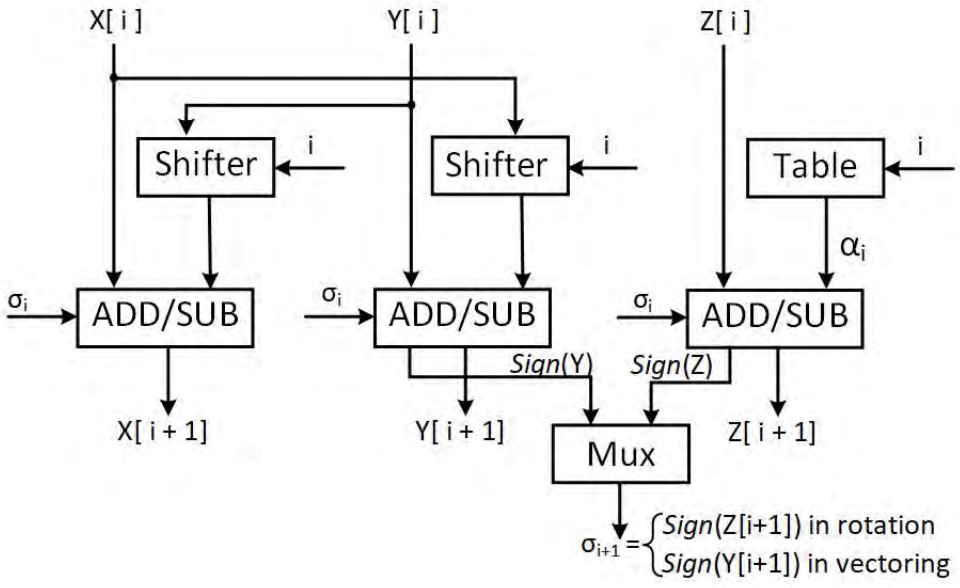


Figure 4.10: Implementation of one iteration.

systems in both, *Rotation* and *Vectoring*, modes of operation.

$$\begin{aligned} x_{i+1} &= x_i - m \cdot y_i \cdot 2^{-i} \cdot d_i \\ y_{i+1} &= y_i + x_i \cdot 2^{-i} \cdot d_i \\ z_{i+1} &= z_i - \alpha_i \cdot d_i \end{aligned} \tag{4.15}$$

The direction of rotation, d_i, depends on z and y for *Rotation* and *Vectoring* modes, respectively. d_i is 1 if $z > 0$ and is -1 otherwise in *rotation* mode whereas in *vectoring*, d_i is 1 if $y < 0$ and is -1 otherwise. Additionally, the coordinate system variable m and arctan variable α depends on the computed function and cause minor changes in the generalized CORDIC equations. Table 4.2 provides the values m and α assumes for various functions computed using CORDIC along with the respective seed values x_0, y_0, z_0 [119].

CORDIC Architectures and Applications

Few architecture for mapping CORDIC algorithm onto hardware are discussed in this section. In general, the architectures are classified as folded and unfolded, based upon the realization of the three iterative equations. Duplicating each of the CORDIC algorithm difference equation into hardware and time multiplexing

Table 4.2: The hyperbolic variable (m), the mode of operation, arctan variable (α) along with intial guess and post-processing for the supported functions.

Mode of operation	Function	m	α	Initial Guess	Post-Processing
Rotation	Rsinθ, Rcosθ,Polar to Rectangular	1	$\tan^{-1}(2^{-i})$	x_0=R=$\mid P_0 \mid$, y_0=0, z_0=θ	$\tan\theta = \sin\theta/\cos\theta$
	Rsinhθ, Rcoshθ, tanhθ, Re^{θ}	-1	$\tanh^{-i}(2^{-i})$	x_0=R=$\mid P_0 \mid$, y_0=0, z_0=θ	$\tanh\theta = \sinh\theta/\cosh\theta$, $e^{\theta} = \sinh\theta + \cosh\theta$
	$(a \cdot b)$	0	2^{-i}	x_0=a, y_0=0, z_0=b	Not Required
	$\tan^{-1}$	1	$\tan^{-1}(2^{-i})$	x_0=a, y_0=b, z_0=0	Not Required
	$\tanh^{-1}$	-1	$\tanh^{-1}(2^{-i})$	x_0=a, y_0=b, z_0=0	Not Required
Vectoring	$\sin^{-1}$,$\cos^{-1}$	1	$\tan^{-1}(2^{-i})$	x_0=1, y_0=a, z_0=0	$\cos^{-1}(a) = \sin^{-1}(a)$ - $(\pi/2)$
	(a/b)	0	2^{-i}	x_0=b, y_0=a, z_0=0	Not Required
	$\sqrt{a}$, $\ln a$	-1	$\tan^{-1}(2^{-i})$	x_0=a+1, y_0=a-1, z_0=0	$\ln(a) = 2 \cdot z_n$

m takes the value 1, 0 and -1 for Circular, Linear and Hyperbolic coordinate systems, respectively.

the iterations constitute folded architecture. Folding architectures can be further categorized as bit-serial and word-serial architectures depending upon whether the functional unit implements logic for one bit or one word of each iteration. The folded architectures are inherently slow as they trade-off area for time. [114] implemented CORDIC using the bit serial architecture where same hardware was used to execute all the iterations. The word serial architecture in [117] (Fig. 4.11), modifies the shifters with each iteration required to obtain the desired shift and elementary angles were accessed through the lookup table. This architecture suffered in performance due to the carry/borrow propagation delay in critical paths. These limitations were overcome by unfolding the iteration process. This was achieved by adopting hard-wired shifters replacing the barrel shifters, and elimination of ROM this ensures each iteration is carried out in different part of the hardware. The pipelined architectures also offer throughput improvement.

CORDIC algorithm has attracted a lot of attention from academia and industry primarily because of simplified computation and architecture as well as for its various applications in DSP, biomedical signal processing, software defined radio,

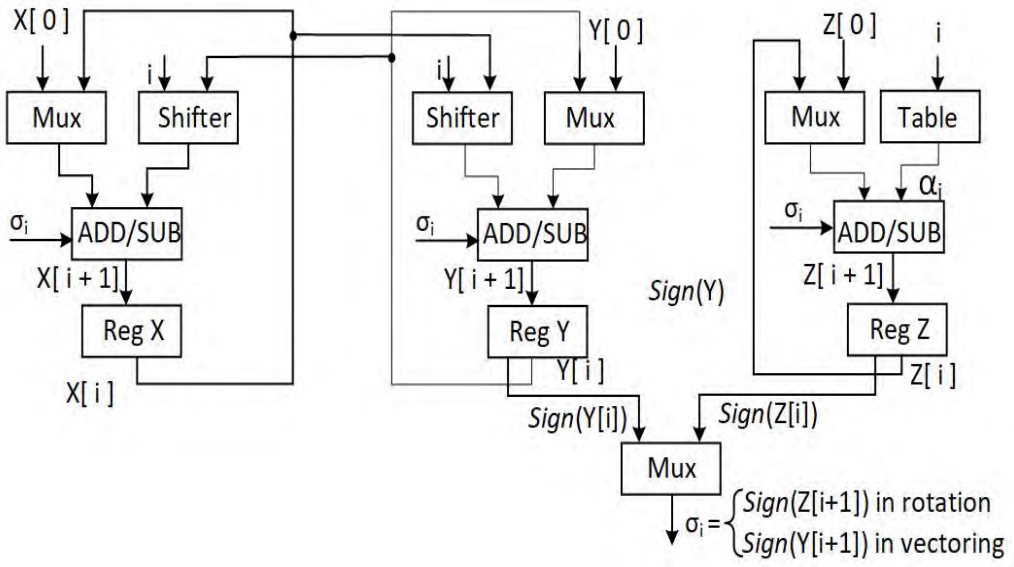


Figure 4.11: Word-serial implementation.

neural networks, and MIMO (Multiple-Input Multiple-Output) systems *etc*. In signal processing, CORDIC techniques are used in adaptive filtering and computation of sinusoid transforms such as DFT (Discrete Fourier Transfrom) [120], DCT (Discrete Cosine Transform), DHT (Discrete Hartley Transform) [121] *etc*. The use of FFT (Fast Fourier Transform) for spectral analysis in biomedical signal processing is worth mentioning. The spectral information obtained from FFT is a useful clinical format for interpreting biomedical signals [122] and is used to monitor muscle fatigue from EMG [123], diagnosing disorders related to stomach muscles and intestines from EGG (ElectroGastroGram) [124], analysis of cardiac arrhythmias using ECG [125–127]. Many CORDIC based FFT processors [128–132] are discussed in literature owing to benefits of reduced hardware implementation without compromising the operation speed.

Mapping Methodology of CORDIC

Each CORDIC iteration requires implementing three equations (4.15) calculating x, y and z. The CORDIC computation uses an adder/subtractor and the previous iteration result. Right-shift operation ($>>$) can be visualized as multiplication with 2^{-i}. It is fair to assume that the proposed SAC architecture can be used to realize CORDIC that supports weighted sum and difference equations. The

SAC architecture performs multiplication by means of AND-XOR operation in RU, where the 9-b coefficient register is loaded with the content 2^{-i} required for the i^{th} iteration. The data register is loaded with the x, y or z variable (or α value) computed in the previous iteration while subsequent additions are carried out in the CU.

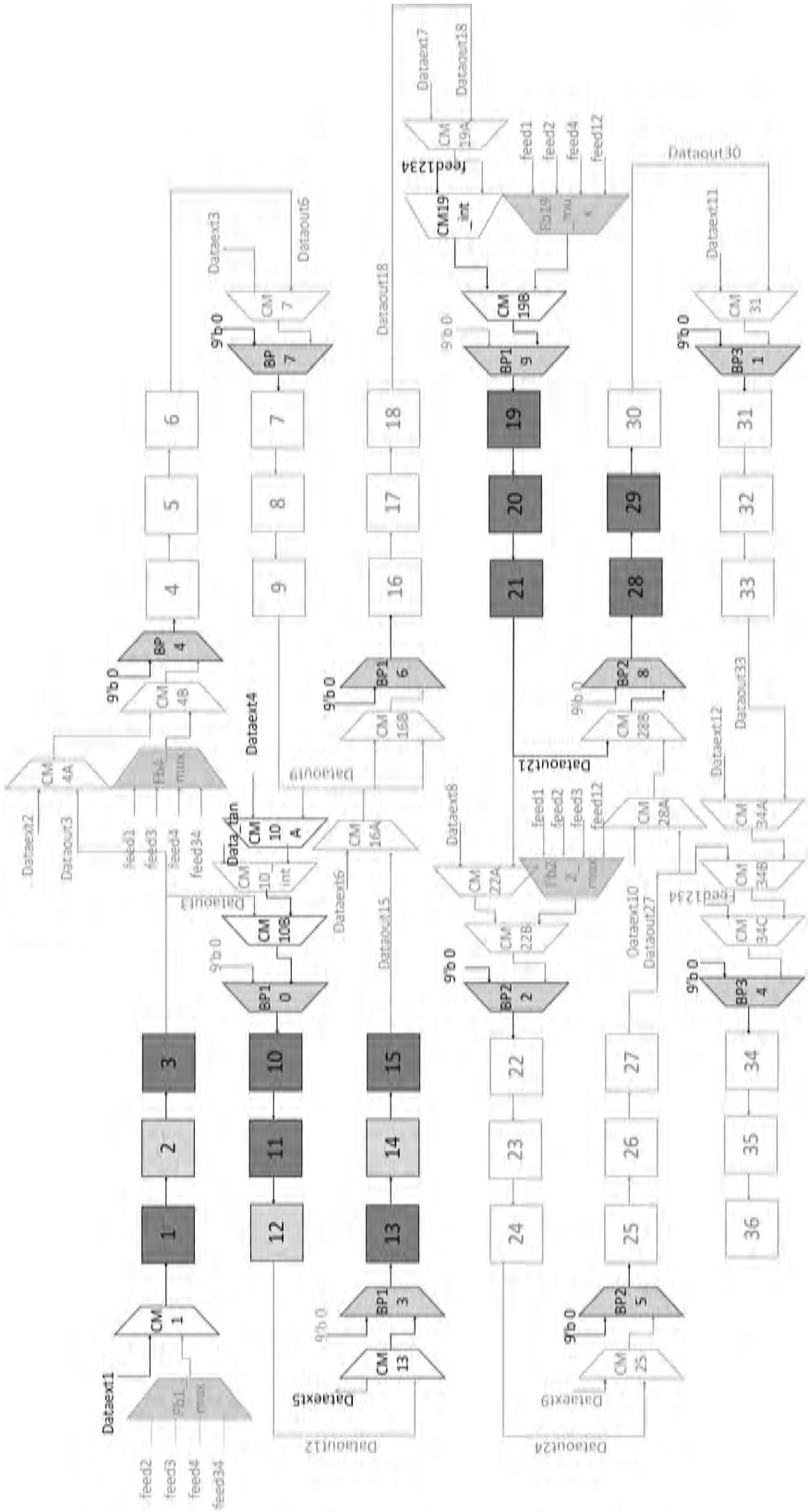


Figure 4.12: Active RUs and datapath for CORDIC mapping.

Tile #1 and 3 are used for CORDIC mapping. The RU triplets are not connected with each other in both the tiles except for RU# 19-21 with 28-30 forming a 6 RU chain as shown in Fig. 4.12. Among these RUs the light gray shaded blocks do not contribute to the computations according to the developed mapping methodology though they are the part of active RUs. Two internal multiplexers *CM10_int* and *CM19_int* are used in CORDIC mapping and are controlled by the CORDIC algorithm state machine. The variables (x, y or z) are computed sequentially leading to serial computation of the three variables, once every 8 clock cycles. Therefore, it takes 24 clock cycles to complete one iteration. The outcomes of the i^{th} iteration, x_i, y_i and z_i, serves as input for the next iteration and thus needs to be stored and fed back to the data register of RU. To provide an overview of the operation, we discuss how each of the variables is computed in the proposed architecture in the following steps A - G.

A. During the initial step, the direction of rotation is determined for the first iteration and it depends on the mode of operation *i.e.* rotation or vectoring and is as following:

Mode = Rotation	Mode = Vectoring
If $z_0 \geq 0$(positive) $\Rightarrow$ d_0 = +1, otherwise d_0 = -1.	If $y_0 \geq 0$(positive) $\Rightarrow$ d_0 = -1, otherwise d_0 = +1.

   In the SAC architecture, the sign bit (MSB) of z_0 or y_0 indicates whether the data is positive or negative. This sign bit is XORed with the sign (without the effect of direction) of the equation ('0' for addition and '1' for subtraction) to produce the final sign. For example, in eqn 4.15, for x_{i+1}, the actual sign of the equation is negative (-) *i.e* '1' in sign bit. Now, if the direction variable is positive (+) or '0' in the sign bit, the final sign is obtained by XORing the two sign bits *i.e.* '1' $\oplus$ '0' = '1'(-ve).

B. Then architecture is loaded with y_0 in RU #1, z_0 in RU #10, and x_0 in RU #13 through *Dataext1*, *Dataext4* and *Dataext5* inputs of SAC architecture.

C. Then y_1 is computed, which is dependent on the values of y_0 and x_0. There-

fore, coefficient of the RU containing y_0 and x_0 are set with non-zero coefficients whereas the remaining coefficients are set to 0. After 8 clock cycles, y_1 is computed. It is stored in RU #19 (fedback) through the *feed1234* input and *CM19_int* (see Fig. 4.12) multiplexer at the 8^{th} clock cycle. The CORDIC algorithm state machine ensures *feed1234* is passed through the *CM19_int* multiplexer by clearing the respective select line. Table 4.3 shows the RU occupancy at different clock instances along with coefficients applied to coefficient registers.

D. Variable y_1 (the newly computed output) is stored in RU #19 at the 8^{th} clock cycle. At the same time, the data register contents shift into the next in line RU after every 8 clock cycles. Thus, x_0 is shifted from RU #19 to RU #20; y_0 is shifted to RU #2. Similarly, z_0 is shifted to RU #11 and α_0 is applied to RU #10.

E. After this, computation for z_1 commences. The state machine switches the RU #7 input from z_0 to arctan variable using the *CM10_int* internal multiplexer for further computations and *Data_tan* input is applied to RU #7 data register. The z variable is a function of z_0 and arctan variable (α_0) that is stored in RU #11 and RU #10, respectively. The coefficient for RU #11 and RU #10 are non-zero and all the remaining coefficients (including the previous non-zero coefficients) are set to 0. After 8 clock cycles, z_1 gets computed and z_1 gets stored in RU #19 at 16^{th} clock.

F. In this step x_1 computation begins that is a function of x_0 and y_0. Therefore, the coefficient for RU #15 and RU #3 are non-zero and rest are 0. Variable x_1 is computed in the next 8 clock cycles.

   It should be noted that the x variable is computed at the end because it has no role in deciding the direction variable (discussed in step A). Furthermore, the time in which x computation is carried out, the computation for the next iteration decision variable is done in parallel.

G. The x variable is stored in RU #19 and the data register shifts (and copies)

Table 4.3: The RU coefficients for the CORDIC variables computation

Iteration #	Variable Computed	Variable (RU #)	Coefficient (in 0X)
1	y_1	$\langle y_0(1), x_0(13)\rangle$	80,80
	z_1	$\langle z_0(11), \tan^{-1}(2^0)(10)\rangle$	80,80
	x_1	$\langle x_0(15), y_0(3)\rangle$	80,80
2	y_2	$\langle y_1(21), x_1(19)\rangle$	80,40
	z_2	$\langle z_1(21), \tan^{-1}(2^{-1})(10)\rangle$	80,80
	x_2	$\langle x_1(21), y_1(29)\rangle$	80,40
3	y_3	$\langle y2(21), x_2(19)\rangle$	80,20
	z_3	$\langle z_2(21), \tan^{-1}(2^{-2})(10)\rangle$	80,80
	x_3	$\langle x_2(21), y_2(29)\rangle$	80,10
4	y_4	$\langle y_3(21), x_3(19)\rangle$	80,40
	z_4	$\langle z_3(21), \tan^{-1}(2^{-3})(10)\rangle$	80,80
	x_4	$\langle x_3(21), y_3(29)\rangle$	80,10
5	y_5	$\langle y_4(21), x_4(19)\rangle$	80,08
	z_5	$\langle z_4(21), \tan^{-1}(2^{-4})(10)\rangle$	80,80
	x_5	$\langle x_4(21), y_4(29)\rangle$	80,08
6	y_6	$\langle y_5(21), x_5(19)\rangle$	80,04
	z_6	$\langle z_5(21), \tan^{-1}(2^{-5})(10)\rangle$	80,80
	x_6	$\langle x_5(21), y_5(29)\rangle$	80,04
7	y_7	$\langle y_6(21), x_6(19)\rangle$	80,02
	z_7	$\langle z_6(21), \tan^{-1}(2^{-6})(10)\rangle$	80,80
	x_7	$\langle x_6(21), y_6(29)\rangle$	80,02
8	y_8	$\langle y_7(21), x_7(19)\rangle$	80,02
	z_8	$\langle z_7(21), \tan^{-1}(2^{-7})(10)\rangle$	80,80
	x_8	$\langle x_7(21), y_7(29)\rangle$	80,02
In General for j > 0			
j	y_{j+1}	$\langle y_{(j)}(21), x_{(j)}(19)\rangle$	80, $(j+1)^{th}$ bit from MSB is set
	z_{j+1}	$\langle z_j(21), \tan^{-1}(2^{-(j)})(10)\rangle$	80,80
	x_{j+1}	$\langle x_{(j)}(21), y_{(j)}(29)\rangle$	80,$(j+1)^{th}$ bit from MSB is set

its content to the next in line RU. This marks the end of first iteration which takes 24 clock cycles.

H. The steps from A to G are repeated 7 more times with the subscript index (i) increased by 1 everytime. The variables y_8, z_8 and x_8 are computed after eight iterations as a final step.

The above steps are explained considering the following example for comput-

ing sin30°. In this case, the CORDIC is operated in non-hyperblic rotation mode with seed values provided in Table 4.2. The decision variable (based on the sign bit of z_0) is +1. Therefore, the CORDIC equations retains its actual sign as the final sign. As listed in steps A - G, y_1 is evaluated where the coefficient for RU #1 and RU #13 are set to 0x80 with 7-bit precision. The resultant coefficient value is therefore 1.0000000b (= 1d) where the decimal point is taken after the 7^{th} bit position. After 8 clock cycles y_1 is computed and stored in RU #19. The RU data registers shifts content to the next in line RU. Afterwards z_1 computation is initiated. For z_1, RU #11 and RU #10 coefficients are set to 0x80. At 16^{th} clock cycle, z_1 is stored in RU #19 and the RUs shift its data to the following RU. Finally, x_1, is computed after this, by loading 0x80 in coefficients of RU #15 and RU #3. At 24^{th} clock cycle, the iteration ends with x_1 being computed and stored in RU #19. It continues for 8 iterations for y_8, z_8 and x_8 computations following the occupancy and coefficients mentioned in Table 4.3.

It must be noted that, previous value of variable under computation is present in RU #21 for all three variables. The other variable in the equation (x or y) lies in RU #19 and 29, and is present in shifted form after 1^{st} iteration. The same sequence is followed for the following iterations, using this methodology the RU sites for variable computations is fixed *i.e.* x variable is computed using RU #21 and 29 for all iterations after 1^{st}. Additionally, this helps in designing a state machine with fewer states as RU occupancy follows a certain pattern as opposed to being random. The algorithmic state machine shown in Fig. 4.13 consists of six states and depicts the RU coefficients loaded in each state during the coarse of CORDIC computation.

An alternate mapping scheme would result by computing variables in three different tiles owing to the internally bifurcated structure of CU. However, the computed variable are required to be fed back to RUs to retain their previous values for further computations. This loading takes 8 cycles per variable and require memory to store the variables computed in parallel. The adopted mapping scheme computing variables sequentially also takes 24 cycles for each iteration but stores results internally in RUs thereby does not use additional memory.

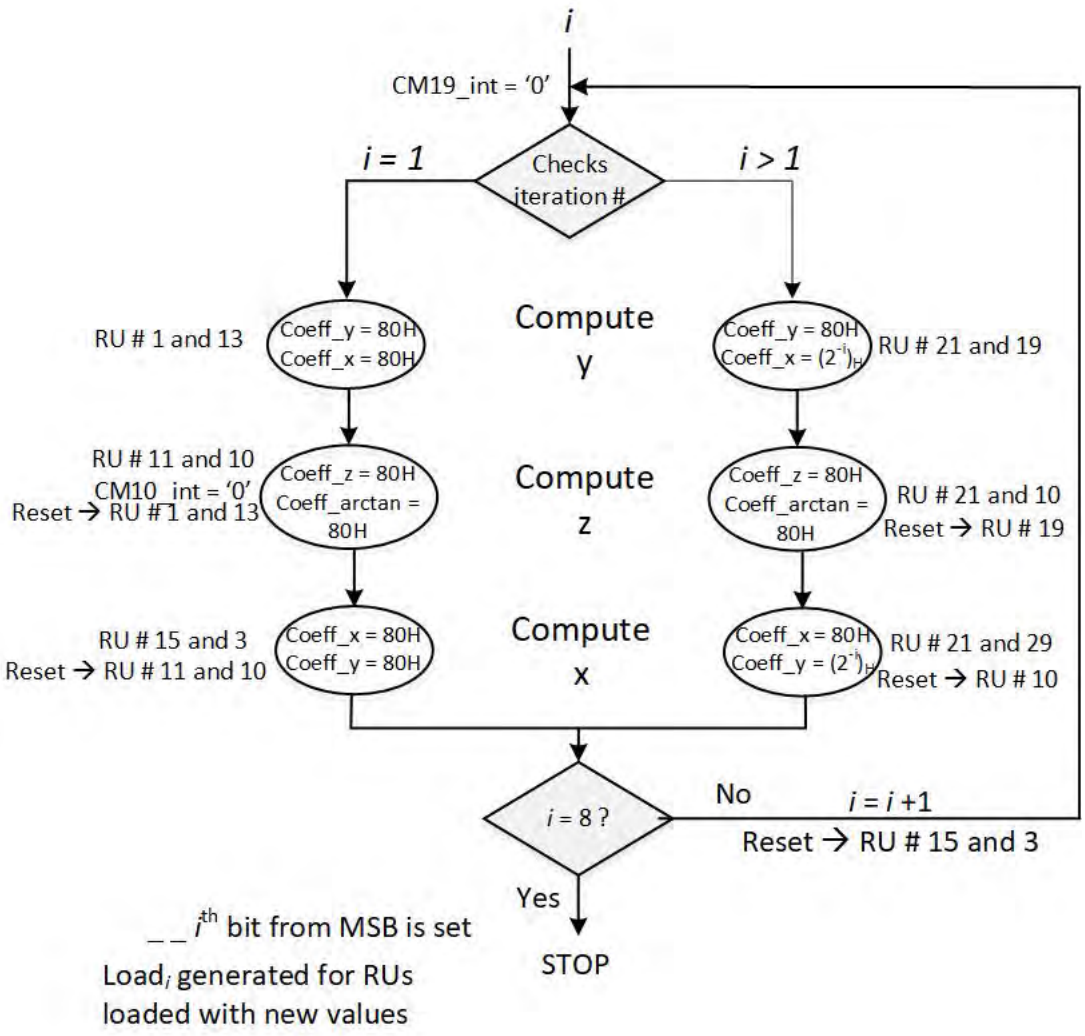


Figure 4.13: Flow chart of CORDIC state machine.

The seed for x, y and z are provided simultaneously from three read ports of circular memory. The coefficients for RUs used in CORDIC algorithm mapping are applied through the CORDIC input of coefficient multiplexers. A CORDIC multiplexer shown in Fig. 4.14 is connected to the coefficient multiplexer CORDIC input of RU #19 and 29. The multiplexer has 2^{-i} terms in binary as inputs and its select line is controlled by the state machine.

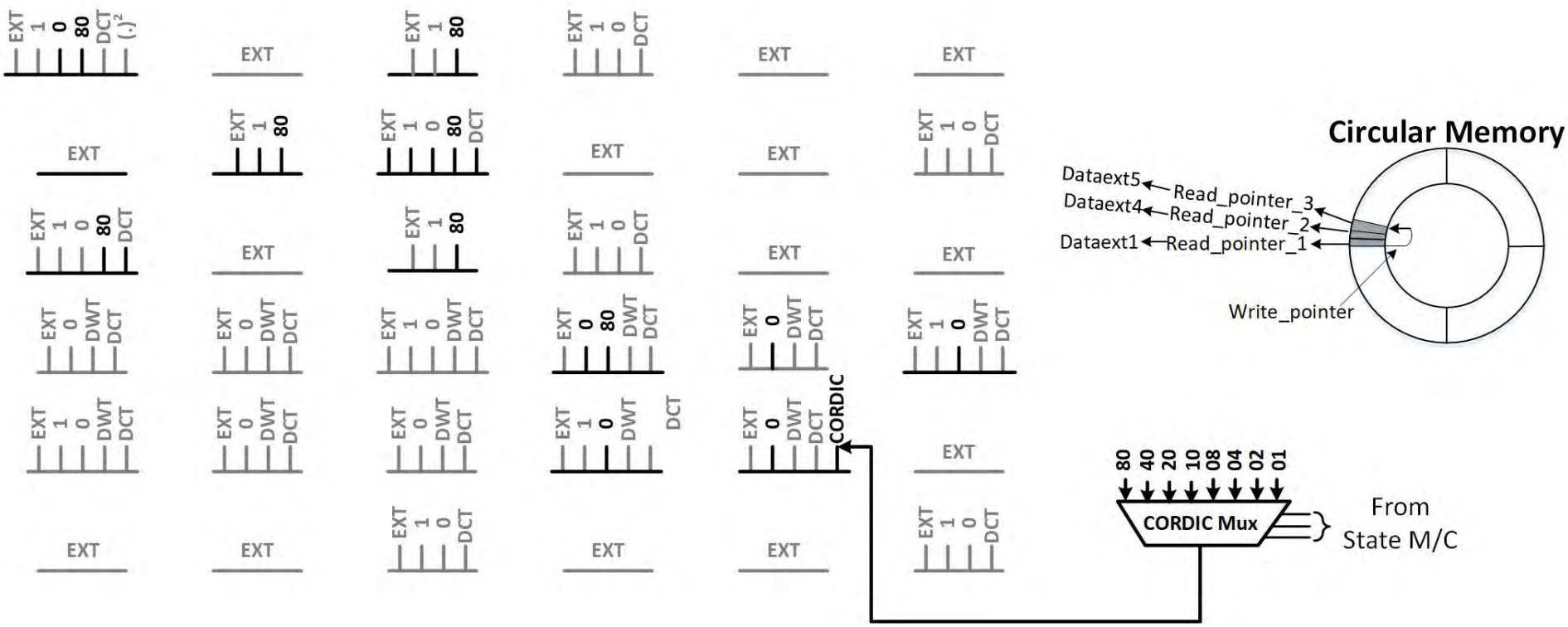


Figure 4.14: The circular memory and coefficient multiplexer connections for CORDIC mapping.

4.2.2 Compression Algorithms

Data compression is the process of encoding a set of data such that the size of the encoded set is smaller than the size of the original, unencoded data. It exists in all forms of digital communications, where maximising the amount of data that can be transmitted with a limited bandwidth is important. Compression algorithms involve two components; the compression algorithm which is used to create a representation of the data in its reduced form, and the reconstruction (or decompression) algorithm that is used to convert the compressed signal back to its original form. Data compression can be divided into two types: lossless and lossy [133]. Lossless compression involves no loss of information between the original data set and the one created after the compressed data has been decompressed/reconstructed, *i.e.* perfect reconstruction. Lossy compression involves some loss of signal information such that the reconstructed data is not exactly equal to that of the original. While lossless compression may seem like the best choice from the perspective of signal fidelity, the compression gains that can be achieved are generally quite limited. Lossy compression on the other hand allows for an inexact reconstruction of the data set. This allows the algorithm to discard elements of the signal it deems unnecessary to preserve the reconstructed data set or represent them in reduced form. The amount of loss allowed is usually application-specific and set by the system designer. Generally speaking, the greater the acceptable levels of loss, the higher the compression gains that can be achieved.

Discrete Wavelet Transform (DWT)

The Fourier transform has been a powerful tool in data analysis from a long time. However, it does not represent the abrupt changes efficiently. The reason being that the Fourier transform represent the data as the sum of sine waves, which are not localized in time or space. The underlying principle of this problem is the Heisenberg uncertainty principle, which states that a signal cannot be localized in time and frequency simultaneously. Thus, the researchers came out with a solution of splitting a signal into components, called wavelets, which are not

completely a sine wave [134] and are thus localized in both time and frequency. A wavelet is a short duration finite energy function with zero mean.

$$\int_{-\infty}^{+\infty} \psi(t)dt = 0 \tag{4.16}$$

where ψ is often referred to as the mother wavelet. This mother wavelet is then used to create other wavelets by means of dilating and shifting. Shifting helps in analyzing the specific portion of a signal whereas the scaling helps in differentiating the smooth and abrupt changes in the signal. Mathematically, a wavelet is given by the eqn 4.17 where a is the scaling parameter and b is the shifting parameter.

$$\Psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\Psi\left(\frac{t-b}{a}\right) \tag{4.17}$$

The DWT reveals that a signal's energy is often focused in a small number of coefficients, with the others tending towards zero. By exploiting this redundancy, it is possible to represent the signal in a more compact form by setting coefficients below a threshold value to zero [135]. In 1910, Alfréd Haar proposed a simple piecewise constant function whose dilation and translations generate an orthonormal basis in [136]. In 1988, Ingrid Daubechies published her seminal work on orthonormal wavelets of compact support [137]. This led to the development of filter bank based transforms [138] and biorthogonal wavelet bases [139, 140].

The 1-D DWT decomposes a signal of one level into two signals, approximation and detail, of another higher level. For example, if $A_i(n)$ and $D_i(n)$ are approximation and detail coefficients at level i then the approximation and detail coefficients of next higher level ($i + 1$) are given by eqns 4.18 and 4.19, where $h(k)$ is the low pass filter and $g(k)$ is the high pass filter and L is the size of the filter.

$$A_{i+1}(n) = \sum_{k=0}^{L-1} h(k)A_i(2n-k) \tag{4.18}$$

$$D_{i+1}(n) = \sum_{k=0}^{L-1} g(k)D_i(2n-k) \tag{4.19}$$

However, in 2-D DWT the same decomposition occurs in both row and col-

umn dimensions. Thus, the 2-D DWT decomposes a signal in four frequency sub-bands, namely LL (approximation (cA) matrix), LH (horizontal (cH) matrix), HL (vertical (cV) matrix) and HH (diagonal (cD) matrix). The Fig. 4.15 depicts the decomposition of an $N \times N$ Lena image into four sub-bands and its reconstruction from those sub-bands by convolution and sampling. Here, $h\_d(n)$ is low pass decomposition filter coefficients,$g\_d(n)$ is high pass decomposition filter coefficients, $h\_r(n)$ is low pass reconstruction filter coefficients and $g\_r(n)$ is high pass reconstruction filter coefficients. The sampling done is dyadic *i.e.* alternate samples are retained.

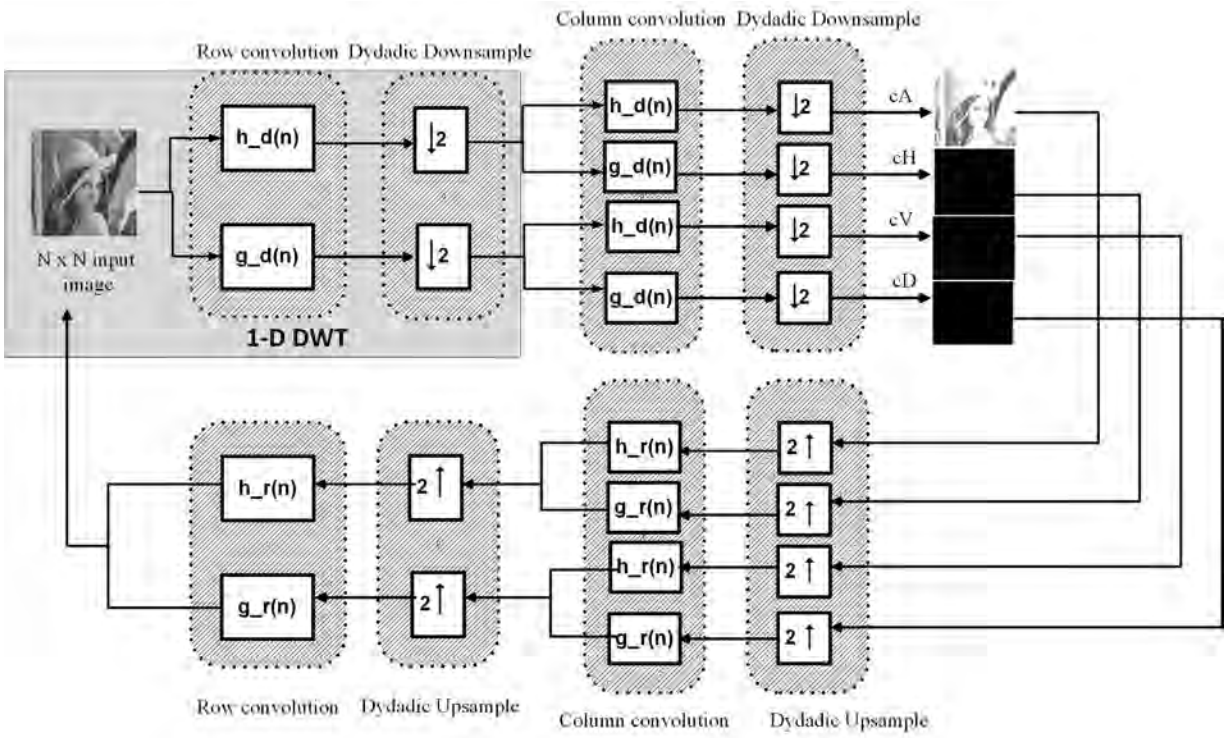


Figure 4.15: Decomposition and reconstruction of an image using 2D-DWT

DWT Architectures and Applications

The straight forward implementation for DWT is to construct the low-pass and high-pass filters independently. However, such direct mapping results in computational waste of upto 50% because the computed signals will be down-sampled by 2 in the DWT process and conversely half signal will be 0 in IDWT process. This can be minimized by the use of polyphase decomposition technique, which

moves the downsampling/upsampling operations to the front/back of the filter banks. A convolution based architecture is usually used if the required throughput is two-input/two-output per clock cycle in minimum latency and hardware resources. Fig. 4.16 illustrates one of the possible convolution based architecture for 1-D DWT computations. Few of the alternate architectures are based on lifting scheme, B-spline scheme *etc* that are discussed in [141].

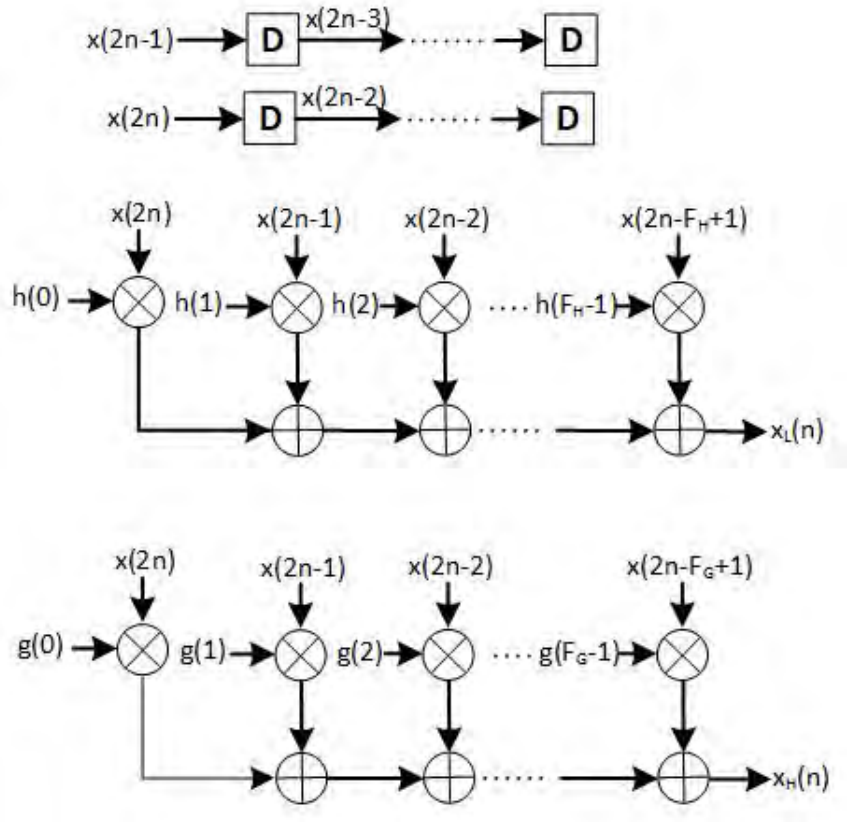


Figure 4.16: Convolution-based architecture using parallel filters. F_H and F_G are the length of low-pass and high-pass filters, respectively.

As far as 2-D DWT architectures are concerned, RAM-based architectures [142] are most practical for real-life design due to their inherent regularity and storage density [143]. The large amount of internal memory size and external access bandwidth is the bottleneck of the 2-D DWT implementations. Consequently, the memory issues dominate the hardware cost and architectural complexity.

In the *direct scan* implementation, 2-D DWT is computed by successively performing 1-D DWT in two directions *i.e.* row and column. An external frame memory is used to store the result of 1-D DWT in one direction commonly called

intermediate coefficients. For the other decomposition levels, the LL sub-band of the present level is treated as input signal for next level. This scheme requires no internal memory but huge external memory access. As, the priorities of row and column directions 1-D DWT are identical, the order can be assumed as row-column for odd-level decompositions and column-row for even-level decompositions [144]. This interpretation leads to *row-column-column-row (RCCR)* scan which decreases the external memory access bandwidth by one-half for every level except for the first level decomposition as compared to direct scan method. A data buffer is used to reduce the power intensive external memory access [143] in the *Line-based scan* scheme. The coefficients are stored on on-chip buffer after the 1-D DWT limiting the external memory access to reading image data and writing 2-D DWT coefficients.

DWT is often used as compression methods in image and biomedical signals by exploiting signal sparsity, most commonly, in frequency domain [145,146]. Authors in [147] present a wavelet and wavelet packet based EEG compression algorithm that uses a lossy compression scheme where the acceptable level of loss is largely application dependent. Another EEG compression approach based on WT and Embedded Zero-tree Wavelet (EZW) encoding is presented in [148]. In this method, differential encoding is leveraged to maximize compression by exploiting the intra-channel redundancy present in EEG signals. An ECG compression approach is outlined in [149] includes the use of Huffman entropy coding along with novel thresholding method for wavelet coefficients. Apart from compression usage, there exist several techniques for acquisition, delineation and characterization of bioelectric signals that are based on wavelet transform. The use of wavelet transform for detection of ECG characteristic points was first reported in [150]. Later, derived wavelets, such as Quadratic Spline Wavelet Transform (QSWT) reported in [61,65], were used for QRS detection. QSWT could be readily implemented using FIR filter with reduced taps due to its compact support characteristics. The R-wave gets transformed to a modulus maxima pair after QSWT operation from which R-wave can be determined conveniently by spotting zero crossing. Several approaches based on hybrid wavelet [151], multi-domain anal-

ysis [152], continuous WT [153], lifting-scheme [154] are reported in literature. Among them, [151, 152] are crafted for abnormal ECG waveforms providing acceptable results while detecting a variety of arrhythmias including right bundle branch block, atrial fibrillation, normal sinus rhythm *etc.* whereas [153, 154] are more suited for real time analysis and processing of ECG.

Mapping Methodology of DWT

A generic mapping methodology is developed that supports twenty eight wavelets with filter size <8 [155]. The DWT algorithm first computed low pass filter convolution followed by factor 2 down sampling. The resultant vector is convolved with high pass filter and DWT matrix is obtained by another factor 2 down sampling. The generic methodology is developed for filter length = 8. Additionally, periodic padding is necessary to ensure that at both boundaries of the signal, atleast one signal sample exists and spatially corresponds to each coefficient of the filter mask. The number of additional samples required at the left (*exthl*) and right (*exthr*) of the signal is filter-length Len_{filter} dependent [156] and is given by eqns 4.20 and 4.21. For the wavelets with filter length < 8, zeros are padded on both sides at appropriate locations. For example, for Biorthogonal wavelet bior2.2, $Len_{filter} = 5$ which is increased to 8 by padding 1 zero on right and 2 zeros on left. The resulting filter coefficients are shown in Fig. 4.17. DWT computation steps are shown in Fig. 4.18 wherein the row wise convolution is carried out on 14×8 padded image and intermediate matrix, Y is generated. Columns 0-3 of padded image is same as columns 5-8 of the actual image. Thus, the convolution filter mask is modified by swapping the first four terms with the last four terms. This eliminates the use of excess memory elements required to store the padding. Furthermore, the mask advances by two steps as opposed to one step in the conventional convolution performing down sampling along with the intermediate matrix generation. This technique also reduces the number of computations by limiting the calculations to those elements that are retained after down sampling. The second set of convolution is performed on transposed padded Y matrix. The redundant computations elimination and mask manipulations are repeated generating the *cA DWT* ma-

trix. In order to achieve image compression, LL sub-band *i.e.* the approximation coefficient is retained and the other three set of coefficients namely horizontal, vertical and diagonal are considered zero to achieve a compression of 75% after 1-level of decomposition. However, the compression can be further increased by the increasing the number of levels of decomposition.

$$exthl = floor(lenh/2) \tag{4.20}$$

$$exthr = lenh - exthl - 2 \tag{4.21}$$

0	0	-0.1768	-0.3536	1.0607	-0.3536	-0.1768	0

Figure 4.17: Coefficients of bior2.2 after zero padding

The following 28 wavelets can be supported through the developed methodology and Table 4.4 shows their upscaled decomposition matrix with filter size 8.

- Haar Wavelet
- Biorthogonal Wavelets : bior3.1, bior2.2, bior1.3, bior3.3
- Reverse Biorthogonal Wavelets: rbio1.1, rbio1.3, rbio1.5, rbio2.2, rbio2.4, rbio2.6, rbio2.8, rbio3.1, rbio3.3, rbio3.5, rbio3.7, rbio3.9, rbio4.4
- Fejer-Korovkin: fk4, fk6, fk8
- Coiflets : coif1
- Daubechies: db2, db3, db4
- Symlets: sym2, sym3, sym4

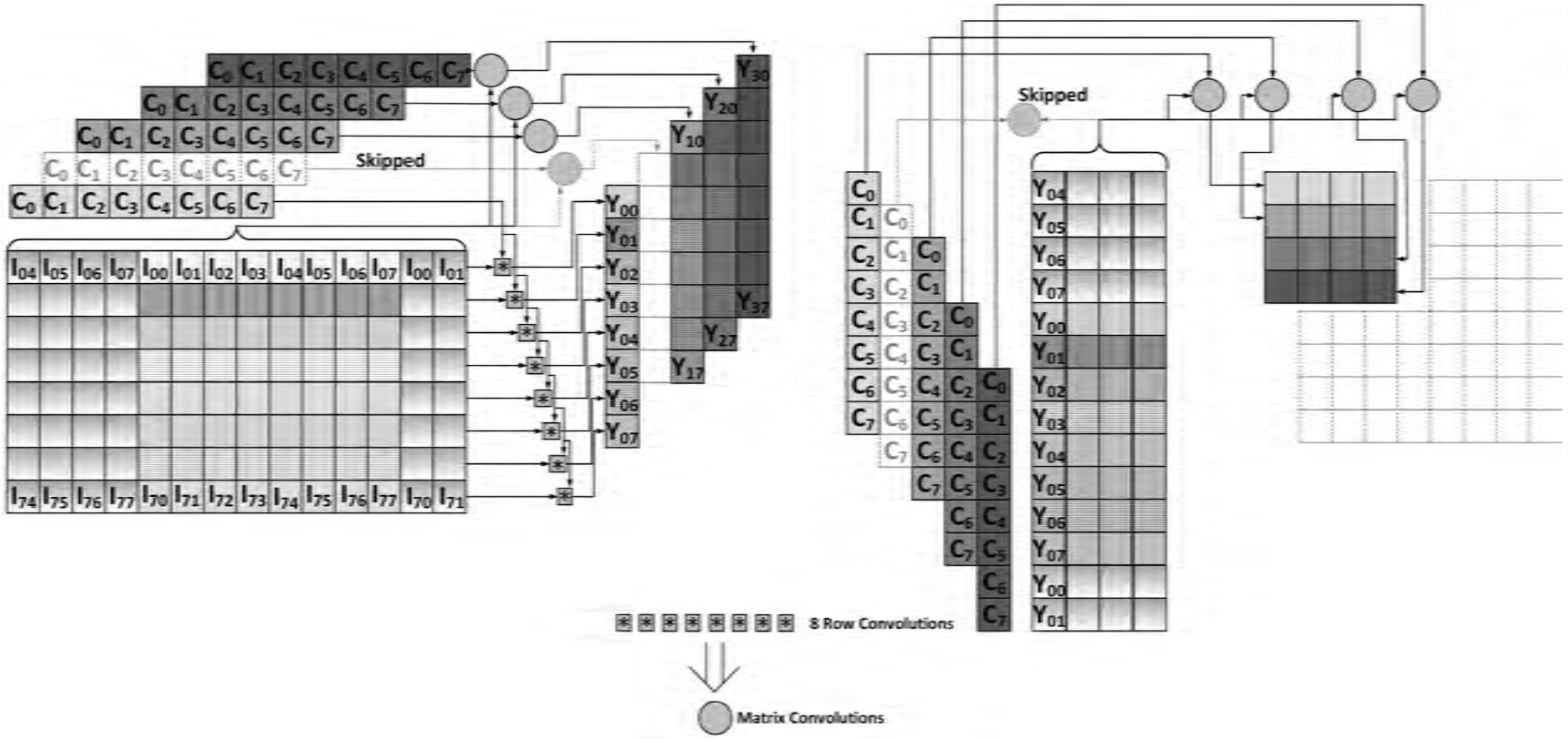


Figure 4.18: Pictorial representation of 2-D DWT algorithm.

Table 4.4: Filter coefficients of the targetted wavelets

Wavelet	C_8	C_7	C_6	C_5	C_4	C_3	C_2	C_1
Haar	0	0	0	0.7071	0.7071	0	0	0
bior3.1	0	0	-0.3536	1.0607	1.0607	-0.3536	0	0
bior2.2	0	0	-0.1768	-0.3536	1.0607	-0.3536	-0.1768	0
bior1.3	0	-0.0884	0.0884	0.7071	0.7071	0.0884	-0.0884	0
bior3.3	0.0663	-0.1989	-0.1547	0.9944	0.9944	-0.1547	-0.1989	0.0663
rbio1.1	0	0	0	0.7071	0.7071	0	0	0
rbio1.3	0	0	0	0.7071	0.7071	0	0	0
rbio1.5	0	0	0	0.7071	0.7071	0	0	0
rbio2.2	0	0	0	0.3536	0.7071	0.3536	0	0
rbio2.4	0	0	0	0.3536	0.7071	0.3536	0	0
rbio2.6	0	0	0	0.3536	0.7071	0.3536	0	0
rbio2.8	0	0	0	0.3536	0.7071	0.3536	0	0
rbio3.1	0	0	0.1768	0.5303	0.5303	0.1768	0	0
rbio3.3	0	0	0.1768	0.5303	0.5303	0.1768	0	0
rbio3.5	0	0	0.1768	0.5303	0.5303	0.1768	0	0
rbio3.7	0	0	0.1768	0.5303	0.5303	0.1768	0	0
rbio3.9	0.1768	0.5303	0.5303	0.1768	0	0	0	0
rbio4.4	-0.0645	-0.0407	0.4181	0.7885	0.4181	-0.0407	0.0645	0
fk4	0	0	-0.0462	0.0532	0.7533	0.6539	0	0
fk6	0	0.0406	-0.0772	-0.1464	0.3564	0.8129	0.4279	0
fk8	-0.0190	0.0426	0.0431	-0.1600	-0.0997	0.4753	0.7827	0.3492
coif1	0	-0.0157	-0.0727	0.3849	0.8526	0.3379	-0.727	0
db2	0	0	-0.1294	0.2241	0.8365	0.4830	0	0
db3	0	0.0352	-0.0854	-0.1350	0.4599	0.8069	0.3327	0
db4	-0.0106	0.0329	0.0308	0.1870	-0.0280	0.6309	0.7148	0.2304
sym2	0	0	-0.1294	0.2241	0.8365	0.4830	0	0
sym3	0	0.0352	-0.0854	-0.1350	0.4599	0.8069	0.3327	0
sym4	-0.0758	-0.0296	0.4976	0.8037	0.2979	-0.0992	-0.0126	0.0322

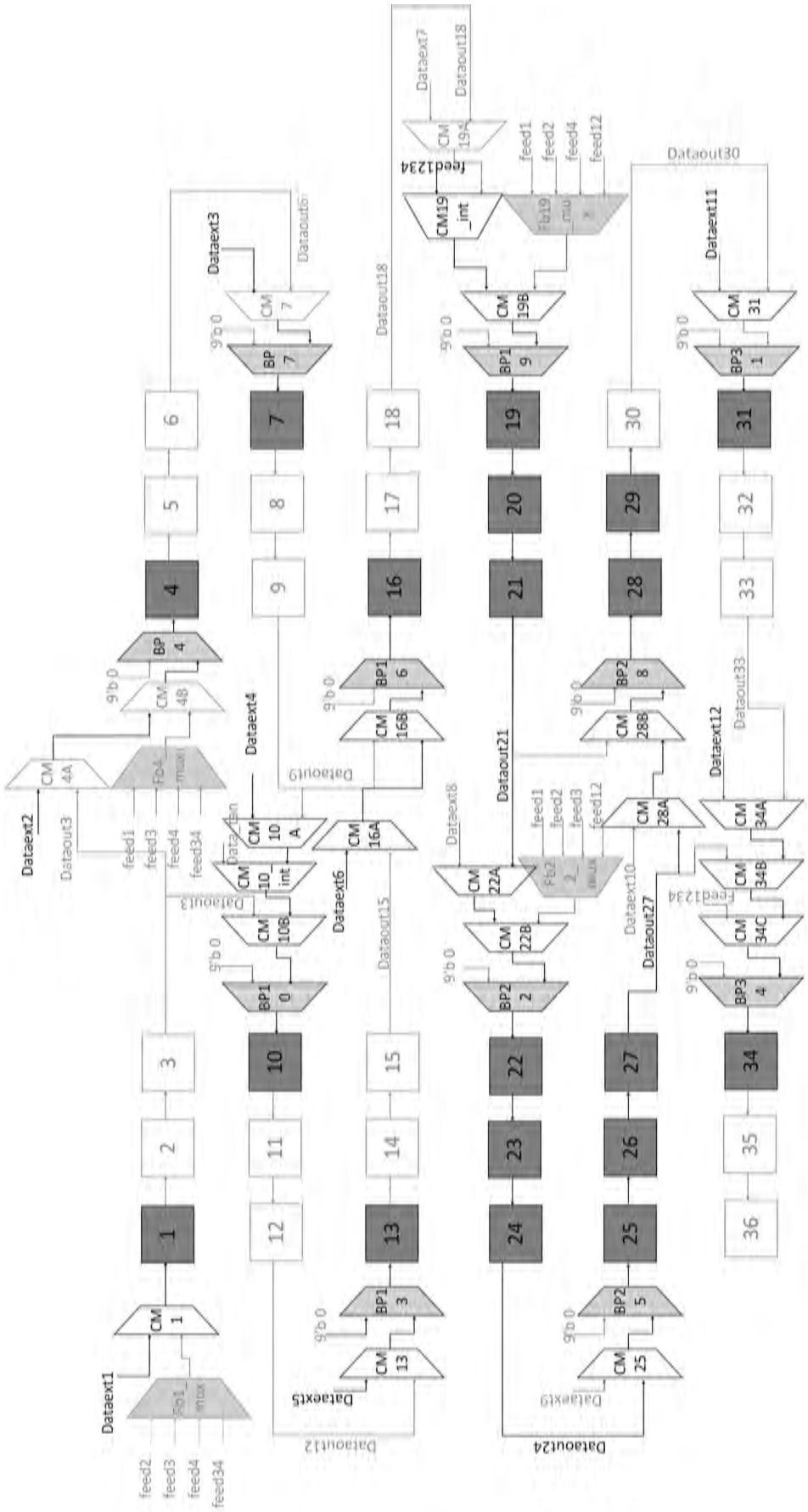


Figure 4.19: Active RUs and datapath for 2-D DWT mapping.

The architecture supports 8×8 block 2-D DWT operation. The RU triplets are left unconnected except for RU #19-30 chain. The image data is forced on RU #1, 4, 7, 10, 13, 16, 31 and 33 for Y matrix computations. The Y matrix is stored in RU #19-29 and filter mask is forced in the coefficients following the mask manipulations. The active RUs in the mapping scheme are indicated in Fig. 4.19. The following steps are performed while performing 2-DDWT on N×N image. The input data is considered as an image for convenience, compression being primary function of DWT.

1. Periodically pad the 8×8 block obtained from $N \times N$ image with four columns on left and two columns on right. Thus, the padded image (Pi) is of 8×14 dimension.

2. To perform row convolution over Pi, take 8×8 block of Pi and multiply each element of the first row of Pi_8 *i.e.* I_{51}, I_{61}, ...,I_{21} with the corresponding coefficient *i.e.* C_8, C_7, ...,C_1 and add all the products to form X_{11}. Now, move the mask over second row of Pi_8 again multiply and add to form X_{21}. Do this for all the 8 rows and thus we get the first column of intermediate matrix (X).

3. Repeat the step 3 for other three 8×8 blocks formed by taking alternate 8×8 block of Pi. We are taking alternate 8×8 block so to eliminate the need of downsampling thus reducing the number of computations. This will give us the intermediate matrix X.

4. To perform column convolution over X, periodically pad four rows on top and two rows on bottom of X giving Px matrix.

5. Now, take take 8×1 column vector of Px and multiply each element *i.e.* X_{51}, X_{61}, ...,X_{21} with the corresponding coefficient *i.e.* C_8, C_7, ...,C_1 and add all the products to form cA_{11}. Now, move the mask over alternate 8×1 column vector of Px again multiply and add to form cA_{21}. Do this for all the 4 8×1 column vector and thus we get the first column of DWT approximation matrix (cA).

6. Repeat the step 6 for other three columns of Px matrix. This will give us the 4×4 approximation matrix cA.

7. Repeat steps 1 - 7 for all other 8×8 blocks of N×N image. Thus, we will get the approximation matrix of dimension $N/2 \times N/2$.

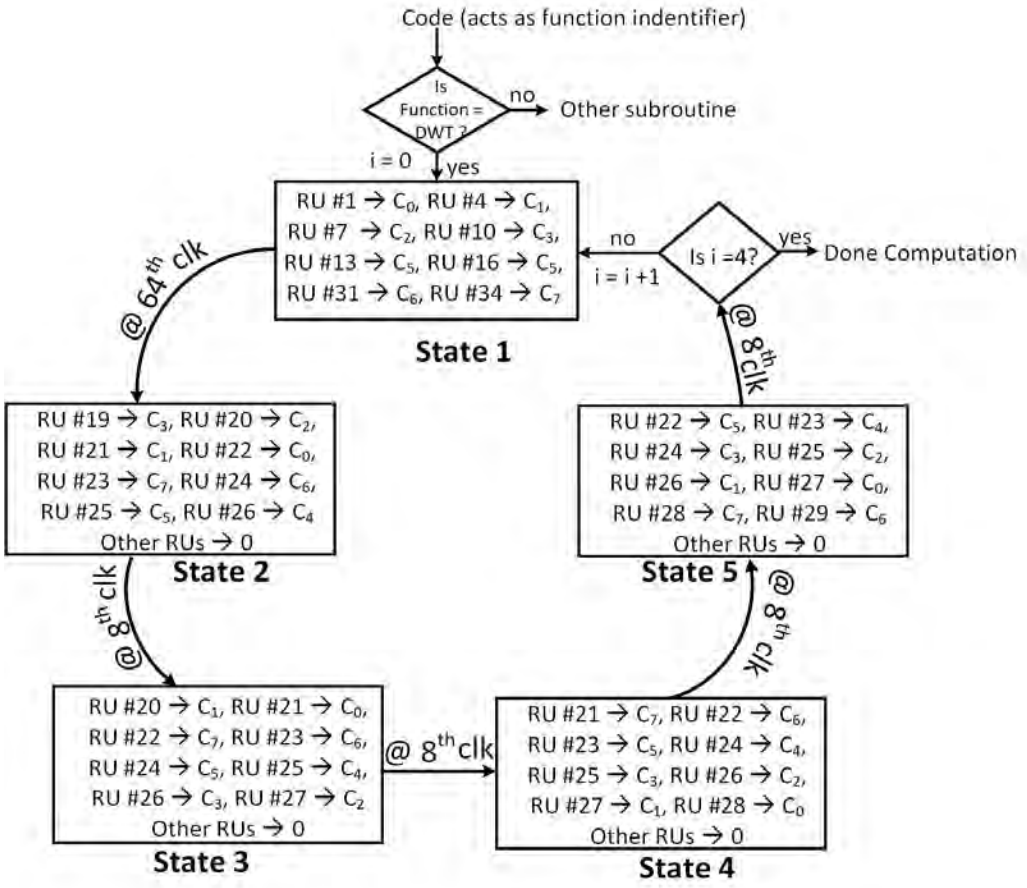


Figure 4.20: Flow chart of 2-D DWT state machine.

The state machine (shown in Fig. 4.20) controlling the DWT computations contains five states. The intermediate matrix (Y) rows are computed in first state and the other four states are visited to compute the first row of DWT matrix. The convolution filter coefficients provided by the state machine are forced on RU #1, 4, 7, 10, 13, 16, 31 and 33 and are multiplied with the first row of image matrix to generate the first element of Y matrix. Similarly, rest of the seven elements of first row of Y matrix are generated and the entire computation takes 64 cycles in total.

RU #19-26 holds the first row which must be padded on either sides according to eqns 4.20 and 4.21 as shown in Fig. 4.18(a). This is done by means of mask

Table 4.5: The active RUs, coefficients loaded and result computed in various states of 2-D DWT state machine.

State	Active RUs	Coefficients loaded	Value Computed
1	1, 4, 7, 10, 13, 16, 31 and 34	Ext1, 4, 7, 10, 13, 16, 31 and 34	Y (row 1)
2	19-26	Ext10, 7, 4, 1, 34, 31, 16 and 13	cA_{11}
3	20-27	Ext4, 1, 34, 31, 16, 13, 10 and 7	cA_{12}
4	21-28	Ext34, 31, 16, 13, 10, 7, 4 and 1	cA_{13}
5	22-29	Ext16, 13, 10, 7, 4, 1, 34 and 31	cA_{14}
after updating circular memory			
1	1, 4, 7, 10, 13, 16, 31 and 34	Ext1, 4, 7, 10, 13, 16, 31 and 34	Y (row 2)
2	19-26	Ext10, 7, 4, 1, 34, 31, 16 and 13	cA_{21}
3	20-27	Ext4, 1, 34, 31, 16, 13, 10 and 7	cA_{22}
4	21-28	Ext34, 31, 16, 13, 10, 7, 4 and 1	cA_{23}
5	22-29	Ext16, 13, 10, 7, 4, 1, 34 and 31	cA_{24}
after updating circular memory			
1	1, 4, 7, 10, 13, 16, 31 and 34	Ext1, 4, 7, 10, 13, 16, 31 and 34	Y (row 3)
2	19-26	Ext10, 7, 4, 1, 34, 31, 16 and 13	cA_{31}
3	20-27	Ext4, 1, 34, 31, 16, 13, 10 and 7	cA_{32}
4	21-28	Ext34, 31, 16, 13, 10, 7, 4 and 1	cA_{33}
5	22-29	Ext16, 13, 10, 7, 4, 1, 34 and 31	cA_{34}
after updating circular memory			
1	1, 4, 7, 10, 13, 16, 31 and 34	Ext1, 4, 7, 10, 13, 16, 31 and 34	Y (row 4)
2	19-26	Ext10, 7, 4, 1, 34, 31, 16 and 13	cA_{41}
3	20-27	Ext4, 1, 34, 31, 16, 13, 10 and 7	cA_{42}
4	21-28	Ext34, 31, 16, 13, 10, 7, 4 and 1	cA_{43}
5	22-29	Ext16, 13, 10, 7, 4, 1, 34 and 31	cA_{44}

manipulation explained here by means of an example. Let us try to compute cA_{11} *i.e.* first element of DWT matrix. According to Fig. 4.18(a), the first column of padded Y matrix when multiplied by the convolution filter yields cA_{11}. Padded Y matrix consist of padded elements concatenated on top and bottom of Y matrix column *i.e.* $Y_{04} - Y_{07}$ and $Y_{00} - Y_{01}$ on top and bottom, respectively. The process of padding posses a challenge in hardware implementations as it would translate into sending data to secondary memory for appropriate padding. Later, the padded data can be provided to the architecture after the writing and reading operation of circular memory. As a consequence, the computation latency increases dramatically. This is addressed by manipulating the convolution filter mask which substitutes for additional padding of Y matrix. The convolution filter is rearranged by moving the first four coefficients towards the end. By doing this,

the padded elements which are at the bottom of unpadded image gets multiplied with the respective coefficients yielding the same result as padded image with unmodified filter mask. This is illustrated in Fig. 4.21. The mask manipulation enables computing cA elements without external padding on the intermediate matrix which further permits storing the intermediate matrix within the SAC architecture.

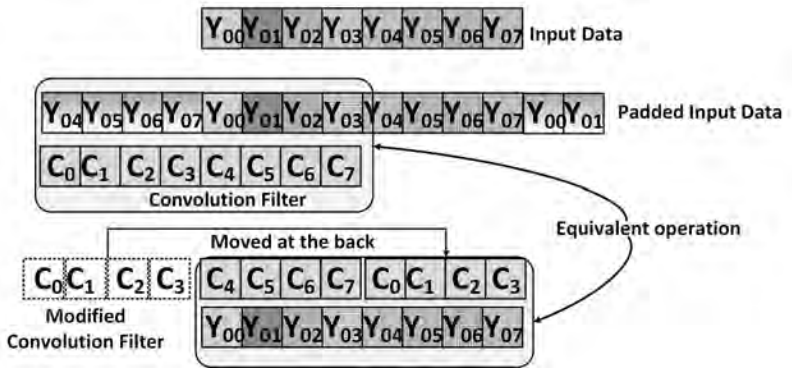


Figure 4.21: The DWT mask manipulation.

The data registers shift their values to next in line RU due to the nature of the developed SAC architecture. The coefficients loaded in states #2-5 follow this trend and are pushed down the RU chain accordingly as seen in Table 4.5. The second set of convolution on the Y matrix is carried out in these states with every state generating an element of the LL sub-band of DWT matrix in eight clock cycles each. The entire DWT matrix *i.e.* four rows of LL sub-band are generated by traversing the state machine four times. The elements generated in each state and the associated active RUs are shown in Table 4.5.

The circular memory holds the input data, image pixels in this case. The write pointer is used to write 8×8 (= 64) image data into memory which is stored column-wise in different sections of the memory in an interleaved manner. The interleaved manner of storing data is desirable as it simplifies the reading operation and ensures moving the convolution mask by 2 steps, a scheme adopted in the mapping to eliminate down sampling later. Consequently, multiple read ports ensure reading all columns simultaneously resulting in computation of Y matrix terms in least computation time the architecture supports *i.e.* eight clock cycles. The Fig. 4.22 shows image data stored in interleaved manner in the circu-

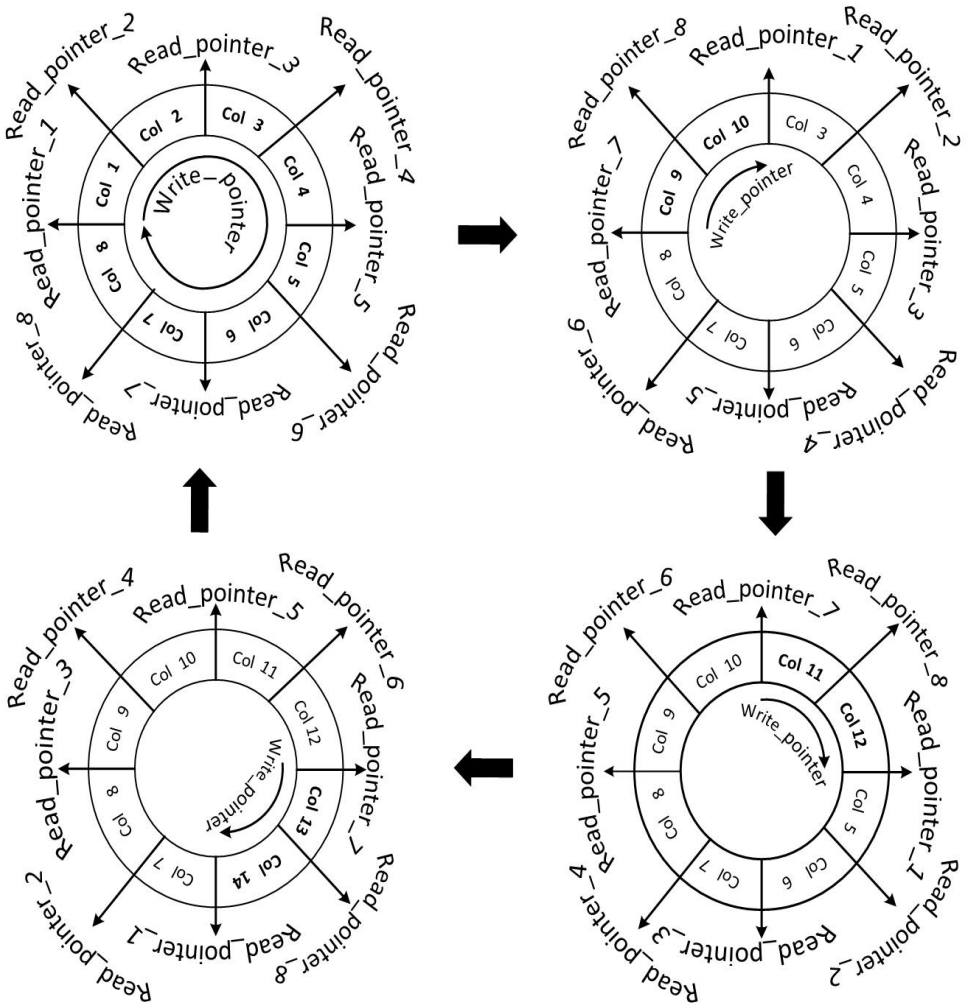


Figure 4.22: Data stored in interleaved manner in the circular memory with the write and read pointers

lar memory with the write and read pointers. Every time states #1-5 are traversed the circular memory is updated replacing two columns at a time as shown in Fig. 4.22.

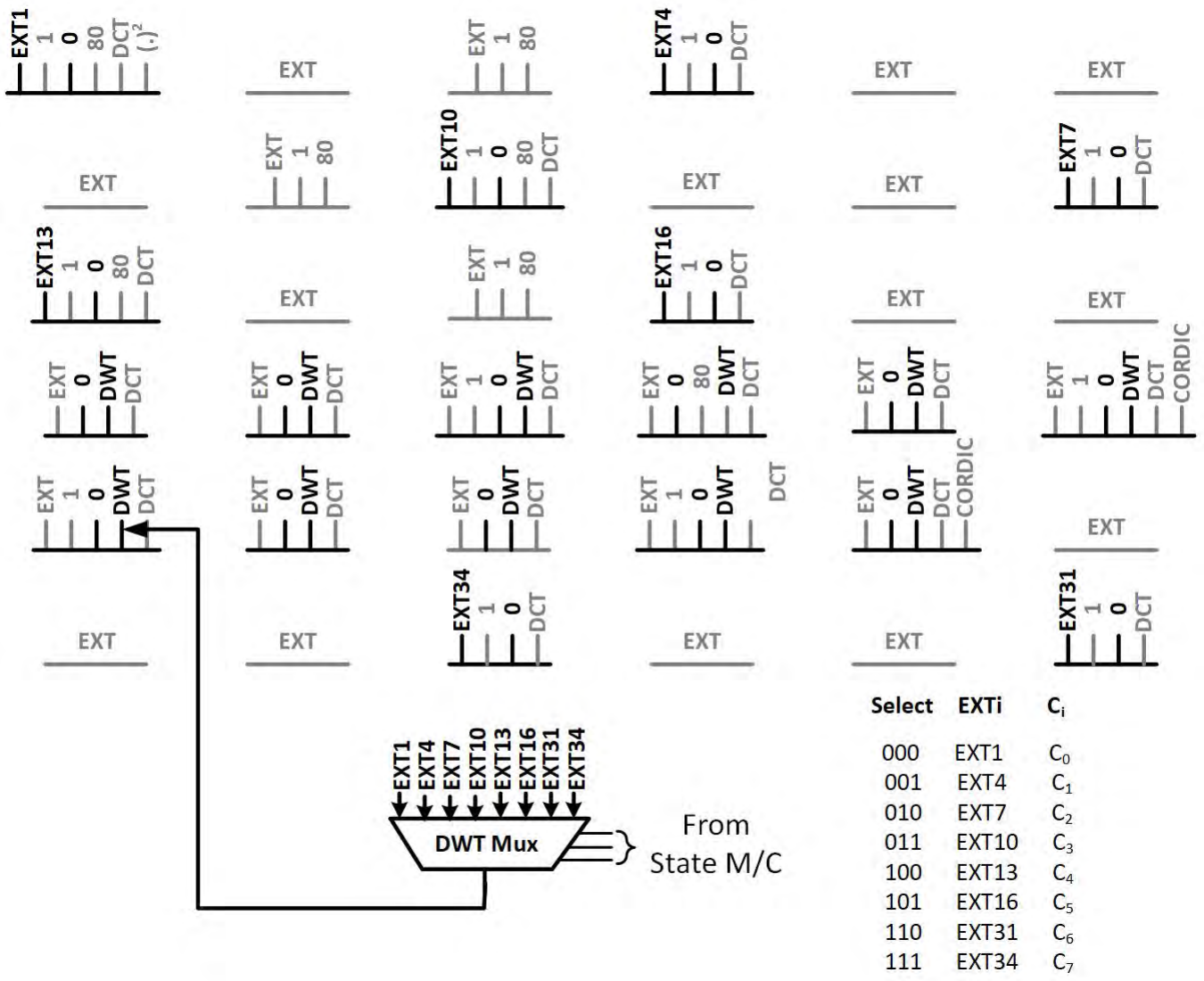


Figure 4.23: The coefficient multiplexers for 2-D DWT.

The DWT coefficients are applied to the RU coefficient registers through coefficient multiplexers. The DWT coefficient multiplexer shown in Fig. 4.23 is a 9-b 8:1 multiplexer which derives its select lines from the state machine. The multiplexer inputs are external coefficients obtained from the coefficient memory at the time of configuration. Connecting the external coefficients to multiplexer provide three benefits. Firstly, it supports multiple wavelets realization by loading external coefficients following Table 4.4, which would not have been possible if the multiplexer inputs were fixed. Secondly, it reduces the memory overhead arising out of storing varied coefficients of all target wavelets. Thirdly, it yields a cycle efficient realization as the coefficients are obtained from the coefficient memory as opposed to loading new coefficients externally which changes with wavelets as well as states. The DWT multiplexers are placed on RU #19-29 and are active during States #2-5. The state machine provide the select for desired coefficient. The multiplexer selects and selected coefficient are shown in Fig. 4.23.

The mapping methodology modifies slightly when 1-D DWT is implemented. The low-pass and high-pass filters can be mapped individually in different RU sections. The filter coefficients are applied on the RU coefficient registers in an interleaved manner ensuring the dydadic sampling and elimination of redundant computations.

4.2.3 Discrete Cosine Transform

The discrete cosine transform was first proposed in 1974 by N. Ahmed et al. for the purpose of pattern recognition and Wiener filtering in the area of image processing [157]. The algorithm was found comparatively optimal, in terms of emulating the Karhunen-Loève transform, as compared to other orthogonal transforms, such as discrete fourier transform (DFT), Walsh-Hadamard transform (WT) and Haar transform (HT). The widely used image compression application of DCT was pioneered by Chen and Pratt [158] after almost a decade of its discovery. Since then, the DCT has escalated to be the central mathematical operation for numerous standards of image and video coding including JPEG [159], MPEG-1 [160], MPEG-2 [161], H.261 [162], H.263 [163], H.264 [164], and the recent

HEVC [37,38]. These coding standards are ubiquitous in DSP applications and are also found in a multitude of biomedical applications related to image and video compression [165–168]. In the scenario of ever increasing medical imaging data volume (35.5 TB/day in 2010) [169], storage and transfer, the use of optimized methods like approximate DCT is advocated in [170] over exact computations to reduce the computation complexity. DCT aided ECG signal compression is presented in [171, 172]. Techniques like approximating higher order coefficients by their average value, dynamic thresholding and variable quantization are adopted to maximize compression for varying data patterns in real time and ambulatory settings.

The definition of 2-D DCT for an input image 'A' and output image 'B' is defined as follows:

$$B_{pq} = \alpha_p \, \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \, \cos\left(\frac{\pi \, (2m+1) \, p}{2M}\right) \, \cos\left(\frac{\pi \, (2n+1) \, q}{2N}\right), \quad 0 \leq p \leq M-1; 0 \leq q \leq N-1 \tag{4.22}$$

where,

$$\alpha_p = \frac{1}{\sqrt{M}}, \; p = 0$$
$$= \sqrt{\frac{2}{M}}, \; 1 \leq p \leq M-1$$

$$\alpha_q = \frac{1}{\sqrt{N}}, \; q = 0$$
$$= \sqrt{\frac{2}{N}}, \; 1 \leq q \leq N-1$$

'M' and 'N' are the row and column size of 'A', respectively.

The DCT transform equation (5.1) can be expressed as-

$$B_{pq} = \alpha_p \, \alpha_q \sum_{m=0}^{M-1} \cos(\frac{\pi \, (2m+1) \, p}{2M}) \sum_{n=0}^{N-1} A_{mn} \, \cos(\frac{\pi \, (2n+1) \, q}{2N}), \quad 0 \leq p \leq M-1; 0 \leq q \leq N-1 \tag{4.23}$$

This is possible because 2-D DCT follows the separability property. Thus, according to eqn 4.23, B_{pq} can be computed in two steps by successive 1-D operations on rows and columns of an image. Thus, 2-D DCT can be decomposed into two 1-D DCT as shown in Fig. 4.24.

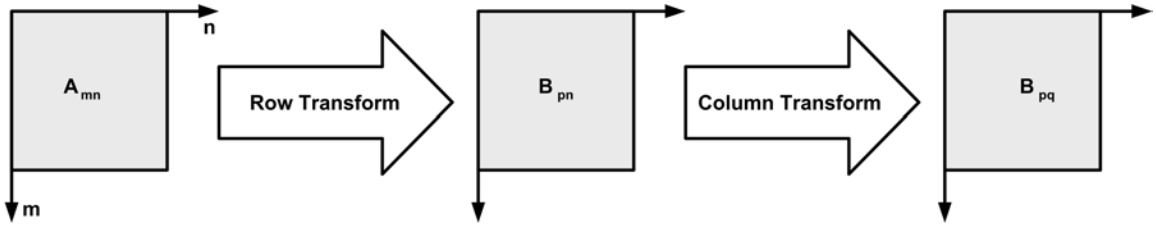


Figure 4.24: Computation of 2-D DCT using separability property

The 2-D DCT matrix computation of an image requires two matrix multiplication operations : $DCT = DCT_{mat} * Image * (DCT_{mat})^T$. The DCT transform matrix (DCT_{mat}), are cosine of angles determined by the DCT mask size ($M \times M$) and contains orthogonal basis vectors given by $C(m,n) = l_m \cos(\pi m(2n+1)/2M)$ where, l_m is a constant determined by M. the DCT transform matrix DCT_{mat} of size $M \times M$ is given as-

$$(DCT_{mat})_{xy} = \frac{1}{\sqrt{M}}, \; x = 0, \; 0 \leq y \leq M-1 \tag{4.24}$$

$$= \sqrt{\frac{2}{M}} \cos(\frac{\pi \, (2y+1) \, x}{2M}), \quad 1 \leq x \leq M-1, 0 \leq y \leq M-1 \tag{4.25}$$

The coefficients of DCT_{mat} matrix obtained by the above equation for M = 8 is-

$$\begin{bmatrix}
0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\
0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\
0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\
0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\
0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\
0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\
0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\
0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975
\end{bmatrix}$$

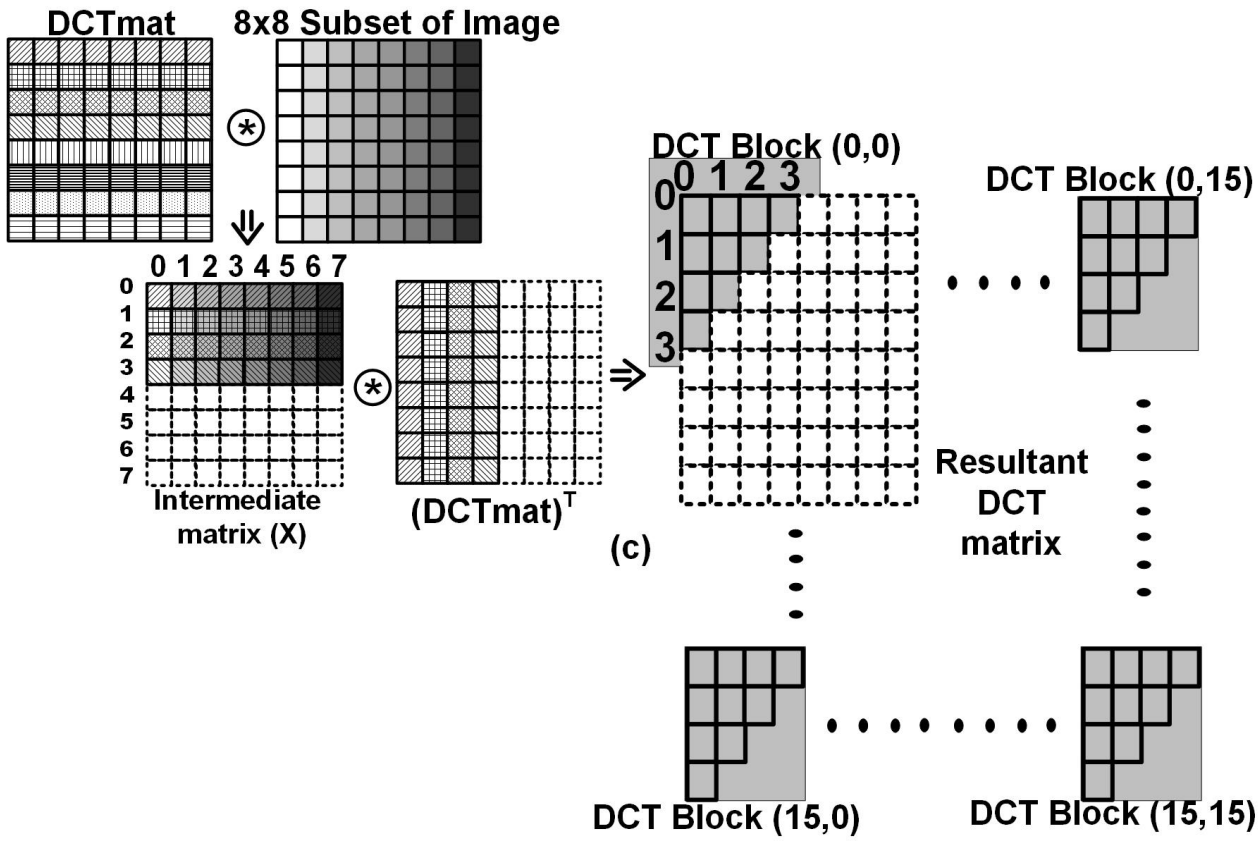
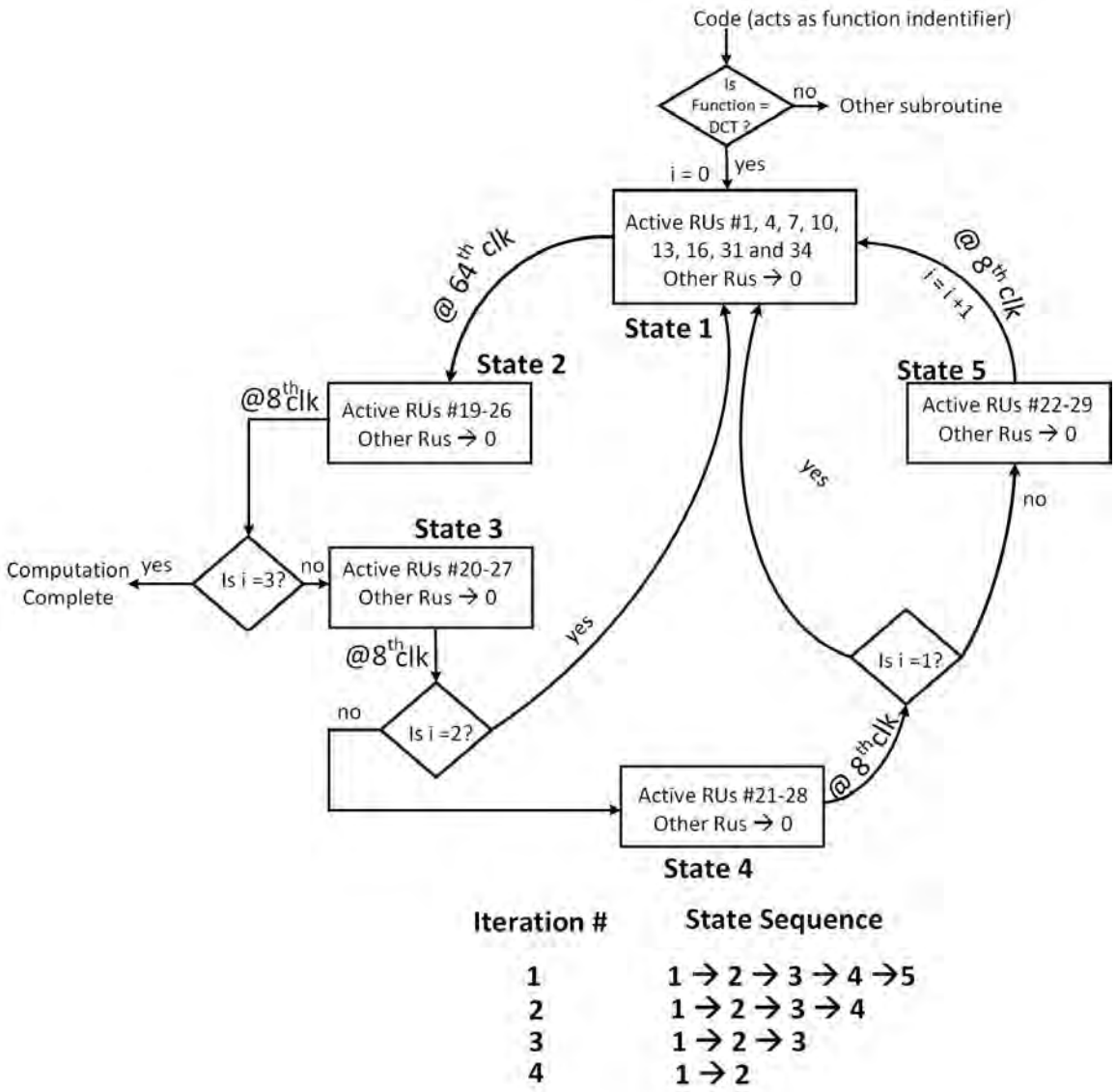


Figure 4.25: The 2-D DCT algorithm steps.

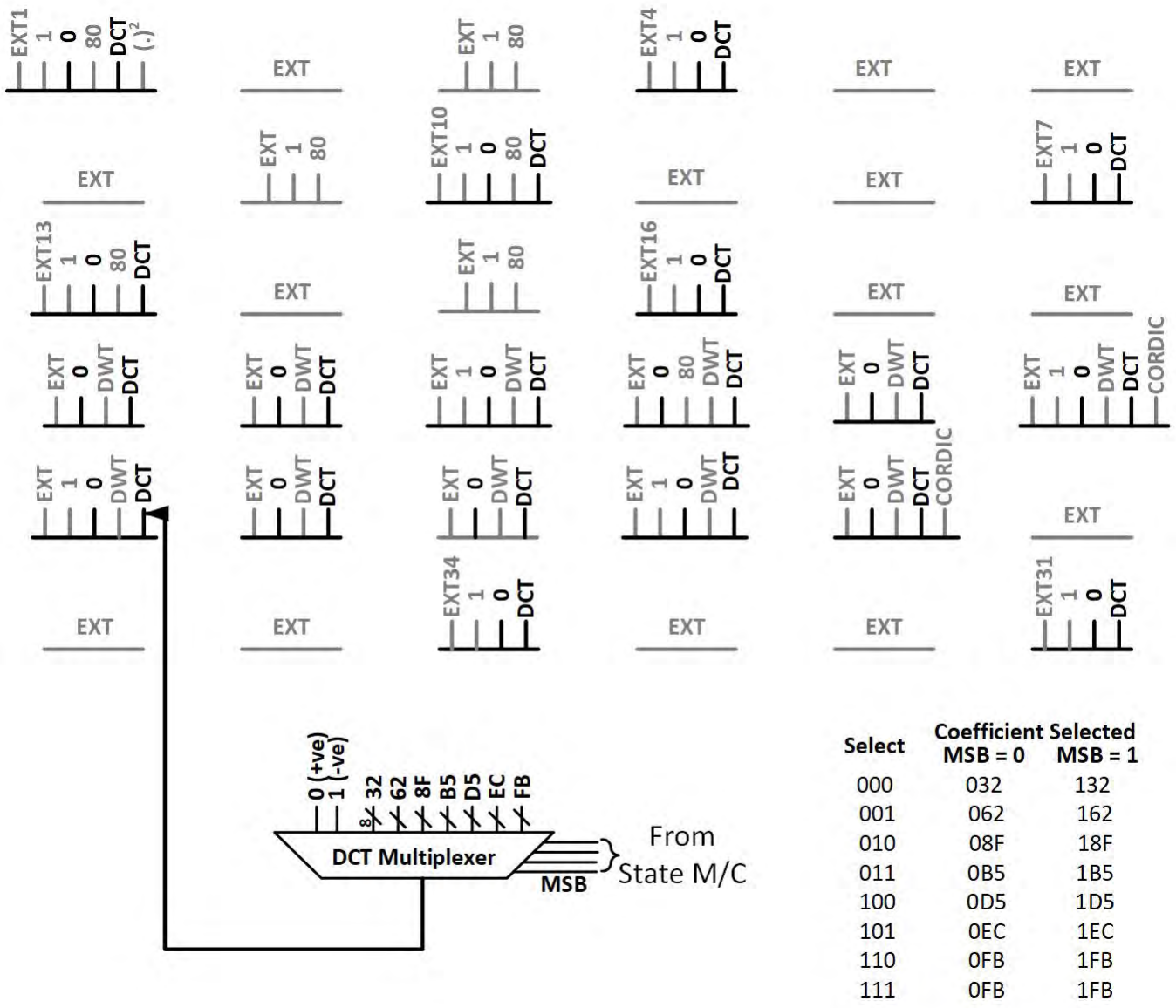
Mapping Methodology of DCT

Matrix multiplication can be realized on the SAC architecture by loading row and column elements of the two matrices in place of coefficients and data registers respectively. This methodology is adopted for DCT computations. The 2-D DCT algorithmic steps are shown in Fig. 4.25. The architecture supports 8×8 block wise DCT and provides a compression of $\approx$ 84%. This requires calculating 10 of 64 outputs in the DCT matrix. The mapping methodology for DCT exhibits remarkable resemblance to 2-D DWT mapping and hence is discussed here in brief. RU #19-30 are connected in a chain whereas rest of the triplets are not connected with each other resulting in an active RU picture similar to Fig. 4.19. The first matrix multiplication or the intermediate matrix *i.e.* $Int_{mat} = DCT_{mat} * Image$, row wise image data and first row of DCT_{mat} are forced on RU #1, 4, 7, 10, 13, 16, 31 and 33 as data and coefficient, respectively.



Iteration #	State Sequence
1	1 → 2 → 3 → 4 → 5
2	1 → 2 → 3 → 4
3	1 → 2 → 3
4	1 → 2

Figure 4.26: Flow chart of 2-D DCT state machine.



Select	Coefficient Selected MSB = 0	MSB = 1
000	032	132
001	062	162
010	08F	18F
011	0B5	1B5
100	0D5	1D5
101	0EC	1EC
110	0FB	1FB
111	0FB	1FB

Figure 4.27: The coefficient multiplexer for 2-D DCT mapping.

Table 4.6: DCT multiplexer coefficients in decimal and hexadecimal.

DCT Multiplexer Coefficients	
Decimal	Hexadecimal
0.0975	32
0.1913	62
0.2778	8F
0.3536	B5
0.4157	D5
0.4619	EC
0.4904	FB

The results forms first row of Int_{mat} that is periodically stored from RU #19-26. This has to be further multiplied with DCT_{mat}^{T} or rows of DCT_{mat}. The row coefficients of transform matrix is forced on coefficient registers of RU #19-26 and the resultant $DCT(0,0)$ is obtained. On forcing subsequent DCT_{mat} rows in the coefficient of RU #19-30 chain, the remaining elements of first row of DCT are computed. It must be noted that, Int_{mat} stored in RU #19-26 experiences shifting down the RU chain every 8 clock cycles and the state machine is designed incorporating this shift. The state machine consist of 5 states traversed four times, fully or partially, that computes Int_{mat} and DCT in successive computations. The state machine and its sequence is shown in Fig. 4.26. The state sequence differs with the iteration# because the number of elements computed in the DCT matrix decreases by one with every row resulting in a upper triangular matrix. The Int_{mat} matrix is computed in state #1 and DCT matrix is computed in states #2-5. The image data is stored column wise in an interleaved manner in circular memory similar to data storage in DWT mapping discussed earlier. The coefficients for 2-D DCT are applied through DCT multiplexers attached to RU #1, 4, 7, 10, 13, 16, 19-29, 31 and 34 (shown in Fig. 4.27). The DCT multiplexer is designed as a composite structure of 8-b 7:1 multiplexer and a 1-b 2:1 multiplexer. This was possible because the 8×8 DCT coefficient matrix consists of seven distinct entries (highlighted in DCT_{mat} matrix) in positive and negative forms. Thus it was beneficial to apply the numeric value and sign of the coefficients separately yielding gate savings. The 7:1 multiplexer provides the former whereas the sign is assigned by the 2:1 multiplexer. The distinct DCT matrix coefficients and equivalent hexadecimal are

tabulated in Table 4.6.

4.3 Conclusion

The mapping methodologies for DSP functions frequently encountered in biomedical signal processing on the SAC architecture are presented in this chapter. The data and coefficient registers of the architecture are interpreted perspicaciously maximizing the number of functions mapped on the architecture while keeping the configuration overheads in check. The heart of the configuration lies in the multiplexers placed in the datapath which gives rise to efficient usage of architecture resources by offering multiple connection topologies and isolating unused blocks. The light-weight data memory is tightly coupled with the architecture and exhibits a circular architecture. It has the ability to adapt according to data requirements of different functions mapped by means of varying read pointers. The circular memory architecture provides an added advantage in case of 2-D DWT and 2-D DCT wherein the data storage technique results into simplistic data fetching. In few cases, especially while realizing fixed coefficient functions, the coefficient multiplexers are employed to provide the coefficients by means of a state machine controlling the function execution. The intermediate data is stored within the architecture providing easy access to this data for subsequent operations. In general, the developed mapping methodologies discussed in this chapter uses methods to optimize latency and architecture utilization at multiple stages.

CHAPTER 5

System Verification on an FPGA

The FPGA (Field Programmable Gate Array) implementation of SAC architecture along with its associated memories is discussed in this chapter. The configurable datapath of the SAC architecture is set by the 54-b control word that contains *resolution, code, config* and *feedback* fields. The control word is discussed in perspective of different topologies as well as target functions. The configuration scheme of the architecture consists of different phases wherein the control word, coefficients and data are supplied to the architecture followed by computation and storage of results. The VGA (Video Graphics Array) and UART (Universal Asynchronous Receiver Transmitter) interfaces are developed on the FPGA development board for visual perception and offline data processing, respectively. The hardware computed results are compared with their software (MATLAB) counterparts for various target functions. Later in the chapter, the clock profile for target functions is presented along with gate count of the SAC architecture and interfacing memories.

5.1 Number System

The SAC architecture works on 9-b fixed point arithmetic and uses the sign-magnitude convention to represent signed numbers. This convention is preferred because multiplication being the primary operation in SAC architecture does not yield same results as two numbers multiplied together when their 2's complement equivalent numbers are multiplied. For instance, -2$\times$-4 in 2's complement artihmetic would be "1110"$\times$"1100" = "10101000" $\neq$ 8. While interpreting the 9-b number,

MSB denotes the sign and remaining 8 bits represent the magnitude.

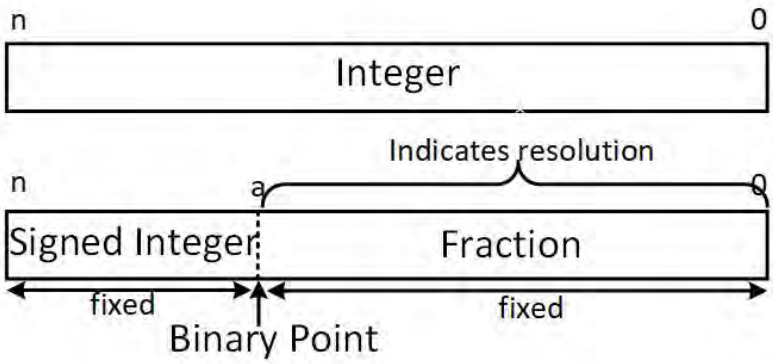


Figure 5.1: The notional representation of binary point in fixed-point number system.

The range of the numbers following this convention is +511 to -511. However, the resolution of these numbers is programmable which results in multiplying the said range with a fixed scaling factor. This can also be interpreted as repositioning the binary point in the number resulting in a notional split as shown in Fig. 5.1. The scaling factor in binary is 2 raised to the number of bits to the right of the binary point. While interpreting a binary number, the binary point could be moved in three ways listed below. For the following discussion, i denotes the number of bit positions the binary point has moved.

- **Moved to right:** This case represents that the actual number is greater than the corresponding stored binary number and additional zeros are appended if the binary point is moved beyond the LSB. The scale factor in this case is 2^i. Such interpretation may be of use when the higher order bits indicate significant information.

- **Remained at its position:** This case indicates that the actual number is the same as its binary representation and scale factor is unity.

- **Moved to left:** This case indicates that the actual number is smaller than the represented binary number and scale factor is 2^{-i}. This interpretation is beneficial when the input resolution is of significance *i.e.* input data range is small and minute changes in data translates into large variation in output.

5.2 Control Word for Various functions and Topologies

The control word contains four field, namely *resolution, code, config* and *feedback*, which give rise to various architecture topologies. Collectively these fields administer the datapath, data input through circular memory, generated outputs of the SAC architecture in ways discussed next. The *code* and *config* field for targeted functions are presented in Table 5.1. The *code* acts as target function identifier and controls the data read write operations of circular memory. This includes amount of data written into and data reading methods (single or multiple read pointers) of circular memory. The *config* field is primarily responsible for configuring the architecture datapath using configuration and bypass multiplexers. There are a few cases where two functions share the same *config* whereas have different codes. This is because these functions have different coefficient selections in the coefficient multiplexers which is based on *code* field. For example, multiplication and square have same *config* but external data is forced in coefficient in former whereas data and coefficient both are the same in latter case. The *feedback* word depends on the function requirements *i.e.* feedback exists or not. In case a feedback exist in a function, for example an IIR (Infinite Impulse Response) filter, the output can be fed back to the architecture data register by forcing appropriate data in the *feedback* field. For instance, with *FB4* ="00", the Tile #1 output is fed to RU #4 (tile #2) data register, resulting in a configuration involving tiles #1 and 2 collectively producing the final result. One can also feed the collective output of two tiles (*feed12* and *feed34*) as feedback.

The bits 4-0 of the control word are connected to the 2:1 multiplexer in the RU. It is preferable to forward unregistered input (*fbi* = '0') to RU when feedback data is involved as registering the feedback value consumes a clock cycle and upsets the computation synchronization. For instance, going back to the IIR example spread over Tile #1 and 2, output ($y[0]$) is generated in eight clock cycles (8^{th} clock) and is forwarded to RU #4 data register. By the time $y[0]$ is registered *i.e.* 9^{th} clock, the next output ($y[1]$) computation has already begun and coeffi-

Table 5.1: Control word for target functions and topologies.

Function	Code	Config
Multiplication	01	00000000
Square	02	00000000
Addition/MAC	03/07(3-tap)	00D00000
	04/08(6-tap)	0AD60000
	05/09(9-tap)	0AD640D
	06/0A(12-tap)	0AD66AB7
FIR, $\frac{d}{dx}$, MA	0B(9-tap)	00700000
	0C(18-tap)	2FFE0000
	0D(27-tap)	2FFF4070
	0E(36-tap)	2FFF6FFF
DWT	0F	0AB66FD7, fb19='1'
2-D Convolution	10	00D00000
CORDIC	11	00D04040, fb19='1'
DCT	12	0AB66FD7, fb19='1'
Topology	**Config**	**Feedback**
Four Tile	0E724E73	–
Systolic Tiles		
Tile #1 output fed to Tile#2 (or #1–>#2)	1E720000	0008
#1–>#2–>#3	1E72C070	008C
#1–>#2–>#3–>#4	1E72DE73	00CE
RU#1-18–>#3–>#4	2FFEDE73	01C6

cient in Tile #1 RUs are rotated by one place. The partial products accumulated at this stage is $PP7_{Tile\#1}$ and does not include partial product generated from RU #4. The partial products from RU #4 starts generating from the 10^{th} clock. The partial product accumulation in this clock is $(PP6_{Tile\#1} + PP7_{RU\#4})$, which results in erroneous computation of all the subsequent outputs after $y[0]$. Alternatively, if unregistered output is propagated through, the partial product accumulation term at 9^{th} clock becomes $PP7_{Tile\#1} + PP7_{RU\#4}$ and thus accounts for the feedback output. The feedback field value for few topologies is listed in bottom section of Table 5.1. The first entry depicts a configuration where the output from Tile #1 is fed as input to Tile #2 or RU #4. *FB4* is set to"00" selecting *feed1* in feedback multiplexer *Fb4_mux* and all *fbi* but *fb4* is set.

Resolution is of prime importance in functions with feedback and ensures significant bits are involved in computation especially in binary system. For instance,

1.100000000 10-b binary output with one bit in the integer part and 9 bits fractional part representing a resolution of 2^{-9}. If the feedback bits are not configurable but fixed to, lets say, 7^{th}-0^{th} bits where 0^{th} bit represent the LSB, zero would be fed back instead of the significant bits of the output resulting in erroneous output calculations. This problem is addressed by making the resolution configurable by means of selecting the resolution bits in the control word. These bits determines the output slice fed back to architecture. Resolution bits when set to "0000", feeds back the 7^{th}-0^{th} bit section *i.e.* decimal equivalent of resolution bits decides the lower bound of feedback section. For the 10-b output example chosen earlier, the setting the resolution bits to "0010" would result in feeding back 11000000 output slice and retains the significant section of output for computations. For a 22-b output, there exists fifteen 8-b sections. These sections are tabulated in Table 5.2.

Table 5.2: The resolution bits are respective output sections selected for feedback.

Resolution Bits	Output section fedback
0000	7^{th} - 0^{th}
0001	8^{th} - 1^{st}
0010	9^{th} - 2^{nd}
0011	10^{th} - 3^{rd}
0100	11^{th} - 4^{th}
0101	12^{th} - 5^{th}
0110	13^{th} - 6^{th}
0111	14^{th} - 7^{th}
1000	15^{th} - 8^{th}
1001	16^{th} - 9^{th}
1010	17^{th} - 10^{th}
1011	18^{th} - 11^{th}
1100	19^{th} - 12^{th}
1101	20^{th} - 13^{th}
1110	21^{th} - 14^{th}

5.3 Configuration of the Architecture

The SAC architecture is developed in VHSIC (Very High Speed Integrated Circuit) Hardware Description Language, commonly known as VHDL, and tested on ModelSim for functional integrity. We now present the various function con-

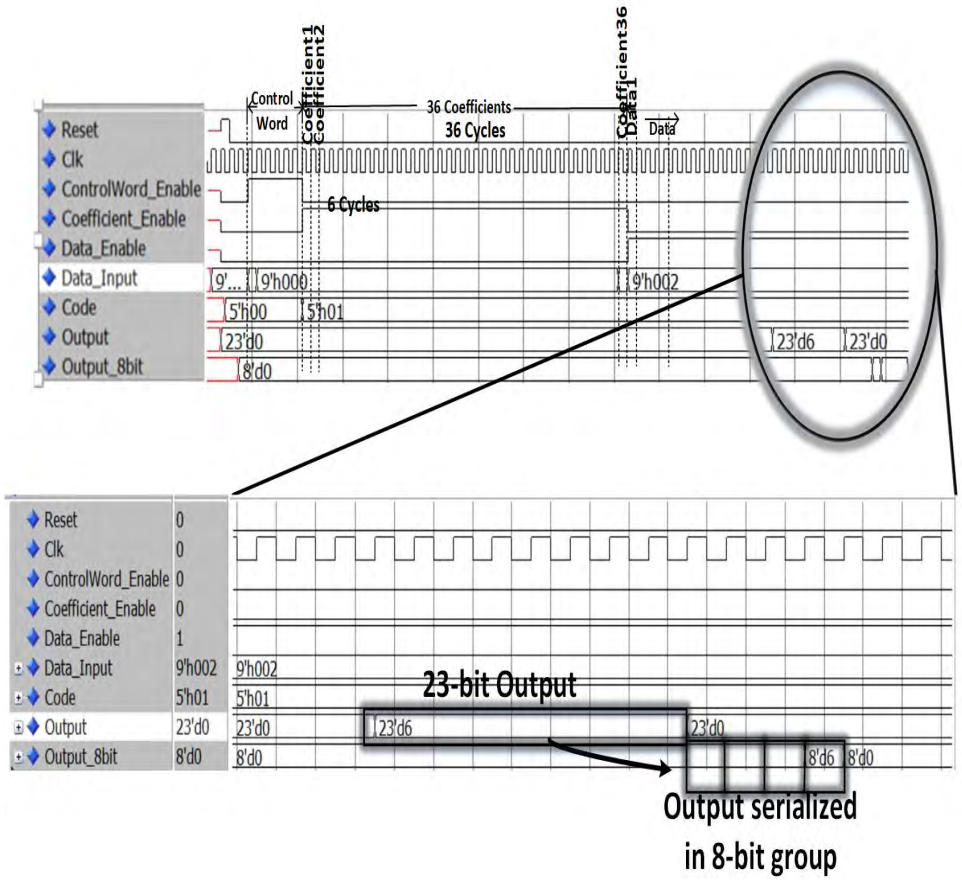


Figure 5.2: Architecture input bus. The zoomed section denotes the multiplication results and 8-b serialized output.

figuration details. The architecture is said to be configured once the control word, external coefficients and data are loaded in their respective memories. These values are provided to the architecture through a 9-b input bus in a time multiplexed manner. The 54-b control word is applied on the 9-b input bus in chunks starting with the LSB chunk *i.e.* 8^{th} - 0^{th} bits. Every time a new chunk arrives the previous chunk is shifted down the control word register. Following this process, the control word takes 6 cycles for loading. The external coefficients are loaded next and comprise of 36 9-b binary numbers. The coefficient loading follows the shifting down technique employed for control word. They are loaded into the coefficient memory in 36 clock cycles starting with coefficient #1 which shifts down to its own memory location once all the 36 coefficients are loaded. Following this, input bus carries the data designated for the 9-b wide 64 position deep data memory. However, the amount of data and consequently the clock cycles to load it, is highly

dependent on input data requirements of the target functions. The estimates of data loading clock cycles are presented in a later section (Sec. 5.10) in this chapter.

The interpretation of input bus contents as control word, external coefficient or data requires proper delineation. This distinction is achieved by means of control signals generated using state machine controlling the function execution on the architecture. The control signals comprise of three enable signals, namely, $ControlWord\_Enable, Coefficient\_Enable$ and $Data\_Enable$, shown in Fig. 5.2. The $ControlWord\_Enable$ signal is held 'high' for the first six clock cycles after reset and control word is forced on the input bus in a sliced chunk manner. The $Coefficient\_Enable$ signal is held 'high' for the next thirty-six clock cycles treating the input bus as carrying external coefficient. Following this, the $Data\_Enable$ signal is made 'high' indicating input bus holds the data. The generated control signals are mutually exclusive and no two control signals go 'high' at the same time. As an illustration (see Fig. 5.2), let the architecture be configured as multiplier with *code* = 01H, multiplicand = 02H and multiplier = 03H. The multiplier is loaded as data and multiplicand as coefficient #1. The output bus in holds the output = 06H also seen in the zoomed section of the figure. This 23-b output is divided into four groups of 8-b starting from LSB for interfacing with the conventional 8-b output ports and limiting the number of I/Os. The architecture proceeds to the computation phase after configuration.

5.4 Simulation Results

5.4.1 FIR Filtering

The FIR filtering operation is functionally verified on the architecture using the Low Pass Filtering (LPF) computation of the Pan-Tompkins Algorithm (PTA) [2] signal chain. In the biomedical signal processing domain, PTA is a widely used algorithm for QRS detection in Electrocardiogram (ECG) signal. The PTA signal chain, shown in Fig. 5.3, is a combination of linear and non-linear functions consisting of noise filtering, differentiation, square and moving average operations. The LPF of PTA is mathematically expressed as $y[n] = x[n] - 2 \cdot x[n-6] + x[n-$

$12] + 2 \cdot y[n-1] - y[n-2]$ represents an IIR (Infinite Impulse Response) filtering operation consisting of three input and two output samples in present and delayed forms.

The input samples are spread across a window of size 13, hence the 18 RUs connected in a systolic array is suitable. The 18 RU chain can be realized by connecting Tiles #1 and 2 together and RU# 1, 7 and 13 are loaded with coefficients 1, -2 and 1, respectively. The output feedback samples can be applied to RU #34 as they are spread across a window of size 2 making RU # 34 and 35 as active RUs. The coefficient of other RUs is maintained at 0. Thus, the PTA LPF can be mapped on the SAC architecture in Tiles #1, 2 and 4.

The standard MIT/BIH (Beth Israel Hospital) arrhythmia database [55, 56] is used as input. It encompasses the ECG signal plotted in Fig. 5.4(a) after normalization. The hardware results (thin line) are plotted in Fig. 5.4(b) along with the MATLAB results (thick line). It can be seen that minor variations are flattened in hardware results due to limited resolution in output feedback. On the other hand, the hardware and MATLAB results exhibit good coherence at higher signal range in positive as well as negative excursions.

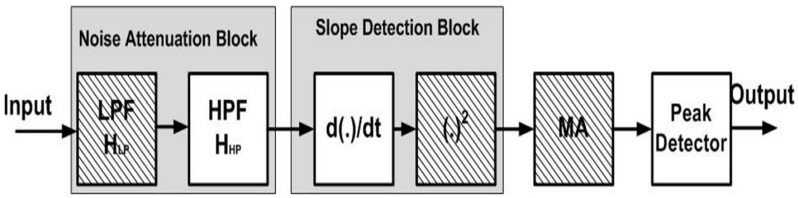


Figure 5.3: The Pan-Tompkins Algorithm signal chain.

Table 5.3: Range and Resolution of CORDIC on developed Architecture

Functions	**H/W Input Range**	**HDL resolution**	
		Output	**Input**
sin, cos	-90° to $+90^\circ$	2^{-7}	2^{-1}
sinh , cosh, exp	-1.118 to +1.118	2^{-7}	2^{-6}
ln	0.1068 to 9.360	2^{-6}	2^{-4}

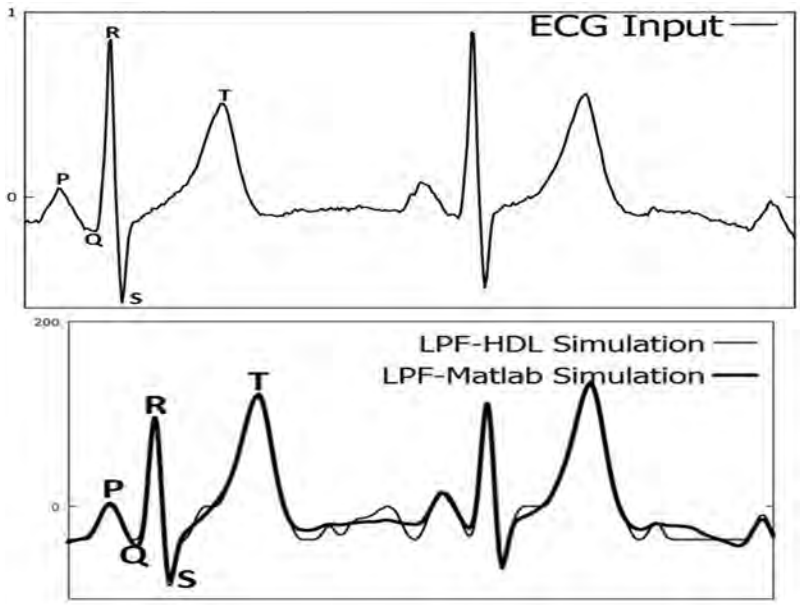


Figure 5.4: Hardware and MATLAB results for LPF in PTA.

5.4.2 CORDIC Algorithm

The SAC architecture supports radix-2 CORDIC in its conventional range and resolution mentioned in Table 5.3. The position of the binary point is indicated by the resolution. For instance, binary point between the 1^{st} and 0^{th} bit represent resolution 2^{-1} . Additionally, input resolution denotes the precision of input provided to the architecture and gains importance for functions yielding large change in output with small change in input (Ex: exponential function). The target range for circular trigonometric functions covers the entire spectrum of input whereas hyperbolic trigonometric, exponential and natural logarithmic functions range is meagre due to convergence issues. The range for hyperbolic functions can be increased by repeating certain iterations [173]. However, any mechanism to expand the CORDIC range is not attempted in this thesis.

The hardware simulation results are shown in Fig. 5.5. The signal names filled with gray are internal to the architecture and provide CORDIC mode of operation information *i.e.* rotation or vectoring and circular, linear or hyperbolic. After eight iterations, the variables y_8, z_8 and x_8, takes values 03F, 003 and 06A having 0.4921875, 1.5° and 0.829125 as their decimal equivalents while computing sin30° and cos30° using CORDIC in rotation mode. The results obtained from hardware

simulation agrees well with the ideal (MATLAB) results for functions sin(cos); sinh(cosh); e^x and ln(x) and are shown over their respective conventional range in Fig. 5.6(a)-(d). The architecture computes cosine with accuracy ranging from 98%-83%. The resolution is 2^{-1}(= 1 bit) when the binary point is between 1^{st} and 0^{th} bit position. Thus, the smallest degree with which the variable can change is 0.5° while the remaining 7 bits denotes range of the input data. The range of % relative accuracy of functions is provided in Table 5.4. The Cosine function suffers a decrease in accuracy in small values of cosine which augments the error.

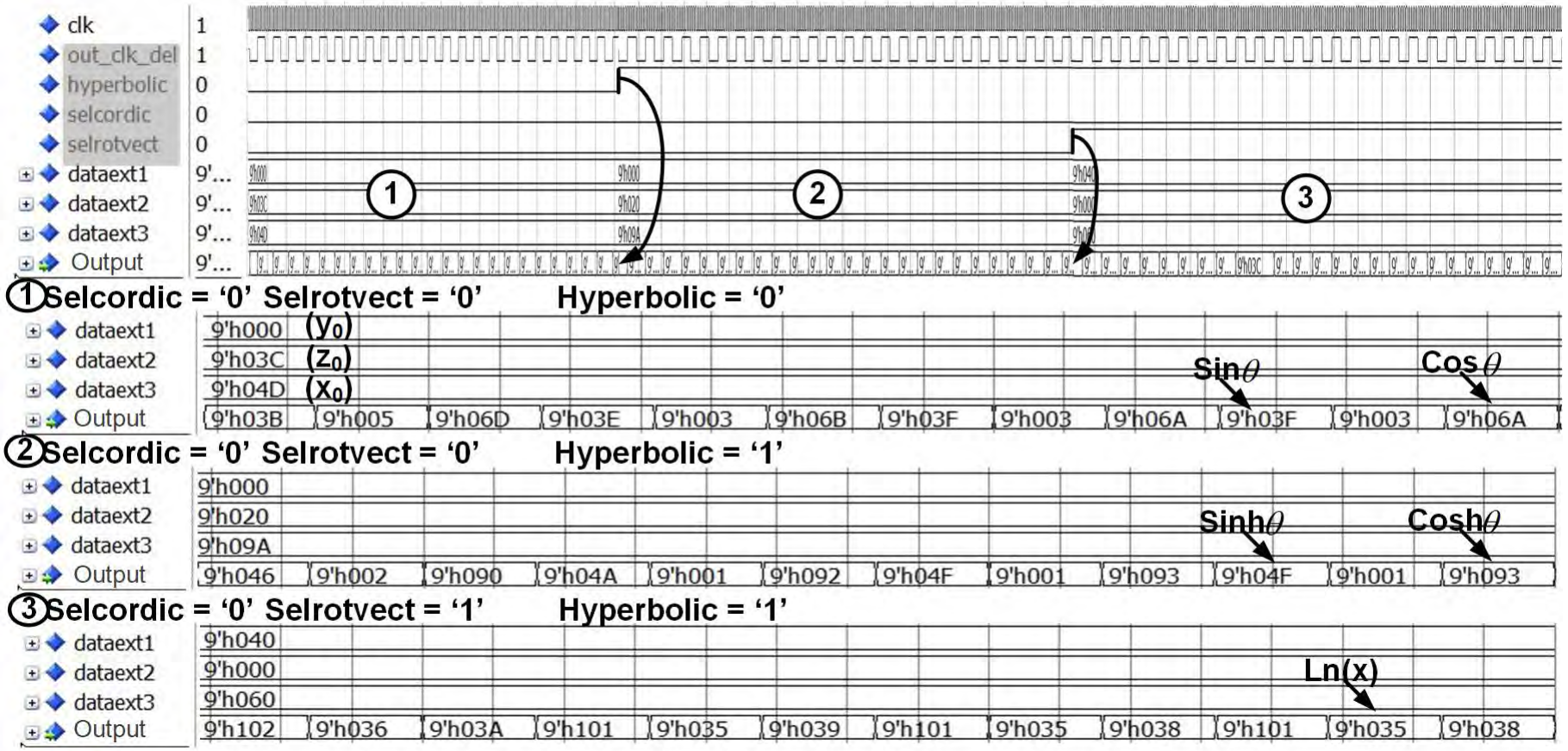


Figure 5.5: Hardware simulation results of CORDIC in (1) Rotation non-hyperbolic computing sine and cosine (2) Rotation Hyperbolic computing hyperbolic sine and cosine and (3) Vectoring Hyperbolic modes computing Ln(x).

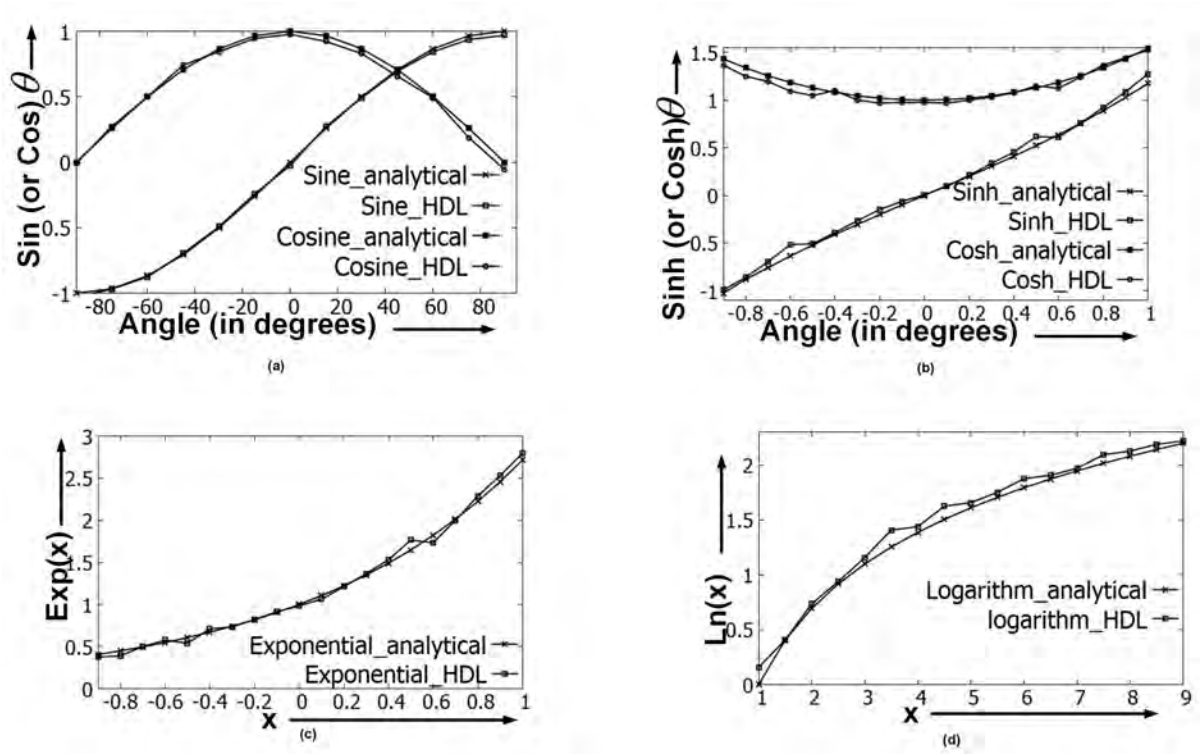


Figure 5.6: The hardware and MATLAB results for (a) Sine and Cosine (b) Hyperbolic Sine and Cosine (c) exponential (d) natural logarithm functions.

Table 5.4: % relative accuracy for CORDIC functions.

Function	Relative Accuracy Range (in %)
Sine	99.8-93.6
Cosine	98.4-72.5
Ln	99.23-87.8

5.4.3 Discrete Cosine Transform (2-D DCT)

The results for 2-D DCT operation are presented in terms of image compression, its most common application. The fundamental basis for certain design choices is discussed first. The computation complexity of multiplications and additions in matrix multiplication are both order of $O(n^3)$, where n is the dimension of the matrix. Therefore, computing the 2-D DCT of an image becomes challenging task for hardware implementation as dimension of image increases. In such cases, using the block wise DCT operation is preferred, wherein the image is partitioned into small blocks and then DCT is performed on these blocks or sub-images indepen-

dently. In addition to reducing the computational burden, storing only necessary blocks of image instead of the entire image reduces the memory requirements.

Three parameters are observed while selecting the size of DCT matrix which are computational complexity, compression achieved, and the error. In this case, the error is defined by the difference between the test image and the reconstructed image obtained by performing DCT and IDCT operation, correspondingly on the test image. Thus, the DCT of a standard Lena intensity image of size 128×128 is analyzed by varying both the mask size as well as the coefficients to be retained after DCT transform in MATLAB. To quantify the error between original image and the reconstructed image, Mean Square Error (MSE) is used, which is given by eqn. 5.1.

$$\text{MSE} = \frac{1}{128 \times 128} \sum_{x=0}^{127} \sum_{y=0}^{127} [Img_{actual}(x,y) - Img_{reconstructed}(x,y)]^2 \tag{5.1}$$

Table 5.5: Comparison of various mask size with respect to computation, memory and MSE metrics.

Mask	∼ 50% Compression	∼2.9e-3% MSE
Size	<Computation, Mem,MSE>	<Computation, Mem, Compression>
2×2	<9.8e4, 8kB, 1e-3>	<3.6e4, 4kB, 75% >
4×4	< 2.2e5, 8kB, 3e-4>	< 6.7e4, 3kB, 81%>
8×8	<4.9e5, 8kB, 1e-3>	<1.3e5, 2.5kB, 84%>
16×16	<1.01e6, 8kB, 7.4e-4>	<2.3e5, 2.3kB, 86%>

It can be observed from the Table 5.5 that for a particular compression, the number of computations reduces with smaller masks. However, with smaller mask size, reduced number of DCT coefficients are retained which affects the quality of the image and is reflected in the table as degrading MSE. The 8 block size is adopted in this thesis because of its wide acceptability [174] based on a balanced trade-off between the number of computation and error encountered. Furthermore, the compression ratio of an image, defined as the number of frequency components retained over the total number of such components, needs to be optimal along with the amount of computations and error. For block size of 8×8, the number of computations and the MSE converge at an optimal point of

10 (shown in Fig. 5.7). This leads to retaining 10 coefficients out of 64 coefficients for a standard 8×8 mask which results to a compression ratio of 84.38%.

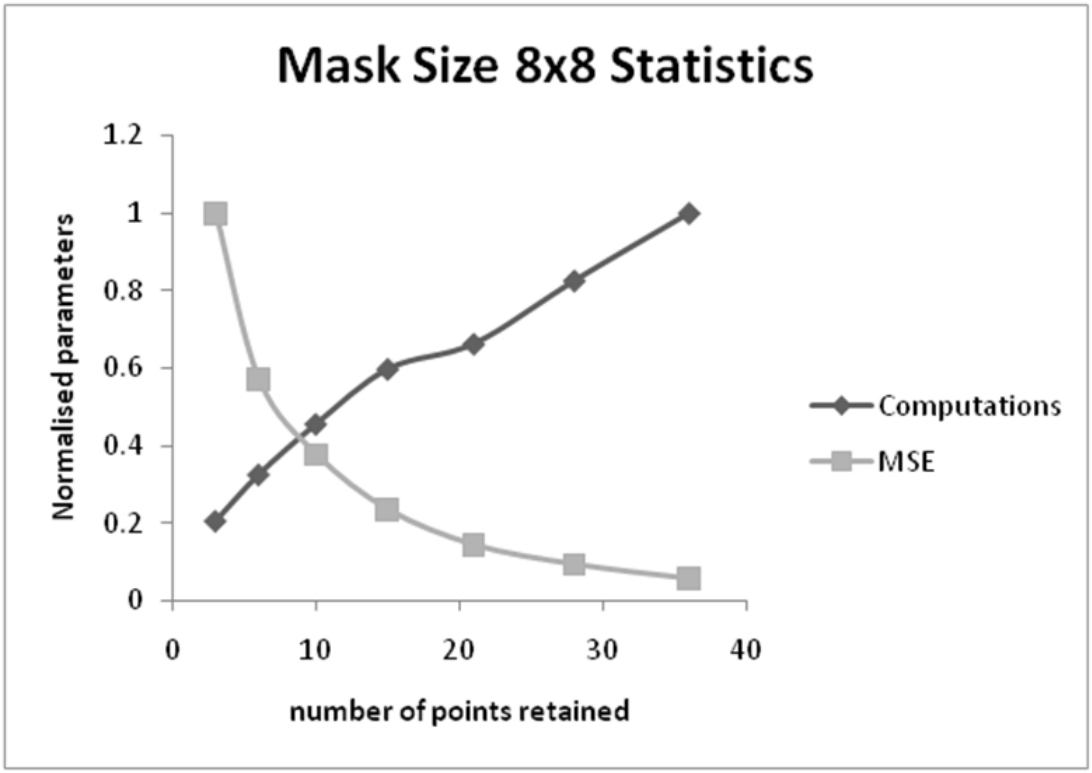


Figure 5.7: Convergence point for DCT matrix of size 8x8.

The test setup for DCT is shown in Fig. 5.8(a). The 8×8 image sub-blocks undergo 2-D DCT operation on the SAC architecture and results are fed to MATLAB which reconstructs the image by performing offline block wise 2-D IDCT. The reconstructed image obtained from hardware simulations and MATLAB (Fig. 5.8(b) and (c)) is compared with the original image using L2 norm, that is reported as 15.77 and 15.75, respectively.

5.5 Hardware Results

The architecture is ported on Virtex-II pro FPGA development board. A brief discussion on the various components of the FPGA development board is presented in section 5.6. The operating frequency of FPGA implementation is found to be 46.9 M*Hz*. The system level interactions of various modules of the architecture is shown in Fig. 5.9. Apart from the SAC architecture, the system has multiple mem-

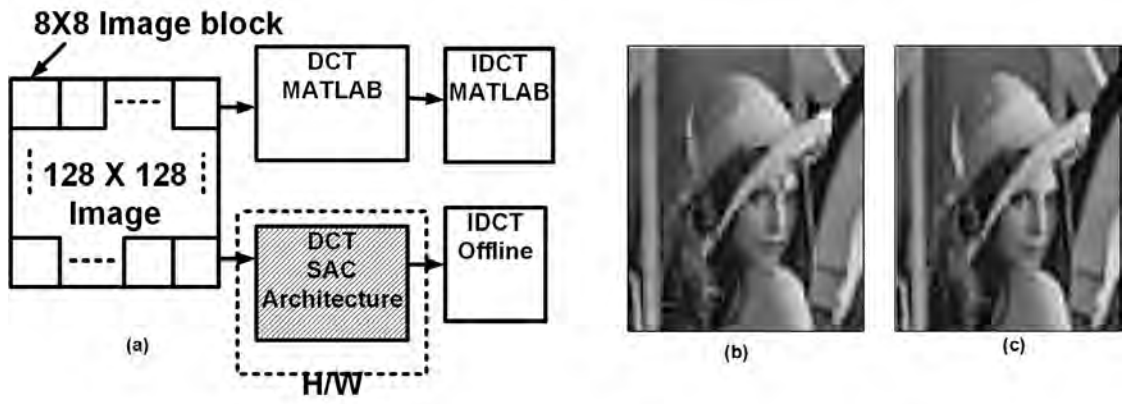


Figure 5.8: DCT (a) Hardware setup block diagram (b) Hardware computed (c) MATLAB computed results.

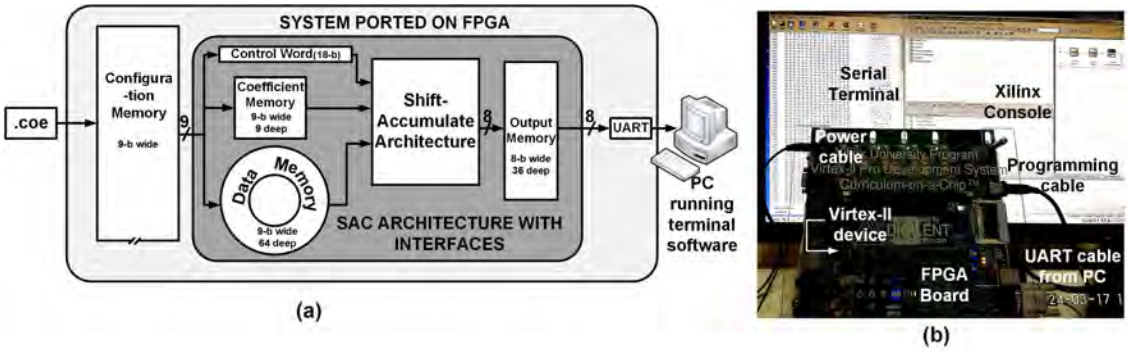


Figure 5.9: (a)System level interaction of SAC Architecture with peripherals (b) Actual hardware setup.

ory modules for configuration, control word, coefficient, data and output. Inbuilt IP core generator is used for configuration memory and it provides control word, coefficients and data to the architecture that are stored in their respective memories. The configuration memory is initialized using a coefficient file (.coe) that can be generated using MATLAB for large data. A UART serial port (bit rate = 115200 bps) is interfaced with the architecture to migrate generated output to computers for analysis and necessary post-processing. For this, the output is temporarily stored in output memory which when full signals the start of UART transmission. A Video Graphics Array (VGA) interface is developed for visual perception of data, images in particular. The design and development of interfaces is discussed in detail in section 5.7.

5.6 Overview of Field Programmable Gate Array (FPGA) Board

In chip design, logic errors need to be eliminated early in the design to avoid costly hardware re-spins. Thus, prior to the Application Specific Integrated Circuit (ASIC) implementation of the platform, the design requires to be verified against the design failures that may happen due to erroneous human behavior. The functionality of the design can be verified in simulation environment but it does not guarantee its hardware counterpart. Hence, it is preferred to opt for hardware emulation. FPGA chips present a viable technology for emulating complex algorithms, leveraging advances in performance, capacity, and software support for FPGAs. The FPGA development Board speed up the verification process by providing suitable interfaces to the FPGA chip, which help in verification of the design. The FPGA board used in the thesis is Xilinx University Program Virtex-II Pro Development Board [175] (shown in Fig. 5.10), which has interface VGA and RS-232 ports among many other peripheral interfaces. It has an on-board clock oscillator of 100 M*Hz* and a provision of external clock input. The FPGA chip embedded in the board is Virtex-II Pro "XC2VP30", which has 30,816 logic cells, distributed RAM of 428 Kb, block RAM of 2448 Kb, 8 Digital Clock Manager (DCM), 644 user I/O pads, 2 hardcore PowerPC RISC cores, *etc.*

5.7 Design of Peripheral Interface Controllers

Interfaces are vital in any system and are of importance in the developed SAC architecture for bringing the generated output out of the FPGA environment. This data can be further sent to a display unit or a computer that validates the accuracy of results and/or performs necessary post processing steps. This section presents the two interfaces employed in the thesis with its design in VHDL, hardware emulation on the FPGA followed by the results obtained. . Their protocol and function of the interface controller are provided in Appendix A.

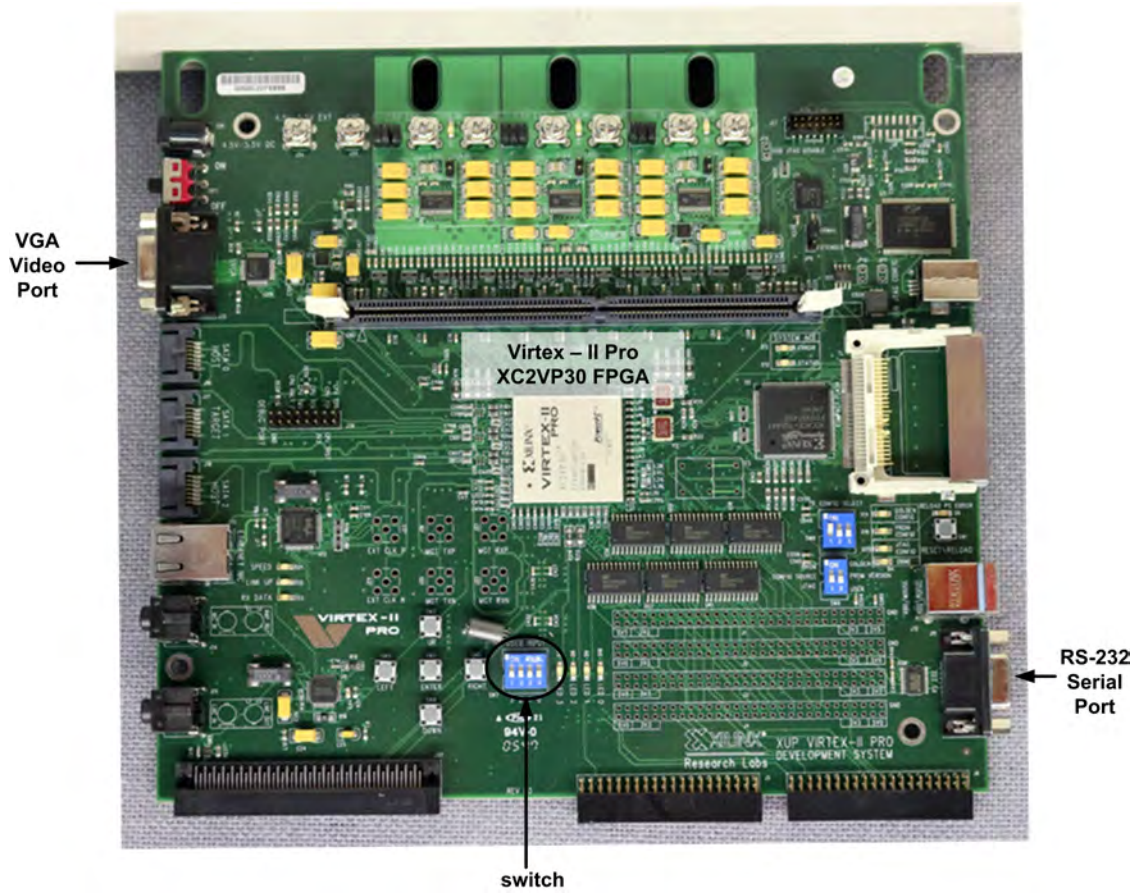


Figure 5.10: XUP Virtex-II Pro Development System Board Photo

Design of VGA Controller

A controller is designed which drives these 5 pins of the port based on the timing parameters tabulated above. In order to ensure proper functioning of the designed controller, a simple demonstration application is chosen. The application requires flipping of an image along its two principal axes depending on the position of two switch (as shown in Fig. 5.10) present on the FPGA board and display result on a display unit.

The controller module (shown in Fig. 5.11) is divided into different sub module units based on the functionality called VGA timing unit, Pixel clock generation unit, RGB video unit, Transformation unit and the Memory unit. The design methodology of these modules is discussed in the following subsections.

- **Pixel Clock Generation Unit:** The clocks available on the board run at 100 M*Hz*, whereas the required pixel clock frequency is 25 M*Hz* as specified

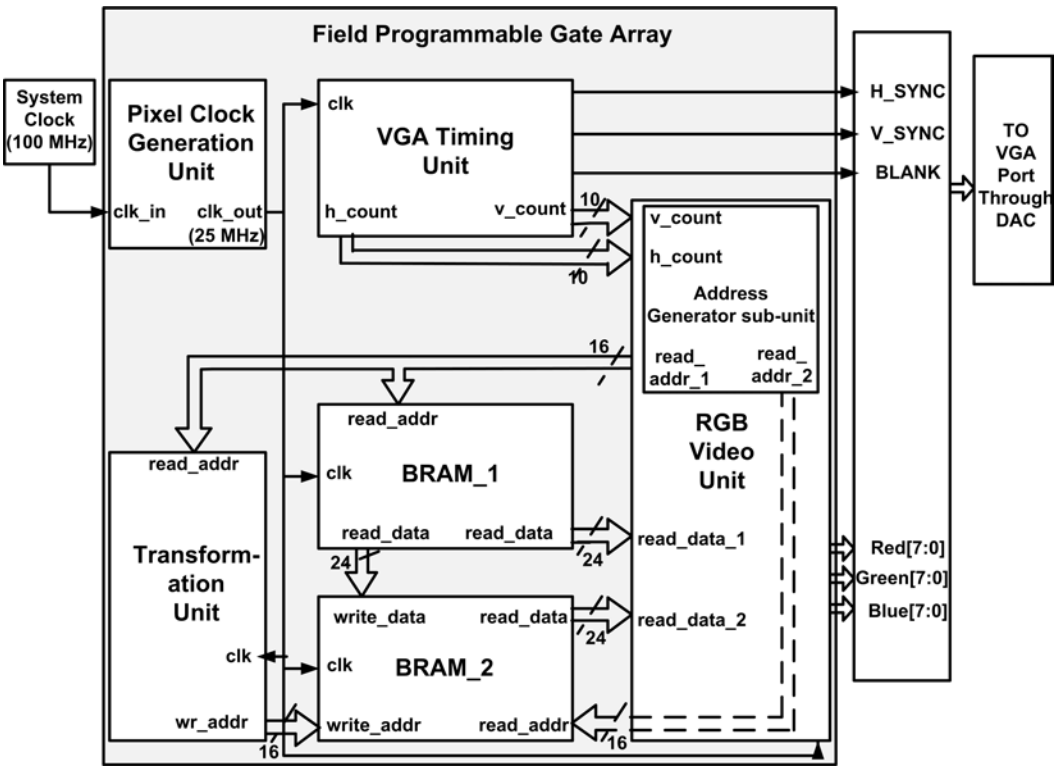


Figure 5.11: Block Diagram of Image Processing Module

by the resolution standard. The FPGA has Digital Clock Manager (DCM) unit as a primitive to derive clock of different frequencies and phases from the master clock of 100 M*Hz*. The advantage of DCM is that uses global buffers (BUFG) which have outputs that are specifically designed to tolerate the load of a large fan-out which prevents clock skew.

- **Memory Unit:** The actual image is stored in the Block RAM (BRAM) of the FPGA. Due to the size constraint of the BRAM of the board, which is 2448 Kb, the actual image of resolution "240×160×24" is chosen, which occupies 900 Kb (240×160×24) of BRAM. The resolution "240×160×24" specifies 240 rows and 160 columns of pixels, where each pixel is represented by 24 bits (8 bits for each of the three color component). Hence, the FPGA can hold two image files, the actual and its transformed version in BRAM. Two memory modules, BRAM_1 and BRAM_2 which will be storing the two images are generated, each having dimension of 38400 locations corresponding to the 38400 (240×160) pixels of the image with each memory location being 24-b wide. The BRAM is initialized with the pixel values of the image using a

coefficient file. The initialization requires the pixel values of the image to be specified in a specific file format called the coefficient (.coe) format. In this format, the data (pixels in present context) has to be specified in binary or hexadecimal format separated by comma character. The data values get assigned to the memory locations in ascending order, which means the first data gets assigned to memory address 0, second data to address 1, and so on. A MATLAB script is written which converts the image to corresponding .coe file format.

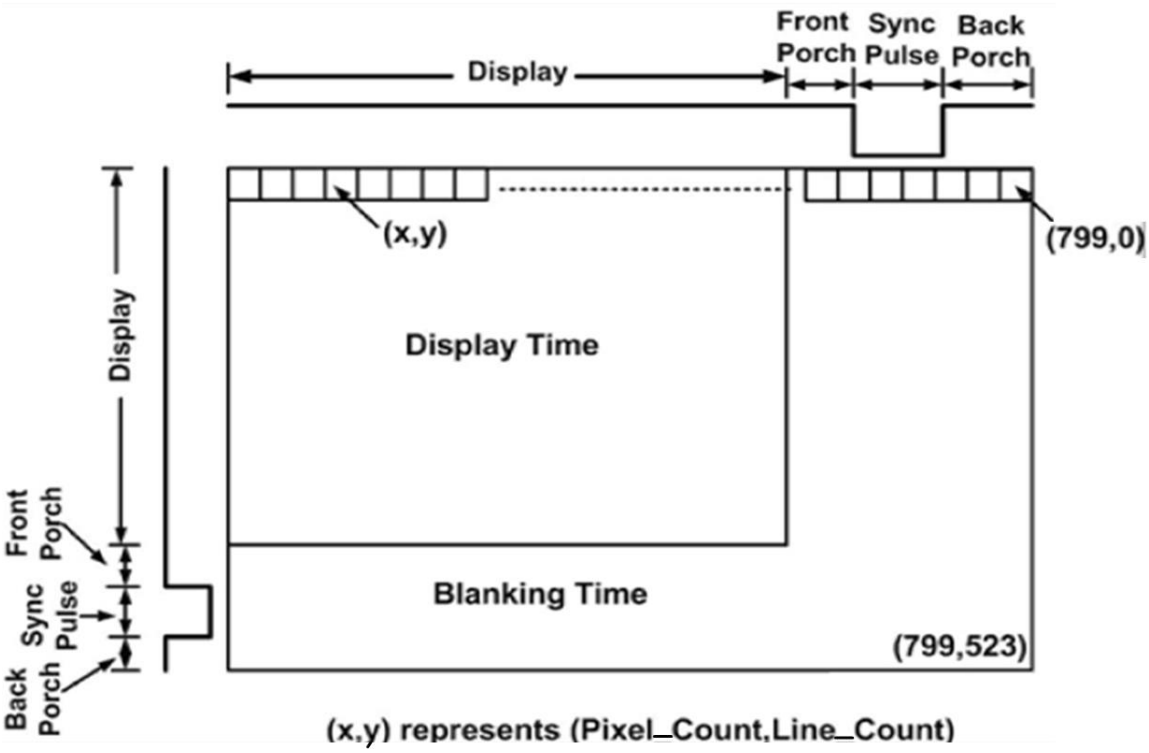


Figure 5.12: Relation of sync signals with the pixels of the screen

- **VGA Timing Unit:** The timing unit generates *h_sync*, v_*sync* and *blank* signals. The *h_sync* and *v_sync* signals collectively determine the pixel coordinates of the screen whereas the *blank* signal blanks out the screen at edges and during retrace period. The mapping of these signals on a display screen is shown in Fig. 5.12.

  The *h_sync* and *v_sync* signals are generated using counters that form the timing unit. The counters keep track of the number of pixels in the horizontal and vertical direction. According to Table A.1, there are 800 pixels in the

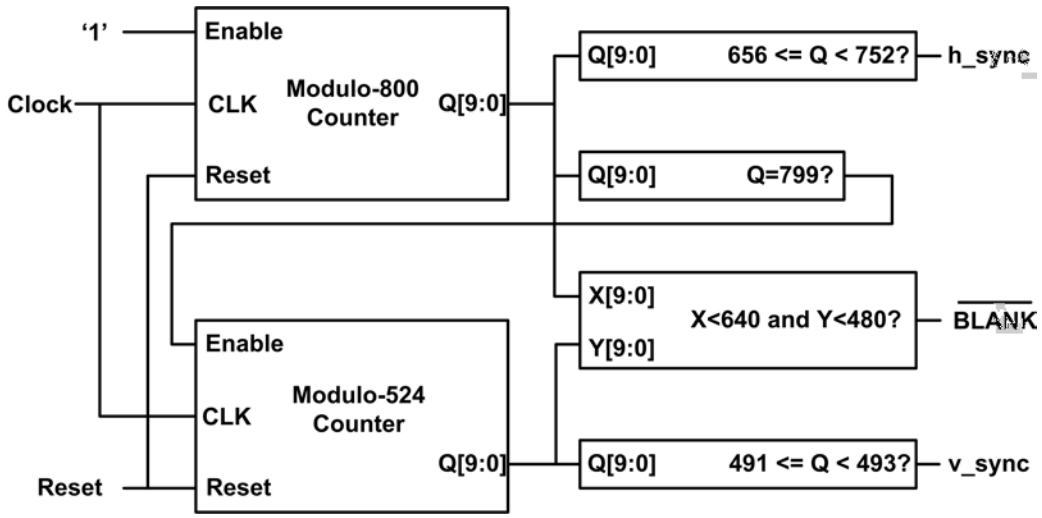


Figure 5.13: Logic for Timing Generation module

horizontal direction and 524 pixels in the vertical direction. To locate a pixel in the 2-D plane, two 10-bit counters called horizontal counter and vertical counter, referred as *h_count* and *v_count*, respectively are used. The three signals, *h_sync, v_sync* and *blank* are varied according to the count values of the two counters as shown in the Fig. 5.13. The sync signals are active high whereas the blank signal is active low. The pseudo- code for the timing unit consists of a mod-800 (pixel counter) and mod-525 counter (line counter). The conditions for sync and blank signal are presented below:

h_sync <= '0' when ((h_count > 655) and (h_count < 752)) else '1'

v_sync <= '0' when ((v_count > 490) and (v_count < 493)) else '1'

blank <= '1' when ((h_count > 640) and (v_count < 480)) else '0'

- **RGB Video Unit:** The image pixels stored in the memory has to be mapped to the corresponding x-y coordinate of the screen. The screen scan coordinates are derived from the synchronization signals (*h_sync* and *v_sync*). The RGB module generates the read address of BRAM_1 and read address of

BRAM_2 to access the 8-b *Red*, *Green* and *Blue* signals of the corresponding pixel to be displayed, based on the x-y coordinate of the screen being scanned. The three 8-b color components of the pixel is connected to the onboard Digital to Analog Converter (DAC) IC, which generates the equivalent analog signals for *Red*, *Green* and *Blue* pins of the VGA port.

- **Transformation Unit:** The flipping operation requires the pixel data present in BRAM_1 to be rearranged in BRAM_2. Thus, a correlation is derived between the write address of BRAM_2 and the read address of BRAM_1, depending on the transformation function. The transformations incorporated are the Identity, Vertical Flip, Inversion and InvertFlip. The identity transform duplicates the image in BRAM_2, Vertical Flip operation flips the image along vertical axis, Inversion flips it along the horizontal axis and InvertFlip performs flip along horizontal and vertical axis simultaneously. The logic for the said transforms is shown in Table 5.6 in the form of mathematical equations. In order to select between the four transforms, the two switches present on the board is used. The position of the switch selects one of the four transforms and accordingly the transformed image is stored in BRAM_2.

Table 5.6: Relation between write address of BRAM_2 with read address of BRAM_1

Position of Switch	Transform	Write Address of BRAM_2
00	Identity	wr_addr = read_addr
01	Inversion	wr_addr = read_addr + (38399) - (240*count)
10	Vertical Flip	wr_addr = (240*count) - read_addr -1
11	InvertFlip	wr_addr = (38399) - read_addr

Results

The loons image along with the four transformations- Identity, Inversion, Vertical flip, and InvertFlip is demonstrated on the monitor. The actual image is shown in Fig. 5.14 and the corresponding transformed images are shown in Fig. 5.15 (a-d).

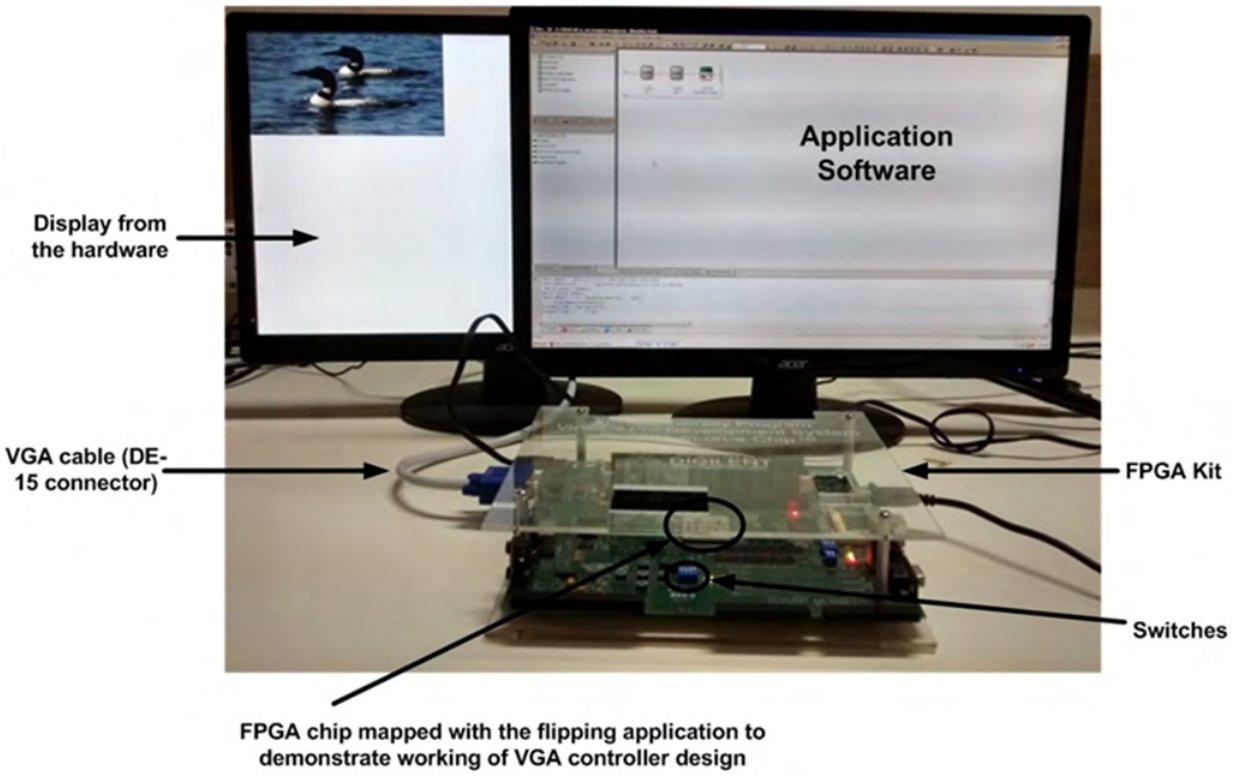


Figure 5.14: Hardware Setup for image transformations using VGA interface.

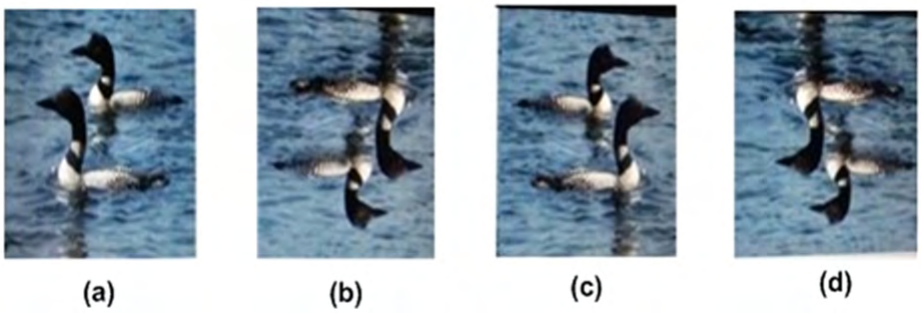


Figure 5.15: Results for (a) Identity (b) Inversion (c) Vertical Flip (d) InvertFlip transforms on the "240 x 160" loons image.

5.7.1 RS-232 Communication Interface

For objective analysis of the result, the output data must be compared with the ideal counterparts obtained from MATLAB and deviation must be quantified. A FPGA to computer communication interface aids providing/obtaining data to/from FPGAs. The interface also serves the debugging purpose as data can be easily comprehended. The RS-232 communication interface is adopted in this thesis for this purpose.

For a standard image of size "1280×720" where each pixel is represented by 8 bits, the total number of bits are 7.3728 Mbit (1280×720×8). According to RS-232 protocol, every byte has an overhead of 2 bits (1 start + 1 stop) resulting total number of bits to 9.216 Mbit. A Baud Rate of 115200 bps imply transmission of 115200 bits in one sec, hence "1280×720" image require 80 sec (= 9.216 M / 115200) to get transferred from FPGA to computer.

FSM of UART Controller

The transmitter component of UART is employed in the thesis to transmit the data from FPGA to PC. The UART is represented as a Finite State Machine (FSM) shown in Fig. 5.16. It consists of four states *idle*, *start*, *data* and *stop*. The application employing UART for transfer of data requires necessary signals to be generated at suitable intervals which governs the transition of states. Accordingly, the FSM is controlled by four signals *tx_start*, *end_of_data*, *data_bit* and *uart_clk*. They are defined as-

- **tx_start**: It implies start of communication through UART and is driven by a signal that indicates that the data is ready to be sent.
- **end_of_data:** It is asserted when the entire data is read completely. It is used to stop transmission of same data repeatedly.
- **uart_clk:** It is oversampled (16 times higher) clock signal of the baud rate clock signal generated by the Baud Rate Generator module.Thus, there are 16 clock cycles of uart_clk in a single clock of baud rate.
- **data_bit:** It is a counter which counts the number of bits transmitted in data state.

The detailed explanation of the four phases of the UART transmitter is now discussed. When the application employing the UART transmitter has data to be transmitted through the interface, it dumps the data in a memory and generates a signal (interfaced to the *tx_start* of UART transmitter) signifying the start phase of transmission. When the entire data is transmitted it de-asserts this signal signifying the end phase of transmission. The duration between the start phase and stop

phase is reserved for transmission and is called transmission phase. The UART transmitter stays in *idle* state when it is not in the transmission phase. The *tx* line of the RS-232 port is held high (logic value '1') in this state. The transmission of data occurs in *data* state. Data is transmitted in frames of bits, where each frame has a variable run length of data bits (5-8) and an optional parity bit. The start bit is active low which drives the *tx* pin from high state to low state indicating the start of data transmission. The stop bit is active high in order to support the transition of high state to low state when consecutive frames are transmitted.

These phases are obtained by means of a state machine shown in Fig. 5.16. The period of start bit, stop bit and data bit is same and decided by the baud rate. When UART transmitter needs to transmit data, it asserts the *tx_start* signal to '1' until the entire data is entirely transmitted. The high *tx_start* signal transits the FSM from *idle* state (*tx* pin is high) to *start* state (*tx* pin is low). The UART stays in *start* state for a period of 16 cycles of *uart_clock* (1 cycle of baud clock) and then transits to *data* state, where the data is transmitted over the *tx* pin. The 8 bits (preset to 8) of data is transmitted in little endian format and then the FSM transits to *stop* state (*tx* pin is high). Now, if the status of *tx_start* (= '1') and *end_of_data* (= '0') is satisfied, the FSM transits to *start* state and repeats the pattern. The *end_of_data* signal is asserted when the application still needs to transmit data which is under computation.

Design of UART Controller

The control signals necessary for transfer of data from FPGA board to computer are generated by a custom designed UART controller based on the RS-232 standard. The designed controller reads data from the memory in parallel manner (8 bits at a time) and transmit it on the *tx* pin serially. When the memory is completely filled, a *write_complete* signal is asserted, which is connected to *tx_start* of the controller. The active high *tx_start* marks the initiation of data transmission through the UART.

The controller is divided into 3 sub-modules- Baud Rate Generator, Parallel to Serial Converter and the Memory Controller as shown in Fig. 5.17. The Parallel to

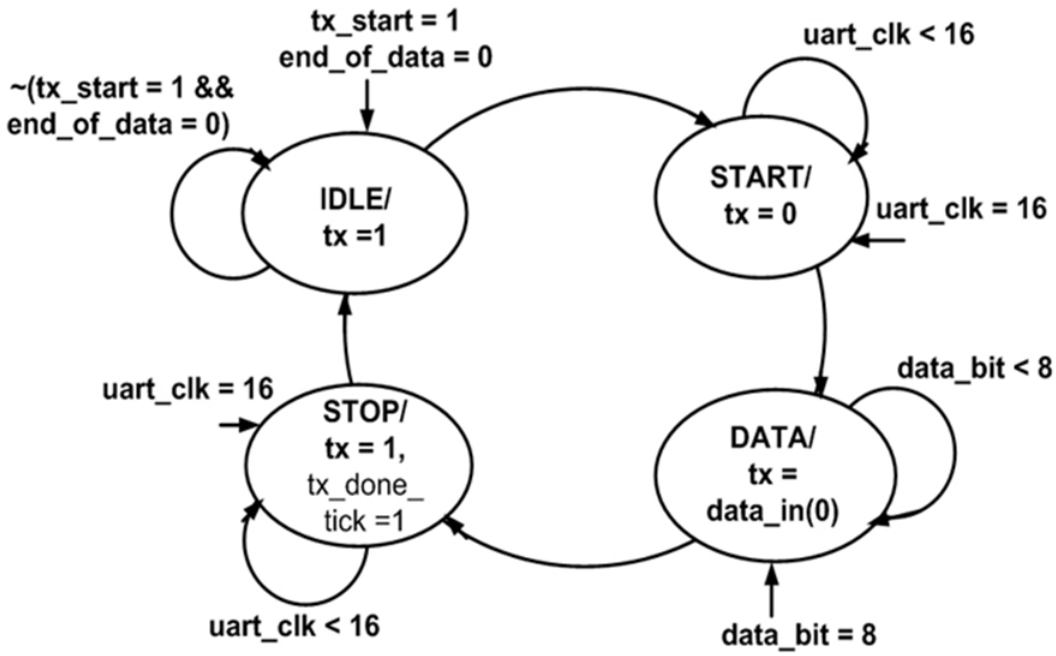


Figure 5.16: State Machine of UART Controller

Serial Converter is a shift register that loads data in parallel and then shifts it out bit by bit at the preset baud rate. The FPGA board clock of 100 M*Hz* is required to be down converted to sixteen times the baud rate frequency to preserve the bit value transmitted. Hence, a clock of 1.8432 M*Hz* (=16×115200) needs is generated using the baud rate generator which consists of a mod-54 counter. It divides the 100 M*Hz* clock by 54 resulting in a clock frequency of 1.851 M*Hz*. The Memory controller generates the read address of the memory. When the last location of the memory is reached, *end_of_data* signal is asserted which marks the completion of transmission through the *tx* line.

Result

The data received at computer through UART can be viewed using any terminal software. As a test case, a 16×16 checker board image is operated upon by 3×3 gaussian smoothing transform application. The computation is mapped on the SAC architecture and UART interface is used to transfer the smoothed image to computer. The output data received from UART is shown in Fig. 5.18 in the CoolTerm serial terminal.

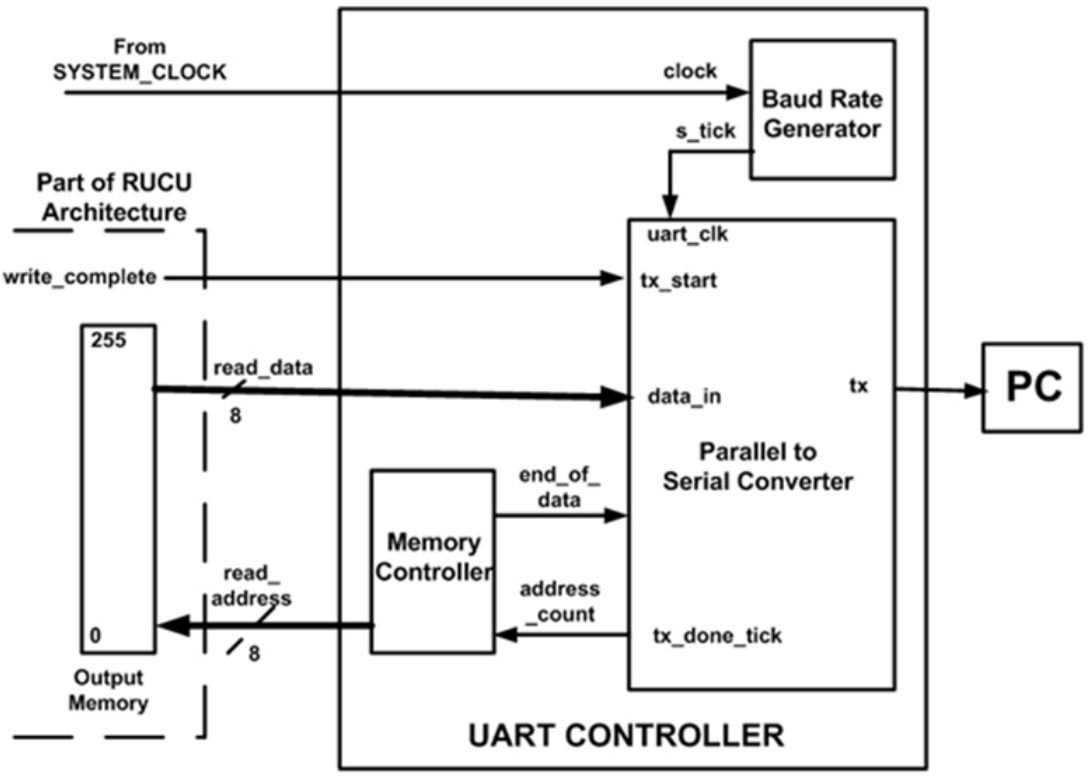


Figure 5.17: Block diagram of UART Transmitter Controller.

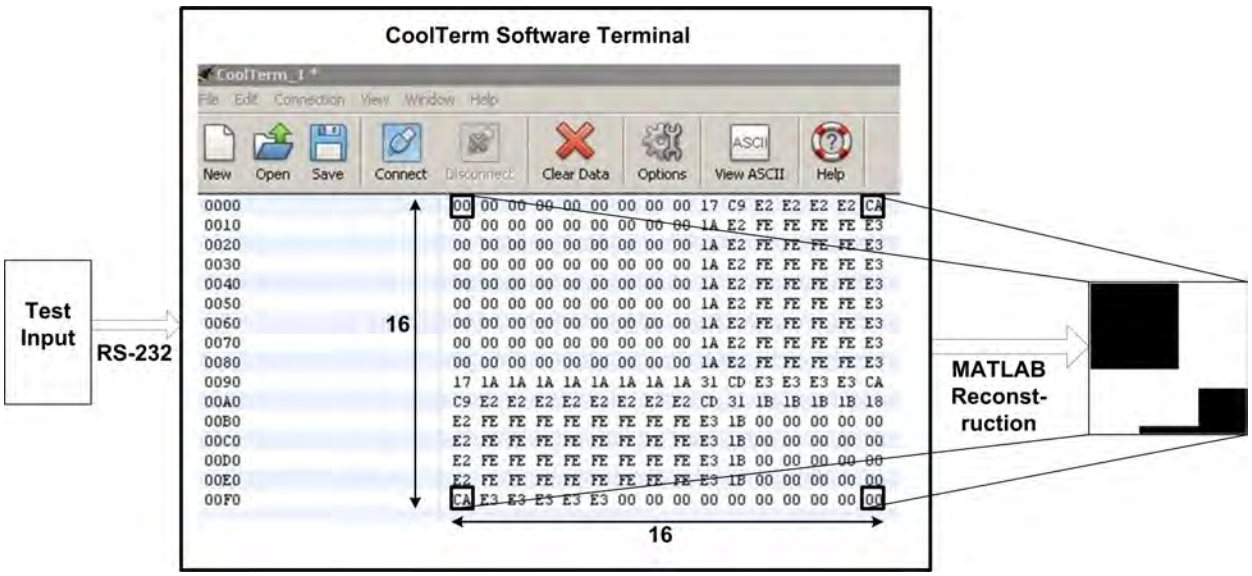


Figure 5.18: Screenshot of Coolterm Terminal showing data received in hexadecimal format

5.8 Discrete Wavelet Transform (2-D DWT)

The 2-D DWT operation of an image in one shot requires large memory which varies according to the different sized images it operates upon. Therefore, DWT is

Table 5.7: Comparison of the actual and reconstructed Lena images

Image1	Image2	PSNR(dB)	Mean Square Error	L2-Norm
Actual	Matlab Reconstructed	74.8819	0.0020	0.9904
Actual	Hardware Reconstructed	74.6535	0.0022	0.9514
Matlab Reconstructed	Hardware Reconstructed	87.7629	1.0883e-04	0.9606

implemented by block wise approach in this thesis. The two parameters observed while selecting the block size are computational complexity and mean square error between the actual and reconstructed images. Thus, the 2-D DWT of a Lena intensity image of size 128×128 is analyzed by varying the mask size in MATLAB. The result obtained is presented in Table 5.8 as given below.

Table 5.8: The comparison of various 2-D DWT block sizes.

Block Size	75% Compression
	<Computation, MSE >
2x2	<3.6e4, 2.04E-03>
4x4	<8.6e4, 1.83E-03 >
8x8	<1.3e5,1.06E-04>
16x16	<3.8e5, 2.89E-04>

It can be seen from Table 5.8, that to achieve the same compression, the smaller masks require fewer computations as compared to the longer masks but they suffer from a higher MSE. For instance, performing 75% compression using 2x2 mask requires only 9.4% (= 3.6e4) of the computations required by 16x16 mask (= 3.8e5), but at the same time the MSE experienced is 7.06 (= 2.04e-3/2.89e-4) times of the later. As shown in Fig. 5.19, the computation and MSE constraints converge at a point close to mask size of 8×8. Thus, to balance this trade-off, the standard practice of using an 8×8 mask size is followed.

5.8.1 Block ROM

The control word, wavelet filter coefficients and the image pixels are supplied into the FPGA using a single port block Read Only Memory (ROM) available on the FPGA board. The required memory depth is equal to 42 +(7/4) N^2 (6 bytes of the

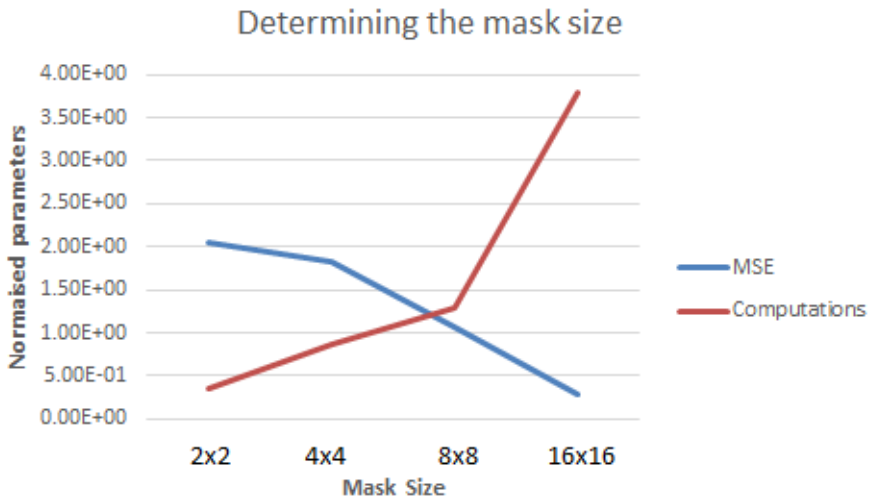


Figure 5.19: The MSE and # of computations plotted against different 2-D DWT block sizes.

control word, 36 bytes of filter coefficients and $(7/4)N^2$ bytes image pixels after padding the N $\times$ N image).

5.8.2 Digital clock manager (DCM)

The operating frequency of the architecture is 46.9 MHz, which is reduced to match up with the UART frequency to avoid losing data due to overwriting. The baud rate of the UART is 115200 bps and takes 69.4 μs to read 1 byte. The resulting frequency of the UART becomes 14.4 kHz. The lowest frequency that is possible to generate from the DCM is equal to $1/16^{th}$ of the input frequency, thus generating a clock frequency of 6.25 MHz since the clock frequency of the FPGA is 100 MHz, which is further reduced using a counter. The architecture generates 128 bytes of computed data in 440 cycles for one 8 $\times$ 14 block of padded image. A 256 byte buffer is used which will first write complete 128 bytes of data coming from architecture and then once the UART will start the reading that 128 bytes of data in the meantime the architecture would have written another set of computed 128 bytes working @ 50kHz.

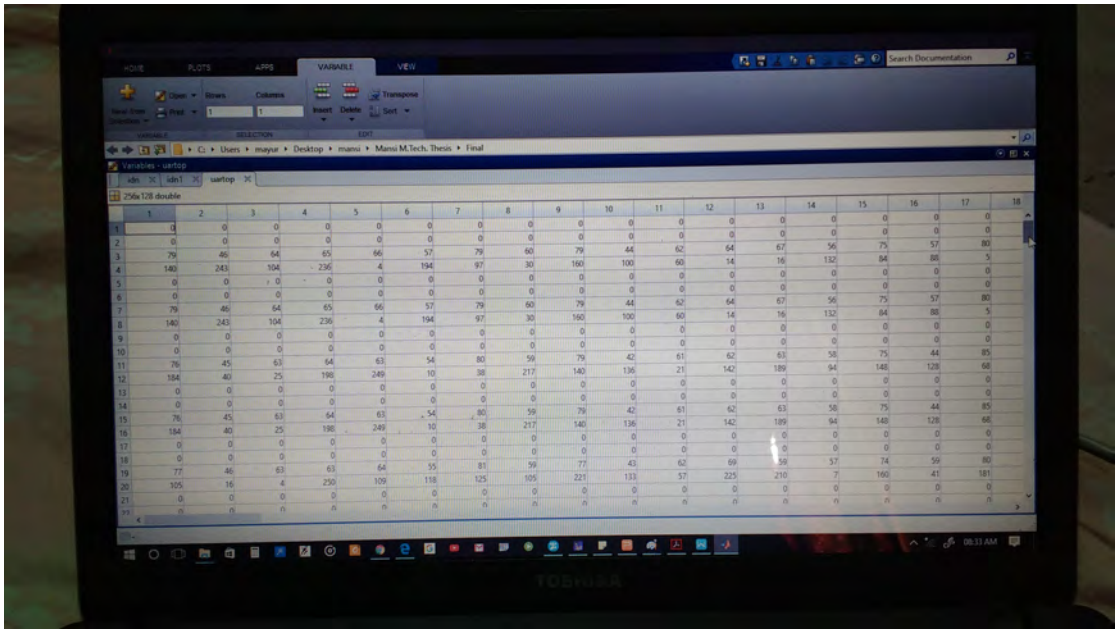


Figure 5.20: Reading the approximation coefficient in MATLAB

5.8.3 Reconstructed image

We performed 2-D DWT on the architecture taking two standard test images as the input, a 16 $\times$ 16 checker board image with Haar wavelet and a 128 $\times$ 128 Lena image with bior2.2 wavelet. The architecture computes the approximation coefficients (*i.e.* LL sub-band) for both these images using respective wavelets. Thus, the cA matrix obtained is of one-fourth the order of the original image. For example, for 16 $\times$ 16 checker board image the cA matrix is of 4 $\times$ 4. These cA coefficients are transferred from FPGA to PC through the developed RS-232 communication interface. The Fig. 5.20 shows the approximation coefficients being read in MATLAB after performing 2-D DWT on Lena image using Bior2.2 wavelet.

The image is reconstructed using these coefficients which denotes 75% compression *i.e.* only 25% of data (the cA matrix) since cH, cV and cD are discarded and considered as null matrix. Reconstruction is done the reverse way of decomposition, first is upsampling with zeros followed by periodic padding and convolution with the wavelet coefficients. The Fig. 5.21 shows the reconstruction of image I_R from C coefficient matrix.

The mean square error between the coefficient matrix C (consisting of cA, cH, cV and cD) and the reconstructed image matrix I_R of both checkerboard and lena im-

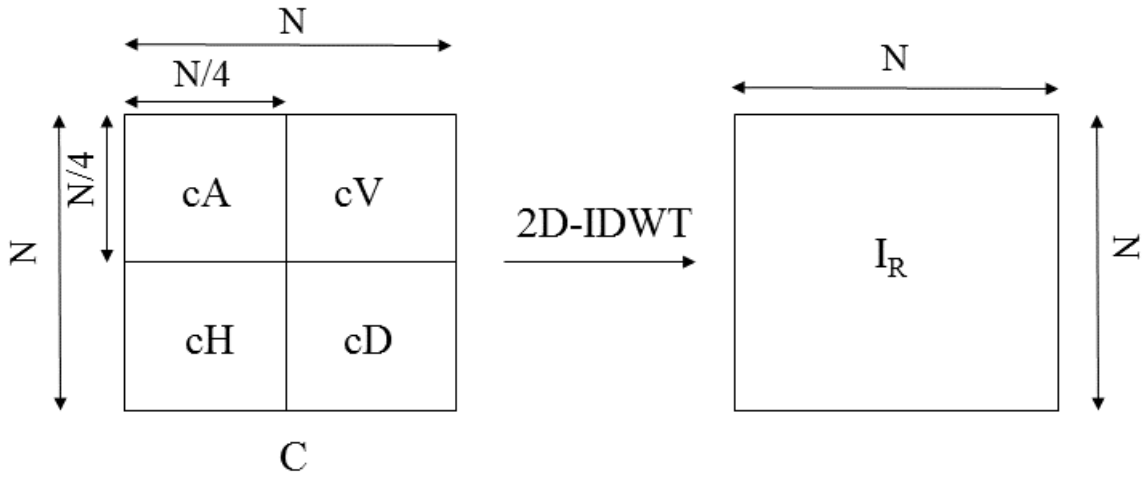


Figure 5.21: Reconstruction of image I_R from the coefficient matrix C where cA is the computed coefficients and cH, cV and cD are the null matrices

age is 0.3957 and 0.3022, respectively. The actual and the reconstructed images both from the MATLAB simulation and the hardware implemented are shown in Fig. 5.22 and Fig. 5.23, respectively.

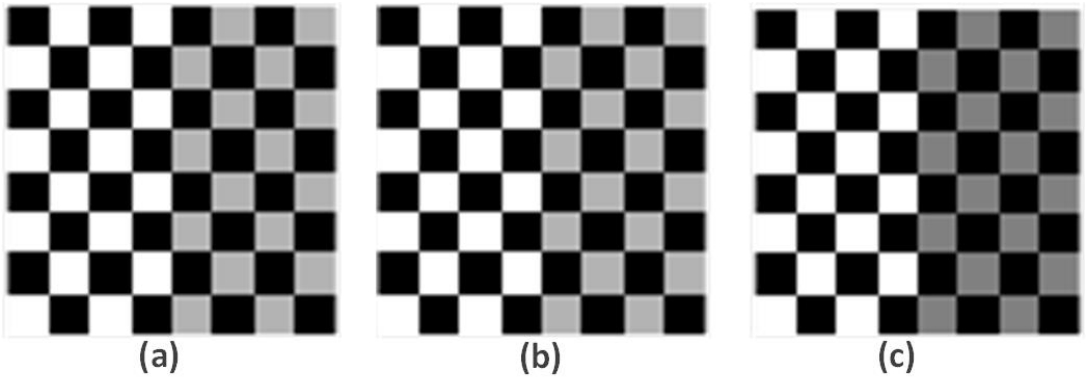


Figure 5.22: DWT Results on 16×16 checkerboard image (a) Actual (b) MATLAB simulated (c) Hardware computed.

These images are compared with MATLAB compressed counterpart in Table 5.9 for checkerboard 16 × 16 and in Table 5.10 for Lena 128 × 128.

Device utilization summary for the reconfigurable architecture with 2D-DWT function for 128× 128 Lena image is given below:

Selected Device : 2vp30ff896-6

Number of Slices: 6234 out of 13696 : 45%

Number of Slice Flip Flops: 3915 out of 27392 : 14%

Figure 5.23: DWT Results on 128×128 Lena intensity image (a) Actual (b) MATLAB simulated (c) Hardware computed.

Table 5.9: Comparison of the original and reconstructed checkerboard images.

Image1	Image2	PSNR(dB)	Mean Square Error	L2-Norm
Actual	Matlab Reconstructed	Infinite	0.00	1.0000
Actual	Hardware Reconstructed	68.1271	0.01	0.8389
Matlab Reconstructed	Hardware Reconstructed	68.1271	0.01	1.1920

Table 5.10: Comparison of the original and reconstructed Lena images.

Image1	Image2	PSNR(dB)	Mean Square Error	L2-Norm
Actual	Matlab Reconstructed	74.8819	0.0020	0.9904
Actual	Hardware Reconstructed	74.6535	0.0022	0.9514
Matlab Reconstructed	Hardware Reconstructed	87.7629	1.0883e-04	0.9606

Number of 4 input LUTs: 11462 out of 27392 : 41%

Number used as logic: 11206

Number used as RAMs: 256

Number of IOs: 3

Number of bonded IOBs: 3 out of 556 : 0%

Number of BRAMs: 15 out of 136 : 11%

Number of GCLKs: 6 out of 16 : 37%

Number of DCMs: 1 out of 8 : 12%

The clock cycles required for computing 2-D DWT is 496 per 8 × 8 block of a

N $\times$ N image. In general, the for level-1 decomposition the total number of clock cycles is $(31/4)N^2$. For level-2 decomposition $(31/4)N^2$ + $(3/2)N^2$ clock cycles are required and for level-3 decomposition $(31/4)N^2$ + $(3/2)N^2$ + $(3/8)N^2$ clock cycles are required. Thus, in total (77/8) N^2 clock cycles are required for 3-level decomposition.

5.9 Multiple Function realization

Tile #1 and 2 are mapped with FIR filtering leaving Tile #3 and 4 unused in the four tile topology. The results for such configuration is captured in the serial port window and shown in Fig. 5.24. The first 8-b section of output is forced to "FF" to mark the start of result. The CU output is selected in such a way that every result is repeated twice. The zero initial output indicates the cycles exhausted in configuring the architecture and writing data into memory.

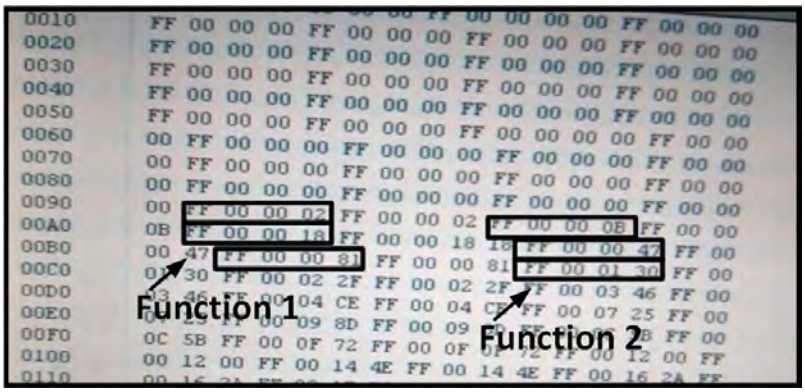


Figure 5.24: Hardware results for simultaneous mapping of two functions on SAC Architecture.

5.10 Clock Profiling

The clock profile of the architecture can be broadly categorised into configuration and computation phases. Configuration phase concludes once control word, coefficient and data are loaded through the 9-b input bus. The 54-b control word is loaded serially and takes 6 clock cycles. Another 36 clock cycles are consumed in loading 36 9-b coefficients. Data loading takes 1-64 clock cycles as required by the

target function. For instance, multiplication requires one data whereas 64 data is written in memory for DCT operation. The computation phase takes 8 clock cycles to generate one output. However, certain functions may require intermediate results computations prior to final output. Additionally, CORDIC computation involve calculating three variable in each iteration. The computation cycles for target functions is listed below:

- CORDIC - 24 cycles/iteration
- DCT - 336 cycles/8×8 block
- DWT - 496 cycles/8×8 block
- Others - 8 cycles for every output

Other activities like storing output does not consume additional cycles as they are performed in parallel with configuration and computation phases resulting in cycle optimizations. Additionally, cycle saving is achieved by adopting mapping methodology that eliminates redundant computations as well as stores intermediate results within the architecture.

5.11 Gate Profiling

Each RU contains 9 XOR, 8 AND, 27 flip-flops and 1 2:1 multiplexer that totals to 242 gates as 2:1 multiplexer and flip-flop have 4 and 7 gates, respectively. There are 208 equivalent 1-bit adders in CU along with 42 flip-flops amounting to 4376 gates. The configurable datapath multiplexers have 396 equivalent 2:1 multiplexer whereas coefficient multiplexers are the largest contributor with 9608 gates owing to large DCT, DWT and CORDIC multiplexers. This brings the SAC architecture gate count to 24280 gates. The interfacing memory has 954 flip-flops and adds another 6678 gates to the system.

5.12 Comparison with Similar Works

The architecture developed in this work acts as multiple accelerator unit because of its configurable datapath and adopts effective mapping methodologies translating to gate and clock savings. Authors in [22] advocates the use of accelerators dedicated for commonly used signal processing operations like FIR filtering, CORDIC algorithm, moving average intergration *etc*. The demonstarted biomedical signal processing algorithm in [22] makes use of of the aforementioned functions and the gate count reported for them is 11k, 9.3k and 37k, respectively with operating frequency of 1MHz for individual execution of these functions. [21] implements ECG signal processing algorithm on a custom microprocessor and reports 36k equivalent NAND2 gates at 600kHz operating frequency. Another biomedical signal processing system in [77] reports 195k equavilent NAND2 gates This work reports a gate count of 24280 gates providing $\approx 1.5\times$ and $8\times$ gate (and area) savings with respect to [21] and [77] , respectively. Performance advantage of $\approx 45\times$ and 78 $\times$ is achieved as compared to [77] and [21].

Table 5.11: Comparison with state of the art.

	[22]	[21]	[77]	This work
Equivalent Gate Count	81.3k	36k	195k	24,280
f[MHz]	10	0.6	1	46.9
Target functions	32-tap FIR, 65-pt median, 512-pt FFT, CORDIC	LPF,HPF, Derivative-square, moving average, peak detection	CWT based QRS detection	FIR, Moving Average, CORDIC, DCT, DWT, Multiplication, Addition, MAC, Square
On-Chip Memory	64kb	-	2Mb	954b
Platform	Hardware accelerators	Custom μP	CoolFlux BSP	Custom Reconfigurable Architecture

5.13 Conclusion

The hardware implementation of the SAC architecture on the FPGA development board is presented in this chapter. The configuration methodology of the architecture is discussed which comprises of loading the control word, coefficients and data. This is achieved using a time multiplexed 9-b input bus. The control word for target functions and different architecture topologies is provided. The fields of control word can be configured independently which gives rise to numerous architecture topologies and thus enables efficient realization of a wide variety of functions. The simulation and hardware computed results for target functions are provided along with their resolution and input range. The underlying methodology behind choosing the mask size for DCT and DWT is governed by the computational complexity and mean square error between the actual and reconstructed data. This approach led to a optimal hardware implementation for these functions. Two interfaces, VGA and RS-232, developed for visual perception and output communication between the FPGA and computer. The protocol, required input signals and controller fundamentals are discussed in detail. An interesting topology of realizing two functions simultaneously is also presented. This realization further propelled this work towards incorporation of on-the-fly reconfigurability. The cycle and gate count of the profile is presented followed by the comparison with state-of-the-art realizations pertaining to biomedical signal processing applications. It is observed, that the SAC architecture realization reports minimum gate count among dedicated or software-hardware implementations with similar target functions.

CHAPTER 6

On-the-fly Application Reconfigurability

Often signal processing applications consist of a sequence of operations required to be performed in succession. This is particularly true for applications pertaining to the biomedical signal processing domain. bioelectric signals are analyzed for certain characteristic features present in them which are indicatives of deviation from their normal behaviour. For instance, QRS complex present in ECG signal serves as arrhythmia indicator. For this purpose, there may exist a couple of signal processing blocks dedicated for feature extraction, classification *etc.* In general, the low amplitudes of bioelectric signals are susceptible to noise arising out of multiple sources such as environmental, instrumental, physiological *etc.* Therefore, noise removal block precedes the bioelectric signal analysis blocks. Furthermore, static configurable system emulating the bioelectric signal processing chain would require separate blocks configured for each of the functions in signal chain connected in tandem. Alternatively, a single block with configurable interconnection network can be used repeatedly to perform the signal chain functions one after the other. Following this scheme, the hardware resources, memory and I/Os are shared between the functions in a time-multiplexed manner with dynamically changing datapath.

In this chapter, the possibility of incorporating the aspect of dynamic configuration on the SAC architecture is explored and demonstrated by means of a state-of-the-art QRS detection application. The SAC architecture has configurable interconnections which favours multiple and varied function/application emulation. Adding the provision of dynamic configuration enables changing the datapath on-the-fly without user interference. The elements of configuration, including the

control word and various topologies exhibited, are discussed. SAC architecture is demonstrated to support different biomedical signal processing operations in previous chapters (Chapters 4 and 5) along with its supporting memories. The existing data memory structure is conceptually divided into various sections to store results of intermediate functions. Additionally, there exists few DSP operations that have data dependencies, thus the methodology for function state restoration is discussed. SAC architecture emulating the QRS detection application is ported on the FPGA development board. Further, the circuit analysis of the design is carried out for accurate energy and timing estimations. The on-the-fly reconfigurability concept is applied on the 3×3 variant of the SAC architecture. This limits the number of functions that can be emulated on the platform. Nonetheless, the scalable nature of the developed methodology enables adding multiple 3×3 blocks in the architecture and extracting higher order or complex functionalities, that an individual 3×3 block might not be sufficient to support.

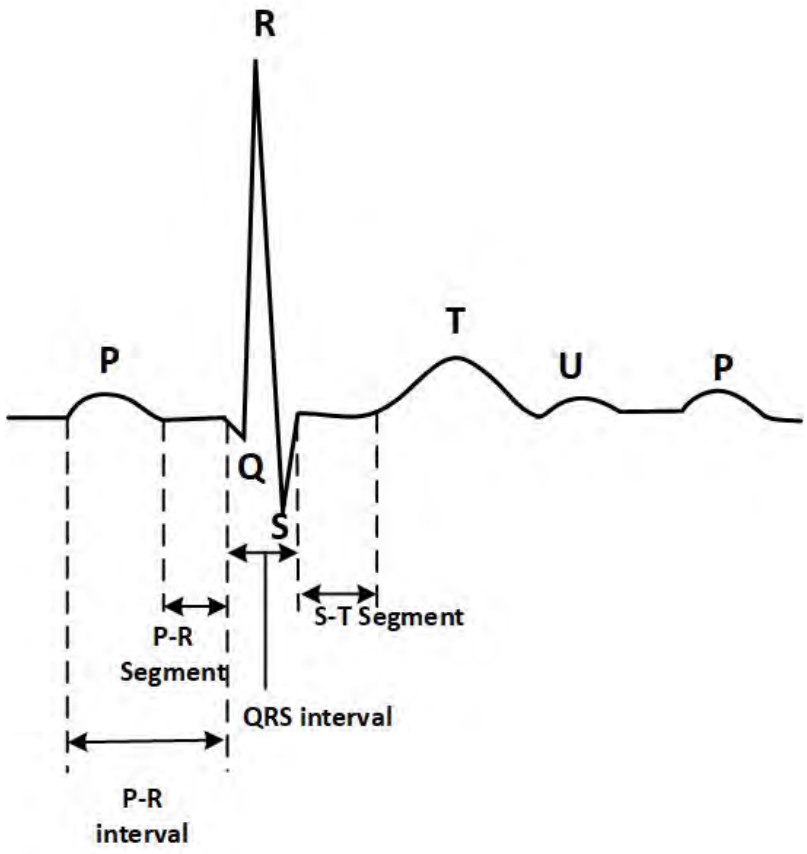


Figure 6.1: ECG fudicial points.

6.1 Morphology of ECG signal

Much information provided by the ECG is contained in the morphologies of three principal waveforms (a) P wave, (b) QRS complex, and (c) T wave. The cardiac activity, magnitude and duration of characteristic features of ECG signal (shown in Fig. 6.1) are presented in Table 6.1. Appendix B contains details of different ECG lead systems.

Table 6.1: ECG characteristic features specifications for normal ECG signal.

ECG Feature	Biological Interpretation	Contour	Amplitude (mV)	Duration (ms)
P wave	Atrial activation	smooth	<0.2*, <0.1@	<120
PR interval	impulse travel from atria to ventricles	-	-	100-210
QRS Complex	Ventricular depolarization	sharp and peaked	<0.5*, <1@	70-110
ST segment	transition between depolarization and repolarization	-	-	
T wave	Ventricular repolarization	smooth and rounded	<0.5*, <1.5@	(460-110)
U wave	prolonged recovery or after potentials	10% of T-wave		

* Frontal plane leads,
@ Tranverse plane leads

6.2 Pan-Tompkins Algorithm (PTA) for QRS detection

QRS detection in ECG processing is an essential tool in diagnosing different medical conditions [176]. A standard approach to detect the QRS complex is the Pan Tompkins algorithm that detects QRS complexes with accuracy up to 99.3% [2] for a 24 hour MIT/BIH arrhythmia database. The algorithm is modelled with steps shown in Fig.6.2(a). The input ECG signal is passed through a band pass filter to remove any spurious noise present in the signal that restricts the signal to QRS

frequency band of 5*Hz* to 15*Hz*. The algorithm facilitates QRS detection by identifying the high slope QRS complex in the ECG signal by means of differentiation and squaring. Following this, the moving average (MA) block extracts the QRS width and the peak detection block detects the R waves based on the amplitude, slope and width of the ECG signal [2]. The transfer function of the blocks is shown in Table 6.2.

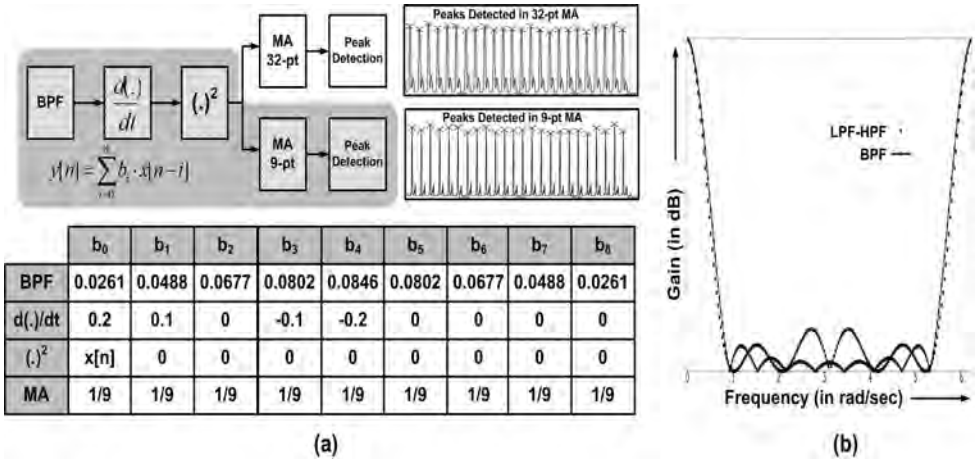
Table 6.2: Transfer function of PTA blocks [2].

Block Name	**Transfer Function**
LPF	$(1\text{-}2z^{-6}+z^{-12})/(1\text{-}2z^{-1}+z^{-2})$
HPF	$(-1+32z^{-16}+z^{-32})/(1+z^{-1})$
Differentiator	$(z^{-2}\text{-}2z^{-1}+2z^{1}+z^{2})/8$
Squaring	$x[n]^2$
Moving Average	$\frac{1}{k}\sum_{i=0}^{k-1} x[n-i]$

6.2.1 Pan-Tompkins Algorithm: Analysis and Discussion

In [2], the combination of low pass and high pass filters yielded integer coefficients for the frequency band of interest. However, the non-performance intensive nature, in terms of operating frequency as well as incoming data rates, of biomedical signal processing applications seldom requires coefficient restriction to integers or power of 2 [36]. However, to ensure minimum hardware implementation, we reduced the LPF and HPF functions of PTA to a band pass filter (BPF) using filter design and analysis (FDA) tool in MATLAB by specifying its lower and upper cutoff frequencies as 5 *Hz* and 15 *Hz*, respectively at the sampling rate of 200 samples/s. The BPF coefficients obtained from the FDA tool are shown in Fig. 6.2(a). To ascertain whether 9-tap filter is adequate for acceptable noise removal in the ECG signal the frequency response of BPF and LPF-HPF combination are plotted using MATLAB (shown in Fig.6.2(b)). Approximately 78% of the BPF response data points are limited within 1σ of LPF-HPF and 91% within 3σ, indicating good coherence between both the magnitude responses. Fig. 6.3 (a) shows the ECG signal processed using the PTA defined LPF-HPF combination and the 9-tap BPF obtained from MATLAB after normalization. The absolute error between the two

is plotted in Fig. 6.3(b). The maximum absolute and % relative error are 0.034 and 4.59%, respectively.



	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
BPF	0.0261	0.0488	0.0677	0.0802	0.0846	0.0802	0.0677	0.0488	0.0261
d(.)/dt	0.2	0.1	0	-0.1	-0.2	0	0	0	0
$(.)^2$	x[n]	0	0	0	0	0	0	0	0
MA	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9

Figure 6.2: (a) Reduced PTA signal chain. (b) Magnitude response of LPF-HPF and BPF in PTA.

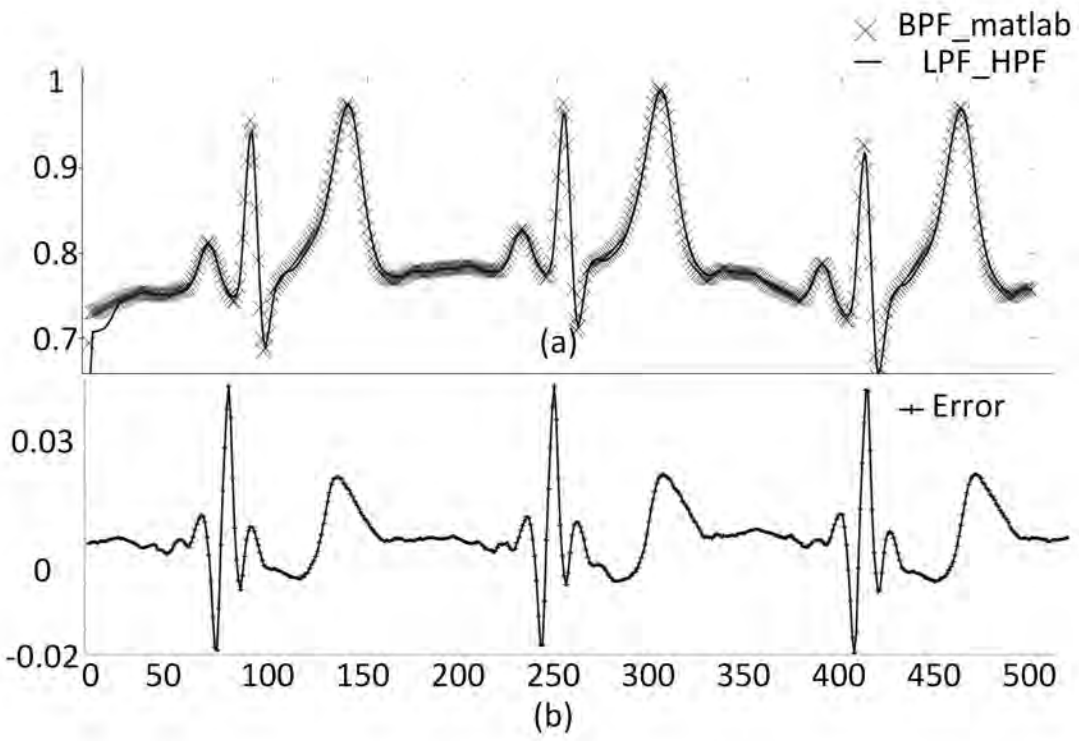


Figure 6.3: Noise attenuation block. (a) The LPF-HPF (PTA) and modified BPF operated ECG signal. (b) Absolute error between them.

In case of moving average (MA), selection of the right window size is crucial, as too wide a window results in merging of both the R and T peaks into one peak and too small a window causes multiple peaks in the QRS complex. The PTA

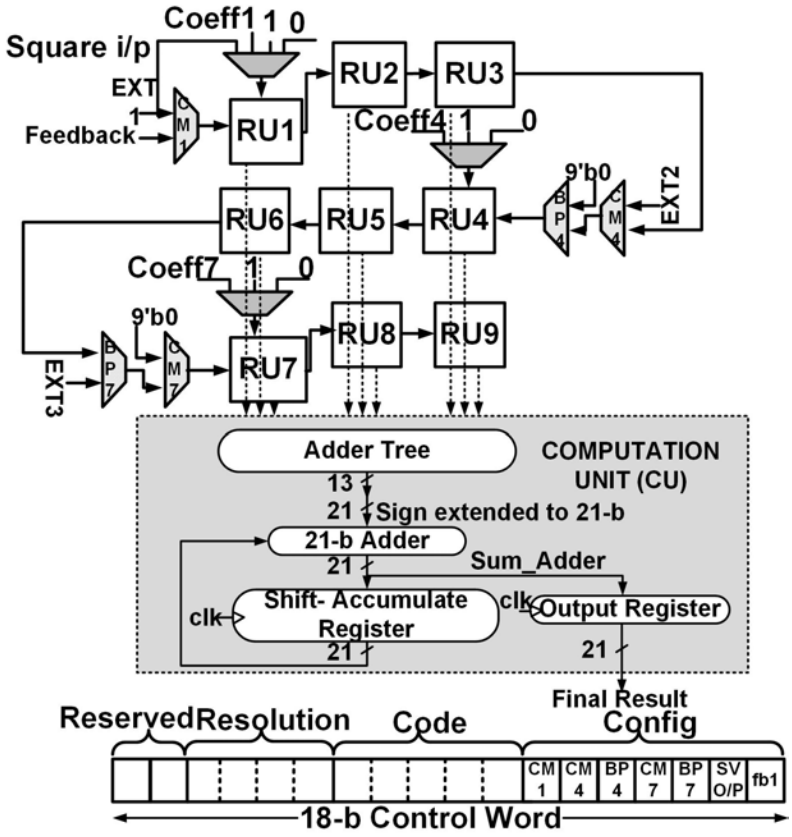


Figure 6.4: The (3×3) SAC architecture along with its 18-b control word.

chain is analyzed for 9-point MA using a running window peak detection algorithm having window size of 165 samples, which roughly indicates the frequency of the ECG signal and yields a heart rate of 72.72 beats/s.

The peaks detected in 32-point and 9-point integrated ECG signals are shown in Fig. 6.2(a) and 9-point MA also results in accurate peak detection. Therefore, 9-points MA operation is an acceptable solution while requiring lower number of taps. The reduced PTA signal chain is shown as cross hatched blocks in Fig. 6.2(a).

6.3 The Shift-Accumulate (SAC) Architecture

Careful observation of the transfer functions of QRS signal chain indicates that BPF, differentiation, squaring and MA functions can be realized by a series of shift-accumulate operations resulting in multiplier-less implementation of these functions. The modified PTA is mapped on the 3×3 variant of the SAC architecture shown in Fig. 6.4. The partial products generated in register units (RU) are

added in ripple carry adder tree in the computation unit (CU), producing the final result after eight clock cycles. The RU structure and operational details of RU and CU is discussed in Chapter 3.

6.3.1 Control Word and Configurable Datapath

The architecture is configured for multiple functions through the 18-b control word that has two fields, namely, *Code* and *Config*. As shown in Fig. 6.4, the *code* field is a 5-b binary data used to recognize the operation under execution. For example, “00001" serves as FIR/differentiation/MA code, “00111" is code for squaring *etc.* The *config* field is primarily responsible for defining the data path for the function of interest as well as facilitates minimum RU implementation and is composed of select lines of the configuration (*CMx*) and bypass multiplexers (*BPx*), output tracking bit (*SVO/P*) and the feedback bit. The architecture contains three 2:1 9-b configuration multiplexers and two 2:1 9-b bypass multiplexers. The configuration multiplexer makes the selection between the external input, the data from the previous RU and the feedback data (in case of *CM1*) using a 1-b select line. The multiplexers *CM1* feeds 9-b truncated output as input to the RU creating a feedback path. This is advantageous to implement functions that need output feedback as is often required by DSP functions, IIR filtering for instance.

The bypass multiplexer disconnects RU from the active datapath when its select line is clear and vice versa, and feeds zero to the RU data register. The data supplied to the data register of RUs through the configuration multiplexers is shown in Table 6.3. The output tracking bit when set ensures storing of the output in the memory for further use in other functions. The feedback bit (denoted as *fb1* in the control word) is set for functions involving feedback and is clear otherwise. The remaining bits in the control word are reserved for future modifications. For instance, “0111110" in the config field connects the nine RUs in a sequential chain as *CM4*, *CM7* and the bypass multiplexers are set. Additionally, the generated output is stored in the architecture (*SVO/P* = ‘1’) and the function implemented on the architecture does not involve feedback (*fb* = ‘0’).

Table 6.3: Data selection by Configuration and Bypass Multiplexers

RU1, RU2 and RU3 are always active having RU1→RU2→RU3 interconnection				
CM4	**BP4**	**CM7**	**BP7**	**RU Interconnection**
Both BP4 and BP7 are '0'				
X	0	X	0	RU1→RU2→RU3
Either BP4 = '1' and BP7 = '0' or BP4 = '0' and BP7 = '1'				
0	1	X	0	RU4→RU5→RU6,External data feeding to RU4
1	1	X	0	RU1→RU2→RU3→RU4→RU5→RU6
X	0	0	1	RU7→RU8→RU9,External data feeding to RU7
X	0	1	1	Not feasible combination as RU7 is connected to RU6 but RU4 is bypassed
Both BP4 and BP7 are '1'				
0	1	0	1	RU4→RU5→RU6; RU7→RU8→RU9, both connected to External inputs
0	1	1	1	RU4→RU5→RU6→ RU7→RU8→RU9, RU4 connected to External input
1	1	0	1	RU1→RU2→RU3→RU4→RU5→RU6, RU7→RU8→RU9, RU7 connected to External input
1	1	1	1	RU1→RU2→RU3→RU4→RU5→RU6→ RU7→RU8→RU9

6.4 On-the-fly Reconfigurability

Biomedical processing algorithms often require multiple processing steps with data processing rates ranging from 1k to 10k samples per second [36]. The SAC architecture can support higher frequency ($\geq$45 MHz) meeting the data processing rates in addition to the multiple configurations on-the-fly.

For example, in PTA, the hardware can be first configured as BPF for n_1 clock

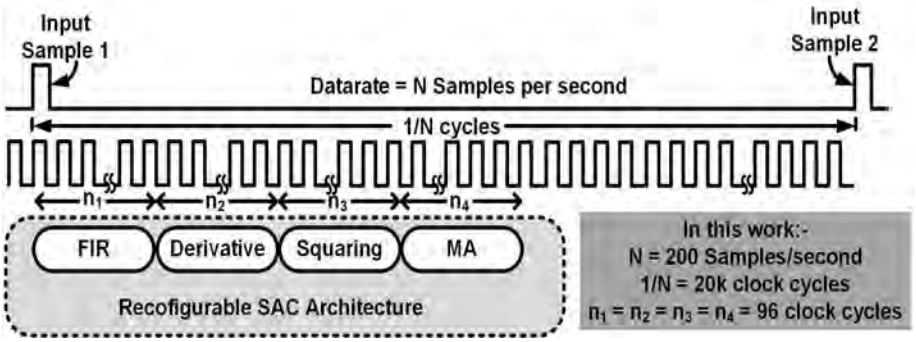


Figure 6.5: Timing Diagram of QRS signal chain in respect to the input sample frequency.

cycles. Secondly, the control word can configure the SAC to a derivative block for n_2 clock cycles. Assuming the architecture takes n_3 and n_4 clock cycles to configure and execute squaring and moving average, respectively, the total cycles ($n_{tot} = n_1+n_2+n_3+n_4$) can still be smaller than incoming data sample rate as shown in Fig. 6.5. This indicates that the hardware spend majority of its time in idle state. Additionally, on-the-fly configuration requires storing intermediate results and restoring the state of intermediate functions prior to computation. This requires effective memory management methodology if a large memory is to be avoided.

6.4.1 Memory Management and Interpretation Methodology

The circular data memory is interpreted in a novel manner resulting in a memory management methodology for storing and feeding the data. The memory has one write port and multiple read ports, each pointing to a separate memory subsection. These subsections provide easy access of data by direct addressing of the read ports and is found particularly beneficial supporting multiple functions for on-the-fly reconfigurability.

The 9-b 64 deep data memory stores intermediate data. Restoring the state of a function requires nine data to be restored because of nine RUs present in the signal chain. This restricts the signal chain to a maximum of seven functions with the current 64 deep memory structure as executing seven functions one after the other would require 63 (= 9×7) data to be restored. The memory structure is shown in one of its many possible forms in Fig. 6.6(a), wherein it is divided into four subsections, each 16 deep. For one after the other function realization, the memory is divided into 9 deep seven subsections and one of the memory location remains unused as shown in Fig. 6.6(b). As an example, the data input to function1 (f_1) is stored in Location_0 of memory subsection I, denoted as <0,I>. This data is processed through the SAC architecture and the resulting output is stored at <0,II>. Function2 (f_2) uses <0,II> as input and its output is stored at <0,III>. Following the same sequence, output of the intermediate functions of the signal chain are stored in the first location (<0,*i*>, where *i* can be I, II,...., VII) of memory subsections as shown in Fig. 6.6(c) whereas the final result of the

signal chain, output7, can be propagated to a visual display or UART interface. On repeated parsing of signal chain the input data as well as intermediate outputs are stored in the subsequent locations in the respective memory subsection. Once the memory subsection is full, *i.e.* all the nine locations are occupied, the write pointers for the subsections are reset and the new data (or intermediate result) is now written starting from Location_0 in every subsection.

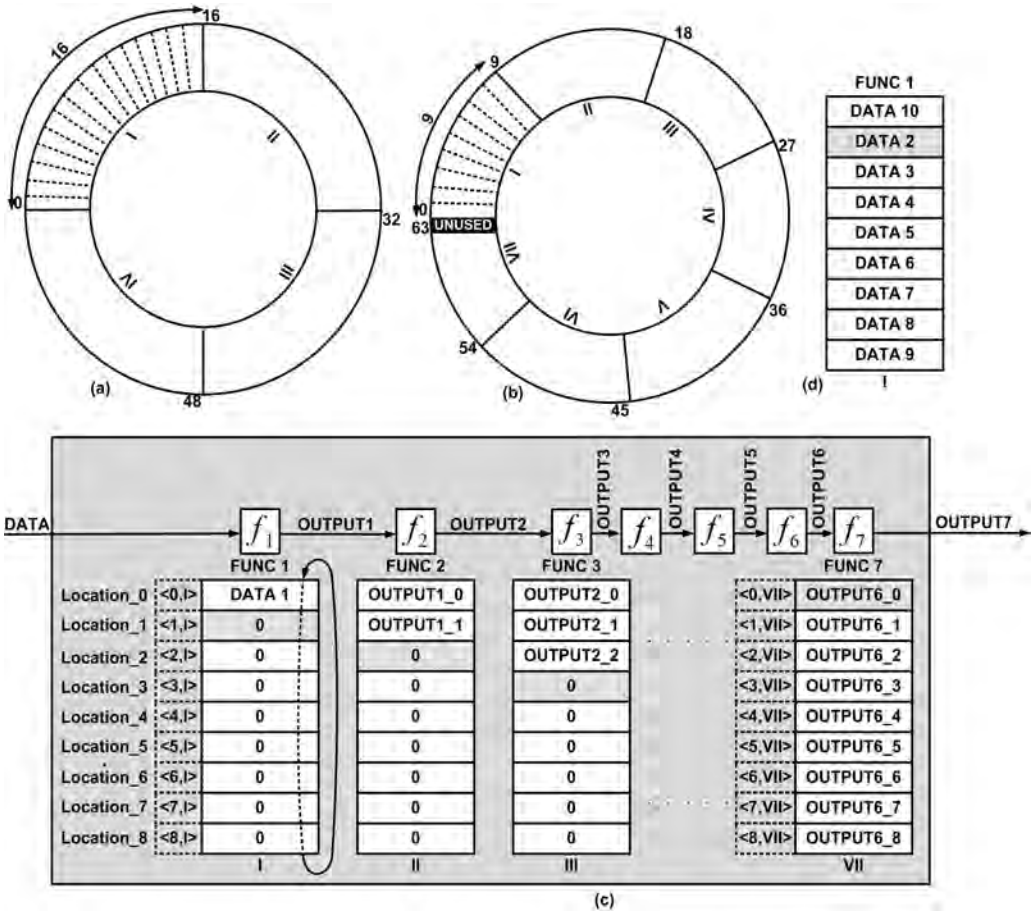


Figure 6.6: (a) A general circular memory structure (b) Memory subsections for on-the-fly reconfigurability (c) The state restore methodology in each memory subsection (d) Data wrap around on every 9^{th} traversal of signal chain.

6.4.2 State Restoring Methodology

The data restoring starts from the oldest (last) data in the function window due to the inherent data shifting nature of the architecture. For example, while parsing the signal chain for the first time, Location_0 holds the most recent data and data restoration starts from the oldest data which is held in Location_1. (hatched

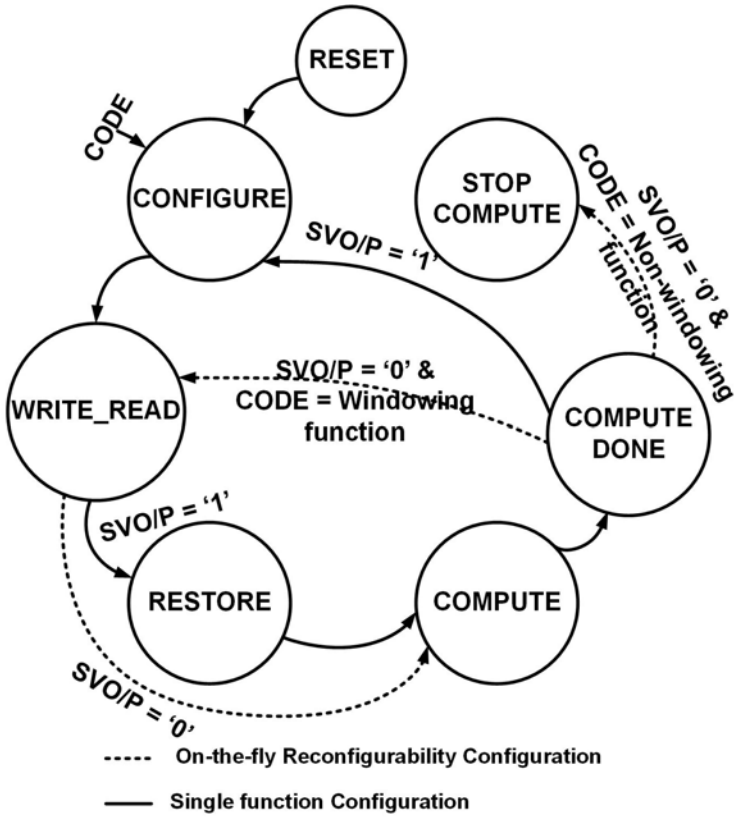


Figure 6.7: The function execution state machine.

block in memory subsection I shown in Fig. 6.6(c)) for all the functions, f_1 to f_7. In the second parsing, the input data is stored in Location_1 of memory subsection I and subsequently the output of intermediate functions (f_1 to f_6) are also stored in Location_1 of the memory subsections making it the most recent data and Location_2 the oldest. Therefore, restoring starts from Location_2 (hatched block in memory subsection II shown in Fig. 6.6(c)) during second parsing. It can be stated that for n^{th} parsing of the signal chain, the restoring starts from Location_i, where i is {(n+1) *mod* 9}. Therefore, the memory structure is designed to accommodate this regular updation of the restoring location with each parsing. The restoring address wraps around after every nine parsing and 'data1' is replaced by 'data10' as shown in Fig. 6.6(d), however, the restoring methodology remains the same.

6.4.3 Configuration Methodology

The state machine for the complete execution is shown in Fig. 6.7. Firstly, it enters the RESET state on reset for initialization to '0'. Then, it proceeds to the

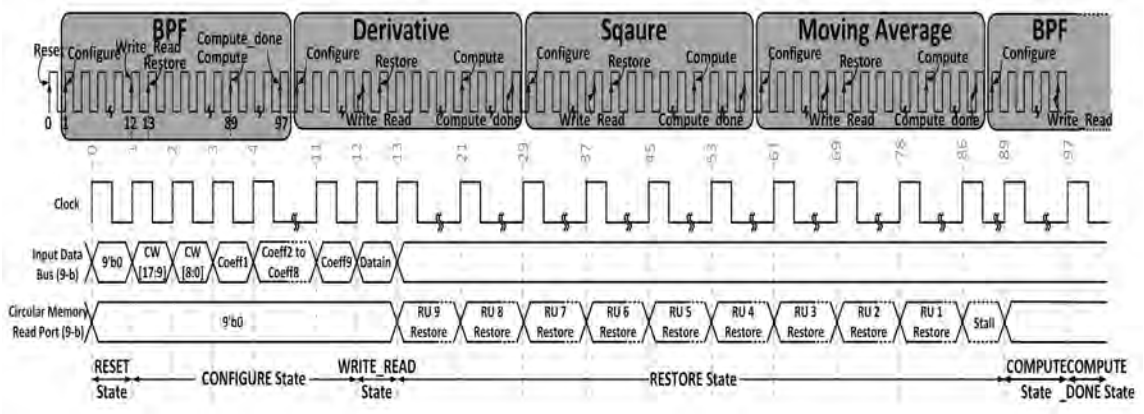


Figure 6.8: Timing diagram for QRS configuration.

CONFIGURE state where the control word and coefficients are loaded. In the following WRITE_READ state, valid data is written into (and later read from) the data memory depending on the information in the *code* field of the control word. In case of on-the-fly reconfiguration of the architecture, only one data is written into memory and reading is done in the RESTORE state. In the RESTORE state, the data is fed to the architecture according to the restore location as discussed in section 6.4.2. Once the state of the function is restored, the execution moves to COMPUTE for computing the result. The computed result is stored (in case of intermediate functions) or directed to appropriate output device (for the final output) in the COMPUTE_DONE state and the execution is again handed over to the CONFIGURE state. This configures the core for the next function in the signal chain. The execution follows the path denoted in solid line across the states in on-the-fly reconfiguration mode. However, in single function mode, the execution follows the path shown in dotted line in Fig. 6.7. When the architecture is configured only to perform one function and not chain of functions, the RESTORE state is skipped to proceed directly to COMPUTE state after the WRITE_READ state. This is because, reading (and supplying) data to the architecture is concluded in the WRITE_READ state itself. Additionally, once an output is computed in the COMPUTE_DONE state, the execution can proceed to either STOP_COMPUTE state or WRITE_READ state based on the processed function.

The execution stays in the STOP_COMPUTE state, unless reset in case of operations that generates result in one computation like squaring, multiplication,

addition and shift. However, while performing window operations like FIR and 2-D convolution that generally operate on stream of data, the execution moves to WRITE_READ state that provides data to architecture as well as capture input data samples.

QRS Configuration

The timing diagram for QRS configuration is shown in Fig. 6.8. Following the RESET state, the SAC architecture is configured for the BPF function in the CONFIGURE state. The *code* for BPF “00001" causes loading the BPF coefficients into coefficient registers of RUs. External input is written to the circular memory in WRITE_READ state and execution proceeds to RESTORE state on sensing the *SVO/P* bit. Output of BPF is computed in COMPUTE state once the architecture state is restored. The output is stored in circular memory sensing the on-the-fly reconfiguration mode. Further, the *code* field directs the execution to CONFIGURE state where the architecture is configured for the next function *i.e.* derivative. The aforesaid sequence of states is repeated for the complete QRS signal chain.

6.5 Results and Discussion

6.5.1 System overview

To demonstrate the feasibility of the proposed architecture, the design is targeted onto an Virtex-II FPGA board (XUPV2P). The operating frequency of FPGA implementation is found to be 46.9 MHz.Among various methods of initializing configuration data, the .coe (coefficient) file format is adopted in this work. For on-the-fly reconfiguration, a configuration is represented as a set of twelve 9-b data wherein first two data represent the control word, the following nine data represent the coefficients and last data denote the input sample to be processed which is loaded into the 9-b wide configuration memory generated using the in-built memory IP.

The control word, coefficient and data are distinguished and stored in their respective memories by means of three control signals generated in the CONFIG-

URE state of the state machine (Fig. 6.7). The SAC architecture generates 21-b output every eight clock cycles which is stored in the 8-b 36 deep output memory in slices of 8-b with three zero padded in the MSB side. Therefore, three 8-b data in the output memory denotes an output of the architecture and the memory can hold twelve such outputs. The data stored in output memory is sent to the computer using UART module developed following the RS-232 standard with a baud rate of 115200 bps. The UART is triggered to start transmission once the output memory is full and the execution in the architecture is stopped which is later resumed once UART is finished sending the data from output memory. Due to its relatively small size, both the output and data memories are implemented as synthesized register files.

Table 6.4: Range and Resolution of functions in PTA signal chain

Function	**Range ⟨Max,Min⟩**		**Required**
	Dec	**Hex**	**Resolution**
Normalized ECG I/P	⟨1,0.49⟩	⟨80,3F⟩	2^{-7}(=7.81e-3)
BPF O/P	⟨0.45,1.63e-2⟩	⟨E9,08⟩	2^{-9} (=1.95e-3)
Derivative O/P	⟨4.96e-2,1.2e-8⟩	⟨CB,00⟩	2^{-12} =(2.44e-4)
Squaring O/P	⟨2.5e-4,1.45e-16⟩	⟨A3,00⟩	2^{-16}(=1.52e-5)
MA O/P	⟨1.4e-4,1.96e-9⟩	-	-

Table 6.5: Resolution of PTA functions and their representation in the control word

Function	**Resolution**		**Resolution bits of**
	⟨Coefficient,Input⟩	**Final**	**Control_Word**
BPF	$\langle 2^{-11}, 2^{-7} \rangle$	2^{-18}	1001
Derivative	$\langle 2^{-10}, 2^{-9} \rangle$	2^{-19}	0111
Square	$\langle 2^{-12}, 2^{-12} \rangle$	2^{-24}	1000
MA	$\langle 2^{-11}, 2^{-16} \rangle$	2^{-27}	0011

6.5.2 QRS Detection

Range and Resolution of PTA functions

Storing the 21-b intermediate results in 9-b memory results in loss of accuracy due to limited resolution. This can be minimized by storing the nine most signif-

icant bits of the result located by identifying the range of functions (shown in Table 6.4). For example, the normalized ECG signal from the MIT/BIH arrhythmia database [55, 56] has values between 1 to 0.4943. As the usable peak information lies in the maximum value of the ECG signal, the resolution is chosen to accommodate the maximum value. This is because the minimum value is of secondary importance in case of QRS detection and represents only the S-peak information. The architecture employs saturated mathematics and clips values of signal smaller than the resolution to zero. Figure 6.9 (a-l) shows the MATLAB generated waveforms at the output of every function block in PTA chain computed in double and saturated mathematics. It can be observed that the usable peak information is retained in the waveforms obtained after saturated logic. Figures 6.9 (e,f,i,j) shows the zoomed sections of differentiation and squaring outputs, respectively about zero axis. The minor variation about the zero axis are flattened out in the outputs computed due to saturated maths for the reason discussed now. The data range for differentiation output listed in Table 6.4 is from 4.96e-2 to 1.2e-8. A precision of 2^{-5} is required to represent the upper limit 4.96e-2 *i.e.* the binary point is assumed four positions to the right of the MSB. However, as the four leading bits in representing the required range are always zero, the MSB can be assumed to represent 2^{-5} weight which leads to realizing a higher resolution with smaller number of bits. For instance, if we assume the number to be 8-b wide, a resolution of 2^{-12} as opposed to 2^{-8} is achieved with MSB weight equals to 2^{-5}. The MSB weights for the remianing intermediate outputs is assumed likewise according to the upper bound of the output range. However, it can be observed from Table 6.4 that while keeping the upper bound in range the lower bound of differentiation and squaring outputs which is smaller than and hence can not be represented correctly in the achievable resolution . This causes the values falling below the achievable resolution to be clipped to zero. The number of values clipped to zero for differentiation and squaring are 769 and 3079, respectively.

Resolution of PTA functions cannot be predicted apriori as it largely depends on function under execution as well as the input data set. This is solved by sending the resolution to the architecture as a part of the control word. The resolution

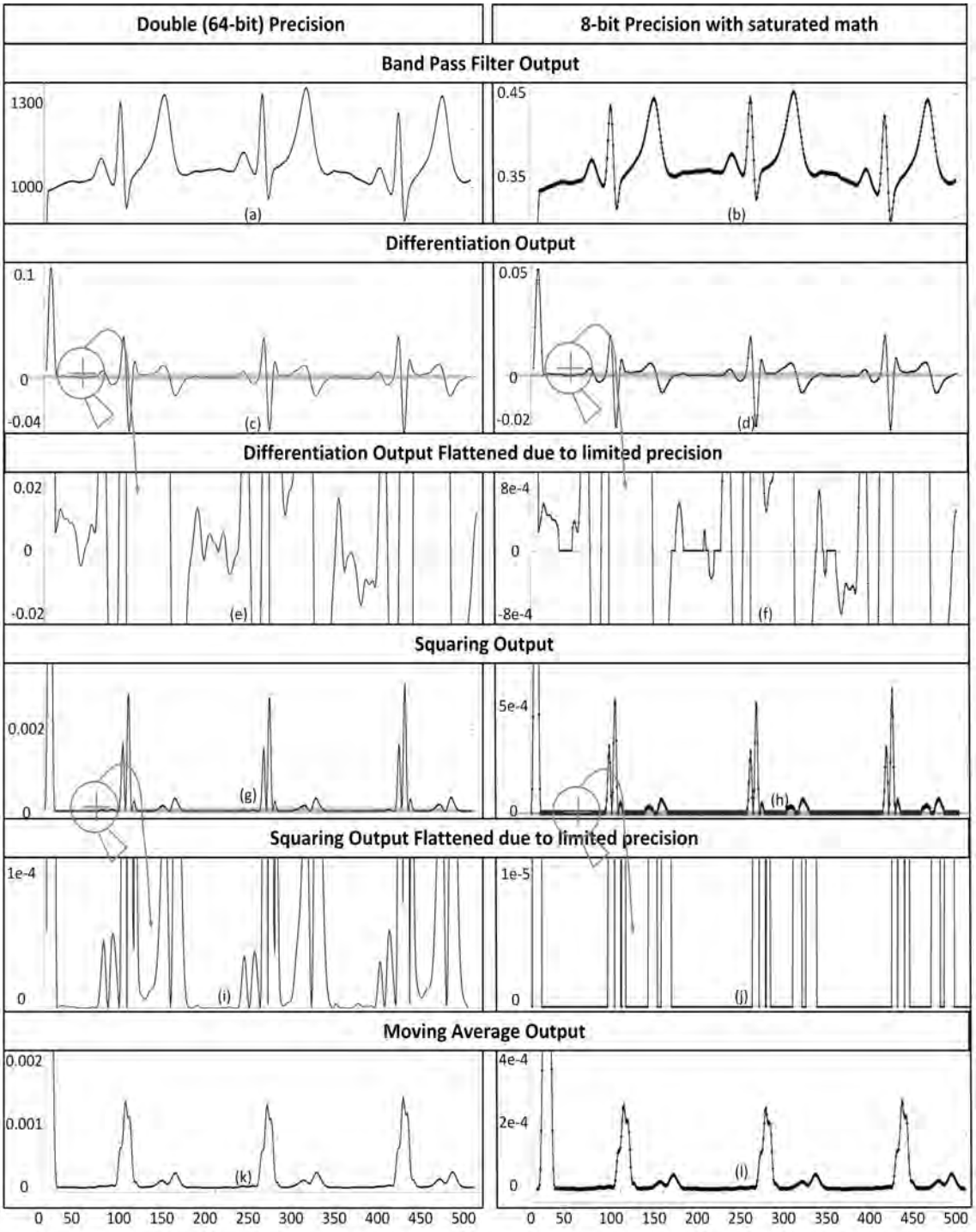


Figure 6.9: The MATLAB computed results of each PTA block in double and 8-b precision. (a,b) BPF. (c,d) Differentiation. (e,f) Differentiation zoomed about zero line. (g,h) Squaring. (i,j) Squaring zoomed about zero line. (k,l) Moving average.

mentioned in control word decides which section of the 21-b output is to be retained. For example, if the mentioned resolution is "0000", then the last eight bit section *i.e.* bit 0 to bit 7 along with the sign bit is retained. Table 6.5 presents the

data and coefficient resolution PTA functions along with resolution bits of the control word. The final resolution indicates the resolution of 21-b outcome generated as the result of multiplying data and coefficient. For instance, the BPF function has resolution 2^{-18} *i.e.* in the 21-b output, three bits (starting from the MSB) indicates whole number part whereas remaining bits indicate fractional part of the result. On analyzing output of PTA in MATLAB, we figured that only a resolution of 2^{-9} is required to represent the entire range of BPF output (see Table 6.4). Therefore, we only store the relevant section *i.e.* bits 2^{-2} to 2^{-9} or 16^{th} bit to 9^{th} bit. Hence the bits indicating resolution in the control word are set to "1001".

Mapping on SAC Architecture

The *code* and *config* fields of control word for the functions in the PTA is shown in Table 6.6. The BPF, derivative and moving average functions decompose into the fundamental multiply-accumulate operation using all the nine taps, and hence have the same control word. The save output bit (*SVO/P*) is set to save output of current computation for subsequent use and *fb1* bit is reset as feedback does not exist in PTA transfer functions.

Table 6.6: Control word of the PTA based QRS detection signal chain

Function	**Control Word**							
	Code	**Config**						
		CM1	CM4	BP4	CM7	BP7	SVO/P	fb1
BPF Derivative MA	00001	0	1	1	1	1	1	0
Square	00111	0	0	0	0	0	1	0

PTA Implementation - Output obtained from MATLAB and FPGA

Under PTA, the ECG signal is passed through a band pass filter followed by slope detection, slope accentuation and running integration. The PTA is executed both in MATLAB and hardware and the stage wise results are presented in Fig. 6.10. The moving average output is plotted in Figure 6.11 separately for direct comparison with MATLAB output. The dashed line and the triangular points indicate

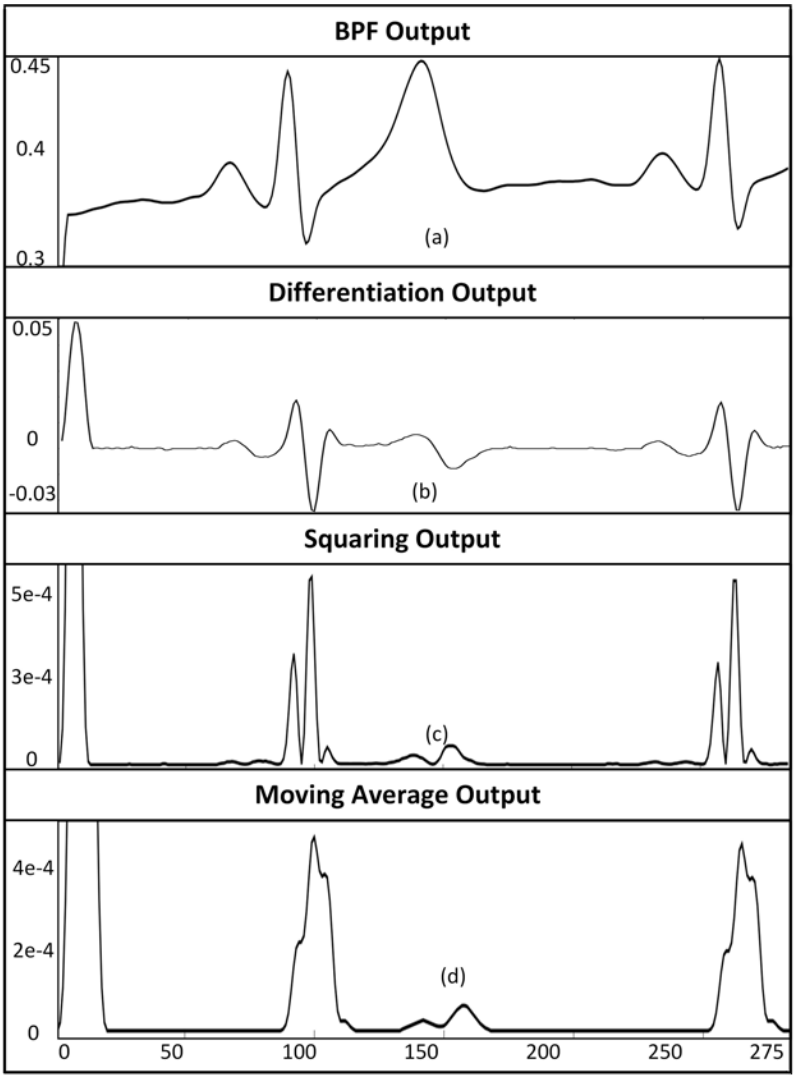


Figure 6.10: Hardware computed PTA results (a) BPF. (b) Differentiation. (c) Squaring (d) Moving Average.

MATLAB and hardware computed results, respectively, when output at intermediate stages is saturated (or clipped). It can be seen that both the waveforms overlap completely. The output has an initial peak due to the dc component present in the ECG signal and is of secondary importance while detecting QRS complex. The other two peaks, occurring at sample #100 and #155, represents the integrated QRS and T peaks. On applying the peak detection algorithm on the moving integrated results, the QRS peak (precisely the R-peak) being the maximum (ignoring the dc component peak) gets easily detected.

The solid line in Fig. 6.11 indicates the moving average output computed by MATLAB without output saturation (or clipping) *i.e.* the entire 21-b output

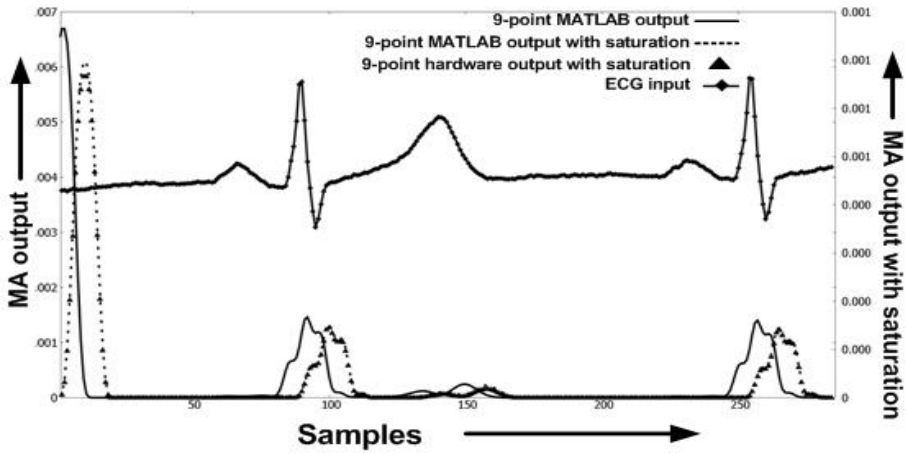


Figure 6.11: The comparison of MATLAB and hardware computed results with and without bit truncation along with the ECG signal.

is used for computation. The unsaturated result is observed to be scaled and slightly shifted in samples when compared with the saturated result. As both the waveforms retain the same shape, scale-up or sample shifting hardly affects the detection of QRS complex.

Table 6.7: Clock cycle consumed in Configuration and Computation of the Architecture

Reset	1
Configure	11
Write_Read	1
Restore	75
Compute	8
Compute_done	1
Total cycles consumed in executing one function = 97	
Total cycles consumed in executing seven functions = 97+6$\times$96 = 673	

Clock Profiling

We now discuss, the number of clock cycles that is taken while executing the state machine shown in Fig. 6.7. Once the configuration memory is written, the code execution reaches CONFIGURE state after RESET. In this state, all the necessary data to initiate computation *i.e.* control word, coefficients and input data is loaded through a 9-b input bus. The control word is 18-b wide and requires two cycles for loading. Following the control word, nine 9-b coefficients are loaded that takes nine clock cycles. This concludes the configuration in a total of 11 clock cycles and

execution moves forward to the WRITE_READ state. The WRITE_READ state takes one clock cycle in case of on-the-fly reconfiguration and execution advances to the RESTORE state. The previous inputs or intermediate outputs are restored in the RUs from the circular memory in this state. The data is restored sequentially and takes eight clock cycles to move from one RU to another. Therefore, it requires 72 (= 9×8) cycles to restore nine data. The execution stays in this state for another 3 cycles and generates the load and control signals for the coefficient and data necessary for synchronized results generation before proceeding to COMPUTE state which takes 8 clock cycles while performing multiplication by means of serial generation of partial products.

Once the result is computed, it is stored in the circular memory in the COMPUTE_DONE state and requires one clock cycle. The execution again moves to the CONFIGURE state for subsequent functions configuration and computation. The clock cycles consumed in traversing the signal chain of seven functions are presented Table 6.7 along with state wise cycle bifurcation.

Gate Count

The number of gates of the SAC architecture is estimated as 3192 gates and are presented in Table 6.8. The adder tree consists of ripple carry adders (RCA) equivalent to 123 1-b full adders. In addition to the gates in SAC architecture, 36×8-b output memory elements, 64×9-b circular memory elements and 18 control word memory elements are present as part of memory interfaces.

Table 6.8: Gate Count of the Architecture

		AND	XOR	OR	NOT	Reg-Gates
RU		26	9	9	9	189
RU X 9		234	81	81	81	1701
CU	Adder Tree	246	246	123	-	-
	21-b Adder	42	42	21	-	-
	Sum_Acc Reg	-	-	-	-	147
	Output Reg	-	-	-	-	147
TOTAL = 522 + 369 + 225 + 81 + 1995 = 3192						

6.5.3 Related work

[177–183] discuss various approaches to detect QRS complex and its implementation on hardware. Among them [177,179,181] adopt PTA based QRS detection. A median based thresholding is further used in [177] for PTA. Other reported QRS detection techniques include combination of morphological filtering and wavelet transform [178,180]. A slope based thresholding [182] and adaptive lifting scheme is discussed in [183]. The comparison of QRS detection methodology adopted in this thesis with aforementioned literature is sin Table 6.9. It should be noted that apart from [181], none of the architectures report reconfigurability, whihown ch limits the utility of these algorithms to QRS detection only. However, [181] allows patient-specific multi-parameter monitoring in blood pressure, heart rate and ECG.

The SAC architecture supports on-the-fly reconfigurablity thus allowing various BSN signal processing algorithm realizations and reports notably higher cell utilization as compared to [180,184]. However, when compared with system supporting reconfigurability [181] it uses $\approx 4\times$ less logic cells [181]. The proposed architecture reports the highest operating frequency of 46.9 MHz (1.3-2.3$\times$ higher) and power consumption of 568μW/MHz including the memory overheads much lower than [183]. This is because, the architecture is active for a small proportion of task and remains idle otherwise. Therefore, the overall power consumption comes down due to the dominance of static power throughout the operation. The power consumption reported in [181] does not include the memory overheads.

Table 6.9: Comparison with State-of-the-art

	Principle adopted	Operating Frequency (in MHz)	Architectural Configurability	FPGA	Logic Utilization, Logic Cells	Power (in W)
[177]	PTA+ Median thresholding	-	No	Spartan XC3S500E	76%,7962	-
[184]	Entropy-based	-	No	Cyclone EP1C6Q240C8	98%, 5865	-
[178]	Morphological filtering + QSWT + MMPR	35.23	No	Virtex-4 SX35	95%,33064	-
[179]	PTA	-	No	Virtex-4 LX25	54%,13000	-
[180]	Integer wavelet transform	27.23	No	Cyclone EP1C12Q240	11%,651	-
[185]	PTA	-	No	Spartan XC3S500E	32%,3352	-
[181]	Multi-parameter monitoring	-	Yes	Virtex-5 FX70	25.23%,82500	300μ
[183]	Adaptive lifting scheme	20	No	Virtex-2 XC2VP30	8%,2465	2.5mW/MHz
This thesis	PTA	46.9	Yes	Virtex-2 XC2VP30	72%,22188	568μW/MHz

6.6 Circuit Implementation

The circuit level implementation of the SAC architecture is attempted in Virtuoso© for energy estimation of the platform. This is done by importing the structural VHDL code of the architecture (including the memories and state machine) in Virtuoso©. In addition to energy, an accurate estimate of gates of the system, consisting of SAC architecture, state machine, memory interfaces and configurable logic, is obtained from the structural code which was not achievable in the FPGA implementation attempted earlier because of the code optimizations and automatic selection of logic cells by the compiler.

Table 6.10: Basic gates characterized in terms of delay, power and energy at three different (1.8V, 0.7V and 0.5V) V_{DD}

Gate	t_{hl}(in ps)	t_{lh}(in ps)	Power	Energy
		V_{DD} = 1.8V		
Inverter	12.185	15.573	71.9p	431.4f
Nand2	15.69	14.71	153.4p	1.24p
And2	37.247	44.307	195.6p	1.6p
Or2	58.53	53.88	330.6p	2.536p
Xor2	119.01	57.44	554p	4.452p
Nand3	17.888	14.685	531.2p	4.249p
And3	68.568	112.84	628.5p	5.081p
DFF	389.024	593.684	6.368u	210.1n
		V_{DD} = 0.7V		
Inverter	42.095	130.88	50.05n	200.2p
Nand2	99.4	115.279	14.96n	134.4p
And2	279.783	366.12	14.96n	134.5pp
Or2	507.353	346.74	14.94n	134.3p
Xor2	1008.76	467.75	14.96n	134.5p
Nand3	161.53	114.166	11.26n	135.1p
And3	526.554	985.88	11.27n	135.2p
DFF	18.597e6	3.110e3	164.2p	5.252p
		V_{DD} = 0.5V		
Inverter	178.5	1292.2	6.012p	72.15f
Nand2	671.2	1138.9	3.105p	37.26f
And2	2501.4	2853.6	6.912p	82.95f
Or2	4488.5	2681.9	15.11p	181.3f
Xor2	8338.7	3802.8	14.42p	173.1f
Nand3	1404.1	1123.14	1.843p	22.12f
And3	4529.85	8065.8	11.82p	141.8f
DFF	56.39e6	7.42e3	45.85p	1.467p

The structural VHDL code consists of certain basic gates whereas other gates/blocks are derived from these gates. The basic gates used in the SAC architecture are designed in CMOS logic using 0.18μm UMC technology and are characterized for delay, power and energy at three different V_{DD} voltages presented in Table 6.10. It can be seen that the delay increases as V_{DD} decreases causing an increase in power and energy dissipation. However, further scaling down the V_{DD} voltage to the near threshold regime causes the energy to decrease because decrease in supply voltage supersedes the effect of increased delay.

The SAC architecture is demonstrated to emulate functions listed in Table 6.11 using the 18-b control word. The power consumed by the architecture while emulating various functions is also provided in the table at the operating voltage and frequency of 0.5V and 1MHz, respectively. The SAC architecture undergoes many states starting from getting configured to computing the emulated function and finally storing the result in memory. The state-wise execution is discussed in subsection 6.4.3. The state-wise energy dissipation is provided in Fig. 6.12.

Table 6.11: The Control Word and Energy Consumption (0.5V, 1MHz) of target functions.

Function	***Code***	***Config***	**Power (in μW)**
9-tap FIR filter	00001	0111100	3.369
MAC2	00010	0010000	3.774
ADD2	00011	0010000	9.44
MAC3	00100	0010100	3.681
ADD3	00101	0010100	10.92
Multiply	00110	0000000	9.967
Square	00111	0000000	9.967
Shift Left	01000	0000000	5.732
Shift Right	01001	0000000	5.732
3×3 Convolution	01010	0010100	8.846

The configuration phase includes reset, datapath structuring and data exchange between architecture and data memory. Furthermore, in case of on-the-fly reconfigurability, the configuration phase also includes the state restoring state. From the energy breakup it is evident that the configuration phase is the major energy consumer. This is because it programmes the architecture by activating various components involved in the process such as datapath multiplexers, circular mem-

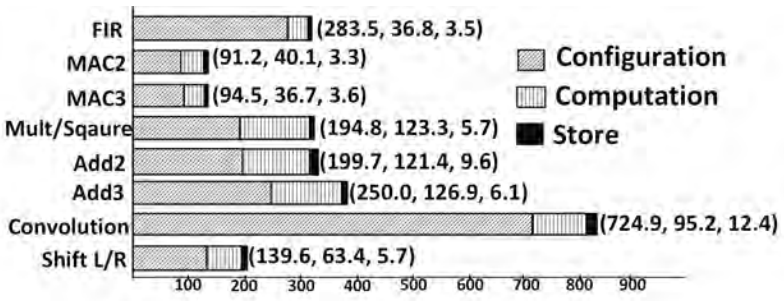


Figure 6.12: The state-wise energy dissipation for target functions.

ory *etc*. Furthermore, it can be observed that windowing functions consume more configuration energy as compared to the non-windowing functions. This is because in case of windowing functions more number of data is written into circular memory. The number of active components on the datapath contributes to the configuration as well as the computation energy. Greater number of active RUs result in increased switching activity in the CU leading to increased computation energy.

Table 6.12: The cycle counts of target functions.

Function	**Cycle Count s(Fixed[#]+ WRITE_READ+STOP_COMPUTE)**	
	Single Function Mode	On-the-fly Reconfiguration Mode
FIR, Convolution	22+64+0	158
MAC2, ADD2	22+2+1	97
MAC3, ADD3	22+3+1	98
Multiply, Sqaure, ShiftL, ShiftR	22+1+1	96

Fixed = Reset (1) + Configure (12) + Compute (8) + Store (1)
On-the-fly Reconfiguration adds another 72 cycles to cycle count.

The function wise cycle profile is provided in Table 6.12. The architecture takes 25-159 cycles based on implemented function. The range of the cycle count accounts for the different data requirements of the target functions. In case of windowing functions, entire circular memory is written with data in WRITE_READ (64 cycles) whereas variable number of data is written for non-windowing functions. The cycles consumed while writing data into the circular memory is provided under the WRITE_READ head in the table. Every function undergoes RESET(1 cycle), CONFIGURE (12 cycles), COMPUTE (8 cycles) and COMPUTE_DONE (1 cycle) states when mapped on the architecture, therefore the combined cycles of these states are provided as fixed cycles in Table 6.12. Additionally, on-the-fly

reconfiguration mode adds another 72 cycles to the cycle count on account of the RESTORE state.

The PTA based QRS detection application is mapped on the SAC architecture, wherein the architecture configures itself as Band Pass Filter (BPF), differentiator, squaring circuit and moving average circuit on-the-fly. The gates accounting for computational logic (RU + CU), memory interface, configuration and reconfiguration are presented in Table 6.13. The datapath multiplexers are grouped in the configuration head whereas the state machine gate count enabling on-the-fly reconfiguration is included under reconfiguration head. The pie chart in Fig. 6.13 represents the percentage breakup of major components of the architecture and its supporting logic. The computational logic amounts for 16% of the total gate count and the remaining 84% accounts for the supporting logic among which the reconfiguration state machine contributes 61% of gates. The increased share of reconfiguration logic is due to varied write and read address logic resulting from the varied memory interpretation mechanisms arising out of target functions.

Table 6.13: Gate count breakup of the architecture along with its interfaces and supporting state machine.

Component	**# of Gates**
9*RU + CU	3192
Memory	4032
Configuration	432
Reconfiguration	12018
Total = 19694	

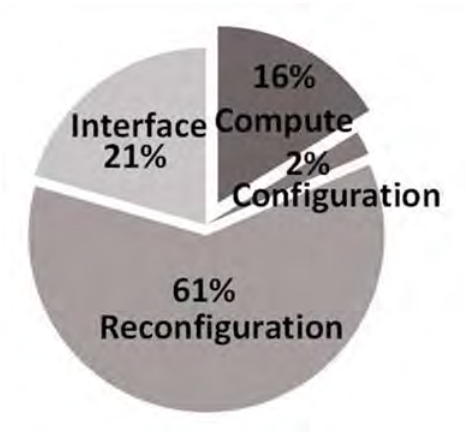


Figure 6.13: Percentage gate breakup of major components of the architecture.

The SAC architecture consumes 4.19 nJ when operated at 0.5V and 1MHz while realizing PTA based QRS detection in which it gets reconfigured four times. The percentage energy breakup averaged over four reconfigurations is provided in pie chart of Fig. 6.14. The configuration phase emerged as the major energy contributor because of the memory hits caused during data exchanges between memory and architecture. The compute energy accounts for the energy consumed during partial product generation and accumulation.

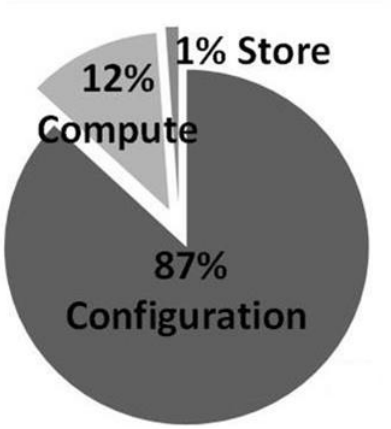


Figure 6.14: Percentage energy breakup of the architecture averaged over four reconfigurations.

Literature dedicated to biomedical application processor includes [21, 22, 77] and each of them use a different hardware architecture to realize the ECG processing application. A dedicated ASIC is used to implement ECG processor based on PTA in [21] whereas [77] uses a low power general purpose processor, CoolFlux BSP for detecting the QRS complex based on continuous wavelet transform (CWT). A multiple hardware accelerators approach is employed in [22] wherein the low power microprocessor offloads computationally intensive tasks on the accelerators. The accelerators are chosen in such a way that a combination of operations performed by them spans majority of the biomedical signal processing applications. All the three architectures achieve low energy results owing to near/sub threshold operation and multiple power domains. Among them, [22] and [21] reports energy per sample in nJ range where [77] reports dissipation in μJ range as well as has the highest gate count owing to the general instruction set architecture of CoolFlux BSP. The dedicated ASIC implementation in [21] reports the

lowest gate count among three at the cost of flexibility. However, the SAC architecture, because of it ability to be reconfigured on-the-fly, offers a single platform serving a multitude of functions encountered in the ECG application and thus reports the lowest gate count among [21, 22, 77]. Additionally, it provides a flexible platform allowing algorithm improvements and optimization due to its coarse grained structure. The energy per sample reported is comparable to [22] because of near-threshold operation and supports data rates upto 1 MHz. Further reduction in energy can be achieved by defining different power domains in the architecture wherein the computation section of the architecture can be powered down during configuration phase and vice versa. However, these schemes are not attempted in this thesis.

Table 6.14: Comparison with state-of-the-art.

Process Technology	180nm CMOS full custom			
Logic	19694			
Voltage	0.5V			
Energy/Cycle/sample	4.19fJ			
Comparison with State-of-the-art				
	[22]	[77]	[21]	This Work
Application	EEG, ECG	ECG	ECG	ECG
Processor Core	H/W accelerator	CoolFlux BSP	Custom ECG μP	Configurable accelerator
Technology (in nm)	130	90	45	180
V (in V),f(in MHz)	0.7,-	0.4,1	0.34,0.6	0.5,1
Energy/sample (in J)	37.6n, 53.05n	47μ	312n	4.19n
Gates (in k)	81.3	195	37	19.6

6.7 Conclusion

The SAC architecture is coupled with the on-the-fly reconfigurability in this chapter. This was explored due to the function-chain like structure of biomedical signal processing applications. The SAC architecture could support on-the-fly reconfigurability because it was demonstrated to emulate individual functions in previous chapter at performance sufficiently large for reconfiguration while maintaining pace with the input datarates of biomedical signal processing applications. Fur-

thermore, on-the-fly reconfigurability is incorporated with the architecture while keeping its interfaces unchanged by introducing a memory management methodology. The results for functions appearing in the midst of the function-chain are stored inside the memory following this methodology demonstrating adherence to the light-weight nature of the architecture. The state machine controlling the execution of functions in the on-the-fly reconfigurable manner contains configuration, restore, compute, store states. The status of the function is captured and later restored in the restore state beneficial particularly in windowing functions. A modified QRS detection algorithm is mapped on the architecture realized on FPGA for proof-of-concept. Additionally, the architecture is characterized for energy, gate count and cycle count in the circuit level implementation. The architecture is compared on the basis of gate count and energy with other hardware implementations realizing QRS detection. The architecture offers comparable energy and ≈ 10 times reduced gate count in comparison to other literature.

CHAPTER 7

Conclusion and Future Work

A light-weight and reconfigurable architecture is developed in this thesis targeting the biomedical signal processing application domain. The fundamental operation of the developed architecture is a shift-accumulation which results in a low gate count or light-weight platform primarily because of elimination of multipliers. The architecture has the capability to perform various digital signal processing functions by means of configurable datapath. Additionally, the architecture offers flexibility (coarse-grained) and various topologies for realizing functions and applications unaccounted for during design time. The mapping methodologies for target functions are developed considering optimized resource and cycle utilization thereby complying to the light-weight nature of the architecture. Due to the innate low performance nature of biosignal datarates, the on-the-fly reconfigurability aspect strikes an optimal balance between energy and throughput. The hardware realization of the architecture with various target functions mapped confirms functional integrity. As a proof of concept, the architecture is realized on the Field Programmable Gate Array (FPGA) with energy and cycle quantifications in the circuit level simulations. The architecture is demonstrated with widely used biomedical application, QRS Detection in ECG signal, utilizing the on-the-fly reconfigurability feature. In the process of developing the configurable platform, several supporting interfaces are also implemented. The architecture is compared on the basis of gate count and energy with other hardware implementations realizing QRS detection. The architecture offers comparable energy and ≈ 10 times reduced gate count in comparison with other literature. Furthermore, additional topology of multiple 9-FU SAC units connected together is analyzed

and demonstrated by mapping 8×8 DCT. These units perform individual computations and exchanges data among each other using a control unit. Additionally, the control unit can selectively power down/up a SAC unit once its computations finishes/commences irrespective of state of other units.

7.1 Summary of Contributions

This thesis focuses on low-gate count reconfigurable architectures development with efficient mapping methodologies for biomedical signal processing, targeting ambulatory medical monitoring as the end application. Details of the contributions are summarized below.

Architecture development

- The developed SAC architecture realizes multiply-accumulate, the fundamental operation in majority of biomedical signal processing functions, by serial computation through the process of shift-accumulation. The configurable datapath coupled with the multiplier-less and serial implementation results in a light-weight configurable architecture having low area cost and utility across applications specifically in domains where high performance is not a pre-requisite.

- The thirty-six functional unit (FU) of the developed architecture has simplistic design and small footprint as compared to other coarse-grained FUs. The FUs are connected in a systolic array structure through configurable datapath that has the ability to disconnect/bypass the unused FUs leading to a hardware-optimized implementation.

- The datapath enables realizing multiple topologies on the architecture for easy emulation of varied functions/algorithms. The topologies also support simultaneous realization of multiple functions by exploiting parallelization in an algorithm thereby increasing throughput.

- The architecture exhibits hardware configuration by means of a control word and does not explicitly require a dedicated compiler/scheduler/synchronizer.

- A light weight unified data memory is proposed with structure adaptable to the varied requirements of target functions/algorithms. The circular memory architecture provides an added advantage in case of 2-D DWT and 2-D DCT wherein the data storage technique results into simplistic data fetching.

Mapping Methodologies

- The mapping methodologies were developed with due consideration to exploiting the regularity of target algorithms, clock-efficient execution, reduced memory accesses through storage of intermediate results within architecture, which collectively resulted in an hardware-efficient realization of target functions/algorithms.

- Elimination of redundant computations reduced the associated computational cost leading to a cycle and energy-efficient realization.

- The target algorithms were mapped using small state machines that force the necessary internally stored data on the FUs which ensures reduced stalling of the architecture and further improves the cycle count of computations.

- The feature of storing intermediate data within the architecture provides easy access to this data for subsequent operations.

- The target DSP functions are mapped on the architecture by means of a control word that acts as function identifier, defines datapath and restructure the memory peripherals. The fields of control word can be configured independently which gives rise to numerous architecture topologies and thus enables hardware efficient realization of target functions

On-the-fly Reconfigurability

- On-the-fly reconfiguration favours concatenated algorithm and ensures increased usage of the hardware but at the cost of reconfiguration overhead.

- The scaled SAC architecture with nine FUs (9-FU SAC) is demonstrated to exhibit on-the-fly reconfiguration by means of a state machine.

- Context switching is attained by sectioning the circular memory which supports reconfiguration upto 7 times in its present structure. This approach utilized the available memory efficiently resulting in a cycle and area efficient implementation.

- The biomedical application of QRS detection, is demonstrated using the on-the-fly reconfiguration feature.

- Furthermore, the circuit level implementation of the architecture is attempted to estimate the energy of the target QRS detection application. The custom library cells are analyzed at different voltage levels to examine the variation of energy with voltage scaling. The architecture is compared on the basis of gate count and energy with other hardware implementations realizing QRS detection.

7.2 Modularity of the SAC Architecture

The (3×3) SAC architecture with reconfigurability feature targets functions that appear predominantly in digital/biomedical signal processing applications but are not sufficient to emulate the entire application on the architecture. Additionally, the span of target applications gets limited as major functions performing vital tasks like feature extraction are not included in the list of target functions. These left out functions are generally computation intensive and include (but are not limited to) CORDIC, DCT/DFT, FFT *etc.* The (3×3) SAC architecture has the capability to emulate these functions but may require multiple stages of configuration while doing so. Two mapping schemes are presented in Appendix C. The mapping scheme for DCT takes 64×4 cycles to compute intermediate matrix and additional 80 cycles for DCT results amounting to 336 cycles per 8×8 image block. The cycle efficient realization is obtained at the cost of 4× increase in hardware as compared to single (3×3) SAC unit emulation. Nevertheless, the simplified control structure developed for the scheme along with the activation/deactivation of various components makes it easier to comprehend. Mapping methodologies for

other computation functions such as CORDIC, DFT, FFT can be developed similarly.

7.3 Future Work

This thesis focussed on the signal processing architecture for the biomedical applications. However, clearly there are many interesting studies that can be pursued further. Some of the directions for future work are mentioned below.

Approximate Computations

Approximate computing has emerged as a paradigm for enabling energy efficent hardware software implementations by exploiting the inherent resilience of the applications to impreciseness in their underlying computations. This effectively leads to trading off accuracy for better performance and energy efficiency. In recent years, this has turned out to be an attractive approach for many resource-frugal applications. The biomedical signal processing can benefit from the energy efficiency ensured by approximate computing, but its applicability in the said domain is yet to be explored. In hardware, the techniques of simplifying/modifying the design at various levels of abstraction, dynamically reducing the precision of the computations *etc* constitute the recent trend.

Multiple Power Domains

The SAC architecture adopts hardware optimized approach through bypassing unused FUs coupled with low overall gate count and leverage energy benefits as a spinoff. However, this leads to little gain in the energy savings. Dedicated power domains have been proposed in literature [77, 100] yielding many folds energy advantages by means of operating different components at different supply voltages. A control scheme can be developed for the SAC architecture at system level which applies multiple power domains to various components currently working on the same supply voltage.

CORDIC Architectures

The radix-2 CORDIC architecture is emulated in this thesis because of the simple decision scheme it has for the rotation direction determination. However, there

exists higher radix CORDIC algorithms reported for increased throughput and resolution along with reduced computation time. Novel methodologies can be developed to express the complex decision making and shifting logic in a light weight and energy aware manner.

References

[1] J. H. Nagel, "Biopotential Amplifiers," in *The Biomedical Engineering Handbook: Medical Devices and Systems* (J. D. Bronzino, ed.), pp. 1–14, Boca Raton: CRC Press LLC, 2000.

[2] J. Pan and W. J. Tompkins, "A real-time QRS Detection Algorithm.," *IEEE Transactions on Biomedical Engineering*, vol. BME-32, no. 3, pp. 230–236, March 1985.

[3] V. Pop, R. de Francisco, H. Pflug, J. Santana, H. Visser, R. Vullers, H. de Groot, and B. Gyselinckx, "Human++: Wireless Autonomous Sensor Technology for Body Area Networks," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 561–566, January 2011.

[4] A. Shoeb, H. Edwards, J. Connolly, B. Bourgeois, S. T. Treves, and J. Guttag, "Patient-Specific Seizure Onset Detection," *Epilepsy and Behavior*, vol. 5, no. 4, pp. 483 – 498, August 2004.

[5] B. S. Emmanuel, "A Review of Signal Processing Techniques for Heart Sound Analysis in Clinical Diagnosis," *Journal of Medical Engineering & Technology*, vol. 36, no. 6, pp. 303–307, July 2012.

[6] A Single-Chip Pulsoximeter Design Using the MSP430. [Online]. Available: http:// www.ti.com/litv/pdf/slaa274.

[7] K. Sembulingam and P. Sembulingam, *Essentials of Medical Physiology*. Jaypee Brothers Medical Publishers, 5th ed., 2010.

[8] J. A. Staessen, L. Thijs, R. Fagard, E. T. O'brien, D. Clement, P. W. de Leeuw, G. Mancia, C. Nachev, P. Palatini, G. Parati, *et al.*, "Predicting Cardiovas-

cular Risk using Conventional vs Ambulatory Blood Pressure in Older Patients with Systolic Hypertension," *JAMA*, vol. 282, no. 6, pp. 539–546, August 1999.

[9] T. W. Hansen, M. Kikuya, L. Thijs, K. Björklund-Bodegård, T. Kuznetsova, T. Ohkubo, T. Richart, C. Torp-Pedersen, L. Lind, J. Jeppesen, *et al.*, "Prognostic Superiority of Daytime Ambulatory over Conventional Blood Pressure in Four Populations: A Meta-Analysis of 7030 Individuals," *Journal of hypertension*, vol. 25, no. 8, pp. 1554–1564, August 2007.

[10] A. Hara, K. Tanaka, T. Ohkubo, T. Kondo, M. Kikuya, H. Metoki, T. Hashimoto, M. Satoh, R. Inoue, K. Asayama, *et al.*, "Ambulatory Versus Home Versus Clinic Blood Pressure - The Association with Subclinical Cerebrovascular Diseases: The Ohasama Study," *Journal of Hypertension*, vol. 59, no. 1, pp. 22–28, January 2011.

[11] M. E. Ernst and G. R. Bergus, "Favorable Patient Acceptance of Ambulatory Blood Pressure Monitoring in a Primary Care Setting in the United States: A Cross-sectional Survey," *BMC family practice*, vol. 4, no. 1, pp. 1–6, October 2003.

[12] Y.-N. Kim, D. G. Shin, S. Park, and C. H. Lee, "Randomized Clinical Trial to assess the Effectiveness of Remote Patient Monitoring and Physician Care in reducing Office Blood Pressure," *Hypertension Research*, vol. 38, no. 7, pp. 491–497, July 2015.

[13] N. J. Holter and W. R. Glasscock, "3,215,136: Electrocardiographic means."

[14] H. Reinhold Jr. and A. A. Auerback, "4,531,527: Ambulatory Monitoring System with Real-Time Analysis and Telephone Transmission."

[15] J. Chauhan and S. Bojewar, "Sensor networks based healthcare monitoring system," in *International Conference on Inventive Computation Technologies (ICICT)*, vol. 2, pp. 1–6, August 2016.

[16] J. Wan, M. A. A. H. Al-awlaqi, M. Li, M. O'Grady, X. Gu, J. Wang, and N. Cao, "Wearable IoT enabled real-time Health Monitoring System," *EURASIP Journal on Wireless Communications and Networking*, vol. 298, no. 1, pp. 298–307, 2018.

[17] M. Magno and D. Boyle, "Wearable Energy Harvesting: From Body to Battery," in *International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*, pp. 1–6, April 2017.

[18] R. Kanan and R. Bensalem, "Energy Harvesting for Wearable Wireless Health Care Systems," in *IEEE Wireless Communications and Networking Conference*, pp. 1–6, April 2016.

[19] D. Reda El-Damak, *Power management circuits for ultra-low power systems*. PhD thesis, Massachusetts Institute of Technology (MIT), Massachusetts, June 2015.

[20] M. Alhawari, T. Tekeste, B. Mohammad, H. Saleh, and M. Ismail, "Power Management Unit for Multi-Source Energy Harvesting in Wearable Electronics," in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1–4, October 2016.

[21] R. A. Abdallah and N. R. Shanbhag, "An Energy- Efficient ECG Processor in 45-*nm* CMOS using Statistical Error Compensation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 11, pp. 2882–2893, November 2013.

[22] J. Kwong and A. P. Chandrakasan, "An Energy- Efficient Biomedical Signal Processing Platform," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 7, pp. 1742–1753, July 2011.

[23] N. Verma, A. Shoeb, J. Bohorquez, J. Dawson, J. Guttag, and A. P. Chandrakasan, "A Micro- Power EEG acquisition SoC with Integrated Feature Extraction Processor for a Chronic Seizure Detection System," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 804–816, April 2010.

[24] J. M. Kahn, R. H. Katz, and K. S. J. Pister, "Emerging Challenges: Mobile Networking for Smart Dust," *Journal of Communications and Networks*, vol. 2, no. 3, pp. 188–196, September 2000.

[25] B. Warneke, B. Atwood, and K. S. J. Pister, "Smart Dust Mote Forerunners," in *International Conference on Micro Electro Mechanical Systems (MEMS) Digest of Technical Papers*, pp. 357–360, January 2001.

[26] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with Zebranet," *ACM SIGARCH Computer Architecture News - Special Issue: Annual Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 30, no. 5, pp. 96–107, October 2002.

[27] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," in *ACM International Workshop on Wireless Sensor Networks and Applications*, pp. 88–97, September 2002.

[28] R. Tapiador-Morales, A. Rios-Navarro, A. Jimenez-Fernandez, J. Dominguez-Morales, and A. Linares-Barranco, "System based on Inertial Sensors for Behavioral Monitoring of Wildlife," in *International Conference on Computer, Information and Telecommunication Systems (CITS)*, pp. 1–4, July 2015.

[29] J. P. Dominguez-Morales, A. Rios-Navarro, M. Dominguez-Morales, R. Tapiador-Morales, D. Gutierrez-Galan, D. Cascado-Caballero, A. Jimenez-Fernandez, and A. Linares-Barranco, "Wireless Sensor Network for Wildlife Tracking and Behavior Classification of Animals in Doñana," *IEEE Communications Letters*, vol. 20, no. 12, pp. 2534–2537, December 2016.

[30] P. Mathur, R. H. Nielsen, N. R. Prasad, and R. Prasad, "Wildlife Conservation and Rail Track Monitoring using Wireless Sensor Networks," in *International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE)*, pp. 1–4, May 2014.

[31] X. Shen, Z. Wang, and Y. Sun, "Wireless Sensor Networks for Industrial Applications," in *World Congress on Intelligent Control and Automation*, vol. 4, pp. 3636–3640, June 2004.

[32] P. Casari, A. P. Castellani, A. Cenedese, C. Lora, M. Rossi, L. Schenato, and M. Zorzi, "The Wireless Sensor Networks for City-Wide Ambient Intelligence (WISE-WAI) Project," *Sensors*, vol. 9, no. 6, pp. 4056–4082, May 2009.

[33] A. Ali, Y. Ming, S. Chakraborty, and S. Iram, "A Comprehensive Survey on Real-Time Applications of WSN," *Future Internet*, vol. 9, no. 4, pp. 77:1–77:22, November 2017.

[34] M. Patel and J. Wang, "Applications, Challenges, and Prospective in Emerging Body Area Networking Technologies," *IEEE Wireless Communications*, vol. 17, no. 1, February 2010.

[35] S. Sudevalayam and P. Kulkarni, "Energy Harvesting Sensor Nodes: Survey and Implications," *IEEE Communications Surveys Tutorials*, vol. 13, no. 3, pp. 443–461, July 2011.

[36] A. Chandrakasan, N. Verma, and D. C. Daly, "Ultralow-Power Electronics for Biomedical Applications.," *Annual Review of Biomedical Engineering*, vol. 10, pp. 247–274, April 2008.

[37] L. S. Y. Wong, S. Hossain, A. Ta, J. Edvinsson, D. H. Rivas, and H. Naas, "A Very Low-Power CMOS mixed-signal IC for Implantable Pacemaker Applications," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 12, pp. 2446–2456, December 2004.

[38] S. Kim, N. Cho, S.-J. Song, D. Kim, K. Kim, and H.-J. Yoo, "Clearphone : A 0.9V 96 μW Digital Hearing Aid System," in *IEEE Biomedical Circuits and Systems Conference (BioCAS) Digest of Technical Papers*, pp. 182–185, November 2006.

[39] B. Gyselinckx, C. V. Hoof, J. Ryckaert, R. F. Yazicioglu, P. Fiorini, and V. Leonov, "Human++: Autonomous Wireless Sensors for Body Area Net-

works," in *IEEE Custom Integrated Circuits Conference*, pp. 13–19, September 2005.

[40] H. Kim, S. Kim, N. V. Helleputte, A. Artes, M. Konijnenburg, J. Huisken, C. V. Hoof, and R. F. Yazicioglu, "A Configurable and Low-Power Mixed Signal SoC for Portable ECG Monitoring Applications," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 8, no. 2, pp. 257–267, April 2014.

[41] Y. Chen, D. Jeon, Y. Lee, Y. Kim, Z. Foo, I. Lee, N. B. Langhals, G. Kruger, H. Oral, O. Berenfeld, Z. Zhang, D. Blaauw, and D. Sylvester, "An Injectable 64 nW ECG Mixed-Signal SoC in 65 nm for Arrhythmia Monitoring," *IEEE Journal of Solid-State Circuits*, vol. 50, pp. 375–390, Jan 2015.

[42] E. S. Winokur, T. O'Dwyer, and C. G. Sodini, "A Low-Power, Dual-Wavelength Photoplethysmogram (PPG) SoC with Static and Time-Varying Interferer Removal," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 9, no. 4, pp. 581–589, August 2015.

[43] X. Xie, G. Li, X. Chen, X. Li, and Z. Wang, "A Low-Power Digital IC Design Inside the Wireless Endoscopic Capsule," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 11, pp. 2390–2400, November 2006.

[44] M. Etemadi, O. T. Inan, J. A. Heller, S. Hersek, L. Klein, and S. Roy, "A Wearable Patch to Enable Long-Term Monitoring of Environmental, Activity and Hemodynamics Variables," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, no. 2, pp. 280–288, April 2016.

[45] L. Mo, S. Liu, R. X. Gao, D. John, J. W. Staudenmayer, and P. S. Freedson, "Wireless Design of a Multisensor System for Physical Activity Monitoring," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 11, pp. 3230–3237, November 2012.

[46] U. Ha, J. Lee, M. Kim, T. Roh, S. Choi, and H. Yoo, "An EEG-NIRS Multimodal SoC for Accurate Anesthesia Depth Monitoring," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 6, pp. 1830–1843, June 2018.

[47] Texas Instruments CC2420 Datasheet. [Online]. Accessed: 23-11-2018. Available: http://www.ti.com/lit/ds/symlink/cc2420.pdf.

[48] Nordic Semiconductor nRF24L01 Datasheet. [Online]. Accessed: 23-11-2018. Available: https://www.nordicsemi.com/DocLib?Product=nRF24.

[49] B. Gyselinckx, R. Vullers, C. V. Hoof, J. Ryckaert, R. F. Yazicioglu, P. Fiorini, and V. Leonov, "Human++: Emerging Technology for Body Area Networks," in *International Conference on Very Large Scale Integration*, pp. 175–180, October 2006.

[50] H. Kim, R. F. Yazicioglu, T. Torfs, P. Merken, C. V. Hoof, and H. J. Yoo, "An Integrated Circuit for Wireless Ambulatory Arrhythmia Monitoring Systems," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, pp. 5409–5412, September 2009.

[51] W. Rhee, G. Burra, K. Arimoto, P. Harpe, B. Otis, and D. Ruffieux, "Low-Power Radios for Sensor Networks," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, pp. 518–519, February 2014.

[52] R. Piyare, A. L. Murphy, C. Kiraly, P. Tosato, and D. Brunelli, "Ultra Low Power Wake-Up Radios: A Hardware and Networking Survey," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2117–2157, July 2017.

[53] Y. Liu, X. Huang, M. Vidojkovic, A. Ba, P. Harpe, G. Dolmans, and H. d. Groot, "A 1.9nJ/b 2.4Ghz Multistandard (Bluetooth Low Energy/Zigbee/IEEE802.15.6) Transceiver for Personal/Body-Area Networks," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, pp. 446–447, February 2013.

[54] M. Ding, P. Zhang, C. Lu, Y. Zhang, S. Traferro, G. van Schaik, Y. Liu, J. Huijts, C. Bachmann, G. Dolmans, and K. Philips, "A 2.4Ghz BLE-compliant Fully-Integrated Wakeup Receiver for Latency-Critical IoT Applications using a 2-Dimensional Wakeup Pattern in 90nm CMOS," in *IEEE Radio Frequency Integrated Circuits Symposium (RFIC)*, pp. 168–171, June 2017.

[55] A. L. Goldberger, L. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C. K. Peng, and H. E. Stanley, "Physiobank, Physiotoolkit, and Physionet: Components of a new research resource for complex Physiologic Signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, June 2013. circulation Electronic Pages: http://circ.ahajournals.org/cgi/content/full/101/23/e215.

[56] G. B. Moody and R. G. Mark, "The impact of the MIT-BIH Arrhythmia Database," *IEEE Engineering in Medicine and Biology Magazine*, vol. 20, no. 3, pp. 45–50, May 2001.

[57] G. B. Moody and R. G. Mark, "A Database to support development and evaluation of Intelligent Intensive Care Monitoring," in *Computers in Cardiology (CinC) Digest of Technical Papers*, pp. 657–660, September 1996.

[58] W. Zong, G. B. Moody, and D. Jiang, "A Robust Open-Source Algorithm to detect Onset and Duration of QRS Complexes," in *Computers in Cardiology (CinC) Digest of Technical Papers*, pp. 737–740, September 2003.

[59] J. Kim and H. Shin, "Simple and Robust Realtime QRS Detection Algorithm Based on Spatiotemporal Characteristic of the QRS Complex," *PLOS ONE*, vol. 11, no. 3, pp. 1–13, March 2016.

[60] M. R. Arefin, K. Tavakolian, and R. Fazel-Rezai, "QRS Complex Detection in ECG Signal for Wearable Devices," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, pp. 5940–5943, August 2015.

[61] J. P. Martinez, R. Almeida, S. Olmos, A. P. Rocha, and P. Laguna, "A Wavelet-based ECG Delineator: Evaluation on Standard Databases," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 4, pp. 570–581, April 2004.

[62] D. Chen, S. Wan, J. Xiang, and F. S. Bao, "A high-performance seizure detection algorithm based on Discrete Wavelet Transform (DWT) and EEG," *PLOS ONE*, vol. 12, no. 3, pp. 1–21, March 2017.

[63] A. Subasi, "Automatic Recognition of Alertness Level from EEG by using Neural Network and Wavelet Coefficients," *Expert Systems with Applications*, vol. 28, no. 4, pp. 701–711, May 2005.

[64] T. Rusch, R. Sankar, and J. Scharf, "Signal Processing Methods for Pulse Oximetry," *Computers in Biology and Medicine*, vol. 26, no. 2, pp. 143 – 159, March 1996.

[65] C. Ieong, P. Mak, C. Lam, C. Dong, M. Vai, P. Mak, S. Pun, F. Wan, and R. P. Martins, "A 0.83-μw QRS Detection Processor using Quadratic Spline Wavelet Transform for Wireless ECG Acquisition in 0.35-μmCMOS," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, no. 6, pp. 586–595, December 2012.

[66] N. Ravanshad, H. Rezaee-Dehsorkh, R. Lotfi, and Y. Lian, "A Level- Crossing based QRS Detection ALgorithm for Wearable ECG Sensors," *IEEE Journal of Biomedical and Health Informatics*, vol. 18, no. 1, pp. 183–192, January 2014.

[67] S. K. Jain and B. Bhaumik, "An Energy Efficient ECG Signal Processor detecting Cardiovascular Diseases on Smartphone," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 2, pp. 314–323, April 2017.

[68] P. Li, X. Zhang, M. Liu, X. Hu, B. Pang, Z. Yao, H. Jiang, and H. Chen, "A 410-nW Efficient QRS Processor for Mobile ECG monitoring in 0.18-μm CMOS," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 6, pp. 1356–1365, December 2017.

[69] N. Bayasi, T. Tekeste, H. Saleh, B. Mohammad, A. Khandoker, and M. Ismail, "Low-Power ECG-based Processor for Predicting Ventricular Arrhythmia," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 5, pp. 1962–1974, May 2016.

[70] T. Tekeste, H. Saleh, B. Mohammad, and M. Ismail, "Ultra-Low Power QRS Detection and ECG Compression Architecture for IoT Healthcare Devices,"

IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 66, no. 2, pp. 669–679, February 2019.

[71] W. Shih, K. Huang, C. Chen, W. Fang, G. Cauwenberghs, and T. Jung, "An Effective Chip Implementation of a Real-Time Eight-Channel EEG S ignal Processor based on on-line Recursive ICA Algorithm," in *IEEE Biomedical Circuits and Systems Conference (BioCAS) Digest of Technical Papers*, pp. 192–195, November 2012.

[72] M. B. Malarvili and M. Mesbah, "Combining newborn EEG and HRV information for Automatic Seizure Detection," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, pp. 4756–4759, August 2008.

[73] H. Abdullah, G. Holland, I. Cosic, and D. Cvetkovic, "Correlation of sleep EEG frequency bands and Heart Rate Variability," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Digest of Technical Papers*, pp. 5014–5017, September 2009.

[74] K. Lin, J. Liao, and W. Fang, "A highly integrated Biomedical Multiprocessor SoC design for a Wireless Bedside Monitoring System," in *IEEE International Symposium on Circuits and Systems (ISCAS) Digest of Technical Papers*, pp. 1392–1395, June 2014.

[75] W. Fang, C. Chen, E. Chua, C. Fu, S. Tseng, and S. Kang, "A Low Power Biomedical Signal Processing System-on-Chip Design for Portable Brain-Heart Monitoring Systems," in *International Conference on Green Circuits and Systems*, pp. 18–23, June 2010.

[76] M. Ashouei, J. Hulzink, M. Konijnenburg, J. Zhou, F. Duarte, A. Breeschoten, J. Huisken, J. Stuyt, H. de Groot, F. Barat, J. David, and J. V. Ginderdeuren, "A Voltage-Scalable Biomedical Signal Processor running ECG using 13pJ/cycle at 1MHz and 0.4V," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, pp. 332–334, February 2011.

[77] J. Hulzink, M. Konijnenburg, M. Ashouei, A. Breeschoten, T. Berset, J. Huisken, J. Stuyt, H. de Groot, F. Barat, J. David, and J. Van Ginderdeuren, "An Ultra Low Energy Biomedical Signal Processing System Operating at Near-Threshold," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 5, no. 6, pp. 546–54, December 2011.

[78] Y. Zhang, Y. Tian, Z. Wang, Y. Ma, and Y. Ma, "An ECG Intelligent Monitoring System with MSP430 Microcontroller," in *International Workshop on Systems, Signal Processing and their Applications (WoSSPA)*, pp. 214–219, May 2013.

[79] K. Siwiec, K. Marcinek, P. Boguszewicz, T. Borejko, A. Halauko, A. Jarosz, J. KopaÅĎski, E. Kurjata-Pfitzner, P. Narczyk, M. Plasota, A. Wielgus, and W. A. Pleskacz, "BioSoC: Highly Integrated System-on-Chip for Health Monitoring," in *International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS) Digest of Technical Papers*, pp. 1–6, April 2016.

[80] V. Shnayder, B.-R. Chen, K. Lorincz, T. R. F. Fulford-Jones, and M. Welsh, "Sensor Network for Medical Care," Tech. Rep. TR-08-05, Division of Engineeering and Applied Sciences, Hardward University, 2005.

[81] R. DeVaul, M. Sung, J. Gips, and A. Pentland, "MIThril 2003: Applications and Architecture," in *IEEE International Symposium on Wearable Computers*, pp. 4–11, October 2003.

[82] T. Gao, T. Massey, L. Selavo, D. Crawford, B. Chen, K. Lorincz, V. Shnayder, L. Hauenstein, F. Dabiri, J. Jeng, A. Chanmugam, D. White, M. Sarrafzadeh, and M. Welsh, "The Advanced Health and Disaster Aid Network: A Light-Weight Wireless Medical System for Triage," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, no. 3, pp. 203–216, September 2007.

[83] F. C. Bauer, D. R. Muir, and G. Indiveri, "Real-Time Ultra-Low Power ECG Anomaly Detection using an Event-Driven Neuromorphic Processor," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1575–1582, December 2019.

[84] S. Razmpour, A. M. Sodagar, M. Faizollah, M. Y. Darmani, and M. Nourian, "Reconfigurable Biological Signal Co-processor for Feature Extraction dedicated to Implantable Biomedical Microsystems," in *IEEE International Symposium on Circuits and Systems (ISCAS) Digest of Technical Papers*, pp. 861–864, May 2013.

[85] T. Roh, S. Lee, and H. Yoo, "A 15ÂţW 16 channel ExG Processor with Data Transition Memory-Quad Level Vector for Wearable Healthcare Platform," in *IEEE Biomedical Circuits and Systems Conference (BioCAS) Digest of Technical Papers*, pp. 325–328, November 2011.

[86] S. R. Sridhara, M. DiRenzo, S. Lingam, S. J. Lee, R. Blazquez, J. Maxey, S. Ghanem, Y. H. Lee, R. Abdallah, P. Singh, and M. Goel, "Microwatt Embedded Processor Platform for Medical System-on- Chip Applications," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 4, pp. 721–730, April 2011.

[87] B. Mishra and B. M. Al-Hashimi, "Subthreshold FIR Filter Architecture for Ultra Low Power Applications," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation (PATMOS)* (L. Svensson and J. Monteiro, eds.), pp. 1–10, Springer Berlin Heidelberg, 2009.

[88] M. H. Sunwoo and S. K. Oh, "A Multiplierless 2- D Convolver Chip for real-time Image Processing," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 38, no. 1, pp. 63–71, August 2004.

[89] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, February 2007.

[90] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, May 2000.

[91] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained

Reconfigurable Matrix," in *Field Programmable Logic and Application. FPL 2003. Lecture Notes in Computer Science* (P. Y. K. Cheung and G. A. Constantinides, eds.), vol. 2778, pp. 61–70, Springer, Berlin, Heidelberg, 2003.

[92] Y. Park, H. Park, and S. Mahlke, "CGRA Express: Accelerating Execution using Dynamic Operation Fusion," in *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pp. 271–280, October 2009.

[93] C. Liang and X. Huang, "SmartCell: A Power-Efficient Reconfigurable Architecture for Data Streaming Applications," in *IEEE Workshop on Signal Processing Systems*, pp. 257–262, October 2008.

[94] T. Miyamori and K. Olukotun, "REMARC: Reconfigurable Multimedia Array Coprocessor," in *IEICE Transactions on Information and Systems E82-D*, pp. 389–397, February 1998.

[95] Amber ARM-compatible core. [Online]. Accessed: 25-11-2018. Available: https://opencores.org/projects/amber.

[96] K. Patel and C. J. Bleakley, "Coarse Grained Reconfigurable Array based Architecture for Low Power Real-Time Seizure Detection," *Journal of Signal Processing Systems*, vol. 82, no. 1, pp. 55–68, January 2016.

[97] K. Patel, S. McGettrick, and C. J. Bleakley, "SYSCORE: A Coarse Grained Reconfigurable Array Architecture for Low Energy Biosignal Processing," in *Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 109–112, May 2011.

[98] J. Lopes, D. Sousa, and J. C. Ferreira, "Evaluation of CGRA architecture for real-time processing of biological signals on wearable devices," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–7, December 2017.

[99] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim, "ULP-SRP: Ultra Low Power Samsung Reconfigurable Processor for Biomedical

Applications," in *International Conference on Field-Programmable Technology*, pp. 329–334, December 2012.

[100] M. Konijnenburg, Y. Cho, M. Ashouei, T. Gemmeke, C. Kim, J. Hulzink, J. Stuyt, M. Jung, J. Huisken, S. Ryu, J. Kim, and H. d. Groot, "Reliable and Energy-Efficient 1Mhz 0.4V Dynamically Reconfigurable SoC for ExG Applications in 40nm LP CMOS," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, pp. 430–431, February 2013.

[101] L. Duch, S. Basu, R. Braojos, D. Atienza, G. Ansaloni, and L. Pozzi, "A Multi-Core Reconfigurable Architecture for Ultra-Low Power Bio-Signal Analysis," in *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 416–419, October 2016.

[102] R. Zimmermann, *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, July 1998.

[103] R. Hegde and N. R. Shanbhag, "Soft Digital Signal Processing," *IEEE Trans. Very Large Scale Integration Systems*, vol. 9, no. 6, pp. 813–823, January 2001.

[104] A. M. Obeid, S. M. Qasim, M. S. BenSaleh, Z. Marrakchi, H. Mehrez, H. Ghariani, and M. Abid, "Flexible Reconfigurable Architecture for DSP Applications," in *IEEE International System-on-Chip Conference (SOCC) Digest of Technical Papers*, pp. 204–209, September 2014.

[105] M. Lanuzza, S. Perri, P. Corsonello, and M. Margala, "A New Reconfigurable Coarse-Grain Architecture for Multimedia Applications," in *Conference on Adaptive Hardware and Systems (AHS)*, pp. 119–126, August 2007.

[106] M. Shoba and R. Nakkeeran, "Energy and area efficient hierarchy multiplier architecture based on vedic mathematics and GDI Logic," *Engineering Science and Technology*, vol. 20, no. 1, pp. 321 – 331, February 2017.

[107] Y. H. Seo and D. W. Kim, "A new VLSI Architecture of Parallel Multiplier-Accumulator based on Radix-2 Modified Booth Algorithm," *IEEE Transac-*

tions on Very Large Scale Integration (VLSI) Systems, vol. 18, no. 2, pp. 201–208, February 2010.

[108] M. Jeevitha, R. Muthaiah, and P. Swaminathan, "Review Article: Efficient Multiplier Architecture in VLSI Design," *Journal of Theoretical and Applied Information Technology*, vol. 38, no. 2, pp. 196–201, April 2012.

[109] M. K. Jaiswal and R. C. Cheung, "Area-Efficient Architectures for Double Precision Multiplier on FPGA with Run-time-Reconfigurable Dual Single Precision Support," *Microelectronics Journal*, vol. 44, no. 5, pp. 421 – 430, May 2013.

[110] A. Suhre, F. Keskin, T. Ersahin, R. Cetin-Atalay, R. Ansari, and A. E. Cetin, "A Multiplication-free Framework for Signal Processing and Applications in Biomedical Image Analysis," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1123–1127, May 2013.

[111] J. G. Proakis and D. K. Manolakis, *Digital Signal Processing*. Prentice-Hall, Inc., 4^{th} ed., 2006.

[112] R. S. Khandpur, *Handbook of Biomedical Instrumentation*. McGraw-Hill Education (India) Pvt Limited, 2^{nd} ed., 2003.

[113] W. J. Tompkins, ed., *Biomedical Digital Signal Processing: C-language Examples and Laboratory Experiments for the IBM PC*. Prentice-Hall, Inc., 1993.

[114] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, September 1959.

[115] J. E. Volder, "The Birth of CORDIC," *Journal of VLSI Signal Processing Systems for Signal, Image and Video technology*, vol. 25, no. 2, pp. 101–105, June 2000.

[116] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers Inc., 1st ed., 2003.

[117] J. S. Walther, "A Unified Algorithm for Elementary functions," in *Proceedings of Spring Joint Computer Conference*, pp. 379–385, May 1971.

[118] J. S. Walther, "The Story of Unified CORDIC," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, vol. 25, no. 2, pp. 107–112, June 2000.

[119] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*. Birkhauser, 22^{nd} ed., 2005.

[120] Jayshankar, "Efficient Computation of the DFT of a 2N-point Real Sequence using FFT with CORDIC Based Butterflies," in *IEEE Region 10 Conference TENCON*, pp. 1–5, November 2008.

[121] P. K. Meher, J. Valls, T. Juang, K. Sridharan, and K. Maharatna, "50 Years of CORDIC: Algorithms, Architectures, and Applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, September 2009.

[122] A. P. Yoganathan, R. Gupta, and W. H. Corcoran, "Fast fourier Transform in the analysis of Biomedical Data," *Medical & Biological Engineering*, vol. 14, pp. 239–45, April 1976.

[123] M. Cesarelli, F. Clemente, and M. Bracale, "A Flexible FFT algorithm for processing Biomedical Signals using a Personal Computer," *Journal of Biomedical Engineering*, vol. 12, no. 6, pp. 527 – 530, November 1990.

[124] C. J. Pfister, J. W. Hamilton, N. Nagel, P. Bass, J. G. Webster, and W. J. Tompkins, "Use of Spectral Analysis in the detection of Frequency Differences in the Electrogastrograms of Normal and Diabetic Subjects," *IEEE Transactions on Biomedical Engineering*, vol. 35, no. 11, pp. 935–941, November 1988.

[125] P. Mahalakshmi and T. Jagadesh, "Spectral Analysis of Fast Fourier Transform for ECG Signal Processing Applications," *International Journal of Applied Engineering Research*, vol. 9, no. 25, pp. 8627–8632, January 2014.

[126] C.-H. Lin, "Frequency-domain features for ECG Beat Discrimination using Grey Relational Analysis-Based Classifier," *Computers and Mathematics with Applications*, vol. 55, no. 4, pp. 680 – 690, February 2008.

[127] H. Gothwal, S. Kedawat, and R. Kumar, "Cardiac Arrhythmias Detection in an ECG Beat Signal using Fast Fourier Transform and Artificial Neural Network," *Journal of Biomedical Science and Engineering*, vol. 4, no. 4, pp. 289–296, April 2011.

[128] R. Sarmiento, F. Tobajas, V. de Armas, R. Esper-Chain, J. F. Lopez, J. A. Montiel-Nelson, and A. Nunez, "A CORDIC Processor for FFT Computation and its implementation using Gallium Arsenide Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 1, pp. 18–30, March 1998.

[129] A. Banerjee, A. S. Dhar, and S. Banerjee, "FPGA realization of a CORDIC based FFT Processor for Biomedical Signal Processing," *Microprocessors and Microsystems*, vol. 25, no. 3, pp. 131 – 142, May 2001.

[130] M. Bekooij, J. Huisken, and K. Nowak, "Numerical Accuracy of Fast Fourier Transforms with CORDIC Arithmetic," *Journal of VLSI Signal Processing Systems*, vol. 25, no. 2, pp. 187–193, June 2000.

[131] M. Garrido and J. Grajal, "Efficient Memoryless CORDIC for FFT Computation," in *IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP*, vol. 2, pp. II.113–II.116, April 2007.

[132] T.-Y. Sung, H.-C. Hsin, and Y.-P. Cheng, "Low-Power and High-Speed CORDIC-based Split-Radix FFT Processor for OFDM Systems," *Digital Signal Processing*, vol. 20, pp. 511–527, March 2010.

[133] K. Sayood, *Introduction to Data Compression*. Morgan Kaufmann Publishers Inc., 3^{rd} ed., 2005.

[134] S. Mallat, *A Wavelet Tour of Signal Processing*. Academic press, 1999.

[135] C. K. Chui, *An Introduction to Wavelets*, vol. 1. Academic Press Professional, Inc., 1 ed., 1992.

[136] A. Haar, "Zur Theorie der orthogonalen Funktionensysteme," *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, September 1910.

[137] I. Daubechies, "Orthonormal Bases of Compactly Supported Wavelets," *Communications on Pure and Applied Mathematics*, vol. 41, no. 7, pp. 909–996, October 1988.

[138] W. Sweldens, "The Lifting Scheme: A Construction of Second Generation Wavelets," *SIAM Journal on Mathematical Analysis*, vol. 29, no. 2, pp. 511–546, January 1998.

[139] A. Cohen, I. Daubechies, and J.-C. Feauveau, "Biorthogonal Bases of Compactly Supported Wavelets," *Communications on Pure and Applied Mathematics*, vol. 45, no. 5, pp. 485–560, June 1992.

[140] I. Daubechies, *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992.

[141] C.-T. Huang, P.-C. Tseng, and L.-G. Chen, "Analysis and VLSI Architecture for 1-D and 2-D Discrete Wavelet Transform," *IEEE Transactions on Signal Processing*, vol. 53, no. 4, pp. 1575–1586, April 2005.

[142] P.-C. Tseng, C.-T. Huang, and L.-G. Chen, "Generic RAM-based Architecture for two-dimensional Discrete Wavelet Transform with Line-Based Method," in *Asia-Pacific Conference on Circuits and Systems*, pp. 363–366, October 2002.

[143] N. D. Zervas, G. P. Anagnostopoulos, V. Spiliotopoulos, Y. Andreopoulos, and C. E. Goutis, "Evaluation of Design Alternatives for the 2-D-Discrete Wavelet Transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 12, pp. 1246–1262, December 2001.

[144] P.-C. Tseng, C.-T. Huang, and L.-G. Chen, "VLSI Implementation of Shape-Adaptive Discrete Wavelet Transform," in *SPIE International Conference on Visual Communications and Image Processing*, pp. 655–666, January 2002.

[145] M. Hilton, B. D. Jawerth, and A. Sengupta, "Compressing Still and Moving Images with Wavelets," *Multimedia Systems*, vol. 2, no. 5, pp. 218–227, December 1994.

[146] M. L. Hilton, "Wavelet and Wavelet Packet Compression of Electrocardiograms," *IEEE Transactions on Biomedical Engineering*, vol. 44, no. 5, pp. 394–402, May 997.

[147] J. Cardenas-Barrera, J. V. Lorenzo-Ginori, and E. Rodríguez-Valdivia, "A Wavelet-Packets based Algorithm for EEG Signal Compression.," *Medical informatics and the Internet in medicine*, vol. 29, no. 1, pp. 15–27, March 2004.

[148] V. R. Dehkordi, H. Daou, and F. Labeau, "A Channel Differential EZW Coding Scheme for EEG Data Compression," *IEEE Transactions on Information Technology in Biomedicine*, vol. 15, no. 6, pp. 831–838, November 2011.

[149] M. S. Manikandan and S. Dandapat, "Wavelet threshold based ECG Compression using USZZQ and Huffman coding of DSM," *Biomedical Signal Processing and Control*, vol. 1, no. 4, pp. 261 – 270, October 2006.

[150] C. Li, C. Zheng, and C. Tai, "Detection of ECG Characteristic Points using Wavelet Transforms," *IEEE Transactions on Biomedical Engineering*, vol. 42, no. 1, pp. 21–28, January 1995.

[151] J. C. Hsieh, W. C. Tzeng, Y. C. Yang, and S. M. Shieh, "Detecting ECG Characteristic Points by Novel Hybrid Wavelet Transforms: An Evaluation of Clinical SCP-ECG Database," in *Computers in Cardiology (CinC)*, pp. 751–754, September 2005.

[152] K. K. Patro, P. R. Kumar, and T. Viswanadham, "An Efficient Signal Processing Algorithm for Accurate Detection of Characteristic Points in Abnormal

ECG Signals," in *International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pp. 1476–1479, March 2016.

[153] T. H. Tran, "Improvement of Methods for Detection of Characteristic Points of Biomedical Signals based on Continuous Wavelet Transformation in the Continuous Flow of Data," in *International Conference on Human Factors in Complex Technical Systems and Environments (ERGO)*, pp. 189–192, July 2018.

[154] M. Bsoul and L. Tamil, "Using Second Generation Wavelets for ECG Characteristics Points Detection," in *Middle East Conference on Biomedical Engineering*, pp. 375–378, February 2011.

[155] N. Jain, M. Singh, and B. Mishra, "Image Compression using 2 D- Discrete Wavelet Transform on a Light- Weight Reconfigurable Hardware," in *International Conference on VLSI Design and Embedded Systems (VLSID)*, pp. 61–66, January 2018.

[156] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 Still Image Compression Standard," *IEEE Signal processing magazine*, vol. 18, no. 5, pp. 36–58, September 2001.

[157] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transform," *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 90–93, January 1974.

[158] W.-H. Chen and W. Pratt, "Scene Adaptive Coder," *IEEE Transactions on Communications*, vol. 32, no. 3, pp. 225–232, March 1984.

[159] G. K. Wallace, "The JPEG Still Picture Compression Standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, February 1992.

[160] N. Roma and L. Sousa, "Efficient Hybrid DCT-Domain Algorithm for Video Spatial Downscaling," *EURASIP Journal on Advances in Signal Processing*, vol. 1, no. 1, pp. 1–16, September 2007.

[161] "International Organisation for Standardisation. Generic coding of Moving Pictures and associated Audio Information –part 2: Video. –coding of Moving Pictures and Audio," Tech. Rep. ISO/IEC JTC1/SC29/WG11, ISO, 1994.

[162] "International Telecommunication Union. ITU-T recommendation H.261 version 1: Video Codec for Audiovisual Services at p $\times$ 64 kbits," tech. rep., ITU-T, 1990.

[163] "International Telecommunication Union. ITU-T recommendation H.263 version 1: Video Coding for Low Bit Rate Communication," tech. rep., ITU-T, 1995.

[164] "Joint Video Team. Recommendation H.264 and ISO/IEC14 496 –10 AVC: Draft ITU-T recommendation and final draft International Standard of Joint Video Specification," tech. rep., ITU-T, 2003.

[165] A. Nait-Ali and C. Cavaro-Menard, *Compression of Biomedical Images and Signals*. Wiley-IEEE Press, 1^{st} ed., 2008.

[166] A. Chaabouni, Y. Gaudeau, J. Lambert, J.-M. Moureaux, and P. Gallet, "H.264 Medical Video Compression for Telemedicine: A Performance Analysis," *IRBM, Special Issue : Medical Image Analysis for Computer Aided Diagnosis*, vol. 37, no. 1, pp. 40 – 48, February 2016.

[167] S. Gujjunoori and B. Amberker, "DCT based Reversible Data Embedding for MPEG-4 Video using HVS Characteristics," *Journal of Information Security and Applications*, vol. 18, no. 4, pp. 157 – 166, December 2013.

[168] K. Najarian and R. Splinter, *Biomedical Signal and Image Processing*. CRC Press, Boca, Raton, 1^{st} ed., 2005.

[169] D. Pinto dos Santos, F. Jungmann, C. Friese, C. DÃijber, and P. Mildenberger, "Irreversible bilddatenkompression in der radiologie," *Der Radiologe*, vol. 53, pp. 257–260, 03 March 2013.

[170] H. A. F. Almurib, T. N. Kumar, and F. Lombardi, "Approximate DCT Image Compression using Inexact Computing," *IEEE Transactions on Computers*, vol. 67, no. 2, pp. 149–159, February 2018.

[171] L. V. Batista, E. U. K. Melcher, and L. C. Carvalho, "Compression of ECG signals by Optimized Quantization of Discrete Cosine Transform Coefficients," *Medical Engineering & Physics*, vol. 23, no. 2, pp. 127 – 134, March 2001.

[172] V. A. Allen and J. Belina, "ECG Data Compression using the Discrete Cosine Transform (DCT)," in *Proceedings Computers in Cardiology (CinC)*, pp. 687–690, October 1992.

[173] X. Hu, R. G. Harber, and S. C. Bass, "Expanding the range of convergence of the CORDIC algorithm," *IEEE Transactions on Computers*, vol. 40, no. 1, pp. 13–21, January 1991.

[174] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard*. Kluwer Academic Publishers, 1^{st} ed., 1992.

[175] Xilinx University Program Virtex-II Pro Development System (XUPV2P) Manual. [Online]. Accessed: 23-11-2018. Available: https://forums.xilinx.com/xlnx/attachments/xlnx/XLNXBRD/11238/3/userguide.pdf.

[176] R. E. Kleiger, P. K. Stein, and J. T. Bigger, "Heart Rate Variability: Measurement and Clinical Utility," *Annals of Noninvasive Electrocardiology*, vol. 10, no. 1, pp. 88–101, January 2005.

[177] A. Shukla and L. Macchiarulo, "A Fast and Accurate FPGA based QRS Detection System," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 4828–4831, August 2008.

[178] C. I. Ieong, M. I. Vai, and P. U. Mak, "ECG QRS Complex Detection with Programmable Hardware," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, pp. 2920–2923, August 2008.

[179] F. Zhang, J. Tan, and Y. Lian, "An Effective QRS Detection Algorithm for Wearable ECG in Body Area Network," in *IEEE Biomedical Circuits and Sys-*

tems Conference (BioCAS) Digest of Technical Papers, pp. 195–198, November 2007.

[180] R. Stojanović, D. Karadaglić, M. Mirković, and D. Milošević, "A FPGA system for QRS Complex Detection based on Integer Wavelet Transform," *Measurement Science Review*, vol. 11, no. 4, pp. 131–138, September 2011.

[181] H. Alemzadeh, Z. Jin, Z. Kalbarczyk, and R. K. Iyer, "An Embedded Reconfigurable Architecture for Patient-Specific Multi-Paramater Medical Monitoring," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, pp. 1896–1900, August 2011.

[182] H. K. Chatterjee, R. Gupta, J. N. Bera, and M. Mitra, "An FPGA implementation of real-time QRS detection," in *International Conference on Computer and Communication Technology (ICCCT)*, pp. 274–279, September 2011.

[183] Y. Li, H. Yu, L. Jiang, L. Ma, and Z. Ji, "Adaptive Lifting Scheme for ECG QRS Complexes Detection and its FPGA Implementation," in *International Conference on Biomedical Engineering and Informatics*, vol. 2, pp. 721–724, October 2010.

[184] S. Roy Chowdhury, "Field Programmable Gate Array based Fuzzy Neural Signal Processing System for differential diagnosis of QRS Complex Tachycardia and Tachyarrhythmia in noisy ECG Signals," *Journal of medical systems*, vol. 36, no. 2, pp. 765–75, July 2010.

[185] H. Zairi, M. Kedir-Talha, S. Benouar, and A. Ait-Amer, "Intelligent System for Detecting Cardiac Arrhythmia on FPGA," in *International Conference on Information and Communication Systems (ICICS)*, pp. 1–5, April 2014.

[186] P. P. Chu, *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 version*. John Wiley & Sons, 2011.

[187] H. J. Wellens, F. W. Bär, E. J. Vanagt, and P. Brugada, "Medical Treatment of Ventricular Tachycardia: Considerations in the Selection of Patients for

Surgical Treatment," *The American Journal of Cardiology*, vol. 49, no. 1, pp. 186 – 193, January 1982.

CHAPTER A

VGA and UART Controllers basics

A.1 VGA Interface

Video Graphics Array (VGA) is a popular display standard developed by IBM in the year 1987, for controlling analog monitors. A VGA connector is a three-row 15-pin connector and is commercially called DE15 port. The 15-pin VGA connector can be found on many video cards, computer monitors, and high definition television sets. On laptop computers or other small devices, a mini-VGA port is sometimes used in place of the full-sized VGA connector. VGA connectors and cables carry component RGBHV (red, green, blue, horizontal sync, vertical sync) video signals. This interface is commonly available on most of the FPGA development boards. The computer controls this port by driving appropriate signals on the pins of the port by means of device driver to display image/video according to the target application. These device drivers are not inherently supported by the FPGA boards. Hence, controllers are required to be designed for these ports which govern their functionality. The controllers drive the appropriate signals to the port pins based on the standards defined for the port, VGA standard in this case. The functional description of the controller is written in VHDL language and is provided to the Xilinx Toolset which maps it to the FPGA chip of the development board.

A.1.1 Basic operation of a Display Panel:

The electron gun of the display unit or the monitor generates a focused electron beam which hits the screen showing pixels of an image. The monitor's internal oscillators and amplifiers generate sawtooth waveforms to control the traversal of electron beam. The electron beam traverses (*i.e.* scans) the screen systematically in a fixed pattern, from left to right and from top to bottom. For example, the electron beam moves from the left edge to the right edge as the voltage applied to the horizontal deflection coil of the monitor gradually increases. After reaching the right edge, the beam returns rapidly to the left edge *i.e.* retraces when the voltage changes to 0. The generation of the sawtooth waveform is controlled by two external synchronization signals, *h_sync* and v*_sync* which are digital signals in nature. The relationship between the *h_sync* signal and the horizontal sawtooth is shown in Fig. A.1. The on and off period of the *h_sync* signal correspond to the rising and falling ramp of the sawtooth waveform.

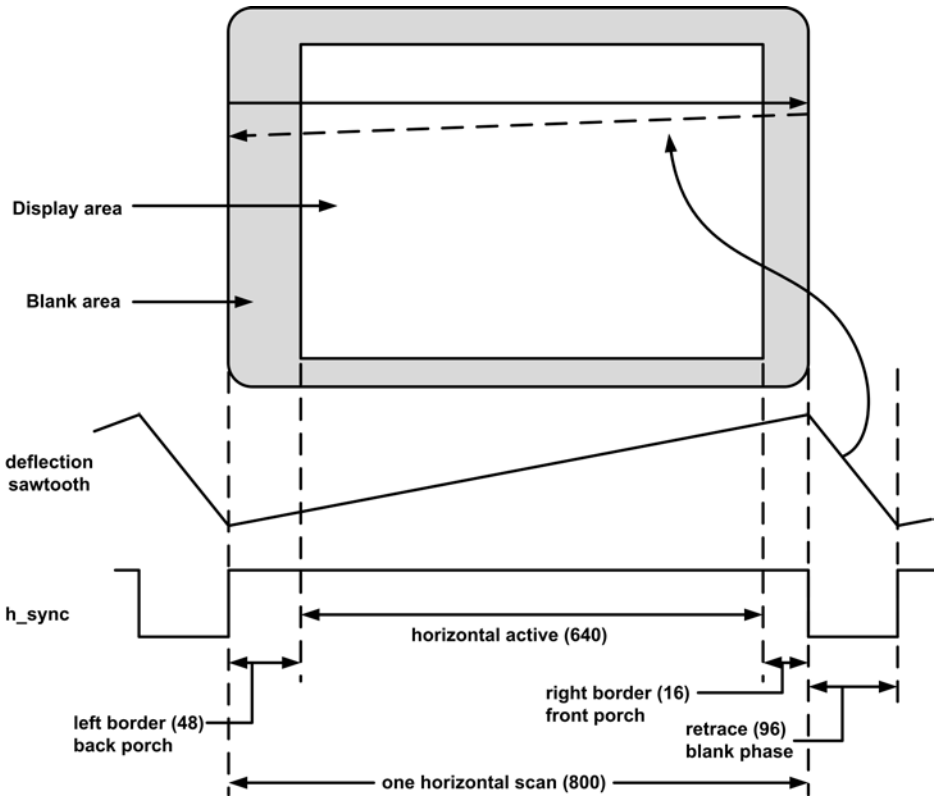


Figure A.1: Timing diagram of a horizontal scan.

Timing of VGA controller

The port requires only 5 of 15 pins for enabling the display unit while the remaining pins are either reserved or driven to ground or supply (as shown in Fig. A.2). The usable pins are - *Red* (analog), *Green* (analog), *Blue* (analog), *h_sync* (digital) and *v_sync* (digital). The *Red*, *Green* and *Blue* are the three color component of the pixel data to be displayed while the *h_sync* and *v_sync* are the synchronization signals. The state of the synchronization signals is dependent on the resolution mode of the display. The VGA standard supports a wide range of resolution formats. The resolution format specifies the attribute parameters which defines the timing of the synchronization signals. The nine attribute parameters which govern the active period of the two synchronization signals (*h_sync* and *v_sync*) are *pixel clock frequency, horizontal active, horizontal front porch, horizontal back porch, horizontal blank, vertical active, vertical front porch, vertical back porch*, and *vertical blank*.

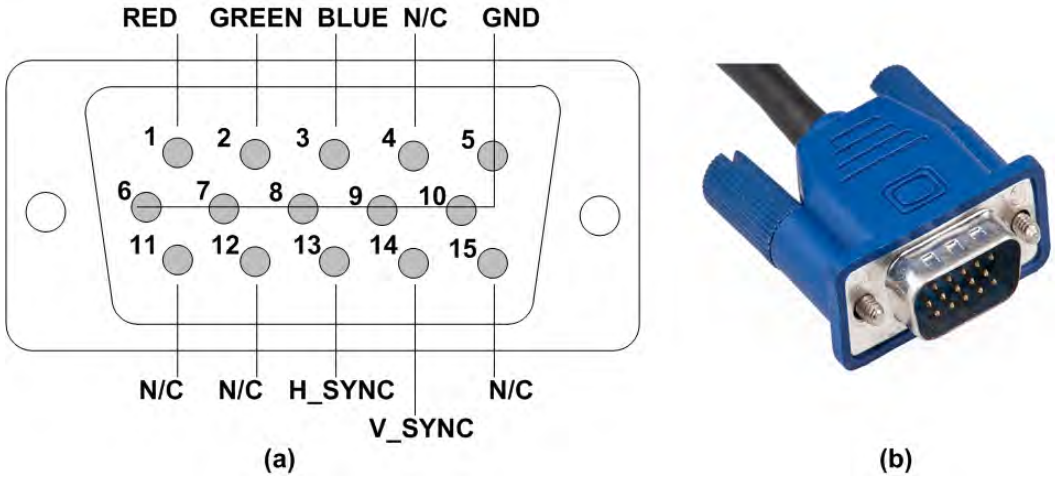


Figure A.2: (a) Pin diagram of VGA connector (DE15 port). (b) VGA cable

The VGA mode employed in the thesis is 640-by-480 (horizontal-by-vertical) resolution with a refresh rate of 60 *Hz*, represented as "640×480@60*Hz*" in general. This mode has a frame of 800-by-524 pixels refreshed at 60 Hz, which accounts to the *pixel clock* frequency of 800×524× 60 = 25.152 M*Hz*. The remaining eight timing parameters collectively generate the two synchronization signals and their duration is specified in Table A.1 [186]. The *h_sync* signal specifies the time required to traverse (scan) a row and the *v_sync* signal specifies the time required

to traverse (scan) all columns of the screen. A detailed timing diagram of one horizontal scan is shown in Fig. A.1. The screen of the monitor usually includes a small black border (shown in Fig. A.1). The inner rectangle forms the visible portion. A period of the *h_sync* signal contains 800 pixels and can be divided into four regions:

- **Display or horizontal active:** It is the region of the display where the pixels are actually displayed on the screen.
- **Retrace or blank phase:** It is the region of the display in which the electron beams return to the left edge. The video signal should be disabled or kept black.
- **Front porch:** It is the region of the display that forms the right border of the display region. The video signal is disabled.
- **Back porch:** It is the region of the display that forms the left border of the display region. The video signal is disabled.

Table A.1: VGA Resolution Timing Parameters.

Horizontal Sync		Vertical Sync	
Name	Duration (in pixels)	Name	Duration (in pixels)
Video Active	640	Video Active	480
Front porch	16	Front porch	11
Blank Phase	96	Blank Phase	2
Back Porch	48	Back Porch	31
Total	800	Total	524

A.2 RS-232 Communication Interface

The visual perception alone is highly subjective and is regarded an insufficient metric to benchmark results. For objective analysis of the result, the output data must be compared with the ideal counterparts obtained from MATLAB and deviation must be quantified. A FPGA to computer communication interface aids

providing/obtaining data to/from FPGAs. The interface also serves the debugging purpose as data can be easily comprehended. The RS-232 communication interface is adopted in this thesis for this purpose.

RS-232 port and its controller

The development board contains the 9-pin serial connector, commercially known as RS-232 serial port. The PC controls this port by driving appropriate signals on the pins of the port with the help of device driver to transmit as well as receive data serially according to the application requirements. These device drivers are not inherently supported by the FPGA boards. Hence, controllers are required to be designed for these ports which govern their functioning. The controllers drive the appropriate signals to port pins based on the RS-232 standard. The 2 pins ,*tx* and *rx*, are used in this thesis for transmission and reception of data whereas the remaining pins are dedicated to supply, ground and optional hardware flow control (as shown in Fig. A.3).

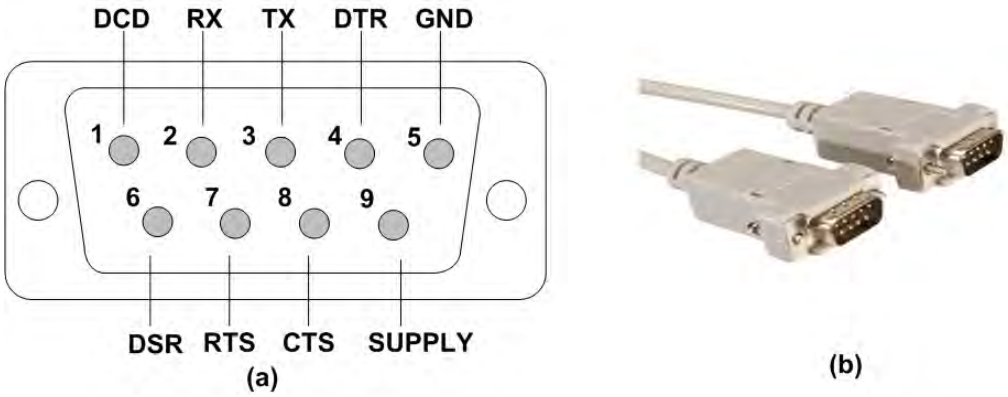


Figure A.3: (a) Pin diagram of RS-232 port. (b) RS-232 cable.

RS -232 Standard

RS-232 standard specifies the electrical, mechanical, functional and procedural characteristics of two data communication equipment communicating using Universal Asynchronous Receiver and Transmitter (UART) circuit. UART is a circuit that sends parallel data through a serial line. This interface uses an asynchronous protocol which means that no clock signal is transmitted along the data. Hence,

certain parameters need to be set before communication. RS-232 standard does it in following way:

- Both sub-systems agree in advance on the communication parameters (speed, format). This is done manually before communication starts.
- The transmitter sends *idle* (= '1') as long as the line is idle.
- The transmitter sends *start* (= '0') before each byte transmitted signaling the receiver that a byte is sent on the communication line.
- The 8 data bits of the byte are sent in serial manner.
- The transmitter sends *stop* (= '1') after each byte.

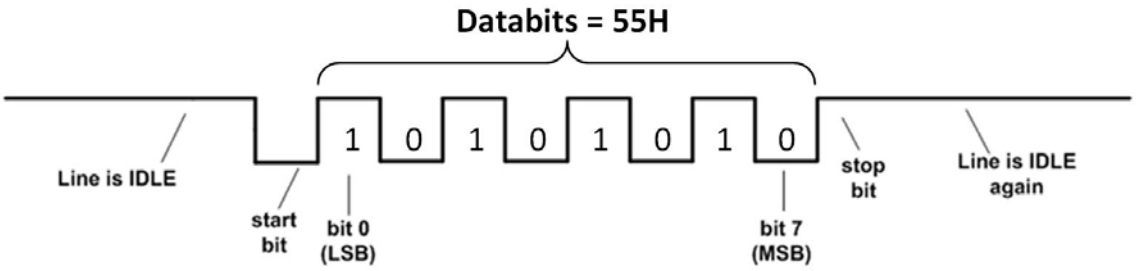


Figure A.4: State of serial transmission line of RS-232 port when 0x55 is sent as data.

For instance, a byte value 0x55 needs to be transmitted from FPGA to PC. The *tx* (or *rx* when data are received) line of UART behaves as shown in Fig. A.4. Multiple Baud rate (specified in bits per second or bps) options available for this communication *viz.* 9600 bps, 19200 bps, 115200 bps *etc.* For a standard image of size "1280×720" where each pixel is represented by 8 bits, the total number of bits are 7.3728 Mbit (1280×720×8). According to RS-232 protocol, every byte has an overhead of 2 bits (1 start + 1 stop) resulting total number of bits to 9.216 Mbit. A Baud Rate of 115200 bps imply transmission of 115200 bits in one sec, hence "1280×720" image require 80 sec (= 9.216 M / 115200) to get transferred from FPGA to computer. The time required by the UART interface with baud rate 115200 bps to transfer image of different sizes is shown in Table A.2.

Table A.2: Time required by UART to transfer image of different sizes

Image Size (each pixel 8-bit wide)	Time for transfer using RS-232
8×8	5.56 ms
16×16	22.22 ms
128×128	1.42 sec
256×256	5.69 sec
720×480	30 sec
1280×720	80 sec

CHAPTER B

ECG Concepts

B.1 The ElectroCardioGram (ECG)

An ECG is the recording (*gram*) of the electrical activity (*electro*) generated by the cells of heart (*cardio*) that reaches the body surface. It is this electrical activity that initiates the heart's muscular contraction (depolarization) leading to pumping of blood to the body. A brief description of the cardiac cycle is presented with the intention of gaining a better insight into what does an ECG waveform represent. The impulses generated by the cells responsible for pacemaking and conduction actions create a rhythmic repetition of events called the *cardiac cycles*. These electrical events further initiates the mechanical events that triggers the cardiac muscle cells (myocardial cells) causing the pumping action of heart. The cardiac muscle cells have specialized *contractile* proteins which slide over each other when actuated by the electrical activity causing depolarization and repolarization (recovery) phases. The depolarization (and repolarization) waves spread through all the myocardial cells resulting in a potential large enough to be reflected on body surface. The process of depolarization produces a high frequency positive deflection on the ECG whereas repolarization generates an oppositely directed low frequency negative waveform.

A single repolarized myocardial cell undergoing depolarization and returning to recovery state is shown in Figure B.1 representing the cardiac cycle. The associated electrical activity is shown by the potential variation and representative ECG. The electrical activity of a single cell or a small group of cells is not produce enough voltage to be recorded on the body surface and requires a large number

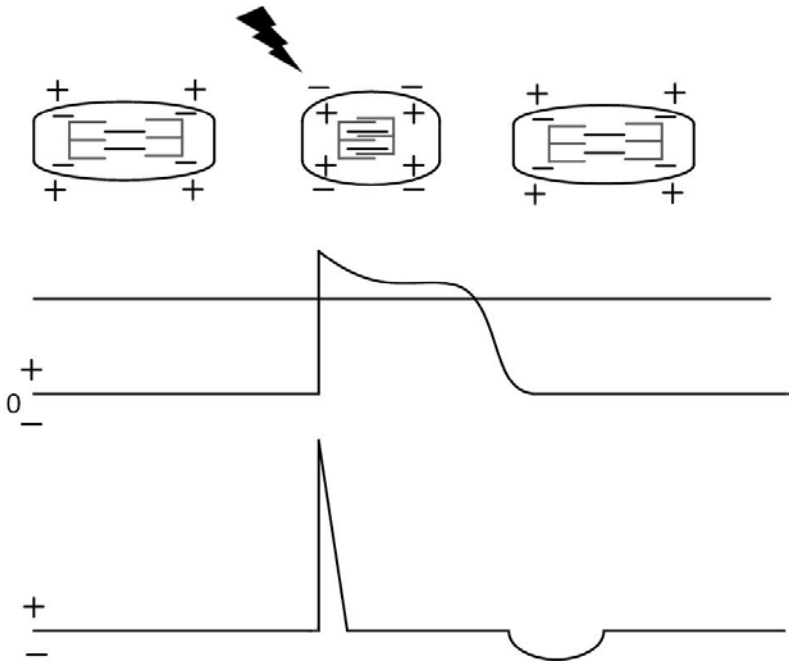


Figure B.1: Depolarization and repolarization of single myocardial cell during the cardiac cycle along with potential variation and representative ECG.

of excited atrial as well as ventricular cells for sufficiently large potential. The myocardial cells itself lacks the ability to generate and rapid propagate electrical impulses and this purpose is served by specialized cells dedicated for pacemaking and conduction system. These cells are arranged in nodes, bundles and bundle branches strategically located through the heart for spontaneous generation and controlled spread of electrical conduction. The *sinoatrial* (SA) or sinus node is the predominant cardiac pacemaker is located high in the right atrium. The atrioventricular (AV) node located low in the right atrium conducts impulses from atria to the ventricles, these chambers are otherwise separated by non-conducting structures. The impulses in the ventricles are further spread using right and left bundle branches of the *bundle of His* and *Purkinje* cells.

B.1.1 Standard 12-lead ECG

The standard 12-lead ECG provides 12 viewpoints for recording cardiac electrical activity. Each lead contains a positive and negative pole and records the potential difference between them. Six of the leads provide *frontal plane* views and the remaining six provide the *transverse plane* views of the body.

Einthoven placed recording electrodes on the right and left arms and left leg resulting in first ever measurement of what he called *Elektrokardiogramme* (EKG) using the string galvanometer in 1901. He introduces three leads (I, II, and III) measurement system shown in Fig. B.2, with each lead consisting of a pair of electrodes serving as positive and negative pole. The three ECG leads (I, II, and III) form an equilateral triangle known as the Einthoven triangle.

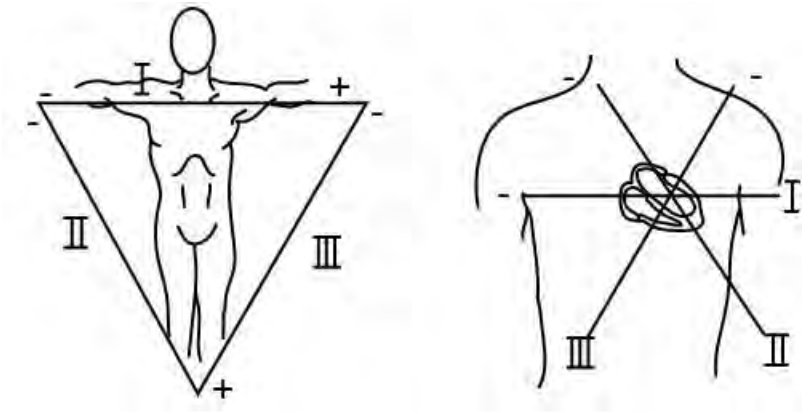


Figure B.2: Einthoven's triangle and its position in relation to heart's schematic.

The 60° angles between the leads leave wide gap between the three views they provide. By the efforts of Wilson et al. these gaps were filled without any additional body surface electrode by creating a central terminal. The leads using the central terminal as their negative terminals are termed as *V leads*. The central terminal is created by connecting all the three limb electrodes together and led to decreased amplitude in the positive leads because of partial signal cancellation. Goldberger et al. presented the concept of *augmented lead*, wherein the limb at which the positive electrode is connected in excluded while creating the central terminal. Hence, the positive lead signal amplitude was increased or augmented and such an augmented V lead is termed as *aV lead*. The augmented leads aVR, aVF and aVL are shown in Fig. B.3. These leads fill the gaps between leads I and II, II and III, and III and I, respectively. Modern digital electrocardiographs record leads I and II only and calculated the remaining leads (III and aV) using Einthoven law and analytical formulas. The six leads of the frontal plane are also referred to as the hexaxial leads system.

The remaining six leads introduced by Wilson belongs to the *transverse plane*

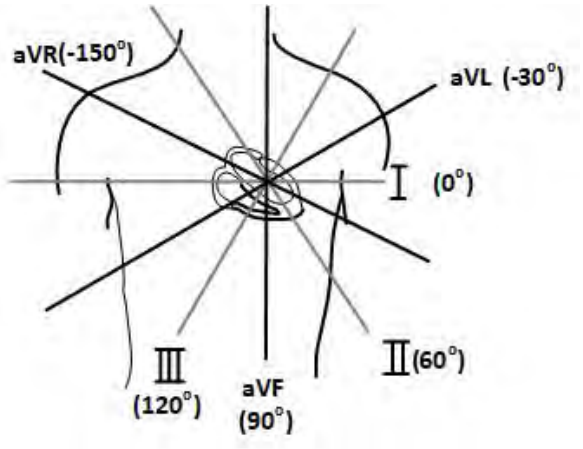


Figure B.3: Frontal plane limb leads.

of the body. The positive electrodes belonging to these leads are positioned at the anterior and left lateral chest using the bony landmarks of the thorax as a guide. The central terminal of the hexaxial system is used as the negative electrode for these leads and therefore are V leads requiring no augmentation. Hence, they are simply terms as leads V1 through V6 leads shown in Fig. B.4.

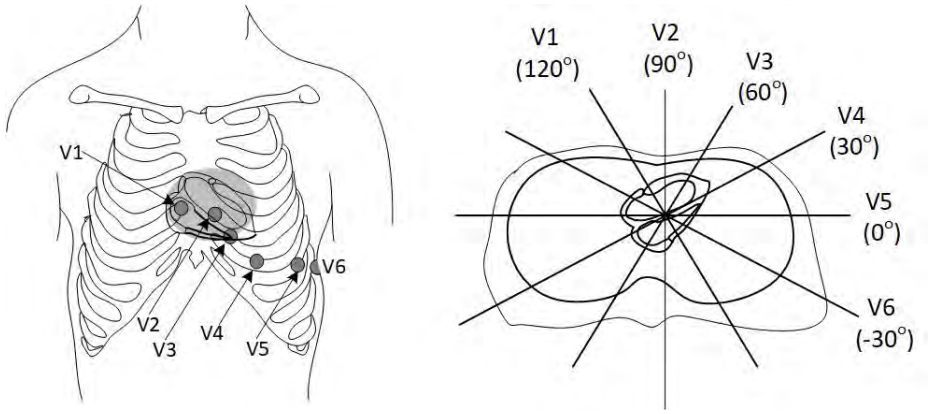


Figure B.4: Front view of the transverse leads and its position in relation to heart's schematic.

Analysing various ECG features serve as an important diagnostic tool indicating the state of various parts involved in cardiac activity. Rate and regularity are two primary features in the evaluation of cardiac rhythm. In a very broad sense, all rhythms other than regular sinus rhythm (60 to 100 beats per minute) are referred to as arrhythmias. The QRS complex is vital in identifying a number of arrhythmias including electrolyte abnormalities (hyperkalemia and hypercalcemia), rhythm abnormalities (bradyarrhythmia and tachyarrhythmia), myocar-

dial ischemia *etc.* as well as distinguishing between conditions of ventricular or supraventricular origins. The latter is important as both conditions follow different line of treatment [187].

CHAPTER C

Mapping 8×8 DCT on 3×3 SAC Architecture

C.1 Mapping scheme for 8×8 DCT on (3×3) SAC architecture

Consider mapping 8×8 DCT on (3×3) SAC architecture. The intermediate matrix rows can be generated by letting the eight input data (corresponding to first column of image) occupy eight RU sites, say RU #1-8 and apply DCT matrix coefficient on the respective coefficients resulting in first element of first row of intermediate matrix. The second element can be generated by replacing (flushing and loading) the first image column by the second image column. Similarly remaining elements of intermediate matrix can be generated. These elements needs to be stored in a memory and applied to the architecture for subsequent computation of DCT results. Additionally, loading image column takes 16 cycles, resulting in 128 (= 16×8) cycles and 64 (= 8 ×8) computation cycles for each intermediate row calculation. Subsequently, every DCT result takes (16+8)×8 cycles to compute every intermediate row and 80 computation cycles to compute 10 DCT results amounting to a total cycle count of 848 (= (16+8)×8×4 + 80) per 8×8 block. Furthermore, 8-deep 9-b wide register file is required to store intermediate matrix elements. Apart from the strenuous cycle and memory needs, the architecture is required to be stalled while one row of image is replacing the other row as well as when intermediate row data is loaded into the circular data memory. Such details are difficult to comprehend for a user and result in a complex synchronization mechanisms and memory interpretation contrary to the simplified configuration

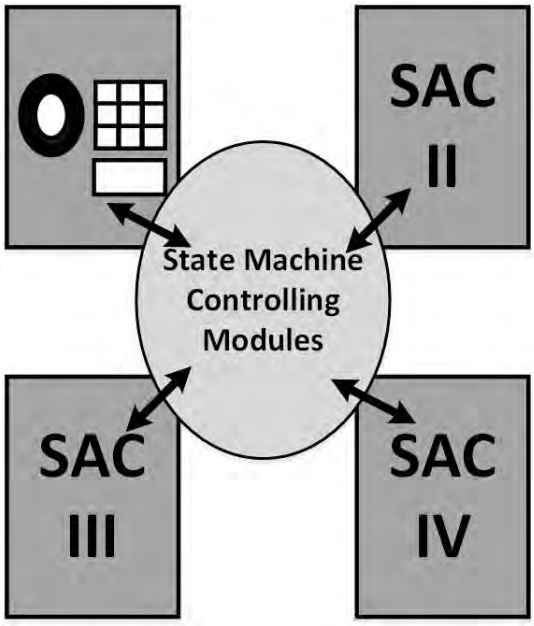


Figure C.1: A conceptual diagram of four (3×3) SAC units connected together.

scheme of the architecture.

Alternatively, mapping scheme for computation intensive functions can be developed by deploying multiple (3×3) units of the SAC architecture. The modular nature of the architecture allows seamless data exchanges and expands the horizon of target functions many folds. Such a scheme allows the user to realize functions not considered for mapping during architecture development due to multitude of possible topologies and coarse user control over units. Primarily, the user controls 'when' and 'where' the data must flow among the units by means of a state machine designated to activate/deactivate units in a time-controlled manner. A conceptual diagram of four (3×3) SAC units connected side by side is shown in Fig. C.1.

A mapping scheme for 8×8 DCT using four (3×3) SAC units is discussed now. The RU-CU and associated data memory are embedded with the ability to be activated/deactivated individually by means of separate resets. The steps presented below define the structure of state machine coordinating with the four SAC units and is independent of the state machine internal to the architecture.

- Memories #1, 2 and 3 are activated. Memory #1 is loaded with image rows 1, 2 and 3. Memory #2 is loaded with image rows 4, 5 and 6. Memory #3 is

loaded with image rows 7 and 8.

- RU-CU #1, 2 and 3 are activated and accepts data from corresponding memories. The result from all RU-CUs are generated at every 8^{th} clock and are accumulated using two adders. The accumulated result corresponds to a row element of intermediate matrix. The accumulated results are stored in memory #4.
- RU-CU #4 is activated and generates DCT results every 8^{th} clock. The memory and RU-CU of remaining SAC units is deactivated.
- The sequence of steps is repeated for four times to generate the upper triangular DCT matrix with 10 terms.