Development of FPGA based Coprocessors for Signal Processing Applications

A thesis submitted in partial fulfillment of the requirements
for the award of the degree of

DOCTOR OF PHILOSOPHY

in

Electronic Sciences

By
Rangababu P.
(06PHPE01)

School of Physics
University of Hyderabad
Hyderabad-500 046
India

August-2013

School of Physics
University of Hyderabad
Hyderabad, India

DECLARATION

I, Mr.RANGABABU P. (06PHPE01) here by declare that the work embodied in this dissertation entitled "Development of FPGA based Coprocessors for Signal Processing Applications" submitted to the University of Hyderabad, Hyderabad, for partial fulfillment of the degree of Doctor of Philosophy in Electronic Sciences has been carried out by me under the supervision of Dr. SAMRAT L. SABAT, School of Physics, University of Hyderabad. To the best of my knowledge, this work has not been submitted for any other degree in any university.

RANGABABU P. (06PHPE01)
Ph.D (Electronic Sciences)
Reg. No: 06PHPE01
Date:

School of Physics
University of Hyderabad
Hyderabad, India

CERTIFICATE

This is to certify that the work described in this thesis entitled "Development of FPGA based Coprocessors for Signal Processing Applications" has been carried out by RANGABABU P. (06PHPE01) under my direct supervision and this has not been submitted for any degree or diploma at this or any other university.

Dr. SAMRAT L. SABAT,
Reader,
School of Physics,
University of Hyderabad.
Date:

Prof. S. CHATURVEDI,
Dean,
School of Physics,
University of Hyderabad.
Date:

To

My Beloved Grand Parents

(Late Sri. Peesapati Chinni Krishnamacharyulu)

&

(Late. Smt. Peesapati Seshamma),
who expired during my term here as Ph.D scholar.
May God give peace to their kind souls . . .

Acknowledgements

This work would have never been accomplished without God blessings and his power that work within me.

I would like to express my sincere gratitude to my research supervisor **Dr. Samrat L. Sabat** for his constant support, encouragement and introducing me to the exciting research topic: System-on-Chip on various signal processing applications, for giving me the opportunity to work, teaching me many principles and techniques, and for his guidance during the research. He has always been approachable, helpful, and extremely patient in his guidance throughout my research period. The research has been a learning and growing experience for further exploration of new technologies for me.

I would thank Prof.S. Chaturvedi, Dean, for providing all the necessary facilities to carry out my work. I also thank Prof.G. Rajaram, Prof.M. Ghanashyam Krishna, Prof.K.C. Jamesraju, Dr.K. Venu, Dr.P.A. Govindacharyulu, Dr.S.V.S Nageswara Rao, and former M.Sc teachers Dr.T.S.N. Somayaji, Dr.D. Sharma, Mr.K. Durgaprasad, Mr.R. Satyanarana. Mr.S.V.N.L. Narayana, Dr. Perisastry, Mr.B.V.G. Krishnarao and Mrs.B.V.G. Vasanta for their valuable suggestions and encouragement.

Financial assistance from the Research Center Imrat (RCI), DRDO, Department of Science and Technology (DST) and Indian Space Research Organization (ISRO) India is gratefully acknowledged. I am also deeply indebted to the authorities at RCI, Engineers, Scientists, for their suggestions and thus help in various aspects of my research. I wish to extend my sincere thanks to the University authorities and Xilinx (XUP) for providing all the necessary facilities for this work and their technical support team Hemasundar, Balakrishna-ahirwal, Saritha.

I am thankful to my lab mates A.Kiran kumar, K.Shravan, K.P.Karthik, B.Swamy, Sai, Gopalakrishna, A.Sivaram, A.Bharadwaj, Layak Ali, N.Giribabu, B.Ravikishore reddy, M.Narshimappa, S.Srinu, P.Srihari, D.Ajay, D.Kishore, Lakshminarayana, K.Sridevi, K.Vasu, B.Sithalakshmi, B.Yugandhar, Ramu, Senthil, Venkiah. Also my school and college mates D.Phanikumar, Ch.Satyasai brothers, M.Balakrishna, M.Jagadeesh, T.B.S.Prasad, K.Ravikumar, P.Yellaji, Chittiannaya, P.Ranganath, Ch.Sudhnvacharyulu, B.Gopalacharyulu, P.S.M.Raghavan, K.M.S.Srinivas for their co-operation and suggestions. I wish to take this opportunity to thank all my friends who encouraged and helped me throughout my research work.

I wish to thank Mr.Abraham for his help in all aspects of administration. I would like to thank Bansilal, subbiramireddy and other non-teaching staff members of School of Physics for their help and co-operation, during my research tenure.

I would be failing in my duties if i do not make a mention of my family members including my mother, father, grand mother, brother, sister, brother-in-law, uncle & aunt and my wife and son for providing moral support, without which this work would not have been completed.

I would like to thank everyone who has helped me, knowingly or unknowingly during my whole research work.

Rangababu Peesapati

Abstract

Now-a-days there has been an increase in demand for designing reconfigurable embedded systems in signal processing, multimedia and evolutionary computation applications. Embedded processors alone, cannot achieve the desired computational capability to fulfill the requirements of massive parallelism, higher memory bandwidth, higher execution speed to execute these applications. In order to meet these requirements, Field Programmable Gate Arrays (FPGAs) are used by exploiting the reconfigurable resources. Beyond their well-known flexibility, FPGAs offer the versatility of running software applications on embedded processors and at the same time taking the advantage of available reconfigurable resources, all on same package.

FPGA based System on Chip (SoC) design solution replaces traditional System on Board (SoB) design concept and is often referred as Programmable SoC (PSoC). This platform consists of hard/soft embedded processors, external memory and custom Intellectual Properties (IPs). These IPs are used to accelerate the computational task of an algorithm. This involves developing dedicated IP and its integration in SoC platform. There are mainly two types of IP interfacing techniques, i.e., Slave Unit (SU) and Auxiliary Processing Unit (APU). The SU interface has Register/First-In-First-Out (FIFO) connected to the processor through shared system bus (Processor Local Bus (PLB)). Although this interface is simpler in design, the main bottleneck is bus arbitration, which lowers the total execution speed. The other bus interface is APU (only for PowerPC440), which can be directly connected to the custom IP through a dedicated Fabric coprocessor Bus (FCB). This interface has no communication overhead and allows quick synchronization between the processor and IP. Custom IPs have been developed and integrated using APU interface for maximizing the portability and modularity.

The work presented in this thesis concentrates on developing efficient coprocessors for three signal processing applications of different complexities in Xilinx (Virtex-5FX70T-1136) development platform. In this work, several important issues related to the efficiency of bus interface, data transfer overhead, and acceleration factors are analyzed. In the first case study, Adaptive Moving Average Dual Mode Kalman Filter (AMADMKF) algorithm is proposed for denoising the FOG signal under both the static and dynamic environments. Performance of the AMADMKF algorithm is compared with other denoising algorithms Discrete Wavelet Transform (DWT) and Kalman Filter (KF). Allan Variance analysis, standard deviation and Signal-to-Noise Ratio (SNR) are used to measure the efficiency of the algorithm. The experimental results have shown that AMADMKF algorithm reduces the standard deviation or drift of the signal by an order of 100 and improves the SNR by approximately 80dB. The Allan Variance analysis result has shown that this algorithm reduces different type of random errors significantly. Further, a hardware IP of the algorithm is developed for SoC implementation using Xilinx Virtex-5 FPGA. The developed IP is interfaced as a coprocessor/APU with the PowerPC440 (PPC440) embedded processor within the FPGA. Hardware acceleration of the developed coprocessor is found to be 65x with respect to its equivalent software implementation.

In the second case study, IPs for fixed and floating point Differential Evolution (DE) algorithm has been developed. The IPs are interfaced using both APU and SU interface techniques with the PPC440 processor. In order to reduce the bus overhead in hardware software co-design platform, the fitness evaluation and DE algorithm are combined together as a single module rather than fitness evaluation in software and DE in hardware. The hardware acceleration of the IP (fixed & float DE) using both interfaces are evaluated with respect to its equivalent software implementation. The performance of IP using both the interfaces are compared. For the purpose of validation, initially, the DE IP is tested by solving benchmark test functions followed by a case study of solving system identification problem. The performances (i.e. acceleration and power consumption) of IPs are measured on 32-bit X86, PPC440 and Microblaze (MB) processors with enabling/disabling hard floating point unit (FPU).

The experimental results reveal that, both interfacing techniques give same acceleration factor. The floating point IP gives higher acceleration compared to the fixed point IP. This is because the floating point software implementation

takes higher execution time. As far as power consumption is concerned, both interfaces (SU, APU) consumed same power. Fixed point DE SoC consumes marginally higher power for optimizing simpler functions whereas, floating point DE SoC consumes more power for optimizing complex functions. Finally, as a case study, an Infinite Impulse Response (IIR) filter based system identification task is implemented using the developed fixed and float DE IP cores (as an APU interface) on the SoC platform. The experimental result has shown that the fixed and float IP attained an acceleration of 11x and 150x with respect to its software implementation in the PPC440 processor.

In the third case study, a single IP core for Baseline profile H.264 decoder is developed. The SoC platform is developed using the open source hardware and software of the H.264 decoder. The developed H.264 IP is interfaced to the PPC440 embedded processor using APU interface and it is tested with different video sequences. It is observed that the coprocessor gives 6-7x acceleration compared to its equivalent software decoder. Finally, this thesis concludes that, the coprocessors developed for above signal processing case studies have achieved adequate acceleration.

Contents

List of Figures

List of Tables

Abbreviations

ADE - Adaptive Differential Evolutionary
ASP - Advanced Simple Profile
API - Application Programming Interface
APU - Auxiliary Processing Unit
AKF - Adaptive Kalman Filter
AVA - Allan Variance analysis
AVC - Advanced Video Coding
ASIC - Application-Specific Integrated Circuit
ASSP - Application-Specific Standard Part
AMA - Adaptive Moving Average
AMADMKF - Adaptive Moving Average based Dual Mode Kalman Filter
BRAM - Block Random Access Memory
BS - Boundary Strength
BSB - Base System Builder
CAD - Computer-Aided Design
CD -Compact Disk
CPU - Central Processing Unit
CIF - Common Intermediate Format
CLB - Combinational Logic Block
CHE -Complete Hardware Evolution
CPLD Combinational Programmable Logical Devices
CDMA - Central Direct Memory Access
CAVLC - Context Adaptive Arithmetic coding
CABAC - Context Adaptive Binary Arithmetic Coding
CRC -Cyclic Redundancy Check
CUDA -Computer Unified Device Architecture
DCM -Digital Clock Manager

DWT - Discrete Wavelet Transform
DBF - Deblocking Filter
DE -Differential Evolution
DVB - Digital Video Broadcasting
DVI - Digital Video Interface
DDR2 - Double Density RAM
DDR-SRAM - Double Data Rate Static Random Access Memory
DCR - Device Control Register
DCT - Discrete Cosine transform
DPB - Decoded Picture Buffer
DSP - Digital Signal Processing
DMA - Direct Memory Access
DA - Distributed Arithmetic
EA - Evolutionary Algorithm
ECG - Electro-Cardio-Graphy
EDK - Embedded Development Kit
EMAC - Ethernet Media Access Control
ESL- Electronic System Level
FCB -Fabric Coprocessor Bus
FCM -Fabric Coprocessor Module
FPU - Floating Point Unit
FIFO - First In-First-Out
FIR - Finite Impulse Response
FPGA - Field Programmable Gate Array
FFT - Fast Fourier Transform
FCC -Flexible Computational Component
FSL - Fast Simple Link
FSM - Finite State Machine
FOG - Fiber Optic Gyroscope
FTIR -Fourier Transform Infrared
FPU -Floating Point Unit
FF - Flip Flop
GA - Genetic Algorithm
GE - Genetic Evolution
GPIO - General Purpose Input and Output

GPU - Graphic Processing Unit
GPP - General Purpose Processor
GPS - Global Positioning System
HDTV - High Definition Television
HEVC -High Efficiency Video Coding
HIL - Hardware-in-the-Loop
HDL - Hardware Description Language
HW - Hardware
I/O - Input-Output
IBM - International Business Machines
IC - Integrated Circuit
ICON -Integrated Connect
IDE - Integrated Development Environment
IIR - Infinite Impulse Response Filter
ILA -Integrated Logic analyzer
ISE -Integrated Software Environment
ITU -International Telecommunication Union
JTAG - Joint Test Action Group
IDCT - Inverse Discrete Transform
IDWT -Inverse Discrete Wavelet Transform
OFDM -Orthogonal Frequency Division Multiplexing
IPCM - Intra Pulse Code Modulation
IQ - Inverse Quantization
IT - Inverse Transformation
ITU - International Telecommunication Union
ISO - International Standard Organization
IP - Intellectual Property
IPIF - Intellectual Peripheral Interface
IPIC - Intellectual Peripheral Interconnect
OTS - Off-the-Shelf
IMU-Inertial Measurement Unit
JVT -Joint Video Team
KF - Kalman Filter
LMB - Local Memory Block
LFSR -Linear Feedback Shift Register

LOD - Level of Decomposition
LL - Local Link interface
LUT - Look Up Table
LPM -Longest Prefix Match
LMS -Least Mean Square
HEVC -High Efficiency Video Coding
MAC - Multiply and Accumulate
MMKF - Multiple Model Kalman Filter
MB - Microblaze
μC - Micro-Controllers
MC - Motion Compensation
MCI - Memory Controller Interface
MIG -Memory Interface Generator
ME - Motion Estimation
MEMS - Micro electric Miniaturize system
MFPE - Multi Function Processing Element
MSE - Mean Square Error
MHz - Mega Hertz
MPEG - Motion Picture expert Graphic
MVP - Motion Vector Prediction
MVD - Motion Vector Difference
MV - Motion Vector
MVC - Multi View Coding
MPMC - Multi Port Memory Controller
MMKF - Multiple model Kalman filter
MRA - Multi Resolution Analysis
MUX - Multiplexer
MVC -Multi Video Coding
NAL - Network Abstraction Layer
NoC - Network on Chip
PC - Personal Computer
PSNR - Peak Signal to Noise Ratio
PLB - Processor Local Bus
PLL -Phased Locked Loop
PPC - Power PC Processor

PSO - Particle Swarm Optimization
PSoC - Programmable System on Chip
QCIF - Quarter Common Intermediate Format
QP - Quantization Parameter
RISC - Reduced Instruction Set Computer
RLE - Run Length Encoding
RAM -Random Access Memory
RBSP - Raw Byte Sequence Payload
RNG -Random Number Generation
RTOS -Real Time Operating System
SoC - System on Chip
STD - Standard Deviation
SNR - Signal to Noise Ratio
SRAM - Static Random Access Memory
SDK - Software Development Kit
SAD - Sum of Absolute Difference
SDTV - Standard Definition Television
SDK - Software Development Kit
SDRAM - Single Data Rate RAM
SURE - Steins Unbiased Risk Estimate
SVC - Scalable Video Coding
SW - Software
SU -Slave Unit
SPI -Serial Peripheral Interface
UART - Universal Synchronous and Asynchronous Receive and Transmit
UDI - User Defined Instructions
VGA - Video Graphic Array
VBS -Variable Block Shape
VLSI-Very Large Scale Integration
VLE- Variable Length Coding
VHDL-VHSIC Hardware Description Language
VCL - Video Coding Layer
XPS - Xilinx Platform Studio
XST - Xilinx Synthesis Tool

List of Publications

I. Research Papers published in International Refereed Journals

1. **Rangababu Peesapati**, Samrat L. Sabat, Kiran kumar Anumandla, Palani Karthik Kandyala, and Nayak Jagnnath *FPGA based Embedded Co-Processor for Real time Fiber Optic Gyroscope signal denoising*, International Journal of Digital Signal Processing, Elsevier, 2013.
   (in press, doi:10.1016/j.dsp.2013.04.010), **IF 1.91**.

2. **Rangababu Peesapati**, Samrat L. Sabat, K.P. Karthik, M. Narasimhappa, N. Giribabu, and J. Nayak *FPGA based Embedded platform for Fiber Optic Gyroscope signal denoising*, International Journal of Circuit Theory and Applications, Willey Blackwell, 2012.
   (in press, doi:10.1002/cta.1883), **IF 1.29**.

3. Kiran kumar. Anumandla, **Rangababu Peesapati**, Samrat L. Sabat, Siba K. Udgata and Ajith Abraham, *FPGA based Differential Evolution Co-processor: A case study of spectrum allocation in cognitive radio network*, International Journal of Computer and Digital Techniques (IET), IET, 2013.
   (in press, doi:10.1049/iet-cdt.2012.0109), **IF 0.9**.

4. **Rangababu Peesapati**, Samrat L. Sabat, K.P. Karthik, N. Giribabu and J. Nayak, *Efficient Hybrid Kalman Filter for denoising Fiber Optic Gyroscope Signal*, International Journal of Optik, Elsevier, 2013.
   (in press, doi :10.1016/j.ijleo.2013.02.013), **IF 0.51**.

5. Kiran kumar Anumandla, **Rangababu Peesapati**, Samrat L. Sabat and Siba K. Udgata *FPGA-based implementation of the Differential Evolution for embedded applications with a case study on system identification*, International Journal of Design Automation and Embedded Systems, Springer, 2013.**(in press, doi: 10.1007/s10617-013-9107- 4)**, **IF 0.26**.

II. Research Papers published in Peer Reviewed Int. Conferences

1. Narasimhappa.M, **Rangababu P.**, Samrat L. Sabat and Jagannath Nayak *A Modified Sage–Husa Adaptive Kalman filter method for denoising of Fiber Optic Gyroscope signal*, In proceedings of International Conference on Engineering Sustainable Solutions (INDICON), 2012, India, pp. 1266-1271.

2. K.P. Karthik, **Rangababu P.** and Samrat L. Sabat, *System on Chip implementation of Adaptive moving average based multiple-model Kalman filter for denoising Fiber Optic Gyroscope signal*, In proceedings of International Symposium on Electronics System Design (ISED), 2011, India, pp. 170-175.

3. Samrat L. Sabat, **Rangababu P.** and K.P. Karthik, *System on chip implementation of 1-D Wavelet transform based denoising of Fiber Optic Gyroscope signal on FPGA*, In proceedings of International Conference on Engineering Sustainable Solutions (INDICON), 2011, India, pp. 155-162.

4. Samrat L. Sabat, Shravan Kumar K. and **Rangababu P.** *Differential Evolution Algorithm for Motion Estimation*, In proceedings of 5^{th} Multi-Disciplinary trends in Artificial Intelligence International Workshop on Artificial Intelligence (MIWAI), 2011, India, pp.309-316.

5. Samrat L. Sabat, AjayKumar D. and **Rangababu P.** *Reliable Data acquisition system*: In Proceedings of 2nd International Conference on Emerging Trends in Engineering and Technology (ICETET), 2009, India, pp. 392-396.

III. Research Papers Communicated to International Refereed Journals

1. **Rangababu Peesapati**, Samrat L. Sabat and Kirankumar Anumandla, *Performance evolution of fixed and floating point accelerators for Differential Evolution algorithm in SoC platform*, International Journal of Circuit Theory and Applications, Willey Blackwell, **(Revision submitted), 2013**.

2. **Rangababu Peesapati**, Samrat L. Sabat and Kirankumar Anumandla, *Performance evaluation of floating point Hardware Accelerator for Differential Evolution algorithm*, International Journal of Microprocessors and Microsystems, **(Revision submitted), 2013**.

3. **Rangababu Peesapati**, Samrat L.Sabat, *An FPGA based APU accelerator for H.264 decoder*, International Journal of Computers and Electrical Engineering, Elsevier, **(Under preparation)**.

Chapter 1

Introduction

In the recent years, signal processing applications like data streaming, image processing, signal denoising, video decoding etc., are used in a wide range of electronic devices as embedded applications. The execution time of the Digital Signal Processing (DSP) algorithm increases with the complexity of the application. This limits its real-time applications in low end embedded processors. Embedded applications are constrained by space, power, and cost [8, 9]. Now-a-days due to the advantages of reconfigurability, low design costs and time-to-market, DSP applications are developed using FPGAs [1, 10]. During the last decade, modern FPGAs are available with soft and hard embedded processor cores to enable the designer for building complex embedded applications. To enhance the execution speed of the algorithm, dedicated hardware accelerators have been developed and interfaced with the processor as a coprocessor in System on Chip (SoC) platform.

There are several choices for platform selection such as Micro-Controllers (μC), Digital Signal Processors (DSP), FPGA and Application Specific Integrated Circuits (ASIC), for developing an embedded system. In order to achieve higher performance, applications need to be implemented either in multi-processors or in a dedicated hardware accelerators/coprocessors. The selection of platform depends on factors such as performance, power consumption, cost per chip, ease of tools accompanied by a specific platform to assist the developers for developing the system within the constraints of system cost and project time [11]. The platforms such as μC and DSP make use of embedded software oriented methodologies to develop the system. However, the designers who use FPGAs as their development

platform, have the flexibility to use either the processor-based approach, developing their system partly in firmware and partly in hardware, or developing their system entirely in the hardware [10]. In this work, FPGA based SoC design is chosen because of the following reasons:

1. Selecting off-the-shelf (OTS) microprocessor for a particular application which can meet all system requirements (like floating point arithmetic, power, speed, ease of tool) is time-consuming. So it is advantage to find an alternative which can allow designer to tailor a processor and a specific set of features and peripherals for the application to be implemented [11]. FPGA based design gives this flexibility to the designer compared to either μC or DSP processor based system.

2. The designer of FPGA based embedded system has flexibility to customize the design by adding any combination of peripherals and controllers. A unique set of peripherals can also be designed for specific applications, and the designer has privilege to add as many peripherals to meet the system requirements, which cannot be done in μC or DSP processor based system. Features which are not present in the initial phase of the design can also be added in the later part of the design [10].

3. Hardware and software concurrent development and co-existence on a single chip, is one of the compelling reason for choosing FPGA/SoC platform. If a segment of the algorithm is computationally complex then a custom co-processor (Auxiliary Processing Unit (APU)) can be designed to eliminate such problems [12, 13].

4. These kind of systems are preferred over ASIC based SoC solution, due to its re-programmability, Intellectual property (IP) reuse and cheaper development cost. Although ASIC has advantages over FPGA based SoC in terms of customized chip size, power, delay etc. FPGAs are feasible solutions for prototyping a device before building an ASIC chip [14].

5. FPGA enables selection of an optimal platform of SoC configuration for a typical application involving trade-offs between flexibility, cost, performance and power consumption.

1.1 Motivation

Research efforts have been made to design customized coprocessors for signal processing applications like signal denoising, evolutionary computation, video decoding in SoC platform.

Fiber Optic Gyroscope (FOG) sensor is used in Inertial Navigational System (INS) for measuring the angular rotation of an object. In general the gyroscope output is noisy, which effects the accuracy of measurement. Furthermore, the complex signal processing algorithms for denoising the signal with small footprint device demands to develop efficient algorithm and its hardware implementation to meet the cost, area and power requirements. For real-time applications, the signal processing algorithms need to be implemented in the hardware and integrate it to the FOG sensor board. Although digital signal processors are popular for implementing the signal processing algorithms. In present days, FPGAs are the popular choice for realization of DSP algorithms due to its affordable cost, reconfigurability and faster processing. This motivated the author to develop an efficient denoising algorithm for FOG sensor and implement the same in SoC platform.

Evolutionary Algorithms (EA) are used in real time embedded applications like motion estimation, on-line pole placement of digital filter among many others. These embedded applications suffer from the time-consuming evolution process of EA to derive an optimal solution. Very little work has been reported to enhance the execution speed of EAs for embedded applications. Thus there is a need to develop coprocessor for computing EA in SoC platform.

H.264 video decoder is the recent decoding standard used for video compression. Although it has better compression efficiency, it has higher computational complexity. This limits its implementation in general purpose and DSP processor for embedded application. Research efforts have been made to accelerate subtasks of the decoder by developing accelerators. However the overall acceleration decreases with increase in data communication between the processor and hardware modules. So there is a need to develop a coprocessor for the complete H.264 decoder in SoC platform.

Thesis objectives and contributions

The main objectives of this thesis are: i) to develop System on Chip (SoC) solutions for three different signal processing applications ranging from low to high complexities using Xilinx Virtex-5 FPGA, ii) analyze the hardware acceleration achieved due to SoC implementation of all the three applications. Each signal processing application is considered as a case study.

The objective of the first case study is to minimize the gap between idea/ algorithm development and embedded system design. In this case, a suitable algorithm is developed for signal denoising and then a hardware Intellectual Property (IP) of the algorithm is developed. Later the hardware IP/accelerator is interfaced with the embedded processor and the hardware acceleration is evaluated.

In the second case study, a hardware accelerator for Differential Evolution (DE) algorithm is developed and interfaced as an Auxiliary processor unit (APU) with the embedded processor (PPC440). In general, because of the complexity of algorithm and the fitness function, DE algorithm is executed either in the high-end processor or in Graphic Processing Unit (GPU). In this case-study, the performances (execution speed, power) of APU interface are compared with the Slave unit (SU) interface. Furthermore, a system identification application is implemented using the fixed/float DE IPs and their hardware acceleration is evaluated.

In the third case study, a hardware accelerator for H.264/Advanced video coding (AVC) is developed and interfaced with the PPC440 embedded processor using the APU interface. The hardware acceleration due to the developed IP is evaluated. The IPs of first, second and third case studies have achieved an acceleration of (65x, 30x-230x and 6-7x) compared to its equivalent software implementation. In summary, the work presented in this thesis concentrates on developing coprocessors for three signal processing applications of different complexity.

Thesis organization

The thesis is organized as follows: Chapter 2 describes the background of the thesis that includes SoC design flow and concepts. Chapter 3 provides the first case study of the work. It presents a detail analysis of denoising algorithm and its SoC implementation. Chapter 4 presents the second case study of the work. It presents the detail hardware implementation of Differential Evolution algorithm with an application to IIR filter system identification. Chapter 5 presents the third case study of the work. It presents the coprocessor architecture of H.264 video decoder and its SoC implementation. Chapter 6 presents the conclusions and future scopes of the work.

Chapter 2

Background

This chapter presents an introduction to the System on Chip (SoC) design using Field Programmable Gate Array (FPGA), SoC design flow along with the interface details.

2.1 Field Programmable Gate Array (FPGA)

FPGAs are prefabricated programmable logic devices composed of lookup table based programmable logic blocks connected by a programmable routing network [15]. These devices are programmed on field as opposed to devices whose internal functionality is fixed and hardwired by the manufacturer such as Application Specific Integrated Circuits (ASICs) and Application-Specific Standard Parts (ASSPs) [16]. These devices have the resources such as Flip-Flops (FF), Random Access Memory (RAM), Multiply and Accumulate (MAC), DSP48E and microprocessor cores. Due to this, FPGAs are used in embedded as well as Digital Signal Processing (DSP) applications that require massive parallelism, lower turnaround time and low cost etc. The traditional approach for designing the hardware of a system involves developing an Intellectual Property (IP) of the system/subsystems using Hardware Description Languages (HDL) [1]. But many DSP applications require hardware software codesign approach for meeting the real-time specifications. So the alternate choice is to implement the design in the FPGA based SoC platform where the hardware can be implemented in the hardwire logic and software can be implemented in the soft/hard embedded processor core of the FPGA. In the PSoC platform, processors, various standard peripherals and custom designed pe-

ripherals are connected in a single chip. Design of embedded systems for signal processing applications with IPs and on-chip processor cores is a challenging task. A typical architecture of a FPGA is shown in Figure.2.1

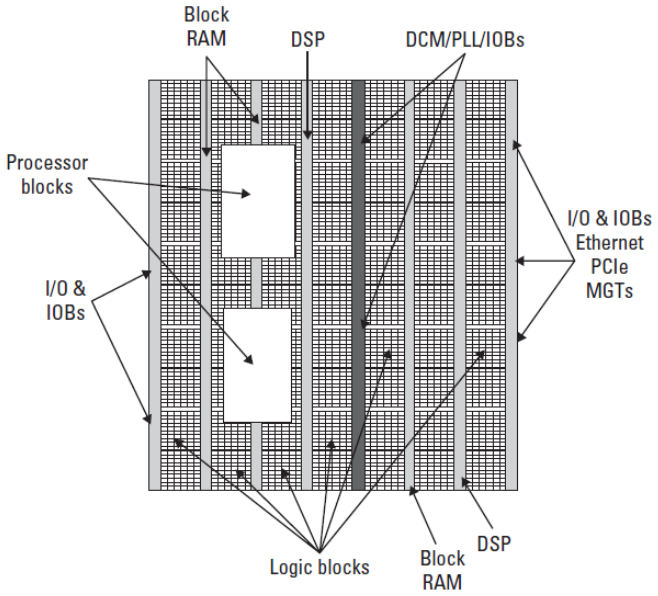


Figure 2.1: FPGA architecture [1]

2.2 Hardware Software Co-design

In general, signal processing applications are developed using DSP processor which has dedicated hardware blocks for certain computations like MAC, multipliers, dividers etc. The disadvantage of using DSP processor is that it executes the instructions in a sequential manner thus limits the speed of the design. So in order to speed up the processing of an application, parallelization is needed. This can be achieved using various ways like multi-threading of application, Graphic Processing Unit (GPU), hardware design using FPGA etc. The main difference between execution of hardware and software tasks is concurrency, which allows the hardware to execute a task much faster than the software in a processor [17].

The designed hardware can be further accelerated by making use of the parallel and pipelined architecture techniques. This would not be possible in a General Purpose Processors (GPP) and DSP processors which performs computational tasks in software by executing the application sequentially [2, 18]. The differ-

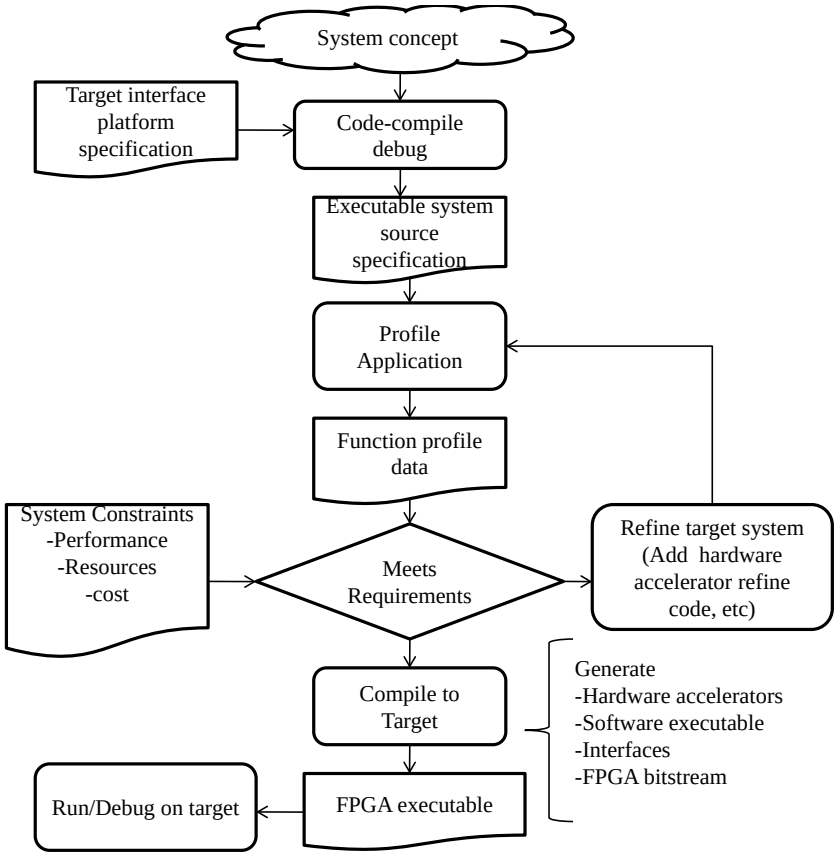


Figure 2.2: Hardware software codesign [2]

ence between these platforms are tabulated in Table.2.1. HW/SW codesign is a popular approach being used to accelerate a computational intensive DSP applications. In codesign approach, most time consuming tasks/subtasks of the application/algorithms are implemented in the hardware while the less computational intense tasks/subtasks are implemented in the embedded processor. In codesign, partitioning of the algorithm into software (SW) and hardware (HW) is a critical task [19]. The partitioning of HW and SW is decided by the profiling results of the application as shown in Figure.2.2. During the profiling, the computational intense tasks/subtasks are identified. Subsequently these are designed and implemented either in the hardwired logic or by using a dedicated coprocesser units [2, 20]. FPGA based embedded system design is still relatively new compared to standard processors. So the software design tools are relatively immature and difficult to debug the entire system [11, 21]. There are prominent issues like IP interface, cross clock handling and memory management that imposes design bottlenecks

in SoC design. So in order to resolve these things new design methodologies and easier integration methods are needed.

Table 2.1: Comparison of Processor/DSP, FPGA, ASIC based solutions [6]

characteristic	**Processor/DSP**	**FPGA**	**ASIC**
Programmability	High	High	Low
Development cycle	HW+SW	HW+SW	HW
Area efficiency	Medium	Low	High
Power efficiency	Medium	Low	High
Performance	Low	Medium	High

2.3 Hardware accelerator

A hardware that accelerates the execution of a task in a separate unit other than processor is referred as hardware accelerator [18, 22]. The accelerators can be designed using ASIC or FPGA approach depending on the specification of applications. ASIC-based accelerators cannot be usually leveraged by a gained speedup due to larger design development costs and longer development cycle [1]. Moreover, an accelerator designed for a specific application cannot be utilized for another application. With the advent of SoC platform using FPGA, the situation is changed. The hardware rigidity is lowered by re-programmability of FPGA devices in SoC platform and it allows interfacing of designed hardware IPs with the processors for desired DSP applications. The ability of on-demand FPGA reconfiguration also enables the accelerator to adapt to the actual needs of an application executed in the processor [1, 19]. Several issues need to be taken into consideration while designing a SoC system. The important issues are processor to accelerator interface, mutual communication and synchronization [12, 17]. All these effects have crucial impact on the acceleration. The former issue comprises of communication protocol and amount of data transfer. Selection of a proper communication/interface scheme affects the quantity of the data transfer from processor to coprocessor. The improper bus interface may result slowdown of the accelerated system [18, 22]. Another issue is to synchronize the data transfer between processor and hardware unit by handshaking, direct or interrupt mechanism. The selection of the particular synchronization system depends on the

chosen communication granularity and scheduling of the algorithm. The coprocessors can be interfaced with the processors using three different techniques i) System bus connected, ii) I/O connected, and iii) Instruction-pipeline connected [12, 17].

System bus connected

In this approach accelerators are interfaced to the processor as a slave peripheral/slave unit(SU) using system bus i.e. Processor Local Bus (PLB) as shown in Figure.2.4. The accelerator can transfer data and send commands to the processor through the system bus. Typically a single/multi data transactions, consumes many processor cycles due to bus arbitration [12]. These kind of systems have two major bottlenecks i.e. insufficient peripheral bus throughput and bus arbitration, this leads to data transfer and synchronization overheads which in turn lowers the execution speed.

I/O connected

In this approach, the accelerators are interfaced directly to a dedicated I/O port of a processor. In order to reduce the bus overhead and arbitration, a dedicated First-In-First-Out (FIFO) type of interface like Fast Simple Link (FSL) in Microblaze (MB) processor is used. These interfaces are typically clocked faster than the processor bus [12, 23]. Often data and control are typically provided through *GET* or *PUT* instructions. This enables the bus interface to have lower latency and higher data rate compared to the system bus.

Instruction Pipeline connected

In this approach, the accelerators with desired computing core are interfaced directly to the processor. Being coupled to the instruction pipeline, instructions not recognized by the CPU can be executed by the co-processor [12]. This type of accelerators expose no communication overhead and offers quick synchronization between the processor and coprocessor. The bottleneck is the implementation of the acceleration unit itself. If the critical path of the whole system goes through

the acceleration unit, then the whole processor will decrease its operational speed [12, 18]. Recent FPGAs include processors like ARM, PowerPC family processors, which utilize both specialized functional units and instruction set extensions for interfacing an IP. The APU interface is capable of transferring higher data volumes per second, approaching to the speed of Direct Memory Access (DMA) [1].

2.4 Related works

Broadly, there are three different methods (as mentioned above) being used for developing the coprocessors to accelerate the execution speed of a computation intense task. The related literature reports that for coprocessor design, the methodology remains same but tool environment, processors and bus interfaces are vendor specific. In this work we have used instruction pipeline approach for developing the coprocessor for all the three case studies. There are limited works reported in the literature about the development of coprocessor for accelerating a specific task [22, 24]. Xilinx has developed a coprocessor for Inverse Discrete Cosine Transform (IDCT) algorithm of Motion Picture Expert Graphic (MPEG-2) video decoder [23, 25] in a PowerPC405 and Microblaze (MB) processor based SoC platform. Also several other algorithms like Finite Impulse Response (FIR) filter and cordic algorithms are implemented in FPGA which helps to realize smaller DSP applications [26, 27]. Longest Prefix Match (LPM) algorithm is implemented as an IP in SoC platform [28]. This is used to search IP addresses in a routing table and to find a forwarding path for the incoming IP address.

Paolo Zicari et al., has developed a coprocessor for performing matrix product accelerator in both MB and PowerPC405 based SoC in Virtex-II Pro with all different peripherals interfaces. It is demonstrated that APU coprocessor accelerated the design by 9x [29]. Xu Guo et al., developed a 64-bit floating point mathematical operations like multiplication, addition and square roots functions for a public-key crypto systems using Virtex-5 FPGA [30]. Nandy et al., developed a reconfigurable high-performance low-power filter coprocessor for DSP applications. This has the option of reconfiguration to support a wide variety of filtering computations [31]. Michalis et al., developed a coprocessor namely Flexible Computational Components (FCC) that can realize any two-level template of primitive operations of data path. The effectiveness of coprocessor is evaluated in several

real-world DSP applications like Joint Picture Expert Group (JPEG) encoder, Orthogonal Frequency Division Multiplexing (OFDM) transmitter, data compression. The reported overall speedups are in the range of (1.75 to 3.95)x, having an average value of 2.72x [32]. Wassner et al., developed a coprocessor to accelerate the candidate operations of a video content analysis algorithm in Virtex-5 FPGA platform. Results indicated that with a relatively small degree of parallelism, corresponding to modest hardware cost, the overall frame rate is increased to a range of 18 and 105% depending on the processing and application parameters [33]. Mingas et al., developed a coprocessor of genetic algorithm for scan matching simultaneous localization and mapping problem in autonomous robotic application using Virtex-II Pro FPGA. The reported architecture has achieved an acceleration of up to 14.83x compared to the equivalent software implementation in PowerPC405 processor [34]. Vera et al., developed a wavelet coprocessor using model based design technique with PowerPC405 and MB processors [35]. Elhossini et al., developed a Least Mean Square (LMS) adaptive filter for audio signal processing in SoC platform. This platform also uses on board AC97 audio codec in Virtex-II FPGA and achieved speed up to 3.86x [36]. Bekker et al., developed a FPGA based Fourier Transform Infrared (FTIR) spectrometer to measure the Mars atmospheric composition using solar occultation from orbit, in which the design is ported into the hardcore PowerPC processor of Virtex 5-FPGA. By enabling the Floating Point Unit (FPU) of the APU an acceleration of 4x is reported [9, 37, 38]. Several other works for on board signal processing for space applications have also reported the advantages of FPGA based SoC system [8, 39]. Fons et al., developed a FPGA based runtime reconfigurable coprocessor for computational applications using dynamic partial reconfiguration approach and is validated by executing several signal processing algorithms [40, 41, 42].

In present days complex applications need a set of tools with a certain degree of abstraction where, only a description of the final implementation and their desired behavior is described in C [43, 44]. Several software applications are accelerated using impulseC, AUTOESL tools which converts C specification into HDL, later it use as an accelerator in SoC platform. Several different DSP applications like FIR filter, image texturing, edge detection encryption, Electro-Cardio-Graphy (ECG) applications are also developed using the above tools [13, 45, 46, 47]. AUTOESL tool is used for several complex DSP applications like H.264 decoder, FFT etc., [47]. These tools have an advantage of smaller design time frame but debugging

and modifying the design is challenging and cost effective. So the literature survey shows that developing coprocessor for different signal processing applications is a quite demanding area of research. In particular accelerating them using FPGA based SoC design is explored by a limited group of researchers.

2.5 Programmable System on Chip design

FPGA are no longer act as only glue logic resource in a complex hardware system, because the modern FPGAs have processor units along with glue logic as processing elements [1, 48]. This way traditional system on board design has been replaced by SoC design. If the designer need to develop SoC in a reconfigurable approach then FPGA device is used and the platform is named as "Programmable System on Chip (PSoC)". It is an integrated system with processor, peripherals, memories, custom Intellectual Property (IP) components on a single Integrated Circuit (IC) like FPGA [49]. With a provision of including operating system, such as Linux, these systems begin to appear more like a desktop Personal Computer (PC) in terms of functionality and capability on a single IC chip. The general architecture of PSoC platform is given in Figure.2.4.

Y-chart scheme is the common approach for implementing a designed in SoC platform [50, 51]. Firstly, the designer characterizes the target application (applications), and partition it to map the application onto the different architectural components. Then different performance measures are evaluated. After satisfactory performance figures are achieved the designer follows the architecture else the architecture is reconfigured. This section presents the development of SoC system using PowerPC440 processor in Virtex-5 FPGA development board using Xilinx EDK. The procedure to implement a design in this platform is described below.

The PSoC platform can be designed using Xilinx platform studio (XPS) in Xilinx EDK and SDK as shown in Figure.2.3. In has two different design steps namely hardware (HW) and software (SW). These two steps run in parallel. In HW, Base System Builder (BSB) wizard provides an efficient way to create the FPGA based embedded system. The choice of the memory types, memory controllers, peripherals, peripheral controllers, size and type of instruction and data cache memories, and size of local memory, choice of processor, bus and peripheral clock frequency are configured in BSB [16]. The proposed FPGA based SoC system incorporate a coprocessing unit interfaced to the PLB or APU as shown

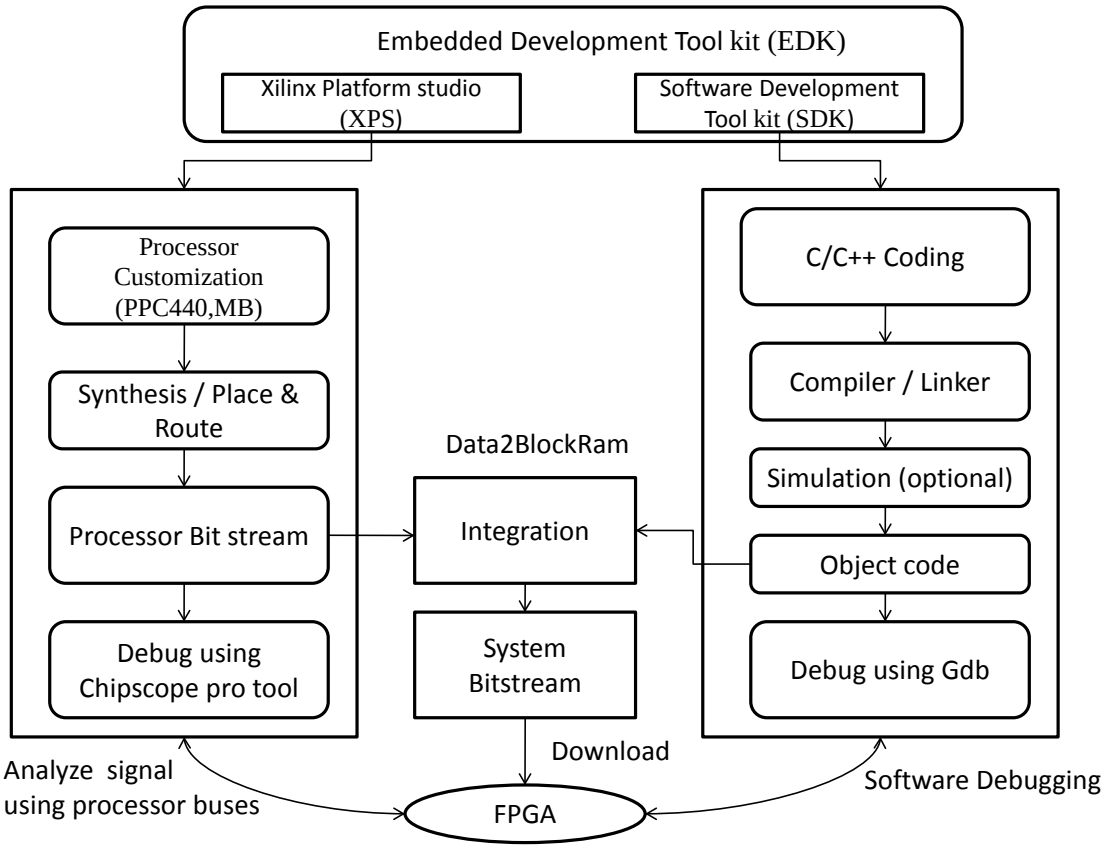


Figure 2.3: Hardware software codesign approach using embedded development kit [3]

in Figure.2.4. On-chip memory and external memory are used for initializing the processor program. The Universal Synchronous and Asynchronous Receive and Transmit (UART) and Joint Test Action Group (JTAG) ports are used to monitor, debug and download the bitstream on to the FPGA. The proposed custom/DSP IP in SoC has design objectives of high-speed in terms of operating frequency and reduced cost in terms of FPGA fabric resources. During simulation, synthesis and compilation of the embedded processor system, an appropriate optimization scheme must be selected to achieve the above design objectives [16]. The processor internal timer along with interrupt is used for measuring the execution time of the algorithm. In this thesis we considered a custom coprocessor designed specifically to target the implementation of DSP algorithm in the design. This is more advantages in applications like FOG denoising, DE system identification, H.264 video decoding. The software SDK tool is used to debug the design. In this dissertation,

PPC440 processor is used because it outperforms the MB processor in terms of processing speed and hardware resource utilization [16, 52].

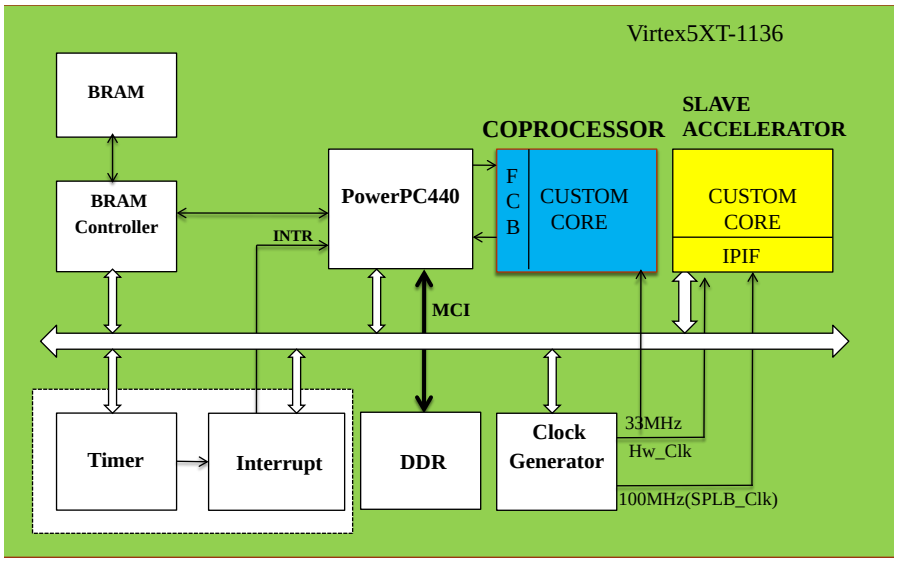


Figure 2.4: Basic SoC system

2.5.1 Embedded Processors

Embedded applications demand on-chip processor core in FPGA to develop single chip solution. Two types of microprocessor cores that are embedded in the FPGA, a) soft processor like Altera Nios II [53] or Xilinx MB [54] and b) hard processor like IBM PowerPC 440/405 [55, 56] in Xilinx Virtex-5 FXT / Virtex-II Pro FPGA family. These cores (soft or hard) help to reduce the footprint of the design, power, cost and time to market.

2.5.2 Memory

The performance of SoC platform depends upon how well the memory is organized. FPGA based SoC system is a memory mapped system in which each peripheral is having some address. Memory attached to SoC system can be divided into internal, external and cache memories [16]. The basic primitive of internal memory is Block RAM with 148 Kbytes for Virtex-5 FPGA. However this size changes from FPGA to FPGA. Similarly, DDR2SDRAM of size 256Mbyte and 1Mbyte of Static Random Access Memory (SRAM) are used as off-chip external memories.

The external memory access time is high compared to on chip memory and SRAM. These can be configured using Multi-Port Memory Controller (MPMC)/ Memory Controller Interface (MCI) in SoC platform. The use of external memory degrades the throughput of design. In the PSoC, if application program fits entirely within the local memory, then the design is likely to achieve optimal memory performance, although it is mostly likely that the embedded programs will exceed the local memory capacity. The program/data memory usage in SoC can be manipulated using the Linker Script of system. This can be mapped either into internal memory such BRAM or external memory.

2.5.3 Peripherals

The objective of the peripheral is to communicate with the processor through different buses. Generally peripherals are classified into two types (a) generic and (b) custom peripherals. Generic peripherals are available as a soft-core or hard-core and are configured during the SoC design phase [57]. The hard-core peripherals are implemented in silicon whereas soft/custom logic peripherals are implemented in the fabric. Custom peripherals are developed by the designer according to the application. It is often in soft logic form developed using HDL language. Some of the examples of custom peripherals are IDCT [25], FFT [58], FIR [59] filter etc. These can be interfaced using various bus interface techniques like SU and APU. In this thesis AMADMKF, DE and H.264 IPs are developed as the part of the work. The details of some of the main standard peripherals used in this work are given below.

Central Direct Memory Access Controller (CDMA)

Some peripherals of SoC might have Direct Memory Access (DMA) capability to improve data bandwidth and performance. Other peripherals might rely on a separate DMA engine to provide this improved data bandwidth between the peripheral and memory. The direct memory access does not necessarily transfer data from different locations of external memory. In fact, transfers between the hardware core on-chip memory to off-chip memory and vice-versa. The central DMA controller in SoC system includes both a master and a slave bus interface [1, 4].

Universal Synchronous and Asynchronous Receive and Transmit (UART)

This peripheral is essentially a parallel-to-serial shift register. This is used in the system for monitoring, debugging and transferring the input/output data between the processor and monitor [1].

Digital Clock Manager (DCM)

Most systems have a single external clock that produces a fixed clock frequency [4, 60]. However, in SoC different modules need to operate at different frequencies such as processor and memory at 200 MHz and custom logic at 50MHz. A Digital Clock Manager (DCM) or Phased Locked Loop (PLL) allows to generate different clock periods from a single reference clock.

2.5.4 Bus interfaces

The processor(s), memory controller(s) and peripheral(s) are connected to the system bus. The interface logic is specific to the particular bus. The bus interface includes a bus arbiter, which controls access to the bus. More details of buses interfaces are available in [4, 61]. In multiple bus design, the bus with the highest bandwidth is connected with the processor, memory controller, and UART [62].

Slave Unit (SU) interface

In SU interface, the IP/ custom peripherals are interfaced with the Processor Local Bus (PLB). The custom peripherals are designed by making use of Intellectual Peripheral Interface (IPIF). It provides optional services through Intellectual Peripheral Interconnect (IPIC) which includes Register, FIFO, DMA, software reset and interrupt support. The user can design peripherals using these flexibilities. The IPIF utilizes the register interface and it takes advantage of the centralized address decoding [63]. There are two ways to connect the IP i.e. either as master or slave peripherals [1, 4]. The master peripheral is initiating communication to other peripheral like processor, whereas Slave Unit (SU) or slave peripherals follows the instructions of master peripheral. In this thesis we are concentrated on

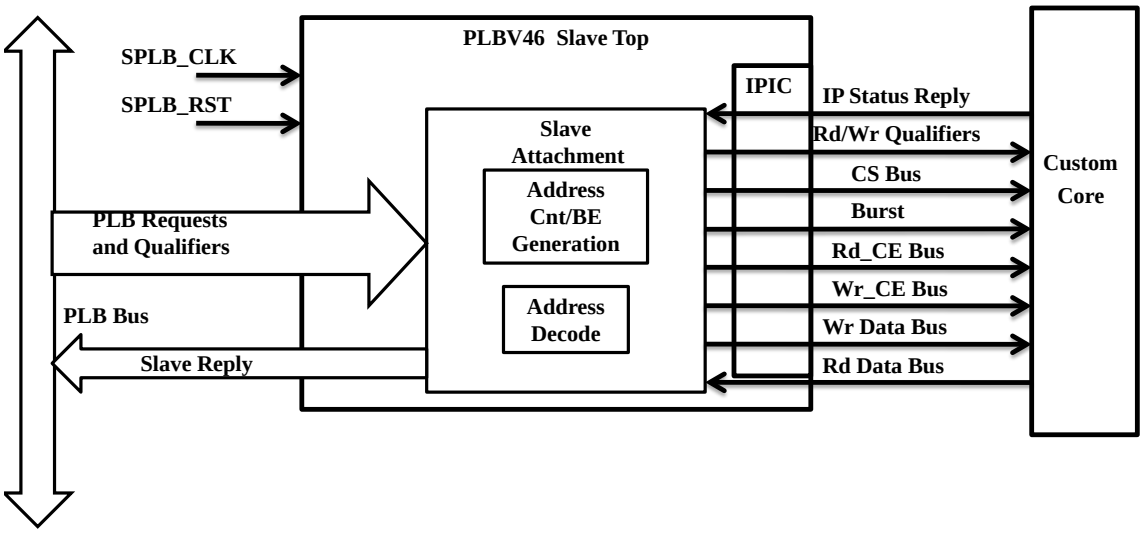


Figure 2.5: Processor Local Bus register interface [4]

designing the system using slave peripherals and coprocessors.

A slave IP core needs *in* and *out* data buses, a few slave registers, read/write request and acknowledge signals. The IP is designed with register interface having minimum 1 to maximum 4096 slave registers. Each register can have read/write operation which can interface any inputs of custom IP cores such as Multiplier, Adder, IDCT etc. The slave registers which are interfaced to the input and output of IP core can read/write the write port as shown in Figure.2.5. The IP accepts inputs from the bus (Bus2IP_data) depending on the number of slave registers. The associated read/write qualifiers are Bus2IP_RegRdCE and Bus2IP_RegWrCE respectively. The qualifiers becomes high during data read/write operation. For reading, the IP may drive non-zero data to IP2Bus_data whenever one of its register-read is active and it must drive valid data during the cycle that it asserts its acknowledgement. The output of IP core (32-bit), is connected to IP2Bus_data by its slave register [62]. The register interface, the Bus2IP_RdReq and Bus2IP_WrReq signals convey redundant information, but may have utility in allowing the IP to easily generate a one cycle acknowledgement, IP2Bus_RdAck or IP2Bus_WrAck, by delaying the corresponding request signal by an appropriate amount of time. So in this work these acknowledgement signals are not used to reduce the latency.

Device Control Register (DCR) interface

The DCR interface supports the PPC440 embedded processor for control and status accesses other peripherals. This interface is interlocked with control signals such that it can be connected to peripheral units which can be clocked at different frequencies from the embedded processor. The DCR interface also allows the PPC440 embedded processor to communicate with peripheral devices without using the PLB interface [1].

Auxiliary Processor Unit (APU) interface

The PPC440 processor in Virtex-5 FPGAs has a fabric coprocessor bus (FCB) (128 bit) through which custom peripherals are interfaced with the processor using an APU controller. The custom peripheral is invoked using the PPC440 extended instruction set i.e. *Load and Store*. This approach provides the flexibility of interfacing a coprocessor with the instruction pipeline [12]. The examples are floating-point unit or other custom/DSP IPs related to signal processing dot product or matrix multiplication [30, 38]. This coprocessor can execute the desired task concurrently with the PPC440 processors extended instructions. The APU controller synchronize the clocks of processor and custom IP and they can run at different frequencies. The APU controller decodes the processor instructions in a pipelined manner resulting faster execution of overall instructions. There are two major classes of Fabric Coprocessor Module (FCM) instructions, (a) storage instructions and (b) non storage instruction. In this work, storage instructions i.e. load and store are used. Non storage instructions includes floating point arithmetic instructions and User Defined Instructions (UDIs) based on opcodes. The APU controller can run at the same speed as the processor. The clock ratio between the processor and APU controller must be an integer multiples of the processor clock range from 1:1 to 16:1.

The key characteristics and features of the APU controller are listed below [4].

1. If FCM is pipelined, it can execute three instructions at a time that do not return data to the processor. This provides a low communication overhead in the instruction issue to the APU controller [4].

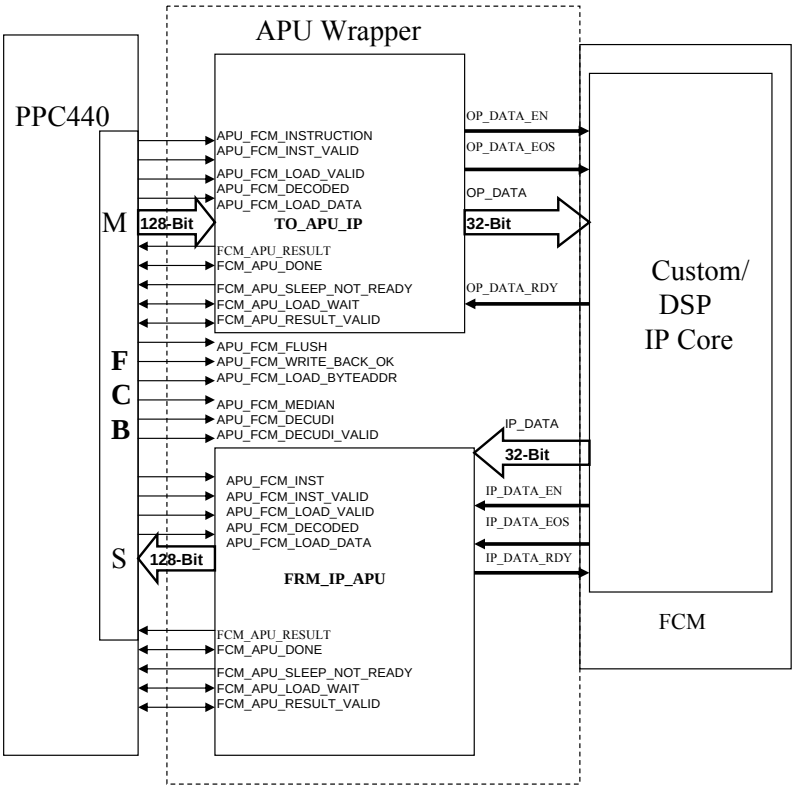


Figure 2.6: APU interface system for PPC440 processor

2. It decodes FCM load and store instructions with byte, halfword, word, double word, and quadword size data transfer.

3. It decodes FPU instructions as well and user defined instructions by using 16 UDIs by configuration registers using full primary and extended opcode [4].

The APU coprocessor using the PPC440 processor for custom DSP application IP (i.e. either denoised core or DE core) is shown in Figure.2.6. It has mainly three modules, i) PowerPC processor, ii) APU wrapper (iii) custom DSP IP core. The objective of the processor is to send and receive the data to and from the APU. The objective of APU wrapper is to interface the IP core with the processor, whereas the objective of the IP core is to process the signal. The APU wrapper contains two different modules namely IP_APU and APU_IP. The APU_IP module receives data from the processor and sends it to custom DSP IP whereas the IP_APU module receives the IP data from the custom DSP IP and sends it back to the processor (PPC440). The APU_IP receives 128 bit signal, but the IP has 32 bit width. The IP receives a full set of data in 4 clock cycles. Similarly the

IP_APU module receives 128 bit of data from the IP in 4 clock cycles. A Finite State Machine (FSM) with 5 states, i.e. load, load_valid, store, store_valid and idle states control the data between the processor, IP_APU and APU_IP. The APU uses some of the load/store input lines like $APU\_FCM\_INSTRUCTION$ and $APU\_FCM\_INSTRUCTION\_VALID$ etc., of the PowerPC processor for the entire interface [4]. The signals $APU\_FCM\_MENDIAN$, APU_FCM_DECUDI and $APU\_FCM\_DECUDI\_VALID$ are not used in this design. The APU wrapper of IP core using six control signals OP_DATA_EN ,OP_DATA_RDY, OP_DATA_EOS, IP_DATA_EN, IP_DATA_RDY, IP_DATA_EOS [49].

2.5.5 Related tools

The SoC development cycle has two different platforms a) Software development b) Hardware development.

Software development platform

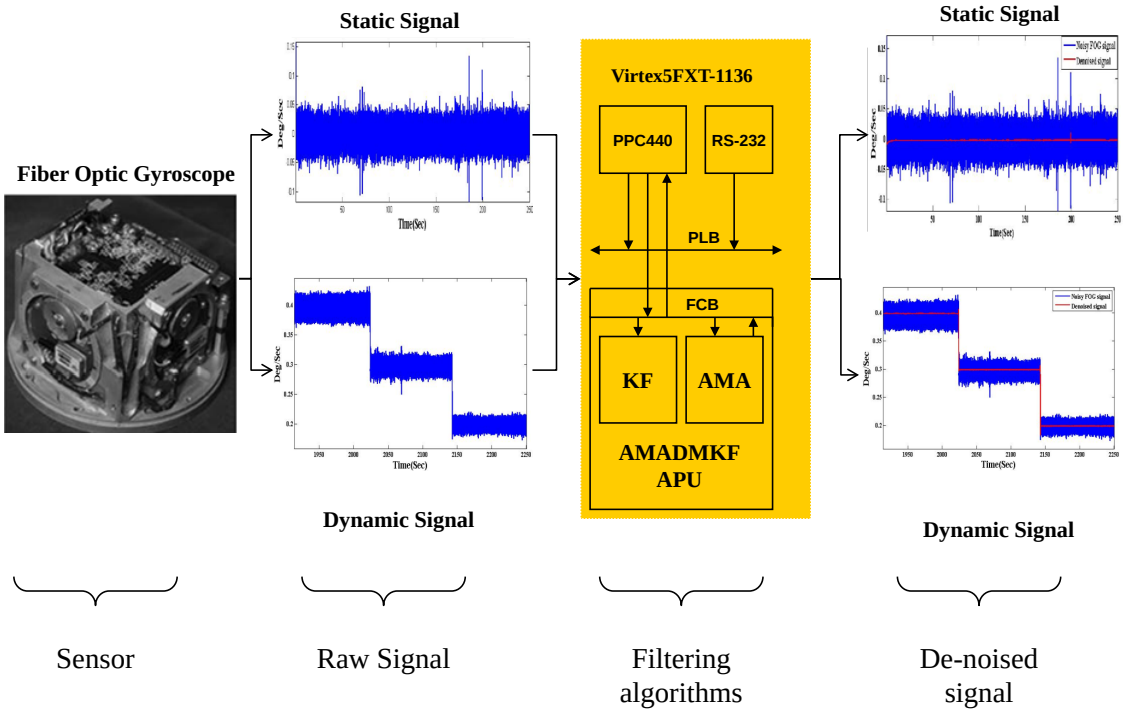
MATLAB and C programming language are used for algorithm development and validation. Ecllipse platform is used to verify H.264 decoder (software). Xilinx Software Development Kit (SDK) is used for profiling the algorithm in PPC440/MB processors.

Hardware development platform

The hardware development cycle uses various tools for HDL coding, simulation, synthesis, debugging. Aldec Rivera and Mentor graphics Modelsim simulation tools are used for coding and simulation. For hardware and SoC development Xilinx Integrated Development Environment (IDE) with Integrated Software Environment (ISE) [64] and Embedded Development Kit (EDK) [3] are used. EDK tool is used for building the SoC and generating custom peripherals [65].

Chapter 3

Coprocessor for Fiber Optic Gyroscope (FOG) signal denoising

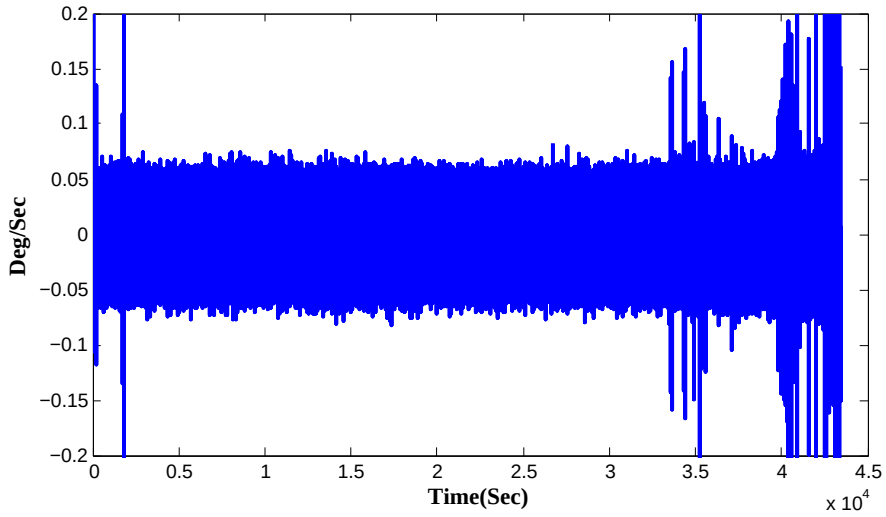


This chapter proposes a denoising algorithm for Fiber Optic Gyroscope (FOG) signal. The performance of this algorithm is compared with existing algorithms such as conventional Kalman Filter (KF), Discrete Wavelet Transform (DWT) with respect to Allan Variance analysis and Signal-to-Noise Ratio (SNR). Further a hardware Intellectual Property (IP) of the proposed algorithm is developed and its performance is evaluated. Finally a System on Chip (SoC) is built using the IP and hardware acceleration due to this configuration is reported.

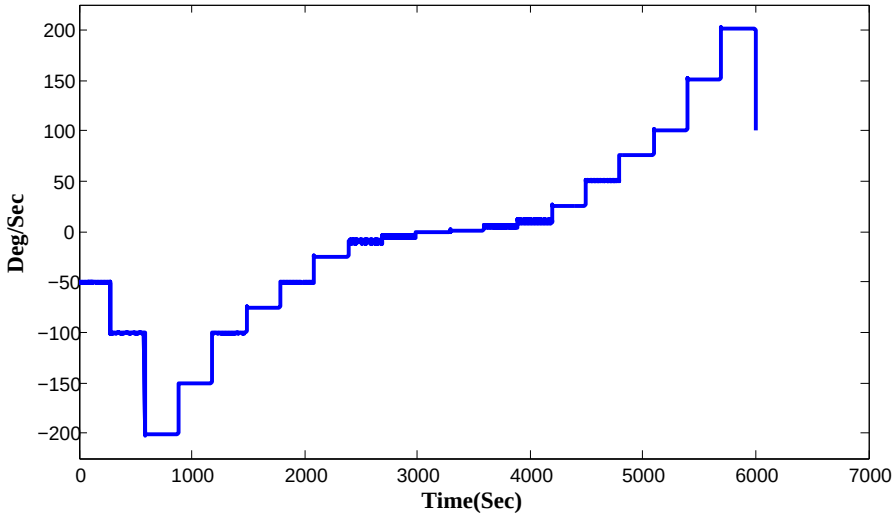
3.1 Introduction

There is an increasing demand for accurate, yet low-cost and highly reliable guidance, control, and navigation systems for measuring the direction and altitude of an object. Gyroscope is the core component for providing this information. Although different type of gyroscopes are available, Fiber Optic Gyroscope (FOG) is a proven technology for measuring angular velocity of an object. It has the advantages of low reaction time, wide dynamic range, high accuracy and reliability [66]. The basic operational principle of FOG is that the optical path difference induced by counter propagating beams in a rotating reference frame is proportional to the absolute rotation rate (Sagnac effect) [67]. When the system is at rest (static condition), the counter propagating beams traverse identical paths resulting to zero phase difference between them. In contrary, when the system rotates with an angular velocity Ω (dynamic condition), the path difference between the two beams will result in a phase difference which measures the rotation rate of the object. Since the rotation angle is evaluated by integrating the measured rotation rate over a period of time, any error in measuring the rotation rate will result error in rotating angle. This error results in long-term offset/ bias drift of FOG [66]. Hence prior to denoise the signal, understanding behavior of FOG signal is necessary.

The measured FOG signal is said to be static when there is no rotation rate or the gyro is in static condition. Similarly it is said to be dynamic when it rotates with a rate in Deg/sec or Deg/hr. The sensitivity of measured angular velocity is more important in the moving environment rather than in the static environment. The denoising of dynamic data is comparatively more complex than the static data as the changes in rotation rates make it more difficult to simultaneously denoise the signal and accurately track the rotation rates. Some times the



(a) FOG raw signal in static condition



(b) FOG raw signal in dynamic condition

Figure 3.1: FOG raw signals in static and dynamic condition

measured signal is noisy during the transitions. FOG signal in dynamic environment, can be approximated as a combination of signals oscillating about a mean value and signals transitioning between two mean values [68]. In this work the transition region is considered as the discontinuity region and remaining region is considered as stationary region. When the gyroscope is in stationary region, the measured signal is noisy and when the gyro changes its rotation rate i.e. in dynamic condition, the measured signal sees a sudden jump or transition in the

signal. Figure.3.1a and Figure.3.1b show the static and dynamic signals of a FOG.

In literature different signal processing algorithms such as Discrete Wavelet Transform (DWT) [69, 70, 71] and Kalman Filter (KF) with properly tuned gain [72, 73], Multiple Model Kalman Filter (MMKF) [74, 75, 76] are being used to denoise the FOG signal in static condition satisfactorily. However these algorithms fail to denoise the signal in dynamic condition. This is because neither the DWT nor the KF algorithm can capture the unpredictable abrupt changes of dynamic signal. Although an appreciable amount of literature is available for denoising gyroscope signals using KF, the problem of adjusting KF parameters for the real-time filtering application is still unsolved and tuning of these parameters varies from sensor to sensor, even from axis to axis. Different approaches such as optimization approach [77] and Adaptive Kalman Filter (AKF) [78] were used for denoising, but still improvements are required for denoising dynamic signal. In Kownacki et al., KF was applied to dynamic signal with a test step signal using optimized KF parameters [77]. These parameters were evaluated in advance and used for denoising the whole signal. Denoising algorithms like AKF, MMKF etc., had applied successfully for denoising the dynamic data of Global Positioning System (GPS) [79, 80], but these algorithms failed to denoise FOG dynamic signal. So development of an efficient denoising algorithm is necessary to improve the accuracy and performance of FOG.

Both manufacturers and users are keen to know about the source of the noise and the amount of noise present in the signal. Noise quantification helps the manufacturer to minimize the noise during the manufacturing process. Complementary to this, the user can also improve the navigation solution by making use of different denoising algorithms. The performance of the denoising algorithms are characterized by quantifying the random noises in the denoised signal using Allan Variance analysis. This analysis helps to detect the FOG signal random drift [66] as Quantization error (Q), Angle random walk error (N), Bias instability error (B), Rate random walk error (K) and Drift rate ramp (R) [81]. Recently, FPGA based Real-time embedded system is developed for vehicular navigation system, consisting of a gyroscope and an odometer or wheel encoders, along with a GPS receiver and KF [11]. In order to have on-board denoising of the FOG signal, the denoising algorithm need to be implemented in the hardware and integrated to the FOG sensor board. This chapter presents a new denoising algorithm namely, Adaptive Moving Average Dual Mode Kalman Filter (AMADMKF) for denoising FOG sig-

nal in both static and dynamic environment. This is a hybrid of the standard Adaptive Moving Average (AMA) and Kalman Filter (KF) algorithms. Now-a-days, SoC design solution replaces system on the board design concept [8]. So, apart from proposing an algorithm for denoisng the FOG signal under static and dynamic environment, we also implemented and tested the proposed algorithm in a FPGA based embedded system platform.

3.2 Denoising algorithms

This section presents different denoising algorithms such as Discrete Wavelet Transform (DWT), Kalman Filter (KF) and proposed algorithm for denoising FOG signal.

3.2.1 Discrete Wavelet Transform (DWT)

During the last decade Wavelet transform became a popular tool for signal processing applications. Most popularly, it is being applied for signal compression, feature extraction and signal denoising [82]. This work focus on the use of Discrete Wavelet Transform (DWT) for denoising FOG. Wavelet transform can be classified into two types i) Continues Wavelet Transform (CWT). ii) Discrete Wavelet Transform (DWT). The DWT of the signal can be computed using decomposition algorithm [83, 84]. In this algorithm at each level of decomposition, the signal is passed through a half band low pass and high pass filters, followed by down sampling by 2. The low frequency (approximated) coefficients contain rotation rate, whereas high frequency (detail) coefficients have the information about rotation rate with noise. Noise from high frequency coefficients are removed by using thresholding the detail coefficients. For reconstruction the signal, reverse process of decomposition is performed. The FOG output signal can be expressed as

$$y_n = x_n + e_n \tag{3.1}$$

where x_n, e_n are the true rotation rate and random noise respectively.

For denoising y_n the algorithmic steps is explained below [85]

1. Select a wavelet basis (Bi-orthogonal or orthogonal). Compute the approximation and detail wavelet coefficients of the signal at each decomposition

level. Let d_j, $j = 1, 2 \ldots J$ are the detail coefficients at jth decomposition level, where J is the maximum level of decomposition.

2. Compute the noise variance σ_j^2 at each level j =$1, 2..J$.

$$\sigma_j^2 = median(d_j)/0.6745 \tag{3.2}$$

3. Compute threshold T_j at each level j as

$$T_j = \sigma_j \sqrt{2ln2^j} \tag{3.3}$$

4. Select one of the level dependent threshold either (soft or hard). Apply the selected threshold to detail coefficients

$$d_j^S = \begin{cases} sgn(d_j)(|d_j| - T_j), & \text{if } d_j \geq T_j \\ 0, & \text{if } d_j < T_j \end{cases} \tag{3.4}$$

$$d_j^H = \begin{cases} d_j, & \text{if } d_j \geq T_j \\ 0, & \text{if } d_j < T_j \end{cases} \tag{3.5}$$

The selection of wavelet depends upon nature of the signal of interest and also its correlation with the signal. For selecting the level of decomposition the mean value of detail wavelet coefficient at each level is computed. Since, white noise has zero mean, maximum level of decomposition is same as the level up to which mean value of detail coefficient is zero. The choice of threshold selection rule also plays an important role in denoising the signal. There are four threshold selection rules, namely Rigrsure, Minimaxi, Sqtwolog and Heursure [82]. In this work, Sqtwolog and hard threshold are used because i) it has shown better denoising result than others and ii) it is easier to implement in the hardware.

3.2.2 Kalman Filter (KF)

Kalman filter uses state space representation of the input and various other parameters. Process noise covariance matrix (Q_k) and measurement noise covariance matrix (R_k) are the two parameters that effect the output of KF. There are several variants of the KF based on the way the values of Q_k and R_k are estimated

[86]. In the basic KF fixed values of Q_k and R_k are considered to filter the signal by assuming that the process noise and measurement noise characteristics are statistically known. The denoising process using KF has two stages [86]

1. The first stage predicts the current state and the error covariance based on the previous state estimation.

2. The second stage corrects the previous predicted values using the present measured value.

These stages are represented as

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{3.6}$$

$$P_k^- = AP_{k-1}A^T + Q_k \tag{3.7}$$

$$K_k = P_k^- H^T (HP_k^- H^T + R_k)^{-1} \tag{3.8}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{3.9}$$

$$P_k = (I - K_k H)P_k^- \tag{3.10}$$

where

x_k is the state vector at time k,

u_k is the optional control input at time k,

Q_k is the process noise at time k,

R_k is the measurement noise at time k,

z_k is the measurement taken at time k,

P_k represents error covariance matrix at time k,

A, B and H are state space representation matrices.

Q_k and R_k are discrete white noises having gaussian distribution with zero mean and covariances respectively.

The Kalman Gain (K_k) is the dominant parameter that affects the filter performance and is calculated based on the different KF parameters. A detailed description of the standard KF is given in [86]. For a fixed set of KF parameters, the Kalman gain and the error covariance converge to almost a constant value. Hence, the KF can be characterized by the Kalman gain alone and the output can be written as [86]

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{x}_k^-) \tag{3.11}$$

3.2.2.1 Adjusting KF parameters

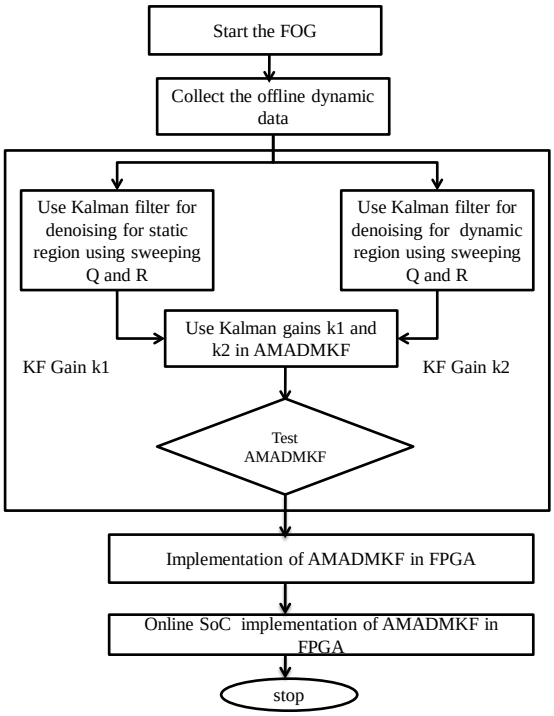
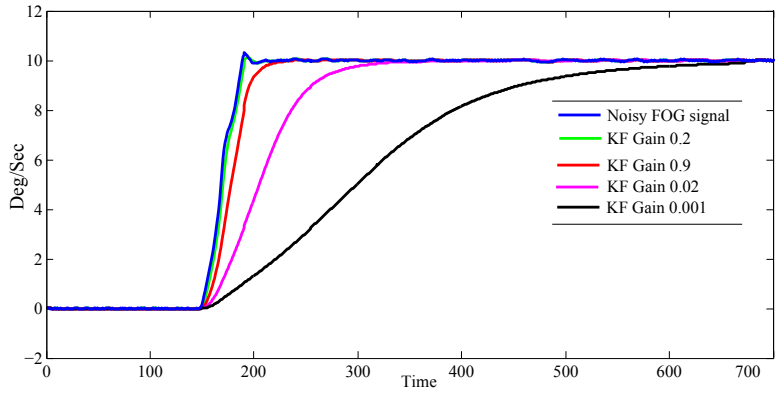


Figure 3.2: Adjusting KF parameters in off-line mode of the AMADMKF

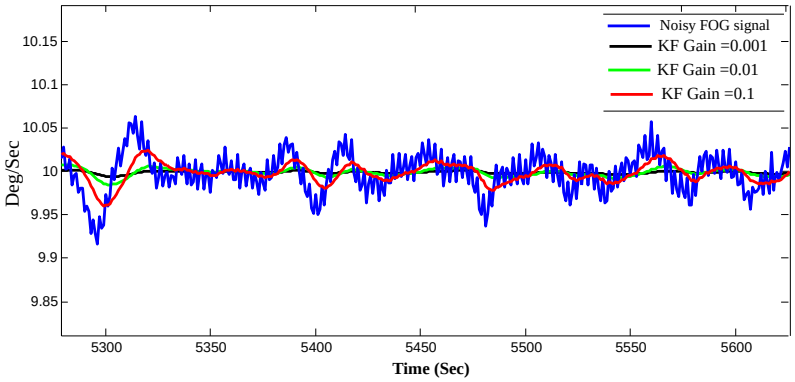
The Kalman gain (K_k) depends on the measurement noise covariance (R_k), process noise covariance (Q_k), and error covariance (P_k) matrices of the filter. The estimation of these parameters for real-time filtering is still unsolved [77]. These parameters values depend on the FOG dynamics which are not explicitly known a priori. The effective denoising can be achieved by appropriate initialization of R_k, Q_k and P_k [87, 88]. For denoising the dynamic noisy signal, there exits a trade-off between quality of denoising and following the trend of the signal. This depends on the proper value of Q_k/R_k. For the dual mode KF, Q_k/R_k is selected such that KF has a choice to choose the optimum value of Q_k/R_k for effective denoising and following the trend of the signal independently [89]. Since the algorithm has two variables (R_k, Q_k), it is not difficult to find the optimum values for the above desired conditions. These can be obtained off-line by sweeping the range of both the variables as shown in Figure.3.3a. The minima can be easily obtained from the sweep plot to fix the value of Q_k/R_k.

For static condition, KF gain $k1$ is calculated for the obtained values of Q_k and R_k using the sweep plot. The output of KF stabilizes after processing certain

number of samples as shown in Figure.3.3a and the denoising effect has shown in Figure.3.3b. This sample delay does not affect much in static condition, whereas in the dynamic condition, this delay is not tolerable. The KF parameters not only impacts the noise level of output signal, but also on the settling time of the KF. It is observed from Figure.3.3a that the settling time decreases with increase of KF gain and also decreases the denoising effect as shown in Figure.3.3b.



(a) Choosing KF gain parameters in transition region



(b) Choosing KF gain parameters in stationary region

Figure 3.3: Selection of KF gain in different conditions for dynamic FOG signal

This analysis shows that one KF parameter (Kalman gain) does not follow the signal trend but denoises the signal effectively and vice-versa. This is shown in Figure.3.3a and Figure.3.3b. These two results have different sets of Q_k and R_k values with the Kalman gains as $k1$ and $k2$. These values vary from FOG to FOG. So the analysis of KF algorithm for a particular FOG is an important step of the algorithm. In literature Dah-jing et al., evaluated the KF parameters using swarm intelligence technique [90], but in order to maintain a feasible solution for the hardware implementation, the gain parameters are estimated off-line as shown

in Figure.3.2.

3.3 Proposed Hybrid Kalman Filter (AMADMKF)

AMADMKF is a hybrid KF technique based on the Adaptive Moving Average (AMA) and Kalman Filter(KF). In this technique the moving average filter length is not constant, rather it adapts according to input signal. A long term moving average filter can denoise the signal effectively but it fails to denoise the signal if it has multiple discontinuities. To avoid this problem, adaptive filters have been used in which the length of moving average adapts according to rate of change in signal. The adaptive moving average technique is used to detect discontinuities in the signal [91]. The discontinuity locations are obtained using noise variance of the signal. The AMADMKF algorithm has two KFs with different gains which are obtained from off-line mode in previous subsection. One gain $k1$ is used when the FOG is on stationary and other $k2$ is used when the FOG changes its rotation rate. The implementation steps of AMADMKF algorithm is explained below

Implementation steps

1. Consider $N = 4096$ number of samples as one frame of the signal.

2. A q point simple moving average filter is applied to suppress the noise in the signal. The denoised output is

$$y_n = \frac{1}{2q+1} \sum_{j=-q}^{q} x_{n+j} \tag{3.12}$$

3. The noise is further reduced by applying an iterative adaptive moving average filter, in which the size of window varies adaptively. By using this filter the output signal Y_n is

$$Y_n = \frac{1}{q_h(n) + q_t(n)} \sum_{i=-q_t(n)}^{q_h(n)} y_{n+i} \tag{3.13}$$

where

$$q_h(n) = \begin{cases} q, & \text{if } D'(n) < 0 \\ f(D(n))q, & \text{if } D'(t) \geq 0 \end{cases} \tag{3.14}$$

$$q_t(n) = \begin{cases} q, & \text{if } D'(n) > 0 \\ f(D(n))q, & \text{if } D'(n) \leq 0 \end{cases} \tag{3.15}$$

$$f(D(n)) = 1 - \frac{D(n)}{max(D(n))} \tag{3.16}$$

$$D(n) = |y(n+q) - y(n-q)| \tag{3.17}$$

$$D'(n) = D(n+1) - D(n) \tag{3.18}$$

4. Iteratively obtain (z times, in this work we have considered z=3) the values of q_h and q_t by giving the adaptive filter output as the input. Select the values of q_h and q_t that correspond to the final iteration.

5. For obtaining the discontinuity location, the sample variance is compared with a threshold (λ). The variances of the samples starting from $(2q+1)^{th}$ sample to $(N-2q-1)^{th}$ sample is

$$\hat{\sigma_n^2} = \frac{\sum_{i=q_t}^{q_h} \{Y_i - \overline{Y}_n\}^2}{q_t + q_h} \tag{3.19}$$

6. Compute the threshold (λ) from as [91]

$$\lambda = \frac{1}{\eta} * (min(\hat{\sigma}_{2q+1}^2, \hat{\sigma}_{2q+2}^2, \hat{\sigma}_{2q+3}^2, ..., \hat{\sigma}_{N-2q-1}^2))^{2/3} \tag{3.20}$$

where η is a constant and $0.1 < \eta < 0.5$. η also can be determined as 95% upper tail of exponential distribution with expected value as mean of sample variances in the current frame [89].

7. Obtain all the discontinuity locations τ_i so that

$$\tau_i = n|\hat{\sigma}_n^2 > \lambda \tag{3.21}$$

where $n = 2q+1, 2q+2, ..., N-2q-1$.

8. If the number of discontinuity locations obtained is less than one, it implies that there is no discontinuity in the present frame of the signal. Hence, denoise the samples using KF with gain $k1$. When the denoising of current frame is complete, skip all the steps below.

9. Otherwise find the consecutive discontinuity locations .

10. Denoise the samples from starting location of first discontinuity region until the end location of the last discontinuity region, using KF with gain $k2$. Rest of the samples are considered to be present in non-discontinuity region and denoise them using KF gain $k1$.

11. Iterate all the above procedural steps for each frame.

The discontinuity detection is illustrated in Figure.3.4

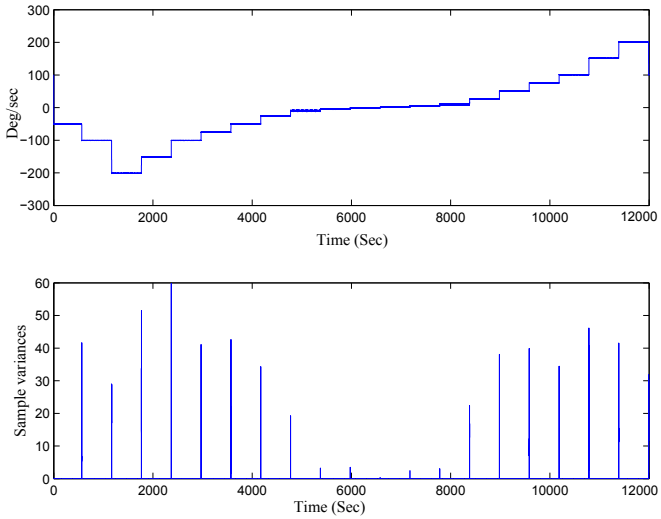


Figure 3.4: Discontinuity detection using AMADMKF algorithm

3.4 Experimental setup

In order to validate the performance of the algorithms, several test signals are acquired from two different closed loop FOG's (one is three axis (x, y, z) FOG and other is single axis FOG) in both static and dynamic conditions. The single axis FOG is shown in Figure.3.5. It uses a 25 volts DC-DC converter as power supply for FOG. Also it sends data through serial data RS422 - RS232 converter to PC

at baud rate of 115200. The data obtained from three different single-axis FOG's are labeled as *set 1*, *set 2*, and *set 3*. These data are collected in static condition for 12.25 hr duration in a rate table with sampling frequency of 400 Hz at room temperature. The dynamic condition test data are collected from three single axis gyros for 1.25 hr duration with different rotation rate/hr ($\pm 2^{\circ} \pm 6^{\circ}$ and $\pm 10^{\circ}$) at room temperature. These are labeled as *set 4*, *set 5*, and *set 6*.
Static data sets (x-axis, y-axis, z-axis) are also collected from the three axis FOG (static condition) for 4.5 hours with a sampling frequency of 200 Hz at room temperature (20°). The dynamic condition test data are collected from the same three axis FOG for 2 hours duration with different rates of rotation ±200 to ±1 Deg/sec at different temperatures i.e., 0°, 20°, 40°and 60°C. The proposed algorithm is applied to all the three axis signals but only *y-axis* results are presented. Similarly, only *x-axis* dynamic FOG signal is presented.

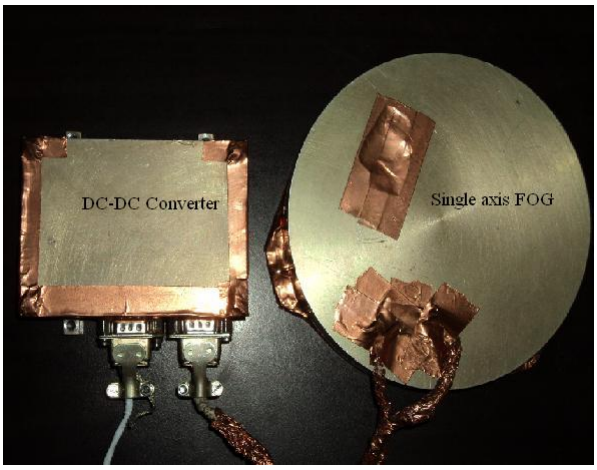


Figure 3.5: Single axis FOG

3.5 Simulation results

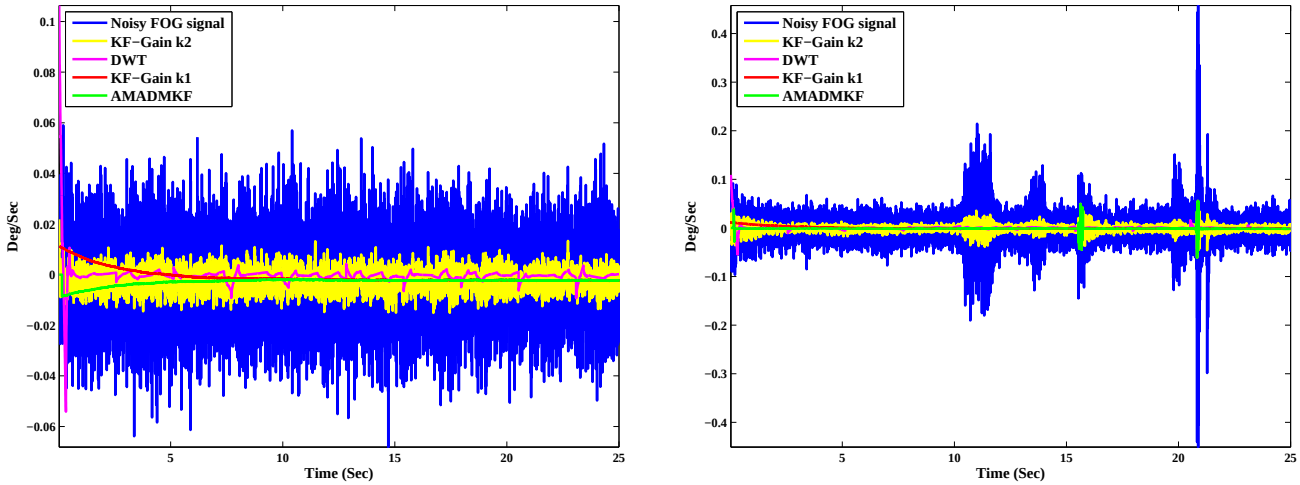
This section presents simulation parameters and results of each of the denoising algorithms that are considered for comparison as discussed in 3.2. The performance of DWT, KF, and AMADMKF algorithms are compared based on the Standard Deviation (SD) and relative Signal to Noise Ratio (SNR) [92].

$$SNR = 10log\left(\frac{\sigma^2_{before}}{\sigma^2_{after}}\right) \tag{3.22}$$

where σ^2_{before} and σ^2_{after} refers to the variance of the signal before and after denoising respectively.

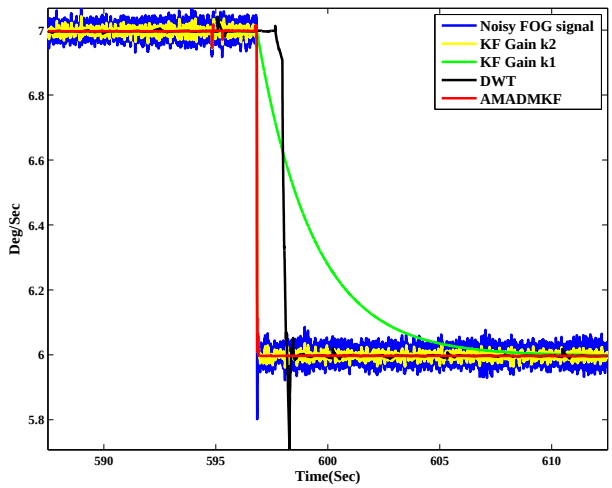
The performance of the denoised algorithm for dynamic signal is expressed in terms of bias drift and response rate [77]. Allan Variance noise analysis is carried out before and after denoising the static signal. Allan variance analysis is commonly used time domain analysis technique to quantify different random errors like (Quantization noise (Q_Z), Angle Random Walk (N), Bias Instability (B), Rate Random walk (K), Drift Rate Ramp (R)) present in the FOG. More details about this can be found in [81].

For DWT algorithm, *db-2* wavelet is chosen as the mother wavelet with the 7-level of decomposition. Different wavelet bases have been tested and it is found that *db-2* is the best suitable one for this application. The threshold method is selected as soft with *Sqt-log mode*. For AMADMKF algorithm, 4096 data samples are considered as one frame, whereas 1024 samples are considered as a frame for DWT algorithm and KF denoised the signal sample by sample. These frames have different lengths, and chosen by conducting many experiments [68, 93]. For KF algorithm the gains are the ratio between Q_k and R_k. The Kalman gain increases as the ratio Q_k/R_k increases. In this work, Kalman gains $k1$ and $k2$ are chosen as 0.2 and 0.001 respectively.

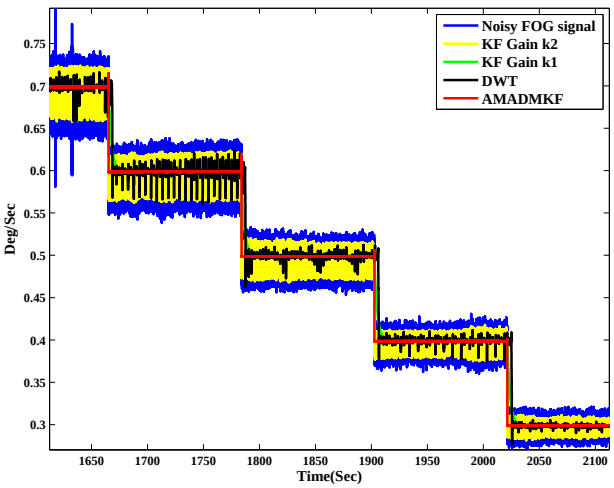


(a) *set 1* data denoised signal (b) *set 3* data denoised signal

Figure 3.6: Comparison of denoised results of *set 1* and *set 3* data under static condition



(a) *set 4* data denoised signal at single transition

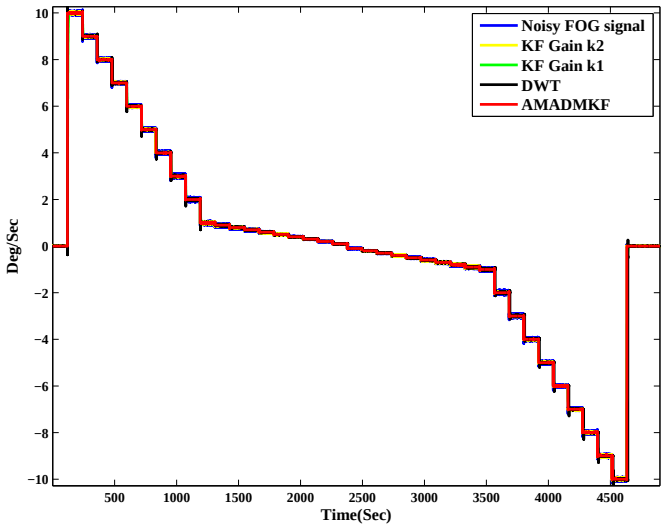


(b) *set 4* data denoised signal at multiple transition

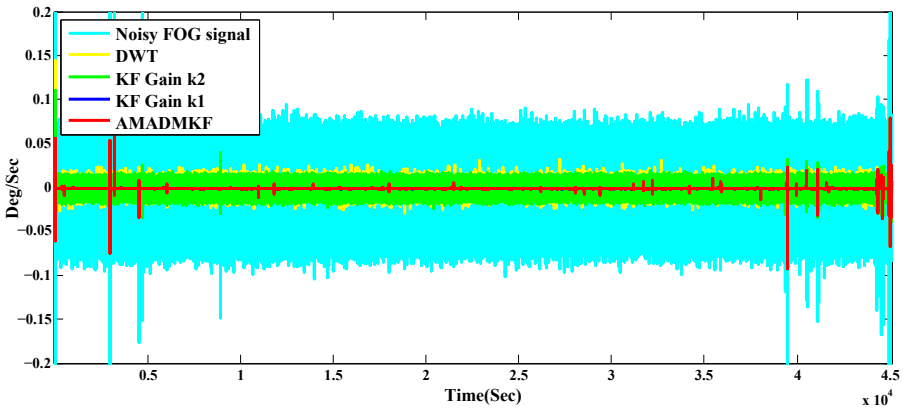
Figure 3.7: Comparison of denoised results of of *set 4* data under dynamic condition

The DWT, KF and AMADMKF algorithms are applied to denoise the data *set 1*, *set 2*, and *set 3* of single-axis FOG. The denoised signal for *set 1* and *set 2* are plotted in Figure.3.6a and Figure.3.6b respectively. The results are plotted for 25 seconds to visualize the denoising effect clearly. However the simulations are carried out for 12.25 hours data. From these figures, it is observed that the DWT algorithm is not able to denoise the signal efficiently, whereas KF and AMADMKF algorithms give competitive performance. This argument is also supported by the improvements observed in the standard deviation and SNR tabulated in Table.3.1 and Table.3.2. Although KF gives quite competitive results with AMADMKF, it fails to denoise the FOG signal under dynamic environment i.e. for data *set 4*, *set 5*, and *set 6*. The denoised dynamic signals of *set 4* are shown in Figure.3.7a and Figure.3.7b respectively. The plotted figures have been taken at different parts of the denoised signals for observing the results more clearly. However, the complete denoised signal of *set 4* and *y-axis* are shown in Figure.3.8a and Figure.3.8b respectively.

The three-axis FOG signals under dynamic conditions are also denoised using the DWT, KF and proposed AMADMKF algorithm. Although the simulations



(a) *set 4* data full denoised signal under dynamic condition

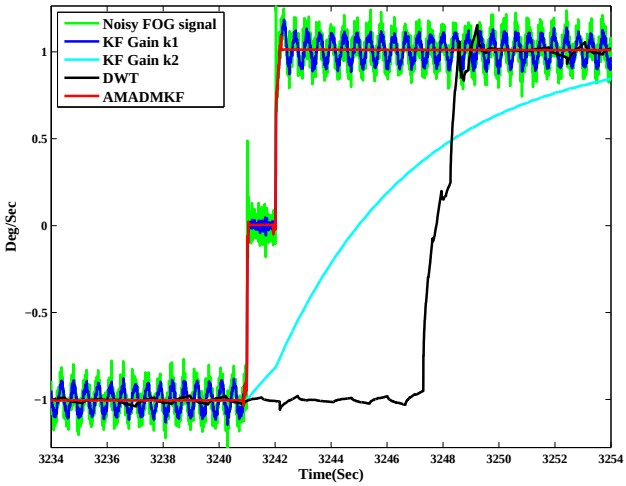


(b) *y-axis* data full denoised signal under static condition

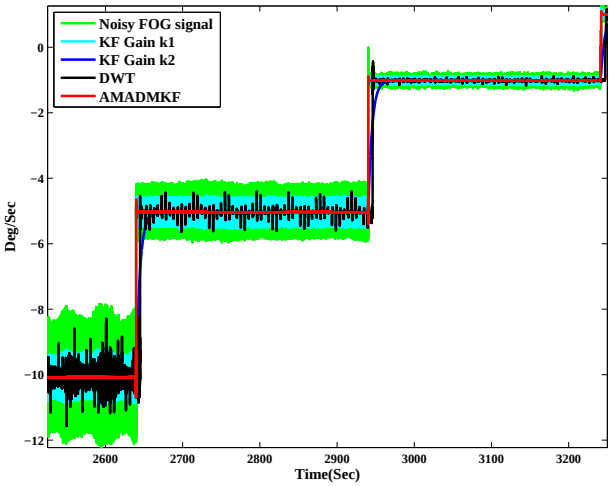
Figure 3.8: Comparison of denoised results of *set 4* and *y-axis* data under dynamic and static conditions

are carried out for complete 1.5hrs data set, for clear observation, 20 seconds of data showing one and multiple transition regions are plotted in Figure.3.9a and Figure.3.9b respectively. In the data set, a transition occurs whenever there is a change in the rotation rate of the object. Figure.3.10a shows the effect of denoising the signal in a static region between two successive transitions whereas and Figure.3.10b shows the complete denoised signal. From the Figures.3.6 to Figure.3.10, the following conclusions have been drawn

1. The DWT algorithm exhibits delay and noisy spikes for initial samples of the signal. This algorithm neither followed the trend of the signal nor denoised

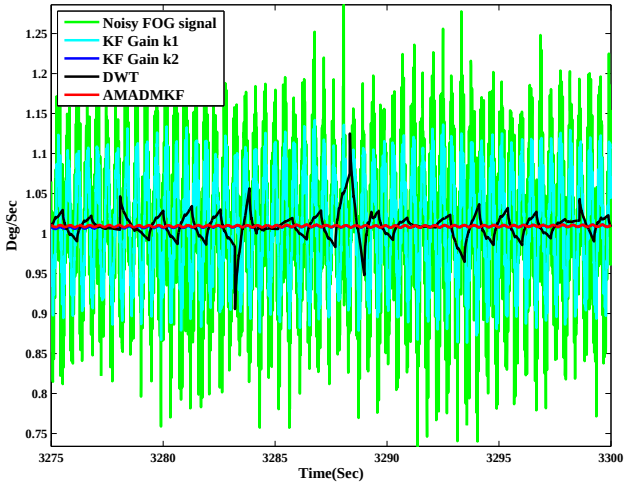


(a) *x-axis* denoised signal at single transition

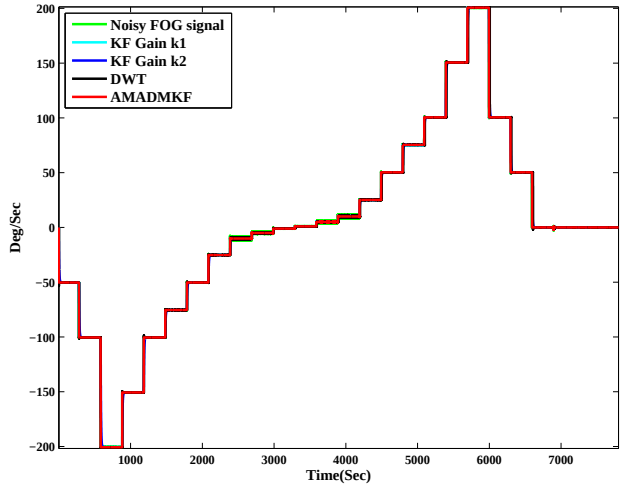


(b) *x-axis* denoised signal at multiple transition

Figure 3.9: Comparison of denoised results of *x-axis* data under dynamic condition



(a) *x-axis* denoised signal under static condition



(b) *x-axis* denoised signal under dynamic condition

Figure 3.10: Comparison of denoised results of *x-axis* data in static and dynamic condition

the signal effectively.

2. The KF-gain $k1$ followed the trend of the signal but could not denoises effectively.

3. The KF-gain $k2$ algorithm denoised the signal effectively in between the

successive transitions and also near the transition regions. However it did not follow the trend of the signal and also it has more settling time.

4. The AMADMKF algorithm denoised the signal in both regions effectively and efficiently. It also followed trend of the signal as shown in in Figure.3.9a and Figure.3.9b.

The above figures reveal that DWT, KF gain $k2$ algorithm is not able to denoise the signal efficiently whereas KF gain $k1$ and AMADMKF algorithms are giving competitive performance in static condition. This argument is also supported by the improvements observed in the bias drift and SNR tabulated in Table.3.1 and Table.3.2. The results demonstrated that AMADMKF algorithm reduces the bias drift of the signal by an order of 100 and improves the SNR approximately by 80 dB.

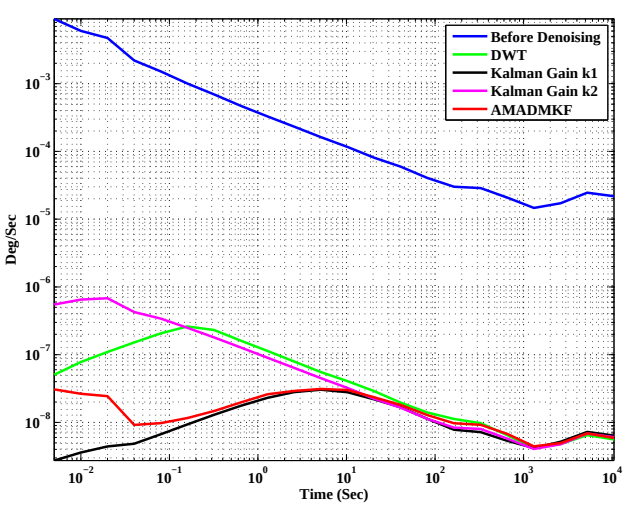
Table 3.1: Comparison of the standard deviation of denoising algorithms

Algorithm	*set 1*(Deg/sec)	*set 2*(Deg/sec)	*set 3*(Deg/sec)
Raw Signal	$161\text{x}10^{-4}$	$184\text{x}10^{-4}$	$205\text{x}10^{-4}$
DWT	$23\text{x}10^{-4}$	$24\text{x}10^{-4}$	$20\text{x}10^{-4}$
KF gain $k1$	$1.8624\text{x}10^{-4}$	$1.8510\text{x}10^{-4}$	$1.9025\text{x}10^{-4}$
KF gain $k2$	$37\text{x}10^{-4}$	$39\text{x}10^{-4}$	$39\text{x}10^{-4}$
AMADMKF	$2.7784\text{x}10^{-4}$	$2.4511\text{x}10^{-4}$	$2.2559\text{x}10^{-4}$

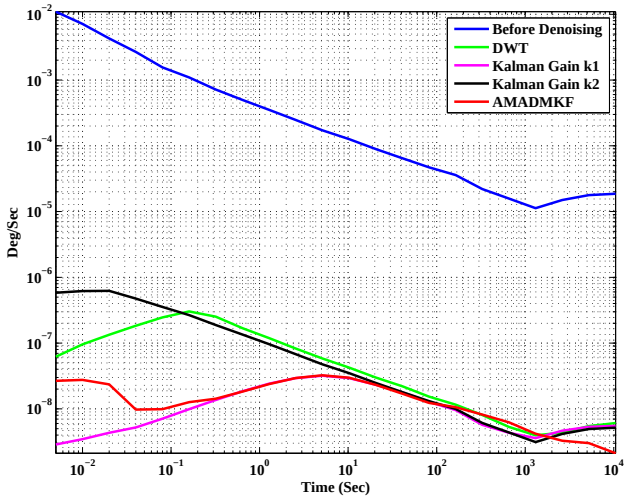
Table 3.2: Comparison of the SNR of denoising algorithms

Algorthim	*set 1*(SNR in dB)	*set 2*(SNR in dB)	*set 3*(SNR in dB)
DWT	39.0252	40.9616	46.4925
KF gain $k1$	89.2074	91.9847	93.3793
KF gain $k2$	29.5478	30.9326	32.9927
AMADMKF	81.0272	86.3687	90.1595

We have also analyzed the performance of denoising algorithms using Allan Variance analysis. The Allan variance is applied to long hour data sets only, whereas dynamic environment data changes its rotation more often, so we cannot apply this analysis to dynamic set data. We carried out Allan Variance analysis on each set of static signals before and after denoising. The Allan Variance plots of *set 1* and *set 3* are plotted in Figure.3.11. Figure.3.11a and Figure.3.11b shows that the AMADMKF and KF gain $k1$ has lower slopes as compared to DWT, KF



(a) Allan Variance analysis of *set 1* data



(b) Allan Variance analysis of *set 3* data

Figure 3.11: Comparison of Allan Variance analysis before and after denoising of single axis static data

Table 3.3: Allan Variance analysis of *set 1* data

Factor	Raw Data	DWT	KF gain $k1$	KF gain $k2$	AMADMKF
Q	1044x10^{-4}	51x10^{-4}	16x10^{-4}	16x10^{-4}	5.7256x10^{-4}
N	20x10^{-4}	0.93556x10^{-4}	0.295x10^{-4}	0.47486x10^{-4}	0.10763x10^{-4}
Bs	861x10^{-4}	83x10^{-4}	27x10^{-4}	38x10^{-4}	9.6754X10^{-4}
K	14.0307	0.5648	0.1839	0.2559	0.0672
R	645.437	27.1835	9.0103	12.2006	3.2853

Table 3.4: Allan Variance analysis of *set 3* data

Factor	Raw Data	DWT	KF gain $k1$	KF gain $k2$	AMADMKF
Q	1633x10^{-4}	51x10^{-4}	61x10^{-4}	32x10^{-4}	5.3396x10^{-4}
N	26x10^{-4}	0.94815x10^{-4}	0.29500x10^{-4}	0.47486x10^{-4}	0.10763x10^{-4}
Bs	861x10^{-4}	83x10^{-4}	27x10^{-4}	38x10^{-4}	9.6754X10^{-4}
K	14.0307	0.5648	0.1839	0.2559	0.0672
R	645.437	27.1835	9.0103	12.2006	3.2853

gain $k2$. The random noise coefficients are quantified from the slopes of the Allan Variance plot and the values are tabulated in Table.3.3 and Table.3.4. From these tables it is evident that there is a significant reduction in the noise coefficients in case of AMADMKF algorithm as compared to DWT and KF gain $k2$, whereas it gives nominal improvement as compared to KF gain $k1$. The noise coefficients like rate random walk, angle random walk etc., are reduced after denoising by

Table 3.5: Bias drift for dynamic condition of *x-axis* data at (20°)

Rotation(Deg/(sec))	Input	DWT	KF gain $k1$	KF gain $k2$	AMADMKF
-200	0.07903444	0.03503364	0.03741309	0.02874328	0.00190266
-175	0.06971800	0.01913449	0.00444798	0.02813906	0.00444798
-150	0.08232674	0.01402021	0.02050005	0.03304942	0.00135209
-125	0.08772601	0.03435027	0.00120231	0.03732033	0.00120231
-100	0.08254453	0.02181158	0.03925280	0.03117872	0.00114857
-75	0.07587522	0.04893878	0.02945189	0.02838094	0.00142887
-50	0.07272912	0.04633337	0.05730649	0.02652248	0.00217494
-25	0.16273886	0.67616173	0.22591008	0.03387397	0.00724240
-10	1.18274250	0.16234799	1.35868718	0.36400719	0.02185515
-5	0.52355471	0.06665217	0.41727810	0.27739167	0.00510942
0	0.09825683	0.01846334	0.34440835	0.06603170	0.00277341
5	0.36446447	0.57405045	0.74097886	0.35384390	0.36799868
10	0.10345610	0.04078017	0.14994349	0.06719986	0.00389824
25	0.52383367	0.26465033	0.36765599	0.27679807	0.01093359
50	1.14341674	0.41860928	0.45381398	0.35124023	0.00697088
75	0.16016677	0.03182775	0.03407524	0.03155125	0.00485489
100	0.07137773	0.05441877	0.02106103	0.02473529	0.00448087
125	0.07409974	0.05401282	0.02335348	0.02792786	0.00262171
150	0.07912759	0.06101953	0.04328984	0.03437041	0.00896167
200	0.08566183	0.05777715	0.03322102	0.03714617	0.00211409

Table 3.6: Comparison of SNR for denoising algorithms for *x-axis* data

Algorithm	-200°	-150°	-100°	-50°	-10°	0°	10°	50°	100°	150°	200°
DWT	16.2820	35.4263	26.6178	9.0240	39.7222	33.4051	18.6178	20.0969	5.4387	5.1968	7.8773
KF gain $k1$	14.9555	27.7989	14.8315	4.7608	-2.7746	-25.0756	-7.4079	18.4821	24.3805	12.0512	18.9661
KF gain $k2$	20.2510	18.2773	19.4476	20.1839	23.5680	7.9674	8.6380	23.6081	21.2299	16.6531	16.7447
AMADMKF	74.5519	81.4780	86.3498	69.9577	79.7814	71.1681	65.5719	101.9170	55.2844	43.4698	74.1783

AMADMKF technique as evident from Table.3.3 and Table.3.4. So Allan Variance analysis concludes that proposed AMADMKF algorithm is superior as compared to other considered algorithms [49, 94]. For dynamic case, the three-axis FOG is subjected to different rotation rates from -200° to 200° with the step of 25° per second. Bias drift is calculated using all the three algorithms in each rotation rate and the values are tabulated in Table.3.5. From this table it is observed that the proposed AMADMKF algorithm reduces the bias drift significantly at most of the rotation steps. Similarly, SNR is evaluated for ten different step rotation of dynamic signal. The results are tabulated in Table.3.6, it is observed that for dynamic signal the AMADMKF algorithm increases SNR of denoised signal sub-

stantially as compared to other algorithms. In case of static signal the AMADMKF algorithm detected some noise as discontinuity, this causes degradation of SNR. However the difference between SNR of KF gain $k1$ and AMADMKF is marginal. Hence, it can be concluded that the AMADMKF algorithm is more suitable for denoising both the static and dynamic conditions of FOG signal.

3.6 FPGA implementation of denoised algorithms

This section presents the architecture (in terms of sysgen modules) of DWT and KF algorithms for denoising the FOG signal.

3.6.1 Hardware architecture of DWT

The DWT hardware is developed using Xilinx system generator for DSP. To optimize time and resources several architectures of the DWT algorithm such as convolution and lifting schemes have been developed for FPGA implementation [95]. These architecture focus on minimizing the latency of the design. The efficiency of an architecture can also be enhanced by minimizing the arithmetic blocks. Distributed arithmetic (DA) is one such popular architecture being used in the design. It is also referred as multiplier free design [96]. In the present work, DA technique is used for computing the convolution of DWT blocks. Although it is concluded (from the previous Section 3.5) that DWT fails to denoise the FOG signal in dynamic environment, we have implemented this in Xilinx sysgen environment to compare the resource utilization with the architecture of other algorithms. Figure.3.12 shows the procedure for algorithm to Intellectual Property (IP) development phase of the DWT algorithm. When the DWT algorithm result matches with the sysgen design result, an IP of the DWT algorithm can be tested and integrated in SoC. In this work we have not built the IP of DWT algorithm, because the algorithmic performance of DWT is not satisfactory.

The efficiency of the architecture plays an important role during the performance evaluation. In DWT architecture group delay effect the synchronization between the wavelet coefficients in various decomposition levels , this delay increases as the level of decomposition increases. To reduce the group delay, convolution architecture based on DA FIR filter is developed. This DWT architecture processes serial samples, frame by frame for denoising the FOG signal. For im-

plementation of DWT in FPGA several half band high pass and low pass filters are realized using DA FIR module [97]. Apart form this, the architecture has up/down-converters, interpolators, summers and reinterpret blocks in decomposition and reconstruction modules. A Black-box module is used for thresholding the wavelet coefficients.

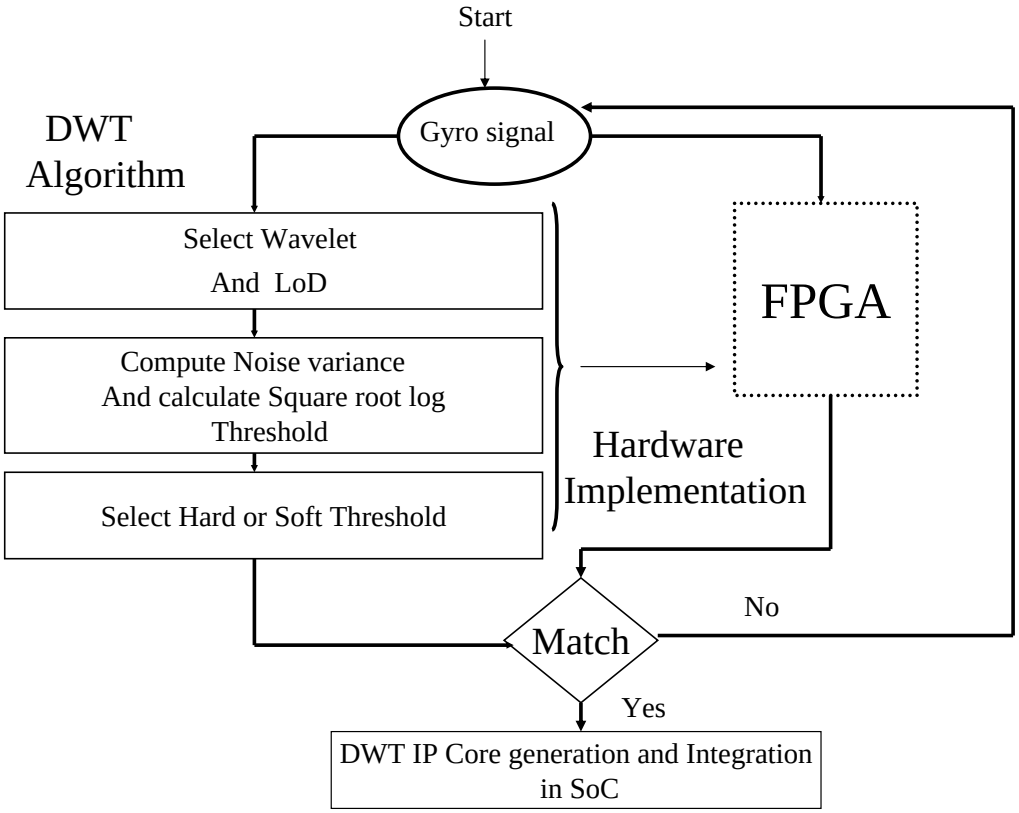


Figure 3.12: DWT implementation procedure

We have selected *db-2* as mother wavelet, Level of Decomposition (LOD) as 7 and the processing of FOG data is carried out frame wise with each frame consisting of 1024 samples. Each decomposition stage has two FIR filters (low pass, high pass) as shown in Figure.3.13. The low pass filter coefficients are further decomposed into second level detail and approximation coefficients. This process is repeated till the last decomposition level is reached. In each level of decomposition, threshold is applied on the high pass filter coefficients as shown in Figure.3.14. Hard threshold with Sqtwolog is selected as threshold selection rule. During reconstruction, Inverse Discrete Wavelet Transform (IDWT) technique is used as shown in Figure.3.15. It uses two filters (lowpass, highpass), the low pass coefficients of seventh level is up-sampled by a factor of 2 and added with last stage

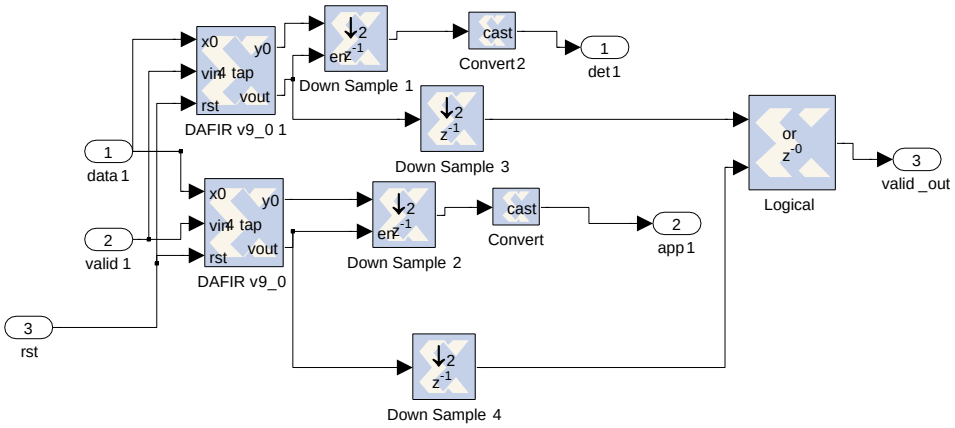


Figure 3.13: Single level decomposition structure

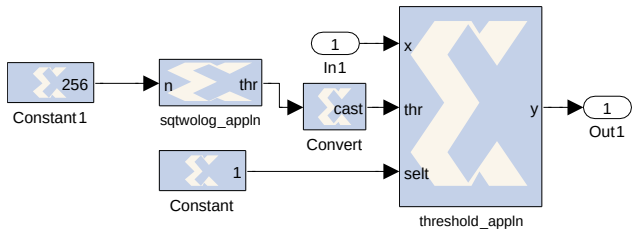


Figure 3.14: Threshold selection

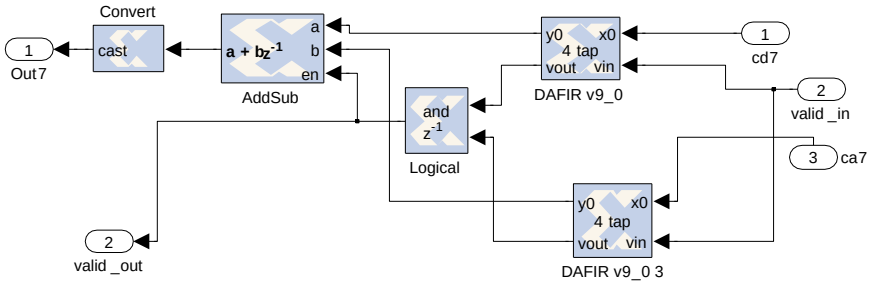


Figure 3.15: Single level reconstruction structure

thresholded high pass filter coefficients in order to first first stage reconstruction.

3.6.2 Hardware architecture of KF

The proposed KF hardware is developed using Xilinx system generator for DSP. The KF hardware is divided into predict and update stages as shown in Figure.3.16. The complete hardware consists of Multiplexers, Divider Generator, Registers, Accumulators, and Assert blocks. In the prediction state, Mux select

line is permanently made logic high, so as to always select previous posteriori error covariance. The value of Q is input through Constant 1 block. The result of Add-Sub block gives the predicted (priori) error covariance. The predicted state estimate is obtained through a feedback from the previous updated (posteriori) state estimate using Register 2 and Mux 1. The output of Add-Sub 5 block gives the KF gain. The innovation is calculated using the predicted state estimate (output of Mux 1) and present measurement z_k (Gateway out) and output of Add-Sub 2 block. Posteriori error covariance is obtained at the output of Multiplier 1. The updated state estimate is obtained at the output of Add-Sub 3 block.

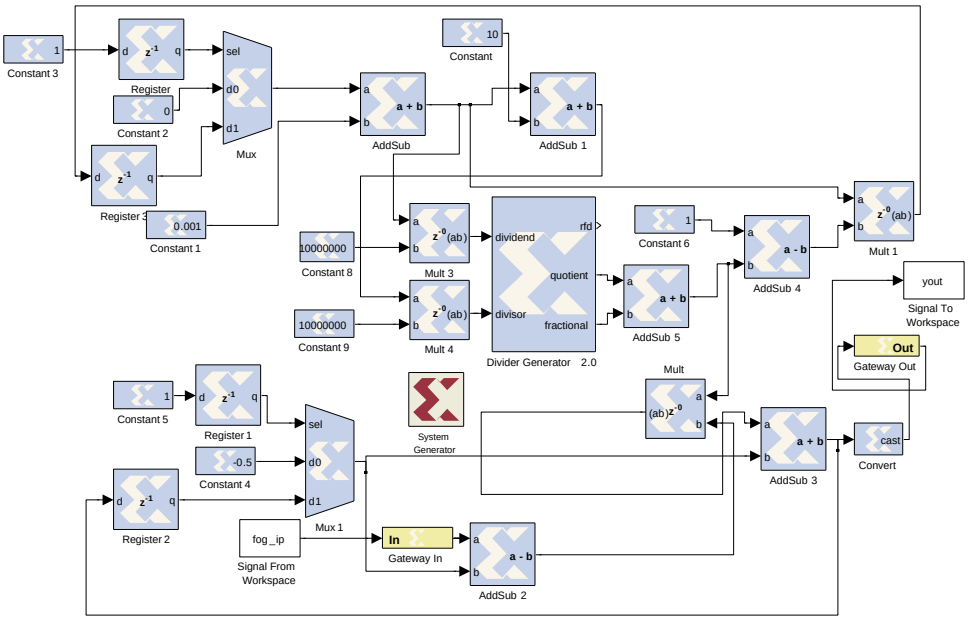


Figure 3.16: Sysgen KF architecture

3.7 Hardware architecture of the proposed algorithm

The proposed AMADMKF algorithm uses four sub-modules, namely Moving Average & Memory module, Difference module (for computing the differences of samples), Variance module, and Threshold module along with the KF module. The flowchart of this algorithm is presented in Figure.3.17. The complete architecture of the system is shown in Figure.3.18. It consists of two asynchronous FIFOs

(depth of 8192 and width of 32 bits), one each at the input and output of the AMADMKF module. The input signal is processed as a stream and each stream has 4096 samples. When the input FIFO (FIFO1) and output FIFO (FIFO2) receive these 4096 samples, then it asserts FIFO1 and FIFO2 as 50% full. This architecture has *IP_Data* and *OP_Data* (32 bit width) data buses for data input and output to and from the IP core respectively. The working principle of the AMADMKF IP core is described below:

1. The AMADMKF IP core receives data from the PowerPC(PPC440) when *IP_Data_En* is logic high, and it receives the data till the FIFO1 is 50% is full. This is ensured by the control signal *IP_Data_End* as logic high.

2. When the AMADMKF IP core is ready for processing, it will give a handshaking signal *Input_Data_Rdy* as logic high. The FIFO1 sends the *Input_Data* to the AMADMKF IP core when *Core_input_En* is logic high, and the data transfer continues till *Core_Input_End* is logic high.

3. When the AMADMKF IP core processes inputs sequentially on the stream, it gives *Core_Input_En* as logic high and this is acknowledged by the FIFO1 with handshaking signal *Core_Input_Rdy*. When this signal is logic high, then the AMADMKF IP core is ready to send the processed samples to the output to FIFO2.

4. AMADMKF IP core communicates with FIFO2 in the same way as FIFO1 communicates with AMADMKF IP. When output data is available at FIFO2, *OP_Data_En* signal becomes logic high and the FCM sends an acknowledge signal *OP_Data_Rdy* as high. Then the *Output_Data* is transferred to FCM till *OP_Data_End* becomes logic high.

3.7.1 Moving average & Memory module (MA & Memory)

This is the first stage of the AMADMKF core and the architecture of this module is shown in the Figure.3.19. The default parameter values are $N = 4096$, $q = 20$, and $z = 3$. These are obtained from the control logic and the other parameters i.e. qt and qh are set to zero initially. Based on the value of z and $selectAMA$,

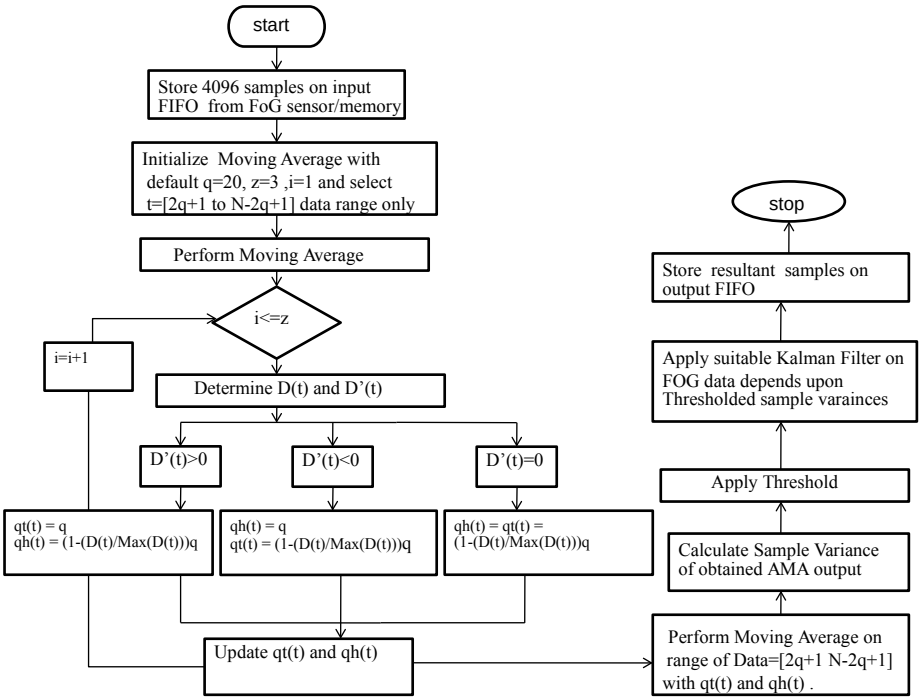


Figure 3.17: Flow chart of the proposed algorithm

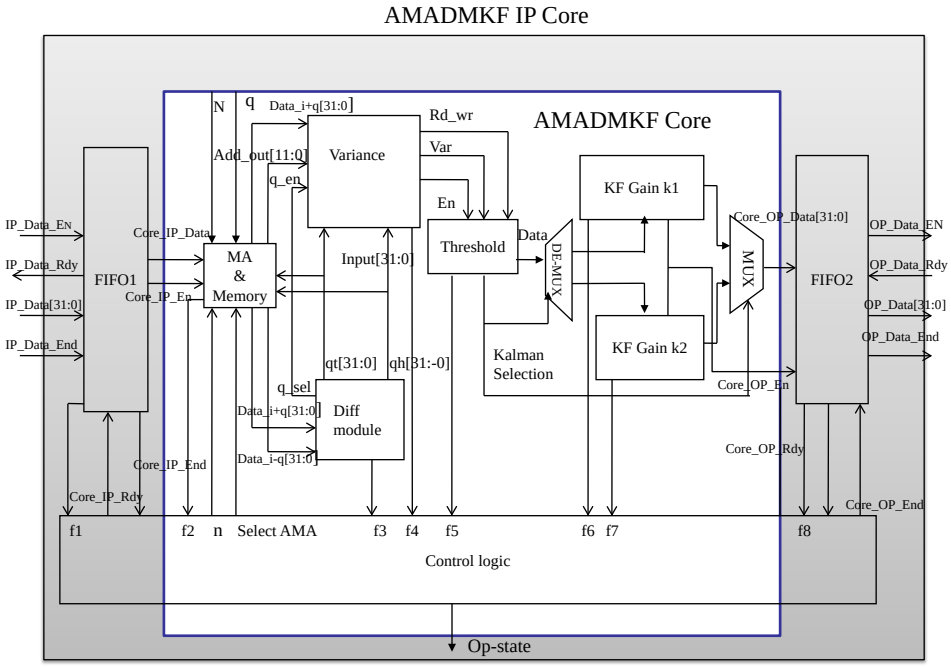


Figure 3.18: Top level architecture of AMADMKF core

the control logic redirects output to Variance or Difference module. From Figure.3.19, MA & Memory module performs q-point MA operation on IP_Data

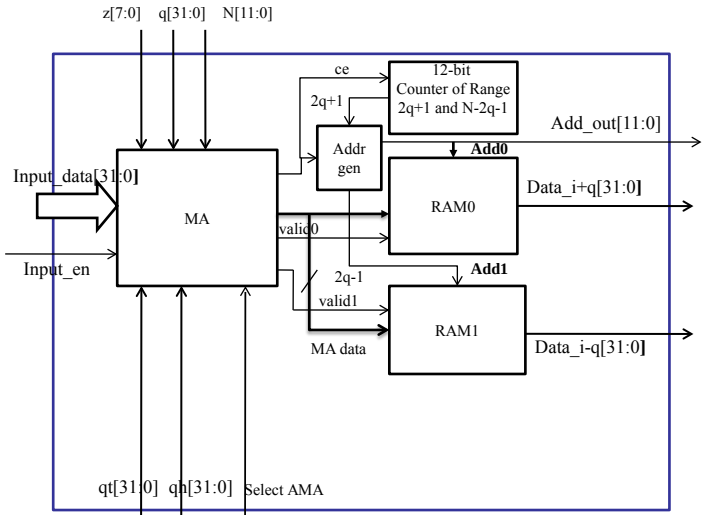


Figure 3.19: Architecture of Moving average and Memory module in AMADMKF core

when the control logic asserts $z = 0$ and $selectAMA = 0$. A programmable 12-bit counter and address generation module ($Addr\_gen$) are used for addressing the RAMs. When $valid0$ and $valid1$ are at logic high, moving average data is stored in RAM0 and RAM1. The data from RAM to output and MA to RAM is transfered simultaneously resulting $Data\_(i+q)$ and $Data\_(i-q)$ for processing in the Difference module. The output data of MA uses only RAM0 Memory module when $selectAMA$=1,Then the output data is transferred to variance module for further processing.

3.7.2 Difference module (Diff)

The output data from MA & Memory module ($Data\_(i+q)$, $Data\_(i-q)$) are input to the Difference module. The architecture of this module is shown in the Figure.3.20. The counter with address generation output is connected to the $Addr\_in$ to address the RAMs, and q_sel is the write enable control signal of RAM. This module performs difference operation $D(t)$ between $x(t+q)$ and $x(t-q))$ samples, and also performs difference $D'(t)$ between consecutive differences $D(t)$. qt and qh values are obtained depending on the different conditions/signs of $D'(t)$. These values are updated using equations (3.14) and (3.15). The Difference module and MA & Memory module is updated z times to update the ping-pong memory buffers with updated qt and qh values for the entire frame as in Figure.3.18

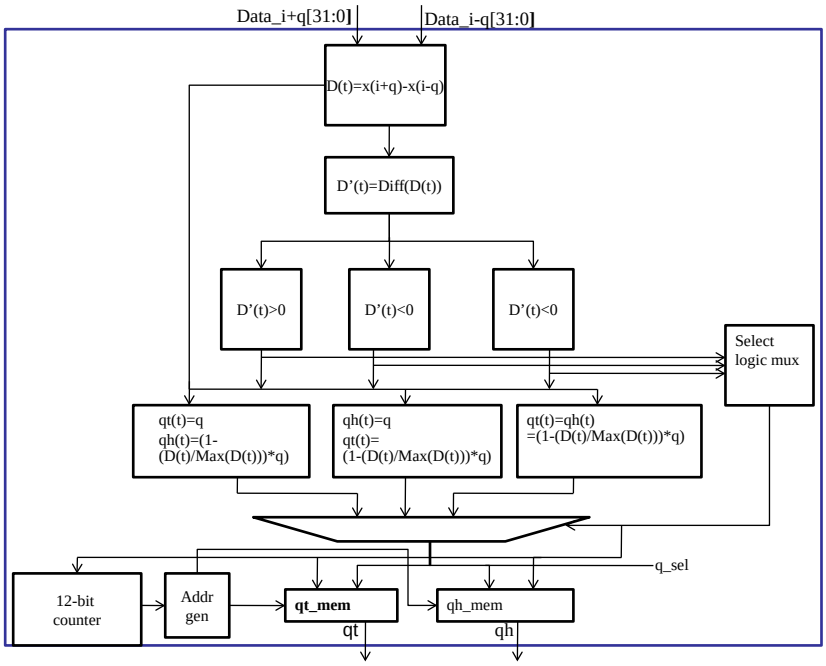


Figure 3.20: Architecture of Difference module in AMADMKF core

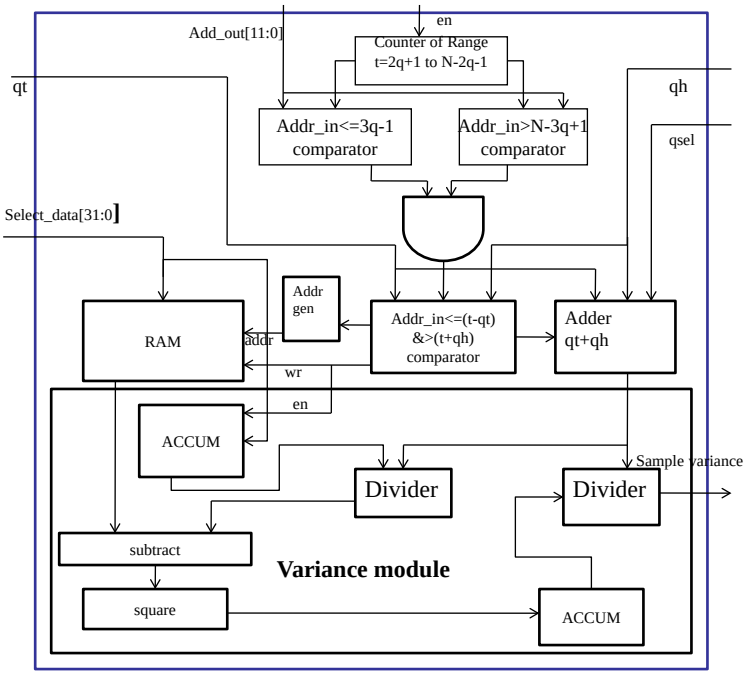


Figure 3.21: Architecture of Variance module in AMADMKF core

3.7.3 Variance module

The detailed architecture of this module is shown in the Figure.3.21. The control logic activates the Variance module when $z > 3$. The qt and qh values (from the Difference module) are used to calculate the MA output in the MA & Memory module. The output of the MA & Memory module ($selectAMA$) is used to calculate the variance. This module has a comparator, which detects the address between the memory locations $(3q + 1, N - 3q - 1)$. This comparator output enables the address generation unit and RAM. The output of AND logic enables the accumulator to perform the average operation for calculating the mean and sample variance.

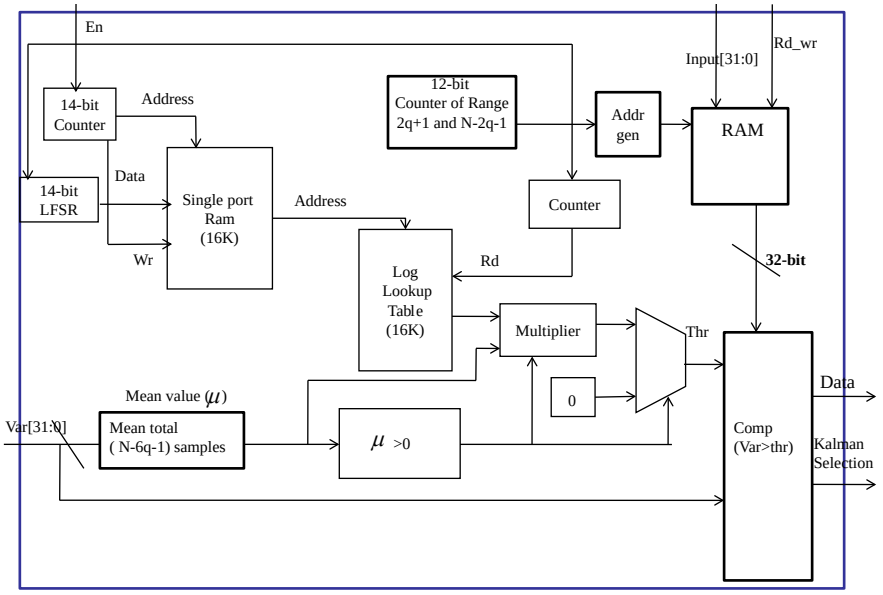


Figure 3.22: Architecture of Threshold module in AMADMKF core

3.7.4 Threshold module

The detailed architecture of threshold module is shown in the Figure.3.22. This module computes mean (μ) of the sample variances obtained from the variance module. The comparator checks the condition $\mu > 0$ and enables it to calculate the threshold by multiplying μ with the distribution values stored in a log look up table. If $\mu < 0$, it makes $thr = 0$. The *Enable* signal starts the counter and 14-bit linear feedback shift register (LFSR) to store the random numbers in 16Kbyte RAM. The predefined 14-bit wide logarithmic random values are read

from the single port ROM and multiplied by μ. Once the threshold is computed, the comparator checks the variance with threshold and proper KF is selected. Then the KF performs the denoising operation according to the gain parameters and resulting denoised samples are transferred to FIFO2 as shown in Figure.3.18.

3.7.5 Control logic

The control logic synchronizes rest of the submodules and generates the necessary inputs to them. It collects all the flags f1,f2,..f7 that are associated with each individual module. These flags are useful to debug various stages of the architecture.

3.8 Programmable System on Chip (PSoC) platform for AMADMKF coprocessor

In this work, a PSoC prototype for denoising FOG signal is developed [89]. It is an integrated system that integrates processors, peripherals, memories and coprocessor on a single chip. The FPGA SoC mainly offers hardware re-use and easier programmability to an user for developing complex systems. The proposed SoC system using Virtex-5 FPGA consists of PPC440 processor and peripherals such as UART (RS-232), DDR2-SDRAM, BRAM, Timer, Interrupt modules, custom hardware accelerators and APU based coprocessors. Processor Local Bus (PLB) is intended for communication between processor and peripherals in FPGA-based SoC. The Fabric Co-Processor Bus (FCB) is a dedicated high-speed, low latency bus between the processor and APU for faster transferring from APU to processor and vice versa as shown in Figure.3.23.

A Fabric Coprocessor Module (FCM) of AMADMKF algorithm is developed and interfaced with embedded PPC440 processor through APU controller to accelerate computation of the algorithm. Using extended instructions of *load and store* for transferring the data between Processor and APU are used [4]. The processor (PPC440) is a preferred choice over other soft core processors like Microblaze in FPGA, due to its high speed of operation and efficient resource utilization [16]. The proposed SoC platform is shown in Figure.3.23. It has three main blocks namely, processor (PPC440) with APU interface, FCM and external peripherals such as memory controller and two UART (RS-232) based communication serial

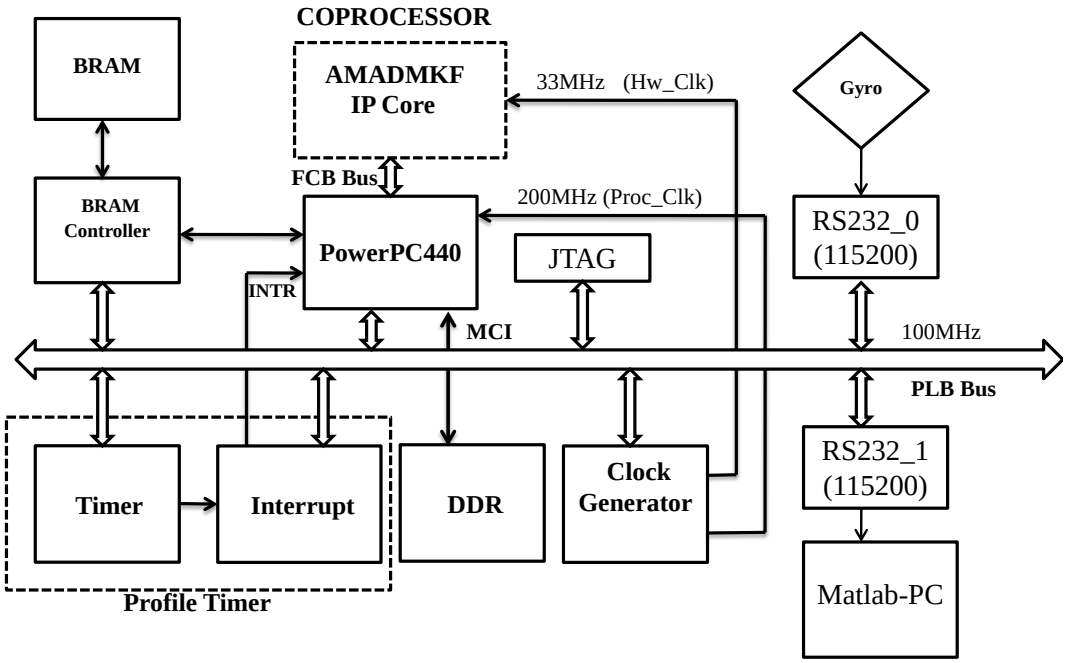


Figure 3.23: FPGA based SoC system

links. The PPC440 is connected to the Processor Local Bus (PLB) via Master port (MPLB) and DDR2-SDRAM is connected through the Memory Controller Interface (MCI). The other slave peripherals like serial communication links UART_0, UART_1 and Flash memory are connected to the common processor local bus (PLB) through Slave PLB port. The FCM for AMADMKF core is interfaced to the PPC440 through the Fabric Coprocessor Bus (FCB). The AMADMKF coprocessor frequency is adjusted by system clock generator to 33 MHz. This clock generator which is not shown in Figure.3.23 not only generates the clock for coprocessor but also it generates clock for the entire SoC system i.e. PPC440 clock, DDR clock, Bus Clock etc.

The UART_0 RX link receives the input from FOG and transmits the data to FPGA for denoising. The hardware module (AMADMKF) denoises the noisy data and transfers denoised data to the PPC440. UART_1 TX transmits the denoised data from the FPGA to Matlab-PC for analyzing and plotting the denoised result. The FOG board has a 25 volts DC-DC converter for power supply and RS422 - RS232 converter for receiving the input from FOG at baud rate of 115200. Noisy data from FOG contains temperature, rotation rate and Cyclic Redundancy Check (CRC) informations. A software code is written on PPC440 for acquiring the gyro data for CRC check and header detection. Then the data samples are input to

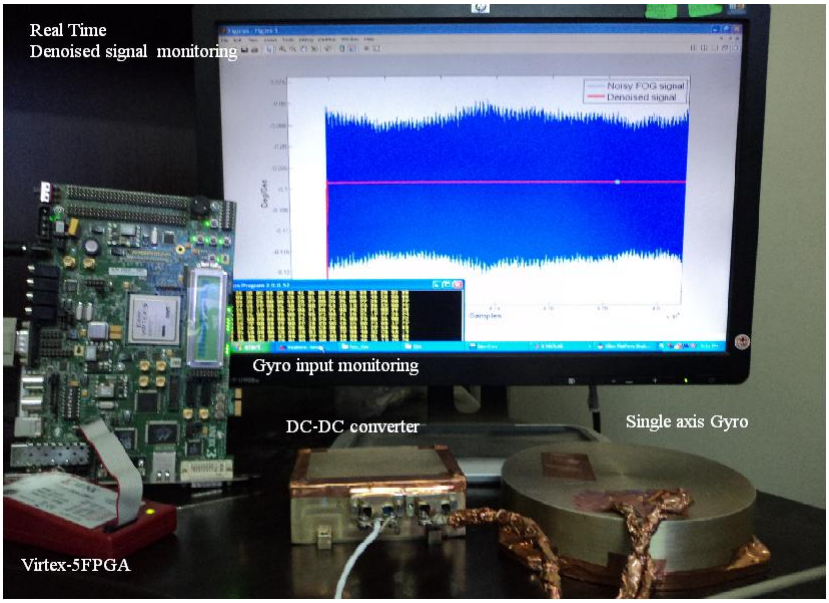


Figure 3.24: FPGA based SoC testbed setup for AMADMKF in real time

the AMADMKF FCM for denoising. Finally the denoised output data are sent to Personal computer (PC) through RS232_1 for further analysis. DDR2-SDRAM is used to store the heap and stack of the software program. Compact flash peripheral is used to test the off-line test of gyro data and also to store the denoised data. In this work, we process only one component of Inertial Measurement Unit (IMU), there are several other modules like processing of other 2 gyroscope signals, 3 accelerometer signals, estimating the navigation information, interfacing and processing with other sensors. Moreover the baud rate has to be increased to 921.6 Kbps. All these combined processing requires higher end FPGA. For these reasons, in the first stage of prototype development Virtex-5 FPGA is selected. Although it is cost effective, but it has enough resources to accommodate the complete process requirements.

The SoC system with APU co-processor using PPC440 processor for denoising the FOG signal is similar to Figure.2.4 of Chapter 1. It has mainly two modules, i) APU wrapper and (ii) AMADMKF IP core. The objective of APU wrapper is to interface FCM (AMADMKF IP core) with processor, whereas AMADMKF IP core is used to denoise the samples. APU wrapper contains two different modules namely *IP_APU* and *APU_IP* interfaced with the IP core similar to Figure.2.6. For load data (*APU_IP*), APU Controller sends entire 128-bit bus along with starting byte address to the FCM and it checks the starting byte address to identify the

valid data. During the store instruction, (*IP_APU*) transfers the denoised samples from FCM to processor (PPC440). A snapshot of the real time test bed is shown in Figure.3.24

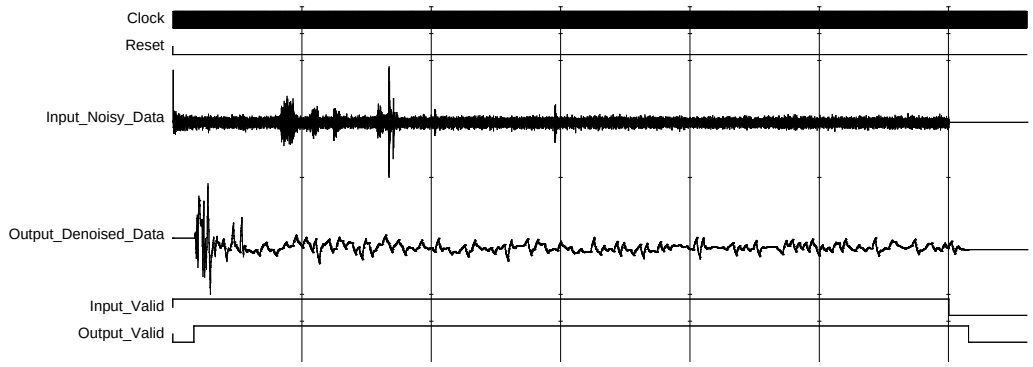
3.9 Implementation results

3.9.1 FPGA implementation of AMADMKF IP core results

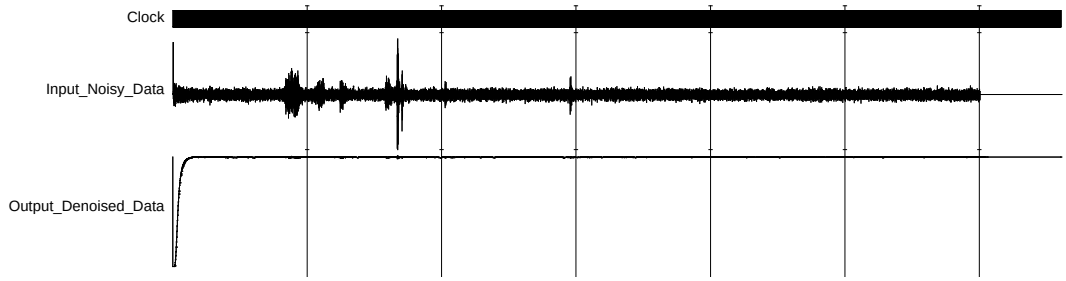
After successful algorithmic simulation, AMADMKF algorithm is coded using HDL language and functional verification is performed. The interface wrapper logic and the AMADMKF core are simulated using a test bench and the results are compared with the Matlab/C software output. The simulation results of DWT, KF, AMADMKF are shown in Figure.3.25. When the *Output_Data_Rdy* signal is logic high, the denoised output data is available at *Output_Data* port which is in fixed point format and are scaled and converted into single precision floating point format to compare with the software MATLAB/C results.

The resource utilization of individual algorithms and total SoC system with AMADMKF IP are tabulated in Table.3.7. This table shows percentage of resources utilized among the individual available resources in the FPGA. The KF algorithm involves arithmetic operators like Dividers, multipliers, adders, and due to this it uses more DSP48E slices, whereas AMADMKF algorithm uses simplified expressions which reduces the computations. Although it has less computation, but since it processes 4096 samples as one frame and denoises the signal frame by frame in contrast to KF algorithm which processes the signal sample by sample, AMADMKF algorithm consumes more resources than KF algorithm. It is observed that the latency of KF is less due to the sequential processing of samples in contrast to frame processing of DWT and AMADMKF algorithm. The execution time of these algorithms are evaluated and used as a performance indicator for comparing different algorithms.

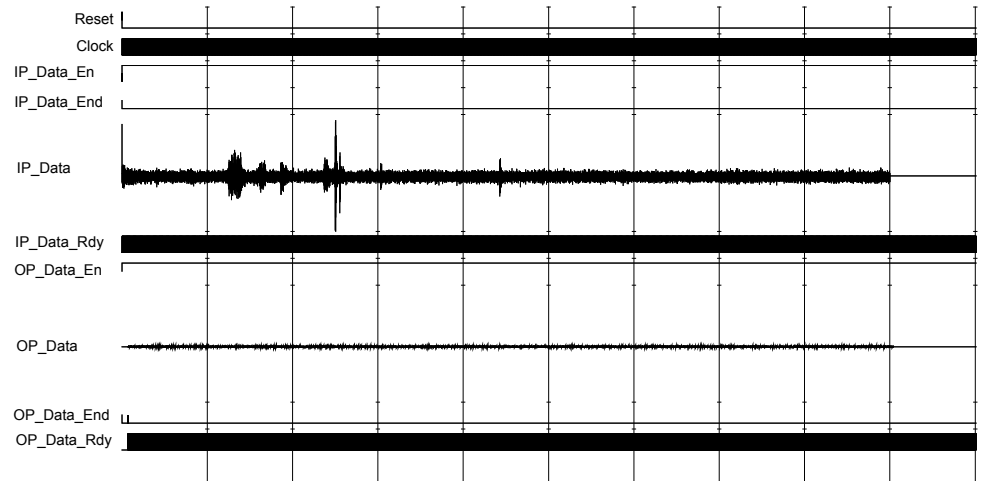
FPGA has a mixture of heterogeneous device primitives (Slice Registers, Occupied Slices, Slice LUTs, BRAMs, and DSP48s). For area comparison, all the device primitives are converted to logic cells [98]. The resource utilization of different algorithms are tabulated in Table.3.7. From this Table, it is observed that DWT algorithm consumed more number of slices and slice registers, KF consumed more



(a) Hardware core simulation of DWT algorithm for *y-axis* static data



(b) Hardware core simulation of KF algorithm for *y-axis* static data



(c) Hardware core simulation of AMADMKF algorithm for *y-axis* static data

Figure 3.25: Comparison of denoised algorithm versus hardware simulation

BRAM and DSP48E slices and AMADMKF algorithm consumed more BRAM and slices. It is also observed from Table.3.7 that the proposed algorithm has a trade-off between execution time, area and SNR. AMADMKF algorithm is superior to both KF and DWT algorithms in terms of SNR and area. In contrast this

algorithm takes more execution time compared to both KF and DWT algorithms

Table 3.7: Comparison of resource utilization for denoised IP cores

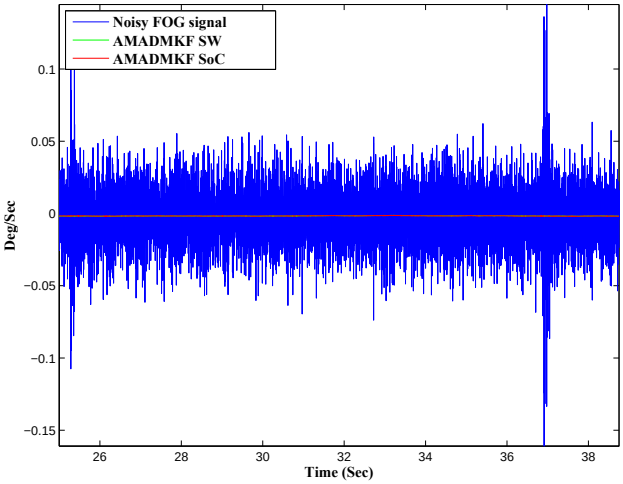
Resources(available)	DWT	KF	AMADMKF	SoC system
Slice Registers(42000)	24177(53%)	4980(11%)	4598(10%)	7469(16%)
Slice LUTs(42000)	18887(42%)	2213(4%)	9542(21%)	12184(27%)
Occupied Slices(11200)	6668(59%)	479(4%)	3407(30%)	8902(79%)
DSP48Es(128)	9(7%)	44(34%)	9(7%)	9(7%)
Block-RAMs(148)	74(50%)	79(53%)	39(36%)	85(57%)
Maximum-frequency(MHz)	236	54	67	200
Latency in cycles	82	4	54	-
Logic cells(48256)	10912	16690	6761	12821
Execution Time(μs)	0.34	0.07	0.8	1.63

It is concluded that from Table.3.7, although AMADMKF algorithm is not optimum with respect to execution time but it is a suitable algorithm for denoising FOG signal in terms of resource utilization and SNR. So SoC system is built using AMADMKF algorithm and the comparative results are plotted in Figure.3.26 and Figure.3.27. The bias drift of IP is calculated and compared with the software result. The results are tabulated in Table.3.8.

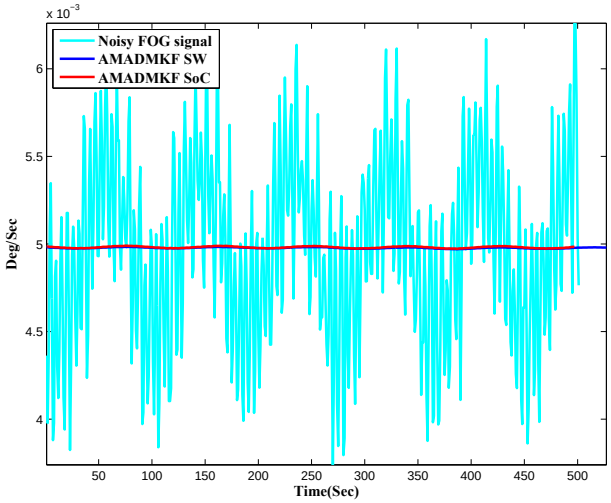
3.9.2 PSoC implementation results

Table.3.7 shows that SoC implementation of AMADMKF core has increase in logic cells compared to AMADMKF core. This is due to the additional resources like processor PPC440, DDR2-SDRAM, internal memory controllers and UARTs in the system. The acceleration factor of the proposed hardware IP with respect to its equivalent software implementation on the embedded processor (PPC440) is tabulated in Table.3.9 which shows that an acceleration of 65x is achieved by the developed hardware IP. This is a significant acceleration. The embedded platform is validated by denoising the collected FOG data sets. The SoC results are compared with the MATLAB simulated results of AMADMKF algorithm and are plotted in Figure.3.27. For clear observation, Figure.3.26a and Figure.3.26b shows a stationary portion of the denoised signal using the MATLAB simulation (AMADMKF) and SoC implementation whereas Figure.3.27a and Figure.3.27b shows the portion of denoised signal in between the transitions of a dynamic signal. This figure confirms that the developed embedded platform denoises the

signal effectively.

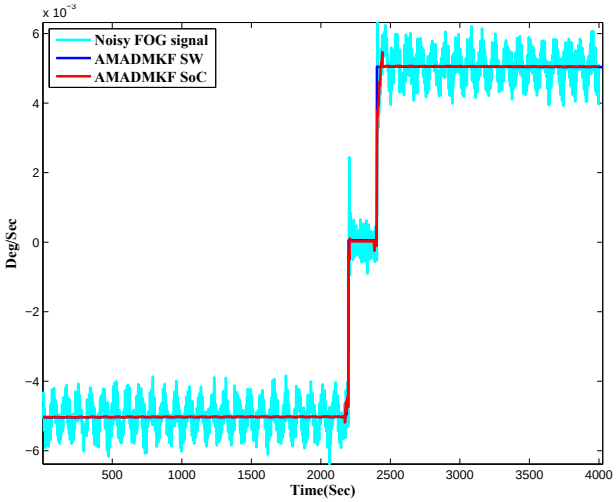


(a) Denoised *x-axis* data in static region (i)

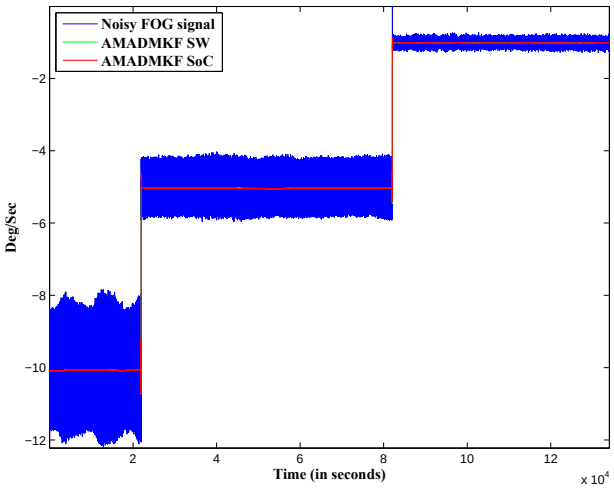


(b) Denoised *x-axis* data in static region (ii)

Figure 3.26: Comparison of algorithm vs. SoC implementation results (static region)



(a) Denoised *x-axis* data at multiple transitions in dynamic condition



(b) Denoised *x-axis* data at single transition in dynamic condition

Figure 3.27: Comparison of algorithm vs. SoC implementation results (dynamic condition)

Table 3.8: Standard deviation of *x-axis* data denoising using software and hardware

Rotation(Deg/(Sec))	Input	AMADMKF	SoC AMADMKF
0 (static)	0.01514121	0.0019	0.0019
-200	0.07903444	0.0019	0.0018
-175	0.06971800	0.0044	0.0044
-150	0.08232674	0.0013	0.0013
-125	0.08772601	0.0012	0.0011
-100	0.08254453	0.0011	0.0011
-75	0.07587522	0.0014	0.0014
-50	0.07272912	0.0021	0.0021
-25	0.16273886	0.0072	0.0072
-10	1.18274250	0.0218	0.0218
-5	0.52355471	0.0051	0.0051
0	0.09825683	0.0027	0.0027
5	0.36446447	0.3679	0.3572
10	0.10345610	0.0038	0.0037
25	0.52383367	0.0109	0.0109
50	1.14341674	0.0069	0.0069
75	0.16016677	0.0048	0.0048
100	0.07137773	0.0044	0.0044
125	0.07409974	0.0026	0.0026
150	0.07912759	0.0089	0.0089
200	0.08566183	0.0021	0.0021

Table 3.9: Execution time of AMADMKF algorithm in SW and HW

No of Frames	SW(μs)	HW(μs)	Acceleration factor ($\frac{SW}{HW}$)
2	548.67	8.46	64.85
10	2795.89	42.54	65.72

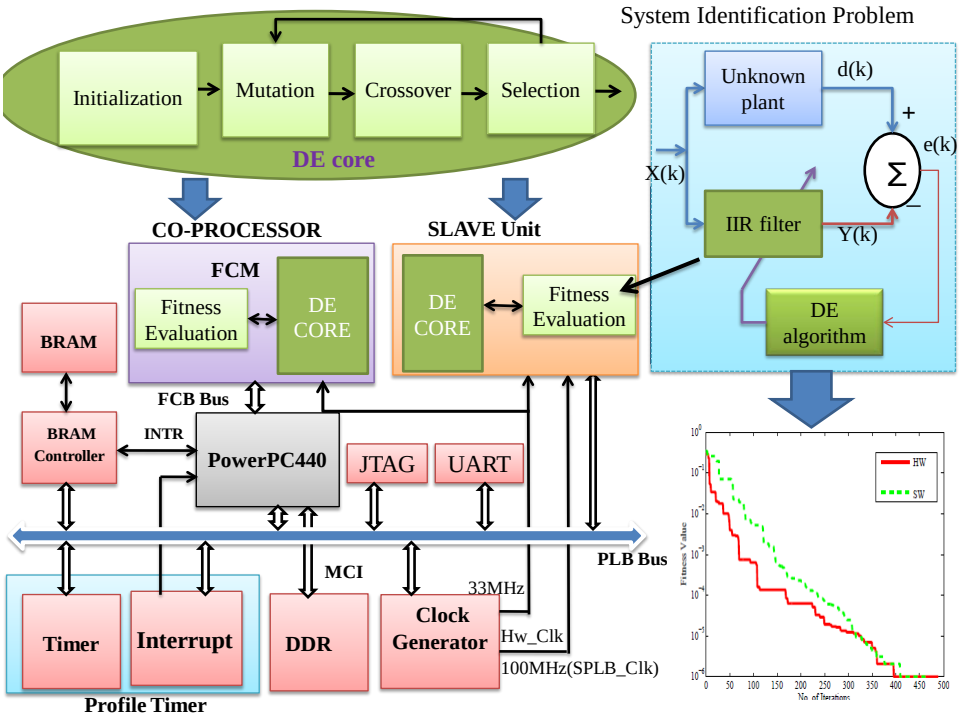
3.10 Conclusions

In this chapter, a new algorithm namely Adaptive Moving Average based Dual Mode Kalman Filter (AMADMKF), is proposed for denoising the Fiber Optic Gyroscope signal. The performance of the algorithm is compared with the Discrete wavelet transform (DWT), and the standard Kalman Filter (KF) algorithm. It is observed that AMADMKF algorithm improves the standard deviation and signal to noise ratio by a factor of 100 and 80 dB respectively. However, the KF, has shown competitive performance as AMADMKF algorithm while considering the standard deviation and signal to noise ratio as a performance indicator for denoising static and dynamic signal. A co-processor for AMADMKF algorithm

is developed and hardware IP simulation results are matched with the algorithmic simulation results. The performance of AMADMKF is compared with DWT and KF algorithm. It is concluded that the proposed algorithm is the optimum algorithm in terms of area and SNR, whereas KF algorithm is better in terms of execution time. A system on chip implementation of the AMADMKF algorithm using Xilinx Virtex-5FX70T-1136 FPGA is proposed and tested. The execution time of the developed AMADMKF IP is 65x faster than the equivalent software execution time in the PowerPC440 embedded processor.

Chapter 4

Coprocessor for Differential Evolution algorithm



This chapter presents the design and implementation of the coprocessor for computing DE algorithm. This is second case study of thesis. The details of proposed architecture of fixed and float DE algorithm along with System on Chip (SoC) implementation is also presented. The Intellectual Property (IP) is interfaced using both Auxiliary Processing Unit (APU) and Slave Unit (SU) interface techniques with the PowerC440 (PPC440) processor in SoC platform. The performances of the IP are evaluated with respect to acceleration and power consumption. Further more, a system identification using Infinite Impulse Response (IIR) filter is realized using the developed IP and the hardware acceleration is reported.

4.1 Introduction

Optimization is the branch of science that deals with methods to obtain best possible solution for a given set of problems. The quality of solutions are expressed in terms of a numerical values. Thus the aim of optimization is to select the best possible decision for a given set of circumstances. In recent years, the subject of computational science has matured and is widely used in science and engineering applications [99]. There are different Evolutionary Algorithms (EAs) such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Differential Evolution (DE) etc. for solving optimization problems, out of which DE is a popular method because i) it is simple to implement less complexity, ii) it has better performance in comparison with other EA algorithms and iii) it has less number of control parameters and less space complexity. Most of the evolutionary techniques have been implemented in a high end desktop computer/processors to solve optimization problems. The applications like Motion estimation [100], pole-placement design of infinite-impulse-response filter [101], future generation evolvable machines [102] use evolutionary algorithm to derive optimal solutions. These applications generally uses low-performance microprocessors with limited computational resources, rather than high-performance desktop personal computers/processors to execute the evolutionary algorithms. The time consuming evolution process limits the use of evolutionary algorithms in an embedded applications. This leads to slow execution speed of the algorithms in an embedded processor. In order to meet the real time execution speed requirement, one can either proceed with the parallelization of the algorithm or implement the design onto the hardware. Generally, embedded systems are designed to perform one dedicated task, which is different from gen-

eral purpose computer system to meet the real-time constraints and quality of the solution. There are several platforms to develop an embedded system with its own advantages and disadvantages and FPGA is one such platform [11]. In this work, we choose FPGA as development platform, because it has the flexibility to design entire system either in firmware or hardware, or the system can be partitioned into hardware and firmware.

DE algorithm has been widely accepted and implemented on general purpose processors, but mostly it is executed in an off-line mode of desktop simulation. It can be implemented in both embedded processor and hardware using either fixed point or floating point arithmetic. Although floating point DE gives better accuracy but at the expense of high computation cost. It leads to slowdown the execution of the algorithm in embedded processor approximately by 5-40x. In the literature, there is no reported work which implements the DE algorithm in hardware for accelerating its execution speed. Hence in this work, fixed and float DE IPs are developed. In floating point DE IP, the fabric FPU is used to carryout the floating point arithmetics. A complete SoC is built by interfacing the IP with the PPC440 using APU & SU interfacing techniques in the Xilinx FPGA development board. The performance of the DE IP for both the interfaces are evaluated. The objective of this work is to implement the DE in the FPGA platform to accelerate the optimization speed. This work focuses on only improving the speed of the optimization time and not to improve the quality of solutions. For improving the quality of solution different variants of DE algorithm can be explored.

In this work, basic DE algorithm is used for hardware implementation. It has three major computational steps, (i) random number generation, (ii) fitness function evaluation, and (iii) population update. For complex applications, execution time of the fitness function evaluation dominates the overall execution time of the DE algorithm. So in general the execution speed can be improved by designing hardware accelerators for fitness evaluation. In this approach, fitness function evaluation module can be in hardware (HW) and remaining logic of DE algorithm can be in the processor/ software (SW). In other way, the fitness function can be implemented in SW and remaining part of the algorithm in the HW (HW/SW co-design), in both the cases the bus transition time will dominate the execution time for evaluating the fitness function. On the other hand, if the total DE algorithm including fitness function evaluation is implemented in the embedded processor of FPGA, then there will be no acceleration in execution speed [103]. All the

above approaches will not give any additional improvement in terms of execution speed. So, an alternate choice is explored in which all the modules of DE including fitness function evaluation module are implemented in hardware, and use it as a dedicated hardware accelerator. There are two different interfacing techniques are adapted for hardware accelerators and these are being used for accelerating DSP algorithms [29]. In the first technique, the IP is interfaced using Auxiliary Processor Unit (APU) to the embedded processor (PPC440) and in other technique, it is interfaced as a Slave Unit (SU) with the shared local bus of embedded processor. The proposed accelerators are scalable in terms of the population size, number of generations and dimension, which can be modified by the user through the embedded processor. In the proposed design, both the fitness function evaluation and DE IP are realized in a single module rather than in different modules [104]. The proposed DE IPs in fixed and floating point modules are interfaced as APU to solve the Infinite Impulse Response (IIR) Filter based system identification problem. In this application IIR filter coefficients are obtained using the developed DE hardware accelerator.

4.2 Literature survey

Numerous hardware implementation of EAs have been described in literature. A comprehensive review of literature related to FPGA implementation of EA is tabulated in Table.4.1. A Complete Hardware Evolution (CHE) of GA was implemented on a single FPGA to evaluate single variable fitness functions [105], and reported that the execution speed is improved significantly as compared to its software implementation. However, this implementation has no provision to configure the GA parameters like mutation, crossover rates, population size and number of generations, and it cannot be directly interfaced to higher dimensional fitness functions. To overcome the above mentioned drawbacks in [105] a customized Intellectual Property (IP) of genetic algorithm was implemented in the Xilinx FPGA and integrated with PowerPC 405 processor based SoC and the speed enhancement up to 5.16x was achieved in Virtex-II Pro development kit [106]. A modular co-design architecture was developed for particle swarm optimization (PSO) algorithm [104], in which particle positions were updated in hardware whereas the fitness function was evaluated on a Nios-II embedded processor. Due to this approach, the design has a flexibility to modify the fitness

functions in the software depending on the applications. With this approach various embedded applications can be developed simply by changing the objective function. This design achieved speedup of 20x in Altera development kit. Hardware architecture of pipelined PSO (PPSO) was developed along with the parallel PSO framework which consists of multiple Nios -II processors using System-on-a-programmable-chip (SOPC) methodology and resulted speedup of 98x compared to the software implementation of the PSO algorithm in Altera development kit [107]. A modular, flexible and reusable multi-swarm PSO parallel hardware architecture was proposed to overcome the drawbacks of software implementation of the PSO algorithm using a Freescale micro-controller and Xilinx MicroBlaze soft processor core [108]. A hardware accelerator for parallel PSO (pPSO) algorithm was reported and validated its performance by optimizing test bench functions on Microblaze (MB) processor based SoC in a Virtex-6 development kit [109]. The PSO algorithm was tested on MB at 200MHz and compared the acceleration with a dedicated PSO IP, which shows 18-135x acceleration.

Table 4.1: Review of existing literature on FPGA implementation of evolutionary algorithms

Work	**Algorithm**	**Processor(Hz)**	**IP Freq (Max Freq)** MHz	**FPU Used**	**Speedup**	**Target Board**
[106]	GA	PowerPC (200)	50 (50)	No	5.16x	Xilinx Virtex-II Pro
[105]	GA	PC	- (50)	No	-	Xilinx SP3E
[104]	PSO	Nios-II (50)	50 (50)	No	20x	Altera DE2-70
[107]	PSO	4 Nios-II (50)	50 (76.3)	No	98x	Altera Stratix
[108]	PSO	Freescale (25)	25 (-)	No	359x-653x	MC9S12DP256B
		Microblaze (25)	25 (42.5) & 25(29.8)		37x-52x	Xilinx Virtex-II Pro Xilinx SP3E
[109]	PSO	Microblaze (200)	- (233)	No	18x-135x	Xilinx Virtex-6
[110]	GA	16 EM64T CPU(3.2G)	8-15(-)	Yes	1.3-3x	Altera Cyclone
[111]	GA	CPU (2.7G)	190(175)	Yes	7-116x	Xilinx Virtex-5
		GPU (Nvidia Quadro FX) (450M)	110(100)		7.3-12.3x	
[112]	PSO	CPU(1.6G) with MATLAB	50(94)	Yes	78-127x	Xilinx Virtex-5
[113]	PSO	Microblaze(50M) CPU(1.6G) with MATLAB	50(99)	Yes	6490-13820x 3.6-4.2x	Xilinx Virtex-5
[114]	PSO	Microblaze(50M) CPU(1.6G)	40(40)	Yes	6465-13888x 1.4 -3.1x	Xilinx Virtex-5
Present work [103]	DE	PPC440(200M)	33(65)	No	80x-150x	Xilinx Virtex-5
Present work[7]	DE	PPC440(200M)	50(120)	Yes	200x	Xilinx Virtex-5

Vijay et al., reported three different parallel processing architectures of GA to improve the performance in terms of execution speed. It uses multiprocessor (PGA), reconfigurable hardware using a pipelined architecture (HGA) and parallel architecture (PHGA). The parallel GA (PGA) in multiprocessor uses Message

Passage Interface (MPI) and executed on Rocks cluster with 16 EM64T CPUs each with a clock frequency of 3.2GHz. Similarly, HGA, PHGA were implemented on a reconfigurable Altera Cyclone FPGA with clock rate between 8 to 15 MHz. PHGA architecture uses both pipeline and duplicated hardware module for fitness evaluation to give additional parallelism. It is reported an acceleration of PHGA architecture for GA IP (1.3-3x) compared to software implementation, and this hardware was used to solve real-time scheduling problem [110].

Vaš et al., developed a hardware accelerator for cartesian genetic programming with multiple fitness units which uses intelligent memory organization and contains multiple virtual reconfigurable circuits to evaluate several candidate solutions in parallel [115]. Further, the hardware was tested with an application of image filter which showed significant speed up over 170x compared to equivalent software implementation on PPC440 in Virtex-5 FPGA.

Munoz et al., implemented the hardware architecture for parallel Particle Swarm Optimization (PSO) using Floating-Point (FP) arithmetic. In this implementation, PSO core was operated at 50 MHz and it achieved an acceleration of 78-127x compared to software implementation while optimizing test functions running at 1.6 GHz speed with 1 GB RAM [112]. The work was extended in [113], which implemented parallel self-adaptive PSO using higher precision with large dynamic range FP arithmetic to perform computations. This hardware used attractive-repulsive scheme to avoid premature convergence of PSO and gave an acceleration of 6490-13820x compared to equivalent software implemented in MB processor on a Virtex-5 FPGA (xc5vlx330). The fully parallel and partial parallel PSO implementation results were compared in similar test conditions [112, 113, 114]. From the Table.4.1, it shows that parallel PSO speed up the software application on Microblaze without FPU by 6465-13888x and partial parallel hardware speed up by 3595-6725x compared to parallel PSO.

The hardware implementation of DE in Graphical Processing Unit (GPU) using Computer Unified Device Architecture (CUDA) was developed for accelerating the execution speed of a co-evolutionary DE algorithm [116]. The computation time of co-evolutionary DE algorithm in CUDA architecture was compared with the computation time of the same algorithm implemented and was reported that the CUDA architecture significantly improves the execution speed [117]. However CUDA is not suitable for embedded applications due to its high power consumption and operating speed.

Gomez-Pulido et al., implemented the GA algorithm on FPGA, CPU and GPU with different fitness functions using FP arithmetic. Applications like i) error correcting codes optimization, ii) radio network design and iii) adjusting of X-ray diffraction profiler. The FPGA hardware has shown an acceleration of 7.3-12.3x and 7-116x compared to GPU and CPU respectively. The use of reconfigurable hardware lowers cost and power consumption compared to GPU, which is desirable when they are applied for intensive scientific calculations in parallel computing environments [111].

Evolutionary algorithms are used in applications like Infinite Impulse Response (IIR) based adaptive filtering, system identification etc. Evolutionary algorithms such as GA [118], swarm intelligence [119, 120], DE [121, 122] have been used to solve system identification problem. So in this work we used the DE IP for solving IIR system identification problem and evaluated the hardware acceleration.

4.3 Differential Evolution algorithm

Differential Evolution (DE) algorithm has been proved to be an efficient algorithm to accurately compute the global optimum solution for optimization problems and multi-modal optimal control problems [123]. DE employs real-coded variables and typically relies on mutation as the search operator [124]. It has evolved to share many features with conventional GA, like both maintain populations of potential solutions and use a selection mechanism for choosing the best individuals from population [125]. It is a parallel direct search method that employs a population of size NP, floating/fixed point encoded individuals or candidate solutions.

Basic DE algorithm has four major steps; (i) population initialization, (ii) mutation operation, (iii) crossover operation, and (iv) selection process as shown in Figure.4.1. The complete pseudo code of DE is given in Algorithm 1. The performance of DE accelerators are tested by optimizing a set of numerical test bench functions as tabulated in Table.4.2 (CEC 2005 and 2010 [126, 127]. These numerical functions include four low-dimensional and two high-dimensional functions as mentioned in Appendix-A. The DE algorithmic control parameters are tabulated in Table.4.3. The DE algorithm starts by initializing parameters and population values. The population of size NP are randomly generated within the predefined range, and these are passed to an objective function for evaluating the fitness values.

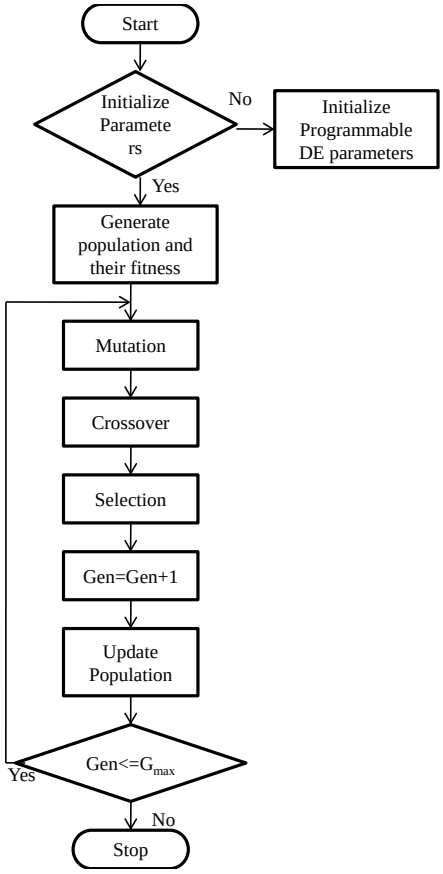


Figure 4.1: Flow chart for DE algorithm

In each generation, individuals of current population become target vectors. For each target vector, mutation operation gives a mutant vector, by adding the weighted difference between two randomly chosen vectors to a third vector. Next crossover operation generates a new vector, called trial vector, by mixing the parameters of the mutant vector with those of the target vector. If the trial vector obtains a best fitness value than the target vector, the trial vector replaces former target vector in the next generation. This process repeats until a maximum number of generation is reached. The flowchart of DE algorithm in shown in Figure.4.1.

Algorithm 1 Pseudo-code for the Differential Evolution Algorithm

Step 1: Read the control parameter values of the DE algorithm :
scale factor (F), crossover rate (CR), maximum number of iterations G_{MAX} the population size NP, and dimension (D) from user.

Step 2: Set the generation number Gen=0 and randomly initialize a population of NP individuals P_G=$[X_1^{(G)}, ..., X_i^{(G)}, ..., X_{NP}^{(G)}]$ with $X_i^{(G)}$=$[x_{1,i}^{(G)}, ..., x_{3,i}^{(G)}, ..., x_{D,i}^{(G)}]$ and each individual uniformly distributed in the range $[X_{min}, X_{max}]$, where X_{min}=$\{x_1^{min}, x_2^{min}, ..., x_D^{min}\}$ and X_{max}=$\{x_1^{max}, x_2^{max}, ..., x_D^{max}\}$ with $i = [1, 2, ..., NP]$.

Step 3:
while ($stopping\ criterion\ not\ satisfied$ **OR** $Max\ no.of\ iterations\ not\ reached$)
For i=1 to NP //do for each individual sequentially

Step 3.1: Mutation Step
Generate a mutant vector $V_i^{(G)}$=$\{v_{1,i}^G, ..., v_{D,i}^G\}$ corresponding to the ith target vector $X_i^{(G)}$ via the differential mutation scheme of DE as:
$V_i^{(G)} = X_{r1}^{(G)} + F.(X_{r2}^{(G)} - X_{r3}^{(G)})$
Vector indices $r1$, $r2$ and $r3$ are randomly chosen, where $r1$, $r2$ and $r3$ {1,...,NP}

Step 3.2: Crossover Step
Generate a trial vector $U_i^{(G)}$=$\{u_{1,i}^{(G)}, ..., u_{D,i}^{(G)}\}$ for the ith target vector $X_i^{(G)}$ through binomial crossover in the following way:
$u_{j,i}^{(G)} = v_{j,i}^{(G)}$, **if** $(rand_{i,j}[0,1] \leq CR\ or\ j = j_{rand})$
$u_{j,i}^{(G)}$=$x_{j,i}^{(G)}$, **otherwise**

Step 3.3: Selection Step
Evaluate the trial vector $U_i^{(G)}$
if $f(U_i^{(G)}) \leq f(X_i^{(G)})$, **then**
$X_i^{(G+1)}$= $U_i^{(G)}$
else
$X_i^{(G+1)}$= $X_i^{(G)}$
endif

Step 3.4: Increase the Generation Count
Gen = Gen + 1

Table 4.2: Benchmark functions used for performance analysis

No.	Function name	Dimension	Optimal Objective Value
Fun1	Rosenbrock	2	0
Fun2	Goldstein	2	3
Fun3	Sphere	3	0
Fun4	Variably dimensioned	4	0
Fun5	Shifted Sphere	32	0
Fun6	Shifted Schwefel	32	0

Table 4.3: Control parameters of the DE algorithm

Control Parameters	Value
Population Size (NP)	8,16,32
Total Number of Independent runs (G_{MAX})	1,50,100
Dimension (D)	8,16,32
Weighting Factor(F)	0.9
Crossover constant(CR)	0.9

4.4 Software profiling of DE algorithm

Profiling is an important step in the development of embedded applications. Through profiling, computationally intensive functions as well as most often computations inside these functions are identified. This information can be used to decide which part of the algorithm should be implemented in hardware (logic blocks of the FPGA) and which part in software (embedded processor of the FPGA). So, software profiling for an algorithm is necessary before implementing the algorithm into hardware. Software profiling of both the fixed and floating point DE algorithm is carried out on PPC440 processor with clock frequency set to 200 MHz. The execution time of the DE algorithm (in embedded processor) for optimizing six different test bench functions are tabulated in Table.4.4. In the Table.4.4, SW denotes execution time for optimizing a test function, in the embedded processor; 'float' and 'fixed' corresponds to floating point and fixed point DE variants respectively. It tabulates the average execution time (in msec) and percentage of standard deviation (Std%) of execution time of both arithmetic of DE algorithm for 20 independent runs implemented in the embedded processor (SW). This table show results with maximum iterations $G_{MAX} = 1, 50, 100$ and for different population sizes $NP = 8, 16, 32$. This table reveals that for optimizing high dimension test functions, fixed point software algorithm gives approximately 4.96 - 7.36x acceleration over the floating point software algorithm. For optimizing low

Table 4.4: Execution time of the DE algorithm implemented in software

		NP =8			NP=16			NP=32		
Test Function	G_{MAX}	**Float SW(ms) (Std%)**	**Fixed SW(ms) (Std%)**	**Acceleration factor**	**Float SW(ms) (Std%)**	**Fixed SW(ms) (Std%)**	**Acceleration factor**	**Float SW(ms) (Std%)**	**Fixed SW(ms) (Std%)**	**Acceleration factor**
Fun1	1	4.91 (3.2)	0.15 (2.8)	32.73	9.44 (2.5)	0.26 (2.2)	36.31	18.36 (1.4)	0.52 (1.2)	35.31
	50	181.05 (1.4)	5.38 (0.9)	33.65	332.37 (0.7)	9.69 (0.4)	34.30	641.81 (0.4)	18.82 (0.2)	34.10
	100	363.17 (1.1)	10.38 (0.5)	34.99	673.21 (1.4)	19.33 (0.4)	34.83	1,301.32 (1.1)	37.51 (0.2)	34.69
Fun2	1	8.01 (1.9)	0.18 (2.2)	44.50	15.02 (1.5)	0.31 (2.3)	48.45	28.73 (0.8)	0.61 (1.2)	47.10
	50	264.53 (1.4)	5.97 (0.9)	44.31	491.39 (0.9)	10.85 (0.4)	45.29	940.12 (0.7)	19.82 (0.2)	47.43
	100	536.05 (1.5)	11.96 (0.5)	44.82	994.24 (0.9)	21.64 (0.3)	45.94	1,897.45 (0.7)	39.26 (0.2)	48.33
Fun3	1	5.12 (1.3)	0.16 (2.7)	32.00	10.13 (1.9)	0.31 (1.3)	32.68	19.88 (0.8)	0.61 (0.6)	32.59
	50	199.54 (0.8)	5.83 (0.6)	34.23	371.94 (0.4)	11.08 (0.3)	33.57	720.03 (0.2)	21.64 (0.2)	33.27
	100	397.91 (0.8)	11.62 (0.5)	34.24	740.14 (0.3)	22.08 (0.3)	33.52	1,432.64 (0.2)	43.16 (0.2)	33.19
Fun4	1	9.99 (1.9)	0.23 (2.2)	43.43	19.36 (0.9)	0.45 (1.2)	43.02	38.38 (0.5)	0.84 (0.6)	45.69
	50	305.79 (0.6)	7.08 (0.4)	43.19	584.59 (0.3)	13.48 (0.2)	43.37	1,145.25 (0.1)	26.46 (0.2)	43.28
	100	612.92 (0.7)	14.11 (0.3)	43.44	1,178.46 (0.5)	26.94 (0.2)	43.74	2,304.13 (0.3)	52.77 (0.2)	43.66
Fun5	1	41 (1.7)	6 (1.6)	6.83	81 (1.2)	11 (1.5)	7.36	162 (1.2)	23 (1.5)	7.04
	50	1,132 (1.3)	207 (1.7)	5.47	2,234 (2.1)	411 (1.6)	5.44	4,439 (2.1)	809 (1.4)	5.49
	100	2,254 (0.8)	412 (0.9)	5.47	4,435 (0.9)	825 (2.1)	5.38	8,809 (1.1)	1,638 (2.1)	5.38
Fun6	1	85 (1.8)	15 (1.9)	5.67	170 (2.1)	30 (2.3)	5.67	339 (1.1)	62 (2.2)	5.47
	50	2,251 (1.2)	446 (1.1)	5.05	4,472 (1.5)	884 (1.7)	5.06	8,916 (2.3)	1,736 (2.1)	5.14
	100	4,476 (1.3)	891 (2.1)	5.02	8,745 (1.3)	1,764 (1.1)	4.96	18,483 (0.9)	3,537 (1.1)	5.23

dimensional functions the execution time 32 - 48.45x is faster compared to the floating point algorithm.

The profiling results of float and fixed point DE algorithm for optimizing Fun3 (Sphere) function are shown in Figure.4.2 and Figure.4.3 with NP and G_{MAX} as 8 and 50 respectively. Table.4.5 shows that the execution time of floating point operations dominate the execution time of other modules of DE algorithm like mutation, crossover and selection. This table also shows that the random number generator and remaining part of the algorithm consumes 42.87% and 40.22% of the total execution time respectively, whereas fitness function evaluation module consumes 10.39% of the total execution time. The remaining 6.52% of execution time is for floating to fixed and fixed to float number conversion. Due to this, fixed point implementation (32-bit data width) of the DE algorithm is chosen for developing the hardware accelerator.

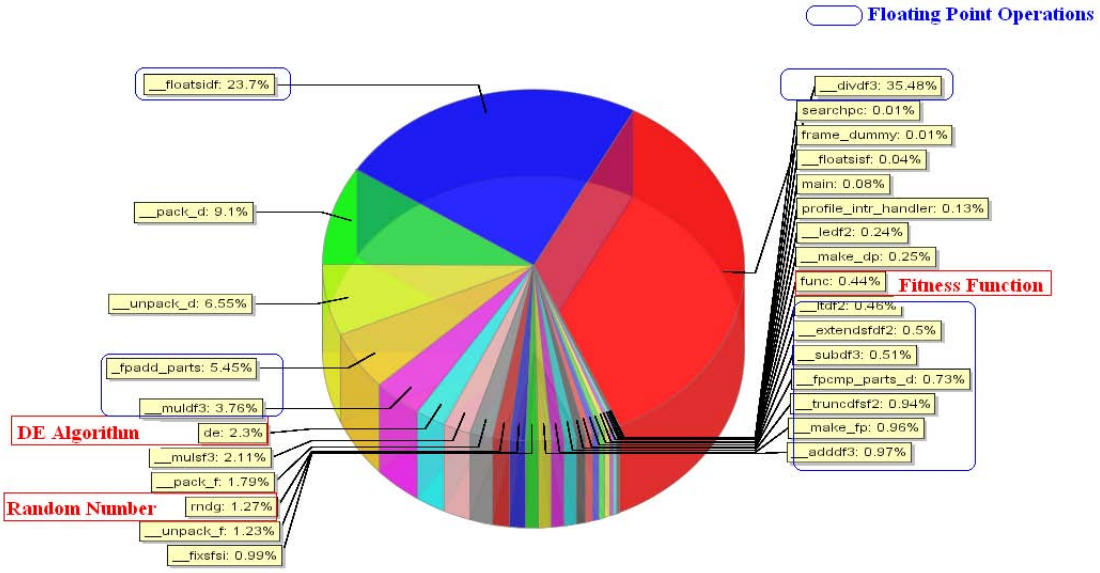


Figure 4.2: Software profiling results of floating point DE algorithm (G_{MAX}=8 and NP=50) for Fun3

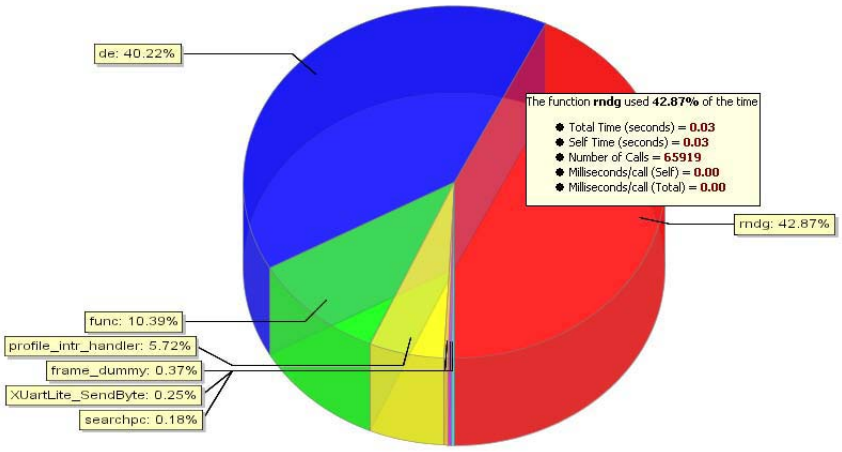


Figure 4.3: Software profiling results of the fixed point DE algorithm (G_{MAX}=8 and NP=50) for Fun3

Table 4.5: Profiling results: percentage of execution time of different DE modules in PPC440 processor (Fun3, G_{MAX}=50 and NP=8)

Type	DE algorithm	Objective function	RNG	Float Operations
Float	2.31%	0.44%	1.27%	95.98%
Fixed	40.22%	10.39%	42.87%	6.52%

Table.4.6 shows profiling results of three computational intense modules (DE algorithm, objective function, and Random Number Generation (RNG)) of the complete DE algorithm for different test functions with maximum generation and population size as 1000 and 8 respectively. From this table, it is observed that the algorithm (except fitness evaluation and RNG) takes more time in floating point environment as compared to fixed point environment for solving each individual functions. This is because of the complexity involved in floating point arithmetic. The floating point arithmetic is carried out in the software FPU library of the PPC440 embedded processor.

We have observed that in case of float DE IP, while optimizing the fitness function (Fun6) in the co-design platform, where fitness function is evaluated in the software and the remaining part of the algorithm is implemented in the hardware, the bus communication time is approximately 106 msec. If the complete DE algorithm along with fitness function is implemented in hardware, it takes approximately 128 msec. This concludes that bus communication overhead is dominating (i.e., 82.8%) the overall hardware execution time. This is observed when the embedded processor was operating at 200 MHz [103]. Similar kind of results observed in cased of fixed DE. So to reduce the bus communication overhead, the total DE algorithm including the fitness function evaluation can be implemented in the embedded processor of the FPGA. This approach may result to a marginal acceleration in optimization time. Both these approaches will not give any additional improvement in terms of the execution speed. So the alternate choice is to implement both the algorithm and fitness function evaluation in the hardware and use it as a dedicated coprocessor.

Table 4.6: Profiling results of the software (SW) DE algorithm (G_{MAX}=1000 and NP=8)

	DE algorithm		**Objective function**		**RNG**		**Float_operations**
Test Function	**SW float(ms)**	**SW fixed(ms)**	**SW float(ms)**	**SW fixed(ms)**	**SW float(ms)**	**SW fixed(ms)**	**SW float(ms)**
Fun1	50 (1%)	30 (43%)	10 (0.21%)	10 (14%)	40 (0.85%)	30 (43%)	4,621 (97.88%)
Fun2	60 (1%)	30 (43%)	60 (0.81%)	10 (14%)	40 (0.54%)	30 (43%)	7,228 (97.83%)
Fun3	90 (2%)	30 (38%)	10 (0.19%)	10 (13%)	50 (0.93%)	40 (50%)	5,220 (97.20%)
Fun4	50 (1%)	40 (44%)	20 (0.26%)	20 (22%)	40 (0.51%)	30 (33%)	7,674 (98.58%)
Fun5	1,488 (5%)	1,245 (69%)	294 (0.97%)	255 (14%)	520 (1.72%)	303 (17%)	27,944 (92.38%)
Fun6	1,824 (6%)	1,330 (29%)	3,640 (11%)	2,983 (64%)	570 (1.72%)	334 (7%)	27,042 (81.75%)

4.5 Hardware architecture of DE algorithm

The block diagram and detail hardware architecture of DE algorithm is shown in Figure.4.4 and Figure.4.5 respectively. The main advantage of this is the hardware concurrency due to the pipelining of the architecture. It is designed in structural hierarchy and having modules as Memory, initialization, Mutation, Crossover, Selection, Random Number Generator and Fitness evaluation modules and their controllers. A Control Finite State Machine (FSM) module is used to synchronize the above mentioned modules as shown in Figure.4.5. Random number generation (RNG) module uses a Linear Feedback Shift Register (LFSR) to generate pseudo random numbers with in the range of 0 to 1. RNG module reads the random seed and inputs to several other modules as shown in Figure.4.4. In floating point DE IP, Floating Point Unit (FPU) module is used to perform the floating point operations involved in various phases of DE algorithm. In FPU, operations like addition, multiplication, division and comparison are performed using IEEE-754 supported Xilinx FPU core 4.0 [128], whereas in software these operations are performed using soft FPU library or hard FPU in MB/PPC440. In case of fixed DE IP these arithmetic operations are calculated on fixed point data path. The FSM has five states i.e. idle (S0), initialization (S1), operation (S3/S4/S5), waiting (S2) and reading states (S6). In the idle state, all modules are in reset condition. The random nature of DE algorithm possess irregular timing for each stage of the architecture pipeline, thus it requires handshaking to synchronize the communication between different modules. This handshaking brings out additional overhead in pipeline operation in the form of wait states to finish the previous operation. When inputs such as maximum number of generations (G_{MAX}), population size (NP), dimension (D), *start* and *Enable* are available at the initialization module, FSM enables the memory module. For fixed/float DE initialization module, population memory (8/16 Kbyte) and fitness memory (128/256 bytes) modules are initialized by randomly generated population members and these are passed to fitness evaluation module to get their fitness value and are stored in their respective memories. Maximum values of NP and D are set to 32 for both float and fixed DE. During the operation state, control FSM enables internal modules (mutation, crossover, selection as well as memory module) and their controllers. These modules are executing in pipelined manner iteratively until the maximum number of iterations G_{MAX} is reached. The final state of FSM is the reading state.

In this state the best fitness value is available at output register.

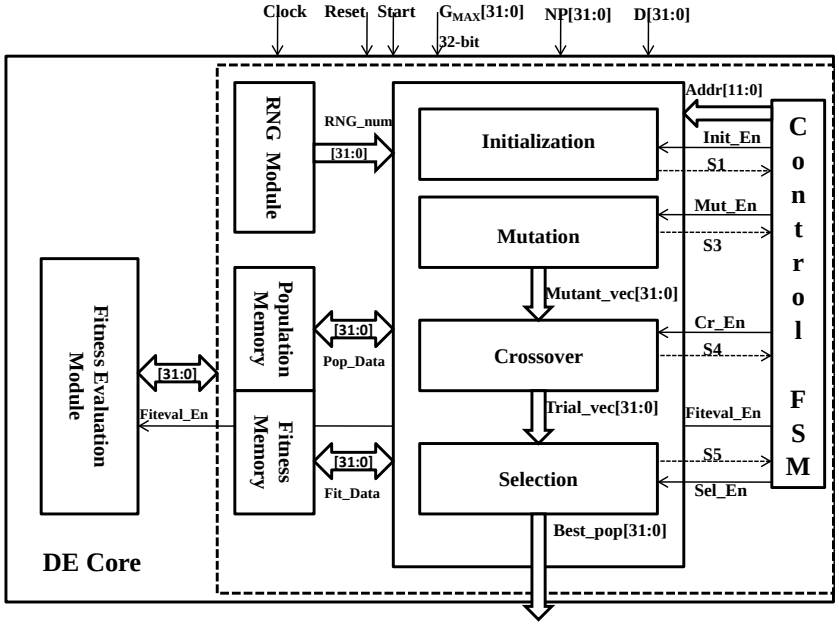


Figure 4.4: Block diagram of fixed DE hardware

In the present work, we propose both fixed [103] and floating point DE architectures [7]. The difference between these two are only the Floating Point Unit (FPU) core. In case of floating point DE core, the floating point core is used for division, multiplication, and adder etc., whereas in fixed point DE core these things are implemented with in the modules itself as shown in Figure.4.4. In floating point IP core the population memory and fitness memory are high compared to fixed point (32-bit) IP core. The details of individual modules of fixed/float DE IP core is explained below.

4.5.1 Initialization module

The initialization module has two separate memories, one is for storing the population values (Population Memory) and other is for storing their corresponding fitness function values (fitness memory) as shown in Figure.4.7. During the initialization state, population values of all the particles (i.e. of size $NP \times D$) are randomly generated within the range of $[X_{min}, X_{max}]$, and stored in the population memory of size (4/8Kbytes). The population values are accessed from the population memory by using a 12/13-bit address. Each population member is of

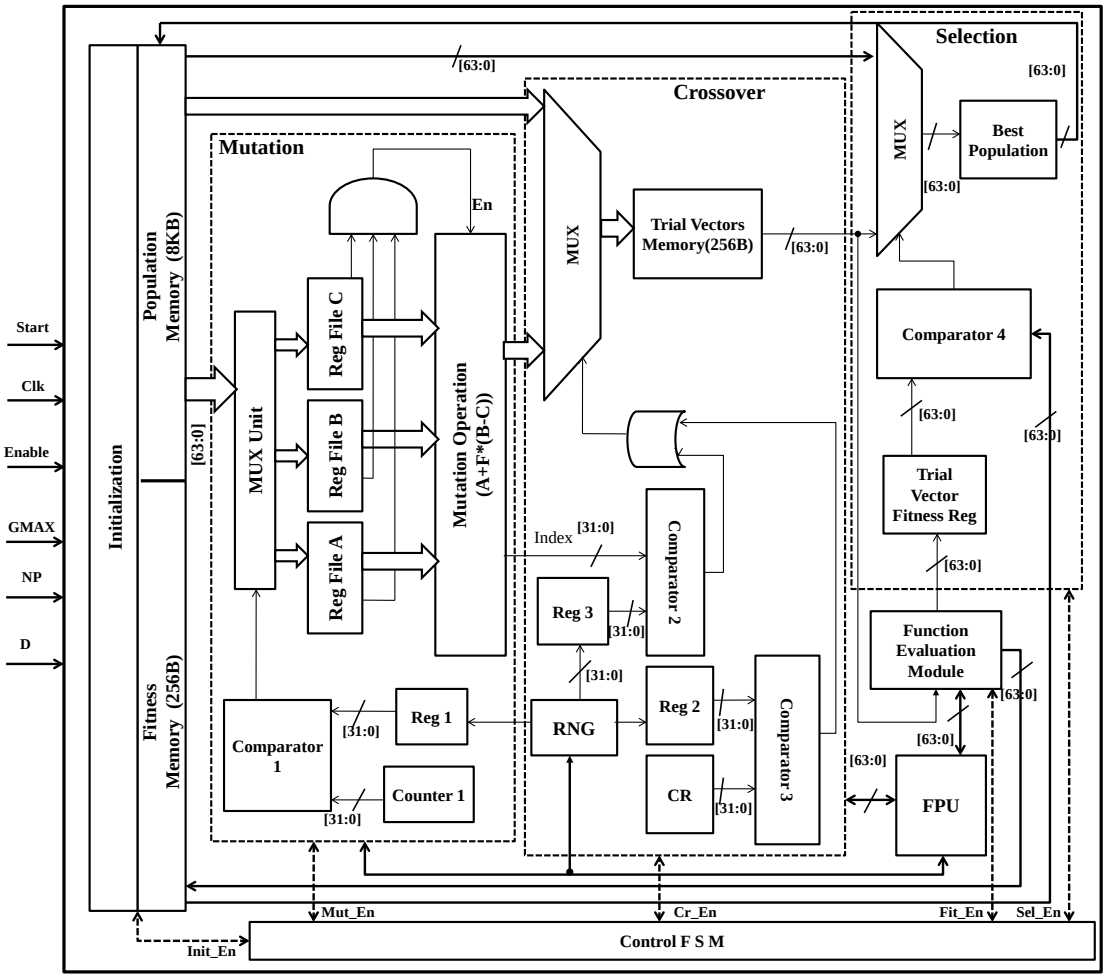


Figure 4.5: Hardware architecture of float DE Algorithm

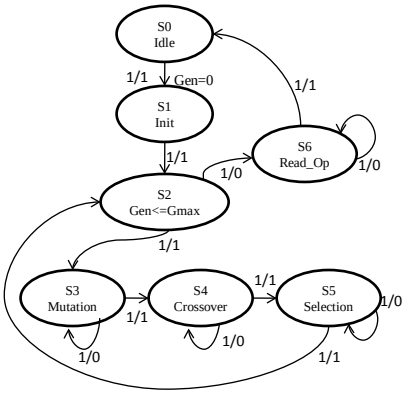


Figure 4.6: Control unit design for hardware implementation of DE algorithm

dimension D (number of variables) and each variable is of size 32/64 bits. The maximum values of NP and D are set to 32. These values are input to the fitness

evaluation module and after evaluating the fitness function the fitness values (of size 32/64-bit) are stored in the fitness memory of size (128/256 bytes). This process is repeated for all the population members.

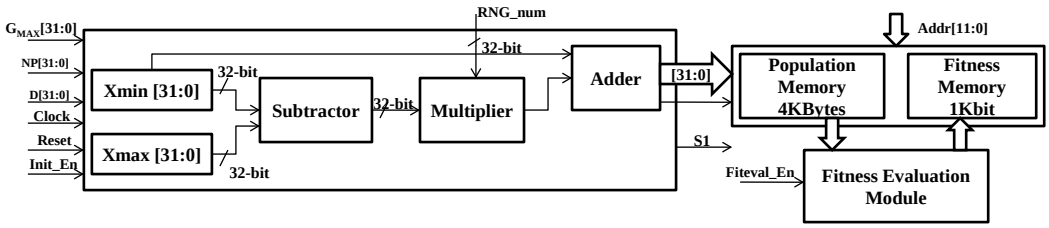


Figure 4.7: Initialization module

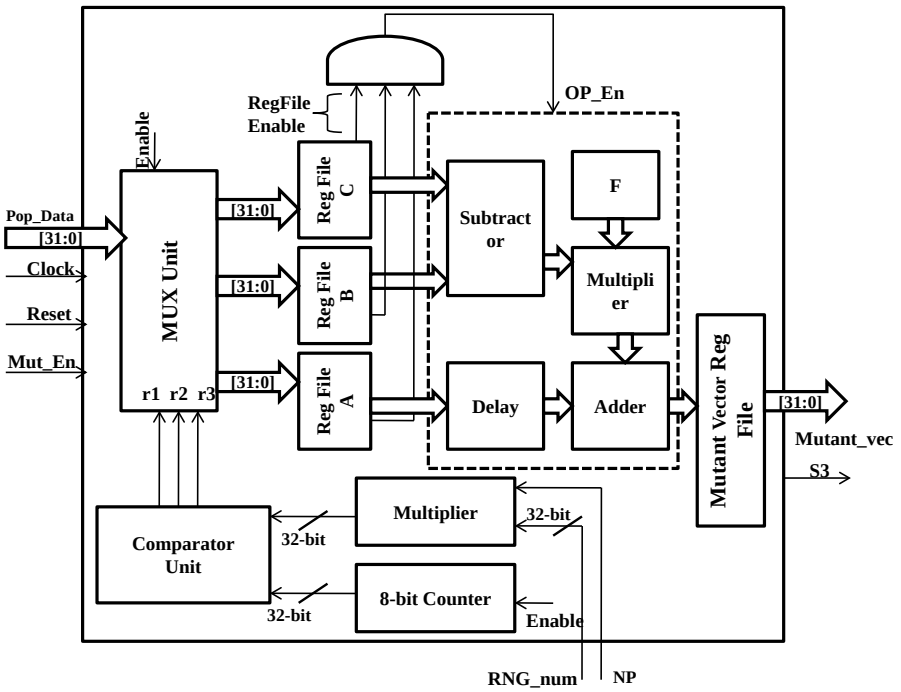


Figure 4.8: Mutation module

4.5.2 Mutation module

After the population is initialized, mutation operation is performed by the mutation module. A mutant vector is generated for every target vector from the current population. In this module a mutant vector of size (128/256) bytes is generated for each population member. Three distinct vector indices r_1, r_2 and r_3 are generated in the range of 1 to NP by comparing the counter value with the value

of multiplier. These indices are connected to the select lines of a MUX unit. Three distinct target vectors each of size (1/2) kbits are obtained from the MUX unit as shown in Figure.4.8. Then the mutation operation is performed by difference of any two of these selected three vectors scaled by a factor F and this difference is added to third one to obtain the mutant vector of size (128/256) bytes. A mutant vector is generated for all the population member of all dimensions.

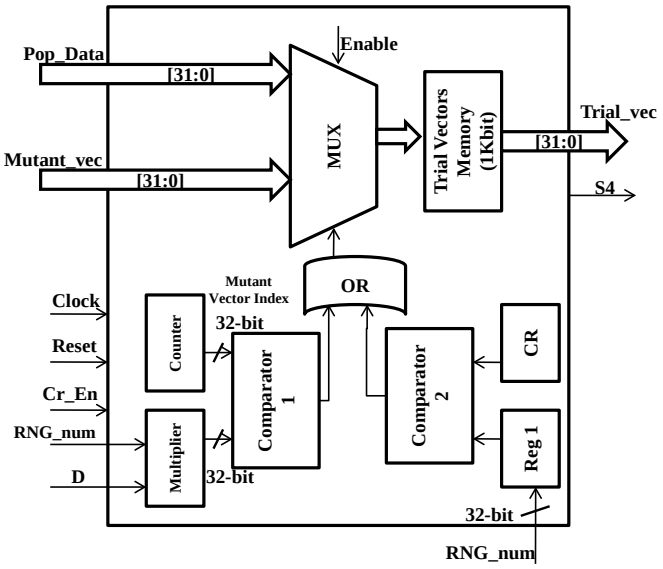


Figure 4.9: Crossover module

4.5.3 Crossover module

The crossover operation is mainly responsible to increase the diversity among the mutant vectors. A trial vector is generated from the output of crossover module with a crossover probability CR as shown in Figure.4.9. This crossover rate controls the diversity of the population and helps the algorithm to escape from the local optima [123, 129], and ensures that the trial vector gets at least one vector from the mutant vector. The register $Reg1$ has a random number stored in it. The output of $Reg1$ and CR are input to the comparator 2 module. The multiplier output and index of population member are input to the comparator 1. The output of both comparator 1 and 2 are input to a logic OR gate. The output of crossover module is either the mutant vector or the population vector as selected by the MUX unit.

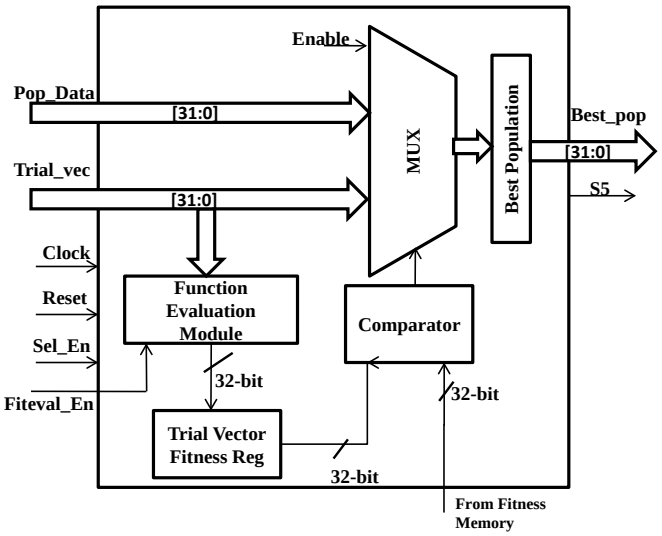


Figure 4.10: Selection module

4.5.4 Selection module

The output of crossover module is the trial vectors. These are input to the selection module as shown in Figure.4.10. The fitness value of trial vector is evaluated by using the fitness evaluation module and if it is less than the fitness of the current population member then it selects the input as trial vector else the current population member is selected as the new population member. The output of multiplexer (MUX) is the updated value of the current population memory. This process is repeated for all the iterations to improve the fitness of individuals and the process is stopped when the maximum number of generations is reached.

4.5.5 Fitness Evaluation module

The Fitness module evaluates the fitness of each individual in accordance with different fitness functions. For each generation, fitness values (32/64-bit) are evaluated for each population using fitness functions and the updated population and fitness values are stored in the population and fitness memory (128/256 bytes) respectively. In this work different test-bench functions as in Table.4.2/Appendix-A are used for fitness evaluation.

4.5.6 Random Number Generator module

Random number generation (RNG) module has great importance for the proper operation of the DE. A Linear Feedback Shift Register (LFSR) is used for generating random numbers, as it is easy to implement and it produces fairly good pseudo-randomness. This module generates random numbers for the initial population module, selection module, crossover and mutation modules. The seed for random number generator is programmable and it is initialized to a non-zero value. If all zero value appears in the seed, then XOR operations continues to generate zeros and output becomes always zero. The architecture of 32-bit LFSR with maximum length polynomial $X^{32} + X^{22} + X^2 + X^1 + 1$ is shown in Figure.4.11. This module generates $2^{32} - 1$ random numbers.

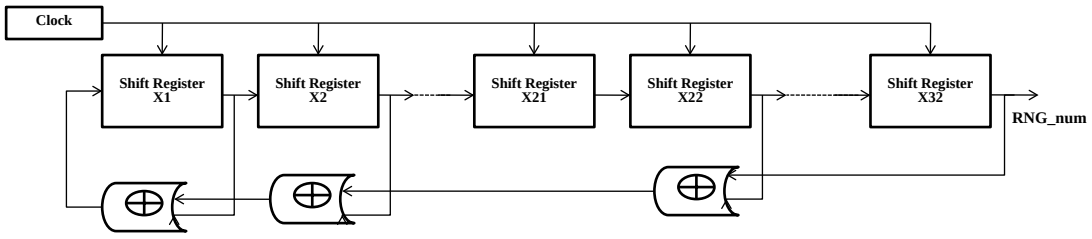


Figure 4.11: Hardware architecture of 32-bit LFSR for random generator

4.5.7 Floating Point Unit

This module performs floating point operations required for mutation, crossover, selection, fitness evaluation and random number generator modules. The operations like addition, multiplication, division, comparison and square-root are executed on the hardware using the IEEE-754 supported Xilinx FPU core 4.0 [128] as shown in Figure.4.5. It is also responsible for converting the data formats like float to fixed, float to integer, double to float and vice versa. This core can be customized for operation, word length, latency, and interface. It allows lower latency as compared to FPU of embedded processor.

4.6 Programmable System on Chip (PSoC) platform for DE algorithm

PSoC is a programmable integrated system that has configurable processors, peripherals, memories, custom IP on a single FPGA. The proposed PSoC platform for implementing the DE algorithm is shown in Figure.4.12. PPC440 processor communicates with external peripherals such as DDR2, Block-RAM (BRAM) Memory controllers, UART (RS-232), Timer and Interrupt controllers, JTAG Controller, Clock generator via processor local bus (PLB). PPC440 is preferred over Microblaze processor due to its high speed of operation and efficient resource utilization. DDR2 and BRAM controllers are used for storing heap and stack of program and data. UART is used for serial data transfer between the end user and processor. Timer and interrupt controllers are used for profiling the application. The clock generator provides necessary clock signals to all the modules and peripherals. USB JTAG controller is used to download the bitstream from host computer to FPGA board. PPC440 is directly coupled to the Auxiliary Processing Unit (APU) controller, which provides flexible high-bandwidth interface to DE Coprocessor via Fabric Coprocessor Bus (FCB). The coprocessor operates as an extension to the PPC440 as explained in Section 2.5.4. The SU and APU of floating and fixed DE IP core frequency is adjusted by a clock generator and set to 33 MHz. However it can be increased up to Max-freq of IP core but need to maintain the desired clock ratio of processor to PLB. The complete SoC system is shown in Figure.4.12. It has the following features:

1. It has PPC440 hard-core and Microblaze soft-core processors with clock frequency (200/125)MHz separately.

2. It consists of several logic resources such as 6-input look-up tables (LUTs), DSP48E, and Slices which are useful to program signal processing algorithms.

3. PLB is used as a shared local bus for all internal data communication between processor and peripherals except APU and DDR2.

4. Phased Locked Loop (PLL) or Digital clock manager (DCM) is used for clock management in SoC system as well as for custom IP clocking (DE IP).

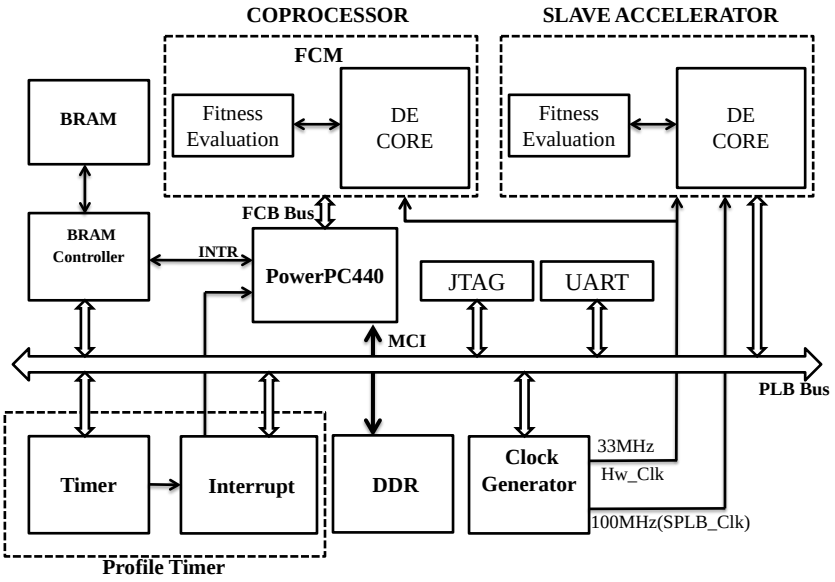


Figure 4.12: PSoC platform for DE algorithm

5. The floating point DE IP core has separate FPU, RNG, Fitness evaluation modules. These are connected to DE core internally. The descriptions of these modules are explained in 4.5 sections.

In this platform, the developed DE IP cores (both fixed and float architectures) are interfaced using SU and APU interfacing techniques, for evaluating performances. To accelerate the execution time, in the first case, DE core is interfaced as a SU to the embedded processor by using the shared bus, and in the second case the DE IP is interfaced as an APU of the PPC440 processor. The interface detail is shown in Figure.4.13. Both accelerators (either APU or SU) use the same DE IP core with some modifications in the wrapper logic to make it compatible with the bus interface. The DE IP has both the optimization algorithm module and fitness function evaluation module. These two modules are internally connected to minimize data transfer between DE algorithm and fitness function evaluation. This also helps to minimize the number of read/write cycles.

1. In the APU interface the Master of PPC440 is connected to the slave of the DE IP core through FCB.

2. In the SU interface of DE IP is connected to the processor through PLB .

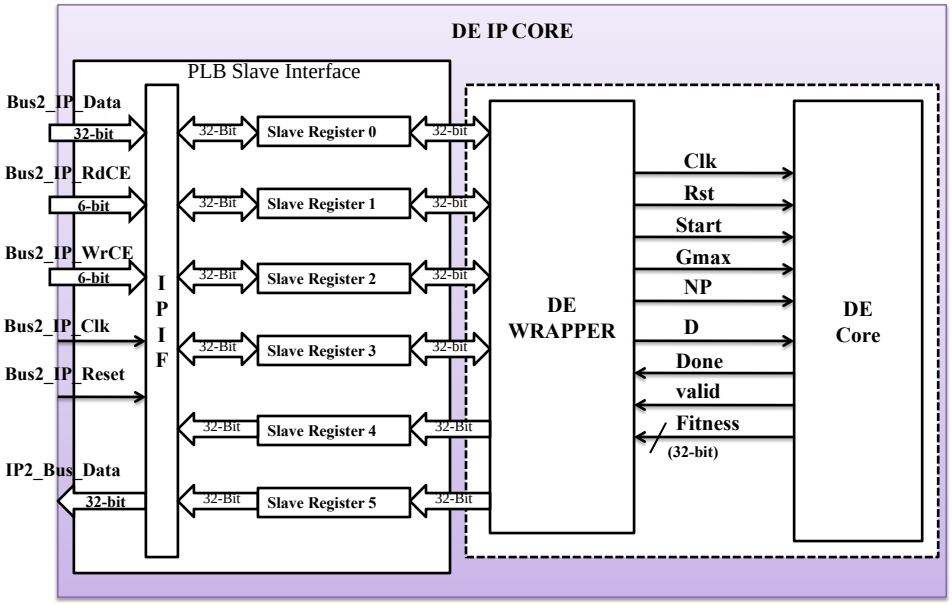


Figure 4.13: Design of slave peripheral in SoC

4.6.1 Interfacing the DE IP as a Slave Unit

The DE IP core is interfaced with the processor local bus through an Intellectual Peripheral Interface (IPIF) as shown in Figure.4.13. The DE IP core has six slave registers among which first four registers are for read/write operations of $Start$, G_{MAX}, NP, and D signals. The other two are used for reading output from the DE core through wrapper control logic. The DE wrapper has a control logic to receive the values of $Start$, G_{MAX}, NP, and D from the processor using Bus2IP_Data bus. According to the control logic, once the $Start$ signal is logic high, it retains this state for four clock cycles before becoming logic low. When all the inputs such as G_{MAX}, NP, D, and $Start$ are available to the DE core, it starts execution. After the defined number of iterations set by G_{MAX}, $Valid$ signal becomes logical high, and the slave register 5 reads the fitness value from the DE core through IP2Bus_Data. In this interface, Bus2IP_RdCE and Bus2IP_WrCE are the associated read/write qualifiers respectively. For floating point DE IP, the output data is interfaced to two slave registers because output data width is 64-bits and each register is of 32-bit width.

4.6.2 Interfacing the DE IP as an Auxiliary Processor Unit

The second method for developing the hardware accelerator is to build a fabric coprocessor module of DE core and interface to the embedded processor (PPC440) through APU interface. The hardcore APU controller bridges the processor APU interface and the external FCM interface. APU controller along with FCM behaves as a coprocessor. The extended instructions of PPC440 are used to communicate "to and from" the APU [4]. Since the APU is independent of processor to peripheral interface, it does not add any extra overhead to the PLB. The PPC440 supports three primary types of instructions for using the APU [4]. In this work, *load/store* instructions are used for accessing the APU. In this mode a maximum of 128 bits of data can be transferred in a single clock cycle or it can be transferred as four sets of 32-bits. The details of interfacing DE IP with embedded processor using APU interface is shown in Figure.4.14. The FCB bus is specifically targeted to host DE IP, which requires direct access and intervention of the processor instructions.

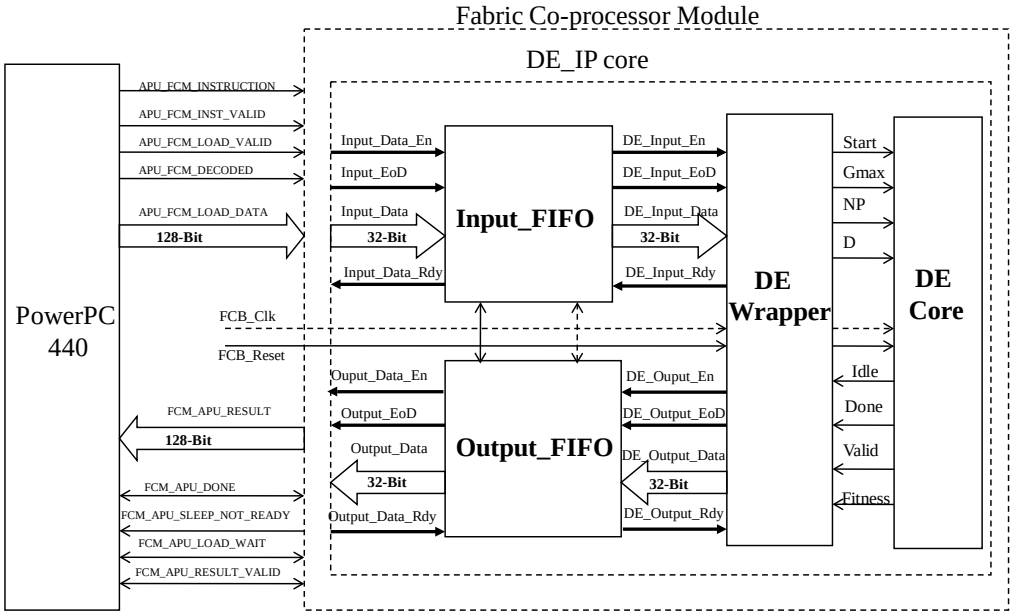


Figure 4.14: Interfacing of DE APU with PowerPC Processor

Figure.4.14 has two asynchronous FIFOs (depth of four and width of 32 bits) interfaced at the input and output of the DE core. The input signal is processed as a stream and each stream has four samples and three of which are used for G_{MAX}, NP and D. The remaining sample is used for checking whether the FIFO

is 50% full or not. In this architecture, *Output_Data* and *Input_Data* are two 32 bit width data buses for data input and output of the IP core respectively. The working principle of the DE IP core is described as below.

1. PowerPC writes the input data G_{MAX}, NP and D in three clock cycles. The IP core receives data from the PowerPC, till the FIFO is full. This is ensured by the control signal *Input_EoD*. When the FIFO is 50% full *Input_EoD* becomes logical high.

2. When the FIFO is 50% full, it will enable *DE_Input_En* as logical high, and when the IP core is ready for processing it will give a handshaking signal *DE_Input_Rdy* as logical high. The FIFO sends the data to the IP core till *DE_Input_EoD* is logical high.

3. When the IP core processes only single sample on the stream, it gives *DE_Output_En* as logical high and this is acknowledged by the output FIFO with handshaking signal *DE_Ouput_Rdy*. When this logical signal is high then the IP core sends the processed samples to the output FIFO till *DE_Output_EoD* is high.

4. When the output FIFO is full, FIFO will send back the data to APU of PowerPC processor.

The APU wrapper contains two different modules namely IP_APU and APU_IP. The APU_IP module receives data from the processor and sends it to DE IP whereas, the IP_APU module receives the final solution from the DE IP core and sends it to the processor (PPC440).

4.7 Experimental setup

In this work, the basic DE algorithm is considered for coprocessor implementation. The DE algorithmic control parameters are tabulated in Table.4.2. The computation time/acceleration factor is considered as the main performance indicator to measure the performance of the accelerators. The DE software code is ported into the PPC440 processor using 32 bit fixed and floating point C code, later algorithm is coded in Verilog language for implementing in the hardware. An Intellectual Property (IP) core for DE algorithm is developed and simulated using Xilinx ISE

10.1, then a synthesized IP core is developed and subsequently a coprocessor is designed for accelerating the DE algorithm. The performance of the coprocessor is evaluated by optimizing six numerical benchmark functions used in CEC 2005 and 2010 competitions [126, 127]. Due to the empirical nature of DE algorithm, evolution parameters are subject to modification. In the proposed coprocessor, population size (NP), number of generations (G_{MAX}) and dimension (D) can be modified by the users through the embedded processor without redesigning the hardware. The developed fixed/float DE IP cores are tested in two different configuration of SoC. In first case, the effect of hard FPU in embedded (PPC440/MB) processor is studied by optimizing three benchmark test functions (Fun2, Fun4, Fun6). The DE software is tested in different processors i.e. X86 (3GHz), MB and PPC440 (125MHz) with enabling/disabling hardware FPU library. In this configuration, bus frequency is set to 100MHz and SU of float DE IP core is set to 50MHz. In second case, the effect of SU, APU interfaces for both fixed & float DE IPs are studied by optimizing six test bench functions tabulated in Table.4.2. In this configuration, processor (PPC440) clock is set to 200MHz, bus frequency is at 100MHz and the DE IP core is set to 33MHz. The performance of APU interface is compared with the SU interface [7].

4.8 Results and Analysis

In this section simulation, resource, timing, convergence and power results of fixed and floating DE IP are presented. Furthermore, power analysis in SoC platform is presented.

4.8.1 Simulation results

This section presents simulation results of float/fixed DE IP core. The RTL code of DE fixed and float architectures are simulated. The IPs are simulated in Xilinx ISE 10.1 functional simulator.

For functional verification, fixed/float DE IP with interface wrapper logic and the DE core are simulated using a test-bench and the results are compared with DE processor results. The simulation results are shown in Figure.4.15 for Fun3 function with G_{MAX}=1, NP=8 D=3. When DE_Output_Rdy signal is logic high, the resultant fitness value is available at DE_Output_Data port which is in fixed

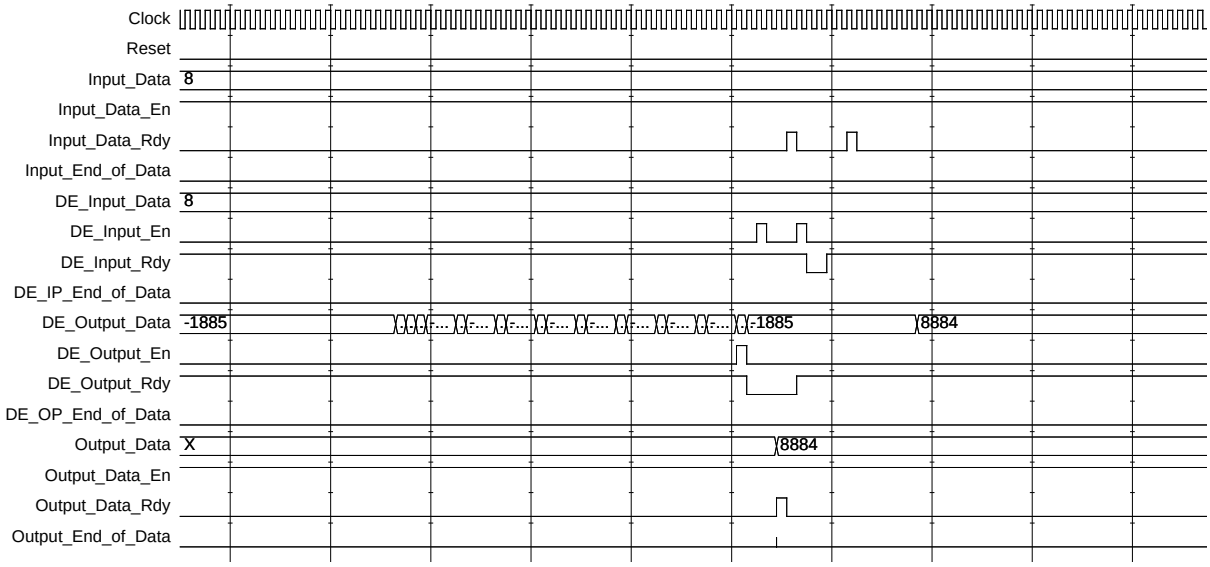


Figure 4.15: Functional simulation of Fun3 fixed DE IP Core (G_{MAX}=1 and NP=8)

point format. After logic high on DE_Output_Rdy signal, DE_Input_Rdy is high due to scheduling for next set of G_{MAX}, NP and D values. From the results, it is observed that the IP core is consistently giving the same results. The DE IP output is scaled and converted to single precision floating point format for comparing the results with output of the processor. For functional verification of float DE IP SU interface logic, DE IP core is simulated using same test bench and the results are compared with DE IP core output as shown in Figure.4.16. From this figure, it is observed that the interface logic of PLB (IP2bus_data) and the DE core output (out_data) consistently give the same result.

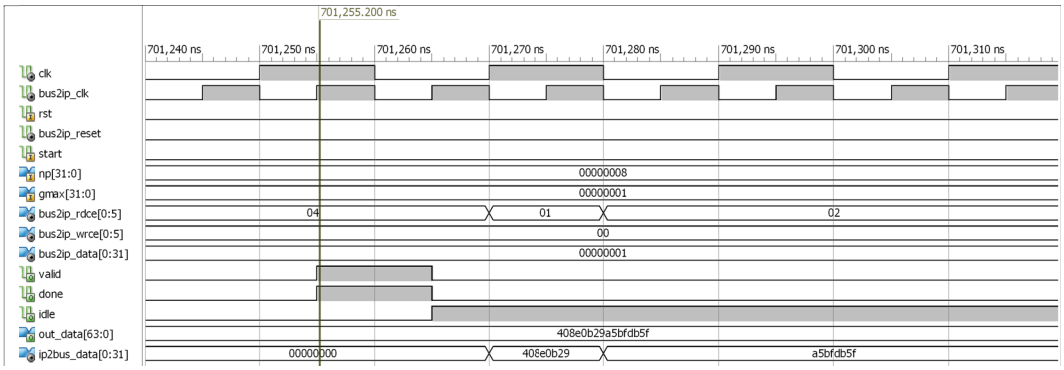


Figure 4.16: Functional simulation of Fun3 float DE IP core(G_{MAX}=1 and NP=8)

Table 4.7: Resource utilization of floating point DE IP core

Test Function	BRAM	DSP48E	Slice Registers	Slice LUTs	Slices	LUT-FF Pairs	MaxFreq (MHz)
Fun1	27(18%)	34(26%)	7576(16%)	10298(22%)	3588(32%)	5302(42%)	120.279
Fun2	6(4%)	84(65%)	10357 (23%)	14583(32%)	5257(46%)	7478(42%)	100.351
Fun3	6(4%)	20(15%)	7056(15%)	10306(23%)	3774(33%)	4668(36%)	98.502
Fun4	10(6%)	20(15%)	6963(15%)	9898(22%)	3065(27%)	5455(47%)	120.598
Fun5	21(14%)	20(15%)	6943(15%)	9646(21%)	3520(31%)	4635(38%)	101.491
Fun6	20(13%)	20(15%)	7129(15%)	9807(21%)	3486(31%)	5063(42%)	98.087

Table 4.8: Resource utilization of fixed point DE IP core

Test Function	BRAM	DSP48E	Slice Registers	Slice LUTs	Slices	LUT-FF Pairs	MaxFreq (MHz)
Fun1	3(2%)	61(47%)	2888(6%)	3936(8%)	1586(14%)	1711(33%)	39.33
Fun2	3(2%)	77(60%)	3150(7%)	7315(16%)	2546(22%)	2152(25%)	36.04
Fun3	3(2%)	20(15%)	3097(6%)	3928(8%)	1522(14%)	2031(40%)	67.04
Fun4	3(2%)	61(47%)	2883(6%)	4065(9%)	1625(14%)	1688(32%)	60.56
Fun5	10(6%)	42(32%)	2849(6%)	3667(8%)	1317(11%)	1890(40%)	64.67
Fun6	10(6%)	41(32%)	2886(6%)	3753(8%)	1485(13%)	1682(33%)	64.72

4.8.2 Synthesis results

The RTL code of floating and fixed point of DE IP cores are synthesized and resource utilization for optimizing different test functions are tabulated in Table.4.7 and Table.4.8 respectively. The resource utilization result shows that the fixed DE IP core consumes lesser resources and lower operating frequency compared to float DE IP. The resource utilization is more for optimizing higher dimension/complex test bench functions.

4.8.3 Timing results

The timing specifications of the proposed float and fixed DE IP core is 33 MHz. The maximum operating frequency of the synthesized float and fixed DE IP core for optimizing different fitness functions is also tabulated in Table.4.7 and Table.4.8 respectively. After synthesizing the fixed and float DE core, accelerators are developed by using both (SU and APU) interfaces. The SoC platform is tested by enabling and disabling FPU of the processor. The performance of the different processors i.e. X86, PPC440 and Microblaze (MB) are evaluated and compared

Table 4.9: Average execution time of DE algorithm in X86, PPC440 and MB processors

		NP =**8**			NP=**16**			NP=**32**		
Function	G_{MAX}	**X86(us) (Std%** [1]**)**	**PPC440(ms)**	**MB(ms)**	**X86(us) (Std%)**	**PPC440(ms)**	**MB(ms)**	**X86(us) (Std%)**	**PPC440(ms)**	**MB(ms)**
	1	3.04 (1.2)	0.23	6.18	5.12 (1.3)	0.35	12.74	10.08 (1.2)	0.78	23.84
Fun2	50	110.61 (1.8)	7.14	194.74	195.52 (1.9)	13.76	376.68	357.12 (1.8)	26.34	736.49
	100	222.06 (2.2)	14.54	396.46	383.83 (2.2)	25.62	778.84	709.93 (2.1)	52.68	1,504.36
	1	4.32 (1.1)	0.32	8.25	7.63 (1.5)	0.63	15.86	14.82 (1.5)	1.14	30.38
Fun4	50	175.23 (1.5)	10.35	245.12	286.22 (2.5)	19.68	468.48	491.24 (2.2)	38.38	916.78
	100	296.31 (2.1)	20.48	484.26	525.91 (2.6)	39.28	936.18	995.56 (2.4)	77.62	1,818.96
	1	60.24 (1.5)	3.27	60.85	113.52 (1.2)	7.24	131.58	224.04 (1.6)	14.82	260.86
Fun6	50	1,607.08 (2.3)	105.48	1,763.24	3,150.06 (2.1)	204.68	3,512.46	6,244.94 (2.3)	397.56	7,006.48
	100	3,194.71 (2.6)	208.46	3,508.48	6,232.92 (2.3)	398.65	6,965.58	12,410.82 (2.6)	797.68	13,886.68

Table 4.10: Average execution time of float DE IP (50MHz) in SU configuration with PPC440 and MicroBlaze based SoC (125MHz)

			NP =**8**			NP=**16**			NP=**32**		
Processor	**Function**	G_{MAX}	**SW(ms) NO FPU** [2] **(Std%)**	**SW(ms) FPU** [3] **(Std%)**	**HW(ms) (Std%)**	**SW(ms) NO FPU (Std%)**	**SW(ms) FPU (Std%)**	**HW(ms) (Std%)**	**SW(ms) NO FPU (Std%)**	**SW(ms) FPU (Std%)**	**HW(ms) (Std%)**
		1	25.68 (1.9)	1.43 (1.5)	0.03 (0.8)	47.85 (1.8)	2.6 (1.5)	0.05 (0.9)	94.68 (1.6)	5.34 (1.4)	0.09 (0.7)
	Fun2	50	800.54 (1.5)	50.42 (0.7)	0.87 (0.6)	1,539.67 (1.9)	96.75 (1.4)	1.57 (0.6)	2,999.25 (1.5)	189.58 (1.2)	3.04 (0.8)
		100	1,646.86 (1.6)	100.95 (0.8)	1.70 (0.7)	3,136.74 (1.7)	192.47 (1.1)	3.12 (0.7)	6,134.56 (1.4)	377.42 (1.0)	6.08 (0.5)
		1	33.36 (1.1)	2.13 (0.6)	0.05 (0.5)	64.16 (1.3)	4.04 (0.9)	0.09 (0.3)	125.87 (1.1)	7.85 (0.6)	0.18 (0.6)
PPC440	**Fun4**	50	1,026.52 (0.9)	72.11 (0.7)	1.50 (0.6)	1,928.86 (1.6)	138.52 (1.1)	2.83 (0.5)	3,793.64 (0.9)	270.82 (0.5)	5.55 (0.4)
		100	2,050.82 (0.7)	144.74 (0.6)	2.70 (0.4)	3,901.74 (1.5)	274.83 (0.8)	5.64 (0.4)	7,635.51 (1.2)	538.94 (0.8)	11.04 (0.7)
		1	267.35 (1.2)	28.67 (0.9)	0.55 (0.7)	533.69 (1.6)	58.14 (0.8)	1.09 (0.7)	1068.76 (1.5)	116.25 (1.1)	2.17 (0.3)
	Fun6	50	7,109.82 (1.4)	828.78 (0.8)	14.64 (0.4)	14,136.76 (1.7)	1,647.18 (1.1)	29.17 (0.5)	28,122.65 (1.3)	3,269.43 (1.0)	58.17 (0.4)
		100	14,134.92 (1.1)	1,645.26 (1.0)	29.05 (0.9)	28,054.82 (1.8)	3,273.28 (1.2)	57.82 (0.8)	56,457.82 (1.1)	6,503.38 (0.8)	115.41 (0.2)
		1	143.82 (1.9)	140.72 (1.2)	0.03 (0.5)	273.51 (1.9)	267.83 (1.2)	0.05 (0.6)	532.78 (1.6)	530.24 (1.1)	0.09 (0.5)
	Fun2	50	4,425.68 (2.0)	4,310.74 (1.3)	0.87 (0.7)	8,568.42 (2.2)	8,331.16 (1.1)	1.57 (0.7)	16,622.53 (1.8)	16,270.18 (1.2)	3.04 (0.7)
		100	9,082.48 (2.1)	8,855.47 (1.1)	1.70 (0.6)	17,659.71 (2.3)	17,259.63 (1.8)	3.12 (0.8)	33,243.15 (2.0)	31,982.95 (0.8)	6.08 (0.4)
		1	165.82 (1.2)	165.82 (0.9)	0.05 (0.8)	312.86 (2.0)	312.86 (0.9)	0.09 (0.9)	627.83 (1.5)	627.83 (0.9)	0.18 (0.3)
MB	**Fun4**	50	5,029.42 (1.1)	5,029.42 (0.8)	1.50 (0.5)	9,531.85 (2.1)	9,531.85 (0.7)	2.83 (0.6)	18,656.74 (2.0)	18,656.74 (1.3)	5.55 (0.8)
		100	10,036.73 (1.3)	10,036.73 (0.9)	2.70 (0.6)	19,083.46 (1.9)	19,083.46 (0.9)	5.64 (0.7)	35,876.97 (2.1)	35,876.97 (1.5)	11.04 (0.7)
		1	1,256.38 (1.4)	1,253.43 (0.6)	0.55 (0.9)	2,519.58 (2.2)	2,513.82 (1.3)	1.09 (0.6)	5,020.64 (2.2)	5,010.19 (1.8)	2.17 (0.7)
	Fun6	50	33,162.74 (1.5)	33,023.53 (0.8)	14.64 (0.7)	67,027.57 (2.3)	67,099.49 (1.1)	29.17 (0.9)	134,376.12 (1.9)	134,370.53 (0.9)	58.17 (0.5)
		100	65,703.08 (1.7)	65,348.99 (0.9)	29.05 (0.5)	132,329.36 (2.1)	132,826.78 (1.5)	57.82 (0.7)	265,760.35 (1.8)	265,243.14 (1.2)	115.41 (0.6)

Table 4.11: Acceleration factor of float DE IP (50 MHz) in SU configuration with PPC440 and MicroBlaze based SoC (125MHz)

Function	G_{MAX}	NP =8				NP=16				NP=32			
		PPC NO FPU	PPC FPU	MB NO FPU	MB FPU	PPC NO FPU	PPC FPU	MB NO FPU	MB FPU	PPC NO FPU	PPC FPU	MB NO FPU	MB FPU
Fun2	1	856.01	47.66	4794.00	4690.66	957.34	52.34	5470.21	5356.62	1052.33	59.33	5919.77	5891.55
	50	920.16	57.95	5086.98	4954.87	980.68	61.62	5457.59	5306.47	986.59	62.36	5467.93	5352.03
	100	968.74	59.38	5342.63	5209.10	1005.3	61.68	5660.16	5531.93	1008.9	62.07	5467.62	5260.35
Fun4	1	667.2	42.6	3316.40	3316.42	712.88	44.88	3476.22	3476.22	699.27	43.61	3487.94	3487.94
	50	684.34	48.07	3352.94	3352.94	681.57	48.94	3368.14	3368.14	683.53	48.79	3361.57	3361.57
	100	759.56	53.60	3717.30	3717.30	691.79	48.72	3383.59	3383.59	691.62	48.81	3249.72	3249.72
Fun6	1	486.09	52.12	2284.32	2278.96	489.62	53.33	2311.54	2306.25	492.51	53.57	2313.65	2308.84
	50	485.64	56.61	2265.21	2255.70	484.63	56.46	2297.82	2300.29	483.45	56.20	2310.05	2309.96
	100	486.57	56.63	2261.72	2249.53	485.20	56.61	2288.64	2297.24	489.19	56.35	2302.74	2298.26

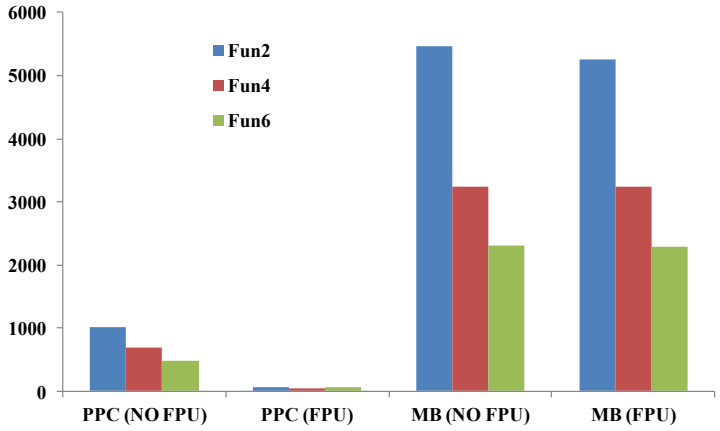


Figure 4.17: Comparison floating point DE in SU configuration on MB, PPC440 based SoC by enable/disable FPU (G_{MAX}=100, NP=32)

in Table4.9. The average execution time for evaluating functions of different complexities with different values of NP and G_{MAX} is tabulated for 20 independent runs. For performance evaluation, the DE floating point IP is executed with different population size (8, 16 and 32) and for different iterations (1, 50 and 100) for three functions (Fun2, Fun4, Fun6). During this implementation the system frequency is fixed at 125 MHz and float DE IP at 50 MHz. From the Table.4.9 it is observed that X86 processor takes less computational time as compared to the embedded processor (PPC440, MB), which is quite obvious because of the difference in their operating frequency. It is also observed that, although PPC440 and MB operates at same operating frequency, PPC440 takes significantly less execution time.

The execution times of PPC440 and Microblaze embedded processor and floating point DE (SU configuration) are evaluated and compared in Table 4.10. In

this table $NOFPU$ and FPU refers that floating point arithmetics are executed using soft FPU library and hard FPU respectively. SW refers that the algorithm is executed in processor (processor execution time) and HW refers to accelerator execution time. The acceleration factor of the DE IP is evaluated as the ratio between software execution time and hardware execution time. This is tabulated in Table 4.11. The tabulated result, reveals that FPU enabled PPC440 based SoC improves execution speed by 8-18x compared to FPU disabled PPC440 based SoC, whereas there is no significant increase of execution speed in case of Microblaze processor based SoC. This trend remains valid for different population size and iterations. These results conclude that for executing DE algorithm in software, FPU enabled PPC440 processor gives superior performance compared to other configurations, however it gives less acceleration factor. This is because, PPC440 processor takes less execution time for executing the software DE algorithm. On the other hand Microblaze processor gives high acceleration factor because it takes more execution time for executing the DE algorithm. This is illustrated in Figure.4.17.The hardware acceleration of float and fixed DE IP using both APU and SU interfaces are evaluated and compared. For performance comparison, the PPC440 and DE IP frequency are set to 200MHz and 33MHz respectively. Initially floating and fixed point software DE algorithm are ported into PPC440 processor. Then the float DE IP is interfaced using APU and SU interfaces. The acceleration achieved for both the interfaces of float DE IP are tabulated in Table.4.12 and Table.4.13 respectively. Similarly the fixed DE IP is interfaced using APU and SU interfaces and the acceleration factors are tabulated in Table.4.14 and Table.4.15 respectively. From the results it is observed both accelerators (SU and APU) of fixed or float DE IP results same acceleration factor. However when we compare SU/APU of float DE IP with SU/APU fixed DE IP, for lower dimension functions fixed DE IP shows higher acceleration as compared to float DE IP. This is shown in Figure.4.18a, whereas for higher dimension functions float DE IP outperforms fixed DE IP as shown in Figure.4.18b (Fun6).

[1]Standard deviation
[2]Soft FPU
[3]Hard FPU

Table 4.12: Average execution time of float DE IP (33MHz) in APU configuration with PPC440 based SoC (200MHz)

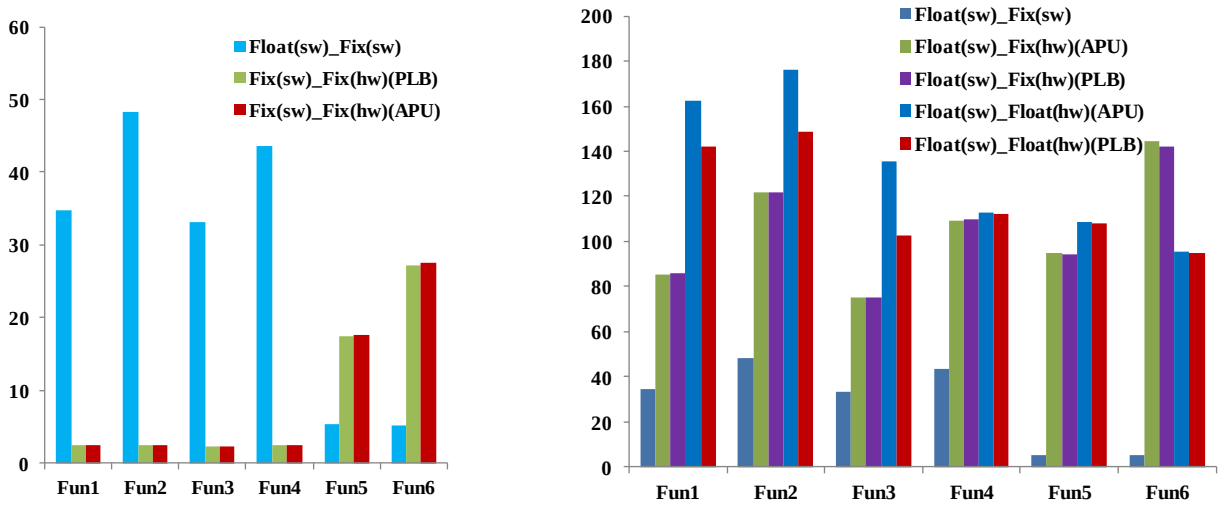
		NP =8			**NP=16**			**NP=32**		
Test Function	G_{MAX}	**Float SW(ms) (Std%)**	**APU HW(ms) (Std%)**	**Acceleration factor**	**Float SW(ms) (Std%)**	**APU HW(ms) (Std%)**	**Acceleration factor**	**Float SW(ms) (Std%)**	**APU HW(ms) (Std%)**	**Acceleration factor**
	1	4.91	0.05	90.93	9.44	0.09	107.27	18.36	0.16	116.20
		(2.8)	(1.1)		(2.2)	(1.4)		(1.2)	(1.0)	
Fun1	50	181.05	2.05	88.36	332.37	3.43	96.87	641.81	6.18	103.87
		(0.9)	(1.2)		(0.4)	(0.7)		(0.2)	(0.2)	
	100	363.17	4.13	87.98	673.21	5.11	131.69	1,301.32	8.00	162.71
		(0.5)	(1.0)		(0.4)	(1.5)		(0.2)	(0.2)	
	1	8.01	0.06	145.64	15.02	0.09	166.89	28.73	0.17	170.00
		(2.2)	(1.5)		(2.3)	(1.3)		(1.2)	(1.4)	
Fun2	50	264.53	1.51	175.77	491.39	3.31	148.41	940.12	5.37	175.23
		(0.9)	(1.2)		(0.4)	(0.5)		(0.2)	(0.3)	
	100	536.05	3.12	171.70	994.24	6.71	148.15	1,897.45	10.77	176.26
		(0.5)	(0.7)		(0.3)	(0.5)		(0.2)	(0.2)	
	1	5.12	0.06	81.27	10.13	0.11	91.26	19.88	0.21	96.98
		(2.7)	(1.1)		(1.3)	(1.8)		(0.6)	(1.3)	
Fun3	50	199.54	2.43	82.15	371.94	4.14	89.82	720.03	7.58	95.02
		(0.6)	(1.1)		(0.3)	(0.5)		(0.2)	(0.3)	
	100	397.91	4.89	81.44	740.14	6.02	122.93	1,432.64	10.57	135.54
		(0.5)	(1.1)		(0.3)	(0.6)		(0.2)	(0.2)	
	1	9.99	0.09	113.52	19.36	0.16	122.53	38.38	0.31	125.84
		(2.2)	(1.6)		(1.2)	(1.9)		(0.6)	(0.9)	
Fun4	50	305.79	3.08	99.35	584.59	5.46	107.11	1,145.25	10.29	111.28
		(0.4)	(1.2)		(0.2)	(0.7)		(0.2)	(0.3)	
	100	612.92	6.13	100.02	1,178.46	10.86	108.50	2,304.13	20.44	112.73
		(0.3)	(1.2)		(0.2)	(0.7)		(0.2)	(0.6)	
	1	41	0.34	119.53	81	0.67	121.80	162	1.32	122.91
		(1.6)	(0.8)		(1.5)	(0.3)		(1.5)	(0.2)	
Fun5	50	1,132	10.83	104.53	2,234	20.75	107.67	4,439	40.77	108.89
		(1.7)	(0.2)		(1.6)	(0.1)		(1.4)	(0.1)	
	100	2,254	21.64	104.17	4,435	41.22	107.60	8,809	80.99	108.77
		(0.9)	(0.1)		(2.1)	(0.1)		(2.1)	(0.1)	
	1	85	0.36	239.44	170	0.92	184.58	339	3.54	95.74
		(1.9)	(0.6)		(2.3)	(0.3)		(2.2)	(0.2)	
Fun6	50	2,251	11.16	201.79	4,472	27.55	162.34	8,916	97.61	91.34
		(1.1)	(0.4)		(1.7)	(0.1)		(2.1)	(0.1)	
	100	4,476	22.21	201.57	8,745	54.86	159.41	18,483	193.83	95.36
		(2.1)	(0.5)		(1.1)	(0.1)		(1.1)	(0.1)	

The acceleration factor of fixed DE IP with fixed point software algorithm is also compared and found that the acceleration is 2-3x for lower complex/dimension functions and 25-27x for higher dimension functions. These results are valid for both SU and APU of fixed DE as shown in Figure.4.18. These tables also reveal that the fixed/ float accelerator enhances the execution speed of the algorithm significantly for higher dimensional complex functions. For analysis purpose, the acceleration factor is calculated for both the designed accelerators (SU and APU)

Table 4.13: Average execution time of float DE IP (33MHz) in SU configuration with PPC440 based SoC (200MHz) [7]

		NP =8			NP=16			NP=32		
Test Function	G_{MAX}	**Float SW(ms) (Std%)**	**SU HW(ms) (Std%)**	**Acceleration factor**	**Float SW(ms) (Std%)**	**SU HW(ms) (Std%)**	**Acceleration factor**	**Float SW(ms) (Std%)**	**SU HW(ms) (Std%)**	**Acceleration factor**
	1	4.91	0.05	94.42	9.44	0.09	109.77	18.36	0.15	120.79
		(2.8)	(1.1)		(2.2)	(1.4)		(1.2)	(1.0)	
Fun1	50	181.05	2.09	86.50	332.37	3.47	95.76	641.81	6.23	103.10
		(0.9)	(1.2)		(0.4)	(0.7)		(0.2)	(0.2)	
	100	363.17	4.22	86.14	673.21	6.92	97.26	1,301.32	9.17	141.85
		(0.5)	(1.0)		(0.4)	(1.5)		(0.2)	(0.2)	
	1	8.01	0.05	151.13	15.02	0.09	166.89	28.73	0.17	174.12
		(2.2)	(1.5)		(2.3)	(1.3)		(1.2)	(1.4)	
Fun2	50	264.53	2.13	124.37	491.39	3.57	137.64	940.12	6.45	145.78
		(0.9)	(1.2)		(0.4)	(0.5)		(0.2)	(0.3)	
	100	536.05	4.22	127.03	994.24	7.08	140.39	1,897.45	12.78	148.48
		(0.5)	(0.7)		(0.3)	(0.5)		(0.2)	(0.2)	
	1	5.12	0.06	80.00	10.13	0.11	90.45	19.88	0.21	96.98
		(2.7)	(1.1)		(1.3)	(1.8)		(0.6)	(1.3)	
Fun3	50	199.54	2.50	79.91	371.94	4.20	88.60	720.03	7.65	94.12
		(0.6)	(1.1)		(0.3)	(0.5)		(0.2)	(0.3)	
	100	397.91	4.95	80.40	740.14	8.37	88.43	1,432.64	13.97	102.55
		(0.5)	(1.1)		(0.3)	(0.6)		(0.2)	(0.2)	
	1	9.99	0.09	114.83	19.36	0.16	121.76	38.38	0.31	125.84
		(2.2)	(1.6)		(1.2)	(1.9)		(0.6)	(0.9)	
Fun4	50	305.79	3.12	97.92	584.59	5.50	106.35	1,145.25	10.32	111.03
		(0.4)	(1.2)		(0.2)	(0.7)		(0.2)	(0.3)	
	100	612.92	6.17	99.37	1,178.46	10.92	107.93	2,304.13	20.49	112.45
		(0.3)	(1.2)		(0.2)	(0.7)		(0.2)	(0.6)	
	1	41	0.34	119.53	81	0.67	121.80	162	1.32	122.54
		(1.6)	(0.8)		(1.5)	(0.3)		(1.5)	(0.2)	
Fun5	50	1,132	10.76	105.17	2,234	20.65	108.20	4,439	41.07	108.08
		(1.7)	(0.2)		(1.6)	(0.1)		(1.4)	(0.1)	
	100	2,254	21.38	105.45	4,435	41.43	107.05	8,809	81.63	107.92
		(0.9)	(0.1)		(2.1)	(0.1)		(2.1)	(0.1)	
	1	85	0.36	237.43	170	0.92	183.98	339	3.55	95.39
		(1.9)	(0.6)		(2.3)	(0.3)		(2.2)	(0.2)	
Fun6	50	2,251	11.16	201.74	4,472	27.75	161.16	8,916	98.03	90.96
		(1.1)	(0.4)		(1.7)	(0.1)		(2.1)	(0.1)	
	100	4,476	22.56	198.44	8,745	57.88	151.09	18,483	194.61	94.97
		(2.1)	(0.5)		(1.1)	(0.1)		(1.1)	(0.1)	

of float DE for different dimensions of Fun5 and Fun6 functions. The software and hardware execution time of these two functions are evaluated for dimensions 8, 16, and 32. The results are tabulated in Table.4.17. From this table, it is clear that for Fun6 function, the acceleration factor increases with the dimension. However, the same is not true for Fun5 because of the simplicity of the function. So it can be concluded that the acceleration factor increases with the complexity of the function to be optimized. The same trend of acceleration is also noticed in case of the SU based accelerator and fixed DE accelerator, so it is not tabulated.



(a) Float DE vs. all accelerators

(b) Fixed DE vs. fixed accelerator

Figure 4.18: Comparison of acceleration factors of fixed and float DE IP in SU and APU configurations (G_{MAX}=100 and NP=32)

4.8.4 SoC Resource and Power results

The SoC resource results of fixed and float DE IP for both SU and APU interfaces for Fun4 are tabulated in Table.4.17 and Table.4.18 respectively. It is observed that the resource utilization is independent of the interface, whereas the SoC with floating point core consumes more resources compared to fixed point IP as expected. We further carried out power analysis of the developed SoC system for both the interfaces of both DE IPs. The power analysis of floating point DE IP with APU and SU interfaces are tabulated in Table.4.19 and Table.4.20 respectively, whereas for fixed point DE IP, it is tabulated in Table.4.21 and Table.4.22 respectively. These tables present details of power consumed by different modules of SoCs. The total APU/SU power is the sum of power consumed by DE IP, FIFO/IPIF, RAM, and remaining logic. System power refer to the sum of power consumed due to peripheral IPs of the design. The power analysis due to the resources of both float and fixed DE IP SoC systems are also tabulated in Table.4.23 and Table.4.24 respectively. From these tables, it can be concluded that power consumption is independent of interface type. The float DE SoC consumes less power while optimizing low dimension functions (Fun4) as shown in Figure.4.19a, whereas fixed DE SoC consumes lesser power for optimizing complex functions

Table 4.14: Average execution time of fixed DE IP (33MHz) in APU configuration with PPC440 based SoC (200MHz)

		NP =8			**NP=16**			**NP=32**		
Test Function	G_{MAX}	**Fixed SW(ms) (Std%)**	**APU HW(ms) (Std%)**	**Acceleration factor**	**Fixed SW(ms) (Std%)**	**APU HW(ms) (Std%)**	**Acceleration factor**	**Fixed SW(ms) (Std%)**	**APU HW(ms) (Std%)**	**Acceleration factor**
	1	0.15	0.05	3.00	0.26	0.1	2.60	0.52	0.2	2.60
		(2.8)	(1.1)		(2.2)	(1.4)		(1.2)	(1.0)	
Fun1	50	5.38	2.13	2.53	9.69	3.9	2.48	18.82	7.6	2.48
		(0.9)	(1.2)		(0.4)	(0.7)		(0.2)	(0.2)	
	100	10.38	4.27	2.43	19.33	8.21	3.94	37.51	15.2	2.47
		(0.5)	(1.0)		(0.4)	(1.5)		(0.2)	(0.2)	
	1	0.18	0.05	3.60	0.31	0.1	3.10	0.61	0.2	3.05
		(2.2)	(1.5)		(2.3)	(1.3)		(1.2)	(1.4)	
Fun2	50	5.97	2.23	2.68	10.85	4.06	2.67	19.82	7.8	2.54
		(0.9)	(1.2)		(0.4)	(0.5)		(0.2)	(0.3)	
	100	11.96	4.43	2.70	21.64	8.11	2.67	39.26	15.6	2.52
		(0.5)	(0.7)		(0.3)	(0.5)		(0.2)	(0.2)	
	1	0.16	0.07	2.29	0.31	0.13	2.38	0.61	0.26	2.35
		(2.7)	(1.1)		(1.3)	(1.8)		(0.6)	(1.3)	
Fun3	50	5.83	2.6	2.24	11.08	4.9	2.26	21.64	9.57	2.26
		(0.6)	(1.1)		(0.3)	(0.5)		(0.2)	(0.3)	
	100	11.62	5.3	2.19	22.08	9.9	2.23	43.16	19.06	2.26
		(0.5)	(1.1)		(0.3)	(0.6)		(0.2)	(0.2)	
	1	0.23	0.08	2.88	0.45	0.15	3.00	0.84	0.3	2.80
		(2.2)	(1.6)		(1.2)	(1.9)		(0.6)	(0.9)	
Fun4	50	7.08	2.9	2.44	13.48	5.4	2.50	26.46	10.5	2.52
		(0.4)	(1.2)		(0.2)	(0.7)		(0.2)	(0.3)	
	100	14.11	5.8	2.43	26.94	10.9	2.47	52.77	21.1	2.50
		(0.3)	(1.2)		(0.2)	(0.7)		(0.2)	(0.6)	
	1	6	0.5	12.00	11	0.8	13.75	23	1.6	14.38
		(1.6)	(0.8)		(1.5)	(0.3)		(1.5)	(0.2)	
Fun5	50	207	11.9	17.39	411	23.5	17.49	809	46.6	17.36
		(1.7)	(0.2)		(1.6)	(0.1)		(1.4)	(0.1)	
	100	412	23.7	17.38	825	46.7	17.67	1,638	92.6	17.69
		(0.9)	(0.1)		(2.1)	(0.1)		(2.1)	(0.1)	
	1	15	0.6	25.00	30	1.2	25.00	62	2.3	26.96
		(1.9)	(0.6)		(2.3)	(0.3)		(2.2)	(0.2)	
Fun6	50	446	16.4	27.20	884	32.4	27.28	1,736	64.5	26.91
		(1.1)	(0.4)		(1.7)	(0.1)		(2.1)	(0.1)	
	100	891	32.6	27.33	1,764	64.4	27.39	3,537	128	27.63
		(2.1)	(0.5)		(1.1)	(0.1)		(1.1)	(0.1)	

(Fun6) as shown in Figure.4.19b. Finally, the power analysis of complete SoC system with float/fixed DE IPs for optimizing Fun6 is tabulated in Table.4.25.

.

Table 4.15: Average execution time of fixed DE IP (33MHz) in SU configuration with PPC440 based SoC (200MHz)

		NP =8			NP=16			NP=32		
Test Function	G_{MAX}	**Fixed SW(ms) (Std%)**	**SU HW(ms) (Std%)**	**Acceleration factor**	**Fixed SW(ms) (Std%)**	**SU HW(ms) (Std%)**	**Acceleration factor**	**Fixed SW(ms) (Std%)**	**SU HW(ms) (Std%)**	**Acceleration factor**
	1	0.15	0.06	2.68	0.26	0.10	2.65	0.52	0.19	2.68
		(2.8)	(1.1)		(2.2)	(1.4)		(1.2)	(1.0)	
Fun1	50	5.38	2.16	2.50	9.69	3.96	2.45	18.82	7.64	2.46
		(0.9)	(1.2)		(0.4)	(0.7)		(0.2)	(0.2)	
	100	10.38	4.36	2.38	19.33	8.02	2.41	37.51	15.19	2.47
		(0.5)	(1.0)		(0.4)	(1.5)		(0.2)	(0.2)	
	1	0.18	0.06	3.10	0.31	0.11	2.90	0.61	0.21	2.90
		(2.2)	(1.5)		(2.3)	(1.3)		(1.2)	(1.4)	
Fun2	50	5.97	2.18	2.73	10.85	4.10	2.65	19.82	7.82	2.54
		(0.9)	(1.2)		(0.4)	(0.5)		(0.2)	(0.3)	
	100	11.96	4.39	2.72	21.64	8.12	2.67	39.26	15.56	2.52
		(0.5)	(0.7)		(0.3)	(0.5)		(0.2)	(0.2)	
	1	0.16	0.07	2.32	0.31	0.13	2.31	0.61	0.27	2.28
		(2.7)	(1.1)		(1.3)	(1.8)		(0.6)	(1.3)	
Fun3	50	5.83	2.62	2.23	11.08	4.92	2.25	21.64	9.55	2.27
		(0.6)	(1.1)		(0.3)	(0.5)		(0.2)	(0.3)	
	100	11.62	5.26	2.21	22.08	9.85	2.24	43.16	19.07	2.26
		(0.5)	(1.1)		(0.3)	(0.6)		(0.2)	(0.2)	
	1	0.23	0.08	2.88	0.45	0.16	2.90	0.84	0.30	2.76
		(2.2)	(1.6)		(1.2)	(1.9)		(0.6)	(0.9)	
Fun4	50	7.08	2.93	2.42	13.48	5.37	2.51	26.46	10.48	2.52
		(0.4)	(1.2)		(0.2)	(0.7)		(0.2)	(0.3)	
	100	14.11	5.89	2.40	26.94	10.77	2.50	52.77	20.97	2.52
		(0.3)	(1.2)		(0.2)	(0.7)		(0.2)	(0.6)	
	1	6	0.41	14.74	11	0.81	13.53	23	1.64	14.03
		(1.6)	(0.8)		(1.5)	(0.3)		(1.5)	(0.2)	
Fun5	50	207	12.06	17.17	411	23.74	17.31	809	47.12	17.17
		(1.7)	(0.2)		(1.6)	(0.1)		(1.4)	(0.1)	
	100	412	23.98	17.18	825	47.16	17.49	1,638	93.58	17.50
		(0.9)	(0.1)		(2.1)	(0.1)		(2.1)	(0.1)	
	1	15	0.59	25.42	30	1.18	25.40	62	2.37	26.19
		(1.9)	(0.6)		(2.3)	(0.3)		(2.2)	(0.2)	
Fun6	50	446	16.66	26.78	884	32.98	26.80	1,736	65.57	26.48
		(1.1)	(0.4)		(1.7)	(0.1)		(2.1)	(0.1)	
	100	891	33.07	26.94	1,764	65.73	26.84	3,537	130.07	27.19
		(2.1)	(0.5)		(1.1)	(0.1)		(1.1)	(0.1)	

Table 4.16: Timing results for different dimensions in APU configuration of fixed DE IP (NP=32) with PPC440 based SoC

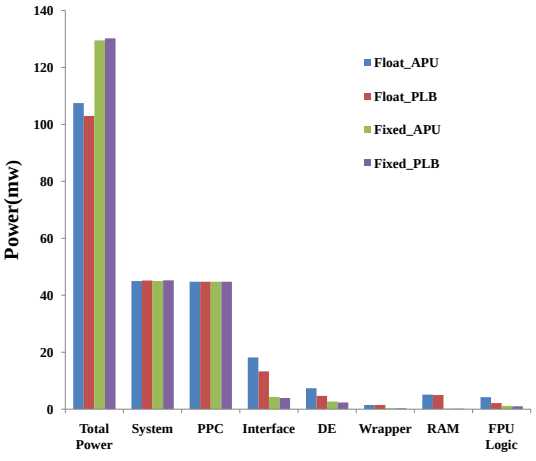
		D =8			D=16			D=32		
Test Function	G_{MAX}	**SW(ms) (Fixed)**	**HW(ms)**	**Acceleration factor**	**SW(ms) (Fixed)**	**HW(ms)**	**Acceleration factor**	**SW(ms) (Fixed)**	**HW(ms)**	**Acceleration factor**
	500	2711	160	16.94	4781	258	18.53	8612	464	18.56
Fun5	1000	5502	320	17.19	9811	516	19.01	18182	929	19.57
	2000	11086	642	17.26	19872	1031	19.27	36654	1857	19.73
	500	3199	171	18.70	6942	309	22.46	17500	655	26.71
Fun6	1000	6477	346	18.71	14215	627	22.67	35526	1321	26.89
	2000	13637	695	19.62	29200	1263	23.12	71683	2651	27.04

Table 4.17: SoC Device Utilization of floating point DE IP for Fun4

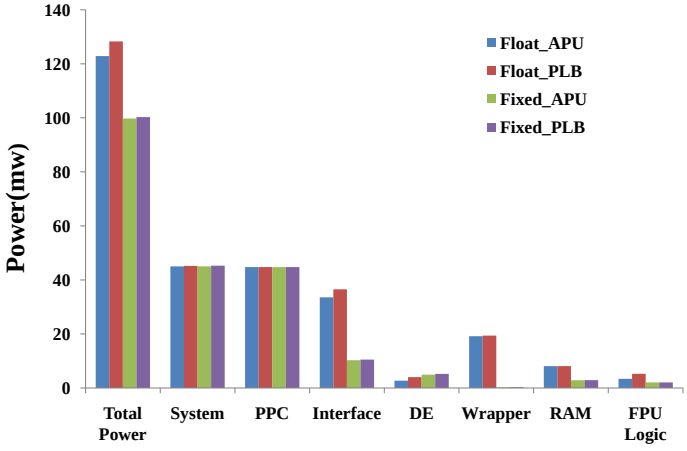
	APU		SU	
Resource Type	**Used**	**Utilization**	**Used**	**Utilization**
PLL_ADVs	1	16%	1	16%
DSP48Es	20	15%	20	15%
JTAGPPCs	1	100%	1	100%
PPC440s	1	100%	1	100%
BlockRAM	26	17%	26	17%
Slices	7190	64%	7230	64%
Slice LUTs	18921	42%	18954	42%
Slice Registers	12097	27%	13433	28%
LUT FF Pairs	8953	40%	9203	41%

Table 4.18: SoC Device utilization of fixed DE IP for Fun4

	APU		SU	
Resource Type	**Used**	**Utilization**	**Used**	**Utilization**
PLL_ADVs	1	16%	1	16%
DSP48Es	61	47%	61	47%
JTAGPPCs	1	100%	1	100%
PowePC440s	1	100%	1	100%
BlockRAM	9	6%	9	6%
Slices	3431	30%	3326	29%
Slice LUTs	7259	16%	7272	16%
Slice Registers	6270	13%	6336	14%
LUT FF Pairs	3615	36%	3829	39%



(a) Power results of Fun4



(b) Power results of Fun6

Figure 4.19: Power results of all the accelerators for Fun4 and Fun6

Table 4.19: Power analysis of floating DE APU accelerator in SoC (mW)

Test Function	Total Power	System	PPC	APU	DE	FIFO	RAM	FPU Logic
Fun1	107.47	44.99	44.74	18.17	7.35	1.47	5.11	4.21
Fun2	114.55	44.99	44.74	23.59	12.57	1.47	5.10	5.17
Fun3	100.17	44.99	44.74	10.77	2.41	1.47	5.10	5.17
Fun4	100.57	44.99	44.74	11.38	2.69	1.47	4.98	2.23
Fun5	123.29	45.00	44.74	38.35	2.72	19.17	8.1	3.58
Fun6	122.85	45.00	44.74	33.55	2.72	19.17	8.1	3.41

Table 4.20: Power analysis of floating DE SU accelerator in SoC (mW)

Test Function	Total Power	System	PPC	SA	DE	IPIF	RAM	FPU Logic
Fun1	102.96	45.18	44.74	13.26	4.65	1.47	4.98	2.15
Fun2	107.62	45.21	44.74	17.76	6.98	1.47	4.18	4.20
Fun3	104.18	45.17	44.74	14.65	4.05	1.47	4.98	4.12
Fun4	104.06	45.16	44.74	14.36	5.22	1.47	4.98	3.50
Fun5	127.41	45.19	44.74	37.93	4.88	19.38	8.10	5.52
Fun6	128.27	45.19	44.74	36.54	4.02	19.38	8.10	5.25

Table 4.21: Power analysis of fixed DE APU accelerator in SoC (mW)

Test Function	Total Power	System	PPC	APU	DE	FIFO	RAM	Logic
Fun1	129.50	45.00	44.74	4.30	2.69	0.30	0.18	1.13
Fun2	106.34	45.01	44.74	17.07	10.80	0.35	0.32	5.60
Fun3	94.91	45.00	44.74	5.65	3.10	0.31	0.38	1.84
Fun4	96.06	45.00	44.74	6.72	3.82	0.34	0.33	2.57
Fun5	99.37	45.00	44.74	10.15	4.91	0.32	2.93	1.98
Fun6	99.76	45.00	44.74	10.27	4.92	0.31	2.87	2.05

Table 4.22: Power analysis of fixed DE SU accelerator in SoC (mW)

Test Function	Total Power	System	PPC	SA	DE	IPIF	RAM	Logic
Fun1	130.21	45.23	44.74	3.93	2.36	0.31	0.19	1.03
Fun2	130.22	45.21	44.74	17.87	12.24	0.27	0.18	5.14
Fun3	94.99	45.21	44.74	5.44	2.87	0.29	0.38	1.87
Fun4	96.06	45.14	44.74	6.66	3.83	0.29	0.33	2.50
Fun5	99.23	45.20	44.74	9.74	4.82	0.29	2.93	1.89
Fun6	100.27	45.28	44.74	10.50	5.21	0.30	2.92	2.05

4.8.5 Convergence results

Although the objective of this work is intended to improve the execution speed and not the quality of the solution, convergence tests are also performed to verify the functionality of the DE algorithm by using two accelerators. The convergence graphs of DE algorithm for Fun2 and Fun3 test functions are plotted in Fig-

Table 4.23: Power analysis of resources in complete SoC using floating DE IP in APU and SU configurations (mW)

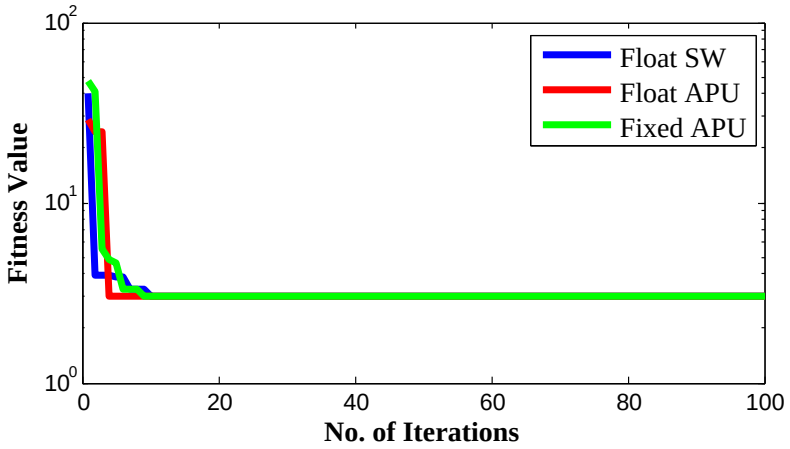
	Fun1		Fun2		Fun3		Fun4		Fun5		Fun6	
Resources	**PLB**	**APU**	**PLB**	**APU**	**PLB**	**APU**	**PLB**	**APU**	**PLB**	**APU**	**PLB**	**APU**
Clocks	121.67	120.31	134.57	129.97	121.55	122.14	133.04	135.06	136.87	142.80	139.83	138.86
Logic	2.09	3.16	2.46	3.78	2.62	2.97	1.201	1.505	2.06	2.18	2.09	2.29
Signals	5.66	8.92	7.51	12.92	6.45	8.63	3.56	3.86	5.27	5.69	5.59	5.13
IOs	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22
BRAMs	11.78	11.95	11.77	11.95	11.76	11.73	11.76	11.76	3.208	3.208	3.208	3.208
DSPs	0.08	0.08	2.55	2.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
PPC440s	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74
PLLs	34.62	34.62	34.62	34.62	34.62	34.62	34.62	34.62	34.62	34.62	34.62	34.62
Total Quiescent Power	1849.34	1850.19	1854.11	1854.67	1849.64	1851.43	1842.44	1852.31	1834.14	1860.64	1840.04	1852.31
Total Dynamic Power	3520.00	3523.15	3537.56	3539.63	3521.11	3527.69	3524.64	3530.94	3459.64	3561.46	3496.18	3530.94
Total Power	5369.34	5373.35	5391.67	5394.31	5370.75	5379.11	5337.12	5383.25	5319.78	5422.10	5539.72	5383.25

Table 4.24: Power analysis of resources in complete SoC using fixed DE IP in APU and SU configurations (mW)

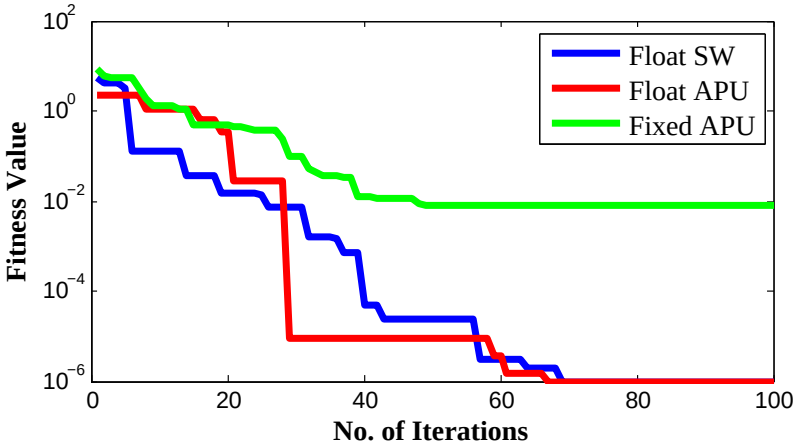
	Fun1		Fun2		Fun3		Fun4		Fun5		Fun6	
Resources	**PLB**	**APU**	**PLB**	**APU**	**PLB**	**APU**	**PLB**	**APU**	**PLB**	**APU**	**PLB**	**APU**
Clocks	107.69	107.55	108.14	100.79	94.79	104.57	96.91	103.20	95.89	94.61	101.04	106.13
Logic	1.72	1.51	1.70	4.41	2.49	2.42	2.35	2.23	2.26	2.19	2.44	2.32
Signals	4.14	3.53	4.17	12.65	4.50	4.50	4.42	4.54	4.93	5.14	5.59	5.19
IOs	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22	3298.22
BRAMs	3.62	3.62	3.62	4.20	4.20	4.20	4.20	4.20	7.35	7.35	7.35	7.35
DSPs	0.83	0.83	0.83	1.74	0.44	0.40	1.74	1.70	1.33	1.33	1.54	1.54
PPC440s	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74	44.74
PLLs	69.24	69.24	69.24	36.12	36.12	34.62	34.62	34.62	34.62	34.62	34.62	34.62
Total Quiescent Power	1937.78	1937.51	1937.90	1841.77	1839.93	1842.54	1840.79	1842.48	1841.37	1841.06	1843.04	1849.27
Total Dynamic Power	3532.47	3531.51	3532.92	3502.51	3485.15	3498.60	3488.35	3494.64	3490.50	3489.35	3496.18	3501.26
Total Power	5470.25	5469.03	5470.83	5349.59	5325.09	5337.40	5329.14	5337.12	5331.86	5330.40	5339.72	5345.53

Table 4.25: Power analysis of SoC system for Fun6 consisting floating/fixed point DE IP

Resource Type	**Float Power(mW)**	**Fixed Power(mW)**
Total SoC system	122.85	96.06
PowerPC440	44.74	44.74
APU	33.55	6.72
Clockgen	38.35	38.35
DDR2	5.40	5.47
Compact flash	0.24	0.25
RS232_Uart_1	0.15	0.12
PLB	0.02	0.02
proc_sys_reset_0	0.05	0.04
xps_timer_0	0.04	0.04
xps_timer_1	0.03	0.03
xps_intc_0	0.02	0.02
xps_bram_cntlr	0.01	0.01
xps_bram	0.00	0.00



(a) Comparison of float and fixed DE result in APU configuration for Fun2



(b) Comparison of float and fixed DE result in APU configuration for Fun3

Figure 4.20: Convergence rate comparison of float and fixed DE in APU configuration with software

ure.4.20a and Figure.4.20b respectively. These figures compare the convergence results of floating point DE algorithm implemented in embedded processor (as SW) with floating and fixed APU hardware accelerator (as HW). It is observed that, the floating point hardware results approximately same quality of solution and convergence rate as the processor whereas, the fixed point APU gives the solution of less quality because of its bit-width limitation.

4.9 Case Study: Infinite Impulse Response (IIR) system identification using DE algorithm

Designing of an adaptive digital IIR filter is an emerging research area for many years. This has been used in different practical applications [130, 131, 132]. However, the IIR filter does not guarantee satisfactory performance if the coefficients are not chosen accurately. It has also been recognized that error surface is usually non-quadratic and multi-modal with respect to filter coefficients. So, to improve the robustness of the design, evolutionary algorithms have been applied, for the design of IIR filter [118, 120, 121, 122, 133]. In these approaches, the filter coefficients are updated using evolutionary techniques which help to avoid local minimas in multimodal error surface. The objective of this case study is to verify the performance of developed DE accelerator by implementing the IIR system identification task in FPGA.

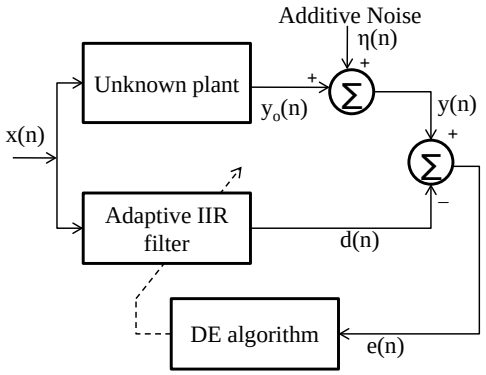


Figure 4.21: Block diagram of DE based IIR system identification

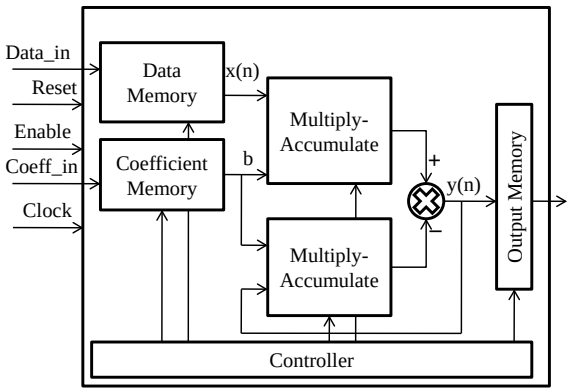


Figure 4.22: Hardware architecture for IIR filter

The block diagram of IIR based system identification using DE algorithm is shown in Figure.4.21. The coefficients of IIR filter are obtained by minimizing the difference between the output of the filter and unknown system using DE algorithm [119]. The system can be modeled using an IIR filter as equation 1.

$$y_0(n) + \sum_{i=1}^{M} b_i y(n-i) = \sum_{i=0}^{L} a_i x(n-i) \tag{4.1}$$

where $x(n)$, $y(n)$ are the input and output signal of the IIR filter respectively, M ($\geq$ L) is the filter order and a_i, b_i are the filter coefficients. The transfer function of adaptive IIR filter is

$$H_M(z) = \left[\frac{\hat{A}(z)}{\hat{B}(z)}\right] \tag{4.2}$$

where $\hat{A}(z)$ and $\hat{B}(z)$ are the feed-forward and feed-back coefficients of adaptive IIR filter respectively.

$$\hat{A}(z) = \sum_{i=0}^{L} \hat{a}_i z^{-i} \tag{4.3}$$

$$\hat{B}(z) = 1 + \sum_{i=1}^{M} \hat{b}_i z^{-i} \tag{4.4}$$

where $\hat{a}_i$ and $\hat{b}_i$ are the estimated feed-forward and feed-back coefficients of the model respectively. If the transfer function of unknown plant is same as the transfer function of adaptive filter, then the plant is identified by using the model $H_M(z)$. This identification task is formulated as an optimization problem with cost function $J(\overset{\wedge}{\phi})$ as

$$J(\overset{\wedge}{\phi}) = \frac{1}{N}\sum_{n=1}^{N} e^2(n) = \frac{1}{N}\sum_{n=1}^{N}(d(n) - y(n)))^2 \tag{4.5}$$

where $d(n)$ and $y(n)$ are desired and actual responses of the plant and N is number of samples; The model coefficient vector $\overset{\wedge}{\phi}$ can be

$$\overset{\wedge}{\phi} = [a_0 a_1 ..... a_L b_1 ..... b_M]^T \tag{4.6}$$

The overall output of the plant is

$$y(n) = y_0(n) + \eta(n) \tag{4.7}$$

where $\eta(n)$ is additive white Gaussian noise and

$$y_0(n) = F^{-1}[H_s(z).X(z)] \tag{4.8}$$

where $H_s(z)$ and $X(z)$ are the Z-transform of unknown plant and input signal respectively.

For verification purpose a 3rd order plant is modeled as a 3rd order IIR filter with transfer function as

$$H_M(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - b_1 z^{-1} - b_2 z^{-2} - b_3 z^{-3}} \tag{4.9}$$

The transfer function of the 3rd order plant is given by [119]

$$H_s(z) = \frac{-0.2 - 0.4z^{-1} + 0.5z^{-2}}{1 - 0.6z^{-1} + 0.25z^{-2} - 0.2z^{-3}} \tag{4.10}$$

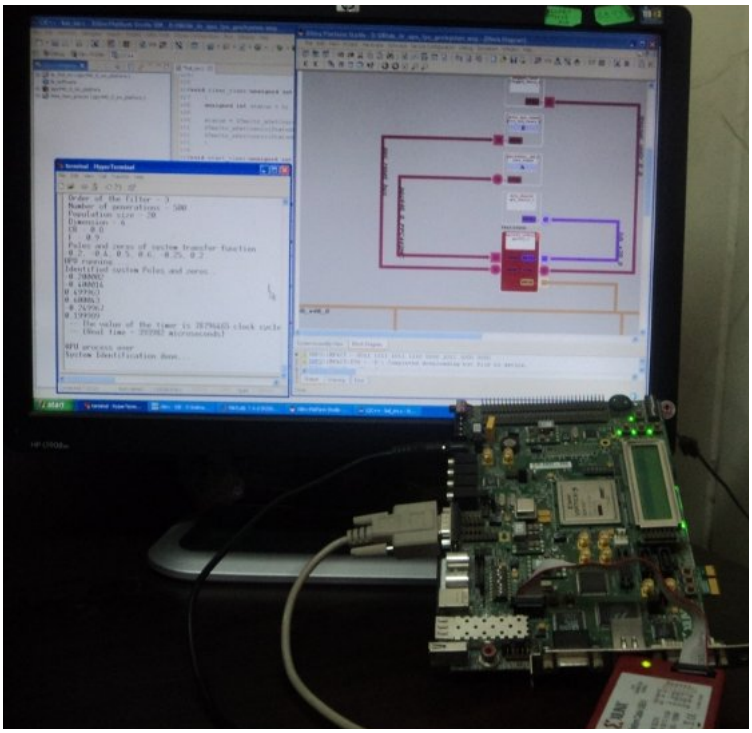


Figure 4.23: Experimental test setup for system identification

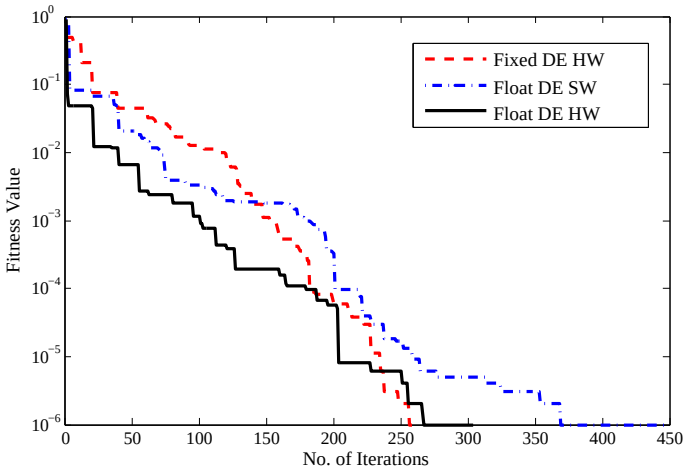


Figure 4.24: Convergence graphs of system identification problem in the HW and SW

Table 4.26: Estimated parameters of 3rd order IIR filter in SoC

Parameter	Actual value	Estimated value
a_0	-0.20	-0.200
a_1	-0.40	-0.400
a_2	0.50	0.499
b_1	0.60	0.600
b_2	-0.25	-0.249
b_3	0.20	0.199

The hardware architecture of IIR based system identification task is given in Figure.4.22. The filter coefficients are obtained from the DE hardware IP. This algorithm is coded in C-language and implemented in the PowerPC processor of Xilinx Virtex-5 FPGA, subsequently it is implemented on hardware and the estimated coefficients using the fixed/float DE IPs using APU are tabulated in Table.4.26, corresponding to 500 iterations. A snapshot of the real time test bed is shown in Figure.4.23. The error convergence graph is shown in Figure.4.24. This figure shows that the quality of the solution using both software on embedded processor and hardware IPs are comparable. Furthermore, the execution time for performing system identification in embedded processor and both hardware

Table 4.27: Timing results for system identification problem using fixed and float DE IP in APU configuration

IP system	G_{MAX}	**HW(ms)**	**SW(ms)**	**Acceleration factor**
Fixed DE	50	0.376	4.201	11.16
	100	0.738	8.525	11.54
	200	1.496	18.041	12.06
	300	2.216	26.551	11.98
	400	2.952	34.371	11.64
	500	3.707	45.681	12.32
Float DE	50	0.370	60.113	162.46
	100	0.733	117.543	160.35
	200	1.459	233.191	159.82
	300	2.185	353.795	161.91
	400	2.911	462.092	158.73
	500	3.637	583.047	160.30

DE IPs for different number of generation is tabulated in Table.4.27. HW (sec) and SW (sec) denotes the execution time using the hardware accelerator and embedded processor in seconds respectively. From this table, it is observed that the developed floating point accelerator gives an acceleration of approximately 150-160x compared to the software implementation, whereas fixed DE IP gives an acceleration of 11x.

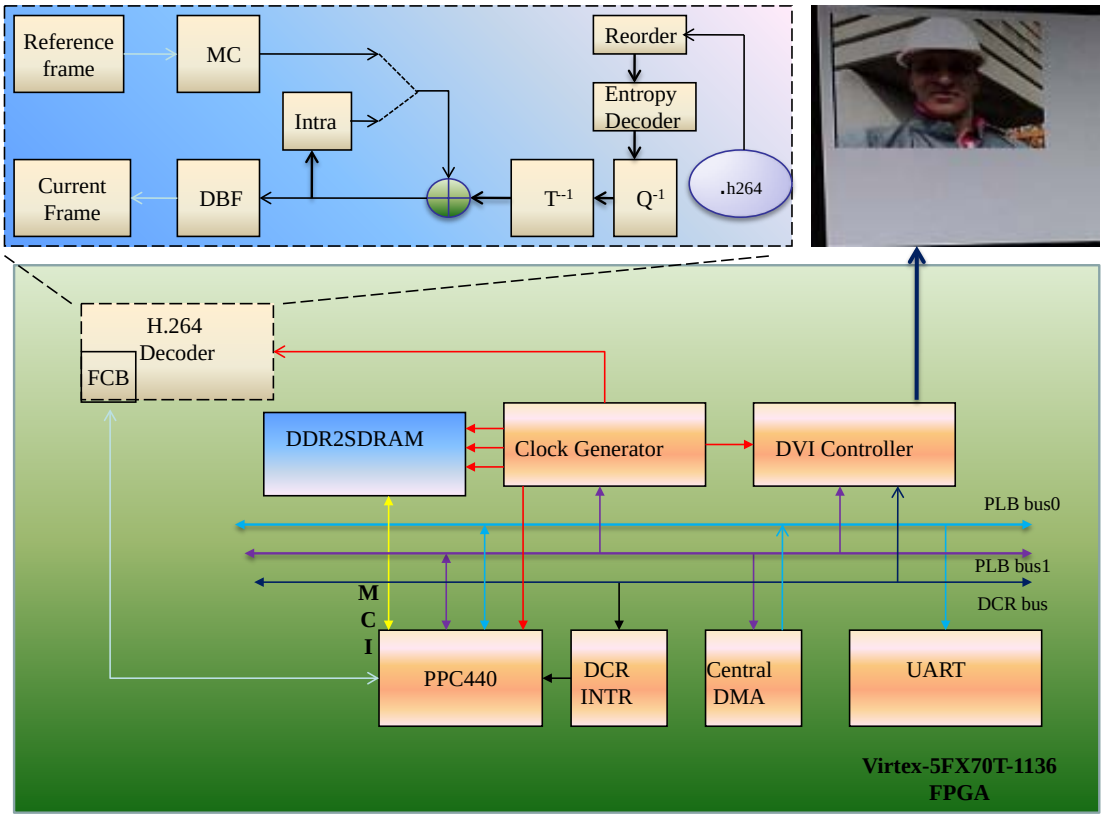
4.10 Conclusions

In this case study, hardware accelerators for computing fixed and floating point DE algorithm are developed. To avoid the bus overhead, the complete DE algorithm is implemented in hardware instead of partitioning the design into software and hardware. These accelerators are interfaced with the PPC440 using both Auxiliary Processing Unit (APU) and Slave Unit (SU) interface techniques. The performance of these interface techniques are studied by optimizing a set of benchmark functions. From the experimental results it is concluded that i) the designed accelerators are giving more acceleration compared to hardware/ software co-design technique, ii) the acceleration factor remains same for optimizing test functions using APU and SU interfaces; However this statistic may change for different applications, iii) The execution time of DE accelerator is compared with

X86, Microblaze, PPC440 processor with different Floating Point Unit (FPU) configurations; It is concluded that hard FPU enabled processor configuration is the optimum configuration for executing the task in the processor. However the hardware accelerator (SU/APU) gives more acceleration compared to the hard/soft FPU enabled processor, iv) it has also been demonstrated that the acceleration factor increases with the increase in complexity and dimension of the fitness function irrespective of interfacing technique, v) the resource utilization and power analysis concluded that the power consumption and resource utilization are same for both interfaces. However, floating point DE IP consumes more power and resources compared to fixed point DE IP, vi) further IIR system identification is solved using the developed fixed & float DE IP and it attained an acceleration of approximately 11x and 150x respectively. In future, this design approach can be used for implementing evolutionary embedded applications.

Chapter 5

Coprocessor for H.264 video decoder



This chapter presents the third case study of the thesis. In this case study a hardware accelerator for H.264 video decoder is developed by using the open source reference hardware design. The accelerator is interfaced as an Auxiliary Processing Unit (APU) with the embedded PPC440 processor in System on Chip (SoC) platform. The complete SoC is validated in Xilinx Virtex-5 development board using different test video sequences and the execution time of accelerator is evaluated.

5.1 Introduction

A video codec is a procedure that allow encoding, that allows encoding and decoding of digital video. Encoding results in compression and decoding results in decompression of digital video. Standardization of video coding formats play a vital role in digital video applications. For playing 30/60 frames per second (fps) video sequences with a resolution of 1920 x 1080 pixels (High Definition Digital Television (HDTV)), with 24 bits per pixel, the required transmission rate is 1.5 billion bits per second (1.5 Gbps) and it would take 112 billion bytes (112 GB) to store a video of 10 minutes duration [5]. Thus compression of video data is required. In general the digital video captured from a camera undergoes certain pre-processing stages before encoded into bit format. This bit format file is either stored or transmitted through the channel. In the receiver section, a decoder decodes the video and displays in the monitor after an optional post-processing steps. Several video coding standards were developed by aggregating the technical algorithms to efficiently compress the video data. H.264 (Advanced Video Coding (AVC) or MPEG-4 part 10) is the latest video coding standard defined by International Telecommunication Union (ITU) [134]. Apart from H.264, different standards such as H.261 (1990), MPEG-1 Video (1993), MPEG-2 Video (1994), H.263 (1995, 1997), MPEG-4 Visual (1998) have been adapted for video coding [135]. Still newer standards such as Scalable Video Coding (SVC) and Multi View Coding (MVC) and latest is High Efficiency Video Coding (HEVC) are evolving for better compression. These standards were developed for different network, mobile broadcast applications.

The applications are ranging from video telephony, Compact Disk (CD), broadcast of Standard Definition Television (SDTV) or High Definition Television (HDTV). The modern applications like video conferencing, mobile television demands new

video coding standard, which can provide enhanced video compression, low latency and streaming of video data [135, 136]. In order to meet these requirements, H.264 video codec is developed. The compression efficiency of H.264/AVC standard is up to two times more compared to MPEG-2 standard. It also can save 25 to 45% and 50 to 70% of bit rates compared with MPEG-4 and MPEG-2 codecs respectively [137, 138]. Although H.264 has better compression efficiency, lower communication channel bandwidth, the main bottleneck is its computational complexity. Due to complexity of the algorithm involved in H.264, the general purpose processors fail to decode the high-resolution video in real time at a rate of 30 fps. This becomes even more critical while decoding video data in the embedded systems, where the constraint is not limited to only speed but also includes power consumption and chip area [139]. Thus, there is a demand for implementing the H.264 decoder in complete hardware which can fulfill the speed, power and area requirements. Hence in this work a hardware accelerator of H.264 decoder is developed.

According to the AVC standard the video coding process is divided into several stages or modules. Each stage can be designed separately in the hardware to optimize speed, area and power consumption. The AVC modules are parsing and entropy decoding, Context Adaptive Variable Length Coding (CAVLC), Inverse Discrete Cosine transform (IDCT) for Inverse Transformation (IT), Inverse Quantization (IQ), Intra-frame Prediction (IP), Motion Compensation (MC) followed by Deblocking Filter (DBF).

In the SoC platform, different hardware interfacing approaches are presented in chapter 2, for accelerating a design. The coprocessor solution gives higher amount of acceleration as compared to other methods, however the acceleration is decreasing with more interactions between the coprocessor and the embedded processor [12]. So in order to limit these interactions a complete H.264 IP with all the modules and internal memories is developed in a single Intellectual Property (IP) core. In this case study we have used the video decoder architecture reported in [140, 141] which is targeted for ASIC platform. As a starting point we have considered the same open-source H.264 video decoder for developing the coprocessor for FPGA platform with modifications listed in section 5.4. The objective of this case study is to develop a coprocessor of entire H.264 decoder and implement in a FPGA based system on chip (SoC) platform. The developed H.264 IP is inter-

faced to the PPC440 embedded processor using APU interface in SoC platform. The developed coprocessor is tested in Xilinx SoC platform with different video sequences and performance of the coprocessor is evaluated by comparing the execution time of the software H.264 decoder implemented in processor and hardware.

In this work the computational complexity of each module is analyzed, by software profiling of the reference video decoder [142] (JM 9.4) targeted to mobile TV application [176 X 144, 30 fps]. Profiling results help to identify the critical module of the design. Since general purpose processors are not suitable for this kind of application, PPC440 embedded processor is used for profiling the H.264 decoder. The xil_profiler an instruction level profiler is used for profiling the software H.264 decoder [143, 144]. The profiling results show that MC, DBF, are computation-intensive modules, which dominates 70% of the total execution time of the decoder [145, 146].

5.2 Related works

The literature reports the hardware development of specific module such as IDCT, DBF, MC to accelerate the H.264 decoder. However limited works are reported about the development of H.264 coprocessor in FPGA. A prototype implementation of H.264 decoder using FPGA is also reported [147]. Agostani et al., presented a main profile decoder of H.264 decoder with HW/SW codesign methodology in Virtex-II Pro FPGA and later they implemented the design in ASIC chip [148]. Different modules of H.264 decoder CAVLC, IDCT, MC, DBF are interfaced to the Processor Local Bus (PLB) in SoC platform and subsequently an ASIC for H.264 decoder is developed [148, 149, 150, 151]. Similarly FPGA based H.264 decoder is also developed for Common Intermediate Format (CIF) and HD resolutions [152]. KeXu et al., proposed an open source (RTL-code) baseline profile decoder of H.264 decoder and implemented in both ASIC and Virtex-4 FPGA for Quarter Common Intermediate Format (QCIF) resolution [153]. This decoder uses self adaptive pipeline for intra/inter modules. This enables the decoder to decode 30 fps at 1.5MHz [140, 141]. Warsaw et al., and Chen.et.al., developed a HDL based H.264 decoder which decodes 30 fps on both FPGA and ASIC technologies [154, 155]. Werda et al., developed a baseline profile decoder on DSP processor TMS-320C6416. This decoder also achieved 30 fps [156]. Huan et al., developed

an embedded platform using ARM CPU core running at 130 MHz, SRAM, multiple dedicated accelerators for (IDCT, CAVLC, MC, DBF), with 32-bit Advanced High Performance Bus (AHB) bus interface and an external memory interface. The decoding process can be partitioned either to execute software on the ARM CPU or on the dedicated hardware running in parallel with CPU (software and hardware). The acceleration due to hardware is 2x time faster than software and even after this speedup, the decoder platform is unable to decode QCIF video sequences more than 10 fps [157]. Starnbeck et al., developed a coprocessor to accelerate the execution speed of H.264 decoder for Digital Video Broadcasting (DVB) [158]. Along with this there are different methodologies such as Bluespec Verilog for developing H.264 decoder [159, 160, 161]. Kthiri et al., developed a SoC system using PPC440 with Xenomai Real-Time Operating System (RTOS) with a coprocessor for DBF [146]. Table.5.1 briefly tabulates some of the related works important to the thesis work.

Table 5.1: Review of existing literature on H.264 decoder

Work	profile	Decoder	Approach	Technology
[148]	Main	H.264 SoC	HW-SW co-design	Virtex-II Pro FPGA & ASIC
[141]	Baseline	H.264 IP	HW	Virtex4 FPGA &ASIC
[156]	Main	H.264 IP	SW	TMS DSP processor
[152, 154]	Main	H.264 IP	HW	FPGA
[155, 162]	Main	H.264 IP	HW	ASIC
[157]	Baseline	H.264 SoC	HW-SW co-design	FPGA-ARM platform
[158]	Main	H.264 IP	HW-SW co-design	FPGA
Present work	Baseline	H.264 SoC	HW-SW co-design	Virtex-5 FPGA

5.2.1 Profiles and Levels

H.264 standard defines mainly a set of three profiles (Baseline, Main, High), each support a particular set of modules such as CAVLC, IDCT, MC etc., and also each specify the requirements of encoder or decoder that complies with a particular profile [135, 137]. The Baseline profile supports intra, inter-coding (using I-slices and P-slices) and entropy coding with CAVLC techniques. The Main profile supports interlaced video, inter-coding using B-slices with weighted prediction and entropy coding using Context Adaptive Binary Arithmetic Coding (CABAC) technique. The Extended profile does not support interlaced video or CABAC

but enable efficient switching between coded bit-streams (SP and SI slices) and has improved error resilience. Applications of the Baseline profile include video telephony, video conferencing and wireless communications; potential applications of the Main profile include television broadcasting and video storage; and the Extended profile may be particularly useful for streaming media applications. Figure.5.1 shows the H.264 decoder profiles and their features. From this figure, it is clear that the Baseline profile is a subset of the Extended profile, but not of the Main profile. Performance limits for codecs are defined by a set of Levels, each has limits on parameters such as sample processing rate, picture size, coded bit-rate and memory requirements [5].

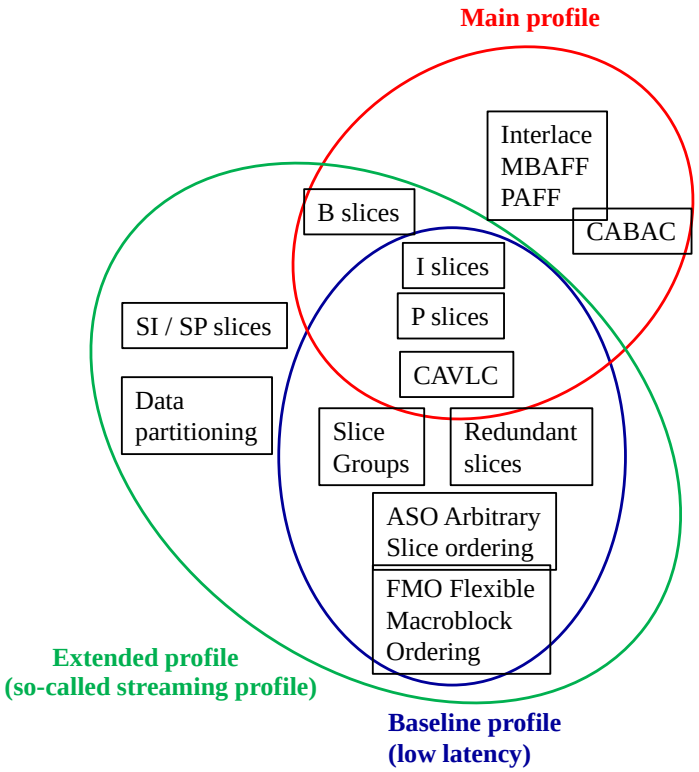


Figure 5.1: H.264 decoder profile configurations [5]

5.2.2 Encoder (forward path)

In the video compression unit, the raw input image is in RGB format. As human eyes are more sensitive to brightness than the color informations, the video signals are processed in YCbCr format (Y is the luminance component which represent the brightness information whereas Cr and Cb are the chrominance components which

represents the color information). In the compression system, an image is divided into macro blocks. As human eyes are more sensitive to luminance information, the idea is to use higher resolution for the luminance component (more samples) and smaller resolutions for the chrominance components. According to the target applications three types of sub-sampling i.e. 4:4:4, 4:2:0 and 4:2:2 techniques are used. In the baseline profile decoder 4:2:0 format is used in which both the luminance and the chrominance components of the signal are sub-sampled by a factor of 2 in both horizontal and vertical direction. Therefore, a macroblock consists of one block of 16 x16 pixels for the luminance component and two blocks of 8 x 8 pixels for the color components [5].

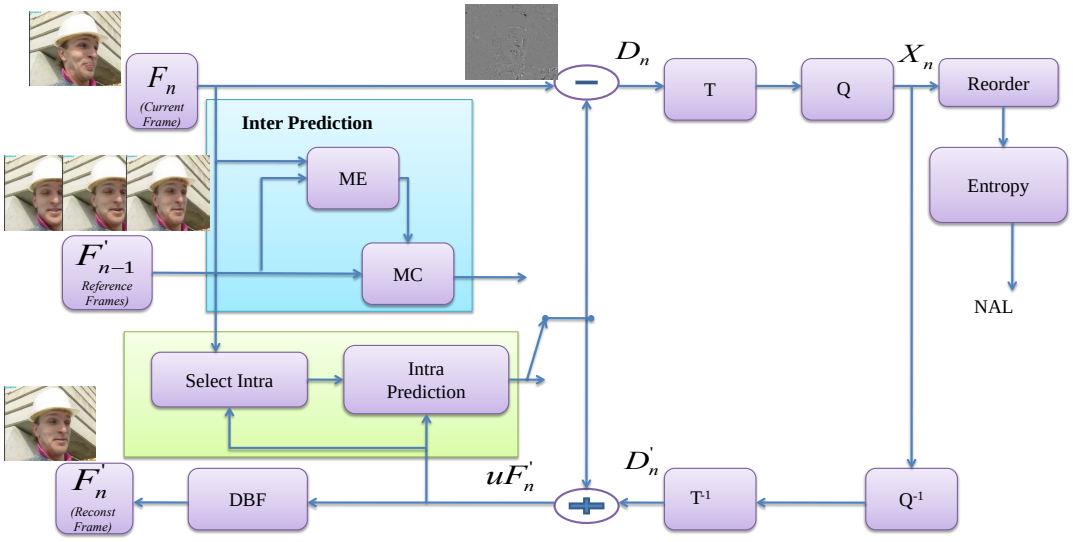


Figure 5.2: H.264 encoder [5]

The video encoder shown in Figure.5.2 includes two data flow paths, a) forward path and b) reconstruction path. The data flow path in the video decoder is shown in Figure.5.3. Before examining the details of H.264 decoder, the main steps in encoding and decoding a frame of video signal is briefly explained below. An input frame F_n is processed in units of macroblocks. Each macroblock is encoded using either intra or inter mode. For each frame, the current macroblock is predicted using either the neighboring macro blocks of same frame or reference frame. In intra mode, prediction of macroblock is from the neighboring macro blocks of current frame that have been previously encoded, decoded and reconstructed ($uF'n$ is the unfiltered samples are used to form prediction). In case

of the inter mode, prediction is carried out by motion-compensated prediction using a reference frame. In the Figure.5.2, the reference frame is shown as the previous encoded frame F'_{n-1}. A macroblock partition (in inter mode) may be chosen from a selection of past frames that have already been encoded, reconstructed and filtered. The predicted macroblock/frame is subtracted from the current macroblock/frame to produce a residual (difference) block $D'n$ that is transformed (using a block transform), quantized to give Xn followed by reordering and entropy encoding. The entropy-encoded coefficients, together with control informations (prediction modes, quantizer parameter, motion vector information, etc.,) form a Network Abstraction Layer (NAL) for transmission or storage [5].

Encoder (Reconstruction path)

For encoding and transmitting each macroblock, the encoder decodes each macroblock in a frame and the reconstructed frame is set as the reference frame for further predictions. The coefficients X_n are scaled (Q^{-1}) and inverse transformed (T^{-1}) to produce a difference block $D'n$ as shown in Figure.5.2. The prediction block is added to $D'n$ to create a reconstructed block $uF'n$ (a decoded version of the original block; u indicates that it is unfiltered). A Deblocking filter (DBF) is applied to reduce the effects of blocking distortions and the reconstructed reference frame $F'n$ is created from a series of macroblocks [5].

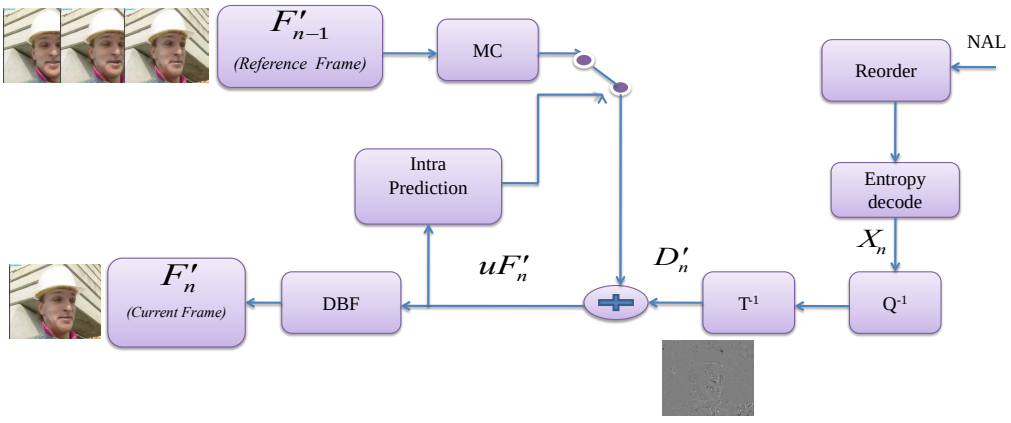


Figure 5.3: H.264 decoder [5]

5.2.3 Decoder

The decoder receives the compressed bitstream from the NAL unit, the entropy decoder decodes the data elements to produce a set of quantized coefficients X_n as shown in Figure.5.3. These are scaled and inverse transformed to give $D'n$. Using the control informations of the bitstream, the decoder creates a prediction block, identical to the original prediction block formed in the encoder. This predicted block is added to $D'n$ to produce $uF'n$ which is filtered to reconstruct the frame $F'n$.

5.3 FPGA implementation of H.264 decoder

A key design step for implementing the H.264 decoder is the selection of implementation strategy, optimizing at different design levels, and properly partitioning the decoding tasks between different blocks. A video codec has to decode the video signal greater than or equal to 30 fps. In order to speed up the execution of the decoder, the time-consuming operations like interpolation in MC and DBF should be carefully optimized or implemented in the hardware. It is difficult to achieve higher acceleration in case of general purpose processor. This is because of sequential processing nature of the processor. In case of the DSP processors decoders need special instructions with parallel processing capability to speed-up some critical modules of the H.264 decoder [146, 163, 164]. In case of FPGA, mapping of video processing algorithms into its resources provide inherent concurrency and parallelism in execution results to superior acceleration over DSP, GPP hardware [148].

The importantstep towards H.264 decoder realization is to prototype it in the FPGA and use it in commercial products. Implementation of H.264 in FPGA faces multiplelevels of complexity because, real-time video compression requires high throughput, larger memory bandwidth etc. In the present work a dedicated hardware/coprocessor for H.264 is developed in FPGA for achieving higher acceleration. Table.5.2 shows that as video resolution increases operating frequency of the decoder also increases. This enhances the memory requirements from (QCIF (176x144) to HDTV 1080p (1920x1088)) [6].

Table 5.2: Frequency requirement for processing 30 fps for different video resolutions [6]

Resolution	**Macroblock perframe)**	**number of Macroblock**	**frequency**
QCIF	99	2,970	1.78MHz
CIF	396	11,880	7.13MHz
4CIF	1,584	47,520	28.5MHz
720p	3,600	108,000	64.8MHz
1080p	8,160	244,800	146.9MHz

Generally a video decoder requires a lot of resource for hardware implementation for faster processing of macroblocks. The key aspect of H.264 hardware is memory management and pipeline processing. The decoder demands more memory access since the macroblocks are stored in the memory. So memory management is a critical task while designing the hardware for H.264 decoder [149, 165]. Accessing large memory introduces more routing delay that results in low operational frequency. This greatly effects the throughput of the design. So, FPGA based design can fulfill these demands with a trade-off between area and speed.

A complete video decoding system comprises of four parts, a memory for storing encoded bitstream, bitstream controller, reconstruction data-path, and a display controller, as depicted in Figure 5.4. Bitstream data is fetched by the bit stream controller according to the structure of the encoded bitstream (each video standard has its own specific bitstream structure). The bitstream parser processes the syntax elements ranging from frame level to macroblock level, while the CAVLC decoder handles transformed/quantized residues. In this process each macroblocks is decomposed into 16 luma of 4x4 blocks and 8 chroma (Cb and Cr) of 4x4 blocks. Intra/ Inter prediction modules are used for prediction of the macro block, according to the current macroblock mode (i.e. 4x4 or 8x4 or 16x16 etc.,). After intra/inter prediction, the destination of predicted 4x4 blocks are controlled by pipeline synchronizer; if it arrives earlier than the residual 4x4 block, it is put into the prediction buffer and waits for residual arrival; once residuals are recovered by Q^{-1} and T^{-1} decoder they are added with predicted residual 4x4 blocks and filtered using DBF. The final decoded macroblock pixels are sent to display engine [140]. The reconstructed video is usually in YCbCr (YUV) format. The decoded YUV video sequence is sent to display controller, which converts YUV format into RGB format for display.

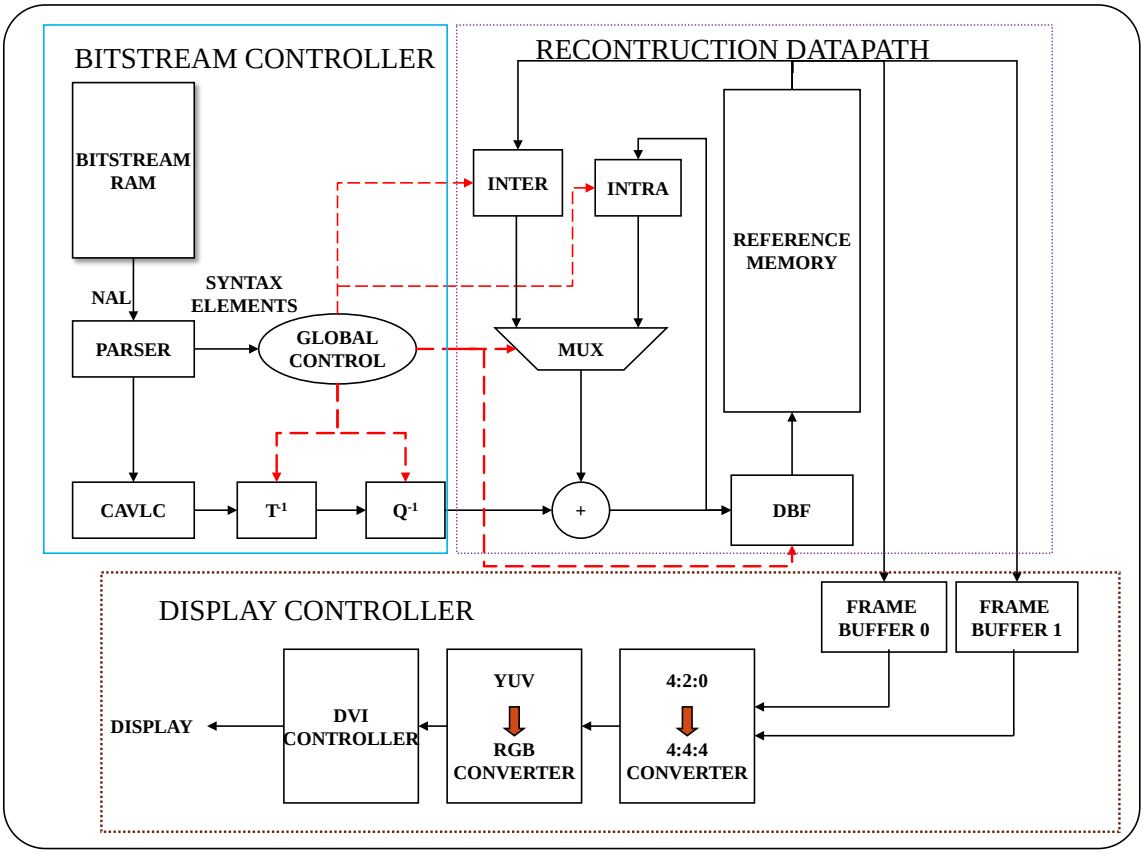


Figure 5.4: H.264 Hardware Block diagram

The considered reference hardware architecture uses pipelining for processing of both 16x16 and 4x4 blocks [6]. The whole decoder is designed by pipelining 4x4 block, except deblocking filter which requires pipelining of 16x16 macroblock. However 16x16 block processing suffers several disadvantages such as it requires intermediate buffer of several mega bytes size to save temporary results of the whole 16x16 macroblock. Also, there is no data reuse inside each macroblock because the entire macroblock is processed as a whole [166]. The modules apart from the DBF, intra prediction, MC, IDCT uses 4x4 block pipeline have three major advantages: first, it matches the smallest block size specified in H.264/AVC standard, second, temporary memory, such as registers or internal memory used to back up intermediate data can be substantially smaller than 16x16 (in fact it uses 4x4). This saves computational time to store the samples. At last, it exploits the data reuse capability between neighboring 4x4 blocks inside a macroblock. The disadvantage of 4x4 pipeline is the degradation of system throughput. This

is because the prediction need to be synchronized for every 4x4 block. However, this can be partially compensated by several other design techniques, such as self-adaptive pipeline and parallel architecture [167, 168]. In order to reuse the 4x4 blocks, the processing order of this block can be either of 1x4 column decoding or 4x1 row decoding [6]. In the considered design 1x4 column decoder is used for Q^{-1}, T^{-1} and intra/inter prediction whereas for sum (Adder) block, 4x1 row order is used. Verilog code of the decoder is used and synthesized in Virtex-5 development board. The important features of the considered open-source decoder is given below [141].

- It supports H.264AVC baseline decoding of QCIF resolution.
- It uses pipelining, parallel architecture along with data reuse mechanism to improve the throughput of decoder.
- It uses clock gating technique to reduce power consumption.
- It uses self adaptive hybrid pipeline architecture for inter/intra prediction module with hierarchical memory organization.
- It uses Multi functional processing elements (MFPE) and seed method for planar mode prediction in intra prediction.
- Inter prediction uses Variable Block Shape (VBS) with pipelined luma/chroma interpolators.

5.3.1 Bitstream controller

The bitstream controller of H.264/AVC baseline profile decoder is shown in Figure.5.4. It comprises of bitstream buffer, parser and hybrid length decoder. The hybrid length decoder has three dedicated decoders for intra-prediction mode selection, motion vectors and Boundary Strength (BS) calculation for DBF.

5.3.1.1 Bitstream buffer

The bitstream buffer is shown in the bitstream controller (Figure.5.4). It serves as a bridge between the off-chip bitstream RAM and the on-chip hybrid length decoder. It keeps the current 128-bit bitstream to be processed. The addressing of bitstream buffer is done in bitwise fashion instead of byte wise, because the input

bitstream has been coded as hybrid length without a fixed byte boundary [169]. The bitstream buffer is modeled as a circular buffer having two hardware processes which manage the communication between the off-chip RAM, the bitstream buffer and the hybrid length decoder [6].

5.3.1.2 Bitstream parser

The bitstream parser fetches encoded bitstream from the bitstream buffer and parsing of the bitstream is handled by a control FSM [168, 169, 170]. The control FSM identifies the codeword and subsequently boundary of each codeword is identified, further the whole codeword is extracted and decoded according to the syntax of control information. The codeword has the informations about the header and residuals. CAVLC decoding is performed for coded residuals while fixed/variable-length decoding is performed on the control data. Intra and inter prediction mode and motion vector are derived from both current decoded syntax value and neighboring coding information available in the control information. The control informations have also the BS of each macroblock required for proper decoding the video.

5.3.1.3 Hybrid length decoder

The hybrid-length decoder consists of three different modules, namely Exp-Golomb, fixed-length and CAVLC. Once the bitstream buffer gets the data it sends to the hybrid length decoder. In this module (as per the input codeword type) the heading-one-detector is invoked for identifying the starting position of the codeword by detecting first appeared 1 inside the current syntax element [6]. CAVLC module is invoked when syntax of residual information is appeared, while for syntax other than residual, either Exp-Golomb or fixed-length decoding is performed. When explicitly encoded syntaxes are decoded, control parameters and dependent variables are decoded subsequently [168].

5.3.2 Reconstruction data path

The reconstruction data path of H.264/AVC baseline profile decoder is shown in Figure.5.4. It comprises of intra/inter prediction, DBF modules with on chip memory controllers. Predicting pixels in sequential fashion degrades the system performance, so in order meet the real time decoding requirements of 30fps at

1.5MHz, adaptive pipelines and parallel processing of the data path is employed in the architecture [168]. Intra/Inter prediction module, is the major computational bottle neck in H.264 decoder, this prediction scheme demand a large amount of memory accesses and account for up to 80% of the total decoding cycle in the complete baseline profile decoder [167]. In order to reduce the bus latency on-chip memory is used for all memory operations [6].

5.3.2.1 Intra prediction

The intra prediction module is shown in Figure.5.4. Each macroblock can be encoded into one of several intra coding types, which are denoted as intra 4x4 or intra 16x16, together with the chroma prediction and Intra Pulse Code Modulation (IPCM). There are nine prediction modes for luma 4x4 block, four modes for luma 16x16 block and four modes for chroma 8x8 block components. The encoder selects the suitable mode for the prediction and transmits this information to the decoder as control information [6, 167]. Intra module uses 4x4 level pipeline to reduce the number of processing cycles. The intra prediction module uses three-level memory hierarchy with data reuse mechanism, Multi Function Processing Element (MFPE) and plane mode decomposition modules are used for macroblock prediction.

5.3.2.2 Inter prediction

The inter prediction module is a computational intensive module in the decoder as is shown in Figure.5.4. Inter prediction is mainly divided into two macro stages, reference data fetch and interpolation stages [167]. In general, for inter prediction, all reference blocks need to be fetched from the external memory, this poses a heavy burden on off-chip memory bandwidth. So the hierarchical memory organization is used to reduce the off chip memory access by using on chip memory. It uses self-adaptive pipeline and tree-structured motion compensation algorithm to eliminate unnecessary circuit switching which in turn increases the operating speed while retaining the constant throughput [6]. The self adaptive pipeline mechanism uses 4x4 block level pipeline, process 1x4 pixels in a column wise fashion simultaneously. It uses different macro block partitions and motion vectors for pixel prediction. The pipeline structure recognizes all the possible combinations of blocks using Variable Block Shape (VBS) technique. The Inter prediction module consists of interpolators for luma/chroma components and they process

the samples sequentially. It is reported that the inter prediction module takes 500 cycles for decoding a single macro block [167].

5.3.2.3 Deblocking filter

Deblocking filter (DBF) is used to reduce the blocking distortion of each 16x16 macroblock after reconstruction of the frame as shown Figure.5.4. The DBF consumes one-third of the computational requirements of a H.264/AVC decoder [140, 166]. This filtering process requires larger memory bandwidth because almost every sample of a reconstructed frame need to be reused from memory, either to be modified or used to determine whether neighboring samples should be modified or not. The processing of pixels in the DBF takes place on macroblock basis, starts with vertical edges being filtered, followed by horizontal edges. Filtering in both horizontal and vertical directions for each macroblock should be completed before accessing the next macroblock. The filtering process starts with luma, followed by chroma Cb and the chroma Cr.

Finite Impulse Response (FIR) filters of different length are applied to modify the pixel values of the boundary of each macro block. The filter length depends on the the BS of the macro block. All boundary edges should be conditionally filtered as per the video coding standard [135, 166]. The condition depends on the BS and the pixel gradient across the boundary. This information is accessed from the BS decoding module which is inside the hybrid length decoding module. The BS values lies between (0, 4), and is assigned to each sub edges before the entire deblocking process starts. BS equals to 4 refers to strong filtering; BS equals to 0 means no filtering; otherwise, a more common-mode filter is applied for BS (1, 2, 3). The BS of any chroma edge is identical to its corresponding luma edge. The pipeline architecture of DBF reduces the average time required for filtering operation. The present work uses 5-stage pipeline and single edge filter. This architecture reduces 55% of total cycles. It consumes 204 cycles for filtering of one 16x16 macroblock without utilizing dual-port or two-port SRAM [166, 6].

5.3.3 Display controller

The display controller can be regarded as post-processing block of video decoder. First, 4:2:0 YCbCr format is padded to 4:4:4 format. Then the padded frame is converted to RGB format. Each pixel is of 24-bits and stored in a large display memory. According to the timing requirement of the display i.e. 25MHz, RGB pictures are sequentially fetched from display memory and displayed on the Liquid Crystal Display (LCD) monitor [171].

5.4 Programmable System on Chip (PSoC) platform for H.264 decoder

In this case study, an IP core of H.264 decoder is developed using the open source RTL code and validated in the SoC platform. Although the reference design was implemented in ASIC and FPGA (Virtex-4) [141, 168], but not explored in FPGA based SoC platform. The uncertainty of resource quality is the one of the biggest problems to use open source IP, after a series of comprehensive testing only, we can use it in real time systems. The following modifications are done in the reference design in order to implement in the FPGA based SoC platform.

Replacing behavioral RAM

The decoder is modified by replacing behavioral RAMs by on chip memories Xilinx single port or dual port memories (version 6.2) are used in intra prediction, MC and DBF modules. Along with this two ping-pong dual port memories and one display memory are used for displaying output frame. The bit stream memory controller uses 64Kbytes of memory for storing 300 frames of data. The bit stream buffer uses two memories to store the present frame (10Kx32 bytes). The intra prediction module uses single port memory of 256x32 bytes. Similarly DBF uses three memories one of 352x32 bytes size and other two are of 96x32 bytes size. One memory controller of 10Kx32 bytes for processing the YUV data. After conversion of YUV data to RGB format followed by sub-sampling, the updated RGB data is stored in a memory controller of size 30Kx24 bytes. This buffer is used as a display buffer for storing the RGB data before sending it to display [168]. The

future work involves the use of external memory controller for all the modules, for higher resolution video.

Removal of clock gating

As the reference design was targeted for ASIC implementation, clock gating technique was used to reduce power consumption [168, 169]. In order to implement the reference design in the FPGA clock gating should be removed and chip enable logic need to be included [168]. In the modified design, clock gating is removed and distributed chip enable signals are included using the global control module.

Display buffer

The modified design is interfaced with the YUV-RGB converter and color space conversion module (4:2:0) to (4:4:4) for displaying the frame in monitor. The Virtex-5 FPGA development board has a DVI controller which converts 24-bit into 12-bit DVI standard data and displays [171].

Clock generation

The modified design of H.264 decoder works at 1.5 MHz and LCD controller works at 25 MHz. The complete decoder has an input clock frequency of 25 MHz. It has two Digital Clock Managers (DCM 1 and DCM 2) which are connected in cascaded manner [60]. The DCM 1 is used for dividing the input clock by 16 and DCM 2 is used (low frequency mode) to divide the DCM 1 output by two. The output frequency of DCM 2 (1.5 MHz) and is fed to the input of H.264 decoder. Input frequency of 25MHz clock is also fed to input of LCD controller. Since Virtex-5 Device does not have DCM, Xilinx Synthesizer tool (XST) infers the DCM as PLL.

5.4.1 SoC platform details.

The reference design is modified as [140, 141] and successfully implemented in Virtex-5 FPGA development board. For testing the design, the internal (on-chip) memory is used to store the input encoded bit stream. This internal memory uses Block-RAM which gives enough flexibility for accessing the video at a faster rate,

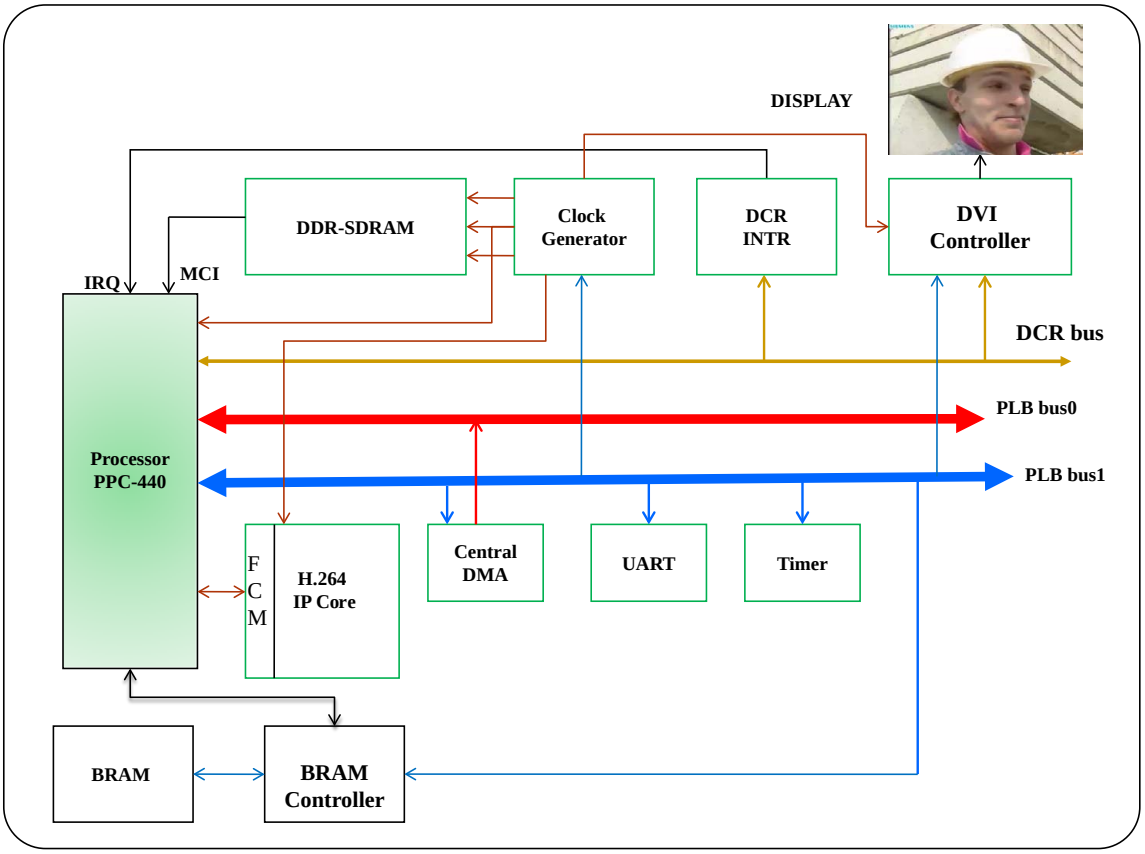


Figure 5.5: PSoC platform for H.264 decoder IP

however in SoC system this is replaced with external DDR2 interface for storing the input encoded video data.

The complete H.264 IP is developed and integrated into the SoC platform as shown in the Figure.5.5. The IP is interfaced with the PPC440 processor using the APU interface technique. The system include IPs like PowerPC440 (PPC440), PLB bus, DDR2 (256 Mbytes), Clock generator, UART, Central DMA Controller, Interrupt Controller and LCD Display Controller. DDR2 is used for storing the heap and stack of the software as well as to store the input bitstream data. The SoC platform has two PLB buses (PLB0 and PLB1) and 1 DCR bus and 1 FCB bus. The two PLB buses are connected to the processor through PLB0 and PLB1 bus interface. PLB0 and central DMA are connected as Master port of the processor and PLB1 of processor is shared with all other peripherals including the processor and DMA. The central DMA has both slave and master ports, this helps to transfer the DDR2 data to memory controller through PLB0 Bus. DCR

bus and DCR interrupt controller are useful for setting the display address of DVI controller. All the IPs except H.264 decoder IP are interfaced using the PLB bus whereas the decoder IP is interfaced using the FCB bus. The FCB bus is connected to H.264 decoder. The profile timer and interrupt controller shown in Figure.5.5) are also connected to slave bus PLB0 to evaluate the execution time of the both hardware and software of the H.264 decoder. The frequency of SoC platform, display controller and the H.264 decoder IP are set to 200 MHz, 25MHz and 1.5 MHz respectively. All these frequencies are configured by using a clock generator IP as shown in Figure.5.5.

The H.264 video decoder IP has internal modules for intra frame prediction, IDCT, IQ, MC and DBF. This IP has two generic FIFO interfaces as discussed in chapter 1. The H.264 APU is called through a Application Programming Interface (API). Once it is invoked, the operation of H.264 hardware module starts working. The H.264 IP has two internal data buffers one is for storing the H.264 encoded bitstream data from DDR2-SDRAM and other is for storing the frame data for displaying the decoded data in the monitor. In this case, the central DMA controller transfers the bitstream data of 300 frames onto the on chip input buffer for processing. The on-chip bit stream memory is used to reduce the bus latency. The bitstream parser processes the on-chip buffer data according to the syntax of the bitstream data and subsequently the decoding process is completed as mentioned in previous Section 5.3 and the decoded video is displayed in the monitor.

5.5 Results and analysis

The SoC system is tested using the Virtex-5FXT FPGA development board. For validating the IP, different video sequences are decoded and the performance of the SoC design is evaluated. This is tabulated in Table.5.4. Before validating the complete SoC, individual modules of the IP and the complete IP are simulated at various levels of design, such as behavioral simulations shown in Figure.5.6, Figure.5.7 and post synthesis simulation, post place and route simulations as shown in Figure.5.8 and Figure.5.9 respectively.

All the simulations are compared with the behavioral simulation, it is observed that the results are matching except some additional delays observed in post layout simulation. The behavioral simulation shows the output decoded data after

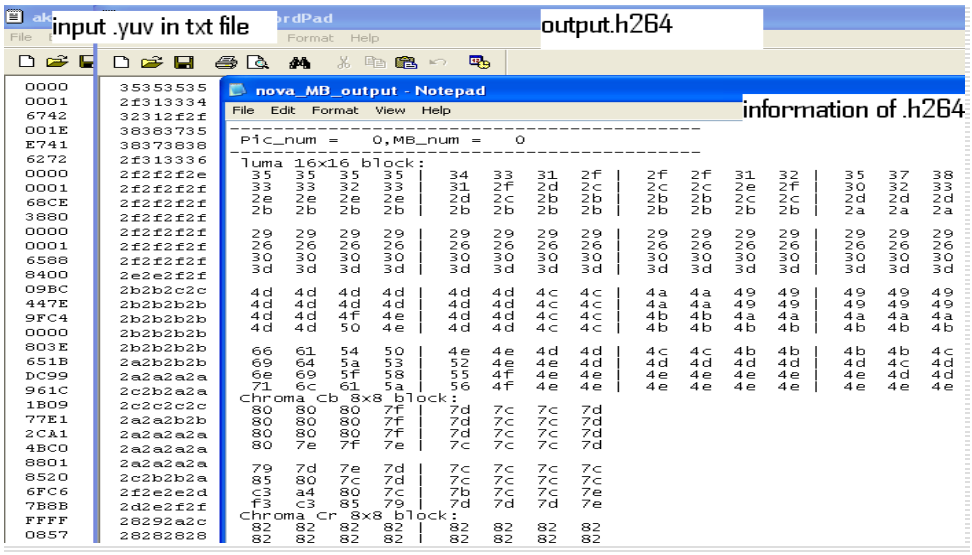


Figure 5.6: Behavioral simulation of H.264 decoder in text file

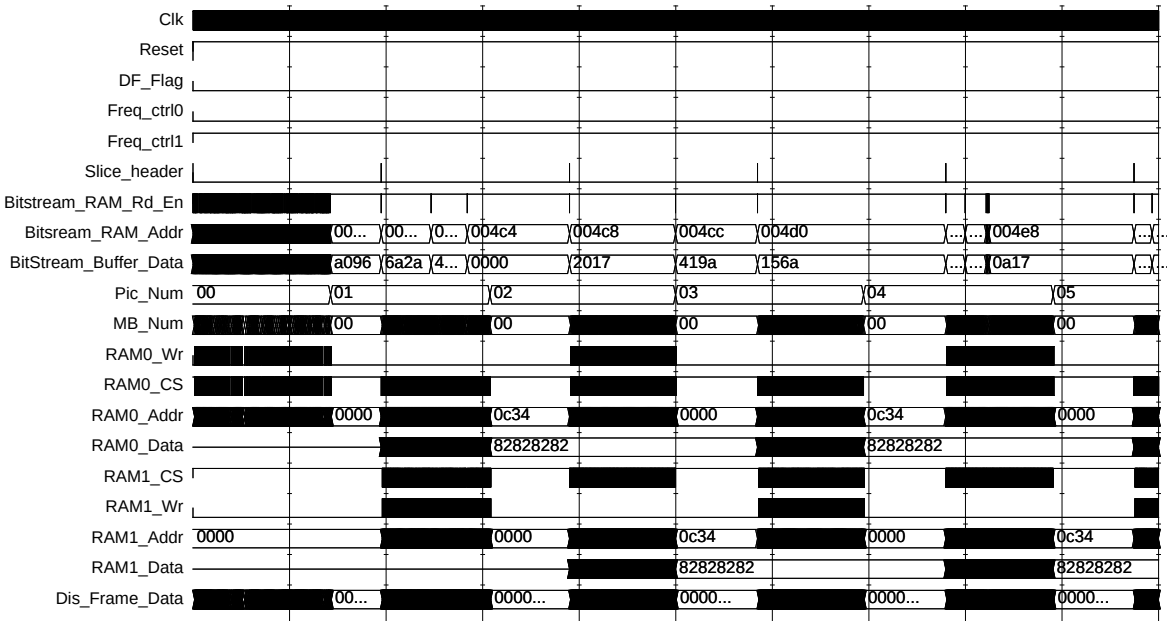


Figure 5.7: Behavioral simulation of H.264 decoder

40ms. The first frame is decoded and indicated by frame-end (eof) signal. The other signals like number of macroblocks (mb_num) and frame number (pic_num) are used to verify and debug the design. After this, the complete SoC is developed as shown in Figure.5.5 using Xilinx EDK environment. The input stimuli are applied to the prototype system through DDR2 and Central Direct Memory Access (CDMA). For functional and timing verification, the SoC decoded results

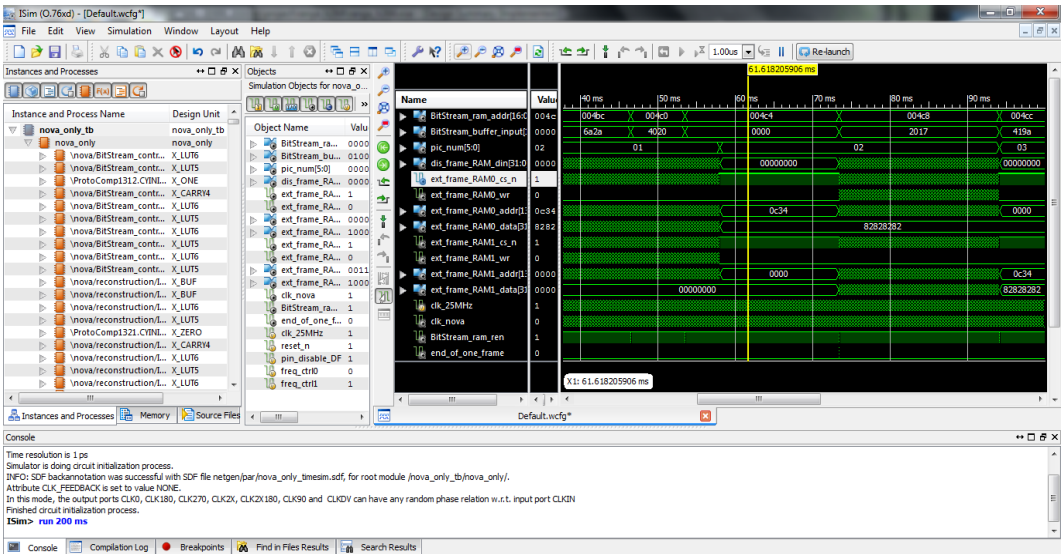


Figure 5.8: Post-synthesis simulation of H.264 decoder

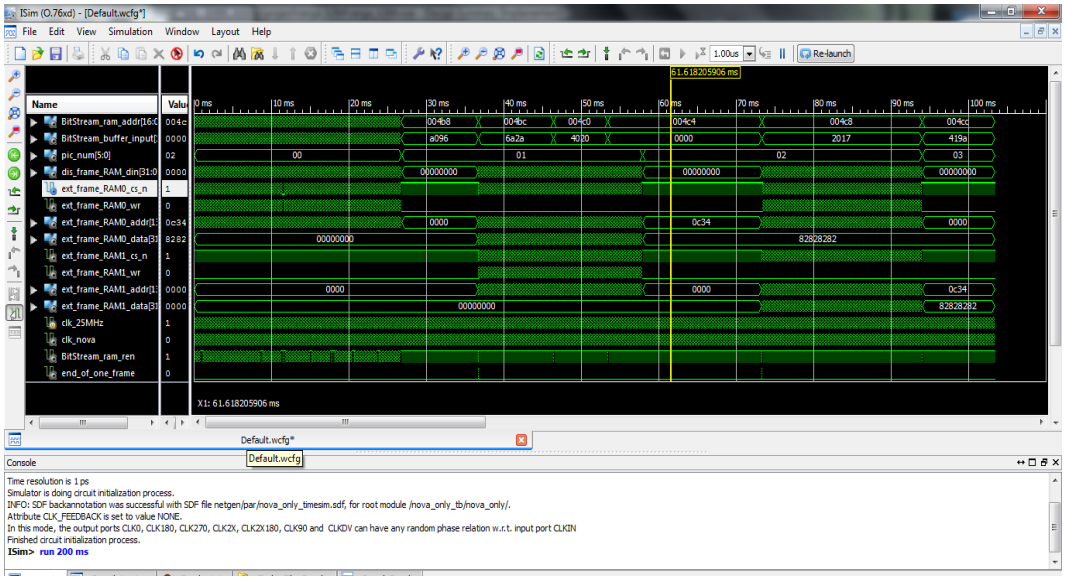


Figure 5.9: Post-layout simulation of H.264 decoder

are compared with the reference software results, and resulting video on the display monitor.

Prior to this, software profiling of H.264 decoder of (JM 9.4) in PPC440 processor with standalone and Petalinux RTOS are carried out. The profiling results are shown in Figure.5.10 and Figure.5.11. It is observed that DBF, MC together consumes 70% of the total computation time for decoding a test video sequence

(Foreman). Further, resource utilization of the complete SoC is carried out and it is found that 81% of resources are consumed due to the H.264 IP and additional 6% of resources are due to the DDR2 controller and other peripherals as shown in Table.5.3. The maximum frequency of H.264 decoder and operation frequency of SoC, H.264 IP and LCD clock are tabulated in Table.5.3. The test bed for H.264 coprocessor platform is shown in Figure.5.12. The acceleration is measured in terms of no of frames processed with respect to time. It is observed that for decoding 300 frames of foreman sequence the embedded processor (PPC440) took 72.06 seconds whereas the hardware H.264 took 11.40 seconds. i.e. the software H.264 decoder decoded very few frames (4-5 fps) per second compared to the hardware (26-30 fps). This statistics remains valid for all tested video sequences. The results are tabulated in Table.5.4. From this it can be concluded that the developed accelerator gives an acceleration of 6-7x over software implementation.

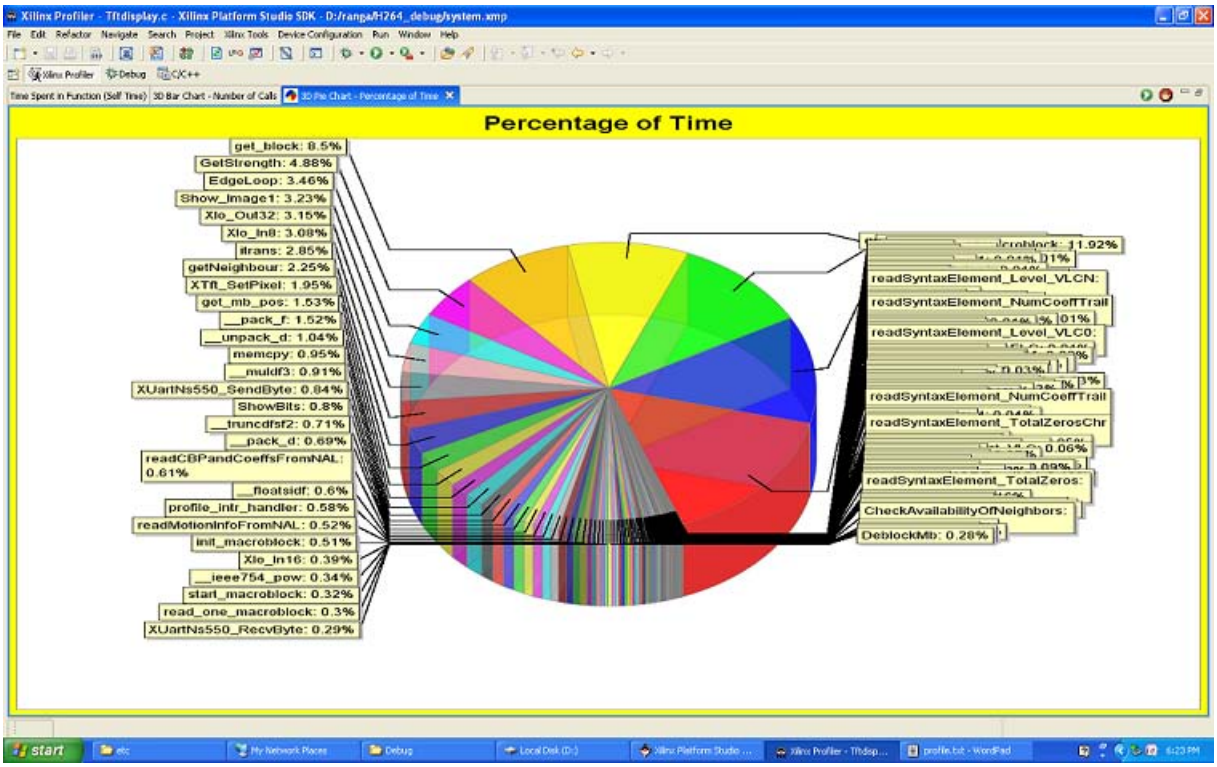
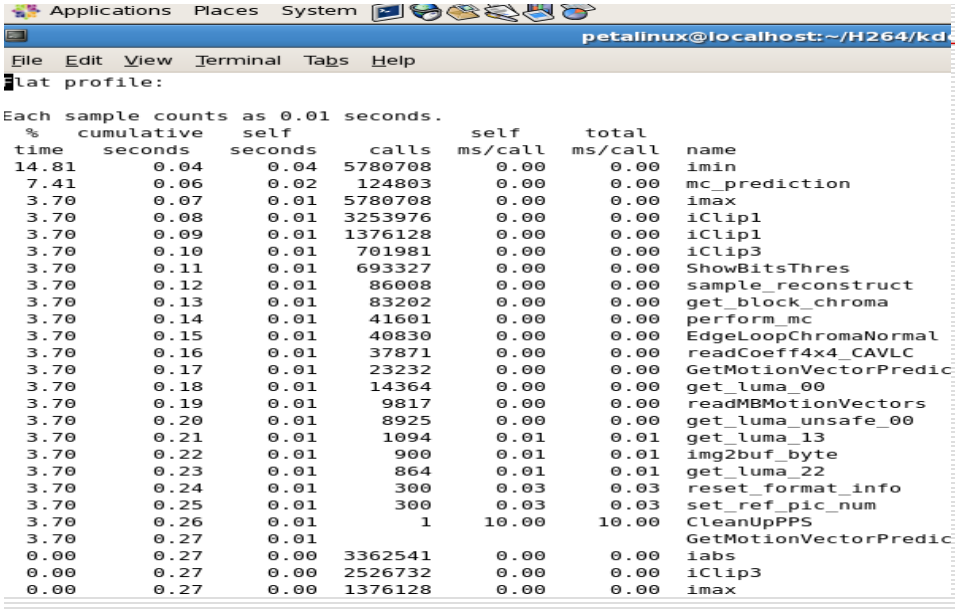


Figure 5.10: Software H.264 decoder profiling on PPC440 processor in standalone



Applications Places System

petalinux@localhost:~/H264/kd

File Edit View Terminal Tabs Help

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 14.81      0.04     0.04  5780708     0.00     0.00  imin
  7.41      0.06     0.02   124803     0.00     0.00  mc_prediction
  3.70      0.07     0.01  5780708     0.00     0.00  imax
  3.70      0.08     0.01  3253976     0.00     0.00  iClip1
  3.70      0.09     0.01  1376128     0.00     0.00  iClip1
  3.70      0.10     0.01   701981     0.00     0.00  iClip3
  3.70      0.11     0.01   693327     0.00     0.00  ShowBitsThres
  3.70      0.12     0.01    86008     0.00     0.00  sample_reconstruct
  3.70      0.13     0.01    83202     0.00     0.00  get_block_chroma
  3.70      0.14     0.01    41601     0.00     0.00  perform_mc
  3.70      0.15     0.01    40830     0.00     0.00  EdgeLoopChromaNormal
  3.70      0.16     0.01    37871     0.00     0.00  readCoeff4x4_CAVLC
  3.70      0.17     0.01    23232     0.00     0.00  GetMotionVectorPredic
  3.70      0.18     0.01    14364     0.00     0.00  get_luma_00
  3.70      0.19     0.01     9817     0.00     0.00  readMBMotionVectors
  3.70      0.20     0.01     8925     0.00     0.00  get_luma_unsafe_00
  3.70      0.21     0.01     1094     0.01     0.01  get_luma_13
  3.70      0.22     0.01      900     0.01     0.01  img2buf_byte
  3.70      0.23     0.01      864     0.01     0.01  get_luma_22
  3.70      0.24     0.01      300     0.03     0.03  reset_format_info
  3.70      0.25     0.01      300     0.03     0.03  set_ref_pic_num
  3.70      0.26     0.01        1    10.00    10.00  CleanUpPPS
  3.70      0.27     0.01                             GetMotionVectorPredic
  0.00      0.27     0.00  3362541     0.00     0.00  iabs
  0.00      0.27     0.00  2526732     0.00     0.00  iClip3
  0.00      0.27     0.00  1376128     0.00     0.00  imax
```

Figure 5.11: H.264 decoder profiling on PPC440 processor in petalinux OS

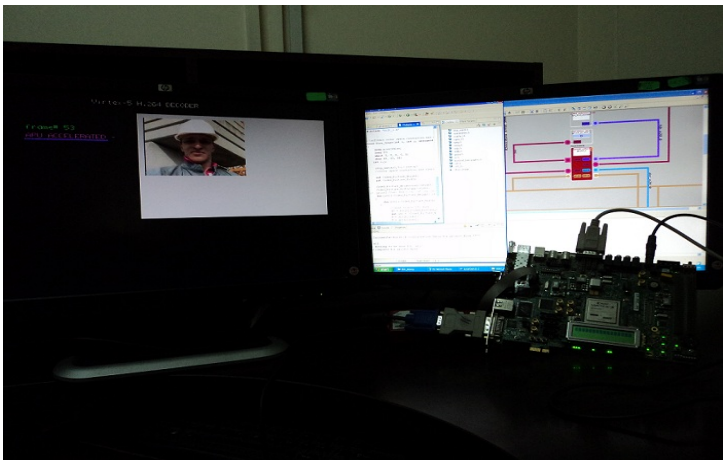


Figure 5.12: H.264 Hardware coprocessor testbed setup

Table 5.3: Resource utilization of H.264 decoder in SoC

Decoder/SoC system	BRAM	DSP48E	Slice Registers	Slice LUTs	Slices	Max Freq (MHz)
H.264 IP	105 (70%)	4 (3%)	9209 (20%)	26883 (60%)	33600 (75%)	107.773
H.264 SoC	122 (86%)	24 (19%)	16452 (37%)	35004 (78%)	35432 (78%)	200 & 1.5& 25

5.6 Conclusions

In this case study an open source H.264 decoder is used for developing a coprocessor to the PPC440 processor and validated in SoC platform. The H.264 decoder

Table 5.4: Evaluation of speed up for different sequences for 300 frames for Quantization Parameter (QP)=28

Sequence	**SW H.264 F/s**	**SW H.264 Sec**	**HW H.264 F/s**	**HW H.264 Sec**
News	4.68	63.84	29.4	10.20
Akiyo	4.74	63.11	28.9	10.36
Salesman	4.42	67.61	28.6	10.52
grandma	4.46	67.04	28.1	10.64
Mother-Daughter	4.37	68.42	28.0	10.70
Claire	4.33	68.98	27.6	10.84
Carphone	4.25	70.41	26.7	11.20
Foreman	4.15	72.06	26.3	11.40

has several advantages such as pipeline and parallel processing, memory access reduction etc. In this work, the open source H.264 decoder is modified and used as coprocessor. The modified decoder is implemented in FPGA and verified its functionality before integrating as IP into SoC. In this work the H.264 IP is tested in various simulation levels and it is observed that the behavioral and post layout simulations are matching with each other. Further it is used as a coprocessor in SoC by interfacing the IP as an APU to the PPC440 processor. Software profiling of H.264 decoder is also performed in SoC platform. The proposed SoC solution of H.264 decoder gives an acceleration of 6-7x compared to the equivalent software implementation. The future work involves i) modification of IP to cater higher resolution of video by replacing internal memories with external memory, ii) power analysis of the SoC, iii) interfacing the developed IP as a Slave Unit (SU) and compare its acceleration with APU interfacing.

Chapter 6

Conclusions

The computational complexity of signal processing and multimedia algorithms limit its implementation in embedded processors. This thesis explores the design methodology for designing coprocessors of signal processing algorithms and shown that coprocessor achieves significant acceleration for executing the algorithms. Three different case studies are examined and to accelerate the design speed hardware Intellectual Property (IP) for each case study was developed, The IPs were interfaced with PowerPC440 (PPC440) embedded processor using Auxiliary Processing Unit (APU) interfacing technique, and validated in Xilinx Virtex-5 FPGA development board. The thesis compared the acceleration due to the coprocessor with respect to the software execution of the algorithm in an embedded processor. In summary, the conclusions of three case studies are given below.
In the first case study, an efficient algorithm for denoising Fiber Optic Gyro (FOG) signal is proposed. The performance of the algorithm is compared with the Discrete Wavelet Transform (DWT) and conventional Kalman Filter (KF) technique. The experimental results concluded that the proposed algorithm (AMADMKF) denoises the signal efficiently compared to other algorithms. Furthermore, a hardware IP of the algorithm is developed and interfaced with the PPC440 processor using APU interface technique and found that the developed IP gives an acceleration of 65x compared to its equivalent software implementation.

In embedded system design hardware/software co-design is a popular method for designing complex systems. Although co-design platform gives flexibility to change the design without redesign but the bus overhead due to the communication between processor and IP dominates the overall execution time of the design.

To address this, in the second case study we have considered an optimization algorithm (Differential Evolution (DE)) for building the coprocessor. This algorithm has two major components, i) the algorithm and ii) the fitness function evaluation. In this thesis both are implemented in a single IP to reduce the bus overhead. Two separate IP's for float and fixed point DE algorithm are developed. In these IP, both DE logic and fitness evaluation modules are coupled to a single module. The software of fixed and float DE algorithm are ported in PPC440 processor and profiled at 200 MHz frequency. The execution time for optimizing different test functions are evaluated. The hardware IP of both fixed and float DE IPs are connected to the PPC440 processor using both APU and Slave Unit (SU) interfacing techniques. The performance of both interface methods are evaluated in terms of acceleration and power consumption with respect to equivalent software implementation. Furthermore, for performance evaluation DE software code is ported in different processors and acceleration factors are compared. . In the SoC platform, effect of hard Floating Point Unit (FPU) on processor is also studied by enabling/disabling hard FPU of Microblaze (MB), PPC440 processors. It is concluded that the hardware acceleration of MB based systems are more compared to PPC440 based system. Because, PPC440 based systems takes less execution time compared to MB based system. Although PPC440 with hard FPU is best configuration for software, but floating point DE with either APU or SU interface gives much better acceleration compared to software implementation. It is also concluded that APU and SU interface of both the IPs give same acceleration factor. Acceleration factor increases with the dimension/complexity of the test functions.

The resource and power analysis of different interfacing accelerators have shown that floating point DE IP consumes more resources and power as compared to fixed point IP. The developed IP is also used to solve system identification task using IIR filter in FPGA. The filter coefficients are evaluated using the developed IP with APU interface to PPC440 processor. Results have shown that the IIR filter coefficients are matched with reference coefficients and resulted an acceleration of 11x and 150x with fixed and float DE IPs respectively.

In the third case study, a coprocessor for multimedia H.264 codec is developed using the open source Verilog code. H.264 IP is implemented in Virtex-5 FPGA. The equivalent software C code of H.264 is ported in PPC440 processor and profiled at 200 MHz frequency. The H.264 IP is tested in various simulation

levels for functional verification. H.264 IP is interfaced with the PPC440 processor using APU interfacing method. The performance of the accelerator is validated by decoding different standard video sequences. The results have shown that the accelerator achieved an acceleration of 6-7x compared to its equivalent software implementation in PPC440 processor.

Future scopes

In this thesis, we have concentrated on three different case studies in three chapters having their own contributions and future works. There are several interesting directions for further research and development based on the work in this thesis. In the first case study, the future work will be to extend the hardware for multi sensor environment, ii) Optimizing the AMADMKF IP for lesser resource utilization and high frequency of operation. Finally, the implementation can be extended to Adaptive Kalman Filter (AKF) and other Kalman Filtering techniques without changing the SoC platform and interface details. In the second case study, the future work will be solving real-time optimization problems like parameter adaption of Kalman Filter, Motion estimation in video sequences, radio network design in embedded processor. Different variants of DE algorithm can be implemented in hardware to improve the quality of solution. Finally development of real-time evolutionary engine for future evolutionary system is the future work of this study.

In the third case study, the H.264 IP can be further accelerated for processing higher resolution video by replacing on chip memory with external memory. The performance comparison of SU interface with APU interface for this application is an interesting future work of this thesis. Finally,this work can be extended to implement High Efficiency Video Coding (HEVC) codec in FPGA.

Appendix A

There are six test functions [126, 127] we employed in this paper, which are given below.

Function 1 (Two variables): Rosenbrock function:
$f(x) = 100.(x_2 - x_1^2)^2 + (1 - x_1)^2$
Search domain:$-9 < x_j < 11 \;\; j = 1, 2$
One global optimum with $f = 0 \;\; at \;\; (1, 1)$

Function 2 (Two variables): Goldstein function:
$f(x) = [1 + (x_1 + x_2 + 1)^2 \times (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$
Search domain:$-2 < x_j < 2, \;\; j = 1, 2.$
One global optimum with $f = 3 \;\; at \; (0, -1)$

Function 3 (Three variables): Sphere function:
$f(x) = x_1^2 + x_2^2 + x_3^2$
Search domain: $-5.12 < x_j < 5.12 \quad j = 1, 2, 3$
One global optimum with $f = 0 \;\; at \;\; (0, 0, 0)$

Function 4 (Four variables): Variably dimensioned function:
$f(x) = \sum_{i=1}^{4}(x_i - 1)^2 + \left[\sum_{i=1}^{4} i(x_i - 1)\right]^2 + \left[\sum_{i=1}^{4} i(x_i - 1)\right]^4$
Search domain:$-9 < x_j < 11, \;\; j = 1, 2, 3, 4$
One global optimum with $f = 0 \; at \; (1, 1, 1, 1)$

Function 5 (32 variables): Shifted Sphere function:
$f(x) = \sum_{i=1}^{D} x_i^2 \;\; x = [x_1, x_2, x_3, ..., x_D]$
Search domain:$-100 < x_j < 100 \;\; j = 1, 2, .., 32$

One global optimum with $f(x) = 0 \ \ at \ \ (0, 0, .., 0)$

Function 6 (32 variables): Shifted Schwefel's function:
$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_i \right)^2$
Search domain:$-100 < x_j < 100, \ j = 1, 2, .., 32$
One global optimum with $f = 0 \ at \ (0, 0, .., 0)$.

References

[1] Ron Sass and Andrew G. Schmidt. *Embedded Systems Design with Platform FPGAs Principles and Practices*. Elsevier Inc, USA, 2010.

[2] Don Davis,Srinivas Beeravo,Ranjesh Jaganathan. Hardware / Software Codesign for Platform FPGAs. *Xilinx Application Notes*, 2005.

[3] EDK Concepts, Tools, and Technique: A Hands-On Guide to Effective Embedded System Design . *Xilinx User guide*, (683), 2011.

[4] Xilinx. Reference Guide: Embedded Processor Block in Virtex-5 FPGAs. *Xilinx User guide*, (200), 2008.

[5] Luciano Volcan Agostani. *Developing Architectures for High Performance Dedicated Video Compression According to the H.264/AVC Standard*. PhD thesis, Department of Computer science, Federal University of Rio Grande do Sul, Brazil, 2007.

[6] Ke, Xu. *Power-efficient Design Methodology for Video Decoding*. PhD thesis, Department of Electronic Engineering, The Chinese University of Hong Kong, china., 2007.

[7] Anumandla, KiranKumar and Peesapati, Rangababu and Sabat, SamratL. and Udgata, SibaK. SoC based floating point implementation of differential evolution algorithm using FPGA. *Design Automation for Embedded Systems*, doi:10.1007/s10617-013-9107-4:1–20, 2013.

[8] P.J. Pingree, J.-F.L. Blavier, G.C. Toon, and D.L. Bekker. An FPGA/SoC Approach to On-Board Data Processing Enabling New Mars Science with Smart Payloads. In *Proceedings of the IEEE Conference on Aerospace*, pages 1–12, mar 2007.

[9] Bekker, Dmitriy L and Blavier, J-FL and Toon, Geoffrey C and Servais, Christian. An FPGA-based data acquisition and processing system for the MATMOS FTIR instrument. In *Proceedings of the IEEE Aerospace conference*, pages 1–11. IEEE, 2009.

[10] Walid Farid Abdelghaphar Abdelfatah. Real time embedded system design and realization for integrated navigation systems. Master's thesis, Department of Electrical and Computer Engineering, Queens University, Kingston, Ontario, Canada, 2010.

[11] Abdelfatah, Walid Farid and Georgy, Jacques and Iqbal, Umar and Noureldin, Aboelmagd. FPGA-Based Real-Time Embedded System for RISS/GPS Integrated Navigation. *Sensors*, 12(1):115–147, 2011.

[12] Ansari, Ahmad and Ryser, Peter and Isaacs, Dan. Accelerated System Performance with APU-enhanced processing. *Xcell Journal, first quarter*, 2005.

[13] Pellerin, David and Edvenson, Greg and Shenoy, Kunal and Isaacs, Dan. Accelerating PowerPC software applications. *Xilinx Xcell Embedded Magazine*, 2005.

[14] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on,Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, feb 2007.

[15] Yiannacouras, Peter. *FPGA-based soft vector processors*. PhD thesis, Department of Electrical and computer Engineering, University of toronto, Toronto, 2009.

[16] Vincent Andrew Akpan. *Development of New Model-Based Adaptive Predictive Control Algorithms and their Implementation on Real-Time Embedded Systems*. PhD thesis, Department of Electrical and Computer Engineering, Thessaloniki, Greece, 2011.

[17] Charuchandra, Prabhushankar. FPGA prototyping of the MSP430F5172. Master's thesis, Department of Computer Science and Engineering, University of Gothenburg, Gteborg, Sweden, aug 2010.

[18] Tomáś Brabec. Algorithm Acceleration using FPGAs. Technical report, Czech Technical University, Prague, 2004.

[19] Youn-Long Steve Lin. *Essential Issues in SOC Design Designing Complex Systems-on-Chip*. Springer, P.O. Box 17, 3300 AA Dordrecht, Netherlands, 2006.

[20] Darringer, J. A. and Bergamaschi, R. A. and Bhattacharya, S. and Brand, D. and Herkersdorf, A. and Morrell, J. K. and Nair, I. and Sagmeister, P. and Shin, Y. Early analysis tools for system-on-a-chip design. *IBM J. Res. Dev.*, 46(6):691–707, nov 2002.

[21] Holger Lange. *Reconfigurable Computing Platforms and Target System Architectures for Automatic HW/SW Compilation*. PhD thesis, Department of computer science, Darmstadt Technical University, Germany, 2011.

[22] Zou, Yi. *Coprocessor Acceleration for Domain-Specific Computing*. PhD thesis, Department of Computer Science, University of California, Los Angeles, USA, 2012.

[23] Rosinger, Hans-Peter. Connecting customized IP to the MicroBlaze soft processor using the Fast Simplex Link (FSL) channel. *Xilinx Application Note*, (529), 2004.

[24] Garcia, Philip and Compton, Katherine and Schulte, Michael and Blem, Emily and Fu, Wenyin. An overview of reconfigurable hardware in embedded systems. *EURASIP Journal on Embedded Systems*, 2006, 2006.

[25] Ng, Harn Hua and Pillai, Latha. Accelerated system performance with the APU controller and XtremeDSP slices. *Xilinx Application Notes*, (717), 2005.

[26] Gupta, Gaurav and Jones, Ben and Steiner, Glenn C. PowerPC Processor with Floating Point Unit for Virtex-4 FX Devices. *Xilinx Application Notes*, 2006.

[27] Richard Griffith and Felix Pang. MicroBlaze System Performance Tuning. *Xilinx Application Notes*, (348), 2008.

[28] Steiner,Glenn. Code Acceleration with an APU Coprocessor:a Case Study of an LPM Algorithm. *Xilinx Application Notes*, (738), 2008.

[29] Zicari, Paolo and Corsonello, Pasquale and Perri, Stefania and Cocorullo, Giuseppe. A matrix product accelerator for field programmable systems on chip. *Microprocessors and Microsystems*, 32(2):53–67, 2008.

[30] Xu Guo and M. Gora. An Instruction Set Extension of the Virtex-5 PowerPC 440 for Elliptic Curve Cryptography. Technical report, dec 2008.

[31] Ramanathan, S. and Nandy, S.K. and Visvanathan, V. Reconfigurable Filter Coprocessor Architecture for DSP Applications. *Journal of VLSI signal processing systems for signal, image and video technology*, 26(3):333–359, 2000.

[32] Galanis, Michalis D. and Dimitroulakos, Gregory and Goutis, Costas E. Exploring the speedups of embedded microprocessor systems utilizing a high-performance coprocessor data-path. *J. Supercomput.*, 39(3):251–271, mar 2007.

[33] Wassner, J. and Zahn, K. and Dersch, U. Hardware-software codesign of a tightly-coupled coprocessor for video content analysis. In *IEEE Workshop on Signal Processing Systems (SIPS)*, pages 87–92, 2010.

[34] Mingas, Grigorios and Tsardoulias, Emmanouil and Petrou, Loukas. An FPGA implementation of the SMG-SLAM algorithm. *Microprocessors. Microsystems*, 36(3):190–204, may 2012.

[35] Vera, Alonzo and Meyer-Baese, Uwe and Pattichis, Marios. An FPGA-based rapid prototyping platform for wavelet coprocessors. In *Defense and Security Symposium*, pages 657615–657615. International Society for Optics and Photonics, 2007.

[36] Elhossini, Ahmed and Areibi, Shawki and Dony, Robert. An FPGA implementation of the LMS adaptive filter for audio processing. In *IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig)* , pages 1–8. IEEE, 2006.

[37] Wait, CD. IBM PowerPC 440 FPU with complex-arithmetic extensions. *IBM Journal of Research and Development*, 49(2.3):249–254, 2005.

[38] Dmitriy L. Bekker. Hardware and Software Optimization of Fourier Transform Infrared Spectrometry on Hybrid-FPGAs. Master's thesis, Department of Computer Engineering, Rochester Institute of Technology, Rochester, New York, USA, 2007.

[39] Ahsan Shabbir, Akash Kumar, Bart Mesman, and Henk Corporaal. Enabling MPSoC Design Space Exploration on FPGAs. In *Wireless Networks, Information Processing and Systems*, volume 20 of *Communications in Computer and Information Science*, pages 412–421. Springer Berlin Heidelberg, 2009.

[40] Fons Lluís, Francisco and others. *Embedded Electronic Systems Driven by Run-Time Reconfigurable Hardware* . PhD thesis, Department of Electrical Electronic Engineering and Automation, University Rovira i Virgili, Tarragona, Spain, 2012.

[41] Fons, Francisco and Fons, Mariano and Cantó, Enrique and López, Mariano. Deployment of Run-Time Reconfigurable Hardware Coprocessors Into Compute-Intensive Embedded Applications. *Journal of Signal Processing Systems*, 66(2):191–221, 2012.

[42] Chun-Hsian Huang and Pao-Ann Hsiung. Software-controlled dynamically swappable hardware design in partially reconfigurable systems. *EURASIP J. Embedded Syst.*, pages 4:1–4:11, jan 2008.

[43] David Pellerin and Scott Thibault. *Practical FPGA Programming in C.* Prentice Hall PTR, Massachusetts, USA, 2005.

[44] Rajanish K. Kamat and Santhosh A. Shinde and Vinod G. Shelake. *Unleash the System On Chip using FPGAs and Handel C.* Springer, Dordrecht Heidelberg, New York, 2010.

[45] Subramanian, Nikhil. A C-to-FPGA solution for accelerating tomographic reconstruction. Master's thesis, Department of Electrical Engineering, University of Washington, USA, 2009.

[46] Kunal Shenoy. Accelerating Software Applications Using the APU Controller and C-to-HDL Tools. *Xilinx Application Notes*, (901), 2005.

[47] Cong, Jason and Liu, Bin and Neuendorffer, Stephen and Noguera, Juanjo and Vissers, Kees and Zhang, Zhiru. High-level synthesis for FPGAs: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, 2011.

[48] Liu,Yanhong. *System-level modeling and analysis of multimedia-soc platforms*. PhD thesis, Institute of Computing Technology, National University of Singapore, 2007.

[49] Peesapati, Rangababu and Sabat, Samrat L. and K. P., Karthik and M., Narasimhappa and N., Giribabu and Nayak, J. FPGA-based embedded platform for fiber optic gyroscope signal denoising. *International Journal of Circuit Theory and Applications*, doi:10.1002/cta.1883, 2013.

[50] Balarin, Felice. *Hardware-software co-design of embedded systems: the POLIS approach*. Kluwer Academic Pub, 1997.

[51] Kienhuis, Bart and Deprettere, Ed and Vissers, Kees and Van Der Wolf, Pieter. An approach for quantitative analysis of application-specific dataflow architectures. In *Proceedings of International Conference on Application-Specific Systems, Architectures and Processors*, pages 338–349. IEEE, 1997.

[52] Fletcher, Bryan H. FPGA embedded processors. In *Embedded Systems Conference*, pages 1–18, 2005.

[53] Pei-Yin Chen and Ren-Der Chen and Yu-Pin Chang and Leang-san Shieh and Malki, H.A. Hardware Implementation for a Genetic Algorithm. *IEEE Transactions on Instrumentation and Measurement*, 57(4):699 –705, Apr 2008.

[54] Xilinx MicroBlaze Processor. http://www.xilinx.com/tools/microblaze.html [Accessed :21-07-2013].

[55] PowerPC 405 Processor Block Reference Guide. *Xilinx User guide*, (018), 2010.

[56] Virtex-5 FPGA Embedded Processor Block with PowerPC 440 Processor . *Xilinx Data sheet*, (621), 2011.

[57] Doug Amos and Austin Lesea and René Richter. *FPGA-Based Prototyping Methodology Manual Best Practices in Design-for-Prototyping.* Synopsys, Inc, Mountain View, CA, USA, 2010.

[58] EECG Toronto. http://www.eecg.toronto.edu/ pc/courses/432/ [Accessed :21-07-2013].

[59] Tue University. http://www.win.tue.nl/ wsinmak/Education/2IN35/lab/ [Accessed :21-07-2013].

[60] Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs. *Xilinx Application Notes*, (462), 2006.

[61] Lee, Tien-Lung and Bergmann, Neil W. Interfacing methodologies for IP re-use in reconfigurable system-on-chip. In *Microelectronics, MEMS, and Nanotechnology*, pages 454–463. International Society for Optics and Photonics, 2004.

[62] Justin Thiel. Splice: A Standardized Peripheral Logic and Interface Creation Engine. Master's thesis, Department of Computer Science Engineering, Washington University in St. Louis, USA, 2007.

[63] OPB IPIF Architecture. Technical Report 414, Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124-3400, 2004.

[64] Xilinx. Synthesis and Simulation Design Guide. *Xilinx User guide*, (626), 2006.

[65] Tutorial for FPGA based SoC system. http://www.fpgadeveloper.com/ [Accessed :21-07-2013].

[66] Jagannath Nayak. Fiber-optic gyroscopes: from design to production. *Appl. Opt.*, 50(25):E152–E161, sep 2011.

[67] H. Kajioka, H. Kumagai, T. Nakai, T Dohsho, H. Soekawa, and T Yuhara. Commercial applications of mass-produced fiber optic gyros. In *Proc.SPIE*, number 2837, pages 61–71, 1996.

[68] Samrat L. Sabat, P. Rangababu, K.P. Karthik, G. Krishnaprasad, and J. Nayak. System on chip implementation of 1-D Wavelet transform based

denoising of Fiber Optic Gyroscope signal on FPGA. In *Proceedings of the International Conference on Engineering Sustainable Solutions (INDICON)*, pages 1–5, dec 2011.

[69] Yanbo Li, Yu Liu, Baoku Su, and Yansong Jiang. Modified wavelet filtering algorithm applied to gyro servo technology for the improvement of test-precision. *Journal of Systems Engineering and Electronics*, 22(3):488 –492, jun 2011.

[70] Mao Ben, Wu Jun Wei, Wu Jian Tong, and Zhou Xue Mei. MEMS Gyro Denoising Based on Second Generation Wavelet Transform. In *Proceedings of the on 9th International Conference on Pervasive Computing Signal Processing and Applications (PCSPA)*, pages 255 –258, sep 2010.

[71] S.M. Paniit and Wibang Zhang. Modeling Random Gyro Drift Rate by Data Dependent Systems. *IEEE Transactions on Aerospace and Electronic Systems.*, AES-22(4):455 –460, jul 1986.

[72] Gannan Yuan, Haibo Liang, Kunpeng He, and Yanjun Xie. Research on signal de-noising technique for MEMS gyro. In *Proceedings of the 3rd International Symposium on Systems and Control in Aeronautics and Astronautics (ISSCAA)*, pages 1288 –1291, jun 2010.

[73] Kezhi Zhang, Weifeng Tian, and Feng Qian. A novel adaptive filter mechanism for improving the measurement accuracy of the fiber optic gyroscope in the maneuvering case. *Measurement Science and Technology*, 18(9):2777, 2007.

[74] Kai Xiong, Tang Liang, and Lei Yongjun. Multiple Model Kalman Filter for Attitude Determination of Precision Pointing Spacecraft. *Acta Astronautica*, 68(7-8):843–852, 2011.

[75] Liang Xue, Cheng-Yu Jiang, Hong-Long Chang, Yong Yang, Wei Qin, and Wei-Zheng Yuan. A novel Kalman filter for combining outputs of MEMS gyroscope array. *Measurement*, 45(4):745 – 754, 2012.

[76] Lu Di, Yao Yu, and He Fenghua. Sensor management based on cross-entropy in interacting multiple model Kalman filter. In *Proceedings of the American Control Conference*, volume 6, pages 5381 –5386, 2004.

[77] Cezary Kownacki. Optimization approach to adapt Kalman filters for the real-time application of accelerometer and gyroscope signals filtering. *Digital Signal Processing*, 21(1):131 – 140, 2011.

[78] A.H. Mohamed and K.P Schwarz. Adaptive Kalman Filtering for INS/GPS. *Journal of Geodesy*, 73:193–203, 1999.

[79] Sameh Nassar and Naser El Sheimy. A combined algorithm of improving INS error modeling and sensor measurements for accurate INS/GPS navigation. *GPS Solutions*, 10:29–39, 2006.

[80] Sergio Baselga, Luis Garca Asenjo, Pascual Garrigues, and José Luis Lerma. Inertial Navigation System Data Filtering Prior to GPS/INS Integration. *The Journal of Navigation*, 62(04):711–720, 2009.

[81] Allan, D.W. and Barnes, J.A. A Modified Allan Variance with Increased Oscillator Characterization Ability. In *Proceedings of 35th Annu.Freq.Control Symp*, pages 470–475, Baltimore, MD, USA, nov 1981.

[82] M. Vetterli. Wavelets, approximation, and compression. *IEEE, Signal Processing Magazine*, 18(5):59 –73, sep 2001.

[83] S.G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on, Pattern Analysis and Machine Intelligence*, 11(7):674 –693, jul 1989.

[84] Ingrid Daubechies. *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[85] S.L. Sabat, N. Giribabu, J. Nayak, and K. Krishnaprasad. Characterization of Fiber Optics Gyro and Noise Compensation Using Discrete Wavelet Transform. In *Proceedings of the 2nd International Conference on Emerging Trends in Engineering and Technology (ICETET)*, pages 909 –913, dec. 2009.

[86] Yuanxi Yang. Adaptively Robust Kalman Filters with Applications in Navigation. In *Sciences of Geodesy - I*, pages 49–82. Springer Berlin Heidelberg, 2010.

[87] Mehra, R. On the identification of variances and adaptive Kalman filtering. *IEEE Transactions on Automatic Control*, 15(2):175–184, apr 1970.

[88] Dah-Jing Jwo and Ta-Shun Cho. A practical note on evaluating kalman filter performance optimality and degradation. *Applied Mathematics and Computation*, 193(2):482 – 505, 2007.

[89] Rangababu Peesapati and Samrat L. Sabat and Kiran Kumar Anumandla and Palani Karthik Kandyala and Jagannath Nayak. Design and implementation of a realtime co-processor for denoising Fiber Optic Gyroscope signal. *Digital Signal Processing*, doi: 10.1016/j.dsp.2013.04.010, 2013.

[90] Dah-Jing Jwo, Shun-Chieh Chang. Particle swarm optimization for GPS navigation Kalman filter adaptation. *Aircraft Engineering and Aerospace Technology*, 81(4):343– 352, 2009.

[91] I. Zurbenko, P. S. Porter, R. Gui, S. T. Rao, J. Y. Ku, and R. E. Eskridge. Detecting Discontinuities in Time Series of Upper-Air Data: Development and Demonstration of an Adaptive filter technique. *Journal of Climate*, 9(12):3548–3560, 1996.

[92] Mahmoud Lotfy ElGizawy. *Continuous Measurement-While-Drilling Surveying system Utilizing MEMS Inertial Sensors*. PhD thesis, Department of Geomatics Engineering, University of Calgary, Alberta, 2009.

[93] K.P. Karthik, P. Rangababu, Samrat L. Sabat, and J. Nayak. System on Chip Implementation of Adaptive Moving Average Based Multiple-Model Kalman Filter for Denoising Fiber Optic Gyroscope Signal. In *Proceedings of the International Symposium on Electronic System Design (ISED)*, pages 170 –175, dec 2011.

[94] Rangababu Peesapati and Samrat L. Sabat and K.P. Karthik and J. Nayak and N. Giribabu. Efficient hybrid Kalman filter for denoising fiber optic gyroscope signal. *Optik - International Journal for Light and Electron Optics*, doi:10.1016/j.ijleo.2013.02.013, 2013.

[95] Bahoura, Mohammed and Ezzaidi, Hassan. FPGA-implementation of discrete wavelet transform with application to signal denoising. *Circuits, Systems, and Signal Processing*, 31(3):987–1015, 2012.

[96] Rui Guo and L.S. DeBrunner. Two High-Performance Adaptive Filter Implementation Schemes Using Distributed Arithmetic. *IEEE Transactions on Circuits and Systems*, 58(9):600 –604, sept 2011.

[97] Xilinx. System Generator for DSP . *Xilinx User guide*, (10.1), 2008.

[98] Gate Count Capacity Metrics for FPGAs. *Xilinx Application Notes*, (059), 1997.

[99] Ali, layak. *Particle Swarm Optimization techniques for solving numerical and engineering optimization problems*. PhD thesis, School of Physics, University of Hyderabad, India, 500046, 2012.

[100] J. Hay and K.K. Loo. Fast motion estimation using evolutionary strategy search algorithm. In *International Conference on Digital Telecommunications, ICDT '06.*, page 16, aug 2006.

[101] Shing Tai Pan. Evolutionary computation on programmable robust iir filter pole-placement design. *IEEE Transactions on Instrumentation and Measurement*, 60(4):1469 –1479, april 2011.

[102] Lukáš Sekanina. From implementations to a general concept of evolvable machines. In *Proceedings of the 6th European conference on Genetic programming*, EuroGP'03, pages 424–433, 2003.

[103] Anumandla, KiranKumar and Peesapati, Rangababu and Sabat, SamratL. and Udgata, SibaK. and Abraham, Ajith. Field programmable gate arrays based differential evolution coprocessor: a case study of spectrum allocation in cognitive radio network. *IET Computers & Digital Techniques*, doi:10.1049/iet-cdt.2012.0109:1–14, 2013.

[104] Shih-An Li, Chen-Chien Hsu, Ching-Chang Wong, and Chia-Jun Yu. Hardware/software co-design for particle swarm optimization algorithm. *Information Sciences*, 181(20):4582–4596, oct 2011.

[105] A. Swarnalatha and A.P. Shanthi. Optimization of single variable functions using complete hardware evolution. *Applied Soft Computing*, 12(4):1322 – 1329, 2012.

[106] Pradeep R. Fernando, Srinivas Katkoori, Didier Keymeulen, Ricardo Zebulum, and Adrian Stoica. Customizable FPGA IP core Implementation of a General-Purpose Genetic Algorithm Engine. *IEEE Transactions on Evolutionary Computation*, 14(1):133–149, feb 2010.

[107] Farmahini-Farahani, Amin and Vakili, Shervin and Fakhraie, Sied Mehdi and Safari, Saeed and Lucas, Caro. Parallel scalable hardware implementation of asynchronous discrete particle swarm optimization. *Engineering Applications of Artificial Intelligence*, 23(2):177–187, mar 2010.

[108] Girma S. Tewolde and Darrin M. Hanna and Richard E. Haskell. A modular and efficient hardware architecture for particle swarm optimization algorithm. *Microprocessors and Microsystems*, 36(4):289 – 302, 2012.

[109] Rogério M. Calazan and Nadia Nedjah and Luiza M. Mourelle. A hardware accelerator for Particle Swarm Optimization. *Applied Soft Computing*, doi:10.1016/j.asoc.2012.12.034, 2013.

[110] Tirumalai, Vijay and Ricks, Kenneth G. and Woodbury, Keith A. Using parallelization and hardware concurrency to improve the performance of a genetic algorithm. *Concurrency and Computation: Practice and Experience*, 19(4):443–462, 2007.

[111] Gomez-Pulido, Juan A and Vega-Rodriguez, Miguel A and Sanchez-Perez, Juan M and Priem-Mendes, Silvio and Carreira, Vitor. Accelerating floating-point fitness functions in evolutionary algorithms: a FPGA-CPU-GPU performance comparison. *Genetic Programming and Evolvable Machines*, 12(4):403–427, 2011.

[112] Munoz, D.M. and Llanos, C.H. and Coelho, L.D.S. and Ayala-Rincon, M. Hardware Architecture for Particle Swarm Optimization using Floating-point Arithmetic. In *Proceedings of the Ninth International Conference on Intelligent Systems Design and Applications*, pages 243 –248, dec 2009.

[113] Muńoz, D.M. and Llanos, C.H. and Coelho, L.D.S. and Ayala-Rincon, M. Comparison between two FPGA implementations of the Particle Swarm Optimization algorithm for high-performance embedded applications. In *In the Proceedings of Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pages 1637–1645, 2010.

[114] Munoz, D.M. and Llanos, C.H. and Coelho, L.D.S. and Ayala-Rincon, M. Hardware Particle Swarm Optimization Based on the Attractive-Repulsive Scheme for Embedded Applications. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, pages 55 –60, dec 2010.

[115] Vašíček, Zdeněk and Sekanina, Lukáš. Hardware accelerator of cartesian genetic programming with multiple fitness units. *Computing and Informatics*, 29(6):1359–1371, 2012.

[116] Fábio Fabris and Renato A. Krohling. A co-evolutionary differential evolution algorithm for solving minmax optimization problems implemented on GPU using C-CUDA. *Expert Systems with Applications*, 39(12):10324 – 10333, 2012.

[117] Qin, A. K. and Raimondo, Federico and Forbes, Florence and Ong, Yew Soon. An improved CUDA-based implementation of differential evolution on GPU. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation*, GECCO '12', pages 991–998, 2012.

[118] Ng, S.C. and Leung, S.H. and Chung, C.Y. and Luk, A. and Lau, W.H. The genetic search approach. A new learning algorithm for adaptive IIR filtering. *IEEE Transactions on Signal Processing*, 13(6):38 –46, nov 1996.

[119] Ganapati Panda and Pyari Mohan Pradhan and Babita Majhi. IIR system identification using cat swarm optimization. *Expert Systems with Applications*, 38(10):12671 – 12683, 2011.

[120] Lipika Gupta and Rajesh Mehra. Modified PSO based Adaptive IIR Filter Design for System Identification on FPGA. *International Journal of Computer Applications*, 22(5):1–7, may 2011.

[121] Karaboga, Nurhan. Digital IIR Filter Design Using Differential Evolution Algorithm. *EURASIP Journal on Applied Signal Processing*, 2005:1269–1276, jan 2005.

[122] Wu, Fang Jian-an, Tang Yang, Zhang Wenbing and Du Wei Zhu. Digital IIR Filters Design Using Differential Evolution Algorithm with a Controllable Probabilistic Population Size. *PLoS ONE*, 7(7):e40549, jul 2012.

[123] Storn, Rainer and Price, Kenneth. Differential Evolution A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, dec 1997.

[124] Sum-Im, T and Taylor, GA and Irving, MR and Song, YH. Differential evolution algorithm for static and multistage transmission expansion planning. *IET Generation, Transmission & Distribution*, 3(4):365–384, 2009.

[125] Vesterstrom, J. and Thomsen, R. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Congress on Evolutionary Computation*, volume 2, pages 1980 – 1987, June 2004.

[126] Suganthan, P. N. and Hansen, N. and Liang, J. J. and Deb, K. and Chen, Y. P. and Auger, A. and Tiwari, S. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical report, Nanyang Technological University, Singapore, 2005.

[127] Ke Tang, Xiaodong Li, Ponnuthurai Nagaratnam Suganthan, Zhenyu Yang, and Thomas Weise. Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization. Technical report, University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL): China, 2010.

[128] Floating Point Operator v5.0. Technical report, Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124-3400, March 2011.

[129] S. Das and P.N. Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 60(4):1469–1479, apr 2011.

[130] M.B. Yeary and N.C. Griswold. Adaptive IIR filter design for single sensor applications. *IEEE Transactions on Instrumentation and Measurement*, 51(2):259 –267, apr 2002.

[131] Liu, Ying-Hong and Kuo, Chih-Yu and Chang, Chien C. and Wang, Chang-Yi. Electro-cosmotic flow through a two-dimensional screen-pump filter. *Phys. Rev. E*, 84:036301, sep 2011.

[132] Tang, Y. and Gao, H. and Kurths, J. and Fang, J.-A. Evolutionary Pinning Control and Its Application in UAV Coordination. *IEEE Transactions on Industrial Informatics*, 8(4):828 –838, nov 2012.

[133] A new design method based on artificial bee colony algorithm for digital IIR filters. *Journal of the Franklin Institute*, 346(4):328 – 348, 2009.

[134] T. Wiegand, G. Sullivan, and A. JVT(eds) Luthra. Draft (ITU-T) recommendation and final draft international standard of joint video specification (ITU-T Rec.H.264 ISO/IEC 14496-10 AVC). JVT-G050r1, 2003.

[135] G.J. Sullivan and T. Wiegand. Video Compression - From Concepts to the H.264/AVC standard. *Proceedings of IEEE Conference on*, 93(1):18–31, jan 2005.

[136] D. Marpe, T. Wiegand, and G.J. Sullivan. The H.264/MPEG4 advanced video coding standard and its applications. *IEEE Communications Magazine*, 44(8):134–143, aug 2006.

[137] Iain E Richardson. *The H.264 Advanced Video Compression Standard; 2nd ed.* John Wiley and Sons Ltd, 2010.

[138] Micheal C.Brogioli. *Reconfigurable heterogeneous DSP/FPGA Based Embedded Architectures for Numerically Intensive Computing Workloads.* PhD thesis, Department of Electrical and Computer Engineering, Rice University, Kingston, Houston, Texas, 2007.

[139] Lin, Chien-Chang. *An Efficient Architecture for H.264 Video Decoder.* PhD thesis, Department of Information Engineering, National Chung Cheng University, 2007.

[140] Ke, Xu and Chiu-Sing Choy. Low-power H.264/AVC baseline decoder for portable applications. In *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 256 –261, aug 2007.

[141] Ke, Xu and Zhang, Min and Choy, Chiu. Design a Low-Power H.264/AVC Baseline Decoder at All Abstraction LevelsA Showcase. *Journal of Signal Processing Systems*, 67:317–330, 2012.

[142] Available on: http://iphome. hhi. de/suehring/tml/9.8/. JM Software.

[143] Lindroth, T. and Avessta, N. and Teuhola, J. and Seceleanu, T. Complexity Analysis of H.264 Decoder for FPGA Design. In *Proceedings of the IEEE International Conference on Multimedia and symposium*, pages 1253 –1256, jul 2006.

[144] Profiling Guide - A Guide to Profiling in EDK. *Xilinx User guide*, (448), 2010.

[145] Ostermann, Jörn and Bormans, Jan and List, Peter and Marpe, Detlev and Narroschke, Matthias and Pereira, Fernando and Stockhammer, Thomas and Wedi, Thomas. Video coding with H. 264/AVC: tools, performance, and complexity. *Circuits and Systems magazine*, 4(1):7–28, 2004.

[146] Kthiri, Moez and Kadionik, Patrice and Le Gal, B and Levi, H and Ben Atitallah, A. Performances analysis and evaluation of Xenomai with a H. 264/AVC decoder. In *International Conference on Microelectronics (ICM)* , pages 1–4. IEEE, 2011.

[147] Tung-Chien Chen and Chung-Jr Lian and Liang-Gee Chen. Hardware architecture design of an H.264/AVC video codec. In *Proceedings of the International Conference on Design Automation, Asia and South Pacific*, page 8, jan 2006.

[148] Agostini, Luciano V. and Azevedo Filho, Arnaldo P. and Staehler, Wagston T. and Rosa, Vagner S. and Zatt, Bruno and Pinto, Ana Cristina M. and Porto, Roger Endrigo and Bampi, Sergio and Susin, Altamiro A. Design and FPGA prototyping of a H264/AVC main profile decoder for HDTV. *Journal of the Brazilian Computer Society*, 12:25–36, mar 2007.

[149] Bonatto, Alexsandro C. and Soares, André B. and Renner, Adriano and Susin, Altamiro A. and Silva, Leandro Max and Bampi, Sergio. A 720p H.264/AVC decoder ASIC implementation for digital television set-top boxes. In *Proceedings of the 23rd symposium on Integrated circuits and system design*, (SBCCI), pages 168–173, New York, NY, USA, 2010.

[150] Rosa, Vagner S and Staehler, Wagston T and Azevedo, Arnaldo and Zatt, Bruno and Porto, Roger E and Agostini, Luciano Volcan and Bampi, Sergio and Susin, Altamiro Amadeu. FPGA Prototyping Strategy for a H.

264/AVC Video Decoder. In *Proceedings of the 18th IEEE/IFIP International Workshop on Rapid System Prototyping*, pages 174–180. IEEE, 2007.

[151] Leanardo Max De Lima Silva. Physical Implementation of Hardware Architectures for Digital Video Decoding According to the H.264/AVC Standard. Master's thesis, Department of Computer science, Federal University of Rio Grande do Sul,Brazil, 2010.

[152] Roszkowski, Mikoĺaj and Abramowski, Andrzej and Wieczorek, Michaĺ and Pastuszak, Grzegorz. Architecture design of the hardware h.264/avc video decoder. *International Journal of Electronics and Telecommunications*, 56(3):291–300, 2010.

[153] Ke Xu. H.264 Baseline profile decoder. *http://opencores.org/project,nova*, (nova. Rev13), 2009.

[154] Warsaw, Thomas and Lukowiak, Marcin. Architecture design of an H.264/AVC decoder for real-time FPGA implementation. In *Proceedings of the IEEE 17th International Conference on Application-specific Systems, Architectures and Processors*, (ASAP), pages 253–256, Washington, DC, USA, 2006.

[155] Tung-Chien Chen, Chung-Jr Lian, and Liang-Gee Chen. Hardware architecture design of an H.264/AVC video codec. In *Proceedings of the Design Automation Conference Asia and South Pacific*, (ASP-DAC), pages 750–757. IEEE Press, 2006.

[156] Werda, Imen and Dammak, Taheni and Grandpierre, Thierry and Ayed, Mohamed Ali Ben and Masmoudi, Nouri. Real-time H. 264/AVC baseline decoder implementation on TMS320C6416. *Journal of Real-Time Image Processing*, 7(4):215–232, 2012.

[157] Huan-Kai Peng and Chun-Hsin Lee and Jian-Wen Chen and Tzu-Jen Lo and Yung-Hung Chang and Sheng-sung Hsu and Yuan-Chun Lin and Ping Chao and Wei-Cheng Hung and Kai-Yuan Jan. A Highly Integrated 8mW H.264/AVC Main Profile Real-time CIF Video Decoder on a 16MHz SoC Platform. In *Design Automation Conference,ASP-DAC '07. Asia and South Pacific*, pages 112 –113, jan 2007.

[158] B. Stabernack, K.-I. Wels, and H. Hubert. A System on a Chip Architecture of an H.264/AVC Coprocessor for DVB-H and DMB Applications. *IEEE Transactions on Consumer Electronics*, 53(4):1529 –1536, nov 2007.

[159] Hristo Nikolov, Todor Stefanov, and Ed F. Deprettere. Automated Integration of Dedicated Hardwired IP Cores in Heterogeneous MPSoCs Designed with ESPAM. *EURASIP J. Emb. Sys.*, pages 1–16, 2008.

[160] Hristo Nikolov, Adarsha Rao, Ed F. Deprettere, S. K. Nandy, and Ranjani Narayan. A H.264 decoder: a design style comparison case study. In *Proceedings of the 43rd Asilomar conference on Signals, systems and computers*, Asilomar, pages 236–242. IEEE Press, 2009.

[161] Adarsha Rao, S. K. Nandy, Hristo Nikolov, and Ed F. Deprettere. USHA: Unified software and hardware architecture for video decoding. *Symposium on Application Specific Processors*, 0:30–37, 2011.

[162] Dajiang Zhou, Jinjia Zhou, Xun He, Jiayi Zhu, Ji Kong, Peilin Liu, and S. Goto. A 530 Mpixels/s 4096x2160@60fps H.264/AVC High Profile Video Decoder Chip. *IEEE Journal of Solid-State Circuits,*, 46(4):777 –788, apr 2011.

[163] Y. Moshe and N. Peleg. Implementations of H.264/AVC baseline decoder on different digital signal processors. In *Proceedings of the 47th International Symposium ELMAR*, pages 37–40, jun 2005.

[164] Ye Xien, Zhou Haiyong, and Tao Weijiong. Real-time H.264/AVC Decoder Implementation on PXA270. In *Proceedings of International Conference on Communications, Circuits and Systems (ICCCAS)*, pages 819 –821, jul 2007.

[165] Soares, André Borin and Bonatto, Alexsandro Cristóvão and Susin, Altamiro Amadeu. Development of a SoC for Digital Television Set-Top Box: Architecture and System Integration Issues. *International Journal of Reconfigurable Computing*, 2013.

[166] Ke, Xu and Chiu-Sing Choy. A Five-Stage Pipeline, 204 Cycles/MB, Single-Port SRAM-Based Deblocking Filter for H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology* , 18(3):363 –374, mar 2008.

[167] Ke, Xu and Chiu-Sing Choy. A Power-Efficient and Self-Adaptive Prediction Engine for H.264/AVC Decoding. *IEEE Transactions on,Very Large Scale Integration (VLSI) Systems*, 16(3):302 –313, mar 2008.

[168] Ke, Xu and Liu, Tsu-Ming and Guo, Jiun-In and Choy, Chiu-Sing. Methods for Power/Throughput/Area Optimization of H.264/AVC Decoding. *Journal of Signal Processing Systems*, 60:131–145, 2010.

[169] Ke, Xu and Chiu-Sing Choy and Cheong-Fat Chan and Kong-Pang Pun. Power-Efficient VLSI Realization of a Complex FSM for H.264/AVC Bitstream Parsing. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 54(11):984 –988, nov 2007.

[170] Ke, Xu and Choy, Chiu-Sing and Chan, Cheong-Fat and Pun, Kong-Pang. Priority-based heading one detector in H.264/AVC decoding. *EURASIP J. Embedded Syst.*, 2007(1):18–18, jan 2007.

[171] DVI, VGA, and Component Video Demonstration . *Xilinx User guide*, (248), 2006.