IMPROVING PERFORMANCE OF QUALITY PROGRAMMABLE VECTOR PROCESSOR USING DIFFERENT OPTIMIZATIONS

A Project Report

submitted by

DEBPRATIM ADAK

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **IMPROVING PERFORMANCE OF QUALITY PROGRAMMABLE VECTOR PROCESSOR USING DIFFERENT OPTIMIZATIONS** , submitted by **DEBPRATIM ADAK**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr V. KAMAKOTI
Research Guide
Professor
Department of Computer science and engineering
IIT-Madras, 600 036

Place: Chennai

Date: 8th May 2017

ACKNOWLEDGEMENTS

I would like to take the opportunity to reflect on the brilliant people who enabled me to accomplish this with their invaluable guidance, support and motivation.

Foremost, I would like to express my sincere gratitude to my guide, Dr. V. Kamakoti whose patience, enthusiasm and immense knowledge has inspired me to work efficiently on this project. I also thank him for allowing me freedom and exibility while working on the project.

Many thanks to my co-guide Dr.Nitin Chandrachoodan and faculty advisor Dr.Shreepad Karmalkar, for their guidance, encouragement and insightful inputs.

My deepest gratitude to Neel Gala who has been more than supportive. He has enriched the project experience with his active participation and invaluable suggestions.

Last but not the least many thanks to my fellow lab-mates Zaid, Vishvesh, Vinod and Arjun for their consistent help and support.

ABSTRACT

Applications with significant data level parallelism and higher computation requirement use vector processor for faster execution. A vector processor has large number of processing elements which operate concurrently and this leads to higher power consumption. Inexact calculation of these applications can reduce power consumption. QUORA , a quality programmable vector processor has the ability to evaluate accuracy level of a given application and scale the input operand accordingly. Instruction of QUORA contains quality field which is used to scale input operands. Architecture of QUORA is designed to have higher performance for these specific applications. In this thesis I have discussed basic architecture of QUORA.

To achieve better performance I have used some optimizations which are instruction level parallelism, on chip multi bank memory and pipelined multiplier. Design challenges and modifications in architecture for inclusion of these optimizations in QUORA have been studied in this thesis. And finally performance achieved by having different optimization in QUORA is analyzed.

Contents

List of Tables

List of Figures

ABBREVIATIONS

BSV	Bluespec System Verilog
FIFO	First In First Out
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Language
ILP	Instruction-Level Parallelism
DCT	Discrete Cosine Transform
IDCT	Inverse Discrete Cosine Transform

Chapter 1

Introduction

Nowadays one of the critical issues in processor design is reducing power consumption. Emerging applications that operates on huge amount of data prefers vector architecture. In a vector architecture a single instruction operates on multiple data that are stored in register files and these operations are independent of each other. A vector processor can have multiple processing units and each can operates on different vector registers at the same time. Execution of a loop without any dependences between iterations is much faster in vector processor. One of the key aspect of a application for preferring vector architecture is data level parallelism.

Since multiple processing units in a vector processor operate together power consumption has become more. Power consumption can be reduced by making some lsb bits of the input operands zero. Applications state how much error it can accept and input operands are approximated according to that. Quality programmable vector processor can program the quality of the output by approximating its operands. QUORA is an example of quality programmable vector processor.

To improve performance of QUORA three optimization techniques have been exploited. They are instruction level parallelism, on chip multi bank memory and pipelined multiplier.

In QUORA next instruction is not fetched until execution of current instruction is over. Execution of instruction that operates on vector data usually takes multiple cycle and execution of next instruction can be started while current instruction is in execution state if there is no conflict and in this way ILP can be achieved in QUORA. In case of conflict program counter is stalled. QUORA has multiple fifo buffers and all of those need to be loaded before 2D vector reduction operation can start. Time required to load all of these fifos is very high and to reduce it multi bank memory is put with the processor. These banks can be operated concurrently to reduce the time to load fifo buffers.Multiplier lies in the critical path of the design. To increase frequency of operation multiplier is pipelined. Only one stage pipelined is discussed in this thesis.

Chapter 2

QUORA architecture

2.1 Overview

In this section complete architecture of a Quality Programmable Vector Processor (QUORA) is discussed.In research paper [1] some specific applications with resilience towards approximate computing are found. Applications that can accept approximated results are recognition, mining, synthesis, video processing, search etc. It is found that these application have lots of matrix-matrix operation and matrix-vector operations. Applications with this property have significant data-level- parallelism. So a vector architecture is needed for QUORA.

QUORA consists of one 2D array of processing elements called APE (Approximate Processing Element) , two 1D array of processing elements called MAPE (Mixed Accuracy Processing Element) located at the right and bottom border of the APE array and another processing element called CAPE (Completely Accurate Processing Element). Both APE array and MAPE array operate on data vectors. Matrix-matrix and matrix-vector operations are executed by APE array. MAPE array performs reduction operations , SIMD ops etc. CAPE are used for scalar instructions like load-store of a register, branch instructions etc.

Instruction set of QUORA contains specific instructions that operate on 2D data streams and 1D data streams stored in streaming memory bank. These instructions takes multiple cycles to execute. Multiple processing elements are involved during execution of an instruction of this category. It is obvious that energy dissipated during execution of these instruction is higher than the execution of other instructions. Input data vectors can be approximated for these instructions in favour of energy saving. These instruction have quality field that dictates desired level of accuracy of the output and input data vectors are scaled accordingly. Quality control unit of QUORA translates quality field to precision control value for input operands according to the functionality of the instruction. Approximation of operand takes place in psc_unit. Psc_units take operands from streaming memory and bitwise AND operation is performed between operand and precision control value to generate new operand.

Decode unit of QUORA performs instruction fetch and instruction decode. Instruction width of QUORA is either 64 bits or 96 bits. Since bus width is 32 bits, multiple cycles

is required for decode unit to fetch and decode an instruction completely. Until execution of an instruction is complete, program counter is stalled. This means program counter is stalled for multiple cycle while executing a vector instruction.

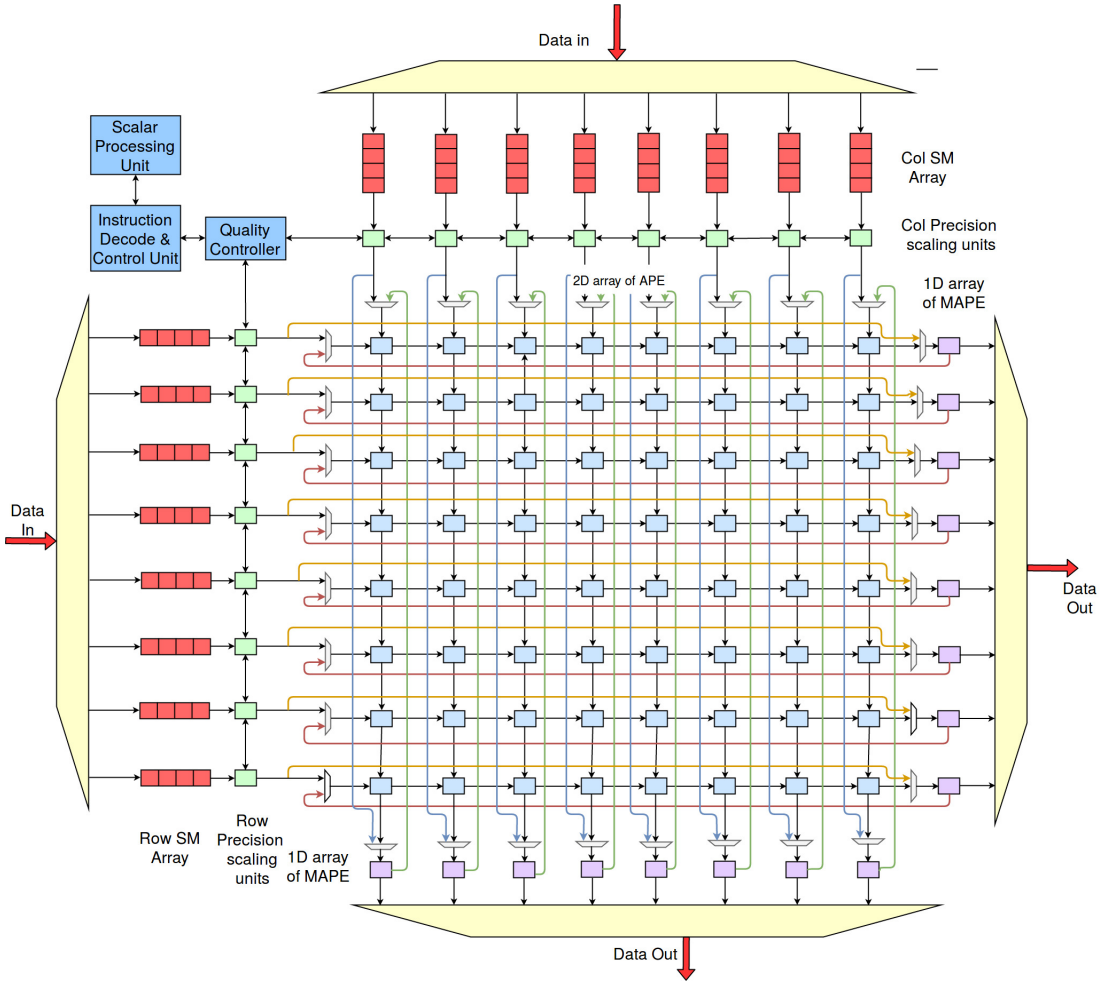

Figure 2.1: QUORA architecture

2.2 Instruction Types

Instruction set of QUORA is divided in 6 categories. These catagories are described below.

Table 2.1: Description of different instructions

Instruction type	Operation	Instruction format	Description

Scalar instruction	Load data
	Arithmetic and logical operations
	Branch
		LDRI Rname,value
		ADDRI RD,RS,value
		
		BGZDI Rname,Rel.addr ,value
			RISC type of instruction, Instruction width-64bits

Streaming memory instruction	Load data from memory to streaming memory	LDSM length,stride,burst, start_addr,row_en,col_en	Data is read in burst fashion from the memory. First address of the next burst is stride number of location apart from the last address of previous burst. row_en and col_en are the enable signals for streaming memory.
2D array reduction instruction	APE array operates on the data present in streaming memory	qpMAC length row_pe_en ,col_pe_en,q_type,q_value	row_pe_en and col_pe_en are the enable signals for the APE array. q_type and q_value specify error type and error value respectively
1D array reduction instruction	MAPE array operates on data present in either the streaming memory or the APE array	qpACC length row_pe_en ,col_pe_en ,q_type q_value qpMAX <row/col> sreg, pe_en	row_pe_en and col_pe_en are the enable signal for MAPE array Data in accumulator of APE is streamed into the MAPE. <row/col> decides whether it is a row scan or column scan.Pe_en specifies which MAPEs are active.
1D-array Self Operand Instructions	It operates on mask,sreg,accumulator located in MAPE	qpLFSTO <row/col>, pe_en,shift	In the example instruction shift is the immediate operand
Store Instruction	It stores data present in the accumulator of APE or MAPE	STR <row/col>,stride burst,start_address,pe_en	Fields in this instruction are already discussed in other examples

2.3 Modules in QUORA

- Processing elements
  - Approximate Processing Element
  - Mixed Accuracy Processing Element
  - Completely Accurate Processing Element
- Streaming memory banks
- Decode unit
- Quality control unit
- Precision scaling unit

In this section these modules are described in details.

2.4 Processing Elements

QUORA has three types of processing elements. They operates on vector or scalar data and results are stored in either the accumulator of the processing element or the register in Reg file.Each processing element targets operations with different aspect.

2.4.1 Approximate Processing Element

Approximate Processing Element or APE operates on data vectors which are stored in streaming memory bank. QUORA uses multiple APEs which are arranged in 2D array. Target operations for this block are Matrix-matrix and matrix-vector operations.

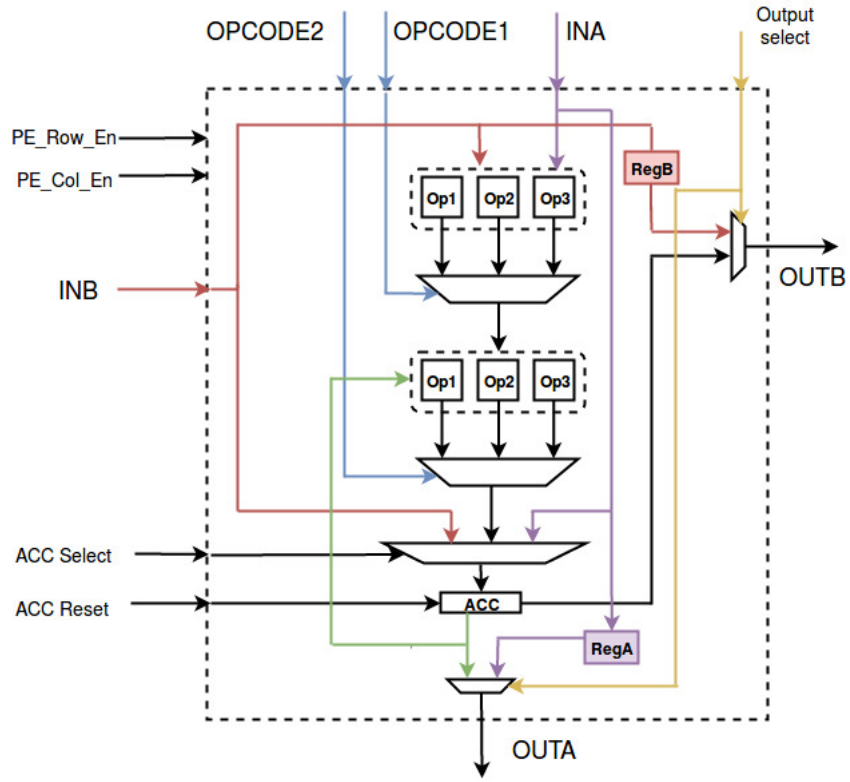

Figure 2.2: Approximate Processing Element

Interface

1. Input interface
   - Control signals
     - OPCODE1
     - OPCODE2
     - ACC_select
     - ACC_Reset
     - Output_Select
   - Data Inputs
     - InputA
     - InputB
2. Output interface
   - Data Outputs
     - OutputA
     - OutputB

Micro-architecture of an APE

InputA and InputB come from top and left side respectively. Each APE has two level of operation. OPCODE1 is responsible for first level and OPCODE2 is responsible for second level of operation. Result is stored in the accumulator. A multiplexer is used to select input for the accumulator. Inputs to this mux are second level output, InputA,InputB and ACC_select signal selects one of the inputs. Accumulator is reset if ACC_Reset signal is high.
Each APE has 3 registers. They are accumulator, RegA, RegB. RegA stores InputA and RegB stores InputB.
Each APE has two outputs. OutputA and OutputB are at the bottom and the right respectively. output data is selected by a multiplexer with select signal Output Select. Mux for outputA has two inputs and they are RegA and accumulator.Mux for outputB has two inputs and they are RegB and accumulator. For 2D reduction instruction RegA and RegB are chosen for the outputs.

Interconnection between APEs

APEs are interconnected in a systolic fashion. InputA of the APEs located at the top boundary, receive data from either the MAPE which is located at the same column or the column streaming memory. InputB of the APEs located at the left boundary get data from either from the MAPE which is located at the same row or from the Row streaming memory. Data comes from streaming memory for 2D reduction instruction and from MAPE for 1D reduction or store instruction. For 1D reduction or store instruction data is scanned along the row or column. If its a row scan data in APE array will move from left to right and for column scan data will move from top to bottom.

APE Enable Signals

PE_Row_En and PE_Col_En are two enable signals for a single APE. For column scan PE_Col_En should be true and for row scan PE_Row_En must be true. In case of 2D reduction instruction both PE_Row_En and PE_Col_En should be high since data in APE proceeds along the row and column simultaneously.

2.4.2 Mixed Accuracy Processing Element

QUORA has two 1D MAPE array block which are located at the bottom and the right side of 2D APE array block. This block is used to perform 1D-array Reduction Operations, 1D-array Self Operand Operations, Store Operations.

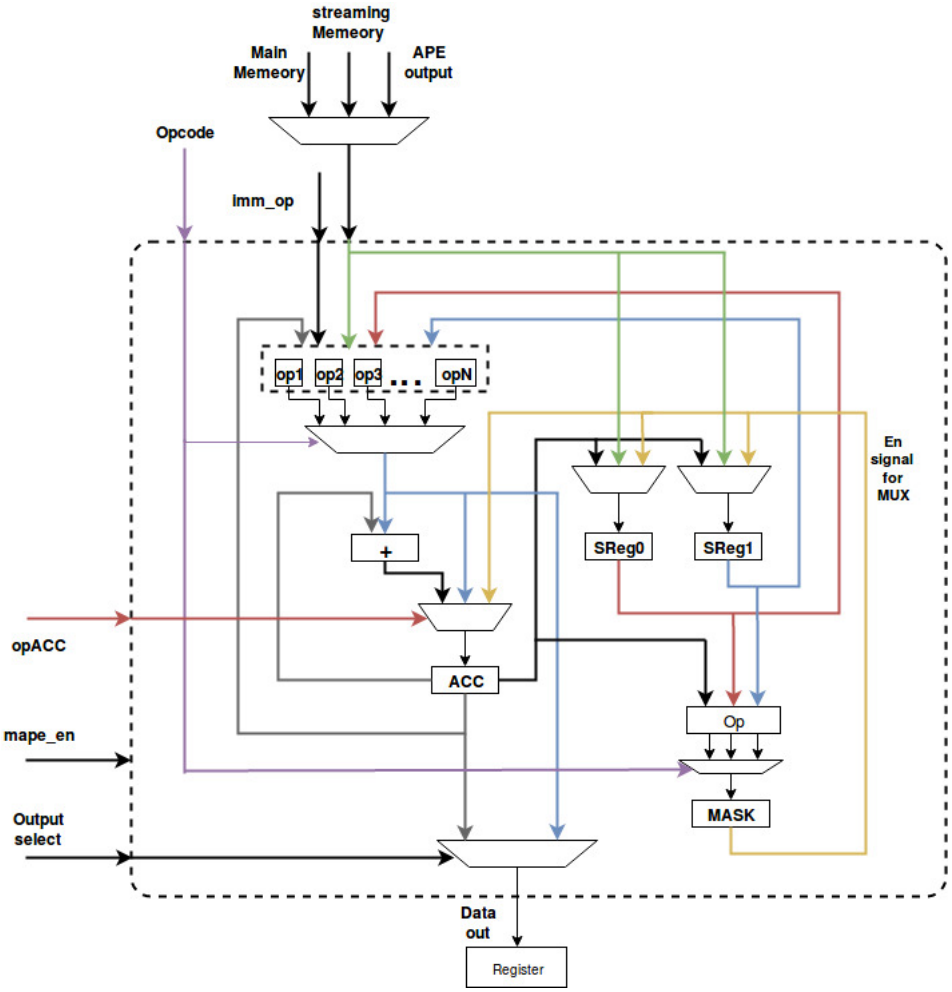

Figure 2.3: Mixed Accuracy Processing Element

Interface

1. Input Interface
   - Control signals
     - opACC
     - Output Select
     - Opcode
     - mape_en
   - Data inputs

  - opA
  - imm_op

2. Output Interface
   - Data out

Micro-architecture of a MAPE

MAPE has four registers which are accumulator, Scratch registers and mask. In this architecture one two scratch registers are kept in MAPE and they are denoted as SReg. SReg is loaded either from accumulator or from data input. QUORA isa has instructions that writes into mask register. Any operation on SReg or accumulator is valid if mask register is set.
Opcode decides inputs for accumulator, SReg and mask. MAPE has one output. Output is stored in a register located outside MAPE. Output data is chosen between accumulator and ALU output using a multiplexer whose select signal is Output Select. Each MAPE has two inputs and they are imm_op and opA. Input opA is received from a multiplexer whose inputs are streaming memory,main memory and APE output. Imm_op is the immediate operand present in the instructions.
A particular MAPE is enabled if mape_en is true.

2.4.3 Completely Accurate Processing Element

QUORA has only one CAPE. Scalar instructions are executed in this module. This unit contains one register file. Scalar instruction fetches operand from register file , operates on it and results are written back to the registers.It also executes branch instruction and set branch control signal if branch is true.Halt, load from memory,store to memory,branch are some of the example operations that are computed in this unit.

2.5 Streaming memory bank

QUORA has 2 streaming memory banks located at top and left boundary. Each bank contains multiple FIFO buffers. Streaming memory instructions are used to load this buffers.

2.6 Quality control unit

Quality control unit translates error field in instructions to precision control value. This translation procedures is well documented in research paper [2]. After execution of a vector instruction maximum possible error for the output that might occur due to approximation of input data vectors , is stored in register Final_err.

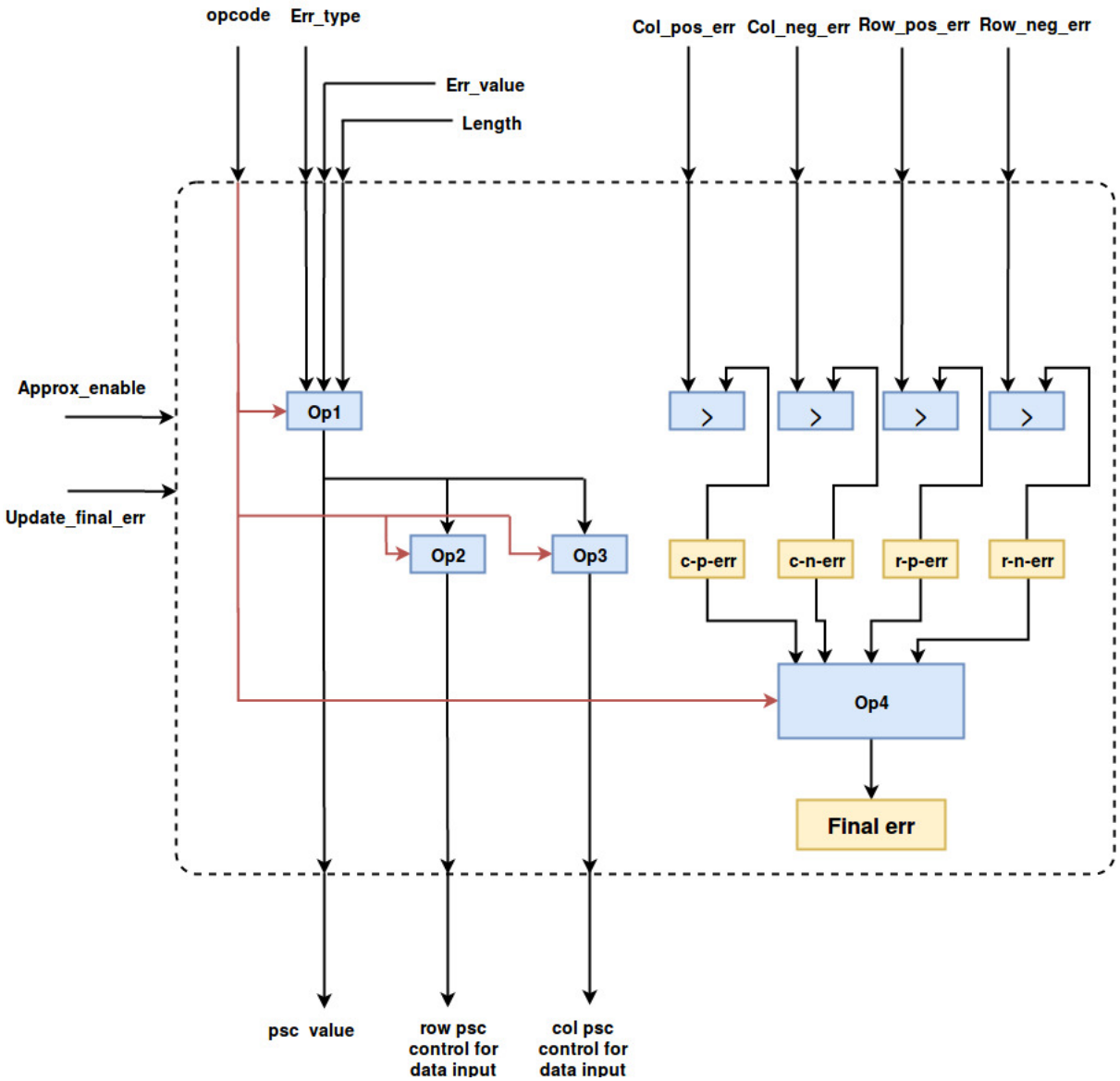

Figure 2.4: Quality Control Unit

Interface

1. Input Interface
   - Control signals

    - Approx_enable
    - Update_final_err
    - Opcode
  - Data inputs
    - Error_value
    - Length
    - col_pos_err
    - col_neg_err
    - row_pos_err
    - row_neg_err

2. Output Interface
   - Psc_value
   - Row psc control for operands
   - Column psc control for operands

Micro-architecture

This module has three outputs. Number of lsb bits of operands for vector type instructions that can be ignored is specified by psc_value. Row_psc_control and Col_psc_control are used to perform bitwise AND with row inputs and column inputs from streaming memory to get approximated inputs with psc_value number of lsb bits made zero.
During execution of a vector type instruction operands from streaming memory passes through precision control unit where operands are scaled according to quality value mentioned in the quality field of the instruction. If operands are scaled up positive error is imposed in the calculation and in case of operands being scaled down negative error is forced. Each precision control unit stores total positive error and negative error occurred during execution. After execution is over these error registers are streamed into quality unit to find maximum value of positive error and negative error among row and column precision control units. Register c-p-err and c-n-err store maximum positive error and maximum negative error respectively among column precision control units. Register r-p-err and r-n-err store maximum positive error and maximum negative error respectively among row precision control units. Maximum possible error for output is calculated using these values and stored in Final_err register.
Precision scaling values are calculated if approx_enable is true. Final_error register gets updated when update_final_err signal becomes high.

2.7 Precision scaling unit

QUORA has two 1D array of precision scaling units along the top and left border between 2D APE array and streaming memory banks. It generates approximate value for operands that are stored in streaming memory banks and scaled operands are forwarded to either APE or MAPE. Psc_value stores number of lsb bits that can be ignored. Bit width of psc_control is same as input operand. Except psc_value number of lsb bits, all bits of psc_control are high.

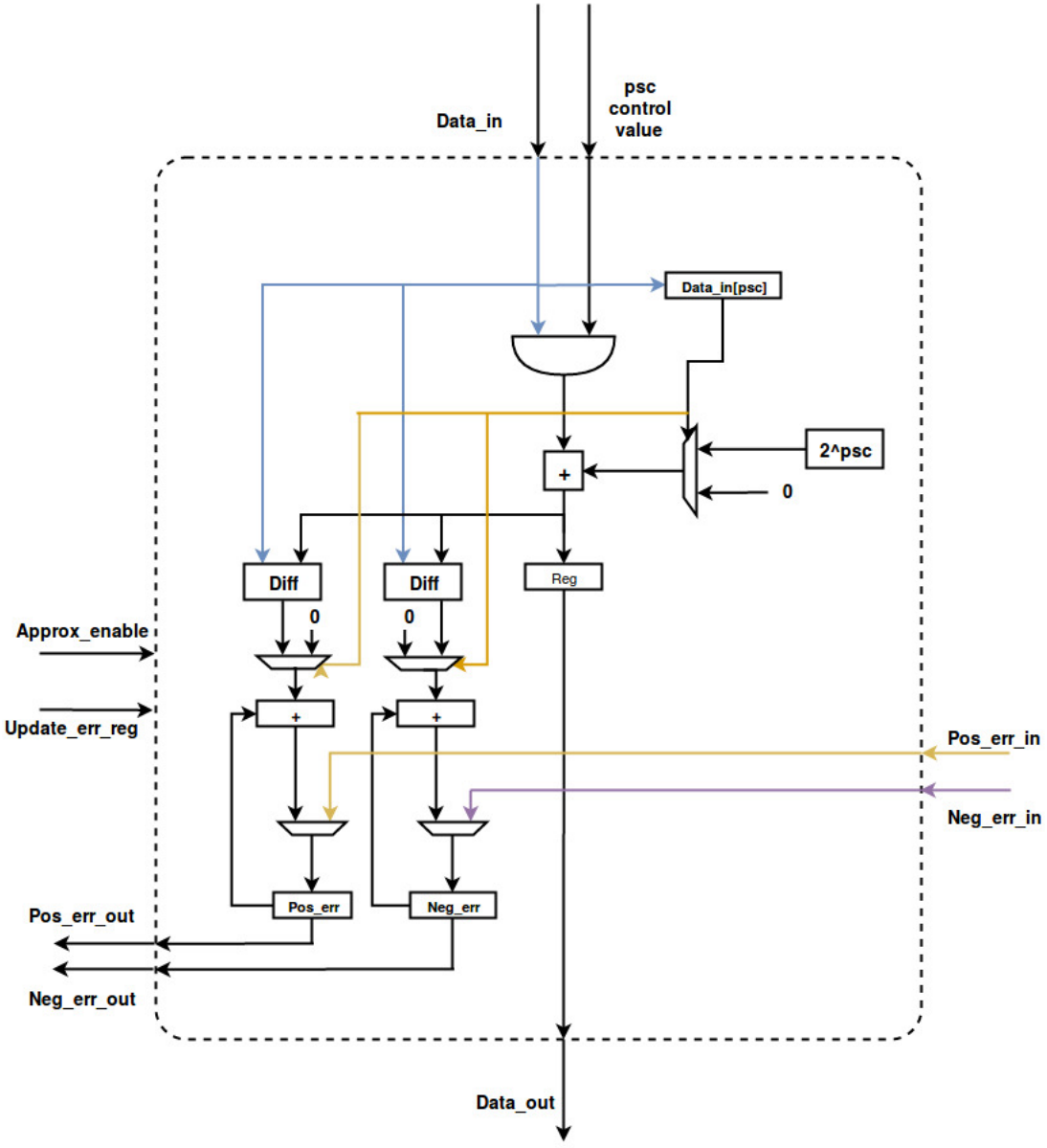

Figure 2.5: Precision scaling unit

Interface

1. Input Interface
    - Control signals
        - approx_enable
        - update_err_reg
    - Data inputs
        - psc_control
        - Data_in
        - pos_err_in
        - neg_err_in
2. Output Interface
    - Data outputs
        - Data_out
        - Pos_err_out
        - Neg_err_out

Micro-architecture

Bitwise AND is performed between input data and psc_control to get approximate version of input data. So maximum possible error is equal to $2^{psc} - 1$. To reduce this error 2^{psc} is added to the result in case of lsb psc bits being more than or equal to 2^{psc-1}. This reduces maximum possible error to 2^{psc-1}. This method of scaling is called up-down precision control.
Three registers located in this module are data_out, pos_err, neg_err. Register data_out stores inexact input data. During approximation of input data up scaling results in positive error and down scaling gives negative error. Pos_err and neg_err registers accumulate these errors and are streamed to quality control unit at the end of the instruction execution.
Approximation takes place while approx_enable is true. Error registers are streamed to quality control unit if update_err_reg is high.
Data_out proceeds to either APE or MAPE according to the instruction. Pos_err_out and neg_err_out is connected to previous precision scaling unit to stream pos_err and neg_err to the quality control unit.

2.8 Decode unit

Decode unit generates control signals for all other modules present in QUORA. Instruction width of QUORA is either 64 bits or 96 bits and bus width is 32 bits. So multiple fetch is required to read one whole instruction.

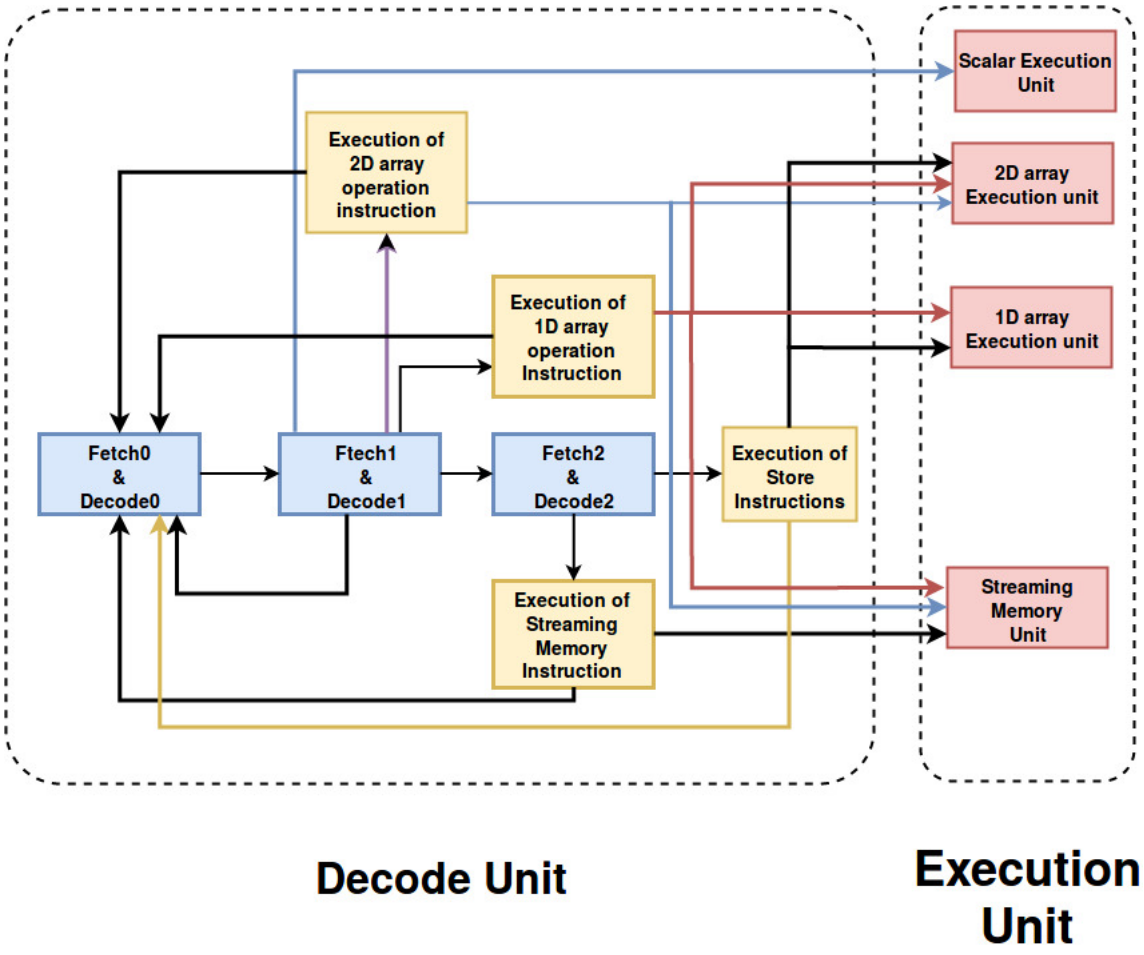

Figure 2.6: Control flow diagram of decode unit

In figure 1.6 control flow diagram of decode unit is shown. Fetch0 indicates fetching of first 32 bit of an instruction which contains opcode. So which control path should be chosen is known after first fetch. If instruction width is 64 bits, execution state of that instruction is reached after two clock cycle and stays there until execution is over. After execution is over control comes to fetch0 and decode0 state.

2.9 Target Operations

Operations required to execute target applications are discussed in this section.

Scalar Operations

Load-store of register located in reg file and branch operations belong to this section. Scalar instructions are responsible for these operations. These instructions operate on the registers in reg file. Scalar instruction width is 64 bits. Decoding of the last 32 bits and execution of the instruction take place in the same cycle. So 2 clock cycles are needed to complete execution.

Streaming Memory Operations

This includes loading of streaming memory from main memory. Streaming memory instructions are used for these operations. Operands of this instruction can be either immediate or register. Instruction width is 96 bits, 3 cycles is required to fetch and decode. Execution time equals to the number of elements to be loaded to the streaming memory. Decode unit provides memory address and also enables fifos specified in the instruction.

2D-array Reduction Operations

This operation is performed in the 2D APE array block. This multi cycle operation operates on data from column and row streaming memories. APE located at top left corner gets operands from row and column streaming memory. APEs located at the left boundary get operands from row streaming memory and APE located above it. APEs located at the top boundary get operands from column streaming memory and left APE. All other APEs get operands from APEs located at its left and top. After each cycle result is added to the accumulator present in APE and operands are forwarded to next APEs. Instruction width equals to 64 bits.
Matrix multiplication is an example of such operation. Let's assume we have two matrix , A and B of dimension x, and we need to find matrix C which equals to $A * B$. Successive rows of matrix A should be stored in consecutive row streaming memories and successive columns of matrix B should be stored in consecutive column streaming memories. So each row streaming memory contains a distinct row of matrix A and each column streaming memory contains a distinct column of matrix B. To generate one element of matrix C, x number of multiply and accumulate operations(MAC) are needed. So at least x cycles are

needed to get one element of matrix C as APE performs single cycle execution of MAC operation. Accumulator of the APE located in the top right corner stores value of C_{18}. First element of first row reaches to this APE after 7 th cycle. So we need to stall reading of 8th column streaming memory till 7th cycle and after 7+x number of clock cycle calculation of C_{18} is finished. Same number of cycle needed to calculate C_{81}. After 14+x clock cycle APE located at bottom right corner will contain C_{88}. Since data comes through precision scaling unit total 16+x clock cycles are needed to complete execution of these instructions assuming its a $8 * 8$ APE array.

1D-array Reduction Operations

MAPE array is used for this operation. Data present in accumulator of an APE is streamed along the row or column. MAPE finds min/max among input data vectors and store it in the register present in it. If it's a row scan number of clock cycle needed equals to row width of APE array and for column scan it's column width of APE array. This operation uses 64 bit instruction.

1D-array Streaming Operations

This operates on either the data present in APE or the data from streaming memory. Instruction width equals to 64 bits.

- Data present in the accumulator of APEs are streamed and operated in MAPE and the result is returned back to the same APE. If its row scan number of clock cycle needed equals to row width of APE array and for column scan it's column width of APE array.

- Input comes from streaming memory and operated in the MAPE and result is stored in the accumulator located in it. Number of clock cycles taken in execution equals to number of elements specified in the instruction.

1D-array Self Operand Operations

This single cycle operation operates on accumulator,SReg and mask register present in MAPE. Mask register are used for if operations. From the name it can be understood this operation does not take any external input. Instruction width equals to 64 bits.

Store Operations

In this multi cycle operation data present in the accumulator of APE and MAPE are stored in the memory. In this section row width indicates row width of APE array and col width indicates column width of APE array. Instruction width equals to 96 bits.

- Store instruction for MAPE requires row width number of clock cycles for row scan and col width number of clock cycles for column scan.
- Store instruction for APE requires $row\ width * col\ width$ units of time. If its a row scan data from APE comes to MAPE located at the right border, before proceeding to the memory. Only after all data in an APE row is stored, storing of next row starts.

2.10 Reduction in execution time for accurate vector operations

Vector instruction with quality field stores error imposed on input operand in precision scaling units. Maximum possible error for the output of these instruction is calculated using the errors stored in precision scaling unit. Quality control unit calculates this and stores it in Final error register. So we need to stream error values stored in precision scaling unit to quality control unit. This operation takes some extra cycles after execution of that instruction is over.
If the quality value present in the quality field is zero instruction is executed accurately and there is no need to calculate final error. So those extra cycle taken by these type of instruction to calculate maximum possible error for output can be removed.
This processor detects instruction that does not ask for approximation and takes less cycle to complete execution of those instructions.

Chapter 3

Instruction level parallelism

3.1 Introduction

To improve performance of QUORA first optimization considered is instruction level parallelism. This is achieved by overlapping execution of multiple instructions. To exploit ILP instructions with no conflicts must be find out first. Decode unit in this processor has the ability to find out parallelism dynamically. Program counter is stalled if read instruction has any conflict with instruction that is currently in execution state. Hazards possible in this method are structural hazards and data hazards.

3.2 Grouping of instructions for concurrent execution

Instructions in QUORA ISA is divided among groups so that instruction from different group can execute together. These categories are described below and each group is given a name for later use.

Table 3.1: Grouping of instructions

Loadf	Loading of data from main memory to streaming memories
Vector_op	Instructions that operates on MAPE and APE array
Scalar_op	Scalar instructions

3.3 Hazards

Only data hazard and structural hazards are present in this processor. Since this processor is not pipelined there is no possibility of control hazards.In case of hazard program counter is stalled until that hazard is resolved.

3.3.1 Data hazard

In QUORA data hazard happens only if there is data dependence between two instructions. In that case we need to execute those instructions sequentially.
LDSM length,stride,burst,start_addr,row_en,col_en
qpMAC length,row_pe_en,,col_pe_en,q_type,q_value
In the above example in order execution is required though these instructions belong to different categories. LDSM instruction loads data to streaming memories. qpMAC performs 2D reduction operation with data present in the streaming memory bank. So qpMAC can not start until streaming memories are loaded. This hazard occurs if Loadf instruction precedes a Vector_op instruction that takes operand from streaming memories.

3.3.2 Structural hazard

If two instructions use same resources structural hazard takes place. Then those instruction must be computed sequentially.

- In all three categories some instructions are present that communicates with main memory. These instructions are loading streaming memory from main memory(Loadf), loading register from main memory(Scalar_op), storing data from APE and MAPE array block to main memory(vector_op). These instructions can not execute concurrently.

- Loadf loads data to streaming memory banks and 2D reduction instruction and 1D reduction instruction which belongs to Vector_op, reads from streaming memory banks. Both instructions use same resource and that is streaming memory bank.
  If Loadf instruction comes before 2D reduction instruction or 1D reduction instruction data hazard takes place and it is discussed before. There is possibility of structural hazard if Loadf instruction comes while 2D reduction instruction or 1D reduction instruction is in execution state. Streaming memories are fifo buffer, so dequeue and enqueue can be done in the same clock cycle.
  qpACC length quality type quality value row/col enable inst1
  LDSM length,stride,burst,start_addr,row_en,col_en inst2
  Instruction qpACC accumulates data from streaming memory and stores the result in MAPE accumulator. Since dequeue and enqueue can be done in the same cycle inst2 can be executed while inst1 in execution state.

  qpMAC length,row_pe_en,,col_pe_en,q_type,q_value inst3

LDSM length,stride,burst,start_addr,row_en,col_en inst4
QpMAC instruction executes 2D reduction operation and this operation is discussed in Target Operations section. It is assumed dimension of 2D APE array is 8. Dequeuing of the last (in this case 8th) fifo buffer starts after 7 th clock cycle since qpMAC starts to execute. So we need to stall LDSM instruction for 7 clock cycles.

3.4 Modified Decode Unit

First block fetch0 and decode0 fetches first 32 bits of an instruction and then it checks if there is any conflict between this instruction and the instruction which is already in execution state. In case of no conflict instruction is decoded otherwise instruction is stalled until conflicts are resolved.

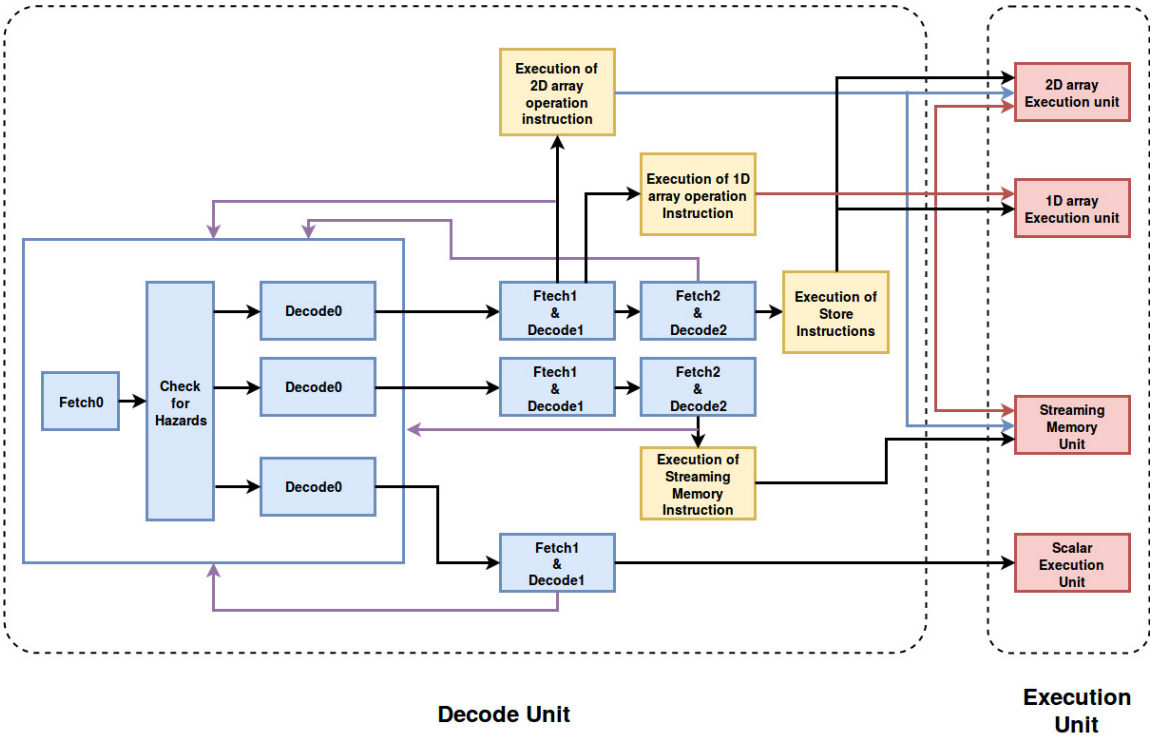

Figure 3.1: Modified control flow diagram of decode unit

If no hazard is present control proceeds through one of the three control paths. Since instructions are grouped in three category three different control paths are needed. One other modification is after fetching of last 32 bits of an instruction , next instruction is fetched in the next cycle. In old decode unit next instruction is read only after execution of

current instruction is finished.

3.5 Drawbacks

Instruction in QUORA takes multiple cycles to complete execution. So control signals for a instruction must be stored in a register until execution is over. Instructions can share same set of control registers if QUORA does not have ILP feature. Instructions in QUORA with ILP feature can only share control registers if they belong to same category. So we need more number of control registers in the decode unit. This is the only drawbacks of QUORA with instruction level parallelism.

Chapter 4

On chip multi bank memory

4.1 Introduction

This processor mainly works on data vectors from streaming memory banks. Till loading data to streaming memory bank is not finished operation can not be started. So it is important to load fifo buffers as quickly as possible. Keeping this in view this processor is equipped with multi bank on chip memory.The example given below describes the need of having multi bank memory architecture.
Time taken to multiply two matrix a and b with length l (Tl) = Time taken to load those matrix in streaming memory(Tload) + time taken for matrix multiplication operation in 2D APE array (Tmul).

$$Tload = 2 * l * l * tclk$$

$$Tmul = (Row\_width + col\_width + l)tclk$$

$$Tl = Tload + Tmul = 2 * l * l * tclk + (Row\_width + col\_width + l)tclk$$

It is apparent that time taken to load the matrix is much more than the time taken for matrix multiplication operation. Tload can be reduce by using multi bank memory.
Multiple load units are needed for concurrent operations on different memory banks. Interconnection network is used to put memory request to memory banks and to get the read data from memory. Only one store unit is present in this processor. Delta network is used as interconnection network. It has as many inputs as the number of memory banks. To get the data from memory one more delta network is kept.

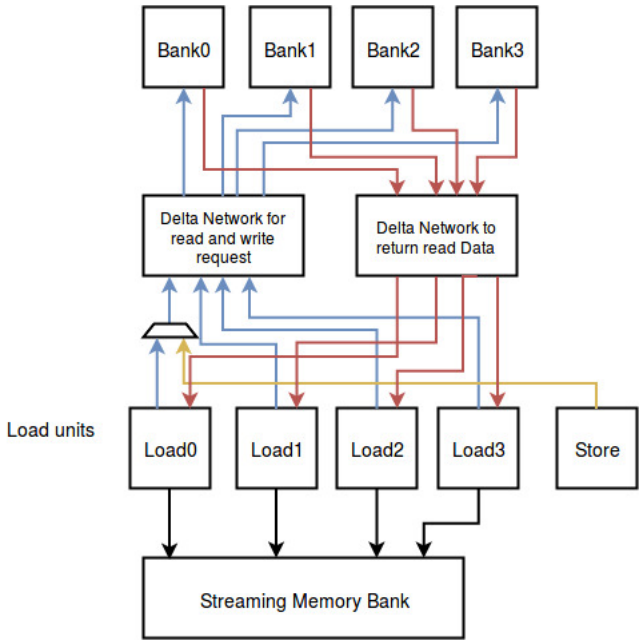

Figure 4.1: Memory architecture

4.2 Delta Network

Basic building block of a delta network is a $2 * 2$ crossbar switch. A $2^n * 2^n$ delta network has n number of stages and each stage contains 2^{n-1} number of $2 * 2$ crossbar switch.

4.2.1 2*2 crossbar switch

Two input ports and two output ports are present in a 2*2 crossbar switch. Data present in an input port can go to any one of the output ports based on the control bit. If control bit is 0 data is forwarded to output port out0 and in case of control bit being 1 data goes to output port out1.

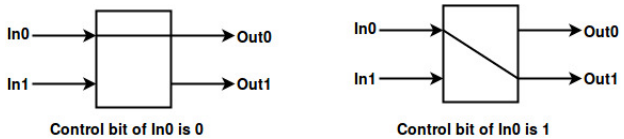

Figure 4.2: 2*2 crossbar switch

If control bit of both input ports are same , only data from one of them will be forwarded

to that particular output port and the other input port will be given the higher priority next time in the same scenario.

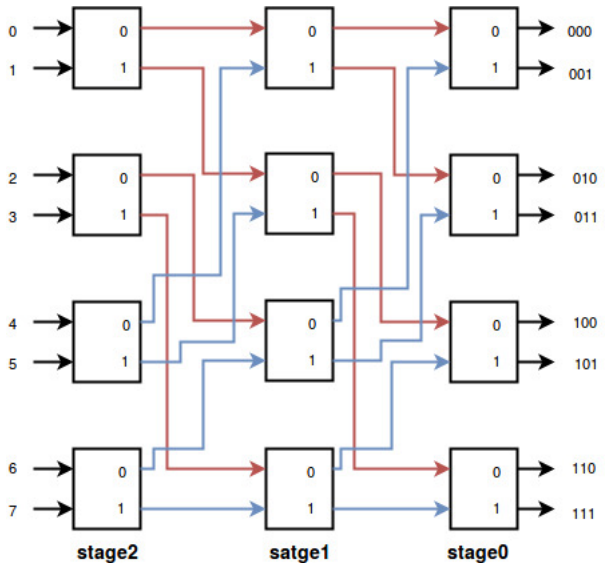

Figure 4.3: Delta Network

4.2.2 Interconnection between stages

A $N * N$ delta network (N = 2^n) has n number of stages. Output port of one stage is connected to input port of next stage. Ports are numbered from 0 to N-1. Output port i of a stage is connected to input port j of next stage according to following formula.

$$j = 2i \; mod \; (N-1) \qquad i! = N-1$$

$$= i \qquad\qquad\qquad\qquad i = N-1$$

4.2.3 Control bit for each stage

Every input port uses one control bit to choose between two output ports located in the same crossbar switch. For $2^n * 2^n$ delta network n bits are needed to address all the outputs. Stages of a delta network are numbered as shown in the figure 3.3. If destination address is $(a_2a_1a_0)_2$ a_0 , a_1 and a_2 are the control bits for stage0, stage1 and stage2 respectively.

4.2.4 Interface signals

Two delta network is used. Interface signals of the input port of the network that receive memory request from load units are

- Request
  - Request must be high for read and write request to memory.
- Load unit address
  - Address of the load unit sending memory request for reading data from memory. Delta network that sends read data to load unit must know from which load unit this request has come.
- Data that need to be stored
  - This is forwarded by store unit.
- Memory address
  - Memory address of the data for storing or loading must be forwarded to the bank through the network. Least significant bits are used to choose the proper path form the input to the output of the delta network.
- Read request
  - True if it's a read request.
- Write request
  - True if it's a write request.
- Success
  - Delta network return this signal to load unit. In a crossbar switch if both ports request for same output port one of the request is forwarded. Load unit of the rejected request has to know it's request has been discarded. Success signal will be false for rejected request.

Interface signals of the ports of the network that collects the data from memory banks and returns it to the particular load unit are

- Request
  - True if any load unit has requested for data to this particular bank.
- Address
  - Address of the load unit that has requested for the data.
- Data
  - Requested data from memory bank.

4.3 Memory bank select

We know n bits are needed to select a bank among 2^n number of banks of memory. In this design n lsb address bits are used for bank select. So addresses with same modulo (2^n) value belongs to same memory bank.

4.4 Modified decode unit

The new decode unit has to replicate load unit as many time as the number of memory banks. Control flow diagram of changed fetch0 and decode0 block is shown in figure 3.4. If the new instruction is a streaming memory instruction in the absence of hazard decode unit will look for an idle load unit . If no load unit is idle program counter will be stalled, otherwise decode unit will assign one of the idle load units to execute current streaming memory instruction.

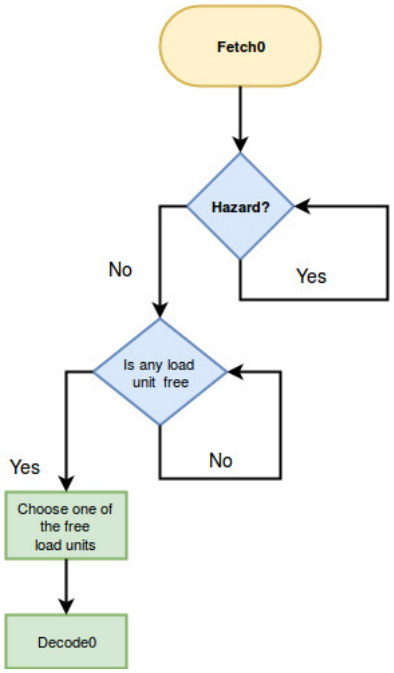

Figure 4.4: Fetch0 and decode0 block of modified decode unit

4.5 Streaming memory input

Streaming memory receives input from one of the load units. So a multiplexer with select signal load unit select is used to choose the input. Fifo enable is the enable signal of the

multiplexer. If none of the load unit is instructed to put data to a streaming memory fifo enable will be false for that particular fifo buffer.

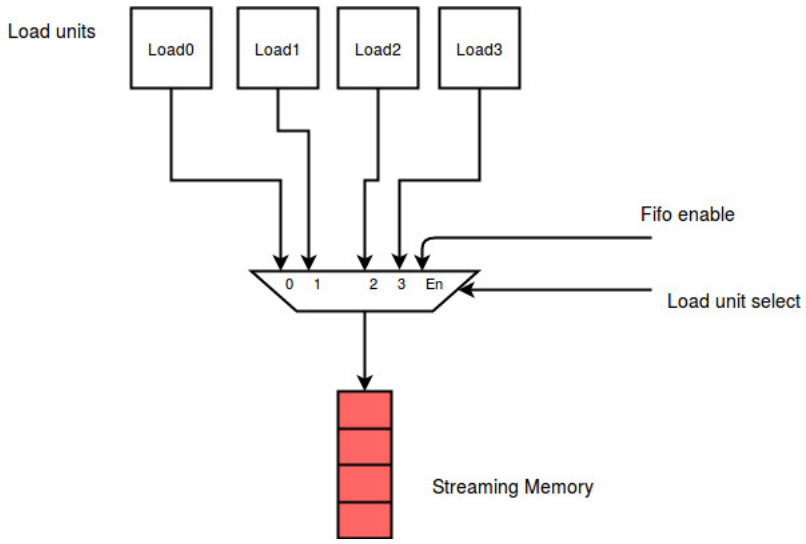

Figure 4.5: Streaming memory input

4.6 Drawbacks

Drawback of this new design is additional area is needed to accommodate on chip memory, interconnection network and multiple load unit.

Chapter 5

Improvement of maximum operating frequency

5.1 Introduction

To get better performance higher operating frequency is required and the critical path is needed to be pipelined to achieve that. To attain higher operating frequency finding out the best position of pipeline register is important. Virtex ultrascale fpga is used for this purpose. In this discussion it is assumed that this processor is not equipped with multi bank on chip memory.

5.2 Method to reduce critical path delay

This design has been synthesized and implemented in vivado.

Target Board: Virtex Ultrascale
Target Part: xcvu095-ffva2104-2-e

After implementation vivado generates a timing report which contains the critical path. Delay of the critical path is maximum and maximum operating frequency is decided by this delay. If design can not run at target frequency negative slack present in the critical path will be shown. In that case target frequency need to be reduced and in case of positive slack target frequency need to be increased. In this way we can find out least critical path delay.
CAPE module contains a divider and naturally this will be the critical path. For the time being divider is removed since pipelined divider is not available now. Quality unit, APE and MAPE have a multiplier which lies in the critical path. So multiplier needs to be pipelined.
Only two stage pipeline is used and the pipeline register is placed at such a position that it divides the critical path equally. After keeping the pipeline register design is synthesized and implemented in vivado to find out the new critical path and the pipeline register is moved towards the critical path. The above step is repeated until either the path delay for those two paths become almost same or some other path becomes critical.

5.3 Pipelined multiplier

A pipelined booth multiplier is designed for this purpose. Booth multiplier is implemented by generating all the partial products and adding them afterwards. This task has been divided by putting a pipeline register in between.

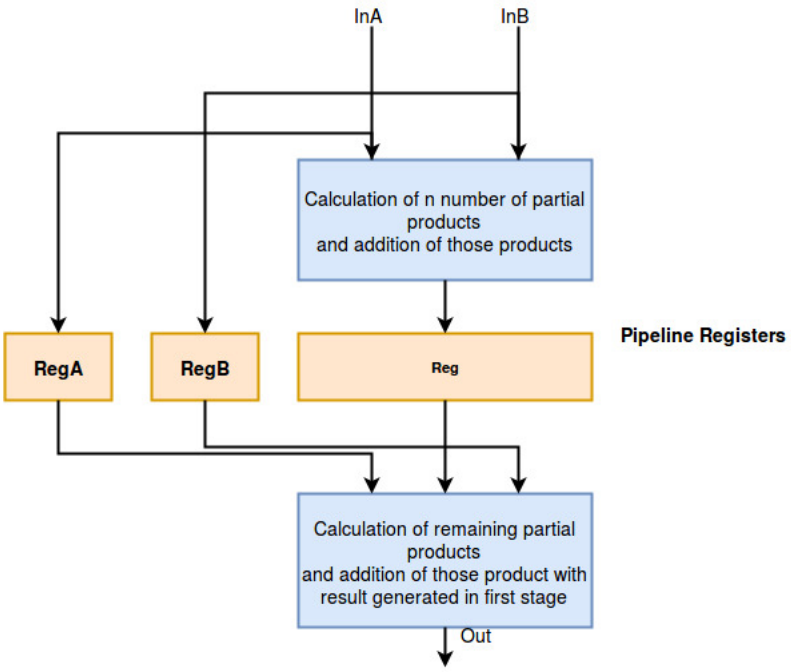

Figure 5.1: Pipelined booth multiplier

Let's assume total N number of partial products are need to be evaluated and added to get the final output. As it is shown in figure 4.1 first stage finds out first n number of partial products , adds all those products and result is forwarded to the next stage. Next stage evaluates remaining N-n number of partial products and adds those partial products to the result generated in the first stage to get the final result.
Since second stage calculates some partial products operands must be stored in pipeline registers.

5.4 Position of pipeline register

32 bit radix-4 booth multiplier is used in the design and it generates 17 partial products. Position of the pipeline register for least critical path delay are as follows
Multiplier located in APE: n=3

Multiplier located in Quality control unit: n=3
Multiplier located in MAPE: n=4
Before putting pipeline register maximum path delay for this design is 15.4 ns. After putting pipeline register least path delay among the paths that has pipeline register is 7.2 ns. Least critical path delay for this design is 7 ns.

- Source : Register located in reg file
- Destination: Register present in load unit.

5.5 Instructions affected by pipeline

2D reduction instruction qpMAC, qpMOD2 and qpMACSR use multiplier present in APEs. Since only operand in an APE is forwarded to next APE, there is no need to wait for multiplication to be finished before operand is forwarded. So pipelined architecture will take only one extra clock cycle to execute qpMAC and qpMACSR but it will take two extra cycles for qpMOD2 since it uses pipeline multiplier present in quality control unit to calculate final error.
Two other instruction that are affected by pipelined architecture are qpMULO and qpMULIOP. For both of these instructions execution time increases by one clock cycle.

5.6 Drawbacks

We need pipeline register for every multiplier in QUORA. QUORA of dimension 8 has total 8*8 + 8*2 + 8*2 = 96 multiplier. So to implement this optimization 96 pipeline registers are needed. So area of QUORA will increase.

Chapter 6

RTL design, FPGA implementation and performance analysis

6.1 RTL Design

This processor is designed in bluespec system verilog. Modules designed for this processor are as follows

1. Top

   This is the top module of my design. It includes decode unit, instruction memory.

   (a) Decode unit

   It takes instruction from memory and produces control signal. This module includes Ram blocks, execution units , quality control units,delta network.

   i. Quality control units

   It generates psc control value and forward it to decode unit.

   ii. Execution unit

   This module contains streaming memory, 2D APE array, 1D MAPE array and psc units.

   iii. Delta network

   It contains stages for delta network and interconnection between them.

   iv. Ram block

   Block ram provided in BRAMCore package of bluespec is used.

6.1.1 Design parameters

Design parameters and value chosen for those parameters in my design are noted in the following table.

Table 6.1: Design parameters

Array dimension	8*8
Fifo depth	65
Fifo data width	16
No of Load units	4
Delta network dimension	4*4
Memory size	4 Mbyte

6.2 Sample program

8*8 discrete cosine transform is performed on a 256*256 gray image. So we need to load the image and 8*8 DCT matrix(T) in data memory. Pixel values of a gray image varies from 0 to 256 where a totally black pixel is represented by 0 and totally white pixel is represented by 255.
DCT is designed to work on pixel values ranging from -128 to 127 . So all pixel values are subtracted by 128. DCT is operated on all 8*8 blocks(M) present in the image.
DCT formula is given by TMT'.
This program is divided into 3 parts.

1. Operation performed by 1st part

   Loads pixel values in streaming memory, subtracts 128 from each pixel in MAPE array and stores into memory.

2. Operation performed by 2nd part

   Calculates T*M for each 8*8 block present in the image. It loads 8*8 block from 256*256 image and DCT matrix in column streaming memory and in row streaming memory respectively. Matrix multiplication of those two matrix is performed in 2D APE array and then result is stored in the main memory.

3. Operation performed by 3rd part

   Performs (T*M)*T' for each 8*8 block present in the image. It loads 8*8 block from 256*256 array which is generated by the last block(T*M) and DCT matrix in row streaming memory and in column streaming memory respectively. Matrix multiplication of those two matrix is performed in 2D APE array and then result is stored in the main memory.

6.2.1 Machine code

Machine codes for the sample program is given in appendix A.

6.2.2 Results

After calculation of DCT of an image, IDCT is performed on the result in python and the outcome is shown in figure 5.1. This program does not use any approximation. For approximate DCT calculation qpMACSR instruction of 2nd part of the program must have non zero quality value. For approximation of 1 lsb bit it is changed to 53018FFF and for approximation of 2 lsb bits it is changed to 53014FFF.

Figure 6.1: IDCT of DCT output

Figure 6.2: IDCT of 1 lsb bit approximated DCT output

Figure 6.3: IDCT of 2 lsb bit approximated DCT output

To evaluate amount of noise introduced in approximated DCT calculation PSNR of figure 5.2 and figure 5.3 with respect to figure 5.1 is calculated.

Table 6.2: PSNR values

Image	PSNR value
1 lsb bit is zero	41.3042 dB
2 lsb bits are zero	43.4604 dB

6.3 FPGA Implementation and Performance analysis

Initial architecture is improved using three optimizations. They are instruction level parallelism, multi bank on chip memory and pipelined multiplier. Let's define four processors $processor_1$, $processor_2$, $processor_3$, $processor_4$. Initial design is $processor_1$ and other

processors are the optimized versions of $processor_1$. Optimization used in $processor_2$ is ILP, Optimizations used in $processor_3$ are ILP and multi bank on chip memory, Optimizations used in $processor_4$ are pipelining the multiplier, ILP and multi bank on chip memory.

6.3.1 FPGA implementation

$Processor_3$ and $processor_4$ are implemented in vivado.
Target Board: Virtex Ultrascale
Target Part: xcvu095-ffva2104-2-e

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	81646	0	537600	15.19
LUT as Logic	81646	0	537600	15.19
LUT as Memory	0	0	76800	0.00
CLB Registers	26448	0	1075200	2.46
Register as Flip Flop	26448	0	1075200	2.46
Register as Latch	0	0	1075200	0.00
CARRY8	1914	0	67200	2.85
F7 Muxes	461	0	268800	0.17
F8 Muxes	22	0	134400	0.02
F9 Muxes	0	0	67200	0.00

Figure 6.4: Synthesis report of $processor_3$

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	155254	0	537600	28.88
LUT as Logic	155254	0	537600	28.88
LUT as Memory	0	0	76800	0.00
CLB Registers	32055	0	1075200	2.98
Register as Flip Flop	32055	0	1075200	2.98
Register as Latch	0	0	1075200	0.00
CARRY8	4936	0	67200	7.35
F7 Muxes	592	0	268800	0.22
F8 Muxes	81	0	134400	0.06
F9 Muxes	0	0	67200	0.00

Figure 6.5: Synthesis report of $processor_4$

Operating frequency

$Processor_3$
Operating frequency of $processor_3$ in virtex ultrascale fpga is 64.9 MHz. Figure 5.6 shows the maximum delay path for $processor_3$. Multiplier located in the APE lies in the critical path.

```
Max Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             0.021ns  (required time - arrival time)
  Source:                 exe_unit/pe_1_1/rg_op0_reg[0]/C
                            (rising edge-triggered cell FDRE clocked by CLK  {rise@0.000ns fall@7.700ns period=15.400ns})
  Destination:            exe_unit/pe_2_1/rg_op0_reg[31]/D
                            (rising edge-triggered cell FDRE clocked by CLK  {rise@0.000ns fall@7.700ns period=15.400ns})
  Path Group:             CLK
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            15.400ns  (CLK rise@15.400ns - CLK rise@0.000ns)
  Data Path Delay:        14.846ns  (logic 7.543ns (50.808%)  route 7.303ns (49.192%))
  Logic Levels:           28  (CARRY8=5 DSP_A_B_DATA=2 DSP_ALU=3 DSP_M_DATA=2 DSP_MULTIPLIER=2 DSP_OUTPUT=3 DSP_PREADD_DATA=2
  Clock Path Skew:        -0.558ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    4.053ns = ( 19.453 - 15.400 )
    Source Clock Delay      (SCD):    5.077ns
    Clock Pessimism Removal (CPR):    0.466ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
  Clock Net Delay (Source):      3.389ns (routing 1.255ns, distribution 2.134ns)
  Clock Net Delay (Destination): 2.730ns (routing 1.154ns, distribution 1.576ns)
```

Figure 6.6: Maximum delay path of $processor_3$

$Processor_4$

Operating frequency of $processor_4$ in virtex ultrascale fpga is 102 MHz. Figure 5.7 shows the maximum delay path for $processor_4$. Critical path starts at Ram register output and finishes at streaming memory input.

```
Max Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             0.043ns  (required time - arrival time)
  Source:                 ram1_3/RAM_reg_3_6/CLKARDCLK
                            (rising edge-triggered cell RAMB36E2 clocked by CLK  {rise@0.000ns fall@4.900ns period=9.800ns})
  Destination:            exe_unit/col_fifo_7_rvData_reg[790]/D
                            (rising edge-triggered cell FDRE clocked by CLK  {rise@0.000ns fall@4.900ns period=9.800ns})
  Path Group:             CLK
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            9.800ns  (CLK rise@9.800ns - CLK rise@0.000ns)
  Data Path Delay:        9.461ns  (logic 2.034ns (21.499%)  route 7.427ns (78.501%))
  Logic Levels:           9  (LUT3=2 LUT4=1 LUT6=4 MUXF7=1 MUXF8=1)
  Clock Path Skew:        -0.321ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    3.864ns = ( 13.664 - 9.800 )
    Source Clock Delay      (SCD):    4.679ns
    Clock Pessimism Removal (CPR):    0.494ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
  Clock Net Delay (Source):      3.134ns (routing 0.929ns, distribution 2.205ns)
  Clock Net Delay (Destination): 2.738ns (routing 0.854ns, distribution 1.884ns)
```

Figure 6.7: Maximum delay path of $processor_4$

6.3.2 Performance analysis

In the following table number of cycles taken to execute the sample program and execution time taken by different processors are given.

Table 6.3: Performance comparison

Processor	Clock cycles to complete execution	Execution time(msec)
$Processor_1$	1009440	15.54
$Processor_2$	758808	11.68
$Processor_3$	693583	10.68
$Processor_4$	695631	6.79

Performance of the final architecture is 2.28 times higher than the basic QUORA architecture.

Chapter 7

Conclusion

QUORA, a quality programmable vector processor targets applications like recognition, mining, synthesis, video processing, search because of their acceptability of inexact result. Keeping in mind the behavior of these applications QUORA's architecture is designed. To improve performance of QUORA three optimizations are used and they are ILP,on chip multi bank memory and pipelined multiplier. In this thesis I have described hardware challenges and how to resolve those issues to incorporate those optimization in QUORA. To compare performance I have used a sample program and showed performance of optimized version of QUORA is much higher than the basic QUORA architecture.

Bibliography

[1] Bluespec Inc.Bluespec System Verilog Reference Guide,Revision 30 July 2014.

[2] Quality programmable vector processors for approximate computing,46th Annual IEEE/ACM Symposium on Microarchitecture (MICRO-39), 2013

[3] Performance of Processor-Memory Interconnections for Multiprocessors, IEEE Transactions on Computers (Volume: C-30, Issue: 10, Oct. 1981)

[4] Computer Architecture: A Quantitative Approach, David Patterson and John L. Hennessy

Chapter 8

Appendix A

Machine code of sample program is as follows

1st part

```
38000001      LDSRO load sreg of row oppe with -128('h80 )
00010042      starting address
FFFFFFFF
C0010000      LDRI load reg_01 with 64
00000040
C0020000      LDRI load reg_02 with 0
00000000
C0050000      LDRI load reg_05 with 01
00000001
C0060000      LDRI load reg_05 with 01
FFFFFFFF
C0070000      LDRI load reg_07 with 0
00000000
C0090000     load reg_09 with loop variable = row(256) * col(256) / 512 -1
0000007F
C0040000      ** loop3 starting
00000007
C0030000      load reg_03 (used to specify row_fifo_en to load fifo)
00010000
1F010001     ** loop1 starting, LDSM (in one row fifo 64 elements are loaded)
40000002
00000003
CF020200      ADDRI start_addr is incremented by 64('h40)
00000040
C8030300      LSFTRI 1 bit left shift to enable next fifo
00000001
EF04FFF8      BGZDI loop 8 times
00000001      end of loop1
C0080000      LDRI load reg_08 with 1(loop2 cond)
0000003F
87050000     ** loop2 starting, qpACC (only one element is added)
00060000
```

```
BD000000    QpADDO sreg(-128) is added to accumulator
00060000
37000040    STROR store values in MAPE to memory
00000007
00060000
CF070700    ADDRI start address is incremented by 1
00000001
EF08FFF6     loop 64 times
00000001     end of loop2
CF070700     ADDRI start_addr is incremented by 448
000001C0
EF09FFE3     loop 8 times
00000001     end of loop 3
```

2nd part

```
C0090000     This reg is used to load next matrix located after 8 rows
00000000
C00A0000     no of iteration of loop7 (no of row(256)/8 -1 = 31)
0000001F
CF060900    ** starting of loop 7
00000000
C0070000     loop 6 cond variable (no of col(256)/8-1 = 31)
0000001F
C0010000    **loop 6 starting, next block fills up row fifo with dct coeff
00000008    length = 8
C0020000     starting addr = 65536
00010000
C0030000     Reg for row_fifo_en
00010000
C0040000     loop variable
00000007
1F010001    **loop 4 starting, 8 elements are stored in one fifo
08000002
00000003
CF020200     starting address is incremented by 8
00000008
C8030300     Reg_9 is modified to enable next row fifo
00000001
EF04FFF8     branch instruction
00000001     loop4 finishes here
C0010000     next block stores one 8*8 block from image in col fifo (8*8)
00000008     length = 8
CF020600     start address = 0
00000000
```

```
C0030000      Reg for column fifo enable
00000080
C0040000      loop variable
00000007
1F010100     ** loop 5, LDSMR (8*8 matrix is stored from image)address stride is 256
01000002
00000003
CF020200      start address incremented by 1
00000001
C9030300      Reg_9 is modified to enable next col fifo
00000001
EF04FFF8      branch instruction
00000001     *** loop 5 finishes here
C0010000      Reg to store the length for qpMACSR inst
00000008      and the burst size for store inst
C0020000      Reg for APE enable
FFFFFFFF
C0030000      Reg for APE enable
FFFFFFFF
53010000     qpMACSR
00020003
C0040000     specifies stride for store instruction
000000F9      address stride formula – ((image dimension)256-7 = 249 ('hF9))
CF050600      starting address
00000000
2F000004      STRR
01000005
00020003
CF060600     Reg_6 is modified to load next matrix block
00000008
7F000000      Reset all APEs
FFFFFFFF
EF07FFCA
00000001     loop 6 finishes here
CF090900      Reg_09 is modified to load next matrix, increment by dimension*8 = 2048
00000800
EF0AFFC2
00000001     loop 7 finishes here
```

3rd Part

```
C0090000      Reg_9 is used to load next matrix located after 8 rows
00000000
C00A0000      loop 11 condition variable (no of row(256)/8 -1 = 31)
0000001F
```

```
CF060900    ** loop 11 starts here
00000000    this reg is used to load matrix from next 8 rows
C0070000    loop 10 condition variable (no of col(256)/8-1 = 31)
0000001F
C0010000    **loop 10 starts here, This loop fills up column fifo with dct coefficient
00000008    length = 8
C0020000    starting addr = 65536
00010000
C0030000    Reg for column fifo enable
00000080
C0040000    loop variable for loop 8
00000007
1F010001    ** loop 8 starts here
08000002
00000003
CF020200    starting address is incremented by 8
00000008
C9030300    Reg_3 is modified to enable next column fifo
00000001
EF04FFF8
00000001    loop 8 finishes here
C0010000    next loop stores next 8*8 block from image in row fifo (8*8)
00000008    length = 8
CF020600    start addr = 0
00000000
C0030000    Reg for row fifo enable
00010000
C0040000    loop variable
00000007
1F010001    ** loop 9 starts here
08000002
00000003
CF020200    starting address is incremented by 256 (dimension of image)
00000100
C8030300    Reg_3 is modified to enable next row fifo
00000001
EF04FFF8
00000001    loop 9 finises here
C0010000    Reg to store the length for qpMACSR instruction
00000008    and the burst size for store inst
C0020000    Reg for APE enable
FFFFFFFF
C0030000    Reg for APE enable
FFFFFFFF
53010000    qpMACSR
00020003
```

C0040000 specifies stride for store instruction
000000F9 address stride formula – ((length of row)256-7 = 249 (’hF9))
CF050600 starting address
00000000
2F000004 STRR
01000005
00020003
CF060600 Reg_6 is modified to load next matrix(row wise)
00000008
7F000000 reset APE
FFFFFFFF
EF07FFCA loop 10 finishes here
00000001
CF090900 This reg is modified to load next matrix(row wise),increment by dimension * 8 = 2048
00000800
EF0AFFC2
00000001 loop 11 finishes here
FFFFFFFF HALT