AN EFFECTIVE FPU STREAMING PROCESSOR FOR FPGA ACCELERATORS

[1]CHINTA SRAVANI, [2]Dr. PRASAD JANGA, [3]Mrs. SRIBINDU

[1]PG Scholar,[2]Associate Professor,[3]Associate Professor
Department of ECE, CMR Institute of Technology,
Kandlakoya, Hyderabad, Telangana, India

Abstract : In order to increase the speed and decrease the taxation on the hardware, other components are used in the system such as accelerators which can boost speed of the circuit. Hardware accelerators use computer hard ware for performing functions more efficiently than software running on a more general purpose CPU. To perform operations at high speed we have custom circuits where flexibility of circuit is static and soft-process approach where there is only register to register transfer is present whose performance is not much efficient.

To improve the flexibility and to overcome the faults in existing system a high performance streaming processor, known as streaming accelerator element, is proposed which realizes accelerators as large scale custom multicore networks. By implementing this approach with advanced program control and memory addressing capabilities, we can see that the program inefficiencies can be almost eliminated to enable performance and cost, which are not possible among other software-programmable solutions. When used to realize accelerators for matrix multiplication it is shown how the proposed architecture enables real-time performance. For accommodating the floating point operations we add Floating Point Unit (FPU) to the ALU of processing elements which performs IEEE754 2008 single precision floating point operations addition, multiplication, and subtraction.

Index Terms -Field Programmable Gate Array (FPGA), Accelerators, Floating Point, Matrix Multiplication, Streaming Elements

1.INTRODUCTION

In order to increase the speed of operations in computers accelerators are used. As we can see that the hardware acceleration is the use of computer hardware circuit for performing some functions more efficiently than that is possible through software running on a more general-purpose CPU. This hardware which performs the acceleration may be part of a general-purpose CPU, or it is designed as a separate unit. In the second case, where the circuit is designed as a separate circuit is referred to as a hardware accelerator.

Processors were designed as sequential circuits which executes the instructions one by one, and these sequential circuits are designed to run general purpose algorithms controlled by instruction fetch which performs operations such as moving temporary results to and from a register file. Hardware accelerators improve the performance of an algorithm by allowing specific data-paths for its temporaries and reducing the overhead of instruction control. We see that the modern processors are multi-core and operate on parallel SIMD units. Basically, we see that there are different types of techniques for which we implement accelerators i.e., custom circuits, general purpose circuits, softcore processors, and software accelerating elements.

Custom circuits are the circuits which are developed depending upon the required requirement. We see that in custom circuits the performance is achieved for maximum extent while cost of this procedure goes for a very high value which makes this a bit expensive. For overpowering such cost issues, we have approached a softcore processor.

These softcore processors are implemented depending on the software approach where it follows a path with out any external circuit design. But this procedure does not show such compatible results concerning the performance of custom circuits. This technique is not cost effective but does not compare when compared with performance results of custom circuits. In order to have a procedure which compares with the performance achieved by custom circuits we use streaming accelerating elements for acceleration.

The assets accessible inside present day FPGA, which might be utilized to form the accelerators, are uncommon: per-second access to trillions of multiply– gather (MAC) tasks and bit-level memory areas through on-chip DSP units and block RAM (BRAM). These check FPGA as perfect hosts to superior custom registering designs for flag, picture, and information preparing. Be that as it may, as the scale and the advancement of FPGA gadgets increment with each passing age, saddling these assets turns out to be progressively difficult. Customarily, accomplishing essential execution and cost has required manual outline of custom circuits at enroll move level in an equipment plan dialect. This is an exceedingly viable approach, yet forces an overwhelming improvement stack because of the low level of outline reflection.

Soft processors have been proposed to lighten this plan burden by employing a predominately software-based development course, yet at introduce, embracing such an approach requests significant trade off on performance and cost. No approach has been appeared to help execution and cost even near custom circuits planned through the customary approach.

To resolve this issue a novel streaming accelerating element (SAE) is introduced which empowers programming based streaming element advancement, while keeping up the execution and cost of custom circuits.

By application of streaming elements for matrix multiplication (MM), the accompanying commitments are made.

1) A novel streaming processor for FPGA, the SAE, is depicted and appeared to conquer the execution confinements of existing soft processors.

2) It is indicated how the SAE is exceptional among soft cores in empowering ongoing accelerators, for example, H.264 video.

3) It is indicated how SAE-based accelerators are one of a kind in showing execution and cost which are profoundly focused with custom circuits.

4) It is demonstrated how SAE show execution and cost up to two requests of extent past that of existing delicate processors. To the best of our insight, the SAE is the most noteworthy execution, least cost programming programmable segment on record for FPGA and the first to empower signal and image handling streaming elements with execution and cost comparable with custom circuits.

Single Precision Representation

Single precision, also called as "float" in the C language family, is a binary format that occupies 32 bits (4 bytes) and its significant has a precision of 24 bits (about 7 decimal digits).

The single precision floating point format has an 8 bit exponent and 26 bit mantissa plus a sign bit. It is a 32 bit representation and its bias value is equal to 127.

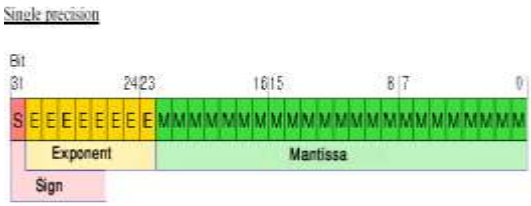

Figure 1.1 IEEE-754 Single Precision Floating Point Format

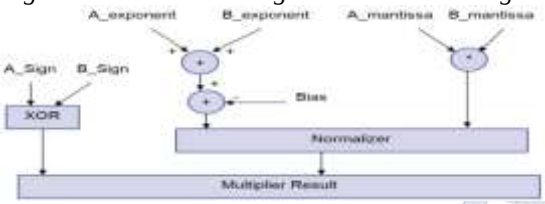

Figure 1.2 Flowchart for Multiplication

Floating point representation is useful to obtain the most accurate and nearest absolute value. Decimal representation of any number is possible and the operations between them are quiet easily analyzed. We have different two different types of representation one is single precession and other double precision representation.

2. The FPGA Processing Element

The FPE instruction set architecture is a RISC load store PE, SIMD, and SISD (i.e., single-path SIMD) variations of which are shown in Fig.1. This architecture composes of program counter (PC), program memory (PM), Register File (RF), Instruction decoder(ID), branch detection, Data Memory (DM), immediate memory, and an ALU in view of the DSP48E in Xilinx FPGA. A COMM (communication) module allows coordinate inclusion/extraction of information into and out of the FPE pipeline. The FPE is extremely lean, fusing just those parts basic to programming programmability.

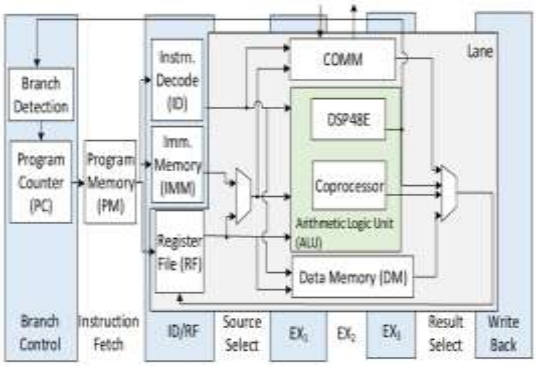

Fig.1 FPE in SISD mode

By guaranteeing total most minimal cost FPE structure, the economies of scale deliver sensational diminishments in multicore asset cost. Be that as it may, this comes at the cost of flexibility: once blended, the FPE does not show an indistinguishable level of flexibility from a general delicate processor in light of the fact that the engineering is exceptionally compelled at configuration time to help the coveted task with most astounding execution and least cost; henceforth, while it might be reprogrammed after blend, it can't empower broadly useful activity in the way of a standard softcore. Likewise, to limit cost while supporting programming programmability, the FPE works under two considerable outright confinements.

1) Processor and ISA: The FPE is a load store processor which can just source non-consistent ALU operands and deliver results to RF.

2) Addressing Modes: The FPE underpins just direct memory addressing.

2.1 Instruction set Architecture:

Instruction Set Architecture (ISA) is a theoretical model of a PC. It is likewise alluded to as design or PC engineering. An acknowledgment of an ISA is called a usage. An ISA licenses different execution that may change in execution, physical size, and cost (in addition to other things); in light of the fact that the ISA fills in as the interface amongst programming and equipment. Programming that has been composed for an ISA can keep running on various executions of the same ISA. These advancements have brought down the cost of PCs and to build their appropriateness. Thus, the ISA is a standout amongst the most imperative deliberations in processing today.

An ISA characterizes everything a machine dialect developer has to know so as to program a PC. What an ISA characterizes contrasts between ISAs; when all is said in done, ISAs characterize the upheld information writes, what state there is, (for example, the principle memory and registers) and their semantics, (for example, the memory consistency and tending to modes), the guideline set (the arrangement of machine directions that includes a PC's machine dialect), and the input/output demonstrate. In order to analyze the flow of the data from one block to another we need a set of instructions which can help us declare the data accordingly.

	Instruction	Function
CTRL	LOOP/RPT	loop/repeat
	BEQ/BGT/BLT	branch if equal/greater/less
	JMP	jump
	GET/PUT	load/push data from/to channel
	GETCH/CLRCH	load data from/clear channels
	NOP	no operation
ALU	MUL/ADD/SUB	multiply/add/subtract
	MULADD/MULSUB(FWD)	multiply-add/subtract (& forward)
	COPROC	coprocessor access
MEM	LD/ST	load/store data from/to memory
	LDIMM/STIMM	load/store data from/to IMM
	LDIAR	updata IMM address register

Table-1 FPE Instruction Set

These instructions are further declared with a particular width and depth of the components for further analysis.

3. Implementation of Streaming Accelerator Elements:

To help achieve the required demands, a novel SAE is proposed. The SAE keeps up independent conduct and a product programmable lean design, however, the requirement is the capacity to stream information into and out of task sources and goals and through the ALU without the requirement for load and store cycles. This gushing takes two structures.

1. Internal: Peer/direct access to RF, DM, COMM, and IMM without the requirement for delay cycles i.e., load store cycles.
2. External: Unbuffered gushing of information from input FIFOs to output FIFOs by means of just ALU.

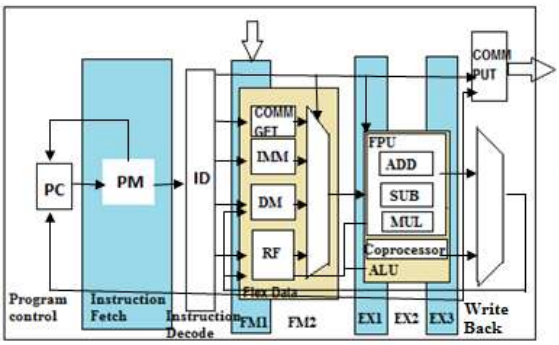

Fig. 3.1 SISD SAE Architecture

As we see in the above figure there are different blocks for defining a path. By using the streaming accelerator elements in SISD approach we see that it possess three distinguish characteristics a) Independent ID block is designed.(Defined block)

b) Flex Data realisation.

c) Comm_get and Comm_put blocks which define a ditinguished path.

In the SAE, ID and FlexData rule full pipeline stages. The ID decides the source/goal of any direction operand/result, with the greater part of the potential sources or goals of information joined in FlexData to enable each to be tended to with break even with inactivity; this flat memory engineering is interesting to the SAE and particular from that utilized by some other softcore processor. Its impact is to lessen the multifaceted nature of getting to every one of the unmistakable operand sources by means of a normal dataflow.

On the off chance that these were not in a similar pipeline arrange, guideline unravel and pipeline administration would be significantly muddled to adjust the information touching base at the ALU with variable idleness. Subsequently, information operands and results might be sourced/created to any of IMM, RF, DM, or COMM with indistinguishable pipeline control and without the requirement for express load and store cycles or guidelines for DM or COMM.

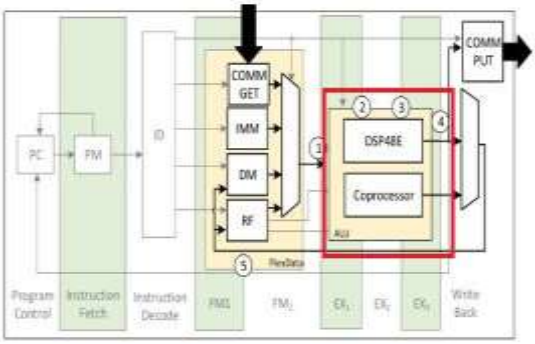

Fig. 3.2 SAE ALU access path

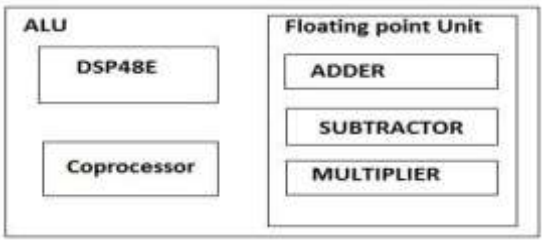

Fig. 3.3 ALU with FPU Internal Block Diagram

In addition , in order to enable unbuffered streaming operation from input to output FIFOs through ALU, synchronous read/write with outer FIFOs is required, with access to ALU in the two bearings.

Keeping in mind the end goal to help this capacity, decoupled COMMGET and COMMPUT segments are conveyed in the SAE inside FlexData. Note that these both live in a similar pipeline arrange and, consequently, fit in with the consistent dataflow pipeline kept up over the rest of FlexData. Also, since all of COMMGET, COMMPUT, DM, RF, and IMM get to unmistakable memory assets (with isolated memory banks utilized inside the SAE and a FIFO utilized per off-SAE correspondence channel), there is no memory transfer speed coming about because of decoupling .

3.2 Instruction Declarations:

1. RPT repeat_num, repeat_count
2. BEQ/BGT/BLT offset, opA, opB
3. JMP offset
4. GET/PUT addr, dest
5. MUL/ADD/SUB dest, opA, opB
6. LD/ST mem_addr, reg_location
7. LDIMM/STIMM dest, source

These are some configurations of the instructions that are to be declared in the above pattern. Their width and depth are consired as shown the below table.

To permit include (yield) of information from (to) the suitable source (goal), both the physical source segment (RF, COMMGET, COMMPUT, DM, and IMM) and the fitting locations inside each (i.e., memory area or correspondence channel) must be transferred inside the guideline. To oblige this, SAE ALU directions are communicated in the accompanying arrangement:

INSTR dest, opA, opB

where INSTR is the guideline class, dest identifies the outcome goal/output, and opA and opB recognize the source operands.

Table 3.2 Instruction coding of ALU

Op	Source/Sink	x
Rx	RF	Register location
&x	DM	DM address
^x	COMMGET/COMMPUT	IPC channel no.
x	IMM	Constant value

The conceivable encodings of every one of dest, opA, opB and the goal are depicted in Table 3.2. As depicted in Table 3.2, all of RF, DM, COMMGET, and COMMPUT are tended to straightforwardly by means of the total locations of the source/sink registers, memory areas, or outside channel, individually.

Steady operands are hard-coded into the direction and IMM areas distributed by the constructing agent. The sizes of the address fields in the final directions are powerfully produced to coordinate the configuration of the processor and program—i.e., five bits are relegated for enroll area for a 32-component RF, six bits for a 64-component RF, et cetera. Guideline fields for RF, DM, COMMGET, and COMMPUT addresses are correspondingly decided at compile time by the SAE constructing agent as depicted in table 4.1.

The considerations taken for the instructions are listed below with their opcode,with, considered depth and bitwise declarations for inputs and outputs.

The exceptionally adaptable nature of the FPE is kept up in the SAE, with the expansion of parameters specific to the configurable FlexData.

Table 3.3 Configuration parameters of Flexdata

Parameter	Meaning	Values
data_ws	Data Wordsize	16, 32
imm_depth	IMM Capacity	$\mathbb{N} \in [1, 2^{32} - 1]$
dm_depth	DM Locations	$\mathbb{N} \in [1, 2^{32} - 1]$
rf_depth	RF Capacity	$\mathbb{N} \in [1, 2^{32} - 1]$
pp_depth	Pipeline Depth	0, 1, 2
dm_thr	DM Threshold	$\mathbb{N} \in [1, 2^{32} - 1]$

The sizes of FlexData's constituent segments can be defined presynthesis such to empower activity specific cost enhancement by means of the configuration parameters depicted in Table 3.3.

These configuration parameters empower generous customization: data_ws controls the information word estimate for the SAE, while the profundity of each the IMM, DM, and RF is set by imm_depth, dm_depth, andrf_depth, separately. Furthermore, the quantity of physical defer cycles embedded by FlexData is defined pre-union by means of pp_depth: any of zero, maybe a couple cycles might be embraced.

There are two certain issues of note concerning configuration parameters. For the situation where any of imm_depth, dm_depth, andrf_depth are zero, the related segment (IMM, DM, and RF) is missing from the incorporated form of FlexData; this permits irrefutably the base arrangement of assets required to understand an offered task to be acknowledged, limiting expense, with the additional benefit of decreasing the span of the FlexData multiplexer whose size is configurable as per similar parameters. Moreover, the nature of individual parts can change.

For example, the DM segment is configured to permit realization as either distributed RAM (DisRAM) acknowledged in the FPGA programmable rationale LUTs, or by utilizing the devoted on-chip BRAM. The limit for this choice is configurable as dm_thr. Specifically, if the DM limit does not surpass dm_thr, at that point it will be acknowledged utilizing DisRAM; else, it will be acknowledged utilizing BRAM.

This configurability enables this significant structural choice to be made in an application specific way. For the rest of this paper, this limit is taken to be 256 words. The SAE shapes the essential building square of huge scale streaming multicore structures in a way like that of the FPE—it is a configurable-width SIMD with coordinate outer correspondence capacity from which systems might be created by means of FIFO lines.

3.3 Streaming processing block:

The efficiency increments coming about because of the spilling idea of the SAE are profoundly reassuring, yet in numerous activities, tending to modes other than basic direct tending to are fundamental; for example, an ordered guideline breakdown for the augmentation of two 32 × 32 matrix is appeared in Fig. 3.3.

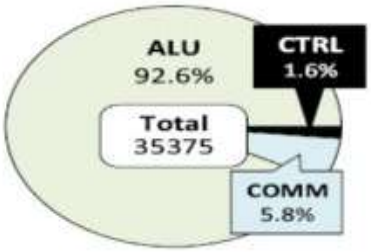

Figure 3.3 SAE Itemised MM operations

These report an indistinguishable high yet in addition to a great degree expansive projects—35375 directions for MM. This places an overwhelming interest on FPGA memory assets for PM.

These extraordinary sizes take after from the limitation to coordinate tending to, which directs that the quantity of guidelines is limited beneath, by the quantity of ALU activities; for MM, this deciphers a substantial number of directions. MM out a given activity commonly, over and again, on little subsets of the info information at consistently divided memory areas prompting profoundly redundant conduct on routinely dispersed memory areas

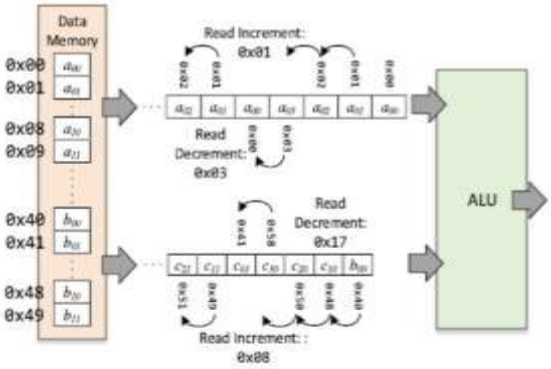

Fig. 3.3 Matrix multiply operand addressing of SPE block

For example, consider square MM of two networks A ∈ Rm×n and B ∈ Rn×p when m = n = p = 8 through four 4×4 submatrices. Expecting that An and B are put away in DM adjacently and in push major order and that C is derived in push major order, and the operand memory get to is appeared in Fig. 4.4. To register a component of a submatrix of C, the inward result of a four-component vector of bordering areas in An (a line of the submatrix) and a four-element vector of components separated by eight areas in B (a segment of the submatrix) is framed.

A while later, either or both of the line of A or segment of B are increased to determine the following component of C, before the task continues to the following submatrix. The subsequent memory gets to are profoundly unsurprising: a normal rehashed increase along the lines of An and the segments of B and intermittent realignment to another line of An or potentially segment of B rehashed numerous circumstances previously realigning for resulting submatrices.

```
repeat(k; M){
 repeat(j; P){
     repeat(l; N){
        C[k][j]+=A[k][l]*B[l][j];
     }     }     }  }
```

Table 3.4 Streaming matrix multiplication code

This conduct is minimally spoken to utilizing a circle based code—for instance, MM pseudocode is given in table 3.4. Each rehash activity understands a predefined number of cycles over its body, with affine mixes of the iterators j, k, orl ordering the operands. To help this very minimal articulation of conduct for activities, for example, MM, the SAE needs to consolidate two offices: rehash style conduct with the capacity for a solitary guideline to address pieces of memory
at consistently dispersed areas when summoned various circumstances by a repeat. While repeat compose directions are clear in regular processors, there is no record of a softcore processor for FPGA which understands these abilities and all things considered their acknowledgment inside the stringent cost limitations of the FPGA accelerators.

A few chips enables parts of the program memory to be adjusted in pieces (sections), yet you can't store factors in the program memory. It is regularly conceivable to store constants - i.e. introduced factors that you don't change - in the program memory.
Your PC likewise has data memory and program memory. However, the program memory is little in the PC - it is only for

capacity of the boot messages you see when the PC boots, and (frequently, yet not generally) the configuration pages where you characterize on the off chance that you have a floppy introduced, if the PC should bolster a USB console and so on.

Program is guideline what CPU executes, information will be data that program utilizes for customization and capacity of how the program should convey those directions.

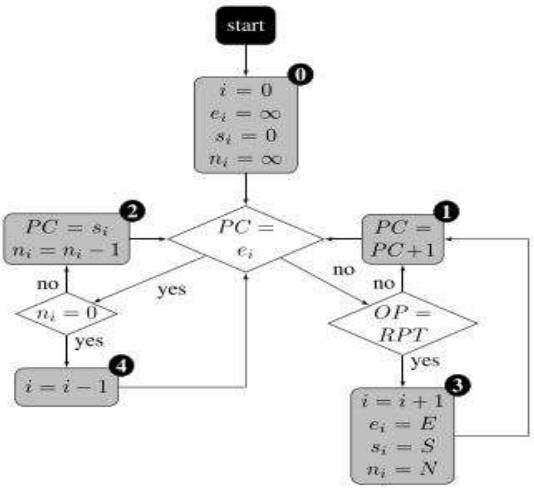

Fig. 3.6 Flow of PCM

The SAE is enlarged with the capacity to perform rehash write conduct. This implies dealing with the PC, to such an extent that in case of such a guideline, the body of the rehash explanation is executed a number of times. This task is fulfilled by a PC manager (PCM). The structure of the SAE PC and PCM and the conduct of the PCM are appeared in Fig. 4.5. The PCM controls the refresh of the PC given its past esteem and the guideline referenced in PM given snippets of data—the begin and end lines of the body articulations to be rehashed S and E, the quantity of reiterations N. These are encoded in a RPT direction added to the SAE guideline set. These guidelines are encoded as RPT N S E.

The PCM mediates the PC to guarantee the right number of redundancies of the body proclamation and to help the development of settled rehash activities by establishing the flowchart in Fig. 10. Specifically, for a n-level home, it keeps up n + 1-component arrangements of measurements, with an extra component added to help infinite reiteration of the best level program, thought to be a verifiable infinite rehash guideline. For layer I of the circle settle, the begin line, end line, and number of redundancies are put away in component I +1 of the rundowns s, e, and n, individually.

In all cases s0 = 0, e0 =∞and n0 =∞to speak to the begin line, end line, and number of reiterations of the best level program [Fig. 3.6].

Every time a rehash guideline is experienced I, the present record into s, e, andn is increased and the estimations of the new component introduced utilizing S, E, andN from the decoded direction in (3).

General PC refreshing at that point continues (1) until the point when either another rehash guideline is identified or until the point when ei is experienced.

In the last case, the quantity of cycles of the present explanation is decremented (2), or if ni = 0, the majority of the emphasess of the present repeat proclamation have been finished and control of the circle settle returns to the past level (4).

The PCM activity depicted in Fig. 3.6 was acknowledged utilizing behavioral VHDL and blended. The cost of the essential PCM segment is 36 LUTs, a cost which should just be acknowledged in situations where it can empower a significant cost/execution benefit, for example when it can considerably diminish program measure and, along these lines, PM cost, with the sparing ideally exceeding the cost of the PCM.

To permit least cost for each application, the PC is configurable by means of the parameters recorded in .

The pcm_en dictates whether the PCM is incorporated into the blended design or else it takes a Boolean esteem. For the situation where a PCM is incorporated, the greatest profundity of circle settle is configurable by means of pcm_en which can take, theoretically, any esteem. In that capacity, the PCM perhaps included or excluded and hence, imposes no cost when it isn't required; besides, when it is incorporated, its cost can be tuned to the current application by changing the most extreme profundity of circle settle.

4. Simulation and Synthesis Results

Figure 4.1 is the simulation wave form of project, Here the instructions are given as input through input variables and top module is generated which shows the floating point operations carried out. By observing the the synthesis results we can say that the speed of operation is increased.

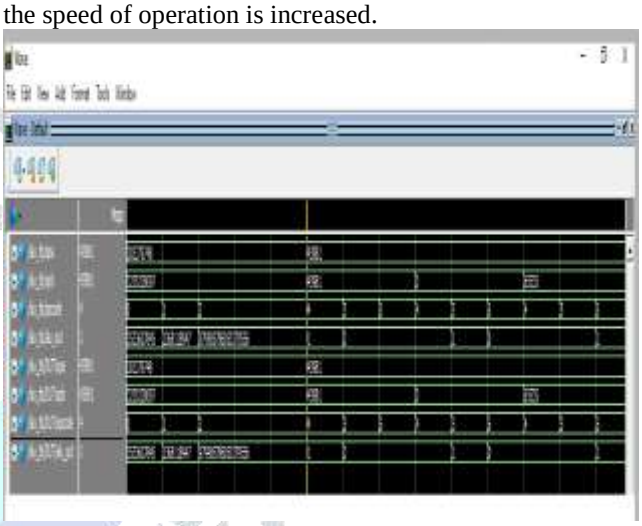

Figure 4.1 Wave form of Top Module for floating point operations

Figure 4.2 is the simulation wave form of memory module .

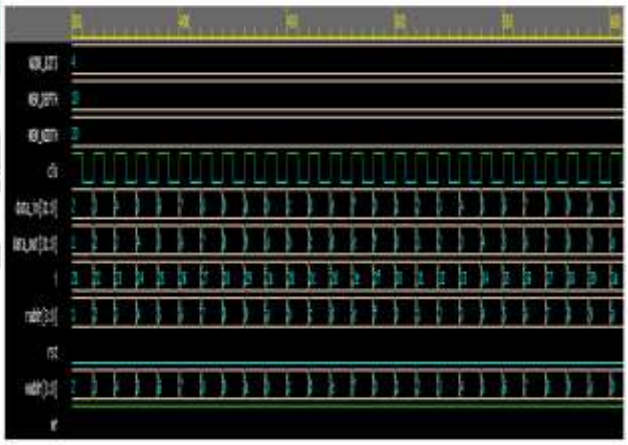

Figure 4.2 Wave form of memory module

5. Conclusion

Soft processors for FPGA experience the effects of generous degradation in performance and cost with respect to custom circuits. The SAE implemented uses a technique which allows for maximum efficiency which is compatible with the custom circuits. We are implementing this for floating point numbers which perform arithmetic operations like addition, subtraction, and multiplication for matrix multiplication. These empower efficiency routinely more than 90% and execution and cost which are practically identical with custom circuit accelerators and well ahead of time of existing soft processors. Also, it is indicated how SAE-based MM (Matrix Multiplication) accelerators offer changes in asset/cost by up to three degrees of magnitude. To the best of our insight, these capacities are extraordinary, for FPGA, as well as for any semiconductor innovation.

Advantages:

- It is more performance efficient compared to softcore processors and achieves almost as good performance as that of custom circuits.

Future scope:

As future work, however furthermore the FPGA accelerators might be utilized to additionally facilitate the outline procedure. For instance, programmability of the SAE implies that it might likewise be utilized as a memory controller to execute custom memory access.

References

[3] H. Y. Cheah, F. Brosser, S. A. Fahmy, and D. L. Maskell, "The iDEA DSP square based delicate processor for FPGAs," ACM Trans. Reconfigurable Technol. Syst., vol. 7, no. 3, Aug. 2014, Art. ID 19

[2] A. Severance and G. Lemieux, "VENICE: A reduced vector processor for FPGA applications," in Proc. Int. Conf. Field-Program. Technol. (FPT), Dec. 2012

[3] X. Chu and J. McAllister, "Programming defined circle unraveling for FPGA-based MIMO location," IEEE Trans. Flag Process., vol. 60, no. 11, Nov. 2012.

[4] P. Wang and J. McAllister, "Delicate center stream processor for sliding window applications," in Proc. IEEE Workshop Signal Process. Syst. (Tastes), Oct. 2013

Author Profile:

Chinta Sravani She received Bachelor's Degree in 2015 from Electronics and Communication of Engineering from Institute of Aeronautical Engineering. She is pursuing M. Tech in VLSI System Design from CMR Institute of Technology

Dr. Prasad Janga He received Bachelor's from V.R. Siddhartha Engineering College and has accomplished Master's Degree from SITAMS. He is pursuing PhD from (NIU)-Delhi. He has an academic experience of 9 years in teaching field and is working as Associate Professor in CMR Institute of Technology.

Mrs. S. SriBindu She is working as an Associate Professor in CMR Institute of Technology.