Implementation of a 32 – bit RISC processor with floating point unit in FPGA platform

Sushma S1, Smruthi Koushika Ravindran2, Pavan Rajendar Nadagoudar3, P.Augusta Sophy4*

1,2,3,4School of electronics and Communication Engineering, VIT Chennai.

augustasophyt.p@vit.ac.in

Abstract. With the ever-growing trends in the latest technologies, digital signal processing is one of the known platforms which is used for many applications. RISC processor is the main element in all high-speed digital signal processing applications. The aim is to develop a RISC processor by including a single precision floating point unit in the instruction set along with other instructions and a complex multiplication instruction as it is commonly used in the DSP application. The floating-point unit is written according to the IEEE 754 standard and to introduce Vedic mathematics concept to compare and estimate the performance in terms of delay. The idea is to provide an optimized floating-point arithmetic unit and thereby provide an additional feature for the RISC processor instruction set. By using the Vedic multiplier, the delay is reduced by 55% compared to the normal array multiplier. The entire design is coded in Verilog HDL simulated on Xilinx ISE 14.7 platform.

1. Introduction
As the year passes by, the demand for technology with high speed and low power has become inevitable. This gives us the scope for improvising the design for both software and hardware part of the technology. The never-ending greed for refining the existing devices and coming up with an optimum solution in terms of area, power and speed is the target of many researches over the past years. In this context, the processor which is the critical part of digital signal processing applications has also gone through many changes.

The processor is something that processes the information through set of instructions and produces the outcome required by the user. In simple terms, its function is to performs the fetch, decode and execution of the instruction. Among the processor architectures, usage of the RISC (Reduced Instruction Set Architecture) processor has been increasingly used in the recent years. CISC architecture consists of more instructions and the addressing modes compared to the RISC architecture. In the CISC architecture, the time taken for the decoding of information takes time sometimes it processes and evaluates the codes which just waste the clock cycle. This factor turns out to be a disadvantage for the CISC architecture and there is problem related to the stalling of the clock cycles. In contrast, RISC processor is generally having small instruction set with few numbers of instructions which can be specific for the application. Here the instruction is executed in a single cycle for the non-pipelined architecture. Even though RISC architecture has a complex complier design, the execution time is reduced tremendously and the problem of stalling in the CISC architecture is also reduced to an extent [8]. Mostly the RISC processor is used for the application of high end like the telecommunication industry and in the processing of video and image. Currently the video and image
processing play a vital role in almost all the application of the current trend. The processor speed and the power consumption turn out to be a very big concern as the transistor size is reducing and the density of the number of transistors on the IC is increasing [9]. This calls out for a RISC processor which can provide space for more application and to reduce the delay for the processing of instruction.

The objective of this work is to develop a 32-bit RISC processor along with an instruction set which includes the floating-point unit and the complex multiplication unit. These units are included as they are more commonly used in the digital signal processing application. These additional units provide platform for new applications. To improvise the performance of the floating-point unit, Vedic mathematics concept is included for the multiplication as general fact that multiplication takes more time for execution compare to other instructions. By including the Vedic concept, the comparison is given in terms of delay.

The summarization of the process that is carried out in this paper

- Initial the architecture for the 32-bit RISC processor is designed with two major blocks as the data path unit and the control unit. To design the control unit the FSM concept is used
- The instruction set is framed based on the number of instructions required and the addressing modes
- The instruction queue, register, decoder, register bank and program counter are designed. Floating point unit and the complex multiplication unit is coded
- Vedic mathematics concept is included in the floating-point unit to provide an improvisation in terms of delay
- The results are measured and tabulated using the Xilinx software and the Verilog code is used for designing

2. Literature Survey

The survey study is focused on the designing of the RISC processor unit for all the application related to convolutions. In this paper, Sakthikumaran and co-authors introduced the Wallace tree multiplier in the ALU unit to reduce the power consumption and improvise the speed of the processor [1]. The entire design is designed using the Verilog HDL and for synthesis purpose synopsis tool with 90nm CMOS technology is used. One of the main features designed is to execute the operation in single cycle. The architecture of this 16-bit instruction unit includes the Winograd algorithm modified version for the better performance. The designed processor is promised to be useful for all the convolution related application.

The focus of the paper is on the power utilization of the processors that are used for the general purpose. Author Ghaturle, Mrudul S and all employed the booth multiplier concept for reducing the power and thereby increasing the battery life of the devices [2]. In order make both ends meet between the area and speed the design is done based on the pipelined approach. In this way, the problem of synchronization is also mitigated.

The main aim of this paper is to design a processor with high speed so that it will be feasible for the application of low throughput algorithms of cryptography. One of the important features of today’s industry is security so the idea of author Nima, Karimpour Darav, and all is to provide high performance processor that performs processing of data with high speed [3]. The processing of the data includes the encryption and decryption of the information. This architecture is meant to reduce the delay taken for the processing of the instruction related to the cryptography algorithms.

The objective was to explore floating point unit with the IEEE standard 754. The authors of the paper Even, Guy, and Wolfgang J. Paul. basically, identified the exception in the rounding off part during the overflow and the underflow in the floating-point unit and formulated a mathematically expression that can mitigate the errors in the result. Representation of the bit level is avoided, and the arguments are used to provide the clarity in the proposed algorithm. The detecting of the loopholes in the exception is not an easy task and the authors proposed the mathematical expression which can equalize the definition of exceptions. The easier identification of the exception is done in the rounding unit [4].
The paper focuses on using the Vedic mathematics in the complex multiplication and thereby reducing the delay caused due to the performance of the complex multiplier in the digital signal processing applications. According to author Tomaskar, Rupa A. and all have introduced one of the best sutras that is always used for the purpose of multiplication as to reduce the number of multiplication and addition is Urdhav Tiryakbhyam [5]. It is concluded that the sutra is best suited for the high-speed application of the DSP but mostly restricted to lesser number of bits for the implementation.

3. Design of the Architecture

The design of the 32-bit RISC architecture is being discussed in detail in the below passages

Proposed Architecture
The architecture is designed with 32-bit instruction word format and non-pipelined architecture as shown in the Figure 2. The architecture is divided into two segments control path unit and the data path unit. The data path unit consist of the program counter, instruction queue, instruction fetch, instruction decoder, floating point unit, ALU, mux, register bank. The instruction set consists of totally 38 instructions which include the floating-point instructions and the complex multiplier instruction. The instruction word format is shown in Figure 1

| OPCODE | ADDRESSING MODE | SOURCE REGISTER | DESTINATION REGISTER | IMMEDIATE NUMBER |
|--------|-----------------|----------------|----------------------|------------------|
| [8 BITS] | [1 BITS] | [4 BITS] | [4 BITS] | [16 BITS] |

Figure 1. Instruction Word Format

Figure 2. Proposed RISC Architecture
3.1. Control Unit
The control unit as shown in Figure 3 is designed using the FSM logic. The inputs to the control unit are the global clock and global reset. The control signals from the control unit is sent to all the blocks in the data path unit.

![Figure 3. Control Unit](image)

3.2. Program Counter
The program counter basically provides the address from which the instruction must be fetched, and it sends the address to the instruction queue. The program counter decides whether it must increment to the next address or jump to some address location depending on the control unit signal. When the global reset is given the program counter address is reset to 0. The Figure 4 represents the program counter.

![Figure 4. Program Counter](image)

3.3. Instruction Queue
The instruction queue basically fetches the instruction address from the program counter and the corresponding 32-bit instruction data-in is fetched. The control unit as shown in Figure 5 sends the signal whether to read or write the instruction in the queue. At first the instruction is written in the queue and then it is read from the queue and given to the output i.e. to the instruction register.

![Figure 5. Instruction Queue](image)
3.4. Instruction Register
This instruction register fetches the instruction word from the instruction queue. The instruction register consists of the register file which has the source address and the destination address of the data. It is said that the data will be received from the user during the run time and store at the locations. This instruction register output as shown in Figure 6 is given to the decoder for further processing of the instruction word that has been fetched.

![Figure 6. Instruction Register](image)

3.5. Instruction Decoder
The instruction decoder is used to decode the opcode and determine what operation to be performed in the arithmetic unit or floating-point unit. This decoder unit also sends the enable signal for the unit that needs to be operated. The Figure 7 represents the instruction decoder.

![Figure 7. Instruction Decoder](image)

3.6. Register Bank
The register is designed with 16x32 memory, so that the values of the data and the output is stored in this location. The Figure 8 represents the Register bank.
3.7. **ALU Unit**

The ALU gets the data from the register bank and performs the operation corresponding to the opcode that has been decoded by the instruction decoder. In the ALU all the logical operations like AND, OR, XOR, fixed point arithmetic operations like addition, subtraction, multiplication, shift operations and the increment operations. The output of the ALU is stored in the register bank at the destination location decoded by the instruction decoder.

3.8. **Floating point Unit and Complex Unit**

The floating-point addition/subtraction, multiplication and division are performed according to the IEEE standard 754. According to the standard the 32-bit single precision input data is divided into 1 bit of sign, 8 bits of exponent and 23 bits of mantissa part.

**Floating point algorithm for ADD/SUB.** Initial the exponent of both the input data is equalized. After that the mantissa bits are add/sub using the ripple carry adder. An enable symbol is provided to decide whether to do addition or the subtraction is decided. The mantissa obtained after the processes is normalized [4]. Exponents are added or subtracted corresponding and finally added to the result.

**Floating point algorithm for Multiplication.** For the multiplication, initially it is checked whether any one of the inputs is zero, if it is zero then the answer 0 is send to the output directly. Otherwise, the sign bit of both the inputs are xored; the mantissa is multiplied using the normal array multiplier. The exponent can be both positive and negative depending on the values given. So, the exponents are added, and the resultant is subtracted from bias. For the single precision floating point the bias value is $(127)_{10}$ as per IEEE standard [4].

In this instead of going for a normal array multiplier for the mantissa calculation, Vedic multiplier is employed and inside the Vedic multiplier for addition purpose carry look ahead adder is used. 32 x 32 Vedic multiplier is designed to calculate the mantissa multiplication part [7]. The most common used Urdhav Tiryakhbyam Vedic multiplier concept is used. This concept is employed to reduce the delay consumed by the normal array multiplier. Generally, in any system this multiplier unit consumes more power compared to other units. So, the aim is to reduce the delay caused by the multiplier unit.

**Floating point algorithm for Division.** In the division also initially, it is checked whether any one of the inputs is zero. The output tends to infinity, if the divisor input is given as zero, if numerator is zero, output is zero and if both are zero, then the output shows invalid. Otherwise, first the MSB bit i.e. the sign bit is calculated by xoring, the mantissa is divided by padding zeros that are required for the calculation purpose [4]. The exponent can be both positive and negative depending on the values given. So, the exponents are subtracted, and the resultant is added from bias. For the single precision floating point the bias value is $(127)_{10}$ as per IEEE standard.

3.9. **Complex Multiplier**

The complex multiplier is constructed using only two multipliers, as shown in Figure 9, instead of four multiplications [5].

Say, $S = P \ast Q = (P_{re} + jP_{img}) \ast (Q_{re} + jQ_{img})$
\[ S_{re} = Q_{img} \times (P_{re} - P_{img}) + P_{re} \times (Q_{re} - Q_{img}) \]

\[ S_{img} = Q_{img} \times (P_{re} - P_{img}) + P_{img} \times (Q_{re} + Q_{img}) \]

Here, as the number of multipliers is reduced so delay can be reduced to an extent in the processor design.

Figure 9. Complex Multiplier

4. Results
The results are obtained after the synthesis of the architecture using the Xilinx tool. The obtained results are tabulated, and the graph is plotted. The Table 1 shows the area, delay, and power of the individual module in the proposed RISC architecture. Table 2 shows the comparison of the delay between the floating-point array and Vedic multiplier. The delay of the Vedic circuit is less than the array multiplier by 55%. This will be very helpful in the DSP applications.

Table 1. Area and Power of the Architecture blocks

| Module Name      | Area Cell Count | Area Total Area | Power Leakage (nW) | Power Dynamic (nW) | Data Arrival Time (ps) |
|------------------|-----------------|-----------------|---------------------|--------------------|------------------------|
| Program Counter  | 25              | 492.3           | 13.5                | 97624.9            | 1032                   |
| Instruction Queue| 953             | 42478.2         | 1326.6              | 14623630.6         | 1001                   |
| Instruction Register| 64          | 2661.1          | 88.3                | 623741.8           | 183                    |
| Instruction Decoder| 42           | 2518.1          | 80.3                | 974569.4           | 251                    |
| Execution Unit   | 3534            | 84673.5         | 3002.4              | 10177463.3         | 196023                 |
| Register Bank    | 954             | 42458.2         | 1325.4              | 14601076.2         | 1005                   |

Table 2. Comparison of Floating – point multiplier delay

| FPGA Family | Array Multiplier logic path delay (ns) | Vedic multiplier logic path delay (ns) |
|-------------|----------------------------------------|----------------------------------------|
| Virtex6     | 5.123                                  | 2.500                                  |
The table 3 gives the comparison of area and delay of the complex multiplier, which is constructed by using two, three, four multiplier architecture.
CM2 – complex multiplier architecture with two multipliers
CM3 – complex multiplier architecture with three multipliers
CM4 – complex multiplier architecture with four multipliers

| Architecture     | Occupied Slice | No. of slice LUTs | Delay (ns) |
|------------------|----------------|-------------------|------------|
| CM2              | 12             | 48                | 7.953      |
| CM3              | 56             | 224               | 13.708     |
| CM4              | 32             | 128               | 10.574     |

In the above Table 3 shows that the complex multiplier constructed using two multipliers has reduced area compared to the other models.

Figure 10. Comparison between Complex Multiplier architectures

5. Comparison
The research work basically aimed at providing a RISC processor along with the floating-point unit and the complex multiplication as one of the instructions. The floating-point unit is optimized in terms of delay by using the Vedic multiplier concept; the critical path delay of the logic path is reduced around 55% compared to the array multiplier. The complex multiplier unit which is included is also constructed using the two multipliers architecture so that the area consumed by the complex multiplier
is also reduced compared to other architectures. The architecture is designed in such a way that it includes additional instruction with less delay utilization.

References

[1] Sakhikumaran, Samiappa, Salivahanan S and VS Kanchana Bhaaskaran 2011 16-Bit RISC processor design for convolution application Proc. Int. Conf. on Recent Trends in Information Technology (ICRTIT)

[2] Ghaturle, Mrudul S and R D Kadam. 2017 Review Paper on 32-Bit RISC Processor with Floating Point arithmetic. Int. Research Journal of Engineering and Technology (IRJET)

[3] Nima, Karimpour Darav, et al. 18th -20th March 2012 CIARP: Crypto instruction-aware RISC processor Proc. IEEE Symp. on Computers & Informatics (ISCI)

[4] Even, Guy, and Wolfgang J Paul 2013 On the design of IEEE compliant floating-point units IEEE Trans. on Computers 49 5 pp 398-413

[5] Tomaskar, Rupa A and Gopichand D Khandale 2014 FPGA Implementation of Complex Multiplier Using Urdhva Tiryakbhyam Sutra of Vedic Mathematics Int. J. of Engineering Research and Applications, ISSN: 2248 9622 4 5 pp 1-5.

[6] S. Palekar and N. Narkhede, 2016 . High speed and area efficient single precision floating point arithmetic unit, IEEE Int. Conf. on Recent Trends in Electronics, Information & Communication Technology (RTEICT) pp. 1950-1954.

[7] Y. S. Rao, M. Kamaraju and D. V. S. Ramanjaneyulu, 2015 An FPGA implementation of high speed and area efficient double-precision floating point multiplier using Urdhva Tiryagbhyam technique Conf. on Power Control, Communication and Computational Technologies for Sustainable Growth (PCCCTSG) pp. 271-276

[8] S. Palekar and N Narkhede 2016 32-Bit RISC processor with floating-point unit for DSP applications IEEE Int. Conf. on Recent Trends in Electronics, Information & Communication Technology (RTEICT) pp. 2062-2066.

[9] Galani Tina, G, Riya Saini, and R D Daruwala 2013 Design and Implementation of 32 bit RISC Processor using Xilinx Int. J. of Emerging Trends in Electrical and Electronics (IJETEE) ISSN 2320-9569 5 1 p 18-24.

[10] Joseph, Neenu, S Sabarinath, and K Sankarapandiammal. 2009 FPGA based implementation of high-performance architectural level low power 32-bit RISC core Int. Conf. on Advances in Recent Technologies in Communication and Computing