9th IEEE International Conference on Communication Systems and Network Technologies

Implementation and Extension of Bit Manipulation Instruction on RISC-V Architecture using FPGA

Vineet Jain*, Abhishek Sharma*and Eduardo Augusto Bezerra‡
*Dept. of Electronics and Communication Engineering
The LNM Institute of Information and Technology , Jaipur,
INDIA Email: {17uec135 , abhishekshar3ma @lnmiit.ac.in}
‡Dept. de Engenharia Elétrica Campus Trindade
Universidade Federal de Santa Catarina, Florianopolis, SC,
BRAZIL Email : Eduardo.Bezerra@ufsc.br

***Abstract*—Consumer electronic computational device requires an efficient system, having minimal Cost and Power Consump- tion, with high energy efficiency and security. RISC-V is a widely accepted Instruction set architecture (ISA) due to its compatibility with direct native hardware implementation rather than simulations and has support for extensive ISA extensions with specialized variants. Bit Manipulation Instructions (BMIs) were introduced by ARM and Intel to improve the runtime efficiency and power dissipation of the program although RISC- V ISA is popular it currently supports only two basic BMIs.**

This paper presents a simplified architecture of a fully Synthesiz- able 32-bit processor "bitRISC" based on the open-source RISC- V (RV32I) ISA and also introduced two new RISC-V BMI's and implemented it on our designed processor, targeted for low- cost Embedded/IoT systems to optimize power, cost and design complexity. The "bitRISC" is a single cycle processor designed using Verilog HDL and our simplified architecture and is further prototyped on "ZedBoard" FPGA.

***Index Terms*—RISC-V, RV32I ISA, Micro-architecture, FPGA, Bit Manipulation, microprocessor**

I. Introduction

Consumer electronic computational devices generally require a high efficient system, having minimal Cost and Power Consumption, with improved energy efficiency and security. There is trade-off between Cost and Performance. To achieve optimized Performance for an IOT/Embedded system, ARM based processor has been broadly used, as it requires typically less amount of Digital logic than a complex instruction ar- chitecture does, thereby increasing its performance, reducing complexity, cost, power usage, heat dissipation, etc. Embed- ded/IoT market is dominated by the x86 and ARM processors [2]. Many RISC based architecture has been adopted and are available in different flavors for specific optimized applica- tions, targeting low cost and low power embedded system devices. Computer Architecture has been an area of research over few decades resulting in optimization and formation of new efficient ISA.

RISC-V is a new Open source instruction-set architecture (ISA) that has become freely available for industry and academics [1]. It is compatible for implementation on the native hardware directly rather than simulations with support for extensive ISA extensions and specialized variants. The RISC- V ISA comprises of 4 base integer ISA, which are minimal adequate instructions set, designed to support (32,64,128 bits) encoding and for Embedded Application. There are also different optional Standard Extension ISA to support general purpose software and hardware development.

Previously, Intel and ARM have introduced Bit-Manipulation Instruction (BMI), which were extensions to x86 and ARM ISA for microprocessors, with the purpose of improving the speed of bit manipulation operation [3]. The BMI's are crucial for improving power efficiency and performance of the processor because it reduces the complexity of the operations, which would have otherwise created a complex hardware, utilizing more resources and power. This becomes more important to consider while targeting for Low-Cost Embedded/IoT devices when complex operations like (square root, etc.) need to be implemented. In the field of Machine Learning, Steganography and cryptography, the algorithms generally works on binary data and usually use operations like bit-rotation, grouping, shifting, etc. the use of BMI here can considerably reduce the dissipated power and utilization of resources on the device, hence optimizing it.

This Paper presents, two new RISC-V BMI Instruction as an addition to official RISC-V ISA and is implemented, by using a Simple and efficient 32-bit "bitRISC" RISC-V processor architecture with RV32I base-instruction set implemented. The paper also presents the way to embed BMI's to current RISC- V ISA and also to the "bitRISC" processor for their execution. The "bitRISC" processor is a single cycle, single-core archi- tecture with all RV32I instructions. Its application domains can be real-time embedded systems, IoT, Signal processing, etc. As the target in our mind was, low-cost electronic systems, our focus is to enhance the architecture for power, cost, and design intricacy at the expense of having precise timing constraints.

978-1-7281-4976-9/20/$31.00 ©2020 IEEE
DOI: 10.1109/CSNT.2020.32

Also, the new BMI discussed is Implemented on the "bitRISC" processor keeping the same target in mind. The processor is used as an accelerator, prototyped on FPGA (ZedBoard), to increase the efficiency for complex operation on our new BMI processor implemented. The article is organized as follows. Section II Introduces RV32I in brief, the comparison between different ISA and the related work in the field. Section III describes processor architecture and its working. Section IV presents our new RISC-V BMI's and their applications in various domains. Section V shows the results, simulation, and utilized FPGA resources. Section VI ends the paper with a conclusion and some remarks on future work.

II. LITERATURE REVIEW

RISC-V ISA is world-wide accepted due to its features. In [4] analysis on base ISA amongst MIPS, RISC-V, OpenRISC, ARM, etc. were done. RISC-V ISA provide features like no other ISA do, with no branch delay and condition slots ,it avoids over-architecting and contains support for optional variable-length instruction encoding space for improving per- formance, static code size, and energy efficiency[1]. RV32I was designed to minimize the hardware requirement of a system. There are in total 6 instruction format in RV32I showing immediate variants: R, I, S, B, U and J [1]. Each format uses different encoding, in total RV32I has 40 Unique instruction. Every instruction is 32-bits wide have byte addressing. [5] also implemented the RV32I in its implementation but uses different architectures. Intel and ARM introduced their BMI's in around 2012-13[3]. Deep analysis on x86 BMI and their advantages were done, together with a proposal of new RISC-V BMIs [6]. They used barrel shifters for *rotl*, *rotr* operations and used divide and conquer algorithm for *clz* and *ctz* instruction, which when implemented on RoketCPU shows significant reduction in time and power utilization. Our work introduces the new type of BMI's that can be used is various fields. [9] also introduced application-specific instructions and their bit manipulation unit (BMU), to efficiently support in DSP applications.

III. METHODOLOGY

In most embedded systems for small applications, a 32-bit processor is used as it reduces the cost and complexity. For foundational performance analysis the single cycle design was chosen. The following subsections describe the top-level view of the processor and micro-architecture of this design in detail.

A. Top module view

The single-cycle processor implemented is divided into three parts; Datapath Unit, Control Unit and Memory. The Datapath Unit forms a skeleton and contains the architecture of the processor. Control Unit acts as a decoder and gives an idea about the working of various components on a new instruction, it acts as a brain for the processor. Memory is divided into two parts Instruction Memory, which contains a set of instructions to be executed and Data Memory, which acts as a storage element.

The Datapath Unit contains a dedicated adder for calculation of the value of PC (program counter) for branch and jump instruction and incrementing PC at every clock cycle. The Register file module contains an array of 32 general-purpose registers of width 32-bits each from R0 to R31. R0 is hardwired to the ground or logic zero i.e. any operation with R0 will result in R0 equals zero. The ALU module operates on its inputs according to the value of *ALUFunct* provided by the Control Unit. Control Unit generates the control signal according to instructions fetched on the same cycle.

ALU deals with operations only related to R-R(register-register) and R-I(register-immediate) operations. For branch instructions, the ALU calculates the condition for the branch and gives the output to the control unit, which controls the next PC value according, however, the ALU does not involve in their address calculation, as separate adders are dedicated to the same.

B. Micro-Architecture and its Working

At every clock cycle, PC is incremented by 4(because of byte addressing) or according to values of *pcsel* in Fetch and Control block, indicating the address of next instruction to be fetched from program memory form the address indicated by the value of PC. The instruction is also is given to Control Unit via Fetch and Control block. In a 32-bit instruction(RV32I)[1], according to the value of "Opcode" (0 to 6 bits), "func3"(12 to 14 bits) and "Inst" (30th bit), the control unit decides the specific operation to be performed. The Opcode (0 to 6 bits) decides which type of Instruction encoding is used e.g. R, I, B, S, J, U, etc. The Control Unit takes 2 signals as input (instruction, branch) and gives 7 output signals to the Datapath Unit.

The *we* is write enable of the register file. The *memread* and *memwrite* is read and write enable of data memory respectively. "Func3" describes the operation to be performed by ALU on that encoding e.g. add, sub, shift, and, xor, etc. The *pcsel, bsel, slsel* acts as the select lines for the next value of PC, for 2nd operand to ALU (B-select Block), for inputs to Register-file (Store Logic Block), respectively.

The Sign-immediate Extension values for various Instruction-encoding are implemented as *immS*(for Store), *immI* (for I encoding), *immB*(for Branch), *immU* (for U encoding), *immJ* (for J encoding).The value of these extensions is evaluated on every cycle but is used according to immediate instruction and the values of *pcsel*,*bsel*, *slsel* which are the select lines for their blocks. Because of the single-cycle processor, some level of simplicity is needed to

achieve a balance between component utilization in between clock cycles, hence separate adders are used for incrementing PC, branch and jump address calculation, etc. The maximum size of Instruction and Data memory can be up to 4GB because of the 32-bit address line, but for the sake of simulation and complexity, implemented instruction and data memory are of 16KB.An example of workflow goes as follows in Figure 1 For ADD Instruction, According to the value of *pcsel* the PC will fetch the instruction(i.e. ADD), then Control Unit will generate the control signals as *we* = 1'b1, *pcsel* = 2'b00, *bsel* = 3'b000, *slsel* =2'b01, *memread* = 1'bz, *memwrite* = 1'bz. According to the value *bsel* the ALU will get input A, B from the register file via 'RD1' and 'RD2', the ALU evaluates addition and gives the result as C. According to the value of *slsel* the result gets written to Register file at address giving by rd, because write enable for register file is also high.

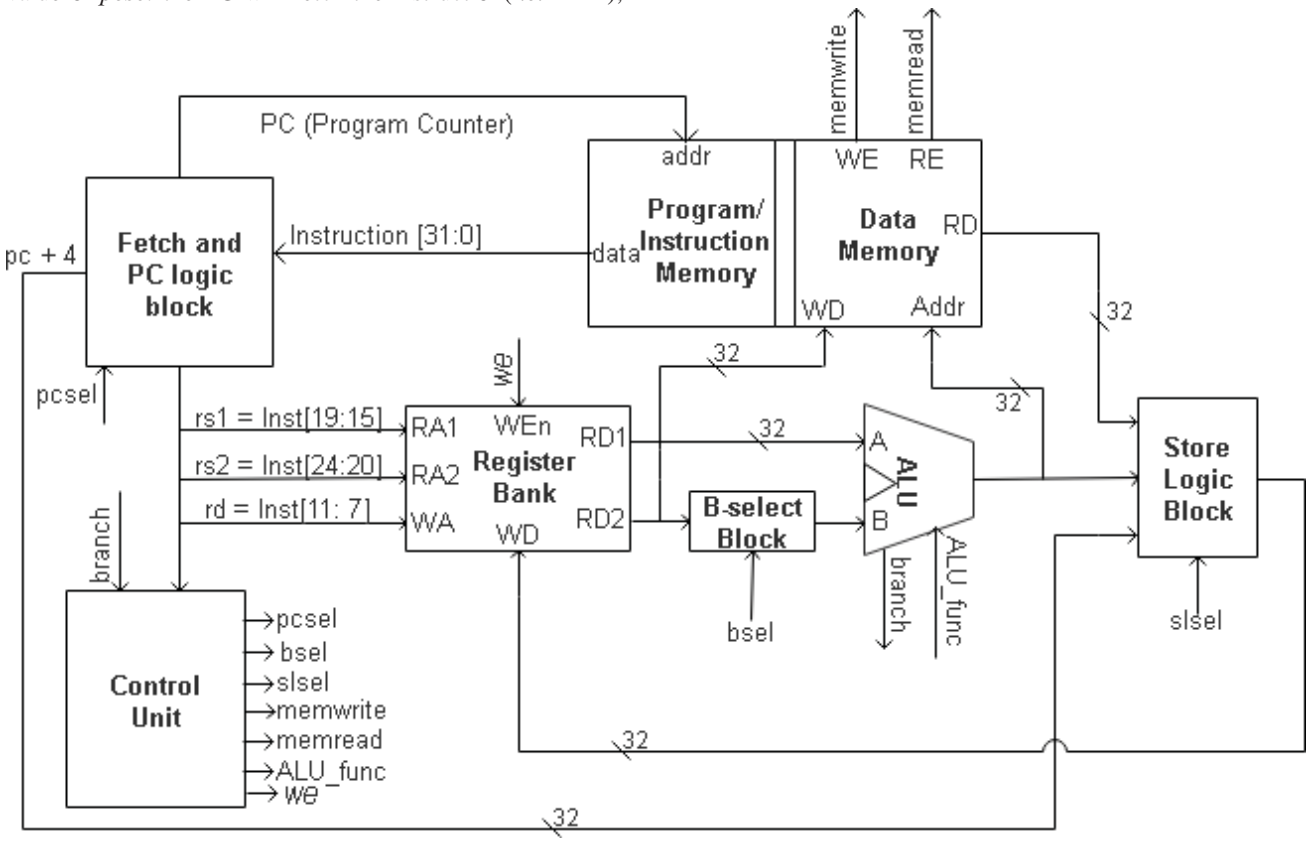


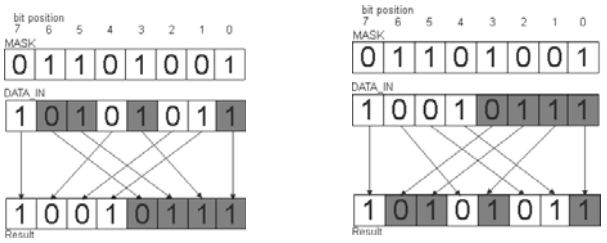
Fig. 1: Designed RISC-V Micro-Architecture.

C. Bit Manipulation Instruction

In this work, two new Instruction "grup" and "degrup" are introduced. We constraint our method and implementation simple to target low-cost devices. Suggested encoding for the instruction is shown in Table 1. R-type encoding is chosen as the format for our instructions is the same as of R-type encoding. Using another opcode would increase machine complexity. For example, corresponding to bit 2 of the mask which is '1' its *data in* value gets selected and inserted to the result as shown. Since 6th and 7th bit of mask is also '1' its corresponding *data in* value gets selected and shifted to the right side. Similarly, the bit 1,3,4,5,8 of the mask which is '0' its corresponding *data in* value gets shifted to the left, hence grouping them in action. Therefore, the process is called grouping, as it groups the bits according to the 1's and 0's in mask and the ordering of bits in the formed groups will also be the same. TABLE I: Suggested Encoding of the grup and degrup instruction *grup* implements the grouping of data bits according to 1's and 0's in mask bits. The instruction permutes its *data input* as in figure 2 by grouping the bits corresponding to 1's on *mask* bit to the right side and with 0's to another side, hence maintaining the order of the bits within each group

TABLE I: R-TYPE ENCODING

funct7	rs2	rs1	Funct3	rd	Opcode	
0011000	rs2	rs1	000	rd	1101011	gp
0010000	rs2	rs1	000	rd	1101011	
31 25	24 20	19 15	14 12	11 7	6 0	degrup



(a) 8-bit grup (b) 8-bit degrup

Fig. 2: Operation of grup and degrup examples with 8-bit input data and mask

The workflow for the grup instruction is implemented as figure 3

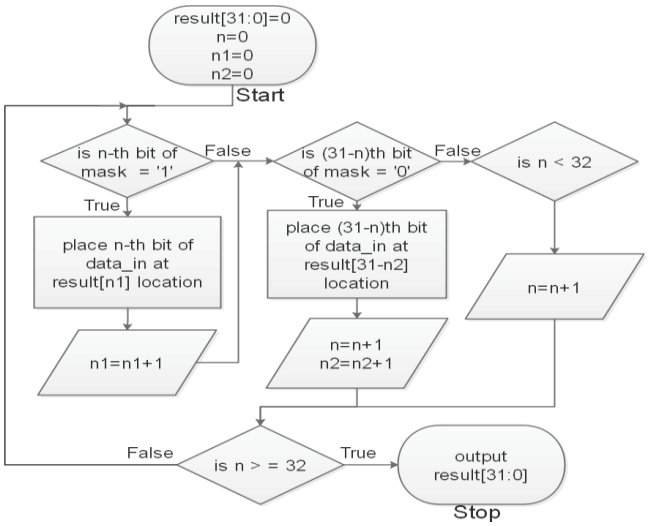


Fig. 3: Flowchart of grup instruction Disgrouping

degrup does exactly the opposite/inverse oper-ation of what *grup* do. In contrast to *grup*, the *data in* and results just get interchanged. As in figure 4b, the places where '1' is occurring in the mask is the position of accumulation of *data in* bits in result. Since bit-2,6,7 of the mask is '1' the *data in* bits from right to left gets placed in results corresponding to '1' in mask and the same goes with the bit '0', hence depositing or disgrouping the bits. There are some advantages of *grup* instruction as it can be used in an algorithm for fast radix-2 sorting and the cryptographic primitive field. It can be used in various applications related to Steganography which deals with the process of hiding a secret message, Compression and Decompression of bits, with further improvement in the instruction and bit arithmetic and representation techniques it can be used in clustering algorithm like K-mean clustering, etc.

IV. SIMULATION AND RESULTS

The "bitRISC" Processor was designed and synthesized us- ing behavioral modeling in Verilog HDL. For both simulation and synthesis, Xilinx Vivado 2018.3.1 and GTKWave tools were used. The hardware implementation for the new RISC-V BMI's was also integrated within our processor. In general, the flow of evaluation is, the instructions to be executed are written in assembly language which is then con- verted to its machine code though some compilers or assembly translators and the resulting machine code is dumped on the instruction Memory. On execution of the machine cycle of the processor, the resulting outputs of each instruction executed are tested and evaluated in the simulation environment through benchmarks. After successful validation the processor was then prototyped as an accelerator and programmed on FPGA, displaying the results on LEDs. The idea was to Write a C-program in Xilinx Vivado Software Development Kit (SDK) to interface the created "bitRISC" processor accelerator with inbuilt GPIO's (switch, buttons, LED) on Programmable Logic (PL) region of the FPGA and with the inbuilt Zynq@7000 processor on (PS) region of ZedBoard. In this validation suit, the assembler or compilers were not used but the Instruction can be given in its machine code format originally either though Dual-port BRAM or directly initializing from memory (to perform an only particular task) or though C-program.

TABLE II: Code snippet for our example

Assembly Code	Operation
# Code groups the bits according to mask bit placed at memory[7] from the product of two numbers placed at memory[1] and memory[2]. Output result at memory[4]	
LW R1, 1(R0)	load Reg[1] <= Mem[Reg[0] + (1)]
LW R2, 2(R0)	load Reg[2] <= Mem[Reg[0] + (2)]
ADDI R5, R0, R2	addition Reg[5] <= Reg[0] + Reg[2] or mov R5 , R2
ADDI R9, R2, -1	addition Reg[9] <= Reg[2] + SXT(-1)
ADDI R2, R0, R9	addition Reg[2] <= Reg[0] + Reg[9] or mov R5 , R9
loop : ADD R6, R5, R1	addition Reg[6] <= Reg[1] + Reg[5]
ADDI R5, R0, R6	addition Reg[5] <= Reg[0] + Reg[6] or mov R5 , R6
ADDI R7, R2, -1	addition Reg[7] <= Reg[2] + SXT(-1)
ADDI R2, R0, R7	addition Reg[2] <= Reg[0] + Reg[7] or mov R2 , R7
BNE R2, R0 , loop	branch if not equal to Zero branch <= (Reg[0] != Reg[2]) ? 1 : 0
SW R5, 3(R0)	store Mem[Reg[R0] + (3)] <= Reg[5]
LW R1, 7(R0)	load Reg[1] <= Mem[Reg[0] + (7)]
grup R8, R5, R1	group R5 according to R1 and store in R8
SW R8, 4(R0)	store Mem[Reg[R0] + (4)] <= Reg[8]

The processor was tested at different modules by having ded- icated Test-Benches for each module to verify the correctness of its architecture and result of the system

thereby increasing the Robustness of the model. For final testing, we implemented examples like multiplication by repetitive addition, the sum of N-natural numbers, square series generation to some number, etc.
The setup for FPGA prototyping is shown in fig. The Simulation for Synthesized was done at the RTL-logic level. The design was implemented on ZEDBoard FPGA. The on-chip BRAM of 16KB for instruction memory and data memory was used. Also, on-chip GPIOS's such as buttons, switches and LED's were used.

However, for the sake of this article, an easy example is used to verify the experiment, the multiplication of two numbers by repeated addition and will use *grup* command to mask out useful bits. The assembly code for the following is given in Table II.

The hardware utilization report is shown in Table III. It is observed that our design can obtain upto 100MHz operating clock frequency. Total consumed power on chip is reported around 1.109W.

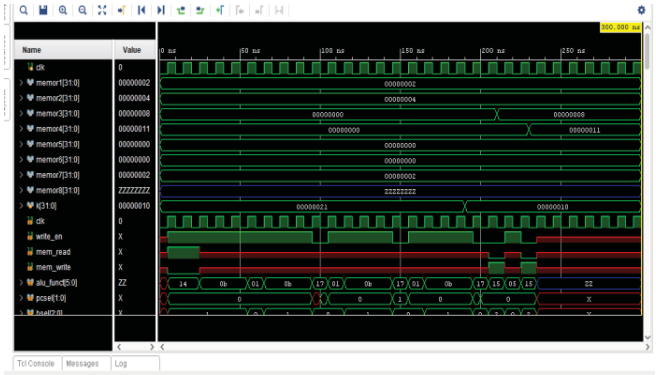


Fig. 4: Simulation output memory

Figure 4 contains final output, which has the description in its contemporary table. The simulation for the code snippet is shown in figure 4. The experiment setup on ZedBoard is shown in figure 5.

TABLE III: Resource Utilization

Resources	Utilization	Available	Utilization %
LUT	2312	53200	4.35
LUTRAM	62	17400	0.36
FF	2035	106400	1.91
IO	21	200	10.50
BUFG	13	32	40.63

Fig 5: Setup for Implementation

V. CONCLUSION

The current paper shows the implementation of a RISC-V processor "bitRISC" and also introduced two new RISC-V BMI's *grup* and *degrup*. The future goal is integrate further RISC-V extensions to the system like, M (Multiplication and Division unit), F(Single precision Floating-Point Unit), etc. to make an accelerator on FPGA for Software Defined Radio(SDR) filtering application using the existing and pre- sented BMI's resulting in significantly reduction in cost and increasing Throughput. These results might be in a future paper to help creating opportunities for simple and faster designs.

REFERENCES

[1] "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Docu- ment Version 20191213", Editors- Andrew Waterman and Krste Asanovi c, RISC-V Foundation, December 2019.

[2] Wikipedia, "Arm architecture — Wikipedia, the free encyclope- dia", [Online; last accessed 31-Janurary-2020]. [Online]. Available: https://en.wikipedia.org/wiki/ARMarchitecture.

[3] Wikipedia, "Bit Manipulation Instruction Sets —Wikipedia, the free encyclopedia", [Online; last accessed 3-Febuary-2020] [Online]. Available: https://en.wikipedia.org/wiki/*Bit$_M$anipulation$_I$nstruction$_S$ets*.

[4] K. Asanovi and D. A. Patterson, "Instruction sets should be free: The case for risc-v," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146, Aug 2014. [Online]. Available:http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-46.html .

[5] Don Kurian Dennis, Ayushi Priyam, Sukhpreet Singh Virk, Sajal Agrawal, Tanuj Sharma, Arijit Mondal, Kailash Chandra Ray. "Single cycle RISC-V micro architecture processor and its FPGA prototype", 2017 7th Interna- tional Symposium on Embedded Computing and System Design (ISED), 2017.

[6] Bastian Koppelmann, Peer Adelt, Wolfgang Mueller, Christoph Scheytt. "RISC-V Extension for Bit Manipulation Instructions", 2019 29th Inter- national Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2019.

[7] A. Traber et al., "Pulpino: A small single-core risc-v soc," in 2016 RISC- V Workshop, 2016.

[8] A. Menon et al., "Shakti-t: A risc-v processor with light weight security extensions," in Proceedings of the Hardware and Architectural Support for Security and Privacy, ser. HASP '17. New York, NY, USA: ACM, 2017.

[9] Sug H. Jeong,Myung H. Sunwoo,Seong K. Oh "Bit Manipulation Accel- erator for Communication Systems Digital Signal Processor" in EURASIP Journal on Applied Signal Processing 2005:16, 2655–2663.

[10] GS Tomar, Marcus George , "Modified Binary Multiplier Architecture to Achieve Reduced Latency and Hardware Utilization", Wireless Personal Communication, 2018, Vol.98, No.4, pp.3554-3561