Chandaka Venkata Lakshmi* et al.
(IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH
Volume No.5, Issue No.3, April – May 2017, 6115-6119.

Implementation of Low Power and Delay Scalable Channel Parallel NAND Flash Memory Controller Architecture Using ALU

CHANDAKA VENKATA LAKSHMI
Avanthi Institute of Engineering and Technology, Bhogapuram Mandal, Tagarapuvalasa, Vizianagaram Dist

Dr.G.V.SRIDHAR, M.E ,PhD.
Professor & HOD, Avanthi Institute of Engineering and Technology, Bhogapuram Mandal, Tagarapuvalasa, Vizianagaram Dist

Abstract: **RISC refers to Reduced Instruction Set Computer. Which means the computer that consists of RISC processor contains reduced (simple) instructions for performing necessary and required operations. Any chip if considered as processor, it should have the capability of performing certain operations like arithmetic, logical, control and data transfer. For performing these operations, a processor should contain some major blocks as Control unit (CU), Flexible computational unit (FCU), Program counter (PC), Accumulator, Instruction register, Memory and additional logic.**

RISC actually enhances the performance of processor by considering the factors like simple architecture construction and instruction set, easy instruction set for decoding and simplified control architecture. This paper proposes a simple 32 bit RISC processor by using Peres reversible logic gates, which is expected to reduce the size then the conventional architecture that is based on carry save logic adder approach. The synthesis and simulation is carried out using XILINX ISE 12.3i and HDL is developed using VERILOG language.

Key words: **RISC; Reversible Logic Gates; Carry Save Logic; XILINX;**

I. INTRODUCTION

The design of 32-bit RISC processor incorporates various design blocks like Flexible computational unit (FCU), Accumulator, Program Counter (PC), Instruction Register (IR), Memory, Control Unit (CU), and additional logic. The design incorporates some the following issues which are based on 32 bit data and 28 bit address. It Uses fixed instruction format of length 32 bit, Size of opcode is of 4 bit, handling 15 instructions, has 256 memory locations, 32-bit registers (IR,ACC), Implements 2-staged pipelining i.e overlaps of fetch and execute cycles, No interrupts and No conditional branches, Data that it handles is unsigned integer type.

The FCU performs both arithmetic and logical operations and as well as control of transfer instructions. It takes data and acc as inputs to generate output according to the opcode. An execlk is given as input for synchronization and the output is available at positive edge of the execlk. It performs arithmetic and logic instructions directly and control of transfer instructions are performed with the help of control and logic decoder.

The flexible computational unit (FCU) performs all arithmetic operations (addition, Subtraction, multiplication, and division) and logic operations. Logic operations test various conditions encountered during processing and allow for different actions to be taken based on the results. The data required to perform the arithmetic and logical functions are inputs from the designated CPU registers and operands. The FCU relies on basic items to perform its operations. These include number systems, data routing circuits (adders/ subtracters), timing, instructions, operands, and registers.

The result of an FCU operation is always stored in accumulator at some specified time based on the control logic instruction and also the execlk. This output is again fed to FCU as input. If Reset =0, the output of accumulator is cleared to zero. When reset is high and load accumulator signal is set high, the output of the FCU is loaded in to the accumulator at the neg edge of the execlock. Used for writing data in to memory. When it is required to write data in to the memory, then necessary control signals are generated at the buffer. Buffer is used for achieving bi-directional operation of the data bus.

Multiplexer provides memory access to either IR (instruction register) or PC (program counter) based on Fetch signal. Memory has both data and instructions .In order to access both data and instructions through a single address port a multiplexer is needed. It selects address according to the fetch signal. If the fetch signal is high pcout is selected and irout is selected when fetch signal is low.

The program counter (PC) contains the address of the next instruction. The CPU fetches an instruction from memory, executes it and increments the content of PC. Thus in the next instruction cycle, it will fetch the next instruction of the program pointed out by the program counter.

The instructions are executed sequentially unless an instruction changes the content of program counter.

Instruction register is a 32-bit register which gets loaded with data from the memory. When LdIr signal is high, the data bus contents get loaded into the Instruction register. The 4-MSB's of this loaded data [31:28] are the OpCode of the Instruction and the remaining bits [27:0] are the address of a memory location tofetch the subsequent data.

On the falling edge of the asynchronous signal Rst the IR gets cleared irrespective of any condition. Outputs of IR module are IrOut (lower 28-bits) and OpCode (upper 4-bits) of the data fetched from the memory.

The control logic generates all the necessary control signals required for satisfactory operation of the CPU. The control signals are load accumulator, load instruction register, increment program counter, load program counter, memory read, memory write. When LdAcc is set high, the output of the FCU is loaded into the accumulator. When Ldpc is set high program counter is loaded with the address from where the next instruction is to be fetched. When Ldir is set high the instruction register is loaded with the instruction that is to be executed. If increment program counter signal is set high, program counter is incremented by one to the previous value .If write signal is set high and read low then the data bus is being written on the address of the memory location indicated by thIf read signal is high and write low this is memory read operation and the data at the memory address indicated is placed on the data bus.

II. FLEXIBLE PROCESSOR ARCHITECTURE

Let us consider an instance when some information is stored in the memory. Now when the system is switched on, CPU is initialized. In order to fetch an instruction, as a result the program goes to the location in the memory that is pointed out by the program counter. After some instance, the instruction from the memory is put on the data bus. This cycle is called the instruction fetch cycle. The instruction is now available at the data bus. at next instance; the instruction is loaded into the instruction register. This is called the instruction load. In this cycle the 4 msb's of the instruction are separated and put in the opcode register and are loaded to control unit as well as FCU. The rest of the bits are sent out as Irout. The outputs of the instruction register and the program counter are connected to a mux. During the negative edge of the fetch signal, the output of the instruction register is selected, while the output from the program counter is selected during the positive edge of fetch cycle.

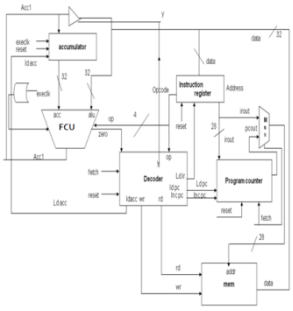


fig.1: Flexible processor architecture.

Now when the fetch signal goes low the mux selects the output from the instruction register and it points to the location of the operand. Now the operand present in the location is placed on the data bus. After an instruction is fetched the program counter is incremented. It points to the next location. Now the operand is available at the FCU. The operand is taken in by the FCU and operates on it.

Now the result is available at acc1 at positive edge of execlk. During the negative edge of execlk, the result at the Acc1 register is placed on the data bus, which is sent and loaded into the accumulator for any further operations. If the data has to be stored into the memory, then during this clock cycle, Rd and Wr has to be 0 & 1 respectively. As a result the accumulator is connected to the memory and the value in the accumulator is sent back to a location in the memory through a module named Buffer. A characteristic of RISC processor is their ability to execute one instruction per clock cycle.

The Flexible computational Unit (FCU) performs arithmetic (addition, subtraction, multiplication) and/or logical operations (and,or nand etc). Two bits selection determines which operation takes place at a particular time. All the modules in the FCU design are realized using VHDL.

Design functionalities are validated through simulation. Besides verifying the outputs, the outputs' timing diagram and interfacing signals are also tracked to ensure the design specifications. Successful implementation of FCU using VHDL fulfills the needs for different high-performance applications.

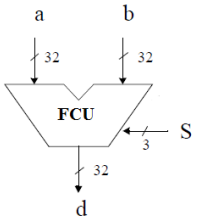


Fig.2: FCU block diagram.

In computing, **flexible computational unit** (**FCU**) is a digital circuit that performs arithmetic and logical operations. The FCU is a fundamental

building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units accommodate very powerful and very complex FCUs; a single component may contain a number of FCUs. Mathematician John von Neumann proposed the FCU concept in 1945, when he wrote a report on the foundations for a new computer. An FCU must process numbers using the same format as the rest of the digital circuit. The format of modern processors is almost always the two's complement binary number representation. Early computers used a wide variety of number systems, including ones' complement, two's complement sign-magnitude format, and even true decimal systems, with ten tubes per digit.

FCUs for each one of these numeric systems had different designs, and that influenced the current preference for two's complement, as this is the representation that makes it easier for the FCUs to calculate additions and subtractions. The ones' complement and two's complement number systems allow for subtraction to be accomplished by adding the negative of a number in a very simple way which negates the need for specialized circuits to do subtraction; however, calculating the negative in two's complement requires adding a one to the low order bit and propagating the carry. An alternative way to do two's complement subtraction of A−B is to present a one to the carry input of the adder and use ¬B rather than B as the second input. Most of a processor's operations are performed by one or more FCUs. An FCU loads data from input registers, an external Control Unit then tells the FCU what operation to perform on that data, and then the FCU stores its result into an output register. The Control Unit is responsible for moving the processed data between these registers, FCU and memory.

Most FCUs can perform operations such as Bitwise logic operations (AND, NOT, OR, XOR), Integer arithmetic operations (addition, subtraction, and sometimes multiplication though this is more expensive), Bit-shifting operations (snhifting or rotating a word by a specified number of bits to the left or right, with or without sign extension). Shifts can be seen as multiplications and divisions by a power of two.

Engineers can design an Flexible computational unit to calculate any operation. The more complex the operation, the more expensive the FCU is, the more space it uses in the processor, the more power it dissipates. Therefore, engineers compromise. They make the FCU powerful enough to make the processor fast, but yet not so complex as to become prohibitive.

III. REVERSIBLE LOGIC GATES

A reversible logic gate is an n-input n-output logic device with one-to-one mapping. This helps to determine the outputs from the inputs and also the inputs can be uniquely recovered from the outputs. Also in the synthesis of reversible circuits direct fan-out is not allowed as one–to-many concept is not reversible. A reversible circuit should be designed using minimum number of reversible logic gates. From the point of view of reversible circuit design, there are many parameters for determining the complexity and performance of circuits.

In this, an improved design of reversible multiplier with respect to its previous counterparts is proposed. Multiplier circuits play an important role in computational operation using computers. There are many arithmetic operations which are performed, on a computer FCU, through the use of multipliers. Design and implementation of digital circuits using reversible logic has attracted popularity to gain entry into the future computing technology.

Feynman Gate

Figure 3 shows a 2*2 Feynman gate . Quantum cost of a Feynman gate is 1.Feynman gate is called as Controlled NOT gate or CNOT gate. It is equivalent to single control input tofili gate.

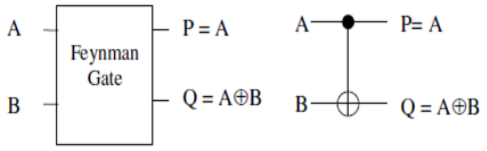


Fig.3: feynman gate and its symbolic representation.

Toffoli Gate

Figure 4 shows a 3*3 Toffoli gate The input vector is I(A, B, C) and the output vector is O(P,Q,R). The outputs are defined by P=A, Q=B, R=A(B xor C). Quantum cost of a Toffoli gate is 5. It has two control inputs.

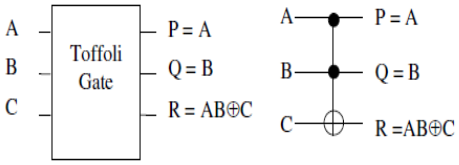


Fig.4: Toffoli gate and its symbolic representation.

Peres Gate:

Figure 5 shows 3*3 Peres gate. The input vector is I (A,B,C) and the output vector is O (P,Q,R). the output is defined by P=A, Q= A^B and R= A&B

^C. quantum cost of a Peres is 4. It needs two Toffoli gates for its construction.

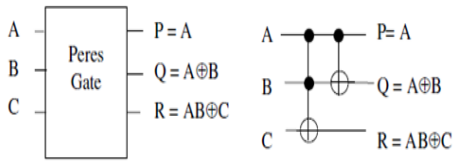


Fig.5: Peres gate and its symbolic representation.

BVPPG gate:

BVPPG gate is a 5 * 5 reversible gate and its logic diagram is as shown in figure 6. Its quantum cost is 10. Ffoli representation of the BVPPG gate is a shown.

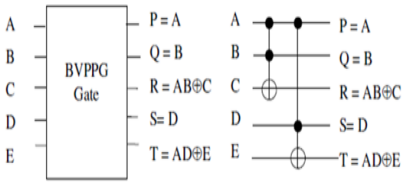


Fig.6: BVPPG gate and its symbolic representation.

The BVPPG gate is used to construct the partial product generator which has resulted in least number of gates, least quantum cost and least number of garbage outputs. The two product terms are available at the outputs R and T of the BVPPG gate with C and E inputs maintained constant at 0. The other outputs namely P, Q and S are used for fan-out of the multiplier operands as shown in figure. This reduces the number of external fan-out gates to zero in our design which is main design feature. The proposed design is compared with the existing designs

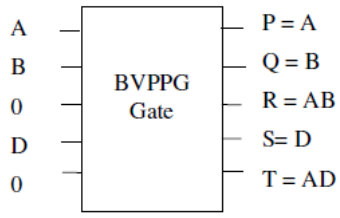


Fig. 7: Producing product terms and duplication of the inputs

CNOT GATE

CNOT gate is also known as controlled-not gate. It is a 2*2 reversible gate. The CNOT gate can be described as:

Iv = (A, B) ; Ov = (P= A, Q= A B)

Iv and Ov are input and output vectors respectively.m Quantum cost of CNOT gate is 1. Figure shows a 2*2 CNOT gate and its symbol.

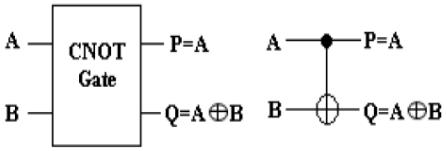


Fig.8: CNOT gate and its logic symbol.

NFT Gate:

It is a 3x3 gate and its logic circuit and its quantum implementation is as shown in the figure. It has quantum cost five

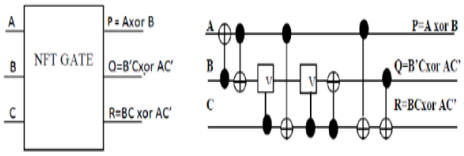
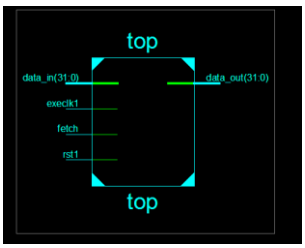


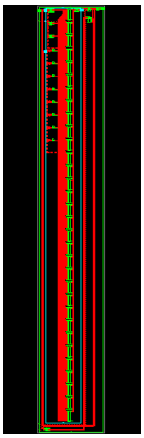
Fig.9: NFT gate and its Logic symbol.

IV. RESULTS

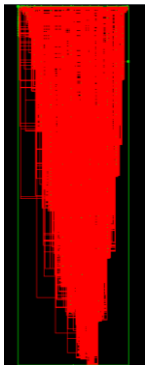
RTL schematic



Internal architecture



Technological schematic



V. CONCLUSION

RISC actually enhances the performance of processor by considering the factors like simple architecture construction and instruction set, easy instruction set for decoding and simplified control architecture. For performing these operations, this processor contain the major blocks as Control unit (CU), Flexible computational unit (FCU), Program counter (PC), Accumulator, Instruction register, Memory and additional logic. In the proposed design, the logic used is RCA with reversible logic gates which consumes less power 20.520 w and occupies less area 270 in terms of LUT's when compared with the existing carry save technique with a power consumption of 25.232 w and with area of 332 in terms of LUT's . The synthesis and simulation is carried out using XILINX ISE 12.3i and HDL is developed using VERILOG language.

VI. REFERENCES

[1] P. Ienne and R. Leupers, Customizable Embedded Processors: Design Technologies and Applications. San Francisco, CA, USA: Morgan Kaufmann, 2007.

[2] P. M. Heysters, G. J. M. Smit, and E. Molenkamp, "A flexible and energy-efficient coarse-grained reconfigurable architecture for mobile systems," J. Supercomput., vol. 26, no. 3, pp. 283–308, 2003.

[3] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in Proc. 13th Int. Conf. Field Program. Logic Appl., vol. 2778. 2003, pp. 61–70.

[4] M. D. Galanis, G. Theodoridis, S. Tragoudas, and C. E. Goutis, "A high-performance data path for synthesizing DSP kernels," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 25, no. 6, pp. 1154–1162, Jun. 2006.

[5] K. Compton and S. Hauck, "Automatic design of reconfigurable domainspecific flexible cores," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 16, no. 5, pp. 493–503, May 2008.

[6] S. Xydis, G. Economakos, and K. Pekmestzi, "Designing coarse-grain reconfigurable architectures by inlining flexibility into custom arithmetic data-paths," Integr., VLSI J., vol. 42, no. 4, pp. 486–503, Sep. 2009.

[7] S. Xydis, G. Economakos, D. Soudris, and K. Pekmestzi, "High performance and area efficient flexible DSP datapath synthesis," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 19, no. 3, pp. 429–442, Mar. 2011.

[8] G. Ansaloni, P. Bonzini, and L. Pozzi, "EGRA: A coarse grained reconfigurable architectural template," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 19, no. 6, pp. 1062–1074, Jun. 2011.

[9] M. Stojilovic, D. Novo, L. Saranovac, P. Brisk, and P. Ienne, "Selective flexibility: Creating domain-specific reconfigurable arrays," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 32, no. 5, pp. 681–694, May 2013.

[10] R. Kastner, A. Kaplan, S. O. Memik, and E. Bozorgzadeh, "Instruction generation for hybrid reconfigurable systems," ACM Trans. Design Autom. Electron. Syst., vol. 7, no. 4, pp. 605–627, Oct. 2002.

[11] [Online]. Available: http://www.synopsys.com, accessed 2013.

[12] T. Kim and J. Um, "A practical approach to the synthesis of arithmetic circuits using carry-save-adders," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 19, no. 5, pp. 615–624, May 2000.

[13] A. Hosangadi, F. Fallah, and R. Kastner, "Optimizing high speed arithmetic circuits using three-term extraction," in Proc. Design, Autom. Test Eur. (DATE), vol. 1. Mar. 2006, pp. 1–6.

[14] A. K. Verma, P. Brisk, and P. Ienne, "Data-flow transformations to maximize the use of carry-save representation in arithmetic circuits," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 27, no. 10, pp. 1761–1774, Oct. 2008.

