Design and Implementation of Pipelined 8-Bit RISC Processor using Verilog HDL on FPGA

Akshatha S Patil, B G Shivaleelavathi

M. Tech, (VLSI Design and Embedded Systems), JSSATE, Bengaluru, Karnataka, India
Professor, Dept. of ECE, JSSATE, Bengaluru, Karnataka, India

---***---

Abstract - *This paper describes an eight-bit RISC processor design, the usage of Verilog hardware Description Language (HDL) on FPGA board. The proposed 8-bit RISC processor may be carried out with the help of separate data and instruction memory i.e. Harvard structure. The critical feature of proposed RISC processor architecture is pipelining, it is used for boosting the general performance, such that on each clock cycle one instruction can be performed. Another feature is that the proposed RISC processor which can be very compact, simple and clean to investigate and contains 24 instructions. The proposed RISC processor architecture two 8-bit I/O ports, arithmetic and logical unit serial-in/serial-out ports, eight 8-bit widespread cause registers, 4-bit flag sign up and priority based totally 3 vectored interrupts. The processor code is synthesized on Spartan 3E Starter Board FPGA.*

Key Words: Harvard structure, Pipelining, Registers, Instruction Set, Interrupts.

1. INTRODUCTION

In an embedded machine, so called the reconfigurable processor has extended significantly in the course of long time. A computer architecture which is called as reconfigurable computing in which it has been associated with the strength of software program with the total hardware performance with the resource of processing which is very useful and versatile for computing programmable arrays.

In this project, we are going to instruct the processor that what all operations have to be performed. In any processor there will be an instruction set architecture according to which instructions are given and specific operations are performed. Like all the processors, this RISC processor has a unique instruction set which is different from all the other processors. The instructions are given in the program and the particular operation will be performed according to the given instruction.

The instructions will be stored in the Instruction Memory and the values will be stored in the registers. The registers used in this processor are 8-bit. The instruction should be written in the program and the program will be executed. The output obtained will be desired output according to input given. For example, if we give 0990, addition operation will be performed according to this unique instruction set architecture, especially for this proposed RISC processor. And in the similar way operations like subtraction, AND, OR, XOR, etc. can be performed.

And even we can perform operations like load, jump and branching instructions. ModelSim is the software which is used to execute the programs. The programs written will be dumped on to an FPGA board and then simulation results will be obtained.

2. PROCESSOR ARCHITECTURE

The processor is designed with the help of having separate data and instruction memory. The processor architecture describes the complete architecture of proposed system. It consists of eight-bit ALU, two eight-bit I/O ports and 8 eight-bit trendy registers, three interrupts, serial-in/serial-out ports and 4-bit flag sign up having zero flag (Z), carry flag (C), borrow flag (B) and parity flag (P).

The processor works at 2.5 voltages deliver 25MHz clock frequency. For transferring statistics among extraordinary modules 8-bit system bus is used. To maintain the mathematics and logical operations an eight bit accumulator is used. For speaking among exclusive modules the instruction and facts reminiscence have exquisite buses.

The interrupt module consists of 3 interrupts, which might be precedence based totally and one of the interrupt is masks in a position. The architecture having 34 instructions in the instruction set which are probably clean and clean to research. Serial module enables entire duplex serial conversation with the assist of UART protocol.

Unique modules can be clocked, which uses clock gating, awesome whilst required for decrease of the energy. Loading to the registers takes place in unspecified amount of time. The modules which use clock gating are records memory and preferred purpose registers.

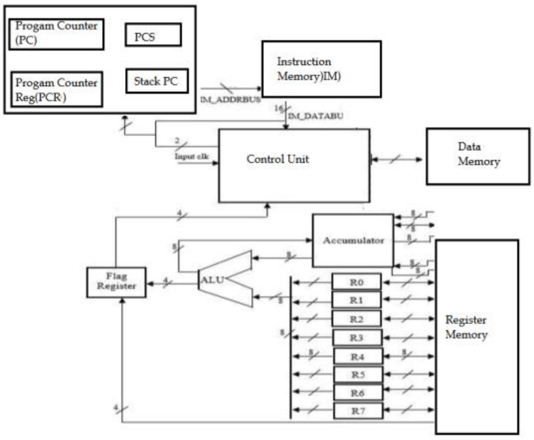


Fig: Processor Architecture

3. PIPELINE ARCHITECTURE

With the help of pipelining computer architecture allows the next instruction to be fetched while the processor is performing on another instruction. Pipelining is the same old trait used in RISC processor for enhancing the performance and to reduce execution time in keeping with instruction.

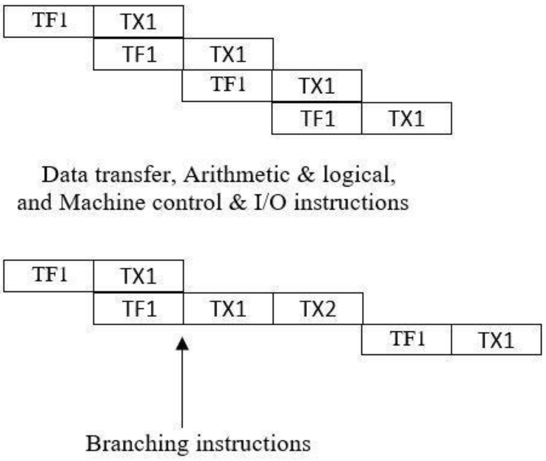

TF1
TX1
TF1
TX1
TF1
TX1
TF1
TX1
Data transfer, Arithmetic & logical, and Machine control & I/O instructions
TF1
TX1
TF1
TX1
TX2
TF1
TX1
Branching instructions

Fig: Pipelining

The processor requires clock cycles for the execution of the fetch execution cycle such that they're collectively special. While executing one instruction next instruction is being fetched, such that on each and every clock cycle one of the instructions will be carried out.

4. FUNCTIONAL MODULES

The important blocks of the proposed processor are Instruction and Data Memory, Register set, I/O Module, Program Counter Unit, Control Unit, Arithmetic and Logical Unit, Interrupt Module and Serial Module.

4.1 Program Counter Unit

Software counter unit consists of Program Counter Register, Program Counter, Program Counter Save and STACK PC. In this proposed processor PCR is used to keep the deal with of the preparation, while leap is carried out. Program Counter that is a sixteen-bit tremendous registers which incorporates the cope with being executed on the modern-day-day time.

As one of the instructions is being fetched, the counter will growth the stored price with the aid of manner of element one. Throughout the fetch cycle, IM ADDRBUS, an 18-bit deal with bus transfers the program counter content material to guidance reminiscence (IM) while the corresponding sign is enabled. PCS shops 18-bit. Whilst SAV practice is completed application counter (PC) is incremented with the aid of using values. All through the execution of the interrupt provider routine, STACKPC is used to store the current value of PC.

4.2 Instruction memory

Instruction Memory having 262,144 address locations and is 16-bit wide.

4.3 Control Unit

The control unit consists of IR, IRX, tstate counter, Low energy Unit (LPU) and decoder. All through fetch cycle, IR gets the acknowledgement for decoding. IR content must be stored in another register for the execution because during execution of one instruction subsequent instruction is being fetched. All through each rising fringe of the clock of execution cycle, IR content material is moved to IRX. A fetch and execution cycle is probably generated for the proper operating of the processor. At the growing fringe of the clock, TF1 (fetch cycle), TX1 (execution cycle), TX2 (execution cycle 2 for branching education) are generated. Inside the tstate counter module pipelining function is implemented. Interrupt precedence and exceptions in instructions, like soar, also are considered in tstate counter module. Each time IRX is loaded with a valid instruction, decoder will generate manage signals required for the modules. In LPU of manage Unit, clock gating for facts reminiscence and extensive purpose take a look at in set are carried out. Manage Unit (Control Unit) will obtain inputs from Serial Module, Interrupt Module and flag sign up. For the proper working of the modules, manage unit will take input clock from supply clock of FPGA and generate manipulate indicators and clock signals. The clock alerts embody gated clock signs for sign in set and statistics reminiscence modules, baud price clock for serial module and 25MMHz output clock sign for all the different modules. Clock gating is specially used to lessen the energy dissipation. The number one memory elements, records memory and join up set, so simplest gated clock are supplied to these modules. If there can be a loading to the memory/stylish –purpose sign within the corresponding module may be activated.

4.4 Data Memory (DM)

Memory having 4096 cope (instruction) with locations and is 8-bit extensive. And it receives the specified location with the help of bus referred to as DM ADDRBUS that's 12-bit wide from the manipulate unit. Examine and write can be accessed by means of information bus called machine _DATABUS (SYSTEM) which is 8-bit extensive.

4.5 Arithmetic and Logical Unit (ALU)

ALU includes AND, OR, XOR, ADD, SUB operations. ALU will be connected to the Accumulator and trendy registers through its 8-bit buses ALU DATABUS A and ALU DATABUS B. Accumulator stores the result of the operation. The four-bit flag sign in shops the zero flag (Z), carry flag (C), and borrow flag (B) and parity flag. Parity flag can be set while resultant of ALU operation consists of peculiar wide variety of ones.

4.6 Accumulator

Accumulator is a vital part of the proposed processor, as information (data) transfer that's 8-bit considerable, ALU operations and that I/O operation takes via it. For sending the facts required for transmission through serial-out port and storing the information obtained via serial-in-port accumulator is installed to serial module. Accumulator additionally contains increment, decrement, and compliment, rotate right and rotate left operations.

4.7 Register Set

Registers are used for storing the facts that are regularly used. The sign up set contains eight eight-bit registers R0, R1, R2, R3, R4, R5, R6, and R7. ALU is attached to Register Set for arithmetic and logical operations. For loading and storing the records its miles connected to SYSTEM_DATABUS.

4.8 Interrupt Module

The interrupt module consists of one timer interrupt and two outside (external) interrupts I0 and I1. Interrupt module as proposed on this RISC structure is based totally on priority, outdoor interrupt I0 is having maximum precedence and I1 is having least priority. I0 interrupt isn't mask successful and I1 interrupt is mask in a position. To govern the competencies, interrupt module is having a 3-bit check in INTCON. For permitting and disabling timer interrupt bit 2 is used, bit 1 is used for allowing and disabling the outdoor interrupts and bit 0 is used for shielding I1. Timer module has 10-bit TIMER sign in to depend the predefined c program language period. TMF0 flag receives set while the register reaches the maximum charges. To clean the TMF0 flag CLRTMRF is used.

4.9 I/O Modules

The module has one 8-bit input port and one eight-bit output port for speaking with external environment. The input and output port are externally connected to accumulator. Whilst manage (control) signals are enabled records will transfer to and from the accumulator.

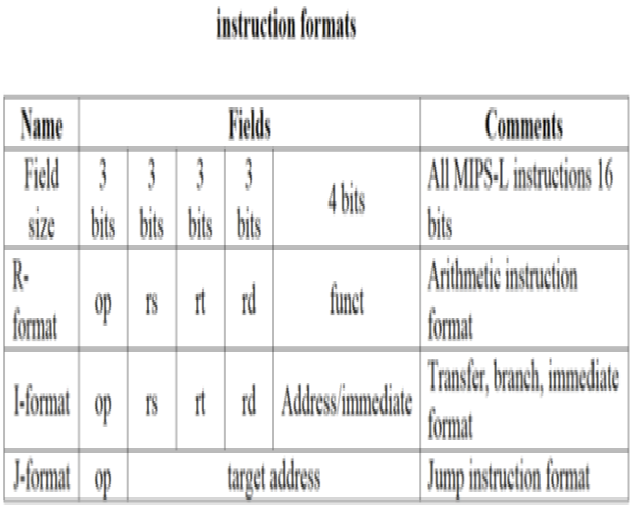
4.10 Serial Module

The serial conversation is full duplex that it can transmit and obtain simultaneously, primarily based on UART protocol. The serial module carries rxin as serial-in-port and txout as serial-out port. The baud charge used is 115200. The statistics transmission starts with a start bit of 0, followed by way of the information bits of the word with the Least full-size Bit (LSB) being dispatched first after which is being dispatched.

TBUFF and RBUFF are the 8-bit registers used for storing the information both during transmission and reception during serial communication. The facts stored in TBUFF register is shifted out throughout serial facts transmission and manner is vice versa for RBUFF register. The baud price is supplied by means of the manipulate unit required for the serial verbal exchange.

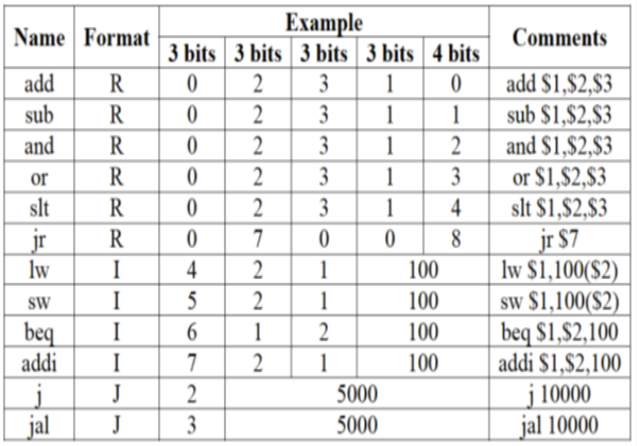
5. INSTRUCTION SET ARCHITECTURE

In this proposed RISC architecture four different types of instructions are present, arithmetic and logical instructions, branching instructions, data transfer instructions, machine control and I/O instructions.
8 registers in the register file for the processor, numbered 0 to 7. Table 1 summarizes the 8 register conventions processor.



instruction formats

Name	Fields					Comments
Field size	3 bits	3 bits	3 bits	3 bits	4 bits	All MIPS-L instructions 16 bits
R-format	op	rs	rt	rd	funct	Arithmetic instruction format
I-format	op	rs	rt	rd	Address/immediate	Transfer, branch, immediate format
J-format	op	target address				Jump instruction format

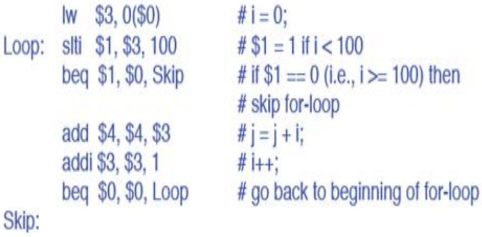


Name	Format	Example					Comments
		3 bits	3 bits	3 bits	3 bits	4 bits	
add	R	0	2	3	1	0	add $1,$2,$3
sub	R	0	2	3	1	1	sub $1,$2,$3
and	R	0	2	3	1	2	and $1,$2,$3
or	R	0	2	3	1	3	or $1,$2,$3
slt	R	0	2	3	1	4	slt $1,$2,$3
jr	R	0	7	0	0	8	jr $7
lw	I	4	2	1	100		lw $1,100($2)
sw	I	5	2	1	100		sw $1,100($2)
beq	I	6	1	2	100		beq $1,$2,100
addi	I	7	2	1	100		addi $1,$2,100
j	J	2	5000				j 10000
jal	J	3	5000				jal 10000

machine language

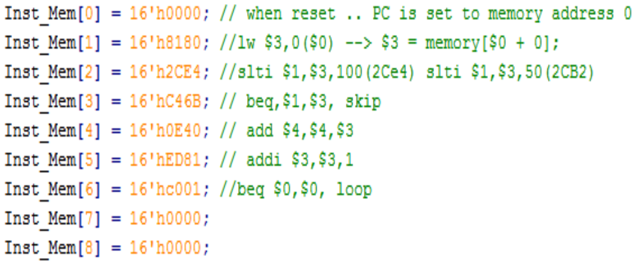
6. EXPECTED RESULTS AND OUTCOMES

The simulation results have been performed by Modelsim and synthesis using Xilinx Spartan 3E Starter Board FPGA. Instructions are derived from the Op code table, for a code snippet, the code snippet which we have considered is a loop as show below



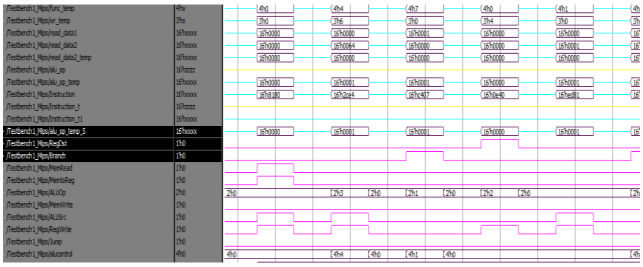
```
        lw   $3, 0($0)        # i = 0;
Loop:   slti $1, $3, 100      # $1 = 1 if i < 100
        beq  $1, $0, Skip     # if $1 == 0 (i.e., i >= 100) then
                              # skip for-loop
        add  $4, $4, $3       # j = j + i;
        addi $3, $3, 1        # i++;
        beq  $0, $0, Loop     # go back to beginning of for-loop
Skip:
```

Instruction derived based on the instructions

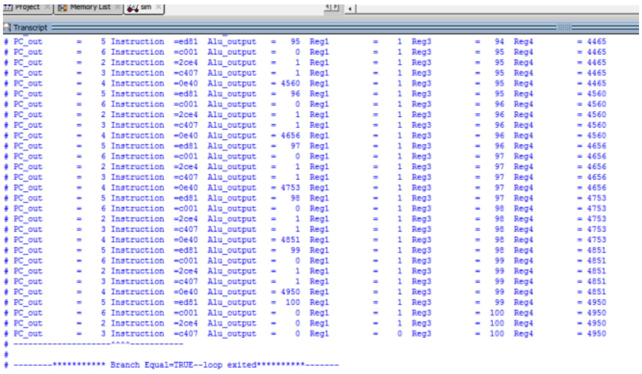


```
Inst_Mem[0] = 16'h0000; // when reset .. PC is set to memory address 0
Inst_Mem[1] = 16'h8180; //lw $3,0($0) --> $3 = memory[$0 + 0];
Inst_Mem[2] = 16'h2CE4; //slti $1,$3,100(2Ce4) slti $1,$3,50(2CB2)
Inst_Mem[3] = 16'hC46B; // beq,$1,$3, skip
Inst_Mem[4] = 16'h0E40; // add $4,$4,$3
Inst_Mem[5] = 16'hED81; // addi $3,$3,1
Inst_Mem[6] = 16'hc001; //beq $0,$0, loop
Inst_Mem[7] = 16'h0000;
Inst_Mem[8] = 16'h0000;
```

6.1 Simulation Results


/Testbench1_Mips/func_temp 4'hx
/Testbench1_Mips/wr_temp 3'hx
/Testbench1_Mips/read_data1 16'hxxxx
/Testbench1_Mips/read_data2 16'hxxxx
/Testbench1_Mips/read_data2_temp 16'hxxxx
/Testbench1_Mips/alu_op 16'hzzzz
/Testbench1_Mips/alu_op_temp 16'hxxxx
/Testbench1_Mips/Instruction 16'hxxxx
/Testbench1_Mips/Instruction_t 16'hzzzz
/Testbench1_Mips/Instruction_t1 16'hxxxx
/Testbench1_Mips/alu_op_temp_S 16'hxxxx
/Testbench1_Mips/RegDst 1'h0
/Testbench1_Mips/Branch 1'h0
/Testbench1_Mips/MemRead 1'h0
/Testbench1_Mips/MemtoReg 1'h0
/Testbench1_Mips/ALUOp 2'h0
/Testbench1_Mips/MemWrite 1'h0
/Testbench1_Mips/ALUSrc 1'h0
/Testbench1_Mips/RegWrite 1'h0
/Testbench1_Mips/Jump 1'h0
/Testbench1_Mips/alucontrol 4'h0
4'h0
4'h4
4'h7
4'h0
4'h1
3'h0
3'h6
3'h0
3'h4
3'h0
16'h0000
16'h0064
16'h0001
16'h8180
16'h2ce4
16'hc407
16'h0e40
16'hed81
2'h0
2'h3
2'h1
2'h2
4'h4
4'h1

6.2 Simulation Log for the code Code:



```
# PC_out       =    5 Instruction  =ed81  Alu_output  =   95  Reg1        =   1  Reg3       =   94  Reg4       = 4465
# PC_out       =    6 Instruction  =c001  Alu_output  =    0  Reg1        =   1  Reg3       =   95  Reg4       = 4465
# PC_out       =    2 Instruction  =2ce4  Alu_output  =    1  Reg1        =   1  Reg3       =   95  Reg4       = 4465
# PC_out       =    3 Instruction  =c407  Alu_output  =    1  Reg1        =   1  Reg3       =   95  Reg4       = 4465
# PC_out       =    4 Instruction  =0e40  Alu_output  = 4560  Reg1        =   1  Reg3       =   95  Reg4       = 4465
# PC_out       =    5 Instruction  =ed81  Alu_output  =   96  Reg1        =   1  Reg3       =   95  Reg4       = 4560
# PC_out       =    6 Instruction  =c001  Alu_output  =    0  Reg1        =   1  Reg3       =   96  Reg4       = 4560
# PC_out       =    2 Instruction  =2ce4  Alu_output  =    1  Reg1        =   1  Reg3       =   96  Reg4       = 4560
# PC_out       =    3 Instruction  =c407  Alu_output  =    1  Reg1        =   1  Reg3       =   96  Reg4       = 4560
# PC_out       =    4 Instruction  =0e40  Alu_output  = 4656  Reg1        =   1  Reg3       =   96  Reg4       = 4560
# PC_out       =    5 Instruction  =ed81  Alu_output  =   97  Reg1        =   1  Reg3       =   96  Reg4       = 4656
# PC_out       =    6 Instruction  =c001  Alu_output  =    0  Reg1        =   1  Reg3       =   97  Reg4       = 4656
# PC_out       =    2 Instruction  =2ce4  Alu_output  =    1  Reg1        =   1  Reg3       =   97  Reg4       = 4656
# PC_out       =    3 Instruction  =c407  Alu_output  =    1  Reg1        =   1  Reg3       =   97  Reg4       = 4656
# PC_out       =    4 Instruction  =0e40  Alu_output  = 4753  Reg1        =   1  Reg3       =   97  Reg4       = 4656
# PC_out       =    5 Instruction  =ed81  Alu_output  =   98  Reg1        =   1  Reg3       =   97  Reg4       = 4753
# PC_out       =    6 Instruction  =c001  Alu_output  =    0  Reg1        =   1  Reg3       =   98  Reg4       = 4753
# PC_out       =    2 Instruction  =2ce4  Alu_output  =    1  Reg1        =   1  Reg3       =   98  Reg4       = 4753
# PC_out       =    3 Instruction  =c407  Alu_output  =    1  Reg1        =   1  Reg3       =   98  Reg4       = 4753
# PC_out       =    4 Instruction  =0e40  Alu_output  = 4851  Reg1        =   1  Reg3       =   98  Reg4       = 4753
# PC_out       =    5 Instruction  =ed81  Alu_output  =   99  Reg1        =   1  Reg3       =   98  Reg4       = 4851
# PC_out       =    6 Instruction  =c001  Alu_output  =    0  Reg1        =   1  Reg3       =   99  Reg4       = 4851
# PC_out       =    2 Instruction  =2ce4  Alu_output  =    1  Reg1        =   1  Reg3       =   99  Reg4       = 4851
# PC_out       =    3 Instruction  =c407  Alu_output  =    1  Reg1        =   1  Reg3       =   99  Reg4       = 4851
# PC_out       =    4 Instruction  =0e40  Alu_output  = 4950  Reg1        =   1  Reg3       =   99  Reg4       = 4851
# PC_out       =    5 Instruction  =ed81  Alu_output  =  100  Reg1        =   1  Reg3       =   99  Reg4       = 4950
# PC_out       =    6 Instruction  =c001  Alu_output  =    0  Reg1        =   1  Reg3       =  100  Reg4       = 4950
# PC_out       =    2 Instruction  =2ce4  Alu_output  =    0  Reg1        =   1  Reg3       =  100  Reg4       = 4950
# PC_out       =    3 Instruction  =c407  Alu_output  =    0  Reg1        =   0  Reg3       =  100  Reg4       = 4950
# ------------------^^^^-----------
#
# --------*********** Branch Equal=TRUE--loop exited**********-------
```

6.3 ADVANTAGES

- The processor achieves higher performance.
- Lower area.
- Compact in size.
- Low power dissipation.
- Achieve highest system speed.
- Reduces power, cost and complexity.
- Fast concurrent programming.
- Reconfigurable of logic is possible.

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my teacher Dr. B. G. Shivaleelavathi, Professor and Guide, as well as our Principal, Dr. Mrityunjaya V. Latte who gave me an opportunity to do this project on the topic "Design and Implementation of 8-bit RISC Processor using Verilog HDL on FPGA ", which helped me to know so many things. I am really thankful to them.

REFERENCES

[1] H.Dao and V.G.Oklobdzija. "Application of logical effort techniques for speed optimization and analysis of representative adders". In 35th annual Asilomar Conference on Signal, Systems and Computers.

[2] A.Farooqui, V.G.Oklobdzija and F.Chehrazi. Multiplexer based adder for media signal processing. In International Symposium on VLSI Technology, Systems and Applications,1999.

[3] T.Han, D.A. Carlson, and S.P.Levitan. "VLSI design of high-speed low-area addition circuitry". In Proceedings of the IEEE International Conference on Computer Design: VLSI in computers processors.

[4].S. Knowles. A family of adders. In IEEE symposium on Computer Arithmetic.

[5] P.M.Kogge and H.S.Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. In IEEE Transactions on computers.