International Journal of
Scientific Engineering and Technology Research

ISSN 2319-8885
Vol.03,Issue.05,
April & May-2014,
Pages:0735-0738

www.semargroup.org,
www.ijsetr.com

Design of Arithmetic Logic Unit using Finite State Machine in Verilog

SHASHANK KAITHWAS[1], P.K.JAIN[2], D.S.AJNAR[3]

[1]PG Scholar, Dept of Electronics & Instrumentation, S.G.S.I.T.S, Indore, India,
E-mail: shashankkaithwas09@gmail.com.
[2]Assoc Prof, Dept of Electronics & Instrumentation, S.G.S.I.T.S, Indore, India.
[3]Assoc Prof, Dept of Electronics & Instrumentation, S.G.S.I.T.S, Indore, India.

Abstract: This paper present design concept of Arithmetic Logic Unit (ALU). Design methodology has been changing from schematic design to HDL based design. We proposed Arithmetic Logic using State Machine in Verilog HDL based design. The State Machine can be started from any State and can jump on any state in between. Functionalities are validated through synthesis and simulation process. Besides verifying outputs, the timing diagram and interfacing signals are also tracked to ensure that they adhere to the design specification. ALU using State Diagram in Verilog language fulfils the needs for different high performance applications.

Keywords: Arithmetic Logic Unit (ALU), Hardware Description Language (HDL).

I. INTRODUCTION

The ALU, or the arithmetic and logic unit is the section of the processor that is involved with executing operations of an arithmetic or logical nature. In ECL, TTL and CMOS, there are available integrated packages which are referred to as arithmetic and logic units (ALU). The logic circuitry in this units is entirely combinational (i.e. consists of gates with no feedback and no flip-flops).The ALU is an extremely versatile and useful device since, it makes available, in single package, facility for performing many different logical and arithmetic operations. Arithmetic Logic Unit (ALU) is a critical component of a microprocessor and is the core component of central processing unit. ALU can perform various logic operations or different Arithmetic instructions include addition, subtraction, While logic instructions include Boolean comparisons, such as AND, OR, NAND, NOR, XOR and NOT operation.

II. STATE DIAGRAM DESCRIPTION

In the State Diagram let us assume that we are in the State0 then if the RESET is set i.e. logic 1 then the Next State will be State0 and if the RESET is set to logic 0 then depending on the value of START the Next State will be change. Also we can jump from a particular State to any State and if the value of START does not change then the Next State will not change and the Machine will remain in same State as shown in figure 1. Each State has a particular operation as mention in the Verilog code. Arithmetic and logic unit consists of two blocks for different operations-

- Arithmetic operations.
- Logical operations.

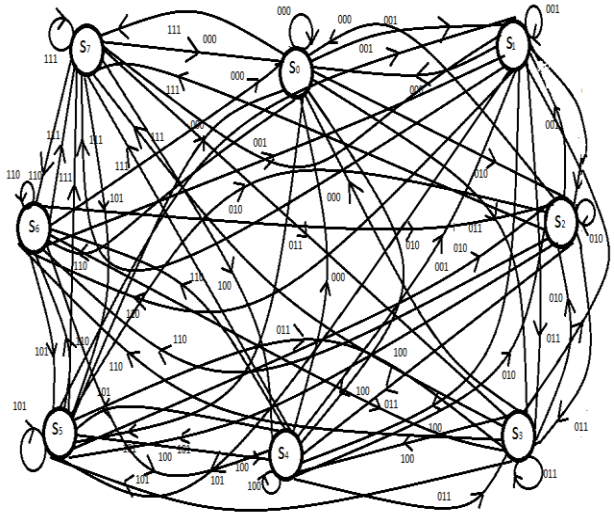

Figure1: State Diagram of ALU.

Addition and subtraction: These two tasks are performed by constructs of logic gates, such as half adders and full adders. While they may be termed 'adders', with the aid of they can also perform subtraction via use of inverters and 'two's complement' arithmetic. A binary adder-subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers. Connecting n full adders in cascade produces a binary adder for two n-bit numbers.

Logical operations: Further logic gate s are used within the ALU to perform a number of different logical tests,

including seeing if an operation produces a result of zero. Most of these logical tests are used to then change the values stored in the flag register, so that they may be checked later by separate operations or instructions.

III. VERILOG CODE OF ALU

The Verilog Code of ALU for the various state is given below:

```
module alu (clk, rst, start, a, b, zout);
input clk, rst;
input [2:0] start;
wire [2:0] start;
input [3:0] a;
wire [3:0] a;
input [3:0] b;
wire [3:0] b;
output [3:0] zout;
reg [3:0] zout;
parameter
state0=0,state1=1,state2=2,state3=3,state4=4,state5=5,state6
=6,state7=7;
reg [2:0] state, nxt_st;
always @ (state or start)
begin: next_state_logic
case (state)
state0: begin
if (start==1) nxt_st=state1;
else if (start==2) nxt_st=state2;
else if (start==3) nxt_st=state3;
else if (start==4) nxt_st=state4;
else if (start==5) nxt_st=state5;
else if (start==6) nxt_st=state6;
else if (start==7) nxt_st=state7;
else nxt_st=state0;
end
state1: begin
if (start==2) nxt_st=state2;
else if (start==3) nxt_st=state3;
else if (start==4) nxt_st=state4;
else if (start==5) nxt_st=state5;
else if (start==6) nxt_st=state6;
else if (start==7) nxt_st=state7;
else if (start==0) nxt_st=state0;
else nxt_st=state1;
end
state2: begin
if (start==3) nxt_st=state3;
else if (start==4) nxt_st=state4;
else if (start==5) nxt_st=state5;
else if (start==6) nxt_st=state6;
else if (start==7) nxt_st=state7;
else if (start==0) nxt_st=state0;
else if (start==1) nxt_st=state1;
else nxt_st=state2;
end
state3: begin
if (start==4) nxt_st=state4;
else if (start==5) nxt_st=state5;
else if (start==6) nxt_st=state6;
else if (start==7) nxt_st=state7;
else if (start==0) nxt_st=state0;
else if (start==1) nxt_st=state1;
else if (start==2) nxt_st=state2;
else nxt_st=state3;
end
state4: begin
if (start==5) nxt_st=state5;
else if (start==6) nxt_st=state6;
else if (start==7) nxt_st=state7;
else if (start==0) nxt_st=state0;
else if (start==1) nxt_st=state1;
else if (start==2) nxt_st=state2;
else if (start==3) nxt_st=state3;
else nxt_st=state4;
end
state5: begin
if (start==6) nxt_st=state6;
else if (start==7) nxt_st=state7;
else if (start==0) nxt_st=state0;
else if (start==1) nxt_st=state1;
else if (start==2) nxt_st=state2;
else if (start==3) nxt_st=state3;
else if (start==4) nxt_st=state4;
else nxt_st=state5;
end
state6: begin
if (start==7) nxt_st=state7;
else if (start==0) nxt_st=state0;
else if (start==1) nxt_st=state1;
else if (start==2) nxt_st=state2;
else if (start==3) nxt_st=state3;
else if (start==4) nxt_st=state4;
else if (start==5) nxt_st=state5;
else nxt_st=state6;
end
state7: begin
if (start==0) nxt_st=state0;
else if (start==1) nxt_st=state1;
else if (start==2) nxt_st=state2;
else if (start==3) nxt_st=state3;
else if (start==4) nxt_st=state4;
else if (start==5) nxt_st=state5;
else if (start==6) nxt_st=state6;
else nxt_st=state7;
end
endcase
end
always @ (posedge clk or posedge rst)
begin: register_generation
if(rst)state=state0;
else state=nxt_st;
end
```

```
always @ (state)
begin: output_logic
case(state)
state0:zout=a + b;
state1:zout=a - b;
state2:zout=a | b;
state3:zout=a & b;
state4:zout=~(a & b);
state5:zout=~(a | b);
state6:zout=a ^ b;
state7:zout=a ^ ~b;
default:zout = 4'b0000;
endcase
end
endmodule
```

IV. ALU SPECIFICATIONS

TABLE 1

OPCODE	OPERATION	SPECIFICATION
000	a+b	zout is assigned the value of a+b
001	a-b	zout is assigned the value of a-b
010	a OR b	zout is assigned the value of a OR b
011	a AND b	zout is assigned the value of a AND b
100	a NAND b	zout is assigned the value of a NAND b
101	a NOR b	zout is assigned the value of a NOR b
110	a XOR b	zout is assigned the value of a XOR b
111	a XNOR b	zout is assigned the value of a XNOR b

V. SIMULATION RESULTS

Design is verified through simulation, which is done in a bottom-up fashion. Small modules are simulated in separate test benches before they are integrated and tested as a whole. All four arithmetic operations available in the design are tested with the same inputs. The sequence of operations done in the simulation is addition. The results of operation on the test vectors are manually computed and are referred to as expected result. By simulation for a and b where a = 4 & b = 2, zout gives following results:

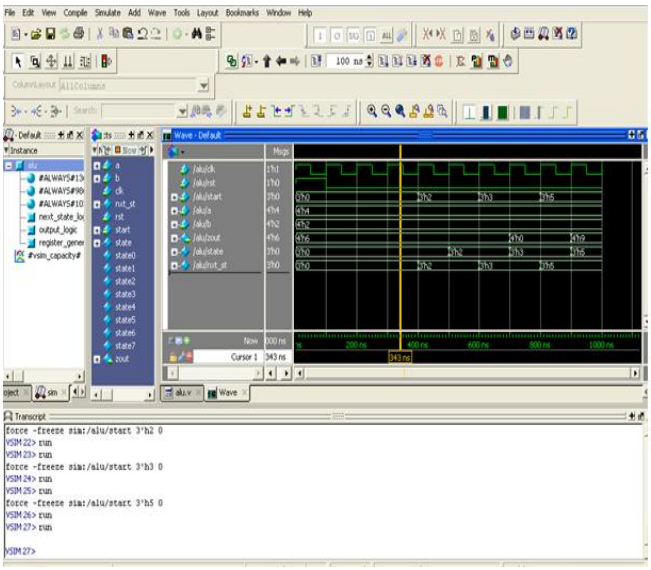

Figure2:

VI. CONCLUSION

In this paper, we have proposed efficient Verilog coding verification method. We have also proposed several algorithms using different design levels. Our proposals have been implemented in Verilog and verified using Mentor Graphics using Questa Sim Simulator 10.2a. We can increase the number of operation by simply adding the number of States in the design. This ALU design using Verilog is successfully designed, implemented, and tested. Currently, we are conducting further research that considers the further reductions in the hardware complexity in terms of synthesis and finally download the code into Altera SPARTEN-3E: FPGA chip on LC84 package for hardware realization.

VII. REFERENCES

[1] J.Bhaskar, Verilog HDL Synthesis, A Practical primer.

[2] Douglas j.Smith, HDL Chip Design: A Practical guide for Desigining, Synthesizing and Simulation ASICs and FPGAs using VHDL or Verilog. JUNE 1996.

[3] James M. Lee, Verilog Quickstart. Hardcover Published by Kluwer Academic Pub. MAY 1997.

[4] Palnitkar, Samir. Verilog HDL - A Guide to Digital Design and Synthesis.

[5] Fraunhofer IIS, "From VHDL and Verilog to System".www.iis.fraunhofer.de/bf/ic/icdds/arb_sp/vhdl.

[6] Shikha khurana, Kanika kaur. "IMPLEMENTATION OF ALU USING FPGA" International Journal of Emerging Trends & Technology in Computer Science (IJETTCS). Volume 1, Issue 2, July – August 2012.

[7] Jiang Hao, Li Zheying, "FPGA design flow based on a variety of EDA tools" in Micro-computer information, 2007(23)11-2:201-203.
[8] Bob Zeidman, Verilog Designer's Library. Prentice hall.

VII. ACKNOWLEDGMENT

We gratefully acknowledge the Almighty GOD who gave us strength and health to successfully complete this venture. We wish to thank lecturers of our college for their helpful discussions. We also thank the other members of the Verilog synthesis group for their support.

Author's Profile:

Shashank kaithwas is currently persuing M.Tech with specialization in Microelectronics and VLSI Design at S.G.S.I.T.S, Indore, India. He received his Bachelor degree in Electronics and Instrumentation Engineering from Acropolis Institute of Technology and Research, Indore. His field of interest includes Digital VLSI Design, EDA, RTL simulation and synthesis, Verilog HDL.

P.K. Jain received the B.E. degree in Electronics and communication Engineering from D.A.V.V. University, India in 1987 and M.E. Degree in Digital Techniques & Instrumentation Engineering from Rajiv Gandhi Technical University Bhopal, India in 1993. He has been teaching and in research profession since 1988. He is now working as Associate Professor in Department of Electronics & Instrumentation Engineering, S.G.S.I.T.S.

D.S.Ajnar received the B.E. degree in Electronics and Communication Engineering from D.A.V.V University, India in 1993 and M.E. Degree in Digital Techniques & Instrumentation Engineering from Rajiv Gandhi Technical University Bhopal, India in 2000. He has been teaching and in research profession since 1995. He is now working as Associate Professor in Department of, Electronics & Instrumentation Engineering S.G.S.I.T.S, Indore, India. His interest of research is in Designing of analog filter and Current-Conveyor.