A Comparison of VHDL and Verilog HDL Through Design of a General-Purpose Microprocessor

[1]M. Chatrapathi SaiTeja, [2] O. Nageswaramma, [3] Vishal Kumar
[1] PG Scholar (DSCE), [2] PG Scholar (DSCE), [3] PG Scholar (DSCE)
[1] Department of Electronics & Communication Engineering,
[1] JNTUH College of Engineering Hyderabad, Hyderabad, India.

Abstract: A General-purpose microprocessor with *limited processes*, is designed using both VHDL and Verilog HDL individually and are simulated using ModelSim-Intel FPGA Starter Edition Software as to demonstrate the operation of the processor and are synthesized and implemented in hardware on a field programmable gate array (FPGA) using Intel Quartus Prime Lite Design Software. A comparison is made between the two models designed using VHDL and Verilog HDL in terms of device utilization (logic elements), interface (pins), system timing (timing analysis)**.**

Index Terms – General-purpose microprocessor, VHDL, Verilog.

I. INTRODUCTION

A General-purpose microprocessor is capable of performing a variety of computations. In this, a sequence of instructions in the form of program are stored in the memory and are executed by the microprocessor and not directly hardwired each computation into the processor. Another computation can be performed by changing the program in the memory. It can also be viewed as a dedicated microprocessor when it is made to perform only one function, and that is to execute the program instructions.

VHDL (IEEE 1076-2019) and Verilog (IEEE 1364-2005) are IEEE standard languages used for the design and testing of electronic systems. They can be used to express designs in structural, behavioral or register-transfer-level architectures for the same circuit functionality and are implemented by the synthesis tools (computer aided design tools).

The goal of this paper is to compare VHDL and Verilog HDL through design of a General-purpose microprocessor using computer aided design tools such as ModelSim and Quartus Prime.

II. GENERAL-PURPOSE MICROPROCESSOR ARCHITECTURE

A General-Purpose Microprocessor can perform a variety of tasks under the control of software instructions. It mainly consists of Control Unit and Datapath and Interconnecting Buses. A General-Purpose Microprocessor called EC2 is designed in VHDL and Verilog.

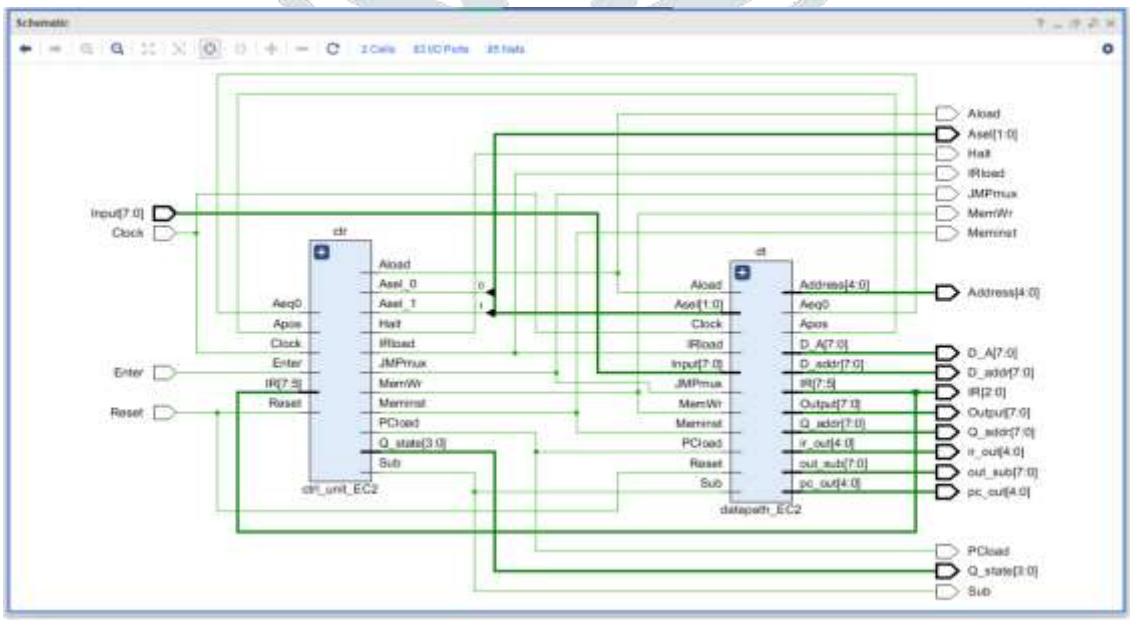


Fig.1 Logic Circuit of General-Purpose Microprocessor EC2 with Control Unit and Datapath.

The Datapath is responsible for all data manipulations such as storing of data, processing of data and flow & control of data. It consists of registers for temporary storage of the data, ALU for processing of data, Multiplexers and Data Bus for Data control and flow.

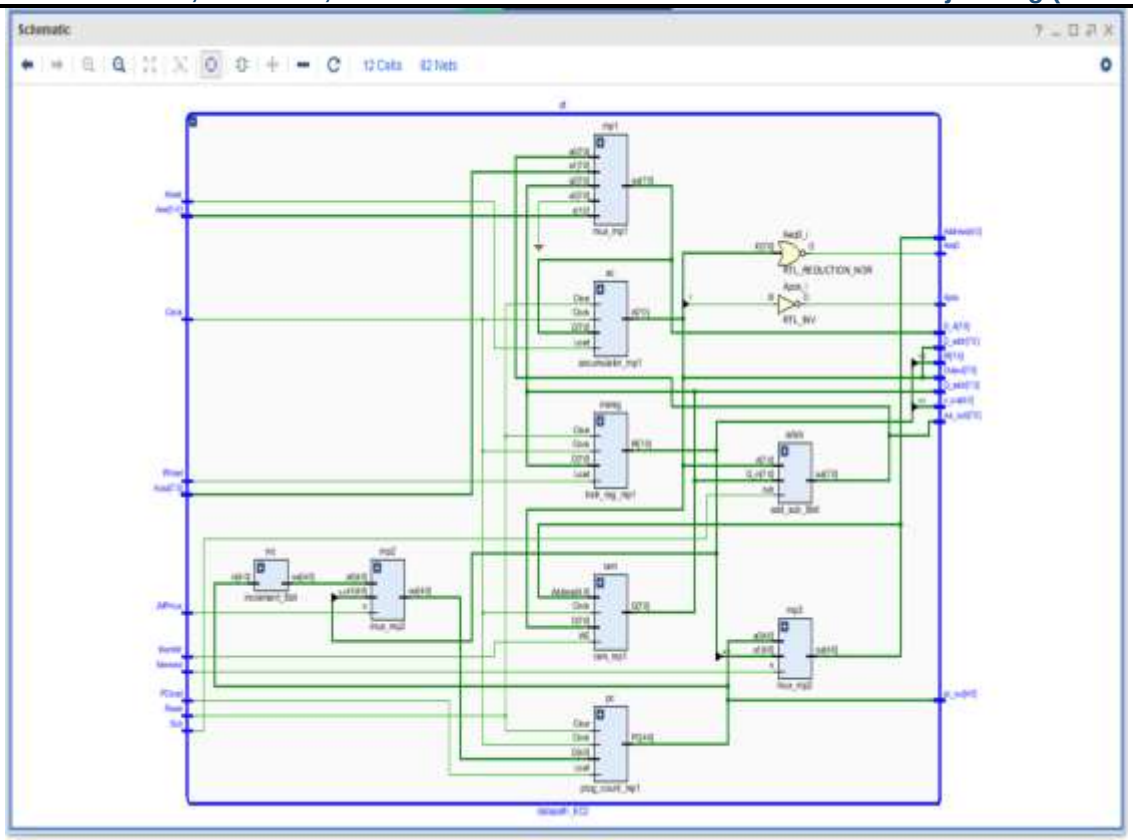


Fig.2 Logic Circuit of Datapath of General-Purpose Microprocessor EC2

The Control unit, also known as the controller, controls all of the operations of the Datapath, and therefore, the operations of the entire microprocessor. The control unit is a finite state machine (FSM) because it is a machine that executes by going from one state to another and that there are only a finite number of states for the machine to go to. The control unit consists of three parts: the next-state logic, the state memory, and the output logic. The purpose of the state memory is to remember the current state that the FSM is in. The next-state logic is the circuit for Digital Logic and Microprocessor Design with determining what the next state should be for the machine. And the output logic is the circuit for generating the actual control signals for controlling the Datapath.

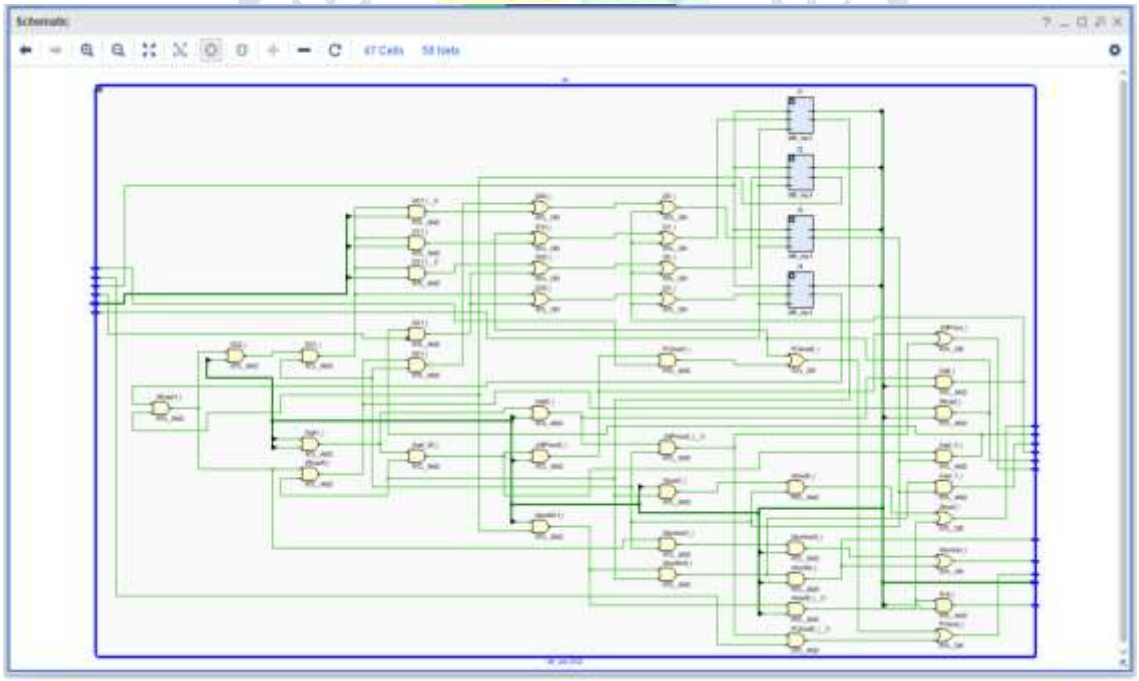


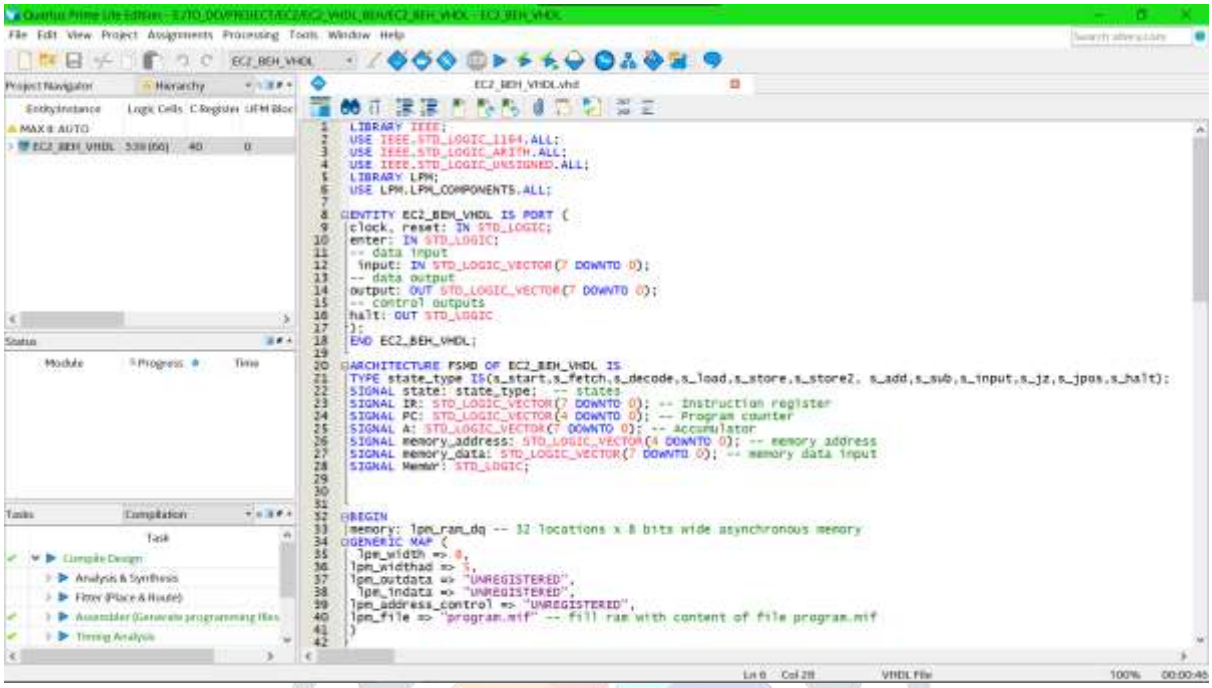
Fig.3 Logic Circuit of Control Unit of General-Purpose Microprocessor EC2

EC2 General-Purpose Microprocessor design consists:

i) Ports including clock (system clock reference), reset (Asynchronous Clear), enter (Input Enable), input (data input), output (data output), halt (control output);
ii) States (s_start, s_fetch, s_decode, s_load, s_store, s_store2, s_add, s_sub, s_input, s_jz, s_jpos, s_halt) for Control Unit (FSM);
iii) Instruction register (IR) (Holds Instructions), Program counter (PC) (stores memory address of current instruction), Accumulator(A), RAM (memory_address, memory_data, Memwr) for Datapath operations.

III. DESIGN USING VHDL

Entity is declared with ports: clock, reset, enter, input, output and halt.
Architecture consists of states declared as state_type (user defined data type using “TYPE”).IR, PC, A and RAM ports are declared as signals so as to store past values (register).
Asynchronous Ram is instantiated from altera LPM Library as lpm_ram_dq using port mapping by named association.
Process statement is used with clock and reset as sensitivity list. Case, When, If-else statements are used inside process.



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
LIBRARY LPM;
USE LPM.LPM_COMPONENTS.ALL;

ENTITY EC2_BEH_VHDL IS PORT (
clock, reset: IN STD_LOGIC;
enter: IN STD_LOGIC;
-- data input
 input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
-- data output
output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
-- control outputs
halt: OUT STD_LOGIC
);
END EC2_BEH_VHDL;

ARCHITECTURE FSMD OF EC2_BEH_VHDL IS
TYPE state_type IS(s_start,s_fetch,s_decode,s_load,s_store,s_store2, s_add,s_sub,s_input,s_jz,s_jpos,s_halt);
SIGNAL state: state_type;  -- states
SIGNAL IR: STD_LOGIC_VECTOR(7 DOWNTO 0); -- Instruction register
SIGNAL PC: STD_LOGIC_VECTOR(4 DOWNTO 0); -- Program counter
SIGNAL A: STD_LOGIC_VECTOR(7 DOWNTO 0); -- Accumulator
SIGNAL memory_address: STD_LOGIC_VECTOR(4 DOWNTO 0); -- memory address
SIGNAL memory_data: STD_LOGIC_VECTOR(7 DOWNTO 0); -- memory data input
SIGNAL Memwr: STD_LOGIC;


BEGIN
memory: lpm_ram_dq -- 32 locations x 8 bits wide asynchronous memory
GENERIC MAP (
 lpm_width => 8,
lpm_widthad => 5,
lpm_outdata => "UNREGISTERED",
 lpm_indata => "UNREGISTERED",
lpm_address_control => "UNREGISTERED",
lpm_file => "program.mif" -- fill ram with content of file program.mif
)
```

Fig.4 VHDL Design Code of General-Purpose Microprocessor.

IV. DESIGN USING VERILOG:

Module is declared with ports: clock, reset, enter, input, output and halt.
Since Verilog doesn’t support string types, states are encoded as 4-bit binary constants using “parameter”.
IR, PC, A and RAM ports (except memory_data) are declared as reg type so as to store past values (register) and memory_data is declared as wire so as to transfer output data from RAM.
Asynchronous Ram is instantiated from altera LPM Library as lpm_ram_dq using port mapping by named association.
always block is used with posedge clock and posedge reset as sensitivity list. Case, If-else statements are used inside always block.

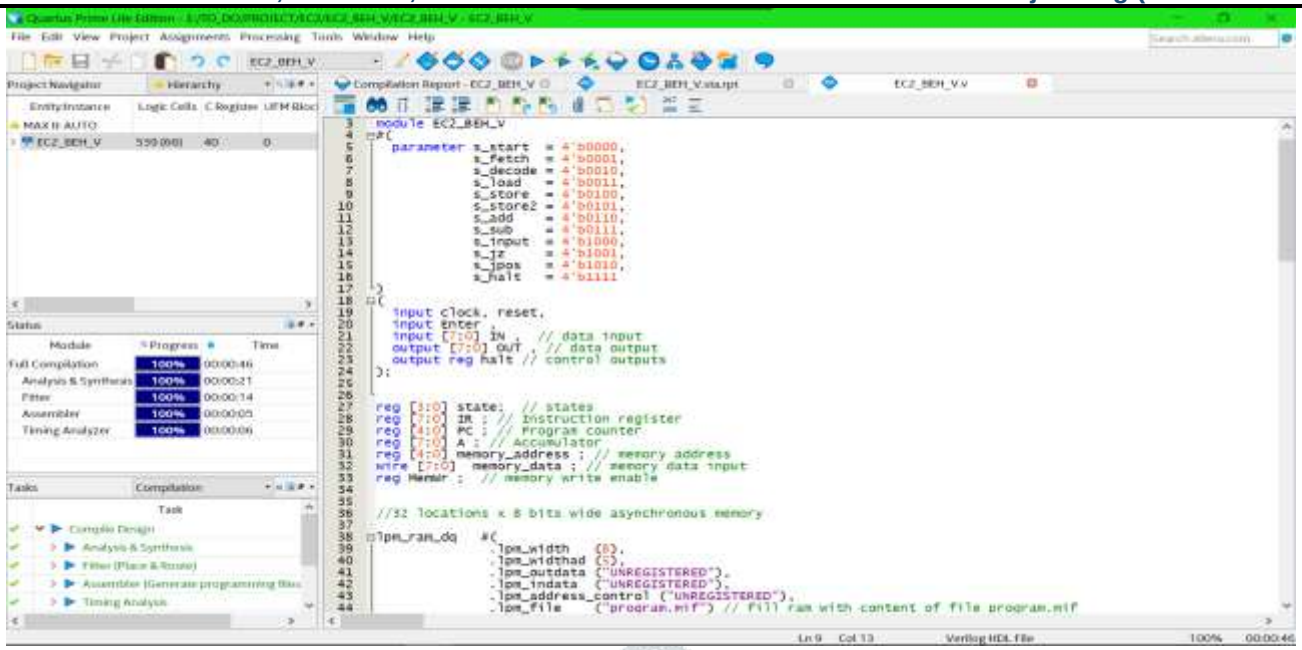


Fig.5 Verilog Design Code of General-Purpose Microprocessor.

V. RESULTS

5.1 Results of Simulation of the designs in ModelSim-Intel FPGA Starter Edition Software

Both designs are simulated in ModelSim and the outputs from the resulted waves are found to be same (i.e., both can produce the same output for the same inputs) as shown in Figure 6 and Figure 7.

Hence from the simulation perspective based on the result, it can be said that the two designs are equivalent.

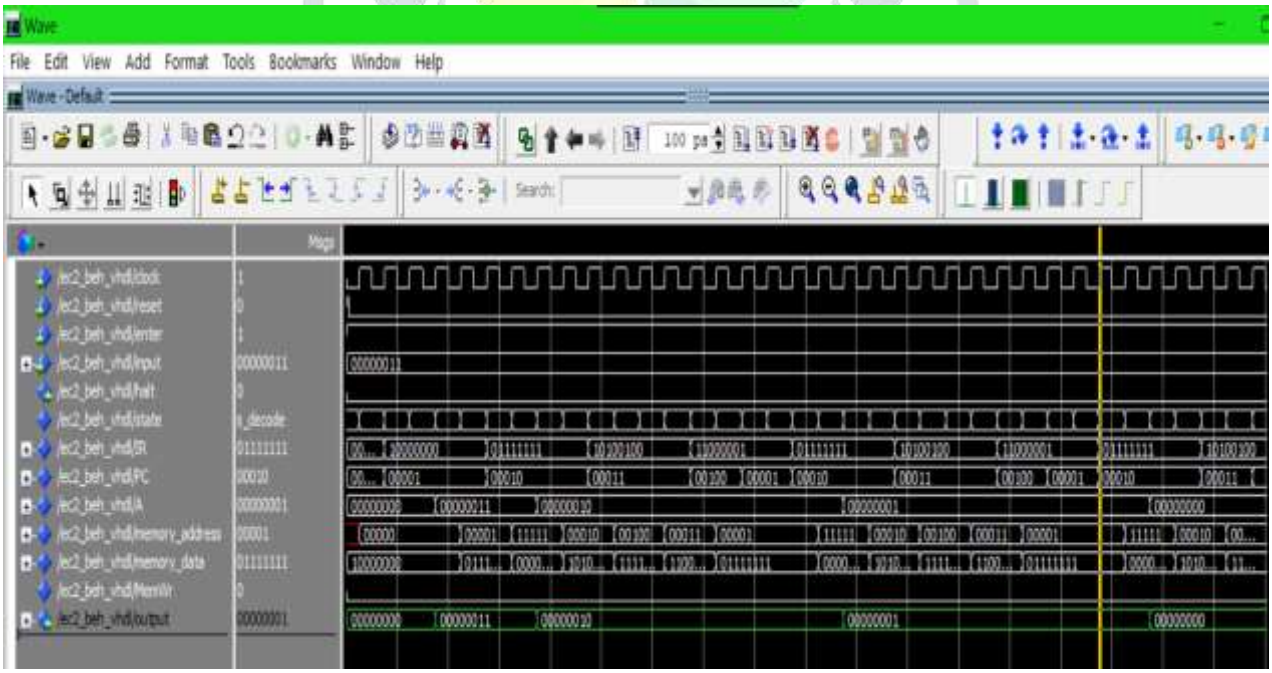


Fig.6 Simulation Result of Design Modelled Using VHDL

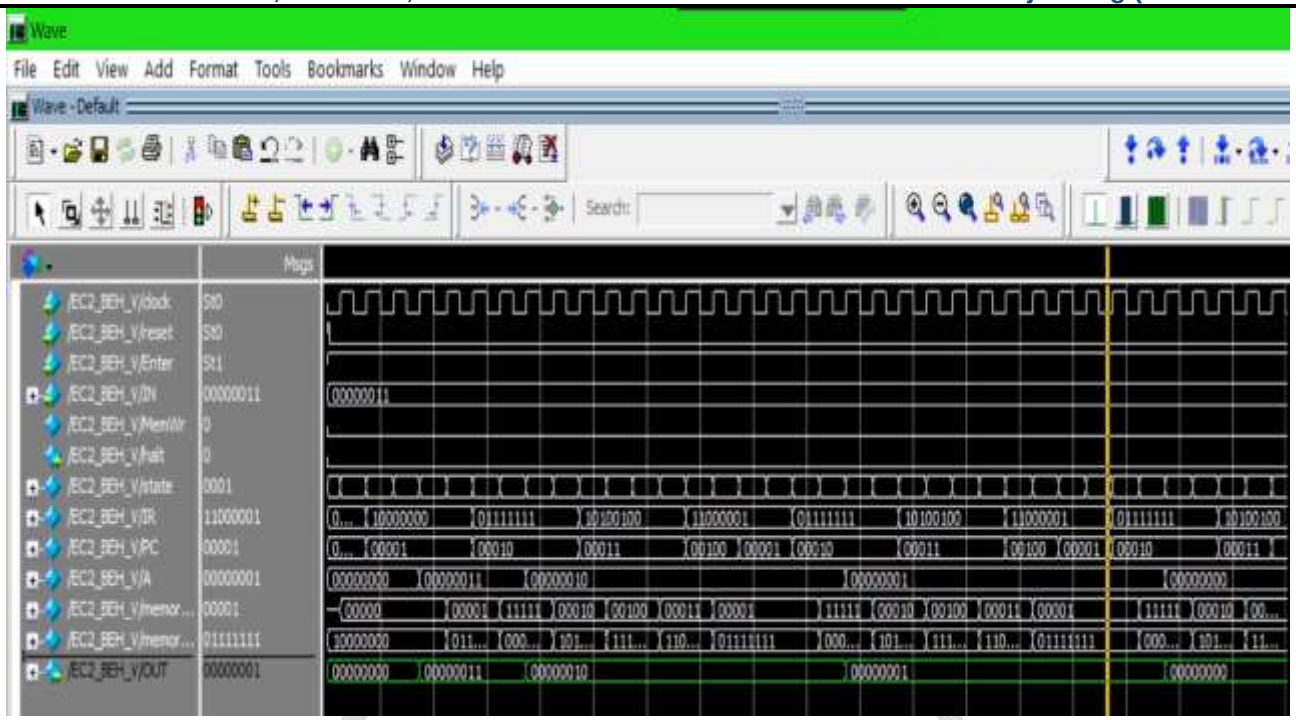


Fig.7 Simulation Result of Design Modelled Using Verilog

5.2 Results of Analysis & Synthesis of the designs in Intel Quartus Prime Lite Design Software

Both designs are fully compiled in Quartus Prime and are analyzed and synthesized and are found to be having same number of total logic elements as given in Figure 8 and Figure 9.
(As design becomes complex there may be difference in the number of logic elements)

For this design, from Analysis & Synthesis perspective the two designs are said to be equivalent.

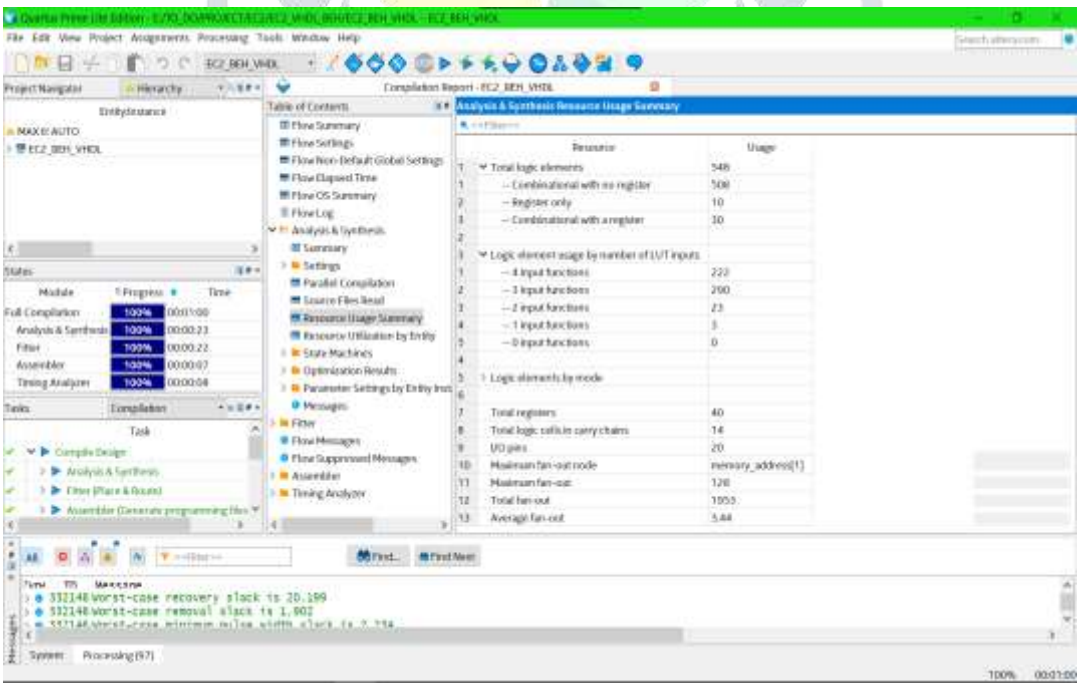


Fig.8 Resource Usage Summary of the Design Modelled Using VHDL

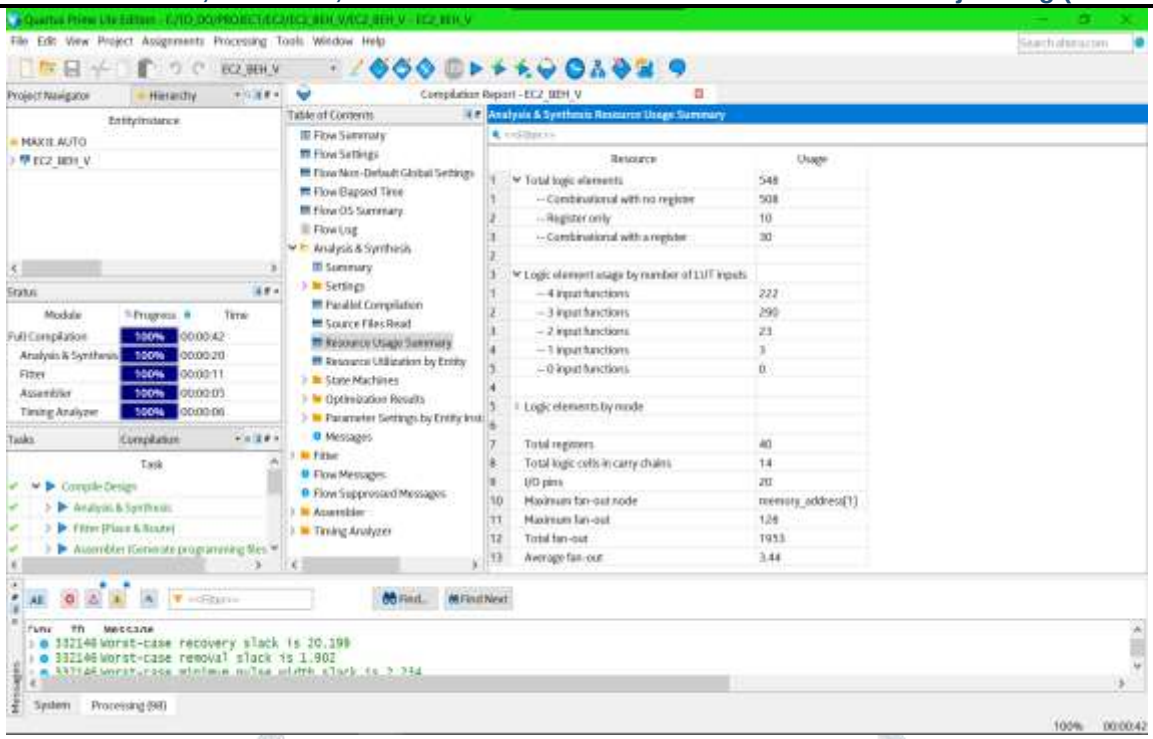


Fig.9 Resource Usage Summary of the Design Modelled Using Verilog

5.3 Results of Timing Analysis of the designs in Intel Quartus Prime Lite Design Software

Both designs are fully compiled in Quartus Prime and from Timing Analysis done based on Synopsys Design Constraints (SDC) provided while analysis, Maximum Frequency of operation (Fmax) is found to be same. It can be observed seen in Figure 10 and Figure 11.
(As design becomes complex there may be difference in the number of logic elements and hence difference is Fmax)

For this design, from Timing Analysis perspective the two designs are said to be equivalent.

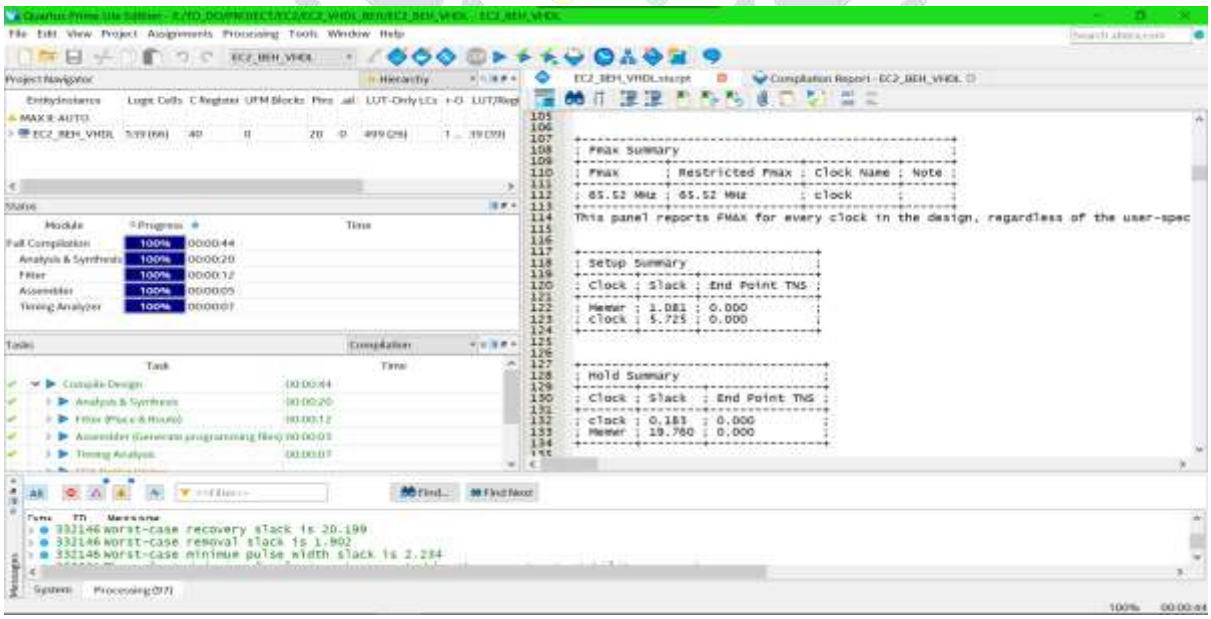


Fig.10 Static Time Analysis of Design Modelled Using VHDL

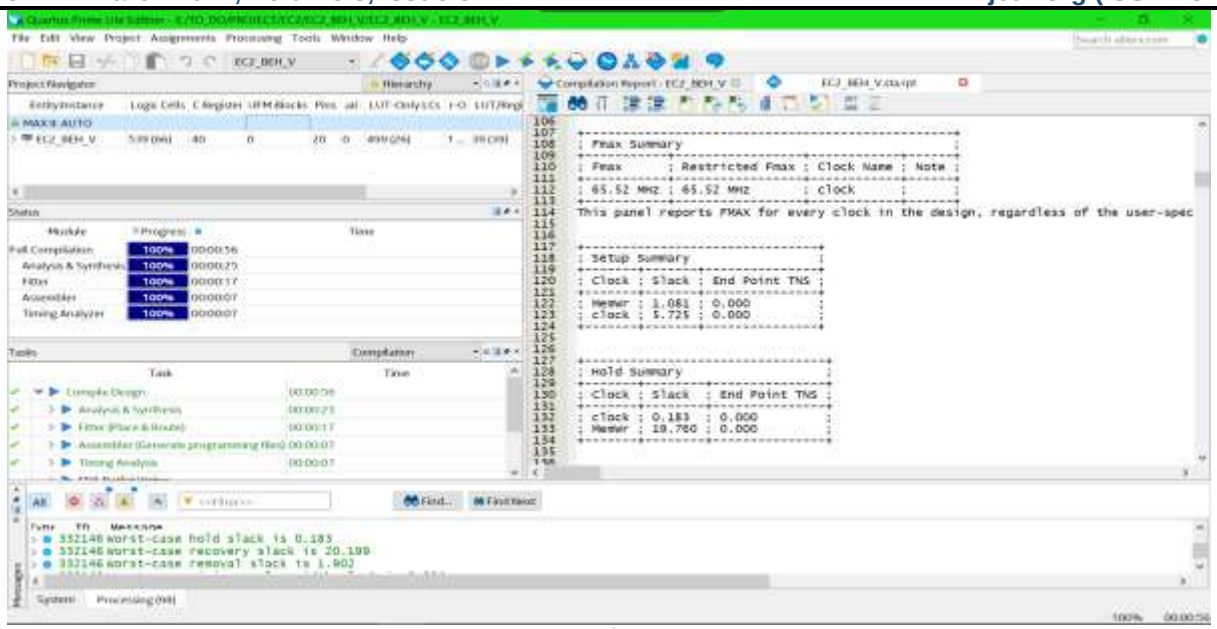


Fig.11 Static Time Analysis of Design Modelled Using Verilog

VI.CONCLUSION

For the EC2 General-Purpose Microprocessor, results from behavioral simulation, synthesis and timing analysis show that the designs modelled using both VHDL and Verilog are equivalent.

Depending upon the requirement of different designs the outcome may be differed as VHDL provides high-level hardware modeling while Verilog low-level hardware modeling

REFERENCES

[1] Maity, Niladri & Maity, Reshmi. (2007). VHDL and Verilog: Unbiased Compared and Contrasted. International Journal HIT Transaction on ECCN. 2. 356-363.
[2] Enoch O. Hwang, "Digital Logic and Microprocessor Design with VHDL", Nelson Engineering Publication.
[3] IEEE, "IEEE Standard Verilog: Language Reference Manual", IEEE,2005, std.1036-2005.
[4] IEEE, "IEEE Standard VHDL: Language Reference Manual", IEEE,2019, std. 1076-2019.
[5] Synopsys Inc., "HDL Compiler for Verilog Reference Manual", Version 2000.05, May.2000.
[6] Samir Palnitkar, "Verilog HDL, A Guide to Digital Design and Synthesis", Pearson Education Publication
[7] V.A. Pedroni, "Circuit Design and Simulation with VHDL", Prentice Hall India Learning Private Limited Publication