Design of an 8-Bit Five Stage Pipelined RISC Microprocessor for Sensor Platform Application

Reuben James L. Austria, Annaliza L. Sambile, Kahr-Lile M. Villegas, Jay Nickson T. Tabing
College of Engineering, First Asia Institute of Technology and Humanities
Tanauan City, Batangas, Philippines
reubenjamesaustria@gmail.com, annalizasambile@gmail.com, kahrlvillegas2@gmail.com, jttabing@firstasia.edu.ph

***Abstract*— Due to ongoing advancement in the technology known as the Internet of Things, different devices that utilize sensors have drawn a strong interest in the field of research and application in recent years. Hence, a specific device that could perform the overall operation and could process the data of the sensor platform known as microprocessor is also a requirement. Some existing general-purpose processors could be integrated in a sensor platform to provide the necessary processing needs. These processors are designed to execute multiple applications and perform multiple tasks. However, these general-purpose processors can be quite expensive especially for devices that are designed to perform specific tasks, such as sensor platforms. Therefore, application specific processors emerged as a solution for high performance and cost-effective processors. The main objective of this paper is to design an 8-bit five stage pipelined RISC Microprocessor for Sensor Platform applications. By gathering related researches, creating a prescriptive model and evaluating different microprocessor architectures, parameters for designing the specific microprocessor were identified. Instructions in the instruction set had been chosen by considering functionality, executing speed, and hardware cost. ASIC implementation is achieved through following the semi-custom ASIC digital design flow from RTL specification written in Verilog HDL, to functional verification using Universal Verification Methodology (UVM), to synthesis, and lastly verification of gate-level netlist. The design is also tested through FPGA for its hardware verification. The microprocessor can be programmed using an open source AVR assembler and is designed in a 90nm process technology using Synopsys tools.**

Keywords—RISC, Microprocessor, UVM, IoT

I. INTRODUCTION

Nowadays, technology has advanced to a level wherein it enables different devices to communicate with humans precisely and accurately. This development is achieved through the Internet of Things' (IoT) main principle which is to communicate devices with other devices without too much human intervention that is achieved by integrating sensors. These sensors are part of the major building blocks of IoT, for without them, there will be no data to be processed and passed around the IoT system. Part of this IoT system is the device that will process the data from the sensors and then pass it to the applications for proper utilization of all the data collected. The said device together with the sensor forms a "sensor node" in the system. A sensor node must be capable of some data processing, gathering sensory information and must be able to communicate with the network. Sensors alone cannot function as a sensor node, devices such as a sensor platform can provide the necessary functions. Sensor platform has an embedded microprocessor for data processing, can make use of the data from the sensors because it also has an Analog to Digital Converter (ADC), and can communicate with the network through its interfaces. Hence, sensors and a sensor platform together will form a sensor node which can be used in an IoT system. One of the most important part of the sensor platform will be the microprocessor. This will be responsible for the overall operation of the sensor platform. Sensors will provide the data but the microprocessor will perform computations and will do something useful with the data. With this, a microprocessor must be able to provide the proper series of high speed operations or tasks that will be used effectively to process data in a sensor platform.

II. RELATED STUDIES

In designing a microprocessor, there are several parameters that must be considered. Some of these parameters include: speed grade, throughput, number of bits that the microprocessor deals with at a time, number of instructions the microprocessor can execute, and other considerations that contributes for the performance of the microprocessor. Also, factors that include complexity, feasibility for implementation, design structure, and ability to be implemented in the available tools were also considered. In order to address and to know the considerations for these parameters, the following studies and microprocessor architectures were reviewed and analyzed.

1) *Unpipelined vs. Pipelined:* A study entitled "Review of five stage Pipelined Architecture of 8-bit Pico Processor" (2014) was conducted by S. Mishra and N. Sarwade. The paper was about the study of an unpipelined pico processor (pP) and how its overall throughput can be increased by implementing pipelining. This study takes multiple clock cycles to execute one instruction. On each cycle, one step of instruction interpretation is performed, and the machine state is updated at the end of the cycle. Different instructions may take different number of cycles, depending on the interpretation steps required. Though this organization allows faster cycle times, the overall throughput can still be improved by pipelining.

978-1-5090-1134-6/17/$31.00 ©2017 IEEE

Applying the five stage pipelined architecture shown in Figure 2.1 to the processor improved its overall throughput. This is because as one instruction is being executed the next instruction is already being fetched increasing the amount of data being processed per clock cycle (throughput).

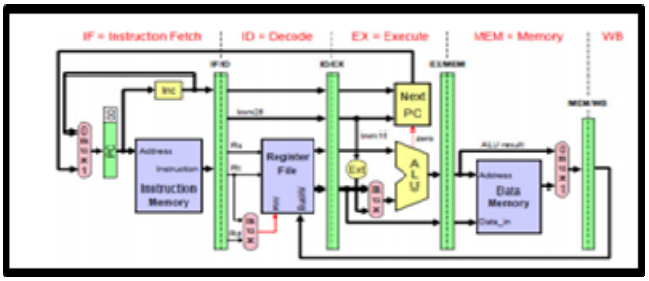

Figure 2.1 Five stage pipelined architecture with registers

2) *Architecture Comparison:* A study entitled "The Architecture Comparison and the VLSI Implementation of the 32-bit Embedded RISC" (2003) was implemented by X. Ke, et al. The goal of this study is to implement a low-cost and high speed 32-bit microprocessor. In order to do this, several aspects of computer architecture were compared and the most suitable sets was chosen. The study is divided into the following categories:

- *Comparison in internal storage architecture.* There are three major classification for internal storage namely, stack architecture, accumulator architecture or general-purpose register architecture (GPR). Both stack and accumulator follows implicit operands while GPRs have only explicit operands—either registers or memory locations. According to this study, GPR is most likely to be used since registers are faster than memories and these are easier for a compiler to use. In terms of timing, stack machines must spend clock cycles to move operands to the top of the stack and move results off the top of the stack. On the other hand, a register machine spends no clock cycles since a register file provides random access to many different values simultaneously.

- *Comparison in memory combination.* Von Neumman architecture is defined as the combination of instruction memory and data memory. These memories are placed in a physical memory which can help in reducing the total number of the data bus and address bus. However, its disability of simultaneous fetching data and instruction can greatly harm the pipeline flow in the instruction execution which contributes significantly to the overall speed of the processor. These unnecessary interlock cycles have to be eliminated during high performance microprocessor design. In order to do so, Harvard architecture must be adopted which has separate instruction and data memory, thus allowing instruction fetch to occur in parallel with data access.

- *Comparison in pipeline architecture.* Pipeline is the key implementation technique used to speed up microprocessors today. There are two most popular pipeline stage partitions: three-stage pipeline and the five-stage pipeline. A three-stage pipeline is composed of Instruction Fetch (IF) that fetch the instruction from instruction memory, Instruction Decode (ID) that decodes the instruction latched from the IF stage, generate control signals for the following stage, and Execute (EXE) that read operands, execute in ALU, access memory and write the results back. With this, the pipeline is not perfectly balanced because the last stage is burdened with too many works and thus becomes the critical path of the design and limits the frequency improvement of the processor. On the other hand, five-staged pipeline is composed of Instruction Fetch, Instruction Decode, EXE, Direct Memory (DM) which provides access to memory, and Write Back (WB) that writes the data latched in DM stage back to register file. Compared with three-staged pipeline, the EXE stage is divided into three independent stages as EXE, DM and WB.

This study was verified and compared to the commercial ARM7TDMI, which is a Von Neumman architecture (CISC) with three-staged pipeline, their design outweighs the latter in the maximum frequency as well as overall speed.

With the ideas presented, the proponents decided to study and design an 8-bit five stage pipelined RISC Microprocessor for Sensor Platform applications to provide high performance and cost-effective microprocessors specifically designed for a sensor platform with the following specifications as shown in Table 1.

TABLE 1
MICROPROCESSOR SPECIFICATIONS

Microprocessor Characteristic	**Specification**
Number of bits:	8-bits
Pipelining Technique:	In-order execution
Number of pipeline stages:	Five stages
Architecture:	Reduced Instruction Set Computer (RISC)
I/O Design and Organization	Interrupt-driven
Internal Architecture:	Register to register
Data Hazard	Mixing stalls and forwarding
Control Hazard	Delayed branch

III. Experimental Sections

The overview of the microprocessor block diagram shown in Figure 3.1 was designed to be implemented as a five-stage pipeline. It is composed of Fetch Stage, Decode Stage, Execute Stage, Memory Stage and Write Back Stage. Between these stages, pipeline registers are placed. These pipeline

registers allow the processor to start on a new instruction, without waiting for the current instruction to finish. These registers also ensure stability of control signals and data inside the processor, which in turn ensures proper execution of instructions.

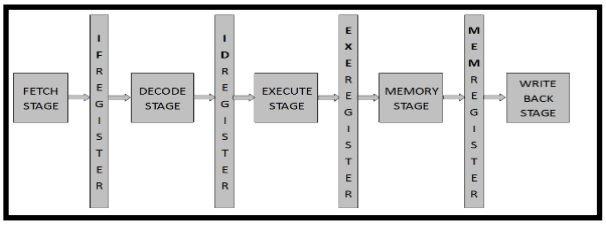

Figure 3.1 Overview of the microprocessor design

A. Five Stage Pipeline Design

1) Fetch Stage: The fetch stage contains the program counter (PC) and the program memory. The PC is a register which points to an address in the program memory. The value of PC is incremented every clock cycle, unless there is an instruction which disrupts the program flow, where a new value will be loaded to the program counter. The value of PC is also held during pipeline stalls. The PC is also loaded with the interrupt vector during interrupts.

2) Decode Stage: The decode stage contains the control unit, GPR and multiplexer modules. The instruction from the fetch stage will be decoded in this stage. The addresses of the operand registers are given to the GPR to obtain the values from these addresses. The control unit decodes the opcode and generates necessary control signals for the subsequent stages.

3) Execute Stage: The execute stage contains the Arithmetic Logic Unit (ALU), the carry register and the zero register. This stage is responsible for executing the instructions interpreted from the previous stage. The ALU performs the arithmetic and logic functions of the processor. Its inputs include the two operands and the control signals from the decode stage. The carry register holds the value of carry from the result of the ALU. This module is responsible for saving and returning the carry value during interrupts. On the other hand, the zero register which holds the value of the zero from the ALU is responsible for saving and returning the zero value during interrupts.

4) Memory Stage: The memory stage contains the data memory. Instructions associated with the data memory like the LDS (load) and STS (store) instructions are performed on this stage.

5) Write Back Stage: The write back stage is where the output data from the execute stage is written back to the address of the destination register. This destination address is decoded from the instruction.

Pipeline Hazards: Pipeline hazards occurs when the pipeline must stall because conditions do not permit continued execution. Data Hazard occurs when there is a conflict in access of an operand location or during data dependencies where the result of the execution is needed in the next instruction. Control Hazard, occurs when the pipeline makes the wrong decision on a branch prediction, where the processor looks ahead in the instruction code fetched from memory and predicts which branches are likely to be processed. Therefore, this hazard brings instructions into the pipeline that must subsequently be discarded.

Data hazards are resolved by use of data forwarding. If the source address of the current instruction is the destination address of the previous instruction, the data is sent back to the ALU. This means that output data from the ALU can be used by the succeeding instructions without waiting for the data to be written back to the register file. This also applies to dependent instructions one clock cycle apart, it would only mean that instead of forwarding the data from the ALU, data from the data memory will be forwarded.

Output data from the ALU and data memory are connected to the operand inputs of the ALU through a multiplexer. Control signals for these multiplexers are generated by the data forwarding unit. The data forwarding unit checks if the current output will be written back to the register file, then checks if the source address at the execute stage is equal to the destination address at the memory or write-back stages.

Another data hazard arises when the current instruction relies from the previous LW (Load Word instruction), the data from the memory will only be valid when it is loaded to the MEM/WB register, meaning the data will not be valid by the time the execute stage will need the data. This hazard cannot be resolved by using the forwarding unit. This data hazard can be resolved by stalling the pipeline using Hazard Detection Unit. This can be done by preventing the program counter from incrementing, and writing a NOP (No operation) instruction in the ID/EX register.

The overall circuitry of the five stage pipelined microprocessor block diagram is shown in Figure 3.2 where all the components of each stage in the block diagram of the microprocessor are connected.

Since this microprocessor is specifically designed for sensor platform applications, it is only capable of performing necessary instructions that could be used by sensor platforms. The instruction set summary of this microprocessor is shown in Table 2.

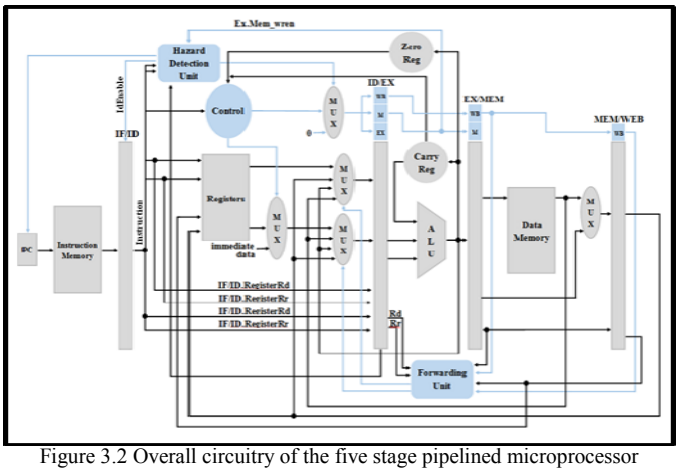

Figure 3.2 Overall circuitry of the five stage pipelined microprocessor

TABLE 2
INSTRUCTION SET SUMMARY

MNEMONIC	DESCRIPTION
ADD	Adds the contents of two registers
SUBI	Subtracts the content of a register and an immediate value K
SUB	Subtracts the contents of two registers
AND	Bit-wise logical AND operation on the contents of two registers
ANDI	Bit-wise logical AND operation on the contents of a register and an immediate value
OR	Bit-wise logical OR operation on the contents of two registers
EOR	Bit-wise EOR operation on the contents of two registers
ORI	Bit-wise logical OR operation on the contents of a register and an immediate value
COM	One's complement
NEG	2's complement
LSL	Logical shift left right of the contents of Rd
LSR	Logical shift right of the contents of Rd
CP	Compares if Rd and Rr are equal
CPI	Compares the contents of Rd with a constant K
BRNE	If the contents of Rd and Rs are not equal, the PC is loaded with PC+1+k
BREQ	If the contents of Rd and Rs are equal, the PC is loaded with PC+1+k
LDS	Loads a word from the data memory on address k to Rd
STS	Stores the word from Rd to address d on the data memory
RETI	Loads the return address to PC
IN	Reads data from I/O device at address p and stores the data to Rd
OUT	Stores data from Rs to I/O device at address p
NOP	No operation
SBC	Subtracts the value of the carry and the data from the source register from the data at the destination register
ADC	Adds the value of the carry and the data from the source register to the data at the destination register
RJMP	PC is loaded with PC+1+k
CPC	Compares the data at the destination register to the data at the source register plus the carry
BRCS	If the carry is set, PC is loaded with PC+1+k
BRCC	If the carry is not set, PC is loaded with PC+1+k
INC	Increments the data at the destination register by 1
DEC	Decrements the data at the destination register by 1
RET	Loads PC with the saved address
RCALL	PC is loaded with PC+1+k, current PC is saved
MOV	Copies the data from the source register to the destination register

B. Procedure

This design follows the ASIC digital design flow from design specifications up to Logical Verification and Testing.

1) *Design Specifications:* The first step in designing is to identify the design specifications wherein all the required researches in designing a microprocessor must be considered and evaluated if it meets the parameters that were set.

2) *Behavioral Description:* After listing all the specifications of the design, the next step in the design flow is to come up with Structural and Functional Description. Meaning, the designer has to decide what kind of structure/architecture would be used for the design (e.g. RISC/CISC, pipelining, etc.).

3) *RTL Description:* The RTL is expressed in Verilog HDL which the digital system (e.g. microprocessor, memory, etc) of the behavioral description are described.

4) *Functional Verification:* Functional Verification must be performed to ensure the RTL designed matches the specifications. If it doesn't match with the specification, the RTL must be checked and edited. At this stage, UVM was used wherein a controlled environment shown in Figure 3.3 was used.

5) Different tests were performed in order to functionally verify the RTL. A snip of the

verification plan for the top module is shown in Figure 3.4.

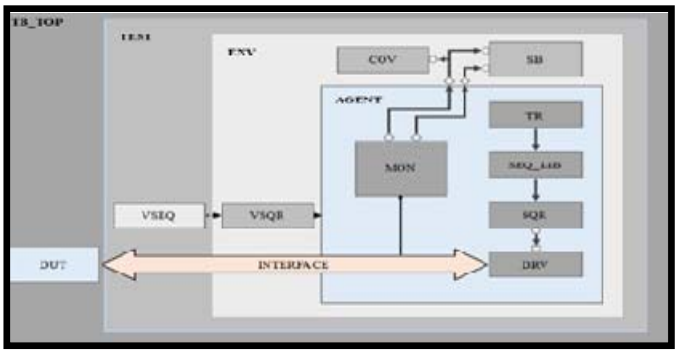

Figure 3.3 Testbench environment

Name	Type (chk, test, cov)	Priority	Description/Implementation Notes
test_reset_random	test	1	1. generate clk signals. 2. set random reset signal after some posedge clk. 3. generate random interrupt_request, interrupt_vector, data_in and opcode_in.
test_opcode	test	1	1. generate clk signals. 2. set reset signal from 0 to 1 and then 1 to 0 in order to reset. 3. generate opcode_in for each instruction. 4. generate random interrupt_request, interrupt_vector, and data_in.
test_data_in	test	1	1. generate clk signals. 2. set reset signal from 0 to 1 and then 1 to 0 in order to reset. 3. generate all possible data_in to test the ggr. 4. generate random interrupt_request, interrupt_vector, and opcode_in.
test_interrupt	test	1	1. generate clk signals. 2. set reset signal from 0 to 1 and then 1 to 0 in order to reset. 3. test interrupt by asserting the interrupt_request signal for all the instructions. 4. generate random interrupt_vector, data_in and opcode_in.

Figure 3.4 Top module verification plan

6) *Logic Synthesis, Gate Level Netlist and Logical Verification and Testing:* Once the RTL is verified, it will then be converted into an optimized gate level netlist. This is done using Synopsys tool which is Design Compiler. This takes an RTL hardware description together with a standard cell library as input and produces a gate level netlist as output. Constraints such as timing, area, testability, and power are considered. Synthesis tools try to meet constraints, by calculating the cost of various implementations and then generate the best gate level implementation. This netlist is a completely structural description with only standard cells. This must also be verified in order to know if the gate level conversion has been correctly performed.

IV. Results and Discussions

A. Verification Results

In order to verify the functionality of the RTL design, it is verified through software and hardware. Software verification utilizes UVM while the hardware verification is done through FPGA implementation. Software verification focuses on verifying and checking if the logic design, RTL, conforms to specifications. On the other hand, hardware verification determines if the RTL correctly implements the specifications.

Software Verification: Different verification plans and tests were performed in order to verify the functionality of the designed RTL. The waveform result for the top module verification of the microprocessor is shown in Figure 4.1. It shows that the implementation of pipeline since fetch, decode, execute and memory processes and produce outputs every next posedge clk. Therefore, as one instruction is being processed per clk cycle, another instruction is already being fetched without waiting for the current instruction to be written back.

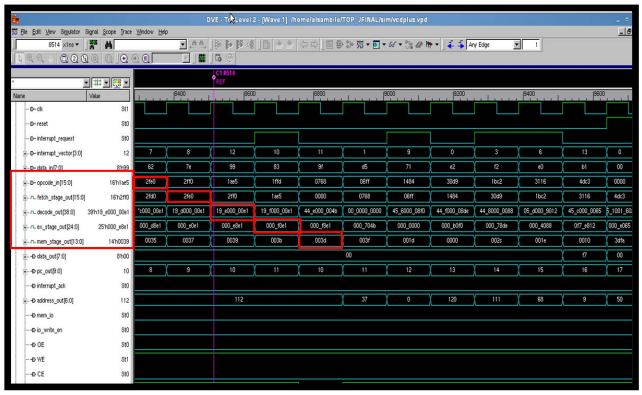

Figure 4.1 Functional verification result

The summary of the result of the tests as discussed in the verification plan for the top module is shown in Table 3.

TABLE 3
TOP MODULE VERIFICATION SUMMARY

Test Name	Remarks
Test_reset_random	*DUT Passed*
Test_opcode	*DUT Passed*
Test_data_in	*DUT Passed*
Test_interrupt	*DUT Passed*

Separate verification environments for each stage and the top block was created. This is supported by simulation results presented wherein the design passed all the tests and obtained 99.67% code coverage and 100% functional coverage.

Hardware Verification: Hardware verification was done by implementing the RTL design in an FPGA. Program code was written in assembly language, then it was converted into intel hex file format using avr assembler. The output file was then converted into a Xilinx coefficient file, this coefficient file was used to initialize the single port ROM instantiated in the FPGA which serves as the program memory. The data memory was implemented using a 128x8 RAM with asynchronous read and synchronous write. Other modules such as I/O registers, Seven-segment led decoders and others were also instantiated with the microprocessor depending on the desired output. The designed microprocessor was able to perform the assembly coded functions such as to control ADC and clock which are the main components in a sensor platform.

The tests performed include LED Curtain and Up-Counter with UART interrupt. The LED curtain test aims to verify the capability of the processor to write desired outputs to the

connected peripherals. On the other hand, the Up-Counter with UART interrupt test aims to verify the interrupt-handling capabilities of the processor. A module which asserts its output for one clock cycle every second is also instantiated, this output is connected to the interrupt request port of the processor. During interrupt requests, the microprocessor reads from the FPGA GPIOs and execute the ADC output to ASCII converter routine. The output of the interrupt routine was then written to the memory of the UART transmitter and then sent to the PC through the UART module and the AccessPort application. The result of the LED curtain test and up-counter with UART interrupt test is shown in Figure 4.2.

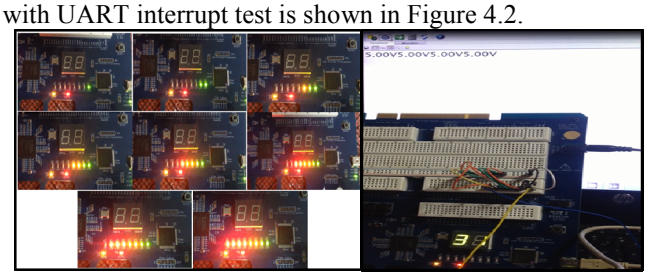

Figure 4.2 FPGA results

1) Synthesis Results: The constraints such as area and timing of the design which is determined by the design's architecture and the coding style were applied. The translation of the design from RTL code to the design implementation in terms of generic boolean logic gates with an operating frequency of 8MHz is shown in Figure 4.3. The report on timing is shown in Figure 4.4 which resulted to a positive slack.

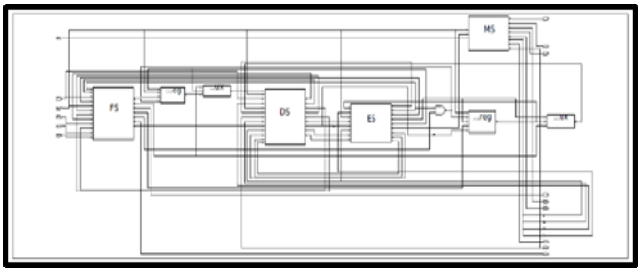

Figure 4.3 Synthesized Design

```
****************************************
Report : timing
Design : RISC_TOP
Version: L-2016.03-SP2
Date   : Thu Feb  9 14:00:15 2017
****************************************


  Startpoint: data_in[7] (input port clocked by clk)
  Endpoint: data_out[7]
            (output port clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  RISC_TOP           35000                 saed90nm_typ

  Point                                    Incr       Path
  ------------------------------------------------------------
  input external delay                    37.00      37.00 r
  data_in[7] (in)                          0.00      37.00 r
  MS/MDN1/U2/Q (AO22X1)                    0.12      37.12 r
  data_out[7] (out)                        0.00      37.12 r
  data arrival time                                  37.12

  max_delay                               60.00      60.00
  output external delay                  -17.00      43.00
  data required time                                 43.00
  ------------------------------------------------------------
  data required time                                 43.00
  data arrival time                                 -37.12
  ------------------------------------------------------------
  slack (MET)                                        5.88
```

Figure 4.4 Timing Report

2) *Formal Verification Result:* After synthesizing, both the RTL and the netlist underwent formal verification. It is the process of checking whether a design satisfies some requirements. Figure 4.5 shows the result of the formal verification wherein all the compare points matched.

```
************************** Matching Results **************************
491 Compare points matched by name
0 Compare points matched by signature analysis
0 Compare points matched by topology
31 Matched primary inputs, black-box outputs
0(0) Unmatched reference(implementation) compare points
0(0) Unmatched reference(implementation) primary inputs, black-box outputs
**********************************************************************

Status:  Verifying...

************************** Verification Results **************************
Verification SUCCEEDED
--------------------------
 Reference design: r:/WORK/RISC_TOP
 Implementation design: i:/WORK/RISC_TOP
 491 Passing compare points
--------------------------
Matched Compare Points    BBPin    Loop    BBNet     Cut     Port     DFF     LAT    TOTAL
--------------------------------------------------------------------------------------
Passing (equivalent)          0       0        0       0       31     460       0      491
Failing (not equivalent)      0       0        0       0        0       0       0        0
```

Figure 4.5 Formal Verification Result

3) Functional verification of gate-level netlist: The synthesized design underwent a functional verification to verify if the outputs of the RTL code is equivalent to the translated gate-level netlist's output. The waveform diagram in Figure 4.6 shows that the design is the same to the synthesized design since all the tests performed on the top module passed. Therefore, it can be said that the timing requirements of the synthesis is enough for the gate-level netlists to drive inputs and generate outputs.

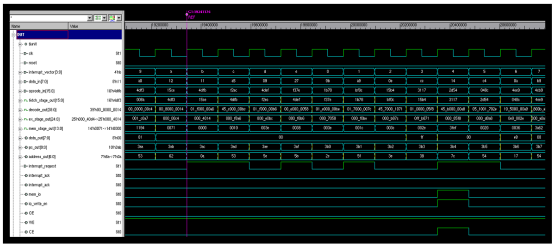

Figure 4.6 Functional verification of gate-level netlist result

V. Conclusions

This paper has presented a design, verification and synthesis of an 8-bit five stage pipelined RISC microprocessor for sensor platform applications. A RISC architecture is used that has separate instruction and data memory which allows pipelining wherein it improves the speed and throughput of the microprocessor. UVM was used in testing and the design passed all the test cases and obtained high coverage results. The design is synthesized in a 90nm process technology using the saedPDK Synopsys library. After synthesizing, a gate-level netlist was generated which underwent functional and formal verification. As a result, the RTL is correctly represented as a netlist. The design was also implemented and tested in FPGA and it executed the Assembly coded programs.

VI. Recommendations

For future works on designing a microprocessor for sensor platform applications, the researchers would like to recommend that the design must undergo the process of place and route, and incorporate the design in a sensor platform.

References

[1] Ke, X., Kejia, Z., Qiang, L., & Hao, M. The Architecture Comparison and the VLSI Implementation of the 32-Bit Embedded RISC. 2003.

[2] Mishra, S. K., Sarwade, D. P. Review of Five Stage Pipelined Architecture of 8-Bit Pico Processor. 2014.

[3] Sil, A., et, Al. An Energy-Efficient 32-Bit Processor for Sensor Platform in 90nm Technology. 2012. University of Louisiana at Lafayette.