UNIVERSITAS INDONESIA

CALCULATOR PROJECT USING VHDL

SKRIPSI

MOHAMAD SANDI SURIAGEMILANG
0405830075

DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNIK UNIVERSITAS INDONESIA
DEPOK
JANUARI, 2010

UNIVERSITAS INDONESIA

CALCULATOR PROJECT USING VHDL

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik

MOHAMAD SANDI SURIAGEMILANG
0405830075

DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNIK UNIVERSITAS INDONESIA
DEPOK
JANUARI, 2010

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar

Nama	: Mohamad Sandi Suriagemilang
NPM	: 0405830075
Tanda Tangan	:
Tanggal	: 21 Januari 2010

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh :
Nama : Mohamad Sandi Suriagemilang
NPM : 0405830075
Program Studi : Teknik Elektro Internasional
Judul Skripsi : *Calculator Project Using VHDL*

Telah berhasil dipertahankan dihadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Strata 1 pada Program Studi Teknik Elektro, Fakultas Teknik, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing : Muhammad Salman, ST., MIT ( )

Penguji : Abdul Muis, ST ( )

Penguji : Prof. Dr. Ir. NR. Poespawati, MT. ( )

Ditetapkan di : Depok

Tanggal : 6 Januari 2010

KATA PENGANTAR

Puji syukur penulis sampaikan kepada Allah SWT atas segala karunia dan Rahmat-Nya sehingga skripsi ini dapat terselesaikan. Saya menyadari bahwa tanpa bantuan dan bimbingan dari berbagai pihak, dari masa perkuliahan sampai pada penyusunan skripsi ini, sangatlah sulit bagi saya untuk memperoleh gelar sarjana. Penulis mengucapkan terima kasih kepada :

1. Dr. Jasmine Banks dari Queensland University of Technology (QUT) dan Muhammad Salman, ST., MIT sebagai dosen pembimbing yang telah meluangkan waktunya untuk memberikan arahan, bimbingan dan diskusi sehingga skripsi ini dapat terselesaikan dengan baik.
2. Orangtua yang tidak kenal lelah untuk selalu mendukung dan membuka wawasanku, kakak dan adik-adikku yang selalu memberikan motivasi
3. Rekan satu tim dalam skripsi ini, Faristama Aryasa yang telah membantu saya dalam mengerjakan skripsi ini
4. Seluruh civitas akademika Departemen Elektro Universitas Indonesia yang tidak dapat disebutkan satu persatu.

Akhir kata, semoga Allah SWT. Membalas segala kebaikan semua pihak yang telah membantu penyusunan skripsi ini dengan balasan yang setimpal. Semoga skripsi ini dapat berguna bagi siapa saja yang membacanya

Depok, 21 Januari 2010

Penulis

HALAMAN PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Sebagai civitas akademika Universitas Indonesia, saya yang bertanda tangan di bawah ini :

Nama : Mohamad Sandi Suriagemilang
NPM : 0405830075
Program Studi : Elektro Program Internasional
Departemen : Elektro
Fakultas : Teknik
Jenis Karya : Skripsi

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif** ***(Non-Exclusive Royalty-Free Right)*** atas karya ilmiah saya yang berjudul :

CALCULATOR PROJECT USING VHDL

Dengan Hak Bebas Royalti Non Eksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk Basis Data, merawat dan mempublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis / pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya

Dibuat di : Depok
Pada tanggal : 21 Januari 2010

Yang menyatakan

(Mohamad Sandi Suriagemilang)

ABSTRAK

Nama : Mohamad Sandi Suriagemilang
Program Studi : Teknik Elektro Internasional
Judul : *Calculator Project Using VHDL*

Skripsi ini membahas kemampuan mahasiswa Fakultas Teknik UI angkatan 2005 dalam membuat suatu program kalkulator digital dengan menggunakan VHDL. Percobaan ini adalah percobaan kualitatif dengan desain rekayasa pemrograman. Hasil percobaan menunjukan bahwa program kalkulator dapat terselesaikan dengan mengacu kepada prinsip-prinsip digital dan pemrograman *assembly.* Terdapat setidaknya empat keuntungan dalam pemrograman menggunakan VHDL : spesifikasi exekusi, ketidakbergantungan pada teknologi dan peralatan, data desain yang dapat dibawa-bawa, dan mensimulasikan secara awal dan cepat

Kata kunci :

program kalkulator digital, prinsip – prinsip digital dan pemrograman assembly.

ABSTRACT

Name : Mohamad Sandi Suriagemilang
Study Program : Teknik Elektro Internasional
Title : *Calculator Project Using VHDL*

The focus of this study is the freshmen student of Department of Electrical Engineering at University of Indonesia experience of making a digital calculator program by using VHDL. The result of this project shows that calculator program can be made based on digital principals and assembly programming. There are at least four benefits by programming using VHDL : executable specification, technology and tool independence, portable design data , and simulate early and fast.

Keywords:
digital calculator program, digital principals and assembly programming

DAFTAR ISI

CHAPTER I
INTRODUCTION

As part of their Engineering Degree undergraduate students at Queensland University of Technology. undertaking EEB839-2, were allocated into groups or single person to complete a supervised project

The initial aim of this project is to create FPGA through the use of Xilinx's Spartan-E development board Starter kit. The author had done it for EEB989-1, but because of one, reason, the supervisor of this project changed the title become Calculator Project Using VHDL. The reason was the author hadn't taken ENB.344 which was the requirement to do the project.

Students were, to interact with their d present to them regular progress reports. The scope of each project encompassed: the definition of the project topic, completion of literature review, formulation of the project specification, planning and timetabling of the project for the completion of tasks, a. discussion of the obtained results, and a submission of final report for assessment.

In conclusion, Mohamad Sandi Suryagemilang and Faristama Aryasa, after discussion with their supervisor, decided to research Field Programmable Gate Arrays (FPGAs), in particular making calculator via VHDL(VHSIC Hardware Description Language).

CHAPTER II
LITERATURE REVIEW

2.1 Introduction

FPGA refers, to Field Programmable Gate Arrays, It has a ability of complete re. programmability

Initially, in this project, the writer used FPGA through the use**,** of Xilinx's Spartan-3E development board starter kit, and afterward it would go to a more complex project e.g. uploading the Pico Blaze microcontroller.

2.1.1 FPGA

In industry applications, there ax two types of chips that are commonly used in, The first one is FPGA and second one is Application Spec[i]fic Integrated Circuit (ASIC)

FPGA was PCB contained many transistors came from logic units like adders, counters, ;mixes and even microconrollers.

11.2 VHDL

VHDL is abbreviation of VHSIC (Very High Speed Integrated Circuits) Hardware Description Language. It is a programming language as link of FPGAs and ASICs

```
ENTITY dff is                          - Entity declaration
Port (d, clk : IN STD LOGIC            - Two inputs of the type Std Logis
q.qbar: OUT STND_LOGIC)                - Two outputs of the type Std_Logic
END dff;                               - End of entity declaration
```

Keyword such as entiry, port, ht, out and are reserved words and comments are preceded by a (-)

The operation of the D Filip-Flop is specified by the architecture declaration. The code uses conditional statements to emulate the flip flop.

```
ARCHITECTURE behavioral OF dff IS  - Architechure of the Entity begn
Output Process                     - Process named output declared
Wait Until (clk EVENT AND clk = 'I')  - Continues when clock rises q<d:
qbar <= NOT d;                     - input is also generated
END PROCESS OUTPUT                 - Process terminates and waits
END behavioral :                   - End of Archirecture
```

Note that architecnture name "behavioral" is arbitary but the entity name "dff must match the entity declaration (5)

This project will cover mainly on VHDL, the writer will explain it furhermore in next chapter

2.2. PicoBlaze

Xilinx's PicoBlaze is a microcontroller designe to be embedded into Sparatan-3 gamily of FPGAs. One of its benefits is the ability to modify components of microcontroller.

2.3. Implementation of PicoBlaze onto the Spartan-3E FPGA

Unlike dedicated microcontrollers supplied by companies like ATMEL where programs written in assembly language are compiled and uploaded onto the chip, assembly programs for Pieoblaze are uploaded in HDI. and are included with the instructions for the design of the microcontroller This is achieved by using any text editor to write an assembly Program with a psm file extension and then compiling the Program using KCPSm3.exe (supplied by Xilinx). This in turn creates a H'DL file to lie included with the microcontroller kepsm3. vhd file for compiling and uploading onto the FPGA PROM (programmable read-onlyd-only memory). The PROM is *a* separate on-board memor^{y} component which is used to pro$_{g}$ram the FPGA at start *up.*

Features of PicoBlazc

- load-store architecture
- contains 16 internal 8 bit registers
- program size a maximum of 1024 instructions
- multiple processors can be connected for increased performance
- stack can support 31 levels of subroutine nesting
- only 57 instructions

2.4 Spartan-3E Development Board

There are several components in board that writer used in EF11889-1, they are LCD screen, pushbuttons and serial communications between the board and a computer.

There arc some advantages in using the Spartan Chip-,

1. Integrated external system component s C,

   Power regulator and in line filters arc no longer needed in PCB

2. Large IP Library

I will reduce the time spent on product development

The PCB includes: LCD screen, 21 pushbutton and swich inputs and 8 LED outputs. We can see front diagraw below:

Gambar

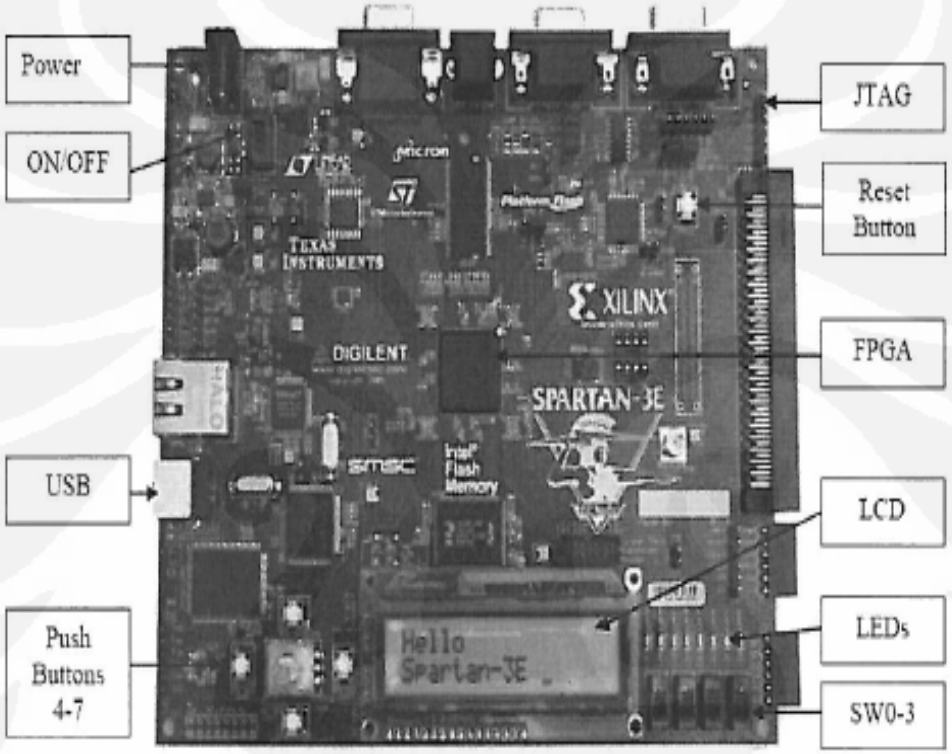


Figure 3 Spartan-3E Development Board

The chip communicates with the LCD screen using 7 pains as shown below

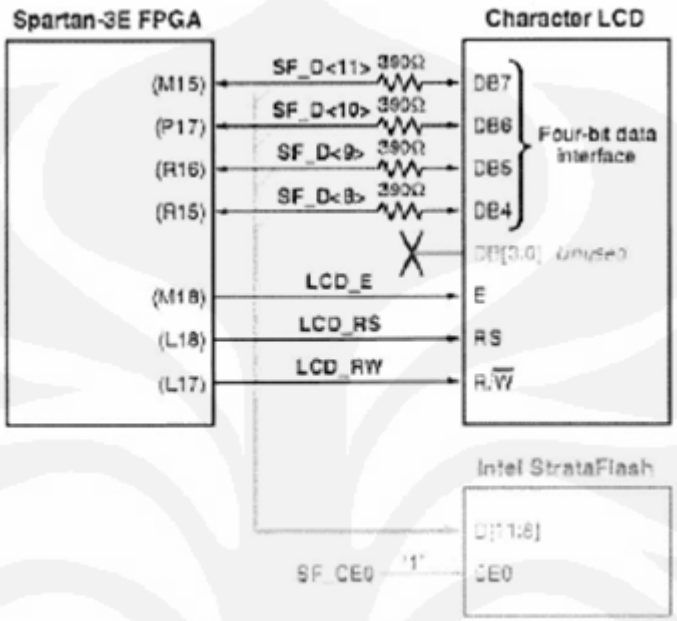


Figure 4 Connection of LCD to Spartan-3E [14, p43]

Fiqure 4 Connection of LCD to Spratan-3E (14,p43)

The top 4 pains are he data pains the bottom 3 pins are interface pins; these must switch on and off in specific patterns while the display is configured.

The FPGA connects to the LEDs using the following pins:

Name	Pin
LED0	F12
LED1	E12
LED2	E11
LED3	F11
LED4	C11
LED5	D11
LED6	E9
LED7	F9

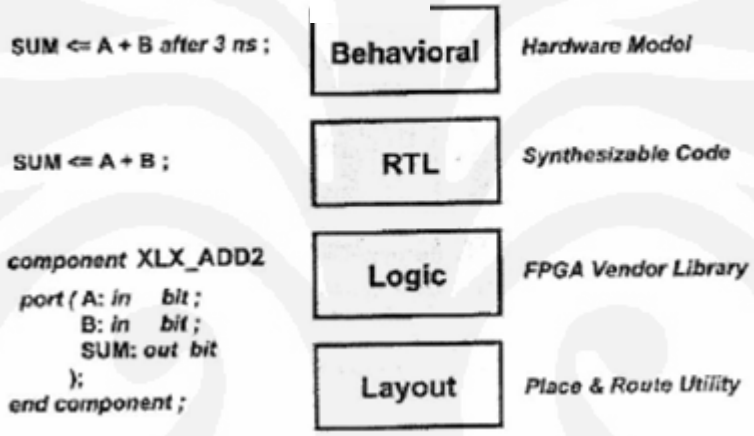
CHAPTER III
VHDL

3.1 Introduction to VHDL

VHDL is an abbreviation of VHSIC Hardware Description Language. Furthermore, VHSIC stands for Very High-Speed Integrated Circuit. VHDL is the toot for hardware modeling. It had been used for simulation or to synthesis.

There are many benefits using VHDL The first **is:** executable specification. A VHDL specification can be executed in order to achieve, a high level of confidence in its correctness before commencing design, and may simulate one to two orders of magnitude faster than a gate level description. The second is: Technolo^{g}y and tool independence (though FPGJA features may be unexploited). VHDL permits technology independent design through support for top down design and logic synthesis. - To move a design to a new technology you need not start from scratch or reverse-engineer a specification - instead yew go back up the design tree to a behavioural VHDL description, then implement that in the new technology knowing that the correct functionality will be preserved. The third one is: portable design data (Protect investment). VHDL descriptions of hardware design and test benches are portable between design tools, and portable between design centres and project. partners. You can safely invest in VHDL modelling effort and training, knowing that you will not be tied in to a single tool vendor, but will be free to preserve your investment across tools and platforms. Also, the design automation tool vendors are themselves making a large investment in VHDL, ensuring a continuing supply of state-of-the-art VHDL tools. The fourth one is: Simulate early and fast (Manage

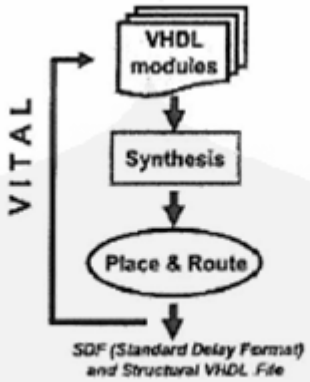
complexity). Behavioural simulation can reduce design time by allowing design problems **to be** detected early on, avoiding the aced to rework designs at gate level. Behavioural simulation also permits design optimization by exploring alternative architectures, resulting in better designs. With all those benefits, therefore, VHDL activies have received aplenty of attentions.



HDL Description Levels

The are 4 rules regarding partitioning a design for VHDL design entry:

- Must be consistent with RTL coding structure
- Minimize the number of clocks per block
- Maintain critical signals within a block
- Make a quick, relatively simple test bench for each submodule



Recommended Design Verification Stages

There are there levels of design verification in VHDL:

- Behavioral/RTL simulation: Execute RTL source code and the testbench
- Post-synthesis VHDL simulation: Execution VHD file and the testbench
- VHDL timing simulation: Execute post-layout structural VHD and SDF file and the testbech

3.1.0. VHDL Language Concepts

VHDL is composed of design units, they are :

- Entity
- Architecture
- Package
- Package Body
- Configuration

3.1.2.1 Entity

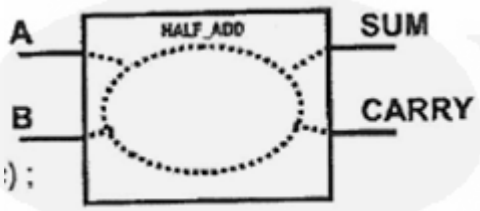
An entity describes the external interface to the hardware module

Entitiy Half_ADD is

Port (A, B : in std_logic;

SUM, CARRY : out std_logic) ;

End entity HALF_ADD



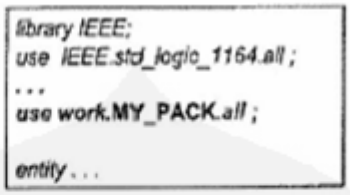
3.1.2.2 Architecture

```
architecture RTL of HALF_ADD is
begin
   SUM <= A xor B ;
   CARRY <= A and B ;
end architecture RTL ;
```

Note: VHDL-87 does not allow restatement of the keyword *architecture* at the end of the declaration

An architecture describes the internal operation of is associated (primary) entity

3.1.2.3. Package



3

A package stores the data that will be used repeatedly throughout a design, project or organization

CHAPTER IV
SIMULATIONS AND RESULT

4.1. Theoritical Analysis

The purpose of this project was to create a simple calculator project. The calculator project consisted of 4 parts: ALU, MEM. CTRL_FS M. COMP with MY_CALC_MODULE as a package far all of them. All of that parts was divided for two persons, Faristama Aryasa explained about ALU and MEM and Mohammad Sandi Suryagemilang explained about CTRl_,FSM, COMP and MY_CALC_MODULE..

4.1.1. ALU

ALU: Arithmetic Logic Unit is one of the blocks that consist in the project. The basic knowledge for operators is compulsory. The Logic has their calculation on

its block. The block performs calculation such as arithmetic and logical operation. It is one of the essential blocks. Fundamental operations were calculated in this block. Depending on the, input which in this case. is the OP_CODE the ALU takes A and B two bits operand, as well as carry input C_IN, and it will do 32 operation with 2 types, one with the carry in and the other is not. The calculation will he helpful for the whole system once it has boon done.

While the testbench is on' the appendix

The 32 operations are:

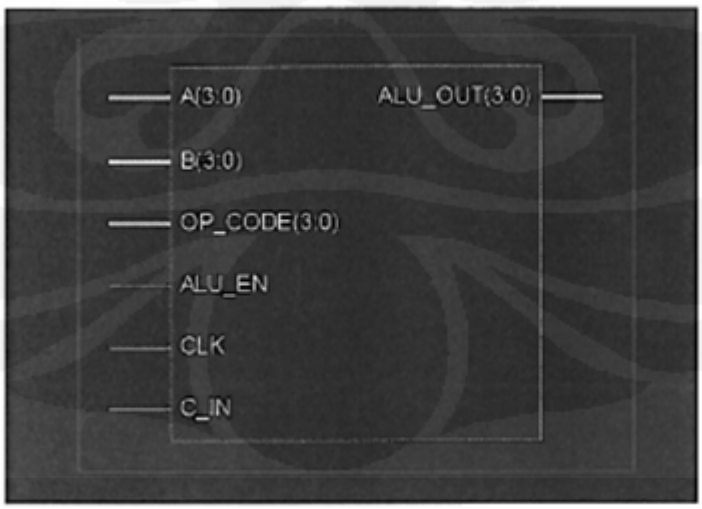
Operation	OP_CODE & C_in	Carry in	Result
txa	00000	0	A
xta	00001	1	A
inc	00010	0	A + 1
inc	00011	1	A + 1
add	00100	0	A + B
add	00101	1	A + B
addc	00110	0	A + B + C_IN
addc	00111	1	A + B + C_IN
sub	01000	0	A + not B + 1
sub	01001	1	A + not B + 1
comp	01010	0	not A
comp	01011	1	not A
neg	01100	0	not A + 1
neg	01101	1	not A + 1
dec	01110	0	A – 1
dec	0111	1	A – 1
txb	10000	0	B
txb	10001	1	B
and	10010	0	A and B
and	10011	1	A and B
or	10100	0	A or B
or	10101	1	A or B
xor	10110	0	A xor B
xor	10110	1	A xor B
sl	11000	0	A (2 downto 0) & ‘0’ Shift left
sl	11001	1	A (2 downto 0) & ‘0’ Shift left
sr	11010	0	0’ shift right & A (3 downto 0)

sr	11011	1	0’ shift right & A (3 downto 0)
Par	11100	0	000 & ((A(3) xor A(2)) xor (A(1) Xor A(0)))
par	11101		1 & ((A(3) xor A(2)) xor (A(1) Xor A(0)))
zero	11110	0	0
zero	11111	1	0

All operation is a basic operation although view syntax of VHDL language might occur. A wide description of the operation will be revealed.

As known that every single block have to run into each test benches according to their block.

The test benches apply on the simulation only, they are also important to design a process. Since the simulation only, they are also important to design should be checked first before it works correctly. This testing can also be carried out for the RTL schematic.



The RTL Schematic shows that A, B, C and OP_CQDE (4 bits) are the inputs along clock and the carrier, while the Output will be in form of 4 bits, It later on will useful for the circuit analyzing.

The test bench already existed. However some addition to fulfill the requirement of all operations had to be completed, Exact coding will be discussed on the next part. See Appendix for ALU test bench.

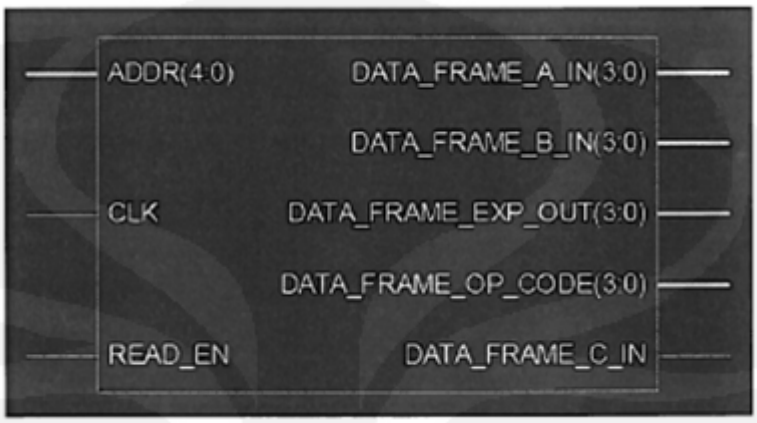
4.1.2 MEM

MEM: Memory management system is a block that implements the MY_ROM which is the memory temp that every sin[g]le input should have the same output, as in the Memory management, The MEM would create output with exact match for what the aiding says ¡n the MY_ROM, The Addresses were counted by the NICK Where when certain address with certain OP_CODE and time were delivered. Output generates a memory management to construct the A, B, C_in, the expected output to march the OP_CODE.

This MEM block also store 32 memory arrays,. This 32 memory corresponds to the address that we compute from 0 to 31. As in ALU, this block is also important for the whole system. At the end the control finite state machine control al the process for combining these, blocks.

As shown above, the expected results for certain OP_CODE has been determined. These will shows how the output going to be. The test bench again will

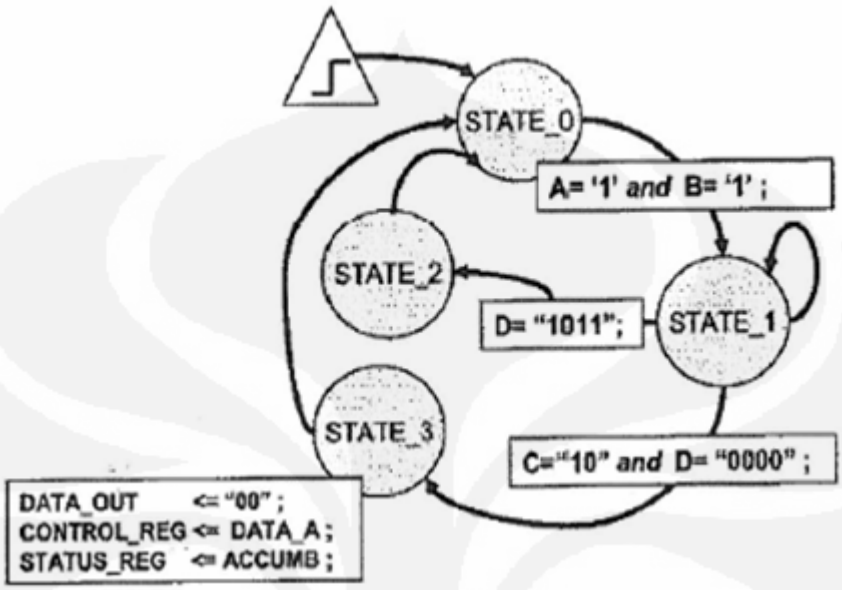
do the testing simulation for its block, The RTL schematic model could be delivered, The A and B are default outputs which are determined.



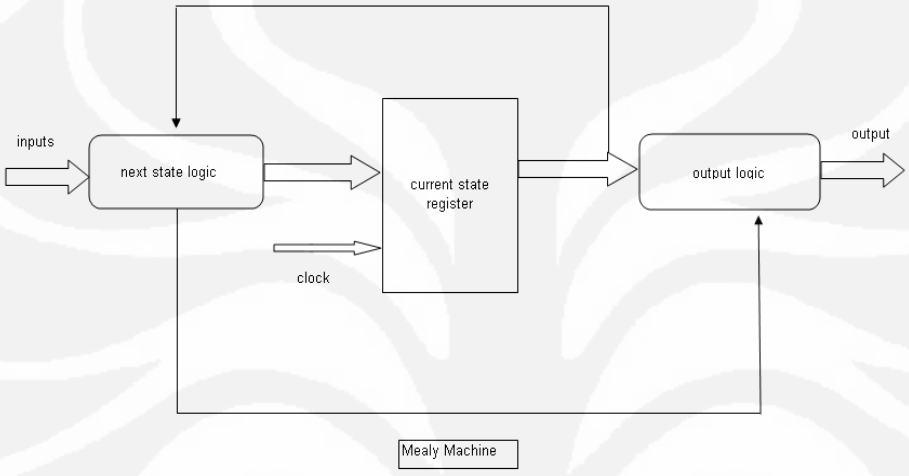
The behavioral simulation will deliver the data frame output which contains all the output, from that data frame the checking of an appropriate memory management should be done, 17he discussion for checking of the data frame would be presented furthermore.

4.13 CTRL_FSM

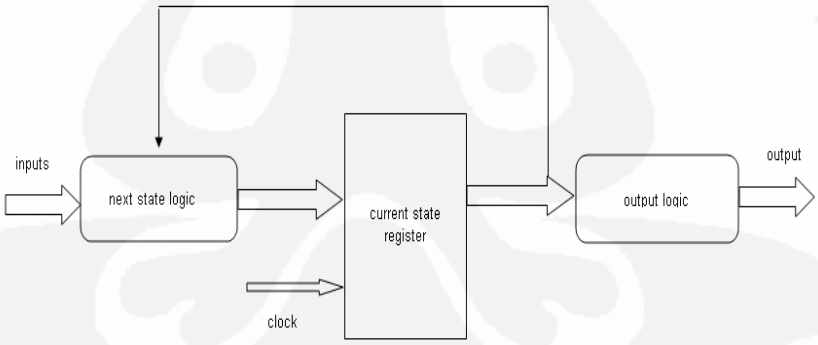
In this part, the writer implemented CTRI_.FSM and a test bench for it. FSM stands for Finite State Machine.



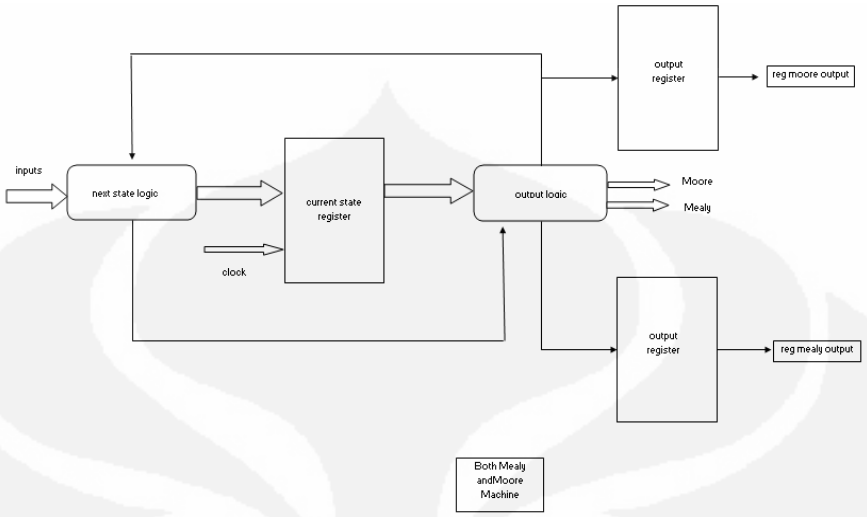
The first thing when coding an FSM in VHDL is whether it has Mealy outputs, Moore, outputs or both. The difference of them is in how their outputs are computed. Outputs of Mealy FSM's arc a function of both its inputs and its present state. Because of this, the outputs of a mealy FSM can change as soon as any of is inputs change. However, its state still cannot change until a triggering clock edge. A Moore FSM's outputs are a function of only its present state**.** Since a Moore FSM's outputs are a function of only is present state, they Can change only at a triggering clock edge. Generally, any sequential function can be implemented by either Mealy FSM or a Moore FSM, bu also it can be a combination of both, having sonic outputs that depend on the present state and present inputs (Mealy outputs) and other outputs depend only the present state (Moore outputs)



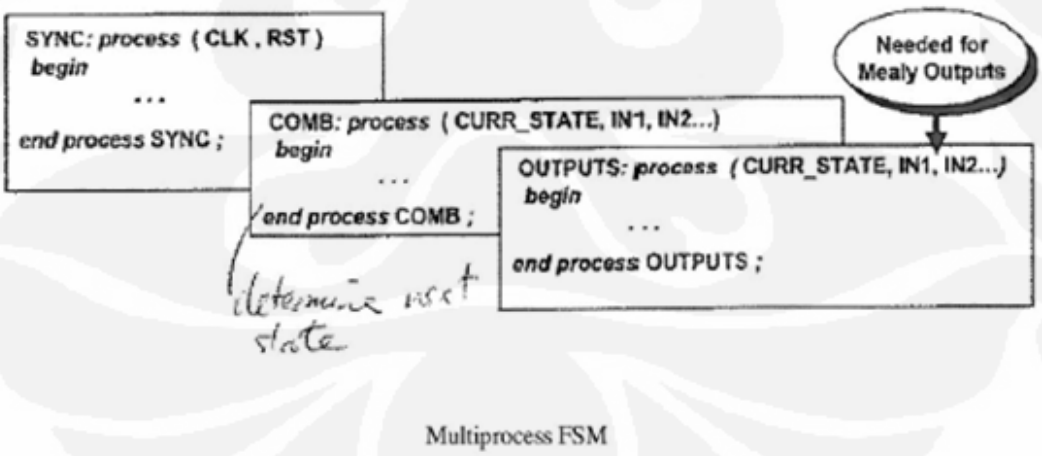
Mealy Machine



Moore Machine



Multiprocess FSM is very recommended. Multiprocess means separate processes to model the sequential, combinational an output logic is used. It requires less coding, because big coding is seperated into small picces. Furthermore, because the coding is separated, it also can be more readable and intuitive. The last benefit is it can allow more compiler optimization by no forcing registered outputs.



Multiprocess FSM

Multiprocess FSM

VHDL process block described both synchronous and combinational logic portions of FSM.

```
Architecture RTL of FSM is begin….
SYNC process (CLK, RST) begin
If (RST = ‘1’) then
Current_State<=Init;
Elsif rising_edge (CLK) then
Current_State<=Next_State end if;
End process Sync;
```

Current state logic described by syanchronous process

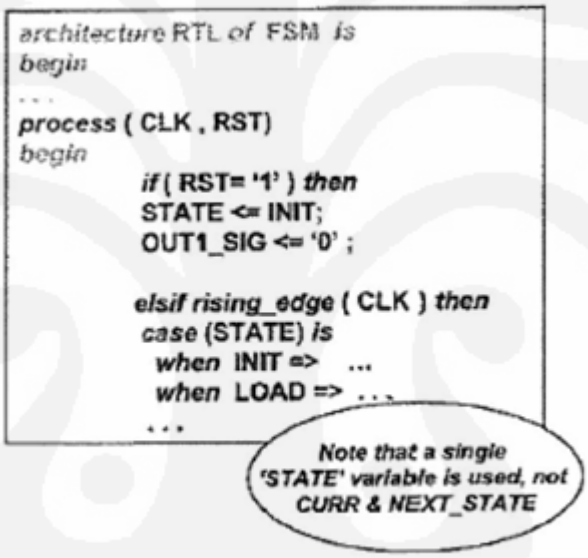
```
Archilocture Rtl of FSM is begin ….
Case (Current_State) is when Init =>
Out1_Sig <= ‘0’
Status_Reg <= ‘10’;


When Load =>
Out1_SIG <= ‘1’;
Status_Reg <= “0”;
Moore Qutputs in next state logic

Architecture RTL of FSM is begin ….
Process (Current_State, IN1, In2, In3) begin
Case (Current_State) is when Init =>
If (IN1 and In2) = ‘1’ then
Out1_Sig <= ‘0’
```

Status_Reg <= “0”;

When Load = >

Mealy Outputs Next State Logic (Inputs are inserted in process)

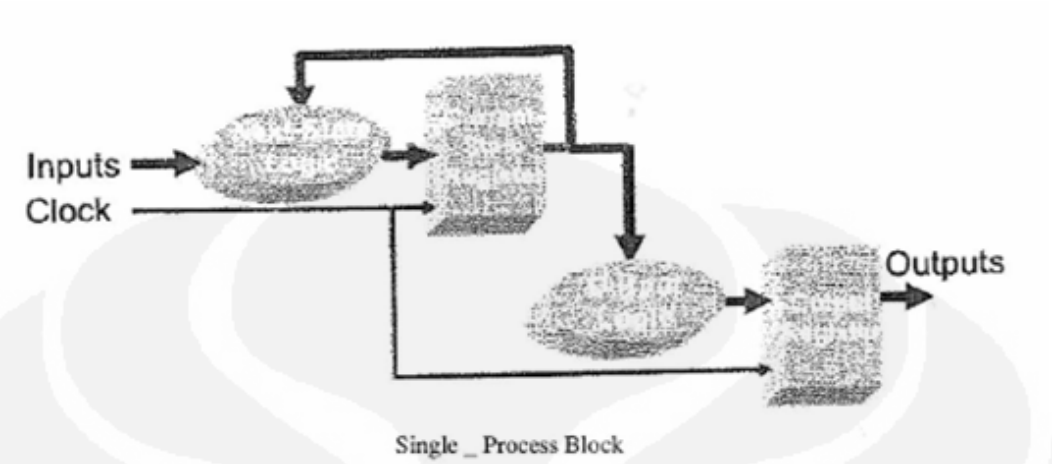
FSM can also coded by single clocked process, but it’s no recommended because it consuming more resources that resulted by all outputs are registered. The second reason is because state events usually occurred one clock cycle after entering the state.



Single Process FSM

Single Process FSM

Furthermore, in single clocked process, there is another registers on the output that increase amount of next state combinational logic necessary to drive them.



Single_Process Block

4.1.4. Comp

A comp or comparator a block that takes two numbers as input in binary form and determines whether one input greater than, less than or equal to the other input.

The operation of a single bit digital comparator can be expressed as a truth table:

Inputs	**Outputs**			
A	**B**	**A<B**	**A=B**	**A>B**
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

That means the binary variable (A=B) is 1 only if all pairs of digits of the two numbers are same, furthermore, (A>B) and (A<B) are output binary variables, which are equal to 1 when A>B or A<B.

4.1.5 My Calc Module

My Calc Module had a purpose to Complete the calculator project. In this section, he author combined the submodules that were created in the previous lab exercises to complete the SIMPLE_CALC module.

4.2. Experimental Procedure or Design Procedure

4.2.1 ALU

The procedure to do the ALU is first to create the coding with VHD file. There ar The test bench for the ALU Rick- and the calculation to do the calculation. The V11D called ALU.VHD, the test bench called ALU_TB.VHD and the calculator is MY_CALC_PACKAGE.VHD. After the entire file complete, a test bench could be simulated!

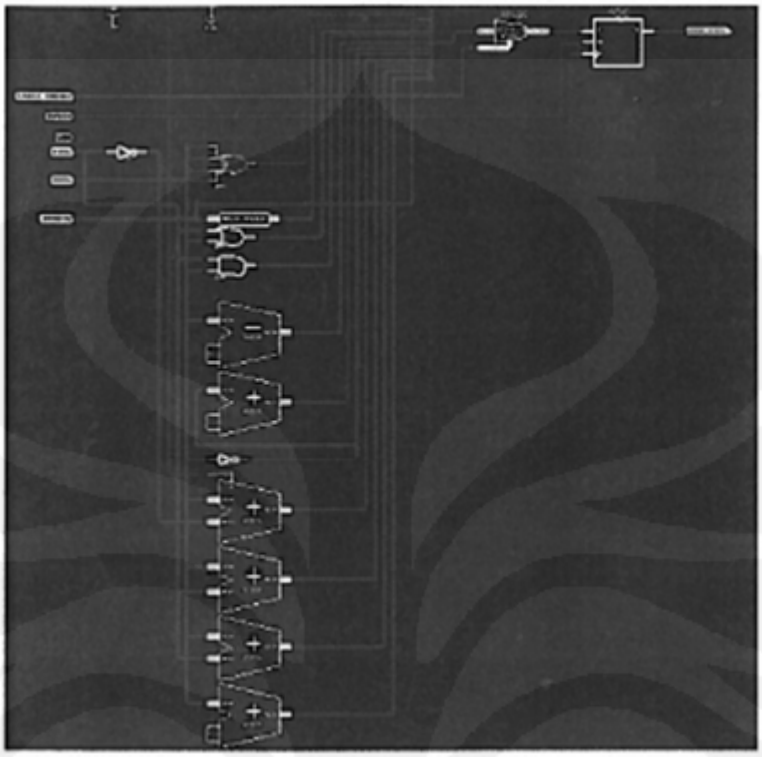
Since the, entire file had been generated, the, project can be build with:

- Click the Xilinx ISE 92. Software then clicks the project navigator.
- At the opened window, click the file tab -> new project until -> new project wizard window appeared
- Type, ALU_AB (means that it has been tested as the project name -> choose HDL as the top level source type -> then click next.
- Keep clicking next until a new window called add existing code has appear -> click add source and chums the three files of the VHD files, The calculation, ATJJ, and the test bench for the ALU -> Click next again.
- Now the ALU block project has been created. The window will appear like this.

Gambar

Clicking the source code, either ALU_TB or the unit under test will result as a coding for the test bench and also the VHD. To see through the RTL schemantic, simply put the source for synthesis/implementation and expand the synthesize XST then click the view RTL schematic on the processes window.

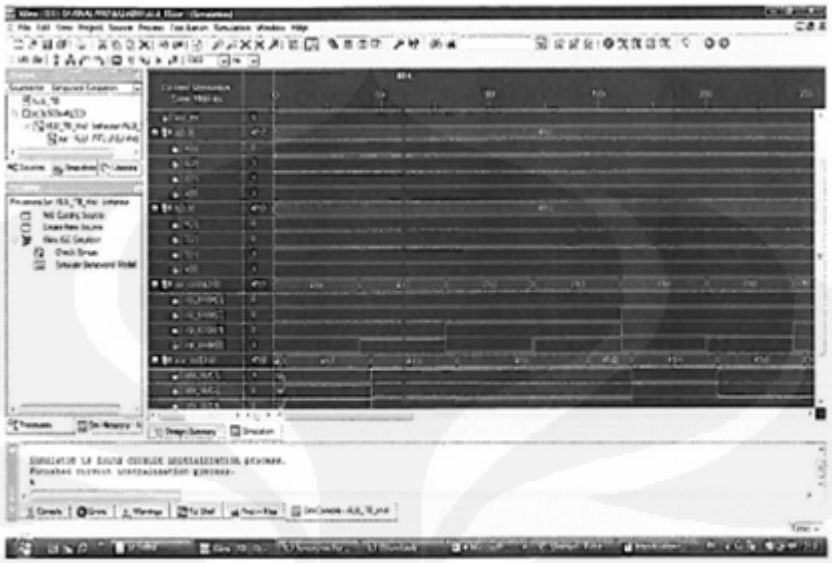
The RTL schematic will appear as:



Clicking the RTL schematic gets the schematic to the inside as if like as a circuit with gates

Discover the simulation for behavioral model

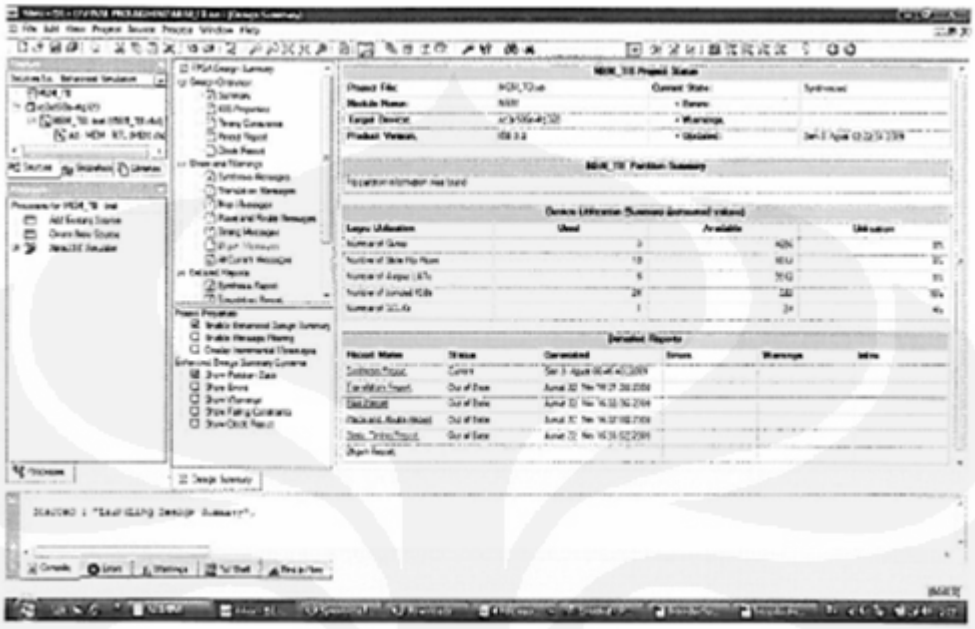
- Change the source for behavioral again -> expand the Xilinx ISE simulator -> click the simulate behavioral model.
- Wait for a while to check the syntax of the system -> a new window called simulation should appear. The outputs are expandable to unsure manual checking for the operation of ALU (Arithmetic Logic Unit).



42.2 M E M

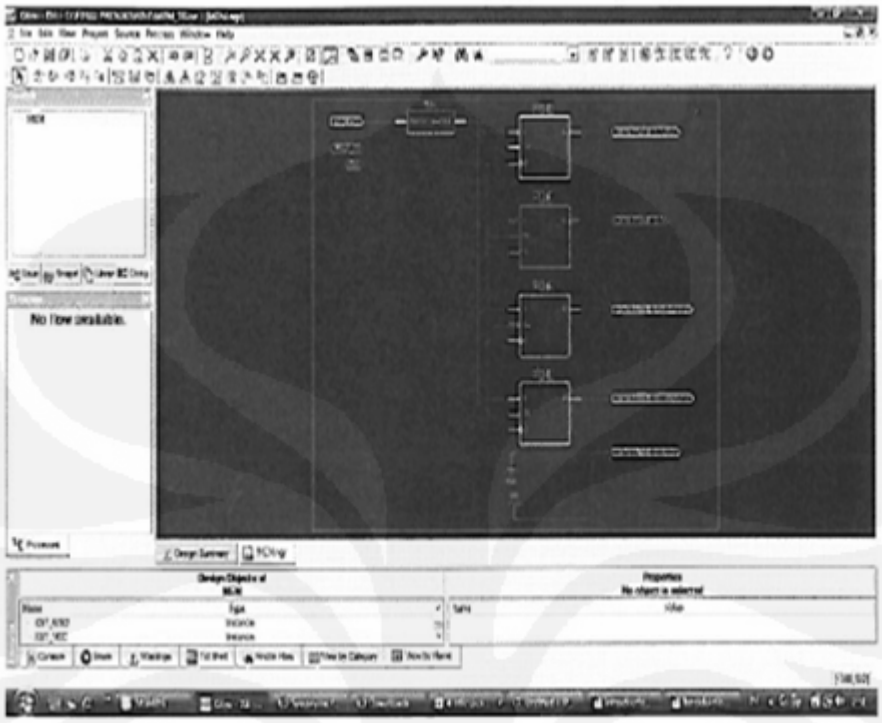
The MEM block project also needs 3 kinds of files. The files are MEM. VHD, MEM_TB.VHD, and also the MY_CALC_PACKAGE.VHD. Same as before, the steps to make the project are:

- Click the Xilinx ISE 92, Software then clicks' the project navigator.
- At the opened window, click the file tab -> new project until -> a new project wizard window appeared,
- Type MEM_TB (means that it has been tested) as the project name -> choose HDL. as the top level source type -> then click next.
- Keep clicking next until a new window called add existing code has appear click add source and choose the three files of the VHD files, MEM.VHD, MEM_TB. and, and also the MY_CALC_PACKAGE.VHD -> Click next again. Remember that the 3 files are there,
- Now the ALU block project has been created. The window will appear like this



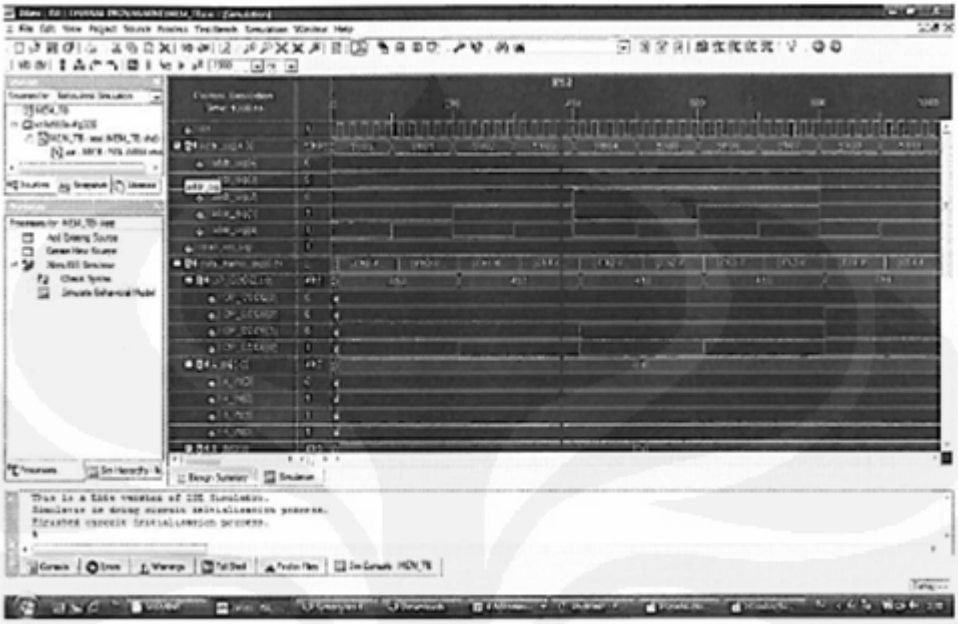
The RTL Schematic for, can be maximized and shown through these steps:

- On the Top left window-> change the source for synthesis/implementation.
- Expand the synthesize-XST on the processes window -> then click on View RTL schematic -> click again on schematic to reveal the register transfer level.
- The window should appear as this.



Furthermore, same procedure to show the simulation behavior: and in this case is the data frame of the MEM memory management:

- Change the source for behavioral again -> expand the Xilinx ISE simulator -> click the simulate behavioral model.
- Wait for a while to check the syntax of the system - > a new window called simulation should appear. The output are expandable to help manual checking for the operation of NEM (Arithmetic Logic Unit).
- The data frame should appear as below:

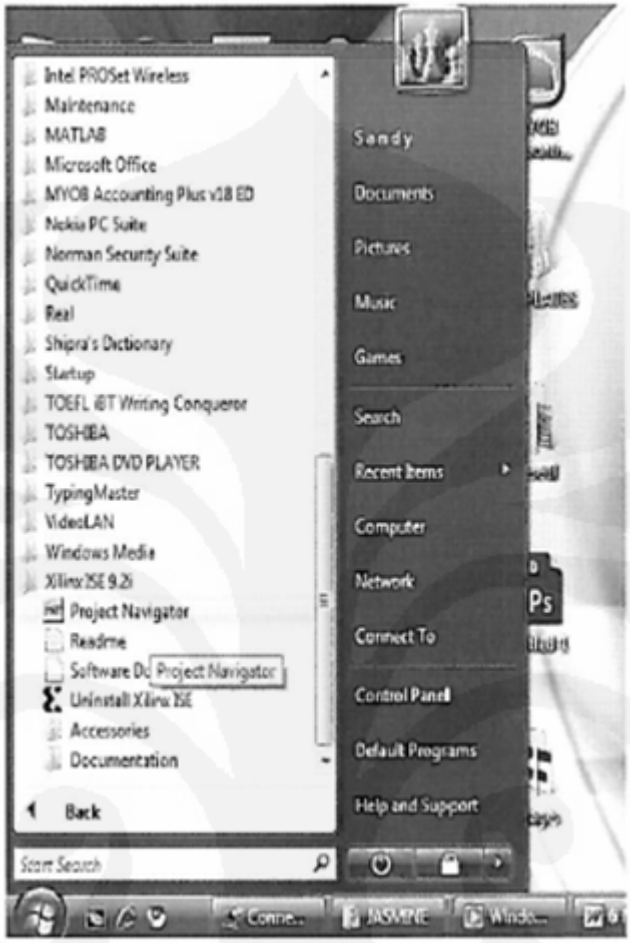


The data frame can be expandable so the bits will show. At the end, simulation will be used for checking the error for the block and also the Memory management of the. MEM. The data frame output should match with the OP_CODE which we stored all the value in MY_ROM array.

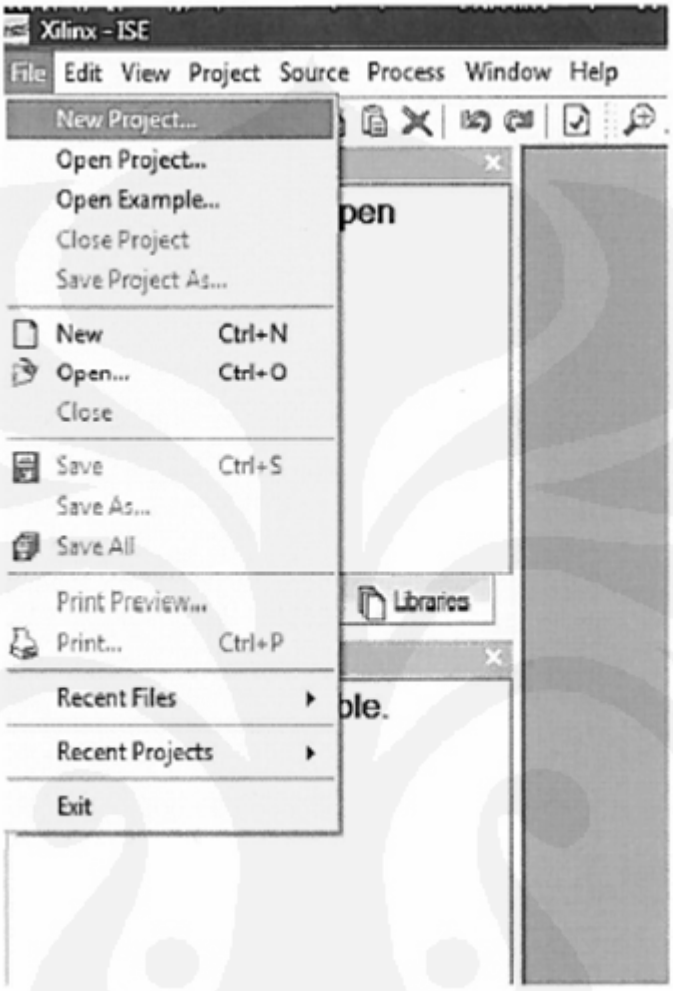
4.23. CTRL_FSM

In this project the writer implemented CTRL_FSNM and made a test bench for it, 3 files was put together in a project called “CTRL_FSM_TB”. They are MY_ALC_PACKAGE. vhd, CNML_FSM_TB.vhd and C'I'R_FSM,vhd. Simulation code wrote in CTRL_FSM_TB.VHD.

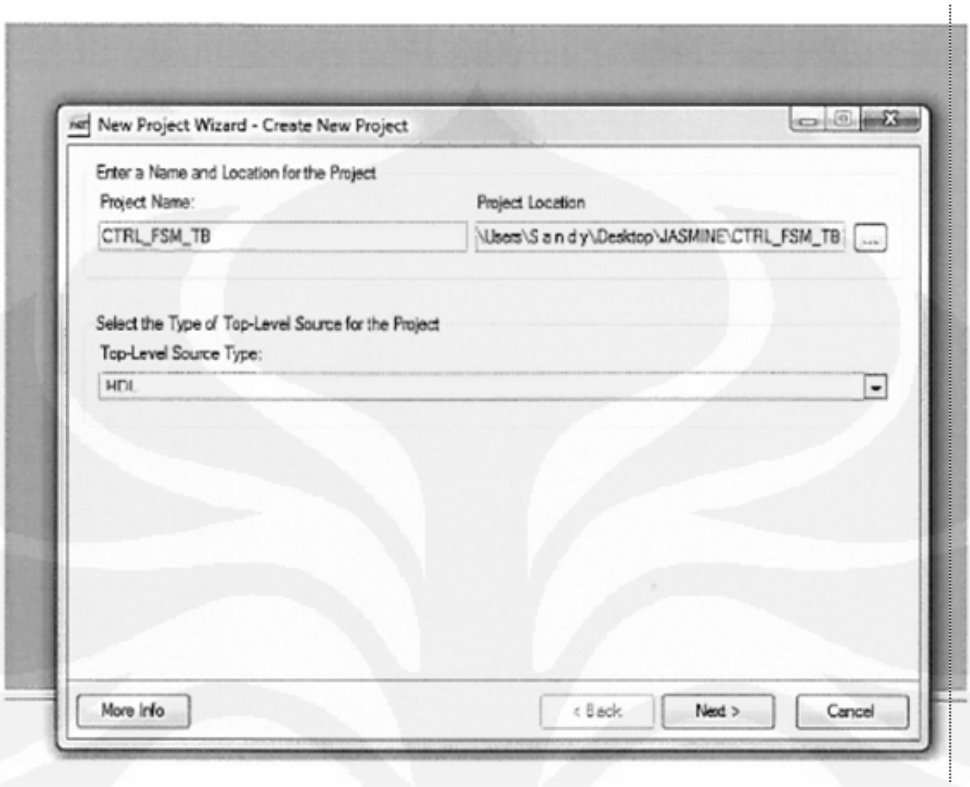
Firstly, the author made a project called "CTRL_FSM_TB”. Initially, The author went to start menu and then clicked Xilinx ISE 9.2 I, and then clicked Project Navigator.



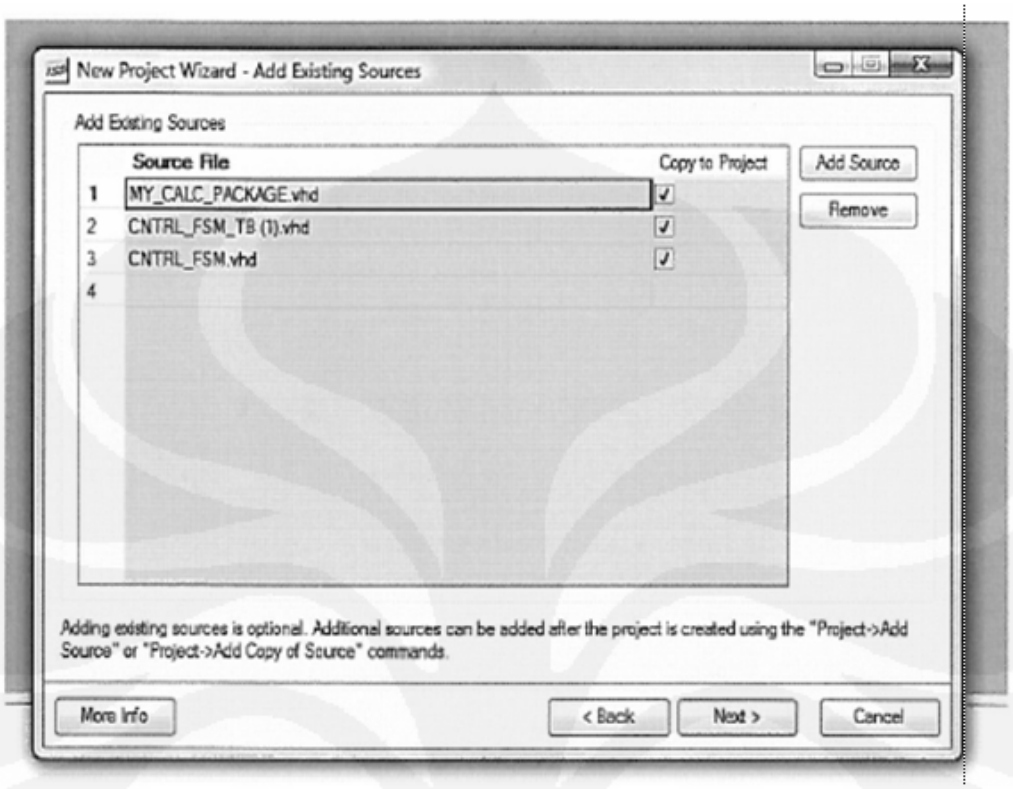
After that, the author clicked file and then clicked new project



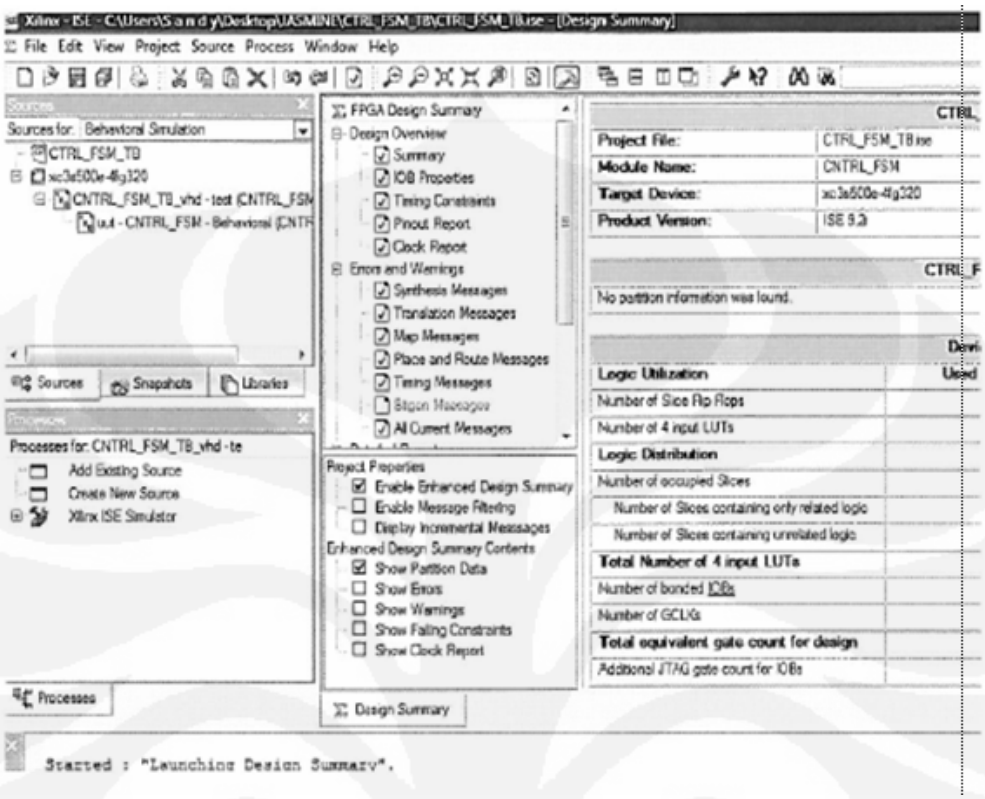
And then, the writer typed project name which was CTRL_FSM_TB



Furthermore, the writer was kept clicking was kept clicking next until he found window of new project wizard-Add existing sources. When he got there, he inserted those there files and then he clicked next and finish.



The screen became like this below:



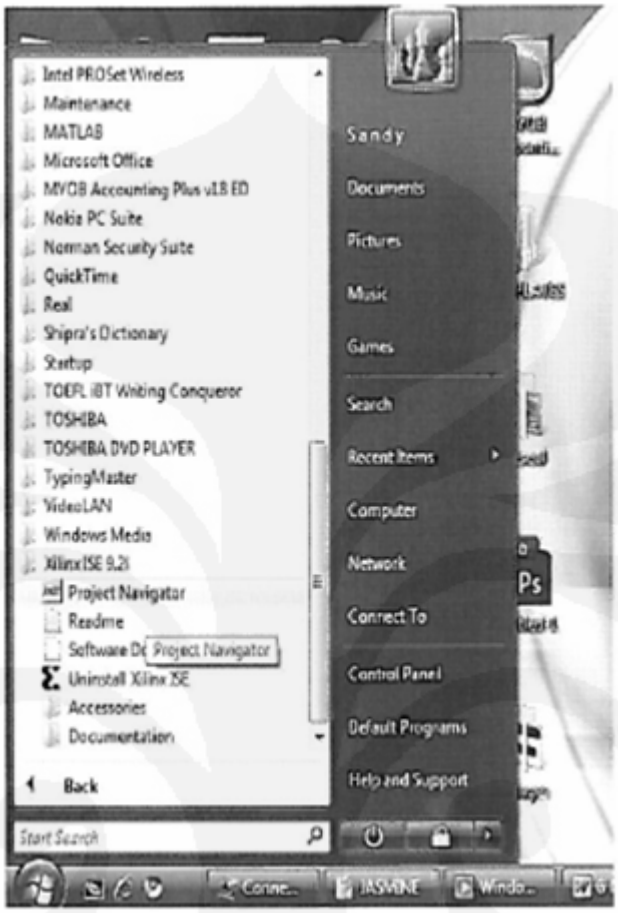
After that, the writer expanded Xilink ISE Simula tor and double-clicked simulate model

4.2.4 COMP

Moreover in this part, the writer created a file called comp_tb.vhd and wrote code for the test bench for it using a file called COMP.vhd.

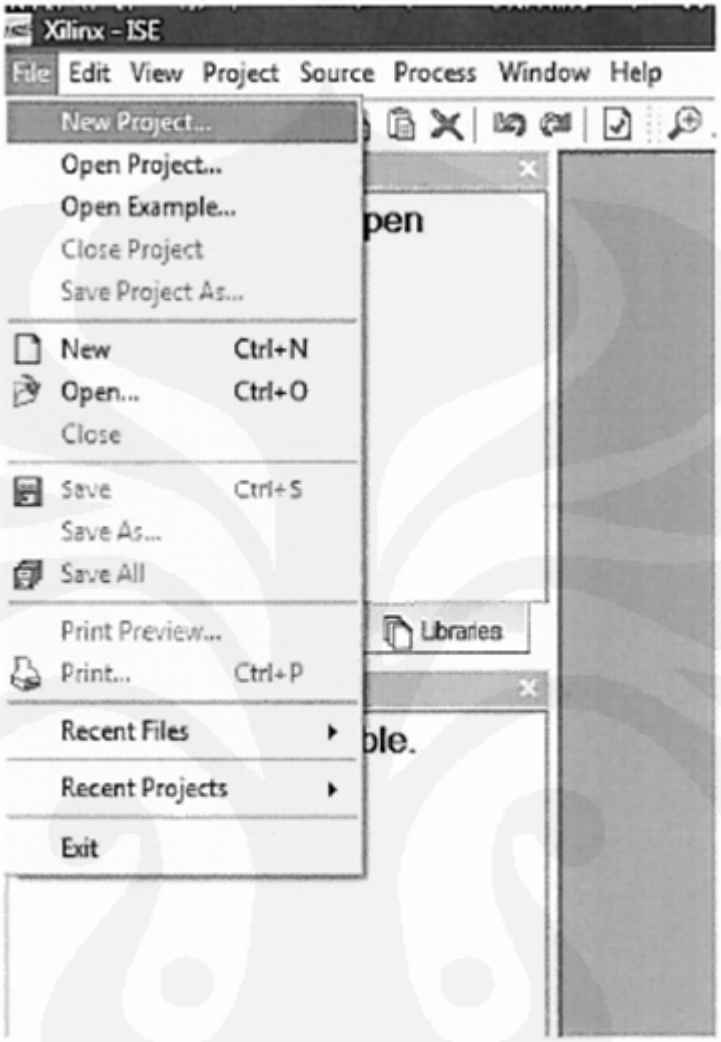
Furthermore, the writer created a project called comp_tb and add comp-tb.vhd and COMP.vhd into comp_tb. And then, writer ran the test bench simulation.

Firstly, the author made a project called “comp_tb”. Initially, the author went to start menu and then clicked Xilinx ISE 9.2.I, and then clicked project navigator.



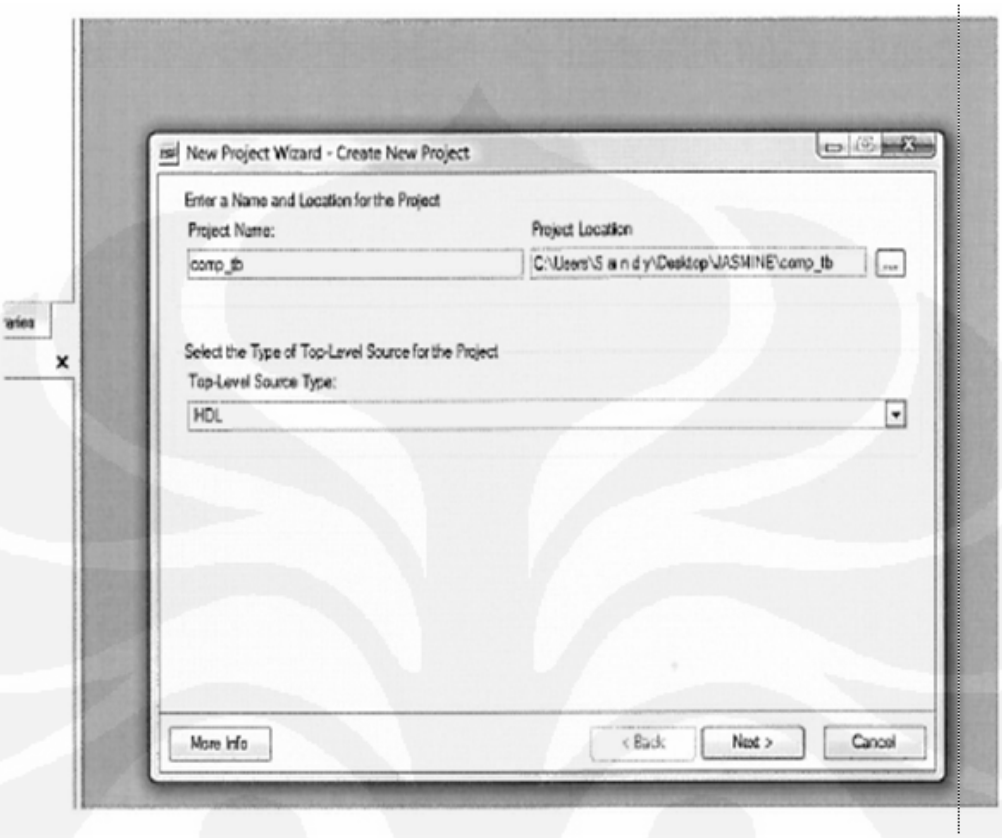
Gambar

After that, the author clicked file and then clicked new project



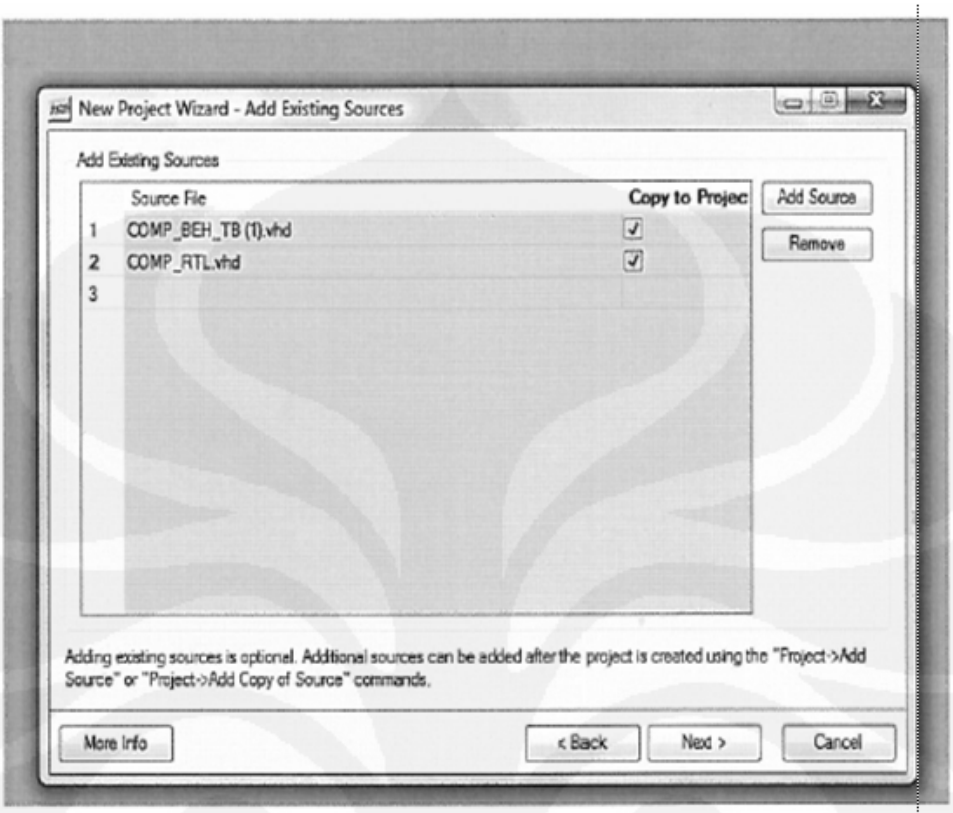
Gambar

And then, the writer typed project name which was comp_tb



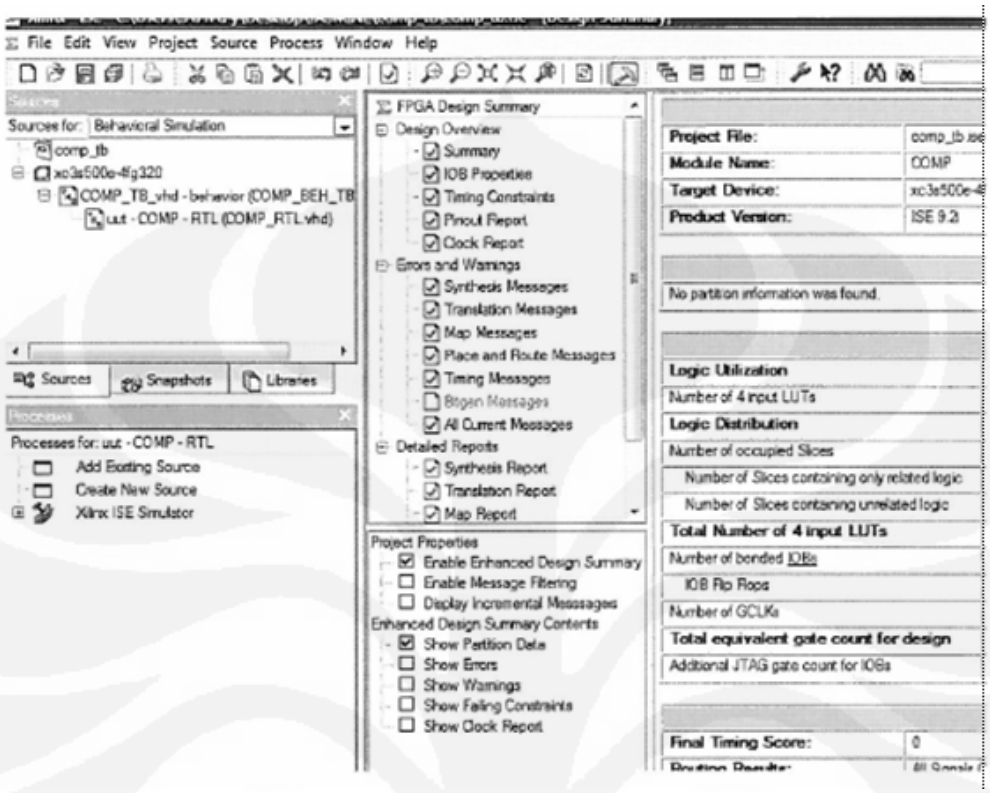
Gambar

Furthermore, the writer was kept clicking next until he found window of new project wizard add existing sources. When he got there, he inserted those two files and then he clicked next and finish.

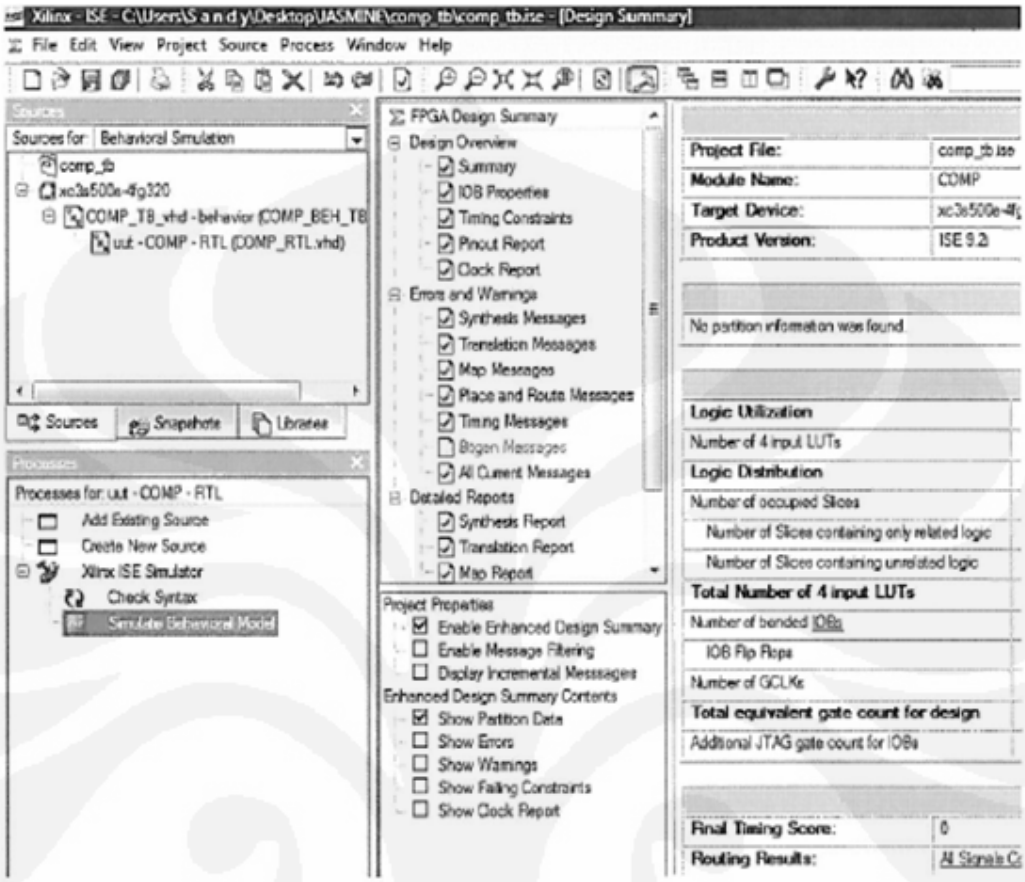


Gambar

The screen became like this below:



After that, the writer expanded Xilink ISE simulator and doble clicked simulate behavioral model

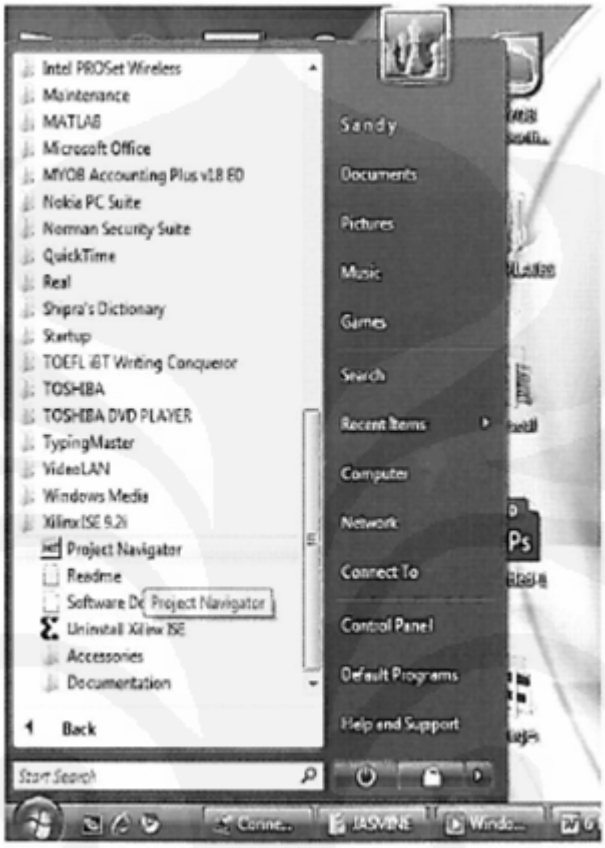


4.2.5 CALC

For he next task, the writer made a project with my_cale.vhd as well as his completed files ALU.vhd, CNTRL_FSM.vhd, MEM.vhd, COMP.vbd and MY_CALC_PACKAGE.vhd.

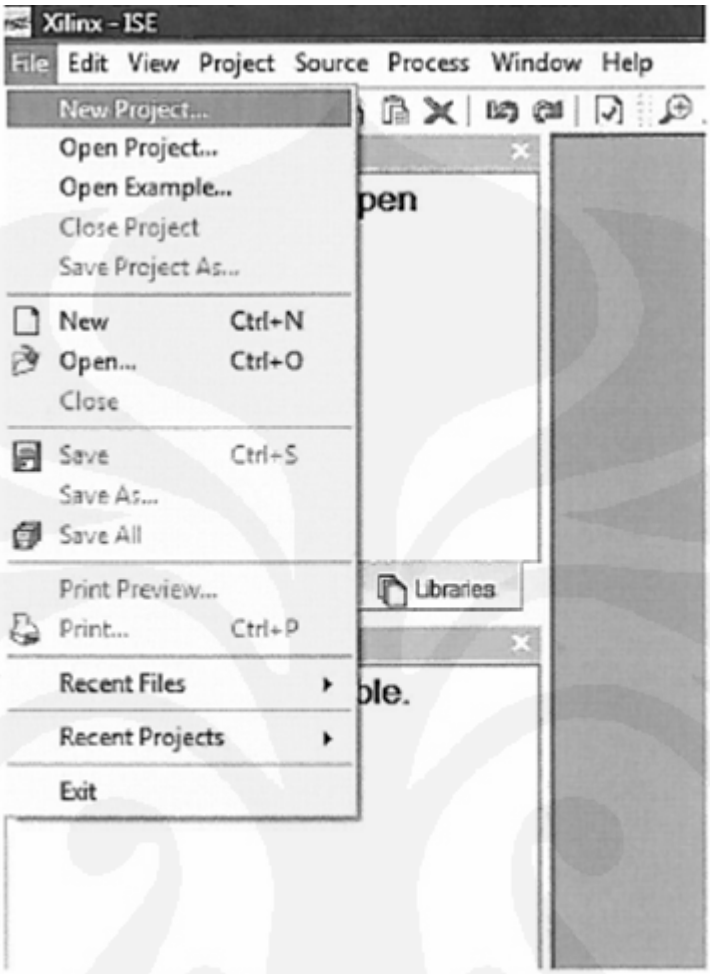
After that, the writer made a yest bench MY_CALC_TB.vhd for this project.

Firstly, the author made a project called “MY_CALC_TB”. Initially, the author went to start menu and then clicked Xilinx ISE 9.2.I and then clicked project navigator.



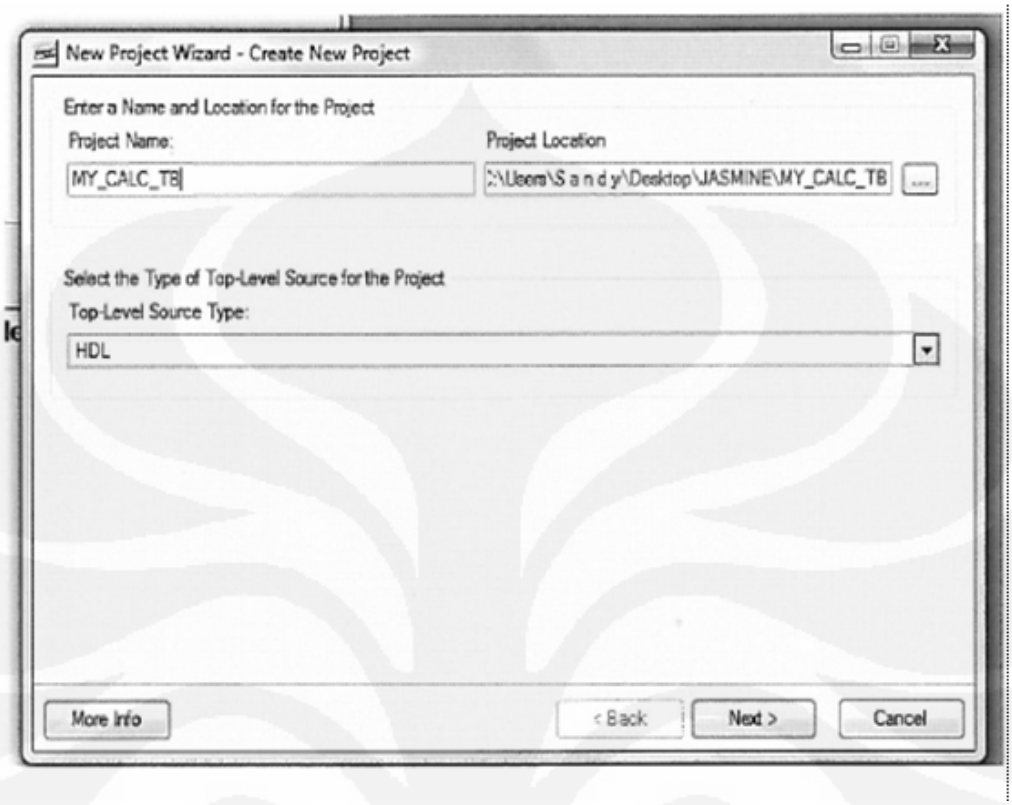
Gambar

After that, the author clicked file and then clicked new project

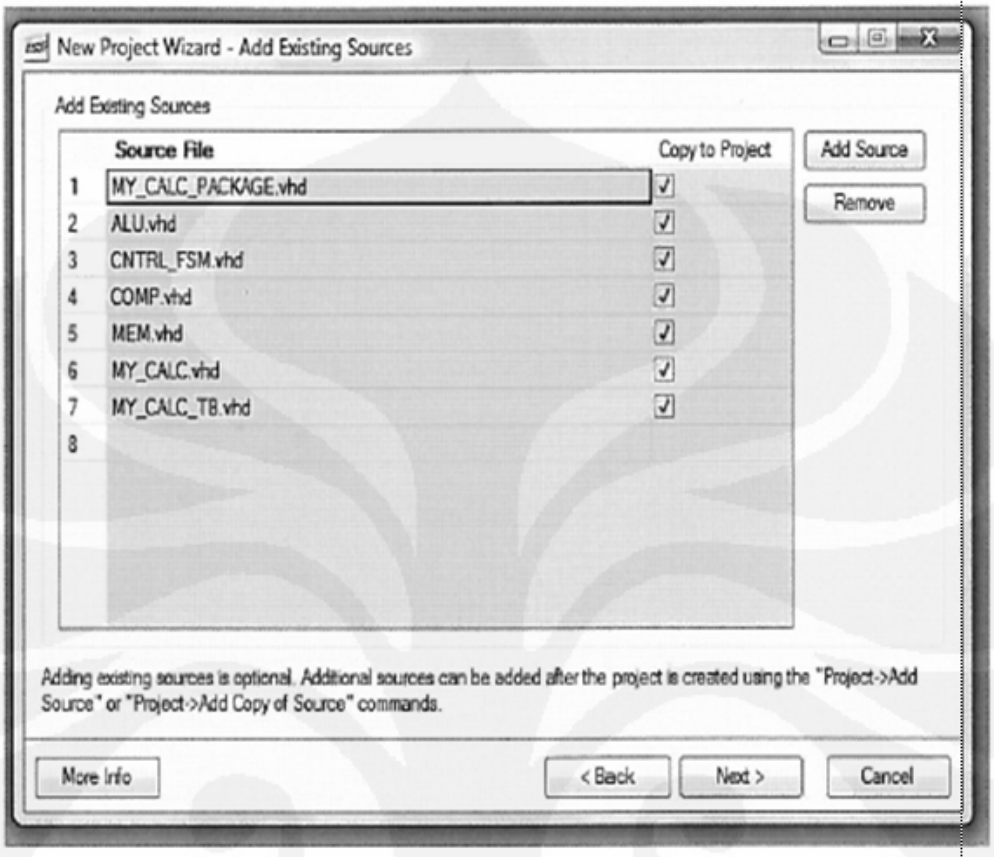


Gambar

And then, the writer typed project name which was MY_CALC_TB

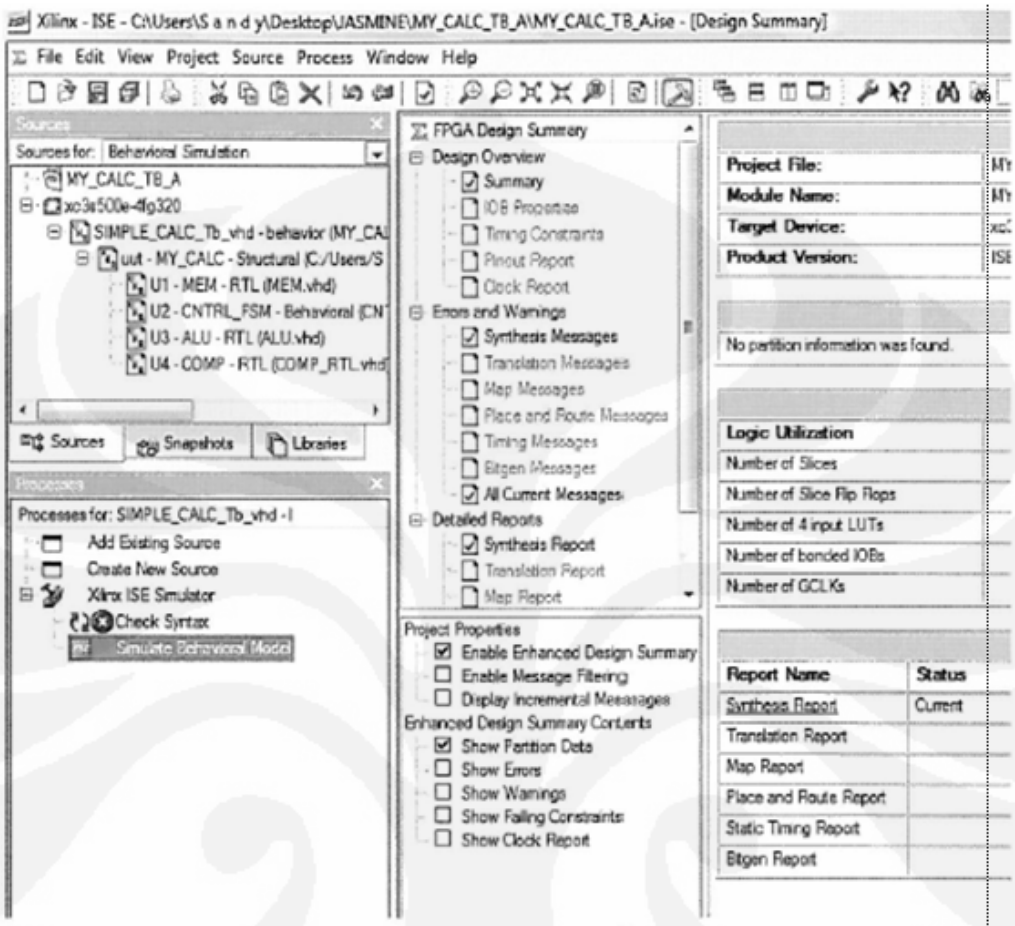


Furthermore, the writer was kept clicking next unit he found window of new project wizard add existing sources. When he got there, the inserted those seven and then he clicked next and finish.



The screen became like this below:


File Edit View Project Source Process Window Help
Sources for: Behavioral Simulation
MY_CALC_TB_A
xc3s500e-4fg320
SIMPLE_CALC_Tb_vhd - behavior (MY_CAL
uut - MY_CALC - Structural (C:/Users/S
U1 - MEM - RTL (MEM.vhd)
U2 - CNTRL_FSM - Behavioral (CN
U3 - ALU - RTL (ALU.vhd)
U4 - COMP - RTL (COMP_RTL.vhd)
Sources
Snapshots
Libraries
Processes for: SIMPLE_CALC_Tb_vhd - I
Add Existing Source
Create New Source
Xilinx ISE Simulator
Processes
FPGA Design Summary
Design Overview
Summary
IOB Properties
Timing Constraints
Pinout Report
Clock Report
Errors and Warnings
Synthesis Messages
Translation Messages
Map Messages
Place and Route Messages
Timing Messages
Bitgen Messages
All Current Messages
Detailed Reports
Synthesis Report
Translation Report
Map Report
Project Properties
Enable Enhanced Design Summary
Enable Message Filtering
Display Incremental Messages
Enhanced Design Summary Contents
Show Partition Data
Show Errors
Show Warnings
Show Failing Constraints
Show Clock Report
Design Summary
Project File:
Module Name:
Target Device:
Product Version:
No partition information
Logic Utilization
Number of Slices
Number of Slice Flip Fl
Number of 4 input LUT
Number of bonded IOE
Number of GCLKs
Report Name
Synthesis Report
Translation Report
Map Report
Place and Route Repo
Static Timing Report
Bitgen Report



4.3 Results or Final Design

4.3.1 ALU

Checking the ALU test bench should refer on the operation of the operators. There are 16 operators to be used, in the case of checking from the test bench project.

The ALU performs a very well test bench_ 31 operators including with carry done, success parity. In this case the parity with carrier sets to 0. The same coding for the parity without carrier cannot be applied on this operator. Some other coding should be implemented. In the test bench, the A and B have a default value of '0111' and '0011'

In this report, 10 Results can be observed white the rest results could be taken from the CD that includes the, report.

Operator:

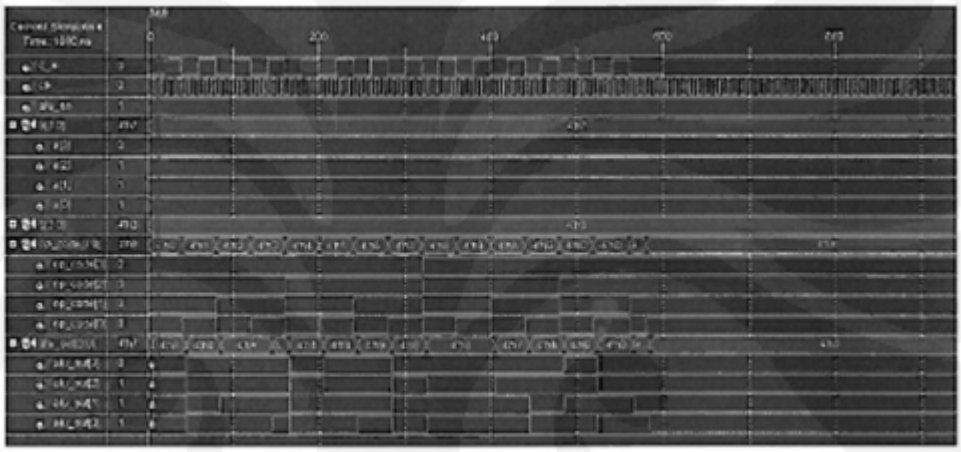
1. Txa: The ALIT. output is simply the value of A default. Both same with or without carrier. Certain OP_CODE & C IN, in example of 00000,

   OP_CODE & C_IN: 00000

   A : 0111

   B : 0011

   ALL_OUT: 0 111



2. Txb: The ALU output is simply the v aloe of B default. Both same with or without earlier. Certain OR_CODE & C_IN, in example of 00001,

   OP_CODE &C_M- 00001

   A. 0111

   B. 0011

   ALU_OUT: 0011

3. OR. The ALU output is A 'OR' B

   OP_CODE & C_IN: 10101

   ALU OUT : 0111

4. AND The A117 output is A 'AND' B

   OP_CODE & C_TN: 0010

ALU_OUT: 0011

5. COMPLEMENT: The ALU output is NOTA
   OP_CODE: 01100
   ALU_OUT:1000

6. XOR: The ALL output is A.'XOR"B
   OP_CODE: 101 10
   ALU_OUT: 0100

7. SL : The ALL, output is shift left A
   OP_CODE: 11000
   ALU_OUT: 1110

8. DEC: The ALU output is A-1
   OP_CODE: 01110
   ALU_OUT: 0110

9. INC: The INC output is A-1
   OP_CPDE : 00010
   ALU_OUT : 1000

10. ADD: The. ADD output is A + 1
    OP_CODE : 00100
    ALU_OUT : 1010

4.3.2 MEM

On the IMLIM block the checkin[g] is more focusing on the memory management. The storage MY_ROM give the clue of how the results going to be

depend on the input. In this case again. a test bench shows the MEM data frame output, which will carry on for the whole project of calculation package with 4 blocks.

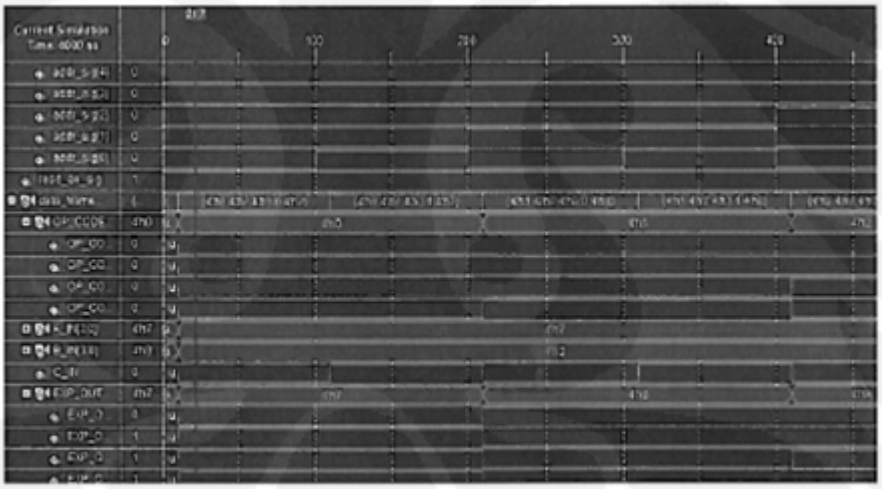
The array MY_ROM has 32 addresses and with A (0111) and B (0011) for the default test bench. The 10 results are being presented while the other result can he checked through the CD within.

1. Txa : for txa address ‘o’ (00000)

   Data frame :

   OP_CODE : 0000

   EXP_OUT : 0111



2. Txb : for txb address ‘16’ (10000)

   Data frame

   OP_CODE : 1000

   EXP_OUT : 0011

3. OR : for OR address ‘20’ (10100)

   Data frame

OP_CODE : 10010

EXP_OUT : 00111

4. AND : for AND address ‘18’ (10010)

Data frame :

OP_CODE : 1001

EXP_OUT : 0011

5. COMP : for COMPLEMENT address ‘10’ (01010)

Data frame :

OP_CODE : 0101

EXP_OUT : 1000

6. XOR : for XOR address ‘22’ (10110)

Data frame :

OP_CODE : 1011

EXP_OUT : 0100

7. SL : for Shift left address ‘24’ (11000)

Data frame :

OP_CODE : 1100

EXP_OUT : 1110

8. DEC : for decrement address ‘14’ (01110)

Data frame :

OP_CODE : 0111

EXP_OUT : 0110

9. INC : for Increment address ‘2’ (00010)

Data frame :

OP_CODE : 0001

EXP_OUT : 1000

10. ADD : for Adder address ‘4’ (00100)

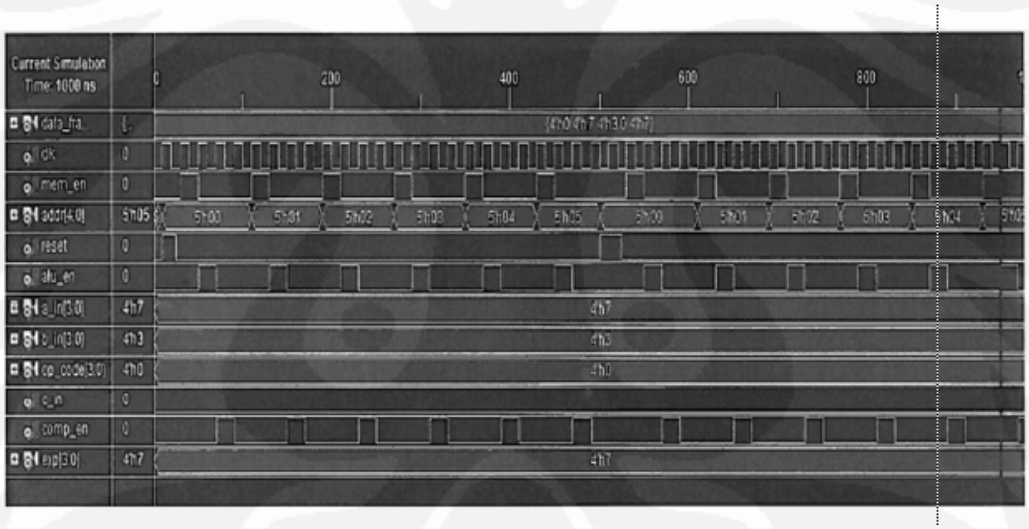
Data frame :

OP_CODE : 0010

EXP_OUT : 1010

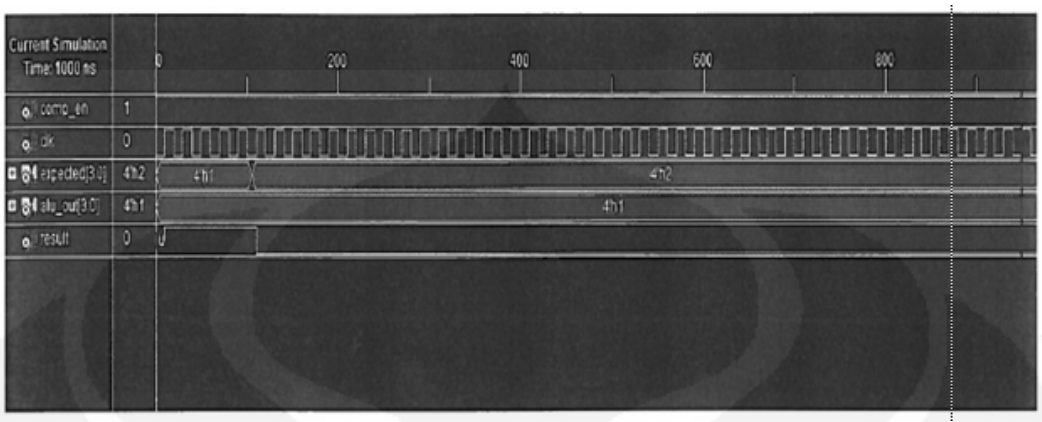
4.3.3 Result of CTRL_FSM

This is the result of Simulation of CTRL_FSM_TB.vhd



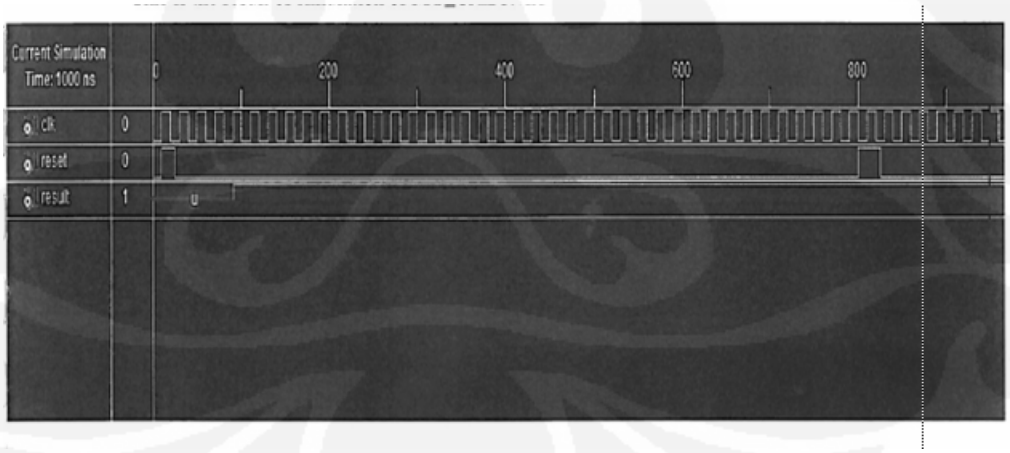
4.3.4 Result of COMP

This is the result of simulation of comp_tb.vhd



4.3.5 Result of CALC

This is the result of simulation of MY_CALC.vhd



CHAPTER V
DISCUSSION.

5.1 ALU

The results show that ALU works well enough aside from I of the operation was being, failed. Affect of that particular operation is not very significant. The whole project should still he working without that operation, The, top 2 levels of HDL description level have reached a successful project. The behavioral and RTL goes well.

The abstraction levels with view details and simulation were achieved by doing the test bench before the implementation. In order to get a correct project for implementation, the pre synthesize state should be done. In this ease Synthesizable

code and hardware modeling were completed.

As for the results for the test bench, for

- Txa : The output was 0111 -> which is correctly operated by ALU since 0111 is the default value of A.
- Txb - The output was 0011 -> which is correctly operated by ALU since 0011 is the default value of &
- OR . The output was 0111 -> which is correctly operated by ALU since. OR become true if either one of the operands are true.

  A (0111) OR B (0011) => ALU_OUT (0111)

  AND: output was 0011 -> which is correctly operated by ALU since AND become false if either one of the operands are true.

  A (0111) AND B (0011) => ALU_OUT (0111)
- COMP. The output was IWO -> which is correctly operated by ALU since it is A complement, Complement A (0111) ALU_0UT (1110)
- XOR : The output was 01W -> which is correctly operated by ALU since XOR become true if true if one and only one of the operands are true.

  A (0111) XOR 11 (0011) => ALU_OUT (01 00)
- SL: The output was 1110 -> which is correctly operated by ALU since SL shift the bits to the lfet, with an extra bill "0'.

  Shift left A (0111) => ALU._OUT (1110)
- DEC : The output vas 0110 ->which is correctly operated by ALU since the, A was decresed by one

  A (0111) – I => ALU_O 54))
- INC **:** The output was 1000 -> which is correctly operatd by ALU since the A was increased by one T,

  A. (0111) 11.) + => ALU_OLTT (1000)
- ADD :The output was 1010 -> which is correctly operated by ALU by using A + B was

5.2 MEM

In the test bench, the MEM black works perfectly. There was not any mistake in the syntax or even on the output. Tile MY_ ROM arrays have matched the specification, wwhenever the specific input come the output data frame consist of A B C _ in and. the OP–CODE.

Txa : Data frame match with, MY_ROM storage.

Txb : Data frame match with MY_ROM storage.

OR : Data frame match with MY_ROM storage.

AND : Data frame match with MY_ROM storage.

COMP : Data frame match with MY ROM storage.

XOR : Data frame match with MY_ROM storage.

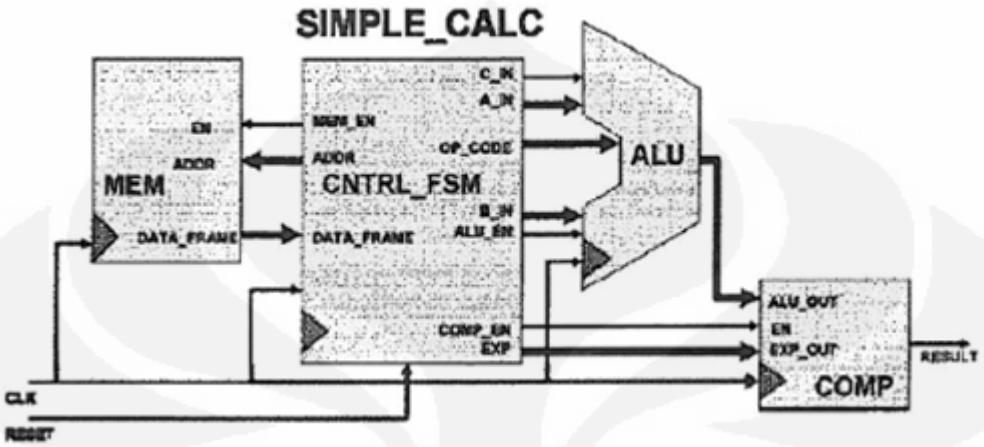
SL : Data frame match with MY_ROM storage,

DEC. : Data frame match with MY_ROM storage.

INC : Data frame match with MY_ROM storage.

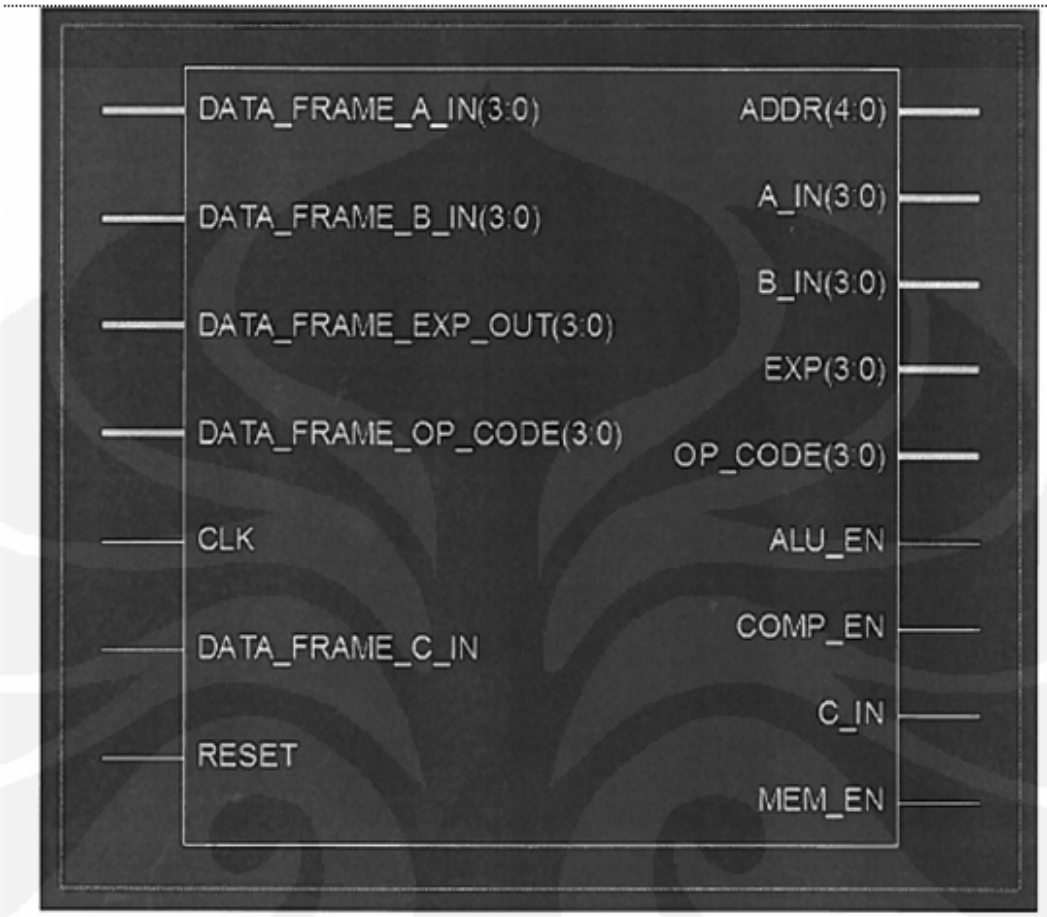
ADD : Data frame match with MY_ROM storage.

5.3. CTRL_FSM



Schematic for a Simple Calculator Circuit

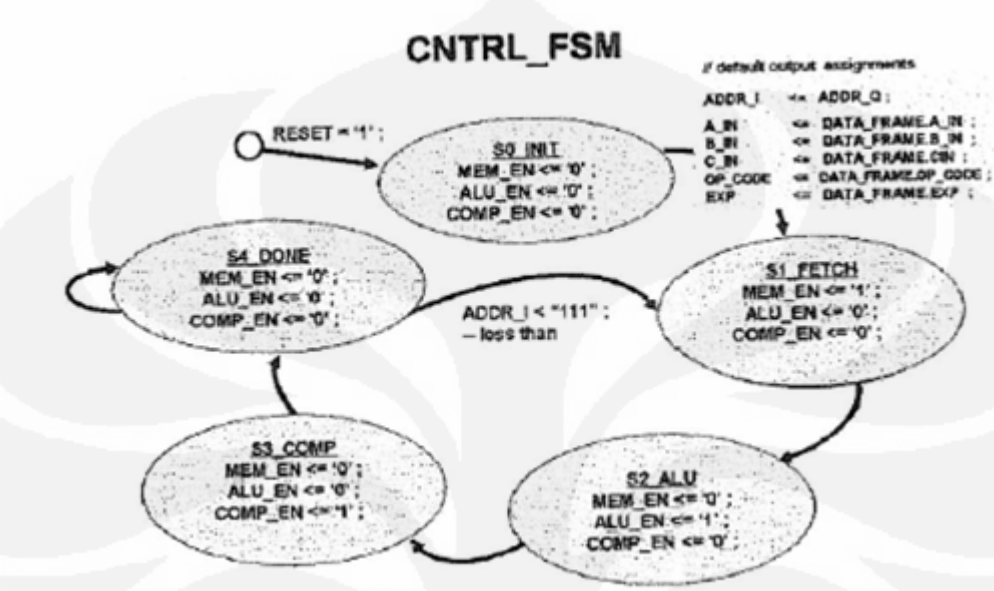
Schematic for a simple calculator circuit

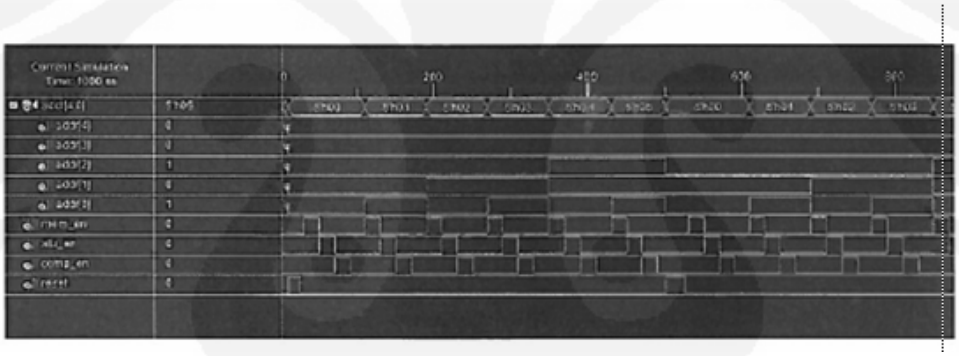


Schematic of CTRL_TB.vhd

It can tic seen from 2 diagrams above, here are 7 inputs 0 CTRL_FSM block, they are: DATA_FRAME_A_IN, DATA_FRAME_B_IN, DATA_FRAME_C_IN DATA-FRAME_OP_CODE, DATA _FRAME_EXP_OUT, CLK and reset (3:0) means 3 downs, to zero as we can see in CTRL_FSM_TB.vhd, or we can say it is 4 bits code. All DATA_FRAMEs is output of MEM block. Clock is useful as a timing signal. It controls '.when all the, memory elements can change state, Reset is synonymus wilt clear, Any synchronous inputs must be kept unasserted for normal synchronous operation., Furthermore, them are 9 outputs going out from C:I'RL FSM block. They are: A-IN, B_IN, C_IN, OP_CODE, ALU_EN, COMP_EN, EXP, MEM_EN, ADDR, A_IN, B_IN, C_IN, OP_CODE and ALU_EN goto ALU BLOCK, COMP_EN and EXP go to MEM block and CTRL_FSM block involving

their inputs and outputs. It means, CTRL_FSM output become MEM inputs and vice versa.





The two diagram above is the state of CTRL_FSM. It shows transition from one state to another.

1. SO Init

   This is the first step of this process. From two diagrams above , we can see that Initial state happened when MEM_FSM = 0, ALU_EN = 0 and COMP_EN = 0 in other words, we can say that init is intial state when it is reset

2. S1 FETCH

   In this state, CTRL_FSM get the next instruction from the MEM , From two diagrams above, we can see that this state is happened when MAM_EN = 1

ALU_EN = 0 and COMP_EN = 0

3. S2 ALU

In this state, CTRL_FSM block execute the instruction on the ALU. From two diagrams above we can see that Initial state is happened when, MEM_EN = 0, ALU_EN = 1 and COMP_EN =0

4. S3 COMP

In this state, CTRL_FSM compares the result from. the ALU with the expected result . From two diagrams above, we can see that Initial state is happened when MEM_EM = 0, ALU_EN = 0 and COMP_EM = 1

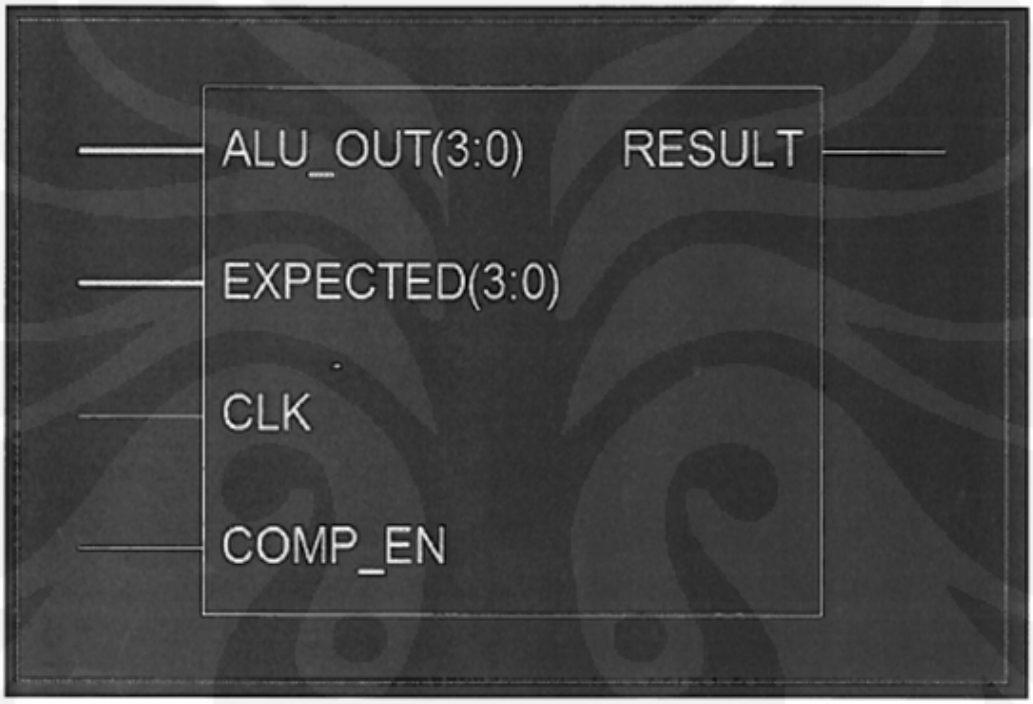
5. S4 DONE

In this state, CTRL_FSM checks if we have gone through all instructions, if not, return to FETCH otherwise, we are finished, stay in DONE., FEOM two diagrams above, we can see that Initial state is happened when. MEM_EM = 0, ALU_EN = 0 and COMP_EN = 0

Those sequences above was repeated 31 times, because it should happened 32 times (from ADDR 00000 to ADDR. 11111)

The red box with a letter inside ^{in} on the graph above means the Stale was still undefined. Furthermore. we can see from CTRL_FSM_TB.vhd that RESET <= '0%1' after 25ns,T after 500ns,0 after 525ns, it was proved from the diagram above

5.4. COMP



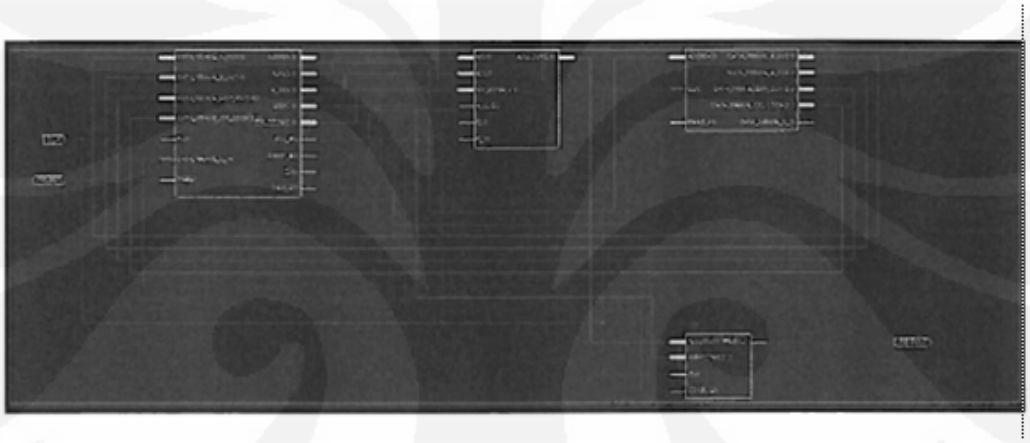
It can be seen. from a diagram above**,** there axe 4 inputs of COMP block, they are: ALU_OUT_EXPECTED, CLK, COMP_EN. (3:0) means 3 downs to zero as we zero as we can see in COMP_TB.vhd, or we can say it is 4 bits code. COMP_EN and EXPECTED are output of CRTL_FSM block ALU_OUT is output of ALU block. Clock is useful as a timing signal. It controls whwn all the memory elements can change state.

Inside COMP block, there is a process of comparison between ALU_OUT and EXPECTED. If both of them equal to 1, then the result is 1. otherwise, it will

become zero. The result become the output the output calculation.

In COMP_TB.vhd, there is a line of EXPECTED<= “0010” after 100ns, it made the RESULT became zero after 100ns because EXPECTED was not equal to 0001. we can see it on the result section.

5.5. CALC_TB



CALC_TB is a tenchbench to combine all four vhds, ALU, MEM, CTEL_FSM and COMP. as da diagram above, there are two inputs of it, CLK and reset. Furthermore the only output is result. In this CTRL_FSM became UUT1, ALU became UUT 2, MEM became UUT 3, COMP became UUT 4, UUT means Unit Under Test.

CHAPTER VI
CONCLUSION

1. Being able to understand the key concepts to create a VHDL project fulfilling the pre-requirement. The blocks are Important to be tested first before going to implementation.

2. VHDL language has been studied throughout the whole process. The basic programming style can be developed more to ge the real tolls such as Spartan 3e.

3. The project was held successfully since minor mistake did not have any significant matter to the project and test benching for all the 4 blocks uphold the project to be called as a succesfull project thus the comparison between the block are being presented.

BIBLIOGRAPHY

R. wain. (Nov. 2006) An Overview of FPGAS and FPGA programming. http://www.cse.scitech.ac.uk/disco/publications/FPGA_overview.pdf. Accessed Aug 08

FPGA4Fun. (Feb. 2008) How FPGAs Work. http://www.fpga4fun.com/FPGAinfo2.html. Accessed Aug 08

A.J. Coppola, (Oct. 1993), PLD. Complex PLD, and FPGA applications using VHDL, IEEE, Portland Oregon.

Department of Computer Science, University of Regina. http://www.es.uregina.ca/Links/class-info/301/register/lecture.html. accessed Aug 08

C.H. Roth, (1977) Digital Systems Design Using VHDL. PWS Publishing Company. ISBN 0-534-95099

M. Predko, Programming and customizing PICmicro MCU microcontroller McGraw-Hill ISBN 0-07-136172-3

M68HC11, (July, 2005) Freescale Semiconductor [Online] Available: http://www.freescale.com Accessed Sept. 2008

T. Jamil, (Aug-Sep, 1995), RISC versus CISC Potentials, IEEE Volume 14, Issue 3, Page (s): 13-16

K. Champan, Xilinx user community forums [Online] Available: http://forums.xilinx.com/xlnx/board/message?board.id=PicoBlaze&message.id=24 Accessed 2008, Aug 22

K. Chapman. Creating Embedded Microcontrollers (Programmable State Machines) [Online] Available: http://www.xilinx.com. Accessed Sept. 2008

Prof E. Roberts, The Intellectual Excitement of Computer Science RISC Architecture [Online] Available: http://cse.stanford.edu/class/sophomore-college/projects-00/rise/risccise/, Accessed Sept. 2008

K. Chapman. (2003, Oct) PicoBlaze KCPSM3 8-bit Microcontroller for Spartan-3 Xilinx Ltd, rev 7, www.xilinx.com. Accessed Spet. 2008

PicoBlaze 8-bit Embedded Microcontroller User Guide for Spartan-3, VIrtex-II, and Virtex-II Pro FPGAs UG129 (vl.1.2) June 24, 2008 [online] Available: http://www.xilinx.com. Accessed Aug 08

"Spartan-3E FPGA Starter Kit Board User Guide", ug230 vl.1 June 20, 2008, Xilinx Inc. http://www.Xilinx.com. Accessed Sept. 2008

W. Kilroy, M. Baker, M. Grace. FPGA Project Report, 2008: QUT.

http://en.wikipedia.org/wiki/digital_comparator. Accessed August 2009

Armstrong J.R. and F.G. Gray, VHDL Design Representation and Synthesis, Englewood Cliffs, NJ: Prentice Hall, 2nd Edition, 2000.

Bhasker J., VHDL Primer, Englewood Cliffs, NJ: Prentice Hall, 3rd Edition, 1999

Yalamanchi S, VHDL Starter's Guide, Englewood Cliffs, NJ: Prentice Hall, 1998

Appendix A : VHDL Codes

1. MY_CALC_PACKAGE.vhd :

```
Library IEEE;
Use IEEE.STD_CALC_PAK is

Package MY_CALC_PAK is

Type MY_ RECORD is
Record
    OP_CODE : std_logic_vector(3 downto 0); -- opcode
A_IN : std_logic_vector(3 downto 0); -- A operand
B_IN : std_logic_vector(3 downto 0); -- B operand
C_IN : std_logic:              -- C_In operand
    Exp_out : std_logic_vector(3 downto 0); -- expected output
End record;
end MY_CALC_PAK;
```

2. CTRL_FSM_TB.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

use CALC1_PAK.ALL;

ENTITY CNTRL_FSM_TB-vhd IS
END CNTRL_FSM_TB_vhd;

ARCHITECTUTER behavior OF CNTRL_FSM_TB-vhd IS
```

```
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT CNTRL_FSM
PORT(
        DATA_FRAME : IN MY_RECORD;
        CLK : IN std_logic;
        RESET : IN std_logic;
        MEM_EN : OUT std_logic;
        ADDR : OUT std_logic_vector(3 downto 0);
        ALU_EN : OUT std-logic;
        A_IN : OUT std_logic_vector(3 downto 0);
        B_IN : OUT std_logic_vector(3 downto 0);
        OP_CODE : OUT std_logic_vector(3 downto 0);
        C_IN : OUT std_logic;
        COMP_EN : OUT std_logic;
        EXP : OUT std_logic_vector(3 downto 0);
        );
END COMPONENT;

--Inputs
SIGNAL DATA_FRAME : MY-RECORD :=
(A_IN=>*0111*,B_IN=>"0011*,OP_CODE=>*0000*,C_IN=>'0',EXP_OUT=>"0111");
SIGNAL CLK : std_logic :='0';
SIGNAL RESET : std_logic :='0';

--Outputs
SIGNAL MEM_EN : std_logic;
SIGNAL ADDR : std_logic_vector(3 downto 0);
SIGNAL ALU_EN : std_logic;
SIGNAL A_IN : std_logic_vector(3 downto 0);
SIGNAL B_IN : std_logic_vector(3 downto 0);
SIGNAL OP_CODE : std_logic_vector(3 downto 0);
SIGNAL C_IN : std_logic;
```

```
        SIGNAL COMP_EN : std_logic;
        SIGNAL EXP : std_logic_vector(3 downto 0);



BEGIN

        -- Instantiate the Unit Under Test (UUT)
        uut: CNTRL_FSM PORT MAP(
                DATA_FRAME => DATA_FRAME,
                CLK => CLK
                RESET => RESET.
                MEM_EN => MEM_EN.
                ADDR => ADDR.
                ALU_EN => ALU_EN,
                A_IN => A_IN,
                B_IN => B_IN,
                OP_CODE => OP_CODE,
                C_IN => C_IN,
                COMP_EN => COMP_EN,
                EXP => EXP
        );

CLK <= not CLK after 1 Ons; -- 50 MHz clock
RESET <= '0'.'1' after 1 ons, '0' after 25ns, '1' after 800ns, '0' after 825ns;

        --test_proc : PROCESS
        --BEGIN
-- test bench code here
tb : PROCESS
BEGIN
        DATA_FRAME <= ("1000", "0100", "0000", "0000");
```

```
        wait for 100 ns;
        DATA_FRAME <=(“1000”,”0100”,”0101”,”0”,”0000”);
        wait for 100 ns;
        DATA_FRAME <=(“1000”,”0100”,”0101”,”0”,”0000”);
        Wait; - will wait forever
    --END PROCESS test-proc;


    END;
```

3. CTRL_FSM.vhd :

```
    library ieee;
    use IEEE.STD_LIGIC_1164.ALL;
    use IEEE.STD_LIGIC_ARITH.ALL;
    use IEEE.STD_LIGIC_UNSIGNED.ALL;

    use MY_CALC_PAK.ALL

    entity CNTRL_FSM is
        port (DATA_FRAME : in MY_RECORD;
            CLK : in STD_LOGIC;
            RESET : in STD_LOGIC;
            MEM-EN : out STD_LOGIC;
            ADDR : out STD_LOGIC_VECTOR(4 downto 0);
            ALU_EN : out STD_LOGIC;
            A_IN : out STD_LOGIC_VECTOR(3 downto 0);
            B_IN : out STD_LOGIC_VECTOR(3 downto 0);
            OP_CODE : out STD_LOGIC_VECTOR(3 downto 0);
            C_IN : out STD_LOGIC;
                                COMP_EN : out std_logic;
                                EXP : out std_logic_vector(3 downto 0);
    end CNTRL_FSM;
```

```
architecture Behavioral of CNTRL_FSM is

type State is (INIT, FETCH, ALU, COMP, DONE);
signal Curr_State, Next_State: State;
signal ADDR_INT : std_logic_vector(4 downto 0);
signal ADDR_Q : std_logic_vector(4 downto 0);


begin
ADDR <= ADDR_Q

Sync: process (CLK.RESET)
begin
if RESET = T then
    Curr_State <= Next_State;
        ADDR_Q <= (others => '0')

elsif rising_edge(CLK) then
    curr_State <= Next_State;
        ADDR_Q <= ADDR_INT;
end if;
end process Syne;

find_Next_State: process (Curr_State,DATA_FRAME,ADDR_Q)
begin
A_IN <= DATA_FRAME.A_IN;
B_IN <= DATA_FRAME.B_IN;
C_IN <= DATA_FRAME.C_IN;
OP_CODE  <= DATA_FRAME.OP_CODE;
ADDR_INT <= ADDR_Q
EXP <= DATA_FRAME.EXP_OUT;
```

```
case (Curr_State) is
    when INIT => -- set up initial defaults
        MEM_EN <= '0';
            Alu_en <= '0';
        COMP_EN <= '0';

        ADDR_INT <= (others => '0');
        Next_State <= FETCH;

when FETCH =>
MEM_EN <= '1';
        ALU_EN <= '0';
        COMP_EN <= '0';
Next_State <= ALU;

when ALU =>
MEM_EN <= '0';
        ALU_EN <= '1';
        COMP_EN <= '0';
Next_State <= COMP;

when COMP =>
MEM_EN <= '0';
        ALU_EN <= '0';
        COMP_EN <= '1';

when DONE =>
MEM_EN <= '0';
        ALU_EN <= '0';
        COMP_EN <= '0';

IF ADDR_Q < "11111" then
        ADDR_INT <= ADDR_Q + '1';
```

```
            Next_State <= FETCH;
            else
            Next_State <= DONE;
    end if;

    when others =>
    MEM_EN <= '0';
            ALU_EN <= '0';
            COMP_EN <= '0';
            Next_State <= INIT;
    end case;
    end process Find_Next_State;

    end Bhavioral;
```

4. COMP.vhd :

```
    library IEEE;
    use IEEE.STD_LIGIC_1164.ALL;
    use IEEE.STD_LIGIC_ARITH.ALL;
    use IEEE.STD_LIGIC_UNSIGNED.ALL;

    entity COMP is
        port (COMP_EN : in STD_LOGIC;
            CLK : in STD_LOGIC;
            EXPECTED : in STD_LOGIC_VECTOR(3 downto 0);
            ALU_OUT : in STD_LOGIC_VECTOR(3 downto 0);
            RESULT : out STD_LOGIC;
    End COMP;

    Architecture RTL of COMP is

    Begin
```

```
Process (CLK)
Begin
If rising_edge(CLK) then
    If comp_en = 'I' then
        If ALU_OUT = EXPECTED then
            RESULT <= '1';
            Else
            RESULT <= '0';
            End if;
    End if;
End if;
End process;

End RTL;
```

5. comp_tb.vhd :

```
LIBRARY ieee;
USE ieee.std_ligic_1164.ALL;
USE ieee.std_ligic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY COMP_TB_vhd IS
END COMP_TB_vhd;

ARCHITECTURE test OF COMP_TB_vhd IS

        -- Component Declaration for the Unit Under Test (UUT)
        COMPONENT COMP
```

```
PORT(
                                        COMP_EN : in STD_LOGIC;
        CLK : in STD_LOGIC;
        EXPECTED : in STD_LOGIC_VECTOR(3 downto 0);
        ALU_OUT : in STD_LOGIC_VECTOR(3 downto 0);
        RESULT : out STD_LOGIC
        );
END COMPONENT;

--Inputs
SIGNAL COMP_EN: std_logic : = '1'
SIGNAL CLK : std_logic : = '0'
SIGNAL EXPECTED : std_logic_vector(3 downto 0);= "0001";
SIGNAL ALU_OUT : std_logic_vector(3 downto 0);= "0001";

--Output
SIGNAL RESULT : std_logic;



BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: COMP PORT MAP(
        COMP_EN => COMP_EN,
        CLK => CLK,
        EXPECTED => EXPECTED,
        ALU_OUT => ALU_OUT,
        RESULT => RESULT
);
```

```
CLK <= not CLK after 10ns;

        test_proc : PROCESS
        BEGIN
Wait for 5ns;
        Expected <= “0010” after 100ns

        Wait;

        END PROCESS test_proc;

END;
```

6. MY_CALC.vhd :

```
library IEEE;
use IEEE.STD_LIGIC_1164.ALL;
use IEEE.STD_LIGIC_ARITH.ALL;
use IEEE.STD_LIGIC_UNSIGNED.ALL;




use MY_CALC_PAK.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

Entity MY_CALC is
    --generic (SYNTH : Boolean := false);
```

```
    Port (CLK : in STD_LOGIC;
        RESET : in STD_LOGIC;
        RESULT : out STD_LOGIC;
and MY_CALC;

architecture Structural of MY_CALC is

component MEM
port (CLK : in STD_LOGIC;
    ADDR : in STD_LOGIC_VECTOR (4 downto 0);
    READ_EN : in STD_LOGIC;
    DATA_FRAME : out MY_ RECORD);
End component;

Component CNTRL_FSM
Port (DATA_FRAME : in MY_RECORD

    CLK : in STD_LOGIC;
    RESET : in STD_LOGIC;
    MEM_EN : out STD_LOGIC
    ADDR : out STD_LOGIC_VECTOR(4 downto 0);
    ALU_EN : out STD_LOGIC
    A_IN : out in STD_LOGIC_VECTOR(3 downto 0);
    B_IN : out STD_LOGIC_VECTOR(3 downto 0);
    OP_CODE : out STD_LOGIC_VECTOR(3 downto 0);
    C_IN : out STD_LOGIC

                    COMP_EN : out std_logic;
                    EXP : out std_logic_vector(3 downto 0);
End component;

Component ALU
Port (A : in STD_LOGIC_VECTOR(3 downto 0);
```

```
    B : in STD_LOGIC_VECTOR(3 downto 0);
    C_IN : in STD_LOGIC
    OP_CODE : in STD_LOGIC_VECTOR(3 downto 0);
    CLK : in STD_LOGIC
    ALU_EN : in STD_LOGIC
    ALU_OUT : out STD_LOGIC_VECTOR(3 downto 0);
End component;

Component COMP
Port (COMP_EN : in STD_LOGIC;
    CLK : in STD_LOGIC
    EXPECTED : in STD_LOGIC_VECTOR(3 downto 0);
    ALU_OUT : in STD_LOGIC_VECTOR(3 downto 0);
    RESULT : out STD_LOGIC);
End component;

Signal DATA_FRAME_SIG : MY_RECORD;
Signal ADDR_SIG : STD_LOGIC_VECTOR(4 downto 0);
Signal MEM_EN_SIG, ALU_EN_SIG, COMP_EN_SIG, C_IN_SIG : std_logic;
Signal A-IN_SIG, B_IN_SIG, OP_CODE_SIG, EXP_OUT_SIG, ALU_OUT_SIG;
STD_LOGIC_VECTOR(3 downto 0);
Begin
U1 : MEM port map
(CLK=>CLK.ADDR=>ADDR_SIG.READ_EN=>MEM_EN_SIG.DATA_FRA
ME=>DATA_PRAME_SIG):

U2 : CNTRL_FSM port map
(DATA_FRAME=>DATA_FRAME_SIG,CLK=>CLK,RESET=>RESET,MEM
_EN=>MEM_EN_SIG,
ADDR=>ADDR_SIG,ALU_EN=>ALU_SIG,A_IN=>A_IN_SIG,B_IN=>B_IN_
SIG,OP_CODE=>OP_CODE_SIG.
C_IN=>C_IN_SIG,COMP_EN=>COMP_EN_SIG,EXP=>EXP_OUT_SIG);
```

```
U3 : ALU port map
(A=>A_IN_SIG,B=>B_IN_SIG,CLK=>CLK,EXPECTED=>EXP_OUT_SIG,AL
U_OUT=>ALU_OUT_SIG,
RESULT=>RESULT);

End Structural;
```

7. MY_CALC_TB.vhd

```
LIBRARY ieee;
USE ieee.std_ligic_1164.ALL;
USE ieee.std_ligic_unsigned.all;

USE ieee.numeric_std.ALL;

ENTITY MY_CALC_TB_vhd IS
END MY_CALC_TB_vhd;

ARCHITECTURE test OF MY_CALC_TB_vhd IS

        --Components Declaration for the Unit Under Test (UUT)
        COMPONENT MY_CALC
        PORT(
                CLK_IN : IN std_logic;
                RESET : IN std_logic;
                RESULT_OUT : OUT std_logic
                );
        END COMPONENT;

        --Input

        SIGNAL CLK : std_logic : = '0';
        SIGNAL RESET_TB : std_logic : = '0';
```

```
        --Output
        SIGNAL RESULT : std_logic;

BEGIN
        --Instantiate the Unit Under Test (UUT)
        Uut: MY_CALC PORT MAP(

                CLK_IN=>CLK_TB,
                RESET=>RESET_TB
                RESULT_OUT=>RESULT
        );

CLK_TB <= not CLK_TB after 10ns;
RESET_TB <= T after 15ns, '0' after 25ns, T after 700ns, '0' after 725 ns;
End architecture TEST;
```

7. ALU,VHD file

```
USE ieee.std_ligic_1164.ALL;                    --Library Declaration
USE ieee.std_ligic_unsigned.all;        --Packages std_logic_vector
USE ieee.numeric_std.ALL;                       --Packages numeric operation

ENTITY ALU_TB_VHD IS                                    --Primary design unit
END ALU_TB_VHD;

Architecture behavior OF ALU_TB_vhd IS

        --Component Declaration for the Unit Unver Test (UUT)
        COMPONENT ALU
        PORT(
                A : IN std_logic_vector(3 downto 0);            --Block port
                B : IN std_logic_vector(3 downto 0);
```

```
                C_IN : IN std_logic;
                OP_CODE : IN std_logic_vector(3 downto 0);
                CLK : IN std_logic;
                ALU_EN : IN std_logic;
                ALU_OUT : OUT std_logic_vector(3 downto 0);
                );
        END COMPONENT;

        --Input
        SIGNAL C_IN : std_logic : = '0';          --input signals
        SIGNAL CLK : std_logic : = '0';
        SIGNAL ALU_ EN : std_logic : = '1';
        SIGNAL A : std_logic_vector(3 downto 0) : = "0111";
        SIGNAL B : std_logic_vector(3 downto 0) : = "0011";
        SIGNAL OP_CODE : std_logic_vector(3 downto 0) : = (others=> '0');

        --Output
        SIGNAL alu_out : std_logic_vector(3 downto 0);          --Output signal

BEGIN

        --Instantiate the Unit Under Test (UUT)
        Uut: ALU PORT MAP(
                A => A
                B => B,
                C_IN => C_IN,
                OP_CODE => OP-CODE,
                CLK => CLK,
                ALU_EN => ALU_EN,
                ALU_OUT => ALU_OUT
        );
CLK <= not CLK after 5ns;
```

```
        Test_proc : PROCESS
        BEGIN
OP_CODE (3 downto 0) <= “0000”; --txa
        C_IN <= ‘0’;
        wait for 20ns;
OP_CODE (3 downto 0) <= “0001”; --txa
        C_IN <= ‘1’;
        wait for 20ns;

OP_CODE (3 downto 0) <= “0001”; --inc
        C_IN <= ‘0’;
        wait for 20ns;

OP_CODE (3 downto 0) <= “0001”; --inc
        C_IN <= ‘1’;
        wait for 20ns;

OP_CODE (3 downto 0) <= “0010”; --add
        C_IN <= ‘0’;
        wait for 20ns;

OP_CODE (3 downto 0) <= “0010”; --add
        C_IN <= ‘1’;
        wait for 20ns;

OP_CODE (3 downto 0) <= “0011”; --adde
        C_IN <= ‘0’;
        wait for 20ns;

OP_CODE (3 downto 0) <= “0011”; --adde
        C_IN <= ‘1’;
        wait for 20ns;
```

```
OP_CODE (3 downto 0) <= “0100”; --sub
        C_IN <= ‘0’;
        wait for 20ns;

OP_CODE (3 downto 0) <= “0100”; --sub
C_IN <= ‘1’;
wait for 20ns;


OP_CODE (3 downto 0) <= “0101”; --comp
C_IN <= ‘0’;
wait for 20ns;

OP_CODE (3 downto 0) <= “0101”; --comp
C_IN <= ‘1’;
wait for 20ns;

OP_CODE (3 downto 0) <= “0110”; --neg
C_IN <= ‘0’;
wait for 20ns;

OP_CODE (3 downto 0) <= “0110”; --neg
C_IN <= ‘1’;
wait for 20ns;

OP_CODE (3 downto 0) <= “0111”; --dec
C_IN <= ‘0’;
wait for 20ns;

OP_CODE (3 downto 0) <= “0111”; --dec
C_IN <= ‘1’;
wait for 20ns;

OP_CODE (3 downto 0) <= “1000”; --txb
```

```
C_IN <= '0';
wait for 20ns;

OP_CODE (3 downto 0) <= "1000"; --txb
C_IN <= '1';
wait for 20ns;

OP_CODE (3 downto 0) <= "1001"; --and
C_IN <= '0';
wait for 20ns;


OP_CODE (3 downto 0) <= "1001"; --and
C_IN <= '1';
wait for 20ns;

OP_CODE (3 downto 0) <= "1010"; --or
C_IN <= '0';
wait for 20ns;

OP_CODE (3 downto 0) <= "1010"; --or
C_IN <= '1';
wait for 20ns;

OP_CODE (3 downto 0) <= "1011"; --xor
C_IN <= '0';
wait for 20ns;

        OP_CODE (3 downto 0) <= "1011"; --xor
C_IN <= '1';
wait for 20ns;

OP_CODE (3 downto 0) <= "1100"; --sl
C_IN <= '0';
```

```
wait for 20ns;

        OP_CODE (3 downto 0) <= “1100”; --sl
C_IN <= ‘1’;
wait for 20ns;

OP_CODE (3 downto 0) <= “1101”; --sr
C_IN <= ‘0’;
wait for 20ns;

OP_CODE (3 downto 0) <= “1101”; --sr
C_IN <= ‘1’;
wait for 20ns;


OP_CODE (3 downto 0) <= “1110”; --par
C_IN <= ‘0’;
wait for 20ns;

OP_CODE (3 downto 0) <= “1111”; --zero
C_IN <= ‘0’;
wait for 20ns;

OP_CODE (3 downto 0) <= “1111”; --zero
C_IN <= ‘1’;
wait for 20ns;

wait;
END PROCESS tes_proc;

END;
```

8. MEM_TB.VHD file

```
library IEEE;                                   --Library Declaration
use IEEE.STD_LIGIC_1164.ALL;                    --Packed Declaration
use IEEE.STD_LIGIC_ARITH.ALL;                   --Packed operation and function
use IEEE.STD_LIGIC_UNSIGNED.ALL; --Packages std_logic_vector

use work. MY_CALC_PAK.ALL;

----any Xilinx primitives in this code,
--library UNISIM;
--use UNISIM.VComponents.all;

Entity MEM_TB is                                -- Primary design unit MEM
End MEM_TB;                                     -- Close entity declaration

Architecture test of MEM_TB is     -- Secondary design unit

Component MEM
        Port (CLK : in STD_LOGIC;
    ADDR : in STD_LOGIC_VECTOR (4 downto 0); --Port declaration
    READ_EN : in STD_LOGIC;
    DATA_FRAME : out MY_RECORD);
End component;

Signal CLK; std_logic ;='0'
Signal ADDE_SIG : std_logic vector (4 downto 0) : = "0000"; --local signals
Signal READ_EN_SIG : std_logic ;= '1';
Signal DATA_FRAME_SIG : MY_RECORD;

Begin                                                   -- Begin architecture

Uut: MEM port map (CLK => CLK,
    ADDR => ADDR_SIG,                   -- unit under testing for testing
    READ_EN => READ_EN_SIG,
```

```
    DATA_FRAME =>DATA_FRAME_SIG);

CLK <= not CLK after 10ns;

Process
Begin                                                    -- Begin the process

ADDR_SIG <= “00000”; wait for 100ns;
ADDR_SIG <= “00001”; wait for 100ns;
ADDR_SIG <= “00010”; wait for 100ns;
ADDR_SIG <= “00100”; wait for 100ns;
ADDR_SIG <= “00101”; wait for 100ns;
ADDR_SIG <= “00110”; wait for 100ns;
ADDR_SIG <= “00111”; wait for 100ns;
ADDR_SIG <= “01000”; wait for 100ns;
ADDR_SIG <= “01001”; wait for 100ns;
ADDR_SIG <= “01010”; wait for 100ns;     --Statements operate sequentially
ADDR_SIG <= “01011”; wait for 100ns;
ADDR_SIG <= “01100”; wait for 100ns;
ADDR_SIG <= “01101”; wait for 100ns;
ADDR_SIG <= “01110”; wait for 100ns;
ADDR_SIG <= “01111”; wait for 100ns;
ADDR_SIG <= “10000”; wait for 100ns;
ADDR_SIG <= “10001”; wait for 100ns;
ADDR_SIG <= “10010”; wait for 100ns;
ADDR_SIG <= “10011”; wait for 100ns;
ADDR_SIG <= “10100”; wait for 100ns;
ADDR_SIG <= “10101”; wait for 100ns;
ADDR_SIG <= “10110”; wait for 100ns;
ADDR_SIG <= “10111”; wait for 100ns;
ADDR_SIG <= “11000”; wait for 100ns;
ADDR_SIG <= “11001”; wait for 100ns;
ADDR_SIG <= “11010”; wait for 100ns;
```

```
ADDR_SIG <= “11011”; wait for 100ns;
ADDR_SIG <= “11100”; wait for 100ns;
ADDR_SIG <= “11101”; wait for 100ns;
ADDR_SIG <= “11110”; wait for 100ns;
ADDR_SIG <= “11111”; wait for 100ns;


Wait;

End process;                                -- End process

End architecture test;                      -- End the body architecture
```

9. The memory management in MY_ROM;

```
(0 =>
(OP_CODE=>”0000”,A_IN=>”0111”,B_IN=>”0011”,c_in=>’0’,EXP_OUT=>”0111”), --txa
        1 =>
(OP_CODE=>”0000”,A_IN=>”0111”,B_IN=>”0011”,c_in=>’0’,EXP_OUT=>”0111”), --txa
        2 =>
(OP_CODE=>”0001”,A_IN=>”0111”,B_IN=>”0011”,c_in=>’0’,EXP_OUT=>”1000”), --inc
        3 =>
(OP_CODE=>”0001”,A_IN=>”0111”,B_IN=>”0011”,c_in=>’0’,EXP_OUT=>”1000”), --inc
        4 =>
(OP_CODE=>”0010”,A_IN=>”0111”,B_IN=>”0011”,c_in=>’0’,EXP_OUT=>”1010”), --
add
        5 =>
(OP_CODE=>”0010”,A_IN=>”0111”,B_IN=>”0011”,c_in=>’0’,EXP_OUT=>”1010”), --
add
        6 =>
(OP_CODE=>”0011”,A_IN=>”0111”,B_IN=>”0011”,c_in=>’0’,EXP_OUT=>”1010”), --
addc
        7 =>
```

```
(OP_CODE=>"0011",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"1011"), --
addc
        8 =>
(OP_CODE=>"0100",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0100"), --
sub
        9 =>
(OP_CODE=>"0100",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0100"), --
sub
        10 =>
(OP_CODE=>"0101",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"1000"), --
comp
        11 =>
(OP_CODE=>"0101",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"1000"), --
comp
        12 =>
(OP_CODE=>"0110",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"1001"), --
neg




        13 =>
(OP_CODE=>"0110",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"1001"), --
neg
        14 =>
(OP_CODE=>"0111",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0110"), --
dec
        15 =>
(OP_CODE=>"0111",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0110"), --
dec
        16 =>
(OP_CODE=>"1000",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0011"), --
txb
        17 =>
```

```
(OP_CODE=>"1000",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0011"), --
txb
        18 =>
(OP_CODE=>"1001",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0011"), --
and
        19 =>
(OP_CODE=>"1001",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0011"), --
and
        20 =>
(OP_CODE=>"1010",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0111"), --or
        21 =>
(OP_CODE=>"1010",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0111"), --or
        22 =>
(OP_CODE=>"1011",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0100"), --
xor
        23 =>
(OP_CODE=>"1011",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0100"), --
xor
        24 =>
(OP_CODE=>"1100",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"1110"), --sl

        25 =>
(OP_CODE=>"1100",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"1110"), --sl
        26 =>
(OP_CODE=>"1101",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0011"), --sr
        27 =>
(OP_CODE=>"1101",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0011"), --sr
        28 =>
(OP_CODE=>"1110",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0001"), --
par
        29 =>
(OP_CODE=>"1110",A_IN=>"0111",B_IN=>"0011",c_in=>'0',EXP_OUT=>"0000"), --
par
        30 =>
```

(OP_CODE=>”1111”,A_IN=>”0111”,B_IN=>”0011”,c_in=>’0’,EXP_OUT=>”0000”), --zero

31 =>
(OP_CODE=>”1111”,A_IN=>”0111”,B_IN=>”0011”,c_in=>’0’,EXP_OUT=>”0000”), --zero