|  |
| --- |
| 10-bit Single-Cycle CPU |
| ECE3570 Lab3b |
| Junliang Fei, Adam Dubs, Connelly Bale |
| 03/21/2018 |

**Table of Contents**

[1. Overview of ISA 2](#_Toc509267836)

[2. Design Result 4](#_Toc509267837)

[2.1. instr\_ROM 4](#_Toc509267838)

[2.1.1. Interface 4](#_Toc509267839)

[2.1.2. Internal Logic 4](#_Toc509267840)

[2.1.3. Schematic 4](#_Toc509267841)

[2.1.4. Simulation 4](#_Toc509267842)

[2.2. RAM 5](#_Toc509267843)

[2.2.1. Interface 5](#_Toc509267844)

[2.2.2. Internal Logic 6](#_Toc509267845)

[2.2.3. Schematic 6](#_Toc509267846)

[2.2.4. Simulation 6](#_Toc509267847)

[2.3. CPU10Bits 8](#_Toc509267848)

[2.3.1. Interface 8](#_Toc509267849)

[2.3.2. Internal Logic 8](#_Toc509267850)

[2.3.3. Schematic 8](#_Toc509267851)

[2.3.4. Simulation 9](#_Toc509267852)

[2.3.5. Implementation 14](#_Toc509267853)

# Overview of ISA

There is **NO CHANGE** in the design result of lab1. The ISA architecture of lab1 is showed as below.

**BASIC INSTRUCTION FORMATS**

R-TYPE (opcode = 00 or 01)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| opcode | | function | | rs | | | rt or immediate | | |
| bit9 | bit8 | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
|  |  |  |  |  |  |  |  |  |  |

I-TYPE (opcode = 10)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| opcode | | immediate | | | | | | | |
| bit9 | bit8 | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
|  |  |  |  |  |  |  |  |  |  |

J-TYPE (opcode = 11)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| opcode | | function | | address | | | | | |
| bit9 | bit8 | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
|  |  |  |  |  |  |  |  |  |  |

**INSTRUCTION SET**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **NAME** | **MNEMONIC** | **FORMAT** | **Example** | **RTL Example** | **OPCODE/**  **FUNCT** |
| 1 | Add | add | R | add $s0, $s1 | reg\_t0 = reg\_s0 + reg\_s1; | 00/00 |
| 2 | Sub | sub | R | sub $s0, $s1 | reg\_t0 = reg\_s0 - reg\_s1; | 00/01 |
| 3 | Compare | cmp | R | cmp $s0, $s1 | if(reg\_s0 =reg\_s1), reg\_t0 = 1; | 00/10 |
| 4 | Less Than | slt | R | slt $s0, $s1 | if(reg\_s0 <reg\_s1), reg\_t0 = 1; | 00/11 |
| 5 | Move | mov | R | mov $s0, $t0 | reg\_s0 = reg\_t0; | 01/00 |
| 6 | Load Word | load | R | load 4($sp) | mem\_rd\_addr = reg\_sp+4;  reg\_t1 = mem\_rd\_data; | 01/01 |
| 7 | Store Word | str | R | str 0($sp) | mem\_wr\_addr = reg\_sp+4; | 01/10 |
| 8 | Return | ret | R | ret | instr\_addr = reg\_ra; | 01/11 |
| 9 | load immediate | ldi | I | ldi 17 | reg\_t1 = 17 | 10 |
| 10 | Jump | j | J | j LOOP | instr\_addr = instr\_addr + jump\_addr; | 11/00 |
| 11 | Jump True | je | J | je LOOP | if(reg\_t0 = 1)  instr\_addr = instr\_addr + jump\_addr; | 11/01 |
| 12 | Jump and link | jal | J | jal LOOP | reg\_ra = instr\_addr +1;  instr\_addr = instr\_addr + jump\_addr; | 11/10 |
| 13 | Halt | halt | J | halt | No operation | 11/11 |

**REGISTER LIST**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Name** | **Use** | **Bit** | **Notes** |
| 1 | $s0 | Saved Temporary 0 | 000 |  |
| 2 | $s1 | Saved Temporary 1 | 001 |  |
| 3 | $s2 | Saved Temporary 2 | 010 |  |
| 4 | $s3 | Saved Temporary 3 | 011 |  |
| 5 | $t0 | Temporary 0 | 100 | $t0 is used for saving the return value of add, sub, slt and cmp |
| 6 | $t1 | Temporary 1 | 101 | $t1 is used for saving the return value of load, str and ldi |
| 7 | $ra | Return Address | 110 |  |
| 8 | $sp | Stack Pointer | 111 |  |

# Design Result

## instr\_ROM

This unit realizes the function as instruction memory to hold instructions. It is initialized with the algorithms of lab1.

### Interface

The interface list of this unit is showed in *Table1*.

Table1. instr\_ROM Interface List

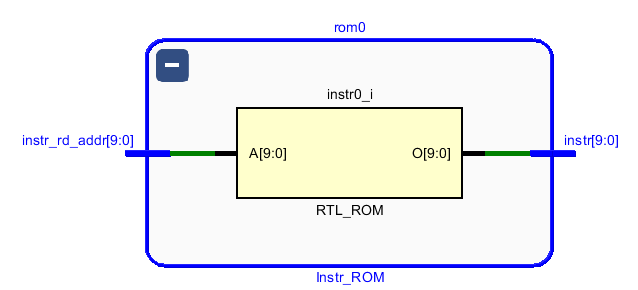
| **No** | **Name** | **Bit Number** | **IN/OUT** | **Function** |
| --- | --- | --- | --- | --- |
| 1 | instr\_rd\_addr | 1 | Input | instruction read address |
| 2 | instr | 1 | output | instr |

### Internal Logic

Because the largest instruction count of the three algorithms is 46, the width of the ROM is set to be 10 bits and the depth is 46(from 45 to 0)*.* According to the input read address, the data in the address is output.

### Schematic

The schematic of instr\_ROM is showed in *Figure1*.



Figuire1 Schematic of instr\_ROM module

### Simulation

The simulation specification is showed as below*.*

1. Initialize the ROM with the data showed as below.

Address 0x0 --- Data 0x0

Address 0x1 --- Data 0x1

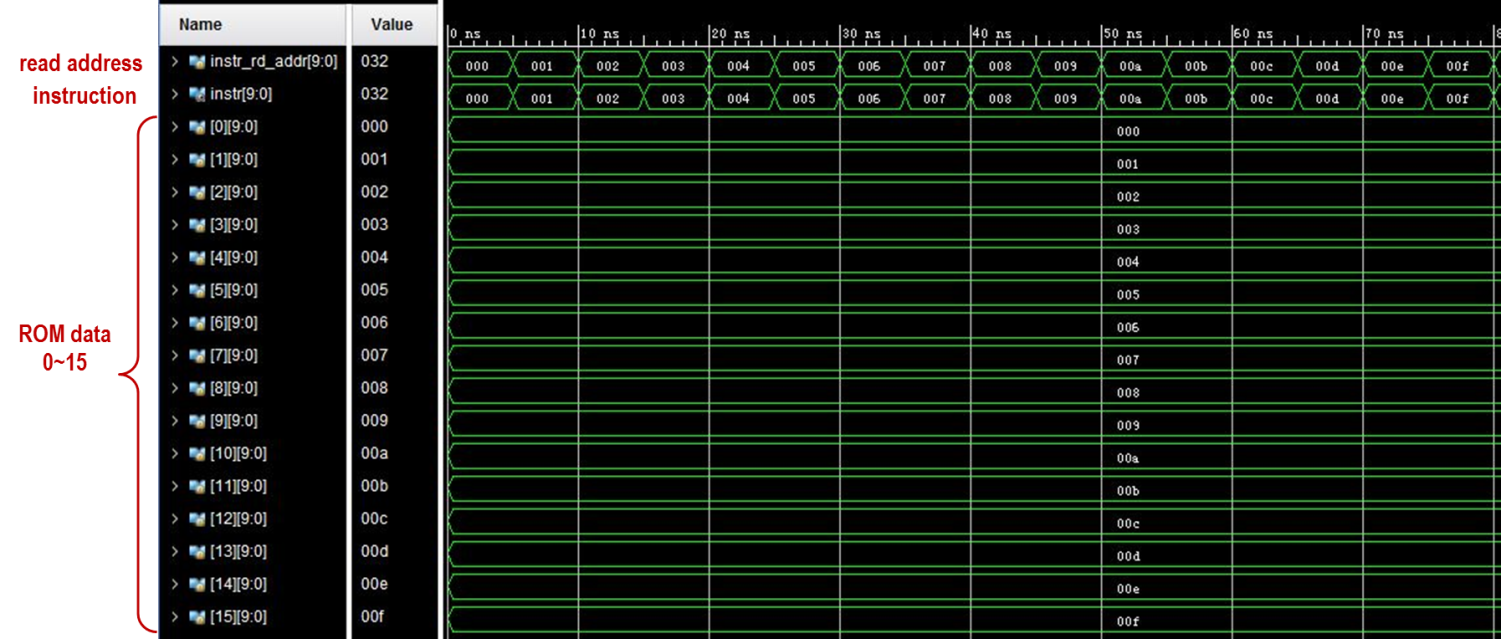
Address 0x2 --- Data 0x2

… … …

Address 0x2D --- Data 0x2D

1. Read data from the ROM. The address increases 1 every 5ns.

Moreover, the status of all RAM address is checked through the simulation waveform. It’s all correct. The simulation result is showed in *Figure2*. The data is read from ROM correctly.



Figuire2 Simulation Result of instr\_ROM

## RAM

This module realizes the function of data memory that holds program data.

### Interface

The interface list of this unit is showed in *Table4*.

Table4.RAM Interface List

| **No** | **Name** | **Bit Number** | **IN/OUT** | **Function** |
| --- | --- | --- | --- | --- |
| 1 | clk | 1 | Input | clock signal |
| 2 | alu\_out | 10 | Input | the output data of ALU module |
| 3 | t1 | 10 | Input | write data of RAM |
| 4 | ldst\_en | 2 | Input | control signal |
| 5 | ram\_out | 10 | output | output data of RAM module |

### Internal Logic

The description of this module is showed as below.

1. Write operation

When **ldst\_en** = **3**, the data of **t1** is written into the address of **alu\_out**.

1. Read operation

When **ldst\_en** = **2**, the data is read from address **alu\_out** and output on the interface of **ram\_out**. Otherwise, the **ram\_out** is the same as **alu\_out**.

The read logic is showed in *Table5.*

Table5. RAM Output Logic

| **No** | **ldst\_en** | **ram\_out** |
| --- | --- | --- |
| 1 | 2 | memory[alu\_out] |
| 2 | others | alu\_out |

### Schematic

There are two MUXs and one RAM in this module. The schematic of fetch unit is showed in *Figure3*.

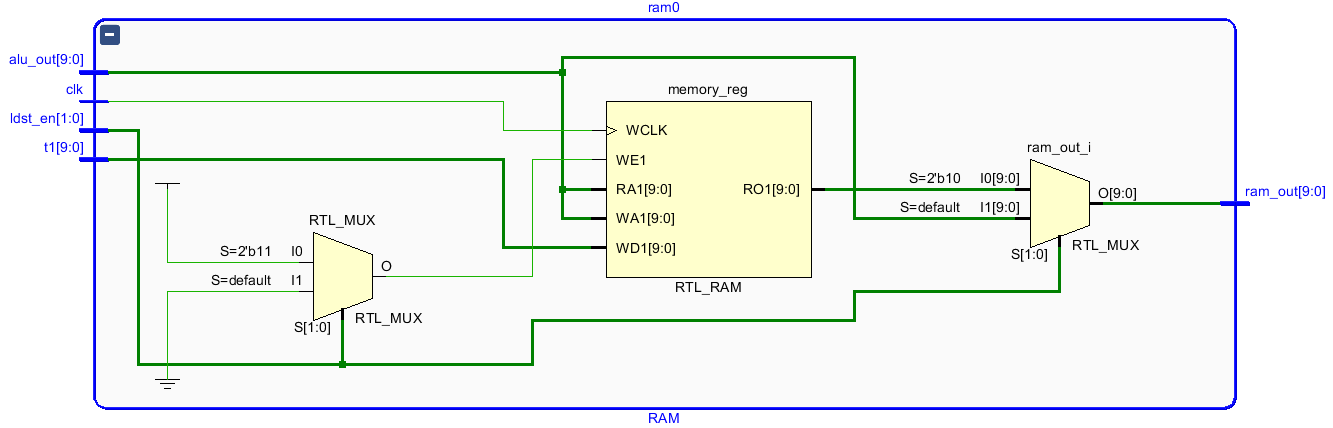


Figure3 Schematic of RAM module

### Simulation

The simulation specification is showed as below*.*

1. Initialize the data in all of the address to be 0x0.
2. Write data into RAM every 10ns. The value is **address + 0x200**, which is showed as below.

Address 0x0 --- Data 0x200

Address 0x1 --- Data 0x201

Address 0x2 --- Data 0x202

… … …

Address 0x1FF --- Data 0x3FF

Address 0x200 --- Data 0x0

Address 0x201--- Data 0x1

… … …

Address 0x3FF --- Data 0x1FF

1. Read data from the RAM. The address increases 1 every 10ns.

Moreover, the status of all RAM address is checked through the simulation waveform. It’s all correct.

The write simulation result is showed in *Figure4*.

1. The data write into RAM correctly.
2. The output data is equal to alu\_out.

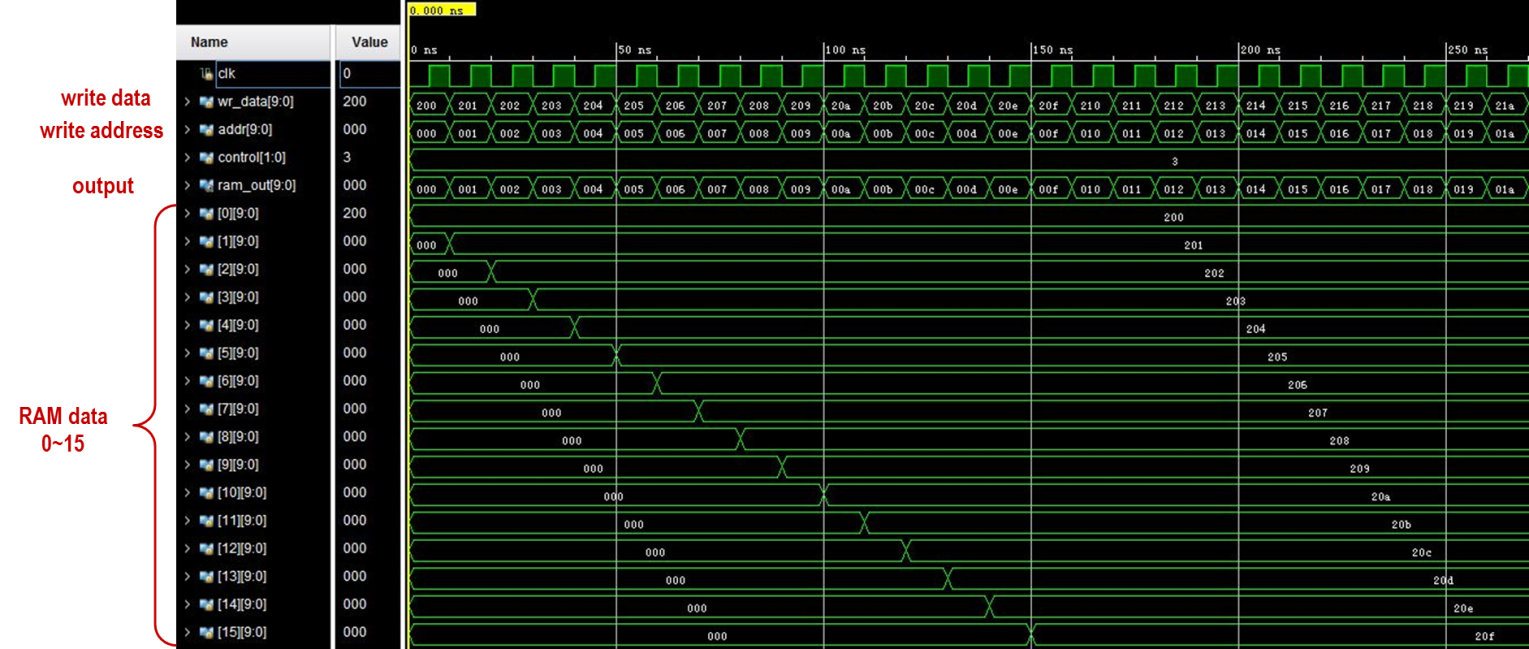


Figure4 Simulation result of write operation of RAM

The read simulation result is showed in *Figure5*. The data is read from RAM and output correctly.

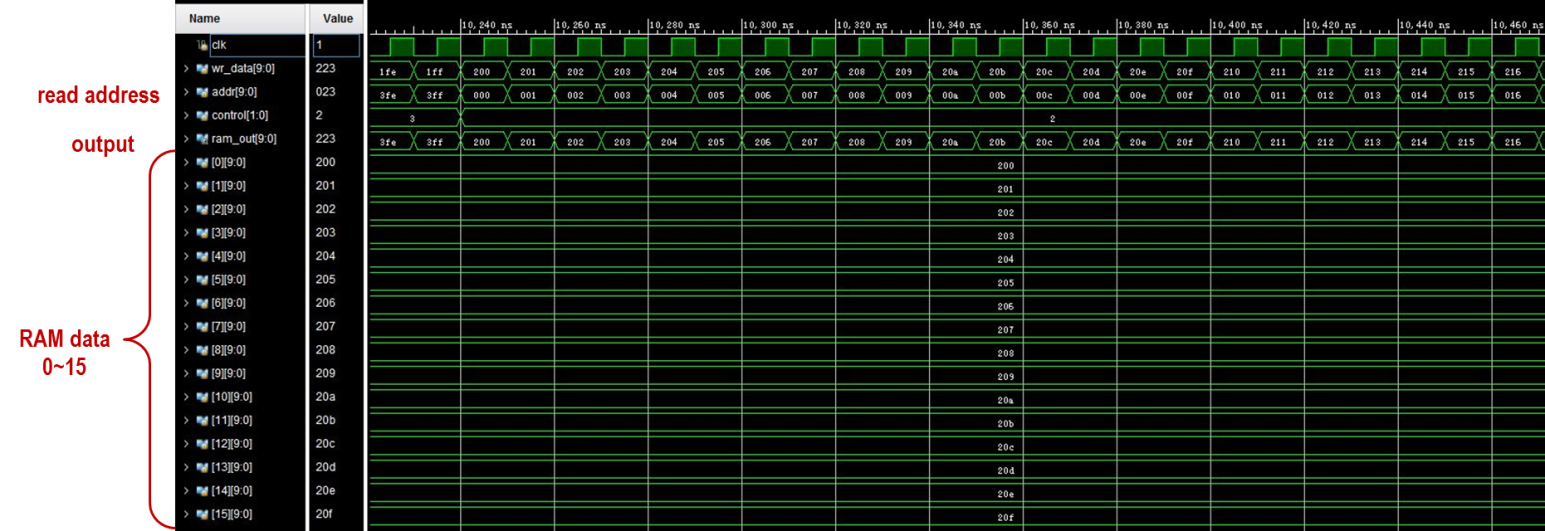


Figure5 Simulation result of read operation of RAM

## CPU10Bits

This is the top level of this project which is the connection of the sub-modules.

### Interface

The interface list of this unit is showed in *Table6*.

Table6. CPU10Bits Interface List

| **No** | **Name** | **Bit Number** | **IN/OUT** | **Function** |
| --- | --- | --- | --- | --- |
| 1 | clk | 1 | Input | clock signal |
| 2 | reset | 1 | input | reset signal |
| 3 | donebit | 1 | output | the program is done |
| 4 | inst\_value | 10 | output | the value of current instruction |

### Internal Logic

Besides the connection of sub-modules, there is logic for generating **inst\_value**.

When **wr\_en** is 1, **inst\_value** is equal to **ram\_out**. Otherwise, **inst\_value** is “*XXXXXXXXXX*”.

### Schematic

The schematic of CPU10Bits is showed in *Figure7*.

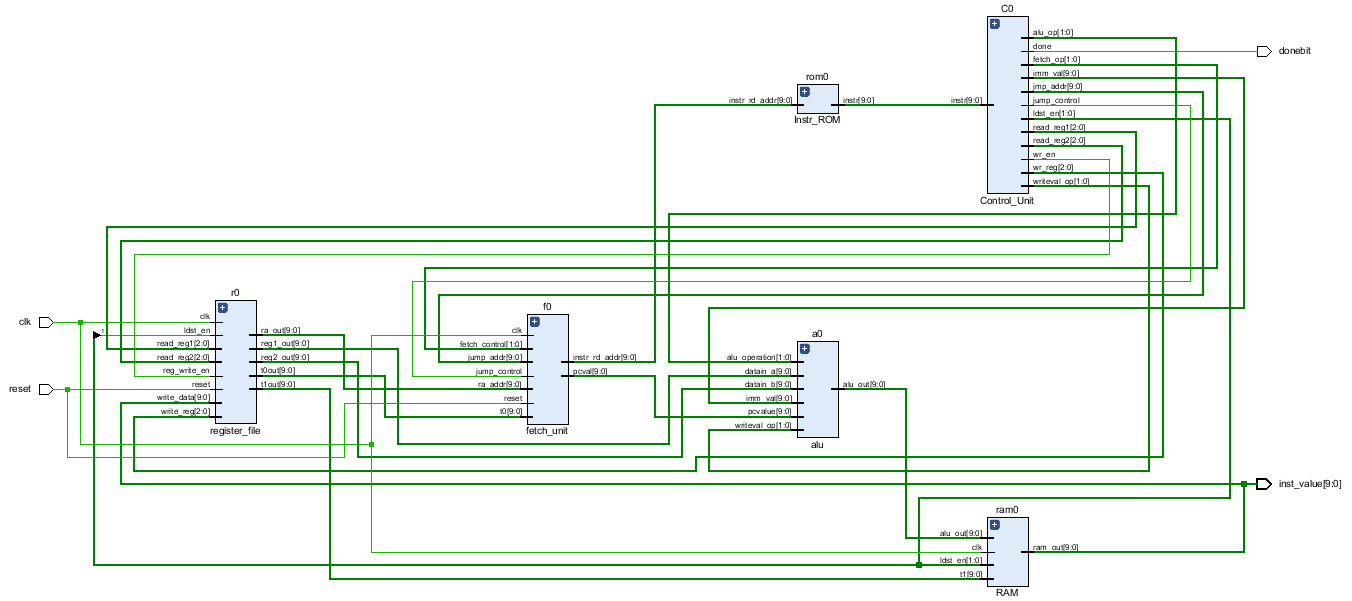


Figure7 Schematic of CPU10Bits module

### Simulation

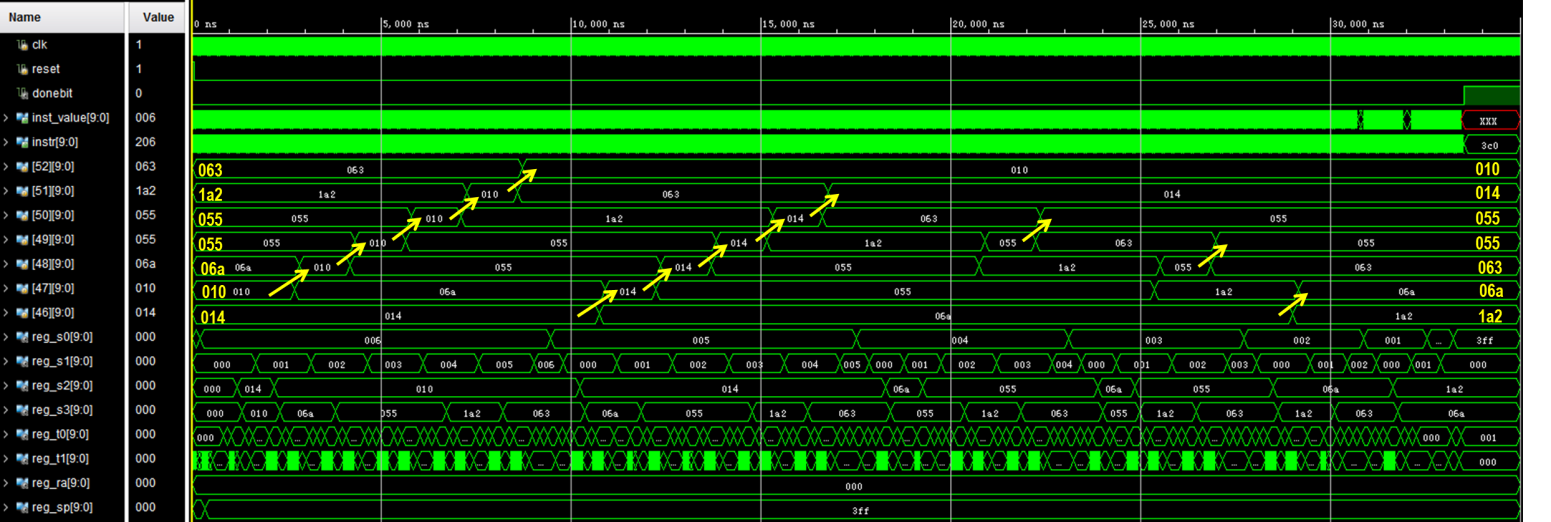
1. **Bubble Sort**

The simulation specification and result is showed as *Table7*.

Table7. Bubble Sort Simulation Specification and Result

| **No** | **Address** | **Original Value** | **Sorted Value** | **Expected Value** | **Result** |
| --- | --- | --- | --- | --- | --- |
| 1 | 46 | 0x14 | 0x1a2 | 0x1a2 | OK |
| 2 | 47 | 0x10 | 0x6a | 0x6a |
| 3 | 48 | 0x6a | 0x63 | 0x63 |
| 4 | 49 | 0x55 | 0x55 | 0x55 |
| 5 | 50 | 0x55 | 0x55 | 0x55 |
| 6 | 51 | 0x1a2 | 0x14 | 0x14 |
| 7 | 52 | 0x63 | 0x10 | 0x10 |

The simulation result is showed in *Figure8*. The data is sorted correctly.



Figuire8 Simulation Result of Bubble Sort

1. **X\*Y-4**

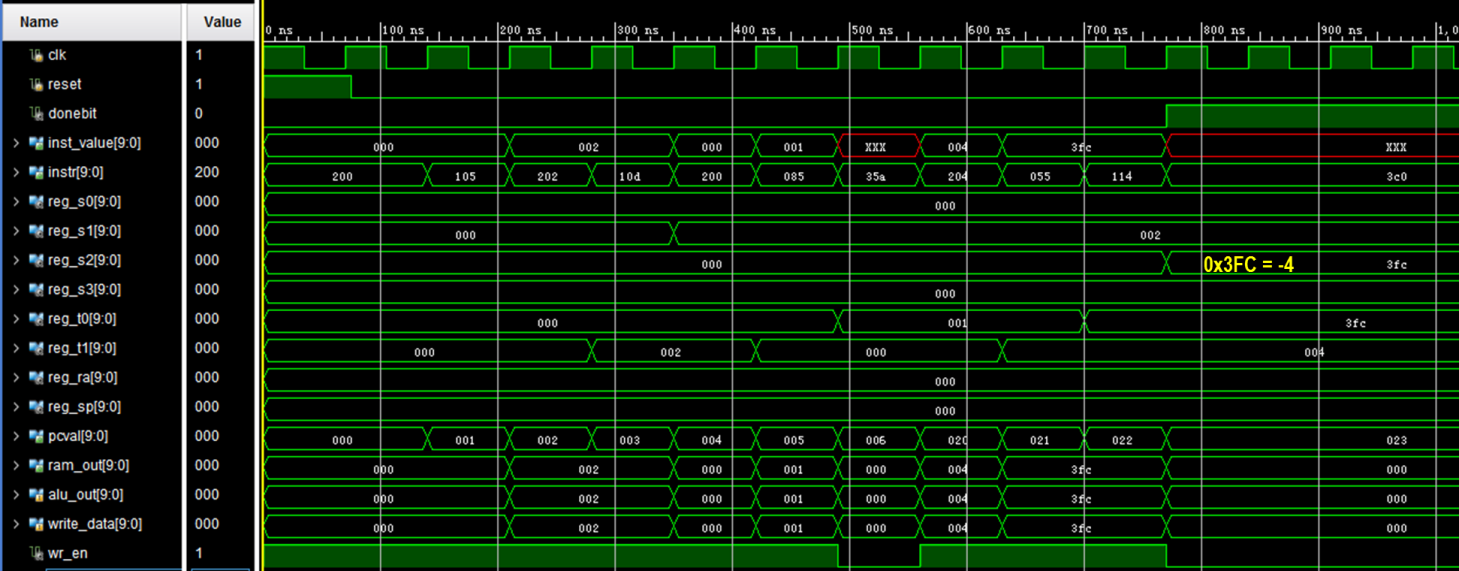
The simulation specification and result is showed as *Table7*. (The result is stored in **$S2**)

Table7. X\*Y-4 Simulation Specification and Result

| **No** | **X** | **Y** | **Calculated Value** | **Decimal Value** | **Expected Value** | **Result** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 2 | 0x3FC | -4 | -4 | OK |
| 2 | 2 | 0 | 0x3FC | -4 | -4 |
| 3 | 2 | 15 | 0x1A | 26 | 26 |
| 4 | -2 | 15 | 0x3DE | -34 | -34 |
| 5 | 2 | -5 | 0x3F2 | -14 | -14 |
| 6 | -3 | -5 | 0x00B | 11 | 11 |

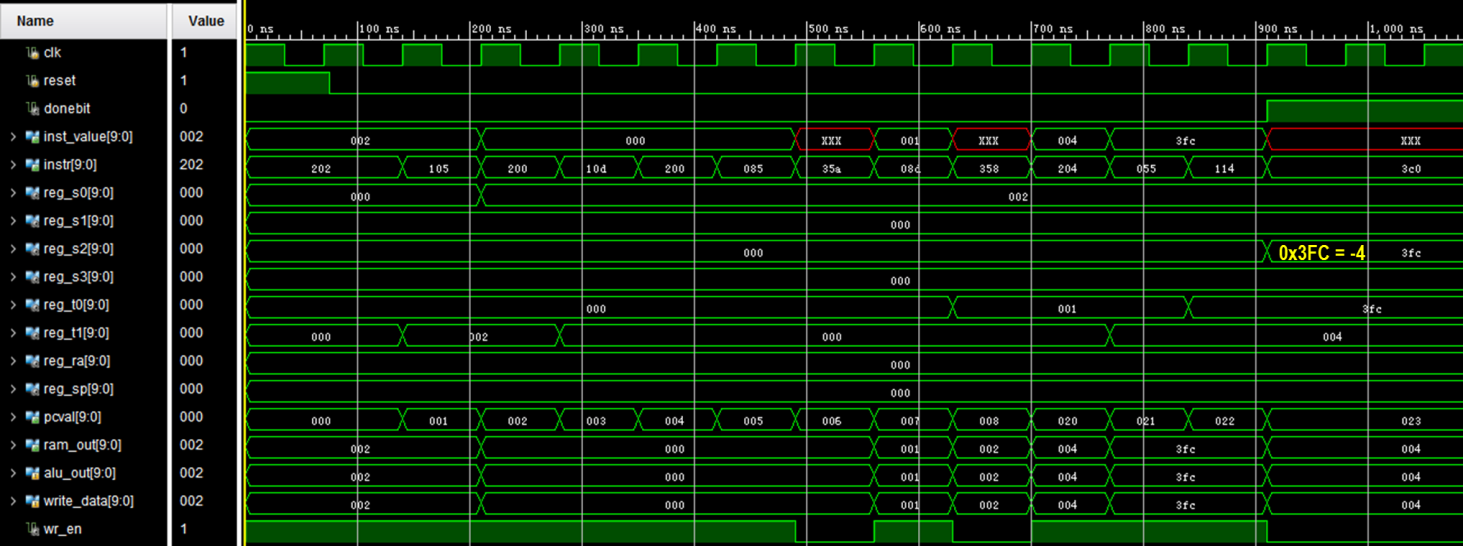
The simulation result is showed in *Figure9-14*. The calculation is correct.

* **X =2, Y=0**



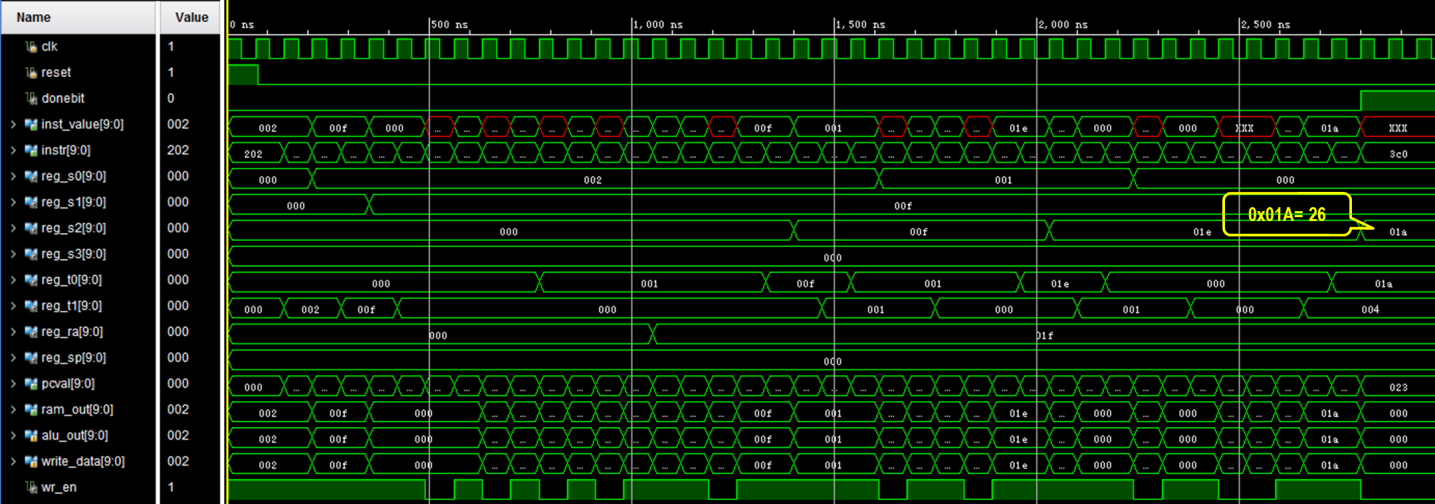
Figuire9 Simulation Result of X=2, Y=0(0x3FC=-4)

* **X =0, Y=2**



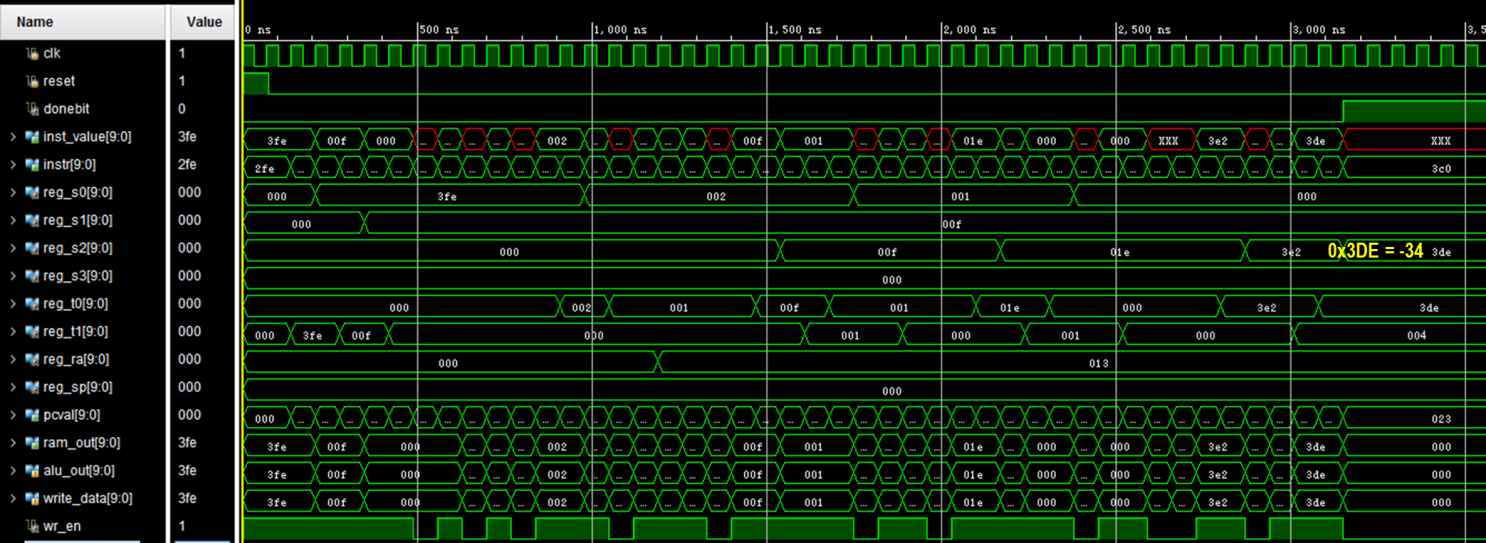
Figuire10 Simulation Result of X=0, Y=2 (0x3FC=-4)

* **X =2, Y=15**



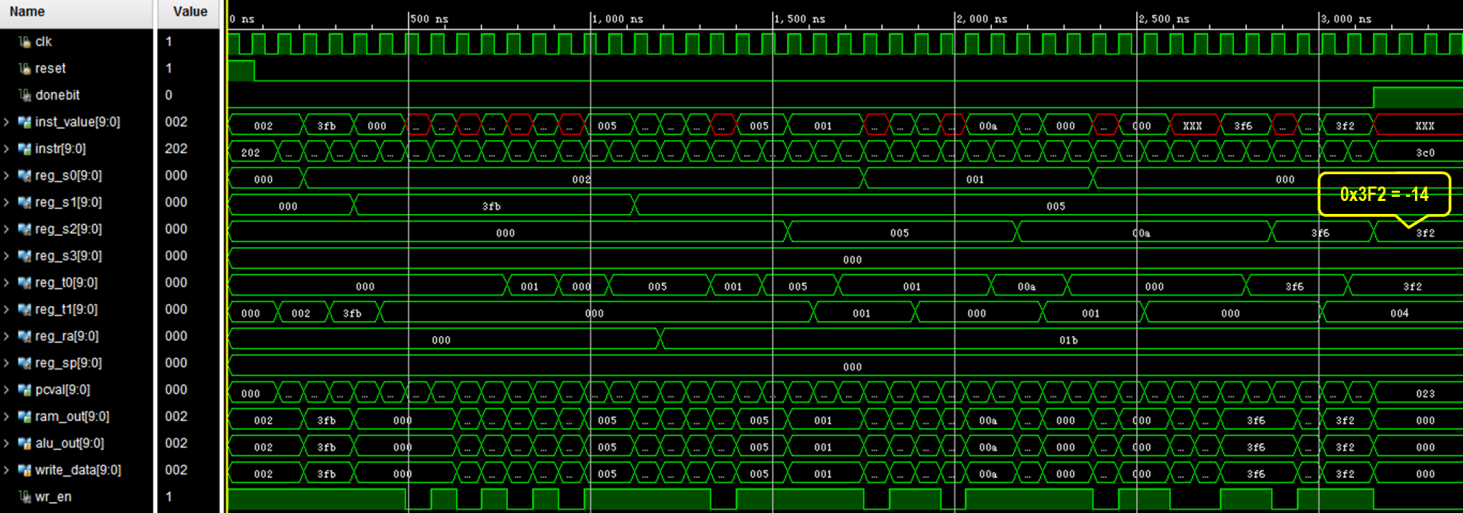
Figuire11 Simulation Result of X=2, Y=15 (0x1A=26)

* **X =-2, Y=15**



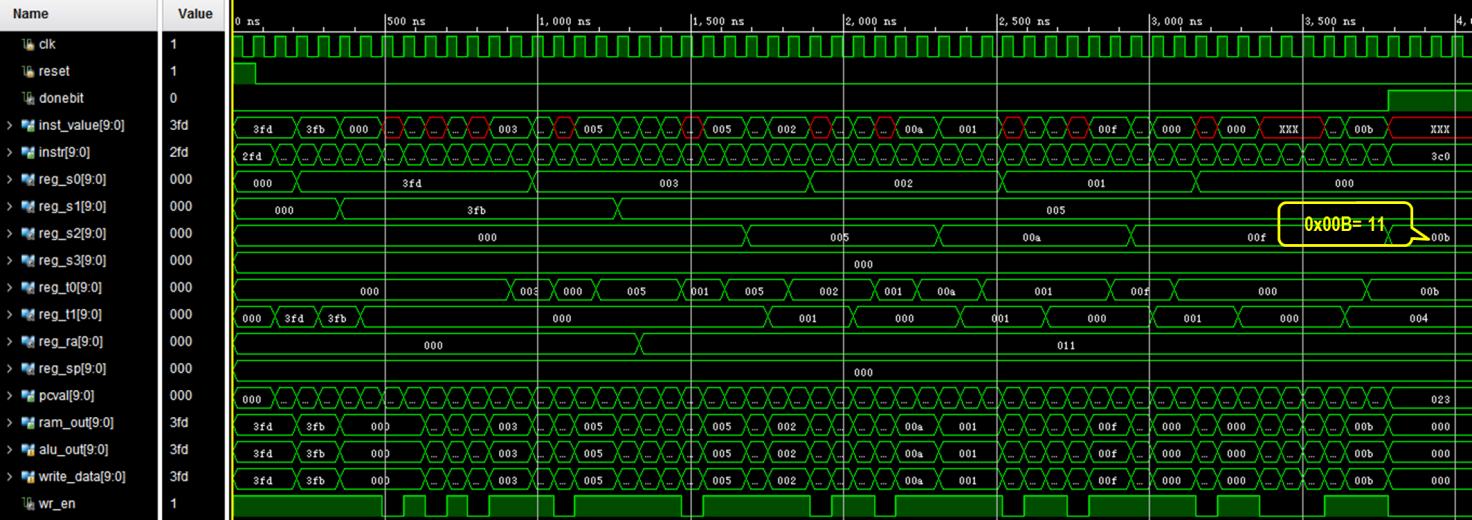
Figuire12 Simulation Result of X=-2, Y=15 (0x3DE=-34)

* **X =2, Y=-5**



Figuire13 Simulation Result of X=2, Y=-5 (0x3F2=-14)

* **X =-3, Y=-5**



Figuire14 Simulation Result of X=-3, Y=-5 (0x00B=11)

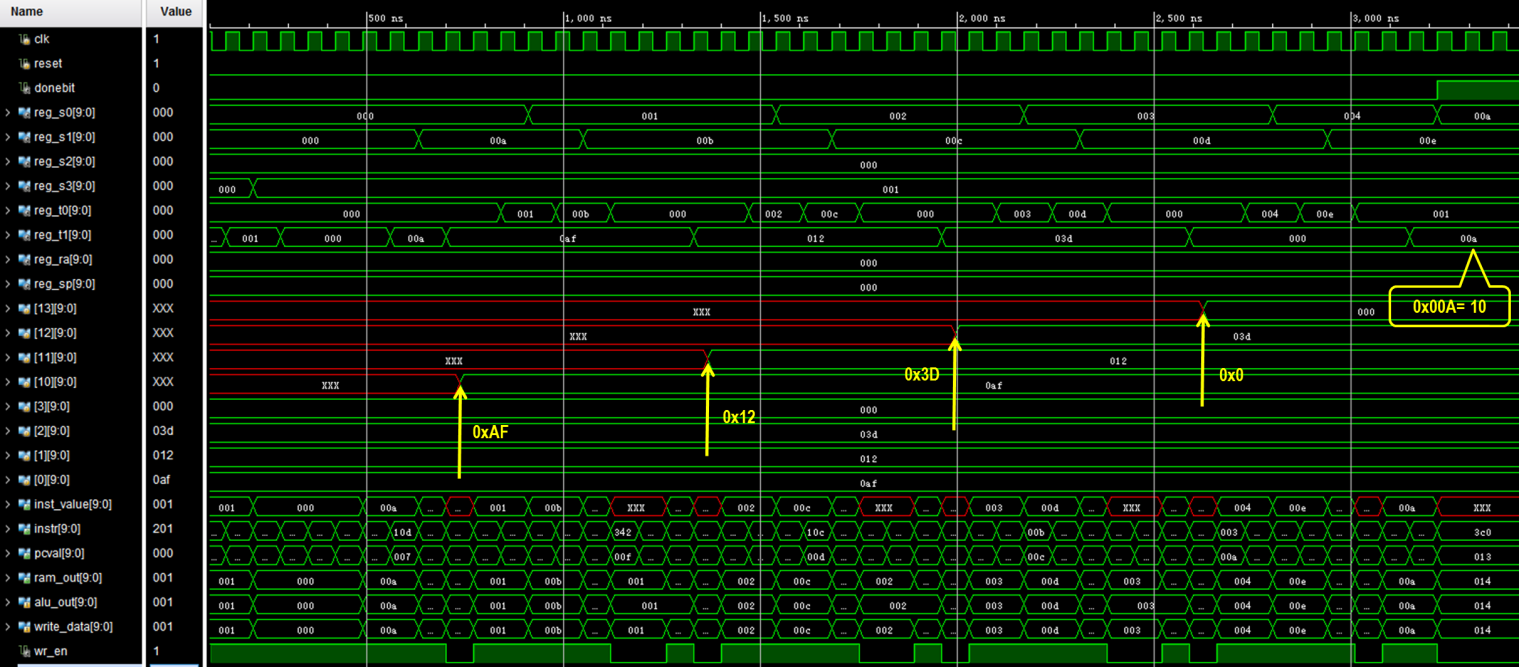
1. **Moving Array in Memory**

The simulation specification and result is showed as *Table8*.

Table8. Moving Array Simulation Specification and Result

| **No** | **Original Address** | **Value** | **Target Address** | **Expected Value** | **Result** |
| --- | --- | --- | --- | --- | --- |
| 1 | 0x0 | 0xAF | 0xA | 0xAF | OK |
| 2 | 0x1 | 0x12 | 0xB | 0x12 |
| 3 | 0x2 | 0x3D | 0xC | 0x3D |
| 4 | 0x3 | 0x00 | 0xD | 0x00 |

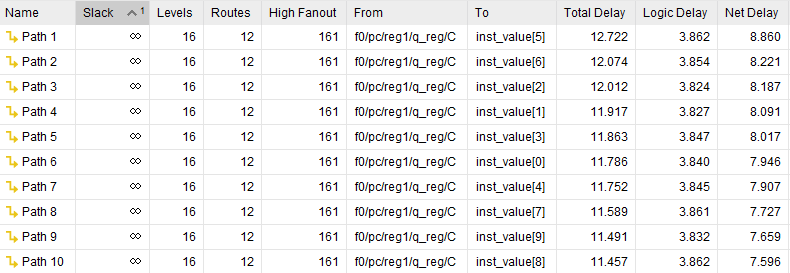
The simulation result is showed in *Figure15*. The data is moved correctly.



Figuire15 Simulation Result of Moving Array

### Implementation

The timing report is showed in *Figure16*. The longest path is from ***f0/pc/reg1/q\_reg/C***  to ***instr\_value[5]*** whose delay is **12.722ns**. So, the minimum clock period time is **12.722ns**.



Figuire16 Timing Report

The running time is showed in *Table9*.

Table9. Moving Array Simulation Specification and Result

| **No** | **Program** | **Clock Count** | **Clock Period** | **Run Time** |
| --- | --- | --- | --- | --- |
| 1 | bubble sort | 749 | 12.722ns | 9528.778ns |
| 2 | X\*Y-4 | 11 | 139.942ns |
| 54 | 686.988ns |
| 3 | moving array | 47 | 597.934ns |