《数字电路》课程实验

实验2：CPU设计

----实验过程与结果分析报告

选课编号：XX 姓名（邓皓宇）选课号（XX）

选课编号：XX 姓名（熊仔羿）选课号（XX）

信息与通信工程学院

2022年11月

## 实验目的

1.掌握简化CPU的结构组成、模块划分和工作原理。

2.掌握精简指令集的概念，并设计CPU的指令集。

3.用Verilog语言设计编写并调试CPU内部各模块电路，并在此基础上设计完整CPU。

4.掌握激励文件的编写方法，学会验证各个模块功能时序的正确性。

5.掌握外围接口电路的设计方法，并学会利用FPGA开发板的按键、LED等外设构造简单的控制系统。

6.编写汇编程序，并依据设计的指令集编译为对应的机器码，控制CPU实现特定功能。

## 二、实验内容

任务1：基础模块（ALU、寄存器阵列、寻址单元）的建模及仿真

任务2：数据通路的设计

任务3：控制通路的设计

任务4：CPU的整合及验证

任务5：XXXXXXXXXXX（如果有做选做内容可以填入）

## 三、实验设备

软件： PC 机操作系统 WinXP 或 Win7、 Quartus II 13.1、 ModelSim-Altera 10.1d

硬件： DE1-SOC 实验板，计算机一台

仪器：示波器，逻辑分析仪

## 四、实验原理

任务1

**目标：** **基础模块（ALU、寄存器阵列、寻址单元）的建模及仿真**

**任务计划：**

课前预习任务：算术逻辑单元模块（ArithmeticUnit.v）设计与仿真。 （检查）  
课上实验任务：寄存器阵列模块（RegisterFile.v）设计与仿真。 （检查）

**实验原理：**

任务2

**目标：**

**任务计划：**

课前预习任务： 寻址单元（AddressLogic.v， ProgramCounter.v） 设计与仿真。  
（检查）  
课上实验任务：数据通路（DataPath.v）整合。 （检查）

**实验原理：**

任务3

**目标：**

**任务计划：**

课前预习任务：

课上实验任务：（检查）

**实验原理：**

任务4

**目标：**

**任务计划：**

课前预习任务：

课上实验任务：（检查）

**实验原理：**

任务5

**目标：**

**任务计划：**

课前预习任务：

课上实验任务：（检查）

**实验原理：**

自行按上诉格式添加任务3、4、5……

## 五、实验步骤

**任务1的步骤如下：**

1. 第一步

使用QuartusⅡ13.1打开工程文件，并且打开ArithmeticUnit.v和RegisterFile.v文件，书写Verilog代码

1. 第二步

写完代码更换顶层文件，之后对整个工程进行编译，并且生成Testbench(.vt)文件

1. 第三步

编写Testbench，并且配置好仿真设置

1. 第四步

启动Modelsim软件，并且观察仿真现象。

1. XXXXXXXXX（自行添加）

**任务2的步骤如下：**

1. 第一步

使用QuartusⅡ13.1打开工程文件，并且打开AddressLogic.v， ProgramCounter.v文件，书写Verilog代码

1. 第二步

写完代码更换顶层文件，之后对整个工程进行编译，并且生成Testbench(.vt)文件

1. 第三步

编写Testbench，并且配置好仿真设置

1. 第四步
2. 启动Modelsim软件，并且观察仿真现象。
3. XXXXXXXXX（自行添加）

自行按上诉格式添加任务2、3、4、5……

## 六、实验结果

**任务1**

ALU：

1. verilog关键代码（带注释）

//----------------------------------------------------------------------

//--SAYEH (Simple Architecture Yet Enough Hardware) CPU

//----------------------------------------------------------------------

//Arithmetic Logic Unit (ALU)

`timescale 1 ns /1 ns

//以下为用户的宏定义，以方便操作

`define B15to0H 10'b1000000000

`define AandBH 10'b0100000000

`define AorBH 10'b0010000000

`define notBH 10'b0001000000

`define shlBH 10'b0000100000

`define shrBH 10'b0000010000

`define AaddBH 10'b0000001000

`define AsubBH 10'b0000000100

`define AmulBH 10'b0000000010

`define AcmpBH 10'b0000000001

//模块的声明

module ArithmeticUnit (

A, B,

B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB,

aluout, cin, cout, zout

);

input [15:0] A, B;//两个输入的16位运算数

input B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB;//模式选择

input cin;//C标志位输入

output [15:0] aluout;//数据输出

output cout,zout;//C，Z的标志位输出

//开始声明存储变量

reg [15:0] aluout;

reg cout,zout;

//进行运算

//运算模式判断

always @(

A or B or B15to0 or AandB or AorB or notB or shlB or shrB or AaddB or AsubB or AmulB or AcmpB or cin

)

begin

//清空标志位以及输出

cout = 0; aluout = 0;zout=0;

//模式选择

case ({B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB})

`AcmpBH: begin

aluout = A;

if (A> B) cout = 1; else cout = 0;//进行A和B的大小判断

end

`AaddBH: begin

aluout = A + B + cin;//输出为A + B + cin

end

`AsubBH: begin

aluout = A - B - cin;//输出为A - B - cin

end

`AmulBH: begin

aluout = A[7:0] \* B[7:0];//输出为A[7:0] \* B[7:0]

end

`shlBH: begin

aluout = {B[14:0],B[15]};//循环左移一位

end

`shrBH: begin

aluout = {B[0],B[15:1]};//循环右移一位

end

`notBH: begin

aluout = ~B;//B取反之后输出

end

`B15to0H: begin

aluout = B;//B直接输出

end

`AandBH: begin

aluout = A & B;//A和B按位取与并且输出

end

`AorBH: begin

aluout = A | B;//A和B按位取或并且输出

end

//其他意外模式选择则输出0

default: aluout = 0;

endcase

if (aluout == 0) zout = 1'b1;

end

endmodule

1. testbench测试文件

// Copyright (C) 1991-2013 Altera Corporation

// Your use of Altera Corporation's design tools, logic functions

// and other software and tools, and its AMPP partner logic

// functions, and any output files from any of the foregoing

// (including device programming or simulation files), and any

// associated documentation or information are expressly subject

// to the terms and conditions of the Altera Program License

// Subscription Agreement, Altera MegaCore Function License

// Agreement, or other applicable license agreement, including,

// without limitation, that your use is for the sole purpose of

// programming logic devices manufactured by Altera and sold by

// Altera or its authorized distributors. Please refer to the

// applicable agreement for further details.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// This file contains a Verilog test bench template that is freely editable to

// suit user's needs .Comments are provided in each section to help the user

// fill out necessary details.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Generated on "11/24/2022 13:48:23"

// Verilog Test Bench template for design : ArithmeticUnit

//

// Simulation tool : ModelSim-Altera (Verilog)

`timescale 1 ps/ 1 ps

module ArithmeticUnit\_vlg\_tst();

// constants

// general purpose registers

reg eachvec;

// test vector input registers

reg [15:0] A;

reg AaddB;

reg AandB;

reg AcmpB;

reg AmulB;

reg AorB;

reg AsubB;

reg [15:0] B;

reg B15to0;

reg cin;

reg notB;

reg shlB;

reg shrB;

// wires

wire [15:0] aluout;

wire cout;

wire zout;

// assign statements (if any)

ArithmeticUnit i1 (

// port map - connection between master ports and signals/registers

.A(A),

.AaddB(AaddB),

.AandB(AandB),

.AcmpB(AcmpB),

.AmulB(AmulB),

.AorB(AorB),

.AsubB(AsubB),

.B(B),

.B15to0(B15to0),

.aluout(aluout),

.cin(cin),

.cout(cout),

.notB(notB),

.shlB(shlB),

.shrB(shrB),

.zout(zout)

);

initial

begin

A = 16'b1010111100001001;

B = 16'b0101111100001111;

cin = 0;

#10 B15to0 = 0;AandB = 0;AorB = 0;notB = 0;shlB = 0;shrB = 0;AaddB = 0;AsubB = 0;AmulB = 0;AcmpB = 0;

#10 B15to0 = 1;AandB = 0;AorB = 0;notB = 0;shlB = 0;shrB = 0;AaddB = 0;AsubB = 0;AmulB = 0;AcmpB = 0;

#10 B15to0 = 0;AandB = 1;AorB = 0;notB = 0;shlB = 0;shrB = 0;AaddB = 0;AsubB = 0;AmulB = 0;AcmpB = 0;

#10 B15to0 = 0;AandB = 0;AorB = 1;notB = 0;shlB = 0;shrB = 0;AaddB = 0;AsubB = 0;AmulB = 0;AcmpB = 0;

#10 B15to0 = 0;AandB = 0;AorB = 0;notB = 1;shlB = 0;shrB = 0;AaddB = 0;AsubB = 0;AmulB = 0;AcmpB = 0;

#10 B15to0 = 0;AandB = 0;AorB = 0;notB = 0;shlB = 1;shrB = 0;AaddB = 0;AsubB = 0;AmulB = 0;AcmpB = 0;

#10 B15to0 = 0;AandB = 0;AorB = 0;notB = 0;shlB = 0;shrB = 1;AaddB = 0;AsubB = 0;AmulB = 0;AcmpB = 0;

#10 B15to0 = 0;AandB = 0;AorB = 0;notB = 0;shlB = 0;shrB = 0;AaddB = 1;AsubB = 0;AmulB = 0;AcmpB = 0;

#10 B15to0 = 0;AandB = 0;AorB = 0;notB = 0;shlB = 0;shrB = 0;AaddB = 0;AsubB = 1;AmulB = 0;AcmpB = 0;

#10 B15to0 = 0;AandB = 0;AorB = 0;notB = 0;shlB = 0;shrB = 0;AaddB = 0;AsubB = 0;AmulB = 1;AcmpB = 0;

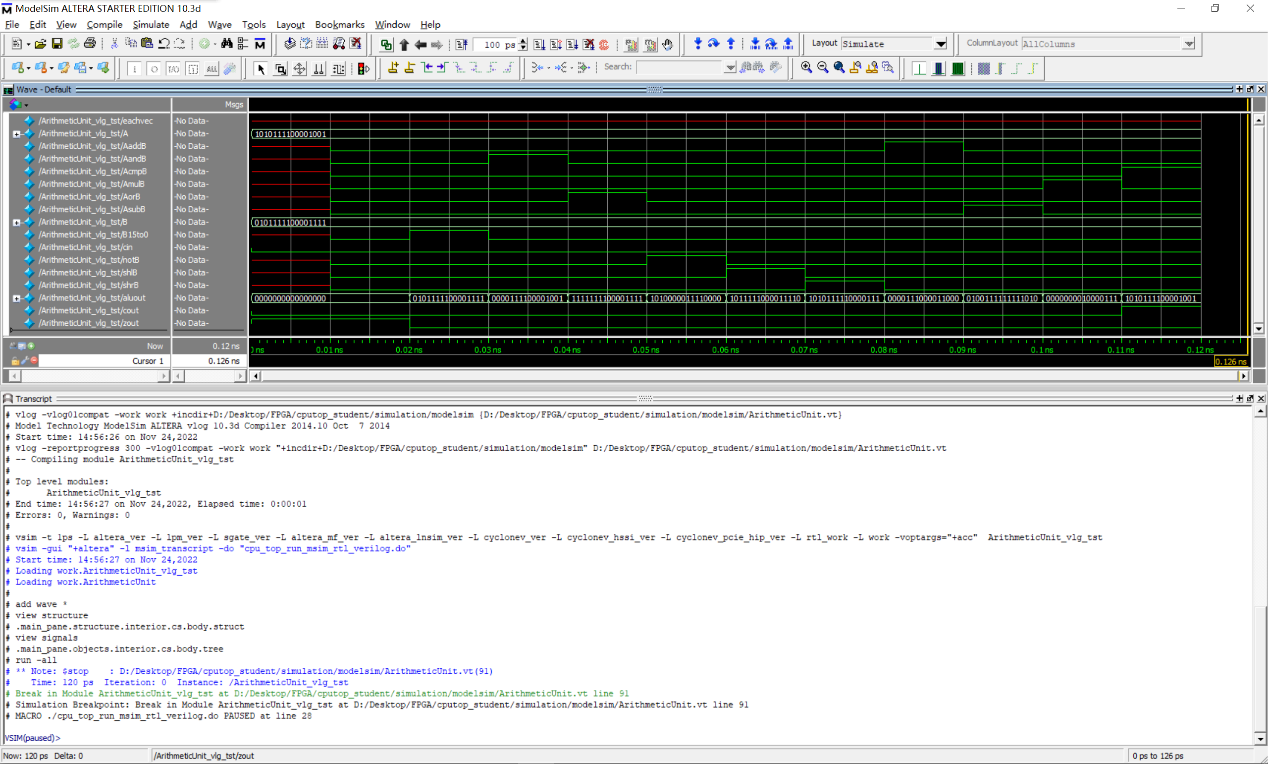
#10 B15to0 = 0;AandB = 0;AorB = 0;notB = 0;shlB = 0;shrB = 0;AaddB = 0;AsubB = 0;AmulB = 0;AcmpB = 1;

#10 $stop;

end

endmodule

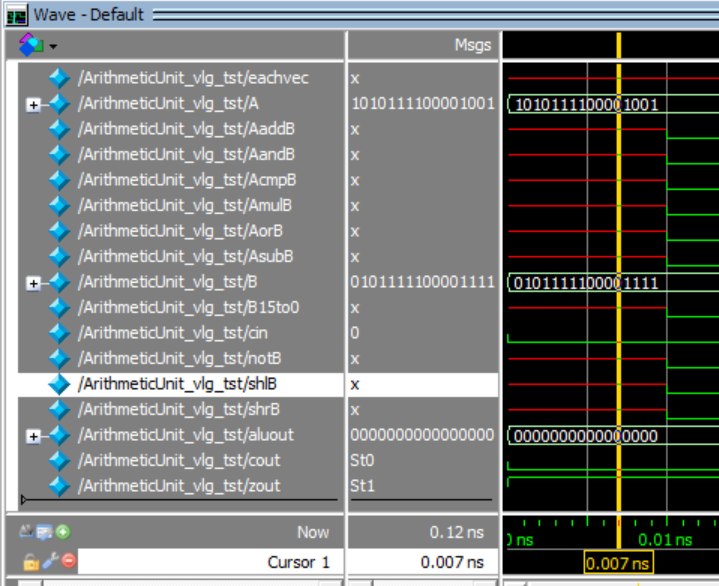
3、仿真结果截图（带文字分析）



进行仿真后的波形如图所示

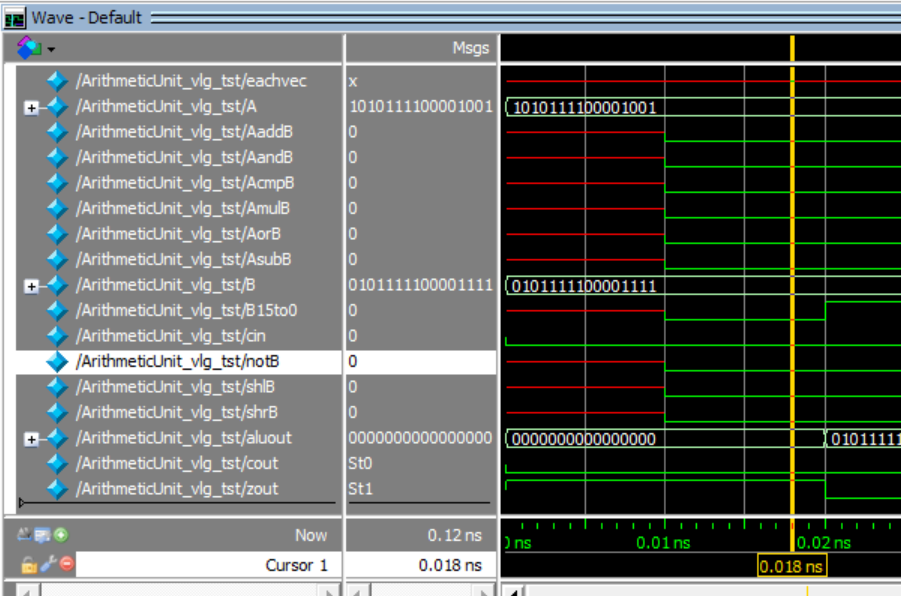
下面开始进行分析

1. 初始化状态



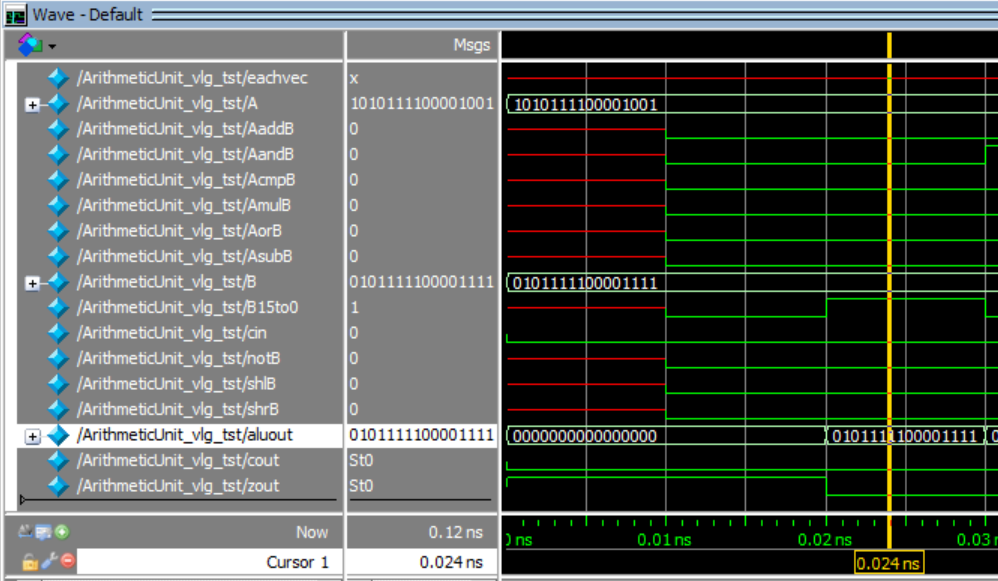
如图所示，在初始化状态时没有模式选择输入，只有对于A，B的赋值以及状态位的初始化。所以其他的线都是红色，表示未知态x。

1. 模式初始化阶段



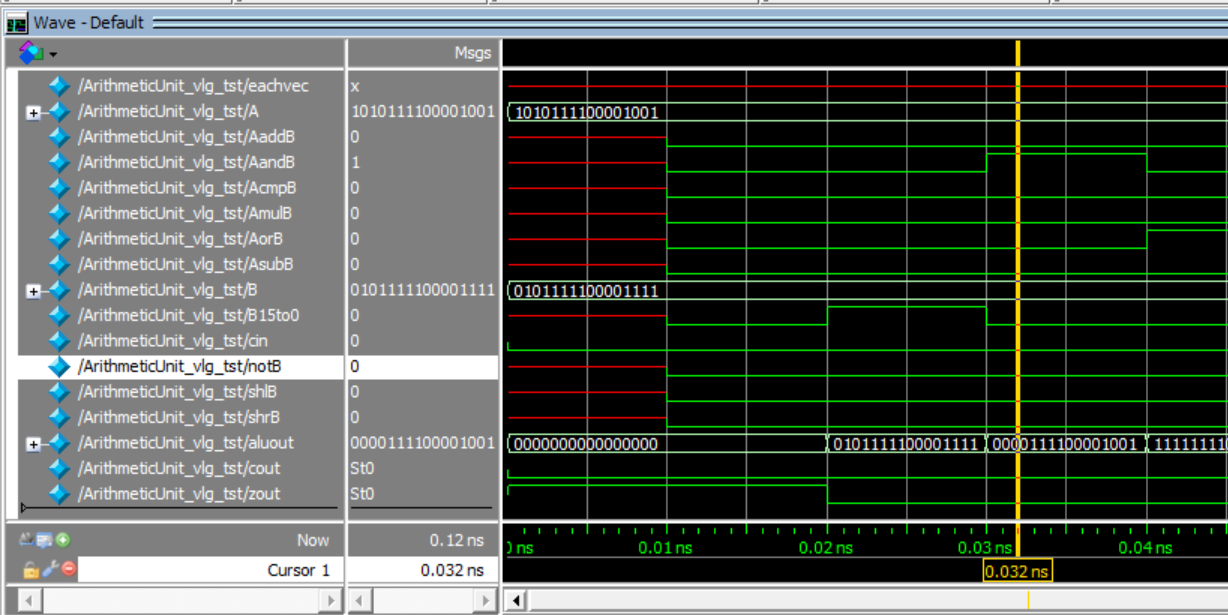
此时对于模式选择输入进行了初始化，即先全部置0，此时无输出

1. 直接输出B模式



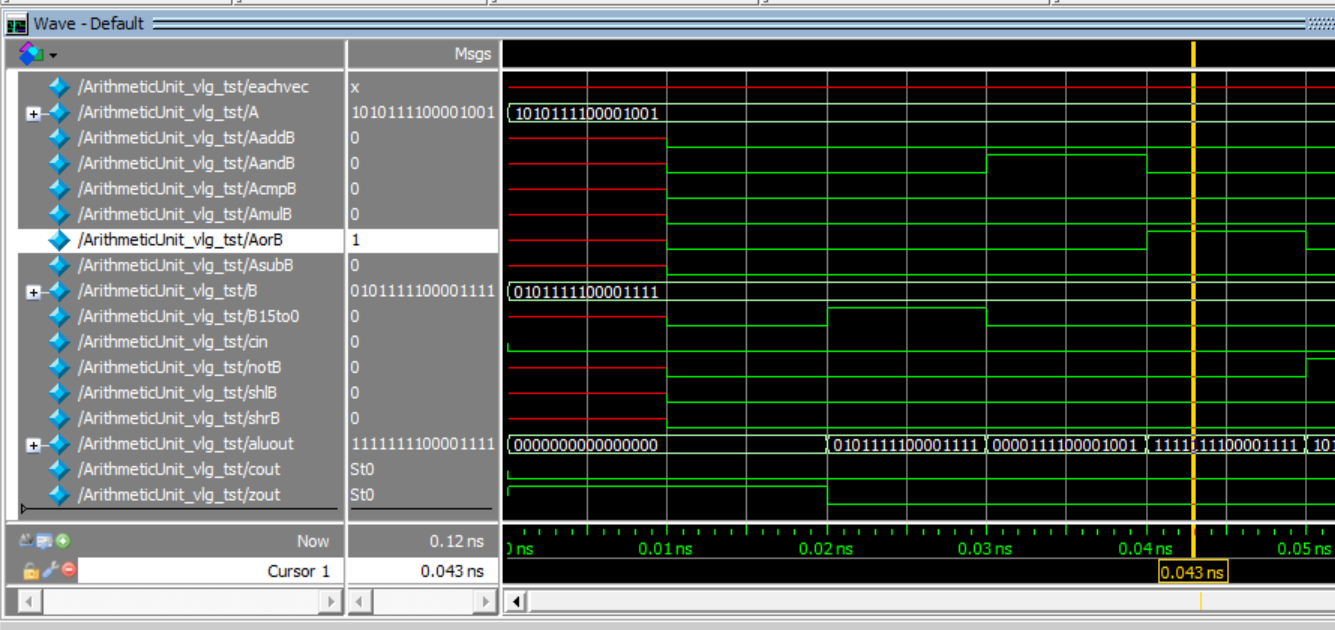
可以看到，现在输出值就是B[15:0]，其他模式为0

1. A和B按位与模式



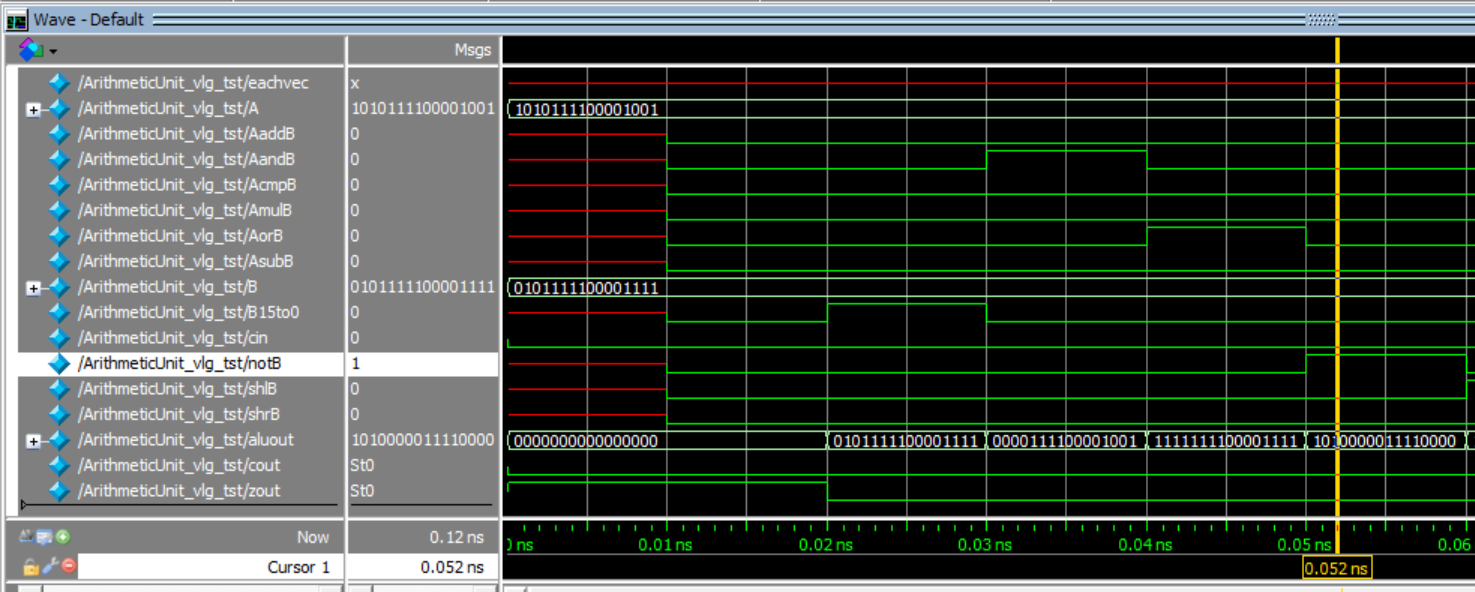
现在可以看到选取的是A与B按位与模式，输出则是A和B按位与运算的结果

1. A和B按位或模式



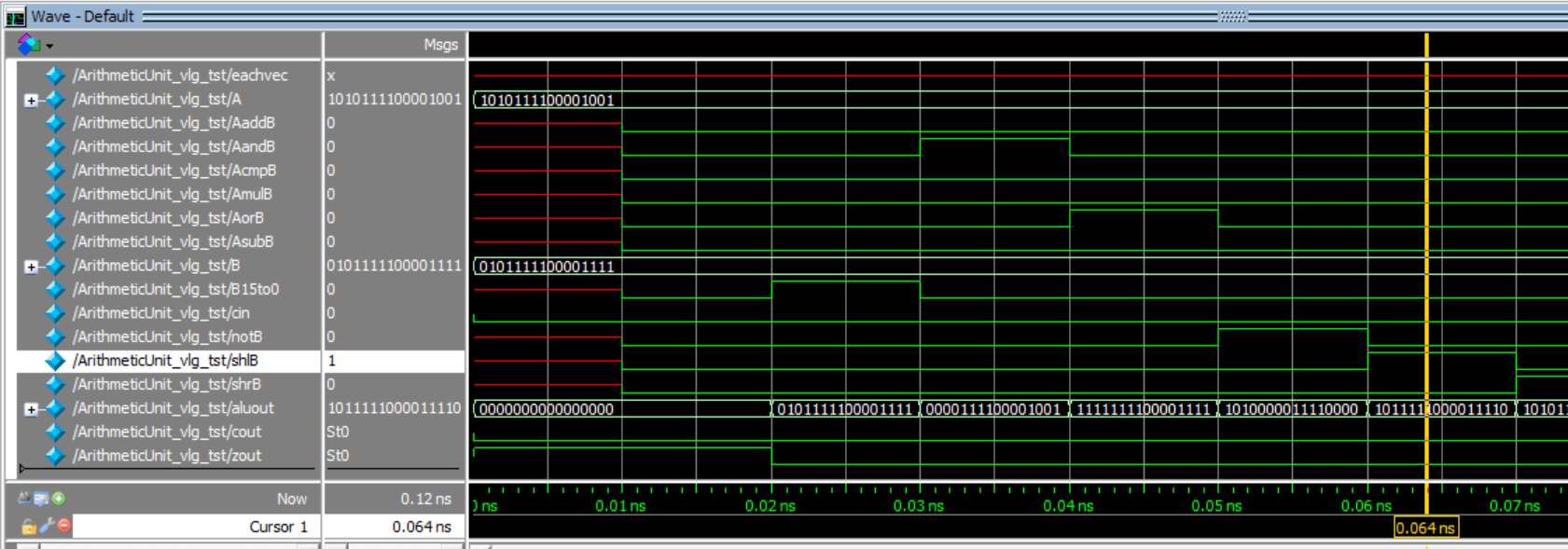
现在可以看到选取的是A与B按位或模式，输出则是A和B按位或运算的结果

1. A与B按位取反模式



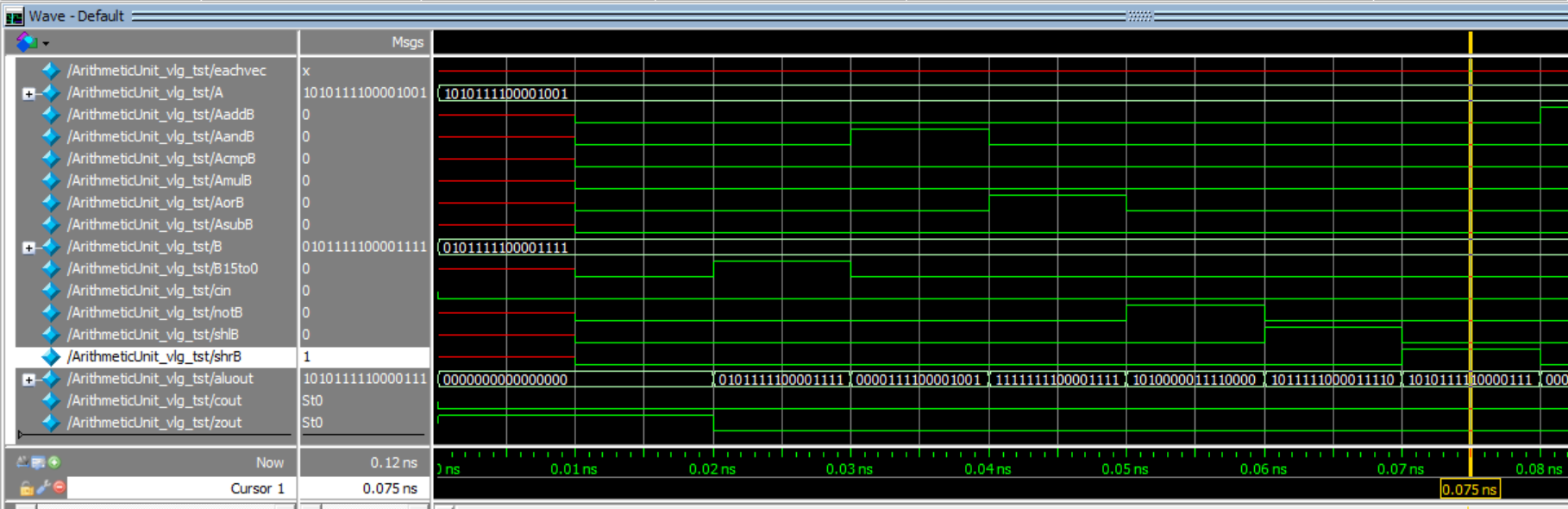
现在可以看到选取的是B按位取反模式，输出则是B按位取反运算的结果

1. 循环左移模式



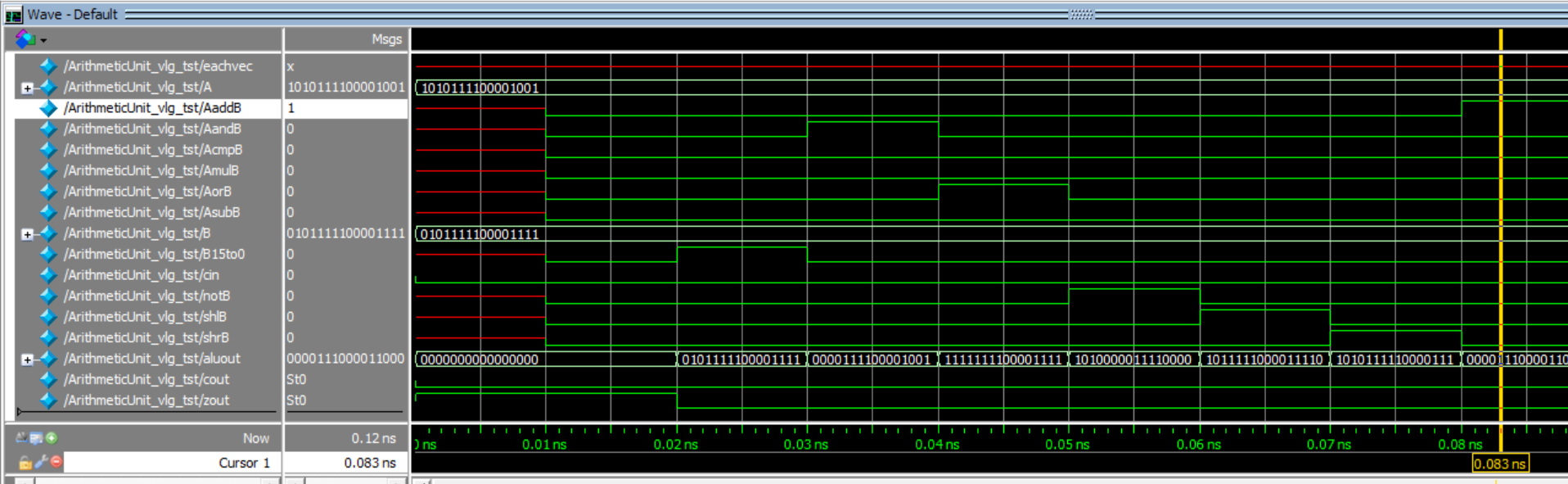
现在可以看到选取的是循环左移模式，输出则是B循环左移运算的结果

1. 循环右移模式



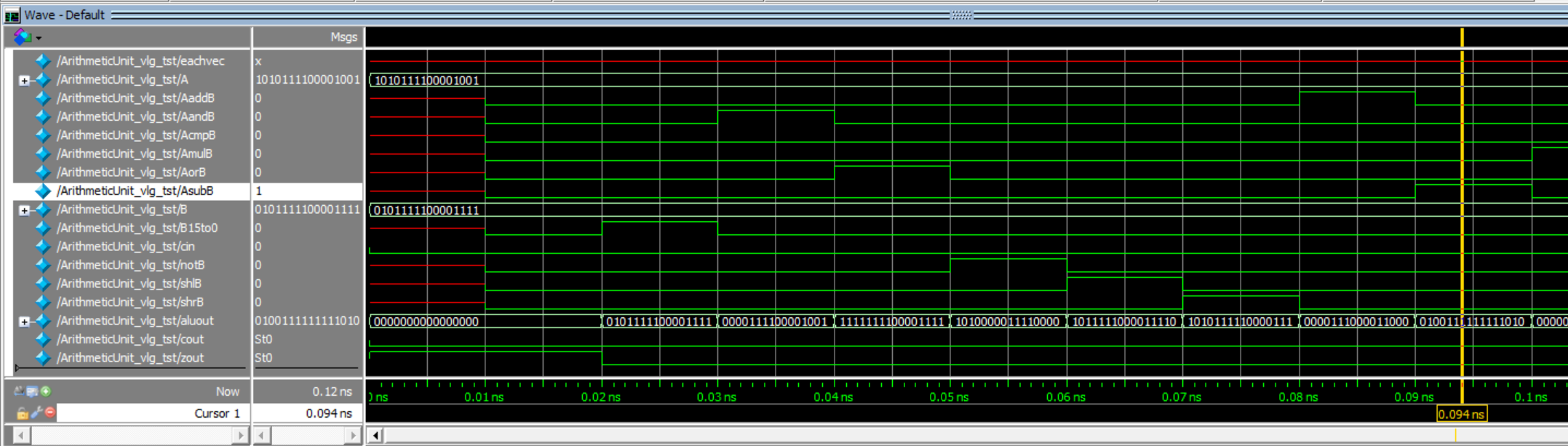
现在可以看到选取的是循环右移模式，输出则是B循环右移运算的结果

1. A+B模式



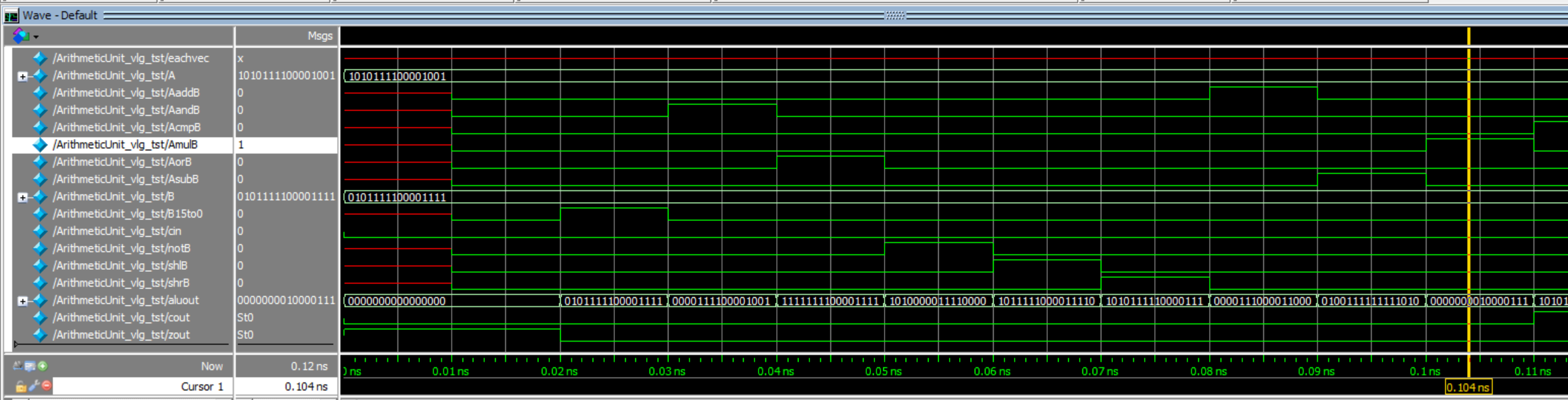
现在可以看到选取的是A+B模式，输出则是A+B运算的结果

1. A-B模式



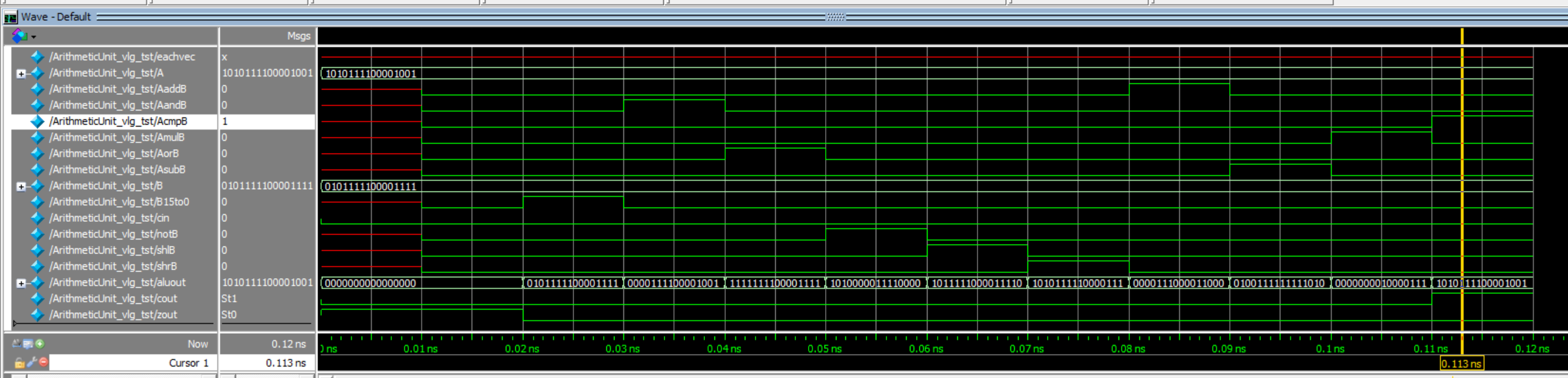
现在可以看到选取的是A-B模式，输出则是A-B运算的结果

1. A\*B模式



现在可以看到选取的是A\*B模式，输出则是A\*B运算的结果

1. A和B比较模式



现在可以看到选取的是A和B进行比较模式，输出则是A，count现在是１，符合预期

至此，仿真结束。

寄存器阵列：

1. verilog关键代码（带注释）

//----------------------------------------------------------------------

//--SAYEH (Simple Architecture Yet Enough Hardware) CPU

//----------------------------------------------------------------------

// Register File

`timescale 1 ns /1 ns

module RegisterFile (

input [15:0] in,

input clk, RFLwrite, RFHwrite,

input [1:0] Laddr, Raddr,

output [15:0] Lout, Rout

);

reg [15:0] MemoryFile [0:3];

reg [15:0] data\_temp;

//将指令寄存器当中的数据传输给ALU

assign Lout = MemoryFile [Laddr];//输出到目的寄存器

assign Rout = MemoryFile [Raddr];//输出到源寄存器

always @(negedge clk) begin

data\_temp = MemoryFile [Laddr];//先读取数据

//开始写数据

//低8位有效

if (RFLwrite == 1) begin

data\_temp[7:0] = in[7:0];//低 8 位写有效，Laddr内存储的数据低 8 位=输入数据 in 的低 8 位;

end

else begin

data\_temp = data\_temp; //不变

end

//高8位有效

if (RFHwrite == 1) begin

data\_temp[15:8] = in[15:8];//高 8 位写有效，Laddr内存储的数据高 8 位=输入数据 in 的高 8 位;

end

else begin

data\_temp = data\_temp; //不变

end

MemoryFile [Laddr] = data\_temp;//输出到目的寄存器

end

endmodule

1. testbench测试文件

`timescale 1 ps/ 1 ps

module RegisterFile\_vlg\_tst();

// constants

// general purpose registers

reg eachvec;

// test vector input registers

reg [1:0] Laddr;

reg RFHwrite;

reg RFLwrite;

reg [1:0] Raddr;

reg clk;

reg [15:0] in;

// wires

wire [15:0] Lout;

wire [15:0] Rout;

// assign statements (if any)

RegisterFile i1 (

// port map - connection between master ports and signals/registers

.Laddr(Laddr),

.Lout(Lout),

.RFHwrite(RFHwrite),

.RFLwrite(RFLwrite),

.Raddr(Raddr),

.Rout(Rout),

.clk(clk),

.in(in)

);

//以下是testbench

parameter PERIOD=10;//宏定义

initial

begin

clk=1'b0;

forever #(PERIOD/2)

clk=~clk;//不断反转clk

end

initial

begin

in = 16'b1111000011110000;

//标志位清零

RFHwrite = 0;

RFLwrite = 0;

#10;

in = 16'b1010101010101010;Laddr = 2'b00;Raddr = 2'b00;RFHwrite = 1'b1;RFLwrite = 1'b1;

#10;

in = 16'b1111000011110000;Laddr = 2'b00;Raddr = 2'b00;RFHwrite = 1'b0;RFLwrite = 1'b0;

#10;

in = 16'b1111000011110000;Laddr = 2'b00;Raddr = 2'b00;RFHwrite = 1'b0;RFLwrite = 1'b1;

#10;

in = 16'b1111000011110000;Laddr = 2'b00;Raddr = 2'b00;RFHwrite = 1'b1;RFLwrite = 1'b0;

#10;

in = 16'b1111000011110000;Laddr = 2'b00;Raddr = 2'b00;RFHwrite = 1'b1;RFLwrite = 1'b1;

#10;

in = 16'b0000000000000000;Laddr = 2'b01;Raddr = 2'b01;RFHwrite = 1'b1;RFLwrite = 1'b1;

#10;

in = 16'b1111000011110000;Laddr = 2'b01;Raddr = 2'b01;RFHwrite = 1'b0;RFLwrite = 1'b0;

#10;

in = 16'b1111000011110000;Laddr = 2'b01;Raddr = 2'b01;RFHwrite = 1'b0;RFLwrite = 1'b1;

#10;

in = 16'b1111000011110000;Laddr = 2'b01;Raddr = 2'b01;RFHwrite = 1'b1;RFLwrite = 1'b0;

#10;

in = 16'b1111000011110000;Laddr = 2'b01;Raddr = 2'b01;RFHwrite = 1'b1;RFLwrite = 1'b1;

#10;

in = 16'b1111111111111111;Laddr = 2'b10;Raddr = 2'b10;RFHwrite = 1'b1;RFLwrite = 1'b1;

#10;

in = 16'b1111000011110000;Laddr = 2'b10;Raddr = 2'b10;RFHwrite = 1'b0;RFLwrite = 1'b0;

#10;

in = 16'b1111000011110000;Laddr = 2'b10;Raddr = 2'b10;RFHwrite = 1'b0;RFLwrite = 1'b1;

#10;

in = 16'b1111000011110000;Laddr = 2'b10;Raddr = 2'b10;RFHwrite = 1'b1;RFLwrite = 1'b0;

#10;

in = 16'b1111000011110000;Laddr = 2'b10;Raddr = 2'b10;RFHwrite = 1'b1;RFLwrite = 1'b1;

#10;

in = 16'b0101010101010101;Laddr = 2'b11;Raddr = 2'b11;RFHwrite = 1'b1;RFLwrite = 1'b1;

#10;

in = 16'b1111000011110000;Laddr = 2'b11;Raddr = 2'b11;RFHwrite = 1'b0;RFLwrite = 1'b0;

#10;

in = 16'b1111000011110000;Laddr = 2'b11;Raddr = 2'b11;RFHwrite = 1'b0;RFLwrite = 1'b1;

#10;

in = 16'b1111000011110000;Laddr = 2'b11;Raddr = 2'b11;RFHwrite = 1'b1;RFLwrite = 1'b0;

#10;

in = 16'b1111000011110000;Laddr = 2'b11;Raddr = 2'b11;RFHwrite = 1'b1;RFLwrite = 1'b1;

#10;

$stop;

end

endmodule

3、仿真结果截图（带文字分析）

寻址单元：

1、verilog关键代码（带注释）

2、testbench测试文件

3、仿真结果截图（带文字分析）

**任务2**

数据通路：

1、verilog关键代码（带注释）

2、关键代码分析（简述代码设计思路）

**任务3**

控制通路：

1、verilog关键代码（带注释）

2、关键代码分析（简述代码设计思路）

**任务4**

1、汇编代码对应的机器码

2、下板引脚绑定

**任务5**

1、汇编代码对应的机器码

2、下板引脚绑定

自行按上诉格式添加任务5……

## 七、自主设计部分（选做，可加分）

**目标：**

**任务计划：**

**仿真：**

**实验现象：**

## 八、实验中遇到的问题及解决方法

## 九、思考题

## 十、实验分工说明

## 十一、实验感想和建议