# 实验一 模块设计

1. 实验要求

1.1 掌握仿真软件modelsim和mips汇编工具mars的用法。

1.2 掌握Verilog HDL语言设计硬件的基本方法。包括组合逻辑和时序逻辑的描述风格，模块化的设计方法，testbench程序的写法。

1.3 设计单周期cpu的基本模块，包括：pc模块（程序计数器），im模块（指令存储器），gpr模块（通用寄存器），alu模块（算术逻辑单元），dm模块（数据存储器）。

1.4 连接基本模块，构造能执行addu指令的单周期CPU。

1. 实验过程
   1. pc模块
      1. 功能描述

程序计数器，指向指令所在的存储单元。

* + 1. 端口说明

输入：clock, reset, npc，输出：pc

1.clock上升沿有效，reset低电平有效；

2. reset信号有效时，pc复位为0x0000\_3000；采用异步复位。

* + 1. 实现过程

module pc(pc,clock,reset,npc);

output [31:0] pc;

input clock;

input reset;

input [31:0] npc;

reg [31:0] data;

always @(posedge clock or negedge reset)

begin

if (reset == 0)

data <= 32'h00003000;

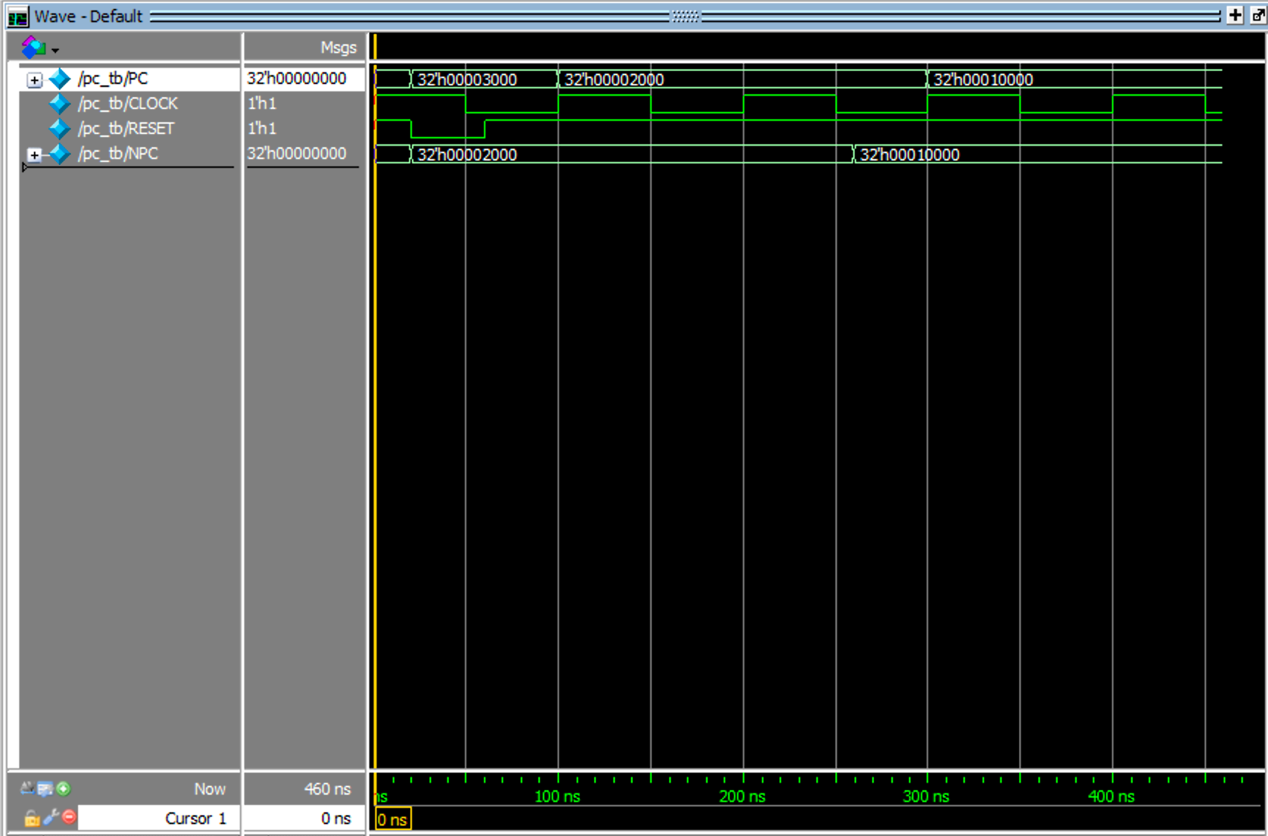
else

data <= npc;

end

assign pc = data;

endmodule



当reset后PC赋初值32’h00003000，之后在每次时钟上升沿更新PC的值为与nPC相同。

* 1. im模块
     1. 功能描述

指令存储器，取指，从pc的地址读出指令。

* + 1. 端口说明

输入：pc，输出：instruction

im模块的输入pc为32位，但指令存储器只有4kB大小，所以取指令时只取pc的低12位作为地址。

* + 1. 实现过程

module im(instruction,pc);

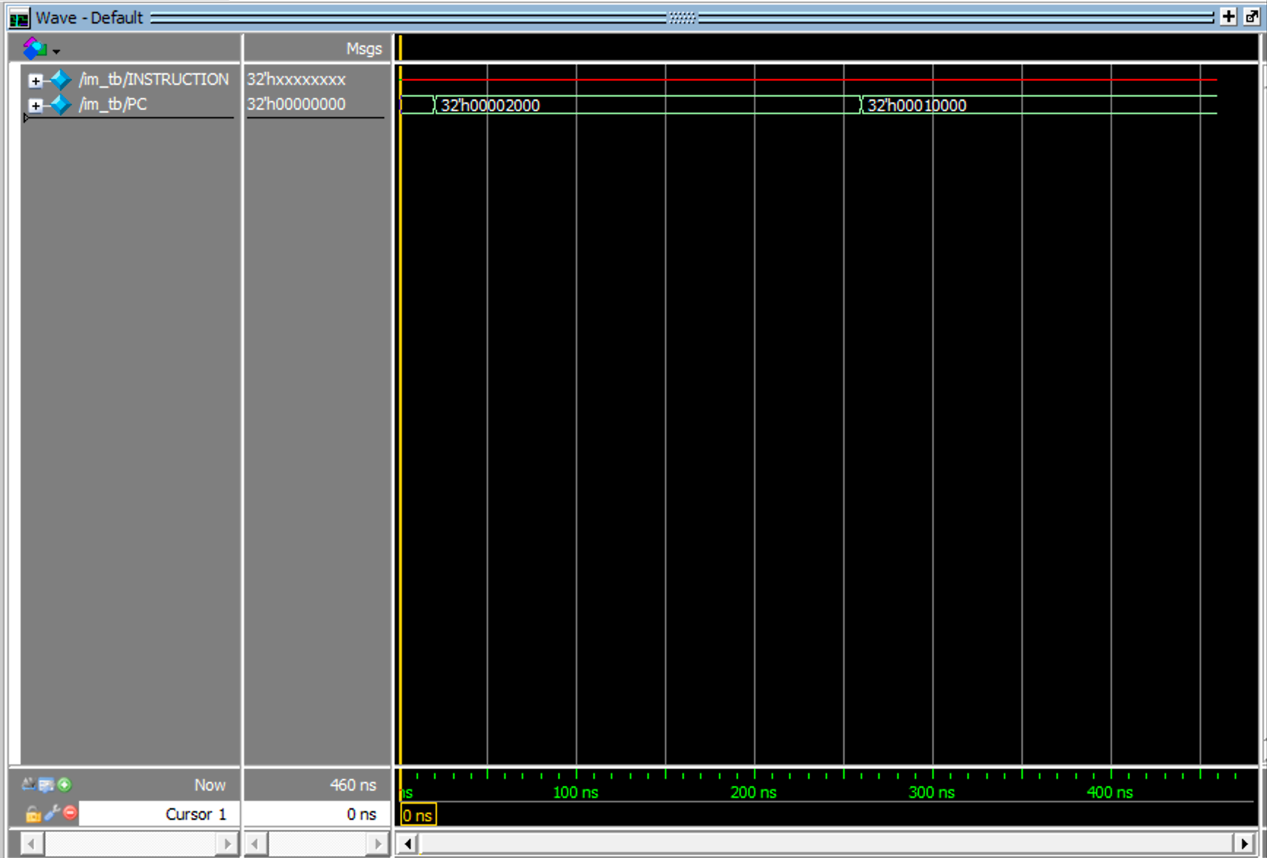
output [31:0] instruction;

input [31:0] pc;

reg [31:0] ins\_memory[1023:0]; //4k指令存储器

assign instruction = ins\_memory[pc[11:0]>>2];

endmodule



从ins\_memory中取指，但由于ins\_memory还没存东西所以读不出来。

* 1. gpr模块
     1. 功能描述

寄存器堆，根据rs和rt读出a和b的值。

* + 1. 端口说明

输入：clock, reg\_write, rs, rt, num\_write, data\_write，输出：a, b。

gp\_registers[0] 永远等于0。

* + 1. 实现过程

module gpr(a,b,clock,reg\_write,num\_write,rs,rt,data\_write);

output [31:0] a;

output [31:0] b;

input clock;

input reg\_write;

input [4:0] rs; //读寄存器1

input [4:0] rt; //读寄存器2

input [4:0] num\_write; //写寄存器

input [31:0] data\_write; //写数据

reg [31:0] gp\_registers[31:0]; //32个寄存器

integer i;

initial

begin

for (i = 0; i < 32; i = i + 1)

gp\_registers[i] = 32'b0;

end

assign a = gp\_registers[rs];

assign b = gp\_registers[rt];

always @(posedge clock)

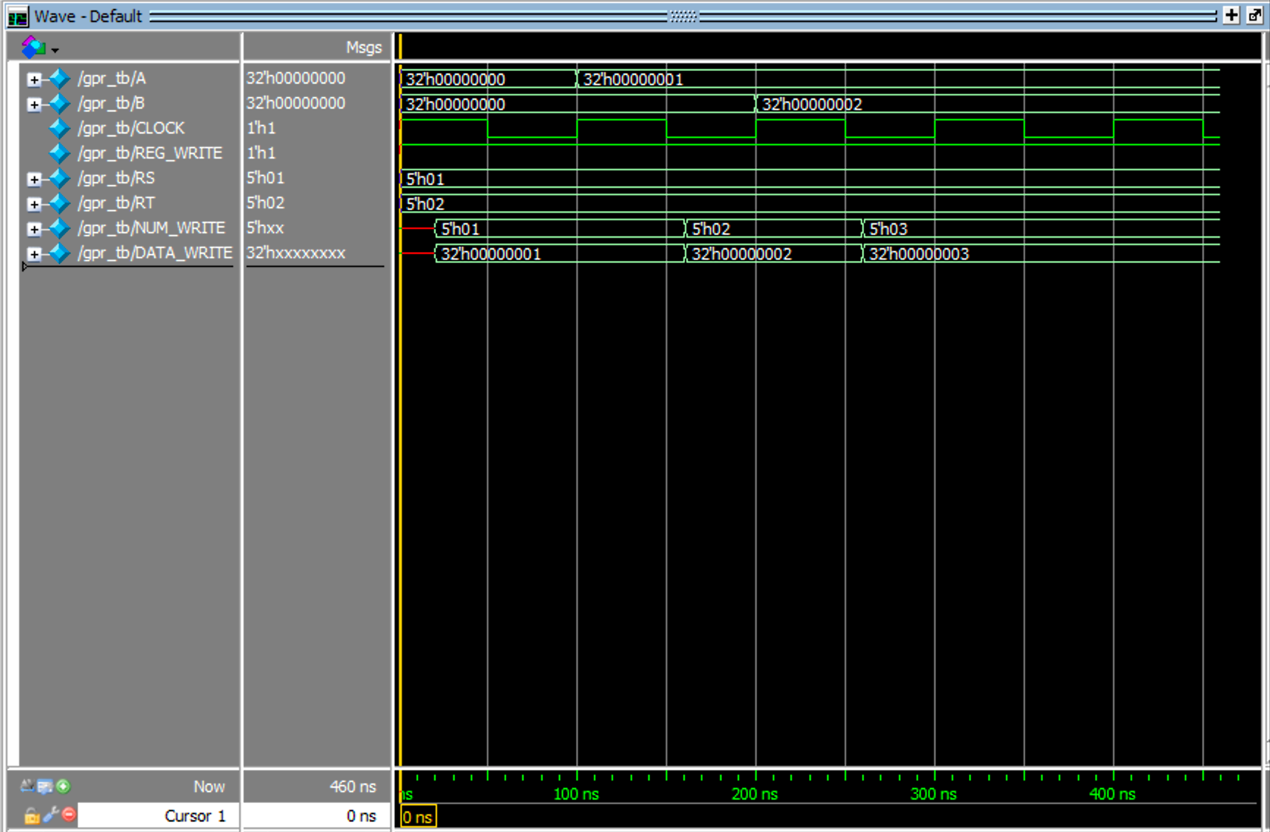
begin

if (reg\_write == 1 && num\_write != 0)

gp\_registers[num\_write] <= data\_write;

end

endmodule



每当时钟上升沿写入数据，这里写入的是gp\_registers[1]=1，gp\_registers[2]=2，从gp\_registers[rs]和gp\_registers[rt]读出a和b的值。

* 1. alu模块
     1. 功能描述

计算用，目前只是实现 + 功能。其他功能和输入输出信号根据需要慢慢添加。

* + 1. 端口说明

输入a和b，输出c。

* + 1. 实现过程

module alu(c,a,b);

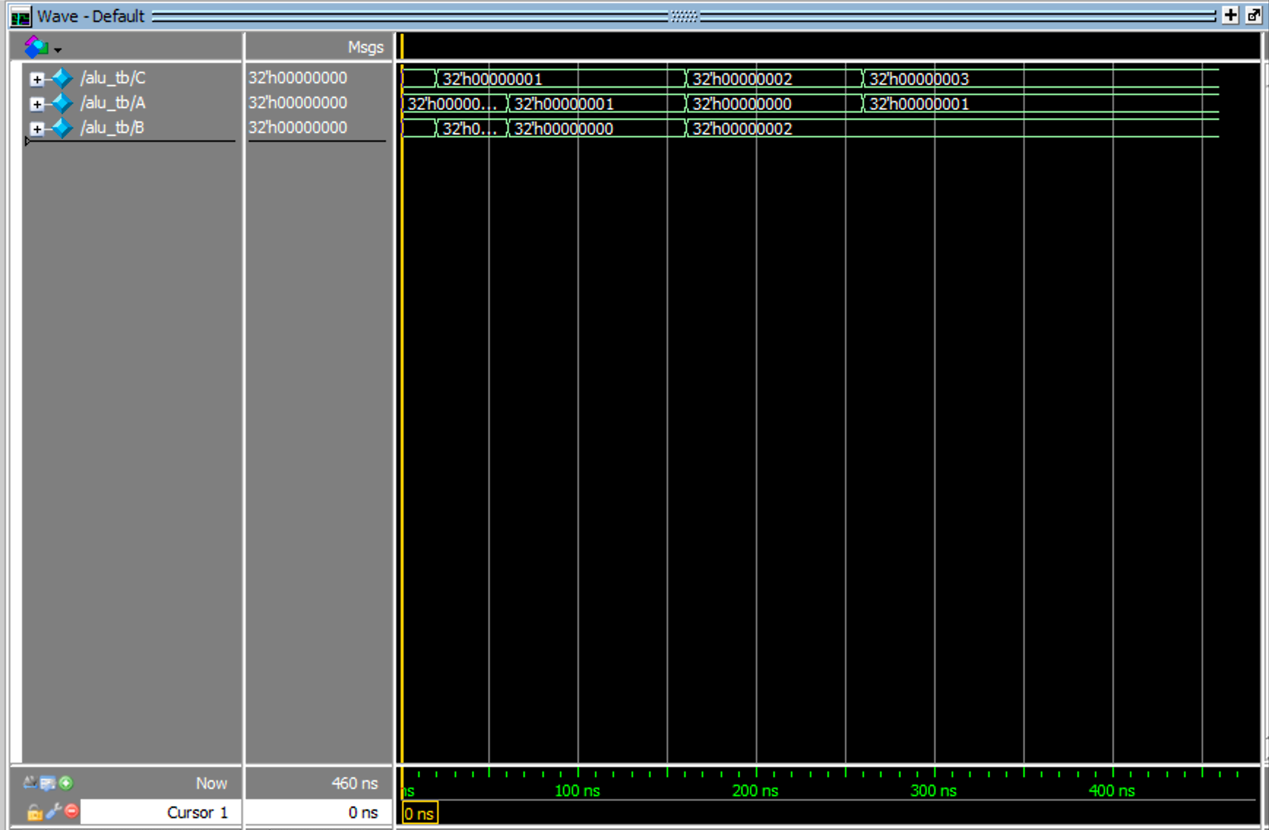
output [31:0] c;

input [31:0] a;

input [31:0] b;

assign c = a + b;

endmodule



计算c = a + b，图中0 + 1 = 1，0 + 2 = 2和1 + 2 = 3均符合预期。

* 1. dm模块
     1. 功能描述

数据存储器，数据存储器模块的功能是建立一个4kB大小的存储器以及读写存储器。

对于读操作，data\_out一直输出address地址处的值。

对于写操作，在clock上升沿且mem\_write信号有效时data\_in被写入address地址处。

* + 1. 端口说明

输入clock, mem\_write, address, data\_in，输出：data\_out。

* + 1. 实现过程

module dm(data\_out,clock,mem\_write,address,data\_in);

output [31:0] data\_out;

input clock;

input mem\_write;

input [31:0] address;

input [31:0] data\_in;

reg [31:0] data\_memory[1023:0]; //4K数据存储器

assign data\_out = data\_memory[address>>2];

always @(posedge clock)

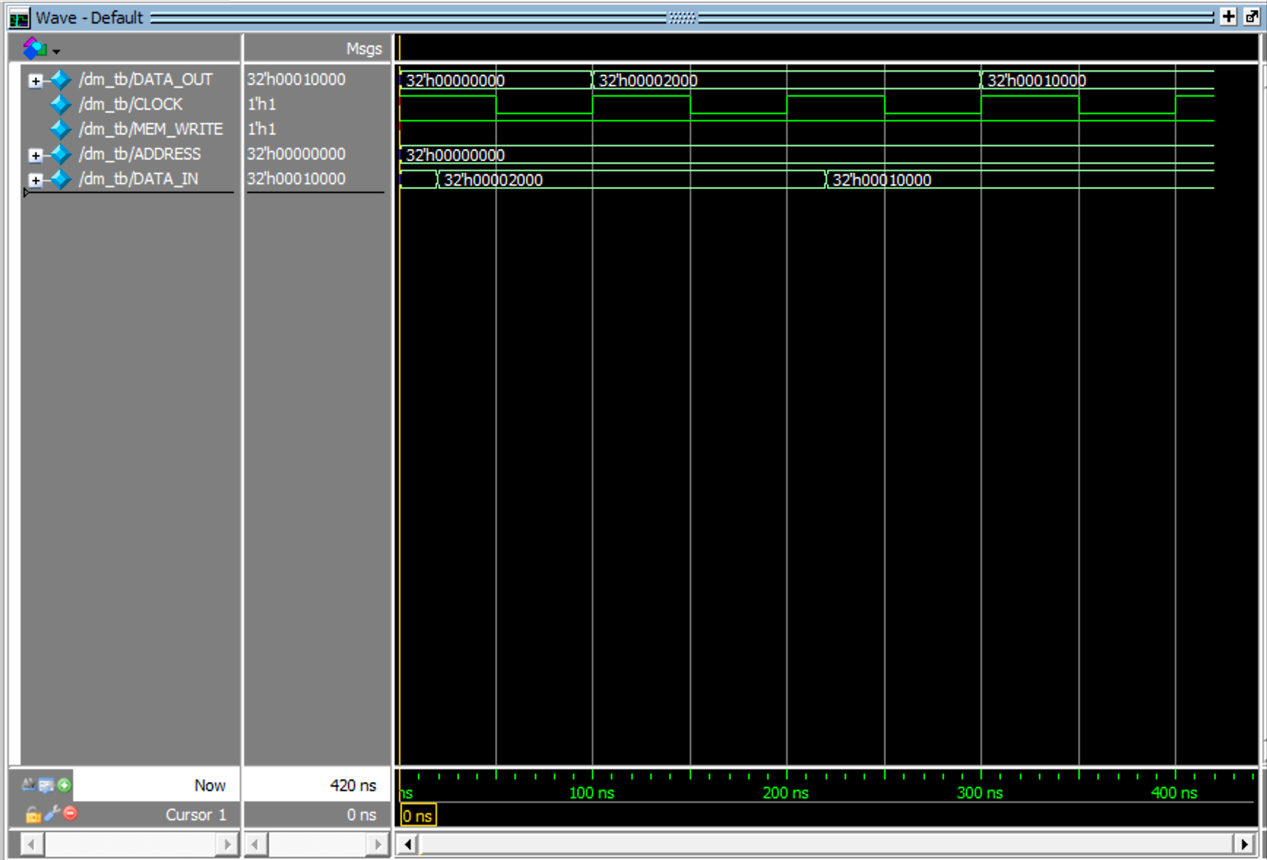
begin

if (mem\_write)

data\_memory[address[11:2]] <= data\_in;

end

endmodule



DATA\_OUT在每个时钟上升沿等于DATA\_IN，并储存下来（图中看不到）。

* 1. s\_cycle\_cpu模块(ADDU)
     1. 功能描述

利用实现的各个基本模块，实现一个能执行addu指令的 单周期CPU。

* + 1. 端口说明

输入：clock, reset。

* + 1. 实现过程

module s\_cycle\_cpu(clock,reset);

//输入

input clock;

input reset;

wire [31:0] npc;

wire [31:0] pc;

wire [31:0] instruction;

wire [4:0] rs; //读寄存器1

wire [4:0] rt; //读寄存器2

wire [4:0] rd;

wire reg\_write;

wire [31:0] a;

wire [31:0] b;

wire [31:0] c;

pc PC(.pc(pc),.clock(clock),.reset(reset),.npc(npc));

assign npc = pc + 4;

im IM(.instruction(instruction),.pc(pc));

assign rs = instruction [25:21];

assign rt = instruction [20:16];

assign rd = instruction [15:11];

assign reg\_write = 1;

gpr GPR(.a(a),.b(b),.clock(clock),.reg\_write(reg\_write),.num\_write(rd),.rs(rs),.rt(rt),.data\_write(c));

alu ALU(.c(c),.a(a),.b(b));

endmodule



可以看到每次指令后pc增加4.

1. 实验总结

做了个可以计算加法的简单cpu。

遇到问题：刚接触很多东西不会做，CPU的构造不够熟悉，verilog程序不太会写，导致刚开始交的几个程序都会报各种错。认真学习相关内容、请教同学后逐渐熟悉并成功完成实验。

比如gpr模块报错“monitor 信号有误 list index out of range 信号读取失败”，赋初值后问题得到解决。

最后CPU连接上忘记给reg\_write赋值也导致结果错误，直接将其赋为1即可解决。

1. 作业

4.1总结parameter的用法，包括作用，定义和实例化。

在模块之间传递参数，可以用几个字母代替一个很长的数字，用起来方便。

示例：parameter xxx = 233;

4.2 模块实例化有哪几种方法？

显式实例化：模块实例的端口采用名字映射的方式，实例化时不考虑端口声明的变量顺序，而是将模块实例外部的信号直接对应于模块端口的变量名。如：

     add add\_inst1 (.sum(sum1),.cout(cout1),.a(a1),.b(b1),.cin(cin1));

隐式实例化：模块实例的端口采用位置映射的方式，即实例化模块时外部信号需要与该模块端口声明的顺序一致。如：

       fulladder add\_inst1 ( a1, b1, cin1, sum1, cout1)

如果实例化时端口长度不匹配，则采取右对齐方式。

4.3 $monitor和$display的用法和区别。

用$monitor()监控和输出参数列表中的表达式或变量值。

用$display()系统任务来显示当前变量的值。

4.4 Verilog实现组合逻辑的方法有哪几种？

① assign xxx = yyy;

② always@(xxx) yyy;

4.5 同步复位和异步复位的区别。

同步复位：复位信号只有在时钟上升沿到来时，才能有效。

异步复位： 无论时钟沿是否到来，只要复位信号有效，就对系统进行复位。

4.6 在verilog语言中，assign语句，阻塞赋值语句（=），非阻塞赋值语句（=>）的用法。

assign里面用=，比如assign xxx = yyy;

<=用在always里，比如xxx <= yyy;