**计算机组成原理**

**课程设计报告**

**学 号\_\_\_\_\_\_\_\_20074214\_\_\_\_\_\_\_\_\_\_**

**姓 名\_\_\_\_\_\_\_\_周子渔\_\_\_\_\_\_\_\_\_\_\_\_**

**指导教师\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**提交日期\_\_\_\_\_\_\_\_2022.7.2\_\_\_\_\_\_\_\_\_\_\_\_**

**成绩评价表**

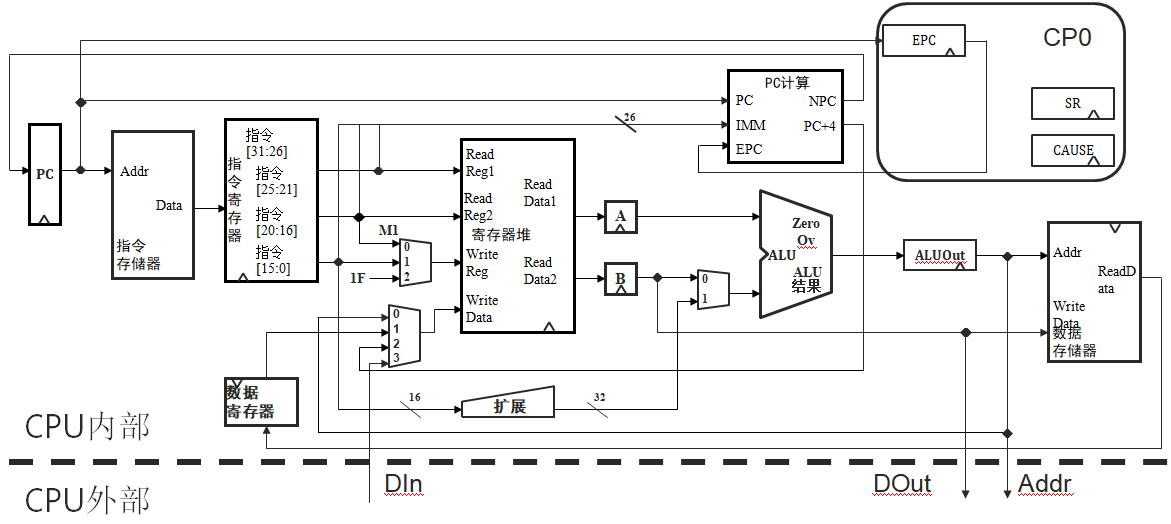
|  |  |  |
| --- | --- | --- |
| **报告内容** | **报告结构** | **报告最终成绩** |
| **□丰富正确**  **□基本正确**  **□有一些问题**  **□问题很大** | **□完全符合要求**  **□基本符合要求**  **□有比较多的缺陷**  **□完全不符合要求** |  |
| **报告与Project功能一致性** | **报告图表** | **总体评价** |
| **□完全一致**  **□基本一致**  **□基本不一致** | **□符合规范**  **□基本符合规范**  **□有一些错误**  **□完全不正确** |  |

**教师签字:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**一、设计说明**

1. MIPS 微系统应包括：MIPS处理器、系统桥和 1 个定时器，32位输入设备、32 位输出设备。
2. MIPS处理器应实现MIPS-Lite3指令集。
   1. MIPS-Lite3＝{MIPS-Lite2，ERET、MFC0、MTC0 }。
   2. MIPS-Lite2＝{addu，subu，ori，lw，sw，beq，lui，addi，addiu，slt， j，jal，jr，lb，sb }。
   3. addi应支持溢出，溢出标志写入寄存器$30中第0位。
3. MIPS处理器为多周期设计。
4. MIPS 微系统支持定时器硬件中断。

**二、数据通路**



**三、模块定义**

# 模块定义

**（1）顶层模块定义**

1）基本描述

主要功能是完成取指令功能。IFU内部包括PC、IM(指令存储器)以及其他相关逻辑。IFU除了能执行顺序取值令外，还能根据BEQ指令的执行情况决定顺序取值令还是转移取值令。

2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号。  1：复位  0：无效 |
| DEV2\_RD | I | 模拟输入设备，输入32位2进制数 |

3）功能定义

连接所有模块，模拟输入设备

4）程序截图



**（2）GPR模块定义**

1）基本描述

GPR包括了所有的寄存器，有两个读端口和一个写端口。寄存器的读输出总是对应于读寄存器号，不需要其他控制信号。但是写寄存器必须明确写使能控制信号。注意写操作是边沿触发的，所以所有的写操作的输入（要写的内容、寄存器号、写控制信号）必须在时钟边沿有效。因为寄存器的写入是边沿触发的，故可以在同一时钟周期内读出和写入同一寄存器：读操作读出以前写入的内容，而写入的内容在下一时钟周期才可读。寄存器号的输入都是5位的，数据线为32位。

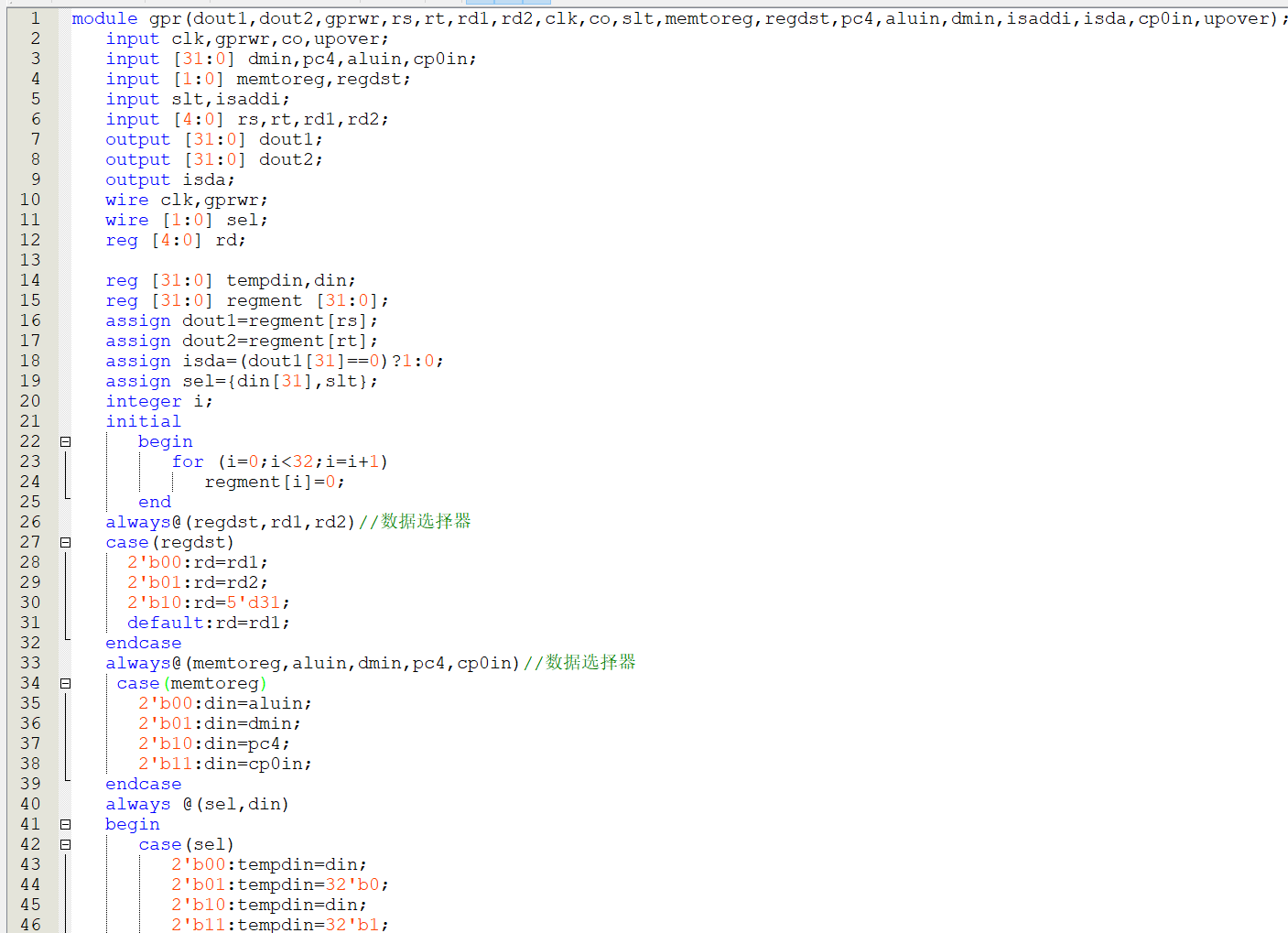
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| rs | I | rs寄存器号 |
| rt | I | rt寄存器号 |
| rw | I | rw寄存器号 |
| regwrite | I | 寄存器写信号。 0：寄存器写使能无效 1：写使能有效 |
| wd | O | 写入的内容 |
| Dout1 | O | rs寄存器内容 |
| Dout2 | O | rt寄存器内容 |
| Upover | I | addi是否溢出信号，0为不溢出，1为溢出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读寄存器 | 根据读寄存器的地址从寄存器中读数据，输出 |
| 2 | 写寄存器 | 如果寄存器的写使能有效，且在clk时钟信号触发边沿，则根据寄存器地址，将要写的数据写入指定地址的寄存器当中。读寄存器与写寄存器可以同时进行：读操作读出以前写入的内容，而写入的内容在下一时钟周期才可读。 |

（4）程序截图



**（3）ALU模块定义**

1）基本描述

ALU算数逻辑单元，是计算机的核心，在本次处理器设计中，ALU可以执行的算数运算包括加法、减法，可以执行的逻辑运算包括或操作。

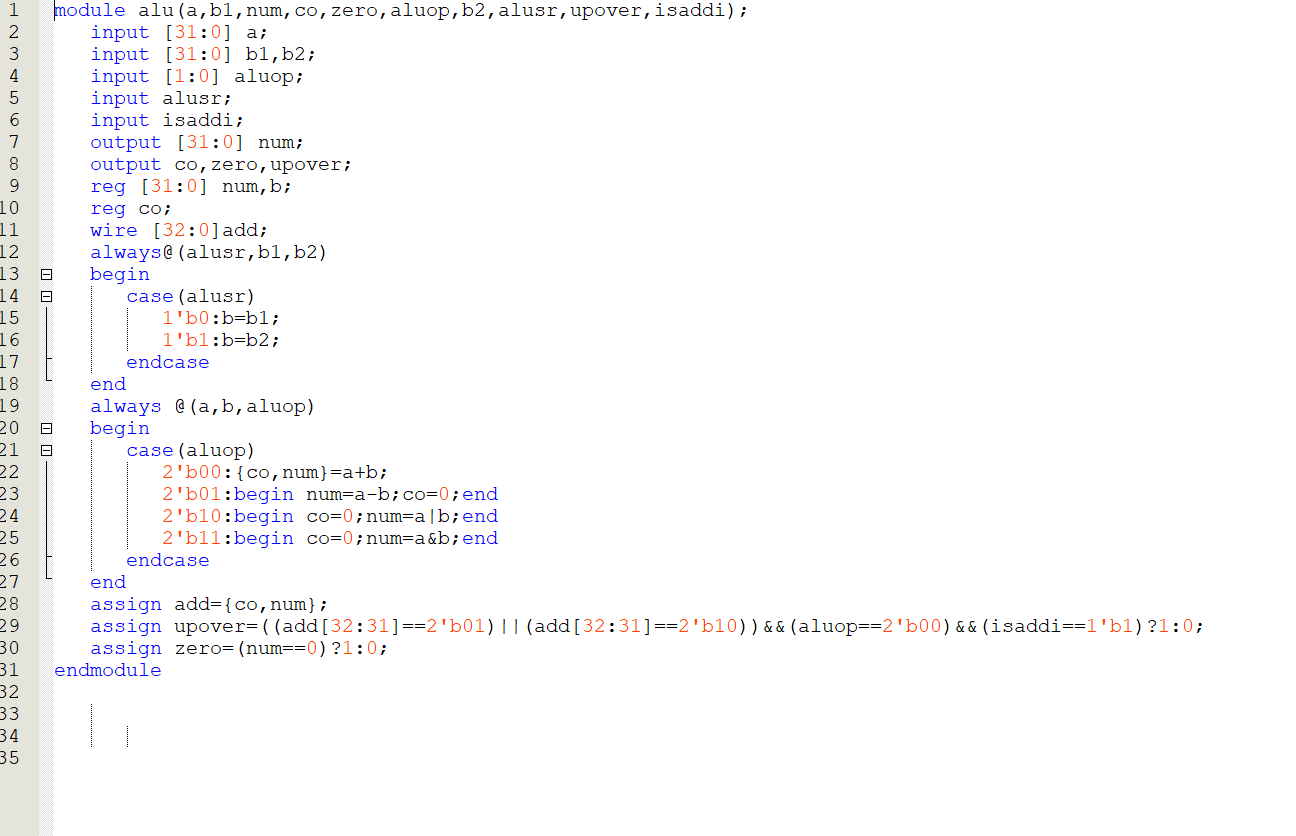
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A | I | 参与运算的输入数据1 |
| B | I | 参与运算的输入数据2 |
| ALU\_op | I | ALU控制信号  Switch (ALUop)  00: alu\_out = A + B  01: alu\_out = A - B  10: alu\_out = A || B |
| alu\_out | O | ALU运算结果 |
| zero | O | 运算结果是否为0的标志位，用于beq相等则分支指令 |
| Upover | O | 判断addi是否溢出 |
| slt | I | 是否为slt指令 |
| isaddi | I | 是否为addi指令 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加运算 | 若ALUop = 00，则执行加运算。把两个32-bit的输入数据相加，然后将结果Result输出。在本实验中，不支持溢出判断。 |
| 2 | 减运算 | 若ALUop = 01，则执行减运算。把两个32-bit的输入数据相减，然后将结果Result输出。 |
| 3 | 按位或 | 若ALUop = 10，则执行按位或运算。把两个32-bit的输入按位或，然后将结果Result输出。 |

（4）程序截图



**（4）EXT模块定义**

1）基本描述

EXT是符号拓展单元，有一个16位的输入，符号拓展为32位后输出。在lw取数、sw存数、beq相等则分支等指令中的立即数字段包含一个无符号/有符号二进制补码表示的16位数。为了将这个立即数字段加到一个32位的寄存器，计算机必须将这个16位的数转换成数值上相等的32位的数。

对于有符号数来说，这种方法就是将原有的16位数简单赋值到32位新数的低16位，其最高有效位（符号位）则以复制的方式填满新数的高16位，实现符号拓展。

对无符号数来说，将原有的16位数简单赋值到32位新数的低16位，用0填满新数的高16位。

对于lui指令，需要加载至高16位，将原有的16位数简单赋值到32位新数的高16位，低16位用0填充。

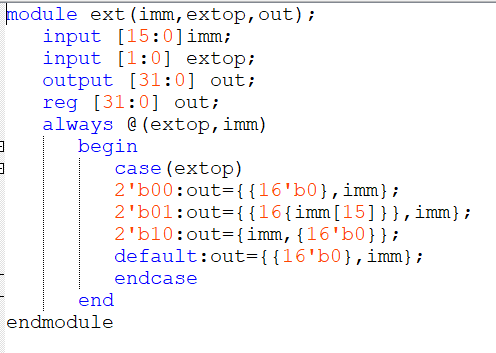
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm16 | I | 16位立即数 |
| extop | I | 符号拓展控制信号  Switch (EXTOp)  00: imm32 = unsign 16 to 32 extend  01: imm32 = sign 16 to 32 extend  10: imm32 = extend to high 16 bit |
| imm32 | O | 符号拓展结果输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 有符号拓展 | 将原有的16位数简单赋值到32位新数的低16位，其最高有效位（符号位）则以复制的方式填满新数的高16位，实现符号拓展。 |
| 2 | 无符号拓展 | 将原有的16位数简单赋值到32位新数的低16位，用0填满新数的高16位。 |
| 3 | 拓展至高16位 | 将原有的16位数简单赋值到32位新数的高16位，低16位用0填充。 |

（4）程序截图



**（5）IM模块定义**

1）基本描述

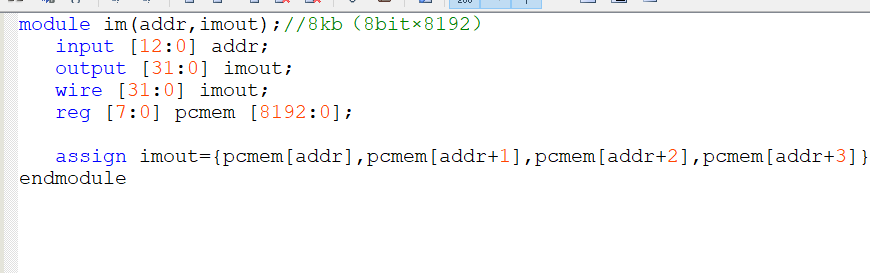
指令寄存器

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr | I | 参与运算的输入数据1 |
| dout | O | 输出指令 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读指令 | 根据地址Addr从相应位置读出数据，dout为数据输出 |



**（6）DM模块定义**

1）基本描述

数据寄存器

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr | I | 参与运算的输入数据1 |
| din | I | 输出 |
| dout | O | 输出 |
| we | I | 写使能信号 |
| clk | I | 时钟信号 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 根据地址addr从相应位置读出数据，dout为数据输出 |
| 2 | 写数据 | 如果we写使能有效，根据地址addr将数据din写入存储器的相应位置 |

（4）程序截图



**（7）NPC模块定义**

1）基本描述

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| curpc | I | 当前pc |
| imm26 | I | 26位立即数 |
| register | I | jr的寄存器输入 |
| npc\_sel | I | npc功能选择  0：pc+4  1：beq  2：j |
| zero | O | 零信号 |
| nxtpc | O | 下一个pc |
| pc\_add4 | O | jal的存储的pc+4 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算下一条pc | 根据npc\_sel计算下一条pc |
| 2 | 改写pc | 进入中断程序后把pc改写为中断地址4180 |

（4）程序截图



**（8）Controller模块定义**

1）基本描述

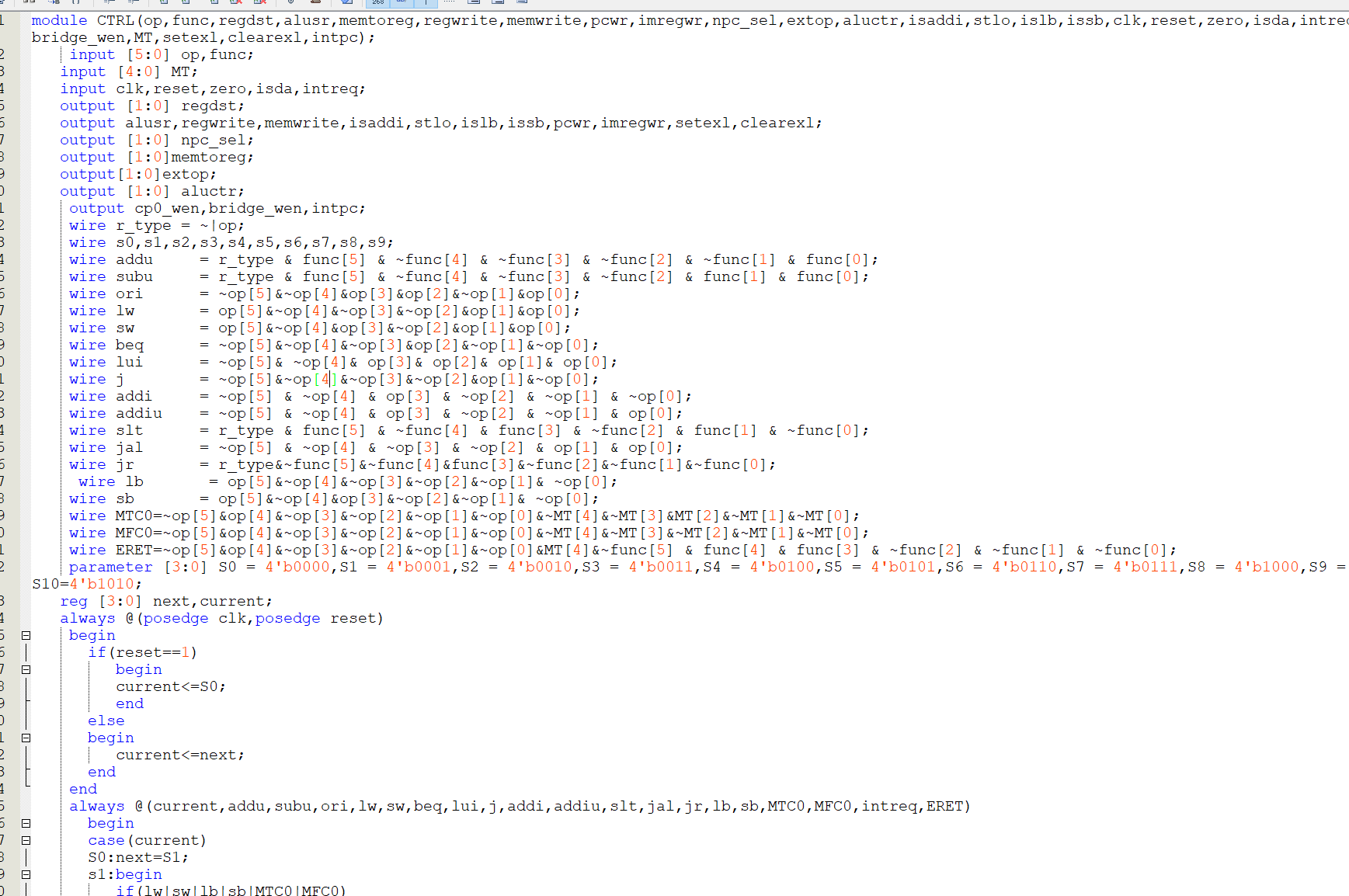
Controller是控制单元。控制单元以指令为输入，能够产生每个状态单元的写信号，每个多选器的选择信号和ALU的控制信号。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| func | I | 指令的func字段 |
| op | I | 指令的opcode字段 |
| reg\_sel | O | 寄存器选择 |
| wd\_sel | O | 写入数据选择 |
| alu\_op | O | 控制ALU执行的运算 |
| we | O | 数据寄存器写使能 |
| npc\_sel | O | npc选择 |
| ext\_op | O | ext选择 |
| regwrite | O | 寄存器写使能 |
| alu\_sel | O | 第二个ALU操作数的来源 |
| addi | O | 是否为addi |
| slt | O | 是否为slt |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 产生控制信号 | 控制单元以指令为输入，能够产生每个状态单元的写信号，每个多选器的选择信号和ALU的控制信号。 |
| 2 | 控制状态转移 | 根据每条指令的不同改变当前状态机的现态 |
| 3 | 检测中断 | 检测每条指令的最后一个状态是不是s10 |

****

**（9）CP0模块**

（1）基本描述

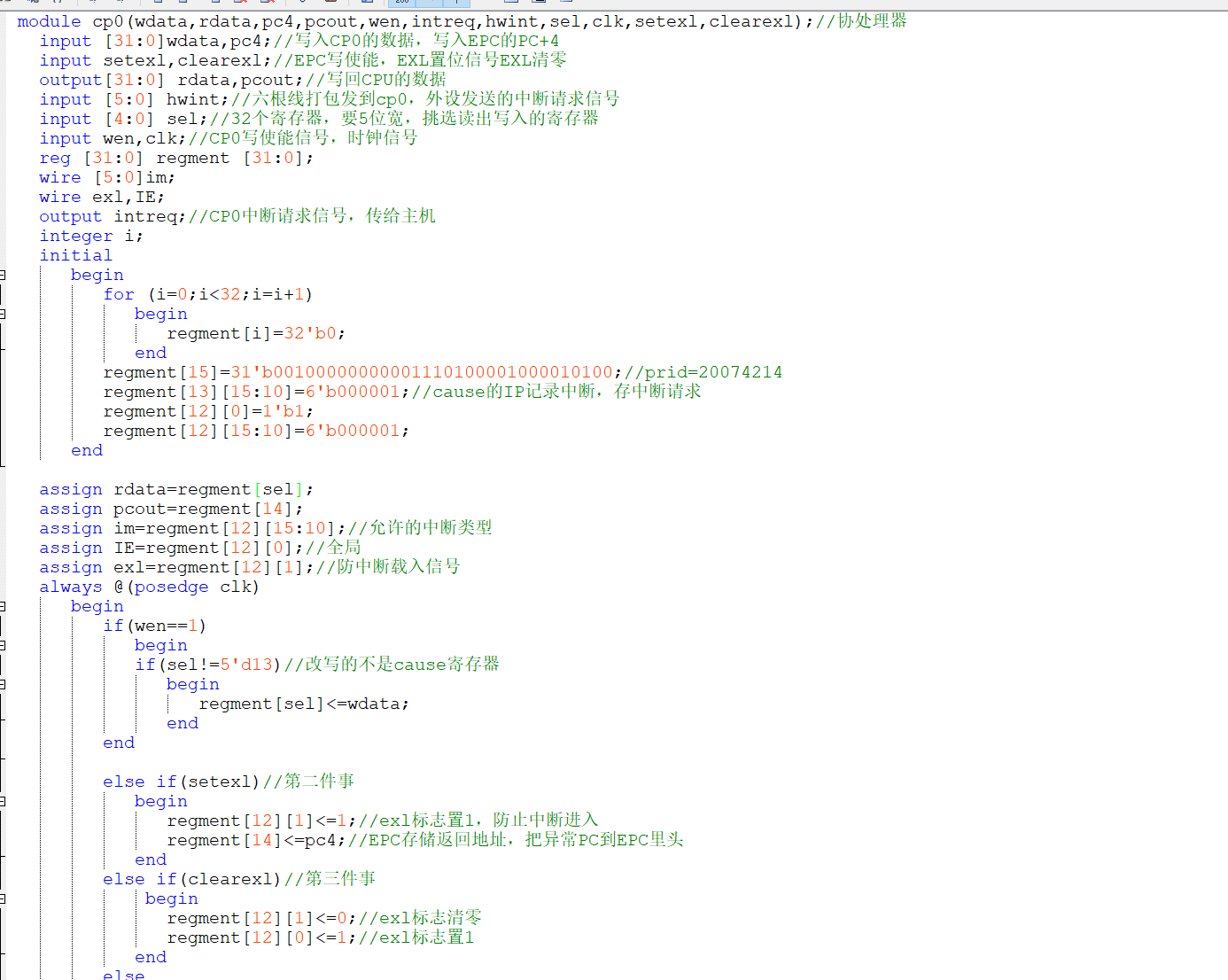
CP0主要完成功能是处理中断。CP0内部包含EPC、SR、Cause、Prid寄存器，EPC用来保存中断时的pc值，SR用来对系统进行控制，Cause用来读取指令，写入硬件控制，Prid用来实现个性编码。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号。 |
| reset | I | 复位信号。  1：复位  0：无效 |
| we | I | CP0寄存器组写使能信号。  1：允许当前指令写入CP0寄存器组  0：不允许当前指令写入CP0寄存器组 |
| exlset | I | 程序已进入中断标志。  1：程序进入中断  0：程序未进入中断 |
| exlclr | I | 清除程序已进入中断标志。  1：清除程序已进入中断  0：不清除程序已进入中断 |
| pcin[31:0] | I | 用于给epc提供中断时的pc值。 |
| datain[31:0] | I | 输入到CP0寄存器组的数据。 |
| hwint[5:0] | I | 用于记录六个硬件中断。 |
| regaddr[4:0] | I | 用于选择向CP0寄存器组的哪个寄存器写入数据。 |
| intreq | O | 中断标志。  1：中断  0：不中断 |
| epcout[31:0] | O | 中断处理完毕后写入npc的值。 |
| dataout[31:0] | O | CP0寄存器组向GPR输出的数据。 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，所有寄存器数据清零。 |
| 2 | 取数据 | 根据regaddr的地址从寄存器里取出数据。 |
| 3 | 存数据 | 当CP0寄存器组使能端打开时，根据regaddr的地址将GPR输入的数据写入对应寄存器；当CP0寄存器使能端关闭时，根据regaddr的地址将与中断有关数据写入对应寄存器。 |
| 4 | 个性化编码 | 对prid寄存器进行赋值，实现个性化编码。 |



**（10）out模块**

（1）基本描述

out模块模拟输出设备。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| en | I | 输出设备写使能 |
| sel | I | 输出寄存器a1、a2选择信号 |
| din | I | 输入数据 |
| dout | O | 输出数据 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出 | 根据sel选择信号输出a1\a2寄存器数据。 |



**（10）counter模块**

（1）基本描述

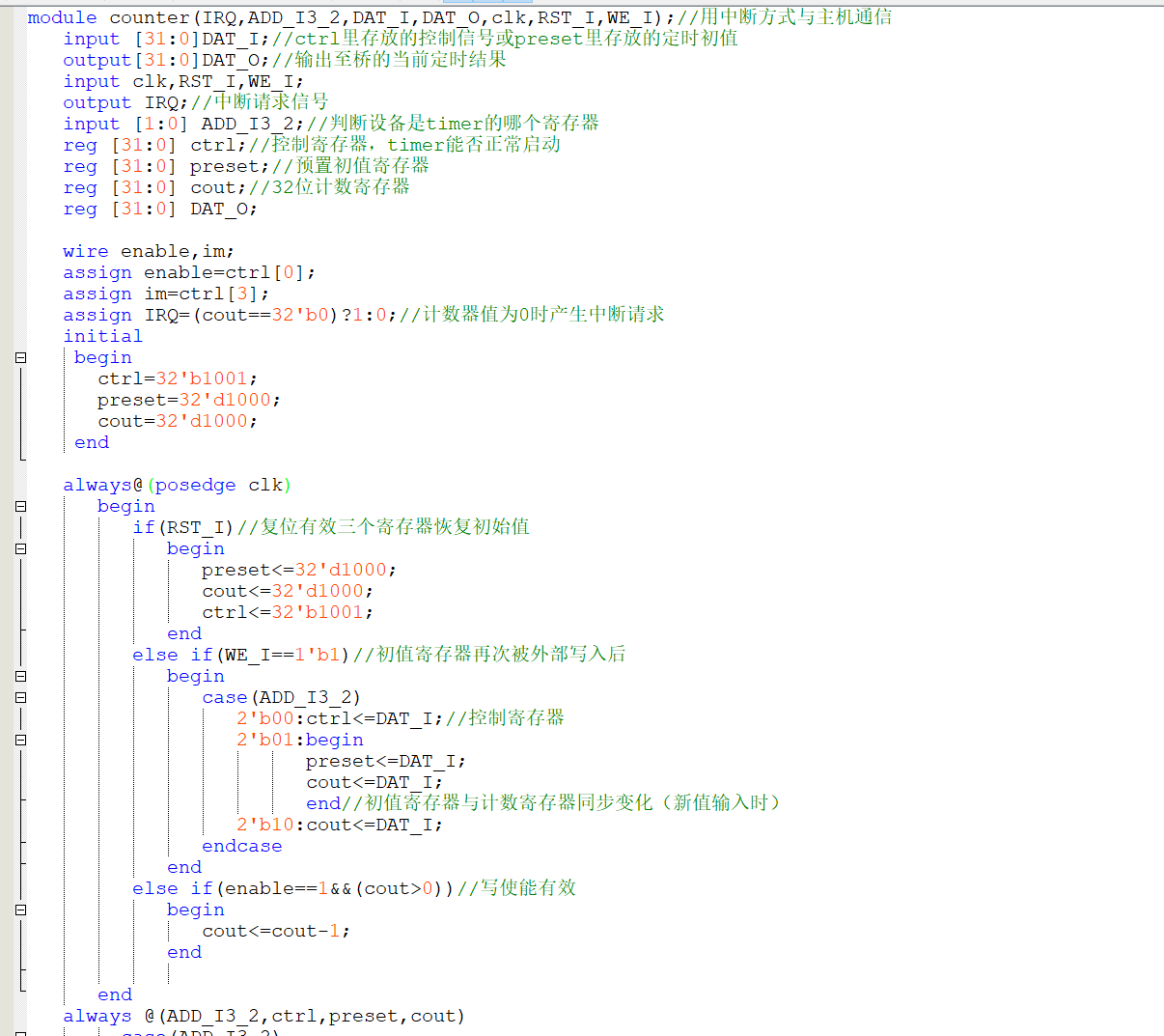
Timer由控制寄存器(ctrl)、初值寄存器(preset)、32位计数器(count)及中断产生逻辑构成。Timer的主要功能是根据不同的计数模式，在计数为0后，计数器或者自动装填初值并重新倒计数，或者保持在0值直至初值寄存器再次被装载，当计数器工作在模式0并且在中断允许的前提下，当计数器计数值为0时，中断产生逻辑产生中断请求(intreq为1)。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| RST\_I | I | 复位信号  0：复位  1：无效 |
| ADD\_I3\_2[1:0] | I | 判断设备是Timer中的哪个寄存器。  00：ctrl  01：preset  10：count |
| DAT\_I[31:0] | I | ctrl里存放的控制信号或preset里存放的定时初值。 |
| DAT\_O[31:0] | O | 输出至桥的当前定时结果。 |
| IRQ | O | 中断请求 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，所有寄存器清零。 |
| 2 | 计数 | 模式0：当计数器倒计数为0后，计数器停止计数。当初值寄存器再次被外部写入后，初值寄存器值再次被加载至计数器，计数器重新启动倒计数；  模式1：当计数器倒计数为0后，初值寄存器值被自动加载至计数器，计数器继续倒计数。 |
| 3 | 产生中断 | 当计数器工作在模式0并且在中断允许的前提下，当计数器计数值为0时，中断产生逻辑产生中断请求(intreq为1)。 |



**（11）bridge模块**

（1）基本描述

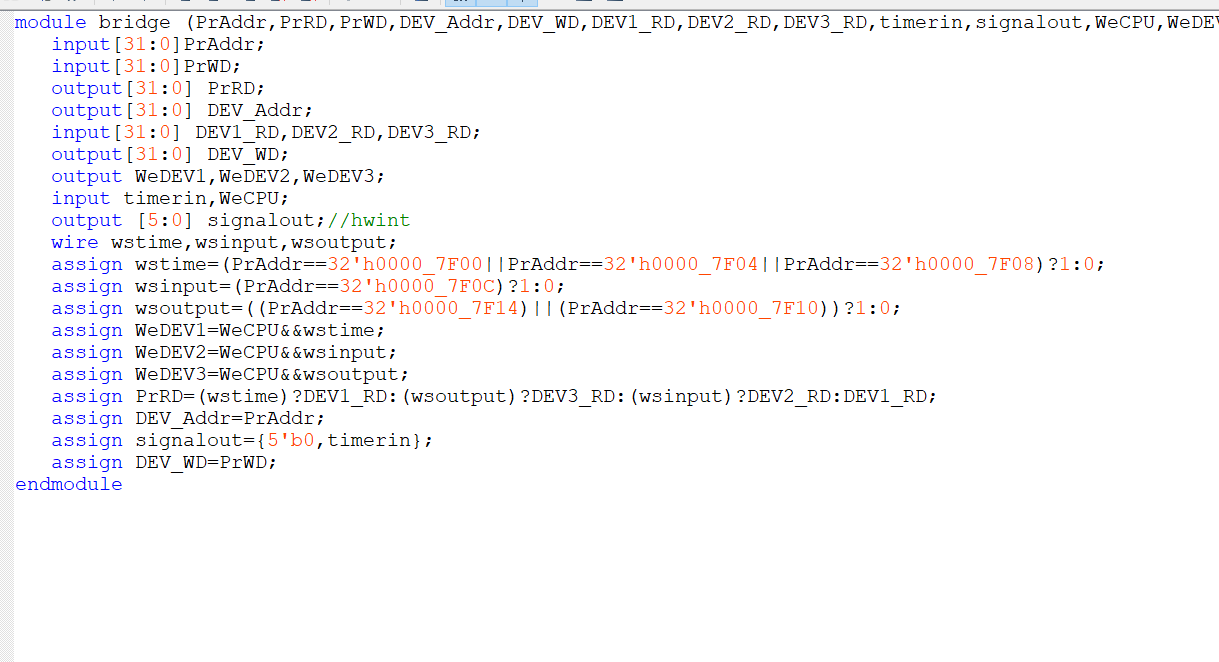
Bridge主要功能是连通外设和CPU，用于输出地址、地址匹配和CPU读数据。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| wecpu | I | 写数据使能端。  1：外设可写  0：外设不可写 |
| praddr[31:0] | I | 用于判断地址范围处于哪个外设。 |
| wdin[31:0] | I | CPU输出至桥的数据。 |
| wdout[31:0] | O | wdout=wdin |
| hwint[5:0] | I | 用于记录六个硬件中断。 |
| hwintout[5:0] | O | hwintout=hwint |
| DEV1\_RD [31:0] | I | 输入设备输入至桥的数据。 |
| DEV2\_RD [31:0] | I | 输出设备输入至桥的数据。 |
| DEV3\_RD [31:0] | I | 定时器输入至桥的数据。 |
| DEV\_RD[31:0] | O | 桥输出至CPU的数据。 |
| DEV\_WD [1:0] | I | 判断设备是Timer中的哪一个寄存器、  或Out中的哪一个寄存器。  00：ctrl \ a1  01：preset\ a2  10：count |
| outwe | O | 输出外设使能端。 |
| tcwe | O | 定时器外设使能端。 |

(3)功能定义

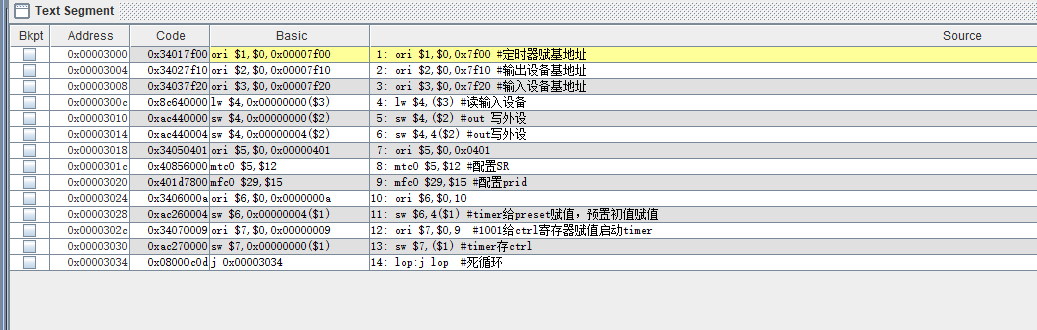
|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 与CPU对接读写数据 | 根据lw、sw计算的地址范围，判断该地址范围是否命中任意一个外设，若命中且为sw指令，打开桥使能端将数据写入外设；若命中且为lw指令，将数据写入GPR中。 |
| 2 | 地址匹配 | 设备地址译码，基地址高位用于译码选择设备。 |
| 3 | 输出地址 | 将praddr[x:2]直接输出，x为N个设备中地址空间需求最大者。 |

****

**四、程序测试**

**1.主程序及中断程序**

**** ****





机器码

34017f00

34027f10

34037f20

8c640000

ac440000

ac440004

34050401

40856000

401d7800

3406000a

ac260004

34070009

ac270000

08000c0d

中断子程序：

8c680000

8c490000

11090003

ac480000

ac480004

10000003

8c4a0004

254a0001

ac4a0004

340b000a

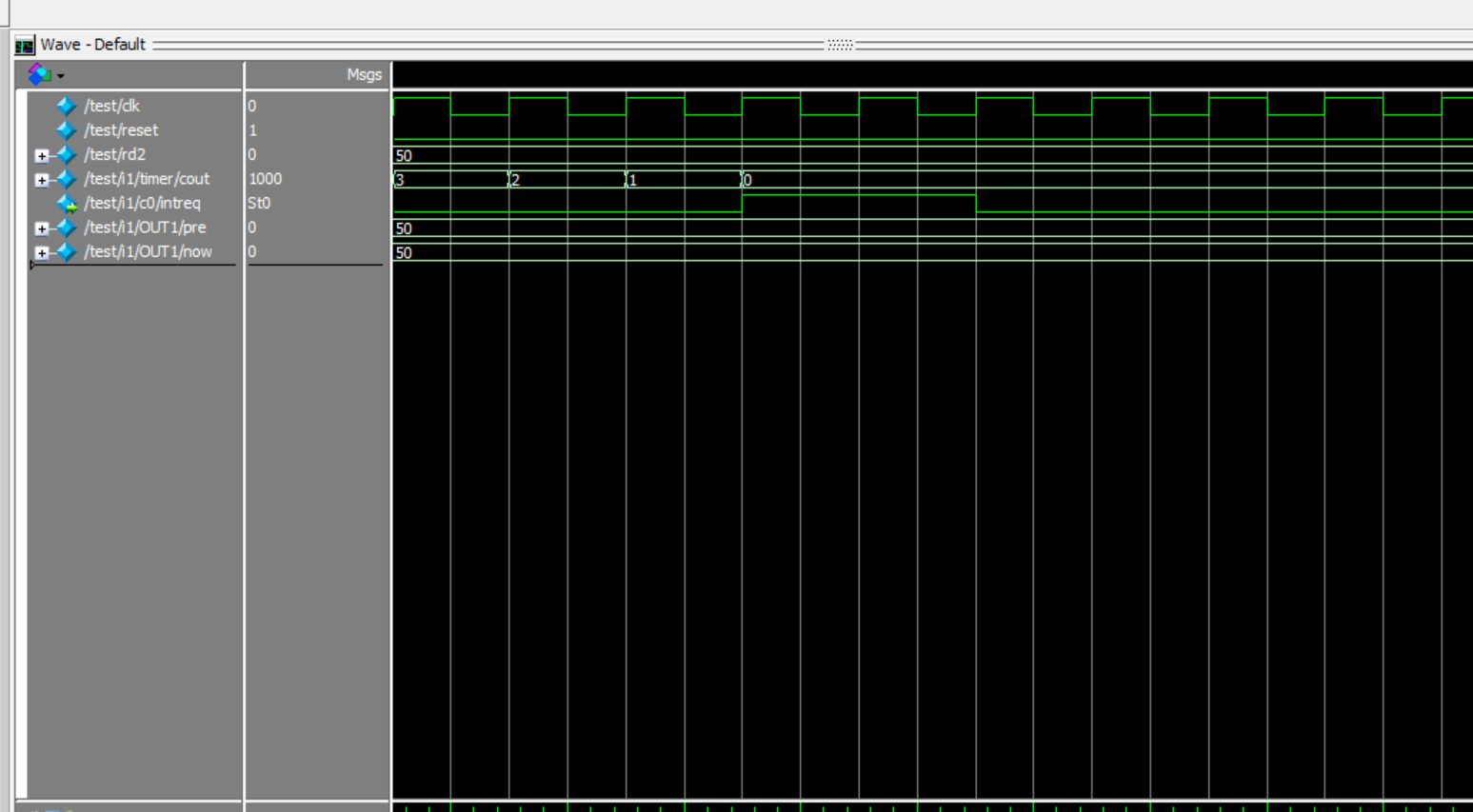
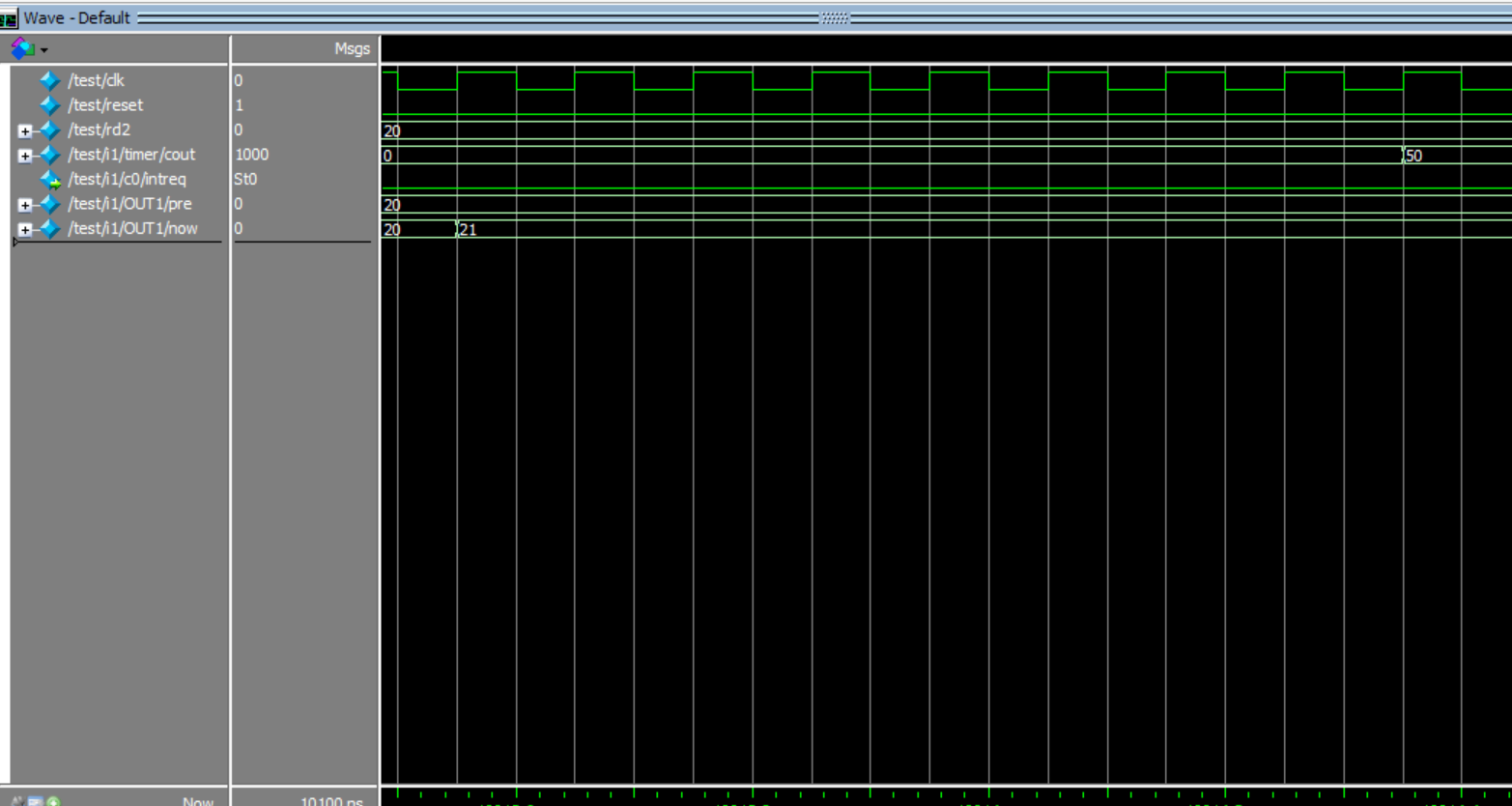
ac2b0004

340c0009

ac2c0000

42000018

**五、测试结果**

**** ****

**波形解释：**首先在主程序中设置的input的两个输入设备值都为50，前面正常执行主程序，进入第一个中断的时候（12.5nm），将此时的pc地址3044写入cp0，开始读取中断指令。4180是终端的pc，第一条中断指令是8c8f0000，之后开始判断比较，执行到acd00000（15.1nm）后写先提取当前的now然后完成+1，再把now+1写回当前now就是51，以上是输入设备和输出设备值相同时的情况都是50，实现的功能是正常+1。之后到10000nm的时候，test程序里实现输入设备值更改，从50变到20（10001.2nm），进入中断程序比较后发现这个时候输入设备和输出设备的值不相同，一个是20一个是50，所以就完成输出设备的改写，执行acaf0000指令，对应的就是中断程序中的sw指令实现pre的改写，之后执行accf0000指令完成另一条sw指令实现对now的改写。之后从20开始+1，直到中断程序（10013.3nm）结束执行42000018指令回到主程序的408c6000。

**中断产生：**counter（timer里的计数输出cout）倒计时完整一秒钟，时间到之后，计时计到0，产生一个中断请求信号irq，这个信号进入bridge，形成一个六位hwint信号，定义这个信号的高五位为0，第0位为timer发来的中断信号。这个信号传入cp0进行运算，后将通过公式将intreq信号置1，表示此时产生中断。intreq信号传给CPU之后，在controller新增一个fsm状态s10判断是否有中断请求。如果有中断请求，还要在同一个时间周期里完成三步，以进入中断服务程序：首先prwr置1，才能把npc的值变成规定的中断请求的地址4180；其次epcwr置1，把pc+4，就是返回地址存到cp0和epc寄存器里（第14位）；第三步将cp0里exl标志位，也就是进入中断时的标记置1，防止再有其他中断进入。此时进入了中断程序，结束后还要完成2步：首先epc把地址取出来回写给npc，其次把进入中断的标记exl复原。

**六、心得体会**

这次的p2难度比较大，主要集中在cp0和timer的设计上，为了解决这些问题，我反复研究学习了老师上课的ppt，根据ppt上cp0入口和出口参数的设计进行编程，起到了事半功倍的效果。同时在测试程序和中断程序的设计上我也出现了一些不懂的问题，通过和同学交流也得到了解决，这也告诉我如果遇到了想不明白的问题时，与同学的及时交流和讨论往往是解决问题的好方式。