2020CPU大作业报告

小组成员：金冬阳11911221　马卓远11912412

一． CPU功能及使用说明

1. 该CPU的CPI为1，即每个指令的运行周期数为1个周期。
2. 该CPU为单周期CPU。
3. 该CPU的复位信号为FPGA芯片上的P5按键，按动P5按键即对CPU进行reset操作。
4. uart Uart启动的按钮FPGA芯片上的P1按键，按动P1按键计可启动Uart。

二． 顶层模块设计

1. 该CPU的输入为

fpga_clk　　FPGA芯片时钟周期

fpga_rst　　FPGA芯片复位信号

switch2N4　　拨动开关信号

start_pg　　uart接口的启动信号

rx　　uart的接收数据线

输出信号为

led2N4　　led灯信号

Y　　七段数码显示管显示0~F的信号

DIG　　控制八个七段数码显示管显示信息

tx　　uart发送数据线

顶层设计文件

```
`timescale 1ns / 1ps
module
CPU(fpga_clk,fpga_rst,switch2N4,led2N4,Y,DIG,start_pg,rx,tx);//dis和sel
是新加的！！！
output  [7:0] Y;//dis
output  [7:0]DIG;//sel
input fpga_clk;//可能要改成fpga_clk!!!!
input fpga_rst;//可能要改成fpga_rst!!!!
input[23:0] switch2N4;
output wire [23:0]led2N4;
input start_pg;   //Active High//Uart启动的按钮！！！
input rx;        // receive data by UART接收数据线
output tx;       // send data by UART   发送数据线

wire rst;
wire upg_clk,upg_clk_o;
wire upg_wen_o; // Uart write out enable
wire upg_done_o;// Uart rx data have done
//data to which memory unit of program _rom/dmemory32
wire[14:0] upg_adr_o;
wire[31:0]upg_dat_o;

wire spg_bufg;
BUFG U1(.I(start_pg),.O(spg_bufg)); //de-twitter  缓冲器！！去抖  自带的
直接这么用就行了
//Generate UART Programmer reset signal
reg upg_rst;
always@(posedge fpga_clk)begin //应该clk换成fpga_clk!!!
   if(spg_bufg) upg_rst = 0;
   if (fpga_rst) upg_rst=1;
   end
   assign rst = fpga_rst|!upg_rst;

wire clock;
wire [31:0]Instruction;
wire [31:0]PC_plus_4_out;
wire [31:0]Addr_result;
```

```
wire [31:0]Read_data_1;
wire Branch,nBranch,Jmp,Jal,Jr,Zero;
wire [31:0]opcplus4;
wire [31:0] next_PC;
wire [31:0]PC;
//
wire [31:0] Read_data_2;
wire [31:0] read_data;
wire RegWrite;
wire RegDst;
wire [31:0]Sign_extend;

//
wire ALUSrc;
wire MemorIOtoReg;
wire MemRead;
wire MemWrite;
wire IORead;
wire IOWrite;
wire I_format;
wire Sftmd;
wire [1:0]ALUOp;
//
wire [4:0]Shamt;
wire [31:0]ALU_Result;
wire[31:0] read_data_fromMemory;
wire[31:0] address;
wire[31:0] write_data;
wire [15:0]ioread_data;
wire switchcs;
wire [1:0]switchaddr;
wire [15:0]switchrdata;
wire [23:0]switch_i;

wire ledcs;
wire [1:0]ledaddr;


clk1 cpuclk (
.clk_in1(fpga_clk),//原来是clk
.clk_out1(clock),
.clk_out2(upg_clk)
);
```

```
uart_bmpg_0 uartpg(
.upg_clk_i(upg_clk),
.upg_rst_i(upg_rst),
.upg_clk_o(upg_clk_o),
.upg_wen_o(upg_wen_o),
.upg_adr_o(upg_adr_o),
.upg_dat_o(upg_dat_o),
.upg_done_o(upg_done_o),
.upg_rx_i(rx),
.upg_tx_o(tx)
);

 wire  upg_wen_i;
 assign upg_wen_i=upg_wen_o&upg_adr_o[14];

 wire  upg_wen_i1;
 assign upg_wen_i1=upg_wen_o&(!upg_adr_o[14]);



Ifetc32 ifetch(
.Instruction(Instruction),.pco(PC),.Addr_result(Addr_result),.Read_data_
1(Read_data_1),.Branch(Branch),.nBranch(nBranch),.Jmp(Jmp),.Jal(Jal),
.Jr(Jr),.Zero(Zero),.clock(clock),.reset(rst),.link_addr(next_PC),.branc
h_base_addr(PC_plus_4_out),
.upg_rst_i(upg_rst),.upg_clk_i(upg_clk_o),
.upg_wen_i(upg_wen_i1),.upg_adr_i(upg_adr_o[13:0]),.upg_dat_i(upg_dat_o)
,.upg_done_i(upg_done_o)
);

Idecode32 idecode(
.read_data_1(Read_data_1),.read_data_2(Read_data_2),.Instruction(Instruc
tion),.read_data(read_data),
.ALU_result(ALU_Result),.Jal(Jal),.RegWrite(RegWrite),.MemtoReg(MemorIOt
oReg),.RegDst(RegDst),
.imme_extend(Sign_extend),.clock(clock),.reset(rst),.opcplus4(opcplus4)
);

control32 control(
.Opcode(Instruction[31:26]),.Function_opcode(Instruction[5:0]),.Alu_resu
ltHigh(ALU_Result[31:10]),.Jr(Jr),
.RegDST(RegDst),.ALUSrc(ALUSrc),
.MemorIOtoReg(MemorIOtoReg),.RegWrite(RegWrite),.MemRead(MemRead),.MemWr
ite(MemWrite),.IORead(IORead),
.IOWrite(IOWrite),.Branch(Branch),.nBranch(nBranch),
```

```
.Jmp(Jmp),.Jal(Jal),.I_format(I_format),.Sftmd(Sftmd),.ALUOp(ALUOp)
);

Executs32 execute(
.Read_data_1(Read_data_1),.Read_data_2(Read_data_2),.Imme_extend(Sign_ex
tend),.Function_opcode(Instruction[5:0]),     //好几个都是instruction的部
.opcode(Instruction[31:26]),.ALUOp(ALUOp),.Shamt(Instruction[10:6]),
.ALUSrc(ALUSrc),.I_format(I_format),
.Zero(Zero),.Jr(Jr),.Sftmd(Sftmd),.ALU_Result(ALU_Result),
.Addr_Result(Addr_result),.PC_plus_4(PC_plus_4_out)
);


dmemory32 memory(.read_data(read_data_fromMemory),.address(address),
.write_data(write_data),.Memwrite(MemWrite),.clock(clock),
.upg_rst_i(upg_rst),.upg_clk_i(upg_clk_o),
.upg_wen_i(upg_wen_i),.upg_adr_i(upg_adr_o[13:0]),.upg_dat_i(upg_dat_o),
.upg_done_i(upg_done_o)
);

memorio memio(
.caddress(ALU_Result),.memread(MemRead),.memwrite(MemWrite),.ioread(IORe
ad),.iowrite(IOWrite),
.mread_data(read_data_fromMemory),.ioread_data(switchrdata),.wdata(Read_
data_2),
.rdata(read_data),.write_data(write_data),.address(address),.LEDCtrl(led
cs),.SwitchCtrl(switchcs)
);

Switch switch24(
.switclk(clock),.switrst(rst),.switchread(IORead),.switchcs(switchcs),
.switchaddr(address[1:0]),.switchrdata(switchrdata),.switch_i(switch2N4)
);

display display24(
.led_clk(clock),
.ledrst(rst),
.ledwrite(IOWrite),
.ledcs(ledcs),
.ledaddr(address[1:0]),
.ledwdata(write_data[31:0]),
.Y(Y),
.DIG(DIG)
);
```

```
Led led(
.led_clk(clock),
.ledrst(rst),
.ledwrite(IOWrite),
.ledcs(ledcs),
.ledaddr(address[1:0]),
.ledwdata(write_data[16:0]),
.ledout(led2N4)
);
endmodule
```

三. 子模块设计说明

1. 子模块功能

clk1	CPU时钟周期
Ifetc32	存储指令并取出指令
ProgramROM_UART	Ifetc32的一个子模块，实现了区分CPU工作时的uart状态和正常状态
Idecode32	解码器
control32	控制信号
Executs32	ALU
dmemory32	储存器（模拟memory的）
memorio	判断是从memory读取信息还是从外部读取输入信息
Switch	读取开关上的信息
display	根据CPU的信息，七段数码管显示数据
segtube	display的子模块，七段数码管。
frequency	segtube的子模块，分频
control	segtube的子模块，对应0到f的显示
Led	根据CPU的信息，led灯显示数据

2.具体功能分析

clk1

输入：

(clk_in1) fpga_clk　　位宽：1bit

输出：

(clk_out1) clock　　位宽：1bit

(clk_out2) upg_clk　　位宽：1bit

为用于Uart通信的IP核添加一个频率为10 Mhz的新clk_out信号（clk_out2）

Ifetc32

输入：

Addr_result(Addr_result)　　位宽：32bit

来自执行单元,算出的跳转地址

Read_data_1(Read_data_1)　　位宽：32bit

来自译码单元，jr指令用的地址

Branch(Branch)　　位宽：1bit

来自控制单元

nBranch(nBranch)　　位宽：1bit

来自控制单元

Jmp(Jmp)　　位宽：1bit

来自控制单元

Jal(Jal)　　位宽：1bit

来自控制单元

Jr(Jr) 位宽：1bit

来自控制单元

Zero(Zero) 位宽：1bit

来自执行单元

clock(clock) 位宽：1bit

时钟信号

reset(rst) 位宽：1bit

复位信号

upg_rst_i(upg_rst) 位宽：1bit

upg_clk_i(upg_clk_o) 位宽：1bit

upg_wen_i(upg_wen_i1) 位宽：1bit

upg_adr_i(upg_adr_o[13:0]) 位宽：14bit

upg_dat_i(upg_dat_o) 位宽：32bit

upg_done_i(upg_done_o) 位宽：1bit

输出：

pco(PC) 位宽：32bit

(pc+4)送执行单元 pco

Instruction(Instruction) 位宽：32bit

输出指令到其他模块

link_addr(next_PC) 位宽：32bit

JAL指令专用的PC+4

branch_base_addr(PC_plus_4_out) 位宽：32bit

在每次上升沿和下降沿的时候更新next_PC的值，每次clock的下降沿的时候更新PC的值

ProgramROM_UART

输入：

rom_clk_i(clock)　　位宽：1bit

rom_adr_i(PC[15:2])　　位宽：14bit

upg_rst_i(upg_rst_i)　　位宽：1bit

upg_clk_i(upg_clk_i)　　位宽：1bit

upg_wen_i(upg_wen_i)　　位宽：1bit

upg_adr_i(upg_adr_i)　　位宽：14bit

upg_dat_i(upg_dat_i)　　位宽：32bit

upg_done_i(upg_done_i)　　位宽：1bit

输出：

Instruction_o(Instruction)　　位宽：32bit

Idecode32

输入：

Instruction(Instruction)　　位宽：32bit

read_data(read_data)　　位宽：32bit

ALU_result(ALU_Result)　　位宽：32bit

Jal(Jal)　　位宽：1bit

RegWrite(RegWrite)　　位宽：1bit

MemtoReg(MemorIOtoReg)　　位宽：32bit

RegDst(RegDst)　　位宽：32bit

clock(clock)　　位宽：1bit

reset(rst)　　位宽：1bit

输出：

read_data_1(Read_data_1)　　位宽：32bit

read_data_2(Read_data_2)　　位宽：32bit

imme_extend(Sign_extend)　　位宽：32bit

opcplus4(opcplus4)　　位宽：32bit

在每次上升沿和下降沿的时候根据情况更新write_register_address、write_data的值

并且在clock的上升沿更新对应的register的值

memorio

输入：

caddress　　位宽：32bit

from alu_result in executs32

memread　位宽：32bit

read memory, from control32

memwrite　　位宽：32bit

write memory, from control32

ioread　　位宽：32bit

read IO, from control32

iowrite 位宽：32bit

write IO, from control32

mread_data 位宽：32bit

data from memory

ioread_data 位宽：16bit

data from io,16 bits

wdata 位宽：16bit

the data from idecode32,that want to write memory or io

输出：

rdata 位宽：16bit

data from memory or IO that want to read into register

write_data 位宽：16bit

data to memory or I/O

address; 位宽：16bit

address to mAddress and I/O

在上升沿和下降沿的时候，更新write_data的值

dmemory32

输入：

clock 位宽：1bit

Clock signal

address 位宽：32bit

the address of memory unit which is to be read/write

Memwrite 位宽：32bit

used to determine to write or read

write_data 位宽：32bit

data tobe wirte into the memory unit

upg_rst_i 位宽：1bit

UPG reset (Active High)

upg_clk_i 位宽：1bit

UPG ram_clk_i (10MHz)

upg_wen_i 位宽：1bit

UPG write enable

upg_adr_i 位宽：14bit

UPG write address

upg_dat_i 位宽：32bit

UPG write data

upg_done_i 位宽：1bit

1 if programming is finished

输出：

read_data 位宽：32bit

data tobe read from memory unit

Switch

输入：

switclk(clock)　　位宽：1bit

时钟信号

switrst(rst)　　位宽：1bit

复位信号

switchread(IORead)　　位宽：1bit

读信号

switchcs(switchcs)　　位宽：1bit

从memorio来的switch片选信号

switchaddr(address[1:0])　　位宽：2bit

到switch模块的地址低端

switch_i(switch2N4)　　位宽：24bit

从板上读的24位开关数据

输出：

switchrdata(switchrdata)　　位宽：16bit

送到CPU的拨码开关值注意数据总线只有16根

在switclk的下降沿或switrst的上升沿时，根据情况赋值或更新switchrdata的值

Led

输入：

led_clk(clock)　　位宽：1bit

时钟信号

ledrst(rst) 位宽：1bit

复位信号

ledwrite(IOWrite) 位宽：1bit

写信号

ledcs(ledcs) 位宽：1bit

从memorio来的，由低至高位形成的LED片选信号

ledaddr(address[1:0]) 位宽：2bit

到LED模块的地址低端

ledwdata(write_data[16:0]) 位宽：17bit

写到LED模块的数据，注意数据线只有17根

输出：

ledout(led2N4) 位宽：24位

向板子上输出的24位LED信号

在led_clk的上升沿或ledrst的上升沿时，赋值或更新ledout的值。

display

输入：

led_clk(clock) 位宽：1bit

ledrst(rst) 位宽：1bit

ledwrite(IOWrite) 位宽：1bit

ledcs(ledcs) 位宽：1bit

ledaddr(address[1:0])　　位宽：2bit

ledwdata(write_data[31:0])　　位宽：32bit

输出：

Y(Y)　　位宽：8bit

DIG(DIG)　　位宽：8bit

在led_clk的上升沿或ledrst的上升沿时，赋值或更新ledtemp的值。

四．问题与总结

1．CPU顶层模块中，各个子模块连接错误。
2．第一个测试场景中，延迟1s显示需要两个循环，寄存器发生错误。
3．第二个测试场景中，七段数码显示管显示错误。
4．没有考虑延时问题，导致led灯看不到结果。
5．uart接口连接问题，子模块修改错误。
6．测试场景一中越界问题处理，加一个led灯显示更高位。