计算机组成原理实验2报告

PB20000156
徐亦昶

Task 1 寄存器堆

ppt中已有代码，但针对本题的条件，有两点不同：

- 需要将rf[0]单独设为0，并在always语句中避开对它的赋值。
- 注意本题是写有限，因此两部分always的时序和组合性需要调换一下（assign视为always@(*)）。
  最终代码：

```
`timescale 1ns / 1ps
module register_file #(
    parameter AW=5,
    parameter DW=32
)(
    input clk,
    input [AW-1:0] ra0,ra1,
    output reg[DW-1:0] rd0,rd1,
    input [AW-1:0] wa,
    input [DW-1:0] wd,
    input we
);
reg [DW-1:0] rf [0:(1<<AW)-1];
initial
rf[0]=0;
always@(*)
begin
    if(we && wa!=0) rf[wa]=wd;
end
always@(posedge clk)
begin
    rd0<=rf[ra0];
    rd1<=rf[ra1];
end
endmodule
```

仿真文件如下：

```
`timescale 1ns / 1ps
module test_bench();
reg clk,we;
reg[31:0] wd;
reg[4:0] ra0,ra1,wa;
wire[31:0] rd0,rd1;
register_file register_file(clk,ra0,ra1,rd0,rd1,wa,wd,we);
initial
begin
clk=0;
```

```
we=0;
ra0=0;
ra1=0;
wa=0;
wd=0;
end
initial
forever #1
clk=~clk;
initial
begin
#1 ra0=0;
ra1=0;
#1 ra0=5'b1;
#2 ra0=5'h0;
#1 ra1=5'b1;
end
initial
begin
#1 we=1'b1;
wd=32'b1;
#1 wa=1;
#4 wd=32'b10;
end
endmodule
```

本模块改变读取数据的地址并进行数据的写操作，仿真结果如下：

均符合要求。

Task 2 分布式和块存储器的研究和对比

读操作

仿真文件定义了一个分布式存储器和一个读优先的块存储器，会依次读取各个地址的数据。存储器是256*16的位的，存储的256个字节被初始化为从1到256的256个数。

```
`timescale 1ns / 1ps
module test_bench();
reg[7:0] a;
reg[15:0] d;
reg clk,we;
wire[15:0] spo;
wire[15:0] douta;

dist_mem_gen_0 distributed(.a(a),.d(d),.clk(clk),.we(we),.spo(spo));
block_read_fst block_r
(.clka(clk),.ena(1'b1),.wea(we),.addra(a),.dina(d),.douta(douta));

initial
begin
    clk=0;
    we=0;
    d=0;
```

```
    a=0;
end

initial
forever #1
clk=~clk;

initial
begin
    forever #5
    a=a+8'b1;
end
endmodule
```

可见相比于分布式存储器，块式存储器的数据读取有一定的延迟。

读优先和写优先的对比

```
`timescale 1ns / 1ps
module wf_or_rf();
reg clka,wea;
reg[15:0] dina;
reg[7:0] addra;
wire[15:0] douta_rf;
wire[15:0] douta_wf;
initial
begin
    clka=0;
    wea=0;
    dina=0;
    addra=0;
end

initial
forever #1
clka=~clka;

initial
begin
#4.3 wea=1'b1;
#4 wea=0;
end

initial
begin
#4.3 dina=16'h1111;
#3 dina=16'h2222;
end

initial
begin
#4.5 addra=8'h1;
#3 addra=8'h2;
```

```
#3 addra=8'h3;
#3 addra=8'h4;
end

block_read_fst block_r
(.clka(clka),.ena(1'b1),.wea(wea),.addra(addra),.dina(dina),.douta(douta_rf));
block_write_fst block_w
(.clka(clka),.ena(1'b1),.wea(wea),.addra(addra),.dina(dina),.douta(douta_wf));
endmodule
```

以两个存储器输出的不同为例来说明。在读优先存储器周期开始时，1111刚被写入，但因为是读优先，因此读出的是之前的旧数据0002，然后在写一个周期才转回1111；在写优先存储器中，1111被立刻写入，这时读出的就是新的数据1111。

排序电路

电路设计

数据通路：

其中driver表示控制信号的驱动模块，比如addr_driver即为地址信号的驱动。最上面是数码管的驱动，是按照FPGAOL的十六进制数码管来写的，后来发现FPGA用的是七段数码管，把它替换成了新的编码器。

各模块代码实现

本程序是模块化编写，并在每一个模块编写好即对其进行仿真。早期一些简单模块的仿真文件被覆盖，但保留了关键的仿真文件。
1、开关输入DPE模块：

```
//DPE.v
module DPE(
input clk,
input[15:0] x,
output reg[3:0] h,
output p);
wire[15:0] sw_edge,sw_clean;
jitter_clr jitter_clr(.clk(clk),.button(x),.button_clean(sw_clean));
double_edge double_edge(.clk(clk),.button(sw_clean),.button_edge(sw_edge));
assign p=|sw_edge;
always@(*)
begin
    if(sw_edge[15]==1'b1) h=4'hf;
    else if(sw_edge[14]==1'b1) h=4'he;
    else if(sw_edge[13]==1'b1) h=4'hd;
    else if(sw_edge[12]==1'b1) h=4'hc;
    else if(sw_edge[11]==1'b1) h=4'hb;
    else if(sw_edge[10]==1'b1) h=4'ha;
    else if(sw_edge[9]==1'b1) h=4'h9;
    else if(sw_edge[8]==1'b1) h=4'h8;
    else if(sw_edge[7]==1'b1) h=4'h7;
    else if(sw_edge[6]==1'b1) h=4'h6;
    else if(sw_edge[5]==1'b1) h=4'h5;
    else if(sw_edge[4]==1'b1) h=4'h4;
```

```
    else if(sw_edge[3]==1'b1) h=4'h3;
    else if(sw_edge[2]==1'b1) h=4'h2;
    else if(sw_edge[1]==1'b1) h=4'h1;
    else if(sw_edge[0]==1'b0) h=4'h0;
    else h=4'h0;
end
endmodule
```

其中包括用于去抖动的jitter_clr模块和用于取双边沿的double_edge模块：

```
//jitter_clr.v
module jitter_clr(
input clk,
input[15:0] button,
output[15:0] button_clean);
_jitter_clr _jitter_clr0(clk,button[0],button_clean[0]);
_jitter_clr _jitter_clr1(clk,button[1],button_clean[1]);
_jitter_clr _jitter_clr2(clk,button[2],button_clean[2]);
_jitter_clr _jitter_clr3(clk,button[3],button_clean[3]);
_jitter_clr _jitter_clr4(clk,button[4],button_clean[4]);
_jitter_clr _jitter_clr5(clk,button[5],button_clean[5]);
_jitter_clr _jitter_clr6(clk,button[6],button_clean[6]);
_jitter_clr _jitter_clr7(clk,button[7],button_clean[7]);
_jitter_clr _jitter_clr8(clk,button[8],button_clean[8]);
_jitter_clr _jitter_clr9(clk,button[9],button_clean[9]);
_jitter_clr _jitter_clr10(clk,button[10],button_clean[10]);
_jitter_clr _jitter_clr11(clk,button[11],button_clean[11]);
_jitter_clr _jitter_clr12(clk,button[12],button_clean[12]);
_jitter_clr _jitter_clr13(clk,button[13],button_clean[13]);
_jitter_clr _jitter_clr14(clk,button[14],button_clean[14]);
_jitter_clr _jitter_clr15(clk,button[15],button_clean[15]);
endmodule
```

```
//double_edge.v
module double_edge(
input clk,
input[15:0] button,
output[15:0] button_edge);
reg[15:0] button_r1_pos,button_r1_neg,button_r2_pos,button_r2_neg;
always@(posedge clk)
begin
button_r1_pos<=button;
button_r1_neg<=~button;
end
always@(posedge clk)
begin
button_r2_pos<=button_r1_pos;
button_r2_neg<=button_r1_neg;
end
assign button_edge=button_r1_pos&~button_r2_pos|button_r1_neg&~button_r2_neg;
endmodule
```

在jitter_clr中，包含去除单个按键抖动的原子模块。原子模块的命名在原模块上加下划线加以区分。

```
//_jitter_clr.v
module _jitter_clr(
input clk,
input button,
output reg button_clean);
reg[1:0] cnt;
reg button_before;
initial
begin
cnt=0;
button_before=0;
end
always@(posedge clk)
button_before<=button;
always@(posedge clk)
begin
    if(button!=button_before)
        cnt<=4'h0;
    else if(cnt<2'h3)
        cnt<=cnt+1'b1;
end
always@(*)
if(cnt[1]==1'b1) button_clean=button;
endmodule
```

DPE模块会输出一个最近的按键h和脉冲p，其中p仅会在有按键状态改变后存在一周期。
2、普通按钮处理模块button_process
包括按钮的去抖动button_jitter_clr和取单边沿button_edge两个子模块。

```
//button_processs.v
module button_process(
input clk,btn,
output btn_p);
wire btn_clean;
button_jitter_clr button_jitter_clr(clk,btn,btn_clean);
button_edge button_edge(clk,btn_clean,btn_p);
endmodule
```

```
//jitter_clr.v
module button_jitter_clr(
input clk,
input button,
output button_clean);
reg[1:0] cnt;
initial
    cnt=0;
always@(posedge clk)
begin
    if(button==0)
        cnt<=4'h0;
    else if(cnt<2'h3)
        cnt<=cnt+1'b1;
end
```

```
assign button_clean=cnt[1];
endmodule
```

```
//button_edge.v
module button_edge(
input clk,
input button,
output button_edge);
reg button_r1,button_r2;
always@(posedge clk)
    button_r1 <= button;
always@(posedge clk)
    button_r2 <= button_r1;
assign button_edge = button_r1 & (~button_r2);
endmodule
```

3、两个数据选择器mux1和mux16，mux后的数字表示位数。

```
//mux1.v
module mux1(
input sr0,sr1,sel,
output d);
assign d=sel?sr1:sr0;
endmodule
```

```
//mux16.v
module mux16(
input[15:0] sr0,sr1,
input sel,
output[15:0] d);
assign d=sel?sr1:sr0;
endmodule
```

3、存储器模块：使用但端口分布式RAM存储器，命名为memory。
4、S_driver：控制mux16的数据选择信号：

```
//S_driver.v
module S_driver(
input clk,rstn,chk_p,p,del_p,data_p,addr_p,
output reg S);
always@(posedge clk)
begin
    if(rstn==0)
        S<=0;
    else
    begin
        if(chk_p)
            S<=0;
        else if(p)
            S<=1'b1;
        else if(del_p)
            S<=1'b1;
```

```
        else if(data_p)
            S<=0;
        else if(addr_p)
            S<=0;
    end
end
endmodule
```

仿真文件：

```
//test_S_driver.v
module test_S_driver();
reg clk,rstn,chk_p,p,del_p,data_p,addr_p;
wire S;

initial
begin
clk=0;
rstn=0;
data_p=0;
p=0;
chk_p=0;
addr_p=0;
del_p=0;
end

initial
forever #1
clk=~clk;

initial
#2 rstn=1'b1;

initial
begin
#3 chk_p=1'b1;
#2 chk_p=0;
#2 p=1'b1;
#2 p=0;
#2 data_p=1'b1;
#2 data_p=0;
#2 del_p=1'b1;
#2 del_p=0;
#2 addr_p=1'b1;
#2 addr_p=0;
end

S_driver S_driver(clk,rstn,chk_p,p,del_p,data_p,addr_p,S);
endmodule
```

仿真结果：

结果正确。

5、data_driver：选择合适的数据源并输出。

```
//data_driver.v
module data_driver(
input clk,busy,p,rstn,del_p,data_p,addr_p,
input[3:0] h,
input[15:0] sort_d,
output reg[15:0] d);
always@(posedge clk)
begin
    if(rstn==0)
        d<=0;
    else
        begin
        if(busy==0)
        begin
            if(p)
                d<={d[11:0],h};
            else if(del_p)
                d<={4'b0,d[15:4]};
            else if(addr_p|data_p)
                d<=0;
        end
        else
            d<=sort_d;
    end
end
endmodule
```

仿真文件:

```
//test_data_driver.v
module test_data_driver();
reg clk,busy,p,rstn,del_p,data_p,addr_p;
reg[3:0] h;
reg[15:0] sort_d;
wire[15:0] d;

initial
begin
clk=0;
busy=0;
p=0;
rstn=0;
del_p=0;
data_p=0;
addr_p=0;
h=0;
sort_d=16'h123c;
end

initial
forever #1
clk=~clk;

initial
```

```
#2 rstn=1'b1;

initial
begin
#3 p=1'b1;
data_p=1'b1;
#2 p=0;
data_p=0;
#2 p=1'b1;
data_p=1'b1;
#2 p=0;
data_p=0;
end

initial
begin
#3 h=4'he;
#4 h=4'h1;
end

initial
begin
#9 del_p=1'b1;
#2 del_p=0;
end

initial
#13 busy=1'b1;

data_driver data_driver(clk,busy,p,rstn,del_p,data_p,addr_p,h,sort_d,d);
endmodule
```

仿真结果：

先模拟按键输入和删除，再模拟排序核心输入，结果均无误。
6、sort_core：排序的核心，本质上是一个7状态图灵机。它和外围模块通信进行数据的传输，并在模块内部对数据进行排序。

```
//sort_core.v
module sort_core(
input clk,rstn,run_p,
input[15:0] spo,
output reg[15:0] cnt,sort_d,
output reg[7:0] sort_addr,
output reg sort_we,busy);
reg[7:0] i,j;
reg[2:0] cs,ns;
reg[15:0] copy[0:255];
always@(posedge clk) //cs
begin
    if(rstn==0)
        cs<=0;
    else
        cs<=ns;
```

```
end

always@(*) //ns
begin
    if(rstn==0)
        ns<=0;
    else
    begin
        case(cs)
        3'b000:
        begin
            if(run_p)
                ns<=3'b001;
            else
                ns<=3'b000;
        end
        3'b001:
            ns<=3'b010;
        3'b010:
        begin
            if(sort_addr==8'b1111_1111)
                ns<=3'b011;
            else
                ns<=3'b010;
        end
        3'b011:
        begin
            if(i==8'b1111_1110)
                ns<=3'b100;
            else
                ns<=3'b011;
        end
        3'b100:
            ns<=3'b101;
        3'b101:
        begin
            if(sort_addr==8'b1111_1111)
                ns<=3'b110;
            else
                ns<=3'b101;
        end
        3'b110:
            ns<=3'b000;
        endcase
    end
end
always@(posedge clk) //cnt
begin
    if(rstn==0)
        cnt<=0;
    else
    begin
        if(cs==3'b000)
            cnt<=cnt;
        else if(cs==3'b001)
```

```
            cnt<=0;
        else
            cnt<=cnt+1;
    end
end
always@(posedge clk) //i,j
begin
    if(rstn==0)
    begin
        i<=0;
        j<=8'b1;
    end
    case(cs)
        3'b001:
        begin
            i<=0;
            j<=8'b1;
        end
        3'b011:
        begin
            if(j==8'b1111_1111)
            begin
                if(i!=8'b1111_1110)
                begin
                    i<=i+1;
                    j<=i+2;
                end
            end
            else
                j<=j+1;
        end
   endcase
end
always@(*) //sort_we
begin
    if(rstn==0)
        sort_we=0;
    else
    begin
        case(cs)
        3'b000:
            sort_we=0;
        3'b100:
            sort_we=1'b1;
        3'b110:
            sort_we=0;
        endcase
    end
end
always@(posedge clk) //sort_addr
begin
    if(rstn==0)
        sort_addr<=0;
    else
    begin
```

```
        case(cs)
        3'b001:
            sort_addr<=0;
        3'b010:
                sort_addr<=sort_addr+1;
        3'b100:
            sort_addr<=0;
        3'b101:
                sort_addr<=sort_addr+1;
        endcase
    end
end
always@(posedge clk) //copy
begin
    case(cs)
    3'b010:
        copy[sort_addr]<=spo;
    3'b011:
    begin
        if(copy[i]>copy[j])
        begin
            copy[i]<=copy[j];
            copy[j]<=copy[i];
        end
    end
    endcase
end
always@(posedge clk) //sort_d
begin
    if(rstn==0)
        sort_d<=0;
    else
    begin
        case(cs)
            3'b000:
                sort_d<=0;
            3'b100:
                sort_d<=copy[0];
            3'b101:
                sort_d<=copy[sort_addr+1];
            default:
                sort_d<=0;
        endcase
    end
end
always@(posedge clk) //busy
begin
    if(rstn==0)
        busy<=0;
    else
    begin
        if(cs==3'b001)
            busy<=1'b1;
        else if(cs==3'b000)
            busy<=0;
```

```
    end
end
endmodule
```

这里简略介绍七个状态。

- 状态1：初始状态。run按钮按下前一直在这个状态。
- 状态2：为状态3的读数据设置使能位并初始化循环变量。
- 状态3：读入存储器的数据，保存在模块内部的寄存器中。
- 状态4：选取两个数据进行比较操作，即进行排序。该状态需要特别多的时钟周期才能转移到下一个状态。
- 状态5：为状态6的写操作设置使能位并初始化。
- 状态6：讲排好序的数据写回存储器。
- 状态7：恢复各项设置，如将写使能置回0，随后转移到状态1。

仿真文件：

```
test_sort_core.v
`timescale 1ns / 1ps
module test_sort_core();
reg clk,rstn,run_p;
wire[15:0] spo;
reg[15:0] cnt,sort_d;
reg[7:0] sort_addr;
reg sort_we,busy;
initial
begin
clk=0;
rstn=0;
run_p=0;
end
initial
forever #1
clk=~clk;
initial
#2 rstn=1'b1;
initial
begin
    #3 run_p=1'b1;
    #2 run_p=0;
end
memory memory(.clk(clk),.a(sort_addr),.spo(spo),.we(sort_we),.d(sort_d));

reg[7:0] i,j;
reg[2:0] cs,ns;
reg[15:0] copy[0:255];
always@(posedge clk) //cs
begin
    if(rstn==0)
        cs<=0;
    else
        cs<=ns;
end
```

```
always@(*) //ns
begin
    if(rstn==0)
        ns<=0;
    else
    begin
        case(cs)
        3'b000:
        begin
            if(run_p)
                ns<=3'b001;
            else
                ns<=3'b000;
        end
        3'b001:
            ns<=3'b010;
        3'b010:
        begin
            if(sort_addr==8'b1111_1111)
                ns<=3'b011;
            else
                ns<=3'b010;
        end
        3'b011:
        begin
            if(i==8'b1111_1110)
                ns<=3'b100;
            else
                ns<=3'b011;
        end
        3'b100:
            ns<=3'b101;
        3'b101:
        begin
            if(sort_addr==8'b1111_1111)
                ns<=3'b110;
            else
                ns<=3'b101;
        end
        3'b110:
            ns<=3'b000;
        endcase
    end
end
always@(posedge clk) //cnt
begin
    if(rstn==0)
        cnt<=0;
    else
    begin
        if(cs==3'b000)
            cnt<=cnt;
        else if(cs==3'b001)
            cnt<=0;
        else
```

```
            cnt<=cnt+1;
    end
end
always@(posedge clk) //i,j
begin
    if(rstn==0)
    begin
        i<=0;
        j<=8'b1;
    end
    case(cs)
        3'b001:
        begin
            i<=0;
            j<=8'b1;
        end
        3'b011:
        begin
            if(j==8'b1111_1111)
            begin
                if(i!=8'b1111_1110)
                begin
                    i<=i+1;
                    j<=i+2;
                end
            end
            else
                j<=j+1;
        end
   endcase
end
always@(*) //sort_we
begin
    if(rstn==0)
        sort_we=0;
    else
    begin
        case(cs)
        3'b000:
            sort_we=0;
        3'b100:
            sort_we=1'b1;
        3'b110:
            sort_we=0;
        endcase
    end
end
always@(posedge clk) //sort_addr
begin
    if(rstn==0)
        sort_addr<=0;
    else
    begin
        case(cs)
        3'b001:
```

```
            sort_addr<=0;
        3'b010:
                sort_addr<=sort_addr+1;
        3'b100:
            sort_addr<=0;
        3'b101:
                sort_addr<=sort_addr+1;
        endcase
    end
end
always@(posedge clk) //copy
begin
    case(cs)
    3'b010:
        copy[sort_addr]<=spo;
    3'b011:
    begin
        if(copy[i]>copy[j])
        begin
            copy[i]<=copy[j];
            copy[j]<=copy[i];
        end
    end
    endcase
end
always@(posedge clk) //sort_d
begin
    if(rstn==0)
        sort_d<=0;
    else
    begin
        case(cs)
            3'b000:
                sort_d<=0;
            3'b100:
                sort_d<=copy[0];
            3'b101:
                sort_d<=copy[sort_addr+1];
            default:
                sort_d<=0;
        endcase
    end
end
always@(posedge clk) //busy
begin
    if(rstn==0)
        busy<=0;
    else
    begin
        if(cs==3'b001)
            busy<=1'b1;
        else if(cs==3'b000)
            busy<=0;
    end
end
```

```
endmodule
```

为了看清各个寄存器的值，直接把sort_core的代码复制到了仿真文件。该仿真文件模拟了一次排序，时钟周期太长，故省略。已将仿真波形图一个状态一个状态地比对，运行无误。
7、sort：排序主模块，按照数据通路将各模块拼接。

```
module sort(
input clk,rstn,
input[15:0] x,
input del,
input addr,
input data,
input chk,
input run,
output busy,
output [7:0] an,
output [6:0] seg,
output[15:0] cnt);
wire data_p,chk_p,addr_p,sort_we,we,S,p,del_p,run_p;
wire[7:0] sort_addr,addr_out;
wire[15:0] seg_d;
wire[3:0] h;
wire[15:0] d,spo,sort_d;
button_process button_process1(clk,data,data_p);
button_process button_process2(clk,chk,chk_p);
button_process button_process3(clk,addr,addr_p);
button_process button_process4(clk,del,del_p);
button_process button_process5(clk,run,run_p);
addr_driver
addr_driver(.clk(clk),.busy(busy),.rstn(rstn),.data_p(data_p),.chk_p(chk_p),.addr_p
(addr_p),.sort_addr(sort_addr),.d(d),.addr(addr_out));
mux1 mux1(.sr0(data_p),.sr1(sort_we),.sel(busy),.d(we));
memory memory(.a(addr_out),.clk(clk),.we(we),.d(d),.spo(spo));
mux16 mux16(.sr0(spo),.sr1(d),.sel(S),.d(seg_d));
S_driver
S_driver(.rstn(rstn),.chk_p(chk_p),.p(p),.del_p(del_p),.data_p(data_p),.addr_p(addr
_p),.clk(clk),.S(S));
data_driver
data_driver(.addr_p(addr_p),.h(h),.p(p),.rstn(rstn),.del_p(del_p),.sort_d(sort_d),.
data_p(data_p),.busy(busy),.clk(clk),.d(d));
sort_core
sort_core(.spo(spo),.rstn(rstn),.run_p(run_p),.clk(clk),.cnt(cnt),.sort_d(sort_d),.
sort_addr(sort_addr),.sort_we(sort_we),.busy(busy));
DPE DPE(.x(x),.h(h),.p(p),.clk(clk));
seg_driver seg_driver(clk,rstn,addr_out,seg_d,an,seg);
endmodule
```

在尽可能消除遇到的警告后，直接上板运行成功。

遇到的困难

- 状态机的cs和ns延时造成死循环。比如cs为1时，ns设为2，下一个时钟周期cs变成2，但部分条件改变，ns依赖于上一个周期的cs=1进行调整，改回1，这样导致死循环。解决方法是ns应该用always@(*)表示而不是always@(posedge clk)。