

实验三 模型机组合部件的实现（二）

班级 信安 2101 班 姓名 孙照海 学号 202109070105

一、实验目的

1. 了解简易模型机的内部结构和工作原理。
2. 分析模型机的功能，设计 8 重 3-1 多路复用器。
3. 分析模型机的功能，设计移位逻辑。
4. 分析模型机的工作原理，设计模型机控制信号产生逻辑。

二、实验内容

1. 用 VERILOG 语言设计模型机的 8 重 3-1 多路复用器；
2. 用 VERILOG 语言设计模型机的移位模块；
3. 用 VERILOG 语言设计模型机的控制信号产生逻辑。

三、实验过程

1、8 重 3-1 多路复用器

A) 创建工程（选择的芯片为 family=Cyclone II；name=EP2C5T144C8）

步骤：左上角 file->New Project Wizard->选择工程位置和工程名->选择芯片 Cyclone II，available device 中选择 EP2C5T144C8->点击 next->最后点击 finish 完成创建工程

工程创建图：



B) 编写源代码

根据实验指导和要求实现的功能写出对应的 Verilog 代码。

步骤：左上角 file->new->Verilog hdl file->编写代码（模块名需与工程名一致）->编译成功后保存到工程文件中。

代码截图：

```

1 module mux3_1(a,b,c,madd,y);
2     input [7:0] a,b,c;
3     input [1:0] madd;
4     output reg[7:0] y;
5
6     always @(*) //always模块中的任何一个输入信号或电平发生变化时，该语句下方的模块将被执行
7     begin
8         if(madd==2'b00) y=a;
9         else if(madd==2'b01) y=b;
10        else if(madd==2'b10) y=c;
11        else y=8'hZZ;
12    end
13 endmodule

```

C) 编译与调试（包含编译调试过程中的错误、警告信息以及资源消耗）

确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译，编译成功，保存文件。

无错误。

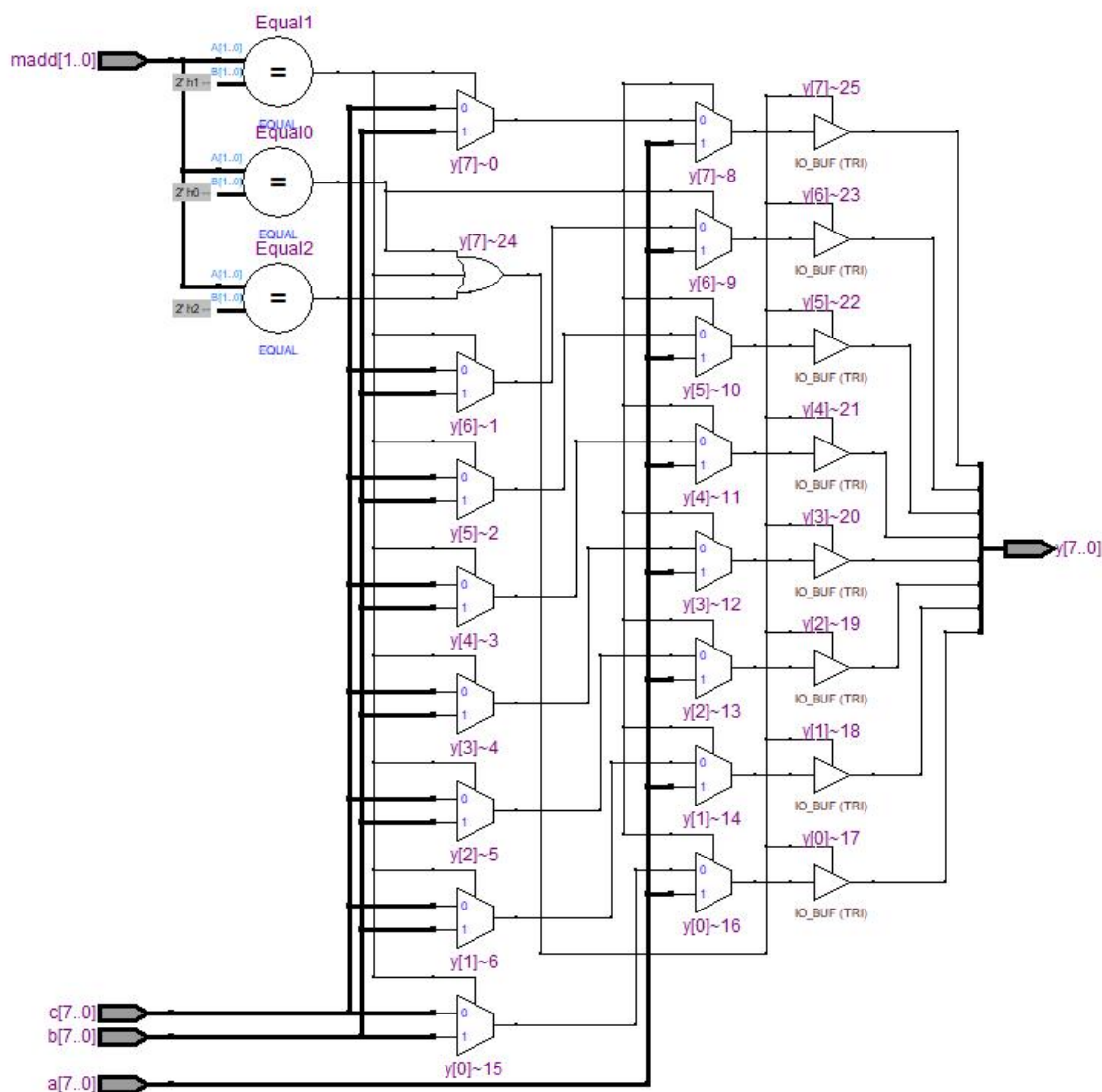
警告信息：

Type	Message
Warning	Warning: Feature LogicLock is not available with your current license
Warning	Warning: No exact pin location assignment(s) for 34 pins of 34 total pins
Warning	Warning: Found 8 output pins without output pin load capacitance assignment
Warning	Warning: The Reserve All Unused Pins setting has not been specified, and will default to 'As output driving ground'.

资源消耗：

Flow Status	Successful - Mon Nov 21 22:43:18 2022
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	mux3_1
Top-level Entity Name	mux3_1
Family	Cyclone II
Device	EP2C5T144C8
Timing Models	Final
Met timing requirements	Yes
Total logic elements	17 / 4,608 (< 1 %)
Total combinational functions	17 / 4,608 (< 1 %)
Dedicated logic registers	0 / 4,608 (0 %)
Total registers	0
Total pins	34 / 89 (38 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)

D) RTL 视图



视图分析及结论：

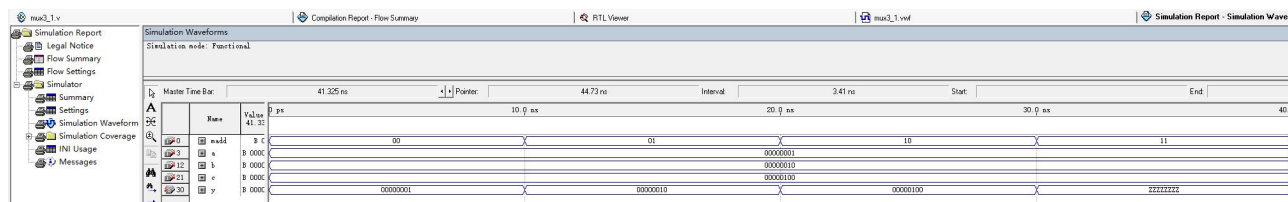
视图分析：视图左边为输入，右边为输出。其中连接有大量的元器件。例如比较器：当输入相等时输出 1，不相等时输出 0；还由大量的 2-1 选择器构成，当控制信号为 0 时，输出第一位，控制信号 1 时，输出第二位。图中输入信号为 `madd` 和 `a`、`b`、`c`，输出信号为 `y`。各个输出端口之间通过导线相连。

结论：8 重 3-1 多路复用器功能需要经过多重门的处理后才能实现，其内部原理结构图十分复杂。

E) 功能仿真波形

步骤：新建一个 vector waveform file。通过操作：右击 -【insert】-【insert node or bus】-【node finder】(pins=all; 【list】) -【>>】-【ok】-【ok】，在【assignments】-【setting】-【simulator settings】-【simulation mode】中，将 `timing` 修改为 `functional`，然后再点击【processing】-

【Generate Functional Simulation Netlist】，接着设置输入波形，成功后点击【start simulation】开始仿真，查看输出功能仿真波形图。
功能仿真波形图：



结果分析及结论：

结果分析：功能仿真是指不考虑器件延时和布线延时的理想情况下对源代码进行逻辑功能的验证。由仿真波形可得，对于输入状态的变化，输出结果实时变化，没有延迟，其结果与电路设计的真值表的结果相对应。

madd=00 时，控制输出 y 等于 a，正确；

madd=01 时，控制输出 y 等于 b，正确；

madd=10 时，控制输出 y 等于 c，正确；

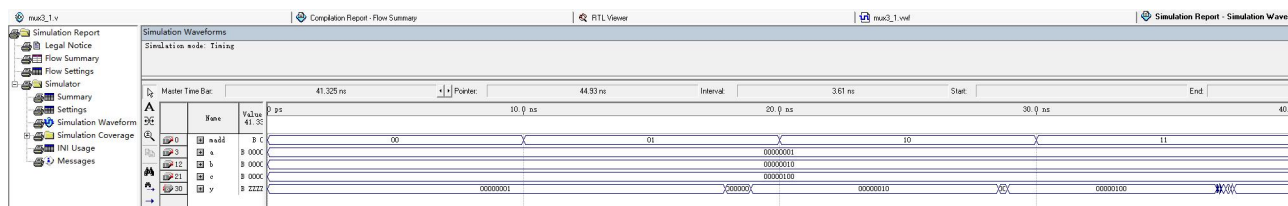
madd=11 时，控制输出高阻态，正确；

结论：功能仿真是指在一个设计中，在设计实现前对所创建的逻辑进行的验证其功能是否正确。布局布线以前的仿真都称作功能仿真，它包括综合前仿真（Pre-Synthesis Simulation）和综合后仿真（Post-Synthesis Simulation）。综合前仿真主要针对基于原理框图的设计；综合后仿真既适合原理图设计，也适合基于 HDL 语言的设计。功能仿真操作简单，能体现和验证实验的功能，但忽略延迟的影响会使结果与实际结果，有一定误差。

F) 时序仿真波形

步骤：新建一个 vector waveform file。通过操作：右击 - 【insert】 - 【insert node or bus】 - 【node finder】（pins=all; 【list】） - 【>>】 - 【ok】 - 【ok】。接着设置输入波形，然后点击【start simulation】开始仿真，查看时序仿真输出波形图。

时序仿真图：



结果分析及结论：

结果分析：时序仿真是指在布线后进行，是最接近真实器件运行的仿真，它与特定的器件有关，又包含了器件和布线的延时信息。由波形可得，当输入状态发生改变时，输出结果并未同时改变，而是有一定延迟，同时由于输入状态的变化，导致电路出现“冒险”，导致输出结果并未与预期结果相同。

结论：时序仿真使用布局布线后器件给出的模块和连线的延时信息，在最坏的情况下对电路的行为作出实际地估价。时序仿真使用的仿真器和功能仿真使用的仿真器是相同的，所需的流程和激励也是相同的；唯一的差别是为时序仿真加载到仿真器的设计包括基于实际布局布

线设计的最坏情况的布局布线延时，并且在仿真结果波形图中，时序仿真后的信号加载了时延，而功能仿真没有。时序仿真可以用来验证程序在目标器件中的时序关系。同时考虑了器件的延迟后，其输出结果跟接近实际情况，但是考虑的情况过多，不容易操作，容易产生错误。时序仿真不仅反应出输出和输入的逻辑关系，同时还计算了时间的延时信息，是与实际系统更接近的一种仿真结果。不过，要注意的是，这个时间延时是仿真软件“估算”出来的。

G) 时序分析

操作方法是：编译后，在 compilation report 中选择【timing analysis】-【summary】和【tpd】

Timing Analyzer Summary									
	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	15.940 ns	a[7]	y[7]	--	--	0
2	Total number of failed paths								0

tpd					
	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	15.940 ns	a[7]	y[7]
2	N/A	None	15.244 ns	b[7]	y[7]
3	N/A	None	14.364 ns	a[2]	y[2]
4	N/A	None	13.873 ns	a[6]	y[6]
5	N/A	None	13.721 ns	b[4]	y[4]
6	N/A	None	13.695 ns	b[1]	y[1]
7	N/A	None	13.661 ns	a[3]	y[3]
8	N/A	None	13.520 ns	b[2]	y[2]
9	N/A	None	13.392 ns	a[1]	y[1]
10	N/A	None	13.143 ns	c[1]	y[1]
11	N/A	None	13.055 ns	a[4]	y[4]
12	N/A	None	12.975 ns	a[5]	y[5]
13	N/A	None	12.970 ns	b[3]	y[3]
14	N/A	None	12.893 ns	c[4]	y[4]
15	N/A	None	12.808 ns	c[7]	y[7]
16	N/A	None	12.784 ns	b[5]	y[5]
17	N/A	None	12.772 ns	c[2]	y[2]
18	N/A	None	12.650 ns	b[6]	y[6]
19	N/A	None	12.585 ns	c[6]	y[6]
20	N/A	None	12.416 ns	c[3]	y[3]
21	N/A	None	12.403 ns	c[0]	y[0]
22	N/A	None	12.004 ns	c[5]	y[5]
23	N/A	None	10.761 ns	madd[1]	y[7]
24	N/A	None	10.384 ns	madd[0]	y[7]
25	N/A	None	9.094 ns	madd[1]	y[4]
26	N/A	None	9.017 ns	madd[1]	y[2]
27	N/A	None	8.942 ns	madd[0]	y[2]
28	N/A	None	8.871 ns	madd[0]	y[1]
29	N/A	None	8.714 ns	madd[0]	y[4]
30	N/A	None	8.697 ns	madd[1]	y[6]
31	N/A	None	8.599 ns	madd[1]	y[1]
32	N/A	None	8.555 ns	madd[0]	y[3]
33	N/A	None	8.325 ns	madd[0]	y[6]
34	N/A	None	8.283 ns	madd[1]	y[3]
35	N/A	None	8.193 ns	madd[1]	y[5]
36	N/A	None	8.054 ns	a[0]	y[0]
37	N/A	None	8.012 ns	b[0]	y[0]
38	N/A	None	7.887 ns	madd[1]	y[0]
39	N/A	None	7.858 ns	madd[0]	y[0]
40	N/A	None	7.821 ns	madd[0]	y[5]

结果分析及结论：

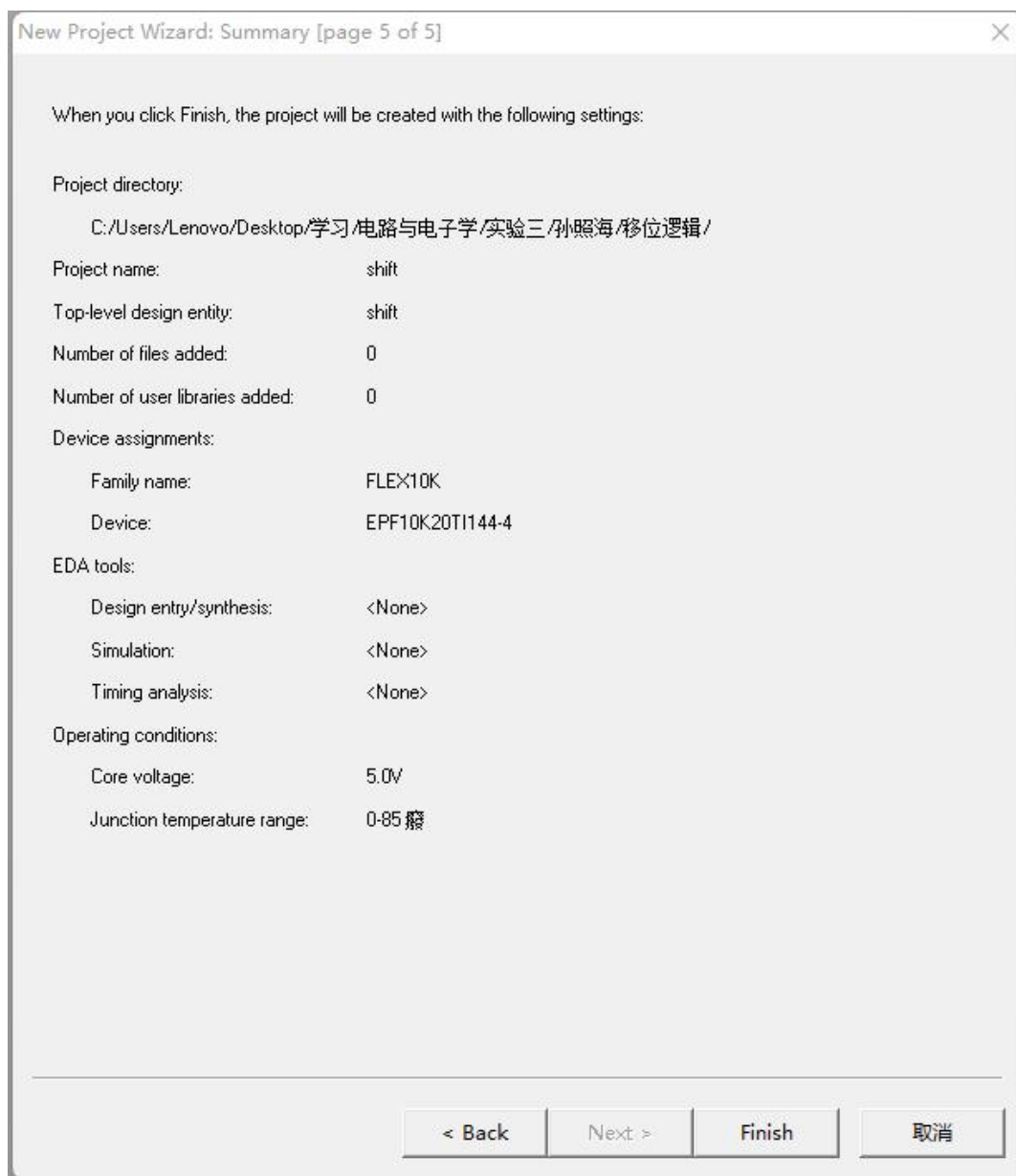
结果分析：由图可得，Timing Analyzer Summary 总结所有经典定时分析的结果，并报告每个定时特性的最坏情况定时。比如从 a[7]到 y[7]的最坏定时情况的 tpd 为 15.940ns。下面的 tpd 报告表则给出了源节点和目标节点之间的 tpd 延迟时间，比如第二行中 b[7]到 y[7]的 tpd 为 15.244ns。

结论：实际连接图中个元器件连接之间是存在时间延迟的，而且不同的元器件之间的时间延迟也不相同。

2、移位逻辑

A) 创建工程（选择的芯片为 family=FLEX10K; name=EPF10K20T1144-4）

步骤：左上角 file->New Project Wizard->选择工程位置和工程名->选择芯片 FLEX10K, available device 中选择 EPF10K20T1144-4->点击 next->最后点击 finish 完成创建工程
工程创建图：



B) 编写源代码

根据实验指导和要求实现的功能写出对应的 Verilog 代码。

步骤：左上角 file->new->Verilog hdl file->编写代码（模块名需与工程名一致）->编译成功后保存到工程文件中。

代码截图：

```

1  module shift(fbush,flbush,frbush,a,w,cf);
2  input wire fbush,flbush,frbush;
3  input wire[7:0] a;
4  output reg[7:0] w;
5  output reg cf;
6
7  always @(*)
8  begin
9      cf=0;
10     if(fbush==1'b1&&flbush==1'b0&&frbush==1'b0) w=a;
11     else if(fbush==1'b0&&flbush==1'b1&&frbush==1'b0)
12     begin
13         w[7:1]=a[6:0];
14         w[0]=a[7];
15         cf=a[7];
16     end
17
18     else if(fbush==1'b0&&flbush==1'b0&&frbush==1'b1)
19     begin
20         w[6:0]=a[7:1];
21         w[7]=a[0];
22         cf=a[0];
23     end
24
25     else w=8'hZZ;
26 end
27 endmodule

```

C) 编译与调试（包含编译调试过程中的错误、警告信息以及资源消耗）

确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译，编译成功，保存文件。

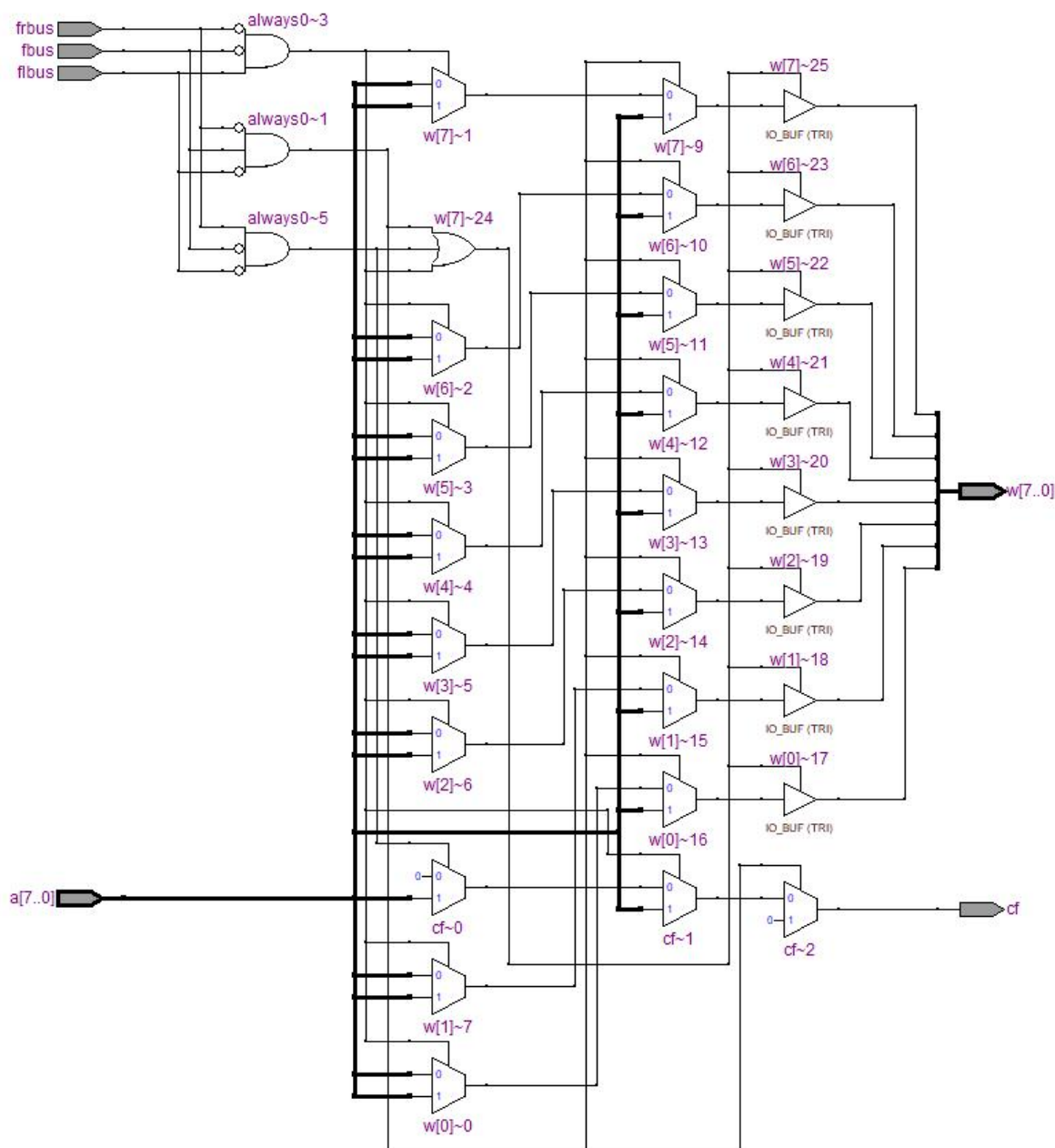
无错误。

无警告信息：

资源消耗：

Flow Status	Successful - Mon Nov 21 23:28:20 2022
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	shift
Top-level Entity Name	shift
Family	FLEX10K
Device	EPF10K20TI144-4
Timing Models	Final
Met timing requirements	Yes
Total logic elements	21 / 1,152 (2 %)
Total pins	20 / 102 (20 %)
Total memory bits	0 / 12,288 (0 %)

D) RTL 视图



视图分析及结论：

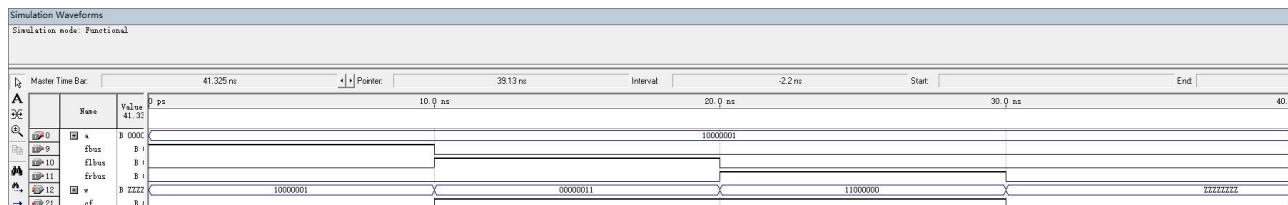
视图分析：视图左边为输入，右边为输出。其中连接有大量的元器件。例如比较器：当输入相等时输出 1，不相等时输出 0；还由大量的 2-1 选择器构成，当控制信号为 0 时，输出第一位，控制信号 1 时，输出第二位。图中输入信号为 fbus, frbus, flbus 和 a，输出信号为 w。各个输出端口之间通过导线相连。

结论：移位逻辑功能需要经过多重门的处理后才能实现，其内部原理结构图十分复杂。

E) 功能仿真波形

步骤：新建一个 vector waveform file。通过操作：右击 -【insert】-【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】，在【assignments】-【setting】-【simulator settings】-【simulation mode】中，将 timing 修改为 functional，然后再点击【processing】-【Generate Functional Simulation Netlist】，接着设置输入波形，成功后点击【start simulation】开始仿真，查看输出功能仿真波形图。

功能仿真波形图：



结果分析及结论：

结果分析：功能仿真是指不考虑器件延时和布线延时的理想情况下对源代码进行逻辑功能的验证。由仿真波形可得，对于输入状态的变化，输出结果实时变化，没有延迟，其结果与电路设计的真值表的结果相对应。

当 fbus=1, frbus=0, flbus=0, 不执行移位操作，输出等于输入，cf 不改变；

当 fbus=0, frbus=1, flbus=0, 执行右移，输出等于输入右移移位，有进位的话 cf 为 1；

当 fbus=0, frbus=0, flbus=1, 执行左移，输出等于输入左移一位，cf 不改变；

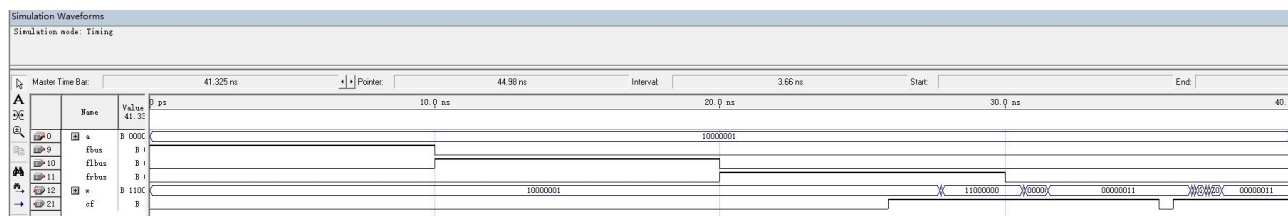
当 fbus=0, frbus=0, flbus=0, 输出为高阻态，正确；

结论：功能仿真是指在一个设计中，在设计实现前对所创建的逻辑进行的验证其功能是否正确。布局布线以前的仿真都称作功能仿真，它包括综合前仿真（Pre-Synthesis Simulation）和综合后仿真（Post-Synthesis Simulation）。综合前仿真主要针对基于原理框图的设计；综合后仿真既适合原理图设计，也适合基于 HDL 语言的设计。功能仿真操作简单，能体现和验证实验的功能，但忽略延迟的影响会使结果与实际结果，有一定误差。

F) 时序仿真波形

步骤：新建一个 vector waveform file。通过操作：右击 -【insert】-【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】。接着设置输入波形，然后点击【start simulation】开始仿真，查看时序仿真输出波形图。

时序仿真图：



结果分析及结论：

结果分析：时序仿真是指在布线后进行，是最接近真实器件运行的仿真，它与特定的器件有关，又包含了器件和布线的延时信息。由波形可得，当输入状态发生改变时，输出结果并未同时改变，而是有一定延迟，同时由于输入状态的改变，导致电路出现“冒险”，导致输出

结果并未与预期结果相同。

结论：时序仿真使用布局布线后器件给出的模块和连线的延时信息，在最坏的情况下对电路的行为作出实际地估价。时序仿真使用的仿真器和功能仿真使用的仿真器是相同的，所需的流程和激励也是相同的；唯一的差别是为时序仿真加载到仿真器的设计包括基于实际布局布线设计的最坏情况的布局布线延时，并且在仿真结果波形图中，时序仿真后的信号加载了时延，而功能仿真没有。时序仿真可以用来验证程序在目标器件中的时序关系。同时考虑了器件的延迟后，其输出结果跟接近实际情况，但是考虑的情况过多，不容易操作，容易产生错误。时序仿真不仅反应出输出和输入的逻辑关系，同时还计算了时间的延时信息，是与实际系统更接近的一种仿真结果。不过，要注意的是，这个时间延时是仿真软件“估算”出来的。

G) 时序分析

操作方法是：编译后，在 compilation report 中选择【timing analysis】-【summary】和【tpd】

Timing Analyzer Summary									
	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	21.700 ns	fbus	w[5]	--	--	0
2	Total number of failed paths								0

tpd					
	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	21.700 ns	fbus	w[5]
2	N/A	None	21.700 ns	fbus	w[5]
3	N/A	None	21.500 ns	fbus	w[1]
4	N/A	None	21.500 ns	fbus	w[1]
5	N/A	None	21.200 ns	fbus	w[5]
6	N/A	None	21.000 ns	fbus	w[1]
7	N/A	None	20.900 ns	fbus	w[3]
8	N/A	None	20.900 ns	fbus	w[3]
9	N/A	None	20.700 ns	fbus	w[7]
10	N/A	None	20.700 ns	fbus	w[7]
11	N/A	None	20.700 ns	fbus	w[2]
12	N/A	None	20.700 ns	fbus	w[2]
13	N/A	None	20.700 ns	fbus	w[0]
14	N/A	None	20.700 ns	fbus	w[0]
15	N/A	None	20.600 ns	fbus	w[6]
16	N/A	None	20.600 ns	fbus	w[6]
17	N/A	None	20.600 ns	fbus	w[4]
18	N/A	None	20.600 ns	fbus	w[4]
19	N/A	None	20.400 ns	fbus	w[3]
20	N/A	None	20.200 ns	fbus	w[7]
21	N/A	None	20.200 ns	fbus	w[2]
22	N/A	None	20.200 ns	fbus	w[0]
23	N/A	None	20.100 ns	fbus	w[6]
24	N/A	None	20.100 ns	fbus	w[4]
25	N/A	None	20.000 ns	a[4]	w[5]
26	N/A	None	19.400 ns	a[1]	w[2]
27	N/A	None	19.400 ns	a[1]	w[0]
28	N/A	None	19.200 ns	a[4]	w[3]
29	N/A	None	19.100 ns	a[5]	w[6]
30	N/A	None	19.100 ns	a[5]	w[4]
31	N/A	None	19.000 ns	a[6]	w[5]
32	N/A	None	18.500 ns	a[6]	w[7]
33	N/A	None	18.500 ns	a[3]	w[2]
34	N/A	None	17.900 ns	a[3]	w[4]
35	N/A	None	17.300 ns	a[5]	w[5]
36	N/A	None	17.300 ns	a[1]	w[1]
37	N/A	None	17.200 ns	a[0]	w[1]
38	N/A	None	17.100 ns	a[2]	w[1]
39	N/A	None	16.400 ns	a[0]	w[7]
40	N/A	None	16.400 ns	a[7]	w[0]
41	N/A	None	16.300 ns	a[2]	w[3]
42	N/A	None	16.100 ns	a[0]	cf
43	N/A	None	16.000 ns	a[7]	w[6]
44	N/A	None	16.000 ns	a[4]	w[4]
45	N/A	None	15.900 ns	fbus	cf
46	N/A	None	15.900 ns	fbus	cf
47	N/A	None	15.900 ns	fbus	cf
48	N/A	None	15.300 ns	a[3]	w[3]
49	N/A	None	15.000 ns	a[6]	w[6]
50	N/A	None	13.500 ns	a[7]	w[7]
51	N/A	None	13.500 ns	a[0]	w[0]
52	N/A	None	13.400 ns	a[2]	w[2]
53	N/A	None	12.700 ns	a[7]	cf

结果分析及结论：

结果分析：由图可得，Timing Analyzer Summary 总结所有经典定时分析的结果，并报告每个定时特性的最坏情况定时。比如从 flbus 到 w[5]的最坏定时情况的 tpd 为 21.700ns。下面的 tpd 报告表则给出了源节点和目标节点之间的 tpd 延迟时间，比如第三行中 flbus 到 w[1]的 tpd 为 21.500ns。

结论：实际连接图中个元器件连接之间是存在时间延迟的，而且不同的元器件之间的时间延迟也不相同。

3、控制信号产生逻辑

A) 创建工程（选择的芯片为 family=FLEX10K; name=EPF10K20T1144-4）

步骤：左上角 file->New Project Wizard->选择工程位置和工程名->选择芯片 FLEX10K，available device 中选择 EPF10K20T1144-4->点击 next->最后点击 finish 完成创建工程
工程创建图：



B) 编写源代码

根据实验指导和要求实现的功能写出对应的 Verilog 代码。

步骤：左上角 file->new->Verilog hdl file->编写代码（模块名需与工程名一致）->编译成功后保存到工程文件中。

代码截图：

```

1 module con_signal(
2     input mova,movb,movc,add,sub,and1,not1,rsr,rs1,jmp,jz,z,jc,c,in1,out1,nop,halt,sm,
3     input [7:0] ir,
4     output reg[1:0] reg_ra,reg_wa,madd,
5     output reg[3:0] alu_s,
6     output reg pc_ld,pc_inc,reg_we,ram_xl,ram_dl,alu_m,shi_fbus,shi_fibus,shi_frbus,ir_ld,cf_en,zf_en,sm_en,in_en,out_en);
7     always @(*)
8     begin
9         sm_en=~halt;
10        ir_ld=~sm;
11        ram_dl=(~sm)|movc|jmp|(jz&z)|(jc&c);
12        ram_xl=movb;
13        shi_fbus=mova|movb|add|sub|and1|not1|out1;
14        shi_fibus=rs1;
15        shi_frbus=rsr;
16        alu_s=ir[7:4];
17        alu_m=and1|not1|add|sub|rsr|rs1|out1;
18        cf_en=add|sub|rsr|rs1;
19        zf_en=add|sub;
20        pc_ld=jmp|(jc&c)|(jz&z);
21        pc_inc=(~sm)|(jc&(~c))|(jz&(~z));
22        reg_we=~(sm&(mova|movc|add|sub|and1|not1|rsr|rs1|in1));
23        in_en=in1;
24        out_en=out1;
25        reg_ra=ir[1:0];
26        reg_wa=ir[3:2];
27        if(sm==1&&movc==1) madd=01;
28        else if(sm==1&&movb==1) madd=2'b10;
29        else madd=00;
30    end
31 endmodule


```

C) 编译与调试（包含编译调试过程中的错误、警告信息以及资源消耗）

确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译，编译成功，保存文件。

无错误。

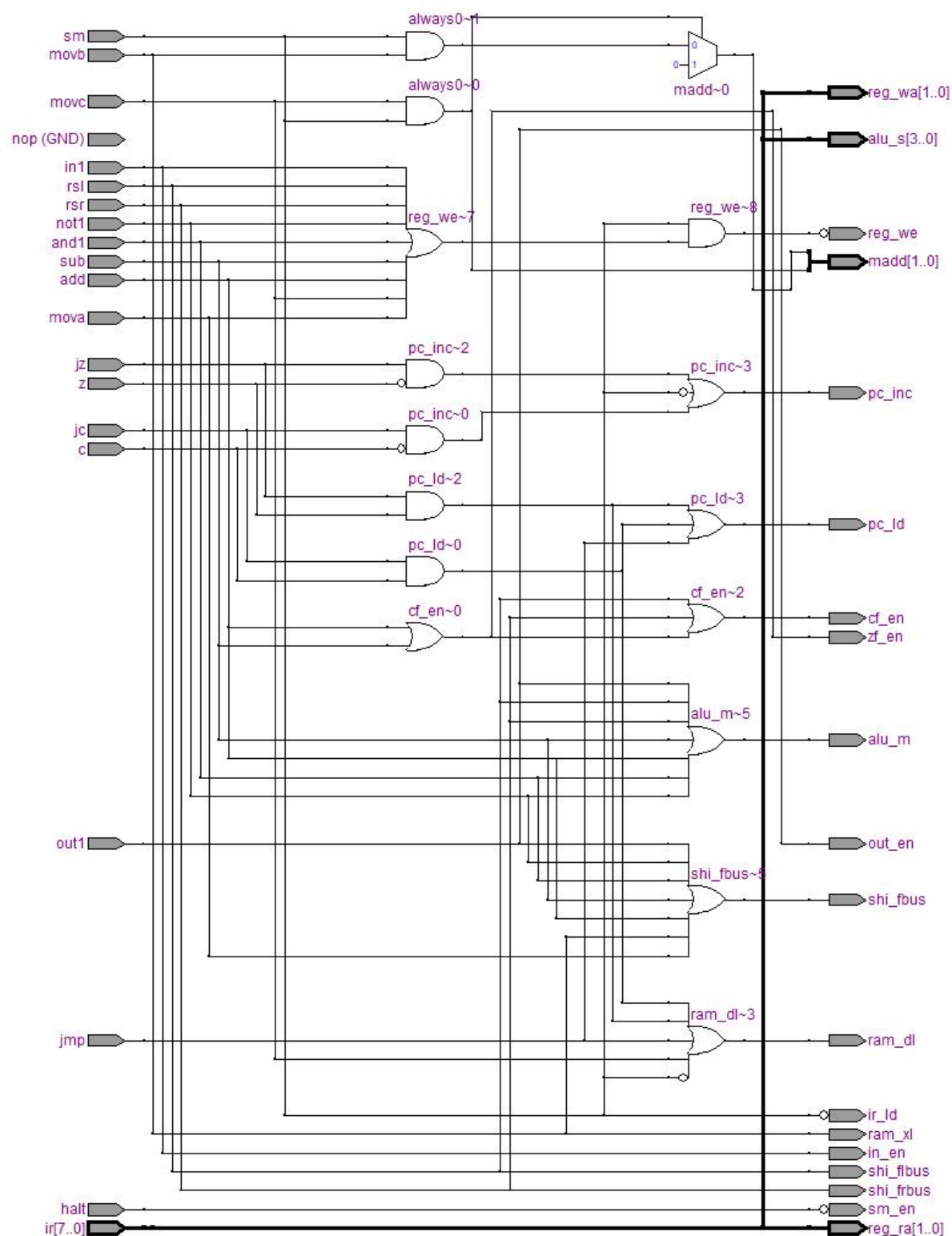
警告信息：

Type	Message
	Warning: Design contains 1 input pin(s) that do not drive logic

资源消耗：

Flow Status	Successful - Tue Nov 22 01:05:58 2022
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	con_signal
Top-level Entity Name	con_signal
Family	FLEX10K
Device	EPF10K20TI144-4
Timing Models	Final
Met timing requirements	Yes
Total logic elements	30 / 1,152 (3 %)
Total pins	52 / 102 (51 %)
Total memory bits	0 / 12,288 (0 %)

D) RTL 视图



视图分析及结论:

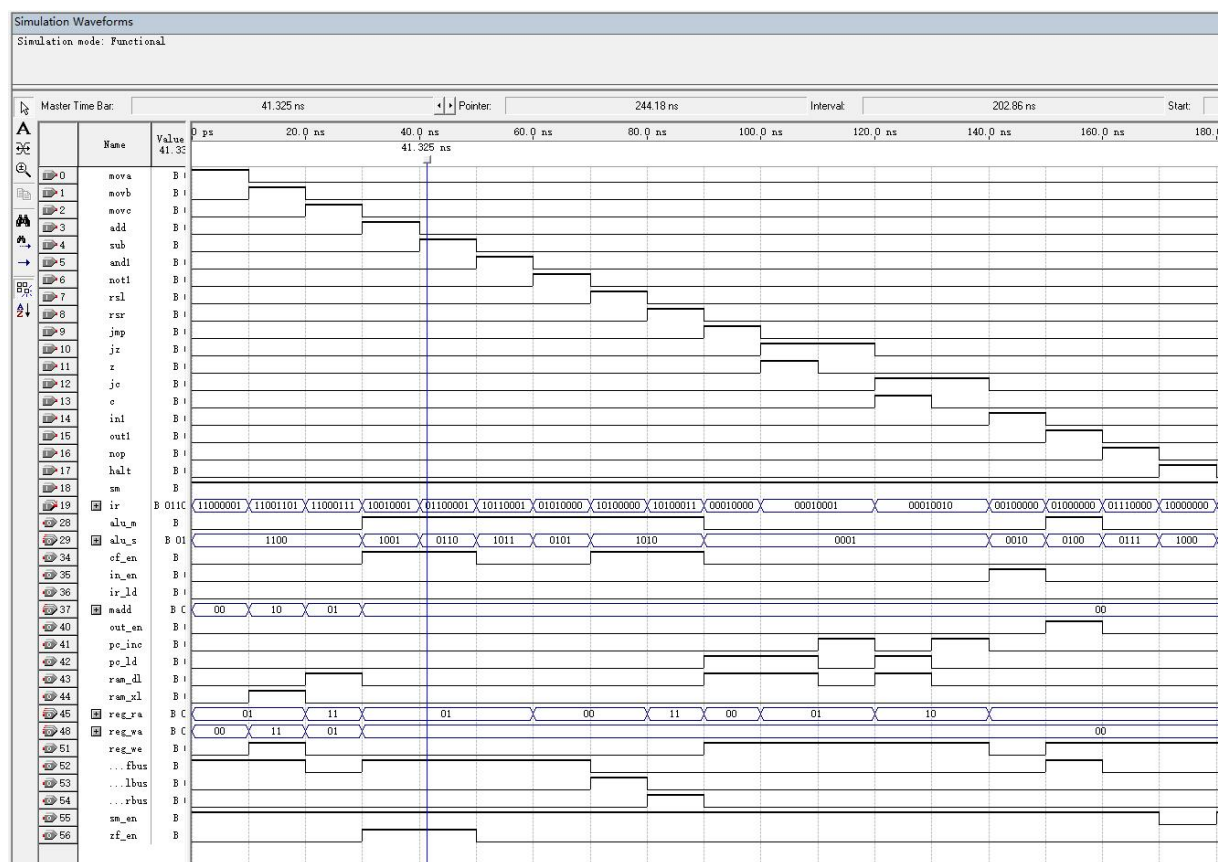
视图分析: 视图左边为输入, 右边为输出。其中连接有大量的元器件。例如比较器: 当输入相等时输出 1, 不相等时输出 0; 还由大量的 2-1 选择器构成, 当控制信号为 0 时, 输出第一位, 控制信号 1 时, 输出第二位。图中输入信号为 sm 等 20 个输入, 输出信号包括 reg_ra 等 19 种情况。各个输出端口之间通过导线相连。

结论：控制信号产生逻辑功能需要经过多重门的处理后才能实现，其内部原理结构图十分复杂。

E) 功能仿真波形

步骤：新建一个 vector waveform file。通过操作：右击 -【insert】-【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】，在【assignments】-【setting】-【simulator settings】-【simulation mode】中，将 timing 修改为 functional，然后再点击【processing】-【Generate Functional Simulation Netlist】，接着设置输入波形，成功后点击【start simulation】开始仿真，查看输出功能仿真波形图。

功能仿真波形图：



结果分析及结论：

结果分析：功能仿真是指不考虑器件延时和布线延时的理想情况下对源代码进行逻辑功能的验证。由仿真波形可得，对于输入状态的变化，输出结果实时变化，没有延迟，其结果与电路设计的真值表的结果相对应。

1. 当 mova 指令执行时，shi_fbus 和 sm_en 输出 1，其他输出为 0，madd 输出 00，alu_s 输出为 1100，reg_ra 输出 01，reg_wa 输出 00，正确
2. 当 movb 指令执行时，ram_xl 和 shi_fbus 和 reg_we 和 sm_en 输出为 1，其他输出为 0，madd 输出为 10，alu_s 输出为 1100，reg_ra 输出 01，reg_wa 输出 11，正确
3. 当 movc 指令执行时，ram_dl 和 sm_en 输出为 1，其他输出为 0，madd 输出 01，alu_s 输出 1100，reg_ra 输出 11，reg_wa 输出 01，正确
4. 当 add 指令执行时，shi_fbus，alu_en，cf_en，zf_en，sm_en 输出为 1，其他输出为 0，

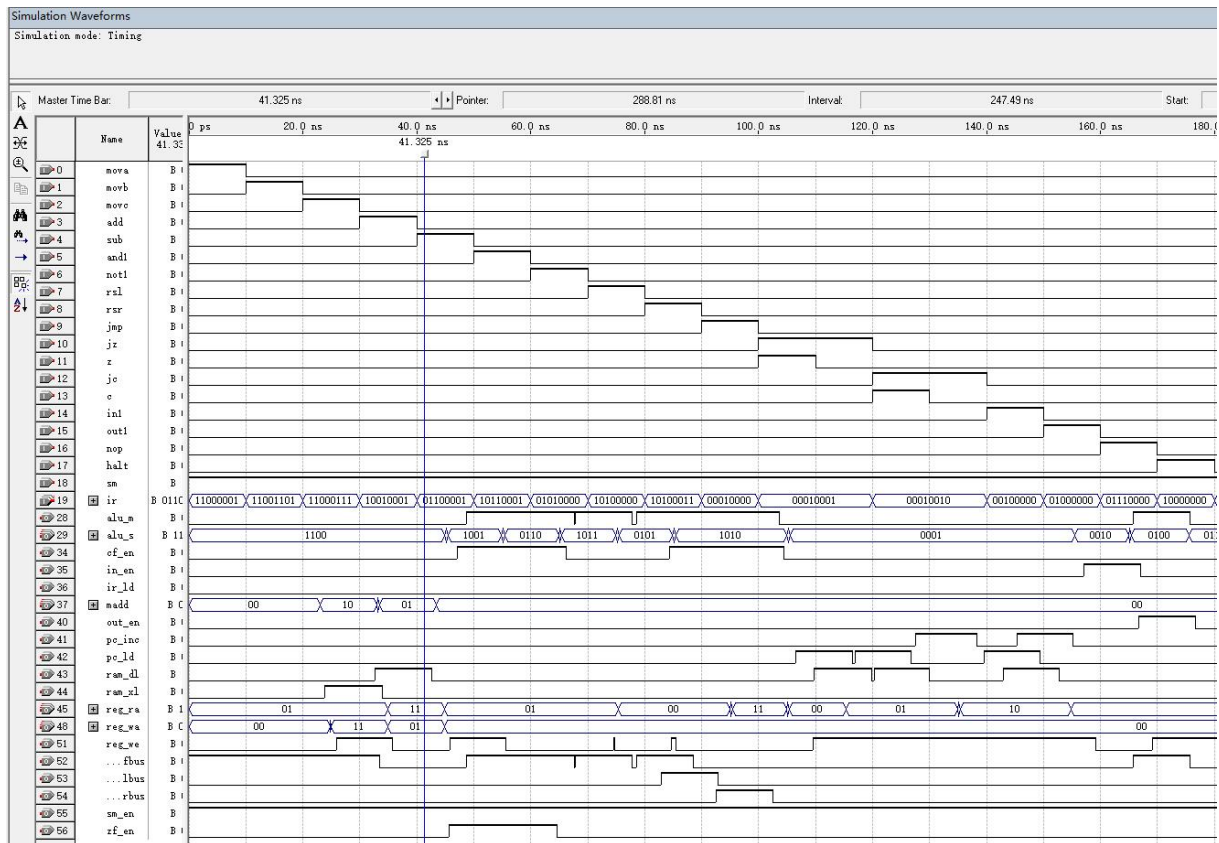
- alu_s 为 1001, reg_ra 输出 01, reg_wa 输出 00, 正确
5. 当 sub 指令执行时, shi_fbus 和 alu_m, cf_en, zf_en 和 sm_en 输出为 1, 其他输出为 0, alu_s 输出 0110, reg_ra 输出 01, reg_wa 输出 00, 正确
 6. 当 andl 指令执行时, shi_fbus 和 alu_m 和 sm_en 输出 1, 其他输出 0, alu_s 输出 1011, reg_ra 输出 01, reg_wa 输出 00, 正确
 7. notl 指令执行时, shi_fbus 和 alu_m 和 sm_en 输出 1, 其他输出 0, alu_s 输出 0101, reg_ra 输出 00, reg_wa 输出 00, 正确
 8. rsl 指令执行时, shi_fbus 和 alu_m 和 cf_en 和 sm_en 输出 0, 其他输出 0, alu_s 输出 1010, reg_ra 和 reg_wa 输出 00, 正确
 9. rsr 指令执行时, shi_fbus 和 alu_m 和 cf_en 和 sm_en 输出 1, 其他输出 0, alu_s 输出 1010, reg_ra 输出 11, reg_wa 输出 00, 正确
 10. jmp 指令执行时, ram_dl, pc_ld, reg_we 和 sm_en 输出 1, 其他输出 0, alu_s 输出 0001, reg_ra 和 reg_wa 输出 00, 正确
 11. jz 指令为 1 和 jc 指令为 1 时, 若 z 和 c 为 1 时, ram_dl 和 pc_ld 和 reg_we 和 sm_en 输出为 1, 其他输出为 0, 正确
若 z 和 c 为 0 时, pc_inc 和 reg_we 和 sm_en 输出 1, 其他输出 0, 正确
 12. inl 指令执行时, sm_en 和 in_en 输出 1, 其他输出 0, 正确
 13. outl 指令执行时, sm_en 和 out_en 输出 1, 其他输出 0, 正确
 14. nop 指令执行时, sm_en 输出 1, 其他输出 0, 正确
 15. halt 指令执行时, 输出全为 0, 正确

结论: 功能仿真是指在一个设计中, 在设计实现前对所创建的逻辑进行的验证其功能是否正确。布局布线以前的仿真都称作功能仿真, 它包括综合前仿真 (Pre-Synthesis Simulation) 和综合后仿真 (Post-Synthesis Simulation)。综合前仿真主要针对基于原理框图的设计; 综合后仿真既适合原理图设计, 也适合基于 HDL 语言的设计。功能仿真操作简单, 能体现和验证实验的功能, 但忽略延迟的影响会使结果与实际结果, 有一定误差。

F) 时序仿真波形

步骤: 新建一个 vector waveform file。通过操作: 右击 - **【insert】** - **【insert node or bus】** - **【node finder】** (pins=all; **【list】**) - **【>>】** - **【ok】** - **【ok】**。接着设置输入波形, 然后点击 **【start simulation】** 开始仿真, 查看时序仿真输出波形图。

时序仿真图:



结果分析及结论:

结果分析: 时序仿真是指在布线后进行, 是最接近真实器件运行的仿真, 它与特定的器件有关, 又包含了器件和布线的延时信息。由波形可得, 当输入状态发生改变时, 输出结果并未同时改变, 而是有一定延迟, 同时由于输入状态的改变, 导致电路出现“冒险”, 导致输出结果并未与预期结果相同。

结论: 时序仿真使用布局布线后器件给出的模块和连线的延时信息, 在最坏的情况下对电路的行为作出实际地估价。时序仿真使用的仿真器和功能仿真使用的仿真器是相同的, 所需的流程和激励也是相同的; 唯一的差别是为时序仿真加载到仿真器的设计包括基于实际布局布线设计的最坏情况的布局布线延时, 并且在仿真结果波形图中, 时序仿真后的信号加载了时延, 而功能仿真没有。时序仿真可以用来验证程序在目标器件中的时序关系。同时考虑了器件的延迟后, 其输出结果跟接近实际情况, 但是考虑的情况过多, 不容易操作, 容易产生错误。时序仿真不仅反应出输出和输入的逻辑关系, 同时还计算了时间的延时信息, 是与实际系统更接近的一种仿真结果。不过, 要注意的是, 这个时间延时是仿真软件“估算”出来的。

G) 时序分析

操作方法是: 编译后, 在 compilation report 中选择【timing analysis】-【summary】和【tpd】

Timing Analyzer Summary									
	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	25.600 ns	add	reg_we	--	--	0
2	Total number of failed paths								0

实验三 模型机组部件的实现（二）

tpd					
	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	25.600 ns	add	reg_we
2	N/A	None	25.500 ns	not1	reg_we
3	N/A	None	24.700 ns	and1	reg_we
4	N/A	None	24.600 ns	sub	reg_we
5	N/A	None	22.900 ns	jc	ram_dl
6	N/A	None	22.700 ns	c	ram_dl
7	N/A	None	20.300 ns	jz	ram_dl
8	N/A	None	20.000 ns	z	ram_dl
9	N/A	None	20.000 ns	rsl	reg_we
10	N/A	None	19.800 ns	jmp	ram_dl
11	N/A	None	19.600 ns	jc	pc_ld
12	N/A	None	19.500 ns	rsr	reg_we
13	N/A	None	19.400 ns	c	pc_ld
14	N/A	None	19.200 ns	in1	reg_we
15	N/A	None	18.700 ns	add	shi_fbus
16	N/A	None	18.700 ns	add	alu_m
17	N/A	None	18.600 ns	not1	shi_fbus
18	N/A	None	18.600 ns	not1	alu_m
19	N/A	None	18.300 ns	jz	pc_inc
20	N/A	None	17.800 ns	and1	shi_fbus
21	N/A	None	17.800 ns	and1	alu_m
22	N/A	None	17.700 ns	sub	shi_fbus
23	N/A	None	17.700 ns	sub	alu_m
24	N/A	None	17.500 ns	z	pc_inc
25	N/A	None	17.100 ns	in1	in_en
26	N/A	None	17.100 ns	add	cf_en
27	N/A	None	17.000 ns	jz	pc_ld
28	N/A	None	16.700 ns	out1	out_en
29	N/A	None	16.700 ns	z	pc_ld
30	N/A	None	16.500 ns	jmp	pc_ld
31	N/A	None	16.200 ns	sub	cf_en
32	N/A	None	16.000 ns	movb	reg_we
33	N/A	None	15.800 ns	movc	reg_we
34	N/A	None	15.700 ns	out1	shi_fbus
35	N/A	None	15.700 ns	out1	alu_m
36	N/A	None	15.600 ns	add	zf_en
37	N/A	None	15.500 ns	ir[5]	alu_s[1]
38	N/A	None	15.500 ns	ir[4]	alu_s[0]
39	N/A	None	15.400 ns	c	pc_inc
40	N/A	None	15.400 ns	ir[0]	reg_ra[0]
41	N/A	None	15.100 ns	jc	pc_inc
42	N/A	None	14.900 ns	halt	sm_en
43	N/A	None	14.900 ns	ir[7]	alu_s[3]
44	N/A	None	14.900 ns	ir[6]	alu_s[2]
45	N/A	None	14.900 ns	ir[3]	reg_wa[1]
46	N/A	None	14.900 ns	ir[1]	reg_ra[1]
47	N/A	None	14.800 ns	ir[2]	reg_wa[0]
48	N/A	None	14.600 ns	sub	zf_en
49	N/A	None	14.400 ns	rsl	cf_en
50	N/A	None	14.400 ns	rsr	cf_en
51	N/A	None	13.900 ns	movb	ram_xl
52	N/A	None	13.600 ns	rsl	alu_m
53	N/A	None	13.600 ns	rsr	alu_m
54	N/A	None	13.500 ns	movb	shi_fbus
55	N/A	None	13.500 ns	movb	shi_fbus
56	N/A	None	13.400 ns	sm	reg_we
57	N/A	None	13.400 ns	movc	madd[0]
58	N/A	None	13.000 ns	movc	madd[1]
59	N/A	None	13.000 ns	movb	madd[1]
60	N/A	None	12.900 ns	sm	ir_ld
61	N/A	None	12.900 ns	rsl	shi_fbus
62	N/A	None	12.900 ns	sm	pc_inc
63	N/A	None	12.900 ns	sm	madd[0]
64	N/A	None	12.700 ns	movc	ram_dl
65	N/A	None	12.700 ns	sm	ram_dl
66	N/A	None	12.700 ns	sm	madd[1]
67	N/A	None	12.500 ns	rsr	shi_fbus

结果分析及结论：

结果分析：由图可得，Timing Analyzer Summary 总结所有经典定时分析的结果，并报告每个定时特性的最坏情况定时。比如从 add 到 reg_we 的最坏定时情况的 tpd 为 25.600ns。下面的 tpd 报告表则给出了源节点和目标节点之间的 tpd 延迟时间，比如第二行中 not 到 reg_we 的 tpd 为 25.500ns。

结论：实际连接图中个元器件连接之间是存在时间延迟的，而且不同的元器件之间的时间延迟也不相同。

四、思考题

1. 移位逻辑不工作时，输出应该为何值？为什么？

答：应该输出高阻。移位逻辑不工作时，为了预防移位逻辑输出数据到总线中和其他位置输出到总线中的数据起冲突，应该阻止移位逻辑向总线输出数据，故此时移位逻辑输出应该为高阻。

2. 移位逻辑的输出 Cf 应该如何处理？

答：输出 Cf 只受算术运算和移位指令影响，其他时刻应当保持原状态，故这两个信号应该借助触发器/锁存器储存

3. 如何产生正确的控制信号以及具体的编程实现？

答：应当逐个分析每个控制信号在不同的指令下对应的状态，利用逻辑函数进行状态的总和。

五、实验总结、必得体会及建议

1、从需要掌握的理论、遇到的困难、解决的办法以及经验教训等方面进行总结。

答：这次实验我基本掌握了 8 重 3-1 多路复用器，移位逻辑，控制信号产生逻辑的工作原理及如何构造，让我对模型机的工作原理有了更加深入的了解，同时在运行调试过程中的锁存器问题也让我对组合电路以及时序电路的区别与表现理解更为细致。本次实验的核心在于理解 CPU 中命令的执行情况和各种控制信号对应的输出变化。在编程实现时虽然实现语句很简单，但需要非常细致，对每一步命令执行的输入输出非常了解，不然很容易在逻辑上出现错误。在编写模型机各个部件时也应当具有全局观念，要考虑把这个部件最终组合后可能的运行情况，使得设计的差错尽可能在最初的时候就能避免。Verilog 程序调试时会经常遇到各种各样的问题，但通过上网查询相关资料和询问同学后得以解决问题。以后再遇到相关问题时，可以通过先上网查询资料，再询问老师和同学来解决。

2、对本实验内容、过程和方法的改进建议（可选项）。

答：最好能给出控制信号产生逻辑的各个控制信号的输出功能表