哈理工集成 19 FPGA 期末复习资料

1. IP 核在 EDA 技术和开发中具有十分重要的地位，IP 是指（ A ）。
   A：知识产权　B：互联网协议　C：网络地址　D：都不是
2. 在 verilog HDL 的 always 块本身是（ B ）语句
   A? 程序语句执行顺序均为顺序执行，always 块之间是并行
   A：顺序　B：并行　C：顺序或并行　D：串行
3. 大规模可编程器件主要有 FPGA、CPLD 两类，下列对 FPGA 结构与工作原理的描述中，正确的是（ C ）。
   A：FPGA 是基于乘积项结构的可编程逻辑器件；
   B：FPGA 是全称为复杂可编程逻辑器件；
   C：基于 SRAM 的 FPGA 器件，在每次上电后必须进行一次配置；
   D：在 Altera 公司生产的器件中，MAX7000 系列属 FPGA 结构。
4. 下列 EDA 软件中，哪一个不具有逻辑综合功能：（B ）。
   A：ISE　B：ModelSim　C：Quartus II　D：Synplify
5. 关于 Verilog HDL 中的数字，请找出以下数字中最大的一个：（A ）。
   A：8′b1111_1110　B：3′o276　C：3′d170　D：2′h3E

大规模可编程器件主要有 FPGA、CPLD 两类，下列对 CPLD 结构与工作原理的描述中，正确的是（ C ）。

6. A：CPLD 是基于查找表结构的可编程逻辑器件；
   B：CPLD 即是现场可编程逻辑器件的英文简称；
   C：早期的 CPLD 是从 GAL 的结构扩展而来；
   D：在 Xilinx 公司生产的器件中，XC9500 系列属 CPLD 结构；
7. IP 核在 EDA 技术和开发中具有十分重要的地位；提供用 VHDL 等硬件描述语言描述的功能块，但不涉及实现该功能块的具体电路的 IP 核为（ D ）。
   A：瘦 IP　B：固 IP　C：胖 IP　D：都不是
8. 不完整的 IF 语句，其综合结果可实现（ A ）。
   A：时序逻辑电路　B：组合逻辑电路　C：双向电路　D：三态控制电路
9. CPLD 的可编程是主要基于什么结构（ D ）。
   A：查找表（LUT）　C：PAL 可编程　B：ROM 可编程　D：与或阵列可编程
10. IP 核在 EDA 技术和开发中具有十分重要的地位，以 HDL 方式提供的 IP 被称为：（ C ）
    A：硬 IP　B：固 IP　C：软 IP　D：都不是；
11. FPGA 可编程逻辑基于的可编程结构基于（ A ）。
    A：LUT 结构　B：乘积项结构　C：PLD　D：都不对
12. CPLD 可编程逻辑基于的可编程结构基于 （ B ）。
    A：LUT 结构　B：乘积项结构　C：PLD　D：都不对
13. 将设计的系统按照 EDA 开发软件要求的某种形式表示出来，并送入计算机的过程，称 为（ A ）。
    A：设计的输入　B：设计的输出　C：仿真　D：综合
14. 一般把 EDA 技术的发展分为（B ）个阶段。
    A：2　B：3　C：4　D：5
15. 设计输入完成之后，应立即对文件进行（ A ）。
    A：编译　B：编辑　C：功能仿真　D：时序仿真
16. 基于硬件描述语言的数字系统设计目前最常用的设计方法称为（ B ）设计法。
    A：自底向上　B：自顶向下　C：积木式　D：顶层
17. 在 EDA 工具中，能将硬件描述语言转化为硬件电路的重要工具软件为（ B ）。
    A：仿真器　B：综合器　C：适配器　D：下载器
18. 在 EDA 工具中，能完成在目标系统器件上布局布线的软件称为（ C ）。
    A：仿真器　B：综合器　C：适配器　D：下载器
19. 逻辑器件（ A ）属于非用户定制电路。
    A：逻辑门　B：PROM　C：PLA　D：GAL
20. 可编程逻辑器件 PLD 属于（ A ）电路。
    A：半用户定制　B：全用户定制　C：自动生成　D：非用户定制

21. 不属于 PLD 基本结构部分的是（ C ）。
A：与门阵列　B：输入缓存　C：与非门阵列　D：或门阵列
22. 嵌套的 if 语句，其综合结果可实现（ A ）。
A：条件相与的逻辑　B：条件相或的逻辑 C：条件相异或的逻辑　D：三态控制电路
23. 嵌套的使用 if 语句，其综合结果可实现（ A ）。
A：带优先级且条件相与的逻辑电路　B：双向控制电路
C：三态控制电路　D：条件相异或的逻辑电路
24. 下列哪个 FPGA/CPLD 设计流程是正确的（ A ）。
A：原理图/HDL 文本输入->功能仿真->综合->适配->编程下载->硬件测试
B：原理图/HDL 文本输入->适配->综合->功能仿真->编程下载->硬件测试
C：原理图/HDL 文本输入->功能仿真->综合->编程下载->适配->硬件测试
D：原理图/HDL 文本输入->适配->功能仿真->综合->编程下载->硬件测试
25. FPGA 的片上 RAM 资源，不可以在那个设计中应用？（ A ）
A: Shift Register　B: ROM
C: RAM　D: FIFO
26. 下列哪些属于时钟约束？（ D ）

C? A 是失败路径设置 B 是假的 C 我也觉得是假的 D 是多周期约束
A: set_false_path　B: set_input_path
C: set_max_delay　D: set_multicycle　path
27. FPGA 可以有哪些工艺？（ D ）
A: SDRAM　B: SRAM　C: EEPOM　D: DDR
28. 下列哪个不是 FPGA 片内资源？（ D ）
A: RAM　B: LUT　C: DSP　D: SDRAM
29. 下列哪个选项是 FPGA 设计中必须的设计约束？（ C ）
A: 管脚约束　B: 跨时钟域约束
C: 时钟周期约束　D: 片上 RAM 位置约束

30. 速度优化算法
流水线模型
模块复制
寄存器配平
关键路径法
乒乓操作
串并转换

31. 面积优化算法
串行化
资源共享
逻辑优化

32. Latch 和 Register 区别？行为描述中 Latch 如何产生？

区别：锁存器是电平触发。 寄存器是边沿触发。
寄存器在同一时钟边沿触发下动作，符合同步电路设计思想，而锁存器则属于异步电路设计，往往会导致时序分析困难，不适当的应用锁存器则会大量浪费芯片资源。时序设计中尽量使用寄存器触发。
产生原因：if 或者 case 语句的逻辑表达不完全。在行为描述中，如果对应所有可能的输入条件，有的输入没有对应的明确的输出，系统会综合出锁存器。比如缺少 else 语句

33. FPGA 中 LE 的基本结构包含那几部分？

一个四口输入的查找表（LUT)，以实现四种变量的任何功能
一个可编程的寄存器
一个进位链连接
一个寄存器链连接

1、查询表： 用于完成用户需要的逻辑功能，CYCLONE II 系列的查询表是 4 输入 1 输出的，可以完成任意 4 输入 1 输出的组合逻辑。
2、可编程寄存器：可以配置成 D 触发器，T 触发器，JK 触发器，SR 触发器。每个寄存器包含 4 个输入信号，数据输入、时钟输入、时钟使能、复位输入。

34． 设计分频电路，会奇数分频和偶数分频，各种占空比分频方法。

奇分频

```
module odd_div(clk,rst_n,n,clk_odd);
 input clk,rst_n;
 input [7:0] n;
 output clk_odd;

 reg [7:0] cntp;
always @(posedge clk or negedge rst_n)
begin
 if(!rst_n)
        cntp<=8'd0;
 else if(n[0]==1'b0)
        cntp<=8'd0;
 else if(cntp==n-1'b1)
        cntp<=8'd0;
 else
        cntp<=cntp+1'b1;
end

 reg clk_p;
always @(posedge clk or negedge rst_n)
begin
        if(!rst_n)
                clk_p<=1'b1;
        else if(n[0]==1'b0)
                clk_p<=1'b1;
        else if(n==8'd1)
                clk_p<=1'b1;
        else if(cntp<n/2)
                clk_p<=1'b1;
        else
                clk_p<=1'b0;
end

 reg [7:0] cntn;
always @(negedge clk or negedge rst_n)
begin
 if(!rst_n)
        cntn<=8'd0;
 else if(n[0]==1'b0)
        cntn<=8'd0;
 else if(cntn==n-1'b1)
        cntn<=8'd0;
 else
        cntn<=cntn+1'b1;
end
```

```
 reg clk_n;
always @(negedge clk or negedge rst_n)
begin
 if(!rst_n)
        clk_n<=1'b1;
 else if(n[0]==1'b0)
        clk_n<=1'b1;
 else if(n==8'd1)
        clk_n<=1'b1;
 else if(cntn<n/2)
        clk_n<=1'b1;
 else
        clk_n<=1'b0;
end

assign clk_odd = clk_p | clk_n;

endmodule
```

偶分频：

```
module even_div(clk,rst_n,n,clk_even);
  input clk,rst_n;
  input [7:0] n;
  output reg clk_even;

  reg [7:0] cnt;
always @(posedge clk or negedge rst_n)
begin
  if(!rst_n)
        cnt<=8'd0;
  else if(n==8'd0)
        cnt<=8'd0;
  else if(cnt==n/2-1'b1)
        cnt<=8'd0;
  else
        cnt<=cnt+1'b1;
end

always @(posedge clk or negedge rst_n)
begin
  if(!rst_n)
        clk_even <= 1'b0;
  else if(n==8'd0)
        clk_even <= 1'b0;
  else if(n==8'd2)
        clk_even<=~clk_even;
  else if(n[0]==1'b0)
  begin
            if(cnt==n/2-1'b1)
                  clk_even <= ~clk_even;
            else
                  clk_even <= clk_even;
  end
end

endmodule
```

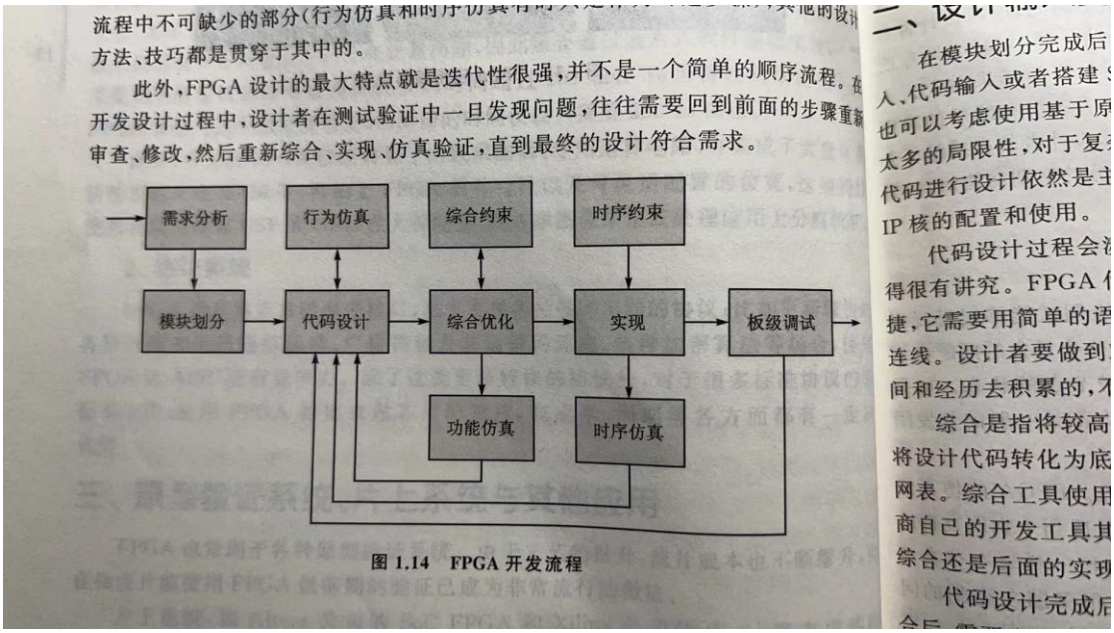
35. 简述 FPGA 等可编程逻辑器件设计流程

系统设计电路构思
设计说明与设计划分
电路设计与输入（HDL 代码，原理图）
功能仿真与测试
逻辑综合
门级综合
逻辑验证与测试（综合后仿真）
布局布线
时序仿真
版图验证与仿真
加载配置
在线调试

或者

通常可将 FPGA/CPLD 设计流程归纳为以下 7 个步骤：
1. 设计输入：Verilog 或 VHDL 编写代码；
2. 功能仿真：设计的电路必须在布局布线前验证电路功能是否有效；
3. 设计编译：设计输入之后从高层次系统行为设计向门级逻辑电路设转化翻译过程；
4. 优化：对于上述综合生成的网表，根据布尔方程功能等效的原则，用更小更快的综合结果代替一些复杂的单元，并与指定的库映射生成新的网表；
5. 布局布线；
6. 时序仿真：利用在布局布线中获得的精确参数再次验证电路的时序；
7. 生产：布线和后仿真完成之后，开始 ASCI 或 PLD 芯片的投产。

或者



36. 比较下面两个代码，利用与门、D 触发器等器件，分别画出如下两段代码的框图。指出在

FPGA 中采用哪种处理方式比较合理，为什么？

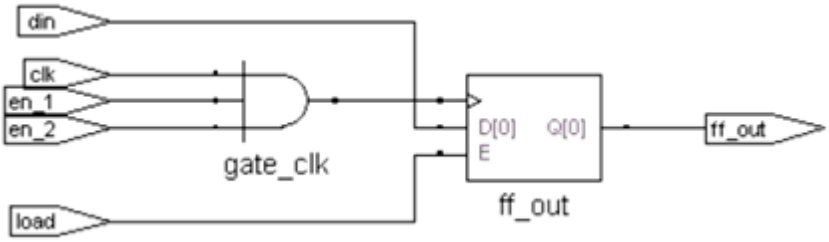
（1）

```
//en_1,en_2:使能信号
assign  gate_clk = (en_1 & en_2 & clk);
always @(posedge gate_clk)
begin
    if(load)
       ff_out <= din; //ff_out:flip-flop output
end
```

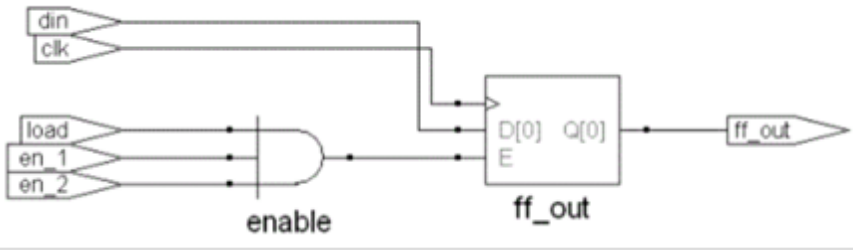
（2）

```
//enable:使能
assign enable = (en_1 & en_2 & load);
always @(posedge clk)
begin
    if (enable)
       ff_out <= din;
end
```

代码一框图



代码二框图

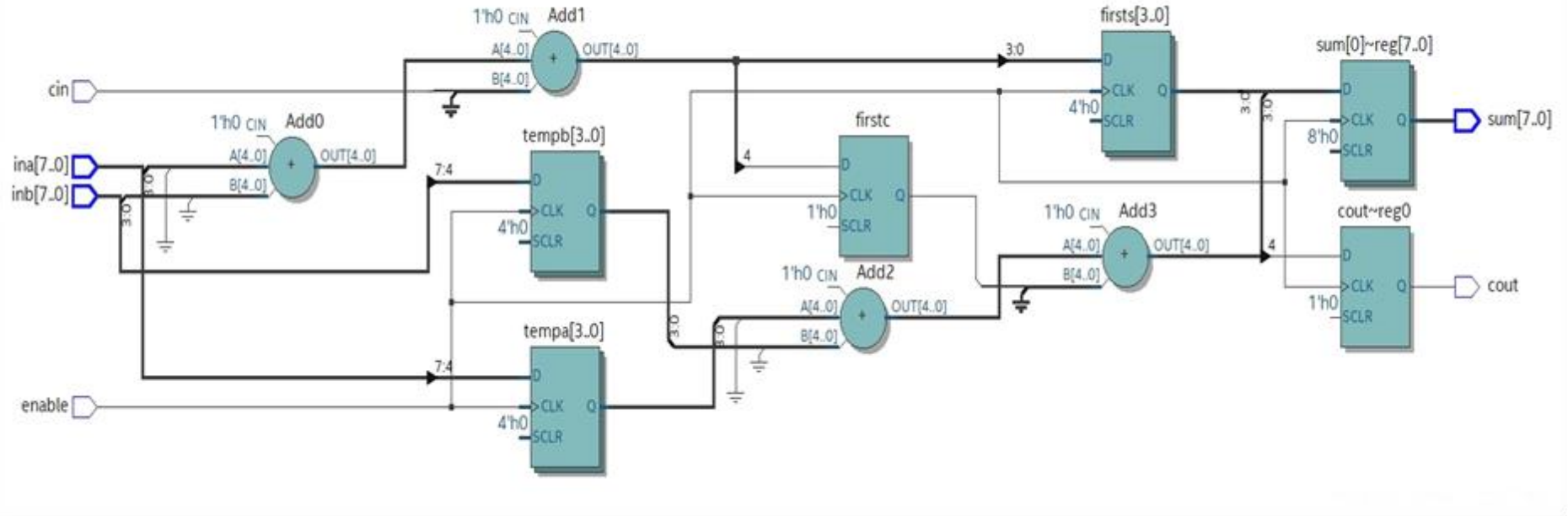


第二种方案比较合理。第一种方案增加了时钟的时延，容易不稳定，第二种设计方法时钟延迟稳定不容易产生毛刺。

37. 设计一个带有二级流水的八位加法器。写出代码，并且画出最终实现的 RTL 级图。

```
module adder_pipeline(cout, sum, ina, inb, cin, enable);
    output cout;
    output [7:0] sum;
    input [7:0] ina, inb;
    input cin, enable;
    reg cout;
    reg [7:0] sum;
    reg [3:0] tempa, tempb, firsts;
    reg firstc;
    always @(posedge enable)
    begin
        {firstc,firsts} = ina[3:0] + inb[3:0] + cin;
         tempa = ina[7:4]; //高 4 位输入寄存，使其与 sum 低 4 位在下级流水线同步输入。
```

```
            tempb = inb[7:4]; //否则 sum 的高 4 位，与低四位分两个时钟周期输出
        end
always @(posedge enable)
        begin
            {cout,sum[7:4]} = tempa + tempb + firstc;
            sum[3:0] = firsts; //不能合并为{cout, sum} = {tempa + tempb + firstc, firsts}; 位宽不匹配
        end
endmodule
```


cin
ina[7..0]
inb[7..0]
enable
1'h0 CIN
Add0
A[4..0]
B[4..0]
OUT[4..0]
Add1
Add2
Add3
tempa[3..0]
tempb[3..0]
firstc
firsts[3..0]
sum[0]~reg[7..0]
cout~reg0
sum[7..0]
cout
D
CLK
Q
SCLR
4'h0
8'h0
1'h0
3:0
7:4
4

38. 比较下面两个代码，利用 FF、Adder、Mux、Demux 等元件画出对应的框图，并比较这两个设计的不同以及优劣？（注意：不要对加法器等进行细化处理，如加法可以利用进位链实现，只要利用现有模块，如 DFF、Adder、减法器等）

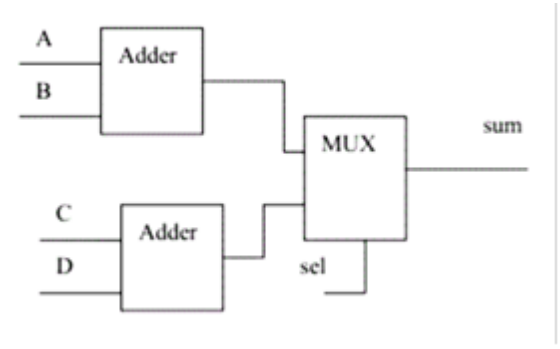
(1) always @ (A or B or C or D)

```
sum = sel ? (A+B): (C+D)
```

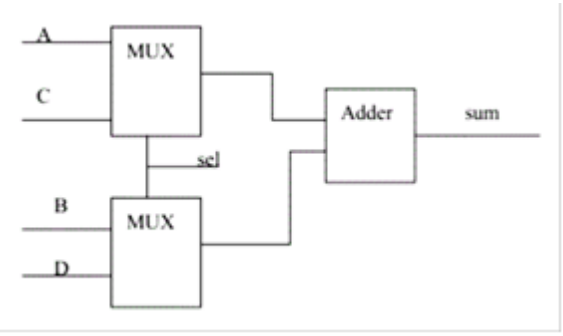
(2) always @ (A or B or C or D)

```
begin
   switch0 = sel ? A:C;
   switch1 = sel ? B:D;
end
assign sum = switch0 + switch1;
```

代码一对应框图



代码二对应框图



第二种设计比第一种设计少一个加法器，设计资源消耗减少，提高了资源的利用率。

39. 给了外部 io 端口参数，写出 io 端口约束值的设定方法

参数：

Clocks Summary

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by	Phase	Offset	Edge List	Edge Shift	Inverted	Master	Source	Targets
1	C100	Generated	10.000	100.0 MHz	0.000	5.000		1	1					false	clk_100MHz	clk_in_100mhz	{ inst1|altpll_component|pll|clk[0] }
2	C100_out	Generated	10.000	100.0 MHz	0.000	5.000		1	1					false	clk_100MHz	clk_in_100mhz	{ inst1|altpll_component|pll|clk[2] }
3	C200	Generated	5.000	200.0 MHz	0.000	2.500		1	2					false	clk_100MHz	clk_in_100mhz	{ inst1|altpll_component|pll|clk[1] }
4	clk_100MHz	Base	10.000	100.0 MHz	0.000	5.000											{ clk_in_100mhz }

	Minimum	*Maximum*
Clock-to-output Delay (ns)	1	3
Estimated PCB Data Trace Delay (between devices) (ns)	0.5	1
Board Clock Skew (ns)	-0.5	0.5

2. Use the table and diagram above to fully constrain all input data ports with respect to **clk_in_100mhz**. To refresh your memory, the equations are below:

set_input_delay (max) = board delay (max) – clock skew (min) + ext. t_{co} (max)

set_input_delay (min) = board delay (min) – clock skew (max) + ext. t_{co} (min)

设定方法：

```
create_clock -name clk_100MHz -period 10.000 [get_ports {clk_in_100mhz}]
create_generated_clock -name C100 -source [get_ports {clk_in_100mhz}] -divide_by 1 -multiply_by 1 [get_pins {inst1|altpll_component|pll|clk[0]}]
create_generated_clock -name C200 -source [get_ports {clk_in_100mhz}] -divide_by 1 -multiply_by 2 [get_pins {inst1|altpll_component|pll|clk[1]}]
create_generated_clock -name C100_out -source [get_ports {clk_in_100mhz}] -divide_by 1 -multiply_by 1 [get_pins {inst1|altpll_component|pll|clk[2]}]
 set_input_delay -clock { clk_100MHz } -min 1 [get_ports {din_*}]
 set_input_delay -clock { clk_100MHz } -max 4.5 [get_ports {din_*}]
set_output_delay -clock { clk_out } -max 0.5 [get_ports {multout_ab*}]
set_output_delay -clock { clk_out } -min -0.5 [get_ports {multout_ab*}]
```

或者

类似逻辑综合输入，输出端口的设定
输入延迟
输出延迟

40. 时钟周期 T，触发器 D1 的寄存器到输出时间最大为 T1max，最小为 T1min。组合逻辑电路最大延迟为 T2max，最小为 T2min。问，触发器 D2 的建立时间 T3 和保持时间应满足什么条件？

建立时间：T3setup > T+T2max
保持时间：T3hold > T1min+T2min

（没认真验证过是否正确

41. 什么是同步设计规则？

同步时序设计三条原则
基本原则：使用时钟沿触发所有的操作。
满足 Setup 时间原则：在有效时钟沿到达前，数据输入至少已经稳定了采样寄存器的 Setup 时间那么久
满足 Hold 时间原则：在有效时钟沿到达前，数据输入至少还将稳定保持采样寄存器的 Hold 时间那么久

单单单单单单

1单所有的数据都要通过组合逻辑和延时单元（典型的延时单元为触发器，这些触发器被一个时钟信号所同步）

2单延时总是由延时单元来控制，而不是由组合逻辑来控制

3、组合逻辑所产生的信号不能在没有通过一个同步延时单元的情况下反馈给同一个组合逻辑

4、时钟信号不能被门控，必须直接到达延时单元的时钟输入端，而不是经过任何组合逻辑

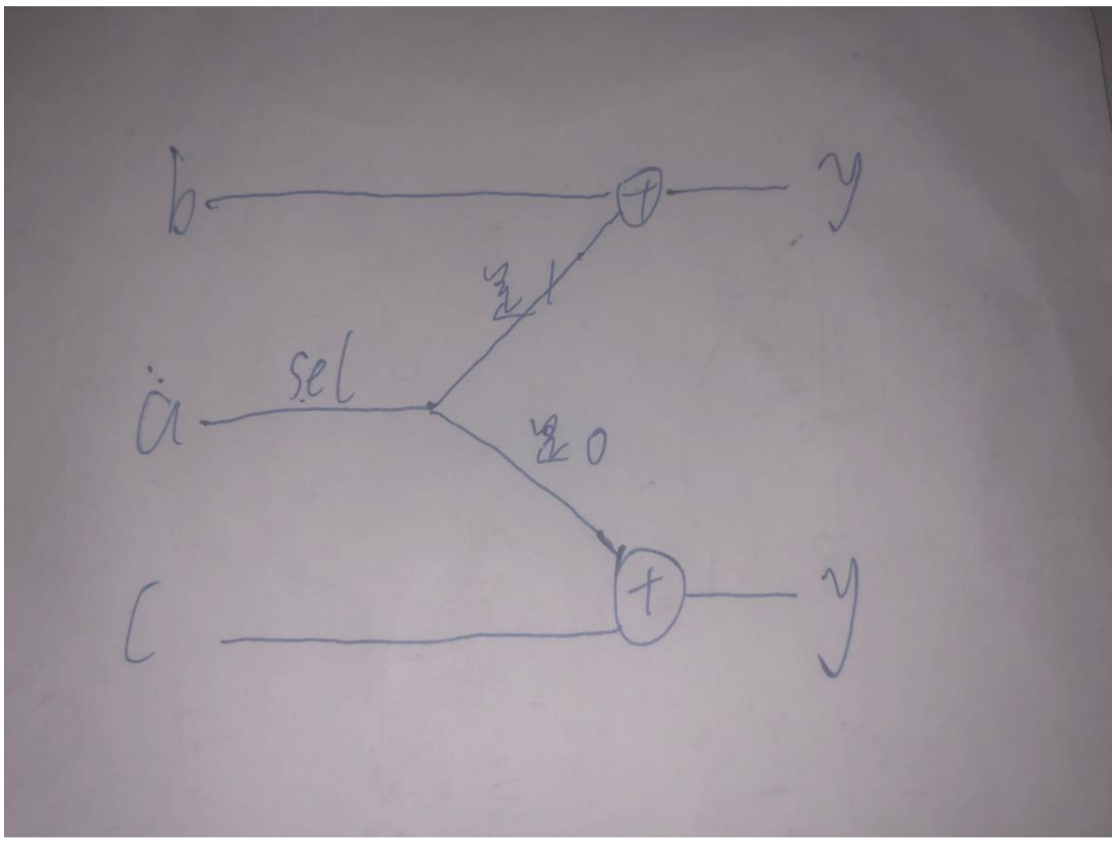
5、数据信号必须只通过组合逻辑或延时单元的数据输入端

单单单单单单

把通过两个不同时钟作用区域之间的信号作为异步信号处理

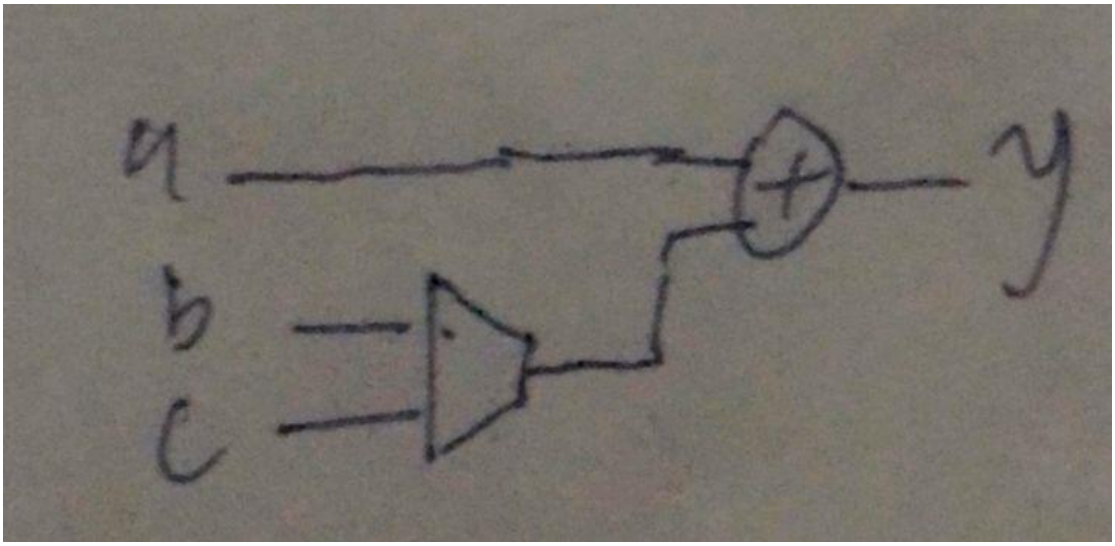
42. 分析下列代码，画出该代码的逻辑图，改写该程序，使其资源占用率尽可能的少，并画出改写后代码的逻辑图。 单单单单38单

```
always@ (sel or a or b or c)
    if(sel)
        y = a + b;
    else
        y = a + c;
```

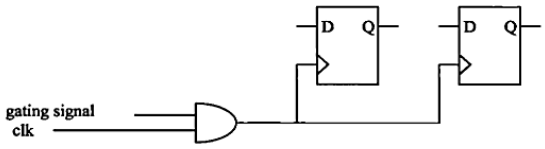


改写后的代码

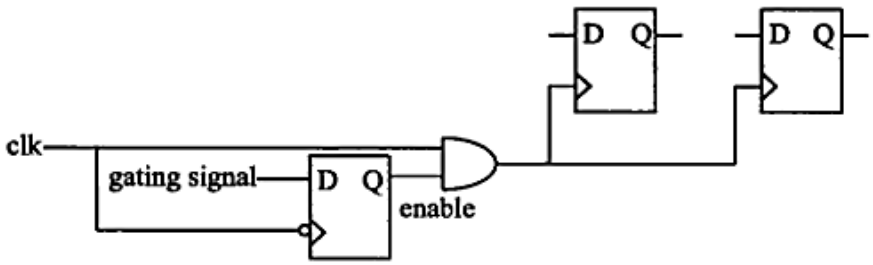
```
wire temp1=sel?b:c;
assign sum =a+temp1;
```



43. 下图是通过门控时钟进行的低功耗设计，请说出该电路的缺点，以及列举出一种解决方法，即可以达到低功耗的目的，又可以避免该电路的缺点。



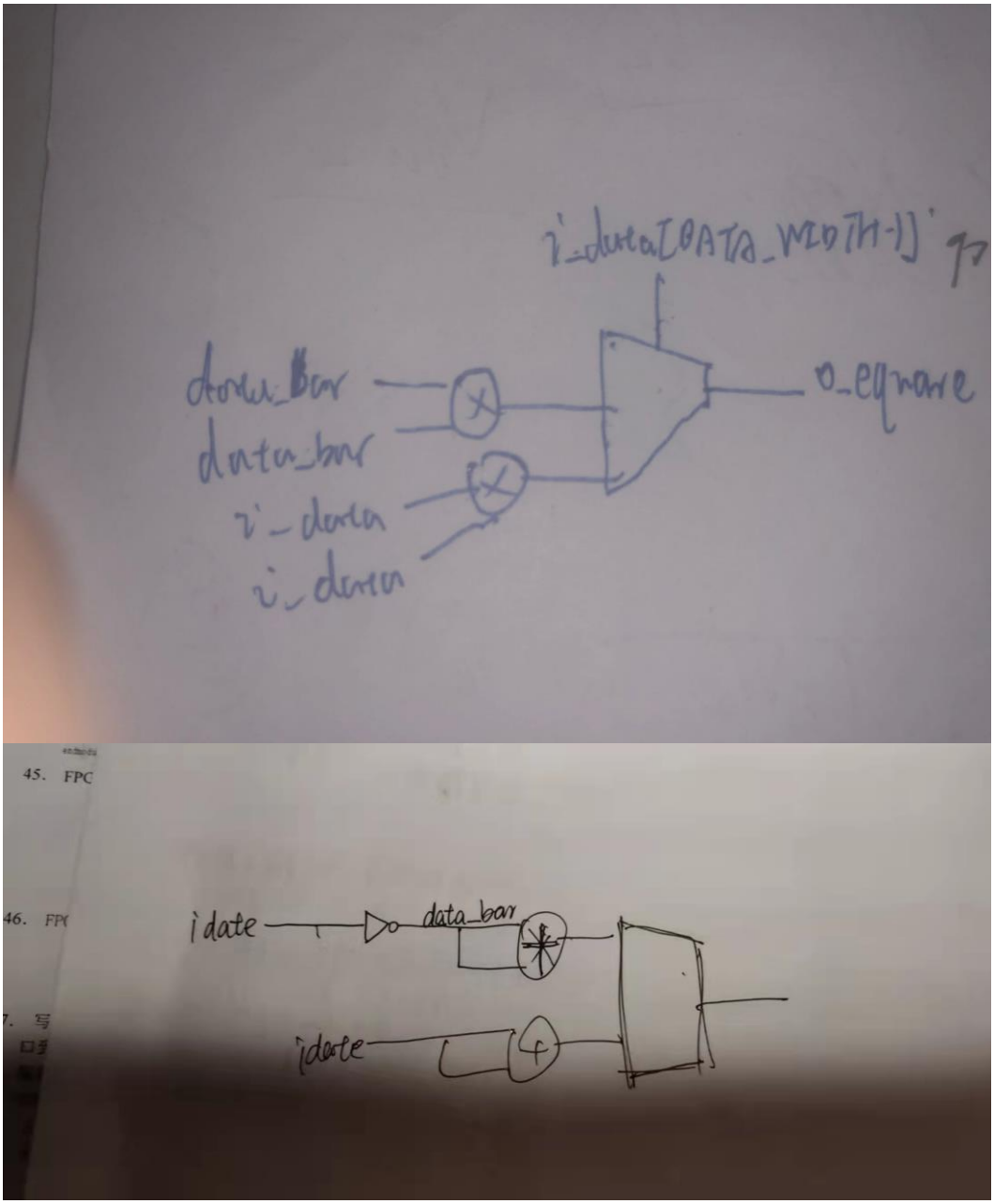
缺点：门控时钟信号出现毛刺
解决方法：添加使能信号



44. 查看下面代码，画出对应框图，从资源共享角度，重新编写代码，采用一个乘法器实现下面代码相同的功能，请写出相关代码。

```
module resoure_share (i_data,o_square)

    parameter  DATA_WIDTH = 8;
    parameter  PRO_WIDTH  = 16;

    input  [DATA_WIDTH-1:0] i_data;
    output [PRO_WIDTH-1:0]  o_square;

    wire   [DATA_WIDTH-1:0] data_bar;

    assign data_bar = ~i_data + 1;

    assign o_square = (i_data[DATA_WIDTH-1])?(data_bar*data_bar):(i_data*i_data);

endmodule
```

框图：



代码：

```
module resoure_share(i_data,o_square)

    parameter DATA_WIDTH=8;
    parameter PRO_WIDTH=16;

    input [DATA_WIDTH-1:0] i_data;
    output [PRO_WIDTH-1:0] o_square;

    wire [DATA_WIDTH-1:0] data_bar;

    assign data_bar=~i_data+1;
    wire temp1=(i_data[DATA_WIDTH-1])?data_bar:i_data;
    assign o_square = temp1*temp1;
```

```
endmodule
```

45. FPGA 芯片内有哪两种存储器资源？

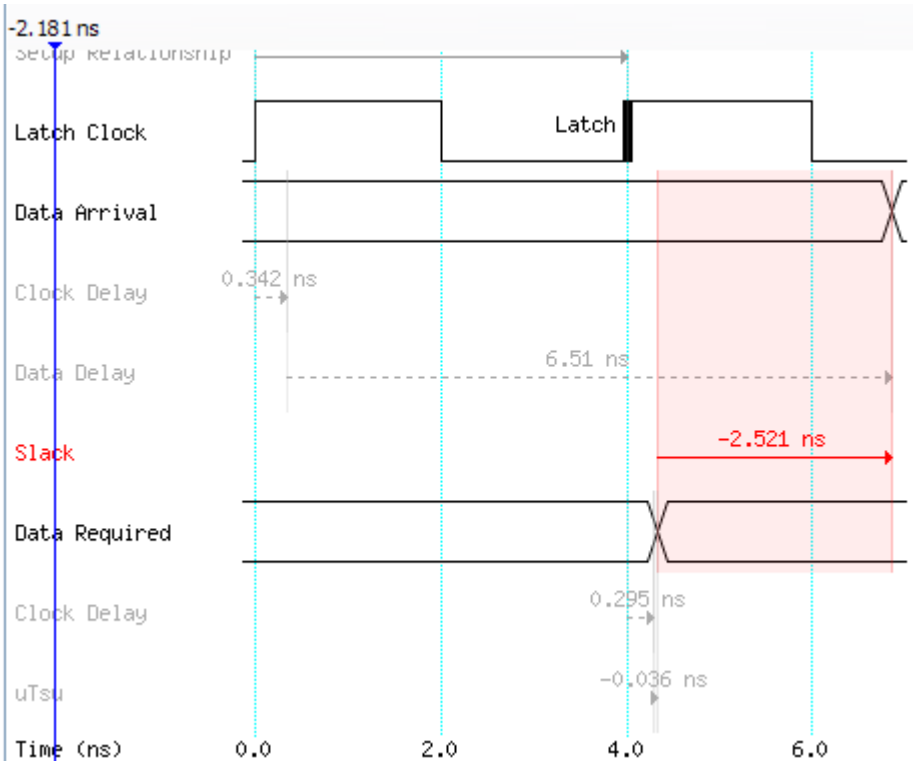
BLOCK RAM、由 LUT 配置成的内部存储器

FPGA 芯片内有两种存储器资源：一种叫 block ram（储存器块），另一种是由 LUT 配置成的内部存储器（也就是分布式存储器）。block ram 由一定数量固定大小的存储块构成的，使用 block ram，不占用额外的逻辑资源，并且速度快。但是使用的时钟消耗 block ram 的资源是其大小的整数倍。

46. FPGA 选型时要考虑哪些方面？

成本
配置方式、开发工具
规模大小 或者 片内资源 或者 容量
速度需求 或者 时钟速度
引脚
IP 的可用性
器件的可用性
功耗

47. 写出建立时间和保持时间检查公式，包括寄存器到寄存器、输入端口到寄存器、输出端口到寄存器。并画出时序图图解时序余量，能够看懂余量波形图、时序分析报告，时钟偏斜多少，发送沿锁存沿关系是多少？数据到达时间，数据需求时间，能够找到，能够判断出是建立时间余量还是保持时间余量。不能收敛是后怎么办？



Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	⊟ 2.929	2.929					clock path
3	2.929	2.929	R				clock network delay
4	⊟ 9.673	6.744					data path
5	3.163	0.234		uTco	36	M4K_X17_Y11	...pk1:auto_generated|ram_block1a0~portb_address_reg0
6	6.537	3.374	FF	CELL	3	M4K_X17_Y11	...onent|auto_generated|ram_block1a0|portbdataout[26]
7	7.080	0.543	FF	IC	1	LCCOMB_X16_Y11_N0	iPACKET_CHECK|Equal2~7|dataa
8	7.593	0.513	FF	CELL	1	LCCOMB_X16_Y11_N0	iPACKET_CHECK|Equal2~7|combout
9	8.428	0.835	FF	IC	1	LCCOMB_X18_Y11_N0	iPACKET_CHECK|Equal2~9|datac
10	8.750	0.322	FF	CELL	1	LCCOMB_X18_Y11_N0	iPACKET_CHECK|Equal2~9|combout
11	9.056	0.306	FF	IC	1	LCCOMB_X18_Y11_N24	iPACKET_CHECK|sop_error~0|datab
12	9.577	0.521	FR	CELL	1	LCCOMB_X18_Y11_N24	iPACKET_CHECK|sop_error~0|combout
13	9.577	0.000	RR	IC	1	LCFF_X18_Y11_N25	iPACKET_CHECK|sop_error|datain
14	9.673	0.096	RR	CELL	1	LCFF_X18_Y11_N25	PACKET_CHECK:iPACKET_CHECK|sop_error

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	5.000	5.000					latch edge time
2	⊟ 7.814	2.814					clock path
3	7.814	2.814	R				clock network delay
4	7.852	0.038		uTsu	1	LCFF_X18_Y11_N25	PACKET_CHECK:iPACKET_CHECK|sop_error

（太多了，懒得梳理，我也不是很记得，看看就行，想补充的自行阅读 ppt06）

建立时间检查公式：

*建立时间余量=最小数据要求时间（建立）-最长数据到达时间；

* Clock Setup Slack Time = Data Required Time - Data Arrival Time;
* Data Arrival Time = 时钟到达前级寄存器的时刻+ 前级寄存器时钟到后级寄存器数据输入的延迟;
* Data Required Time = 时钟到达后级寄存器的时刻 - 后级寄存器的建立时间

寄存器到寄存器：

*Clock Setup Slack = Data Required Time - Data Arrival Time;
* Data Arrival Time = Launch Edge + Clock Network Delay Source Register +

μtco + Register-to-Register Delay;
* Data Required Time = Clock Arrival Time - μtsu - Setup Uncertainty;
* Clock Arrival Time = Latch Edge + Clock Network Delay to Destination Register

输入端口到寄存器：
* Clock Setup Slack Time = Data Required Time－Data Arrival Time;
* Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + Input Maximum Delay of Pin + Pin-to-Register Delay;
* Data Required Time = Clock Arrival Time－μtsu;
* Clock Arrival Time = Latch Edge + Clock Network Delay to Destination Register

输出端口到寄存器：
* Clock Setup Slack Time = Data Required Time - Data Arrival Time;
* Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + μtco + Register-to-Pin Delay;
* Data Required Time = Clock Arrival Time - Output Maximum Delay of Pin;
* Clock Arrival Time = Latch Edge + Clock Network Delay to Destination Register

保持时间检查公式：
*保持时间余量= 数据到达时间-数据要求时间；

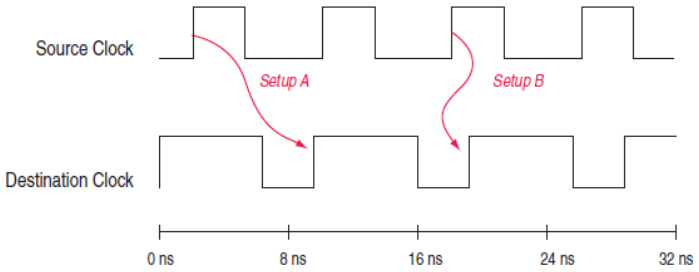
寄存器到寄存器：
* Clock Hold Slack = Data Arrival Time - Data Required Time;
* Data Arrival Time = Launch Edge + Clock Network Delay to Source Register +μtCO + Register to Register Delay;
* Data Required Time = Clock Arrival Time + μtH + Hold Uncertainty;
* Clock Arrival Time = Latch Edge + Clock Network Delay to Destination Register

输入端口到寄存器：
* Clock Hold Slack Time = Data Arrival Time - Data Required Time;
* Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + Input Minimum Delay of Pin + Pin to Register Delay;
* Data Required Time = Clock Arrival Time + μtH;
* Clock Arrival Time = Latch Edge + Clock Network Delay to Destination Register

输出端口到寄存器：
* Clock Hold Slack Time = Data Arrival Time - Data Required Time;
* Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + μtCO + Register to Pin Delay;
* Data Required Time = Clock Arrival Time - Output Minimum Delay of Pin;
* Clock Arrival Time = Latch Edge + Clock Network Delay to Destination Register

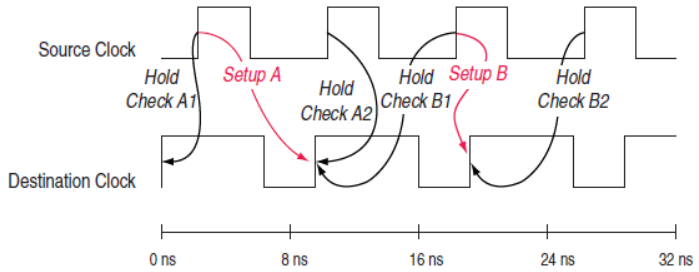
建立时间时序图

Figure 7–8. Setup Check



保持时间时序图

Figure 7–9. Hold Checks



48． 去抖的基本思想，及其核心代码。

去抖的基本思想是延时检测。

下面的这段代码就是去抖动的核心代码。

```
parameter T20ms = 20'd9;

reg key_r1, key_r2;

always @ (posedge clk or negedge rst_n)
  if(!rst_n)
    begin
      key_r1 <= 1'b1;
      key_r2 <= 1'b1;
    end
  else
    begin
      key_r1 <= key;
      key_r2 <= key_r1;
    end

wire key_negede = !key_r1 & key_r2;

reg [19:0] cnt;

always @ (posedge clk or negedge rst_n)
   if(!rst_n)
     cnt <= 20'd0;
   else if(key_negede)
```

```
    cnt <= 20'd0;
  else if (cnt > T20ms)
    cnt <= cnt;
  else if(!key)
    cnt <= cnt + 1'd1;
  else
    cnt <= cnt;
```

49. 给了相应的程序要能够判断出电路是否会生成锁存器，并解释为什么

生成原因

1.if 和 case 的情况没写全

2.没在所有条件下对信号进行赋值，同时单个 always 模块用了不同变量进行赋值并且没提前进行默认赋值

会生成锁存器的例子

①

```
always@( a or b)
    begin
        if(a==1)
            q <= b;
    end
```

缺少 else 语句,a 是其他值时会默认保持 q 的值

②

```
always @(*)
    begin
        if(d)
            a = b;
         else
            a = a;
    end
```

逻辑分支不同时使用到的变量不同

③

```
always @(b or d)
    case(d)
        2’b00: a=b>>1;
        2’b11: c=b>>1;
        default:
        begin
            a=b; c=b;
        end
    endcase
```

逻辑分支不同时使用到的变量不同

不会生成的例子

```
always @(b or d)
begin
    a=b; c=b;
    case(d)
```

```
        2’b00: a=b>>1;
         2’b11: c=b>>1;
    endcase
end
```

对 a 和 c 进行了默认赋值，且没有生成时序逻辑

非满分答案，希望本文的不足之处能作为读者独立解决问题的锻炼机会