**四、名词解释**

1. **SoC：**片上系统（System on Chip），将处理器、存储器、外设集成到单一芯片。

2. **LUT：**查找表（Look-Up Table），FPGA 中实现组合逻辑的基本单元。

3. **JTAG：**联合测试行动组接口，用于 FPGA 配置和调试。

4. **ISP：**在系统编程（In-System Programmability），无需移除芯片即可烧录。

5. **下载：**将设计文件写入 FPGA 配置存储器的过程。

6. **FPGA：**现场可编程门阵列，通过配置实现定制硬件电路。

7. **wire 型：**Verilog 中表示物理连线的数据类型，用于模块间连接。

8. **reg 型：**存储逻辑值的数据类型，常用于 always 块中的赋值。

9. **布局布线**：将逻辑网表映射到 FPGA 物理资源并连接的过程。

10. **逻辑综合**：将 RTL 代码转换为门级网表。

11. **RTL 设计**：寄存器传输级设计，描述数据在寄存器间的流动。

12. **时序约束**：指定时钟频率、信号延迟的设计要求。

13. **Verilog HD**L：硬件描述语言，用于数字电路建模。

14. **时序分析**：验证设计是否满足时序约束（如建立/保持时间）。

15. **综合优化**：在综合阶段减少面积或提高速度的技术。

16. **CPLD：**复杂可编程逻辑器件，基于乘积项结构。

17. **IP 核：知**识产权核，预设计的可重用功能模块。

18. **ISP 下载：**通过 JTAG 或专用接口在线烧录 FPGA。

19. **EDA：**电子设计自动化，涵盖电路设计、仿真、综合等工具。

20. **ASIC：**专用集成电路，为特定应用定制的芯片。

21. **FIFO：**先进先出存储器，用于数据缓冲。

22. **时序逻辑**：输出取决于当前输入和状态的电路。

23. **约束文件**：指定时序、引脚分配等约束的配置文件。

24. **行为级建**模：描述电路功能而非结构的抽象设计层次。

25. **综合优化**：综合优化是将 HDL 代码转换为 FPGA 硬件结构的过程中，编译器自动进行的逻辑优化。

26. **时序路径**：信号从起点到终点的路径。

27. **综合约束**： 综合约束是用户通过特定语法，向综合工具明确指定的优化目标或限制条件，用于指导综合过程。

28. **Testbench**：用于验证设计的测试环境。

29. **组合逻辑**：输出仅取决于当前输入的电路（无记忆性）。

30. **硬件仿真**：使用专用硬件加速仿真过程（如 FPGA 原型验证）。

五、电路设计题

1. 设计实现下表 1 功能的电路程序(模块名称 dff）

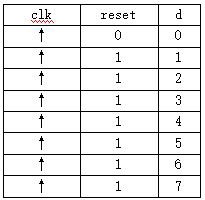
表 1 功能表

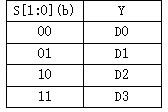
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| clk | clr | reset | d | q | qn |
| ↑ | 0 | x | x | 0 | 1 |
| ↑ | 1 | 0 | x | 1 | 0 |
| ↑ | 1 | 1 | ０ | ０ | １ |
| ↑ | 1 | 1 | １ | １ | ０ |

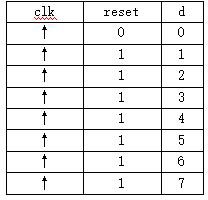
1. 设计实现下表 2 功能的电路程序(模块名称 decode）

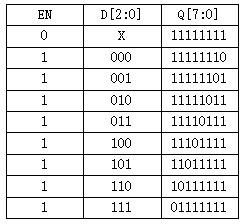
表 2 功能表

|  |  |  |
| --- | --- | --- |
| EN | D2D1D0(d) | Q[7:0](b) |
| 0 | x | 11111111 |
| 1 | 0 | 11111110 |
| 1 | 1 | 11111101 |
| 1 | 2 | 11111011 |
| 1 | 3 | 11110111 |
| 1 | 4 | 11101111 |
| 1 | 5 | 11011111 |
| 1 | 6 | 10111111 |
| 1 | 7 | 01111111 |









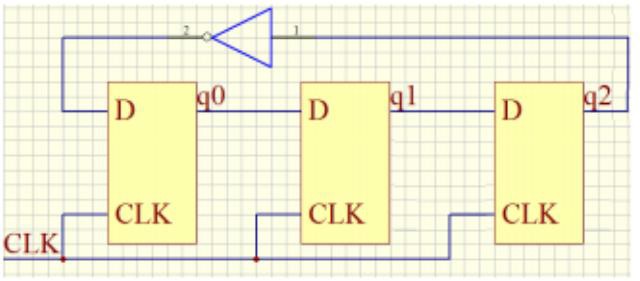


图 1 电路图

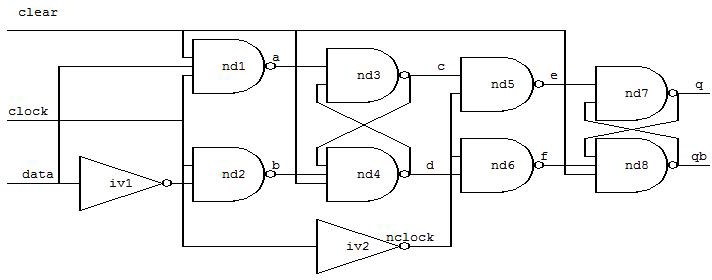


图 2 门级 D 触发器

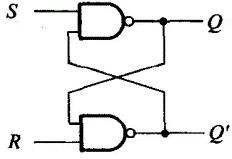


图 3 电路图

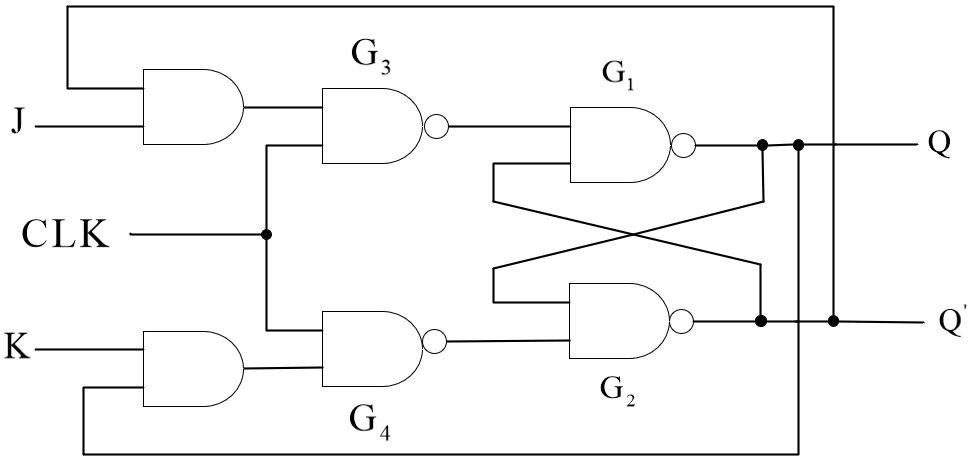


图 4 电路图

11.设计下图 5 电路图的程序(模块名 muxtwo)。

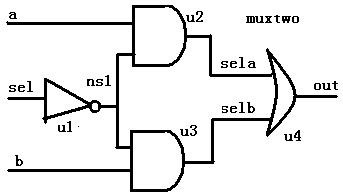


图 5 电路图

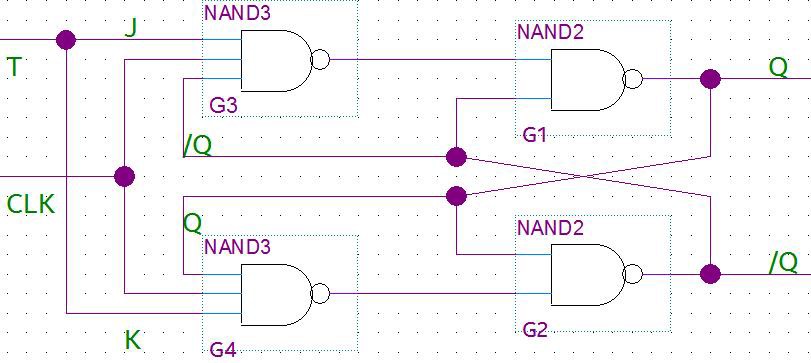


图 6 T 触发器电路图

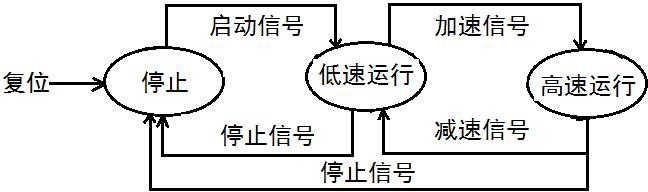


图 7 电机控制图

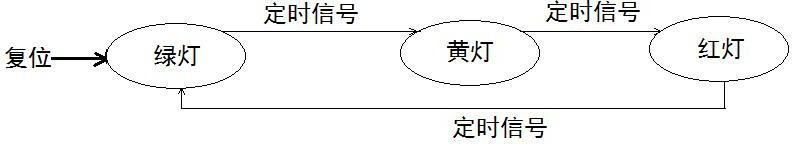


图 8 序列检测电路图

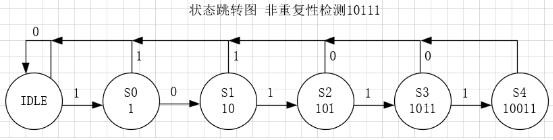


图 9 序列检测电路

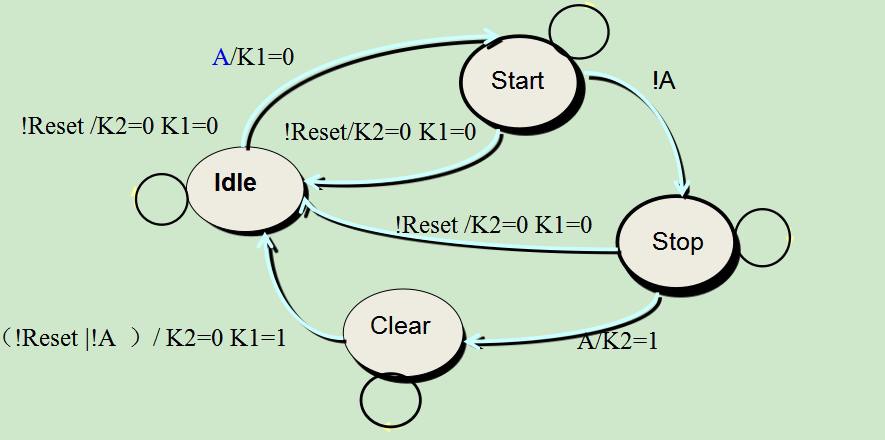


图 10 状态转移电路图

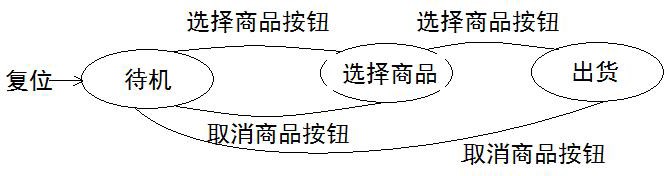


图 11 自动售货机电路

六、综合题

1.试分析下面程序，找出程序中存在的语法错误，写出正确的完整程序。

//对 CPA 四分频，且占空比为 1/3 module div4(CPA，CPB)

output CPB； input CPA;

Reg CPB;

always @ (posedge CPA) if (i==3)

begin i=0；CPB<=b’1；end else

begin i=i+1；CPB<=0；end

end endmodule

**答：**

|  |  |  |
| --- | --- | --- |
| **module** | **div4(CPA，CPB);** | |
| **output** | **reg CPB;** | |
| **input** | **wire CPA;** | |
| **reg** | **[1:0]i = 2’b00;** | |
| **always @ (posedge** | | **CPA)** |

**begin**

**if (i==2’b10)**

**begin i=2’b00;CPB<=2’b01;end**

**else**

**begin i=i+1;CPB<=0; end**

**end**

**endmodule**

//带异步清零的 JK 触发器逻辑功能描述 module JKCFQ(Q,J,K,CLK,RD)

output Q;

input J,K,CLK,RD; always @ (posedge CLK)

begin

if(!RD) Q<=0;

else case(J,K)

2’b00 : Q<=Q;2’b01 : Q<=0;

2’b10 : Q<=1;2’b11 : Q<=~Q;

default : Q<=1’bx; endcase

end Endmodule

**答：**

**module JKCFQ(Q,J,K,CLK,RD)**

**output reg [1:0]Q;**

**input J,K,CLK,RD;**

**always @ (posedge CLK)**

**begin**

**if(!RD) Q<=1’b0;**

**else**

**begin**

**case({J,K})**

**2’b00 : Q<=Q; 2’b01 : Q<=0;**

**2’b10 : Q<=1; 2’b11 : Q<=~Q;**

**default : Q<=1’bx;**

**endcase**

**end**

**end**

**endmodule**

//异步清零的 4 位计数器

module Hardreg(d,clk,clrb,q) input clk,clrb;

input [3:0]d;

output [3:0]q;

Reg [3:0]q;

always @ (posedge clk) if (clrb)

begin q<=b’1；end else

begin q<=d+1;end

end endmodule

**答：**

**module Hardreg(d,clk,clrb,q)**

**input clk,clrb;**

**input reg [3:0]d;**

**output reg [3:0]q;**

**always @ (posedge clk or negedge clrb)**

**begin**

**if (clrb == 0)**

**begin q<=4’b0000；end**

**else**

**begin q<=d+1;end**

**end**

**endmodule**

module Example(clk, data\_in, data\_out); input clk;

input [3:0] data\_in; output data\_out;

reg temp;

always @(posedge clk) begin temp = data\_in[0];

end

always @(data\_in) begin case(data\_in)

4'b0001: data\_out = 1'b1; 4'b0010: data\_out = 1'b0;

endcase

end

assign data\_out = temp;

**答：**

**module Example(clk, data\_in, data\_out);**

**input clk;**

**input reg [3:0] data\_in;**

**output data\_out;**

**reg temp;**

**always @(posedge clk) begin**

**temp <= data\_in[0];**

**end**

**always @(data\_in) begin**

**case(data\_in)**

**4'b0001: data\_out = 1'b1;**

**4'b0010: data\_out = 1'b0;**

**default: data\_out = temp;**

**endcase**

**end**

**endmodule**

module Counter(clk, rst, cnt) input clk, rst;

output cnt;

always @(posedge clk) begin if (rst) cnt = 0; else cnt <= cnt + 1;

always @(\*) begin

$display("Current count: %d", cnt); end

initial begin cnt <= 0; end

endmodule

**答：**

**module Counter(clk, rst, cnt)**

**input clk, rst;**

**output reg [3:0] cnt;**

**always @(posedge clk) begin**

**if (rst) cnt = 4’b0000;**

**else cnt <= cnt + 1;**

**always @(posedge clk) begin**

**$display("Current count: %d", cnt);**

**end**

**endmodule**

module Mux(input [1:0] sel, input [3:0] D, output Y); reg Y;

always @(sel or D) begin casez(sel)

2'b00: Y = D[0];

2'b01: Y = D[1];

2'b10: Y = D[2];

endcase

end

assign D = {D[2:0], 1'b0}; initial begin

#10 Y <= 0;

end

**答：**

**module Mux(input [1:0] sel, input [3:0] D, output reg Y);**

**always @(sel or D) begin**

**case(sel)**

**2'b00: Y = D[0];**

**2'b01: Y = D[1];**

**2'b10: Y = D[2];**

**default: Y = D[3];**

**endcase**

**end**

**endmodule**

1. 分析电路描述，画出产生的波形。 module CLK\_tb(clk);

output clk; reg clk; initial

begin

#3 clk = 1;

#4 clk = 0;

#6 clk = 1;

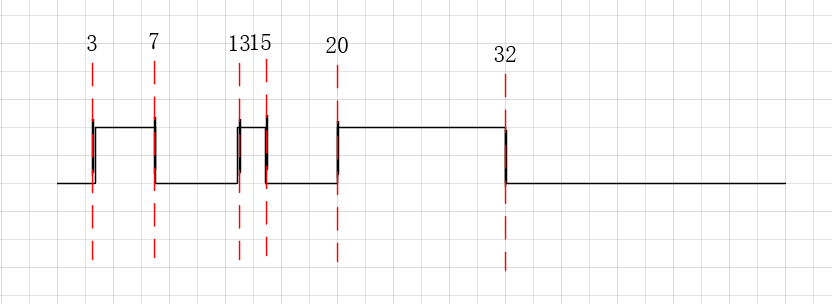
#2 clk = 0;

#5 clk = 1;

#12 clk = 0;

end endmodule

**答：**



1. 分析电路描述，画出产生的波形。 module Pulse (clk);

output clk; reg clk; initial

fork

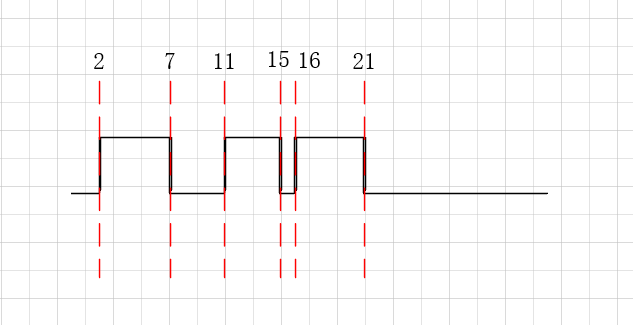
#2 clk = 1; #7 clk = 0;

#11 clk = 1;#15 clk = 0;

#16 clk = 1; #21 clk = 0; join

Endmodule

**答：**



1. 分析电路描述，画出产生的波形。 module Clk\_tb (clk);

output clk; reg clk;

initial clk = 1; initial

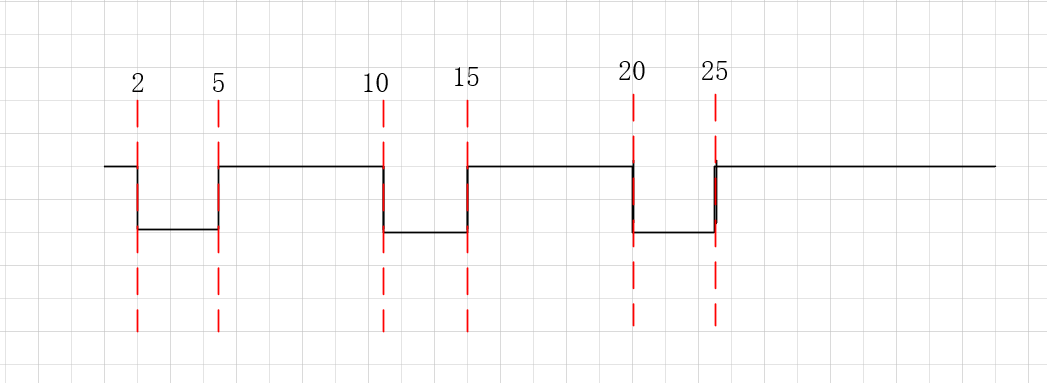
fork

#2 clk = ~clk; #5 clk = ~clk;

#10 clk = ~clk; #15 clk = ~clk; #20 clk = ~clk; #25 clk = ~clk;

join Endmodule

答：



1. 分析电路描述，画出产生的波形。 module WAVE\_tb;

reg clk; initial begin

clk = 0;

forever #5 clk = ~clk; end

endmodule

**答：**

1. 分析下图 12 中的波形，设计一个测试模块，产生同样的波形（`timescale 1us/1us）。

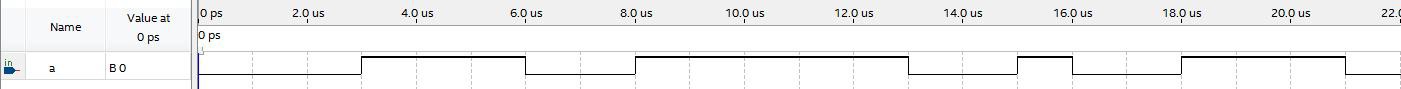


图 12 波形图

**答：**

`timescale 1us/1usmodule waveform\_tb;

// 声明测试信号

reg a;

initial begin

// 初始化信号

a = 0;

// 按照波形时序控制信号变化

// 0~4us：a=0

#4;

// 4~8us：a=1

a = 1;

#4;

// 8~12us：a=0

a = 0;

#4;

// 12~16us：a=1

a = 1;

#4;

// 16~20us：a=0

a = 0;

#4;

// 20~24us：a=1（延续波形趋势，可按需调整）

a = 1;

#4;

// 仿真结束（可根据需要延长或修改）

$stop;

end

endmodule

1. 分析下图 13 中的波形， 设计一个测试模块（ LED\_tb ） ， 产生同样的波形（ `timescale 1us/1us）。

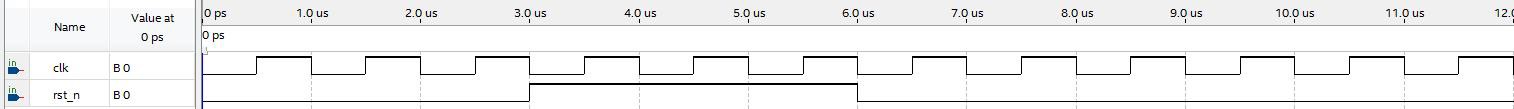


图 13 波形图

**答：**

`timescale 1us/1usmodule LED\_tb;

reg clk;

reg rst\_n;

initial begin

// 初始化信号

clk = 0;

rst\_n = 0;

// 生成 rst\_n 波形：先保持低电平，然后在 3us 时置高

#3 rst\_n = 1;

// 持续运行足够长时间以观察波形

#9; // 运行到 12us 左右，可根据需要调整

end

// 生成 clk 波形：每 1us 翻转一次（周期 2us ）

**always** #1 clk = ~clk;

endmodule

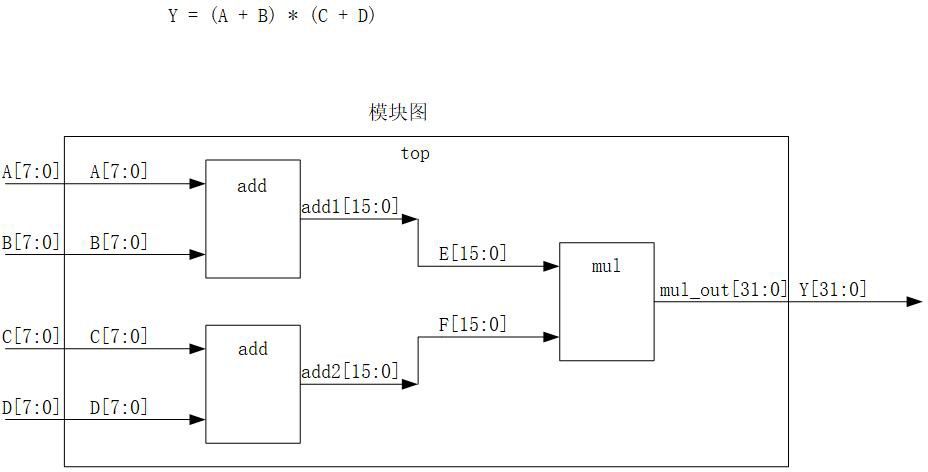


图 14 层次化设计模块图

**答：**

### **1. 加法模块（add）**

module add(

input [7:0] a,

input [7:0] b,

output [15:0] add\_out);

assign add\_out = a + b;

endmodule

### **2.乘法模块（mul）**

module mul(

input [15:0] e,

input [15:0] f,

output [31:0] mul\_out);

assign mul\_out = e \* f;

endmodule

### **3. 顶层模块（top）**

module top(

input [7:0] A,

input [7:0] B,

input [7:0] C,

input [7:0] D,

output [31:0] Y);

wire [15:0] add1\_out;

wire [15:0] add2\_out;

// 实例化第一个加法模块

add add\_inst1(

.a(A),

.b(B),

.add\_out(add1\_out)

);

// 实例化第二个加法模块

add add\_inst2(

.a(C),

.b(D),

.add\_out(add2\_out)

);

// 实例化乘法模块

mul mul\_inst(

.e(add1\_out),

.f(add2\_out),

.mul\_out(Y)

);

endmodule

### **测试模块（top\_tb）**

module top\_tb;

reg [7:0] A;

reg [7:0] B;

reg [7:0] C;

reg [7:0] D;

wire [31:0] Y;

// 实例化顶层模块

top top\_inst(

.A(A),

.B(B),

.C(C),

.D(D),

.Y(Y)

);

initial begin

// 测试用例 1

A = 8'd2;

B = 8'd3;

C = 8'd4;

D = 8'd5;

#10; // 等待 10 个时间单位，让电路稳定输出

// 预期结果：(2+3)\*(4+5) = 5\*9 = 45

// 测试用例 2

A = 8'd10;

B = 8'd20;

C = 8'd30;

D = 8'd40;

#10;

// 预期结果：(10+20)\*(30+40) = 30\*70 = 2100

// 测试用例 3（包含更大数值）

A = 8'd100;

B = 8'd150;

C = 8'd200;

D = 8'd250;

#10;

// 预期结果：(100+150)\*(200+250) = 250\*450 = 112500

$stop; // 停止仿真

end

endmodule

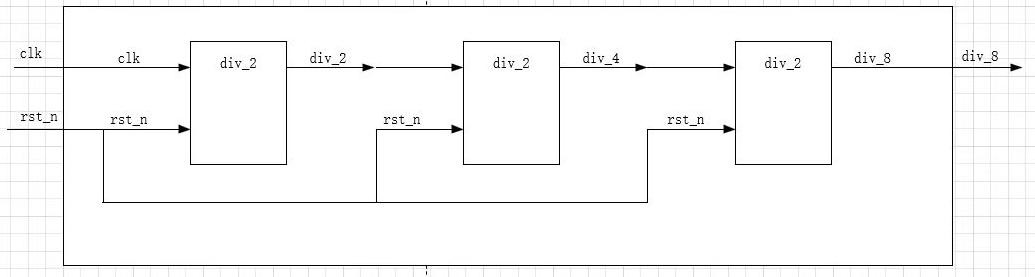


图 15 层次化设计模块图

**答：**

### **二分频模块（div\_2）**

module div\_2 (

input clk, // 输入时钟

input rst\_n, // 异步复位（低有效）

output reg div // 分频输出);

**always @**(posedge clk or negedge rst\_n) begin

if (!rst\_n) begin

div <= 1'b0; // 复位时输出低电平

end else begin

div <= ~div; // 时钟上升沿翻转输出

end

end

endmodule

### **八分频顶层模块（div\_8）**

module div\_8 (

input clk, // 系统时钟输入

input rst\_n, // 异步复位（低有效）

output div\_8 // 8 分频输出);

// 内部信号：二分频输出

wire div\_2\_out1; // 第一级二分频输出

wire div\_4\_out; // 第二级二分频（4 分频）输出

// 第一级二分频（2 分频）

div\_2 u\_div2\_1 (

.clk (clk),

.rst\_n (rst\_n),

.div (div\_2\_out1)

);

// 第二级二分频（4 分频）

div\_2 u\_div2\_2 (

.clk (div\_2\_out1),

.rst\_n (rst\_n),

.div (div\_4\_out)

);

// 第三级二分频（8 分频）

div\_2 u\_div2\_3 (

.clk (div\_4\_out),

.rst\_n (rst\_n),

.div (div\_8)

);

endmodule

### **测试模块（div\_8\_tb）**

module div\_8\_tb;

// 测试输入

reg clk;

reg rst\_n;

// 测试输出

wire div\_8;

// 实例化 8 分频模块

div\_8 uut (

.clk (clk),

.rst\_n (rst\_n),

.div\_8 (div\_8)

);

// 生成 50MHz 时钟（周期 20ns）

initial begin

clk = 0;

forever #10 clk = ~clk; // 每 10ns 翻转一次

end

// 测试用例

initial begin

// 1. 复位

rst\_n = 1'b0; // 复位有效

#100; // 等待 100ns

// 2. 释放复位

rst\_n = 1'b1; // 复位释放

#500; // 运行 500ns，观察 8 分频输出

// 3. 再次复位

rst\_n = 1'b0; // 复位有效

#100; // 等待 100ns

$stop; // 停止仿真

end

endmodule

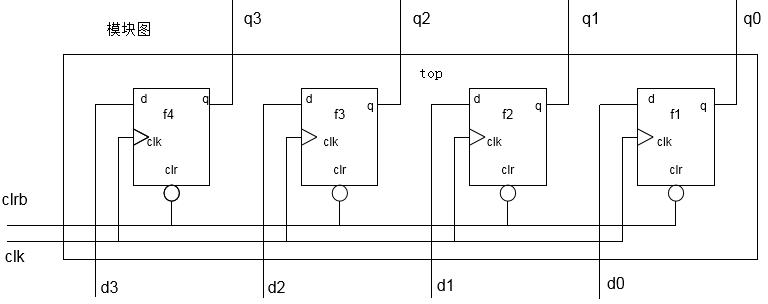


图 16 层次化设计模块图

**答：**

### **1. D 触发器模块（D\_FF ）**

module D\_FF (

input d,

input clk,

input clrb,

output reg q);

**always @**(posedge clk or negedge clrb) begin

if (!clrb) begin

q <= 1'b0; // 异步清零，低电平有效

end else begin

q <= d; // 时钟上升沿触发，将 d 的值存入 q

end

end

endmodule

### **4 位 D 触发器顶层模块（D\_4FF ）**

module D\_4FF (

input [3:0] d,

input clk,

input clrb,

output [3:0] q);

// 实例化 4 个 D 触发器

D\_FF ff0 (

.d(d[0]),

.clk(clk),

.clrb(clrb),

.q(q[0])

);

D\_FF ff1 (

.d(d[1]),

.clk(clk),

.clrb(clrb),

.q(q[1])

);

D\_FF ff2 (

.d(d[2]),

.clk(clk),

.clrb(clrb),

.q(q[2])

);

D\_FF ff3 (

.d(d[3]),

.clk(clk),

.clrb(clrb),

.q(q[3])

);

endmodule

### **3. 测试模块（d\_4\_tb ）**

module d\_4\_tb;

reg [3:0] d;

reg clk;

reg clrb;

wire [3:0] q;

// 实例化 4 位 D 触发器模块

D\_4FF uut (

.d(d),

.clk(clk),

.clrb(clrb),

.q(q)

);

// 生成时钟信号

initial begin

clk = 0;

forever #5 clk = ~clk; // 每 5 个时间单位翻转一次时钟，模拟时钟信号

end

// 测试激励

initial begin

// 初始化信号

d = 4'b0000;

clrb = 0; // 清零

#10; // 保持清零一段时间

clrb = 1; // 释放清零

// 输入数据并观察

d = 4'b1010;

#20; // 等待几个时钟周期

d = 4'b1111;

#20;

clrb = 0; // 再次清零

#10;

$stop; // 停止仿真

end

endmodule

数器，设计相关电路模块，同时设计测试模块（DIV\_16\_tb）。

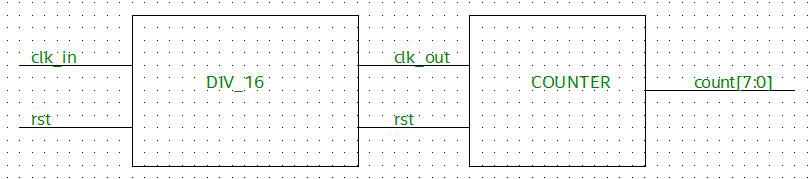


图 17 层次化设计模块图

**答：**

### **分频器模块 DIV\_16**

module DIV\_16 (

input clk\_in,

input rst,

output reg clk\_out);

reg [3:0] count;

**always @**(posedge clk\_in or posedge rst) begin

if (rst) begin

count <= 4'b0;

clk\_out <= 1'b0;

end else begin

if (count == 4'd7) begin // 计数到 7 时翻转时钟，实现 16 分频（从 0 数到 15 是 16 个周期 ）

count <= 4'b0;

clk\_out <= ~clk\_out;

end else begin

count <= count + 1'b1;

end

end

end

endmodule

### **2. 计数器模块 COUNTER**

module COUNTER (

input clk\_out,

input rst,

output reg [7:0] count);

**always @**(posedge clk\_out or posedge rst) begin

if (rst) begin

count <= 8'b0;

end else begin

count <= count + 1'b1;

end

end

endmodule

### **3. 顶层模块 TOP\_MODULE**

module TOP\_MODULE (

input clk\_in,

input rst,

output [7:0] count);

wire clk\_out;

// 实例化分频器模块

DIV\_16 u\_div (

.clk\_in(clk\_in),

.rst(rst),

.clk\_out(clk\_out)

);

// 实例化计数器模块

COUNTER u\_cnt (

.clk\_out(clk\_out),

.rst(rst),

.count(count)

);

endmodule

### **测试模块 DIV\_16\_tb**

module DIV\_16\_tb;

reg clk\_in;

reg rst;

wire [7:0] count;

// 实例化顶层模块

TOP\_MODULE u\_top (

.clk\_in(clk\_in),

.rst(rst),

.count(count)

);

// 生成时钟信号

initial begin

clk\_in = 0;

forever #5 clk\_in = ~clk\_in; // 每 5 个时间单位翻转一次，模拟 100MHz 左右时钟（周期 10 时间单位 ）

end

// 测试激励

initial begin

rst = 1;

#20; // 保持复位 20 时间单位

rst = 0;

#500; // 运行 500 时间单位，观察计数变化

$stop; // 停止仿真

end

endmodule

1. 根据下图 18 层次化模块图设计电路模块及测试模块（ 模块名 ADDER\_8BIT ， 测试模块

ADDER\_8BIT\_tb）。

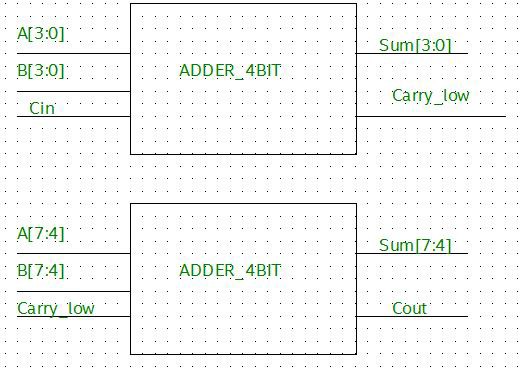


图 18 层次化设计模块图

**答：**

### **4 位加法器模块（子模块 ADDER\_4BIT）**

module ADDER\_4BIT (

input [3:0] A, // 4 位加数

input [3:0] B, // 4 位加数

input Cin, // 低位进位输入

output [3:0] Sum, // 4 位和

output Cout // 高位进位输出);

// 组合逻辑实现 4 位加法

assign {Cout, Sum} = A + B + Cin;

endmodule

### **8 位加法器模块（顶层模块 ADDER\_8BIT）**

module ADDER\_8BIT (

input [7:0] A, // 8 位加数

input [7:0] B, // 8 位加数

input Cin, // 最低位进位输入

output [7:0] Sum, // 8 位和

output Cout // 最高位进位输出);

// 内部信号：低位加法器的进位输出

wire Carry\_low;

// 实例化低位 4 位加法器（bit3~bit0）

ADDER\_4BIT adder\_low (

.A (A[3:0]),

.B (B[3:0]),

.Cin (Cin),

.Sum (Sum[3:0]),

.Cout (Carry\_low)

);

// 实例化高位 4 位加法器（bit7~bit4）

ADDER\_4BIT adder\_high (

.A (A[7:4]),

.B (B[7:4]),

.Cin (Carry\_low),

.Sum (Sum[7:4]),

.Cout (Cout)

);

endmodule

### **测试模块（ADDER\_8BIT\_tb）**

module ADDER\_8BIT\_tb;

// 测试输入

reg [7:0] A;

reg [7:0] B;

reg Cin;

// 测试输出

wire [7:0] Sum;

wire Cout;

// 实例化 8 位加法器

ADDER\_8BIT uut (

.A (A),

.B (B),

.Cin (Cin),

.Sum (Sum),

.Cout (Cout)

);

// 测试用例

initial begin

// 测试用例 1: 无进位

A = 8'd10; // 10

B = 8'd20; // 20

Cin = 1'b0; // 无进位

#10; // 等待仿真

// 预期结果：Sum=30, Cout=0

// 测试用例 2: 有进位

A = 8'd200; // 200

B = 8'd60; // 60

Cin = 1'b1; // 有进位

#10; // 等待仿真

// 预期结果：200+60+1=261 → Sum=8'd261[7:0]=5'd261（二进制 100000101，截断为 8 位是 00000101？不对，200+60=260 +1=261，8 位二进制是 100000101，但 8 位只能存 0~255，所以实际是 261-256=5，Sum=5，Cout=1）

// 测试用例 3: 最大值

A = 8'd255; // 255

B = 8'd255; // 255

Cin = 1'b1; // 进位

#10; // 等待仿真

// 预期结果：255+255+1=511 → Sum=8'd511[7:0]=255（511-256\*2= -1？不对，实际是 511 的 8 位二进制是 11111111，Sum=255，Cout=1）

$stop; // 停止仿真

end

endmodule

1. 根据层次化模块图 19 设计电路程序及测试模块（模块名 MULTIPLEXER，测试模块 mux\_tb）。

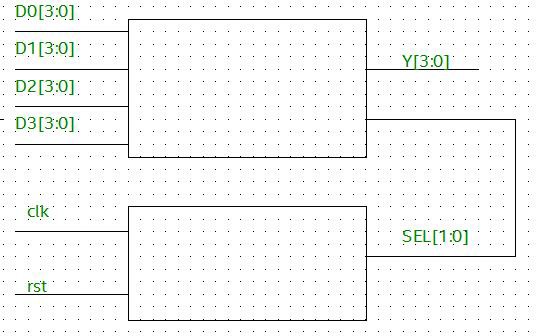


图 19 层次化设计模块图

**答：**

### 顶层模块 MULTIPLEXER 代码、

module MULTIPLEXER(

input clk,

input rst,

input [3:0] D0,

input [3:0] D1,

input [3:0] D2,

input [3:0] D3,

output [3:0] Y);

// 内部信号声明，用于连接选择信号生成模块和多路选择器wire [1:0] SEL;

// 实例化选择信号生成模块（这里假设是计数器模块，你可根据实际功能修改）

SEL\_GENERATOR sel\_gen (

.clk(clk),

.rst(rst),

.SEL(SEL));

// 实例化4选1多路选择器模块

MUX\_4TO1 mux (

.D0(D0),

.D1(D1),

.D2(D2),

.D3(D3),

.SEL(SEL),

.Y(Y));

endmodule

// 选择信号生成模块（以简单计数器为例，假设实现循环计数生成SEL[1:0]）module SEL\_GENERATOR(

input clk,

input rst,

output reg [1:0] SEL);

**always @**(posedge clk or posedge rst) begin

if (rst) begin

SEL <= 2'b00;

end else begin

SEL <= SEL + 1'b1;

endend

endmodule

// 4选1多路选择器模块module MUX\_4TO1(

input [3:0] D0,

input [3:0] D1,

input [3:0] D2,

input [3:0] D3,

input [1:0] SEL,

output reg [3:0] Y);

**always @**(\*) begin

case (SEL)

2'b00: Y = D0;

2'b01: Y = D1;

2'b10: Y = D2;

2'b11: Y = D3;

default: Y = 4'b0000;

endcase

end

endmodule

### 测试模块 mux\_tb 代码

module mux\_tb;

// 测试模块输入输出信号声明reg clk;reg rst;reg [3:0] D0;reg [3:0] D1;reg [3:0] D2;reg [3:0] D3;wire [3:0] Y;

// 实例化被测模块

MULTIPLEXER uut (

.clk(clk),

.rst(rst),

.D0(D0),

.D1(D1),

.D2(D2),

.D3(D3),

.Y(Y));

// 时钟生成initial begin

clk = 0;

forever #5 clk = ~clk; // 每5个时间单位翻转一次时钟，模拟时钟信号end

// 测试激励initial begin

// 初始化信号

rst = 1;

D0 = 4'b0001;

D1 = 4'b0010;

D2 = 4'b0100;

D3 = 4'b1000;

#10; // 等待一段时间

rst = 0; // 释放复位

#50; // 运行一段时间，观察输出变化

// 改变输入数据，继续测试

D0 = 4'b0011;

D1 = 4'b0110;

D2 = 4'b1100;

D3 = 4'b1001;

#50;

$stop; // 停止仿真end

endmodule