Verilog HDL语言

Verilo简单数字电路设计

主讲：卢　萍

华中科技大学计算机科学与技术学院

主要内容

- 组合电路设计
- 时序电路设计
- 数据通路
- 有限状态机

Verilog组合电路设计

- 全加器
- 比较器
- 译码器
- 选择器

全加器

```
module full_adder(a, b, c_in, sum, c_out);
   parameter WIDTH = 8; //本module内有效，可用于参数传递
   input  [WIDTH-1:0]    a, b;
   input  c_in;
   output [WIDTH-1:0]    s;
   output                c_out;

   assign { c_out, sum } = a + b + c_in;
endmodule

// 调用语法
full_adder  #(4) add(x,y, ci, s,co);  //调用时可以像
                                      端口信号传递一样进行参数传递
full_adder  #(.WIDTH(32))  add( x,y, ci, s,co);  //名称关联
```

比较器

```
module comparator(A, B, is_equal, is_great, is_less);
    parameter WIDTH = 8;
    input [WIDTH-1:0] A, B;
    output is_equal, is_great, is_less;

    assign is_equal = (A==B)? 1'b1 : 1'b0;
    assign is_great = (A>B)?    1'b1 : 1'b0;
    assign is_less    = (A<B)?    1'b1 : 1'b0;
endmodule
```

- 译码器：是一种具有“翻译”功能的逻辑电路，能将输入二进制代码的各种状态，按照其原意翻译成对应的输出信号。译码器在数字系统中有广泛的用途，如，终端的数字显示，存贮器寻址等。

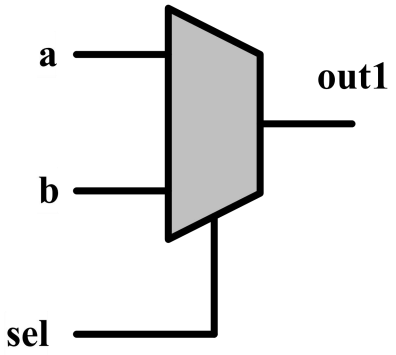
  - **n---2^n线译码器、显示译码器（二进制数 -> 七段码）**

3-8 译码器

```
module decoder_38(out,in);
    output reg [7:0] out;
    input      [2:0] in;
      always @(in) begin
          case(in)
              3'd0: out=8'b11111110;
              3'd1: out=8'b11111101;
              3'd2: out=8'b11111011;
              3'd3: out=8'b11110111;
              3'd4: out=8'b11101111;
              3'd5: out=8'b11011111;
              3'd6: out=8'b10111111;
              3'd7: out=8'b01111111;
           endcase
    end
endmodule
```

2选1 选择器

```
module mux2_1(out1, a, b, sel) ;
  parameter WIDTH = 8;

   output      [WIDTH-1:0] out1;
   input       [WIDTH-1:0] a, b;
   input                   sel;

   assign      out1 =  sel == 0: a? b ;

endmodule
```



- 4选1 选择器（用if-else 语句描述）

```
module MUX4_1(out, in0, in1, in2, in3, sel);
    output out;
    input in0, in1, in2, in3;
    input[1:0] sel;
    reg out;

    always @ (in0 or in1 or in2 or in3 or sel)
    begin
        if (sel==2'b00)       out=in0;
        else if (sel==2'b01) out=in1;
        else if (sel==2'b10) out=in2;
        else                          out=in3;
    end
endmodule
```

- 4选1 选择器（用case 语句描述）

```
module MUX4_1(out, in0, in1, in2, in3, sel);
    output out;
    input in0,in1,in2,in3;
    input[1:0] sel;
    reg out;

    always @ (in0 or in1 or in2 or in3 or sel)
    begin
        case(sel)
            2'b00: out=in0;
            2'b01: out=in1;
            2'b10: out=in2;
            default: out=in3;
        endcase
    end
endmodule
```

if-else和case的区别

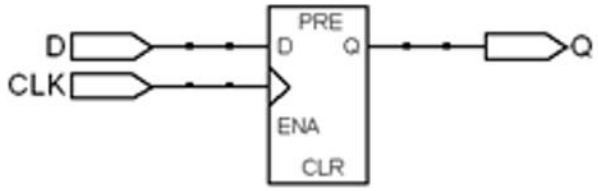
- case语句各个分支之间是平行的，优先级相同；
  if语句则具有优先级。
- 每个else if 都可以判断不同的条件，比较灵活; case语句比较的是一个公共的控制表达式.
- if-else结构速度较慢；case结构速度较快。

Verilog时序电路设计

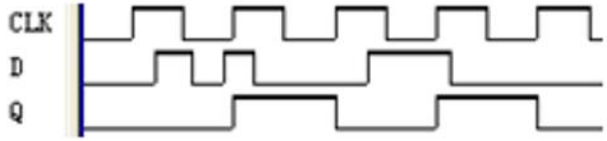
- 触发器
- 锁存器
- 计数器
- 寄存器
- 存储器

基本D触发器

- 触发器（Flip-flop）是具有记忆功能的二进制存储器
- 由时钟信号的跳边沿触发“存储”



边沿触发型 D 触发器



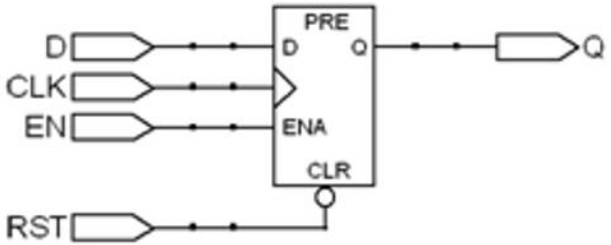
D 触发器时序波形

```
module DFF1(CLK,D,Q);
     output Q ;
     input  CLK, D ;
      reg Q;
   always @(posedge CLK )
      Q <= D;
Endmodule
```

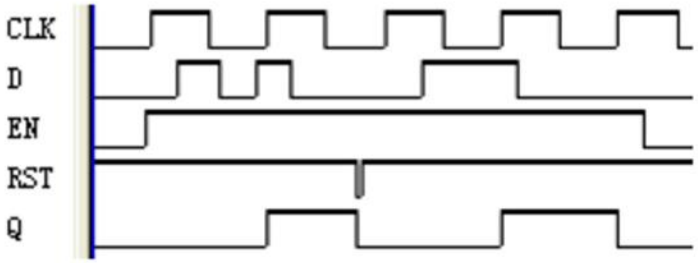
- <= :非阻塞赋值, 使用在时序逻辑电路中
- = : 阻塞赋值, 用于组合逻辑电路

含异步复位和时钟使能的D触发器

- 异步复位：复位信号不受时钟控制，通常“低电平”有效
- 使能信号：“启用”信号，它有效时（高电平），触发器才工作
- 复位的优先级是最高的



含使能和复位控制的 D 触发器

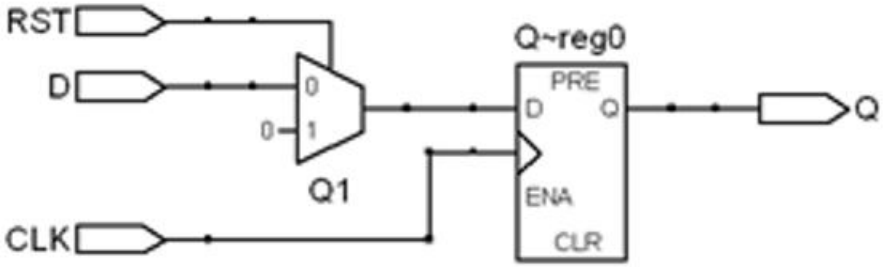


D 触发器的时序图

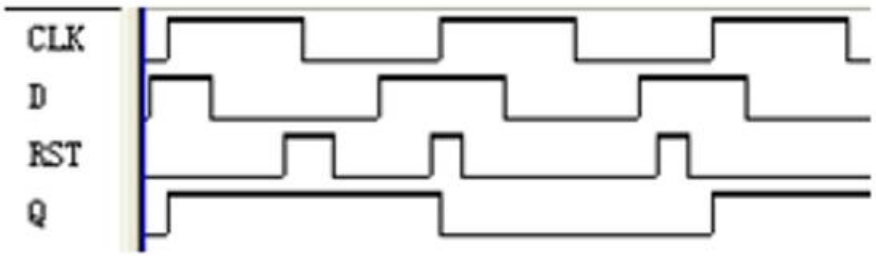
```
module DFF2(CLK,D,Q,RST,EN);
  output Q;
  input CLK,D,RST,EN;
  reg Q;
always @(posedge CLK or negedge RST)
   begin
        if (!RST)  Q <= 0;
   else if (EN)    Q <= D;
   end
endmodule
```

含同步复位的D触发器

- 同步复位：复位受时钟控制，在时钟信号跳边沿检测复位信号



含同步清 0 控制的 D 触发器

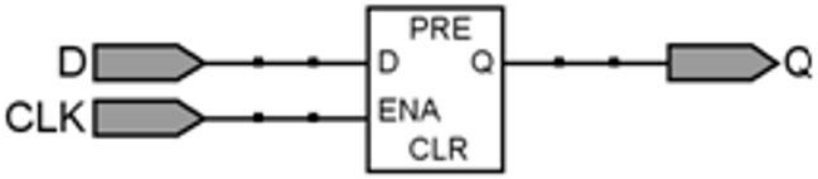


含同步清 0 控制 D 触发器的时序图

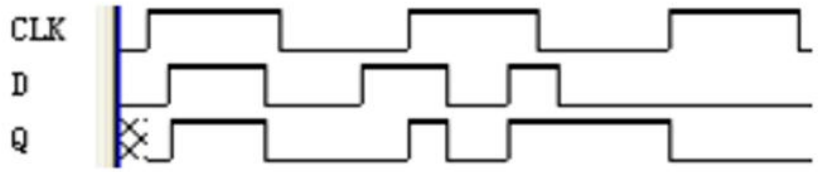
```
module DFF3(CLK,D,Q,RST);
    output Q ;
    input CLK,D,RST ;
    reg Q;
    always @(posedge CLK )
        if (RST==1)  Q = 0;
   else if (RST==0)  Q = D;
endmodule
```

基本锁存器

- 锁存器（Latch）也是具有记忆功能的二进制存储器
- 由电平信号触发“存储”



锁存器模块

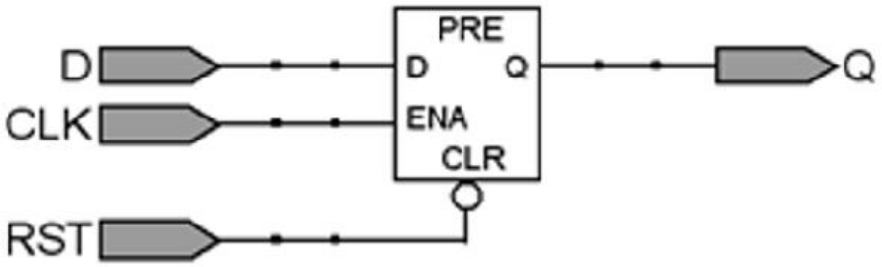


锁存器的时序波形

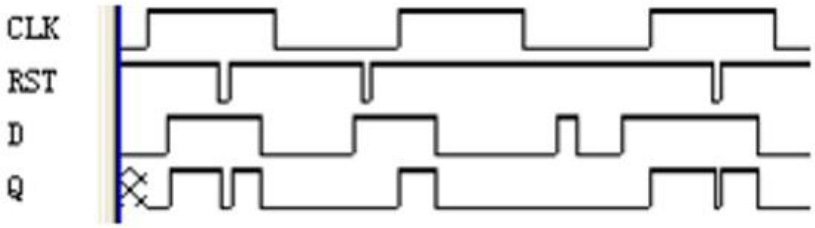
```
module LATCH1(CLK,D,Q);
      output Q ; input CLK,D;
      reg Q;
    always @(D or CLK)
      if(CLK)  Q <= D;
endmodule
```

- CLK 为高时，数据可以通过，输出端随输入而变，即存入新的数据
- CLK为低时，输出端被锁定，即记忆刚才存入的数据
- CLK 被称为 LE (latch enable)

含异步复位的锁存器



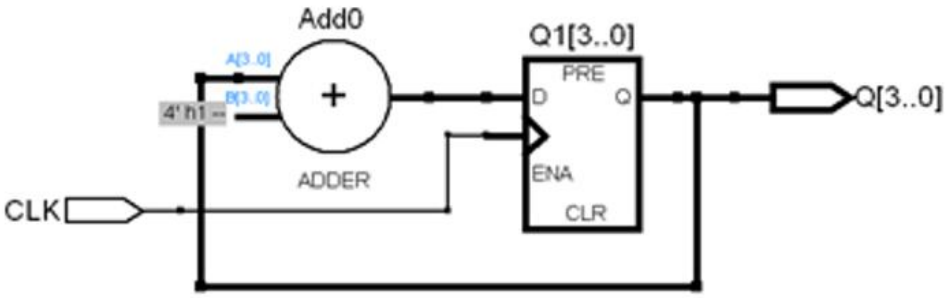
含异步清 0 的锁存器



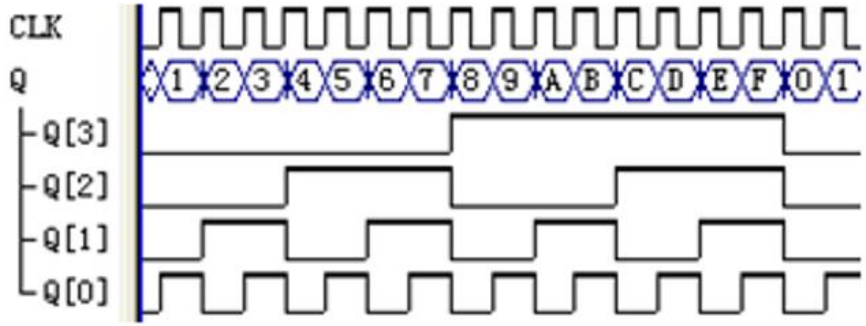
含异步清 0 的锁存器的仿真波形

```
module LATCH3 (CLK,D,Q,RST);
    output Q ;
    input CLK,D,RST;
    reg Q;
    always @(D or CLK or RST)
            if(!RST)  Q<=0;
      else  if(CLK)   Q<=D;
endmodule
```

简单加法计数器



4 位加法计数器 RTL 电路图

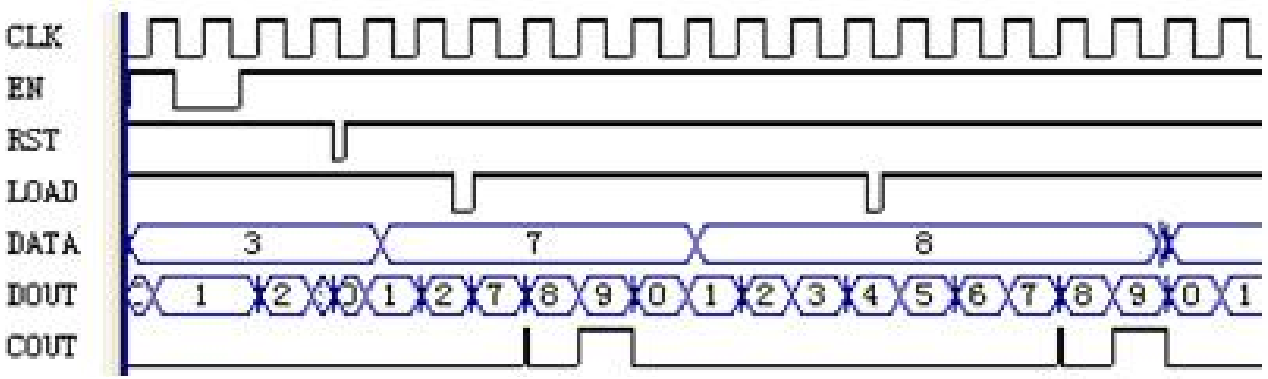


4 位加法计数器工作时序

```
module CNT4 (CLK,Q);
    output [3:0] Q ;
    input  CLK;
     reg [3:0] Q ;
   always @(posedge CLK)
       Q <= Q+1 ;
  endmodule
```

实用加法计数器

- 异步复位 RST,“低电平”有效（RST_n）
- 使能 EN
- 数据加载 LOAD：预置一个计数初值data，“低电平”有效
- 计数初值data:
- 计数溢出 cout：计数满时的输出信号



实用加法计数器

```
module CNT10 (CLK,RST,EN,LOAD,COUT,DOUT,DATA);
   input CLK,EN,RST,LOAD ;      // 时钟，时钟使能，复位，数据加载控制信号;
   input [3:0] DATA ;           // 4 位并行加载数据
   output [3:0] DOUT ;          // 4 位计数输出
   output COUT ;                // 计数进位输出
   reg [3:0] Q1 ;     reg COUT ;
   assign DOUT = Q1;            // 将内部寄存器的计数结果输出至 DOUT
   always @(posedge CLK or negedge RST)   //时序过程
     begin
           if (!RST)   Q1 <= 0;     //RST=0 时，对内部寄存器单元异步清 0
      else  if (EN)  begin          //同步使能 EN=1，则允许加载或计数
             if (!LOAD)     Q1<=DATA;  //当 LOAD=0，向内部寄存器加载数据
          else  if (Q1<9)  Q1 <= Q1+1; //当 Q1 小于 9 时，允许累加
          else   Q1 <= 4'b0000; end    //否则一个时钟后清 0 返回初值
      end
   always @(Q1)                        //组合过程
       if (Q1==4'h9)  COUT = 1'b1;  else   COUT = 1'b0;
   endmodule
```

寄存器

- 用来暂存数据的存储单元。
- 用触发器来实现

```
module register(clk, rst_n, en, d, q);
  parameter WIDTH = 8;
  input clk, rst_n, en;
  input [WIDTH-1:0] d;
  output reg [WIDTH-1:0] q;

  always @(posedge clk) begin
    if (!rst_n) q <=0;
    else if (en) q <= d;
  end
endmodule
```

存储器

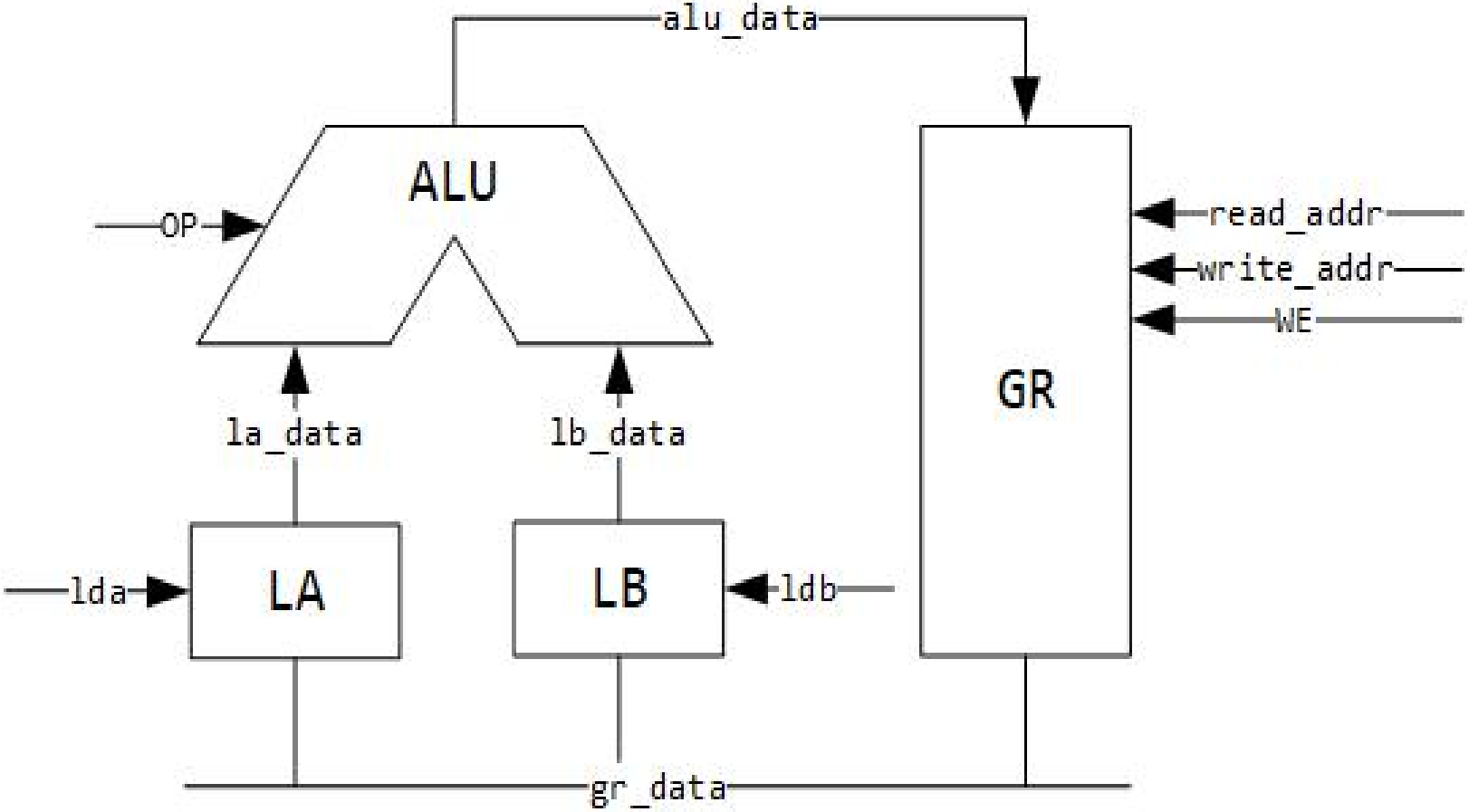
- 用来存放数据的若干存储单元
  - ROM
  - RAM
- 有地址信号 addr
- 声明 reg 型数组
  - 数组大小size 取决于地址宽度addr_width
  - size = 2^{addr_width}
  - reg [n-1:0] mem[2**m-1:0]; // m个n位的存储器

    n 为数据宽度，m为地址宽度

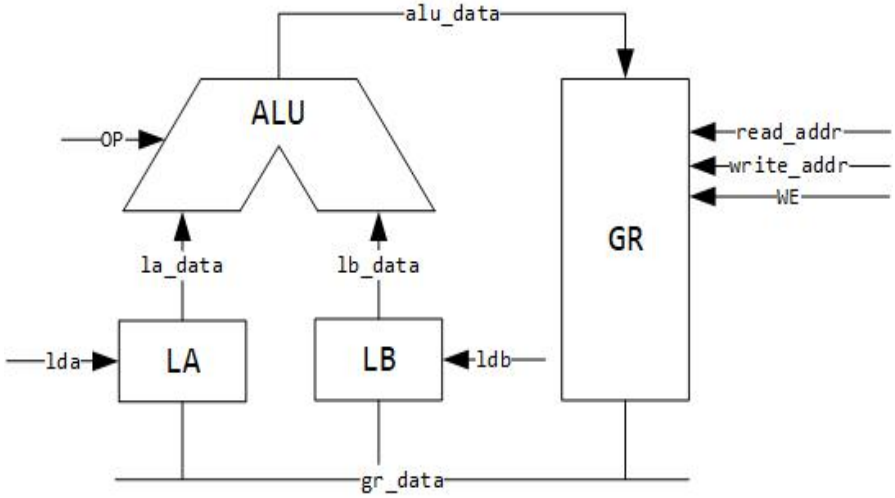
```
// 双端口存储器：同一个存储器具有两组相互独立的读写控制线路
module dual_port_ram( data, read_addr, write_addr, clk, we, q)
    parameter DATA_WIDTH=8;
    parameter ADDR_WIDTH=6;

    input [DATA_WIDTH-1:0]   data;
    input [ADDR_WIDTH-1:0]  read_addr, write_addr;
    input we, clk;          // we--写使能，高有效
    output reg [DATA_WIDTH-1:0]  q ;

    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];  // 申明reg型数组

    initial
      $readmemh("ram_init.txt", ram); //从文件中读取数据到存储器ram

    always @ (posedge clk) begin
          if (we)    ram[write_addr] <= data;    // 将data写入存储器
          q <= ram[read_addr];                   // 读数据由q输出
    end
endmodule
```

数据通路

数据通路示例


alu_data
ALU
OP
read_addr
write_addr
WE
GR
la_data
lb_data
lda
LA
LB
ldb
gr_data

```
module alu_circuit(a, b, op, result);
    parameter WIDTH = 8;
    input [WIDTH-1:0] a, b;
    input [1:0] op;
    output reg [WIDTH-1:0] result;

    always @* begin
        case(op)
            2'b00: result = a + b;
            2'b01: result = a & b;
            2'b10: result = a ^ b;
            2'b11: result = a | b;
            default: result = 0;
        endcase
    end
endmodule
```



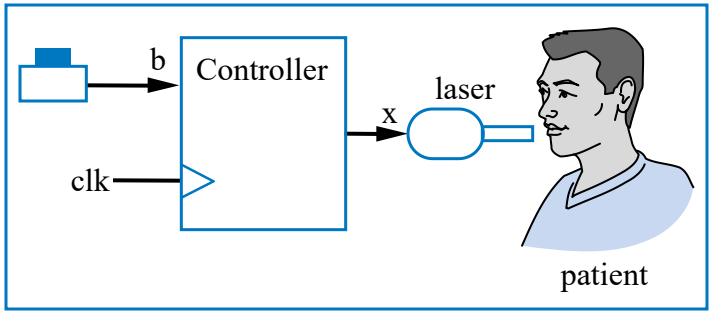

```
module datapath_top (clk, rst, lda, ldb, read_addr, write_addr,we, op)
    wire [31:0] la_data, lb_data, alu_data, gr_data
    wire lda, ldb, we;
    wire [4:0] read_addr, write_addr;
    wire [1:0] op;

     register      #(32) LA (clk, rst, lda, gr_data,la_data);
     register      #(32) LB (clk, rst, ldb, gr_data,lb_data);
  dual_port_ram #(32, 5) GR (alu_data, read_addr, write_addr, we, clk, gr_data);
    alu_circuit #(32) ALU (la_data, lb_data, op, alu_data);

endmodule
```

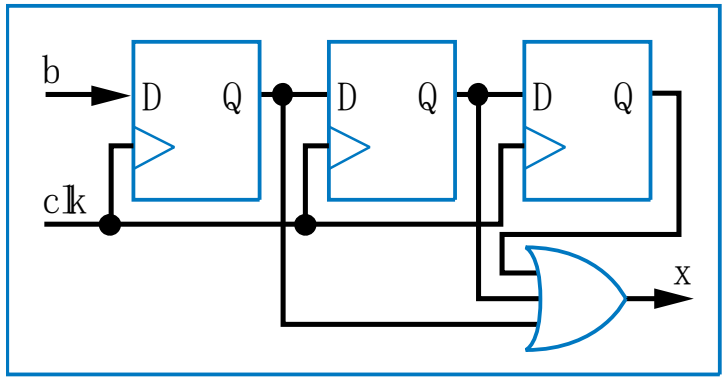
有限状态机

FSM：Finite State Machine

例：激光计时器

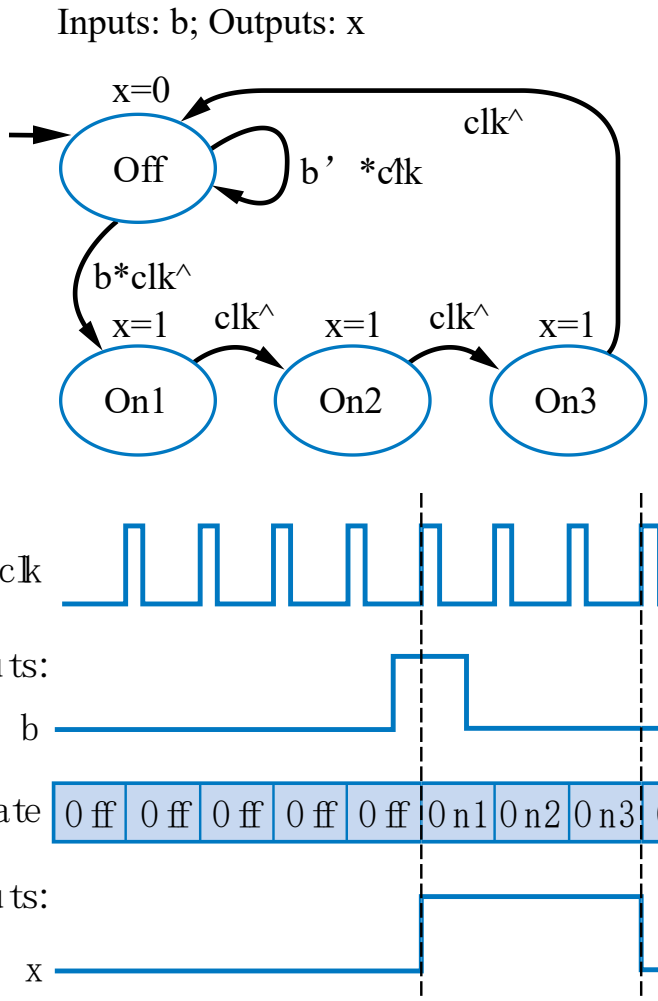
- 要求：按1次按钮b， x输出3个周期宽的1
- 方案1：使用 3 个级联的 D 触发器和1个或门。
- 方案2：使用有限状态机
  - FSM是一种用来设计时序逻辑的通用模型





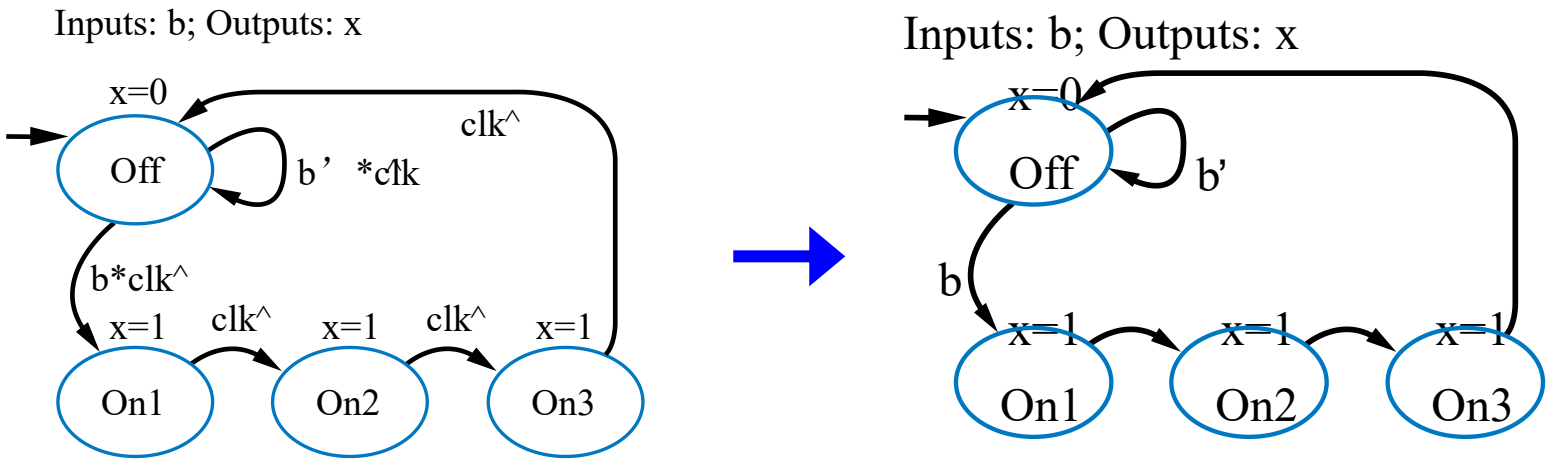
基于FSM的激光计时器

- **有限状态机的要素：现态、条件（事件）、动作、次态，**当事件发生后，根据当前状态，决定执行的动作，并设置下一个状态。
- **状态机解题的两个主要步骤**
  - ✓ 确定系统总共有几个状态
  - ✓ 确定状态之间的迁移过程（状态迁移图）
- **激光计时器**
  - ✓ 4个状态: {Off, On1, On2, On3}
  - ✓ b为0时 “Off” 态 （初始状态)
  - ✓ b为1时 (且时钟上升沿), 迁移到“On1”态，x输出1
  - ✓ 接着的2个时钟沿，分别迁移到“On2”，“On3”， x均输出1
  - ✓ 按钮b按下后，x=1保持了3个时钟周期



基于FSM的激光计时器

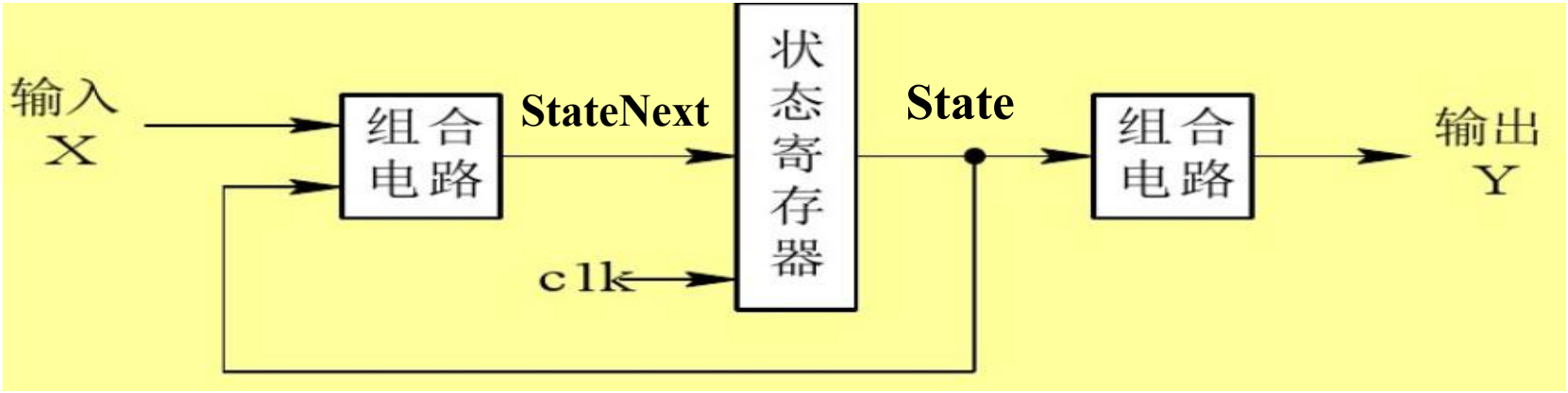
- 对于同步时序电路，clk^ 是必然迁移条件。



FSM的设计

涉及三方面内容：

- 实现状态迁移：在 clk 的上升沿将次态迁移到现态，是典型的“同步时序逻辑”。
- 确定下一个状态：与现态及输入条件有关，与 clk 无关，是“组合逻辑”。
- 确定输出：输出仅和现态有关（ Moore 型 FSM），输出和现态、输入有关（ Mealy 型），是“组合逻辑”。

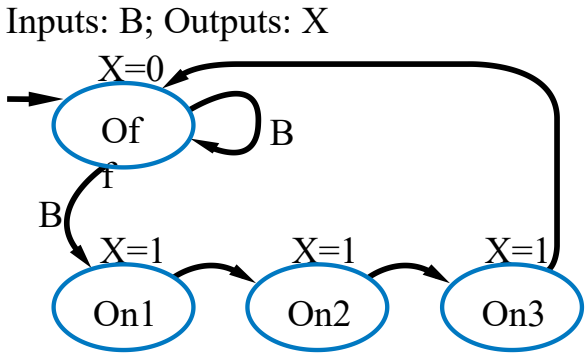


Moore 型 FSM的典型结构

FSM的Verilog实现

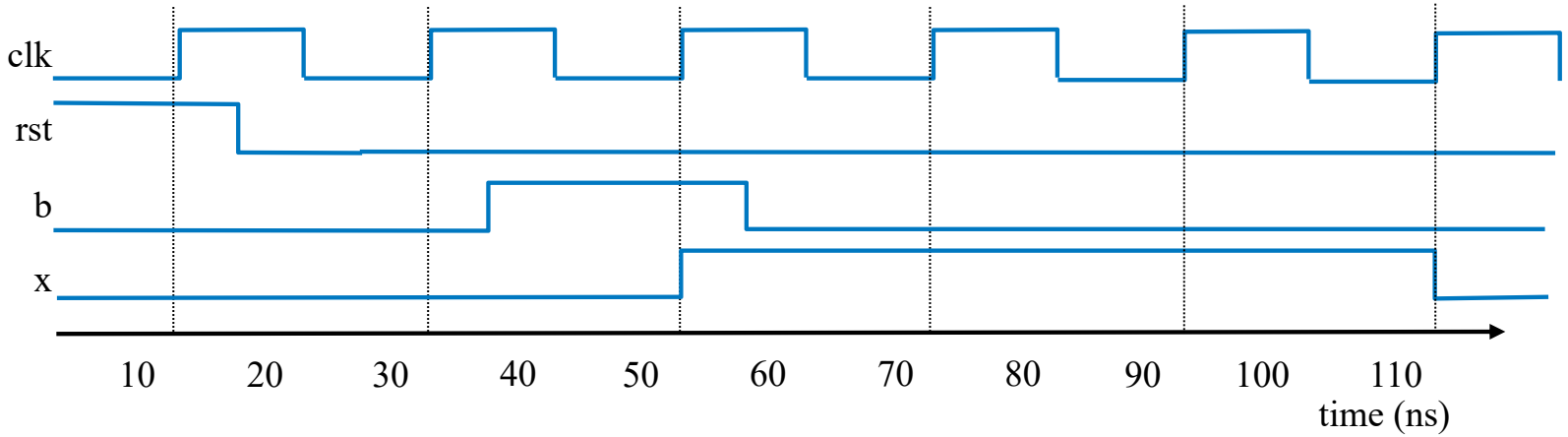
- 一段式
  - 把 FSM 的 3 部分内容写到一个always块里；
  - 缺点：结构不清晰，不容易调试，易出错。
- 两段式
  - 用两个always块，其中一个采用同步时序描述状态转移；另一个采用组合逻辑判断状态转移条件，描述状态转移规律和输出。
  - 优点：结构清晰，容易调试。
- 三段式
  - 用三个always块，一个采用同步时序描述状态转移，一个采用组合逻辑判断状态转移条件，描述状态转移规律，另一个采用同步时序描述输出。
  - 优点：相比两段式，输出毛刺少，更适用于全同步时序电路

二段式FSM的模板




```
module LaserTimer(Clk, Rst, B, X);

   input Clk, Rst;
   input B;
   output reg X;

   localparam Off = 0, On1 = 1,
              On2 = 2, On3 = 3;

   reg [1:0] State, StateNext;

   // 组合逻辑
   always @(State, B) begin
      case (State)
         Off: begin
            X = 0;    // 输出
            if (B == 0) //状态转移条件
               StateNext = Off;
            else
               StateNext = On1;
          end
        On1: begin
            X = 1;
            StateNext = On2;
         end
        On2: begin
            X = 1;
            StateNext = On3;
          end
        On3: begin
            X = 1;
            StateNext = Off;
         end
      endcase
   end

   // 状态寄存器
   always @(posedge Clk) begin
      if (Rst == 1 )  //同步复位使状态初始化
         State <= Off; //FSM的初始态
      else
         State <= StateNext; //状态转移
   end

endmodule
```

LaserTimer_tb




```
`timescale 1ns / 1ps

module laser_timer_tb( );

   reg clk, rst, b;
   wire x;

   LaserTimer dut(clk, rst, b, x);

   // 产生时钟信号
   always begin
      clk = 0;
      #10;
      clk = 1;
      #10;
   end

   initial begin
      rst = 1;          // 复位信号
      b = 0;            // 未按键
      @(posedge clk);
      #5 rst = 0;
      @(posedge clk);
      #5 b = 1;              // 按下按键
      @(posedge clk);
      #5 b = 0;              // 松开按键
      @(posedge clk);
      @(posedge clk);
      @(posedge clk);
   end
endmodule
```

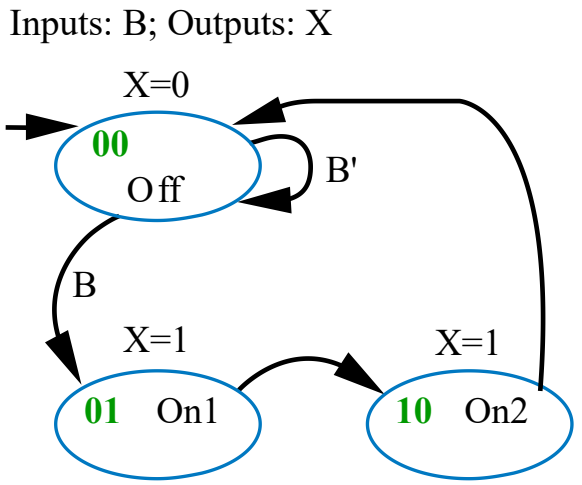
TestBench的自检

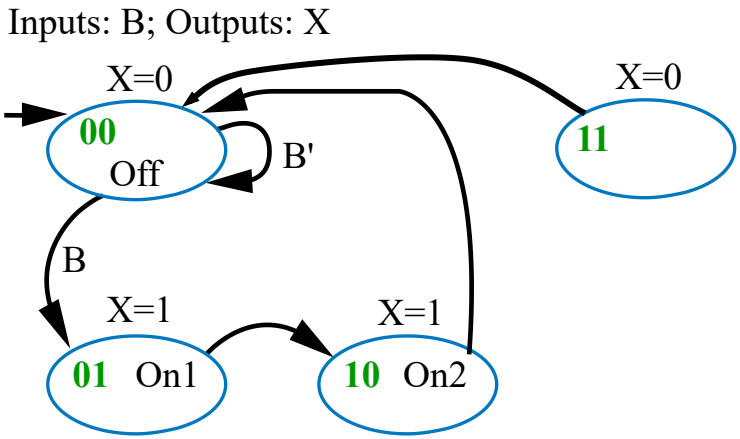
- 读波形容易出错
- 带有自检的测试文件：*自动将真实的结果和预期的结果进行比对。*

```
`timescale 1ns / 1ps

module laser_timer_tb( );

   reg clk, rst, b;
   wire x;

   LaserTimer dut(clk, rst, b, x);

   always begin // 产生时钟信号
      clk = 0;
      #10;
      clk = 1;
      #10;
   end

   initial begin
      rst = 1;        // 复位信号
      b = 0;          // 未按键
      @(posedge clk);
      #5 if (x != 0)
       $display("%t: Reset failed", $time);
      rst = 0;
      @(posedge clk);
      #5 b = 1;          // 按下按键
      @(posedge clk);
      #5 b = 0;          // 松开按键
      if (x != 1)
        $display("%t: First x=1 failed", $time);
      @(posedge clk);
      #5 if (x != 1)
        $display("%t: Second x=1 failed", $time);
      @(posedge clk);
      #5 if (x != 1)
        $display("%t: Third x=1 failed", $time);
      @(posedge clk);
      #5 if (x != 0)
        $display("%t: Final x=0 failed", $time);

       end
   endmodule
```

安全的 FSM

- **安全的 FSM：如果进入非法状态，可以自动转移到合法状态。**
- 示例：右上图状态机只有三个状态
  - ✓2-bit 状态码，其中11是非法状态（不可达状态）
  - ✓正常情况下，状态机不会进入该状态
  - ✓但是，噪声等干扰会引起误码，0->1 或 1->0
- **安全设计：**能够从错误中恢复，转移到合法状态





描述安全的FSM

```
...
    reg [1:0] State, StateNext;

    always @(State, B) begin
      case (State)
         S_Off: begin
            X = 0;
            if (B == 0)
               StateNext = S_Off;
            else
               StateNext = S_On1;
         end
         S_On1: begin
            X = 1;
            StateNext = S_On2;
         end
         S_On2: begin
            X = 1;
            StateNext = S_Off;
         end
         default: begin
            X = 0;
            StateNext = S_Off;
         end
      endcase
    end
...
```