```
      addr_write:          //地址的输入。
        begin
        if(finish_F==0)
          begin  shift8_out; end
        else
          begin
             sh8out_state <= idle;
             sh8out_buf   <= data;
                    state <= data_write;
                finish_F <= 0;
          end
        end

      data_write:        //数据的写入。
        begin
        if(finish_F==0)
          begin  shift8_out; end
        else
          begin
              link_write <= 0;
                   state <= stop_ack;
                 finish_F <= 0;
                      ack <= 1;
          end
        end

      stop_ack:              //完成应答。
        begin
          ack <= 0;
          state <= idle;
        end

      endcase
    end

task shift8_out;                 //串行写入。
  begin
     case(sh8out_state)

     idle:
       begin
           link_write  <= 1;
          sh8out_state <= bit0;
```

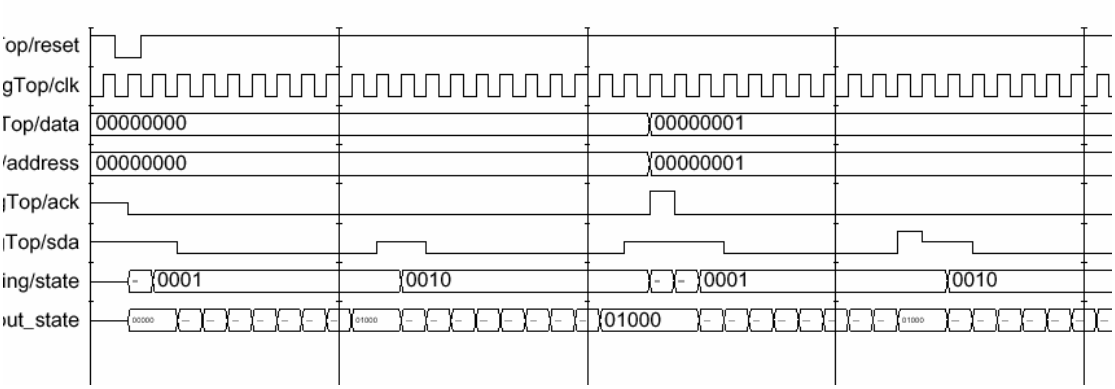
```
    end

bit0:
  begin
      link_write <= 1;
    sh8out_state <= bit1;
      sh8out_buf <= sh8out_buf<<1;
  end

bit1:
  begin
    sh8out_state<=bit2;
    sh8out_buf<=sh8out_buf<<1;
  end

bit2:
  begin
    sh8out_state<=bit3;
    sh8out_buf<=sh8out_buf<<1;
  end

bit3:
  begin
    sh8out_state<=bit4;
    sh8out_buf<=sh8out_buf<<1;
  end

bit4:
  begin
    sh8out_state<=bit5;
    sh8out_buf<=sh8out_buf<<1;
  end

bit5:
  begin
    sh8out_state<=bit6;
    sh8out_buf<=sh8out_buf<<1;
  end

bit6:
  begin
    sh8out_state<=bit7;
    sh8out_buf<=sh8out_buf<<1;
  end
```

```
     bit7:
       begin
         link_write<= 0;
         finish_F<=finish_F+1;
       end

     endcase
  end
endtask

endmodule
```

测试模块源代码：

```
`timescale 1ns/100ps
`define clk_cycle 50
module writingTop;
  reg reset,clk;
  reg[7:0] data,address;
  wire ack,sda;

  always #`clk_cycle  clk = ~clk;

  initial
    begin
        clk=0;
        reset=1;
        data=0;
        address=0;
        #(2*`clk_cycle) reset=0;
        #(2*`clk_cycle) reset=1;
        #(100*`clk_cycle) $stop;
    end

  always @(posedge ack)      //接收到应答信号后，给出下一个处理对象。
    begin
        data=data+1;
        address=address+1;
    end
  writing writing(.reset(reset),.clk(clk),.data(data),
                  .address(address),.ack(ack),.sda(sda));
endmodule
```

仿真波形：



练习：仿照上例，编写一个实现 EPROM 内数据串行读取的模块。编写测试模块，给出仿真波形。

练习十. 通过模块之间的调用实现自顶向下的设计

目的：学习状态机的嵌套使用实现层次化、结构化设计。

现代硬件系统的设计过程与软件系统的开发相似，设计一个大规模的集成电路的往往由模块多层次的引用和组合构成。 层次化、结构化的设计过程，能使复杂的系统容易控制和调试。 在 Verilog HDL 中，上层模块引用下层模块与 C 语言中程序调用有些类似，被引用的子模块在综合时作为其父模块的一部分被综合，形成相应的电路结构。在进行模块实例引用时，必须注意的是模块之间对应的端口，即子模块的端口与父模块的内部信号必须明确无误地一一对应，否则容易产生意想不到的后果。

下面给出的例子是设计中遇到的一个实例，其功能是将并行数据转化为串行数据送交外部电路编码，并将解码后得到的串行数据转化为并行数据交由 CPU 处理。显而易见，这实际上是两个独立的逻辑功能，分别设计为独立的模块，然后再合并为一个模块显得目的明确、层次清晰。

```
// -----------------    p_to_s.v ------------------------------------
module  p_to_s(D_in,TO,data,SEND,ESC,ADD_100);
    output        D_in,TO;              // D_in 是串行输出，TO 是移位时钟并给
                                        // CPU 中断，以确定何时给出下个数据。
    input   [7:0] data;                 //并行输入的数据。
    input         SEND,ESC,ADD_100;     //SEND、ESC 共同决定是否进行并到串
                                        //的数据转化。ADD_100 决定何时置数。
    wire      D_in,TO;
    reg [7:0] DATA_Q,DATA_Q_buf;

    assign        TO = ! (SEND & ESC);      //形成移位时钟。.
    assign        D_in = DATA_Q[7];         //给出串行数据。

    always @(posedge TO or negedge ADD_100)  //ADD_100 下沿置数，TO 上沿移位。
      begin
```

```
        if(!ADD_100)
          DATA_Q = data;
        else
          begin
          DATA_Q_buf = DATA_Q<<1;          //DATA_Q_buf 作为中介，以令综合器
          DATA_Q = DATA_Q_buf;             //能辨明。
          end
      end

endmodule
```

在 p_to_s.v 中，由于移位运算虽然可综合，但是不是简单的 RTL 级描述，直接用 DATA_Q<=DATA_Q<<1 的写法在综合时会令综合器产生误解。另外，在该设计中，由于时钟 T0 的频率较低，所以没有象以往那样采用低电平置数，而是采用 ADD_100 的下降沿置数。

```
//---------------------- s_to_p.v ----------------------------
module s_to_p(T1, data, D_out,DSC,TAKE,ADD_101);
       output       T1;                      //给 CPU 中断，以确定 CPU 何时取转化
                                             //得到的并行数据。
       output [7:0] data;
       input   D_out, DSC, TAKE, ADD_101;  //D_out 提供输入串行数据。DSC、TAKE
                                             //共同决定何时取数。
       wire   [7:0] data;
       wire         T1,clk2;
       reg    [7:0] data_latch, data_latch_buf;

       assign       clk2 = DSC  & TAKE ;   //提供移位时钟。
       assign       T1 = !clk2;

       assign       data =  (!ADD_101) ? data_latch : 8'bz;
       always@(posedge clk2)
            begin
               data_latch_buf = data_latch << 1;   //data_latch_buf 作缓冲
               data_latch     = data_latch_buf;   //，以令综合器能辩明。
                data_latch[0] = D_out;
           end
endmodule
```

将上面的两个模块合并起来的 sys.v 的源代码：

```
//------------------- sys.v ----------------------------
`include "./p_to_s.v"
`include "./s_to_p.v"
module sys(D_in,T0,T1, data, D_out,SEND,ESC,DSC,TAKE,ADD_100,ADD_101);
  input         D_out,SEND,ESC,DSC,TAKE,ADD_100,ADD_101;
  inout  [7:0]  data;
  output        D_in,T0,T1;
```

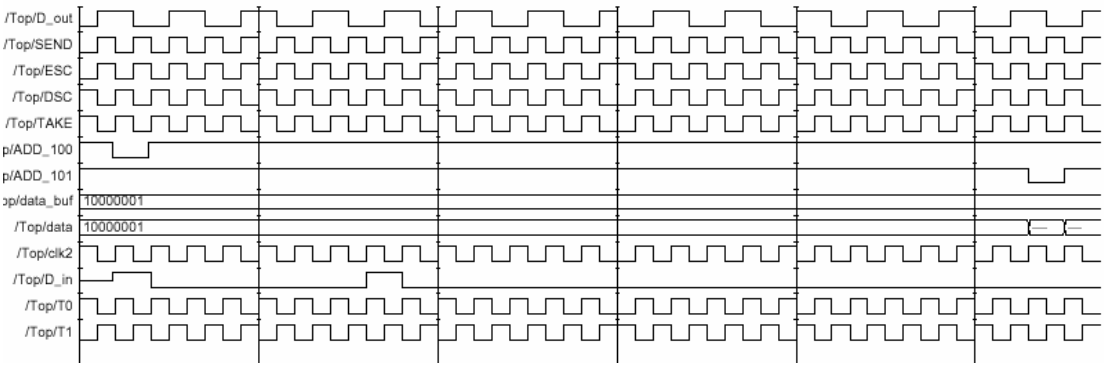
```
  p_to_s   p_to_s(.D_in(D_in),.T0(T0),.data(data),
                .SEND(SEND),.ESC(ESC),.ADD_100(ADD_100));
  s_to_p   s_to_p(.T1(T1),.data(data),.D_out(D_out),
          .DSC(DSC),.TAKE(TAKE),.ADD_101(ADD_101));

endmodule
```

测试模块源代码：

```
//-------------Top test file for sys.v ------------------
`timescale 1ns/100ps
`include "./sys.v"
module Top;
 reg D_out,SEND,ESC,DSC,TAKE,ADD_100,ADD_101;
 reg[7:0] data_buf;
 wire [7:0] data;
 wire clk2;

 assign  data = (ADD_101) ? data_buf : 8'bz;
                                  //data 在 sys 中是 inout 型变量，ADD_101
                                  //控制 data 是作为输入还是进行输出。
assign  clk2 =DSC && TAKE;

initial
  begin
     SEND = 0;
     ESC = 0;
     DSC = 1;
     TAKE = 1;
     ADD_100 = 1;
     ADD_101 = 1;
  end

initial
  begin
    data_buf = 8'b10000001;
    #90 ADD_100 = 0;
    #100 ADD_100 = 1;
  end

always
  begin
    #50;
    SEND = ~SEND;
```

```
    ESC = ~ESC;
  end

initial
  begin
    #1500 ;
    SEND = 0;
    ESC  = 0;
    DSC  = 1;
    TAKE = 1;
    ADD_100 = 1;
    ADD_101 = 1;
    D_out = 0;
    #1150 ADD_101 = 0;
    #100 ADD_101 =1;
    #100 $stop;
  end

always
  begin
    #50 ;
    DSC = ~DSC;
    TAKE = ~TAKE;
  end

always @(negedge clk2) D_out = ~D_out;

sys    sys(.D_in(D_in),.T0(T0),.T1(T1),.data(data),.D_out(D_out),
             .ADD_101(ADD_101), .SEND(SEND),.ESC(ESC),.DSC(DSC),
                                .TAKE(TAKE),.ADD_100(ADD_100));
endmodule
```

仿真波形：



练习：设计一个序列发生器。要求根据输入的 8 位并行数据输出串行数据,如果输入数据在 0—127 之间则输出一位 0，如果输入数据在 128—255 之间则输出一位 1，同步时钟触发；并且和范例 8 的序列检测器搭接，形成一个封闭系统。编写测试模块，并给出仿真波形。

练习十一. 简单卷积器的设计

下面我们将共同来完成一个用于教学的但有实际接口器件背景的小型设计—“简单卷积器的设计”。希望通过这个设计，使同学们建立起专用数字计算系统设计的基本概念。设计分成许多步骤进行，具体过程排列如下：

1）明确设计任务：

在设计之前必须明确设计的具体内容。卷积器是数字信号处理系统中常用的部件。它对模拟输入信号实时采样，得到数字信号序列。然后对数字信号进行卷积运算，再将卷积结果存入RAM中。对模拟信号的采样由A/D转换器来完成，而卷积过程由卷积器来实现。为了设计卷积器，首先要设计RAM和A/D转换器的Verilog HDL模型。在电子工业发达的国家，可以通过商业渠道得到非常准确的外围器件的虚拟模型。如果没有外围器件的虚拟模型，就需要仔细地阅读和分析RAM和A/D转换器的器件说明书，来自行编写。因为RAM和A/D转换器不是我们设计的硬件对象，所以需要的只是它们的行为模型，精确的行为模型需要认真细致地编写，并不比可综合模块容易编写。它们与实际器件的吻合程度直接影响设计的成功。在这里我们把重点放在卷积器的设计上，直接给出RAM 和A/D转换器的Verilog HDL模型和它们的器件参数(见附录),同学们可以对照器件手册,认真阅读RAM和A/D转换器的Verilog HDL模型。对RAM 和A/D转换器的Verilog HDL模型的详细了解对卷积器的设计是十分必要的。

到目前为止，我们对设计模块要完成的功能比较明确了。总结如下：首先它要控制AD变换器进行AD变换，从AD变换器得到变换后的数字序列，然后对数字序列进行卷积，最后将结果存入RAM。 下面让我们一起来设计它。

2）卷积器的设计

通过前面的练习我们已经知道，用高层次的设计方法来设计复杂的时序逻辑，重点是把时序逻辑抽象为有限状态机，并用可综合风格的Verilog HDL把这样的状态机描述出来。 下面我们将通过注释来介绍整个程序的设计过程。我们选择8位输入总线，输出到RAM的数据总线也选择8位，卷积值的高、低字节被分别写到两个RAM中。地址总线为11位。为了理解卷积器设计中的状态机，必须对A/D转换器和RAM的行为模块有深入的理解。

```
`timescale 100ps/100ps
module  con1(address,indata,outdata,wr,nconvst,nbusy,
                                        enout1,enout2,CLK,reset,start);

 input  CLK,      //采用10MHZ的时钟
        reset,    //复位信号
        start,    //因为RAM的空间是有限的，当RAM存满后采样和卷积都会停止。
                  //此时给一个start的高电平脉冲将会开始下一次的卷积。
        nbusy;    //从A/D转换器来的信号表示转换器的忙或闲
 output wr,       //RAM 写控制信号
        enout1,enout2, //enout1是存储卷积低字节结果RAM的片选信号
                       //enout2是存储卷积高字节结果RAM的片选信号
```

```
       nconvst, //给 A/D 转换器的控制信号,命令转换器开始工作,低电平有效
       address;  //地址输出

input [7:0]  indata;  //从 A/D 转换器来的数据总线
output[7:0] outdata;  //写到 RAM 去的数据总线

wire  nbusy;
reg   wr;
reg   nconvst,
      enout1,
      enout2;
reg[7:0]  outdata;

reg[10:0] address;
reg[8:0]  state;
reg[15:0] result;
reg[23:0] line;
reg[11:0] counter;
reg    high;
reg[4:0] j;
reg    EOC;



parameter  h1=1,h2=2,h3=3;  //假设的系统系数
parameter  IDLE=9'b000000001,  START=9'b000000010,  NCONVST=9'b000000100,
           READ=9'b000001000,  CALCU=9'b000010000,  WRREADY=9'b000100000,
             WR=9'b001000000,  WREND=9'b010000000,  WAITFOR=9'b100000000;

parameter  FMAX=20;  //因为 A/D 转换的时间是随机的，为保证按一定的频率采样，A/D
                     //转换控制信号应以一定频率给出。这里采样频率通过 FMAX 控制
                     // 为 500KHZ。

always @(posedge CLK)
      if(!reset)
         begin
              state<=IDLE;
              nconvst<=1'b1;
              enout1<=1;
              enout2<=1;
              counter<=12'b0;
              high<=0;
              wr<=1;
              line<=24'b0;
```