实验八　状态机及键盘输入

2020 年秋季学期

We know the state of the system if we know the sequence of symbols on the tape, which of these are observed by the computer (possibly with a special order), and the state of mind of the computer.

–"On Computable Numbers, with an Application to the Entscheidungsproblem", A. M. Turing

有限状态机 FSM（Finite State Machine）简称状态机，是一个在有限个状态间进行转换和动作的计算模型。有限状态机含有一个起始状态、一个输入列表（列表中包含了所有可能的输入信号序列）、一个状态转移函数和一个输出端，状态机在工作时由状态转移函数根据当前状态和输入信号确定下一个状态和输出。状态机一般都从起始状态开始，根据输入信号由状态转移函数决定状态机的下一个状态。

有限状态机是数字电路系统中十分重要的电路模块，是一种输出取决于过去输入和当前输入的时序逻辑电路，它是组合逻辑电路和时序逻辑电路的组合。其中组合逻辑分为两个部分，一个是用于产生有限状态机下一个状态的次态逻辑，另一个是用于产生输出信号的输出逻辑，次态逻辑的功能是确定有限状态机的下一个状态；输出逻辑的功能是确定有限状态机的输出。除了输入和输出外，状态机还有一组具有 "记忆" 功能的寄存器，这些寄存器的功能是记忆有限状态机的内部状态，常被称作状态寄存器。

本实验的目的是学习状态机的工作原理，了解状态机的编码方式，并利用 PS/2 键盘输入实现简单状态机的设计。

8.1 状态机

8.1.1 有限状态机

在实际应用中，有限状态机被分为两种：Moore（摩尔）型有限状态机和 Mealy（米里）型有限状态机。Moore 型有限状态机的输出信号只与有限状态机的当前状态有关，与输入信号的当前值无关，输入信号的当前值只会影响到状态机的次态，不会影响状态机当前的输出。即 Moore 型有限状态机的输出信号

是直接由状态寄存器译码得到。Moore 型有限状态机在时钟 CLK 信号有效后经过一段时间的延迟，输出达到稳定值。即使在这个时钟周期内输入信号发生变化，输出也会在这个完整的时钟周期内保持稳定值而不变。输入对输出的影响要到下一个时钟周期才能反映出来。Moore 有限状态机最重要的特点就是将输入与输出信号隔离开来。Mealy 状态机与 Moore 有限状态机不同，Mealy 有限状态机的输出不仅仅与状态机的当前状态有关，而且与输入信号的当前值也有关。Mealy 有限状态机的输出直接受输入信号的当前值影响，而输入信号可能在一个时钟周期内任意时刻变化，这使得 Mealy 有限状态机对输入的响应发生在当前时钟周期，比 Moore 有限状态机对输入信号的响应要早一个周期。因此，输入信号的噪声可能影响到输出的信号。

8.1.2 简单状态机

本节通过设计一个实际的状态机来了解状态机的工作过程和设计方法。

请设计一个区别两种特定时序的有限状态机 FSM：该有限状态机有一个输入 w 和一个输出 z。当 w 是 4 个连续的 0 或 4 个连续的 1 时，输出 z=1，否则 z=0，时序允许重叠。即：若 w 是连续的 5 个 1 时，则在第 4 个和第 5 个时钟之后，z 均为 1。图 8-1是这个有限状态机的时序图。

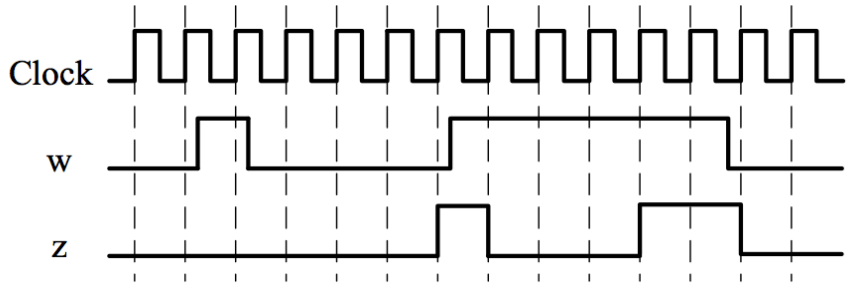


图 8-1: FSM 的时序图

这个状态机的状态图如图 8-2所示。

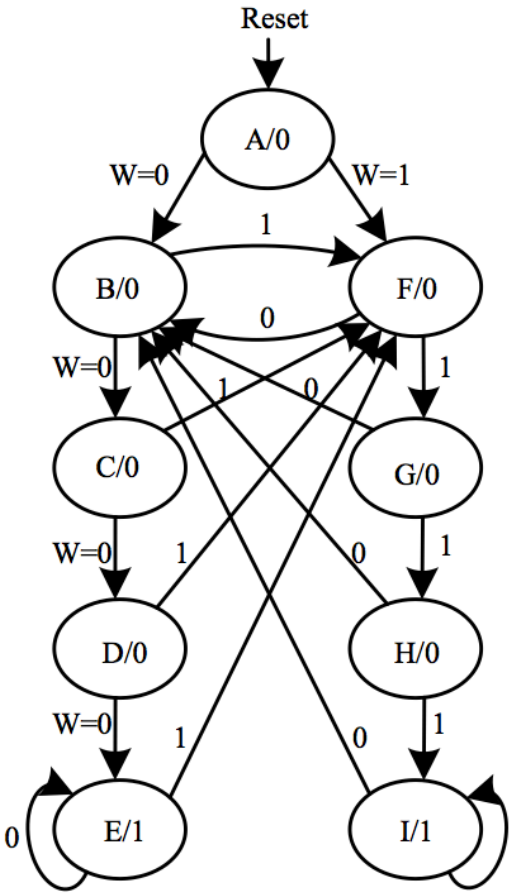


图 8-2: FSM 的时序图

用 Verilog 代码实现如图 8-2所示的状态机。此状态机有 9 个状态，至少需要 4 个状态寄存器按二进制编码形式来寄存这些状态值，如表 8-1所示。

表 8-1: FSM 的二进制编码

状态	y3	y2	y1	y0
A	0	0	0	0
B	0	0	0	1
C	0	0	1	0
D	0	0	1	1
E	0	1	0	0
F	0	1	0	1
G	0	1	1	0
H	0	1	1	1
I	1	0	0	0

建立一个 Verilog 文件，用 SW0 作为 FSM 低电平有效同步复位端，用 SW1 作为输入 w，用 KEY0 作为手动的时钟输入，用 LEDR0 作为输出 z，用 LEDR4-LEDR7 显示 4 个触发器的状态，完成该状态机的具体设计。

表 8-2: 区别两种输入状态的状态机

```
module FSM_bin
  (
    input   clk, in, reset,
    output reg out
  );

  reg [3:0] state;
  // Declare states
  parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3,
            S4 = 4, S5 = 5, S6 = 6, S7 = 7, S8 = 8;

  // Output depends only on the state
  always @ (state) begin
     case (state)
        S0:      out = 1'b0;
        S1:      out = 1'b0;
        S2:      out = 1'b0;
        S3:      out = 1'b0;
        S4:      out = 1'b1;
        S5:      out = 1'b0;
        S6:      out = 1'b0;
        S7:      out = 1'b0;
        S8:      out = 1'b1;
        default: out = 1'bx;
      endcase
  end

  // Determine the next state
  always @ (posedge clk) begin
    if (reset) state <= S0;
    else
      case (state)
         S0: if (in) state <= S5; else  state <= S1;
         S1: if (in) state <= S5; else  state <= S2;
         S2: if (in) state <= S5; else  state <= S3;
         S3: if (in) state <= S5; else  state <= S4;
         S4: if (in) state <= S5; else  state <= S4;
         S5: if (in) state <= S6; else  state <= S1;
         S6: if (in) state <= S7; else  state <= S1;
         S7: if (in) state <= S8; else  state <= S1;
         S8: if (in) state <= S8; else  state <= S1;
       endcase
   end

endmodule
```

编译工程，编写测试代码，对此状态机进行仿真。图 8-3是此状态机的功能仿真结果。

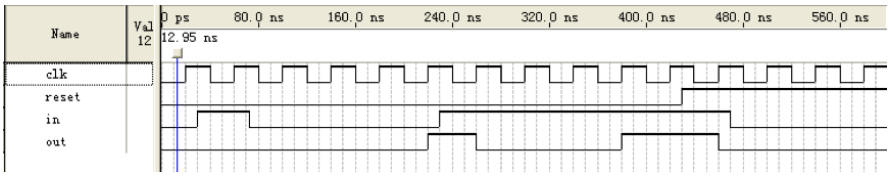


图 8-3: FSM 的功能仿真图

请自行对电路进行引脚约束，并下载到开发板上，验证其功能。

上述为一个摩尔型的状态机设计实例，请查阅相关资料，研究米里型状态的设计与此有何不同?

8.1.3 状态机的编码方式

上一节例子中的状态机的状态寄存器采用顺序二进制编码 binary 方式，即将状态机的状态依次编码为顺序的二进制数，用顺序二进制数编码可使状态向量的位数最少。如本例中只需要 4 位二进制数来编码。节省了保存状态向量的逻辑资源。但是在输出时要对状态向量进行解码以产生输出（某个状态有特定的输出，因此输出时要对此状态进行解码，满足状态编号时才可输出特定值），这个解码过程往往需要许多组合逻辑。

另外，当芯片受到辐射或者其他干扰时，可能会造成状态机跳转失常，甚至跳转到无效的编码状态而出现死机。如：状态机因异常跳转到某状态，而此状态需要等待输入，并作出应答，此时因为状态运转不正常，不会出现输入，状态机就会进入死等状态。

One-hot 编码也是状态机设计中常用的编码，在 one-hot 编码中，对于任何给定的状态，其状态向量中只有 1 位是 “1”，其他所有位的状态都为 “0”，n 个状态就需要 n 位的状态向量，所以 one-hot 编码最长。one-hot 编码对于状态的判断非常方便，如果某位为 “1” 就是某状态，“0” 则不是此状态。因此判断状态输出时非常简单，只要一、两个简单的 “与门” 或者 “或门” 即可。

One-hot 编码的状态机从一个状态到另一个状态的状态跳转速度非常快，而顺序二进制编码从一个状态跳转到另外一个状态需要较多次跳转，并且随着状态的增加，速度急剧下降。在芯片受到干扰时，one-hot 编码一般只能跳转到无效状态（如果跳到另一有效状态必须是当前为 “1” 的为变为 “0”，同时另外

一位变成由 “0” 变为 “1”，这种可能性很小），因此 one-hot 编码的状态机稳定性高。

格雷码 gray-code 也是状态机设计中常用一种编码方式，它的优点是 gray-code 状态机在发生状态跳转时，状态向量只有 1 位发生变化。

一般而言，顺序二进制编码和 gray-code 的状态机使用了最少的触发器，较多的组合逻辑，适用于提供更多的组合逻辑的 CPLD 芯片。对于具有更多触发器资源的 FPGA，用 one-hot 编码实现状态机则更加有效。所以 CPLD 多使用 gray-code，而 FPGA 多使用 one-hot 编码。对于触发器资源非常丰富的 FPGA 器件，使用 one-hot 是常用的。

在表 8-3中，状态机有 9 个状态，我们可以用 9 个触发器来实现这个状态机的电路，这 9 个状态触发器用 y8 y7 y6 y5 y4 y3 y2 y1 y0 表示。该状态机的 one-hot 编码如表 8-3所示。

表 8-3: FSM 的 One-Hot 编码

状态	y8	y7	y6	y5	y4	y3	y2	y1	y0
A	0	0	0	0	0	0	0	0	1
B	0	0	0	0	0	0	0	1	0
C	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	1	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0
H	0	1	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0

8.2 PS/2 接口控制器及键盘输入

PS/2是个人计算机串行 I/O 接口的一种标准，因其首次在 IBM PS/2（Personal System/2）机器上使用而得名，PS/2 接口可以连接 PS/2 键盘和 PS/2 鼠标。所谓串行接口是指信息是在单根信号线上按序一位一位发送的。

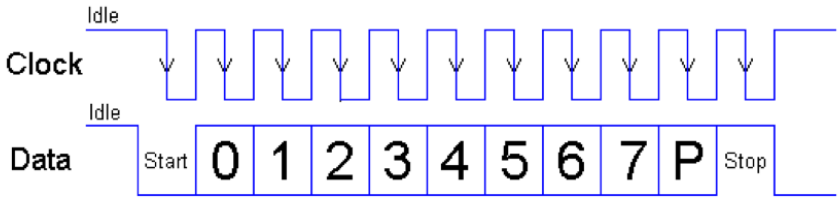


图 8-4: 键盘输出数据时序图

8.2.1 PS/2 接口的工作时序

PS/2 接口使用两根信号线，一根信号线传输时钟 PS2_CLK，另一根传输数据 PS2_DAT。时钟信号主要用于指示数据线上的比特位在什么时候是有效的。键盘和主机间可以进行数据双向传送，这里只讨论键盘向主机传送数据的情况。当 PS2_DAT 和 PS2_CLK 信号线都为高电平（空闲）时，键盘才可以给主机发送信号。如果主机将 PS2_CLK 信号置低，键盘将准备接受主机发来的命令。在我们的实验中，主机不需要发命令，只需将这两根信号线做为输入即可。

当用户按键或松开时，键盘以每帧 11 位的格式串行传送数据给主机，同时在 PS2_CLK 时钟信号上传输对应的时钟（一般为 10.0–16.7kHz）。第一位是开始位（逻辑 0），后面跟 8 位数据位（低位在前），一个奇偶校验位（奇校验）和一位停止位（逻辑 1）。每位都在时钟的**下降沿**有效，图 8-4显示了键盘传送一字节数据的时序。在下降沿有效的主要原因是下降沿正好在数据位的中间，因此可以让数据位从开始变化到接收采样时能有一段信号建立时间。

键盘通过 PS2_DAT 引脚发送的信息称为扫描码，每个扫描码可以由单个数据帧或连续多个数据帧构成。当按键被按下时送出的扫描码被称为 “通码（Make Code）”，当按键被释放时送出的扫描码称为 “断码（Break Code）”。以 “W” 键为例，“W” 键的通码是 1Dh，如果 “W” 键被按下，则 PS2_DAT 引脚将输出一帧数据，其中的 8 位数据位为 1Dh，如果 “W” 键一直没有释放，则不断输出扫描码 1Dh 1Dh . . . 1Dh，直到有其他键按下或者 “W” 键被放开。某按键的断码是 F0h 加此按键的通码，如释放 “W” 键时输出的断码为 F0h 1Dh，分两帧传输。

多个键被同时按下时，将逐个输出扫描码，如：先按左 “Shift” 键（扫描码为 12h）、再按 “W” 键、放开 “W” 键、再放开左 “Shift” 键，则此过程送出的全部扫描码为：12h 1Dh F0h 1Dh F0h 12h。

8.2.2 键盘扫描码

每个键都有唯一的通码和断码。键盘所有键的扫描码组成的集合称为扫描码集。共有三套标准的扫描码集，所有现代的键盘默认使用第二套扫描码。图 8-5显示了键盘各键的扫描码（以十六进制表示），如 Caps 键的扫描码是 58h。由图 8-5可以看出，键盘上各按键的扫描码是随机排列的，如果想迅速的将键盘扫描码转换为 ASCII 码，一个最简单的方法就是利用查找表 (LookUp Table, LUT)，扫描码到 ASCII 码的转换表格请读者自己生成。

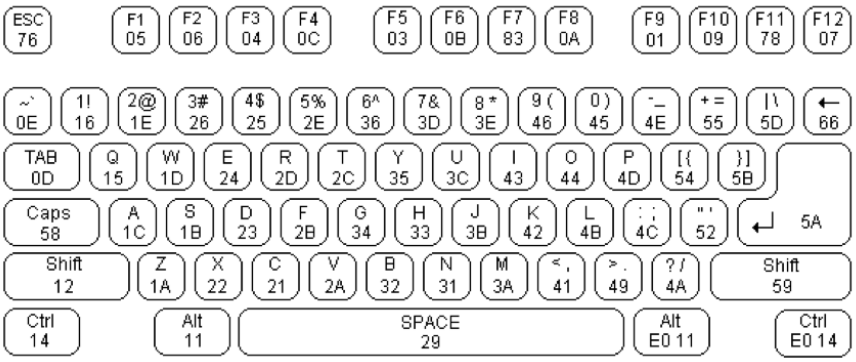


图 8-5: 键盘扫描码

图 8-6是扩展键盘和数字键盘的扫描码。

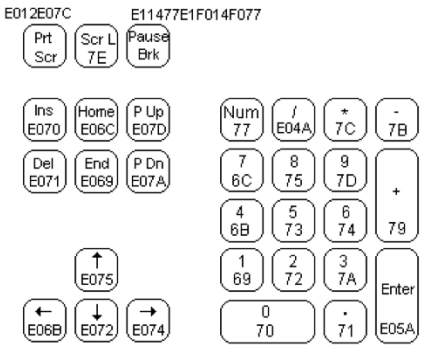
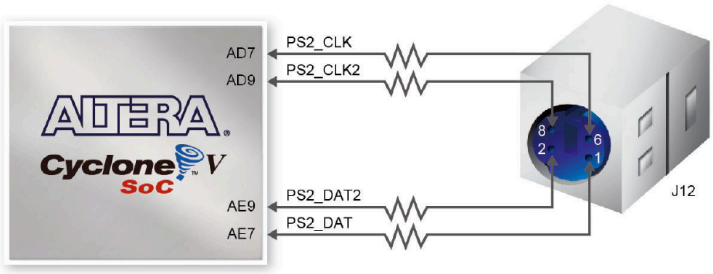


图 8-6: 扩展键盘和数字键盘的扫描码

8.2.3 PS/2 接口与 FPGA 的连接

图 8-7(a)描述了 DE10-Standard 开发板上的 PS/2 接口。该接口可以通过如图 8-7(b)的 Y 型转接口连接两个设备。当只连接一个设备时，信号连接至 PS2_DAT 和 PS2_CLK。



(a) PS/2 连线

(b) Y 型转接口

图 8-7: DE10-Standard 开发板上的 PS/2 接口

图 8-8为 DE10-Standard 上的 PS/2 接口引脚列表。

Signal Name	FPGA Pin No.	Description	I/O Standard
PS2_CLK	PIN_AB25	PS/2 Clock	3.3V
PS2_DAT	PIN_AA25	PS/2 Data	3.3V
PS2_CLK2	PIN_AC25	PS/2 Clock (reserved for second PS/2 device)	3.3V
PS2_DAT2	PIN_AB26	PS/2 Data (reserved for second PS/2 device)	3.3V

图 8-8: DE10-Standard 开发板 PS/2 引脚

8.2.4 PS/2 键盘控制器的设计

以下为接收键盘数据的 Verilog HDL 代码，此代码只负责接收键盘送来的数据，如何识别出按下的到底是什么按键由其他模块来处理。如何显示出这些数据或键符也请读者自行设计。

表 8-4: 键盘控制器

```
module ps2_keyboard(clk,clrn,ps2_clk,ps2_data,data,
                    ready,nextdata_n,overflow);
    input clk,clrn,ps2_clk,ps2_data;
    input nextdata_n;
    output [7:0] data;
    output reg ready;
    output reg overflow;     // fifo overflow
    // internal signal, for test
    reg [9:0] buffer;        // ps2_data bits
    reg [7:0] fifo[7:0];     // data fifo
        reg [2:0] w_ptr,r_ptr;   // fifo write and read pointers
    reg [3:0] count;  // count ps2_data bits
    // detect falling edge of ps2_clk
    reg [2:0] ps2_clk_sync;

    always @(posedge clk) begin
        ps2_clk_sync <=  {ps2_clk_sync[1:0],ps2_clk};
    end

    wire sampling = ps2_clk_sync[2] & ~ps2_clk_sync[1];

    always @(posedge clk) begin
        if (clrn == 0) begin // reset
            count <= 0; w_ptr <= 0; r_ptr <= 0; overflow <= 0; ready<= 0;
        end
        else begin
            if ( ready ) begin // read to output next data
                 if(nextdata_n == 1'b0) //read next data
                 begin
                    r_ptr <= r_ptr + 3'b1;
                    if(w_ptr==(r_ptr+1'b1)) //empty
                         ready <= 1'b0;
                 end
            end
            if (sampling) begin
               if (count == 4'd10) begin
                 if ((buffer[0] == 0) &&  // start bit
                    (ps2_data)       &&  // stop bit
                    (^buffer[9:1])) begin      // odd  parity
                    fifo[w_ptr] <= buffer[8:1];  // kbd scan code
                    w_ptr <= w_ptr+3'b1;
                    ready <= 1'b1;
                    overflow <= overflow | (r_ptr == (w_ptr + 3'b1));
                 end
                 count <= 0;     // for next
               end else begin
                 buffer[count] <= ps2_data;  // store ps2_data
                 count <= count + 3'b1;
               end
             end
        end
    end
    assign data = fifo[r_ptr]; //always set output data

endmodule
```

代码首先通过 ps2_clk_sync 记录 PS2 时钟信号的历史信息，并检测时钟的下降沿，当发现下降沿时将 sampling 置一。然后开始逐位接收数据并放入缓冲区 fifo 队列，收集完 11 个 bit 后将缓冲区转移至数据队列 fifo。

键盘控制器模块设置了一个 8 字节的 fifo 队列，以防止键盘数据发送过快，处理模块来不及取走数据而丢失。这类 fifo 队列在数字系统中很常见，它主要用于在两个处理速度不同的模块之间传递数据。上游模块负责在队列中添加数据，下游模块负责取出数据进行处理。fifo 队列的缓冲作用可以在下游处理模块来不及处理数据时临时存放数据。fifo 是一个先进先出的队列，配有写指针和读指针。当队列不空时，送出 ready 信号，表示此时有数据要处理；当队列溢出时，送出 overflow 信号。按键处理系统调用该模块时，需要在键盘控制器 ready 信号为 1 的情况下读取键盘数据，确认读取完毕后将 nextdata_n 置零**一个周期**。这时，键盘控制器模块收到确认读取完毕的信号，将读指针前移，准备提供下一数据。请读者自行考虑处理模块与本模块的配合时序，避免漏键或者重复读取。当然，也可自行设计两个模块交互的时序。

图 8-9是这段代码的仿真波形，显示的是接收左 “Shift” 键和 “W” 键的扫描码 “12h” 和 “1Dh” 的情形。请注意，以接收 “12h” 为例，PS/2 接口数据传送顺序为：起始位（1’b0）+ 八位数据位（由低到高）+ 奇校验位 + 停止位 (1’b1)，那么传送 “12h” 时从 PS2_DAT 端送出的数据顺序应该为 “0010 0100 011”。

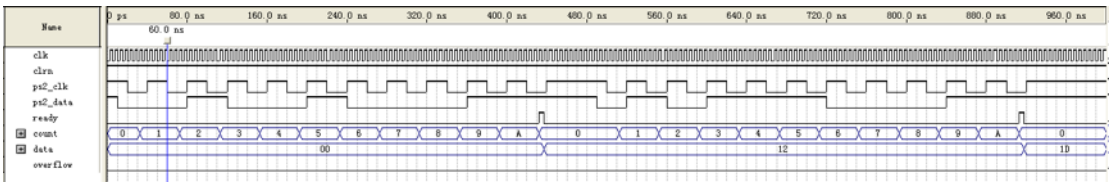


图 8-9: 键盘仿真波形

8.3 FPGA 调试指导

本次实验涉及到 FPGA 连接键盘，相比 switch，led 等，键盘的接口逻辑要复杂很多，加上需要状态机处理输入数据，不可避免需要多个模块。工程复杂后，选择合适的调试工具会很重要。如果使用开发板逐步调试，这将是一件费时费力的事情，因为生成 bitstream 文件是个比较漫长的过程。这种情况下，我们一般会选择先做仿真，验证代码逻辑部分是否正确。

前面的实验中，输入的器件一般是 button 和 switch，他们很容易使用 input array 仿真模拟。本次实验需要自己编写仿真模型，模拟前文所述的键盘接口时序。

表 8-5: 键盘仿真模型

```
`timescale 1ns / 1ps
module ps2_keyboard_model(
    output reg ps2_clk,
    output reg ps2_data
    );
parameter [31:0] kbd_clk_period = 60;
initial ps2_clk = 1'b1;

task kbd_sendcode;
    input [7:0] code; // key to be sent
    integer i;

    reg[10:0] send_buffer;
    begin
        send_buffer[0]   = 1'b0;  // start bit
        send_buffer[8:1] = code;  // code
        send_buffer[9]   = ~(^code); // odd parity bit
        send_buffer[10]  = 1'b1;  // stop bit
        i = 0;
        while( i < 11) begin
            // set kbd_data
            ps2_data = send_buffer[i];
            #(kbd_clk_period/2) ps2_clk = 1'b0;
            #(kbd_clk_period/2) ps2_clk = 1'b1;
            i = i + 1;
        end
    end
endtask

endmodule
```

表 8-5中，我们创建了 ps2_keyboard_model 模块，这个模块的输出对应键盘的两个接口信号，模块中主要是 kbd_sendcode task。

Verilog 语言中具有类似 C 语言函数的结构有 task 和 function，他们可以增加代码可读性和重复使用性。Function 用来描述组合逻辑，只能有一个返回值，function 的内部不能包含时序控制。Task 类似 procedure，执行一段 verilog 代码，task 中可以有任意数量的输入和输出，task 也可以包含时序控制。

kbd_sendcode task 用来控制键盘接口发送一个键盘码，通码或断码，只需要将 8bit 码输入，task 内部会添加 start bit，odd parity bit 和 stop bit。注意：

表 8-6: 键盘测试代码

```
`timescale 1ns / 1ps
module keyboard_sim;

/* parameter */
parameter [31:0] clock_period = 10;

/* ps2_keyboard interface signals */
reg clk,clrn;
wire [7:0] data;
wire ready,overflow;
wire kbd_clk, kbd_data;
reg nextdata_n;

ps2_keyboard_model model(
    .ps2_clk(kbd_clk),
    .ps2_data(kbd_data)
);

ps2_keyboard inst(
    .clk(clk),
    .clrn(clrn),
    .ps2_clk(kbd_clk),
    .ps2_data(kbd_data),
    .data(data),
    .ready(ready),
    .nextdata_n(nextdata_n),
    .overflow(overflow)
);

initial begin /* clock driver */
    clk = 0;
    forever
        #(clock_period/2) clk = ~clk;
end

initial begin
    clrn = 1'b0;  #20;
    clrn = 1'b1;  #20;
    model.kbd_sendcode(8'h1C); // press 'A'
    #20 nextdata_n =1'b0; #20 nextdata_n =1'b1;//read data
    model.kbd_sendcode(8'hF0); // break code
    #20 nextdata_n =1'b0; #20 nextdata_n =1'b1; //read data
    model.kbd_sendcode(8'h1C); // release 'A'
    #20 nextdata_n =1'b0; #20 nextdata_n =1'b1; //read data
    model.kbd_sendcode(8'h1B); // press 'S'
    #20 model.kbd_sendcode(8'h1B); // keep pressing 'S'
    #20 model.kbd_sendcode(8'h1B); // keep pressing 'S'
    model.kbd_sendcode(8'hF0); // break code
    model.kbd_sendcode(8'h1B); // release 'S'
    #20;
    $stop;
end

endmodule
```

kbd_clk_period 设置为 60ns，实际的键盘时钟没有这么快，这里是为了加速仿真。

keyboard_sim 中，分别将 ps2_keyboard_model 和 ps2_keyboard 实例化，并连接起来。在 initial 部分，可以直接调用 model.kbd_sendcode 发送特定的扫描码，请修改这部分代码，模拟实验需要的键盘按键序列。模拟代码中对读取信号采用直接设置的方式，也可以根据自己实现上层模块控制读取信号。

在实际物理键盘测试时，建议先将键盘控制模块的 ready、sampling 或 overflow 等重要信号引至顶层模块用 LED 显示，确保键盘基本通信正常。然后如果需要测试键码的准确性，可将收到的每个键码用 2 个七段数码管显示出来。开发板上的 6 个七段数码管可以显示三位键码，如果每次将前面收到的键码左移，就可以看到历史记录中最新收到的三个键码。在这种情况下认真反复测试，确保没有丢键码，重复键码的情况。例如按下并放开 “A” 键一次，七段显示上应该显示 “1C F0 1C”。

8.4 实验内容

自行设计状态机，实现单个按键的 ASCII 码显示。

☞ **基本要求**

- 七段数码管低两位显示当前按键的键码，中间两位显示对应的 ASCII 码（转换可以考虑自行设计一个 ROM 并初始化）。只需完成字符和数字键的输入，不需要实现组合键和小键盘。
- 当按键松开时，七段数码管的低四位全灭。
- 七段数码管的高两位显示按键的总次数。按住不放只算一次按键。只考虑顺序按下和放开的情况，不考虑同时按多个键的情况。

☞ **高级要求（选做）**

- 支持 Shift，CTRL 等组合键，在 LED 上显示组合键是否按下的状态指示
- 支持 Shift 键与字母/数字键同时按下，相互不冲突
- 支持输入大写字符，显示对应的 ASCII 码