

**课 程 实 验 报 告**

**课程名称： 硬件描述语言与数字系统设计实验**

**专业班级： 信安21XX**

**学 号： U2021XXXX**

**姓 名： MTX**

**指导教师： 鲁赵骏老师**

**报告日期： 2023年6月18日**

**网络空间安全学院**

**目 录**

[实验1 4-bit 全加器 1](#_Toc139144878)

[1.1实验目的与内容 1](#_Toc139144879)

[1.2程序设计与实现 1](#_Toc139144880)

[1.2.1程序算法设计 1](#_Toc139144881)

[1.2.2算法实现 1](#_Toc139144882)

[1.3实验结果与分析 2](#_Toc139144883)

[1.3.1程序测试 2](#_Toc139144884)

[1.3.2结果分析 2](#_Toc139144885)

[实验2 8-bit 跑马灯 3](#_Toc139144886)

[2.1实验目的与内容 3](#_Toc139144887)

[2.2程序设计与实现 3](#_Toc139144888)

[2.2.1程序算法设计 3](#_Toc139144889)

[2.2.2算法实现 3](#_Toc139144890)

[2.3实验结果与分析 4](#_Toc139144891)

[2.3.1程序测试 4](#_Toc139144892)

[2.3.2结果分析 4](#_Toc139144893)

[实验3 状态机实现按键防抖 5](#_Toc139144894)

[3.1实验目的与内容 5](#_Toc139144895)

[3.2程序设计与实现 5](#_Toc139144896)

[3.2.1程序算法设计 5](#_Toc139144897)

[3.2.2算法实现 6](#_Toc139144898)

[3.3实验结果与分析 8](#_Toc139144899)

[3.3.1程序测试 8](#_Toc139144900)

[3.3.2结果分析 8](#_Toc139144901)

[实验4 自动售货机 9](#_Toc139144902)

[4.1实验目的与内容 9](#_Toc139144903)

[4.2程序设计与实现 9](#_Toc139144904)

[4.2.1程序算法设计 9](#_Toc139144905)

[4.2.2算法实现 10](#_Toc139144906)

[4.3实验结果与分析 12](#_Toc139144907)

[4.3.1程序测试 12](#_Toc139144908)

[4.3.2结果分析 12](#_Toc139144909)

[实验5 串口通信（附加题） 13](#_Toc139144910)

[5.1实验目的与内容 13](#_Toc139144911)

[5.2程序设计与实现 13](#_Toc139144912)

[5.2.1程序算法设计 13](#_Toc139144913)

[5.2.2算法实现 14](#_Toc139144914)

[5.3实验结果与分析 15](#_Toc139144915)

[5.3.1程序测试 15](#_Toc139144916)

[5.3.2结果分析 16](#_Toc139144917)

[实验总结&心得体会 17](#_Toc139144918)

[6.1实验总结 17](#_Toc139144919)

[6.2心得体会 17](#_Toc139144920)

[参考文献 19](#_Toc139144921)

# 实验1 4-bit 全加器

## 1.1实验目的与内容

在FPGA上，分别用组合逻辑和时序逻辑实现 4-bit 全加器。

参考课堂 ppt 和开源库代码，先实现 1-bit 全加器，然后实现 4-bit 全加器。其中SW[3:0]为 In\_a，SW[15:12]为 In\_b，结果为 LED[4:0]。

## 1.2程序设计与实现

### 1.2.1程序算法设计

对于1bit全加器来说，一共有三种写法，如代码1-1所示：

代码1-1 1bit全加器的三种代码实现方式

1. // 1st
2. assign #TCQ {Out\_carry\_0, Out\_sum\_0} = In\_a + In\_b + In\_carry;
3. // 2nd
4. assign #TCQ Out\_sum\_1 = In\_a ^ In\_b ^ In\_carry;
5. assign #TCQ Out\_carry\_1 = (In\_a & In\_b) | (In\_carry & (In\_a ^ In\_b));
6. // 3rd
7. always @ (In\_a **or** In\_b **or** In\_carry) **begin**
8. s1          = #1 In\_a ^ In\_b;
9. out\_sum\_2   = #1 s1 ^ In\_carry;
10. c1          = #1 s1 & In\_carry;
11. c2          = #1 In\_a & In\_b;
12. out\_carry\_2 = #1 c1 | c2;
13. **end**

具体来说，第一种是利用Verilog自带的语法进行实现的，第二种是通过组合逻辑进行分析实现的，而第三种将检测输入变量的变化后逐一计算各输出的结果，具有时序逻辑的思想。

借助1bit全加器实现中的第一和第三种实现方式，来分别用组合逻辑和时序逻辑来实现4bit全加器。

### 1.2.2算法实现

根据1.2.1中的设计思想，首先采用Verilog自带的语法来实现组合逻辑的4bit全加器。接着，参考代码1-1中的第三种实现方式，在时钟上升沿时进行计算操作，使用串行进位计算的方法，利用Verilog的循环结构实现从低至高依次通过1bit全加器来计算结果和进位，最后assign至输出值，实现时序逻辑的4bit全加器。整体的代码如代码1-2所示。

代码1-2 分别用组合逻辑和时序逻辑实现4bit全加器

1. // 组合逻辑
2. assign #TCQ {Out\_carry, Out\_sum} = In\_a + In\_b + In\_carry;
3. // 时序逻辑
4. assign #TCQ Out\_sum1 = out\_sum;
5. assign #TCQ Out\_carry1 = out\_carry;
6. integer i;
7. always @ (posedge Clk) **begin**
8. out\_carry = #TCQ In\_carry;
9. **for**(i = 0;i < 4;i = i + 1) **begin**
10. out\_sum[i] = #TCQ In\_a[i] ^ In\_b[i] ^ out\_carry;
11. out\_carry = #TCQ (In\_a[i]&In\_b[i])|(out\_carry & (In\_a[i]^In\_b[i]));
12. **end**
13. **end**

## 1.3实验结果与分析

### 1.3.1程序测试

完成1.2.2中的代码编写后做仿真测试，结果如图1.1所示。



图1.1 程序测试图

### 1.3.2结果分析

如图1.1所示，组合逻辑和时序逻辑的运算结果相同且正确，功能通过了测试。综上所述，经过验证后我们的程序是符合实验要求，完全达到预期的实验目的的。

# 实验2 8-bit 跑马灯

## 2.1实验目的与内容

在FPGA上，实现一个自动化的8bit跑马灯。

参考课堂 ppt 和开源库代码，LED[7:0]作为跑马灯，每隔 1s 改变状态，实现从左到右然后从右到左依次点亮一个 led 灯。

## 2.2程序设计与实现

### 2.2.1程序算法设计

在此实验中，将分为三个模块。第一个模块将处理时间相关的信息，在计时至1秒时将变量置0，以此来判断是否达到改变跑马灯状态的条件。

第二个模块将决定跑马灯的点亮顺序，是从左往右还是从右往左。具体来说，在点亮最左边LED灯时，需要让其从左往右点亮；而在点亮最右边的LED灯时，需要让其从右往左点亮。当然，这一过程可以在初始化后，判断是否点亮最左边或者最右边LED灯后进行取反，来实现点亮顺序的判定。

最后一个模块将要实现LED的顺序点亮，具体来说，就是在第一个模块产生改变信息后，安装第二个模块确定的顺序，往左或往右对LED灯的值进行循环移位即可。

### 2.2.2算法实现

根据2.2.1中的设计思想，分别实现三个模块如下：

在第一个模块中，在计时至1秒后将置0，否则加1，如代码2-1所示：

代码2-1 实现跑马灯的计时模块

1. always @(posedge Clk) **begin**
2. **if**(Rst == 1'b1)
3. t\_cnt <= #TCQ 0;
4. **else**
5. **if**(t\_cnt == T\_INR\_CNT\_SET)
6. t\_cnt <= #TCQ 0;
7. **else**
8. t\_cnt <= #TCQ t\_cnt + 1'b1;
9. **end**

而在第二个模块中，将分别在点亮最右边和最左边LED灯时改变输出顺序，其中为1时将从左往右点亮，为0时将从右往左点亮，如代码2-2所示：

代码2-2 实现决定跑马灯输出状态的模块

1. always @(posedge Clk ) **begin**
2. **if**(Rst == 1'b1)
3. flag <= #TCQ 1'b1;
4. **else** **begin**
5. **if**(out\_led == 8'b00000001)
6. flag <= #TCQ 1'b0;
7. **else** **if**(out\_led == 8'b10000000)
8. flag <= #TCQ 1'b1;
9. **end**
10. **end**

最后将根据前两个模块的条件，实现LED灯的顺序点亮，如代码2-3所示：

代码2-3 实现按序点亮LED灯的模块

1. always @(posedge Clk) **begin**
2. **if**(Rst == 1'b1)
3. out\_led <= #TCQ 8'b10000000;
4. **else** **if**(t\_cnt == 0) **begin**
5. **if**(flag == 1)
6. out\_led <= #TCQ {out\_led[0],out\_led[7:1]};
7. **else**
8. out\_led <= #TCQ {out\_led[6:0],out\_led[7]};
9. **end**
10. **end**

## 2.3实验结果与分析

### 2.3.1程序测试

完成代码编写后，编译运行程序，在经过Rst信号进行复位后，程序开始自动运行，仿真结果如图2.1所示：

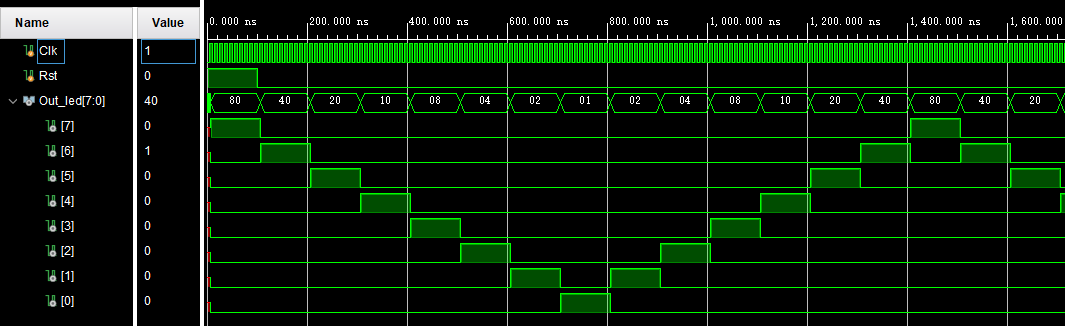


图2.1 程序测试图

### 2.3.2结果分析

根据图2.1的结果，经过验证后我们的程序是符合实验要求，完全达到预期的实验目的的。

# 实验3 状态机实现按键防抖

## 3.1实验目的与内容

在FPGA上，使用 Mealy 型三段状态机的编程模板实现按键防抖，任选一个按键与一个led 灯，按一次按键，led 点亮，再按一次熄灭。

## 3.2程序设计与实现

### 3.2.1程序算法设计

首先根据图3.1的按键防抖示意图，我们可以分别设计出五种不同的状态，如图3.2所示。其中STA\_IDLE代表初始状态，STA\_1是按下抖动阶段，STA\_2是检测到按下后输出信号的暂态， STA\_3是按下稳定阶段，STA\_4是松开抖动阶段。

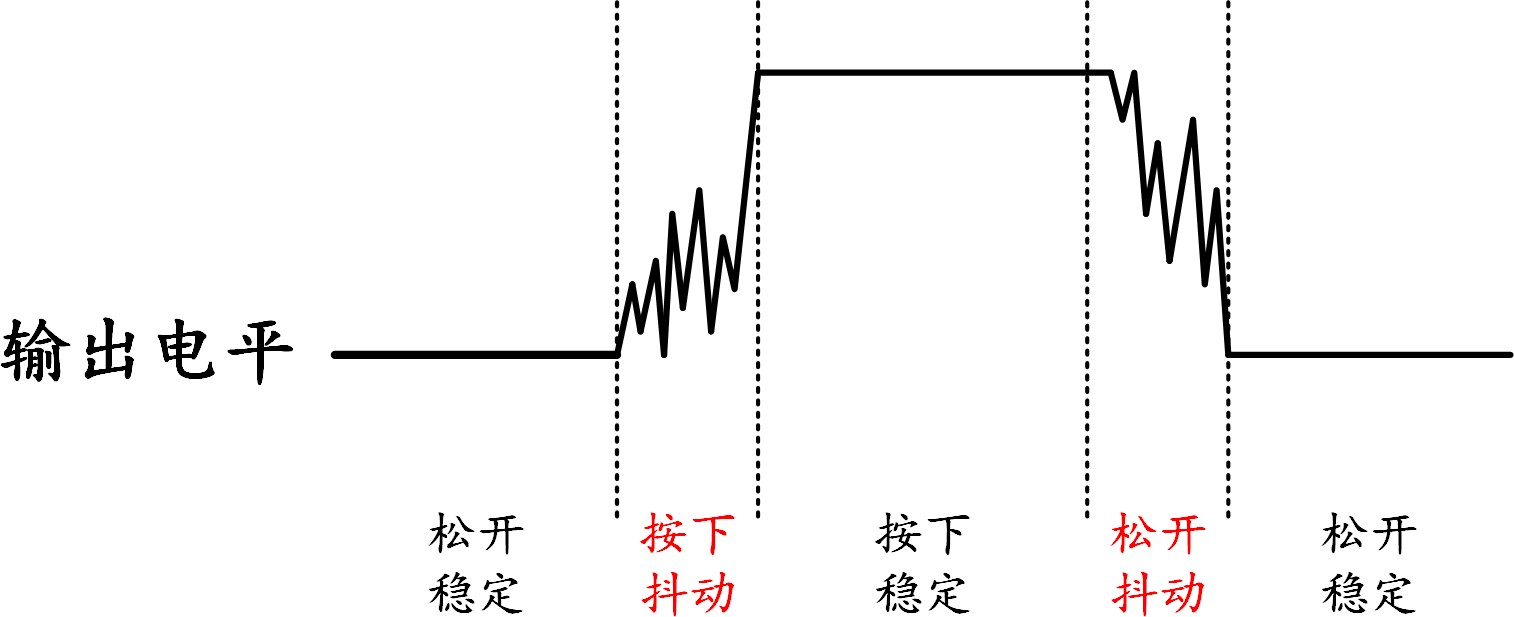


图3.1 按键防抖示意图

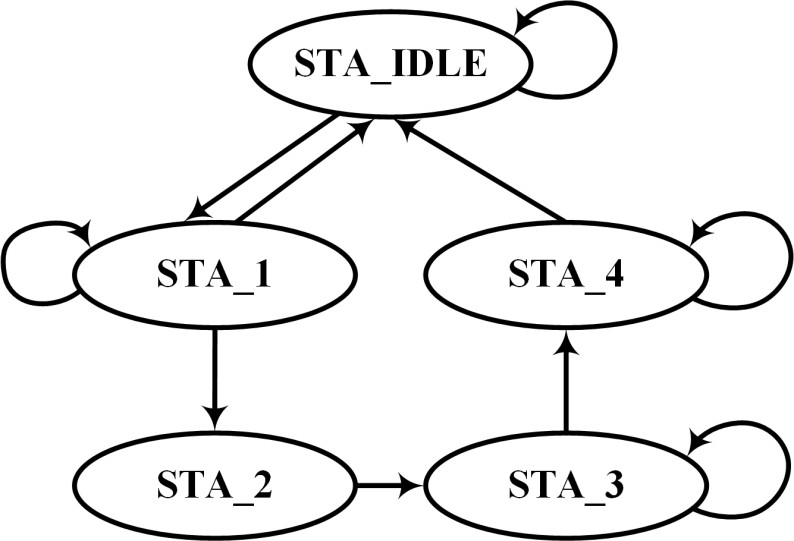


图3.2 按键防抖状态设计图

当STA\_IDLE阶段检测到按键按下时，将进入STA\_1状态，否则将维持不变。随后在STA\_1状态中若检测到按键松开则说明是一次抖动，回到STA\_IDLE阶段，而若没有松开，则说明是一次按下，进入STA\_2阶段。而STA\_2仅仅是为了返回一个状态信息，将标志位置高一个周期，随后立即进入STA\_3。此时若检测到按键松开则进入STA\_4阶段，否则将保持STA\_3阶段。最后在STA\_4阶段若检测到按键保持松开则回到初始STA\_IDLE阶段，否则将保持不变。

此外，为了实现按一次按键点亮，再按一次熄灭的效果，在初始将灯熄灭后，检测到STA\_2的标志信号即可将灯状态置反，实现了点亮后熄灭的效果。

### 3.2.2算法实现

根据3.2.1中的设计思想，将使用三段式状态机实现上述功能，而计时模块与实验2中的代码2-1基本一致，在此不再重复给出。

首先给出状态转换模块，如代码3-1所示，其将不断地将现态转至对应的次态，或者重置为STA\_IDLE状态。

代码3-1 实现按键防抖的状态转换模块

1. //State Tranfer
2. always @(posedge Clk) **begin**
3. **if**(Rst == 1'b1)
4. curr\_state <= #TCQ STA\_IDLE;
5. **else**
6. curr\_state <= #TCQ next\_state;
7. **end**

其次给出计算次态模块，如代码3-2所示。

代码3-2 实现按键防抖的计算次态模块

1. //Calculate the next state
2. always @( \* ) **begin**
3. **case**(curr\_state)
4. STA\_IDLE            : **begin**
5. **if**((en\_10ms == 1'b1) && (In\_key == 1'b1))
6. next\_state = STA\_1;
7. **else**
8. next\_state = STA\_IDLE;
9. **end**
10. STA\_1               : **begin**
11. **if**((en\_10ms == 1'b1) && (In\_key == 1'b1))
12. next\_state = STA\_2;
13. **else** **if**((en\_10ms == 1'b0) && (In\_key == 1'b1))
14. next\_state = STA\_1;
15. **else**
16. next\_state = STA\_IDLE;
17. **end**
18. STA\_2               : **begin**
19. next\_state = STA\_3;
20. **end**
21. STA\_3               : **begin**
22. **if**((en\_10ms == 1'b1) && (In\_key == 1'b0))
23. next\_state = STA\_4;
24. **else**
25. next\_state = STA\_3;
26. **end**
27. STA\_4               : **begin**
28. **if**((en\_10ms == 1'b1) && (In\_key == 1'b0))
29. next\_state = STA\_IDLE;
30. **else**
31. next\_state = STA\_4;
32. **end**
33. default             : **begin**
34. next\_state = STA\_IDLE;
35. **end**
36. **endcase**
37. **end**

最后给出输出模块，如代码3-3所示，其将决定着LED灯的输出情况以及按下按键的标志信息的输出，而这两个信号的改变都是在STA\_2中进行的，因此更多地关注此状态前后的信号变化即可。

代码3-3 实现按键防抖的输出模块

1. //state output
2. always @(posedge Clk) **begin**
3. **if**(Rst == 1'b1) **begin**
4. out\_key\_cap      <= #TCQ 1'b0;
5. out\_led          <= #TCQ 1'b0;
6. **end**
7. **else** **case**(curr\_state)
8. STA\_IDLE            : **begin**
9. out\_key\_cap      <= #TCQ 1'b0;
10. **end**
11. STA\_2               : **begin**
12. out\_key\_cap      <= #TCQ 1'b1;
13. out\_led          <= #TCQ ~out\_led;
14. **end**
15. STA\_3               : **begin**
16. out\_key\_cap      <= #TCQ 1'b0;
17. **end**
18. **endcase**
19. **end**

## 3.3实验结果与分析

### 3.3.1程序测试

完成代码编写后，编译运行程序，在经过Rst信号进行复位后，程序开始自动运行，仿真结果如图3.3所示：

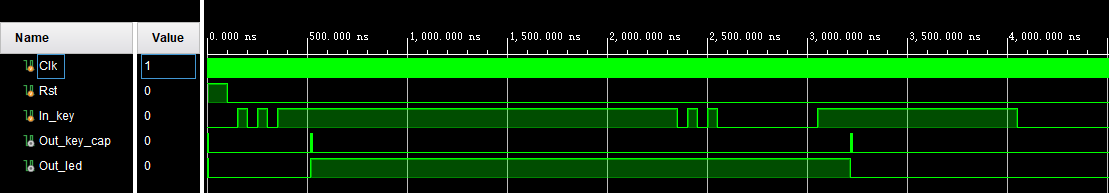


图3.3 程序测试图

### 3.3.2结果分析

根据图3.3的结果，经过验证后我们的程序是符合实验要求，完全达到预期的实验目的的。

# 实验4 自动售货机

## 4.1实验目的与内容

在FPGA上，实现一个自动售货机，有如下要求：

1. 任选两个按键，按键 A 按一下表示投入 1 元，按键 B 按一下表示投入 0.5 元；

2. 商品有 8 种，分别为 5 元、4.5 元、4 元、3.5 元、3 元、2.5 元、2 元、1.5 元；

3. 自动售货机具备选择商品功能、投币功能、找零功能、撤回功能、回位功能等基本功能。

## 4.2程序设计与实现

### 4.2.1程序算法设计

首先根据自动售货机的几种状态，以及4.1中的设计要求，我们可以分别设计出六种不同的状态，如图4.1所示。其中STA\_IDLE代表初始状态，STA\_SELECT是选择货物阶段，STA\_COIN是投币阶段， STA\_CHANGE是找零阶段，STA\_DONE是正常结束流程后的退出阶段。而在STA\_SELECT和STA\_COIN阶段中，如果超过了规定的时限，则进入STA\_OUT阶段准备退出。

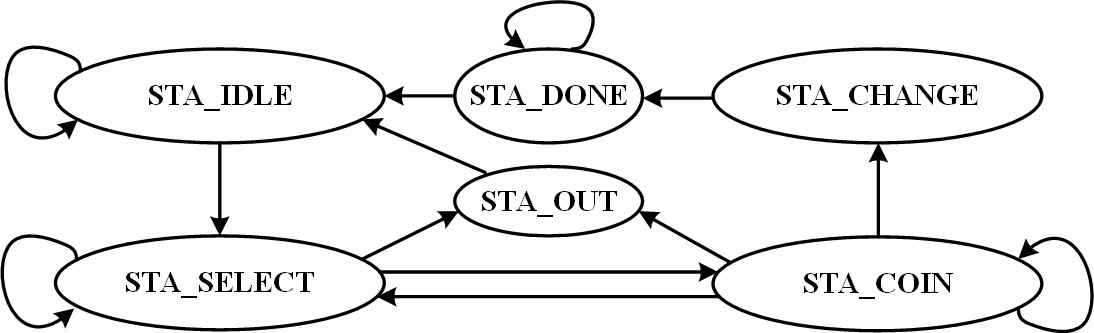


图4.1 自动售货机状态设计图

首先在STA\_IDLE阶段，如果收到了开始选择商品的使能信号，则进入STA\_SELECT阶段，否则将持续等待。而STA\_SELECT阶段中，用户将在选择好商品后输入选好商品的使能信号，则此时进入STA\_COIN阶段，否则将保持现态，直至超时进入STA\_OUT阶段。在STA\_COIN阶段中，将根据按下的按键情况来计算投币情况，如果达到要求金额则进入STA\_CHANGE阶段，同时若接收到选择商品信号置低，则重新进入STA\_SELECT阶段选择商品，否则将持续等待直到超时进入STA\_OUT阶段。STA\_CHANGE的找零阶段中将通过LED灯显示投币金额和商品价格间的差值，然后立即进入STA\_DONE阶段。在STA\_DONE中将持续显示找零情况，在标志位都置0，或者超时后，都将回到STA\_DONE阶段是正常结束流程后的退出阶段。而在STA\_SELECT和STA\_COIN阶段中，如果超过了规定的时限，则进入STA\_OUT阶段准备退出。最后在STA\_OUT阶段，程序将无条件进入STA\_OUT状态。

### 4.2.2算法实现

根据4.2.1中的设计思想，将使用三段式状态机实现上述功能，在此首先将给出自动售货机的计算次态模块，如代码4-1所示。

代码4-1 实现自动售货机的计算次态模块

1. //Calculate the next state
2. always @( \* ) **begin**
3. **case**(curr\_state)
4. STA\_IDLE            : **begin**
5. **if**(In\_enable == 1'b1)
6. next\_state = STA\_SELECT;
7. **else**
8. next\_state = STA\_IDLE;
9. **end**
10. STA\_SELECT          : **begin**
11. **if**(en\_30s == 1'b1)
12. next\_state = STA\_OUT;
13. **else** **if**(In\_select == 1'b1)
14. next\_state = STA\_COIN;
15. **else**
16. next\_state = STA\_SELECT;
17. **end**
18. STA\_COIN                : **begin**
19. **if**(en\_30s == 1'b1)
20. next\_state = STA\_OUT;
21. **else** **if**(In\_select == 1'b0)
22. next\_state = STA\_SELECT;
23. **else** **if**(out\_led >= goods[In\_goods])
24. next\_state = STA\_CHANGE;
25. **else**
26. next\_state = STA\_COIN;
27. **end**
28. STA\_CHANGE          :
29. next\_state = STA\_DONE;
30. STA\_DONE            : **begin**
31. **if**((en\_30s == 1'b1)||(|{In\_enable,In\_select,In\_goods} == 1'b0))
32. next\_state = STA\_IDLE;
33. **else**
34. next\_state = STA\_DONE;
35. **end**
36. STA\_OUT             :
37. next\_state = STA\_IDLE;
38. default             :
39. next\_state = STA\_IDLE;
40. **endcase**
41. **end**

最后给出输出模块，其中省略了一些初始化的过程，例如初始化商品信息，标志位信息，状态信息等等，如代码4-2所示。

代码4-2 实现自动售货机的输出模块

1. //state output
2. always @(posedge Clk) **begin**
3. **if**(Rst == 1'b1)
4. // 初始化代码, 省略......
5. **else** **case**(curr\_state)
6. STA\_IDLE            :
7. // 初始化代码, 省略......
8. STA\_SELECT          : **begin**
9. timer\_begin      <= #TCQ 1'b1;
10. out\_done         <= #TCQ 1'b0;
11. out\_led          <= #TCQ 4'b0000;
12. **end**
13. STA\_COIN            : **begin**
14. timer\_begin      <= #TCQ 1'b1;
15. out\_done         <= #TCQ 1'b0;
16. **if**(out\_key\_cap\_half == 1'b1)
17. out\_led         <= #TCQ out\_led + 4'b0001;
18. **else** **if**(out\_key\_cap\_one == 1'b1)
19. out\_led         <= #TCQ out\_led + 4'b0010;
20. **else**
21. out\_led         <= #TCQ out\_led;
22. **end**
23. STA\_CHANGE          :
24. out\_led          <= #TCQ out\_led - goods[In\_goods];
25. STA\_DONE            :
26. out\_done         <= #TCQ 1'b1;
27. STA\_OUT             :
28. timer\_begin      <= #TCQ 1'b0;
29. **endcase**
30. **end**

## 4.3实验结果与分析

### 4.3.1程序测试

完成代码编写后，将测试正常流程下选择3号商品（3元）时，依次投入1元、1元、0.5元、1元后的执行情况。编译运行程序，结果如图4.2所示。

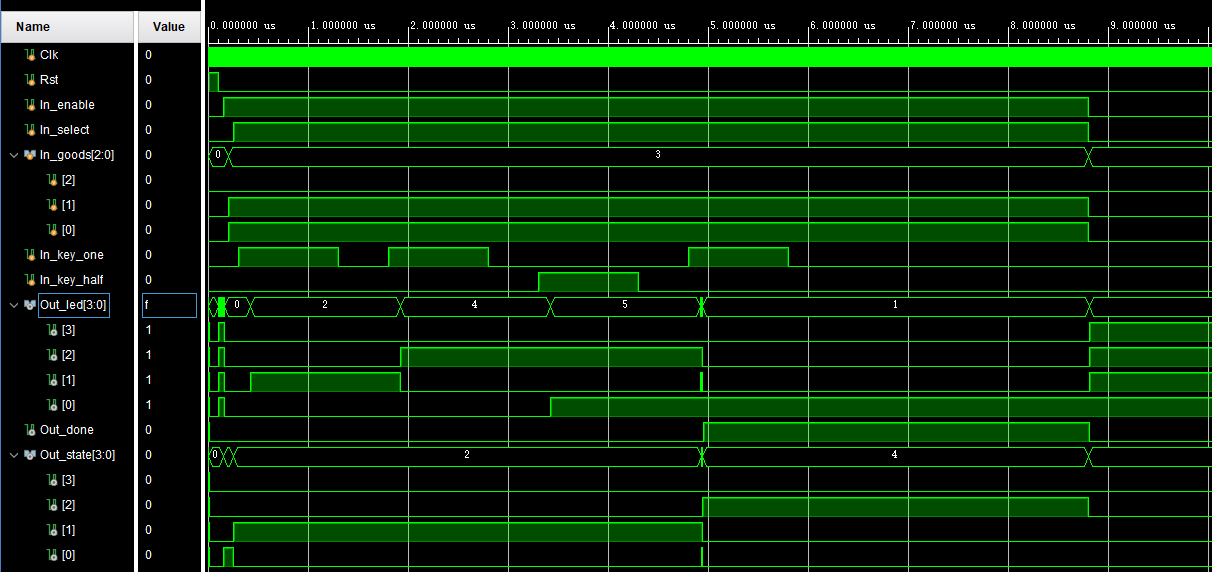


图4.2 程序测试图

### 4.3.2结果分析

根据图4.2的结果，经过验证后我们的程序是符合实验要求，完全达到预期的实验目的的。

# 实验5 串口通信（附加题）

## 5.1实验目的与内容

在FPGA上，实现与PC间的串口通信，有如下要求：

1. 掌握串口通信原理；

2. 实现基本的接收和发送模块；

3. FPGA 与 PC 端串口调试助手实现数据传输，PC 每发送 4 bytes 数据给 FPGA，FPGA 将 4 bytes 数据返回给 PC 并在串口调试助手上现实。

## 5.2程序设计与实现

### 5.2.1程序算法设计

根据接收器的功能，首先给出如图5.1所示的状态转换图，其中STA\_IDLE为初始状态，STA\_RX\_START为检测输入阶段，STA\_RX\_REC\_DATA为接收数据阶段，STA\_RX\_STOP为停止接收阶段。

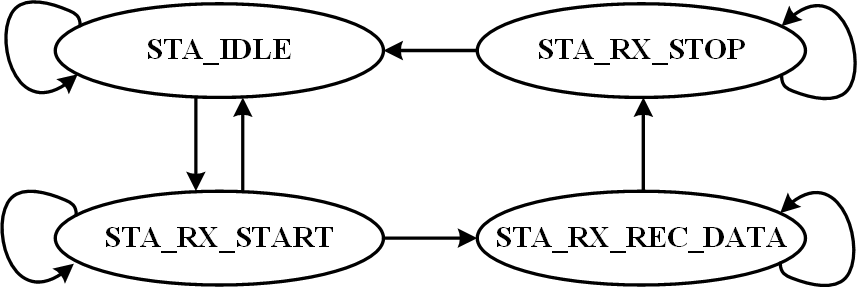


图5.1 接收器状态设计图

首先在STA\_IDLE阶段，如果检测到rx信号下降沿，说明可能有信号输入，进入STA\_RX\_START阶段。此时，若下一个bound标志不稳定为0，说明不是一次有效输入，回到初始状态；若稳定为0，则是一次有效输入，进入STA\_RX\_REC\_DATA阶段。在接收数据阶段，将经历8个bound标志，即接收8bit的数据后，进入STA\_RX\_STOP阶段，准备结束接收。在停止接受阶段，需要将标志位置高一个周期后，到达下一个bound标志时立即进入STA\_IDLE初始状态，等待下一次接收。

随后，将要实现PC端设计，分为三个状态。STA\_IDLE状态为初始状态，STA\_RECEIVING状态为接收输入数据阶段，STA\_REPLYING为发送数据阶段。整体的状态转移如图5.2所示。

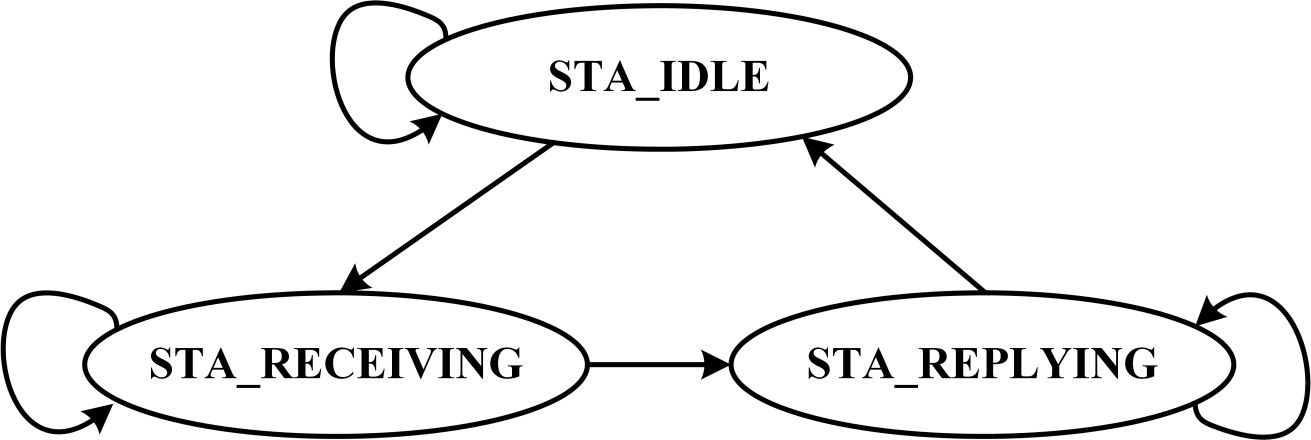


图5.2 PC端状态转移图

现开始设计PC端各状态之间的转换条件，以及输出信息。首先在STA\_IDLE阶段，如果收到了0xFF这个起始信号，则说明开始接收数据，进入STA\_RECEIVING状态。在此阶段中，将持续接收4bytes的数据后，进入到STA\_REPLYING阶段。在STA\_REPLYING阶段中，将持续发送数据，直至发送完后回到初始STA\_IDLE状态。

### 5.2.2算法实现

首先仿照实验3、4中的三段式状态机设计，根据5.2.1中对接收端和发送端的状态转换和输出信息的描述，进行程序设计，在此不再过多赘述。

在本章节中，将着重给出5.2.1中未提及的发送器模块，具体如下：

首先将实现一个类似实验2中代码2-1的计时模块，但是在代码5-1中是对baund进行计时，而不是对时钟周期进行计时，如代码5-1所示。

代码5-1 实现发送器的计时模块

1. always @(posedge Clk) **begin**
2. **if**(Rst == 1'b1) **begin**
3. baund\_cnt    <= #TCQ 16'b0;
4. baund\_en     <= #TCQ 1'b0;
5. **end**
6. **else** **if**(sending == 1'b1) **begin**
7. **if**(baund\_cnt == BAUND\_EN\_INTERVAL - 1'b1)
8. baund\_cnt    <= #TCQ 16'b0;
9. **else**
10. baund\_cnt    <= #TCQ baund\_cnt + 1'b1;
11. **if**(baund\_cnt == 16'b0)
12. baund\_en     <= #TCQ 1'b1;
13. **else**
14. baund\_en     <= #TCQ 1'b0;
15. **end**
16. **else** **begin**
17. baund\_cnt    <= #TCQ 16'b0;
18. baund\_en     <= #TCQ 1'b0;
19. **end**
20. **end**

随后，将进入到数据处理的阶段。在接收器接收到数据并使能标志位时，发送器将数据暂存至send\_buf中等待发送。随后在baund周期到来时，会逐个取出数据至out\_tx中进行发送，直到最后全部发送完毕，清空标志位。整个过程如代码5-2所示。

代码5-2 实现发送器的数据处理模块

1. always @(posedge Clk) **begin**
2. **if**(Rst == 1'b1) **begin**
3. // 初始化代码, 省略......
4. **end**
5. **else** **begin**
6. **if**(In\_data\_vld == 1'b1) **begin**
7. send\_buf      <= #TCQ {1'b1, In\_data, 1'b0};
8. sending       <= #TCQ 1'b1;
9. bit\_cnt       <= #TCQ 4'b0;
10. **end**
11. **else** **if**(baund\_en == 1'b1) **begin**
12. out\_tx        <= #TCQ send\_buf[0];
13. send\_buf[9:0] <= #TCQ {1'b1, send\_buf[9:1]};
14. bit\_cnt       <= #TCQ bit\_cnt + 1'b1;
15. **if**(bit\_cnt == 4'd10) **begin**
16. sending     <= #TCQ 1'b0;
17. send\_done   <= #TCQ ~send\_done;
18. **end**
19. **end**
20. **end**
21. **end**

最后，发送器将输出状态信息，来说明是否发送完毕。整个过程如代码5-3所示。

代码5-3 实现发送器的信号输出模块

1. always @(posedge Clk) **begin**
2. send\_done\_r     <= #TCQ send\_done;
3. out\_send\_done   <= #TCQ send\_done^send\_done\_r;
4. **end**

## 5.3实验结果与分析

### 5.3.1程序测试

完成代码编写后，将测试输入1字节的数据时，FPGA处理的情况，如图5.3所示。：

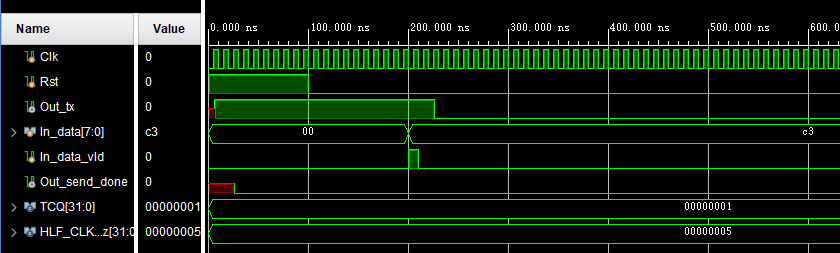


图5.3 程序测试图

### 5.3.2结果分析

根据图5.3的结果，可以发现在发送了数据后，接收使能In\_data\_vld置高，说明已经接收到信息。但是由于只收到了1字节的信息，因此发送位不变。

综上所述，经过验证后我们的程序是符合实验要求，完全达到预期的实验目的的。

# 实验总结&心得体会

## 6.1实验总结

这次实验是在学习硬件描述语言与数字系统设计这一门课之后，对相关知识的应用，通过使用Verilog语言和vivado软件，完成了基础部分和进阶部分。其中，基础部分包括4-bit全加器的实现，8-bit跑马灯的实现以及状态机实现按键防抖功能。而进阶部分，也就是有限状态机的综合实验中，完成了自动售货机和串口通信两部分。

将上课学到的一些理论知识在具体的实践任务中进一步巩固，同时也学到了相关软件的应用知识，对Verilog语言的编写和分析也得到了提升。同时将电脑和开发板相连，第一次具体的看到了程序的实现状况，并且进行调试，实现了软硬件结合的学习模式。

## 6.2心得体会

在这门计算机实验课程中，我学到了很多有关硬件描述语言和实际应用的知识，是在数字电路逻辑和计算机组成原理学习知识的进一步提升，对我个人的学习和职业发展有着重要的意义。

首先，通过实验课程，我进一步的掌握了更多的计算机硬件知识，如4-bit全加器，8-bit跑马灯的实现等，让我对计算机底层硬件的知识得到了丰富。同时通过亲自动手实践，我更好地理解了这些概念，并学会了如何配置和管理计算机系统。

其次，实验课程让我熟悉了常用的硬件描述语言verilog和开发工具软件。我学习了对程序的编写和调试，以及相关开发工具的应用使用了一些常见的编程语言和集成开发环境。

此外，通过与同学们的合作和讨论，我培养了团队合作和沟通能力。我们在实验中遇到问题时，能够相互帮助和协作解决。这种合作精神不仅提高了我们的实验效率，也增强了我们的学习体验。

最后，实验课程也提供了很好的实践机会，让我将理论知识应用到实际问题中。通过老师给我们发放的开发板，我们可以将电脑和开发板相连，将自己的代码通过硬件得到了实现，并且基于此进行进一步的调试和运行，让我们感到很有成就感。

总的来说，这门硬件描述语言与数字系统设计实验课程对我来说是一次非常宝贵的学习经历。通过实践和合作，我不仅增加了硬件系统的理解，还提高了编程和解决问题的能力。我相信这些收获将对我的未来学习和职业发展产生积极的影响。

# 参考文献

1 《密码硬件工程实践》教材及课程相关代码. URL：<https://gitee.com/lzj_cse/cryptographic-hardware-engineering>

2 密码硬件系列课程辅助视频. URL：<https://space.bilibili.com/6876595/channel/seriesdetail?sid=3136970>