

电子技术基础实验第十周实验报告

王磊 2022012972

2023 年 11 月 29 日

1 task1&task2

由于task2是task1的延续，所以将两个任务放在一起说明。

1.1 mif文件的生成

为减少ROM的使用，我通过IP核生成了一个8位宽，768位深的ROM，将其分为三部分分别存储正弦波、三角波和方波。生成mif的matlab代码如图1所示。

```
% a generator produce 3 waves
width = 8;
depth = 768;

fid = fopen('final_waveforms.mif', 'w');
fprintf(fid, 'WIDTH=%d;\n', width);
fprintf(fid, 'DEPTH=%d;\n', depth);
fprintf(fid, 'ADDRESS_RADIX=DEC;\n');
fprintf(fid, 'DATA_RADIX=DEC;\n');
fprintf(fid, 'CONTENT BEGIN\n');

for i = 0:depth-1
    You, 前天 | 1 author (You)
    if i < depth/3
        sin_data = floor((sin(2*pi*i/(depth/3)) + 1) * 0.5 * (2^width - 1));
        fprintf(fid, '%d:%d;\n', i, sin_data);
        You, 前天 | 1 author (You)
    elseif i < 2*depth/3
        triangle_data = abs(floor((2 * abs(mod(i, (2 * (depth / 3))) - (depth / 3)) / (depth / 3) - 1) * (2 ^ width - 1)));
        fprintf(fid, '%d:%d;\n', i, triangle_data);
        You, 前天 | 1 author (You)
    else
        square_data = floor(((mod(i, depth/3) < depth/6) * (2^width - 1)));
        fprintf(fid, '%d:%d;\n', i, square_data);
    end
end

fprintf(fid, 'END ;\n');
fclose(fid);
```

图 1: matlab代码

1.2 地址生成模块

地址生成模块的代码如下：

```
1  module addr_tx_en2 (  
2      input clk,  
3      input clk_origin,  
4      input rst,  
5      input [2:0] switch,  
6      output reg [9:0] addr,  
7      output reg tx_en  
8  );  
9  
10     reg [9:0] addr_temp;  
11     always @(posedge clk, posedge rst) begin  
12         if (rst) begin  
13             addr_temp <= 10'd0;  
14         end else begin  
15             if (addr_temp == 10'd255) begin  
16                 addr_temp <= 10'd0;  
17             end else begin  
18                 addr_temp <= addr_temp + 1'b1;  
19             end  
20         end  
21     end  
22  
23     always @(posedge clk, posedge rst) begin  
24         if (rst) begin  
25             addr <= 10'd0;  
26         end else begin  
27             if (switch[0]) begin  
28                 addr <= addr_temp;  
29             end else if (switch[1]) begin  
30                 addr <= addr_temp + 10'd256;  
31             end else if (switch[2]) begin  
32                 addr <= addr_temp + 10'd512;  
33             end else begin  
34                 addr <= 10'd0;  
35             end  
36         end  
end
```

```

37     end
38
39     reg pulse1, pulse2, pulse3;
40     wire clk_posedge;
41     always @(posedge clk_origin, posedge rst) begin
42         if (rst) begin
43             pulse1 <= 1'b0;
44             pulse2 <= 1'b0;
45             pulse3 <= 1'b0;
46         end else begin
47             pulse1 <= clk;
48             pulse2 <= pulse1;
49             pulse3 <= pulse2;
50         end
51     end
52     assign clk_posedge = pulse2 & ~pulse3;
53
54     always @(posedge clk_origin, posedge rst) begin
55         if (rst) begin
56             tx_en <= 1'b0;
57         end else begin
58             if (clk_posedge) begin
59                 tx_en <= 1'b1;
60             end else begin
61                 tx_en <= 1'b0;
62             end
63         end
64     end
65
66 endmodule
67

```

其中，根据switch的值，addr_temp会分别加上0、256、512，从而实现地址的切换。tx_en则是通过检测分频时钟的上升沿在每次地址切换时产生一个脉冲，用于触发输出模块。

1.3 顶层模块

顶层模块的代码如下：

```
1 // Purpose: Top level module for UART transmit.
2 `include "addr_tx_en2.v"
3 `include "uart_transmit2.v"
4 `include "uart2_fre_div.v"
5 `include "three_waves_rom.v"
6 module uart2_top(
7     input clk,
8     input rst,
9     input [2:0] switch,
10    output sci_tx
11 );
12
13 wire clk_div_addr;
14 uart2_fre_fiv uut1(
15     .clk(clk),
16     .rst(rst),
17     .clk_div_addr(clk_div_addr)
18 );
19
20 wire [9:0] addr;
21 wire tx_en;
22 addr_tx_en2 uut2(
23     .clk_origin(clk),
24     .clk(clk_div_addr),
25     .rst(rst),
26     .switch(switch),
27     .addr(addr),
28     .tx_en(tx_en)
29 );
30
31 wire [9:0] data_temp;
32 three_waves_rom uut3(
33     .address(addr),
34     .clock(clk),
35     .q(data_temp)
36 );
37
38 uart_transmit2 uut4(
39     .clk(clk),
```

```

40     .rst(rst),
41     .tx_en(tx_en),
42     .rx_d(data_temp),
43     .sci_tx(sci_tx)
44 );
45
46
47
48 endmodule

```

RTL图如图2所示。

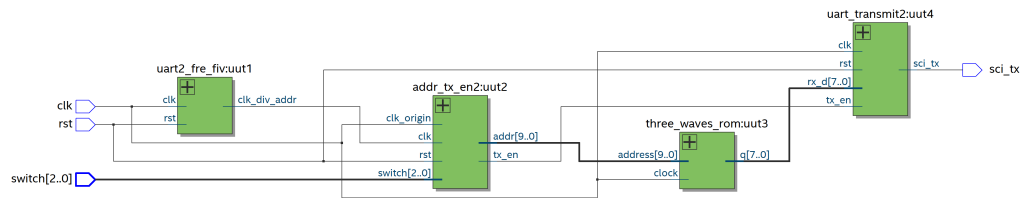


图 2: RTL图

1.4 输出结果

输出结果如图3、4、5所示。三个波形的频率均为50Hz，符合task1的要求。



图 3: 正弦波

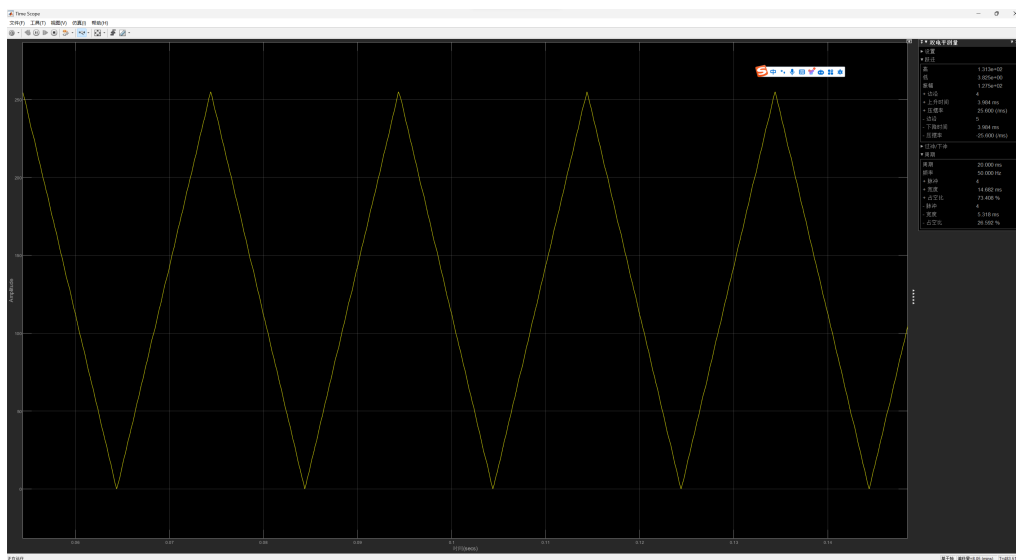


图 4: 三角波

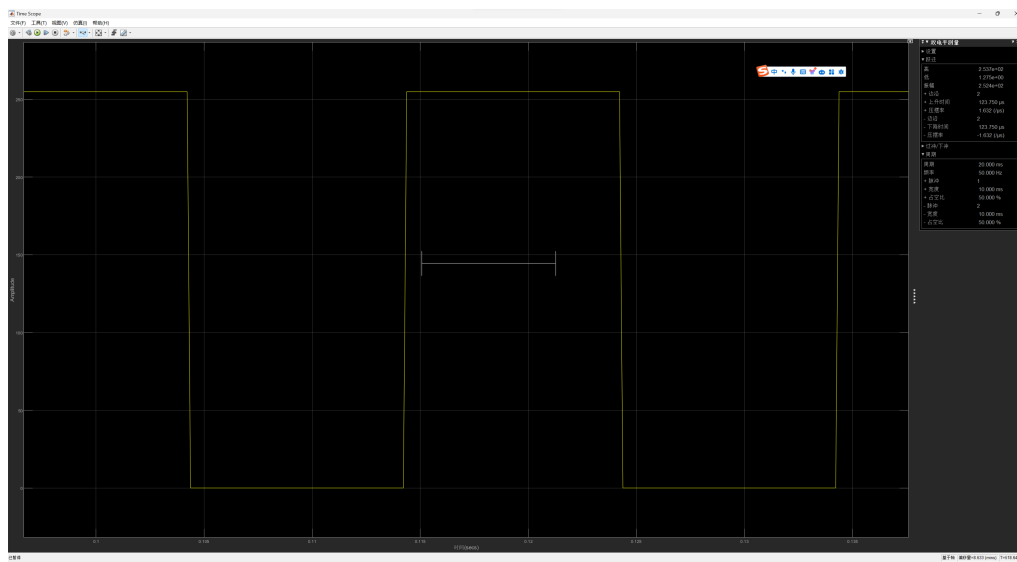


图 5: 方波

2 task3

task3中FPGA计算及显示的部分与上周任务相似，只需要更改数据转换部分。数码管部分不加赘述。

2.1 串口接收模块

串口接收模块的代码如下：

```
1  module uart_receive #(
2      parameter BAUD_RATE = 'd115200,
3      parameter CLK_FREQ  = 25000000
4  ) (
5      input clk,
6      input rst,
7      input rx,
8      output reg [7:0] data,
9      output reg ready
10 );
11
12 //localparam define
13 localparam BAUD_CNT_MAX = CLK_FREQ / BAUD_RATE;
14
15 //reg define
16 reg rx_reg1;
17 reg rx_reg2;
18 reg rx_reg3; //for pulse
19 reg start_nedge;
20 reg work_en;
21 reg [12:0] baud_cnt;
22 reg bit_flag;
23 reg [3:0] bit_cnt;
24 reg [7:0] rx_data;
25 reg rx_flag;
26
27 always @(posedge clk, posedge rst) begin
28     if (rst) begin
29         rx_reg1 <= 1'b1;
30         rx_reg2 <= 1'b1;
31         rx_reg3 <= 1'b1;
32     end else begin
```



```
33         rx_reg1 <= rx;
34         rx_reg2 <= rx_reg1;
35         rx_reg3 <= rx_reg2;
36     end
37 end
38
39 always @(posedge clk, posedge rst) begin
40     if (rst) begin
41         start_nedge <= 1'b0;
42     end else if ((~rx_reg2) && rx_reg3) begin
43         start_nedge <= 1'b1;
44     end else begin
45         start_nedge <= 1'b0;
46     end
47 end
48
49 always @(posedge clk, posedge rst) begin
50     if (rst) begin
51         work_en <= 1'b0;
52     end else if (start_nedge) begin
53         work_en <= 1'b1;
54     end else if ((bit_cnt == 4'd8) && (bit_flag == 1'b1)) begin
55         work_en <= 1'b0;
56     end
57 end
58
59 always @(posedge clk, posedge rst) begin
60     if (rst) begin
61         baud_cnt <= 13'd0;
62     end else if ((baud_cnt == BAUD_CNT_MAX - 1) || (work_en == 1'b0)) begin
63         baud_cnt <= 13'd0;
64     end else if (work_en) begin
65         baud_cnt <= baud_cnt + 1'b1;
66     end
67 end
68
69 always @(posedge clk, posedge rst) begin
70     if (rst) begin
71         bit_flag <= 1'b0;
72     end else if (baud_cnt == BAUD_CNT_MAX / 2 - 1) begin
```

```
73         bit_flag <= 1'b1;
74     end else begin
75         bit_flag <= 1'b0;
76     end
77 end
78
79 always @(posedge clk, posedge rst) begin
80     if (rst) begin
81         bit_cnt <= 4'd0;
82     end else if ((bit_cnt == 4'd8) && (bit_flag == 1'b1)) begin
83         bit_cnt <= 4'd0;
84     end else if (bit_flag == 1'b1) begin
85         bit_cnt <= bit_cnt + 1'b1;
86     end
87 end
88
89 always @(posedge clk, posedge rst) begin
90     if (rst) begin
91         rx_data <= 8'd0;
92     end else if ((bit_cnt >= 4'd1) && (bit_cnt <= 4'd8) && (bit_flag == 1'b1))
          begin
93         rx_data <= {rx_reg3, rx_data[7:1]};
94     end
95 end
96
97 always @(posedge clk, posedge rst) begin
98     if (rst) begin
99         rx_flag <= 1'b0;
100    end else if ((bit_cnt == 4'd8) && (bit_flag == 1'b1)) begin
101        rx_flag <= 1'b1;
102    end else begin
103        rx_flag <= 1'b0;
104    end
105 end
106
107 always @(posedge clk, posedge rst) begin
108     if (rst) begin
109         data <= 8'd0;
110     end else if (rx_flag) begin
111         data <= rx_data;
```

```

112         end
113     end
114
115     always @(posedge clk, posedge rst) begin
116         if (rst) begin
117             ready <= 1'b0;
118         end else begin
119             ready <= rx_flag;
120         end
121     end
122 endmodule

```

主要实现了如下核心功能：

- 通过三个reg将rx打慢两拍，以消除亚稳态。
- 通过计数器实现指定波特率。
- 通过计数器实现中点读取数据，保证数据的稳定性。
- 在8位数据读取完毕后，将数据存入data中，同时将ready置1。

2.2 通讯协议指定

虽然任务要求使用十六进制发送并用数字指代运算符，但是我认为这种形式不是十分优雅，所以我将通讯协议改为了如下形式：**前数+运算符+后数+=**。其中+=均有对应的ascii码，可以直接输入。而与、或和比较分别用A、O、C表示。这样的好处是串口输入较为直观，例如输入**1+2=**即可得到3。同时，这样的协议也可以很方便地扩展到更多的运算符上。

2.3 数据转换模块

数据转换模块的代码如下：

```

1  //process the origin input to a simple one
2      reg [3:0] input_type;
3      parameter NUMBER = 4'b0001;
4      parameter OPERATOR = 4'b0010;
5      parameter EQUAL = 4'b0100;
6      parameter RESET = 4'b1000;
7      reg [4:0] input_temp;

```

```
8     reg input_flag;
9
10    always @(posedge clk, posedge rst) begin
11        if (rst) begin
12            input_type <= 3'b000;
13            input_temp <= 5'b00000;
14            input_flag <= 1'b0;
15        end else if (ready) begin
16            case (rx_data)
17                8'd48: begin
18                    input_type <= NUMBER;
19                    input_temp <= 5'b00000;
20                end
21                8'd49: begin
22                    input_type <= NUMBER;
23                    input_temp <= 5'b00001;
24                end
25                8'd50: begin
26                    input_type <= NUMBER;
27                    input_temp <= 5'b00010;
28                end
29                8'd51: begin
30                    input_type <= NUMBER;
31                    input_temp <= 5'b00011;
32                end
33                8'd52: begin
34                    input_type <= NUMBER;
35                    input_temp <= 5'b00100;
36                end
37                8'd53: begin
38                    input_type <= NUMBER;
39                    input_temp <= 5'b00101;
40                end
41                8'd54: begin
42                    input_type <= NUMBER;
43                    input_temp <= 5'b00110;
44                end
45                8'd55: begin
46                    input_type <= NUMBER;
47                    input_temp <= 5'b00111;
```

```
48         end
49     8'd56: begin
50         input_type <= NUMBER;
51         input_temp <= 5'b01000;
52     end
53     8'd57: begin
54         input_type <= NUMBER;
55         input_temp <= 5'b01001;
56     end
57     8'd43: begin
58         input_type <= OPERATOR;
59         input_temp <= ADD_OPERATOR;
60     end
61     8'd45: begin
62         input_type <= OPERATOR;
63         input_temp <= SUB_OPERATOR;
64     end
65     8'd65: begin
66         input_type <= OPERATOR;
67         input_temp <= AND_OPERATOR;
68     end
69     8'd79: begin
70         input_type <= OPERATOR;
71         input_temp <= OR_OPERATOR;
72     end
73     8'd67: begin
74         input_type <= OPERATOR;
75         input_temp <= COMPARE_OPERATOR;
76     end
77     8'd61: begin
78         input_type <= EQUAL;
79         input_temp <= 5'b00000;
80     end
81     8'd82: begin
82         input_type <= RESET;
83         input_temp <= 5'b00000;
84     end
85     default: begin
86         input_type <= 3'b000;
87         input_temp <= 5'b00000;
```

```

88         input_flag <= 1'b0;
89     end
90 endcase
91 end else begin
92     input_type <= 3'b000;
93     input_temp <= 5'b00000;
94     input_flag <= 1'b0;
95 end
96 end

```

以上代码将串口传入的字符处理为了上周所完成的计算器所需要的格式。其中，input_type用于指示当前输入的类型，input_temp则是处理后的数据。input_flag用于指示是否有新的输入。计算过程中还使用了mealy状态机，其示意图如图6所示。

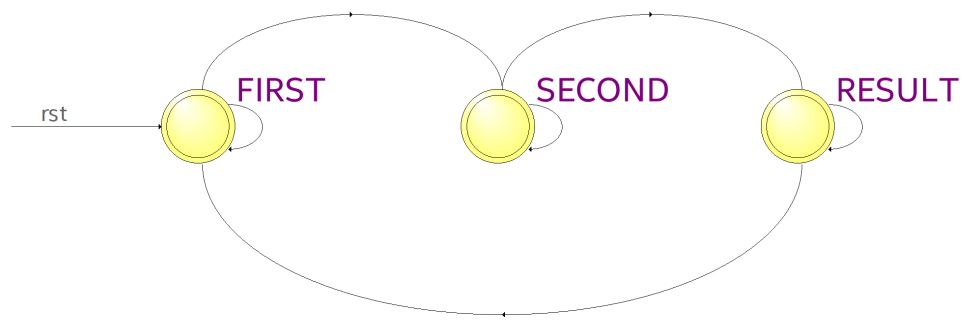


图 6: mealy状态机

2.4 顶层模块设计

顶层模块的代码如下：

```

1  'include "uart_receive.v"
2  'include "uart_calculator.v"
3  'include "uart3_fre_div.v"
4  'include "uart_digit.v"
5  module uart_calculator_top(
6      input clk,
7      input rst,

```

```
8     input rx,
9     output [7:0] seg,
10    output [3:0] digit
11 );
12
13 wire [7:0] data;
14 wire ready;
15 uart_receive uut1(
16     .clk(clk),
17     .rst(rst),
18     .rx(rx),
19     .data(data),
20     .ready(ready)
21 );
22
23 wire [15:0] display_data;
24 uart_calculator uut2(
25     .clk(clk),
26     .rst(rst),
27     .ready(ready),
28     .rx_data(data),
29     .display_data(display_data)
30 );
31
32 wire clk_div;
33 uart3_fre_div uut3(
34     .clk(clk),
35     .rst(rst),
36     .clk_div(clk_div)
37 );
38
39 uart_digit uut4(
40     .clk(clk_div),
41     .rst(rst),
42     .data(display_data),
43     .digit(digit),
44     .seg(seg)
45 );
46 endmodule
```

RTL图如图7所示。

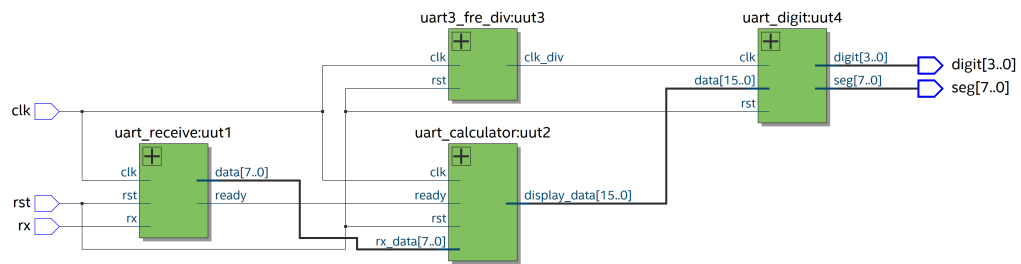


图 7: RTL图