# 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：

```
module addr_tx_en2 (
    input clk,
    input clk_origin,
    input rst,
    input [2:0] switch,
    output reg [9:0] addr,
    output reg tx_en
);

    reg [9:0] addr_temp;
    always @(posedge clk, posedge rst) begin
        if (rst) begin
            addr_temp <= 10'd0;
        end else begin
            if (addr_temp == 10'd255) begin
                addr_temp <= 10'd0;
            end else begin
                addr_temp <= addr_temp + 1'b1;
            end
        end
    end

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            addr <= 10'd0;
        end else begin
            if (switch[0]) begin
                addr <= addr_temp;
            end else if (switch[1]) begin
                addr <= addr_temp + 10'd256;
            end else if (switch[2]) begin
                addr <= addr_temp + 10'd512;
            end else begin
                addr <= 10'd0;
            end
        end
```

```
    end

    reg pulse1, pulse2, pulse3;
    wire clk_posedge;
    always @(posedge clk_origin, posedge rst) begin
        if (rst) begin
            pulse1 <= 1'b0;
            pulse2 <= 1'b0;
            pulse3 <= 1'b0;
        end else begin
            pulse1 <= clk;
            pulse2 <= pulse1;
            pulse3 <= pulse2;
        end
    end
    assign clk_posedge = pulse2 & ~pulse3;

    always @(posedge clk_origin, posedge rst) begin
        if (rst) begin
            tx_en <= 1'b0;
        end else begin
            if (clk_posedge) begin
                tx_en <= 1'b1;
            end else begin
                tx_en <= 1'b0;
            end
        end
    end


endmodule
```

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：

```
// Purpose: Top level module for UART transmit.
`include "addr_tx_en2.v"
`include "uart_transmit2.v"
`include "uart2_fre_div.v"
`include "three_waves_rom.v"
module uart2_top(
    input clk,
    input rst,
    input [2:0] switch,
    output  sci_tx
);

wire clk_div_addr;
uart2_fre_fiv uut1(
    .clk(clk),
    .rst(rst),
    .clk_div_addr(clk_div_addr)
);

wire [9:0] addr;
wire tx_en;
addr_tx_en2 uut2(
    .clk_origin(clk),
    .clk(clk_div_addr),
    .rst(rst),
    .switch(switch),
    .addr(addr),
    .tx_en(tx_en)
);

wire [9:0]data_temp;
three_waves_rom uut3(
    .address(addr),
    .clock(clk),
    .q(data_temp)
);

uart_transmit2 uut4(
    .clk(clk),
```

```
    .rst(rst),
    .tx_en(tx_en),
    .rx_d(data_temp),
    .sci_tx(sci_tx)
);



endmodule
```

RTL图如图2所示。

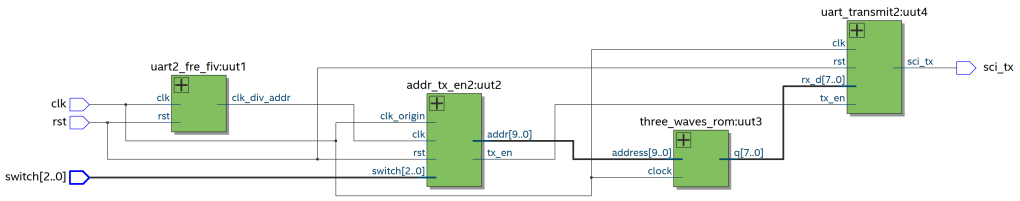


图 2: RTL图

## 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。

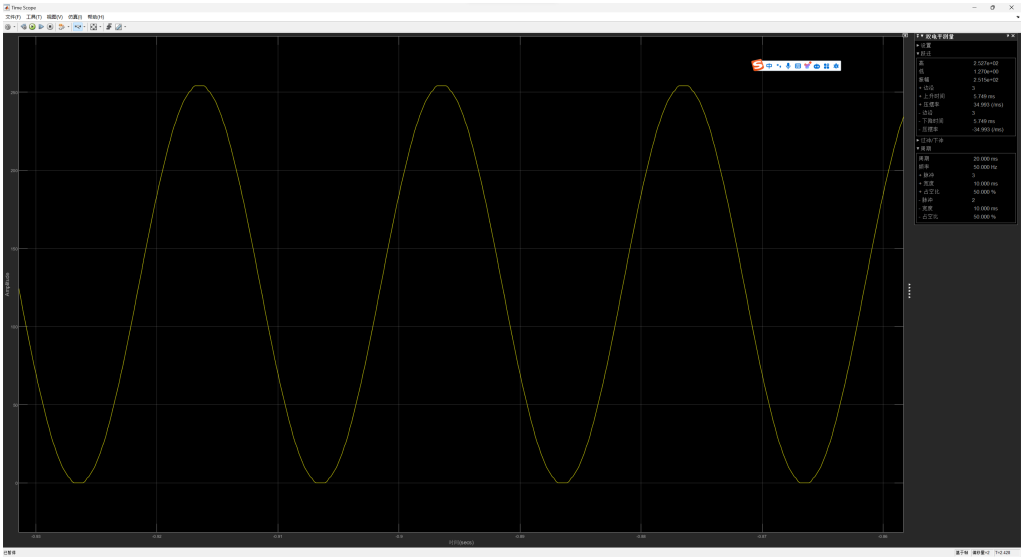

图 3: 正弦波

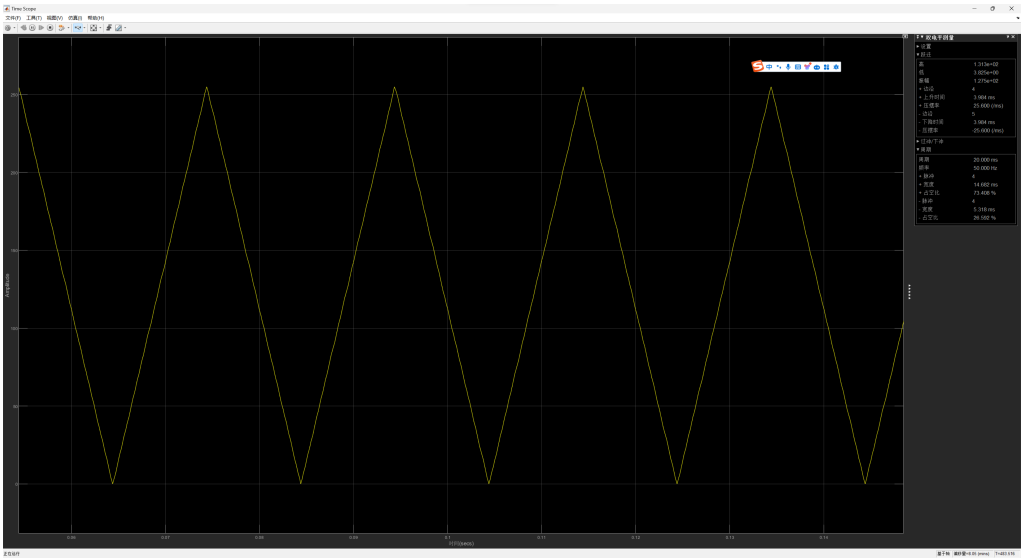

图 4: 三角波

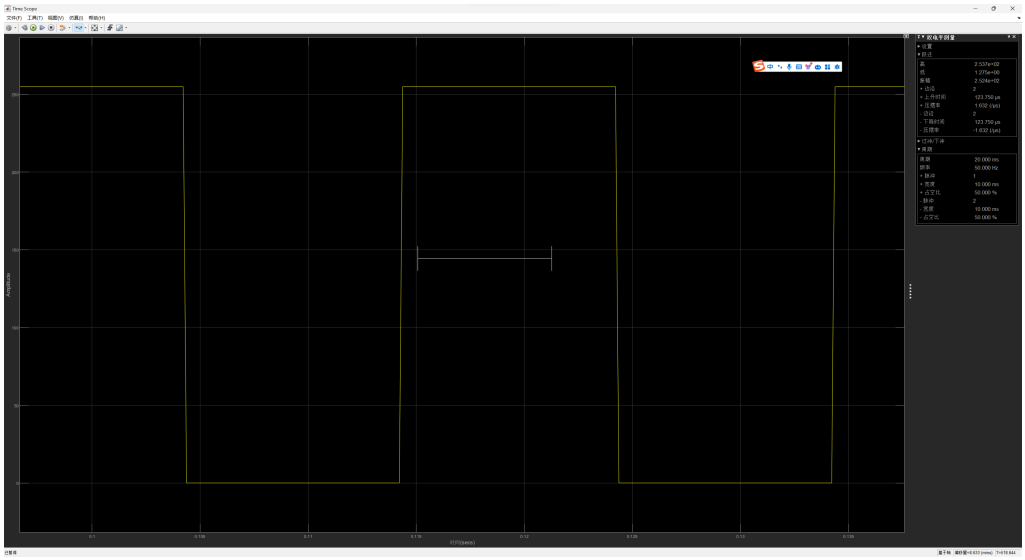

图 5: 方波

# 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：

```
module uart_receive #(
    parameter BAUD_RATE = 'd115200,
    parameter CLK_FREQ  = 25000000
) (
    input clk,
    input rst,
    input rx,
    output reg [7:0] data,
    output reg ready
);

    //localparam define
    localparam BAUD_CNT_MAX = CLK_FREQ / BAUD_RATE;

    //reg define
    reg rx_reg1;
    reg rx_reg2;
    reg rx_reg3;  //for pulse
    reg start_nedge;
    reg work_en;
    reg [12:0] baud_cnt;
    reg bit_flag;
    reg [3:0] bit_cnt;
    reg [7:0] rx_data;
    reg rx_flag;

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            rx_reg1 <= 1'b1;
            rx_reg2 <= 1'b1;
            rx_reg3 <= 1'b1;
        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
```

```
            bit_flag <= 1'b1;
        end else begin
            bit_flag <= 1'b0;
        end
    end

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            bit_cnt <= 4'd0;
        end else if ((bit_cnt == 4'd8) && (bit_flag == 1'b1)) begin
            bit_cnt <= 4'd0;
        end else if (bit_flag == 1'b1) begin
            bit_cnt <= bit_cnt + 1'b1;
        end
    end

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            rx_data <= 8'd0;
        end else if ((bit_cnt >= 4'd1) && (bit_cnt <= 4'd8) && (bit_flag == 1'b1))
             begin
            rx_data <= {rx_reg3, rx_data[7:1]};
        end
    end

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            rx_flag <= 1'b0;
        end else if ((bit_cnt == 4'd8) && (bit_flag == 1'b1)) begin
            rx_flag <= 1'b1;
        end else begin
            rx_flag <= 1'b0;
        end
    end

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            data <= 8'd0;
        end else if (rx_flag) begin
            data <= rx_data;
```

```
        end
    end

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            ready <= 1'b0;
        end else begin
            ready <= rx_flag;
        end
    end
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ascll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：

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

```
    reg input_flag;

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            input_type <= 3'b000;
            input_temp <= 5'b00000;
            input_flag <= 1'b0;
        end else if (ready) begin
            case (rx_data)
                8'd48: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b00000;
                end
                8'd49: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b00001;
                end
                8'd50: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b00010;
                end
                8'd51: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b00011;
                end
                8'd52: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b00100;
                end
                8'd53: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b00101;
                end
                8'd54: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b00110;
                end
                8'd55: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b00111;
```

```
                end
                8'd56: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b01000;
                end
                8'd57: begin
                    input_type <= NUMBER;
                    input_temp <= 5'b01001;
                end
                8'd43: begin
                    input_type <= OPERATOR;
                    input_temp <= ADD_OPERATOR;
                end
                8'd45: begin
                    input_type <= OPERATOR;
                    input_temp <= SUB_OPERATOR;
                end
                8'd65: begin
                    input_type <= OPERATOR;
                    input_temp <= AND_OPERATOR;
                end
                8'd79: begin
                    input_type <= OPERATOR;
                    input_temp <= OR_OPERATOR;
                end
                8'd67: begin
                    input_type <= OPERATOR;
                    input_temp <= COMPARE_OPERATOR;
                end
                8'd61: begin
                    input_type <= EQUAL;
                    input_temp <= 5'b00000;
                end
                8'd82: begin
                    input_type <= RESET;
                    input_temp <= 5'b00000;
                end
                default: begin
                    input_type <= 3'b000;
                    input_temp <= 5'b00000;
```

```
                    input_flag <= 1'b0;
                end
            endcase
        end else begin
            input_type <= 3'b000;
            input_temp <= 5'b00000;
            input_flag <= 1'b0;
        end
    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。

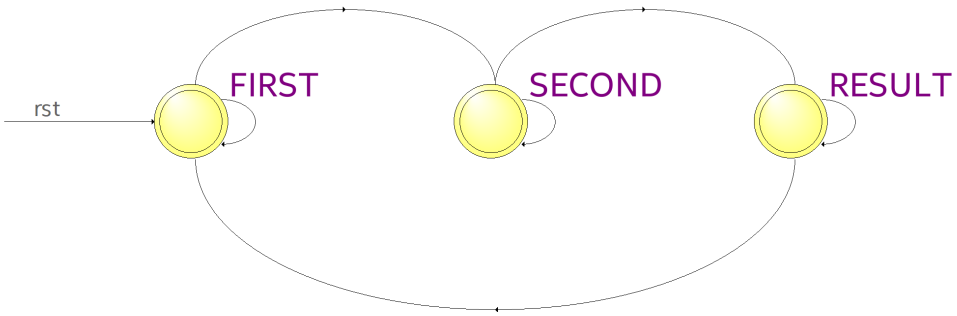


图 6: mealy状态机

## 2.4 顶层模块设计

顶层模块的代码如下：

```
`include "uart_receive.v"
`include "uart_calculator.v"
`include "uart3_fre_div.v"
`include "uart_digit.v"
module uart_calculator_top(
    input clk,
    input rst,
```

```
    input rx,
    output [7:0] seg,
    output [3:0] digit
);

wire [7:0] data;
wire ready;
uart_receive uut1(
    .clk(clk),
    .rst(rst),
    .rx(rx),
    .data(data),
    .ready(ready)
);

wire [15:0] display_data;
uart_calculator uut2(
    .clk(clk),
    .rst(rst),
    .ready(ready),
    .rx_data(data),
    .display_data(display_data)
);

wire clk_div;
uart3_fre_div uut3(
    .clk(clk),
    .rst(rst),
    .clk_div(clk_div)
);

uart_digit uut4(
    .clk(clk_div),
    .rst(rst),
    .data(display_data),
    .digit(digit),
    .seg(seg)
);
endmodule
```

RTL图如图7所示。

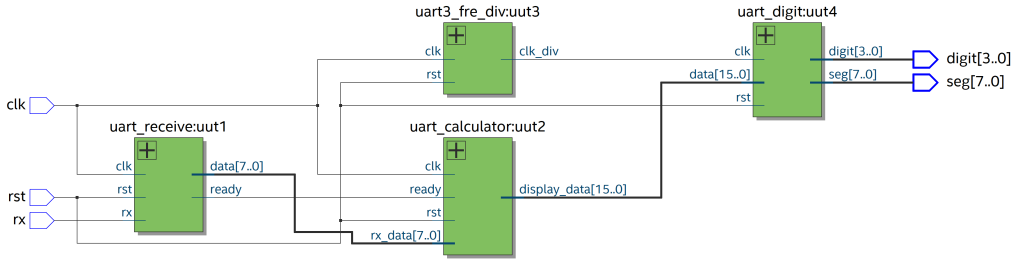


图 7: RTL图