电子技术基础实验第十一周实验报告

王磊　2022012972

2023 年 12 月 6 日

1　模块设计

1.1　16位正弦数据生成

使用matlab生成用于初始化ROM的mif文件，其中存储了256个16位的正弦数据，代码如图1所示。

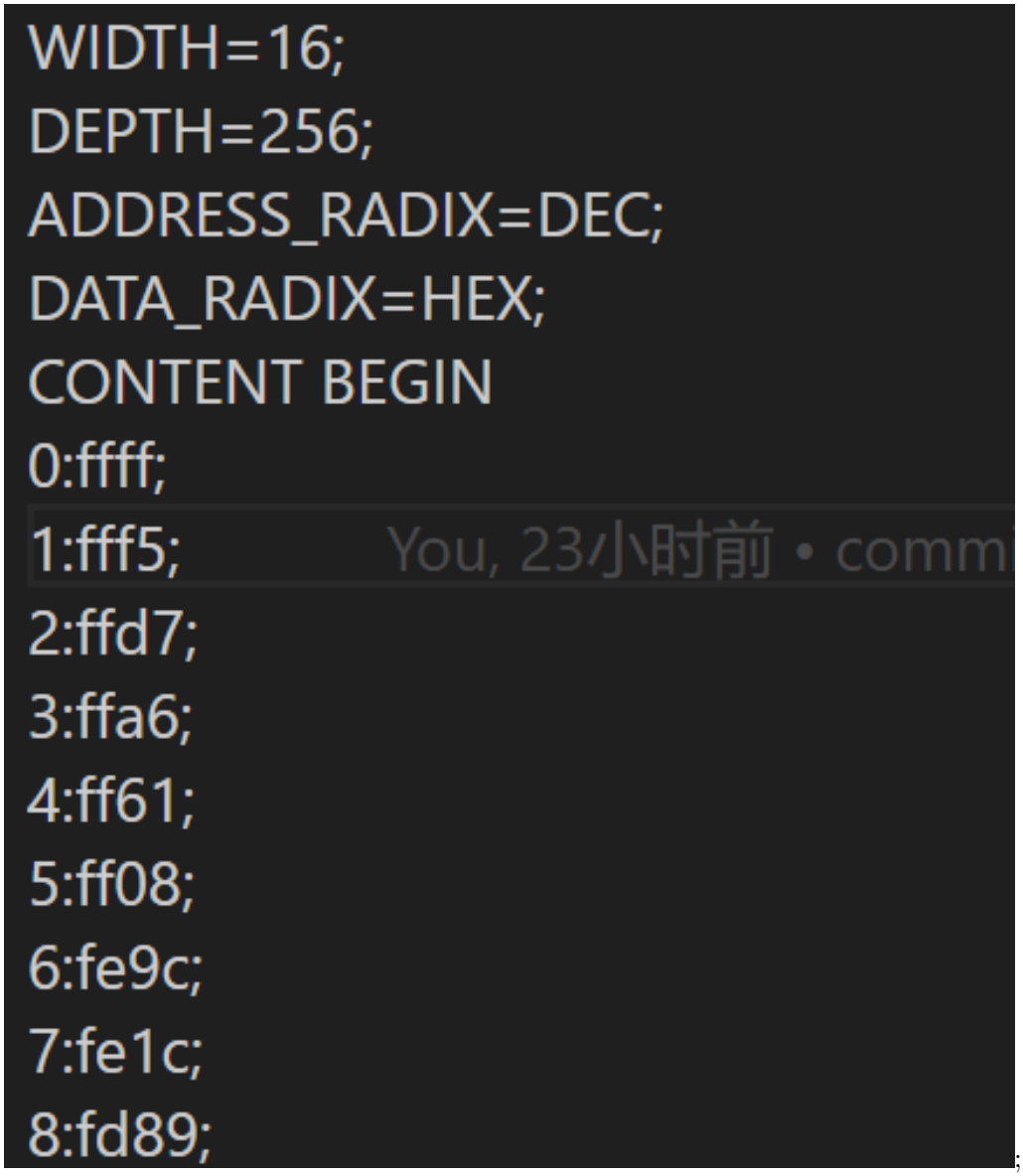
```
% 设置参数
width = 16; % 每个数据元素的位宽为16位
depth = 256; % 存储器的深度，即数据元素的数量为256个
%phase = 0;      % 相位初始为0，这里被注释掉
phase = pi / 2; % 相位设置为π/2，即90度

% 打开文件并写入文件头信息
fid = fopen('sin_phase90_16bit.mif', 'w');
fprintf(fid, 'WIDTH=%d;\n', width);
fprintf(fid, 'DEPTH=%d;\n', depth);
fprintf(fid, 'ADDRESS_RADIX=DEC;\n');
fprintf(fid, 'DATA_RADIX=HEX;\n');
fprintf(fid, 'CONTENT BEGIN\n');

% 循环生成16位正弦波数据并写入文件
for i = 0:depth - 1
    sin_data = floor((sin(2 * pi * i / depth + phase) + 1) * 0.5 * (2 ^ width - 1));
    fprintf(fid, '%d:%x;\n', i, sin_data); % 以十六进制格式写入文件
end

% 写入文件尾部信息并关闭文件
fprintf(fid, 'END ;\n');
fclose(fid);          You, 23小时前 • commit task1
```

;

图 1: matlab代码

;

生成的mif文件内容如图2所示。



;

图 2: mif文件内容

;

1.2 实现60个16位数据加帧头发送

1.2.1 顶层模块

顶层模块代码如下：

```
1 // This module is the top-level module for transmitting UART frames.
2 // It includes various sub-modules for address generation, data generation, and
      UART transmission.
3
```

```
`include "addr_tx_en.v" // Include module for address and transmit enable generation.
`include "sin_16_rom.v" // Include module for generating sine wave data.
`include "fre_div.v" // Include module for clock frequency division.
`include "uart_byte_controller.v" // Include module for byte-level UART transmission control.
`include "uart_2byte_controller_withoutHeader.v" // Include module for 2-byte UART transmission control without header.
`include "uart_tx_byte.v" // Include module for UART byte transmission.

module uart_frame_transmit_top (
    input  clk, // Clock input
    input  rst, // Reset input
    output sci_tx // UART transmit output
);

    wire clk_div_addr; // Clock divided address signal
    fre_div #(25'd12500) uut1 ( // Instantiate frequency divider module
        .clk(clk),
        .rst(rst),
        .clk_div_addr(clk_div_addr)
    );

    wire [7:0] addr; // Address signal
    wire send_en; // Transmit enable signal
    wire [15:0] data_out; // Output data signal

    addr_tx_en uut2 ( // Instantiate address and transmit enable module
        .clk_origin(clk),
        .clk(clk_div_addr),
        .rst(rst),
        .addr(addr),
        .data_temp(data_temp),
        .data_out(data_out),
        .tx_en(send_en)
    );

    wire [15:0] data_temp; // Temporary data signal
    sin_16_rom uut3 ( // Instantiate sine wave ROM module
        .address(addr),
```

```
        .clock(clk),
        .q(data_temp)
    );

    wire tx_en; // Transmit enable signal
    wire [7:0] tx_d; // Transmit data signal
    wire tx_done; // Transmission completion signal

    uart_2byte_controller_withoutHeader uut4 ( // Instantiate 2-byte UART
        controller module without header
        .clk(clk),
        .rst(rst),
        .send_en(send_en),
        .tx_en(tx_en),
        .tx_done(tx_done),
        .data(data_out),
        .tx_d(tx_d)
    );

    uart_tx_byte uut5 ( // Instantiate UART byte transmission module
        .clk(clk),
        .rst(rst),
        .tx_en(tx_en),
        .tx_done(tx_done),
        .rx_d(tx_d),
        .sci_tx(sci_tx)
    );

endmodule
```

顶层模块的RTL图如图3所示。

具体的实现逻辑为：

1. 通过分频模块分出1000Hz的时钟，用于触发地址模块。
2. 地址模块每隔1ms产生一个地址，用于读取ROM中的数据。
3. 地址模块同时具有计数与数据处理功能，从ROM中拿到的数据会先进入地址模块，按照任务要求加帧头帧尾后发送给16位UART发送模块。

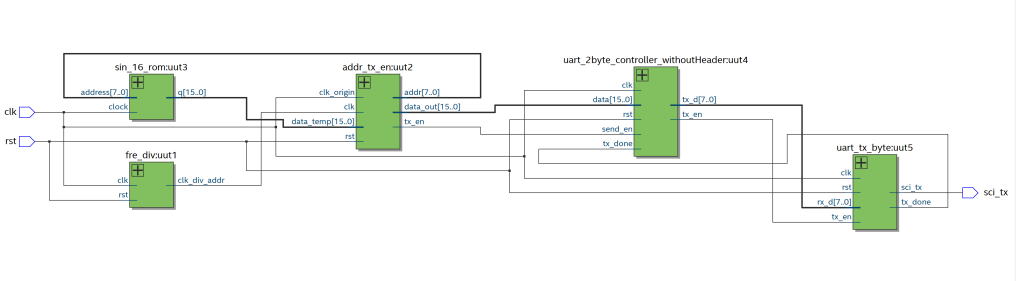


图 3: 顶层模块RTL图

1.2.2 地址模块

该模块为本次任务的核心模块，代码如下：

```
// an idiot tried to write a counter in the state machine, and he failed.
// maybe put the counter in this module will work.

module addr_tx_en #(
    parameter FRAMENUM = 60
) (
    input clk,  // Clock input
    input clk_origin,  // Original clock input
    input rst,  // Reset input
    input [15:0] data_temp,  // Temporary data input
    output reg [15:0] data_out,  // Output data
    output reg [7:0] addr,  // Address output
    output reg tx_en  // Transmit enable output
);

    reg [5:0] frame_cnt;  // Frame counter

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            addr <= 8'b00000000;  // Reset address
            frame_cnt <= 6'b000000;  // Reset frame counter
        end else begin
            if (addr == 8'b11111111) begin  // Check if address is at maximum
                value
                addr <= 8'b00000000;  // Reset address
            end else begin
```

```
                if (frame_cnt == FRAMENUM - 1) begin  // Check if frame counter is
                    at maximum value
                    addr <= addr;  // Keep current address
                    data_out <= 16'h4545;  // Set output data
                    frame_cnt <= 6'b000000;  // Reset frame counter
                end else if (frame_cnt == 0) begin  // Check if frame counter is
                   at minimum value
                    addr <= addr;  // Keep current address
                    data_out <= 16'h5353;  // Set output data
                    frame_cnt <= frame_cnt + 1'b1;  // Increment frame counter
                end else begin
                    addr <= addr + 1'b1;  // Increment address
                    data_out <= data_temp;  // Set output data
                    frame_cnt <= frame_cnt + 1'b1;  // Increment frame counter
                end
            end
        end
    end

    reg pulse1, pulse2, pulse3;  // Clock pulse signals
    wire clk_posedge;  // Positive edge clock signal

    always @(posedge clk_origin, posedge rst) begin
        if (rst) begin
            pulse1 <= 1'b0;  // Reset pulse1
            pulse2 <= 1'b0;  // Reset pulse2
            pulse3 <= 1'b0;  // Reset pulse3
        end else begin
            pulse1 <= clk;  // Assign pulse1 with clock input
            pulse2 <= pulse1;  // Assign pulse2 with pulse1
            pulse3 <= pulse2;  // Assign pulse3 with pulse2
        end
    end

    assign clk_posedge = pulse2 & ~pulse3;  // Calculate positive edge clock
       signal

    always @(posedge clk_origin, posedge rst) begin
        if (rst) begin
            tx_en <= 1'b0;  // Reset transmit enable
```

```
        end else begin
            if (clk_posedge) begin  // Check if positive edge clock signal is high
                tx_en <= 1'b1;  // Set transmit enable
            end else begin
                tx_en <= 1'b0;  // Reset transmit enable
            end
        end
    end

endmodule
```

本模块实现了一个计数器，用于记录发送的16位数据的个数。当计数器的值为0时，发送帧头；当计数器的值为59时，发送帧尾；其他情况下，发送数据。发送帧头帧尾的方式为：使用一个buffer先将ROM中的数读入备用，若需发送帧头或帧尾。扣留数据，使用帧头帧尾替换。同时让addr停止自增，确保数据不会中断。

2 实现结果

使用simulink接收串口数据，结果如图4所示。

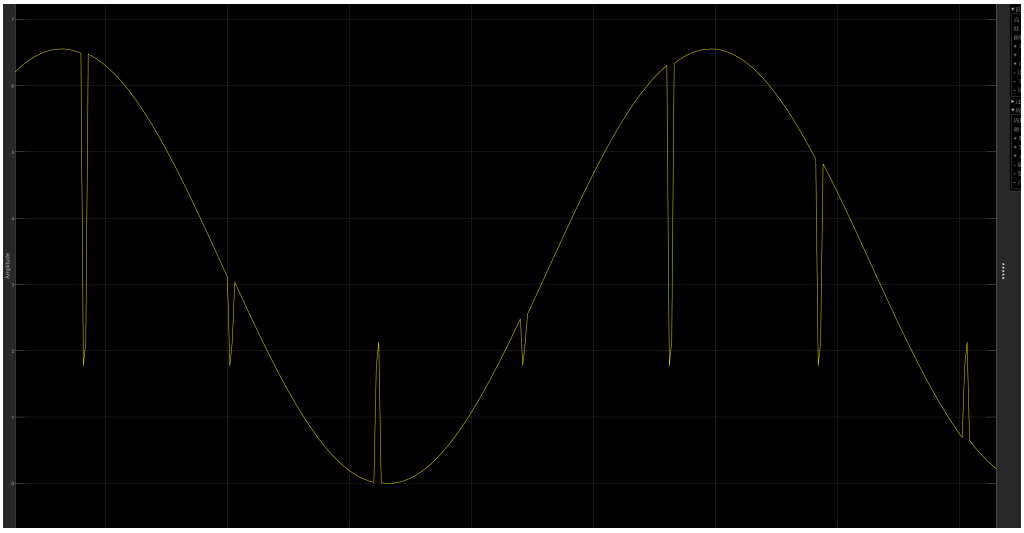


图 4: simulink接收数据

其中定期产生的突刺为帧头帧尾，其他数据为正弦数据，可以看出帧头帧尾的发送不会影响正弦数据的完整性。