SICK
Sensor Intelligence.

SSI 接口驱动开发指引

Synchronous serial interface for absolute encoders

广州市西克传感器有限公司

2016 年 1 月

目录

一 、SSI 接口介绍 3
二 、SSI 接口使用常遇问题 3
三 、本文档适用对象 3
四 、方案涉及使用器件 3
五 、程序 4
六 、 调试 13
七 、数据输出 16
附注： 17

一 、SSI 接口介绍

1. SSI 接口由 SICK Stegmann GmbH 公司开创 现已成为世界通用标准接口协议
2. SSI 接口的物理层基于 RS-422 (EIA 422) 或 RS-485 (EIA 485)
3. SSI 接口由 4 芯同步信号构成
4. 采用双绞线传输 SSI 信号，抗干扰能力强
5. 时钟信号和数据信号采用差分方式传输（有效减少电磁干扰）
6. 单圈分辨率可达 18 位（262144）

二 、SSI 接口使用常遇问题

1. SSI 编码器使用需要对应的接口模块，成本较高。
2. 对于一些简单应用，目前市面上没有 PC 端硬件支持 SSI 接口，客户需增加 PLC 。
3. SICK 编码器单圈分辨率最高 18 位，但市场上主流的 SSI 模块（包括 Simense SM338）只能支持最多单圈 13 位。
4. 客户独立开发 SSI 接口遇到各种问题等。
5. 越来越多的客户需要订制化系统，需将接口集成到其片上系统以适应其应用。

三 、本文档适用对象

1. 对 SSI 接口协议熟悉，对接口信号时序熟悉
2. 对单片机、AVR、STM32 或 FPGA 其中一种有较好的基础，掌握其编程语言
3. 对电子电路及数字电路有一定基础
4. 了解串口协议及会使用相应的串口驱动模块

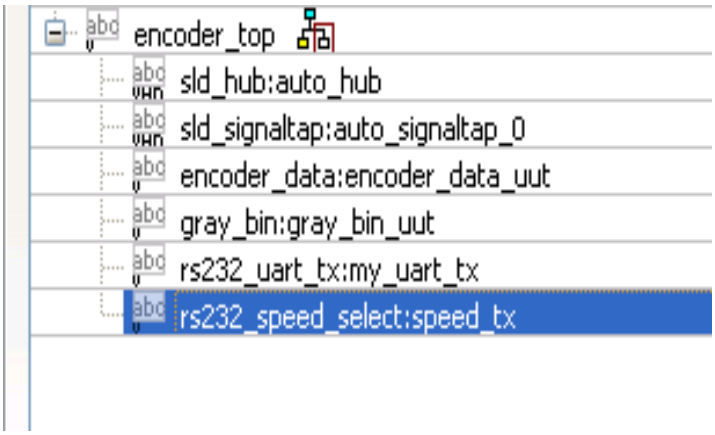
四 、方案涉及使用器件

1. 5V DC 电源
2. AFS60A-S4AK262144 1037443 SSI 接口 18 位单圈绝对值编码器
3. Altera Cyclone FPGA + USB-Blaster
4. Altera Quartus II + Modelsim
5. SN75176A / SN65LBC176A 差分总线收发模块
6. LM1117T 电源模块

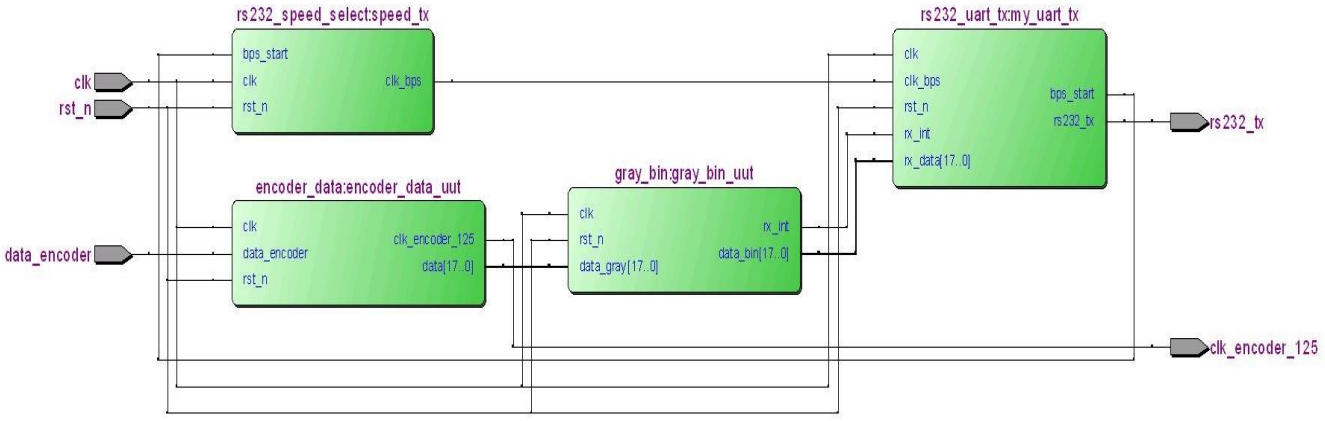
五 、程序

1. 程序文件

encoder_top : 顶层模块
encoder_data : 时钟生成及编码器数据采集。
gray_bin : 格雷码转换为二进制
speed_tx : RS232 串口波特率生成
my_uart_tx : RS232 串口数据发送



2. RTL 视图及信号注释




```
1 module encoder_top(
2                     clk ,                 // 50MHz 时钟
3                     rst_n ,               // 复位，低电平有效
4                     data_encoder ,        // 编码器数据输出 data
5                     clk_encoder_125 ,     // 编码器时钟 clk
6                     rs232_tx              // 串口发送数据
7 ) ;
8
```

3. 编码器数据处理

```
module encoder_data(
                    clk ,
                    rst_n ,
                    data_encoder ,
                    data ,
                    clk_encoder_125
) ;
input clk ,
      rst_n ;
input data_encoder ;            // 编码器的输出 Data+ 、 Data-信号 。
output[17:0] data ;
output clk_encoder_125 ;
parameter       ENCODER_RESOLUTION = 18 ;
parameter       TP_INTERVAL        = 8  ;
/* 计数器 cnt_8us 计数 200 次 */
reg[7:0] cnt_8us ;
always@( posedge clk or negedge rst_n )
    if( !rst_n )
        cnt_8us <= 0 ;
    else if( cnt_8us == 199 )
        cnt_8us <= 0 ;
    else
        cnt_8us <= cnt_8us + 1'b1 ;
reg clk_8us ;
always@( posedge clk or negedge rst_n )
    if( !rst_n )
        clk_8us <= 1 ;
    else if( cnt_8us == 199 )
        clk_8us <= ~clk_8us ;
    else
        clk_8us <= clk_8us ;
/* clk_encoder_125 , 每输出 18 个， 间隔 Tp 时间。常态为高电平 */
reg[4:0] cnt_clk_encoder ;  // 计数 18+8=26 个。
always@( posedge clk_8us or negedge rst_n )
    if( !rst_n )
        cnt_clk_encoder <= 0 ;
    else if( cnt_clk_encoder == ( ENCODER_RESOLUTION + TP_INTERVAL - 1 ) )  // 18+8-1
= 25
        cnt_clk_encoder <= 0 ;
```

```
    else
        cnt_clk_encoder <= cnt_clk_encoder + 1'b1 ;
reg tp ;
always@( posedge clk or negedge rst_n )
    if( !rst_n )
        tp <= 1 ;
    else if( cnt_clk_encoder < ENCODER_RESOLUTION + 1 )
        tp <= 0 ;
    else
        tp <= 1 ;
reg clk_encoder_125 ;  // 将 clk_encoder_125 时钟从 clk_8us 同步过来
always@( posedge clk or negedge rst_n )
    if( !rst_n )
        clk_encoder_125 <= 1 ;
    else if( !tp )
        clk_encoder_125 <= clk_8us ;
reg monoflop ;
always@( negedge clk_8us or negedge rst_n )
    if( !rst_n )
        monoflop <= 1 ;
    else if( ( monoflop ) && ( cnt_clk_encoder < 20 ) ) // 18 位数据 + 16us 延时
        monoflop <= 0 ;
    else if( ( !monoflop ) && ( cnt_clk_encoder == 20 ) )
        monoflop <= 1 ;
    else
        monoflop <= monoflop ;
reg[17:0] data_temp ;
always@( negedge clk_encoder_125 or negedge rst_n )
    if( !rst_n )
        data_temp <= 18'd0 ;
    else if( !monoflop )
        data_temp[ ENCODER_RESOLUTION - cnt_clk_encoder ] <= data_encoder ;
reg[17:0] data ;
always@( posedge clk )
    if( ( monoflop ) && ( tp ) )
        data <= data_temp ;
    else
        data <= data ;

endmodule
```

4. 格雷码转换为二进制码

```
module gray_bin(
                clk ,
                rst_n ,
                data_gray,
                data_bin ,
                rx_int
) ;
parameter DATA_WIDTH = 18 ;          // 设置二进制格雷码的位宽
parameter TIME_50MS  = 2500000 ;     // 计时 50ms 计数器
input clk ;
input rst_n ;
input[DATA_WIDTH - 1 : 0] data_gray ;
output[DATA_WIDTH - 1 : 0] data_bin ;
output rx_int ;
integer i;
reg[DATA_WIDTH - 1 : 0] data_bin_temp ;
always@( posedge clk )
    begin
        data_bin_temp[DATA_WIDTH - 1] <= data_gray[DATA_WIDTH - 1];
        for( i = 1 ; i <= DATA_WIDTH - 1 ; i = i + 1 )
            data_bin_temp[i - 1] <= ( data_gray[i - 1] ^ data_bin_temp[i] );
    end
reg[DATA_WIDTH - 1 : 0] data_bin_temp2 ;
always@( posedge clk )
    data_bin_temp2 <= data_bin_temp ;
reg[DATA_WIDTH - 1 : 0] data_bin ;
always@( posedge clk )
    if( data_bin_temp == data_bin_temp2 )
        data_bin <= data_bin_temp2 ;     // 编码器数据转换为二进制。
reg[24:0] cnt_50ms ;                     // RS232 串口输出数据刷新频率为 50ms
always@( posedge clk or negedge rst_n )
    if( !rst_n )
        cnt_50ms <= 0 ;
    else if( cnt_50ms == TIME_50MS - 1 )
        cnt_50ms <= 0 ;
    else
        cnt_50ms <= cnt_50ms + 1'b1 ;
reg rx_int ;
always@( posedge clk or negedge rst_n )
```

```
    if( !rst_n )
        rx_int <= 0 ;
    else if( cnt_50ms == TIME_50MS - 1 )
        rx_int <= 1 ;
    else
        rx_int <= 0 ;
endmodule
```

4. 串口波特率生成模块

串口波特率可设置为 9600～115200bps ，此处程序中使用波特率 9600bps

```
`timescale 1ns / 1ps
module rs232_speed_select(
                clk ,
                rst_n ,
                bps_start ,
                clk_bps
            );
input clk;              // 50MHz 主时钟
input rst_n;            //低电平复位信号
input bps_start;        //接收到数据后，波特率时钟启动信号置位
output clk_bps;         // clk_bps 的高电平为接收或者发送数据位的中间采样点
/*
parameter       bps9600     = 5207, //波特率为 9600bps
                bps19200    = 2603, //波特率为 19200bps
                bps38400    = 1301, //波特率为 38400bps
                bps57600    = 867,  //波特率为 57600bps
                bps115200   = 433;  //波特率为 115200bps
parameter       bps9600_2   = 2603,
                bps19200_2  = 1301,
                bps38400_2  = 650,
                bps57600_2  = 433,
                bps115200_2 = 216;
*/
    //以下波特率分频计数值可参照上面的参数进行更改
`define     BPS_PARA        5207    //波特率为 9600 时的分频计数值
`define     BPS_PARA_2      2603    //波特率为 9600 时的分频计数值的一半，用于数
据采样
reg[12:0] cnt;                      //分频计数
reg clk_bps_r;                      //波特率时钟寄存器
```

```
//-------------------------------------------------------------
reg[2:0] uart_ctrl; // uart 波特率选择寄存器
//-------------------------------------------------------------
always @ (posedge clk or negedge rst_n)
    if(!rst_n) cnt <= 13'd0;
    else if((cnt == `BPS_PARA) || !bps_start) cnt <= 13'd0; //波特率计数清零
    else cnt <= cnt+1'b1;                                    //波特率时钟计数启动
always @ (posedge clk or negedge rst_n)
    if(!rst_n) clk_bps_r <= 1'b0;
    else if(cnt == `BPS_PARA_2) clk_bps_r <= 1'b1;      // clk_bps_r 高电平为接
收数据位的中间采样点,同时也作为发送数据的数据改变点
    else clk_bps_r <= 1'b0;
assign clk_bps = clk_bps_r;
endmodule
```

5. 串口数据输出模块

```
`timescale 1ns / 1ps
module rs232_uart_tx(
                    clk ,
                    rst_n ,
                    rx_data ,
                    rx_int ,
                    rs232_tx ,
                    clk_bps ,
                    bps_start
            );
input clk;          // 50MHz 主时钟
input rst_n;        //低电平复位信号
input clk_bps;      // clk_bps_r 高电平为接收数据位的中间采样点,同时也作为发送数
据的数据改变点
input rx_int;       //接收数据中断信号,接收到数据期间始终为高电平,在该模块中利
用它的下降沿来启动串口发送数据
input[17:0] rx_data;    //待发送数据寄存器
output rs232_tx;    // RS232 按位发送的数据
output bps_start;   //接收或者要发送数据，波特率时钟启动信号置位
//-------------------------------------------------------------
reg rx_int0 , rx_int1 , rx_int2 ;   //rx_int 信号寄存器，捕捉下降沿滤波用
wire neg_rx_int;    // rx_int 下降沿标志位
always @ (posedge clk or negedge rst_n) begin
```

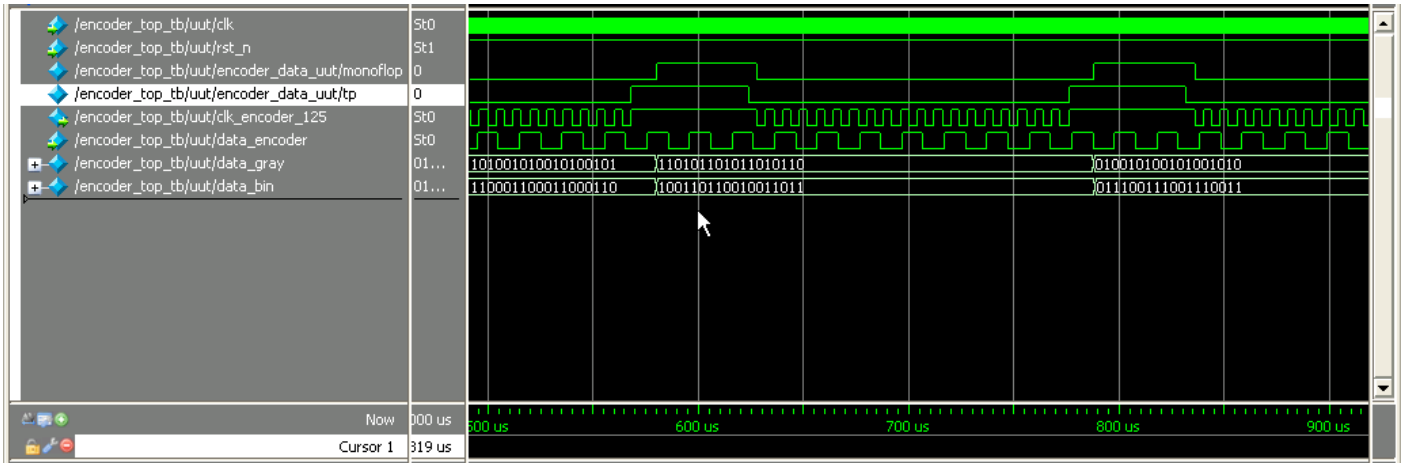
```
    if(!rst_n) begin
            rx_int0 <= 1'b0;
            rx_int1 <= 1'b0;
            rx_int2 <= 1'b0;
        end
    else begin
            rx_int0 <= rx_int;
            rx_int1 <= rx_int0;
            rx_int2 <= rx_int1;
        end
end
assign neg_rx_int =  ~rx_int1 & rx_int2;    //捕捉到下降沿后，neg_rx_int 拉高保
持一个主时钟周期
//-------------------------------------------------------
reg[17:0] tx_data  ;                    //待发送数据的寄存器
//-------------------------------------------------------
reg bps_start_r;
reg tx_en;  //发送数据使能信号，高有效
reg[7:0] num;
always @ (posedge clk or negedge rst_n) begin
    if(!rst_n) begin
            bps_start_r <= 1'b0;
            tx_en <= 1'b0;
//          tx_data <= 18'd0;
        end
    else if(neg_rx_int) begin   //将 rx_int 拉低表示要将当前数据通过串口发出
            bps_start_r <= 1'b1;
//          tx_data <= rx_data; //把接收到的数据存入发送数据寄存器
            tx_en <= 1'b1;      //进入发送数据状态中
        end
    else if(num==8'd30) begin   //数据发送完成，复位
            bps_start_r <= 1'b0;
            tx_en <= 1'b0;
        end
end
assign bps_start = bps_start_r;
/* 待发送数据更新，保证串口发送期间，其数据不能变 */
always@( posedge clk or negedge rst_n )
    if( !rst_n )
        tx_data <= 18'd0 ;
```

```
    else if( !tx_en )
    begin
        if( tx_data != rx_data)
            tx_data <= rx_data ;
    end
    else
        tx_data <= tx_data ;
//-------------------------------------------------------
reg rs232_tx_r;

always @ (posedge clk or negedge rst_n) begin
    if(!rst_n) begin
            num <= 8'd0;
            rs232_tx_r <= 1'b1;
        end
    else if(tx_en) begin
            if(clk_bps) begin
                    num <= num+1'b1;
                    case (num)
                8'd0: rs232_tx_r <= 1'b0;     //发送起始位
                    8'd1: rs232_tx_r <= tx_data[16];
                    8'd2: rs232_tx_r <= tx_data[17];
                8'd9: rs232_tx_r <= 1'b1;     //发送结束位

                8'd10: rs232_tx_r <= 1'b0;    //发送起始位
                    8'd11: rs232_tx_r <= tx_data[8];
                    8'd12: rs232_tx_r <= tx_data[9];
                    8'd13: rs232_tx_r <= tx_data[10];
                    8'd14: rs232_tx_r <= tx_data[11];
                    8'd15: rs232_tx_r <= tx_data[12];
                    8'd16: rs232_tx_r <= tx_data[13];
                    8'd17: rs232_tx_r <= tx_data[14];
                    8'd18: rs232_tx_r <= tx_data[15];
                8'd19: rs232_tx_r <= 1'b1;    //发送结束位

                8'd20: rs232_tx_r <= 1'b0;    //发送起始位
                    8'd21: rs232_tx_r <= tx_data[0];
                    8'd22: rs232_tx_r <= tx_data[1];
                    8'd23: rs232_tx_r <= tx_data[2];
                    8'd24: rs232_tx_r <= tx_data[3];
```

```
                        8'd25: rs232_tx_r <= tx_data[4];
                        8'd26: rs232_tx_r <= tx_data[5];
                        8'd27: rs232_tx_r <= tx_data[6];
                        8'd28: rs232_tx_r <= tx_data[7];
                    8'd29: rs232_tx_r <= 1'b1;   //发送结束位

                        default: rs232_tx_r <= 1'b0;
                        endcase
                end
            else if(num==8'd30) num <= 8'd0;    //复位
        end
end
assign rs232_tx = rs232_tx_r;
endmodule
```

六 、 调试

1. 仿真波形。



与 SSI 接口手册中对比。输出波形时序符合手册要求。

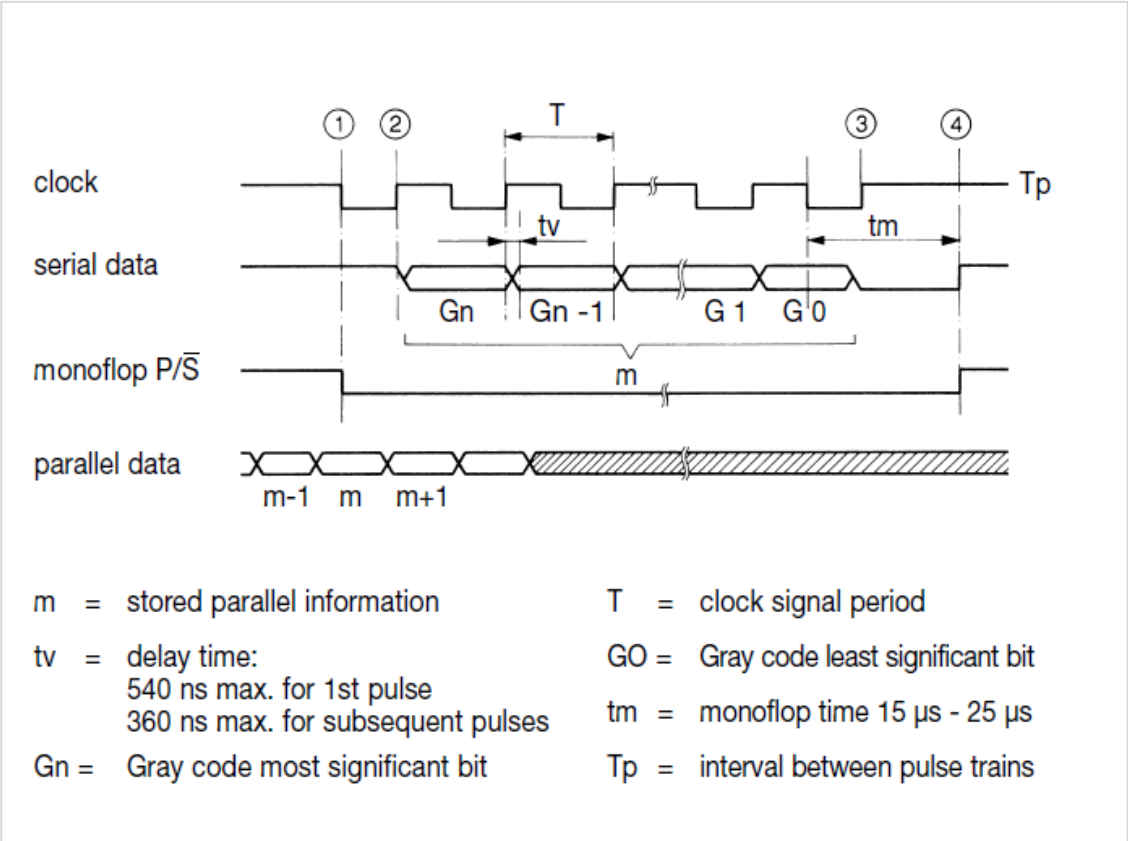
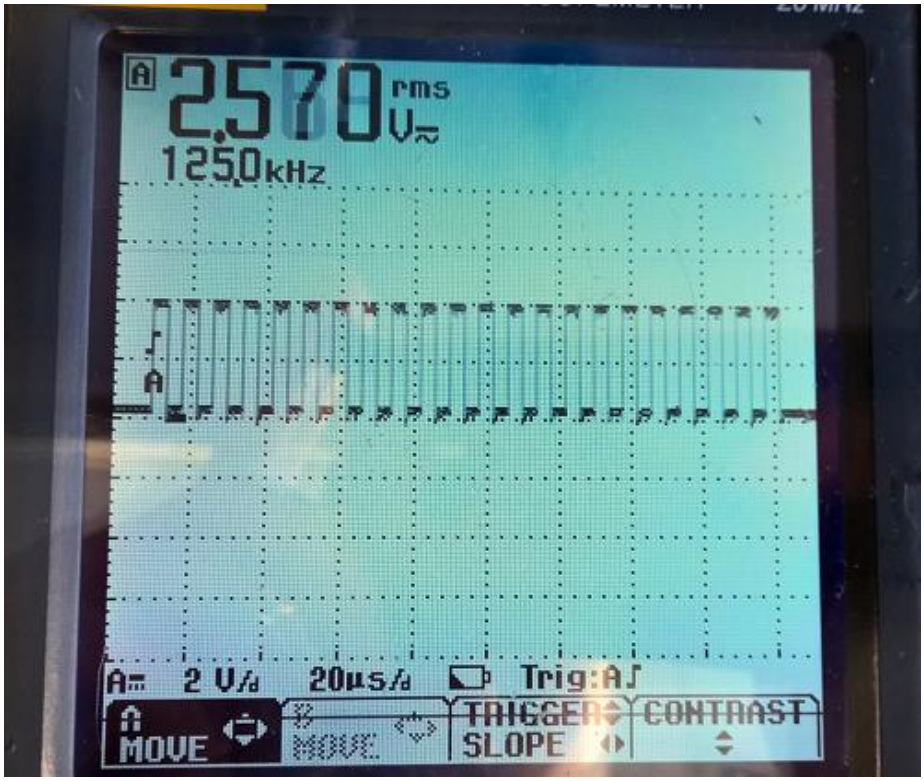


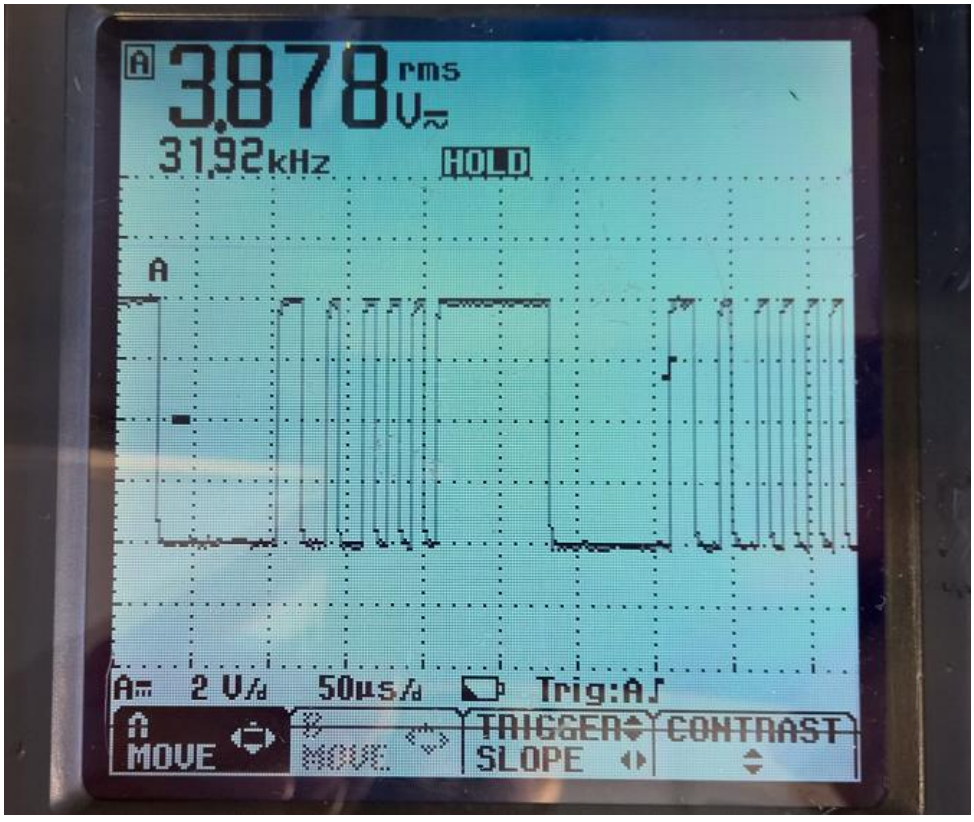
Diagram 11: data transmission timing diagram

2. 示波器检测编码器时钟、数据波型

Clk 时钟频率为 125KHz 。



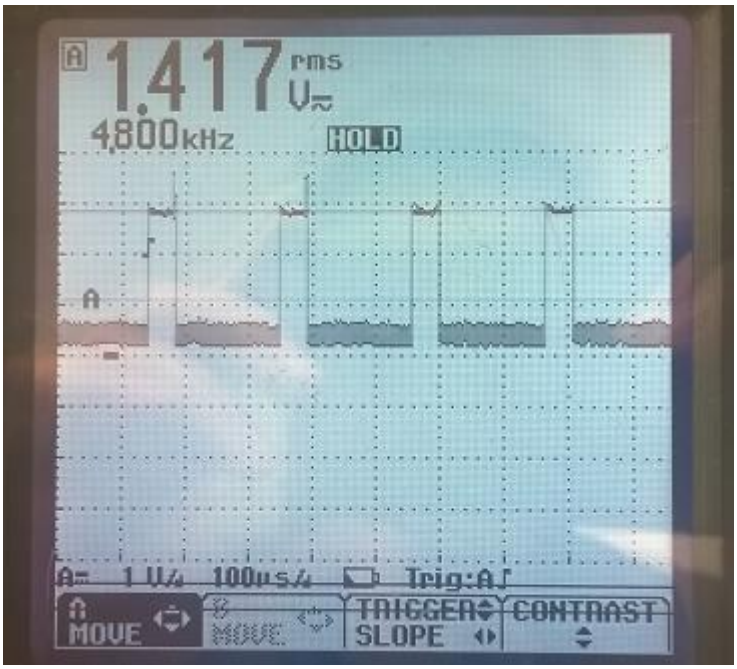
Data 数据



3. 注意事项

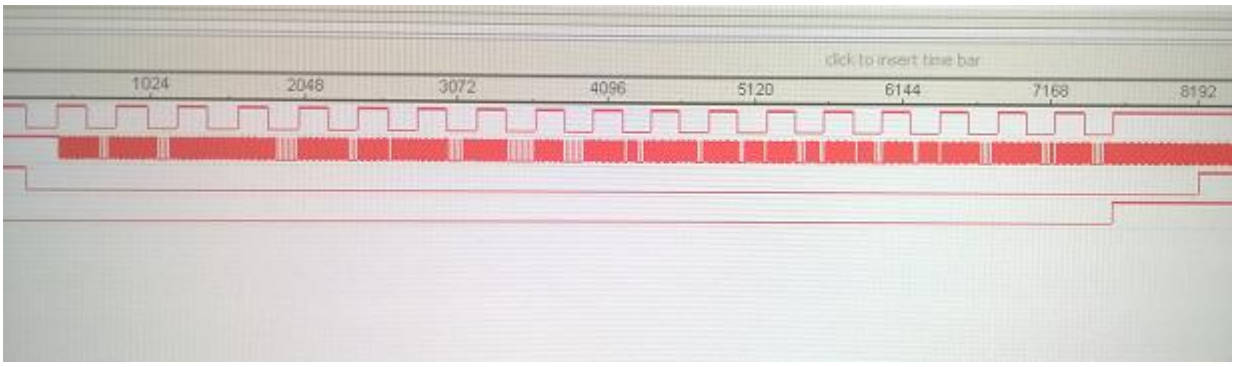
FPGA 对时序要求很高，当使用时钟频率 50MHz 时，其时钟周期为 20ns ，且对干扰敏感。需注意电路接线优化及信号共地等 EMC 处理。

如下图为示波器检测编码器 Data 数据输出波形（编码器位置输出为 0 时），可看到编码器数据有干扰，但干扰电压不超过 0.5V



使用串口监视编码器输出时，发现其输出数值不稳定，有跳动。

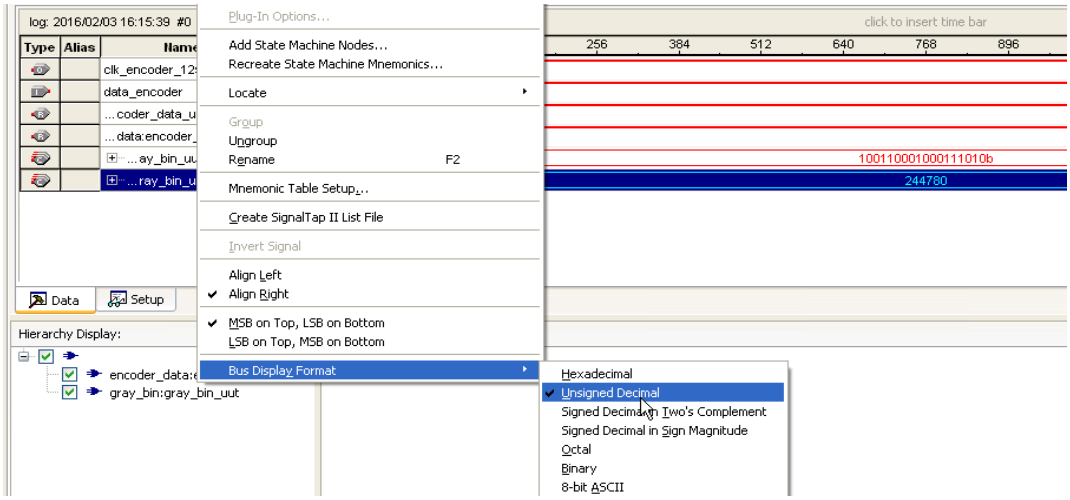
使用 SignalTag 在线分析仪检测编码器信号，发现其有很多的干扰



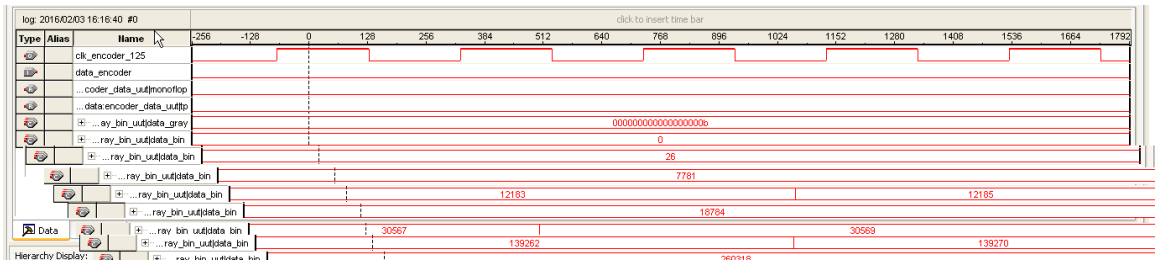
此类情形下，示波器检测的波形与 FPGA 实际接收到的波形并不相符。说明此情况下示波器不再适用 。建议使用 SignalTag 在线分析仪调试。

七 、数据输出

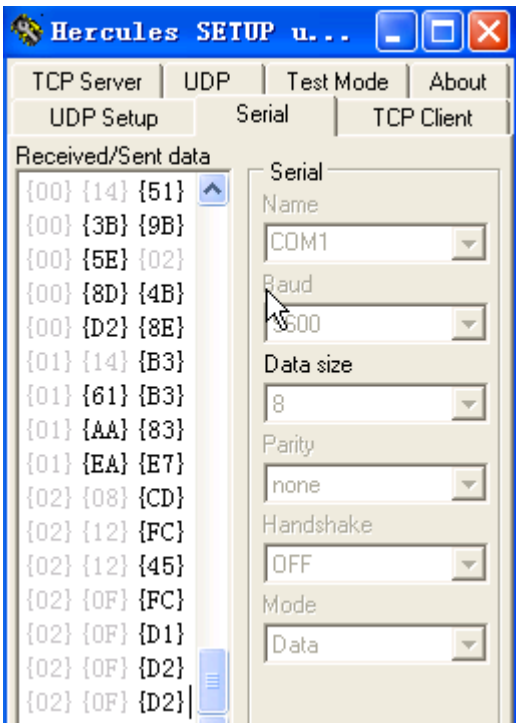
使用逻辑在线分析仪监控 data_bin 输出数据（以十进制显示）



转动编码器，观测输出变化



使用串口调试助手，转动编码器，观测其输出变化（输出数据为 16 进制）



附注：

系统特性（以下数据格式及频率都可以根据需要更改）：

采样时钟	20ns
编码器时钟频率	125KHz
tp_interval	56us
monoflop	48us
RS232 波特率	9600Bps
串口数据格式	8，n，1
串口数据输出频率	20HZ

参考文献

[1] Synchronous serial interface for absolute encoders -- SICK Stegmann
[2] Cyclone II Device Handbook -- Altera
[3] BJ-EPM240 串口通信实验 -- 吴厚航
[4] FPGA 开发实用教程 -- 徐方波 、 田耘
[5] SN75176A Differential Bus Transceiver -- TI