**2020年新工科联盟-Xilinx暑期学校团队项目设计文档**

**设计文稿提交格式**

**(Project Paper Submission Template)**

|  |  |
| --- | --- |
| **设计作品名称** | A01：电机控制 |
| **参赛队员姓名、学号、学校及所在院系** | 高初缘 06117108 东南大学 电子科学与工程学院  梅凯迪 06017142 东南大学 电子科学与工程学院 |
| **Github链接** |  |

**第一部分**

设计概述 /Design Introduction

（请概括地描述一下你的设计，包括设计目的、应用领域及适用范围等。撰写过程中应注重突出设计实现的主要/特色功能）

选题：A01电机控制

选题简介：通过USB转TTL,编码器（测电机转速），电机以及若干杜邦线实现PWM控制电机转动，同时使用编码器进行测速，将测速产生的脉冲通过杜邦线发送给sea,经过计算处理后再通过UART将处理所得数据传输给电脑端。

设计目的：在工业和日常生活中中有许许多多地方需要用到测量转速以及转向，比如汽车，飞机，各种家用电器都需要用到电机，而在其中引入电机测速这一模块可以实时监控该设备是否保持在正常工作状态

应用领域：物联网，智能家居

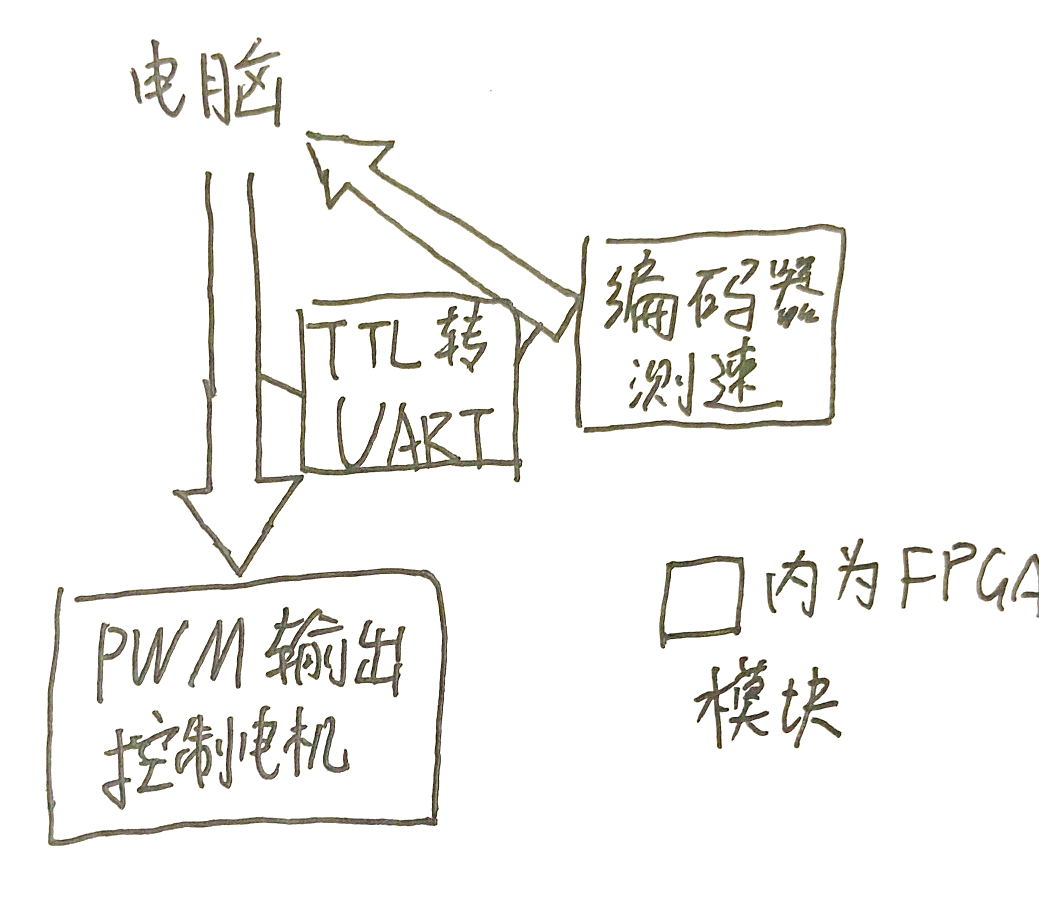
适用范围：由于所采用电机驱动电压较小，应用将仅限于家庭，如小型灌溉装置，抽水装置，自动窗帘，自动门及可以拓展到一些微波炉转盘，洗衣机和空调冰箱压缩机的转速监控，也可以继续开发如：一旦检测到电机转速达到某一速度或低于某一速度时，FPGA向该电机设备输入新的PWM波来改变其转速。

**第二部分**

系统组成及功能说明 /System Construction & Function Description

（请详细说明你作品要实现的所有功能以及如何组建系统以实现该功能，还包括为实现该功能需要用到的所有参数和所有操作的详细说明，必要的地方多用图表形式表述）

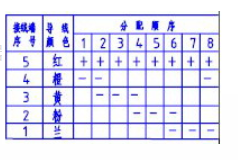
1.系统框图



本系统由三部分组成，通过TTL转UART的模块实现电脑控制PWM输出频率，从而控制电机的转动，编码器测电机的转速，并将数据传输给电脑，通过串口小助手显示测得的结果。

2、PWM输出控制电机输出

单片机控制步进电机的方法很简单，可以用IO口输出脉冲控制线圈电流驱动电机旋转，可分为四相四拍，四相八拍。四相电机，有四相四拍运行方式AB-BC-CD-DA-AB，四相八拍运行方式即 A-AB-B-BC-C-CD-D-DA-A。本系统中，我们采用了八拍运行方式来控制五线四相步进电机。



我们自己编写了电机输出控制模块，并进行了仿真，该模块的代码如下：

module STEP\_M(

input clk0,

output reg[3:0] out,

input rst

);

reg[2:0] current;

wire reset;

parameter s0=3'b000,s1=3'b001,s2=3'b010,

s3=3'b011,s4=3'b100,s5=3'b101,

s6=3'b110,s7=3'b111;

assign reset = ~rst;

always@(posedge clk0 or negedge reset)

begin

if(!reset)

begin

current <= s0;

out <= 4'b1001;

end

else begin

case(current)

s0:

begin

current <= s7;

out <= 4'b1000;

end

s1:

begin

current <= s0;

out <= 4'b1001;

end

s2:

begin

current <= s1;

out <= 4'b0001;

end

s3:

begin

current <= s2;

out <= 4'b0011;

end

s4:

begin

current <= s3;

out <= 4'b0010;

end

s5:

begin

current <= s4;

out <= 4'b0110;

end

s6:

begin

current <= s5;

out <= 4'b0100;

end

s7:

begin

current <= s6;

out <= 4'b1100;

end

endcase

end

end

endmodule

该代码s0到s7代表8个状态，每个状态对应一拍，对应out端一种情况的输出，通过GPIO接到电机的A、B、C、D四相上，电机的红线接到板子的5V上。

3、UART通信模块

UART作为异步[串口通信协议](https://baike.baidu.com/item/%E4%B8%B2%E5%8F%A3%E9%80%9A%E4%BF%A1%E5%8D%8F%E8%AE%AE)的一种，工作原理是将传输数据的每个字符一位接一位地传输。接口通过三个引脚从外部连接到其它设备。任何UART双向通信均需要至少两个引脚：接收数据输入引脚 (RX) 和发送数据引脚输出 (TX)：  
　　**RX：**接收数据输入引脚就是串行数据输入引脚。过采样技术可区分有效输入数据和噪声，从而用于恢复数据。  
　　**TX：**发送数据输出引脚。如果关闭发送器，该输出引脚模式由其 I/O 端口配置决定。如果使 能了发送器但没有待发送的数据，则 TX 引脚处于高电平。在单线和智能卡模式下，该 I/O 用于发送和接收数据（USART 电平下，随后在 SW\_RX 上接收数据）。

我们采用TTL电平标准，硬件上使用TTL转USB与电脑连接进行通信。

4、编码器测速模块

编码器的构成及测速原理：

编码器由一个中心有轴的光电码盘，其上有环形通、暗的刻线，有光电发射和接收器件读取，获得四组正弦波信号组合成A、B、C、D,每个正弦波相差90度相位差（相对于一个周波为360度），将C、D信号反向，叠加在A、B两相上，可增强稳定信号；另每转输出一个Z相脉冲以代表零位参考位。由于A、B两相相差90度，可通过比较A相在前还是B相在前，以判别编码器的正转与反转，通过零位脉冲，可获得编码器的零位参考位。

**第三部分**

完成情况及性能参数 /Final Design & Performance Parameters

完成情况：分模块完成作品（已实现的功能）：

1、电机模块完成情况

编写仿真代码，如下：

module test();

reg clk0;

reg rst;

wire[3:0] out;

initial

begin

clk0=0;

rst=0;

end

always #100

begin

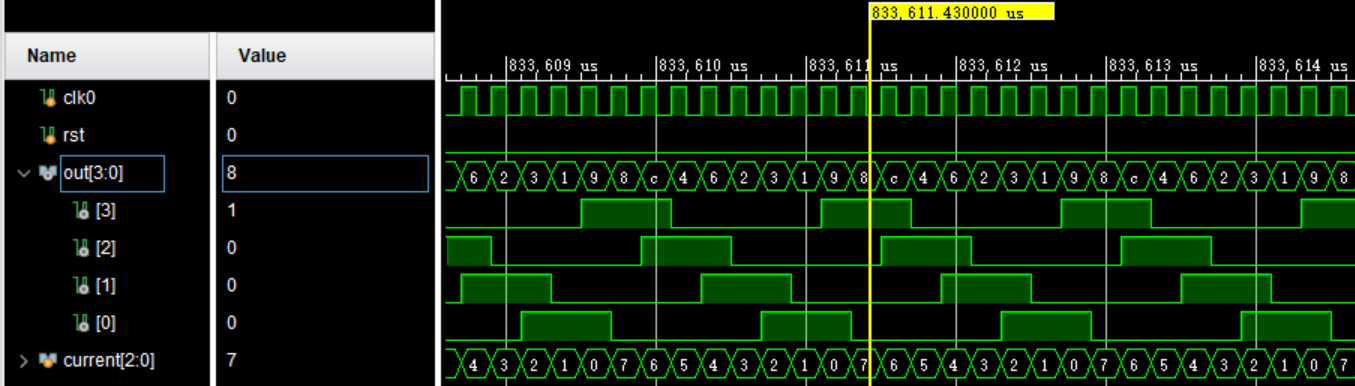
clk0 =~clk0;

end

STEP\_M test(.clk0(clk0),.rst(rst),.out(out));

endmodule

仿真图如下：



将电机模块例化，rst复位口接在一个按键上，实现了按键按下时电机转动，未按下时电机不转动，单一转速电机转动。

2、UART通信完成情况

UART通信模块各个模块代码：

时钟分频模块：

module Clk\_Division(

   input clk\_100MHz,

   input [30:0] clk\_mode,

   output clk\_out

   );

   reg Clk=0;

   reg flag=0;

   reg Is\_Odd=0;

   integer Count=0;

   //Half cycle flag update

   always @(negedge clk\_100MHz)

        begin

            Is\_Odd=clk\_mode[0];         //Determine whether the frequency division coefficient is an odd number

            if(Count==clk\_mode/2)

                flag=1;

            else

                flag=0;

        end

   //Frequency division count

   always @(posedge clk\_100MHz)

      begin

        if(Is\_Odd)

            if(Count==clk\_mode-1)

                begin

                    Count=0;

                    Clk=~Clk;

                end

            else if(Count==clk\_mode/2)

                begin

                    Count=Count+1;

                    Clk=~Clk;

                end

            else

                Count=Count+1;

       else

            if(Count==clk\_mode/2-1)

                begin

                     Count=0;

                     Clk=~Clk;

                end

            else

                Count=Count+1;

      end

   //Frequency divided clock output

   assign clk\_out=Clk|(flag&Is\_Odd);

endmodule

发送接收时序：

module UART\_Send(

    input clk\_10MHz,

    input Tx\_ACK,

    input Enable,

    input [15:0]Temp\_Data,

    input [15:0]Gyro\_Data\_X,

    input [15:0]Gyro\_Data\_Y,

    input [15:0]Gyro\_Data\_Z,

    input [15:0]Mag\_Data\_X,

    input [15:0]Mag\_Data\_Y,

    input [15:0]Mag\_Data\_Z,

    output reg Tx\_En=0,

    output reg[7:0]Send\_Buffer=0

    );

    //Defining the sending status

    parameter State\_Send\_Temp=0;

    parameter State\_Send\_Gyro=1;

    parameter State\_Send\_Mag=2;

    //Defining the sending status

    reg [1:0]Send\_State=0;

    reg [3:0]State\_Temp\_Cnt=0;

    reg [3:0]State\_Gyro\_Cnt=0;

    reg [3:0]State\_Mag\_Cnt=0;

    //Trigger pulse

    reg [1:0]Pulse\_Init\_Flag=0;

    //Give the rising edge of the sender

    always@(posedge clk\_10MHz or posedge Tx\_ACK)

        begin

            if(Tx\_ACK)

                begin

                    Tx\_En<=~Tx\_En;

                    Pulse\_Init\_Flag<=2;

                end

            else if(Pulse\_Init\_Flag==0)

                begin

                    Tx\_En<=0;

                    Pulse\_Init\_Flag<=1;

                end

            else if(Pulse\_Init\_Flag[0])

                begin

                    Tx\_En<=1;

                    Pulse\_Init\_Flag<=0;

                end

            else

                Tx\_En<=0;

        end

    //send data

    always@(posedge Tx\_ACK)

        begin

            case(Send\_State)

                State\_Send\_Temp:

                    begin

                        case(State\_Temp\_Cnt)

                            0:begin Send\_Buffer<=8'h55;State\_Temp\_Cnt<=State\_Temp\_Cnt+1;end

                            1:begin Send\_Buffer<=8'hAA;State\_Temp\_Cnt<=State\_Temp\_Cnt+1;end

                            2:begin Send\_Buffer<=8'h01;State\_Temp\_Cnt<=State\_Temp\_Cnt+1;end

                            3:begin Send\_Buffer<=Temp\_Data[15:8];State\_Temp\_Cnt<=State\_Temp\_Cnt+1;end

                            4:begin Send\_Buffer<=Temp\_Data[7:0];State\_Temp\_Cnt<=State\_Temp\_Cnt+1;end

                            5:begin Send\_Buffer<=8'hAA;State\_Temp\_Cnt<=0;Send\_State<=State\_Send\_Gyro;end

                        endcase

                    end

                State\_Send\_Gyro:

                    begin

                        case(State\_Gyro\_Cnt)

                            0:begin Send\_Buffer<=8'h55;State\_Gyro\_Cnt<=State\_Gyro\_Cnt+1;end

                            1:begin Send\_Buffer<=8'hAA;State\_Gyro\_Cnt<=State\_Gyro\_Cnt+1;end

                            2:begin Send\_Buffer<=8'h02;State\_Gyro\_Cnt<=State\_Gyro\_Cnt+1;end

                            3:begin Send\_Buffer<=Gyro\_Data\_X[15:8];State\_Gyro\_Cnt<=State\_Gyro\_Cnt+1;end

                            4:begin Send\_Buffer<=Gyro\_Data\_X[7:0];State\_Gyro\_Cnt<=State\_Gyro\_Cnt+1;end

                            5:begin Send\_Buffer<=Gyro\_Data\_Y[15:8];State\_Gyro\_Cnt<=State\_Gyro\_Cnt+1;end

                            6:begin Send\_Buffer<=Gyro\_Data\_Y[7:0];State\_Gyro\_Cnt<=State\_Gyro\_Cnt+1;end

                            7:begin Send\_Buffer<=Gyro\_Data\_Z[15:8];State\_Gyro\_Cnt<=State\_Gyro\_Cnt+1;end

                            8:begin Send\_Buffer<=Gyro\_Data\_Z[7:0];State\_Gyro\_Cnt<=State\_Gyro\_Cnt+1;end

                            9:begin Send\_Buffer<=8'hAA;State\_Gyro\_Cnt<=0;Send\_State<=State\_Send\_Mag;end

                        endcase

                    end

                State\_Send\_Mag:

                    begin

                        case(State\_Mag\_Cnt)

                            0:begin Send\_Buffer<=8'h55;State\_Mag\_Cnt<=State\_Mag\_Cnt+1;end

                            1:begin Send\_Buffer<=8'hAA;State\_Mag\_Cnt<=State\_Mag\_Cnt+1;end

                            2:begin Send\_Buffer<=8'h03;State\_Mag\_Cnt<=State\_Mag\_Cnt+1;end

                            3:begin Send\_Buffer<=Mag\_Data\_X[15:8];State\_Mag\_Cnt<=State\_Mag\_Cnt+1;end

                            4:begin Send\_Buffer<=Mag\_Data\_X[7:0];State\_Mag\_Cnt<=State\_Mag\_Cnt+1;end

                            5:begin Send\_Buffer<=Mag\_Data\_Y[15:8];State\_Mag\_Cnt<=State\_Mag\_Cnt+1;end

                            6:begin Send\_Buffer<=Mag\_Data\_Y[7:0];State\_Mag\_Cnt<=State\_Mag\_Cnt+1;end

                            7:begin Send\_Buffer<=Mag\_Data\_Z[15:8];State\_Mag\_Cnt<=State\_Mag\_Cnt+1;end

                            8:begin Send\_Buffer<=Mag\_Data\_Z[7:0];State\_Mag\_Cnt<=State\_Mag\_Cnt+1;end

                            9:begin Send\_Buffer<=8'hAA;State\_Mag\_Cnt<=0;Send\_State<=State\_Send\_Temp;end

                        endcase

                    end

               default:;

            endcase

        end

endmodule

接线如图所示：

3、编码器模块实现情况

我们查询了编码器的结构组成及相关原理，但由于选购的编码器种类错误，无法匹配我们的开发板，未能实现预期结果。

**第四部分**

总结 /Conclusions

1. 前期由于对相关知识认识不足，在未能充分了解电机，编码器工作原理的情况下仓促采购，在实际操作过程中才发现买的东西不合理如电机驱动电压过高等，浪费了大量的时间和精力，下次应该在充分论证项目的可行性后再开始购买项目所需材料。
2. 通过这次项目，我们充分认识到了同学之间相互合作的重要性，当一些问题解决不了，而又迟迟不能发现问题所在时，问问同学将是个不错的选择，因为很多问题同学也都遇到过，这样将大大减少自己浪费在一个小问题上的时间。就比如在这次的编码器仿真阶段，计数器计数波形没有问题，但是高电平时始终是X，询问了同学后才知道原来是因为在仿真阶段初始化的问题。
3. 要学会利用网上的资料，GitHub，Xilinx社区等网上论坛有许许多多有趣的项目，可以启发你的灵感，sea实验指导书的例程也足够让你掌握一些FPGA的基础操作，平常利用好这些资料，在你自己进行实践操作的时候可以事半功倍

最后感谢各个学校的老师们以及助教对我们孜孜不倦的教导，这次暑期学校给我们打开了FPGA世界的大门，让我们发现FPGA的世界是那么多姿多彩，也让我们看到了将我们脑海中的想法一一通过FPGA实现的可能性。