附录 D 数据采集控制模块 Verilog 源程序

● File name: Encoder.v //Verilog 源程序（文件名和 module 名保持一致）

```
`timescale 1ns / 1ns  //延时和仿真的基本时间单位，不可综合，下略去
module Encoder ( en_n, inChannelTrigger, outChannelAddress );
    input en_n;//此使能信号使得 Encoder 模块能组成 Array 实现多道编码
    input [15:0] inChannelTrigger;
    output [4:0] outChannelAddress;
    reg [4:0] outChannelAddress;

    always @ ( inChannelTrigger or en_n )
        begin
            if ( en_n )
                outChannelAddress = 5'b00000;
            else
                begin
                casex( inChannelTrigger )//使用 casex 语句，实现优先编码
                16'bxxxxxxxxxxxxxxx1   :   outChannelAddress = 5'b10000;
                16'bxxxxxxxxxxxxxx10   :   outChannelAddress = 5'b10001;
                16'bxxxxxxxxxxxxx100   :   outChannelAddress = 5'b10010;
                16'bxxxxxxxxxxxx1000   :   outChannelAddress = 5'b10011;
                16'bxxxxxxxxxxx10000   :   outChannelAddress = 5'b10100;
                16'bxxxxxxxxxx100000   :   outChannelAddress = 5'b10101;
                16'bxxxxxxxxx1000000   :   outChannelAddress = 5'b10110;
                16'bxxxxxxxx10000000   :   outChannelAddress = 5'b10111;
                16'bxxxxxxx100000000   :   outChannelAddress = 5'b11000;
                16'bxxxxxx1000000000   :   outChannelAddress = 5'b11001;
                16'bxxxxx10000000000   :   outChannelAddress = 5'b11010;
                16'bxxxx100000000000   :   outChannelAddress = 5'b11011;
                16'bxxx1000000000000   :   outChannelAddress = 5'b11100;
                16'bxx10000000000000   :   outChannelAddress = 5'b11101;
                16'bx100000000000000   :   outChannelAddress = 5'b11110;
                16'b1000000000000000   :   outChannelAddress = 5'b11111;
                default                :   outChannelAddress = 5'b00000;
                endcase
                end
        end
endmodule
```

● File name: EncodeArray.v

```
`timescale 1ns / 1ns
module EncodeArray ( inChannelTrigger,outChannelAddress, inLock ) ;
input inLock;
input [31:0] inChannelTrigger;
output [4:0] outChannelAddress;
wire [4:0] ChannelAddress1;
wire [4:0] ChannelAddress2;
wire [4:0] tmpAddress = {ChannelAddress2[4],ChannelAddress1[3:0] | ChannelAddress2[3:0]}; //将两个 Encoder 模块的输出合成 5bits 地址码
assign outChannelAddress = inLock ? outChannelAddress : tmpAddress;
//使用 inLock 信号锁住地址信号
Encoder U1      //Encoder 模块实例化
(   .inChannelTrigger(inChannelTrigger[15:0]),
    .outChannelAddress(ChannelAddress1),
    .en_n(1'b0)   );
Encoder U2
(   .inChannelTrigger(inChannelTrigger[31:16]),
    .outChannelAddress(ChannelAddress2),
    .en_n(ChannelAddress1[4])   );
endmodule
```

● File name: Ser2Par.v

```
module Ser2Par ( rst, clk, i_Conv_n, i_ConvOver, i_CS, i_MagData, i_AddData,
o_Data, o_DataFull );
    input rst;
    input clk;
    input i_Conv_n;         //A/D 转换开始，低电平有效
    input i_ConvOver;       //A/D 转换结束，表示串行数据输入结束
    input i_CS ;            //片选信号，高电平时输出数据，低电平时输出高阻
    input i_MagData;        //A/D 转换芯片 ADS7818 的串行幅值数据
    input [4:0] i_AddData;      //5 bits 的地址数据
    output [12:0] o_Data ;      //13 位并行输出，高 5 位为地址，低 8 位为幅值
    output o_DataFull ;         //并行输出数据 Ready 的标志信号
    reg r_SerDataIn; //串行数据输入开始标志寄存器
    reg r_DataFull ;  //o_DataFull 的输出寄存器
    reg [11:0] r_MagData;       //幅值数据的并行寄存器
    reg [3:0] r_ShiftState;     //状态寄存器
    reg tmpCS, r_CS;            //片选信号两级缓冲，保证数据的有效时间

    parameter
    SH0 = 4'b0000, SH1 = 4'b0001, SH2 = 4'b0010, SH3 = 4'b0011, SH4 =
    4'b0100,SH5 = 4'b0101, SH6 = 4'b0110, SH7 = 4'b0111, SH8 = 4'b1000,
    SH9 = 4'b1001, SH10 = 4'b1010, SH11 = 4'b1011, SH12 = 4'b1100;
    //状态机的状态值

    always @ ( rst or i_Conv_n or i_ConvOver )
        if ( rst | i_ConvOver )
            r_SerDataIn <= 1'b0;
        else if ( ~i_Conv_n )
            r_SerDataIn <= 1'b1;//串行数据输入标志位的置位和复位

    always @ ( posedge clk or posedge rst )
        if ( rst )
            begin
                r_DataFull <= 0;
                r_MagData <= 12'b000000000000;
                r_ShiftState <= SH0;
            end
        else if ( r_CS )
            begin    //片选信号到来，复位 r_DataFull 和状态机存器
                r_DataFull <= 1'b0;
                r_ShiftState <= SH0;
            end
        else if ( r_SerDataIn )     //状态机读入串行幅值数据到并行寄存器
            begin
                case (r_ShiftState)
                    SH0:
                        begin
                            r_ShiftState <= SH1;
                        end
                    SH1:
                        begin
                            r_MagData[11] <= i_MagData;
                            r_ShiftState <= SH2;
                        end
                    SH2:
                        begin
                            r_MagData[10] <= i_MagData;
                            r_ShiftState <= SH3;
                        end
                    SH3:
                        begin
                            r_MagData[9] <= i_MagData;
                            r_ShiftState <= SH4;
                        end
```

```
SH4:
    begin
        r_MagData[8] <= i_MagData;
        r_ShiftState <= SH5;
    end
SH5:
    begin
        r_MagData[7] <= i_MagData;
        r_ShiftState <= SH6;
    end
SH6:
    begin
        r_MagData[6] <= i_MagData;
        r_ShiftState <= SH7;
    end
SH7:
    begin
        r_MagData[5] <= i_MagData;
        r_ShiftState <= SH8;
    end
SH8:
    begin
        r_MagData[4] <= i_MagData;
        r_ShiftState <= SH9;
    end
SH9:
    begin
        r_MagData[3] <= i_MagData;
        r_ShiftState <= SH10;
    end
SH10:
```

```
                        begin
                            r_MagData[2] <= i_MagData;
                            r_ShiftState <= SH11;
                        end
                    SH11:
                        begin
                            r_MagData[1] <= i_MagData;
                            r_ShiftState <= SH12;
                        end
                    SH12:
                        begin
                            r_MagData[0] <= i_MagData;
                            r_ShiftState <= SH0;
                            r_DataFull <= 1;
                        end
                    default: r_ShiftState <= SH0;
                endcase
            end

    always @ ( posedge clk ) //片选信号两级缓冲，保证数据的有效时间
        begin
            tmpCS <= i_CS;
            r_CS <= tmpCS;
        end

    assign o_DataFull = r_DataFull;
    assign o_Data = i_CS ? { i_AddData, r_MagData[11:4] }:
    13'bzzzzzzzzzzzzz ; //片选信号有效时输出有效，反之输出高阻
                        //为保证精度和方便设计，取 12 位幅值数据高 8 位
endmodule
```

● File name: Counter.v

```
module Counter (rst, clk, inCountEn, outConv_n, outConvOver);
    input rst;
    input clk;
    input inCountEn;      //计数器使能，开始计数
    output outConv_n;     //A/D 转换启动信号，低电平有效
    output outConvOver; //A/D 转换结束信号
    reg [4:0] regCount;   //计数数值寄存器

    always @(posedge clk or posedge rst)
        begin
            if (rst | ~inCountEn)
                regCount <= 5'b00000;
            else
                regCount <= regCount + 1;
        end
    assign outConv_n = regCount[4] | regCount[3] | ~regCount[2] | regCount[1] |
    regCount[0];      //保证 A/D 芯片 4 个时钟的采样时间后启动 A/D 转换
    assign outConvOver = regCount[4] & ~regCount[3] & ~regCount[2] &
    ~regCount[1] & regCount[0];      //计数为 17 的时候，结束 A/D 转换
endmodule
```

- File name: AdHost.v

```
module AdHost ( inPeakArrive, inConv_n, inConvOver, rst, clk, outCountEn,
outDischarge );
    input inPeakArrive;
    input inConv_n;
    input inConvOver;
    input rst;
    input clk;
    output outDischarge;
    output outCountEn;
    reg outCountEn;
    reg outDischarge;
    reg [2:0] fsmState;
    parameter
    IDLE = 3'b001,   SAMPLE = 3'b010, DISCHARGE = 3'b100;
    //状态机的状态值

    always @ ( posedge clk or posedge rst )
        if ( rst )
            begin
                outCountEn <= 0;
                outDischarge <= 0;
                fsmState <= IDLE;
            end
        else
            begin
                case(1'b1)
                fsmState[0]:
                    if ( inPeakArrive )
                        begin
                            outCountEn <= 1;
                            fsmState <= SAMPLE;
                        end
                fsmState[1]:
                    if ( ~inConv_n )
                        begin
                            outDischarge <= 1;
                            fsmState <= DISCHARGE;
                        end
                fsmState[2]:
                    if ( inConvOver )
                        begin
```

```
                              outDischarge <= 0;
                              outCountEn <= 0;
                              fsmState <= IDLE;
                         end
                   default fsmState <= IDLE;
                   endcase
              end
 endmodule
```

- File name: AdConvertor.v

```
module AdConvertor ( clk, outMagData, inConv_n );
    input clk;
    input inConv_n;             //ADS7818 的 CONV 输入
    output outMagData;  //ADS7818 的 DATA 输出
    reg outMagData;
    reg [3:0] regCycleNum;
    reg[11:0] tmpData = 12'b011010010010;    //任意预置一个转换的结果
    initial outMagData =1'bz; //初始化输出

    always @ (negedge clk )
    begin
         if ( ~inConv_n )
              begin
                   #20 outMagData = 0;
                   taskShiftOut;      //如果 inConv_n 为低，调用结果输出任务
              end
         else outMagData = 1'bz;
    end

    task taskShiftOut;      //按 clk 输出预置的 A/D 转换结果，12 位串行数据
    begin
              tmpData = (tmpData+3'd5);
```

```
                @ (negedge clk ) #30 outMagData = tmpData[11];
                @ (negedge clk ) #30 outMagData = tmpData[10];
                @ (negedge clk ) #30 outMagData = tmpData[9];
                @ (negedge clk ) #30 outMagData = tmpData[8];
                @ (negedge clk ) #30 outMagData = tmpData[7];
                @ (negedge clk ) #30 outMagData = tmpData[6];
                @ (negedge clk ) #30 outMagData = tmpData[5];
                @ (negedge clk ) #30 outMagData = tmpData[4];
                @ (negedge clk ) #30 outMagData = tmpData[3];
                @ (negedge clk ) #30 outMagData = tmpData[2];
                @ (negedge clk ) #30 outMagData = tmpData[1];
                @ (negedge clk ) #30 outMagData = tmpData[0];
                @ (negedge clk ) #30 outMagData = 1'bz;
                @ (negedge clk ) #70 outMagData = 1'bz;
            end
    endtask
endmodule
```

- File name: SigGen.v

```
module SigGen ( inDischarge, clk, rst, outChannelTrigger, outPeakArrive, o_CS,
i_DataFull);
    input inDischarge;     //模拟放电输入信号
    input i_DataFull; //模拟接口电路的数据 Ready 输入信号
    output clk;
    output rst;
    output [31:0] outChannelTrigger;  //模拟通道触发信号
    output outPeakArrive;      //模拟脉冲峰值到达信号
    output o_CS;     //模拟接口电路的片选信号
    reg o_CS;
    reg clk;
    reg rst;
    reg [31:0] outChannelTrigger;
reg outPeakArrive;

//模拟第一个输入脉冲产生，触发，峰值到达
initial
    begin
    clk = 0;
    outPeakArrive = 0;
    o_CS = 0;
    outChannelTrigger = 32'b00000000000000000000000000000000;
    #100 rst = 1;
    #300 rst = 0;
    #200 outChannelTrigger = 32'b00000000000000000000000010000001;
    #200 outPeakArrive = 1;
    end

always #50   clk = ~clk;   //生成 10M 时钟
always #5000            //周期性输入脉冲产生，触发，峰值到达
    begin
        #200 outChannelTrigger = (outChannelTrigger<<1);
        #200 outPeakArrive = 1;
    end

//模拟放电输入信号跳高，电容放电，峰值到达信号跳低
always @ ( posedge clk )
    begin
    if ( inDischarge )
        #100 outPeakArrive <= 0;
    else
        outPeakArrive <= outPeakArrive;
    end
    //模拟数据 Ready 时，输出片选信号读取数据
    always @ ( posedge i_DataFull )
        begin
            o_CS = 1;
            #300 o_CS = 0;
        end
endmodule
```

● File name: UsbCtrler.v //数据排序和重组模块

```
module UsbCtrler (clr, i_Read, i_Data, i_DataFull, o_ParDataRDY, o_CS, o_ParData );
     input clr;   //clear 信号，在每次读完数据时复位寄存器
     input i_Read ;
     input [12:0] i_Data;      //8 路采集电路输出数据总线
     input [7:0] i_DataFull;
     output [7:0] o_CS ;             //采集电路的片选信号
     output o_ParDataRDY;        //USB 接口芯片数据有效
     output [15:0] o_ParData;//USB 接口芯片数据

     reg [11:0] r_CtrlData;    //控制数据，最高位为 o_ParDataRDY 标志位
                               //[10:8]位为数据有效的采集电路的地址码
                               //低 8 位为对数据有效的采集电路的片选信号
     always @ ( i_DataFull or clr )
        begin
            if ( clr )
                r_CtrlData = 12'b000000000000;
            else
                begin
                casex( i_DataFull )   //使用优先编码，实现输入数据排序
                    8'bxxxxxxx1:       r_CtrlData = 12'b100000000001;
                    8'bxxxxxx10:       r_CtrlData = 12'b100100000010;
                    8'bxxxxx100:       r_CtrlData = 12'b101000000100;
                    8'bxxxx1000:     r_CtrlData = 12'b101100001000;
                    8'bxxx10000:     r_CtrlData = 12'b110000010000;
                    8'bxx100000:     r_CtrlData = 12'b110100100000;
                    8'bx1000000:     r_CtrlData = 12'b111001000000;
                    8'b10000000:     r_CtrlData = 12'b111110000000;
                    default   :     r_CtrlData = 12'b000000000000;
                endcase
                end
            end

    assign   o_CS = i_Read ? o_CS : r_CtrlData[7:0] ;
    assign   o_ParData = i_Read ? o_ParData : {r_CtrlData[10:8], i_Data};
    assign   o_ParDataRDY =   i_Read ? 1'b0 : r_CtrlData[11];
endmodule
```

● File name: MCU.v //USB 接口芯片模块(芯片中的 MCU 实现数据读取)

```
module MCU ( i_ParDataRDY, i_ParData, o_Read, clr);
    input i_ParDataRDY;
    input [15:0] i_ParData;
    output o_Read, clr;
    reg o_Read, clr;
    reg [15:0] r_ParData;

    initial   //信号初始化
        begin
            o_Read = 0;
            clr = 0;
            #20 clr = 1;
            #10 clr = 0;
        end

    //模拟 USB 接口芯片在数据有效时读取数据
    always    @ (i_ParDataRDY)
        while (i_ParDataRDY)
            begin
            #10 o_Read = 1;
                r_ParData = i_ParData;
            #40 o_Read = 0;
            end
endmodule
```

● File name: DataAqr.v   //模拟 8 路信号采集电路

```
module DataAqr( o_DataFull, i_CS, o_Data );
    input [7:0] i_CS;
    output [7:0] o_DataFull;
    reg [7:0] o_DataFull;
    reg [7:0] tmpDataFull;
    output [12:0] o_Data;
    reg [12:0] o_Data;

    //任意设定了 8 路信号采集电路的不同的输出数据
    parameter
        Data0=13'b0000000000001,
        Data1=13'b0000000000010,
        Data2=13'b0000000000100,
        Data3=13'b0000000001000,
        Data4=13'b0000000010000,
        Data5=13'b0000000100000,
        Data6=13'b0000001000000,
        Data7=13'b0000010000000;

    initial    //信号初始化
        begin
            o_DataFull = 8'b00000000;
            #40 tmpDataFull = 8'b00010000;
                o_DataFull = tmpDataFull;
        end

    always #1000
        begin
            tmpDataFull = (tmpDataFull<<1);
            o_DataFull = tmpDataFull;
        end

    //模拟片选信号到来，输出对应路的信号采集电路的数据
    always @( i_CS )
        begin
            case (1'b1)
                i_CS[0]:
                    begin
                        o_Data=Data0;
                        #50 o_DataFull[0]=0;
                    end
                i_CS[1]:
                    begin
                        o_Data=Data1;
                        #50 o_DataFull[1]=0;
                    end
                i_CS[2]:
                    begin
                        o_Data=Data2;
                        #50 o_DataFull[2]=0;
                    end
                i_CS[3]:
                    begin
                    o_Data=Data3;
                    #50 o_DataFull[3]=0;
                end
            i_CS[4]:
                begin
                    o_Data=Data4;
                    #50 o_DataFull[4]=0;
                end
            i_CS[5]:
                begin
                    o_Data=Data5;
                    #50 o_DataFull[5]=0;
                end
            i_CS[6]:
                begin
                    o_Data=Data6;
                    #50 o_DataFull[6]=0;
                end
            i_CS[7]:
                begin
                    o_Data=Data7;
                    #50 o_DataFull[7]=0;
                end
            default:
                   o_Data=13'bzzzzzzzzzzzzz;
        endcase
      end
    endmodule
```

个人简历、在学期间发表的学术论文与研究成果

个人简历

1981 年 6 月 21 日出生于浙江省常山县。

1998 年 9 月考入清华大学工程物理系工程物理专业，2002 年 7 月本科毕业并获得工学学士学位。

2002 年 9 月免试进入清华大学工程物理系攻读工程物理专业工学硕士至今。

发表的学术论文

[1] Zhijia, Yuan, Yongjie, Jin, High Performance Data Acquisition Methods Based on Waveform Digitization, 2004 IEEE NSS Conference Record, 2004