# 姿态传感器功能测试设计

### 实验任务

任务：基于STEP-MAX10M08核心板和STEP电赛训练板底板，完成姿态传感器功能测试设计并观察调试结果

要求：驱动底板上的姿态传感器MMA7660进行驱动激活和读数据，并用读取的数据在OLED上显示。

解析：通过FPGA编程驱动姿态传感器MMA7660，进行驱动激活和读数据，然后再从中读取数据，然后经过BCD转码处理并显示到OLED显示屏上。

### 实验目的

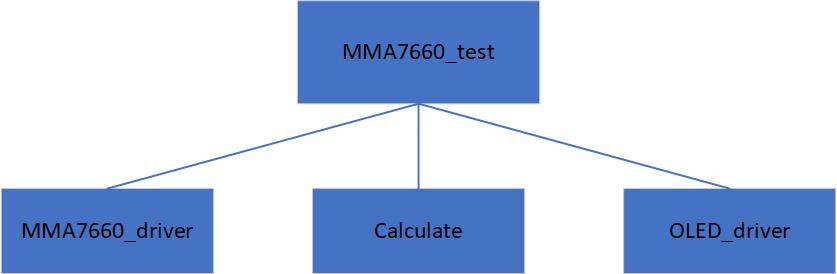
前面的章节中我们学习了OLED模块和BCD转码模块的工作原理及驱动方法，也对I2C总线协议及相关知识，本实验主要对I2C总线驱动方法加以练习，同时熟悉MMA7660芯片的读写操作和FPGA设计中常用运算方法，最终完成姿态传感器功能测试总体设计。

* 复习I2C总线工作原理及通信协议
* 练习I2C接口驱动设计方法，完成姿态传感器MMA7660驱动设计
* 完成OLED显示屏驱动设计
* 完成姿态传感器功能测试总体设计

### 设计框图

根据前面的实验解析我们可以得知，该设计可以拆分成三个功能模块实现，

* MMA7660\_driver：姿态传感器MMA7660 I2C总线通信驱动模块。
* Calculate：将得到的二进制数码转换成BCD码的计算模块。
* OLED\_driver：将得到的数据在OLED显示屏上显示。

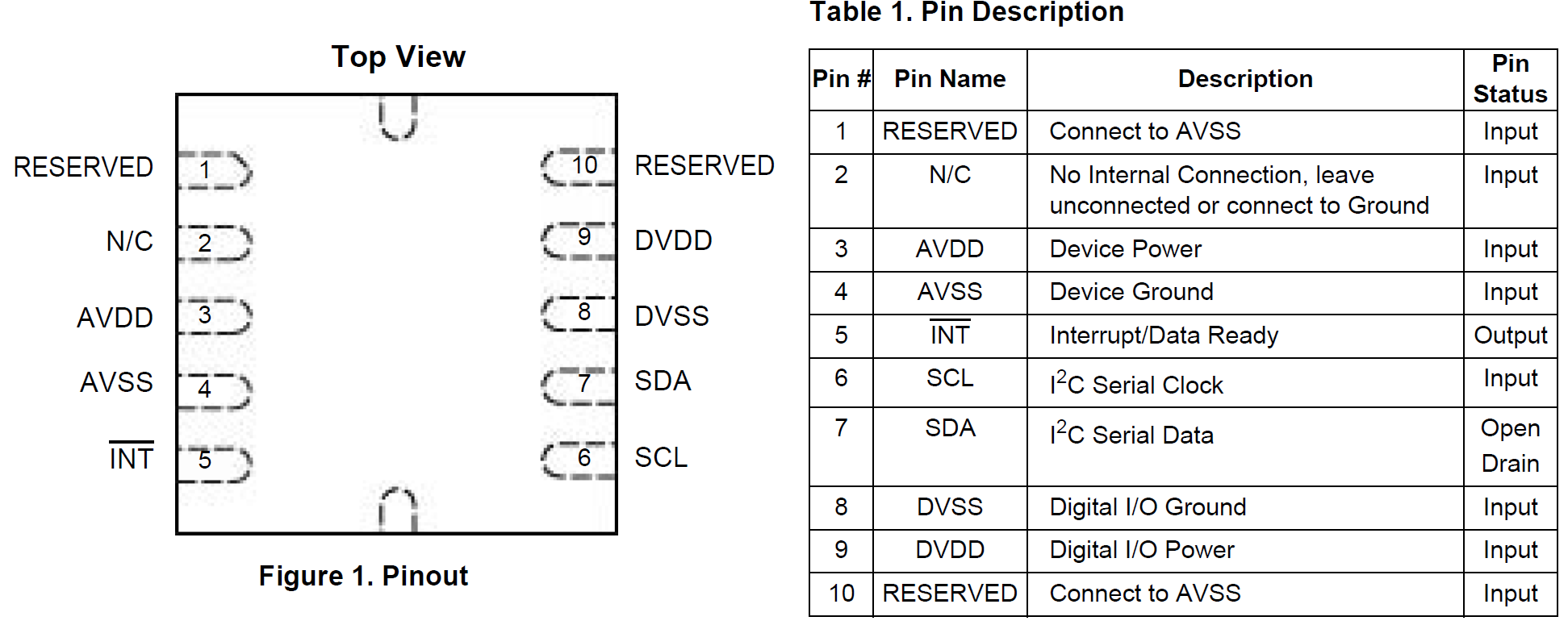


Top-Down层次设计

### 实验原理

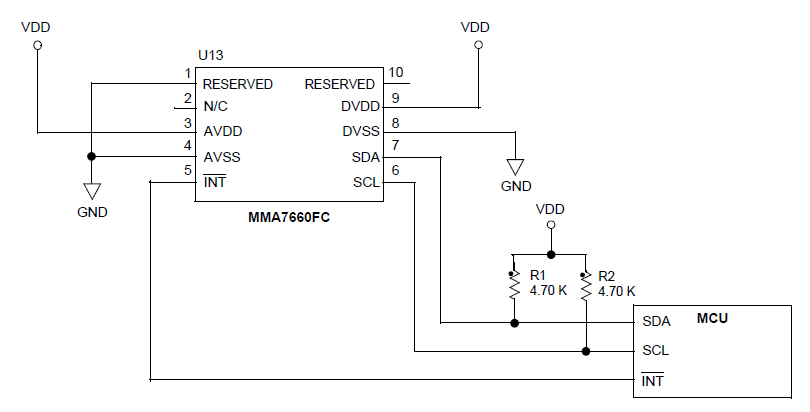
#### MMA7660芯片介绍

MMA7660FC是具有数字输出的I²C、低功耗、紧凑型电容式微机械加速度传感器，提供低通滤波器、零重力加速度偏移和增益误差补偿，并可以转化为6位数字值，用户可配置输出数据的传输速率。该器件可通过中断引脚(INT)识别传感器的数据变化、产品的朝向和姿态等。MMA7660FC采用非常小的3 mm x 3 mm x 0.9 mm DFN封装。完全兼容I2C总线，管脚功能描述如下。



管脚功能描述

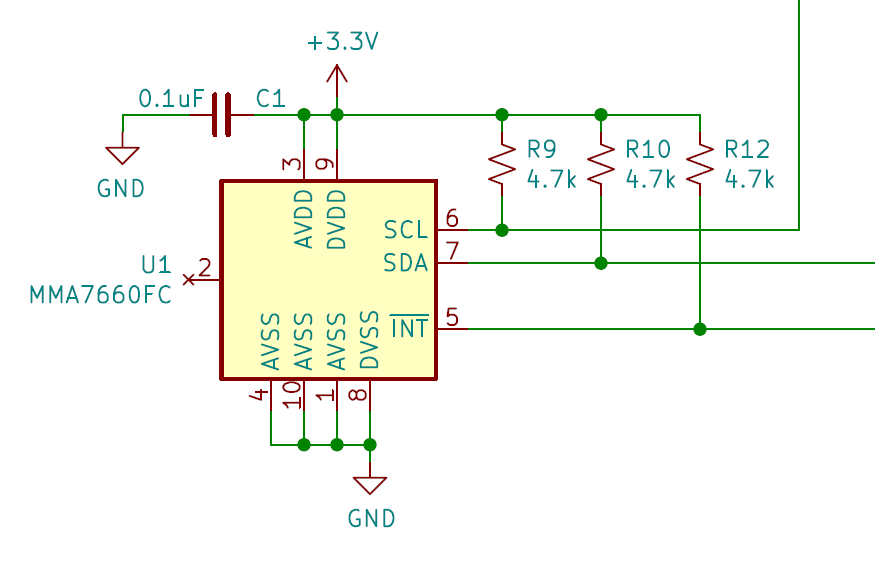
MMA7660芯片典型电路连接如下：



典型电路连接

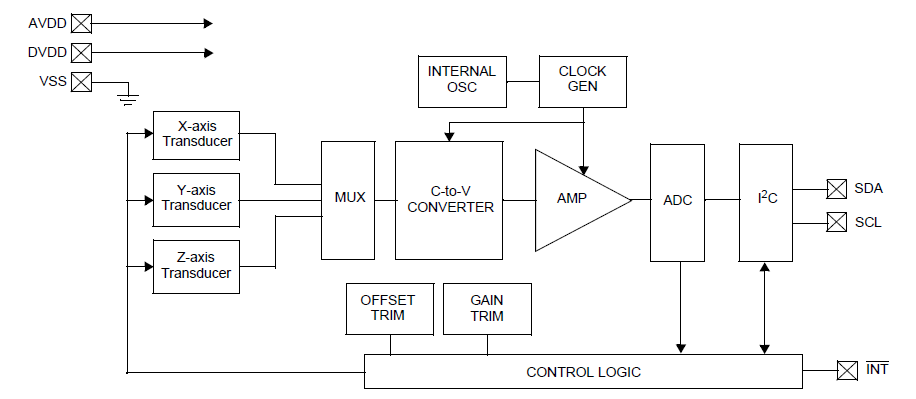
##### MMA7660芯片连接

STEP 电赛训练板底板上的MMA7660模块电路图如下



MMA7660芯片电路

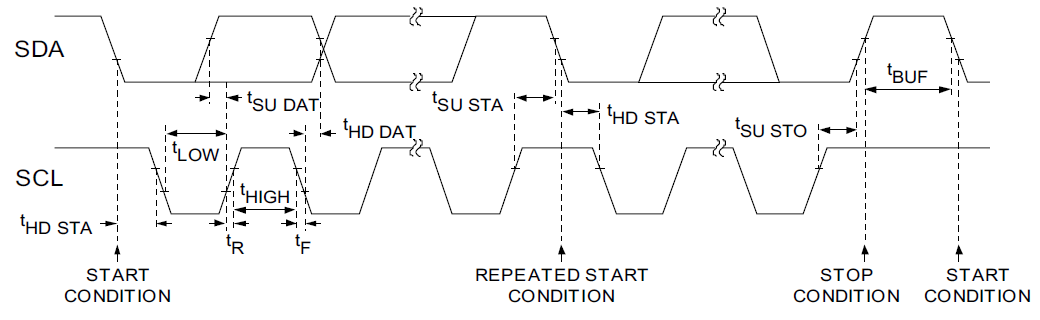
上图为姿态传感器MMA7660模块电路，与FPGA硬件接口有I2C总线（SCL、SDA），MMA7660具有可配置每秒采样率1到120次；低功耗的自动唤醒/休眠功能；低功耗:  关机模式：0.4 µA 待机模式：2 µA 工作模式：可配置，低至47 µA；工作电压：2.4V – 3.6V等特性。



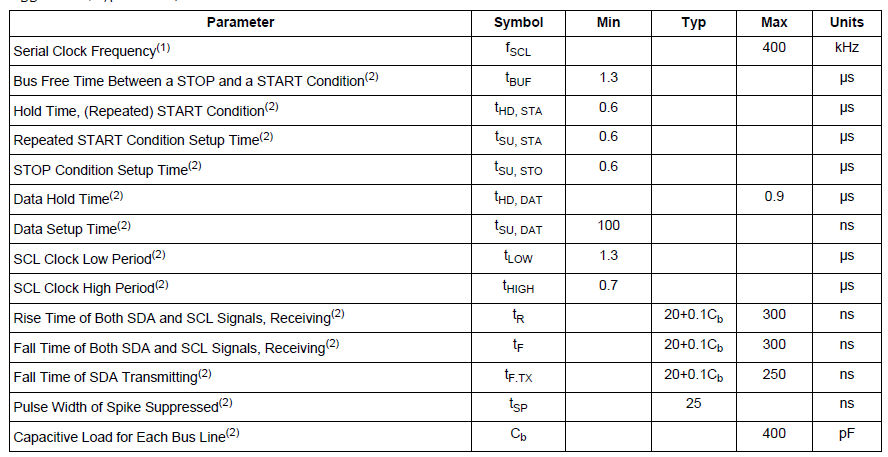
MMA7660内部框图

##### MMA7660驱动设计

通过前面的了解，我们对于整个I2C总线的驱动原理有了一定的了解，接下来我们根据MMA7660的芯片手册了解其驱动方法及参数要点。



MMA7660时序图



MMA7660时序参数

通过MMA7660时序参数了解，MMA7660最高支持I2C通信400KHz。

* 分频得到400KHz的时钟，程序实现同智能接近系统设计实验。

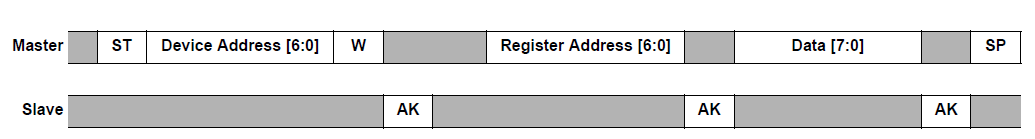
I2C时序基本单元（启动、停止、发送、接收、发应答、读应答）协议里统一的，所以基本单元状态的设计也是不需要调整的。

* 启动时序状态设计程序实现同智能接近系统设计实验。
* 发送单元和读应答单元合并，时序状态设计程序实现同智能接近系统设计实验。
* 接收单元和写应答单元合并，时序状态设计程序实现同智能接近系统设计实验。
* 停止时序状态设计程序实现同智能接近系统设计实验。

MMA7660出厂芯片器件地址为7位0x4C

对于写操作而言，分为两种，单字节写和多字节写，本实验采用了单字节写；对于读操作而言，则有单字节读和多字节读两种，本实验采用了单字节读方式完成。

单字节写操作时序流程如下：



单字节写

我们将这种操作设计成一个一个状态，程序实现如下：

MODE1:begin *//单次写操作*

      if(cnt\_mode1 >= 3'd5)

      cnt\_mode1 <= 1'b0; *//对START中的子状态执行控制cnt\_start*

      else cnt\_mode1 <= cnt\_mode1 + 1'b1;

      state\_back <= MODE1;

      case(cnt\_mode1)

       3'd0: begin state <= START; end *//I2C通信时序中的START*

       3'd1: begin data\_wr <= dev\_addr; state <= WRITE; end *//设备地址*

       3'd2: begin data\_wr <= reg\_addr; state <= WRITE; end *//寄存器地址*

       3'd3: begin data\_wr <= `WRITE\_DATA; state <= WRITE; end *//写入数据*

       3'd4: begin state <= STOP; end *//I2C通信时序中的STOP*

       3'd5: begin state <= MAIN; end *//返回MAIN*

       default: state <= IDLE; *//如果程序失控，进入IDLE自复位状态*

endcase

end

单字节读操作时序流程如下：

MODE2:begin *//随机读操作*

if(cnt\_mode2 >= 4'd8)

cnt\_mode2 <= 1'b0; *//对START中的子状态执行控制cnt\_start*

else

cnt\_mode2 <= cnt\_mode2 + 1'b1;

state\_back <= MODE2;

case(cnt\_mode2)

4'd0: begin state <= START; end *//I2C通信时序中的START*

4'd1: begin data\_wr <= dev\_addr; state <= WRITE; end *//设备地址*

4'd2: begin data\_wr <= reg\_addr; state <= WRITE; end *//寄存器地址*

4'd3: begin state <= START; end *//I2C通信时序中的START*

4'd4: begin data\_wr <= dev\_addr\_r; state <= WRITE; end *//设备地址*

4'd5: begin ack <= NACK; state <= READ; end *//读寄存器数据*

4'd6: begin seg\_data <= data\_r; end

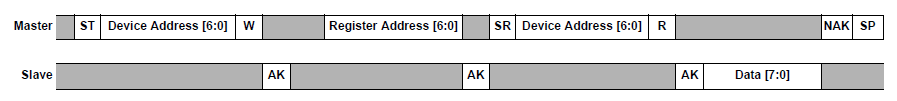
4'd7: begin state <= STOP; end *//I2C通信时序中的STOP*

4'd8: begin state <= MAIN; end *//返回MAIN*

default: state <= IDLE; *//如果程序失控，进入IDLE自复位状态*

endcase

end



单字节读

同理，程序实现如下

最后我们编程控制状态机按照驱动例程代码中流程运行，程序实现如下：

 MAIN:begin

      if(cnt\_main >= 4'd8)

      cnt\_main <= 4'd2;   *//写完控制指令后循环读数据*

      else

      cnt\_main <= cnt\_main + 1'b1;

      case(cnt\_main)

       4'd0: begin dev\_addr <= 8'h98; reg\_addr <= 8'h07; reg\_wr <= 8'h01; state <= MODE1; end *//写入数据*

       4'd1: begin num\_delay <= 24'd2000; state <= DELAY; end *//5ms延时*

       4'd2: begin dev\_addr <= 8'h98; dev\_addr\_r <= 8'h99; reg\_addr <= 8'h01; state <= MODE2; end *//读取数据*

       4'd3: begin xout <= read\_data; end

       4'd4: begin dev\_addr <= 8'h98; dev\_addr\_r <= 8'h99; reg\_addr <= 8'h02; state <= MODE2; end *//读取数据*

4'd5: begin yout <= read\_data; end

       4'd6: begin dev\_addr <= 8'h98; dev\_addr\_r <= 8'h99; reg\_addr <= 8'h03; state <= MODE2; end *//读取数据*

       4'd7: begin zout <= read\_data; end *//读取数据*

       4'd8: begin num\_delay <= 32'd48000; state <= DELAY; end *//120ms延时*

       default: state <= IDLE; *//如果程序失控，进入IDLE自复位状态*

      endcase

     end

##### 系统总体实现

二进制转BCD码模块和OLED显示模块我们前面基础数字电路实验中详细介绍过，这里我们直接调用。

下面将三个模块例化：

综合后的设计框图如下：

 MMA7660\_driver U\_MMA7660\_driver

 (

 .clk   (clk   ), *//系统时钟*

 .rst\_n  (rst\_n   ), *//系统复位，低有效*

 .i2c\_scl  (i2c\_scl  ), *//I2C总线SCL*

 .i2c\_sda  (i2c\_sda  ), *//I2C总线SDA*

 .xout   (xout   ), *//读取数据*

 .yout   (yout   ), *//读取数据*

 .zout   (zout   ) *//读取数据*

 );

 Calculate U\_Calculate

 (

 .rst\_n (rst\_n),

 .xout  (xout),

 .yout  (yout),

 .zout  (zout),

 .x\_data\_bcd  (x\_data\_bcd),

 .y\_data\_bcd  (y\_data\_bcd),

 .z\_data\_bcd  (z\_data\_bcd)

 );

 OLED\_driver U\_OLED\_driver

 (

 .clk   (clk   ), *//系统时钟*

 .rst\_n  (rst\_n  ), *//系统复位，低有效*

 .freq\_100 (freq\_100 ),

 .freq\_10  (freq\_10  ),

 .freq\_1  (freq\_1  ),

 .dc1\_10  (dc1\_10  ),

 .dc1\_1  (dc1\_1  ),

 .dc1\_01  (dc1\_01  ),

 .dc2\_10  (dc2\_10  ),

 .dc2\_1  (dc2\_1  ),

 .dc2\_01  (dc2\_01  ),

 .x\_data\_bcd (x\_data\_bcd ),

 .y\_data\_bcd (y\_data\_bcd ),

 .z\_data\_bcd (z\_data\_bcd ),

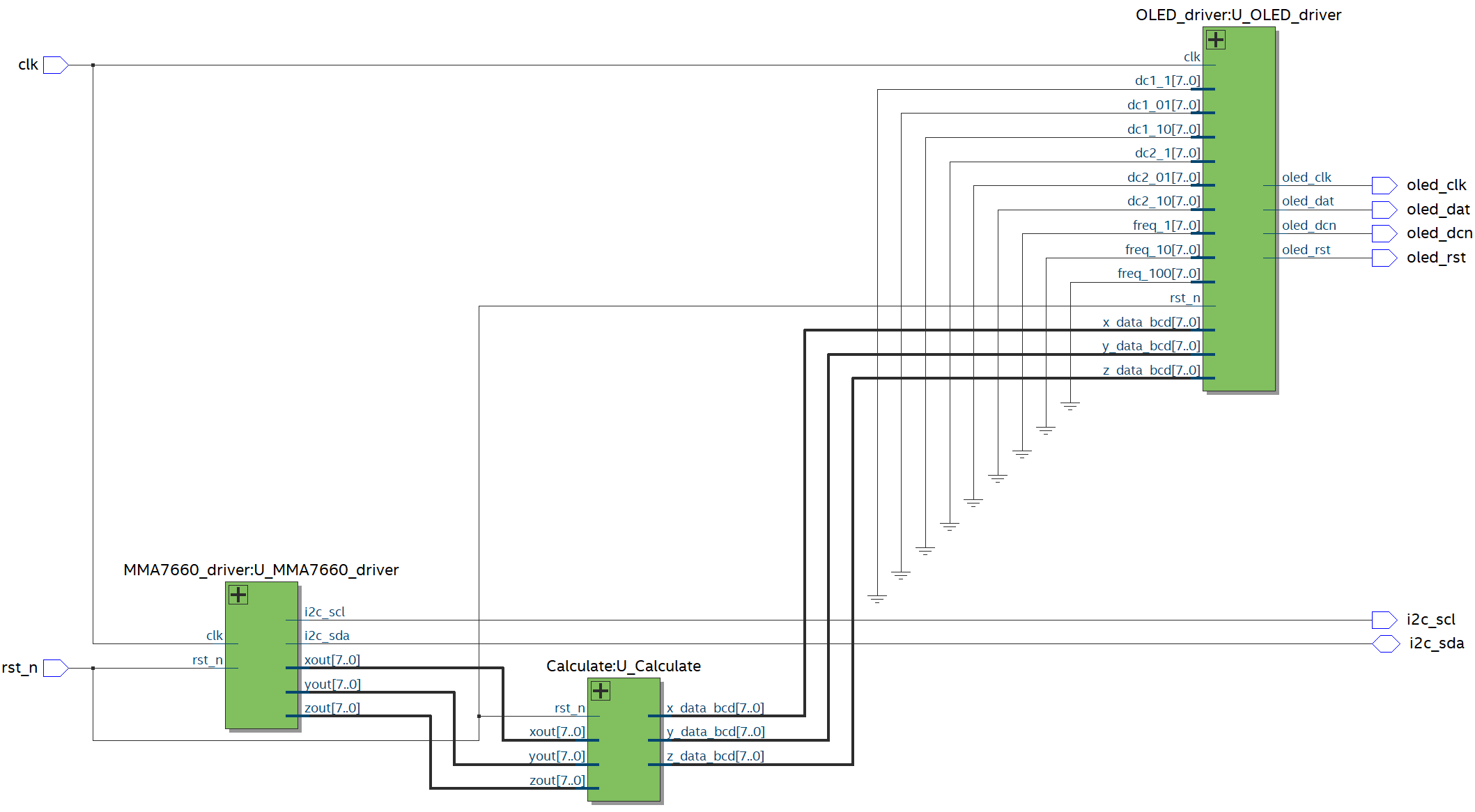
 .oled\_rst (oled\_rst ),

 .oled\_dcn (oled\_dcn ),

 .oled\_clk (oled\_clk ),

 .oled\_dat (oled\_dat )

 );

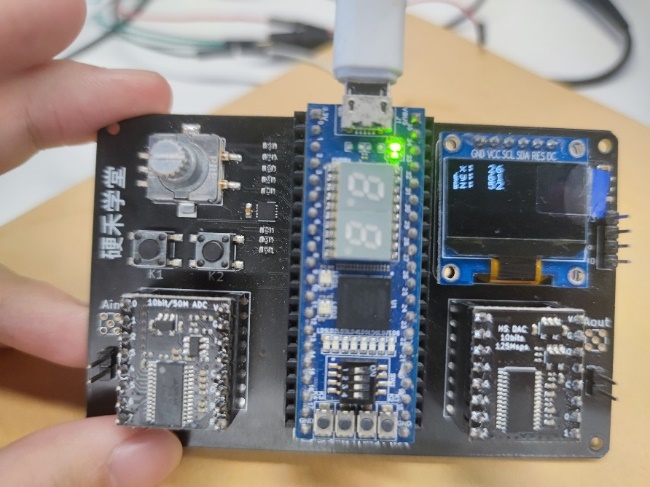


### 实验步骤

1. 双击打开Quartus Prime工具软件；
2. 新建工程：File → New Project Wizard（工程命名，工程目录选择，设备型号选择，EDA工具选择）；
3. 新建文件：File → New → Verilog HDL File，键入设计代码并保存；
4. 设计综合：双击Tasks窗口页面下的Analysis & Synthesis对代码进行综合；
5. 管脚约束：Assignments → Assignment Editor，根据项目需求分配管脚；
6. 设计编译：双击Tasks窗口页面下的Compile Design对设计进行整体编译并生成配置文件；
7. 程序烧录：点击Tools → Programmer打开配置工具，Program进行下载；
8. 观察设计运行结果。

### 实验现象

将程序加载到FPGA，变换装置的空间姿态，观察OLED显示变化。



# ADDA功能测试设计

### 实验任务

任务：基于STEP-MAX10M08核心板和STEP电赛训练板底板，完成ADDA功能测试设计并观察调试结果

要求：驱动底板上的数模转换器3PD5651E和模数转换器3PA1030进行信号传输。

解析：通过FPGA编程驱动的数模转换器3PD5651E和模数转换器3PA1030，把模拟信号先转换成数字信号给到核心板，然后再把数字信号转换成模拟信号，在示波器上观察。

### 实验目的

本实验主要对ADDA的转换原理进行理解并加以练习，同时熟悉3PD5651E,3PA1030的端口，最终ADDA功能测试总体设计。

### 设计流程

根据前面的实验解析我们可以得知，该设计用一个简单的功能模块就能实现，

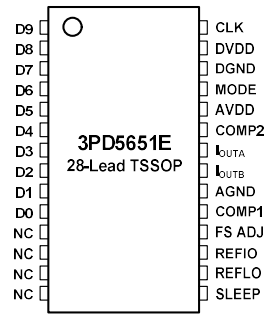
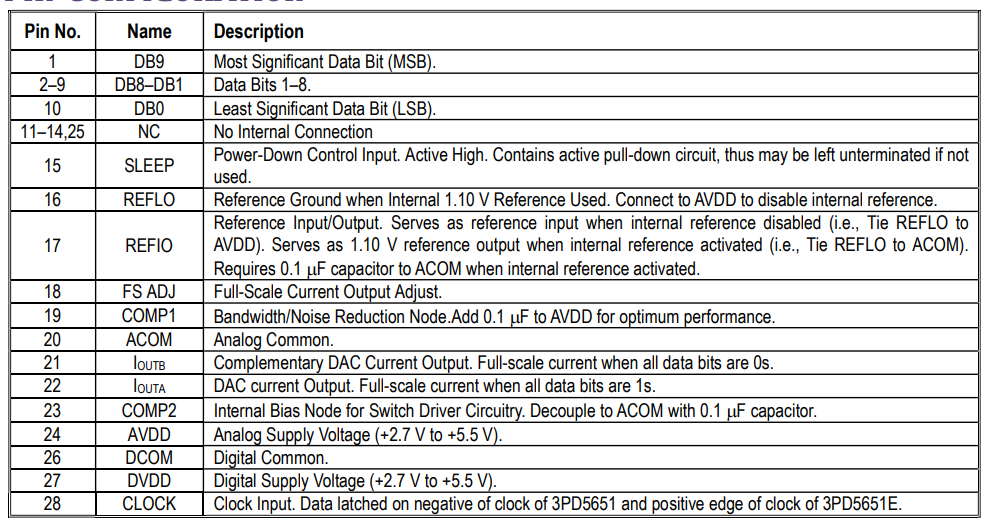
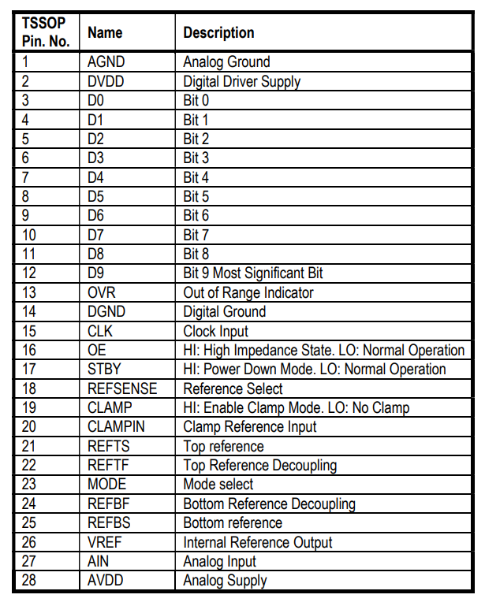
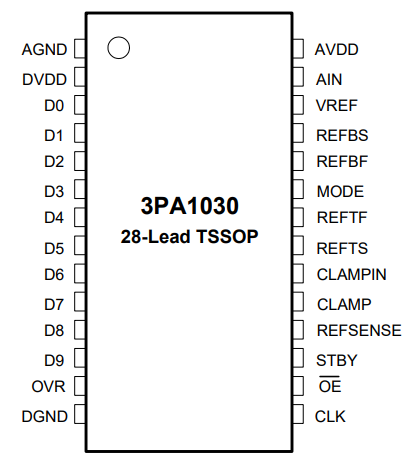
* ADDA\_test：数模转换器3PD5651E和模数转换器3PA1030，把模拟信号先转换成数字信号给到核心板，然后再把数字信号转换成模拟信号的模块。

### 实验原理

#### 3PD5651E,3PA1030芯片介绍

3PD5651E提供卓越的交流和直流性能，同时支持高达125 MSPS的更新速率。3PD5651E灵活的单电源工作范围为+2.7V至+5.5V，功耗低，非常适合便携式和低功耗应用。通过降低满量程电流输出，其功耗可以进一步降低到45mW，而不会显著降低性能。此外，下电模式可将待机功耗降低至约20mW。

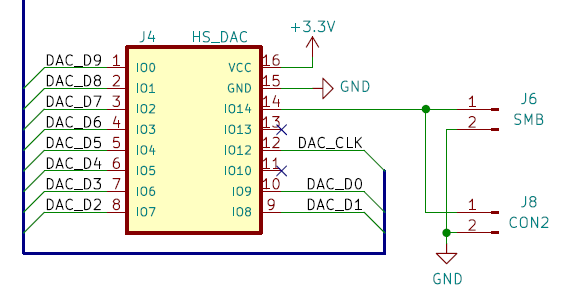
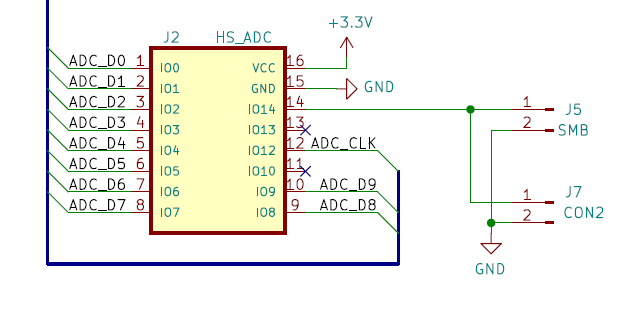
3PA1030是一款单片、单电源、10位、50 MSPS模数转换器，带有片上采样保持放大器和电压基准。3PA1030采用多级差分管道架构，数据速率为50 MSPS，并保证在整个工作温度范围内不会丢失代码。



管脚功能描述

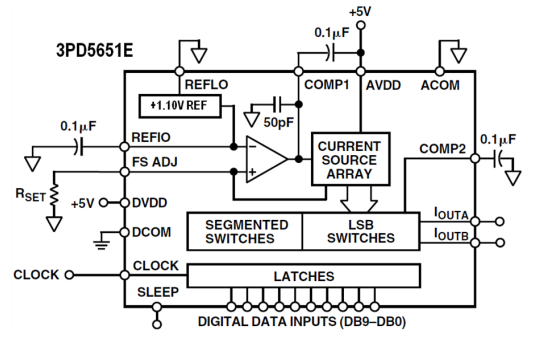
##### 3PD5651E,3PA1030芯片连接

STEP 电赛训练板底板上的3PD5651E,3PA1030芯片电路图如下

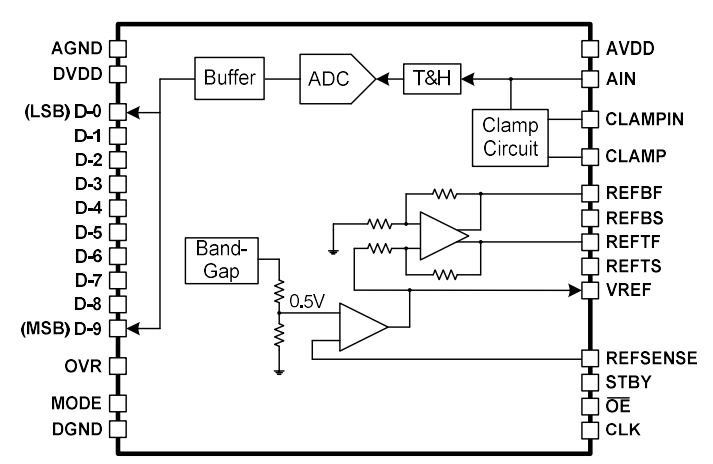


3PD5651E,3PA1030芯片电路

上图为模数转换器3PD5651E,数模转换器3PA1030芯片电路，与FPGA硬件接口分别总线上有十个数据位和一个时钟位。



3PD5651E内部框图



3PA1030内部框图

##### 系统实现

3PD5651E时钟最高频率为125MSPS，3PA1030时钟最高频率为50MSPS，且两者都具有十位数据位。而STEP-MAX10M08核心板时钟频率为12MHz，因此可以直接将核心板时钟信号作为3PD5651E，3PA1030的时钟信号，再把两者数据位相连，就可实现简单的ADDA功能。

module ADDA\_test

(

*// INPUT*

 input clk, *//12M时钟信号*

 input rst\_n, *//复位信号*

 input  [9:0] adc\_data, *//adc输出数据*

*// OUTPUT*

 output reg [9:0] dac\_data,  *//dac输入数据*

 output adc\_clk\_out, *//adc时钟信号*

 output dac\_clk\_out *//dac时钟信号*

);

assign adc\_clk\_out = clk;

assign dac\_clk\_out = clk;

always @(posedge clk or negedge rst\_n)

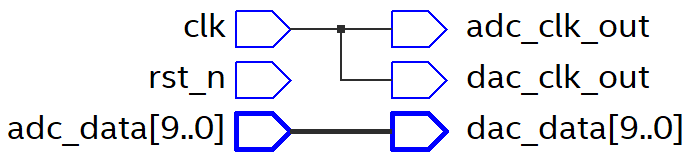
 begin

  dac\_data <= adc\_data; *//adc输出数据赋予dac输入端*

 end

endmodule

综合后的设计框图如下：

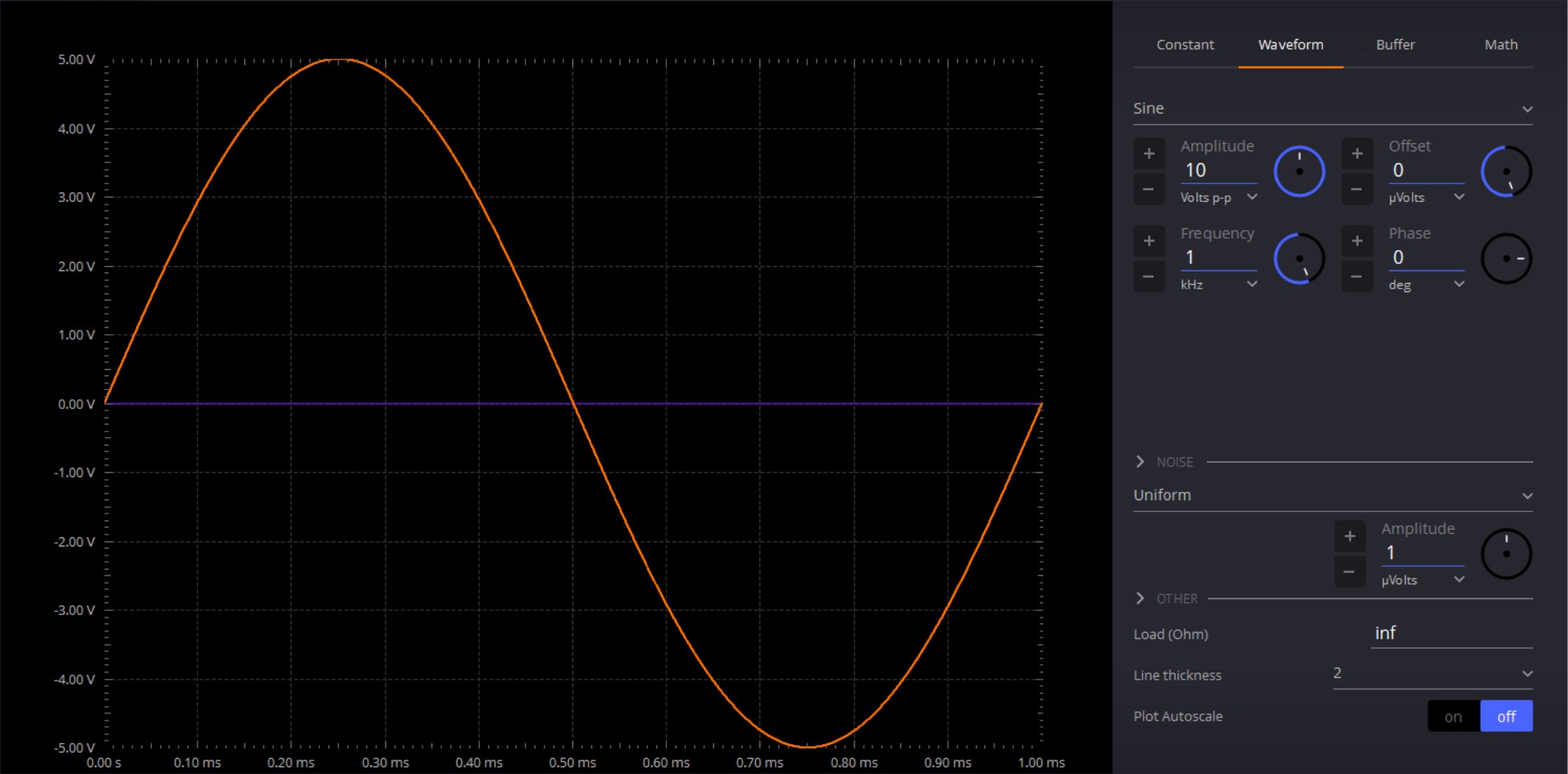


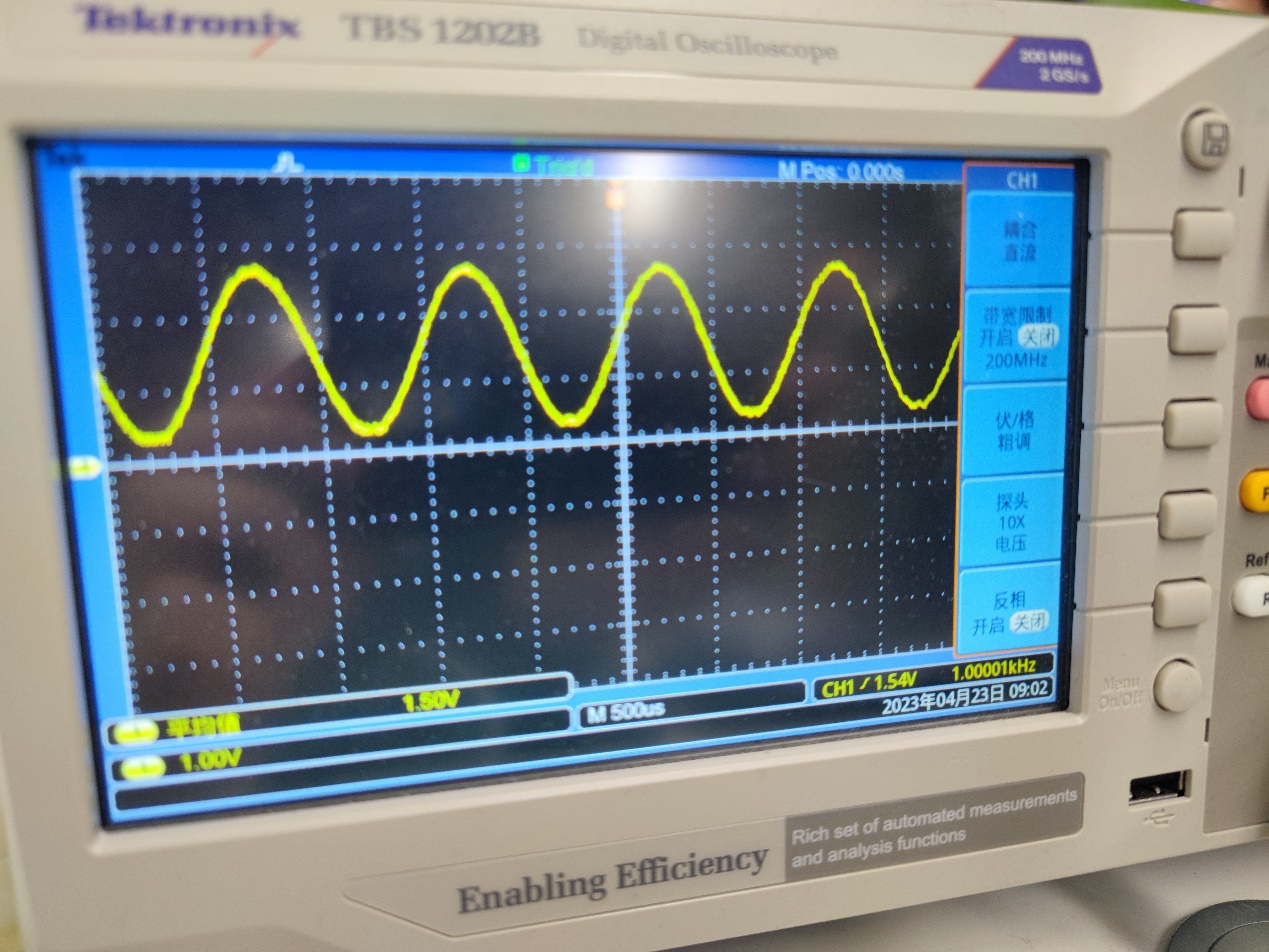
### 实验步骤

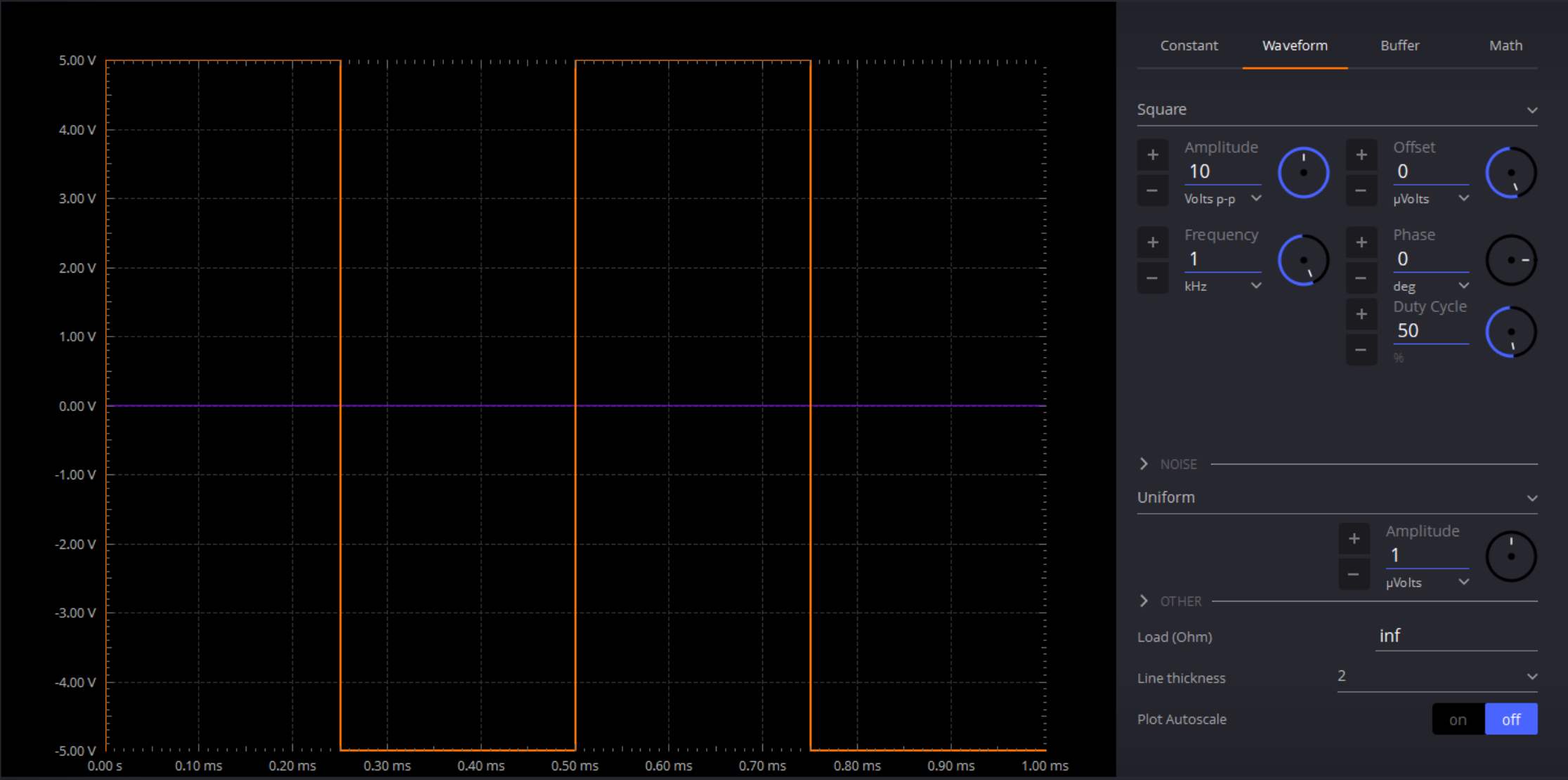
1. 双击打开Quartus Prime工具软件；
2. 新建工程：File → New Project Wizard（工程命名，工程目录选择，设备型号选择，EDA工具选择）；
3. 新建文件：File → New → Verilog HDL File，键入设计代码并保存；
4. 设计综合：双击Tasks窗口页面下的Analysis & Synthesis对代码进行综合；
5. 管脚约束：Assignments → Assignment Editor，根据项目需求分配管脚；
6. 设计编译：双击Tasks窗口页面下的Compile Design对设计进行整体编译并生成配置文件；
7. 程序烧录：点击Tools → Programmer打开配置工具，Program进行下载；
8. 观察设计运行结果。

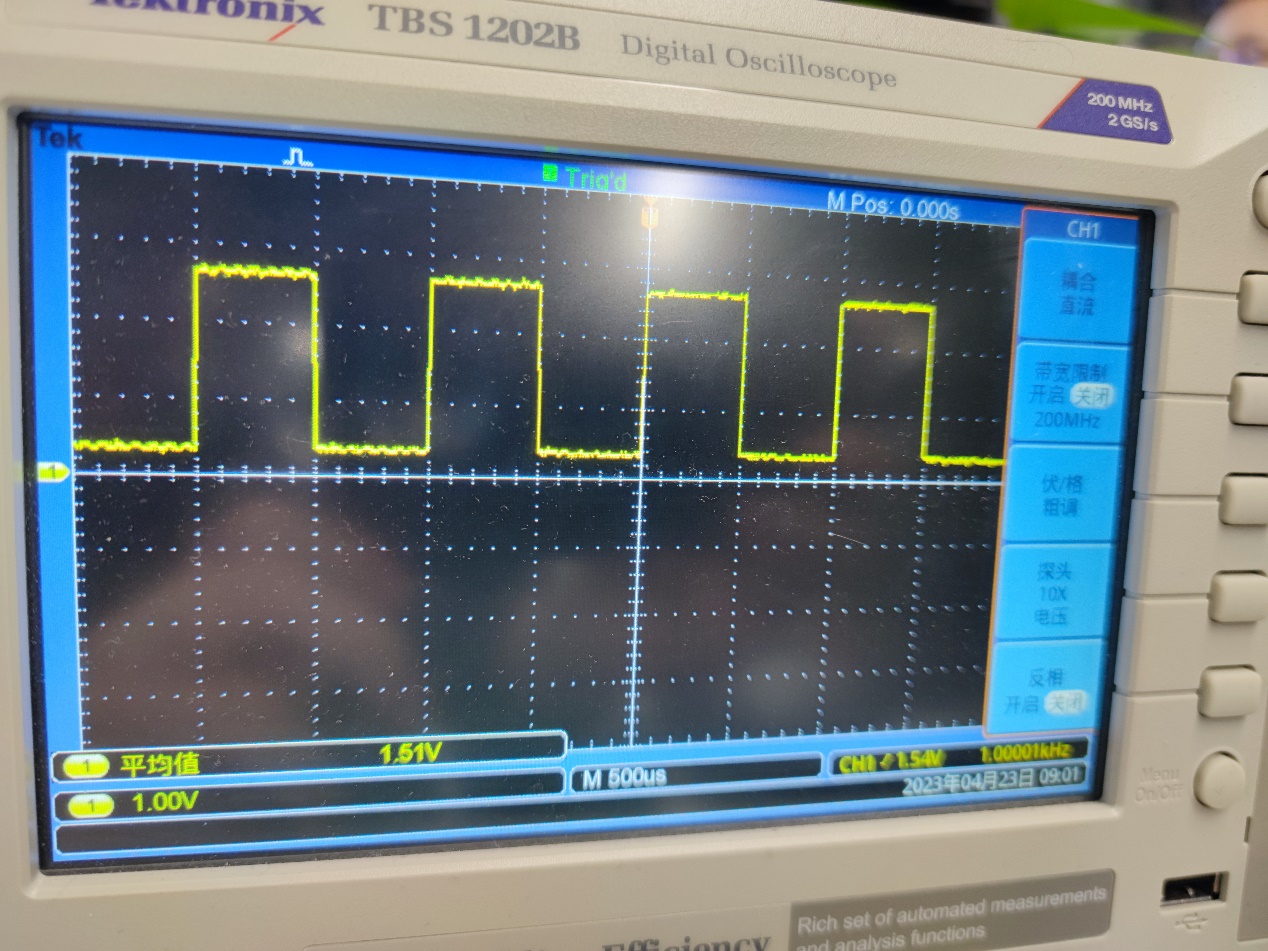
### 实验现象

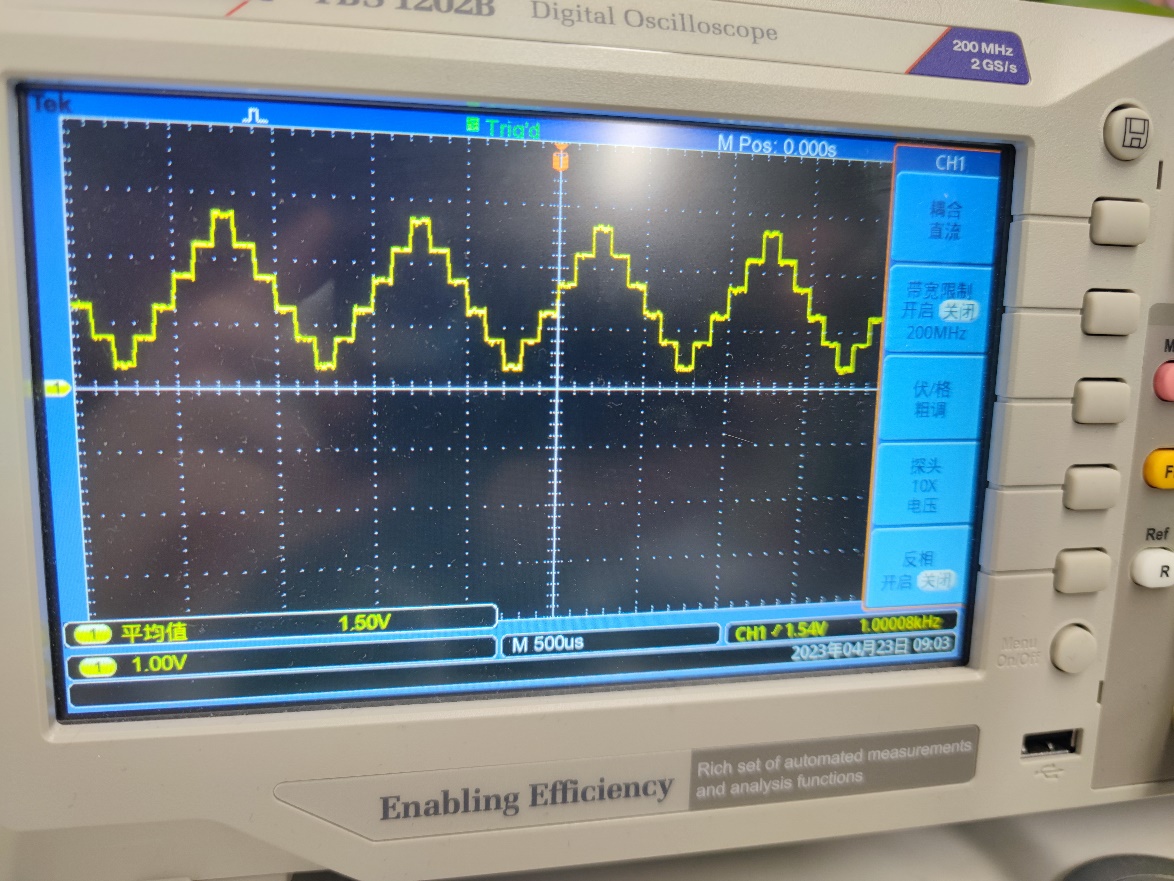
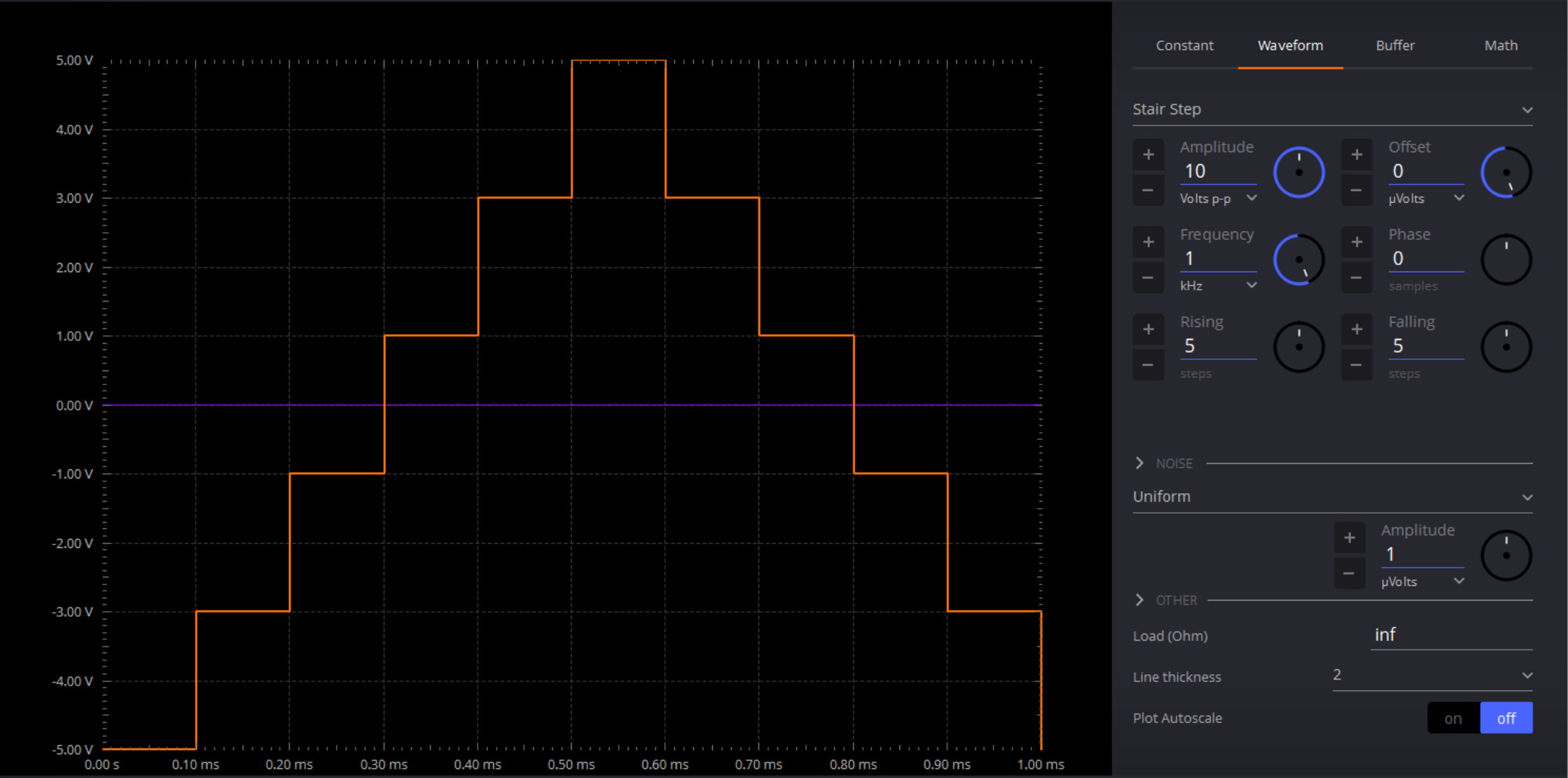
将程序加载到FPGA，将模拟信号连接到输入端，示波器连接到输出端，观察示波器显示变化。











# DDS任意波形发生器功能测试设计

### 实验任务

任务：基于STEP-MAX10M08核心板和STEP电赛训练板底板，完成DDS任意波形发生器功能测试设计并观察调试结果

要求：

1. 通过板上的高速DAC（10bits/125Msps）配合FPGA内部DDS的逻辑，生成波形可调（正弦波、三角波、方波）、频率可调（DC-）的波形
2. 在OLED上显示当前波形的形状
3. 利用板上旋转编码器和按键能够对波形进行切换、进行参数调节

解析：通过FPGA编程驱动旋转编码器，进行旋转调节和按键消抖，然后，基于DAC使用DDS将数字合成的波形转化为模拟电信号。

### 实验目的

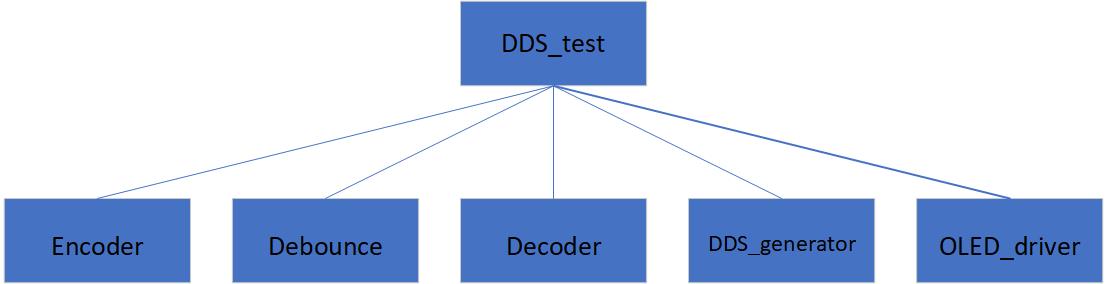
前面的章节中我们学习了OLED模块和旋转编码器模块的工作原理及驱动方法，本实验主要对DDS产生周期性的信号加以练习，同时熟悉OLED模块的读写操作和FPGA设计中常用运算方法，最终完成DDS任意波形发生器功能测试总体设计。

* 复习OLED显示工作原理驱动设计
* 复习旋转编码器模块的工作原理设计方法
* 完成DDS产生周期性的信号
* 完成DDS任意波形发生器功能测试总体设计

### 设计框图

根据前面的实验解析我们可以得知，该设计可以拆分成三个功能模块实现，

* Encoder：旋转编码器旋转驱动模块。
* Debounce：按键消抖模块。
* Decoder：旋转编码器波形选择频率调节驱动模块。
* DDS\_generator：DDS信号产生模块
* OLED\_driver：OLED驱动模块



Top-Down层次设计

### 实验原理

#### DDS“任意”信号生成

随着数字技术在仪器仪表和通信系统中的广泛使用，可从参考频率源产生多个频率的数 字控制方法诞生了，即直接数字频率合成(DDS)。其基本架构如图1所示。该简化模型采 用一个稳定时钟来驱动存储正弦波(或其它任意波形)一个或多个整数周期的可编程只读存储器(PROM)。随着地址计数器逐步执行每个存储器位置，每个位置相应的信号数字幅度 会驱动DAC，进而产生模拟输出信号。最终模拟输出信号的频谱纯度主要取决于DAC。相位噪声主要来自参考时钟。DDS是一种采样数据系统，因此必须考虑所有与采样相关的问题，包括量化噪声、混叠、滤波等。例如，DAC输出频率的高阶谐波会折回奈奎斯特带宽，因而不可滤波，而基于PLL的合成器的高阶谐波则可以滤波。此外，还有其它几种因素需要考虑，稍后将会讨论。

DDS需要一个DAC将数字合成的波形转化为模拟电信号，DAC的构成可以有多种形式：

* 高速并行的DAC；
* 低速串行的DAC；
* R-2R构成的低成本DAC；
* 通过PWM + RC构成的简易DAC。

无论使用任何一种DAC，DDS的内核 - 逻辑构成代码都是一样的，差别在于最终驱动DAC时需要不同的时序逻辑、不同的DAC方式能够支持的输出信号频率上限不同。

为了产生任意波形，DDS依赖两个技巧:

第一个技巧就是把“任意形状”的波形样点保存在一个LUT（查找表，类似组织有序的仓库）中，通过查找表格的方式来实现。

也可以通过其内部的BlockRAM（成块的RAM）来存储。FPGA的厂商在其工具中也都提供了用于产生正弦波表的IP核，可以直接调用来使用。其它形状的波形，乃至调制波形都可以通过一些工具（比如Matlab等）来生成数据，在编译的时候存储在例化好的ROM中，当成一个波形库来使用。

##### 相位累加器

上面讲述了任意波形的实现，那如何实现“任意频率”？这就体会到用“长”相位累加器这个神器的作用了，它使得DDS输出的信号的频率非常灵活。计数器本质上就是一个“相位累加器”，因为每一次它累加，它就会移动正弦波360°/256=1.4°。

现在，如果我们想让正弦波的输出信号频率翻倍（x2），只需要每次相位的累加值由“1”变成“2” 以此类推，可以让输出信号的频率x3、x4….

由于我们将每次的“累加值”增加到了16，而我们在查找表的时候用了phase\_acc[11:4]作为地址，我们没有改变其输出频率，但有了多出来的4位，就使得我们能够让相位累加器实现更高的分辨率. 现在我们当然能够降低输出频率到原来的一半 - 通过将相位累加器的累加值从“16”降低到“8”。

将相位累加器的分辨率提高了16，我们可以以1/16的步长获得原始正弦频率的任意倍数, 我们当然可以在相位累加器中增加超过4位更多的位。典型的DDS系统中使用非常“长”位数的相位累加器，比如28位、32位，来实现极高精度和分辨率的输出信号频率。

##### 旋转编码器驱动设计

旋转编码器按键选择波形输出：

always @(posedge clk or negedge rst\_n) begin

 if(!rst\_n) wave <= SIN;

 else if(O\_pulse)begin

   case(wave)

    SAW: wave <= TRI;

    TRI: wave <= SQU;

    SQU: wave <= SIN;

     SIN: wave <= SAW;

    default: wave <= SAW;

   endcase

end else wave <= wave;

end

旋转编码器旋转改变波形频率：

always@(posedge clk or negedge rst\_n) begin

 if(!rst\_n)

  begin

   f\_increment <= 12'd2048;

  end

 else

  begin

   if(L\_pulse)

    begin

     f\_increment <= f\_increment - 12'd8;

    end

   else if(R\_pulse)

    begin

     f\_increment <= f\_increment + 12'd8;

    end

   else

    begin

     f\_increment <= f\_increment;

    end

   end

  end

生成任意波形信号：

always @(posedge clk or negedge rst\_n)

 begin

 if(!rst\_n)

  phase\_accumulator <= 23'b0;

 else

  phase\_accumulator <= phase\_accumulator + f\_increment;

 end

wire [9:0] sin\_dat; *//正弦波*

wire [9:0] saw\_dat = phase\_accumulator[23:14]; *//锯齿波*

wire [9:0] tri\_dat = phase\_accumulator[23]? (~phase\_accumulator[22:13]) : phase\_accumulator[22:13]; *//三角波*

wire [9:0] squ\_dat = phase\_accumulator[23]? 8'hff : 8'h00; *//方波*

always @(\*) begin

 case(wave)

  2'b00: dac\_data\_out = saw\_dat; *//锯齿波*

  2'b01: dac\_data\_out = tri\_dat; *//三角波*

  2'b10: dac\_data\_out = squ\_dat; *//方 波*

  2'b11: dac\_data\_out = sin\_dat; *//正弦波*

  default: dac\_data\_out = saw\_dat; *//锯齿波*

 endcase

end

assign phase = phase\_accumulator[23:16]; *//相位是高8位*

正弦波形使用查找表完成。

##### 系统总体实现

OLED显示模块和按键消抖模块我们前面基础数字电路实验中详细介绍过，这里我们直接调用。

下面将五个模块例化：

wire L\_pulse,R\_pulse;

Encoder u1

(

.clk     (clk  ), *//系统时钟 12MHz*

.rst\_n    (rst\_n ), *//系统复位 低有效*

.key\_a    (key\_a ), *//旋转编码器EC11的A脚*

.key\_b    (key\_b ), *//旋转编码器EC11的B脚*

.L\_pulse    (L\_pulse ), *//左旋脉冲输出*

.R\_pulse    (R\_pulse ) *//右旋脉冲输出*

);

wire key\_jit,key\_pulse,key\_state;

*//key debounce module*

Debounce u2

(

.clk    (clk  ), *//系统时钟 12MHz*

.rst\_n    (rst\_n  ), *//系统复位 低有效*

.key\_n    (key\_o  ), *//按键信号输入*

.key\_jit   (key\_jit ), *//延时消抖输出*

.key\_pulse   (key\_pulse ), *//消抖脉冲输出*

.key\_state   (key\_state ) *//消抖翻转输出*

);

wire [1:0] wave;

wire [11:0] f\_increment;

Decoder u3

(

.clk      (clk  ), *//系统时钟 12MHz*

.rst\_n     (rst\_n ), *//系统复位 低有效*

.L\_pulse     (L\_pulse ), *//左旋脉冲输出*

.R\_pulse     (R\_pulse ), *//右旋脉冲输出*

.O\_pulse     (key\_pulse),

.wave      (wave   ),

.f\_increment   (f\_increment) *//高4位代表十位，低4位代表个位*

);

DDS\_generator u4

(

.clk      (clk  ), *//系统时钟 12MHz*

.rst\_n     (rst\_n ), *//系统复位 低有效*

.key\_a     (key\_a  ), *//旋转编码器EC11的A脚*

.key\_b     (key\_b  ), *//旋转编码器EC11的B脚*

.wave      (wave   ),

.f\_increment   (f\_increment),

.dac\_clk\_out   (dac\_clk\_out), *//*

.dac\_data\_out   (dac\_data\_out) *//*

);

OLED\_driver u5

(

.clk    (clk   ), *//12MHz系统时钟*

.rst\_n   (rst\_n  ), *//系统复位，低有效*

.freq\_100  (freq\_100 ),

.freq\_10   (freq\_10  ),

.freq\_1   (freq\_1  ),

.dc1\_10   (dc1\_10  ),

.dc1\_1   (dc1\_1  ),

.dc1\_01   (dc1\_01  ),

.dc2\_10   (dc2\_10  ),

.dc2\_1   (dc2\_1  ),

.dc2\_01   (dc2\_01  ),

.wave    (wave   ),

.oled\_rst  (oled\_rst ),

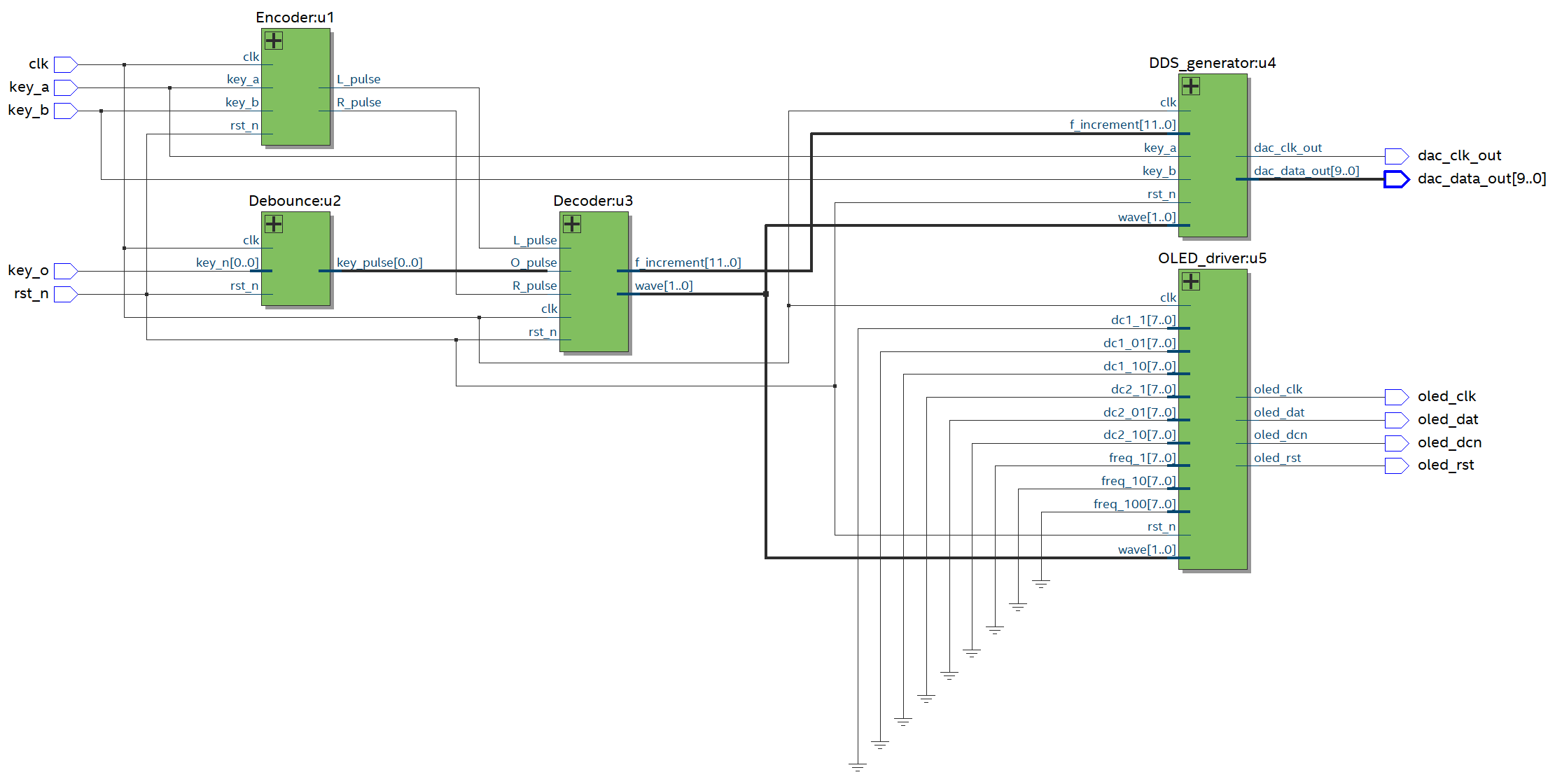
.oled\_dcn  (oled\_dcn ),

.oled\_clk  (oled\_clk ),

.oled\_dat  (oled\_dat )

);

综合后的设计框图如下：



### 实验步骤

1. 双击打开Quartus Prime工具软件；
2. 新建工程：File → New Project Wizard（工程命名，工程目录选择，设备型号选择，EDA工具选择）；
3. 新建文件：File → New → Verilog HDL File，键入设计代码并保存；
4. 设计综合：双击Tasks窗口页面下的Analysis & Synthesis对代码进行综合；
5. 管脚约束：Assignments → Assignment Editor，根据项目需求分配管脚；
6. 设计编译：双击Tasks窗口页面下的Compile Design对设计进行整体编译并生成配置文件；
7. 程序烧录：点击Tools → Programmer打开配置工具，Program进行下载；
8. 观察设计运行结果。

### 实验现象

将程序加载到FPGA，调节旋转编码器，观察示波器和OLED显示变化。

