

**数字EDA课程设计实验报告**

**学院： 计算机学院**

**专业： 信息安全**

**姓名： 陈四贵**

**学号： 182210710119**

**指导老师： 张笑非**

2020年6月24日

**目 录**

[实验一：曼彻斯特编码解码电路 1](#_Toc43917718)

[一、实验环境及相关软件 1](#_Toc43917719)

[二、曼彻斯特编码 1](#_Toc43917720)

[三、曼彻斯特解码 5](#_Toc43917721)

[实验二：帧同步电路 8](#_Toc43917722)

[一、实验目的 8](#_Toc43917723)

[二、实验原理 8](#_Toc43917724)

[三、实验内容 10](#_Toc43917725)

[四、实验环境及工具 10](#_Toc43917726)

[五 、实验流程 10](#_Toc43917727)

[实验三：循环码(7,4)编码解码电路 17](#_Toc43917728)

[一、实验目的 17](#_Toc43917729)

[二、编码器的设计 17](#_Toc43917730)

[四、实验过程 18](#_Toc43917731)

[五、程序设计 18](#_Toc43917732)

[六、系统仿真 23](#_Toc43917733)

[七、 实验结果 23](#_Toc43917734)

实验一：曼彻斯特编码解码电路

一、实验环境及相关软件

1. 操作系统：Windows 10 企业版（64位）
2. 实验软件：Quartus II 13.0sp1 (64-bit)

Modelsim 15.0(64-bit)

二、曼彻斯特编码

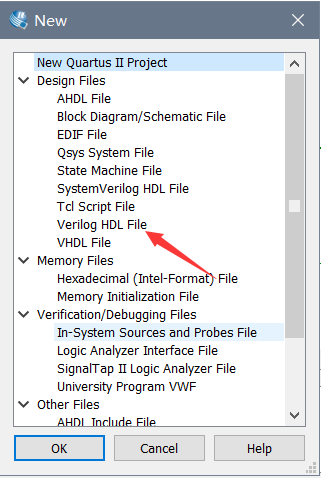
1. 实验目的

对NRZ信号进行Manchester编码。

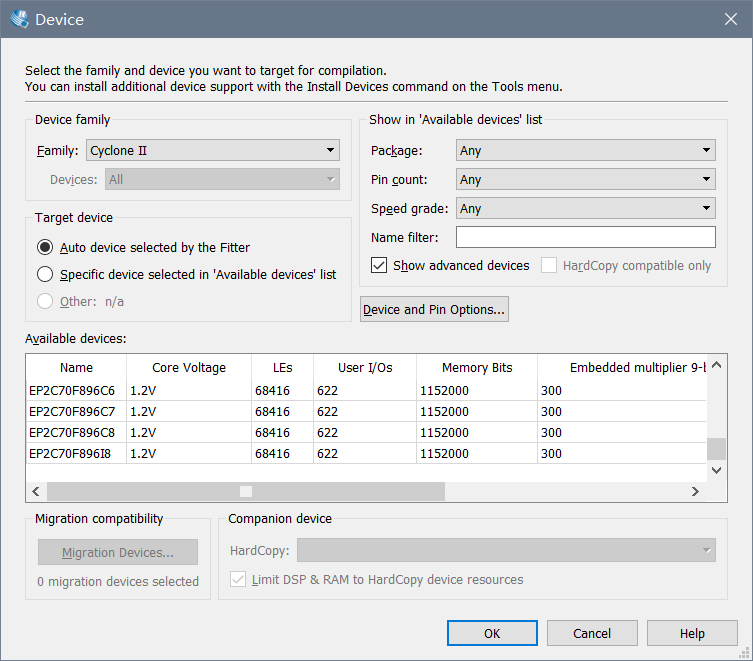
1. 编码规则

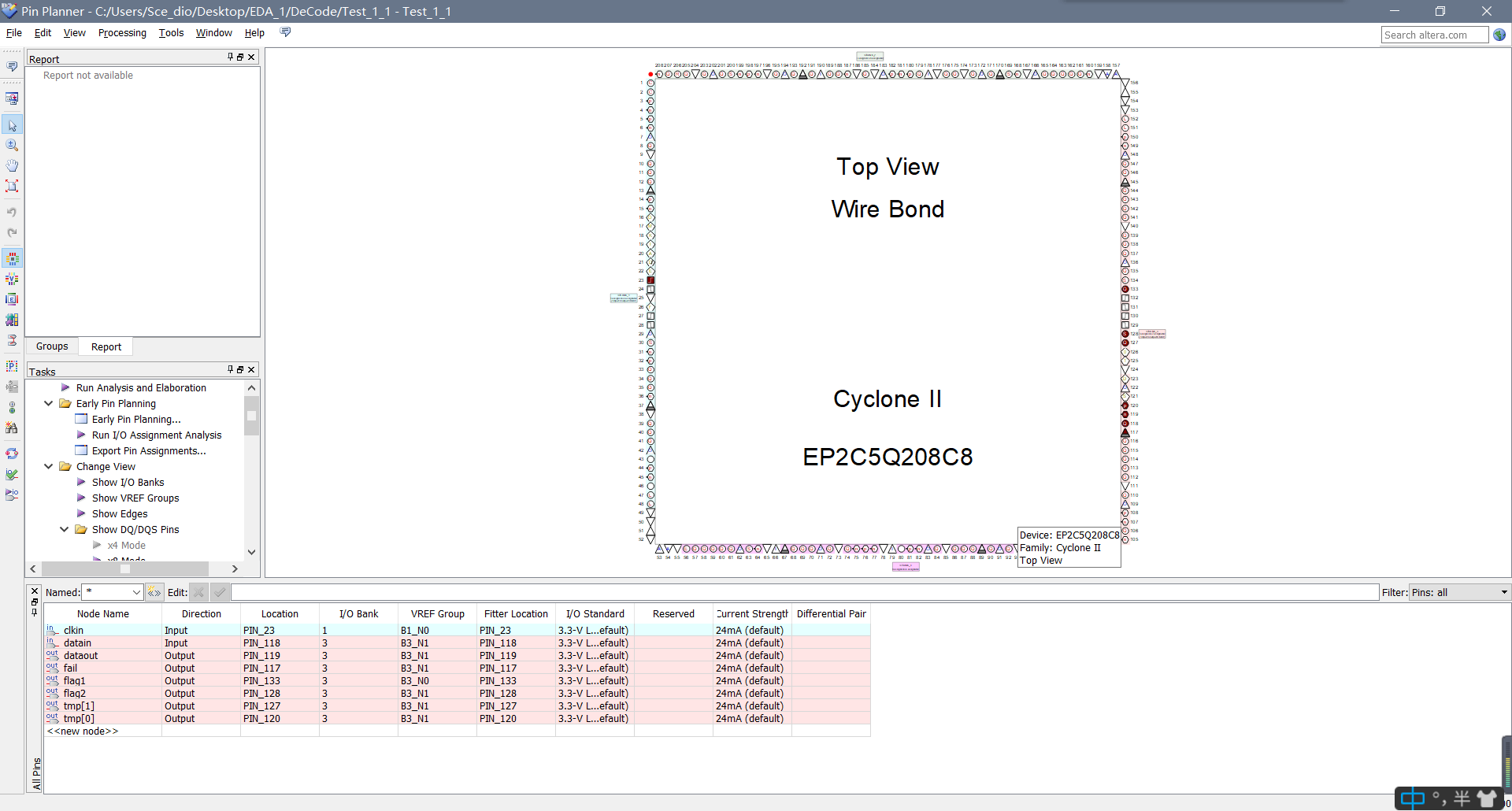
曼彻斯特码是通过电平的跳变来对二进制数据“0”和“1”进行编码的，对于何种电平跳变对应何种数据，实际上有两种不同的数据约定：第一种约定是由G. E. Thomas，Andrew S. Tanenbaum等人在1949年提出的，它规定“0”是由低到高的电平跳变表示，“1”是由高到低的电平跳变；第二种约定则是在IEEE 802.4(令牌总线)以及IEEE 802.3 (以太网)中规定，按照这样的说法，由低到高的电平跳变表示“1”，由高到低的电平跳变表示“0”。在实际工程上，这两种约定在一定范围内均有应用。为了便于描述，若无特殊说明，曼彻斯特码的编码规则均采用第二种约定，即从低电平跳变到高电平表示“1”，从高电平跳变到低电平表示“0”。

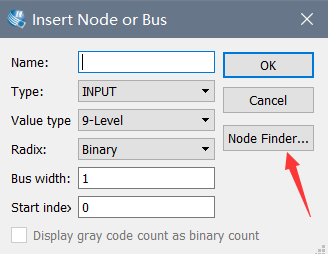
1. 实验过程
   1. 在Quartus中新建项目，完成文件名、保存路径等基本设置；
   2. 下一步可以选择导入已经创建好的Verilog文件，这里是直接点击下一步，然后再编写Verilog文件；



* 1. 选择EP2C5Q208C8（Cyclone II 系列的设备）；

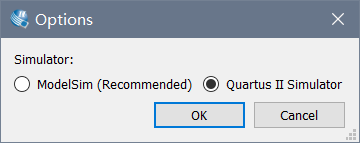


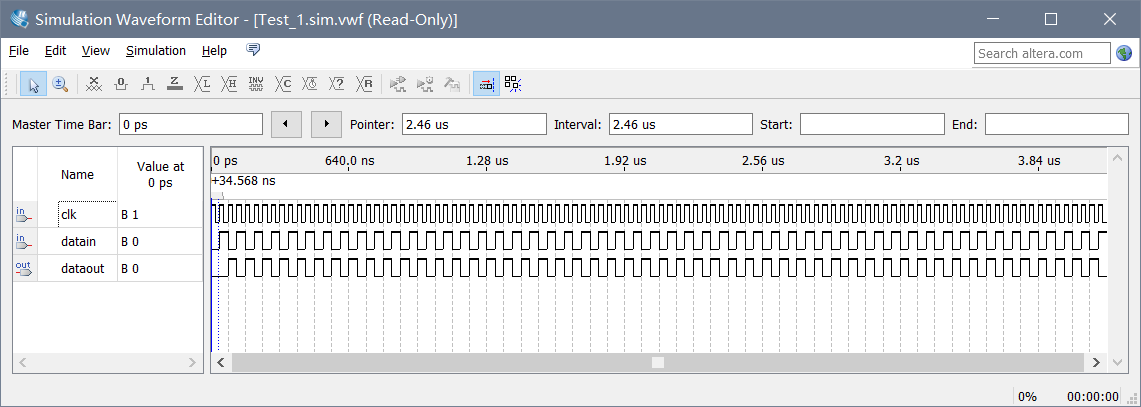
* 1. 项目创建完成，接着新建Verilog文件（注意文件名要与模块名一致），创建完成之后编译代码，修正错误之后开始分配引脚；
  2. 点击工具栏中 Assignments →Pin Planner,进行引脚配置。参考 Fitter Location 中内容将 Location 中的内容与其配置一致，然后点击 File →Export 导出引脚。点 击Assignments →Assignment Editor,可以查看已经配置好的引脚；
  3. 点击 File → New →University Program VWF→OK，建立波形文件，在 VWF 文件的工具栏中点击 Edit → Insert→ Insert Node or Bus。在弹出的 “Insert Node or Bus”对话框中点击“Node Finder”按钮。在弹出的 Node Finder 界面里，点击 List 按钮左边的按钮，然后选择Nodes Found一栏中所有端口，点击



* 1. 为输入端口设置输入信号，选中某个输入，然后点击

设置Cout every，clk为20ns，datain为40ns，最后保存vwf文件；

* 1. 点击 Simulation →Options →Quartus II Simulator →OK；
  2. 点击 Simulation →Run Functional Simulation，进行波形仿真，观察分析得到的仿真波形。



1. 实验结果及分析
   1. Verilog源码

**module** Test\_1**(**clk**,**datain**,**dataout**);**

**input** clk**,** datain**;** //时钟输入，数据输入

**output** dataout**;** //曼彻斯特编码输出

**reg** dataout**,**flag**;** //flag 为标志信号

**reg** **[**1**:**0**]** com**;**

**always** **@(posedge** clk**)**

**begin**

**if(**flag **==** 1'b0**)**

**begin**

**if(**datain **==** 1'b0**)**//当数据为 0 时，转

**begin**

com **<=** 2'b01**;**

**end**

**else** //当数据为 1 时，转换成 10

**begin**

com **<=** 2'b10**;**

**end**

**end**

**end**

**always** **@(posedge** clk**)** //曼彻斯特编码输出过程

**begin**

**if(**flag **==** 1'b1**)**

**begin**

dataout **<=**com**[**1**];**

flag **<=~**flag**;**

**end**

**else**

**begin**

dataout **<=** com**[**0**];**

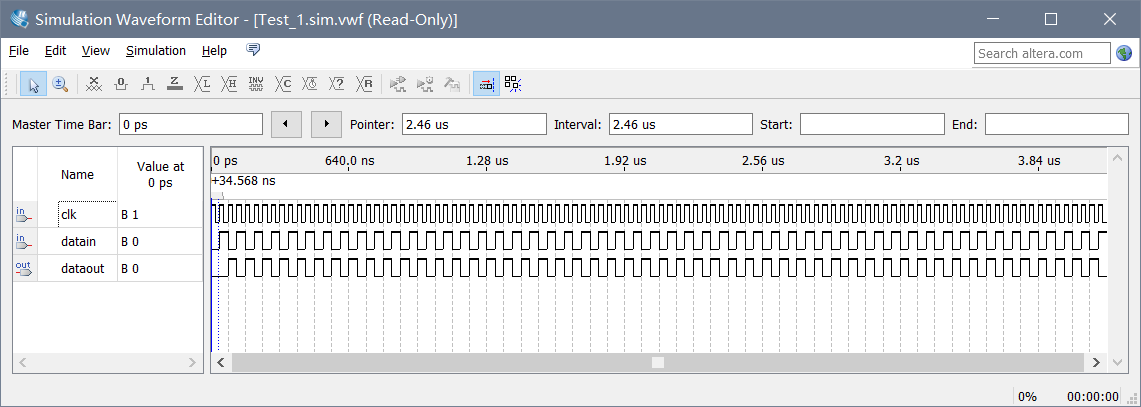
flag **<=~**flag**;**

**end**

**end**

**endmodule**

* 1. 波形分析



输出信号 dataout 符合曼彻斯特编码规则。因为当输入信号datain是“1”时，编码输出信号dataout为“10”，当datain是“0”时，dataout为“01”。这与根据曼彻斯特编码的特点得出的理论波形的形状是一致的。

三、曼彻斯特解码

1. 实验目的

从曼彻斯特编码中恢复数据和时钟信号

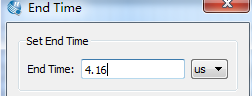
1. 实验原理

曼彻斯特解码与其编码具有相同的原理：设NRZ数据为D，其位率时钟为T，则曼彻斯特编码为: M=D⊕T

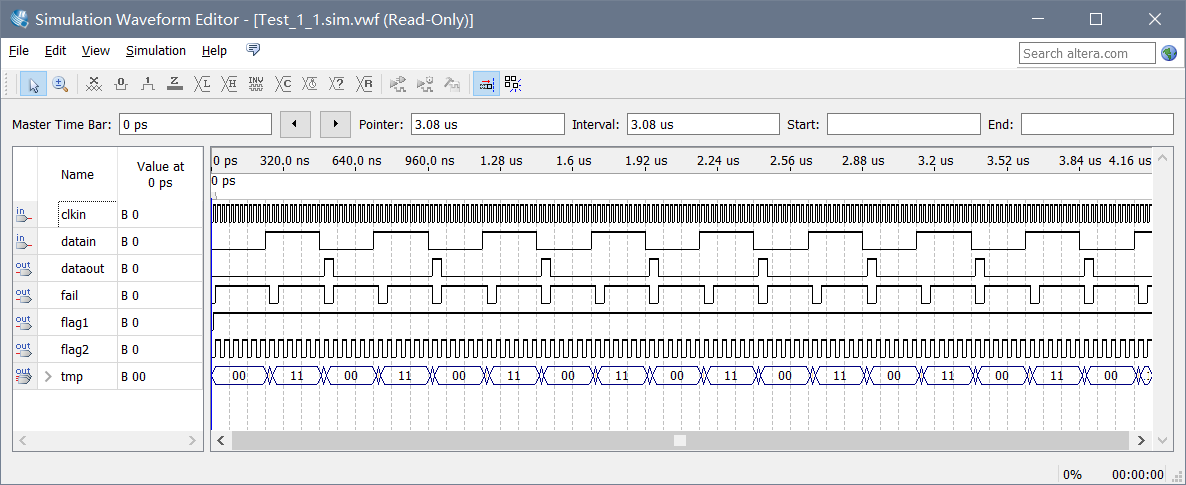
曼彻斯特解码是将编码数据 M 与位率时钟 T 相异或即：

M⊕T = (D⊕T)⊕T = D⊕T⊕T = D⊕(T⊕T) = D⊕0 = D

1. 实验过程（主要步骤与编码部分无异）
   1. 在Quartus中新建项目，完成文件名、保存路径等基本设置；
   2. 点击下一步，选择导入已经创建好的Verilog文件；
   3. 选择EP2C5Q208C8（Cyclone II 系列的设备）；
   4. 项目创建完成，接着编译导入的Verilog文件（注意文件名要与模块名一致），修正错误之后开始分配引脚；
   5. 点击工具栏中 Assignments →Pin Planner,进行引脚配置；
   6. 点击 File → New →University Program VWF→OK，建立波形文件；
   7. 在 VWF 文件的工具栏中点击 Edit →Set End Time，设置 End Time 为 4.16us；



* 1. 设置输入的clkin的 Cout every 为 80us，输入的datain的Count every为240us，保存VWF文件；
  2. 点击 Simulation →Options →Quartus II Simulator →OK；
  3. 点击 ，进行波形的功能仿真，观察分析得到的波形。



1. 实验结果及分析
   1. Verilog源码

**module** Test\_1\_1**(**clkin**,**datain**,**tmp**,**flag1**,**flag2**,**dataout**,**fail**);**

**input** clkin**,**datain**;**

**output** flag1**,**flag2**,**dataout**,**fail**;**//flag 是开始转化标志，flag 是两个译码的标志，fail 是错误标志

**output** **[**1**:**0**]** tmp**;**

**reg[**1**:**0**]** tmp**;**

**reg** flag1**=**0**;**

**reg** flag2**=**0**;**

**reg** dataout**;**

**reg** fail**=**0**;**

**always** **@(posedge** clkin**)**

**begin**

tmp**<={**datain**,**tmp**[**1**]};**

**if(**tmp**==**2'b00 **||** tmp**==**2'b11**)**

flag1**<=**1**;**//表示可以开始转化，因为 00 后面一定是 1,11 后面一定是 0，可以确定开始，相当于 rst

**end**

**always@(negedge** clkin**)**

**begin**

**if(**flag1**==**1**)**

flag2**<=~**flag2**;**//因为是要两个译码，时钟慢一倍，即译码周期是输入时钟周期的 2 倍。

**end**

**always** **@(posedge** flag2**)**

**begin**

**if(**tmp**==**2'b10**)**

**begin**

dataout**<=**0**;**

fail**<=**0**;**

**end**

**else** **if(**tmp**==**2'b01**)**

**begin**

dataout**<=**1**;**

fail**<=**0**;**

**end**

**else** **if(**tmp**==**2'b00**||**tmp**==**2'b11**)**

**begin**

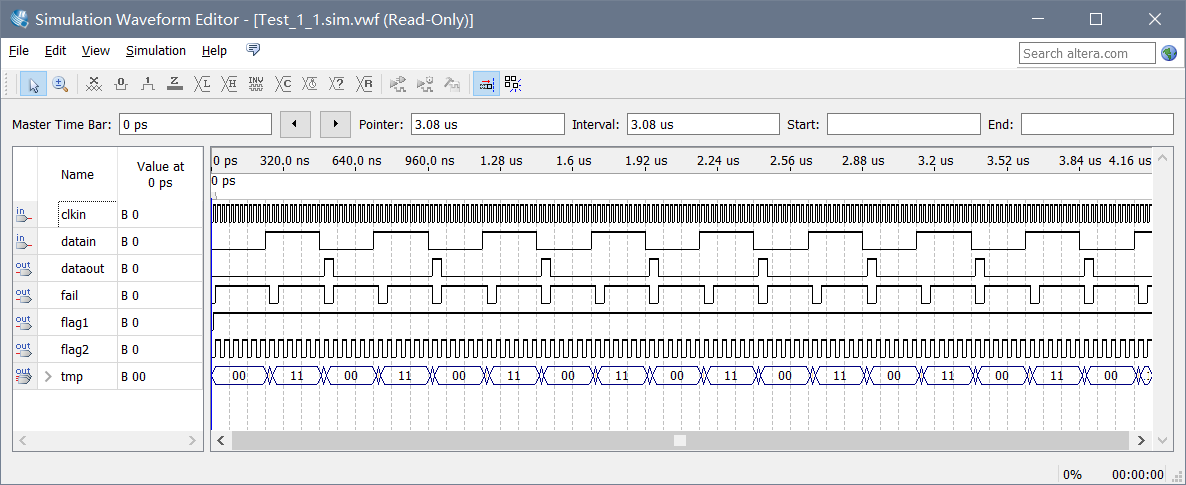
dataout**<=**0**;**

fail**<=**1**;**//有错就置 1，输出置为 0

**end**

**end**

**endmodule**

* 1. 波形分析

波形的时间范围是 0~4.16us，每一小格代表40ns。且tmp[1]比输入数据信号延迟 4 格，即 160ns；tmp[0]比输入数据信号延迟8格，即320ns。所以tmp[0]表示当前时刻的值，tmp[1]表示下一时刻的值。在数组中，tmp[1]为高位，tmp[0]为低位。在 flag2 上升沿，也就是输入时钟的每两个时钟的下降沿，开始译码。当 tmp=10 时，表示当前时刻输入为 0，下一时刻输入为 1。从 0 跳变为 1，输出数据为 0；译 码成功，fail为0。当 tmp=01 时，表示当前时刻输入为 1，下一时刻输入为0。从 1 跳变为0，输出数据为1；译码成功，fail为 0。当 tmp=00（11）时，表示当前时刻输入为0（1），下一时刻输入为0（1）。没有跳变，输出数据为0；译码失败，fail为1。由代码知，flag1初始化为0，所以在开始的80纳秒，flag1置0。当检测到tmp为00时，flag1置1，表示可以开始译码，因为00后面一定是1,11后面一定是0。因为在此之后都是译码，所以flag1一直为 1。

实验二：帧同步电路

一、实验目的

1.理解帧同步原理及实现过程。

2.掌握逐码移位法等各种方法实现帧同步。

3.学会使用verilog语言实现硬件功能。

二、实验原理

**1.帧结构及提取电路的构思**

①帧结构的特点如下：每一帧长度固定为535bit；同步字段SYS固定为1011010的二进制序列；每一帧包含有16个数据块B1～B16，每个数据块为32bit，每个数据块之前由1bit的标志位引导；特征标志位A1～A16排列顺序为：1010111001001XXX，后3位为备用，设计时可不考虑其数值。

②提取电路的构思：由帧同步头的特点及系统性能的要求，考虑使用状态机来实现。帧同头的提取具体步骤如下：

（1）START：当系统复位后系统直接进入本状态，检测当前输入信号是否为1。

（2）STEP1：检测第二时钟来时的输入信号是否为0，如果输入的信号是0，则进入第三个状态STEP2；如果输入信号为1，则表明现在收到的两个数据肯定不是帧头的前两位则保持在本状态，并且认为现在收到的这一数据是帧头的第一位。

（3）STEP2：如果收到数据为1则表明到现在为止收到的3个数据可能是一个帧头的前三位，进入第四个状态STEP3；如果收到的数据是0则表明到现在为止收到的三个数据肯定不是帧头，则返1则保持系统工作在此状态；当输入信号为1时进入下个状态STEP1。回STEP2。

（4）STEP3：如果在本状态下收到数据为1则进入下一个状态STEP4；如果收到的数据为0，状态返回到STEP2。

（5）STEP4：进行第5位的检测，如果接收到的数据为0时系统进入下个状态STEP5；如果接收到的数据为1时，则表明前面收到的数据都不是帧头，系统跳到状态STEP1进行帧头第二位的检测。

（6）STEP5：接收到的数据为1时，系统跳到下个状态STEP6；如接收的数据为0时，此时收到数据为101100肯定都不属于帧头，则系统返回START等待接收第一位帧头。

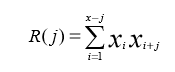
（7）STEP6：接收到的数据为0时表示现在收到的7位数据与帧头相同，状态进入HEADDONE；如果接收到数据是1时表示接受的到后的数据可能是帧头的前4位，则系统跳到STEP4。

（8）HEADDONE：当进入本状态时表明，收到的前7位数据为1011010即有可能收到帧头。然后将本状态下收到的数据装入寄存器FlagBit[CountF]，同时将CountF加1，然后将系统跳转到COUNTDATA状态。当接收完16位后，进行对FlagBit所存数据与1010111001001XXX的比较，如果一样则进行数据的发送。

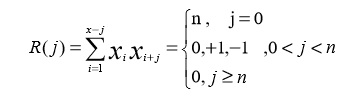
（9）COUNTDATA：系统进行有效数据的接收和暂存，当接收完32位数据后跳转到HEADDON进行标志位的接收。

**2.帧同步实现原理**

①连贯式插入法就是在每帧的开头集中插入特殊的同步码组，该特殊码组要求具有尖锐单峰特性的局部自相关函数，另外识别器应尽量简单。对一个n位的序列｛x1,x2,…,xn｝，其局部自相关函数表达式为：



目前常用的帧同步码组为巴克码，它的局部自相关函数是：



可见巴克码具有尖锐单峰特性。以7位巴克码为例，文献中均给出了用7级移位寄存器、加法器、判决器组成的7位巴克码识别器，如图1所示。该识别器的行为可描述为：当寄存器内的码元与对应的巴克码元相同时，加法器加1，反之加法器减1，并将结果与设定的门限比较输出示位脉冲。

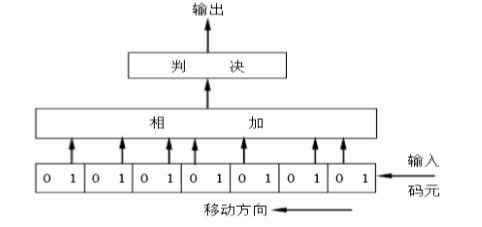


图1巴克码识别器的原理图

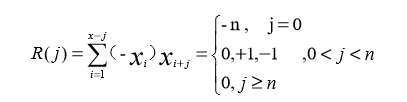
②用巴克码辨别2PSK接收的相位模糊

2PSK解调时本地载波可能会有“相位模糊”问题从而导致“反相工作”，而2DPSK虽然解决了反相工作问题但是需要较高的信噪比，而且会造成误码扩散，不利于前向纠错。所以只要解决了相位模糊问题，2PSK在功率利用和可靠性方面具有优势。

当2PSK接收系统反相工作时，接收的码元与实际码元相反，我们只须简单的将寄存器中的数据取反后送入另一组相加判决器，就能得到反相工作时的帧同步示位脉冲。正相与反相示位脉冲不可能同时出现，据此可以辨别出现相位模糊与否。上述方法虽简单，但需两组相加判决器，巴克码较长时，耗费资源较多。实际上巴克码前后都有随机的信息码，所以图1电路计算的并不是局部自相关函数值，但仍可识别巴克码对图1电路进一步分析，发现当反相的7位巴克码全部进入寄存器时，相加器输出为：

（-1）×1＋（-1）×1+（-1）×1＋1×(－1)+（-1）×1+1×（-1）=－7

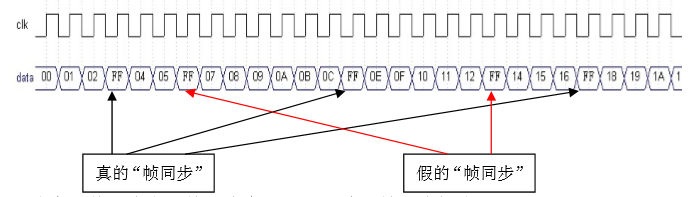
把式（2）中的xi用（-xi）代替，得出：



由此可见图1中的相加器对反相的巴克码有负峰值输出，我们只须在原电路基础上增加一个判决器即可得到反相工作时的帧同步脉冲。

三、实验内容

输入数据data为8bit并行数据流，基本结构为数据帧，帧长为10字节，帧同步字为H“FF”。clk为输入同步时钟。如下图所示：



1.搜索出数据流中的帧同步字信号，并给出帧同步标志。

2.系统工作开始后，要连续3次确认帧同步字进入锁定状态后才输出帧同步标志。

3.在锁定状态时，如连续出现3次错误的帧同步字，则帧同步标志输出无效，系统重新进入搜索状态；否则继续输出有效的帧同步标志。

4.过滤掉虚假的帧同步字（数据载荷中随机的H“FF”）。

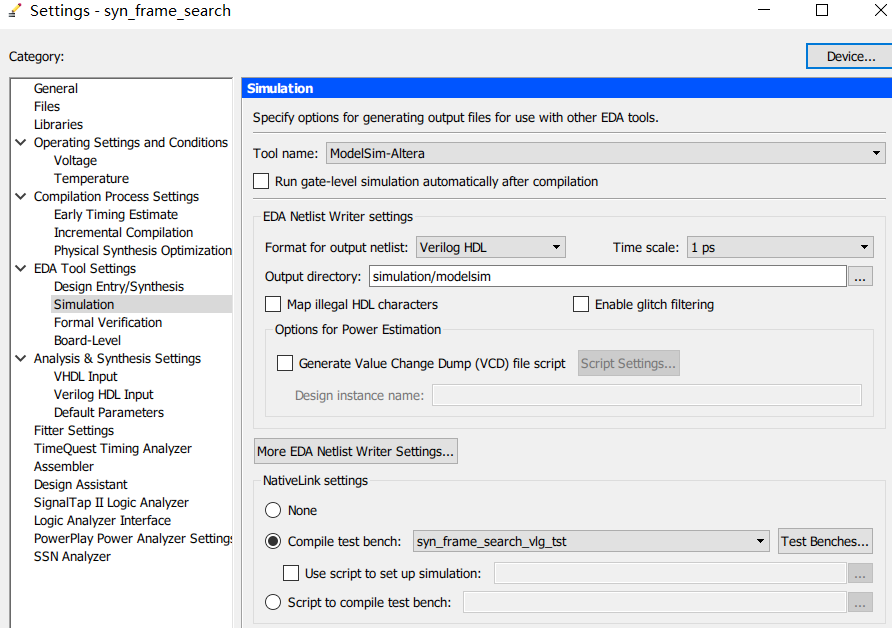
5.完成综合后的时序仿真验证。

四、实验环境及工具

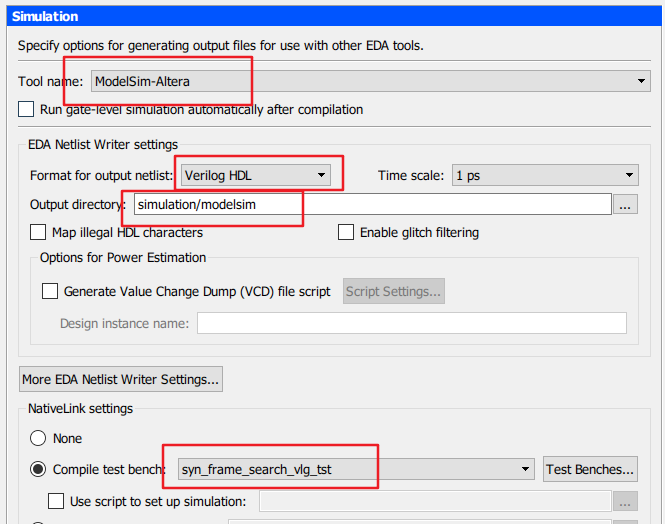
1. Windows 10 操作系统
2. Quartus II 13.0.1开发软件
3. Modelsim仿真器
4. Verilog HDL 语言

五 、实验流程

**1、**打开Quartus，保存文件为“sync\_frame\_search.v”，编译代码。

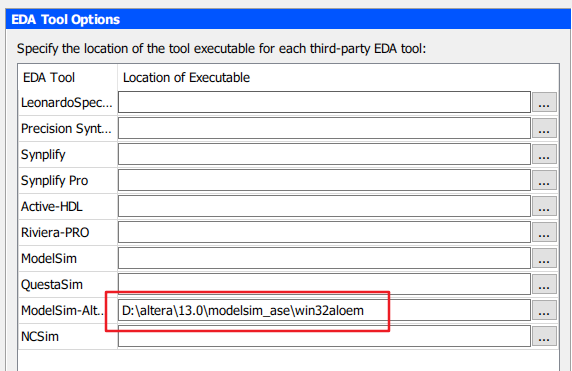


2、生成测试文件，并修改代码使其满足测试要求，载入测试文件：

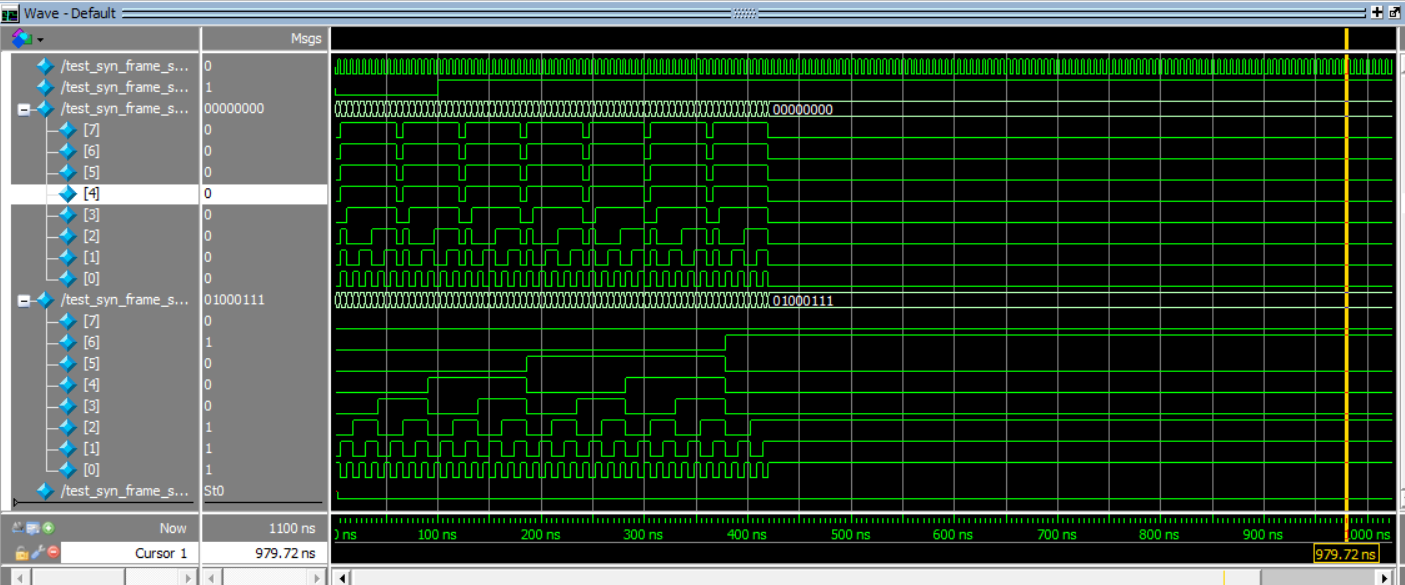


3、安装Modelsim仿真器，并设置插件启动路径：

注意Modelsim的安装路径需要保持和Quartus一致。



4、启动Modelsim，并观察仿真波形



**六、基于Verilog的实验程序**

`timescale 1ns **/** 1ps

**module** Test\_2**(**clk**,**

rst\_n**,**

data**,**

syn\_out\_flag**);**

**input** clk**,**rst\_n**;**//时钟及复位

**input[**7**:**0**]** data**;** //数据流

**output** syn\_out\_flag**;**//同步输出

**parameter** FRAMEHEAD **=** 8'H47**;** //帧同步信号 47H

**parameter** FRAMECOUNT**=** 10**;** // 帧长为 10 字节

**parameter** SEA\_CAP **=** 3 **;** // 搜捕态同步信号持续次数

**parameter** ERROR\_ALLOW **=** 3**;** // 稳定输出时允许同步信号连续出错次数

// state code

**parameter** SEARCH **=** 4'b0001**;**//搜索帧头态

**parameter** CHECK **=** 4'b0010**;**//搜捕态

**parameter** LOCATE **=** 4'b0100**;**//稳定同步态

**parameter** ERROR **=** 4'b1000**;**//容错态

// regs & wires

**reg[**3**:**0**]** state**;**//状态机当前状态变量

**reg[**3**:**0**]** next\_state**;**//状态机下一状态变量

**reg[**3**:**0**]** cnt**;**//定时计数器，为帧同步提供锁定位标志信号

**reg[**1**:**0**]** s**;**//搜捕计数器

**reg[**1**:**0**]** r**;**//容错计数器

**wire** syn\_out\_flag**;**//同步输出

**always** **@** **(** **posedge** clk **)**

**if** **(** **!**rst\_n **)**

state **<=** SEARCH**;**

**else**

state **<=** next\_state**;**

**always** **@** **(** state **or** data **or** cnt **or** s **or** r **)**

**begin**

**case** **(** state **)**

SEARCH**:** **begin** //搜索帧头态，‘是’进入搜捕态，‘否’继续搜索帧头

**if** **(**data **==** FRAMEHEAD**)**

next\_state **=** CHECK**;**

**else**

next\_state **=** SEARCH**;**

**end**

CHECK **:** **begin** //搜捕态

**if** **(** cnt **==** FRAMECOUNT **-** 1 **&&** data **==** FRAMEHEAD **&&** s **==** SEA\_CAP**)**

next\_state **=** LOCATE**;** //搜索到同步信号且搜捕计数器到达设定次数，进入稳定同步态

**else** **if** **(** cnt **==** FRAMECOUNT **-** 1 **&&** data **==** FRAMEHEAD **&&** s **!=** SEA\_CAP**)**

next\_state **=** CHECK**;** //搜索到同步信号但搜捕计数器未到达设定次数，进入稳定同步态

**else** **if** **(** cnt **==** FRAMECOUNT **-** 1 **&&** data **!=** FRAMEHEAD**)**

next\_state **=** SEARCH**;** //未搜索到同步信号进入搜索帧头态

**else**

next\_state **=** CHECK**;**

**end**

LOCATE**:** **begin** //稳定同步态

**if** **(** cnt **==** FRAMECOUNT **-** 1 **&&** data **!=** FRAMEHEAD **)**

next\_state **=** ERROR**;** //未搜索到同步信号，进入容错态

**else**

next\_state **=** LOCATE**;** //搜索到同步信号继续为稳定同步态

**end**

ERROR**:** **begin** //容错态

**if** **(** cnt **==** FRAMECOUNT **-** 1 **&&** data **!=** FRAMEHEAD **&&** r **==** ERROR\_ALLOW **)**

next\_state **=** SEARCH**;** //未搜索到同步信号且搜捕计数器到达设定次数（本次实例为 3 次），进入搜索帧头态

**else** **if** **(** cnt **==** FRAMECOUNT **-** 1 **&&** data **!=** FRAMEHEAD **&&** r **!=** ERROR\_ALLOW **)**

next\_state **=** ERROR**;** //未搜索到同步信号但搜捕计数器未到达设定次数继续容错

**else** **if** **(** cnt **==** FRAMECOUNT **-** 1 **&&** data **==** FRAMEHEAD **)**

next\_state **=** LOCATE**;**//搜索到同步信号进入稳定同步态

**else**

next\_state **=** ERROR**;**

**end**

**endcase**

**end**

**wire** cnt\_en**=(** state **==** CHECK **||** state **==** LOCATE **||** state **==** ERROR **)?** 1'b1 **:** 1'b0**;** //定时计数器使能

**wire** s\_n**=(**state **==** CHECK **)?** 1'b1 **:** 1'b0**;**//搜捕计数器使能

**wire** r\_n**=(**state **==** ERROR **)?** 1'b1 **:** 1'b0**;**//容错计数器使能

**always** **@** **(** **posedge** clk **)**

**if** **(** **!**rst\_n **)**

**begin**

s **<=** 0**;**

r **<=** 0**;**

cnt **<=** 0**;**

**end**

**else**

**begin**

**if** **(** **!**s\_n **)** //搜捕计数器

s **<=** 0**;**

**else**

s **<=** **(** cnt **==** FRAMECOUNT **-** 1 **&&** s\_n **)** **?** s **+** 1 **:** s **;**

**if** **(** **!**r\_n **)** //容错计数器

r **<=** 0**;**

**else**

r **<=** **(** cnt **==** FRAMECOUNT **-** 1 **&&** r\_n **)** **?** r **+** 1**:** r **;**

**if** **(** cnt **==** FRAMECOUNT **-** 1 **)** //定时计数器

cnt **<=** 0**;**

**else**

cnt **<=** **(** cnt\_en **)** **?** cnt **+** 1 **:** 0**;**

**end**

**assign** syn\_out\_flag**=(**cnt**==**FRAMECOUNT**-**1**)** **?** **(** state **==** LOCATE **||** state **==** ERROR **)** **?** 1'b1 **:** 1'b0

**:** 1'b0**;** //在 LOCATA 和 ERROR 态时同步输出

**endmodule**

**七、测试代码**

`timescale 1ns **/** 1ps

**module** test\_syn\_frame\_search**;**

// Inputs

**reg** clk**;**

**reg** rst\_n**;**

**reg** **[**7**:**0**]** data**;**

**reg** **[**7**:**0**]** i**;**

// Outputs

**wire** syn\_out\_flag**;**

// Instantiate the Unit Under Test (UUT)

test\_2 uut **(**

**.**clk**(**clk**),**

**.**rst\_n**(**rst\_n**),**

**.**data**(**data**),**

**.**syn\_out\_flag**(**syn\_out\_flag**)**

**);**

**initial** **begin**

// Initialize Inputs

clk **=** 0**;**

rst\_n **=** 0**;**

data **=** 0**;**

// Wait 100 ns for global reset to finish

**#**100**;**

**#**1000 $finish**;**

// Add stimulus here

**end**

**always** **#**3 clk**=~**clk**;**

**initial** **begin**

**#**100 rst\_n**=**1**;**

**end**

**initial**

**begin**

**for(**i**=**1**;**i**<=**70**;**i**=**i**+**1**)**

**begin**

**@(negedge** clk**)**

**if** **(**data**==**8'b0000\_0000**)**

data**<=**8'b1111\_0111**;**

**else**

data**<=**data**+**8'b0000\_0001**;**

**end**

**#**20 data**=**8'b0000\_0000**;**

**end**

**endmodule**

**八、实验小结**

通过此次实验，我对帧同步器的工作原理与设计、状态机的设计与实现方 法有了更深的理解。我们也掌握了有限状态机的转换过程。在对Quartus、Modelsim的熟悉过程中也遇到了许多问题，比如不知道如何修改和生成、导入测试文档，波形无法实现等，通过查询资料解决了困难。另外，通过 verilog 硬件描述语言，用可编程器件来实现数据流中帧同步信号的提取，更化了设备，使检测电平容易控制，提高了可靠性。

实验三：循环码(7,4)编码解码电路

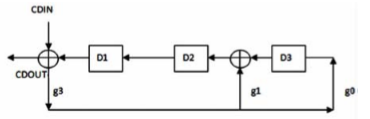
一、实验目的

实现将一组循环码输入，并且能编码，再纠错，再到译码输出的一个过程。可以提高我们对通信电路设计领域的认识，有利于培养我们在电路方面的设计能力。

二、编码器的设计

根据给定的 (n,k) 值选定生成多项式 g (x)，即从 1 n +x 的因子中选一个 (n-k） 次多项式作为，假设给定信息码组为 m(x)=m k−1 ,mk−2 ...m0 )，其次 数小于 k，则 xn-km(x)的次数必定小于 n。用 g (x) 除 xn-km(x)，得到余式 r (x)，r (x) 的次数必定小于 g (x) 的次数，即小于 (n-k)。将此余式 r (x) 加 于信息位之后作为监督位，即 r (x) 和 () xmx kn − 相加，得到多项式必定是一个 码组多项式。因为它必定能被 g (x) 整除，且商的次数不大于 (k-1)。 根据以上原理，循环码的编码步骤可以归纳如下：

1. 用 xn-k 乘信息码 m (x)，这一运算实际上是在信息码后附加上 (n-k) 个 “0”；
2. 用 g (x) 除 xn-km(x) ,得到商 Q（x）；
3. 编出的码组 T (x) = xn-km(x) +r (x)；

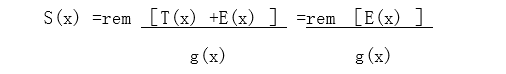
由此可见，编码的核心是如何确定余式 r (x)，找到 r (x)后，可直接将其所 代表的编码位附加到信息位之后，完成编码。编码电路可采用 （n-k） 级反馈移位 寄存器和异或门（模 2 加）组成的除法电路实现。以生成矩阵 g (x) =x^3+x+1 生成 （7，4） 循环码的编码器为例，其除法电路如下图所示： 

此移位寄存器的个数与 g (x) 的次数相等，并且 g (x) 的系数 1 表示有反馈， 0 表示无反馈。每个寄存器的初始值均为 0，将码组 m3m2m1m0 依次串行送入电路

的高端，相当于对其后面补三个零，除法电路经过四次移位后，寄存器 D1D2D3 里的数即为余式 r (x) 。因此，编出码组 T (x) = (m3m2m1m0D1D2D3)。实际设计 中，要有一个信号 K 来控制码组的输入与输出，在 K 的控制下，输入的信息码元， 一方面串行输入通道，另一方面进入除法电路进行计算。当输入末一位信息码元 结束时，K 信号也为零，在时钟脉冲作用下，移位寄存器将计算结果送往通道，即 在末一位信息码元后附加监督码。

**三、译码器的设计**

（1）检错 ：若接收码组 R(x) 与发送码组相同，即 R(x) =T(x)，则 R(x)必定能被 g(x)整除； 若在传输中发生错误，即 R(x)≠T(x)，则 R (x)被 g(x)除时可能除不尽而有余项，从而发生错误。因此，可以以余项是否为零来判断接收码组中有无错误。但是，有错码的接收码组也有可能被 g(x) 整除，这时的错码就不能检出。这种错误称为不 可检错误，其误码必定超过了此编码的检错能力。（2）纠错由于（7，4）循环码的小码距为 d0=3，由d0≥2t+1得，此循环码只能纠正一个错码。当经过通道传输后发生错误时，接收码组多项式R(x)不再是的倍式。其中 S(x)是 R(x)除以g(x)后的余式，是不大于r-1次的码组多项式，称为伴随多项式或校正子多项式。时接收码组多项式 可表示为发送码组多项式 T(x)与差错多项式 E(x)之和，即：R(x) =T(x)+E (x)综合以上式子，确定 E(x)：



由S(x)确定E(x)时同样使用大似然比准则。对小码重的差错多项式 E(x)，由上式求出对应的伴随多项式 S(x)，将 E(x)与 S(x)的对应关系列成译码表。当收到任一码组 R(x)后,利用 S(x)=rem［R(x)/g(x)］求出 S(x)，对照译码表找到 E (x)，再用 R(x)=T(x)+E(x)求T(x)，即T(x)=R(x)+E(x)为已经纠错的原发送码组。纠错完毕之后，只需要取纠错后的前四位，即为所译码元。而在实际电路设计过程中，由于二进制只有0与1两种状态，知道错误位置后，可以直接对该位置的码元取反即可。综上所述，纠错的步骤如下：把R（x) 送入除法电路,得到余数D1D2D3即为S(x)。由典型生成 矩阵G和典型监督矩阵 HT 之间的关系可进一步求得，再由S=E.HT得 E(x)，这样就可以确定错码的位置。根据错误的码位，对该码位的码元取反输出便可得到原发送码组 T(x)。其中，T(x)的前四位即为译出码元 m(x)。

四、实验过程

1.新建工程，为工程选择路径和命名；

2.选择 File→New→Verilog HDL File，点击 OK。在新建的.v文件中输入编码程序并保存。点击工具栏图标，对程序进行编译；

1. 点击 Processing→Start→Start Test Bench Template Writer，在弹出的对话 框中点击 OK。
2. 点击 File→Open，打开之前写好的后缀为 vt 的测试文件；

5.点击Assignments→Settings→EDA Tool Settings→Simulation→Compile test bench 下，点击 Test Benches，添加所写的测试程序；

6. 点击按钮，进行RTL仿真，观察并分析编码仿真结果。

7. 重复上述 1~6 步骤，进行译码仿真。

五、程序设计

程序流程分析：

该程序流程主要通过输入一组四位信息码组，然后根据编码后的余数输出对编

码进行检错判断，若输出余数为 000 则无错码，若输出的余数为下图中，则有错码。

然后纠错并译码输出一个7位信息码组。

实验代码：

代码1：

**module** test**(** clk**,** rst**,** start**,** d\_finish**,** datain**,** dataout **);**

**input** clk**;**

**input** rst**;**

**input** start**;**

**input** datain**;**

**input** d\_finish**;**

**output** **reg** dataout**;**

**reg[**2**:**0**]**cyclic\_reg**;**

**reg[**1**:**0**]**state**;**

**reg[**1**:**0**]**count**;**

**parameter** idle**=**2'b00**;**

**parameter** compute**=**2'b01**;**

**parameter** finish**=**2'b10**;**

**always@(posedge** clk **or** **negedge** rst**)**

**begin**

**case(**state**)**

idle**:begin**

**if(**start**)**

state**<=**compute**;**

**else**

state**<=**idle**;**

**end**

compute**:begin**

**if(**d\_finish**)**

state**<=**finish**;**

**else**

state**<=**compute**;**

**end**

finish**:begin**

**if(**count**==**3**)**

state**<=**idle**;**

**else**

count**<=**count**+**1**;**

**end**

**endcase**

**if(!**rst**)**

**begin**

cyclic\_reg**[**2**:**0**]<=**3'b000**;**

count**<=**2'b00**;**

state**<=**idle**;**

**end**

**else**

**case(**state**)**

idle**:begin**

cyclic\_reg**[**2**:**0**]<=**3'b000**;**

**end**

compute**:begin**

cyclic\_reg**[**0**]<=**cyclic\_reg**[**2**]^**datain**;**

cyclic\_reg**[**1**]<=**cyclic\_reg**[**0**]^**cyclic\_reg

**[**2**]^**datain**;**

cyclic\_reg**[**2**]<=**cyclic\_reg**[**1**];**

dataout**<=**datain**;**

**end**

finish**:begin**

dataout**<=**cyclic\_reg**[**2**];**

cyclic\_reg**[**2**:**0**]<={**cyclic\_reg**[**1**:**0**],**1'b0**}**

**;**

**end**

**endcase**

**end**

**endmodule**

代码2：

**module** cyclic\_decoder**(**

clk**,**

start**,**

d\_finish**,**

datain**,**

outbuf**,**

dataout**,**

**);**

**input** clk**;**

**input** start**;**

**input** datain**;**

**input** d\_finish**;**

**output** **reg** **[**3**:**0**]** dataout**;**

**output** **reg** **[**6**:**0**]** outbuf**;**

**reg** **[**2**:**0**]** cyclic\_reg**=**3'b000**;**

**reg** **[**1**:**0**]** state**=**2'b00**;**

**reg** **[**2**:**0**]** rout**;**

**reg** **[**6**:**0**]** databuf**=**7'b0000000**;**

**integer** index**=**6**;**

**integer** done**=**0**;**

**parameter** idle**=**2'b00**;**

**parameter** compute**=**2'b01**;**

**parameter** finish**=**2'b10**;**

**parameter** none**=**3'b000**;**

**parameter** first**=**3'b011**;**

**parameter** second**=**3'b110**;**

**parameter** third**=**3'b111**;**

**parameter** fourth**=**3'b101**;**

**parameter** fifth**=**3'b001**;**

**parameter** sixth**=**3'b010**;**

**parameter** seventh**=**3'b100**;**

**always@(posedge** clk**)**

**begin**

**case(**state**)**

idle**:begin**

**if(**start**)**

state **<=** compute**;**

**else**

state **<=** idle**;**

**end**

compute**:begin**

**if(**d\_finish**)**

state **<=** finish**;**

**else**

state **<=** compute**;**

**end**

finish**:begin**

state **<=** finish**;**

**end**

**endcase**

**end**

**always@(posedge** clk**)**

**begin**

**case(**state**)**

idle**:begin**

dataout**[**3**:**0**]** **<=**

4'b0000**;**

outbuf**[**6**:**0**]** **<=**

7'b0000000**;**

databuf**[**6**]** **<=** datain**;**

cyclic\_reg**[**0**]** **<=**

cyclic\_reg**[**2**]** **^** datain**;**

cyclic\_reg**[**1**]** **<=**

cyclic\_reg**[**0**]** **^** cyclic\_reg**[**2**]** **^** datain**;**

cyclic\_reg**[**2**]** **<=**

cyclic\_reg**[**1**];**

index**=**5**;**

**end**

compute**:begin**

cyclic\_reg**[**0**]** **<=**

cyclic\_reg**[**2**]** **^** datain**;**

cyclic\_reg**[**1**]** **<=**

cyclic\_reg**[**0**]** **^** cyclic\_reg**[**2**]** **^** datain**;**

cyclic\_reg**[**2**]** **<=**

cyclic\_reg**[**1**];**

databuf**[**index**]** **<=**

datain**;**

index **<=** index**-**1**;**

**end**

finish**:begin**

rout**[**0**]** **<=**

cyclic\_reg**[**0**];**

rout**[**1**]** **<=**

cyclic\_reg**[**1**];**

rout**[**2**]** **<=**

cyclic\_reg**[**2**];**

done**=**1**;**

**end**

**endcase**

**if(**done**==**1**)**

**begin**

**case(**rout**)**

none**:begin**

outbuf**[**6**:**0**]** **<=**

databuf**[**6**:**0**];**

**end**

first**:begin**

outbuf**[**6**:**0**]** **<=**

databuf**[**6**:**0**]^**7'b0000001**;**

**end**

second**:begin**

outbuf**[**6**:**0**]** **<=**

databuf**[**6**:**0**]^**7'b0000010**;**

**end**

third**:begin**

outbuf**[**6**:**0**]** **<=**

databuf**[**6**:**0**]^**7'b0000100**;**

**end**

fourth**:begin**

outbuf**[**6**:**0**]** **<=**

databuf**[**6**:**0**]^**7'b0001000**;**

**end**

fifth**:begin**

outbuf**[**6**:**0**]** **<=**

databuf**[**6**:**0**]^**7'b0010000**;**

**end**

sixth**:begin**

outbuf**[**6**:**0**]** **<=**

databuf**[**6**:**0**]^**7'b0100000**;**

**end**

seventh**:begin**

outbuf**[**6**:**0**]** **<=**

databuf**[**6**:**0**]^**7'b1000000**;**

**end**

**endcase**

dataout**[**3**:**0**]** **<=**outbuf**[**6**:**3**];**

**end**

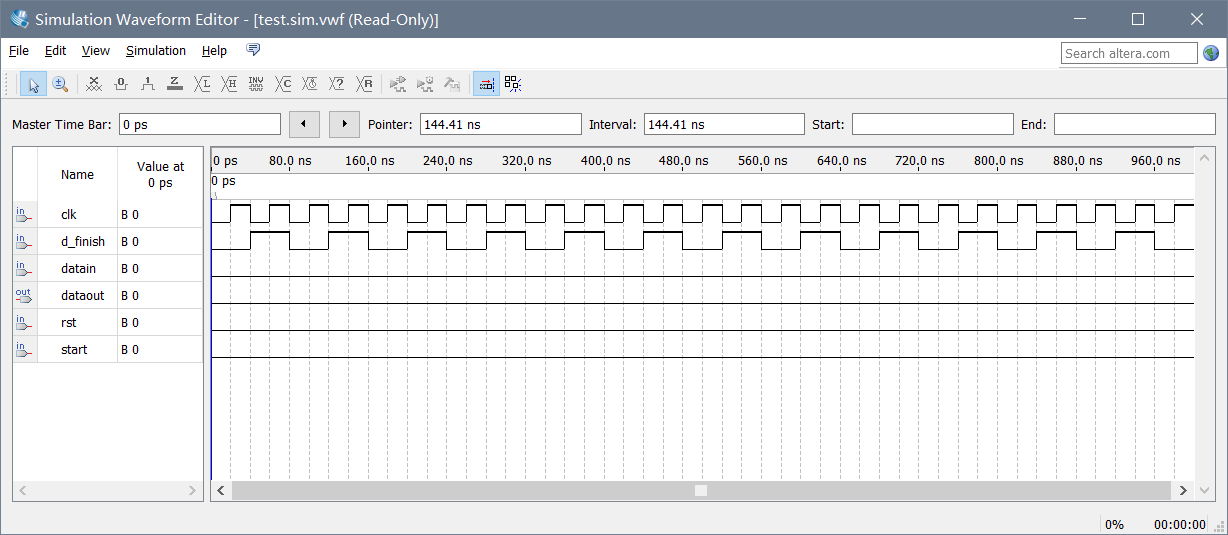
**end**

**endmodule**

六、系统仿真

系统仿真是在实际系统上进行实验研究比较困难时适用的必不可少的工具，它是指通过系统模型实验去研究一个已经存在或正在设计的系统的过程，通俗地讲，就是进行模型实验。因而，系统仿真的结果决定整个课程设计任务完成的到位程度。程序输入完成后进行编译，编译完成后，可以对所进行的设计进行仿真，本课程设计的仿真平台是 Quartus2Ⅱ,通过对源程序进行编译检错，然后创建波形档，加入输入输出变量，选择适用的芯片以及设定仿真结束时间，设置好输入初值进行仿真，得到仿真波形图如图所示：

## 实验结果

编码：

译码：