

数字系统综合设计

实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| **作 者:** | 陶桢 | **学 号：** | 9191040G0428 |
| **学 院:** | 电子工程与光电技术学院 | | |
| **专业(方向):** | 通信工程 | | |
| **班 级:** | 9191040G04 | | |
| **设计名称:** | 直接数字频率合成器 | | |
| **指导老师:** | 姜萍 | | |

**摘要**

|  |
| --- |
| 本次实验利用直接数字频率合成器（*DDS*）技术原理，基于 *Xilinx EGO1 Artix-7 FPGA* 硬件平台设计了一个直接数字频率合器。通过在Xilinx公司的*vivado*软件平台上进行仿真、调试，最后将程序烧录进开发板上，使开发板可以产生方波、正弦波、三角波和锯齿波四种不同的波形，并且通过*EGO1*上集成的8位模数转换芯片（*DAC0832*）使波形可以在示波器上进行观测。同时，该合成器还实现了测量频率和理论频率的功能。  该合成器通过8位拨码开关控制输出的频率，8位七段数码管显示实际测量的频率和理论测量的频率，8位*DIP*开关的前四位控制输出的波形。  **关键字： DDS、Verilog、EGO1、Xilinx、vivado** |

***Abstract***

|  |
| --- |
| In this experiment, a direct digital frequency synthesizer was designed based on the Xilinx EGO1 Artix-7 FPGA hardware platform using the principle of direct digital frequency synthesizer (DDS) technology. Through simulation and debugging on Xilinx's vivado software platform, and finally burning the program into the development board, the development board can generate four different waveforms: square wave, sine wave, triangle wave and sawtooth wave, and integrate them through EGO1 The 8-bit analog-to-digital conversion chip (DAC0832) enables the waveform to be observed on the oscilloscope. At the same time, the synthesizer also realizes the function of measuring frequency and theoretical frequency.  The synthesizer controls the output frequency through an 8-bit DIP switch, an 8-bit seven-segment digital tube displays the actual measured frequency and the theoretically measured frequency, and the first four bits of the 8-bit DIP switch control the output waveform.  **Keywords:** DDS、Verilog、EGO1、Xilinx、vivado |

**目录**

[**1 设计要求说明**](#_Toc742_WPSOffice_Level1) **01**

[1.1 具体要求说明](#_Toc1535_WPSOffice_Level2) 01

[**2 方案论证**](#_Toc1535_WPSOffice_Level1) **02**

[2.1 直接数字频率合成器](#_Toc16100_WPSOffice_Level2) 02

[2.2 *DDS*工作原理](#_Toc31064_WPSOffice_Level2) 02

[**3 各子模块设计原理**](#_Toc22045_WPSOffice_Level1) **04**

[3.1 时钟信号产生模块](#_Toc14363_WPSOffice_Level2) 04

[3.1.1 设计原理](#_Toc1535_WPSOffice_Level3) 04

[3.1.2 模块代码](#_Toc22045_WPSOffice_Level3) 04

[3.1.3 仿真结果](#_Toc16100_WPSOffice_Level3) 06

[3.2 累加器模块](#_Toc28459_WPSOffice_Level2) 06

[3.2.1 设计原理](#_Toc31064_WPSOffice_Level3) 06

[3.2.2 模块代码](#_Toc11025_WPSOffice_Level3) 06

[3.2.3 仿真结果](#_Toc29785_WPSOffice_Level3) 07

[3.3 数码管显示模块](#_Toc26810_WPSOffice_Level2) 07

[3.3.1 设计原理](#_Toc14363_WPSOffice_Level3) 07

[3.3.2 模块代码](#_Toc7480_WPSOffice_Level3) 07

[3.3.3 仿真结果](#_Toc28459_WPSOffice_Level3) 10

[3.4 二进制转换成十进制*BCD*码](#_Toc5928_WPSOffice_Level2) 10

[3.4.1 设计原理](#_Toc5928_WPSOffice_Level3) 10

[3.4.2 模块代码](#_Toc26395_WPSOffice_Level2) 11

[3.4.3 仿真结果](#_Toc20658_WPSOffice_Level2) 13

[3.5 理论频率计算模块](#_Toc14759_WPSOffice_Level2) 13

[3.5.1 设计原理](#_Toc16480_WPSOffice_Level3) 13

[3.5.2 模块代码](#_Toc13489_WPSOffice_Level3) 13

[3.5.3 仿真结果](#_Toc10323_WPSOffice_Level3) 14

[3.6 测频模块](#_Toc16480_WPSOffice_Level2) 14

[3.6.1 设计原理](#_Toc25245_WPSOffice_Level3) 15

[3.6.2 模块代码](#_Toc8030_WPSOffice_Level3) 15

[3.6.3 仿真结果](#_Toc25779_WPSOffice_Level3) 16

[3.7 波形选择模块](#_Toc8030_WPSOffice_Level2) 16

[3.7.1 设计原理](#_Toc9879_WPSOffice_Level3) 16

[3.7.2 模块代码](#_Toc9879_WPSOffice_Level3) 17

[3.7.3 仿真结果](#_Toc23856_WPSOffice_Level3) 18

[3.8 *D/A*转换器模块](#_Toc16480_WPSOffice_Level2) 18

[3.8.1 设计原理](#_Toc25245_WPSOffice_Level3) 19

[3.8.2 模块代码](#_Toc8030_WPSOffice_Level3) 19

[3.8.3 示波器观测](#_Toc25779_WPSOffice_Level3) 20

[3.9 顶层模块](#_Toc16480_WPSOffice_Level2) 21

[3.9.1 设计原理](#_Toc25245_WPSOffice_Level3) 21

[3.9.2 模块代码](#_Toc8030_WPSOffice_Level3) 22

[3.9.3 仿真结果](#_Toc25779_WPSOffice_Level3) 23

[3.9.4 仿真程序](#_Toc16480_WPSOffice_Level2) 24

[**4 引脚分配**](#_Toc28358_WPSOffice_Level3) **25**

[**5 结论**](#_Toc23376_WPSOffice_Level3) **29**

[5.1 问题与反思](#_Toc9879_WPSOffice_Level2) 29

[5.1.1 数码管显示问题](#_Toc25245_WPSOffice_Level3) 29

[5.1.2 二进制转换成*BCD*码](#_Toc8030_WPSOffice_Level3) 29

[5.2 收获与感受](#_Toc23856_WPSOffice_Level2) 30

[**6 致谢**](#_Toc11648_WPSOffice_Level3) **31**

[**7 参考文献**](#_Toc15478_WPSOffice_Level3) **32**

# 1 设计要求说明

利用*Vivado FPGA*设计软件平台、*EGO1Aritix-*7实验板卡硬件平台，使用*Verilog HDL*设计一个能产生多种信号（如正弦波、三角波、矩形波、锯齿波等）的直接数字频率合成器*（Direct Digital Synthesizer）*，采用自顶向下的模块化方法进行设计，并完成频率计算与测量电路的设计。

## 1.1 具体要求说明

1. 使用8位拨码开关实现频率控制字的输入，控制输出信号的频率；

2. 通过实验板卡上的七段显示器显示频率，左边4个数码管显示实际测量频率，右边4个数码管显示理论频率；

3. 通过8位*D/A*转换器输出目标波形：正弦波，三角波，方波，锯齿波。由DIP开关控制波形的选择。

# 2 方案论证

## 2.1 直接数字频率合成器

*DDS*同*DSP*（数字信号处理）一样，也是一项关键的数字化技术。DDS是直接数字式频率合成器（*Direct Digital Synthesizer*）的英文缩写。*DDS*是从相位概念出发直接合成所需要波形的一种新的频率合成技术。与传统的频率合成器相比，*DDS*具有低成本、低功耗、高分辨率和快速转换时间等优点，广泛使用在电信与电子仪器领域，是实现设备全数字化的一个关键技术。

## 2.2 DDS工作原理

*DDS*技术是一种全新的频率合成方法，最早由*JOSEPH TIEREN*Y等三人提出,其具有输出频率分辨率高、功耗低、频率切换速度快且频率切换时输出信号的相位连续的特点。为此，了解*DDS*模块对数字信号处理及其硬件实现有着很重要的作用。

*DDS*的结构主要由相位累加器、波形存储器、数模(*D/A*)转换器和低通滤波器等四个大的结构组成。其结构框图如下

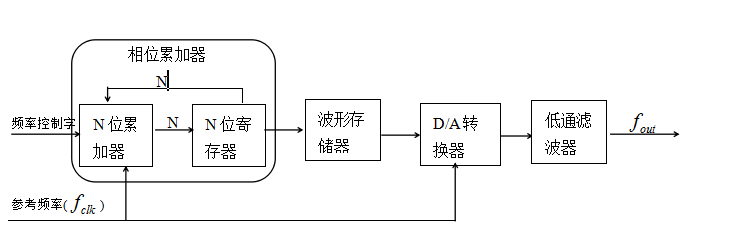


图2.1 *DDS*工作原理图

图中，相位累加器是由*N*位加法器与*N*位累加寄存器构成，它是*DDS*模块中一个极其重要的部分。在参考频率时钟的驱动下，*DDS*模块开始工作；当每来一个参考时钟时，累加器就把频率控制字*K*与寄存器输出的值进行累加，将相加后的结果再输入到寄存器中，而累加寄存器就将在上一个参考时钟作用时产生的数据通过反馈的方式输送到累加器中。这样，在时钟的作用下，就可以不停的对频率控制字进行累加。此时，用相位累加器输出的数据作为地址在波形存储器中通过查找地址所对应的幅值表，就可以完成其从相位到幅值之间的转化。

由于累加器为*N*位，相当于把正弦信号在相位上的精度定为*N*位，所以分辨率为。若系统时钟频率为，频率控制字为*K*，则输出频率。当系统输入时钟频率不变时，输出信号的频率由频率控制字*K*所决定。当*K* =1时，输出最低频率为，而最高输出频率由*Nyquist*采样定理决定，即当时，输出最高频率为，在本次实验中*N*=。

在选取*ROM*地址方面，我们选取8位地址建立正弦函数的*ROM*，在参考时钟 *CLK*控制下，频率控制字*K*完成加运算，并作为下一次加运算的输入值，累加器输出数据作为*ROM*的相位抽样地址值，查找*ROM*中相对应单元的电压幅值，得到波形二进制编码，实现相位到电压幅值的转变。波形二进制编码再通过*D/A*转换器，把数字信号转换成相应的模拟信号，最后可以通过示波器观测到波形。

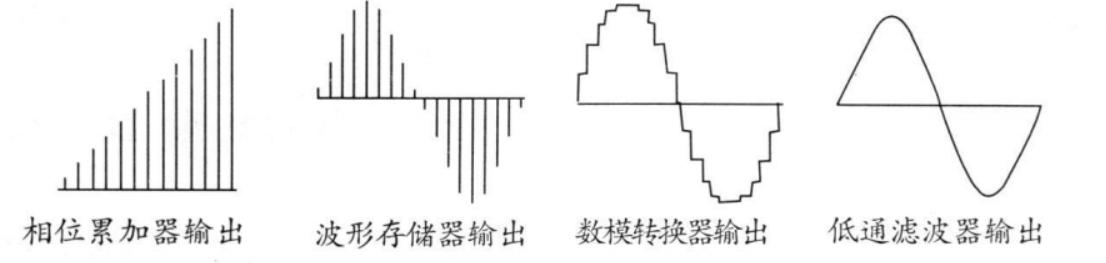


图2.2 *DDS*工作流程图

# 3 各模块及设计原理

## 3.1 时钟信号产生模块

本次实验需要使用到两个不同频率的时钟信号，分别为10*kHz*和0.5*Hz*。其中10*kHz*时钟信号主要为后续模块提供脉冲信号，0.5*Hz*时钟信号主要作为测频模块的基准脉冲信号。这两个时钟信号都可以由*EGO1*开发板上自带的100*MHz*时钟信号经过分频得到，但是经过实测，发现如果直接由100*MHz*时钟信号分频得到0.5*Hz*的时钟信号，可能导致需要分频的次数过多，造成错误。所以这里选择了先将100*MHz*的时钟信号分频得到10*kHz*的时钟信号，再通过10*kHz*的时钟信号得到0.5*Hz*的时钟信号。

### 3.1.1 设计原理

初始信号*signal*先设置成低电平，同时引入一个信号反转的标志位*flag*来提示*signal*应该取低电平还是高电平，当*flag*等于1的时候*signal*取低电平，当*flag*等于0时*signal*取高电平。通过设置一个20位*reg*类型的中间计数的变量*cnt*，每次100*MHz*时钟信号的上升沿来的时候计数一次。当100*MHz*的脉冲信号上升沿达到了*div*次的时候就使信号标志位*flag*发生反转，同时将*cnt*清零重新开始计数。这样，当变量*div*等于5000时，100*MHz*的时钟信号就别分为10*kHz*的时钟信号。产生0.5*Hz*的时钟信号同理。

### 3.1.2 模块代码

//产生10KHz的输出信号

**module** clock**(**

**input** clk**,** //输入100MHz时钟信号

**output** **reg** signal //输出10kHz时钟信号

**);**

**reg** **[**19**:**0**]** cnt **=** 0**;** //中间变量计数

**reg** flag **=** 0**;** //信号反转标志

**reg** signal **=** 0**;** //初始信号从低电平开始

**parameter** div **=** 5000**;**

//分频，最终分出来的频率为100MHz/(div\*2)

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

cnt **<=** cnt **+** 1**;**

//当计数cnt达到div时，信号发生反转，同时计数cnt清零

**if(**cnt **==** div**)** **begin**

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

**if(**flag **==** 0**)**

signal **<=** 1'b1**;**

**else**

signal **<=** 1'b0**;**

flag **<=** **!**flag**;**

**end**

**end**

**endmodule**

//产生0.5Hz的输出信号

**module** clock\_1**(**

**input** clk**,** //输入100MHz时钟信号

**output** **reg** signal\_1 //输出0.5Hz时钟信号

**);**

**wire** signal**;** //接受10kHz的时钟信号

//先产生10kHz时钟信号

clock u0**(**

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

**.**signal**(**signal**)**

**);**

**reg** **[**19**:**0**]** cnt **=** 0**;**

**reg** flag **=** 0**;**

**reg** signal\_1 **=** 0**;**

**parameter** div **=** 10000**;**

//分频

**always** **@(posedge** signal**)** **begin**

cnt **<=** cnt **+** 1**;**

**if(**cnt **==** div**)** **begin**

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

**if(**flag **==** 0**)**

signal\_1 **<=** 1'b1**;**

**else**

signal\_1 **<=** 1'b0**;**

flag **<=** **!**flag**;**

**end**

**end**

**endmodule**

### 3.1.3 仿真结果

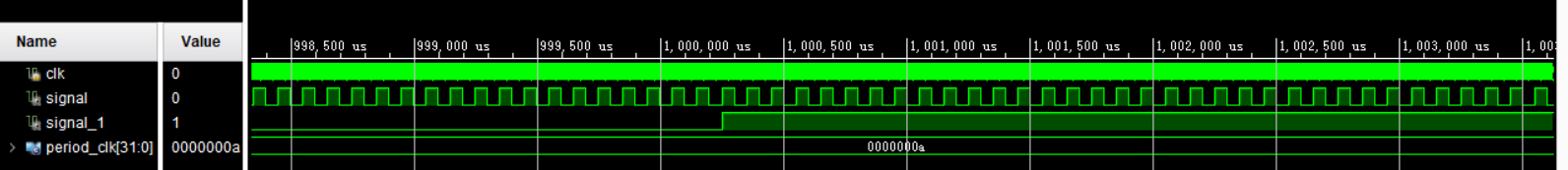


图3.1 时钟信号仿真结果

## 3.2 累加器模块

累加器通过由8位拨动开关控制的频率控制字，对*N*位寄存器进行抽样取值，最后输出*ROM*的取样地址，这样就可把存储在*ROM*内的波形抽样值查出，完成相位到幅值的转换。

### 3.2.1 设计原理

累加器输入10*kHz*时钟信号和频率控制字，输出*ROM*的地址，由8位拨动开关控制输入频率控制字*k*的大小，在时钟信号的控制下对地址*address*进行累加，当满足地址*address*超过8位时，完成一个周期的取样，同时地址*address*清零，进行下一个周期的取样。

### 3.2.2 模块代码

//累加模块，得到ROM地址

**module** add**(**

**input** signal**,** //输入10kHz时钟信号

**input** **wire** **[**7**:**0**]** k**,** //输入8位拨动开关

**output** **reg** **[**7**:**0**]** address //输出ROM地址

**);**

**reg** **[**7**:**0**]** address **=** 0**;** //ROM地址初值为0

//当ROM地址小于1111\_1111时，进行累加

**always** **@(posedge** signal**)** **begin**

**if(!(**address **>=** 8'b1111\_1111**))** **begin**

address **<=** address **+** k**[**7**:**0**];**

**end**

**else**

address **<=** 0**;**

**end**

**endmodule**

### 3.2.3 仿真结果

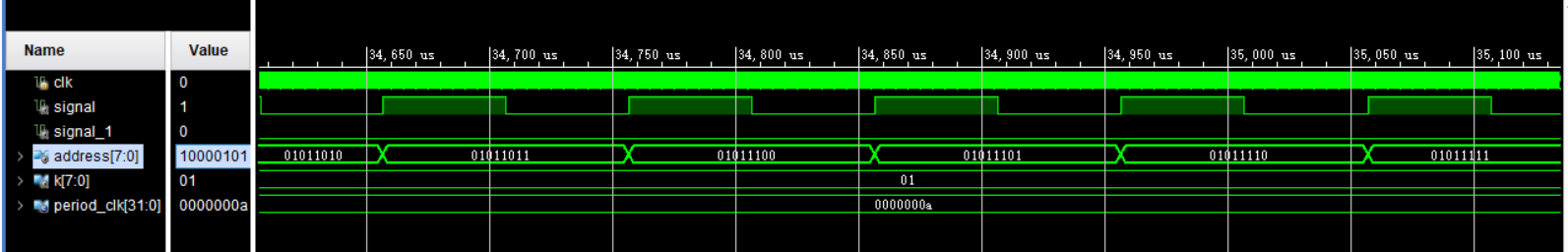


图3.2 累加器仿真

## 3.3 数码管显示模块

*EGO1*上的8位七段共阴数码管，通过段选信号来控制哪位数码管工作，通过位选来确定数码管显示的具体数值。同时，通过间隔极小的不断扫描来达到8位数码管看起来是同时亮起的效果，达到动态显示的效果。这里，将8位数码管分为左四位和右四位来分别控制。

### 3.3.1 设计原理

输入一个16位十进制*BCD*码，变量*sel*控制段选信号由0至4的循环，使用*data*接收输入的16位十进制*BCD*码，当时钟脉冲信号来的时候，显示*data*的后四位，然后右移四位。一个周期过后，*sel*清零，同时*data*重新接收十进制*BCD*码，继续下一个周期的显示，这样重复扫描，形成了动态显示。

### 3.3.2 模块代码

//控制左边4个七段数码管显示实际测频出来的频率值

**module** display\_left**(**

**input** **[**15**:**0**]** number\_1**,**

**input** signal**,**

**output** **reg[**3**:**0**]** dis\_duan\_1**,**

**output** **reg[**6**:**0**]** dis\_wei\_1

**);**

//控制段选信号

**reg** **[**2**:**0**]** sel **=** 0**;**

//接受bin\_to\_bcd转换后的bcd码

**reg** **[**15**:**0**]** data**;**

//控制左端四个数码管显示测频值

**always** **@(posedge** signal**)** **begin**

**if(**sel **==** 5**)** **begin**

sel **<=** 0**;**

**end**

**else** **if(**sel **==** 0**)** **begin**

data **<=** number\_1**;**

sel **<=** sel **+** 1**;**

**end**

**else** **begin**

**case** **(**sel**)**

1**:** dis\_duan\_1 **<=** 4'b0001**;**

2**:** dis\_duan\_1 **<=** 4'b0010**;**

3**:** dis\_duan\_1 **<=** 4'b0100**;**

4**:** dis\_duan\_1 **<=** 4'b1000**;**

**default:** dis\_duan\_1 **<=** 4'b0000**;**

**endcase**

**case** **(**data**[**3**:**0**])**

0**:** dis\_wei\_1 **<=** 7'b1111110**;**

1**:** dis\_wei\_1 **<=** 7'b0110000**;**

2**:** dis\_wei\_1 **<=** 7'b1101101**;**

3**:** dis\_wei\_1 **<=** 7'b1111001**;**

4**:** dis\_wei\_1 **<=** 7'b0110011**;**

5**:** dis\_wei\_1 **<=** 7'b1011011**;**

6**:** dis\_wei\_1 **<=** 7'b1011111**;**

7**:** dis\_wei\_1 **<=** 7'b1110000**;**

8**:** dis\_wei\_1 **<=** 7'b1111111**;**

9**:** dis\_wei\_1 **<=** 7'b1111011**;**

**default:** dis\_wei\_1 **<=** 7'b00000000**;**

**endcase**

sel **<=** sel **+** 1**;**

data **<=** data **>>** 4**;**

**end**

**end**

**endmodule**

**module** display\_right**(**

**input** **[**15**:**0**]** number**,**

**input** signal**,**

**output** **reg[**3**:**0**]** dis\_duan**,**

**output** **reg[**6**:**0**]** dis\_wei**,**

**output** **reg[**3**:**0**]** result

**);**

**reg** **[**2**:**0**]** sel **=** 0**;**

**reg** **[**15**:**0**]** data**;**

**always** **@(posedge** signal**)** **begin**

**if(**sel **==** 5**)** **begin**

sel **<=** 0**;**

**end**

**else** **if(**sel **==** 0**)** **begin**

data **<=** number**;**

sel **<=** sel **+** 1**;**

**end**

**else** **begin**

**case** **(**sel**)**

1**:** dis\_duan **<=** 4'b0001**;**

2**:** dis\_duan **<=** 4'b0010**;**

3**:** dis\_duan **<=** 4'b0100**;**

4**:** dis\_duan **<=** 4'b1000**;**

**default:** dis\_duan **<=** 4'b0000**;**

**endcase**

**case** **(**data**[**3**:**0**])**

0**:** dis\_wei **<=** 7'b1111110**;**

1**:** dis\_wei **<=** 7'b0110000**;**

2**:** dis\_wei **<=** 7'b1101101**;**

3**:** dis\_wei **<=** 7'b1111001**;**

4**:** dis\_wei **<=** 7'b0110011**;**

5**:** dis\_wei **<=** 7'b1011011**;**

6**:** dis\_wei **<=** 7'b1011111**;**

7**:** dis\_wei **<=** 7'b1110000**;**

8**:** dis\_wei **<=** 7'b1111111**;**

9**:** dis\_wei **<=** 7'b1111011**;**

**default:** dis\_wei **<=** 7'b00000000**;**

**endcase**

sel **<=** sel **+** 1**;**

result **<=** data**[**3**:**0**];**

data **<=** data **>>** 4**;**

**end**

**end**

**endmodule**

### 3.3.3 仿真结果

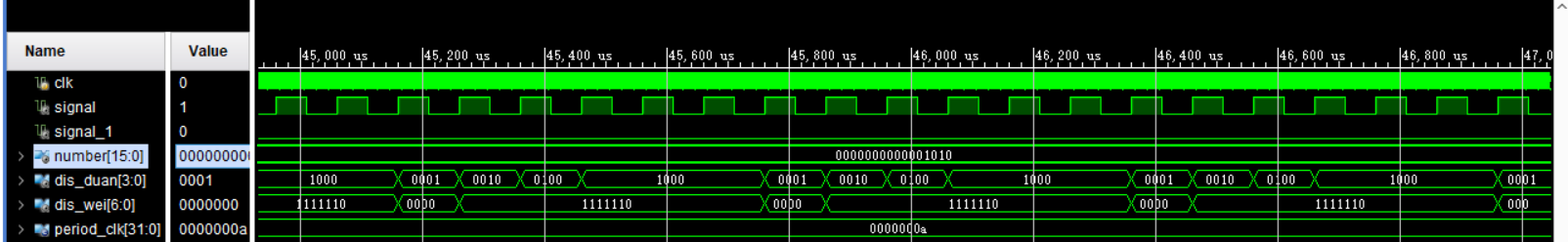


图3.3 数码管显示仿真

## 3.4 二进制转换成十进制BCD码模块

*Verilog*不同于其他的高级语言，*Verilog*的所有变量都是以二进制的形式存储的。所以在计算得到一个二进制数之后，如果需要将该数据显示在数码管上，就需要将这个数据拆分成十进制的*BCD*码的形式，每个位单独显示在数码管上。传统的二进制转换成*BCD*码的方法是转换成对应的十进制数后，进行除法取模操作，但是由于除法取模操作消耗的资源太多，所以这里选择了另一种消耗资源比较少的方法，加三移位（*double**Dabble*）。

### 3.4.1 设计原理

4位二进制数最高可表示到15，4位*BCD*码最高可表示到9，所以，当4位二进制数想转换成4位*BCD*时，只要令4位二进制数大于9的时候向高位进1就可以实现，大于9进一这个操作可以通过加上二进制的6实现。例如：二进制的10（1010），转换成*BCD*码就是（10000），而1010+0110=10000。对于更多位的二进制数转换成*BCD*码可以通过左移后重复上述操作来实现转化。

加三移位法就是由上面的操作演变而来，先加三后移位等同于先移位后加六。由于4+6=10，产生了进位，所以在移位之前判断4位二进制码是否小于4，如果大于4则加3进一，让4位二进制码始终保持在小于9的状态。

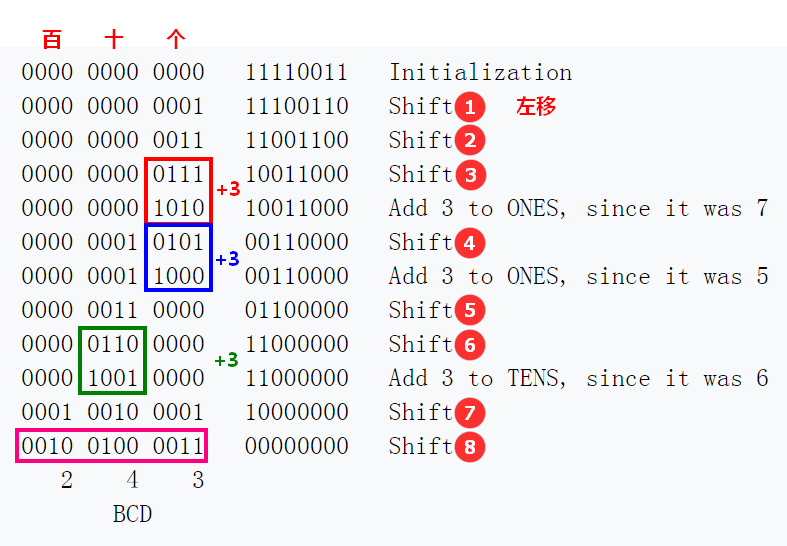


图3.4 *BCD*码转换原理图

### 3.4.2 模块代码

//十进制BCD转换模块，将二进制数转换成十进制BCD码

**module** bin\_to\_bcd**(**

**input** signal**,**

**input** **[**13**:**0**]** bin**,**

**output** **reg** **[**15**:**0**]** bcd

**);**

**localparam** S0\_IDLE **=** 0**;**

**localparam** S1\_SHIFT **=** 1**;**

**localparam** S2\_ADD **=** 2**;**

**localparam** S3\_FINISH **=** 4**;**

**reg** **[**3**:**0**]** fsm **=** S0\_IDLE**;**

**reg** **[**3**:**0**]** cnt\_shift **=** 0**;**

**reg** **[**13**:**0**]** bin\_buf **=** 0**;**

**reg** **[**15**:**0**]** bcd\_buf **=** 0**;**

**always** **@** **(posedge** signal**)** **begin**

**case(**fsm**)**

S0\_IDLE**:** **begin**

**if(**signal**)** **begin**

fsm **<=** S1\_SHIFT**;**

bin\_buf **<=** bin**;**

cnt\_shift **<=** 'h0**;**

bcd\_buf **<=** 'h0**;**

**end**

**end**

S1\_SHIFT**:** **begin**

**if(**cnt\_shift **!=** 14**)** **begin**

cnt\_shift **<=** cnt\_shift **+** 1**;**

**{**bcd\_buf**,** bin\_buf**}** **<=** **{**bcd\_buf**,** bin\_buf**}** **<<** 1**;**

fsm **<=** S2\_ADD**;**

**end**

**else**

fsm **<=** S3\_FINISH**;**

**end**

S2\_ADD**:** **begin**

**if(**cnt\_shift **!=** 14**)** **begin**

fsm **<=** S1\_SHIFT**;**

**if(**bcd\_buf**[**15**:**12**]** **>** 4**)**

bcd\_buf**[**15**:**8**]** **<=** bcd\_buf**[**15**:**8**]** **+** 'd3**;**

**if(**bcd\_buf**[**11**:**8**]** **>** 4**)**

bcd\_buf**[**11**:**8**]** **<=** bcd\_buf**[**11**:**8**]** **+** 'd3**;**

**if(**bcd\_buf**[**7**:**4**]** **>** 4**)**

bcd\_buf**[**7**:**4**]** **<=** bcd\_buf**[**7**:**4**]** **+** 'd3**;**

**if(**bcd\_buf**[**3**:**0**]** **>** 4**)**

bcd\_buf**[**3**:**0**]** **<=** bcd\_buf**[**3**:**0**]** **+** 'd3**;**

**end**

**else** **begin**

fsm **<=** S3\_FINISH**;**

bcd **<=** bcd\_buf**;**

**end**

**end**

S3\_FINISH**:** **begin**

fsm **<=** S0\_IDLE**;**

bin\_buf **<=** 'h0**;**

cnt\_shift **<=** 'h0**;**

bcd\_buf **<=** 'h0**;**

**end**

**endcase**

**end**

**endmodule**

### 3.4.3 仿真结果

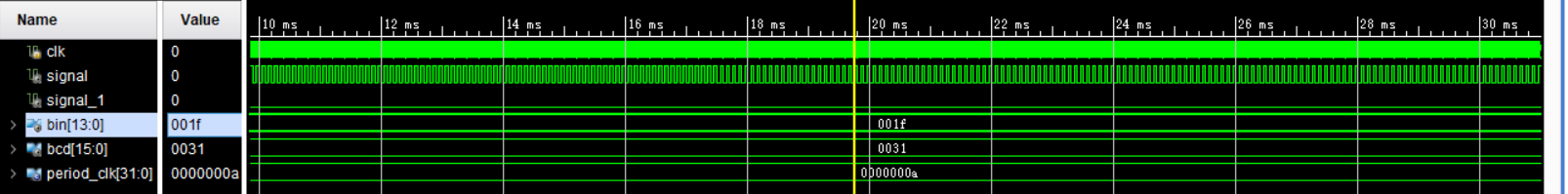


图3.5 *BCD*码转换仿真

## 3.5 理论频率计算模块

输出信号频率的理论值仅和频率控制字 *K* 和时钟频率大小 *f*，以及 *ROM* 的深度有关系，通过分析不难得出输出信号的理论频率为：。

### 3.5.1 设计原理

储存理论二进制值的变量*bin*，给予足够大的位数以至于不会产生溢出。通过公式计算出理论频率后，调用二进制转换成十进制*BCD*码模块将*bin*转换成*BCD*码，然后将*BCD*码传递给显示模块在数码管上进行显示。

### 3.5.2 模块代码

//理论测频模块，计算频率理论值并在右端4个数码管显示理论值

**module** th\_fre**(**

**input** signal**,** //提供10kHz时钟信号

**input** **wire** **[**7**:**0**]** k**,** //8位可拨动开关的输入值

**output** **wire** **[**15**:**0**]** bcd**,** //输出理论值的十进制BCD码

**output** **wire** **[**7**:**0**]** dis\_duan**,** //输出右端数码管的段选信号

**output** **wire** **[**6**:**0**]** dis\_wei //输出右端数码管的位选信号

**);**

**reg** **[**13**:**0**]** bin**;** //储存理论值的二进制数

//当10kHz时钟信号上升沿来的时候，开始计算理论值，最大为5000

**always** **@(posedge** signal**)** **begin**

**if** **(**k**\***10000**/**256 **>** 5000**)** **begin**

bin **<=** 5000**;**

**end**

**else** **begin**

bin **<=** k**\***10000**/**256**;**

**end**

**end**

//将理论值转换成十进制BCD码

bin\_to\_bcd b0**(**

**.**signal**(**signal**),**

**.**bin**(**bin**),**

**.**bcd**(**bcd**)**

**);**

//右端显示理论值

display\_right d0**(**

**.**signal**(**signal**),**

**.**dis\_duan**(**dis\_duan**),**

**.**dis\_wei**(**dis\_wei**),**

**.**number**(**bcd**)**

**);**

**Endmodule**

### 3.5.3 仿真结果

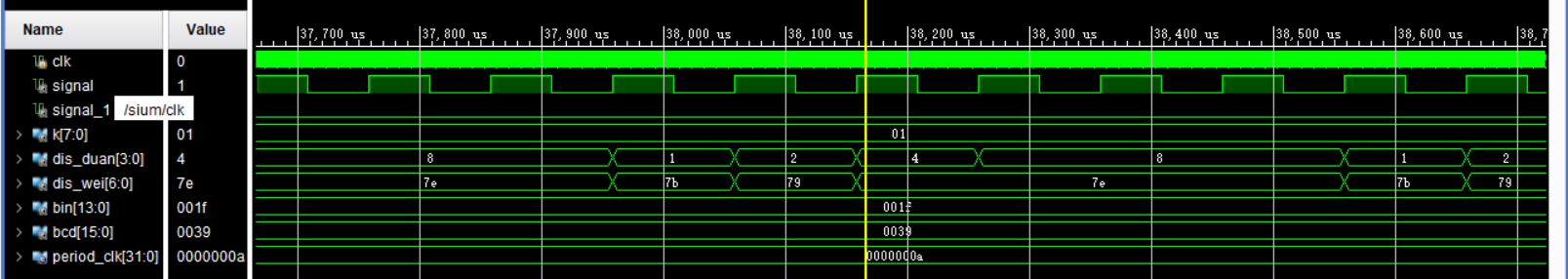


图3.6 理论频率仿真

## 3.6 测频模块

在一秒内，波形重复出现的次数就是频率。由于直接捕捉波形的重复次数有些困难，所以改用地址的重复次数可以让结果更为精确，同时也便于测量。只需要捕捉地址的最高位由1到0的变化就可以知道地址重复出现的次数，进而可以确定出波形重复出现的次数。由于该模块的输入时钟信号是0.5*Hz*，所以高电平1秒，低电平也是1秒，这里选择在高电平上地址重复出现的次数来测量实际频率，当时钟信号下降沿来的时候，将测量统计到的频率返回输出。

### 3.6.1 设计原理

定义一个足够大的中间计数变量*count*，当地址*address*的最高位的下降沿来临的时候，如果这时候0.5*Hz*的时钟信号是高电平，那么*count*加一，等到0.5*Hz*时钟信号是低电平的时候，将计数的结果返回给二进制转换成十进制*BCD*码模块，然后将*BCD*码传递给显示模块在数码管上显示测量的实际频率。

### 3.6.2 模块代码

//实际测频模块测量并在左端4个数码管显示测量的频率值

**module** re\_fre**(**

**input** signal**,** //提供10kHz时钟信号

**input** signal\_1**,** //提供0.5Hz时钟信号

**input** **[**7**:**0**]** address**,** //提供ROM地址

**output** **wire** **[**15**:**0**]** bcd**,** //输出测量的十进制BCD码

**output** **wire** **[**7**:**0**]** dis\_duan\_1**,** //左端数码管位选信号

**output** **wire** **[**6**:**0**]** dis\_wei\_1 //左端数码管段选信号

**);**

**reg** **[**13**:**0**]** count **=** 0**;** //中间变量计数

**reg** **[**13**:**0**]** real\_fre**;** //最后的实际测频结果

//当ROM地址最高位由1变为0时，表示完成一个周期

**always** **@(negedge** address**[**7**])** **begin**

//当0.5Hz时钟信号是高电平时，开始计数

**if(**signal\_1 **==** 1**)** **begin**

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

**end**

//当0.5Hz是低电平时，表示计数停止，count清零

**else** **begin**

count **<=** 0**;**

**end**

**end**

//当0.5Hz时钟信号下降沿来的时候，将技术结果赋给最后的测频结果

**always** **@(negedge** signal\_1**)** **begin**

real\_fre **<=** count**;**

**end**

//调用bin\_to\_bcd模块，将测频结果转换成十进制BCD码

bin\_to\_bcd b1**(**

**.**signal**(**signal**),**

**.**bcd**(**bcd**),**

**.**bin**(**real\_fre**)**

**);**

//调用显示模块，将测频结果显示在数码管上

display\_left d1**(**

**.**signal**(**signal**),**

**.**dis\_duan\_1**(**dis\_duan\_1**),**

**.**dis\_wei\_1**(**dis\_wei\_1**),**

**.**number\_1**(**bcd**)**

**);**

**Endmodule**

### 3.6.3 仿真结果

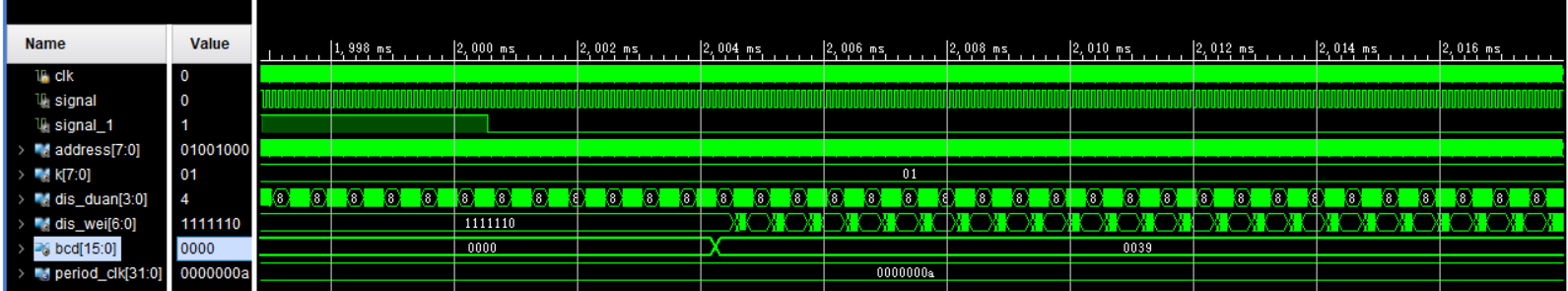


图3.7 测频仿真

## 3.7 波形选择模块

通过*DIP*开关的前四个开关分别控制输出三角波、方波、锯齿波和正弦波。用累加器输出的数据作为*ROM*的地址，进行波形的相位—幅值转换，即可在给定的时间上确定输出的波形的幅值。8位的寻址*ROM*相当于把周期的波形信号离散成具有256个样值的序列，波形*ROM*有8位数据位，则256个样值的幅值以8位二进制数值存在*ROM*中，按照地址的不同可以输出相应相位的波形信号的幅值。同时，*ROM*中还可存放不同类种波形的地址，如正弦波、方波、三角波、锯齿波、调幅波等，这些波均可通过*D/A*转换器将数字信号转化为模拟信号。

### 3.7.1 设计原理

整体思路比较简单，通过控制*DIP*开关前四位的状态来控制波形的选中状态，其中开关向上是高电平，被拨下是低电平。

### 3.7.2 模块代码

//小拨动开关前四位控制输出的波形

**module** switch**(**

**input** **[**7**:**0**]** address**,**

**input** **[**3**:**0**]** sw**,**

**output** **reg** **[**7**:**0**]** spo

**);**

**wire** **[**7**:**0**]** sin**,**sawtooth**,**square**,**triangle**;**

dist\_mem\_gen\_0 d0**(**

**.**a**(**address**),**

**.**spo**(**sin**)**

**);**

dist\_mem\_gen\_2 d2**(**

**.**a**(**address**),**

**.**spo**(**triangle**)**

**);**

dist\_mem\_gen\_1 d1**(**

**.**a**(**address**),**

**.**spo**(**square**)**

**);**

dist\_mem\_gen\_3 d3**(**

**.**a**(**address**),**

**.**spo**(**sawtooth**)**

**);**

**always** **@(\*)** **begin**

**case** **(**sw**)**

4'b0001**:** **begin**

spo **<=** sin**;**

**end**

4'b0010**:** **begin**

spo **<=** sawtooth**;**

**end**

4'b0100**:** **begin**

spo **<=** square**;**

**end**

4'b1000**:** **begin**

spo **<=** triangle**;**

**end**

**default:** spo **<=** 0**;**

**endcase**

**end**

**endmodule**

### 3.7.3 仿真结果

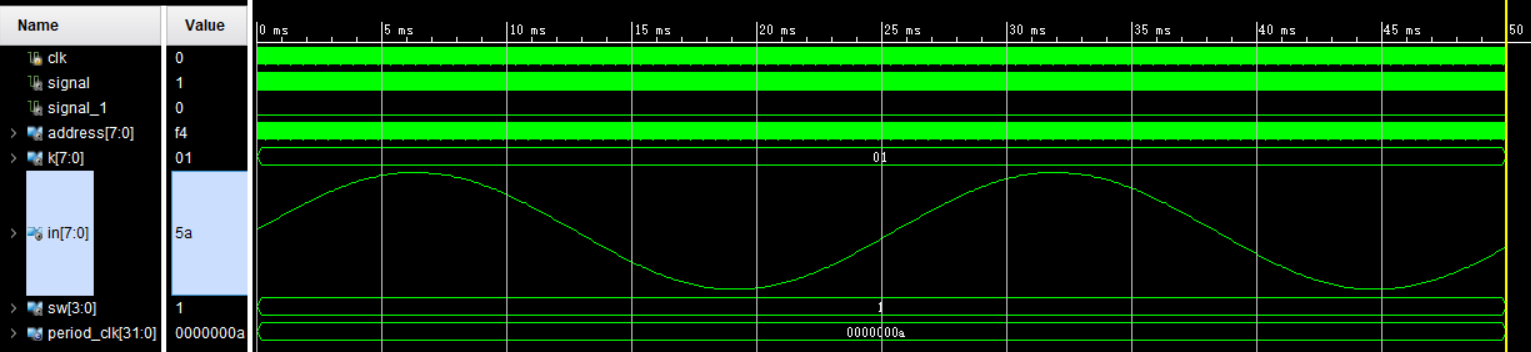


图3.8 波形选择正弦波仿真

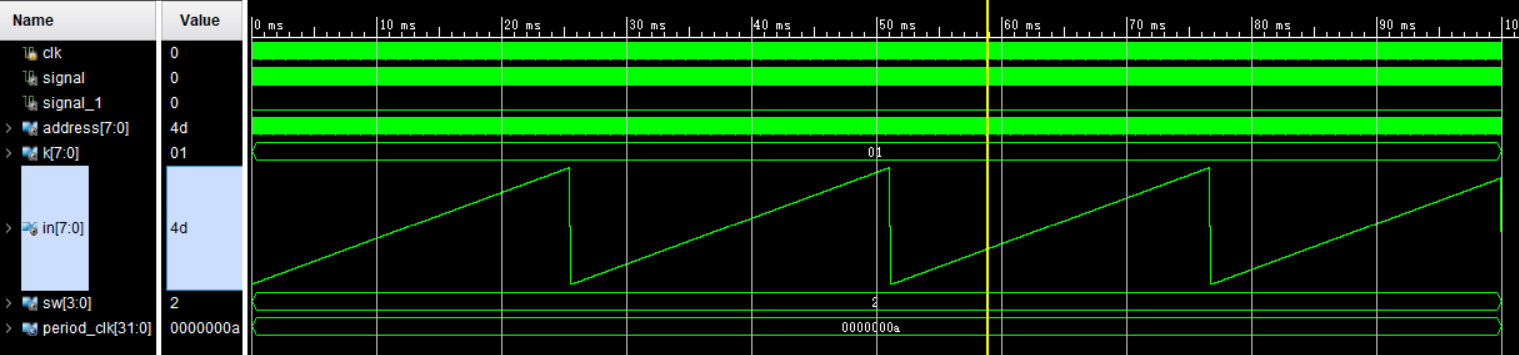


图3.9 波形选择锯齿波仿真

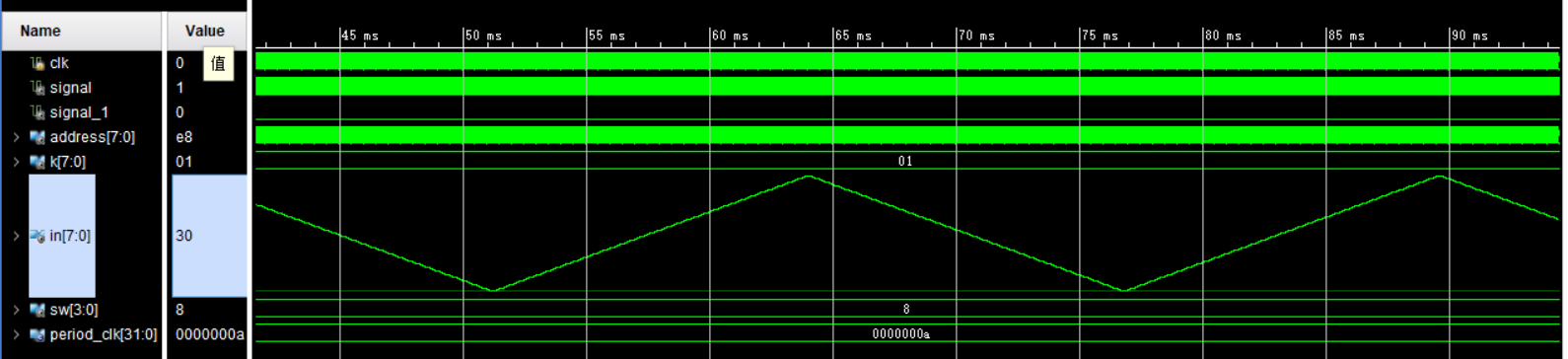


图3.10 波形选择三角波仿真

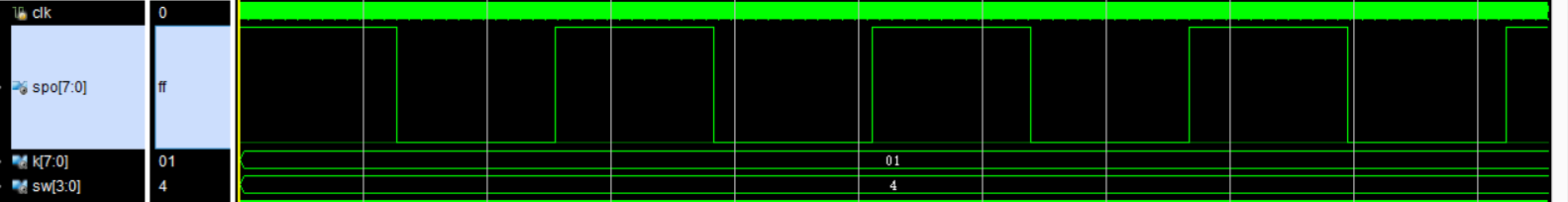


图3.11 波形选择方波仿真

## 3.8 D/A转换器模块

*D/A*转换器的作用是把已经合成的周期波的数字量转换成模拟量。使用*D/A*转换器，须对*D/A*转换器引脚进行配置。波形幅度量化序列*S*（*n*）经 *D/A* 转换后变成了包络为周期波的阶梯波*S*（*t*），*D/A*转换器的分辨率越高，合成的周期波*S*（*t*）台阶数就越多，输出波形的精度也就越高。

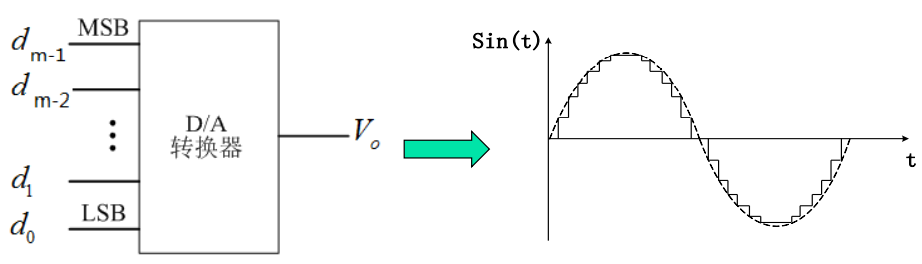


图3.12 *D/A*转换器工作原理

### 3.8.1 设计原理

初始化DA转换器的ILE、CS、WR1、WR2、XEFER端口位为10000，将IP核提供的输出信号提供给DA转换器。

### 3.8.2 模块代码

//数模转换模块，输出波形

**module** DA**(**

**input** **[**7**:**0**]** in**,**

**output** **reg** **[**4**:**0**]** init**,** //输入初始化的端口

**output** **reg** **[**7**:**0**]** spo //输出波形

**);**

**always** **@(\*)** **begin**

spo **<=** in**;**

**end**

**reg** **[**4**:**0**]** init **=** 5'b10000**;**

**endmodule**

### 3.8.3 示波器观测

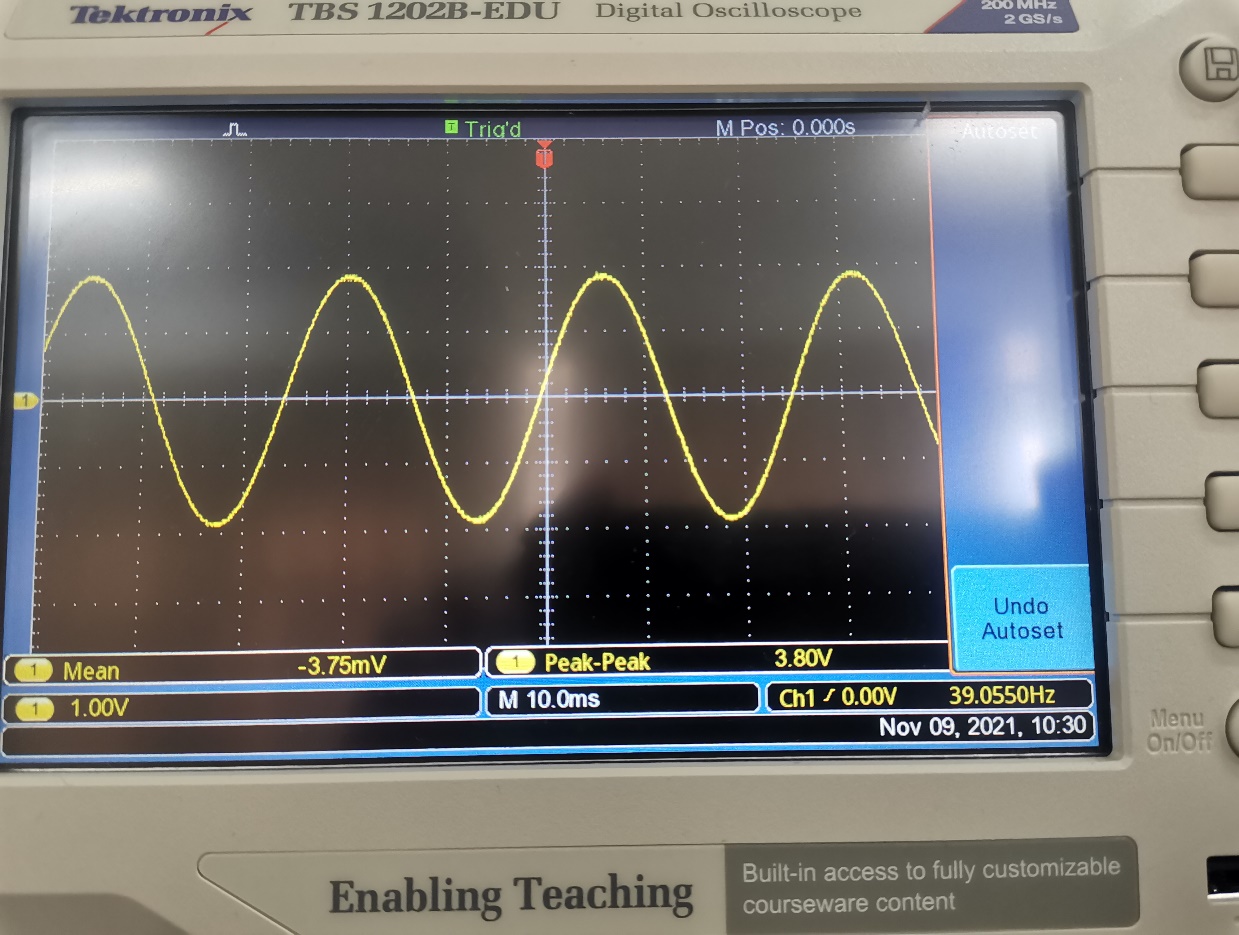


图3.13 正弦波波形

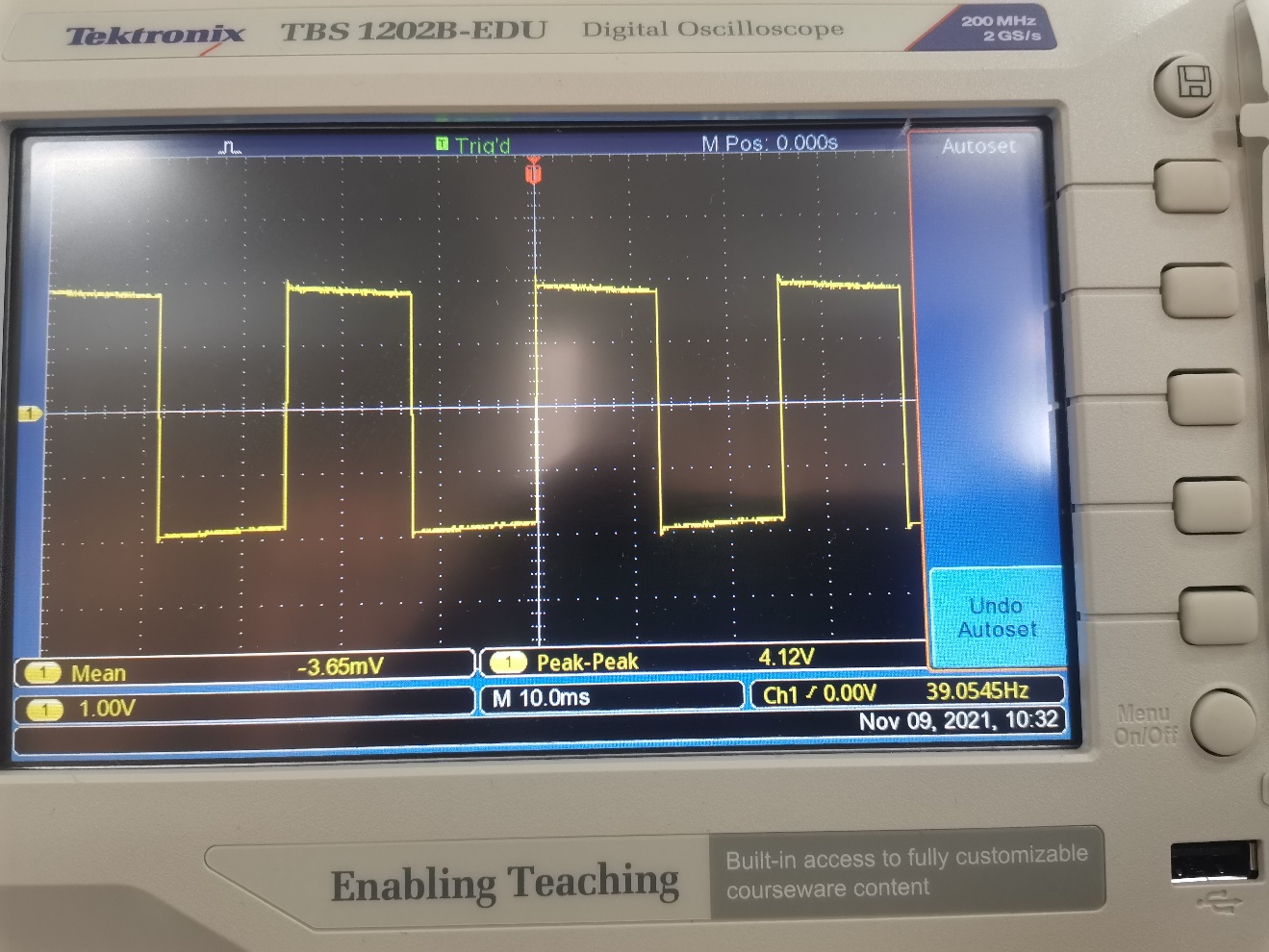


图3.14 方波波形

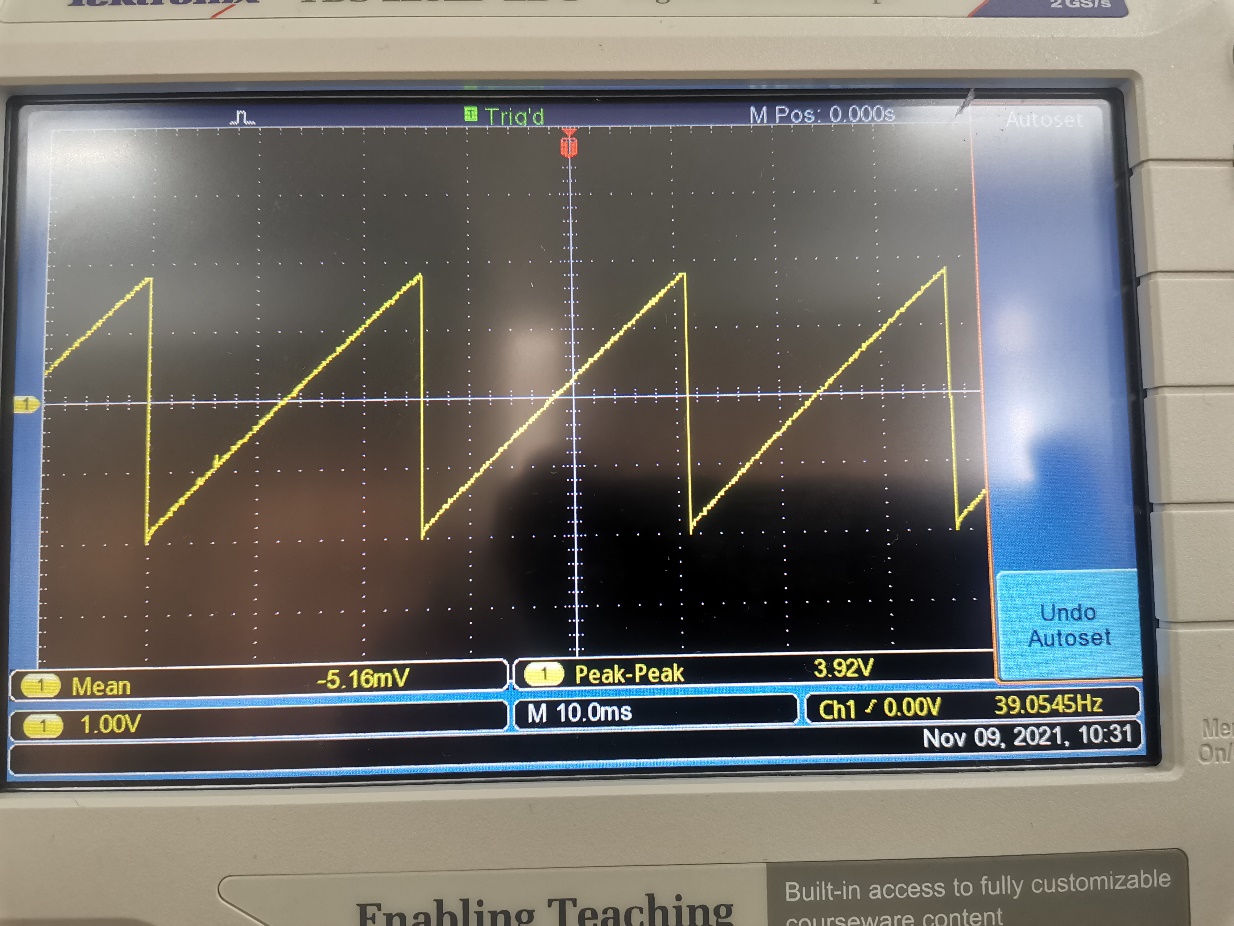


图3.15 锯齿波波形

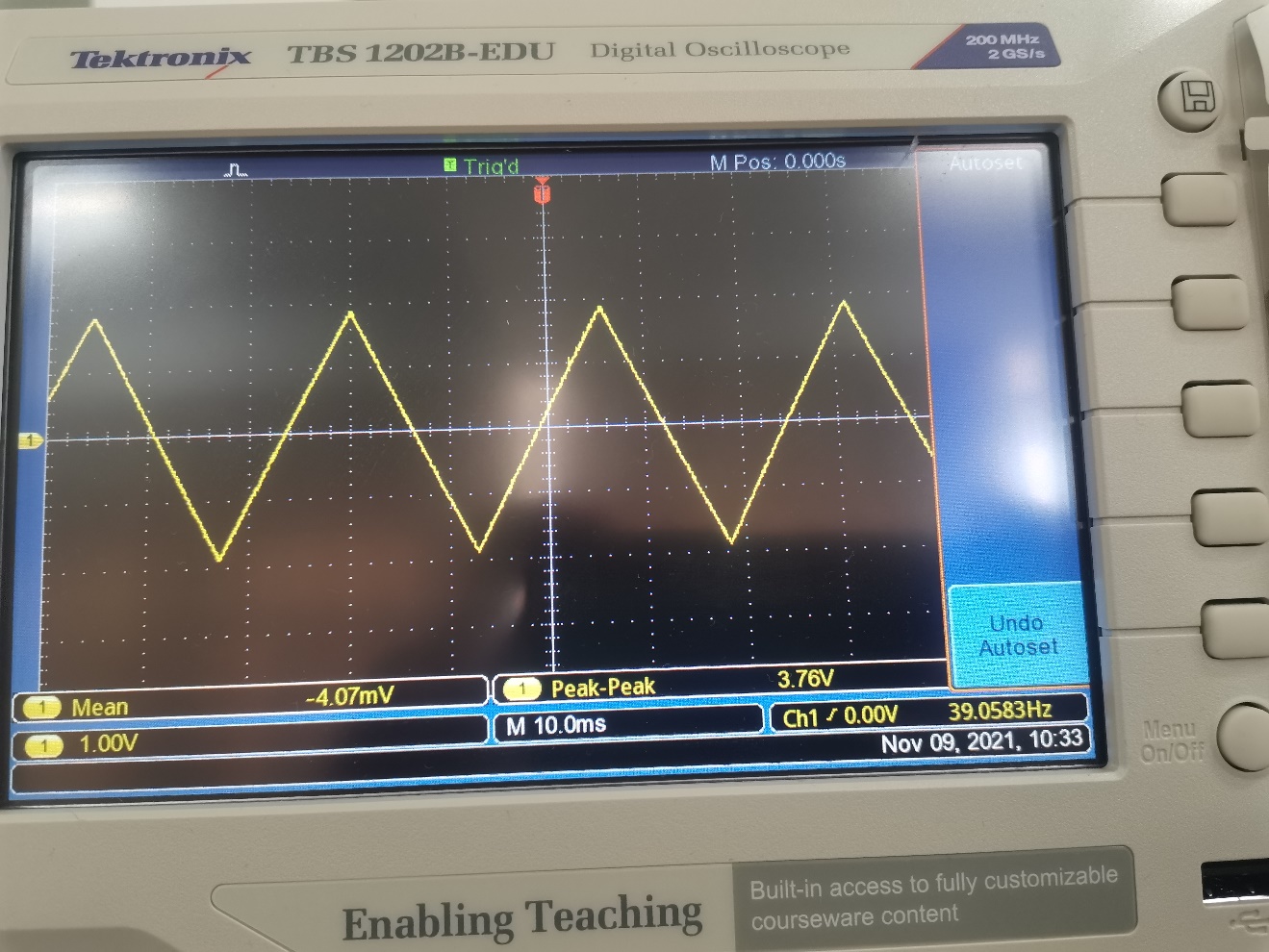


图3.16 三角波波形

## 3.9 顶层模块

顶层类似于*C*语言中的主函数，通过调用其他的模块实现功能的串联实现。

### 3.9.1 设计原理

在选择输入核输出的时候要思考开发板上是否有对应的器件实现功能。调用上述已经阐述过的模块，实现*DDS*最终的结果。

### 3.9.2 模块代码

//顶层模块

**module** top**(**

**input** clk**,** //100MHz时钟信号输入端

**input** **[**7**:**0**]** k**,** //8位拨动开关输入端

**input** **[**3**:**0**]** sw**,** //8位小拨动开关

**output** **[**4**:**0**]** init**,** //初始化DA芯片

**output** **[**7**:**0**]** spo**,** //输出的波形

**output** **wire** **[**3**:**0**]** dis\_duan**,** //右端4位段选信号

**output** **wire** **[**6**:**0**]** dis\_wei**,** //右端4位位选信号

**output** **wire** **[**3**:**0**]** dis\_duan\_1**,** //左端4位段选信号

**output** **wire** **[**6**:**0**]** dis\_wei\_1 //左端4位位选信号

**);**

**wire** signal**;** //10kHz时钟信号

**wire** signal\_1**;** //0.5Hz时钟信号

**wire** **[**7**:**0**]** in**;** //中间变量承载波形

**wire** **[**7**:**0**]** address**;** //ROM地址

**wire** **[**15**:**0**]** bcd**;** //理论值的十进制BCD码

**wire** **[**15**:**0**]** bcd\_1**;** //实际值的十进制BCD码

//顶层调用产生10kHz时钟信号模块

clock c0**(**

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

**.**signal**(**signal**)**

**);**

//顶层调用产生0.5Hz时钟信号模块

clock\_1 c1**(**

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

**.**signal\_1**(**signal\_1**)**

**);**

//顶层调用累加模块

add a0**(**

**.**signal**(**signal**),**

**.**k**(**k**),**

**.**address**(**address**)**

**);**

//顶层调用开关选择波形

switch s0**(**

**.**address**(**address**),**

**.**spo**(**in**),**

**.**sw**(**sw**)**

**);**

//顶层调用数模转换模块

DA D0**(**

**.**in**(**in**),**

**.**init**(**init**),**

**.**spo**(**spo**)**

**);**

//顶层调用理论频率模块

th\_fre t0**(**

**.**signal**(**signal**),**

**.**k**(**k**),**

**.**bcd**(**bcd**),**

**.**dis\_duan**(**dis\_duan**),**

**.**dis\_wei**(**dis\_wei**)**

**);**

//顶层调用实际频率模块

re\_fre r0**(**

**.**signal**(**signal**),**

**.**signal\_1**(**signal\_1**),**

**.**address**(**address**),**

**.**bcd**(**bcd\_1**),**

**.**dis\_duan\_1**(**dis\_duan\_1**),**

**.**dis\_wei\_1**(**dis\_wei\_1**)**

**);**

**endmodule**

### 3.9.3 仿真程序

**module** sium**;**

**reg** clk**;**

**wire** **[**7**:**0**]** spo**;**

**wire** **[**7**:**0**]** k **=** 8'b0000\_0001**;**

**wire** **[**3**:**0**]** sw **=** 4'b1000**;**

**wire** **[**3**:**0**]** dis\_duan**;**

**wire** **[**6**:**0**]** dis\_wei**;**

**wire** **[**3**:**0**]** dis\_duan\_1**;**

**wire** **[**6**:**0**]** dis\_wei\_1**;**

top t0**(**

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

**.**k**(**k**),**

**.**sw**(**sw**),**

**.**init**(**init**),**

**.**spo**(**spo**),**

**.**dis\_duan**(**dis\_duan**),**

**.**dis\_wei**(**dis\_wei**),**

**.**dis\_duan\_1**(**dis\_duan\_1**),**

**.**dis\_wei\_1**(**dis\_wei\_1**)**

**);**

**parameter** period\_clk **=** 10**;**

**always** **begin**

clk **=** 1'b0**;**

**#(**period\_clk**/**2**)** clk **=** 1'b1**;**

**#(**period\_clk**/**2**);**

**end**

**endmodule**

### 3.9.4 仿真结果

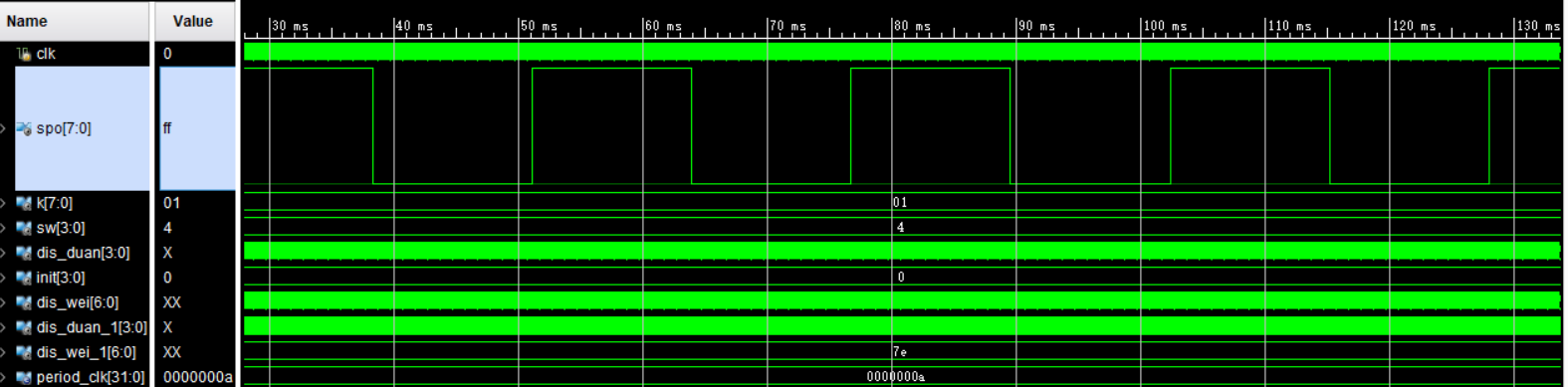


图3.17 顶层仿真

# 4 引脚分配

set\_property PACKAGE\_PIN P17 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[10]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[9]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[8]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_seg[0]}]

set\_property PACKAGE\_PIN G1 [get\_ports {dis\_seg[10]}]

set\_property PACKAGE\_PIN F1 [get\_ports {dis\_seg[9]}]

set\_property PACKAGE\_PIN E1 [get\_ports {dis\_seg[8]}]

set\_property PACKAGE\_PIN G6 [get\_ports {dis\_seg[7]}]

set\_property PACKAGE\_PIN D4 [get\_ports {dis\_seg[6]}]

set\_property PACKAGE\_PIN E3 [get\_ports {dis\_seg[5]}]

set\_property PACKAGE\_PIN D3 [get\_ports {dis\_seg[4]}]

set\_property PACKAGE\_PIN F4 [get\_ports {dis\_seg[3]}]

set\_property PACKAGE\_PIN F3 [get\_ports {dis\_seg[2]}]

set\_property PACKAGE\_PIN E2 [get\_ports {dis\_seg[1]}]

set\_property PACKAGE\_PIN D2 [get\_ports {dis\_seg[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {k[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {k[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {k[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {k[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {k[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {k[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {k[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {k[0]}]

set\_property PACKAGE\_PIN P5 [get\_ports {k[7]}]

set\_property PACKAGE\_PIN P4 [get\_ports {k[6]}]

set\_property PACKAGE\_PIN P3 [get\_ports {k[5]}]

set\_property PACKAGE\_PIN P2 [get\_ports {k[4]}]

set\_property PACKAGE\_PIN R2 [get\_ports {k[3]}]

set\_property PACKAGE\_PIN M4 [get\_ports {k[2]}]

set\_property PACKAGE\_PIN N4 [get\_ports {k[1]}]

set\_property PACKAGE\_PIN R1 [get\_ports {k[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {spo[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {spo[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {spo[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {spo[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {spo[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {spo[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {spo[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {spo[0]}]

set\_property PACKAGE\_PIN U9 [get\_ports {spo[7]}]

set\_property PACKAGE\_PIN R7 [get\_ports {spo[3]}]

set\_property PACKAGE\_PIN T6 [get\_ports {spo[2]}]

set\_property PACKAGE\_PIN R8 [get\_ports {spo[1]}]

set\_property PACKAGE\_PIN G1 [get\_ports {dis\_duan[3]}]

set\_property PACKAGE\_PIN F1 [get\_ports {dis\_duan[2]}]

set\_property PACKAGE\_PIN E1 [get\_ports {dis\_duan[1]}]

set\_property PACKAGE\_PIN G6 [get\_ports {dis\_duan[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_duan[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_duan[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_duan[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_duan[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei[0]}]

set\_property PACKAGE\_PIN D4 [get\_ports {dis\_wei[6]}]

set\_property PACKAGE\_PIN E3 [get\_ports {dis\_wei[5]}]

set\_property PACKAGE\_PIN D3 [get\_ports {dis\_wei[4]}]

set\_property PACKAGE\_PIN F4 [get\_ports {dis\_wei[3]}]

set\_property PACKAGE\_PIN F3 [get\_ports {dis\_wei[2]}]

set\_property PACKAGE\_PIN E2 [get\_ports {dis\_wei[1]}]

set\_property PACKAGE\_PIN D2 [get\_ports {dis\_wei[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_duan\_1[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_duan\_1[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_duan\_1[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_duan\_1[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei\_1[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei\_1[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei\_1[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei\_1[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei\_1[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei\_1[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {dis\_wei\_1[0]}]

set\_property PACKAGE\_PIN G2 [get\_ports {dis\_duan\_1[3]}]

set\_property PACKAGE\_PIN C2 [get\_ports {dis\_duan\_1[2]}]

set\_property PACKAGE\_PIN C1 [get\_ports {dis\_duan\_1[1]}]

set\_property PACKAGE\_PIN H1 [get\_ports {dis\_duan\_1[0]}]

set\_property PACKAGE\_PIN B4 [get\_ports {dis\_wei\_1[6]}]

set\_property PACKAGE\_PIN A4 [get\_ports {dis\_wei\_1[5]}]

set\_property PACKAGE\_PIN A3 [get\_ports {dis\_wei\_1[4]}]

set\_property PACKAGE\_PIN B1 [get\_ports {dis\_wei\_1[3]}]

set\_property PACKAGE\_PIN A1 [get\_ports {dis\_wei\_1[2]}]

set\_property PACKAGE\_PIN B3 [get\_ports {dis\_wei\_1[1]}]

set\_property PACKAGE\_PIN B2 [get\_ports {dis\_wei\_1[0]}]

set\_property PACKAGE\_PIN T8 [get\_ports {spo[0]}]

set\_property PACKAGE\_PIN U6 [get\_ports {spo[4]}]

set\_property PACKAGE\_PIN U7 [get\_ports {spo[5]}]

set\_property PACKAGE\_PIN V9 [get\_ports {spo[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {init[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {init[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {init[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {init[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {init[0]}]

set\_property PACKAGE\_PIN R5 [get\_ports {init[4]}]

set\_property PACKAGE\_PIN N6 [get\_ports {init[3]}]

set\_property PACKAGE\_PIN V6 [get\_ports {init[2]}]

set\_property PACKAGE\_PIN R6 [get\_ports {init[1]}]

set\_property PACKAGE\_PIN V7 [get\_ports {init[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[0]}]

set\_property PACKAGE\_PIN U3 [get\_ports {sw[3]}]

set\_property PACKAGE\_PIN U2 [get\_ports {sw[2]}]

set\_property PACKAGE\_PIN V2 [get\_ports {sw[1]}]

set\_property PACKAGE\_PIN V5 [get\_ports {sw[0]}]

# 5 结论

## 5.1 问题与反思

### 5.1.1 数码管显示的时候最高位总是比其它位更亮

实测将程序烧录进开发板的时候发现，数码管的最高位总是比其他几位要亮一点。老师提出可能是段选信号在一个周期里面扫描1000多于其他几位。查看仿真程序如下，发现问题确实如此。

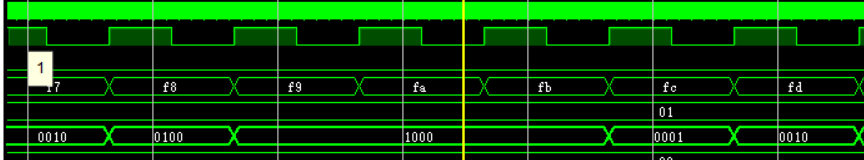


图5.1 问题1

仔细检查代码发现，原模块代码中当*sel*等于5的时候，将*sel*置零，又经过一个上升沿才让*sel*变成1进行下一个周期的循环。

### 5.1.2 二进制数转换成十进制BCD码

将二进制数11111转换成十进制*BCD*码。

1. 二进制数11111是十进制数31，使用*BCD*码表示位0011,0001。所以在二进制数11111前补上八个0，得到0000,0000,11111。
2. 开始左移，同时判断*BCD*码的个位是否大于4，0000,0000,11111左移一位，0000,0001,11110，左移一位，0000,0011,11100，左移一位，0000,0111,11000，*BCD*码个位大于4，加3，0000,1010,11000，左移一位，0001,0101,10000，个位大于4，加3，0001,1000,10000，左移一位，0011,0001,00000。
3. 最后通过移位加3法得到0011,0001,00000，取前面的8位*BCD*码，得到十进制*BCD*码0011,0001。

## 5.2 收获与感受

通过这次数字系统综合设计的实验，我感觉自己收获还是比较大的。从一开始的对*Verilog*语言一无所知到最后能够自己实现一点不算太复杂的功能。可能是受制于*C*++等语言的教学，认为学习一门语言应该要从最基础的开始起，比如定义变量之类的，但是这么课程只有短短两周的时间，所以几乎没有充足的时间来吧*Verilog*语言学习透彻，自己照着教材上的代码敲一敲，照葫芦画瓢，最后也还是能够将实验的基本要求实现出来。这其中就考验到了融会贯通和举一反三的能力，怎么样通过少量的例子来拓展出更为庞大的功能，虽然程序中有时候报错都是因为*Verilog*语言的基础没有掌握好导致的，但是通过查找错误日志，改改变量类型碰碰运气还是能够让程序运行起来。同时*Verilog*语言就像老师说的，不同于其他的高级语言，能够让人天马行空的想象，*Verilog*语言还需要考虑自己设计的程序能否在实际电路中展现出来，这就考验到了数字逻辑电路的知识是否掌握牢固。虽然本人偷懒，在编写代码的过程中尽量避免复杂的逻辑，尽量不使用嵌套这种可能行不通的语句，我认为这样子是不好的，虽然这样做基本不需要考虑在实际电路中，简单逻辑是否能够实现，但是使用简单逻辑也决定了代码的上限很低，不利于更大规模的电路设计，同时可能也极大地降低了编译效率

通过本次实验，我还学习到了有关于*DDS*的很多基础知识，能够亲手制作出一个*DDS*也是一个让人振奋人心的事。除了了解到*DDS*的工作原理之外，还拓展了解到了很多知识，比如正弦波查找表这一块就是比较陌生的，以前只知道计算机可以画出正弦函数图像，通过本次实验，我还了解到，计算机画出正弦波正是基于查找表，否则通过泰勒级数展开来求正弦函数的值消耗的资源可能过多。包括这次的一个模块，二进制转换成十进制*BCD*码，基于已学的知识，我能想到的只有先转换成十进制数后取模这种操作，但是这种操作相比于移位加3法消耗的资源过多，又是一次谈到了资源利用问题。这不禁让我联想到了正在学习的数字信号处理，直接计算*DFT*和快速*DFT*算法的区别，做同一件事情的途径有很多条，但是我们学习就是要成为有智慧的人去寻找一条最快速，消耗资源最少的途径。

# 6 致谢

感谢姜萍老师这两周来的辛勤付出，感谢您在实验过程中的指导与解惑，您的建议让我在本次实验过程中少走了很多弯路，正是您的指导让我看到了自己的不足之处，也正是您的授课让我学到了很多以前不曾接触过的知识。同时在这里也感谢在实验过程中给予我莫大帮助的同学们，感谢你们能够在本次实验中与我一同解决棘手的问题，也感谢你们能够耐心的给予我一些改进程序的意见。最后也要感谢学校和父母，能够让我有机会接触数字系统综合实验，谢谢你们！

# 7 参考文献

【1】蒋立平. 数字逻辑电路与系统设计（第 3 版）. 北京：电子工业出版社，2019

**【2】** 王睿庭 基于 ARM 和 FPGA 的 DDS 扫频信号源设计 [J] 电子电路设计与方案 2020.9

【3】 蒋小军 基于 FPGA 的 DDS 信号发生器设计 [J] 设计与分析 2020.7

【4】 花汉兵，吴少琴 EDA 技术与设计 [M] 电子工业出版社