实验一 ——

多路选择器与 CPU 辅助模块设计实验报告

姓名：方晓霖 学号：3150105184 专业：混合班

课程名称：计算机组成实验 同组学生姓名：无

实验时间：2017-03-07 实验地点：紫金港东 4-509 指导老师：张明敏、洪奇军

一、实验目的和要求

1.1 实验目的

1. 熟练掌握 EDA 开发工具和开发流程
2. 复习数字逻辑设计实现方法
3. 扩展优化逻辑实验基本模块
4. 优化计算机系统实现的辅助模块
5. 了解计算机硬件系统将中到的最基本模块

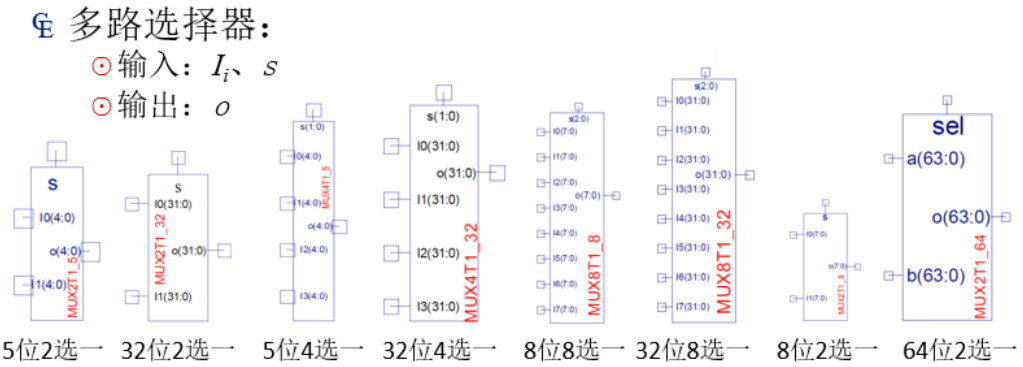
1.2 实验要求

1. 整理设计逻辑实验输出模块
   多路选择器、基本算术逻辑运算模块、数据扩展模块
2. 整理逻辑实验输出的辅助模块
   消除机械抖动模块、通用分频模块
3. 设计存储器 IP 模块
   32 位 ROM、32 位 RAM
4. 设计 CPU 调试测试显示通道模块
   在逻辑实验 Exp13 基础上重建

二、实验内容和原理

2.1 逻辑实验输出模块优化——基础逻辑器件

多路选择器 MUX



当 s 为不同的值时，选择不同的输入 I_i 作为输出。具体将在下一部分详细说明。

2.2 逻辑实验输出模块优化——辅助逻辑部件

1.八数据通路模块：Multi_8CH32

功能：多路信号显示选择控制

用于 CPU 等各类信号的调试和测试，由 1 个或多个 8 选 1 选择器构成。

EN：使能信号(仅控制通道 0)

SW[7:5]：通道选择控制

Point_in(63:0)：小数点输入（每个通道 8 位，共 64 位）

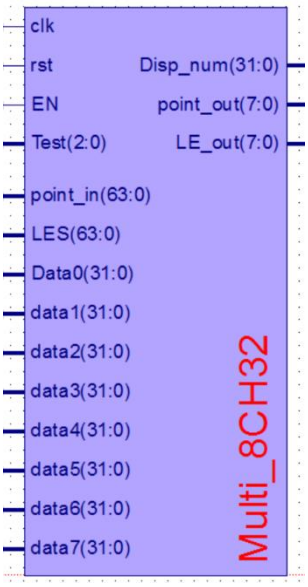
LES(63:0)：使能 LE（闪烁)控制输入（每个通道 8 位，共 64 位）

Data0-Data7[31:0]：数据输入通道(Data0 特殊)

LES_out(7:0) ：当前使能位输出

Point_out(7:0)：当前小数点输出

逻辑符号——



2. 逻辑实验通用分频模块 M1：clk_div.v

功能：32 位计数分频输出：clkdiv

CPU 时钟输出： Clk_CPU

3. 只读存储器 IP 核 M14-1：ROM_32_32
4. 随机存储器 IP 核 M14-2：RAM_32_32
5. 其他模块——
   开关去抖动模块 M2 优化：SAnti_jitter.v
   双 32 位数据输入 IP 核 M4：SEnter_2_32
   七段码显示器 IP 核 M3：SSeg7_Dev
   LED 并行显示模块 M6：SPIO
   ……

   在本实验中，使用提供的文件

三、主要仪器设备

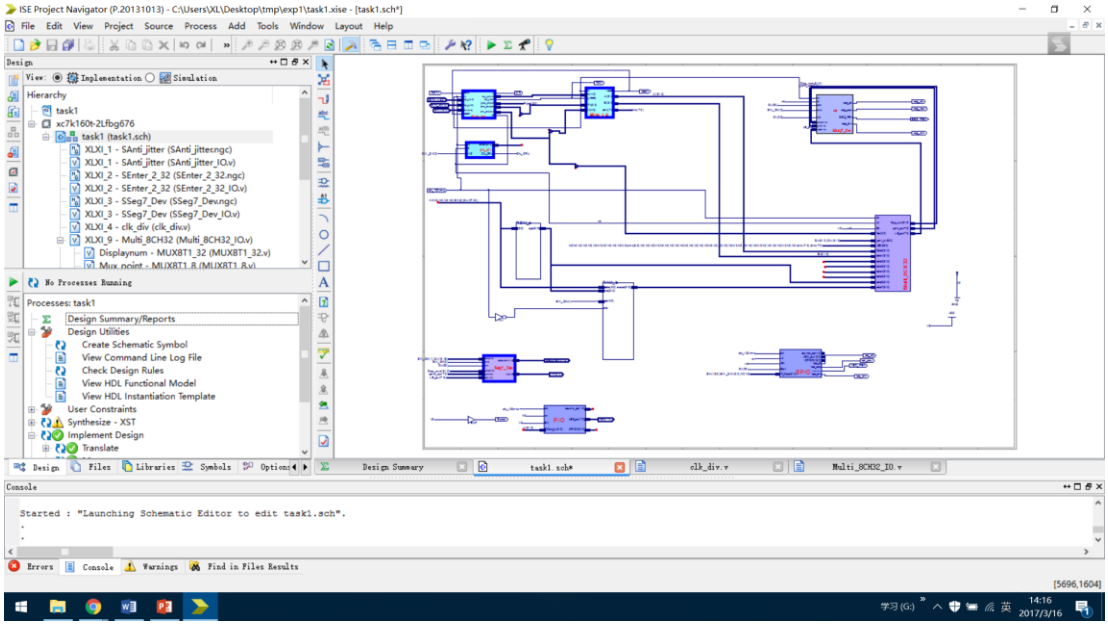
实验设备

装有 Xilinx ISE14.7 的电脑	1 台
Sword 开发板	1 个

四、操作方法与实验步骤

1. 顶层模块

根据原理图模板实现
放置模块，连接若干信号，指定导线输入输出

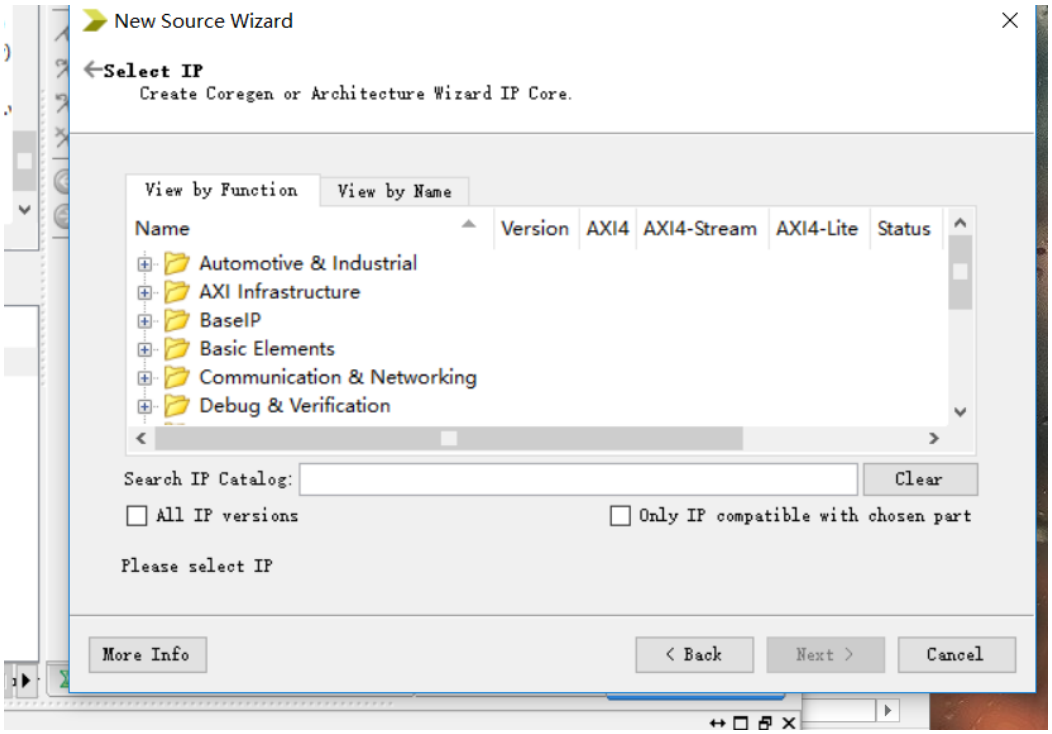


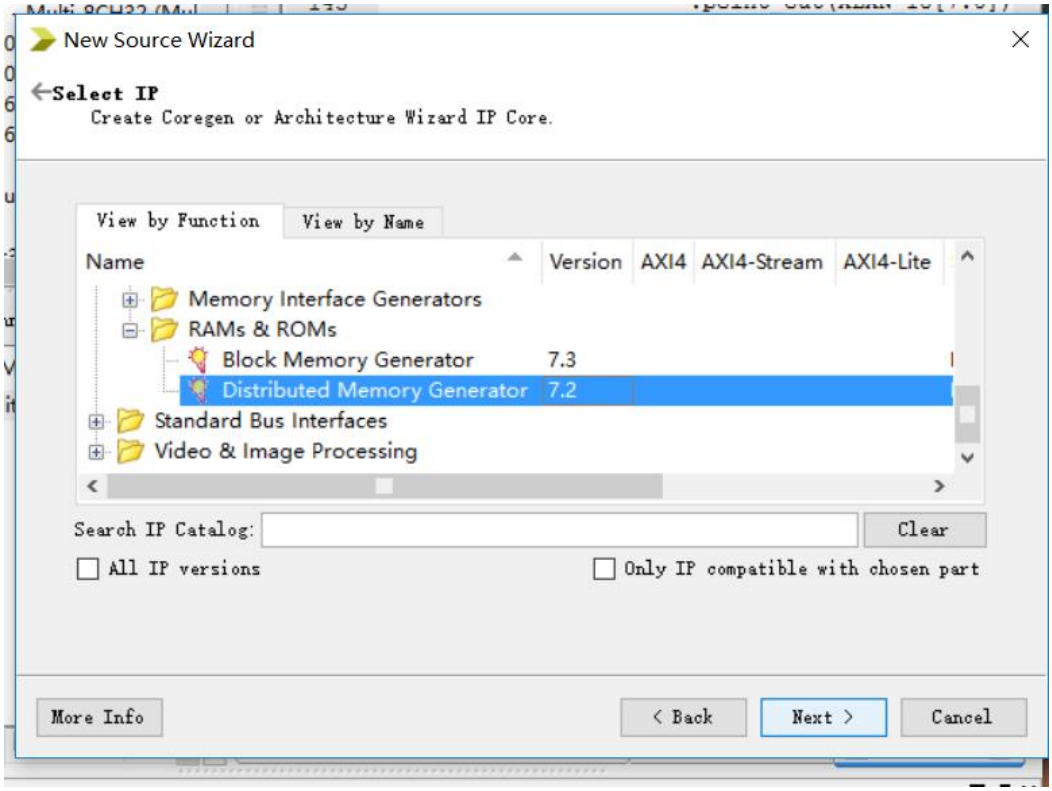
原理图检查无误

2. 存储器 IP 核生成

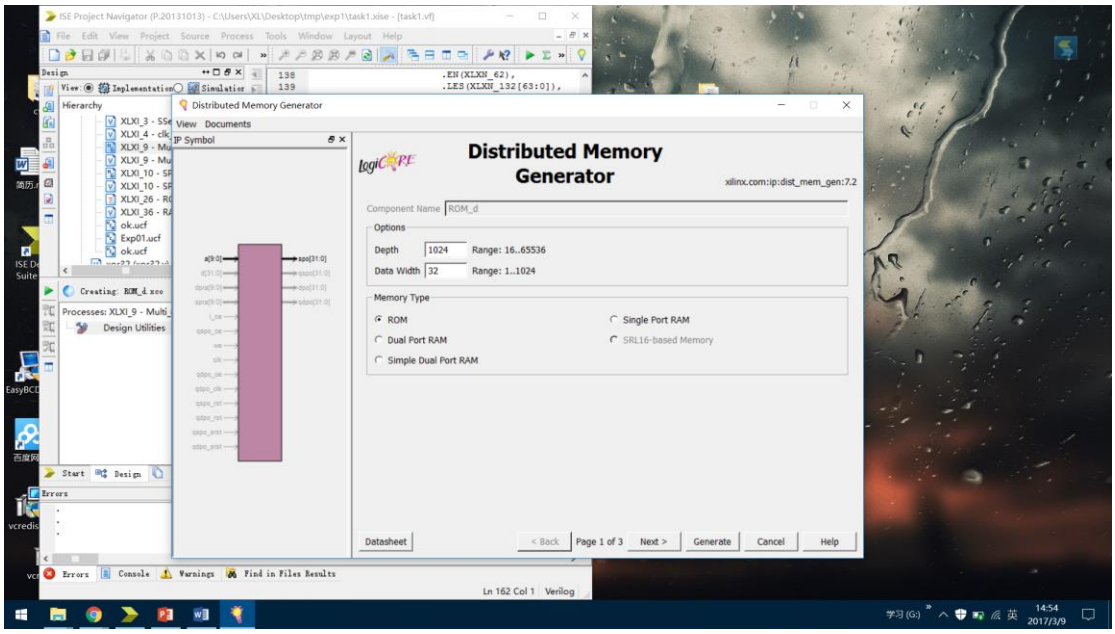
ROM 的 IP 核的生成——

在 ISE 集成菜单上 New source,, 选择新建 IP,点击 Next,选择 Distributed Memory

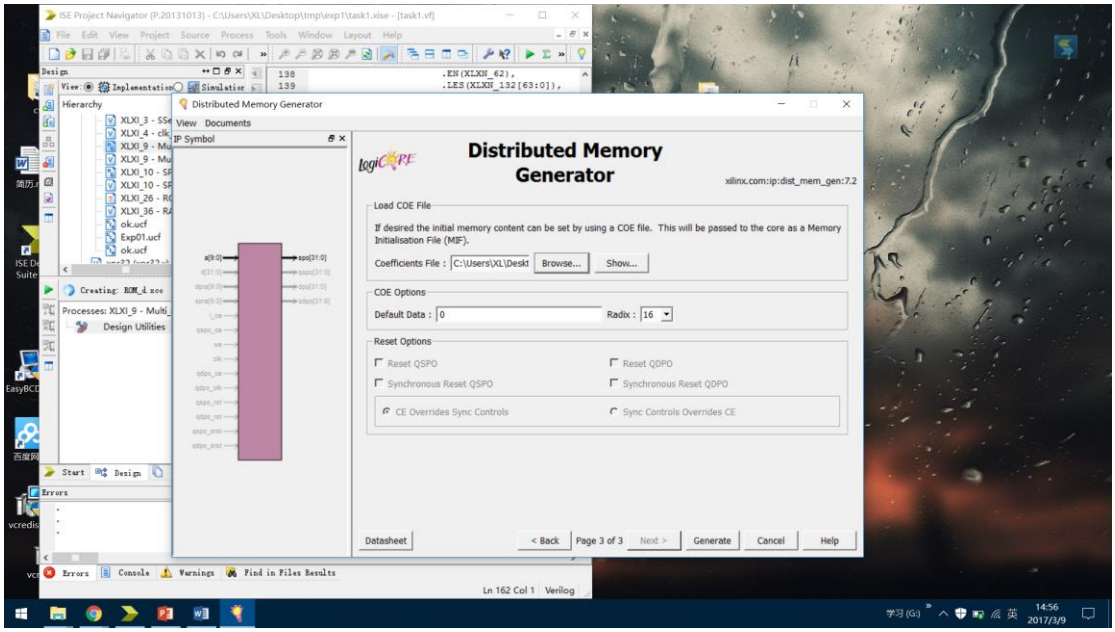




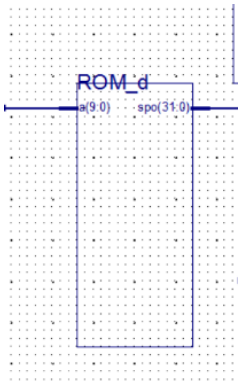
填入 1024 和 32，选择 ROM



点击 Next，点击 Browse…，选择初始化关联文件。ROM.coe 已给出，完成后点击 generate 生成。

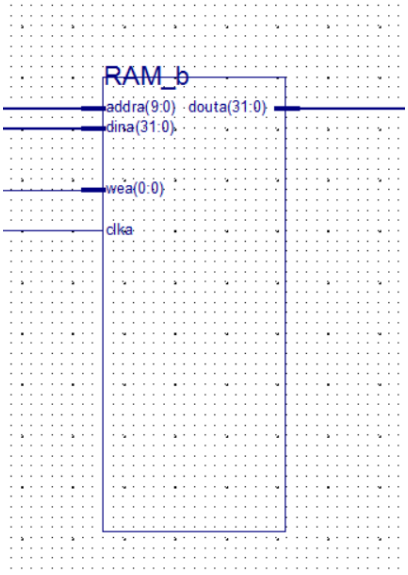


选择 symbol 即可调用新生成的 ROM



RAM 的生成——

过程与 ROM 类似，在第一步选择 Block Memory Generator。生成后调用。生成的逻辑符号——

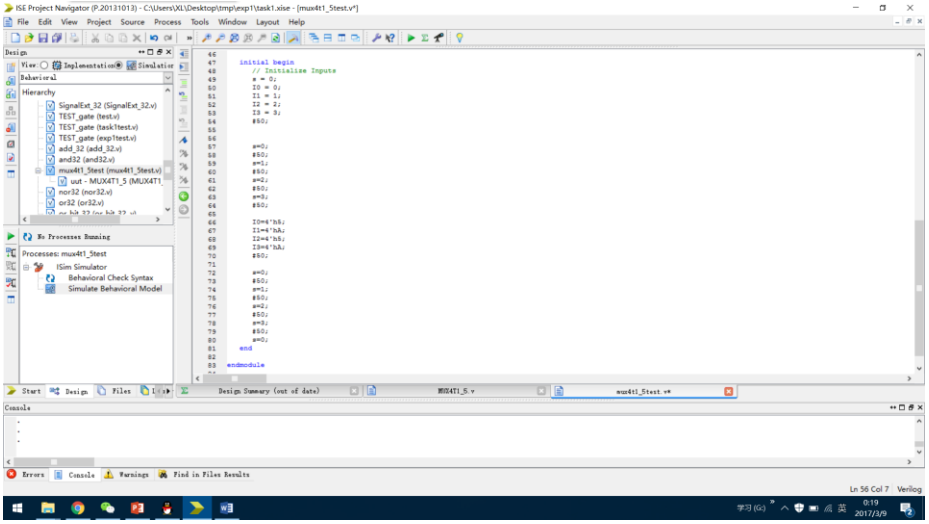


3. 附加要求——基本逻辑模块代码和模拟图

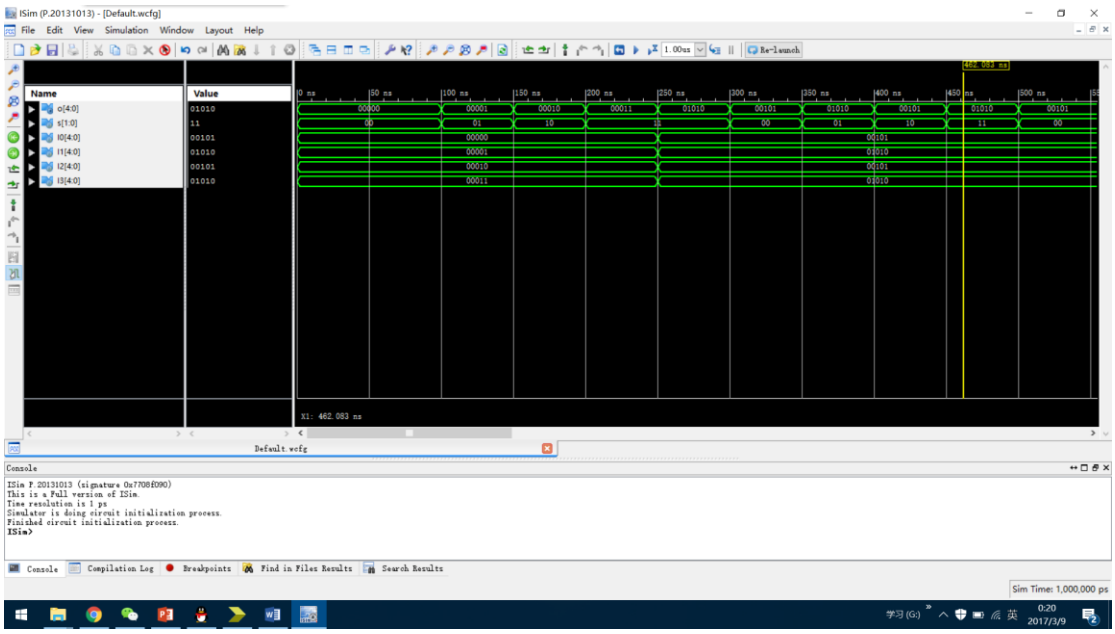
1）5 位 4 选 1——MUX4T1_5

```
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module   MUX4T1_5(input [1:0]s,
22                    input [4:0]I0,
23                    input [4:0]I1,
24                    input [4:0]I2,
25                    input [4:0]I3,
26                    output reg[4:0]o
27                     );
28
29        always@*          //5位4选一,I0、I1、I2、I3对应选择通道0、1、2、3
30           case(s)
31              0:
32                 o=I0;
33              1:
34                 o=I1;
35              2:
36                 o=I2;
37              3:
38                 o=I3;
39           endcase
40
41  endmodule
42
```

模拟所用数据



得到模拟图。



100ns 前，s 为 00，输出选择 I0，在图上为 00 无误。
100ns 时，s 变为 01，输出选择 I1，在仿真图上为 01 无误。
150ns 时，s 变为 10，输出选择 I2，在仿真图上为 10 无误。
200ns 时，s 变为 11，输出选择 I3，在仿真图上为 11 无误。
250ns 时，I0 变为 0101，I1 变为 1010，I2 变为 0101，I3 变为 1010。
300ns 时，s 变为 00，输出选择新的 I0，在仿真图上为 0101 无误。
350ns 时，s 变为 01，输出选择新的 I1，在仿真图上为 1010 无误。
400ns 时，s 变为 10，输出选择新的 I2，在仿真图上为 0101 无误。
450ns 时，s 变为 11，输出选择新的 I3，在仿真图上为 1010 无误。

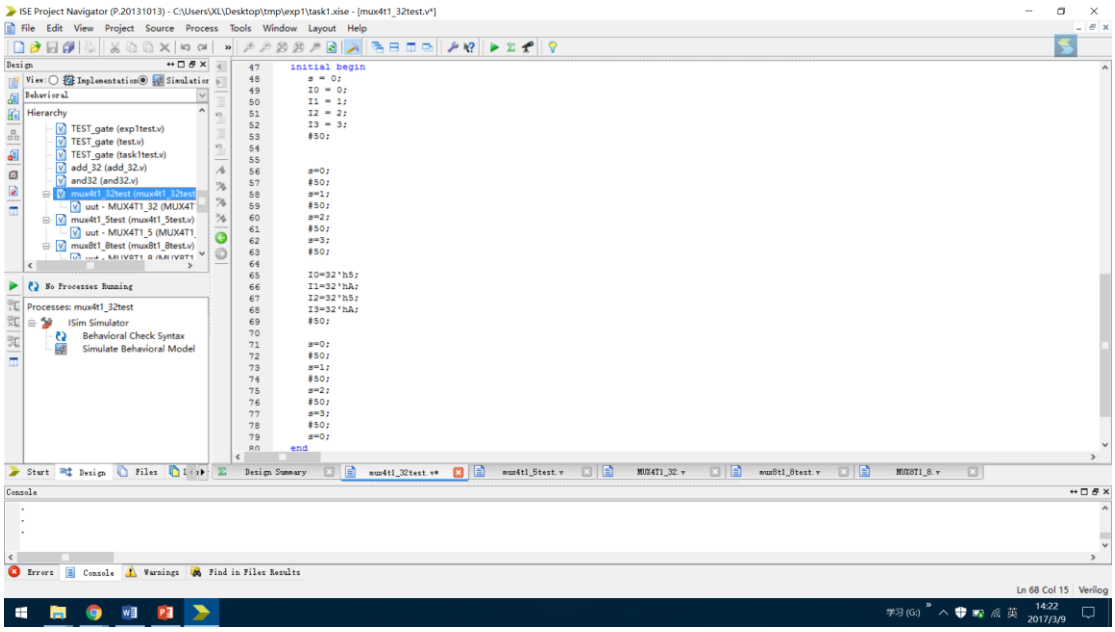
2）32 位 4 选 1——MUX4T1_32

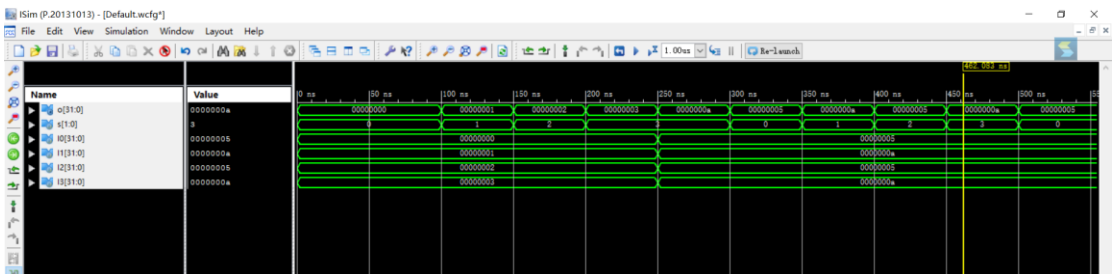
代码：

```
20  //////////////////////////////////////////////////////////////////////////////////
21  module   MUX4T1_32(input [1:0]s,
22                     input [31:0]I0,
23                     input [31:0]I1,
24                     input [31:0]I2,
25                     input [31:0]I3,
26                     output reg[31:0]o
27                     );
28
29        always@*          //32位4选一,I0、I1、I2、I3对应选择通道0、1、2、3
30           case(s)
31              0: o=I0;
32              1: o=I1;
33              2: o=I2;
34              3: o=I3;
35           endcase
36
37  endmodule
38
```

Design Summary | MUX2T1_64.v | MUX2T1_32.v | MUX4T1_32.v | mux4

仿真所用数据：

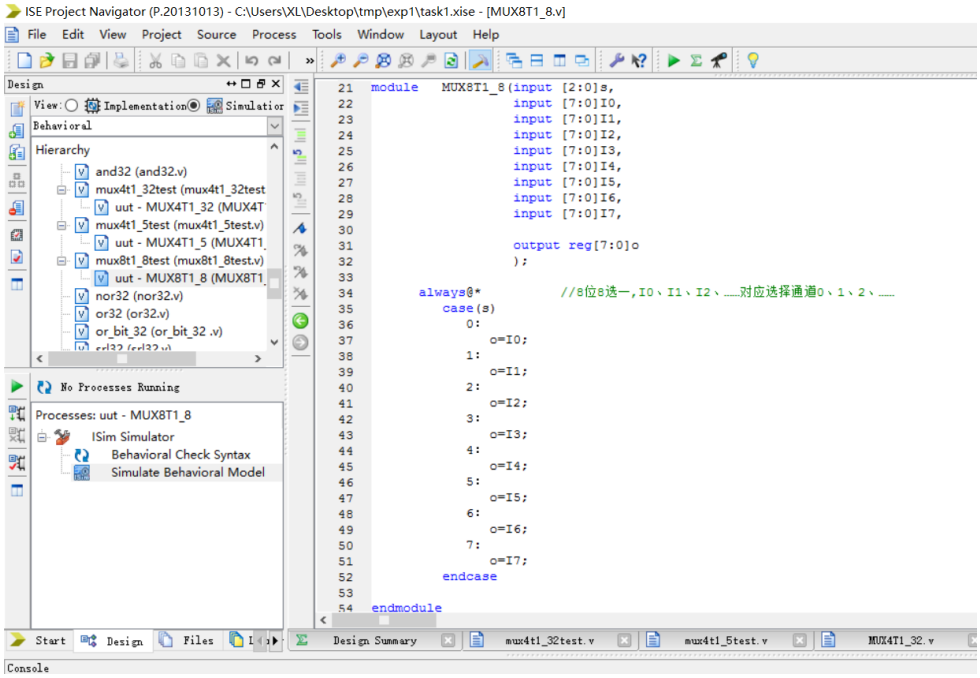




此处，变化与 MUX4T1_5 完全相同，仅输入的 5 位数据改为 32 位数据

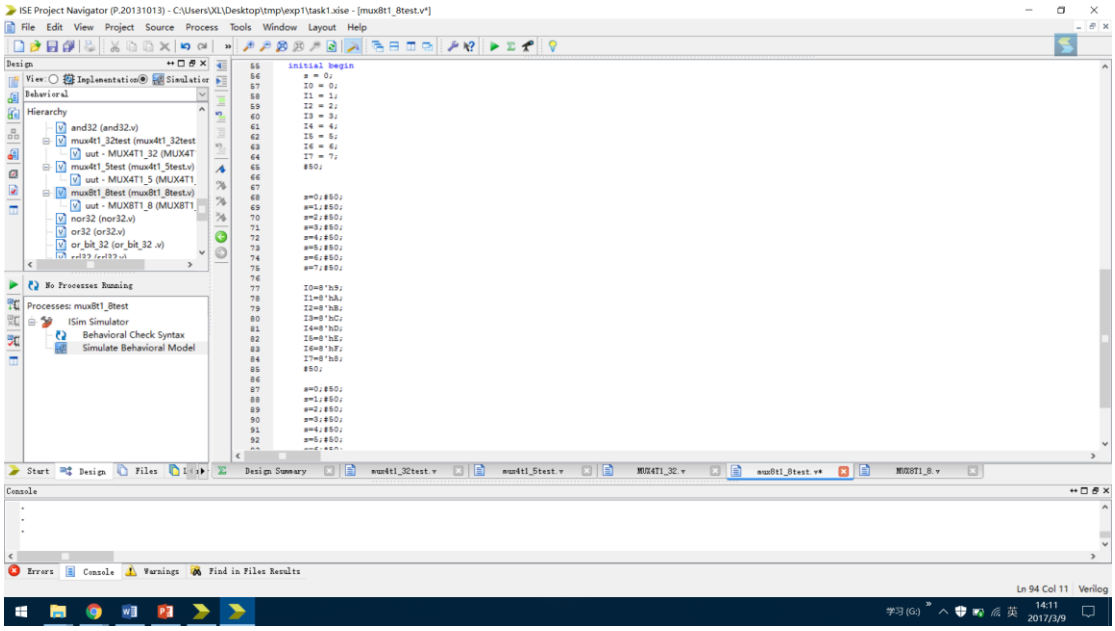
3）8 位 8 选 1——MUX8T1_8

代码：

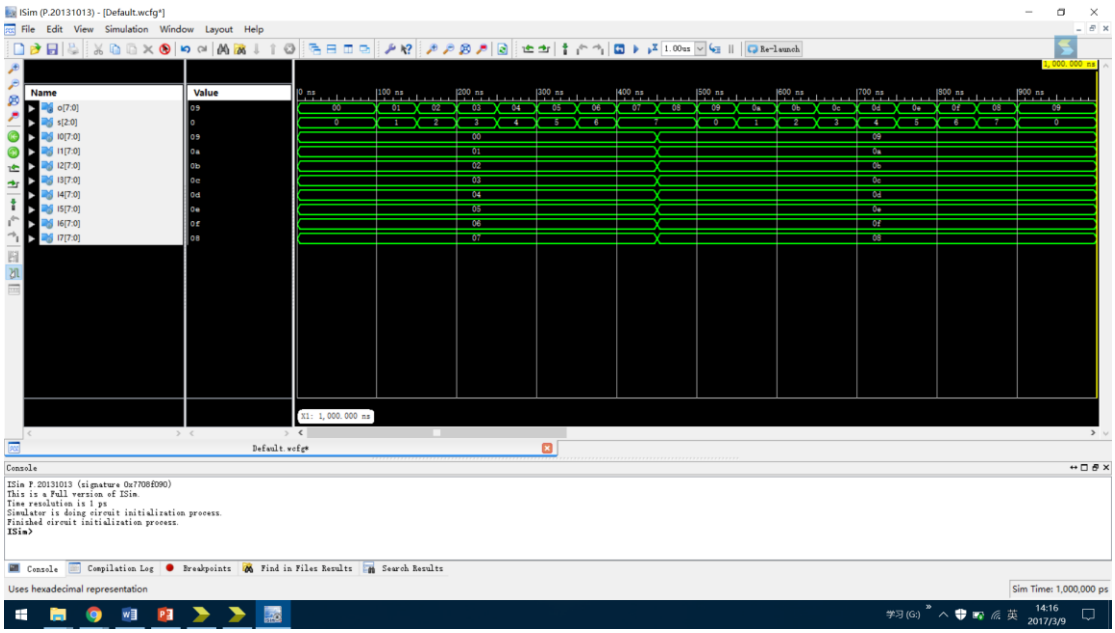


此处，8 选 1，即 s 的输入从原来的 0 到 3 改为 0 到 7。对应选择输出 I0 到 I7。

仿真所用代码



得到仿真图



可以看到每 50ns，s 不断变化，从 0 到 7 时，输出数据 o 从 0 变化到 7。

450ns 时，I0 到 I7 的数据发生变化，o 对应发生改变，选择了新的 I7，输出 8.

500ns 后 s 每次改变时，o 都选择相应新的的 Ii ，输出无误。

其他多路选择器原理均相同，仅输入位数，选择位数不同，此处不再截图说明。

4. 八数据通路模块：Multi_8CH32 设计

调用前一步所设计的多路选择器。,用 HDL 结构化调用和行为混合描述实现。

代码：

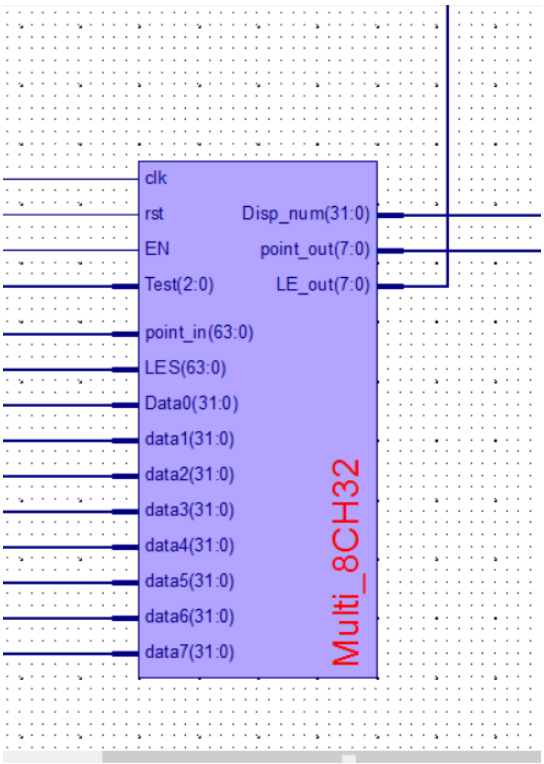
```
module     Multi_8CH32(input clk,
                           input rst,
                           input EN,             //Write EN
                           input[2:0]Test,     //ALU&Clock,SW[7:5]
                           input[63:0]point_in,
                                        //针对 8 位显示输入各 8 个小数点
                           input[63:0]LES,
                                        //针对 8 位显示输入各 8 个闪烁位
                           input[31:0] Data0,   //disp_cpudata
                           input[31:0] data1,
                           input[31:0] data2,
                           input[31:0] data3,
                           input[31:0] data4,
                           input[31:0] data5,
                           input[31:0] data6,
                           input[31:0] data7,
                           output [7:0] point_out,
                           output [7:0] LE_out,
                           output [31:0]Disp_num
                           );
    reg[31:0] disp_data = 32'hAA5555AA;
    reg[7:0]  cpu_blink = 8'b11111111, cpu_point = 4'b00000000;


    MUX8T1_32 Displaynum (
                          .s(Test),
                          .I0(disp_data),
                          .I1(data1),
                          .I2(data2),
                          .I3(data3),
                          .I4(data4),
                          .I5(data5),
                          .I6(data6),
                          .I7(data7),
                          .o(Disp_num)
                          );
    MUX8T1_8 Mux_point(
                          .s(Test),
                          .I0(cpu_point),
                          .I1(point_in[15:8]),
                          .I2(point_in[23:16]),
                          .I3(point_in[31:24]),
                          .I4(point_in[39:32]),
                          .I5(point_in[47:40]),
                          .I6(point_in[55:48]),
                          .I7(point_in[63:56]),
                          .o(point_out)
                          );
    MUX8T1_8 Mux_blink(
                          .s(Test),
                          .I0(cpu_blink),
                          .I1(LES[15:8]),
                          .I2(LES[23:16]),
                          .I3(LES[31:24]),
                          .I4(LES[39:32]),
```

```
                        .I5(LES[47:40]),
                        .I6(LES[55:48]),
                        .I7(LES[63:56]),
                        .o(LE_out)
                        );
    always@(posedge clk)begin
            if(EN) begin
                disp_data <= Data0;
                cpu_blink <= LES[7:0];
                cpu_point <= point_in[7:0];
            end
            else begin
                disp_data <= disp_data;
                cpu_blink <= cpu_blink;
                cpu_point <= cpu_point;
            end
    end


endmodule
```

调用逻辑符号：



五、实验结果与分析

根据给出的 UCF 定义，对应按键功能为——

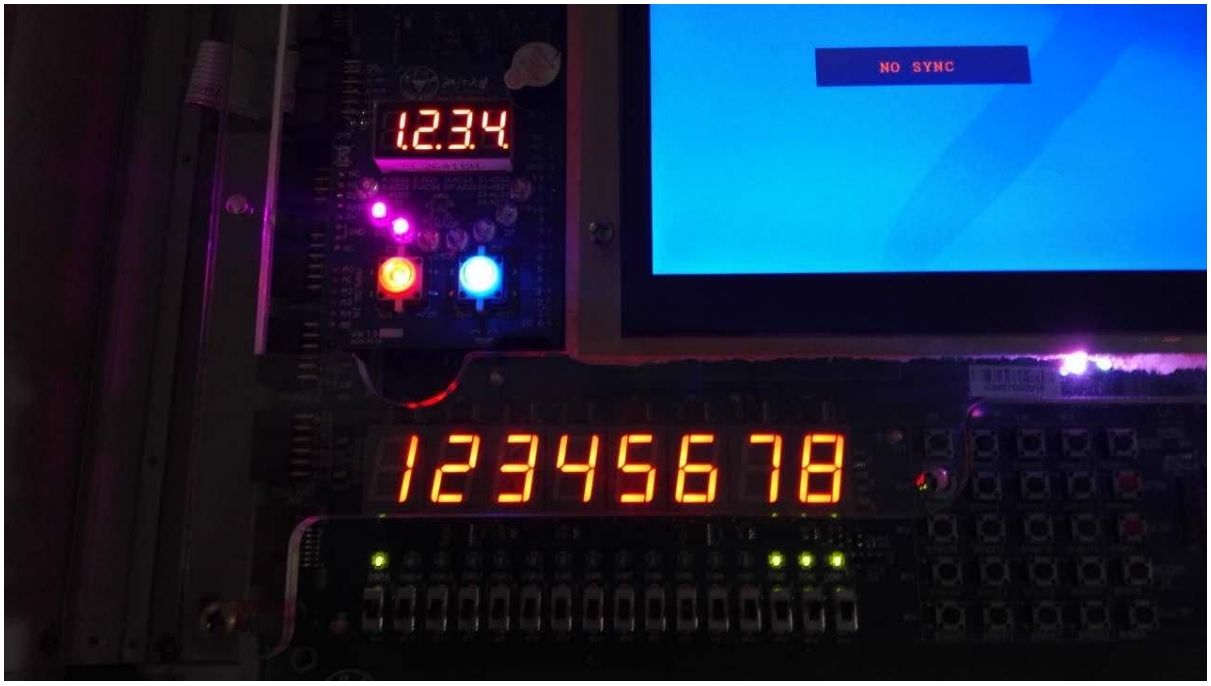
输入设备功能定义

开关定义	=0	=1	备注
SW[0]	图形(七段点阵)	文本(16进制)	
SW[1]	32位二进制高16位	32位二进制低16位	Arduino-Sword 002
SW[2]	CPU全速时钟	CPU单步钟	CPU时钟切换
SW[4]	存储器写禁止	存储器写使能	存储单元写控制
SW[7:5]	=000 =001 =010 =011 =100 =101 =110 =111	通道0 通道1 通道2 通道3 通道4 通道5 通道6 通道7	*Ai* *Bi* SUM(ALU_Out) Sign extension 1 bit Ext. to 32 bits 通用分频输出 ROM_D输出 RAM_B输出D(31:0)
按键定义	**=0**	**=1**	**备注**
BTN[0]		正脉冲左移	SW[15]=0,SW[7:5]<=001
BTN[1]		正脉冲右移	SW[15]=0,SW[7:5]<=001
BTN[2]		正脉冲输入修改	SW[15]=0,SW[7:5]<=001
RSTN			长按复位

SW[0]——文本图形选择

当开关处在低位，图形显示

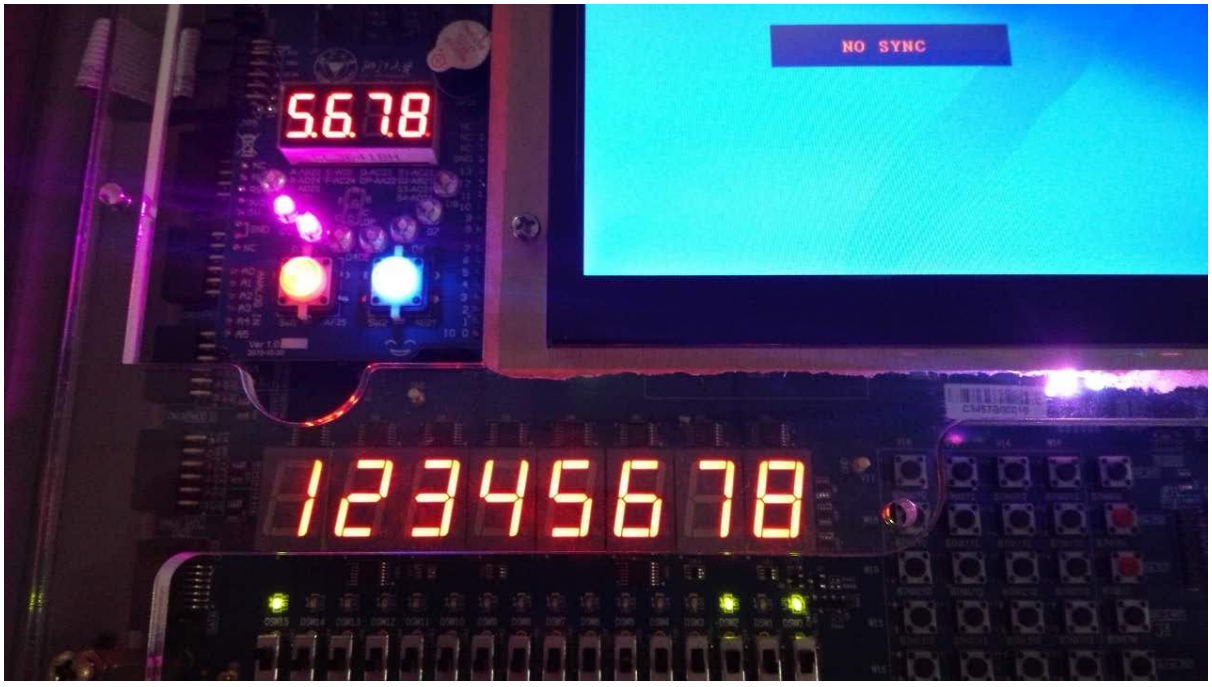
当开关处在高位，文本 16 进制显示数字 1 到 8



SW[1]——高低 16 位选择

当开关在高位，如上图所示，显示前 4 位高位

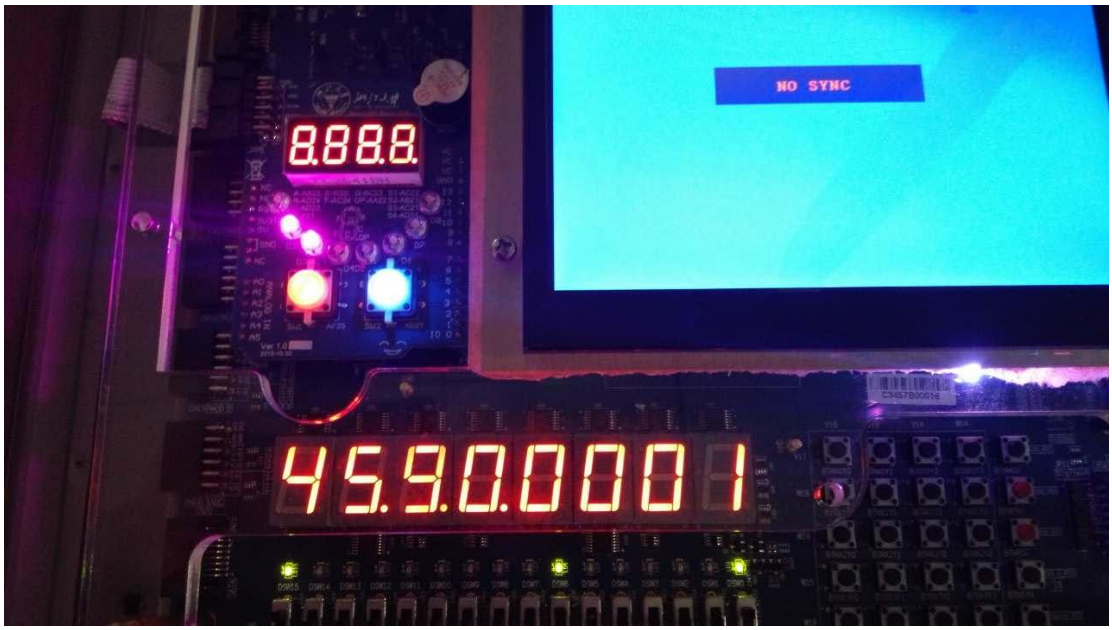
当开关在低位，显示后 4 位低位



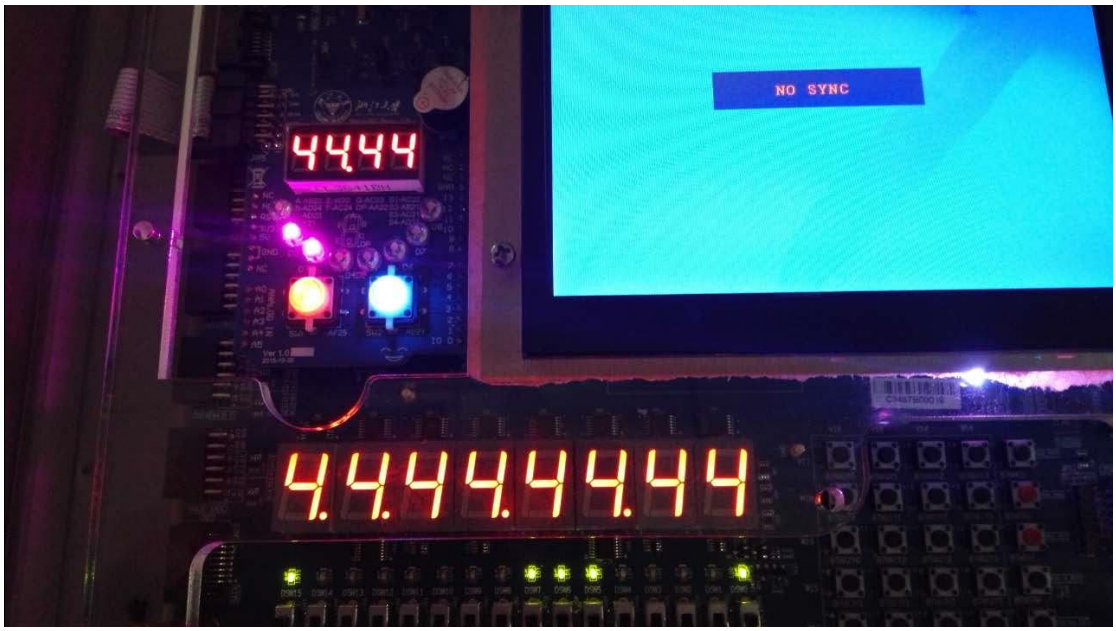
SW[7:5]——显示通道选择

000 时显示 12345678

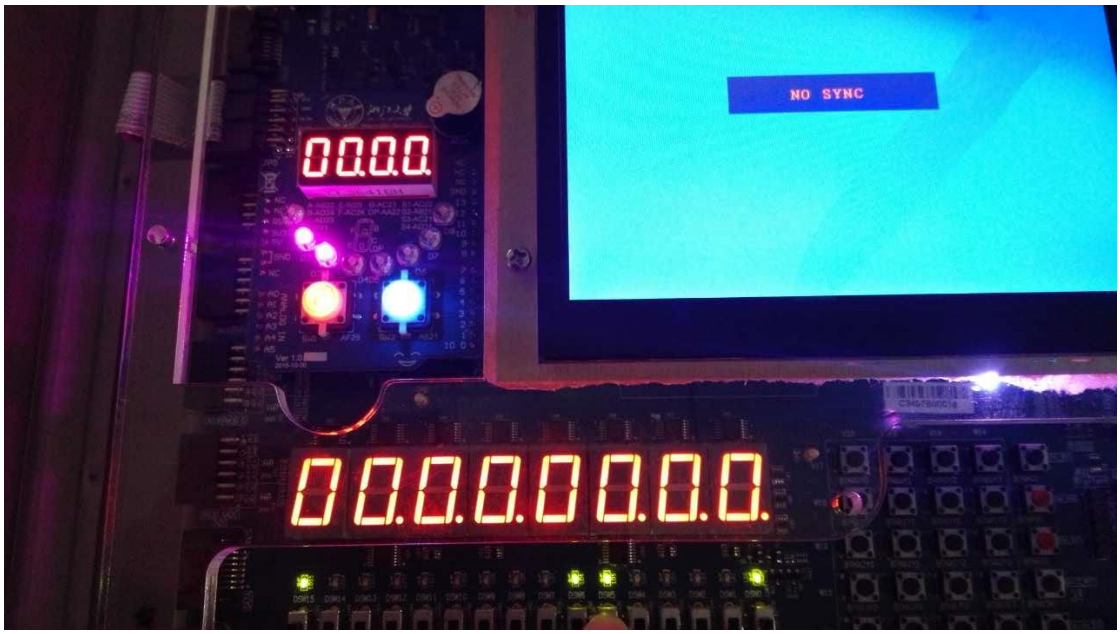
010 时，左 2、3 位快速跳动，左 1 位显示计数，从 0 到 F



111 时，所有位数同时跳动，从 0 到 F



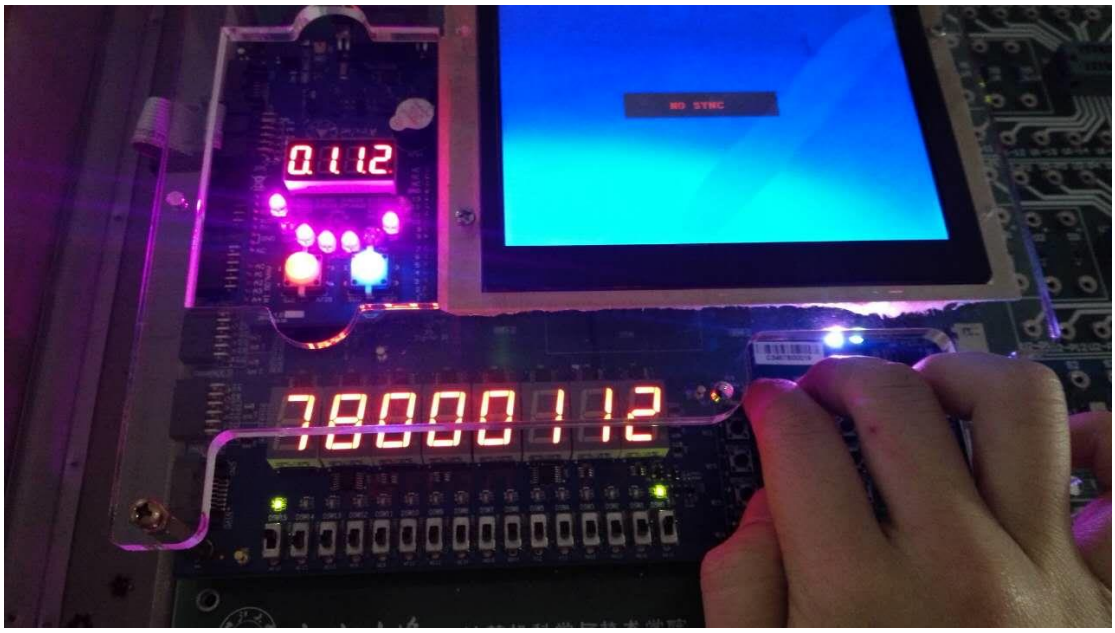
011 时，小数点移动闪烁

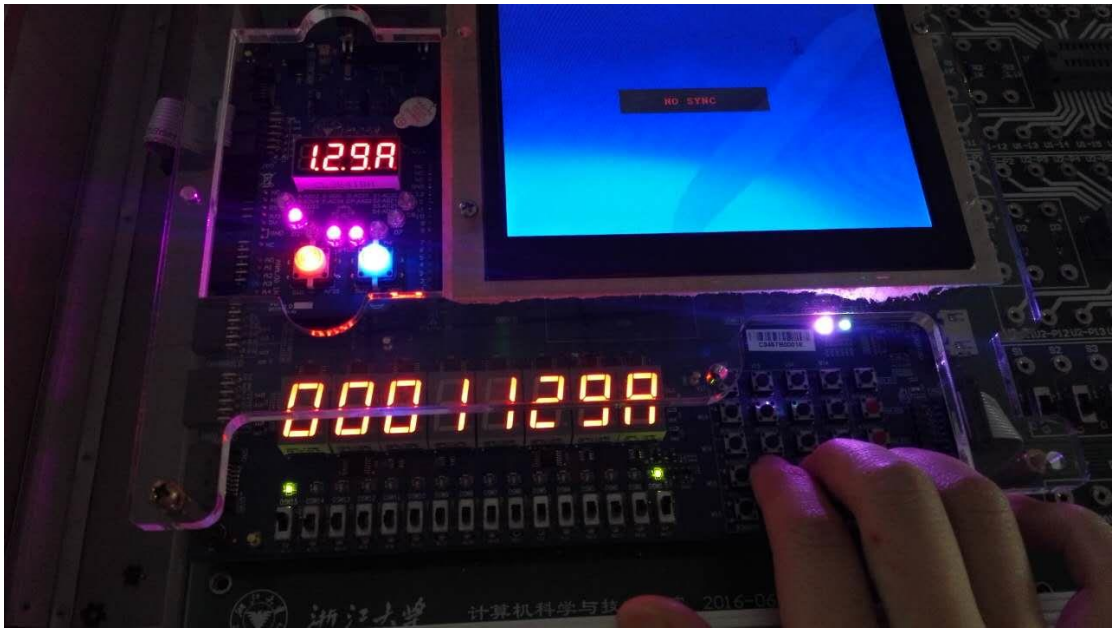


其他每个模式已在验收时展示，此处不全部列出。

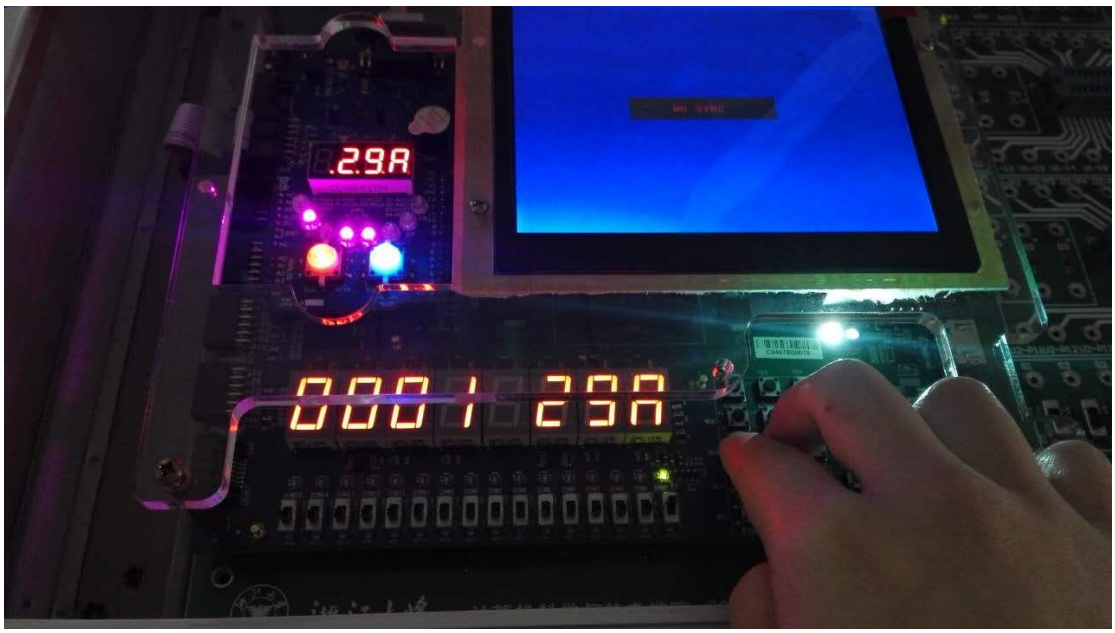
左一开关——按键输入方式选择

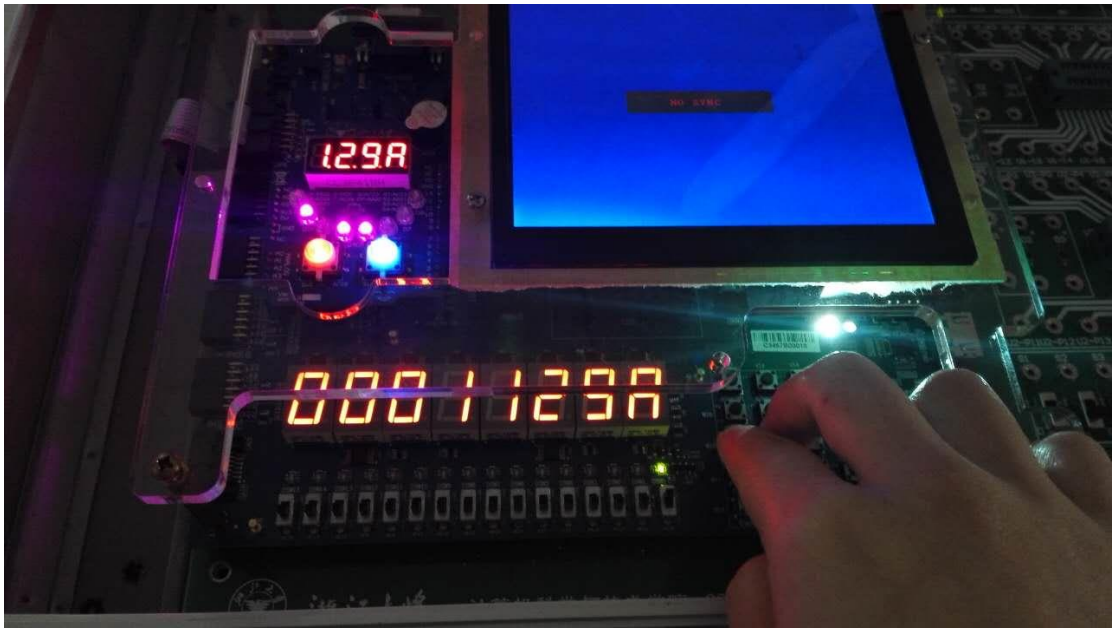
当开关在高位时，移位操作。每次小键盘出现操作，整体左移一位，同时新增位数位小键盘所按下的对应按键





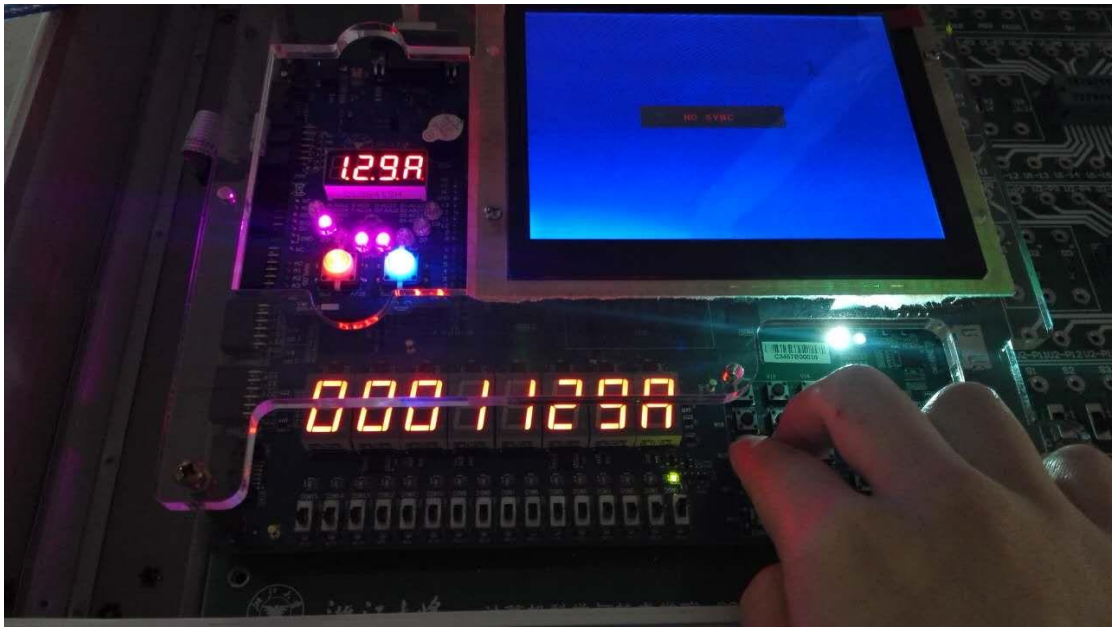
当开关在低位时，修改操作。通过小键盘选择要进行操作的位数，左边第、2 列分别为左移、右移。选中的位数会出现闪烁。

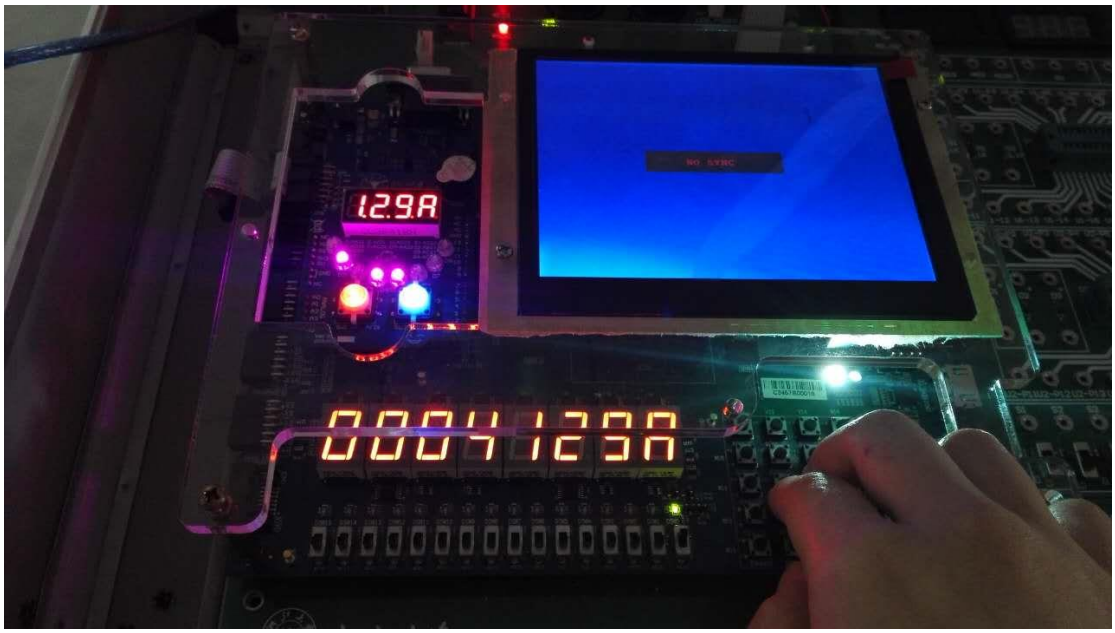




按下小键盘第三列，选中的位数每次加 1.

如图，所选择的位数为左数第四位。





六、讨论、心得

本次实验为此学期的第一次实验。

由于在数字逻辑设计课程中已经基本接触过 ISE 及 verilog 语言，因此此次试验较为简单。

在原理图设计实现过程中，连线时可能出现一些 bus 命名上的问题，需要小心注意，否则如果出现 bug，需要很长时间才能找到并修正。

多路选择器的描述和实现较为简单，在逻辑课上已经做过很多次。此处只需要自己设计一下仿真数据即可。

ROM 和 RAM 的 IP 核生成也较为简单，按步骤操作即可。要注意在选择 IP 时不能选错，对应的 COE 文件不要选反。

经过此次试验，日后实验的基本框架已经基本搭建起来，在 SWORD 实验板上的结果实现无误。如果还有时间，还可以通过修改不同通路的值实现更多功能，如 LED 跑马灯等。