实验一：多路选择器与 CPU 辅助模块设计

姓名：蒋仕彪 学号：3170102587 专业：计算机科学与技术

课程名称：计算机组成 同组学生姓名：无

实验时间：2019 年 3 月 实验地点：紫金港东 4-509 指导老师：马德

个人形象照：

一、实验目的和要求

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

二、实验内容和原理

2.1 实验任务

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

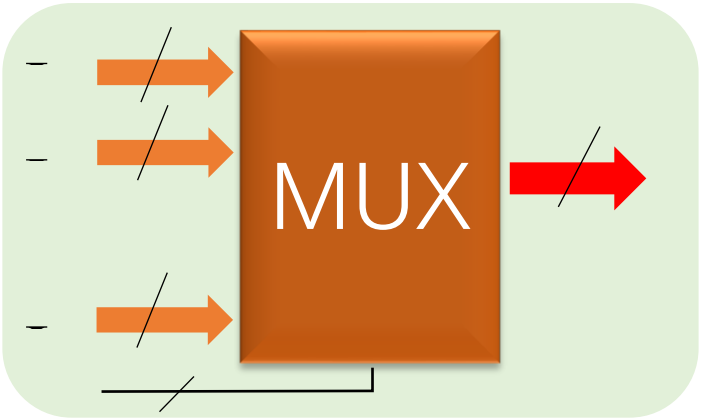
- 32 位 ROM、32 位 RAM

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

2.2 逻辑实验输出模块优化：多路器

- **多数选择器**
  - 逻辑结构如下图所示，在数字逻辑实验课设计过多种
  - 在 CPU 等部件设计将用到的重要模块
  - 本课程将用到的多数选择器有：
    - 2 选 1：5 位、32 位、8 位等
    - 4 选 1：5 位、32 位
    - 8 选 1：8 位、32 位

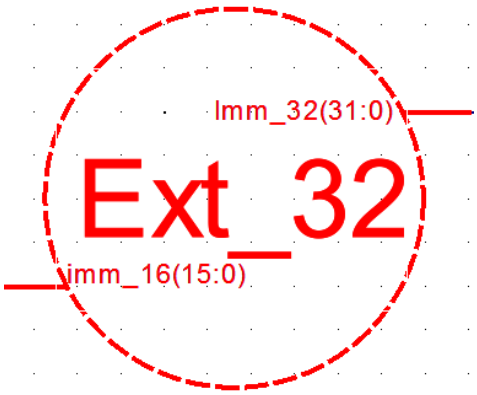


图表 1　多路器

2.2 逻辑实验输出模块优化：位扩展

- **16-32 位数算术扩展：Ext_32**
  - 16 位符号数扩展为 32 位符号数
  - 逻辑图形符号文件： **Ext_32.sym**

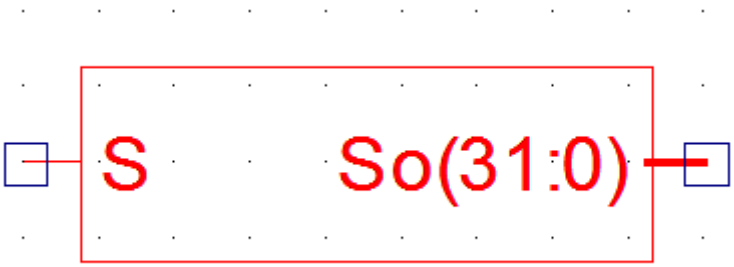
```
module      Ext_32(input   [15:0] imm_16,
                   output[31:0] Imm_32
                  );
      ……
endmodule
```



图表 2 Ext_32.sym

- **1 位信号算术扩展成 32 位： SignalExt_32**
  - 逻辑图形符号文件： **SignalExt_32.sym**

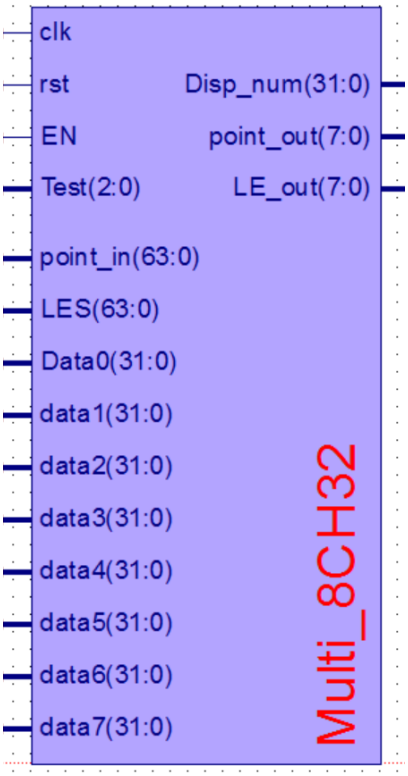
```
module    SignalExt_32 (input S,
                        output [31:0]So
                       );
          assign So = {32{S}};
endmodule
```



图表 3 SignalExt_32.sym

2.3 八数据通路模块：Multi_8CH32

- **多路选择器的简单应用**
  - 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
- **八路数据通路模块接口**
  - 与 8 位七段显示(32 位数据)器连接
  - I/O 接口接口信号功能
    - clk：同步时钟(后期扩展预留)
    - rst：复位信号(后期扩展预留)
    - 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 (闪烁位)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



图表 3 Multi_8CH32.sym

2.4 逻辑实验通用分频模块 M1 优化

- **通用计数分频模块**
  - 用于计算机组成实验辅助模块
  - 逻辑实验通用计数模块改造
  - 增加 CPU 单步时钟输出
  - 器件编号改为 **U8**
- **基本功能**
  - 32 位计数分频输出：clkdiv
  - CPU 时钟输出：Clk_CPU
  - SW[2]控制 Clk_CPU 输出
    - SW[2]=0，全速频率（50MHz 或 25MHz）
    - SW2=[1]，单步频率（2^{24} 分频， clkdiv [24]）
  - 核模块符号文档：clk_div.sym

图表 4 clk_div.sym

2.5 开关去抖动模块 M2 优化：SAnti_jitter.v

- **开关机械抖动消除模块(IP Core)**
  - 用于计算机组成实验辅助模块
  - 逻辑实验去抖动模块改造
  - 器件编号改为 **U9**
  - **Sword 平台提供 U9 的 IP 核**
- **基本功能**
  - 输入机械开关量
  - 输出滤除机械抖动的逻辑值
    - 电平输出：button_out、SW_OK
    - 脉冲输出：button_pluse(仅 Button)
    - RSTN：短按=CR，长按=rst
    - 其余功能不作要求(阵列键盘属接口课内容)
  - 核模块符号文档：SAnti_jitter.sym

2.6 双 32 位数据输入 IP 核 M4

- **32 位数据输入模块(IP Core)**
  - 逻辑实验的辅助模块
  - 本课程用于部件调试的初始值输入
  - 器件编号改为 **U10**
  - **Sword 平台提供 U10 的 IP 核**
- **基本功能**
  - 输入 32 位二进制数据(SW[15]=0)
    - Inc 单键输入：BTN(2)
    - 输出：Ai、Bi
    - 对应显示通道 0(SW[7:5]=000)、通道 1(SW[7:0]=001)
      - BTN(0)=左移、 BTN(1)=右移、修改位由 blink 指示闪烁
- **扩展功能**
    - 阵列式按键扫描码输入：由 SAnti_jitter 模块输出 Key_out
    - Din=5 位扫描码，D_ready=1 按键有效、readn=0 读扫描码
  - 核模块符号文档：SEnter_2_32.sym

图表 5 SEnter_2_32

2.7 七段码显示器 IP 核 M3：SSeg7_Dev

- **8 位七段码显示器(IP Core)**
  - 逻辑实验的输出显示模块
  - 本课程用于调试显示和 CPU 的简单外设
  - 器件编号改为 **U6**
- **基本功能**
  - 输入 32 位二进制数据：Hexs
    - SW[0]=1，显示 8 位 16 进制数， SW[0]=0，显示七段码 LED 点阵
      - SW[0]=1 时：SW[1]=1 高 16 位，SW[1]=0 低 16 位，
    - flash 七码闪烁频率，由通用分频器 U8(Div[25])提供，Start 串行扫描启动，point：七段小数点，LES：七段码使能，闪烁指示
  - 串行输出：seg_clk=时钟，seg_out=串行七段显示数据，SEG_PEN=使能，seg_clrn=清零
- **核模块符号文档：SSeg7_Dev.sym**
  - **由实验二优化扩展，本实验提供 U6 的 IP 核**

2.8 LED 并行显示模块 M6：SPIO

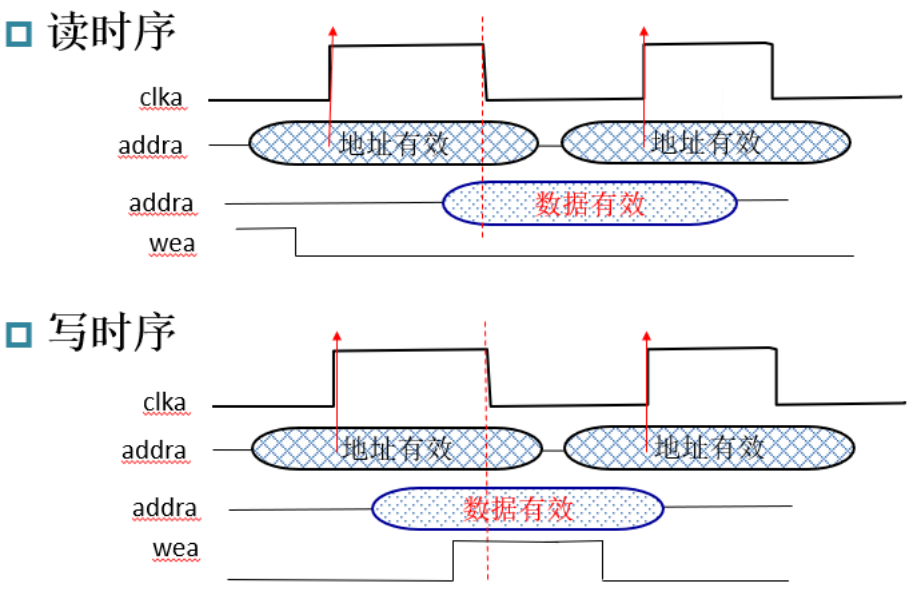
- **15 位 LED 指示灯控制(IP Core)**
  - 逻辑实验的输出 LED 显示模块
    - 相当于通用输入输出接口：GPIO
    - 15 位用于 LED 指示控制，其余用于扩展
  - 器件编号改为 **U7**
  - 本课程用于调试显示和 CPU 的简单外设
- **基本功能**
  - 输入 32 位二进制数据：P_Data
    - clk=时钟，EN：输出使能，Start：串行扫描启动，rst=复位
  - 串行输出：led_clk=时钟，led_sout=串行输出数据，LED_PEN=使能，led_clrn=清零
  - 并行输出：LED_out、counter_set、GPIOf0
- **核模块符号文档：SPIO.sym**
  - **本实验提供 U7 的 IP 核**

2.9 只读存储器 IP 核 M14-1 优化：

- **只读存储器**
  - 用于 CPU 应用的代码存储器
  - 逻辑实验 M14-1 模块优化
  - 模块名改为 ROM_B
  - 器件编号改为 U2
- **基本功能**
  - 容量：1024×32bit
  - 用 FPGA 内部存储器实现
    - Block Memory Generator 或 Distributed Memory Generator
  - 核模块符号文档：ROM_B.sym
    - 自动生成符号不规则，需要修整
  - ROM 初始化文档暂时不变
- **用 ISE 工具生成固核**
  - 用 IP Core Generator 向导生成

2.10 随机存储器 IP 核 M14-2 优化

- **随机存储器**
  - 用于 CPU 应用的数据或代码存储器
  - 逻辑实验 U14-2 模块优化
  - 模块名改为 RAM_B
  - 器件编号改为 U3
- **基本功能**
  - 容量：1024×32bit
  - 用 FPGA 内部存储器实现
    - Block Memory Generator
  - 核模块符号文档：RAM_B.sym
    - 自动生成符号不规则，需要修整
  - RAM 初始化文档无
- **用 ISE 工具生成固核**
  - 用 IP Core Generator 向导生成
  - 核调用模块 RAM_B.xco
  -



图表 6　读时序写时序

三、主要仪器设备

1. 计算机（Intel Core i5 以上，4GB 内存以上）系统
2. Sword 开发板
3. Xilinx ISE14.7 及以上开发工具

四、操作方法与实验步骤

4.1 设计要求

◎ **设计八数据通路模块：Multi_8CH32**

◎ **设计 32 位存储器**

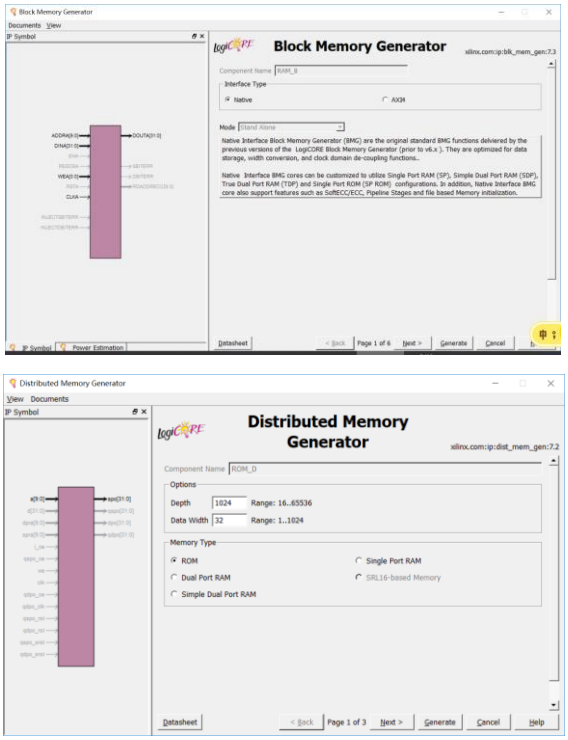
- ROM：32×1024： **ROM_D**
- RAM： 32×1024：**RAM_B**
  - **B = Block Memory**
  - **D = Distributed Memory**

◎ **搭建物理验证输入输出平台**

- 调用核或已设计模块实现
  - 开关去抖模块(IP 核)：U9
  - 数据输入模块(IP 核)：M4
  - 通用分频模块(clk_div)：U8
  - 八数据通路模块(Multi_8CH32)：U5
  - 七段显示模块(SSeg7_Dev 核)：U6
  - LED 显示模块(SPIO 核模块)：U7

4.2 RAM 和 ROM 的设计

已参考 PPT 附页设计。

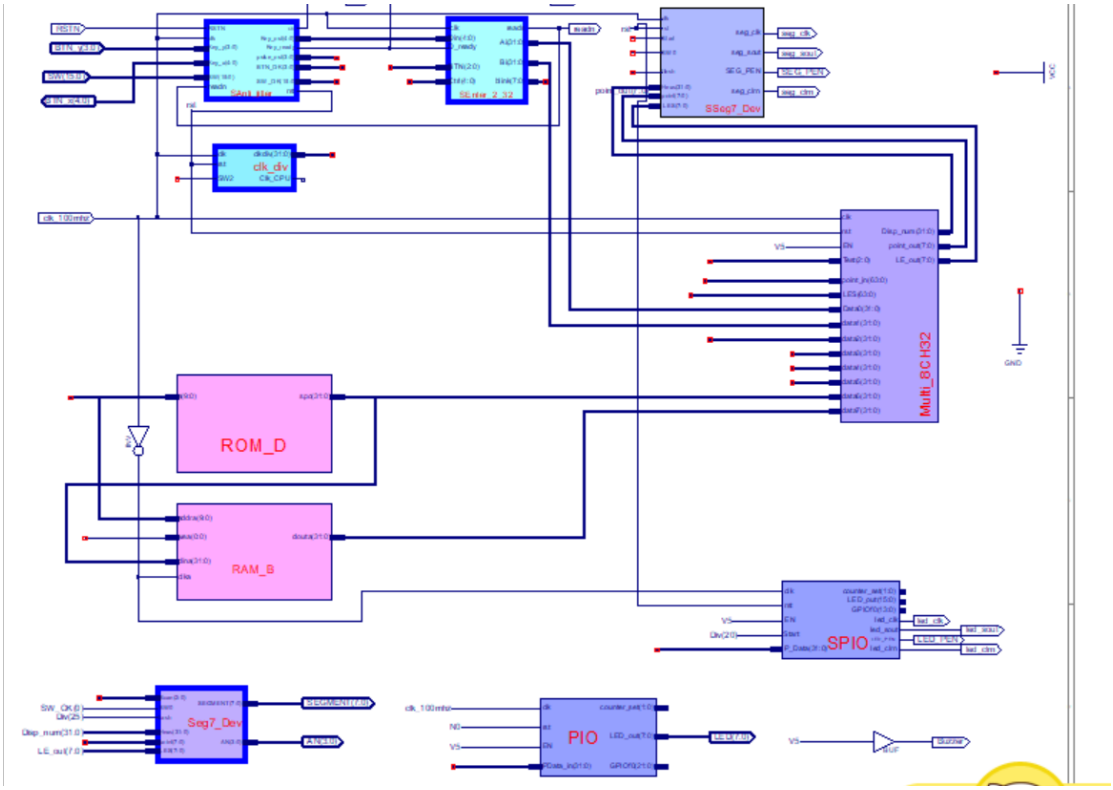


图表 7 ROM 和 RAM 配置

4.3 顶层模块设计

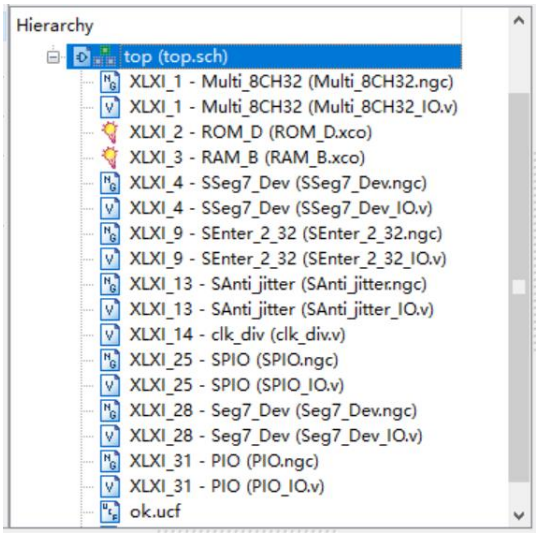
OExp01_MUX.sch 是参考附录输入顶层逻辑电路描述设计。

这个顶层模块比较复杂，需要在载入各个模块逻辑符号后，一丝不苟地用线将它们连起来。由于写 verilog 代码得不到简化，且不利于查错，我依然使用基本连线的方法。



图表 8　顶层结构图

4.4 总的结构图

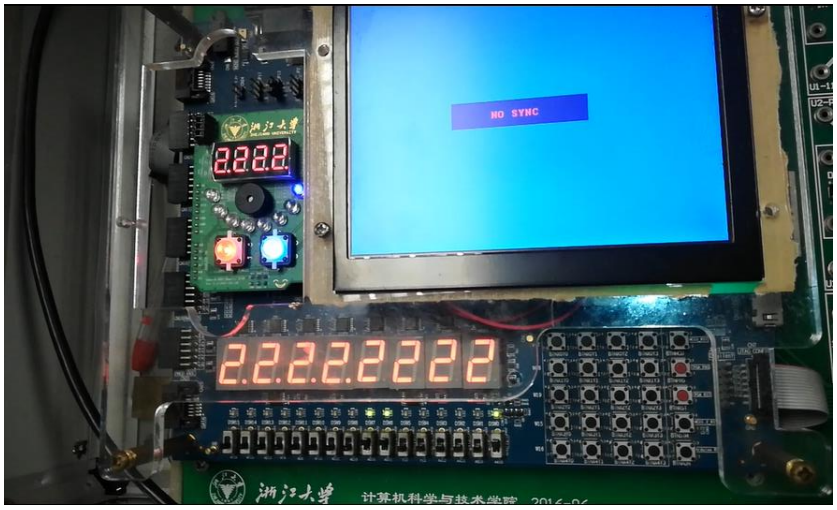


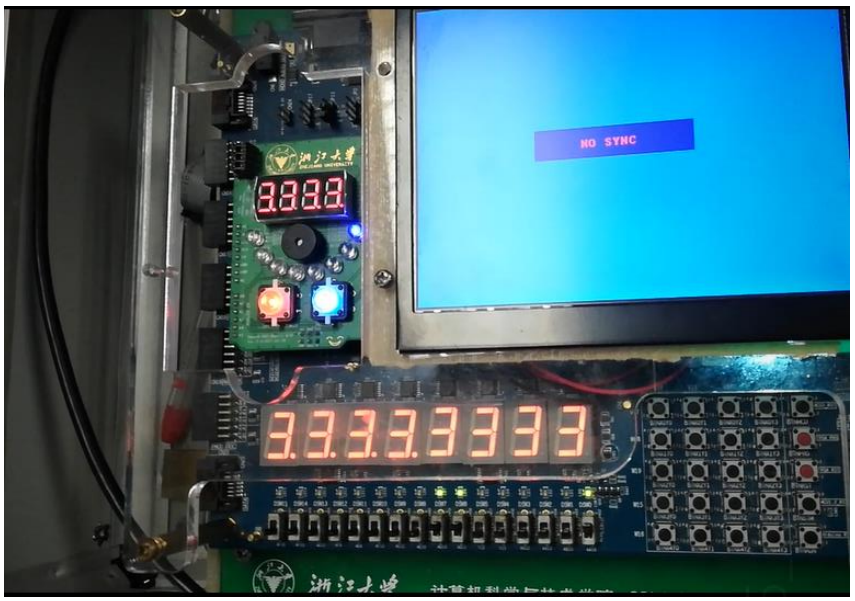
图表 9　总体构架

五、实验结果与分析

5.1 实验现象一

SW[0]=1,SW[7:5]=110。
SW[0]=1 表示文字模式，SW[7:5]表示通道选择，此时选了 ROM 里的数据。
结果是，屏幕上所有数字从 0 开始全体向上跳，每次加一。

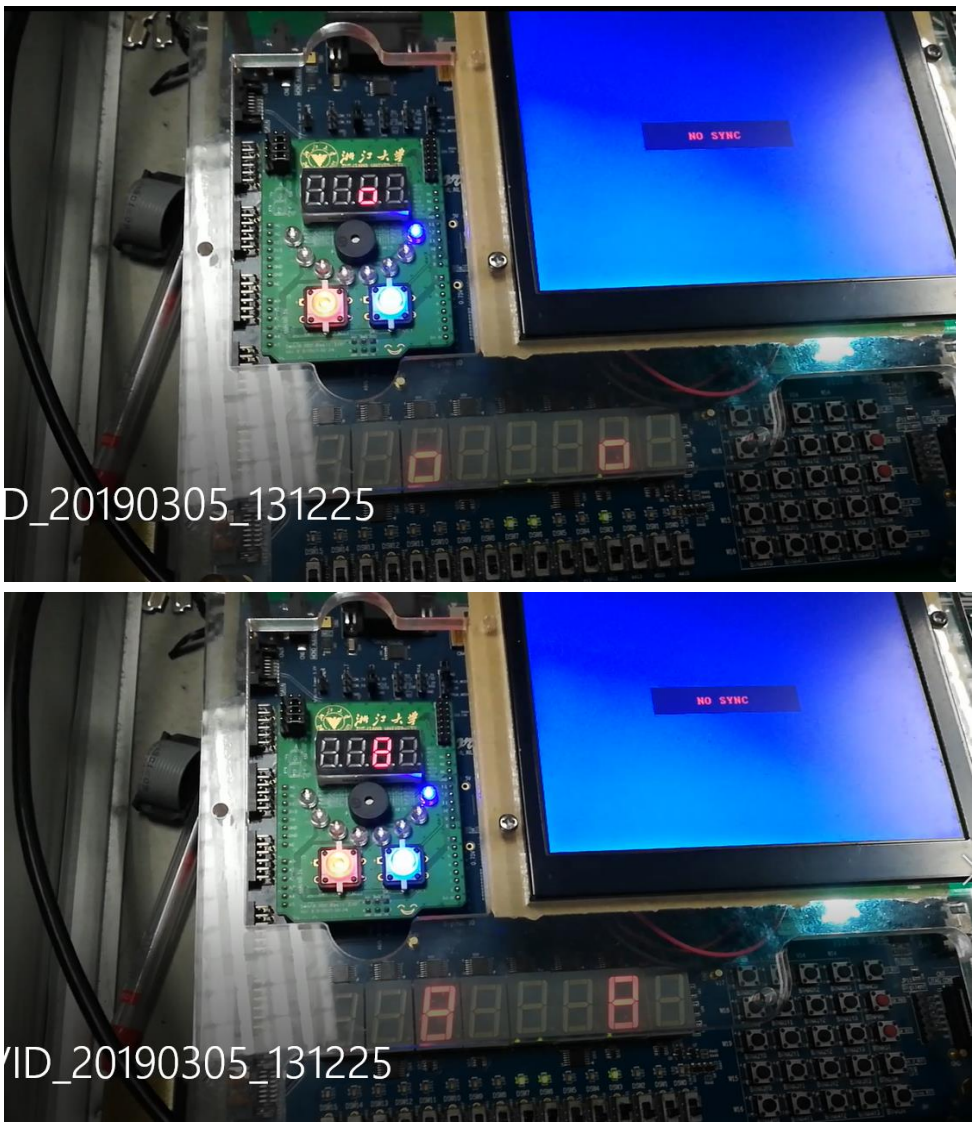




图表 10　实验现象一

5.2 实验现象二

SW[0]=0,SW[7:5]=111。
SW[0]=0 表示图像模式，SW[7:5]表示通道选择，此时选了 RAM 里的数据。
结果是，屏幕上的方块开始跳舞。

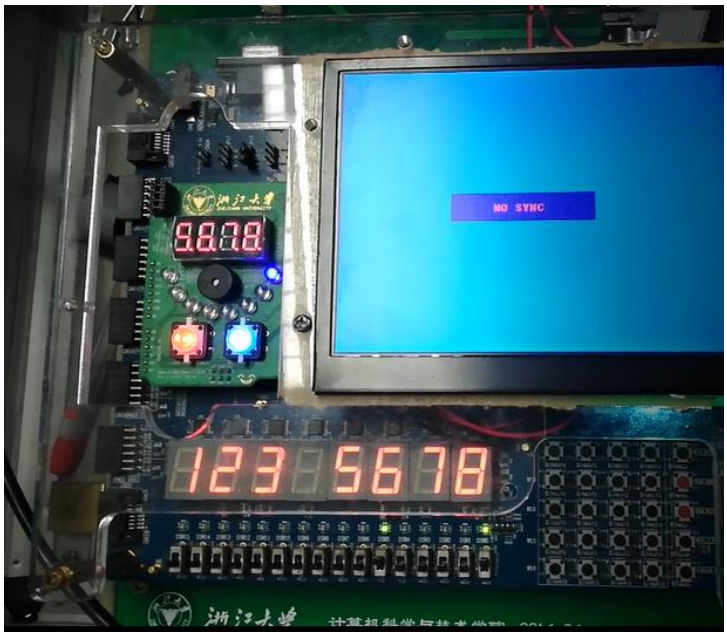


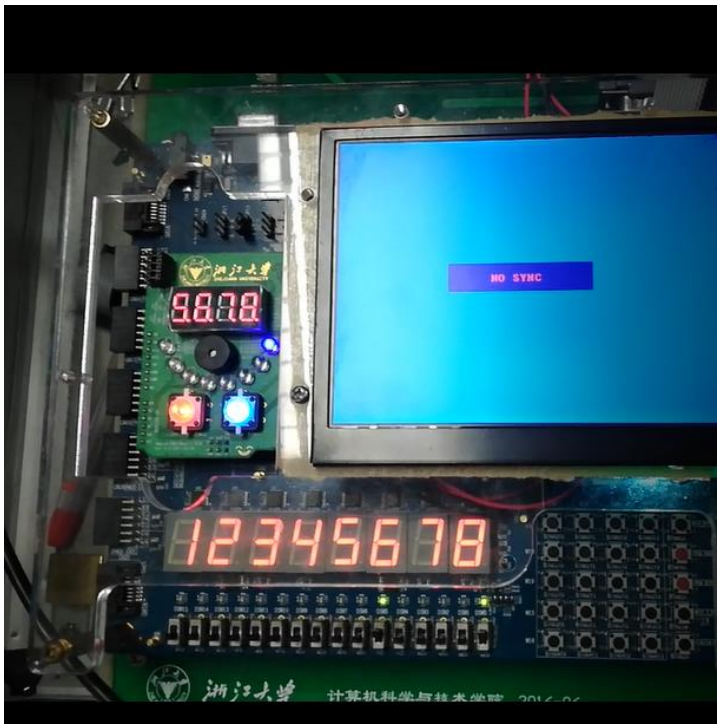
图表 11　实验现象二

5.3 实验现象三

SW[0]=1,SW[7:5]=001,SW[15]=1。

此时位于文字模式，处于修改数字模式。可以移动左右键来控制闪烁的数字，并可以每次加一来修改它。





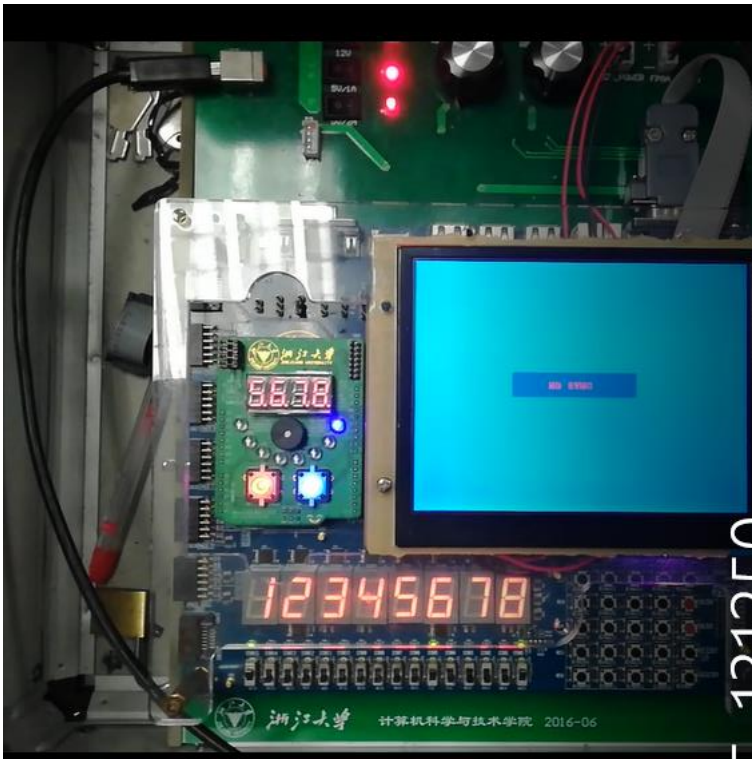
图表 12　实验现象三

5.4 实验现象四

SW[0]=1,SW[7:5]=001。

通道显示 12345678。

此时 SW[15]=1，处于键盘输入模式，可以直接向后追加数字。



图表 13　实验现象四

六、讨论、心得

本次实验是计算机组成的第一个实验。

还好有数字逻辑设计的基础，在前期的画顶层逻辑时，我细致而准确地完成了任务。

但我还是遇到了不少问题。如果一条 out 的线要连到多个 in 的话，第二次连的时候是被不允许的；要在 edit 里，选择 merge bus。我还在 map 时也报错过，因为有些 out 并没有连线。事实证明，即使没有要连出去的地方，依然要导出一条线。

配置和加入 ROM,RAM，应该是计组里新的东西。而通过这个精彩的 PPT，我也领悟了一个紧密系统的魅力。听说今后要一个个部件拆下来手写，感觉好厉害！

实验二：七段显示部件(设备)扩展实验报告

姓名：蒋仕彪 学号：3170102587 专业：计算机科学与技术

课程名称：计算机组成 同组学生姓名：无

实验时间：2019 年 3 月 实验地点：紫金港东 4-509 指导老师：马德

一、实验目的和要求

1. 了解设备与接口
2. 了解人机交互
3. 了解计算机通讯
4. 了解最简单的接口 GPIO
5. 了解用 GPIO 实现简单的人机交互

二、实验内容和原理

2.1 实验任务

1. 优化逻辑实验输出的显示模块

- 将原理转化为结构化行为描述
- 增加七段码文本图形显示

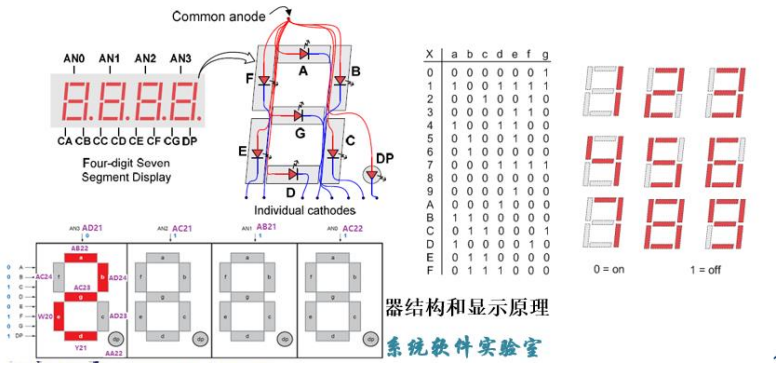
2. 在 Exp01 上增加修改验证

2.2 七段译码器文本与图形显示多数选择器

- **7 段码显示器用途**
  - 在 CPU 应用中可以作为一个外设
    - 简单人机交互显示
  - CPU 设计中用于测试调试显示
    - 硬件调试状态显示
  - 数字系统输出显示

2.2.1 七段译码器文本显示

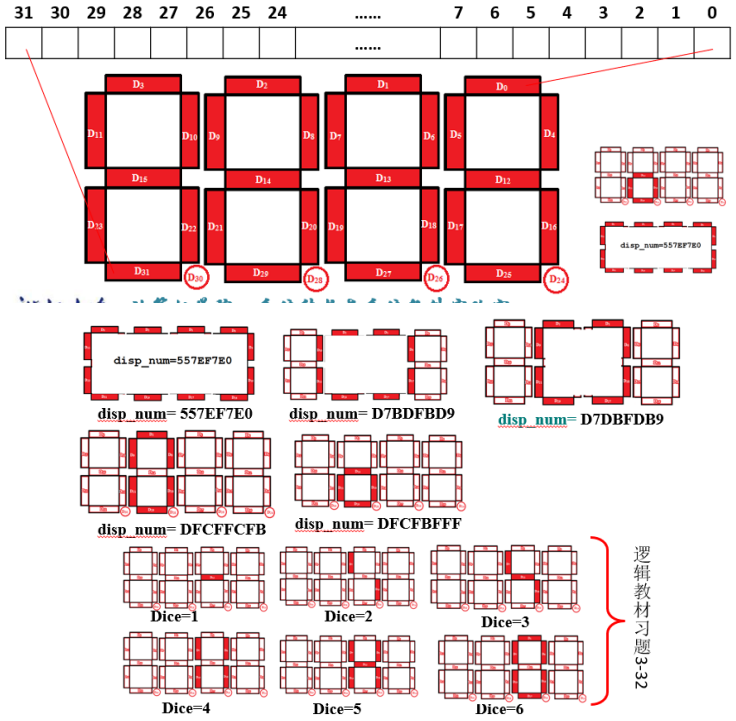
- **32 位数显示**
  - SW[0] =1：32 位数分 2 个 16 位显示
  - SP3 平台需要分二个 16 位显示，用 SW[1]选择；
    - SW[1]=0 低 16 位 SW[1]=1 高 16 位；



- 图表 1 文本显示

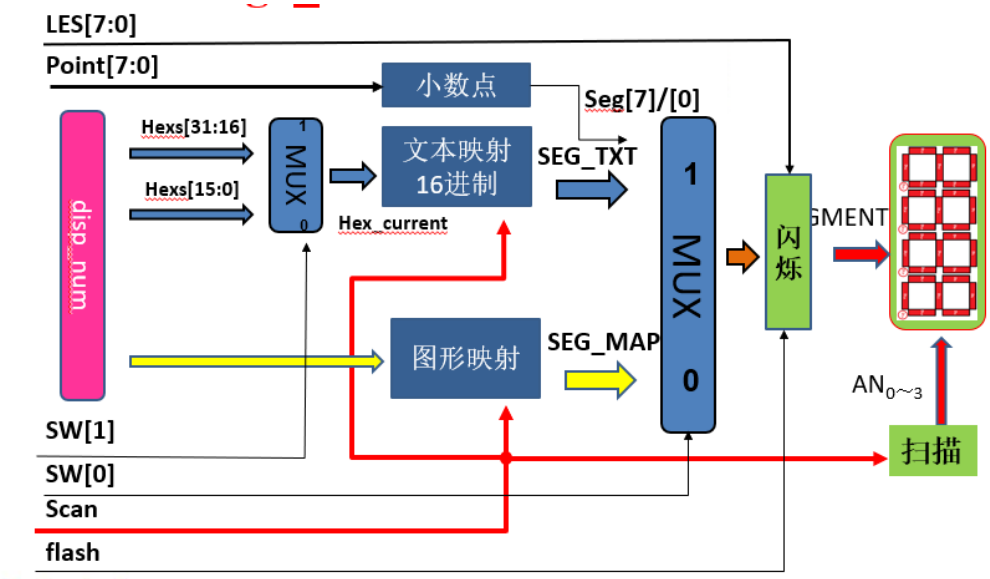
2.2.2 七段译码器图形显示：

- **7 段译码器映射为点阵显示**
  - SW[0] =0：4 位 7 段码映射为图形显示，共 32 段
  - 映射到一个 32 位字：$D_{31},D_{30},\ldots\ldots D_1,D_0$
    - 方便程序控制，显示任意形状



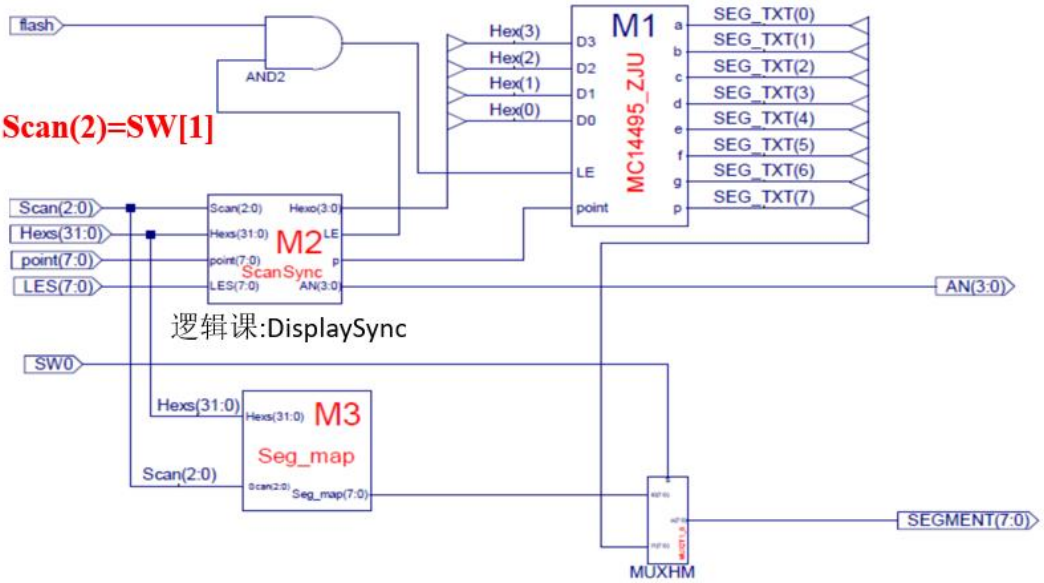
- 图表 2 图像显示

2.3 七段显示器译码电路逻辑结构



图表 3 SP3 平台 Seg7_Dev 逻辑结构（Arduino 板）

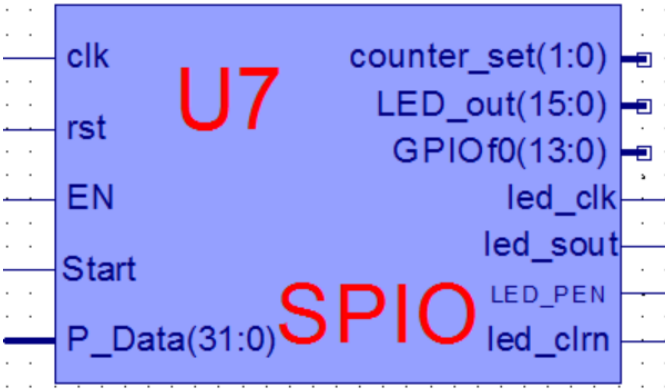
2.4 Arduino Sword-002 四位七段显示结构



图表 4 U61:Seg7_Dev(Seg_Map)七段显示器点阵映射模块

2.5 LED 并行显示模块 U7：SPIO

- **15 位 LED 指示灯控制(IP Core)**
  - 逻辑实验的输出 LED 显示模块
    - 相当于通用输入输出接口：GPIO
    - 15 位用于 LED 指示控制，其余用于扩展
  - 本课程用于调试显示和 CPU 的简单外设
- **基本功能**
  - 输入 32 位二进制数据：P_Data
    - clk=时钟，EN：输出使能，Start：串行扫描启动，rst=复位
  - 串行输出：led_clk=时钟，led_sout=串行输出数据，LED_PEN=使能，led_clrn=清零
  - 并行输出：LED_out、counter_set、GPIOf0
- **核模块符号文档：SPIO.sym**
  - **本实验设计并替换 U7 的 IP 核**



- 图表 5 SPIO

2.6 并转串移位寄存核

- **Sword 平台板七段显示器和 LED**
  - 采用串行输出，由 74LS161 串入并出锁存显示(74LV164A:串行输入并行输出移位寄存器)
  - 8 位七段显示器共 64 个显示点
    - 64 位并串转换：**64 位并入串出移位寄存器**
    - 由 P2S.ngc 核承担
  - 16 个 LED
    - 16 位并串转换： **16 位并入串出移位寄存器**
    - 由 LED_P2S.ngc 核承担

三、主要仪器设备

1. 计算机（Intel Core i5 以上，4GB 内存以上）系统
2. Sword 开发板
3. Xilinx ISE14.7 及以上开发工具

四、操作方法与实验步骤

4.1 设计要求

◎ **设计实现八位七段显示器模块：SSeg7_Dev.v**
- 顶层用逻辑图实现，其余用 HDL 结构化描述

◎ **设计实现并行输出兼 LED 显示模块：SPIO.v**
- 用 HDL 结构化描述

◎ **集成替换实验一的 U6、U7 核**
- 重建实验一工程为：OExp02-7SEG
- 调用核或已设计模块实现
  - ⊙ 开关去抖模块(IP 核)：U9
  - ⊙ 数据输入模块(IP 核)：M4
  - ⊙ 通用分频模块(clk_div)：U8
  - ⊙ 八数据通路模块(Multi_8CH32)：U5
  - ⊙ 七段显示模块(SSeg7_Dev IP)：U6
  - ⊙ LED 显示模块(SPIO 模块)：U7

◎ **新建工程：OExp02-7SEG**

◎ **设计 8 位七段显示器静态译码模块：HexTo8SEG.v**
- 用 HDL 描述
- 仿真调试验证模块
  - ⊙ 激励输入：Hexs=32'h12345678 和 Hexs=32'hA5A5A5A5
  - ⊙ flash=1，LES=1

◎ **设计 8 位七段显示器点阵映射模块：Seg_Map.v（20）**
- 用 HDL 描述
- 暂时不仿真，待物理验证不正确再仿真

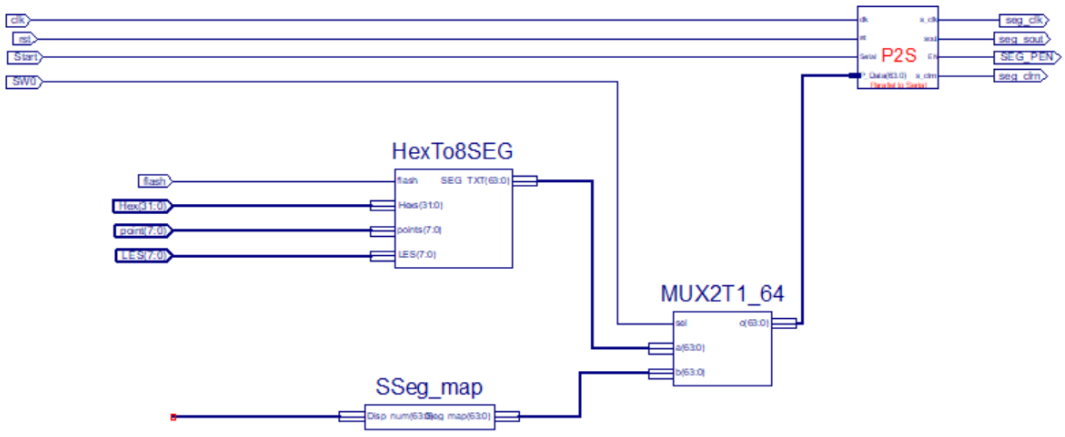
◎ **设计 8 位七段显示器:SSeg7_Dev.sch(25)**
- 用逻辑图描述
- 有核不做仿真

◎ **设计存储器核：同实验一**
- ROM_D、RAM_B

4.2 子模块 SSeg7_Dev.v

4.2.1 顶层模块

逻辑图清晰美观，故顶层模块是画图得到的。



图表 6 SSeg7_Dev.v 顶层模块

4.2.2 设计 8 位七段显示器静态译码模块：HexTo8SEG.v

手写 verilog 代码得到，代码如下：

```
module  HexTo8SEG(input [31:0] Hexs,       //端口变量说明与定义合并
//                    input [2:0] Scan,
                      input [7:0] points,
                      input [7:0] LES,
                      input flash,
                      output[63:0] SEG_TXT
                      );

    Hex2Seg              m0(.Hex(Hexs[3:0]),                              .
LE(LES[0]), .point(points[0]), .flash(flash), .Segment(SEG_TXT[7:0]))
;
    Hex2Seg
m1(.Hex(Hexs[7:4]),    .LE(LES[1]), .point(points[1]), .flash(flash),
.Segment(SEG_TXT[15:8]));
    Hex2Seg
m2(.Hex(Hexs[11:8]),   .LE(LES[2]), .point(points[2]), .flash(flash),
.Segment(SEG_TXT[23:16]));
    Hex2Seg
m3(.Hex(Hexs[15:12]), .LE(LES[3]), .point(points[3]), .flash(flash),
.Segment(SEG_TXT[31:24]));
    Hex2Seg
m4(.Hex(Hexs[19:16]), .LE(LES[4]), .point(points[4]), .flash(flash),
.Segment(SEG_TXT[39:32]));
    Hex2Seg
m5(.Hex(Hexs[23:20]), .LE(LES[5]), .point(points[5]), .flash(flash),
```

```
.Segment(SEG_TXT[47:40]));
    Hex2Seg
m6(.Hex(Hexs[27:24]), .LE(LES[6]), .point(points[6]), .flash(flash),
.Segment(SEG_TXT[55:48]));
    Hex2Seg
m7(.Hex(Hexs[31:28]), .LE(LES[7]), .point(points[7]), .flash(flash),
.Segment(SEG_TXT[63:56]));

endmodule

module  Hex2Seg(input[3:0]Hex,
                    input LE,
                    input point,
                    input flash,
                    output[7:0]Segment
                    );
       wire a,b,c,d,e,f,g,p;
       MC14495_ZJU
MSEG(.D3(Hex[3]), .D2(Hex[2]), .D1(Hex[1]), .D0(Hex[0]),

    .LE(flash&LE), .point(point), .a(a), .b(b), .c(c),
                            .d(d), .e(e), .f(f), .g(g), .p(p));
       assign Segment = {a,b,c,d,e,f,g,p};    //p,g,f,e,d,c,b,a
endmodule
```

4.2.3 HexTo8SEG.v 仿真

仿真代码如下：

```
module Pest;
    reg [31:0] Hexs;
    reg [7:0] points;
    reg [7:0] LES;
    reg flash;
    wire [63:0] SEG_TXT;
    HexTo8SEG uut (
        .Hexs(Hexs),
        .points(points),
        .LES(LES),
        .flash(flash),
        .SEG_TXT(SEG_TXT)
    );
    initial begin
        Hexs = 0;
        points = 0;
        LES = 1;
        flash = 1;

        Hexs=32'h12345678;
        #100;
        Hexs=32'hA5A5A5A5;
        #100;

    end

endmodule
```

仿真结果如下：

图表 7 HexTo8SEG.v

4.2.4 设计 8 位七段显示器点阵映射模块：Seg_Map.v

```
module SSeg_map(input[63:0]Disp_num,
              output[63:0]Seg_map
                   );

   assign   Seg_map   =   {Disp_num[0],   Disp_num[4],      Disp_num[16],
Disp_num[25], Disp_num[17], Disp_num[5],  Disp_num[12], Disp_num[24],

                          Disp_num[1],   Disp_num[6],      Disp_num[18],
Disp_num[27], Disp_num[19], Disp_num[7],  Disp_num[13], Disp_num[26],

                          Disp_num[2],   Disp_num[8],      Disp_num[20],
Disp_num[29], Disp_num[21], Disp_num[9],  Disp_num[14], Disp_num[28],
                          Disp_num[3],    Disp_num[10],    Disp_num[22],
Disp_num[31], Disp_num[23], Disp_num[11], Disp_num[15], Disp_num[30],

                          Disp_num[0],   Disp_num[4],      Disp_num[16],
Disp_num[25], Disp_num[17], Disp_num[5],  Disp_num[12], Disp_num[24],

                          Disp_num[1],   Disp_num[6],      Disp_num[18],
Disp_num[27], Disp_num[19], Disp_num[7],  Disp_num[13], Disp_num[26],

                          Disp_num[2],   Disp_num[8],      Disp_num[20],
Disp_num[29], Disp_num[21], Disp_num[9],  Disp_num[14], Disp_num[28],
                          Disp_num[3],    Disp_num[10],    Disp_num[22],
Disp_num[31], Disp_num[23], Disp_num[11], Disp_num[15], Disp_num[30]
};


endmodule
```

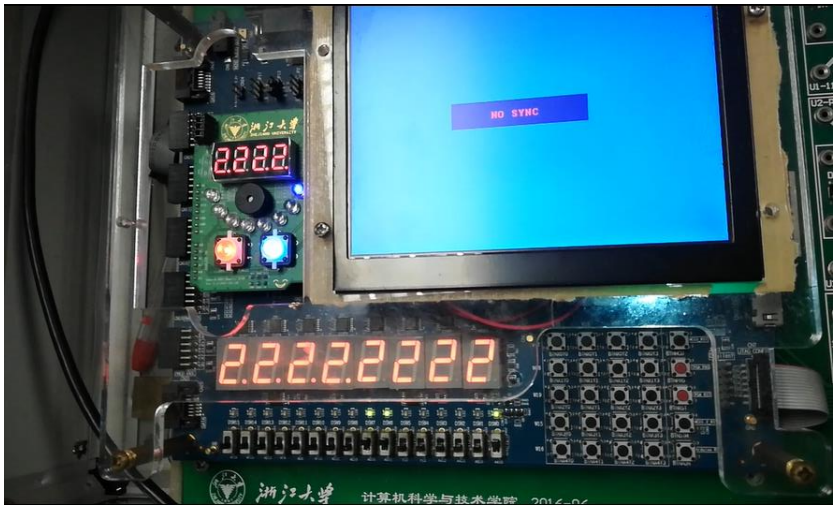
4.3 设计实现并行输出兼 LED 显示模块：SPIO.v

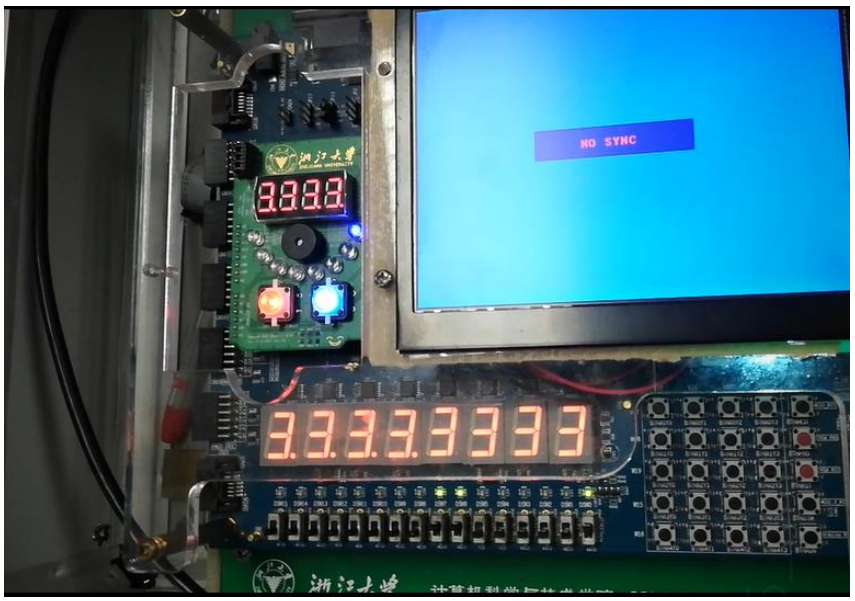
```
module     SPIO(input clk,                          //时钟
                    input rst,                  //复位
                    input Start,                //串行扫描启动
                    input EN,                   //PIO/LED 显示刷新使能
                    input [31:0] P_Data,          //并行输入，用于串行输
出数据
                    output reg[1:0] counter_set,  //用于计数/定时模块控
制，本实验不用
                    output [15:0] LED_out,       //并行输出数据
                    output wire led_clk,         //串行移位时钟
                    output wire led_sout,        //串行输出
                    output wire led_clrn,        //LED 显示清零
                    output wire LED_PEN,         //LED 显示刷新使能
                    output reg[13:0] GPIOf0         //待用：GPIO

                    );

   reg [15:0]LED;
   assign LES_out = LED;
   always @ (negedge clk or posedge rst)
      begin
         if (rst)
            begin
               LED <= 8'h2A;
               counter_set<=2'b00;
            end
         else
            if (EN)
               {GPIOf0[13:0],LED,counter_set} <= P_Data;
            else
               begin
                  LED <= LED;
                  counter_set <= counter_set;
               end
      end

   LED_P2S #(.DATA_BITS(16),.DATA_COUNT_BITS(4))
      LED_P2S(clk, rst, Start,

   {~{LED[0],LED[1],LED[2],LED[3],LED[4],LED[5],LED[6],LED[7],

   LED[8],LED[9],LED[10],LED[11],LED[12],LED[13],LED[14],LED[15]}},
               led_clk,
               led_clrn,
               led_sout,
               LED_PEN
            );

endmodule
```

五、实验结果与分析（同实验一）

5.1 实验现象一

SW[0]=1,SW[7:5]=110。
SW[0]=1 表示文字模式，SW[7:5]表示通道选择，此时选了 ROM 里的数据。
结果是，屏幕上所有数字从 0 开始全体向上跳，每次加一。

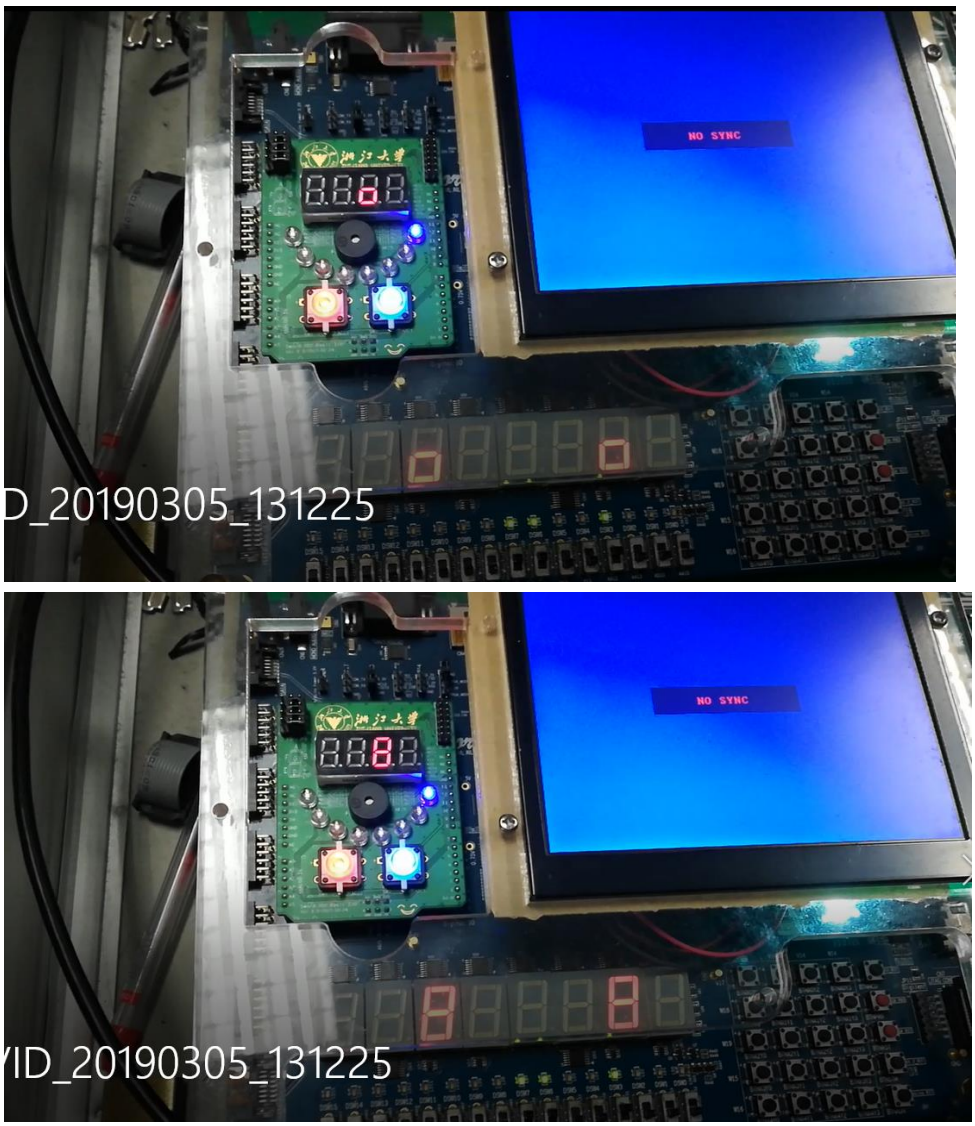




图表 10　实验现象一

5.2 实验现象二

SW[0]=0,SW[7:5]=111。
SW[0]=0 表示图像模式，SW[7:5]表示通道选择，此时选了 RAM 里的数据。
结果是，屏幕上的方块开始跳舞。

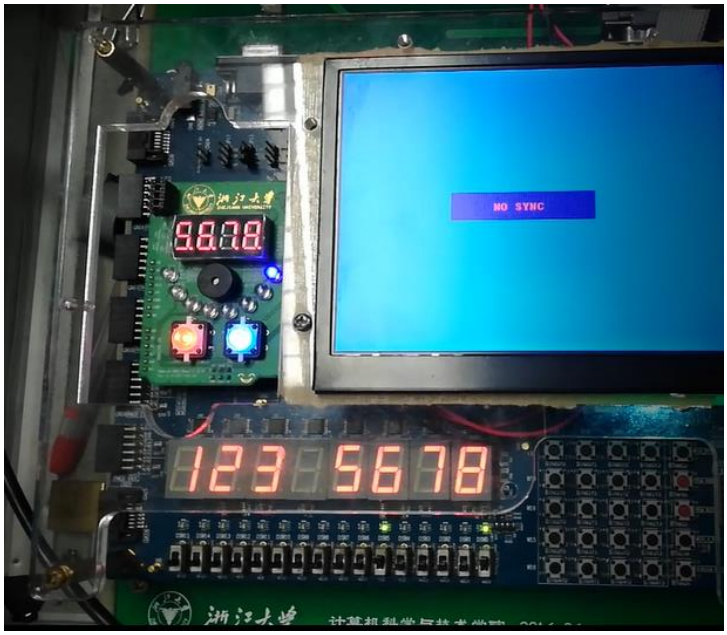


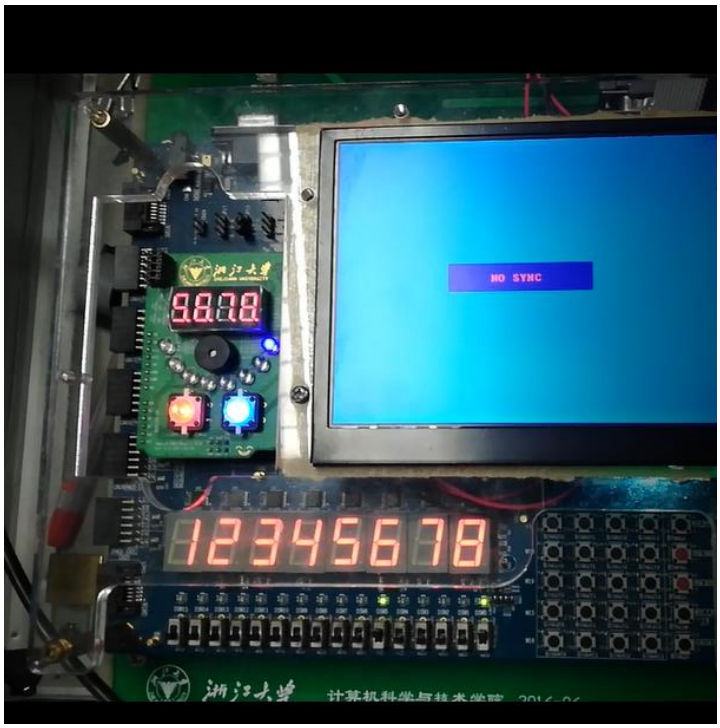
图表 11　实验现象二

5.3 实验现象三

SW[0]=1,SW[7:5]=001,SW[15]=1。

此时位于文字模式，处于修改数字模式。可以移动左右键来控制闪烁的数字，并可以每次加一来修改它。





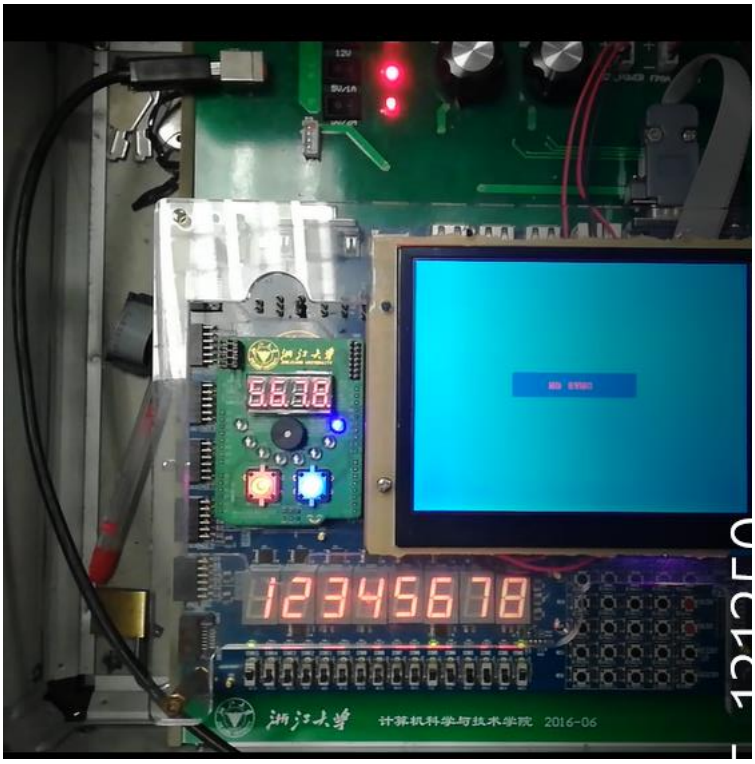
图表 12　实验现象三

5.4 实验现象四

SW[0]=1,SW[7:5]=001。

通道显示 12345678。

此时 SW[15]=1，处于键盘输入模式，可以直接向后追加数字。



图表 13　实验现象四

六、讨论、心得

本次实验是计算机组成的第二个实验。

如果说，第一个实验是“拼装实验”（所有模块都由老师提供，我们负责画顶层模块将他们串接成一个系统），那么第二个实验逐渐开始拆解其中的部分，并换成自己手写。

感觉计组的实验课都设计的很好。第一次实验虽然部件都是以 ngc 形式给出，但拼装成功后，能让我们初尝计组的魅力。之后的实验步步精细，逐渐拆解一个个结构，让我们深入了解其工作原理。而且现在实验的调试挺方便，只要观察一下，装上“自己的部件”后，是否还能像原来一样跑起来。如果哪里有问题，也能针对性地找问题。

实验三—IP 核集成 SOC 设计实验报告

姓名：蒋仕彪 学号：3170102587 专业：计算机科学与技术

课程名称：计算机组成 同组学生姓名：无

实验时间： 2019 年 3 月 实验地点：紫金港东 4-509 指导老师： 马德

一、实验目的和要求

1. 初步了解 GPIO 接口与设备
2. 了解计算机系统的基本结构
3. 了解计算机各组成部分的关系
4. 了解并掌握 IP 核的使用方法
5. 了解 SOC 系统并用 IP 核实现简单的 SOC 系统

二、实验内容和原理

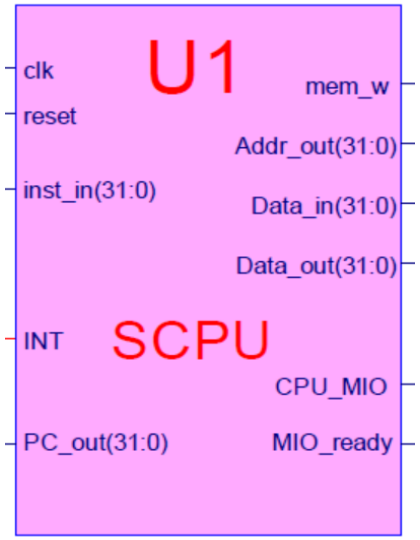
2.1 实验任务

1. **分析基本和接口 IP 核**
2. **设计存储器 IP 模块**
3. **练习掌握 IP 核的使用方法**
4. **选用第三方 IP 核和已有模块集成实现 SOC**
   - 此实验顶层用原理图设计实现

2.2 U1-CPU 模块：SCPU

- **MIPS 构架**
  - RISC 体系结构
  - 三种指令类型
- **实现基本指令**
  - 设计实现不少于下列指令
    - R-Type: add, sub, and, or, xor, nor, slt, srl*, jr, jalr, eret*;
    - I-Type: addi, andi, ori, xori, lui, lw, sw, beq, bne, slti
    - J-Type: J, Jal*;

- **本实验用 IP Core- U1**
  - 核调用模块 SCPU.ngc
  - 核接口信号模块(空文档)：SCPU.v
  - 核模块符号文档：SCPU.sym



图表 1 U1

2.3 U2-指令代码存储模块：ROM_B

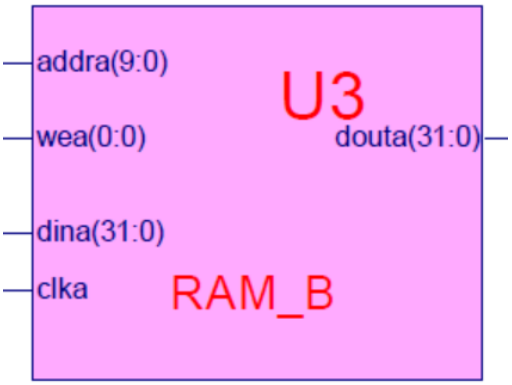
- **ROM_D/B**
  - 用实验一设计的模块
  - FPGA 内部存储器
    - Block Memory Generator 或 Distributed Memory Generator
  - 容量
    - 1024×32bit
  - 核模块符号文档：ROM_B.sym
    - 自动生成符号不规则，需要修整
- **本实验采用实验一生成的固核**
  - ROM 初始化文档：**I9_mem.coe**
  - 核调用模块 ROM_B.xco
    - 生成后自动调用关联，不需要空文档
- **ROM 调用接口信号**

```
ROM_B      U2 ( .clka(clk_m),        //存储器时钟，与 CPU 反向
.addra(PC_out[11:2]),    // ROM 地址 PC 指针，来自 CPU
.douta(inst[31:0])    // ROM 输出作为指令输入 CPU
        );
```

- **图形输入调用**
  - ROM_B.sym
  - **固核调用不需要空模块文档**
  - 接口文档

2.4 U3-数据存储模块：RAM_B

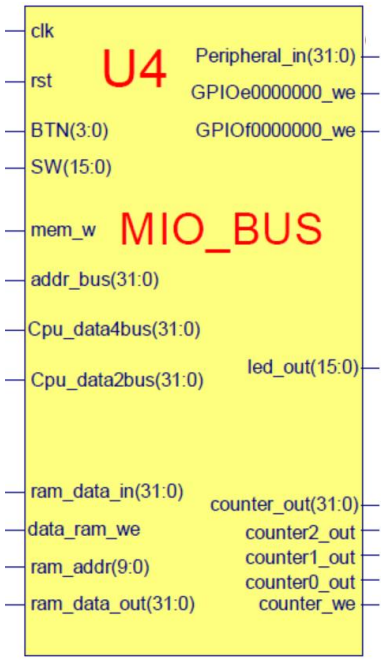
- **RAM_B**
  - FPGA 内部存储器
    - Block Memory Generator
  - 容量
    - 1024×32bit
  - 核模块符号文档：RAM_B.sym
    - 自动生成符号不规则，需要修整
- **本实验采用实验一生成的固核**
  - RAM 初始化文档：**D_mem.coe**
  - 核调用模块 RAM_B.xco
    - 生成后自动调用关联，不需要空文档



- 图表 2 U3

2.5 U4-总线接口模块：MIO_BUS

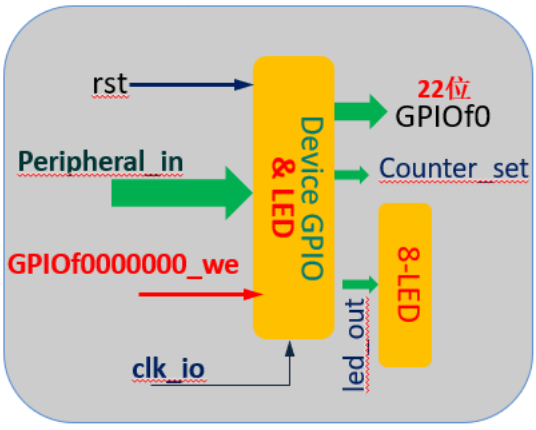
- **MIO_BUS**
  - CPU 与外部数据交换接口模块
  - 本课程实验将数据交换电路合并成一个模块
    - 非常简单，但非标准，扩展不方便
    - 后继课程采用标准总线
      - Wishbone 总线
- **基本功能**
  - 数据存储、7-Seg、SW、BTN 和 LED 等接口
- **本实验用 IP 软核- U4**
  - 核调用模块 MIO_BUS.ngc
  - 核接口信号模块(空文档)：MIO_BUS_IO.v
  - 核模块符号文档：MIO_BUS.sym



图表 3 U4

2.6 U7-外部设备模块：GPIO 接口及设备一 SPIO

- **GPIO 输出设备一**
  - 地址范围=f0000000 - ffffffff0 (ffffff00-fffffff0)
  - 读写控制信号：**GPIOf0000000_we(GPIOffffff00_we)**
  - {GPIOf0[21:0],LED,counter_set}
- **基本功能**
  - LEDs 设备和计数器控制器读写
  - 可回读，检测状态
  - 逻辑实验 LED 模块改造
- **本实验用 IP 软核- U7**
  - 核调用模块 SPIO.ngc
  - 核接口信号模块(空文档)：SPIO_IO.v
  - 核模块符号文档：SPIO.sym



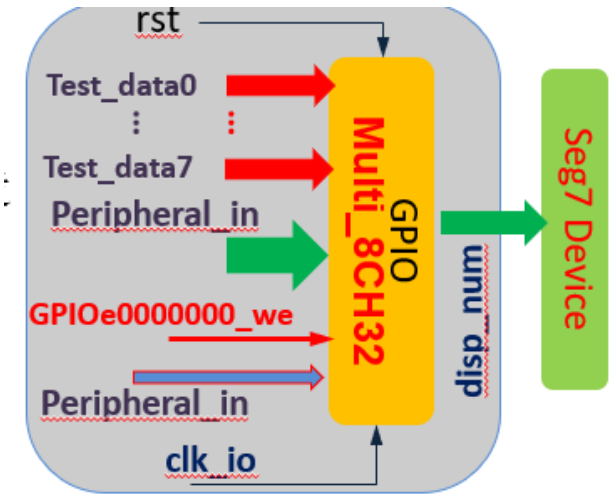
图表 4 U7

2.7 U6-外部设备模块：GPIO 设备二~Seg7_Dev

- 七段码显示输出设备模块
  - 需要通过接口模块 **Multi_8CH32** 与 CPU 连接
  - 地址范围=E0000000 - EFFFFFFF (FFFFFE00-FFFFFEFF)
- 基本功能(参考 OExp02)
  - 4 位 7 段码显示设备
  - 模拟文本，显示 8 位 16 进制数：SW[1:0]=x1
    - SW[1:0]=01，显示低 4 位； Sword 平台不需要
    - SW[1:0]=11，显示高 4 位 Sword 平台不需要
  - 模拟图形显示，4 位 7 段用于 32 个点阵显示，SW[1:0]=x0
  - 逻辑实验 7 段显示模块改造
- 本实验用 IP 软核或 Exp02 设计的模块- **U5**
  - 核调用模块 SSeg7_Dev.ngc
  - 核接口信号模块(空文档)：SSeg7_Dev.v
  - 核模块符号文档：SSeg7_Dev.sym

2.8 U5-通用设备二接口模块

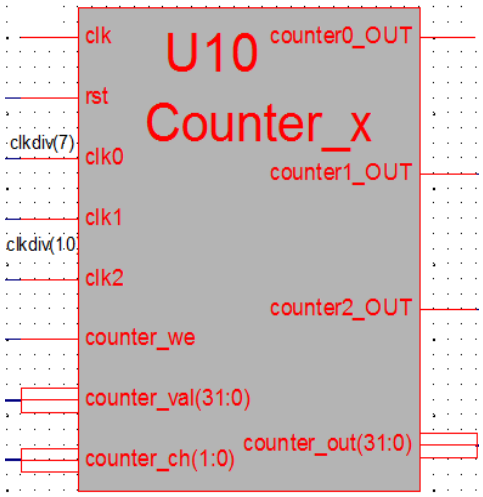
- **GPIO 输出设备二接口模块**
  - 地址范围=E0000000 - EFFFFFFF (FFFFFE00-FFFFFEFF)
  - 读写控制信号：**GPIOe0000000_we(GPIOfffffe00_we)**
- **基本功能(参考 Exp02)**
  - 7 段码输出设备接口模块
  - 逻辑实验显示通道选择模块改造
  - 通道 0 作为显示设备接口
    - **GPIOe0000000_we=1**
    - **CLK 上升沿**
  - 通道 1-7 作为调试测试信号显示
- **本实验用 IP 软核或 EXp02 设计的模块- U6**
  - 核调用模块 Multi_8CH32.ngc
  - 核接口信号模块(空文档)： **Multi_8CH32_IO.v**
  - 核模块符号文档：Multi_8CH32.sym



图表 5 U5

2.9 U10-外部设备五：通用计数器模块

- **通用计数器设备，双向**
  - 地址范围=F0000004 – FFFFFFF4 (FFFFFF04-FFFFFFF4)
  - 读写控制信号：**counter_we**
- **基本功能**
  - 三通道独立计数器，可用于程序定时。
  - 输出用于计数通道设置或计数值初始化
    - counter_set=00、01、10 对应计数通道 0、1、2
    - counter_set=11 对应计数通道工作设置
  - 计数器部分兼容 8253
- **本实验用 IP 软核- U10**
  - 核调用模块 Counter_x.ngc
  - 核接口信号模块(空文档)：Counter_x.v
  - 核模块符号文档：Counter_x.sym



图表 6 U10

2.10 U8-通用分频模块：clk_div

- **计数分频模块**
  - 用于要求不高的各类计数和分频
    - CPU、IO 和存储器等
  - 对延时和驱动有要求的需要 BUFG 缓冲
  - 对于时序要求高的需要用 DCM 实现
- **基本功能**
  - 32 位计数分频输出：clkdiv
  - CPU 时钟输出：Clk_CPU
  - 逻辑实验通用计数模块改造
- **本实验自己设计核(逻辑电路输出)- U8**
  - 核调用模块 clk_div.ngc
  - 核接口信号模块(空文档)：clk_div.v
  - 核模块符号文档：clk_div.sym

2.11 U9-开关去抖动模块：SAnti_jitter

- **开关机械抖动消除模块**
  - 用于消除开关和按钮输入信号的机械抖动
    - CPU、IO 和存储器等
- **基本功能**
  - 输入机械开关量
  - 输出滤除机械抖动的逻辑值
    - 电平输出：button_out、SW_OK
    - 脉冲输出：button_pluse
  - 逻辑实验模块
- **本实验可自己设计或用 IP 软核- U9**
  - 核调用模块 SAnti_jitter.ngc
  - 核接口信号模块(空文档)：SAnti_jitter.v
  - 核模块符号文档：SAnti_jitter.sym

- 图表 7 U9

三、主要仪器设备

1. 计算机（Intel Core i5 以上，4GB 内存以上）系统
2. Sword 开发板
3. Xilinx ISE14.7 及以上开发工具

四、操作方法与实验步骤

4.1 设计要求

◎ **建立 CPU 调试、测试和应用环境**

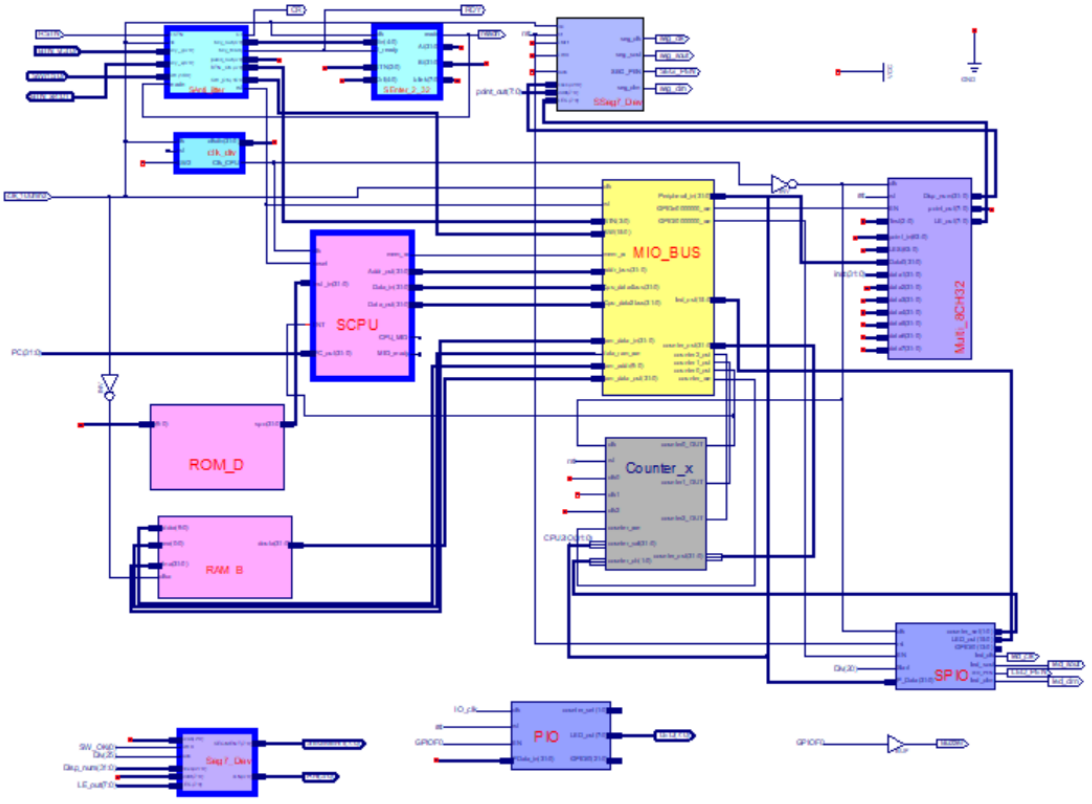
- 顶层用逻辑图实现，调用 IP 核模块
  - 模块名：Top_OExp03_IP2SOC.sch

◎ SOC 集成技术实现系统构架

- 用实验一、二设计的模块和第三方 IP 核
  - ⊙ CPU （第三方 IP 核）：U1
  - ⊙ ROM （ISE 构建核）：U2
  - ⊙ RAM （ISE 构建 IP 核）：U3
  - ⊙ 总线(第三方 IP 核)：U4
  - ⊙ 八数据通路模块(实验一 Multi_8CH32)：U5
  - ⊙ 七段显示模块(实验二 SSeg7_Dev IP)：U6
  - ⊙ LED 显示模块(实验二 SPIO 模块)：U7
  - ⊙ 通用分频模块(clk_div)：U8
  - ⊙ 开关去抖模块(IP 核)：U9
  - ⊙ 数据输入模块(IP 核)：M4(目前没有使用）

4.2 顶层模块再设计

本次实验在实验二的基础上，增加了 SCPU 和 MIO_BUS，主要修改的部分是顶层的逻辑图。现在逻辑图变得更加复杂，连线时也要出出小心，防止连错。



图表 8 顶层模块 sch 图

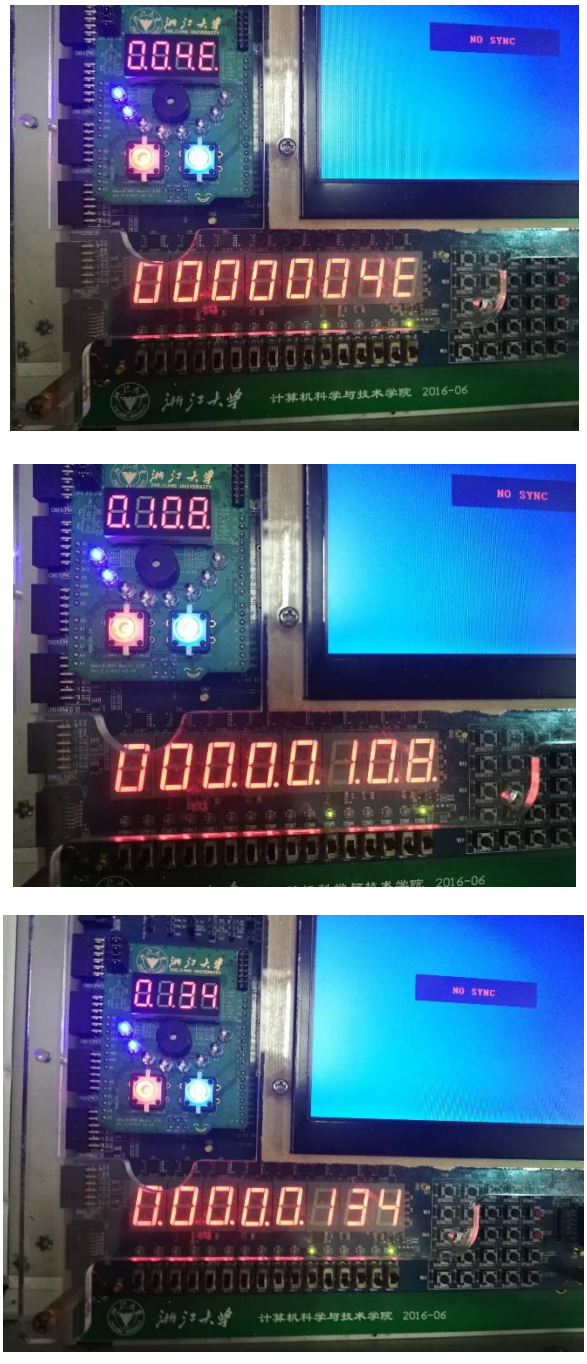
4.3 ROM 和 RAM 的重新载入

重新初始化 IP 核。

五、实验结果与分析

5.1 实验现象一

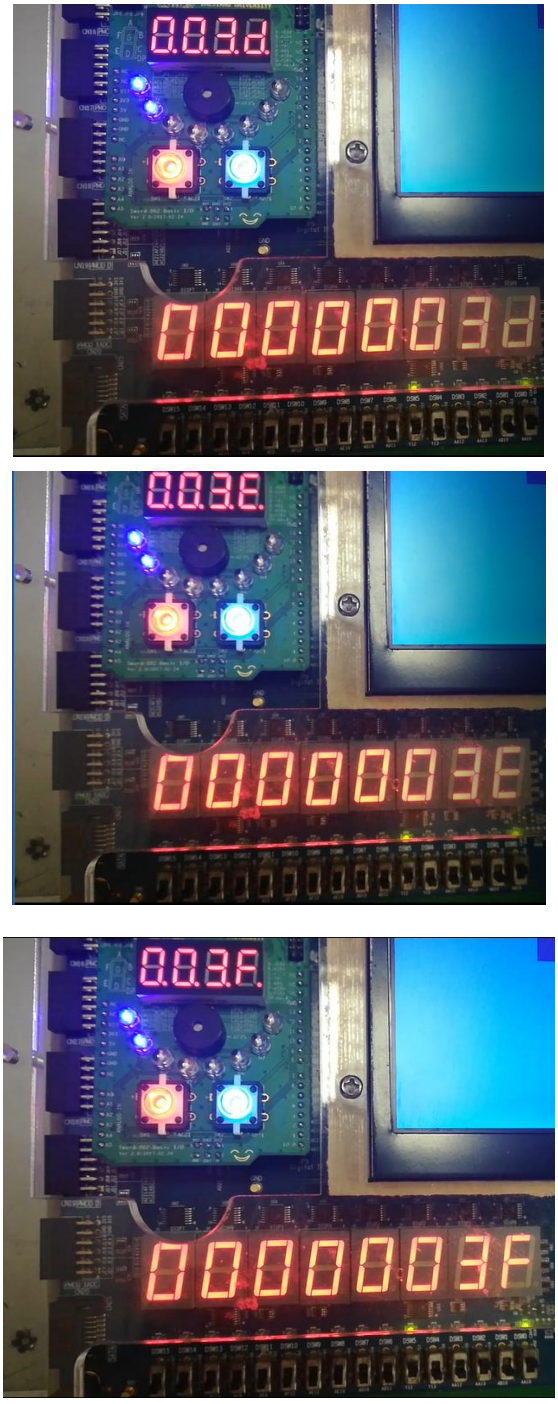
SW[0]=1, SW[2]=1，SW[7:5]=111。
输出 CPU 指令字节地址 PC_out。



图表 9 实验现象 1

5.2 实验现象二

SW[0]=1, SW[2]=1，SW[7:5]=001。
输出 CPU 指令字地址 PC_out[31:2]。



图表 10 实验现象 2

5.3 实验现象三

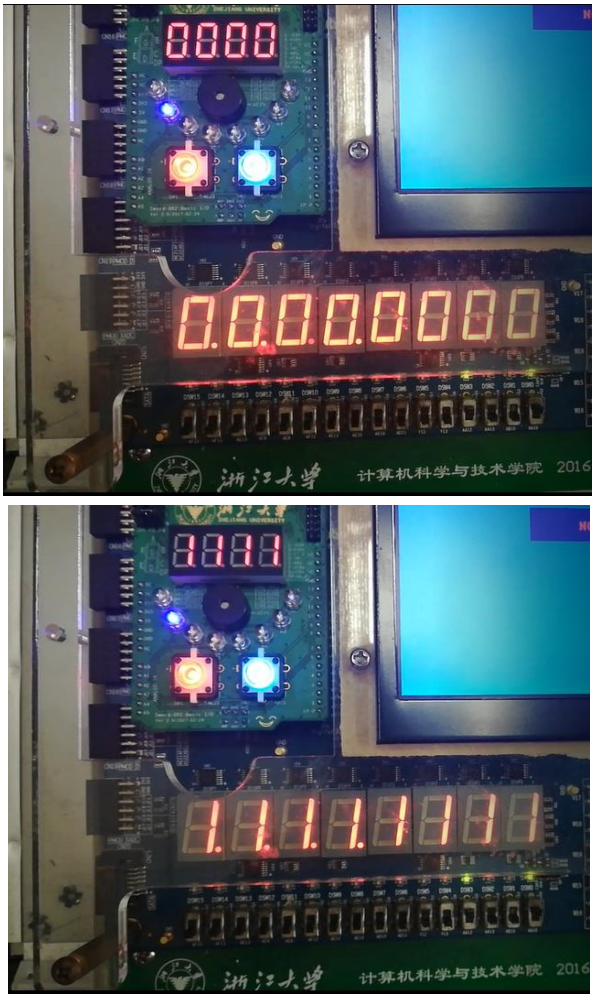
SW[0]=0, SW[3]=1，SW[4]=1。
矩形框跳舞。

图表 11 实验现象 3

5.4 实验现象四

SW[0]=1, SW[3]=1。

全速时钟。



图表 12 实验现象 4

六、讨论、心得

本次实验是计算机组成的第三个实验。

这次在第二个实验的基础上，增添了 CPU 成分。具体地，导入了 SCPU 和 MIO_BUS 两个部件（不过目前还是用老师封装好的 ngc），并重画了顶层构架。

有两次实验的基础，本次实验我操作飞快，进行得较为顺利，对硬件体系与 SOC 也有了进一步的认识。

值得注意的是，我刚做完后，板子上八个数字中，有一个数字会出现迷之错误。排查了半天也没排查出错误（其实只要检查一下八个接口下标的对称性即可），后来发现是板子的问题，换了个板子就好了。这几次实验大大地锻炼了我查错的能力。