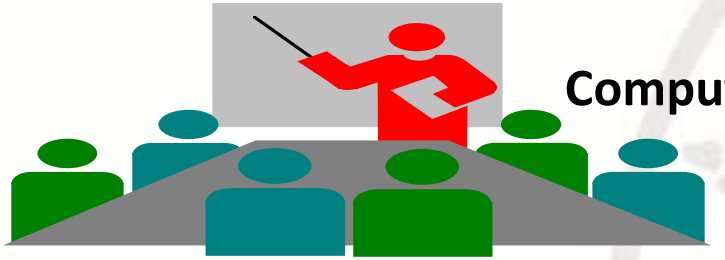


Computer Organization & Design
实验与课程设计

综合性课程设计

简易SOC或微控制器应用设计

施青松
Asso. Prof. Shi Qingsong
College of Computer Science and Technology, Zhejiang University
zjsqs@zju.edu.cn

CPU简单应用：简单SOC/微控制器

- 实验3～实验12的顶层测试环境已经是最简SOC
  - Project硬件工作只要实现存储空间译码即可
  - 重点工作是在自己实现的SOC上运行有意义的应用

基本要求

- **设计简单总线接口**
  - 设计基本GPIO总线接口
    - 支持低速设备
  - 接口功能支持
    - 主存储器
    - 输出：七段显示器、GPIO/LED
    - 输入：GPIO/SW、独立按键
    - 简单计数器
- **任选一款实验设计的CPU**
  - 构建SOC应用系统
    - 其余模块可以用核或自行设计
  - 设计有意义的应用程序
    - 四个简单外设支持的小游戏
      - 参考：手指跳舞毯

高级要求

- **任选或组合均可以**
  - 要在自己应用中体现：根据应用要求决定

1. 扩展阵列键盘输入总线接口
   - 增加阵列键盘输入支持
   - 阵列式键盘预处理模块可以用核或自行设计
2. 扩展VGA文本总线接口
   - 支持VGA 640*480，标准模式
   - 设计VGA显示接口驱动
3. 扩展PS2键盘总线接口
   - 支持PS2键盘
   - 设计PS2接口驱动

- **设计较高级的应用程序**
  - VGA-PS2交互应用
    - 参考：吃豆子、俄罗斯方块

提交内容：最后的机会-权重更高

- 提交内容
  - 课程设计报告，参考FTP课程设计格式模板：
  - 提交工程要求
    - 所有设计代码：.v和.sch文档和测试代码.asm
    - 可执行流文件：.bit文档
    - 最终工程文档：调试通过并清理干净工程压缩文档
      - 含工程文档说明
  - 附3~5分钟视频（清楚、层次分明）：

    1.自我介绍（10秒钟）；

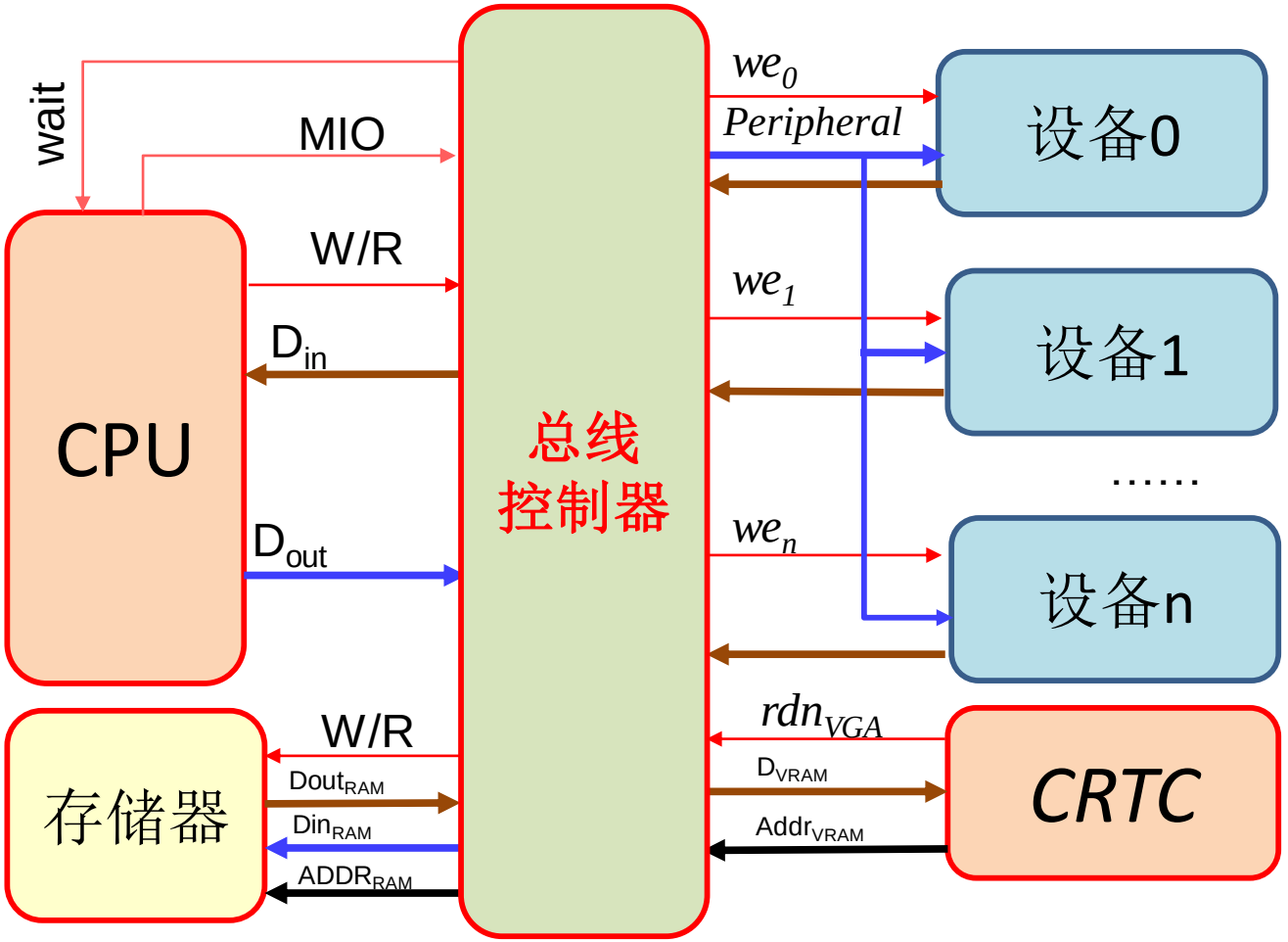
    2. 简要阐述设计方案(配PPT解说)和实现过程(调试主要步骤，1分钟)

    3. 测试操作演示
- 提交文档压缩：
  - 学号_姓名_CPU_APP.rar
- 提交时间
  - 2019年06月24日

单总线控制重点

- 总线主控器
  - VGA优先
  - CPU等待

最简单结构
退化为地址译码



总线接口地址分配

- **基本地址分配**
  - 存储器空间：　　0000000-……
  - GPIO/LED显示地址(写)：　　F0000000/FFFFFF00
  - GPIO/Switch、BTN地址(读)：　　F0000000/FFFFFF00
  - 7段显示器地址(写)：　　E0000000/FFFFFE00
  - 硬件计数器地址*：　　F0000004/FFFFFF04
- **扩展地址分配**
  - 阵列式键盘地址：　　F0000008/FFFFFF08
  - PS2键盘：　　FFFFFD00
  - VRAM地址：　　000C0000H~
    - VGA：640*480，标准字符：8*8

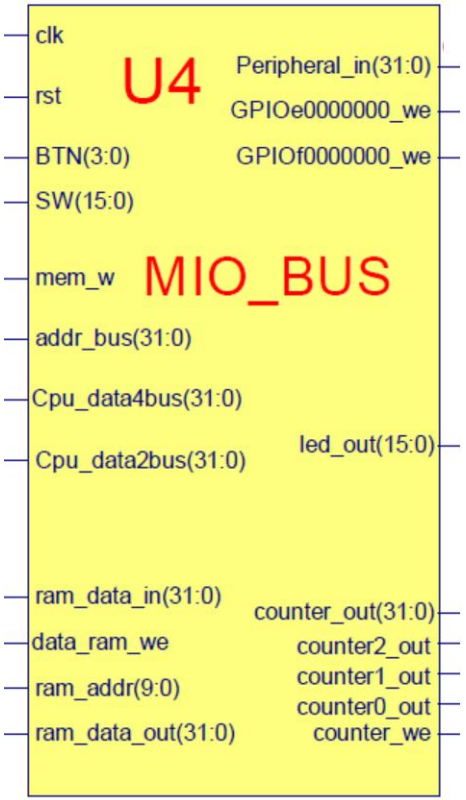
存储与外设地址接口译码结构

- **简单译码电路(接口课程知识点)**
  - 用HDL直接描述非常简单
- **译码电路描述结构**

```
always @* begin
    data_ram_we = 0;              //缺省信号赋值
        ……
    case(addr_bus[31:28])         //译码地址仅高4位，重复地址很多，但简单
      4‘h0: begin     // 内存 (00000000 - bffffffc, actually lower 4KB RAM)
                      ……
            end
      4‘he: begin     // 设备1(e0000000 - efffffff,)
                      ……
            end
      4'hf: begin     // 设备2(f0000000 - ffffffff,)
                      ……
            end
                      ……
        endcase
end
```

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
    - 就是一个地址译码电路
    - 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
    - 控制CPU与外部数据交换
- **实验3开始使用的IP 软核- U4**
  - 本课程设计要求自己设计替换MIO_BUS_IO.v
  - 模块符号文档：MIO_BUS.sym



IO总线接口模块端口信号-MIO_BUS.v

```
module MIO_BUS(  input wire clk, input wire rst,
                 input wire [3:0] BTN,  input wire  [15:0]SW,
                 input wire mem_w,
                 input wire [31:0] Cpu_data2bus,          //data from CPU
                 input wire [31:0] addr_bus,              //addr  from CPU
                 input wire [31:0] ram_data_out,
                 input wire [15:0]  led_out,
                 input wire [31:0] counter_out,
                 input wire counter0_out,
                 input wire counter1_out,
                 input wire counter2_out,

                  output wire [31:0] Cpu_data4bus,        //write to CPU
                  output wire [31:0]  ram_data_in,        //from CPU write to Memory
                  output wire [9:  0]  ram_addr,           //Memory Address signals
                  output wire data_ram_we,
                  output wire GPIOf0000000_w,             // GPIOffffff00_we
                  output wire GPIOe0000000_we,            // GPIOfffffe00_we
                  output wire counter_we,                 //记数器
                  output wire [31:0] Peripheral_in        //送外部设备总线
                 );
endmodule
```

MIO_BUS描述参考

```
module MIO_BUS(端口信号描述);
 reg data_ram_rd, GPIOf0000000_rd, GPIOe0000000_rd, counter_rd;
wire counter_over;                    //变量定义
.............
//RAM & IO decode signals:
    always @* begin
        data_ram_we = 0;                          //主存写信号
        data_ram_rd  = 0;                         //主存读信号
        counter_we    = 0;                        //计数器写信号
        counter_rd    = 0;                        //计数器读信号
        GPIOf0000000_we  = 0;                     //设备1：PIO写信号
        GPIOe0000000_we = 0;                      //计数器：Counter_x写信号
        GPIOf0000000_rd   = 0;                    //设备3、4：SW等读信号
        GPIOe0000000_rd   = 0;                    //设备2：七段显示器写信号
        ram_addr = 10'h0;                         //内存物理地址：RAM_B地址
        ram_data_in = 32'h0;                      //内存读数据：RAM_B输出数据
        Peripheral_in=32'h0;                      //外设总线：CPU输出，外设写入数据
        case(addr_bus[31:28])                     //开始译码
        4'h0: begin                  // data_ram (00000000 - 00000ffc, actually lower 4KB RAM)
              data_ram_we = mem_w;
              ram_addr = addr_bus[11:2];
              ram_data_in = Cpu_data2bus;
              data_ram_rd = ~mem_w;
            ..........
```

Cpu读译码省略，Why？

MIO_BUS描述参考-2

```
4'he: begin                          // 七段显示器  (e0000000 - effffffff, SSeg7_Dev)
GPIOe0000000_we = mem_w;
Peripheral_in = Cpu_data2bus;
GPIOe0000000_rd = ~mem_w;
end
4'hf: begin                          // PIO   (f0000000 - fffffff0, 8 LEDs
                                                    & counter, f000004-fffffff4)
  if(addr_bus[2]) begin                          //f0000004
     counter_we = mem_w;
     Peripheral_in = Cpu_data2bus;               //write Counter Value
     counter_rd = ~mem_w;
  end
  else begin                         //f0000000
  GPIOf0000000_we = mem_w;
  Peripheral_in =Cpu_data2bus;    //write Counter set & Initialization and light LED
  GPIOf0000000_rd = ~mem_w;
  end
  end
  ..................
  endcase
end
```

为什么要*if*设计，请改进

输入设备3、4描述

endmodule

设备三、四

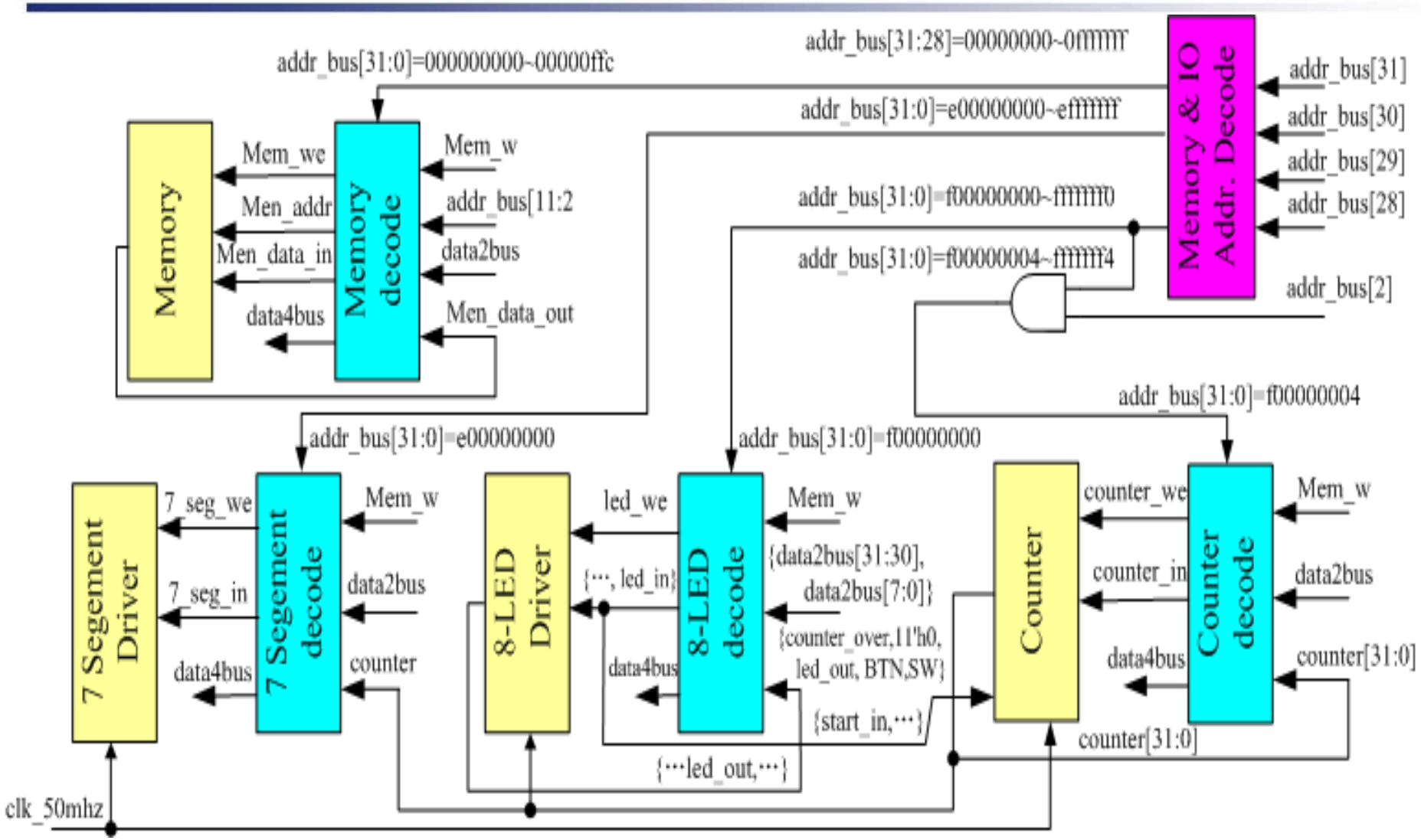
- 基本要求的输入设备
  - 非常简单，直接在接口模块内描述
- 存储器读及输入设备描述

```
always @* begin                          //Data to CPU
    Cpu_data4bus = 32'h0;
    casex({data_ram_rd, GPIOe0000000_rd,counter_rd, GPIOf0000000_rd})
    4'b1xxx: Cpu_data4bus = ram_data_out;                  //read from RAM
    4'bx1xx: Cpu_data4bus = counter_out;                   //read from Counter
    4'bxx1x: Cpu_data4bus = counter_out;                   //read from Counter
    4'bxxx1: Cpu_data4bus = {counter0_out,counter1_out,counter2_out,
                             led_out[12:0], SW};           //read from SW & BTN
    endcase
    end
```

SP3的二个输入设备太简单了，合用一个地址

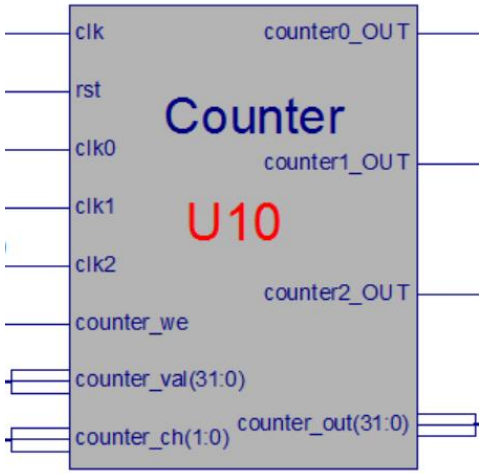
Sword平台资源丰富，可以将阵列式键盘加入需要分开接口等后继课程将深入讨论

对应接口逻辑图：比代码复杂多，但直观



计数器设备

- **计数器是非常有用的**
  - 没有计数器程序轮循非常不方便
  - 没有计数器操作系统进程切换实现困难
- **Intel PC计算机系统的计数器**
  - 采用8253，目前的PC还与它兼容
    - 设计的非常复杂(接口课程将讨论)
- **本系统采用简单的计数器**
  - Counter部分兼容8253，还是复杂
  - 基本计数器功能：简化Counter_x
    - 32位计数：参考clk_div.v
      - 即时计数值读取：counter_out，可以定时用
    - 初始化值Counter_val：CPU输出
      - 相当于并行置入：counter_we=1信号控制写入计数器
    - 计数结束信号，简化为一个：Counter0_OUT
    - 计数时钟，简化为一个：clk0



计数器参考描述

```
Module    Counter(input clk,                    //同步时钟
                  input rst,                    //复位
                  input clk0,                   //计数时钟
                  input counter_we,             //初始化写信号
                  input [31:0] counter_val,     //初始化输入
                  output counter0_OUT,          //计数结束信号
                  output [31:0] counter_out);//计数器值输出

reg [32:0] counter0, counter1,counter2;         //简化设计，省略某些信号
reg [31:0] counter0_Lock,counter1_Lock,counter2_Lock;
reg [23:0] counter_Ctrl;
reg sq0,sq1,sq2,M0,M1,M2,clr0,clr1,clr2;
//Counter read or write & set  counter_ch=SC1 SC0; counter_Ctrl=XX M2 M1 M0 X
```

简化设计需要修改核的逻辑符号

计数器参考描述-1

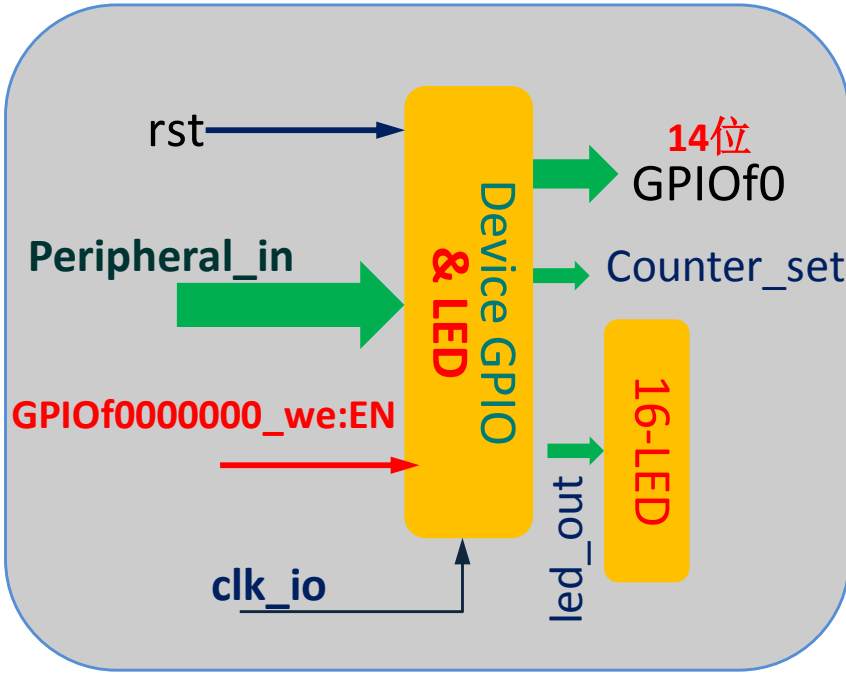
```
always @ (posedge clk or posedge rst) begin
    if (rst ) counter0_Lock;
    else if (counter_we) begin                    //初始化写初值：f0000000
            counter0_Lock <= counter_val;
            M0 <= 1;                              //设置初始化时常数标志
         end
         else begin counter0_Lock <= counter0_Lock;
                  if(clr0) M0 <= 0;               //清初始化时常数标志
         end
end

always @ (posedge clk0 or posedge rst) begin // Counter channel 0 工作
    if (rst ) counter0 <= 0;
     else begin if (M0) begin counter0 <= {1'b0,counter0_Lock}; clr0<=1; end
                else if(counter0[32] == 0)begin
                        counter0 <= counter0 - 1'b1; clr0<=0;  end //已经置初值
end
assign counter0_OUT = counter0[32];

endmodule
```

U7-外部设备模块：GPIO接口及设备一SPIO

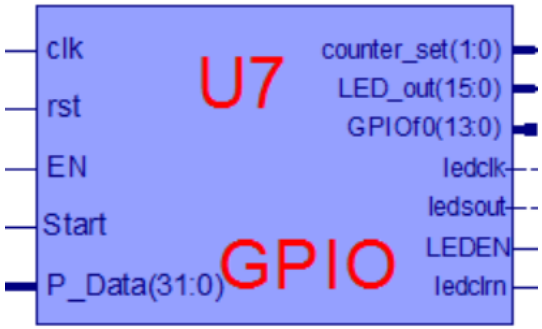
- GPIO输出设备一
  - 地址范围=f0000000 - ffffffff0 (ffffff00-ffffffff0)
  - 读写控制信号：**GPIOf0000000_we(GPIOffffff00_we)**
  - {GPIOf0[13:0],LED,counter_set}

- 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

- 本实验用实验一模块：**U7**
  - 调用模块GPIO.v
  - 模块符号文档：GPIO.sym



通用接口设备—接口信号：GPIO.v



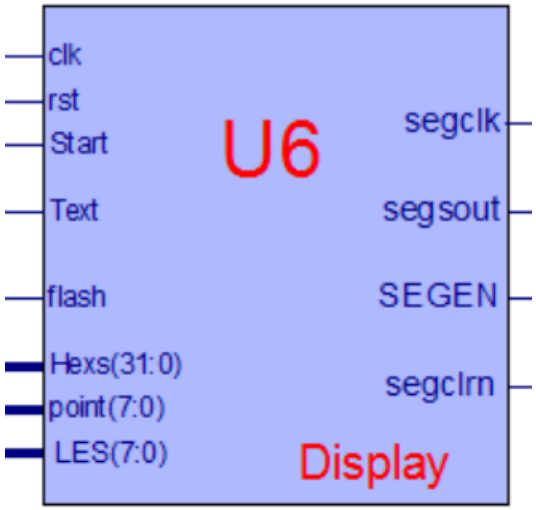

```
module      GPIO(input clk,                    //io_clk，与CPU反向
                 input rst,
                 input EN,                     //来自U4
                 input [31:0] P_Data,          //来自U4
                 input Start                   //串行输出启动
                 output [1:0] counter_set,     //来自U7，后继用
                 output [15:0] LED_out,        //输出到LED,回读到U4
                 output [23:0] GPIOf0          //备用
                 output ledclk,                //串行时钟
                 output ledsout,               //串行LEDE值
                 output LEDEN,                 //LED使能
                 output ledclrn                //LED清零
                 );

endmodule
```

U6-外部设备模块：GPIO设备二

SSeg7_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;
    - SW[1:0]=11，显示高4位
  - 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Exp02设计的模块- **U5**
  - 调用模块SSeg7_Dev.v
  - 模块符号文档：SSeg7_Dev.sym

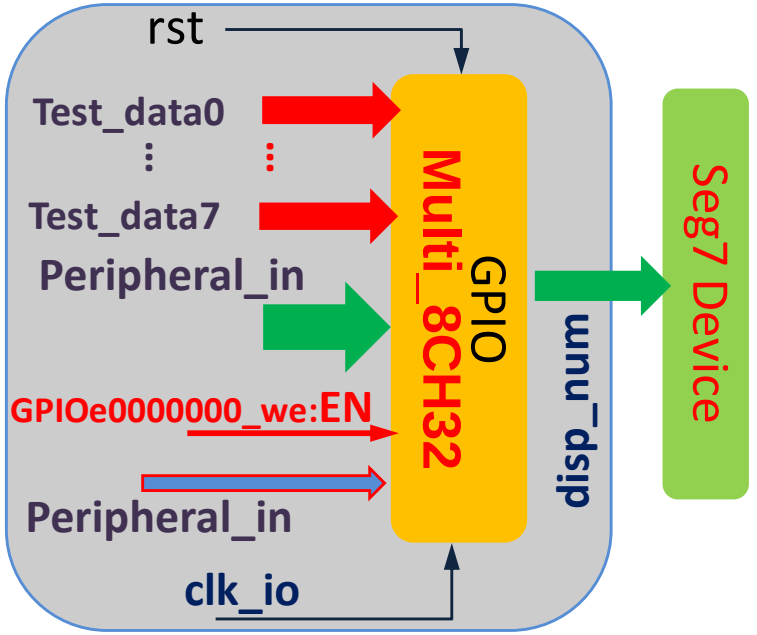
通用设备二接口信号： Display.v



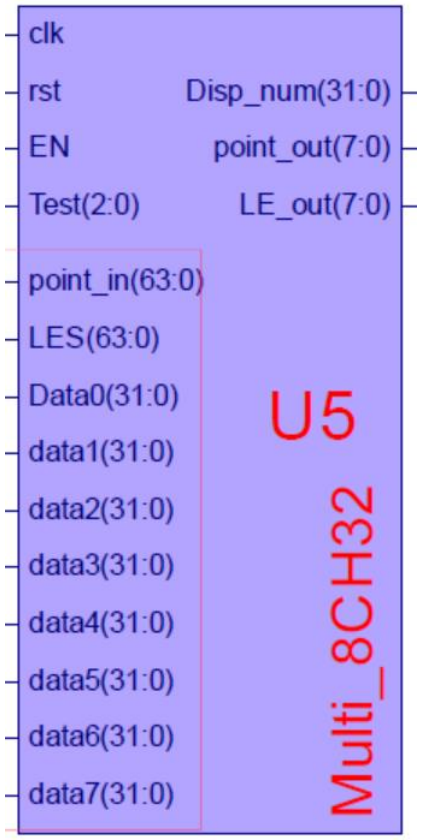

```
module   Display(input clk,              //时钟
                 input rst,              //复位
                 input Start,            //串行扫描启动
                 input Text,             //文本(16进制)/图型(点阵)切换
                 input flash,            //七段码闪烁频率
                 input[31:0]Hexs,        //32位待显示输入数据：来自U5
                 input[7:0]point,        //七段码小数点：8个
                 input[7:0]LES,          //七段码使能：=1时闪烁
                 output segclk,          //串行移位时钟
                 output segsout,         //七段显示数据(串行输出)
                 output SEGEN,           //七段码显示刷新使能
                 output segclrn          //七段码显示清零
                 );

endmodule
```

U5-通用设备二接口模块： Multi_8CH32

- **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
- **用EXp02设计的模块- U6**
  - 调用模块Multi_8CH32.v
  - 核模块符号文档：Multi_8CH32.sym



通用设备二接口模块端口信号 -Multi_8CH32.v




```
module Multi_8CH32 ( input clk,                          //io_clk，同步CPU
                     input rst,
                     input EN,                           //=1, 通道0显示
                     input[63:0]point_in, //针对8个显示通道各8个小数点
                     input[63:0]blink_in,    //针对8个通道各8位闪烁控制
                     input [2:0] Test,                   //通道选择SW[7:5]
                     input [31:0] Data0,                 //通道0
                     input [31:0] data0,                 //通道1
                     input [31:0] data1,                 //通道2
                     input [31:0] data2,                 //通道3
                     input [31:0]  data3,                //通道4
                     input [31:0]  data4,                //通道5
                     input [31:0]  data5,                //通道6
                     input [31:0]  data6,                //通道7

                     output reg[3:0] point_out,          //小数点输出
                     output reg[3:0] blink_out,          //闪烁控制输出
                     output [31:0]    disp_num           //接入7段显示器
                   );

endmodule
```

Multi_8CH32调用信号关系

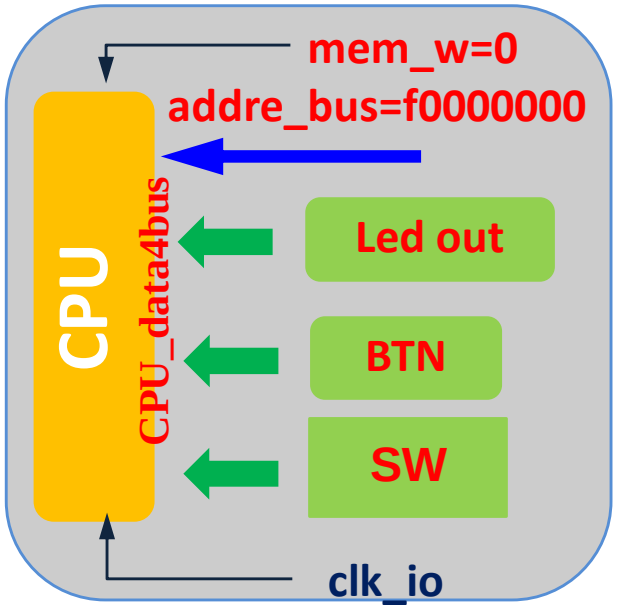
```
Multi_8CH32    U5( .clk(clk_io),   .rst(rst),
                   . EN(GPIOe0000000_we),          //来自U4
                   . point_in(point_in),           //外部输入
                    .blink_in(blink_in),           //外部输入
                   .Test(SW_OK[7:5]),              //来自U9
                   .Data0(Peripheral_in),          //来自U4
                   .data1({2‘b00,PC_out[31:2]}),   //来自U1
                   .data2(counter_out),            //来自U10
                   .data3(Inst),                   //Inst，来自CPU
                   .data4(addr_bus),               //来自CPU
                   .data5(Cpu_data2bus),           //来自CPU
                   .data6(Cpu_data4bus),           //来自CPU
                   .data7(PC_out),                 //来自CPU;

                   .point_out(point_out),          //输出到U6
                   .blink_out(blink_out),          //输出到U6
                   .disp_num(disp_num)             //输出到U6
                   );
```

外部设备模块：通用接口设备三、四

GPIO_SW_BTN

- 太简单直接设计在MIO_BUS模块中
- 输入16位Switch和5位Button
  - addre=f0000000-fffffff0。A[2]=0
  - 直接在译码电路中输入CPU：

```
Cpu_data4bus = {counter0_out,counter1_out,counter2_out,
                led_out[12:0], SW};
```

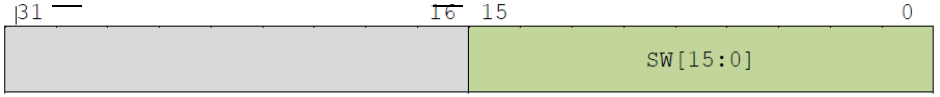


地址分配约定(协议)：与后继设计兼容

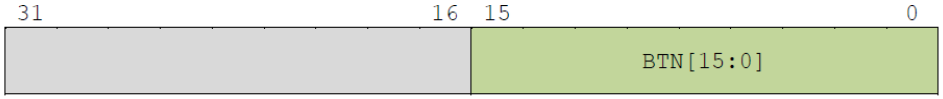
- 总线模块优化设计
- 存储与IO地址空间分配与约定
  - 内存空间
    - **RAM空间4080M：0x0000_0000～0xFEFF_FFFF**
    - ROM空间15.9M：0xFF00_0000～0xFFFE_FFFF
  - IO空间：FFFF_0000共64K空间,分XXYY
    - 设备号：XX=256个
    - 设备内端口：YY=256个
  - 0号设备空间为保留
    - 对应的地址空间为0xFFFF_0000～0xFFFF_00FF

板级IO设备协议

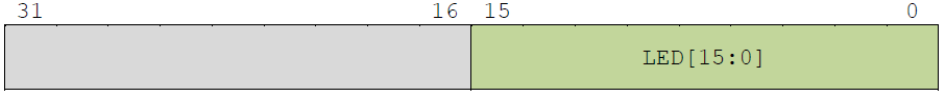
- 设备号0x02
  - 地址：0xFFFF_0200～0xFFFF_02FF
- FPGA平台的基本输入输出接口
  - Button+Switch+LED+7-Segment
- 端口分配
  - 开关Switch(读)：0xFFFF_0200～0xFFFF_0203



  - 按键/钮(读)：0xFFFF_0204～0xFFFF_0207

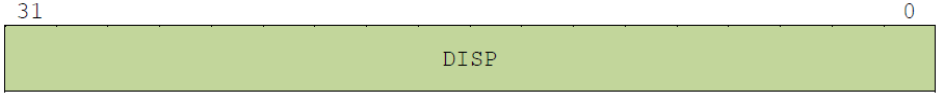


  - LED指示灯(写)：0xFFFF_0210～0xFFFF_0213

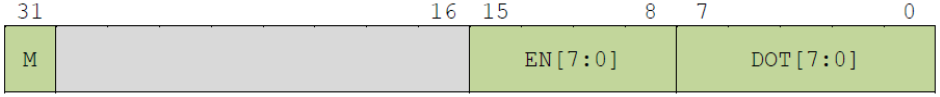


  - 七段码显示(写)：

0xFFFF_0218～0xFFFF_021B



0xFFFF_021C～0xFFFF_021F

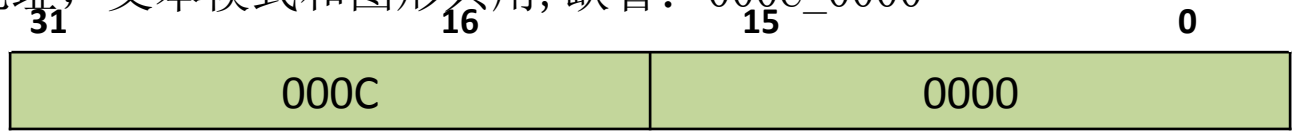


计算机标准设备-VGA

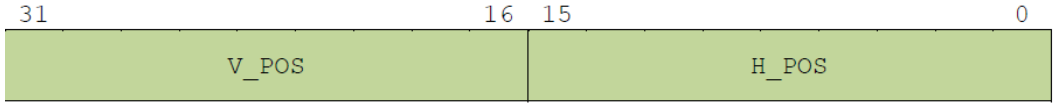
- VGA设备
  - 设备号：0x01
  - 地址：0xFFFF_0100～0xFFFF_01FF
- 端口分配
  - 模式控制(写)：0xFFFF_0100～0xFFFF_0103

M：模式位，置 0为文本模式，置 1为图形模式
C：硬件光标使能位，置1时开启硬件光标　Font：字库模式
R：分辨率选择，0表示关闭 VGA 输出：见下页

  - 显存起始地址(写)：0xFFFF_0104～0xFFFF_0107
    - VRAM起始地址，文本模式和图形共用,缺省：000C_0000



  - 硬件光标地址(写)：0xFFFF_0108～0xFFFF_010B



VGA显示模式约定

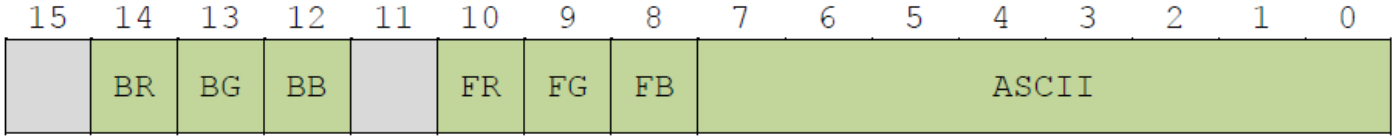
◎VGA模式

模式	分辨率@刷新率	长宽比	标准类型
1	640 * 480 @ 60Hz	4:3	工业标准(启动)
2	640 * 480 @ 72Hz	4:3	VESA标准
3	640 * 480 @ 75Hz	4:3	VESA标准
4	800 * 600 @60Hz	4:3	VESA指导建议
5	800 * 600 @ 72Hz	4:3	VESA标准
6	800 * 600 @ 75Hz	4:3	VESA标准
7	1024 * 768 @ 60Hz	4:3	VESA指导建议

◎字库模式

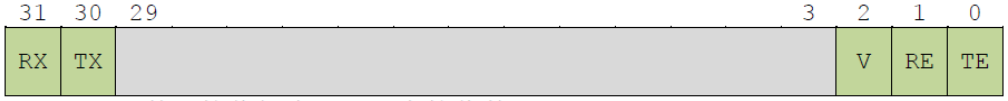
Font：000=英文8*8(标准/启动)　001=英文8*16

010=中文16*16　011=



计算机标准设备-PS2

- 键盘设备
  - 设备号：0x03
  - 地址：0xFFFF_0300～0xFFFF_03FF
  - 端口分配
    - 读键盘状态，用于键盘状态查询：0xFFFF_0300～0xFFFF_0303



RX：数据接收标志，当正在接收数据时置位。
TX：数据发送标志，当正在发送数据时置位。
V：数据有效标志，当已从键盘接收到数据时置位，当CPU将数据读走时复位
RE：读取出错，当从键盘读取数据发生异常时置位，当CPU尝试读取键盘数据时复位。
TE：写入出错，当向键盘写入数据发生异常时置位，当CPU尝试向键盘写

- 读键盘数据： 0xFFFF_030C～0xFFFF_030F


END