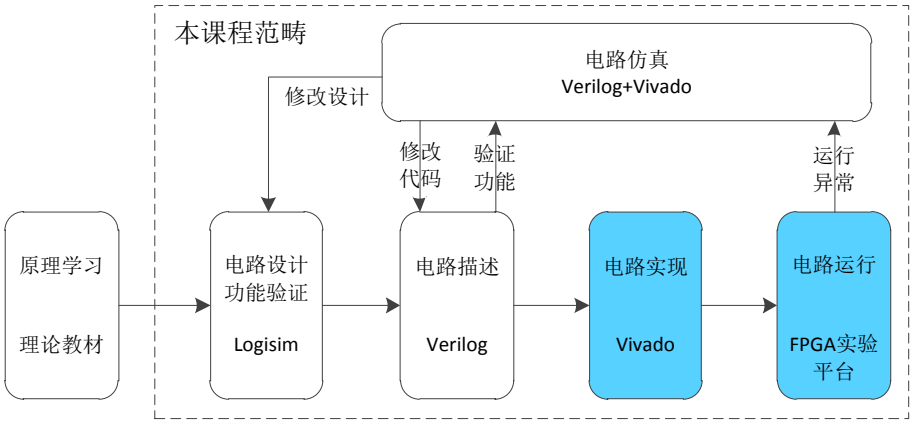
实验 07 FPGA 实验平台及 IP 核使用

简介



前面的实验中，我们完成了一个简单的电路设计，并顺利的烧写到 FPGA 实验平台上。但可能会有部分读者对于其中的一些步骤并不了解，本次实验中，我们将对所使用的 FPGA 实验平台进行介绍，以帮助读者加深对 FPGA 开发流程各环节的理解。此外，我们还会介绍到如何使用 IP 核进行电路设计。

实验目的

熟悉 FPGAOL 在线实验平台结构及使用

掌握 FPGA 开发各关键环节

学会使用 IP 核（知识产权核）

实验环境

VLAB 平台：vlab.ustc.edu.cn

FPGAOL 平台：fpgaol.ustc.edu.cn

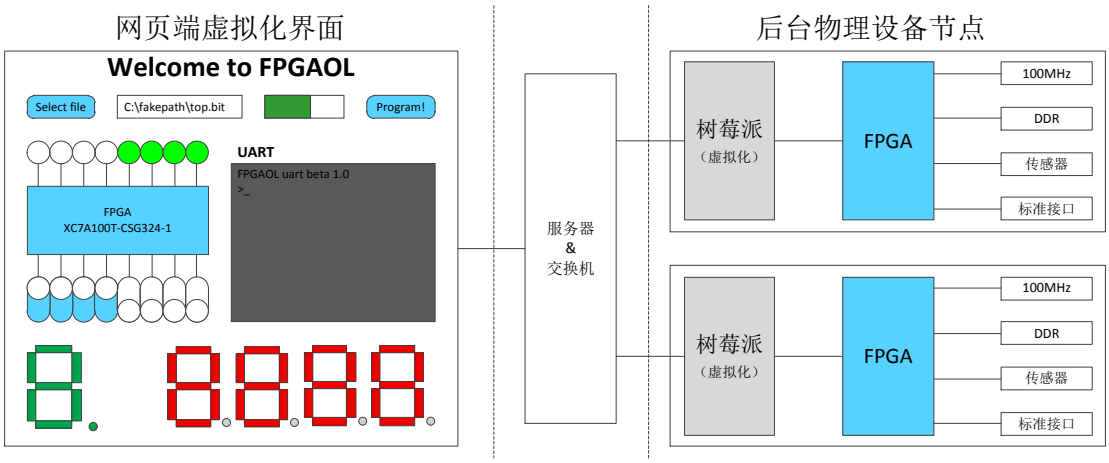
Vivado

Logisim

实验步骤

Step1. FPGAOL 实验平台介绍

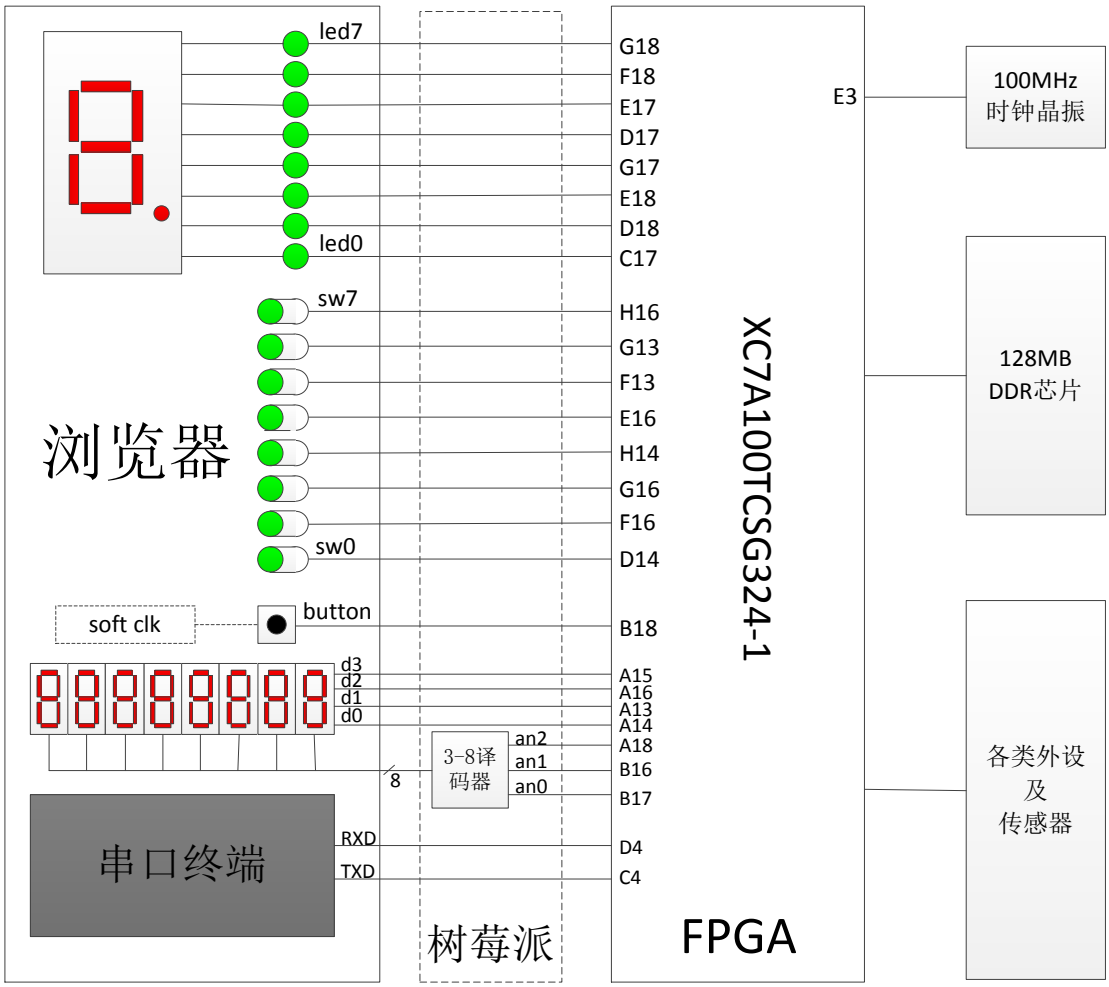
FPGA online（简称 FPGAOL）是中国科学技术大学计算机学院实验中心自主开发的一套在线实验平台，该平台通过分时复用的方式为用户提供 FPGA 设备节点的在线服务，目前 FPGAOL 一代系统已正式对外提供服务，用户可通过网址 fpgaol.ustc.edu.cn 访问平台，并使用统一身份认证登陆。



FPGAOL 平台的每个设备节点都包含了一个树莓派和一个 FPGA 板卡，树莓派与 FPGA 芯片的 26 个 IO 管脚直接相连，并最终映射到网页端，其中包括 8 个对应 LED（同时与七段数码管复用），8 个对应开关，7 个对应 8 位 16 进制数码管，1 个对应按键，2 个对应串口 UART。用户在网页端对虚拟化外设进行的操作都会通过树莓派映射到对应的 FPGA 管脚上。同理，FPGA 管脚的电平变化也会通过树莓派反应到网页端的虚拟化外设上。因此，用户是通过网页端的虚拟化外设实际地控制物理设备节点。

除了网页端的虚拟化外设，FPGA 板卡上还带有板载 100MHz 时钟，128MB DDR 内存芯片，以及各类传感器及接口外设，如温度传感器、

麦克风等，具体管脚对应关系可参考下图或网站提供的 XDC 文件。



Step2. 使用时钟管理单元 IP 核

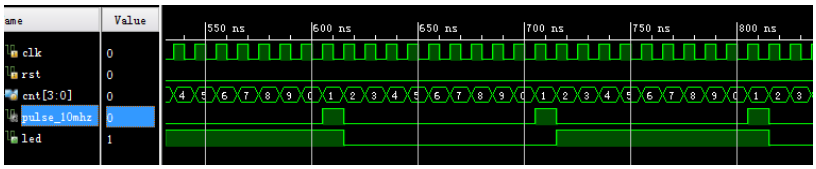
在 FPGA 开发中，有很多常用功能的模块是不需要自己开发的，用户可以复用第三方开发好的模块，这种模块被称为 IP 核。此处我们先学习时钟管理单元 IP 核的使用。

前面已经讲到，FPGA 开发板的 FPGA 芯片 E3 管脚连接了一个 100MHz 频率的时钟晶振，可用作时序逻辑电路的时钟信号。如果我们需要一个其它频率的时钟信号，例如 10MHz，应该怎么办呢？一般的做法是通过计数器产生一个低频的脉冲信号，然后再将该脉冲信号控制其他逻辑的控制信号，如下代码所示，通过 pulse_10mhz 信号控

制 led 信号。

```
module ttt(
input  clk,rst,
output reg led);
reg [3:0]   cnt;
wire        pulse_10mhz;
always@(posedge clk)
begin
    if(rst)
        cnt <= 4'h0;
    else if(cnt>=9)
        cnt <= 4'h0;
    else
        cnt <= cnt + 4'h1;
end
assign pulse_10mhz = (cnt == 4'h1);
always@(posedge clk)
begin
    if(rst)
        led <= 1'b0;
    else if(pulse_10mhz)
        led <= ~led;
end
endmodule
```

其仿真波形如下图所示



有些读者喜欢将分频信号直接作为时钟使用，虽然这种设计也能工作，但我们强烈反对这种实现方式，因为这样做会产生许多无谓的警告，也容易引起电路工作的不稳定。希望读者时刻牢记，在实际的数字电路中（仿真代码除外），时钟信号是非常特殊的一种信号，它不应该出现在过程语句和连续赋值语句内部，如下所示的代码都是不推荐的（clk 为时钟信号 signal_x 为其它普通信号）

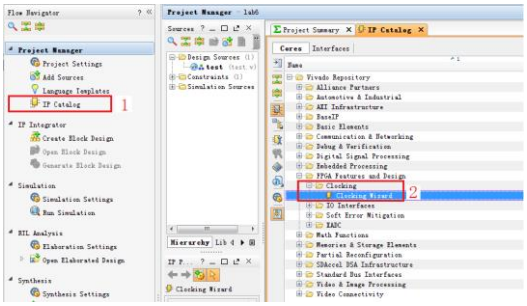
```
错误示例：assign    signal_1   = func(clk,xxx);
```

错误示例：always@(posedge signal_2) //signal_2 由 clk 生成
错误示例：always@(posedge clk or negedge clk) //两边沿不能同时使用

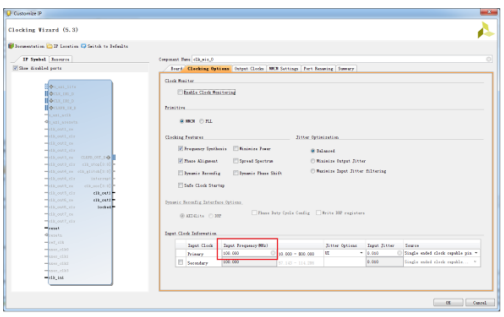
时钟信号只应该出现在 always 语句的时序控制部分

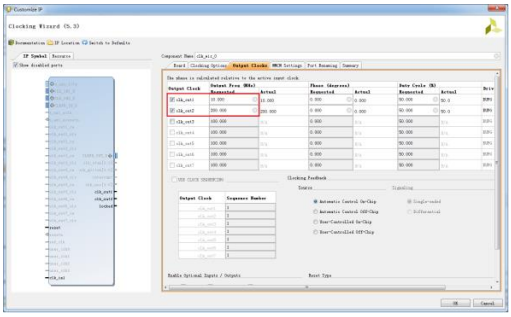
正确示例：always@(posedge clk) //时钟上升沿触发，同步复位或无复位
正确示例：always@(negedge clk) //时钟下降沿触发，同步复位或无复位
正确示例：always@(posedge clk or posedge rst) //异步复位，高有效
正确示例：always@(posedge clk or negedge rst_n) //异步复位，低有效

如果设计中确实需要不同频率的时钟信号应该通过时钟管理单元 IP 核生成。首先，点击“IP catalog”，在对应窗口中选中“Clocking Wizard”并双击。



在弹出的窗口中，对其进行设置，输入时钟频率设置为 100MHz，输出时钟有两个，分别为 10MHz 和 200MHz。





设置完成后点击确认按钮，生成 IP 核。生成的 IP 文件可在“工程目录/工程名.srcs/sources_1/ip/IP 核名称/IP 核名称.v”找到。用户可在设计文件中像调用其它模块一样使用该 IP 核,使用时只需要了解 IP 核的功能及端口信号的含义及时序，而不用关心模块内部的具体实现。如下代码所示，分别同 10M 和 200M 的时钟作为计数器

的计数时钟，通过仿真和烧写 FPGA，可以发现两个时钟频率的明显差异。

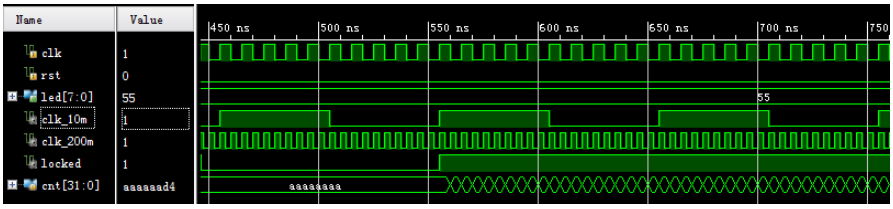
```
module test(
input                clk,
input                rst,
output      [7:0]    led);
wire        clk_10m,clk_200m,locked;
reg  [31:0] cnt_1,cnt_2;
always@(posedge clk_200m)
begin
    if(~locked)
        cnt_1 <= 32'hAAAA_AAAA;
    else
        cnt_1 <= cnt_1+1'b1;;
end
always@(posedge clk_10m)
begin
    if(~locked)
        cnt_2 <= 32'hAAAA_AAAA;
    else
        cnt_2 <= cnt_2+1'b1;;
end
assign  led = {cnt_1[27:24],cnt_2[27:24]};
clk_wiz_0   clk_wiz_0_inst(
.clk_in1    (clk),
.clk_out1   (clk_10m),
.clk_out2   (clk_200m),
.reset      (rst),
.locked     (locked));
endmodule
```

强烈建议读者在综合烧写之前先进行功能仿真，以确保电路功能的正确性，下面是一段最简单的仿真测试文件。

```
`timescale 1ns / 1ps
module tb(  );
reg clk,rst;
initial
begin
    clk = 0;
    forever
    #5 clk = ~clk;
```

```
end
initial
begin
    rst = 1;
    #100 rst = 0;
end
test    test(
.clk    (clk),
.rst    (rst),
.led    ( ));
endmodule
```

其仿真波形如下所示：



对于 FPGA 开发板，我们将 LED 信号分配到 8 个 LED 灯上，其 XDC 文件如下：

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { rst }];
## leds
set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
```

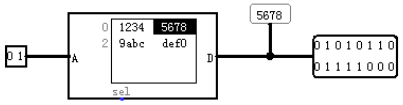
需要特别指出的是，时钟管理单元并不能产生任意频率的时钟信号，以前面为例，输入时钟为 100MHz 时，输出时钟只能是频率范围在 4.687MHz~800MHz 之间的某些频率，如 4MHz 超出范围无法产生，799MHz 虽然在频率范围内，但因精度问题，只能产生以及频率相近的时钟信号，有兴趣的读者可自行验证。

如果电路中需要一个较低频率的时序，例如我们需要一个每秒钟

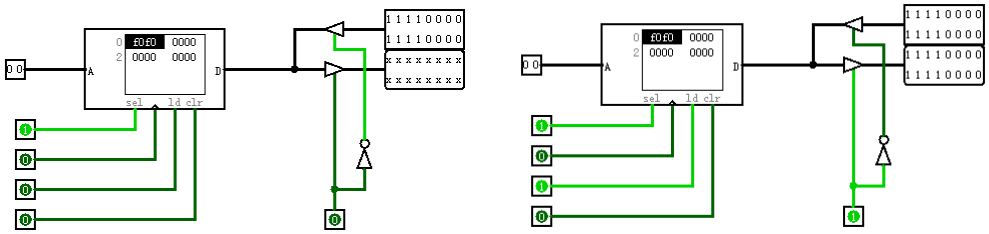
加一的计数器，那我们就只能通过前面介绍的方法，用周期为 1 秒的脉冲信号来控制了。

Step5. 使用片内存储单元

在前面的实验中，我们已经对存储器的行为特性进行了简单的介绍，例如 ROM 为只读存储器，包含了地址和数据两组端口，ROM 内部包含 $2^{地址位宽}$个存储单元，每个单元里存储了与数据端口相同位宽的数据，存储器电路会将与地址相对应单元的数据呈现在数据端口上，下图是一个包含 2^2=4 个存储单元的 ROM，数据位宽为 16 位。

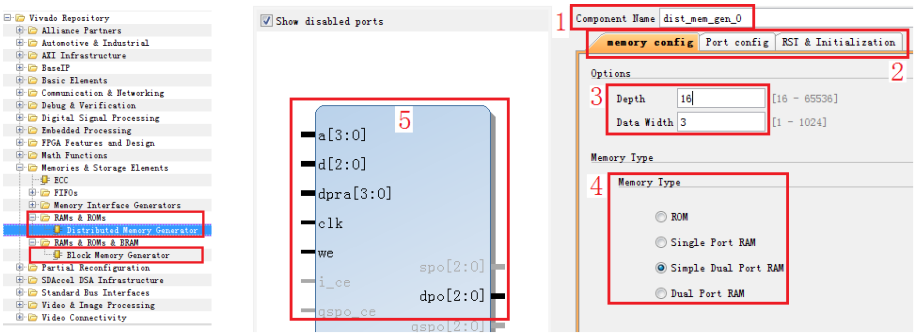


我们可以从 ROM 读取数据，但无法通过端口修改其内容，与此相对应的是 RAM（随机存储器），其内容可读可写,下图为 Logisim 中的 RAM 模型，包含地址（A）、数据（D，输入输入复用）、片选（sel）、时钟（clk，数据可在时钟上升沿写入）、输出使能（ld，为 1 时数据端口为输出，否则为输入）、清空（clr）等端口信号。这种 RAM 包含一套读写端口，因此成为单端口 RAM。读者可在 Logisim 中实际体验各端口信号的功能。



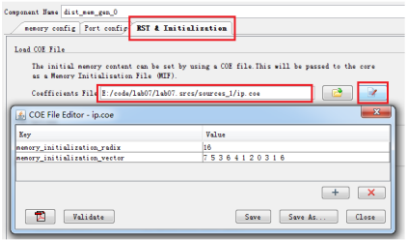
Vivado 中也提供了存储器相关的 IP 核，种类和接口类型比 Logisim 中更加丰富，我们通过实际的例子来学习一下。IP 核目录中提供了两种存储器实现方式:“Distributed Memory”和“Block Memory”

两种，我们以“Distributed Memory”为例进行介绍。



在存储器参数设置页面，用户可以对 IP 核名称、存储单元深度和位宽、存储器类型等进行设置。在页面的左侧是模块的端口图。在上图中，存储器深度设为 16=2^4，数据位宽为 3 位，因此地址信号为 4 位，数据端口位宽为 3。存储器类型选为简单双端口，因此包含了两套端口，其中 dpra、dpo 构成了读端口，d、a、clk、we 构成了写端口。在“RST&Initialization”页面，我们还可以通过后缀为 coe 的文件对存储器进行初始化，如下图右侧所示，coe 文件格式非常简单，可以以文本方式打开和编辑。其中“memory_initialization_radix=16”表示 coe 文件采用的是 16 进制方式显示，“memory_initialization_vector = ...”表示的是用 16 进制显示的初始化向量，下图右侧的示例表示，该存储器地址 0 内的数据为“23f4”，地址 1 内的数据为“0721”，以此类推。本例中的初始化文件内容为：

```
memory_initialization_radix=16;
memory_initialization_vector=7 5 3 6 4 1 2 0 3 1 6;
```



An example COE file:

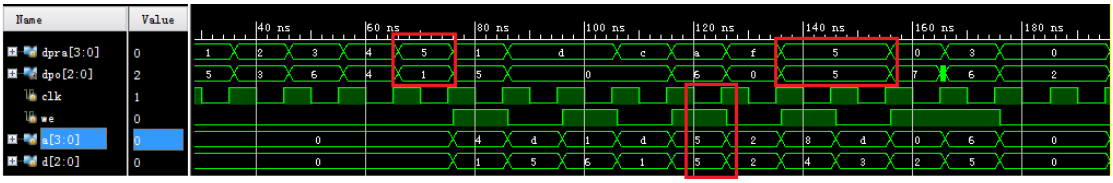
```
memory_initialization_radix = 16;
memory_initialization_vector =
23f4 0721 11ff ABe1 0001 1 0A 0
23f4 0721 11ff ABe1 0001 1 0A 0
23f4 721 11ff ABe1 0001 1 A 0
23f4 721 11ff ABe1 0001 1 A 0;
```

例化完成后，可以在“工程路径\工程名.ip_user_files\ip\dist_mem_gen_0\”目录下找到例化的文件，我们可以像调用其它模块一样调用该模块，编写仿真文件，对该 RAM 进行仿真，仿真文件代码为：

```
`timescale 1ns / 1ps
module tb( );
reg clk;
reg [3:0] a,dpra;
reg [2:0] d;
reg we;
wire [2:0] dpo;
initial
begin
    clk = 0;
    forever
    #5 clk = ~clk;
end
initial
begin
    a = 0;dpra=0;d=0;we=0;
    #20
    repeat(5)
    begin
        @(posedge clk); #1;
        dpra = dpra +1;
    end
    repeat(10)
    begin
        @(posedge clk); #1;
        a = $random%16;
        dpra = $random%16;
        d = $random%8;
        we = $random%2;
    end
```

```
        @(posedge clk); #1;
        a = 0;
        dpra = 0;
        d = 0;
        we = 0;
        #20 $stop;
end
dist_mem_gen_0  dist_mem_gen_0(
.a              (a),
.d              (d),
.dpra           (dpra),
.clk            (clk),
.we             (we),
.dpo            (dpo));
endmodule
```

其仿真波形如下图所示，可以看到，5 号地址的初始值为 1，与 coe 文件相一致，在 225ns 时钟的上升沿处，通过写端口将该地址改写成了 5，因此在 236ns 处读端口从该地读取到的是改写后的数值。



除了时钟管理单元和存储单元外，Vivado 中还提供了许多功能的 IP 核，在 IP 核参数设置界面提供了说明文档的连接，读者可选择自己感兴趣的功能模块自行学习。

实验练习

题目 1. 例化一个 16*8bit 的 ROM，并对其进行初始化，输入端口由 4 个开关控制，输出端口连接到数码管上（可只用一个数码管显示，也可 8 个数码管同时显示相同的数值），控制数码管显示与开关相对应的十六进制数字，例如四个开关输入全为零时，数码管显示“0”，输入全为 1 时，数码管显示“F”。

题目 2. 采用 8 个开关作为输入，两个数码管作为输出，采用时分复用的方式将开关的十六进制数值在两个数码管上显示出来，例如高四位全为 1，低四位全为 0 时，数码管显示“F0”。

题目 3. 利用本实验中的时钟管理单元或周期脉冲技术，设计一个精度为 0.1 秒的计时器，用 4 位数码管显示出来，数码管从高到低，分别表示分钟、秒钟十位、秒钟个位、十分之一秒，该计时器具有复位功能（可采用按键或开关作为复位信号），复位时计数值为 1234，即 1 分 23.4 秒

总结与思考

1.请总结本次实验的收获

2.请评价本次实验的难易程度

3.请评价本次实验的任务量

4.请为本次实验提供改进建议