

## 【实验目的】

学会查看原理图

理解 FPGA 开发各环节

学会使用 IP 核(知识产权核)

## 【实验环境】

PC 一台

Windows 或 Linux 操作系统

Vivado

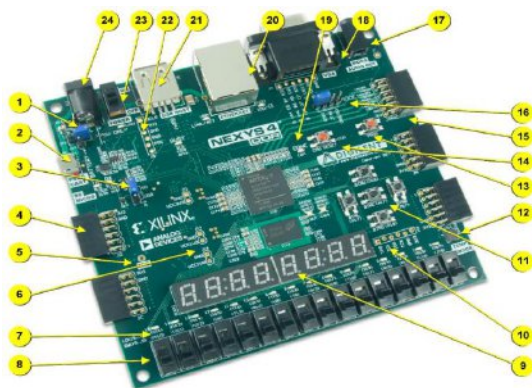
FPGA 实验平台 (Nexys4 DDR)

Logisim

[vlab.ustc.edu.cn](http://vlab.ustc.edu.cn)

## 【实验过程】

Step1: Nexys4 DDR 开发板简介



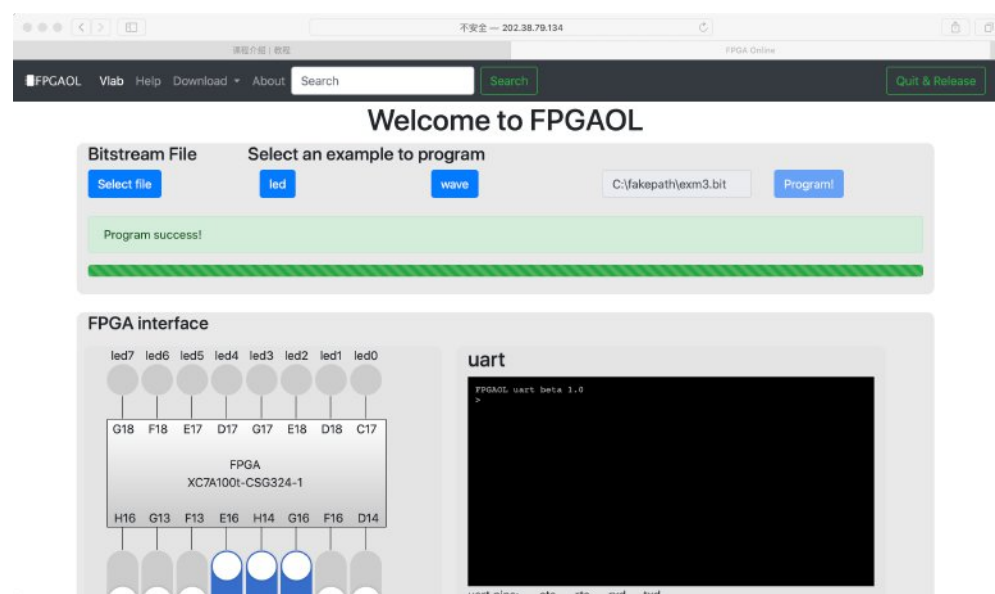
Nexys4 DDR 开发板采用 Xilinx 的 XC7A100TCSG324C-1 为核心芯片，配备了开关、按键、LED 灯、数码管等多种基本外设，还支持 USB、RJ45、VGA 等多种通用接口，以及五个用于扩展的 PMOD 接口，此外 还放置了一颗 128MB 的 DDR 内存颗粒。用户不仅可以在该开发板上完 成各种数字逻辑实验，还可以搭建一个完整的片上系统(SOC)，运 行操作系统，进行单片机或嵌入式系统的开发，功能非常强大。

## Step2: 开发板原理图介绍

虽然 FPGA 的通用管脚既可以设置为输入，也可以设置为输出，但 是在实际使用时，我们会根据其与外部电路的连接关系，固定的用来 做输入或输出，例如与按键、开关相连的引脚用作输入，与 LED、数 码管相连的引脚用作输出。

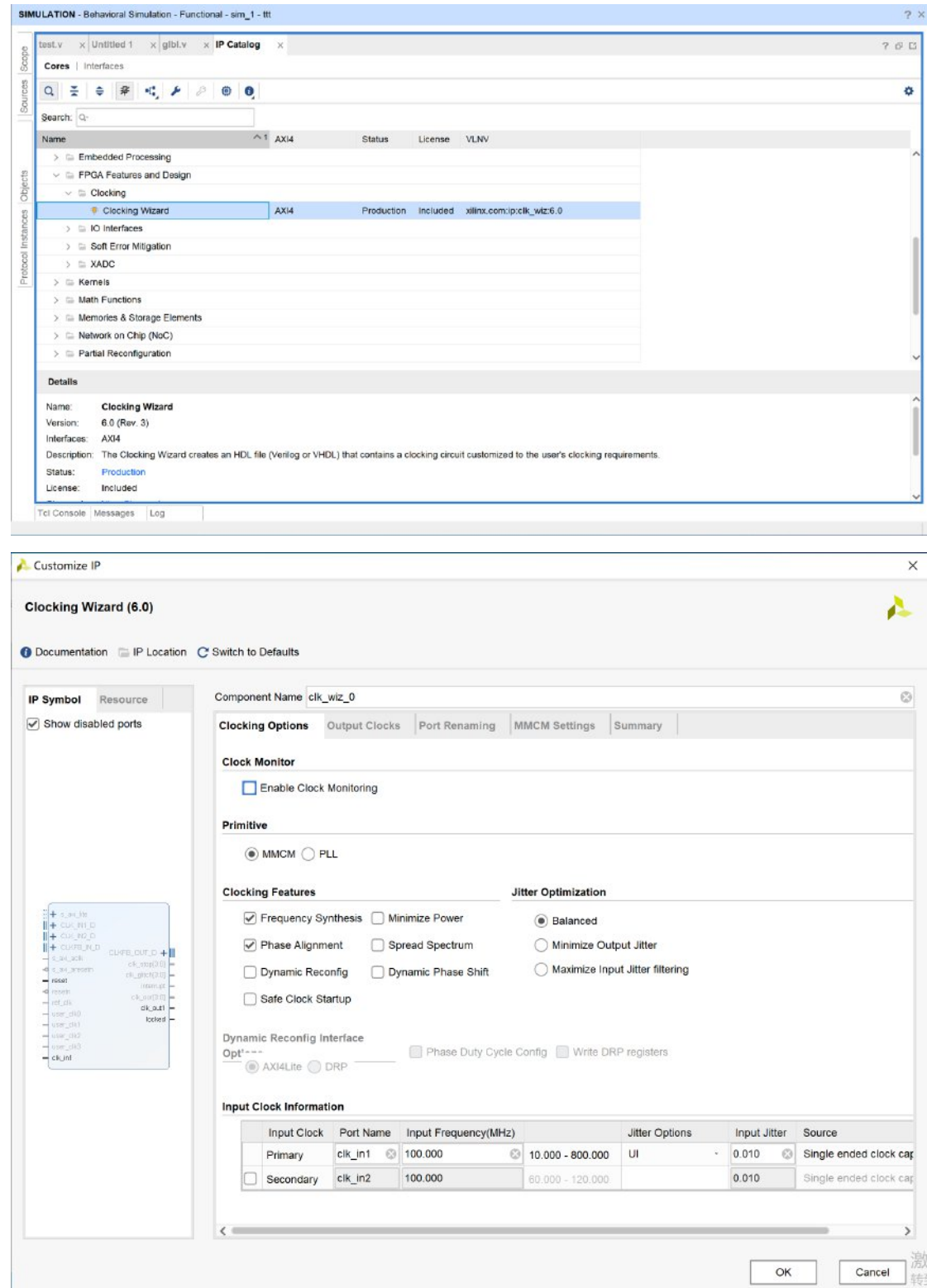
## Step3: FPGAOL 平台介绍

进行了体验：

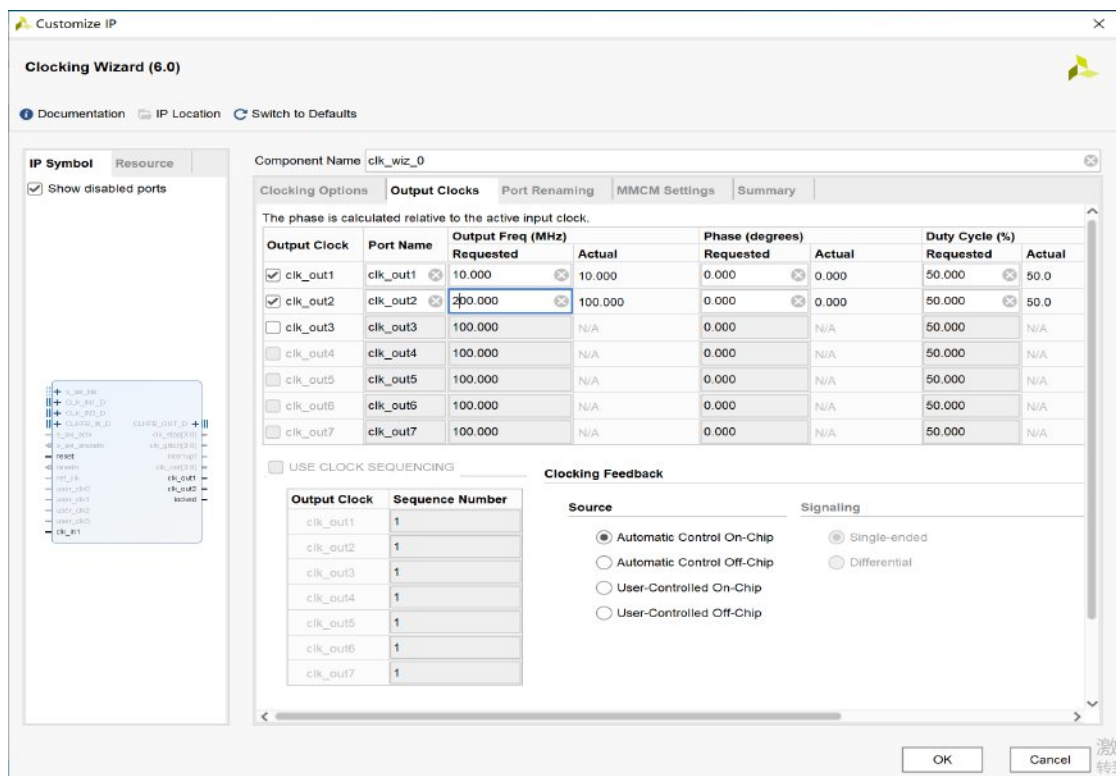


## Step4: 使用时钟管理单元 IP 核

按照讲义中的步骤设置：

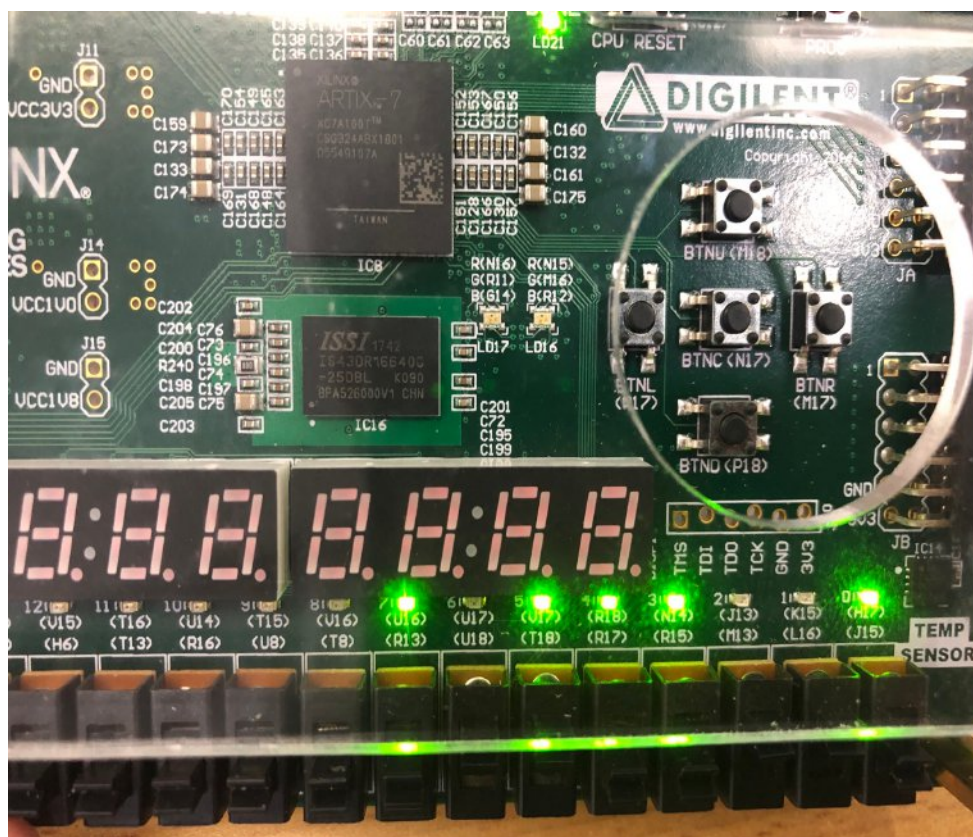


之后将两个输出频率设置为 10M 和 200M HZ



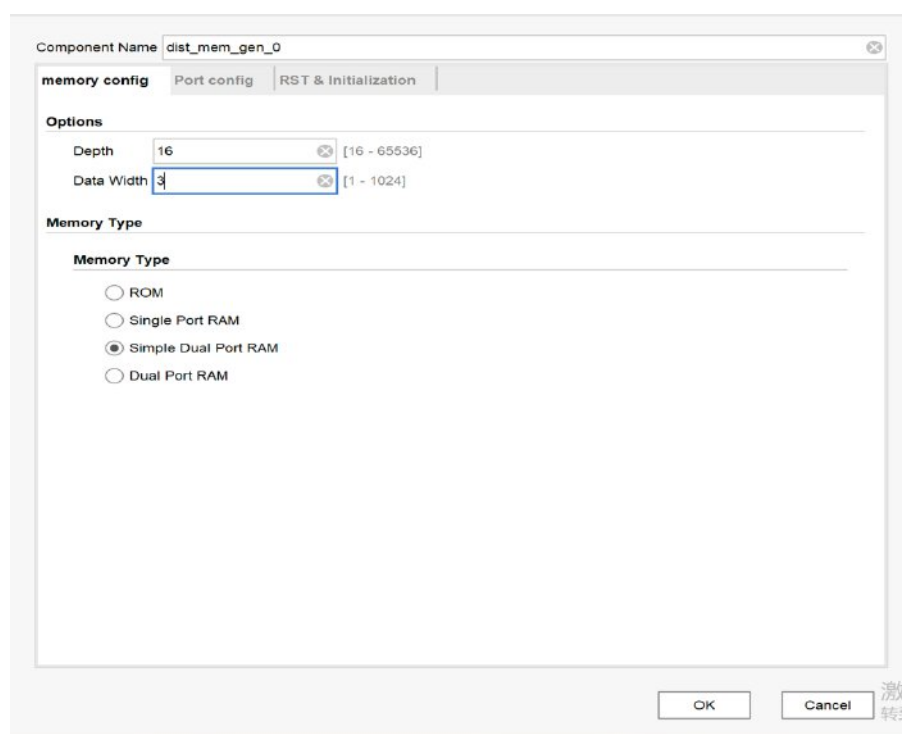
之后使用这两个信号频率来作为两个计数器的技术时钟，

将其烧写至板子上后，可以明显发现左边的 led 灯闪烁频率高于右边：



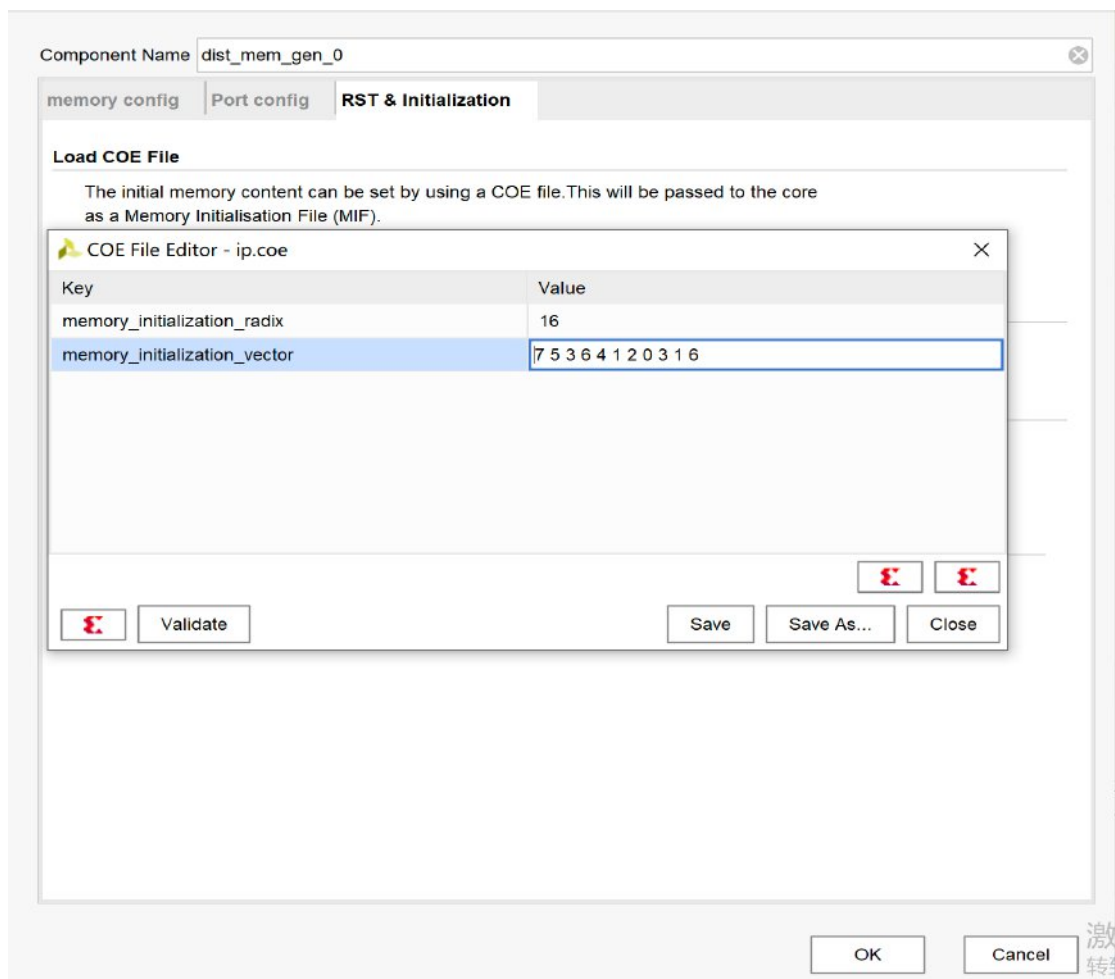
## Step5: 使用片内存储单元

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。按照讲义中内容对 memory config 的内容进行修改，如下图：

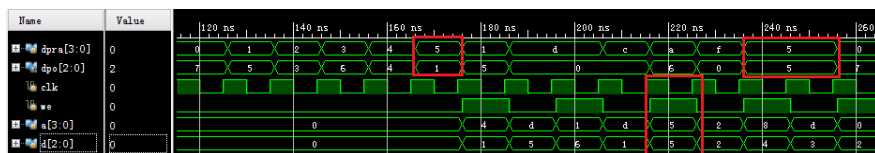


存储器类型选为简单双端口，因此包含了两套端口，其中 dpra、dpo 构成了读端口，d、a、clk、we 构成了写端口。

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，如下图所示，coe 文件格式非常简单，可以以文本方式打开和编辑。



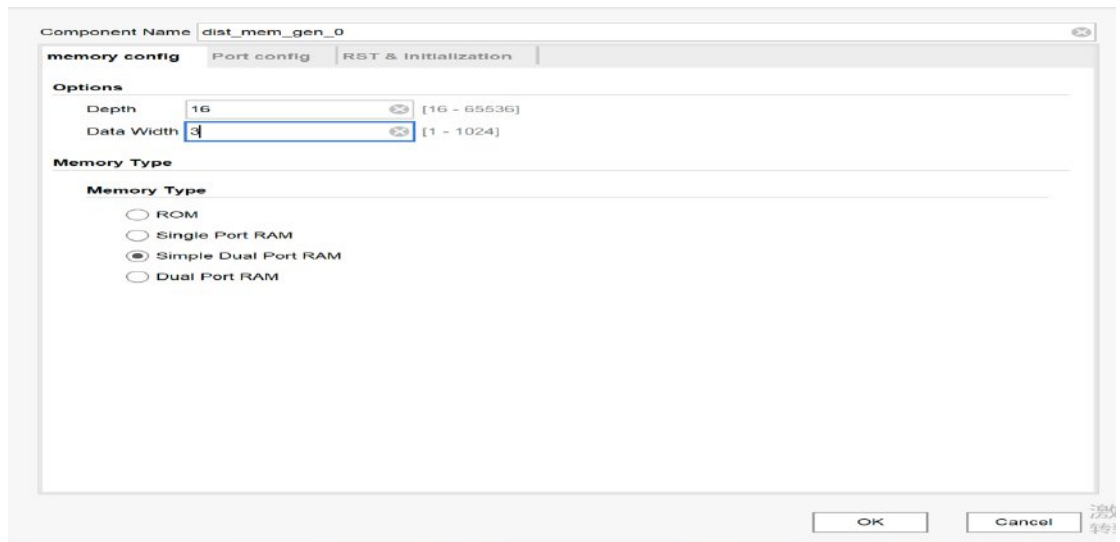
进行仿真后得到如下波形：



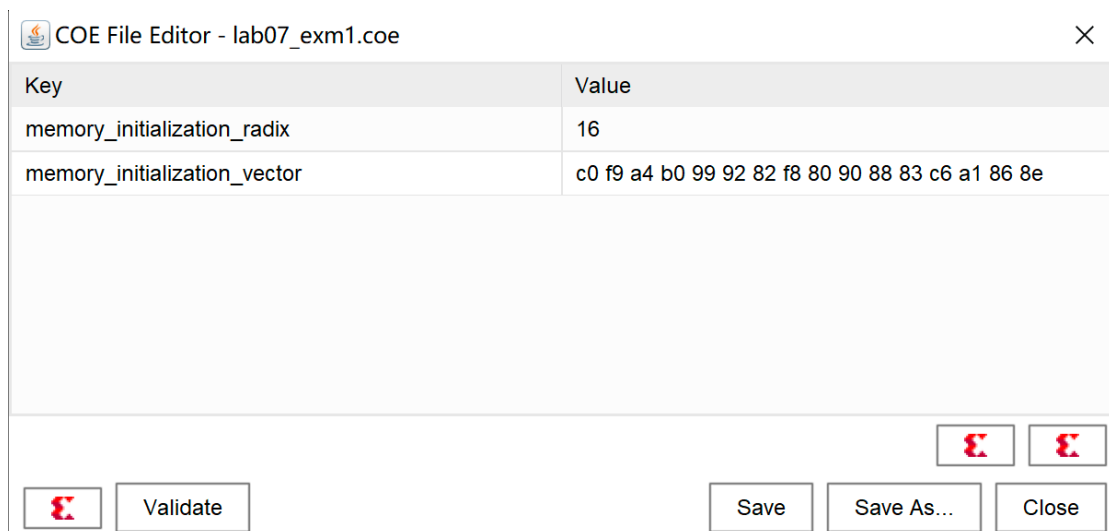
### 【实验练习】

题目 1:

首先创建一个 16\*8 的 ROM

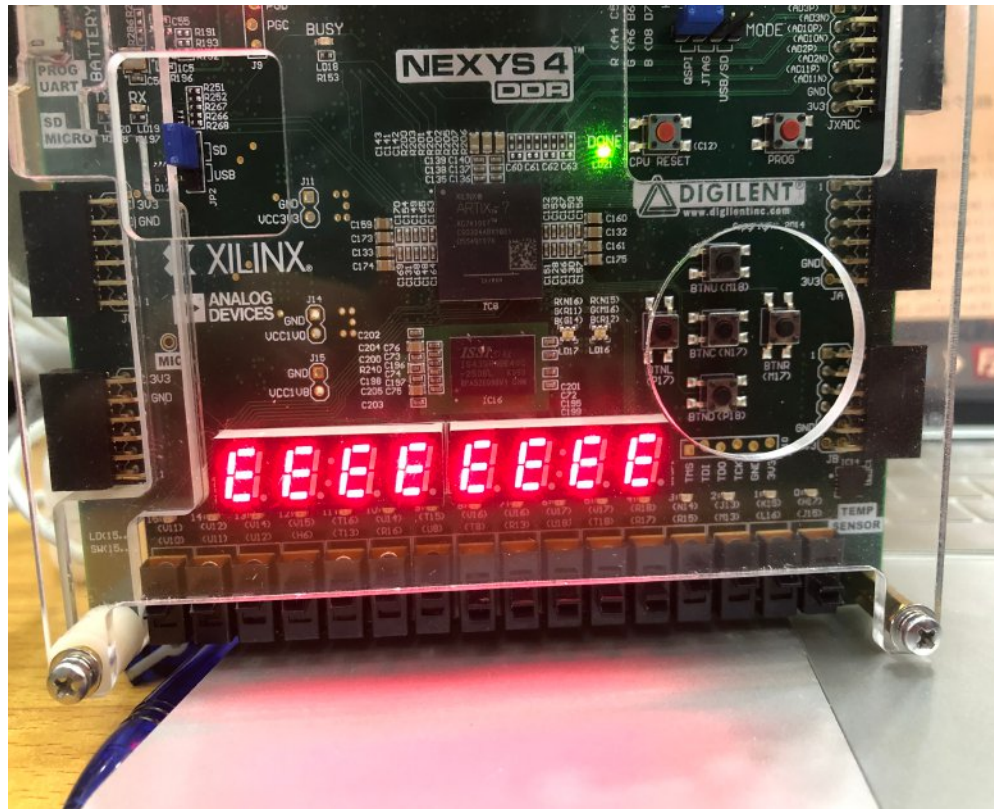


为实现将四位 2 进制数转换为十进制显示的功能，其 IP 核对应的初始化文件（.coe）文件为：



之后生成 bitstream 文件并烧写至开发板上，如下图：





## 题目 2:

与第一题相似，只用自己设置一个计数器，阈值在 10000 左右，通过这个计数器实现 AN 信号及 sw 开关信号的选择性赋值，进而在两个数码管上显示不同的数字。代码如下：



```
decode decode( .data (data), .seg (seg)); //调用译码模块
```

```
always @(posedge clk)
begin
    if(rst)
        cnt<= 15'b000000000000000;
    else if(cnt == 15'b100111000100000)
        cnt<= 15'b000000000000000;
    else
        cnt<= cnt + 15'h000000000000001;
end
always @(posedge clk) //分时复用
begin
    case(cnt)
        15'b010011100010000: an <= 8'b1111_1110;
        15'b100111000100000: an <= 8'b1111_1101;
        default: ;
    endcase
end
always @(posedge clk)
begin
    case(cnt)
        15'b010011100010000: data <= sw[3:0];
        15'b100111000100000: data <= sw[7:4];
        default: ;
    endcase
end
endmodule
```

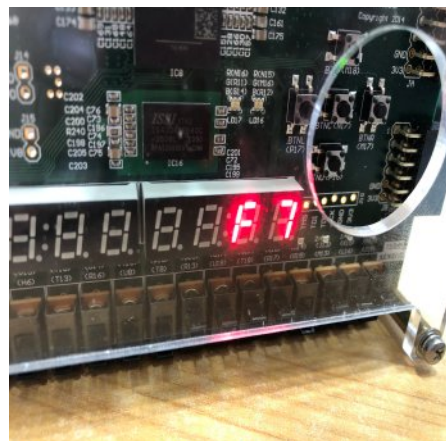
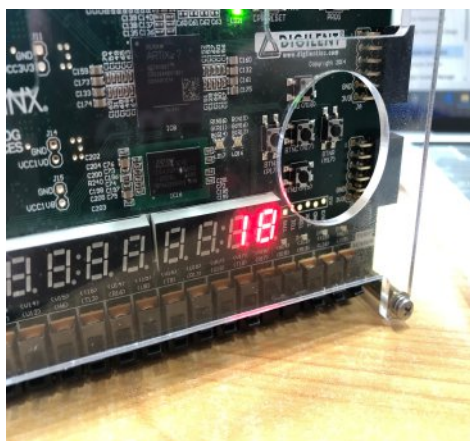
其中将数字值转换成对应数码管信号的模块代码为:

```

module decode(
input      [3:0] data,
output reg [7:0] seg);
always @(*)
begin
case(data[3:0])
4'h0: seg = 8'hc0;
4'h1: seg = 8'hf9;
4'h2: seg = 8'ha4;
4'h3: seg = 8'hb0;
4'h4: seg = 8'h99;
4'h5: seg = 8'h92;
4'h6: seg = 8'h82;
4'h7: seg = 8'hf8;
4'h8: seg = 8'h80;
4'h9: seg = 8'h90;
4'ha: seg = 8'h88;
4'hb: seg = 8'h83;
4'hc: seg = 8'hc6;
4'hd: seg = 8'ha1;
4'he: seg = 8'h86;
4'hf: seg = 8'h8e;
default: seg = 8'b0000_0011;
endcase
end
endmodule

```

之后生成 bitstream 文件并进行烧写，效果如下图所示：



题目 3:

使用周期脉冲技术，设置一个阈值为 10000000 的计数器，产生一个 10HZ 的信号，来实现精度为 0.1s 的计时器：

```

module exm3(
    input clk,rst,
    output reg [7:0]SSEG_CA,
    output reg [7:0]SSEG_AN );

    wire clk10,clk20;    //两个分频信号

    reg [23:0] cnt1;
    reg [9:0] cnt;
    reg [1:0] sel;

    reg [3:0] o1;    //o1到o4表示显示的4个数的具体数值
    reg [3:0] o2;
    reg [3:0] o3;
    reg [3:0] o4;


    wire [7:0] temp1;    //temp1~4表示4个ROM的输出结果，需定义为wire型
    wire [7:0] temp2;
    wire [7:0] temp3;
    wire [7:0] temp4;

```

模块中所使用的输入输出及内部变量如上图所示, 具体作用可以参考  
注释

所用的 ROM 与题目 1 中一致;

Key	Value
memory_initialization_radix	16
memory_initialization_vector	c0 f9 a4 b0 99 92 82 f8 80 90 88 83 c6 a1 86 8e



Validate

Save

Save As...

Close

temp1~4 四个 wire 变量用来临时存储 o1~4 四个数对应的七段数码管信号，他们分别接在四个 ROM 的输出端口上；

之后控制分频信号，cnt1 产生 10HZ 的时钟信号，cnt 控制计数扫描，仿照讲义中的用法编写如下：

```

} always @(posedge clk)
} begin
}   if(rst)
        cnt1 <= 24'b0;
}   else if(cnt1>=100000000)
        cnt1 <= 24'b0;
        else
}       cnt1 <= cnt1 + 24'b1;
} end
    assign clk10=(cnt1==1); //clk10 产生10hz的时钟信号

} always @(posedge clk)
} begin

}   if(cnt>=10000)
        cnt <= 10'b0;
        else
}       cnt <= cnt + 10'b1;
} end
    assign clk20=(cnt==1); //clk20 控制扫描信号sel

```

并用 clk20 控制 sel 信号：

```

always @(posedge clk )
    if(clk20)
        sel<=sel+2'b1; //根据clk20更改选择信号

```

根据选择信号 sel，这四个信号及对应的 AN 信号被赋给模块的两个输出，如下图所示：

```

always @(posedge clk)
if (clk20==0)
    case (sel)
        2'b00:begin
            SSEG_CA<=temp1;
            SSEG_AN<= 8'b11110111;
        end
        2'b01:begin
            SSEG_CA<=temp2;
            SSEG_AN<= 8'b11111011;
        end
        2'b10:begin
            SSEG_CA<=temp3;
            SSEG_AN= 8'b11111101;
        end
        2'b11:begin
            SSEG_CA<=temp4;
            SSEG_AN= 8'b11111110;
        end
    endcase

```

之后设计实现复位、进位功能的代码，（注意“秒”到“分”是 60 进制）：

（注意信号 clk10 的频率是 10HZ，用它来确定基本时间单位）

（计数进位时，采用 if 嵌套的方式实现，避免出现“A”）

代码如下图所示：

```

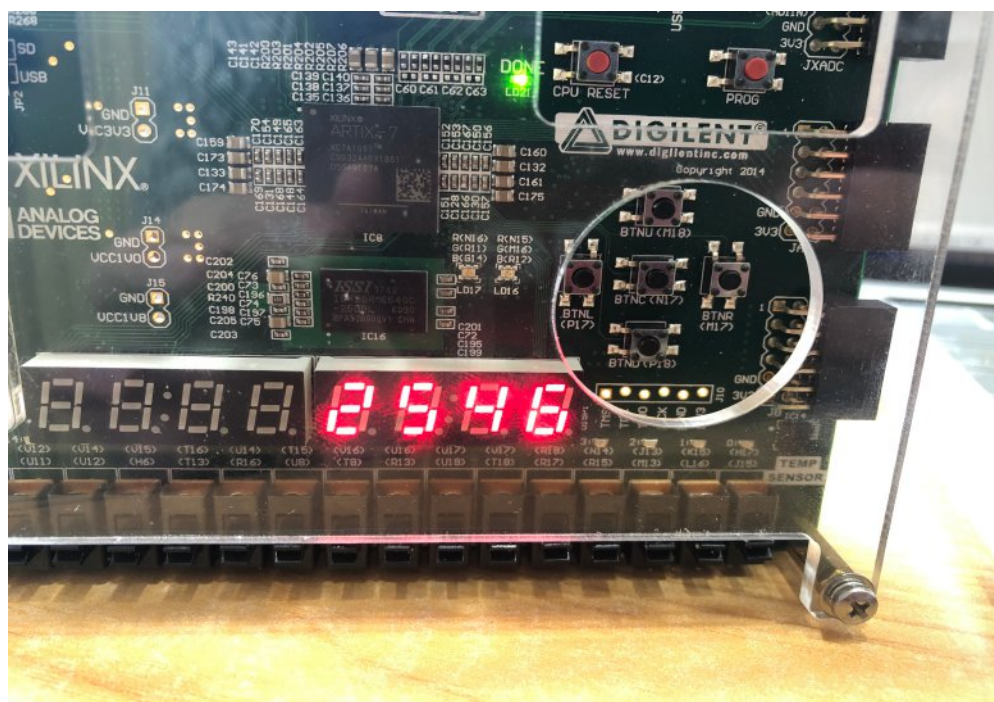
always @(posedge clk )
    if(rst)
    begin
        o1<=4'b0001;
        o2<=4'b0010;
        o3<=4'b0011;
        o4<=4'b0100;
    end
    else if(clk10)
    begin
        if(o4==9)
        begin
            o4<=0;
            if(o3==9)
            begin
                o3<=0;
                if(o2==5)
                begin
                    o2<=0;
                    if(o1 == 9)
                    o1<=0;
                    else
                    o1<=o1+1;
                end
                else
                o2<=o2+1;
            end
            else
            o3<=o3+1;    //进位
        end
        else
        o4<=o4+1;
    end

```

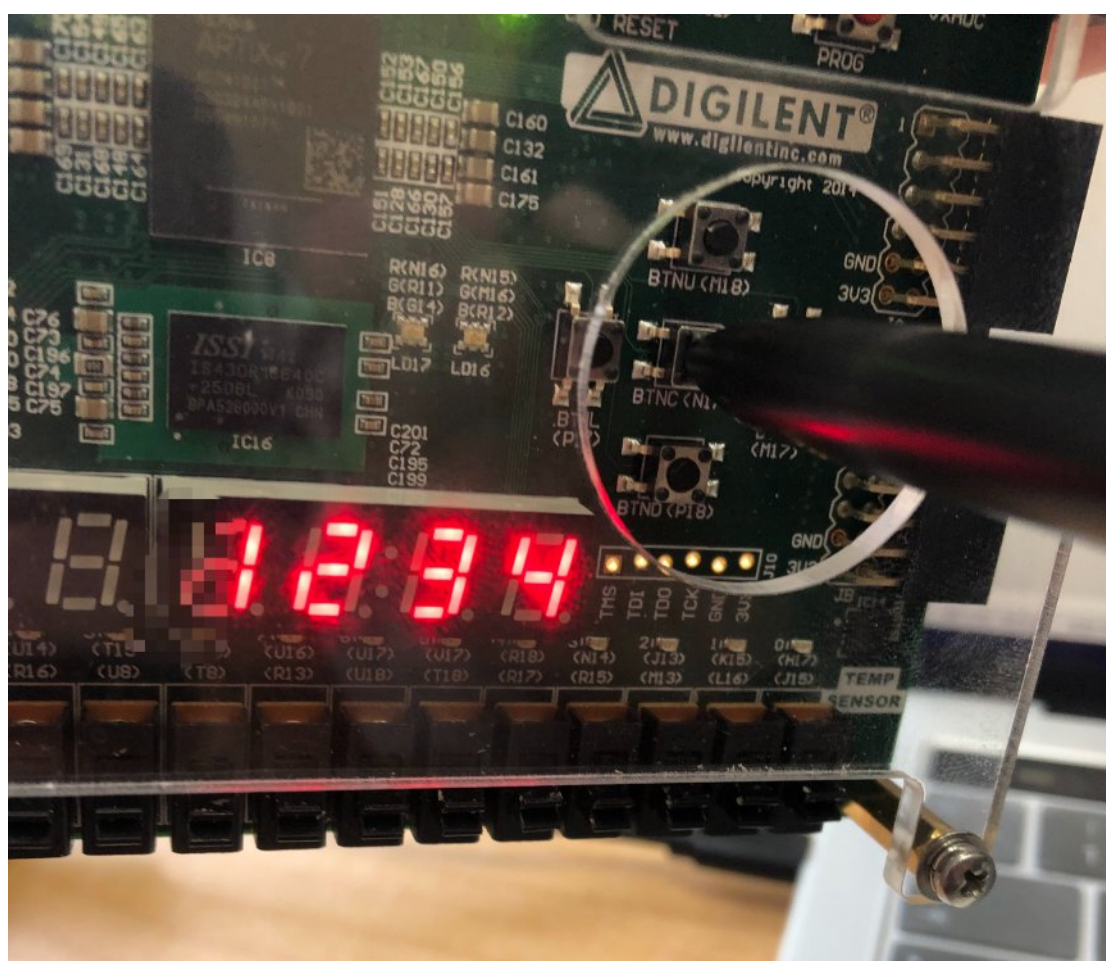
---

之后即可生成 **bitstream** 文件，并且烧写在开发板上，效果如下图：





复位时，数码管显示如下：



## 【总结与思考】

1. (1) 了解了 IP 核的用法与设计，以后自己在开发过程中可以利用第三方提供的 IP 核实现一部分功能

(2) 了解了 Nexys4 开发板上 SSEG\_AN 信号控制七段数码管的作用，学会了用分段扫描的方法，以控制不同的数码管上显示不同的数字

(3) 掌握了用计数器产生低频信号的方法

2. 难

3. 略大

4. 实验讲义关于七段数码管的使用应该添加更详细地讲述；

IP 核部分的讲解也有些过于简略