

中国科学技术大学计算机学院
《数字电路实验》报告



实验题目： FPGA 实验平台以及 ip 核的使用
学生姓名： 牛庆源
学生学号： PB21111733
完成日期： 2022. 11. 24

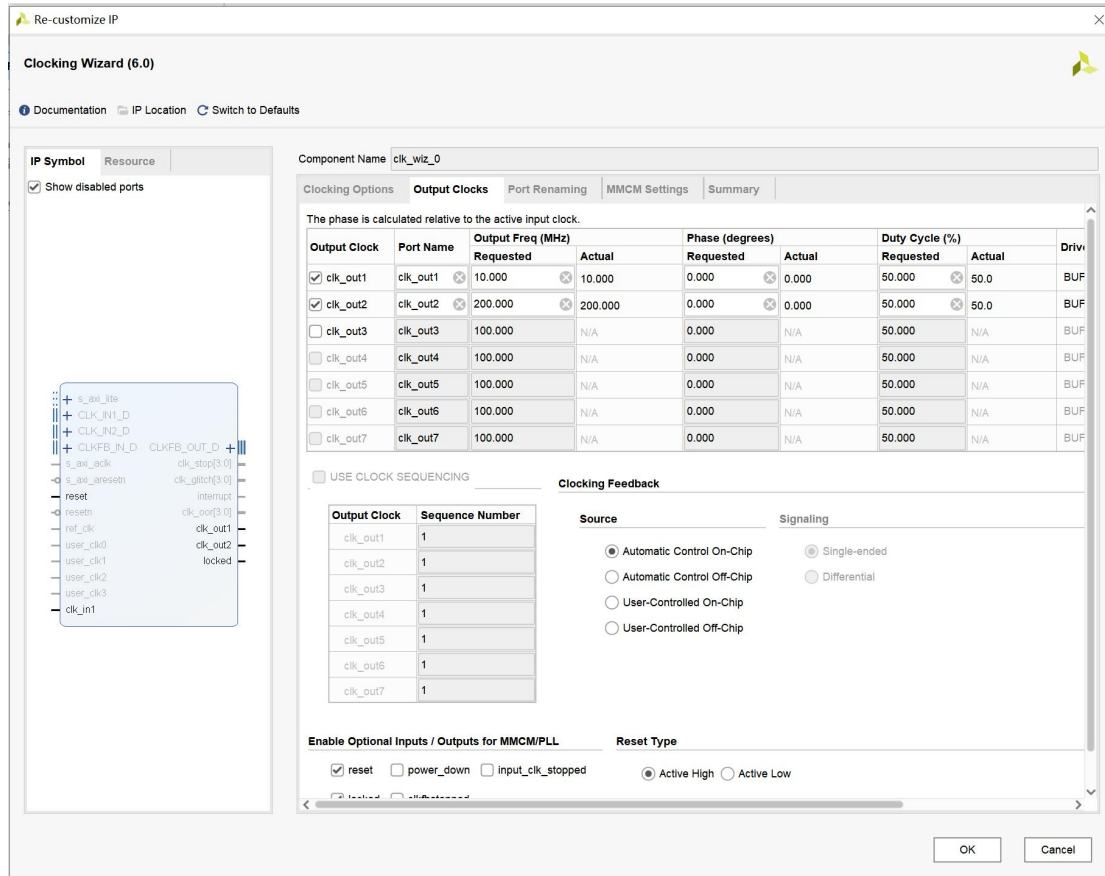
【实验步骤】（展示需要进行操作的 step）

Step3:

1. 设计文件如下:

```
1 | `timescale 1ns / 1ps
2 | ////////////////////////////////////////////...
21 |
22 |
23 | module test(
24 |     input clk,
25 |     input rst,
26 |     output [7:0] led);
27 |     wire clk_10m, clk_200m, locked;
28 |     reg [31:0] cnt_1, cnt_2;
29 |     always@(posedge clk_200m)
30 |     begin
31 |         if(~locked)
32 |             cnt_1 <= 32'hAAAA_AAAA;
33 |         else
34 |             cnt_1 <= cnt_1+1'b1;;
35 |     end
36 |     always@(posedge clk_10m)
37 |     begin
38 |         if(~locked)
39 |             cnt_2 <= 32'hAAAA_AAAA;
40 |         else
41 |             cnt_2 <= cnt_2+1'b1;;
42 |     end
43 |     assign led = {cnt_1[27:24], cnt_2[27:24]};
44 |     clk_wiz_0 clk_wiz_0_inst(
45 |         .clk_in1 (clk),
46 |         .clk_out1 (clk_10m),
47 |         .clk_out2 (clk_200m),
48 |         .reset (rst),
49 |         .locked (locked));
50 | endmodule
51 |
52 |
53 |
54 |
55 |
```

2. Ip 核配置如下:



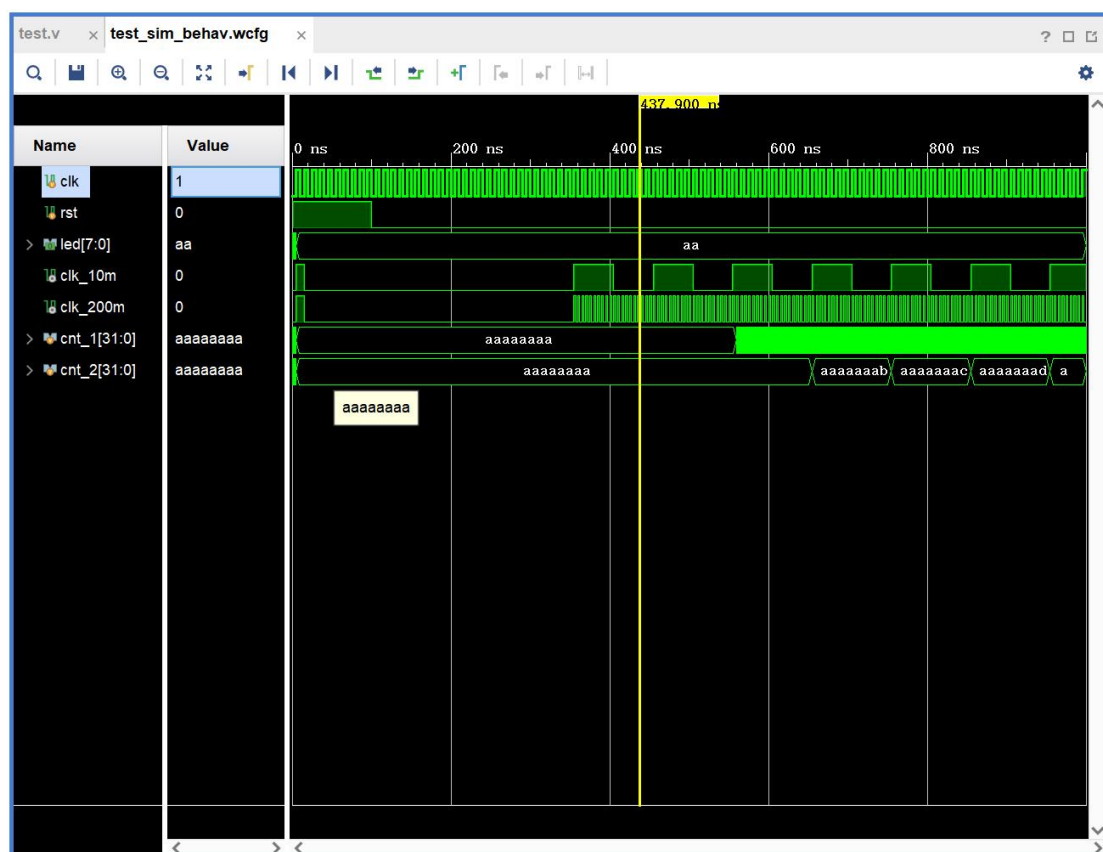
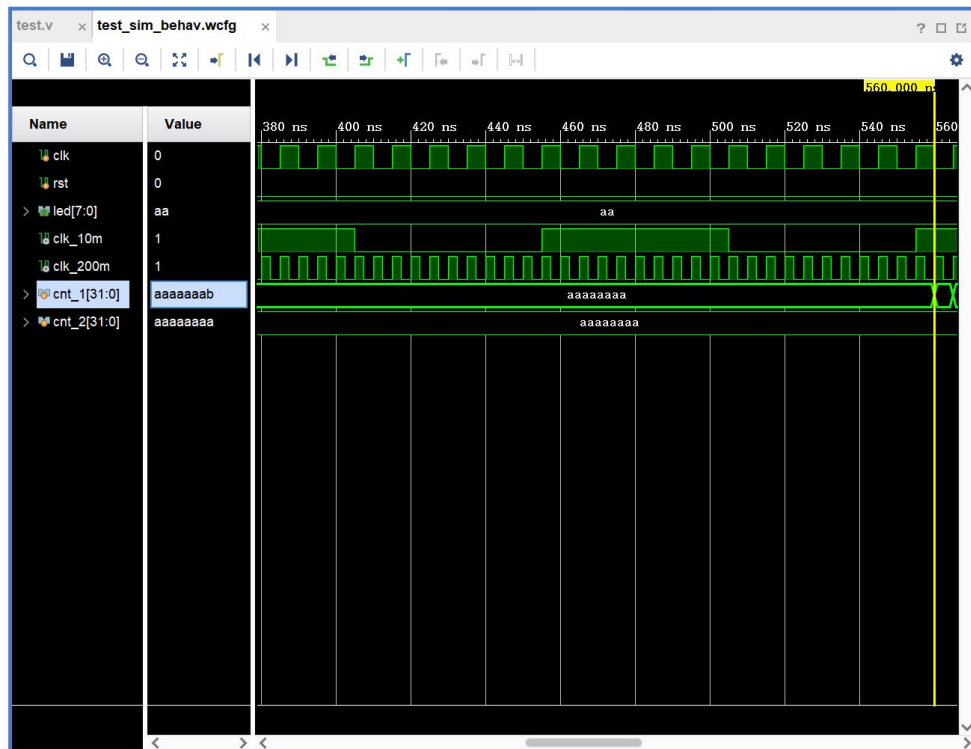
3. 仿真文件如下:

```

1  `timescale 1ns / 1ps
2  ////////////////////////////////////////////...
21
22
23 module test_sim(
24     reg clk,rst;
25     initial
26     begin
27         clk = 0;
28         forever
29             #5 clk = ~clk;
30     end
31     initial
32     begin
33         rst = 1;
34         #100 rst = 0;
35     end
36
37     test test(
38         .clk (clk),
39         .rst (rst),
40         .led ());
41 endmodule
42
43
44
45

```

4. 仿真波形如下: (部分波形和全部波形)



5. 约束文件如下：

```

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

```

Step4

1. Ip 核配置如下:

Re-customize IP

Distributed Memory Generator (8.0)

Documentation IP Location Switch to Defaults

Component Name: dist_mem_gen_0

memory config Port config RST & Initialization

Options

Depth: 16 [16 - 65536]

Data Width: 3 [1 - 1024]

Memory Type

Memory Type

☐ ROM

☐ Single Port RAM








☒ Simple Dual Port RAM

☐ Dual Port RAM

OK Cancel

Port List:

- a[3:0]
- d[2:0]
- dpra[3:0]
- clk
- we
- i_ce
- qspo_ce
- qdpo_ce
- qdpo_clk
- qspo_rst
- qdpo_rst
- qspo_srst
- qdpo_srst
- spo[2:0]
- dpo[2:0]
- qspo[2:0]
- qdpo[2:0]

COE File Editor - ip.coe	
Key	Value
memory_initialization_radix	16
memory_initialization_vector	fc 60 da f2 66 b6 be e0 fe f6 ee 3e 1a 7a 9e 8e
<div>    </div>	
	  
Validate	

2. 例化该 ip 核的仿真文件和得到的仿真波形如下：

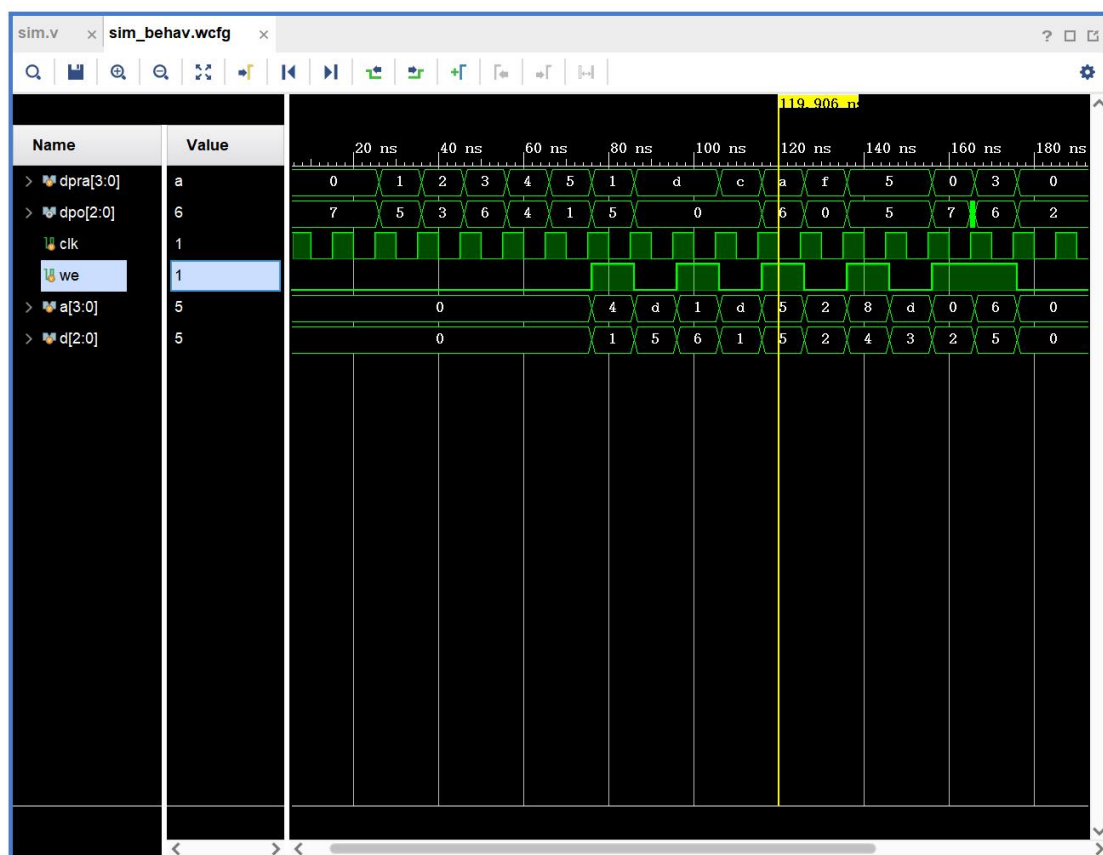
```

1  `timescale 1ns / 1ps
2  module sim( );
3  reg clk;
4  reg [3:0] a, dpra;
5  reg [2:0] d;
6  reg we;
7  wire [2:0] dpo;
8  initial
9  begin
10   clk = 0;
11   forever
12   #5 clk = ~clk;
13 end

14 initial
15 begin
16   a = 0; dpra=0; d=0; we=0;
17   #20
18   repeat(5)
19   begin
20     @(posedge clk); #1;
21     dpra = dpra +1;
22   end
23   repeat(10)
24   begin
25     @(posedge clk); #1;
26     a = $random%16;
27     dpra = $random%16;
28     d = $random%8;
29     we = $random%2;
30   end
31   @(posedge clk); #1;
32   a = 0;
33   dpra = 0;
34   d = 0;
35   we = 0;
36   #20 $stop;
37 end

38 dist_mem_gen_0 dist_mem_gen_0(
39   .a (a),
40   .d (d),
41   .dpra (dpra),
42   .clk (clk),
43   .we (we),
44   .dpo (dpo));
45 endmodule

```



【实验题目】

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，输出端口连接到七段数码管上（七段数码管与 LED 复用相同的一组管脚），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”。
2. 采用 8 个开关作为输入，两个十六进制数码管作为输出，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”。
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。

【实验目的】

1. 学会将 vivado 库中自带的元件（ip 核）例化在自己的设计文件或者仿真文件内。
2. 学会配置改变时钟快慢的 ip 核和 ROM & RAMip 核。对于 ROM 和 RAM 编写 coe 文件控制。
3. 学会使用分时复用的方法来在 FPGA 平台显示多位的七段数码管。
4. 对 100Mhz 时钟频率以及通过 count 计数的方式改变输入时钟频率这种方法产生深刻的理解，学会自己产生指定频率的时钟信号。

【实验环境】

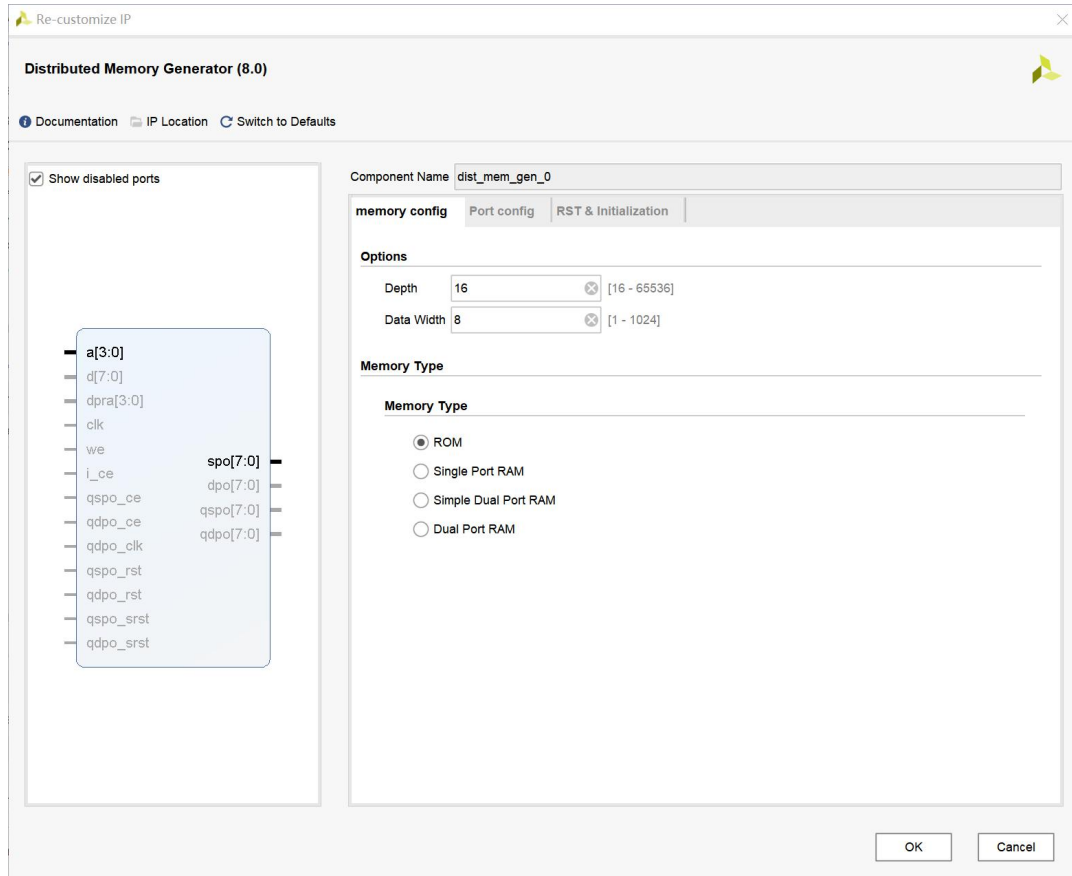
fpgaol.ustc.edu.cn

Vivado 软件

【实验练习】

题目一

要使用一个 16*8bit 的 ROM，可以选取 Distributed Memory Generater 这一 ip 核，配置 ip 核的内容。寄存器类型选择 ROM, depth 设置为 16, data width 设置为 8，如图：



对于 ROM 中 a 与 spo 的选择关系，我们可以编辑 coe 来改变。每一个 a 生成一个对应的 spo，从而满足在七段数码管分别显示 0 到 F。例如生成数字“0”，a 应当为 000，数字“0”对应的二进制由数码管的输入顺序，可以得到为 0011_1111 即十六进制的 3f，由此方法按顺序编辑出 0 到 F，得到的 coe 文件如下：

```
ip.coe - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
memory_initialization_radix=16;
memory_initialization_vector=3f,06,5b,4f,66,6d,7d,07,7f,6f,77,7c,39,5e,79,71;
```

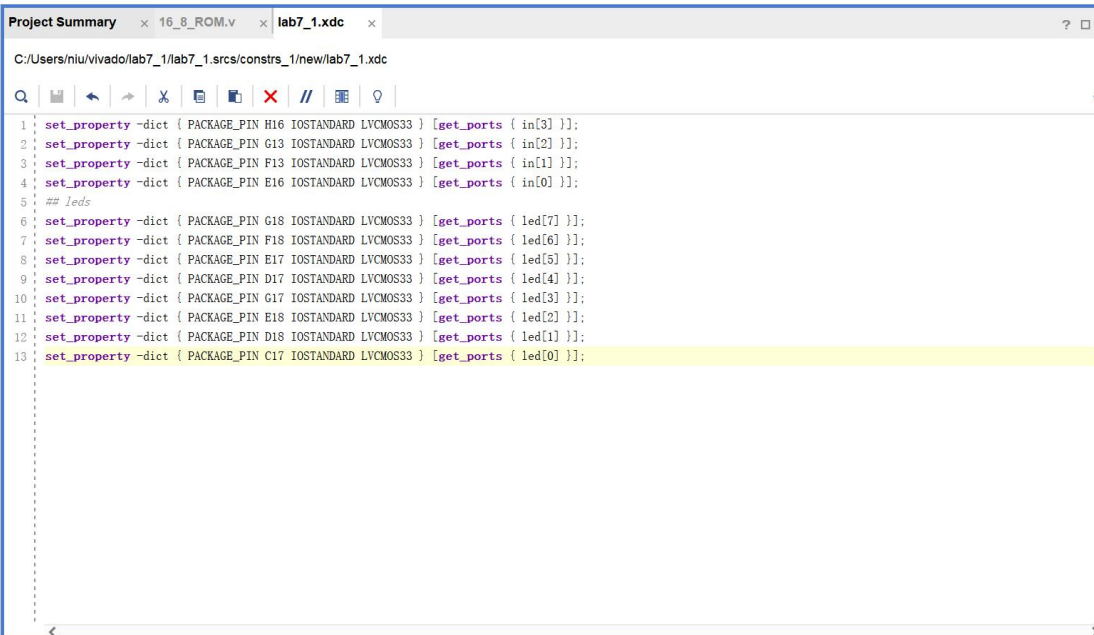
设计文件直接设置四位输入 in 和八位输出 led，并直接例化上述生成的 ip 核即可，文件如下：

```

1  | `timescale 1ns / 1ps
2  | //////////////////////////////////////
21 |
22 |
23 | module ROM(
24 |     input [3:0] in,
25 |     output [7:0] led);
26 |
27 |     dist_mem_gen_0 dist_mem_gen_0(
28 |         .a(in),
29 |         .spo(led));
30 |
31 | endmodule
32 |

```

编写约束文件将 in 和 led 约束到 FPGA 平台的规定引脚上,如下:



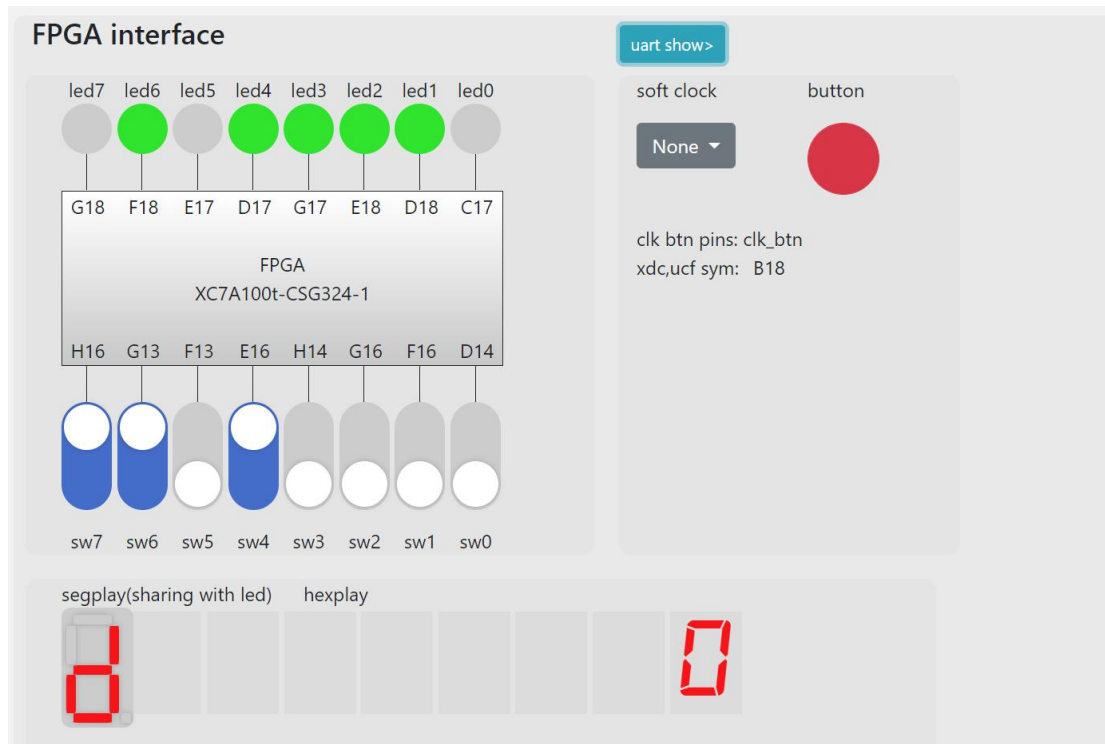
```

Project Summary x 16_8_ROM.v x lab7_1.xdc x
C:/Users/niu/vivado/lab7_1/srcs/constrs_1/new/lab7_1.xdc

1 set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33 } [get_ports { in[3] }];
2 set_property -dict { PACKAGE_PIN G13 IOSTANDARD LVCMOS33 } [get_ports { in[2] }];
3 set_property -dict { PACKAGE_PIN F13 IOSTANDARD LVCMOS33 } [get_ports { in[1] }];
4 set_property -dict { PACKAGE_PIN E16 IOSTANDARD LVCMOS33 } [get_ports { in[0] }];
5 ## leds
6 set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
7 set_property -dict { PACKAGE_PIN F18 IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
8 set_property -dict { PACKAGE_PIN E17 IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
9 set_property -dict { PACKAGE_PIN D17 IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
10 set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
11 set_property -dict { PACKAGE_PIN E18 IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
12 set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
13 set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33 } [get_ports { led[0] }];

```

生成 bits 流并在 FPGA 平台烧写,可以得到结果(这里将 in 约束在了 sw 高四位):



题目二

配置 ip 核与题目一中的 ip 核相同，因为都是利用 16*8bits 的 ROM 在数码管上显示数字。

八个开关作为输入，高四位控制高位数码管，低四位控制低位数码管，高位和低位的数码管可以通过约束 an 的方式选取，由 FPGA 平台给出的约束文件样例可以了解到，选取 an 为 001 和 000 可以分别显示在高位和低位上。选取 an 为 001，此时将输入的高四位 sw[7:4] 赋值给控制端 data；选取 an 为 000，此时将输入的低四位 sw[3:0] 赋值给控制端 data。

由于原时钟频率过快，导致无法分时显示出高位和低位的数码管，所以可以采用计数器的方法生成一个频率较低的时钟，从而便于分时显示，这里采用了一个四位的 count，每一个时钟上升沿将 count 加 1，当 count 值为 4'b1111 时，将设置的时钟 temp 翻转一次，如此

循环得到一个频率较低的时钟信号，用这个时钟信号控制分时复用。

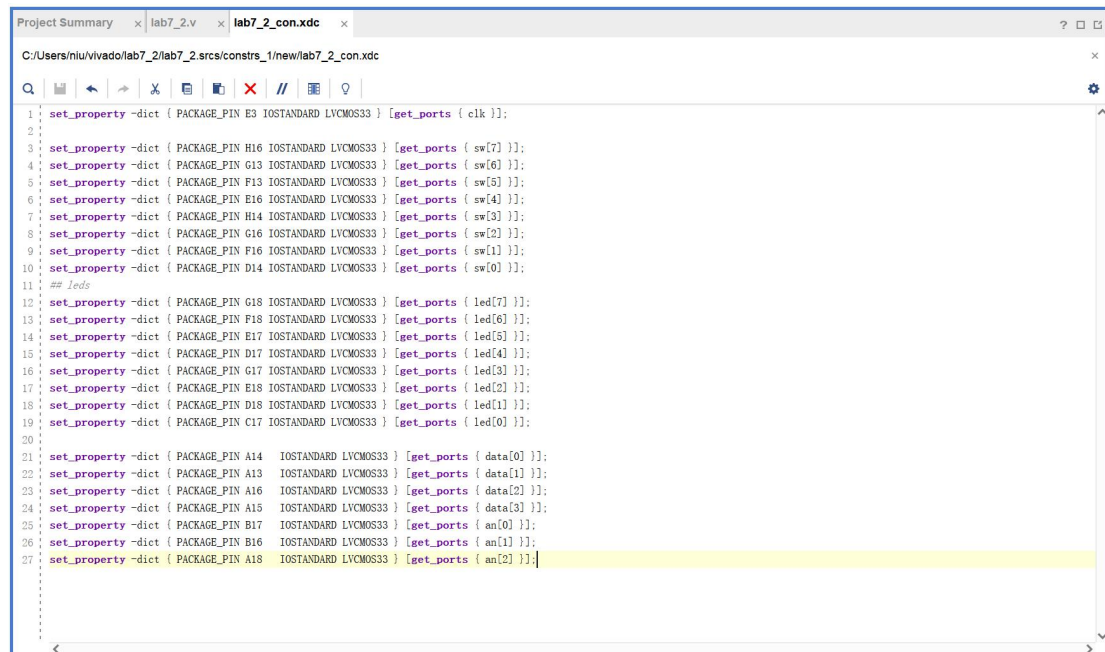
在生成的时钟信号 temp 处于高电平时，选取 an 为 001, 和高四位输入，反之选取 an 为 000, 和低四位输入，例化 ip 核。

通过以上分析编写设计文件如下：

```
23 module lab7_2(  
24     input clk,  
25     input [7:0] sw,  
26     output [7:0] led,  
27     reg [3:0] data,  
28     reg [2:0] an  
29 );  
30 reg [3:0] count;  
31 reg temp = 1'b0;  
32 always @ (posedge clk) begin  
33     count <= count + 1;  
34 end  
35 always @ (*) begin  
36     if(count == 4'b1111)  
37         temp = ~temp;  
38 end  
39 always @ (*) begin  
40     if(temp) begin  
41         an <= 3'b001;  
42         data <= sw[7:4];  
43     end  
44     else begin  
45         an <= 3'b000;  
46         data <= sw[3:0];  
47     end  
48 end  
49 dist_mem_gen_0 dist_mem_gen_0(  
50     .a(data),  
51     .spo(led));  
52 endmodule  
53
```

将输出 led 以及输入 sw 约束在相应管脚,an 和 data 也根据 FPGA 给出的样例约束文件进行约束,同样不要忘记将频率为 100Mhz 的时钟信号约束。

得到约束文件如下:



```
1 set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clk }];
2
3 set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33 } [get_ports { sw[7] }];
4 set_property -dict { PACKAGE_PIN G13 IOSTANDARD LVCMOS33 } [get_ports { sw[6] }];
5 set_property -dict { PACKAGE_PIN F13 IOSTANDARD LVCMOS33 } [get_ports { sw[5] }];
6 set_property -dict { PACKAGE_PIN E16 IOSTANDARD LVCMOS33 } [get_ports { sw[4] }];
7 set_property -dict { PACKAGE_PIN H14 IOSTANDARD LVCMOS33 } [get_ports { sw[3] }];
8 set_property -dict { PACKAGE_PIN G16 IOSTANDARD LVCMOS33 } [get_ports { sw[2] }];
9 set_property -dict { PACKAGE_PIN F16 IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
10 set_property -dict { PACKAGE_PIN D14 IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
11
12 ## leds
13 set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
14 set_property -dict { PACKAGE_PIN F18 IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
15 set_property -dict { PACKAGE_PIN E17 IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
16 set_property -dict { PACKAGE_PIN D17 IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
17 set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
18 set_property -dict { PACKAGE_PIN E18 IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
19 set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
20 set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
21
22 set_property -dict { PACKAGE_PIN A14 IOSTANDARD LVCMOS33 } [get_ports { data[0] }];
23 set_property -dict { PACKAGE_PIN A13 IOSTANDARD LVCMOS33 } [get_ports { data[1] }];
24 set_property -dict { PACKAGE_PIN A16 IOSTANDARD LVCMOS33 } [get_ports { data[2] }];
25 set_property -dict { PACKAGE_PIN A15 IOSTANDARD LVCMOS33 } [get_ports { data[3] }];
26 set_property -dict { PACKAGE_PIN B17 IOSTANDARD LVCMOS33 } [get_ports { an[0] }];
27 set_property -dict { PACKAGE_PIN B16 IOSTANDARD LVCMOS33 } [get_ports { an[1] }];
28 set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33 } [get_ports { an[2] }];
```

生成 bits 流导入 FPGA 平台烧写可以得到如下结果:



题目三

本题目 ip 核同样与题目一和题目二相同，用途都是使数码管显示对应的数字。

本题目要求生成四位的计时器，即采用分时复用的方式在不同的时间产生不同的输入信号，结果 ip 核处理得到不同的数字显示，通过 an 的选取来将不同的数字产生在四位上。

首先产生一个周期为 0.1s 的时钟，即频率为 10hz 的时钟，由于时钟默认产生为 100Mhz 时钟，所以我们需要通过一个计数器将 100Mhz 的时钟降到 10hz，即计数器从 0 记到 9999999 到 10 的八次方时，过去了 0.1s，将 0.1s 设为一个单位时间，在这一个单位时间内进行时钟的变化。

以下是很自然得到的计数器原理：

首先变化 0.1s 的位，每一个单位时间加 1，当大于 8 时不再加 1（此时该位为 9），下个单位时间归为 0，并将 1s 的位加 1。当 1s 的位大于 8 时不再加 1（此时该位为 9），下一个进位的时刻将其归为 0，并将 10s 的位加 1。当 10s 的位大于 4 时不再加 1（此时该位为 5），下个进位的时刻将其归为 0，并将 1min 的位加 1。当 1min 的位大于 8 时不再加 1（此时该位为 9），下一个进位的时刻将其归为 0。

对于 rst 信号，我们将其设置为高电平有效，高电平时将从高到低位赋值为 1 2 3 4，即使用 if 语句判断，然后依次赋值即可。else 则不断计时即可。

采用分时复用的方式，分别当 an 为 011 010 001 000 时将输入的 data 分别赋值为不同位数的 clk: 1min 位， 10s 位， 1s 位， 0.1s 位。

最后例化 ip 核即可。

通过上述的推导可以得到设计文件如下：

```
23 module lab7_3(  
24     input clk, rst,  
25     output [7:0] led,  
26     output reg [3:0] data,  
27     output reg [2:0] an  
28 );  
29     reg [23:0] timer;  
30     reg [3:0] clk_0_1;  
31     reg [3:0] clk_1;  
32     reg [3:0] clk_10;  
33     reg [3:0] clk_60;  
34     reg [3:0] count;  
35     reg temp = 2'b00;
```



```

36 always@(posedge clk) begin
37     if(rst) begin clk_60 <= 4'h1; clk_10 <= 4'h2; clk_1 <= 4'h3; clk_0_1 <= 4'h4; end
38     else begin
39         timer <= 24'h0;
40         if(timer>=9999999) begin
41             if(clk_0_1 > 8) begin
42                 clk_0_1 <= 4'b0000;
43                 clk_1 <= clk_1 + 1;
44                 if(clk_1 > 8) begin
45                     clk_1 <= 4'b0000;
46                     clk_10 <= clk_10 + 1;
47                     if(clk_10 > 4) begin
48                         clk_10 <= 4'b0000;
49                         clk_60 <= clk_60 + 1;
50                         if(clk_60 > 8) begin
51                             clk_60 <= 4'b0000;
52                         end
53                         else clk_60 <= clk_60 + 1;
54                     end
55                     else clk_10 <= clk_10 + 1;
56                 end
57                 else clk_1 <= clk_1 + 1;
58             end
59             else clk_0_1 <= clk_0_1 + 1;
60             timer <= 24'h0;
61         end
62     else
63         timer <= timer + 24'h1;
64     end
65 end

```

```

66 always@(posedge clk)
67 begin
68     count = count + 1;
69     an = count[3:2];
70     case (count[3:2])
71         2'b00 : data <= clk_0_1;
72         2'b01 : data <= clk_1;
73         2'b10 : data <= clk_10;
74         2'b11 : data <= clk_60;
75     endcase
76 end

```

```

77 dist_mem_gen_0 dist_mem_gen_0(
78     .a(data),
79     .spo(led));
80 endmodule

```

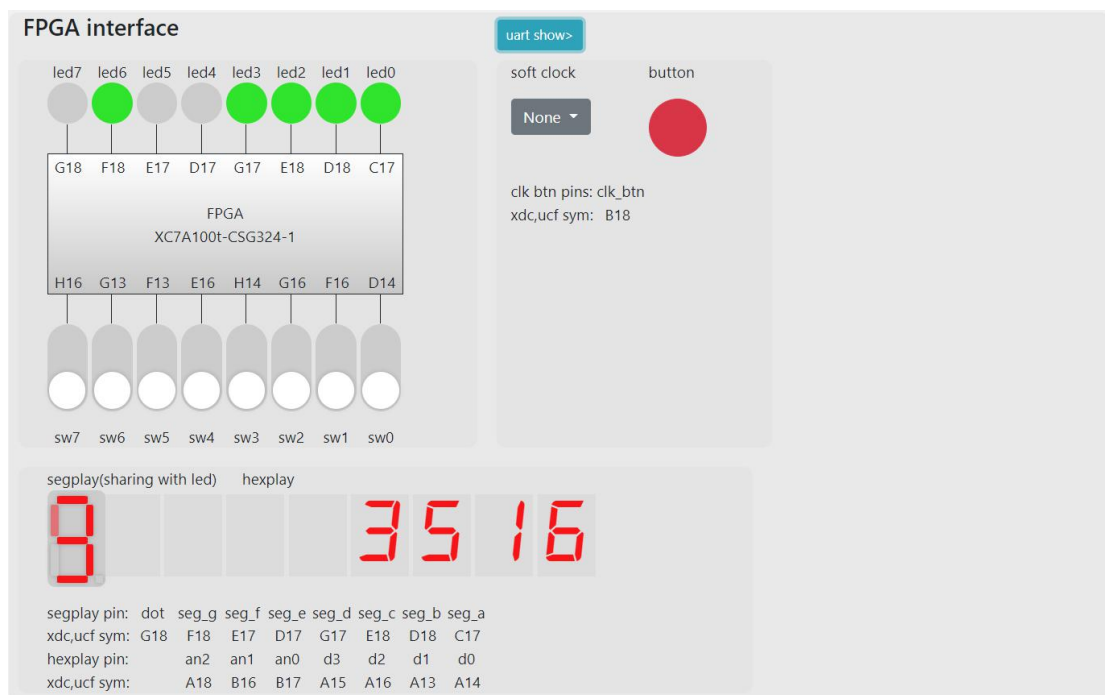
约束文件如下：


```

1  set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clk }];
2  set_property -dict { PACKAGE_PIN B18 IOSTANDARD LVCMOS33 } [get_ports { rst }];
3  ## leds
4  set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
5  set_property -dict { PACKAGE_PIN F18 IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
6  set_property -dict { PACKAGE_PIN E17 IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
7  set_property -dict { PACKAGE_PIN D17 IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
8  set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
9  set_property -dict { PACKAGE_PIN E18 IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
10 set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
11 set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
12
13 set_property -dict { PACKAGE_PIN A14 IOSTANDARD LVCMOS33 } [get_ports { data[0] }];
14 set_property -dict { PACKAGE_PIN A13 IOSTANDARD LVCMOS33 } [get_ports { data[1] }];
15 set_property -dict { PACKAGE_PIN A16 IOSTANDARD LVCMOS33 } [get_ports { data[2] }];
16 set_property -dict { PACKAGE_PIN A15 IOSTANDARD LVCMOS33 } [get_ports { data[3] }];
17 set_property -dict { PACKAGE_PIN B17 IOSTANDARD LVCMOS33 } [get_ports { an[0] }];
18 set_property -dict { PACKAGE_PIN B16 IOSTANDARD LVCMOS33 } [get_ports { an[1] }];
19 set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33 } [get_ports { an[2] }];

```

生成 bits 流在 FPGA 平台烧写可得到（一个瞬间的截图）



【总结与思考】

1. 学会了配置改变时钟快慢的 ip 核和 ROM & RAMip 核，编写了 coe 文件控制 ROM 生成 0 到 F 的数字。
2. 使用分时复用的方法来在 FPGA 平台显示多位的七段数码管。
3. 学会使用 100Mhz 的时钟以 count 计数的方式生成拥有新的频率的

时钟，并采取这种生成方式完成分时复用以及简易计时器的设计。

4. 实验难度：中上，主要是对 ip 核的使用比较茫然。

5. 任务量：中上，两个下午完成。

6. 改进建议：可以在实验步骤的参考代码部分加些许注释便于我们快速阅读和理解，这样可以加快我们学习新东西的速度而不是将时间利用在阅读代码上。