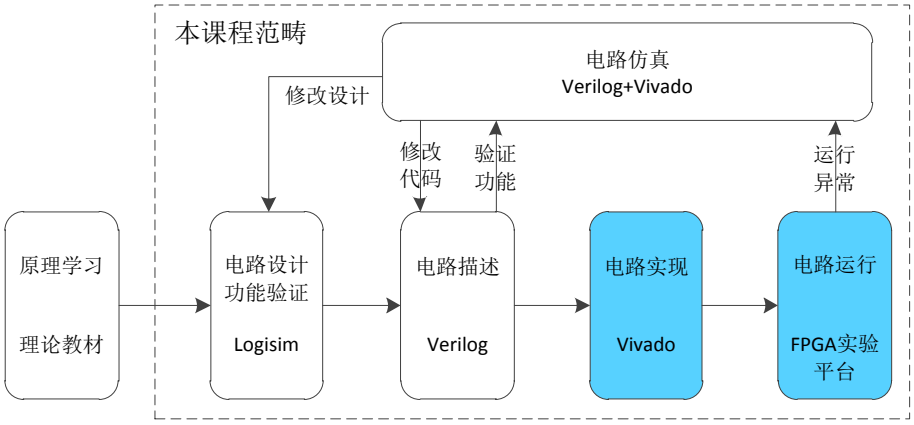
实验 07 FPGA 实验平台及 IP 核使用

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

学会查看原理图

理解 FPGA 开发各关键环节

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）

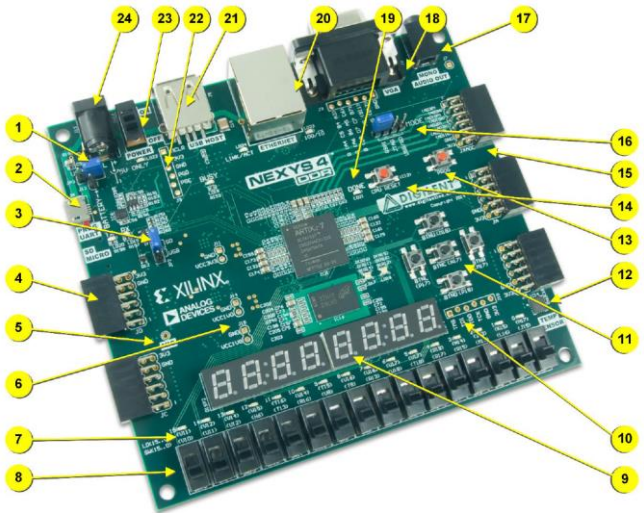
Logisim

vlab.ustc.edu.cn

实验步骤

Step1. Nexys4 DDR 开发板简介

我们对实验中使用的 Nexys4 DDR 开发板进行简单的介绍。



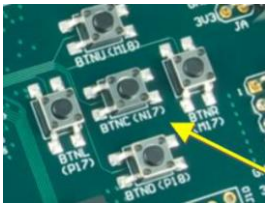
Nexys4 DDR 开发板采用 Xilinx 的 XC7A100TCSG324C-1 为核心芯片，配备了开关、按键、LED 灯、数码管等多种基本外设，还支持 USB、RJ45、VGA 等多种通用接口，以及五个用于扩展的 PMOD 接口，此外还放置了一颗 128MB 的 DDR 内存颗粒。用户不仅可以在该开发板上完成各种数字逻辑实验，还可以搭建一个完整的片上系统（SOC），运行操作系统，进行单片机或嵌入式系统的开发，功能非常强大。

下图为 Nexys4 开发板支持的外设列表。本系列实验课程中，主要用到开关、按键、LED、数码管、PMOD 等基本外设。在前面的实验中，我们实现了用开关控制 LED 灯的逻辑电路，下面我们将通过开发板的原理图讲解这几种外设的工作原理和连接关系，使读者能熟练的使用它们设计出各种功能电路。

Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod connector (XADC)
4	Pmod connector(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

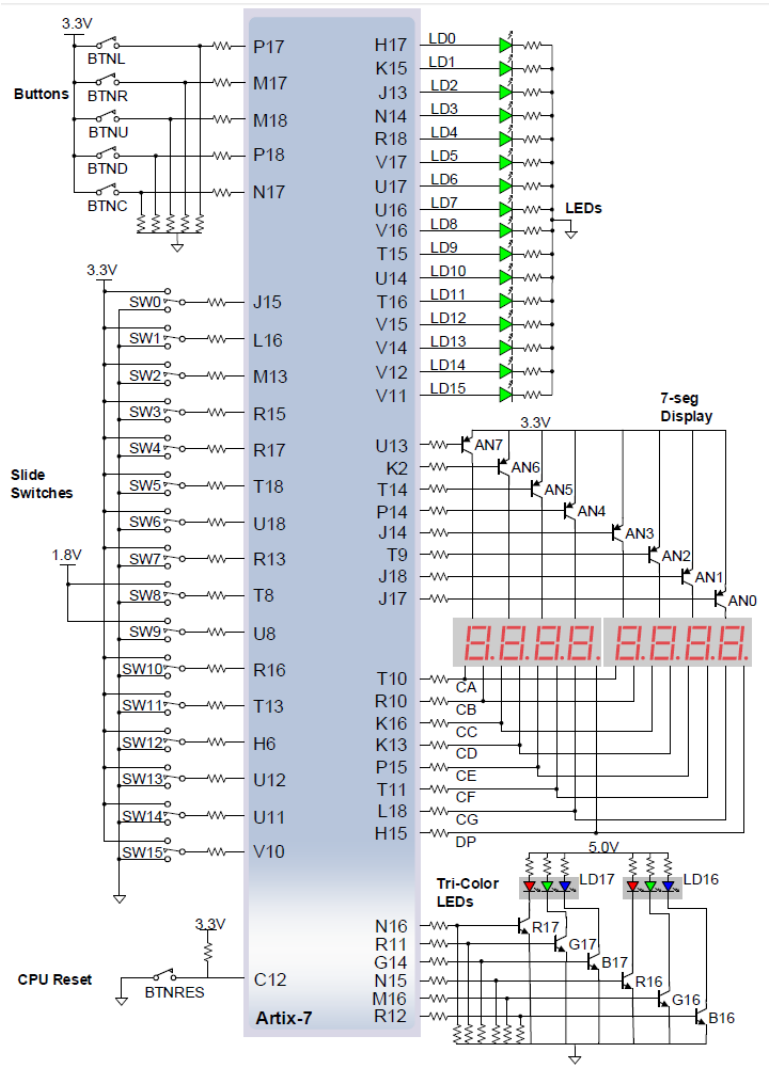
Step2. 开发板原理图介绍

Nexys4 DDR 开发板上通过丝印的形式标注了部分外设与 FPGA 管脚之间的连接方式，如下图所示，五个按键的下方都注明了对应的 FPGA 管脚。这种方式对于比较简单的外设比较使用，一旦引脚数量过多，就不适用了，另外也无法得知更具体的电路连接关系（如按键按下后输出高电平还是低电平）。



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入，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。

了解开发板结构、连接关系以及其它技术细节的最有效的途径是查看说明文档和电路原理图。下图是 Nexys4DDR 开发板说明文档中给出的 FPGA 与各种基本外设的连接关系。



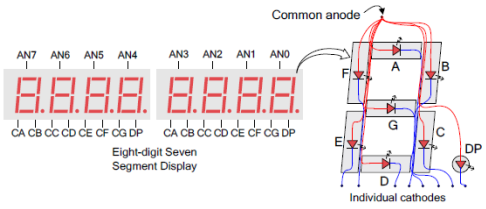
每个拨动开关与中学里讲到的单刀双置开关本质相同，有三个引脚，当开关推上去后，开关的中间引脚与电源相连，FPGA 管脚呈高电平，当开关拉下来后，开关的中间引脚与地相连，FPGA 管脚呈低电平。

五个按键通过下拉电阻与 FPGA 的五个通用管脚相连，当按键没有被按下时，FPGA 管脚通过下拉电阻与地相连，管脚呈低电平状态，当按键被按下后，接通高电平，管脚呈高电平状态。

Nexys4DDR 开发板上有 16 个 LED，每个 LED 都是 P 极接 FPGA 管脚，N 极通过限流电阻接地。当 FPGA 管脚输出高电平时，与其连接的 LED

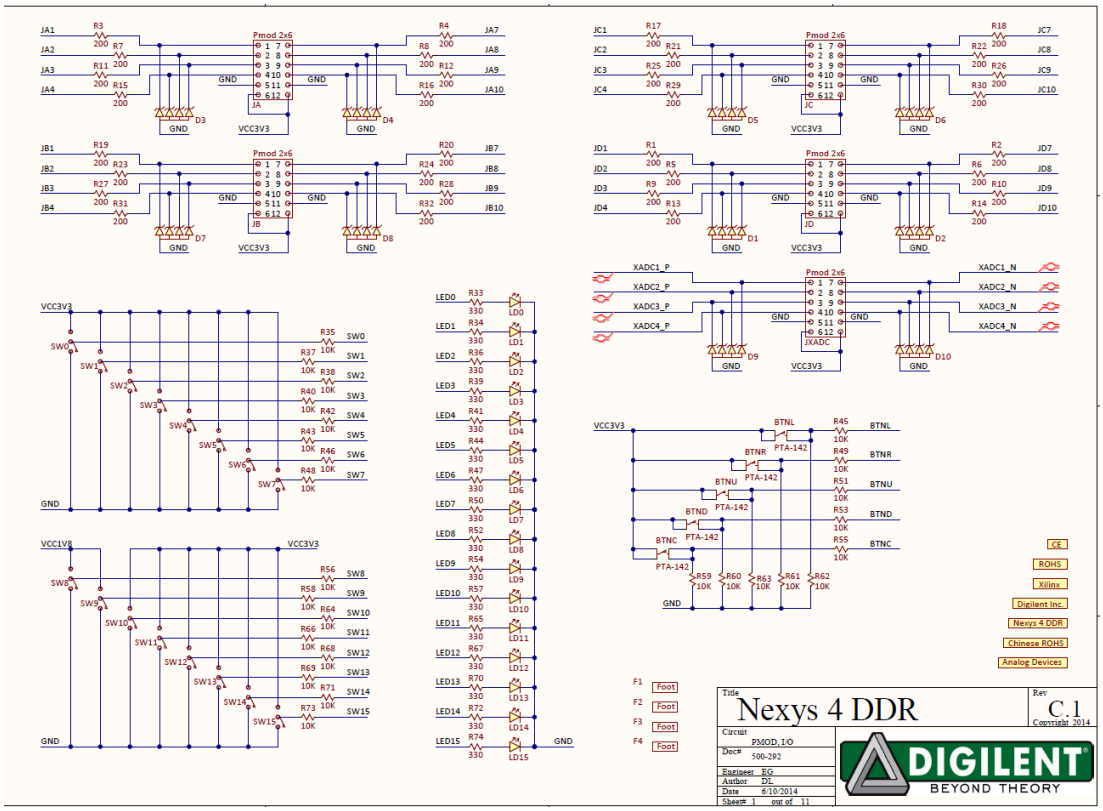
被点亮，当 FPGA 管脚输出低电平时，LED 熄灭。

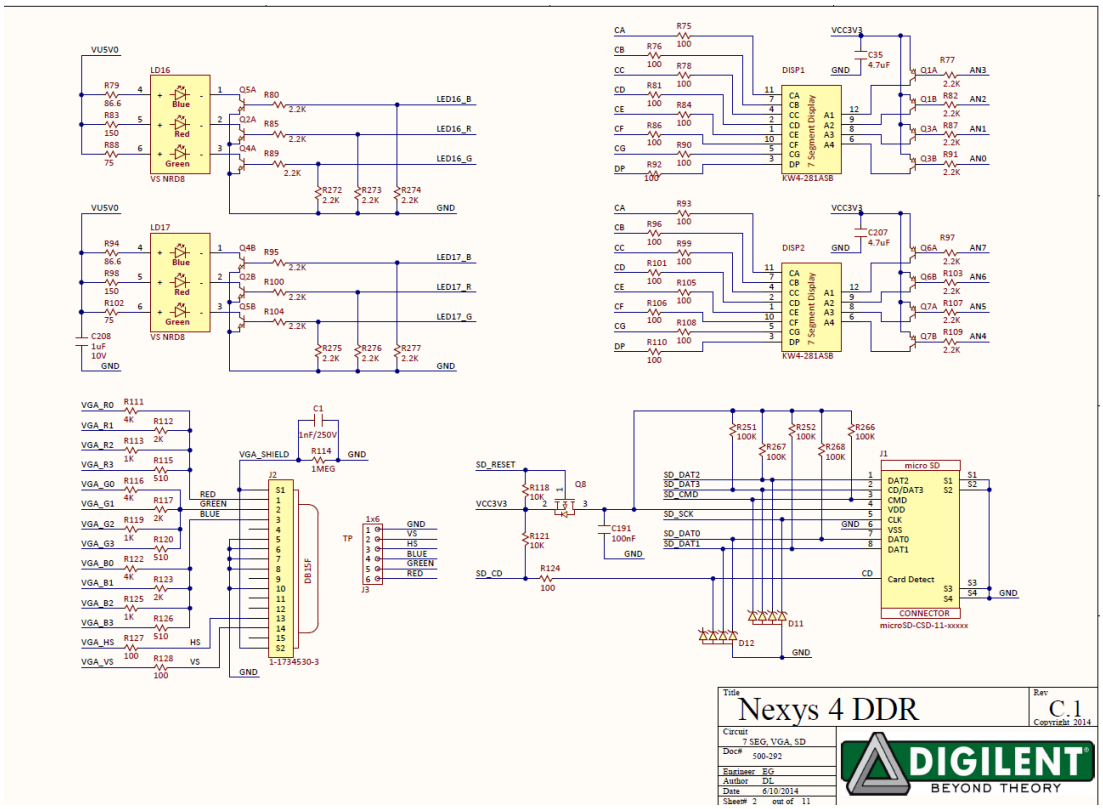
在 4 种基本外设中，数码管电路最为复杂。开发板上使用的是公阳极数码管，FPGA 管脚通过三极管放大电路控制数码管的位选择信号，每个管脚对应一个数码管，通过分析电路可以发现，当对应管脚为低电平时，三极管导通，数码管位选择信号有效。数码管的段选择信号是共用的，每个段选择信号都同时控制 8 个数码管对应的段，低电平有效。经分析可知，数码管的位选和段选信号均为低电平有效，同时为零时对应数码管的相应段点亮，否则不亮。

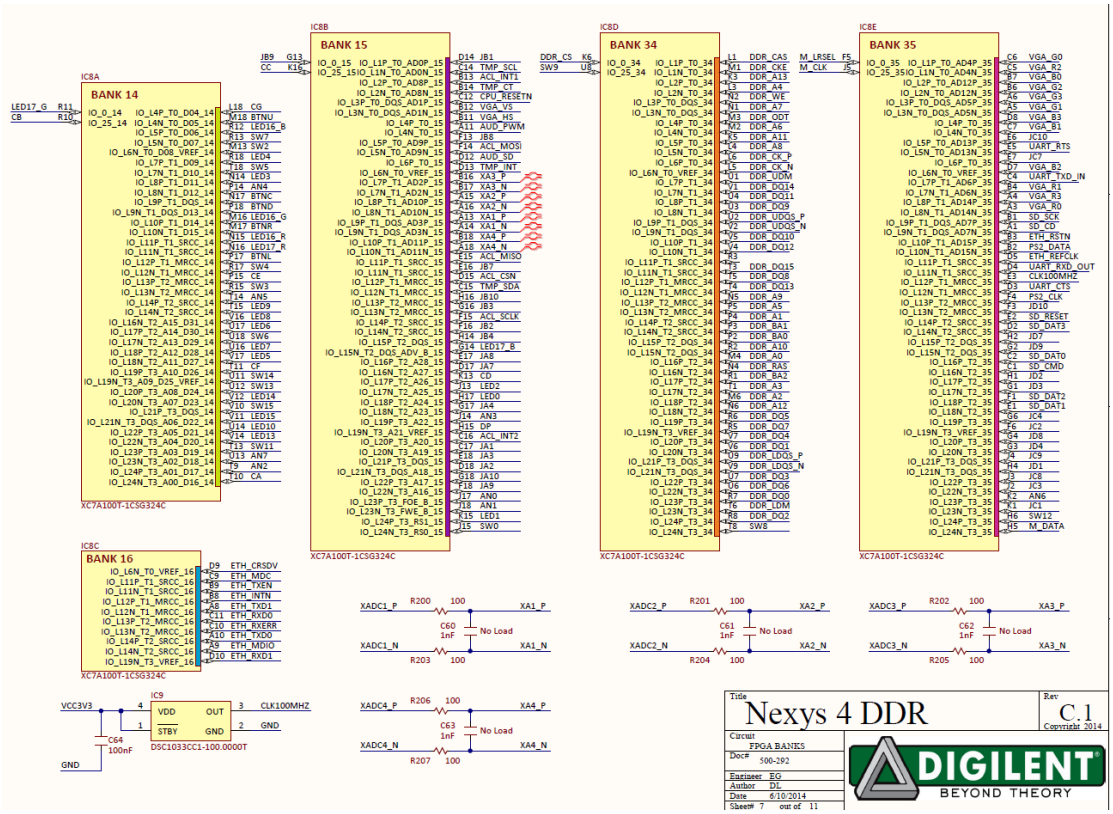


图中还有两个三色 LED，根据连接图可以看出，对应的 FPGA 管脚高电平有效。

下图是 Nexys4 DDR 开发板的部分原理图，包含了与 FPGA 芯片和基本外设相关的部分。通过查阅原理图，我们不仅可以了解到 FPGA 芯片与各种外设之间的连接关系，还能了解到一些更加细节的地方，

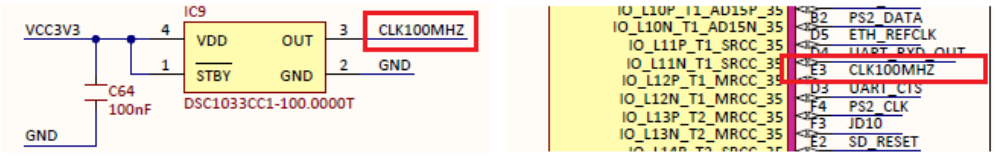

Pmod 2x6
JA
JB
JC
JD
JXADC
VCC3V3
GND
LED0
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
LED9
LED10
LED11
LED12
LED13
LED14
LED15
SW0
SW1
SW2
SW3
SW4
SW5
SW6
SW7
SW8
SW9
SW10
SW11
SW12
SW13
SW14
SW15
BTNL
BTNR
BTNU
BTND
BTNC
PTA-142
CE
ROHS
Xilinx
Digilent Inc.
Nexys 4 DDR
Chinese ROHS
Analog Devices
Foot
Nexys 4 DDR
C.1
Copyright 2014
PMOD, I/O
500-292
EG
DL
6/10/2014
1 out of 11
DIGILENT
BEYOND THEORY


VU5V0
LD16
LD17
Blue
Red
Green
VS NRD8
LED16_B
LED16_R
LED16_G
LED17_B
LED17_R
LED17_G
DISP1
DISP2
7 Segment Display
KW4-281ASB
AN0
AN1
AN2
AN3
AN4
AN5
AN6
AN7
VGA_R0
VGA_R1
VGA_R2
VGA_R3
VGA_G0
VGA_G1
VGA_G2
VGA_G3
VGA_B0
VGA_B1
VGA_B2
VGA_B3
VGA_HS
VGA_VS
VGA_SHIELD
RED
GREEN
BLUE
HS
VS
DB15F
1-1734530-3
SD_RESET
SD_DAT2
SD_DAT3
SD_CMD
SD_SCK
SD_DAT0
SD_DAT1
SD_CD
micro SD
Card Detect
CONNECTOR
microSD-CSD-11-xxxxx
Nexys 4 DDR
C.1
Copyright 2014
7 SEG, VGA, SD
500-292
EG
DL
6/10/2014
2 out of 11
DIGILENT
BEYOND THEORY



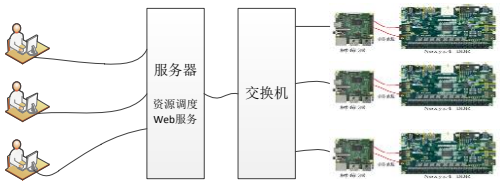
查看原理图时有几点需要特别注意，首先对于引脚数量较多的芯片，在画原理图时会按功能将其分成多个部分，以便于查看，如上图中的 FPGA 芯片，按供电、配置、通用 IO 等功能分成了 6 个部分。第二，简单的引脚连接关系可以通过连线表示，稍复杂一些的可以通过标签表示，如上面的原理图所示，其中一个拨动开关左侧标注了“SW0”的标签，FPGA 的 J15 管脚上（IC8B 最下方）也有同样的标签，说明该开关与 FPGA 的 J15 管脚是连接在一起的。

在上次的实验中，我们还用到了时钟信号，在 XDC 文件中可以看到，该信号被分配到了 E3 管脚，为什么要分配到这个管脚呢？我们通过查看原理图可以发现该 FPGA 管脚是与一个时钟芯片相连接的，该时钟芯片正常工作时能够提供一个频率为 100MHz 的时钟信号。



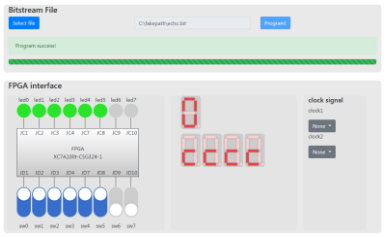
通过查阅原理图我们可以发现，开发板上的 FPGA 芯片的各个引脚都已经与确定的外设连接在一起，我们在使用 Vivado 进行开发时，电路模块分配应该与开发板一致。例如一个输出信号，可以将其分配给 LED 或者数码管，而不应该分配给按键，虽然这样做在 vivado 综合室不会有任何警告，但实际烧写到开发板上是无法正常工作的。为避免每次建立工程时都要通过原理图查看管脚对应关系，我们可以使用官方提供的 Nexys4DDR 的 XDC 文件，里面给出了板上 FPGA 芯片所有的引脚对应关系，用户使用时只需修改 XDC 文件内对应的信号名与自己模块内的端口名称一致即可。

Step3. FPGAOL 平台介绍



我们搭建了一套包含 20 个节点的远程 FPGA 实验平台，其结构如上图所示。每个节点包含一个 Nexys4DDR 开发板和一个树莓派单板，两者使用排线相连。树莓派对用户来说完全透明，负责与服务器端进行通信，完成烧录 FPGA，对 FPGA 管脚信号进行实时采样等功能。

用户通过网页登录系统，申请获取资源，获得设备节点后便可以上传并烧录自己的 bit 文件，并进行一定的交互性操作。



该平台包含开关、LED、七段数码管、十六进制动态扫描数码管等多种外设。此外，用户还可以查看各端口信号的采样波形。该平台操作简单，界面直观，用户不需要购置硬件设备便可以完成 FPGA 开发流程的学习。

Step4. 使用时钟管理单元 IP 核

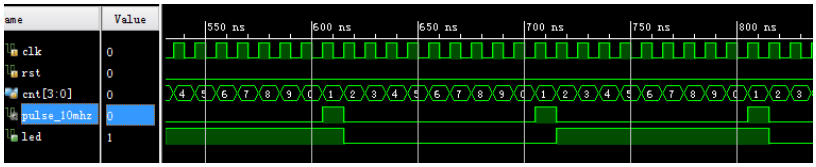
在 FPGA 开发中，有很多常用功能的模块是不需要自己开发的，用户可以复用第三方开发好的模块，这种模块被称为 IP 核。此处我们先学习时钟管理单元 IP 核的使用。

前面已经讲到，Nexys4DDR 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）

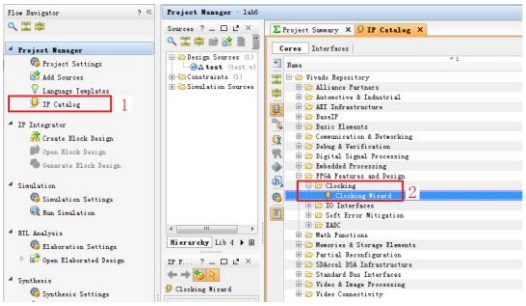
```
错误示例：assign   signal_1  = func(clk,xxx);
错误示例：always@(posedge signal_2) //signal_2 由 clk 生成
错误示例：always@(posedge clk or negedge clk) //两边沿不能同时使用
```

时钟信号只应该出现在 always 语句的时序控制部分

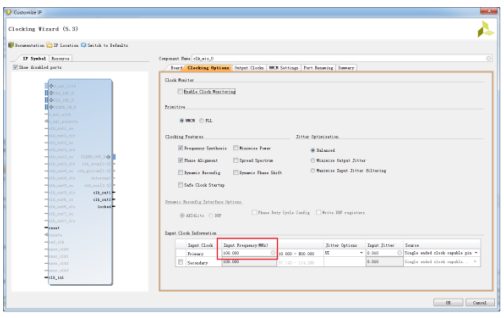
```
正确示例：always@(posedge clk) //时钟上升沿触发，同步复位或无复位
正确示例：always@(negedge clk) //时钟下降沿触发，同步复位或无复位
正确示例：always@(posedge clk or posedge rst) //异步复位，高有效
正确示例：always@(posedge clk or negedge rst_n) //异步复位，低有效
```

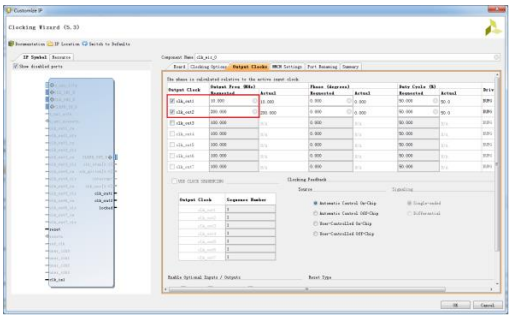
如果设计中确实需要不同频率的时钟信号应该通过时钟管理单元 IP 核生成。首先，点击“IP catalog”，在对应窗口中选中“Clocking

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。

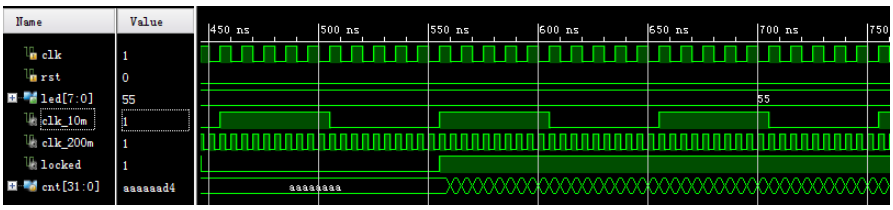
```
module test(
input                clk,
input                rst,
output       [7:0]   led);
wire         clk_10m,clk_200m,locked;
reg  [31:0] cnt_1,cnt_2;
always@(posedge clk_200m)
begin
```

```
    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 Nexys4DDR 开发板，我们将 LED 信号分配到最右侧的 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 N17   IOSTANDARD LVCMOS33 } [get_ports { rst }];
## leds
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
```

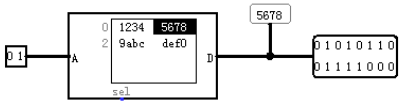
需要特别指出的是，时钟管理单元并不能产生任意频率的时钟信号，以前面为例，输入时钟为 100MHz 时，输出时钟只能是频率范围在 4.687MHz~800MHz 之间的某些频率，如 4MHz 超出范围无法产生，799MHz 虽然在频率范围内，但因精度问题，只能产生以及频率相近的时钟信号，有兴趣的读者可自行验证。

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。

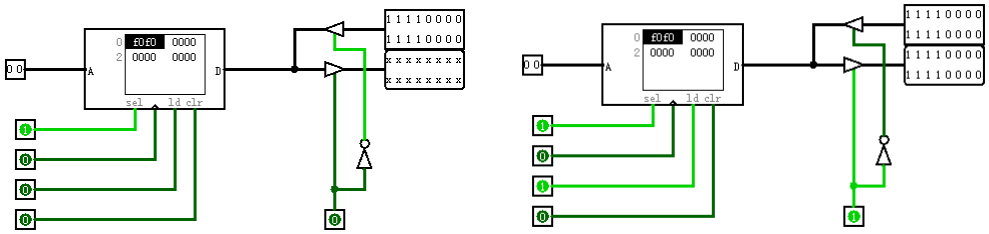
Step5. 使用片内存储单元

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 $2^{\text{地址位宽}}$ 个存储单元，每个单元里存储了与数据端口相同位宽的数

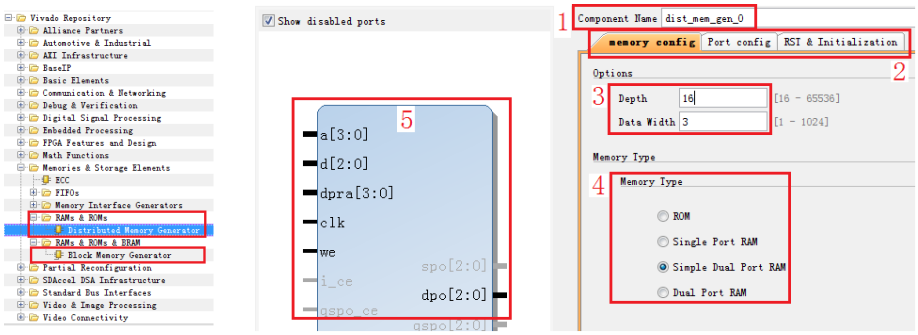
据，存储器电路会将与地址相对应单元的数据呈现在数据端口上，下图是一个包含 2^2=4 个存储单元的 ROM，数据位宽为 16 位。



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。



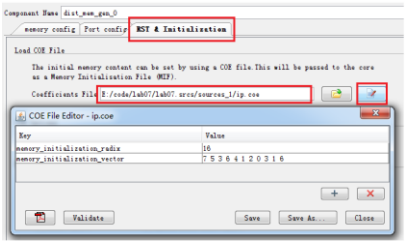
Vivado 中也提供了存储器相关的 IP 核，种类和接口类型比 Logisim 中更加丰富，我们通过实际的例子来学习一下。IP 核目录中提供了两种存储器实现方式:“Distributed Memory”和“Block Memory”两种，我们以“Distributed Memory”为例进行介绍。



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;
```

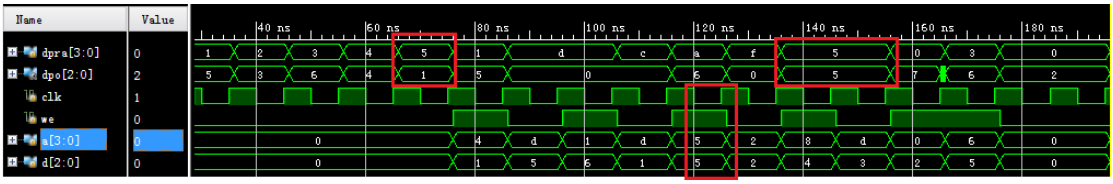
例化完成后，可以在“工程路径\工程名.ip_user_files\ip\dist_mem_gen_0\”目录下找到例化的文件，我们可以像调用其它模块一样调用该模块，编写仿真文件，对该 RAM 进行仿真，仿真文件代码为：

```
`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，输出端口连接到七段数码管上（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

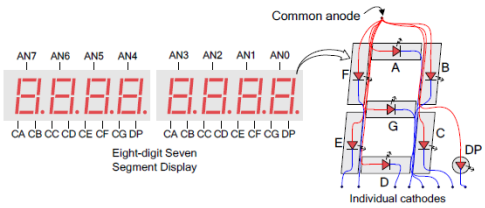
2. 请评价本次实验的难易程度

3. 请评价本次实验的任务量

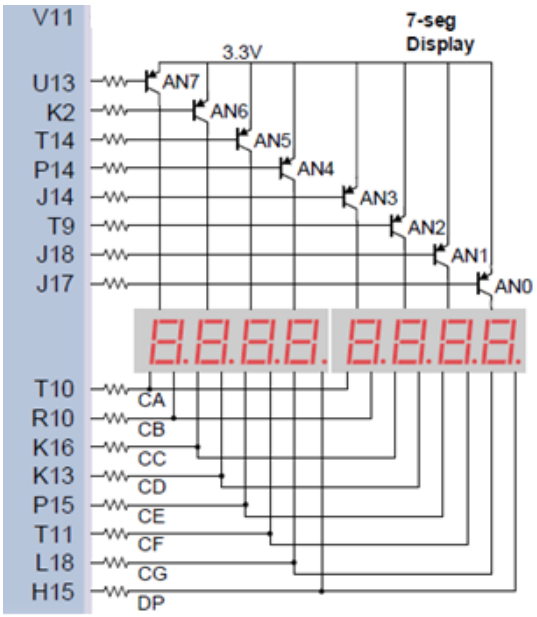
4. 请为本次实验提供改进建议

补充说明：

七段数码管工作原理及参考代码



7 段数码管分为共阴极数码管和共阳极数码管两种，上图数码管为共阳极结构，对于共阳极数码管，每个数码管有一个共同的阳极（Common anode，又称为位选信号），位选信号为高电平时，数码管才可以显示图案或数字，具体显示的内容由 8 个段选信号（CA、CB...CG、DP）控制，段选信号低电平有效。



由原理图可以看出，Nexys4DDR 开发板中采用的是多位数码管的结构，

共包含 8 个七段数码管（称为 8 个位），每个数码管都有单独的位选信号（AN0、AN1...AN7），所有数码管共用段选信号（CA、CB...CG、DP），FPGA 芯片通过 16 个输出管脚对这些数码管进行控制，并且都是低电平有效。

下面，我们将一步步深入介绍数码管的工作原理。

step1.首先是用开关直接控制数码管位选、段选信号，如下代码所示：

```
module example1(
input [15:0] sw,
output [7:0] an,
output [7:0] seg);
    assign an  = sw[15:8];
    assign seg = sw[7:0];
endmodule
```

我们用板上的 16 个开关控制数码管，综合并烧写后，拨动开关并观察数码管的亮灭，可以增加我们对数码管工作原理的理解。

step2.接着，我们继续使用左侧的 8 个开关控制数码管的位选信号，通过最右侧的 4 个开关，经过译码逻辑后控制数码管的段选信号，请阅读如下代码，并将其补充完整，使数码管能正常显示 0~F 所有的数字。

```
module example2(
input       [15:0] sw,
output      [7:0] an,
output reg  [7:0] seg);
assign an  = sw[15:8];
always@(*)
begin
    case(sw[3:0])
        4'h0: seg = 8'b0000_0011; //CA CB CC CD CE CF CG DP
        4'h1: seg = 8'b1001_1111; //CA CB CC CD CE CF CG DP
        ...                       //to be coding
        4'hF: seg = 8'b0111_0001; //CA CB CC CD CE CF CG DP
        default:
```

```
    endcase
end
endmodule
```

step3.最后，我们实现动态扫描功能，以达到多个数码管上能同时显示不同数值的效果，

```
module example3(
input       clk,rst;
input       [15:0] sw,
output      [7:0] an,
output      [7:0] seg);
reg [1:0] cnt;

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

always@(posedge clk) //时钟频率应该 1KHz 左右，太高太低都不好
begin
    if(rst) cnt <= 2'h0;
    else    cnt <= cnt + 2'b1;
end
always@(posedge clk) //分时复用
begin
    case(cnt)
        2'h0: an <= 8'b1111_1110;
        2'h1: an <= 8'b1111_1101;
        2'h2: an <= 8'b1111_1011;
        2'h3: an <= 8'b1111_0111;
    endcase
end
always@(posedge clk)
begin
    case(cnt)
        2'h0: data <= sw[3:0];
        2'h1: data <= sw[7:4];
        2'h2: data <= sw[11:8];
        2'h3: data <= sw[15:12];
    endcase
end
endmodule

module decode(          //译码模块
```

```
input       [3:0] data,
output reg  [7:0] seg);
always@(*)
begin
    case(data[3:0])
        4'h0: seg = 8'b0000_0011; //CA CB CC CD CE CF CG DP
        4'h1: seg = 8'b1001_1111; //CA CB CC CD CE CF CG DP
        ...                       //to be coding
        4'hF: seg = 8'b0111_0001; //CA CB CC CD CE CF CG DP
        default:
    endcase
end
endmodule
```

step4. 在前面的步骤中我们实现了数码管的动态扫面功能，但是要求扫面切换频率为1KHz左右,如果直接用板载的100MHz时钟来驱动，因为扫描频率太快,显示效果反而不好,有兴趣的读者可以实验一下。那如何用一个非常高频的时钟来产生一个较低频率的扫描切换信号呢？这里提供两种实现思路供读者参考。

第一种，采用多bit计数器的高位来作为扫描切换控制信号，如下面的代码所示，这种方式的优点在于实现简单。

```
reg [19:0] cnt;
always@(posedge clk) //100MHz时钟
begin
    if(rst) cnt <= 20'h0;
    else    cnt <= cnt + 20'b1;
end
always@(posedge clk) //分时复用
begin
    case(cnt[19:18])
        2'h0: an <= 8'b1111_1110;
        ...
        2'h3: an <= 8'b1111_0111;
    endcase
end
```

第二种，使用脉冲信号进行控制，先用一个多bit的计数器（cnt）生成一个较低频率周期的脉冲信号（pulse），然后将此脉冲信号作

为使能信号，生成一个以较低频率计数的计数器（scan_cnt），此信号将控制数码管的动态扫描频率。该方法优点在于时序控制更加精准，脉冲信号的周期精度可以精确到一个时钟周期。

```
reg [19:0]  cnt;
reg [1:0]   scan_cnt;
wire        pulse;
always@(posedge clk) //
begin
    if(rst) cnt <= 20'h0;
    else if(cnt==20'hxx) cnt <= 0;
    else    cnt <= cnt + 20'b1;
end
assign pulse = (cnt==20'hxxx) ? 1'b1 : 1'b0;
always@(posedge clk) //pulse作为计数使能信号
begin
    if(rst)          scan_cnt <= 2'h0;
    else if(pulse)   scan_cnt <= scan_cnt + 2'b1;
end
always@(posedge clk) //分时复用
begin
    case(scan_cnt[19:18])
        2'h0: an <= 8'b1111_1110;
        ...
        2'h3: an <= 8'b1111_0111;
    endcase
end
```