

实验一 用 Multisim 设计全加器和八位比较器

一、实验目的

1. 用 Multisim 设计全加器
 - (1) 给出逻辑表达式
 - (2) 截图保存电路图
 - (3) 拍照片, 表示运行结果
2. 4 位比较器扩展为 8 位比较器
 - (1) 给出逻辑表达式
 - (2) 截图保存电路图
 - (3) 拍照片, 表示运行结果

二、实验环境

MultiSim:

MultiSim14 是一种电子设计自动化软件, 可以模拟和分析电路设计。它可以帮助电子工程师在设计电路时预测电路行为, 避免在实际硬件中出现的问题。MultiSim 具有易于使用的界面, 可以从各种库中选择元件, 如电阻器、电容器和晶体管等。用户可以绘制电路图, 并使用模拟器来验证电路的性能, 例如输出波形、功率和幅度等。MultiSim 还可以帮助用户进行信号分析和频谱分析, 以便更好地理解电路的工作原理。除了模拟电路设计, MultiSim 还提供了虚拟仪器, 如示波器、函数发生器和数字万用表等, 这些可以帮助用户在模拟电路时收集和分析数据。总之, MultiSim 是一个功能强大的工具, 可用于各种电路设计和分析任务。

Nexys4 DDR:

Nexys 4 DDR 是一种高性能的 FPGA 开发板, 适用于数字系统设计和嵌入式处理器开发。该板配备了 Xilinx Artix-7 FPGA 芯片, 具有大容量、高速和低功耗等优点。此外, 该开发板还配备了 DDR3 SDRAM、Flash 存储器、USB 接口、以太网接口和多种传感器, 以及其他许多实用的外设。Nexys 4 DDR 支持多种开发环境和语言, 如 Vivado、Verilog、VHDL 和 C/C++ 等。此外, 它还提供了许多教学资源和实验, 可帮助学生和工程师更快地学习和理解数字系统设计和嵌入式开发。Nexys 4 DDR 开发板特别适合用于数字信号处理、通信系统、多媒体处理、机器视觉和人工智能等领域的应用。

三、实验原理

全加器的设计

一位二进制加法器有三个输入量：两个二进制数字 A_i 、 B_i 和一个低位的进位信号 C_i ，这三个值相加产生一个和输出 S_i 以及一个向高位的进位输出 C_{i+1} ，这种加法单元称为全加器，其逻辑方程如下：

$$S_i = A_i \oplus B_i \oplus C_i$$
$$C_{i+1} = A_i B_i + B_i C_i + C_i A_i$$

四位比较器扩展为八位比较器

用来完成两组二进制数大小比较的逻辑电路，称为数据比较器。下图 6.17 给出了 4 位二进制数据比较器的功能框图。 $a_3 a_2 a_1 a_0$ (a_3 为高位) 是一组二进制数输入， $b_3 b_2 b_1 b_0$ (b_3 为高位) 是另一组二进制数输入。 $a > b$, $a = b$, $a < b$ 为级联输入， $A > B$, $A = B$, $A < B$ 为比较结果输出。真值表为：



图 6.17 4 位二进制数比较器

可以用两片 4 位比较器级联而成 8 位比较器。此时低 4 位和高 4 位输入信号，分别加到两个比较器的输入端。低 4 位比较器的三个输出分别对应接到高 4 位比较器的三个级联输入端 $a > b$, $a < b$, $a = b$ ，比较结果由高 4 位比较器输出端输出即可。

逻辑表达式为：

$$F = (\overline{A_0 \oplus B_0}) + (A_1 \oplus B_1) + (A_2 \oplus B_2) + (A_3 \oplus B_3) + (A_4 \oplus B_4) + (A_5 \oplus B_5) + (A_6 \oplus B_6) + (A_7 \oplus B_7)$$

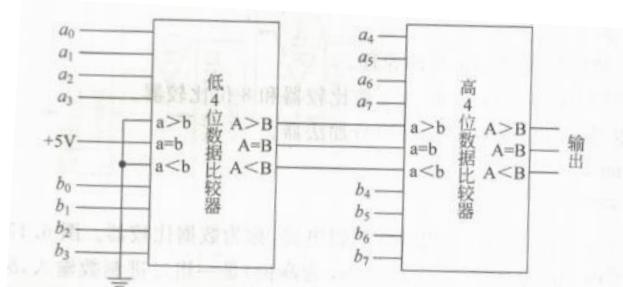
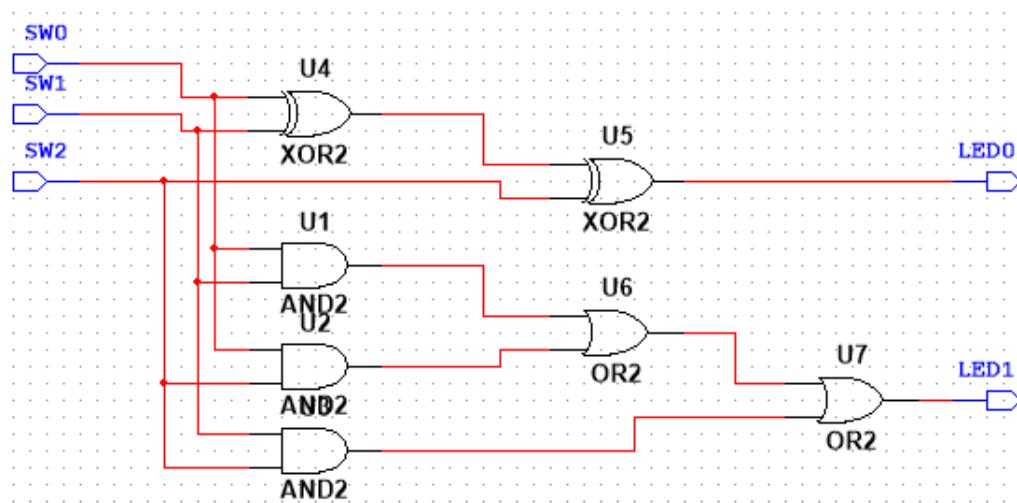


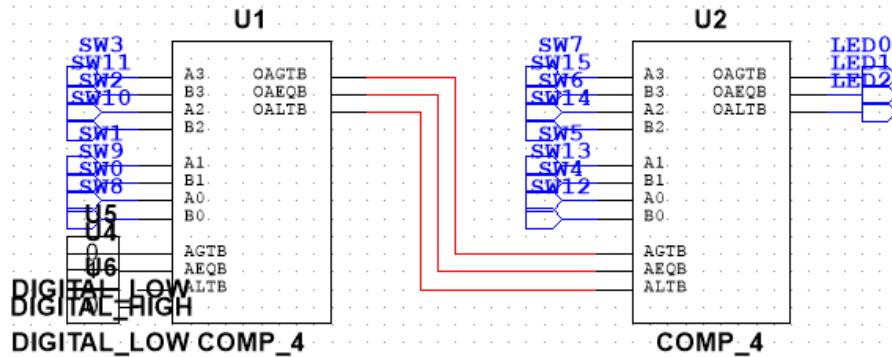
图 6.18 使用两片 4 位比较器组成 8 位比较器

四、 实验内容

全加器的设计的线路图

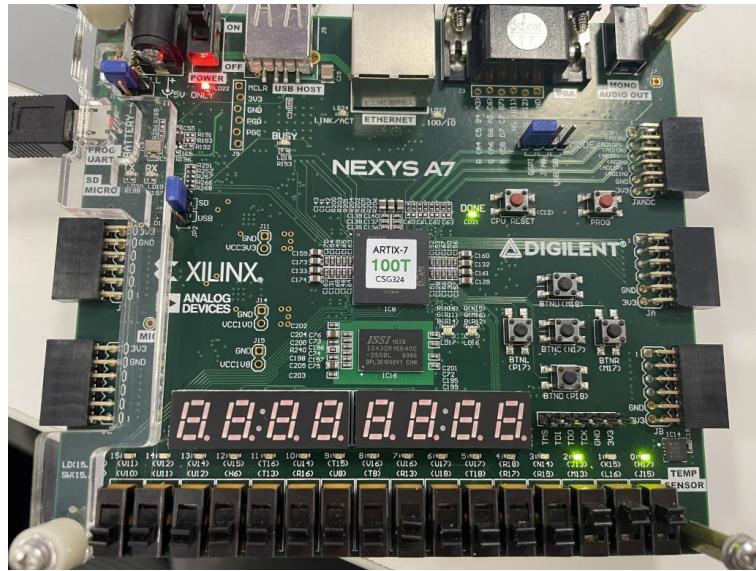


四位比较器扩展为八位比较器

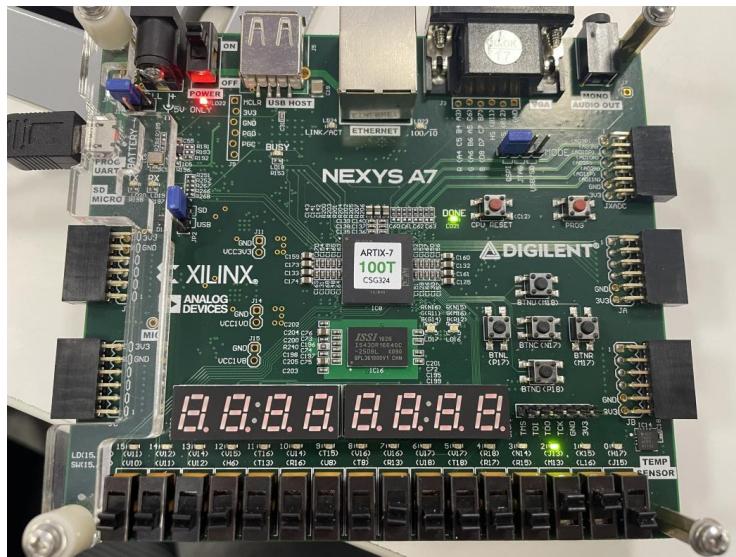


五、实验结果

全加器的设计

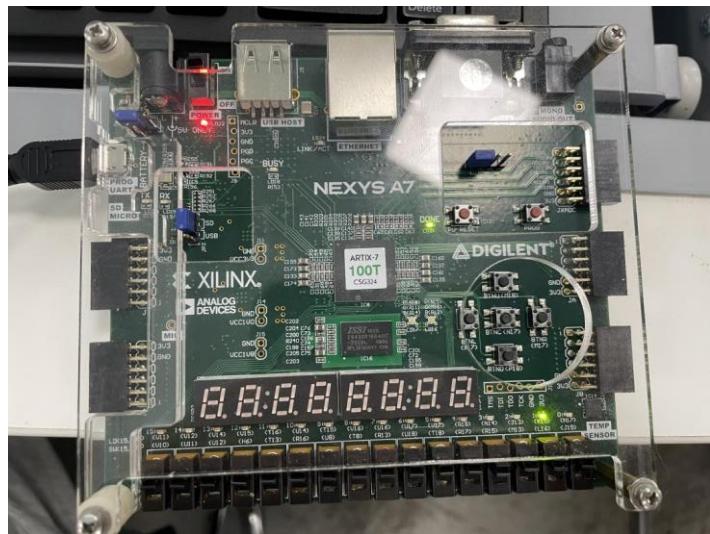


示例 1：当输入 ABC 为 111 时，输出 S 和 Ci 为 11

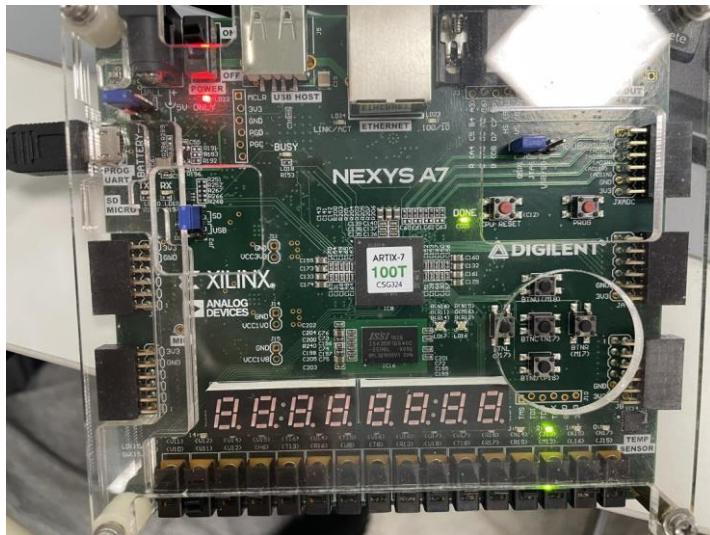


示例 2：当输入 ABC 为 011 时，输出 S 和 Ci 为 01

四位比较器扩展为八位比较器



示例 1：当 A 为 11111111 B 为 11111111 时 输出为 LED1=1，表示 A=B



示例 2：当 A 为 11101111 B 为 11111101 时 输出为 LED2=1，表示 A<B

实验二 用 Multisim 调整计数器频率

并将 4 位计数器，扩展到 8 位

一、 实验目的

本实验利用 MultiSim 软件作电路图，实现：

1) 调整计数器频率

- ①降低 2 倍
- ②降低 4 倍
- ③提高 2 倍
- ④提高 4 倍

2) 4 位计数器扩展到 8 位计数器

- ①频率提高 5 倍
- ②频率提高 10 倍

二、 实验环境

MultiSim:

MultiSim14 是一种电子设计自动化软件，可以模拟和分析电路设计。它可以帮助电子工程师在设计电路时预测电路行为，避免在实际硬件中出现的问题。MultiSim 具有易于使用的界面，可以从各种库中选择元件，如电阻器、电容器和晶体管等。用户可以绘制电路图，并使用模拟器来验证电路的性能，例如输出波形、功率和幅度等。MultiSim 还可以帮助用户进行信号分析和频谱分析，以便更好地理解电路的工作原理。除了模拟电路设计，MultiSim 还提供了虚拟仪器，如示波器、函数发生器和数字万用表等，这些可以帮助用户在模拟电路时收集和分析数据。总之，MultiSim 是一个功能强大的工具，可用于各种电路设计和分析任务。

Nexys4 DDR:

Nexys 4 DDR 是一种高性能的 FPGA 开发板，适用于数字系统设计和嵌入式处理器开发。该板配备了 Xilinx Artix-7 FPGA 芯片，具有大容量、高速和低功耗等优点。此外，该开发板还配备了 DDR3 SDRAM、Flash 存储器、USB 接口、以太网接口和多种传感器，以及其他许多实用的外设。

Nexys 4 DDR 支持多种开发环境和语言，如 Vivado、Verilog、VHDL 和 C/C++ 等。此外，它还提供了许多教学资源和实验，可帮助学生和工程师更快地学习和理解数字系统设计和嵌入式开发。Nexys 4 DDR 开发板特别适合用于数字信号处理、通信系统、多媒体处理、机器视觉和人工智能等领域的应用。

三、 实验原理

计数器

计数器是一种电子电路，用于计数输入脉冲的数量。它通常由多个触发器组成，每个触发器都有一个时钟输入和一个输出。每次时钟输入触发器时，计数器会将其内部状态加 1。当计数器达到最大值时，它会重新从零开始计数。计数器的实现原理可以通过一个简单的二进制计数器来说明。例如，一个 4 位二进制计数器可以由 4 个 D 触发器组成。每个 D 触发器都有一个时钟输入和一个数据输入。当时钟输入为高电平时，数据输入的值会被存储到 D 触发器的输出上。计数器的输出则是所有 D 触发器的输出串联在一起。每次时钟输入都会使计数器的状态加 1，从 0000 到 1111，然后重新从 0000 开始计数。

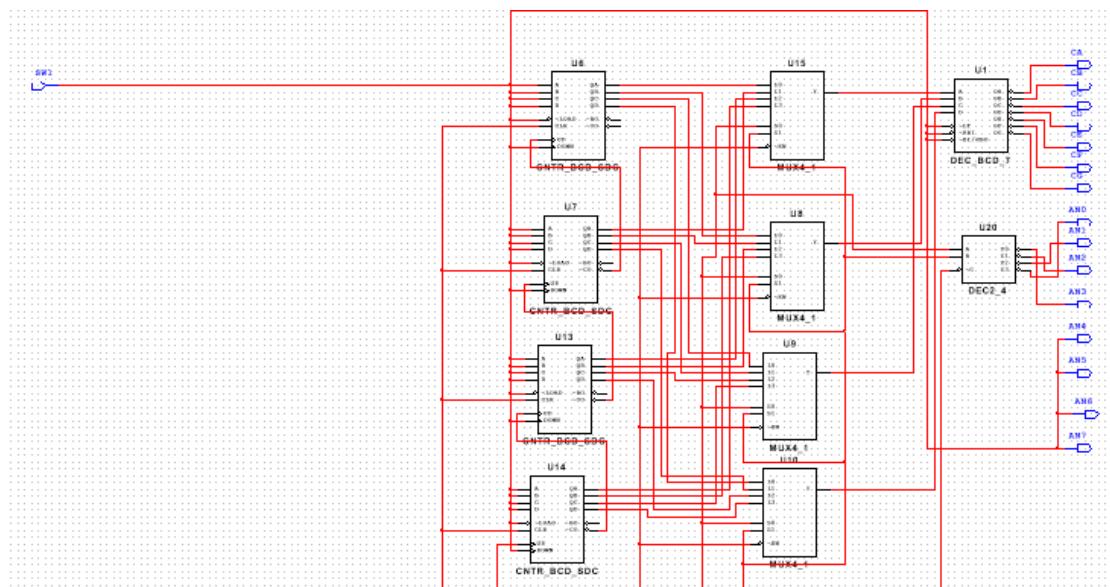
分频器

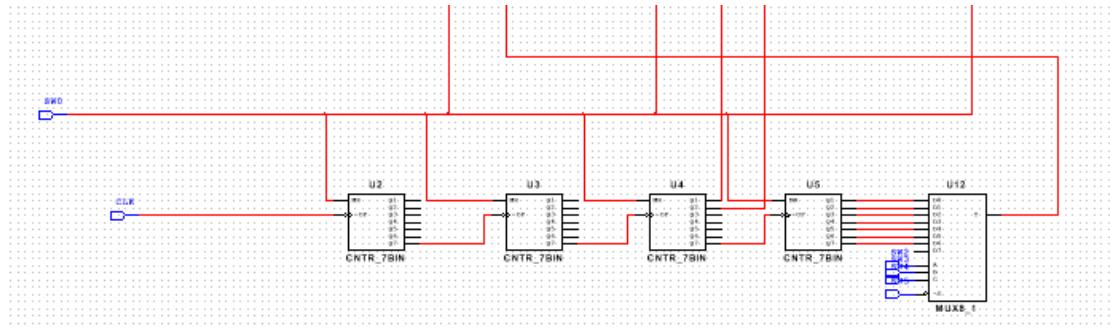
分频器是一种电子电路，用于将输入信号的频率降低到一个较低的值。它的实现原理可以通过一个简单的二分频器来说明。例如，一个 2 分频器可以将输入频率除以 2。这可以通过将输入信号通过一个 D 触发器并将其输出反馈到 D 触发器的数据输入上来实现。当时钟输入为高电平时，D 触发器的输出将变为与其上一个时钟周期相同的值，从而将输入信号除以 2。对于更高分频比，可以通过级联多个分频器来实现。分频器常用于时钟信号的分频、计数器的控制以及频率测量等方面。

本次实验只需要在已有的实验基础上进行修改即可，主要修改为，改变计数上限，可以通过增加计数器和选择器来完成。具体修改见实验内容部分。

四、 实验内容

根据输入来修改计数器的频率





增加八选一计数器，初始接入为 Q3

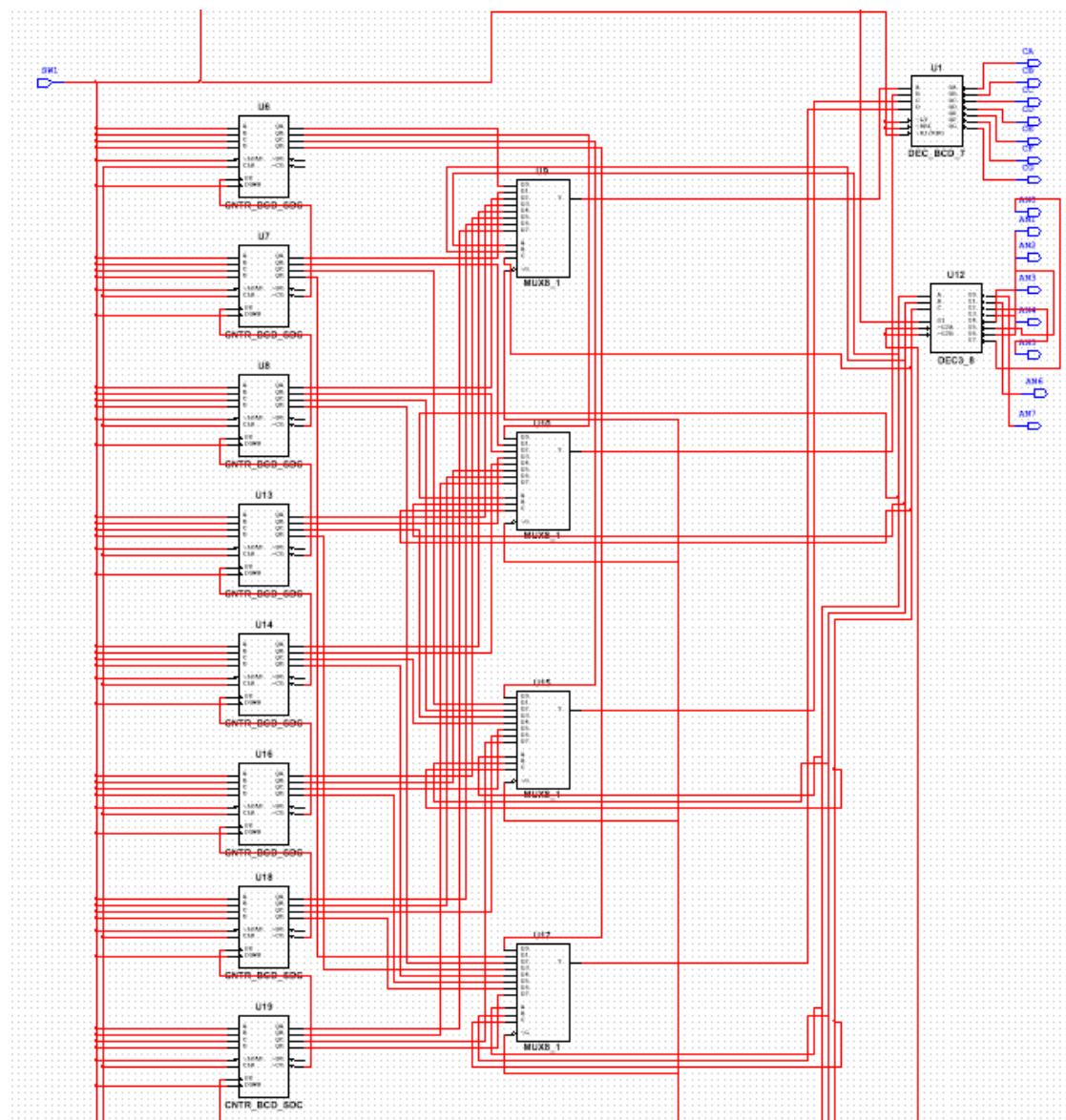
若要实现频率降低 2 倍，则只需要接入 Q2，输入信号为 001

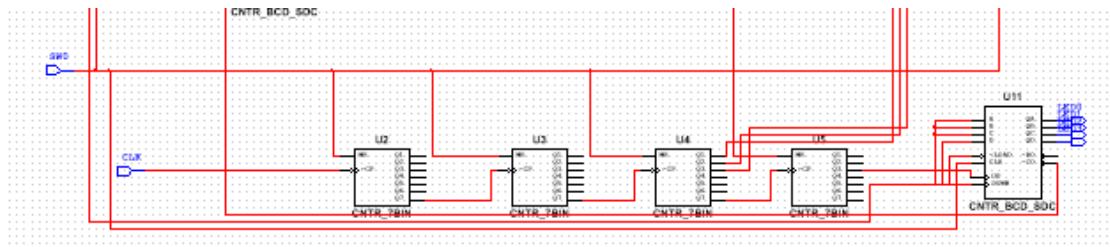
若要实现频率降低 4 倍，则只需要接入 Q1，输入信号为 000

若要实现频率提高 2 倍，则只需要接入 Q4，输入信号为 011

若要实现频率提高 4 倍，则只需要接入 Q5，输入信号为 100

将四位计数器拓展为八位计数器并调整频率



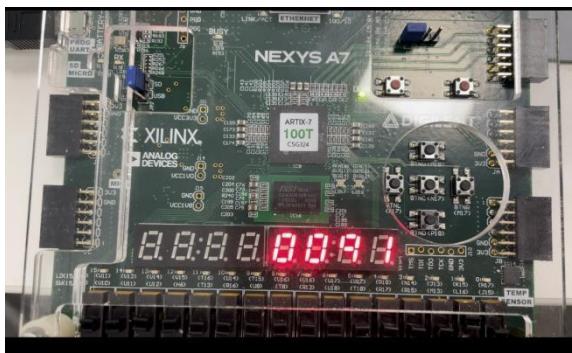
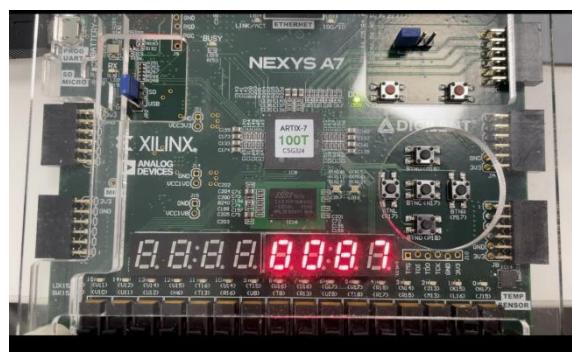


通过增加模十计数器使得频率下降十倍，
若要实现提高十倍则只需将模十计数器去除即可
接入模十计数器并接入 Q2 则可以是频率提高五倍

五、 实验结果

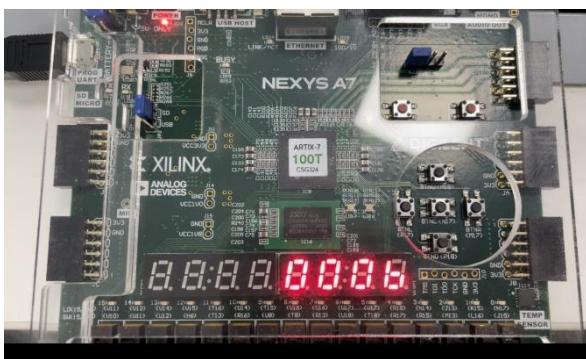
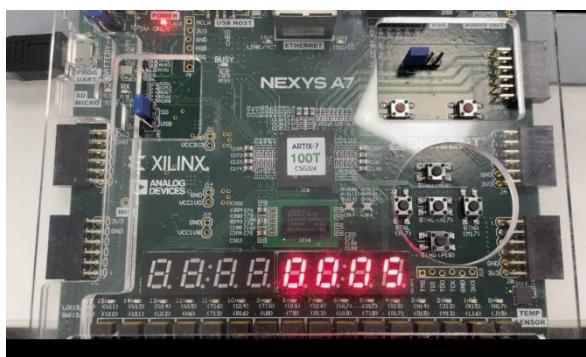
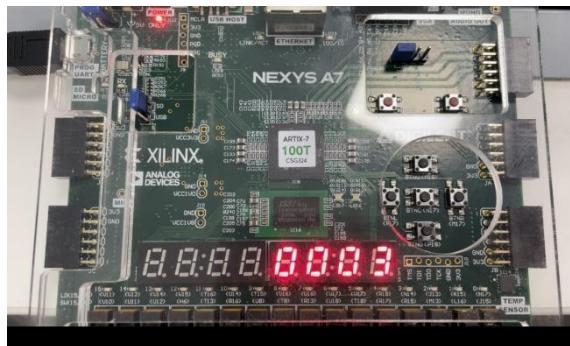
根据输入来修改计数器的频率

降低 2 倍频率



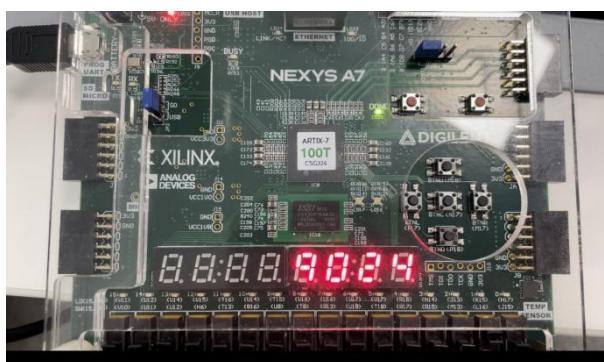
(第 4, 6, 8 秒截屏)

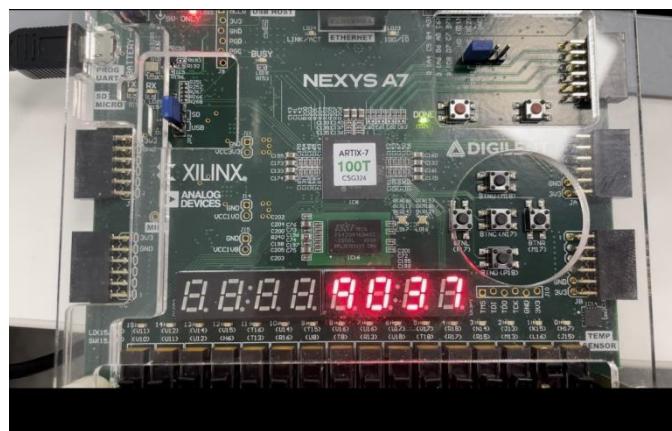
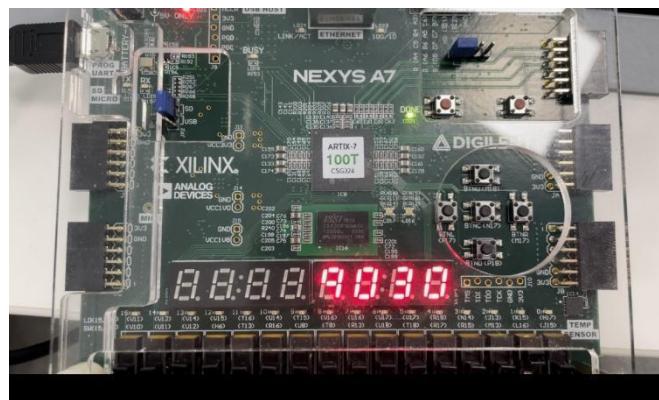
降低 4 倍频率



(第 4, 6, 8 秒截屏)

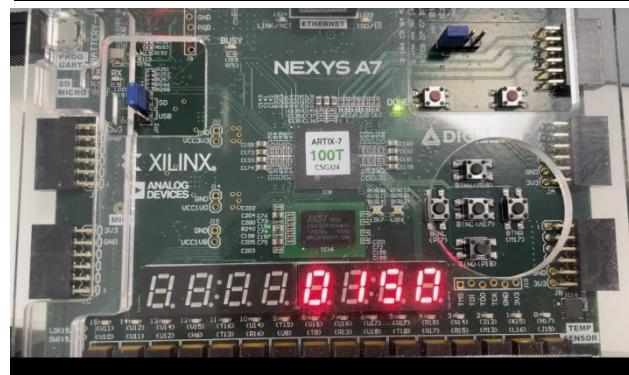
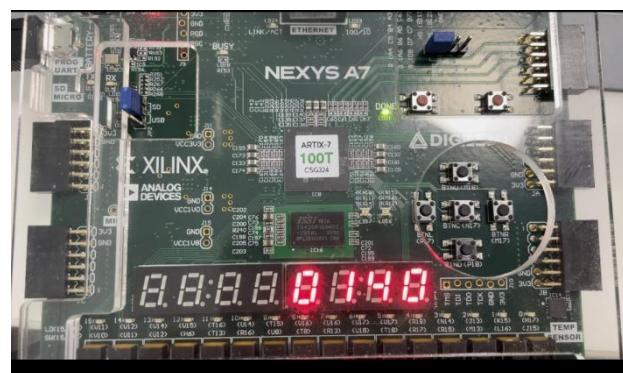
提高 2 倍频率

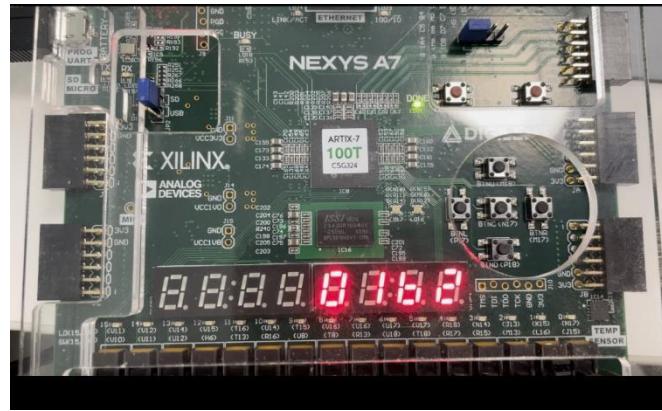




(第 4, 6, 8 秒截屏)

提高 4 倍频率

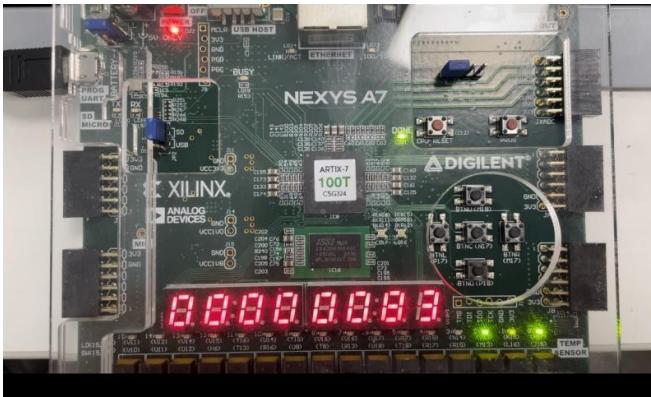
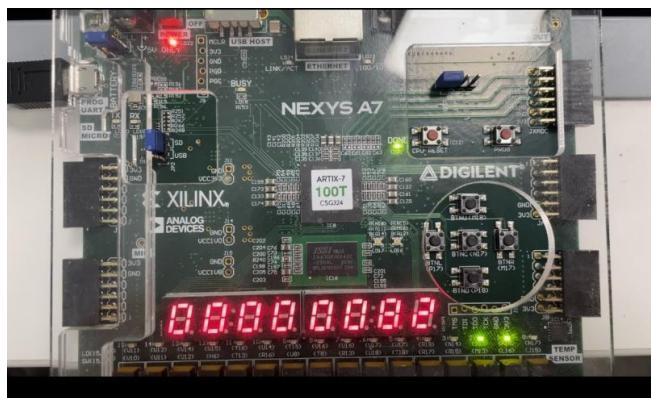
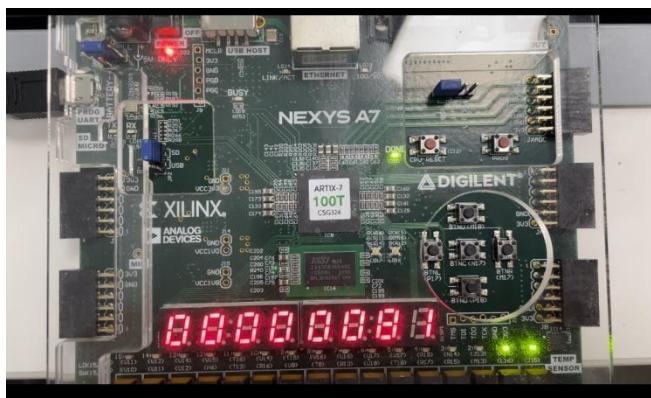




(第 4, 6, 8 秒截屏)

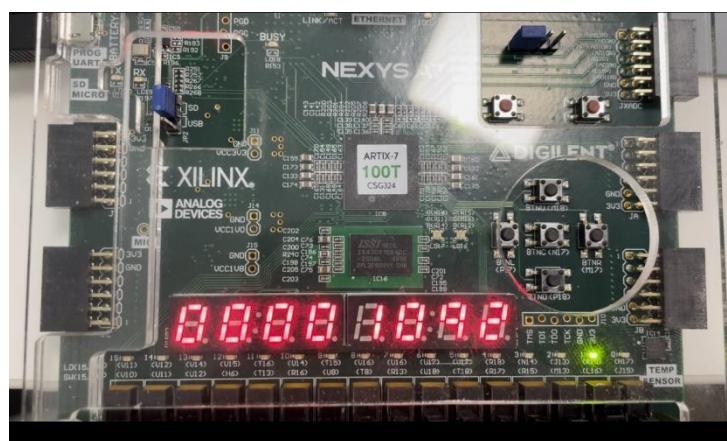
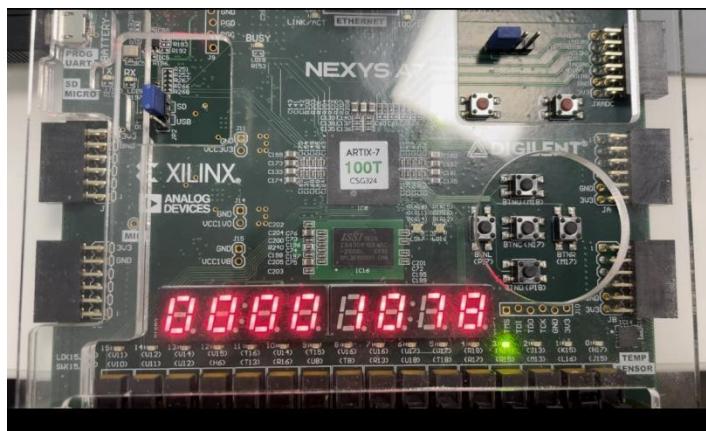
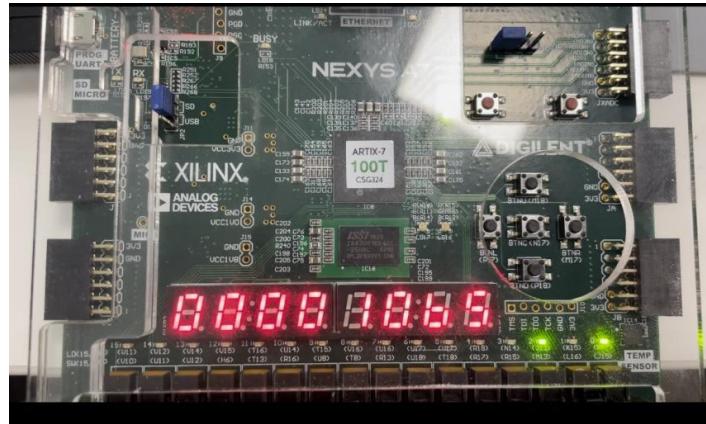
将四位计数器拓展为八位计数器并调整频率

初始状态



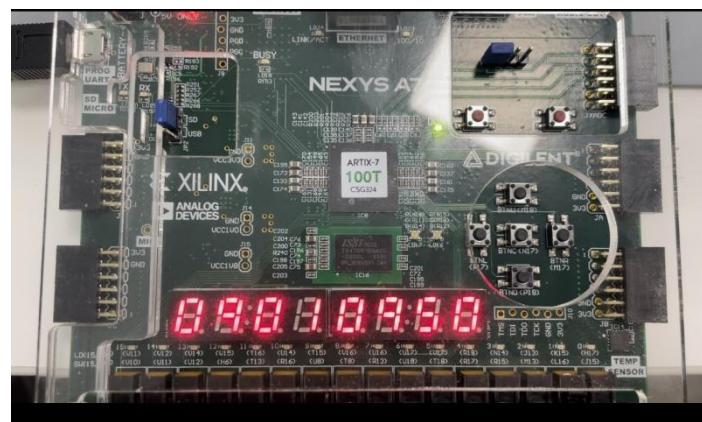
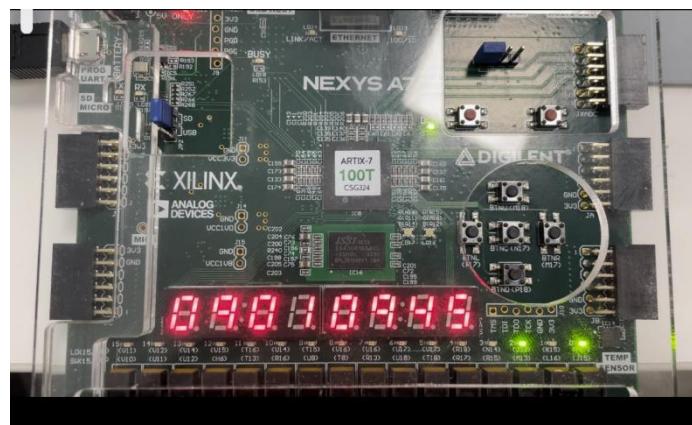
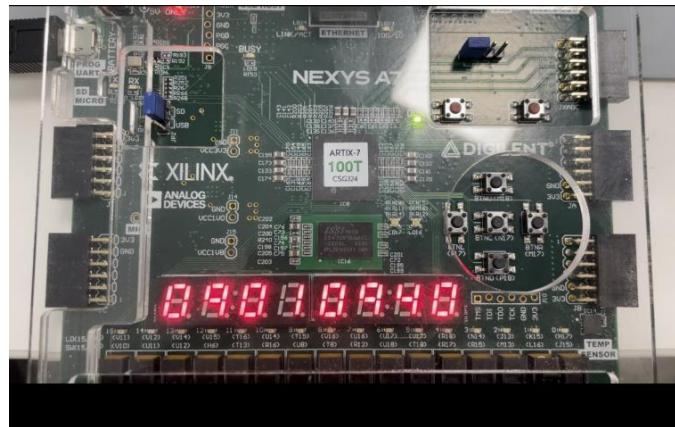
(第 4, 6, 8 秒截屏)

频率提高 10 倍



(第 4, 6, 8 秒截屏)

频率提高 5 倍



(第 4, 6, 8 秒截屏)

实验三 数据比较器和加法器实验

一、实验目的

使用 Verilog HDL 实现 4 位比较器、8 位比较器、串行加法器的设计

- (1) 深入了解比较器和加法器的原理。
- (2) 学习使用 Verilog HDL 设计 4 位比较器和 8 位比较器。
- (3) 学习使用 Verilog HDL 设计 8 位串行加法器

二、实验环境

Verilog HDL:

Verilog HDL 是硬件描述语言 (Hardware Description Language) 之一，它是一种用于设计数字电路和系统的编程语言。它可以用于描述数字电路的结构和行为，并生成可以被计算机执行的代码。Verilog 是由 Cadence Design Systems 在 1984 年开发的，它的设计目标是提供一个标准的硬件描述语言，以便能够方便地进行数字电路的仿真、综合和测试。

Verilog HDL 可以描述数字电路的结构、功能、时序和处理逻辑。它支持不同的建模级别，包括行为级别、寄存器传输级别、门级别和电路级别。使用 Verilog HDL，可以描述各种数字电路和系统，如单片机、处理器、FPGA、ASIC、通信协议等。

Vivado:

Vivado 是 Xilinx 公司开发的一款综合性的集成电路设计软件，它是一种高级的工具，用于设计和验证 FPGA、SoC 以及三维集成电路 (3DIC) 等复杂的数字电路和系统。Vivado 支持多种设计语言，包括 Verilog、SystemVerilog、VHDL 等，并且可以与多种开发工具和硬件平台配合使用，如 ModelSim 仿真器、Zynq SoC、UltraScale+ MPSoC 等。在 Vivado 中，设计人员可以使用可视化界面和多种编辑器，以便更好地创建和管理设计资源。此外，Vivado 还提供了一系列的验证和优化工具，如仿真、调试、布局和布线，以及时序约束生成和验证。这些工具可以帮助设计人员更好地分析和优化电路设计，以达到最佳的性能和功耗。

Nexys4 DDR:

Nexys 4 DDR 是一种高性能的 FPGA 开发板，适用于数字系统设计和嵌入式处理器开发。该板配备了 Xilinx Artix-7 FPGA 芯片，具有大容量、高速和低功耗等优点。此外，该开发板还配备了 DDR3 SDRAM、Flash 存储器、USB 接口、以太网接口和多种传感器，以及其他许多实用的外设。

Nexys 4 DDR 支持多种开发环境和语言，如 Vivado、Verilog、VHDL 和 C/C++ 等。此外，它还提供了许多教学资源和实验，可帮助学生和工程师更快地学习和理解数字系统设计和嵌入式开发。Nexys 4 DDR 开发板特别适合用于数字信号处理、通信系统、多媒体处理、机器视觉和人工智能等领域的应用。

三、实验原理

4位比较器

用来完成两组二进制数大小比较的逻辑电路，称为数据比较器。下图 6.17 给出了 4 位二进制数据比较器的功能框图。 $a_3 a_2 a_1 a_0$ (a_3 为高位) 是一组二进制数输入， $b_3 b_2 b_1 b_0$ (b_3 为高位) 是另一组二进制数输入。 $a > b$, $a = b$, $a < b$ 为级联输入， $A > B$, $A = B$, $A < B$ 为比较结果输出。真值表为：



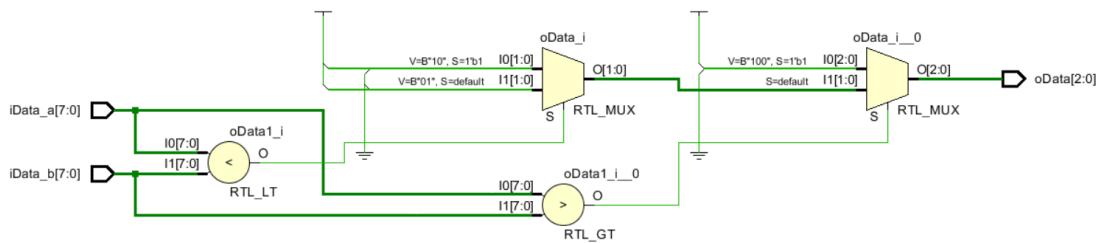
图 6.17 4 位二进制数比较器

8位比较器

当比较位数超过 4 位时，可以将两片或多片级联使用。下图为两片 4 位比较器级联而成的 8 位比较器，此时低 4 位和高 4 位输入信号，分别加到两个比较器的输入端。低 4 位比较器的三个输出分别对应接到高 4 位比较器的三个级联输入端 $a > b$, $a < b$, $a = b$ ，比较结果由高 4 位比较器输出端输出
逻辑表达式为：

$$F = \overline{(A_0 \oplus B_0)} + (A_1 \oplus B_1) + (A_2 \oplus B_2) + (A_3 \oplus B_3) + (A_4 \oplus B_4) + (A_5 \oplus B_5) + (A_6 \oplus B_6) + (A_7 \oplus B_7)$$

原理框图：



串行加法器

加法器是计算机或其他数字系统中对二进制数进行运算处理的组合逻辑构件。本实验主要采用 Verilog HDL 行为描述实现串行加法器，它由多个全加器 (FA) 串行连接而成。每一个全加器是一位加法器，有三个输入（加数 A_i 、被加数 B_i 、低位的进位信号 C_{i-1} ），两个输出（和数 S_i 、向高位的进位信号 C_i ）

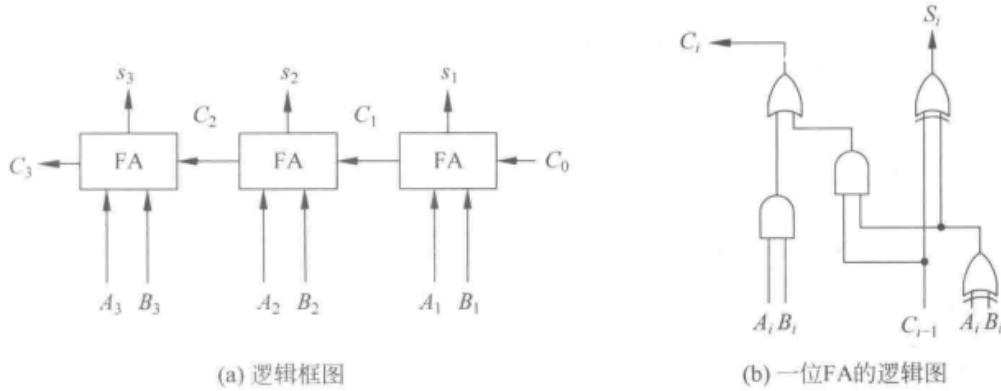


图 6.19 串行加法器框图

四、实验内容

四位比较器代码实现

```

module DataCompare4(
    input [3:0] iData_a,
    input [3:0] iData_b,
    input [2:0] iData,
    output reg [2:0] oData
);

always@(*)
begin
    if( iData == 3'b100) oData = 3'b100;
    else if ( iData == 3'b010 ) oData = 3'b010;
    else
    begin
        if( iData_a[3] > iData_b[3]) oData = 3'b100;
        else if( iData_a[3] < iData_b[3] ) oData = 3'b010;
        else
        begin
            if(iData_a[2] > iData_b[2]) oData = 3'b100;
            else if(iData_a[2] < iData_b[2]) oData = 3'b010;
            else
            begin
                if(iData_a[1] > iData_b[1]) oData = 3'b100;
                else if( iData_a[1] < iData_b[1] ) oData = 3'b010;
            end
        end
    end
end

```

```

pos ;b010;
    else
begin
    if( iData_a[0] > iData_b[0]) oData = 3&apos;b100;
        else if( iData_a[0] > iData_b[0]
) oData = 3&apos;b010;
            else oData = 3&apos;b001;
end
end
end
end
endmodule

```

八位比较器代码实现

```

module DataCompare8(
    input [7:0] iData_a,
    input [7:0] iData_b,
    output reg [2:0] oData
);
    wire [2:0] a,b;
    assign a = 3&apos;b001;
    DataCompare4 V1( iData_a [7:4] , iData_b [7:4] , a , b)
    ;
    DataCompare4 V2( iData_a [3:0] , iData_b [3:0] , b , oDat
a) ;
endmodule

```

串行加法器代码实现

```

module FA(Sum, Co, A, B, Ci) ;
    input A, B, Ci;
    output Sum, Co;
    wire S1, S2, S3;
    xor (Sum, A, B, Ci); //构建 sum
    and (S1, A, B);
    xor (S2, A, B);
    and (S3, Ci, S2);
    or (Co, S1, S3); //构建 co
endmodule

```

```

module Adder(Sum, Co, A, B, Ci) ;
    input [7:0] A, B;

```

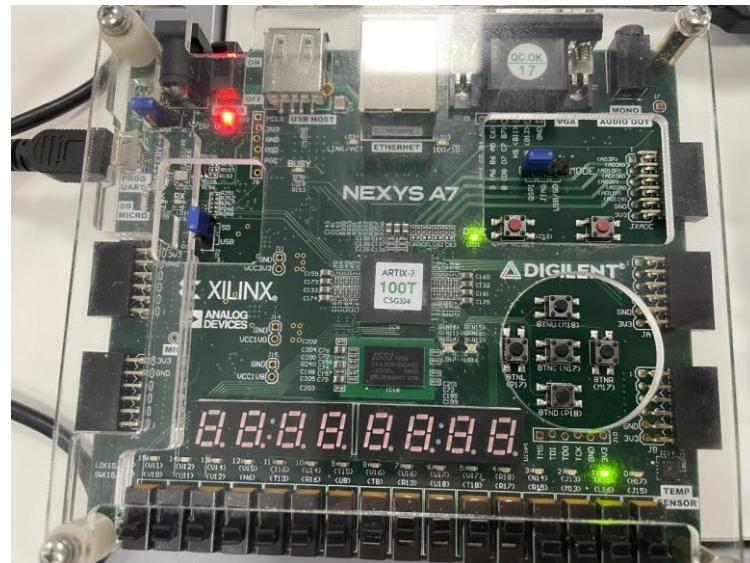
```

input Ci;
output [7:0] Sum, Co;
    FA U0(Sum[0], Co[0], A[0], B[0], Ci);
    FA U1(Sum[1], Co[1], A[1], B[1], Co[0]);
    FA U2(Sum[2], Co[2], A[2], B[2], Co[1]);
    FA U3(Sum[3], Co[3], A[3], B[3], Co[2]);
    FA U4(Sum[4], Co[4], A[4], B[4], Co[3]);
    FA U5(Sum[5], Co[5], A[5], B[5], Co[4]);
    FA U6(Sum[6], Co[6], A[6], B[6], Co[5]);
    FA U7(Sum[7], Co[7], A[7], B[7], Co[6]);
Endmodule:

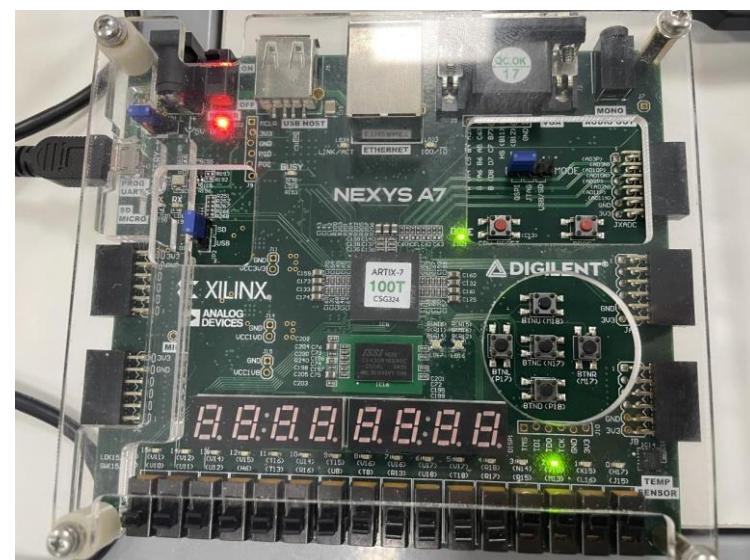
```

五、实验结果

4 位比较器

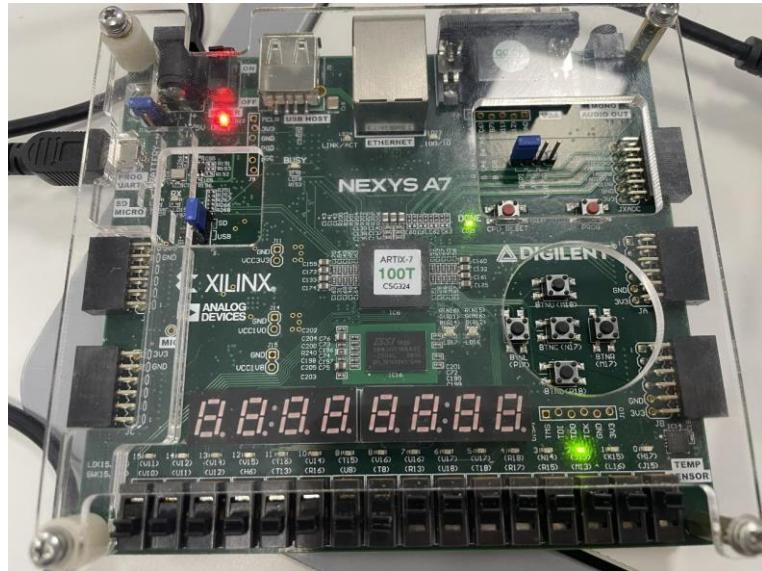


示例 1：当 A 为 0000，B 为 1111 时，输出为 010，说明 A<B

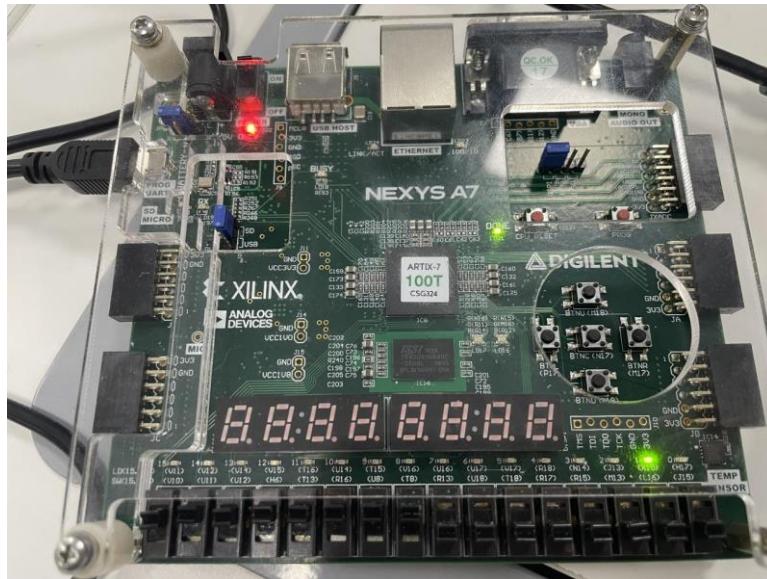


示例 2：当 A 为 1111，B 为 0000 时，输出为 100，说明 A>B

8 位比较器

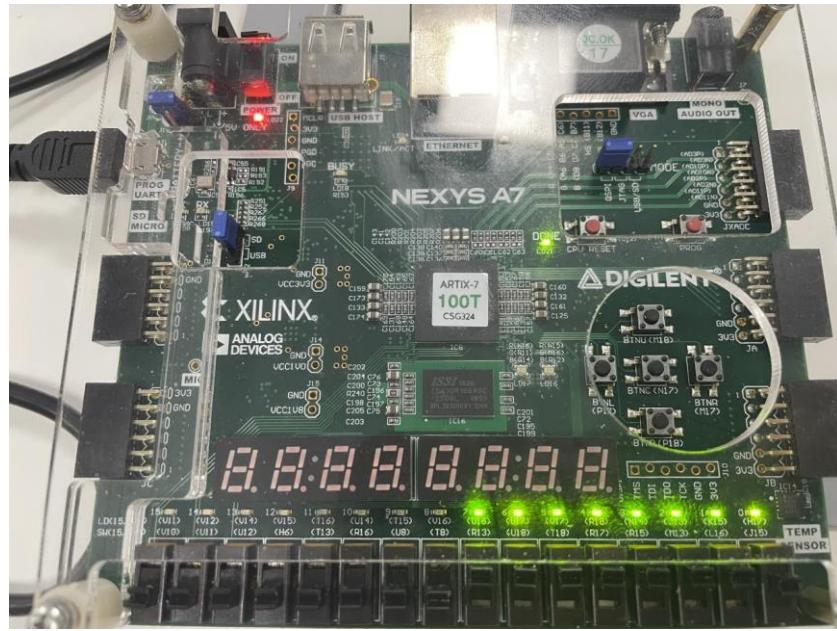


示例 1：当 A 为 11111111，B 为 00000000 时，输出为 100，说明 A>B

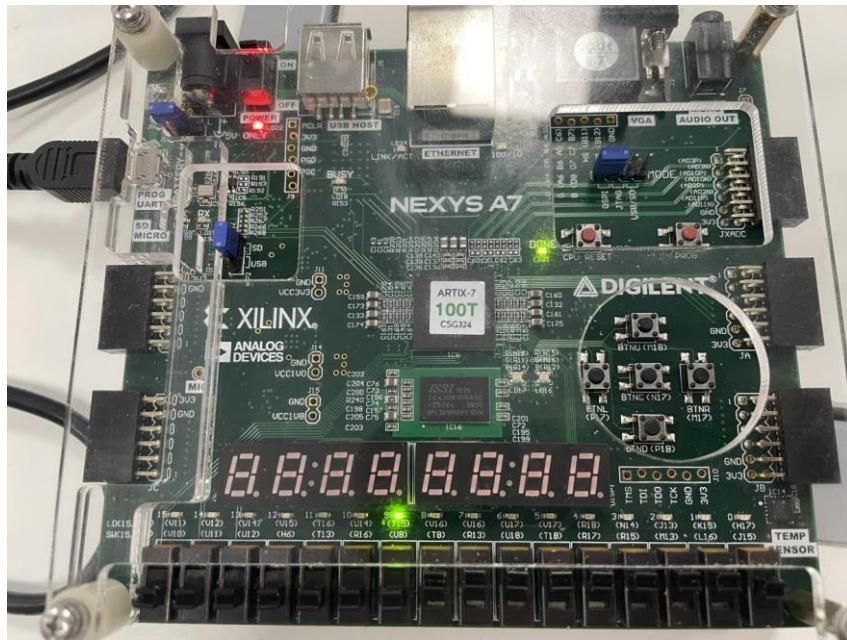


示例 2：当 A 为 00000000，B 为 11111111 时，输出为 010，说明 A<B

串行加法器



示例 1：当 A 为 11111111，B 为 00000000 时，输出为 0000000011111111



示例 2：当 A 为 11111111，B 为 00000001 时，输出为 0000001000000000

实验四：计数器与分频器

一、 实验目的

使用 Verilog HDL 实现计数器和分频器的设计

(1) 深入了解计数器和分频器的原理。学习使用 Verilog 设计实现同步计数器和分频器。

(2) 计数器为模 8 计数器，要求在实验报告中给出次态真值表，激励函数。

(3) 分频器，支持用两个拨码开关，实现频率切换，分频器输出增加十进制数码管显示。

二、 实验环境

Verilog HDL:

Verilog HDL 是硬件描述语言 (Hardware Description Language) 之一，它是一种用于设计数字电路和系统的编程语言。它可以用于描述数字电路的结构和行为，并生成可以被计算机执行的代码。Verilog 是由 Cadence Design Systems 在 1984 年开发的，它的设计目标是提供一个标准的硬件描述语言，以便能够方便地进行数字电路的仿真、综合和测试。

Verilog HDL 可以描述数字电路的结构、功能、时序和处理逻辑。它支持不同的建模级别，包括行为级别、寄存器传输级别、门级别和电路级别。使用 Verilog HDL，可以描述各种数字电路和系统，如单片机、处理器、FPGA、ASIC、通信协议等。

Vivado:

Vivado 是 Xilinx 公司开发的一款综合性的集成电路设计软件，它是一种高级的工具，用于设计和验证 FPGA、SoC 以及三维集成电路 (3DIC) 等复杂的数字电路和系统。Vivado 支持多种设计语言，包括 Verilog、SystemVerilog、VHDL 等，并且可以与多种开发工具和硬件平台配合使用，如 ModelSim 仿真器、Zynq SoC、UltraScale+ MPSoC 等。在 Vivado 中，设计人员可以使用可视化界面和多种编辑器，以便更好地创建和管理设计资源。此外，Vivado 还提供了一系列的验证和优化工具，如仿真、调试、布局和布线，以及时序约束生成和验证。这些工具可以帮助设计人员更好地分析和优化电路设计，以达到最佳的性能和功耗。

Nexys4 DDR:

Nexys 4 DDR 是一种高性能的 FPGA 开发板，适用于数字系统设计和嵌入式处理器开发。该板配备了 Xilinx Artix-7 FPGA 芯片，具有大容量、高速和低功耗等优点。此外，该开发板还配备了 DDR3 SDRAM、Flash 存储器、USB 接口、以太网接口和多种传感器，以及其他许多实用的外设。

Nexys 4 DDR 支持多种开发环境和语言，如 Vivado、Verilog、VHDL 和 C/C++ 等。此外，它还提供了许多教学资源和实验，可帮助学生和工程师更快地学习和理解数字系统设计和嵌入式开发。Nexys 4 DDR 开发板特别适合用于数字信号处理、通信系统、多媒体处理、机器视觉和人工智能等领域的应用。

三、实验原理

计数器

计数器的功能是记忆脉冲的个数，它是数字系统中应用最广泛的基本时序逻辑构件。计数器所能记忆脉冲的最大数目称为该计数器的模，用 M 表示。构成计数器的核心元件是触发器。如图 6.24 所示为 3 位同步模 8 计数器逻辑图，它由 3 个 JK 触发器组成。所有触发器的时钟都与同一个时钟脉冲源连接在一起，每一个触发器的状态变化都与时钟脉冲同步，计数器的模 $M=2^3=8$ （注：各触发器工作前要清 0）。表所示为模 8 计数器的次态真值表。

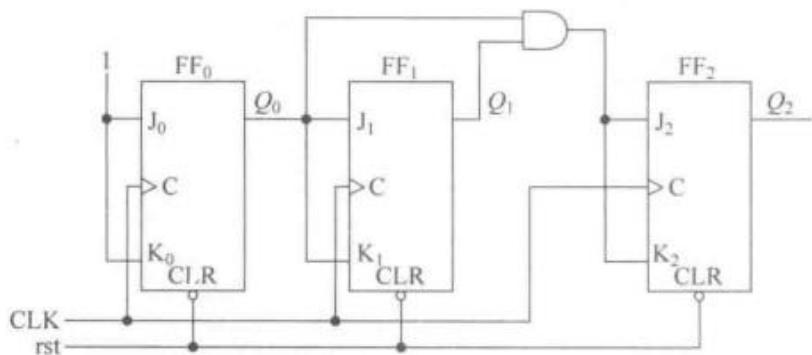


图 6.24 计数方式构成的同步模 8 计数器

Q_2	Q_1	Q_0	Q_2^{n-1}	Q_1^{n-1}	Q_0^{n-1}	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	X	X	0	1	X
0	0	1	0	1	0	0	X	X	1	X	1
0	1	0	0	1	1	0	X	X	X	1	X
0	1	1	1	0	0	1	X	X	X	X	1
1	0	0	1	0	1	X	0	0	0	1	X
1	0	1	1	1	0	X	0	0	1	X	1
1	1	0	1	1	1	X	0	0	X	1	X
1	1	1	0	0	0	X	1	1	X	X	1

(模 8 计数器的次态真值表)

通过次态真值表可以得到激励函数为：

$$J_0 = K_0 = 1 \quad J_1 = K_1 = Q_0 \quad J_2 = K_2 = Q_0 Q_1$$

分频器

每一个计数器的脉冲输出频率等于其输入时钟频率除以计数模值，因此可以很容易地利用计数器由一个输入时钟信号获得分频后的时钟信号，这种应用称为分频器。在实验中，我们使用 Verilog HDL 行为描述方法设计一个分频器。

本次实验只需要在已有的代码上进行修改即可。具体修改见下一部分。

四、 实验内容

seg_led_top 代码部分

```
module seg_led_top(
    //global clock
    input          sys_clk  ,      // 全局时钟信号
    input          sys_rst_n,      // 复位信号 (低有效)
    input          sw,
    //seg_led interface
    output [5:0]   seg_sel  ,      // 数码管位选信号
    output [7:0]   seg_led   ,      // 数码管段选信号
);

//wire define
wire [19:0]  data;           // 数码管显示的数值
wire [5:0]    point;          // 数码管小数点的位置
wire         en;              // 数码管显示使能信号
wire         sign;            // 数码管显示数据的符号位
wire neg_rst_n;
//计数器模块, 产生数码管需要显示的数据
count u_count(
    .clk          (sys_clk ),      // 时钟信号
    .sw           (sw ),           // 复位信号
    // .rst_n       (sys_rst_n),    // 复位信号
    .rst_n       (neg_rst_n),     // 复位信号

    .data         (data ),        // 6 位数码管要显示的数值
    .point        (point ),       // 小数点具体显示的位置,高电平有效
    .en          (en ),           // 数码管使能信号
    .sign         (sign ),        // 符号位
);

//数码管动态显示模块
seg_led u_seg_led(
    .clk          (sys_clk ),      // 时钟信号
    // .rst_n       (sys_rst_n),    // 复位信号
    .rst_n       (neg_rst_n),     // 复位信号

    .data         (data ),        // 显示的数值
    .point        (point ),       // 小数点具体显示的位置,高电平有效
```

```

    .en          (en      ),      // 数码管使能信号
    .sign        (sign    ),      // 符号位, 高电平显示负号(-)

    .seg_sel     (seg_sel ),      // 位选
    .seg_led     (seg_led ),      // 段选
);

assign neg_rst_n = ~sys_rst_n;
endmodule

```

count 代码部分

```

module count(
    //mudule clock
    input                  clk ,      // 时钟信号
    input                  rst_n,     // 复位信号
    input                  sw,       // 

    //user interface
    output reg [19:0]      data ,     // 6 个数码管要显示的数值
    output reg [5:0]       point,    // 小数点的位置,高电平点亮对应数码管位上的小
    数点
    output reg             en ,      // 数码管使能信号
    output reg             sign      // 符号位, 高电平时显示负号, 低电平不显示
    负号
);

//parameter define
parameter MAX_NUM0 = 26'd5000_0000;
parameter MAX_NUM1 = 23'd5000_000;      // 计数器计数的最大值

//reg define
reg  [26:0]   cnt ;                // 计数器, 用于计时 100ms
reg           flag;                // 标志信号

//*****
//**          main code
//*****


//计数器对系统时钟计数达 10ms 时, 输出一个时钟周期的脉冲信号
always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        cnt <= 26'b0;
        flag<= 1'b0;
    end

```

```

else if (sw == 1'b0 && cnt < MAX_NUM0 - 1'b1) begin
    cnt <= cnt + 1'b1;
    flag<= 1'b0;
end
else if (sw == 1'b1 && cnt < MAX_NUM1 - 1'b1) begin
    cnt <= cnt + 1'b1;
    flag<= 1'b0;
end
else begin
    cnt <= 26'b0;
    flag <= 1'b1;
end
end

//数码管需要显示的数据，从 0 累加到 999999
always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)begin
        data  <= 20'b0;
        point <=6'b000000;
        en     <= 1'b0;
        sign   <= 1'b0;
    end
    else begin
        point <= 6'b000000;           //不显示小数点
        en      <= 1'b1;             //打开数码管使能信号
        sign   <= 1'b0;              //不显示负号
        if (flag) begin               //显示数值每隔 0.01s 累加一次
            if(data < 20'd000007)
                data <= data +1'b1;
            else
                data <= 20'b0;
        end
    end
end

endmodule

```

seg_led 代码部分

```

module seg_led(
    input          clk      ,      // 时钟信号
    input          rst_n   ,      // 复位信号
    input [19:0]    data     ,      // 6 位数码管要显示的数值

```

```

        input      [5:0]    point ,      // 小数点具体显示的位置,从高到低,高电平有效
        input          en ,      // 数码管使能信号
        input          sign ,     // 符号位 (高电平显示"-")号)

        output reg [5:0]    seg_sel,    // 数码管位选, 最左侧数码管为最高位
        output reg [7:0]    seg_led     // 数码管段选
    );

//parameter define
localparam CLK_DIVIDE = 4'd10      ;      // 时钟分频系数
localparam MAX_NUM      = 13'd5000   ;      // 对数码管驱动时钟(5MHz)计数 1ms 所需的计数值

//reg define
reg [ 3:0]          clk_cnt ;      // 时钟分频计数器
reg                  dri_clk ;      // 数码管的驱动时钟,5MHz
reg [23:0]          num      ;      // 24 位 bcd 码寄存器
reg [12:0]          cnt0     ;      // 数码管驱动时钟计数器
reg                  flag     ;      // 标志信号 (标志着 cnt0 计数达 1ms)
reg [2:0]           cnt_sel ;      // 数码管位选计数器
reg [3:0]           num_disp;      // 当前数码管显示的数据
reg                  dot_disp;     // 当前数码管显示的小数点

//wire define
wire [3:0]          data0   ;      // 个位数
wire [3:0]          data1   ;      // 十位数
wire [3:0]          data2   ;      // 百位数
wire [3:0]          data3   ;      // 千位数
wire [3:0]          data4   ;      // 万位数
wire [3:0]          data5   ;      // 十万位数

//*****
//**          main code
//*****


//提取显示数值所对应的十进制数的各个位
assign data0 = data % 4'd10;      // 个位数
assign data1 = data / 4'd10 % 4'd10 ;      // 十位数
assign data2 = data / 7'd100 % 4'd10 ;      // 百位数
assign data3 = data / 10'd1000 % 4'd10 ;      // 千位数
assign data4 = data / 14'd10000 % 4'd10;      // 万位数
assign data5 = data / 17'd100000;      // 十万位数

```

```

//对系统时钟 10 分频, 得到的频率为 5MHz 的数码管驱动时钟 dri_clk
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        clk_cnt <= 4'd0;
        dri_clk <= 1'b1;
    end
    else if(clk_cnt == CLK_DIVIDE/2 - 1'd1) begin
        clk_cnt <= 4'd0;
        dri_clk <= ~dri_clk;
    end
    else begin
        clk_cnt <= clk_cnt + 1'b1;
        dri_clk <= dri_clk;
    end
end

//将 20 位 2 进制数转换为 8421bcd 码(即使用 4 位二进制数表示 1 位十进制数)
always @ (posedge dri_clk or negedge rst_n) begin
    if (!rst_n)
        num <= 24'b0;
    else begin
        if (data5 || point[5]) begin      //如果显示数据为 6 位十进制数,
            num[23:20] <= data5;          //则依次给 6 位数码管赋值
            num[19:16] <= data4;
            num[15:12] <= data3;
            num[11:8]  <= data2;
            num[ 7:4]  <= data1;
            num[ 3:0]  <= data0;
        end
        else begin
            if (data4 || point[4]) begin //如果显示数据为 5 位十进制数, 则给低 5 位数码管
                num[19:0] <= {data4,data3,data2,data1,data0};
                if(sign)
                    num[23:20] <= 4'd11; //如果需要显示负号, 则最高位 (第 6 位) 为
                //符号位
                else
                    num[23:20] <= 4'd10; //不需要显示负号时, 则第 6 位不显示任何字
                //符
            end
            else begin                  //如果显示数据为 4 位十进制数, 则给低 4 位
                num[15: 0] <= {data3,data2,data1,data0};
            end
        end
    end

```

```

        num[23:20] <= 4'd10; //第 6 位不显示任何字符
        if(sign)           //如果需要显示负号, 则最高位(第 5 位)为符
号位
        num[19:16] <= 4'd11;
        else             //不需要显示负号时, 则第 5 位不显示任何字
符
        num[19:16] <= 4'd10;
end
else begin           //如果显示数据为 3 位十进制数, 则给低 3 位
数码管赋值
if (data2 || point[2]) begin
    num[11: 0] <= {data2,data1,data0};
        //第 6、5 位不显示任何字符
    num[23:16] <= {2{4'd10}};
    if(sign)           //如果需要显示负号, 则最高位(第 4 位)为符
号位
    num[15:12] <= 4'd11;
    else             //不需要显示负号时, 则第 4 位不显示任何字
符
    num[15:12] <= 4'd10;
end
else begin           //如果显示数据为 2 位十进制数, 则给低 2 位
数码管赋值
if (data1 || point[1]) begin
    num[ 7: 0] <= {data1,data0};
        //第 6、5、4 位不显示任何字符
    num[23:12] <= {3{4'd10}};
    if(sign)           //如果需要显示负号, 则最高位(第 3 位)为符
号位
    num[11:8]  <= 4'd11;
    else             //不需要显示负号时, 则第 3 位不显示任何字
符
    num[11:8] <= 4'd10;
end
else begin           //如果显示数据为 1 位十进制数, 则给最低位
数码管赋值
num[3:0] <= data0;
        //第 6、5 位不显示任何字符
num[23:8] <= {4{4'd10}};
    if(sign)           //如果需要显示负号, 则最高位(第 2 位)为符
号位
    num[7:4] <= 4'd11;
    else             //不需要显示负号时, 则第 2 位不显示任何字
符

```

```

        num[7:4] <= 4'd10;
    end

//每当计数器对数码管驱动时钟计数时间达 1ms, 输出一个时钟周期的脉冲信号
always @ (posedge dri_clk or negedge rst_n) begin
    if (rst_n == 1'b0) begin
        cnt0 <= 13'b0;
        flag <= 1'b0;
    end
    else if (cnt0 < MAX_NUM - 1'b1) begin
        cnt0 <= cnt0 + 1'b1;
        flag <= 1'b0;
    end
    else begin
        cnt0 <= 13'b0;
        flag <= 1'b1;
    end
end
end

//cnt_sel 从 0 计数到 5, 用于选择当前处于显示状态的数码管
always @ (posedge dri_clk or negedge rst_n) begin
    if (rst_n == 1'b0)
        cnt_sel <= 3'b0;
    else if(flag) begin
        if(cnt_sel < 3'd5)
            cnt_sel <= cnt_sel + 1'b1;
        else
            cnt_sel <= 3'b0;
    end
    else
        cnt_sel <= cnt_sel;
end

//控制数码管位选信号, 使 6 位数码管轮流显示
always @ (posedge dri_clk or negedge rst_n) begin
    if(!rst_n) begin
        seg_sel  <= 6'b111111;          //位选信号低电平有效
        num_disp <= 4'b0;
    end
end

```

```

dot_disp <= 1'b1;           //共阳极数码管，低电平导通
end
else begin
    if(en) begin
        case (cnt_sel)
            3'd0 :begin
                seg_sel <= 6'b111110; //显示数码管最低位
                num_disp <= num[3:0]; //显示的数据
                dot_disp <= ~point[0]; //显示的小数点
            end
            3'd1 :begin
                seg_sel <= 6'b111101; //显示数码管第 1 位
                num_disp <= num[7:4];
                dot_disp <= ~point[1];
            end
            3'd2 :begin
                seg_sel <= 6'b111011; //显示数码管第 2 位
                num_disp <= num[11:8];
                dot_disp <= ~point[2];
            end
            3'd3 :begin
                seg_sel <= 6'b110111; //显示数码管第 3 位
                num_disp <= num[15:12];
                dot_disp <= ~point[3];
            end
            3'd4 :begin
                seg_sel <= 6'b101111; //显示数码管第 4 位
                num_disp <= num[19:16];
                dot_disp <= ~point[4];
            end
            3'd5 :begin
                seg_sel <= 6'b011111; //显示数码管最高位
                num_disp <= num[23:20];
                dot_disp <= ~point[5];
            end
        default :begin
            seg_sel <= 6'b111111;
            num_disp <= 4'b0;
            dot_disp <= 1'b1;
        end
    endcase
end
else begin
    seg_sel <= 6'b111111;      //使能信号为 0 时，所有数码管均不显示

```

```

        num_disp <= 4'b0;
        dot_disp <= 1'b1;
    end
end

//控制数码管段选信号，显示字符
always @ (posedge dri_clk or negedge rst_n) begin
    if (!rst_n)
        seg_led <= 8'hc0;
    else begin
        case (num_disp)
            4'd0 : seg_led <= {dot_disp,7'b1000000}; //显示数字 0
            4'd1 : seg_led <= {dot_disp,7'b1111001}; //显示数字 1
            4'd2 : seg_led <= {dot_disp,7'b0100100}; //显示数字 2
            4'd3 : seg_led <= {dot_disp,7'b0110000}; //显示数字 3
            4'd4 : seg_led <= {dot_disp,7'b0011001}; //显示数字 4
            4'd5 : seg_led <= {dot_disp,7'b0010010}; //显示数字 5
            4'd6 : seg_led <= {dot_disp,7'b0000010}; //显示数字 6
            4'd7 : seg_led <= {dot_disp,7'b1111000}; //显示数字 7
            4'd8 : seg_led <= {dot_disp,7'b0000000}; //显示数字 8
            4'd9 : seg_led <= {dot_disp,7'b0010000}; //显示数字 9
            4'd10: seg_led <= 8'b11111111;           //不显示任何字符
            4'd11: seg_led <= 8'b10111111;           //显示负号(-)
        default:
            seg_led <= {dot_disp,7'b1000000};
        endcase
    end
end

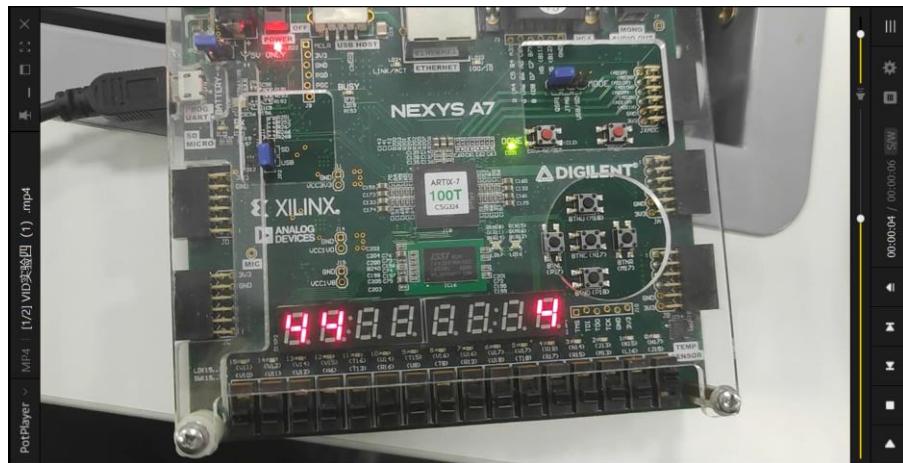
endmodule

```

五、实验结果

VID1 为模 8 计数器降频之后的结果：
(视频在文件夹中)

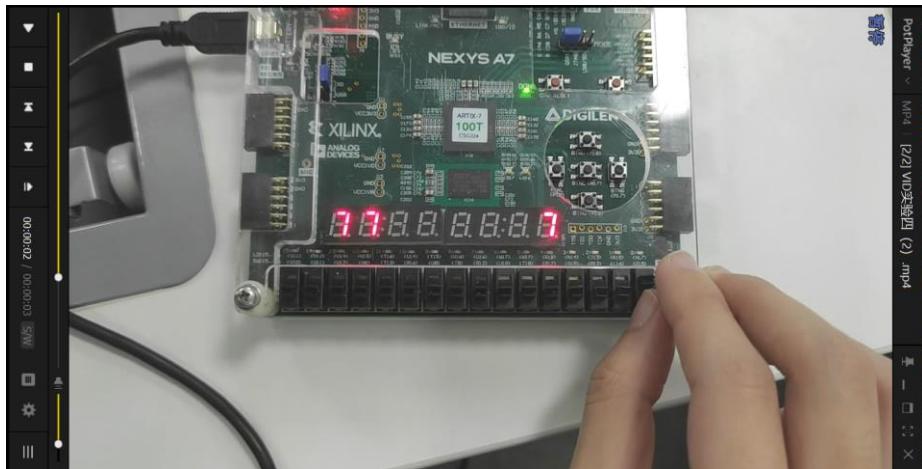




(视频中间时刻截屏)

VID2 为模 8 计数器升频之后的结果:

(视频在文件夹中)



(视频中间时刻截屏)

实验五：数码管动态显示的数字时钟

一、 实验目的

在本次实验中，增加使用数码管动态显示的时钟（时：分：秒）

- (1) 输出增加十进制数码管显示。
- (2) 增加数码管动态显示功能

二、 实验环境

Verilog HDL:

Verilog HDL 是硬件描述语言（Hardware Description Language）之一，它是一种用于设计数字电路和系统的编程语言。它可以用于描述数字电路的结构和行为，并生成可以被计算机执行的代码。Verilog 是由 Cadence Design Systems 在 1984 年开发的，它的设计目标是提供一个标准的硬件描述语言，以便能够方便地进行数字电路的仿真、综合和测试。

Verilog HDL 可以描述数字电路的结构、功能、时序和处理逻辑。它支持不同的建模级别，包括行为级别、寄存器传输级别、门级别和电路级别。使用 Verilog HDL，可以描述各种数字电路和系统，如单片机、处理器、FPGA、ASIC、通信协议等。

Vivado:

Vivado 是 Xilinx 公司开发的一款综合性的集成电路设计软件，它是一种高级的工具，用于设计和验证 FPGA、SoC 以及三维集成电路（3DIC）等复杂的数字电路和系统。Vivado 支持多种设计语言，包括 Verilog、SystemVerilog、VHDL 等，并且可以与多种开发工具和硬件平台配合使用，如 ModelSim 仿真器、Zynq SoC、UltraScale+ MPSoC 等。在 Vivado 中，设计人员可以使用可视化界面和多种编辑器，以便更好地创建和管理设计资源。此外，Vivado 还提供了一系列的验证和优化工具，如仿真、调试、布局和布线，以及时序约束生成和验证。这些工具可以帮助设计人员更好地分析和优化电路设计，以达到最佳的性能和功耗。

Nexys4 DDR:

Nexys 4 DDR 是一种高性能的 FPGA 开发板，适用于数字系统设计和嵌入式处理器开发。该板配备了 Xilinx Artix-7 FPGA 芯片，具有大容量、高速和低功耗等优点。此外，该开发板还配备了 DDR3 SDRAM、Flash 存储器、USB 接口、以太网接口和多种传感器，以及其他许多实用的外设。

Nexys 4 DDR 支持多种开发环境和语言，如 Vivado、Verilog、VHDL 和 C/C++ 等。此外，它还提供了许多教学资源和实验，可帮助学生和工程师更快地学习和理解数字系统设计和嵌入式开发。Nexys 4 DDR 开发板特别适合用于数字信号处理、通信系统、多媒体处理、机器视觉和人工智能等领域的应用。

三、 实验原理

分频器

输入：时钟信号 sys_clk、重置信号 sys_rst_n

输出：分频时钟信号 clk

功能：对 50MHz 分频

板子上电之后需要每隔 1s 进行计数，板子的时钟频率是 50MHz，为了满足这一功能，需要设计一个分频器，对板子的 50MHz 频率进行分频，从而输出一个 1Hz 的时钟信号。

分频器模值、系统时钟和期望输出时钟频率关系为：

$$\text{分频器模值} = \frac{\text{系统时钟频率}}{\text{期望输出时钟频率}} - 1$$

所以，把 50MHz 的时钟分频，输出 1Hz 的时钟，分频器的模值为：

$$M = \frac{50000000}{1} - 1 = 49999999$$

为了保证分频器正常工作，计数器寄存器所能表示的最大值必须大于分频器的模值。这里，设置把计数器寄存器的位数设置为 26 位。

数码管的动态显示

输入：重置信号，时钟信号，计数时间，设定时间，数码管使能信号，模式信号

输出：数码管位选信号，数码管选段信号

功能：将计数时间/设定时间显示在数码管上。

在数码管动态显示模块中，首先需要根据模式信号来判断需要显示的时间，即时间显示模式显示计数时间，时间设定模式显示设定时间。之后，通过整除去尾、取模取尾的计算方式，将时间每个位数上的数据提取出来。比如说，提取十位数的数据公式为：

$$\text{十位数} = \text{data} / 10\%10$$

将每一位上的数据提取出来后，将其转换为 8421BCD 码，与每一个数码管进行绑定。通过对系统时钟进行 10 分频，得到的频率为 5MHz 的数码管驱动时钟，用来控制数码管的位选信号，使每一个数码管以 1ms 的时间周期轮流显示。

当数码管需要显示的时候，通过段选信号将每一位数码管绑定的数据进行转换，从而显示出正确的数值。

本次实验只需要修改给定的代码即可。为了简化实验过程，本实验显示的时钟信号在高位不予显示，只有当该位被使用时才会显示具体数值。

四、实验内容

seg_led_top 代码部分

```
module seg_led_top(
    //global clock
    input          sys_clk  ,      // 全局时钟信号
    input          sys_rst_n,      // 复位信号 (低有效)
    input          sw,
    //seg_led interface
    output     [5:0]  seg_sel  ,      // 数码管位选信号
    output     [7:0]  seg_led       // 数码管段选信号
);

//wire define
wire      [19:0]  data;           // 数码管显示的数值
wire      [ 5:0]   point;         // 数码管小数点的位置
wire      en;                  // 数码管显示使能信号
wire      sign;                // 数码管显示数据的符号位
wire    neg_rst_n;

//*****
//**          main code
//*****
```

//计数器模块，产生数码管需要显示的数据

```
count u_count(
    .clk          (sys_clk  ),      // 时钟信号
    .sw           (sw ),           // 
    // .rst_n       (sys_rst_n),      // 复位信号
    .rst_n       (neg_rst_n),      // 复位信号

    .data         (data      ),      // 6 位数码管要显示的数值
    .point        (point     ),      // 小数点具体显示的位置,高电平有效
    .en          (en      ),      // 数码管使能信号
    .sign         (sign     )       // 符号位
);
```

//数码管动态显示模块

```
seg_led u_seg_led(
    .clk          (sys_clk  ),      // 时钟信号
    // .rst_n       (sys_rst_n),      // 复位信号
```

```

    .rst_n      (neg_rst_n),      // 复位信号

    .data       (data      ),      // 显示的数值
    .point     (point     ),      // 小数点具体显示的位置,高电平有效
    .en        (en       ),      // 数码管使能信号
    .sign      (sign     ),      // 符号位, 高电平显示负号(-)

    .seg_sel   (seg_sel  ),      // 位选
    .seg_led   (seg_led  )      // 段选
);

assign neg_rst_n = ~sys_rst_n;

endmodule

```

count 代码部分

```

module count(
    //mudule clock
    input                  clk ,      // 时钟信号
    input                  rst_n,      // 复位信号
    input                  sw,         

    //user interface
    output reg [19:0]      data ,      // 6 个数码管要显示的数值
    output reg [5:0]       point,      // 小数点的位置,高电平点亮对应数码管位上的小数点
    output reg              en ,       // 数码管使能信号
    output reg              sign ,     // 符号位, 高电平时显示负号, 低电平不显示负号
);

//parameter define
parameter MAX_NUM0 = 26'd5000_0000;
parameter MAX_NUM1 = 23'd5000_000;      // 计数器计数的最大值

//reg define
reg      [26:0]      cnt ;           // 计数器, 用于计时 100ms
reg      flag;                 // 标志信号

//*****
//**          main code
//*****

```

```

//计数器对系统时钟计数达 10ms 时，输出一个时钟周期的脉冲信号
always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        cnt <= 26'b0;
        flag<= 1'b0;
    end
    else if (sw == 1'b0 && cnt < MAX_NUM0 - 1'b1) begin
        cnt <= cnt + 1'b1;
        flag<= 1'b0;
    end
    else if (sw == 1'b1 && cnt < MAX_NUM1 - 1'b1) begin
        cnt <= cnt + 1'b1;
        flag<= 1'b0;
    end
    else begin
        cnt <= 26'b0;
        flag <= 1'b1;
    end
end

//数码管需要显示的数据，从 0 累加到 999999
always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)begin
        data  <= 20'b0;
        point <=6'b000000;
        en     <= 1'b0;
        sign   <= 1'b0;
    end
    else begin
        point <= 6'b000000;           //不显示小数点
        en     <= 1'b1;              //打开数码管使能信号
        sign   <= 1'b0;              //不显示负号
        if (flag) begin             //显示数值每隔 0.01s 累加一次
            if( (data%100) < 20'd000059)
                data <= data +1'b1;
            else
                data <= data + 20'd41;
            if( (data % 20'd10000) == 20'd5959)
                data <= data + 20'd4041;
            if( (data % 20'd1000000) == 20'd240000)
                data <= 20'b0;
        end
    end
end

```

```

    end
end

endmodule

```

seg_led 代码部分

```

module seg_led(
    input          clk      ,      // 时钟信号
    input          rst_n   ,      // 复位信号

    input [19:0]   data     ,      // 6 位数码管要显示的数值
    input [5:0]    point   ,      // 小数点具体显示的位置,从高到低,高电平
有效
    input          en       ,      // 数码管使能信号
    input          sign    ,      // 符号位 (高电平显示"-"号)

    output reg [5:0] seg_sel,      // 数码管位选, 最左侧数码管为最高位
    output reg [7:0] seg_led,      // 数码管段选
);

//parameter define
localparam CLK_DIVIDE = 4'd10      ;      // 时钟分频系数
localparam MAX_NUM      = 13'd5000   ;      // 对数码管驱动时钟(5MHz)计数 1ms
所需的计数值

//reg define
reg [ 3:0]      clk_cnt  ;      // 时钟分频计数器
reg             dri_clk  ;      // 数码管的驱动时钟,5MHz
reg [23:0]      num      ;      // 24 位 bcd 码寄存器
reg [12:0]      cnt0     ;      // 数码管驱动时钟计数器
reg             flag     ;      // 标志信号 (标志着 cnt0 计数达 1ms)
reg [2:0]       cnt_sel  ;      // 数码管位选计数器
reg [3:0]       num_disp ;      // 当前数码管显示的数据
reg             dot_disp ;      // 当前数码管显示的小数点

//wire define
wire [3:0]      data0    ;      // 个位数
wire [3:0]      data1    ;      // 十位数
wire [3:0]      data2    ;      // 百位数
wire [3:0]      data3    ;      // 千位数
wire [3:0]      data4    ;      // 万位数
wire [3:0]      data5    ;      // 十万位数

```

```

//*****
//**          main code
//*****

//提取显示数值所对应的十进制数的各个位
assign data0 = data % 4'd10;           // 个位数
assign data1 = data / 4'd10 % 4'd10 ;  // 十位数
assign data2 = data / 7'd100 % 4'd10 ; // 百位数
assign data3 = data / 10'd1000 % 4'd10; // 千位数
assign data4 = data / 14'd10000 % 4'd10; // 万位数
assign data5 = data / 17'd100000;      // 十万位数

//对系统时钟 10 分频, 得到的频率为 5MHz 的数码管驱动时钟 dri_clk
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        clk_cnt <= 4'd0;
        dri_clk <= 1'b1;
    end
    else if(clk_cnt == CLK_DIVIDE/2 - 1'd1) begin
        clk_cnt <= 4'd0;
        dri_clk <= ~dri_clk;
    end
    else begin
        clk_cnt <= clk_cnt + 1'b1;
        dri_clk <= dri_clk;
    end
end

//将 20 位 2 进制数转换为 8421bcd 码(即使用 4 位二进制数表示 1 位十进制数)
always @ (posedge dri_clk or negedge rst_n) begin
    if (!rst_n)
        num <= 24'b0;
    else begin
        if (data5 || point[5]) begin      //如果显示数据为 6 位十进制数,
            num[23:20] <= data5;       //则依次给 6 位数码管赋值
            num[19:16] <= data4;
            num[15:12] <= data3;
            num[11:8]  <= data2;
            num[ 7:4]  <= data1;
            num[ 3:0]  <= data0;
        end
        else begin
            if (data4 || point[4]) begin //如果显示数据为 5 位十进制数, 则给低 5 位数码管
赋值

```

```

num[19:0] <= {data4,data3,data2,data1,data0};
if(sign)
    num[23:20] <= 4'd11; //如果需要显示负号, 则最高位 (第 6 位) 为
符号位
else
    num[23:20] <= 4'd10; //不需要显示负号时, 则第 6 位不显示任何字
符
end
else begin
    //如果显示数据为 4 位十进制数, 则给低 4 位
数码管赋值
    if (data3 || point[3]) begin
        num[15: 0] <= {data3,data2,data1,data0};
        num[23:20] <= 4'd10; //第 6 位不显示任何字符
        if(sign)           //如果需要显示负号, 则最高位 (第 5 位) 为符
号位
        num[19:16] <= 4'd11;
    else
        //不需要显示负号时, 则第 5 位不显示任何字
符
        num[19:16] <= 4'd10;
    end
else begin
    //如果显示数据为 3 位十进制数, 则给低 3 位
数码管赋值
    if (data2 || point[2]) begin
        num[11: 0] <= {data2,data1,data0};
        //第 6、5 位不显示任何字符
        num[23:16] <= {2{4'd10}};
        if(sign)           //如果需要显示负号, 则最高位 (第 4 位) 为符
号位
        num[15:12] <= 4'd11;
    else
        //不需要显示负号时, 则第 4 位不显示任何字
符
        num[15:12] <= 4'd10;
    end
else begin
    //如果显示数据为 2 位十进制数, 则给低 2 位
数码管赋值
    if (data1 || point[1]) begin
        num[ 7: 0] <= {data1,data0};
        //第 6、5、4 位不显示任何字符
        num[23:12] <= {3{4'd10}};
        if(sign)           //如果需要显示负号, 则最高位 (第 3 位) 为符
号位
        num[11:8]  <= 4'd11;
    else
        //不需要显示负号时, 则第 3 位不显示任何字
符

```

```

        num[11:8] <= 4'd10;
    end
    else begin      //如果显示数据为 1 位十进制数, 则给最低位
数码管赋值
        num[3:0] <= data0;
                    //第 6、5 位不显示任何字符
        num[23:8] <= {4{4'd10}};
        if(sign)      //如果需要显示负号, 则最高位(第 2 位)为符
号位
        num[7:4] <= 4'd11;
        else      //不需要显示负号时, 则第 2 位不显示任何字
符
        num[7:4] <= 4'd10;
    end
    end
end
end

//每当计数器对数码管驱动时钟计数时间达 1ms, 输出一个时钟周期的脉冲信号
always @ (posedge dri_clk or negedge rst_n) begin
    if (rst_n == 1'b0) begin
        cnt0 <= 13'b0;
        flag <= 1'b0;
    end
    else if (cnt0 < MAX_NUM - 1'b1) begin
        cnt0 <= cnt0 + 1'b1;
        flag <= 1'b0;
    end
    else begin
        cnt0 <= 13'b0;
        flag <= 1'b1;
    end
end

//cnt_sel 从 0 计数到 5, 用于选择当前处于显示状态的数码管
always @ (posedge dri_clk or negedge rst_n) begin
    if (rst_n == 1'b0)
        cnt_sel <= 3'b0;
    else if(flag) begin
        if(cnt_sel < 3'd5)
            cnt_sel <= cnt_sel + 1'b1;
    end
end

```

```

        else
            cnt_sel <= 3'b0;
    end
    else
        cnt_sel <= cnt_sel;
end

//控制数码管位选信号，使 6 位数码管轮流显示
always @ (posedge dri_clk or negedge rst_n) begin
    if(!rst_n) begin
        seg_sel <= 6'b111111;           //位选信号低电平有效
        num_disp <= 4'b0;
        dot_disp <= 1'b1;             //共阳极数码管，低电平导通
    end
    else begin
        if(en) begin
            case (cnt_sel)
                3'd0 :begin
                    seg_sel <= 6'b111110; //显示数码管最低位
                    num_disp <= num[3:0]; //显示的数据
                    dot_disp <= ~point[0]; //显示的小数点
                end
                3'd1 :begin
                    seg_sel <= 6'b111101; //显示数码管第 1 位
                    num_disp <= num[7:4];
                    dot_disp <= ~point[1];
                end
                3'd2 :begin
                    seg_sel <= 6'b111011; //显示数码管第 2 位
                    num_disp <= num[11:8];
                    dot_disp <= ~point[2];
                end
                3'd3 :begin
                    seg_sel <= 6'b110111; //显示数码管第 3 位
                    num_disp <= num[15:12];
                    dot_disp <= ~point[3];
                end
                3'd4 :begin
                    seg_sel <= 6'b101111; //显示数码管第 4 位
                    num_disp <= num[19:16];
                    dot_disp <= ~point[4];
                end
                3'd5 :begin
                    seg_sel <= 6'b011111; //显示数码管最高位
                end
            endcase
        end
    end
end

```

```

        num_disp <= num[23:20];
        dot_disp <= ~point[5];
    end
    default :begin
        seg_sel  <= 6'b111111;
        num_disp <= 4'b0;
        dot_disp <= 1'b1;
    end
    endcase
end
else begin
    seg_sel  <= 6'b111111;           //使能信号为 0 时，所有数码管均不显示
    num_disp <= 4'b0;
    dot_disp <= 1'b1;
end
end
end

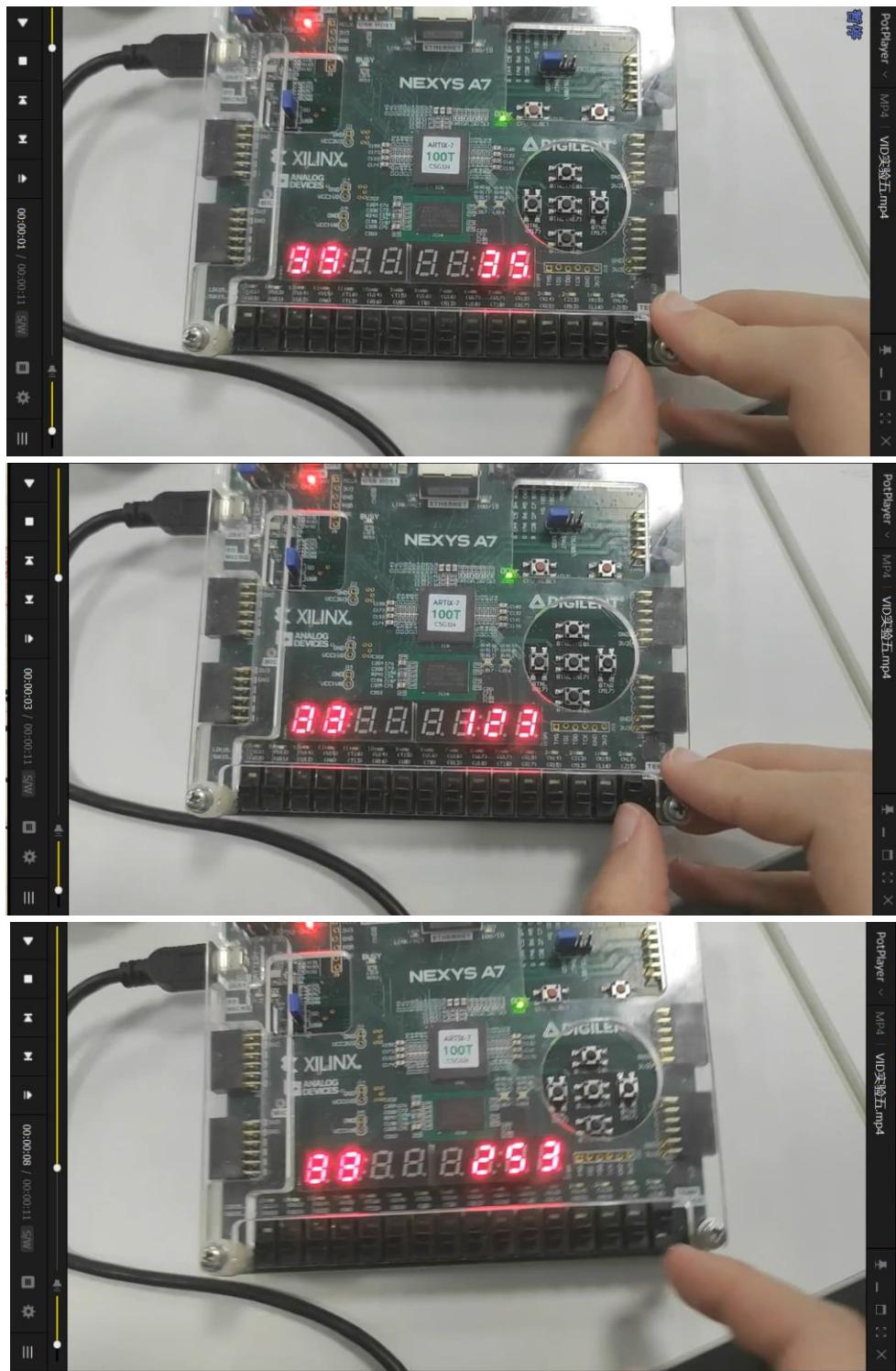
//控制数码管段选信号，显示字符
always @ (posedge dri_clk or negedge rst_n) begin
    if (!rst_n)
        seg_led <= 8'hc0;
    else begin
        case (num_disp)
            4'd0 : seg_led <= {dot_disp,7'b1000000}; //显示数字 0
            4'd1 : seg_led <= {dot_disp,7'b1111001}; //显示数字 1
            4'd2 : seg_led <= {dot_disp,7'b0100100}; //显示数字 2
            4'd3 : seg_led <= {dot_disp,7'b0110000}; //显示数字 3
            4'd4 : seg_led <= {dot_disp,7'b0011001}; //显示数字 4
            4'd5 : seg_led <= {dot_disp,7'b0010010}; //显示数字 5
            4'd6 : seg_led <= {dot_disp,7'b0000010}; //显示数字 6
            4'd7 : seg_led <= {dot_disp,7'b1111000}; //显示数字 7
            4'd8 : seg_led <= {dot_disp,7'b0000000}; //显示数字 8
            4'd9 : seg_led <= {dot_disp,7'b0010000}; //显示数字 9
            4'd10: seg_led <= 8'b11111111;           //不显示任何字符
            4'd11: seg_led <= 8'b10111111;           //显示负号(-)
        default:
            seg_led <= {dot_disp,7'b1000000};
        endcase
    end
end

endmodule

```

五、 实验结果

视频内容包括一般频率计时和快速频率计时：（视频包含在文件夹中）



(图片为视频过程截图)

实验六 状态机实现 LED 流水实验

一、 实验目的

对有限状态机（FSM）做初步了解，用状态机实现 LED 流水

- (1) Gray 编码和 One hot 编码两种状态机；
- (2) 拨码开关作为输入，改变流水灯的方向

二、 实验环境

Verilog HDL:

Verilog HDL 是硬件描述语言（Hardware Description Language）之一，它是一种用于设计数字电路和系统的编程语言。它可以用于描述数字电路的结构和行为，并生成可以被计算机执行的代码。Verilog 是由 Cadence Design Systems 在 1984 年开发的，它的设计目标是提供一个标准的硬件描述语言，以便能够方便地进行数字电路的仿真、综合和测试。

Verilog HDL 可以描述数字电路的结构、功能、时序和处理逻辑。它支持不同的建模级别，包括行为级别、寄存器传输级别、门级别和电路级别。使用 Verilog HDL，可以描述各种数字电路和系统，如单片机、处理器、FPGA、ASIC、通信协议等。

Vivado:

Vivado 是 Xilinx 公司开发的一款综合性的集成电路设计软件，它是一种高级的工具，用于设计和验证 FPGA、SoC 以及三维集成电路（3DIC）等复杂的数字电路和系统。Vivado 支持多种设计语言，包括 Verilog、SystemVerilog、VHDL 等，并且可以与多种开发工具和硬件平台配合使用，如 ModelSim 仿真器、Zynq SoC、UltraScale+ MPSoC 等。在 Vivado 中，设计人员可以使用可视化界面和多种编辑器，以便更好地创建和管理设计资源。此外，Vivado 还提供了一系列的验证和优化工具，如仿真、调试、布局和布线，以及时序约束生成和验证。这些工具可以帮助设计人员更好地分析和优化电路设计，以达到最佳的性能和功耗。

Nexys4 DDR:

Nexys 4 DDR 是一种高性能的 FPGA 开发板，适用于数字系统设计和嵌入式处理器开发。该板配备了 Xilinx Artix-7 FPGA 芯片，具有大容量、高速和低功耗等优点。此外，该开发板还配备了 DDR3 SDRAM、Flash 存储器、USB 接口、以太网接口和多种传感器，以及其他许多实用的外设。

Nexys 4 DDR 支持多种开发环境和语言，如 Vivado、Verilog、VHDL 和 C/C++ 等。此外，它还提供了许多教学资源和实验，可帮助学生和工程师更快地学习和理解数字系统设计和嵌入式开发。Nexys 4 DDR 开发板特别适合用于数字信号处理、通信系统、多媒体处理、机器视觉和人工智能等领域的应用。

三、实验原理

有限状态机是由寄存器组和组合逻辑构成的硬件时序电路，其状态（即由寄存器组的 1 和 0 的组合状态所构成的有限个状态）只可能在同一时钟跳变沿的情况下才能从一个状态转向另一个状态，究竟转向哪一状态还是留在原状态不但取决于各个输入值，还取决于当前所在状态。（这里指的是米里 Mealy 型有限状态机，而莫尔 Moore 型有限状态机究竟转向哪一状态只取决于当前状态。）

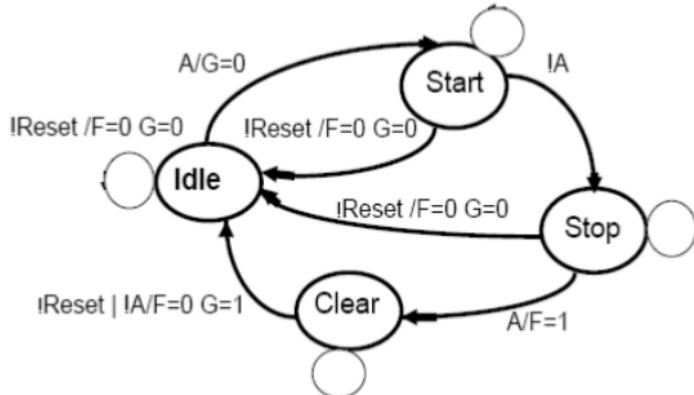


图 4.1、用三种不同编码所实现的状态图

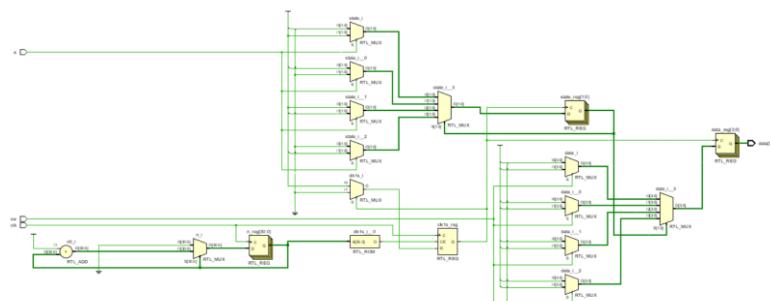
管脚分配表：

程序中管脚名	实际管脚	说明
SW	V17	拨动开关 SW0
DATA[1]	E19	LED LD1
DATA[2]	U19	LED LD2
DATA[3]	V19	LED LD3
DATA[4]	W18	LED LD4

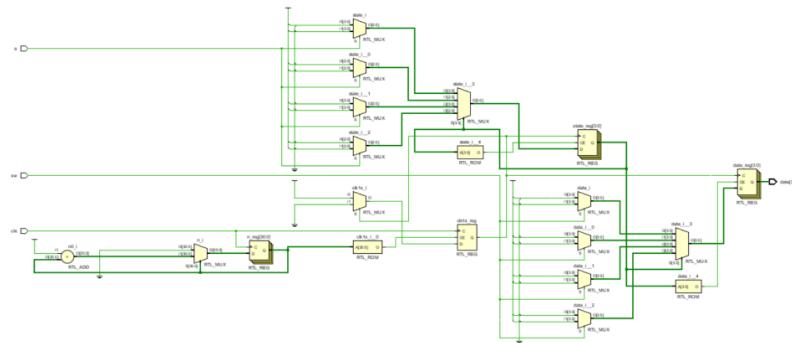
实验结果对照表：

拨动开关 1 脚	LED D1	LED D2	LED D3	LED D4
1	从左至右逐次只亮一个灯			
0	从左至右逐次只灭一个灯			

Grey 码的流水灯原理框图：



One-hot 码的流水灯原理框图：



四、实验内容

Led 代码部分

```
module led(clk,data,sw,change);
input clk,sw,change;
output[3:0] data;
reg clk1s;
parameter max=5000000;
reg[1:0] state=2'b00;
reg[30:0] n;
reg[3:0] data;

always @(posedge clk)begin
if(n==max)begin
  if(!clk1s)clk1s<=1'b1;
  else clk1s<=1'b0;
  n<=0;
end
else n<=n+1;
end

always @(posedge clk1s)begin
  case(state)
  2'b00:begin
    if( change ) state<=2'b11;
    else state<=2'b01;
    if(sw)begin
      data<=4'b1000;
    end
    else begin
      data<=4'b0111;
    end
  end
end
```

```

2'b01:begin
    if( change ) state<=2'b00;
    else state<=2'b10;
    if(sw)begin
        data<=4'b0100;
    end
    else begin
        data<=4'b1011;
    end
end
2'b10:begin
    if( change ) state<=2'b01;
    else state<=2'b11;
    if(sw)begin
        data<=4'b0010;
    end
    else begin
        data<=4'b1101;
    end
end
2'b11:begin
    if( change ) state<=2'b10;
    else state<=2'b00;
    if(sw)begin
        data<=4'b0001;
    end
    else begin
        data<=4'b1110;
    end
end
endcase
end
endmodule

```

代码说明：

always @(posedge clk)begin 的部分实现降频功能

always @(posedge clk1s)begin 的部分实现 LED 流水灯的显示以及换向
定义了两个输入来控制实现不同功能

其中 change 控制的是顺时针和逆时针的变化

其中 sw 控制的是两种不同模式的切换

五、实验结果

视频内容包括两个功能的实现和方向的转换
(视频包含在文件夹中)



VID实验六.mp4

