

实验一 使用 Multisim 实现全加器

一、实验目的

本实验利用 MultiSim 软件作电路图，实现全加器。

二、实验环境

软件：Multisim Ver 14.1

硬件：Nexys A7

三、实验原理

本次实验使用到了异或门、与门、或门等元器件。

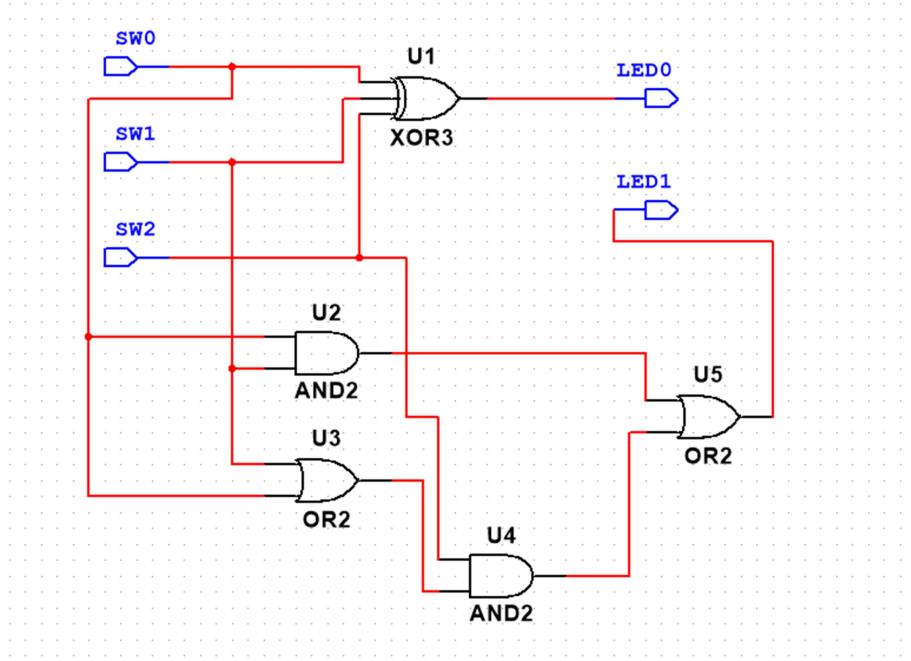
sw0 是加数， sw1 是被加数， sw2 是上一位加法的进位， LED0 是和， LED1 是到下一位的进位。输出函数的逻辑表达式为：

$$LED0 = sw0 \oplus sw1 \oplus sw2$$

$$LED1 = sw0 \cdot sw1 + sw2 \cdot (sw0 + sw1)$$

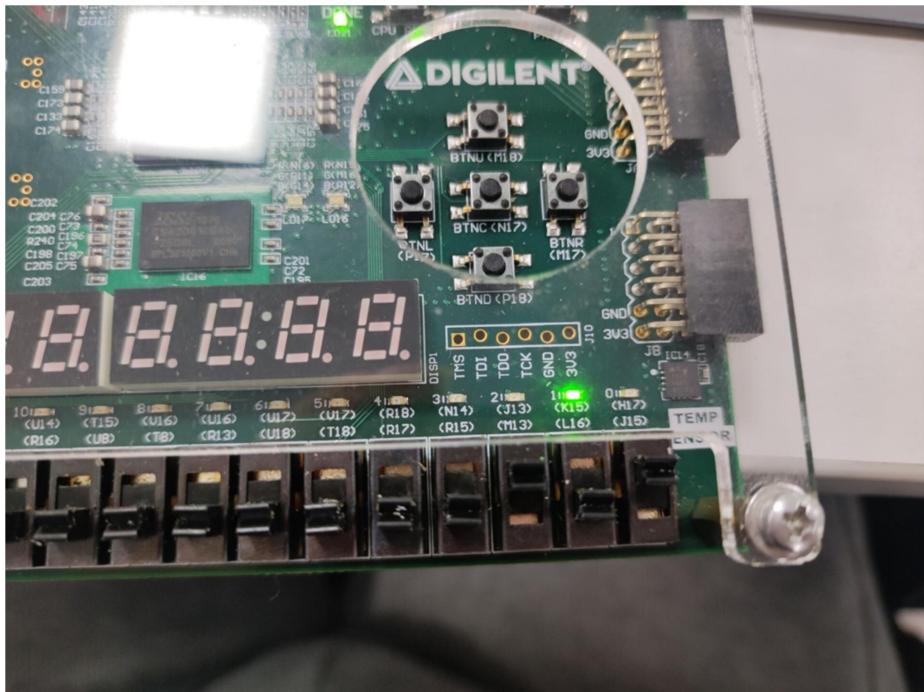
四、实验内容

本次实验的电路图如下所示：

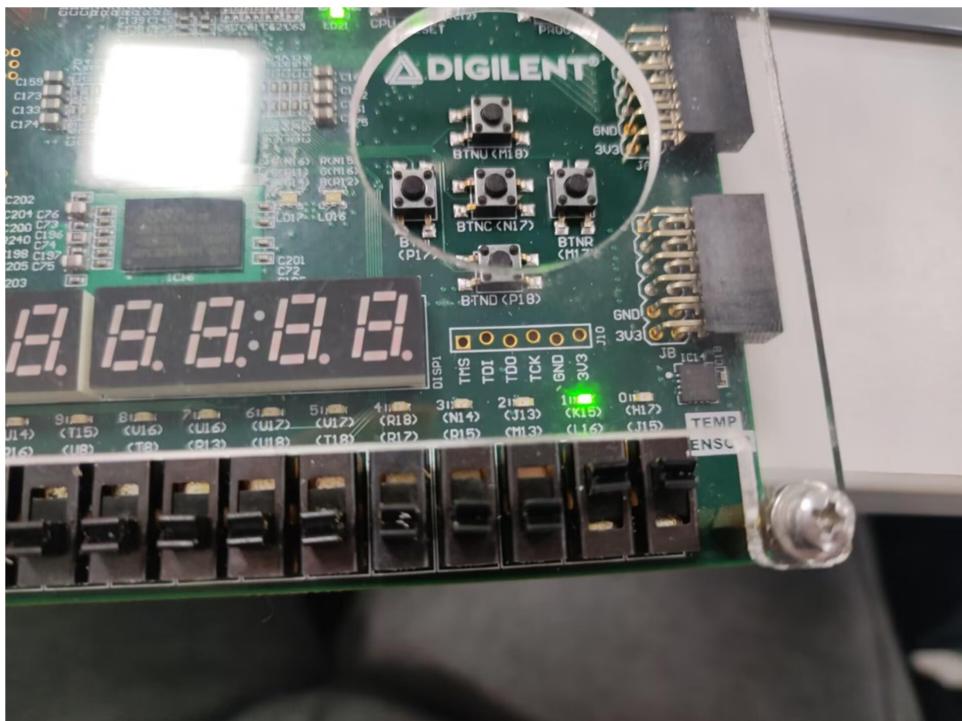


五、实验结果

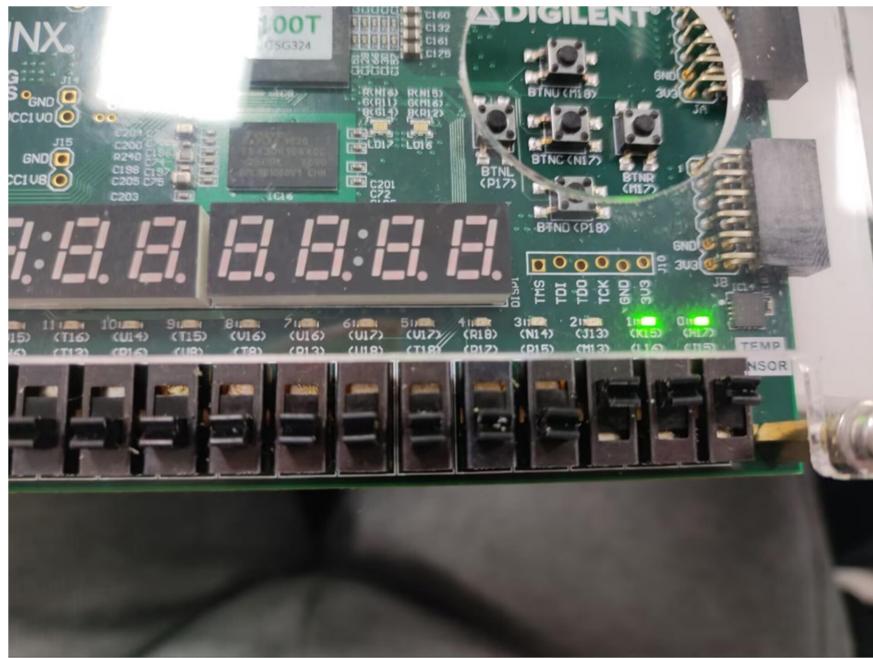
在实验板上运行上述电路图，结果如下所示：



进位为 1 时， $1+0=0$ ，进位 1；



进位为 0 时， $1+1=0$ ，进位 1；



进位为 1 时， $1+1=1$ ，进位 1；
经检验，此加法器工作正常，计算无误。

实验二 4 位比较器扩展为 8 位比较器

一、实验目的

本实验利用 MultiSim 软件作电路图，将 4 位比较器扩展为 8 位比较器。

二、实验环境

软件：Multisim Ver 14.1

硬件：Nexys A7

三、实验原理

本次实验使用了异或非门、与门、非门、或门。实验的思路如下：

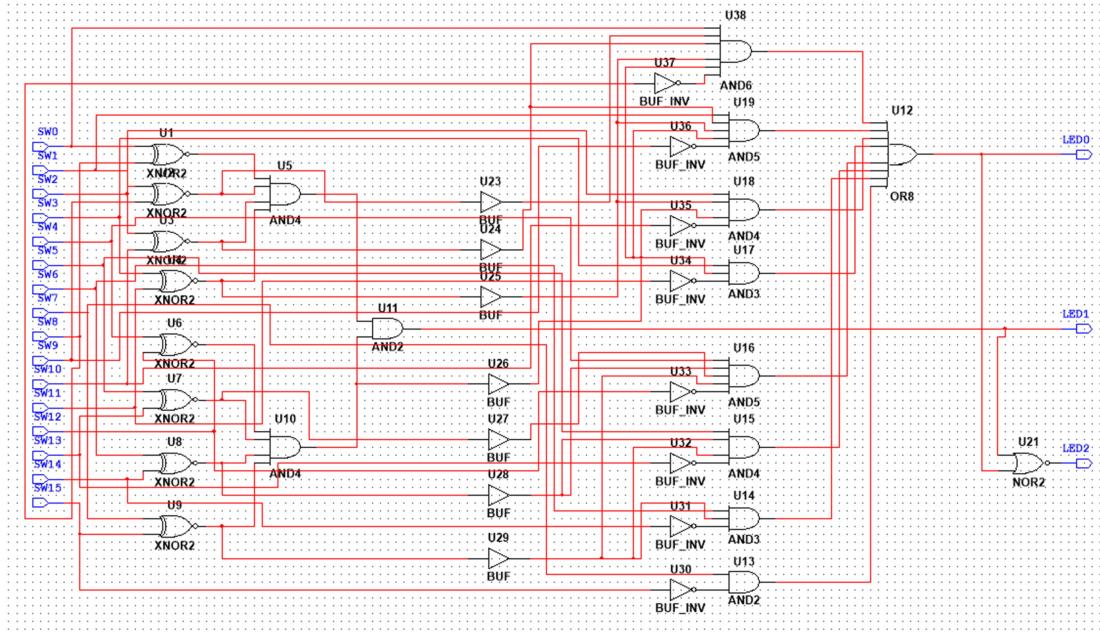
比较器需要同时实现大于判定、等于判定、小于判定。为了方便，我们优先判定等于和大于；如果既不等于又不大于，那么一定小于。

两数各位分别为：sw15-sw8、sw7-sw0。大于判定为：右边数大于左边数。

首先比较最高位：对第一位，如果 sw15=0 且 sw7=1，那么大于判定一定成立；对于其他位，此时的判定条件为：对于该位之前所有的位，两数相等，且在这一位上，左数为 0，且右数为 1。两数的八位均依此做出判断，有一位判定为真时，大于即成立。如果都未成立，且不相等，则小于成立。

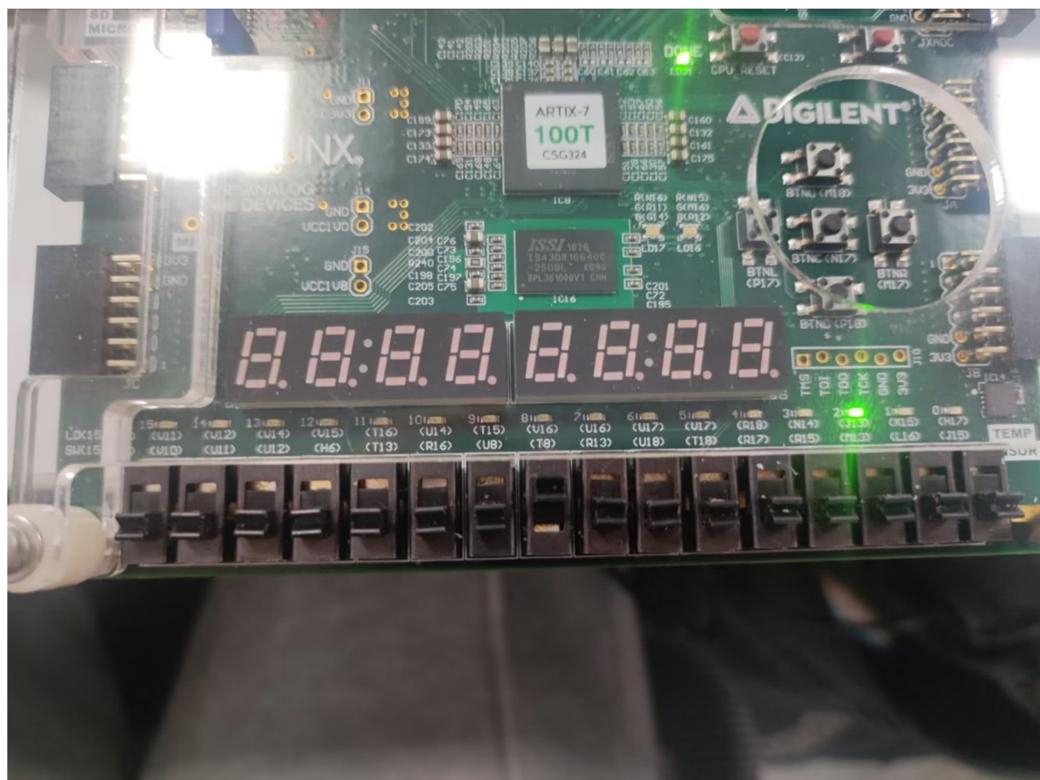
四、实验内容

本次实验的电路图如下所示：

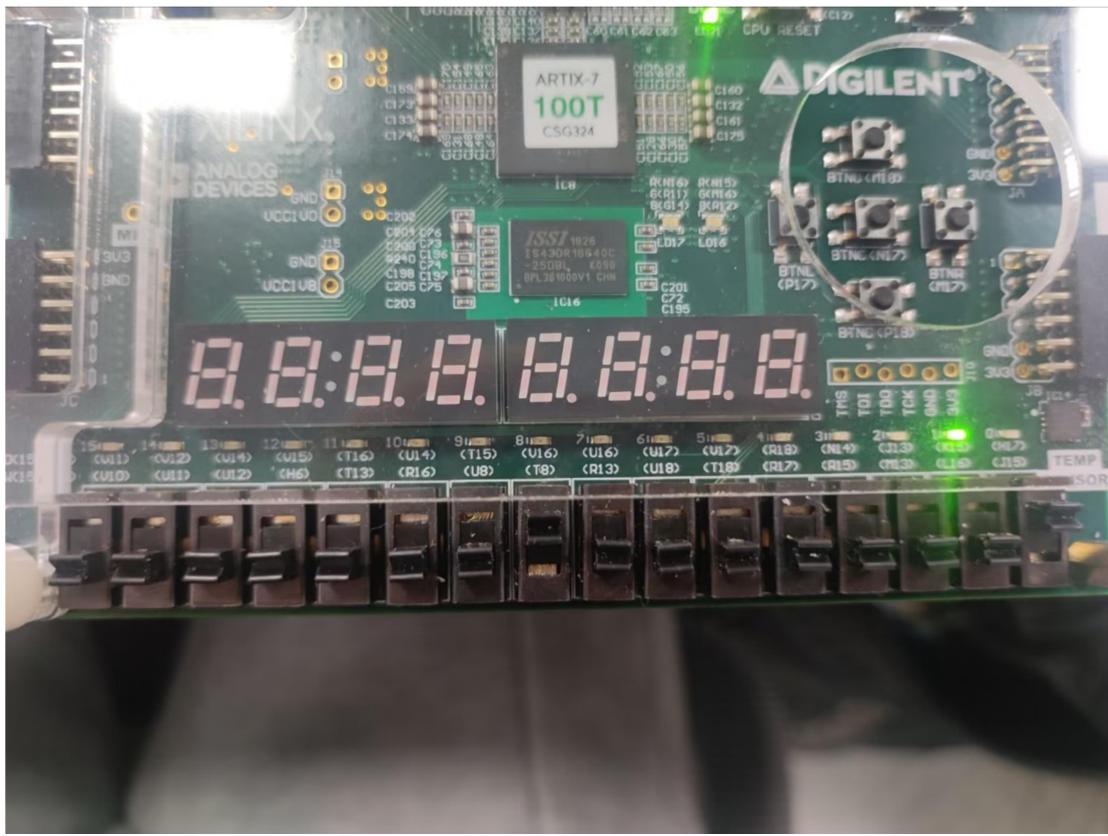


五、实验结果

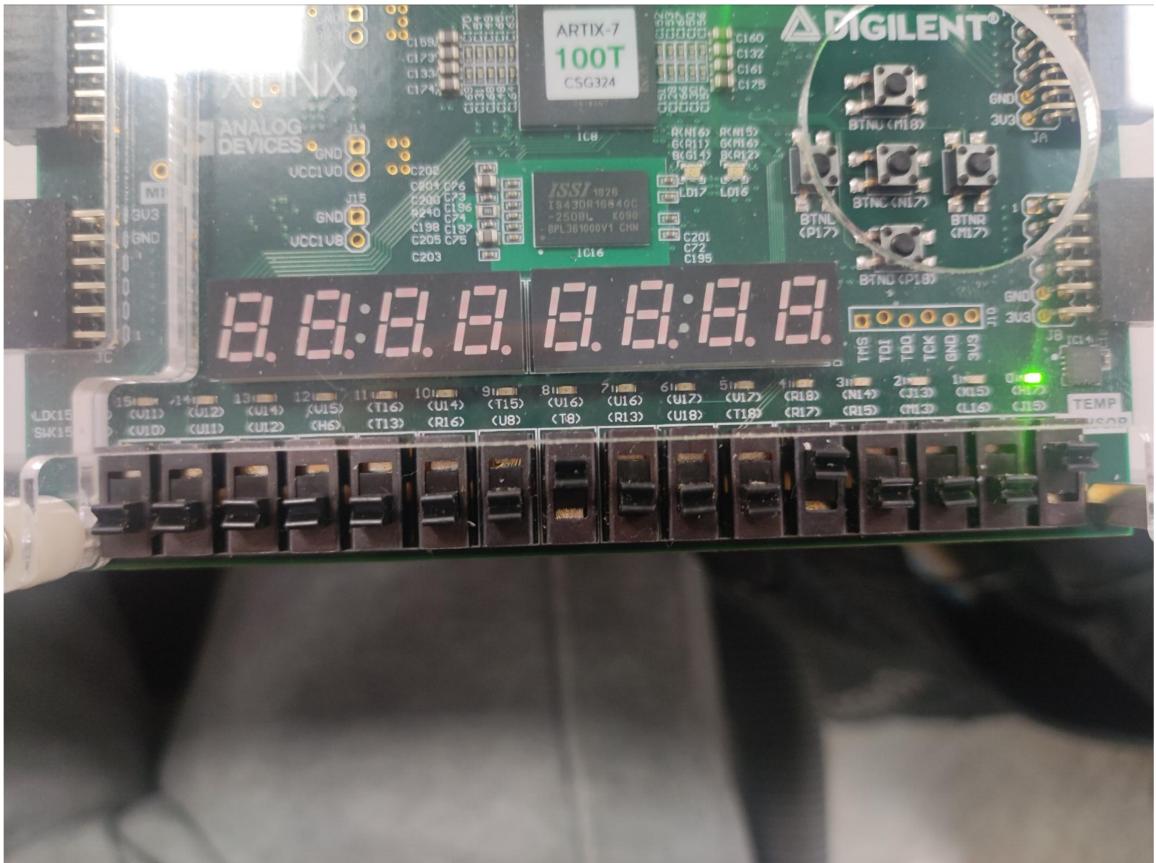
在实验板上运行上述电路图，结果如下所示：



左数（1）大于右数（0），左边灯亮；



左数 (1) 等于右数 (1), 中间灯亮;



左数 (1) 小于右数 (10001), 右边灯亮。

综上：此比较器工作正常，计算无误。

实验三 调整计数器频率

一、实验目的

本实验利用 Multisim 软件作电路图，实现计数器频率的调整。

二、实验环境

软件：Multisim Ver 14.1

硬件：Nexys A7

三、实验原理

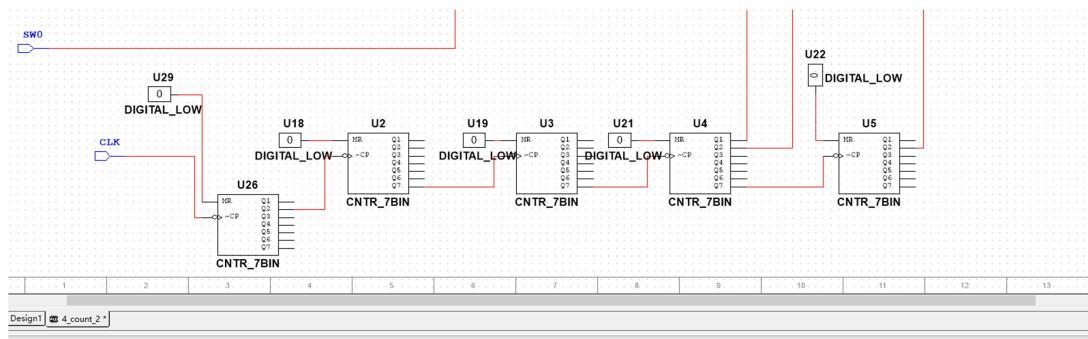
本次实验借助 CNTR_7BIN 计数器对时钟信号进行调整，以达到改变计数器的频率的目的。

将时钟输入接到 CNTR_7BIN 计数器的时钟输入端，输出端作为调整后的时钟信号；通过输出的接口调整，我们可以自行决定计数器频率调整的倍数。

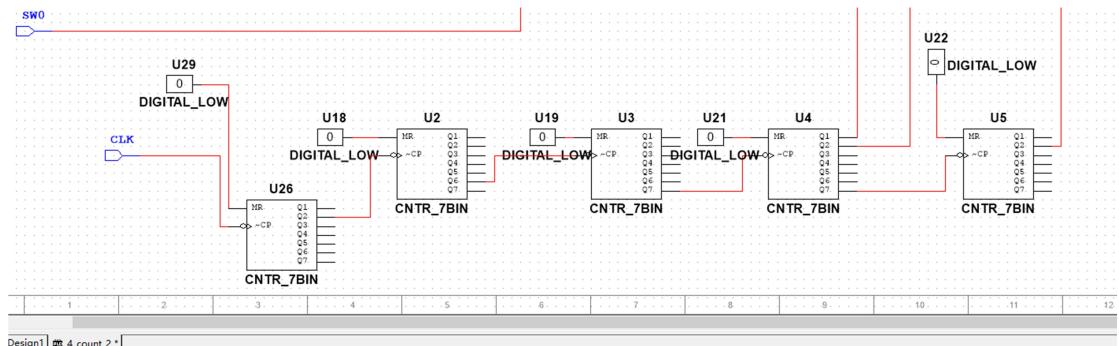
四、实验内容

本次实验的电路图如下所示：

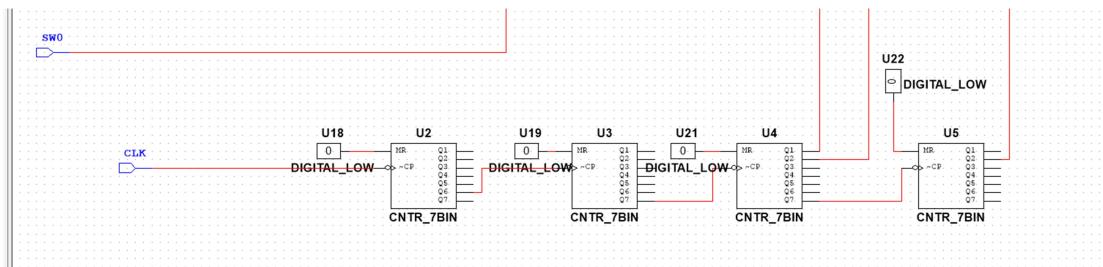
四分之一速：



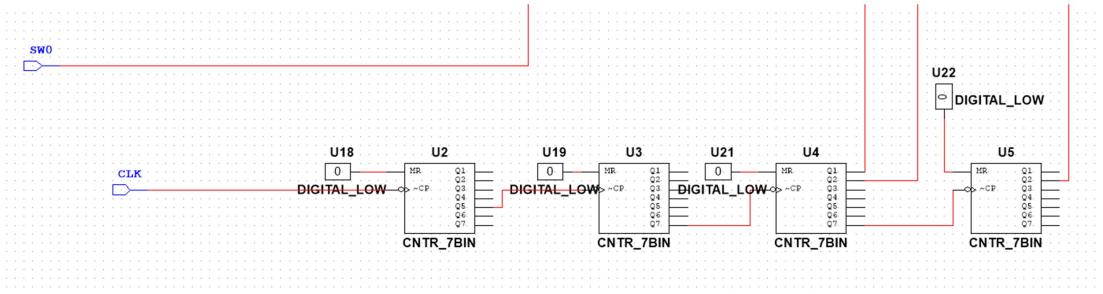
二分之一速：



两倍速:



四倍速:



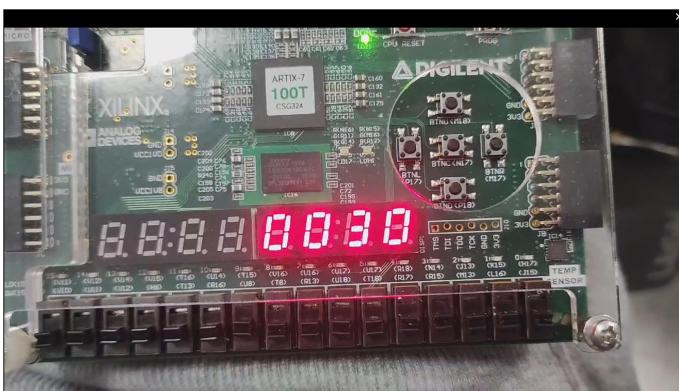
五、实验结果

在实验板上运行上述电路图，结果如下所示（以四分之一速和二分之一速为例）：

四分之一速:



二分之一速:



由于拍摄视频时起始时间和截止时间的限制，其他倍速没有明显的看出倍速的区别；不过从二分之一速和四分之一速的区别可以看出，可以通过调整计数器来改变时钟频率。

实验四 4 位计数器扩展为 8 位计数器

一、实验目的

本实验利用 MultiSim 软件作电路图，实现 4 位计数器扩展为 8 位计数器。

二、实验环境

软件：Multisim Ver 14.1

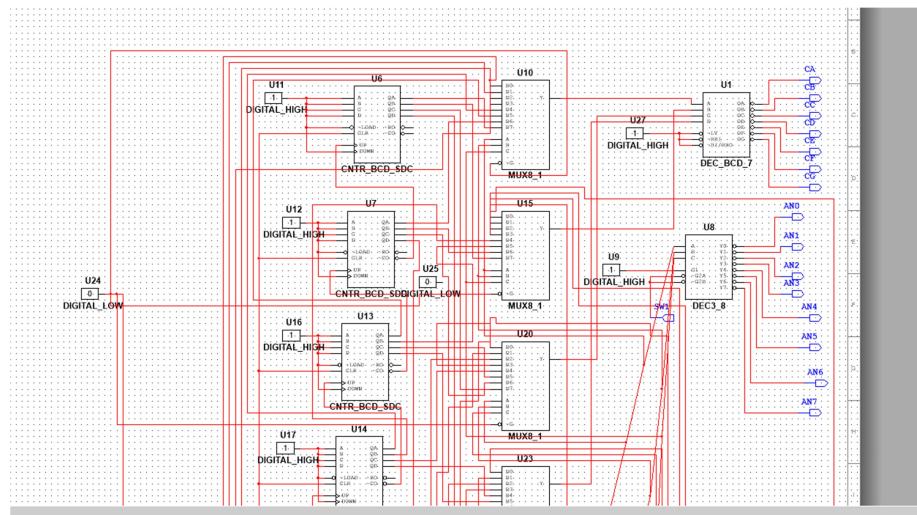
硬件：Nexys A7

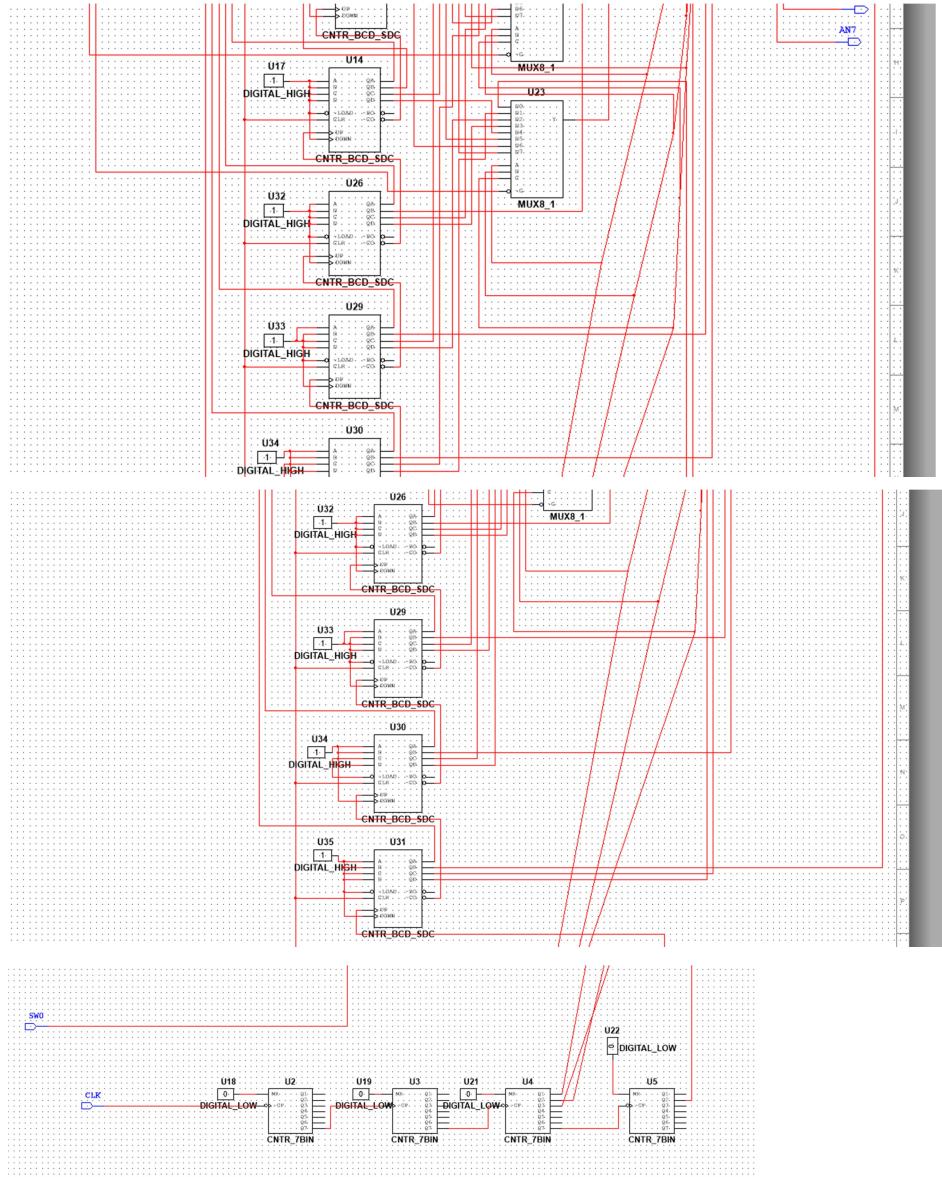
三、实验原理

本次实验使用了 CNTR_7BIN 计数器、CNTR_BCD_SDC 计数器、MUX8_1 选择器、DEC3_8 译码器、DEC_BCD_7 译码器等，具体操作见实验内容板块。

四、实验内容

本次实验的电路图如下所示：





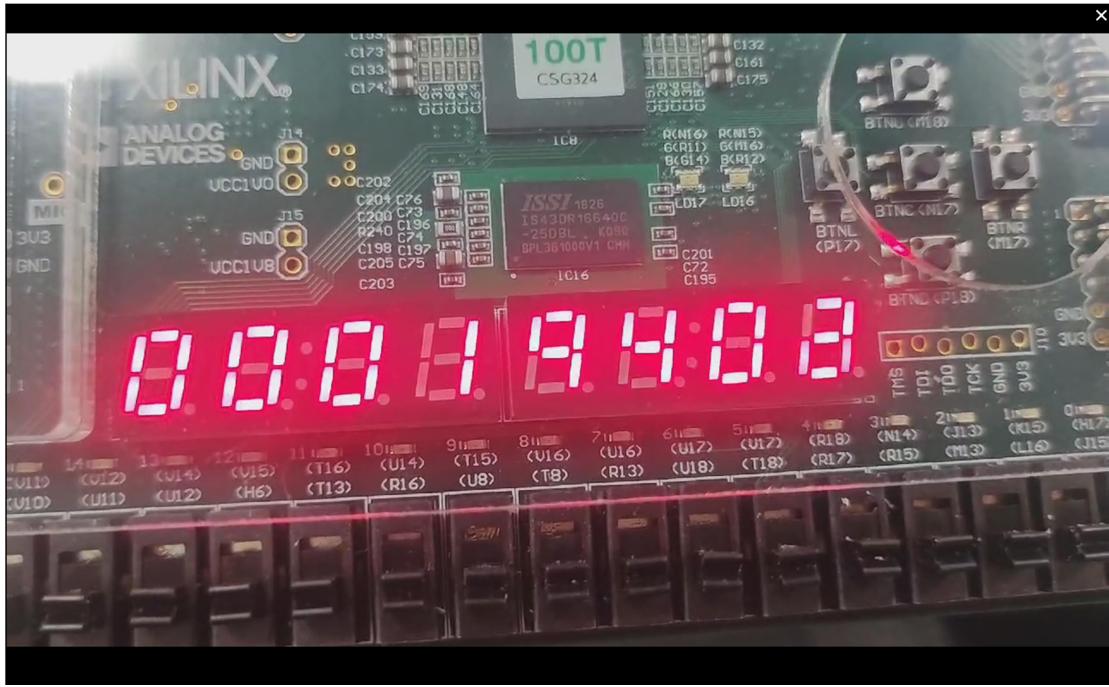
相比于原先的四位计数器，扩展之后的八位计数器做出了如下改动：

- ① 增加了四个 CNTR_BCD_SDC 计数器，作为增加的八位数据的寄存器；
- ② 将原先的 MUX4_1 选择器更换为 MUX8_1 选择器；
- ③ 将 DEC2_4 译码器更换为 DEC3_8 译码器。

以上改动均是为了适应计数器从四位变换到八位的改动。同时，最上面的寄存器保存最高位，四位输出分别接到四个选择器的 D7 端；最下面的寄存器保存最低位，四位输出分别接到四个选择器的 D0 端。中间各个寄存器保存的位数，以及接到选择器的端口则依次递减。

五、实验结果

在实验板上运行上述电路图，结果如下所示：



可以看到，改动后的计数器完成了从四位到八位的扩展。

经检验：扩展后的计数器能正常工作。

实验五 Verilog 实现数据比较器和加法器

一、实验目的

本实验利用 Vivado 软件，使用 Verilog HDL 实现 4 位比较器、8 位比较器、串行加法器的设计。

二、实验环境

软件：Vivado 2017.4

硬件：Nexys A7

三、实验原理

八位比较器由四位比较器扩展实现。第四位先进行比较，输出大小关系；再把低四位的大小关系、两个高四位数作为第二个四位比较器的输入，最终输出即为八进制数的大小关系。

八位加法器由一位全加器实现。将低位进位和两个八进制数的最低位作为第一个全加器的输入，两个输出（和、进位）分别作为最终的输出的最低位和下一个全加器的进位输入。最高位的全加器的进位输出作为整个加法计算结果的输出。

四、实验内容

本次实验所用的的代码如下所示:

四位比较器 com_4:

```
module com_4(
    input [3:0] ia,
    input [3:0] ib,
    input [2:0] icom,
    output [2:0] odata );
    reg [2:0] odata;
    always @*
    begin
        if( ia[3:0] > ib[3:0] ) odata = 3'b001;
        else if ( ia[3:0] < ib[3:0] ) odata = 3'b100;
        else if ( icom[2:0] == 3'b100) odata = 3'b100;
        else if ( icom[2:0] == 3'b001) odata = 3'b001;
        else if ( icom[2:0] == 3'b010) odata = 3'b010;
    end
endmodule
```

八位比较器:

```
module com_8(
    input [7:0] ia,
    input [7:0] ib,
    output [2:0] od
);
    wire [2:0] od;
    wire [2:0] mid;
    wire [2:0] low;
    assign low = 3'b010;
    com_4 U1(ia[3:0], ib[3:0], low[2:0], mid[2:0]);
    com_4 U2(ia[7:4], ib[7:4], mid[2:0], od[2:0]);
endmodule
```

全加器:

```
module FA(
    input ia,
    input ib,
    input ic,
    output os,
    output oc
```

```

);
assign os = ia ^ ib ^ ic;
assign oc = ia & ib | (ia | ib) & ic;
endmodule

```

八位加法器：

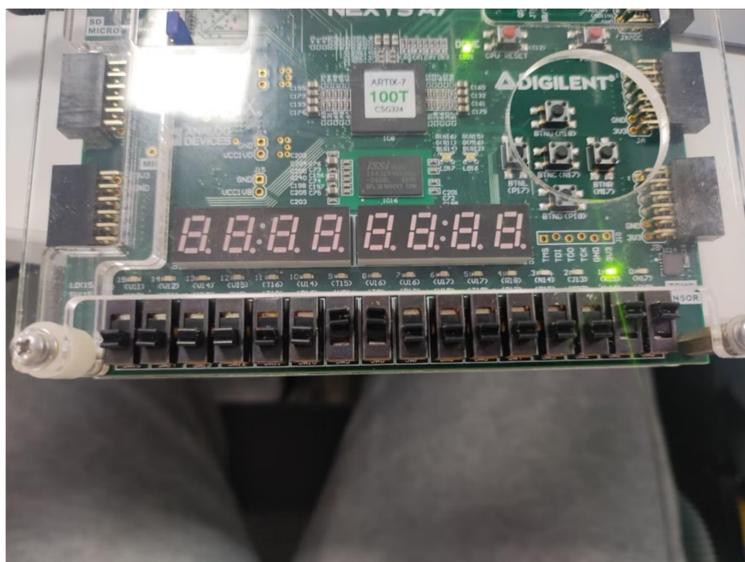
```

module Adder(
input [7:0] ida,
input [7:0] idb,
input ic,
output [7:0] od,
output oc
);
wire [6:0] idc;
FA U1(ida[0], idb[0], ic, od[0], idc[0]);
FA U2(ida[1], idb[1], idc[0], od[1], idc[1]);
FA U3(ida[2], idb[2], idc[1], od[2], idc[2]);
FA U4(ida[3], idb[3], idc[2], od[3], idc[3]);
FA U5(ida[4], idb[4], idc[3], od[4], idc[4]);
FA U6(ida[5], idb[5], idc[4], od[5], idc[5]);
FA U7(ida[6], idb[6], idc[5], od[6], idc[6]);
FA U8(ida[7], idb[7], idc[6], od[7], oc);
endmodule

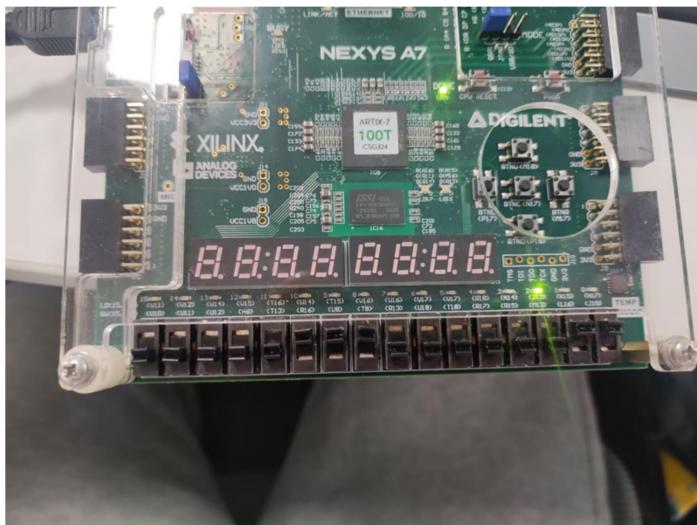
```

五、实验结果

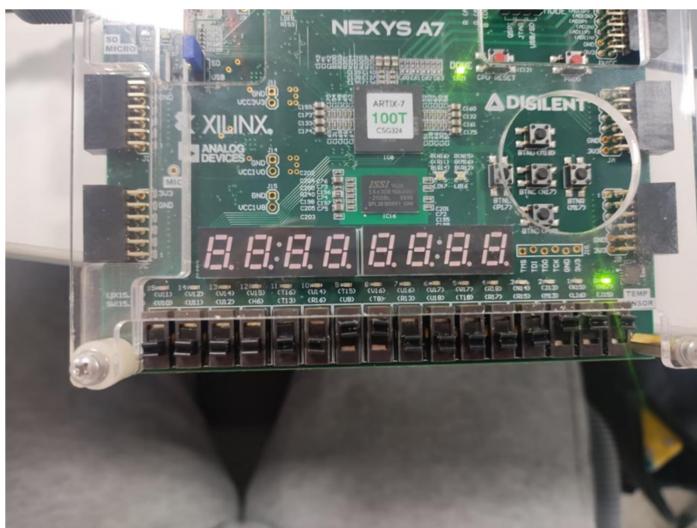
在实验板上运行上述代码，结果如下所示：



高八位（00000011）等于低八位（00000011），中间灯亮；

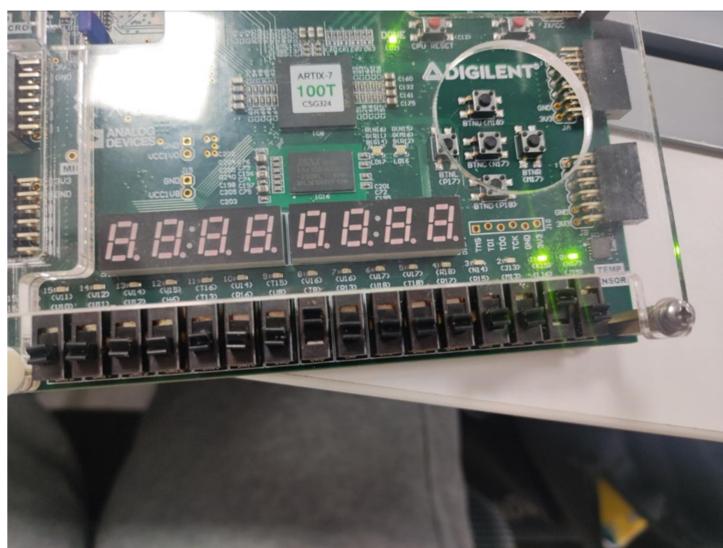


高八位（00000111）大于低八位（00000011），左边灯亮；

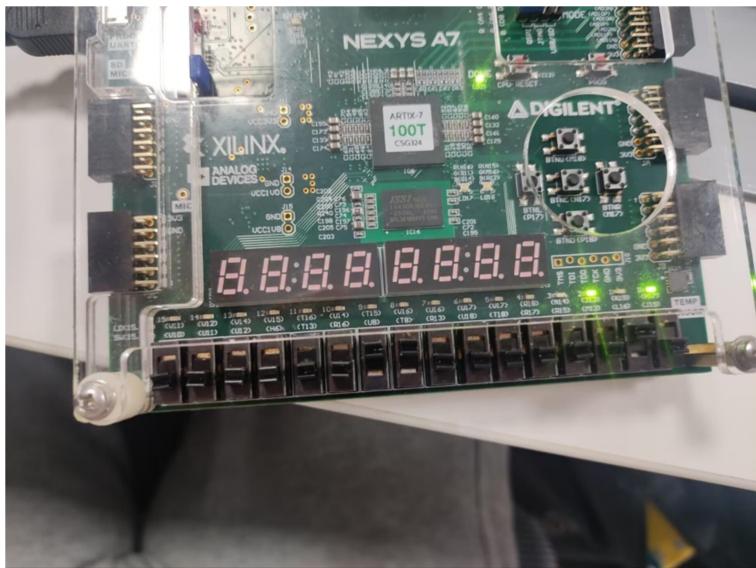


高八位（00000011）小于低八位（00000111），右边灯亮。

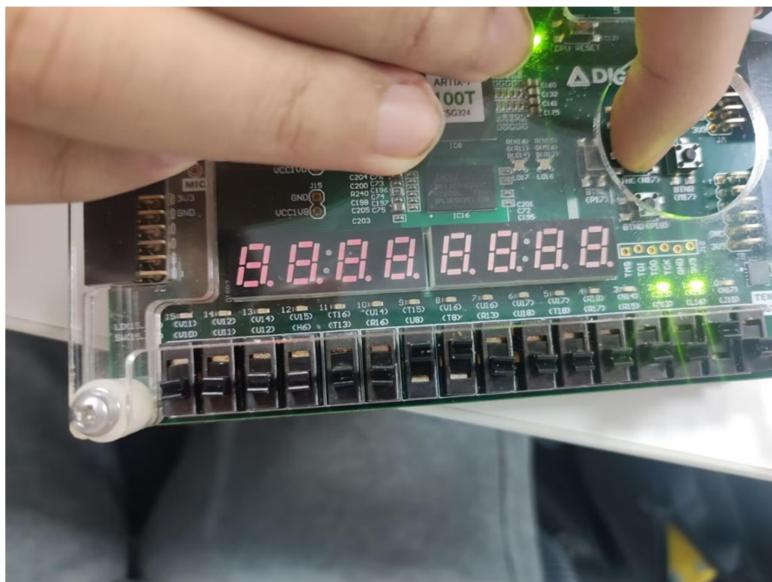
经检验：八位比较器能正常工作。



无低位进位， $01+10=11$ ，无高位进位；



无低位进位， $0011+0010=0101$ ，无高位进位；



有低位进位， $0011+0010=0110$ ，无高位进位。

经检验，八位加法器能正常工作。

实验六 Verilog 实现计数器与分频器

一、实验目的

本实验利用 Vivado 软件，使用 Verilog HDL 实现计数器与分频器的设计。

二、实验环境

软件：Vivado 2017.4

硬件：Nexys A7

三、实验原理

本实验利用三个钟控 JK 触发器，每次时钟上升沿，触发器的值加一，从而达到计数器的效果。同时，通过调整新时钟与旧时钟之间触发器的个数，可以调整时钟的频率，并实现多个频率之间的切换。

四、实验内容

本次实验所用的的代码如下所示：

JK 钟控触发器：

```
module JK(
    input CLK,
    input J,
    input K,
    input rst_n,
    output Q
);
    reg Q;
    always @(posedge CLK or negedge rst_n)
        begin
            if(!rst_n) Q=0;
            else begin
                case({J, K})
                    2'b00:Q=Q;
                    2'b01:Q=0;
                    2'b10:Q=1;
                    2'b11:Q=~Q;
                endcase
            end
        end
endmodule
```

输出到十进制数码管：

```
module display7(
    input [3:0] iData,
    output [6:0] oData
);
    reg [6:0] tData;
    assign oData=tData;
```

```

always @(*)
begin
  case(iData)
    4'b0000: tData=7'b1000000;
    4'b0001: tData=7'b1111001;
    4'b0010: tData=7'b0100100;
    4'b0011: tData=7'b0110000;
    4'b0100: tData=7'b0011001;
    4'b0101: tData=7'b0010010;
    4'b0110: tData=7'b0000010;
    4'b0111: tData=7'b1111000;
    4'b1000: tData=7'b0000000;
    4'b1001: tData=7'b0010000;
    default: tData=7'b1111111;
  endcase
end
endmodule

```

调频：（可以调整为原频率的二的二十四次方分之一、二的二十三次方分之一、二的二十二次方分之一、二的二十一次方分之一）

```

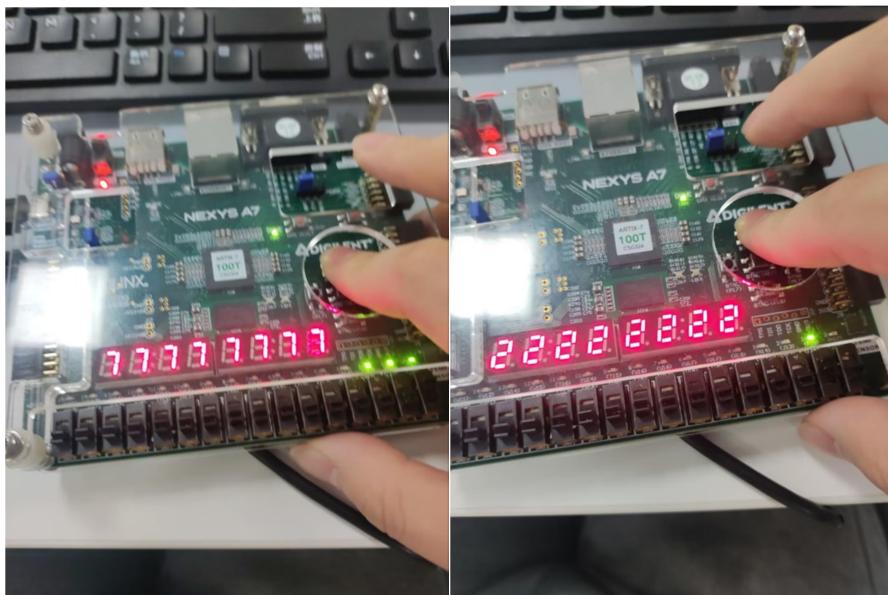
module slow(
  input CLK,
  input [1:0] pace,
  output CLK0
);
  reg [24:0] num;
  reg CLK1;
  assign CLK0 = CLK1;
  always @*
  begin
    case (pace)
      2'b00: CLK1 = num[24];
      2'b01: CLK1 = num[23];
      2'b10: CLK1 = num[22];
      2'b11: CLK1 = num[21];
    endcase
  end
  always @(posedge CLK)
  begin
    num = num + 1;
  end
endmodule

```

```
end  
endmodule
```

五、实验结果

在实验板上运行上述代码，结果如下所示：



经检验，计数器的功能可以无误实现，并可以通过右下角的两个拨片控制时钟频率。

实验七 数码管的多位扫描方式显示

一、实验目的

本实验利用 Vivado 软件，使用 Verilog HDL 实现 8 位计数器，数码管以十进制多位方式显示。

二、实验环境

软件：Vivado 2017.4

硬件：Nexys A7

三、实验原理

LED 有段码和位码之分，所谓段码就是让 LED 显示出“8.”的八位数据，一般

情况下要通过一个译码电路，将输入的 4 位 2 进制数转换为与 LED 显示对应的 8 位段码。位码也就是 LED 的显示使能端，对于共阳级的 LED 而言，高电平使能。要让 8 个 LED 同时工作，显示数据，就是要不停的循环扫描每一个 LED，并在使能每一个 LED 的同时，输入所需显示的数据对应的 8 位段码。虽然 8 个 LED 是依次显示，但是受视觉分辨率的影响，看到的现象是 8 个 LED 同时工作。

多个数码管动态扫描显示，是将所有数码管的相同段并联在一起，通过选通信号分时控制各个数码管的公共端，循环点亮多个数码管，并利用人眼的视觉暂留现象，只要扫描的频率大于 50Hz，将看不到闪烁现象。

通过构造一个计数器模块和一个数码管动态显示模块，我们可以实现这个计数器。

四、实验内容

本次实验所用的的代码如下所示：

计数器模块：

```
module count(
    input          clk ,      // 时钟信号
    input          rst_n,     // 复位信号
    input          path_high, // 频率切换（高位）
    input          path_low,  // 频率切换（低位）

    //user interface
    output reg [26:0]   data ,    // 6 个数码管要显示的数值
    output reg [7:0]    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
reg [31:0]  MAX_NUM;        // 计数器计数的最大值
reg [1:0]   path;

always @* begin
    path <= {path_high, path_low};
    case(path)
```

```

    2' d0: MAX_NUM <= 32'd2500000;
    2' d1: MAX_NUM <= 32'd5000000;
    2' d2: MAX_NUM <= 32'd10000000;
    2' d3: MAX_NUM <= 32'd100000000;
    default: MAX_NUM <= 32'd500000000;
endcase
end

//reg define
reg [31:0] cnt ; // 计数器, 用于计时 100ms
reg flag; // 标志信号

//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 <= 32'b0;
        flag<= 1'b0;
    end
    else if (cnt < MAX_NUM - 1'b1) begin
        cnt <= cnt + 1'b1;
        flag<= 1'b0;
    end
    else begin
        cnt <= 32'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 <= 27'b0;
        point <=8'b00000000;
        en <= 1'b0;
        sign <= 1'b0;
    end
    else begin
        point <= 8'b00000000; //不显示小数点
        en <= 1'b1; //打开数码管使能信号
    end
end

```

```

    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 < 27'd99999999)
            data <= data +1'b1;
        else
            data <= 27'b0;
    end
end

endmodule

数码管动态显示模块:
module seg_led(
    input                  clk      ,      // 时钟信号
    input                  rst_n   ,      // 复位信号

    input [26:0]           data     ,      // 6 位数码管要显示的数值
    input [7:0]             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 [7:0]        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 [31:0]           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
wire [3:0]         data6 ;       // 百万位数
wire [3:0]         data7 ;       // 千万位数

//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 % 4'd10; // 十万位数
assign data6 = data / 20'd1000000 % 4'd10; // 百万位数
assign data7 = data / 24'd10000000 % 4'd10; // 千万位数

//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 <= 32' b0;
    else begin

        if (data7 || point[7]) begin      //如果显示数据为 8 位十进制数, 则给
数码管赋值
            num[31:28] <= data7;
            num[27:14] <= data6;
            num[23:20] <= data5;
            num[19:16] <= data4;
            num[15:12] <= data3;
            num[11:8] <= data2;
            num[ 7:4] <= data1;
            num[ 3:0] <= data0;
        end
        else begin
            if (data6 || point[6]) begin //如果显示数据为 7 位十进制数,
则给低 7 位数码管赋值
                num[27:0] <=
{data6, data5, data4, data3, data2, data1, data0};
                if(sign)
                    num[31:28] <= 4' d11; //如果需要显示负号, 则最高
位 (第 8 位) 为符号位
                else
                    num[31:28] <= 4' d10; //不需要显示负号时, 则第 8
位不显示任何字符
            end
            else begin
                if (data5 || point[5]) begin //如果显示数据为 6 位十进制数, 则给低 6
位数码管赋值
                    num[23:0] <= {data5, data4, data3, data2, data1, data0};
                    num[31:28] <= {4' d10}; //第 8 位不显示任何字符
                    if(sign)
                        num[27:24] <= 4' d11; //如果需要显示负号, 则最高位 (第 7
位) 为符号位
                end
            end
        end
    end
end

```

```

        num[27:24] <= 4'd10; //不需要显示负号时，则第 7 位不显示
任何字符
    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};
        num[31:24] <= {2{4'd10}}; //第 8、7 位不显示任何字符
        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[31:20] <= {3{4'd10}}; //第 8、7、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 8、7、6、5 位不显示任何字符
        num[31:16] <= {4{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 8、7、6、5、4 位不显示任何字符
            num[31:12] <= {5{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 8、7、6、5、4、3 位不显示任何字
符
        num[31:8] <= {6{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
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 7, 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'd7)
            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  <= 8'b11111111;           //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  <= 8'b11111110; //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 <= 8'b11111101; //显示数码管第 1 位
    num_disp <= num[7:4] ;
    dot_disp <= ~point[1];
end
3'd2 :begin
    seg_sel <= 8'b11111011; //显示数码管第 2 位
    num_disp <= num[11:8];
    dot_disp <= ~point[2];
end
3'd3 :begin
    seg_sel <= 8'b11110111; //显示数码管第 3 位
    num_disp <= num[15:12];
    dot_disp <= ~point[3];
end
3'd4 :begin
    seg_sel <= 8'b11101111; //显示数码管第 4 位
    num_disp <= num[19:16];
    dot_disp <= ~point[4];
end
3'd5 :begin
    seg_sel <= 8'b11011111; //显示数码管第五位
    num_disp <= num[23:20];
    dot_disp <= ~point[5];
end
3'd6 :begin
    seg_sel <= 8'b10111111; //显示数码管第六位
    num_disp <= num[27:24];
    dot_disp <= ~point[6];
end
3'd7 :begin
    seg_sel <= 8'b01111111; //显示数码管最高位
    num_disp <= num[31:28];
    dot_disp <= ~point[7];
end
default :begin

```

```

        seg_sel  <= 8'b11111111;
        num_disp <= 4'b0;
        dot_disp <= 1'b1;
    end
endcase
end
else begin
    seg_sel  <= 8'b11111111;           //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

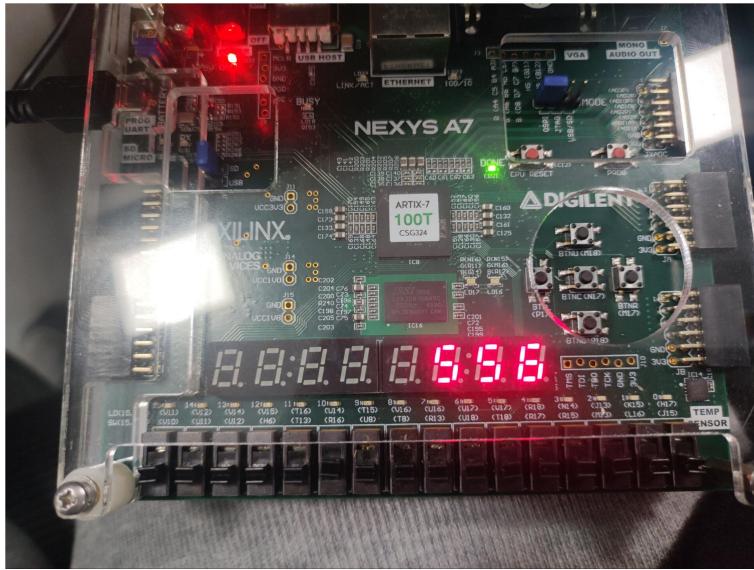
//控制数码管段选信号, 显示字符
always @ (posedge dri_clk or negedge rst_n) begin
    if (!rst_n)
        seg_led <= 8'hc0;
    else begin
        case (num_disp)
            4'd0 : seg_led <= {dot_disp, 7'b1000000}; //显示数字 0
            4'd1 : seg_led <= {dot_disp, 7'b1111001}; //显示数字 1
            4'd2 : seg_led <= {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

在实验板上运行上述代码，结果如下所示：



可以控制计数器只显示低位，而不把高位的 0 补齐。

经检验，计数器的功能可以无误实现，并可以通过右下角的两个拨片控制时钟频率。

实验八 数字时钟

一、实验目的

本实验利用 Vivado 软件，使用 Verilog HDL 实现数字时钟，数码管以十进制多位方式显示。

二、实验环境

软件：Vivado 2017.4

硬件：Nexys A7

三、实验原理

原理大致与上面的计数器相同，我们只需要准备一个计数器，把小时、分钟、秒分离出来，并且通过数码管动态显示模块将其输出即可。

四、实验内容

本次实验所用的的代码如下所示：

```
module Clock(
    input clk,
    input rst_n,
    output reg [7:0] seg_sel,          // 数码管位选，最左侧数码管为最高位
    output [6:0] oDisplay            // 数码管段选
);

parameter max_num = 32'd100000000;           //计数器，每秒递增
reg [31:0] time_num;                      //计时器
reg [31:0] cnt;                          //精确计算一秒
reg flag;                                //每过一秒标记一次

//初始化
initial begin
    time_num <= 86330;
end

//每一秒 flag 输出一次 1 脉冲
always @ (posedge clk or posedge rst_n) begin
    if (rst_n) begin
        cnt <= 32'b0;
        flag <= 1'b0;
    end
    else if (cnt < max_num - 1'b1) begin
        cnt <= cnt + 1'b1;
        flag <= 1'b0;
    end
    else begin
        cnt <= 32'b0;
        flag <= 1'b1;
    end
end

//flag 为 1 时，计时器递增 1
always @ (posedge clk or posedge rst_n) begin
```

```

    if (rst_n) begin
        time_num <= 32'b0;
    end
    else if (flag) begin
        if(time_num < 32'd86400 )
            time_num <= time_num + 1'b1;
        else
            time_num <= 1'b0;
    end
end

//把计时器转换为时、分、秒
wire [3:0] sec1;
wire [3:0] sec2;
wire [3:0] min1;
wire [3:0] min2;
wire [3:0] hour1;
wire [3:0] hour2;
assign sec1 = (time_num % 60) % 10;
assign sec2 = (time_num % 60) / 10;
assign min1 = (time_num / 60 % 60) % 10;
assign min2 = (time_num / 60 % 60) / 10;
assign hour1 = (time_num / 3600) % 10;
assign hour2 = (time_num / 3600) / 10;

//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
reg [20:0] clk_cnt;
reg dri_clk;
always @(posedge clk or posedge rst_n) begin
    if(rst_n) begin
        clk_cnt <= 4'd0;
        dri_clk <= 1'b1;
    end
    else if(clk_cnt == 999) begin
        clk_cnt <= 4'd0;
        dri_clk <= ~dri_clk;
    end
    else begin
        clk_cnt <= clk_cnt + 1'b1;
    end
end

```

```

        dri_clk <= dri_clk;
    end
end

//把输出的时钟转化为 8421 码
reg [23:0] num;
always @ (posedge dri_clk or posedge rst_n) begin
    if (rst_n)
        num <= 32'b0;
    else begin
        num[23:20] <= hour2;
        num[19:16] <= hour1;
        num[15:12] <= min2;
        num[11:8] <= min1;
        num[ 7:4] <= sec2;
        num[ 3:0] <= sec1;
    end
end

//选择当前展示的数码管
reg [2:0] clk_sel;
always @ (posedge dri_clk or posedge rst_n) begin
    if (rst_n)
        clk_sel <= 3'b0;
    else if (dri_clk) begin
        if(clk_sel < 3'd6)
            clk_sel <= clk_sel + 1'b1;
        else
            clk_sel <= 3'b0;
    end
    else
        clk_sel <= clk_sel;
end

//控制数码管位选信号，使 6 位数码管轮流显示
reg [3:0] num_disp;
always @ (posedge dri_clk or posedge rst_n) begin
    if(rst_n) begin
        seg_sel <= 8'b11111111;           //位选信号低电平有效

```

```

    num_disp <= 4'b0;
end
else begin
    case (clk_sel)
        3'd0 :begin
            seg_sel <= 8'b11111110; //显示数码管最低位
            num_disp <= num[3:0] ; //显示的数据
        end
        3'd1 :begin
            seg_sel <= 8'b11111101; //显示数码管第 1 位
            num_disp <= num[7:4] ;
        end
        3'd2 :begin
            seg_sel <= 8'b11111011; //显示数码管第 2 位
            num_disp <= num[11:8];
        end
        3'd3 :begin
            seg_sel <= 8'b11110111; //显示数码管第 3 位
            num_disp <= num[15:12];
        end
        3'd4 :begin
            seg_sel <= 8'b11101111; //显示数码管第 4 位
            num_disp <= num[19:16];
        end
        3'd5 :begin
            seg_sel <= 8'b11011111; //显示数码管第 5 位
            num_disp <= num[23:20];
        end
    default :begin
        seg_sel <= 8'b11111111;
        num_disp <= 4'b0;
    end
    endcase
end
end

//输出
display7 dis(num_disp, oDisplay);

```

```
endmodule
```

数码管显示转换模块:

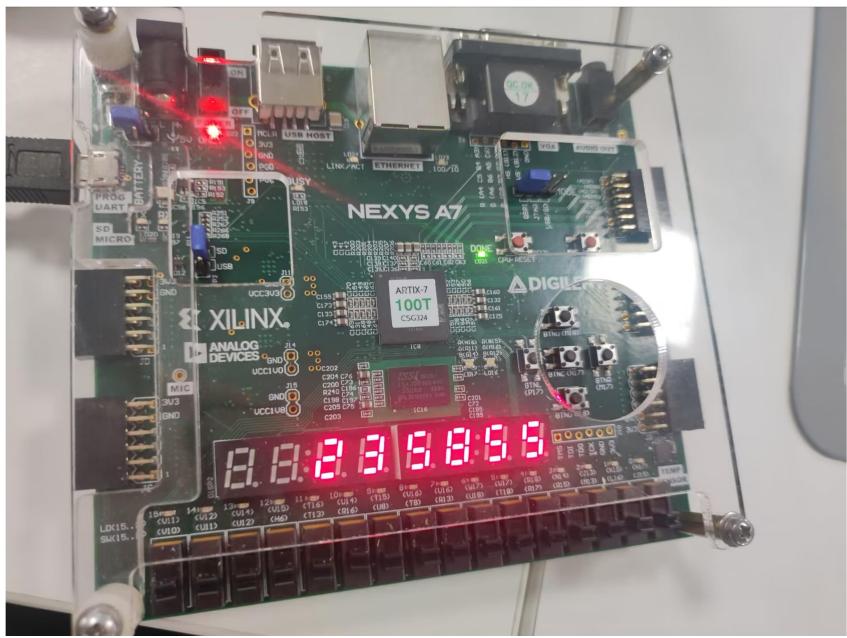
```
module display7(
    input [3:0] iData,
    output [6:0] oData
);

reg [6:0] tData;
assign oData=tData;

always @(*)
begin
    case(iData)
        4' b0000: tData=7' b1000000;
        4' b0001: tData=7' b1111001;
        4' b0010: tData=7' b0100100;
        4' b0011: tData=7' b0110000;
        4' b0100: tData=7' b0011001;
        4' b0101: tData=7' b0010010;
        4' b0110: tData=7' b0000010;
        4' b0111: tData=7' b1111000;
        4' b1000: tData=7' b0000000;
        4' b1001: tData=7' b0010000;
        default: tData=7' b1111111;
    endcase
end
endmodule
```

五、实验结果

在实验板上运行上述代码，结果如下所示：



经检验，时钟的功能可以无误实现，并可以通过右边中间按钮来重置时钟。