

浙江大学

本科实验报告

课程名称:	数字逻辑电路设计
姓 名:	
学 院:	计算机科学与技术学院
专 业:	计算机科学与技术
邮 箱:	
QQ 号:	
电 话:	
指导教师:	洪奇军
报告日期:	2020 年 12 月 22 日

浙江大学实验报告

课程名称：____数字逻辑设计____实验类型：____综合____

实验项目名称：____实验十三——计数器/定时器设计与应用____

学生姓名：____学号：____同组学生姓名：____

实验地点：____紫金港东四 509 室____实验日期：____2020____年____12____月____22____日

一、实验目的

- 1.1 掌握二进制计数器/定时器的工作原理与设计方法
- 1.2 掌握用计数器进行分频的概念和方法
- 1.3 了解计算机中程序计数器（PC）的概念
- 1.4 了解计算机串行数据传送时波特率的概念

二、实验内容和原理

2.1 实验内容

- 设计实现多种进制计数器；
- 设计定时器带报警功能：Timer；
- 设计实现 24 小时挂钟模块：WallClock；
- 集成 WallClock 到实验测试环境 DSTE 测试；

2.2 实验原理

2.2.1 普适计数器

计数器是对指定状态进行遍历的元件，其中指定状态不一定是满足数制序列。

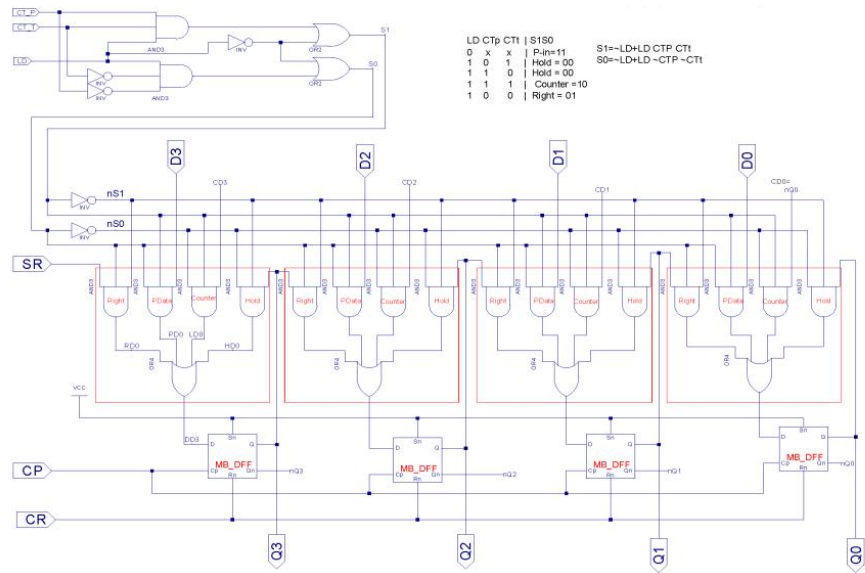
计数器对指定状态的遍历也不一定循环，其中饱和计数器即停止在起始或最后状态的计数器。

模 N 计数器是计数器的一般形式，其循环遍历 N 个固定状态，这 N 个状态序列可以是

任意的。当遍历的序列满足二进制计数序列则为二进制计数器。

2.2.2 常用芯片：74LS161

74LS161 是一种常用的四位二进制可预置的同步加法计数器。其原理图如图表 2.2.2-1。利用 74LS161，可以实现多种计数器。



图表 2.2.2-1 74LS161 原理图

2.2.3 定时：毫秒时钟脉冲

非 2ⁿ 分频需要定时获取分频，定时也是通过计数来获取的。

其中 1ms 定时是计数 1ms 产生一个脉冲，计数至最后一个状态时，将计数器清零并输出一个脉冲。

定时器的核心是计数器，用已知的基准时钟计数构成。其一般有两种实现方式，一是采用计数到某一指定值，二是设定某一时间常数，递减计数至 0。

三、操作方法与实验步骤

3.1 实验设备与材料

- 1. 装有 ISE 14.7 的计算机 1 台
- 2. SWORD 开发板 1 套

3.2 实验步骤

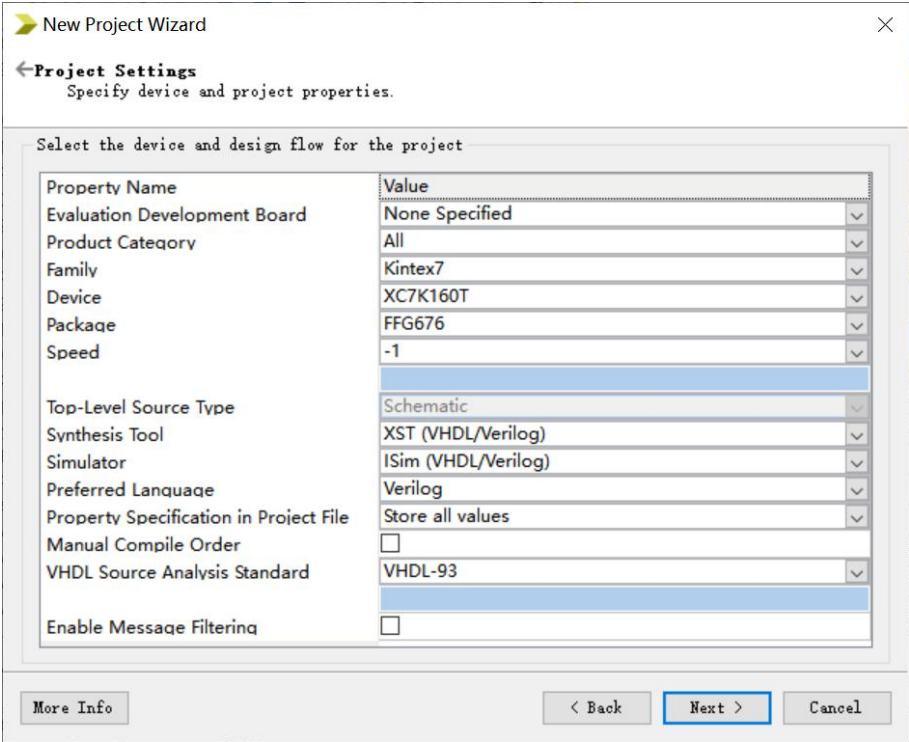
3.2.1 实现毫秒计数器

- 1. 建立工程

打开 ISE Design Suite 14.7，在左上角点击 File，再点击 New Project 命名为 Exp13-Wallclock。

将工程命名，选择 Top-level source type 为 HDL 并选择项目保存位置。

点击 Next 后,出现如图表 3.2.1-1 所示的对话框,如图表所示选择对应的 Family、Device、Package、Speed 等属性,确认无误后一直点击 Next 直至创建工程结束。



The image shows the 'New Project Wizard' dialog box, specifically the 'Project Settings' step. The title bar says 'New Project Wizard' with a close button. Below the title bar, there's a back arrow and the text 'Project Settings' and 'Specify device and project properties.' The main area is titled 'Select the device and design flow for the project'. It contains a table with two columns: 'Property Name' and 'Value'. The properties and their values are: Evaluation Development Board (None Specified), Product Category (All), Family (Kintex7), Device (XC7K160T), Package (FFG676), Speed (-1), Top-Level Source Type (Schematic), Synthesis Tool (XST (VHDL/Verilog)), Simulator (ISim (VHDL/Verilog)), Preferred Language (Verilog), Property Specification in Project File (Store all values), Manual Compile Order (unchecked), VHDL Source Analysis Standard (VHDL-93), and Enable Message Filtering (unchecked). At the bottom, there are three buttons: 'More Info', '< Back', and 'Next >' (which is highlighted with a blue border), and a 'Cancel' button.

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Kintex7
Device	XC7K160T
Package	FFG676
Speed	-1
Top-Level Source Type	Schematic
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

图表 3.2.1-1 项目配置

2. 设计毫秒计数器

在左侧 Sources 窗口空白处右键菜单中选择 New Source。

在对话框中,选择源类型为 Verilog Module,输入文件名 ms1,并且勾选下方 Add to Project。此后,一直点击 Next 直至结束。

此后,双击打开左侧 Sources 窗口中的 ms1.v 文件,输入代码。由于 1ms 不满足 2ⁿ 分频,需要进行计数:

```
`timescale 1ns / 1ps

module ms1(input clk,
           input rst,
           output ms1
);

parameter COUNTER=16;
reg [COUNTER-1:0]count;
reg second_m;
initial count<=0;

always@(posedge clk)begin
    if(rst || (count[15:0]==16'hC34F))begin
        count<=0;
        second_m<=1;
    end
    else begin
        count[15:0]<=count+1;
        second_m<=0;
    end
end
```

```

end

assign ms1=second_m;

endmodule

```

完成后，在 Source 窗口单击选中 ms1.v 文件，然后双击下方 Process 窗口的 Synthesize 下的 Check Syntax 检查语法。

3. 仿真验证

完成后，进行仿真验证。仿真代码如下：

```

`timescale 1ns / 1ps

module ms1_test;

    // Inputs
    reg clk;
    reg rst;

    // Outputs
    wire ms1;

    // Instantiate the Unit Under Test (UUT)
    ms1 uut (
        .clk(clk),
        .rst(rst),
        .ms1(ms1)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 0;

        fork
            forever #10 clk<=~clk;
        begin
            #800; rst=1;
            #100; rst=0;
        end
        join
    end

end

endmodule

```

3.2.2 实现毫秒时钟定时基准并仿真

在 Exp13-Wallclock 工程下，左侧 Sources 窗口空白处右键菜单中选择 New Source。

在对话框中，选择源类型为 Verilog Module，输入文件名 clk_1s，并且勾选下方 Add to Project。此后，一直点击 Next 直至结束。

此后，双击打开左侧 Sources 窗口中的 clk_1s.v 文件，调用上一步完成的 ms1，输入代码如下：

```

`timescale 1ns / 1ps

module clk_1s(input clk,
              input reset,
              output reg [11:0]mms,
              output reg clk_1s

```

```

);

always@(posedge clk)begin
    if(!reset)begin
        mms<=0;
    end
    else begin
        if(mms==12'b100110011001)begin
            mms<=0;
            clk_1s<=1;
        end
        else if(mms[7:0]==8'b10011001)begin
            mms[7:0]<=0;
            mms[11:8]<=mms[11:8]+1;
        end
        else if(mms[3:0]==4'b1001)begin
            mms[3:0]<=0;
            mms[7:4]<=mms[7:4]+1;
        end
        else begin
            mms[3:0]<=mms[3:0]+1;
            clk_1s<=0;
        end
    end
end
end

endmodule

```

完成后，进行仿真验证。仿真代码如下：

```

`timescale 1ns / 1ps

module clk_1s_test;

    // Inputs
    reg clk;
    reg reset;

    // Outputs
    wire [11:0] ms;
    wire clk_1s;

    // Instantiate the Unit Under Test (UUT)
    clk_1s uut (
        .clk(clk),
        .reset(reset),
        .ms(ms),
        .clk_1s(clk_1s)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        reset = 0;

        fork
            forever #10 clk<=~clk;
        begin
            reset=0;
            #1100000;
            reset=1;
        end
    end
endmodule

```

```

        end
    join
end
endmodule

```

3.2.3 设计实现 60 进制分、秒定时基准并仿真

在 Expl3-Wallclock 工程下，左侧 Sources 窗口空白处右键菜单中选择 New Source。

在对话框中，选择源类型为 Verilog Module，输入文件名 count_60，并且勾选下方 Add to Project。此后，一直点击 Next 直至结束。

此后，双击打开左侧 Sources 窗口中的 count_60.v 文件，采用 BCD 码计数，方便后续的输出显示，输入代码如下：

```

`timescale 1ns / 1ps

module count_60(input clk,
                input reset,
                output reg [7:0]six_ten,
                output reg count_carry
);

always@(posedge clk)begin
    if(reset || six_ten==8'b01011001)begin
        count_carry<=1;
        six_ten<=0;
    end
    else if(six_ten[3:0]==4'b1001) begin
        six_ten[7:4]<=six_ten[7:4]+1;
        six_ten[3:0]<=0;
    end
    else begin
        six_ten[3:0]<=six_ten[3:0]+1;
        count_carry<=0;
    end
end
endmodule

```

完成后，进行仿真验证。仿真代码如下：

```

`timescale 1ns / 1ps

module count_60test;

    // Inputs
    reg clk;
    reg reset;

    // Outputs
    wire [7:0] six_ten;
    wire count_carry;

    // Instantiate the Unit Under Test (UUT)
    count_60 uut (
        .clk(clk),
        .reset(reset),
        .six_ten(six_ten),
        .count_carry(count_carry)
    );
endmodule

```

```

);

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;

    fork
        forever #10 clk<=~clk;
    begin
        reset=1;
        #100;
        reset=0;
    end
    join

end

endmodule

```

3.2.4 设计实现 24 进制 ‘时’ 定时基准并仿真

在 Exp13-Wallclock 工程下，左侧 Sources 窗口空白处右键菜单中选择 New Source。

在对话框中，选择源类型为 Verilog Module，输入文件名 count_24，并且勾选下方 Add to Project。此后，一直点击 Next 直至结束。

此后，双击打开左侧 Sources 窗口中的 count_24.v 文件，采用 BCD 码计数，方便后续的输出显示，输入代码如下：

```

`timescale 1ns / 1ps

module count_24(input clk,
                input reset,
                output reg [7:0]two_four,
                output reg count_carry
);

always@(posedge clk)begin
    if(reset || two_four==8'b00100011)begin
        count_carry<=1;
        two_four<=0;
    end
    else if(two_four[3:0]==4'b1001)begin
        two_four[7:4]<=two_four[7:4]+1;
        two_four[3:0]<=0;
    end
    else begin
        two_four[3:0]<=two_four[3:0]+1;
        count_carry<=0;
    end
end

endmodule

```

完成后，进行仿真验证。仿真代码如下：

```

`timescale 1ns / 1ps

module count_24_test;

    // Inputs
    reg clk;

```



```

reg reset;

// Outputs
wire [7:0] two_four;
wire count_carry;

// Instantiate the Unit Under Test (UUT)
count_24 uut (
    .clk(clk),
    .reset(reset),
    .two_four(two_four),
    .count_carry(count_carry)
);

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;

    fork
        forever #10 clk<=~clk;
    begin
        reset=1;
        #100;
        reset=0;
    end
join

end

endmodule

```

3.2.5 集成挂钟模块

1. 建立工程

打开 ISE Design Suite 14.7，在左上角点击 File，再点击 New Project 命名为 Top_Wallclock。

将工程命名，选择 Top-level source type 为 Schematic 并选择项目保存位置。

点击 Next 后，选择对应的 Family、Device、Package、Speed 等属性，确认无误后一直点击 Next 直至创建工程结束，同 3.2.1。

2. 设计时钟模块

新建名为 wallclock 的 Verilog Module 文件，复制设计好的 ms1.v, clk_1s.v, count_60.v, count_24.v 文件到本工程内。在 wallclock.v 内输入代码如下：

```

`timescale 1ns / 1ps

module wallclock(input clk,
                 input reset,
                 input inc,
                 input [2:0]adj_push,
                 output reg [15:0]Time_out,
                 output reg [3:0]s_point,
                 output reg [3:0]t_blink
);

    reg adjust=0;
    reg [1:0]d_state;
    reg t_state;

```

```

    reg d_sec=1, d_min, d_hour;
    wire [7:0]second, minute, hour;
    wire [11:0]msecond;

    ms1 millis(.clk(clk), .rst(reset), .ms1(clk_1ms));
    clk_1s
m_msecond(.clk(clk_1ms), .reset(~reset), .mms(msecond), .clk_1s(clk_1s
)); //produce second
    count_60
m_sec(.clk(clk_1s), .reset(~d_sec), .six_ten(second), .count_carry(clk
_1min)); //produce minute
    count_60
m_min(.clk(clk_1), .reset(reset), .six_ten(minute), .count_carry(clk_1
hour)); //produce hour
    count_24
m_hour(.clk(clk_2), .reset(reset), .two_four(hour), .count_carry(clk_1
day)); //produce day

    always@(posedge adj_push[2])
        adjust<=~adjust;

    always@(posedge adj_push[0])
        if(!adjust)d_state<=d_state+1;
        else t_state<=t_state+1;

    assign clk_1= (d_min & inc) | (!d_min & clk_1min);
    assign clk_2= (d_hour & inc) | (!d_hour & clk_1hour);

    always@*begin
        case(d_state)
            2'b00:begin Time_out<={minute[7:0],second[7:0]};
                        s_point<={second[0],second[0],2'b00};
                        end
            2'b01:begin Time_out<={hour[7:0],minute[7:0]};
                        s_point<={2'b00,second[0],second[0]};
                        end
            2'b10:begin
Time_out<={second[3:0],msecond[11:8],msecond[7:4],msecond[3:0]};
s_point<={second};
                        end
            2'b11:begin
Time_out<={second[3:0],msecond[11:8],msecond[7:4],msecond[3:0]};
s_point<={4'b0000};
                        end
        endcase
        if(!adjust)t_blink<=4'b0000;
        else begin
            case({d_state[0],t_state})
                2'b00:begin t_blink<=4'b0011; d_sec<=~adj_push[1]; end
                2'b01:begin t_blink<=4'b1100; d_min<=adj_push[1]; end
                2'b10:begin t_blink<=4'b1100; d_hour<=adj_push[1]; end
                2'b11:begin t_blink<=4'b0011; d_min<=adj_push[1]; end
            endcase
        end
    end
endmodule

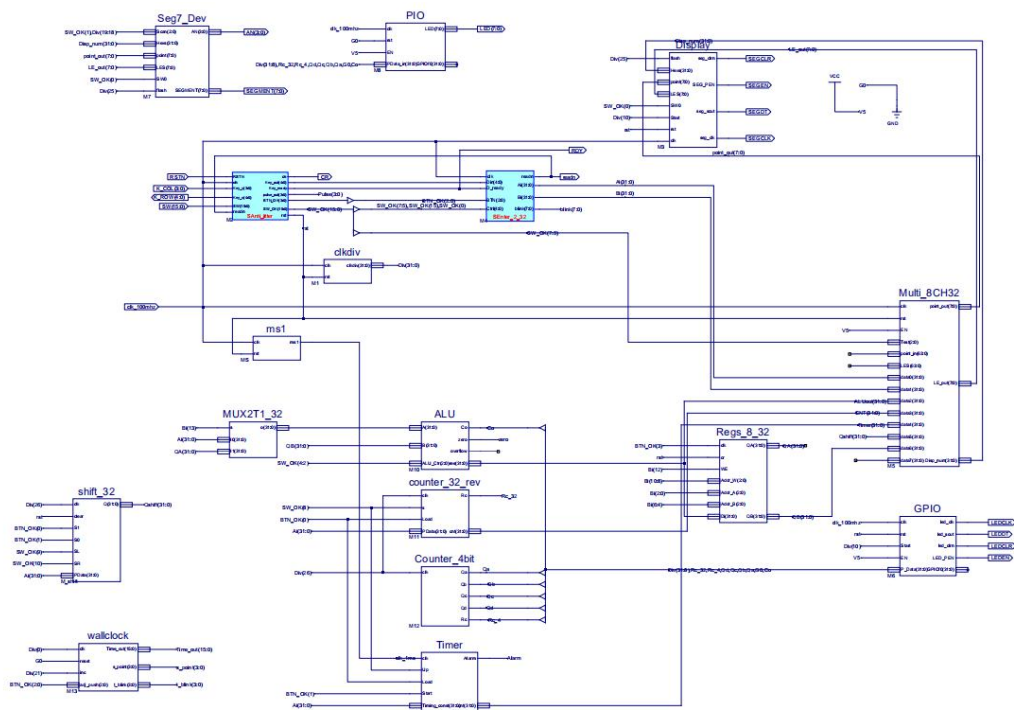
```

完成后，在 Source 窗口单击选中 wallclock.v 文件，然后双击下方 Process 窗口的 Synthesize 下的 Check Syntax 检查语法。语法无误后，点击 Design Utility 中的 Create Schematic Symbol 生成逻辑符号。

3. 集成到实验环境

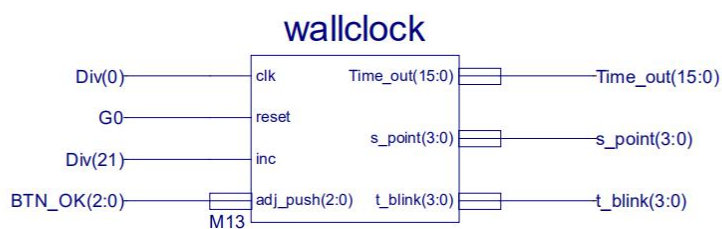
复制实验十二中含有 ALU、Counter_4bit、counter_32_rev 以及 Regs_8_32 的 Framework。
加入本次实验完成的 wallclock。

加入后的原理图如图表 3.2.3-1。

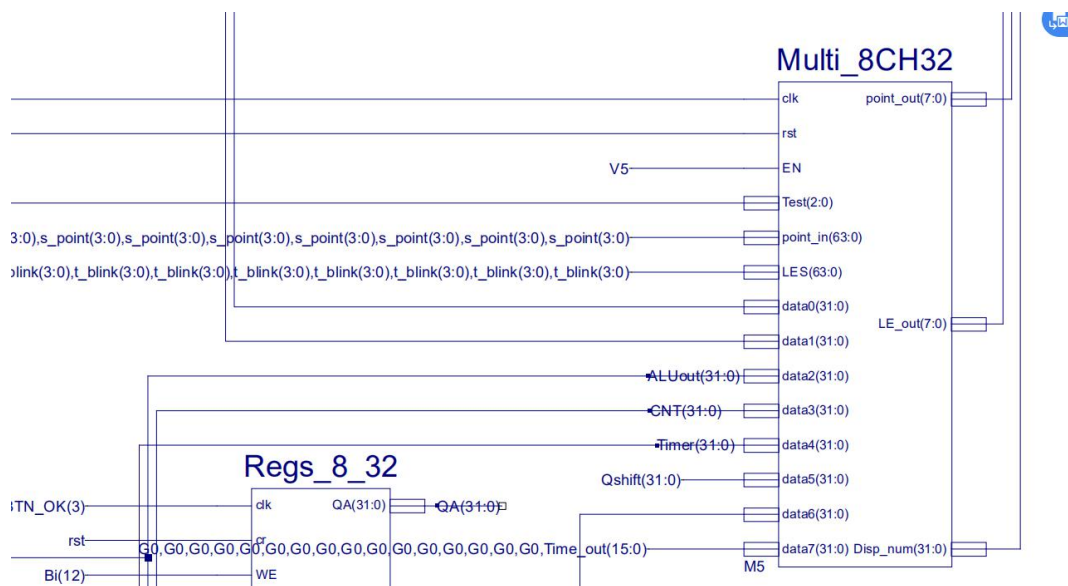


图表 3.2.3-1 Top_Wallclock

其中相比原框架需要修改的部分如图表 3.2.3-2 和 3.2.3-3。



图表 3.2.3-2 修改部分 1



图表 3.2.3-3 修改部分 2

后续步骤则为检查文件，生成比特流文件，并将文件下载到实验板，在实验板物理运行，不再赘述。

四、实验结果与分析

4.1 毫秒时钟定时基准仿真验证

为了验证方便，仿真时我在 clk_1s.v 内调用了 ms1 处理了 clk 信号，即 clk_1s 会每隔 1 毫秒计数一次（不考虑延时）。代码如下：

```
`timescale 1ns / 1ps

module clk_1s(input clk,
              input reset,
              output reg [11:0]ms,
              output reg clk_1s
);

ms1 clkms(.clk(clk), .rst(~reset), .ms1(clk_1ms));

always@(posedge clk_1ms)begin
    if(!reset)begin
        ms<=0;
    end
    else begin
        if(ms==12'b100110011001)begin
            ms<=0;
            clk_1s<=1;
        end
    end
end
```

```

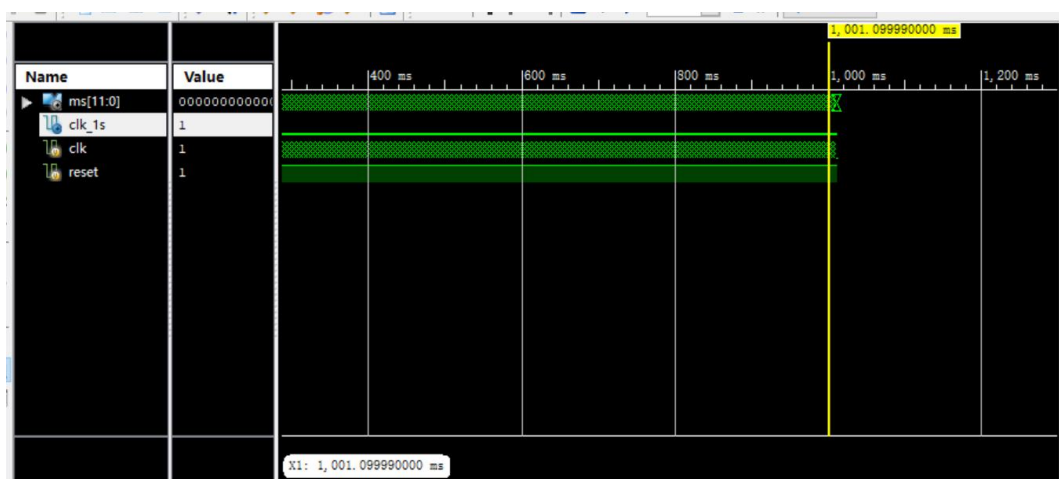
end
else if(ms[7:0]==8'b10011001)begin
    ms[7:0]<=0;
    ms[11:8]<=ms[11:8]+1;
end
else if(ms[3:0]==4'b1001)begin
    ms[3:0]<=0;
    ms[7:4]<=ms[7:4]+1;
end
else begin
    ms[3:0]<=ms[3:0]+1;
    clk_1s<=0;
end
end
end
endmodule

```

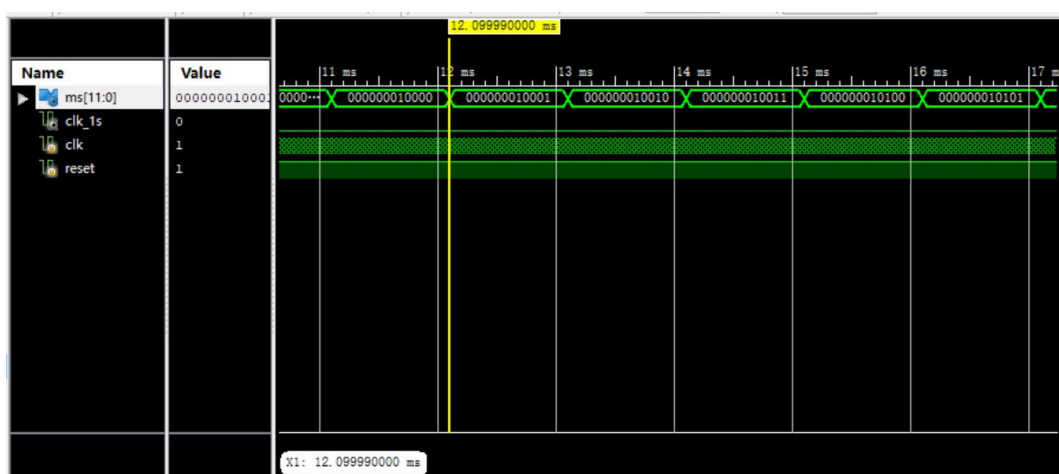
图表 4.1-1 可见，根据代码，在大约 1s 时，clk_1s 第一次输出了一个正脉冲。考虑到延时的存在，iSIM 在约 1001ms 时产生正脉冲是正常的。

如图表 4.1-2，可见 clk_1s 确实是每隔约 1ms 计数一次。结果无误。

由仿真结果可知，clk_1s 与 ms1 设计正确。



图表 4.1-1 毫秒时钟定时基准仿真模拟实验结果 1



图表 4.1-2 毫秒时钟定时基准仿真模拟实验结果 2

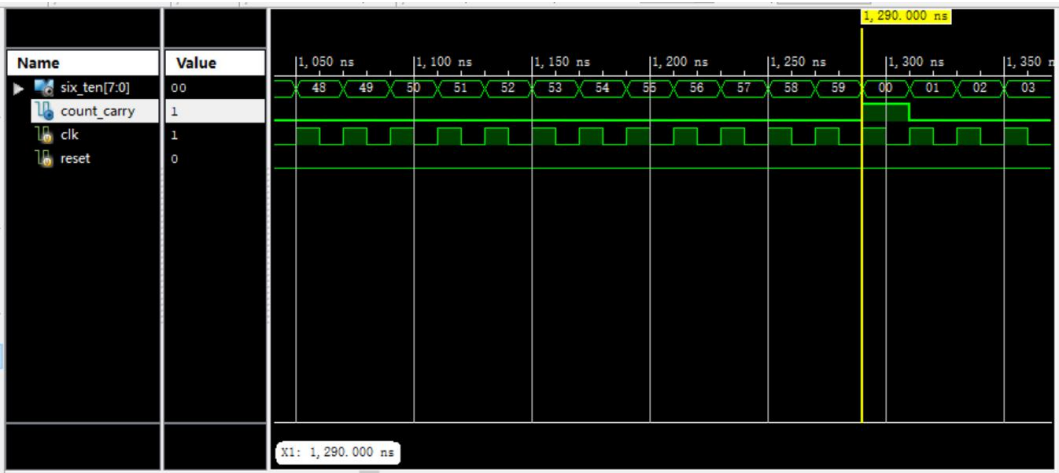
4.2 60 进制分、秒定时基准仿真验证

如图表 4.2-1 可见，根据代码，count_60 每隔 20ns 计数一次。将 six_ten 设置为按照十六进制数显示后，可见其值的变化与十进制下相同。

在其计数至 0x59 后，可见其下一次变化时 six_ten 清零且输出了一个正脉冲。

如图表 4.2-2，reset 正脉冲内的第一个时钟上升沿出现时，six_ten 被清零且输出一个正脉冲。可知重置功能正常。

由仿真结果可知，设计正确。



图表 4.2-1 60 进制分、秒定时基准仿真模拟实验结果 1



图表 4.2-2 60 进制分、秒定时基准仿真模拟实验结果 2

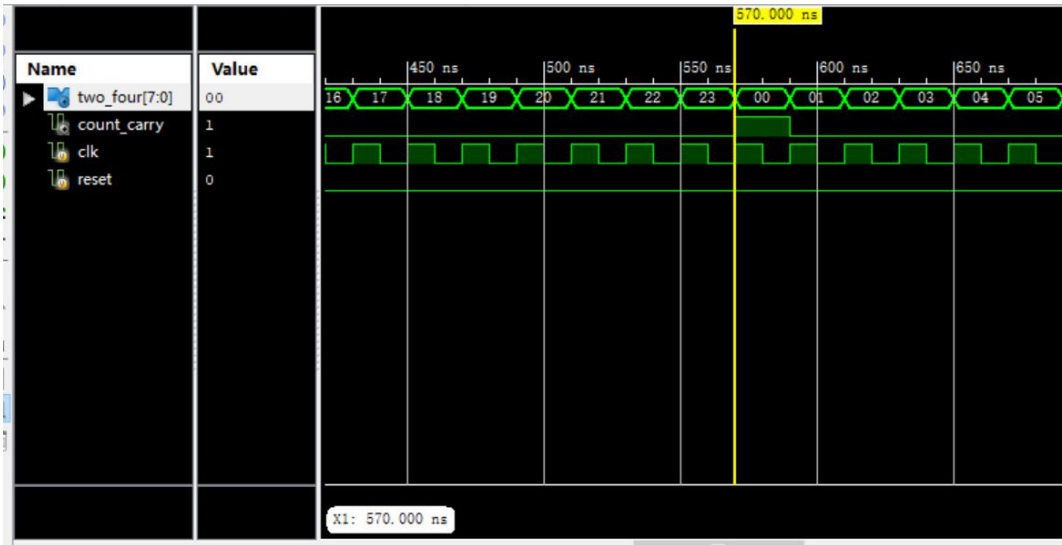
4.3 24 进制‘时’定时基准仿真验证

如图表 4.3-1 可见，根据代码，count_24 每隔 20ns 计数一次。将 two_four 设置为按照十六进制数显示后，可见其值的变化与十进制下相同。

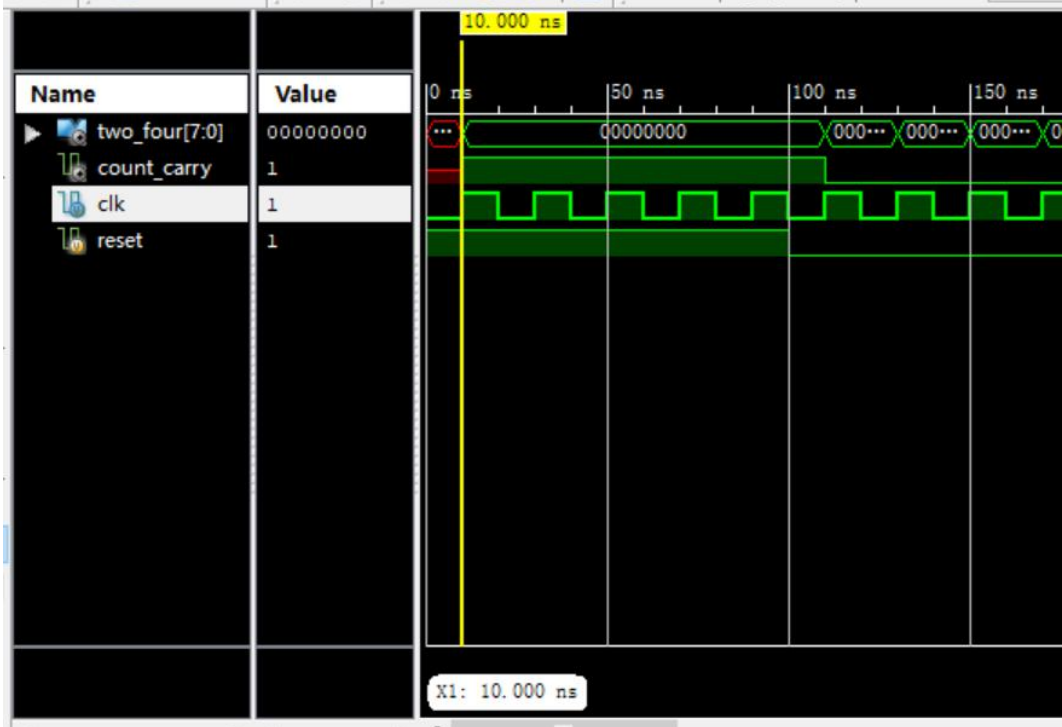
在其计数至 0x23 后，可见其下一次变化时 two_four 清零且输出了一个正脉冲。

如图表 4.3-2，reset 正脉冲内的第一个时钟上升沿出现时，two_four 被清零且输出一个正脉冲。可知重置功能正常。

由仿真结果可知，设计正确。



图表 4.3-1 24 进制‘时’定时基准仿真模拟实验结果 1



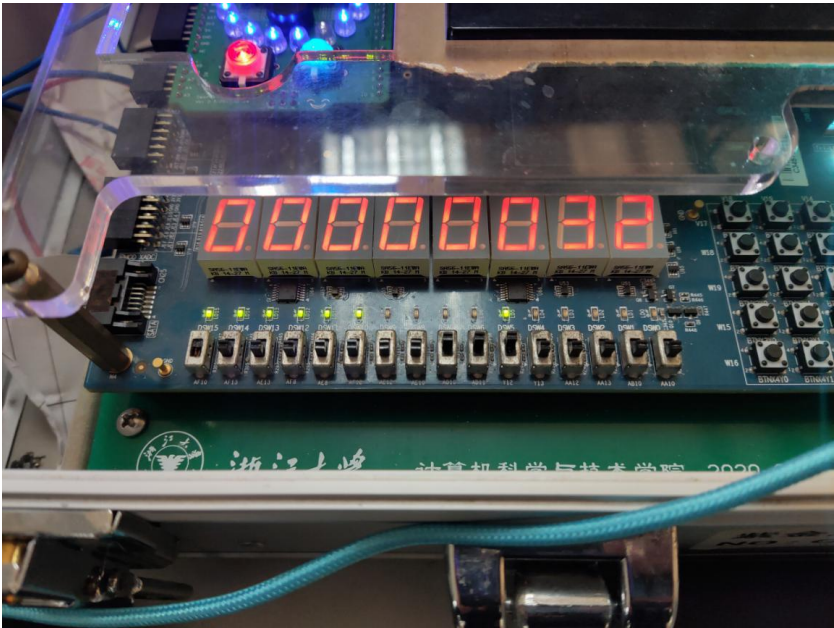
图表 4.3-2 24 进制‘时’定时基准仿真模拟实验结果 2

4.4 时钟模块物理验证

图表 4.4-1 验证结果

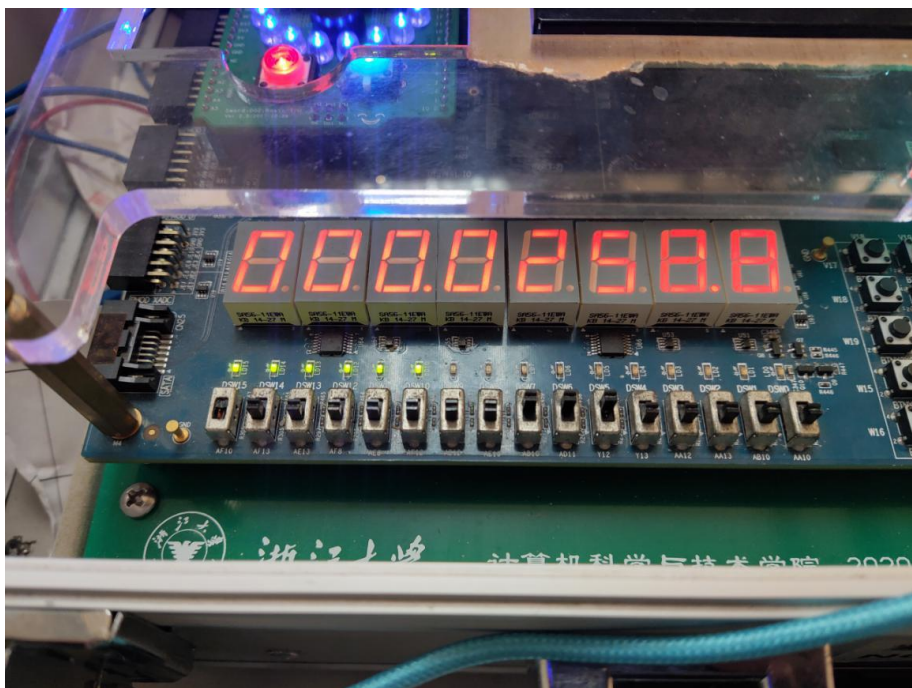
	显示模式	校准模式
BTN[0]	分：秒显示、时：分显示、秒：毫秒 显示之间循环切换	切换校准的数字
BTN[1]	无作用	使小时、分钟数递增； 将秒钟数清零，同时分钟数加一
BTN[2]	进入校准模式，数字开始闪烁	进入显示模式，数字停止闪烁

按下 BTN[0]，可见显示在分：秒、时：分、秒：毫秒之间循环切换。如图表 4.4-2，即为分：秒显示，图示 LED 显示为 0 分 32 秒。



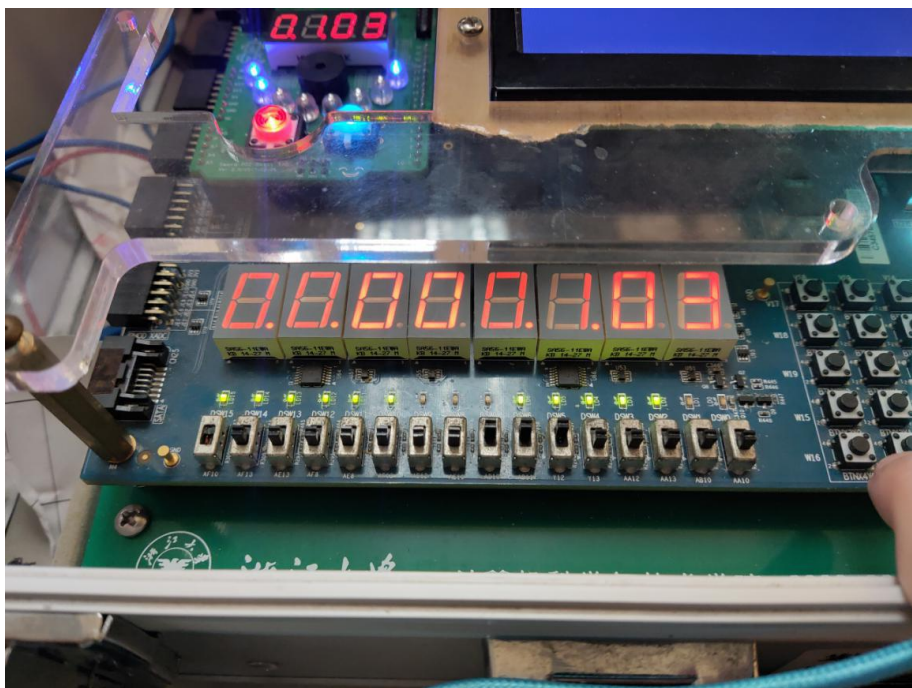
图表 4.4-2 分：秒显示

如图表 4.4-3，即为秒：毫秒显示，图示 LED 显示约为 2.588 秒。由于只采用低四位显示，此处的秒仅显示个位而毫秒显示三位。



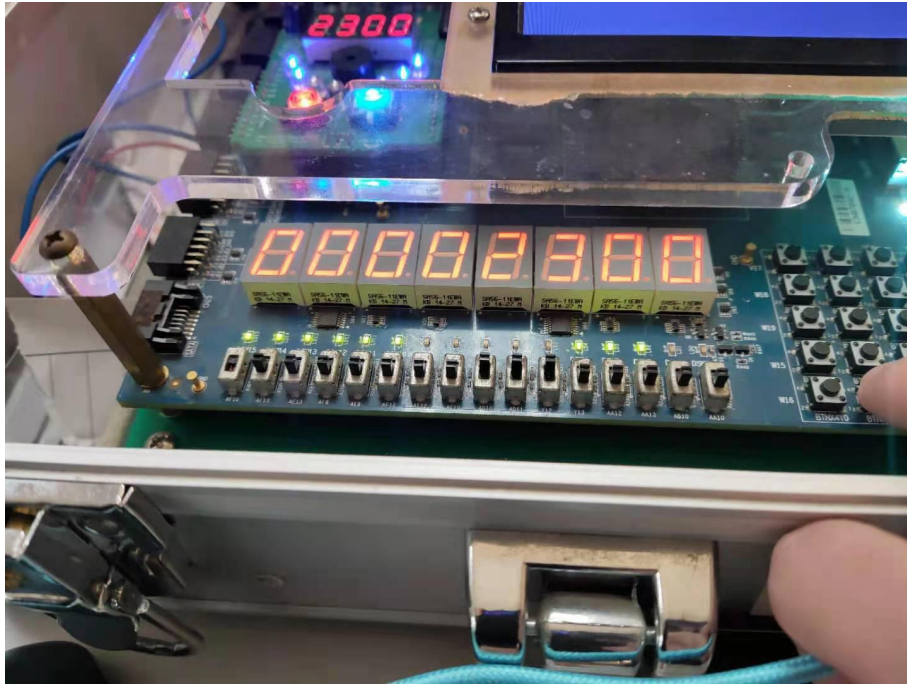
图表 4.4-3 秒：毫秒显示

在时：分显示下按下 BTN[2]，可见 LED 显示的数字开始闪烁，挂钟模块进入校准模式。按下 BTN[1]可切换校准的小时或分钟；按下 BTN[0]可使数字递增。如图表 4.4-4，通过校准可以很快地使挂钟显示 1 小时 3 分钟。



图表 4.4-4 校准时：分

在分：秒显示下按下 BTN[2]，可见 LED 显示的数字开始闪烁，挂钟模块进入校准模式。按下 BTN[1]可切换校准的分钟或秒钟；按下 BTN[0]可使分钟递增，或将秒钟清零并使得分钟加一。如图表 4.4-5，通过校准可以很快地使挂钟显示 23 分 00 秒。



图表 4.4-5 校准分：秒

五、讨论、心得

本次实验我的耗时较长，主要原因在于设计 count 计数器的时候，没有结合考虑到要输出显示，没有采用 BCD 码进行计数而是直接用二进制计数，导致了 LED 显示的问题。

同时，时钟模块的 clk1 和 clk2 信号的设置很巧妙，可以同时实现正常计数与校准递增。是值得花时间借鉴学习的方法。