浙江大学实验报告

专业：信息工程
姓名：卢嘉良
学号：3230104505
日期：2025 年 6 月 5 日
地点：东四-A-216

课程名称：数字系统设计实验　指导老师：屈民军 唐奕　仿真软件：ModelSim
研究内容：数字式秒表　实验序号：LAB 12　工程软件：Vivado

Part.1　Lab10: 学号滚动显示实验

一、 实验目的

1. 掌握译码器，显示译码器，数据选择器，计数器/分频器等功能模块的 Verilog HDL 语言描述。

2. 掌握数码管的动态显示驱动方式。

3. 进一步掌握参数定义和参数传递的方法，进一步理解电路层次结构设计。

4. 掌握具有大分频比的分频器模块的电路仿真。

二、 实验任务

设计滚动显示自己学号的电路，要求：

(1) 在 Basys3 开发板的 LED 数码管（4 位）显示学号（10 位），向左滚动，0.5s 滚动一位。

(2) 在一次学号显示完毕后，插入三个“空格”，即相邻两次学号之间显示的三个数码管不亮。

三、 实验原理

1. LED 数码管动态显示原理

Basys3 开发板的显示数码管的接法如图 1.1 所示。其采用的是动态驱动方式，所有数码管共用一组数据线 (a～g,dp)，数码管轮流被点亮，并且是共阳极的。由于 LED 数码管采用反相驱动，因此位选信号低电平有效。驱动的时序要求如图 1.2 所示，B0 B3 轮流输出低电平，依次点亮四位 LED 数码管。同时，要求在相应数码管点亮时输出该位数据的七段笔画码。为了使所有数码管稳定不闪烁地显示，即显示间隔不超过人眼的暂留时间，则每个数码管必须间隔 5-16ms 点亮一次，即刷新率应该为 60-200Hz。

2. 学号滚动实验原理

根据实验要求，可以画出学号滚动显示电路的原理框图如图 1.3 所示，主要由分频器，循环移位器和动态显示模块三部分组成。

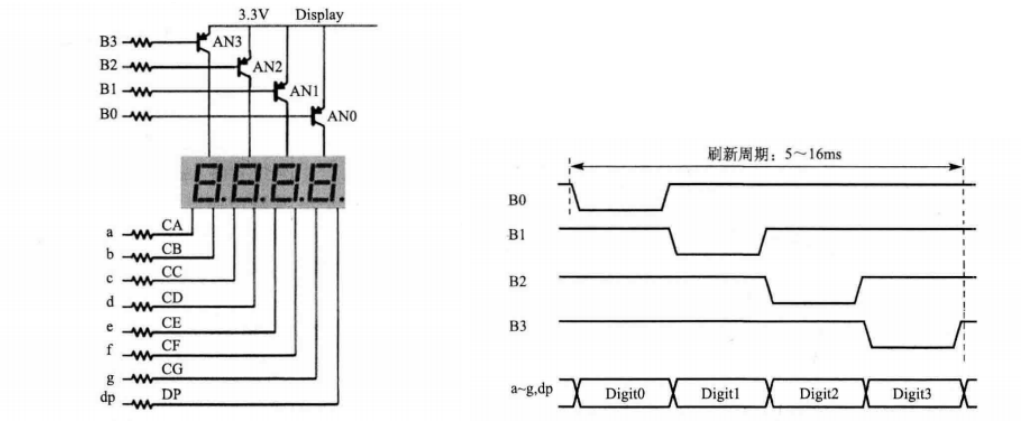

图 1.1: 动态显示数码管的接法 (L)

图 1.2: 动态显示驱动的时序图 (R)

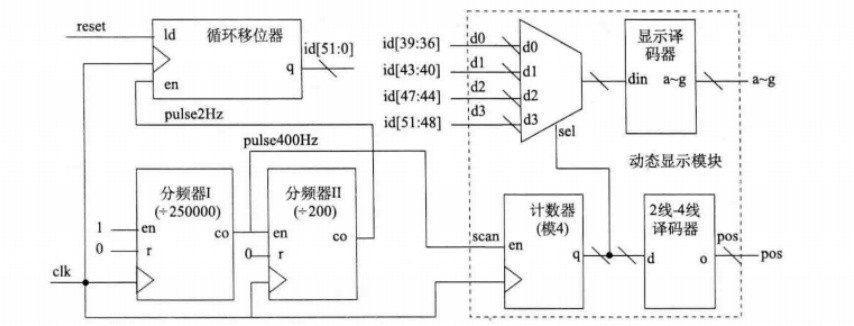

图 1.3: 学号滚动实验原理框图

(1)　分频器

由于系统时钟为 100MHz，因此，先由一个分频比为 250000 分频器 I 产生用于动态显示模块的 pulse400Hz 脉冲；再经分频器 II 对 pulse400Hz 进行 200 分频，产生 2Hz 脉冲信号 pulse2Hz，该脉冲信号控制滚动速度为 0.5s 滚动一次。在仿真时不需要设置工作时的分频比，只需通过传递参数使两个分频器的分频比设置为 4 和 16 即可。

(2)　循环移位器

循环移位器主要产生滚动显示数据，其功能表如下，假设学号为”9876543210”。

ld	en	clk	q*	Function
1	×	↑	52’haaa9876543210	Set Numbers Synchronously
0	1	↑	{q[47:0], q[51:48]}	Cyclic left shift by 4 bits
	0	-	q	Keep State

表 1.1: Cycle Shifter Function Table

(3)　动态显示模块

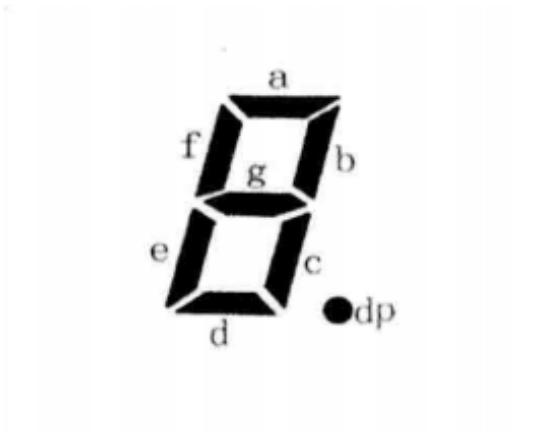

图 1.4: 七段显示译码器端口的对应关系

din[3:0]	{a,b,c,d,e,f,g}
0000	0000001
0001	1001111
0010	0010010
0011	0000110
0100	1001100
0101	0100100
0110	0100000
0111	0001111
1000	0000000
1001	0000100
Invalid	1111111

图 1.5: 7-Segment Decoder Truth Table

图 1.3 中虚框为动态显示模块，四进制计数器状态 q 控制数据选择器依次选出当前显示的 BCD 码送入显示译码器；另外，计数器状态 q 同时表征数据显示在哪个数码管上，通过 2 线-4 线译码器输出位选信号 pos 控制对应的数码管点亮，并且 2 线-4 线译码器输出低电平有效。位选信号 pos[3:0] 从高到低对应图 1.2 的 B3、B2、B1 和 B0 信号。七段译码器的功能表如图 1.5 所示，其中 a-g 的对应关系在图 1.4 中给出。

四、 Verilog HDL Code

说明：对于一些常用的组合电路模块和时序电路模块，此处不给出其 Verilog HDL 代码，而将他们都附在 Appendix A&B 当中供读者查阅。对于该部分给出的代码，他们所使用的 module 名即表明了它们的功能，在 Appendix A&B 中也给出了模块名和实际功能之间的对应关系。在 Part.2 和 Part.3 中也是如此。

1. 动态显示模块

动态显示模块的代码是完全按照图 1.3 中所给的结构来搭建的，在这里使用了一个四选一数据选择器，一个显示译码器，一个 2 线-4 线译码器和显示译码器，并按照图 1.3 所示的逻辑进行连接。此处不再做详细的说明。

Dynamic Display Module

```
module display(clk,scan,d0,d1,d2,d3,a,b,c,d,e,f,g,dp,pos);
//Declaration
input clk,scan;
input [3:0] d0;
input [3:0] d1;
input [3:0] d2;
input [3:0] d3;
output a,b,c,d,e,f,g,dp;
```

```
output [3:0] pos;
//Achievement
wire [3:0] din;
wire [1:0] sel;
reg [6:0] seq = 7'b1111111;
//Instantiate the mod-4 counter
counter_n #(.n(4),.counter_bits(2))
    counter_mod4(.clk(clk),.r(),.en(scan),.co(),.q(sel));
//Instantiate the mux-4to1
mux_4to1 #(.n(4)) mux(.out(din),.in0(d0),.in1(d1),.in2(d2),.in3(d3),.addr(sel));
//Instantiate the 2to4 decoder
decoder_24 decoder(.out(pos),.in(sel));
always @(*)
begin
  case(din) //Global assignment
    4'b0000: seq = 7'b0000001;
    4'b0001: seq = 7'b1001111;
    4'b0010: seq = 7'b0010010;
    4'b0011: seq = 7'b0000110;
    4'b0100: seq = 7'b1001100;
    4'b0101: seq = 7'b0100100;
    4'b0110: seq = 7'b0100000;
    4'b0111: seq = 7'b0001111;
    4'b1000: seq = 7'b0000000;
    4'b1001: seq = 7'b0000100;
    default: seq = 7'b1111111;
  endcase
end
//Show the correspondence
assign a = seq[6];
assign b = seq[5];
assign c = seq[4];
assign d = seq[3];
assign e = seq[2];
assign f = seq[1];
assign g = seq[0];
assign dp=1'b1;
endmodule
```

2. 循环移位器

循环移位器可以看作是一在上升沿动作的组合电路，搭建代码时实际上就是控制该模块在上升沿执行一个移位动作。此外为增加灵活性，模块中还应该包含 **n** 和 **length** 两个参数，分别用于控制移位量和需要移位的字符总长度。该部分代码如下：

Cycle Shifter Module

```
module cycle_shifter(clk,en,ld,q);
//Declaration
parameter n=4; //control the value of shifting
```

```
parameter length=52; //Total length
input clk,en,ld;
output [length-1:0] q;
//Initialize the beginning value
reg [length-1:0] q=52'haaa3230104505;

//Achievement
always @(posedge clk)
begin
  if (ld) q=52'haaa3230104505; //reset to the initial
  //shifting n
  else if (en) q={q[length-n-1:0],q[length-1:length-n]};
end
endmodule
```

编写该部分的测试代码时，主要考虑测试其移位功能，即接收到 **en** 时移位，否则保持；以及其复位功能。因此测试时考虑交替给出复位信号和移位使能信号，观察其工作情况。给出其测试代码:

Cycle Shifter Modul Testbench

```
`timescale 1ns / 1ps
module cycle_shifter_tb;
   parameter DELY = 10;
   parameter SHIFT_COUNT = 13; // 52/4=13次移位后回到初始值
   reg clk,en,ld;
   wire [51:0] q;
   // Instantiate the Unit Under Test (UUT)
   cycle_shifter uut (.clk(clk),.en(en),.ld(ld),.q(q));

   always #(DELY/2) clk = ~clk;
   initial begin
      clk = 0;
      en = 0;
      ld = 0;
      #(DELY*2);
      #(DELY) ld = 1;
      #(DELY*2) ld = 0;
      en = 1;
      repeat(4) #(DELY*2);
      en = 0;
      #(DELY*4);
      repeat(SHIFT_COUNT) #(DELY*2);
      en = 0;
      en = 1;
      ld = 1;
      #(DELY*2);
      ld = 0;
      #(DELY*2);
      $stop;
   end
   reg [51:0] last_q;
```

```
    initial last_q = q;
    always @(posedge clk) begin
        if (q !== last_q) begin
            last_q = q;
        end
    end

endmodule
```

3. 学号滚动显示顶层模块

编写顶层模块时也可考虑直接按照图 1.3 所示进行编写。在顶层模块中实例化循环移位器，分频器，动态显示模块三个子模块，然后按照图示进行连接即可。需要注意的是在顶层模块中需要声明参数 **sim** 来控制仿真时的分频比。

StudentID Top Module

```
module StudentID(clk,reset,a,b,c,d,e,f,g,dp,pos);
//Declaration
parameter sim = 0;
input clk,reset;
output a,b,c,d,e,f,g,dp;
output [3:0] pos;

//Achievement
wire co1,co2;
wire [51:0] id;
//No need to use the decimal point
assign dp = 1'b1;
//Instantiate the freq_devider
counter_n #(.n(sim ? 4 : 25_0000),.counter_bits(sim ? 2 : 20))
    freq_dev1(.clk(clk),.r(1'b0),.en(1'b1),.co(co1),.q());
counter_n #(.n(sim ? 16 : 200),.counter_bits(sim ? 4 : 8))
    freq_dev2(.clk(clk),.r(1'b0),.en(co1),.co(co2),.q());
//Instantiate the dycle_shifter
cycle_shifter cycsh_4(.clk(clk),.ld(reset),.en(co2),.q(id));
//Instantiate the display module
display
    disp_mod(.clk(clk),.scan(co1),.d0(id[39:36]),.d1(id[43:40]),.d2(id[47:44]),.d3(id[51:48]),
```

五、 仿真结果

1. 动态显示模块

动态显示模块的的仿真波形如图 1.6 所示。其中 **num**，**pos** 表示显示数字和位置，**numx** 表示第 x 位的显示数字，这些变量在 testbench 中通过显示值和控制端口 a-g 的输出值之间的关系来定义。因此在仿真没有出现明显问题时不需要分别查看 a-g 的值，只需要查看 **num** 即可判断显示值是否正确。

再结合 **pos** 和 **numx** 来判断显示的时序是否正确，以及通过 **scan** 和 **clk** 来判断刷新率是否正确。仿真时为方便查看，**num** 和 **num** 的显示格式均为 ASCII 码。

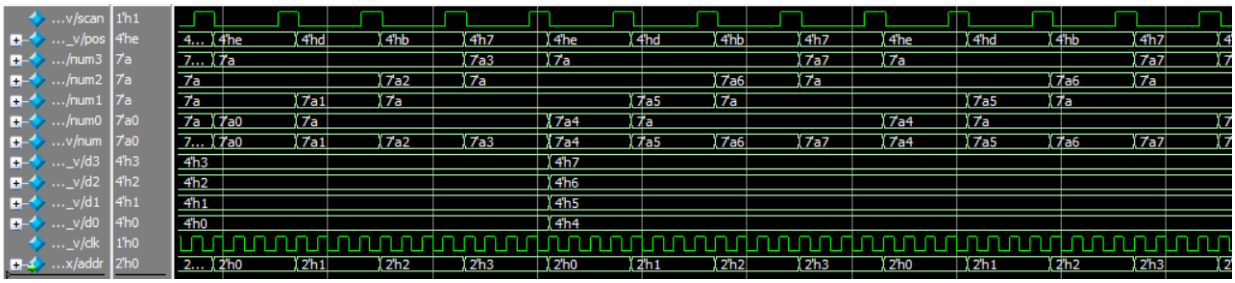

图 1.6: 动态显示模块的仿真波形

仿真结果记录在表 1.2 当中，表中以 i 代替无显示 (Invalid)。观察到数据选择器的地址信号 **addr** 和控制信号 **scan** 同频并且是以 4 个 **scan** 脉冲光为单位循环的，而 **numx** 的值的转换均发生在 **scan** 上升沿后的下一个时钟上升沿，因此显示频率是正确的。由于测试文件中输入值的改变也发生在时钟沿，因此只需要关注输出是否时刻与输入一致即可，而输入在 T_1,T_2,T_3 的取值下输出 **num**，即显示值都与输入 **{d0,d1,d2,d3}** ，故可认为显示值也正确。比较 **numx** 的值的转换历程，会发现其与图 1.2 中所规定的驱动时序是相同的，即四个数码管轮流亮起，再结合显示频率正确，因此显示时序也是正确的，因此可以认为仿真结果没有问题。

Signal	T_1	T_2	T_3
{d0,d1,d2,d3}	4'h0123	4'h4567	4'h89ab
addr	2'h0123	2'h0123	2'h0123
num	0123	4567	89aa
num3	iii3	iii7	iiii
num2	ii2i	ii6i	iiii
num1	i1ii	i5ii	i9ii
num0	0iii	4iii	8iii

表 1.2: 动态显示模块的仿真数据记录

2. 循环移位器的仿真波形

循环移位器的仿真波形如下图所示，由于完整波形较长，截取了其中较为重要的两部分分别放在图 1.7 和图 1.8 中。其中图 1.6 表明在移位使能信号 **en** 没有到来时移位不会发生，当 **en** 为 1 时循环移位器在时钟上升沿移位。图 1.7 则表明当复位信号 **ld** 到来时会将输出重新设置回初始值，复位信号的到来位置已经在图中标注出。故两个功能均正常，认为仿真结果没有问题。

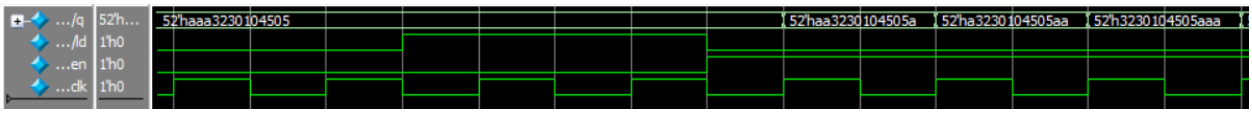

图 1.7: 动态显示模块的仿真波形

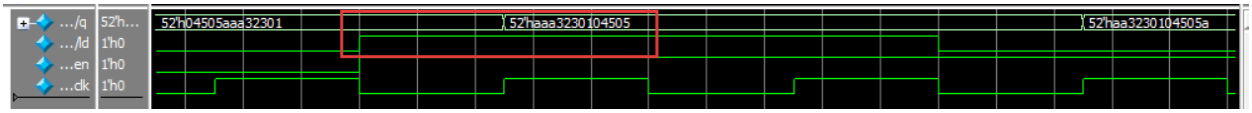

图 1.8: 动态显示模块的仿真波形

3. 学号滚动显示模块的仿真波形

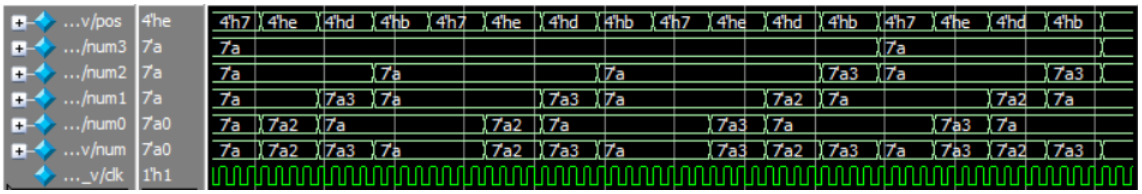

图 1.9: 学号滚动显示模块的仿真波形

如图 1.9 所示的是学号滚动显示模块的仿真波形的部分波形. 与动态显示模块相同，只需要监测 **num** 的值就可以得到此时的显示值，再检测 **numx** 的状态就可获知此时的显示时序是否正常。放大波形可以看出，每一位数码管的显示时间是 4 个 clk，显示内容的移位间隔是 16 个 clk，符合仿真时的设置值，故刷新率正常。此外，观察 **num** 的值可以看出显示值能够正常移位，并且 **numx** 的波形也与驱动时序相符，因此认为仿真结果是正常的。

六、 Vivado 工程

综合电路图如图 1.10 所示。

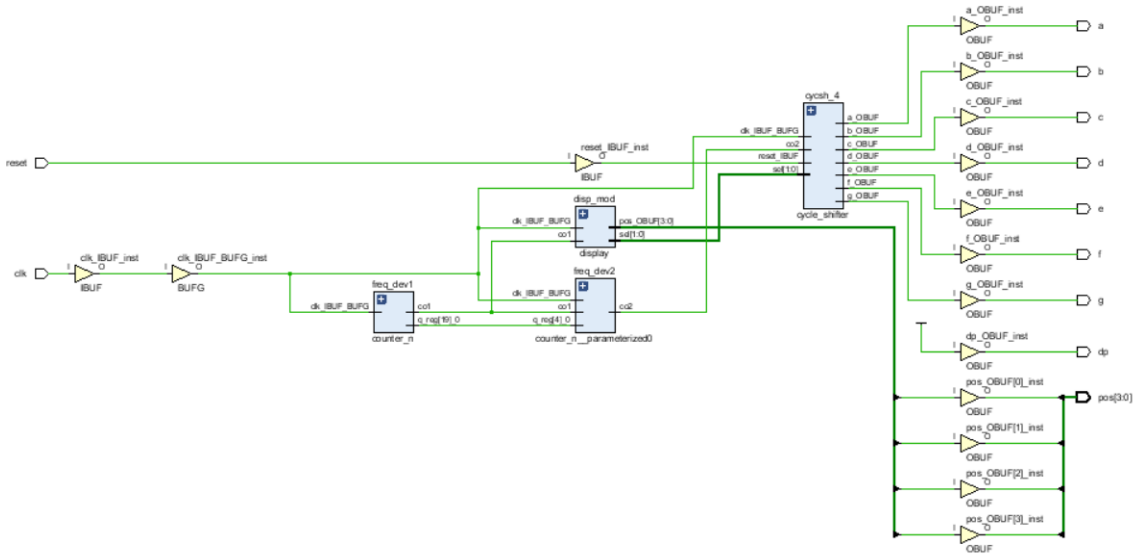

图 1.10: 学号滚动显示模块的综合电路图

Part.2　Lab11: 异步输入的同步器和开关防颤动电路的设计

一、　实验目的

1.　掌握减少亚稳态的方法，了解亚稳态给系统带来的危害。

2.　掌握开关颤动的概念和消除的方法。

3.　初步了解控制器的设计。

二、　实验任务

(1) 设计一个按键处理模块，要求每按一次按键，按键处理模块输出一个正脉冲信号，脉冲的宽度为一个 clk 时钟周期。

(2) 按图 2.1 所示搭建一个按键处理模块测试电路，下载到开发板验证，即每按键一次，led 指示灯更改亮灭状态。

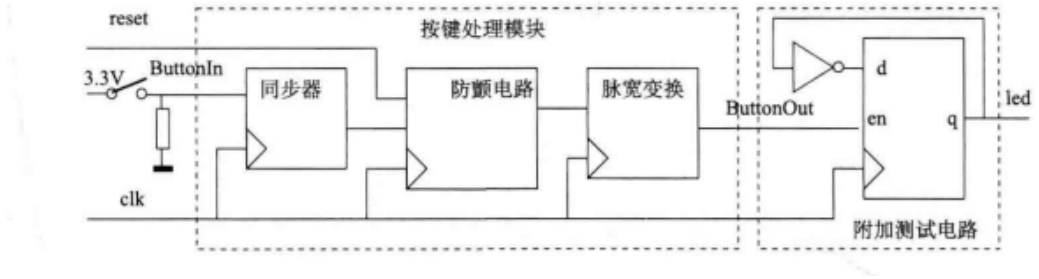

图 2.1: 按键处理模块测试电路

三、　实验原理

1.　概述

异步输入信号主要有两种，其一为开关（或按键）输入，其二为来自不同时钟域的由不同时钟同步的信号。异步输入信号对系统的影响有以下几点：

(1) 异步输入不是总能满足（它们所馈送的触发器）建立和保持时间的要求。因此，异步输入常常会把错误的数据锁存到触发器，或者使触发器进入亚稳定的状态。在该状态下触发器的输出不能识别为 1 或 0，如果没有正确地处理，亚稳态会导致严重的系统可靠性问题。

(2) 异步输入信号的宽度是不确定的，因此系统可能采样不到宽度小于一个时钟周期的异步输入，也可能对大于一个时钟周期的异步输入进行多次采样。

(3) 如果是开关 (或按键) 输入，当弹簧开关被按下或释放时，机械触点将立刻振动几毫秒，产生一个不稳定的开关信号。

针对异步输入信号的特点，可采取以下措施解决：

(1) 输入增加同步器，减少触发器进入亚稳定状态的概率；

(2) 采用脉冲宽度变换电路，将异步输入信号的宽度变换成一个时钟周期；

(3) 如果是开关 (或按键) 输入，在同步器与脉冲宽度变换电路之间插入一个开关防颤动电路。

2. 同步器的设计

同步功能的实现也是由 D 触发器的级联来实现的，如图 2.2 所示。第二级 D 触发器接收第一级 D 触发器的输出作为输入，而两级级 D 触发器的输出只会在时钟边沿进行变化，因此最终的输出一定是时钟同步的。此外，由于两个触发器都是同一个时钟上升沿触发的，而触发器本身具有传输延迟 t_{cd}，因此第二级触发器的同步输出相较第一级输出要延后一个 clk。

然而在异步设计中，完全避免亚稳态是不可能的。设计时首先尽可能减少出现亚稳态的可能性，其次尽可能减少亚稳态给系统带来危害的可能性。对于图 2.2 所示的连接方式，有理论研究表明出现亚稳态的概率将被大大降低，但相较于单触发器，这种方法同时带来了对输入信号的延时。

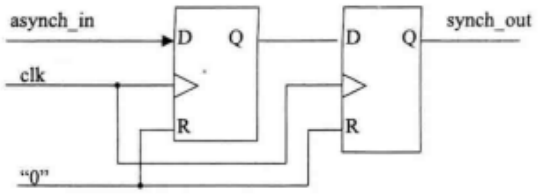

图 2.2: 同步器设计原理图

3. 开关防颤动电路的设计

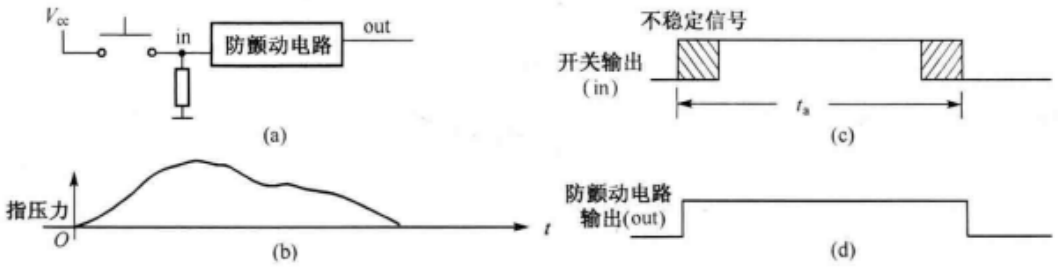

图 2.3: 按键开关的颤动

人类完成一次按键操作的指压力基本如图 2.3(b) 所示，按下与松开时都需要经过数毫秒的颤动，也因此产生几毫秒不稳定的输出，即电平在高低之间无规则摆动，如图 2.3(c) 所示。开关防颤动电路希望在按下一次按键时输出一个稳定的脉冲，即将 2.3(c) 所示的信号转换为 2.3(d) 所示的信号。

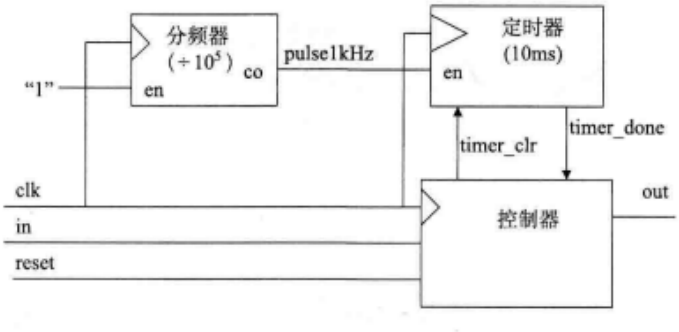

图 2.4: 防颤动电路的原理图

此处给出的解决办法是避免在颤动期采样，考虑到颤动期时间平均在 10ms 左右，电路需要实现在接收到按键按下后等待 10ms 再对输入进行采样，并在释放 10ms 后再对输入进行采样。据此设计的原理图如图 2.4 所示，电路由分频器、定时器和控制器三部分组成。分频器对系统时钟进行分频产生 1kHz 的 **pulse1kHz** 信号，该信号作为定时器的时钟信号。定时器是一个模值为 10 的计数器，当定时器启动时，在 **pulse1kHz** 的上升沿计数，10ms 后输出一个高电平脉冲 **timer_done**。该定时器用来定时颤动期，启动信号 **timer_clr** 由控制器提供，定时结束信号 **timer_done** 回馈给控制器。

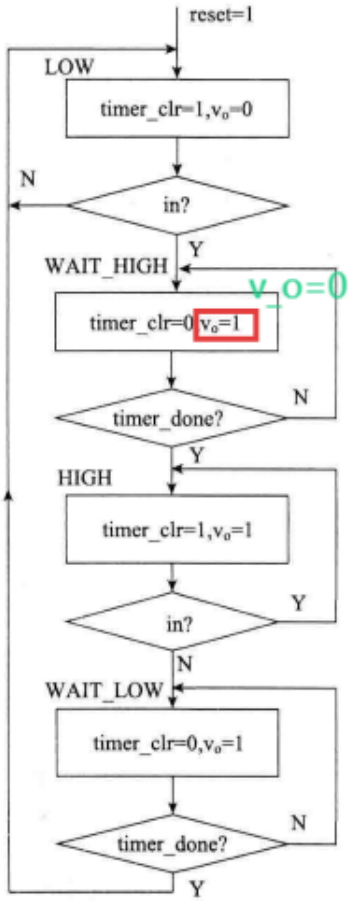

图 2.5: 控制器的 ASM 图

控制器是防颤动电路的核心，综合开关防颤动原理可画出控制器的 ASM 图如图 2.5 所示。一般情况下电路在 **LOW** 状态下等待；当按键按下（检测到 **in**=1），电路转到 **WAIT_HIGH** 状态，启动 10ms 定时器。在原 ASM 图中此处有一错误已修正，若 v_o 即为最终的模块输出，那么在 WAIT_HIGH 状态下仍应输出 0，如果 v_o 控制的是采样动作，那也不应该在此时执行采样，应该在结束后进行采样。当 10ms 定时结束（**timer_done=1**），电路进入 **HIGH** 状态，采样开关输入，等待按键释放；当按键释放（检测到 **in**=0），电路转到 **WAIT_LOW** 状态，启动 10ms 定时器，消除按键释放颤动期；定时结束后回到 **LOW** 状态下等待下次按键操作。

设计控制器时，考虑采用选择器型控制器的设计方法，在该方法中，数据选择器的地址输入即为两个 D 触发器的输出，输出信号直接从 D 触发器的输出经由组合电路引出。由于一共存在 4 种状态，考虑编号为 2'b00-2'b11，并以 2 个 D 触发器来存储。由于地址线为 2 根，故数据选择器应为 4 选 1 数据选择器。由 ASM 图和状态定义方式可以得到如图 2.1 所示的状态转换表，而四个数据选择器的输入要根据表 2.1 来确定。根据状态转换表，可以得到数据选择器的各地址输入如表 2.2 所示 (表中 Addr=Q_A, Q_B)。由表 2.1，容易得到输出方程如下所示：

$$timer\_clr = Q'_A Q'_B + Q_A Q'_B = Q'_B$$
$$v_o = (Q'_A Q'_B)' = Q_A Q_B$$

Present State	Q_AQ_B	Next State	$Q^*_AQ^*_B$	Condition	timer_clr	v_o
LOW	00	WAIT_HIGH	01	**in**	0	0
	00	LOW	00	**in**′	1	0
WAIT_HIGH	01	HIGH	10	**timer_done**	1	1
	01	WAIT_HIGH	01	**timer_done**′	0	0
HIGH	10	WAIT_LOW	11	**in**′	0	1
	10	HIGH	10	**in**	1	1
WAIT_LOW	11	WAIT_LOW	11	**timer_done**′	0	1
	11	LOW	00	**timer_done**	1	0

表 2.1: 控制器的状态转移表

Addr	2'b00	2'b01	2'b10	2'b11
Mux_A	0	**timer_done**	1	**timer_done**′
Mux_B	**in**	**timer_done**′	**in**′	**timer_done**′

表 2.2: 四选一数据选择器的各地址输入

4. 脉宽变换电路的设计

由于开关或按键的输入信号宽度远远大于一个时钟周期，脉宽变换电路的作用是将输入信号宽度变换成一个时钟周期。由于同步器的存在，输入已是同步的，且宽度大于一个时钟的脉冲，因此脉宽变换电路的设计只需如图 2.4 所示即可。一个宽度大于一个时钟的同步脉冲到来时首先会使得输出为 1，经过一个 clk 后 D 触发器读入该脉冲的高电平，输出变回 0，从而实现了一个 clk 宽度的脉冲的输出。

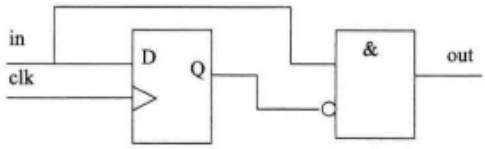

图 2.6: 脉宽变换电路设计原理图

四、 Verilog HDL Code

1. 同步器

同步器部分的结构较为简单，代码编写也较为简单，是直接利用 D 触发器模块根据图 2.2 进行连接来实现的，该模块代码如下：

Synchroniser Module

```
module synchroniser(asynch_in,synch_out,clk);
//Declaration of the ports
input asynch_in,clk;
output synch_out;
//Achievement
wire q;
dffr dff1(.d(asynch_in),.r(1'b0),.clk(clk),.q(q));
dffr dff2(.d(q),.r(1'b0),.clk(clk),.q(synch_out));

endmodule
```

2. 开关防颤动电路

(1) 开关防颤动电路的控制器

开关防颤动电路的控制器是根据表 2.2 和选择器型控制器的基本结构来设计的。所有多路选择器输出的组合就是控制器次态的编码。由于 D 型触发器在其模块定义中已经对于初始值进行了初始化，在该模块内不再进行初始化操作。

Debouncer Controller Module

```
module controller(in,timer_done,clk,timer_clr,out,reset,q1,q2);
//Declaration of the ports
input in,clk,timer_done,reset;
output timer_clr,out;
//Achievement
output q1,q2;
wire d1,d2;
//Mux-type controller
mux_4to1
    mux1(.out(d1),.in0(1'b0),.in1(timer_done),.in2(1'b1),.in3(!timer_done),.addr({q1,q2}));
mux_4to1
    mux2(.out(d2),.in0(in),.in1(!timer_done),.in2(!in),.in3(!timer_done),.addr({q1,q2}));
```

```
//The output of the dff is the position line of mux
dffr dff1(.d(d1),.r(reset),.clk(clk),.q(q1));
dffr dff2(.d(d2),.r(reset),.clk(clk),.q(q2));
assign timer_clr=!q2;
assign out=q1 || q2;

endmodule
```

(2) 开关防颤动电路的顶层模块

同样，顶层模块的代码是按照图 2.4 来进行编写的，需要注意的是，模块的输出 **out** 就是控制器的输出 v_o。**reset** 的复位功能只对控制器的状态进行复位，定时器是受到控制器的控制工作的，不需要进行复位。

Debouncer Top Module

```
module debouncer(in,out,reset,clk);
//Declaration of the ports
parameter sim=0;
input in,clk,reset;
output out;
//Achievement
wire pulse_1kHz;
wire q1,q2;
wire timer_done,timer_clr;
//Module Instantiation
counter_n #(.n(sim?32:10_0000),.counter_bits(sim?5:18))
    counter(.clk(clk),.r(reset),.en(1'b1),.co(pulse_1kHz),.q());
timer timer10ms(.clk(clk),.r(timer_clr),.en(pulse_1kHz),.done(timer_done));
controller ctrl(.in(in),.timer_done(timer_done)
    ,.clk(clk),.timer_clr(timer_clr),.out(out),.reset(reset),.q1(q1),.q2(q2));
endmodule
```

3. 脉宽变换电路

脉宽变换电路的代码编写也较为简单，直接根据图 2.6 进行设计即可，其中的与门直接以逻辑运算代替。

Pulse Width Converter Module

```
module pulse_width_conv(in,out,clk);
//Declaration of the ports
input in,clk;
output out;
//Achievement
wire q; //An auxilliary port
dffr dff1(.d(in),.r(1'b0),.clk(clk),.q(q));
assign out= (!q) && in;
endmodule
```

4. 按键处理模块的顶层代码

由于编写上述各个子模块时对于按键处理模块做的是分划，没有其它需要在顶层模块中声明的部件，因此编写顶层代码时只需注意各端口之间的连接方式，按照图 2.1 进行编写即可。

Button Process Unit Module

```
module button_process_unit(reset,ButtonIn,clk,ButtonOut);

//Declaration of the ports
parameter sim=0;
input ButtonIn,clk,reset;
output ButtonOut;

//Achievement
wire synch_out,tmp_sig;

synchroniser synch(.asynch_in(ButtonIn),.synch_out(synch_out),.clk(clk));
debouncer #(.sim(sim))debonc(.in(synch_out),.out(tmp_sig),.reset(reset),.clk(clk));
pulse_width_conv converter(.in(tmp_sig),.out(ButtonOut),.clk(clk));

endmodule
```

5. 测试电路的顶层代码

为了能在开发板上验证上述模块的设计是否正确，还需要补充一个如图 2.1 所示的附加测试电路。由于附加测试电路部分的结构非常简单，因此考虑在整个测试电路的顶层模块中分别声明附加测试电路中的细节，而不单独定义一个附加测试电路模块；而测试电路顶层模块中必须直接调用按键处理电路的顶层模块而不能拆分调用同步器模块等，这样才能测试按键处理电路模块是否设计正确。编写代码时根据图 2.1 直接进行编写即可。

Button Process Testing Module

```
module test(ButtonIn,clk,led,reset);

//Declaration of the ports
input ButtonIn,clk,reset;
output led;

//Achievement
wire tmp;

button_process_unit(.reset(reset),.ButtonIn(ButtonIn),.clk(clk),.ButtonOut(tmp));
dffre dff1(.d(!led),.en(tmp),.r(1'b0),.clk(clk),.q(led));

endmodule
```

五、 仿真结果

仿真时针对的是按键处理模块进行仿真，并不需要加入附加测试电路的相关部分。测试时显示按键处理模块的输入信号。同步器的输出信号，开关防颤动模块的输出信号和最终脉宽变换电路的输出信号，便于观察每一模块是否实现其功能。

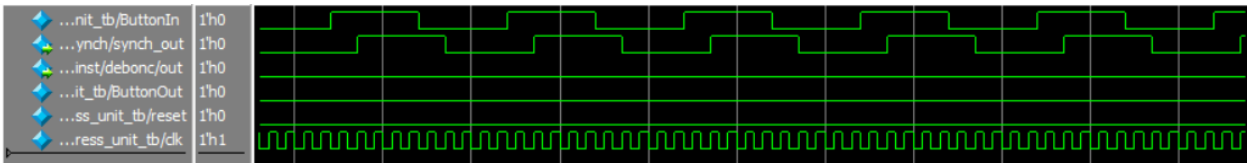

图 2.7: 开关按下颤动期开始时的仿真波形

如图 2.7 所示是仿真开关按下进入颤动期时的仿真结果。可以观察到同步器顺利地完成了同步功能，将颤动脉冲全部同步到了时钟上；并且此时开关防颤动电路不应该有输出，因为此时还在等待时间，并没有进行采样，同理脉宽变换电路也不应当有输出，符合预测。

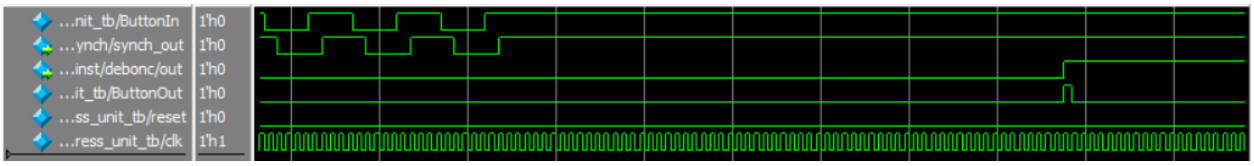

图 2.8: 开关按下颤动期结束时的仿真波形

如图 2.8 所示是仿真开关按下颤动期结束时的仿真波形，可以看到当颤动期结束之后，开关防颤动电路才开始采样并输出。如果对于 clk 进行计数，会发现在图 2.7 第一个同步器上升沿到来 320 个 clk 后开关防颤动电路采样，符合设计时的延迟时间。脉宽变换电路与开关防颤动电路的变化同步，输出一个 clk 宽度的脉冲，仿真结果合理。

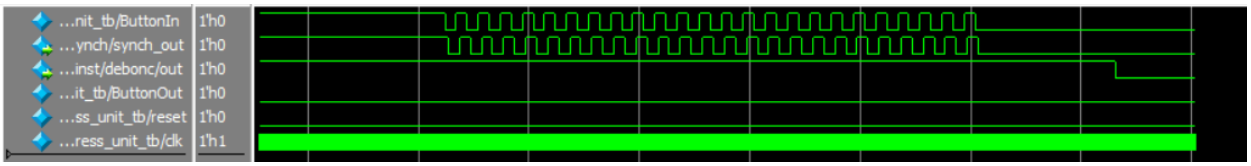

图 2.9: 开关松开颤动期的仿真波形

如图 2.9 所示是仿真开关松开颤动期的仿真波形。可以看到开关防颤动电路的输出才颤动期结束之后才回到 0. 在仿真全波形当中，因为只按下了一次按钮，最终模块输出只输出一次 clk 宽度的脉冲，综上可以认为仿真结果是正确的，电路功能没有问题。

六、 Vivado 工程

综合电路图如图 2.10 所示。

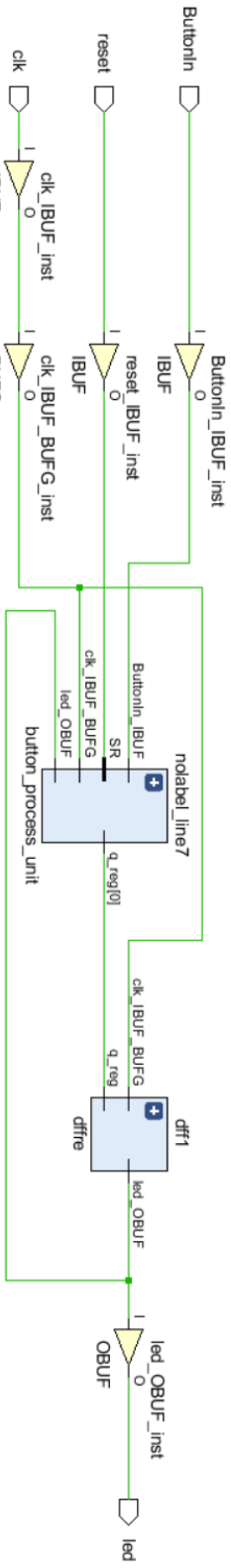

图 2.10: 按键处理模块测试电路的综合电路图

Part.3　Lab12: 数字式秒表

一、　实验目的

1.　掌握计数器的功能和应用。

2.　理解开关防颤动的必要性。

3.　掌握简单控制器的设计方法。

二、　实验任务

设计一个数字秒表电路。设计要求：

(1) 计时范围 0°0’～9°59’9”，分辨率为 0.1s，用数码管显示计时值；

(2) 秒表设有一个功能按键开关 ButtonIn。

设计的秒表具有” 双计时” 功能，即要记录甲、乙两物体同时出发，但不同时间到达终点的时间。当电路处于” 初始” 状态时，第一次按键，开始自动计时；待甲到达终点时再按一下，此时将甲物体的计时值保存下来，但计时并未停止，内部电路仍在继续为乙物体累积计时；当乙物体到达终点时，第三次按键，停止计时，此时可记录并显示乙物体的计时值；第四次按键将保存下来的甲物体计时值显示出来；第五次按键，计数器自动复位为 0’0.0”，即秒表回到” 初始” 状态。

三、　实验原理

根据设计的基本要求，可画出秒表电路的原理框图，如图 3.1 所示，秒表电路由分频模块、按键处理模块、控制器、计时模块、显示选择模块和动态显示模块组成。分频器模块产生其他模块所需的脉冲信号，其中输出的 pulse400Hz 信号是频率为 400Hz、宽度为一个 clk 周期的脉冲信号，该信号用于动态显示扫描模块。而 pulse10Hz 信号是频率为 10Hz、宽度为一个 clk 周期的脉冲信号，该信号为秒表的计时基准信号。按键处理模块完成按键输入的同步器、开关防颤动和脉冲宽度变换等功能，即当按键一次，输出一个宽度为 clk 周期的脉冲信号 ButtonOut，该模块的设计方法参见 Lab11。计时模块可由 1 个十进制计数器（十分之一秒计时）、1 个六十进制 BCD 码秒计数器和 1 个十进制计数器级联而成。动态显示模块接收计数器的输出值，并由控制器的 save 信号控制存入甲的成绩到 D 触发器当中，显示值则由控制器的 disp 信号控制，经由数据选择器控制显示甲的成绩还是乙的成绩。显示采用数码管动态显示技术，设计原理参见 Lab10。与 Lab10 所不同，本实验考虑在第一、第三位显示小数点，用于区分分，秒和十分秒，及对应三个计数器的输出。

其中控制器部分的 ASM 图如图 3.2 所示。在按键控制下，输出 clr，count，save 和 disp 四个信号分别控制计时模块的工作状态。为便于区分，从上至下的状态名分别为 RESET, TIMING1, TIMING2, STOP1, STOP2, 分别代表秒表的初始, 计时甲，计时乙和停止下显示乙，显示甲五个状态，其中 RESET，TIMING 和 STOP 三类状态相应地对应着控制计数器模块清零、计数和保持三种功能，而清零和计数分别由 clr 和 count 信号控制。在清零状态下向计数器输出复位信号置 0，第一次按下按钮开始计时，count 信号置 1；第二次按下按钮，记录甲的成绩，save 信号输出 1，将此时的计数器输出存入 D 触发器当中；第三次按下按钮，count 置 0，结束计数，disp 保持在 0，显示乙的成绩；第四次按下按钮，

disp 置 1，改为显示 D 触发器中存储的值，即甲的成绩；第五次按下按钮，disp 置回 0 且复位信号 clr 置 0，显示计时初始值。

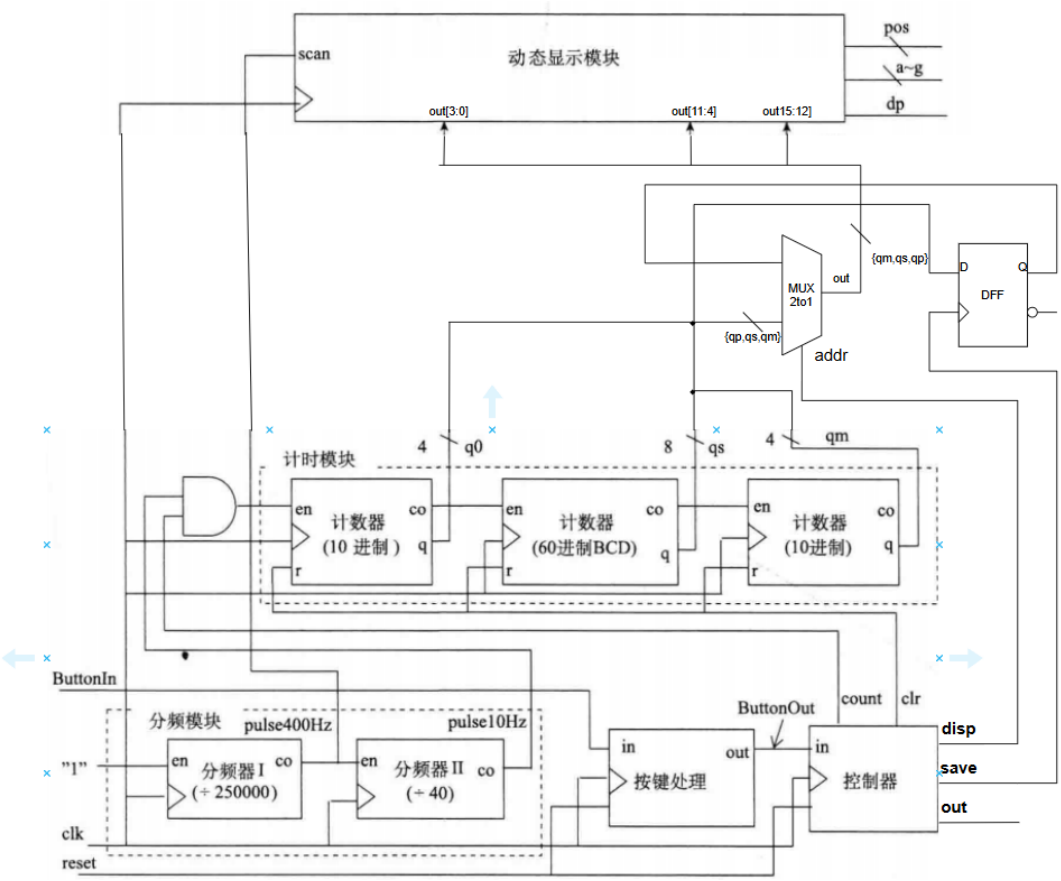

图 3.1: 秒表电路原理框图

四、 Verilog HDL Code

在图 3.1 的原理框图中，子模块只有控制器部分的代码没有给出，其余子模块均可在 Appendex A 中查得或在 Lab10，Lab11 中进行了说明，因此只需要重新编写控制器的子模块代码和数字秒表的顶层模块代码即可。但是为保持结构的简洁性，对于分频模块和计时模块我们也进行了封装处理。

1. 分频模块

如图 3.1 所示，分频模块由两个分频器级联而成，输出两个不同频率的脉冲信号。此外，编写代码时通过传递参数 sim 来控制仿真时的分频比。

Devider Module

```
module Devider(clk,pulse400Hz,pulse10Hz);
//Declaration
parameter sim=1'b0;
input clk;
```

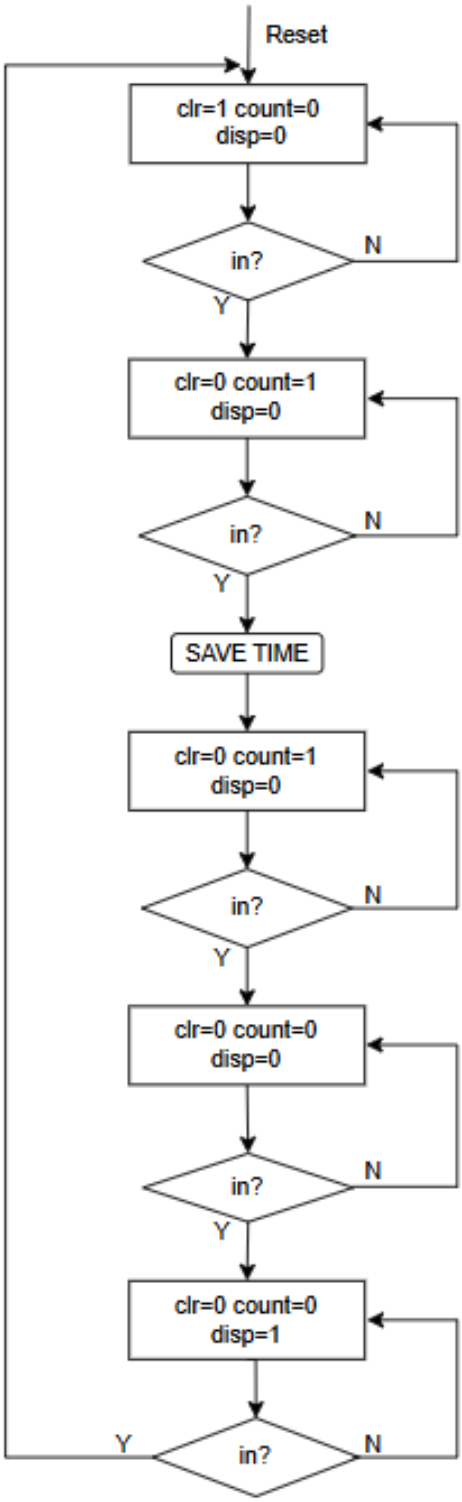

图 3.2: 秒表电路控制器 ASM 图

```
output pulse400Hz,pulse10Hz;
//Achievement
counter_n #(.n(sim ? 2:250000),.counter_bits(sim ?
    2:18))counter_1(.clk(clk),.r(1'b0),.en(1'b1),.co(pulse400Hz),.q());
counter_n #(.n(sim ? 10:40),.counter_bits(sim ?
    4:6))counter_2(.clk(clk),.r(1'b0),.en(pulse400Hz),.co(pulse10Hz),.q());

endmodule
```

2. 计时模块

如图 3.1 所示，计时模块由三个计数器级联而成，输出 3 个随计数使能递增的数字。连接时主要需要注意第一级输出是最低位，中间一级的输出是中间位并且中间一级的高位是输出显示的高位 (即 qm[7:4] 是 qm[3:0] 的高位)，最后一级的输出是最高位。为了使仿真时的波形更容易阅读，此处四位输出被分别拆分成了 4 级计数器来实现。

Counter Module

```
module Counter(clk,r,en,q0,qs,qm);
//Declaration
input clk,r,en;
output [3:0] q0;
output [7:0] qs;
output [3:0] qm;

//Achievement
wire co1,co2,co3;
counter_n #(.n(10),.counter_bits(4))
    counter_add1(.clk(clk),.r(r),.en(en),.co(co1),.q(q0));
counter_n #(.n(10),.counter_bits(4))
    counter_add2(.clk(clk),.r(r),.en(co1),.co(co2),.q(qs[3:0]));
counter_n #(.n(6),.counter_bits(4))
    counter_add3(.clk(clk),.r(r),.en(co2),.co(co3),.q(qs[7:4]));
counter_n #(.n(10),.counter_bits(4))
    counter_add4(.clk(clk),.r(r),.en(co3),.co(),.q(qm));

endmodule
```

3. 控制模块

此处完全采用 Verilog HDL 中状态机的二段式设计方法，根据图 3.2 所示的 ASM 图进行设计。可得到设计完毕的代码如下：

Counter Module

```
module count_controller(clk,reset,in,clr,count,save,disp);
//Declaration of in&out
input clk,reset,in;
output clr,count,save,disp;
```

```
//Define the states
parameter RESET=3'b000;
parameter TIMING_1=3'b001;
parameter TIMING_2=3'b010;
parameter STOP_1=3'b011;
parameter STOP_2=3'b100;

//machine variabe
reg [2:0] cur_state;
reg [2:0] next_state;

//Part1 State Transfer
always @(posedge clk or posedge reset)
begin
 if (reset)
  cur_state=RESET;
 else
  cur_state=next_state;
end

//Part2 state switch
always@(*)
begin
 case(cur_state)
  RESET:
   case(in)
    1'b0:next_state=RESET;
    1'b1:next_state=TIMING_1;
   endcase
  TIMING_1:
   case(in)
    1'b0:next_state=TIMING_1;
    1'b1:next_state=TIMING_2;
   endcase
  TIMING_2:
   case(in)
    1'b0:next_state=TIMING_2;
    1'b1:next_state=STOP_1;
   endcase
  STOP_1:
   case(in)
    1'b0:next_state=STOP_1;
    1'b1:next_state=STOP_2;
   endcase
  STOP_2:
   case(in)
    1'b0:next_state=STOP_2;
    1'b1:next_state=RESET;
   endcase
 endcase
```

```
end

//Assign the output
assign clr = !cur_state[2] && !cur_state[1] && !cur_state[0];
assign count = cur_state[1]^cur_state[0];
assign save = !cur_state[1]&&cur_state[0]&&in;
assign disp = cur_state[2];

endmodule
```

4. 顶层模块

顶层模块的代码编写是按照图 3.1 进行的。编写时考虑到便于仿真，图 3.1 中的总线型数据选择器和 D 触发器均采用分布形式，即各位均采用一个 D 触发器和数据选择器输入到最终的动态显示模块，故声明了 4 个数据选择器和 D 触发器。同时，为了避开 save 信号可能的竞争-冒险现象，没有选择将 save 信号直接连接到 D 触发器的时钟端，而是考虑采用一个带同步使能的 D 触发器，将 save 连接至使能端 en。由于竞争-冒险下的 save 脉冲很短，在使能端下难以使 D 触发器动作；而正确的 save 信号和 in 信号同宽，具有一个时钟宽度，因此可以保证使其动作。

Stopwatch Module

```
module stopwatch(ButtonIn,clk,reset,pos,dp,a,b,c,d,e,f,g);
//Declaration of the ports
parameter sim=0;
input ButtonIn,clk,reset;
output dp,a,b,c,d,e,f,g;
output [3:0] pos;

//Achievement
wire pulse400Hz,pulse10Hz;
Devider
    #(.sim(sim))devider_inst(.clk(clk),.pulse400Hz(pulse400Hz),.pulse10Hz(pulse10Hz));

wire Button_pulse;
button_process_unit
    #(.sim(sim))butt_proc(.reset(reset),.ButtonIn(ButtonIn),.clk(clk),.ButtonOut(Button_pulse));

wire clr,count,save,disp;
count_controller controller(.clk(clk),.reset(reset)
,.in(Button_pulse),.clr(clr),.count(count),.save(save),.disp(disp));

wire en;
assign en = pulse10Hz&&count;
wire [3:0] d0,d1,d2,d3; //to save the output of the d-flip-flop
wire [3:0] q0;//to save the output of the counter
wire [7:0] qs;
wire [3:0] qm;
wire [3:0] out0,out1,out2,out3;//the output of the mux
```

```
Counter counter(.clk(clk),.r(clr),.en(en),.q0(q0),.qs(qs),.qm(qm));
//Saving Module
mux_2to1 #(.n(4))disp0(.out(out0),.in0(q0),.in1(d0),.addr(disp));
mux_2to1 #(.n(4))disp1(.out(out1),.in0(qs[3:0]),.in1(d1),.addr(disp));
mux_2to1 #(.n(4))disp2(.out(out2),.in0(qs[7:4]),.in1(d2),.addr(disp));
mux_2to1 #(.n(4))disp3(.out(out3),.in0(qm),.in1(d3),.addr(disp));

dffre #(.n(4))dff_out0(.d(q0),.en(save),.r(reset),.clk(clk),.q(d0));
dffre #(.n(4))dff_out1(.d(qs[3:0]),.en(save),.r(reset),.clk(clk),.q(d1));
dffre #(.n(4))dff_out2(.d(qs[7:4]),.en(save),.r(reset),.clk(clk),.q(d2));
dffre #(.n(4))dff_out3(.d(qm),.en(save),.r(reset),.clk(clk),.q(d3));

display display_inst(.clk(clk),.scan(pulse400Hz),.d0(out0),.d1(out1)
,.d2(out2),.d3(out3),.a(a),.b(b),.c(c),.d(d),.e(e),.f(f),.g(g),.dp(dp),.pos(pos));

endmodule
```

五、 仿真结果

1. 计时模块

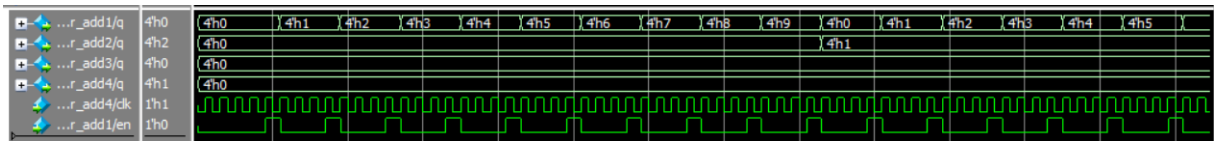

图 3.3: 计时模块仿真波形（第一级）

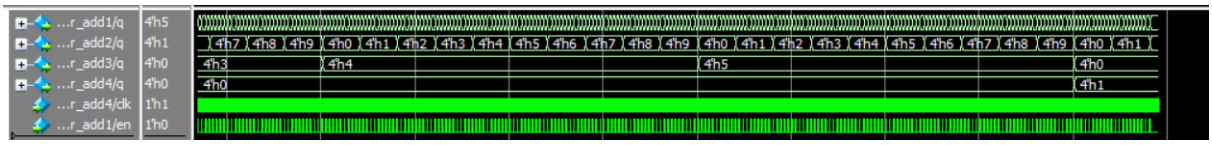

图 3.4: 计时模块仿真波形（第二级和第三级）

图 3.3 和 3.4 所示是计时模块的仿真波形的一部分。由于此处检测的是各级计数器的输出之间的关系，可以看到在仿真波形中第一、二级计数器的进制均是 10，第三级计数器进制是 6，第四级的进制没有仿真，实际上也超出了要求范围。故可以认为仿真结果正确。

2. 控制模块

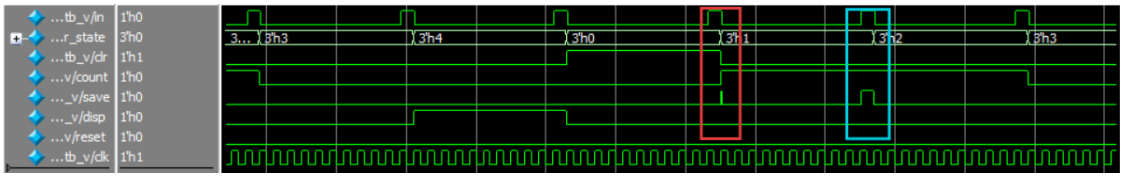

图 3.5: 控制模块仿真波形

在控制模块的仿真中显示了随着按键的输入，控制器的状态转换情况以及各控制信号的输出情况。首先状态编号的转换随着 in 的输入是正确的，除了 save 状态之外，其它控制信号的输出均没有问题，可将仿真波形中各状态下的输出情况列出如表 3.1 所示。以下标 s 标识仿真值，下表 e 标识设计时的理论值。需要说明的是，由于原本的 testbench 测试文件是给非个性化要求制作的，为了使其能够对于个性化要求适配，简单修改了其中的内容，即增加了按键动作执行的次数。由于这一修改较为简单，只是修改了循环执行的次数，因此此处不给出其代码。

State	clr_m/clr_e	$count_m/count_e$	$disp_m/disp_e$
RESET(3'b000)	1/1	0/0	0/0
TIMING1(3'b001)	0/0	1/1	0/0
TIMING1(3'b010)	0/0	1/1	0/0
STOP1(3'b011)	0/0	0/0	0/0
STOP2(3'b100)	0/0	0/0	1/1

表 3.1: 控制模块仿真结果

对于 save 信号来说，他需要在第二次按下按钮时记录时间，也就是应该在 TIMING1 状态下接收到 in 状态后输出。观察图 3.3 中蓝色框出部分，这没有问题；但是图中红色框出部分表明由于仿真时 in 输入信号与时钟周期不完全同步（如图 3.4）所示，导致在 TIMING1 状态开始时会在 save 输出一个脉冲。考虑到在顶层模块当中这个 in 信号是经过同步器处理的同步信号，在这种情形下 save 的脉冲会因为竞争-冒险而产生。如果要对其进行修改，应当考虑用格雷码对状态进行编号。总的来说，仿真没有明显问题。

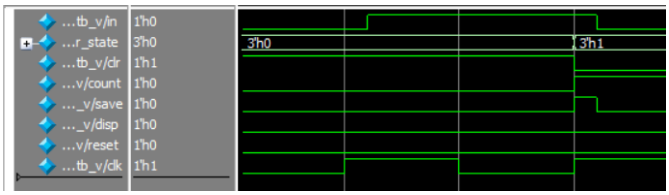

图 3.6: 控制模块仿真波形

3. 顶层模块

顶层模块的仿真结果如图 3.5-3.9 所示，分别截取的是连续地按下 5 次按钮时仿真的波形。同理于 Lab10，通过 numx 来判断数码管显示是否正确。注意到显示值并没有严格按照 pulse10Hz 来变化，这是因为仿真时分频器 I 输出的扫描信号频率是分频器 II 的计时信号频率的 10 倍，意味着每计时加 1

时能输出的位数实际上是 10 位，并不是 4 位数码管显示位数 4 的倍数，因此不会完全同步。在实际使用时，由于倍数变为 40 倍，是 4 的倍数，因此各显示值的显示时长都是一样的，不会出现不一样的情形。

再考虑按下按钮时的动作是否正确。检查图 3.5-3.9，第一次按下前显示值是 0°0’0” 按下按钮后开始计时；第二次按下后的时间是 0°52’5”，继续计时；第三次按下后停在按下按钮前显示的时间，为 1°20’0”. 注意，这里就体现出了显示扫描信号和计时信号的倍数不是显示位数的倍数的坏处。由于每一轮显示都是从最低位开始的。当时间实际上处于 1°19’9” 时，先显示了最低位的 9，再显示了次低位的 9，然而就在这时计数器时间跳变为 1°20’0”，此时显示信号不会回到最低位，而是继续显示次高位的 2 和最高位的 0，好像在显示 1°29’9”，实际上此时记录的计数器输出是 1°20’0”，后面也能够停止并继续保持；要避免此问题可以考虑修改仿真时的分频比，不采用教材上的数值。同样，由于实际使用时显示扫描信号和计时使能信号的频率是满足要求的，不会出现上述问题。当第四次按下按钮时，回到显示甲的成绩，即 0°52’5”，显示正确。第五次按下按钮，清零，回到复位状态，符合波形。故仿真结果正确。

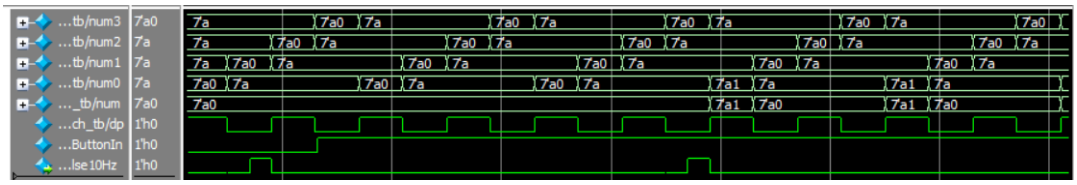

图 3.7: 顶层模块仿真波形（第一次按下按钮）

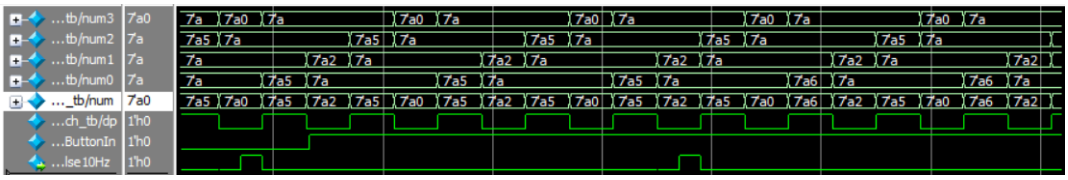

图 3.8: 顶层模块仿真波形（第二次按下按钮）

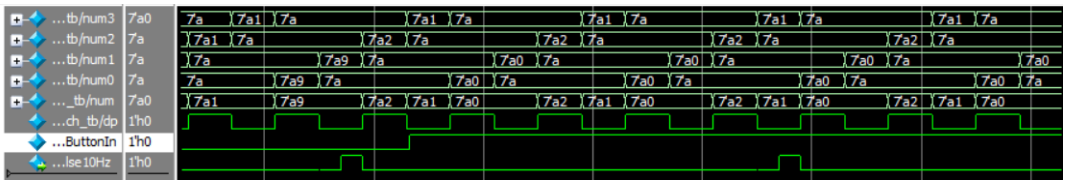

图 3.9: 顶层模块仿真波形（第三次按下按钮）

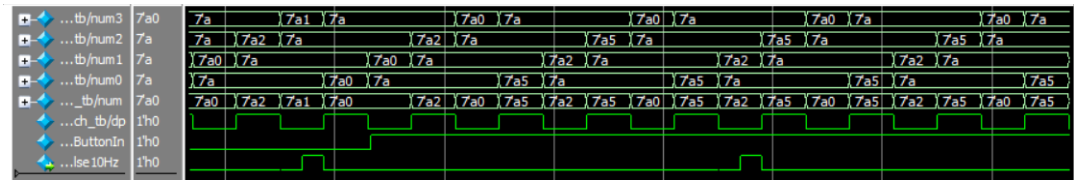

图 3.10: 顶层模块仿真波形（第四次按下按钮）

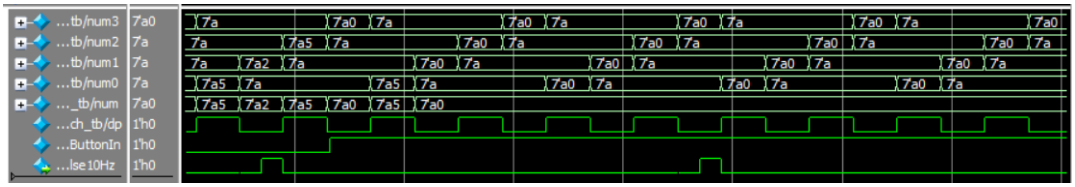

图 3.11: 顶层模块仿真波形（第五次按下按钮）

六、 Vivado 工程

综合电路图如图 3.10 所示。

七、 思考题

1. 为什么分频器中的计数状态可以采用二进制编码，而计时模块中的计数状态必须采用 BCD 编码？

参考 Appendix B 中 **counter_n** 模块的声明就可以看出，计数器的输出形式是由编码方式决定的。二进制的正常在 q 端输出 2 进制码，BCD 编码下虽然输出的 q 也是只包含 0 和 1 的，但这并不是 2 进制码，而是以 4 位为单位作为一个十进制数的 BCD 码。对于动态显示模块而言，它显示的是 10 进制数，为此它接受的信号表示 4 个十进制数字，这是通过对其输入信号进行每四位的裁剪得到的，换言之计时模块输出给动态显示模块的信号必须是 BCD 编码的。对于分频器来说，由于不使用 q 端，只关注其进位信号 co 的情况，所以采用二进制也可以。

八、 实验心得

数字系统设计实验一共就 2 次实验，一次是试手，另一次就是像这样的一个大工程。不得不说，这样的工程确实比较振奋人心，看见电路板在闪烁你能够确信自己完成了某件东西，从头到尾，何况老师在非必要情形下不会主动来插手你的工作，因此对于数字系统的理解以及设计能力在其中都能够得到非常大的提升。

以前确实没有接触过 FPGA 以及 Verilog HDL 语言，但是不得不承认整个实验的核心器件其实是这块 FPGA，它代替我实现了太多功能。由于它是可编程的，你不能不承认它的强大，同时也不得不承认如果真的要从头凭自己的力量搭建一个硬件电路，其中遇到的困难会非常之多。因此在寝室里时你也得小心翼翼地看着 Basys3，遇到一点磨损都是不能接受的。

由于不论是数电实验还是电设实验我选择的都是倒计时定时器/数字式秒表（它们在计时这件事情上其实非常相像），所以舍友会看到我调试完一个定时器之后又掏了一个不一样的出来，其中还带着蜂鸣器，会怪叫起来，有时候代码写得有点问题，导致怪叫止不住，就在寝室里一刻不停地怪叫着，让午睡的舍友惊醒。正是这种相像向我说明了两个系统有着本质的区别——功能看起来很像，但是实现起来的路径完全是天差地别。一个的代码实现是直接利用时钟计时，另一个则是调用一个时间函数；但是回到硬件上好像又是一样的，毕竟计时最终都是通过一个是时钟电路实现的，只是两种板子封装的过程差的实在很多。

在上数电实验前我其实对于学习电路没什么好的印象，因为电子电路基础这门课实在是给了我太大的打击，不仅花了很多时间，还遭遇了人生中感觉最困难的考试，以及大学以来最糟糕的绩点；数电学起来也不算轻松，ISEE 的课程难度有时确实令人头疼；何况在“520”的夜晚抱着电路板共度实在是

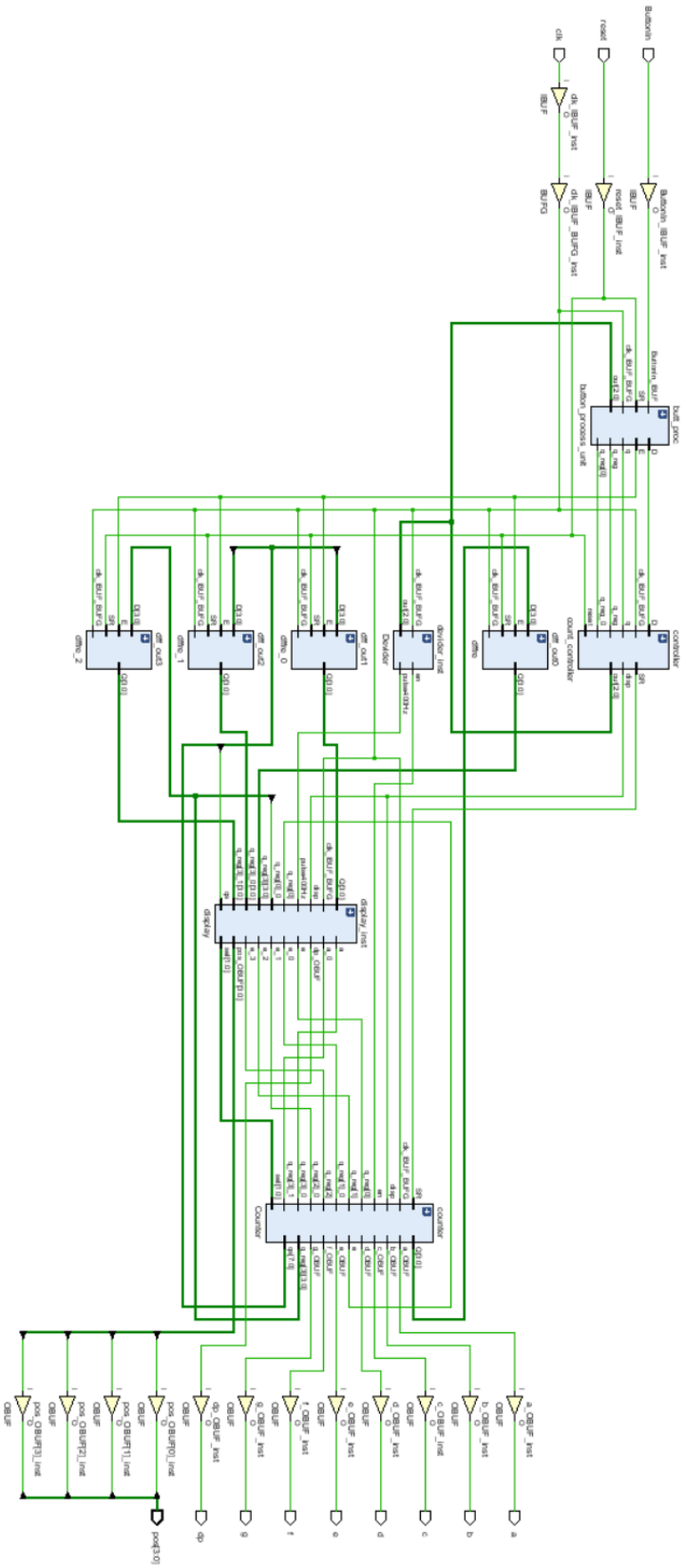

图 3.12: 数字式秒表测试电路的综合电路图

有那么一点点诙谐。但是实际上手调试起来感觉还是非常不错的，这也是我觉得 FPGA 那么厉害的原因。你只是需要一点时间去调试，去修改，去排除故障，然后电路就能给你你想要的结果，在它的世界里不存在没法修改的错误，乃至于像 Basys3 这样修改起来非常的容易，你只需要动动手指就行了——从这方面来说，它比世界上好多事情都要好得多得多。

回到正儿八经的内容，这次实验中我学习了怎么设计数字系统，怎么编写 VerilogHDL 程序，怎么使用 modelsim 软件进行仿真和调试，怎么使用 Vivado 实现电路的实现，怎么调试一个真的 FPGA 板子。总的来说内容很多也很完整，从原理到产品的一条路线贯穿其中。不同于那些理论课程，有时我在想实验课也是学习一门技能或者说手艺。现在的就业前景并不那么明朗，即使是我们这么所谓“hot”的专业也难免受到一些余波，如果将来真的有一天一切都不那么如意，凭着一门技能来获取一份工作，让生活能够继续下去我觉得也是令人高兴的。

最后，感谢任教数字系统设计实验的屈民军老师和唐奕老师，在实验过程中遭遇的困难两位老师都帮助了我很多，处理问题时也非常有耐心，没有两位老师的指导我无法完成我的设计项目。总而言之，如果要我来评价的话，我认为数字系统设计实验是一门非常不错的课程。

附录 A 实验中所使用的基本组合电路模块

1. 2 选 1 数据选择器

mux_2to1

```
module mux_2to1(out,in0,in1,addr);
//Declarations
parameter n=1;
output reg [n-1:0] out;
input [n-1:0] in0,in1;
input addr;

//Achievements
always @(*)
begin
  case(addr)
    1'b0: out = in0;
    1'b1: out = in1;
  endcase
end

endmodule
```

2. 4 选 1 数据选择器

mux_4to1

```
module mux_4to1(out, in0, in1, in2, in3, addr);
//Declarations
parameter n=1;
output reg [n-1:0] out;
input [n-1:0] in0,in1,in2,in3;
input [1:0] addr;

//Achievements
always @(*)
begin
  case(addr)
    2'b00: out = in0;
    2'b01: out = in1;
    2'b10: out = in2;
    2'b11: out = in3;
  endcase
end

endmodule
```

3. 2 线-4 线译码器

decoder

```
module decoder_24(out,in);
//Declarations
output [3:0] out;
input [1:0] in;
reg [3:0] out;

//Achievements
always @(*)
begin
  case(in)
//Low Voltage to be validate
   2'b00: out = 4'b1110;
   2'b01: out = 4'b1101;
   2'b10: out = 4'b1011;
   default: out = 4'b0111;
  endcase
end

endmodule
```

附录 B 实验中所使用的基本时序电路模块

1. 计数器/分频器

counter_n

```
module counter_n(clk,r,en,co,q);
//Declaration
parameter n=2;
parameter counter_bits=1;

input clk,en,r ;
output co;
output [counter_bits-1:0] q;

reg [counter_bits-1:0] q=0;
//Achievement
assign co=(q==(n-1)) && en;
always @(posedge clk)
begin
  if (r) q=0;
  else if (en)
  begin
    if (q==(n-1)) q=0;
           else q=q+1;
```

```
  end
end

endmodule
```

2. 计时器

timer

```
module timer(clk,r,en,done);
//Declaration
parameter n=10;
parameter counter_bits=4;

input clk,en,r;
output done;

reg [counter_bits-1:0] q=0;
//Achievement
assign done=(q==(n-1)) && en;
always @(posedge clk)
begin
  if (r) q=0;
  else if (en) q=q+1;
end

endmodule
```

3. 上升沿触发且初始值为 0 的 D 触发器

dffr

```
module dffr(d,r,clk,q);
//Declaration of the ports
input d,r,clk;
output reg q;

//Achievement
initial
begin
  q=1'b0;
end

always @(posedge clk or posedge r)
begin
  if (r) q=1'b0;
  else q=d;
end
```

```
endmodule
```

4. 带有使能控制端的 D 触发器

dffre

```
module dffre(d,en,r,clk,q);
//Declaration of the ports
parameter n=1;
input en,r,clk;
input [n-1:0]d;
output reg [n-1:0]q;

//Achievement
always @(posedge clk)
  if (r) q={n{1'b0}};
  else if (en) q=d;

endmodule
```