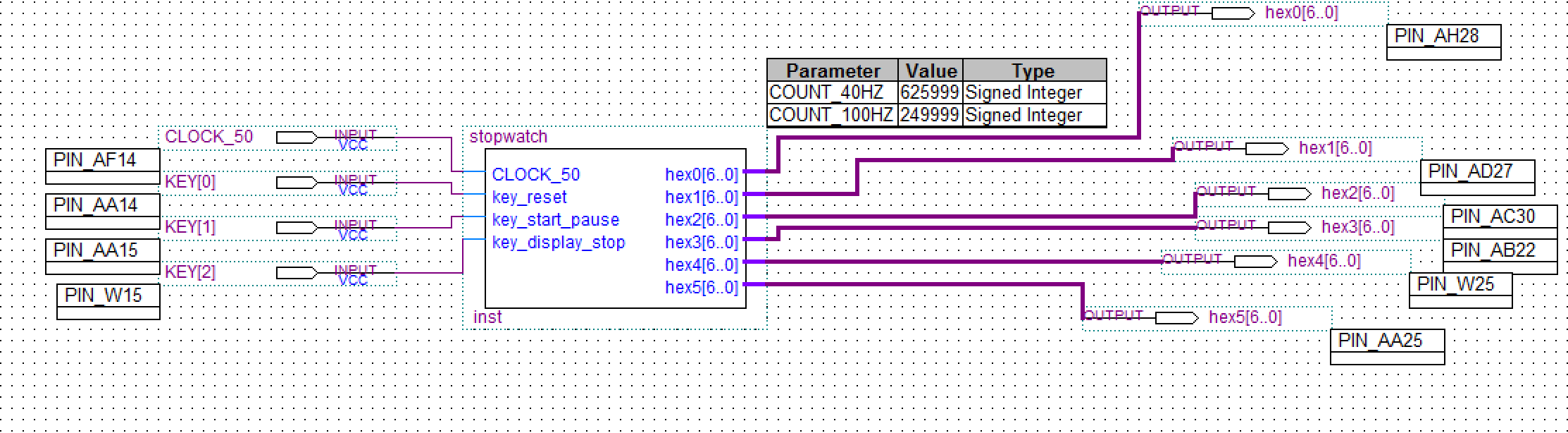
**数字部件秒表实验报告**

515030910211 姜子悦

1. 实验目的
2. 学习Verilog硬件描述语言基本语法，学习使用quantusii以及fpga。
3. 理解并行程序的编程思维和我们平时串行编程思维的不同之处。
4. 理解硬件和软件是相辅相成、并在设计和应用方法上的优势互补的特点。
5. 实验电路



其中，stopwatch模块参考了实验指导第17页上的电路，并加入了两个消抖模块，对key\_start\_pause和key\_display\_stop进行消抖。

1. 功能分析及源代码

本次实验秒表需要有一下功能：基础分-秒-毫秒计时和显示功能，按键控制reset，start/stop，display/stop display的功能；以及按键消抖。

我的设计分为三个模块，sevenseg、stopwatch、eliminate\_jitter。Sevenseg模块和实验书上的实现一样，负责显示管的显示转换；

stopwatch模块为秒表主设计，其流程为：转化一个100hz的时钟信号用以秒表计时，转化一个40hz信号用于消抖模块的时钟，秒表本身是一个6位的加法器，其中毫秒低位，毫秒高位，秒低位，分低位是十进制加法器，其余为6进制加法器。

Eliminate\_jitter模块负责消抖，经历几个版本更迭，目前的做法是每个40hz时钟的上升沿采集一次按键电平，记录下本次采集结果，并输出之前两次采集结果的或。其原理是抖动的周期一般较短，25ms的采集频率下，连续两次获取到电平0，并且两次之间没有key的上升沿被触发，代表按键确实被按下且按下了一次，对于本次试验可以很好的消抖。

：

Stopwatch秒表代码如下：

module stopwatch(

CLOCK\_50, key\_reset, key\_start\_pause, key\_display\_stop,

hex0, hex1, hex2, hex3, hex4, hex5

);

input CLOCK\_50, key\_reset, key\_start\_pause, key\_display\_stop;

output [6:0] hex0, hex1, hex2, hex3, hex4, hex5;

//fir clock 40HZ

parameter COUNT\_40HZ = 625999;

//for clock 100HZ

parameter COUNT\_100HZ = 249999;

reg [3:0] minute\_display\_high;

reg [3:0] minute\_display\_low;

reg [3:0] second\_display\_high;

reg [3:0] second\_display\_low;

reg [3:0] msecond\_display\_high;

reg [3:0] msecond\_display\_low;

reg [3:0] minute\_counter\_high;

reg [3:0] minute\_counter\_low;

reg [3:0] second\_counter\_high;

reg [3:0] second\_counter\_low;

reg [3:0] msecond\_counter\_high;

reg [3:0] msecond\_counter\_low;

reg counter\_stop, display\_stop;

wire counter, display;

//10ms

reg [31:0] counter\_50M;

//50ms

reg [31:0] counter\_250M;

//100HZ clock

reg CLOCK\_100HZ;

//40HZ clock

reg CLOCK\_40HZ;

reg reset\_1\_time;

reg [31:0] counter\_reset;

reg start\_1\_time;

reg [31:0] counter\_start;

reg display\_1\_time;

reg [31:0] counter\_display;

sevenseg LED8\_minute\_display\_high(minute\_display\_high, hex5);

sevenseg LED8\_minute\_display\_low(minute\_display\_low, hex4);

sevenseg LED8\_second\_display\_high(second\_display\_high, hex3);

sevenseg LED8\_second\_display\_low(second\_display\_low, hex2);

sevenseg LED8\_msecond\_display\_high(msecond\_display\_high, hex1);

sevenseg LED8\_msecond\_display\_low(msecond\_display\_low, hex0);

eliminate\_jitter ej\_counter(.CLOCK(CLOCK\_40HZ), .RESET(key\_reset), .key\_in(key\_start\_pause), .key\_out(counter));

eliminate\_jitter ej\_display(.CLOCK(CLOCK\_40HZ), .RESET(key\_reset), .key\_in(key\_display\_stop), .key\_out(display));

//produce 10ms(100HZ) clock

always @ (posedge CLOCK\_50)

begin

if (counter\_50M == COUNT\_100HZ)

begin

counter\_50M <= 0;

CLOCK\_100HZ <= ~CLOCK\_100HZ;

end

else

begin

counter\_50M <= counter\_50M + 1;

end

end

//produce 25ms(40HZ) clock

always @ (posedge CLOCK\_50)

begin

if (counter\_250M == COUNT\_40HZ)

begin

counter\_250M <= 0;

CLOCK\_40HZ <= ~CLOCK\_40HZ;

end

else

begin

counter\_250M <= counter\_250M + 1;

end

end

always @ (posedge key\_start\_pause)

begin

if (!counter)

begin

counter\_stop <= ~counter\_stop;

end

end

always @ (posedge key\_display\_stop)

begin

if (!display)

begin

display\_stop <= ~display\_stop;

end

end

always @ (posedge CLOCK\_100HZ)

begin

//reset

if (~key\_reset)

begin

msecond\_counter\_low <= 0;

msecond\_counter\_high <= 0;

second\_counter\_low <= 0;

second\_counter\_high <= 0;

minute\_counter\_low <= 0;

minute\_counter\_high <= 0;

msecond\_display\_low <=0;

msecond\_display\_high <= 0;

second\_display\_low <= 0;

second\_display\_high <= 0;

minute\_display\_low <= 0;

minute\_display\_high <= 0;

end

//update counter

if (~counter\_stop)

begin

if (msecond\_counter\_low == 4'h9)

begin

msecond\_counter\_low <= 4'h0;

msecond\_counter\_high <= msecond\_counter\_high + 4'h1;

end

else

begin

msecond\_counter\_low <= msecond\_counter\_low + 4'h1;

end

if (msecond\_counter\_high == 4'ha)

begin

msecond\_counter\_high <= 4'h0;

second\_counter\_low <= second\_counter\_low + 4'h1;

end

if (second\_counter\_low == 4'ha)

begin

second\_counter\_low <= 4'h0;

second\_counter\_high <= second\_counter\_high + 4'h1;

end

if (second\_counter\_high == 4'h6)

begin

second\_counter\_high <= 4'h0;

minute\_counter\_low <= minute\_counter\_low + 4'h1;

end

if (minute\_counter\_low == 4'ha)

begin

minute\_counter\_low <= 4'h0;

minute\_counter\_high <= minute\_counter\_high + 4'h1;

end

end

//update display

if (~display\_stop)

begin

msecond\_display\_low <= msecond\_counter\_low;

msecond\_display\_high <= msecond\_counter\_high;

second\_display\_low <= second\_counter\_low;

second\_display\_high <= second\_counter\_high;

minute\_display\_low <= minute\_counter\_low;

minute\_display\_high <= minute\_counter\_high;

end

end

endmodule

消抖模块如下：

module eliminate\_jitter(

CLOCK, RESET, key\_in,

key\_out

);

input CLOCK, RESET, key\_in;

output key\_out;

reg key\_now, key\_pre, key\_prepre;

always @ (posedge CLOCK)

begin

if (~RESET)

begin

key\_now <= 1;

key\_pre <= 1;

key\_prepre <= 1;

end

else

begin

key\_prepre <= key\_pre;

key\_pre <= key\_now;

key\_now <= key\_in;

end

end

wire key\_out = key\_pre | key\_prepre;

endmodule

1. 结果分析与改进

在第一次迭代结束给助教检查时，主要完成了秒表计时功能和三个按键功能，秒表功能比较简单，表现良好。不过当时对消抖的理解有些问题，采用的实现为连续采集多个clk上升沿时的按键电平并计数0和1，如果1的出现次数小于某个设定好的常数，就认为不再抖动。理论分析和实验表明这个方法有问题，因为多个抖动的1电平可能正好落在连续的采集内。

后来在助教的讲解下，对消抖的原理和原因有了正确的理解，于是采取了新的消抖方式。每隔25ms采集一次按键电平，取25ms是尽可能忽略抖动，若按键电平连续出现0而且没有出现按键的上升沿夹杂在其中则代表按键确实被按下。实验表明，这个方法的效果良好，三个按键都可以按下的第一次就生效，而且只生效一次。

1. 实验总结

本次试验让我学会了基本的verilog、quantus、fpga的应用，并且发现了原来板子存在的问题及时更换，学会了在自己的虚拟机中识别开发版的驱动，为之后的实验奠定基础。

实验中让我比较印象深刻的就是并行的思想。

我常常纠结与某几句话到底是被实现为了串行还是并行，这个问题经常会对我编写逻辑造成困然。比如在always下多个if语句的串并行问题。Google结果支持两种的都有，而经过我亲身实验，发现至少在我的实现中（非阻塞赋值，下一句if需要用到上一句的结果），它是串行执行的。不过我不清楚如果多句if没有显式关联，逻辑是否还会是串行执行。

包括多个always、assign的并行执行和一个寄存器不能被多个always赋值等软件编程不会遇到的问题，都令我十分苦恼。好在，fpga最大的好处就在于作为硬件，它可以及时显示出各种结果，很便于亲身实验这类问题的结果。（不过编译一次用时真的好久……）

还有就是对上课时候学到的知识（电路，逻辑）有了更好的理解，作为一个软件的学生，在大物之外很少碰到电路，所以一开始有些手忙脚乱，不知道该怎么连接电路。不过经过first fpga和秒表两个实验，以及quantus方便的功能，使我对连接模块有了一些体会。而且比如消抖这种上课自以为听懂了的知识，只有在亲手做的以后才明白其实自己一知半解。只有动手实践才会真正对这门课有更真实、确切的掌握。