实验三：频率计设计

丁文浩 无 43 2014011079

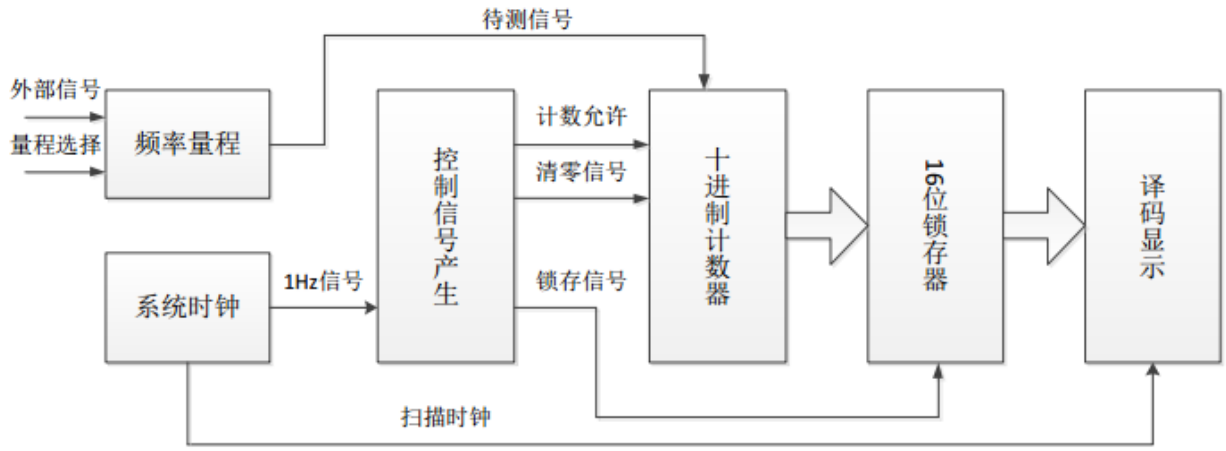
一、实验目的

掌握频率计的原理和设计方法。

二、设计方案

（1）原理

利用系统时钟产生 1Hz 的控制信号，在 1s 的时长内利用计数器对待测信号进行计数，将计数结果锁存（或者保存，不是指 latch） 并输出到数码管中显示。 其中频率量程模块负责根据设定的量程控制信号决定是否对输入信号进行 10 分频；系统时钟模块根据外部输入的参考时钟产生标准 1Hz 的控制信号；控制信号产生模块产生计数所需的使能、清零信号以及保存测量结果所需的锁存信号和扫描显示所需的扫描时钟信号；十进制计数模块在计数使能、清零信号控制下对外部输入信号（或其 10 分频信号）在 1s 周期内对其进行计数操作；锁存器模块在计数完成之后对计数结果进行锁存，保存上一测量周期的测量结果；译码显示模块将测量结果输出到 LED 数码管显示， 采用扫描的方式实现多位数据的同时显示。

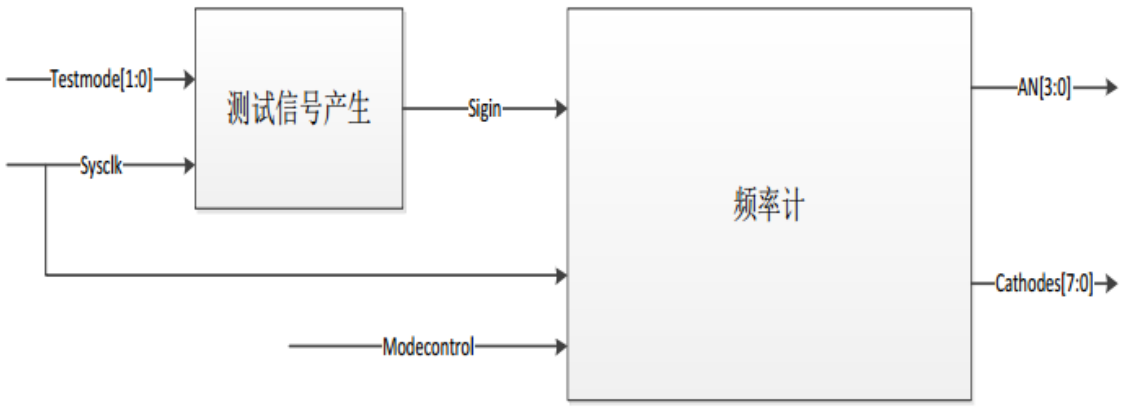


实际方案中模块划分：控制信号产生模块（ControlSignal.v）、十进制计数模块（FourBitDecimalCounter.v）、锁存器模块（Latcher.v）、时钟产生模块（SystemClk.v）、译码模块（Decoder.v）、量程选择模块（FrequencyChoose.v）。

（2）测试模块

测试方法：为了测试方便，在实验指导书的最后提供一个待测信号输入模块

signalinput(testmode[1:0],sysclk,sigin)，其中 testmode[1:0]接到 SW1~SW0 的开关输入， 00,01,10,11 分别选择 4 个不同频率的信号。 sysclk 是开发板提供的 100MHz 时钟， sigin 是输出的待测信号。要求同学按照下图所示对测试信号输入模块以及自行设计的频率计模块进行连接：



（3）关键代码

test.v —— 顶层文件，分为两个部分一个是信号输入模块，产生指定的四个信号，另外一个模块就是编写的频率计模块。

```
module test(testmode, clk, modecontrol, highfreq, cathodes, AN);
  input [1 : 0] testmode;
  input clk;
  input modecontrol;
  output highfreq;
  output [6 : 0] cathodes;
  output [3 : 0] AN;
  wire sigin;

  // input four signals to be detected
  signalinput signalin(.testmode(testmode),
                       .sysclk(clk),
                       .sigin1(sigin));

  // the module of frequency detection
  frequency freq(.sigin(sigin),
                 .testmode(testmode),
                 .sysclk(clk),
                 .modecontrol(modecontrol),
                 .highfreq(highfreq),
                 .cathodes(cathodes),
                 .AN(AN));
endmodule

```

signalinput.v —— 信号输入模块

```v
// use SW0 and SW1 to control the frequency of the signal
module signalinput(testmode, sysclk, sigin1);
    input [1 : 0] testmode;
    input sysclk;
    output sigin1;
    reg[20 : 0] state;
    reg[20 : 0] divide;
    reg sigin;

    assign sigin1 = sigin;
    initial begin
        sigin = 0;
        state = 21'b000000000000000000000;
        divide = 21'b000000_1111_1010_0000000;
    end

    always @(testmode) begin
        case(testmode[1 : 0])
            2'b00 : divide = 21'b000000_1111_1010_0000000;  //3125Hz
            2'b01 : divide = 21'b0000000_1111_1010_000000;  //6250Hz
            2'b10 : divide = 21'b1111_0100_0010_0100_00000; //50Hz
            2'b11 : divide = 21'b00000000_1111_1010_00000;  //12500Hz
        endcase
    end

    always@(posedge sysclk) begin
        if(state == 0)
            sigin = ~sigin;
            state = state + 21'b0_00__0000_0000_0000_0000_10;
        if(state == divide)
            state = 27'b000_0000_0000_0000_0000_0000_0000;
    end
endmodule
```

SystemClk.v —— 产生扫描时钟和控制时钟的模块

```
// input the clock pulse and get two subclk
// one is used as control signal
// the other is used as scan signal to show the leds
module SystemClk(clk, clkControl, clkScan);
  input clk;
  output clkControl, clkScan;
  reg sigmentation1, sigmentation2;
  integer count1, count2;

  // scan signal is 1KHz
  // control signal is 1Hz
  assign clkScan = sigmentation1;
  assign clkControl = sigmentation2;

  initial begin
      sigmentation1 <= 0;
      sigmentation2 <= 0;
      count1 <= 0;
      count2 <= 0;
  end

  // every two times the signal reverse we get one period
  // 1KHz signal for control
  always @(posedge clk) begin
      if(count1 == 49999) begin
          count1 <= 0;
          sigmentation1 <= ~sigmentation1;
      end
      else
        count1 <= count1 + 1;
  end

   // 1Hz clock used for scan
   always @(posedge sigmentation1) begin
       if(count2 == 499) begin
           count2 <= 0;
           sigmentation2 <= ~sigmentation2;
       end
       else
         count2 <= count2 + 1;
   end
endmodule

```

Latcher.v —— 锁存器模块，用于保存十进制计数的结果

```
// when the signal lock is 0, we can transfer
// when the signal is 1, keep the last number
module Latcher(latch, inData, outData);
  input latch;
  input [15 : 0] inData;
  output reg [15 : 0] outData;

  always @(latch, inData) begin
    if(!latch)
        outData <= inData;
    end
endmodule
```

FrequencyChoose.v —— 通过控制开关选择不同的量程。

```v
// we use SW7 to choose the mode
// if the signal is a high frequency signal
// we divide it by 10
// and to show we have done this, using LED7 to bright.
module FrequencyChoose(signalIn,
                       frequencyControl,
                       signalOut,
                       highFrequency);
  input signalIn, frequencyControl;
  output signalOut, highFrequency;

  reg divideFrequency;  // to remember the divided frequency
  integer count;
  wire highFrequency;

  initial begin
    count <= 0;
    divideFrequency <= 0;
  end

  assign signalOut = (frequencyControl == 1)? divideFrequency : signalIn;
  assign highFrequency = frequencyControl;   // light the led

  // we should count the number every 5 times
  // then the signal will be divided into 1/10
  always @(posedge signalIn) begin
    if(count == 4) begin
      count <= 0;
      divideFrequency <= ~divideFrequency;
    end
    else
      count <= count + 1;
  end
endmodule
```

frequency.v —— 频率计模块代码

```
// the whole module concluding all submodules
module frequency(sigin,
                 testmode,
                 sysclk,
                 modecontrol,
                 highfreq,
                 cathodes,
                 AN);

  input sigin, sysclk, modecontrol;
  input [1 : 0] testmode;

  output highfreq;
  output [6 : 0] cathodes;
  output [3 : 0] AN;

  // clear signal is generated by the control module
  // clear is used to reset the counter
  wire sigto, enable, latch, clear, clkscan, clkcont;
  wire [15 : 0] num1, num2;

  // if the signal's frequency is too high, divide it
  FrequencyChoose        fc(.signalIn(sigin),
                            .frequencyControl(modecontrol),
                            .highFrequency(highfreq),
                            .signalOut(sigto));
  // generate all clatch needed
  SystemClk              sc(.clk(sysclk),
                            .clkScan(clkscan),
                            .clkControl(clkcont));
```

```
31    // generate control signal
32    ControlSignal          cs(.clkControl(clkcont),
33                              .testMode(testmode),
34                              .modeControl(modecontrol),
35                              .enable(enable),
36                              .latch(latch),
37                              .clear(clear));
38    // four decimal counter in one module
39    FourBitDecimalCounter  dec(.clk(sigto),
40                               .enable(enable),
41                               .clear(clear),
42                               .num(num1));     // output the number
43    // latcher
44    Latcher                l(.latch(latch),     // latch the number
45                             .inData(num1),
46                             .outData(num2));
47    // to show the number in leds
48    Decoder                de(.clkScan(clkscan),
49                              .inData(num2),
50                              .AN(AN),
51                              .out(cathodes));
52  endmodule
```

FourBitDemicalCounter.v —— 四位十进制计数器，用于记录频率

```v
// DecimalCounter for clock when enable signal is true
module DecimalCounter(clk, enable, clear, out, num);
  input clk, enable, clear;
  output out;
  output [3 : 0] num;
  reg out;
  reg [3 : 0] num;

  always @(posedge clk or posedge clear) begin
    if(clear) begin
      out <= 0;
      num <= 0;
    end
    else if(enable) begin
      if(num == 4'b1001) begin  // reach the number 10
        num <= 0;
        out <= 1;
      end
      else begin
        out <= 0;
        num <= num + 4'b1;   // add 1
      end
    end
  end
endmodule


// using four decimal counter
module FourBitDecimalCounter(clk, clear, enable, num);
  input clk, clear, enable;
  output [15 : 0] num;
  wire c0, c1, c2, c3;
  wire [15 : 0] num;

```

```
35    // asyn counter
36    // when the low counter output a signal, the higer one counts
37    DecimalCounter dc0(.clk(clk),
38                       .clear(clear),
39                       .enable(enable),
40                       .num(num[3 : 0]),
41                       .out(c0));
42    DecimalCounter dc1(.clk(c0),
43                       .clear(clear),
44                       .enable(enable),
45                       .num(num[7 : 4]),
46                       .out(c1));
47    DecimalCounter dc2(.clk(c1),
48                       .clear(clear),
49                       .enable(enable),
50                       .num(num[11 : 8]),
51                       .out(c2));
52    DecimalCounter dc3(.clk(c2),
53                       .clear(clear),
54                       .enable(enable),
55                       .num(num[15 : 12]),
56                       .out(c3));
57  endmodule
```

Decoder.v —— 译码器，包含扫描电路

```v
// the BCD coders for the leds
module BCD7(in, out);
  input [3 : 0] in;
  output [6 : 0] out;

  assign out =
         (in == 4'h0) ? 7'b000_0001:
         (in == 4'h1) ? 7'b100_1111:
         (in == 4'h2) ? 7'b001_0010:
         (in == 4'h3) ? 7'b000_0110:
         (in == 4'h4) ? 7'b100_1100:
         (in == 4'h5) ? 7'b010_0100:
         (in == 4'h6) ? 7'b010_0000:
         (in == 4'h7) ? 7'b000_1111:
         (in == 4'h8) ? 7'b000_0000:
         (in == 4'h9) ? 7'b000_0100:
         7'b0;    // default number is "8"
endmodule


// use four leds to show one thousand numbers
module Decoder(inData, clkScan, AN, out);
  input [15 : 0] inData;       // four number (decimal)
  input clkScan;               // clock pulse
  output [6 : 0] out;          // every leds
  output [3 : 0] AN;           // choose the led
  reg [3 : 0] AN, everyData;

  // the initial value can be any one of these four cases
  initial AN = 4'b1101;

```

```
32    // scan the led and choose one to light
33    always @(posedge clkScan) begin
34      case(AN)
35       4'b1110 : begin
36         AN <= 4'b0111;
37         everyData <= inData[15 : 12];
38       end
39       4'b0111 : begin
40         AN <= 4'b1011;
41         everyData <= inData[11 : 8];
42       end
43       4'b1011 : begin
44         AN <= 4'b1101;
45         everyData <= inData[7 : 4];
46       end
47       4'b1101 : begin
48         AN <= 4'b1110;
49         everyData <= inData[3 : 0];
50        end
51      endcase
52    end
53
54    BCD7 b(.in(everyData), .out(out));
55  endmodule
```

ControlSignal.v —— 产生所有的控制信号

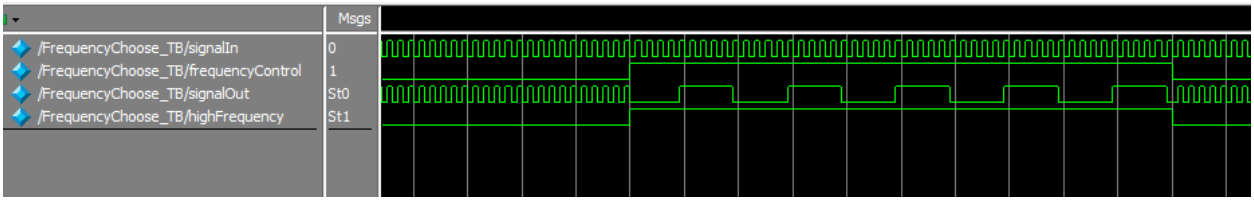
只有在量程开关或者信号选择开关改变的时候才重新计数，每次计数之后下一次时钟上升沿停止计时并把计数的值显示在数码管上。

```
module ControlSignal(clkControl, testMode, modeControl, enable, clear, latch);
  input clkControl, modeControl;
  input [1 : 0] testMode;
  output enable, clear, latch;
  reg enable, clear, latch;

  // remeber last modes
  reg [1 : 0]oldMode;
  reg oldModeControl;

  initial begin
    enable <= 0;
    clear <= 1;
    latch <= 1;
    oldModeControl <= 0;
  end

  // every time when the clk reaches
  always @(posedge clkControl) begin
  // when mode changed we should count
    if(testMode != oldMode || oldModeControl != modeControl) begin
      latch = 0;    // dont lock
      clear = 0;    // clear to be ready to count
      enable = 1;   // start to count
      // mode changed
      oldMode = testMode;
      oldModeControl = modeControl;
    end
    else begin
      enable = 0; // disable to count
      clear = 1;  // don't clear
      latch = 1;  // lock the number
    end
  end
endmodule
```

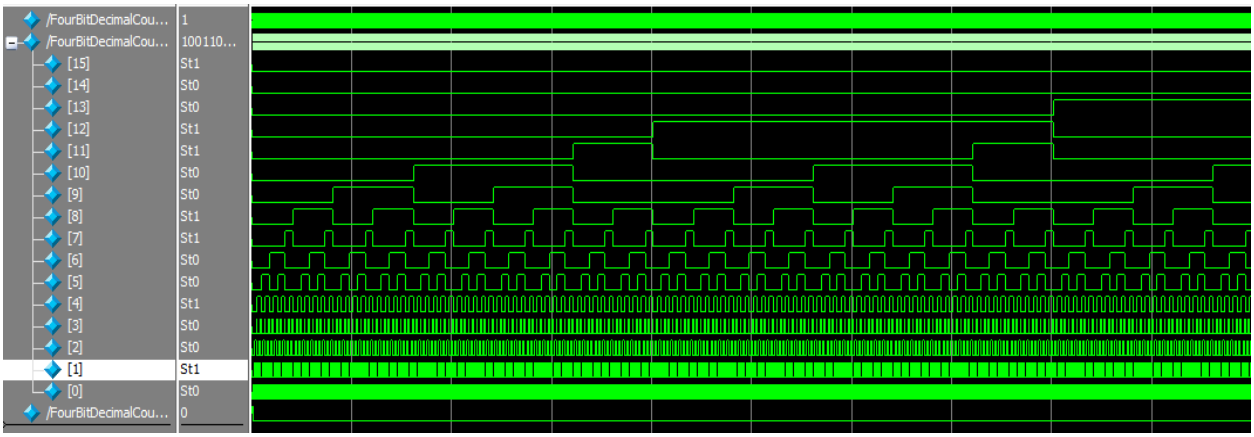
（4）仿真波形

FrequencyChoose_TB：

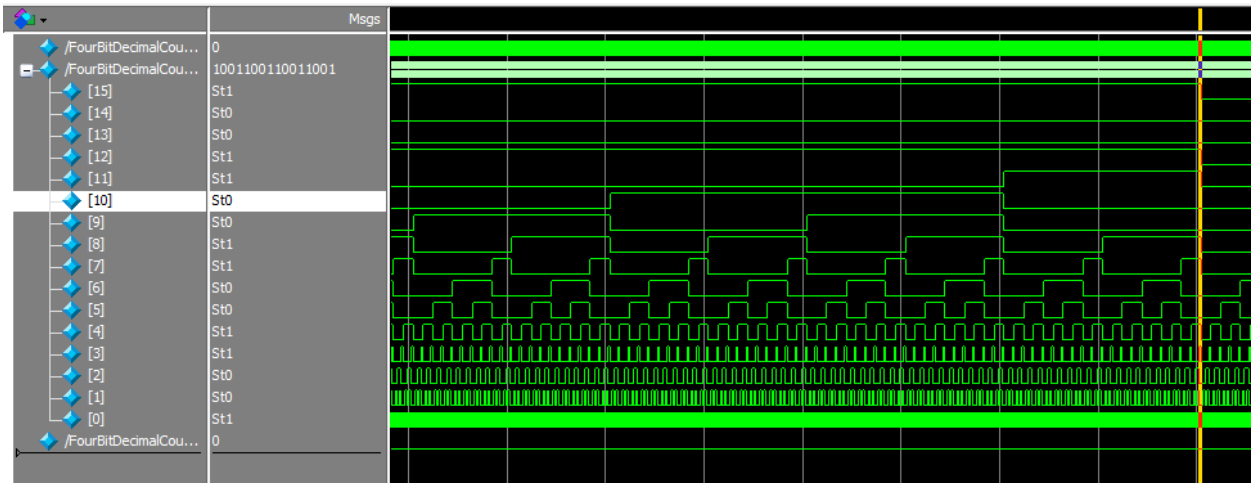
对量程选择模块的测试，可以看到，当 frequencyControl 信号为 0 时输出的时钟即为输入的时钟频率，当 frequencyControl 为 1 时输出的信号为输入信号的十分频，并且在 highFrequency 输出端输出高电平指示目前处于高量程模式。



FourBitDecimalCounter_TB：

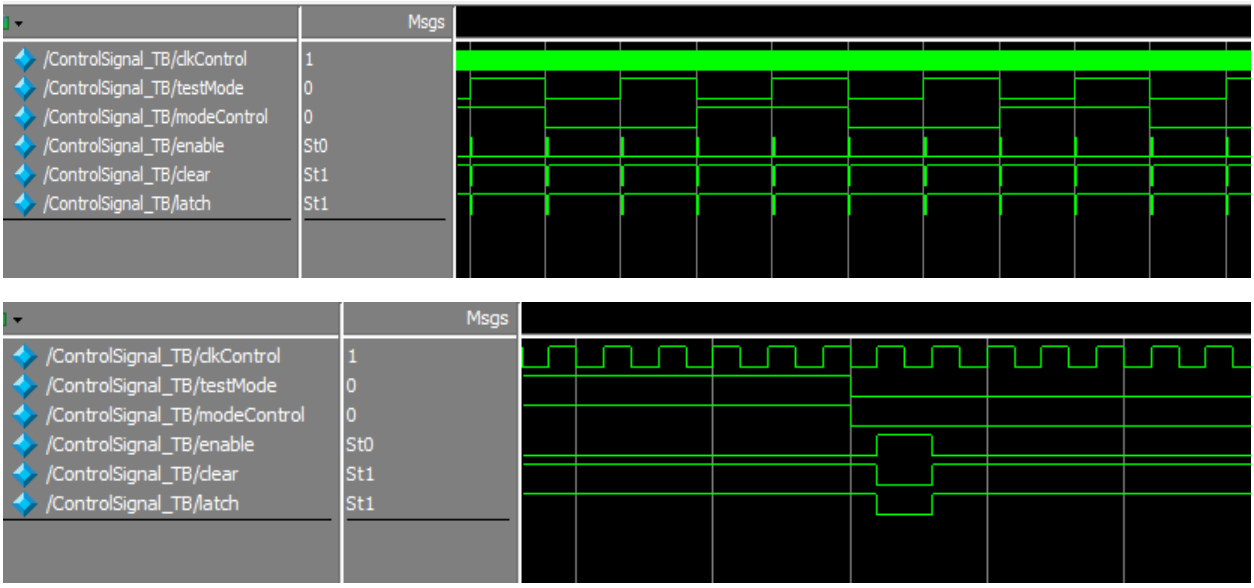


上图为 4 位十进制计数器的技术过程，可以看出从低位到高位正在不断进位。由于是 10 进制计数器，所以低位每次到 9 之后就会进位。



上图位置显示的计数值为 9999，二进制计数为 1001100110011001，下一个时钟上升沿就恢复为 0

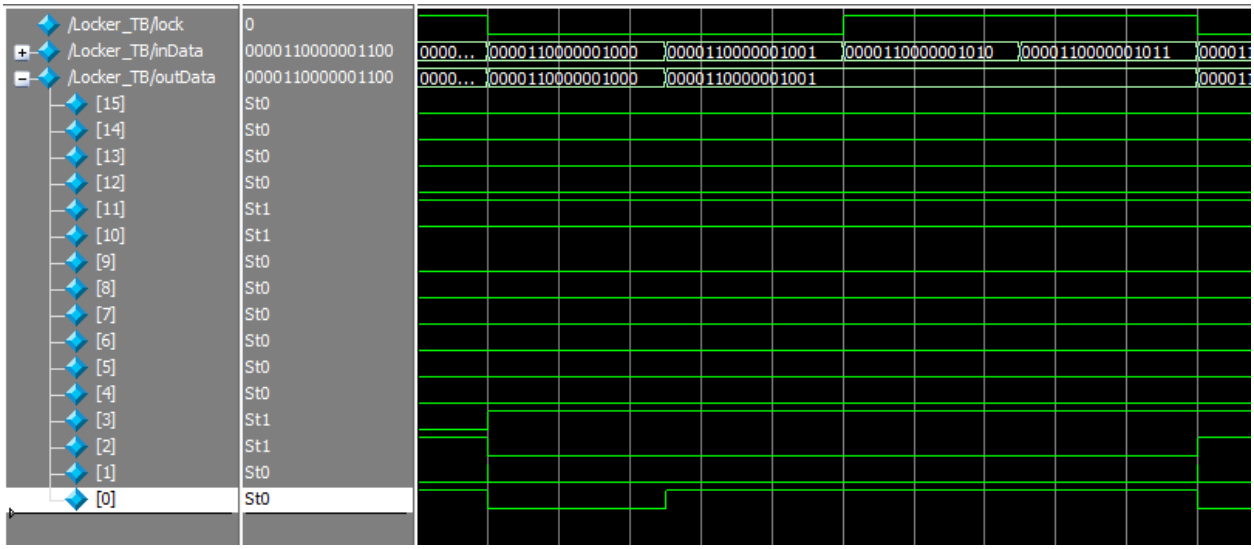
SignalControl_TB.v



有仿真波形可以看出，在每一次量程开关或者信号选择开关改变的时候，下一个时钟上升沿三个信号都会改变，并且在下一个时钟上升沿，三个信号恢复原来的值。

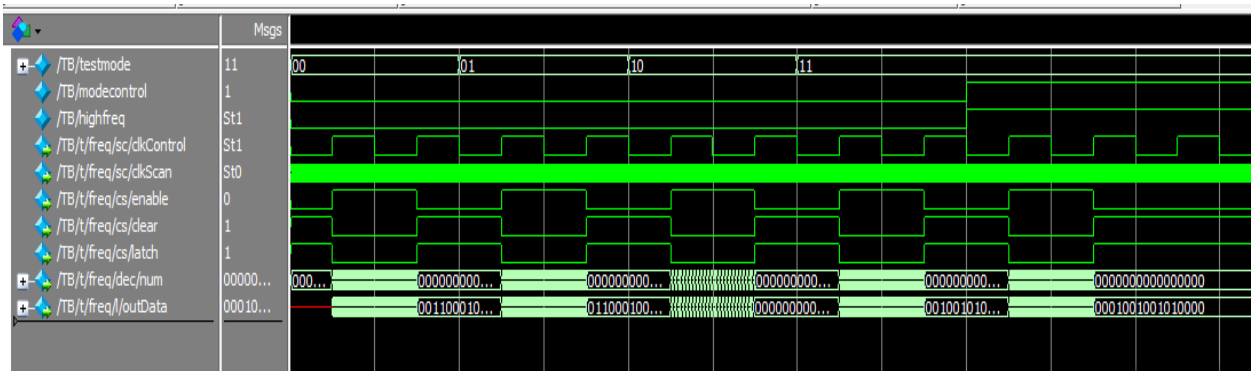
原来的值即为锁存不计数状态。

Latcher_TB.v

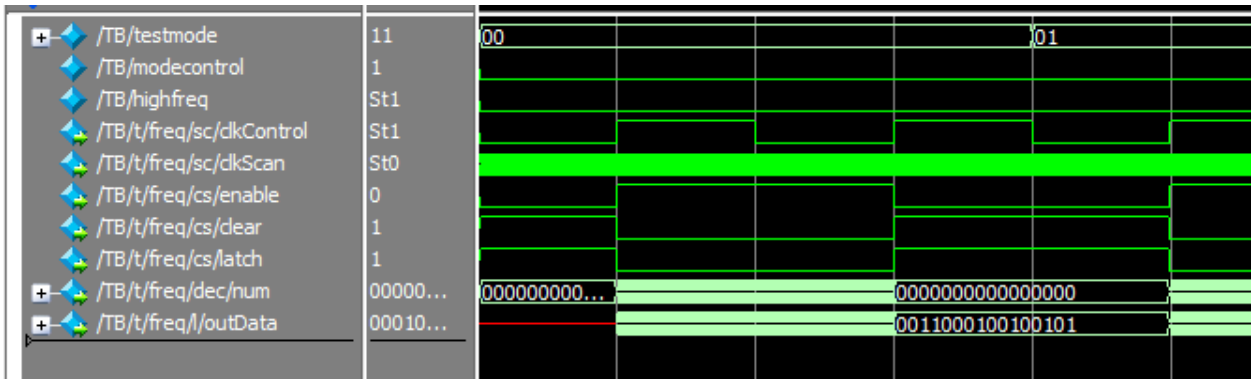


锁存器模块，当锁存信号为 0 的时候透明传输，当锁存信号为 1 的时候保持之前的输出。

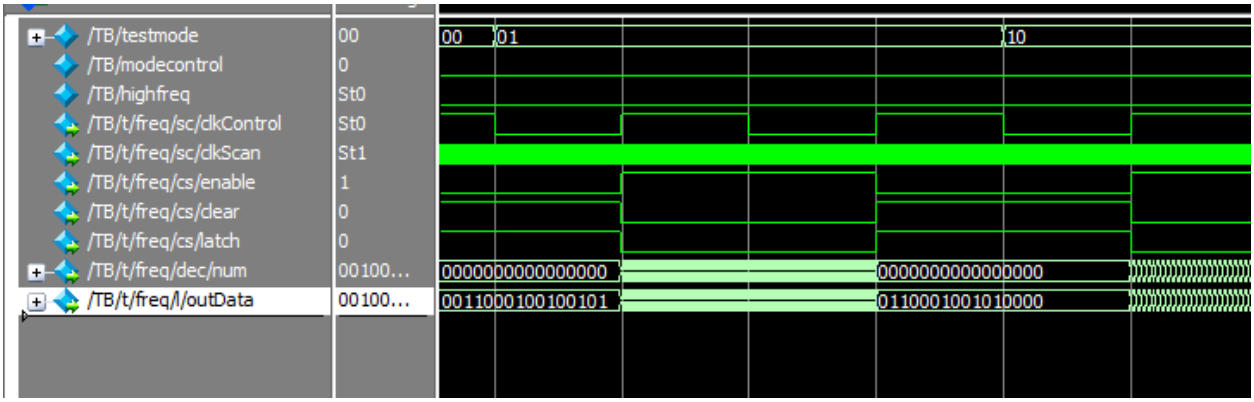
整体模块仿真 top_TB.v



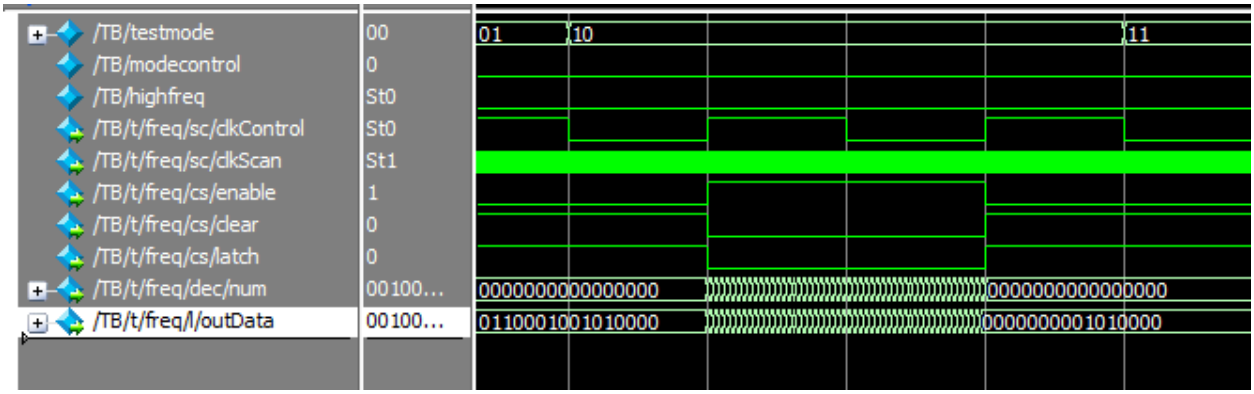
上图是 4 个输入信号以及最后一个 12500Hz 信号使用高量程的结果，可以看出在 clkControl 的上升沿时如果输入信号改变则会开始计时，并且在下一个时钟上升沿停止计时，latch 信号保持上一次的计数值。



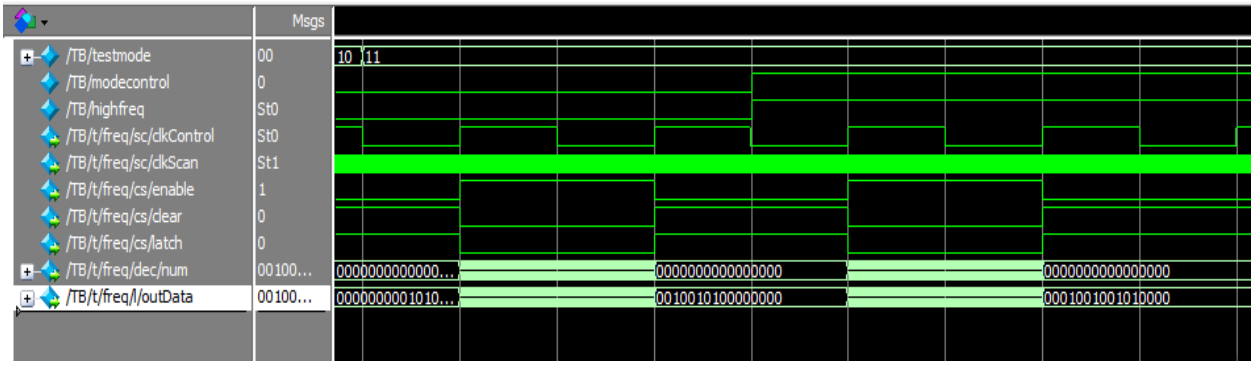
当开关处于 00 时，输入信号的频率为 3125Hz，而计数结果的二进制码为 0011000100100101，转化为十六进制码即为 3125Hz。



当开关处于 01 时，输入信号的频率为 6250Hz，而计数结果的二进制码为 0110001001010000，转化为十六进制码即为 6250Hz。



当开关处于 10 时，输入信号的频率为 50Hz，而计数结果的二进制码为 0000000001010000，转化为十六进制码即为 50Hz。



当开关处于 11 时，输入信号的频率为 12500Hz，而计数结果的二进制码为 0010010100000000，转化为十六进制码即为 2500Hz，这是因为已经超过了计数器的量程高位产生了丢失。在下一个时钟周期令 modeControl 为 1，选择高量程可以看到计数器再次重新计数，计数结果为 0001001001010000，转化为十六进制结果为 1250Hz。

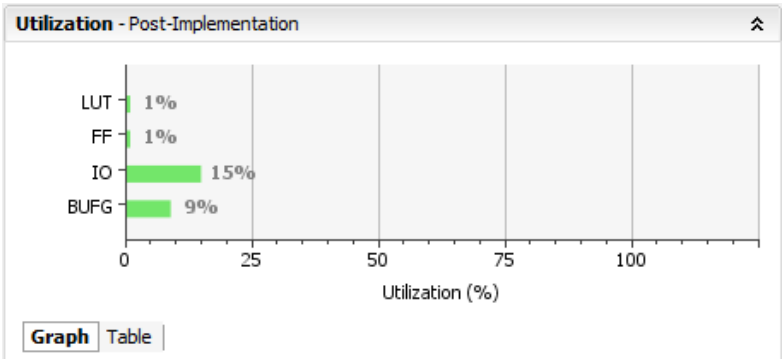
（5）测量范围与测量精度分析

测量范围分析：由于我们可以选择高量程，即 10 倍的关系，而我们的计数上限为 9999，所以为了保证精度不损失的情况下可以测量的最大精确频率为 99990Hz。

测量精度分析：由于采用 4 位数码管，所以只能够测量个位的精度，对于小数级别的频率不能测量。测试分频比为 64000 的信号，其频率应为 1562.5Hz，但实际显示值为 1562Hz。

三、综合情况

1.面积报告



Utilization - Post-Implementation

Resource	Utilization	Available	Utilization %
LUT	71	20800	0.34
FF	150	41600	0.36
IO	16	106	15.09
BUFG	3	32	9.38

Graph **Table**

2.时序性能

（1）时钟分析

Clock Summary

Name	Waveform	Period (ns)	Frequency (MHz)
CLK	{0.000 5.000}	10.000	100.000

（2）建立时间与保持时间和脉宽分析

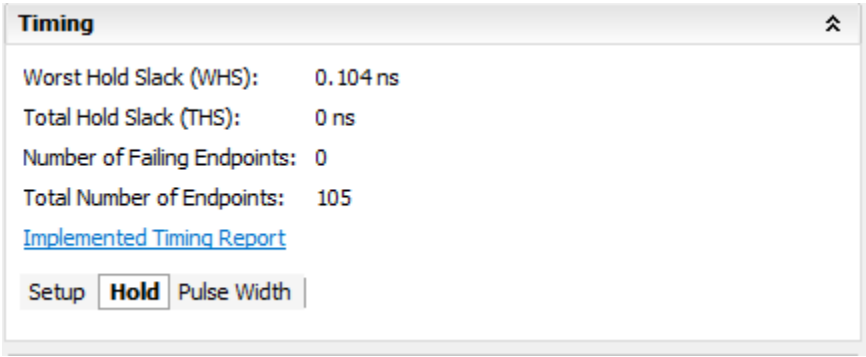
建立时间：

Timing

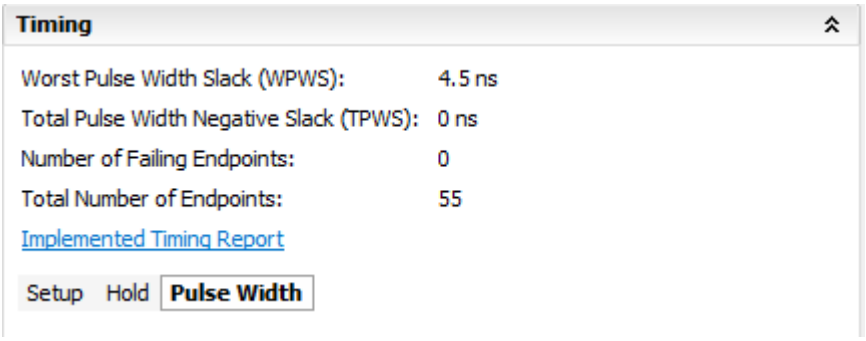
Worst Negative Slack (WNS): 4.202 ns
Total Negative Slack (TNS): 0 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 105
Implemented Timing Report

Setup Hold Pulse Width

保持时间：

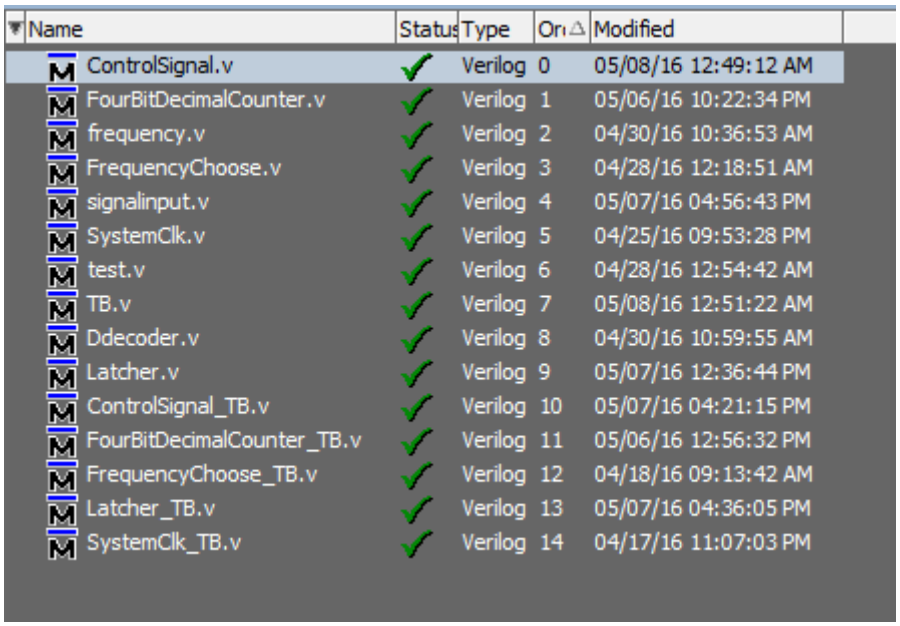

Timing
Worst Hold Slack (WHS): 0.104 ns
Total Hold Slack (THS): 0 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 105
Implemented Timing Report
Setup Hold Pulse Width

脉宽分析：


Timing
Worst Pulse Width Slack (WPWS): 4.5 ns
Total Pulse Width Negative Slack (TPWS): 0 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 55
Implemented Timing Report
Setup Hold Pulse Width

四、实验总结

1.代码清单



Name	Status	Type	Ori	Modified
ControlSignal.v	✓	Verilog	0	05/08/16 12:49:12 AM
FourBitDecimalCounter.v	✓	Verilog	1	05/06/16 10:22:34 PM
frequency.v	✓	Verilog	2	04/30/16 10:36:53 AM
FrequencyChoose.v	✓	Verilog	3	04/28/16 12:18:51 AM
signalinput.v	✓	Verilog	4	05/07/16 04:56:43 PM
SystemClk.v	✓	Verilog	5	04/25/16 09:53:28 PM
test.v	✓	Verilog	6	04/28/16 12:54:42 AM
TB.v	✓	Verilog	7	05/08/16 12:51:22 AM
Ddecoder.v	✓	Verilog	8	04/30/16 10:59:55 AM
Latcher.v	✓	Verilog	9	05/07/16 12:36:44 PM
ControlSignal_TB.v	✓	Verilog	10	05/07/16 04:21:15 PM
FourBitDecimalCounter_TB.v	✓	Verilog	11	05/06/16 12:56:32 PM
FrequencyChoose_TB.v	✓	Verilog	12	04/18/16 09:13:42 AM
Latcher_TB.v	✓	Verilog	13	05/07/16 04:36:05 PM
SystemClk_TB.v	✓	Verilog	14	04/17/16 11:07:03 PM

2.现场验收

现场验收没有问题。

3.实验总结与体会

（1）本次实验的模块非常多，但是只要一步一步地仿真就不会出现大的问题。一开始在仿真的时候没有仿真译码器模块，因为认为没有必要在软件上测试扫描数码管显示，后来烧写之后发现数码管的显示整体向右移了一位，然后再回过头来在译码模块中找 BUG。

（2）本次实验的综合仿真的时间很长，因为需要记录 1s 的计数结果，这也给调试带来了麻烦。

（3）通过本次实验最主要的收获是学会了框图式地分解模块，将一个问题合理地分解成若干个小的模块实现，可以大大地减少问题的复杂性，并且有助于调试。