实验三：频率计

漆耘含

无 63

2016011058

一. 实验目的

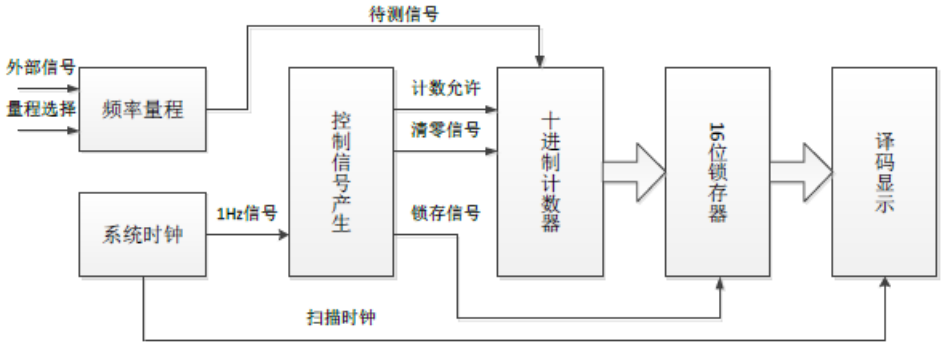
掌握频率计的原理和设计方法

二. 实验原理

1. 原理分析

频率计用于对一个未知频率的周期信号进行频率测量，在 1s 内对信号周期进行计数，即为此周期信号的频率。

频率计内部实现框图如下所示，其内部包括频率量程处理模块（10 分频）、时钟频率产生模块、控制信号产生模块、十进制计数模块、锁存器模块、译码显示模块等。



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，将计数结果锁存并输出到数码管中显示。

2. 模块功能分析

a. 频率量程模块：

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。

b. 系统时钟模块：

根据外部输入的参考时钟（50MHz）产生标准 1Hz 的控制信号。同时也生成扫描时钟。

c. 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。

d. **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 周期内对其进行技术操作。

e. **锁存器模块：**

在计数完成之后对技术结果进行锁存，保存上一测量周期的测量结果

f. **译码显示模块：**

将测量结果输出到 LED 数码管显示。

三. 关键代码及文件清单

1. 文件清单及层级结构

frequency_check.v //top 文件

count_clk.v //生成 1Hz 信号模块

signalinput.v //信号输入模块

signal_process.v //信号处理模块（10 分频）

control_s.v //控制信号生成模块

count.v //计数模块

Lock.v //锁存模块

display.v //显示模块

2. 关键代码及分析

a. frequency_check.v

代码：

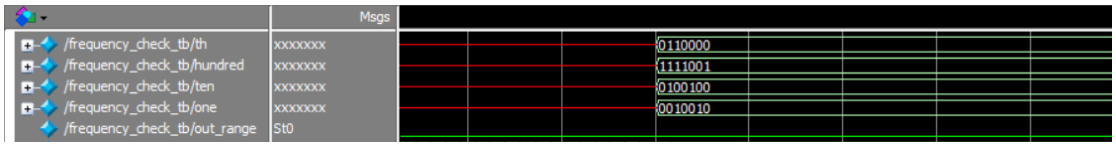
```
module frequency_check(sys_clk,reset,testmode,range,th,hundred,ten,one,out_range);
  input sys_clk,reset,range;
  input [1:0] testmode;
  output [6:0]th;
  output [6:0]hundred;
  output [6:0]ten;
  output [6:0]one;
  output out_range;
  wire [6:0] th;
  wire [6:0] hundred;
  wire [6:0] ten;
  wire [6:0] one;
  wire [3:0] th_d;
  wire [3:0] hundred_d;
  wire [3:0] ten_d;
  wire [3:0] one_d;
  wire [3:0] th_c;
  wire [3:0] hu_c;
  wire [3:0] ten_c;
  wire [3:0] one_c;
  wire w_enable,clear,save;
  wire c_clk;
  wire sigin,o_sigin,r_s;
  wire out_range;

  assign out_range=range;
  count_clk c_c(.sys_clk(sys_clk),.reset(reset),.c_clk(c_clk));
  signalinput s_i(.sysclk(sys_clk),.resetb(reset),.testmode(testmode)
              ,.sigin(sigin));
  signal_process s(.sigin(sigin),.range(range),.o_sigin(o_sigin));
  control_s c_s(.c_clk(c_clk),.reset(reset),.w_enable(w_enable)
                ,.clear(clear),.save(save));
  count c(.w_enable(w_enable),.clear(clear),.sigin(o_sigin),.th_d(th_d)
          ,.hundred_d(hundred_d),.ten_d(ten_d),.one_d(one_d));
  Lock L(.c_clk(c_clk),.save(save),.th_d(th_d),.hundred_d(hundred_d)
          ,.ten_d(ten_d),.one_d(one_d),.th_c(th_c),.hu_c(hu_c)
          ,.ten_c(ten_c),.one_c(one_c));
  display d(.c_clk(sys_clk),.th_c(th_c),.hu_c(hu_c),.ten_c(ten_c)
          ,.one_c(one_c),.th(th),.hundred(hundred),.ten(ten),.one(one));

endmodule
```

代码分析：

这是频率计的 top 设计，sys_clk 是输入的系统时钟（50MHz）,reset 是复位输入，testmode 是输入的两比特的选择信号，range 是选择量程的输入，th 是输出的千位，hundred 是输出的百位，ten 是输出的十位，one 是输出的个位，out_range 是输出的量程选择。

Count_clk 模块生成 1Hz 的时钟信号，signalinput 模块生成测试的信号，signal_process 模块是对待测信号进行处理（是否进行 10 分频），control_s 模块生成的是控制信号，count 模块是计数模块，Lock 模块是锁存模块，display 模块是译码显示模块。

仿真分析：



这是第 1s 和第 2s 的情况，在第 2s 的时候输出低 1s 的计数结果。

b. count_clk

代码：

```
1
2   module count_clk(sys_clk,reset,c_clk);
3     input sys_clk,reset;
4     output c_clk;
5     reg c_clk=1'b0;
6
7     parameter CNT=26'd50000000;
8
9     reg [25:0] cnt=26'd0;
10
11    always @(posedge sys_clk,negedge reset)
12    begin
13      if(~reset)
14        begin
15          cnt<=26'd0;
16          c_clk<=1'b0;
17        end
18      else
19        begin
20          if(cnt>=CNT)
21            cnt<=26'd0;
22          else
23            cnt<=cnt+26'd2;
24
25          if(cnt==26'd0)
26            c_clk=~c_clk;
27        end
28    end
29
30  endmodule
31
```

代码分析：

此模块输入的是 sys_clk，reset，输出的是 c_clk。设置了一个常数 CNT=26’d50000000。当复位信号 reset 来，或者 cnt=CNT 时，计数 cnt 归零，否则，cnt+=2，当 cnt 归零的时候，c_clk 翻转一次，这样输出的就是 1Hz 的信号。

c. signalinput

代码：

```
module signalinput(sysclk,resetb,testmode,sigin);
  input sysclk,resetb;
  input [1:0] testmode;
  output sigin;
  reg sigin;
  reg [20:0] state;
  reg [20:0] divide;

  initial
  begin
    sigin<=1'b0;
    state<=21'd0;
  end
  always @(*)
  begin
    case(testmode)
      2'b00:divide=21'd16000;       //3125Hz
      2'b01:divide=21'd8000;        //6250Hz
      2'b10:divide=21'd1000000;     //50Hz
      2'b11:divide=21'd4000;        //12500Hz
    endcase
  end
  always @(posedge sysclk or negedge resetb)
  begin
    if(~resetb)
      begin
        sigin<=1'b0;
        state<=21'd0;
      end
    else
      begin

        if(state==divide-21'd2)
          state<=21'd0;
        else
          state<=state+21'd2;

        sigin<=(state==21'd0)?~sigin:sigin;
      end
  end
endmodule
```

代码分析：

输入的是 sys_clk，resetb，testmode，输出的是 sigin（待测信号），在刚开始的时候给 sigin 和中间变量 state 赋初值，通过一个 case 语句来选择输出的是哪一种信号，并把对应的值赋给 divide，之后如果复位信号 resetb 来，或者达到 divide-2，则 state 归零，否则 state+=2，当 state=0 的时候，sigin 翻转一次，这样输出的就是想要的待测信号。

d. signal_process

代码：

```
module signal_process(sigin,range,o_sigin);
  input sigin,range;
  output o_sigin;
  reg [2:0] c;
  reg o_sigin1;
  wire sigin1;

  assign o_sigin=range?o_sigin1:sigin;

  initial
  begin
   c<=3'b000;
   o_sigin1<=1'b1;
 end

  always @(posedge sigin )
  begin
    if(range)
      begin
            if(c<3'b100)
              c<=c+3'b001;
            else
              begin
                c<=3'b000;
                o_sigin1=~o_sigin1;
              end
        end
  end
endmodule

```

代码分析：

输入是待测信号 sigin，range，输出的是处理之后的待测信号 o_sigin。从 signalinput 出来的信号都必须经过 signal_process 模块，是否进行 10 分频的依据是 range 为 1 还是为 0.如果 range=0，则直接通过，如果 range=1，则进行十分频。

e. control_s

代码：

```
module control_s(c_clk,reset,w_enable,clear,save);
  input c_clk,reset;
  output w_enable,clear,save;
  reg w_enable1;
  reg clear1;
  reg save1;


  assign w_enable=w_enable1;
  assign clear=clear1;
  assign save=save1;

  initial
  begin
   w_enable1<=1'b1;
   clear1<=1'b0;
   save1<=1'b0;
  end


  always@(posedge c_clk,negedge reset)
  begin
    if(~reset)
      begin
        clear1<=1'b0;
        save1<=1'b0;
      end
    else
      begin
       save1<=~save1;
       clear1<=~clear1;
     end
  end

endmodule
```

代码分析：

输入是 c_clk（1Hz 的信号），reset，w_enable，clear（清零信号），save（锁存信号）。当复位信号来的时候，clear 和 save 清零，否则，在 1Hz 信号触发条件下，save 和 clear 翻转一次。

f. count

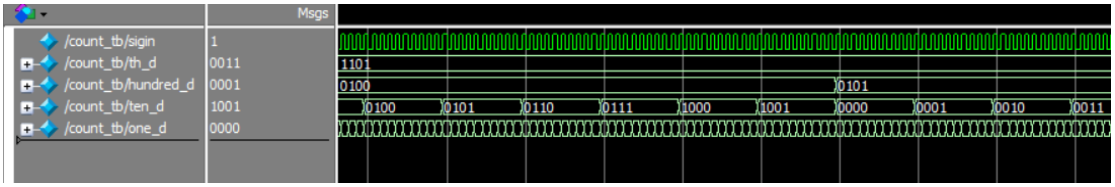
代码：

```
module count(w_enable,clear,sigin,th_d,hundred_d,ten_d,one_d);
  input w_enable,clear,sigin;
  output [3:0]th_d;
  output [3:0]hundred_d;
  output [3:0]ten_d;
  output [3:0]one_d;
  reg [3:0]th_d1;
  reg [3:0]hundred_d1;
  reg [3:0]ten_d1;
  reg [3:0]one_d1;
  assign th_d=th_d1;
  assign hundred_d=hundred_d1;
  assign ten_d=ten_d1;
  assign one_d=one_d1;
  initial
  begin
    th_d1<=4'b0;
    hundred_d1<=4'b0;
    ten_d1<=4'b0;
    one_d1<=4'b0;
  end

  always@(posedge sigin)
  begin
    if(w_enable) begin
        if(clear) begin
            th_d1<=4'b0;
            hundred_d1<=4'b0;
            ten_d1<=4'b0;
            one_d1<=4'b0;
          end
        else begin
            if(one_d1!=4'b1001)
              one_d1<=one_d1+4'b0001;
            else if(ten_d1!=4'b1001) begin
                one_d1<=4'b0;
                ten_d1<=ten_d1+4'b0001;
              end
            else if(hundred_d1!=4'b1001) begin
                one_d1<=4'b0;
                ten_d1<=4'b0;
                hundred_d1<=hundred_d1+4'b0001;
              end
            else begin
                one_d1<=4'b0;
                ten_d1<=4'b0;
                hundred_d1<=4'b0;
                th_d1<=th_d1+4'b0001;
              end
            end
          end
        end
endmodule
```

代码分析：

输入的是计数使能信号 w_enable，clear，sigin，输出的是 th_d，hundred_d，ten_d，one_d。计数增加的方法是低位都计数到 9，下一次计数的时候，下一位+1，这样就能进行 10 进制的计数。

仿真分析：



由输出波形可知，这是一个计数器

Count_tb.v 在附件中

g. Lock

代码：

```
module Lock(c_clk,save,th_d,hundred_d,ten_d,one_d,th_c,hu_c,ten_c,one_c);
  input c_clk,save;
  input [3:0]th_d;
  input [3:0]hundred_d;
  input [3:0]ten_d;
  input [3:0]one_d;
  output [3:0]th_c;
  output [3:0]hu_c;
  output [3:0]ten_c;
  output [3:0]one_c;

  reg [3:0]th_cb;
  reg [3:0]hu_cb;
  reg [3:0]ten_cb;
  reg [3:0]one_cb;

  assign th_c=th_cb;
  assign hu_c=hu_cb;
  assign ten_c=ten_cb;
  assign one_c=one_cb;

  always @(posedge c_clk)
  begin
    if(~save)
      begin
        th_cb<=th_d;
        hu_cb<=hundred_d;
        ten_cb<=ten_d;
        one_cb<=one_d;
      end
  end

endmodule
```

代码分析：

从 count 输出的计数结果，在 save 信号的控制下，进行锁存操作。如果 save=1，则锁存，如果 save=0，则透明输出。

h. Display

代码：

```
module display(c_clk,th_c,hu_c,ten_c,one_c,th,hundred,ten,one);
  input c_clk;
  input [3:0]th_c;
  input [3:0]hu_c;
  input [3:0]ten_c;
  input [3:0]one_c;
  output [6:0]th;
  output [6:0]hundred;
  output [6:0]ten;
  output [6:0]one;

  reg [6:0]th;
  reg [6:0]hundred;
  reg [6:0]ten;
  reg [6:0]one;

  wire s_c;

  scan s(.s_clk(c_clk),.s_c(s_c));

  always @(posedge s_c)
  begin

    case(th_c)
      4'b0000:th=7'b1000000;
      4'b0001:th=7'b1111001;
      4'b0010:th=7'b0100100;
      4'b0011:th=7'b0110000;
      4'b0100:th=7'b0011001;
      4'b0101:th=7'b0010010;
      4'b0110:th=7'b0000010;
      4'b0111:th=7'b1111000;
      4'b1000:th=7'b0000000;
      4'b1001:th=7'b0010000;
    endcase

    case(hu_c)
      4'b0000:hundred=7'b1000000;
      4'b0001:hundred=7'b1111001;
      4'b0010:hundred=7'b0100100;
      4'b0011:hundred=7'b0110000;
      4'b0100:hundred=7'b0011001;
      4'b0101:hundred=7'b0010010;
      4'b0110:hundred=7'b0000010;
      4'b0111:hundred=7'b1111000;
      4'b1000:hundred=7'b0000000;
      4'b1001:hundred=7'b0010000;
    endcase
```

```
51      case(ten_c)
52           4'b0000:ten=7'b1000000;
53           4'b0001:ten=7'b1111001;
54           4'b0010:ten=7'b0100100;
55           4'b0011:ten=7'b0110000;
56           4'b0100:ten=7'b0011001;
57           4'b0101:ten=7'b0010010;
58           4'b0110:ten=7'b0000010;
59           4'b0111:ten=7'b1111000;
60           4'b1000:ten=7'b0000000;
61           4'b1001:ten=7'b0010000;
62         endcase
63
64      case(one_c)
65        4'b0000:one=7'b1000000;
66        4'b0001:one=7'b1111001;
67        4'b0010:one=7'b0100100;
68        4'b0011:one=7'b0110000;
69        4'b0100:one=7'b0011001;
70        4'b0101:one=7'b0010010;
71        4'b0110:one=7'b0000010;
72        4'b0111:one=7'b1111000;
73        4'b1000:one=7'b0000000;
74        4'b1001:one=7'b0010000;
75      endcase
76    end
77  endmodule
```

```
79  module scan(s_clk,s_c);
80    input s_clk;
81    output s_c;
82
83    reg s_c=1'b0;
84
85    parameter CNT=26'd50000;
86
87    reg [25:0] cnt=26'd0;
88
89    always @(posedge s_clk)
90    begin
91        begin
92          if(cnt>=CNT)
93            cnt<=26'd0;
94          else
95            cnt<=cnt+26'd2;
96
97          if(cnt==26'd0)
98            s_c=~s_c;
99        end
100   end
101 endmodule
```

代码分析：

display 模块里面有另一个模块：scan，目的是生成显示输出的扫描信号。display 模块通过译码模块即可生成输出信号。

四. 综合情况

Flow Summary

Flow Status	Successful - Fri May 18 16:06:20 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	frequency_check
Top-level Entity Name	frequency_check
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	207 / 33,216 (< 1 %)
Total combinational functions	195 / 33,216 (< 1 %)
Dedicated logic registers	138 / 33,216 (< 1 %)
Total registers	138
Total pins	34 / 475 (7 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

TimeQuest Timing Analyzer Summary

Quartus II Version	Version 13.0.1 Build 232 06/12/2013 Service Pack 1 SJ Web Edition
Revision Name	frequency_check
Device Family	Cyclone II
Device Name	EP2C35F672C6
Timing Models	Final
Delay Model	Combined
Rise/Fall Delays	Unavailable

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	235.63 MHz	235.63 MHz	Clk_50M	

Setup Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	testmode[*]	Clk_50M	4.532	4.532	Rise	Clk_50M
1	testmode[0]	Clk_50M	4.450	4.450	Rise	Clk_50M
2	testmode[1]	Clk_50M	4.532	4.532	Rise	Clk_50M

五. 实验总结

频率计这个实验有很多的模块，在写之前要充分理解每一个模块的关系以及功能，这样在后面写代码的时候才能更有逻辑。

在写代码的时候我碰到很多问题。第一个问题是关于控制信号该如何产生的问题，因为他是 1s 为周期的，输出的是上一周期的计数结果，那么在每 1s 末的时候，要把计数结果锁存，同时生成清零信号把计数器清零，这个过程应该是先锁存，然后清零，我的疑惑就在于如何先锁存再清零，后来发现直接同时翻转是没有问题的，同时锁存模块的触发条件应该也是 1s，之前我错误的写成了系统信号，所以一直没有锁存住。

第二个问题是在生成待测信号的事后，我在仿真的时候，每一个周期都多了 40ns，也就是说在生成信号的时候，计数多了 2，因此我把多余的减去，结果就对了。

这是第一次写这么多模块，从这个过程中掌握了如何分模块来调试系统，并且更加了解频率计的工作原理。