数字频率计设计报告

——数电实验设计报告

张家豪

院系：数据科学与计算机学院

专业：计算机类

班级：16 级计科 7、8 班

学生姓名：张家豪　16337303
肖　遥　16337258
徐正金　16337267

目录

一、作品简介

1、作品概述：

数字频率计是采用数字电路制作的能实现对周期性变化信号频率测量的仪器。我们在老师课堂上讲解的原理上增加了一些新的功能，包括可改变刷新频率，指数可调的科学计数法显示。

2、灵感来源：

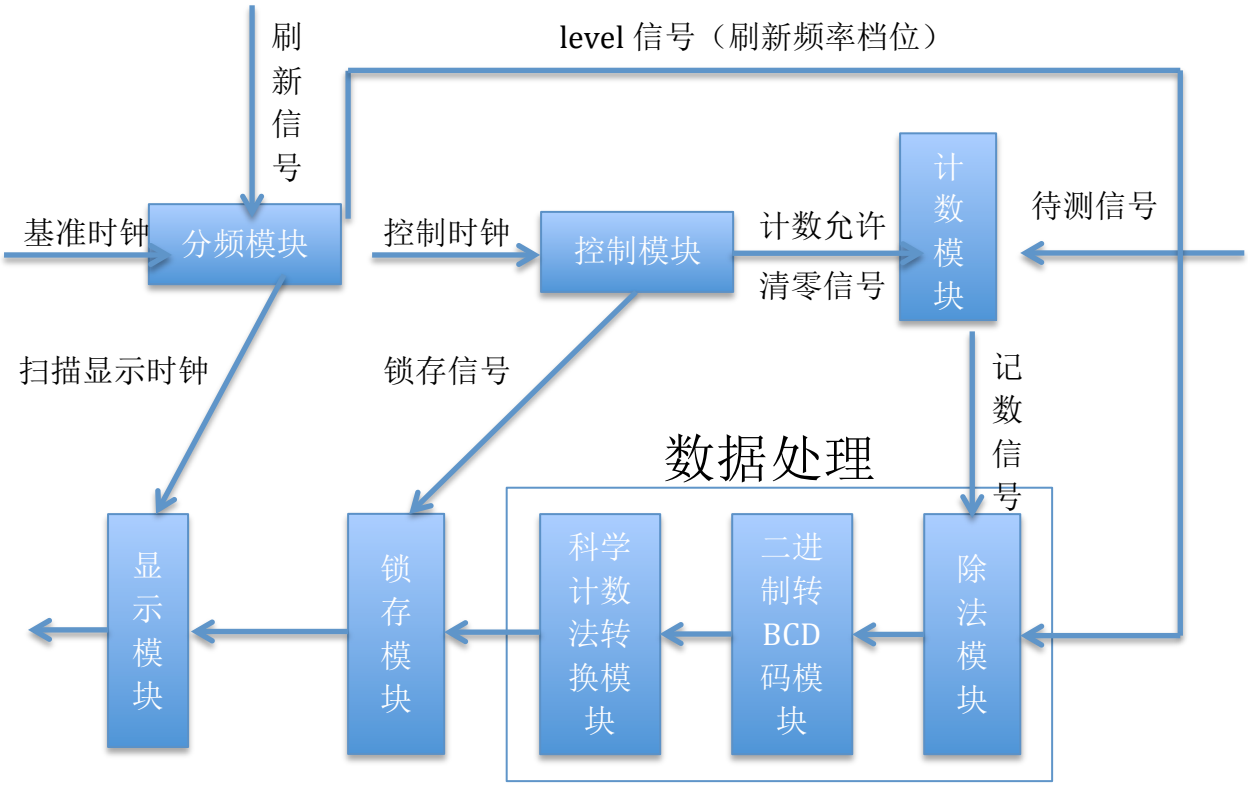
参考题目中对频率计的要求是若量程过大或过小，则在下周期的测量中相应减小或增大量程。我们改进的出发点是将下周期改变量程变为量程及时自适应，即在当前周期就选择最合适的量程。另外考虑到需要有一个数位显示当前量程，这样做出来就与科学计数法类似。因此我们想，不如直接用科学计数法的形式显示频率。

然而，经过初步设想，我们发现只是改成科学计数法似乎比原题更简单，于是考虑再增加一些功能。我们想到了将刷新频率（测量周期）变为可手动调整，以及科学计数法的指数位变为可自动可手动。

二、设计报告

1、设计模块简介

（1）模块组成



（2）模块接口

顶层模块：

输入：CLK_in, CLK_base, RF_up, FR_down, EXP_auto, EXP_up, EXP_down
输出：[6:0] a_to_g, [3:0] EN, DP

分频模块：

输入：CLK_base, RF_up, RF_down
输出：CLK_scan, CLK_ctrl , [1:0] level

控制模块：

输入：CLK_ctrl
输出：CTEN, CLR_counter, LOAD_latch

计数模块：

输入：CLK_in, CTEN, CLR
输出：[20:0] CNT

除法模块：

输入：[20:0]CNT, [1:0] level
输出：[19:0] F

二进制码转 BCD 码模块：

输入：[19:0] BIN
输出：[23:0] BCD

科学计数法转换模块：

输入：[23:0] BCD_raw, EXP_auto， EXP_up, EXP_down
输出：[15:0] BCD_sci

锁存模块：

输入：[15:0] BCD_in, LOAD
输出：[15:0] BCD_out

显示模块：

输入：[15:0] BCD, CLK_scan
输出：[6:0] a_to_g, [3:0] EN, DP

2、设计模块简单详解

（1）顶层模块：

输入有基准时钟信号（CLK_base），待测信号（CLK_in），刷新频率调整信号（RF_up, RF_down），科学计数法指数调整信号（EXP_auto, EXP_up, EXP_down）。基准时钟信号通过调用分频模块得到需要的扫描显示信号和控制时钟信号，待测信号通过计数模块计数，计数完后通过锁存模块储存记数，再通过除法模块，二进制转 BCD 码转换模块和科学计数法转换模块得到可以显示的 4 位十进制码，前三位为有效数字，后一位为指数值，再通过调用显示模块在数码管上显示。

（2）分频模块：

输入基准时钟、刷新频率调整信号，输出控制时钟和扫描显示时钟。4 档刷新频率的计数周期分别为 0.25s，0.5s，1s 和 2s，扫描显示频率为 100Hz，用于循环点亮 4 位 7 段数码管。附加输出 1 个 2 位的信号（level）表明刷新频率档位，用于后面的数据处理。

（3）控制模块：

根据分频信号的输出中的控制时钟，产生计数器的计数允许信号、清零信号和锁存器的锁存信号。

（4）计数模块：

当计数允许信号为低电平时，计数器工作，对输入待测信号的脉冲计数，清零信号下降沿触发。该模块输出的是 21 位 2 进制信号，最大值不超过 2000000_{10}（若达到 2000000 则不再计数，因为一定超出量程）。

（4）除法模块：

我们将计数器得到的计数除以对应测量周期的数值，得到单位时间 1s 内输入信号频率的 2 进制码表示。实现上由于测量周期设定为 2^1，2^0，2^{-1} 和 2^{-2}，除法（或乘法）用左右移代替。若超量程则将输出设为所有二进制位全 1。

（4）二进制码转化 BCD 码模块：

二进制转 BCD 码参照了数电实验课本附录中相应模块的算法，通过不断左移，每四位判断是否大于 4，满足则加 3。该模块输入 20 位二进制码，转换成 6 位 BCD 码输出。

（5）科学计数法转换模块：

当指数自动信号（EXP_auto）为高电平时，该模块会自动将 6 位 BCD 码输入转化为最合适的科学计数法形式（按最高位确定指数）。当 EXP_auot 为低电平时，该模块通过指数控制信号（EXP_up，EXP_down）控制指数的增大和减小。因为将小数点固定在 4 位 7 段数码管的最高位，所以当数值无法用指定指数表示时，我们将前三位设为 FFF。

（6）锁存模块；

当输入信号 LOAD 下降沿到达时，科学计数法转换模块输出的 4 位 BCD 码会被锁存起来。输出作为显示模块的数据输入。

（7）显示模块：

输入 4 位 BCD 码和扫描显示时钟信号，每次只选通一个位，显示该位相应数字，通过快速循环选通实现 4 位“同时”点亮，小数点（DP）只有在显示最高位时选通，其余不选通。4 位 BCD 码中，前 3 位为科学记数法前面的实数表示部分，最后 1 位为指数的值，如 1.233 代表的是 1.23*10^ 3。F 对应的输出为负号 “-”，用于提示超出量程。

三、样例测试

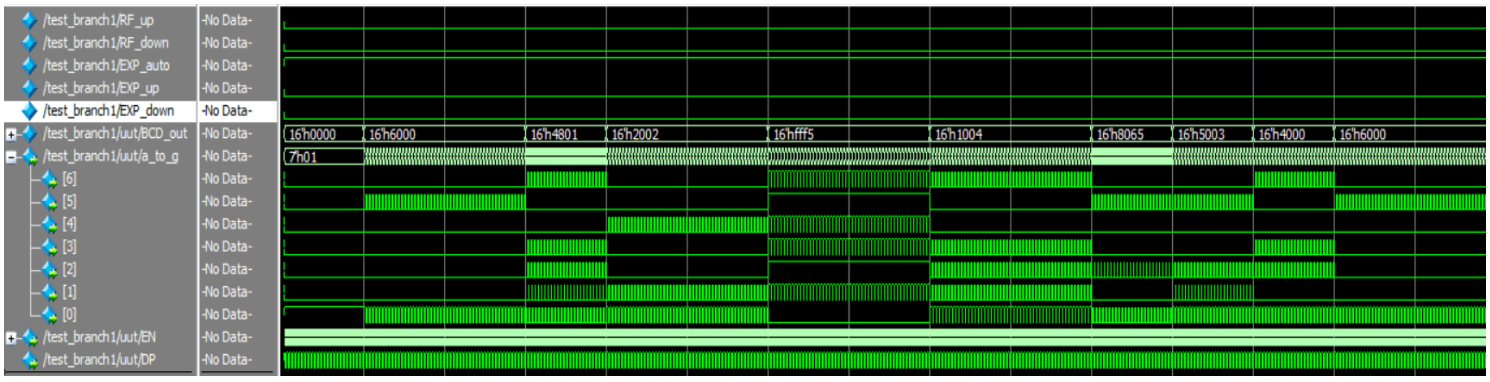
我们一共写了 4 个测试模块，其中第一个测试基本的测频和显示功能；第二个测试手动调整刷新频率的功能；第三个测试手动调整指数的功能；第四个既有自动指数也有手动调整指数，测试两种模式混合时的工作情况。

经测试，各模块都能正常工作，下面我们展示其中第一个测试文件。（其他文件见附录）

```
11  parameter base_period = 10;          // 基准时钟周期：10ns（频率100MHz）
12  parameter period_1s = 1000000000;    // 一秒的周期：10^9ns
13
14  reg [31:0] cnt;          // 基准时钟触发的计数器，用于产生待测信号
15  reg [31:0] period;       // 待测信号的周期（以ns为单位）
16
17  top uut(CLK_in, CLK_base, RF_up, RF_down, EXP_auto, EXP_up, EXP_down, a_to_g, EN, DP);
18
19  initial begin
20      CLK_base = 0;
21      CLK_in = 0;
22      RF_up = 0;
23      RF_down = 0;
24      EXP_auto = 1;    // 指数自动
25      EXP_up = 0;
26      EXP_down = 0;
27      cnt = 0;
28      period = period_1s / 10;
29      #100;
30  end
31
32  always begin
33      #(base_period / 2) CLK_base = ~CLK_base;     // 基准时钟
34  end
35
36  always @(posedge CLK_base) begin
37      cnt = cnt + 1;
38      if (cnt == period / base_period / 2) begin  // cnt*base_period = period/2，即到达半周期
39          CLK_in = ~CLK_in;    // 待测信号
40          cnt = 0;
41      end
42  end
43
44  always @(period) begin
45      cnt = 0;         // 当待测信号周期变化时，计数器应重新开始计数
46  end
```

```
48  always begin
49      period = period_1s / 7;//待测信号为7Hz，后面类似
50      #(period_1s * 2);//维持2s
51
52      // 由于这1s中只有后0.5s计数，故计数期间平均频率为(20*1+60*2)/3=46(Hz)
53      period = period_1s / 10;
54      #(period_1s / 3 + 1);
55      period = period_1s / 20;
56      #(period_1s / 3);
57      period = period_1s / 60;
58      #(period_1s / 3);
59
60      period = period_1s / 200;
61      #(period_1s * 2);
62      period = period_1s / 2000000;
63      #(period_1s * 2);
64      period = period_1s / 10000;
65      #(period_1s * 2);
66      period = period_1s / 800000;     // 实际频率：806451Hz
67      #(period_1s * 1);
68      period = period_1s / 5000;
69      #(period_1s * 1);
70      period = period_1s / 5;
71      #(period_1s * 1);
72
73  end
```

如上所示，我们先后测了 8 组信号，每组维持 1s 或 2s，对应的频率分别为 period_1s 所除的数值，即为 7Hz、46Hz、200Hz、2000000Hz（超出量程）、$1*10^4$Hz、$8*10^5$Hz（由于通过频率计算待测信号的周期时使用了整除，产生了一定误差，所以实际生成的信号频率为 806451Hz）、$5*10^3$Hz 和 5Hz。



经过 ModelSim 仿真，实际得出来的结果分别为 $6.00*10^0$Hz、$4.80*10^1$Hz、$2.00*10^2$Hz、FFF（超出量程）、$1.00*10^4$Hz、$8.06*10^5$Hz、$5.00*10^3$Hz、$4.00*10^0$Hz。

可以看出，实验结果基本符合预期，但在高频率时测量精度高，在低频率时测量精度不足，这是由于实验本身的原理导致的。

四、设计反思

1、我们组采用的方法是脉冲数定时测频法（M 法）：通过记录在确定时间内待测信号的脉冲个数来测量频率。由于需要提高频率计的稳定性，我们设计了刷新频率这个功能，就是通过在不同的时间测量脉冲个数，来提高稳定性，这也一定程度上提高了精度。

2、我们的设计目前还不能测量正弦波等其他信号的频率，若要测量，还需要有模数转换器（ADC）。

3、我们的测量方法实现简单，但是因为测量精度与待测信号频率和门控时间有关，当待测信号频率较低时，误差较大。我们组想了一个解决办法：可以设计一个倍频模块，将待测信号频率放大若干倍，以提高测量精度。但控制电路较为复杂，所以我们还没有实现。

4、除此之外，我们还了解到频率计做法的其他思路，即脉冲周期测频法，在待测信号的一个周期里记录标准信号频率变化次数。此法低频检测时精度高，但高频检测时误差较大，所以提高精度的办法跟 M 法相反，做法为将待测信号进行分频处理。

5、我们还了解到了一种精度高且与待测信号无关的测频方法——多周期同步测频法：是由闸门时间与同步门控时间共同控制结束起计数的一种测量方法，因为闸门时间与被测信号同步，消除了对被测信号计数产生的±1 误差，测量精度大大提高，且测量精度与待测信号无关，实现了在整个测量频段等精度测量。

五、 成员感想：

张家豪：

平时有没有好好学，做做项目就知道，尝试做了频率计的一些子模块，发现自己平时果然是没学好，在自己打码的过程中慢慢掌握了一些语法的规则，分清了自己之前不求甚解的问题：reg 和 wire 的区别。这次抱了大腿，尽量做了自己能做的事，感觉还是有收获的，对各种频率计的认识和理解也更深了一步。

频率计的设计原理并不复杂，可是在实现的过程中遇到的问题和代码之繁琐，测试文件之诡异都让我感到数电这个科目和 vivado 实操都是玄学一样的存在。这是我第一次尝试合作的数电项目，组员之间的分工合作显得尤其重要，虽然代

码水平还有待改进，但是做了尝试做了一些模块，一开始做的模块有的能用有的后来不合适，但是在打代码的过程中逐渐掌握了频率计的原理，本来上网看的晕晕的，但是打一打代码，感觉进步得很快。

做之前我和肖遥一样都以为测试文件不会太难，但是肖遥真的是很有追求，为了调试各个功能能否正常运作，写了 4 个测试文件，而且分频分出来的信号不是 2 分频、4 分频之类的简单分频，而是特定的分频。说实话，当要我写测试文件的时候，我都不知道从何写起，所以测试文件都是肖遥写的。看到它写的测试代码，心里不断惊呼：哇，还有这种操作啊。

到目前为止，我可以说我对频率计有了一定深度的了解。为了加深这次数电实验的印象，我还去查了各种各样的频率计，了解各种测量方法的优缺点，比如说我们这个设计的缺点便是待测信号频率低于一定数值，大概是低于 100Hz 时就会出现±1 的误差。解决方法是通过倍频增加频率，再将测频计测出的频率除以增加的频率。

我对自己的定位还是比较明确的，我属于实力一般的偶像派，所以就打了一些简单的模块，便打便设计实验报告，我觉得我最有优势的便是展示环节，以我的口才和解说能力相信能通过我的心路历程将这个问题向大家解释得很清楚。为了讲解的趣味和实用性同在，也认真得组织了语言。

感觉在这次项目中，体会到了团队合作的重要性，感觉自己打了一个完美的辅助，堪称 MVP（most valued player）辅助。感谢队友，感觉学到了很多。

肖遥：

一开始提出对频率计的几个改进时，我认为实质上没有太多的技术含量，不觉得实现起来会增加很大的难度。但真正着手写一些模块的代码时，才发现很多不太好处理的细节问题。例如科学计数法转换模块中指数与有效数位在两种模式下的变化先后和触发时机的问题，经过多番思考后终于理清其中的逻辑：当输入的 BCD 码或指数控制信号变化时触发指数的变化，而有效数位的变化则由 BCD 码变化和指数变化触发，根据指数确定选取 BCD 码的哪三位。

另外，由于老师给的频率计设计样例中，测量周期为 1s，计数器的结果即为频率，不需要除以周期，故一开始我没有考虑到需要有一个除法模块，计数器直接按照十进制计数。但因为我提出增加的可调节测量周期的功能，使得除法模块变成必须的。曾试想通过将待测信号进行分频等较简单的方法代替，但都不能达到要求，故只能将计数器换为按二进制计数，然后多加一个除法模块进行处理。在写除法模块时，考虑到尽量避免除法的问题，想到了把测量周期定为 2 的整数次幂，这样乘除法可用左右移代替。同时由于改用二进制计数，还需要一个二进制转 BCD 码的转换模块。这个模块我开始并没有想到好的实现方式，于是翻了数电实验课本，发现正好有二进制转 BCD 码的模块，但不能理解其原理。上网搜索后，才知道这里用到一个巧妙的算法，能避免使用除法。

写好模块以后，又一个没有预想到的困难是仿真。仿真文件本身不太容易写，测试中出现的一些问题开始以为是频率计某些模块的问题，最后发现是仿真文件

本身的问题，例如在改变待测信号周期后没有将计数器清零、计算得到的待测信号周期可能因为整除去余产生偏差。另一方面，在我的电脑上用 ModelSim 完整仿真一个测试模块，至少需要十几分钟的时间，使调试变得困难。原本打算对各个模块进行独立测试，通过后再进行综合测试，但考虑到所剩时间不多，每个模块单独写一个测试文件并测试比较耗时，而且很多模块复杂度不高、正确性能保证，于是选择跳过独立测试直接进行综合测试。但从最后调试的感受而言，可能先对一些复杂模块做独立测试会提高效率，因为独立测试时，数据处理类的模块如 BCD 转二进制、科学计数法转换模块不需要仿真时钟信号，因而仿真可以很快出结果。

这次项目设计让我体会到一个项目就算在原理上或总体设计上比较简单，实际实现时还是会遇到很多意想不到的困难，需要想办法解决，有时需要借助一些资料。此外，一个稍微大型一点的项目光靠一个人实现是不现实的。我们在完成顶层设计，安排好每个模块的功能和接口后，分工进行各个模块的代码编写。在此过程中我体会到，采用自顶向下的设计方法有利于团队的分工协作，因为将不同模块的职责与模块间的关系后，各个模块是相互独立的，大家分工后只需关心自己模块内部的实现，而基本不需了解其他模块的实现。总之，我从此次项目实践中有了很多平时难有的收获，以后也需要更多在实战中提升自己。

徐正金：

做项目才能真正运用到自己的所学知识，也看到了自己的许多不足，这次能和两位同学合作感觉很幸运，这次抱了大腿。在打码过程中遇到问题，在解决问题的过程中收获了不少，感觉收获不少。

六、 参考文献

1、https://wenku.baidu.com/view/ef74f1727fd5360cba1adbdc.html

2、http://www.cnblogs.com/guojingdeyuan/p/6718828.html

3、http://blog.csdn.net/jishuzhain/article/details/53148631

七、 附录 A（代码）

顶层模块：

```
module top(
    input CLK_in,
    input CLK_base,
    input RF_up,
    input RF_down,
    input EXP_auto,
    input EXP_up,
    input EXP_down,
    output [6:0] a_to_g,
    output [3:0] EN,
    output DP
);

wire CLK_scan, CLK_ctrl;
wire CTEN, CLR_counter, LOAD_latch;
wire [1:0] level;
wire [20:0] CNT;
wire [19:0] F;
wire [23:0] BCD_raw;
wire [15:0] BCD_sci;
wire [15:0] BCD_out;

ClkDiv clock_divider(
    .CLK_base(CLK_base),
    .RF_up(RF_up),
    .RF_down(RF_down),
    .CLK_scan(CLK_scan),
    .CLK_ctrl(CLK_ctrl),
    .level(level)
);

Control control(
    .CLK_ctrl(CLK_ctrl),
    .CTEN(CTEN),
    .CLR_counter(CLR_counter),
    .LOAD_latch(LOAD_latch)
);

Counter counter(
    .CLK_in(CLK_in),
    .CTEN(CTEN),
```

```
    .CLR(CLR_counter),
    .CNT(CNT)
);

Devider devider(
    .CNT(CNT),
    .level(level),
    .F(F)
);

Bin20ToBCD bin_to_BCD(
    .BIN(F),
    .BCD(BCD_raw)
);

SciConverter sci_notation_converter(
    .BCD_raw(BCD_raw),
    .EXP_auto(EXP_auto),
    .EXP_up(EXP_up),
    .EXP_down(EXP_down),
    .BCD_sci(BCD_sci)
);

Latch latch(
    .BCD_in(BCD_sci),
    .LOAD(LOAD_latch),
    .BCD_out(BCD_out)
);

Display display(
    .BCD(BCD_out),
    .CLK_scan(CLK_scan),
    .a_to_g(a_to_g),
    .EN(EN),
    .DP(DP)
);

endmodule // top
```

分频模块：

```
module ClkDiv(CLK_base, RF_up, RF_down, CLK_scan, CLK_ctrl, level);
input CLK_base, RF_up, RF_down;
output CLK_scan, CLK_ctrl;
output reg [1:0] level;

parameter period_1ms = 100000;

reg [26:0] cnt_scan;
parameter period_scan = 10 * period_1ms;

reg [31:0] cnt_ctrl;
reg [31:0] period_ctrl;

initial begin
    level = 2;
    cnt_ctrl = 0;
    cnt_scan = 0;
    period_ctrl = 500 * period_1ms;
end

always @(posedge RF_down or posedge RF_up) begin
    if (RF_up)
        level = (level == 2'b11) ? 2'b11 : level+1;
    else
        if (RF_down)
        level = (level == 2'b00) ? 2'b00 : level-1;

    case (level)
        2'b00: period_ctrl = 2000 * period_1ms;
        2'b01: period_ctrl = 1000 * period_1ms;
        2'b10: period_ctrl = 500 * period_1ms;
        2'b11: period_ctrl = 250 * period_1ms;
        default: period_ctrl = 500 * period_1ms;
    endcase
end

always @(posedge CLK_base) begin
    cnt_scan <= (cnt_scan < period_scan) ? cnt_scan+1 : 1;
    cnt_ctrl <= (cnt_ctrl < period_ctrl) ? cnt_ctrl+1 : 1;
```

```
end

assign CLK_scan = cnt_scan <= period_scan / 2;
assign CLK_ctrl = cnt_ctrl <= period_ctrl / 2;

endmodule // ClkDiv
```

控制模块：

```
module Control(CLK_ctrl, CTEN, CLR_counter, LOAD_latch);
input CLK_ctrl;
output reg CTEN;
output LOAD_latch, CLR_counter;

initial begin
    CTEN = 0;
end

always @(posedge CLK_ctrl) begin
    CTEN <= ~CTEN;
end

assign LOAD_latch = ~CTEN;
assign CLR_counter = ~CTEN || CLK_ctrl;

endmodule // Control
```

计数模块：

```
module Counter(CLK_in, CTEN, CLR, CNT);
input CLK_in, CTEN, CLR;
output reg [20:0] CNT;

initial begin
    CNT = 0;
end

always @(negedge CLR or posedge CLK_in) begin
```

```
    if (~CLR)
        CNT = 0;
    else if (~CTEN) begin
        if (CNT <= 2000000)
            CNT = CNT + 1;
    end
end

endmodule // Counter
```

除法模块：

```
module Devider(CNT, level, F);
input [20:0] CNT;
input [1:0] level;
output reg [19:0] F;

parameter out_of_range = 20'b11111111111111111111;

initial begin
    F = 0;
end

always @(*) begin
    case (level)
        0: begin
            if (CNT >= 2000000)
                F <= out_of_range;
            else
                F[19:0] <= CNT[20:1];
        end
        1: begin
            if (CNT >= 1000000)
                F <= out_of_range;
            else
                F <= CNT;
        end
        2: begin
            if (CNT >= 500000)
                F <= out_of_range;
```

```
            else begin
                F[19:1] <= CNT[18:0];
                F[0] <= 0;
            end
        end
        3: begin
            if (CNT >= 250000)
                F <= out_of_range;
            else begin
                F[19:2] <= CNT[17:0];
                F[1:0] <= 2'b00;
            end
        end
    endcase
end

endmodule // Devider
```

二进制码转换 BCD 码模块：

```
module Bin20ToBCD(BIN, BCD);
input [19:0] BIN;
output reg [23:0] BCD;

reg [43:0] z;

initial begin
    BCD = 0;
    z = 0;
end

always @(*) begin
    if (BIN == 20'b11111111111111111111) begin
        BCD = 24'hFFFFFF;
    end
    else begin
```

```
        z[43:20] = 0;
        z[19:0] = BIN;
        repeat (20) begin
            if (z[23:20] > 4)
                z[23:20] = z[23:20] + 3;
            if (z[27:24] > 4)
                z[27:24] = z[27:24] + 3;
            if (z[31:28] > 4)
                z[31:28] = z[31:28] + 3;
            if (z[35:32] > 4)
                z[35:32] = z[35:32] + 3;
            if (z[39:36] > 4)
                z[39:36] = z[39:36] + 3;
            if (z[43:40] > 4)
                z[43:40] = z[43:40] + 3;
            z[43:1] = z[42:0];
        end
        BCD = z[43:20];
    end
end

endmodule // Bin20ToBCD
```

科学计数法转换模块：

```
module Base(BCD_raw, exp, base);
input [23:0] BCD_raw;
input [3:0] exp;
output reg [15:4] base;

initial begin
    base = 0;
end

always @(*) begin
    case (exp)
        0: begin
            if (BCD_raw[23:4])
                base[15:4] = 12'hFFF;
            else begin
```

```
                base[15:12] = BCD_raw[3:0];
                base[11:4] = 0;
            end
        end
        1: begin
            if (BCD_raw[23:8])
                base[15:4] = 12'hFFF;
            else begin
                base[15:8] = BCD_raw[7:0];
                base[7:4] = 0;
            end
        end
        2: begin
            if (BCD_raw[23:12])
                base[15:4] = 12'hFFF;
            else
                base[15:4] = BCD_raw[11:0];
        end
        3: begin
            if (BCD_raw[23:16])
                base[15:4] = 12'hFFF;
            else
                base[15:4] = BCD_raw[15:4];
        end
        4: begin
            if (BCD_raw[23:20])
                base[15:4] = 12'hFFF;
            else
                base[15:4] = BCD_raw[19:8];
        end
        5: begin
            if (BCD_raw == 24'hFFFFFF)
                base[15:4] = 12'hFFF;
            else
                base[15:4] = BCD_raw[23:12];
        end
        default: base[15:4] = 9'hFFF;
    endcase
end

endmodule // Base
```

```
module SciConverter (BCD_raw, EXP_auto,EXP_up, EXP_down, BCD_sci);
input[23:0] BCD_raw;
input EXP_auto, EXP_up, EXP_down;
output [15:0] BCD_sci;

reg [3:0] exp;

initial begin
    exp = 0;
end

Base base_part(BCD_raw, exp, BCD_sci[15:4]);

assign BCD_sci[3:0] = exp;

always @(posedge EXP_down or posedge EXP_up) begin
    if (~EXP_auto) begin
        if(EXP_up) begin
            if(exp < 5)begin
                exp = exp + 1;
            end
        end
        else if(EXP_down) begin
            if(exp > 0)begin
                exp = exp - 1;
            end
        end
    end
end

always @ (BCD_raw or EXP_auto) begin
    if(EXP_auto) begin
        if (BCD_raw[23:20])
            exp = 5;
        else if (BCD_raw[19:16])
            exp = 4;
        else if (BCD_raw[15:12])
            exp = 3;
        else if (BCD_raw[11:8])
            exp = 2;
```

```
        else if (BCD_raw[7:4])
            exp = 1;
        else
            exp = 0;
    end
end

endmodule
```

锁存模块：

```
module Latch(BCD_in, LOAD, BCD_out);
input[15:0] BCD_in;
input LOAD;
output reg [15:0] BCD_out;

initial begin
    BCD_out = 0;
end

always @(negedge LOAD) begin
    BCD_out = BCD_in;
end

endmodule
```

显示模块：

```
module Display(BCD, CLK_scan, a_to_g, EN, DP);
input [15:0] BCD;
input CLK_scan;
output reg [3:0] EN;
output reg DP;
output reg [6:0] a_to_g;

reg[1:0] sel;
```

```
reg[3:0] digit;

initial begin
    EN = 4'b1111;
    DP = 1;
    a_to_g = 7'b1111111;
    sel = 0;
    digit = 0;
end

always @(sel) begin
    EN = 4'b1111;
    EN[sel] = 0;
end

always @(posedge CLK_scan) begin
    begin
        sel = (sel == 3)? 0 : sel+1;
        DP = (sel == 3)? 0 : 1;
    end
end

always @(*)
    case(sel)
        0:digit = BCD[3:0];
        1:digit = BCD[7:4];
        2:digit = BCD[11:8];
        3:digit = BCD[15:12];
        default: digit = 4'hF;
    endcase


always @(*)
    case(digit)
        0: a_to_g=7'b0000001;
        1: a_to_g=7'b1001111;
        2: a_to_g=7'b0010010;
        3: a_to_g=7'b0000110;
        4: a_to_g=7'b1001100;
        5: a_to_g=7'b0100100;
        6: a_to_g=7'b0100000;
        7: a_to_g=7'b0001111;
```

```
        8: a_to_g=7'b0000000;
        9: a_to_g=7'b0000100;
        // 'hA: a_to_g=7'b0001000;
        // 'hB: a_to_g=7'b1100000;
        // 'hC: a_to_g=7'b0110001;
        // 'hD: a_to_g=7'b1000010;
        // 'hE: a_to_g=7'b0110000;
        'hF: a_to_g=7'b1111110;
    default: a_to_g=7'b1111110;
    endcase
endmodule
```

八、 附录 B（更多测试样例和波形图）

1、刷新频率调整，指数自动（所需仿真时间 21+sec）

测试代码如下：

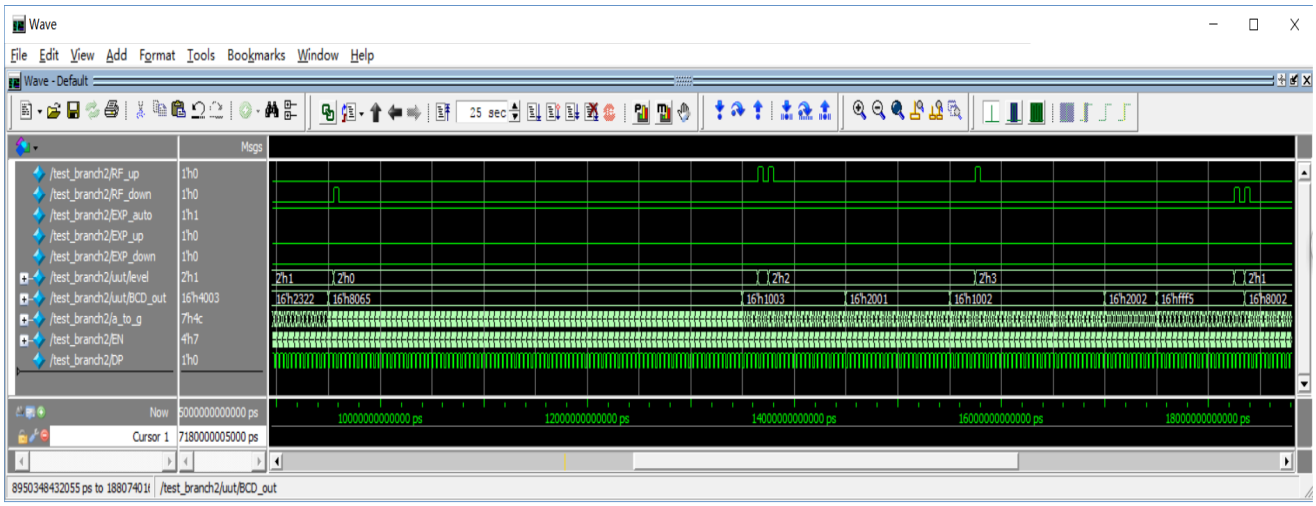
```
`timescale 1ns/1ps

module test_branch2();
reg CLK_in, CLK_base, RF_up, RF_down, EXP_auto, EXP_up, EXP_down;
wire [6:0] a_to_g;
wire [3:0] EN;
wire DP;
```

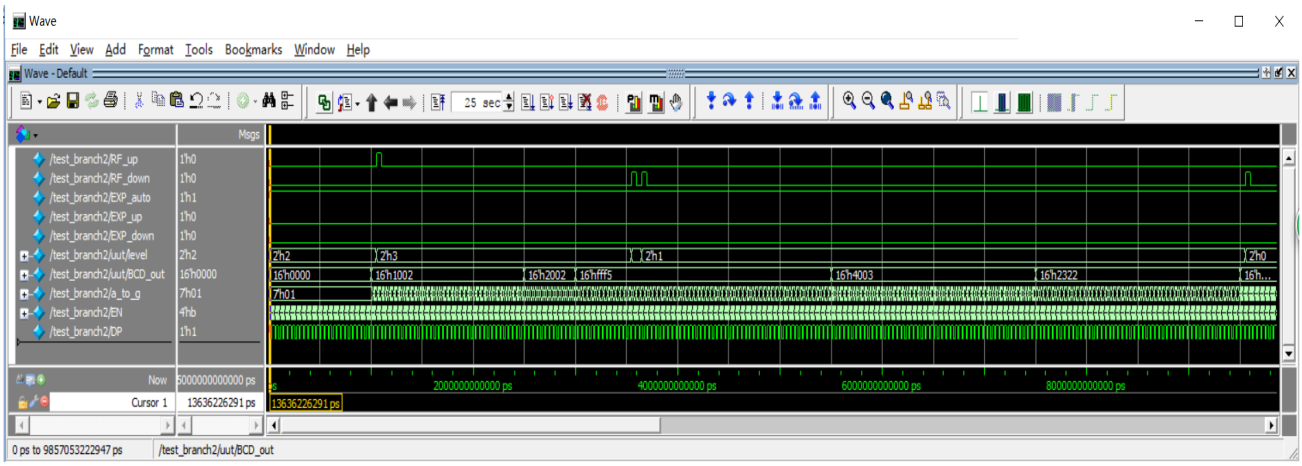
```
parameter base_period = 10;
parameter period_1s = 1000000000;

reg [31:0] cnt;
reg [31:0] period;

top uut(CLK_in, CLK_base, RF_up, RF_down, EXP_auto, EXP_up, EXP_down, a_to_g,
EN, DP);

initial begin
    CLK_base = 0;
    CLK_in = 0;
    RF_up = 0;
    RF_down = 0;
    EXP_auto = 1;
    EXP_up = 0;
    EXP_down = 0;
    cnt = 0;
    period = period_1s / 10;
    #100;
end

always begin
    #(base_period / 2) CLK_base = ~CLK_base;
end

always @(posedge CLK_base) begin
    cnt = cnt + 1;
    if (cnt == period / base_period / 2) begin
        CLK_in = ~CLK_in;
        cnt = 0;
    end
end

always @(period) begin
    cnt = 0;
end

always begin              // level : 2
    period = period_1s / 100;
```

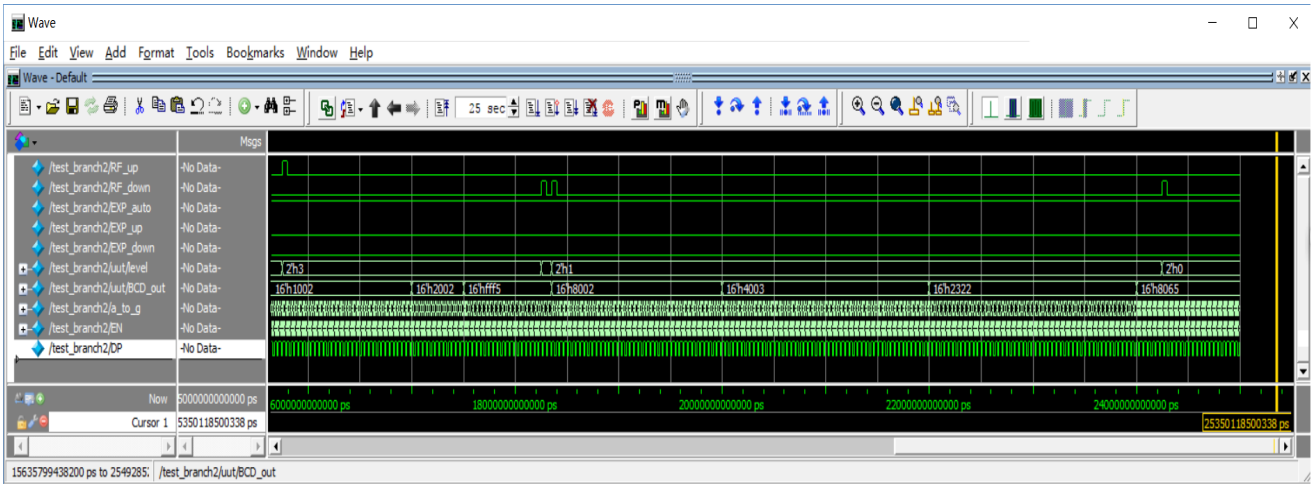
```
    #(period_1s * 1);
    #50000000 RF_up = 1;      // level : 3
    #50000000 RF_up = 0;
    #(period_1s * 1 - 100000000);

    period = period_1s / 200;
    #(period_1s / 2);
    period = period_1s / 2000000;
    #(period_1s * 1);

    period = period_1s / 4000;
    #50000000 RF_down = 1;    // level : 2
    #50000000 RF_down = 0;
    #50000000 RF_down = 1;    // level : 1
    #50000000 RF_down = 0;
    #(period_1s * 2 - 200000000);

    period = period_1s / 100;
    #(period_1s / 3);
    period = period_1s / 200;
    #(period_1s / 3);
    period = period_1s / 400;
    #(period_1s / 3);
    period = period_1s / 100;
    #(period_1s / 3);
    period = period_1s / 200;
    #(period_1s / 3);
    period = period_1s / 400;
    #(period_1s / 3);

    period = period_1s / 800000;    // 实际频率：806451Hz
    #(period_1s * 2);

    period = period_1s / 10000;
    #50000000 RF_down = 1;    // level : 0
    #50000000 RF_down = 0;
    #(period_1s * 4 - 100000000);

    period = period_1s / 1000;
    #(period_1s / 2);
    period = period_1s / 2000;
```

```
    #(period_1s / 2);
    period = period_1s / 500;
    #(period_1s * 1);
    period = period_1s / 1000;
    #(period_1s / 2);
    period = period_1s / 2000;
    #(period_1s / 2);
    period = period_1s / 500;
    #(period_1s * 1);

    period = period_1s / 20;
    #50000000 RF_up = 1;      // level : 1
    #50000000 RF_up = 0;
    #50000000 RF_up = 1;      // level : 2
    #50000000 RF_up = 0;
    #(period_1s * 1 - 100000000);

end

endmodule // test_branch2
```

仿真结果如下：







（高清图片在压缩包中）

2、固定刷新频率为 1Hz，指数手动（所需仿真时间：10+sec）

测试代码入下：

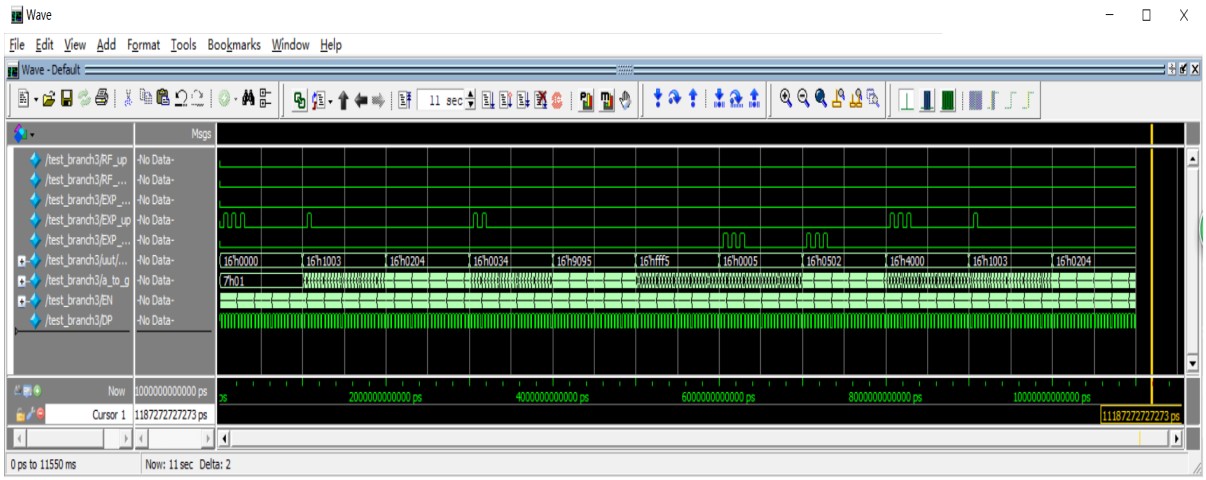
```
`timescale 1ns/1ps

module test_branch3();
reg CLK_in, CLK_base, RF_up, RF_down, EXP_auto, EXP_up, EXP_down;
wire [6:0] a_to_g;
wire [3:0] EN;
wire DP;

parameter base_period = 10;
```

```
parameter period_1s = 1000000000;

reg [31:0] cnt;
reg [31:0] period;

top uut(CLK_in, CLK_base, RF_up, RF_down, EXP_auto, EXP_up, EXP_down, a_to_g,
EN, DP);

initial begin
    CLK_base = 0;
    CLK_in = 0;
    RF_up = 0;
    RF_down = 0;
    EXP_auto = 0;
    EXP_up = 0;
    EXP_down = 0;
    cnt = 0;
    period = period_1s / 10;
    #100;
end

always begin
    #(base_period / 2) CLK_base = ~CLK_base;
end

always @(posedge CLK_base) begin
    cnt = cnt + 1;
    if (cnt == period / base_period / 2) begin
        CLK_in = ~CLK_in;
        cnt = 0;
    end
end

always @(period) begin
    cnt = 0;
end

always begin            // exp : 0
    period = period_1s / 1000;
    #50000000 EXP_up = 1;   // exp : 1
    #50000000 EXP_up = 0;
```

```
    #50000000 EXP_up = 1;   // exp : 2
    #50000000 EXP_up = 0;
    #50000000 EXP_up = 1;   // exp : 3
    #50000000 EXP_up = 0;
    #(period_1s * 1 - 300000000);

    period = period_1s / 2000;
    #50000000 EXP_up = 1; // exp : 4
    #50000000 EXP_up = 0;
    #(period_1s * 1 - 100000000);

    period = period_1s / 200;
    #(period_1s / 4);
    period = period_1s / 400;
    #(period_1s / 4);
    period = period_1s / 200;
    #(period_1s / 4);
    period = period_1s / 400;
    #(period_1s / 4);

    period = period_1s / 900000;    // 实际频率：909091Hz
    #50000000 EXP_up = 1;   // exp : 5
    #50000000 EXP_up = 0;
    #50000000 EXP_up = 1;   // exp : 5
    #50000000 EXP_up = 0;
    #(period_1s * 1 - 200000000);

    period = period_1s / 2000000;   // 超出上界
    #(period_1s * 1);
    period = period_1s / 100;       // 低于下界
    #(period_1s * 1);

    period = period_1s / 50;
    #50000000 EXP_down = 1; // exp : 4
    #50000000 EXP_down = 0;
    #50000000 EXP_down = 1; // exp : 3
    #50000000 EXP_down = 0;
    #50000000 EXP_down = 1; // exp : 2
    #50000000 EXP_down = 0;
    #(period_1s * 1 - 300000000);
```

```
    period = period_1s / 4;
    #50000000 EXP_down = 1; // exp : 1
    #50000000 EXP_down = 0;
    #50000000 EXP_down = 1; // exp : 0
    #50000000 EXP_down = 0;
    #50000000 EXP_down = 1; // exp : 0
    #50000000 EXP_down = 0;
    #(period_1s * 1 - 300000000);

end

endmodule // test_branch3
```

仿真结果入下：



3、固定刷新频率为 1Hz，指数手动（所需仿真时间 10+sec）

测试代码入下：

```
`timescale 1ns/1ps

module test_branch4();
reg CLK_in, CLK_base, RF_up, RF_down, EXP_auto, EXP_up, EXP_down;
wire [6:0] a_to_g;
wire [3:0] EN;
wire DP;

parameter base_period = 10;
```

```
parameter period_1s = 1000000000;

reg [31:0] cnt;
reg [31:0] period;

top uut(CLK_in, CLK_base, RF_up, RF_down, EXP_auto, EXP_up, EXP_down, a_to_g,
EN, DP);

initial begin
    CLK_base = 0;
    CLK_in = 0;
    RF_up = 0;
    RF_down = 0;
    EXP_auto = 0;
    EXP_up = 0;
    EXP_down = 0;
    cnt = 0;
    period = period_1s / 10;
    #100;
end

always begin
    #(base_period / 2) CLK_base = ~CLK_base;
end

always @(posedge CLK_base) begin
    cnt = cnt + 1;
    if (cnt == period / base_period / 2) begin
        CLK_in = ~CLK_in;
        cnt = 0;
    end
end

always @(period) begin
    cnt = 0;
end

always begin            // exp : 0
    period = period_1s / 1000;
    #100000000 EXP_auto = 0;
    #50000000 EXP_up = 1;   // exp : 1
```

```
    #50000000 EXP_up = 0;
    #50000000 EXP_up = 1;   // exp : 2
    #50000000 EXP_up = 0;
    #50000000 EXP_up = 1;   // exp : 3
    #50000000 EXP_up = 0;
    #(period_1s * 1 - 400000000);

    period = period_1s / 200000;
    #100000000 EXP_auto = 1; // exp : auto(5)
    #(period_1s * 1 - 100000000);

    period = period_1s / 200;
    #100000000 EXP_auto = 0;
    #50000000 EXP_down = 1; // exp : 4
    #50000000 EXP_down = 0;
    #(period_1s / 4 - 200000000);
    period = period_1s / 400;
    #(period_1s / 4);
    period = period_1s / 200;
    #(period_1s / 4);
    period = period_1s / 400;
    #(period_1s / 4);

    period = period_1s / 900000;    // 实际频率：909091Hz
    #50000000 EXP_up = 1;   // exp : 5
    #50000000 EXP_up = 0;
    #50000000 EXP_up = 1;   // exp : 5
    #50000000 EXP_up = 0;
    #(period_1s * 1 - 200000000);

    period = period_1s / 20;
    #100000000 EXP_auto = 1; // exp : auto(1)
    #(period_1s * 1 - 100000000);
    period = period_1s / 100;   // exp : auto(2)
    #(period_1s * 1);

    period = period_1s / 50;
    #50000000 EXP_down = 1; // exp : auto(1) （auto状态下手动升降无效）
    #50000000 EXP_down = 0;
    #50000000 EXP_down = 1; // exp : auto(1)
    #50000000 EXP_down = 0;
```

```
    #50000000 EXP_down = 1; // exp : auto(1)
    #50000000 EXP_down = 0;
    #(period_1s * 1 - 300000000);

    period = period_1s / 5; // exp : auto(0)
    #(period_1s * 1);

end

endmodule // test_branch4
```

仿真结果如下