5.3 Verilog 时钟分频

分类 Verilog 教程高级篇

关键词：偶数分频，奇数分频，半整数分频，小数分频

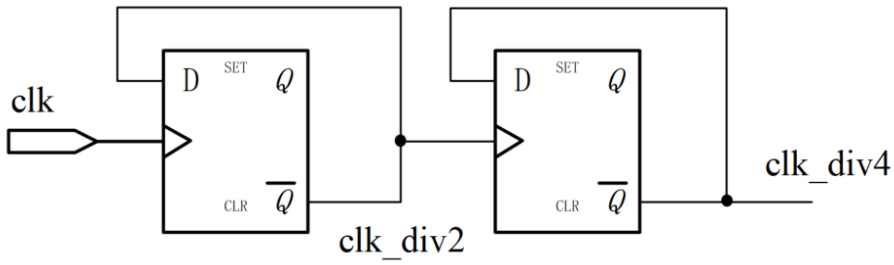
初学 Verilog 时许多模块都是由计数器与分频器组成的，例如 PWM 脉宽调制、频率计等。分频逻辑也往往通过计数逻辑完成。本节主要对偶数分频、奇数分频、半整数分频以及小数分频进行简单的总结。

偶数分频

采用触发器反向输出端连接到输入端的方式，可构成简单的 2 分频电路。

以此为基础进行级联，可构成 4 分频，8 分频电路。

电路实现如下图所示，用 Verilog 描述时只需使用简单的取反逻辑即可。



如果偶数分频系数过大，就需要对分频系数 N 循环计数进行分频。在计数周期达到分频系数中间数值 N/2 时进行时钟翻转，可保证分频后时钟的占空比为 50%。因为是偶数分频，也可以对分频系数中间数值 N/2 进行循环计数。

偶数分频的 Verilog 描述举例如下。

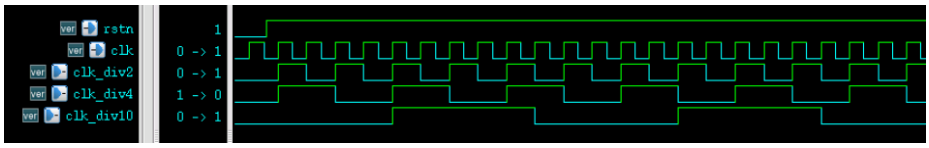
实例

```
module even_divisor
  # (parameter DIV_CLK = 10 )
    (
    input               rstn ,
    input               clk,
    output              clk_div2,
    output              clk_div4,
    output              clk_div10
    );

   //2 分频
   reg                  clk_div2_r ;
   always @(posedge clk or negedge rstn) begin
```

```
    if (!rstn) begin
        clk_div2_r      <= 'b0 ;
    end
    else begin
        clk_div2_r      <= ~clk_div2_r ;
    end
end
assign        clk_div2 = clk_div2_r ;

//4 分频
reg                     clk_div4_r ;
always @(posedge clk_div2 or negedge rstn) begin
    if (!rstn) begin
        clk_div4_r      <= 'b0 ;
    end
    else begin
        clk_div4_r      <= ~clk_div4_r ;
    end
end
assign clk_div4       = clk_div4_r ;

//N/2 计数
reg [3:0]             cnt ;
always @(posedge clk or negedge rstn) begin
    if (!rstn) begin
        cnt     <= 'b0 ;
    end
    else if (cnt == (DIV_CLK/2)-1) begin
        cnt     <= 'b0 ;
    end
    else begin
        cnt     <= cnt + 1'b1 ;
    end
end

//输出时钟
reg                     clk_div10_r ;
always @(posedge clk or negedge rstn) begin
    if (!rstn) begin
        clk_div10_r <= 1'b0 ;
    end
    else if (cnt == (DIV_CLK/2)-1 ) begin
        clk_div10_r <= ~clk_div10_r ;
    end
end
assign clk_div10 = clk_div10_r ;
endmodule
```

testbench 中只需给入激励时钟等信号即可，这里不再列出。

仿真结果如下。



奇数分频

奇数分频如果不要求占空比为 50%，可按照偶数分频的方法进行分频。即计数器对分频系数 N 进行循环计算，然后根据计数值选择一定的占空比输出分频时钟。

如果奇数分频输出时钟的高低电平只差一个 cycle ，则可以利用源时钟双边沿特性并采用"与操作"或"或操作"的方式将分频时钟占空比调整到 50%。

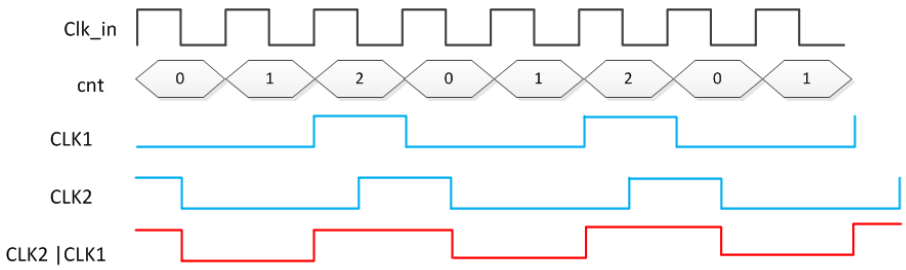
或操作调整占空比

采用"或操作"产生占空比为 50% 的 3 分频时序图如下所示。

利用源时钟上升沿分频出高电平为 1 个 cycle、低电平为 2 个 cycle 的 3 分频时钟。

利用源时钟下降沿分频出高电平为 1 个 cycle、低电平为 2 个 cycle 的 3 分拼时钟。

两个 3 分频时钟应该在计数器相同数值、不同边沿下产生，相位差为半个时钟周期。然后将 2 个时钟进行"或操作"，便可以得到占空比为 50% 的 3 分频时钟。



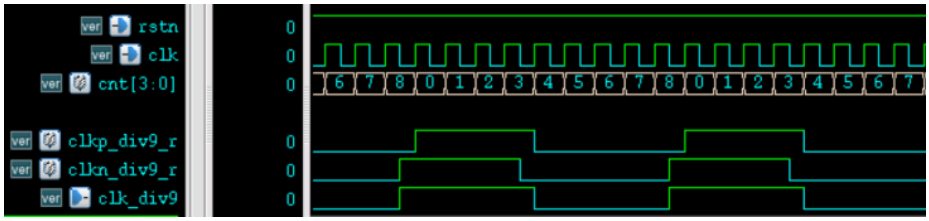
同理，9 分频时，则需要在上升沿和下降沿分别产生 4 个高电平、5 个低电平的 9 分频时钟，然后再对两个时钟做"或操作"即可。Verilog 描述如下。

实例

```
module odo_div_or
  #(parameter DIV_CLK = 9)
   (
    input               rstn ,
    input               clk,
    output              clk_div9
    );

   //计数器
   reg [3:0]            cnt ;
   always @(posedge clk or negedge rstn) begin
      if (!rstn) begin
         cnt    <= 'b0 ;
      end
      else if (cnt == DIV_CLK-1) begin
         cnt    <= 'b0 ;
      end
      else begin
         cnt    <= cnt + 1'b1 ;
      end
   end
```

```
    //在上升沿产生9分频
    reg                     clkp_div9_r ;
    always @(posedge clk or negedge rstn) begin
       if (!rstn) begin
          clkp_div9_r <= 1'b0 ;
       end
       else if (cnt == (DIV_CLK>>1)-1 ) begin //计数4-8位低电
平
         clkp_div9_r <= 0 ;
       end
       else if (cnt == DIV_CLK-1) begin //计数 0-3 为高电平
         clkp_div9_r <= 1 ;
       end
    end

    //在下降沿产生9分频
    reg                     clkn_div9_r ;
    always @(negedge clk or negedge rstn) begin
       if (!rstn) begin
          clkn_div9_r <= 1'b0 ;
       end
       else if (cnt == (DIV_CLK>>1)-1 ) begin
         clkn_div9_r <= 0 ;
       end
       else if (cnt == DIV_CLK-1) begin
         clkn_div9_r <= 1 ;
       end
    end

    //或操作，往往使用基本逻辑单元库
    // or (clk_div9, clkp_div9_r, clkn_div9_r) ;
    assign clk_div9 = clkp_div9_r | clkn_div9_r ;

endmodule
```

仿真结果如下。



与操作调整占空比

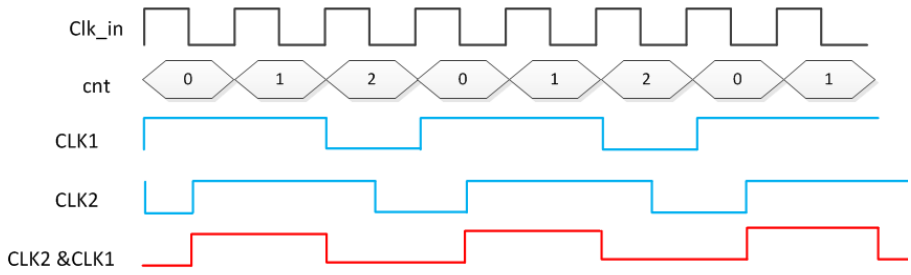
采用"与操作"产生占空比为 50% 的 3 分频时序图如下所示。

利用源时钟上升沿分频出高电平为 2 个 cycle、低电平为 1 个 cycle 的 3 分频时钟。

利用源时钟下降沿分频出高电平为 2 个 cycle、低电平为 1 个 cycle 的 3 分拼时钟。

两个 3 分频时钟应该在计数器相同数值、不同边沿下产生，相位差为半个时钟周期。然后将 2 个时钟进行"与操作"，便可以得到占空比为 50% 的 3 分频时

钟。



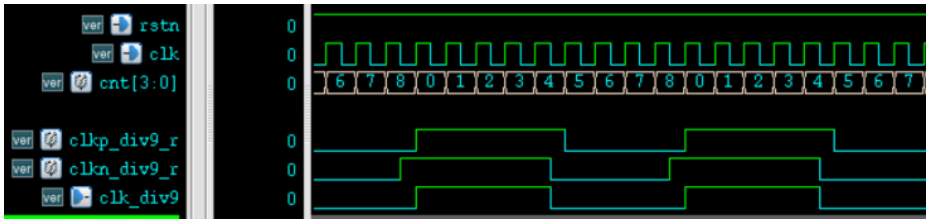
同理，9 分频时，则需要在上升沿和下降沿分别产生 5 个高电平、4 个低电平的 9 分频时钟，然后再对两个时钟做"与操作"即可。Verilog 描述如下。

实例

```
module odo_div_and
   #( parameter DIV_CLK = 9 )
   (
    input               rstn ,
    input               clk,
    output              clk_div9
    );

   //计数器
   reg [3:0]            cnt ;
   always @(posedge clk or negedge rstn) begin
      if (!rstn) begin
         cnt    <= 'b0 ;
      end
      else if (cnt == DIV_CLK-1) begin
         cnt    <= 'b0 ;
      end
      else begin
         cnt    <= cnt + 1'b1 ;
      end
   end

   //在上升沿产生9分频
   reg                  clkp_div9_r ;
   always @(posedge clk or negedge rstn) begin
      if (!rstn) begin
         clkp_div9_r <= 1'b0 ;
      end
      else if (cnt == (DIV_CLK>>1) ) begin //计数5-8位低电平
        clkp_div9_r <= 0 ;
      end
      else if (cnt == DIV_CLK-1) begin //计数 0-4 为高电平
        clkp_div9_r <= 1 ;
      end
   end

   //在下降沿产生9分频
   reg                  clkn_div9_r ;
   always @(negedge clk or negedge rstn) begin
      if (!rstn) begin
```

```
            clkn_div9_r <= 1'b0 ;
        end
        else if (cnt == (DIV_CLK>>1) ) begin
          clkn_div9_r <= 0 ;
        end
        else if (cnt == DIV_CLK-1) begin
          clkn_div9_r <= 1 ;
        end
    end

    //与操作，往往使用基本逻辑单元库
    //and (clk_div9, clkp_div9_r, clkn_div9_r) ;
    assign clk_div9 = clkp_div9_r & clkn_div9_r ;

endmodule
```

仿真结果如下。



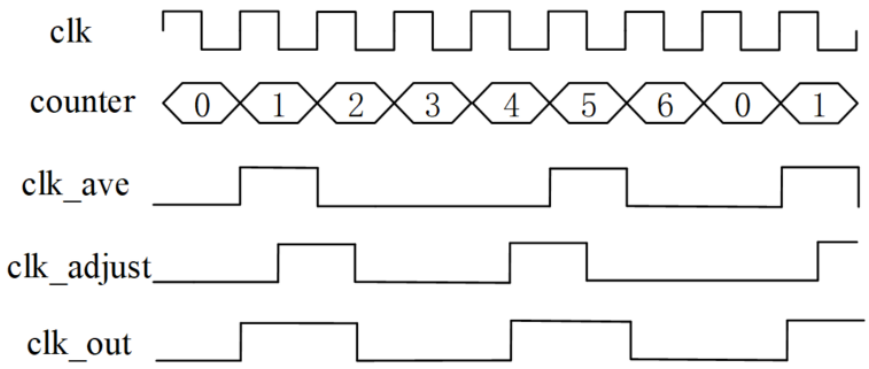
半整数分频

利用时钟的双边沿逻辑，可以对时钟进行半整数的分频。但是无论怎么调整，半整数分频的占空比不可能是 50%。半整数分频的方法有很多，这里只介绍一种和奇数分频调整占空比类似的方法。

(1) 例如进行 3.5 倍分频时，计数器循环计数到 7，分别产生由 4 个和 3 个源时钟周期组成的 2 个分频时钟。从 7 个源时钟产生了 2 个分频时钟的角度来看，该过程完成了 3.5 倍的分频，但是每个分频时钟并不是严格的 3.5 倍分频。

(2) 下面对周期不均匀的分频时钟进行调整。一次循环计数中，在源时钟下降沿分别产生由 4 个和 3 个源时钟周期组成的 2 个分频时钟。相对于第一次产生的 2 个周期不均匀的时钟，本次产生的 2 个时钟相位一个延迟半个源时钟周期，一个提前半个源时钟周期。

(3) 将两次产生的时钟进行"或操作"，便可以得到周期均匀的 3.5 倍分频时钟。分频波形示意图如下所示。



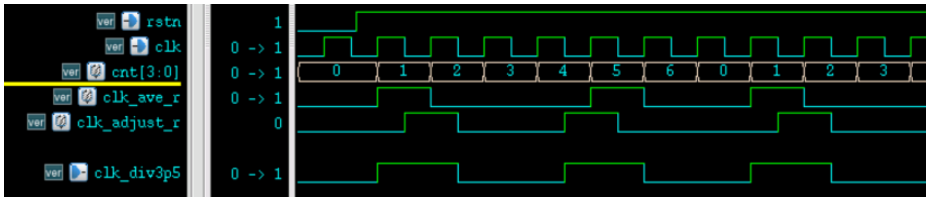
3.5 倍时钟分频的 Verilog 描述如下。

实例

```
module half_divisor(
    input               rstn ,
    input               clk,
    output              clk_div3p5
    );

   //计数器
   parameter            MUL2_DIV_CLK = 7 ;
   reg [3:0]            cnt ;
   always @(posedge clk or negedge rstn) begin
      if (!rstn) begin
         cnt    <= 'b0 ;
      end
      else if (cnt == MUL2_DIV_CLK-1) begin //计数2倍分频比
         cnt    <= 'b0 ;
      end
      else begin
         cnt    <= cnt + 1'b1 ;
      end
   end

   reg                  clk_ave_r ;
   always @(posedge clk or negedge rstn) begin
      if (!rstn) begin
         clk_ave_r <= 1'b0 ;
      end
      //first cycle: 4 source clk cycle
      else if (cnt == 0) begin
         clk_ave_r <= 1 ;
      end
      //2nd cycle: 3 source clk cycle
      else if (cnt == (MUL2_DIV_CLK/2)+1) begin
         clk_ave_r <= 1 ;
      end
      else begin
         clk_ave_r <= 0 ;
      end
   end

   //adjust
```

```
    reg                 clk_adjust_r ;
    always @(negedge clk or negedge rstn) begin
        if (!rstn) begin
            clk_adjust_r <= 1'b0 ;
        end
        //本次时钟只为调整一致的占空比
        else if (cnt == 1) begin
            clk_adjust_r <= 1 ;
        end
        //本次时钟只为调整一致的精确分频比
        else if (cnt == (MUL2_DIV_CLK/2)+1 ) begin
            clk_adjust_r <= 1 ;
        end
        else begin
            clk_adjust_r <= 0 ;
        end
    end

    assign clk_div3p5 = clk_adjust_r | clk_ave_r ;

endmodule
```

仿真结果如下。



小数分频

基本原理

不规整的小数分频不能做到分频后的每个时钟周期都是源时钟周期的小数分频倍，更不能做到分频后的时钟占空比均为 50%，因为 Verilog 不能对时钟进行小数计数。和半整数分频中第一次分频时引入的"平均频率"概念类似，小数分频也是基于可变分频和多次平均的方法实现的。

例如进行 7.6 倍分频，则保证源时钟 76 个周期的时间等于分频时钟 10 个周期的时间即可。此时需要在 76 个源时钟周期内进行 6 次 8 分频，4 次 7 分频。再例如进行 5.76 分频，需要在 576 个源时钟周期内进行 76 次 6 分频，24 次 5 分频。

下面阐述下这些分频参数的计算过程。

当进行 7 分频时，可以理解为 70 个源时钟周期内进行 10 次 7 分频。在 76 个源时钟周期内仍然进行 10 次分频，相当于将多余的 6 个源时钟周期增加、分配到 70 个源时钟周期内，即完成了 7.6 倍分频操作。分频过程中必然有 6 个分频时钟是 8 分频得到的，剩下的 4 个分频时钟则仍然会保持原有的 7 分频状态。

很多地方给出了计算这些分频参数的两元一次方程组，如下所示。其原理和上述分析一致，建议掌握上述计算过程。

```
7N + 8M = 76
N  + M   = 10
```

其中 7 为整数分频，N 整数分频的次数；8 与 M 为整数分频加一后的分频系数及其分频次数。

平均原理

以 7.6 倍分频为例，7 分频和 8 分频的实现顺序一般有以下 4 种：

(1) 先进行 4 次 7 分频，再进行 6 次 8 分频；

(2) 先进行 6 次 8 分频，再进行 4 次 7 分频；

(3) 将 4 次 7 分频平均的插入到 6 次 8 分频中；

(4) 将 6 次 8 分频平均的插入到 4 次 7 分频中。

前两种方法时钟频率不均匀，相位抖动较大，所以一般会采用后两种平均插入的方法进行小数分频操作。

平均插入可以通过分频次数差累计的方法实现，7.6 分频的实现过程如下：

(1) 第一次分频次数差值为 76-10*7 = 6 < 10，第一次进行 7 分频。

(2) 第二次差值累加结果为 6+6=12 > 10，第二次使用 8 分频，同时差值修改为 12-10=2。

(3) 第三次差值累加结果为 2+6=8 < 10，第三次使用 7 分频。

(4) 第四次差值累加结果为 8+6=14 > 10，第四次使用 8 分频，差值修改为 14-10=4。

以此类推，完成将 6 次 8 分频平均插入到 4 次 7 分频的过程。

下表展示了平均插入法的分频过程。

分频次数	差值累加	差值修改	分频周期
1	6	6	7
2	6+6=12	2	8
3	2+6=8	8	7
4	8+6=14	4	8
5	4+6=10	0	8
6	6	6	7
7	6+6=12	2	8
8	2+6=8	8	7

分频次数	差值累加	差值修改	分频周期
9	8+6=14	4	8
10	4+6=10	0	8

设计仿真

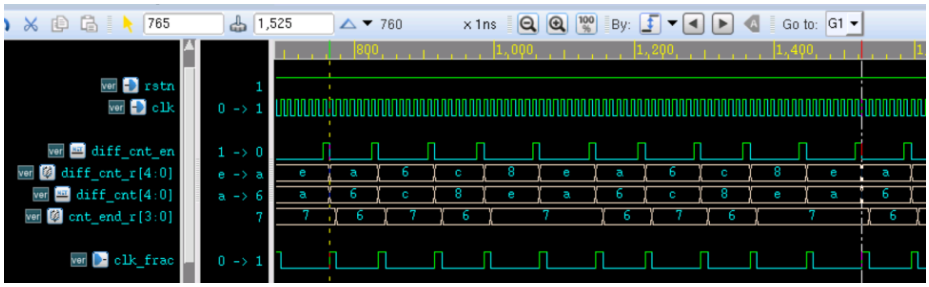
基于上述小数分频实现方法的 Verilog 描述如下。

实例

```
module frac_divisor
  #(
   parameter            SOURCE_NUM = 76 , //cycles in source
clock
   parameter            DEST_NUM   = 10  //cycles in destina
tion clock
   )
   (
    input               rstn ,
    input               clk,
    output              clk_frac
    );

   //7分频参数、8分频参数、次数差值
   parameter    SOURCE_DIV = SOURCE_NUM/DEST_NUM ;
   parameter    DEST_DIV   = SOURCE_DIV + 1;
   parameter    DIFF_ACC   = SOURCE_NUM - SOURCE_DIV*DEST_NU
M ;


   reg [3:0]            cnt_end_r ;  //可变分频周期
   reg [3:0]            main_cnt ;   //主计数器
   reg                  clk_frac_r ; //时钟输出，高电平周期数
为1
   always @(posedge clk or negedge rstn) begin
      if (!rstn) begin
         main_cnt    <= 'b0 ;
         clk_frac_r  <= 1'b0 ;
      end
      else if (main_cnt == cnt_end_r) begin
         main_cnt    <= 'b0 ;
         clk_frac_r  <= 1'b1 ;
      end
      else begin
         main_cnt    <= main_cnt + 1'b1 ;
         clk_frac_r  <= 1'b0 ;
      end
   end
   //输出时钟
   assign       clk_frac        = clk_frac_r ;
   //差值累加器使能控制
   wire         diff_cnt_en     = main_cnt == cnt_end_r ;

   //差值累加器逻辑
   reg [4:0]            diff_cnt_r ;
```

```
    wire [4:0]           diff_cnt = diff_cnt_r >= DEST_NUM ?
                                    diff_cnt_r -10 + DIFF_ACC
:
                                    diff_cnt_r + DIFF_ACC ;

    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            diff_cnt_r <= 0 ;
        end
        else if (diff_cnt_en) begin
            diff_cnt_r <= diff_cnt ;
        end
    end

    //分频周期变量的控制逻辑
    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            cnt_end_r      <= SOURCE_DIV-1 ;
        end
        //差值累加器溢出时，修改分频周期
        else if (diff_cnt >= 10) begin
            cnt_end_r      <= DEST_DIV-1 ;
        end
        else begin
            cnt_end_r      <= SOURCE_DIV-1 ;
        end
    end

endmodule
```

仿真结果如下。



分频小结

偶数分频不使用时钟双边沿逻辑即可完成占空比为 50% 的时钟分频，是最理想的分频状况。

奇数分频如果要产生 50% 占空比的分频时钟，则需要使用时钟的双边沿逻辑。

如果不要求占空比的话，实现方法和偶数分频类似。

半整数分频属于特殊的小数分频，可以用双边沿逻辑进行设计。通过一定逻辑将两个双边沿时钟信号整合为最后的一路输出时钟时，建议不要使用选择逻辑。因为容易出现毛刺现象，电路中又会增加一定的不确定性。例如下面描述是不建议的。

```
assign clk_div3p5 = (cnt == 1 || cnt ==2) ? clk_ave_r
                                          : clk_adjust_r ;
```

小数分频的基本思想是，输出时钟在一段时间内的平均频率达到分频要求即可。

但是考虑到相位抖动，还需要对分频系数变化的分频逻辑进行平均操作。

本章节源码下载

Download

← 5.2 Verilog 时钟简介 5.4 Verilog 时钟切换 →