DDS 电路生成 各种调制 分频器

DDS 电路生成：
matlab 或 c 语言 通过 excel 生成波形数据
先生成所需信号：

```
F1=1; %信号的频率
Fs=2048;%采样频率
P1=0;%信号初始相位
N=2047;%采样点数为 N+1
t=[0:1/Fs:N/Fs];%采样时刻
ADC=511;%直流分量
A=511;%信号幅度
s=A*sin(2*pi*F1*t + pi*P1/180) + ADC;%生成信号
plot(s);%绘制图形
```

在 MATLAB 主界面中，右侧的 workspace 栏中，选中 name 为 s 的一项，双击，便可打开该数组的值。可能需要通过 excel 进行数据处理（单元格选择数值：位数 0），生成的数据即为所需。

Quartus 进行 mif 制作：file→new→Memory Initialization File，设置相应的 mif 文件大小设置选项中。

下面我们推倒 DDS 输出的正弦波的频率公式：

假设采样时钟的频率为 fc，频率控制字为 M，相位累加寄存器的位宽为 n，则相位累加器中的最大值为 2^n-1,当超过该值时，相位累加寄存器中的值就会溢出，然后从 0 开始计数。

因为采样时钟的频率为 fc，频率控制字为 M，则每隔 1/fc,相位累计寄存器中的值就增加 M，所以，在 1s 内，相位累加寄存器中的值增加的大小为(1/(1/fc))*M = fc * M，又因为寄存器中最大表示的值为 2^n-1，所以可以计算出 1s 内溢出的次数为 fc * M / 2^n，因为寄存器中的值从 0 增加到 2^n-1，刚好输出一个周期的正弦波信号。所以，寄存器溢出的次数就是输出正弦波的周期数。

1s 内输出正弦波的周期数就为 DDS 输出的正弦波的频率，也就是我们的目标频率 f0 = fc * M / 2^n。

比如我们上一篇中，fc = 50Mhz，M = 1，n = 8，所以可以计算出目标频率 f0 = 195Mhz；当 M = 2 时，f0 = 390Mhz；M = 2 刚好是 M = 1 时目标频率的 2 倍，

当已知目标频率，求频率控制字 M 时，M = f0 * 2^n / fc。

```
module DDS(
    clk,//时钟
    reset_n,//复位
    Fword,//频率控制字
    Pword,//相位控制字
    DA_Data//数据输出
```

```
);

    input clk;
    input reset_n;
    input [31:0] Fword;//控制位数
    input [10:0] Pword;//Rom 地址 11 位：横坐标最大为：2048
    output [9:0] DA_Data;

    reg [31:0] r_Fword;//频率  寄存器
    reg [10:0] r_Pword;//相位  寄存器

    reg [31:0] Fcnt;//累加器

    wire [10:0] rom_addr;

    always @(posedge clk)
        begin
            r_Fword <= Fword;
            r_Pword <= Pword;
        end


    always @(posedge clk or negedge reset_n)
        if(!reset_n)
            Fcnt <= 32'd0;
        else
            Fcnt <= Fcnt + r_Fword;//Fcnt：相位累加器

    assign rom_addr = Fcnt[31:21] + r_Pword;

    rom rom(
    .address(rom_addr),
    .clock(clk),
    .q(DA_Data)
    );

endmodule
```

```
`timescale 1ns/100ps
module DDS_tb;
    reg clk;
    reg reset_n;
    reg [31:0] Fword;
    reg [10:0] Pword;//Rom 地址 11 位
    wire [9:0] DA_Data;
```

```
    DDS DDS(
    .clk(clk),//时钟
    .reset_n(reset_n),//复位
    .Fword(Fword),//频率控制字
    .Pword(Pword),//相位控制字
    .DA_Data(DA_Data)//数据输出
    );

    initial clk = 1;
    always #10 clk = ~clk;

    initial
        begin
            reset_n = 0;
            Fword = 1000000;
            Pword = 0;
            #1;
            reset_n = 1;
        end
endmodule
```

生成 ROM：Tools→MegaWizard Plug-In Manager
Memory compiler → ROM：1port 给文件命名，选择位宽和字节

激励文件的配置：将编写的激励文件 DDS_tb 添加到 test bench 中
我们已经得到 2 个“.v”文件，“DDS.v”作为顶层设计，包含整个模块的设计信息；“DDS_tb”是激励文件，是模仿外加激励信号将其加入到模块中。
①Assignments→Settings ②Simulation， ③选择 Compile test bench→Test Benches→New
④Test bench name：写入激励文件名：DSS_tb ⑤选择 File name 旁的...，选择 DDS_tb.v 返回上一层单击 add

进行 RTL Simulation 软件自动打开 Model Sim：
设置数据显示：Analog（custom） 字符类型 unsigned 点击 zoom full

m 序列码： // 3 位 起始值 000

```
module m_ser(clk,reset_n,load,m_ser_out);
  input clk;
  input reset_n;
  input load;
  output m_ser_out;
  wire clk;
  wire reset_n;
  wire load;
```

```
  reg m_ser_out;
  reg[2:0] m_code;
  always@(posedge clk or negedge reset_n)
  begin
    if(!reset_n)
      begin
        m_code<=3'b000;
        m_ser_out<=1'b0;
      end
    else
      if(load)
        begin
          m_code<=3'b001;//?????
          m_ser_out<=m_code[2];
        end
      else
        begin
          m_code[2:1]<=m_code[1:0];
          m_code[0]<=m_code[2]^ m_code[0];//?2?0?????????0
          m_ser_out<=m_code[2];
        end
  end
Endmodule
```

---

test_m_ser：

```
module test_m_ser;
  reg clk;
  reg reset_n;
  reg load;
  wire m_ser_out;
  m_ser m_ser(clk,reset_n,load,m_ser_out);

  initial begin
    reset_n=1'b0;
    load=1'b1;
    clk=1'b0;
    #30
    reset_n=1'b1;
    #30
    load=1'b0;
    #1000 $stop;
  end
  always begin
    #10 clk=~clk;
  end
```

```
endmodule
```

ASK 调制：

```
module ask(clk,reset_n,data_in,ask_code_out);
   input clk;
   input reset_n;
   input data_in;
   output ask_code_out;
   wire clk;
   wire reset_n;
   wire data_in;
   reg[2:0]clk_cnt;
   reg clk_div;
   always@(posedge clk or negedge reset_n)//??????
      begin
         if(!reset_n)
            begin
               clk_cnt<=3'd0;
               clk_div<=1'b0;
            end
         else if(clk_cnt==3'd1)
            begin
               clk_div<=~clk_div;
               clk_cnt<=3'd0;
            end
         else
            clk_cnt<=clk_cnt +1'b1;
      end
   assign ask_code_out=(data_in)? clk_div :1'b0;
Endmodule
```

```
module test_ask;
   reg clk;
   reg reset_n;
   reg data_in;
   reg load;
   wire m_ser_out;
   wire ask_code_out;
   m_ser m_ser(clk,reset_n,load,m_ser_out);
   ask ask(clk,reset_n,m_ser_out,ask_code_out);
   initial begin
      reset_n=1'b0;
      load=1'b1;
      clk=1'b0;
      #30
```

```
    reset_n=1'b1;
    #30
    load=1'b0;
    #1000 $stop;
  end
  always begin
    #10 clk=~clk;
  end
Endmodule
```

FSk 调制

```
module fsk_code(clk,m_ser_code_in,fsk_code_sin_out);
  input clk;
  input m_ser_code_in;
  output fsk_code_sin_out;
  wire clk;
  wire m_ser_code_in;
  reg[2:0]cnt=3'b000;
  wire f1;
  reg f2=1'b1;
  always@(posedge clk)
  begin
    if(cnt==3'b010)
      begin
        cnt<=3'd0;
        f2<=~f2;
      end
    else
      cnt<=cnt+1'b1;
  end
  assign f1=clk;
  assign fsk_code_sin_out=(m_ser_code_in)? f2:f1;
Endmodule
```

```
module test_fsk;
  reg clk;
  reg reset_n;
  reg load;
  wire m_ser_out;
  wire ask_code_out;
  m_ser m_ser(clk,reset_n,load,m_ser_out);
  fsk_code fsk_code(clk,m_ser_out,fsk_code_sin_out);
  initial begin
    reset_n=1'b0;
    load=1'b1;
    clk=1'b0;
```

```
    #30
    reset_n=1'b1;
    #30
    load=1'b0;
    #1000 $stop;
  end
  always begin
    #30 clk=~clk;
  end
endmodule
```

一、占空比为 50%的分频器。

1、偶分频：

偶分频电路指的是分频系数为 2、4、6、8 … 等偶数整数的分频电路，我们可以直接进行分频。例如下面 divider.v 中，对输入时钟进行 6 分频，即假设 clk 为 50MHz，分频后的时钟频率为 (50/6) MHz。

程序如下：

设计代码：

```
//rtl
module divider(
    clk,
    rst_n,
    clk_div
);
    input clk;
    input rst_n;
    output clk_div;
    reg clk_div;

    parameter NUM_DIV = 6;
    reg    [3:0] cnt;

always @(posedge clk or negedge rst_n)
    if(!rst_n) begin
        cnt     <= 4'd0;
        clk_div    <= 1'b0;
    end
    else if(cnt < NUM_DIV / 2 - 1) begin
        cnt     <= cnt + 1'b1;
        clk_div    <= clk_div;
    end
    else begin
        cnt     <= 4'd0;
        clk_div    <= ~clk_div;
    end
```

```
 Endmodule
```

```
//tb   仿真程序
 module divider_tb();
      reg clk;
      reg rst_n;
      wire clk_div;
      parameter DELY=100;
divider U_divider(
      .clk      (clk      ),
      .rst_n      (rst_n      ),
      .clk_div(clk_div)
);
      always #(DELY/2) clk=~clk;//产生时钟波形
initial begin
      $fsdbDumpfile("divider_even.fsdb");
      $fsdbDumpvars(0,U_divider);
 end
 initial begin
            clk=0;rst_n=0;
      #DELY rst_n=1;
      #((DELY*20)) $finish;
end
endmodule
```

2、奇分频：

由于奇分频需要保持分频后的时钟占空比为 50% ，所以不能像偶分频那样直接在分频系数的一半时使时钟信号翻转(高电平一半，低电平一半)。在此我们需要利用输入时钟上升沿和下降沿来进行设计。接下来我们设计一个 5 分频的模块，设计思路如下：

采用计数器 cnt1 进行计数，在时钟上升沿进行加 1 操作，计数器的值为 0、1 时，输出时钟信号 clk_div 为高电平；计数器的值为 2、3、4 时，输出时钟信号 clk_div 为低电平，计数到 5 时清零，从头开始计数。我们可以得到占空比为 40% 的波形 clk_div1。

采用计数器 cnt2 进行计数，在时钟下降沿进行加 1 操作，计数器的值为 0、1 时，输出时钟信号 clk_div 为高电平；计数器的值为 2、3、4 时，输出时钟信号 clk_div 为低电平，计数到 5 时清零，从头开始计数。我们可以得到占空比为 40% 的波形 clk_div2。

clk_div1 和 clk_div2 的上升沿到来时间相差半个输入周期，所以将这两个信号进行或操作，即可得到占空比为 50% 的 5 分频时钟。程序如下：

设计代码：

```
//rtl
module divider(
      clk,
      rst_n,
      clk_div
);
      input clk;
      input rst_n;
```

```
    output clk_div;

    parameter NUM_DIV = 5;
    reg[2:0] cnt1;
    reg[2:0] cnt2;
    reg      clk_div1, clk_div2;

always @(posedge clk or negedge rst_n)
    if(!rst_n)
        cnt1 <= 0;
    else if(cnt1 < NUM_DIV - 1)
        cnt1 <= cnt1 + 1'b1;
    else
        cnt1 <= 0;

always @(posedge clk or negedge rst_n)
    if(!rst_n)
        clk_div1 <= 1'b1;
    else if(cnt1 < NUM_DIV / 2)
        clk_div1 <= 1'b1;
    else
        clk_div1 <= 1'b0;

always @(negedge clk or negedge rst_n)
    if(!rst_n)
       cnt2 <= 0;
    else if(cnt2 < NUM_DIV - 1)
       cnt2 <= cnt2 + 1'b1;
    else
       cnt2 <= 0;

always @(negedge clk or negedge rst_n)
    if(!rst_n)
        clk_div2 <= 1'b1;
    else if(cnt2 < NUM_DIV / 2)
        clk_div2 <= 1'b1;
    else
        clk_div2 <= 1'b0;

assign clk_div = clk_div1 | clk_div2;
endmodule
```

```
//tb   仿真
module divider_tb();
    reg clk;
```

```
    reg rst_n;
    wire clk_div;
    parameter DELY=100;
divider U_divider(
    .clk     (clk     ),
    .rst_n   (rst_n   ),
    .clk_div(clk_div)
);
    always #(DELY/2) clk=~clk;//产生时钟波形
initial begin
    $fsdbDumpfile("divider_odd.fsdb");
    $fsdbDumpvars(0,U_divider);
 end
 initial begin
        clk=0;rst_n=0;
    #DELY rst_n=1;
    #((DELY*20)) $finish;
end
endmodule
```

二、任意分频

在 verilog 程序设计中，我们往往要对一个频率进行任意分频，而且占空比也有一定的要求。这样的话，对于程序有一定的要求。现在在前面两个实验的基础上做一个简单的总结，实现对一个频率的任意占空比的任意分频。

比如： FPGA 系统时钟是 50MHz，而我们要产生的频率是 880Hz，那么，我们需要对系统时钟进行分频。很容易想到用计数的方式来分频：50000000/880 = 56818。显然这个数字不是 2 的整幂次方，那么我们可以设定一个参数，让它到 56818 的时候重新计数就可以实现了。程序如下：

```
//rtl
module div(
    clk,
    rst_n,
    clk_div
);
    input clk,rst_n;
    output clk_div;

    reg [15:0] counter;

always @(posedge clk or negedge rst_n)
    if(!rst_n)
        counter <= 0;
    else if(counter==56817)
        counter <= 0;
    else
        counter <= counter+1;
```

```
assign clk_div = counter[15];
endmodule
```

三、任意占空比

下面我们来算一下上面产生的时钟的占空比：

我们清楚地知道，这个输出波形在 counter 为 0 到 32767（2 的 15 次方）的时候为低，在 32768 到 56817 的时候为高，占空比为 40%多一些，如果我们需要占空比为 50%，那么我们需要再设定一个参数，使它为 56817 的一半，使达到它的时候波形翻转，就可以实现结果了。程序如下：

28408=56818/2-1，计数到 28408 就清零，翻转，其余的计数期间，保持不变。

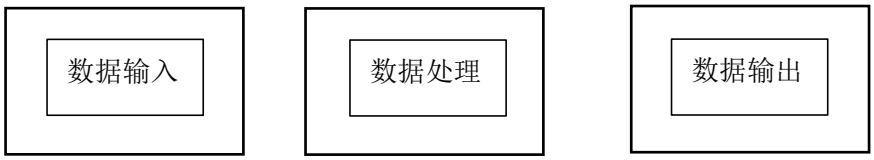
```
//rtl
module div(
    clk,
    rst_n,
    clk_div
);
    input clk,rst_n;
    output clk_div;
    reg clk_div;
    reg [14:0] counter;
always @(posedge clk or negedge rst_n)
    if(!rst_n)
        counter <= 0;
    else if(counter==28408)
        counter <= 0;
    else
        counter <= counter+1;

always @(posedge clk or negedge rst_n)
    if(!rst_n)
        clk_div <= 0;
    else if(counter==28408)
        clk_div <= ~clk_div;
endmodule
```

冒泡排序：

一、 设计思路概述

搭建状态机分布处理，处理过程如下。

二、 逻辑框图




三、 状态机时序

初始状态：S0

S0:是否输入八位 4bit 待比较数据，输入完成则进入 S1

S1:是否处理完冒泡排序，处理完成进入 S2

S2:输出数据，进入 S3

S3:等待一会，回到 S0

```
module bub(
    input clk,
    input rst,
    input [3:0]data_in,
    output reg[2:0]state,
    output reg[3:0]data_out
);
reg[3:0] i,p;//计数器
reg[3:0]content[7:0];

initial begin
state<=0;i<=0;p<=7;
end
always@(posedge clk or posedge rst )
begin
if(rst)begin state<=0;data_out<=0;i<=0;p<=7;end

else begin
    case(state)
        3'b000:begin//读取数据

                content[i]=data_in;
                i=i+1;
                if(i==8)begin i<=0;state<=3'b001;end
        end //000

        3'b001:begin//冒泡排序
           if(i<p)
begin
                    if(content[i]<content[i+1])begin //如果当前位置比后面小
则对调
                       content[i+1]<=content[i];
                       content[i]<= content[i+1];
```

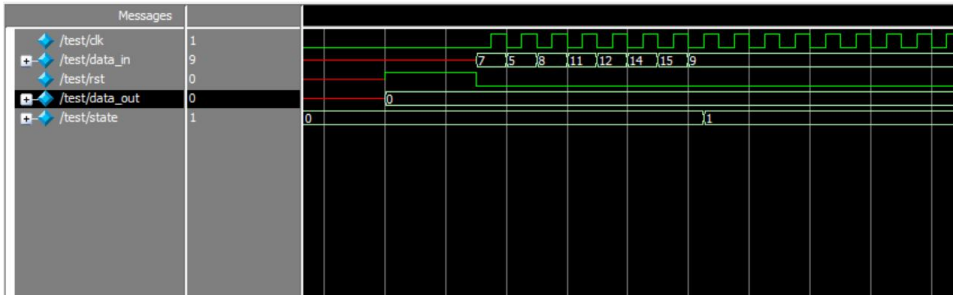
```
                    end
                  i<=i+1;
                 end
           else begin
                 i<=0; p<=p-1;
                 end
           if (p==0&&i==0)  state<=3'b010;

        end//001
        3'b010:begin //发送数据
                data_out=content[i];
                i=i+1;
                if(i==8) begin i=0; state=3'b011; end
        end
        3'b011:begin//延时一段时间
                i=i+1;
                if(i==15)begin i=0;state=0;end
        end//011
    endcase
end//else

end//always
endmodule
```

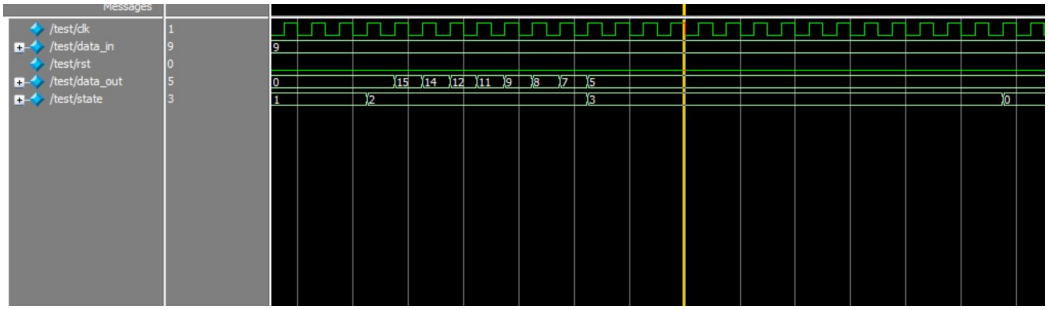
测试文件：

```
`timescale 1 ns/ 1 ps
module test;
reg clk;
reg [3:0] data_in;
reg rst;
wire [3:0]  data_out;
wire [2:0]  state;

// assign statements (if any)
bub i1 (
    .clk(clk),
    .data_in(data_in),
    .data_out(data_out),
    .rst(rst),
    .state(state)
);
initial
begin
    clk=0;
    #100
```

```
    rst=1;
    #30
    rst=0;
    forever #5 clk=~clk;
end
initial
begin
#130
        while(1)
        begin
            data_in=7;
            #10
            data_in=5;
            #10
            data_in=8;
            #10
            data_in=11;
            #10
            data_in=12;
            #10
            data_in=14;
            #10
            data_in=15;
            #10
            data_in=9;
            #1000
            rst=1;
            #30
            rst=0;
      end
end
endmodule
```

仿真结果：




Messages
/test/clk
/test/data_in
/test/rst
/test/data_out
/test/state
1
9
0
5
3
9
0
15
14
12
11
9
8
7
5
1
2
3
0