用 FPGA 产生正弦信号

正弦信号，是一个模拟信号。而 FPGA 只能产生数字信号。因此需要用 DA 将数字量转化为模拟量。这里采用 modelsim 的模拟波形显示，就不需要 DA 模块了。产生正弦信号的方法有很多，这里用的是查找 rom 的方法，产生正弦信号。

正弦信号，是一个介于-1 和 1 之间的模拟量。而正弦信号是周期变化的，因此这里只需要将半个正弦信号周期的值存进 rom 里，其余周期可以根据这个半个周期的值变化可得到。用 matlab 产生正弦信号的值。以 0.01 为步长，从 0 采集到 pi/2。共 158 个点。

因此 sin 的值是小数，而 FPGA 是不能表示小数的，因此需要将小数扩大，以整数来表示。此次是以 12 位二进制来显示这整数。扩大的方法就是乘以 2^12 即可了。12 位二进制数最大能表示 2^12-1 的值。这里用 0 表示最小值 0,2^12-1 表示最大值 1。则 0 到 1 中间的数，就可以扩大到用 0 到 2^12-1 中间的数来表示了。例如，0.3，就可用整数 0.3*2^12=1228.8，在向上取整，为 1228，当然这样表示会有误差。

这里 rom 用的是 ISE 的 IP rom。将 matlab 生成的数据放进 rom 里面。然后依次读取 rom 的值，即可生成正弦信号了。

首先是生成 rom 的初始化文件。Xilinx 的 rom 的初始化文件的后缀为.coe。而这个特殊文件有固定的格式。

```
MEMORY_INITIALIZATION_RADIX = 10;
MEMORY_INITIALIZATION_VECTOR =
0,40,81,122,163,204,245,286,327,368,408,
449,490,530,571,612,652,692,733,773,813,
853,893,933,973,1013,1053,1092,1131,1171,1210,
1249,1288,1327,1365,1404,1442,1481,1519,1557,1595,
1632,1670,1707,1744,1781,1818,1855,1891,1927,1963,
1999,2035,2070,2105,2140,2175,2210,2244,2278,2312,
2346,2379,2413,2446,2478,2511,2543,2575,2607,2638,
2669,2700,2731,2761,2791,2821,2851,2880,2909,2938,
2966,2994,3022,3050,3077,3104,3130,3156,3182,3208,
3233,3258,3283,3307,3331,3355,3378,3401,3424,3446,
3468,3490,3511,3532,3552,3573,3593,3612,3631,3650,
3668,3686,3704,3721,3738,3755,3771,3787,3802,3817,
3832,3846,3860,3873,3887,3899,3912,3924,3935,3946,
3957,3967,3977,3987,3996,4005,4013,4021,4029,4036,
4043,4049,4055,4061,4066,4070,4075,4079,4082,4085,
4088,4090,4092,4094,4095,4095,4095;
```

第一行的 10 表示下面的数字是 10 进制的。。。 后面的数据是依次存入 rom 的值，以逗号分开，最后一个以分号结束。前面两行的内容是固定的。

生成初始化文件的 matlab 如下所示：

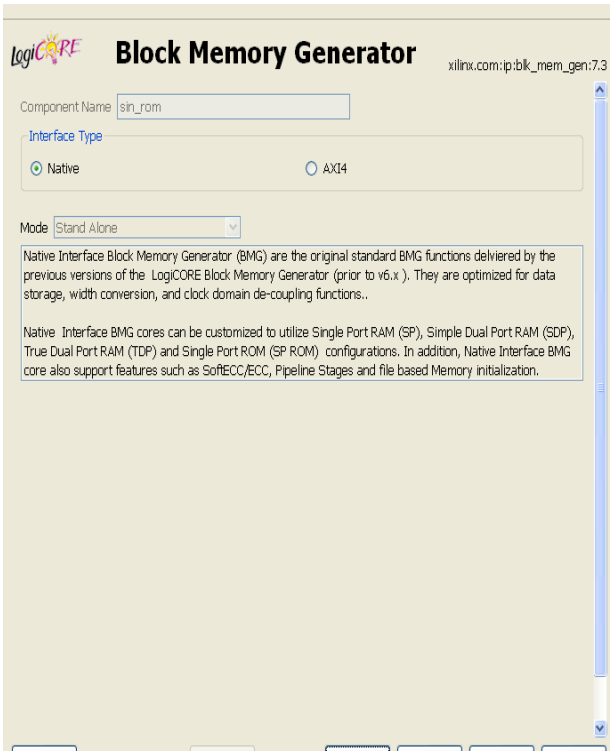
```
fid = fopen('sin_rom.txt','w');
fprintf(fid,'MEMORY_INITIALIZATION_RADIX = 10;\n');
fprintf(fid,'MEMORY_INITIALIZATION_VECTOR =\n');
for i = 0:1:pi/2*100
    y = sin(i/100);
    rom =floor( y * 2^12);
    if i == 157
        fprintf(fid,'%d;',rom);
    else
```

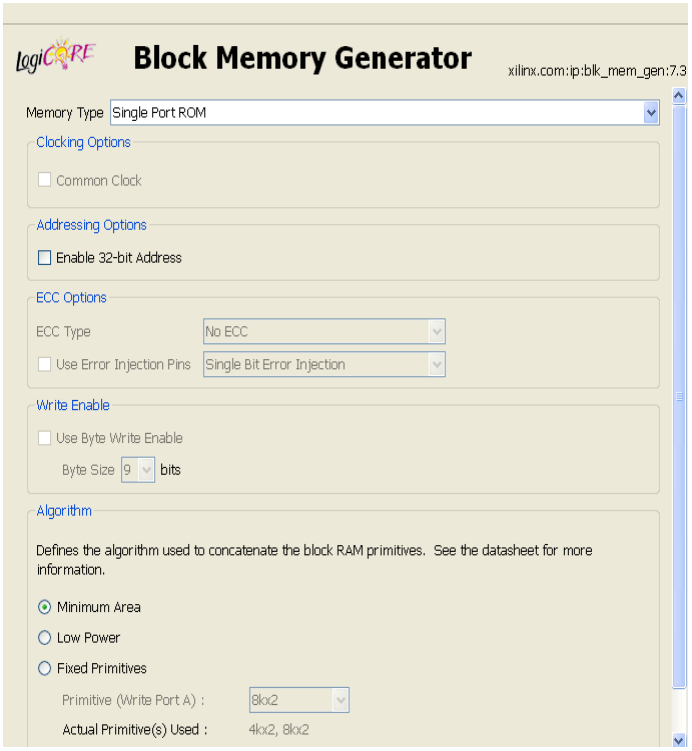
```
        fprintf(fid,'%d,',rom);
    end

    if mod(i,10)==0 && i ~= 0
        fprintf(fid,'\n');
    end
end
fclose(fid);
```

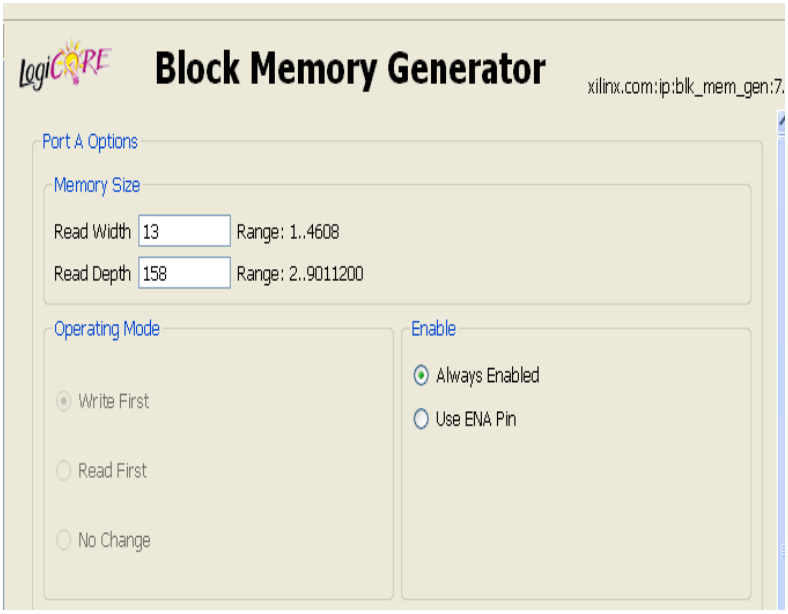
生成的文件是.txt 文件。将后缀直接改为.coe 即可。然后复制到 ISE 分工程目录下。

初始化文件生成后，剩下就是编写 verilog 代码。

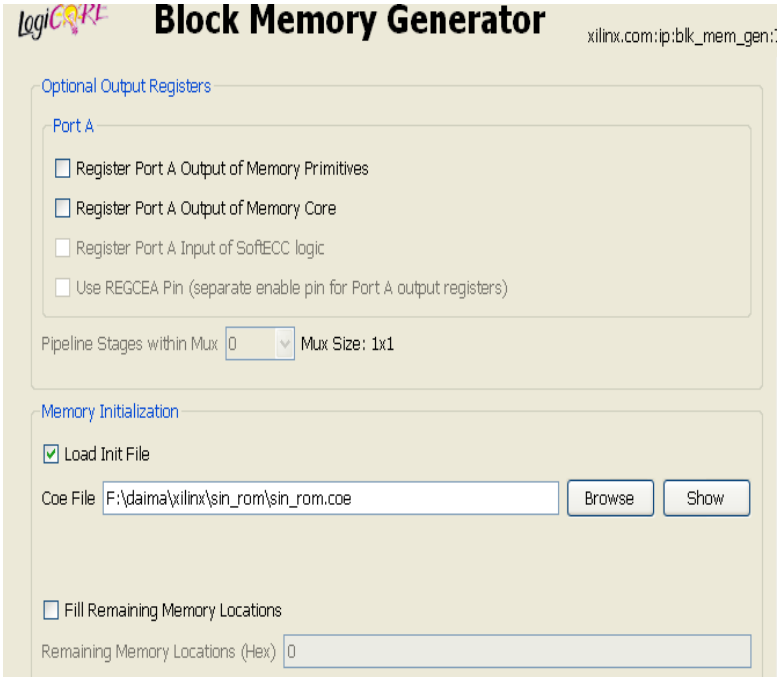
首先建一个工程，然后新建一个 IP。找到 rom，打开。设置按如下设置：







这里是设置位宽，采用的是 13 位（第一位为符号位，后面 12 位为数据位）来显示 sin 的值。因此这里是设置为 13.深度是因为要存 158 个值，所以这里设置为 158.



这里将刚刚生成的.coe 文件载入，如果没有显示红色，就说明正确，否则错误。错误的原因是数据和设定的深度没有对齐。

接下来直接生成就行了。

接着就是编写 verilog 代码了。

首次查看用 ip 生成的 rom 的 HDL 例化代码.

```
sin_rom your_instance_name (
  .clka(clka), // input clka
  .addra(addra), // input [7 : 0] addra
  .douta(douta) // output [12 : 0] douta
```

```
);
```

从上面的程序可看出，只需要给时钟信号，和输入地址，就可以了。输出的就是正弦的数字信号了。分析正弦信号，前 1/4 个周期，地址从 0 自加，一直加到 157（1/4 个周期的点数）。然后再自减，减到 0。然后进入到负半周了。大家都知道，负数的表示是以二进制的补码来表示的，即绝对值数的二进制取反在加一。

因此要编写地址自加自减的代码，然后再根据地址的值，判断输出的值是正数，还是负数，负数的话，rom 的输出值还要取反加一后再输出。

其代码，如下所示：

```
module sin_top(
    input clk,                          //输入时钟信号
    input [9:0] address,                    //输入地址信号
    output reg [12:0] data_out           //输出sin的数字值
);

  reg [7:0] add;
  wire [12:0] douta;
  //  以下是判断地址的值
  always@ *  begin
     if( address <= 157 )
         add = address;
     else if( address <= 315 )
         add = 10'd315 - address;
     else if( address <= 473 )
         add = address - 10'd316;
     else if( address <= 631 )
         add = 10'd631 - address;
     else
         add = 0;
  end

  //例化之前生成的sin_rom
  sin_rom u1_sin_rom (
  .clka(clk), // input clka
  .addra(add), // input [7 : 0] addra
  .douta(douta) // output [12 : 0] douta
  );
  // 判断输出值是正数还是负数。
  always@ *  begin
     if( address <= 315 )
         data_out = douta;
     else if( address <= 631 )
         data_out = ~douta + 1'b1;
     else
```

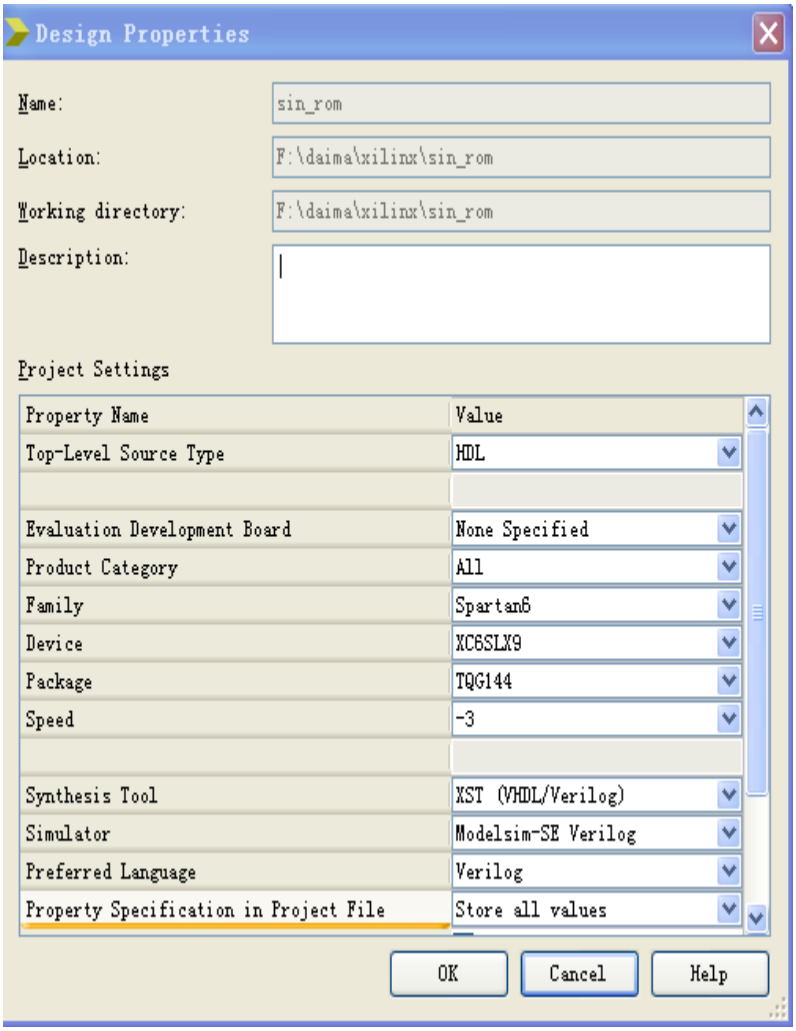
```
        data_out = 0;
  end
endmodule
```

程序写好了，剩下就是要仿真了。仿真的 testbench，只需要输入时钟和地址信号就可以了。地址信号一直加一，直到不小于 631，刚好一个周期结束。就返回 0 值，在继续自加。

测试代码如下所示：

```
module  sin_top_test;
    // Inputs
    reg clk;
    reg [9:0] address;

    // Outputs
    wire [12:0] data_out;
    // Instantiate the Unit Under Test (UUT)
    sin_top uut (
        .clk(clk),
        .address(address),
        .data_out(data_out)
    );

    always#5  clk = ~clk;

    initial begin
        // Initialize Inputs
        clk = 0;
        address = 0;
        // Wait 100 ns for global reset to finish
        while(1)
         begin
            @(negedge(clk));
              if( address < 631 )
                  address = address + 1;
              else
                  address = 0;
         end
    end
endmodule
```

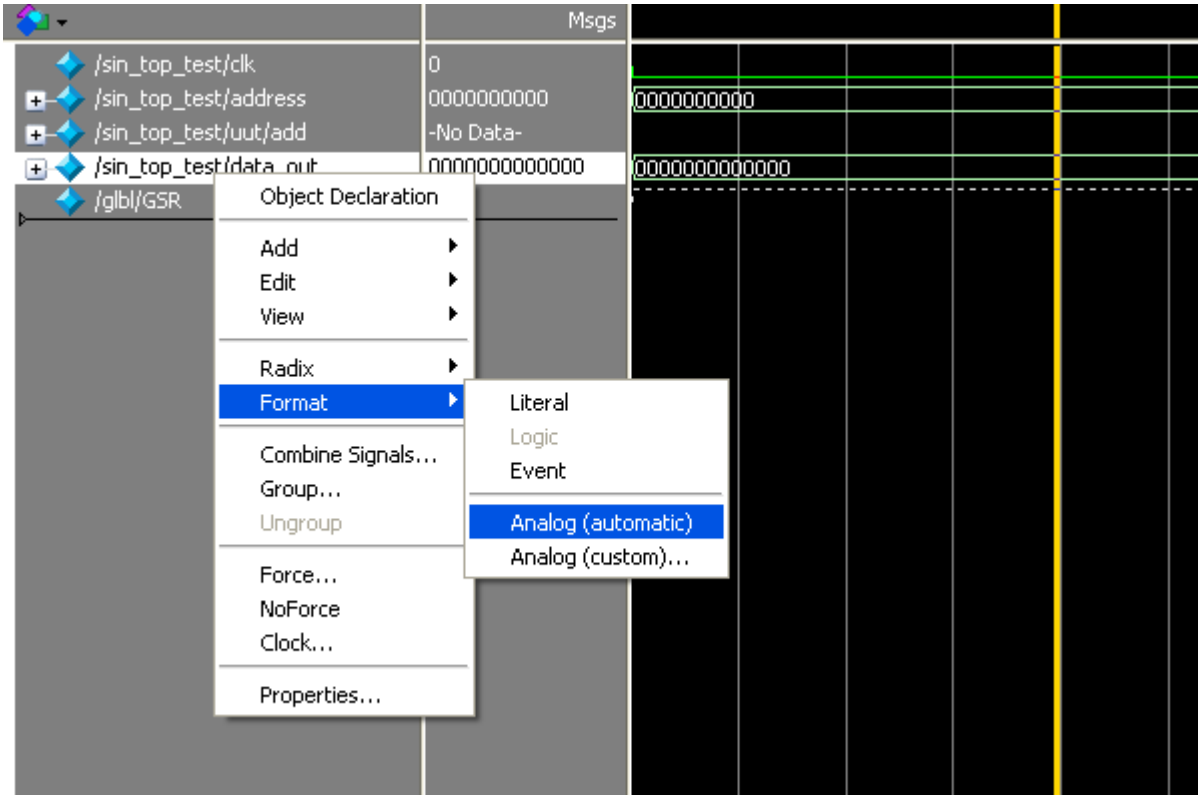
接着用 modelsim 仿真。因为有用到 xilinx 的 IP ROM。因此用 modelsim 单独仿真比较麻烦。因此这里是直接在 ISE 里面调用 modelsim，这样比较简单。



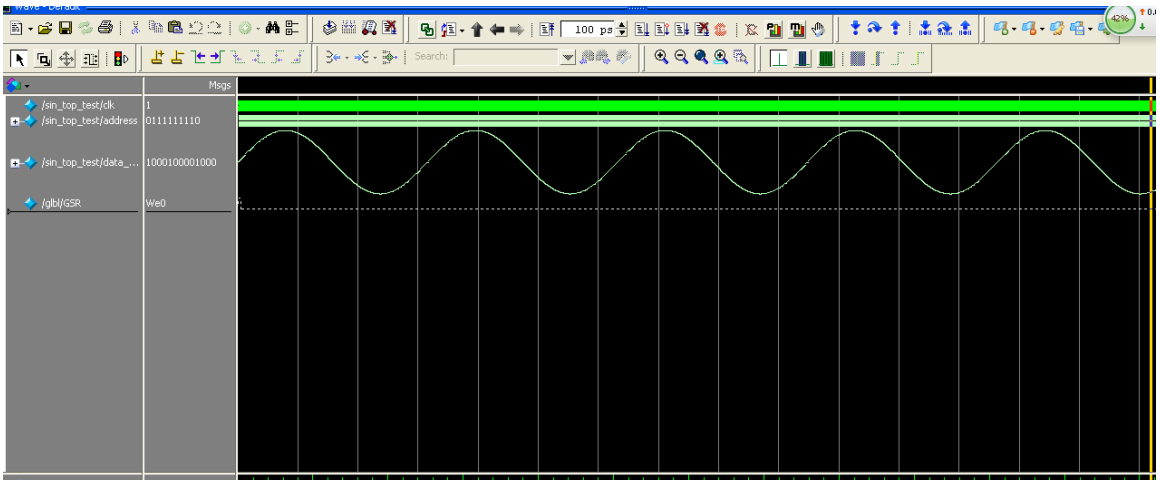
只需在设置器件这里将仿真器设置为 modelsim 就可以了。

然后就可以直接仿真了。

在 modelsim 里面



将 out 的 format 的格式改为 analog。这样一会显示的波形就会以模拟的形式显示了。



仿真后，标准的正弦波形就出来了。是不是很酷啊。。哈哈哈

这里采用的 modelsim 仿真，而不是用的是 ISE 自带的仿真，因为 ISE 的波形不能以模拟波形显示。

这样产生的正弦波，方法比较简单，但是不能调节，调节的话，就要重新生成 rom 的值。而且还伴随着误差。可以增大位数，和减小采样间隔时间来减小误差。