

**《综合电子设计实践》**

**实验报告**

**FPGA实验　函数信号发生器**

**姓 名： 施赛佳 学 号： 22014110**

# 一、实验准备

## 1.实验目的与要求

1）基本要求

(1)频率和周期设定功能

a.输出信号为正弦波或方波，频率范围为1Hz~10MHz;

B.输出信号相对误差的绝对值不大于10-4

1. 时间间隔测量功能

a.输出信号为方波，频率范围为100Hz~1MHz;

b.两路输出信号的时间间隔的范围为9.1us~100ms;

c.相对误差的绝对值不大于10-2

(3)从设定到信号输出的时间不大于2s，输出结果稳定，并能自动显示设定单位。

2）发挥部分

(1)所产生的正弦或方波信号频率范围为1Hz~100MHz，其他要求同基本要求（1）、（3）

(2)所产生的正弦或方波信号的幅度可调100mV~5V，其他要求同基本要求（1）和（3）

(3)其他（例如进一步降低被测信号电压的幅度等）

## 2.实验器材及软件环境

1）器材

Cyclone IV FPGA实验开发板

8591AD/DA转换模块

2）软件环境

Quartus II 13.0

Keil uVision4

MATLAB R2016a

STC-ISP

**3.实验原理及实现方案**

1）方波信号的产生

Cyclone IV FPGA 开发板自身晶振为48MHz，为得到1Hz~10MHz的方波，采用分频的方法来实现。

2）频率的输入及显示

由于红外线遥控器输入和按键输入都存在一定的缺陷，故采用两者结合的方式来输入期望的函数发生器的输出频率。函数发生器的输出频率用四位数字表示，并在四位数码管上显示出来。其中最高位表示10的指数幂，低三位为三位有效数字，其实际的结果为两者的乘积。输入时用按键切换输入的数字位，并用遥控器对当前数字位进行赋值。

3）正弦波的产生

采用查表法产生正弦波，即将正弦函数的函数值写入ROM表中对应的位置，通过之前产生的方波对ROM的地址进行周期性扫描，用8591AD/DA转换芯片将ROM地址对应的数字信号读取并进行DA转换得到正弦波，正弦波的频率变换可以通过切换方波的频率实现。

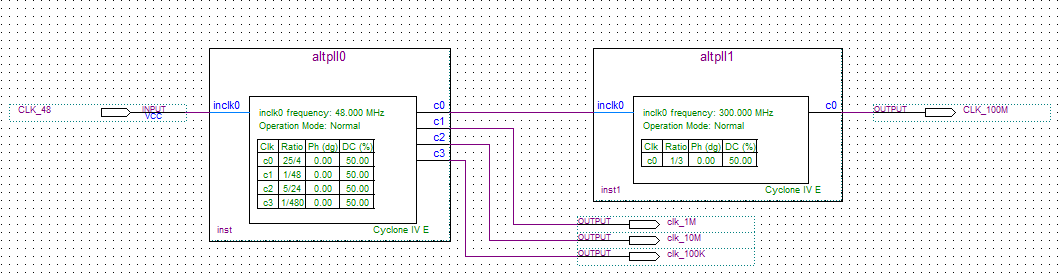
**二、实验验电路设计及代码**

**1.顶层设计思路**

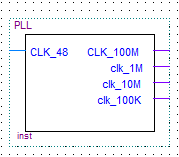


在本次实验设计中，顶层文件采用原理图文件以直观的显示各个模块之间的关系，模块采用Verilog语句进行硬件描述，增强其通用性和可移植性。

## 2.锁相环模块



在顶层文件中的例化形式



该模块采用两个锁相环，先倍频，再分频，以提高得到倍频结果的准确度。

CLK\_48为系统给予的48MHz晶振，CLK\_100M为计数器模块提供标准的时钟信号，其他的输出引起到测试以及为其他模块提供标准时钟信号的作用。

## 3.红外线输入模块

Verilog描述语句：

module ir\_module(

//Clock Input:48M

input clk\_1m,

input ir,

//Return Value

output reg [7:0]Code

);

reg [2:0]IR\_reg;

initial IR\_reg = 3'b0;

always @ (posedge clk\_1m) //1us

begin

IR\_reg[0] <= ir;

IR\_reg[1] <= IR\_reg[0];

IR\_reg[2] <= IR\_reg[1];

end

wire IR\_pos = (IR\_reg[0]==1'b1) & (IR\_reg[1]==1'b0);

wire IR\_pos2= (IR\_reg[1]==1'b1) & (IR\_reg[2]==1'b0);

wire IR\_neg = (IR\_reg[0]==1'b0) & (IR\_reg[1]==1'b1);

wire IR\_neg2= (IR\_reg[1]==1'b0) & (IR\_reg[2]==1'b1);

//状态

parameter ST\_START\_L = 3'b000, ST\_CODE\_P = 3'b001 , ST\_VALUE\_P = 3'b010;

parameter ST\_START\_H = 3'b011, ST\_CODE\_N = 3'b100 , ST\_VALUE\_N = 3'b101;

//宽度

parameter START\_H = 16'd4096; //us

parameter START\_L = 16'd8192; //us

parameter CODE\_0 = 16'd512 + 16'd512 ; //us

parameter CODE\_1 = 16'd1536 + 16'd512 ; //us

reg [2:0]state;

initial state = ST\_START\_L;

reg [15:0]cnt\_h;

initial cnt\_h = 16'b0;

reg [15:0]cnt\_l;

initial cnt\_l = 16'b0;

reg [31:0]T\_Value;

initial T\_Value = 32'b0;

reg [31:0]IR\_Value;

initial IR\_Value = 32'b0;

reg [15:0]cnt\_val;

initial cnt\_val = 16'b0;

reg Flag\_LVL;

initial Flag\_LVL = 1'b0;

reg Flag\_HVL;

initial Flag\_HVL = 1'b0;

always @ (posedge clk\_1m or posedge ir)

begin

if(ir) //高电平复位

cnt\_l <= 16'b0;

else if(cnt\_l[15] & cnt\_l[10]) //低计数溢出复位

cnt\_l <= 16'b0;

else

cnt\_l <= cnt\_l + 1'b1; //低电平计数

end

always @ (negedge clk\_1m)

begin

if(cnt\_l == START\_L)

Flag\_LVL <= 1'b1;

else if(IR\_pos2)

Flag\_LVL <= 1'b0;

end

always @ (posedge clk\_1m or negedge ir)

begin

if(!ir) //低电平复位

cnt\_h <= 16'b0;

else if(cnt\_h[15] & cnt\_h[10]) //高计数溢出复位

cnt\_h <= 16'b0;

else

cnt\_h <= cnt\_h + 1'b1; //高电平计数

end

always @ (negedge clk\_1m)

begin

if(cnt\_h == START\_H)

Flag\_HVL <=1;

else if(IR\_neg2)

Flag\_HVL <= 1'b0;

end

reg [15:0]IR\_code;

always @ (posedge clk\_1m or posedge IR\_neg)

begin

if(IR\_neg)

begin

cnt\_val <= 16'b0;

end

else if(state == ST\_CODE\_P)

begin

if(cnt\_val == CODE\_0)

begin

IR\_code <= CODE\_0;

cnt\_val <= cnt\_val + 1'b1;

end

else if(cnt\_val == CODE\_1)

begin

IR\_code <= CODE\_1;

cnt\_val <= cnt\_val + 1'b1;

end

else

cnt\_val <= cnt\_val + 1'b1;

end

end

wire fault = cnt\_h[15] | cnt\_l[15]; //错误

reg [5:0]cnt\_num;

initial cnt\_num = 6'b0;

always @ (posedge clk\_1m) //1us

begin

case(state)

ST\_START\_L:

begin

cnt\_num <= 6'b0;

if((IR\_pos == 1'b1) & (Flag\_LVL==1'b1))

begin

state <= ST\_START\_H;

end

else if(fault)

state <= ST\_START\_L;

end

ST\_START\_H :

begin

cnt\_num <= 6'b0;

if((IR\_neg == 1'b1) & (Flag\_HVL==1'b1))

begin

state <= ST\_CODE\_P;

end

else if(fault)

state <= ST\_START\_L;

end

ST\_CODE\_P :

begin

if((IR\_neg)&(IR\_code == CODE\_1))

begin

cnt\_num = cnt\_num + 1'b1;

IR\_Value <= {IR\_Value[30:0],1'b1};

end

else if((IR\_neg)&(IR\_code == CODE\_0))

begin

cnt\_num = cnt\_num + 1'b1;

IR\_Value <= {IR\_Value[30:0],1'b0};

end

else if(cnt\_num==6'd32)

begin

cnt\_num <= 6'b0;

T\_Value <= IR\_Value;

state <= ST\_START\_L;

Code <= {IR\_Value[8],IR\_Value[9],IR\_Value[10],IR\_Value[11],IR\_Value[12],IR\_Value[13],IR\_Value[14],IR\_Value[15]};

end

end

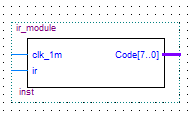
default : state <= ST\_START\_L;

endcase

end

endmodule

在顶层文件中的例化形式



clk\_1m为1MHz的时钟信号，ir为接受来自遥控器的红外线信号，Code[7..0]为输出的8位红外遥控器键码，该模块的作用为将红外信号转换为键码信号，作为后续模块根据按键情况编辑操作的判断条件。

## 4.输出为1Hz的分频器

verilog描述语句

module clk\_hz(clk\_in,clk\_out);

input clk\_in;

output clk\_out;

reg clk\_out;

reg [31:0] cnt;

parameter TIME=1\_000\_000;

always @(posedge clk\_in)

begin

if(cnt == TIME/2)

begin

cnt<= 32'b0;

clk\_out = !clk\_out;

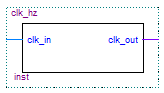
end

else cnt <= cnt + 1'b1;

end

endmodule

在顶层文件中的例化形式



将输入的1MHz信号转换为1Hz

## 5.数码管显示模块

Verilog描述语句

module dt\_module(

//Clock Input:48M

input clk,

input [3:0]num1,

input [3:0]num2,

input [3:0]num3,

input [3:0]num4,

output reg [3:0]ds\_en,

//Digital tube

output reg [6:0]ds\_reg

);

parameter [6:0]NUM\_0=7'b0111111;

parameter [6:0]NUM\_1=7'b0000110;

parameter [6:0]NUM\_2=7'b1011011;

parameter [6:0]NUM\_3=7'b1001111;

parameter [6:0]NUM\_4=7'b1100110;

parameter [6:0]NUM\_5=7'b1101101;

parameter [6:0]NUM\_6=7'b1111101;

parameter [6:0]NUM\_7=7'b0000111;

parameter [6:0]NUM\_8=7'b1111111;

parameter [6:0]NUM\_9=7'b1101111;

parameter [6:0]NUM\_A=7'b1110111;

parameter [6:0]NUM\_B=7'b1111100;

parameter [6:0]NUM\_C=7'b1011000;

parameter [6:0]NUM\_D=7'b1011110;

parameter [6:0]NUM\_E=7'b1111001;

parameter [6:0]NUM\_F=7'b1110001;

parameter [6:0]NUM\_BLK=7'b0000000;

parameter [3:0]EN\_1=4'b1110;

parameter [3:0]EN\_2=4'b1101;

parameter [3:0]EN\_3=4'b1011;

parameter [3:0]EN\_4=4'b0111;

parameter [3:0]EN\_A=4'b0000;

reg [31:0]cnt;

always @(posedge clk)

begin

cnt = cnt + 1'b1;

end

always @(posedge clk)

begin

if(cnt[16:15] == 2'b00)

case (num1)

4'h0 : begin ds\_reg = NUM\_0; ds\_en=EN\_1;end

4'h1 : begin ds\_reg = NUM\_1; ds\_en=EN\_1;end

4'h2 : begin ds\_reg = NUM\_2; ds\_en=EN\_1;end

4'h3 : begin ds\_reg = NUM\_3; ds\_en=EN\_1;end

4'h4 : begin ds\_reg = NUM\_4; ds\_en=EN\_1;end

4'h5 : begin ds\_reg = NUM\_5; ds\_en=EN\_1;end

4'h6 : begin ds\_reg = NUM\_6; ds\_en=EN\_1;end

4'h7 : begin ds\_reg = NUM\_7; ds\_en=EN\_1;end

4'h8 : begin ds\_reg = NUM\_8; ds\_en=EN\_1;end

4'h9 : begin ds\_reg = NUM\_9; ds\_en=EN\_1;end

4'ha : begin ds\_reg = NUM\_A; ds\_en=EN\_1;end

4'hb : begin ds\_reg = NUM\_B; ds\_en=EN\_1;end

4'hc : begin ds\_reg = NUM\_C; ds\_en=EN\_1;end

4'hd : begin ds\_reg = NUM\_D; ds\_en=EN\_1;end

4'he : begin ds\_reg = NUM\_E; ds\_en=EN\_1;end

4'hf : begin ds\_reg = NUM\_F; ds\_en=EN\_1;end

default: begin ds\_reg = NUM\_BLK; ds\_en=EN\_1;end

endcase

if(cnt[16:15] == 2'b01)

case (num2)

4'h0 : begin ds\_reg = NUM\_0; ds\_en=EN\_2;end

4'h1 : begin ds\_reg = NUM\_1; ds\_en=EN\_2;end

4'h2 : begin ds\_reg = NUM\_2; ds\_en=EN\_2;end

4'h3 : begin ds\_reg = NUM\_3; ds\_en=EN\_2;end

4'h4 : begin ds\_reg = NUM\_4; ds\_en=EN\_2;end

4'h5 : begin ds\_reg = NUM\_5; ds\_en=EN\_2;end

4'h6 : begin ds\_reg = NUM\_6; ds\_en=EN\_2;end

4'h7 : begin ds\_reg = NUM\_7; ds\_en=EN\_2;end

4'h8 : begin ds\_reg = NUM\_8; ds\_en=EN\_2;end

4'h9 : begin ds\_reg = NUM\_9; ds\_en=EN\_2;end

4'ha : begin ds\_reg = NUM\_A; ds\_en=EN\_2;end

4'hb : begin ds\_reg = NUM\_B; ds\_en=EN\_2;end

4'hc : begin ds\_reg = NUM\_C; ds\_en=EN\_2;end

4'hd : begin ds\_reg = NUM\_D; ds\_en=EN\_2;end

4'he : begin ds\_reg = NUM\_E; ds\_en=EN\_2;end

4'hf : begin ds\_reg = NUM\_F; ds\_en=EN\_2;end

default: begin ds\_reg = NUM\_BLK; ds\_en=EN\_2;end

endcase

if(cnt[16:15] == 2'b10)

case (num3)

4'h0 : begin ds\_reg = NUM\_0; ds\_en=EN\_3;end

4'h1 : begin ds\_reg = NUM\_1; ds\_en=EN\_3;end

4'h2 : begin ds\_reg = NUM\_2; ds\_en=EN\_3;end

4'h3 : begin ds\_reg = NUM\_3; ds\_en=EN\_3;end

4'h4 : begin ds\_reg = NUM\_4; ds\_en=EN\_3;end

4'h5 : begin ds\_reg = NUM\_5; ds\_en=EN\_3;end

4'h6 : begin ds\_reg = NUM\_6; ds\_en=EN\_3;end

4'h7 : begin ds\_reg = NUM\_7; ds\_en=EN\_3;end

4'h8 : begin ds\_reg = NUM\_8; ds\_en=EN\_3;end

4'h9 : begin ds\_reg = NUM\_9; ds\_en=EN\_3;end

4'ha : begin ds\_reg = NUM\_A; ds\_en=EN\_3;end

4'hb : begin ds\_reg = NUM\_B; ds\_en=EN\_3;end

4'hc : begin ds\_reg = NUM\_C; ds\_en=EN\_3;end

4'hd : begin ds\_reg = NUM\_D; ds\_en=EN\_3;end

4'he : begin ds\_reg = NUM\_E; ds\_en=EN\_3;end

4'hf : begin ds\_reg = NUM\_F; ds\_en=EN\_3;end

default: begin ds\_reg = NUM\_BLK; ds\_en=EN\_3;end

endcase

if(cnt[16:15] == 2'b11)

case (num4)

4'h0 : begin ds\_reg = NUM\_0; ds\_en=EN\_4;end

4'h1 : begin ds\_reg = NUM\_1; ds\_en=EN\_4;end

4'h2 : begin ds\_reg = NUM\_2; ds\_en=EN\_4;end

4'h3 : begin ds\_reg = NUM\_3; ds\_en=EN\_4;end

4'h4 : begin ds\_reg = NUM\_4; ds\_en=EN\_4;end

4'h5 : begin ds\_reg = NUM\_5; ds\_en=EN\_4;end

4'h6 : begin ds\_reg = NUM\_6; ds\_en=EN\_4;end

4'h7 : begin ds\_reg = NUM\_7; ds\_en=EN\_4;end

4'h8 : begin ds\_reg = NUM\_8; ds\_en=EN\_4;end

4'h9 : begin ds\_reg = NUM\_9; ds\_en=EN\_4;end

4'ha : begin ds\_reg = NUM\_A; ds\_en=EN\_4;end

4'hb : begin ds\_reg = NUM\_B; ds\_en=EN\_4;end

4'hc : begin ds\_reg = NUM\_C; ds\_en=EN\_4;end

4'hd : begin ds\_reg = NUM\_D; ds\_en=EN\_4;end

4'he : begin ds\_reg = NUM\_E; ds\_en=EN\_4;end

4'hf : begin ds\_reg = NUM\_F; ds\_en=EN\_4;end

default: begin ds\_reg = NUM\_BLK; ds\_en=EN\_4;end

endcase

end

endmodule

该模块的作用是对数码管的显示进行定义，是其显示的数字为输入的数字。由于该模块是在count模块中使用verilog语句进行调用，故没有进行例化。

## 6.count分频与显示模块

该模块为整个工程的核心部分，其功能是根据红外线信号对4位数据进行赋值，根据按键信号切换数据位，最终根据数据位的录入情况对100MHz的时钟信号进行分频得到与数字位显示情况对应频率的方波。

Verliog描述语句

module count(CLK,KEY1,KEY2,ir\_code,clk\_in,clk\_hz,rst\_n,clk\_out,//DS\_C,DS\_D,DS\_G,DS\_DP);

DS\_A,DS\_B,DS\_E,DS\_F,

DS\_EN1,DS\_EN2,DS\_EN3,DS\_EN4,

DS\_C,DS\_D,DS\_G,DS\_DP);

input KEY1;

input KEY2;

input CLK;

input clk\_in ;

input clk\_hz ;

input ir\_code;

input rst\_n; //PIN 100 全局低电平复位

output clk\_out;//DS\_C,DS\_D,DS\_G,DS\_DP;

output DS\_A,DS\_B,DS\_E,DS\_F;

output DS\_EN1,DS\_EN2,DS\_EN3,DS\_EN4;

output DS\_C,DS\_D,DS\_G,DS\_DP;

parameter TIME=32'd100\_000\_000;

wire [7:0] ir\_code;

reg clk\_out;

reg [31:0] cnt;

reg[31:0] tmp1;

reg[31:0] tmp2;

reg [31:0] num;

reg [1:0] num\_4=2'b00;

reg [3:0] code [3:0];

/\*

reg [3:0]led\_reg;

wire [3:0]led ;

assign {DS\_D,DS\_C,DS\_G,DS\_DP} = led;

\*/

always @ (posedge clk\_hz) //改为1HZ,指示灯

begin

/\*if(KEY1 ==1'b1) //改为按键

begin

code[0]=4'h0;

code[1]=4'h0;

code[2]=4'h0;

code[3]=4'h1;//排除全零

end

else\*/

if (KEY2==1'b0)

//else if(ir\_code==8'h40) //改为按键

begin

num\_4<=num\_4+1'b1;

/\*

case(num\_4)

2'b00:led\_reg=4'b0111;

2'b01:led\_reg=4'b1011;

2'b10:led\_reg=4'b1101;

2'b11:led\_reg=4'b1110;

endcase

\*/

case(ir\_code)

8'h16:code[num\_4]=4'h0;

8'h0C:code[num\_4]=4'h1;

8'h18:code[num\_4]=4'h2;

8'h5E:code[num\_4]=4'h3;

8'h08:code[num\_4]=4'h4;

8'h1C:code[num\_4]=4'h5;

8'h5A:code[num\_4]=4'h6;

8'h42:code[num\_4]=4'h7;

8'h52:code[num\_4]=4'h8;

8'h4A:code[num\_4]=4'h9;

endcase

end

tmp1=TIME+TIME;

case (code[3])

4'h0:begin tmp2<=code[2]\*100+code[1]\*10+code[0]-1;

num=tmp1/tmp2;end

4'h1:begin tmp2<=(code[2]\*100+code[1]\*10+code[0])\*10-1;

num=tmp1/tmp2;end

4'h2:begin tmp2<=(code[2]\*100+code[1]\*10+code[0])\*100-1;

num=tmp1/tmp2;end

4'h3:begin tmp2<=(code[2]\*100+code[1]\*10+code[0])\*1\_000-1;

num=tmp1/tmp2;end

4'h4:begin tmp2<=(code[2]\*100+code[1]\*10+code[0])\*10\_000-1;

num=tmp1/tmp2;end

4'h5:begin tmp2<=(code[2]\*100+code[1]\*10+code[0])\*100\_000-1;

num=tmp1/tmp2;end

/\*

4'h6:begin tmp2<=(code[2]\*100+code[1]\*10+code[0])\*1\_000\_000-1;

num=tmp1/tmp2;end

4'h7:begin tmp2<=(code[2]\*100+code[1]\*10+code[0])\*10\_000\_000-1;

num=tmp1/tmp2;end

\*/

default:begin tmp2<=code[2]\*100+code[1]\*10+code[0]-1;

num=tmp1/tmp2;end

endcase

end

//assign led=led\_reg;

always @ (posedge clk\_in or negedge rst\_n)

begin

if(1'b0==rst\_n)

cnt<=32'd0;

else if (cnt==num/4)

cnt<=32'd0;

else

cnt<=cnt+1'b1;

end

always @ (posedge clk\_in or negedge rst\_n)

begin

if (1'b0==rst\_n)

clk\_out<=1'b1;

else if (cnt==num/4)

clk\_out<=~clk\_out ;

end

wire [6:0]ds\_reg;

wire [3:0]ds\_en;

assign {DS\_G,DS\_F,DS\_E,DS\_D,DS\_C,DS\_B,DS\_A} = ds\_reg ;

assign {DS\_EN1,DS\_EN2,DS\_EN3,DS\_EN4} = ds\_en;

reg [3:0]num\_dt[3:0];

dt\_module dt\_ct(

.clk(CLK),

.num1(num\_dt[0]),

.num2(num\_dt[1]),

.num3(num\_dt[2]),

.num4(num\_dt[3]),

.ds\_en(ds\_en),

.ds\_reg(ds\_reg)

);

reg [31:0]cnt1;

always @(posedge CLK)

begin

cnt1 <= cnt1 + 1;

num\_dt[0] = code[0];

num\_dt[1] = code[1];

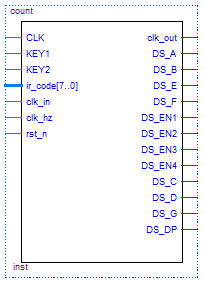
num\_dt[2] = code[2];

num\_dt[3] = code[3];

end

Endmodule

在顶层文件中的例化形式



clk\_in为100MHz时钟信号，clk\_out为分频之后的输出时钟信号

## 7.正弦波转换模块

正弦波转换采用函数查表法构建，即将正弦函数对应的数值存入ROM中，通过地址线循环读取ROM模块中存取的数值，并发送给8591进行DA转换得到对应正弦函数波形。

### 1）地址扫描子模块

Verilog描述语句

module address(CLK,add);

input CLK;

output add;

reg [7:0] add;

always @ (posedge CLK)

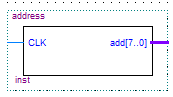
begin

add[7:0]<=add[7:0]+1'b1;

end

endmodule

模块实例化



### 2）ROM模块

受硬件条件限制，正弦波无法生成负电压，所以所取正弦波的每一点的电压值均大于0

在matlab中输入如下运算代码：

t=linspace(0,2\*pi,256);

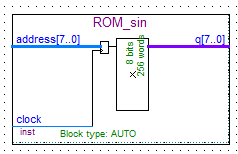
y=100\*(sin(t)+1);

将输出的y值取整得到如下结果：

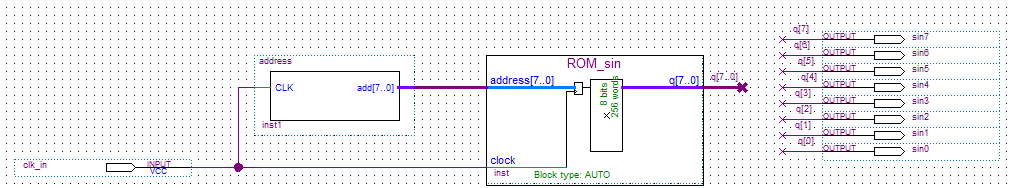
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | +000 | +001 | +010 | +011 | +100 | +101 | +110 | +111 |
| 000000000 | 100 | 102 | 105 | 107 | 110 | 112 | 115 | 117 |
| 000001000 | 120 | 122 | 124 | 127 | 129 | 131 | 134 | 136 |
| 000010000 | 138 | 141 | 143 | 145 | 147 | 149 | 152 | 154 |
| 000011000 | 156 | 158 | 160 | 162 | 164 | 166 | 167 | 169 |
| 000100000 | 171 | 173 | 174 | 176 | 178 | 179 | 181 | 182 |
| 000101000 | 183 | 185 | 186 | 187 | 188 | 190 | 191 | 192 |
| 000110000 | 193 | 193 | 194 | 195 | 196 | 197 | 197 | 198 |
| 000111000 | 198 | 199 | 199 | 199 | 200 | 200 | 200 | 200 |
| 001000000 | 200 | 200 | 200 | 200 | 199 | 199 | 199 | 198 |
| 001001000 | 198 | 197 | 197 | 196 | 195 | 195 | 194 | 193 |
| 001010000 | 192 | 191 | 190 | 189 | 188 | 187 | 185 | 184 |
| 001011000 | 183 | 181 | 180 | 178 | 177 | 175 | 173 | 172 |
| 001100000 | 170 | 168 | 166 | 165 | 163 | 161 | 159 | 157 |
| 001101000 | 155 | 153 | 151 | 148 | 146 | 144 | 142 | 140 |
| 001110000 | 137 | 135 | 133 | 130 | 128 | 126 | 123 | 121 |
| 001111000 | 118 | 116 | 114 | 111 | 109 | 106 | 104 | 101 |
| 010000000 | 99 | 96 | 94 | 91 | 89 | 86 | 84 | 82 |
| 010001000 | 79 | 77 | 74 | 72 | 70 | 67 | 65 | 63 |
| 010010000 | 60 | 58 | 56 | 54 | 52 | 49 | 47 | 45 |
| 010011000 | 43 | 41 | 39 | 37 | 35 | 34 | 32 | 30 |
| 010100000 | 28 | 27 | 25 | 23 | 22 | 20 | 19 | 17 |
| 010101000 | 16 | 15 | 13 | 12 | 11 | 10 | 9 | 8 |
| 010110000 | 7 | 6 | 5 | 5 | 4 | 3 | 3 | 2 |
| 010111000 | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 011000000 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 2 |
| 011001000 | 2 | 3 | 3 | 4 | 5 | 6 | 7 | 7 |
| 011010000 | 8 | 9 | 10 | 12 | 13 | 14 | 15 | 17 |
| 011011000 | 18 | 19 | 21 | 22 | 24 | 26 | 27 | 29 |
| 011100000 | 31 | 33 | 34 | 36 | 38 | 40 | 42 | 44 |
| 011101000 | 46 | 48 | 51 | 53 | 55 | 57 | 59 | 62 |
| 011110000 | 64 | 66 | 69 | 71 | 73 | 76 | 78 | 80 |
| 011111000 | 83 | 85 | 88 | 90 | 93 | 95 | 98 | 100 |

将该组数据录入ROM模块即可完成对ROM模块的描述

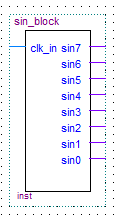
ROM模块的例化



### 3）正弦波转换模块及其在顶层文件中的例化

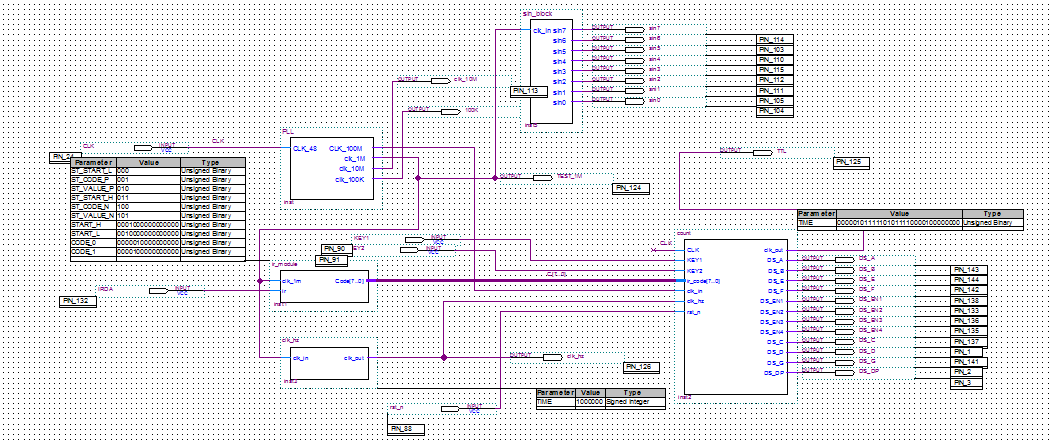


例化



Sin0~sin7为输出的8位2进制数，用来传输电压的幅值。

## 8.顶层原理图文件



# 三、实验总结

经过这次FPGA设计实践，我基本掌握了Verilog语言的编程。中期遇到了相当多的问题，还好有学长相助，顺利地把问题解决了。