数字逻辑实验报告

PB10000603 李博杰

实验一 四位海明校验码的逻辑设计实验

海明校验是一种多重校验。能纠正一位错误，发现二位同时错误。在计算机应用于高可靠性存贮设备，如高速缓存、主存微程序内存。也有用在外存，如磁带、磁盘等处。

一、实验目的

掌握海明校验的编码原理以及设计、调试方法，巩固提高组合逻辑知识，培养实际动手能力。

二、实验要求

①设计信息位为 4 位的内存的海明校验逻辑电路，在读内存储器时，具有一位出错报错和纠正一位错误的功能。

②为了验证其正确性，在读出信息的通路上，要串入造错用逻辑，位数自定。

三、实验原理

①若有 K 位信息位，需要最少校验位元 r，必须满足 2r>=K+ r +1。

②将信息位和校验位混合统一编码，分别用十进制和二进制表示。十进制编号等于 2n（n=0，1，2… …）的位为校验位，依次为 b1，b2… …bj（j=1——r），其余为信息位，依次为 a1，a2,… …ai（i=1——k）。

③校验位 b1 由二进制统一编号，最低位（也就是第一位）为“1”的所有位的信息按位加，我们记作 b=Σ &ai(2°)，b2 由二进制统一编号，第二位为“1”的所有位的信息，记作 n2=Σ +ai([2])，依次类推则 bi=Σ +ai(2j-1)

④校验方程为 sj=bj+Σ +ai（2j-1），其中 bj，ai 为 bjai 写入内存后的读出

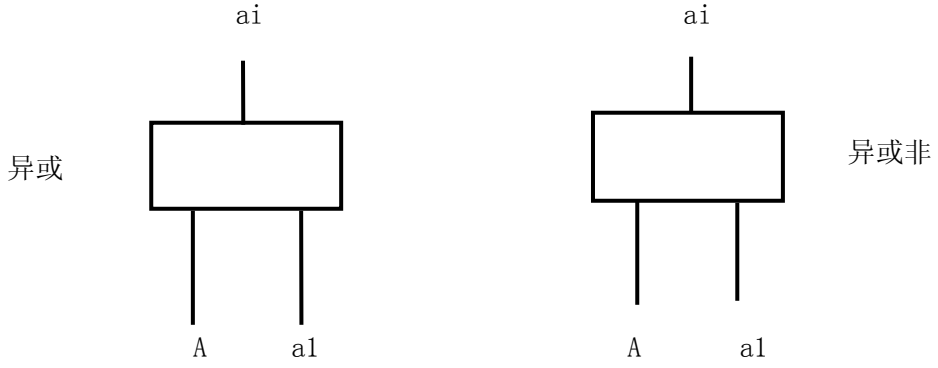
由 sj——s1s2 组排二进制：

若 sj——s1s2=0 则不发生错（指单错）

若 sj——s1s2≠0 则组排成的二进制与前面用二进制统一编号相等的那位出错。

⑤用异或门或异或非门纠错

用某位的出错信号即译码器对应的输出控制二输入之一，达到取反纠错目的，造错原理类同。



出错：A=H　　al=ai　　A=L　ai=al

常态：A=L　　al=ai　　A=H　Ai=al

四、逻辑设计

设 4 个数据位为 d0,d1,d2,d3，根据海明码原理，产生 3 个校验位 r0,r1,r2。用 b7～b1 对应表示：

d3 d2 d1 r2 d0 r1 r0

b7 b6 b5 b4 b3 b2 b1

校验位的取值，就是它所能校验的数据位的异或

b1（r0）为 b3,b5,b7 的异或

b2（r1）为 b3,b6,b7 的异或

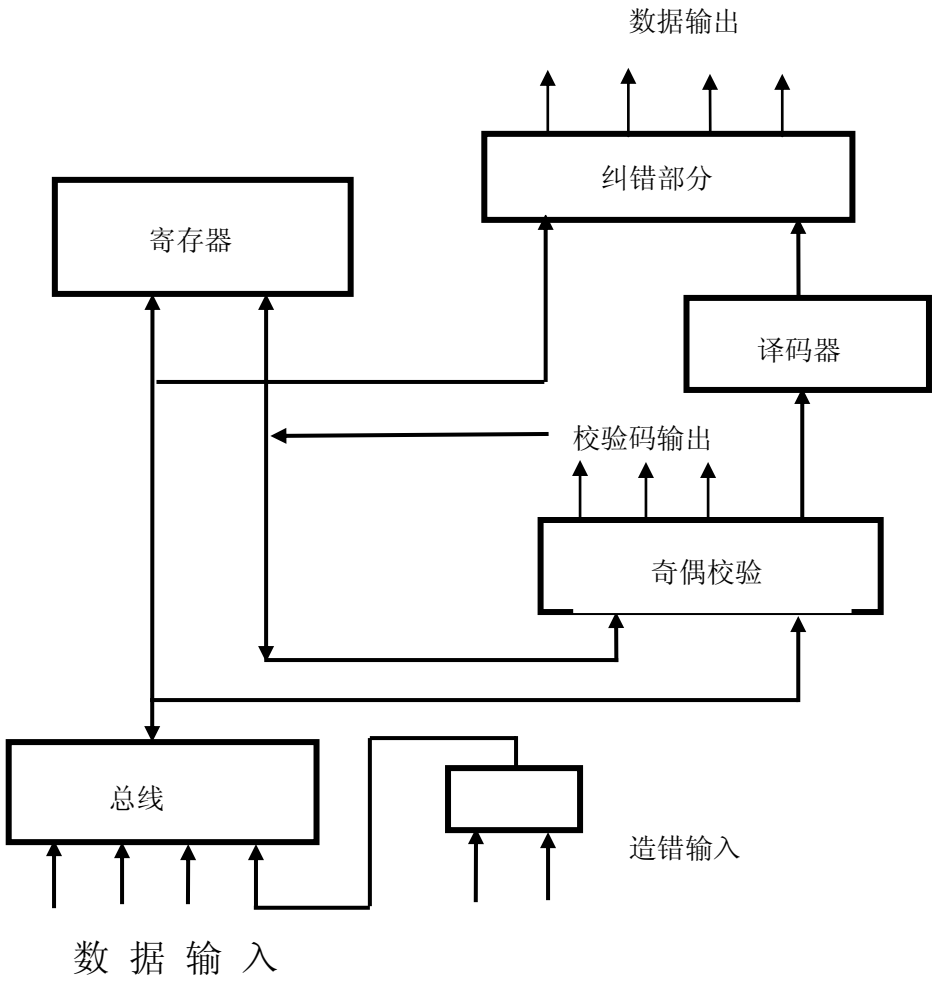
b4（r2）为 b5,b6,b7 的异或

海明 v 传送到接受方后，将上三式的右边（b1,b2,b4）的逻辑表达式分别异或上左边的值就得到了校验方程。此处采用偶校验：

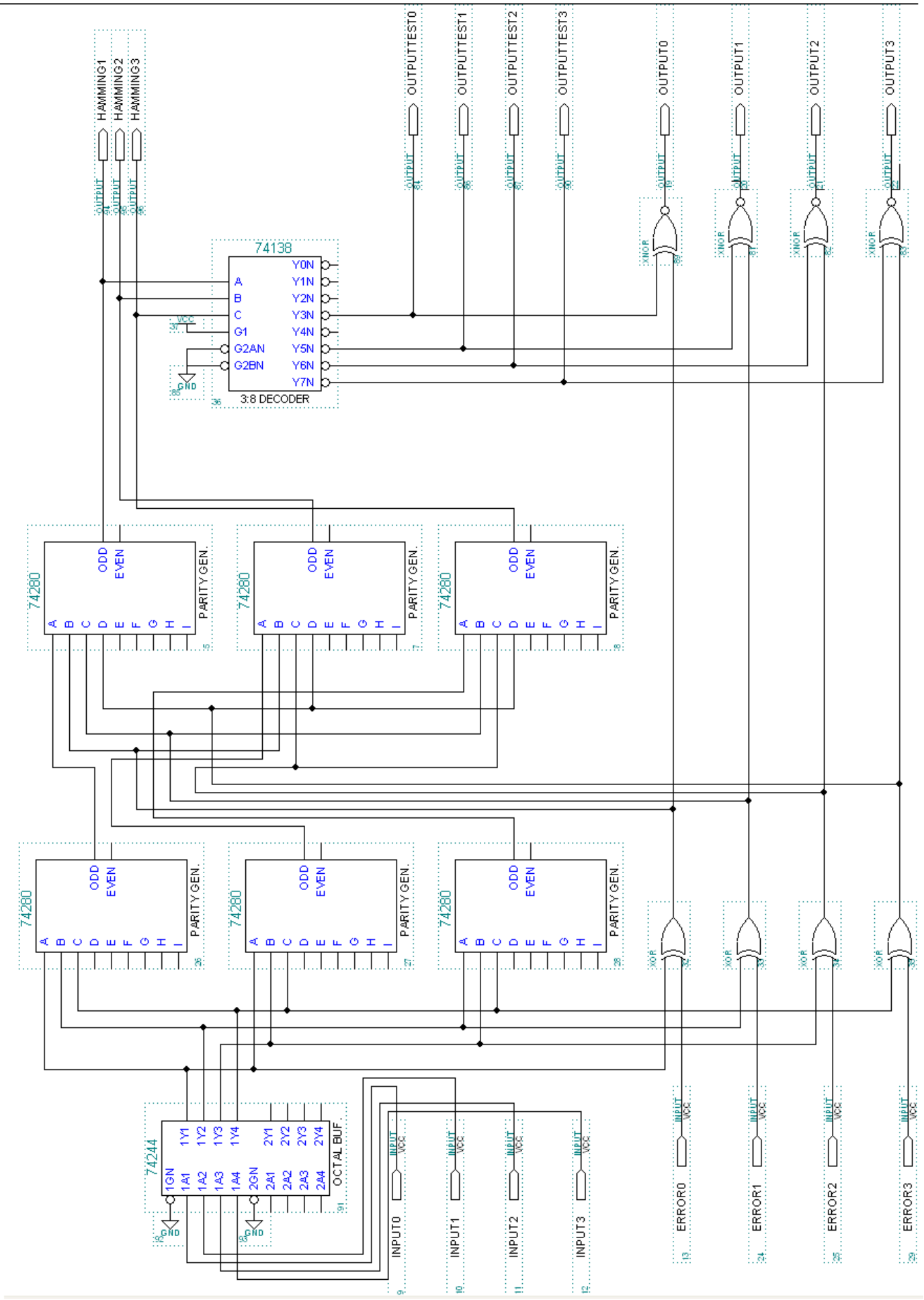
G1 = b1 b3 b5 b7 的异或

G2 = b2 b3 b6 b7 的异或
G3 = b4 b5 b6 b7 的异或
无错误情况应为 G1G2G3=000
若 G1G2G3 为 011 则为 r0 错
若为 101 则为 r1 错
若为 110 则为 r2 错
若为 111 则为 r3 错
否则错误数不止一位，是非法情况。

下面是电路的逻辑结构设计



五、实验电路及仿真结果


HAMMING1
HAMMING2
HAMMING3
OUTPUTTEST0
OUTPUTTEST1
OUTPUTTEST2
OUTPUTTEST3
OUTPUT0
OUTPUT1
OUTPUT2
OUTPUT3
74138
A
B
C
G1
G2AN
G2BN
Y0N
Y1N
Y2N
Y3N
Y4N
Y5N
Y6N
Y7N
3:8 DECODER
74280
ODD
EVEN
PARITY GEN.
XNOR
XOR
74244
1G
2G
OCTAL BUF.
INPUT0
INPUT1
INPUT2
INPUT3
ERROR0
ERROR1
ERROR2
ERROR3
VCC
GND

仿真说明：INPUT3～0 是四个输入位，ERROR3～0 是每个输入位的造错标志（1 表示造错，0 表示无措），OUTPUT3～0 是经过纠错后的输出，HAMMING1～3 是经过海明码校验后的监督关系式满足情况（纠错码），OUTPUTTEST 是经过海明校验后识别出的发生错误的位。

仿真结果表明，在无错误或有一位错误的情况下，原始数据输入与输出相同，即海明码校验成功。

仿真结果存在毛刺，是因为两个造错位同时发生改变。由于不同器件中信号传播的速度不同，会在短暂的时间内出现电路输出不稳定的情况，但很快能回到稳态。

六、Verilog HDL 及仿真结果

设计思路：

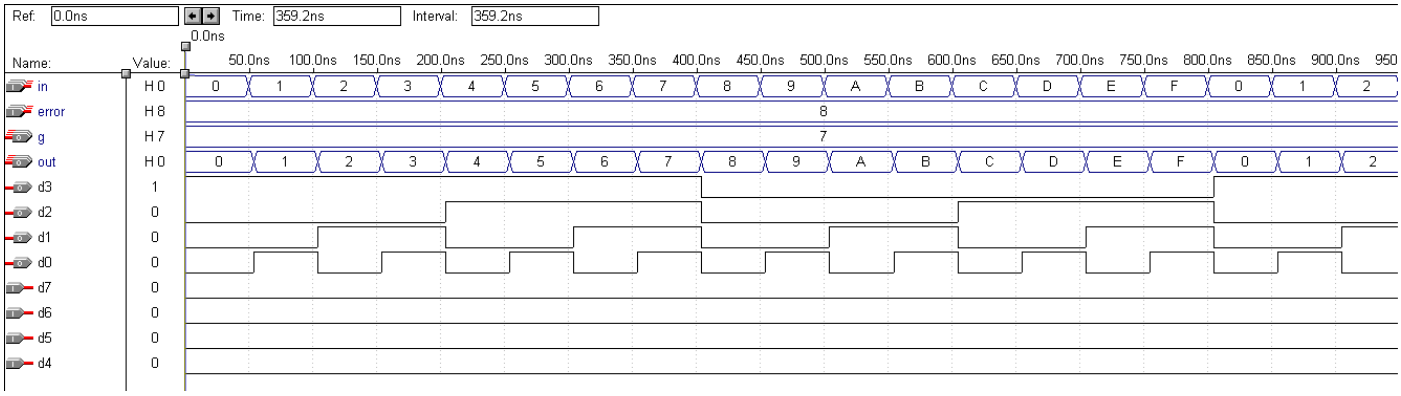
1. 根据原始数据求出海明码校验位（c）
2. 得出造错后的数据（d）
3. 将海明码校验位和造错后的数据送入监督关系式，求出纠错码（g）
4. 根据纠错码恢复原始数据（out）

源代码：

```
module hamming1(in,error,d,g,out);
input [3:0]in;
input [3:0]error;
output [3:1]g; //test
output [3:0]out;
wire [2:0]c; //checkbit
wire [7:1]b; //temp
output [3:0]d; //corrupted data

assign c[0]=in[0]^in[1]^in[3];
assign c[1]=in[0]^in[2]^in[3];
assign c[2]=in[1]^in[2]^in[3];
```

```
assign d[3]=in[3]^error[3];
assign d[2]=in[2]^error[2];
assign d[1]=in[1]^error[1];
assign d[0]=in[0]^error[0];

assign b[7]=d[3];
assign b[6]=d[2];
assign b[5]=d[1];
assign b[4]=c[2];
assign b[3]=d[0];
assign b[2]=c[1];
assign b[1]=c[0];

assign g[1]=b[1]^b[3]^b[5]^b[7];
assign g[2]=b[2]^b[3]^b[6]^b[7];
assign g[3]=b[4]^b[5]^b[6]^b[7];

assign out[0]=d[0]^(~g[3]&g[2]&g[1]);
assign out[1]=d[1]^(g[3]&~g[2]&g[1]);
assign out[2]=d[2]^(g[3]&g[2]&~g[1]);
assign out[3]=d[3]^(g[3]&g[2]&g[1]);

endmodule
```

仿真结果：



仿真说明：仿真图中 in 是输入（四位），error 是造错（四位），out 是输出（四位）。

七、实验总结

通过本实验，了解了 74244、74280、74138 等器件的逻辑功能和接线方法；初步学习了使用 TTL 设计电路并在计算机上进行仿真的环境；初步学习了 Verilog HDL 硬件描述语言的基本语法。

多个输入同时发生变化可能带来电路的不稳定（毛刺）。

为保证电路正确，连线后可以在主要的逻辑步骤，包括海明码生成、造错输出、纠错输出等处加入一些额外输出，以检查电路各部分逻辑的正确性。

关于 Maxplus 软件的使用：在画好电路图或写好 Verilog 准备仿真时，需要 Set project to current file。在设计 scf（波形文件）时，可以充分利用左侧工具栏的按钮，减少手工设置数值的麻烦。

实验二 十六进制译码计数器的设计

一、实验目的

了解可编程芯片的特点；掌握 GAL 芯片的设计和应用方法；掌握 GAL 芯片的设计方式。

二、实验要求

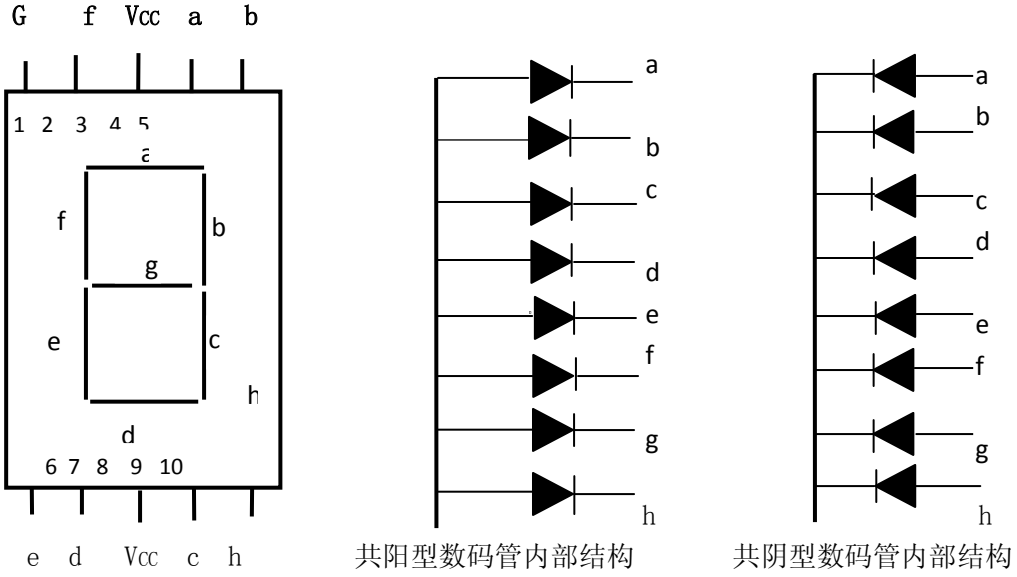
设计一个 16 进制译码器，4 位二进制输入 0～15 代表 0～F 的十六进制数，7 个输出连接到 7 段数码管，输出 0～F 的十六进制字符。使用 GAL 可编程芯片完成逻辑设计。

三、实验原理

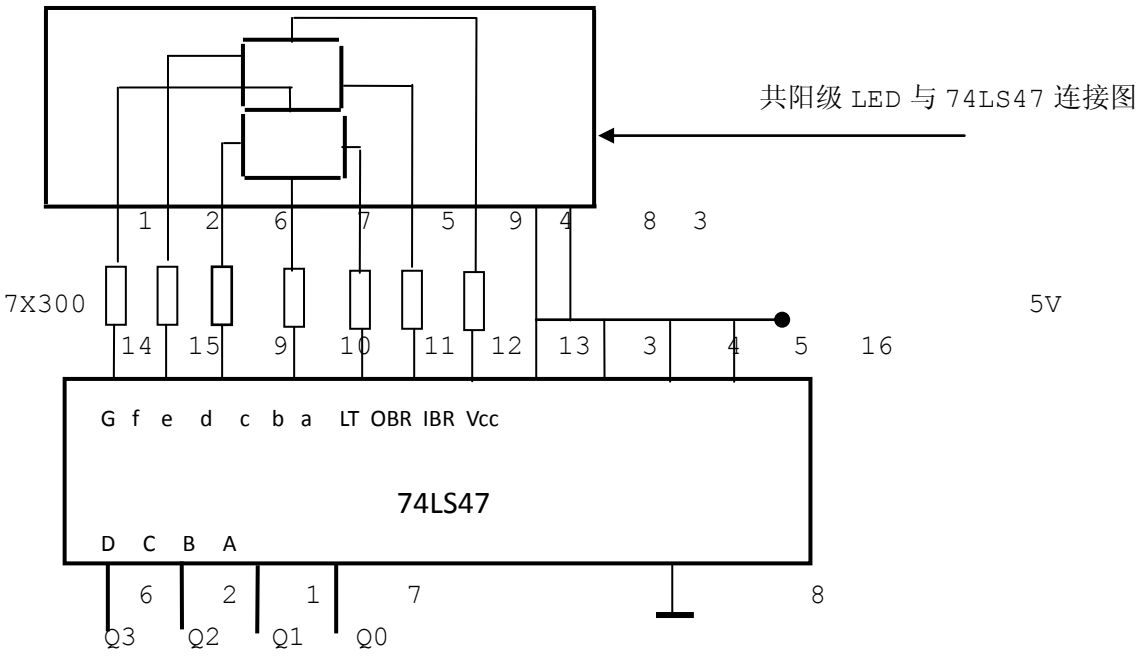
（1）数码管原理

数码管是一种常用器件，当显示十进制数时，一般根据所使用的数码管是共阳极还是共阴极来选择对应芯片。

七段发光二极管（LED）数码显示器的字形与七段荧光数码管一样，外观为平面型。它的 a、b、c、d、e、f、g 段是用发光二极管显示的，并且分为共阳极和共阴极两种。共阳极是七个发光二极管的阳极接在一起，接到高电平（正电源）上，阴极接到译码器的输出端，哪个发光二极管的阴极为低电平，哪个发光二极管就亮，而阴极为高电平的发光二极管就不亮。共阴极是七个发光二极管的阴极接到一起，接到低电平处，哪个发光二极管的阳极接高电平，哪个发光二极管就亮，否则就不亮。这种数码特点是电源电压为 5V，与 TTL 电源一致共阳型数码管内部结构。



共阳极和共阴极两种 LED 数码管内部接线示意图见图 1。与共阳级 LED 数码相接的七段译码器的 a—g 输出必须是低电平有效。例如用 SN74LS47 即可（它的输出级为集电极开路），接线图如下所示。



（2）GAL 可编程芯片原理

GAL，通用阵列逻辑，英文全称：generic array logic。

GAL 的优点：

1.具有电可擦除的功能，克服了采用熔断丝技术只能一次编程的缺点，其可改写的次数超过 100 次；

2.由于采用了输出宏单元结构，用户可根据需要进行组态，一片 GAL 器件可以实现各种组态的 PAL 器件输出结构的逻辑 功能，给电路设计带来极大的方便；

3.具有加密的功能，保护了知识产权；

4.在器件中开设了一个存储区域用来存放识别标志——即电子标签的功能。

GAL 器件的基本结构：

GAL 有五个部分组成：

1.输入端：GAL16V8 的 2～9 脚共 8 个输入端，每个输入端有一个缓冲器，并由缓冲器引出两个互补的输出到与阵列；

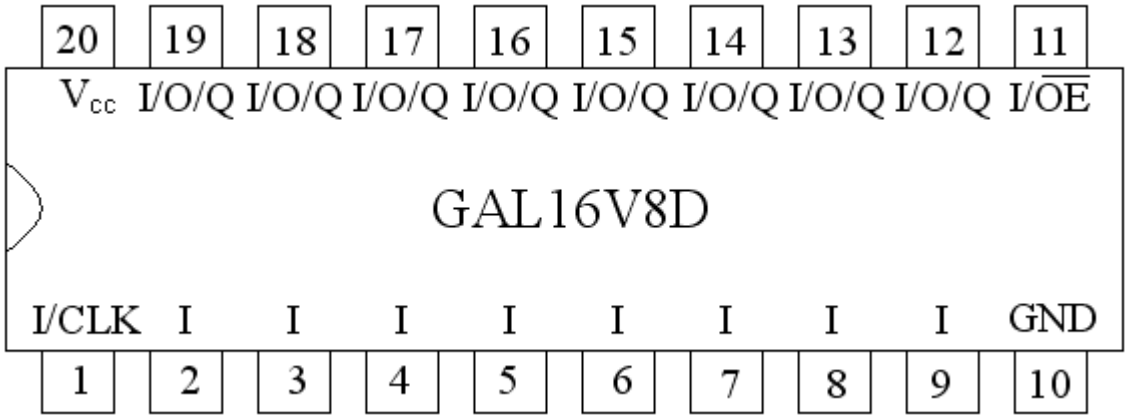
2.与阵列部分：它由 8 根输入及 8 根输出各引出两根互补的输出构成 32 列，即与项的变量个数为 16；8 根输出每个输出对应于一个 8 输入或门（相当于每个输出包含 8 个与项）构成 64 行，即 GAL16V8 的与阵列为一个 32×64 的阵列,共 2048 个可编程单元（或结点）；

3.输出宏单元：GAL16V8 共有 8 个输出宏单元，分别对应于 12～19 脚。每个宏单元的电路可以通过编程实现所有 PAL 输出结构实现的功能；

4.系统时钟：GAL16V8 的 1 脚为系统时钟输入端,与每个输出宏单元中 D 触发器时钟输入端相连，可见 GAL 器件只能实现同步时序电路，而无法实现异步的时序电路；

5.输出三态控制端:GAL16V8 的 11 脚为器件的三态控制公共端。

下图是 GAL 管脚输入输出的规定。



四、逻辑设计

在 GAL 芯片的逻辑设计中，首先列出 7 个输出与 4 个输入的真值表，然后求出最简与或式。

以下用伪代码描述这个真值表。

```
"    a
"   ----
" b|   |c
"   ----d
" e|   |f
"   ----
"    g

hex2led device 'GAL16V8D';
D0,D1,D2,D3 pin 2,3,4,5;
OE pin 11;
```

```
a,b,c,d,e,f,g pin 12,13,14,15,16,17,18;
GND pin 10;
VCC pin 20;
bcd = [OE,D3,D2,D1,D0];
led = [a,b,c,d,e,f,g];

truth_table ( bcd -> led )
[0,0,0,0] -> [1,1,1,0,1,1,1];
[0,0,0,1] -> [0,0,1,0,0,1,0];
[0,0,1,0] -> [1,0,1,1,1,0,1];
[0,0,1,1] -> [1,0,1,1,0,1,1];
[0,1,0,0] -> [0,1,1,1,0,1,0];
[0,1,0,1] -> [1,1,0,1,0,1,1];
[0,1,1,0] -> [1,1,0,1,1,1,1];
[0,1,1,1] -> [1,0,1,0,0,1,0];
[1,0,0,0] -> [1,1,1,1,1,1,1];
[1,0,0,1] -> [1,1,1,1,0,1,1];
[1,0,1,0] -> [1,1,1,1,1,1,0]; "A
[1,0,1,1] -> [0,1,0,1,1,1,1]; "b
[1,1,0,0] -> [0,0,0,1,1,0,1]; "c
[1,1,0,1] -> [0,0,1,1,1,1,1]; "d
[1,1,1,0] -> [1,1,1,1,1,0,1]; "e
[1,1,1,1] -> [1,1,0,1,1,0,1]; "F
```

五、GAL 程序代码

GAL 程序的设计规范：

前四行是说明行，其中第一行是器件名，第二行是实现的功能，第三行是作者，第四行是用途。

接下来的两行依次描述芯片的输入输出针脚。本程序中 D0、D1、D2、D3 是四位二进制输入（正逻辑），a、b、c、d、e、f、g 是七位数码管输出（低电平有效），VCC 和 GND 分别是电源和接地。

以后每行描述一个输出与输入的关系，采用与或式表达。

最后一行 DESCRIPTION 结束程序。

```
PLD16V8
hex2led
boj 2011-10-31
digital circuit

NE D0 D1 D2 D3 NULL1 NULL2 NULL3 NULL4 GND
OE NULL5 g f e d c b a VCC

/a = /D3*/D2*/D0 + /D3*/D2*D1*D0 + /D3*D2*/D1*D0 + /D3*D2*D1 + D3*/D2*/D1 +
D3*/D2*D1*/D0 + D3*D2*D1
/b = /D3*/D1*/D0 + /D3*D2*/D1*D0 + /D3*D2*D1*/D0 + D3*/D2 + D3*D2*D1
/c = /D3*/D2 + /D3*D2*/D1*/D0 + /D3*D2*D1*D0 + D3*/D2*/D1 + D3*/D2*D1*/D0 +
D3*D2*/D1*D0 + D3*D2*D1*/D0
/d = /D3*/D2*D1 + /D3*D2*/D1 + /D3*D2*D1*/D0 + D3
/e = /D3*/D2*/D0 + /D3*D2*D1*/D0 + D3*/D2*/D1*/D0 + D3*/D2*D1 + D3*D2
/f = /D3*/D2*/D1 + /D3*/D2*D1*D0 + /D3*D2 + D3*/D2 + D3*D2*/D1*D0
/g = /D3*/D2*/D1*/D0 + /D3*/D2*D1 + /D3*D2*/D1*D0 + /D3*D2*D1*/D0 + D3*/D2*/D1
```

```
+ D3*/D2*D1*D0 + D3*D2

    DESCRIPTION
```

六、实验步骤

（1）用 GAL 编译软件将 PLD 文件编译成 JED 二进制文件。

（2）将 GAL16V8 芯片放在烧写器上固定好。其中注意芯片应顶在烧写器的末端，芯片上的缺口应紧挨着末端。芯片的放置位置和方向如果错误，不但不能烧写成功，还可能损坏芯片。

（3）将实验台通过 USB 线连接到计算机，打开 TopWin 烧写软件。

（4）将 JED 文件烧写到芯片中，并检查烧写是否成功。

（5）将芯片插入实验台（小心，以免弄弯针脚），按照芯片接线图和程序中关于输入输出的约定连线，注意 VCC 和 GND 的连接。

为保证 VCC 和 GND 连接正确，可以先用导线将此位置与控制台上的发光二极管接口试触，观察发光二极管的反应。

其中，D3～D0 连接到四个输入按钮，a～g 连接到七段数码管。七段数码管希望显示的位应接出一根单独的线到 GND 以使能此位显示。注意数码管都是低电平使能的。

（6）按动实验台上的输入按钮，观察七段数码管的显示。

遇到错误，首先检查接线，纠正接触不良等问题；如果再有问题则是程序问题，需要检查程序、重新烧写。

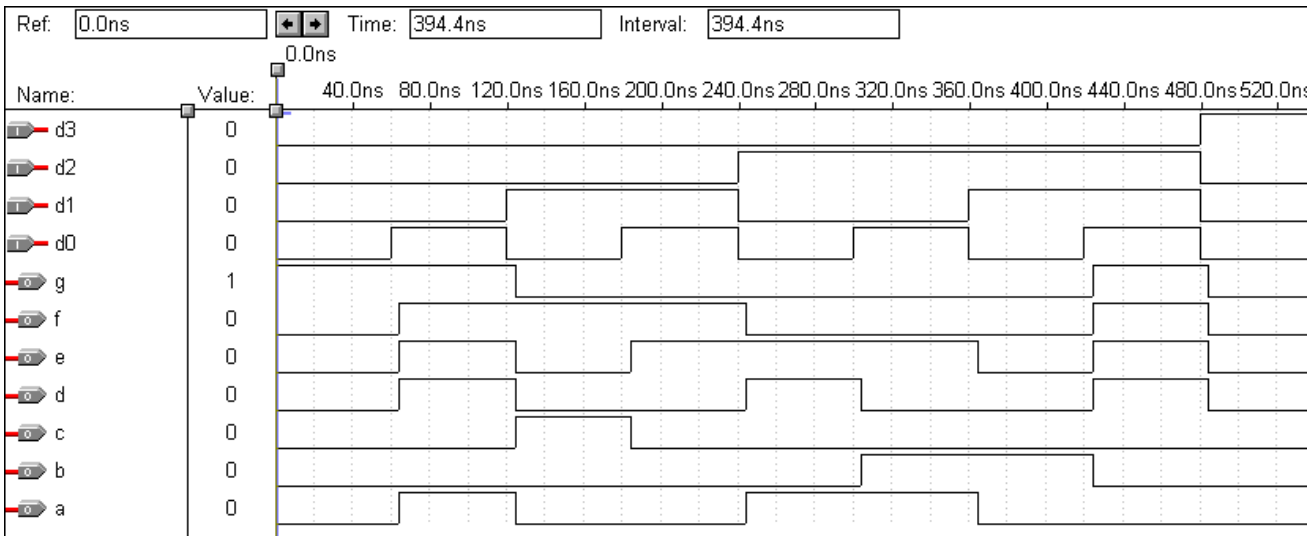
七、Verilog 设计

这个设计就是简单地将 GAL 方程转写成 Verilog 的 assign 语法。

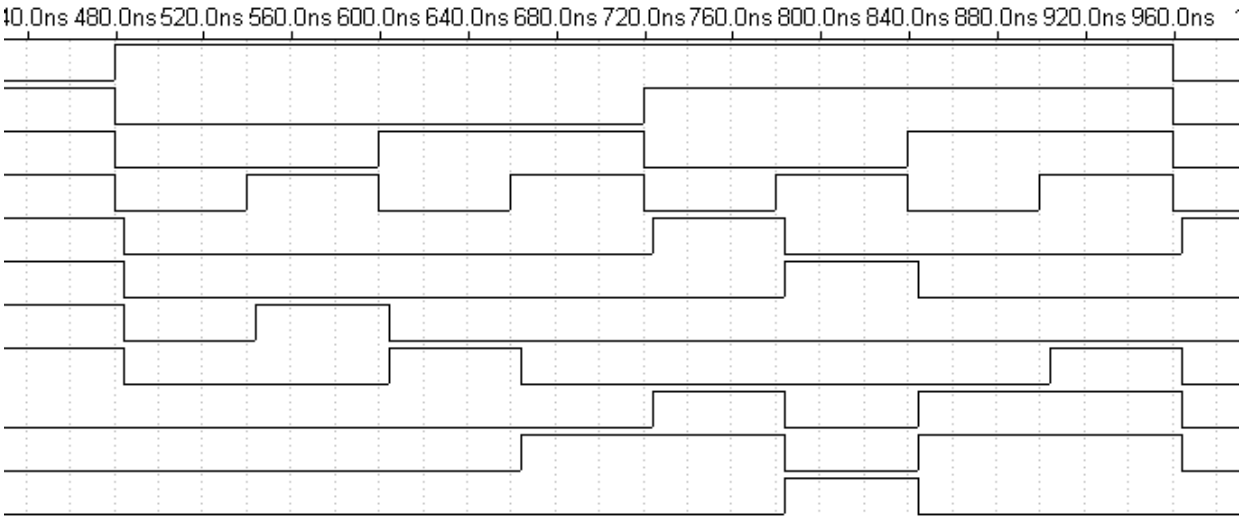
```
module gal(d0,d1,d2,d3,a,b,c,d,e,f,g);
input d0,d1,d2,d3;
output a,b,c,d,e,f,g;

assign a = ~(~d3&d1 | d2&d1 | d3&~d0 | d3&~d2*d0 | ~d2&~d0 | d3&~d2&~d1);
assign b = ~(~d0&~d2 | ~d2&~d1 | ~d2&~d0 | ~d3&~d1&~d0 | d3&~d1&d0 | ~d3&d1&d0);
assign c = ~(d3&~d2 | ~d1&d0 | ~d3&d2 | ~d3&~d1 | ~d3&d0);
assign d = ~(d3&~d1 | ~d2&d1&d0 | d2&~d1&d0 | ~d3&d1&~d0 | d2&d1&~d0 |
~d2&~d1&~d0);
assign e = ~(d3&d2 | d1&~d0 | ~d2&~d0 | d3&d1);
assign f = ~(d3&~d2 | ~d1&~d0 | d3&d1 | d2&~d0 | ~d3&d2&~d0 | d0&~d1&d2&~d3);
assign g = ~(d3&~d2 | d1&~d0 | d3&d0 | ~d2&d1 | ~d3&d2&~d1);
endmodule
```

八、Verilog 仿真

仿真结果说明：d3~d0 表示四位二进制输入，g~a 是七段数码管输出。



为清晰起见，图截成了左右两部分，以下为续图。



九、实验总结

在本实验中求最简与或式时需要细心认真，实验过程中需要检查七段 LED 的每个显示是否正确，如果不正确，应当立即修正程序。

由于七段 LED 是低电平有效，不太符合我们的思考习惯，因此可以先按照高电平有效的方式列出逻辑关系并化简，最后再对输出取反（加一组非门）。

实验三 时序脉冲分频分配延迟与整形电路

一、实验目的

触发器的应用，计数器的设计；移位寄存器的的设计及多种脉冲产生的方式。

掌握同步时序电路和分频，延迟整形的原理和设计方法， 进一步提高实践能力。

二、实验要求

①脉冲源为 10 兆，要求得到主脉冲为 1 兆，即周期 T=lus 脉宽 500ns（占空比 1:1）。

②在主脉冲 CP 下产生单拍脉冲 CPO（可以不做）。

③在主脉冲 CP 下产生三个周期 T1-T2，每个周期包括 2 个主脉冲的分频分配器，输出系统波为 CP1-CP2。

④有 CP1 得到延迟 200ns，波宽为 200ns 的脉冲 CP1.

⑤由 CP2 得波宽为 700ns 的脉冲 CP2.

⑥分频器、周期发生器均设计成同步型,周期发生器用移位方式

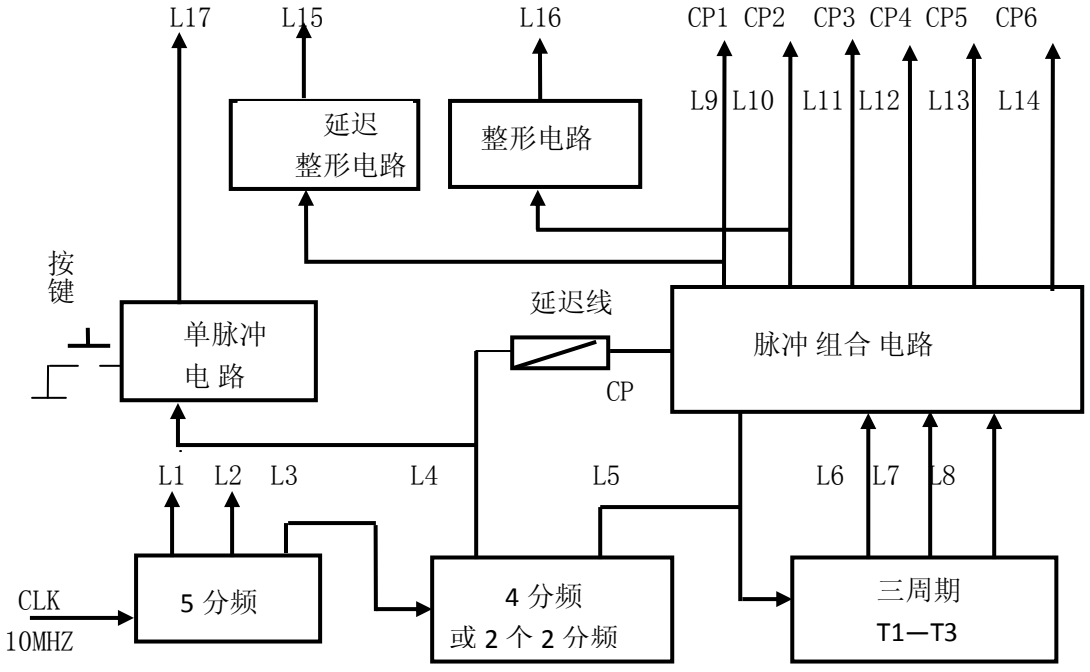
三、实验原理

时序脉冲电路在计算机中是不可缺少的一部分。主要有振荡源（目前都用晶振）、主脉冲、分频器、分配器、延迟和整形电路、单脉冲和定数脉冲电路等组成。以产生周期和所要求的脉冲分配。目前计算机的速度越来越高，对脉冲本身的波形和一致性要求也越来越高。如在高频电路中，为了保证时间配合，防止干扰等，主脉冲在加以驱动后以电平（即宽脉冲）和等长线形式并行送到各插件，各插件以相同电路将宽脉冲整行为窄脉冲（几个 ns 到几十个 ns）使用。

四、逻辑设计

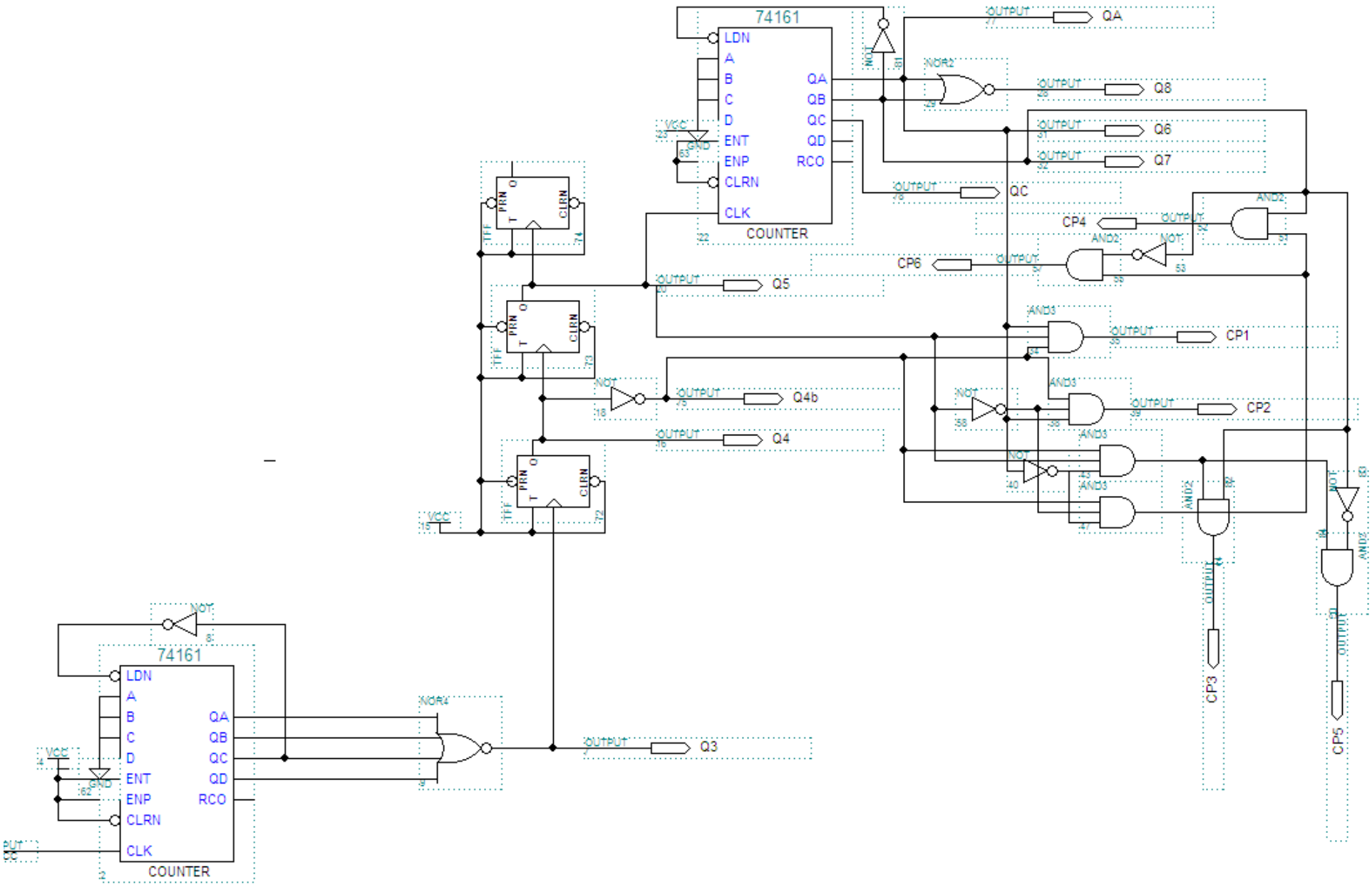
如下图所示，首先使用 74161 计数器实现五分频，然后用两次 T 触发器实现 2 分频、4 分频，再用 74161 在 T 触发器输出上实现 3 分频，最后用各级分频的输出经组合逻辑电路输出希望得到的波形。

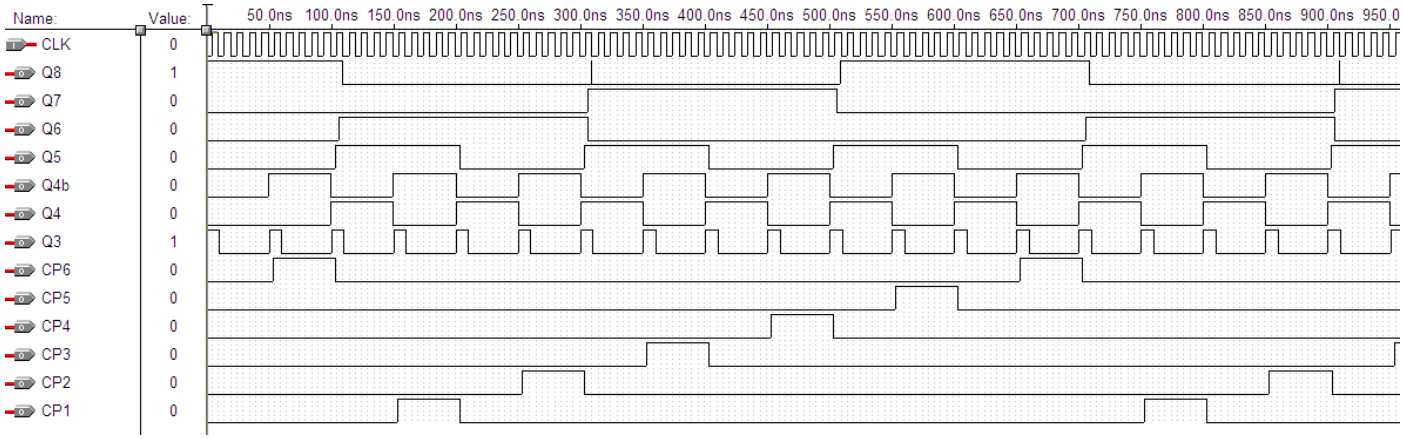
用 74161 实现 n 分频的原理：74161 是同步计数器，利用其复位功能，在 74161 的计数输出恰好为 n-1 时（用组合逻辑中的非门、与门实现），将复位（LDN）置为低电平，使得 74161 在下一时钟周期从零开始计数。这样，74161 的计数输出为 n-1 的占空比恰好为时钟信号的 1/n，实现了 n 分频。



时序脉冲分频，分配，延迟与整形框图

五、电路图设计与仿真结果


74161
LDN
A
B
C
D
ENT
ENP
CLRN
CLK
QA
QB
QC
QD
RCO
COUNTER
VCC
GND
NOT
NOR2
NOR4
AND2
AND3
TFF
PRN
CLRN
OUTPUT
Q8
Q6
Q7
QC
CP4
CP6
Q5
CP1
Q4b
CP2
Q4
Q3
CP3
CP5



仿真结果说明：输出的波形图符合实验设计要求。

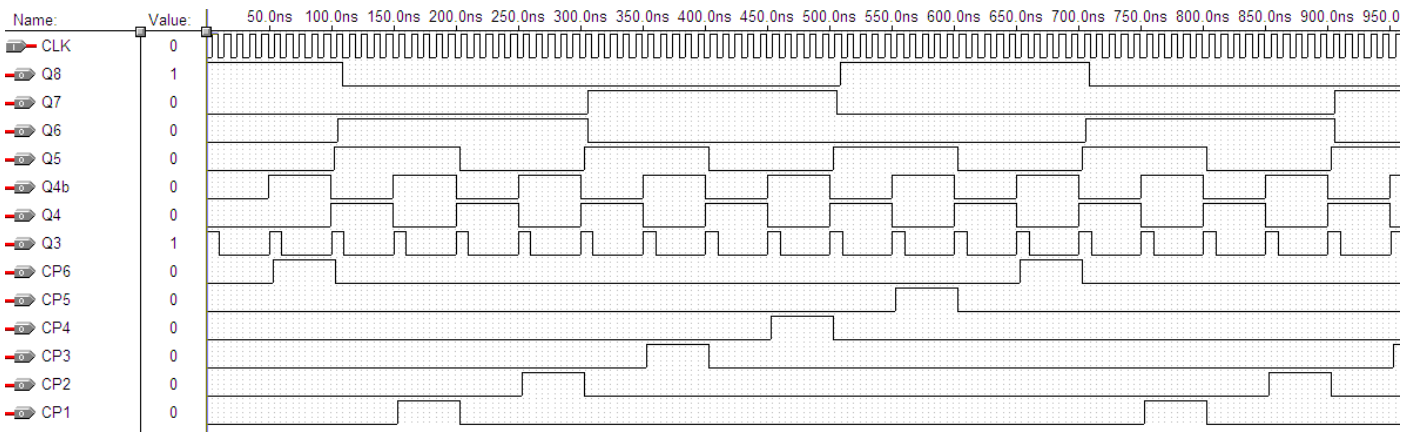
Q8 中存在毛刺，这是由于组合逻辑电路的传输时延不一致造成的。

六、Verilog 与仿真结果

Verilog 的设计基本按照电路图进行。

```
module frequency(CLK,Q8,Q7,Q6,Q5,Q4b,Q4,Q3,CP6,CP5,CP4,CP3,CP2,CP1);
input CLK;
output Q8,Q7,Q6,Q5,Q4b,Q4,Q3,CP6,CP5,CP4,CP3,CP2,CP1;
reg QA,QB,QC,QD;
reg T4,T5;
reg PA,PB;
wire five;

always@(posedge CLK) // 74161
    begin
    if(QC)
        begin
        QA = 0;
        QB = 0;
        QC = 0;
        end
    else
        begin
        QA = ~QA;
        if(!QA)
            begin
            QB = ~QB;
            if(!QB)
                begin
                QC = ~QC;
                end
            end
        end
    end

assign five = ~(QA | QB | QC);
assign Q3 = five;
```

```
always@(posedge five)
    T4 <= ~T4;
assign Q4 = T4;
assign Q4b = ~T4;

always@(posedge T4)
    T5 <= ~T5;
assign Q5 = T5;

always@(posedge Q5) // 74161
    begin
    if(PB)
        begin
        PA = 0;
        PB = 0;
        end
    else
        begin
        PA = ~PA;
        if(!PA)
            PB = ~PB;
        end
    end

assign Q8 = ~(PA | PB);
assign Q6 = PA;
assign Q7 = PB;

assign CP1 = Q4b & Q5 & Q6;
assign CP2 = Q4b & ~Q5 & Q6;
assign CP3 = Q4b & Q5 & ~Q6 & Q7;
assign CP4 = Q4b & ~Q5 & ~Q6 & Q7;
assign CP5 = Q4b & Q5 & ~Q6 & ~Q7;
assign CP6 = Q4b & ~Q5 & ~Q6 & ~Q7;

endmodule
```



七、实验总结

容易发现，Verilog 在设计组合逻辑电路中的简洁性优势是电路图不可比拟的。在 Verilog 中，仅用最后 6 行语句就清晰地描述了逻辑关系，而在电路图中右半部分充满了各种门电路，不仅画图复杂、耗时长，且容易出错。

在电路图调试过程中，出现了三个问题：

①由于右侧的接线较为混乱，花了不少时间才找到接线中的一处错误并将其纠正。

②电路图中元件的管脚（输入输出端）与所有经过它的导线均连接。因此不与元件连接的导线不能“穿过”该元件，而要从尽可能远的地方通过。

③在 MaxPlus 中，元件的位置移动后，会同时移动连接在其上的导线，并连接上“碰上”其管脚的其他导线。其结果往往是我们不希望的。因此在设计电路图时，应尽量使得元件的排放层次分明，不相关的导线减少交叉，不相关的电路逻辑部分在电路图上相距不宜过近。这样，在调整一个元件的位置和方向时，可以减少副作用。

归根结底，此电路调试的困难是由于元件位置和接线的混乱。

最好的解决方法是使用 3-8 译码器或类似的集成电路，一个集成块将内部的复杂逻辑封装起来，不易出错，也使得电路图更加清晰。事实上，此电路的“6 个 assign 语句”，即组合逻辑部分，应当通过 3-8 译码器实现。

即使不能使用集成芯片，以后设计类似的复杂组合逻辑电路时也要事先规划好各元件的位置，待排布整齐后再连线。

实验四 八位数据串入—并出逻辑设计实验

一、实验目的

掌握和了解移位寄存器；并行寄存器和计数器的灵活应用，分析典型逻辑电 74164 功能和使用方法，了解收发之间的通信约定应用，为开拓自己的设计思路和能力提供参考。

二、实验要求

要求设计实现接受 8 位数据串入并出逻辑电路。

要有通信约定，约定接受码 10000001，停止码 01111110。

要注意控制逻辑的设计，注意冒险尖脉冲冒险。

i. 数据输入：

将数据串行输入系统中.对于本实验,以八位数据为基准.即对输入的数据分隔,每八位作为一个整体将其输出.

ii. 数据输出：

当输入的数据满足一定条件时将其每八位并行输出一次.这其中数据需满足的条件是:数据组合必须是每八位组合一次,并且当数据处于接收状态.

iii. 数据控制：

将输入的数据同样按八位读取,当读取到某一特定串行输入数据序列时,控制器处于开启 状态,表示接受数据输入.当输入的八位数据为另外一个特定的串行输入数据序列时,控制器处于关闭状态.并且在这时候控制数据输出是否将串行输入的数据并行输出.由于数据必须满足上述两个条件,所以数据控制部分可分为两个小模块来完成.一部分用于判定数据的输入序列是否为特定的序列.另一个模块用于对串行输入的数据进行计数,看是否已输入八位.

本实验中,自己约定两个特定的序列. "10000001"表示控制器开启,即读取数据开始，开始工作。相反地,"01111110"表示控制器关闭,即读取数据结束，停止工作。

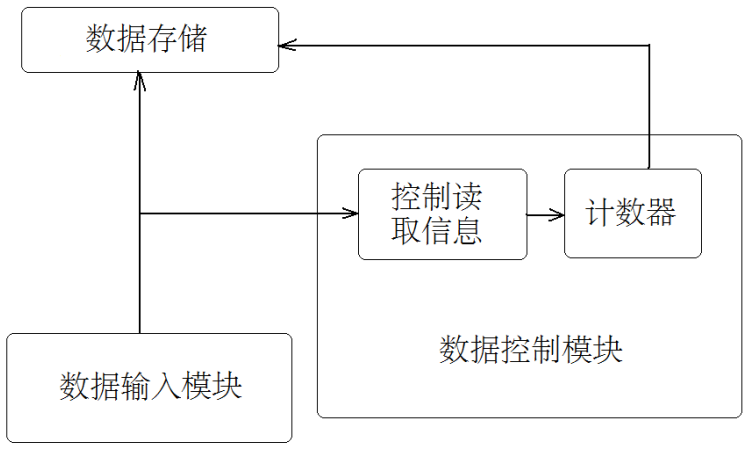
三、实验设计

根据实验的原理及要求,可以将整个系统分为三个模块来设计.

数据输入模块：以时钟信号为基础,将时钟信号作为输入,通过高低电平的控制,实现串行输入数据的序列.此时需将所输入的数据存储在一个存储器中,以便用于后面的输出工作.

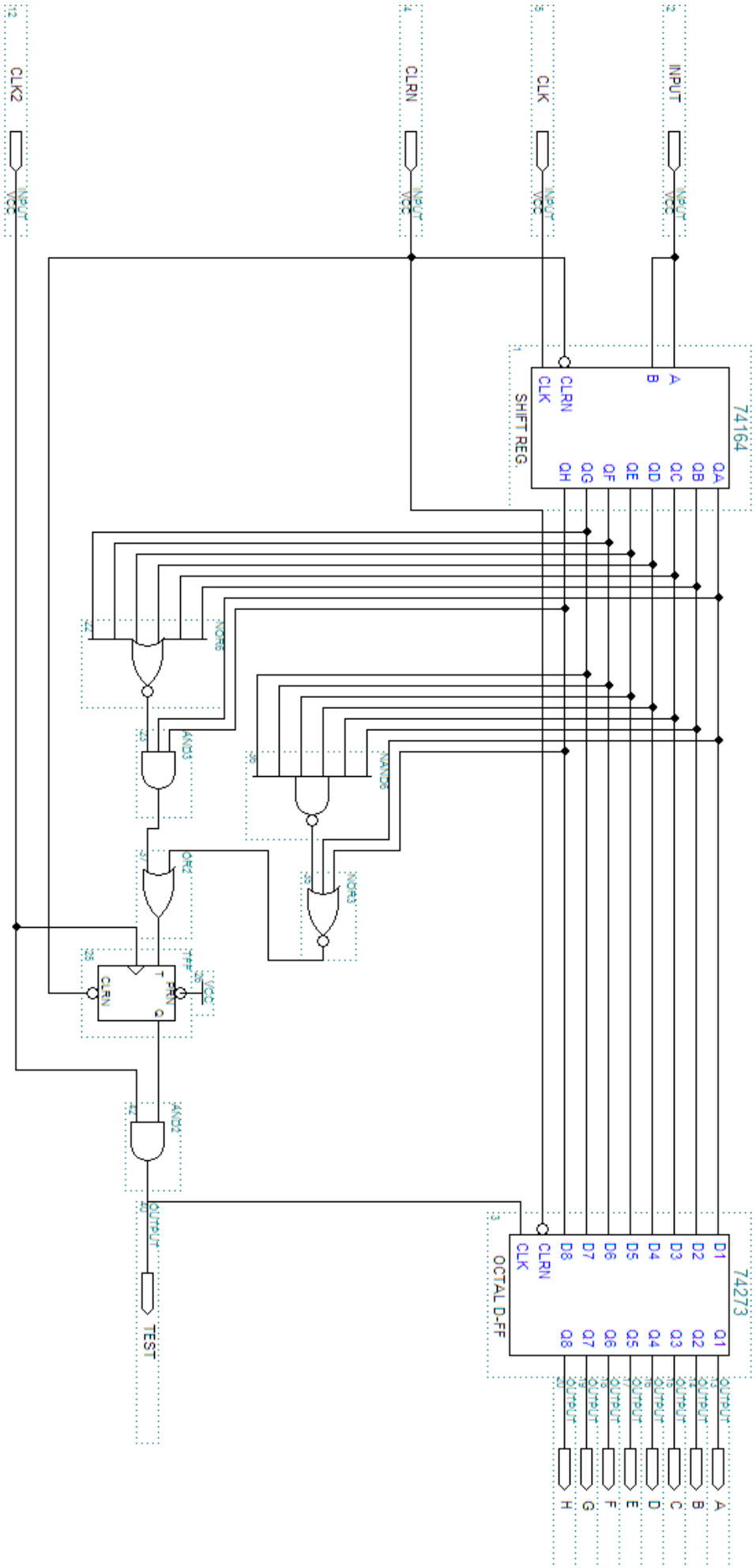
数据控制模块：将数据输入模块所输入的数据输出作为数据控制模块的输入数据序列.根据实验原理及要求部分所述,将”10000001”作为数据控制模块的开启信息. 而”01111110”作为数据控制模块的关闭信息,表示数据读取结束.这两个状态是相反的,可用一个 D 触发器来获得最终的控制结果. 此时还需要作另外一个判断,即串行输入的数据序列是否已满 8 位.因此需要一个计数器来对已输入的数据序列进行计数.当已有 8 位时,数据控制模块的逻辑值应为 1,即将把串行输入的 8 位数据并行输出.

数据输出模块：由于在数据输入模块中已经将数据存储在了一个存储器中,若此时数据输出模块接收到数据控制模块的逻辑值为 1 的信息,那么数据模块将把存储在存储器的数据 8 位并行输出.否则,数据输出模块将不输出数据。

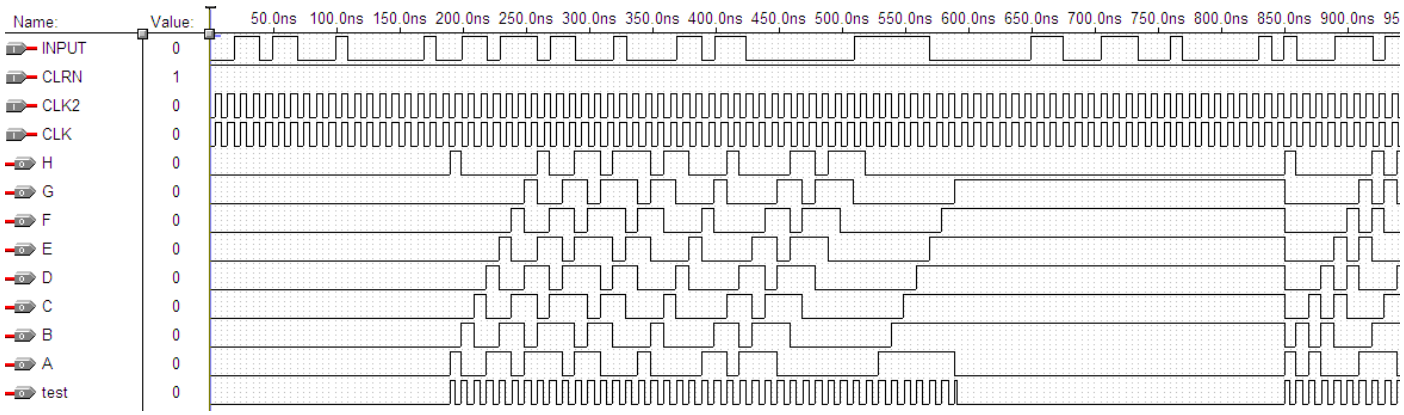


串行输入并行输出实验设计模块图

四、电路图和仿真结果


CLK2
CLRN
CLK
INPUT
74164
SHIFT REG.
74273
OCTAL D-FF
TEST
A
B
C
D
E
F
G
H

设计说明：74164 是移位寄存器。下侧的组合逻辑电路判断当前输入信号是否为 10000001（开始信号）或 01111110（终止信号），并据此改变 T 触发器的状态；T 触发器的输出即为输出使能。输出使能与时钟信号经过“与”，作为 74273 的时钟信号。74273（D 触发器）控制数据是否输出。



仿真结果说明：INPUT 是输入信号。CLK 与 CLK2 是时钟。A~H 是输出。Test 表示控制器开启时，输出移位寄存器的时钟信号。

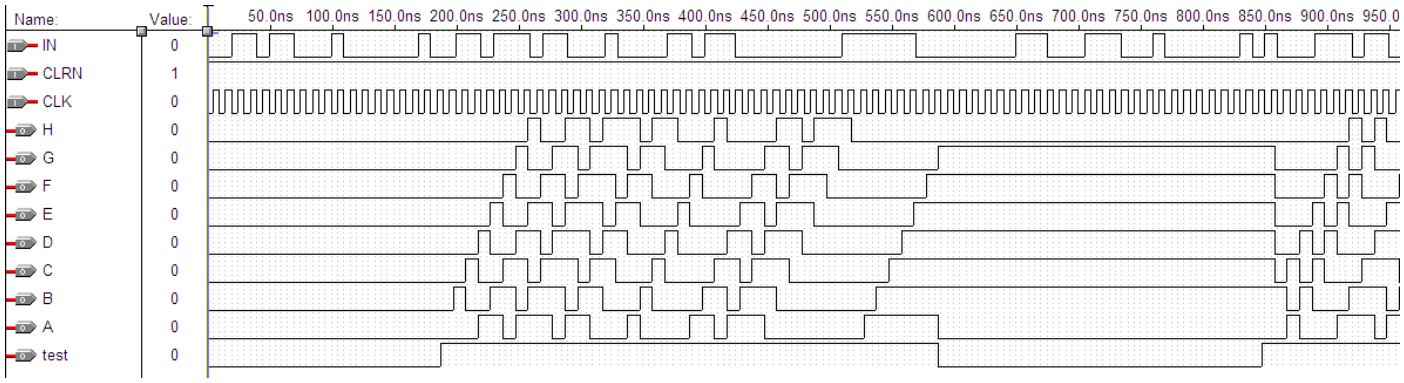
开始 A~H 输出均为低电平。在 100ns~180ns，数据输入为 10000001（开启信号），随后 A~H 开始按照移位方式输出；在 500~580ns，数据输入为 01111110（关闭信号），随后 A~H 均为低电平。在 760~840ns，数据输入又为开启信号，因此 A~H 又开始按照移位方式输出。

五、Verilog 和仿真结果

```
module io(in,clk,clrn,A,B,C,D,E,F,G,H,TEST);
input in,clk,clrn;
output A,B,C,D,E,F,G,H,TEST;
reg SA,SB,SC,SD,SE,SF,SG,SH; // shift reg
reg OA,OB,OC,OD,OE,OF,OG,OH; // output reg
reg t;

always@(posedge clk)
    if(clrn)
    begin
    // 74164 shift reg
        begin
        SA<=in;
        SB<=SA;
        SC<=SB;
        SD<=SC;
        SE<=SD;
        SF<=SE;
        SG<=SF;
        SH<=SG;
        end

    // t flip flop
    if ((SA==1 && SB==0 && SC==0 && SD==0 && SE==0 && SF==0 && SG==0 && SH==1) ||
(SA==0 && SB==1 && SC==1 && SD==1 && SE==1 && SF==1 && SG==1 && SH==0))
        t=~t;
    end
```

```
always@(posedge clk)
    if(clrn)
    if(t) // output reg
    begin
        OA<=SA;
        OB<=SB;
        OC<=SC;
        OD<=SD;
        OE<=SE;
        OF<=SF;
        OG<=SG;
        OH<=SH;
    end

assign A=OA;
assign B=OB;
assign C=OC;
assign D=OD;
assign E=OE;
assign F=OF;
assign G=OG;
assign H=OH;
assign TEST=t;

endmodule
```

仿真结果：



（基本与电路图的结果相同）其中，test 表示控制器开启，数据正在输出。

六、实验总结

本实验中所有元件采用统一的时钟脉冲，这样的时序逻辑电路不容易发生尖脉冲冒险，对输入信号中的“杂音”有较强的抵御能力。事实上，组合逻辑电路中毛刺产生的原因是信号传输的速度不同，在两个或多个输入信号同时改变时，它们造成的影响可能不在同一时刻传播过来。在这段不同步的间隔中，会产生短时间的电路状态不稳定。如果整个电路没有时钟来同步，那么这种不稳定就会反映到输出中，例如海明码实验中的毛刺。

本实验采用了时序逻辑，尽管在内部的组合逻辑部分仍然存在局部信号不稳定的现象，但由于信号的边沿触发特性，在下一个时钟边沿触发的时刻电路早已达到稳态，从而每个模块的输出都是稳定的，且输出的变化总是发生在时钟边沿；这样整个电路的输出也是稳定的，且输出的变化总是发生在时钟边沿，不仅使得波形图更加整洁，也便于送入其他组合或时序逻辑电路进一步处理。

实验五 程序计数器

一、实验目的

了解程序计数器（Program Counter）的原理和设计方式。

二、实验要求

设计程序计数器。程序计数器的输入包括 clk（时钟）、reset（复位）、ir（跳转）、t[1]（跳转使能）、t[0]（计数使能）。输出是程序计数器的值 pc[7..0]。

功能描述：

（1）复位 reset 低电平有效，当 reset 为低电平时，pc 立即恢复到 0，重新开始计数。

（2）不然，当 t[1]=1 且 ir[7]=ir[6]=1 时，程序计数器进行跳转，pc 设置为 ir[5..0]的值。

（3）不然，当 t[0]=0 时，PC 保持当前值不变；当 t[0]=1 时，PC 在时钟上升沿+1，实现计数。

三、实验原理

使用两片 74161 芯片实现。两片 74161 分别保存 PC 的 7~4 位和 3~0 位，再将负责 3~0 位的 74161 的进位连接到负责 7~4 位的 74161 的使能端。

74161 芯片的内部逻辑描述：

```
TITLE "Top-level file for the 74161 macrofunction.  Chooses a device-family
optimized implementation.";
FUNCTION p74161 (clk, ldn, clrn, enp, ent, d, c, b, a)
    RETURNS (qd, qc, qb, qa, rco);
FUNCTION f74161 (clk, ldn, clrn, enp, ent, d, c, b, a)
    RETURNS (qd, qc, qb, qa, rco);

PARAMETERS
(
   DEVICE_FAMILY
);
INCLUDE "aglobal.inc";

SUBDESIGN 74161
(
   clk               : INPUT = GND;
   ldn               : INPUT = VCC;
   clrn              : INPUT = VCC;
   enp               : INPUT = VCC;
   ent               : INPUT = VCC;
   d                 : INPUT = GND;
   c                 : INPUT = GND;
   b                 : INPUT = GND;
   a                 : INPUT = GND;
   qd                : OUTPUT;
   qc                : OUTPUT;
   qb                : OUTPUT;
   qa                : OUTPUT;
   rco               : OUTPUT;
)

VARIABLE
   IF (FAMILY_FLEX() == 1) GENERATE
```

```
        sub  : f74161;
    ELSE GENERATE
        sub  : p74161;
    END GENERATE;

BEGIN
    IF (USED(clk)) GENERATE
        sub.clk = clk;
    END GENERATE;
    IF (USED(ldn)) GENERATE
        sub.ldn = ldn;
    END GENERATE;
    IF (USED(clrn)) GENERATE
        sub.clrn = clrn;
    END GENERATE;
    IF (USED(enp)) GENERATE
        sub.enp = enp;
    END GENERATE;
    IF (USED(ent)) GENERATE
        sub.ent = ent;
    END GENERATE;
    IF (USED(d)) GENERATE
        sub.d = d;
    END GENERATE;
    IF (USED(c)) GENERATE
        sub.c = c;
    END GENERATE;
    IF (USED(b)) GENERATE
        sub.b = b;
    END GENERATE;
    IF (USED(a)) GENERATE
        sub.a = a;
    END GENERATE;
    qd = sub.qd;
    qc = sub.qc;
    qb = sub.qb;
    qa = sub.qa;
    rco = sub.rco;
END;
```

四、电路图与仿真结果

仿真结果与电路图见下两页。

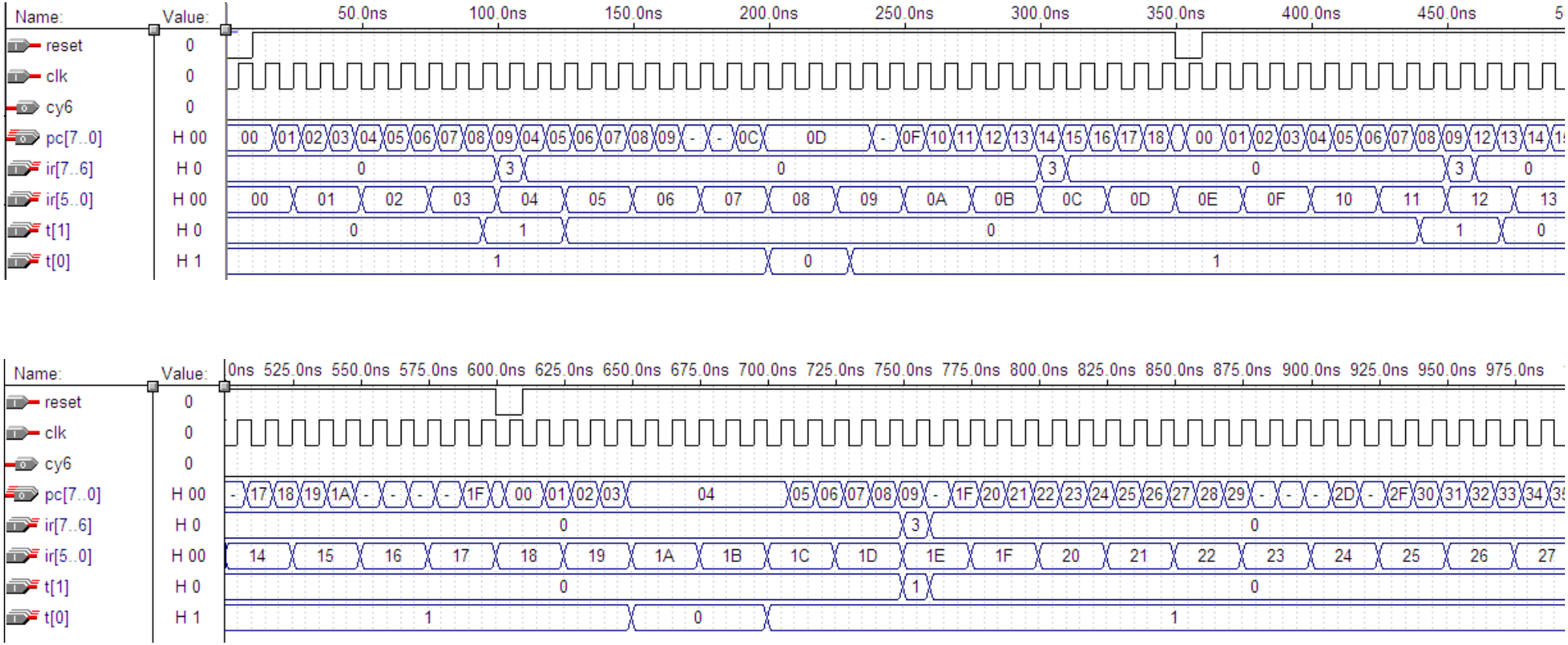
仿真结果说明：reset 是复位标志，低电平有效；clk 是电路的主时钟；PC 是八位程序计数器；ir[5..0]是跳转到的地址；ir[7..6]和 t[1]是跳转使能的标志；t[0]是 PC 计数使能。

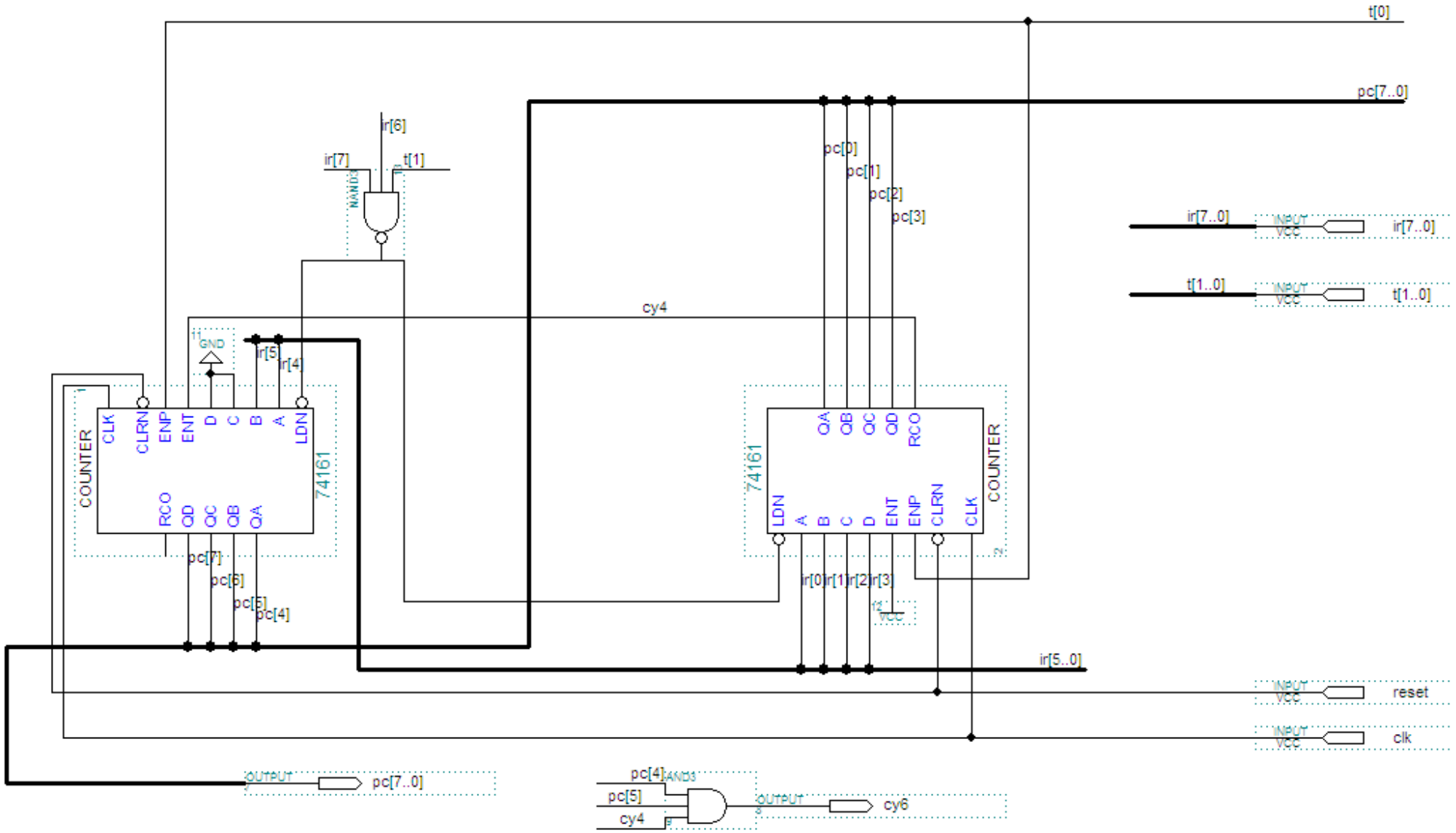
从仿真输入中可见，电路在各种输入状态中的表现均符合实验描述。

五、实验总结

在调试时，要使得波形文件包含所设计电路的各种功能，对电路进行全面测试。

使用导线标号的方式绘制电路图，可以减少导线的交叉，突出数据的主要流向，使电路图更简明。


Name:
Value:
50.0ns
100.0ns
150.0ns
200.0ns
250.0ns
300.0ns
350.0ns
400.0ns
450.0ns
reset
clk
cy6
pc[7..0]
ir[7..6]
ir[5..0]
t[1]
t[0]
H 00
H 0
H 1
0ns
525.0ns
550.0ns
575.0ns
600.0ns
625.0ns
650.0ns
675.0ns
700.0ns
725.0ns
750.0ns
775.0ns
800.0ns
825.0ns
850.0ns
875.0ns
900.0ns
925.0ns
950.0ns
975.0ns


t[0]
pc[7..0]
ir[6]
ir[7]
t[1]
NAND3
pc[0]
pc[1]
pc[2]
pc[3]
ir[7..0]
INPUT
VCC
ir[7..0]
t[1..0]
INPUT
VCC
t[1..0]
cy4
i1
GND
ir[5]
ir[4]
COUNTER
CLK
CLRN
ENP
ENT
D
C
B
A
LDN
74161
RCO
QD
QC
QB
QA
pc[7]
pc[6]
pc[5]
pc[4]
74161
QA
QB
QC
QD
RCO
LDN
A
B
C
D
ENT
ENP
CLRN
CLK
COUNTER
ir[0]
ir[1]
ir[2]
ir[3]
i2
VCC
ir[5..0]
INPUT
VCC
reset
INPUT
VCC
clk
OUTPUT
pc[7..0]
pc[4]
pc[5]
cy4
AND3
OUTPUT
cy6

六、Verilog 与仿真结果

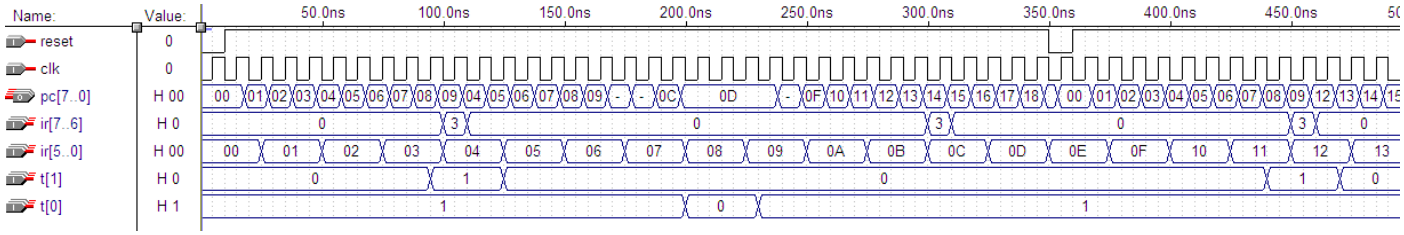
用 Verilog 语言设计时，没有拘泥于电路图设计中各元件的连接，而是直接根据逻辑功能的需求来设计电路。

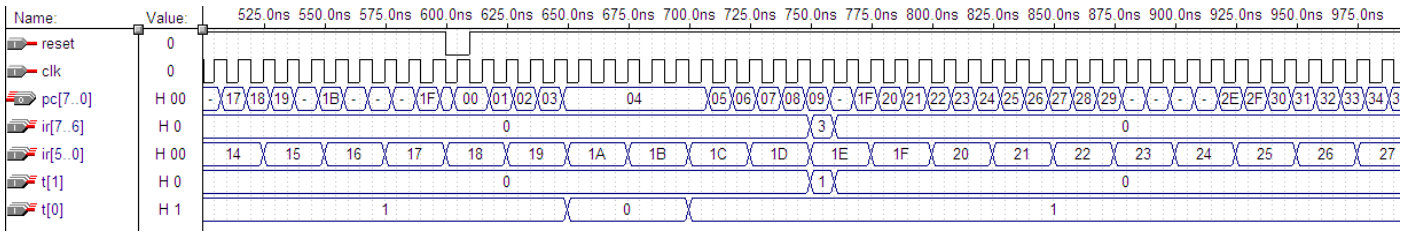
容易看出，相比考虑到芯片功能与连接方式的原理图，Verilog HDL 描述的电路逻辑更加清晰，可读性更好。同时，Verilog 的代码编写速度明显比添加元件、画图的原理图设计效率高、容易修改。因此，采用 Verilog 设计较为复杂的电路是比原理图有优势的。

```
module counter(reset,clk,ir,t,pc);
input reset,clk;
input [7:0]ir;
input [1:0]t;
output [7:0]pc;
reg [7:0]p; //pc in counter

always@(posedge clk)
   begin
   if(!reset)
      begin
      p[0]=0;
      p[1]=0;
      p[2]=0;
      p[3]=0;
      p[4]=0;
      p[5]=0;
      p[6]=0;
      p[7]=0;
      end
   else if(t[1] && ir[7] && ir[6])
      begin
      p[0]=ir[0];
      p[1]=ir[1];
      p[2]=ir[2];
      p[3]=ir[3];
      p[4]=ir[4];
      p[5]=ir[5];
      p[6]=0;
      p[7]=0;
      end
   else if(t[0]) //counter++
      begin
      p[0]=~p[0];
      if(!p[0])
         begin
         p[1]=~p[1];
         if(!p[1])
            begin
            p[2]=~p[2];
            if(!p[2])
               begin
```

```
                p[3]=~p[3];
                if(!p[3])
                    begin
                    p[4]=~p[4];
                    if(!p[4])
                        begin
                        p[5]=~p[5];
                        if(!p[5])
                            begin
                            p[6]=~p[6];
                            if(!p[6])
                                p[7]=~p[7];
                            end
                        end
                    end
                end
            end
        end
    end

assign pc[0] = p[0];
assign pc[1] = p[1];
assign pc[2] = p[2];
assign pc[3] = p[3];
assign pc[4] = p[4];
assign pc[5] = p[5];
assign pc[6] = p[6];
assign pc[7] = p[7];

endmodule
```

仿真结果：（与电路图的仿真结果基本相同）





实验六 十六位运算器的设计

一、实验目的

1、掌握算术逻辑运算器单元 ALU（74LS181）的工作原理

2、ALU 能主要完成对二进制信息定点整数的算术运算，逻辑运算主要有逻辑与、逻辑或、逻辑异或和逻辑非操作

3、了解提前进位 74LS182 芯片的逻辑公式推导

二、实验要求

用 74181，74182 等集成块和数字逻辑实验仪组成 16 位可控运算器。

控制不同运算方式，并列表记录运算结果。

三、实验原理

（1）74181 功能表

Selection				Active High Data			
				M = H	M = L; Arithmetic Operations		
				Logic	/Cn = H	/Cn = L	
S3	S2	S1	S0	Functions	(No Carry)	(With Carry)	
L	L	L	L	F = /A	F = A	F = A plus 1	
L	L	L	H	F = /(A + B)	F = A + B	F = (A + B) plus 1	
L	L	H	L	F = (/A)B	F = A + /B	F = (A + /B) plus 1	
L	L	H	H	F = 0	F = minus 1 (2s Comp)	F = ZERO	
L	H	L	L	F = /(AB)	F = A plus A(/B)	F = A plus A(/B) plus 1	
L	H	L	H	F = /B	F = (A + B) plus A(/B)	F = (A + B) plus A(/B) plus 1	
L	H	H	L	F = A $ B	F = A minus B minus 1	F = A minus B	
L	H	H	H	F = A(/B)	F = A(/B) minus 1	F = A(/B)	
H	L	L	L	F = /A+ B	F = A plus AB	F = A plus AB plus 1	
H	L	L	H	F = /(A $ B)	F = A plus B	F = A plus B plus 1	
H	L	H	L	F = B	F = (A + /B) plus AB	F = (A + /B) plus AB plus 1	
H	L	H	H	F = AB	F = AB minus 1	F = AB	
H	H	L	L	F = 1	F = A plus A*	F = A plus A plus 1	
H	H	L	H	F = A + /B	F = (A + B) plus A	F = (A + B) plus A plus 1	
H	H	H	L	F = A + B	F = (A + /B) plus A	F = (A + /B) plus A plus 1	
H	H	H	H	F = A	F = A minus 1	F = A	

74181 有高电平和低电平两种工作方式，高电平方式采用原码输入输出，低电平方式采用反码输入输出。通常选用高电平方式。

其中：M：算术 / 逻辑运算选择输入 M=0 算术运算 M=1 逻辑运算

Cn：带或不带进位运算选择输入 Cn=0 带进位 Cn=1 不带进

S3~S0：函数选择输入

A3~A0：4 位被加数输入

B3~B0：4 位加数输入

F3~F0：4 位函数输入

Cn+4：进位输出 Cn+4 =0 有进位出 Cn+4 =1 无进位出

（2）超前进位的原理

当需用 4 片 74181 构造 16 位运算器时，为避免组与组（即片与片）之间的串行进位（又称行波进位）、提高运算速度，可采用先行进位发生器 74182 来实现 16 位全并行运算器。

74LS182 功能表

C_{n+x}输出功能表

输入端			输出端
$\overline{G}0$	$\overline{P}0$	C_n	C_{n+x}
L	X	X	H
X	L	H	H
其他所有组合			L

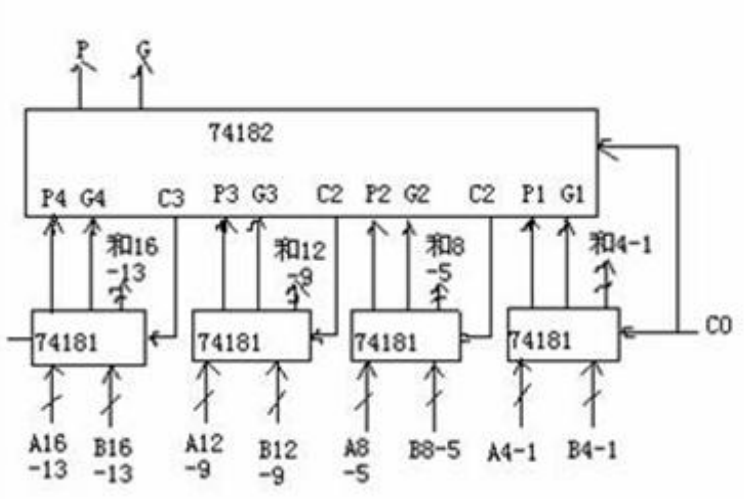
C_{n+y}输出功能表

输入端					输出端
$\overline{G}1$	$\overline{G}0$	$\overline{P}1$	$\overline{P}0$	C_n	C_{n+y}
L	X	X	X	X	H
X	L	L	X	X	H
X	X	L	L	H	H
其他所有组合					L

C_{n+z}输出功能表

输入端							输出端
$\overline{G}2$	$\overline{G}1$	$\overline{G}0$	$\overline{P}2$	$\overline{P}1$	$\overline{P}0$	C_n	C_{n+z}
L	X	X	X	X	X	X	H
X	L	X	L	L	X	X	H
X	X	L	L	L	X	X	H
X	X	X	L	L	L	H	H
其他所有组合							L

四、实验逻辑设计



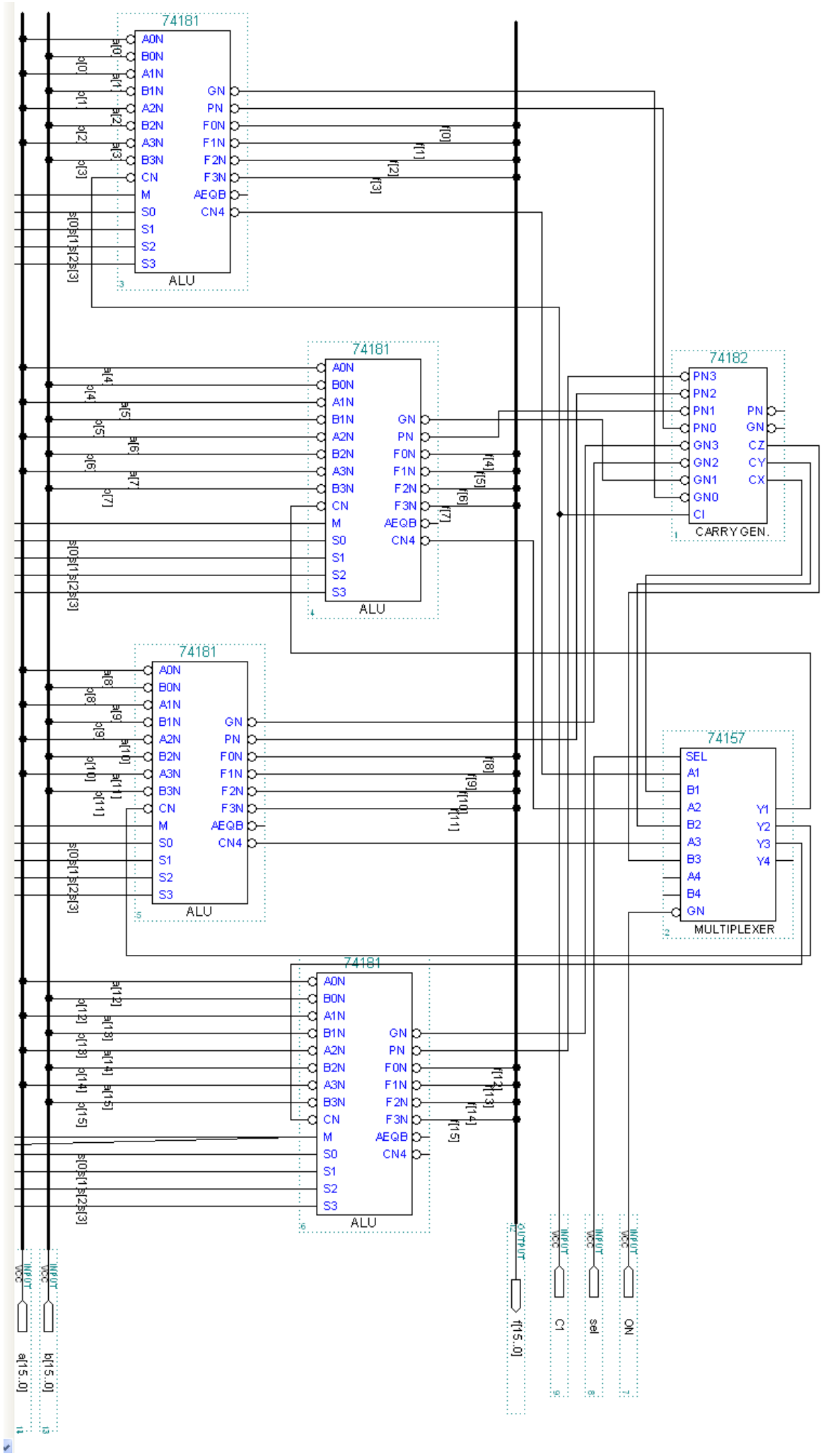
如上图所示，四片 74181 分别负责 0～3、4～7、8～11、12～15 各组的计算，然后通过 74182、74157 生成超前进位。

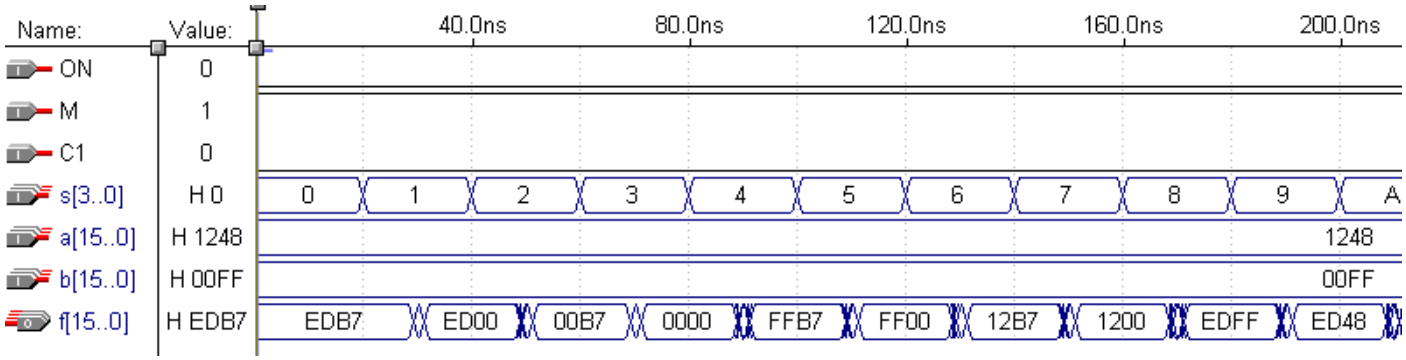
五、实验电路图与仿真结果

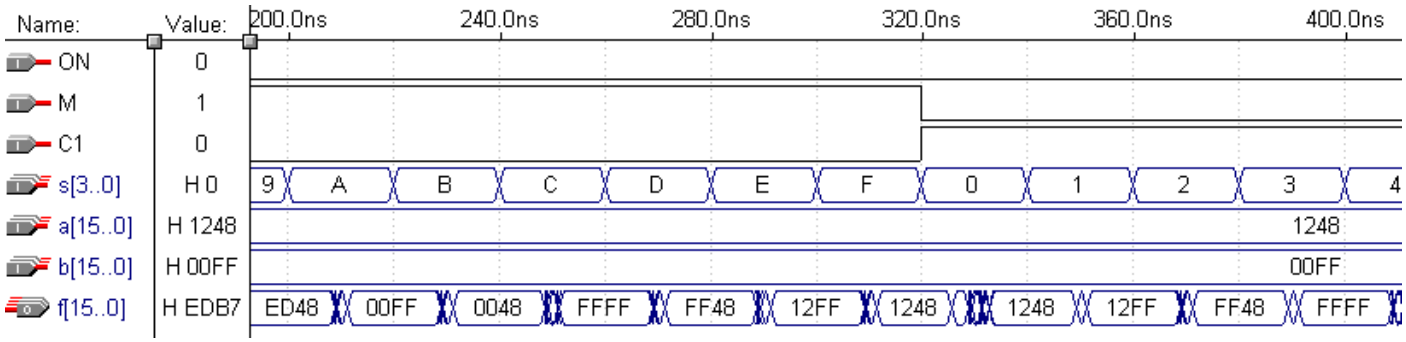
实验电路是按照逻辑设计，根据各芯片的针脚连接方式绘出的。

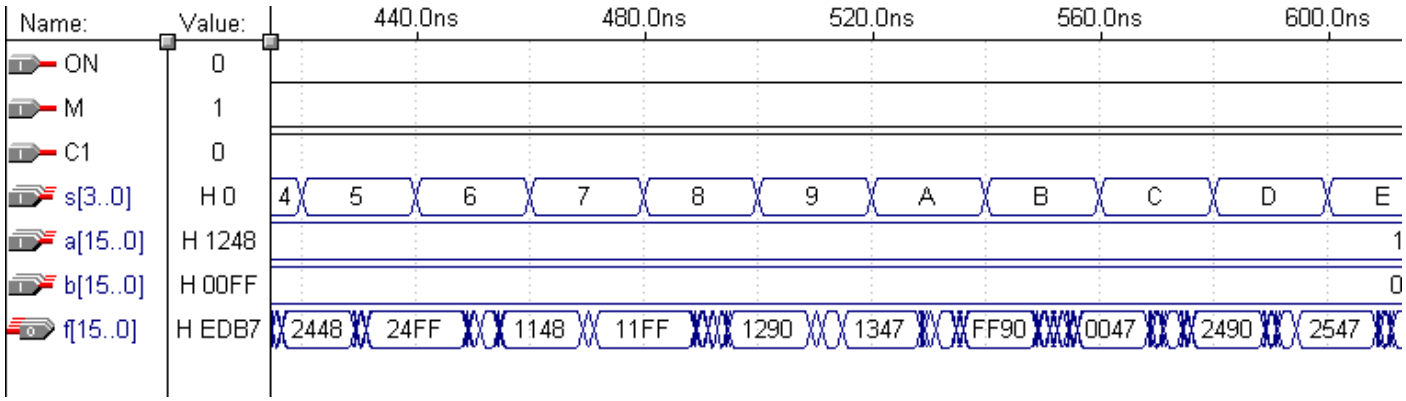
通过本实验，了解了总线、线路标号的使用方法。总线和线路标号可以极大地简化电路，使得复杂电路的各针脚不至于杂乱无章地纠缠在一起，而是以逻辑上有序的方式排列起来，简化了电路图的绘制流程，使电路图简洁明了，不易出错。

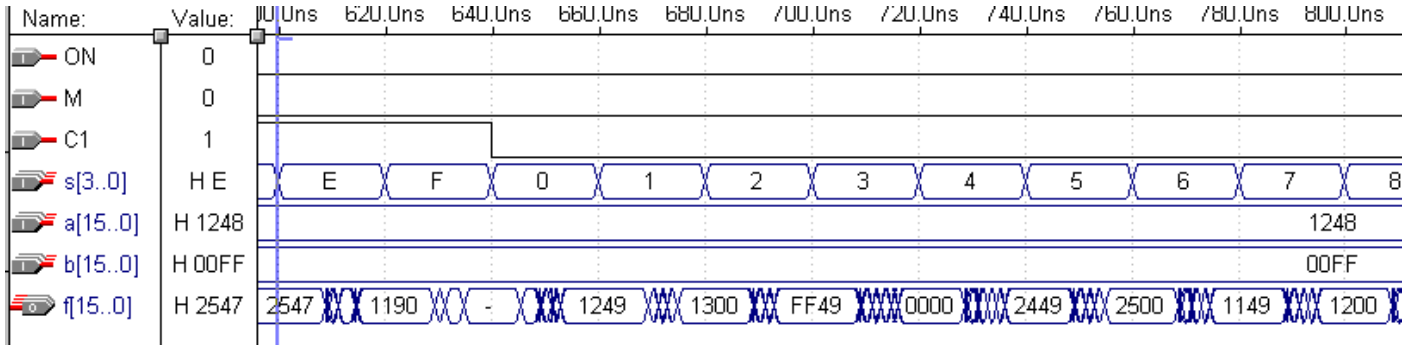
在连线的过程中，需要注意各芯片管脚的相对顺序，以免发生失误。

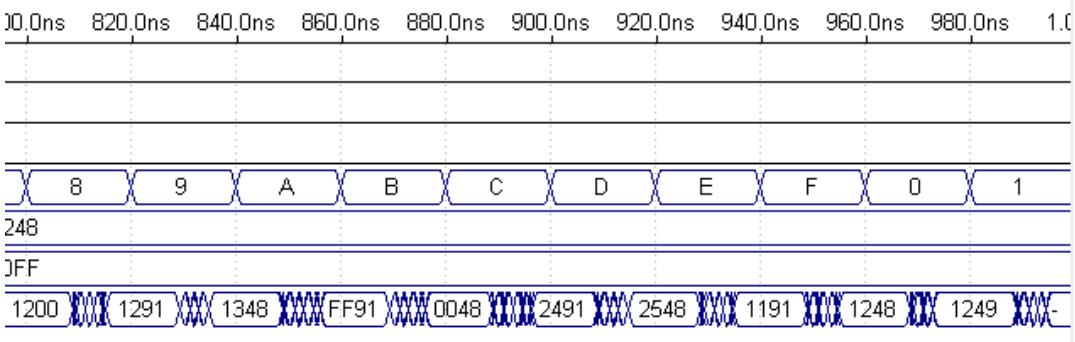

74181
A0N
B0N
A1N
B1N
A2N
B2N
A3N
B3N
CN
M
S0
S1
S2
S3
GN
PN
F0N
F1N
F2N
F3N
AEQB
CN4
ALU
74182
PN3
PN2
PN1
PN0
GN3
GN2
GN1
GN0
CI
PN
GN
CZ
CY
CX
CARRY GEN.
74157
SEL
A1
B1
A2
B2
A3
B3
A4
B4
GN
Y1
Y2
Y3
Y4
MULTIPLEXER
INPUT
VCC
OUTPUT
a[15..0]
b[15..0]
f[15..0]
C1
sel
ON


Name:
Value:
40.0ns
80.0ns
120.0ns
160.0ns
200.0ns
ON
0
M
1
C1
0
s[3..0]
H 0
0 1 2 3 4 5 6 7 8 9 A
a[15..0]
H 1248
1248
b[15..0]
H 00FF
00FF
f[15..0]
H EDB7
EDB7 ED00 00B7 0000 FFB7 FF00 12B7 1200 EDFF ED48


Name:
Value:
200.0ns
240.0ns
280.0ns
320.0ns
360.0ns
400.0ns
ON
0
M
1
C1
0
s[3..0]
H 0
9 A B C D E F 0 1 2 3 4
a[15..0]
H 1248
1248
b[15..0]
H 00FF
00FF
f[15..0]
H EDB7
ED48 00FF 0048 FFFF FF48 12FF 1248 1248 12FF FF48 FFFF


Name:
Value:
440.0ns
480.0ns
520.0ns
560.0ns
600.0ns
ON
0
M
1
C1
0
s[3..0]
H 0
4 5 6 7 8 9 A B C D E
a[15..0]
H 1248
1
b[15..0]
H 00FF
0
f[15..0]
H EDB7
2448 24FF 1148 11FF 1290 1347 FF90 0047 2490 2547


Name:
Value:
ON
0
M
0
C1
1
s[3..0]
H E
E F 0 1 2 3 4 5 6 7 8
a[15..0]
H 1248
1248
b[15..0]
H 00FF
00FF
f[15..0]
H 2547
2547 1190 - 1249 1300 FF49 0000 2449 2500 1149 1200


820.0ns
840.0ns
860.0ns
880.0ns
900.0ns
920.0ns
940.0ns
960.0ns
980.0ns
8 9 A B C D E F 0 1
248
FF
1200 1291 1348 FF91 0048 2491 2548 1191 1248 1249

仿真结果解释：将 a、b 设为 0x1248 与 0x00FF 不变，依次在 M=1；M=0，Cn=1；M=0，Cn=0 三种情况下设置操作码 s[3..0]的 16 种情况，经检查稳定状态（非毛刺震荡期间）的 48 个值均符合功能表。

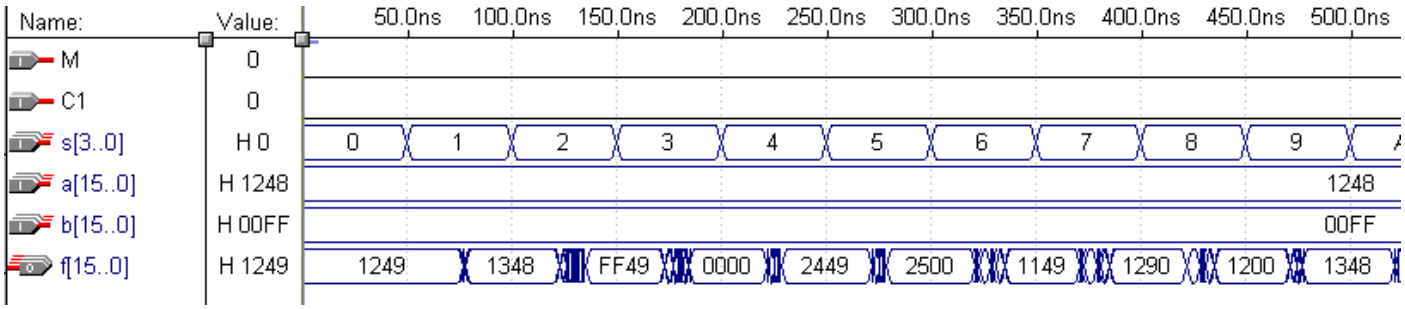
六、Verilog HDL 与仿真结果

用 Verilog HDL 对 74181 逻辑电路所实现逻辑功能进行逐个描述。

```
module alu1(M,C1,s,a,b,f);
input M, C1;
input [3:0]s;
input [15:0]a;
input [15:0]b;
output [15:0]f;
reg [15:0]t;

always@(M or C1 or s or a or b)
   begin
   if(M) //logical
      case(s)
         4'b0000: t = ~a;
         4'b0001: t = ~(a | b);
         4'b0010: t = ~a & b;
         4'b0011: t = 0;
         4'b0100: t = ~(a & b);
         4'b0101: t = ~b;
         4'b0110: t = a ^ b;
         4'b0111: t = a & (~b);
         4'b1000: t = ~a | b;
         4'b1001: t = ~(a ^ b);
         4'b1010: t = b;
         4'b1011: t = a & b;
         4'b1100: t = 1;
         4'b1101: t = a | ~b;
         4'b1110: t = a | b;
         4'b1111: t = a;
      endcase
   else // arithmetic
   if(!C1) // carry enabled
      case(s)
         4'b0000: t=a +1;
         4'b0001: t=a|b +1;
         4'b0010: t=a|(~b) +1;
         4'b0011: t=0;
         4'b0100: t=a+(a&~b) +1;
         4'b0101: t=(a|b)+(a&~b) +1;
         4'b0110: t=a-b;
         4'b0111: t=a+(a&b);
         4'b1000: t=a&~b +1;
         4'b1001: t=a+b +1;
         4'b1010: t=(a|~b)+a&b +1;
```

```
            4'b1011: t=(a&b);
            4'b1100: t=a+a +1;
            4'b1101: t=(a|b)+a +1;
            4'b1110: t=(a|~b)+a +1;
            4'b1111: t=a;
        endcase
    else // carry disabled
        case(s)
            4'b0000: t=a;
            4'b0001: t=a|b;
            4'b0010: t=a|~b;
            4'b0011: t=-1;
            4'b0100: t=a^(a&~b);
            4'b0101: t=(a|b)^(a&~b);
            4'b0110: t=a^(~b)^(-1);
            4'b0111: t=(a&~b)^(-1);
            4'b1000: t=a^(a&b);
            4'b1001: t=a^b;
            4'b1010: t=(a|~b)^(a&b);
            4'b1011: t=(a&b)^(-1);
            4'b1100: t=0;//a^a
            4'b1101: t=(a|b)^a;
            4'b1110: t=(a|~b)^a;
            4'b1111: t=a^(-1);
        endcase
    end

assign f = t;
endmodule
```

在编译时，提示元件数目过多（700 多个，超过 512 个），因此需要使用更大的封装，这里使用了 MAX9000。



七、实验总结

从结果上看，电路图与 Verilog HDL 实现的逻辑功能相同，但通过 Verilog 制作的电路在模拟时延迟较大。电路图的设计中延迟约为 10ns，而 Verilog 设计中延迟约为 25ns。这可能是因为 Verilog 在编译成芯片设计时使用的是行波进位方式，从而输出的延迟较大，且毛刺较多；电路图设计中采用了 74181 和 74182、74157 实现了超前进位，因此输出延迟较小。

如果要从头设计 ALU（如 74181 芯片），应当注意各逻辑功能之间的关系，尽量减少逻辑冗余。此外，使用超前进位机制可以提高响应速度，减少电路不稳定的时间，进而可以提高 ALU 单位时间内可以执行的运算次数。

实验七 2K*8 存储器的设计

一、实验目的

了解存贮器的基本构成，掌握扩大存贮器容量和存贮器字长的设计方法，了解 2114 芯片基本功能

二、实验要求

用 1K×4 的 MOS SRAM2114,74161,74244 等集成电路构成容量为 2K×8 的功能较完整的存储器。

三、实验原理

本实验是选做实验，没有提供 2114 芯片，Max+Plus 软件中也没有提供 2114 芯片。但结合软件帮助文件，在 Max+Plus 软件的 mega_lpm 元件库找到了功能相似的存储器 LPM_RAM_DP。

四、逻辑设计

将四片 LPM_RAM_DP 芯片分别设置为字长为 4，地址长为 10，即 1K*4 的存储器。

四个存储器分别负责：

①地址高位为 1 的 1K 内存，每个字的 0~3 位

②地址高位为 1 的 1K 内存，每个字的 4~7 位

③地址高位为 0 的 1K 内存，每个字的 0~3 位

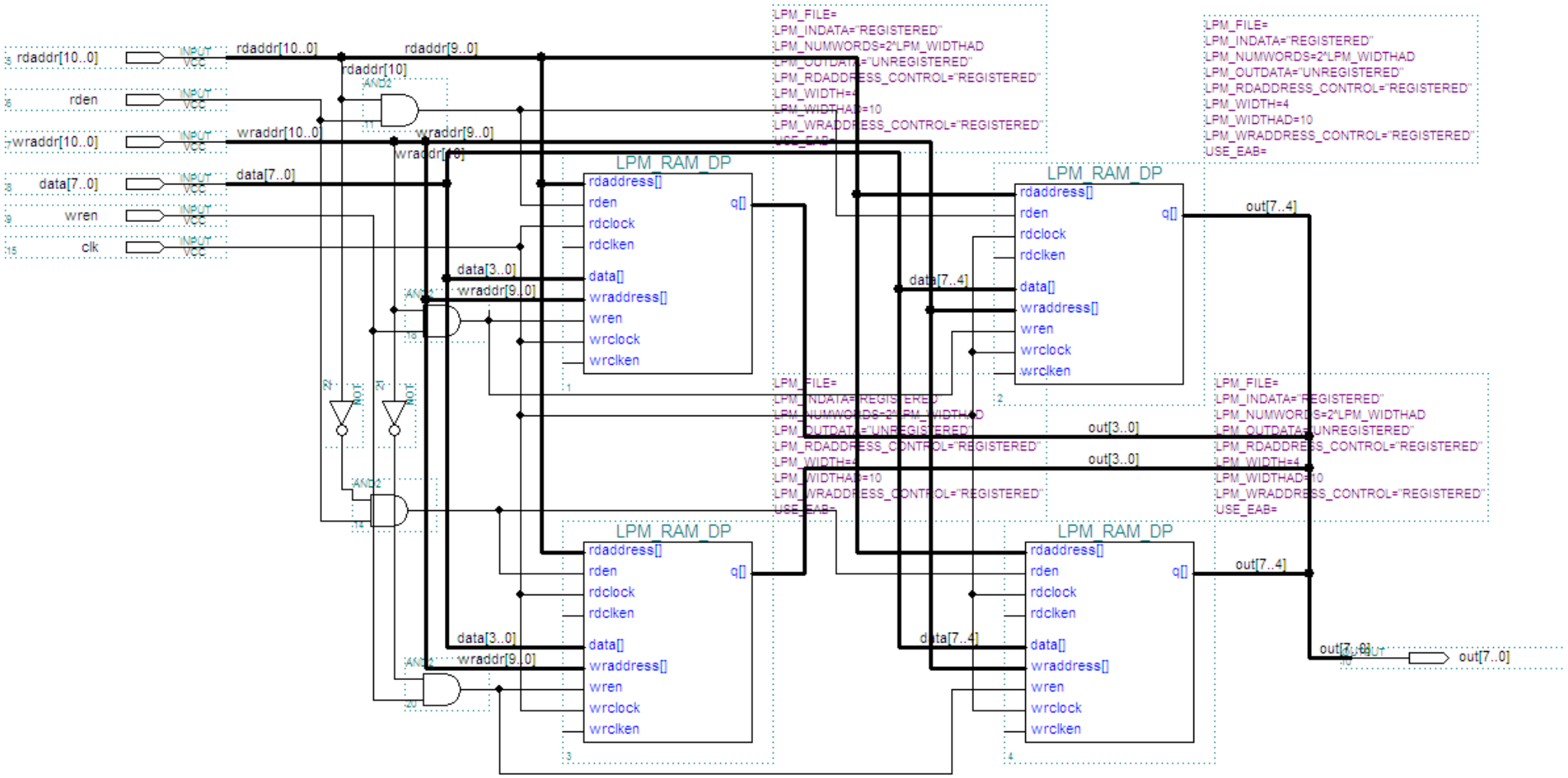
④地址高位为 0 的 1K 内存，每个字的 4~7 位。

这样，将输入 2K*8 存储器的 11 根地址线的最高位与 OE/WE 通过与门，控制对应的 1K*4 存储器是否允许读写；剩余 10 位直接输入到 1K*4 存储器中；将输入 2K*8 存储器的 8 位数据线的前 4 位和后 4 位分别连接到对应的 1K*4 存储器。

在电路图接线中，总线的使用大大简化了电路，也使得图幅更加清晰，是个较好的方法。

在编译过程中，提示出错，但错误提示是在 LPM_RAM_DP 芯片的内部描述语言中，无法解决。可能是对 LPM_RAM_DP 的使用方式和接线规范不够了解所致。从逻辑上讲，如果把 LPM_RAM_DP 芯片看作是一个理想黑盒，电路图中所描述的逻辑功能确为 2K*8 存储器。

（电路图见下页）


rdaddr[10..0]
rden
wraddr[10..0]
data[7..0]
wren
clk
INPUT
VCC
rdaddr[9..0]
rdaddr[10]
wraddr[9..0]
wraddr[10]
data[3..0]
AND2
NOT
LPM_RAM_DP
rdaddress[]
rden
rdclock
rdclken
q[]
data[]
wraddress[]
wren
wrclock
wrclken
LPM_FILE=
LPM_INDATA="REGISTERED"
LPM_NUMWORDS=2^LPM_WIDTHAD
LPM_OUTDATA="UNREGISTERED"
LPM_RDADDRESS_CONTROL="REGISTERED"
LPM_WIDTH=4
LPM_WIDTHAD=10
LPM_WRADDRESS_CONTROL="REGISTERED"
USE_EAB=
data[7..4]
out[7..4]
out[3..0]
out[7..0]
OUTPUT

五、Verilog 设计

在 Verilog 设计中，为简便起见，直接声明了一个大数组用于存储，每 8 位为一组，自行处理位运算，没有使用 1K*4 存储器单元。

由于元件数过多，无法找到合适的芯片，因此无法仿真。

Verilog HDL 程序：

```
module memory(rdaddr,rden,wraddr,data,wren,clk,out);
input [10:0]rdaddr;
input rden;
input [10:0]wraddr;
input [7:0]data;
input wren;
input clk;
output [7:0]out;
reg [7:0]p; //prepare to go out
reg [16383:0]m;//memory 2K*8
reg [13:0]base;

always@(posedge clk)
   begin
   if(rden)
      begin
      base = rdaddr<<3;
      p[0] = m[base];
      p[1] = m[base+1];
      p[2] = m[base+2];
      p[3] = m[base+3];
      p[4] = m[base+4];
      p[5] = m[base+5];
      p[6] = m[base+6];
      p[7] = m[base+7];
      end
   if(wren)
      begin
      base = wraddr<<3;
      m[base] = data[0];
      m[base+1] = data[1];
      m[base+2] = data[2];
      m[base+3] = data[3];
      m[base+4] = data[4];
      m[base+5] = data[5];
      m[base+6] = data[6];
      m[base+7] = data[7];
      end
   end
assign out = p;
endmodule
```