存储器的设计

寻址存储器（RAM 和 ROM）

ROM 和 RAM 属于通用大规模器件，一般不需要自行设计，特别是采用 PLD 器件进行设计时；

但是在数字系统中，有时也需要设计一些小型的存储器件，用于特定的用途：临时存放数据，构成查表运算等。

此类器件的特点为地址与存储内容直接对应，设计时将输入地址作为给出输出内容的条件；

RAM　随机存储器

RAM 的用途是存储数据，其指标为存储容量和字长；

RAM 的内部可以分为地址译码和存储单元两部分；

外部端口为：　wr　写读控制　　cs　片选

d　数据端口　　adr　地址端口

例 16x8 位 RAM 的设计

设计思想：

将每个 8 位数组作为一个字（word）；总共存储 16 个字；

将 ram 作为由 16 个字构成的数组，以地址为下标；

通过读写控制模式实现对特定地址上字的读出或写入；

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity kram is
  port ( clk,wr,cs: in std_logic;
              d: inout std_logic_vector(7 downto 0);
              adr: in std_logic_vector(3 downto 0));
end kram;

architecture beh of kram is
subtype word is std_logic_vector(7 downto 0);
type memory is array (0 to 15) of word;
signal adr_in:integer range 0 to 15;
signal sram:memory;
begin
adr_in<=conv_integer (adr);
process(clk)
   begin
      if(clk'event and clk='1') then
         if (cs='1'and wr='1') then
         sram (adr_in)<=d;
         end if;
         if (cs='1'and wr='0' ) then
         d<=sram (adr_in);
         end if;
      end if;
end process;
end beh;
```


RAM 的数据端口通常为 inout 模式，设置仿真输入时只能在写入时将信号加到该输入端口上，在其他时候输入应设置为高阻态；

RAM 设计时需要注意器件的大小，一个 16x8 位的 RAM 大约占用 200 个门，而 256x16 的 RAM 则需要 6200 门以上，

因此大规模 RAM 不适合于采用 PLD 设计，最好采用通用器件；

ROM 只读存储器

ROM 的内容是初始设计电路时就写入到内部的，通常采用电路的固定结构来实现存储；ROM 只需设置数据输出端口和地址输入端口；

例 1　简单 ROM 的设计（16x8 位 ROM）

设计思想：采用二进制译码器的设计方式，但将每个输入组态对应的输出与一组存储数据对应起来；


```
library ieee;
use ieee.std_logic_1164.all;

entity rom is
   port(dataout: out std_logic_vector(7 downto 0);
        addr: in std_logic_vector(3 downto 0);
        ce: in std_logic);
end rom;

architecture d of rom    is
   begin
      dataout <= "00001111" when addr ="0000" and ce='0' else
                 "11110000" when addr ="0001" and ce='0' else
                 "11001100" when addr ="0010" and ce='0' else
                 "00110011" when addr ="0011" and ce='0' else
                 "10101010" when addr ="0100" and ce='0' else
                 "01010101" when addr ="0101" and ce='0' else
                 "10011001" when addr ="0110" and ce='0' else
```

```
                "01100110" when addr ="0111" and ce='0' else
                "00000000" when addr ="1000" and ce='0' else
                "11111111" when addr ="1001" and ce='0' else
                "00010001" when addr ="1010" and ce='0' else
                "10001000" when addr ="1011" and ce='0' else
                "10011001" when addr ="1100" and ce='0' else
                "01100110" when addr ="1101" and ce='0' else
                "10100110" when addr ="1110" and ce='0' else
                "01100111" when addr ="1111" and ce='0' else
                   "XXXXXXXX";
end d;
```

对于较大的 ROM，可以采用结构设计的方法，直接调用参数化模块进行设计；

例　ROM 的 LPM 设计　（256x8 位 ROM）

```
library ieee;
use ieee.std_logic_1164.all;
library lpm;
use lpm.lpm_components.all;

entity romlpm is
   port(address: in std_logic_vector(7 downto 0);
        inclock: in std_logic;
        q: out std_logic_vector(7 downto 0));
end romlpm;

architecture str of romlpm    is
signal sub_wire0:std_logic_vector(7 downto 0);
begin
   q<=sub_wire0(7 downto 0);
   lpm_rom_component:lpm_rom
generic map(
      lpm_width =>8,
      lpm_widthad =>8,
```

```
    lpm_address_control=>"registered",
    lpm_outdata => "unregistered",
    lpm_file=> "krom2.mif")
port map(
    address=>address,
    inclock=>inclock,
    q =>sub_wire0);
end str;
```

ROM 的初始化

在 ROM 的设计中，必须要预先设置好数据存储文件，这是一种以.mif 为后缀的文本文件，在任何文本编辑器中，按如下文件格式写入：

```
DEPTH = 16;          字线数量
WIDTH = 4;           位线数量
ADDRESS_RADIX = HEX;    地址与数据的表达类型
DATA_RADIX = HEX;   可以选择：HEX   OCT   DEC   BIN
CONTENT              存储内容    地址 ：数据；
BEGIN
[0..F]     :     3;
2          :     4 5 6 7;
8          :     F E 5;
END ;
```


文件写完后，保存为.mif 即可。

例：4 位查表式乘法器设计

功能：将两个 4 位二进制数 A 和 B 相乘，输出乘积结果 C（8 位二进制数）；

设计方案：采用 256x8 位 ROM 实现，8 位地址输入（高 4 位为 A，低 4 位为 B），256 个存储字；8 位数据输出；

数据存储文件（krom2.mif）： 填写相应的乘法表即可

```
    depth = 256; width = 8;
```

```
    address_radix = hex; data_radix = hex;
    content
    begin
[00..0f] : 00 ;
10 : 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f;
20 : 00 02 04 06 08 0a 0c 0e 10 12 14 16 18 1a 1c 1e;
30 : 00 03 06 09 0c 0f 12 15 18 1b 1e 21 24 27 2a 2d;
40 : 00 04 08 0c 10 14 18 1c 20 24 28 2c 30 34 38 3c;
50 : 00 05 0a 0f 14 19 1d 23 28 2d 32 37 3c 41 46 4b;
60 : 00 06 0c 12 18 1e 24 2a 30 36 3c 42 48 4e 54 5a;
70 : 00 07 0e 15 1c 23 2a 31 38 3f 46 4d 54 5b 62 69;
80 : 00 08 10 18 20 28 30 38 40 48 50 58 60 68 70 78;
90 : 00 09 12 1b 24 2d 36 3f 48 51 5a 63 6c 75 7e 87;
a0 : 00 0a 14 1e 28 32 3c 46 50 5a 64 6e 78 82 8c 96;
b0 : 00 0b 16 21 2c 37 42 4d 58 63 6e 79 84 8f 9a a5;
c0 : 00 0c 18 24 30 3c 48 54 60 6c 78 84 90 9c a8 b4;
d0 : 00 0d 1a 27 34 41 4e 5b 68 75 82 8f 9c a9 b6 c3;
e0 : 00 0e 1c 2a 38 46 54 62 70 7e 8c 9a a8 b6 c4 d2;
f0 : 00 0f 1e 2d 3c 4b 5a 66 78 87 96 a5 b4 c3 d2 e1;
end;
```


该乘法器 ROM 设计完毕后，将其设置为符号文件，将来就可以在 VHDL 程序中用 component 语句调用了。

在 maxplus2 的仿真器窗口内，也可以直接生成 ROM 的初始化文件，其步骤如下：

选择 Initialize/Initialize Memory…；

按 ROM 地址输入数据；

存盘即可生成指定的 mif 文件（文件名已经在结构体内指明）；

顺序存储器（堆栈和 FIFO）

顺序存储器的特点是不设置地址，所有数据的写入和读出都按顺序进行；数据写入或读出时通常会进行移位操作；在设计时必须考虑各存储单元的存储状态；

堆栈（后进先出存储器）

要求：存入数据按顺序排放，存储器全满时给出信号并拒绝继续存入；读出时按后进先出原则；存储数据一旦读出就从存储器中消失；

设计思想：

将每个存储单元设置为字（word）；存储器整体作为由字构成的数组；为每个字设置一个标记（flag），用以表达该存储单元是否已经存放了数据；每写入或读出一个数据时，字的数组内容进行相应的移动，标记也做相应的变化；

程序示例：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity stack is
   port(datain: in std_logic_vector(7 downto 0);
        push,pop,reset,clk:in std_logic;
        stackfull:out std_logic;
        dataout: buffer std_logic_vector(7 downto 0));
end stack;
```

```
architecture b of stack    is
type arraylogic is array(15 downto 0) of std_logic_vector(7
downto 0);
signal data :arraylogic;
signal stackflag:std_logic_vector(15 downto 0);
begin
   stackfull<=stackflag(0);
process(clk,reset,pop,push)
   variable selfunction: std_logic_vector(1 downto 0);
   begin
   selfunction:=push & pop;
   if reset='1' then
      stackflag<=(others=>'0'); dataout<=(others=>'0');
      for i in 0 to 15 loop
         data(i)<="00000000";
      end loop;
   elsif clk'event and clk='1' then
      case selfunction is
         when "10" =>
            if stackflag(0)='0' then
            data(15)<=datain;
            stackflag<='1'&stackflag(15 downto 1);
            for i in 0 to 14 loop
            data(i)<=data(i+1);
            end loop;
            end if;
         when "01" =>
            dataout<=data(15);
            stackflag<=stackflag(14 downto 0)&'0';
            for i in 15 downto 1 loop
            data(i)<=data(i-1);
            end loop;
         when others=>null;
      end case;
```

```
    end if;
end process;
end b;
```

以上程序是基于移位寄存器的设计思想；若基于存储器的设计思想，则可以设置一个指针（point），表示出当前写入或读出单元的地址，使这种地址进行顺序变化，就可以实现数据的顺序读出或写入；

程序示例

library 说明和 entity 设计与上面程序完全相同；

```
architecture b of stack    is
type  arraylogic  is  array(15  downto  0)  of  std_logic_vector(7
downto 0);
signal data :arraylogic;
begin
process(clk,reset,pop,push)
    variable p:natural range 0 to 15;
    variable selfunction: std_logic_vector(1 downto 0);
    variable s:std_logic;
    begin
stackfull<=s;
selfunction:=push & pop;
    if reset='1' then
        p:=0;dataout<=(others=>'0');s:='0';
        for i in 0 to 15 loop
            data(i)<="00000000";
        end loop;
    elsif clk'event and clk='1' then
        if p<15 and selfunction="10" then
                data(p)<=datain;      p:=p+1;
        end if;
        if p=15 and selfunction="10" and s='0' then
```

```
            data(p)<=datain;      s:='1';
        end if;
        if p>0 and selfunction="01" and s='0' then
            p:=p-1;    dataout<=data(p);
        end if;
        if p=15 and selfunction="01" and s='1' then
            dataout<=data(p);    s:='0';
        end if;
    end if;
end process;
end b;
```

FIFO（先进先出存储器）

要求：存入数据按顺序排放，存储器全满时给出信号并拒绝继续存入，全空时也给出信号并拒绝读出；读出时按先进先出原则；存储数据一旦读出就从存储器中消失；

设计思想：

结合堆栈指针的设计思想，采用环行寄存器方式进行设计；分别设置写入指针 wp 和读出指针 rp，标记下一个写入地址和读出地址；地址随写入或读出过程顺序变动；设置全空标记和全满标记以避免读出或写入的错误；

设计时需要注意处理好从地址最高位到地址最地位的变化；

程序示例

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
```

```
entity kfifo is
   port(datain: in std_logic_vector(7 downto 0);
            push,pop,reset,clk:in std_logic;
            full,empty:out std_logic;
            dataout: out std_logic_vector(7 downto 0));
end kfifo;

architecture b of kfifo    is
type  arraylogic  is  array(15  downto  0)  of  std_logic_vector(7
downto 0);
signal data :arraylogic;
signal  fi,ei:std_logic;--为全满全空设置内部信号，以便内部调
用；
signal wp,rp:natural range 0 to 15;    --指针；
begin
process(clk,reset,pop,push)
   variable selfunction: std_logic_vector(1 downto 0);
   begin
   full<=fi;empty<=ei;
   selfunction:=push & pop;

if reset='1' then
wp<=0;rp<=0;fi<='0';ei<='1'; --初始指针处于 0 位；
dataout<=(others=>'0');
      for i in 0 to 15 loop
         data(i)<="00000000";
      end loop;

elsif clk'event and clk='1' then

--write;
       if fi='0' and selfunction="10" and wp<15 then
            data(wp)<=datain;
```

```
            wp<=wp+1;
            if wp=rp then fi<='1';end if;
            if ei='1' then ei<='0';end if;
        end if;

        if fi='0' and selfunction="10" and wp=15 then
            data(wp)<=datain;
            wp<=0;
            if wp=rp then fi<='1';end if;
            if ei='1' then ei<='0';end if;
        end if;
--read;
        if ei='0' and selfunction="01" and rp<15 then
            dataout<=data(rp);
            rp<=rp+1;
            if wp=rp then ei<='1';end if;
            if fi='1' then fi<='0';end if;
        end if;

        if ei='0' and selfunction="01" and rp=15 then
            dataout<=data(rp);
            rp<=0;
            if wp=rp then ei<='1';end if;
            if fi='1' then fi<='0';end if;
        end if;
end if;
end process;
end b;
```


采用参数化模块直接形成 FIFO 的设计

由于各类存储器通常都会占用较多的硬件资源，直接采用已得到优化的参数化模块通常可以取得较好的效果；

程序示例

```
library ieee;
```

```
use ieee.std_logic_1164.all;
library lpm;
use lpm.lpm_components.all;

entity fifo2 is
   port(data: in std_logic_vector(7 downto 0);
        wrreq,rdreq,clock,aclr:in std_logic;
        q: out std_logic_vector(7 downto 0);
        full,empty: out std_logic;
        usedw: out std_logic_vector(3 downto 0));
end fifo2;

architecture str of fifo2    is
signal sub_wire0:std_logic_vector(3 downto 0);
signal sub_wire1:std_logic;
signal sub_wire2:std_logic_vector(7 downto 0);
signal sub_wire3:std_logic;
begin
   usedw<=sub_wire0(3 downto 0);
   empty<=sub_wire1;
   q<=sub_wire2(7 downto 0);
   full<=sub_wire3;
   lpm_fifo_component:lpm_fifo
generic map(
      lpm_width =>8,
      lpm_numwords =>16,
      lpm_widthu=>4,
      lpm_showahead => "off",
      lpm_hint=> "use_eab=on,maximize_speed=5")
port map(
      rdreq=>rdreq,aclr=>aclr,clock=>clock,
      wrreq=>wrreq,data=>data,
      usedw=>sub_wire0,
      empty=>sub_wire1,
```

```
        q =>sub_wire2,
        full=>sub_wire3);
end str;
```