第五讲 存储器设计

- 5.1 随机存取存储器（RAM）设计
- 5.2 只读存储器（ROM）设计
- 5.3 双端口RAM设计
- 5.4 先进先出（FIFO）设计
- 5.5 CRC校验电路设计

存储器分类

- 半导体存储器
  - RAM
    - 双极型
    - 金属氧化物（MOS）
      - 静态：SRAM
      - 动态：DRAM
        - SDRAM
        - DDR SDRAM
        - DDR2 SDRAM
        - DDR3 SDRAM
  - ROM
    - 掩模工艺ROM
    - 可一次编程ROM：PROM
    - 可擦写的PROM
      - EPROM
      - E²PROM
        - E²PROM
        - Flash
  - 其他存储器
    - ◆ 双端口存储器
    - ◆ 先进先出（FIFO）存储器
    - ◆ 铁电存储器

存储器分类

- 半导体存储器
  - RAM
    - 双极型
    - 金属氧化物（MOS）
      - 静态：SRAM
      - 动态：DRAM
  - ROM
    - 掩模工艺ROM
    - 可一次编程ROM：PROM
    - 可擦写的PROM
      - EPROM
      - E^2PROM
        - E^2PROM
        - Flash
  - 其他存储器
    - 双端口存储器
    - 先进先出（FIFO）存储器
    - 铁电存储器

设计方法

- 模块功能与原理分析
- 模块结构与电路模型
- VHDL语言设计实现
- FPGA验证

5.1 静态随机存储器SRAM设计

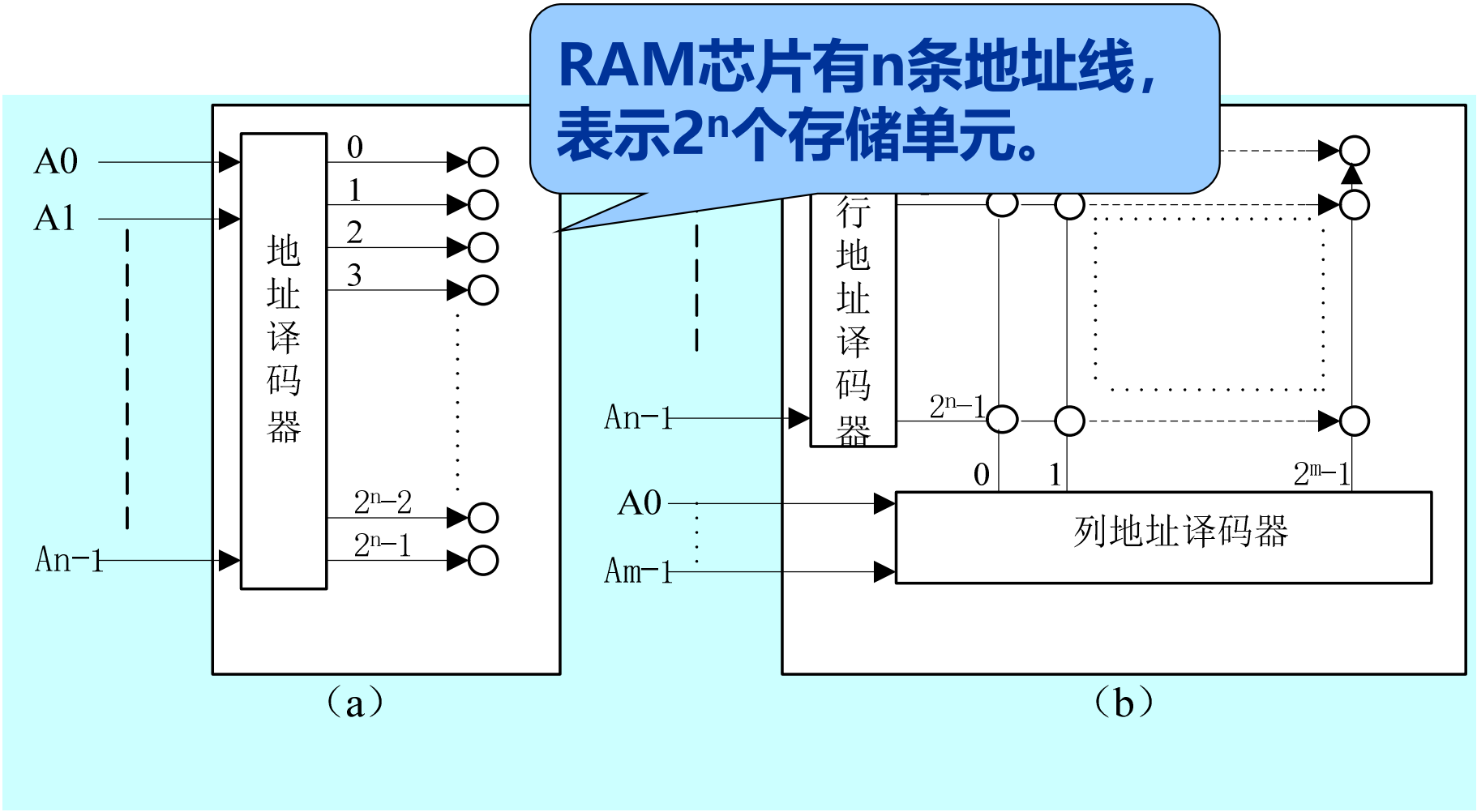
- 数据存储功能
- 地址控制功能
- 写入与读出功能

数据总线

地址总线

控制信号

5.1.1 RAM地址译码方式



一维译码

二维译码

5.1.2 SRAM 6264芯片

- 存储容量8K×8
  - ① A12～A0
  - ② D0～D7
  - ③ $\overline{CS1}$、CS2
  - ④ $\overline{OE}$
  - ⑤ $\overline{WE}$

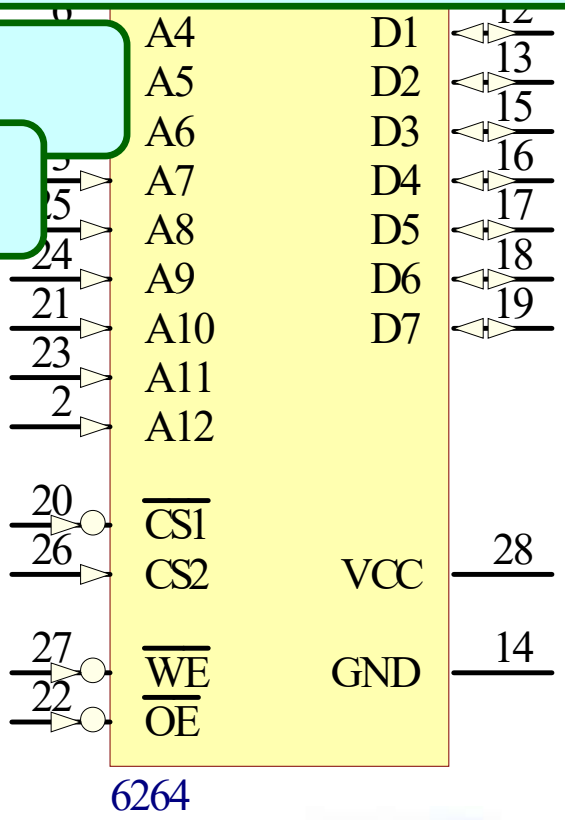
决定存储单元的容量，一般
1K～256M → 地址总线数：10～28

决定存储单元的宽度（位数，bit）

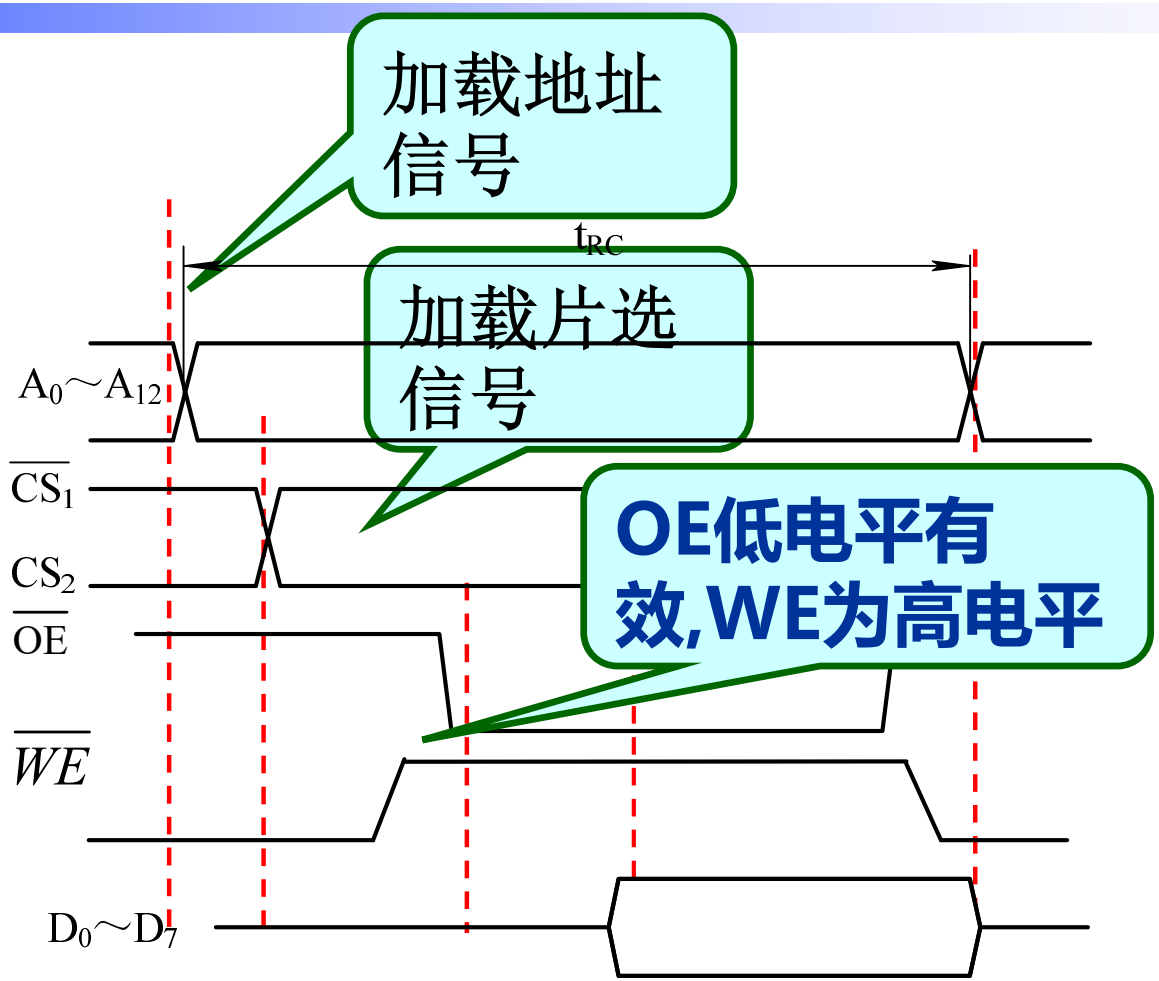
片选 → 地址译码

输出允许（读）

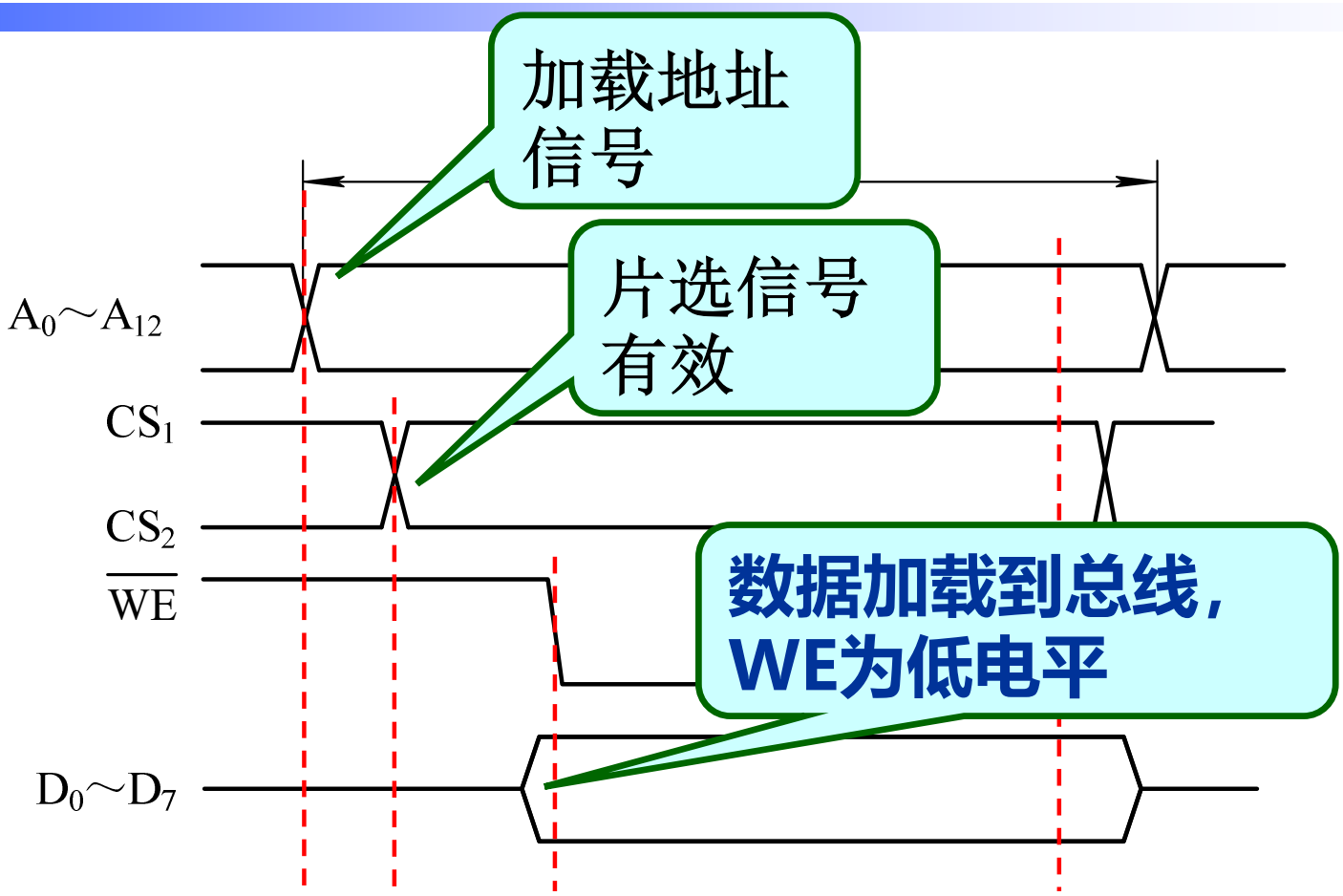
写允许



(1)SRAM 读出时序


加载地址信号
t_{RC}
加载片选信号
$A_0 \sim A_{12}$
$\overline{CS_1}$
CS_2
$\overline{OE}$
OE低电平有效,WE为高电平
$\overline{WE}$
$D_0 \sim D_7$

(2)SRAM 写入时序


加载地址信号
片选信号有效
数据加载到总线，WE为低电平
A_0～A_{12}
CS_1
CS_2
$\overline{WE}$
D_0～D_7

(3)SRAM的VHDL程序实现

- 端口定义

```
PORT(address : IN STD_LOGIC_VECTOR(3 DOWNTO
  0);
       cs , oe , we: IN STD_LOGIC;
       data : INOUT STD_LOGIC_VECTOR(7 DOWNTO
  0));
```

(4)SRAM的VHDL程序实现

```
ARCHITECTURE behav OF ram16x8 IS
      SUBTYPE word IS STD_LOGIC_VECTOR(7 DOWNTO 0);
      TYPE ram_array IS ARRAY (0 TO 15) OF word;
      SIGNAL index : IN INTEGER RANGE 0 TO 15;
      SIGNAL sram_store : ram_array;
BEGIN
     index<= CONV_INTEGER(address);
  PROCESS(address, cs , oe , we , data)
   BEGIN
      IF cs  = '0' THEN
         IF we='1' THEN
            sram_store(index) <= data;
         ELSIF oe  = '0' THEN
            data <=sram_store(index);
         ELSE
            data <= "ZZZZZZZZ";
         END IF;
      ELSE
         data <= "ZZZZZZZZ";
      END IF;
   END PROCESS;
END  behav;
```

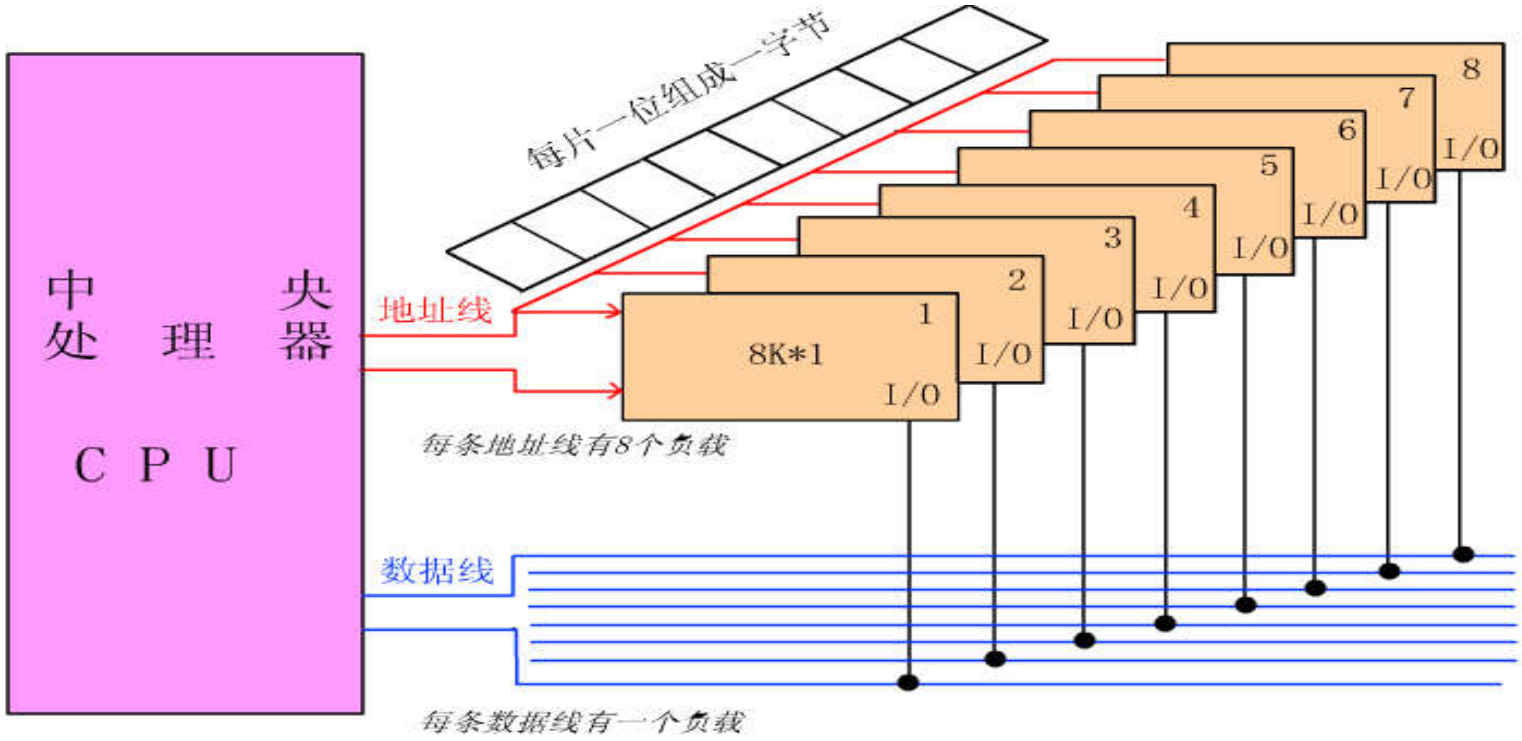
写入数据

读出数据

总线三态

5.1.3 RAM容量扩展

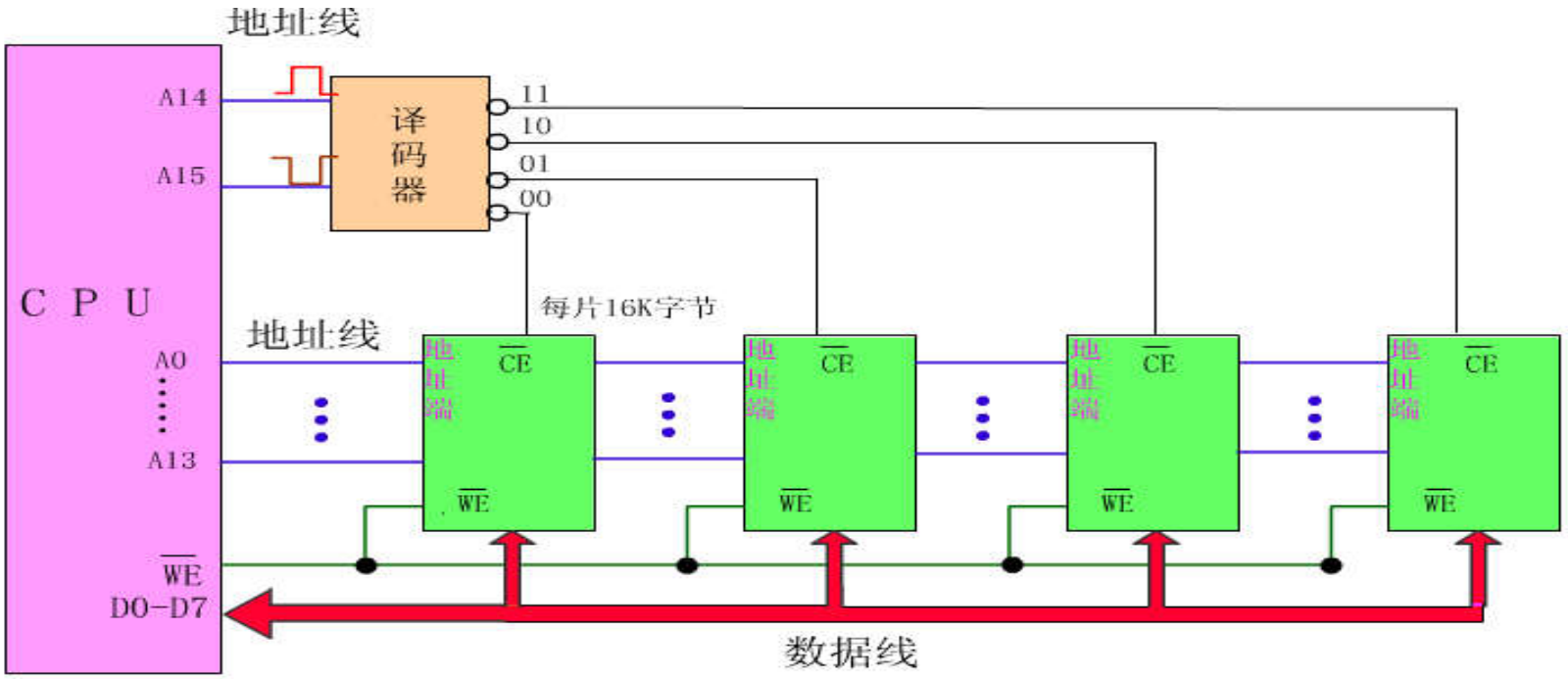
存储器与CPU连接——位扩展法



位扩展法组成8K RAM

5.1.4 随机读写存储器RAM

存储器与CPU连接——字扩展法



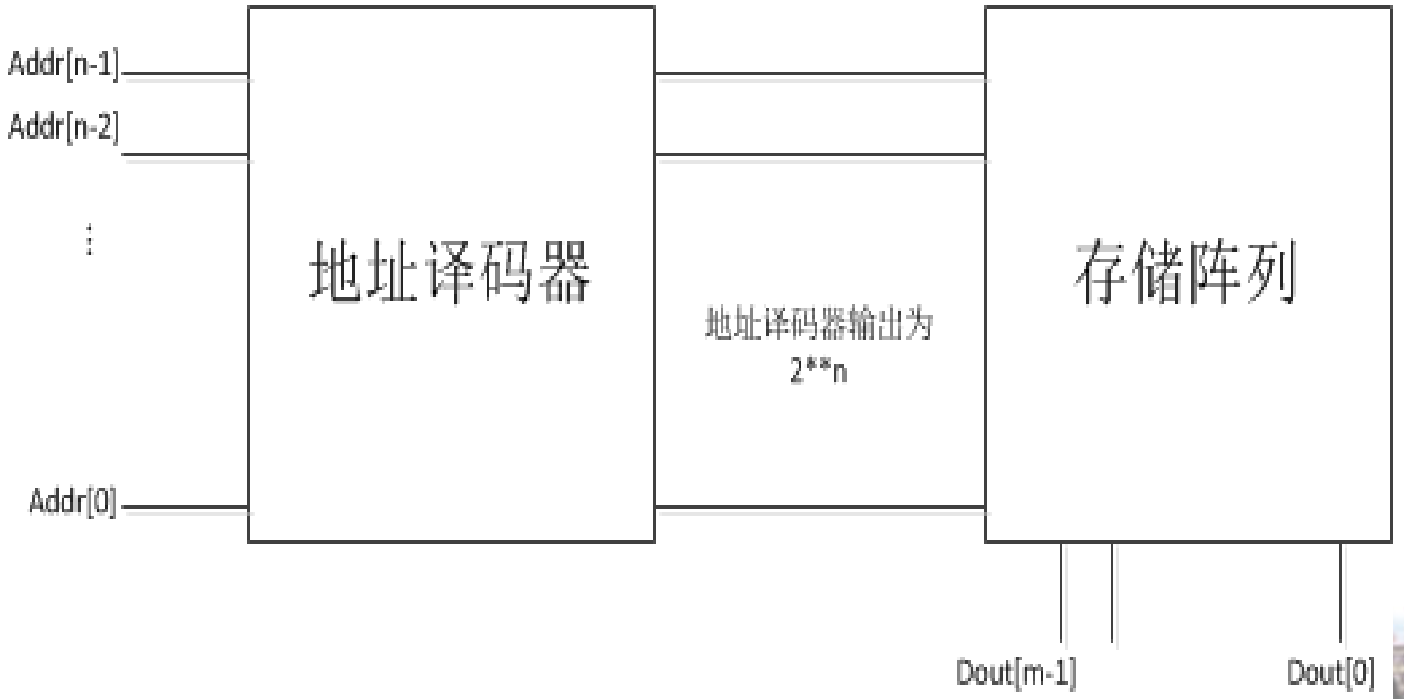
字扩展法组成64K RAM

5.2 只读存储器ROM的设计

- 只读存储器（ROM）的内容是初始设计电路时就写入到内部的，通常用于存储固件。
- ROM主要用于计算机基本输入输出系统（BIOS）的存储和用作嵌入式系统中的程序存储器。
- ROM只需设置数据输出端口和地址输入端口。

5.2.1只读存储器ROM的电路结构

- 存储矩阵
- 地址译码器
- 输出缓冲器



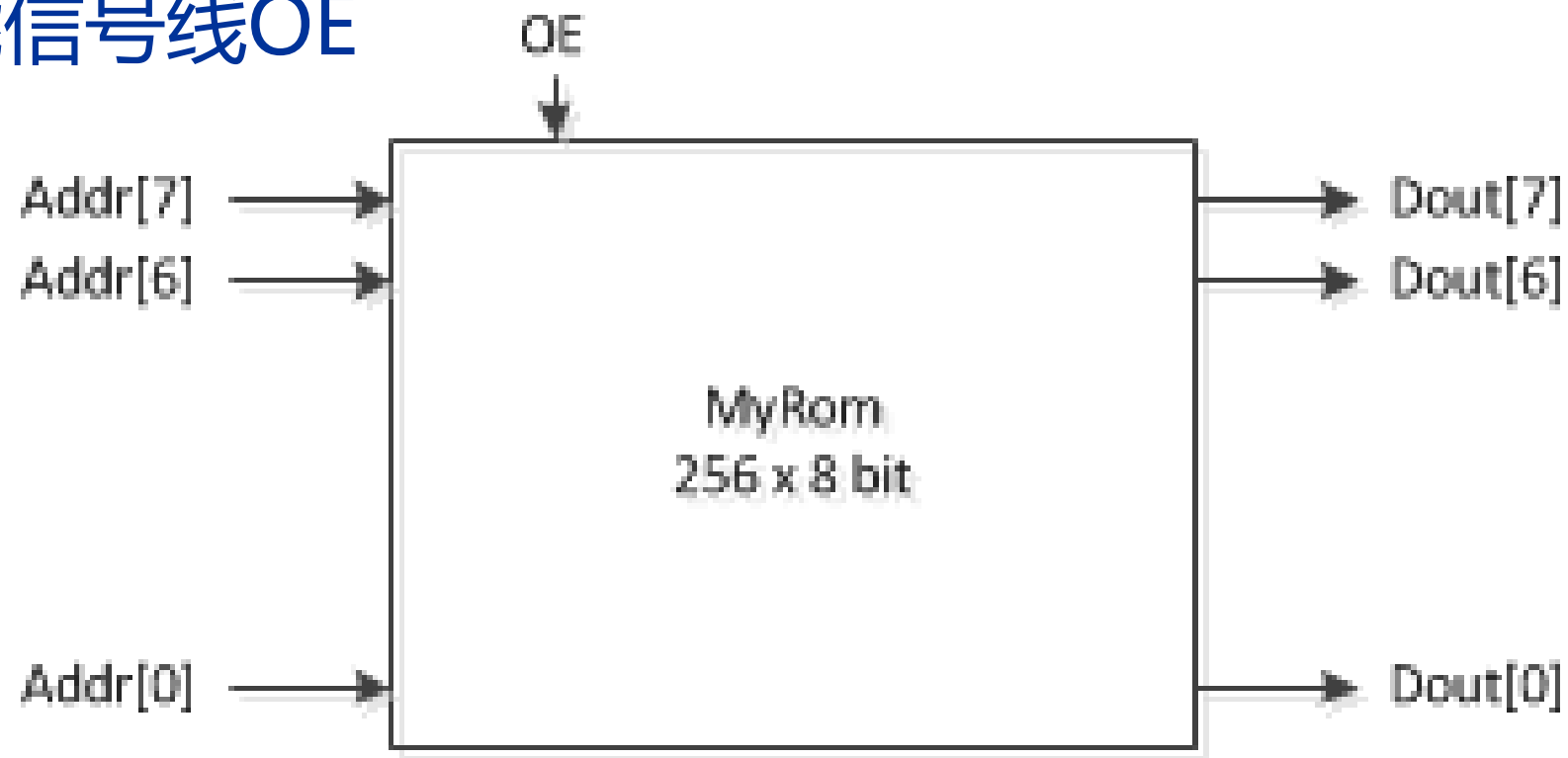
5.2.2 简单 ROM的设计

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

```
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;
```

5.2.3 通用ROM的VHDL设计

- **设计一个容量为256*8bit的ROM**
- 8位地址线Addr[7..0]
- 8位数据输出线Dout[7…0]
- 使能信号线OE



5.2.3 通用ROM的VHDL设计

- VHDL数据对象--文件类型应用

```
library IEEE;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
use std.textio.all;
```

- 端口定义

```
entity MyRom is
  generic (WORDLENGTH : integer := 8;
     ADDRLENGTH : integer := 8);
  port(
    addr:in std_logic_vector(ADDRLENGTH-1 downto 0);
    oe  :in std_logic;
    dout:out std_logic_vector(WORDLENGTH-1 downto 0)
      );
end MyRom;
```

5.2.4 通用ROM的VHDL设计

- **结构体实现**

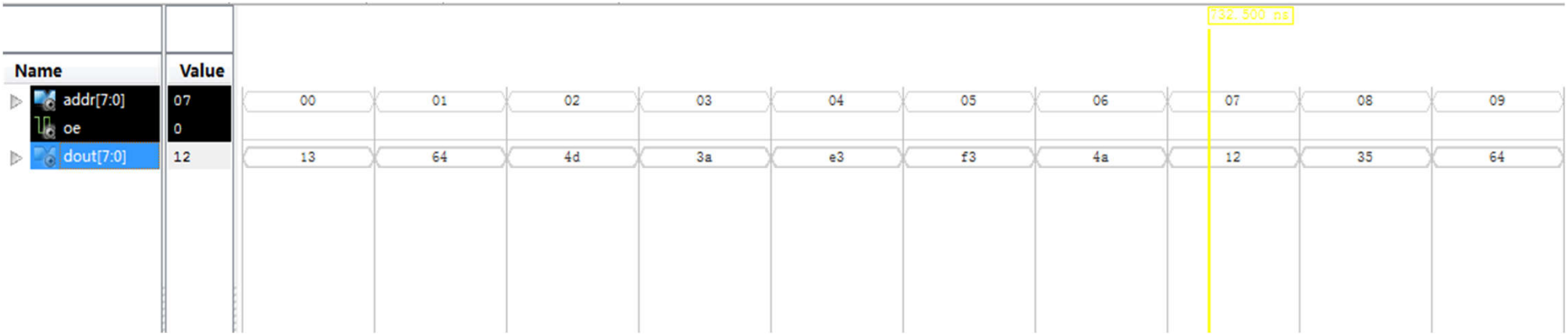
```
architecture Behavior of MyRom is
  type matrix is array (integer range<>) of std_logic_vector(WORDLENGTH-1 downto 0);
  signal rom:matrix (0 to 2**ADDRLENGTH-1);

  procedure load_rom(signal data_word:out matrix) is

    file romfile : text open read_mode is "romfile.dat";
    variable lbuf:line;
    variable i :integer := 0;--循环变量
    variable fdata : std_logic_vector(7 downto 0);
  begin
    --读数据直至文件末尾
    while not endfile(romfile) loop
      readline(romfile,lbuf);--逐行读数据
      read(lbuf,fdata);--将行数据保存到变量fdata
      data_word(i) <= fdata;--将fdata保存到内存信号量中
      i:=i+1;
    end loop;
  end procedure;

begin
  load_rom(rom);
  dout <= rom(conv_integer(addr)) when oe = '0' else
    (others => 'Z');
end Behavior;
```

5.2.5 通用ROM验证

romfile.dat - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
00010011 --13
01100100 --64
01001101 --4d
00111010 --3a
11100011 --e3
11110011 --f3
01001010 --4a
00010010 --12
00110101 --35
```



5.3 双端口RAM

- 双端口RAM是在1个SRAM存储器上具有两套完全独立的数据线、地址线和读写控制线，并允许两个独立的系统同时对其进行随机性访问的存储器（共享式多端口存储器）。
- 双口RAM最大的特点是存储数据共享，并且必须具有访问仲裁控制。

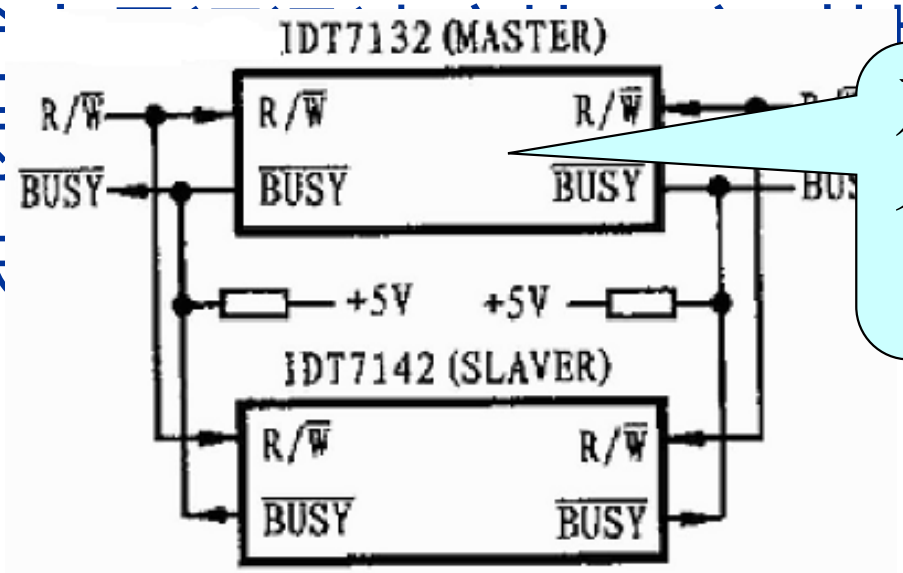
5.3 双端口RAM

- 通用集成电路组成的双端口

- 若两个CPU在同一时间段访问RAM发生竞争，则由仲裁电路迫使后访问的CPU处于等待状态。
- 特点：成本低、简单且存储容量大。
- 缺点：发生竞争，一个CPU必须等待，降低了访问效率。

5.3 双端口RAM

- 专用双端口RAM芯片，如IDT7132/7142、DS1609、CY7C08D53、CY7C024等。芯片有两套完全独立的数据线、地址线和读写控制线，可使两个 CPU分时独立访问其内部RAM资源。
- 优
两
- 缺



双端口RAM 内有一个总线抢占优先级比较器

5.3.1 两种方案应用场合

- 在要求存储量较大时，一般采用通用集成电路组成的双端口RAM；
- 在通信实时性要求较高的而通信数据量不大时，一般采用专用双端口RAM芯片。

5.3.2 双端口RAM设计

- 端口定义

```
entity blk_mem is
   generic(
      data_width : integer := 8;       -- used to change the memory data's width
      addr_width : integer := 8);    -- used to change the memery address' width
   port (--端口 A 信号
      clka   : in   std_logic;
      dina   : in   std_logic_vector(data_width - 1 downto 0);
      addra : in   std_logic_vector(addr_width - 1 downto 0);
      ena    : in   std_logic;
      wea    : in   std_logic;
      douta : out std_logic_vector(data_width - 1 downto 0);
      --端口 B 信号
      clkb   : in   std_logic;
      dinb   : in   std_logic_vector(data_width - 1 downto 0);
      addrb : in   std_logic_vector(addr_width - 1 downto 0);
      enb    : in   std_logic;
      web    : in   std_logic;
      doutb : out std_logic_vector(data_width - 1 downto 0));
end blk_mem;
```

5.3.2 双端口RAM设计

- 结构体设计
- RAM数据类型定义

```
type ram_template is array(2 ** addr_width - 1 downto 0) of std_logic_vector(data_width - 1 downto 0);
shared variable ram1 : ram_template;
```

- 端口A对RAM操作
- 端口B对RAM操作

```
process (clkb)
begin   -- process
  if clkb'event and clkb = '1' then     -- rising clock edge
    if enb = '1' then
      doutb <= ram1(conv_integer(addrb));
      if web = '1' then
        ram1(conv_integer(addrb)) := dinb;
      end if;
    else
      doutb <= (others => '0');
    end if;
  end if;
end process;
```

5.4 先进先出（FIFO）设计

- 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。

5.4 先进先出（FIFO）设计

- 先进先出（ First In First Out，FIFO）与普通存储器的区别是**没有外部读写地址线**，其数据地址由内部读写指针自动加减1完成。
- FIFO通常利用双口RAM和读写地址产生模块来实现其功能。

5.4 先进先出（FIFO）设计

- 先进先出（ First In First Out，FIFO）与普通存储器的区别是**没有外部读写地址线**，其数据地址由内部读写指针自动加减1完成。
- FIFO通常利用双口RAM和读写地址产生模块来实现其功能。
- 满标志
- 空标志

5.4.1 FIFO类型

- 同步控制的FIFO

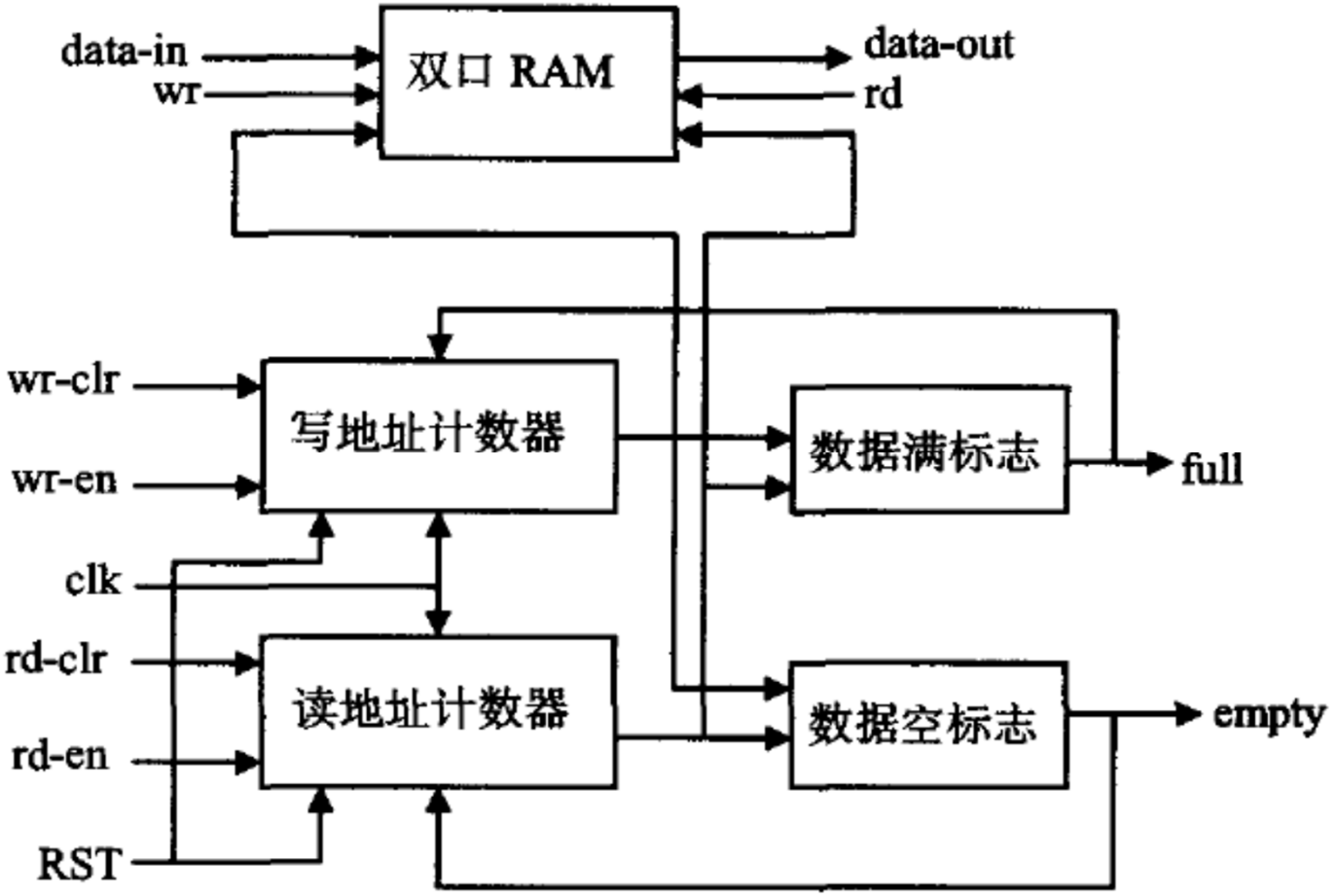
  FIFO的读写时钟相同。
- 异步控制的FIFO

➢ 用于跨时钟域的数据交换；

➢ FIFO的读写时钟不同；

➢ 读写时钟之间不一定存在相位、周期方面的约束关系。

同步FIFO设计


data-in
wr
双口 RAM
data-out
rd
wr-clr
wr-en
写地址计数器
数据满标志
full
clk
rd-clr
rd-en
读地址计数器
数据空标志
empty
RST

5.4.2 同步FIFO设计

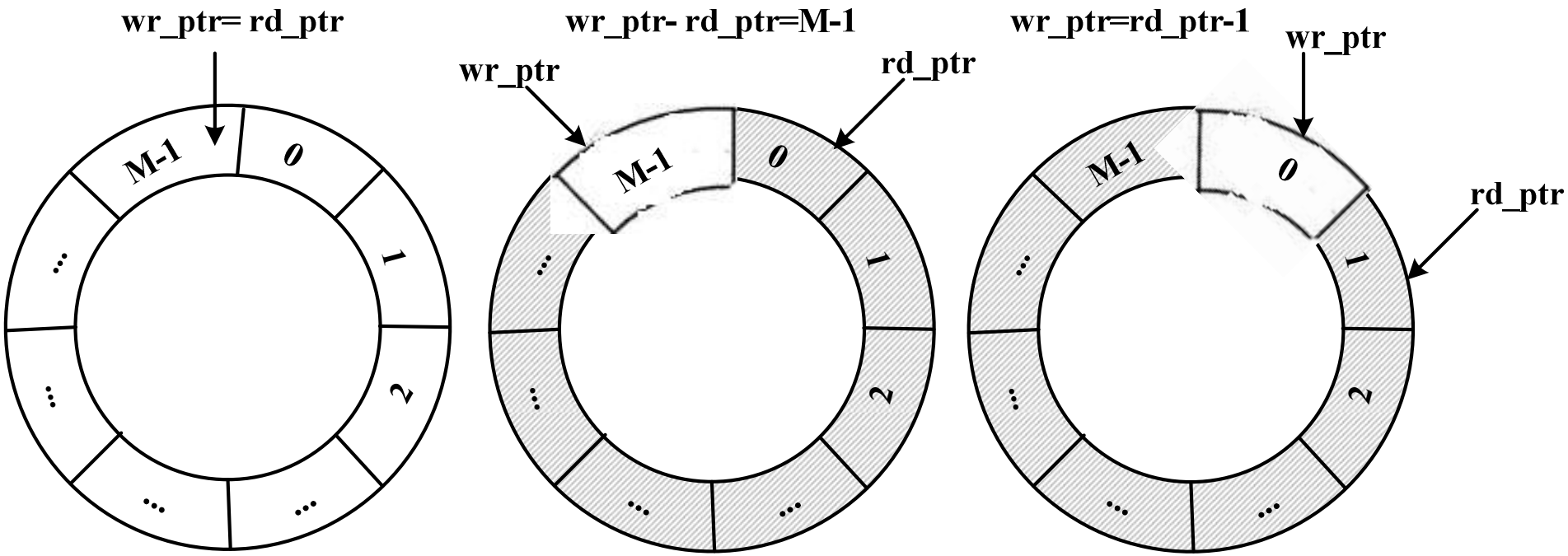
- FIFO为空，不可从FIFO读数据，但可写；
- FIFO为满，不可向FIFO写数据，但可读；
- 非空非满时，FIFO可读、可写。
- FIFO的读写受同一时钟控制；
- FIFO的大小为N。

5.4.2 同步FIFO设计

问题：

1. 如何判断FIFO为空、满？
2. FIFO的读写操作的位置如何判定？

5.4.2 同步FIFO设计



FIFO为空状态

FIFO为满状态

5.4.2 同步FIFO设计

- 当wr_ptr=rd_ptr时，FIFO数据为空；
- 当wr_ptr-rd_ptr=M-l或rd_ptr-wr_ptr=l时，FIFO数据为满；
- 当wr_ptr>=rd_ptr时，wr_ptr-rd_ptr为FIFO内数据个数；
- 当wr_ptr<=rd_ptr时，M-(rd_ptr-wr_ptr)为FIFO内数据个数。

5.4.3 同步FIFO的VHDL实现

- (1)双端口RAM
- 端口定义

```
entity dualram is
  generic
  (
    width : positive  := 8;
    depth : positive  := 8
  );
  port
  (
    ------------------------ port a is only for writing --
    clka  : in  std_logic;
    wr    : in  std_logic;
    addra : in  std_logic_vector(depth - 1 downto 0);
    datain  : in  std_logic_vector(width - 1 downto 0);
    ------------------------------------------------------
    ------------------------ port b is only for reading --
    clkb  : in  std_logic;
    rd    : in  std_logic;
    addrb : in  std_logic_vector(depth - 1 downto 0);
    dataout : out std_logic_vector(width - 1 downto 0)
    ------------------------------------------------------
  );
end entity dualram;
```

5.4.3 同步FIFO的VHDL实现

- (1)双端口RAM
- 结构体实现

```
architecture Behavioral of dualram is

type ram is array(2 ** depth - 1 downto 0) of std_logic_vector(width - 1 downto 0);
signal  dualram   : ram;

begin

  process(clka, clkb)
  begin
    if clka'event and clka = '1' then
      if wr = '0' then
        dualram(conv_integer(addra)) <= datain;
      end if;
    end if;
  end process;

  process(clkb)
  begin
    if clkb'event and clkb = '1' then
      if rd = '0' then
        dataout <= dualram(conv_integer(addrb));
      end if;
    end if;
  end process;

end Behavioral;
```

5.4.3 同步FIFO的VHDL实现

- (2)写地址计数器

```
entity write_pointer is
  generic
  (
    depth : positive
  );
  port
  (
    clk   : in  std_logic;
    rst   : in  std_logic;
    wq    : in  std_logic;
    wr_pt : out std_logic_vector( depth - 1 downto 0)
  );
end entity write_pointer;

architecture RTL of write_pointer is
  signal wr_pt_t : std_logic_vector(depth - 1 downto 0);  -- writer pointer counter

begin
  process(rst, clk)
  begin
    if rst = '0' then
      wr_pt_t <= (others => '0');
    elsif clk'event and clk = '1' then
      if wq = '0' then
        wr_pt_t <= wr_pt_t + 1;
      end if;
    end if;
  end process;
  wr_pt <= wr_pt_t;
end RTL;
```

5.4.3 同步FIFO的VHDL实现

- (3)读地址计数器

```
entity read_pointer is
 generic
 (
  depth : positive
 );
 port
 (
  clk   : in std_logic;
  rst   : in std_logic;
  rq    : in std_logic;
  empty : in std_logic;
  rd_pt : out std_logic_vector(depth - 1 downto 0)
 );
end entity read_pointer;
architecture RTL of read_pointer is
signal rd_pt_t : std_logic_vector(depth - 1 downto 0);  -- read pointer counter
begin
 process(rst, clk)
 begin
  if rst = '0' then
   rd_pt_t <= (others => '0');
  elsif clk'event and clk = '1' then
   if rq = '0' and empty = '0' then
    rd_pt_t <= rd_pt_t + 1;
   end if;
  end if;
 end process;
 rd_pt <= rd_pt_t;
end RTL;
```

5.4.3 同步FIFO的VHDL实现

- (4)空满状态产生器
- 端口定义

```
entity judge_status is
  generic
  (
    depth : positive
  );
  port
  (
    clk   : in  std_logic;
    rst   : in  std_logic;
    wr_pt : in  std_logic_vector(depth - 1 downto 0);
    rd_pt : in  std_logic_vector(depth - 1 downto 0);
    empty : out std_logic;
    full  : out std_logic
  );
end entity judge_status;
```

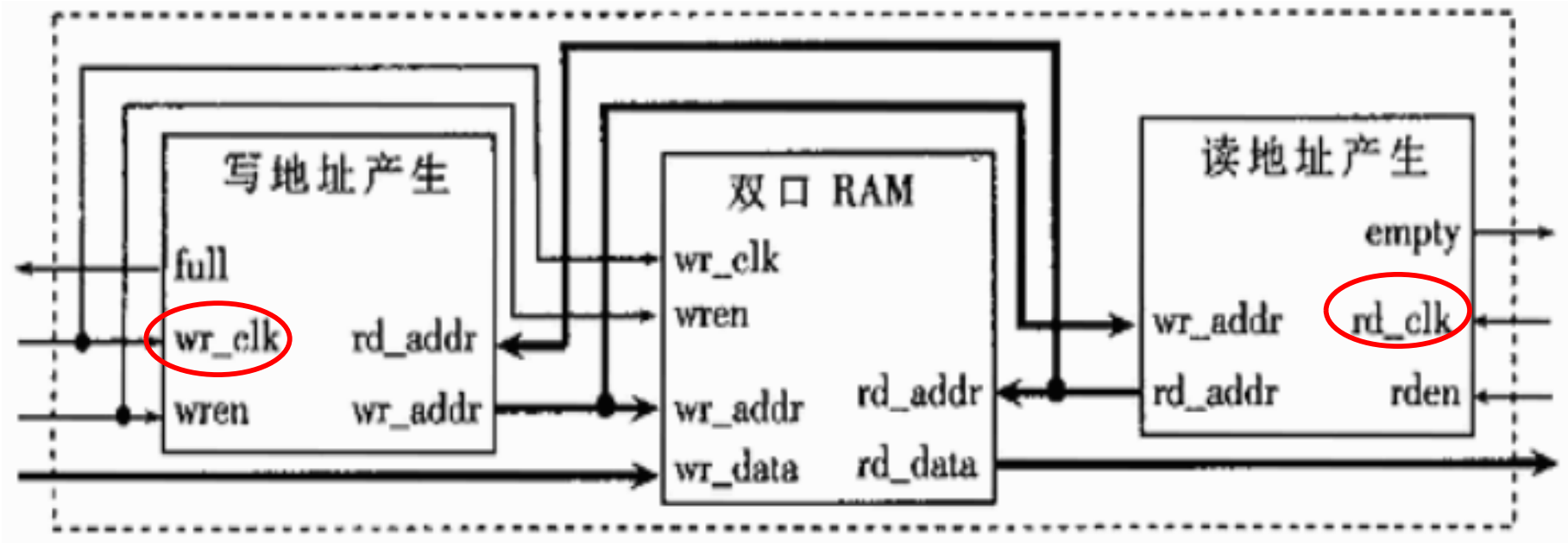
5.4.3同步FIFO的VHDL实现

- (4)空满状态产生器
- 结构体实现

```
architecture RTL of judge_status is
begin
  process(rst, clk)
  begin
    if rst = '0' then
      empty <= '1';
    elsif clk'event and clk = '1' then
      if wr_pt = rd_pt then
        empty <= '1';
      else
        empty <= '0';
      end if;
    end if;
  end process;

  process(rst, clk)
  begin
    if rst = '0' then
      full <= '0';
    elsif clk 'event and clk = '1' then
      if wr_pt > rd_pt then
        if (rd_pt + depth) = wr_pt then
          full <= '1';
        else
          full <= '0';
        end if;
      else
        if (wr_pt + 1 ) = rd_pt then
          full <= '1';
        else
          full <= '0';
        end if;
      end if;
    end if;
  end process;

end RTL;
```

5.4.4 异步FIFO设计

- 读写时钟信号不相同
- 通过地址编码方式解决读写地址变化不同步而引起的空满标志错误的问题。



5.4.6 存储器设计总结

- **存储单元数据结构**

➢ **整数数组**

```
TYPE memory IS ARRAY (INTEGER RANEG <>) OF
    INTEGER;
```

➢ **位矢量**

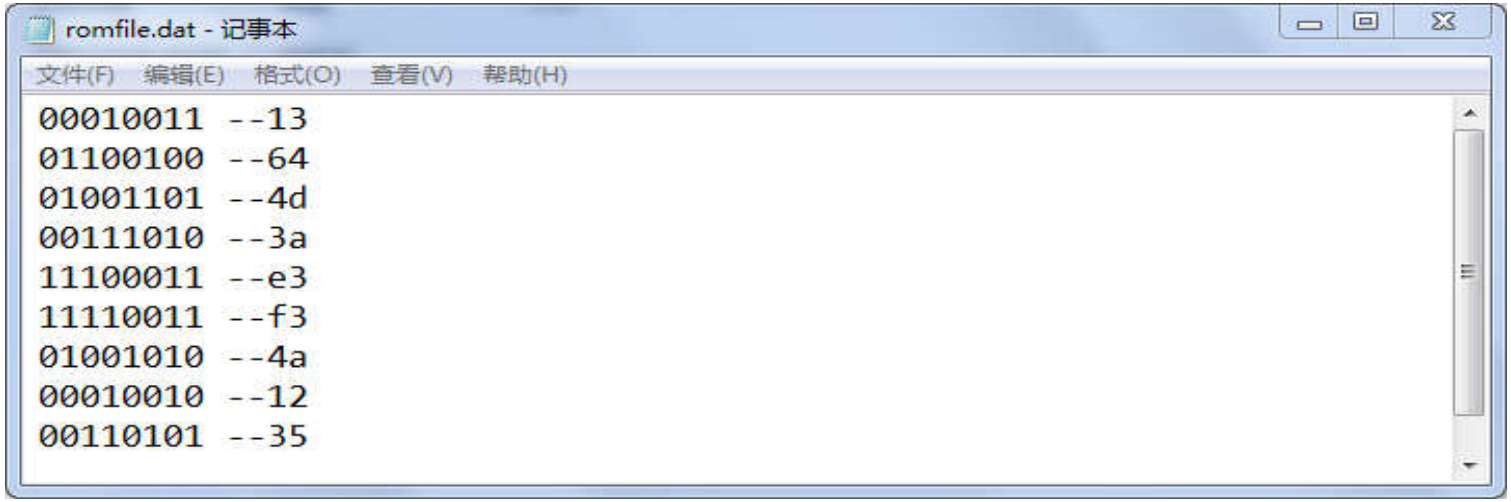
```
SUBTYPE word IS STD_LOGIC_VECTOR(k-1 DOWNTO
    0);
TYPE memory IS ARRAY(0 TO 2**w-1) OF word;
```

5.4.6 存储器设计总结

- **存储单元初始化（外部文件读取）**
  - 自定义数据格式文件
  - .COE文件

(1) 自定义数据格式文件

- VHDL文本输入输出包集合（TEXTIO）
- VHDL语言对文件格式不作任何限制。
- TEXTIO按行进行处理，一行为一个字符串，以回车、换行符作为行结束符。



(2).COE文件

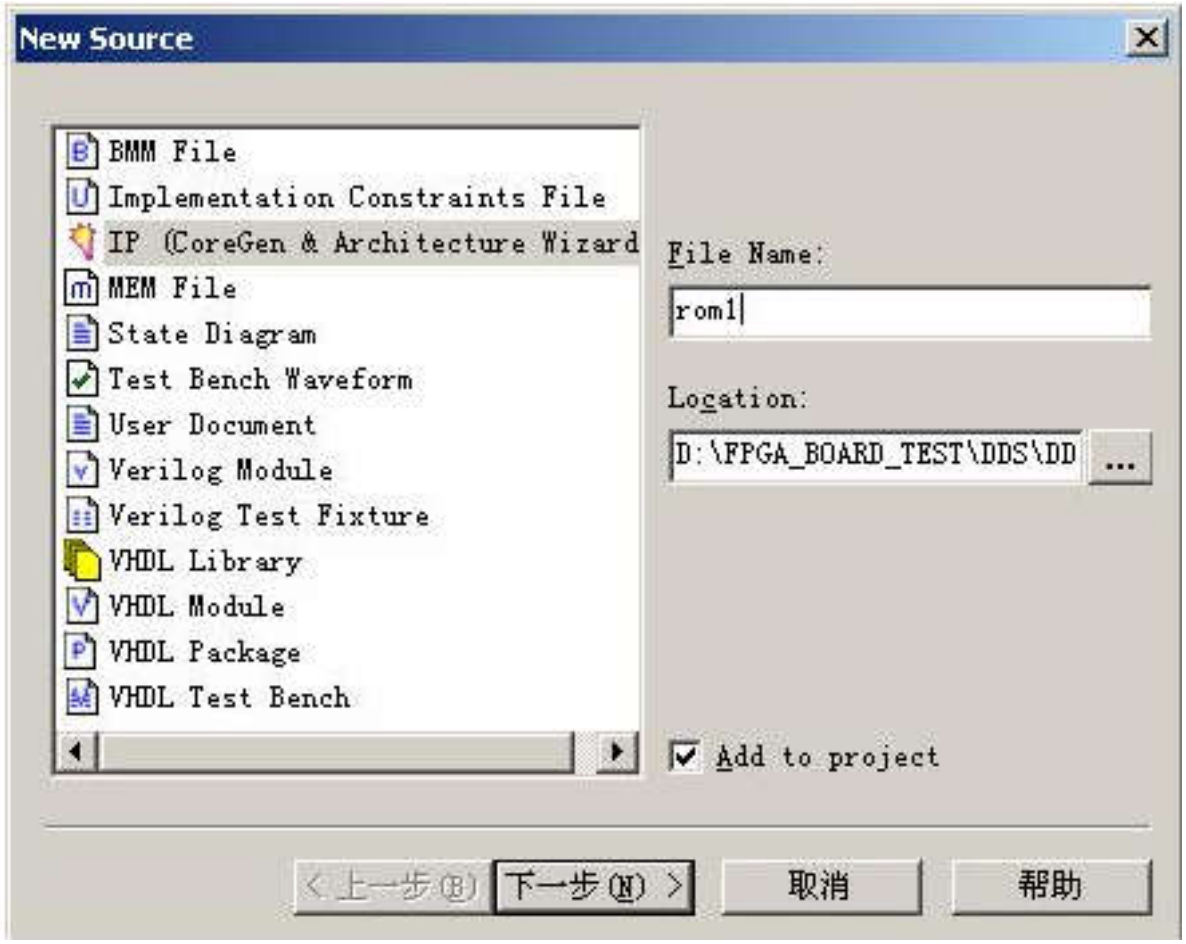
- MEMORY_INITIALIZATION_RADIX=2;
  MEMORY_INITIALIZATION_VECTOR=
  01110100,
  00100000,
  11110101,
  10000000,
  01111000,
  00100010,
  00000001,
  00010100,

数据格式，当前设定为2进制，还可以为8，10，16进制

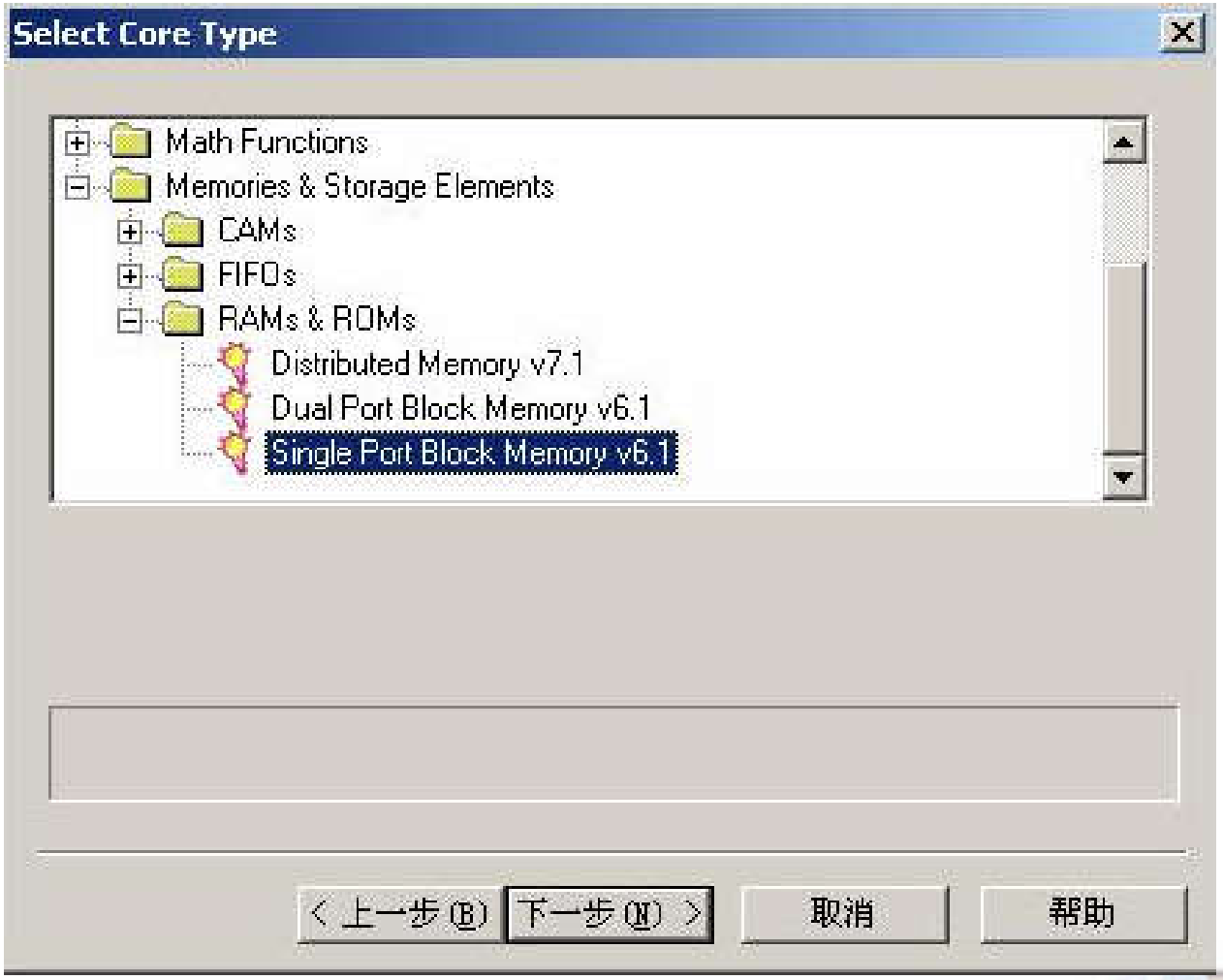
(3)Xilinx FPGA内部IP核设计

- FPGA具有内嵌的BLOCK RAM（BRAM）来扩展其应用范围和系统集成能力（SOC）。
- BRAM可用于配置为单端口RAM、双端口RAM、内容地址存储器（CAM）以及FIFO等常用存储结构。
- BRAM内部每个单位即单片块RAM大小为18Kbit（即位宽为18bit深度为1024，Spartan-3E FPGA）。

(4)FPGA内部IP核设计



(4)FPGA内部IP核设计



5.5 CRC校验电路设计

- CRC原理分析
- CRC电路设计

数据通信差错检测

可靠性

快速性

在数字通信系统中可靠与快速往往是矛盾的。
如何合理地解决可靠与速度这一对矛盾呢?

数据检测技术

- **奇偶校验**
- **和校验**
- **循环冗余码校验CRC**

奇偶校验只需要1

和校验把消息当成若干个8位（或16、32位）的整数序列，相加得到校验码。

5.5.1 CRC原理

将待发送的位串看成系数为 0 或 1 的多项式;

例1： C = 1100101

$$C(x) = 1x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 0x + 1$$
$$= x^6 + x^5 + x^2 + 1$$

◆收发双方约定一个生成多项式 G(x)（其最高阶和最低阶系数必须为1）。

◆发送方用位串及 G(x)进行某种运算得到校验和，并在帧的末尾加上校验和，使带校验和的帧的多项式能被 G(x) 整除。

◆ 接收方收到后，用 G(x) 除多项式，若有余数，则传输有错。

(1)CRC校验和计算方法

◆1.若生成多项式G(x)为r阶(即r + 1位位串)，原帧为m位， 其多项式为M(x)，则在原帧后面添加r个0，即循环左移r位，帧成为m+r位，相应多项式成为$x^rM(x)$;

◆2.按模2除法用G(x)对应的位串去除对应于$x^rM(x)$的位串, 得余数R(x);

◆3. $x^rM(x)$的位串加上余数R(x),结果即传送的带校验和的帧多项式T(x)。

$$T(x) = x^rM(x) + R(x)$$

(2)CRC的特点


CRC的主要特点
检错能力极强
开销很小
易于实现
应用范围广
1. ARJ,LHA,ZIP等压缩软件采用的是CRC-32；
2. GIF,TIFF等图像存储格式；
3. 所有链路层或网络接口层协议中，例如HDLC、DDCMP等众多领域。

(3)CRC 校验简单证明

发送方

设 $x^r M(x)$ 除以 $G(x)$ 的商和余数分别为 $Q(x)$ 和 $R(x)$。则有:

$$x^r M(x) = G(x)\,Q(x) + R(x)$$

即:

$$\frac{x^r M(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

接收方

接收方收到带CRC校验和的帧多项式 $T(x) = x^r M(x) + R(x)$。

$$\frac{T(x)}{G(x)} = \frac{x^r M(x) + R(x)}{G(x)} = \frac{x^r M(x)}{G(x)} + \frac{R(x)}{G(x)}$$

$$= Q(x) + \frac{R(x)}{G(x)} + \frac{R(x)}{G(x)} = Q(x)$$

由于模2加减相当于异或运算,于是接收方模2除后商Q(x),余数0.**得证!**

(4)CRC校验码生成

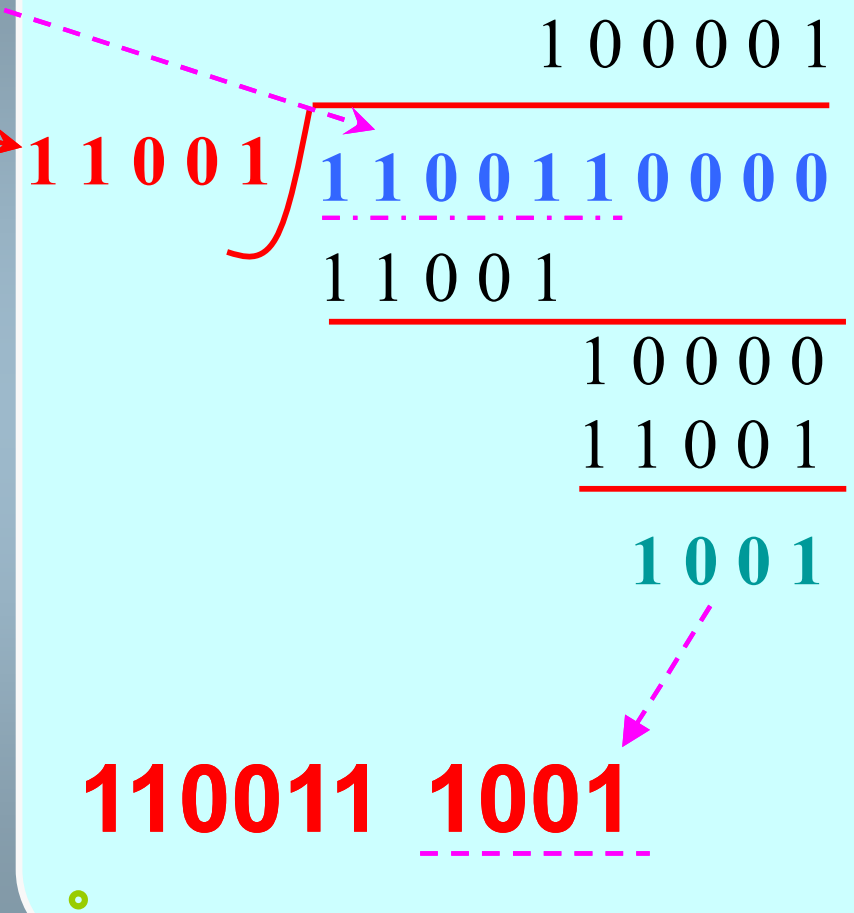
（1）发送数据110011；

（2）生成多项式G(x)= $x^4 + x^3 + 1$；

（3）将要发送的数据系列左移4位，新的序列为1100110000；

（4）按模2算法，将生成的新序列除以生成多项式序列；

（5）将余数多项式比特序列加到新的序列中即得发送端传送序列。



(5)生成多项式选择

- 生成多项式应包含X^0，即常数项1。
- R阶的系数为1，即必须有X^r。
- 发送码T(X)，发送过程产生错误。
- 接收端数据T(X)+E(X)
- 能够检测出的差错情况：
  - 1位差错
  - 奇数位差错
  - 突发性差错
  - 2位差错

(6)CRC检错性能

- 能检测出全部单个错误
- 能检测出全部随机二位错误
- 能检测出全部奇数个错误
- 能检测出全部长度小于k位的突发错误
- 能以[1-（1/2）$^{k-1}$]概率检测出长度为（k+1）位的突发性错误

(7)生成多项式 G(x) 的国际标准

- CRC-8：x^8+x^2+x+1
- CRC-10：$x^{10}+x^9+x^5+x^4+x^2+1$
- CRC-12：$x^{12}+x^{11}+x^3+x^2+x+1$
- CRC-16：$x^{16}+x^{15}+x^2+1$
- CRC-CCITT：$x^{16}+x^{12}+x^5+1$
- CRC-32：$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

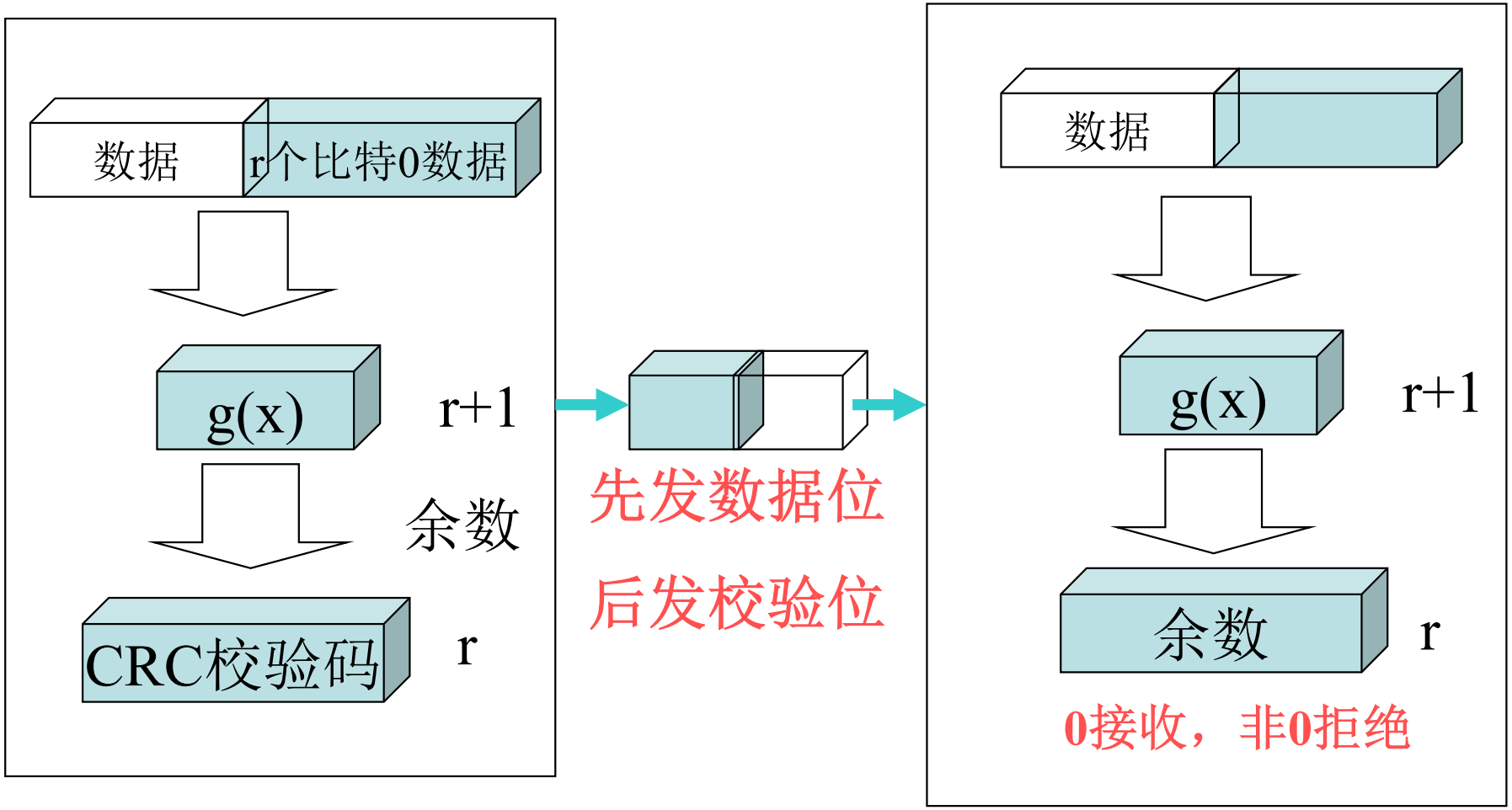
(8)接收方校验方案

接收方校验方案

方案一：直接用接收到的序列除以生成多项式G(x),如果余数R’(x) = 0,则证明传输正确。

方案二：提取接收到序列的信息码元，重复发送方的操作$x^rM(x)$，再除以生成多项式G(x),如果余数R’(x) = R(x),则证明传输正确。

(9)CRC校验码的生成器和校验器



5.5.2 CRC校验生成程序

- 端口定义

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity crcm is
        port(--clk,hrecv,datald:in std_logic;   --datald  (sdata装载信号，为'1'时锁定，为0时装入信号)
                clk ,datald:in std_logic; --hrecv 握手信号(--->hrecv检测模块接受) clk  时钟触发信号
                sdata :in std_logic_vector(11 downto 0); --待发送的12位数据
                --crc校验生成模块==>data(12位数据)+crc(5位校验码)
                datacrco :out std_logic_vector(16 downto 0);--crc校验校验模块,接收生成模块16位data+crc
                hsend:out std_logic); ---hsend  握手信号(crc校验生成模块发送使用----->hrecv)
end crcm;
architecture comm of crcm is
        constant multi_coef: std_logic_vector(5 downto 0):="110101";--多项式系数，MSB一定为'1
        signal cnt,rcnt:std_logic_vector(4 downto 0);              --用于计数
        signal dtemp,sdatam,rdtemp:std_logic_vector(11 downto 0);
        signal rdatacrc:std_logic_vector(16 downto 0);
        signal st,rt:std_logic;
```

5.5.2 CRC校验生成程序

- 结构体实现

```
begin
process(clk)
        variable crcvar: std_logic_vector(5 downto 0);
begin
        if(clk'event and clk='1') then        ---时钟触发信号
          if(st='0' and datald='1') then
             dtemp<=sdata;                    sdatam<=sdata;   cnt<=(ot[illegible]);hsend<='0';st<='1';
           elsif(st='1' and cnt<7) then cnt<=cnt+1;
              if(dtemp(11)='1') then crcvar:=dtemp(11 downto 6) xor multi_coef;
                dtemp<=crcvar(4 downto 0)& dtemp(5 downto 0) & '0';
             else dtemp<=dtemp(10 downto 0)&'0' ;
              end if;
           elsif(st='1' and cnt=7) then
              datacrco<=sdatam& dtemp(11 downto 7);
              hsend<='1';st<='0';
           end if;
        end if;
end process;
end comm;
```

数据位串除以生成多项式

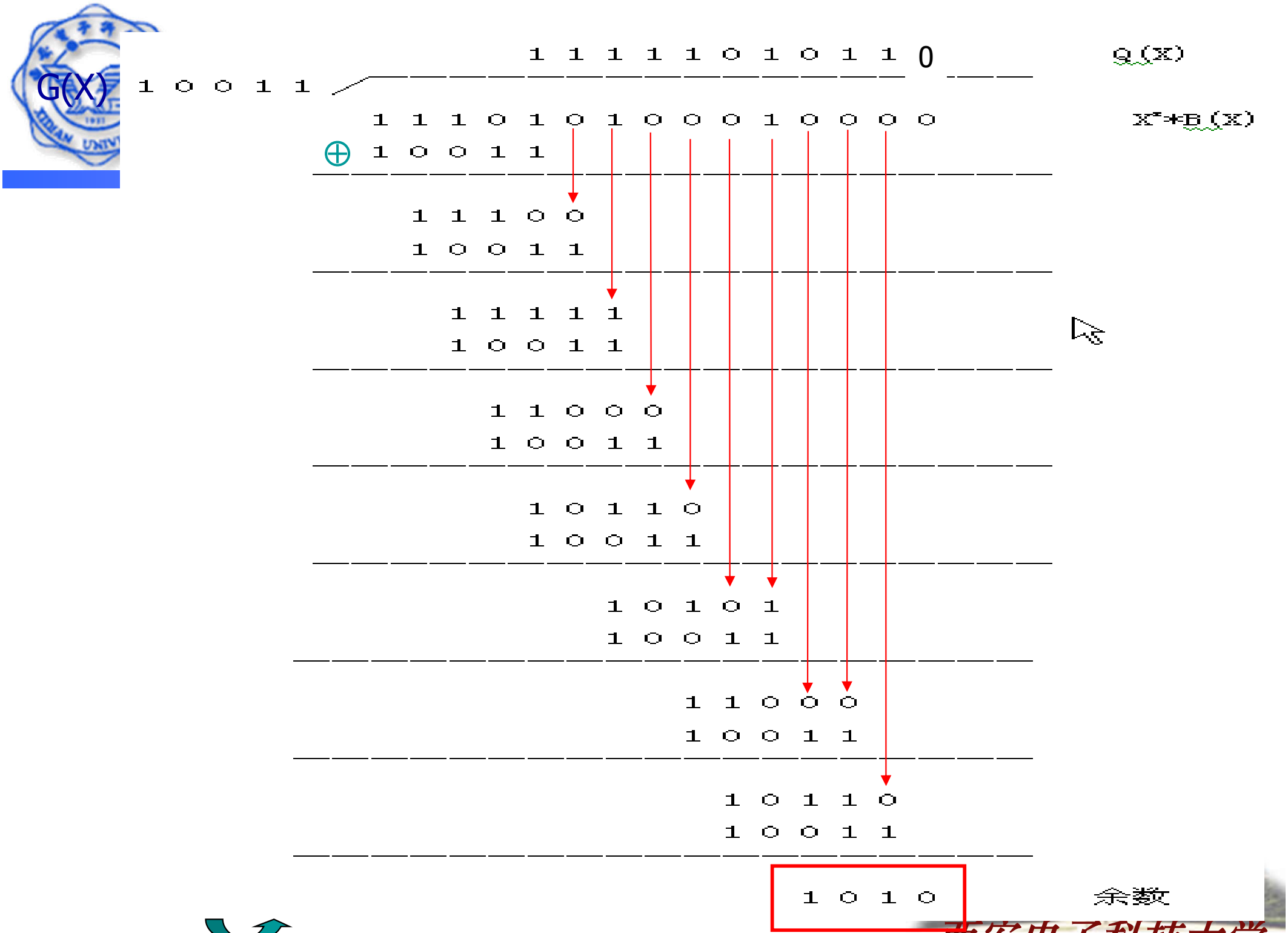
发送数据=数据位+余数

5.5.3 CRC校验检测程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity crcn is
        port(clk,hrecv:in std_logic;  --hrecv 握手信号(  ->hrecv检测模块接受),clk 时钟触发信号
                datacrcr :in std_logic_vector(16 downto 0);--crc校验生成模块==>data(12位数据)+crc(5位校验码)
                rdata :out std_logic_vector(11 downto 0);--提取12位有效数据
                datafini:out std_logic;--flag数据接受校验完成状态显示位-->LED
                ERROR0:out std_logic);---ERROR0 误码警告信号-->beep;--hsend 握手信号(crc校验生成模块发送使用----->hrecv)
end crcn;
architecture comm of crcn is
        constant multi_coef: std_logic_vector(5 downto 0):="110101";--多项式系数，MSB一定为'1'
        signal cnt,rcnt:std_logic_vector(4 downto 0);          --用于计数
        signal dtemp,sdatam,rdtemp:std_logic_vector(11 downto 0);
        signal rdatacrc:std_logic_vector(16 downto 0);
        signal st,rt:std_logic ;  --st,rt状态转移标志位
begin
process(hrecv,clk)
        variable rcrcvar: std_logic_vector(5 downto 0);
```

5.5.3 CRC校验检测程序

```
begin
process(hrecv,clk)
        variable rcrcvar: std_logic_vector(5 downto 0);
begin
        if(clk'event and clk='1') then
                if(rt='0' and hrecv='1') then
                        rdtemp<=datacrci(16 downto 5);
                        rdatacrc<=datacrci;
                        rcnt<=(others=>'0');
                        ERROR0<='0';
                        rt<='1';
                elsif(rt='1' and rcnt<7) then
                        datafini<='0';
                        rcnt<=rcnt+1;
                        rcrcvar:= rdtemp(11 downto 6)xor multi_coef;
                        if(rdtemp(11)='1') then
                        rdtemp<=rcrcvar(4 downto 0) & rdtemp(5 downto 0) & '0';
                        else rdtemp<=rdtemp(10 downto 0)&'0';
                        end if;
                elsif(rt='1' and rcnt=7) then datafini<='1';
                        rdata<=rdatacrc(16 downto 5);
                        rt<='0';
                        if(rdatacrc(4 downto 0) /=rdtemp(11 downto 7)) then
                                        ERROR0<='1';
                        end if;
                end if;
end if;
end process;
end comm;
```


G(X)
1 0 0 1 1
1 1 1 1 1 0 1 0 1 1 0
Q(X)
1 1 1 0 1 0 1 0 0 0 1 0 0 0 0
X⁴*B(X)
⊕ 1 0 0 1 1
1 1 1 0 0
1 0 0 1 1
1 1 1 1 1
1 0 0 1 1
1 1 0 0 0
1 0 0 1 1
1 0 1 1 0
1 0 0 1 1
1 0 1 0 1
1 0 0 1 1
1 1 0 0 0
1 0 0 1 1
1 0 1 1 0
1 0 0 1 1
1 0 1 0
余数

CRC冗余校验码

- 11101010001 1010
- CRC校验码
- 信息码
- CRC冗余校验码