硬件描述语言VHDL及其应用

哈工大微电子中心

一、目的

1. 了解目前电子设计系统方法及流程
2. 了解/掌握综合与验证工具
3. 能用VHDL设计复杂功能电路

二、内容

1. 高层次设计概述
2. 如何写优化的VHDL代码
3. examples
4. SoC设计方法学
5. 设计工具使用

三、如何学习本课程

带着实际课题学习，多提问题，一起分析、讨论

一、高层次设计概述

1. EDA工具发展
2. 设计方法
3. 深亚微米设计问题
4. 测试综合（可测性设计）
5. Top-down设计流程
6. 硬件描述语言
7. 综合
8. VHDL设计小结

1.1 EDA工具发展

年代	名称	硬件	特征
70's	CAD	16位小型机	图型编辑，设计规则检查
80's	CAE	32位工作站	LVS工具
90's	EDA	32位工作站	逻辑/行为综合工具
Now	SoC ?	32位工作站	物理综合工具，IP复用技术

1.1 EDA工具发展（Cont.）

CAD: 逻辑图输入、逻辑模拟、电路模拟、版图设计和版图验证分别进行，需要对两者结果进行多次比较、修改。设计规模较小

CAE: 集逻辑图输入、逻辑模拟、测试码生成、电路模拟、版图设计、版图验证等工具一体，构成一个较完整的IC设计系统

EDA: HDL取代逻辑输入，逻辑网表由综合工具自动产生，可管理性增强，易于维护和数据交换

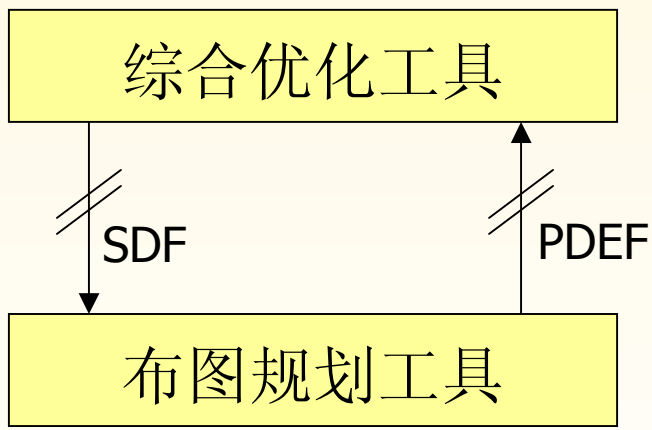
SoC: 采用深亚微米工艺生产技术，基于平台设计和IP复用技术，时序收敛性为首要目标

1.2 设计方法

- 自底向上设计方法（Bottom-up）：系统功能划分➡单元设计➡功能模块设计➡子系统设计➡系统总成
- 自顶向下设计方法（Top-down）：系统行为设计➡结构设计➡逻辑设计➡电路设计➡版图设计
- 基于平台设计方法（Platform-based）：SoC设计普遍采用的方法，SoC平台和IP—Intellectual Property
- 其它设计：嵌入式设计方法，层次式设计方法等

1.3 深亚微米设计问题

- 连线延时
- 时序模型
- 器件模型
- 信号完整性
- 电磁干扰
- 功耗
- 设计工具



SDF—标准数据格式

PDEF—物理设计交换格式

1.4 测试综合

目的：

- 集成电路的测试简单化
- 嵌入可测试结构，加速可测性设计
- 产品制造前就可评价设计的可测性
- 消除冗余逻辑
- 诊断不可测的逻辑结构

内容：

测试嵌入、设计规则检查、测试码生成、故障模拟/诊断和输出测试图样

测试综合包括了使测试成功的每一步骤：如加入带测试因素的电路，对逻辑综合增加约束条件以满足测试要求及对高级语言描述的可测结构的综合等都可归结为测试综合

1.4 测试综合（Cont.）

方法：

- Full Scan
- Partial Scan
- BIST
- Boundary Scan

分类：

1—Pass

2—Pass

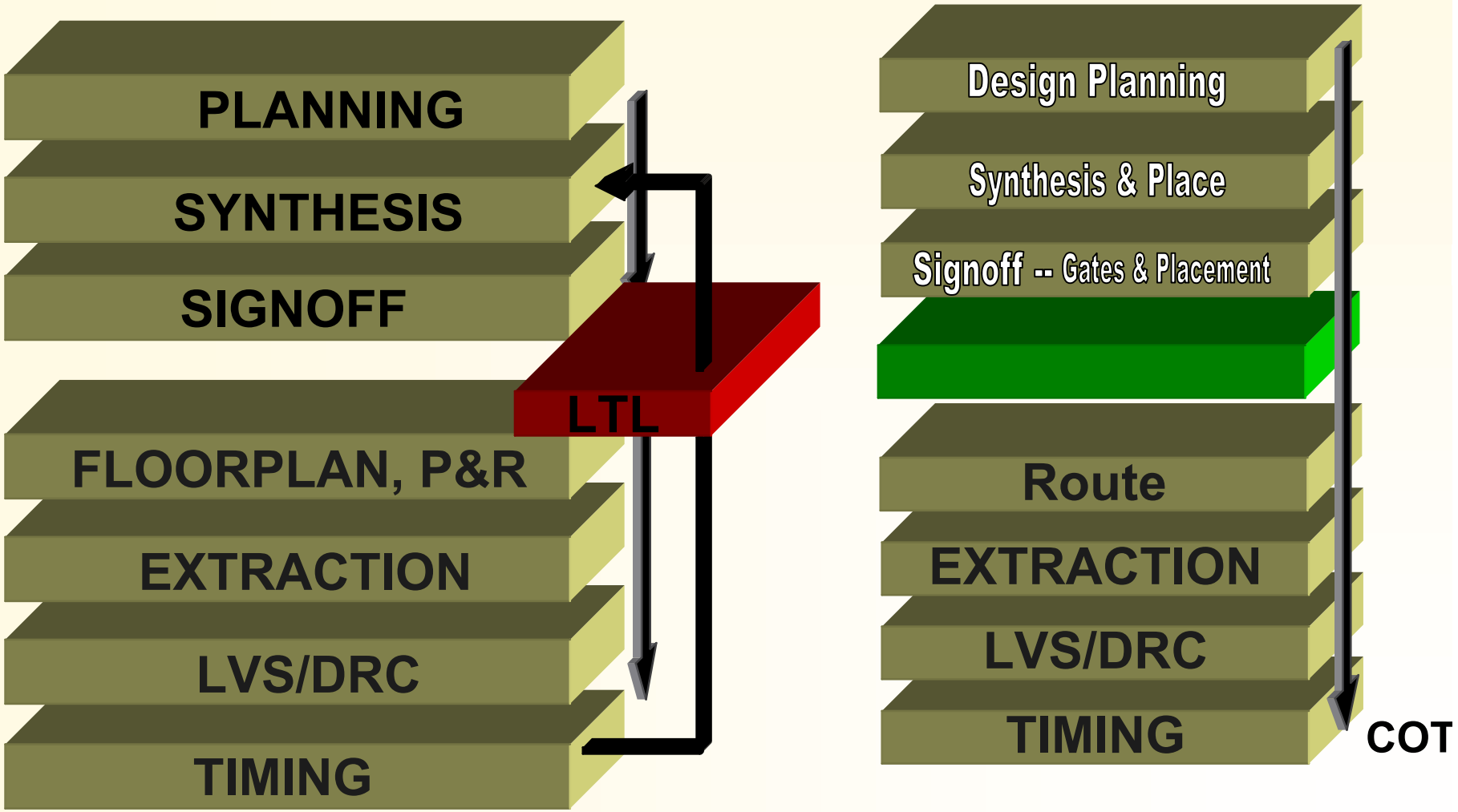
标准/规范：

- IEEE 1149
- IEEE P1500
- VSIA Related Spec.

SoC可测试设计：

- IP可测试设计
- Glue Logic可测试设计
- 测试存取结构

1.5 Top-down设计流程



1.6 硬件描述语言

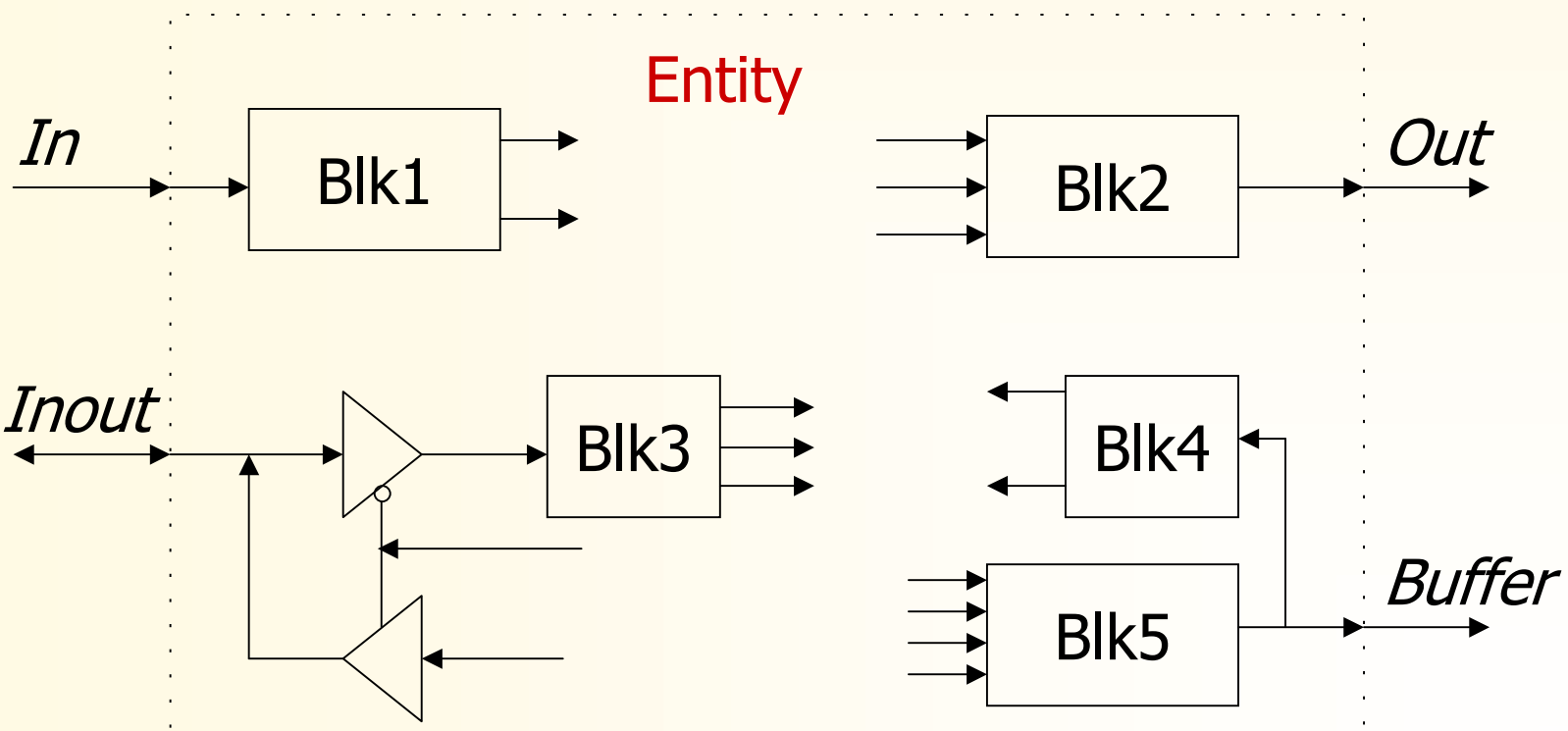
1) VHDL & Verilog

2) VHDL Object

- Entity—I/O界面描述
- Architecture —功能定义
- Process —行为模块
- Library —VHDL Object的集合
- Package —数据类型、子程序、子单元的集合
- Configuration —Architecture/Parameter选择

1.6 硬件描述语言（Cont.）

3) VHDL中的端口：In　Out　Inout　Buffer



1.7 综合

Definition：Synthesis = Translation + Optimization

- HDL code ⟶ gtech logic netlist
- Optimization & technology Mapping ⟶
  min(Speed X Area X Power)

- Behavioral Synthesis:

  Scheduling and Allocation Algorithm

1.8 VHDL设计小结

i. 一个完整的设计由一些子单元相互连接而成

ii. 每个子单元有一个Entity和至少一个Architecture

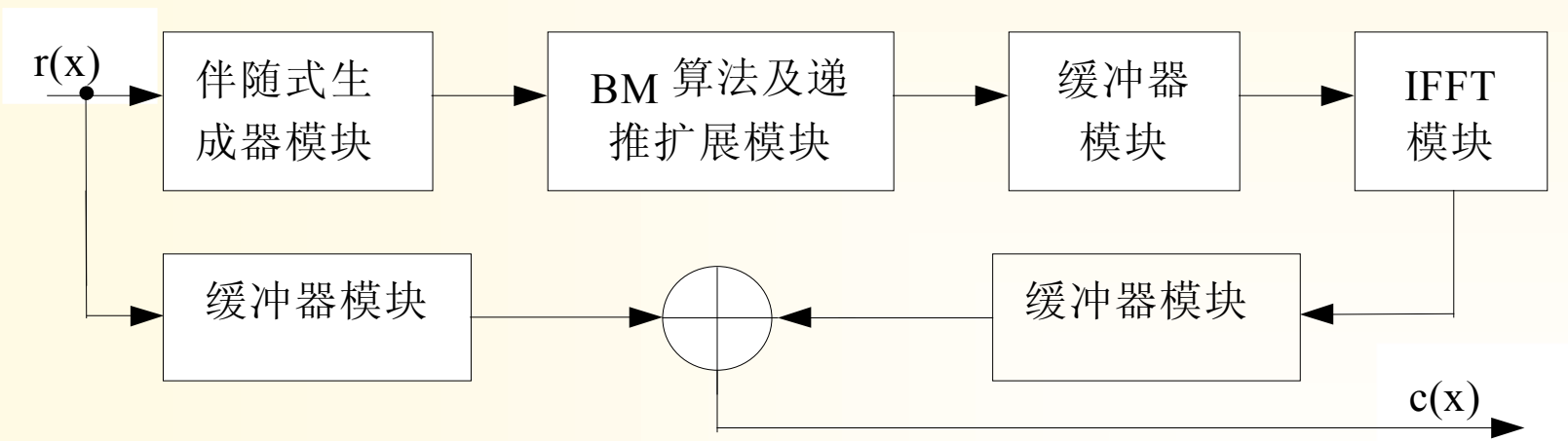
iii. 单元间数据传递是通过在Entity中描述中所声明的端口进行，通信端口的信号类型、端口宽度以及端口方向要匹配

iv. 一个Architecture可包括Behavioral、Dataflow和Structure风格语句

v. 子单元（Component）在使用之前要声明

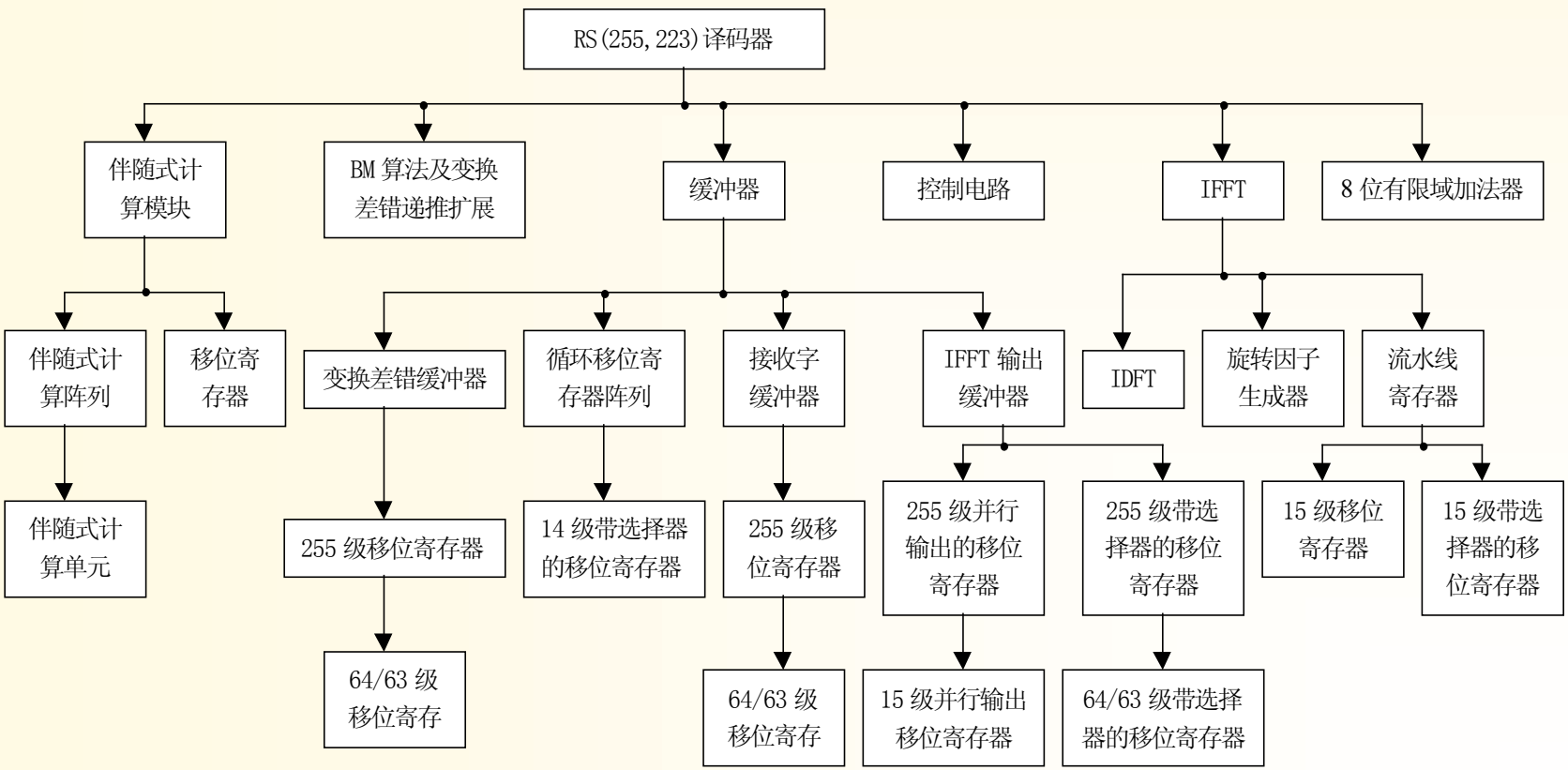
1.8 VHDL设计小结（Cont.）

RS(255, 223)码译码器Top框图



1.8 VHDL设计小结（Cont.）

RS(255, 223)码译码器详细模块图



1.8 VHDL设计小结（Cont.）

```
entity rsdecoder is
    port (reset , clk : in std_logic;
          decin : in bit8;
          decout : out bit8);
end rsdecoder;
```

1

```
architecture structural of rsdecoder is
    component syndrome
        port (reset, clk : in std_logic;
              rec : in bit8;
              synfb : in bit8;
              syndout : out bit8);
    end component;
```

```
component bmexpand
        port(reset, clk : in std_logic;
         synin : in bit8;
         cnt : in rsInt;
         synout : out bit8;
         lstsfe : out bit8;
         bout : out bit8);
end component;
component bmfftbuf
        port(reset, clk : in std_logic;
             ctl255 : in std_logic;
             syno, addo : in bit8;
             bmfo : out rsbit8_vector(0 to N2 - 1));
end component;
```

2

1.8 VHDL设计小结（Cont.）

```
component ifft
        port(reset, clk : in std_logic;
             ctlN1m1, ctlN1 : in std_logic;
             ctl254 : in std_logic;
             buffin : in rsbit8_vector(0 to N2 - 1);
             iffto : out bit8);
end component;
```

3

```
component fftobuf
        port(reset, clk : in std_logic;
             ctlobf : in std_logic;
             din : in bit8;
             fbo : out bit8);
end component;
```

```
component decbuf
        port(reset, clk : in std_logic;
             din : in bit8;
             dout : out bit8);
end component;
```

```
component control
        port(reset, clk : in std_logic;
             ctlN1m1, ctlN1 : out std_logic;
             ctl254, ctl255 : out std_logic;
             ctlobf : out std_logic;
             synfb : out std_logic;
             cntout : buffer rsInt);
end component;
```

4

1.8 VHDL设计小结（Cont.）

```
component xor8
        port(in1, in2 : in bit8;
             xout : out bit8);
end component;
```

5

```
signal ctlN1m1, ctlN1, ctl254, ctl255 : std_logic;
signal ctlobf, synfb : std_logic;
signal fbo, dout, synout, lstsfe, bout : bit8;
signal cntout : rsInt;
signal iffto, syndout : bit8;
signal bmfo : rsbit8_vector(0 to N2 - 1);
```

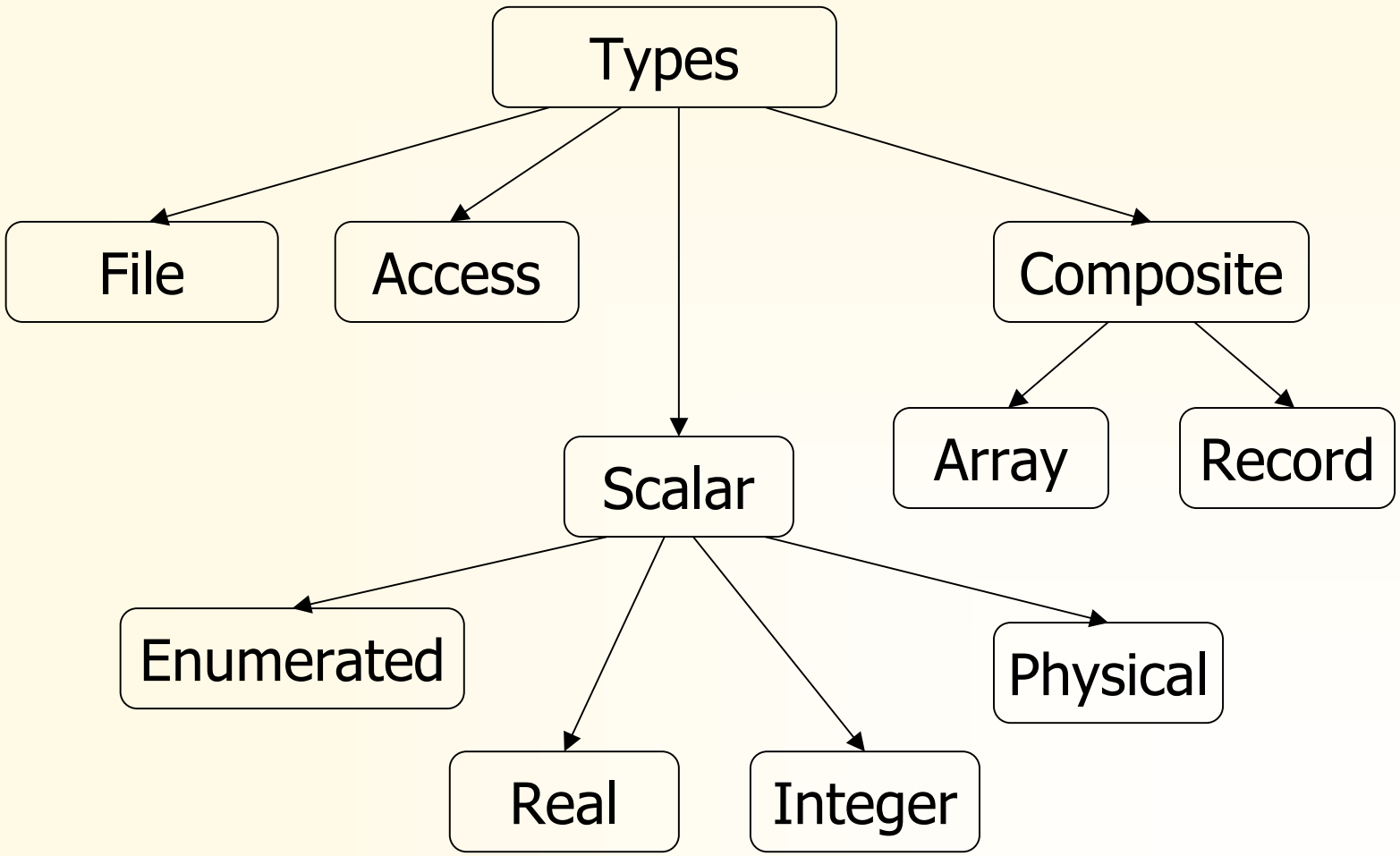
1.8 VHDL设计小结（Cont.）

```
begin
    u1 : control port map(reset, clk, ctlN1m1, ctlN1, ctl254, ctl255, ctlobf,
synfb, cntout);
    u2 : syndrome port map(reset, clk, decin, synfb, syndout);
    u3 : bmexpand port map(reset, clk, syndout, cntout, synout, lstsfe, bout);
    u4 : bmfftbuf port map(reset, clk, ctl255, synout, lstsfe, bmfo);
    u5 : ifft port map(reset, clk, ctlN1m1, ctlN1, ctl254, bmfo, iffto);
    u6 : fftobuf port map(reset, clk, ctlobf, iffto, fbo);
    u7 : decbuf port map(reset, clk, decin, dout);
    u8 : xor8 port map(fbo, dout, decout);
end structural;
```

二、如何写优化的VHDL代码

1. 数据类型
2. 并发/顺序赋值语句
3. 小结
4. Process语句
5. 资源共享
6. 其它

2.1 数据类型


Types
File
Access
Composite
Array
Record
Scalar
Enumerated
Physical
Real
Integer

2.1 数据类型(Cont.)

标准数据类型：

bit, bit_vector, std_ulogic, std_logic,

std_logic_vector, boolean, integer, etc.

复合数据类型：

array, record, sub_type, new type

2.1.1 赋值语句

i/ signal t, s : bit;

s <= '1'; -- s <= t;

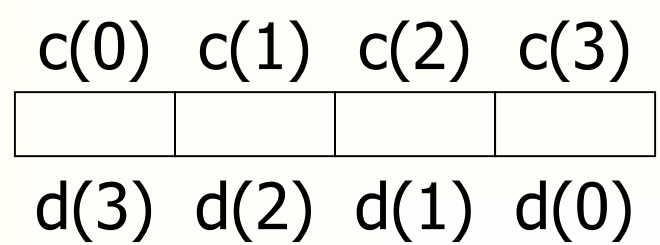
ii/ signal c : bit_vector(0 to 3);

signal d : bit_vector(3 downto 0);

c <= "1011";

d <= c; -- ok ?

c(0 to 3) <= d(0 to 3) – No?



2.1.1 赋值语句（Cont.）

iii/ signal s, t, w, m : bit;

signal c : bit_vector(0 to 3);

c <= “1011”;

c <= s & t & w & m;

c <= (‘1’, ‘0’, ‘1’, ‘1’);

c <= 3; --No

c <= (0 -> ‘1’, 1 -> s,

2-> ‘1’, 3 -> ‘1’); --Ok

Ok?

2.1.1 赋值语句（Cont.）

iv/ signal a_vec : bit_vector(0 to 11);

a_vec <= B"1100_0011_0011_1100";

a_vec <= "110000110011100";

a_vec <= X"C33C";

a_vec <= X"C3_3C";

a_vec <= "1100_0011_0011_1100";

a_vec <= "C33C";

No!

2.1.1 赋值语句（Cont.）

位串中的进制表示：

二进制—B（Binary）

八进制—O（Octal）

十六进制—X（Hexadecimal）

十进制—D（Decimal） No!

2.1.1 赋值语句（Cont.）

v/ signal A, B, C : bit_vector(3 downto 0);

C <= A and B;

≙

C(3) <= A(3) and B(3);

C(2) <= A(2) and B(2);

C(1) <= A(1) and B(1);

C(0) <= A(0) and B(0);

A =	‘0’	‘1’	‘1’	‘0’
B =	‘1’	‘1’	‘0’	‘1’
C =	‘0’	‘1’	‘0’	‘0’

C <= not A

OK!

A =	‘0’	‘1’	‘1’	‘0’
C =	‘1’	‘0’	‘0’	‘1’

2.1.1 赋值语句（Cont.）

```
signal C : bit_vector(0 to 7);
C(4) <= '1';
C(0 to 3) <= "1001";
```

```
vi/ slice of array
entity VHDL is
    port(A : in bit_vector(0 to 7);
         outp : out bit);
end VHDL;
architecture E1 of VHDL is
begin
    outp <= A(5);
end;
```

2.1.1 赋值语句（Cont.）

vii/ Composite data type

```
type date is record
    year : integer range 1980 to 2030;
    month : integer range 1 to 12;
    day : integer range 1 to 30;
end record;
subtype bit8 is bit_vector(7 downto 0);
```

2.1.1 赋值语句（Cont.）

vii/ Composite data type

```
signal weekday, today : date;
weekday.year <= 2003;
weekday.monty <= 2;
weekday.day <= 14;
today <= weekday;
```

2.1.2 数据类型转换

强类型语言：VHDL具有丰富的数据类型，不同类型的对象（信号、变量）不能直接赋值

数据类型转换三种方法：类型标记转换法、函数转换法和常数转换法

经常转换的数据类型：std_logic, bit, std_ulogic, boolean, signed unsigned, std_logic_vector, bit_vector

2.1.2 数据类型转换(Cont.)

类型标记转换法

std_logic and std_ulogic,

std_logic_vector and signed

std_logic_vector and unsigned

integer and real等

```
signal a std_logic_vector(0 to 7);
signal b unsigned(0 to 7);
b <= unsigned(a);
```

2.1.2 数据类型转换(Cont.)

函数转换法

std_logic and bit

std_ulogic and bit, boolean and bit,

std_logic_vector and bit_vector

integer and std_logic_vector/unsigned等

```
signal a std_logic_vector(0 to 7);
signal b integer range 0 to 255;
a <= to_stdlogicvector(X"AF");
b <= conv_ingeter(a);
```

2.1.2 数据类型转换(Cont.)

常数转换法

```
type typeconv_type is array(std_ulogic'low to
                                    std_ulogic'high) of bit;
constant typeconv : typeconv_type := ('0' | 'L'
=> '0', '1' | 'H' => '1', others => '0');
signal s : std_ulogic;
signal a : bit;
a <= typeconv(s);
```

2.1.2 数据类型转换(Cont.)

Discussion

How to transform bit type to boolean type?

```
signal bitty : bit;
signal booly : boolean;
booly <= (bitty = '1');
```

2.1.3 逻辑运算与关系运算

运算符:

and, or, not, xor, nand, nor

=, /=, <, <=, >, >=

Discussion: What is the result of the following relational statement?

"1000" > "1111" = false

2.1.4 算术操作

运算符:

+， −， *， mod， /， rem

右操作数必须为2的指数！！

操作数类型:

std_logic_vector, integer, signed, unsigned

use ieee.std_logic_unsigned.all;

2.1.4 算术操作（Cont.）

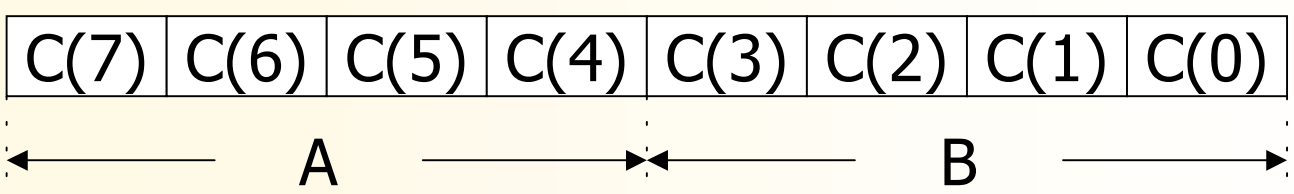
```
signal a, b : std_logic_vector(3 downto 0);
q1 <= unsigned(a) + unsigned(b);   -- 4bit
q2 <= unsigned(a) + signed(b);     -- 5bit
q3 <= signed(a) + signed(b);       -- 4bit
q4 <= a + b;                       -- 4bit
q5 <= ('0' & a) + b;               -- 5bit
```

2.1.5 连字符和聚集

连字符：concatenation operator

聚集： aggregates

```
signal A, B : std_logic_vector(3 downto 0);
signal C : std_logic_vector(7 downto 0);
signal D : std_logic;
C <= A & B;
```



2.1.5 连字符和聚集(Cont.)

```
C(7) <= 'Z';

C(6 downto 3) <= A;

C(2 downto 0) <= '0' & A(1 downto 0);

C <= (7 => '1', 6 => D, 5 downto 2 => '1',
        others => '0');

C <= "00000000";  -----初始化

C <= (others => '0');
```

2.2 并发/顺序赋值语句

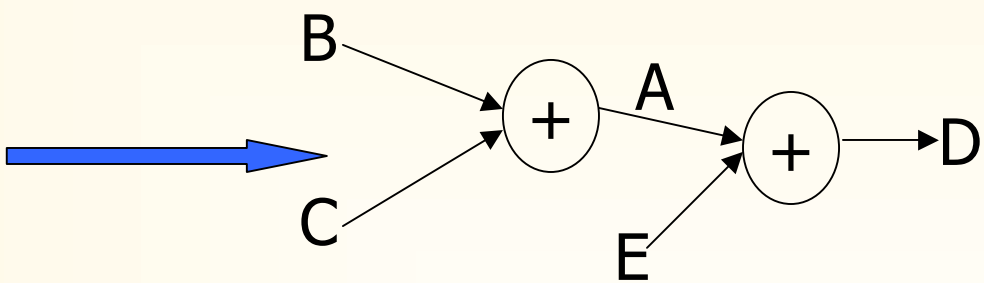
并发赋值语句在architecture的begin和end之间，与书写顺序无关，每一条并发语句均可用一个process语句等价

顺序赋值语句只能在process和子程序的begin和end之间，它除信号赋值语句外，还有变量赋值

2.2.1 并发赋值语句

D <= A + E;

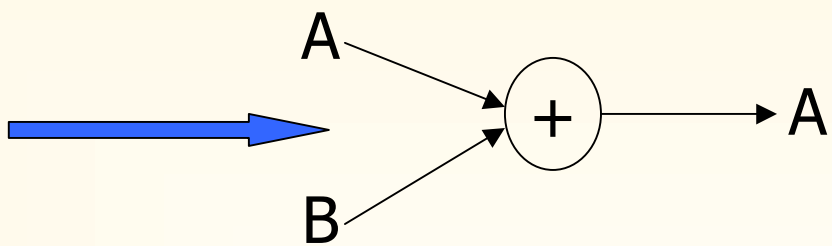
A <= B + C;



2.2.1 并发赋值语句(Cont.)

并发赋值语句：

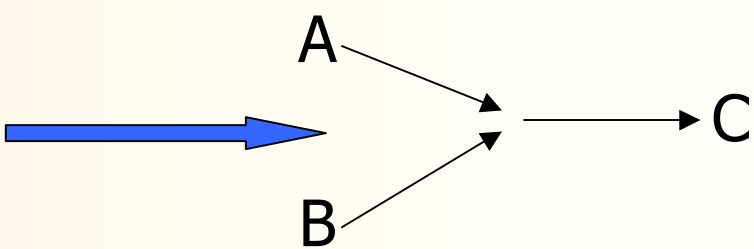
A <= B + A;



组合逻辑环路！！

C <= A;

C <= B;

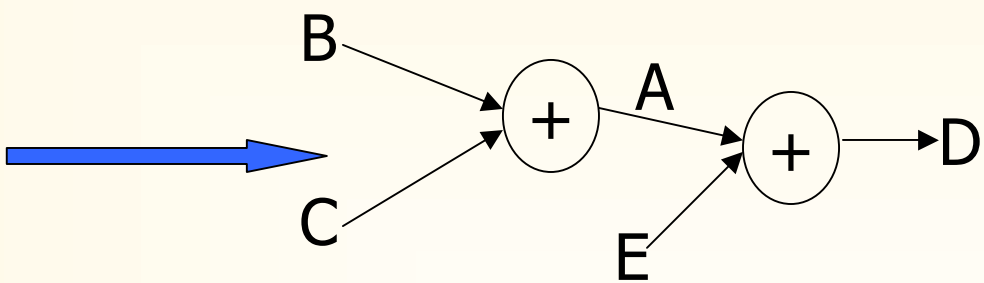


Multi-driver, need resolved function

2.2.2 顺序赋值语句

D <= A + E;

A <= B + C;

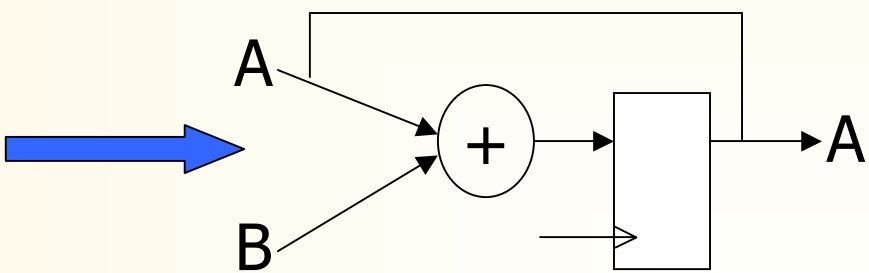


2.2.2 顺序赋值语句(Cont.)

C <= A;

C <= B;

A <= B + A;



组合进程：No!

时序进程：Ok!

2.3 小结

i. 一个设计由entity和architecture描述

ii. 数据对象类型有bit, boolean, integer, std_logic等标准类型和用户自定义类型

iii. 并发语句执行与书写顺序无关（order independent）

iv. VHDL是一种强类型语言（类型不能自动相互转换）

v. 元件例化语句不能在process中，if语句和for语句用于并发语句时，只能是generate语句，并发语句无变量赋值语句

2.3 小结(Cont.)

```
entity shift is
    port( reset, clk : in std_logic;
          din : in std_logic;
          dout : out std_logic);
end shift;
architecture str of shift is
    component dff
        port(reset, clk : in std_logic;
             d : in std_logic;
             q : out std_logic);
    end component;
    signal tv : std_logic_vector(0 to 7);
begin
    for I in 0 to 7 generate
        if I = 0 generate
            u1 : dff port map(reset, clk,
        din, tv(0));
        end generate;
        if I > 0 and I <= 7 generate
            u2 : dff port map(reset, clk,
        tv(I – 1), tv(I));
        end generate;  end generate;
    dout <= tv(7);      end str;
```

2.4 process

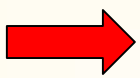
描述电路功能或行为。由于综合后的电路对所有输入信号变化敏感，因此所有被读信号均应包含在敏感表中，否则，综合前的模拟结果与综合后的模拟结果不一致！

2.4.1 syntax

```
[process_label :]  process(sensitivity list)
   [declarations;]
begin
      statements;    -- if, loop, case, subprogram call etc
end process [process_label];
```

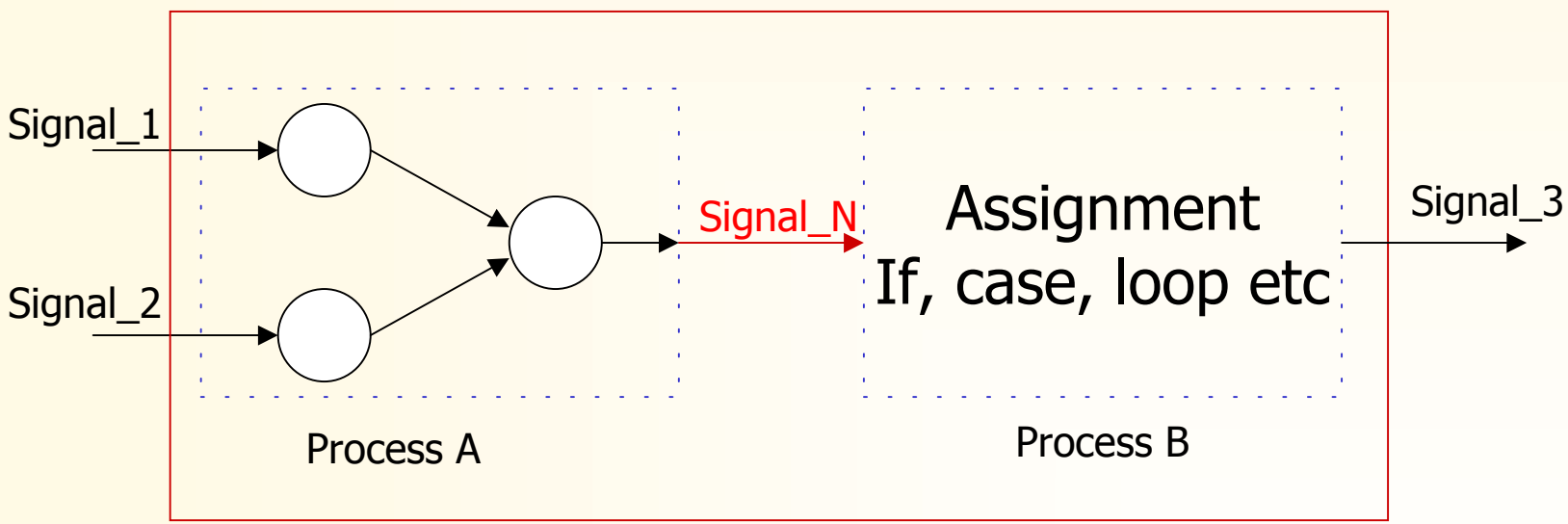
2.4.1 syntax

```
P1 : process(A, B)
Begin
    D < = A or B and C;
End process P1;
```



```
P1 : process(A, B, C)
Begin
    D < = A or B and C;
End process P1;
```

2.4.2 communication among process



architecture example

2.4.3 if then else语句综合

i/. 引入寄存器

```
entity dff is
    port (d : in std_logic;
          clk : in std_logic;
          q : out std_logic);
End dff;
```

```
architecture rtl of dff is
begin
    infer_reg : process(d, clk)
    begin
        if (clk'event and clk = '1') then
            q <= d;
    end process infer;
end rtl;
```

2.4.3 if then else语句综合(Cont.)

ii/. 引入锁存器

```
Infer_latch : process(A, B)
begin
    if (A = '1') then
        x <= B;
end process infer_infer_latch;
```

隐含了A = '0'时

x <= x;

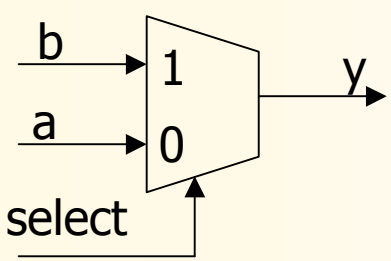
不完全的else

```
Infer_latch : process(A, B)
Begin
    x <= '0';
    if (A = '1') then
        x <= B;
end process infer_infer_latch;
```

2.4.3 if then else语句综合(Cont.)

iii/. 组合电路

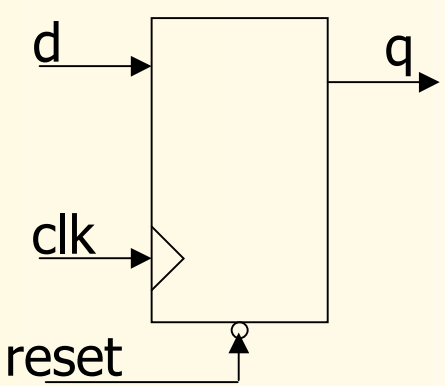
```
entity comb is
    port(a, b : in bit;
         select : in bit;
         y : out bit);
end comb;
```




```
architecture arch of comb is
begin
    process(a, b, select)
    begin
        if (select = '1') then
            y <= b;
        else y <= a;
        end if;
    end process;
end arch;
```

2.4.3 if then else语句综合(Cont.)

iv/. 异步复位

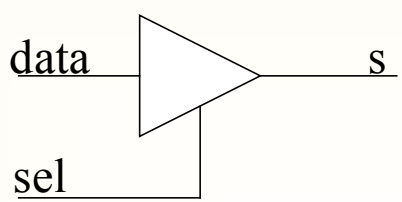



```
process(reset, clk, d)
Begin
    if (reset = '0') then
        q <= '0';
    elsif (clk'event and clk = '1') then
        q <= d;
    end if;
end process;
```

2.4.3 if then else语句综合(Cont.)

v/. 三态逻辑 1

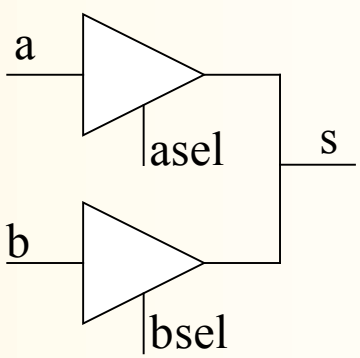
```
signal s, sel, data : std_logic;
process(sel, data)
Begin
    if (sel = '1') then
        s <= data;
    else  s <= 'Z';
    end if;
end process;
```



2.4.3 if then else语句综合(Cont.)

v/. 三态逻辑 2

```
architecture beh of tribuf is
    signal asel, bsel, a, b, s : std_logic;
begin    pa : process(asel, a)
    begin
        s <= 'Z';
        if (asel = '1') then
            s <= a;
        end if;
    end process pa;
```



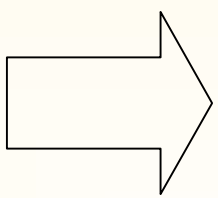
Multi driver!!

```
    pb : process(bsel, b)
    begin
        s <= 'Z';
        if (bsel = '1') then
            s <= b;
        end if;
    end process pb;
end beh;
```

2.4.3 if then else语句综合(Cont.)

Discussion 1

```
signal a, b, use_b : bit;
process(a, b, use_b)
Begin
    if (use_b = '1') then
        s <= b;
    elseif(use_b = '0') then
        s <= a;
    end if;
end process;
```

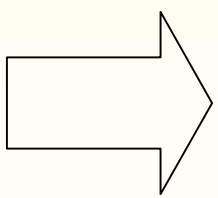


Latch?

2.4.3 if then else语句综合(Cont.)

Discussion 2

```
signal a, b, use_b : std_logic;
process(a, b, use_b)
Begin
    if (use_b = '1') then
        s <= b;
    elseif(use_b = '0') then
        s <= a;
    end if;
end process;
```

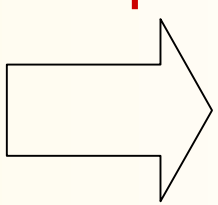


Latch?

2.4.3 if then else语句综合(Cont.)

Discussion 3

```
signal a, b, use_b : std_logic;
process(a, b, use_b)
Begin
    if (use_b = '1') then
        s <= b;
    elseif(use_b = '0') then
        s <= a;
    else assert false
        report "invalid use_b"
        severity error;
    end if;
end process;
```

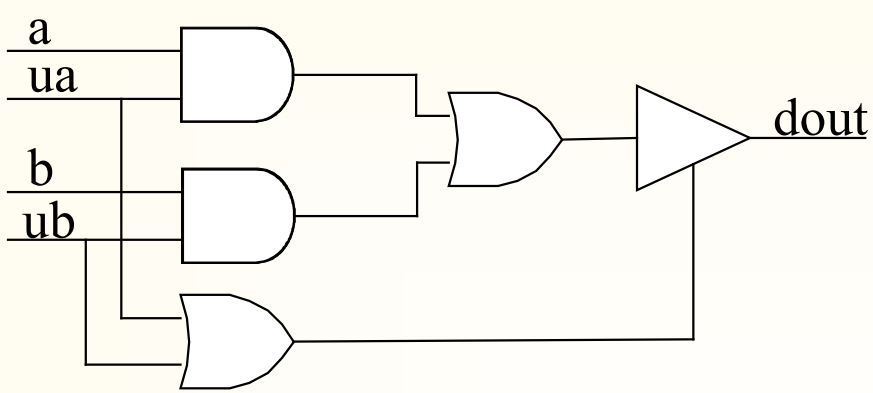


what?

2.4.3 if then else语句综合(Cont.)

vi/. 在一个进程中，一个信号只能对应一个三态驱动

```
process(b, ub, a, ua)
Begin
    dout <= 'Z';
    if (ub = '1') then
        dout <= b;
    end if;
    if (ua = '1') then
        dout <= a;
    end if;     end process;
```



2.4.3 if then else语句综合(Cont.)

vii/. ‘Z’值使用规则

1. 某个信号被赋值‘Z’值时，将会引入三态驱动，但‘Z’值不能用于复杂的表达式中（逻辑/算术表达式）

   如：dout <= ‘Z’ and din;

2. 当信号与‘Z’值比较时，结果总为false，引用这样的关系表达式将导致模拟与综合结果不匹配

   如：if (sel = ‘Z’) then 相当于

   if false then

2.4.3 if then else语句综合(Cont.)

vii/. if then else语句小结

1. 可以描述顺序行为
2. 生成组合逻辑
3. 如果缺少else语句，可能生成latch
4. 可以生成三态逻辑
5. 可以引入时序元件
6. 一般用于process和subprogram中

2.4.4 case 语句

i/. syntax

```
case (expression) is
    when value  =>  statements;
    when value | value  => statements;
    when value1 to value2  =>  statements;
                    ……
    when others  =>  statements;
end case;
```

2.4.4 case 语句(Cont.)

ii/. case 语句规则

- 任意value互不相同
- 如无others分枝，必须穷尽所有可能的表达式值
- 如有others分枝，必须至少有一个表达值未列出
- 任意分枝中的语句可以有多条

2.4.4 case 语句(Cont.)

```
entity decoder is
    port(din : in std_logic_vector(0 to 2);
         q : out std_logic_vector(0 to 7));
end decoder;
architecture ex of decoder is
begin
    process(din)
    begin
        case (din)
        when “000” => q <= “00000001”;
        when “001” => q <= “00000010”;
        when “010” => q <= “00000100”;
        when “011” => q <= “00001000”;
        when “100” => q <= “00010000”;
        when “101” => q <= “00100000”;
        when “110” => q <= “01000000”;
        when “111” => q <= “10000000”;
        when others => q <= “--------”;
        end case;
    end process;
end ex;
```

2.4.5 for loop语句

i/. 语法

loop_label : for loop_var in range loop

    sequential statements;

end loop loop_label;

说明：loop_label可选，loop_var不需声明，综合时，循环语句将unrolled

Why loop statement ?

2.4.5 for loop语句(Cont.)

```
entity parsum is
    port(word : in std_logic_vector(0 to 7);
         par : out std_logic);
end parsum;
architecture addxor of parsum is
begin
    process(word)
        variable ts : std_logic;
    begin
        ts : = '0';
        for I in word'range loop
            ts := ts xor word(I);
        end loop;
        par <= ts;
    end process;
end addxor;
```

Discussion: 如果ts为信号，结果是什么？

2.4.6 数据对象

constant variable signal

i/. constant and variable

格式

constant const_name : type := value;

variable var_name : type;

constant mask : std_logic_vector(0 to 7) := “00000000”;

constant led_zero : bit_vector(0 to 7) := X”7E”;

variable tsum : std_logic_vector(3 downto 0);

2.4.6 数据对象(Cont.)

ii/. signal and variable

相同点：值可变，可综合为逻辑或线

不同点：变量赋值有立即性，且只用于process和subprogram中（VHDL-1076-87），而信号除此之外，还可用于并行语句中

应用：简单计算 ⟶ signal

复杂计算 ⟶ variable

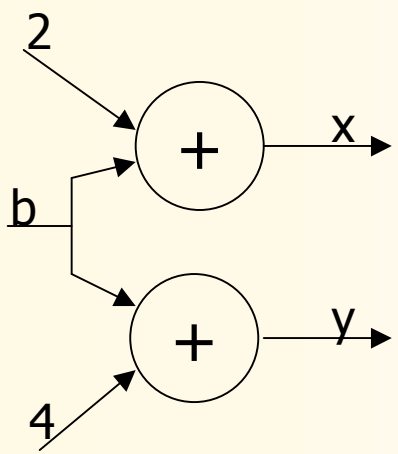
中间结果 ⟶ variable

2.4.6 数据对象(Cont.)

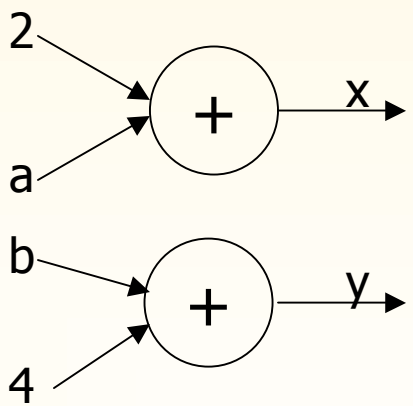
```
signal a, b, c, x, y : short
process(a, b, c)
begin
    c <= a;
    x <= c + 2;
    c <= b;
    y <= c + 4;
end process;
```

```
signal a, b, x, y : short
process(a, b, c)
    variable c : short;
begin
    c := a;
    x <= c + 2;
    c := b;
    y <= c + 4;
end process;
```

2.4.6 数据对象(Cont.)



signal



variable

2.4.6 数据对象(Cont.)

```
entity sigvar is
    port(a, b, c, d : std_logic;
         q : out std_logic;
end sigvar;
architecture sig of sigvar is
    signal int : std_logic;
Begin
    process(a, b, c, d, int)
    begin
        int <= a and b and c;
        q <= int or d;
    end process;
end sig;
```

```
entity sigvar is
    port(a, b, c, d : std_logic;
         q : out std_logic;
end sigvar;
architecture sig of sigvar is
Begin
    process(a, b, c, d)
        variablel int : std_logic;
    begin
        int := a and b and c;
        q <= int or d;
    end process;
end sig;
```

2.4.7 wait 语句

wait until, wait, wait on, wait for

- wait语句表明了信号激活process的条件
- 在process中，如有wait语句，敏感表必须取消
- 在process中，如既无wait语句，也无敏感表，则不能正确模拟
- 综合工具仅支持wait until语句（引入寄存器）!!!

2.4.7 wait 语句(Cont.)

```
library ieee;
Use ieee.std_logic_1164.all;
entity count4 is
    port(updown : in std_logic;
         clk : in std_logic;
         dout : out integer range 0 to 15);
end count4;
architecture behave of count4 is
    signal cnt : integer range 0 to 15;
begin
    process
    begin
        wait until clk'event and clk = '1';
        if (updown = '1') then
            if (cnt = 15) then
                cnt <= 0;
            else cnt <= cnt + 1;
            end if;
        else
            if (cnt = 0) then
                cnt <= 15;
            else cnt <= cnt – 1;
            end if;
        end if;
    end process;
end behave;
```

2.5 资源共享

硬件资源：

关系运算：=，/=，<，<=，>，>=

算术运算：+，—，*，/

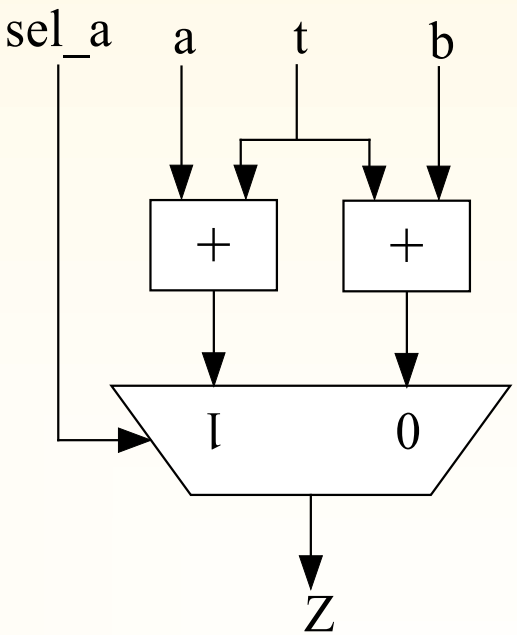
每个运算符均要消耗大量的硬件资源，并产生延时!!!

如何用最小的资源实现相同的功能？

2.5 资源共享(Cont.)

i/. 1

```
……
if (sel_a = ‘1’) then
    z <= a + t;
end if;
if (sel_a = ‘0’) then
    z <= b + t;
end if;
……
```



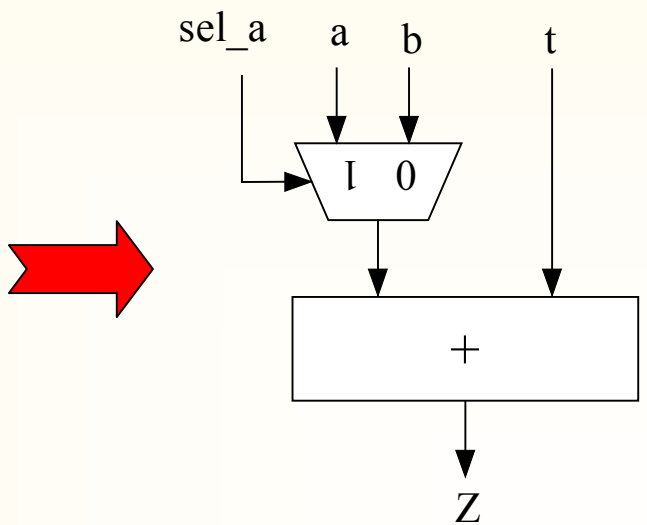
2.5 资源共享(Cont.)

i/. 2

```
z <= a + t  when  sel_a = '1'  else
        b + t;
```

或

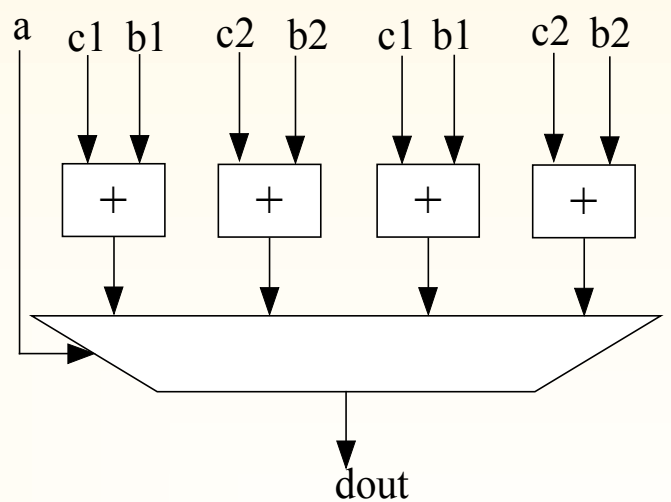
```
if (sel_a = '1') then
    z <= a + t;
else
    z <= b + t;
end if;
```



2.5 资源共享(Cont.)

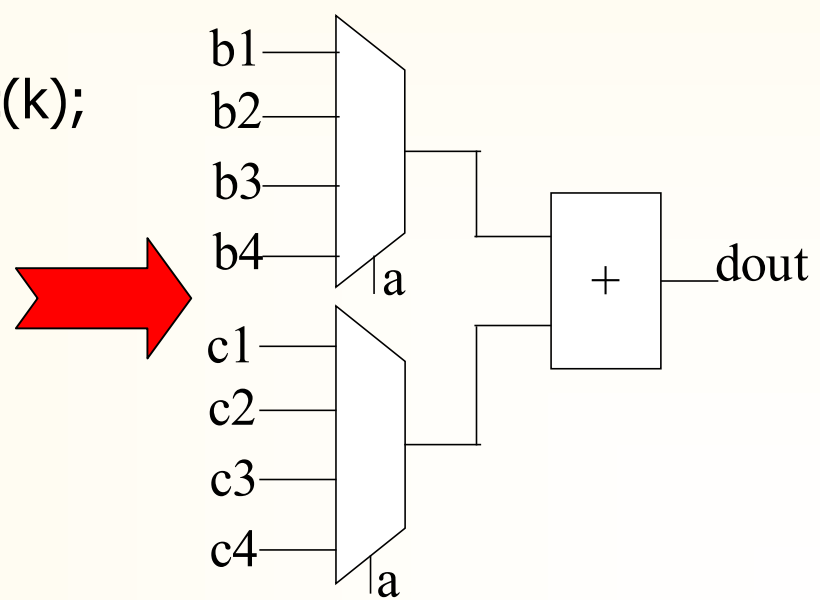
ii/. 1

```
dout := 0;
found := false;
for k in 1 to 4 loop
    if(found = false) then
        if (a(k) = '1') then
            dout := b(k) + c(k);
            found := true;
        end if;
    end if;
end loop;
```



2.5 资源共享(Cont.)

```
ii/.  t1 := 0;   t2 := 0;
2  found := false;
   for k in 1 to 4 loop
      if(found = false) then
         if (a(k) = '1') then
            t1 := b(k);  t2 := c(k);
            found := true;
         end if;
      end if;
   end loop;
   dout := t1 + t2;
```



2.5 资源共享(Cont.)

iii/. 多进程 1

```
P1 :  process
begin
    wait until clk'event and clk = '1';
    if (sel = "00") then
        reg_0 <= dA + dB;
    end if;
end process P1;
```

```
P2 :  process
begin
    wait until clk'event and clk = '1';
    if (sel = "01") then
        reg_1 <= dA - dB;
    end if;
end process P2;
```

2.5 资源共享(Cont.)

iii/. 多进程 2

```
P3 :  process
begin
    wait until clk'event and clk = '1';
    if (sel = "10") then
        reg_2 <= dB - dA;
    end if;
end process P1;
```

```
dout <= reg_0 when sel = "00" else
        reg_1 when sel = "01" else
        reg_2 when sel = "10" else
        0;
```

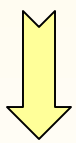
资源：3个加法器/减法器，资源共享不能在多进程间实现！

2.5 资源共享(Cont.)

iv/.

```
if((addr(31 downto 20) <= “000000000110”) and
   (addr(31 downto 20) >= “000000000001”)) then
```

资源：两个12位比较器



```
if((addr(31 downto 23) = “000000000”) and
   (addr(22 downto 20) /= “111”) and
   (addr(22 downto 20) /= “000”)) then
```

资源：两个3位比较器+一个9位比较器

2.5 资源共享(Cont.)

v/. 1

```
s(0) <= A(0) and A(1);
for k in 1 to 7 loop
    if (k > A – 1) then
        s(k) <= ‘1’;
    else
        s(k) <= ‘0’;
    end if;
end loop;
```

资源：1个减法器，7个比较器

说明：模拟时间长，因为每进入循环均要计算一次减法；综合时间也会较长，因为综合工具要移去循环中的定值表达式

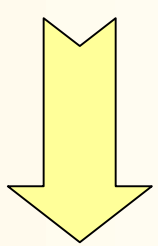
2.5 资源共享(Cont.)

v/. 2

```
s(0) <= A(0) and A(1);
tv := A – 1;
for k in 1 to 7 loop
    if (k > tv) then
        s(k) <= ‘1’;
    else
        s(k) <= ‘0’;
    end if;
end loop;
```

资源：1个减法器，7个比较器

说明：模拟、综合时间短



2.5 资源共享(Cont.)

v/. 3

```
s(0) <= A(0) and A(1);
for k in 1 to 7 loop
    if (k + 1 > A) then
        s(k) <= '1';
    else
        s(k) <= '0';
    end if;
end loop;
```

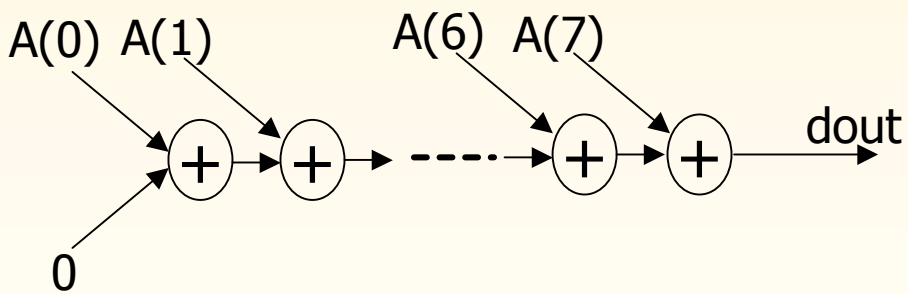
资源：0个减法器，7个比较器

说明：下标运算不产生额外资源消耗，综合工具自动用2 to 8与A比较；模拟、综合时间短

2.5 资源共享(Cont.)

vi/. 1

```
sum := 0;
for k in 0 to 7 loop
    sum := sum + A(k);
end loop;
dout <= sum;
```

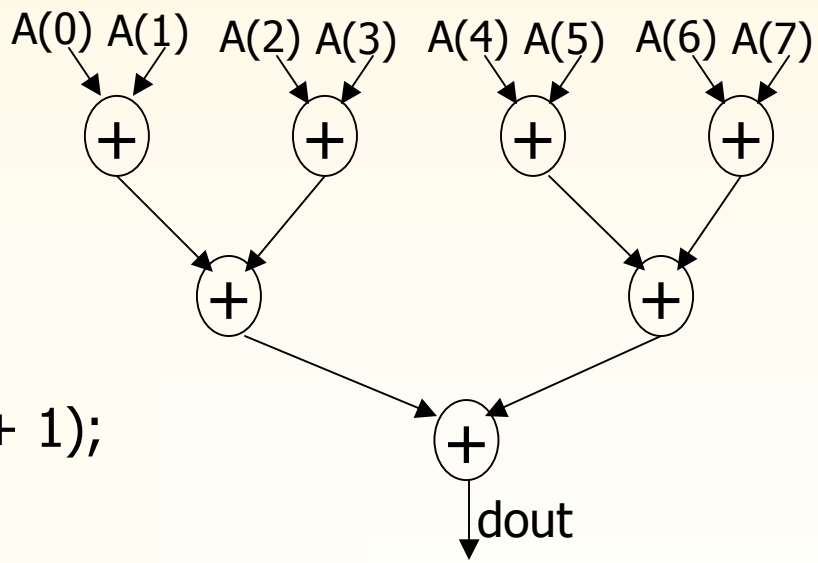


8个加法器，8级加法器延迟

2.5 资源共享(Cont.)

vi/. 2

```
cnt := 8;
for l in 0 to 2 loop
    cnt := cnt/2;
    for k in 0 to cnt – 1 loop
        A(k) := A(k*2) + A(k*2 + 1);
    end loop;
end loop;
dout <= A(0);
```

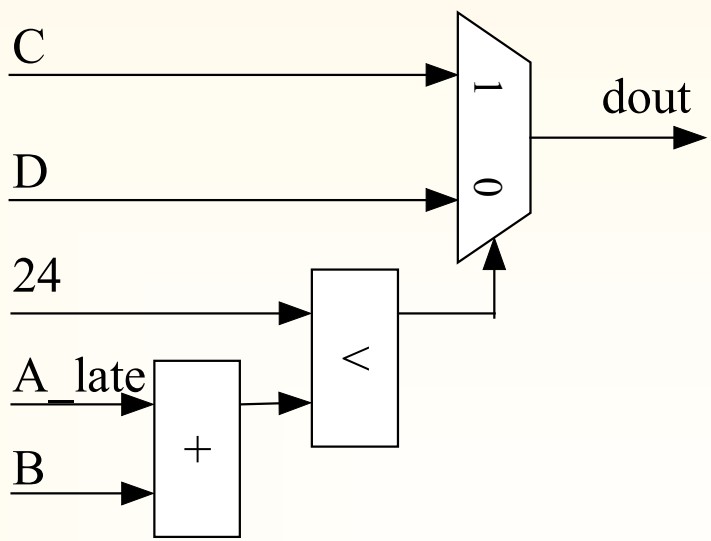


7个加法器，3级加法器延迟

2.6 其它

i/. 正确使用后到达的信号 1

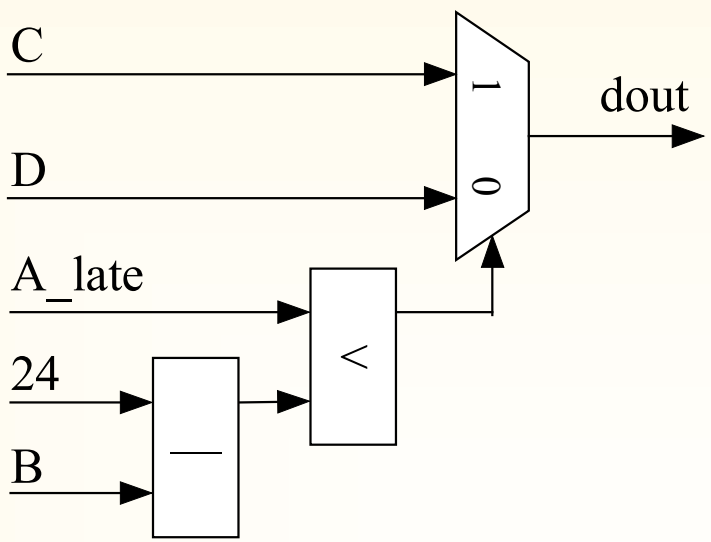
```
process(A_late, B, C, D)
begin
    if (A_late + B < 24) then
        dout <= C;
    else dout <= D;
    end if;
end process;
```



2.6 其它(Cont.)

i/. 正确使用后到达的信号 ②

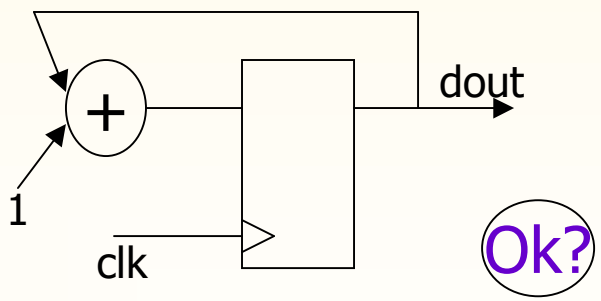
```
process(A_late, B, C, D)
begin
    if (A_late < 24 - B) then
        dout <= C;
    else dout <= D;
    end if;
end process;
```



2.6 其它(Cont.)

ii/. 仿真与综合结果不匹配 1

```
process
    variable cnt : integer range 0 to 255;
begin
    wait until clk'event and clk = '1';
    cnt := cnt + 1;
    dout <= cnt;
end process;
```

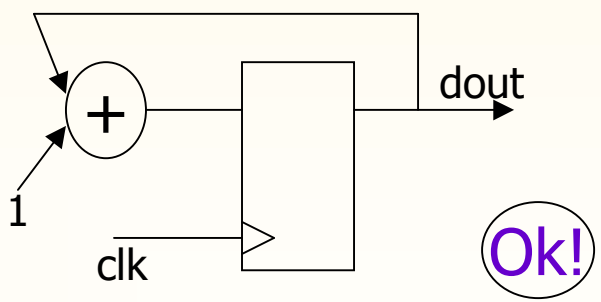


变量将引入额外的寄存器！

2.6 其它(Cont.)

ii/. 仿真与综合结果不匹配 ②

```
    signal cnt : integer range 0 to 255;
process
begin
    wait until clk'event and clk = '1';
    cnt <= cnt + 1;
end process;
dout <= cnt;
```

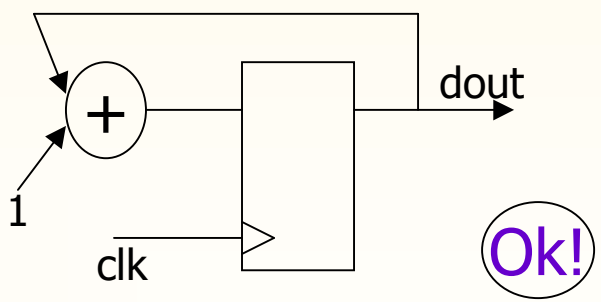


但是，仿真时，当cnt的值为255时，再一次进入该process时，将报告越界错误！

2.6 其它(Cont.)

ii/. 仿真与综合结果不匹配 3

```
    signal cnt : integer range 0 to 255;
process
begin
    wait until clk'event and clk = '1';
    if (cnt = 255) then
        cnt <= 0;
    else  cnt <= cnt + 1;
end process;
dout <= cnt;
```



2.6 其它(Cont.)

iii/. 避免不必要的重复调用函数

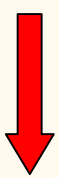
```
function add8(in1, in2 : bit_vector(7 downto 0))
is return bit_vector(7 downto 0);

                ……

sign_bit := add8(A, B)(7);

lower_nibble := add8(A, B)(3 downto 0);

                ……
```



```
tv := add8(A, B);

sign_bit := tv(7);

lower_nibble := tv(3 downto 0);
```

2.6 其它(Cont.)

iv/. 元件例化端口映射问题 1

```
inst1 : comp1  port map(din_1 => '0';
                        din_2 => con_A;
                        dout => con_out);
```

```
signal gnd : std_logic;
        ……
gnd <= '0';
inst1 : comp1  port map(din_1 => gnd;
                        din_2 => con_A;
                        dout => con_out);
```

当某一输入端口接固定电平时，必须引入中间信号，且中间信号不能在说明时赋初值！

2.6 其它(Cont.)

iv/. 元件例化端口映射问题 2

```
component dff
    port(reset, clk : in std_logic;
         d : in std_logic;
         q, qn : out std_logic);
end dff;
```

当某一输出端悬空时，应连接**open**关键字！

```
u1 : dff port map(reset => reset, clk => clk;
                  q => dout, qn => open);
```

2.6 其它(Cont.)

v/. 避免阵列方向错误

```
signal data8 : bit_vector(0 to 7);
signal dout : bit_vector(7 downto 0);
            ……
dout <= data8(7 downto 0);
```

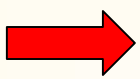
```
for I in 0 to 7 loop
    dout(I) <= data8(7 – I);
end loop;
```

2.6 其它(Cont.)

vi/. 避免低效率语句 1

```
    signal : I1, I2 : bit;
P1 : process(I1, I2, A, B, C, D)
begin
    case (I1 & I2) is
    when "00" => dout <= A;
    when "01" => dout <= B;
    when "10" => dout <= C;
    when "11" => dout <= D;
    end case;
end process P1;
```

No!



```
    signal : I1, I2 : bit;
P2 : process(I1, I2, A, B, C, D)
begin
    if (I1 & I2 = "00") then
        dout <= A;
    elsif (I1 & I2 = "01") then
        dout <= B;
    elsif (I1 & I2 = "10") then
        dout <= C;
    else  dout <= D;
    end if;
end process P2;
```

2.6 其它(Cont.)

vi/. 避免低效率语句 2

```
    signal : l1, l2 : bit;
P3 : process(l1, l2, A, B, C, D)
    subtype twobits is bit_vector(0 to 1);
begin
    case (twobits'(l1 & l2)) is
    when "00" => dout <= A;
    when "01" => dout <= B;
    when "10" => dout <= C;
    when "11" => dout <= D;
    end case;
end process P3;
```

2.6 其它(Cont.)

vii/. 桶移位寄存器 1

```
entity barrel_shifter is
    port(clk : in bit;
         din : in bit_vector(0 to 31);
          k : in integer range 0 to 31;
         dout : out bit_vector(0 to 31));
end barrel_shifter;
```

```
architecture behave of barrel_shifter is
begin
    process
    begin
        wait until clk'event and clk = '1';
        dout <= din(k to 31) & din(0 to k);
    end process;
end behave;
```

上述模型不可综合，k是0~31中任意的数，
属于不可计算的，但上述模型可仿真！

2.6 其它(Cont.)

vii/. 桶移位寄存器 2

```
architecture syn1 of barrel_shifter is
begin
    process
    begin
        wait until clk'event and clk = '1';
        if (k = 0) then dout <= din;
        else
            for l in 1 to 31 loop
                if (l = k) then
                    dout <= din(l to 31) & din(0 to l);
                end if;
            end loop;
        end if;
    end process;
end syn1;
```

2.6 其它(Cont.)

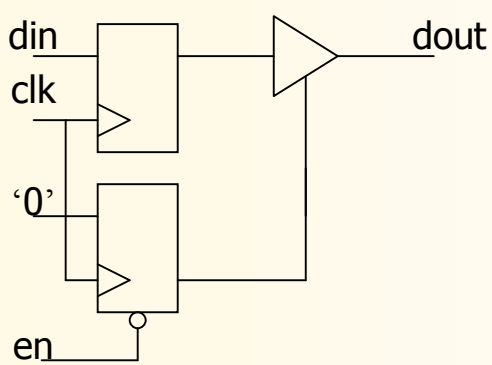
vii/. 桶移位寄存器 3

```
architecture syn2 of barrel_shifter is
begin
    process
    begin
        wait until clk'event and clk = '1';
        case (k) is
            when 0 => dout <= din;
            when 1 => dout <= din(1 to 31) & din(0);
                ……
            when 31 => dout <= din(31) & din(0 to 30);
        end case;
    end process;
end syn2;
```

2.6 其它(Cont.)

viii/. 三态使能寄存器（或锁存器）

```
entity trienreg is
    port(clk : in std_logic;
         en : in std_logic;
         din : in std_logic;
         dout : out std_logic);
end trienreg;
```

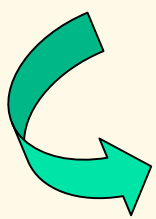



```
architecture rtl of trienreg is
begin
    process(clk, en, din)
    begin
        if (en = '0') then
            dout <= 'Z';
        elsif (clk'event and clk = '1') then
            dout <= din;
        end if;
    end process;
end rtl;
```

2.6 其它(Cont.)

ix/. 向量操作与for loop

```
process(scalar_A, vector_B)
begin
    for k in vector_B'range loop
        vector_C(k) <= vector_B(k) and scalar_A);
    end loop;
end process;
```

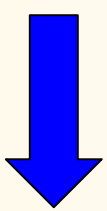


```
process(scalar_A, vector_B)
    variable tmp : std_logic_vector(vector_B'range);
begin
    tmp := (others => scalar_A);
    vector_C <= tmp and vector_B;
end process;
```

2.6 其它(Cont.)

ix/. 向量范围

```
entity case_ex is
    port(a : in std_logic_vector(4 downto 0);
         q : out std_logic_vector(2 downto 0));
end case_ex;
architecture rtl of case_ex is
begin
    process(a)
    begin
        case a is
            when “00000” => q <= “011”;
            when “00001” to “11100” =>
                                q <= “101”;
            when others => q <= “000”;
        end case;
    end process;
end rtl;
```



2.6 其它(Cont.)

ix/. 向量范围

```
process(a)
    variable int : integer range 0 to 31;
begin
    int := conv_integer(a);
    case int is
        when 0 => q <= “011”;
        when 1 to 30 => q<= “101”;
        when others => q<= “000”;
    end case;
end process;
```

向量没有范围，因此，
不能指定向量的范围！！

2.6 其它(Cont.)

x/. 时钟信号

```
type not_single_bit is ('X', '0', '1', 'Z');
signal invalid_clk : not_signal_bit;
        ……
wait until invalid_clk'event and invalid_clk = '1';
-- or
if(invalid_clk'event and invalid_clk = '1') then
```

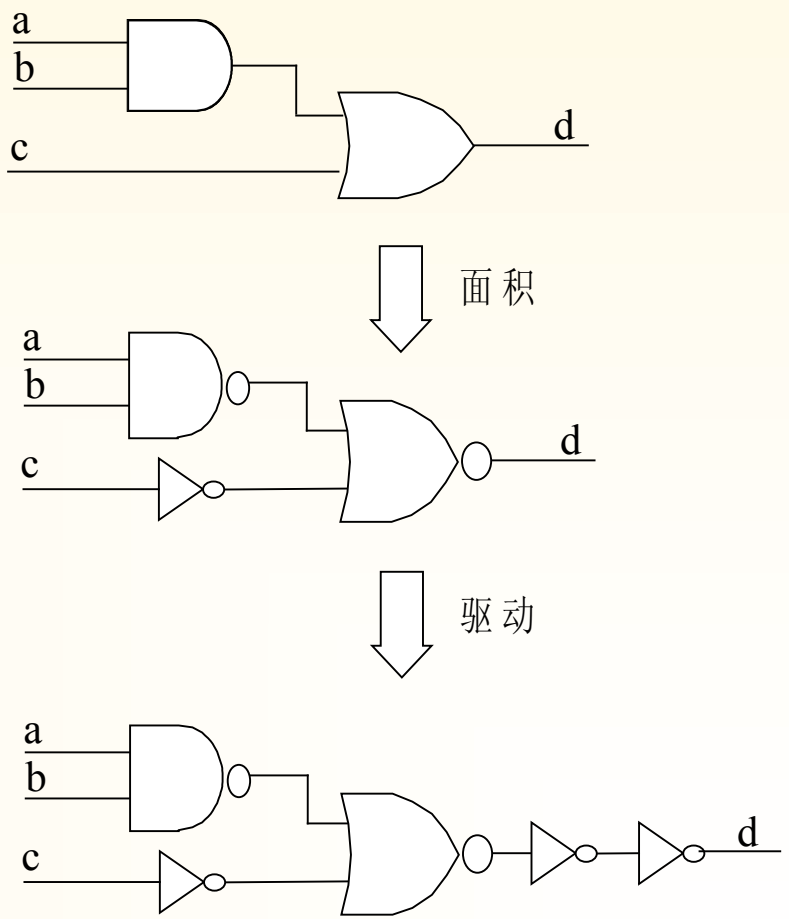
illegal usage!

沿触发时钟信号必须是能够用单比特编码的枚举类型，但std_logic类型是一个例外！

2.6 其它(Cont.)

xi/. 综合结果多样性

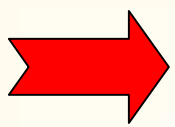
```
entity comb is
    port(a, b, c : in std_logic;
         d : out std_logic);
end comb;
architecture rtl of comb is
begin
    d <= (a and b) or c;
end rtl;
```



2.6 其它(Cont.)

xii/. 使用中间数据项

```
Sum1 <= a + b;
Sum2 <= a + b + c;
Sum3 <= a + c;
```

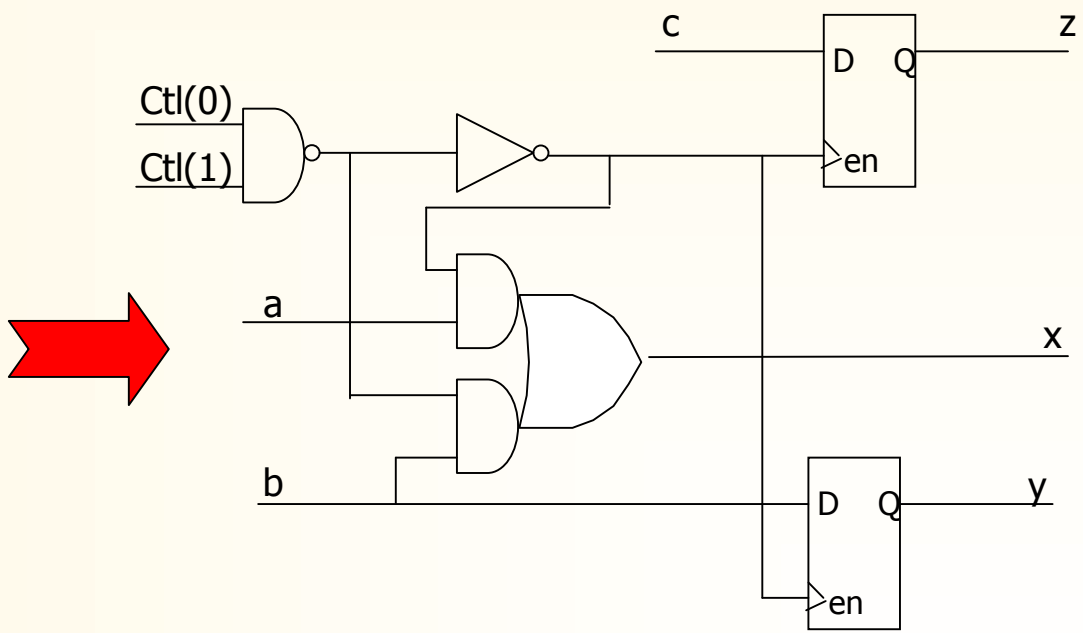


```
Sum1 <= a + b;
Sum2 <= Sum1 + c;
Sum3 <= a + c;
```

2.6 其它(Cont.)

xiii/. 消除潜在的latch 1

```
process(ctl, a, b, c)
begin
    if (ctl > 2) then
        x <= a;
        y <= b;
        z <= c;
    else x <= b;
    end if;
end process;
```



2.6 其它(Cont.)

xiii/. 消除潜在的latch 2

```
process(bcd)
begin
    case(bcd) is
    when “000” => zero<= ‘1’; one <= ‘0’; two <= ‘0’;
    when “001” => zero<= ‘0’; one <= ‘1’; two <= ‘0’;
    when “010” => zero<= ‘0’; one <= ‘0’; two <= ‘1’;
    when others => zero<= ‘-’; one <= ‘-’; two <= ‘-’;
    end case;
end process;
```

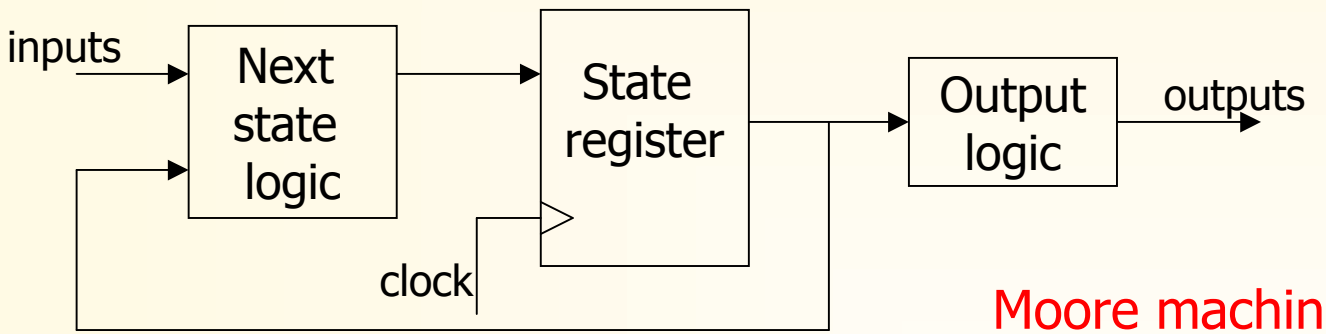
2.6 其它(Cont.)

xiii/. 消除潜在的latch 2

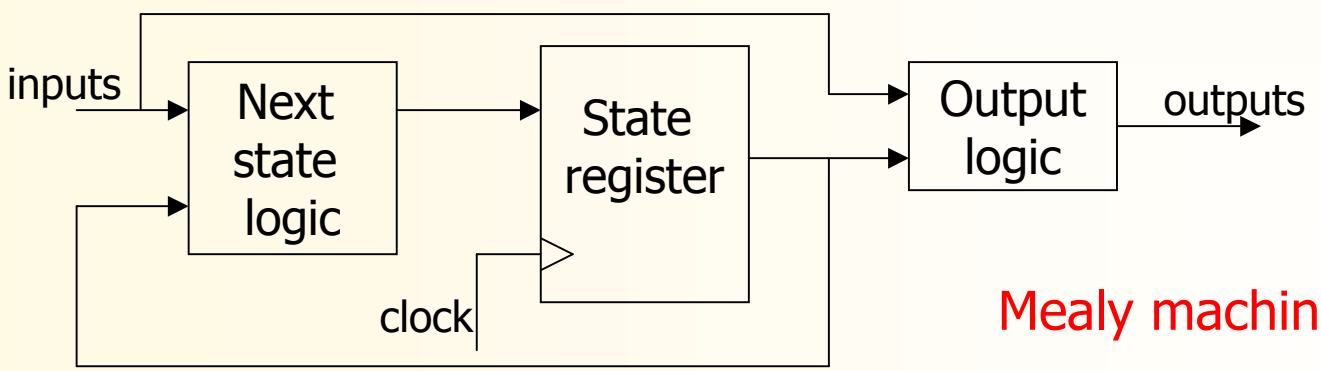
```
process(bcd)
begin
    case(bcd) is
    when “000” => zero<= ‘1’;
    when “001” => one <= ‘1’;
    when “010” => two <= ‘1’;
    when others => zero<= ‘-’; one <= ‘-’; two <= ‘-’;
    end case;
end process;
```

2.6 其它(Cont.)

xiv/. 有限状态机 1



Moore machine

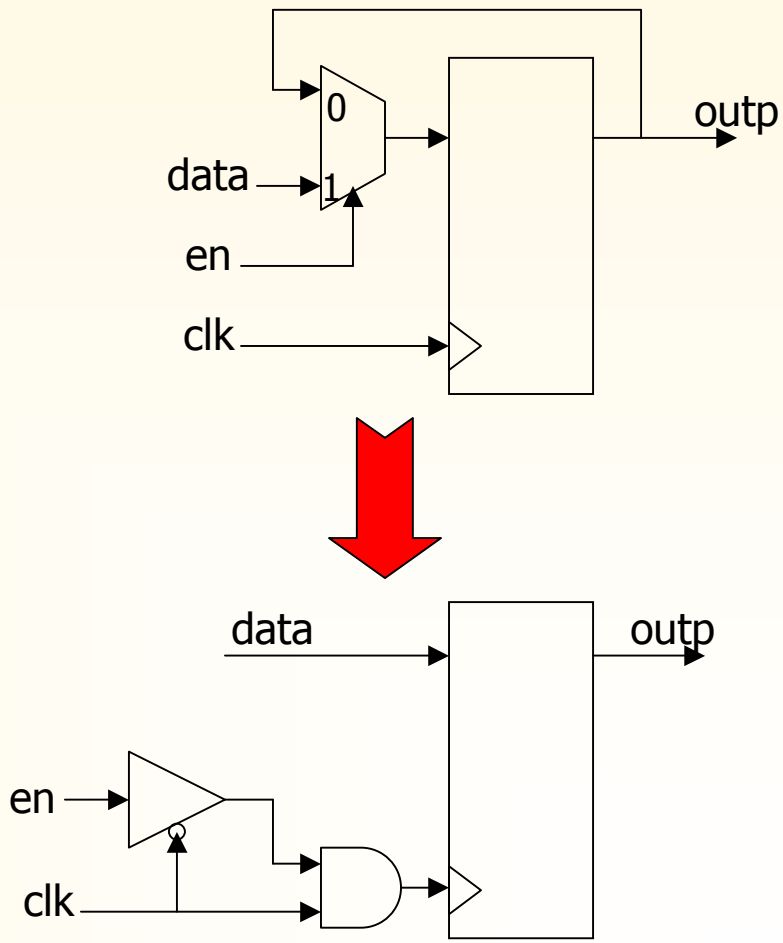


Mealy machine

2.6 其它(Cont.)

xv/. Clock gating

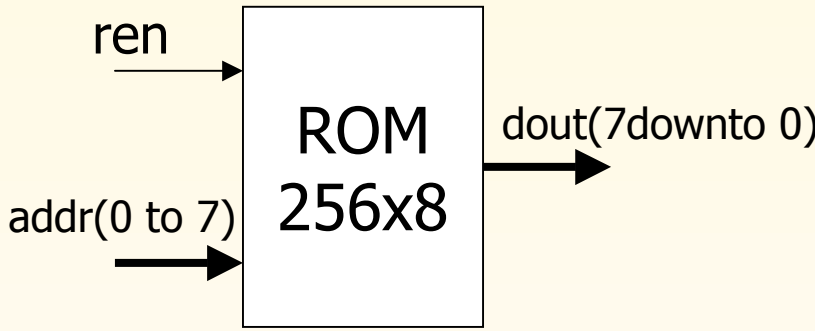
```
module clkgate(en, data, clk, outp)
input en, clk;
input [7:0] data;
output [7:0] outp;
reg [7:0] outp;
Always @(posedge clk)
    if (en) outp <= data;
endmodule
```



三、设计实例

- ROM
- RAM
- FIFO
- ADDER
- ALU
- MCU

3.1 ROM




```
library ieee;
use ieee.std_logic_1164.all;
package ROM_P is
    subtype byte is std_logic_vector(7 downto 0);
    type memory is array (0 to 255) of byte;
    constant rom_data : memory := ((“11010101”),
                (“01001000”),
                      ……
                (“10101010));
end ROM_P;
```

3.1 ROM(Cont.)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.ROM_P.all;
entity rom_e is
    port(ren : in std_logic;
         addr : in byte;
         dout : out byte);
end rom_e;
```

```
architecture rom_a of rom_e is
begin
    process(ren, addr)
    begin
        if (ren = '1') then
            dout <= rom_data(conv_integer(addr));
        else
            dout <= (others =>'Z');
        end if;
    end process;
end rom_a;
```

3.2 优先编码器

inp7	inp6	inp5	inp4	inp3	inp2	inp1	inp0	y(2 downto 0)
x	x	x	x	x	x	x	0	111
x	x	x	x	x	x	0	1	110
x	x	x	x	x	0	1	1	101
x	x	x	x	0	1	1	1	100
x	x	x	0	1	1	1	1	011
x	x	0	1	1	1	1	1	010
x	0	1	1	1	1	1	1	001
0	1	1	1	1	1	1	1	000

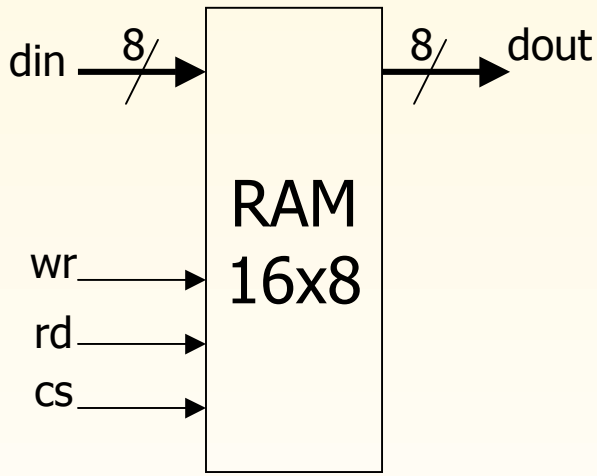
3.2 优先编码器(Cont.)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.ROM_P.all;
entity encoder is
    port(inp : in
std_logic_vector(7 downto 0);
         y : out
std_logic_vector(2 downto 0));
end rom_e;
```

3.2 优先编码器(Cont.)

```
architecture encoder_a of encoder is
begin
    process(inp)
    begin
        if (inp(0) = '0') then
            y <= "111";
        elsif(inp(1) = '0') then
            y <= "110";
        elsif(inp(2) = '0') then
            y <= "101";
        elsif(inp(3) = '0') then
            y <= "100";
        elsif(inp(4) = '0') then
            y <= "011";
        elsif(inp(5) = '0') then
            y <= "010";
        elsif(inp(6) = '0') then
            y <= "001";
        else
            y <= "000";
        end if;
    end process;
end encoder_a;
```

3.3 RAM




```
entity sram16x8 is
    generic(k : integer := 8;
            w : integer := 4);
    port(wr, rd, cs : in std_logic;
         addr : in std_logic_vector(w – 1 downto 0);
         din : in std_logic_vector(k – 1 downto 0);
         dout : out std_logic_vector(k – 1 downto 0));
end sram16x8;
```

3.3 RAM(Cont.)

```
architecture behav of sram16x8 is
    subtype word is std_logic_vector(k – 1 downto 0);
    type memory is array (0 to 2 ** w – 1) of word;
    signal addr_tmp : integer range 0 to 2 ** w – 1;
    signal sram : memory;
    signal din_change, wr_rise : TIME := 0 ns;
begin
    addr_tmp <= conv_integer(addr);
    process(wr)
    begin
        if (wr'event and wr = '1') then
```

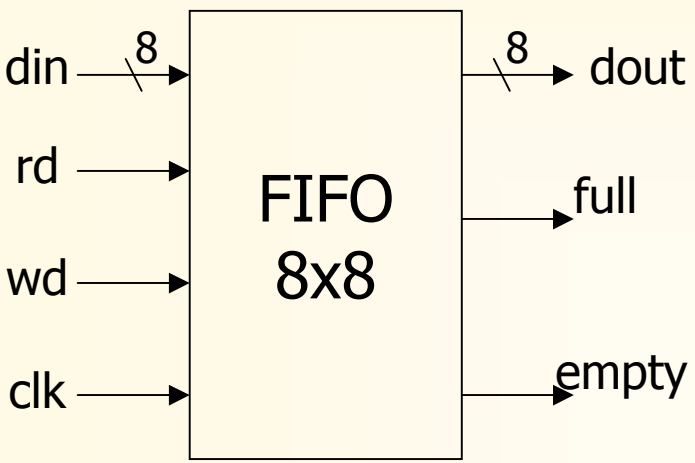
3.3 RAM(Cont.)

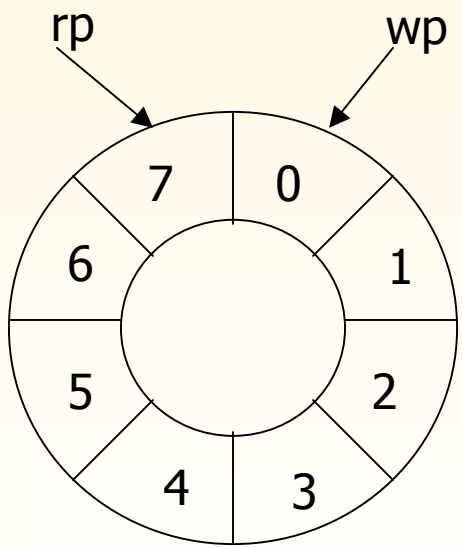
```
        if (cs = '1' and wr = '1') then
            sram(addr_tmp) <= din after 2 ns;
        end if;
    end if;
    wr_rise <= NOW; --rising time of wr
    assert(NOW – din_change >= 800 ps)
    report "setup error din(sram)"
    severity WARNING; -- din setup time checking
end process;
```

3.3 RAM(Cont.)

```
process(rd, cs)
begin
    if (cs = '1' and rd = '0') then
            dout <= sram(addr_tmp) after 3 ns;
    else dout <= "ZZZZZZZZ";
    end if;
end process;
process(din)
begin
    din_change <= NOW;
    assert (NOW – wr_rise >= 300 ns)
        report "hold error din(sram)"
        severity WARNING;
    end process;
end behav;
```

3.4 FIFO





满、空状态：rp = wp – 1

满状态的前一状态：rp = wp – 1，再写则为满状态

空状态的前一状态：rp = wp – 2，再读则为空状态

3.4 FIFO(Cont.)

```
library ieee;
Use ieee.std_logic_1164.all;

entity fifo8x8 is
    generic(w : integer := 8;
            k : integer := 8);
    port(clk, reset, wr, rd : in std_logic;
         din : in std_logic_vector(k – 1 downto 0);
         dout : out std_logic_vector(k – 1 downto 0);
         full, empty : out std_logic);
end fifo8x8;
```

- wr, rd：低电平有效！
- 该FIFO设计成写当前地址，读下一地址数据

3.4 FIFO(Cont.)

```
architecture behav of fifo8x8 is
    type memory is array( 0 to w – 1) of std_logic_vector(k – 1 downto 0);
    signal ram : memory;
    signal wp, rp : integer range 0 to w – 1;
    signal in_full, in_empty : std_logic;
begin
    full <= in_full;
    empty <= in_empty;
    dout <= ram(rp);
```

3.4 FIFO(Cont.)

```
process(clk)
begin
    if (clk'event and clk = '1') then
        if (wr = '0' and in_full = '0') then
            ram(wp) <= din;
        end if;
    end if;
end process;
```

数据写堆栈

```
process(clk, reset)
begin
    if (reset = '1') then
        wp <= 0;
    elsif (clk'event and clk = '1') then
        if (wr = '0' and in_full = '0') then
            if (wp = w – 1) then
                wp <= 0;
            else wp <= wp + 1;
            end if;
        end if;
    end if;
end process;
```

wp 修改描述

3.4 FIFO(Cont.)

```
process(clk, reset)
begin
    if (reset = '1') then
        rp <= w – 1;
    elsif (clk'event and clk = '1') then
        if (rd = '0' and in_empty = '0') then
            if (rp = w – 1) then
                rp <= 0;
            else rp <= rp + 1;
            end if;
        end if;
    end if;
end process;
```

rp 修改描述

3.4 FIFO(Cont.)

```
process(clk, reset)
begin
    if (reset = '1') then
        in_empty <= '1';
    elsif (clk'event and clk = '1') then
        if ((rp = wp – 2 or (rp = w – 1 and wp = 1) or
            (rp = w – 2 and wp = 0)) and (rd = '0' and wr = '1')) then
            in_empty <= '1';
        elsif (in_empty = '1' and wr = '0') then
            in_empty <= '0';
        end if;
    end if;
end process;
```

Empty标志产生描述

3.4 FIFO(Cont.)

```
process(clk, reset)
begin
    if (reset = '1') then
        in_full <= '1';
    elsif (clk'event and clk = '1') then
        if (rp = wp and rd = '1' and wr = '0') then
            in_full <= '1';
        elsif (in_full = '1' and rd = '0') then
            in_full <= '0';
        end if;
    end if;
end process;
end behav;
```

Full标志产生描述

3.5 参数可变的加减器

```
package mymath is
    function add_sub (L, R : bit_vector;
                      ADD : boolean) return bit_vector;
end mymath;
package body mymath is
    function add_sub (L, R : bit_vector;
                      ADD : boolean) return bit_vector is
        variable carry : bit;
        variable A, B, Sum : bit_vector(L'length – 1 downto 0);
    begin
        if ADD then
            A := L;
            B := R;
```

3.5 参数可变的加减器(Cont.)

```
            carry := '0';
        else
            A := L;
            B := not R;
            carry := '1';
        end if;
        for I in 0 to A'left loop
            Sum(I) : A(I) xor B(I) xor carry;
            carry := (A(I) and B(I)) or (A(I) and carry) or (B(I) and carry);
        end loop;
        return Sum;
    end add_sub;
end mymath;
```

3.5 参数可变的加减器(Cont.)

```
library ieee;
use ieee.std_logic_1164.all;
use work.mymath.all;
entity varlen is
    port(op1, op2 : in bit_vector(0 to 7);
         as : in boolean;
         result : out bit_vector(0 to 7));
end varlen;
```

```
architecture func of varlen is
begin
    process(op1, op2, as)
    begin
        res <= add_sub(op1, op2, as);
    end process;
end func;
```

3.6 流水结构的微控器

1. 定义微控器的指令集
2. 定义微控器的体系结构
3. 定义微控器及其模块的界面
4. testbench

3.6流水结构的微控器（Cont.）

本微控器的一些假定:

- 是一个最简单的微控器，仅有8个指令
- 无外部存储器接口
- 无寄存器记分板
- 寄存器位宽为32位
- 有16个通用寄存器
- 假定有一个指令缓存和指令存储器为该微控器提供指令和数据

3.6 流水结构的微控器（Cont.）

指令集定义

Instruction Set	Description
MOVE src1 dest	Move the contents of src1 to dest
ADD src1 src2 dest	Add the contents of src1 with contents of src2, and put the result in dest
SUB src1 src2 dest	Sub the contents of src1 with contents of src2, and put the result in dest
MUL src1 src2 dest	Multiply the contents of src1 with the contents of src2, and put the result in dest

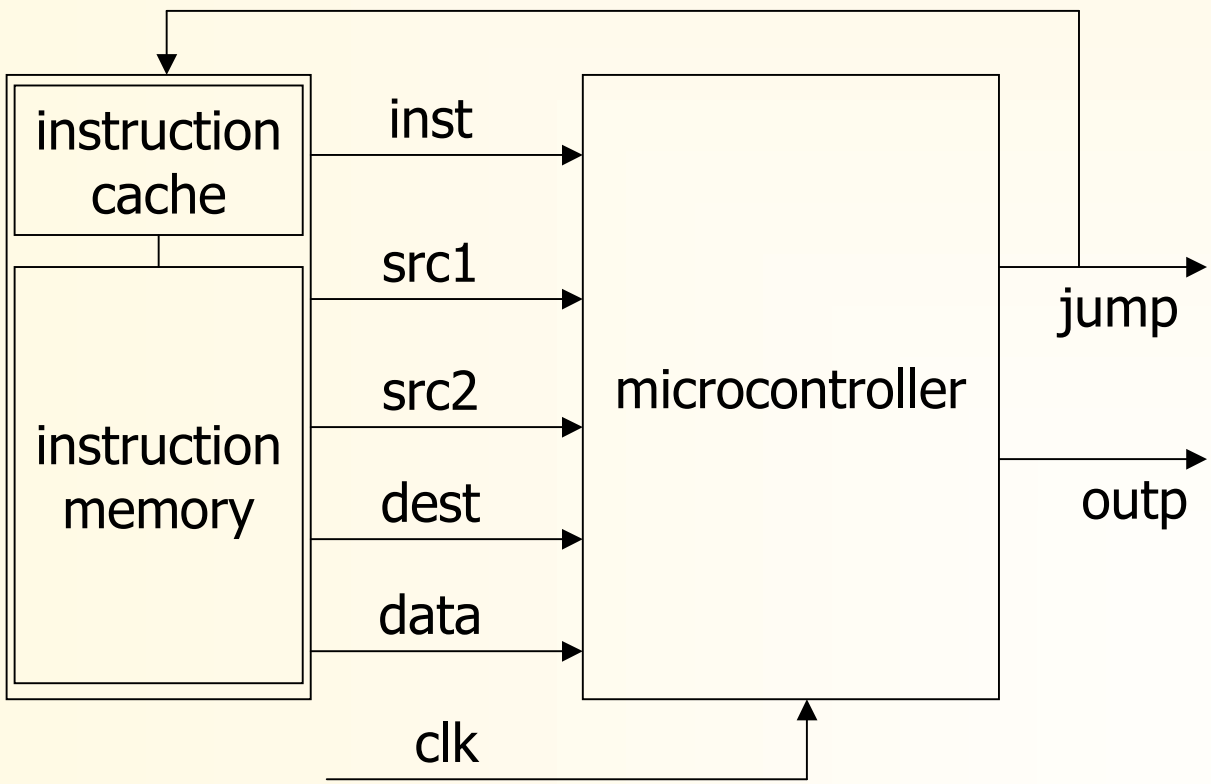
3.6 流水结构的微控器（Cont.）

指令集定义

Instruction Set	Description
CJE src1 src2 CODE	Compare contents of src1 with the contents of src2, if they are equal, jump to the the portion of the instruction labeled CODE
LOAD value dest	Load the contents of value into destination
READ dest	Read the contents of dest and drive the data on the output pin
NOP	No instruction

3.6 流水结构的微控器（Cont.）

微控器界面



3.6 流水结构的微控器（Cont.）

微控器信号描述

Pins	Input/Output	Bit Size	Description
inst	Input	3	Instructions of the microcontroller 000-----MOVE 001-----ADD 010-----SUB 011-----MUL 100-----CJE 101-----LOAD 110-----READ 111-----NOP

3.6 流水结构的微控器（Cont.）

微控器信号描述

Pins	Input/Output	Bit Size	Description
src1	Input	4	Source1 address
src2	Input	4	Source2 address
dest	Input	4	destination address
data	Input	32	Input data for LOAD instruction
clk	Input	1	
jump	output	1	Assert high when the CJE instr. compares src1 and src2 as having the same value
outp	output	32	Output of data for READ instr.

3.6 流水结构的微控器（Cont.）

微控器内部寄存器

src1/src2/dest	register	Register name
0000	register 0	reg0
0001	register 1	reg1
0010	register 2	reg2
0011	register 3	reg3
0100	register 4	reg4
0101	register 5	reg5
0110	register 6	reg6
0111	register 7	reg7
1000	register 8	reg8

3.6 流水结构的微控器（Cont.）

微控器内部寄存器

src1/src2/dest	register	Register name
1001	register 9	reg9
1010	register 10	reg10
1011	register 11	reg11
1100	register 12	reg12
1101	register 13	reg13
1110	register 14	reg14
1111	register 15	reg15

3.6 流水结构的微控器（Cont.）

微控器流水线结构

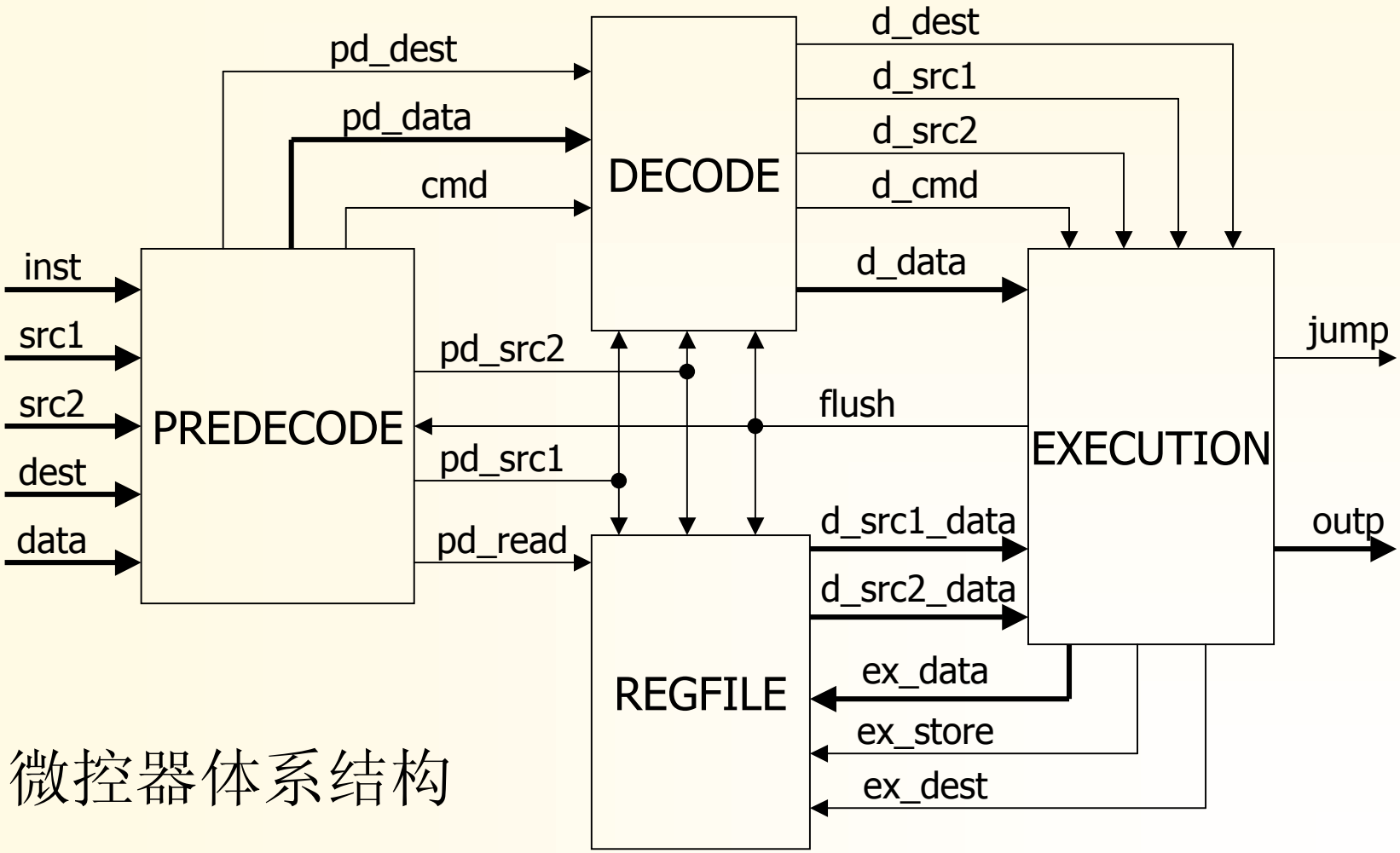
```
LOAD  #FA, reg0
LOAD  #1, reg1
ADD   reg0, reg1, reg13
READ  reg13
```

3.6 流水结构的微控器（Cont.）

微控器流水线结构

TIME	Predecode stage	Decode stage	Execute stage
1	**LOAD #FA, reg0**		
2	**LOAD #1, reg1**	**LOAD #FA, reg0**	
3	**ADD reg0, reg1, reg13**	**LOAD #1, reg1**	**LOAD #FA, reg0**
4	**READ reg13**	**ADD reg0, reg1, reg13**	**LOAD #1, reg1**
5		**READ reg13**	**ADD reg0, reg1, reg13**
6			**READ reg13**

3.6 流水结构的微控器（Cont.）



微控器体系结构

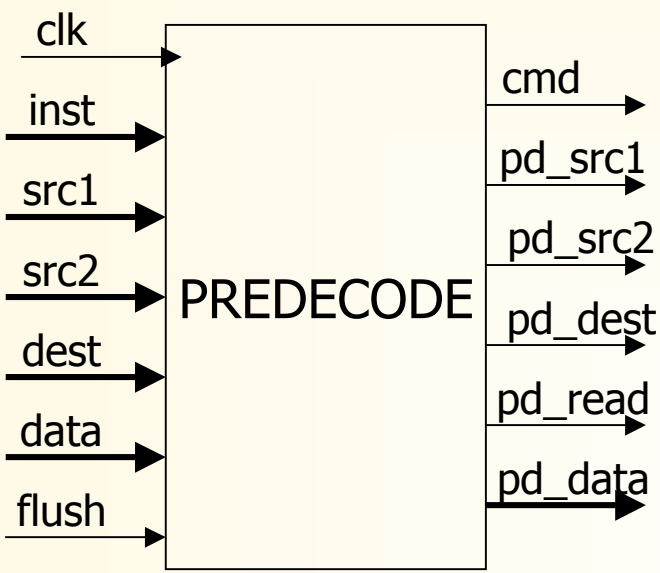
3.6 流水结构的微控器（Cont.）

数据类型定义：

```
library ieee;
use ieee.std_logic_1164.all;
package pipeline_pkg is
    type command_type is (MOVE, ADD, SUB, MUL, CJE, LOAD, READ, NOP);
    type reg_type is (reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7, reg8,
                      reg9, reg10, reg12, reg13, reg14, reg16);
    type array_size is array (0 to 15) of std_logic_vector(31 downto 0);
    constant ZERO : std_logic_vector(31 downto 0) := others => '0');
end pipeline_pkg;
```

3.6 流水结构的微控器（Cont.）

PREDECODE Block:



3.6 流水结构的微控器（Cont.）

Signals decription of PREDECODE:

Signal Name	I/O	Description
clk	input	clock input
inst	input	3 bit instruction bus
src1	input	4-bit bus to specify src1 register
src2	input	4-bit bus to specify src2 register
dest	input	4-bit bus to specify dest register
data	input	32-bit bus for data input during LOAD instruction
flush	input	assert high when a branch occurs
cmd	output	instruction to be passed to decoder block, it is com

3.6 流水结构的微控器（Cont.）

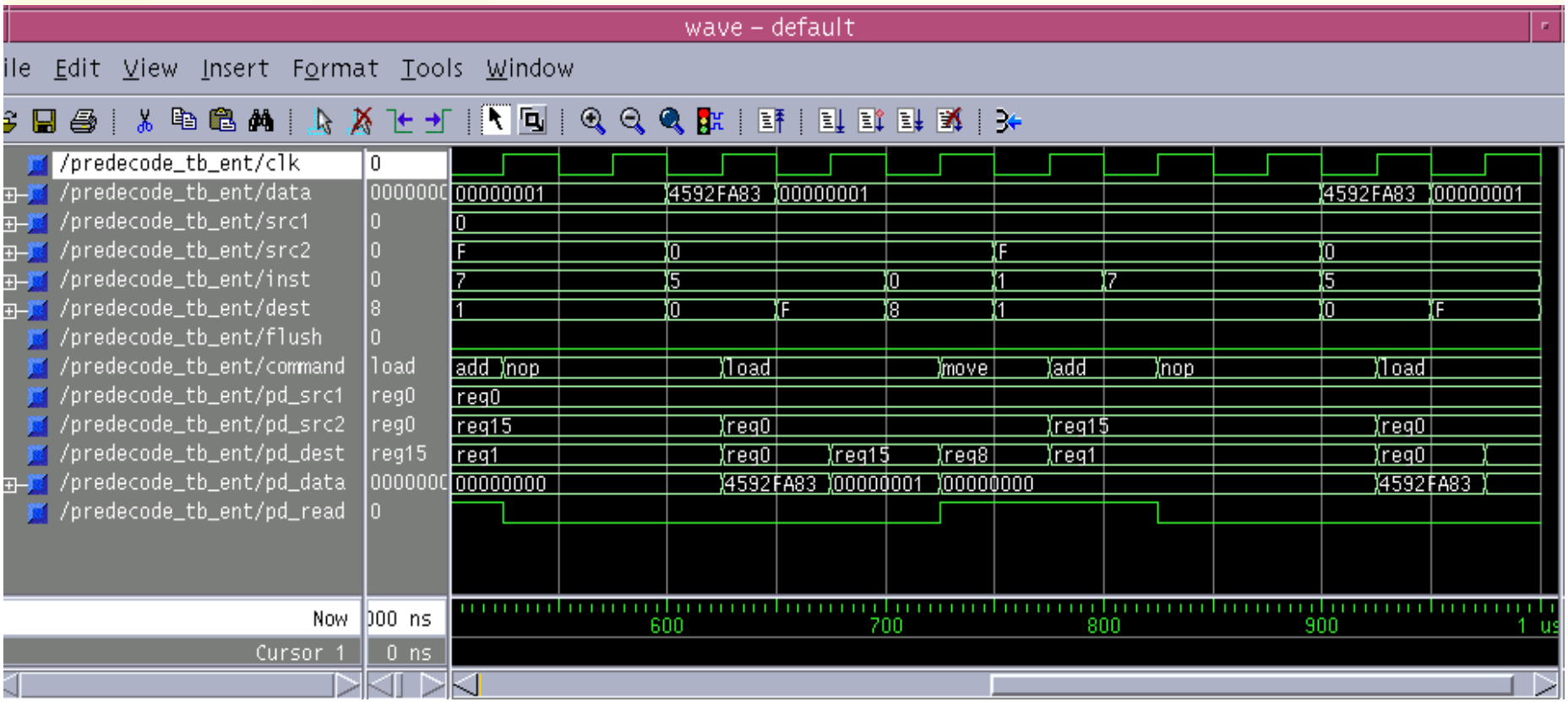
Signals decription of PREDECODE:

Signal Name	I/O	Description
pd_src1	output	indication of src1, reg_type
pd_src2	output	indication of src1, reg_type
pd_dest	output	indication of dest, reg_type
pd_data	output	32-bit bus for passing of data to DECODE block
pd_read	output	single bit and asserted high to indicate to register file to read its internal register pd_src1, pd_src2

3.6 流水结构的微控器（Cont.）

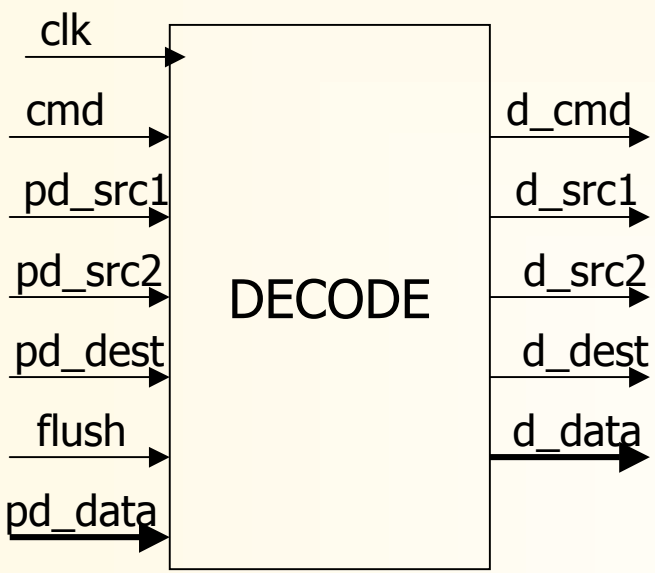
RTL code of PREDECODE and simulation results:

predecode.vhd



3.6 流水结构的微控器（Cont.）

DECODE Block:



3.6 流水结构的微控器（Cont.）

Signals decription of DECODE:

Signal Name	I/O	Description
clk	input	clock input
cmd	input	command_type
pd_src1	input	reg_type
pd_src2	input	reg_type
pd_dest	input	reg_type
pd_data	input	32-bit bus used to pass data between PREDECODE block and DECODE block
pd_flush	input	When asserted high, DECODE block must be in flush mode

3.6 流水结构的微控器（Cont.）

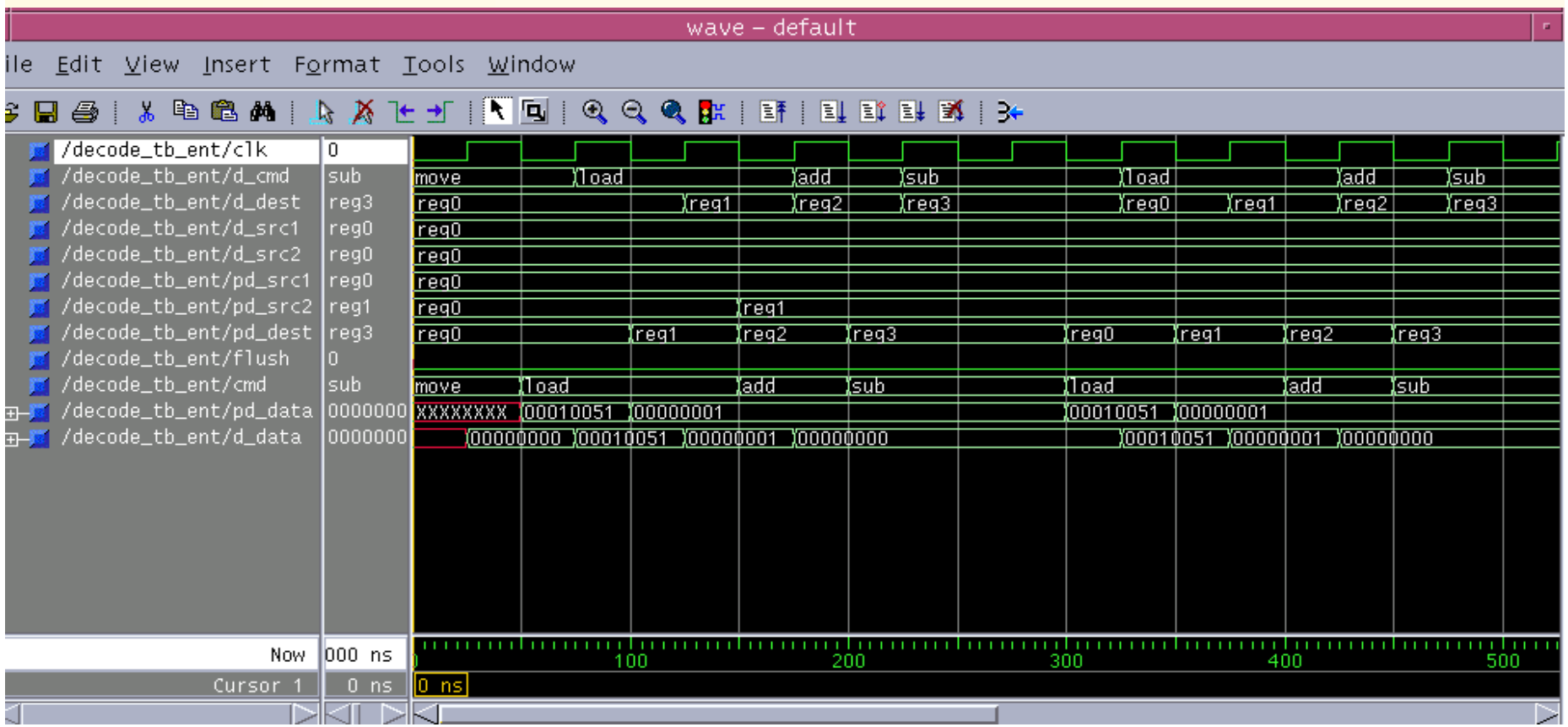
Signals decription of DECODE:

Signal Name	I/O	Description
d_src1	output	passes pd_src1 to EXECUTE block
d_src2	output	passes pd_src2 to EXECUTE block
d_dest	output	passes pd_dest to EXECUTE block
d_data	output	passes pd_data to EXECUTE block during a LOAD instruction
d_cmd	output	passes instruction to EXECUTE block

3.6 流水结构的微控器（Cont.）

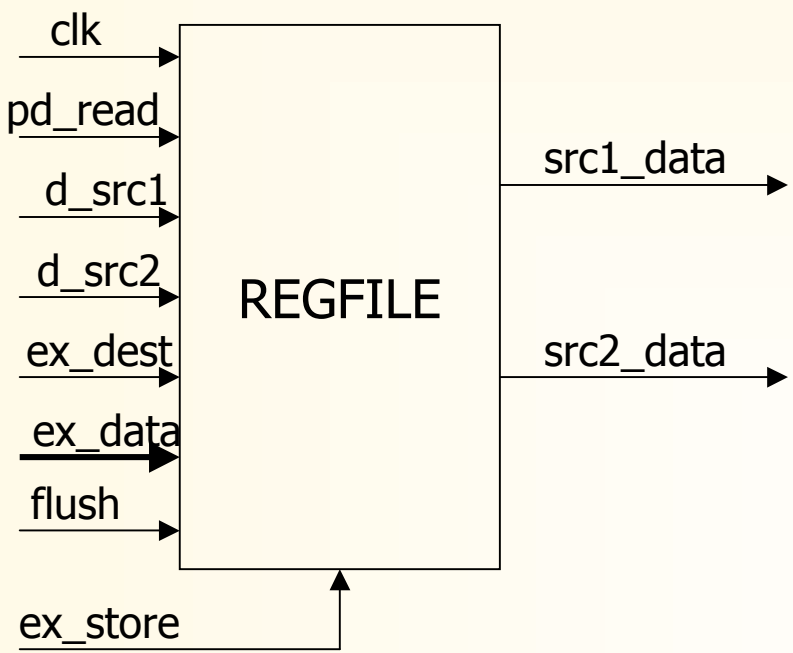
RTL code of DECODE and simulation results:

Decode.vhd



3.6 流水结构的微控器（Cont.）

REGFILE Block:



3.6 流水结构的微控器（Cont.）

Signals decription of REGFILE:

Signal Name	I/O	Description
clk	input	clock signal
flush	input	when asserted to ‘1’, output is 0
pd_read	input	when asserted to ‘1’, the content of registers labed pd_src1 and pd_src2, passed to src1_data and src2_data
pd_src1	input	the contents of register labeled pd_src1 are to be passed to src1_data
pd_src2	input	

3.6 流水结构的微控器（Cont.）

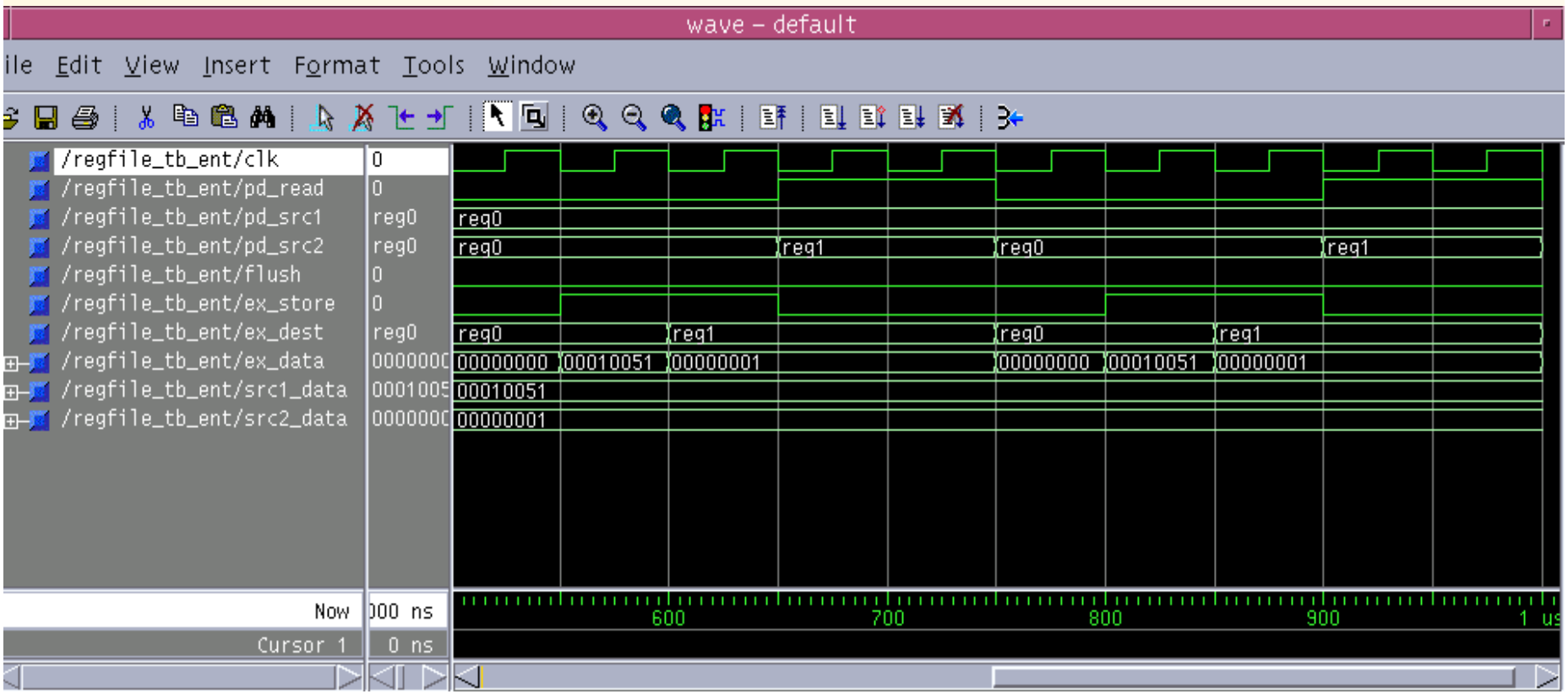
Signals decription of REGFILE:

Signal Name	I/O	Description
ex_store	input	when asserted to '1', ex_data is to be stored in register labeled ex_dest
ex_data	input	transfer data from EXECUTE block to REGFILE block
ex_dest	input	destination of ex_data should be store in REGFILE block
src1_data	output	output of contents of src1
src2_data	output	output of contents of src1

3.6 流水结构的微控器（Cont.）

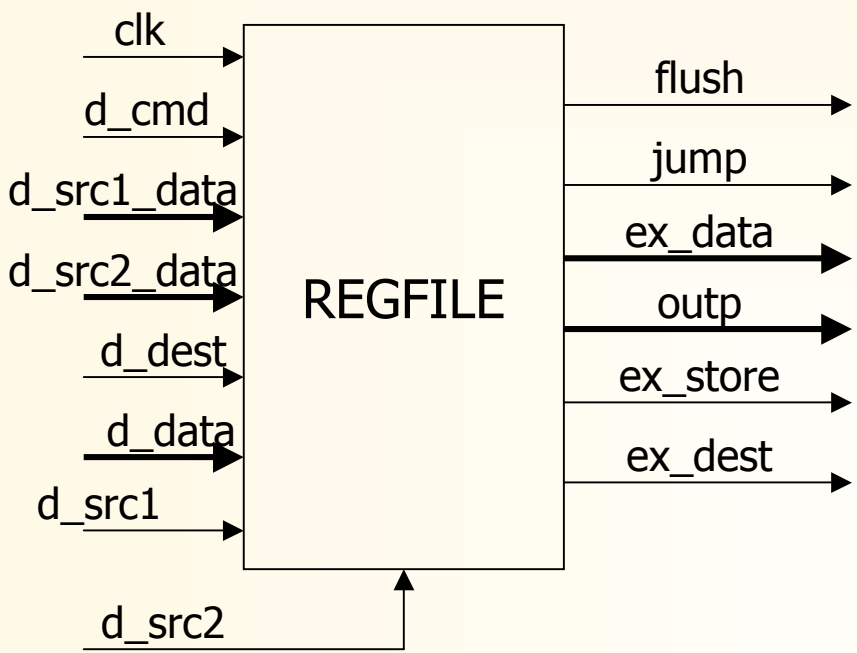
RTL code of REGFILE and simulation results:

Regfile.vhd



3.6 流水结构的微控器（Cont.）

EXECUTE Block:



3.6 流水结构的微控器（Cont.）

Signals decription of EXECUTE:

Signal Name	I/O	Description
d_cmd	input	command_type, from decode block
d_src1_data	input	32 bits input from regfile block
d_src2_data	input	same as above
d_dest	input	register_type, from decode block
d_src1	input	register_type, from decoder block

3.6 流水结构的微控器（Cont.）

Signals decription of EXECUTE:

Signal Name	I/O	Description
d_src2	input	register_type, from decode block
d_data	input	32 bits input from decode block, only valid during LOAD instruction
clk	input	clock signal
flush	output	asserted to ‘1’ when a branch occurs. and other block flush all existing instructions
ex_data	output	32 bits results of arithmetic operation, to regfile block

3.6 流水结构的微控器（Cont.）

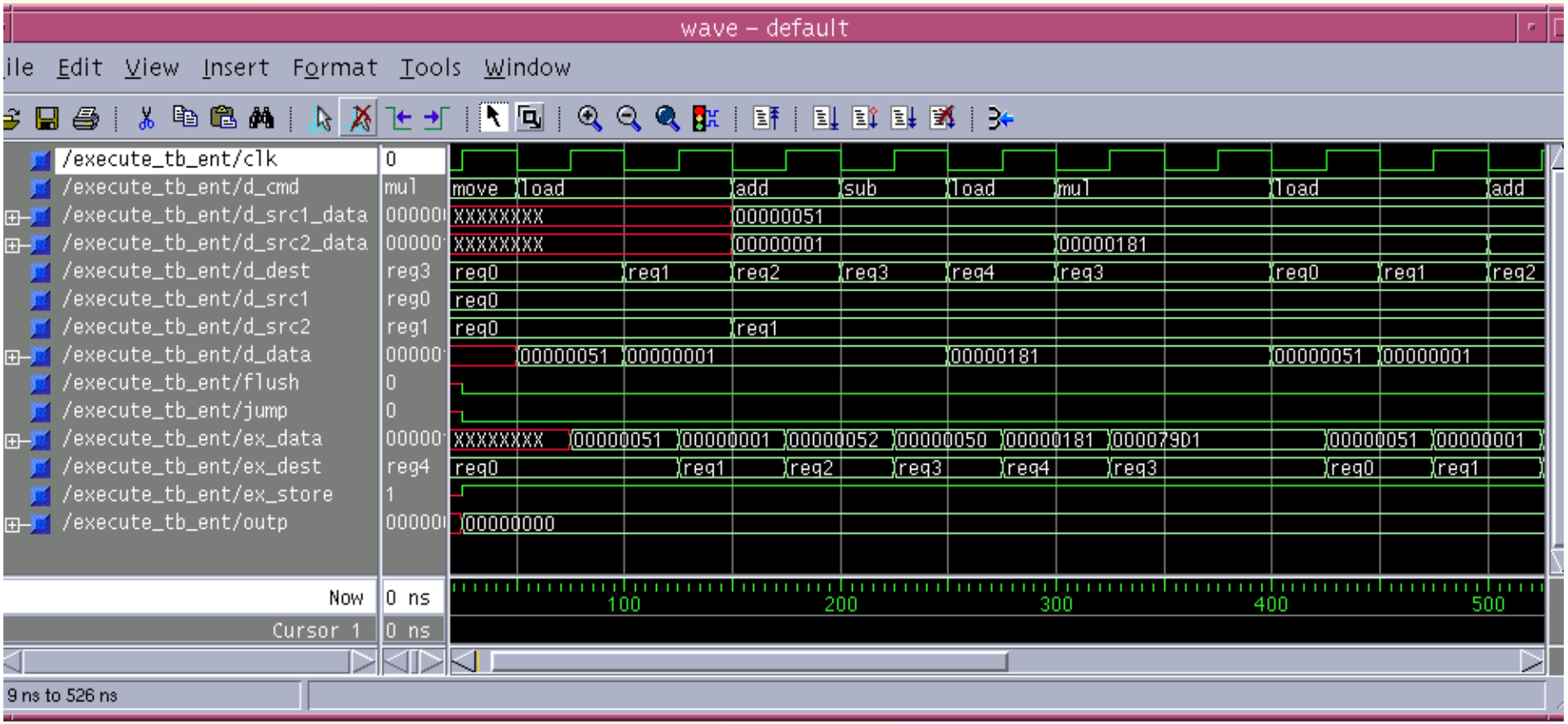
Signals decription of EXECUTE:

Signal Name	I/O	Description
ex_dest	output	register_type, it indicates register address in regfile to be used to store ex_data
jump	output	asserted to ‘1’ when a branch occurs.
outp	output	output of excute block, during a READ instruction, the content of register designated by <destination> are driven on this output bus

3.6 流水结构的微控器（Cont.）

RTL code of EXECUTE and simulation results:

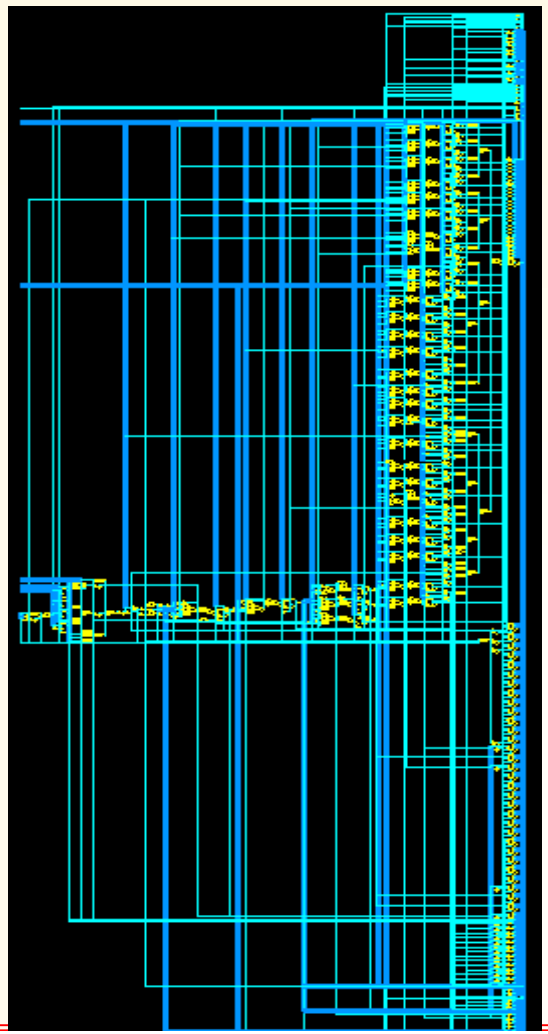
Execute.vhd

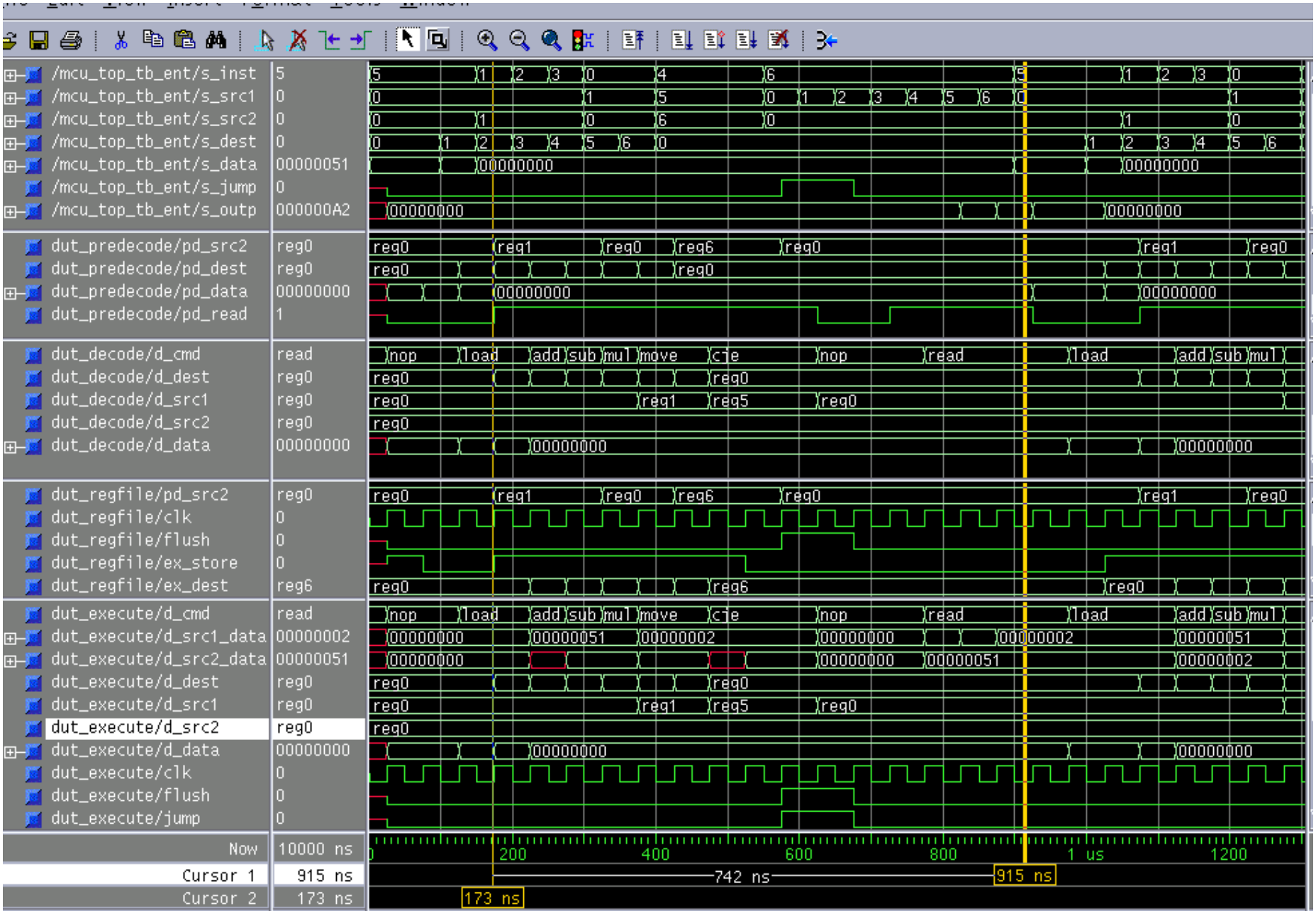


3.6 流水结构的微控器（Cont.）

Synthesys result

of EXECUTE:




/mcu_top_tb_ent/s_inst
/mcu_top_tb_ent/s_src1
/mcu_top_tb_ent/s_src2
/mcu_top_tb_ent/s_dest
/mcu_top_tb_ent/s_data
/mcu_top_tb_ent/s_jump
/mcu_top_tb_ent/s_outp
dut_predecode/pd_src2
dut_predecode/pd_dest
dut_predecode/pd_data
dut_predecode/pd_read
dut_decode/d_cmd
dut_decode/d_dest
dut_decode/d_src1
dut_decode/d_src2
dut_decode/d_data
dut_regfile/pd_src2
dut_regfile/clk
dut_regfile/flush
dut_regfile/ex_store
dut_regfile/ex_dest
dut_execute/d_cmd
dut_execute/d_src1_data
dut_execute/d_src2_data
dut_execute/d_dest
dut_execute/d_src1
dut_execute/d_src2
dut_execute/d_data
dut_execute/clk
dut_execute/flush
dut_execute/jump
Now
10000 ns
Cursor 1
915 ns
Cursor 2
173 ns
742 ns
200
400
600
800
1 us
1200

3.6 ALU

1. 1 bit half adder

X(n)	y(n)	F(n)
0	0	0
0	1	1
1	0	1
1	1	0

$$F(n) = x(n)\overline{y}(n) + \overline{x}(n)y(n) = x(n) \text{ xor } y(n)$$

3.6 ALU(Cont.)

2. 1 bit full adder

真值表:

X(n)	y(n)	c(n-1)	F(n)	c(n)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3.6 ALU(Cont.)

2. 1 bit full adder

逻辑表达式：

$$F(n) = x(n)*\overline{y(n)}*\overline{c(n-1)} + \overline{x(n)}*y(n)*\overline{c(n-1)} + \overline{x(n)}*\overline{y(n)}*c(n-1) + x(n)*y(n)*c(n-1)$$

$$= x(n) \text{ xor } y(n) \text{ xor } c(n-1)$$

$$C(n) = x(n)*y(n) + x(n)*c(n-1) + y(n)*c(n-1)$$

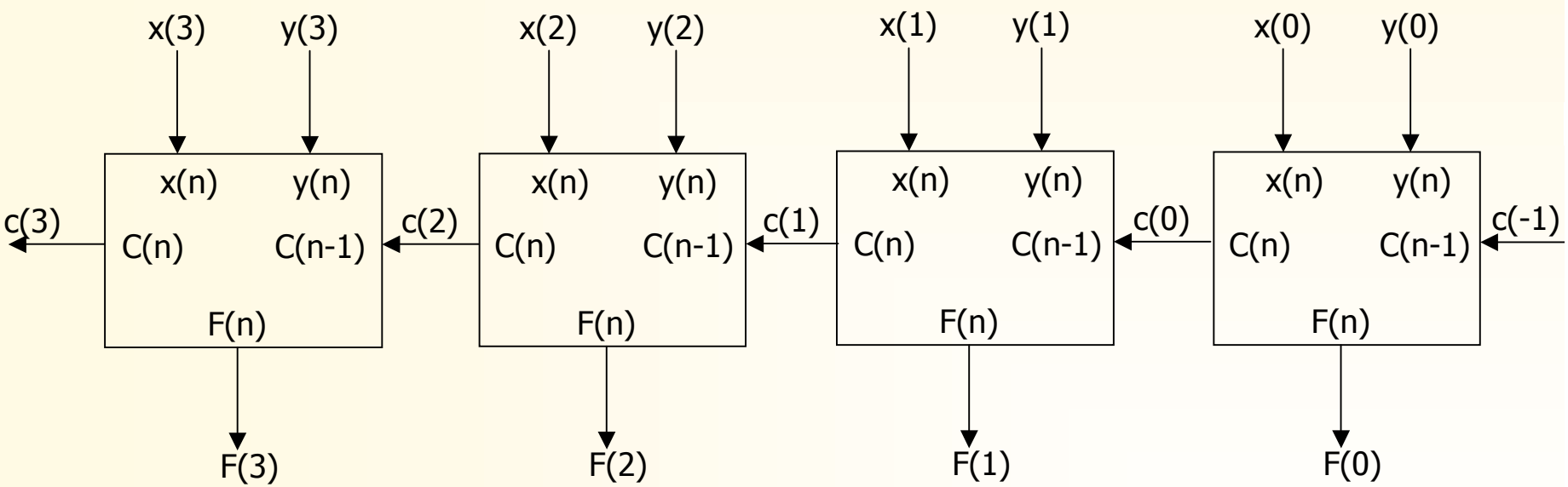
$$= x(n)*y(n) + (x(n) + y(n))*c(n-1)$$

$$\overline{F(n)} = \overline{x(n)}*y(n)*c(n-1) + x(n)*\overline{y(n)}*c(n-1) + x(n)*y(n)*\overline{c(n-1)} + \overline{x(n)}*\overline{y(n)}*\overline{c(n-1)}$$

$$\overline{C(n)} = \overline{x(n)}*\overline{y(n)} + \overline{x(n)}*\overline{c(n-1)} + \overline{y(n)}*\overline{c(n-1)}$$

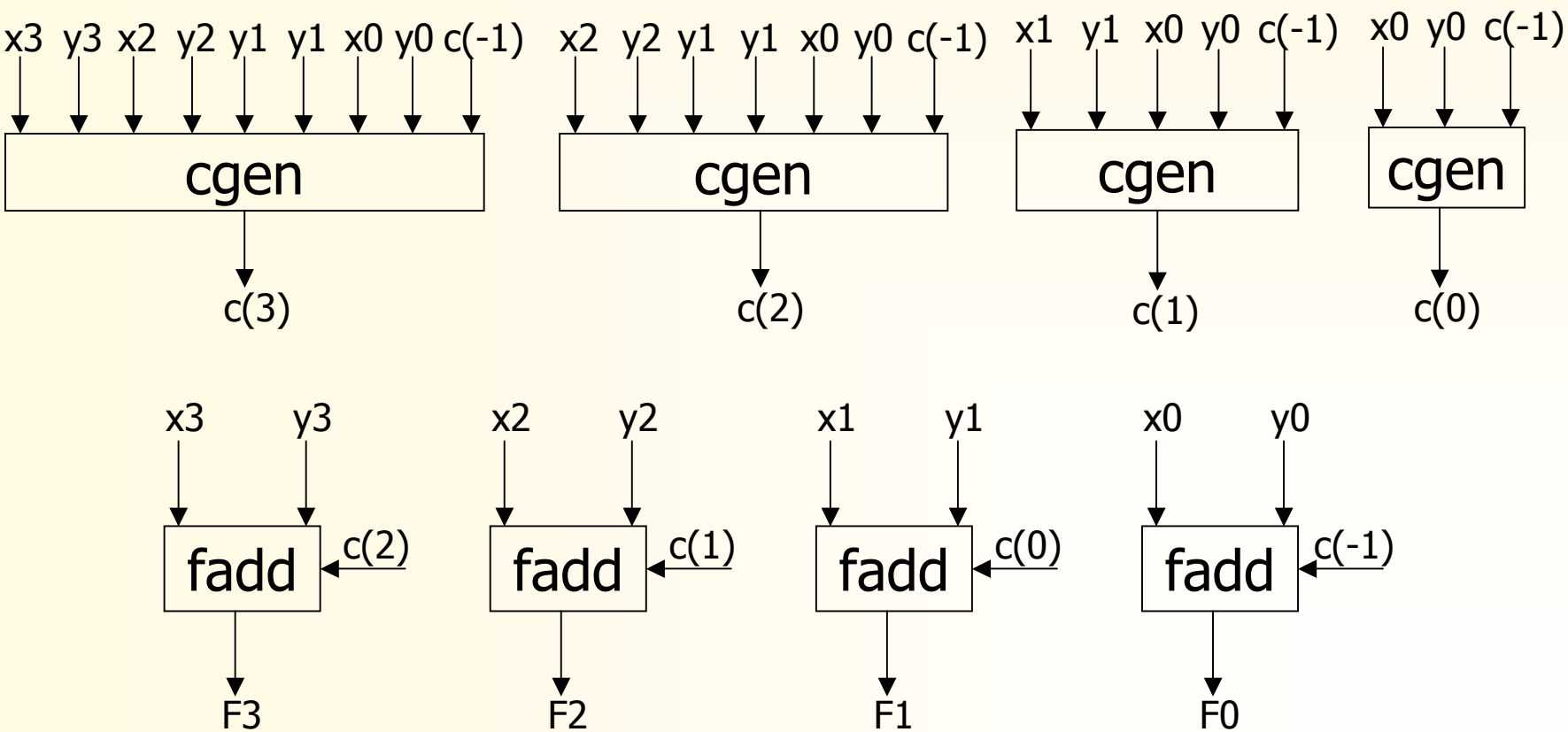
3.6 ALU(Cont.)

3. 4 bits ripple carry adder



3.6 ALU(Cont.)

4. 4 bits carry look-ahead adder



3.6 ALU(Cont.)

4. 4 bits carry look-ahead adder

$p(n) = x(n) + y(n)$　　进位传输

$g(n) = x(n)y(n)$　　进位产生

$c(n) = x(n)y(n) + (x(n) + y(n))c(n-1)$

$=g(n) + p(n)c(n-1)$

$F(n) = x(n) \text{ xor } y(n) \text{ xor } c(n-1)$

$c(0) = g(0) + p(0)c(-1)$

$c(1) = g(1) + p(1)c(0) = g(1) + g(0)p(1) + p(1)p(0)c(-1)$

$c(2) = g(2) + g(1)p(2) + g(0)p(2)p(1) + p(2)p(1)p(0)c(-1)$

$c(3) = g(3) + g(2)p(3) + g(1)p(3)p(2) + g(0)p(3)p(2)p(1) + p(3)p(2)p(1)p(0)c(-1)$

3.6 ALU(Cont.)

4. 4 bits carry look-ahead adder

$$\overline{c(0)} = \overline{p(0)} + \overline{g(0)}*\overline{c(-1)}$$

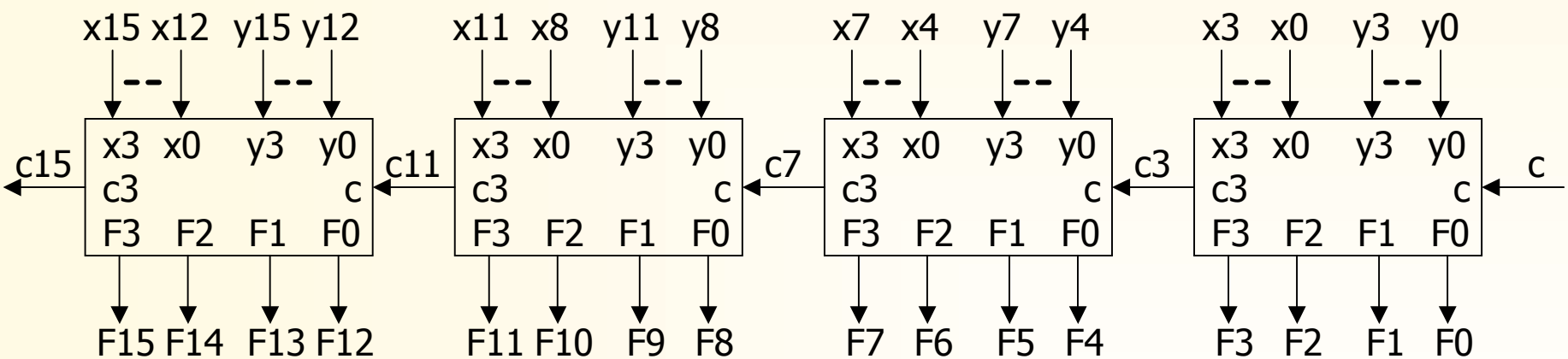
$$\overline{c(1)} = \overline{p(1)} + \overline{p(0)}*\overline{g(1)} + \overline{g(1)}*\overline{g(0)}*\overline{c(-1)}$$

$$\overline{c(2)} = \overline{p(2)} + \overline{p(1)}*\overline{g(2)} + \overline{p(0)}*\overline{g(2)}*\overline{g(1)} + \overline{g(2)}*\overline{g(1)}*\overline{g(0)}*\overline{c(-1)}$$

$$\overline{c(3)} = \overline{p(3)} + \overline{p(2)}\overline{g(3)} + \overline{p(1)}\overline{g(3)}\overline{g(2)} + \overline{p(0)}\overline{g(3)}\overline{g(2)}\overline{g(1)} + \overline{g(3)}\overline{g(2)}\overline{g(1)}\overline{g(0)}\overline{c(-1)}$$

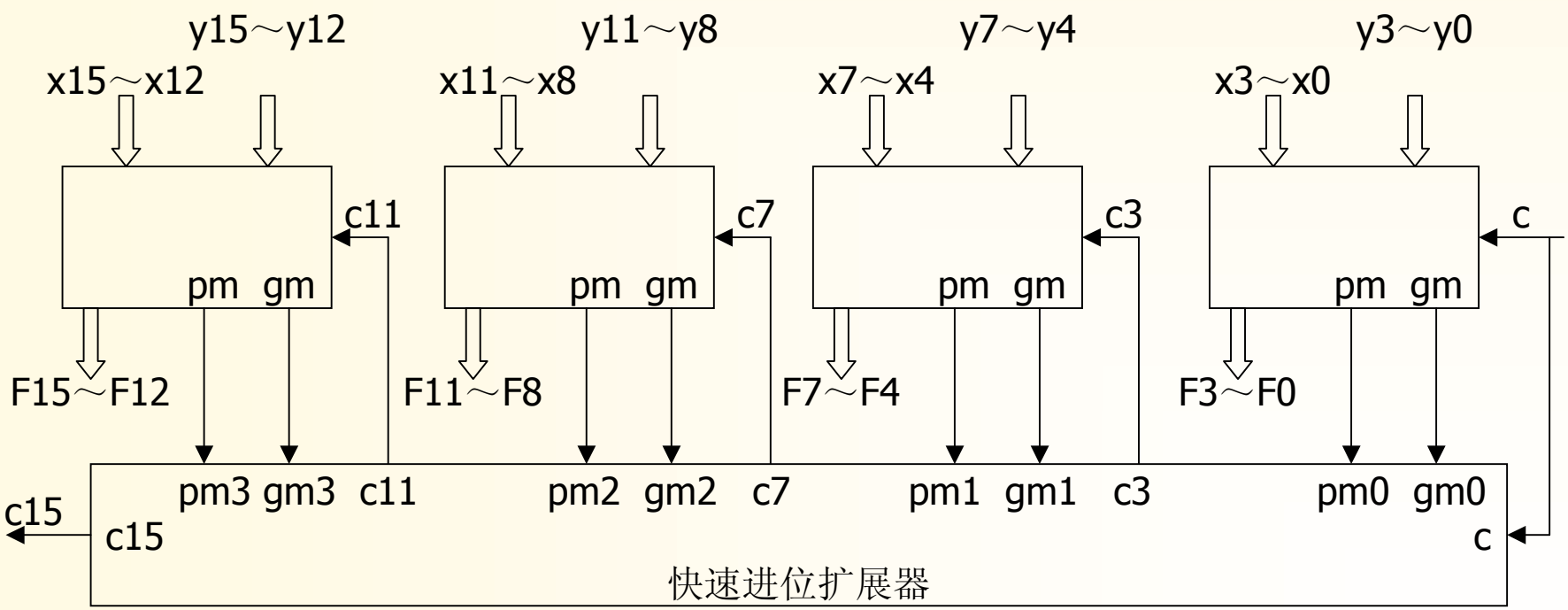
3.6 ALU(Cont.)

5. 16 bits carry look-ahead adder



3.6 ALU(Cont.)

5. 16 bits carry look-ahead adder



3.6 ALU(Cont.)

5. 16 bits carry look-ahead adder

$$c3 = g3 + p3*g2 + p3*p2*g1 + p3*p2*p1*g0 + p3*p2*p1*p0*c = gm0 + pm0*c$$

$$c7 = (g7 + p7*g6 + p7*p6*g5 + p7*p6*p5*g4) + p7*p6*p5*p4(g3 + p3*g2 + p3*p2*g1 + p3*p2*p1*g0 + p3*p2*p1*p0*c)$$
$$= gm1 + pm1(gm0 + pm0*c) = gm1 + pm1*gm0 + pm1*pm0*c$$

$$c11 = (g11 + p11*g10 + p11*p10*g9 + p11*p10*p9*g8) + p11*p10*p9*p8((g7 + p7*g6 + p7*p6*g5 + p7*p6*p5*g4) + p7*p6*p5*p4(g3 + p3*g2 + p3*p2*g1 + p3*p2*p1*g0 + p3*p2*p1*p0*c))$$
$$= gm2 + pm2(gm1 + pm1(gm0 + pm0*c))$$
$$= gm2 + pm2*gm1 + pm2*pm1*gm0 + pm2*pm1*pm0*c$$

$$c15 = gm3 + pm3*gm2 + pm3*pm2*gm1 + pm3*pm2*pm1*gm0 + pm3*pm2*pm1*pm0*c$$

3.6 ALU(Cont.)

5. 16 bits carry look-ahead adder

gm0 = g3 + p3*g2 + p3*p2*g1 + p3*p2*p1*g0

gm1 = g7 + p7g6 + p7p6g5 + p7p6p5g4

gm2 = g11 + p11*g10 + p11*p10*g9 + p11*p10*p9*g8

gm3 = g15 + p15*g14 + p15*p14*g13 + p15*p14*p13*g12

pm0 = p3*p2*p1*p0

pm1 = p7*p6*p5*p4

pm2 = p11*p10*p9*p8

pm3 = p15*p14*p13*p12

3.6 ALU(Cont.)

6. 4 bits ALU

$$cp = \overline{g0} * \overline{g1} * \overline{g2} * \overline{g3}$$

$$cg = \overline{p3} + \overline{g3} * \overline{p2} + \overline{g3} * \overline{g2} * \overline{p1} + \overline{g3} * \overline{g2} * \overline{g1} * \overline{p0}$$

$$cr = \overline{\overline{g3} * \overline{g2} * \overline{g1} * \overline{g0} * \overline{cin}}$$

$$c0 = \overline{\overline{g0} * \overline{cin} + \overline{p0} * \overline{s1}}$$

$$c1 = \overline{\overline{g1} * \overline{g0} * \overline{cin} + \overline{g1} * \overline{p0} * \overline{s1} + \overline{p1} * \overline{s1}}$$

$$c2 = \overline{\overline{g2} * \overline{g1} * \overline{g0} * \overline{cin} + \overline{g2} * \overline{g1} * \overline{p0} * \overline{s1} + \overline{g2} * \overline{p1} * \overline{s1} + \overline{p2} * \overline{s1}}$$

$$c3 = cg * cr$$

$$= \overline{p3} + \overline{g3} * \overline{p2} + \overline{g3} * \overline{g2} * \overline{p1} + \overline{g3} * \overline{g2} * \overline{g1} * \overline{p0} + \overline{g3} * \overline{g2} * \overline{g1} * \overline{g0} * \overline{cin}$$

3.6 ALU(Cont.)

6. 4 bits ALU

$h0 = \overline{s2} * \overline{p0} * \overline{g0}$

$h1 = \overline{s2} * \overline{p1} * \overline{g1}$

$h2 = \overline{s2} * \overline{p2} * \overline{g2}$

$h3 = \overline{s2} * \overline{p3} * \overline{g3}$

F0 =cin xor h0

F1 =c1 xor h1

F2 =c2 xor h2

F3 =c3 xor h3

s1	s2	功能
0	0	加
0	1	未用
1	0	$\overline{Xn \text{ xor } Yn}$
1	1	Xn * Yn