



LUND
UNIVERSITY

EITF35: Introduction to Structured VLSI Design

Part 2.1.2: VHDL-2

Liang Liu
liang.liu@eit.lth.se



Outline

- VHDL Objects
- VHDL Data Types
- Operators in VHDL
- Optimizing VHDL Code
 - Operator sharing



VHDL Objects

□ There are four types of objects in VHDL

- Constants
- Signals
- Variables
- Files

□ Data objects must be *declared before* they are used.

□ Signals

- Can be considered as *wires* in a schematic.
- Can have current value and previous values (FFs).

□ Variables and Constants

- Used to model the *behavior* of a circuit.
- Used in processes, procedures and functions.



VHDL Objects

□ Constant

- Improve the readability of the code
- Allow for easy updating

```
CONSTANT <constant_name> : <type_name> := <value>;  
  
CONSTANT PI : REAL := 3.14;  
CONSTANT WIDTH : INTEGER := 8;
```



VHDL Objects

□ Variables

- Used **only** in processes and subprograms (functions and procedures)
- All variable assignments take place immediately (Contains no “timing info”)
- ***No direct hardware counterpart***

```
VARIABLE <variable_name> : <type_name> := [<value>];  
VARIABLE opcode : BIT_VECTOR (3 DOWNTO 0) := "0000";  
VARIABLE freq : INTEGER;
```

Don't unless you know what exactly the resulting hardware will be!



VHDL Objects

□ Signals

- Used for communication between components
- Ports in entity declaration are considered as signals
- Can be seen as *real, physical wires*

```
SIGNAL <signal_name> : <type_name>;  
  
SIGNAL enable : BIT;  
SIGNAL output : bit_vector(3 downto 0);  
bit_vector <= "0111";
```

Recommended for beginners

Suggestion: Do not assign initial value to signals



VHDL Objects: Scoping Rules

□ VHDL limits the visibility of the objects, depending on *where they are declared*

□ The scope of the object is as follows

- Objects declared in a package are global to all entities that use that package
- Objects declared in an entity are global to all architectures that use that entity
- Objects declared in an architecture are available to all statements in that architecture
- Objects declared in a process are available to only that process

□ Scoping rules apply to constants, variables, signals and files



Outline

- VHDL Objects
- **VHDL Data Types**
- Operators in VHDL
- Optimizing VHDL Code
 - Operator sharing



Data type

□ VHDL is a **STRONGLY-TYPED** language

- An object can only be *assigned* with a value of its type
- Only the *operations* defined with the data type can be performed on the object
- Type conversion

□ Standard VHDL

□ IEEE std_logic_1164 package

□ IEEE numeric_std package

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



Data types in standard VHDL

□ integer:

- Range: -2^{31} to $2^{31}-1$

□ boolean: (false, true)

□ bit: ('0', '1') - **Not capable enough**

□ bit_vector: a one-dimensional array of bit



IEEE std_logic_1164 package

- What's wrong with bit?
- New data type: `std_logic`, `std_logic_vector`
- `std_logic`, 9 values: ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')
 - '0', '1': forcing logic 0 and forcing logic 1
 - 'Z': high-impedance, as in a tri-state buffer.
 - 'L', 'H': weak logic 0 and weak logic 1, as in wired-logic
 - 'X', 'W': “unknown” and “weak unknown”
 - 'U': for uninitialized
 - '-': don't-care.



std_logic_vector

□ std_logic_vector

- An array of elements with std_logic data type
- Imply a **bus**

```
signal a: std_logic_vector(7 downto 0);
```

- Another form (**less desired**)

```
signal a: std_logic_vector(0 to 7);
```

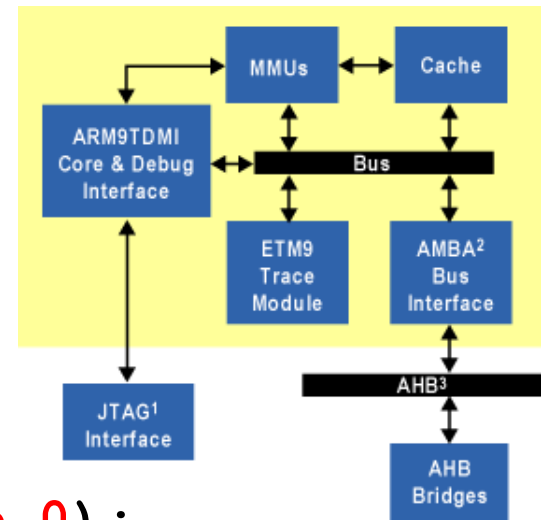
□ Need to invoke package to use the data type:

```
library ieee;
```

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

□ Recommended Data Type:

- **Integer**: to model generics or **constants**
- **std_logic**: for one bit signals
- **std_logic_vector**: A bus of std_logic, e.g., counters, addresses, etc.



IEEE numeric_std

□ How to infer arithmetic operators?

□ In standard VHDL:

```
signal a, b, sum: integer;  
.  
.  
.  
sum <= a + b;
```

□ What's the limitation of integer data type?

- Difficult to realize in hardware, because the *number of bits (word-length)* is not indicated



IEEE numeric_std

- IEEE `numeric_std` package: define integer as an array of elements of `std_logic`
- Two new data types: `unsigned`, `signed`
 - These are declared in a similar method to 'std_logic_vector'
 - Can be used to declare signals, variables, even ports in an entity
- The array interpreted as an `unsigned` or `signed` binary number

```
signal x, y: signed(15 downto 0);
```

- Need to invoke package

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



Type Conversion

data type of a	to data type	conversion function / type casting
unsigned, signed	std_logic_vector	std_logic_vector(a)
signed, std_logic_vector	unsigned	unsigned(a)
unsigned, std_logic_vector	signed	signed(a)
unsigned, signed	integer	to_integer(a)
integer	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)



Type Conversion: Example

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;  
. . .  
signal s1, s2, s3, s4, s5, s6: std_logic_vector(3 downto 0);  
signal u1, u2, u3, u4, u5, u6: unsigned(3 downto 0);  
signal sg: signed(3 downto 0);
```

- Ok

```
u3 <= u2 + u1; --- ok, both operands unsigned
```

```
u4 <= u2 + 1; --- ok, operands unsigned and natural
```

-Wrong

```
u5 <= sg; -- type mismatch
```

```
u6 <= 5; -- type mismatch
```

- Fix

```
u5 <= unsigned(sg); -- type casting
```

```
u6 <= to_unsigned(5,4); -- conversion function
```



Outline

- VHDL Objects
- VHDL Data Types
- **Operators in VHDL**
- Optimizing VHDL Code
 - Operator sharing



Operators in Standard VHDL

operator	description	data type of operand a	data type of operand b	data type of result
a ** b	exponentiation	integer	integer	integer
abs a	absolute value	integer		integer
not a	negation	boolean, bit, bit_vector		boolean, bit, bit_vector
a * b	multiplication	integer	integer	integer
a / b	division			
a mod b	modulo			
a rem b	remainder			
+ a	identity	integer		integer
- a	negation			
a + b	addition	integer	integer	integer
a - b	subtraction			
a & b	concatenation	1-D array, element	1-D array, element	1-D array



Operators in Standard VHDL (cont'd)

a sl b	shift left logical	bit_vector	integer	bit_vector
a srl b	shift right logical			
a sla b	shift left arithmetic			
a sra b	shift right arithmetic			
a rol b	rotate left			
a ror b	rotate right			
a = b	equal to	any	same as a	boolean
a /= b	not equal to			
a < b	less than	scalar or 1-D array	same as a	boolean
a <= b	less than or equal to			
a > b	greater than			
a >= b	greater than or equal to			
a and b	and	boolean, bit,	same as a	boolean, bit,
a or b	or	bit_vector		bit_vector
a xor b	xor			
a nand b	nand			
a nor b	nor			
a xnor b	xnor			



Operators in Standard VHDL (Precedence)

Precedence	Operators
Highest	** abs not * / mod rem + - (identity and negation) & + - (addition and subtraction) sll srl sla sra rol ror = /= < <= > >=
Lowest	and or nand nor xor xnor



□ Example: $a+b>c$ or $a<d \Rightarrow ((a+b)>c)$ or $(a<d)$

Suggestion: Use parentheses, even when they are not needed.



Overloaded operator

IEEE std_logic_1164 package

- Which standard VHDL operators can be applied to `std_logic` and `std_logic_vector`?
- Overloaded operators in `std_logic_1164` package

overloaded operator	data type of operand a	data type of operand b	data type of result
<code>not a</code>	<code>std_logic_vector</code> <code>std_logic</code>		same as a
<code>a and b</code>			
<code>a or b</code>			
<code>a xor b</code>	<code>std_logic_vector</code>	same as a	same as a
<code>a nand b</code>	<code>std_logic</code>		
<code>a nor b</code>			
<code>a xnor b</code>			



Operators Over an Array Data Type

□ Relational operators for array

- Operands must have the *same element type*
- But their *lengths may differ*
- Two arrays are compared element by element, from the *left most element*
- All following returns true

"011"="011" ,

"011">"010" ,

"011">"01000" ,

"0110">"011"

"0110"="011" returns false

Suggestion: Avoid use different length !



Operators Over an Array Data Type (Cont'd)

□ Concatenation operator (&)

□ Combine segments of elements and smaller arrays to form a larger array.

- Shift the elements of the array to the right by two positions and append two 0's to the front:

```
y <= "00" & a(7 downto 2);
```

- Append the MSB to the front (known as an arithmetic shift):

```
y <= a(7) & a(7) & a(7 downto 2);
```

- *Exercise: rotate the elements to the right by two positions:*

```
y <= a(1 downto 0) & a(7 downto 2);
```



Array Aggregate

□ Aggregate is a VHDL construct to assign a value to an array-typed object

□ Example1: they are the same

```
a <= "10100000";
```

```
a <= (7=>'1', 6=>'0', 0=>'0', 1=>'0', 5=>'1',  
4=>'0', 3=>'0', 2=>'1');
```

```
a <= (7|5=>'1', 6|4|3|2|1|0=>'0');
```

```
a <= (7|5=>'1', others=>'0');
```

□ Example2:

```
a <= "00000000"
```

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



Overloaded Operator

IEEE numeric_std package

overloaded operator	description	data type of operand a	data type of operand b	data type of result
abs a - a	absolute value negation	signed		signed
a * b a / b a mod b a rem b	arithmetic operation	unsigned unsigned, natural signed	unsigned, natural unsigned signed, integer	unsigned unsigned signed
a + b a - b		signed, integer	signed	signed
a = b a /= b a < b a <= b a > b a >= b	relational operation	unsigned unsigned, natural signed signed, integer	unsigned, natural unsigned signed, integer signed	boolean boolean boolean boolean



Overloaded Operator

IEEE numeric_std package: comparison

□ std_logic_vector

```
"011" > "0100" -- true
```

□ unsigned

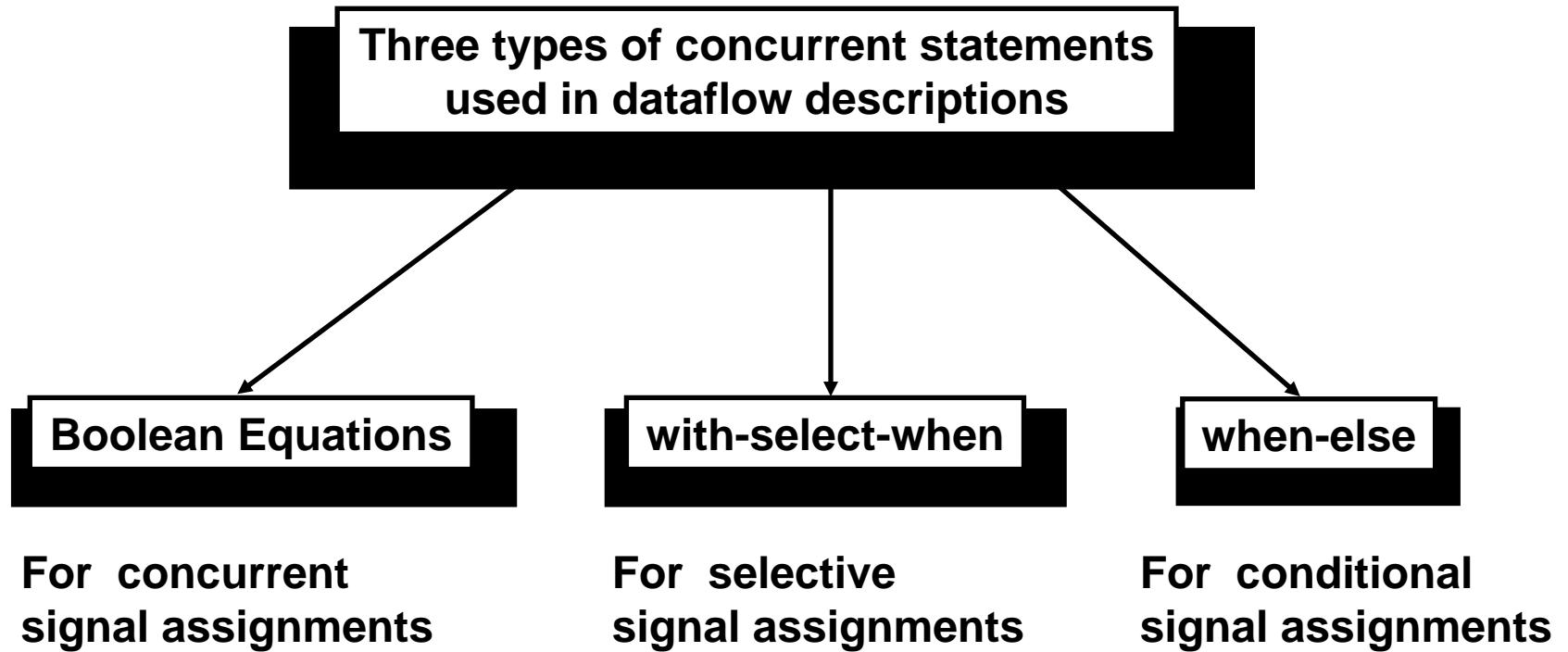
```
"011" > "0100" -- false
```

□ signed

```
"011" > "1000" -- true
```

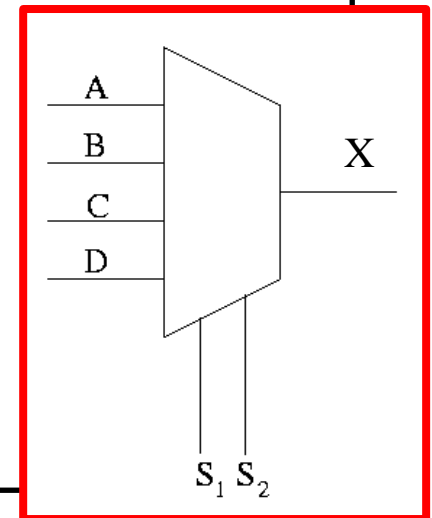


Concurrent Statements



Concurrent Statements: When-else

```
entity mux is port(a,b,c,d: in
std_logic_vector(3 downto 0);
  s: in std_logic_vector(1 downto 0);
  x: out std_logic_vector(3 downto 0));
end mux;
architecture mux_arch of mux is
begin
  x <= a when (s = "00") else
      b when (s = "01") else
      c when (s = "10") else
      d;
end mux_arch;
```



Can not be used in process!!!

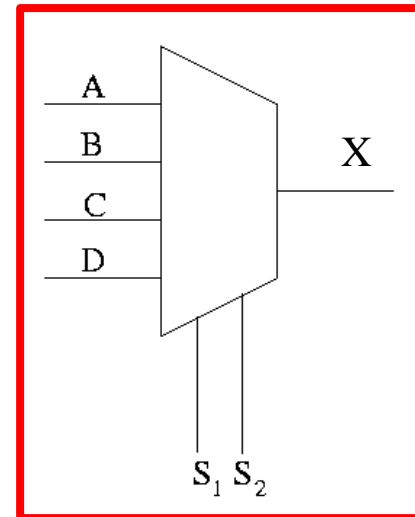


Concurrent Statements: with-select-when

```
entity mux is port(a,b,c,d: in
std_logic_vector(3 downto 0);
  s: in std_logic_vector(1 downto 0);
  x: out std_logic_vector(3 downto 0));
end mux;
```

```
architecture mux_arch of mux is
begin
```

```
with s select
  x<= a when "00",
    b when "01",
    c when "10",
    d when others;
end mux_arch;
```



Outline

- VHDL Objects
- VHDL Data Types
- Operators in VHDL
- **Optimizing VHDL Code**
 - Operator sharing



Optimize VHDL Code: Operator Sharing

- Circuit complexity of VHDL operators varies
- Arithmetic operators
 - Large implementation
 - *Limited optimization* by synthesis software
- Manual optimization is forced by operator sharing in RTL

Suggestion: Optimize As Early As Possible!



Operator Sharing

□ Resource Sharing

- Identify the resources that can be used by different operations

□ Multiplexing network are mutually exclusively:

- **Only one result** is routed to output
- **Only one operation** is active at a particular time

```
with select_expression select  
    sig_name <= value_expr_1 when choice_1,  
    value_expr_2 when choice_2,  
    value_expr_3 when choice_3,  
    . . .  
    value_expr_n when choice_n;
```



Example I

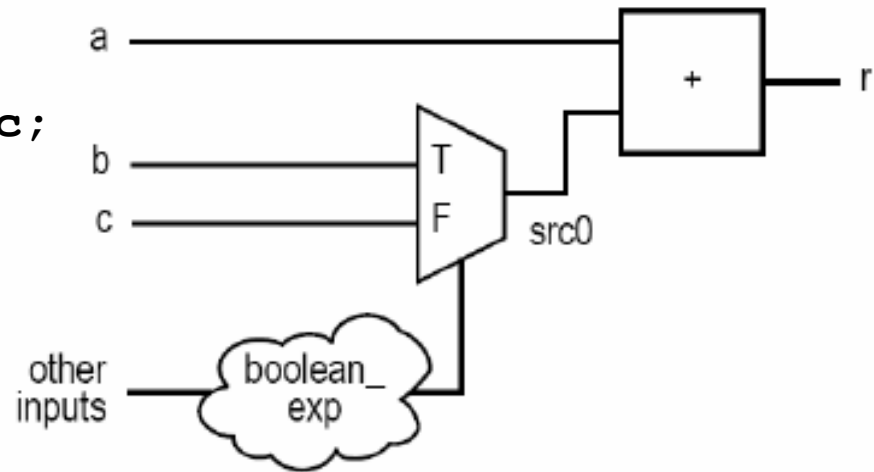
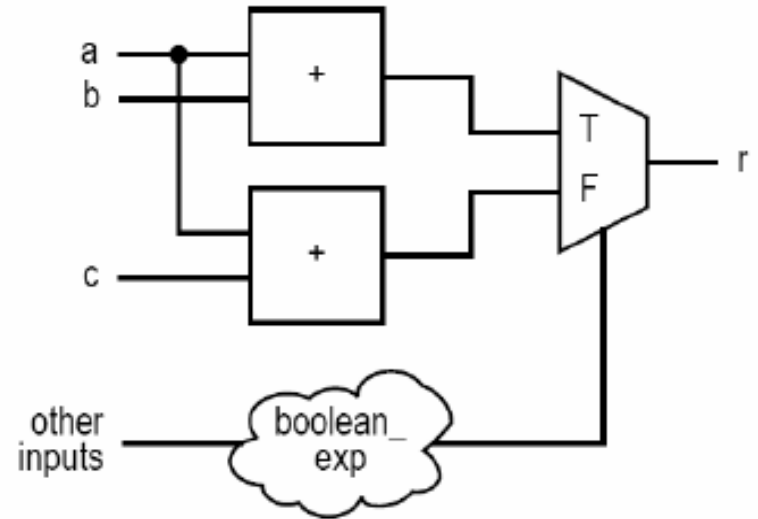
Original code:

```
r <= a+b when boolean_exp else a+c;
```

write VHDL code that reduces the number of adders to "1!"

Revised code:

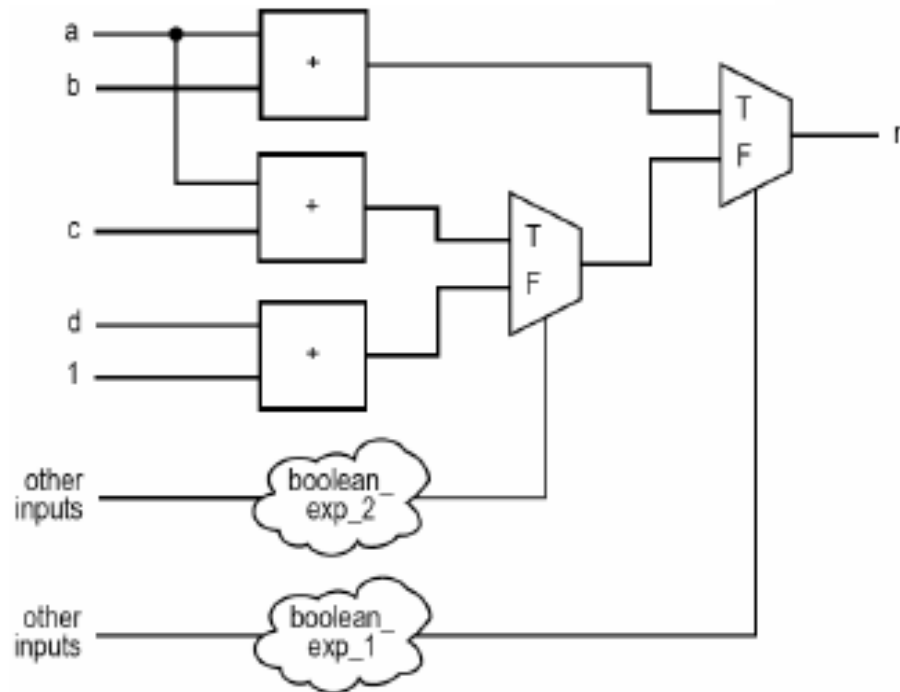
```
src0 <= b when boolean_exp else c;  
r <= a + src0;
```



Example II

Original code:

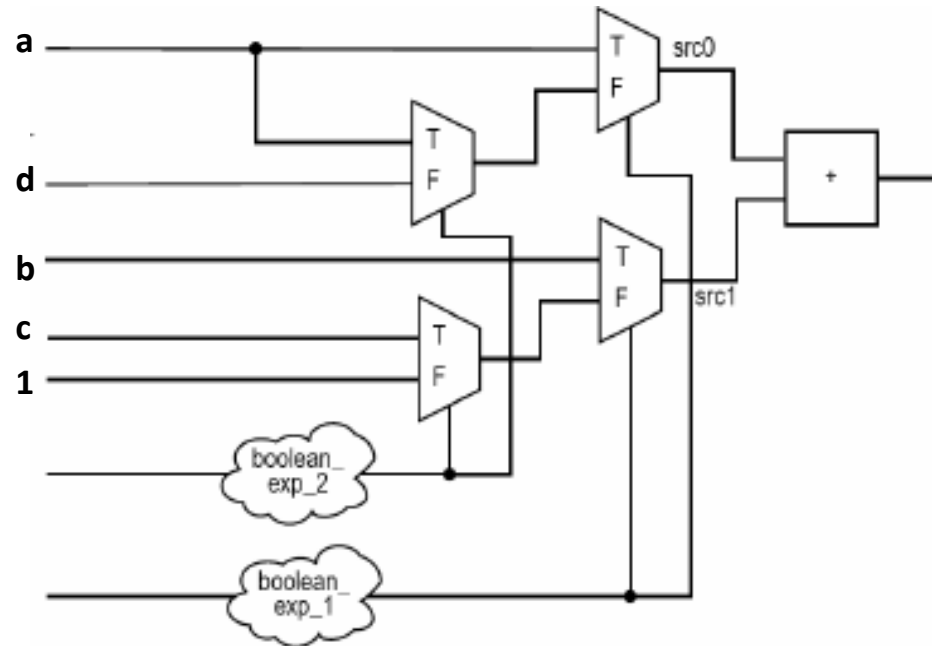
```
process (a,b,c,d,...)
begin
  if boolean_exp_1 then r <= a+b;
  elsif boolean_exp_2 then r <= a+c;
  else r <= d+1;
end if
end process;
```



Example II

□ Revised code:

```
process (a,b,c,d,...)
begin
  if boolean_exp_1 then
    src0 <= a;
    src1 <= b;
  elsif boolean_exp_2 then
    src0 <= a;
    src1 <= c;
  else
    src0 <= d;
    src1 <= "00000001";
  end if;
end process;
r <= src0 + src1;
```



5 min: Draw the hardware of the revised code



Something to Remember!



Comments "--": may be "more important"

```
-----  
-- Design : CARRIER SENSE  
-- File Name : CARRIER_SENSE.vhdl  
-- Purpose : Model of CARRIER SENSE process in PCS (IEEE Std 802.3)  
-- Limitation : none  
-- Errors : none known  
-- Include Files: none  
-- Author : Liang Liu, liang.liu@eit.lth.se, Lund University  
-- Simulator : ModelSim 6.5  
-----
```

-- Revision List:

```
-----+  
-- | Version | Author          | Date       | Changes                |  
-----+  
-- | 1.0      | Liang Liu       | 2001/08/03 | original created       |  
-- | 1.1      | Liang Liu       | 2002/01/04 | disable TX_EN to CRS   |  
-- |          |                  |              | in repeater mode      |  
-----+
```

***Make your code readable!!!
Maintenance and re-usability***



DO NOT introduce uncertainty

□ 'X' and 'U' are legal VHDL, but the synthesized circuit won't behave like you expect.

```
architecture behv of ALU
is
begin
process (A,B,Sel) begin
case Sel is
when "00" =>
Res <= A + B;
when "01" =>
Res <= A + (not B) + 1;
when "1X" =>
Res <= A and B;
when "1U" =>
Res <= A or B;
when others =>
Res <= A;
end case;
end process;
end behv;
```

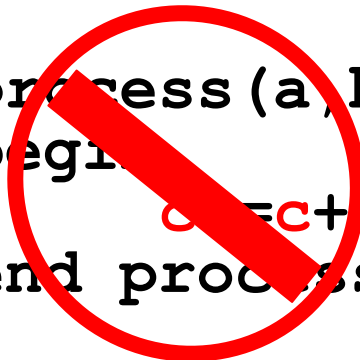
```
architecture behv of
ALU is
begin
process(A,B,Sel) begin
case Sel is
when "00" =>
Res <= A + B;
when "01" =>
Res <= A + (not B) + 1;
when "10" =>
Res <= A and B;
when "11" =>
Res <= A or B;
when others =>
Res <=A;
end case;
end process;
end behv;
```



Feedbacks

- ❑ Don't use a **combinational feedback**
- ❑ You never really need them.

```
process (a, b)  
begin  
    c = a + b;  
end process;
```



Multiple Source

- ❑ Drive every signal from exactly one process or concurrent assignment.

Architecture arc_ad of ad is
begin
 e <= a+b;
 e <= c+d;
end arc_ad;



Boolean v.s. std_logic

❑ Don't assign Boolean to std_logic

```
signal a : std_logic;  
signal b : unsigned(7 downto 0);  
a <= (b = x"7E"); -- BAD: result is Boolean,  
a <= '1' when b = x"7E" else '0'; --OK
```

❑ Don't test std_logic in a Boolean context

```
signal a, b, foo : std_logic;  
if a then -- BAD: A is not Boolean  
foo <= '1';  
end if;  
b <= '0' when a else '1'; -- BAD: a is not Boolean  
if a = '1' then -- OK  
foo <= '1';  
end if;  
b <= '0' when a = '1' else '0'; -- OK
```



Reading advice

RTL Hardware Design Using VHDL: P51-P69, P163-P178

VHDL HandBook

<http://www.csee.umbc.edu/portal/help/VHDL/VHDL-Handbook.pdf>

