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
- **Signals**
  - Can be considered as ***wires*** in a schematic.
  - Can have current value and previous values (***registers***).
- **Variables and Constants**
  - **NO** clear mapping to circuit
  - Can be used to improve coding efficiency

VHDL Objects

Constant

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

```
CONSTANT <constant_name>  :  <type> := <value>;

CONSTANT PI : REAL := 3.14;
CONSTANT WORDLENGTH: INTEGER := 8;
```

VHDL Objects

Variables

- Used **ONLY** in processes and subprograms (functions and procedures)
- All variable assignments take place immediately
- ***NO direct hardware counterpart***

```
VARIABLE <variable_name> : <type_name> := [<value>];

VARIABLE opcode : BIT_VECTOR (3 DOWNTO 0) := "0000";
VARIABLE freq : INTEGER;
```

NO variables!!!

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 or registers***

```
SIGNAL <signal_name> : <type_name>;

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

Do NOT assign initial value to signals

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
- **Packages defining data types**
  - Focus on data types that can be **syntehsized**
  - Standard VHDL
  - IEEE std_logic_1164 package
  - IEEE numeric_std package

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

Data types in standard VHDL

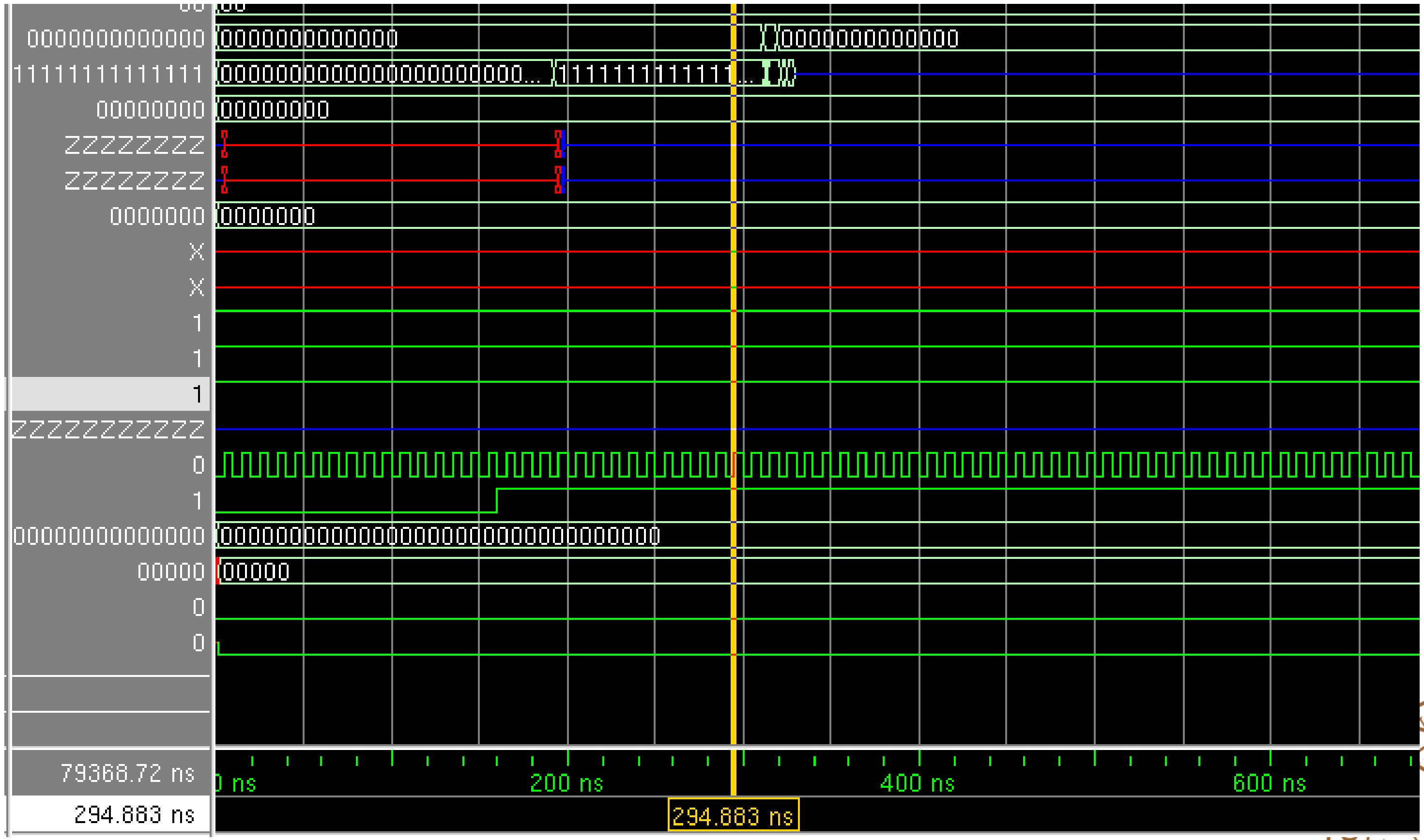
- **integer:**
  - Range: -2^31 to 2^31-1
- **boolean: (false, true)**
- **bit: ('0', '1') - Not capable enough**
- **bit_vector: a one-dimensional array of bit**

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.

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','-')`



std_logic_vector

- **std_logic_vector**
  - An array of elements with std_logic data type

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

- **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 (not signal)**
  - ***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?
  - Word-length

IEEE numeric_std

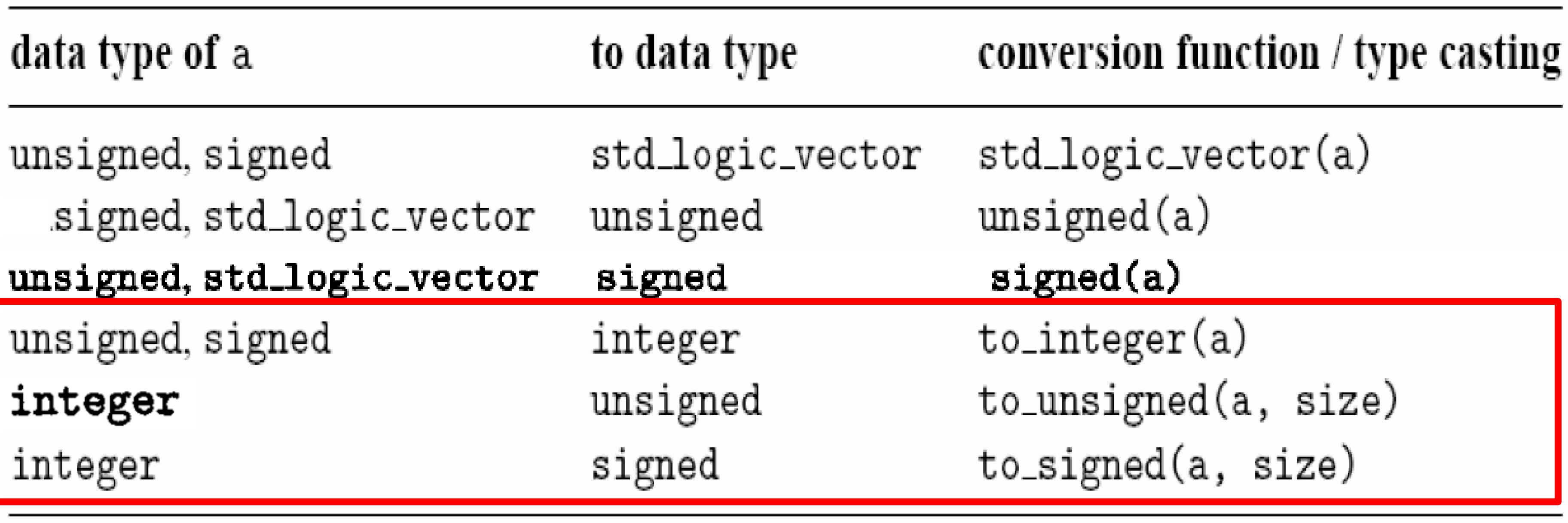
- IEEE **`numeric_std`** package
  - Define integer as an array of elements of **`std_logic`**
- **Two new data types: `unsigned`, `signed`**
  - These are declared in a similar method to ‘`std_logic_vector`’
- **The array interpreted as an `unsigned` or `signed` binary number**

```
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)

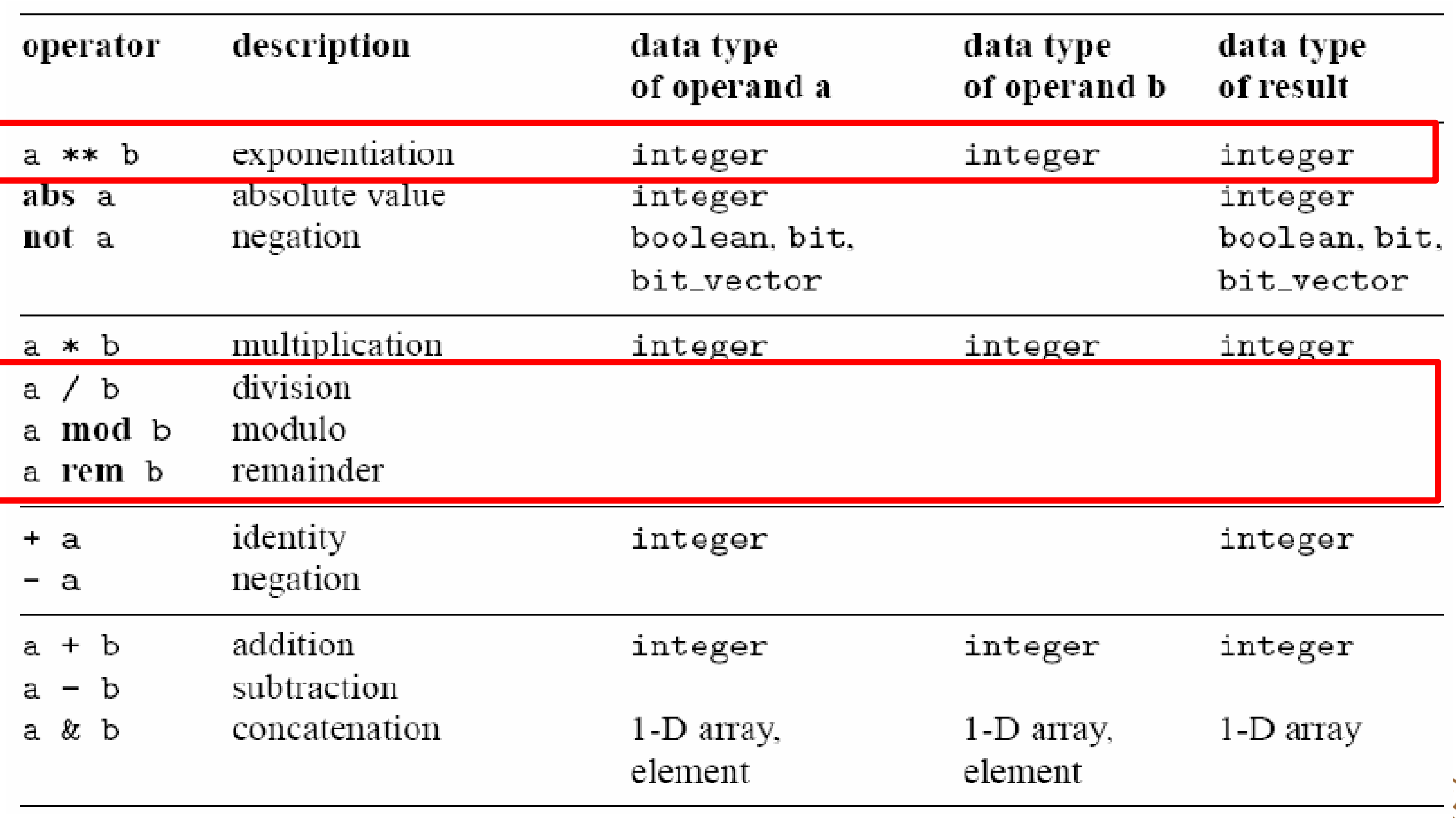
Type Conversion: Example

```
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

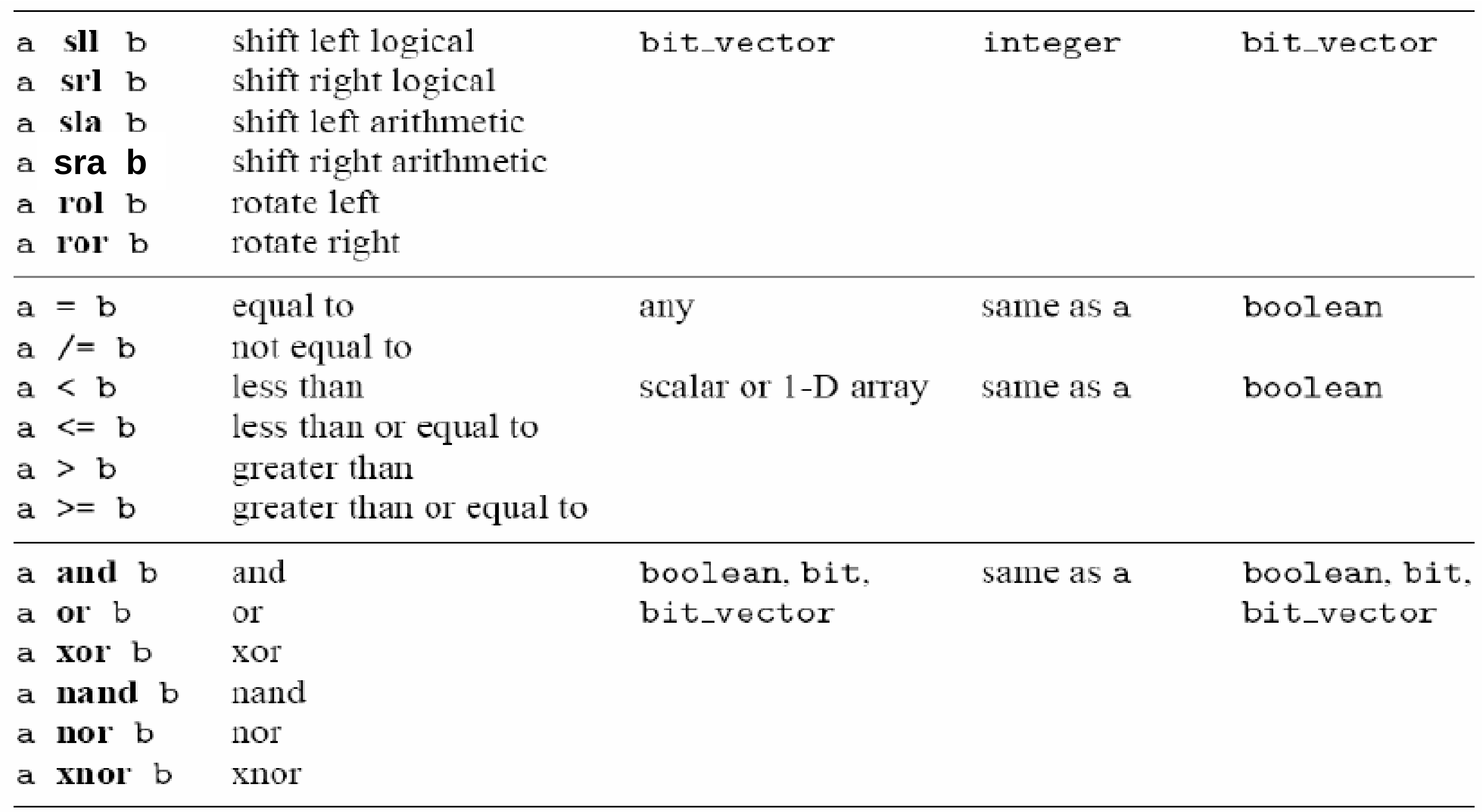
- VHDL Objects
- VHDL Data Types
- **Operators in VHDL**
- Optimizing VHDL Code
  - Operator sharing

Operators in Standard VHDL



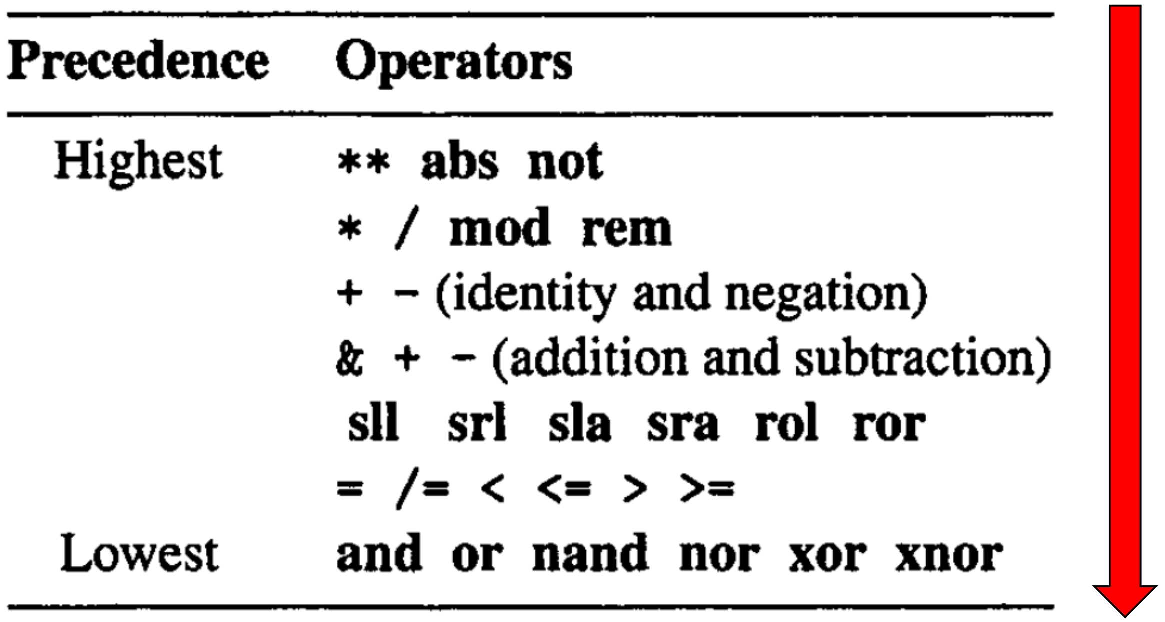
operator	description	data type of operand a	data type of operand b	data type of result
`a ** b`	exponentiation	`integer`	`integer`	`integer`
`abs a`	absolute value	`integer`		`integer`
`not a`	negation	`boolean, bit, bit_vector`		`boolean, bit, bit_vector`
`a * b`	multiplication	`integer`	`integer`	`integer`
`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 **sll** 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, bit_vector	same as a	boolean, bit, bit_vector
a **or** b	or			
a **xor** b	xor			
a **nand** b	nand			
a **nor** b	nor			
a **xnor** b	xnor			

Operators in Standard VHDL (Precedence)



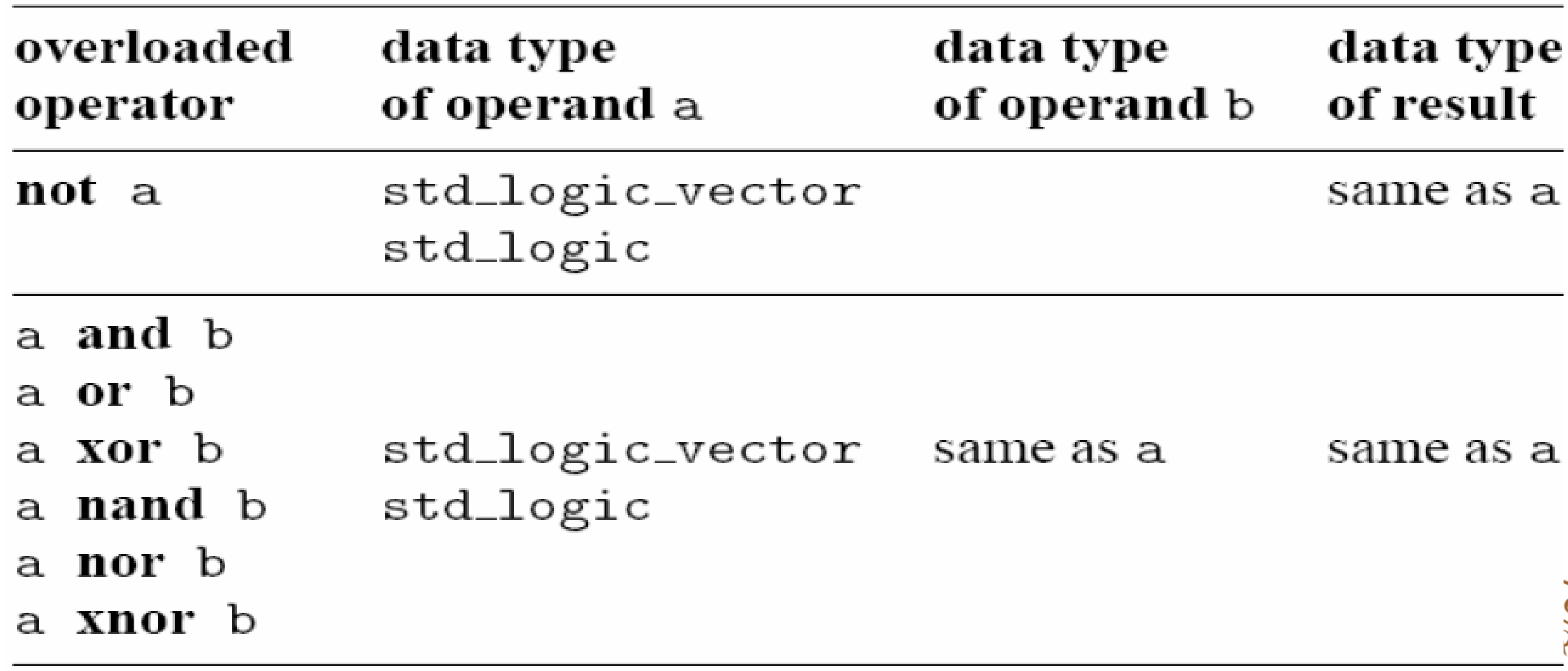
Precedence	Operators
Highest	** abs not
	* / mod rem
	+ − (identity and negation)
	& + − (addition and subtraction)
	sll srl sla sra rol ror
	= /= < <= > >=
Lowest	and or nand nor xor xnor

- Example: `a+b>c or a<d => ((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
not a	std_logic_vector std_logic		same as a
a **and** b a **or** b a **xor** b a **nand** b a **nor** b a **xnor** b	std_logic_vector std_logic	same as a	same as a

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, form 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:***

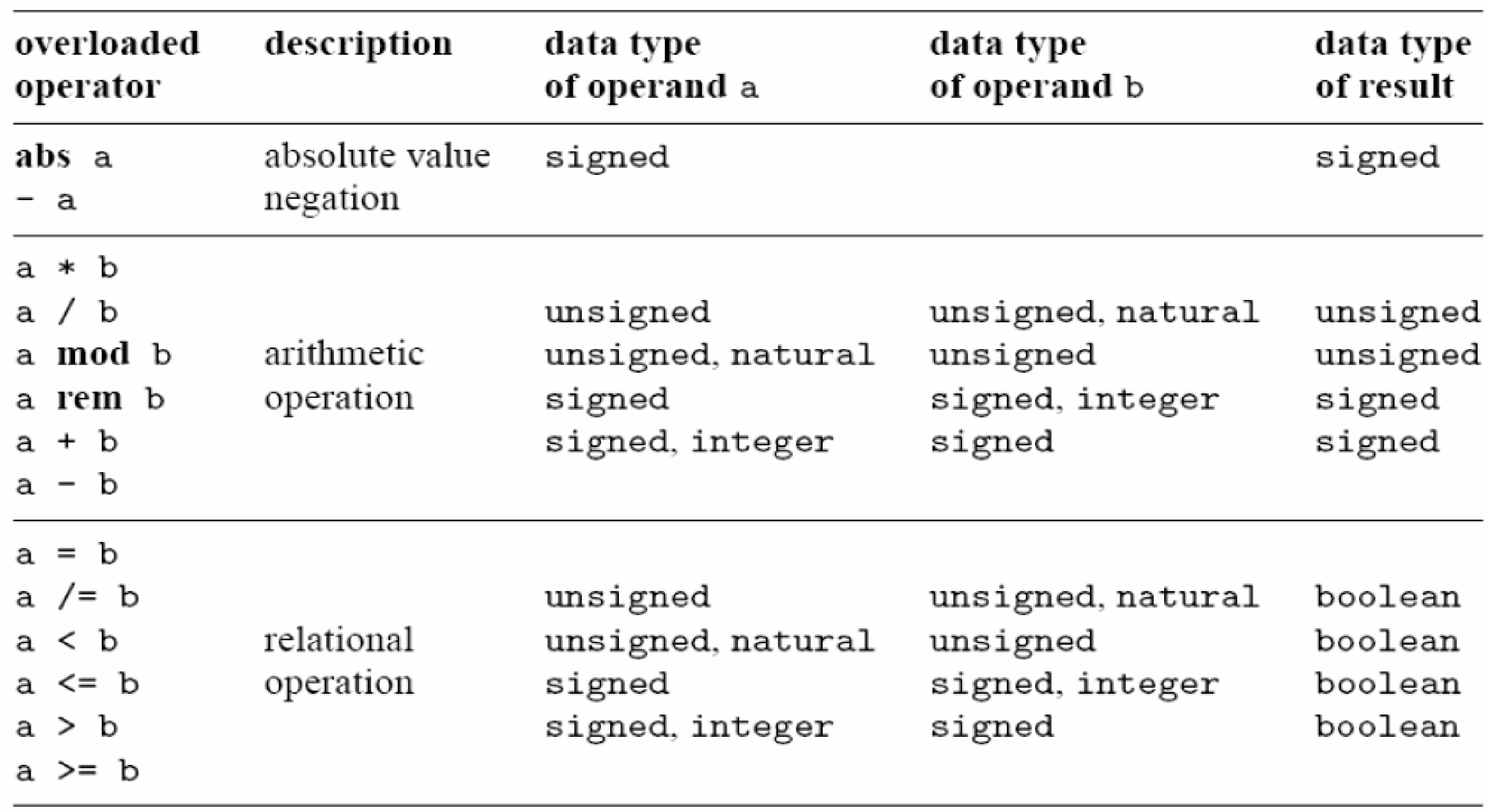
```
y <= a(1 downto 0) & a(7 downto 2);
```

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');
```

Overloaded Operator
IEEE numeric_std package

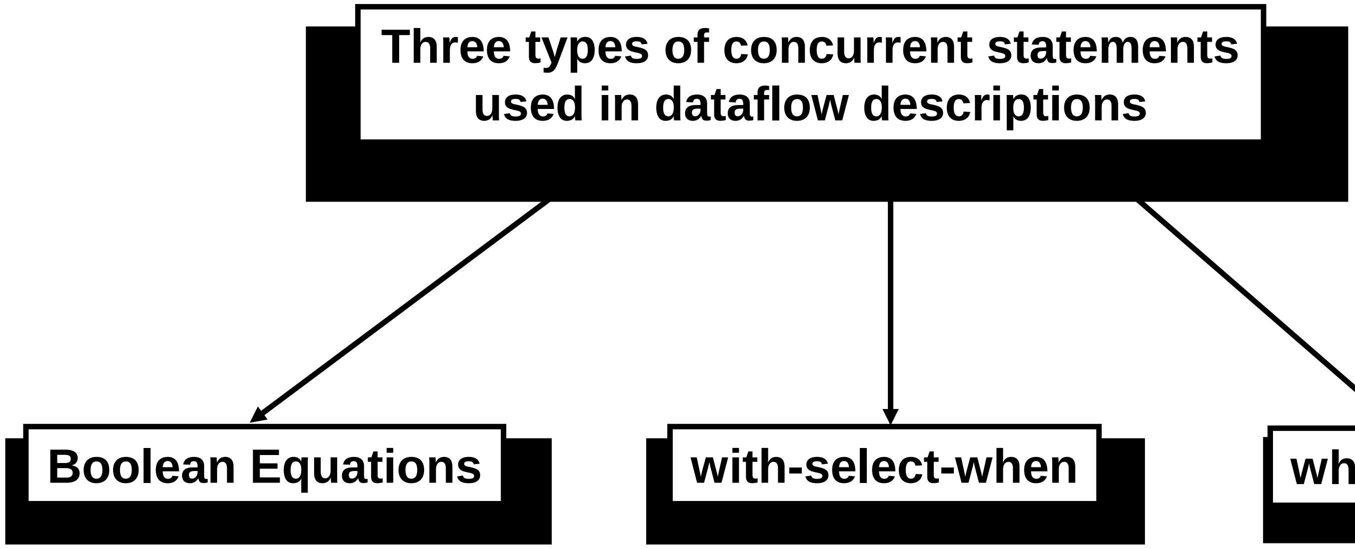


overloaded operator	description	data type of operand a	data type of operand b	data type of result
abs a	absolute value	signed		signed
– a	negation			
a * b				
a / b		unsigned	unsigned, natural	unsigned
a **mod** b	arithmetic	unsigned, natural	unsigned	unsigned
a **rem** b	operation	signed	signed, integer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

Overloaded Operator
IEEE numeric_std package: comparison

- std_logic_vector
```
"011" > “0100”   -- true
```
- unsigned
```
"011" > “0100“   -- false
```
- signed
```
"011" > "1000“   -- true
```

Concurrent Statements


Three types of concurrent statements used in dataflow descriptions
Boolean Equations
with-select-when
when-else

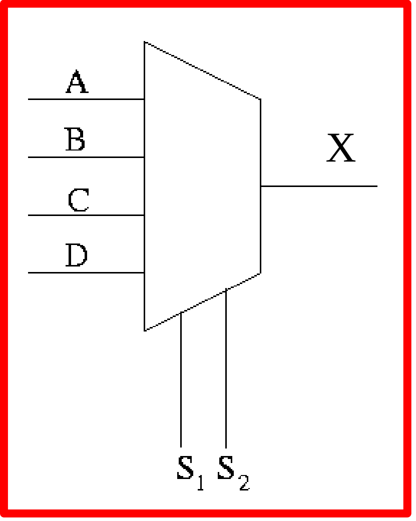
For concurrent signal assignments

For selective signal assignments

For conditional signal assignments

Concurrent Statements: When-else

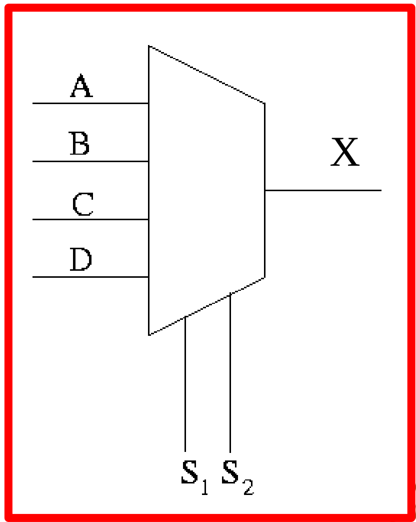
```
entity mux is port(a,b,c,d: in
std_logic_vector(3 downto 0);
   s: in std_logic_vector(1 downto 0);
   x: out std_logic_vector(3 downto 0));
end mux;
architecture mux_arch of mux is
begin
   x   <=   a when (s = "00") else
            b when (s = "01") else
            c when (s = "10") else
   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



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

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

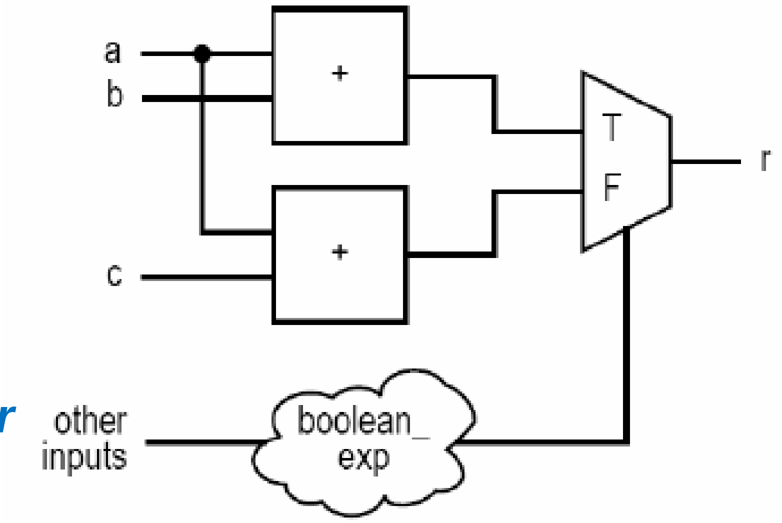
```
with select_expression select
     sig_name <= value_expr_1 when choice_1,
                 value_expr_2 when choice_2,
                 value_expr_3 when choice_3,
                      . . .
                 value_expr_n when choice_n;
```

Example I

- **Original code:**

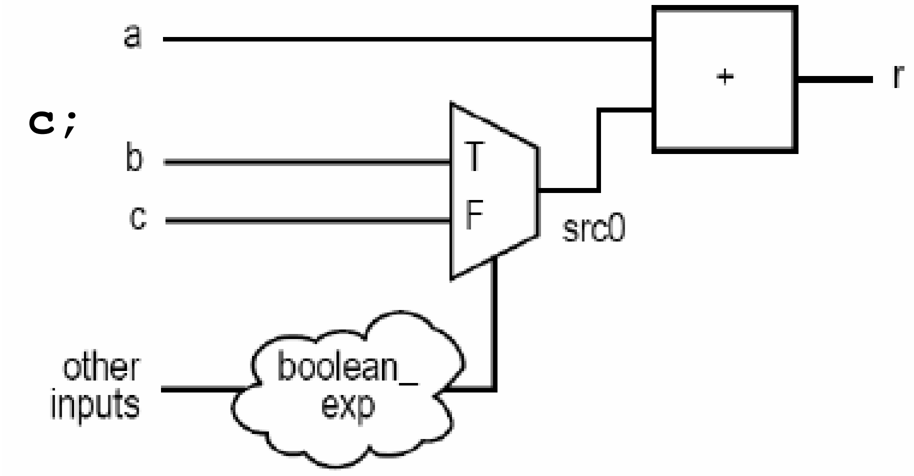
```
r <= a+b when boolean_exp else a+c;
```

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



- **Revised code:**

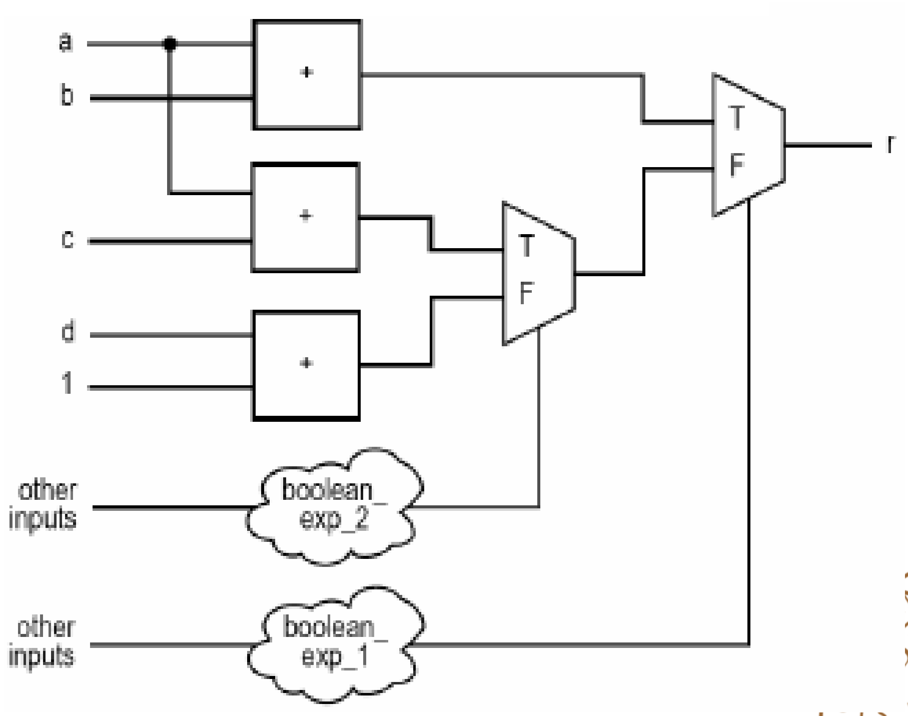
```
src0 <= b when boolean_exp else c;
r <= a + src0;
```



Example II

- **Original code:**

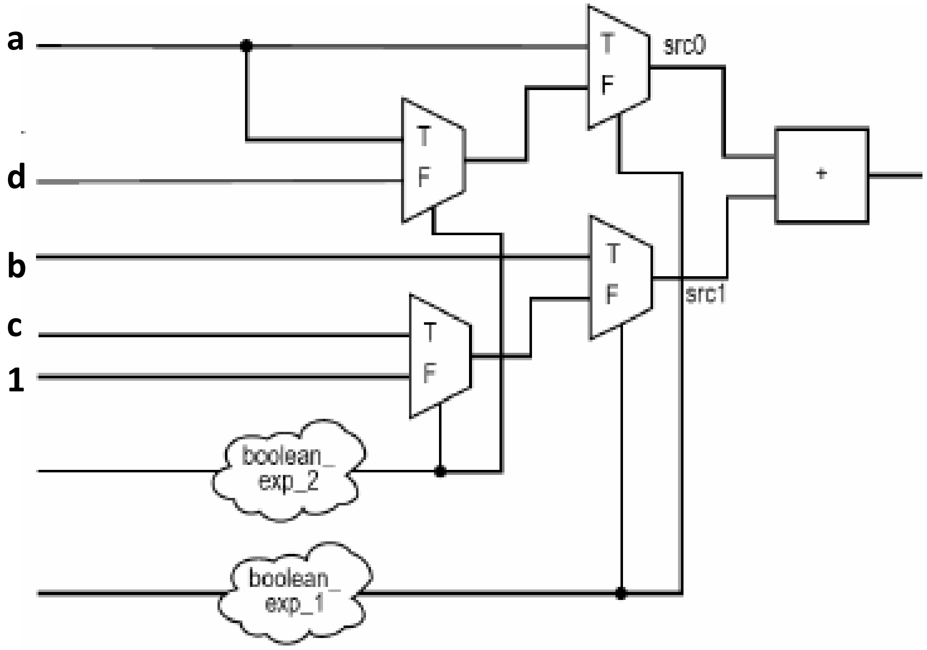
```
process(a,b,c,d,...)
begin
  if boolean_exp_1 then r <= a+b;
  elsif boolean_exp_2 then r <= a+c;
  else r <= d+1;
end if
end process;
```



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

Reading advice

FSM: RTL Hardware Design Using VHDL: P313-337

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

VHDL HandBook

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