ECE 545
Lecture 6

Dataflow Modeling in VHDL

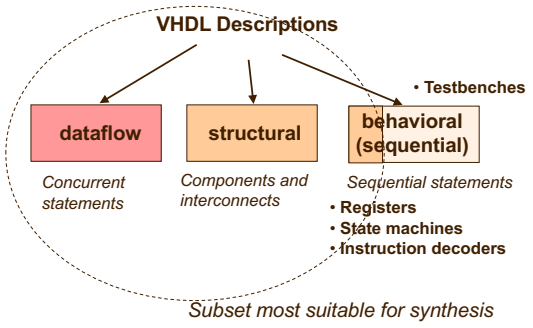
George Mason University

Required reading

- P. Chu, *RTL Hardware Design using VHDL*

Chapter 4, Concurrent Signal Assignment Statements of VHDL

Types of VHDL Description



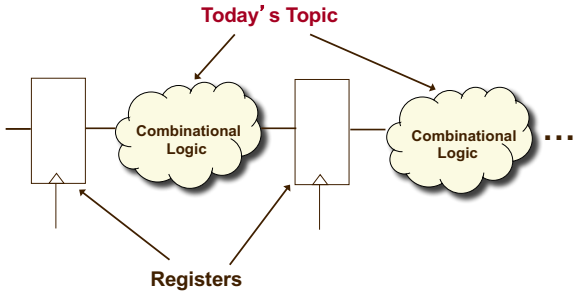
Subset most suitable for synthesis

Synthesizable VHDL

Dataflow VHDL Description ⇒ VHDL code synthesizable

Dataflow VHDL Description ⇏ VHDL code synthesizable

Register Transfer Level (RTL) Design Description



Data-Flow VHDL

Concurrent Statements

- **simple concurrent signal assignment (⇐)**
- **conditional concurrent signal assignment (when-else)**
- **selected concurrent signal assignment (with-select-when)**

Simple concurrent signal assignment

<=

```
target_signal <= expression;
```

7

Conditional concurrent signal assignment

When - Else

```
target_signal <= value1 when condition1 else
                 value2 when condition2 else
                   . . .
                 valueN-1 when conditionN-1 else
                 valueN;
```

8

Selected concurrent signal assignment

With –Select-When

```
with choice_expression select
   target_signal <= expression1 when choices_1,
                    expression2 when choices_2,
                       . . .
                    expressionN when choices_N;
```

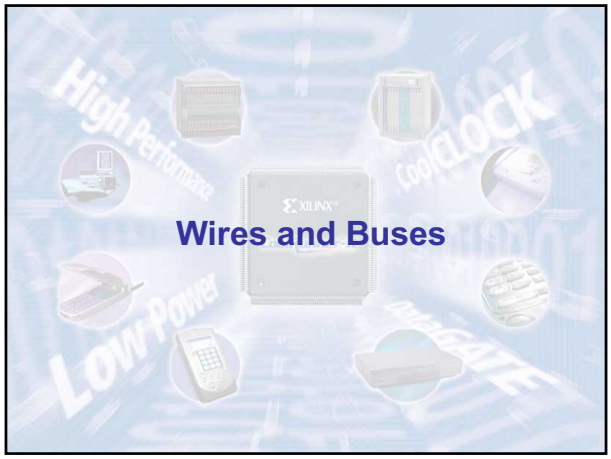
9

Data-Flow VHDL

Concurrent Statements

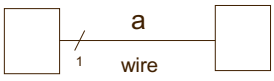
- **simple concurrent signal assignment (⇐)**
- **conditional concurrent signal assignment (when-else)**
- **selected concurrent signal assignment (with-select-when)**

10

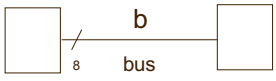


Signals

SIGNAL a : STD_LOGIC;

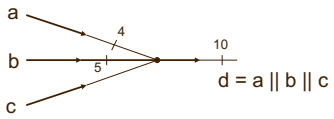


SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);



12

Merging wires and buses

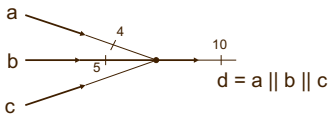



```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

d <=
```

13

Merging wires and buses

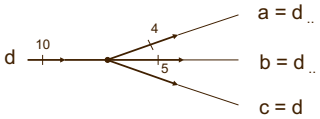



```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

d <= a & b & c;
```

14

Splitting buses

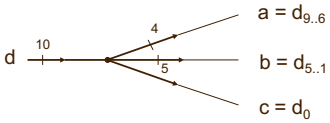



```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

a <=
b <=
c <=
```

15

Splitting buses

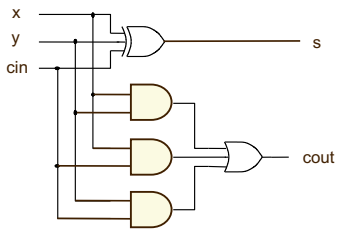



```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

a <= d(9 downto 6);
b <= d(5 downto 1);
c <= d(0);
```

16

Data-flow VHDL: Example



17

Data-flow VHDL: Example (1)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY fulladd IS
        PORT ( x      : IN      STD_LOGIC ;
               y      : IN      STD_LOGIC ;
               cin    : IN      STD_LOGIC ;
               s      : OUT     STD_LOGIC ;
               cout   : OUT     STD_LOGIC ) ;
END fulladd ;
```

18

Data-flow VHDL: Example (2)

```
ARCHITECTURE dataflow OF fulladd IS
BEGIN
        s     <=  x XOR y XOR cin ;
        cout  <=  (x AND y) OR (cin AND x) OR (cin AND y) ;
END dataflow ;
```

equivalent to

```
ARCHITECTURE dataflow OF fulladd IS
BEGIN
        cout  <=  (x AND y) OR (cin AND x) OR (cin AND y) ;
        s     <=  x XOR y XOR cin ;
END dataflow ;
```


Logic Operators

- Logic operators

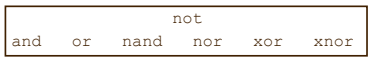


- Logic operators precedence

only in VHDL-93 or later

Highest

Lowest



No Implied Precedence

Wanted: y **= ab + cd**

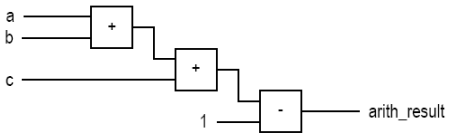
Incorrect

y <= a **and** b **or** c **and** d ;
equivalent to
y <= ((a **and** b) **or** c) **and** d ;
equivalent to
y = (ab + c)d

Correct

y <= (a **and** b) **or** (c **and** d) ;

arith_result <= a + b + c – 1;



Signal assignment statement with a closed feedback loop

- a signal appears in both sides of a concurrent assignment statement
- E.g.,
  q <= ((**not q**) **and** (**not** en)) **or** (d **and** en);
- Syntactically correct
- Form a closed feedback loop
- Should be avoided

Data-Flow VHDL

Concurrent Statements

- **simple concurrent signal assignment (⇐)**
- **conditional concurrent signal assignment (when-else)**
- **selected concurrent signal assignment (with-select-when)**

Conditional concurrent signal assignment

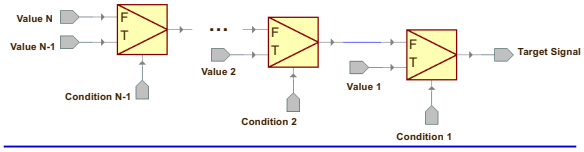
When - Else

```
target_signal <= value1 when condition1 else
                 value2 when condition2 else
                   . . .
                 valueN-1 when conditionN-1 else
                 valueN;
```


Most often implied structure

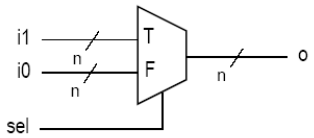
When - Else

```
target_signal <= value1 when condition1 else
                 value2 when condition2 else
                   . . .
                 valueN-1 when conditionN-1 else
                 valueN;
```

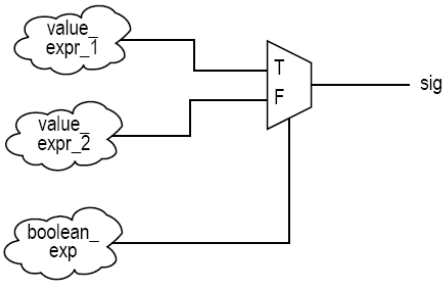


2-to-1 “abstract” mux

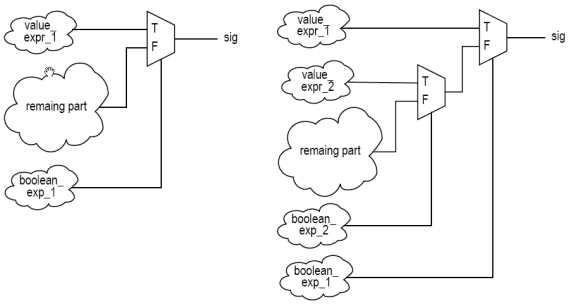
- sel has a data type of boolean
- If sel is true, the input from “T” port is connected to output.
- If sel is false, the input from “F” port is connected to output.

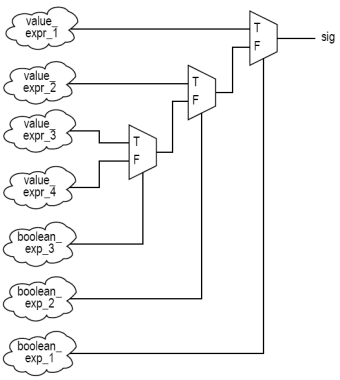



```
signal_name <= value_expr_1 when boolean_expr_1 else
               value_expr_2;
```



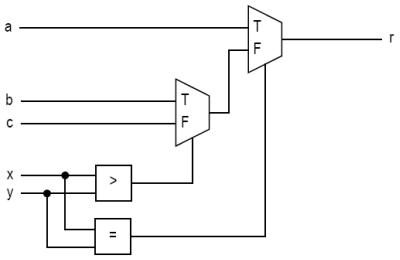

```
signal_name <= value_expr_1 when boolean_expr_1 else
               value_expr_2 when boolean_expr_2 else
               value_expr_3 when boolean_expr_3 else
               value_expr_4;
```





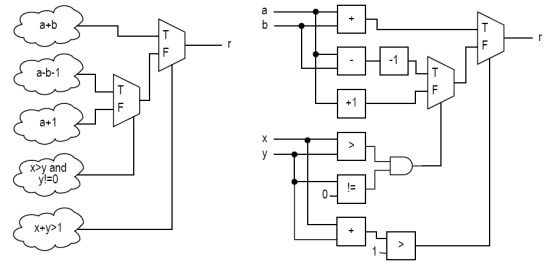
- E.g.,

```
signal a,b,c,x,y,r: std_logic;
. . .
r <= a when x=y else
     b when x>y else
     c;
```



- E.g.,

```
. . .
signal a,b,r: unsigned(7 downto 0);
signal x,y: unsigned(3 downto 0);
. . .
r <= a+b when x+y>1 else
     a-b-1 when x>y and y!=0 else
     a+1;
```



Signed and Unsigned Types

Behave exactly **like**
STD_LOGIC_VECTOR
plus, they determine whether a given vector should be treated as a signed or unsigned number.
Require
USE ieee.numeric_std.all;

Operators

- Relational operators

=	/=	<	<=	>	>=

- Logic and relational operators precedence

Highest ↓ Lowest

not					
=	/=	<	<=	>	>=
and	or	nand	nor	xor	xnor

Priority of logic and relational operators

compare a = bc

Incorrect
… when a = b **and** c else …
equivalent to
… when (a = b) **and** c else …

Correct
… when a = (b **and** c) else …

Data-Flow VHDL

Concurrent Statements

- **simple concurrent signal assignment (⇐)**
- **conditional concurrent signal assignment (when-else)**
- **selected concurrent signal assignment (with-select-when)**

Selected concurrent signal assignment

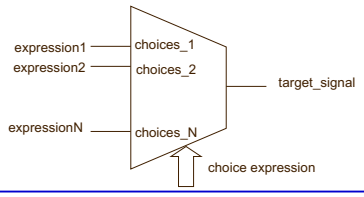
With –Select-When

```
with choice_expression select
   target_signal <= expression1 when choices_1,
                    expression2 when choices_2,
                        . . .
                    expressionN when choices_N;
```


Most Often Implied Structure

With –Select-When

```
with choice_expression select
   target_signal <= expression1 when choices_1,
                    expression2 when choices_2,
                        . . .
                    expressionN when choices_N;
```



Allowed formats of *choices_k*

WHEN value

WHEN value_1 | value_2 | .... | value N

WHEN OTHERS

Allowed formats of *choice_k - example*

```
WITH sel SELECT
     y <= a WHEN "000",
          c WHEN "001" | "111",
          d WHEN OTHERS;
```


Syntax

- Simplified syntax:

  with select_expression **select**
  signal_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;

- select_expression
  - Discrete type or 1-D array
  - With finite possible values
- choice_i
  - A value of the data type
- Choices must be
  - mutually exclusive
  - all inclusive
  - **others** can be used as last choice_i

E.g., 4-to-1 mux

```
architecture sel_arch of mux4 is
begin
   with s select
     x <= a when "00",
          b when "01",
          c when "10",
          d when others;
end sel_arch ;
```

input s	output x
00	a
01	b
10	c
11	d

- Can "11" be used to replace **others**?

```
with s select
  x <= a when "00",
       b when "01",
       c when "10",
       d when "11";
```


E.g., 2-to-2^2 binary decoder

```
architecture sel_arch of decoder4 is
begin
   with sel select
     x <= "0001" when "00",
          "0010" when "01",
          "0100" when "10",
          "1000" when others;
end sel_arch ;
```

input s	output x
00	0001
01	0010
10	0100
11	1000

E.g., simple ALU

```
architecture sel_arch of simple_alu is
   signal sum, diff, inc: std_logic_vector(7 downto 0);
begin
   inc <= std_logic_vector(signed(src0)+1);
   sum <= std_logic_vector(signed(src0)+signed(src1));
   diff <= std_logic_vector(signed(src0)-signed(src1));
   with ctrl select
      result <= inc            when "000"|"001"|"010"|"011",
                sum            when "100",
                diff           when "101",
                src0 and src1  when "110",
                src0 or src1   when others;
end sel_arch;
```

input ctrl	output result
0--	src0 + 1
100	src0 + src1
101	src0 - src1
110	src0 **and** src1
111	src0 **or** src1

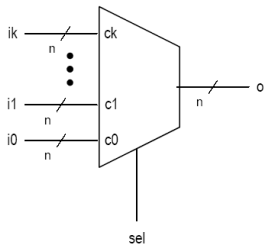
E.g., Truth table

```
library ieee;
use ieee.std_logic_1164.all;
entity truth_table is
   port(
      a,b: in  std_logic;
      y: out   std_logic
   );
end truth_table;
architecture a of truth_table is
   signal tmp: std_logic_vector(1 downto 0);
begin
   tmp <= a & b;
   with tmp select
      y <= '0' when "00",
           '1' when "01",
           '1' when "10",
           '1' when others; -- "11"
end a;
```

input a b	output y
00	0
01	1
10	1
11	1

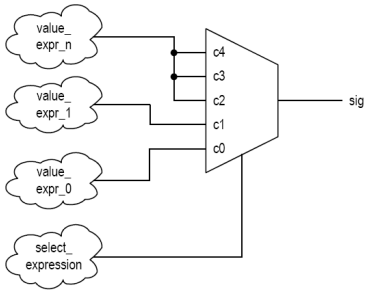
Conceptual implementation

- Achieved by a multiplexing circuit
- Abstract (k+1)-to-1 multiplexer
  - sel is with a data type of (k+1) values: c0, c1, c2, . . . , ck



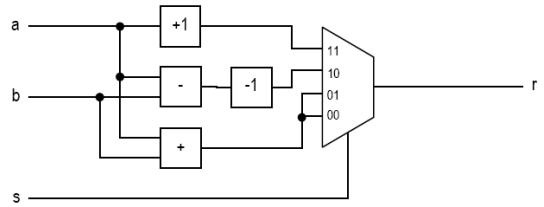
– select_expression is with a data type of 5 values: c0, c1, c2, c3, c4

```
with select_expression select
   sig <= value_expr_0 when c0,
          value_expr_1 when c1,
          value_expr_n when others;
```

- E.g.,

```
signal a,b,r: unsigned(7 downto 0);
signal s: std_logic_vector(1 downto 0);
   . . .
   with s select
      r <= a+1   when "11",
           a-b-1 when "10",
           a+b   when others;
```



3. Conditional vs. selected signal assignment

- Conversion between conditional vs. selected signal assignment
- Comparison

From selected assignment to conditional assignment

```
with sel select
   sig <= value_expr_0 when c0,
          value_expr_1 when c1|c3|c5,
          value_expr_2 when c2|c4,
          value_expr_n when others;


sig <=
   value_expr_0 when (sel=c0) else
   value_expr_1 when (sel=c1) or (sel=c3) or (sel=c5) else
   value_expr_2 when (sel=c2) or (sel=c4)else
   value_expr_n;
```


From conditional assignment to selected assignment

```
sig <= value_expr_0 when bool_exp_0 else
       value_expr_1 when bool_exp_1 else
       value_expr_2 when bool_exp_2 else
       value_expr_n;


sel(2) <= '1' when bool_exp_0 else '0';
sel(1) <= '1' when bool_exp_1 else '0';
sel(0) <= '1' when bool_exp_2 else '0';
with sel select
   sig <= value_expr_0 when "100"|"101"|"110"|"111",
          value_expr_1 when "010"|"011",
          value_expr_2 when "001",
          value_expr_n when others;
```

Comparison

- Selected signal assignment:
  - good match for a circuit described by a functional table
  - E.g., binary decoder, multiplexer
  - Less effective when an input pattern is given a preferential treatment

- Conditional signal assignment:
  - good match for a circuit that needs to give preferential treatment for certain conditions or to prioritize the operations
  - E.g., priority encoder
  - Can handle complicated conditions. e.g.,

```
pc_next <=
   pc_reg + offset when (state=jump and a=b) else
   pc_reg + 1 when (state=skip and flag='1') else
. . .
```


- May "over-specify" for a functional table based circuit.
- E.g., mux

```
x <= a when (s="00") else
     b when (s="01") else
     c when (s="10") else
     d;

x <= c when (s="10") else
     a when (s="00") else
     b when (s="01") else
     d;

x <= c when (s="10") else
     b when (s="01") else
     a when (s="00") else
     d;
```


when-else vs. with-select-when (1)

"when-else" should be used when:

1) there is only one condition (and thus, only one else), as in the 2-to-1 MUX
2) conditions are independent of each other (e.g., they test values of different signals)
3) conditions reflect priority (as in priority encoder); one with the highest priority need to be tested first.

when-else vs. with-select-when (2)

"with-select-when" should be used when there is

1) more than one condition
2) conditions are closely related to each other (e.g., represent different ranges of values of the same signal)
3) all conditions have the same priority (as in the 4-to-1 MUX).