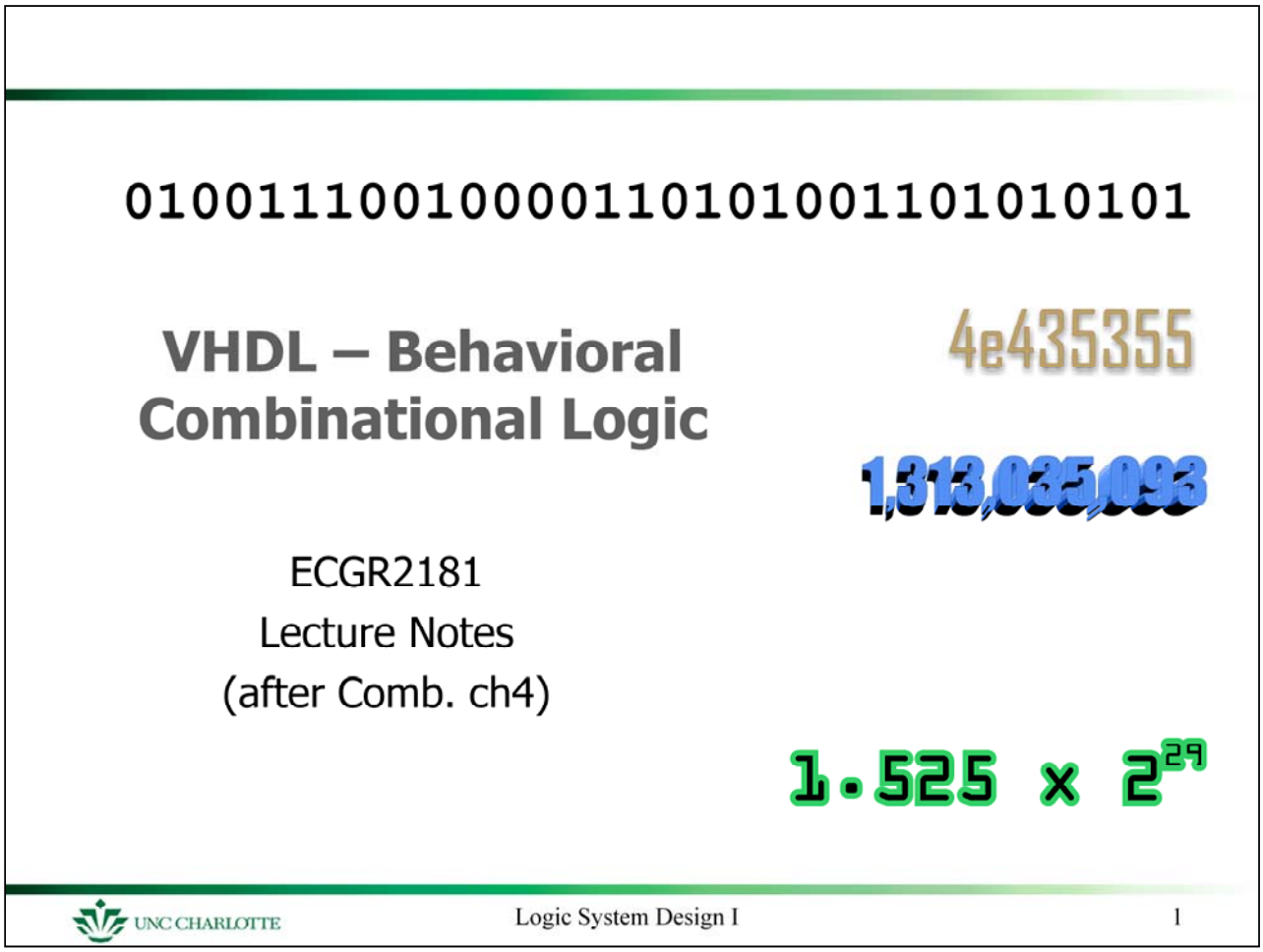


These are all different interpretations of the same bit string.

Behavioral Design

- Structural design <- model of circuit describing it's component pieces
  - Used to bring together various models and/or components to make a larger circuit … analogous to the top-level schematic
  - Good for CAD tools to create automated model generation from schematics … Xilinx Synthesis tool does this.
- Behavioral design <- modeling circuit by describing it's behavior
  - Key benefit of HDLs … being able to easily describe complex logic behavior using simple statements
  - Eg.,

    `S <= A + B;` … would describe an adder

    `Y <= i0 when s = '0' else i1;` … is a 2x1 mux

Some Syntax

- Concurrent signal-assignment statement:

  signal-name <= *expression*;

  - <= is the assignment operator

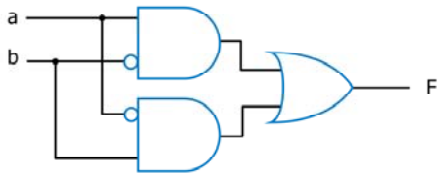
  - Eg.,

    ```
    F <= a and not b or not a and b;
    ```

    same as, but easier to read …

    ```
    F <= (a and not(b)) or (not(a) and b);
    ```

  – Precedence … *not*, *and*, then *or*



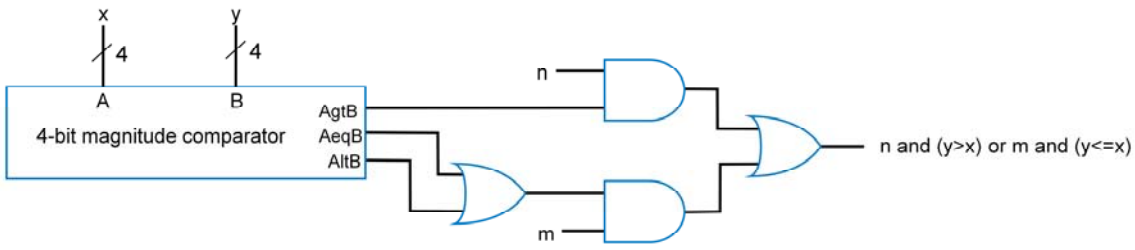
Conditional Assignment

- Conditional concurrent signal-assignment statement:

```
signal-name <= expression  when  boolean-expression  else
               expression  when  boolean-expression  else
                 . . .
               expression  when  boolean-expression  else
               expression;
```

  - *boolean-expression* – combines individual boolean terms like *and, or,* and *not*.
    - Also, includes relational operators …
      =, /= (inequality), >, >=, <, and <=
      (in this context <= means *'less then equal to'*)

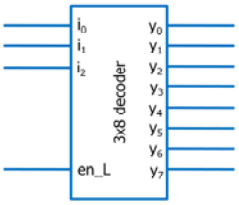
Eg. a boolean-expression could be … `n and (y<x) or m and (y>=x)`



Conditional Assignment (cont)

- Example:
  - For a decoder, the input code word is i(2 downto 0) and output code word is y(7 downto0), with active-low enable (en_L):

```
y   <=  "00000001"     when  en_L='0' and i="000"  else
        "00000010"     when  en_L='0' and i="001"  else
        "00000100"     when  en_L='0' and i="010"  else
        "00001000"     when  en_L='0' and i="011"  else
        "00010000"     when  en_L='0' and i="100"  else
        "00100000"     when  en_L='0' and i="101"  else
        "01000000"     when  en_L='0' and i="110"  else
        "10000000"     when  en_L='0' and i="111"  else
        "00000000";
```



Selected Signal-Assignment

• This statement is given an *expression* and assigns a value (*signal-value*) to the *signal-name* when the expressions value is among the choices:

```
with expression select
   signal-name <=      signal-value  when  choices,
                       signal-value  when  choices,
                         . . .
                       signal-value  when  choices;
```

• Eg., for a prime number detector … for a 4-bit input N(3 downto0)

```
with N select
     F <= '1' when "0001",
          '1' when "0010",
          '1' when "0011" | "0101" | "0111",
          '1' when "1011" | "1101",
          '0' when others;
```

Selected Signal-Assignment

- Or, Prime number detector could be performed like this …

```
with CONV_INTEGER(N) select
    F  <=  '1' when 1|2|3|5|7|11|13,
           '0' when others;
```

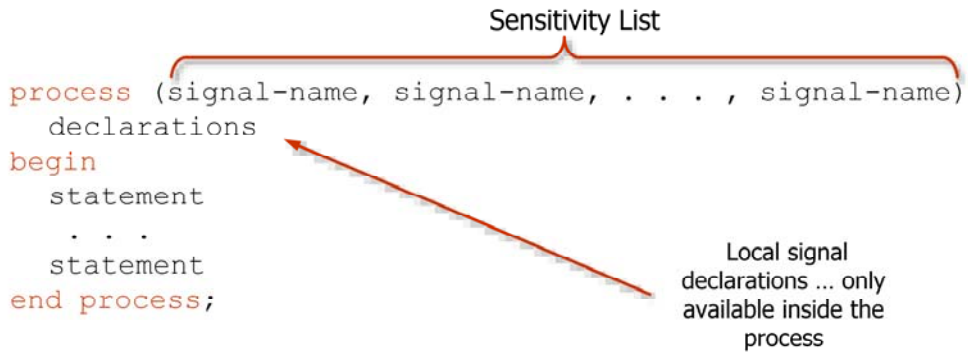
  - CONV_INTEGER() is a useful built in function that allows the user to enter decimal numbers instead of binary sequences.

Process Statements

- So far, we have discussed concurrent statements … unlike computer software, HDL's process all concurrent statements at the same time.

- One exception is the process statement, while the whole process block runs at the same time as concurrent statements, the statements in the process statement can be thought of as executing sequentially.

- The process statement will be either running or suspended.



Processes (cont.)

• If a value in the sensitivity-list changes, the statements within the process stmt block will execute sequentially starting with the first expression.

• Actually, when creating a *synthesizable model*, the process statement is interpreted in specific ways (based on synthesizer CAD tool.) Because actual components operate concurrently.

• *Synthesizable model* – model that can be converted directly into an actual operational circuit.

- Not all VHDL code is synthesizable …
- Just because it synthesizes does not mean it is being synthesized the way you think.
- One Pit-fall is to think that if it simulates it will work the way you think it will … Modelsim interprets VHDL … be careful.

VHDL is not a software language!

Warning …

VHDL is not a SOFTWARE language!!!

Do not be tempted … if you treat it as such, you will get bitten.

Hopefully, it will not cost you your grade or your boss his budget.

`if` Statements

- The `if` statement can only be used inside a process statement.

```
if boolean-expression then statements
end if;


if boolean-expression then
   statements (true)
else
   statements (false)
end if;


if boolean-expression then statements
elsif boolean-expression then statements
end if;


if boolean-expression then statements
elsif boolean-expression then statements
 . . .
elsif boolean-expression then statements
else statements
end if;
```

`if, elsif` Example

- Prime number detector example:

```
Process (N)
begin
   if N="0001" or N="0010" or N="0011" then
         F <= '1';
   elsif N="0101" or N="0111" then
         F <= '1';
   elsif N="1011" or N="1101" then
         F <= '1';
   else
         F <= '0';
   end if;
end process;
```

Another `if, elsif` example

- Model a comparator circuit; A and B are both bit vectors (busses) they could be any size 3 or 128 bits, no limits …

```
process (A, B)
begin
      aLTb <= '0'; aGTb <= '0'; aEQb <= '0';
      if A<B then
            aLTb <= '1';
      elsif A>B then
            aGTb <= '1';
      elsif A=B then
            aEQb <= '1';
      end if;
end process;
```

`case` Statements

- Method for easily replacing multiple nested `if` statements

```
case expression is
        when choices =>
                statements
        when choices =>
                statements
          . . .
        when choices =>
                statements
end case;
```

- `case` statements can be used inside or outside of `process` statements.

case example

• Behavioral model for 8x1 Mux using a case statement. Muxes can be implemented in other ways … this is one example.

– sel is a 3-bit bus {{ sel(2 downto 0) }}

```
case sel is
   when "000" =>
            y <= i0;
   when "001" =>
            y <= i1;
   when "010" =>
            y <= i2;
   when "011" =>
            y <= i3;
   when "100" =>
            y <= i4;
   when "101" =>
            y <= i5;
   when "110" =>
            y <= i6;
   when "111" =>
            y <= i7;
   when others =>
            y <= '0';
end case;
```

Signal Concatenation

- Signals can be combined with other signals to treat them as a whole.
  - Just use the concatenation operator &

- Eg. f <= en_L & g(3 downto 0);
  -- how many wires in the f bus? (assume en_L is a discrete signal)

  - btw: '--' signifies the rest of the line is a comment

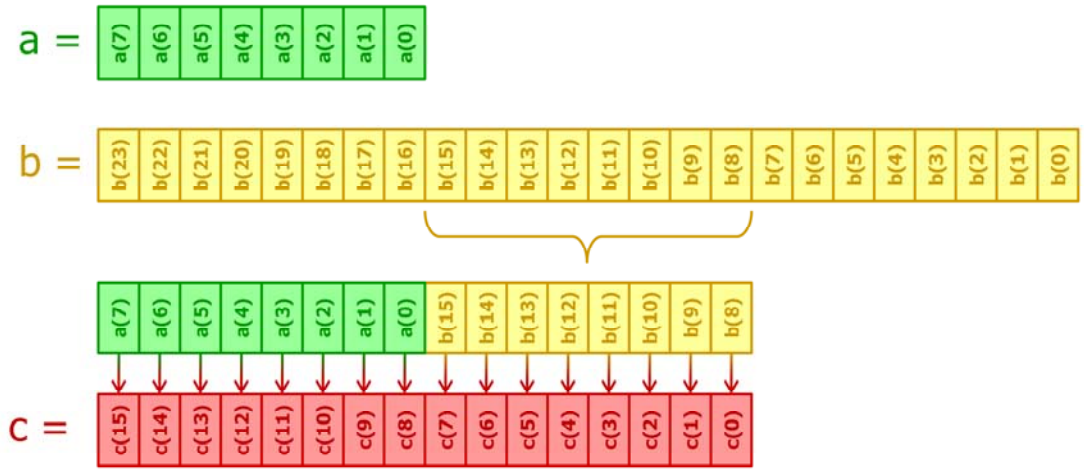
- From the previous example; this time a 4x1 mux with an active high enable (en):

```
case en & sel is
   when "100" =>
                    y <= i0;
   when "101" =>
                    y <= i1;
   when "110" =>
                    y <= i2;
   when "111" =>
                    y <= i3;
   when others =>
                    y <= '0';
end case;
```

Concatenating Busses

- Concatenating busses
  - a(7 downto 0) and b(23 downto 0)
  - make a new bus called c(15 downto 0)

```
c <= a & b(15 downto 8);
```



Internal Signals

- Signals internal to the design need to be declared.
- For this course we will be giving all signals the type *std_logic*
- Internal signals are declared between the Achitecture line and begin

```
architecture prime_arch of prime9 is
        -- declarations
begin

        -- behavioral model

end case;
```

Internal Signals (cont.)

- Discrete signals – a single signal … one wire

```
signal signal-name: std_logic;
```

- Bus (or bit-vector):

```
signal bus-name: std_logic_vector(size-1 downto 0);
```

```
architecture prime_arch of prime9 is
        signal prime_int: std_logic;
        signal inp: std_logic_vector(3 downto 0);
begin

        -- behavioral model

end case;
```

Inside Synthesis w/ Xilinx (XST)

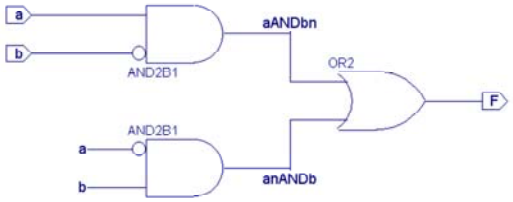
- For the circuit given to the below ...

  … the following VHDL is generated by Xilinx Synthesis Tool (XST).

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
-- synopsys translate_off
library UNISIM;
use UNISIM.Vcomponents.ALL;
-- synopsys translate_on

entity xorTwo is
   port ( a : in    std_logic;
          b : in    std_logic;
          F : out   std_logic);
end xorTwo;

architecture BEHAVIORAL of xorTwo is
   attribute BOX_TYPE   : string ;
   signal aANDbn : std_logic;
   signal anANDb : std_logic;
   component AND2B1
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of AND2B1 : component is "BLACK_BOX";
   component OR2
      port ( I0 : in    std_logic;
             I1 : in    std_logic;
             O  : out   std_logic);
   end component;
   attribute BOX_TYPE of OR2 : component is "BLACK_BOX";
begin
```




```
begin
   XLXI_1 : AND2B1
      port map (I0=>b,
                I1=>a,
                O=>aANDbn);

   XLXI_2 : OR2
      port map (I0=>aANDbn,
                I1=>anANDb,
                O=>F);

   XLXI_3 : AND2B1
      port map (I0=>a,
                I1=>b,
                O=>anANDb);

end BEHAVIORAL;
```