Selected Concurrent Signal Assignment

The selected concurrent signal assignment statement is modeled after the "case statement" in software programming languages.

The general form of this statement:

```
WITH discriminant SELECT
  target_signal <= value1 WHEN choice1,
                   value2 WHEN choice2,
                   value3 WHEN choice3,
                     ........
                   valueN WHEN choiceN;
                  [default_value WHEN OTHERS];
```

This statement executes when any the discriminant, value or choice expressions changes value. When it does execute, the choice clauses are evaluated. The target signal is assigned the value corresponding to the choice that matches the discriminant.

Important points for this statement:

- The discriminant must have finite discrete values. (can be enumerated).
  ```
  ERROR: Expression must return a discrete value.
  ```
- You must use or list all possible values for "choice".
  ```
  ERROR: Case statement only covers 5 out of 729 cases.
  ```
- Only one of the choices can match the discriminant.
  ```
  ERROR: Case choice has already been specified on line 32
  ```

About "OTHERS"

The keyword **OTHERS** can be powerfully used in many situations. In general it is used to allow matching to an unspecified number of possible values of a variable. There ay be only one alternative that uses the others choice and if included in a list, it must be the last choice. In essence, it says, if a match has not yet been found and the value of the variable is within range of its type, then match with **OTHERS**.

We will see several other uses of **OTHERS** in the future.

Selected Concurrent Signal Assignment

An example from “SPAM”

```
----------------------------------------------------------------------
--2:1 mux, 16 bits wide
----------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY mux2_1_16wide IS
 PORT(
      in_a   : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);  --input a
      in_b   : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);  --input b
      sel    : IN  STD_LOGIC;                      --select input
      output : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)   --data output
      );
  END mux2_1_16wide;

ARCHITECTURE beh OF mux2_1_16wide IS
  BEGIN
    WITH sel SELECT
      output <= in_a WHEN ‘0’,
                in_b WHEN ‘1’,
                (OTHERS => ‘X’) WHEN OTHERS;
END beh;
```

OTHERS again

Here we see **OTHERS** used to match cases where sel is not ‘1’ or ‘0’ in the **WHEN OTHERS** clause. i.e.:

(OTHERS => ‘X’) <u>WHEN OTHERS</u>;

OTHERS is also used to provide a shorthand method of saying, “make all the bits of the target signal ‘X” for however many bits are in target signal.

<u>(OTHERS => ‘X’)</u> WHEN OTHERS;

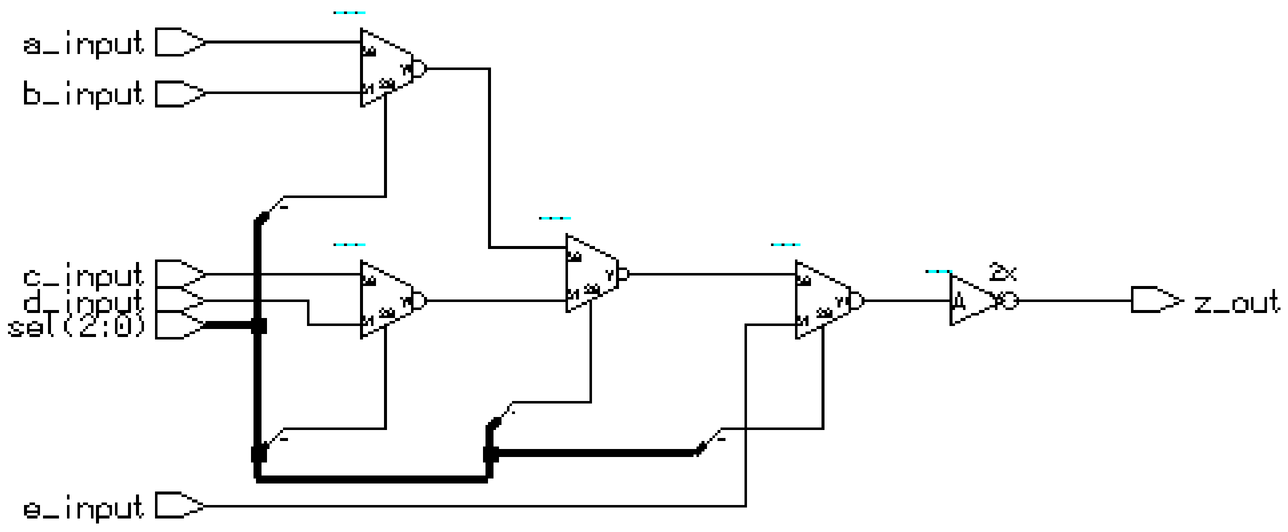
Why was ‘X ‘assigned to output when sel was neither ‘0’ or ‘1’?

Selected Concurrent Signal Assignment

A more simple example with synthesis results.

```
--5:1 mux, 1 bit wide
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mux5_1_1wide IS
  PORT(
     a_input   : IN STD_LOGIC;  --input a
     b_input   : IN STD_LOGIC;  --input b
     c_input   : IN STD_LOGIC;  --input c
     d_input   : IN STD_LOGIC;  --input d
     e_input   : IN STD_LOGIC;  --input e
     sel       : IN STD_LOGIC_VECTOR(2 DOWNTO 0);  --sel input
     z_out     : OUT STD_LOGIC  --data out
     );
END mux5_1_1wide;
ARCHITECTURE beh OF mux5_1_1wide IS
  BEGIN
     WITH sel SELECT
     z_out <= a_input WHEN “000”,
              b_input WHEN “001”,
              c_input WHEN “010”,
              d_input WHEN “011”,
              e_input WHEN “100”,
             ‘X’ WHEN OTHERS; --if sel could be be 110, 111? correct?
 END beh;
```



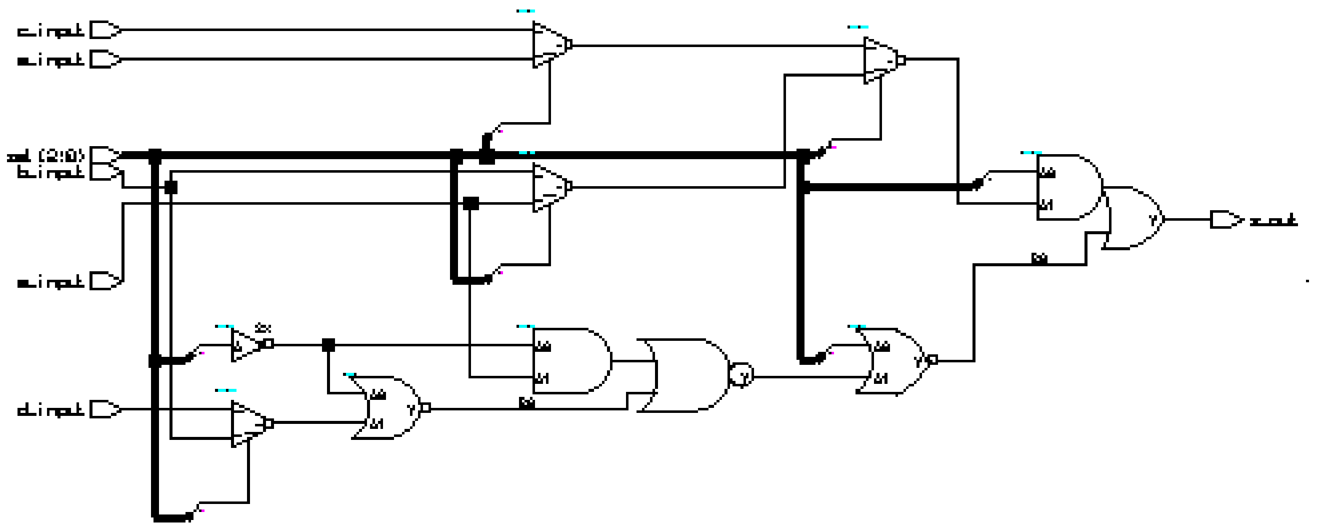
How will this circuit react to sel(2:0) values greater than “100”?

Making Choices

When we want the same target signal assignment to happen for several discriminant choices how do we specify it? Lets alter the function of our mux example as follows. The entity declaration is identical to before.

```
ARCHITECTURE beh OF mux5_1_1wide IS
  BEGIN
     WITH sel SELECT
     z_out <= a_input WHEN “000” | “001” | “111”,
              b_input WHEN “011” | “101”,
              c_input WHEN “010”,
              d_input WHEN “100”,
              e_input WHEN “110”,
             ‘X’ WHEN OTHERS;
 END beh;
```

The signal z_out gets the value of a_input when sel is equal to “000”, “001” or “111”. Signal z_out gets the value of b_input when sel is equal to “011” or “101”. The synthesized version of this mux looks like this:



As you can see, once a model is synthesized it can be hard to figure out how it works.