CPE 462
VHDL: Simulation and Synthesis

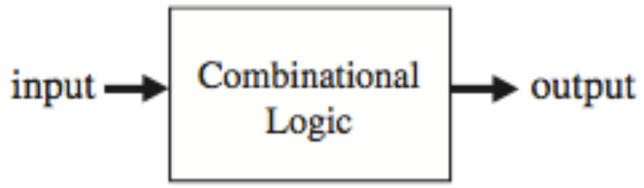
Topic #05 - a) WHEN statements

Concurrent code

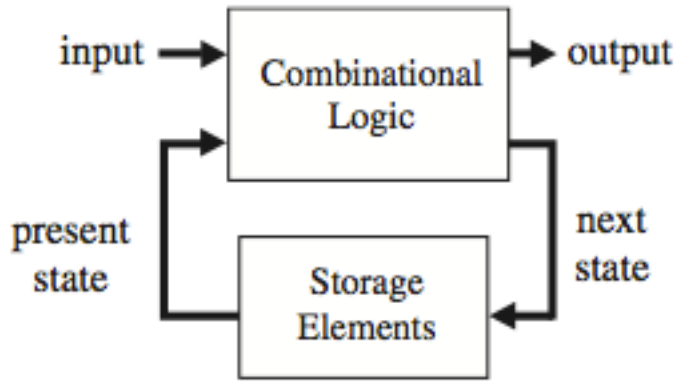
- We finished the basic foundations of VHDL, now we focus on the design (code) itself.
- VHDL code can be concurrent (parallel) or sequential. First we will discuss concurrent code.
- Concurrent code can be constructed with:
  - WHEN and GENERATE statements
  - Operators (AND, NOT, +, *, sll, etc.)
  - BLOCK

Concurrent versus sequential logic

- Combinational logic is that in which the output of the circuit depends solely on the current inputs.
- The system requires no memory and can be implemented using conventional logic gates.

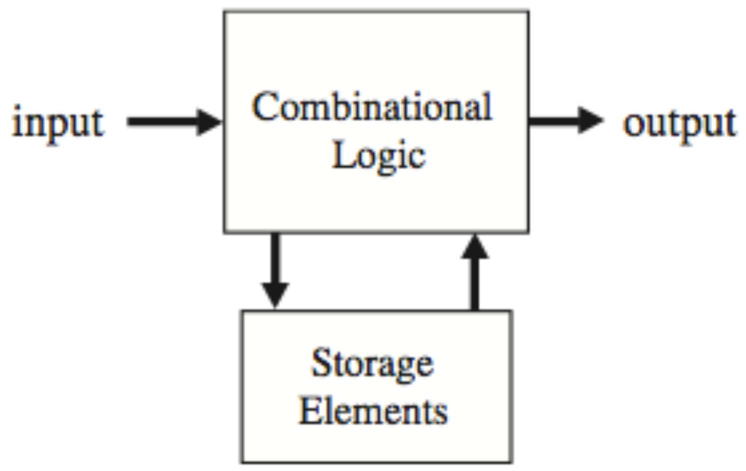


- Sequential logic is defined as that in which the output does depend on previous inputs.
- Storage elements are required, which are connected to the combinational logic block through a feedback loop.



Common mistake

- Not every circuit that has storage elements (flip-flops) is sequential.
- For example a in a RAM cell the storage elements appear in a forward path rather than in a feedback loop.
- The memory-read operation depends only on the address vector presently applied to the RAM input, with the retrieved value having nothing to do with previous memory accesses.



Concurrent versus sequential code

- VHDL code is inherently concurrent (parallel).
- Only statements placed inside a PROCESS, FUNCTION, or PROCEDURE are sequential.
- Within these blocks the execution is sequential, however the block, is concurrent with any other (external) statements.
- Concurrent code is also called dataflow code.

```
entity main_block is
   port (clk, a, b : in bit);
end main_block;

architecture myarch of main_block is
signal x,y,z : bit;

begin
   x <= a AND b;
   y <= a XOR b;

   process (clk)
      begin
         y <= a OR b;
      end process

end architecture;
```

Example

- Consider a code with three concurrent statements (stat1, stat2, stat3).

$$\begin{array}{c}\text{stat1}\\ \text{stat2}\\ \text{stat3}\end{array} \equiv \begin{array}{c}\text{stat3}\\ \text{stat2}\\ \text{stat1}\end{array} \equiv \begin{array}{c}\text{stat1}\\ \text{stat3}\\ \text{stat2}\end{array} \equiv \text{etc.}$$

- Any of the these alternatives will render the same physical circuit
- Since the order does not matter, purely concurrent code can not be used to implement synchronous circuits (only exception is a GUARDED BLOCK).
- In general we can only build combinational logic circuits with concurrent code. To obtain sequential logic circuits, sequential code must be used.

Concurrent code outline

In summary, in concurrent code the following can be used:

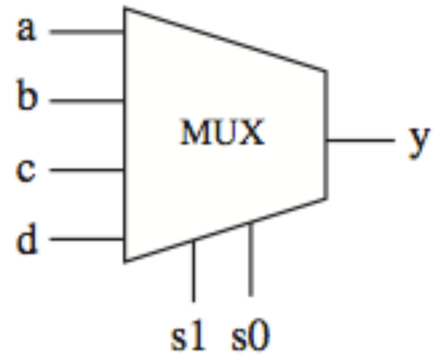
- Operators
- The WHEN statement (WHEN/ELSE or WITH/SELECT/WHEN)
- The GENERATE statement
- The BLOCK statement

Important: We haven't studied these, but WHEN and GENERATE can only be used outside PROCESSES, FUNCTIONS, or PROCEDURES.

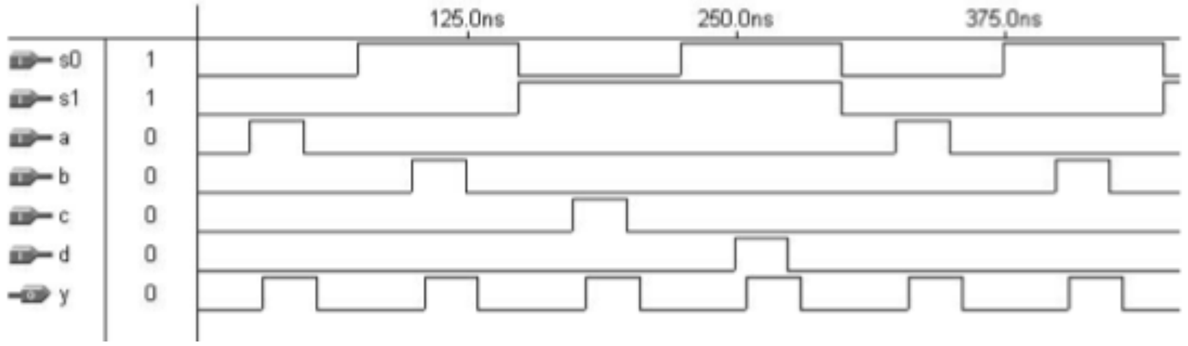
Concurrent code with operators

- We've seen this before.
- It's a 4-input, one bit per input multiplexer done using only operators.
- The output must be equal to the input selected by the selection bits, s1-s0.
- Not a very user friendly implementation!

4-to-1 MUX



```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d, s0, s1: IN STD_LOGIC;
          y: OUT STD_LOGIC);
END mux;
---------------------------------------------
ARCHITECTURE pure_logic OF mux IS
BEGIN
   y <=  (a AND NOT s1 AND NOT s0) OR
         (b AND NOT s1 AND s0) OR
         (c AND s1 AND NOT s0) OR
         (d AND s1 AND s0);
END pure_logic;
---------------------------------------------
```



WHEN (simple)

WHEN is one of the fundamental concurrent statements.

It appears in two forms: simple and selected

Let's start with the syntax for simple WHEN:

```
assignment WHEN condition ELSE
assignment WHEN condition ELSE
...;
```

```
------ With WHEN/ELSE -------------------------
outp <= "000" WHEN (inp='0' OR reset='1') ELSE
        "001" WHEN ctl='1' ELSE
        "010";
```

this is obvious not the complete VHDL code (ie : entity and architecture are missing)

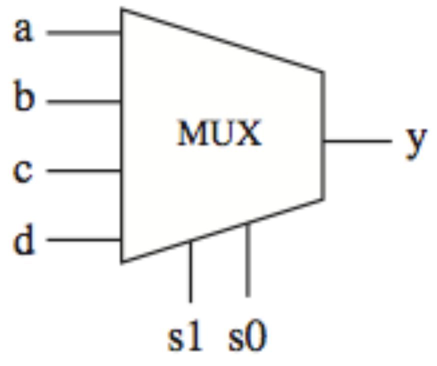
Three forms of WHEN value

Another important aspect related to the WHEN statement is that the “WHEN value” shown in the syntax above can indeed take up three forms:

```
WHEN value                  -- single value
WHEN value1 to value2       -- range, for enumerated data types
                            -- only
WHEN value1 | value2 |...   -- value1 or value2 or ...
```

4-to-1 MUX using WHEN (simple)

- Much simpler than the previous implementation with logic operators.
- The output will be identical



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
          y: OUT STD_LOGIC);
END mux;
---------------------------------------------
ARCHITECTURE mux1 OF mux IS
BEGIN
   y <=  a WHEN sel="00" ELSE
         b WHEN sel="01" ELSE
         c WHEN sel="10" ELSE
         d;
END mux1;
---------------------------------------------
```

WHEN (selected)

WHEN with the selected form is very similar to the simple WHEN.

Here is its syntax:

```
WITH identifier SELECT
assignment WHEN value,
assignment WHEN value,
...;
```

UNAFFECTED is a very useful keyword

```
---- With WITH/SELECT/WHEN --------------------
WITH control SELECT
   output <= "000" WHEN reset,
             "111" WHEN set,
             UNAFFECTED WHEN OTHERS;
-----------------------------------------------
```

4-to-1 MUX using WHEN (selected)

- The form for simple or selected WHEN, are very similar and will yield the same outcome.

WHEN (simple)

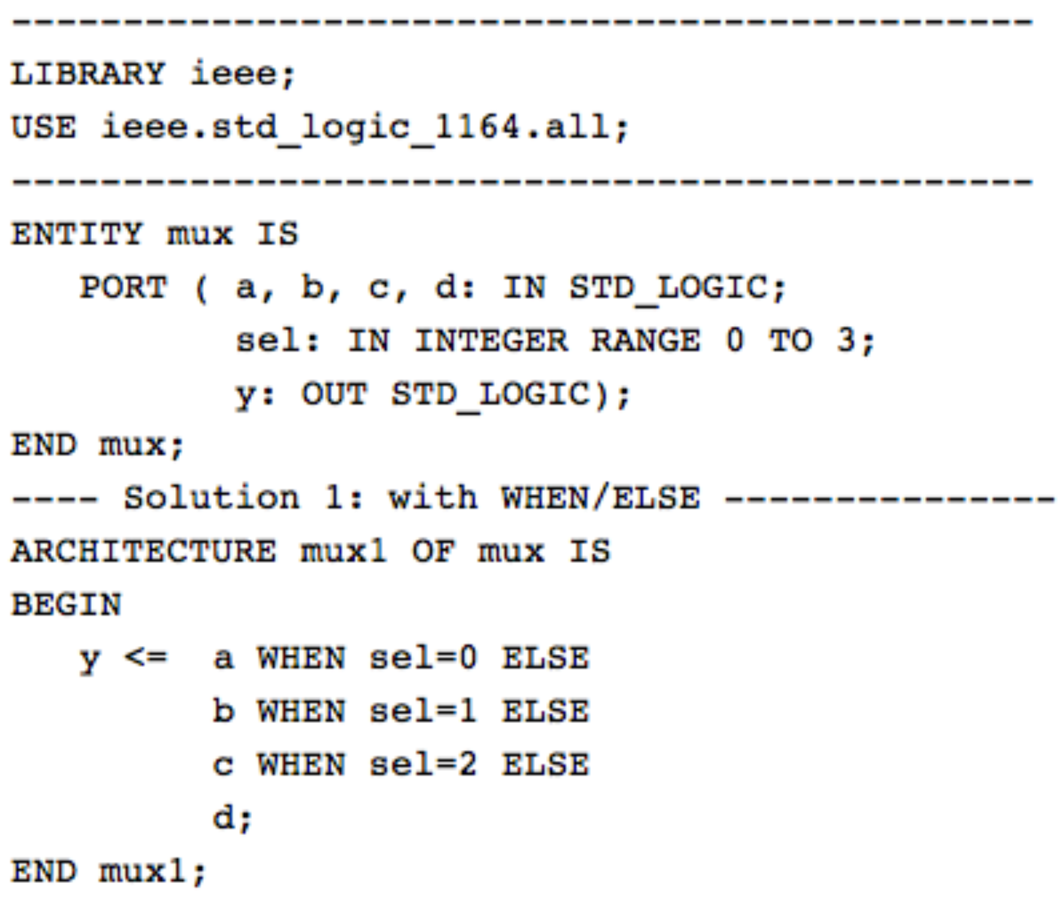
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
          y: OUT STD_LOGIC);
END mux;
-------------------------------------------
ARCHITECTURE mux1 OF mux IS
BEGIN
   y <=  a WHEN sel="00" ELSE
         b WHEN sel="01" ELSE
         c WHEN sel="10" ELSE
         d;
END mux1;
-------------------------------------------
```

WHEN (selected)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
          y: OUT STD_LOGIC);
END mux;
-------------------------------------------
ARCHITECTURE mux2 OF mux IS
BEGIN
   WITH sel SELECT
      y <= a WHEN "00",       -- notice "," instead of ";"
           b WHEN "01",
           c WHEN "10",
           d WHEN OTHERS;     -- cannot be "d WHEN "11" "
END mux2;
--------------------------------------------
```

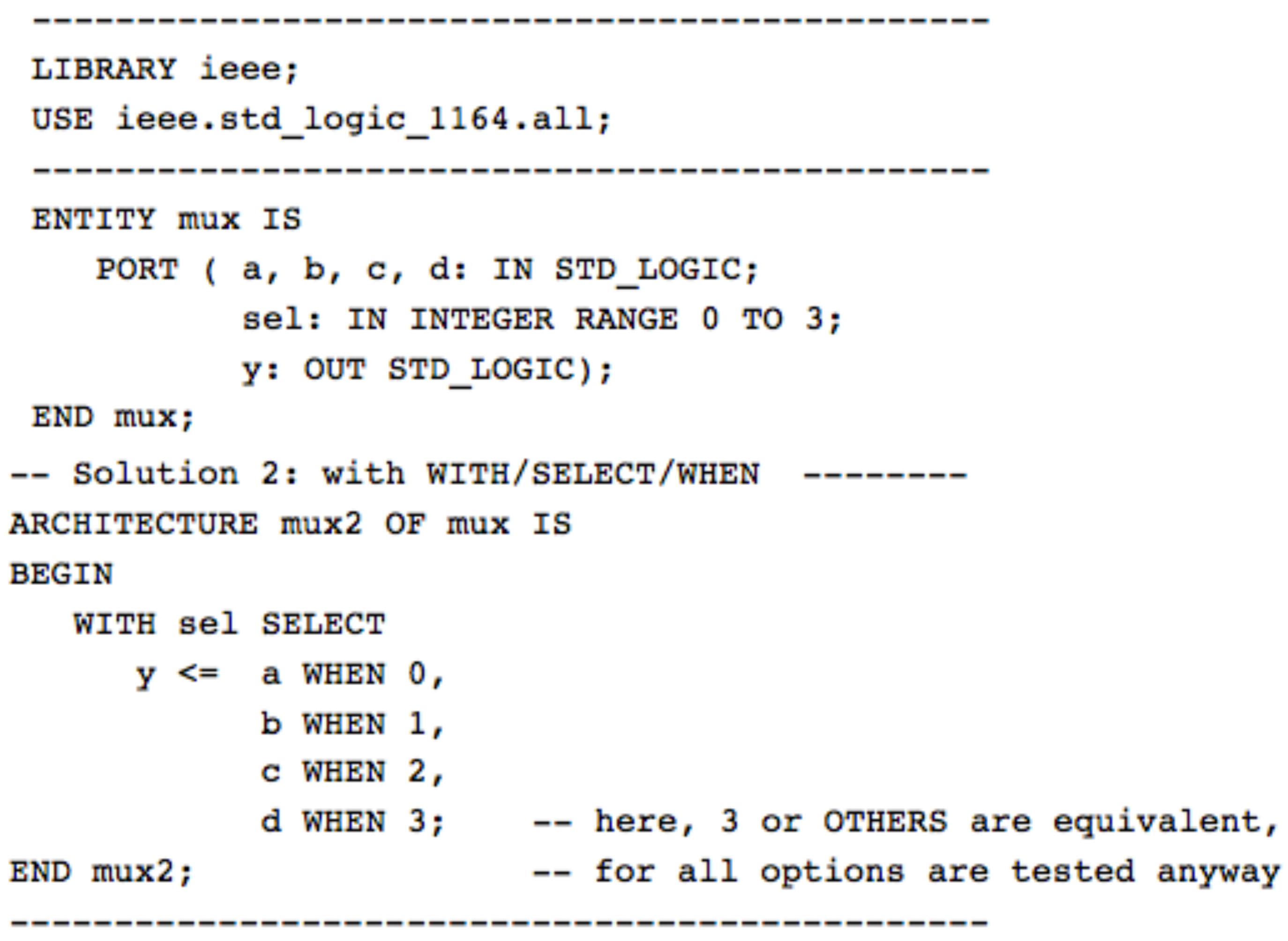
Replacing sel (input) with INTEGER

WHEN (simple)



```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
          sel: IN INTEGER RANGE 0 TO 3;
          y: OUT STD_LOGIC);
END mux;
---- Solution 1: with WHEN/ELSE ---------------
ARCHITECTURE mux1 OF mux IS
BEGIN
   y <=  a WHEN sel=0 ELSE
         b WHEN sel=1 ELSE
         c WHEN sel=2 ELSE
         d;
END mux1;
```

WHEN (selected)

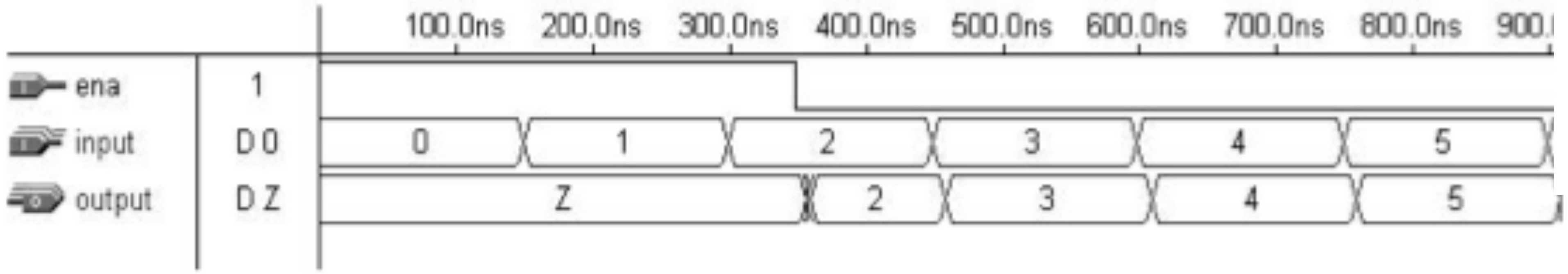


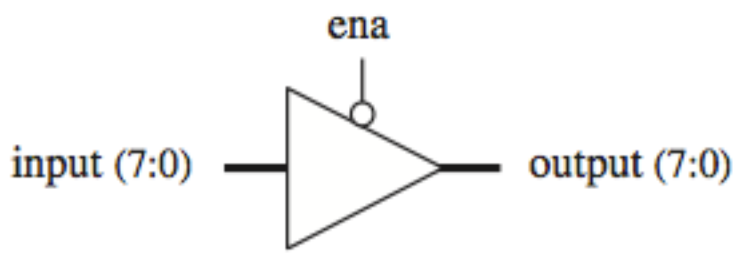
```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
          sel: IN INTEGER RANGE 0 TO 3;
          y: OUT STD_LOGIC);
END mux;
-- Solution 2: with WITH/SELECT/WHEN  --------
ARCHITECTURE mux2 OF mux IS
BEGIN
   WITH sel SELECT
      y <=  a WHEN 0,
            b WHEN 1,
            c WHEN 2,
            d WHEN 3;    -- here, 3 or OTHERS are equivalent,
END mux2;                -- for all options are tested anyway
---------------------------------------------
```

Another example: tri-state buffer

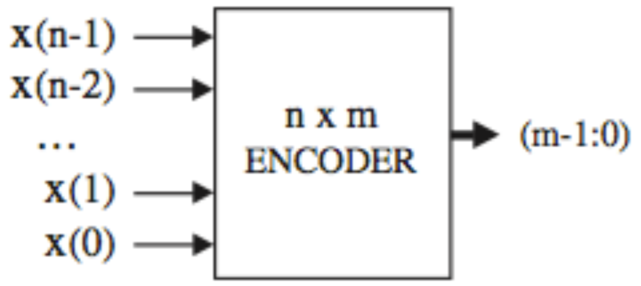
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------------
ENTITY tri_state IS
   PORT ( ena: IN STD_LOGIC;
          input: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END tri_state;
----------------------------------------------
ARCHITECTURE tri_state OF tri_state IS
BEGIN
   output <= input WHEN (ena='0') ELSE
             (OTHERS => 'Z');
END tri_state;
----------------------------------------------
```

- This is another example that illustrates the use of WHEN.
- The 3-state buffer must provide output = input when ena (enable) is low, or output = "ZZZZZZZZ" (high impedance) otherwise.





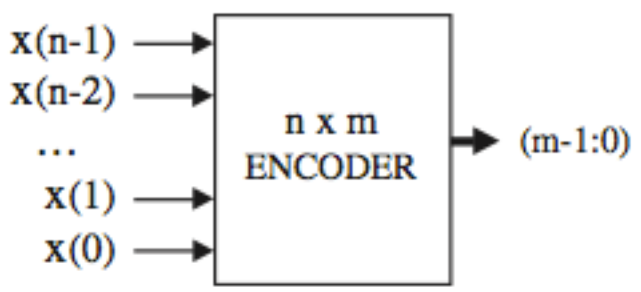
Another example : decoder with with simple WHEN



- We assume that n is a power of two, so m=log2(n)
- For simplicity, lets assign n=3
- One and only one input bit is expected to be high at a time, whose address must be encoded at the output

```
---- Solution 1: with WHEN/ELSE -------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY encoder IS
   PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END encoder;
---------------------------------------------
ARCHITECTURE encoder1 OF encoder IS
BEGIN
   y <=   "000" WHEN x="00000001" ELSE
          "001" WHEN x="00000010" ELSE
          "010" WHEN x="00000100" ELSE
          "011" WHEN x="00001000" ELSE
          "100" WHEN x="00010000" ELSE
          "101" WHEN x="00100000" ELSE
          "110" WHEN x="01000000" ELSE
          "111" WHEN x="10000000" ELSE
          "ZZZ";
END encoder1;
---------------------------------------------
```

Another example : decoder with selected WHEN



- We assume that n is a power of two, so m=log2(n)
- For simplicity, lets assign n=3
- One and only one input bit is expected to be high at a time, whose address must be encoded at the output

```
---- Solution 2: with WITH/SELECT/WHEN ------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY encoder IS
   PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END encoder;
---------------------------------------------
ARCHITECTURE encoder2 OF encoder IS
BEGIN
   WITH x SELECT
      y <=   "000" WHEN "00000001",
             "001" WHEN "00000010",
             "010" WHEN "00000100",
             "011" WHEN "00001000",
             "100" WHEN "00010000",
             "101" WHEN "00100000",
             "110" WHEN "01000000",
             "111" WHEN "10000000",
             "ZZZ" WHEN OTHERS;
END encoder2;
---------------------------------------------
```