INTRODUCTION TO VHDL

Aniseh Dorostkar

VHDL Module

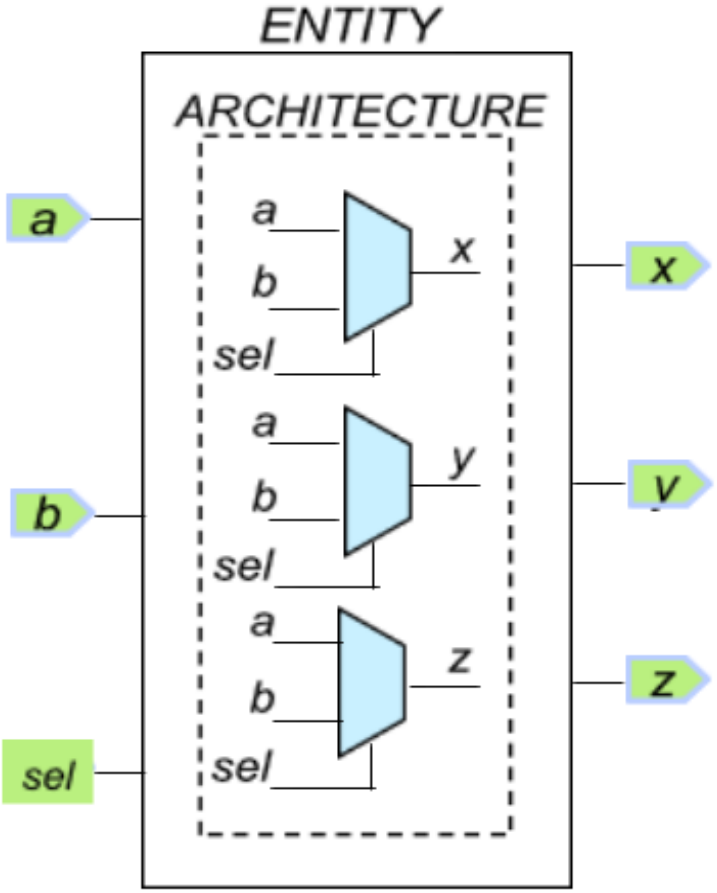
Libraries

Entity

Architecture

Configuration

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY cmpl_sig IS
   PORT (
      a, b, sel : IN BIT;
      x, y, z : OUT BIT
      );
END ENTITY cmpl_sig;
ARCHITECTURE logic OF cmpl_sig IS
BEGIN
   -- simple signal assignment
   x <= (a AND NOT sel) OR (b AND sel);

   -- conditional signal assignment
   y <= a WHEN sel='0' ELSE b;

   -- selected signal assignment
   WITH sel SELECT
      z <= a WHEN '0',
           b WHEN '1',
           0' WHEN OTHERS;
END ARCHITECTURE logic;
CONFIGURATION cmpl_sig_conf OF cmpl_sig IS
         FOR logic
         END FOR;
END CONFIGURATION cmpl_sig_conf;
```



VHDL Libraries

- library IEEE;
- use IEEE.std_logic_1164.all;
  - *std_logic*
    - Single-bit signals
  - *std_logic_vector*
    - Multi-bit signals
- use IEEE.std_logic_unsigned.all;
  - *Overloaded operators*
  - *Conversion functions*

Entity Declaration

```
ENTITY <entity_name> IS
    Generic declarations
    Port Declarations
END ENTITY <entity_name> ; (1076-1993 version)
```

```
ENTITY  <entity_name>  IS
   GENERIC (
      CONSTANT tplh , tphl : time := 5 ns;
      -- Note constant is assumed and is not required
      tphz, tplz : TIME := 3 ns;
      default_value : INTEGER := 1;
      cnt_dir : STRING := "up"
      );
   Port Declarations
END ENTITY <entity_name>;
```

Generic Declaration

```
ENTITY  <entity_name>  IS
   Generic Declarations
   PORT (
      SIGNAL clk, clr : IN  BIT;
      --Note: SIGNAL is assumed and is not required
      q : OUT BIT
      );
END ENTITY <entity_name> ;
```

Port Declaration

Architecture

```
ARCHITECTURE <identifier> OF <entity_identifier> IS
--Architecture declaration section (Three example declarations below;
--    examples of other possible declaration types not shown)
    SIGNAL temp : INTEGER := 1;  -- signal declarations :=1 is default value optional
    CONSTANT load : BOOLEAN := true;  --constant declarations
    TYPE states IS ( S1, S2, S3, S4 ) ;   --type declarations
    --Component declarations (discussed later)
    --Subtype declarations (discussed later)
    --Attribute declarations
    --Attribute specifications
    --Subprogram declarations
    --Subprogram body
BEGIN
   Process statements
   Concurrent procedural calls
   Concurrent signal assignment
   Component instantiation statements
   Generate statements
END ARCHITECTURE <identifier>;
```

VHDL Data Types

- Bit
  - *2 logic value system ('0' or '1')*
    - SIGNAL a : BIT;
- Bit_Vector
  - *Array of bits "00", "01", "10", ...*
  - *SIGNAL temp: BIT_VECTOR (3 DOWNTO 0);*
  - *SIGNAL temp : BIT_VECTOR (0 TO 3);*
- Boolean
  - *FALSE or TRUE*
- Time
  - *integer includes units of time : fs, ps, ns, us, ms, ...*
- Integer
  - *Possitive and negative values in decimal*
  - *SIGNAL int_temp : INTEGER ; -- 32-bits number*
  - *SIGNAL int_tmp: INTEGER 0 RANGE 255 ; -- 8 bits number*
- Positive
  - *Integer with range 0 to 2^{32}*
- Natural
  - *Integer with range 1 to 2^{32}*
- Real
  - *Double-precision floating point numbers*
- Character
  - *'a', 'b', '1', '2', ...*
- *String*
  - Array of Characters
- Enumeration Type
  - *User defined*
  - *Type State IS {s0, s1, s2};*

IEEE 1164 Standard Logic

- **Type STD_LOGIC**
  - *9-Valued Logic System*
    - 'U' Uninitialized
    - 'X' Forcing Unknown
    - '0' Forcing 0
    - '1' Forcing 1
    - 'Z' High Impedance
    - 'W' Weak Unknown
    - 'L' Weak 0
    - 'H' Weak 1
    - '-' Don't Care

VHDL Operators

Operator Type		Operator Name/Symbol	Priority
Logical		NOT AND OR NAND NOR XOR XNOR(1)	Low
Relational		= /= < <= > >=	
Shifting(1) (2)		SLL SRL SLA SRA ROL ROR	
Arithmetic	Addition & Sign	+ -	
	Concatenation	&	
	Multiplication	* / MOD REM	
Miscellaneous		** ABS	High

** = *exponentiation*
abs = absolute value

(1) *Not supported in VHDL '87*
(2) *Supported in NUMERIC_STD package for SIGNED/UNSIGNED data types*

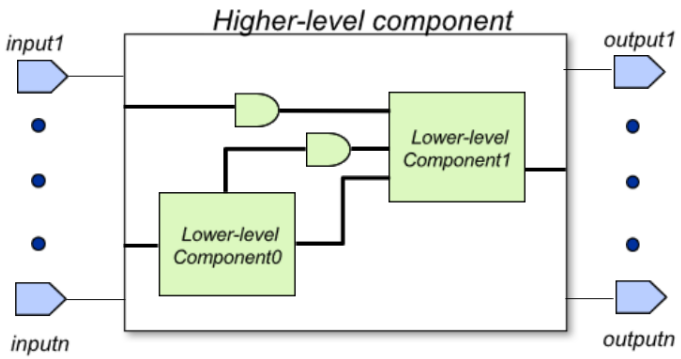
Component Declaration

- Component Declaration

```
COMPONENT <lower-level_design_name>
   PORT (
      <port_name>  :  <port_type> <data_type>;
                      .
                      .
      <port_name>  :  <port_type> <data_type>
      );
      END COMPONENT;
```

- Component Instantiation

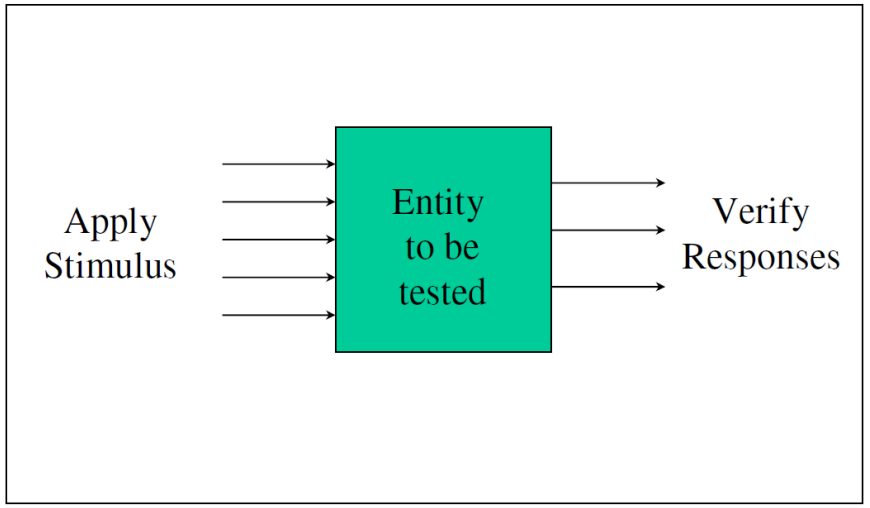
```
<instance_name> : <lower-level_design_name>
   PORT MAP(<lower-level_port_name>  => <current_level_port_name>,
              …,
            <lower-level_port_name> => <current_level_port_name>
            );
```



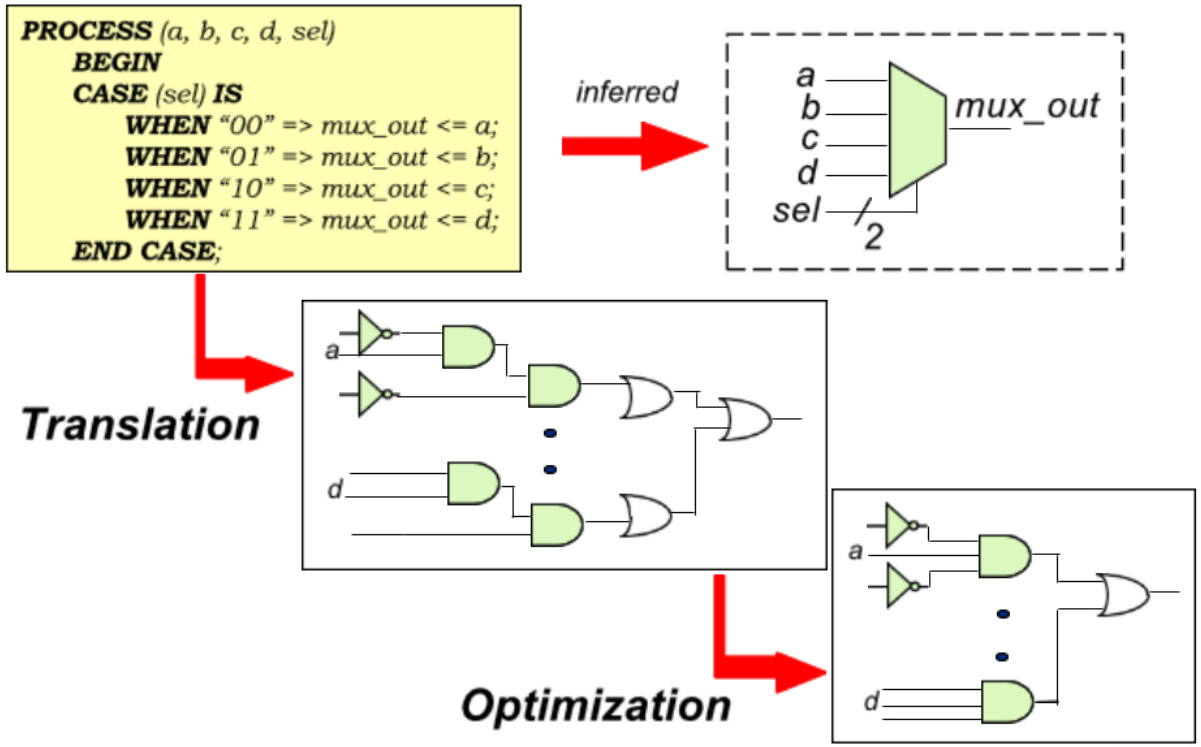
TestBench

- To simulate your design you need to produce an additional ENTITY and ARCHITECTURE design.
  - Usually referred to as a TEST BENCH
  - Not hardware, just additional VHDL!

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   --Entity of TestBench
5   entity decoder_tb is
6   end entity decoder_tb;
7
8   architecture test of decoder_tb is
9
10  -- Component Declaration for the Unit Under Test (UUT)
11  component Decoder_vec is
12  port(
13  input: in std_logic_vector(2 downto 0);
14  output: out std_logic_vector(7 downto 0)
15  );
16  end component;
17
18  --Inputs
19  signal i :std_logic_vector(2 downto 0);
20  --Outputs
21  signal o : std_logic_vector(7 downto 0);
22
23  begin
24  -- Instantiate the Unit Under Test (UUT)
25  dec:Decoder_vec port map ( input => i, output => o);
26
27
28  i <= "000", "001" after 100 ns, "101" after 200 ns;
29  end test;
```



Synthesis


PROCESS (a, b, c, d, sel)
BEGIN
CASE (sel) IS
WHEN "00" => mux_out <= a;
WHEN "01" => mux_out <= b;
WHEN "10" => mux_out <= c;
WHEN "11" => mux_out <= d;
END CASE;
inferred
a
b
c
d
sel
2
mux_out
Translation
a
d
Optimization
a
d

Sequential Statements
Process

Processes are Synthesizable

Syntax:

```
[<process_name>:] process (<sensitive
signals>)
variable declarations
constant declarations
...
Begin

Statements
...
end process;
```

Example:

```
output_process: process(flag_signal)
begin

if flag_signal = '1' then
output_vector <= "010;"
else
output_vector <= "101;"
end if;
end process;
```

Sequential Statements
If statement

Syntax:

```
if <condition> then
statements
...
[
elsif <condition> then
statements
...
Else
Statements
...
]
endif;
```

Examples:

```
if boolean_v then
output_1 <= '1';
end if

if condition_v_1 = '1' then
out_vector <= "001"
elsif condition_v_2 = '1' then
out_vector <= "110"
...

Else
out_vector <= "000"
end if;
```

The **if** statement is generally Synthesizable.

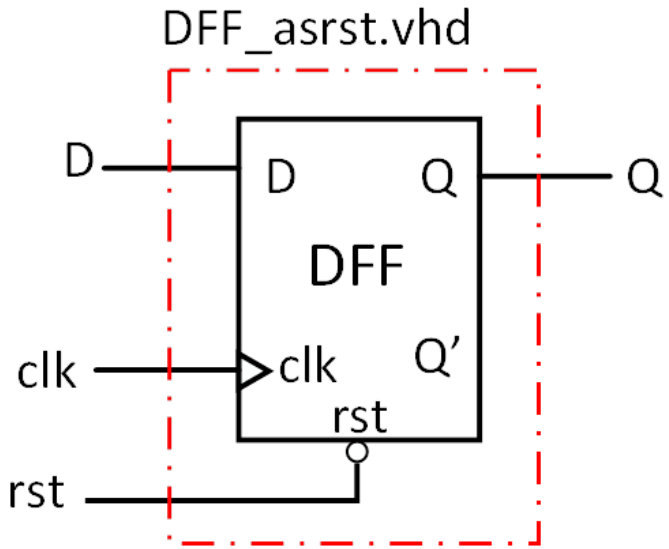
Positive edge-triggered D flip-flop.

DFF with Asynchronous clear

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY dff_aclr IS
   PORT (
      d, clk, clr : IN std_logic;
      q : OUT std_logic
      );
END ENTITY dff_aclr;
ARCHITECTURE rtl OF dff_aclr IS
BEGIN
   PROCESS (clk, clr)
   BEGIN
      IF clr = '0' THEN
         q <= '0';
      ELSIF rising_edge(clk) THEN
         q <= d;
      END IF;
   END PROCESS;
END ARCHITECTURE rtl;
```

DFF with Synchronous clear

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY dff_sclr IS
   PORT (
      d, clk, clr : IN std_logic;
      q : OUT std_logic
      );
END ENTITY dff_sclr;
ARCHITECTURE rtl OF dff_sclr IS
BEGIN
   PROCESS (clk)
   BEGIN
      IF rising_edge (clk) THEN
         IF clr = '0' THEN
            q <= '0';
         ELSE
            q <= d;
         END IF;
      END IF;
   END PROCESS;
END ARCHITECTURE rtl;
```



Falling_edge
Rising_edge
(clk'event and "clk==1")
(clk'event and "clk==0")

Sequential Statements
Case Statements

Syntax:

```
case <expression> is
when <choice(s)> =>
    <expression>;
 ...
when ...
[when others => ... ]
end case;
```

The CASE statement is generally Synthesizable

Examples:

```
case scancode is
when x"14" =>
    integer_signal <= 1;
when x"18" =>
    integer_signal <= 2;
when x"19" | x"20" | x"21" =>
    integer_signal <= 3;
when others =>
    integer_signal <= 0;
end case;
```

Sequential Statements
For loop Statements

- **Syntax:**

```
for parameter in range loop
        sequential
statements;
        …
end loop;
```

- **Example**

```
process (A)
Begin
        Z <= "0000";
        for i in o to 3 loop
                if (A = i) then
                        Z(i) <= '1';
                end if;
        end loop;
end process;
```

The for loop is supported for synthesis, providing:

1. the loop range is static (i.e. implies a definite number of iterations), and
2. the loop contains no **wait** statements.

Sequential Statements
while loop Statements

Syntax:

```
while condition loop
      sequential statements;
      …
end loop;
```

While is supported by some logic synthesis tools, with certain restrictions

Example

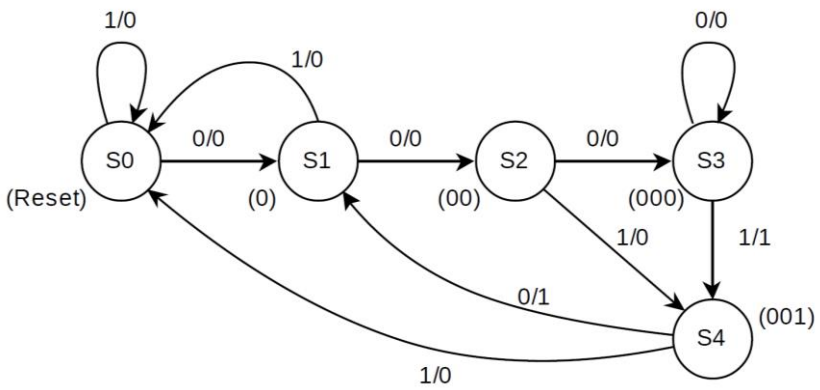
```
process (A)
    variable I :integer range 0 to 4;
begin
    Z <= "0000";
    I := 0;
    while (I <= 3) loop
      if (A = I) then
        Z(I) <= '1';
      end if;
      I := I + 1;
end loop;
end process;
```

Mealy State Machine to detect 0010 or 0001

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   entity mealy_detector is
4   port(
5   input : in std_logic ;
6   detected : out std_logic;
7   clk : in std_logic;
8   rst : in std_logic
9   );
10  end mealy_detector;
11
12  architecture mealy_behav of mealy_detector is
13  type states is (s0, s1 , s2, s3,s4);
14  signal present_state : states := s0;
15  signal next_state : states := s0;
16  begin
17
18  clock_process : process(clk)
19  begin
20          if (clk'event and clk = '1') then
21                  if (rst = '0') then
22                          present_state <= s0;
23                  else
24                  present_state <= next_state;
25          end if;
26          end if;
27  end process clock_process;
```

```
31  mealy_process : process(present_state , input)
32  begin
33          case present_state is
34                  when s0=>
35                          if (input = '1') then
36                                  next_state <= s0;
37                                  detected <= '0';
38                          else
39                                  next_state <= s1 ;
40                                  detected <= '0';
41                          end if;
42                  when s1=>
43                          if(input = '1') then
44                                  next_state <= s0;
45                                  detected <= '0';
46                          else
47                                  next_state <= s2;
48                                  detected <= '0';
49                          end if;
50                  when s2=>
51                          if(input = '1') then
52                                  next_state <= s4;
53                                  detected <= '0';
54                          else
55                                  next_state <= s3;
56                                  detected <= '0';
57                          end if;
58                  ...
59
60                  when others=>
61                          next_state <= s0;
62                          detected <= '0';
63                  end case;
64  end process mealy_process;
65  end mealy_behav;
```

P_state	Next state		detected	
	X=0	X=1	X=0	X=1
s0	s1	s0	0	0
s1	s2	s0	0	0
s2	s3	s4	0	0
s3	s3	s4	0	1
S4	s1	s0	1	0

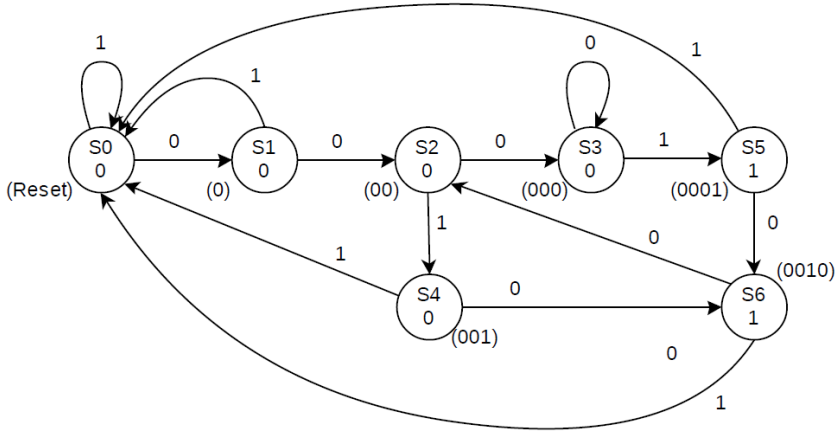


Moore State Machine to detect 0010 or 0001

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   entity moore_1101_detector is
4   port(
5   input : in std_logic ;
6   clk : in std_logic;
7   rst : in std_logic;
8   detected : out std_logic
9   );
10  end moore_1101_detector;
11
12  architecture moore_1101_behav of moore_1101_detector is
13  type states is (s0, s1 , s2, s3, s4, s5, s6);
14  signal present_state : states := s0;
15  signal next_state : states := s0;
16  begin
17
18  clock_process : process(clk)
19  begin
20          if (clk'event and clk = '1') then
21                  if (rst = '0') then
22                          present_state <= s0;
23                  else
24                  present_state <= next_state;
25          end if;
26          end if;
27  end process clock_process;
```

```
31  moore_process : process(present_state , input)
32  begin
33          case present_state is
34                  when s0=>
35                          if (input = '0') then
36                                  next_state <= s1;
37                          else
38                                  next_state <= s0 ;
39                          end if;
40                  when s1=>
41                          if(input = '0') then
42                                  next_state <= s2;
43                          else
44                                  next_state <= s0 ;
45                          end if;
46                  when s2=>
47                          if(input = '0') then
48                                  next_state <= s3;
49                          else
50                                  next_state <= s4;
51                          end if;
52                  ...
53
54                  when others=>
55                          next_state <= s(
56                  end case;
57  end process moore_process;
58
59
60  with present_state select
61  detected <= '1' when s5,
62          '1' when s6,
63          '0' when others;
64
65  end moore_1101_behav;
```

P_state	Next state		detected
	X=0	X=1	
s0	s1	01	0
s1	s2	s0	0
s2	s4	s0	0
s3	s3	s5	0
s4	s0	s6	0
s5	s6	s0	1
s6	s2	s0	1

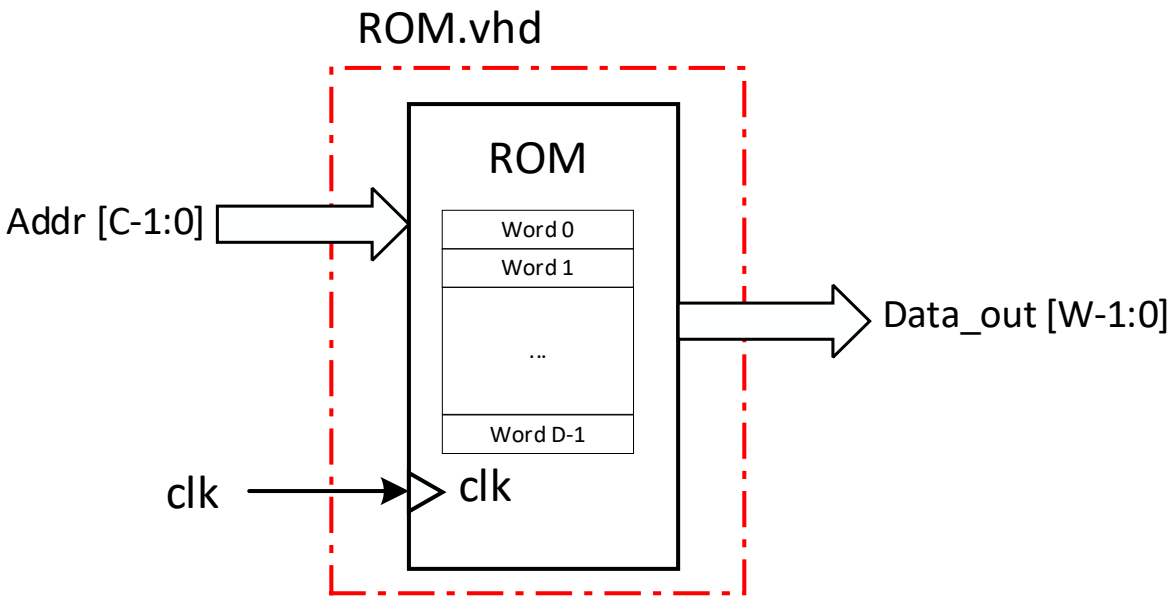


TestBench of sequential circuit

```
USE ieee.std_logic_1164.ALL;

ENTITY tb_moore_1101_detector IS
END tb_moore_1101_detector;

architecture test of tb_moore_1101_detector is
component moore_1101_detector is
port (
        input, clk, rst : in std_logic;
        detected : out std_logic
);
end component;
signal x_in : std_logic := '0';
signal clk_in : std_logic := '0';
signal rst_in : std_logic := '0';
signal detect : std_logic;
constant clk_period : time := 100 ns;
begin

uut: moore_1101_detector port map (input => x_in, clk => clk_in, rst => rst_in, detected => detect);

   clk_process : process
   begin
                clk_in <= '0';
                wait for clk_period/2;
                clk_in <= '1';
                wait for clk_period/2;
   end process;

   stim_proc: process
   begin
                rst_in <='0';
                wait for clk_period;
                rst_in <='1';
                wait for clk_period;
                --0001101010110111011
                x_in<= '0';
                wait for clk_period;
                x_in<= '0';
                wait for clk_period;
                ...
   end process;
end test;
```

ROM in VHDL

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE IEEE.std_logic_arith.all;
--USE IEEE.numeric_std.ALL;
entity ROM2 is
generic (
W1 : integer := 8;  -- number of word bit
D: integer := 4;   -- address bit
C: integer := 16   -- number of word
);
port (
        clk : in std_logic;
        addr   : in std_logic_vector(D-1 downto 0);
        data_out  : out std_logic_vector(W1-1 downto 0));
end ROM2;
architecture ROM_arch of ROM2 is
type mem_type is array (C-1 downto 0) of std_logic_vector (W1-1 downto 0);
constant ROM_block : mem_type := ( "00111000",
                "00000001",
                "00000010",
                "00000011",
                "00000100",
                "00000101",
                "00000110",
                "00000111",
                "00001000",
                "00001001",
                "00001010",
                "00001011",
                "00001100",
                "00001101",
                "00001110",
                "00001111");
begin
process (clk)
begin
        if (rising_edge(clk)) then
                data_out <= ROM_block(conv_integer(addr));
end if;
end process;
end architecture ROM_arch;
```



Concatenate signals

- Signal S: std_logic_vector (3 downto 0) := “1010”;
- Signal C : std_logic := ‘1’;
- A_1 <= ‘0’ & C & S(3 downto 1) & “01”; -- A <= “0110101”

Convertion Data type

- The **library:numeric_std** package in the **ieee** library