Combinational network examples in VHDL

Tutorial 04 on Dedicated systems

Teacher: Giuseppe Scollo

University of Catania
Department of Mathematics and Computer Science
Graduate Course in Computer Science, 2019-20

Table of Contents

this tutorial deals with VHDL descriptions of frequently used hardware components:

- tri-state buffers
- decoders:
  - *n*-input decoders
  - 7-segment display decoders
- multiplexers
- ALU functions

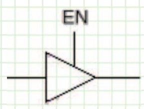
various aspects and constructs of the VHDL language are introduced in the context of the proposed examples

furthermore, a lab experience is proposed which collects various aspects of VHDL that are illustrated by the examples presented here, and where some of these components may be reused for the implementation of the proposed experiment

the standard, two-valued boolean type BIT, does not suffice to represent all situations that may come into play in circuit design

e.g. a circuit line may need be disconnected dynamically, in order to allow bus access by multiple drivers, one at a time

tri-state gates are typically utilized to this purpose



tri-state buffer

the IEEE 1164 standard defines the nine-valued std_ulogic type:

‘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

yet std_ulogic does not allow multiple driver access to the same line

its subtype std_logic is endowed with a resolution function to resolve the contention, which is allowed thus

	U	X	0	1	Z	W	L	H	–
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
–	U	X	X	X	X	X	X	X	X

contention resolution between values of type std_logic

```
library ieee;
use ieee.std_logic_1164.all;
entity tri_state is
  port (x, en : in std_logic;
        y : out std_logic);
end entity tri_state;
architecture when_else of tri_state is
begin
  y <= x when en = '1' else 'Z';
end architecture when_else;
```

functional unit that proves useful to diverse purposes, for example:

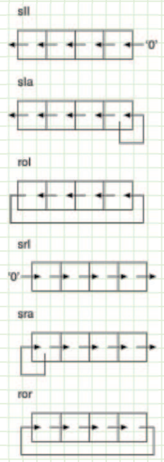
- memory chip/row selection
- decoding of a machine instruction opcode

VHDL specification of the 2→4 decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder2 is
  port (
    a : in std_logic_vector(1 downto 0);
    z : out std_logic_vector(3 downto 0)
  );
end entity decoder2;

architecture when_else of decoder2 is
begin
  z <= "0001" when a = "00" else
       "0010" when a = "01" else
       "0100" when a = "10" else
       "1000" when a = "11" else
       "XXXX";
end architecture when_else;
```

generic VHDL specification of the decoder (Zwoliński, Sect. 4.2.3):

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity decoder is
  generic (n : POSITIVE);
  port (
    a : in std_logic_vector(n-1 downto 0);
    z : out std_logic_vector(2**n-1 downto 0)
  );
end entity decoder;

architecture rotate of decoder is
  constant z_out : BIT_VECTOR(2**n-1 downto
0) :=
    (0 => '1', others => '0');
begin
  z <= to_StdLogicVector (z_out sll
    to_integer(unsigned(a)));
end architecture rotate;
```



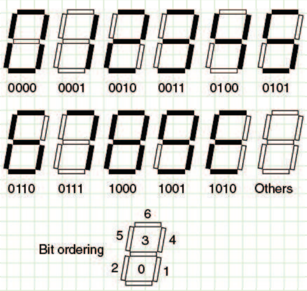
VHDL shift operators

familiar device of daily use ...

a decoder for a seven-segment display converts a 4-bit input to the 7-bit configuration of the display segment states that visualizes the character corrisponding to the (hex or decimal) digit which has the value coded by the input

the following example (taken from Zwoliński, Sect. 4.2.3)

- visualizes decimal digits
- visualizes E if the input value is ≥ 10
- in all other cases the display is blanked



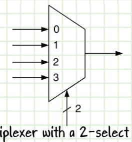
Zwoliński, Figure 4.3 - Seven-segment display

```
library ieee;
use ieee.std_logic_1164.all;

entity seven_seg is
  port (a : in std_logic_vector(3 downto 0);
        z : out std_logic_vector(6 downto 0));
end entity seven_seg;

architecture with_select of seven_seg is
begin
  with a select
    z <= "1110111" when "0000",
         "0010010" when "0001",
         "1011101" when "0010",
         "1011011" when "0011",
         "0111010" when "0100",
         "1101011" when "0101",
         "1101111" when "0110",
         "1010010" when "0111",
         "1111111" when "1000",
         "1111011" when "1001",
         "1101101" when "1010"|"1011"|"1100"|"1101"|"1110"|"1111",
         "0000000" when others;
end architecture with_select;
```

multiplexers

VHDL description of a multiplexer with a single select input:

```
library ieee;
use ieee.std_logic_1164.all;

entity mux1 is
  port (
    s : in std_logic;
    a : in std_logic;
    b : in std_logic;
    y : out std_logic
  );
end entity mux1;

architecture basic of mux1 is
begin
  y <= a when s = '0' else
       b when s = '1' else
       'X';
end architecture basic;
```



multiplexer with a 2-select input

how to generalize to *n*-select input?

description of an *n*-input OR gate:

```
library ieee;
use ieee.std_logic_1164.all;
entity wide_or is
  generic (n : POSITIVE);
  port (
       x : in std_logic_vector(n-1 downto 0);
       y : out std_logic);
end entity wide_or;
```

***n*-select input multiplexer**

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
  generic (n : POSITIVE := 1);
  port (
    s : in std_logic_vector(n-1 downto 0);
    a : in std_logic_vector(2**n-1 downto 0);
    z : out std_logic);
end entity mux;
```

instantiation of the generic multiplexer

```
library ieee;
use ieee.std_logic_1164.all;
entity mux2 is
  port (
    s : in std_logic_vector(1 downto 0);
    a : in std_logic_vector(3 downto 0);
    z : out std_logic);
end entity mux2;
```

```
architecture sequential of wide_or is
begin
  process (x) is
    variable z : std_logic;
  begin
    z := x(0);
    if n > 1 then
      for i in 1 to n-1 loop
        z := x(i) or z;
      end loop;
    end if;
    y <= z;
  end process;
end architecture sequential;

architecture structural of mux is
  signal decout : std_logic_vector(2**n-1 downto 0);
  signal andout : std_logic_vector(2**n-1 downto 0);
begin
  di: entity WORK.decoder generic map (n)
       port map (a => s, z => decout);
  oi: entity WORK.wide_or generic map (2**n)
       port map (x => andout, y => z);
  andout <= a and decout;
end architecture structural;

architecture instance of mux2 is
begin
  di: entity WORK.mux generic map (2)
       port map (s, a, z);
end architecture instance;
```


ALU functions

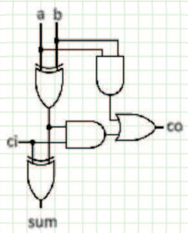
an ALU is a multifunction unit: a control input selects the operation to be executed

an example of multifunction logic unit has the selection as specified in the table and may be described in VHDL as follows:

S	function
00	Q <= A or B
01	Q <= A and B
10	Q <= not B
11	Q <= A xor B

```
library ieee;
use ieee.std_logic_1164.all;
entity alu_logic is
  generic ( n : POSITIVE := 16 );
  port (
    a : in std_logic_vector((n−1) downto 0);
    b : in std_logic_vector((n−1) downto 0);
    s : in std_logic_vector(1 downto 0);
    q : out std_logic_vector((n−1) downto 0)
  );
end entity alu_logic;

architecture with_select of alu_logic is
  constant undefined : std_logic_vector(n-1 downto 0) := (others => 'X');
begin
  with s select
    q <= a or b    when "00",
         a and b when "01",
         not b;    when "10",
         a xor b  when "11",
         undefined when others;
end architecture with_select;
```

for operands wider than one bit, ALU's arithmetic function may be specified in either structural or functional style; the latter as follows



1-bit full-adder

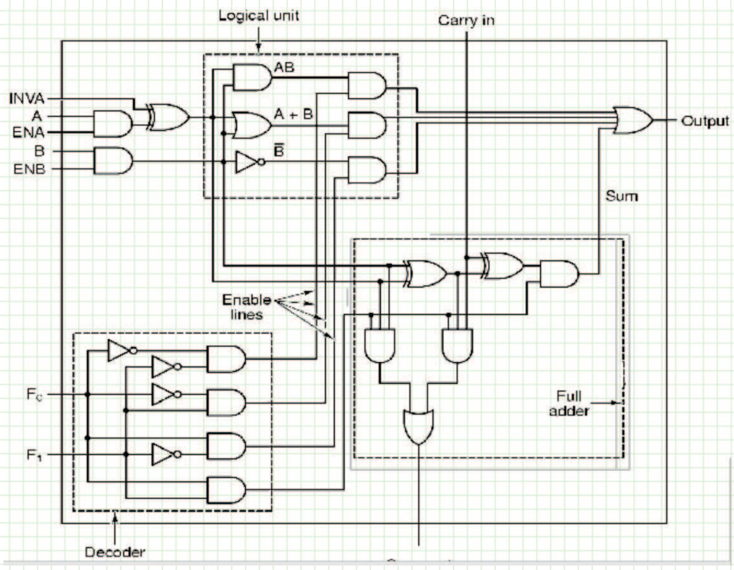
```
library ieee;
use IEEE.std_logic_1164.all;
entity adder is
  generic(n-1 : POSITIVE := 16);
  port (
    a : in std_logic_vector (n-1 downto 0);
    b : in std_logic_vector (n-1 downto 0);
    ci : in std_logic;
    sum : out std_logic_vector (n-1 downto 0);
    co : out std_logic
  );
end entity adder;

architecture sequential of adder is
begin
  process(a,b,ci)
    variable carry : std_logic;
    variable psum : std_logic_vector(n-1 downto 0);
  begin
    carry := ci;
    for i in 0 to n-1 loop
      psum(i) := a(i) xor b(i) xor carry;
      carry := (a(i) and b(i)) or (((a(i) xor b(i)) and carry);
    end loop;
    sum <= psum;
    co <= carry;
  end process;
end architecture sequential;
```

the 8-bit ALU simulator implemented by S. Lentini and G. Nicotra illustrates how one can iteratively construct the 8-bit ALU by suitably composing 1-bit ALU stages

the schematic and an animated illustration of the ALU functions are presented in the "Arithmetic logic unit" part of the Flash animation of that project

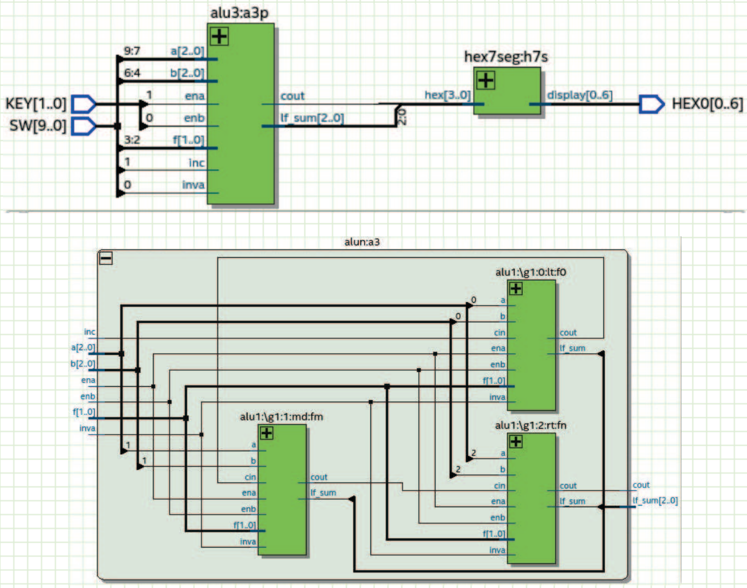
using the VHDL constructs seen in the examples through this tutorial:

1. build a VHDL model of the 1-bit ALU; an equivalent solution to that which is represented by the schematic reproduced next, may be obtained by appropriate use of a multiplexer in place of a decoder
2. building on this result, construct a generic model of the *n*-bit ALU
3. compose a 3-bit instance of the generic ALU model with a VHDL model of the hex to 7-segment display decoder (available in the reserved lab area), mapping the I/O ports of the obtained model to pins of the DE1-SoC FPGA, as specified next and further illustrated in the subsequent RTL schematic
4. program the FPGA by using its 10 switches and 2 keys for the ALU input (the 2 keys for the ENA, ENB inputs, 6 switches for the A, B operands, the remaining switches for the other control inputs) and a 7-segment display for the visualization of the result (carry included, for addition)
5. test the functioning of the 3-bit ALU on the FPGA by acting on the switches and assigned keys



schematic of a 1-bit ALU (Figure 3-19 in (Tanenbaum, 2006))

RTL schematic of a 3-bit ALU on FPGA



RTL schematic of a 3-bit ALU generated by Quartus Prime Lite 16.1

references

recommended readings: Zwoliński, Ch. 4, Sect. 4.4-4.6

useful materials for the proposed lab experience
(sources: DMI Library, Intel Corp. - FPGA University Program, 2016)

- Tanenbaum: *Structured computer organization, Fifth Edition* (Pearson, 2006) Sect. 3.2.3
- Making Qsys Components - For Quartus Prime 16.1, Appendix A
- Quartus Prime Introduction Using VHDL Designs - For Quartus Prime 16.1, Sect. 9
- DE1-SoC Quartus Settings File with Pin Assignments
- Problem when programming Cyclone V (DE1-SoC)

VHDL sources (in the reserved lab space):

- examples in this presentation
- examples in Zwoliński's book