Addendum on the VHDL introduction

Antoine Lavault[1,2]

[1]Apeira Technologies

[2]UMR CNRS 9912 STMS, IRCAM, Sorbonne Université

October 26, 2022

Contents

Contents

Generic keyword

generic is a special kind of constant which can be applied to an entity during its initialization

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity vote is
    generic ( W : positive := 4); -- here
    port (
    votes: in std_logic_vector(W - 1 downto 0);
    approval : out std_logic
    );
end vote;
architecture behavioural of vote is
constant nMin : natural := W / 2; -- used to estimate nMin
begin
    process(votes)
    variable compte : natural range 0 to votes'length;
    begin
        compte := 0;
        for k in votes'range loop
            if votes(k) = '1' then
                compte := compte + 1;
```

Subtypes

```
type in_lowert_20 is range 0 to 19;
type real_matrix_1_by_10 is array (1 to 10) of real;
type arraySLV3 is array (natural range <>) of std_logic_vector(2 downto
constant vectors: tableauSLV3 :=
("000", "001", "010", "011", "100", "101", "110", "111");
type month_name is (january, february, march, avril, may, june,
july, august, september, october, november, december);
type date is record
    day : integer range 1 to 31;
    month : month_name;
    year : positive range 1 to 3000;
end record;
constant independance : date := (4, october, 1830;
```

Attributes cont.

- A'left , A'right : leftmost, rightmost index of A
- A'high , A'low : highest, lowest index of A
- A'range : the range of indices in A
- A'length: number of elements in A
- T'left/right/high/low: leftmost, ..., values for type T
- T'image(x): string representation of X that is of type T.
- T'value(x):value of type T converted from the string X.

```
process(votes)
variable count : natural range 0 to votes'length;
begin
count := 0;
for k in votes'range loop
...
```

Functions and Procedures

- Functions and procedures can be defined in the declarative part of an architecture or in a package.
- Functions and procedures can be overloaded i.e same identifier, different parameters.

```
function andGate(V: std_logic_vector) return std_logic is
variable resultat : std_logic := '1';
begin
    for k in V'range loop
        resultat := resultat and V(k);
    end loop;
    return resultat;
end;
```

Functions vs Procedures

- A function
  - Is a sub-program returning a unique result
  - The function always returns something
  - If parameters are given, they cannot be modified
  - A function is called in an expression
- A procedure
  - doesn't return a value like a function does, but it can return values by declaring out or inout signals in the parameter list.
  - parameters in the parameter list can be of in, out, inout type
  - Is part of the concurent domain.

Basics for sequential VHDL - D latch

```
library ieee;
use ieee.std_logic_1164.all;
entity D_latch is
port (
    G : in STD_LOGIC; -- control
    D : in STD_LOGIC; -- data
    Q : out STD_LOGIC
);
end D_latch;
architecture D_latch of D_latch is
Begin
    process(G, D) is
    begin
    if (G = '1') then
        Q <= D;
    -- else -- implicit :: do not change Q
    end if;
    end process;
end D_latch;
```

Sequential VHDL - DFF

```
library ieee;
use ieee.std_logic_1164.all;
entity D_FF is
port (
CLK : in STD_LOGIC; -- clock
D : in STD_LOGIC; -- in
Q : out STD_LOGIC -- out
);
end D_FF;
architecture D_FF of D_FF is
begin
process(CLK) is
begin
if (CLK = '1' and CLK'event) then
Q <= D;
end if;
end process;
end D_FF;
```

Contents

Example circuit

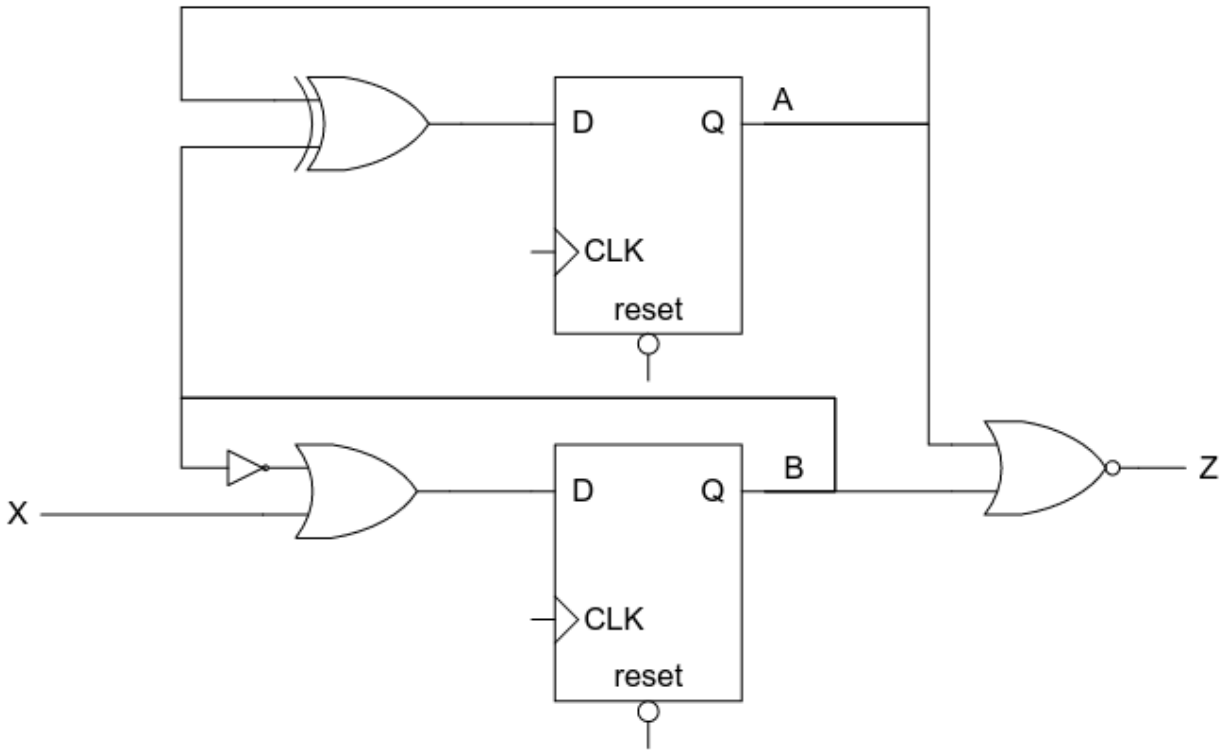

Figure: Caption

How to?

- Define the entity and its ports
- Define the architecture and the output of the FFs
- Model the flip-flops:
  - General form (process...)
  - Reset signal
  - Equations of the FF inputs
- Model the output

Which gives us...

```
library IEEE;
use IEEE.std_logic_1164.all;
entity seq_crct is
port (reset, CLK, X : in STD_LOGIC; Z : out STD_LOGIC);
end seq_crct;
architecture arch1 of seq_crct is
signal A, B : STD_LOGIC;
begin
process(CLK, reset) is
begin
if (reset = '0') then
A <= '0';
B <= '0';
elsif (rising_edge(CLK)) then
A <= A xor B;
B <= x or not(B);
end if;
end process;
z <= not(A or B);
end arch1;
```

Contents

Example

We want to make an inkjet printer. The usual colors used here are cyan, magenta, yellow and black (key): the CYMK system. The black cartridge will used as much as possible to not use too much of the color cartridges.
To convert RGB to CYMK, we have the following set of equations :

$$C_t = 255 - R \tag{1}$$
$$Y_t = 255 - G \tag{2}$$
$$M_t = 255 - B \tag{3}$$
$$K = min(Ct, Mt, Yt) \tag{4}$$
$$C = C_t - K \tag{5}$$
$$Y = Y_t - K \tag{6}$$
$$M = M_t - K \tag{7}$$
$$\tag{8}$$

CYMK conversion

We know that :

- the values are integers and are between 0 and 255.
- valid types are `Integer`, `Natural`, `Signed` and `Unsigned`
- we need a substraction
- we need a minimum

In VHDL

```
...
entity convRGB2CMYK is port (
red, green, blue : in unsigned(7 downto 0);
cyan, magenta, yellow, black : out unsigned(7 downto 0));
end convRGB2CMYK;
architecture arch2 of convRGB2CMYK is
begin
process(red, green, blue)
    variable cyant, magentat, yellowt, blackt1, blackt2 : unsigned(7 do
    begin
        cyant := 255 - red;
        magentat := 255 - green;
        yellowt := 255 - blue;
        if cyant < magentat then blackt1 := cyant;
        else blackt1 := magentat; end if;
        if blackt1 < yellowt then blackt2 := blackt1;
        else blackt2 := yellowt; end if;
        cyan <= cyant - blackt2; magenta <= magentat - blackt2;
        yellow <= yellowt - blackt2; black <= blackt2;
end process;
end arch2;
```

Assignment cont.

Be wary of type conversions and sizes !

```
library ieee;
use ieee.numeric_std.all;
entity democode2 is
port (
A8, B8 : in signed(7 downto 0);
R8 : out signed(7 downto 0);
R9, S9, T9 : out signed(8 downto 0);
R7 : out signed(6 downto 0)
);
end;
architecture arch of democode2 is
begin
R8 <= A8 + B8; -- ok, might overflow
S8 <= A8 + 100; -- ok, might overflow
R9 <= (A8(A8'left) & A8) + (B8(B8'left) & B8); -- ok
S9 <= resize(A8, S9'length) + resize(B8, S9'length); -- ok
T9 <= A8 + B8; -- no, incompatible ranges
R7 <= A8(6 downto 0) + B8(6 downto 0); -- ok, might overflow
end arch;
```

Aggregates

```
-- positional association
variable Data_1 : BIT_VECTOR (0 to 3) := ('0','1','0','1');
-- named association
variable Data_2 : BIT_VECTOR (0 to 3) :=
↪  (1=>'1',0=>'0',3=>'1',2=>'0');
-- positional association with range
signal Data_Bus : Std_Logic_Vector (15 downto 0);
Data_Bus <= (15 downto 8 => '0', 7 downto 0 => '1');
-- Named association in record
variable Status_Var : Status_Record := (Code => 57, Name =>
↪  "MOVE");
-- keyword others may be used to refer to all elements not
↪  already mentioned:
signal Data_Bus : Std_Logic_Vector (15 downto 0);
Data_Bus <= (14 downto 8 => '0', others => '1');
-- Use of keyword others
signal Data_Bus : Std_Logic_Vector (15 downto 0);
Data_Bus <= (others => 'Z');
```

Contents

Why use file IO ?

Reading a file is useful during tests with test-benches.

- The system to desribe might have previously implemented with a high-level system like Python, Matlab or even C. This is useful to generate a lot of test cases (input + output)
- Using this ground truth would be beneficial for a more exhaustive test policy.

The other way round, writing to files is interesting when:

- the simulation is extremely long
- we want partial results
- we want to process the results further (e.g an image)

Example

Let's suppose we want to test the following code :

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;
entity detectFirst is
port (
I : in unsigned(5 downto 0);
F : out std_logic
);
end detectFirst;
architecture dataflow of detectFirst is
begin
    with to_integer(I) select
    F <=
    '1' when 2 | 3 | 5 | 7 | 11 | 13 | 17 |
    19 | 23 | 29 | 31 | 37 | 41 | 43 |
    47 | 53 | 59 | 61 | 63,
    '0' when others;
end dataflow;
```

The test file

Our test file is similar to:

```
-- column 1 : integers from 0 to 63
-- column 2: F for first, N for not first
0 N
1 N
2 F
3 F
4 N
5 F
...
```

Test-bench with IO

Import the package textio !

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
entity detectFirstTB is
end detectFirstTB;
architecture arch2 of detectFirstTB is
signal I : unsigned(5 downto 0); -- signal for test vectors
signal F : std_logic; -- signal for the test output
constant filename : string := "first.txt";
file vectors : text open read_mode is filename;
begin
UUT : entity detectFirst(flotdonnees) port map (I, F);
```

Test-bench with IO

```
process
    variable linebuffer : line; -- pointer to a string object
    variable n : integer;
    variable c : character;
    begin
    while not endfile(vectors) loop
    readline(vectors, linebuffer);
    if linebuffer(1 to 2) /= "--" then -- passer les lignes de commenta
    read(linebuffer, n); -- reading the integer
    read(linebuffer, c); -- reading the space
    read(linebuffer, c); -- reading the output: first or not
    I <= to_unsigned(n, 6);
    wait for 10 ns;
    assert ((c = 'P') = (F = '1') and (c = 'N') = (F = '0'))
    report "input error " & integer'image(n) severity error;
    end if;
    end loop;
    deallocate(linebuffer); -- release the memory
    report "all done" severity failure;
end process;
end arch2;
```

Writing with IO

```
file results : text open write_mode is “results.txt”;
process
    variable buffer_ : line; -- pointer to string object
    begin
    -- writeline frees the pointer when it's done. We need a copy if we
    write(buffer_, string'(" ** simulation output, detectFirstTB.vhd **
    writeline(results, buffer_); -- write in file
    for k in 0 to 63 loop -- exhaustive test
        I <= to_unsigned(k, 6);
        wait for 10 ns;
        write(buffer_, string'("time: ")); write(buffer_, now, unit => 
        write(buffer_, string'(", int: ") & integer'image(k));
        write(buffer_, string'(", out: ") & std_logic'image(F));
        writeline(results, buffer_); --  write in file
    end loop;
    report "all done" severity failure;
end process;
```

Contents

Quisktart

Blinking LEDs.

Exercise 1

Give the VHDL for the following register:

- 8 bits
- Loading on a falling clock edge and when *load* is high
- Synchronous reset to $0xf4$ when *reset* is low

Donnez le code VHDL pour le registre suivant : 8 bits; loadment sur une transition négative de l'horloge si le signal load = '1'; réinitialisation synchrone à F4h quand le signal reset est égal à '0'.

Solution 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity register is
        generic (W : integer := 8);
        port(reset, CLK, load : in STD_LOGIC;
                D : in STD_LOGIC_VECTOR(W - 1 downto 0);
                Q : out STD_LOGIC_VECTOR(W - 1 downto 0));
end register;
architecture arch of register is
begin
        process (CLK, reset)
        begin
            if falling_edge(CLK) then
                    if reset = '0' then
                            Q <= X"F4";
                    elsif load = '1' then
                            Q <= D;
                    end if;
                end if;
        end process;
end arch;
```

Exercise 2

What does this do ?

```
library ieee;
use ieee.std_logic_1164.all;
entity mystery is
        port (a, b, c: in std_logic;
        s : in std_logic_vector (1 downto 0);
        o : out std_logic);
end mystery;
architecture architecture of mystery is
begin
    process (a, b, c, s)
        begin
                if (s = "00") then o <= a;
                        elsif (s = "01") then o <= b;
                        elsif (s = "10") then o <= c;
                        else o <= c;
                end if;
        end process;
end architecture;
```

Solution 2

This is a 4 input multiplexer with control signal s. Note that input 2 and 3 are linked to output c.