

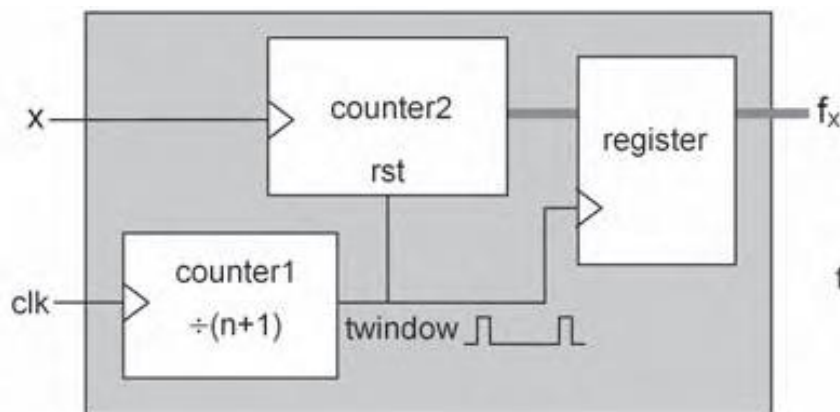
<p align="center">Examen CE311</p> <p align="center">Partie matérielle</p> <p align="center">« conception numérique & VHDL »</p>	<p align="center">3^{ème} année 2017-2018</p> <p align="center">Durée indicative de la partie : 90 mn</p> <p align="center"><u>Document autorisé : syntaxe VHDL essentielle</u></p> <p align="center">Calculatrice interdite</p>
<p><i>Les points donnés dans l'énoncé entre crochets [X] après chaque question représentent le barème et indiquent le temps à passer en minutes sur chaque question.</i></p> <p><i>Rédigez sur des feuilles séparées les 2 parties de cet examen.</i></p>	

Conception et validation d'un fréquencemètre [90+10]

L'objectif de cette partie est de concevoir et valider un fréquencemètre, c'est-à-dire un circuit qui mesure la fréquence d'un signal binaire périodique x .

- clk est l'horloge du système, de fréquence f_{clk}
- x est le signal dont on veut mesurer la fréquence, de fréquence f_x

Dans notre cas la fréquence f_x est **largement supérieure à la fréquence f_{clk}** . De ce fait il est possible de faire la mesure avec le circuit représenté sur la figure ci-dessous.



Le compteur *counter1* divise la fréquence de clk par $n+1$ de façon à générer un signal *twindow* qui reste à 1 pendant T_0 (une période de clk) et reste à 0 pendant nT_0 . Dans la suite, *count1* est le nom de la valeur qui s'incrémente dans ce compteur.

On choisit $n=f_{clk}$ de manière à obtenir $nT_0=1$ seconde.

Le signal *twindow* commande la capture de la valeur *count2* du compteur *counter2* dans le registre *register*. Il commande également la mise à 0 synchrone de ce compteur.

1 En considérant $f_x=2.f_{clk}$ et $n=8$, faire un chronogramme montrant l'évolution des signaux : clk , x , *twindow*, *count1*, *count2*, et f_x [20].

2 Quelle est la fréquence f_{clk} de clk et donc la fréquence théorique de f_x dans le cas précédent [4]?

3 A partir du chronogramme précédent, vérifiez que la valeur f_x obtenue correspond à cette fréquence de x théorique (avec éventuellement un décalage de ± 1). Expliquez pourquoi la valeur stockée dans le registre correspond à cette fréquence [4+4].

Vous allez maintenant décrire les 3 blocs du schéma précédent en VHDL.

4 Commencez par décrire en VHDL RTL l'entité *register* et architecture *rtl* du registre [6+6].

Pour cela vous considérerez :

- que la valeur maximale du compteur *counter2* est *fxmax*, qui est un paramètre générique de type entier de l'entité
- pour l'entrée *data_in* et la sortie *data_out* du registre utilisez un type « *integer range 0 to fxmax* ».

5 Combien de bascules nécessitera le circuit *register* pour *fxmax=20* ? [4]

6 Continuez en décrivant en VHDL RTL l'entité *counter2* et l'architecture *rtl* du bloc *counter2* [2+8].

Pour cela vous considérerez :

- que la valeur maximale du compteur *counter2* est un paramètre générique (comme dans le circuit précédent) est vaut *fxmax*.
- que son reset nommé *reset* est synchrone au signal d'horloge *clk*
- que la sortie du compteur *count2* est de mode *out*.

7 Combien de bascules nécessitera le circuit *counter2* pour *fxmax=20* ? [4]

8 Donnez la description VHDL RTL de l'entité *counter1* et de l'architecture *rtl* du troisième bloc *counter1* [2+8].

Pour cela vous considérerez :

- que la valeur maximale de ce compteur *fclock* est un paramètre générique de type entier de l'entité *counter1*
- seule la sortie *twindow* est disponible en sortie
- pour stocker la valeur du compteur *count1* utilisez un signal interne de type « *integer range 0 to fclk* ».

9 Combien de bascules nécessitera le circuit *counter1* pour *fclock=8* ? [4]

Maintenant que les 3 blocs ont été décrits individuellement il faut les interconnecter pour créer le circuit complet.

10 Donnez l'entité *freqmeter* et la **description structurelle** (sans process explicite mais uniquement avec des instanciations) de l'architecture *struct* du circuit complet [4+10].

Pour cela vous considérerez :

- que la valeur maximale du compteur *counter2* est *fxmax*, qui est un paramètre générique de type entier de l'entité *struct*
- que la valeur maximale du compteur *counter1* est *fclock*, qui est un paramètre générique de type entier de l'entité *struct*
- des instanciations directes pour éviter de devoir déclarer les 3 composants

11 Optionnel : Donnez l'entité *tb_freqmeter* et l'architecture *rtl* d'un testbench de *freqmeter* permettant de retrouver **exactement le même chronogramme que celui que vous avez donné à la question 1** [+2+8]

Pour cela vous considérerez que *fxmax=20*

VHDL Quick Reference Card

This file has been created thanks to the VHDL Quick Reference Card from the College of New Jersey.

1. Introduction

VHDL is a **case insensitive** and **strongly typed** language. Comments start with two adjacent hyphens (--) and end at end of line.

2. Compilation Units

Library Usage Declarations	-- ref. 11
Entity Declarations	-- ref. 3
Architecture Declarations	-- ref. 4
Package Declarations	-- ref. 10
Configuration Declarations	-- ref. 14

3. Entity Declaration

```
entity n_input_nand is
    generic ( n : integer := 2);
    port (    data : in bit_vector( 1 to n );
            result : out bit );
end n_input_nand;
-- port directions : in | out | inout
```

4. Architecture Declaration

```
architecture behave of n_input_nand is
    -- declarations                                -- ref. 7
begin
    -- concurrent statements                        -- ref. 9
end behave;
```

5. Operators

logical operators : **and**, **or**, **xor**, **nand**, **nor**, **xnor**, **not**
relational operators : **=**, **/=**, **<**, **<=**, **>**, **>=**
shift left/right logical operators : **sll**, **srl**
shift left/right arithmetic operators : **sla**, **sra**
rotate left/right logical operators : **rol**, **ror**
other operators : **+**, **-**, **&**, *****, ******, **/**, **mod**, **abs**, **rem**
eg. **&** : concatenation, '1' & "10" = "110"
****** : exponentiation, 2 ** 3 = 8
rem : remainder, 7 rem 2 = 1
mod : division modulo, 5 mod 3 = 2

6. Data Types

6.1 Predefined Data Types

bit	'0' and '1'
bit_vector	Array of "bit"
boolean	true and false

character	7-bit ASCII
integer	signed 32 bit at least
natural	integer >= 0
positive	integer > 0
real	Floating point, min : +1e38 to -1e38
string	Array of characters
time	hr, min, sec, ms, us, ns, ps, fs

6.2 User Defined Data Types

```
-- type range is range 0 to 100000
units
    meter;                                -- base unit
    kilometer = 1000 meter;
end units distance;
-- type number is integer;
-- type voltage is range 0 to 5;
-- type current is range 1000 downto 0;
-- type d_bus is array ( range <> ) of bit;
-- type instruction is record
    opcode : bit;
    operand : bit;
end record;
-- type int_file is file of integer;
-- type pointer_to_integer is access integer;
-- subtype positive_number is integer range 0 to 100000
-- type fourval is ( X, L, H, Z );
-- subtype resolve_n is resolve twoval;
```

7. Declarations

```
-- constant bus_width : integer := 32
-- variable read_flag : bit := 0;
-- only in processes
-- and subprograms
-- signal clock " bit;
-- file f3 : int_file open write_mode is "test.out";
-- alias enable : bit is addr(31);
-- attribute delay : time;
-- component n_input_nand
    generic ( n : integer := 2 );
    port ( data : in bit_vector ( 1 to n );
           result : out bit );
end component n_input_nand;
-- function square ( i : integer ) return integer;
-- for store : use configuration latch;
```

8. Attributes

```
-- type my_array is array ( 9 downto 0 ) of any_type;
-- variable an_array : my_array;
-- type fourval is ( '0', '1', 'Z', 'X' );
-- signal sig : sigtype;
-- constant T : time := 10 ns;
```

Attribute	Result type	Result
my_array'high	any_type	9
my_array'left	any_type	9
my_array'low	any_type	0

my_array'right	any_type	0
my_array'ascending	boolean	false
my_array'length	integer	10
my_array'range	integer	9 downto 0
my_array'reverse_range	integer	0 to 9
fourval'leftof('0')	fourval	error
fourval'leftof('1')	fourval	'0'
fourval'pos('Z')	integer	2
fourval'pred('1')	fourval	'0'

9. Statements

9.1 Concurrent Statements

```
-- state_mach : process ( state ) -- label is optional
-- variable declarations -- ref. 7
begin
    -- sequential statements -- ref. 9
end process;
-- U1_n_input_nand : n_input_nand
    generic map ( n => 2 )
    port map ( data => my_data;
              result => my_res );
-- top_block : block
-- declaration -- ref. 7
begin
    -- concurrent statements -- ref. 9
end block;
-- label1 : for i in 1 to 3 generate
    label2 : nand2 port map ( a(i), b(i), c(i) );
end generate
-- label3 : if ( i < 4 ) generate
    label4 : nor2 port map( a(i), b(i), c(i) );
end generate;
```

9.2 Sequential Statements

```
-- null;-- does nothing
-- wait on sig1, sig2 until ( sig = '1' ) for 30 ns;
-- wait until ( clock'event and clock = '1' );
-- read_flag := 0; -- read_flag is a variable
-- if ( x < y ) then max := y;
-- elsif ( x > y ) then max := x; -- optional
-- else max := x; -- optional
-- end if;
-- case is
    when '1' | '0' => d <= '1';
    when 'Z' => d <= '0';
    when others => d <= 'X'; -- optional
end case;
-- while ( x < y ) loop
    next when ( x > 5 ); -- usage of next
    x := x + 1;
end loop;
-- for i in ( 0 to 100 ) loop
    x := x + i;
    exit when ( x = 0 ); -- usage of exit
end loop;
```

9.3 Concurrent and Sequential Statements

```
.. enable <= select after 1 ns;
.. assert ( a = b )
.. report "a is not equal to b"
.. severity note;
-- severity levels : note | warning | error | failure
```

10. Package Declarations

```
.. package two_level is
-- type, signal, functions declarations -- ref. 7
end two_level;
.. package body two_level is
-- subprogram definitions -- ref. 7
end two_level;
```

11. Library Usage Declarations

```
-- using the two_level package.
library work;
use work.two_level.all; -- all objects used
use work.two_level.vcc; -- only the "vcc" object used
```

12. Subprograms

```
function bool_2_level ( boolean : in_bool )
return two_level is
variable return_val : two_level;
begin
if ( in_bool = true ) then
return_val := high;
else return_val := low;
end if;
return return_val;
end bool_2_level;
```

```
procedure clock_buffer
( signal local_clk : inout bit;
signal clk_pin : in bit;
constant clock_skew : in time ) is
begin
-- example of side effects in a procedure
global_clk <= local_clk after clk_skew;
local_clk <= clk_pin;
end clock_buffer;
```

13. Predefined Subprograms

```
.. enable <= '1' when ( now < 2 ns ) else '0';
.. variable ptoi : pointer_to_integer;
ptoi := new integer; -- usage of new
deallocate ( ptoi );
.. variable status : file_open_status;
file my_file : int_file;
file_open( status, my_file, "in.dat", read_mode );
.. end_file ( my_file ); -- returns true/false
.. variable int_var : integer;
read ( my_file, int_var );
```

```
.. file_close ( my_file );
```

14. Configuration Declarations

```
configuration input_8 of n_nand is
for customizable
for a1 : nand_2
use entity work..nand_2 ( n_nand_arch );
end for;
end for;
end input_8;
```

15. Non-synthesizable Constructs

Most tools will not synthesize :

access, after, alias, assert, bus, disconnect, file, guarded, inertial, impure, label, linkage, new, on, open, postponed, pure, reject, report, severity, shared, transport, units, with.

16. Standard Packages

Samplings of a subset of standard packages the language provides.

16.1 IEEE.STD_LOGIC_1164 Package

```
type std_ulogic is ( 'U', 'X', '0', '1', 'W', 'L', 'H', 'Z', '-' );
-- MLV9

type std_ulogic_vector is array
( natural range <> ) of std_ulogic;
function resolved ( s : std_ulogic_vector ) return std_ulogic;
subtype std_logic is resolved std_ulogic;
type std_logic_vector is array
( natural range <> ) of std_logic;
function to_bit ( s : std_ulogic; xmap : bit := '0' ) return
bit; function to_bitvector
( s : std_logic_vector; xmap : bit := '0'
) return bitvector;
function to_stdlogicvector ( b : bit_vector
) return std_logic_vector;
function rising_edge ( signal s : std_ulogic ) return boolean;
function falling_edge ( signal s : std_ulogic ) return boolean;
function is_x ( s : std_logic_vector ) return boolean;
```

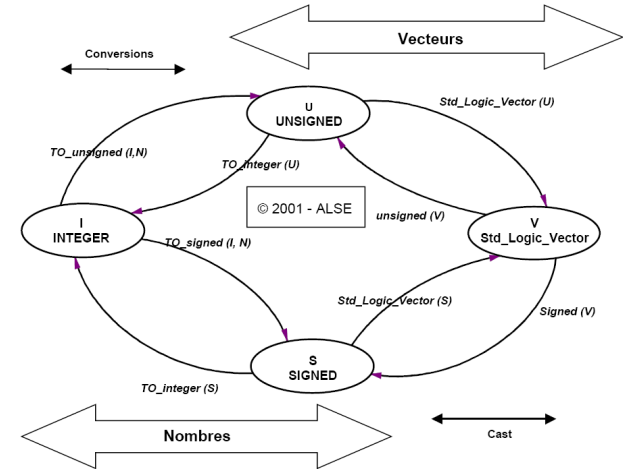
16.2 STD.TEXTIO Package

```
type line access string;
type text is file of
string; type side is (
right, left ); subtype
width is natural;
file input : text open read_mode is "std_input";
file output : text open write_mode is "std_output";
procedure readline ( file f : text; I : out line );
procedure writeline ( file f : text; I : in line );
procedure read ( I
: inout line;
value : out bit;
good : out boolean );
```

```
procedure write ( I : inout line;
value : in bit;
justified : in side := right;
field : in width := 0 );
-- The type of "value" can be bit_vector | boolean |
-- character | integer | real | string | time.
-- There is no standard package for textio operations on
std_logic. Tools vendors may provide their own.
```

16.3 IEEE.NUMERIC_STD Package

```
type unsigned is array ( natural range <> ) of std_logic;
type signed is array ( natural range <> ) of std_logic;
function shift_left ( arg : unsigned; count : natural )
return unsigned;
-- Other functions: shift_right(), rotate_left(),
rotate_right()
function rsize ( arg : signed; new_size : natural
) return signed;
```



VHDL Quick Reference Card is intended for quick reference.
Please use VHDL LRM of 1993 for details.