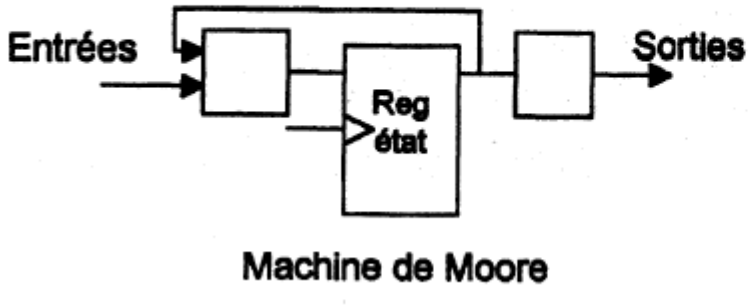
Machine à Etats finis FSM

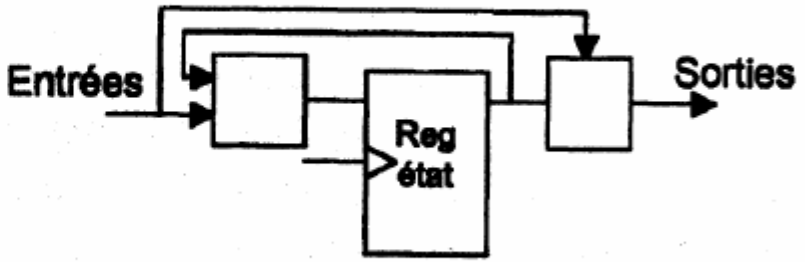
- FSM *Finite State Machine*
- Outil pour représenter un système séquentiel
- On définit différents états dans lesquels peut être le système
- Le passage d'un état à un autre s'effectue si une condition sur les entrées est remplie
- Les sorties du système dépendent de l'état courant (machine de Moore) ou de l'état courant et des entrées (machine de Mealy)

Machine à Etats finis FSM

- Deux architectures courantes :



Machine de Moore



Machine de Mealy

- Des bascules enregistrent l'état courant
- Des circuits combinatoires sont placés avant et après les bascules pour déterminer l'état suivant et la valeur des sorties

Machine à Etats finis FSM

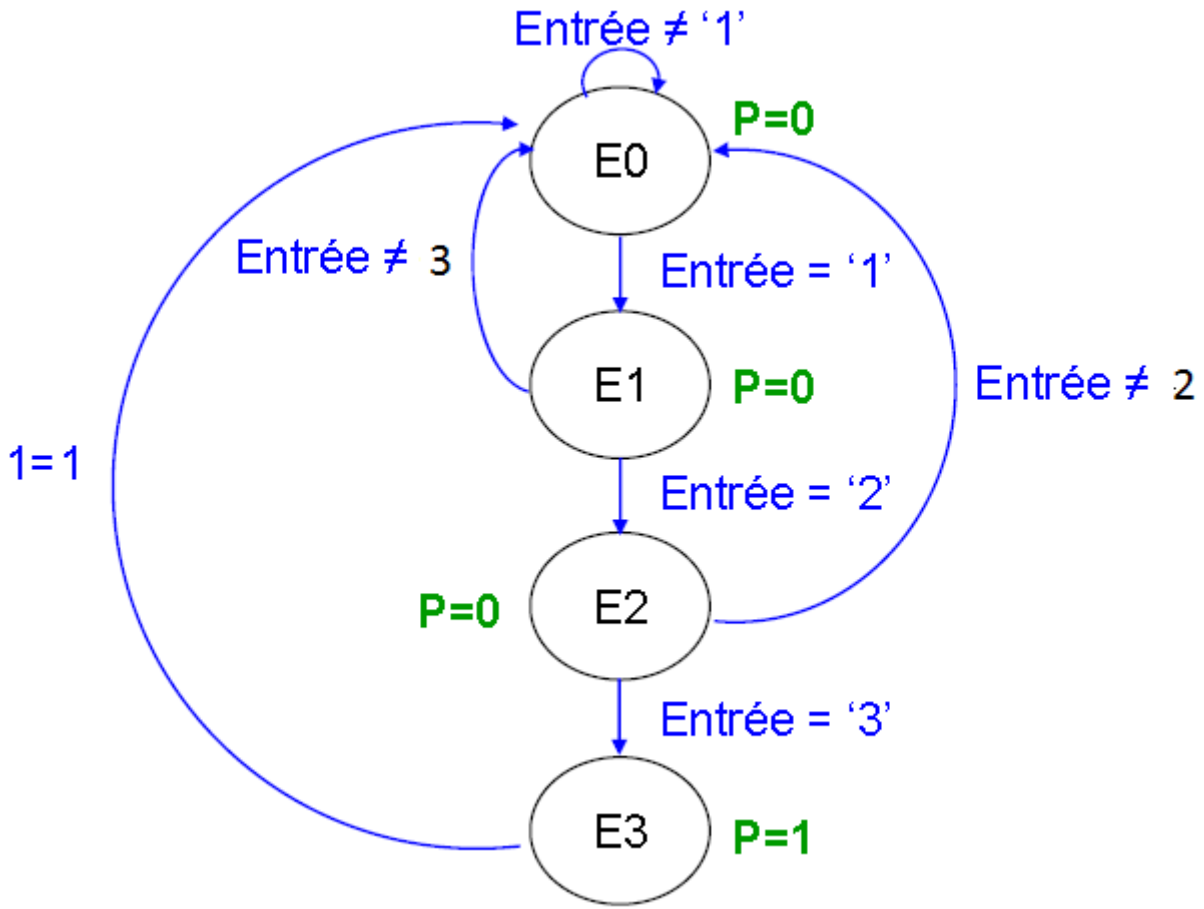
Exp: Détecteur de séquence

La porte ne s'ouvre que si l'on tape la séquence '1' '2' '3'

- Etat 0 : le système attend un '1' en entrée, la porte est fermée (P=0)
- Etat 1 : le système attend un '2' en entrée, la porte est fermée (P=0)
- Etat 2 : le système attend un '3' en entrée, la porte est fermée (P=0)
- Etat 3 : la bonne séquence a été entrée, la porte est ouverte (P=1)

Machine à Etats finis FSM

On représente une machine à état par un graphe d'état

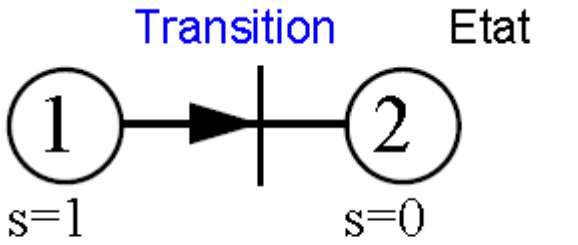


Machine à Etats finis FSM

Graphe d'états

Définition :

- **Un diagramme ou graphe d'états** permet d'avoir une représentation graphique d'un système séquentiel.
- Il est constitué par l'énumération de tous les états possible du système.
- Un seul de ces états peut être actif à la fois.
- A chaque état est associé la valeur de la (ou des) grandeur(s) de sortie.



Valeur de la (ou des) sortie(s)

Programmer un FPGA avec un FSM

- Avec le logiciel Quartus, on peut décrire une FSM
  - en utilisant une architecture de Moore ou de Mealy que l'ont fait soit même
  - schématiquement, en rentrant directement le graphe d'état
  - en la décrivant en VHDL

Description d'un FSM en VHDL

- **Entité**

```
Entity machine is
port(
    clk : in std_logic;
    bouton : in integer range 0 to 9;
    P : out std_logic);
end machine
```

- **Architecture**

Déclaration d'un nouveau type énuméré contenant les noms des états

```
architecture arch of machine is

type StateType is (etat0, etat1, etat2, etat3);

signal etat : StateType;

begin
```

Déclaration d'un signal du nouveau type juste déclaré

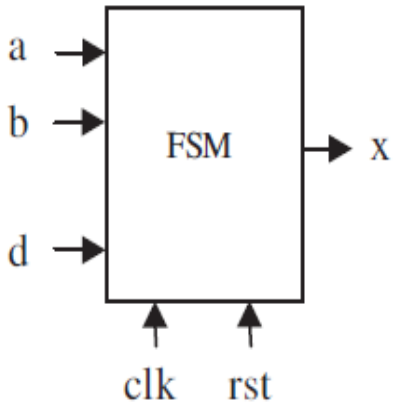
Description d'un FSM en VHDL

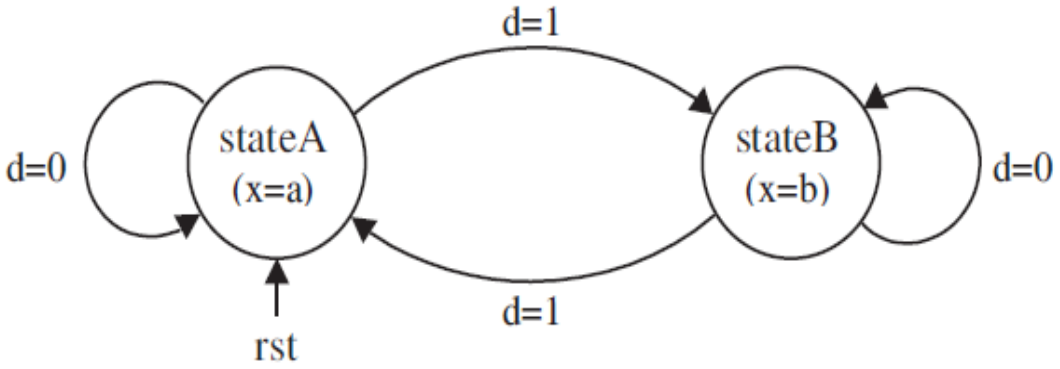
```
begin

    process(clk)
    begin
        if clk'event and clk='1' then
            case etat is
                when etat0 => if bouton = 1 then etat <= etat1;
                                else etat <= etat0; end if;
                when etat1 => if bouton = 2 then etat <= etat2;
                                else etat <= etat0; end if;
                when etat2 => if bouton = 3 then etat <= etat4;
                                else etat <= etat0; end if;
                when etat3 => etat <= etat0;
            end case;
        end if;
    end process;

    P <= '1' when (etat =etat3) else '0';

end arch;
```

Gestion des états

Gestion de la valeur de la sortie

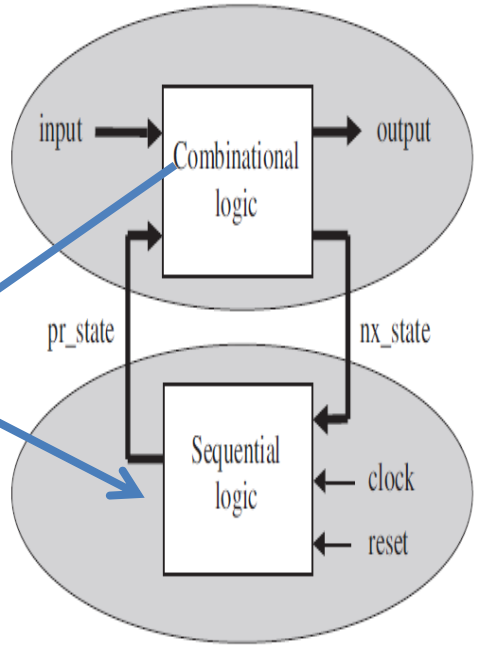
Description d'un FSM en VHDL



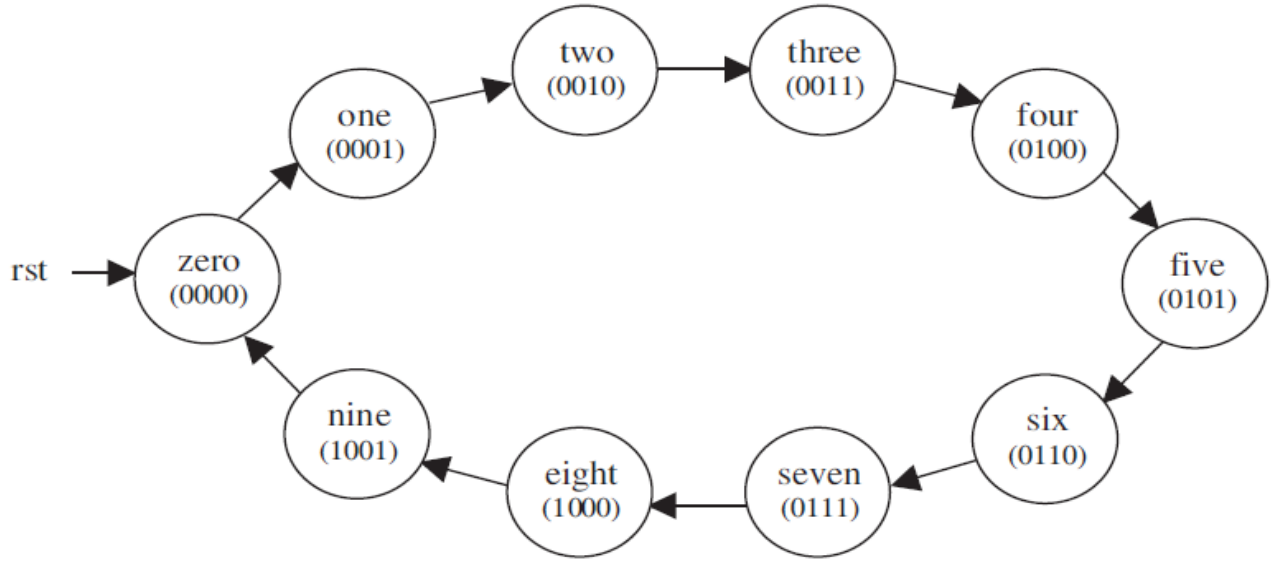


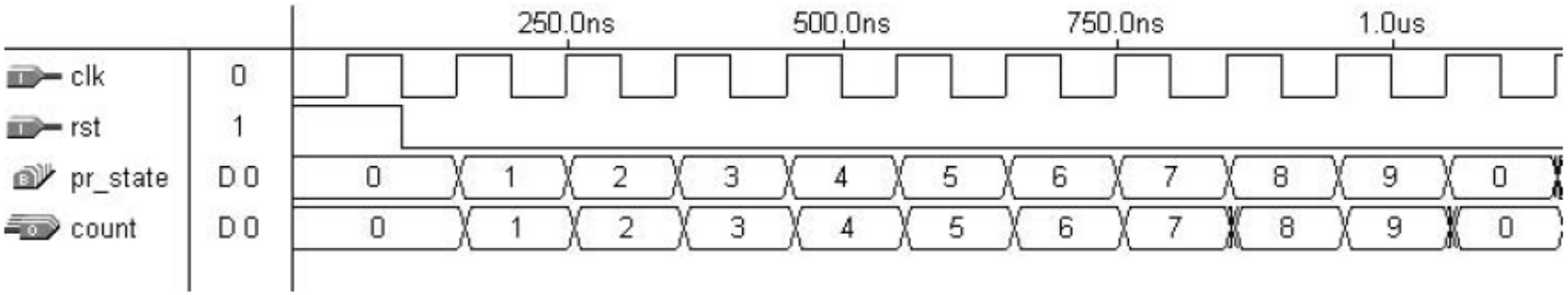
```
1  ------------------------------------------
2  ENTITY simple_fsm IS
3     PORT ( a, b, d, clk, rst: IN BIT;
4            x: OUT BIT);
5  END simple_fsm;
6  ------------------------------------------
7  ARCHITECTURE simple_fsm OF simple_fsm IS
8     TYPE state IS (stateA, stateB);
9     SIGNAL pr_state, nx_state: state;
10 BEGIN
11    ----- Lower section: --------------------
12    PROCESS (rst, clk)
13    BEGIN
14       IF (rst='1') THEN
15          pr_state <= stateA;
16       ELSIF (clk'EVENT AND clk='1') THEN
17          pr_state <= nx_state;
18       END IF;
19    END PROCESS;
20    ---------- Upper section: ---------------
21    PROCESS (a, b, d, pr_state)
22    BEGIN
23       CASE pr_state IS
24          WHEN stateA =>
25             x <= a;
26             IF (d='1') THEN nx_state <= state
27             ELSE nx_state <= stateA;
28             END IF;
29          WHEN stateB =>
30             x <= b;
```

Reset asynchrone pour l'initiation de l'etat de l'FSM



Exp: conter BCD/ décrire le fct de ce circuit en se basant sur le diagramme d'états suivant:





```
1  ------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------------
5  ENTITY counter IS
6     PORT ( clk, rst: IN STD_LOGIC;
7            count: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
8  END counter;
9  ------------------------------------------------
10 ARCHITECTURE state_machine OF counter IS
11    TYPE state IS (zero, one, two, three, four,
12       five, six, seven, eight, nine);
13    SIGNAL pr_state, nx_state: state;
14 BEGIN
15    ------------ Lower section: ----------------
16    PROCESS (rst, clk)
17    BEGIN
18       IF (rst='1') THEN
19          pr_state <= zero;
20       ELSIF (clk'EVENT AND clk='1') THEN
21          pr_state <= nx_state;
22       END IF;
23    END PROCESS;
24    ------------ Upper section: ----------------
25    PROCESS (pr_state)
26    BEGIN
27       CASE pr_state IS
28          WHEN zero =>
29             count <= "0000";
30             nx_state <= one;
31          WHEN one =>
32             count <= "0001";
33             nx_state <= two;
```

```
34          WHEN two =>
35             count <= "0010";
36             nx_state <= three;
37          WHEN three =>
38             count <= "0011";
39             nx_state <= four;
40          WHEN four =>
41             count <= "0100";
42             nx_state <= five;
43          WHEN five =>
44             count <= "0101";
45             nx_state <= six;
46          WHEN six =>
47             count <= "0110";
48             nx_state <= seven;
49          WHEN seven =>
50             count <= "0111";
51             nx_state <= eight;
52          WHEN eight =>
53             count <= "1000";
54             nx_state <= nine;
55          WHEN nine =>
56             count <= "1001";
57             nx_state <= zero;
58       END CASE;
59    END PROCESS;
60 END state_machine;
61 -------------------------------------------------
```

Library

- LIBRARY declarations:
  - contient une liste de tous les bibliothèques à utiliser dans le design
  - Une collection de toutes pieces (parties) communes de code. Placez ces pieces dans une “LIBRARY” les permetent d'être réutiliser et partager avec d’autres designs.

Library

Library Déclarations:

✓Deux lignes de code sont nécessaires, un contenant le nom de la bibliothèque et un autre contenant « a use clause »:

```
LIBRARY library_name;
USE library_name.package_name.package_parts;
```

exp: Library ieee, std, work, etc

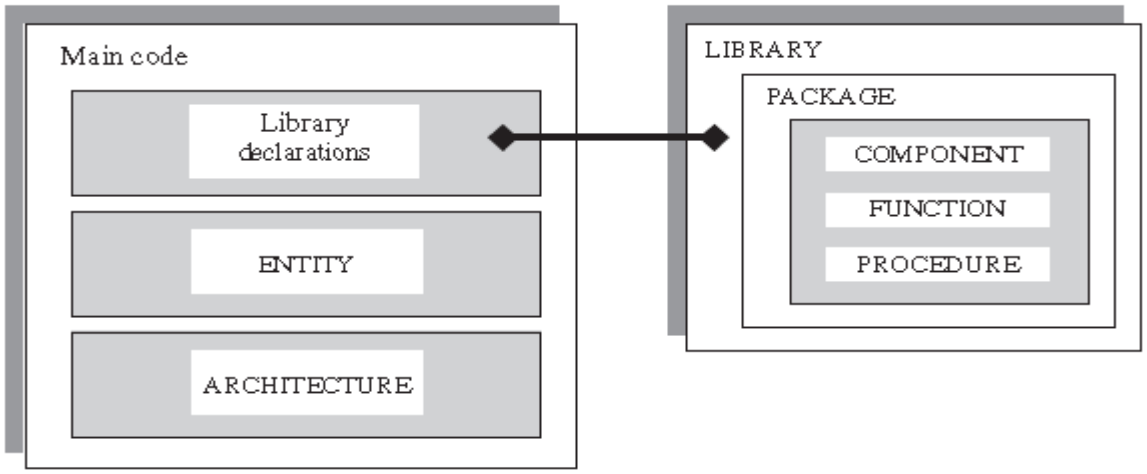
```
LIBRARY ieee;                       -- A semi-colon (;) indicates
USE ieee.std_logic_1164.all;        -- the end of a statement or

LIBRARY std;                        -- declaration, while a double
USE std.standard.all;               -- dash (--) indicates a comment.

LIBRARY work;
USE work.all;
```

Library

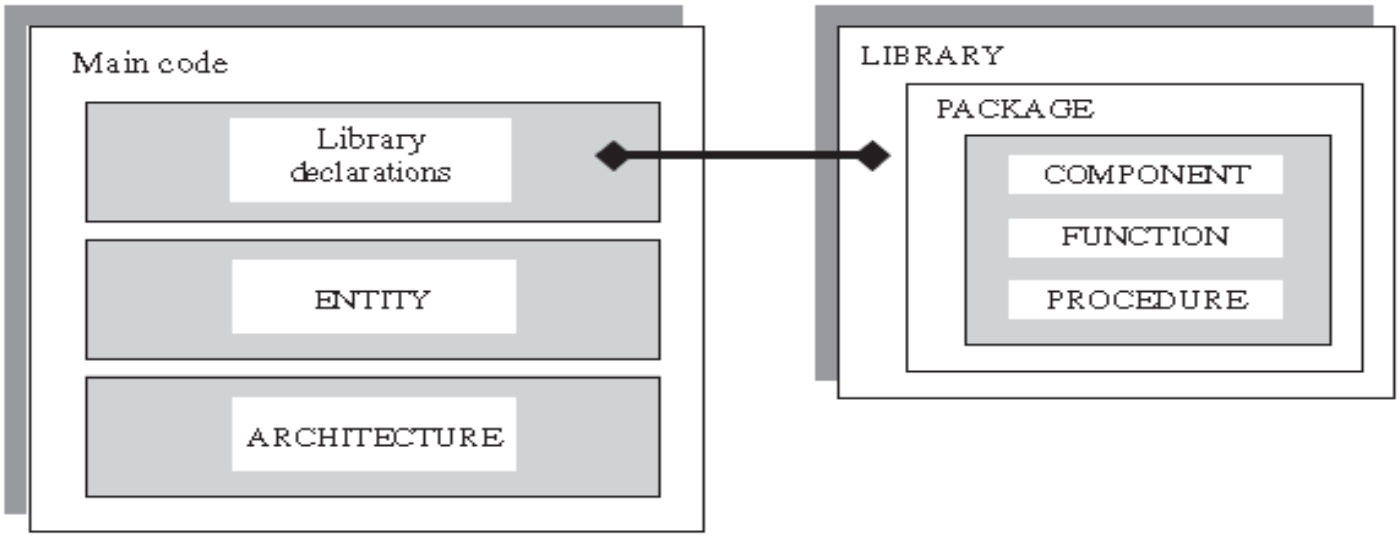
✓std_logic_1164 pakage de Libra IEEE specifie un system logique à multi-niveaux

✓std: Libra de ressources (type de données, text I/O,) pour l'environnement de design VHDL Library

✓Work:library de sauvegarde de notre design(.VHDL, fichiers crés par le compiler, simulator)



Library



Ces parties de code peuvent être écrites:

-components

-fonctions

-procédures

Placer dans un « Package »

Package

User developped package

```
use package_name.all;
```

Example

```
library lib_name;          -- make library visible
 use lib_name.pkg_name.all;   -- make package visible
```

- Package *standard* of library *std*: Defines BIT, BOOLEAN, INTEGER, and REAL data types.
- Package *std_logic_1164* of library *ieee*: Defines STD_LOGIC and STD_ULOGIC data types.
- Package *std_logic_arith* of library *ieee*: Defines SIGNED and UNSIGNED data types, plus several data conversion functions, like *conv_integer(p)*, *conv_unsigned(p, b)*, *conv_signed(p, b)*, and *conv_std_logic_vector(p, b)*.
- Packages *std_logic_signed* and *std_logic_unsigned* of library *ieee*: Contain functions that allow operations with STD_LOGIC_VECTOR data to be performed as if the data were of type SIGNED or UNSIGNED, respectively.

Package

Package declaration format:

```
package package_name is
  ... exported constant declarations
  ... exported type declarations
  ... exported subprogram declarations
end package_name;
```

Package body format:

```
package body package_name is
    ... exported subprogram bodies
    ... other internally-used declarations
end package_name;
```

Example:

```
package ee530 is
    constant maxint: integer := 16#ffff#;
    type arith_mode_type is (signed, unsigned);
    function minimum(constant a,b: in integer) return integer;
end ee530;
```

Package

```
package body ee530 is
    function minimum (constant a,b: integer) return integer is
      variable c: integer; -- local variable
        begin
          if a < b then
             c := a;  -- a is min
          else
             c := b;  -- b is min
          end if;
          return c;  -- return min value
        end;
  end ee530;
```