

## Apéndices

### Apéndice A : empaquetamiento STANDARD

**package** STANDARD is

----- **types** enumeración predefinidos:

**type** BOOLEAN is (FALSE, TRUE);

**type** BIT is ('0', '1');

**type** CHARACTER is (

NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,  
BS, HT, LF, VT, FF, CR, SO, SI,  
DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,  
CAN, EM, SUB, ESC, FSP, GSP, RSP, USP,  
' ', '!', '"', '#', '\$', '%', '&', "'",  
'(', ')', '\*', '+', ',', '-', '.', '/',  
'0', '1', '2', '3', '4', '5', '6', '7',  
'8', '9', ':', ';', '<', '=', '>', '?',  
'@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',  
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',  
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',  
'X', 'Y', 'Z', '[', '\', ']', '^', '\_',  
'"', 'a', 'b', 'c', 'd', 'e', 'f', 'g',  
'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',  
'p', 'q', 'r', 's', 't', 'u', 'v', 'w',  
'x', 'y', 'z', '{', '|', '}', '~', DEL );

**type** SEVERITY\_LEVEL is (NOTE, WARNING, ERROR, FAILURE);

----- **types** numéricos predefinidos:

**type** INTEGER is range -2147483648 to 2147483647;

**type** REAL is range -1.0E38 to 1.0E38;

----- **type** TIME predefinido:

**type** TIME is range -2147483648 to 2147483647;

units fs; -- femtosecond

ps = 1 000 fs -- picosecond

ns = 1000 ps; -- nanosecond

us = 1000 ns; -- microsecond

ms = 1000 us; -- millisecond

sec = 1000 ms; -- second

min= 60 sec; -- minute

hr = 60 min; -- hour

**end** units;

----- función que devuelve el tiempo actual de simulación:

**function** NOW **return** TIME;

----- **subtypes** numéricos predefinidos:

**subtype** NATURAL is INTEGER range 0 to INTEGER'HIGH;

**subtype** POSITIVE is INTEGER range 1 to INTEGER'HIGH;

----- **types** array predefinidos:

**type** STRING is array (POSITIVE range <>) of CHARACTER;

**type** BIT\_VECTOR is array (NATURAL range <>) of BIT;

**end** STANDARD;

## **Apéndice B : empaquetamiento TEXTIO**

```
package TEXTIO is
type LINE is access STRING;           -- A LINE is a pointer to a STRING value
type TEXT is file of STRING;          -- a file of variable-length ASCII records
type SIDE is (RIGHT, LEFT);           -- for justifying output data within fields
subtype WIDTH is NATURAL;             -- for specifying widths of output fields
```

```
file INPUT: TEXT is in "STD_INPUT"; ----- Standard Text Files
file OUTPUT: TEXT is out "STD_OUTPUT";
```

```
procedure READLINE (F.: in TEXT; L: out LINE); ----- Input Routines for Standard Types
procedure READ (L: inout LINE; VALUE: out BIT; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out BIT);
procedure READ (L: inout LINE; VALUE: out BIT_VECTOR; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out BIT_VECTOR);
procedure READ (L: inout LINE; VALUE: out BOOLEAN; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out CHARACTER; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out CHARACTER);
procedure READ (L: inout LINE; VALUE: out INTEGER; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out INTEGER);
procedure READ (L: inout LINE; VALUE: out REAL;GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out REAL);
procedure READ (L: inout LINE; VALUE: out STRING;GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out STRING);
procedure READ (L: inout LINE; VALUE: out TIME;GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out TIME);
```

```
procedure WRITELINE(F: out TEXT; L: in LINE); ----- Output Routines for Standard Types
procedure WRITE (L: inout LINE; VALUE: in BIT;
  JUSTIFIED: in SIDE:= RIGHT; FIELD:in WIDTH := 0);
procedure WRITE (L: inout LINE;VALUE: in BIT_VECTOR;
  JUSTIFIED: in SIDE:= RIGHT; FIELD:in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in BOOLEAN;
  JUSTIFIED: in SIDE:= RIGHT; FIELD:in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in CHARACTER;
  JUSTIFIED: in SIDE:= RIGHT; FIELD:in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in INTEGER;
  JUSTIFIED: in SIDE:= RIGHT; FIELD:in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in REAL;
  JUSTIFIED: in SIDE:= RIGHT; FIELD:in WIDTH := 0;DIGITS: in NATURAL := 0);
procedure WRITE (L: inout LINE; VALUE: in STRING;
  JUSTIFIED: in SIDE:= RIGHT; FIELD:in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in TIME;
  JUSTIFIED: in SIDE:= RIGHT; FIELD:in WIDTH := 0;UNIT: in TIME := NS);
```

```
function ENDLINE(L: in LINE) return BOOLEAN; -----File Position Predicates
----- function ENDFILE(F: in TEXT) return BOOLEAN; implicita cuando se declara tipo FILE
end TEXTIO;
```

**Apéndice C : empaquetamiento “modelos”**

```
PACKAGE modelos IS
```

```
  CONSTANT K : POSITIVE;
```

```
  TYPE cuad IS ('0','1','Z','X');
```

```
  TYPE cuad_vector IS ARRAY (NATURAL RANGE <>) OF cuad;
```

```
  TYPE cuad_nibble IS ARRAY ( 3 DOWNT0 0 ) OF cuad;
```

```
  TYPE cuad_byte IS ARRAY ( 7 DOWNT0 0 ) OF cuad;
```

```
  TYPE cuad_word IS ARRAY (15 DOWNT0 0 ) OF cuad;
```

```
  TYPE cuad_2por4 IS ARRAY ( 1 DOWNT0 0, 0 TO 3) OF cuad;
```

```
  TYPE cuad_1dim IS ARRAY (cuad) OF cuad;
```

```
  TYPE cuad_2dim IS ARRAY (cuad,cuad) OF cuad;
```

```
  TYPE serie IS ARRAY ( NATURAL RANGE <> ) OF INTEGER ;
```

```
  TYPE resistencia IS RANGE 0 TO 1E16
```

```
  UNITS
```

```
    mo; -- miliohms (unidad )
```

```
    ohms = 1000 mo;
```

```
    kohms      = 1000 ohms;
```

```
    mohms      = 1000 kohms;
```

```
  END UNITS;
```

```
  TYPE capacidad IS RANGE 0 TO 1E16
```

```
  UNITS
```

```
    ffr; -- femto faradios (unidad)
```

```
    pfr  = 1000 ffr;
```

```
    nfr  = 1000 pfr;
```

```
    ufr  = 1000 nfr;
```

```
    mfr  = 1000 ufr;
```

```
    far  = 1000 mfr;
```

```
  END UNITS;
```

```
  FUNCTION fmn (x,y,z:BIT) RETURN BIT;
```

```
  FUNCTION figl (x,y,i:BIT) RETURN BIT;
```

```
  FUNCTION "NOT" (a :cuad) RETURN cuad;
```

```
  FUNCTION "AND" (a,b :cuad) RETURN cuad;
```

```
  FUNCTION "OR" (a,b :cuad) RETURN cuad;
```

```
  FUNCTION incrementar ( V: BIT_VECTOR ) RETURN BIT_VECTOR;
```

```
  FUNCTION decrementar ( v: BIT_VECTOR ) RETURN BIT_VECTOR;
```

```
  FUNCTION binario_a_entero ( dato: BIT_VECTOR ) RETURN INTEGER;
```

```
  FUNCTION entero_a_bin ( entero:INTEGER; bits: POSITIVE ) RETURN BIT_VECTOR;
```

```
  PROCEDURE enter_a_binar ( entero: IN INTEGER; binario : OUT BIT_VECTOR);
```

```
  PROCEDURE tst_vect ( SIGNAL vectores : OUT BIT_VECTOR ;
```

```
                      CONSTANT valores : IN serie;
```

```
                      CONSTANT periodo : IN TIME);
```

```
  PROCEDURE bustest (SIGNAL busX : IN cuad_vector; SIGNAL error: OUT BOOLEAN);
```

```
COMPONENT    INVER      PORT (e1      : IN BIT; sal: OUT BIT);    END COMPONENT;
COMPONENT    AND2       PORT (e1,e2 : IN BIT; sal: OUT BIT);    END COMPONENT;
COMPONENT    OR2        PORT (e1,e2 : IN BIT; sal: OUT BIT);    END COMPONENT;
COMPONENT    XR2        PORT (e1,e2 : IN BIT; sal: OUT BIT);    END COMPONENT;
COMPONENT    NAND2      PORT (e1,e2 : IN BIT; sal: OUT BIT);    END COMPONENT;
COMPONENT    NAND3      PORT (e1,e2,e3 : IN BIT; sal: OUT BIT);  END COMPONENT;
COMPONENT    SEMISUM     PORT (a,b      : IN BIT; ca, su: OUT BIT);  END COMPONENT;
COMPONENT    SUMADOR     PORT (x,y,ci : IN BIT; sum,co: OUT BIT);  END COMPONENT;
COMPONENT    MULTUNBI    PORT (x,y,z,w : IN BIT; co, pr: OUT BIT);  END COMPONENT;
```

END modelos;

-----  
PACKAGE BODY modelos IS

CONSTANT K: POSITIVE:= 4;

```
FUNCTION fmn (x,y,z: BIT) RETURN BIT IS
BEGIN
    RETURN ( x AND z) OR (NOT y AND z) OR (x AND NOT y);
END fmn;
```

```
FUNCTION figl (x,y,i: BIT) RETURN BIT IS
BEGIN
    RETURN ( x AND y AND i) OR (NOT x AND NOT y AND i);
END figl;
```

```
FUNCTION "NOT" (a :cuad) RETURN cuad IS
CONSTANT tabla_not_cuad : cuad_1dim :=      ('1','0','0','X');
BEGIN
    RETURN tabla_not_cuad (a);
END "NOT";
```

```
FUNCTION "AND" (a,b :cuad ) RETURN cuad IS
CONSTANT tabla_and_cuad : cuad_2dim := (      ('0','0','0','0'),
                                                ('0','1','1','X'),
                                                ('0','1','1','X'),
                                                ('0','X','X','X'));
BEGIN
    RETURN tabla_and_cuad (a,b);
END "AND";
```

```
FUNCTION "OR" (a,b :cuad ) RETURN cuad IS
CONSTANT tabla_or_cuad : cuad_2dim := (      ('0','1','1','X'),
                                                ('1','1','1','1'),
                                                ('1','1','1','1'),
                                                ('X','1','1','X'));
BEGIN
    RETURN tabla_or_cuad (a,b);
END "OR";
```

```
FUNCTION incrementar ( V: BIT_VECTOR ) RETURN BIT_VECTOR IS
  VARIABLE bv : BIT_VECTOR(V'LENGTH-1 DOWNT0 0);
BEGIN
  bv := v;
  FOR i IN 0 TO bv'HIGH LOOP
    IF    bv(i) = '0' THEN    bv(i) := '1';
      EXIT;
    ELSE bv(i) := '0';
    END IF;
  END LOOP;
  RETURN bv;
END incrementar;
```

```
FUNCTION decrementar ( v : BIT_VECTOR ) RETURN BIT_VECTOR IS
  VARIABLE bv : BIT_VECTOR(v'LENGTH-1 DOWNT0 0);
BEGIN
  bv := v;
  FOR i IN 0 TO bv'HIGH LOOP
    IF    bv(i) = '1' THEN    bv(i) := '0';
      EXIT;
    ELSE bv(i) := '1';
    END IF;
  END LOOP;
  RETURN bv;
END decrementar;
```

```
FUNCTION binario_a_entero ( dato: BIT_VECTOR ) RETURN INTEGER IS
  VARIABLE resultado : INTEGER := 0;
BEGIN
  FOR n IN dato'RANGE LOOP
    IF dato(n) = '1' THEN
      resultado := resultado + ( 2** n);
    END IF;
  END LOOP;
  RETURN resultado;
END binario_a_entero;
```

```
FUNCTION entero_a_bin ( entero:INTEGER; bits: POSITIVE ) RETURN BIT_VECTOR IS
  VARIABLE binario      : BIT_VECTOR ( bits -1 DOWNT0 0 );
  VARIABLE numero,      : INTEGER := 0;
  VARIABLE cociente : INTEGER := 0;
BEGIN
  numero := entero;
  FOR i IN binario'RANGE LOOP
    cociente := numero / ( 2 ** i );
    numero := numero REM ( 2** i );
    IF ( cociente = 1 ) THEN
      binario ( i ) := '1' ;
    ELSE
      binario ( i ) := '0' ;
    END IF;
  END LOOP;
  RETURN binario;
END entero_a_bin;
```

```
PROCEDURE enter_a_binar ( entero: IN INTEGER; binario : OUT BIT_VECTOR) IS
    VARIABLE temp : INTEGER;
BEGIN
    temp := entero;
    FOR i IN 0 TO (binario'LENGTH -1) LOOP
        IF ( temp MOD 2 = 1 ) THEN
            binario(i) := '1';
        ELSE
            binario(i) := '0';
        END IF;
        temp := temp / 2;
    END LOOP;
END enter_a_binar;
```

```
PROCEDURE tst_vect ( SIGNAL vectores : OUT BIT_VECTOR ;
    CONSTANT valores : IN serie;
    CONSTANT periodo : IN TIME) IS
    VARIABLE longitud : BIT_VECTOR ( vectores'RANGE );
BEGIN
    FOR i IN valores'RANGE LOOP
        enter_a_binar ( valores(i), longitud );
        vectores <= longitud AFTER i* periodo ;
    END LOOP;
END tst_vect;
```

```
PROCEDURE bustest (SIGNAL busX : IN cuad_vector; SIGNAL error: OUT BOOLEAN) IS
    VARIABLE NOZ : BOOLEAN := FALSE;
BEGIN
    FOR i IN busX'RANGE LOOP
        IF busX(i) /= 'Z' THEN
            NOZ := TRUE;
            RETURN;
        END IF;
    END LOOP;
    error <= NOZ;
END;

END modelos;
```