Tipos de Datos en VHDL

Tipos de Datos

Advertencia. Para escribir código VHDL en forma eficiente, es indispensable conocer los tipos de datos permitidos, como especificarlos y como usarlos.

En VHDL los tipos de datos se dividen en dos categorías:

- Predefinidos. Los primeros son parte del estándar VHDL y están disponibles a través de las bibliotecas.
- Definidos por el usuario. Son creado por el usuario para manejar situaciones especiales.

Tipos de Datos

VHDL incluye los siguientes tipos de datos:

- Tipos enumerados predefinidos
- Tipos enumerados definidos por el usuario
- Tipos de vector de bits
- Tipos de enteros
- Tipos de arreglos multidimensionales
- Tipos de registros

Clasificaciones

Para ayudar a visualizar la naturaleza y alcance de los tipos de datos, estos se clasifican de varias formas:

- Fuente de la declaración.
  - ✓ Predefinidos. Definidos en el estándar.
  - ✓ Definidos por el usuario. Creados por el usuario.

- Acorde a la naturaleza de sus elementos
  - ✓ Tipos numéricos: enteros, punto fijo y punto flotante.

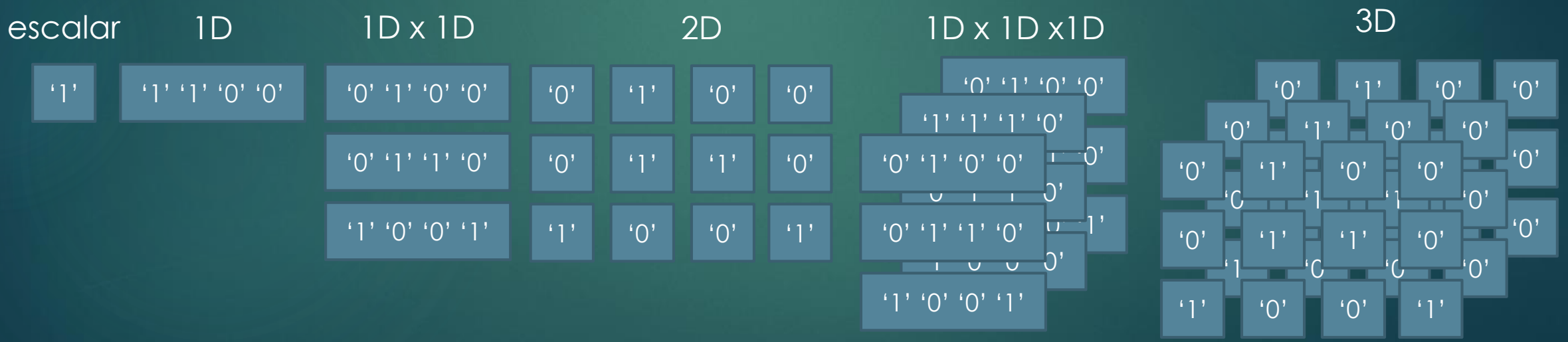
Clasificaciones

- Acorde al número de elementos
  - ✓ Escalar. Un solo elemento (cualquier tipo de dato).
  - ✓ Compuesto. Esta categoría incluye dos categorías llamadas *array* (colección de elementos de un mismo tipo de dato) y *record* (colección de escalares y/o arreglos de elementos que pueden ser de diferentes tipos de datos).

- Acorde al número de bits
  - ✓ Escalar. Un solo bit.
  - ✓ 1D array. Un vector de bits. Ejemplos: “01000”, “111100ZZ”, 255, ‘A’.
  - ✓ !D x 1D array. Un pila de vectores de bit. Ejemplo: (“0000”, “0111”).

Clasificaciones

- ✓ 2D array. Una matriz de bits. Ejemplo: (('0','1','0'),('0','0','0'),('Z','1','Z').
- ✓ 1D x 1D x 1D. Un bloque de vectores.
- ✓ 3D array. Un bloque de bits.



Clasificaciones

- Acorde al paquete de origin
  - ✓ Tipos estándar. Paquete *standard* de la biblioteca *std*.
  - ✓ Tipo estándar-lógico. Paquete *std_logic_1164* de la biblioteca *ieee*.
  - ✓ Tipo con signo/sin signo. Paquete *numeric_std* de la biblioteca *ieee* o paquete *std_logic_arith*.
  - ✓ Tipo punto fijo/punto flotante. Paquete *fixed_pkg* y *float_pkg*.

Tipos de Datos Estándar

Los siguientes tipos de datos son sintetizables y están incluidos en el paquete *standard*.

- BIT. Es un tipo enumerado de dos valores: 0 (cero lógico) y 1 (uno lógico). Soporta operaciones lógicas y de comparación. En términos de números de bits es un escalar. Su definición es la siguiente

```
TYPE BIT IS ('0', '1');
```

Ejemplo

```
SIGNAL a, x, y: BIT;
X <= '1';
Y <= NOT a;
```

Tipos de Datos Estándar

- BIT_VECTOR. Es un vector formado por elementos de tipo BIT. Soporta operaciones lógicas, de comparación, de corrimiento y concatenación. Su definición es la siguiente

```
TYPE BIT_VECTOR IS ARRAY (NATURAL RANGE <>) OF BIT;
```

Nota. La expresión NATURAL_RANGE <> es una especificación que indica que el rango no tiene restricciones. La única condición que debe cumplir es que los valores estén en el rango NATURAL (por defecto de 0 a 2^{31}-1).

Para las operaciones lógicas y de corrimiento es necesario que los vectores sean del mismo tamaño, no así con la operación de comparación.

Tipos de Datos Estándar

Ejemplo

```
SIGNAL a: BIT_VECTOR(7 downto 0);
X <= “11110000”;
```

- BOOLEAN. Es un tipo enumerado de dos valores: false y true. Soporta operaciones lógicas y de comparación. En términos de números de bis es un escalar. Su definición es

```
TYPE BOOLEAN IS (FALSE, TRUE);
```

Tipos de Datos Estándar

Ejemplo

```
SIGNAL ready: BOOLEAN;
x <= "111" WHEN ready ELSE "000";
```

El valor de x cambia de "000" a "111" cuando *ready* es igual a TRUE.

- INTEGER. Soporta operaciones aritméticas y de comparación. Su rango por defecto va de $-(2^{31}-1)$ a $(2^{31}-1)$. Su definición es

```
TYPE INTEGER IS RANGE implementation_defined;
TYPE INTEGER IS -2147483647 a 2147483647;
```

Tipos de Datos Estándar

Los limites son referidos como INTERGER´LOW (limite inferior) y INTEGER´HIGH (limite superior).

Nota. Es importante especificar el rango cada vez que se usa este tipo de dato, de lo contrario el sintetizador emplea 32 bits para representarlo.

Ejemplo

```
SIGNAL a: INTEGER RANGE 0 TO 15;          -- 4 bits
SIGNAL b: INTEGER RANGE -15 TO 15;        -- 5 bits
SIGNAL x: INTENGER RANGE -31 TO 31;       -- 6 bits
X <= a + b;
```

Tipos de Datos Estándar

- NATURAL. Enteros no negativos. Es un subtipo del tipo INTEGER y soporta las mismas operaciones. Su definición es

```
SUBTYPE NATURAL IS INTEGER 0 TO INTEGER´HIGH;
```

- POSITIVE. Enteros positivos. Es un subtipo del tipo INTEGER y soporta las mismas operaciones. Su definición es

```
SUBTYPE POSITIVE IS INTEGER 1 TO INTEGER´HIGH;
```

- INTEGER_VECTOR. Es un vector de INTEGER (arreglo 1D x 1D). Soporta operaciones de concatenación y comparación. Su definición es

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

Tipos de Datos Estándar

- CHARACTER. Es un tipo enumerado de 256 símbolos. Sólo soporta operaciones de comparación. Su definición (parcial) es

```
TYPE CHARACTER IS (NUL, SOH, …, '0', '1', '2', …, 'ÿ';
```

Nota. NUL = "00000000" y ÿ = "11111111". Los primeros 128 símbolos corresponden al código ASCII regular.

Ejemplo

```
SIGNAL char1, char2: CHARACTER;
SIGNAL outp1: BIT;
Outp1 <= '1' WHEN char1 = 'a' OR char1='A' ELSE '0';
```

Tipos de Datos Estándar

- STRING. Es un vector de CHARACTER (arreglo de 1D x 1D). Soporta operaciones de comparación y concatenación. Su definición es

```
TYPE STRING IS ARRAY (POSITIVE RANGE <>) OF CHARACTER;
```

Ejemplo

```
SIGNAL str: STRING (1 TO 4);
SIGNAL output: BIT;
Output <= '1' WHEN str = "VHDL" ELSE '0';
```

Tipos de Datos Estándar-Lógico

Los tipos de datos estándar-lógico (*standard-logic*) son STD_LOGIC y STD_LOGIC_VECTOR. Ambos están definidos en la paquete *std_logic_1164*. STD_LOGIC y STD_LOGIC_VECTOR son el estándar en la industria.

La definición de STD_LOGIC está en función de STD_ULOGIC. El primero es un tipo de dato "resuelto" y el último es "no resuelto".

Su definición es

```
TYPE STD_ULOGIC IS ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
TYPE STD_LOGIC IS resolved STD_ULOGIC;
```

Tipos de Datos Estándar-Lógico

El significado y posible uso para los nueve símbolos STD_(U)LOGIC.

Valor	Significado
‘U’	No inicializado (Uninitialized)
‘X’	Desconocido (Forcing unknown)
‘0’	Bajo (Forcing low)
‘1’	Alto (Forcing high)
‘Z’	Alta impedancia (High impedance)
‘W’	Desconocido débil (Weak unknown)
‘L’	Bajo débil (Weak low)
‘H’	Alto débil (Weak high)
‘-’	No importa (Don´t care)

Tipos de Datos Estándar-Lógico

La principal característica de STD_LOGIC, comparado con el tipo de dato BIT, es la inclusión de los valores 'Z' y '-'. El primero permite la construcción de buffer de tercer estado y el segundo una mejor optimización de tablas de búsqueda.

Se dice que STD_LOGIC es un tipo de dato "resuelto" ya que si más de una fuente maneja un nodo común el resultado del nivel lógico está determinado por una función de resolución.

A continuación se muestra la función de resolución para STD_LOGIC transcrita del paquete *std_logic_1164*.

Tipos de Datos Estándar-Lógico

```
1  ----------------------------------------------------------------
2  CONSTANT resolution_table : stdlogic_table := (
3  ----------------------------------------------------------
4  --U X 0 1 Z W L H - | |
5  ----------------------------------------------------------
6   ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
7   ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
8   ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
9   ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
10  ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
11  ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
12  ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
13  ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
14  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ) -- | - |);
15  ----------------------------------------------------------
16  FUNCTION resolved (s: STD_ULOGIC_VECTOR) RETURN STD_ULOGIC IS
17     VARIABLE result: STD_ULOGIC_ := 'Z'; --weakest state default
18     ATTRIBUTE synthesis_return OF result: VARIABLE IS "WIRED_THREE_STATE";
19  BEGIN
20     IF (s'LENGTH=1) THEN RETURN s(s'LOW);
21     ELSE
22        FOR i IN s'RANGE LOOP
23           result := resolution_table(result, s(i));
24        END LOOP;
25     END IF;
26     RETURN result;
27  END resolved;
```

Tipos de Datos Estándar-Lógico

El nombre la función es *resolved* y recibe un parámetro llamado *s* de tipo STD_ULOGIC_VECTOR, y regresa un tipo de dato STD_LOGIC.

La variable *result* tiene un valor inicial igual a 'Z'. Si el tamaño de *s* es uno entonces se regresa *s* (no hay nada que resolver).

En caso de que *s* contenga más de un bit entonces la tabla se usa para determinar el valor ganador. Por ejemplo, si los valores que compiten son '0', '1', 'Z', entonces el valor resultante es 'X' ('Z' vs '0' = '0', '0' vs '1' = 'X', 'X' vs 'Z' = 'X').

Tipos de Datos Estándar-Lógico

Nota. Desde la perspectiva de diseño, lo importante acerca de este tipo de dato (STD_LOGIC) es que es sintetizable.

Esto es lo que ocurre en el proceso de síntesis:

- '0' y 'L' ambos son sintetizados como '0' (para entradas y salidas).
- '1' y 'H' ambos son sintetizados como '1' (para entradas y salidas).
- 'Z' es sintetizado como 'Z' (para salidas).
- Los otros elementos son sintetizados como '-' (no importa – para salidas).

Tipos de Datos Estándar-Lógico

STD_LOGIC_VECTOR define un vector de STD_LOGIC. Su definición es

```
TYPE STD_ULOGIC_VECTOR IS ARRAY (NATURAL RANGE <>) OF STD_ULOGIC;
TYPE STD_LOGIC_VECTOR IS ARRAY (NATURAL RANGE <>) OF STD_ULOGIC;
```

El paquete *std_logic_1164* define operadores lógicos para STD_(U)LOGIC y STD_(U)LOGIC_VECTOR. Si se declara el paquete *std_logic_(un)signed* en el código, entonces se permiten algunas operaciones aritméticas, de corrimiento y de comparación.

Tipos de Datos Con Signo y Sin Signo

Los tipos de datos sin singo (UNSIGNED) y con signo (SIGNED) están definidos en los paquetes *numeric_std* y *std_logic_arith*. Estos paquetes son parcialmente equivalentes, por lo tanto, solo uno de ellos puede declararse en el código.

El rango para el tipo de dato sin signo es de 0 a 2^N-1 (limitado a INTEGER'HIGH), donde N es el número de bits.

El rango para el tipo de dato con signo es de -2^{N-1} a 2^{N-1}-1 (limitado entre INTEGER'LOW y INTEGER'HIGH), y usa la representación en complemento a 2.

Tipos de Datos Con Signo y Sin Signo

Nota. Desafortunadamente, los nombre elegidos para los paquetes *std_logic_unsigned* y *std_logic_signed* pueden causar confusión. Estos paquetes únicamente definen los operadores (+, -, *, …) sin signo o con singo para STD_LOGIC_VECTOR, y no los tipos de datos SIGNED y UNSIGNED.

En otras palabras, los dos paquetes definen como el operador trata sus operandos. Por ejemplo. Suponga a y b como vectores std_logic y "+" como el operador suma. Si se usa el paquete *std_logic_unsgined,* el operador "+" trata a y b como datos sin signo. Por su parte, si se usa el paquete *std_loigic_signed,* el operador "+" trata a y b como datos con signo.

Tipos de Datos de Punto Fijo y Punto Flotante

- Punto Fijo. Los tipos de punto fijo son con signo y sin singo. Su definición es

```
TYPE UFIXED IS ARRAY (INTEGER RANGE <>) OF STD_LOGIC;      -- UNSIGEND
TYPE SFIXED IS ARRAY (INTEGER RANGE <>) OF STD_LOGIC;      -- SIGNED
```

Los paquete necesarios para usar estos tipos de datos en VHDL 2008 son

- ✓ fixed_pkg.vhdl
- ✓ fixed_generic_pkg.vhdl
- ✓ fixed_generic_pkg-body.vhdl
- ✓ Fixed_float_types.vhdl

Tipos de Datos de Punto Fijo y Punto Flotante

Para la versiones anteriores de VHDL (93 y 2002), son necesarios los siguientes archivos

- ✓ fixed_pkg_c.vhd
- ✓ fixed_float_types_c.vhd

Para implementar circuitos con punto fijo en la versiones previas de VHDL, es necesario crear una carpeta llamada ieee_proposed en el directorio de bibliotecas VHDL del compilador, y pegar los dos archivos antes mencionados.

Tipos de Datos de Punto Fijo y Punto Flotante

Ejemplo

```
x: SIGNAL UFIXED(2 DONWTO -3);   -- esto es "xxx.xxx"
X <= "100011";                   -- 1x2^2 + 0x2^1 + 0x2^0 + 0x2^-1 +1x2^-2 + 1x2^-3 = 4.375
```

```
x: SIGNAL SFIXED(2 DONWTO -3);   -- esto es "xxx.xxx"
X <= "100011";                   -- 100.011 -> en comp a 2 = -3.625
```

Tipos de Datos de Punto Fijo y Punto Flotante

- Punto Flotante. Los tipos de datos de punto flotante son flotante de 32, 64 y 128 bits. Su definición es

```
TYPE FLOAT IS ARRAY (INTEGER RANGE <>) OF STD_LOGIC; -- longitud genérica
SUBTYPE FLOAT32 IS FLOAT (8 DOWNTO -23);        -- 32 bits PF de la IEEE 754
SUBTYPE FLOAT64 IS FLOAT (11 DOWNTO -52);       -- 64 bits PF de la IEEE 754
SUBTYPE FLOAT128 IS FLOAT (15 DOWNTO -112);     -- 128 bits PF de la IEEE 754
```

Los archivos necesarios para usar estos tipos de datos en VHDL 2008 son

- ✓ float_pkg.vhdl
- ✓ float_generic_pkg.vhdl

Tipos de Datos de Punto Fijo y Punto Flotante

- ✓ float_generic_pkg-body.vhdl
- ✓ fixed_float_types.vhdl

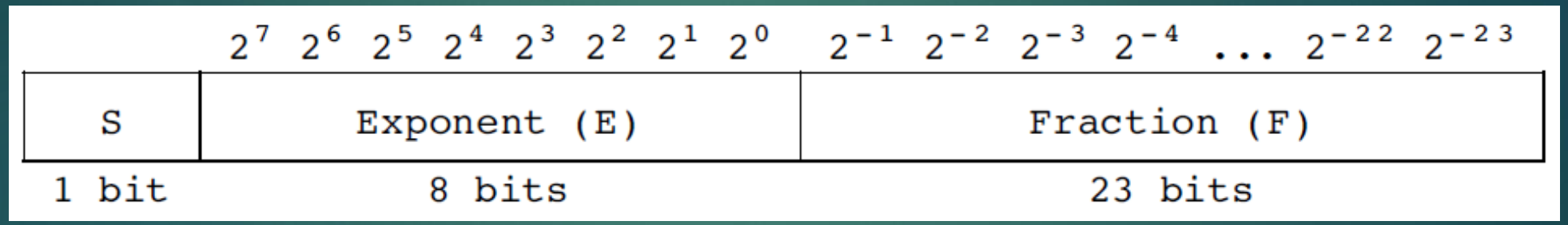
Para versiones previas de VHDL, se requieren lo siguientes archivos

- ✓ fixed_pkg_c.vhd
- ✓ fixed_float_types_c.vhd
- ✓ Float_pkg_c.vhd

Para implementar circuitos de punto flotante en la versiones previas de VHDL, solo hay que crear un carpeta llamada *ieee_proposed* en el directorio de bibliotecas de VHDL, y copiar los archivos antes mencionados.

Tipos de Datos de Punto Fijo y Punto Flotante

La representación de números en punto flotante (de 32 bits) que obedecen el estándar IEEE 754 es la siguiente



Sea x un número en punto flotante. Su valor esta dado por $x = (-1)^S(1+F)2^{E-N}$, donde S es el signo (0 cuando es positivo, 1 cuando es negativo), F es la fracción (o mantisa) y E es el exponente, y N es el factor de normalización dado por $N = (E_{MAX}+1)/2 - 1$.

Tipos de Datos de Punto Fijo y Punto Flotante

Ejemplo

```
-- 3 bits para exponente y 4 bits para la fracción
x <= "10010110";        -- (1)(001)(0100) = -(1+0.375)2^(1-3) = -0.34375
```

Nota. En VHDL, la longitud mínima de un número en punto flotante es de 7 bits.

Resumen de los Tipos de Datos Predefinidos

Todos los tipos de datos predefinidos de VHDL que son sintetizables se muestran a continuación.

Category	Package of origin	Predefined synthesizable types	Dimension
Standard	standard	BIT	Scalar
		BIT_VECTOR	1D
		BOOLEAN	Scalar
		INTEGER	1D
		NATURAL	1D
		POSITIVE	1D
		CHARACTER	1D
		STRING	1Dx1D
	standard (2008 expansion)	BOOLEAN_VECTOR	1D
		INTEGER_VECTOR	1DX1D
	numeric_bit_unsigned (2008)	(only operators for BIT and BV)	---

Resumen de los Tipos de Datos Predefinidos

Standard logic	std_logic_1164	STD_(U)LOGIC	Scalar
		STD_(U)LOGIC_VECTOR	1D
	std_logic_1164 (2008 expansion)	STD_(U)LOGIC	Scalar
		STD_(U)LOGIC_VECTOR	1D
	std_logic_unsigned	(only operators for SLV)	---
	std_logic_signed	(only operators for SLV)	---
	numeric_std_unsigned (2008)	(only operators for SL and SLV)	---
Unsigned and Signed	numeric_bit	UNSIGNED (base=BIT)	1D
		SIGNED (base=BIT)	1D
	numeric_std	UNSIGNED (base=STD_LOGIC)	1D
		SIGNED (base=STD_LOGIC)	1D
	std_logic_arith	UNSIGNED (base=STD_LOGIC)	1D
		SIGNED (base=STD_LOGIC)	1D
Fixed and Floating point	fixed_pkg + associated packages (2008)	UFIXED	1D
		SFIXED	1D
	float_pkg + assoc. pack. (2008)	FLOAT	1D

Tipos de Datos Definidos por el Usuario

Los tipos revisados hasta ahora son tipos de datos predefinidos. Sin embargo, VHDL permite el uso de tipos de datos definidos por el usuario. A continuación hacemos una revisón.

- Tipos Enteros. El tipo INTEGER es sintetizable sin restricciones. Todos los tipos derivados de INTEGER son referidos con tipos enteros, y pueden ser declarados usando la siguiente sintaxis

```
TYPE type_name IS RANGE range_specifications;
```

Tipos de Datos Definidos por el Usuario

Ejemplos

```
TYPE negative IS RANGE INTEGER'LOW TO -1:
TYPE temperatura IS RANGE 0 TO 273;
TYPE my_integer IS RANGE -32 to 32;
```

- Tipos Enumerados. En este caso, los valores de este tipo son representados por los símbolos, lo cuales pueden ser enlistados explícitamente (enumeración). Este tipo de dato fue usado para la creación de varios tipos de datos predefinidos.

Tipos de Datos Definidos por el Usuario

Los tipos enumerados son particularmente útiles en la creación para otros sistemas lógicos y en el diseño de máquinas de estados finitos. Su sintaxis es la siguiente

```
TYPE type_name IS (type_value_list);
```

Ejemplos

```
TYPE BIT IS ('0', '1');
TYPE BOOLEAN IS (FALSE, TRUE);
TYPE STD_ULOGIC IS ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X');
--------------------------------------------------------------------------------
TYPE logic_01Z IS ('0', '1', 'Z');
TYPE state IS (A, B, C, D, E);
TYPE machine_state IS (idle, transmiting, receiving);
```

Tipos de Datos Definidos por el Usuario

- Arreglos. En este caso se divide en dos categorías
  - ✓ Enteros. El tipo de elementos (enteros) es un subtipo del nuevo tipo. Su definición es

```
TYPE type_name IS ARRAY (range_specs) OF int_elements_type;
```

Ejemplo. Arreglo 1D x 1D

```
TYPE type1 IS ARRAY (POSITIVE RANGE <>) OF INTEGER;
CONSTANT const1: type1(1 TO 4) := (5, -5, 3, 0);
--------------------------------------------------------------------------------
TYPE type2 IS ARRAY (0 TO 3) OF NATURAL;
CONSTANT const2: type2 := (2, 0, 9, 4);
```

Tipos de Datos Definidos por el Usuario

Ejemplo. Arreglo 1D x 1D x 1D

```
TYPE type2 IS ARRAY (0 TO 3) OF NATURAL;
TYPE type3 IS ARRAY (1 TO 2) OF type2;
CONSTANT const3: type3 := ( (5, 5, 7, 9), (33, 4, 0, 0) );
```

- ✓ Enumerados. Es tipo requiere por lo general estar restringido. Su definición es

```
TYPE type_name IS ARRAY (range_specs) OF enum_elements_type;
```

Tipos de Datos Definidos por el Usuario

Ejemplo. Arreglo 1D

```
TYPE type1 IS ARRAY (NATURAL RANGE <>) OF STD_LOGIC;
CONSTANT const1: type1(4 DOWNTO 0) := "Z111";
--------------------------------------------------------------------------------
TYPE type2 IS ARRAY (7 DOWNTO 0) OF BIT;
CONSTANT const2: type2 := "00001111";
```

Ejemplo. Arreglo 1D x 1D

```
TYPE type4 IS ARRAY (1 TO 4) OF STD_LOGIC_VECTOR(2 DOWTON 0);
CONSTANT const4: type4 := ("000", "011", "100", "100");
CONSTANT const4: type4 := (('0', '0', '0'), ('0', '1', '1'), ...);
```

Tipos de Datos Definidos por el Usuario

Ejemplo. Arreglo 2D

```
TYPE type5 IS ARRAY (1 TO 3, 1 TO 4) OF BIT;
CONSTANT const5: type5 := (("0000", "0000", "0000");
```

Ejemplo. Arreglo 3D

```
TYPE type7 IS ARRAY (1 TO 2, 1 TO 3, 1 TO 4) OF BIT;
CONSTANT const7: type7 := (("0000", "0000", "0000"), ("0000",
"0000", "0000"));
```

Tipos de Datos Definidos por el Usuario

Para acceder a los elementos de un arreglo es necesario utilizar ().

Ejemplo.

```
TYPE int_row IS RANGE 1 TO 3;
TYPE enum_colum IS ('a', 'b', 'c', 'd');
TYPE matrix IS ARRAY (int_row, enum_colum) OF STD_LOGIC;
CONSTANT my_array: matrix := ("Z101", "0011", "101Z");
-------------------------------------------------------------------------------------------
my_array(1, 'a') -> 'Z';
my_array(1, 'b') -> '1';
my_array(1, 'c') -> '0';
...
my_array(3, 'd') -> 'Z';
```

Registros

Los registros (records) son colecciones de elementos que pueden ser de diferentes. Los tipos pueden ser predefinidos o definidos por el usuario.

Ejemplo

```
TYPE memory_Access IS RECORD
        address: INTEGER RANGE 0 TO 255;
        block: INTEGER RANGE 0 TO 3;
        data: BIT VECTOR (15 DOWNTO 0);
END RECORD;
```

Subtipos

Un subtipo (SUBTYPE) es un tipo de dato con restricciones. La principal razón de usar un subtipo en vez de especificar un nuevo tipo es que, aunque las operaciones entre diferentes tipos de datos no están permitidos, si los están entre el subtipo y el tipo del cual de derivo el subtipo.

Un subtipo (SUBTYPE) puede ser declarado en el mismo lugar que el tipo (TYPE), pero usualmente se hace en la parte declarativa de la arquitectura (ARCHITECTURE).

Subtipos

Ejemplo

```
TYPE _STD_LOGIC IS (‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘-’);
SUBTYPE my_logic IS STD_LOGIC RANGE ‘0’ TO ‘Z’;
--------------------------------------------------------------------------------
TYPE color IS (red, green, blue, white);
SUBTYPE my_color IS color RANGE green TO blue;
```

Conversión de Tipos de Datos

La conversión de tipos se divide en la siguientes categorías

- Conversión Automática. Este es el caso cuando se trata directamente con la base del tipo (*base* type). Por ejemplo. BIT y BIT_VECTOR tiene la misma base de tipo (BIT), así que un solo elemento BIT_VECTOR es automáticamente compatible con un elemento de tipo BIT.

Ejemplo

```
bv(0) <= b;
slv(7) <= s1;
```

Conversión de Tipos de Datos

- Casting. UNSIGNED/SIGNED tiene la misma base de tipo (STD_LOGIC) e indexado (NATURAL) como STD_LOGIC_VECTOR. La conversión entre estos tipos se hace de la siguiente manera
  - ✓ (UN)SIGNED(arg), donde el argumento es STD_LOGIC_VECTOR
  - ✓ STD_LOGIC_VECTOR(arg), donde el argumento es SIGNED o UNSGINED

Ejemplo

```
unsig <= UNSIGNED(slv);
sig <= SIGNED(slv);
slv1 <= STD_LOGIC_VECTOR(unsig);
slv2 <= STD_LOGIC_VECTOR(sig);
```

Conversión de Tipos de Datos

- Funciones de Conversión de Tipo. La última opción de conversión directa entre tipos es usando funciones de conversión de tipo, disponibles en los paquetes VHDL.

From	To	Type conversion function	Package of origin
INTEGER	STD_LOGIC_VECTOR	conv_std_logic_vector(a, cs)	std_logic_arith
	UNSIGNED	to_unsigned(a, cs) conv_unsigned(a, cs)	numeric_std std_logic_arith
	SIGNED	to_signed(a, cs) conv_signed(a, cs)	numeric_std std_logic_arith
	UFIXED	to_ufixed(a, cs)	fixed_generic_pkg
	SFIXED	to_sfixed(a, cs)	fixed_generic_pkg
	FLOAT	to_float(a, cs)	float_generic_pkg
BIT_VECTOR	STD_LOGIC_VECTOR	to_stdlogicvector(a, cs)	std_logic_1164

Conversión de Tipos de Datos

STD_LOGIC_VECTOR	INTEGER	conv_integer(a, cs) conv_integer(a, cs) to_integer(a, cs)	std_logic_signed std_logic_unsigned numeric_std_unsigned
	BIT_VECTOR	to_bitvector(a, cs)	std_logic_1164
	UNSIGNED	unsigned(a) (*) unsigned(a) (*)	numeric_std std_logic_arith
	SIGNED	signed(a) (*) signed(a) (*)	numeric_std std_logic_arith
	UFIXED	to_ufixed(a, cs)	fixed_generic_pkg
	SFIXED	to_sfixed(a, cs)	fixed_generic_pkg
	FLOAT	to_float(a, cs)	float_generic_pkg
UNSIGNED and SIGNED	INTEGER	to_integer(a, cs) conv_integer(a, cs)	numeric_std std_logic_arith
	STD_LOGIC_VECTOR	std_logic_vector(a) (*) std_logic_vector(a) (*) conv_std_logic_vector(a, cs)	numeric_std std_logic_arith std_logic_arith
	UNSIGNED	conv_unsigned(a, cs)	std_logic_arith
	SIGNED	conv_signed(a, cs)	std_logic_arith
	UFIXED (unsigned only)	to_ufixed(a, cs)	fixed_generic_pkg
	SFIXED (signed only)	to_sfixed(a, cs)	fixed_generic_pkg
	FLOAT	to_float(a, cs)	float_generic_pkg

Conversión de Tipos de Datos

UFIXED and SFIXED	INTEGER	to_integer(a, cs)	fixed_generic_pkg
	STD_LOGIC_VECTOR	to_slv(a, cs)	fixed_generic_pkg
	UNSIGNED (ufixed only)	to_unsigned(a, cs)	fixed_generic_pkg
	SIGNED (sfixed only)	to_signed(a, cs)	fixed_generic_pkg
	SFIXED (ufixed only)	to_sfixed(a, cs)	fixed_generic_pkg
	FLOAT	to_float(a, cs)	float_generic_pkg
FLOAT	INTEGER	to_integer(a, cs)	float_generic_pkg
	STD_LOGIC_VECTOR	to_slv(a, cs)	float_generic_pkg
	UNSIGNED	to_unsigned(a, cs)	float_generic_pkg
	SIGNED	to_signed(a, cs)	float_generic_pkg
	UFIXED	to_ufixed(a, cs)	float_generic_pkg
	SFIXED	to_sfixed(a, cs)	float_generic_pkg

(a, cs) = (argument, conversion specifications)
cs may include vector size, left/right range constants, overflow and rounding specs, etc. (consult package)
(*) = type casting