Useful IEEE packages, arithmetic operations, data types and conversion functions

This would be a perfect time for carefully reading or re-reading sections 3.7 and 3.8 of the textbook: Volnei A. Pedroni, *Circuit Design with VHDL*, MIT Press.

Here is a couple of packages worth to learn how to use:

- ieee.std_logic_arith
- ieee.std_logic_unsigned
- ieee.std_logic_signed

Or alternatively:

- ieee.numeric_std

3.7 Signed and Unsigned Data Types

As mentioned earlier, these types are defined in the `std_logic_arith` package of the `ieee` library. Their syntax is illustrated in the examples below.

Examples:

```vhdl
SIGNAL x: SIGNED (7 DOWNTO 0);
SIGNAL y: UNSIGNED (0 TO 3);
```

Notice that their syntax is similar to that of STD_LOGIC_VECTOR, not like that of an INTEGER, as one might have expected.

An UNSIGNED value is a number never lower than zero. For example, “0101” represents the decimal 5, while “1101” signifies 13. If type SIGNED is used instead, the value can be positive or negative (in two’s complement format). Therefore, “0101” would represent the decimal 5, while “1101” would mean \(-3\).

To use SIGNED or UNSIGNED data types, the `std_logic_arith` package, of the `ieee` library, must be declared. Despite their syntax, SIGNED and UNSIGNED data types are intended mainly for arithmetic operations, that is, contrary to
STD_LOGIC_VECTOR, they accept arithmetic operations. On the other hand, logical operations are not allowed. With respect to relational (comparison) operations, there are no restrictions.

Example: Legal and illegal operations with signed/unsigned data types.

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all; -- extra package necessary

SIGNAL a: IN SIGNED (7 DOWNTO 0);
SIGNAL b: IN SIGNED (7 DOWNTO 0);
SIGNAL x: OUT SIGNED (7 DOWNTO 0);

v <= a + b; -- legal (arithmetic operation OK)
w <= a AND b; -- illegal (logical operation not OK)

Example: Legal and illegal operations with std_logic_vector.

LIBRARY ieee;
USE ieee.std_logic_1164.all; -- no extra package required

SIGNAL a: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);

v <= a + b; -- illegal (arithmetic operation not OK)
w <= a AND b; -- legal (logical operation OK)

Despite the constraint mentioned above, there is a simple way of allowing data of type STD_LOGIC_VECTOR to participate directly in arithmetic operations. For that, the ieee library provides two packages, std_logic_signed and std_logic_unsigned, which allow operations with STD_LOGIC_VECTOR data to be performed as if the data were of type SIGNED or UNSIGNED, respectively.

Example: Arithmetic operations with std_logic_vector.

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all; -- extra package included...
SIGNAL a: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
...

v <= a + b;       -- legal (arithmetic operation OK), unsigned
w <= a AND b;    -- legal (logical operation OK)

3.8 Data Conversion

VHDL does not allow direct operations (arithmetic, logical, etc.) between data of different types. Therefore, it is often necessary to convert data from one type to another. This can be done in basically two ways: or we write a piece of VHDL code for that, or we invoke a FUNCTION from a pre-defined PACKAGE which is capable of doing it for us.

If the data are closely related (that is, both operands have the same base type, despite being declared as belonging to two different type classes), then the std_logic_1164 of the ieee library provides straightforward conversion functions. An example is shown below.

Example: Legal and illegal operations with subsets.

TYPE long IS INTEGER RANGE -100 TO 100;
TYPE short IS INTEGER RANGE -10 TO 10;
SIGNAL x : short;
SIGNAL y : long;
...
y <= 2*x + 5;       -- error, type mismatch
y <= long(2*x + 5); -- OK, result converted into type long

Several data conversion functions can be found in the std_logic_arith package of the ieee library. They are:

• conv_integer(p) : Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_LOGIC to an INTEGER value. Notice that STD_LOGIC_VECTOR is not included.

• conv_unsigned(p, b): Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_LOGIC to an UNSIGNED value with size b bits.

• conv_signed(p, b): Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_LOGIC to a SIGNED value with size b bits.
More on the subject:

- `conv_std_logic_vector(p, b)`: Converts a parameter `p` of type INTEGER, UNSIGNED, SIGNED, or STD_LOGIC to a STD_LOGIC_VECTOR value with size `b` bits.

Example: Data conversion.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
...
SIGNAL a: IN UNSIGNED (7 DOWNTO 0);
SIGNAL b: IN UNSIGNED (7 DOWNTO 0);
SIGNAL y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
...
y <= CONV_STD_LOGIC_VECTOR ((a+b), 8);
-- Legal operation: a+b is converted from UNSIGNED to an
-- 8-bit STD_LOGIC_VECTOR value, then assigned to y.
```

Another alternative was already mentioned in the previous section. It consists of using the `std_logic_signed` or the `std_logic_unsigned` package from the `ieee` library. Such packages allow operations with STD_LOGIC_VECTOR data to be performed as if the data were of type SIGNED or UNSIGNED, respectively.

Besides the data conversion functions described above, several others are often offered by synthesis tool vendors.

More on the subject:

<table>
<thead>
<tr>
<th>Data type of <code>a</code></th>
<th>To data type</th>
<th>Conversion function/type casting</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned, signed</td>
<td>std_logic_vector</td>
<td>std_logic_vector(a)</td>
</tr>
<tr>
<td>signed, std_logic_vector</td>
<td>unsigned</td>
<td>unsigned(a)</td>
</tr>
<tr>
<td>unsigned, std_logic_vector</td>
<td>signed</td>
<td>signed(a)</td>
</tr>
<tr>
<td>unsigned, signed</td>
<td>integer</td>
<td>to_integer(a)</td>
</tr>
<tr>
<td>natural</td>
<td>unsigned</td>
<td>to_unsigned(a, size)</td>
</tr>
<tr>
<td>integer</td>
<td>signed</td>
<td>to_signed(a, size)</td>
</tr>
</tbody>
</table>
3.2.1 Relational operators

Six relational operators are defined in the VHDL standard: = (equal to), /= (not equal to), < (less than), <= (less than or equal to), > (greater than), and >= (greater than or equal to). These operators compare operands of the same data type and return a value of the boolean data type. In the book, we will use the boolean data type directly, and it is assumed that all other data types are converted to boolean data types when the comparison is made.

3.2.2 Arithmetic operators

In the VHDL standard, arithmetic operations are defined for the integer data type and for the natural data type, which is a subtype of integer containing zero and positive integers. We usually prefer to have more control in synthesis and define the exact number of bits and format (i.e., signed or unsigned). The IEEE numeric_std package is developed for this purpose. In this book, we use the integer and natural data types for constants and array boundaries but not for synthesis.

IEEE numeric_std package The IEEE numeric_std package adds two new data types, unsigned and signed, and defines the relational and arithmetic operators over the new data types (known as operator overloading). The unsigned and signed data types are defined as an array with elements of the std_logic data type. The array is interpreted as the binary representation of unsigned or signed integers. We have to add an additional use statement to invoke the package:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- invoke numeric_std package
```

Type conversion Because VHDL is a strongly typed language, std_logic_vector, unsigned, and signed are treated as different data types even when all of them are defined as an array with elements of the std_logic data type. A conversion function or type casting is needed to convert signals of different data types. The conversion is summarized in Table 3.3. Note that the std_logic_vector data type is not interpreted as a number and thus cannot be converted directly to an integer, and vice versa.

The following examples illustrate the common mistakes and remedies for type conversion. Assume that some signals are declared as follows:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

signal s1, s2, s3, s4, s5, s6 : std_logic_vector(3 downto 0);
```
signal u1, u2, u3, u4, u5, u6, u7: unsigned(3 downto 0);

Let us first consider the following assignment statements:

\[
\begin{align*}
&u1 := \text{const}; \quad \text{-- not ok, type mismatch} \\
&u2 := 5; \quad \text{-- not ok, type mismatch} \\
&s2 := u3; \quad \text{-- not ok, type mismatch} \\
&s3 := 5; \quad \text{-- not ok, type mismatch}
\end{align*}
\]

They are all invalid because of type mismatch. The right-hand-side expression must be converted to the data type of the left-hand-side signal:

\[
\begin{align*}
&u1 := \text{unsigned}(\text{const}); \quad \text{ok, type casting} \\
&s2 := \text{to unsigned}(5,4); \quad \text{ok, conversion function} \\
&s2 := \text{std_logic_vector}(u3); \quad \text{ok, type casting} \\
&s3 := \text{std_logic_vector}(\text{to unsigned}(5,4)); \quad \text{ok}
\end{align*}
\]

Note that two type conversions are needed for the last statement.

Let us consider statements that involve arithmetic operations. The following statements are valid since the + operator is defined with the unsigned and natural types in the IEEE numeric std package.

\[
\begin{align*}
&u4 := u2 + u1; \quad \text{-- ok, both operands unsigned} \\
&u5 := u2 + 1; \quad \text{-- ok, operands unsigned and natural}
\end{align*}
\]

On the other hand, the following statements are invalid since no overloaded arithmetic operation is defined for the std_logic_vector data type:

\[
\begin{align*}
&s5 := s2 + u1; \quad \text{-- not ok, + undefined over the types} \\
&s6 := s2 + 1; \quad \text{-- not ok, - undefined over the types}
\end{align*}
\]

To fix the problem, we must convert the operands to the unsigned (or signed) data type, perform addition, and then convert the result back to the std_logic_vector data type. The revised code becomes:

\[
\begin{align*}
&s5 := \text{std_logic_vector(unsigned}(u2) + \text{unsigned}(u1)); \quad \text{ok} \\
&s6 := \text{std_logic_vector(unsigned}(u2) + 1); \quad \text{ok}
\end{align*}
\]