THE XILINX HDL ADVISOR

Questions and Comments from Our Readers

by Roberta Fulton, Technical Marketing Engineer, Xilinx, roberta.fulton@xilinx.com

Your feedback concerning the previous columns of "HDL Advisor" has been most welcome. I am glad that so many of you have found this column helpful. While I can't answer each question personally I will address the most useful ones in this column.

Question 1: How can I assign an "integer" value (128) to an 8 bit "STD_LOGIC_VECTOR" defined signal?

Use the NUMERIC_STD package from the "IEEE Standard VHDL Synthesis Packages." The necessary functions, TO_SIGNED and TO_UNSIGNED, can be found on page 35 of the "IEEE Std 1076.3-1997 IEEE Standard VHDL Synthesis Packages."

If you don't have the IEEE standard manual itself, you may search in the NUMERIC_STD package that comes with your synthesis or simulation package for the function names.

```
– Id: D.3
 function TO_UNSIGNED (ARG: INTEGER; SIZE:
 NATURAL) return SIGNED is

– Id: D.4
 function TO_SIGNED (ARG: INTEGER; SIZE: NATURAL)
 return SIGNED ;
```

Make sure your signal (CNT in the example below) is declared either as a SIGNED or UNSIGNED type then use the appropriate function below:

```
CNT <= TO_SIGNED(128,8); or
CNT <= TO_UNSIGNED(128,8);
```

The opposite translation from SIGNED or UNSIGNED to INTEGER can be performed with the TO_INTEGER functions shown on the same page. First, declare CNT as an integer. Be sure to limit its range if you don't need the whole synthesis tool or simulation tool's default integer range. For most tools the range is 32 bits, with one bit for the sign.

Question 2: I assigned the range of an integer signal (CNT) from 0 to 15 and added 1 to this value, how can I make sure the compiler knows the value (CNT) is an unsigned value? e.g. CNT <= CNT + 1;

Declare the signal (CNT, in your example) as the UNSIGNED type as found in the NUMERIC_STD package from the "IEEE Std 1076.3-1997 IEEE Standard VHDL Synthesis Packages."

```
==================================================
– Numeric array type definitions
–
 =================================================

 type UNSIGNED is array (NATURAL range <>) of
 STD_LOGIC;
 type SIGNED is array (NATURAL range <>) of
 STD_LOGIC;
```

This package will also allow you to add the integer 1 to the unsigned value of CNT. Normally two different types cannot be added together in VHDL, but the NUMERIC_STD package allows you to add SIGNED or UNSIGNED types to integers because it has overloaded the operator. This means it does the type conversion for you.

```
– Id: A.8
 function "+" (L: SIGNED; R: INTEGER) return
 SIGNED;
– Result subtype: SIGNED(L'LENGTH-1 downto 0).
Result: Adds a SIGNED vector, L, to an INTEGER, R.

– Id: A.5
 function "+" (L: UNSIGNED; R: NATURAL) return
 UNSIGNED is
 begin
 return L + TO_UNSIGNED(R, L'LENGTH);
 end "+";
```

Please note that for an UNSIGNED number the integer must be natural.

Question 3: How can I compare a "STD_LOGIC_VECTOR" defined signal to a "integer" value? e.g. if (CNT = 38) then ...

If you used SIGNED or UNSIGNED types instead of STD_LOGIC_VECTOR, you could use the NUMERIC_STD package as referred to above for its overloaded relational operators for SIGNED, UNSIGNED and integer types. For example:

```
-- Id: C.29
function "=" (L:UNSIGNED; R: NATURAL) return
BOOLEAN is ...
```

You could also us the type conversion in the NUMERIC_STD pacakge. I don't recommend using more than one numerica package, even if you found another one with overloaded operators for STD_LOGIC_VECTOR and integer types. Some numeric packages in the public doman use the same types and overloaded operators as the IEEE standard. The reader can become confused which type and operator definitions are actually being used.

For clarity, I prefer that type conversions be stated separately rather than as part of the comparison statement itself as given below:

```
NEW_B <= TO_INTEGER(B);
```

Then you can use the new signal in the comparison as in:

```
if (NEW_B > 5) then …….
```

NOTE: Not all synthesis companies may be fully compliant with the IEEE Std 1076.3-1997. You may need to use another comparable numeric or arithmetic package instead.

The package STD_LOGIC_ARITH, created by Synopsys before the IEEE standard was available, is an example of a package that also contains the SIGNED and UNSIGNED types and the overloaded "+" operator. The STD_LOGIC_SIGNED, also from Synopsys, has the boolean comparison operators such as >=. If STD_LOGIC_SIGNED is used however, you must also use STD_LOGIC_ARITH which it calls. In that case NUMERIC_STD should not be used or you could get confused about which package is providing the operator definitions.

Theoretically, the packages should be portable no matter what their source. This is generally true of simulators. In practice, synthesizers may take awhile to adapt to another new package; when they do, they may have to adapt the source code of the package to the synthesis attributes. These attributes have no affect on the simulation.

Reader Comment 1: "... when it comes to VHDL coding approaches and synthesis results, it's very important to us to record the optimal approach(es) for both minimal area and minimal delay."

For the comparators I discuss in my *HDL Advisor Column* for *Xcell 31* (1Q99), the results of "best coding style" differ depending on whether you compile for area or delay.

The "fastest" coding style under one optimization such as "area" may not be the fastest when compiling for "delay." Note that some other variables are the software tool itself and the version of that tool. I am restricted by licensing agreements to not making direct comparisons between synthesizers. Therefore, I take a result from one tool, in the latest version available at the time, to get examples that fairly represent the results for all of the available tools.

For clarity, the article's "best results" were optimized for minimum area, and word_wise compares were faster. I generally try minimum area first as I find the results usually give close to the best delays without exploding the number of CLBs.

If minimum delay is selected, then bit-wise is faster with a sacrifice of using more CLBS.

"Best" really depends on how close you are to meeting area or performance constraints. For one design you may be willing to sacrifice a little area to tweak the performance, or give up a little performance in order to fit in a smaller size device.

As mentioned earlier, for synthesis tools especially, your results may vary. The algorithms change between vendors and even between versions from the same vendor.

I would advise you to try it yourself, with your tool set. And, don't forget that a different word-size causes different algorithms to kick in, so what is true for 4-bits may not be true for 32-bits.

Reader COMMENT 2: "Maybe you could have explained that this shouldn't work if you're following the VHDL-standard , but in practice it's ok for some synthesis vendors."

The reader notes that for the use of "-" as a "don't care" in VHDL from *HDL Advisor Column*, *Xcell 30*, I imply that the tools are broken, if they don't allow this.

As the reader states, the vendor that follows my example is a bit outside of the VHDL standard. They've taken the same approach as in Verilog. So, in practice, it works for some synthesis tools but it is wrong according to the VHDL-standard.

My tendency too is to be a standard-follower, but you must decide whether "standard" or "general practice" is best for you.

And as experienced HDL users have long noted, interpretations of the HDL standards may also be different between different synthesis and simulation vendors, especially IEEE Std 1087-1987 on which some vendors still rely, though the standard was updated in 1993. So, once again, try the examples in your choice of tools.