EE 231L

Using AHDL to Implement Functions

For a complex design, it is best to implement well-defined modules and design your circuit by connecting these modules together. A module will have a set of inputs and outputs, and logic which describes how to generate the outputs from the inputs. To use a module in a design, you need to know what the module does, and what inputs and outputs it has. You don't really need to know the logic which is used to implement the module.

A seven-segment LED is an array of seven LEDs which can be used to display numbers. A seven-segment LED decoder has four inputs and seven outputs — for an input between 0x0 and 0xF (hex 0 to hex F), the outputs will tell which of the segments should be lit to display that number. Here is an AHDL program which will implement a seven-segment LED decoder (modified from a design in Altera's **Max+PLUS II AHDL** manual):

```
%   -a-    %
% f|   |b  %
%   -g-    %
% e|   |c  %
%   -d-    %

SUBDESIGN 7segment
(
    i[3..0]             : INPUT;
    o[6..0]             : OUTPUT;
)
BEGIN
    TABLE
        i[3..0] => o[6..0];  % a b c d e f g %

        H"0"    => B"1111110";
        H"1"    => B"0110000";
        H"2"    => B"1111111";
        H"3"    => B"1111001";
        H"4"    => B"0110011";
        H"5"    => B"1011011";
        H"6"    => B"1011111";
        H"7"    => B"1110000";
        H"8"    => B"1111111";
        H"9"    => B"1111011";
        H"A"    => B"1110111";
        H"B"    => B"0011111";
        H"C"    => B"1001110";
        H"D"    => B"0111101";
        H"E"    => B"1001111";
        H"F"    => B"1000111";
    END TABLE;
```

```
END;
```

If you were designing a circuit which used four of these decoders, it would be tedious to write this code four times. You could instead use this code as a function in another TDF file. Here is an AHDL program which uses the above design file to implement a design to display a 16-bit binary number on a set of four seven-segment LEDs:

```
FUNCTION 7segment (i[3..0])
    RETURNS (o[7..0]);

SUBDESIGN 16display
(   in[15..0]             : INPUT;
    out1[6..0]            : OUTPUT;
    out2[6..0]            : OUTPUT;
    out3[6..0]            : OUTPUT;
    out4[6..0]            : OUTPUT;
)
VARIABLE
    dc1                    : 7segment;
    dc2                    : 7segment;
    dc3                    : 7segment;
    dc4                    : 7segment;
BEGIN
    dc1.i[3..0] = in[15..12];
    out1[6..0] = dc1.o[6..0];
    dc2.i[3..0] = in[11..8];
    out2[6..0] = dc2.o[6..0];
    dc3.i[3..0] = in[7..4];
    out3[6..0] = dc3.o[6..0];
    dc4.i[3..0] = in[3..0];
    out4[6..0] = dc4.o[6..0];
END;
```

This implements four seven-segment decoders as variables. The logic section then indicates how to connect the inputs and outputs of those decoders. A period (`.`) is used to identify the inputs and outputs of the function. For example, `dc1.i[3..0]` refers to the four input lines of the first decoder, and `dc1.o[6..0]` refers to the seven output lines of the first decoder.

Here is another way to do the same thing, using in-line functions:

```
FUNCTION 7segment (i[3..0])
    RETURNS (o[6..0]);

SUBDESIGN 16display
(   in[15..0]             : INPUT;
    out1[6..0]            : OUTPUT;
```

```
    out2[6..0]            : OUTPUT;
    out3[6..0]            : OUTPUT;
    out4[6..0]            : OUTPUT;
)
BEGIN
    out1[6..0] = 7segment(in[15..12]);
    out2[6..0] = 7segment(in[11..8]);
    out3[6..0] = 7segment(in[7..4]);
    out4[6..0] = 7segment(in[3..0]);
END;
```

This example is more C-like — the function `7segment` is called with its inputs as parameters, and its outputs are assigned to outputs in the `16display` design file.

You could have Quartus II create your function prototype for you. Open the file `7segment.tdf` (**File — Open**), then create an include file for the decoder (**File— Create/Update — Create AHDL Include Files for Current File**). This will create the file `7segment.inc`. This is especially useful for a fairly complicated module with lots of inputs and outputs, where it would be easy to make a typo if you were to enter the prototype yourself.

Here is the file `7segment.inc` which was created by Quartus II:

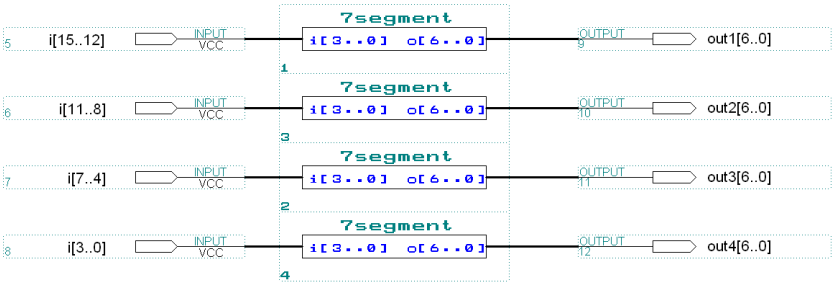
```
FUNCTION 7segment (i[3..0])
RETURNS (o[6..0]);
```

Your 16-bit display file would then look like:

```
INCLUDE "7segment.inc";

SUBDESIGN 16display
(   in[15..0]             : INPUT;
    out1[6..0]            : OUTPUT;
    out2[6..0]            : OUTPUT;
    out3[6..0]            : OUTPUT;
    out4[6..0]            : OUTPUT;
)
BEGIN
    out1[6..0] = 7segment(in[15..12]);
    out2[6..0] = 7segment(in[11..8]);
    out3[6..0] = 7segment(in[7..4]);
    out4[6..0] = 7segment(in[3..0]);
END;
```

You could also use your decoder in a graphics design file. Open the file `7segment.tdf` (**File — Open**), then create a symbol for the decoder (**File — Create Default Symbol**). This will make the file `7segment.sym` which you can then place in a graphics design file. Here is the 16-bit display as a graphics design file:


7segment
i[3..0] o[6..0]
5
i[15..12]
INPUT
VCC
OUTPUT
9
out1[6..0]
1
7segment
i[3..0] o[6..0]
6
i[11..8]
INPUT
VCC
OUTPUT
10
out2[6..0]
3
7segment
i[3..0] o[6..0]
7
i[7..4]
INPUT
VCC
OUTPUT
11
out3[6..0]
2
7segment
i[3..0] o[6..0]
8
i[3..0]
INPUT
VCC
OUTPUT
12
out4[6..0]
4