L-Università ta' Malta

DEPARTMENT OF COMMUNICATIONS AND COMPUTER ENGINEERING

Faculty of Information and Communication Technology

CCE1013—Computer Logic 1

Practical 3

Trevor Spiteri

trevor.spiteri@um.edu.mt

http://staff.um.edu.mt/trevor.spiteri

20 November, 2018

Multipliers

Objective

The objective of this practical is to investigate unsigned and signed multipliers.

Background

Unsigned multiplication

One simple way to perform unsigned multiplication in binary is to perform a simple long multiplication. For example, to perform $a \times b$, with $a = 13_{10} = 1101_2$ and $b = 10_{10} = 1010_2$, we have

	1 1 0 1				13
×	1 0 1 0			×	10
= 0×	~~0 0 0 0 1 1 0 1~~	$b_0 = 0$		= 0×	~~13~~
+1×	0 0 0 1 1 0 1 0	$b_1 = 1$		+1×	26
+0×	~~0 0 1 1 0 1 0 0~~	$b_2 = 0$		+0×	~~52~~
+1×	0 1 1 0 1 0 0 0	$b_3 = 1$		+1×	104
=	1 0 0 0 0 0 1 0			=	130

Algorithm 1: An algorithm to perform unsigned multiplication.

1: $total \leftarrow 0$
2: **for** $i \in \{0, 1, 2, \ldots, n-1\}$ **do**
3: $\quad row \leftarrow a \times 2^i$
4: $\quad$ **if** $b_i = 1$ **then**
5: $\quad\quad total \leftarrow total + row$
6: $\quad$ **end if**
7: **end for**

This method is described in Algorithm 1. Line 1 initializes the total to zero. Then, for each row i, (from 0 to 3 in the example), we find the value of the row that is optionally added to the total. This is done by shifting to the left by i, which is equivalent to multiplying by 2^i, as shown in Line 3. If the corresponding bit b_i is 1, the row is added to the total. In our example, $b_0 = 0$ and $b_2 = 0$, while $b_1 = 1$ and $b_3 = 1$.

Signed multiplication

Signed multiplication is similar, with two main differences:

1. The rows are not padded with zeros to the left, but are sign extended.
2. The last row is not added, but subtracted, since the most significant bit in two's-complement represents -2^i instead of 2^i.

Our example thus becomes $a \times b$ with $a = -3 = 1101_2$ and $b = -6 = 1010_2$.

	1 1 0 1			−3
×	1 0 1 0		×	−6
= 0×	1 1 1 1 1 1 0 1	$b_0 = 0$	= 0×	−3
+1×	1 1 1 1 1 1 0 1 0	$b_1 = 1$	+1×	−6
+0×	1 1 1 1 0 1 0 0	$b_2 = 0$	+0×	−12
−1×	1 1 1 0 1 0 0 0	$b_3 = 1$	−1×	−24
=	0 0 0 1 0 0 1 0		=	18

This is shown in Algorithm 2.

Tasks

1. Create a new project, and add the provided files *mult_unsigned.vhd* and *mult_unsigned_tb.vhd* to the project. This process is similar to

Algorithm 2: An algorithm to perform signed multiplication.

1: $total \leftarrow 0$
2: **for** $i \in \{0, 1, 2, \ldots, n-1\}$ **do**
3: $\quad row \leftarrow a \times 2^i$ {Note: sign extension is required}
4: $\quad$ **if** $b_i = 1$ **then**
5: $\quad\quad$ **if** $i = n-1$ **then** {if this is the most-significant bit}
6: $\quad\quad\quad total \leftarrow total - row$
7: $\quad\quad$ **else** {if this is *not* the most-significant bit}
8: $\quad\quad\quad total \leftarrow total + row$
9: $\quad\quad$ **end if**
10: $\quad$ **end if**
11: **end for**

what you did in Practical 2.

For the file *mult_unsigned.vhd*, set the association to *All*. For the file *mult_unsigned_tb.vhd*, set the association to *Simulation*.

2. Complete the file *mult_unsigned.vhd*. The main work is done inside the `shift_mux` process. Inside the process, **`for`** loops and **`if`** statements can be used as required. The synthesis tool will still be able to generate combinational logic hardware for the process.

   There are two parts you need to complete:

   (a) Set the contents of `row`. This corresponds to Algorithm 1, Line 3. In the given code, `row` is initialized to zero, then you need to find the indices of `row` (marked as `?what?`) so that the correct n bits are set to `a`.

   For example, for the first iteration $i = 0$, you should set $row_{n-1:0} \leftarrow a$, so the indices should evaluate to $n-1$ down to 0. For the last iteration $i = n-1$, you should set $row_{2n-2:n-1} \leftarrow a$, so the indices should evaluate to $2n-2$ down to $n-1$.

   (b) Write an expression for the condition to add the value of the row to the total. This corresponds to Algorithm 1, Line 4.

   Note: Your code should work for different values of n. Initially, $n = 8$, but if n is changed to $n = 16$, you should not need to change anything else.

3. Test the multiplier using the provided test bench. First you need to complete the test bench so that there are six possible hexadecimal values for each of the two multiplicands: 00, 01, $7F$, 80, 81, and FF.

Simulate the test bench and check that all the 36 possible products are correct. It can be helpful to set the radix of the simulation signals to a suitable value. The radix can be set by right-clicking on the signal name and selecting *Radix* ▸. Since this is unsigned multiplication, you can select *Unsigned Decimal*.

4. Measure the hardware area of the multiplier. In the *Implementation* view, select the multiplier. Then, in the *Processes* pane, double click on *Synthesize*. Find the design summary, and in the *Device Utilization Summary* read the *Number of Slice LUTs*. This number gives an indication of the hardware footprint of the design.

5. Obtain a measure of the hardware area for the range $2 \leq n \leq 32$. You can change n inside the generic part of the entity, and then use the method of Task 4 for each value of n. You do *not* need to do this for all 31 possible values of n.

6. Draw a graph of LUTs against n. Is the graph most similar to a linear curve ($y = ax$), a square curve ($y = ax^2$), or a cubic curve ($y = ax^3$)? What is the reason behind this?

7. Copy the file *mult_unsigned.vhd* to *mult_signed.vhd*. Add this new file to the project. In the new file, change any instance of `mult_unsigned` to `mult_signed`. Also, change the `shift_mux` process to perform signed multiplication instead of unsigned multiplication. This requires three steps:

   (a) Change all instances of `mult_unsigned` to `mult_signed`.

   (b) When setting the `row` variable, you need to sign-extend a. To do this, find the lines you modified in Task 2a.

   ```
   row := (others => '0');
   row(?hi? downto ?lo?) := a;
   ```

   Here `?hi?` and `?lo?` are place-holders for the indices you wrote. Just after these two lines, add the following line substituting the correct expression for `?hi?`.

   ```
   row(2*n-1 downto ?hi?+1) := (others => a(n-1));
   ```

   This copies the most significant bit of a to the sign extension bits.

   (c) While in unsigned multiplication the row is added to the total, in signed multiplication the row has to be either added to or subtracted from the total, as shown in Algorithm 2, Lines 5–9.

To do this, find the line

```
total := total + unsigned(row);
```

Replace this line with the following conditional statement, changing `?what?` to the correct condition:

```
if ?what? then
    total := total - unsigned(row);
else
    total := total + unsigned(row);
end if;
```

8. Copy the file *mult_unsigned_tb.vhd* to *mult_signed_tb.vhd*. Add this new file to the project. Again, inside the new file, change any instance of `mult_unsigned` to `mult_signed`.

   Simulate the new test bench and check that all the 36 possible products are correct. Since this is signed multiplication, the best choice of radix in the waveform window is now *Signed Decimal*.

9. Repeat Task 6 for the signed multiplier. To get the area for the signed multiplier, before double-clicking on *Synthesize*, you need to set `mult_signed` as the new Top Module. To do this, right-click on `mult_signed` and select *Set as Top Module*.

Report

Your report should describe your work concisely. It should include the VHDL snippets you wrote, and any observations you make. You should also show a couple of waveforms that verify that your design works, and the requested graphs.