Contents

1 Introduction 2
2 Multiplication 2
3 Creating the Multiplier 3
   3.1 Creating the Clock 4
   3.2 Creating the Multiplier 5
   3.3 Creating the Product 6
4 Testing the Multiplier 8
1 Introduction

This tutorial presents a number of new concepts including registers, multiplexors, busses, clocks and constants. If you have not yet done so, you should go through the first tutorial. At the very least read the sections on setting up SUE and make sure that you can run SUE. This tutorial walks you through the steps in creating a simple 8-bit multiplier. Section 2 presents a high level overview of the algorithm we will use. Following this, section 3 gives step-by-step instructions for creating a circuit to implement the multiplication algorithm. Finally, section 4 goes through a simple process for simulating and testing the circuit. As always, if you have any trouble with this tutorial, or find any bugs in it, please contact the author.

2 Multiplication

We need to create an algorithm to multiply two 8-bit binary numbers together. A good place to start is a review of the method we use to multiply numbers by hand. For example, multiplying 1010 by 1011 looks like:

\[
\begin{array}{c}
1010 \\
\times 1011 \\
\hline
1010 \\
1010 \\
0000 \\
1010 \\
\hline
1101110
\end{array}
\]

We will refer to the first operand as the multiplicand and the second as the multiplier. The result of the multiplication is called a product. Note that the product needs to be twice the size of the operands to avoid overflow. For the purposes of this tutorial we will ignore this and assume that the result fits in eight bits and that all numbers are positive, 8-bit binary values.

The inputs to our multiplier are two 8-bit binary numbers A and B. We will use \(a_i\) to refer to the \(i\)th bit of A and \(b_i\) to refer to the \(i\)th bit of B. Bits are numbered from 0 to 7, going from right to left (so \(a_0\) is the rightmost bit of A). With this in mind, a simple algorithm for multiplying A and B (with A as the multiplier and B as the multiplicand) is the following:

1. Initialize counter and product to zero.
2. If \(a_7 = 0\) go to 4, otherwise continue with 3.
3. Add \(B\) to the current product.
4. Shift \(A\) left one bit.
5. Shift \textit{product} left one bit.

6. \textit{counter} = \textit{counter} + 1

7. If \textit{counter} < 8 go to 2.

It takes eight steps to multiply two 8-bit numbers (one step for each bit in the multiplier). At each step the leftmost bit of the multiplier is checked. If it is a one, the multiplicand is added to the product, otherwise the product is unchanged. After this, the multiplier is shifted left by one to get the next bit and the product is shifted left by one. For the hardware version we will not be using a counter to keep track of the iteration. Rather, we will have a clock generator that resets after every eight cycles. It is for this reason that we are shifting \( A \) left in step 4 rather than just checking bit \( a_{7 - \text{counter}} \) in step 2.

3 Creating the Multiplier

![Diagram of multiplication circuit]

Figure 1: A high level view of the multiplication circuit.

Having created a multiplication algorithm, we will now use SUE to implement it. Figure 1 gives a high-level overview of the hardware components we will use to do this. These include registers, shifters and an adder. To get started, run SUE and select \textit{“new schematic...”} from the File menu. Name the schematic \textit{“mult8.sue”}. The sections below will take you through the steps necessary to create the actual multiplier circuit.
3.1 Creating the Clock

The algorithm we are using requires eight separate steps to multiply two 8-bit numbers. During each of these steps some combinational logic will be run to generate an intermediate result. That result will be stored in registers at the beginning of the next step and then used by the combinational logic to generate another set of intermediate results. We need a way to signal the start of each of these steps and also to store the intermediate results at each step. We can use a clock to solve the first of these problems. Select the clockGenerator icon from the second to bottom list box and place it in the top left hand corner of the schematic drawing area. The clock has two outputs: a reset signal that is asserted when the clock starts and a clock signal that oscillates at some frequency. The three properties of the clock are period, cycles and reset. The period controls the length of the clock cycle and we can safely leave it at 20. For a circuit with more combinational logic delay this might need to be larger. The cycles property gives the number of cycles that are executed before the reset signal is asserted again (i.e. before the clock restarts). The reset property reports how many periods the reset signal is asserted before the clock starts. We will set these values in the testing section below.

We are going to be creating a fairly large circuit with a number of components that need either the clock or reset signals. It would quickly get messy if we tried
to simply draw wires everywhere we needed these signals. Rather than do that, we can assign names to the wires and then just use the names wherever we need the signal. To do this select the name net's component from the bottom list box and place it to the right of the RST port on the clock. Double-click on the named net and type “reset” for the name. We would like the name to appear above the icon, rather than to the right of it, so hold the shift key and double click the named net icon. In the dialog box that comes up, select named net's. Now carefully move the mouse over the named net until you have highlighted the small inner rectangle in its middle. Press and hold the middle mouse button and drag this rectangle a little ways to the right. Now do the same with the small rectangle around the RST port on the clock, dragging it over the named net's port. Repeat the above steps to create a named net for the CLK port on the clock. Call this signal “clock”. See figure 2 for what this should look like.

3.2 Creating the Multiplier

![Multiplier Diagram](image)

Figure 3: The multiplier register and shifter.

The multiplier is one of the two operands in the multiplication. It is shifted left at the end of every step, and its high order bit is used to determine what gets added to the product. We will store the value of the multiplier in an 8-bit register. Select the reg8 component from the list box and place it as shown.
in figure 3. To connect the clock signal to the registers clock input, create a
name_net_s component to the left of the registers clock input. Double click the
net and name it “clock”. The name appears on the right, so press the ‘x’ key
to move it to the left. Now connect the clock net to the registers clock input.
We don’t need a reset signal for this register, so choose the GND component
and connect it. Name the output of the register “multiplier” using the same
technique we used for the clock and reset signals.

The input to this register can come from one of two places. When the
multiplier is first started the input to the register will be some external value
that is to be multiplied. We will simulate this using a constant value. Once the
multiplier is running we will want to continually shift the value in the register
left and then store it for the next step. To implement this, grab a mux2_8
component from the list box and place it as shown in the figure. Connect the
reset signal to its selector and connect its output to the registers input. When
the multiplier is started the reset signal will be asserted for one clock period,
during which the constant value will be loaded. After that reset will be zero
and the shifted value will be loaded at every clock cycle.

Now select a constant8 component from the list box and place it as shown. the
“=0[7:0]” indicates that this is an eight bit bus with value zero. We will
change this value later when we test the multiplier. For now, we need a way to
shift the multiplier value to the left. In order to do this we will make use of a
bus connector. A shift right operation moves all the bits in the value right and
places a zero in the high order bit. Select a bus_combine component from the
list box and place it to the left of the multiplexor. The bus_combine device will
take two separate buses and create a single 8-bit bus on the output. Connect a
GND component to the bus connectors bottom input. This will place a zero in
the low bit of the output. Now connect a named net to the top connector and
name it “multiplier[6:0]”. Note that we have named this with a “[6:0]” instead
of the usual “[7:0]”. This indicates that we only want the low seven bits of the
bus. The bus connector will take these seven bits and combine them with the
ground bit to create a new 8-bit bus. This bus will be a single bit shift left of
the previous multiplier value.

3.3 Creating the Product

The multiplicand is a constant value upon which we perform no operations.
We can represent it using a constant8 component. The product, however, is
accumulated at every step by adding either zero or the multiplicand, depending
on the value of the high order bit of the multiplier. To create the product,
start by making a register. Select the reg8 component and place it as shown in
figure 4. Connect named nets for both the reset and clock, as shown. In this
case we want the product reset to zero when the multiplier starts running (as
opposed to the multiplier register when we didn’t particularly care). Name the
output of this register “product[7:0]” as shown.

The input to the product register is the value from an 8-bit adder. Select
the adder8 component from the list box and place it to the left of the product
register as shown. Connect its output to the input of the register. The adder is going to add two things: the contents of the product register (shifted left one bit) and either a zero or the multiplicand. Let's connect the output of the product register to the A input of the adder. We need to shift the value of the product, much as we did with the multiplier value in the last section. Start by selecting a bus_combine component and placing it as shown. Connect the bottom input of the bus_combine to a GND component. Then connect the top to a named net called “product[6:0]”. The “[6:0]” indicates that we want the low seven bits of the product; the lowest order bit will be replaced by a zero (from the GND component). This effectively causes the value to be shifted left by one bit.

Now we need to connect the other operand for the addition. In this case we are going to have a choice, so we need a multiplexor. Select a mux2_8 component and place it as shown. The selector for the multiplexor is the seventh bit of the multiplier, so connect it to a named net called “multiplier[7]”. When this bit is zero, we want to add a zero to the product, so connect the top input of the multiplexor to a constant8 component with a value of zero. For the time being, do the same with the bottom input of the multiplexor. We will give the input a value when we test the circuit in the next section.
4 Testing the Multiplier

Upon finishing a schematic, it is always necessary to test it and make sure it works as desired. In this case, the testing procedure is relatively straightforward. Add flag components to the various signals as shown in figure 5. Feel free to add more if you like, these are just suggested placements. When we are actually simulating the circuit, these flags will show the current values on those buses. As it currently stands, the inputs to the multiplier are zeros. Fix this by double-clicking on the constant8 components and changing their values to positive numbers. Make sure not to change the constant zero that is the top input to the multiplexor that feeds the adder. You should also double click on the clock generator and set reset to one, period to twenty and cycles to ten (one for the reset and nine for the actual multiplication). The clock generator won’t actually restart, but it will send a stop signal once it has reached ten cycles.

Once you have set everything up, go to the Sim menu and select “verilog netlist” to initialize the simulation. The choose “init probe” from the Sim menu to start the simulation. Pressing the ‘s’ key will execute one step of the simulation. The clock cycle is set to 20 ns and each ‘step’ of the simulation advances time by 10 ns, so you should see the clock shift from one to zero to one as you advance. Once you have advanced by ten cycles you should have the
correct result on the output of the multiplier.