You will implement designs for a 4-bit multiplier. In this lab you will use three design project files from Lab 4; the fulladder, rippleadder8bit, and sevensegment.

**Deliverables:**
0) 4-bit Multiplier
1) 4-bit Multiplier with Hexadecimal Display

**Demonstration Requirement:**
Download and demonstrate the final complete 4-bit multiplier with seven segment decoders on the DE2 board to the instructor.

**Part 0 – A 4-Bit Multiplier**
In this part you will implement a 4-bit multiplier. But for discussion purposes, first consider a 2-bit multiplier.

For a 2-bit multiplier, each input will be a 2-bit value between 00 (decimal 0, Hex 0), and 11 (decimal 3, Hex 3). The biggest output will be $3 \times 3 = 9$ (hex 9) so we need 4-bits for the output.

**Example:** $2 \times 2 = 4$

```
A       10
B       11
00      <- row0 is each bit of A anded with bit B0
01      <- row1 is each bit of A anded with bit B1 and shifted
0100    <- the product
```

A design for a 2-bit multiplier is given in the figure at right. Row0 and row1 need to be added together. If 4-bit adders were used, then we need to pad the inputs to the adders with zeros and shift them as shown. That is, the row0 inputs need 2 zeros on the left to make a total of 4 bits. And the row1 inputs need 1 zero on the right and 1 zero on the left to make a total of 4 bits.
Now consider a 4-bit multiplier. Each input will be a value between 0000 (decimal 0, Hex 0), and 1111 (decimal 15, Hex F). The biggest possible output will be 15 x 15 = 225 (hex E1) so we need 8-bits for the output.

Example: 2 x 2 = 4

\[
\begin{array}{c}
\text{A} \\
0010 \\
\end{array} \quad \begin{array}{c}
\text{B} \\
0010 \\
\end{array} \\
\hline
\begin{array}{c}
0000 \\
\text{row0 is each bit of A anded with bit B0} \\
0010 \\
\text{row1 is each bit of A anded with bit B1, shifted} \\
0000 \\
\text{row2 is each bit of A anded with bit B2, shifted} \\
0000 \\
\text{row3 is each bit of A anded with bit B3, shifted} \\
00000100 \\
\text{the product}
\end{array}
\]

A simplified design for a 4-bit multiplier is given in the figure below. The addition takes 2 stages. In stage 1, row0 and row1 are added, and at the same time, row2 and row1 are added. After that, in stage 2, the 2 sums from the previous stage can be added to get the final result. Note the zero padding and shifting. We can pad zeros by grounding certain inputs of the adder as you will see in the design.

![4 Bit Multiplier Diagram](image)

4 Bit Multiplier

We will need 8 bit adders. Why? The largest values from adding row0 and row1 would be: 15 + 30 = 45 or in binary, 1111 + 11100 = 101101 which is a 6 bit sum. We can use an 8 bit adder and have 2 bits to spare. Same thing for row2 and row3. The largest final answer is 225 (binary 11100001) which we can also handle with an 8 bit adder. We will use the 8-bit ripple adder from Lab 4.

Implement this 4-bit multiplier (Multiplier4bit) design and verify correct operation through simulation.

To start we will implement an "ander" module (suggested project name Andy4) shown in the figure below. Andy4 will be combined with the 8-bit adder design from Lab 4 to create the 4-bit multiplier. Implement Andy4 and simulate its operation.
NOTE: Quartus project name must start with a letter. Andy4 is a valid project name while 4-bitAndy is an invalid name and will result in compiler error.

Andy4 the Ander Module

Below is a decent waveform to test the Andy4 module. Here the inputs and outputs are arranged from msb to lsb (top to bottom) and grouped them. Set a count from 0000 to 1111 on the input group. Set a clock on sel signal. See how the output alternates between zero and the input value, depending on the sel value, exactly as it should.

We will now implement the 4-bit multiplier design (suggested project name Multplier4bit) by combining Andy4 ander module with the 8-bit adder (Rippleadder8bit) from lab 4. Make sure to include Andy4 and BOTH Fulladder and Rippleadder8bit from Lab 4 in your Multplier4bit project library.
See the 4-bit multiplier block diagram design below. If you choose, but not required, you may edit the Rippleadder8bit adder symbol file so that the pins A[7..0] and B[7..0] are separated as shown to allow for easier hook up.

Use the Rippleadder8bit from the lab 4 as the serial adder to add the rows, two rows at a time. The rows are shifted by some bits and padded (connected to ground) with zeros, so make sure to wire up the modules correctly as shown by the GND symbols in the detailed design.
Below is a simulation waveform for a 4 bit multiplier verifying that indeed, $2 \times 2 = 4$. You should check a few other values too.

Group ALL the inputs together and make sure the bits go from MSB to LSB, top to bottom. Set a count on input b to increment every 100ns. Set a count on input a to increment every 1.6us. Set your end time to 100us or less. Set all the radices to unsigned decimal so you can view the decimal results instead of binary.

Note: The university version of Quartus simulator has a limit of some 1000 steps so we set of our end time at 1000 times the smallest count increment of 100ns ($1000 \times 100ns = 100us$) This is sufficient number to verify operation of our design.

![4-Bit Multiplier Waveform (Partial)](image-url)
Part 1 – 4-Bit Multiplier with Seven Segment Decoders

For testing on the DE2 board, it is nice to have the output displayed in hex, so seven segment decoders can be used to do this. Below is a simplified block diagram of the final design. You will need to use a 3rd level of hierarchy. This means, start a new Quartus project and reference the 8-bit multiplier project and your seven seg project, plus all the required subprojects.

You will need to hook up the 8 bit multiplier to 4 instances of the seven segment module. You will have to combine busses and sets of single wires.

Implement this design. (suggested project name “Multiplier8bitdecoded”)
Now demonstrate this part on DE2 hardware using the below pin assignments.

Use the following pins for INPUTS A and B

<table>
<thead>
<tr>
<th>LAB 4 SIGNAL</th>
<th>DE2 Name</th>
<th>DE2 Pin</th>
<th>DE2-115 Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[0]</td>
<td>SW[0]</td>
<td>PIN_N25</td>
<td>PIN_AB28</td>
</tr>
</tbody>
</table>

Here are the pins for the hex displays:

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>DE2 Pin</th>
<th>DE2-115 Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX0[0]</td>
<td>PIN_AF10</td>
<td>PIN_G18</td>
</tr>
<tr>
<td>HEX0[1]</td>
<td>PIN_AB12</td>
<td>PIN_F22</td>
</tr>
<tr>
<td>HEX0[2]</td>
<td>PIN_AC12</td>
<td>PIN_E17</td>
</tr>
<tr>
<td>HEX0[3]</td>
<td>PIN_AD11</td>
<td>PIN_L26</td>
</tr>
<tr>
<td>HEX0[4]</td>
<td>PIN_AE11</td>
<td>PIN_L25</td>
</tr>
<tr>
<td>HEX0[5]</td>
<td>PIN_V14</td>
<td>PIN_J22</td>
</tr>
<tr>
<td>HEX0[6]</td>
<td>PIN_V13</td>
<td>PIN_H22</td>
</tr>
<tr>
<td>HEX1[0]</td>
<td>PIN_V20</td>
<td>PIN_M24</td>
</tr>
<tr>
<td>HEX1[1]</td>
<td>PIN_V21</td>
<td>PIN_Y22</td>
</tr>
<tr>
<td>HEX1[2]</td>
<td>PIN_W21</td>
<td>PIN_W21</td>
</tr>
<tr>
<td>HEX1[3]</td>
<td>PIN_Y22</td>
<td>PIN_W22</td>
</tr>
</tbody>
</table>