Rodrigo Becerril Ferreyra
CECS 361 Section 01
Lab 1
Date: 10 September 2020

# 1 Introduction

The purpose of this lab is to get familiar with the Vivado workflow and to demonstrate knowledge of topics that were covered in class. Specifically, we were instructed to create a four-bit array multiplier in Verilog. An array multiplier is a circuit that takes two inputs—in this case, two four-bit numbers—and multiplies them to get the product. This is achieved by using the standard long multiplication algorithm (details can be found at `https://en.wikipedia.org/wiki/Multiplication_algorithm#Long_multiplication`). In the case of the binary number system, this algorithm can be simplified by the fact that the multiplication table is very simple:

| X | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

Figure 1: Base-2 multiplication table.

Specifically, if the multiplicand (henceforth `B`) is 1, then the multiplier (henceforth `A`) can be copied down as a partial product; if `B` is 0, then zeros can be copied down instead. This fact is exploited in the making of a hardware-based array multiplication circuit.

# 2 Implementation

The implementation of this multiplier circuit relies on two parts: the generation of partial products, and the addition of those partial products. Part 1 can be achieved using multiplexers, and Part 2 can be achieved using half-adders and full-adders.

## 2.1 Partial Products

As previously stated, the partial products can be generated by using a multiplexer. The LSB of `B` is fed into the selection input of the first array of multiplexers. If `B[0]` is 0, then a 0 is placed into the partial product; if `B[0]` is 1, then `A` is placed into the partial product instead. For the next partial product, `B[1]` was fed into the next set of four multiplexers' select input, and so on. This is done simultaneously for all four rows of multiplexers, which takes 16 2-to-1 one-bit multiplexers in total.

## 2.2 Adding the Partial Products

Given only one-bit half-adders and full-adders to work with, adding up all of the partial products is a messy task which involves lots of wires and adders. It is possible to derive the exact circuit which allows the addition of all partial products while taking into account the long multiplication algorithm, which calls for the shifting of the place value of the LSB one to the right for every partial product. However, we were given this circuit so there was no need to.

# 3 Testing and Verification

Once the main module was completed, the time for testing arrived. I created a nested loop that would loop through all 256 possible values for 2 four-bit numbers `A` and `B`. In each iteration, I defined a wire named `expected_product`, which was set to `A*B`. If the product that came from the unit under test did not match this expected product, then an error message would print. However, the simulation ran smoothly and no error was printed. Below is a screenshot of a portion of the timing diagram that was produced; note that the values are represented in unsigned decimal.

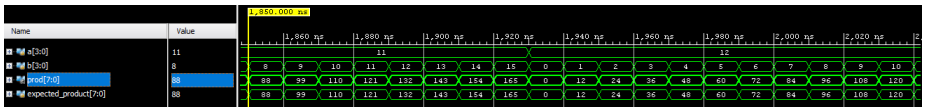


Figure 2: Small snippet of waveform comparing the calculated product `prod` with the expected product.

Here is a small table of several test values. Note that the values are represented in binary.

| Multiplier (`A`) | Multiplicand (`B`) | Product (`prod`) |
|---|---|---|
| 1101 | 1001 | 0111_0101 |
| 0010 | 1111 | 0001_1110 |
| 1101 | 1011 | 1000_1111 |
| 1111 | 0110 | 0101_1010 |
| 0011 | 1010 | 0001_1110 |

Figure 3: Table of values.