Computer Architecture Lab 0

Shane Skikne — Meg McCauley — Ryan Eggert

October 2, 2015

Introduction

In this lab we created a 4 bit Full Adder. This was done by taking the small building blocks of a 1 bit full adder and putting them together to create a 4 bit full adder. All of the code and testing was written in ModelSim. In order to test our code and prove that it worked for all edge cases, we developed a comprehensive set of test benches. After determining that our code was able to fulfill all of these test benches, we turned our code over to hardware. We implement our design in hardware using Vivado, specifically a Zybo FPGA board.

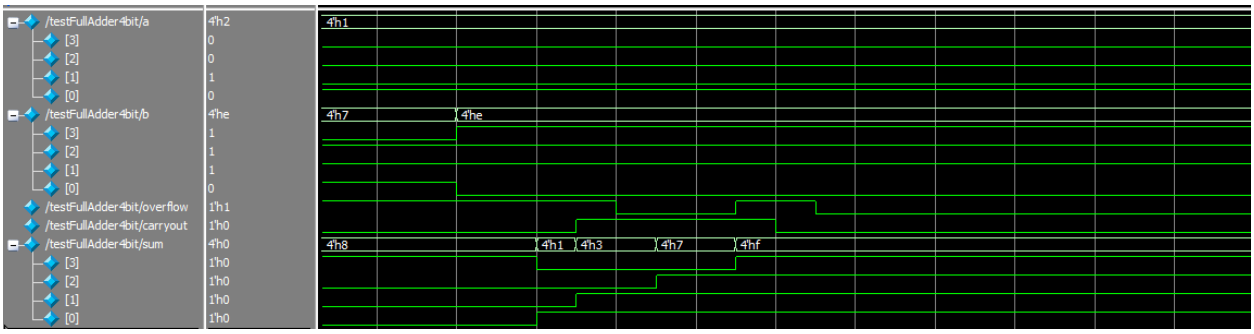

Figure 1: The waveform of our adder when the number we're adding by is changed. Initially, the adder was computing $1_{10}+7_{10}$, but then it switches to computing $1_{10}+-2_{10}$

Results

In Figure1, you can see the waveform from our adder transitioning from computing $1_{10}+7_{10}$ to $1_{10}+-2_{10}$. As you can be seen, there is a lot of variation as the output stabilizes. It easiest to note the propagation in the sum. The input b is changed from 7_{10} to -2_{10} at exactly 6000ns and nothing is affected for 100ns. After that, each bit in the sum starts to switch. Because the largest bit of b is switched, largest bit of the sum switches initially. The rest of the bits switch starting with the lowest bit, with their new value propagating up to the next bit. At the same time, all this switching makes the carryout and overflow flip multiple times before finally stabilizing 450ns after the switch occurred. Practically speaking, this is our adder's worst-case gate delay.

An examination of the schematic of the four-bit adder shows that there should be an extra 50ns of delay, as the four-bit adder carryout signal travels through ten gates (10 time-units, 500ns). However, because the carryin input of the first one-bit adder is hardwired to 0, this 50ns gate delay is not noticeable in execution. This supports our observed 450ns total delay.

Verilog Testbenches

For our testbench, we wanted to test a variety of cases that start out simple and then become involve more complicated binary operations and more complicated two's complement numbers. We start by testing to make sure nothing changes when you add 0. We then moved on to testing with just adding 1_{10}. Two of the more complicated test cases here are summing 1_{10} and -1_{10} to ensure we get 0 with a carryout and adding 1_{10} and 7_{10} to ensure we get an overflow. For the rest of our tests, we had 3 main types: adding two positive numbers to make sure they add correctly or overflow, adding a positive and negative number to make sure they add correctly and don't overflow, and adding two negative numbers to make sure they add correctly or overflow.

Testbench Failures

The single failure we encountered was caused by a mistake in our testbench, not our code. Our design strategy, which we acknowledge won't be optimal for future labs, was to write a first attempt at the Verilog before writing a single test case of 0000 + 0000. We then worked to get our code to actually run, which involved fighting ModelSim, not debugging our code. This allowed us to to be sure that any future issues were from our code, not from our inexperience with ModelSim. Once our code ran, we started writing more test cases to find any errors we had. Luckily, the our first attempt was successful and this strategy. We completely acknowledge that in future labs, having a test bench before code will make debugging and handling edge cases much more quick and simple.

FPGA Tests

The set up on the FPGA involved four buttons, four LEDs, and four switches. The system worked by first using the switches to set the first number to be added. Once this was completed, you would push the first button (far right) to "set" the value for a. This also changed the LEDs to display this number. Then the switches were reconfigured to represent b, the second button was pushed to "set" this value, and the LEDs were again used to represent the number. Pushing the third button changed the display of the LEDs to demonstrate the sum of the two numbers. Pushing the fourth button changed the display of the LEDs to demonstrate both the carry out and the overflow (carry out on the right, overflow on the left).

We selected a subset of our Verilog testbench's test cases to verify on the FPGA. These were selected to check basic operation as well as verifying more advanced overflow and carryout cases. All test cases were successful and we determined that our hardware implementation of the four-bit full adder worked effectively.

As an example, Figure2 on page 3 shows the summation of -1_{10} [1111_2] and -8_{10} [1000_1]. Though $-1+-8 = -9$, we would expect a four-bit adder to overflow and return 0111_2 [7_{10}] with a carryout of 1.

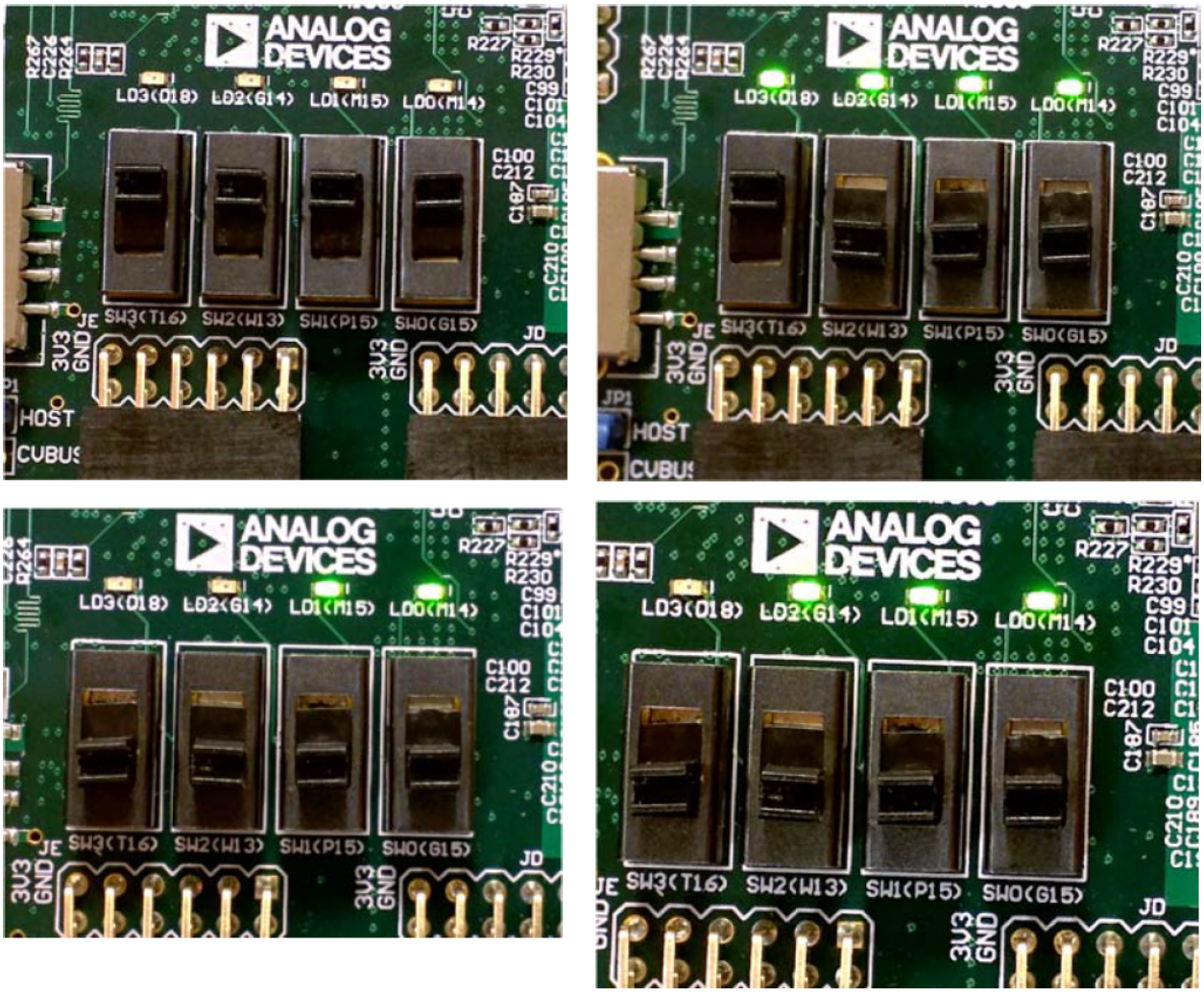

Figure 2: *Clockwise from top left, the adder performs* $1111_2 + 1000_2$. *It returns a sum of* 0111_2 *and shows a carryout of* 1_2 *while also reporting an overflow.*

Resource	Utilization	Available	Utilization (%)
Flip flops (FF)	9	35,200	0.03
Look up tables (LUT)	9	17,600	0.05
Input/Output (I/O)	13	100	13.00
Global clock buffers (BUFG)	1	32	3.12

Table 1: *The expanded table from "Utilization" section of the summary statistics report*

Summary Statistics

Our post-implementation utilization can be seen in Table 1 on page 4. In addition to this segment of results, the full summary statistics report can be seen in Figure 3 on page 4.

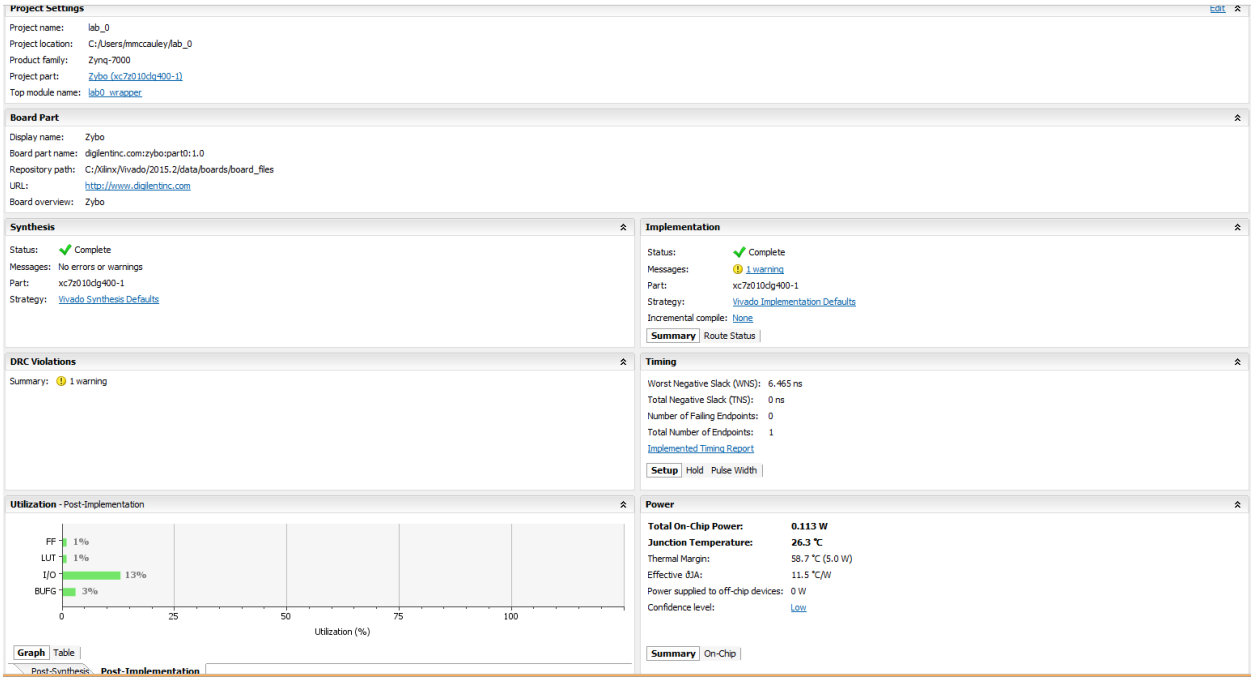

Figure 3: *The summary statistics results from Vivado after uploading to our FPGA board*