Computer Systems, Spring 2020

Week 3 Lab

Digital Circuits

Problems to solve on paper

1. Draw the diagram of a combinational circuit (constructed from logic gates) called and3 that takes three inputs a, b, c and outputs 1 if and only if all inputs are 1. (Hint: use a couple of and2 gates.)

2. Simulate a 4-bit ripple carry adder as it calculates the sum of 6+5, with carry input of 0. To keep this easier, do the simulation showing the inputs and outputs of each fullAdd circuit; don't expand each fullAdd into logic gates. (Hint: Convert the numbers to 4-bit binary words, and use the truth table of the fullAdd circuit to work out the sum and carry bits.)

3. Estimate the gate delay of a 4-bit ripple carry adder. Assume that at time 0 all the inputs to the adder become valid. How many gate delays later will all the outputs be valid?

4. Describe the behaviour of a dff (delay flip flop).

5. Design a 4-bit register circuit. It takes two inputs: load is a bit, and x is a 4-bit word. The register contains a 4-bit state. The output should be the current value of the state. At each clock tick, the register loads x into its state if load=1, but retains the previous state if load=0. (Use four copies of the reg1 circuit as a building block.)

6. Explain why the clock speed in a synchronous circuit depends on the maximum gate delay in the circuit. Discuss what would happen if the clock runs too fast.

7. Recall that a 4-bit word adder takes three inputs: a carry input *cin* and two 4-bit binary words x and y. It produces two outputs: a carry output bit *cout* and a 4-bit sum word s, where $s = x + y + cin$. Design a 4-bit word adder/subtracter circuit called *addsub*. It takes another input *sub*. Its other inputs, and its outputs, are the same as for the adder, except that all the words are two's complement integers. The *addsub* circuit can do either addition or subtraction, and the *sub* input tells it which calculation to do: if *sub*=0 then $s = x + y$, but if *sub*=1 then $s = x - y$. *Hint:* it's a good idea to do this in stages

   (a) You already have an adder

   (b) Design a subtracter; it doesn't take a *sub* input, but always calculates $x - y$. To do this, invert each bit of y (you'll need four inverters) and set the carry input to 1.

   (c) Now introduce the *sub* input.

(d) Set the carry input to *sub* because if you're adding, the carry input should be 0 and if you're subtracting the carry input should be 1. In either case the carry input should be *sub*.

(e) Let's name the second input to the adder z. If $sub = 0$ then $z_i = y_i$, and if $sub = 1$ then $z_i = inv\ y_i$. So $z_i =$ (if sub=0 then y_i else inv y_i). You can do that with a multiplexer.

(f) (Optional.) It's fine to keep the multiplexer, but using either Boolean algebra or truth tables, we can see that $z_i = xor2\ sub\ y_i$, so it's possible just to use one logic gate on each y input.

8. Simulate your *addsub* circuit in order to calculate the following: $2 + 3$, $5 - 3$, $3 - 5$, $4 + (-2)$, $4 - 2$, $-3 - 4$.

Problems to try on the computer

1. Load the 1-bit register circuit: launch LogicSim, click File: Load, then select reusables and then reg1.

   (a) Click Start Simulation. Notice that initially the flip flop contains 0 and both of the inputs are 0. Notice that some of the wires are grey; this means they are not yet valid.

   (b) Let's start by loading a 1 into the register. To do this, we need to set the inputs to the circuit to express a command that says *"at the next clock tick load a 1 into the register"*. To form this command, set the load input to 1 ("load the data input into the register") and set the data input to 1 ("this is the value to load") (Be sure you know which input is the load and which is the data input!)

   (c) A very important concept to understand about synchronous circuits is that the wires carry values which may represent a command, but the command does not actually get executed *until the next clock tick*.

   (d) Click the Gate Delay button; this will cause the wires at the current path depth to become valid (blue means 0, red means 1). Notice that you need to click Gate Delay several times before the output of the Multiplexer becomes valid. Why? Because the multiplexor is actually a circuit with a path depth of 3: it takes three gate delays before its output becomes valid.

   (e) Click Gate Delay repeatedly until every wire is valid. The LogicSim app shows the Path Depth of the circuit (for this circuit it's 3). That means that on each clock cycle, you'll need to click Gate Delay that many times.

   (f) Now click Clock Tick and you'll see the state of the flip flop change from 0 to 1.

   (g) Tip: in a larger circuit, the path depth may be fairly large and it gets tiresome to keep clicking Gate Delay over and over again. If you want to get the "big picture" and just see the changes to the flop flops without having to keep clicking Gate Delay, you can change the simulation mode. Click Options: Do path depth gate delays per

clock tick. Now the app will automatically do enough Gate Delay operations and you won't have to click that button again. But if you want to study how validity propagates through the circuit, turn off that option.

(h) Now simulate the circuit for several clock cycles with suitable input. Use the same input data used in the lecture. The point of this exercise is simply to understand how the register works.

2. Load the add4 circuit (it's also in Reusables). This is a 4-bit ripple carry adder that uses a full adder for each bit position.

   (a) This introduces a few new features of LogicSim: you can combine four wires into a *bus* which shows as a single thick cross-hatched line. The input to a bus can be set using a hexadecimal input box, and you can display the output as a hex digit.

   (b) There are also new boxes called splitter and join. *These are not components, they are just conversions between two notations.* The splitter takes a bus and produces the four individual signals, and join does the opposite. Using splitters, joins, and buses can simplify the diagram for a large circuit. But it's important to understand that these notations don't simplify the circuit, they only simplify the diagram used to describe the circuit.

   (c) Test the ripple carry adder. Try adding 2+3 and see if you get 5. Also, try some additions where the result is too large to fit in 4 bits, for example 6 + 12. In this case, you'll see that the carry output from the leftmost position is 1.

   (d) Run these simulations in the default mode, so you can see how validity ripples across the adder from right to left. That's why it's called a ripple carry adder!

3. Implement your add/subtract circuit (described above in the paper exercises) and test it with the simulator.