Computer Systems, Spring 2019

Week 3 Lab

Digital Circuits

Solutions

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.)
   Solution. output = and2 a (and2 b c)

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.)

   Solution. Convert the inputs to binary: x = 6 = 0110 and y = 5 = 0101. Each full adder receives a bit from x and a bit from y, and a carry input from the full adder to the right (the rightmost full adder receives a carry input which is specified to be 0). Thus the full adders have the following inputs and outputs (the full adders are identified by the column value: 8, 4, 2, 1. The carry output is 0, and the sum is 1011 which is the binary representation of 11 (base 10); thus the circuit has calculated $5 + 6 = 11$.

column value	8	4	2	1
carry input	1	0	0	0
x	0	1	1	0
y	0	1	0	1
sum	1	0	1	1
carry output	0	1	0	0

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?

   Solution. A ripple carry adder consists of a linear connection of full adders. Suppose the gate delay through a logic gate is d; then the gate delay through a full adder is $3 \times d$. The carry input to each full adder comes from the carry input produced by the full adder to the right, so the delay for an n-bit ripple carry adder is $3 \times n \times d$; for a 4-bit adder it's $12 \times d$. Thus the time required by the adder is proportional to the word size: if you double the word size, the adder takes twice as long.

   The important point is that the clock speed must be set slow enough to enable the signals to become valid through the critical path, which is the sequence of logic gates with the *longest* gate delay in the circuit. Usually the critical path contains an adder, which is why the speed of the adder is so important. To put this in perspective, consider that an adder in a modern processor will have a delay of several dozens of gate delays, and the speed of the entire circuit (containing around 50 million components) is determined by these several dozen logic gates. Every other component in the system is irrelevant to the speed. That's one reason that small scale optimisations (Karnaugh maps, etc.) are often useless. There are advanced techniques that enable us to design adders that are much faster than the ripple carry adder, although they are a lot more complicated and harder to understand.

(Optional and advanced point; you don't need to be this precise.) If the full adder circuit is built from two half adders and a logic gate, then the least significant one will take three gate delays and the ones to the left will take only two additional gate delays from the time their carry input becomes valid, resulting in a total delay of $3 + 2 + 2 + 2 = 9$ gate delays.

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

   Solution. A dff has an input x and an output y. It contains an internal state of one bit; the state is a bit that it "remembers".

   - During a clock cycle the dff ignores the value on its input signal x, and the output signal y gives the current value of the internal state.
   - At a clock tick, the dff discards the old value of its state, and puts the value on the input x into the state. This value will now be output on y for the subsequent clock cycle.

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.)

   Solution Suppose we name the bits of the x input x_0, x_1, x_2, x_3. Use four copies of the reg1 circuit; each gets the same load input signal and each gets a corresponding bit of x.

```
reg4 load [x0,x1,x2,x3] =
  [reg1 load x0,
   reg1 load x1,
   reg1 load x2,
   reg1 load x3]
```

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.

   Solution. At a clock tick, all of the flip flops receive a new value (it could be the same as its old value, or it could be different). Also, we assume that

all inputs to the entire circuit can change only at a clock tick. Therefore, after a tick all the circuit inputs and all the flip flop outputs are stable for the entire subsequent clock cycle. These signals may go into a chain of logic gates, and the clock cycle must be long enough for all the signals to settle down to a stable final value. Then, after a little further time has elapsed, there will be a tick and all the inputs to all the flip flops will be stable and correct.
If the clock runs too fast, then some of the signals may not have settled down to a stable value when a tick occurs, and incorrect data may be loaded into flip flops. This is a fatal error, and there is no possible way to recover from it.

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.

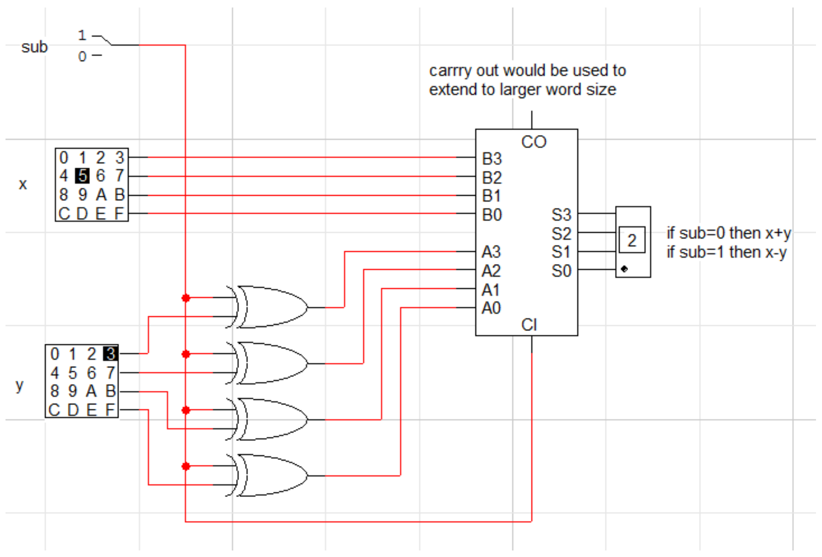
   (c) Now introduce the *sub* input.

   (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.

   Solution. Note that $x - y = x + (-y)$, and to negate y we must invert its bits and add 1. Use a 4-bit ripple carry adder. Its *cin* input is *sub*; i.e. it's 0 if we're doing addition, and 1 if we're doing subtraction. The first word input to the ripple carry adder is x. The second word input is (if *sub*=0 then y else invert y). That second word can be calculated by defining the ith bit position as $(mux1\ sub\ y_i\ (inv\ y_i)$. If you like, this can be simplified to $xor2\ sub\ y_i$.



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

   Solution. For each problem we write out the bit values of x and y (the original numbers being added or subtracted); z (the second input to the adder, which is either y or y with the bits inverted); and *cin*. (Note: we will ignore carry output and overflow; those can be handled too but to save time we aren't going into those details in this course. However, it's worth pointing out that the carry output is *not* the same as overflow. These problems have numbers that are small enough to avoid overflow.)

   - $2+3$
     $x = 0010, y = 0011, z = 0011, cin = 0, result = 00101 = 5$
   - $5-3$
     $x = 0101, y = 0011, z = 1100, cin = 1, result = 0010 = 2$
   - $3-5$
     $x = 0011, y = 0101, z = 1010, cin = 1, result = 1110 = -2$
   - $4+(-2)$
     $x = 0100, y = 1110, z = 1110, cin = 0, result = 0010 = 2$
   - $4-2$
     $x = 0100, y = 0010, z = 1101, cin = 1, result = 0010 = 2$. Notice how this differs from $4+(-2)$.
   - $-3-4$
     $x = 1101, y = 0100, z = 1011, cin = 1, result = 1001 = -7$

Problems to try on the computer

1. reg1 — you have to do this for yourself; looking at a model solution on paper is no substitute for seeing a circuit like this as it changes its state

through a sequence of clock cycles. Make a simulation table, as shown in the lecture, and fill it out by simulating the circuit.

2. add4 — remember that the input words have 16 values: $0, 1, \ldots 15$. Be sure to try several additions, including some that overflow. The largest possible addition is (in decimal) $15 + 15 + 1 = 31$. Since the carry output has a value of 16, you should see a carry output of 1 and a sum of F.