CS99S
Laboratory 3 Preparation

October 8, 2001

Objectives:

1. Understand the principles of combinational logic
2. Understand binary representation of integer numbers
3. Understand binary adders
4. Build a half-adder, a full-adder, and a multi-bit adder

Binary Numbers and Computer Arithmetic

While we are used to representing numbers in decimal notation, numbers within a computer are represented in *binary*, that is as strings containing just the digits 0 and 1. We often refer to these ***bi**nary digi**t**s* as *bits*. As in decimal, the least significant (rightmost) digit is weighted 1; unlike decimal, however, each digit to the left of the rightmost digit has a weight that is twice the weight of the digit immediately to its right. Stated mathematically, the binary number $b_n b_{n-1} \ldots b_1 b_0$ has a value given by $\sum_{i=0}^{n} b_i 2^i$. That is, digit b_i has *weight* 2^i. For example, the number 28_{10} (this means 28 in base 10) is 11100_2 with 1s in the 4s, 8s, and 16s place (since 4+8+16 = 28).

Prep question 1: Convert 19_{10} to binary and convert 10101_2 to decimal.

We add binary numbers just like we add decimal numbers, one digit (bit) at a time carrying value from one bit position to the next when needed. For example, consider adding 1100_2 to 1110_2 (12_{10} to 14_{10}). The step-by-step process is shown below. We start by adding the least significant digit. 0+0 give a result of 0 with a carry (shown in italics of 0) of 0. We then add 0 (the carry) + 0 + 1 to give a result of 1 in the 2s position with a carry of 0 – into the 4s position. In the 4s position we have 0+1+1 giving a sum of 0 with a carry of 1 into the 8s position. In the 8s position we have 1+1+1 giving a sum of 1 with a carry of 1 into the 16s position. The net result is 11010_2 (26_{10}).

```
   0     00    100   1100   1100
1100   1100   1100   1100   1100
1110   1110   1110   1110   1110
----   ----   ----   ----   ----
   0     10    010   1010  11010
```

Note that at each position (column) we sum either 2 bits (in the least significant position) or 3 bits (in all other positions) of the same weight to generate a result between 0 and 3 in binary. The least significant bit of this two-bit result is the contribution to our current bit while the most significant bit is the carry.

We call each position above a "bit slice". Because the operation at each bit slice is identical, we can use the same hardware for each bit slice. A 32 bit adder, for instance, would use 32 bit slices with the same hardware for each slice.

Prep question 2: Add 11011_2 to 10010_2. Express the result in both binary and decimal.

We can write the truth tables for summing 2 bits or 3 bits as shown below. The logic function that sums 2 bits is known as a *half adder*, and the logic function that sums 3 bits is known as a *full adder*. For each row of both of these truth tables, the two bits out are a binary count of the number of 1s on the input side of that row. "a" and "b" are the data inputs (a+b), "s" is sum, "ci" is carry in, "c" or "co" is carry out.

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

a	b	ci	co	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Both the half and full adder are circuits that convert between two representations of the same number. These circuits input a *unary* representation of the number (in which all bits are weighted 1) and output a *binary* representation of the number (in which the s bit is weighted 1 and the c bit is weighted 2).

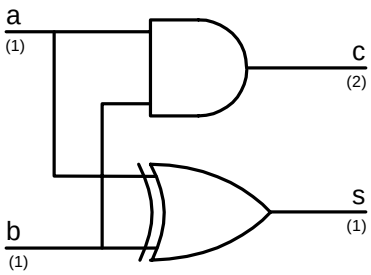
The Adder as Combinational Logic

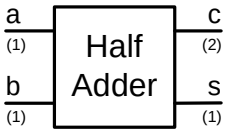
Using the tools we developed in Lab 2, we can convert these truth tables to logic equations and schematics. Recall that to convert a truth table into a logic equation, first write logic expressions for each row in the truth table which have outputs of 1 by ANDing the inputs together. Then OR together each of these expressions to create the final logic equation. (See "s" below for an example.). For the half adder we have

$s = (\sim a \wedge b) \vee (a \wedge \sim b) = a \oplus b$

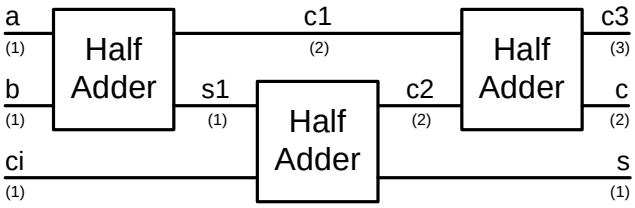
$c = a \wedge b$

which we can draw in schematic form as shown below. The gate-level schematic on the left is represented by the box on the right. The small numbers in parentheses represent the relative weight of the inputs.

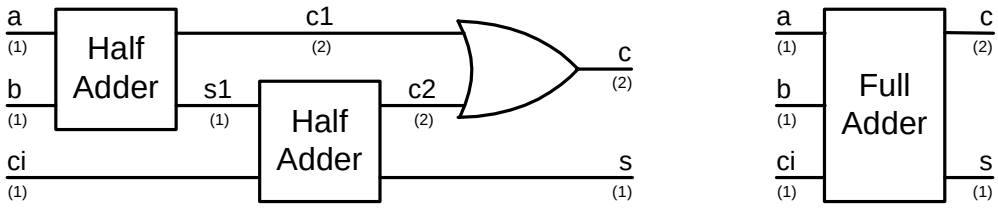




We could do the same thing for the full adder – write down the logic equation and convert it to a schematic. However, it is more interesting to use our half adder to implement a full adder. Using half-adders to sum up signals with identical weight until we have only one of each we can draw the following circuit:



In this circuit the first half adder sums two of the input bits (both weight 1) to generate s1 (weight 1) and c1 (weight 2). The second half adder sums the remaining two weight 1 signals, s1 and ci. This generates a single weight 1 signal, s, and a second weight 2 signal, c2. The second half adder sums the two weight 2 signals c1 and c2 to generate a single weight 2 signal, c and a weight 3 signal, c3. Of course c3 will always be zero since the result is in the range of 0-3 and since c1 and c2 are never high at the same time, we can replace the final half adder with an OR gate giving the following circuit for a full adder which we will represent with the box on the right:



Prep question 3: Draw a gate level schematic for the full adder shown above complete with pin assignments using 74AC86 (4 2-input XOR gates) and 74AC00 (4 2-input NAND gates) chips. (See the schematics linked from the class web page.) Hint: generate the complement[1] of c1 and c2 rather than the true value of these signals so you can use NANDs for both the carry gates in the half adders and for the final OR. You will wire the full adder you build in the lab from this schematic.

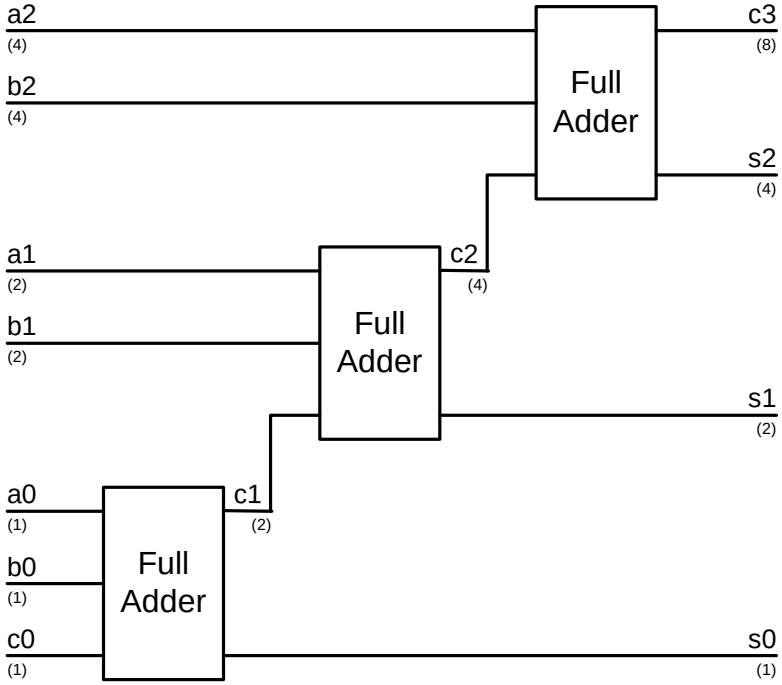
[1] The complement of a signal is its opposite signal; i.e. the complement of a 0 is 1, the complement of a 1 is a 0.

Prep question 4: (optional challenge question) What is the smallest number of transistors you can use to realize a full adder?

Multi-Bit Adders

Once we have a one-bit full adder, we can build multi-bit adders by chaining the carry out of one full adder into the carry in of the full adder for the next bit. Below we show this for a three-bit adder. The circuit below adds the three-bit binary number a2a1a0 to the three bit binary number b2b1b0 and to the one-bit carry in c0 yielding a four-bit result c3s2s1s0. We denote a bit of signal x with weight 2^i as xi – hence the bit of a with weight 4 is a2.

Just as in performing addition by hand, this circuit adds the carry from one column (or bit position) into the next column. Note that the weights of the inputs to a particular full adder are always the same. While in theory we could get by with a half-adder in the least significant (1s) position, in practice we almost always use a full adder in this position so we can perform higher precision arithmetic by saving the carry out from one addition and adding it into a later addition.



The Lab

In the lab you will do the following.

1. Wire up your circuit from prep question 3. First wire up one half adder and verify that it works using your switches and LEDs for input and output. Then wire up the other half adder and verify that the whole full adder works.
2. Once everyone has completed step 1 we will wire all of the lab kits together to make a multi-bit adder. To combine two lab kits we need to first connect a ground wire between them. Then we tie the carry out of one kit to the carry in of the next kit. Once we have constructed a multi-bit adder in this fashion we will have it work a number of addition problems to verify its operation.