

CS 135 Fall 2010 Opional Homework: Due December 10th, 11:59pm

We have discussed performance of programs, and defined it as the execution time of the program. In this assignment, we look at the hardware performance of arithmetic units. Specifically, we explore the efficiency of the full adder circuit.

How can we model the efficiency of a hardware circuit built using basic combinational gates? Specifically, how do you model time and space (the space translates to area needed on the chip to implement the circuit). For “time complexity” we need to model the number of gates in the longest path through the circuit since that plays a role in the time needed to switch all the transistors in that path. Typically, the number of inputs to a gate are ignored – *i.e.*, a two input gate or four input gate are considered equally fast and assume circuits can be built using any k-input gate.

Ques. 1: Recall the circuit for addition of two n-bit 2’s complement numbers – refer to Chapter 3.3.3 of the textbook (on Full Adder), and specifically to Figure 3.16 and Figure 3.15 that describe the circuit for adding two 4-bit binary numbers. The inputs are two n-bit binary numbers $A = a_{n-1} \dots a_1, a_0$ and $B = b_{n-1}, \dots, b_1, b_0$. The output is the sum of the two numbers $S = s_{n-1}, \dots, s_1, s_0$. This addition “algorithm” is known as a *ripple carry adder* since the carry-in for bit $i + 1$ is the carry-out for bit i and requires that carry at bit i be generated before we can add the bits in position $i + 1$. How “efficient” is this algorithm in terms of adding two n-bit numbers.

Ques.2: The goal now is to design a more time efficient circuit for addition that can perform the addition of two n-bit numbers faster than the ripple carry adder.

Consider the boolean function for the sum s_i and carry-out c_{i+1} at bit position i (note that carry-out at bit i is the carry-in c_{i+1} at bit position $i + 1$). By examining the truth table for addition, shown in Figure 3.14 in textbook, we can derive the boolean expressions for Sum s_i and Carry out c_{i+1} at position i in terms of the inputs a_i, b_i, c_i (observe that for $i = 0$, $c_0 = 0$) as:

$$s_i = \bar{a}_i \bar{b}_i c_i + \bar{a}_i b_i \bar{c}_i + a_i \bar{b}_i \bar{c}_i + a_i b_i c_i$$

$$c_{i+1} = b_i c_i + a_i c_i + a_i b_i$$

Using the boolean algebra laws, we can factor the second equation and get:

$$c_{i+1} = a_i b_i + (a_i + b_i) c_i$$

Now let $G_i = a_i b_i$ and let $P_i = a_i + b_i$. Substituting these into the equation for c_{i+1} we get:

$$c_{i+1} = G_i + P_i c_i$$

The G_i and P_i terms are referred to as the **generate** and **propagate** functions, respectively, for the effect they have on the carry. When $G_i = 1$, a carry is generated at stage i . When $P_i = 1$, then a carry is propagated through stage i if either a_i or b_i is a 1.

Based on the above discussions, can you prove the lemma below?

- **Lemma 1:** The G_i and P_i terms can be created in one level of logic since they **only** depend on an AND or an OR of the input variables.

Now consider how the carries, c_i , can be computed for each stage i .

The carry c_1 for stage 0 is

$$c_1 = G_0 + P_0 c_0$$

and since $c_0 = 0$ (for addition), we can rewrite this as $c_1 = G_0$.

Since $c_1 = G_0$, the carry c_2 for stage 1 can be written as

$$c_2 = G_1 + P_1c_1 = G_1 + P_1G_0$$

Since $c_2 = (G_1 + P_1G_0)$, the carry c_3 for stage 2 can be written as

$$c_3 = G_2 + P_2c_2 = G_2 + P_2G_1 + P_2P_1G_0$$

Continuing one more time for stage 3 in a 4-bit adder, the carry c_4 from stage 3 can be written as:

$$c_4 = G_3 + P_3c_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$

Based on the discussions above (and looking at the functions for the carry-in or carry-out at each stage), can you prove the lemma 2 below ?

- **Lemma 2:** Each carry bit, for any bit position i , can be defined entirely in terms of the input values a_i, b_i (and do not require other carry bit values).

Now recall the boolean function for the sum s_i at stage i :

$$s_i = \bar{a}_i\bar{b}_i c_i + \bar{a}_i b_i \bar{c}_i + a_i \bar{b}_i \bar{c}_i + a_i b_i c_i$$

For example, the function for s_2 is:

$$s_2 = \bar{a}_2\bar{b}_2 c_2 + \bar{a}_2 b_2 \bar{c}_2 + a_2 \bar{b}_2 \bar{c}_2 + a_2 b_2 c_2$$

For example, the function for s_3 is:

$$s_3 = \bar{a}_3\bar{b}_3 c_3 + \bar{a}_3 b_3 \bar{c}_3 + a_3 \bar{b}_3 \bar{c}_3 + a_3 b_3 c_3$$

Now revisit the full adder circuit from Figure 3.16 to add two 4-bit numbers. Given the above properties, captured in Lemmas 1 and 2, for generating the carry bits for each stage using the generate and propagate signals, can you design a more efficient (in terms of time) circuit for a 4-bit full adder ? Typically, such an adder is called a *carry lookahead adder*. In terms of adding two n -bit binary numbers, how much more efficient is the carry lookahead addition “algorithm” compared to the ripple-carry addition “algorithm”. Explain.