# CS M151B Homework 3

Charles Zhang

January 20, 2022

# Problem 3.14

**Calculate the time necessary to perform a multiply using the approach given in `Figures 3.3` and `3.5` if an integer is 8 bits wide and each step of the operation takes 4 time units. Assume that in step `1a` an addition is always performed—either the multiplicand will be added, or a zero will be. Also assume that the registers have already been initialized (you are just counting how long it takes to do the multiplication loop itself). If this is being done in hardware, the shifts of the multiplicand and multiplier can be done simultaneously. If this is being done in software, they will have to be done one after the other. Solve for each case.**

Given that each integer is 8 bits wide, we know that this multiplication loop will have 8 iterations. In addition, knowing that step `1a` will always be performed tells us that there are 3 distinct steps in the loop: adding the multiplicand to the product, shifting the multiplicand left, and shifting the multiplier right. It is given that each of these steps takes 4 time units to execute.

In the hardware implementation, these last 2 steps can be done in parallel, leaving us with 2 steps per iteration. This results in:

$$8 \text{ iterations} \times 2 \text{ steps/iteration} \times 4 \text{ time units/step} = \boxed{64 \text{ time units}}$$

In the software implementation, these last 2 steps must be done sequentially, leaving us with 3 steps per iteration. This results in:

$$8 \text{ iterations} \times 3 \text{ steps/iteration} \times 4 \text{ time units/step} = \boxed{96 \text{ time units}}$$

# Problem 3.15

**Calculate the time necessary to perform a multiply using the approach described in the text (31 adders stacked vertically) if an integer is 8 bits wide and an adder takes 4 time units.**

A multiplication implementation using 31 stacked adders allows each step of the multiplication loop to be entirely dependent on the result of the previous adder. As a result, all of the time necessary for this computation is simply:

$$8 \text{ adders} \times 4 \text{ time units/adder} = \boxed{32 \text{ time units}}$$

# Problem 3.16

**Calculate the time necessary to perform a multiply using the approach given in `Figure 3.7` if an integer is 8 bits wide and an adder takes 4 time units.**

This parallel structure for the multiplier takes $\log_2(N)$ add times for an $N$-bit value. Therefore, for an 8-bit value, using adders that take 4 time units, we have:

$$\log_2(8) \text{ add times} \times 4 \text{ time units/add time} = \boxed{12 \text{ time units}}$$

# Problem B.28

**Now calculate the relative performance of adders. Assume that hardware corresponding to any equation containing only OR or AND terms, such as the equations for p$i$ and g$i$ on page B-40, takes one time unit $T$. Equations that consist of the OR of several AND terms, such as the equations for $c1$, $c2$, $c3$, and $c4$ on page B-40, would thus take two time units, $2T$. The reason is it would take $T$ to produce the AND terms and then an additional $T$ to produce the result of the OR. Calculate the numbers and performance ratio for 4-bit adders for both ripple carry and carry lookahead. If the terms in equations are further defined by other equations, then add the appropriate delays for those intermediate equations, and continue recursively until the actual input bits of the adder are used in an equation. Include a drawing of each adder labeled with the calculated delays and the path of the worst-case delay highlighted.**

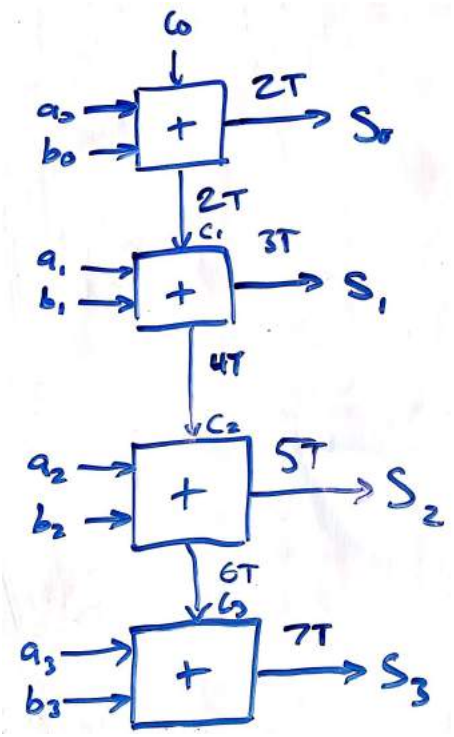
In ripple carry, we know that each result is determined by $(a \oplus b) \oplus C_{\text{in}}$. Similarly, each carry out is determined by $(a \cdot b) + (a \cdot C_{\text{in}}) + (b \cdot C_{\text{in}})$. Since terms involving the OR of several AND terms take $2T$ to compute, we know that $C_{i+1}$ takes $2T$ longer than $C_i$. In addition, we assume $C_0$ is known at $T = 0$. Therefore:

$$C_3 = C_2 + 2T = (C_1 + 2T) + 2T = ((C_0 + 2T) + 2T) + 2T = 6T$$

Finally, we know that result $S_i$ can be determined $T$ time after knowing carry in $C_i$, therefore:

$$S_3 = C_3 + T = 6T + T = \boxed{7T}$$

All intermediate delays can be calculated similarly and are shown as follows:



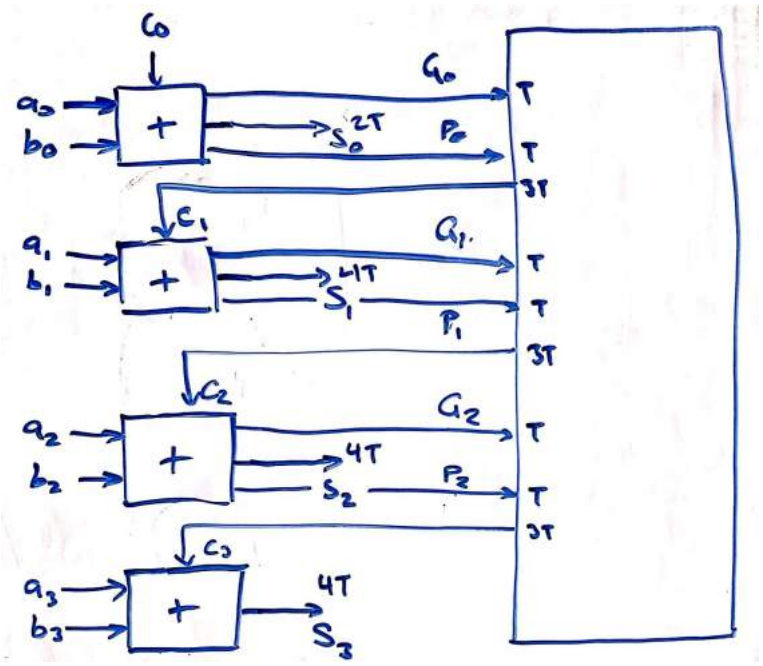
In the carry lookahead implementation, we use two new parameters, $G$ and $P$. Since $G = a \cdot b$ and $P = a \oplus b$, we know that both operations can be completed in $T$. Based on the definitions of $G$ and $P$, we know the following:

$$C_{i+1} = G_i + (C_i \cdot P_i)$$

Recursively applying this equation, we can see that each carry in can be calculated in $2T$ after $G$ and $P$ are available. This means that each carry in is available at $3T$. From there, we can use the same sum logic ($S_i = (a \oplus b) \oplus C_i$) to see that:

$$S_i = C_i + T = 3T + T = \boxed{4T}$$

All intermediate delays can be calculated similarly and are shown as follows:




From these delays, we can see that the performance ratio of the carry lookahead adder to the ripple carry adder is:

$$\frac{7T}{4T} = \boxed{1.75}$$

# Problem B.29

**This exercise is similar to `Exercise B.28`, but this time calculate the relative speeds of a 16-bit adder using ripple carry only, ripple carry of 4-bit groups that use carry lookahead, and the carry-lookahead scheme on page `B-39`.**

In the previous problem, we saw that $C_{i+1}$ takes $2T$ to generate after $C_i$ is made available to us. In addition, we saw that $S_i$ takes $T$ to generate after $C_i$ is made available. Therefore, we can see that the speed of a 16-bit adder using ripple carry only can be calculated as follows:

$$C_{15} = 30T$$

$$S_{15} = C_{15} + T = \boxed{31T}$$

Again, in the previous problem, we saw that a 4-bit adder using carry lookahead generated all carry outs in $3T$ and all result bits in $4T$ after $C_0$ is given. Using the carry in labels $C_\alpha$ for the first group, $C_\beta$ for the second group, $C_\gamma$ for the third group, and $C_\delta$ for the fourth group, we can say the following:

$$C_\delta = C_\gamma + 3T = (C_\beta + 3T) + 3T = ((C_\alpha + 3T) + 3T) + 3T = 9T$$

Now, we can us the fact that results are generated $4T$ after the carry in to say:

$$S_\delta = C_\delta + 4T = \boxed{13T}$$

Finally, assuming the scheme refers to a pure carry lookahead implementation, we noted above that carry lookahead adders have all results $4T$ after the carry in is provided (under the assumption that all sum of products expressions are evaluated in $2T$, regardless of fan-in). Therefore, we can say that this implementation would have a relative speed of:

$$\boxed{4T}$$

# Problem B.31

**Instead of thinking of an adder as a device that adds two numbers and then links the carries together, we can think of the adder as a hardware device that can add three inputs together** $(ai, bi, ci)$ **and produce two outputs** $(s, ci+1)$**. When adding two numbers together, there is little we can do with this observation. When we are adding more than two operands, it is possible to reduce the cost of the carry. The idea is to form two independent sums, called** $S'$ **(sum bits) and** $C'$ **(carry bits). At the end of the process, we need to add** $C'$ **and** $S'$ **together using a normal adder. This technique of delaying carry propagation until the end of a sum of numbers is called carry save addition. The block drawing on the lower right of `Figure B.14.1` shows the organization, with two levels of carry save adders connected by a single normal adder. Calculate the delays to add four 16-bit numbers using full carry-lookahead adders versus carry save with a carry-lookahead adder forming the final sum. (The time unit** $T$ **in `Exercise B.28` is the same.)**

Noting the information from the previous 2 problems, we know a full carry lookahead implementation under these assumptions allows us to compute the sum of 2 $N$-bit values in $4T$. Since the addition of 4 values requires 3 summations, we can say that:

$$S = 3 \times 4T = \boxed{12T}$$

Looking at the diagram given by the problem, we can see that the first layer of carry save adders receives all inputs at $T = 0$. This allows them to produce all outputs in $2T$, as calculating sums and carry outs in a full adder takes $2T$ due to the product of sums form. These can then be routed to the second layer of carry save adders, which in turn, produce their outputs at $4T$. These outputs would then be input to the carry lookahead adder, which we know produces a sum in $4T$. Therefore, the relative time is:

$$S = 4T + 4T = \boxed{8T}$$