Problem 1. Adder Latency

The basic building block used in adder and multiplier circuits is the full adder, a 3-input, 2-output combinational logic circuit we’ve seen before:

\[ C_{OUT} = A \cdot B + C_{IN}(A \oplus B) \]
\[ S = A \oplus B \oplus C_{IN} \]

The simplest adder architecture is the ripple-carry adder shown in the following diagram. This adder gets its name from the observation that the carry signal ripples from one full adder to the next.

In the questions below, assume that (1) the \( t_{PD} \) for each gate (INVERTER, AND, OR, XOR) is 1 time unit, (2) the XOR gate has only two inputs, and (3) the other multi-input gates (AND, OR) can have at most four inputs.

(A) It's clear from the schematic for the ripple-carry adder that the longest path involves the propagation of the carries from the least-significant bit (bit 0) to the most-significant bit (bit \( N-1 \)). Compute \( t_{PD} \) for a 20-bit ripple-carry adder and identify the path through the schematic which has this worst-case delay – this path is called the critical path. What values for the inputs would exercise the critical path, i.e., what value for \( A_0, \ldots, A_{19}, B_0, \ldots, B_{19} \) and \( C_{IN} \) would result in an actual propagation delay that comes closest to \( t_{PD} \)?
Since the $C_{OUT}$ signals of each full adder are in the critical path, let’s see if we can’t improve the
adder’s performance by generating the $C_{OUT}$ signals more quickly. First let’s rewrite the equation
for $C_{OUT}$:

$$C_{OUT} = A B + (A \oplus B) C_{IN} = G + P C_{IN}$$

where $G = A B$ is true if a carry is generated by the full adder and $P = A \oplus B$ is true if the carry
is propagated by the full adder from $C_{IN}$ to $C_{OUT}$. Note that $P$ and $G$ depend only on $A$ and $B$ and
not on $C_{IN}$. We can generalize the notion of $P$ and $G$ to blocks of several bits. For example,
consider a two-bit adder:

Define the block generate signal $G_{0,1}$ as $G_{0,1} = G_1 + G_0 P_1$, i.e., the 2-bit block will generate a
carry if a carry is generated in bit 1 ($G_1$) or if a carry is generated in bit 0 and propagated by bit 1
($G_0 P_1$). Similarly we can define the block propagate signal $P_{0,1}$ as $P_{0,1} = P_0 P_1$, i.e., $C_{IN}$ will be
propagated to $C_{OUT}$ only if both bits are propagating their carry-ins.

(B) Draw a logic diagram using gates for the block generate signal $G_{0,3}$ and block propagate
signal $P_{0,3}$ for a 4-bit adder in terms of the generate signals ($G_0, G_1, G_2, G_3$) and propagate
signals ($P_0, P_1, P_2, P_3$) for each bit.

(C) Using the block generate and block propagate signals, the carry-out of the 4-bit adder is
just $C_{OUT} = C_3 = G_{0,3} + P_{0,3} C_{IN}$. Recalling that gates with up to four inputs have a $t_{PD}$ of 1
time unit, how long will it take to compute $C_{OUT}$ from $A_0, ..., A_3, B_0, ..., B_3$ and $C_{IN}$ if we
use $G_{0,3}$ and $P_{0,3}$? Compare this to the time would take to compute $C_{OUT}$ in a 4-bit ripple
carry adder.

(D) Now suppose we built our 20-bit adder using five of the improved 4-bit blocks from part
(C). How long will it take compute $C_{OUT}$ from $A_0, ..., A_{19}, B_0, ..., B_{19}$ and $C_{IN}$? Compare
this to the time it would take to compute $C_{OUT}$ in a 20-bit ripple carry adder. Hint: the
answer isn’t just five times the delay of a 4-bit block!

(E) Adders built using the architecture suggested in part (D) are called carry-skip adders.
Briefly explain where the name comes from. Hint: look at your timing analysis from part
(D) and think how the carry would propagate from bit 0 to bit 19.

(F) [Hard] There’s no reason that the blocks in a carry-skip adder all have to process the same
number of bits. Use this idea to suggest a slightly different carry-skip adder that has
improves on the latency of $C_{OUT}$ over that of the adder in part (D).
Problem 2. “Go Forth and Multiply”

a) Design logic to perform the multiplication of two, 2-bit unsigned integers producing a 4-bit result (Hint: write a truth table for each output bit). Draw a gate-level-circuit diagram.

b) Determine the function of the following circuit by writing out and examining its truth table.

```
A1
B1
A2
B2
```

Circuit X

```
P1
P2
P4
P8
```

c) What aspects of minimization are not considered in a sum-of-products realization?

d) Assume that the 2-bit multiplier that you designed in part a) is represented as the following function block:

```
A1 A0 B1 B0
```

Mult2

```
P0 P1 P2 P3
```

Use this function block and single-bit full-adders to build a 4-bit multiplier (a multiplier that takes 2, 4-bit inputs and generates an 8-bit result).

e) Design logic to compute the square of a 2-bit unsigned integer. How does this circuit compare to the one you designed in part a)?
Problem 3. “May Your Carries Overflowith”

In Lecture 10, overflows were defined as the case when either the sum two negative numbers gives a positive result, or the sum of two positive numbers gives negative result. This can be expressed as the following expression:

\[ V = A_{n-1}B_{n-1}S_{n-1} + A_{n-1}B_{n-1}S_{n+1} \]

where \( A_{n-1}, B_{n-1} \) are the inputs to most significant adder, and \( S_{n-1} \) is the most significant bit’s sum output.

a) An alternative way of computing overflow considers only the carry inputs and outputs of the most significant bit’s adder, and it can be expressed as: \( V = XOR(C_{\text{out}}, C_{\text{in}}) \). Prove these two different implementations give identical results.

b) Overflows can also be generated by subtraction. In particular, when a negative number subtracted from a positive number yields a negative result, or when a positive number is subtracted from a negative one and a positive result is generated. Compare both of the two given approaches for computing overflows in these cases. Do they work? If so, under what assumptions. If not, explain why.

c) We can also define a notion of overflow when adding unsigned numbers. Basically, an overflow occurs when the correct result cannot be represented given the finite representation. What logical combination of the condition codes \( Z \) (zero), \( N \) (negative), \( C \) (carry), and \( V \) (overflow) detect this case?

d) The distinction between the MIPS’ \textit{add} and \textit{addu} instructions is that the “unsigned” version of the instruction ignores overflows (i.e. it takes no special actions when an overflowed result is generated). Does this mean that the \textit{addu} can be used for signed 2’s-complement arithmetic? Explain why or why not.
Problem #4. “An Adder’s Bite”

Consider the follow two circuits for adding 3, 4-bit numbers (i.e. $W = X + Y + Z$):

![Circuit Diagram]

a) Ignoring the speed of the calculation, do both 3-input adder designs compute the same result? If not, how do they differ?

b) Assume a unit delay for each full-adder (i.e. the output will become valid one time unit after all inputs are valid). Which design computes all bits of the sum fastest? Which design computes the first (least-significant) bit fastest?

c) Obviously, design #1 uses one more adder than design #2. Explain how design #1 can be modified slightly to use one fewer full-adder. After this change, which design computes all bits of its sum fastest?

d) Both designs could be improved by incorporating carry-lookahead logic in place of their carry-propagation chains. Which design would requires less logic to implement carry lookahead? Explain.

e) Modify both designs to support the following operations:

$$W = X + Y + Z \quad \text{or} \quad W = X - Y - Z,$$

based on the value of a control signal (CTRL).