1 Introduction

In this lab preparation a ripple-carry cell will be designed at the transistor level as a CMOS compound gate using the 65 nm process. The design will then be analyzed in order to estimate the logical effort for the input of the gate, the worst-case propagation delay through an 8-bit ripple-carry circuit and the total transistor count, which is proportional to gate area.

2 Boolean Expression

The truth table for a ripple-carry cell is shown in figure 1. A carry cell computes the carry-out value resulting from the addition of three input bits (a, b and C_{in}) and outputs true when two or three inputs are true. This can be abstracted as a 3-input *majority function*, f_{maj}, a *monotonically increasing* function (ie. transitions in one direction on input causes transition in the same direction on the output) that can therefore not be implemented in a single CMOS-stage as CMOS gates always result in *monotonically decreasing/inverting* logic [1]. By this reasoning, however, $\overline{f_{\text{maj}}}$ is possible to implement.

The expression for this inverse function can be derived using a Karnaugh map that groups '0's and gives an expression in *conjunctive* form[1], that can then be converted to complemented *disjunctive* form using deMorgan's theorem

$$\begin{aligned} f(A,B,C_{\text{in}}) = \overline{f_{\text{maj}}} &= \overline{C_{\text{in}}A + C_{\text{in}}B + AB} \\ &= \overline{C_{\text{in}}(A+B) + AB} \end{aligned} \tag{1}$$

The nMOS pull-down network is thus given by the inverse function (as it pulls the output low when the function is true)

$$f_n = C_{\text{in}}(A+B) + AB \tag{2}$$

The derivation of the pMOS pull-up network is greatly facilitated by the *principle of duality*[2] using the following reasoning: if the pull-down network realizes a function f_{n} then we know that the CMOS gate will realize the inverse function $f_p = \overline{f_n(x_1, x_2, ...)}$. Duality then gives $f_p = \overline{f_n(x_1, x_2, ...)} = f_n^D(\bar{x}_1, \bar{x}_2, ...)$. Since pMOS transistors inherently invert their inputs, this becomes $f_p = f_n^D(x_1, x_2, ...)$. Thus, for the *majority* function

$$\begin{aligned} f_p = f_n^D &= (C_{\text{in}} + AB)(A+B) \\ &\Rightarrow C_{\text{in}}A + C_{\text{in}}B + AAB + ABB \\ &= C_{\text{in}}(A+B) + AB \end{aligned} \tag{3}$$

The result is that the *majority* function and its *dual* are equivalent! Due to this fact, we are now free to choose either of the *dual* functions for the networks. Another property of self-duality is that the function inverter (to finally reach the non-inverted carry-cell behaviour) may be placed either on the output or on the inputs.

A	B	Cin	Cout	S (not used)
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

Figure 1: *Truth table for 1-bit ripple-carry cell.*

[1]Also known as product-of-sums. Its counterpart is *disjunctive* form, or sum-of-products.

[2]The *dual* form of a function amounts to converting all ORs to ANDs and vice-versa. In transistor terms, this corresponds to replacing series connections with parallel connections and vice-versa. This also gives rise to a generalised form of deMorgan's theorem, stating

$$f^D(\bar{a}, \bar{b}, ...) = \overline{f(a, b, ...)}$$

3 Transistor-Level Schematic

The previous task gave a few different options for how the schematic could be drawn. One thing to consider is that it is beneficial to reduce length of any series path through the circuit, as each device has its own effective resistance and the width of every device in the path will need to be scaled up to compensate, resulting in a bigger area. The scaling of each device (normalized to the unit size for the process) is shown in schematic below.

An inspection of f_{maj} and ${f_{\text{maj}}}^D$ (eq.2 and 3 resp.) shows that although the same number of terms and gates are used, ${f_{\text{maj}}}^D$ has a maximum series path (terms ANDed together) of 3, whereas f_{maj} has a maximum of only 2. The network given by f_{maj} will therefore be used for both the pull-up and the pull-down networks, resulting in a symmetrical design.

Having the parallel devices connect to the rails rather than the output of the gate also results in a lower output drain-capacitance. The final schematic can be seen in figure 2.

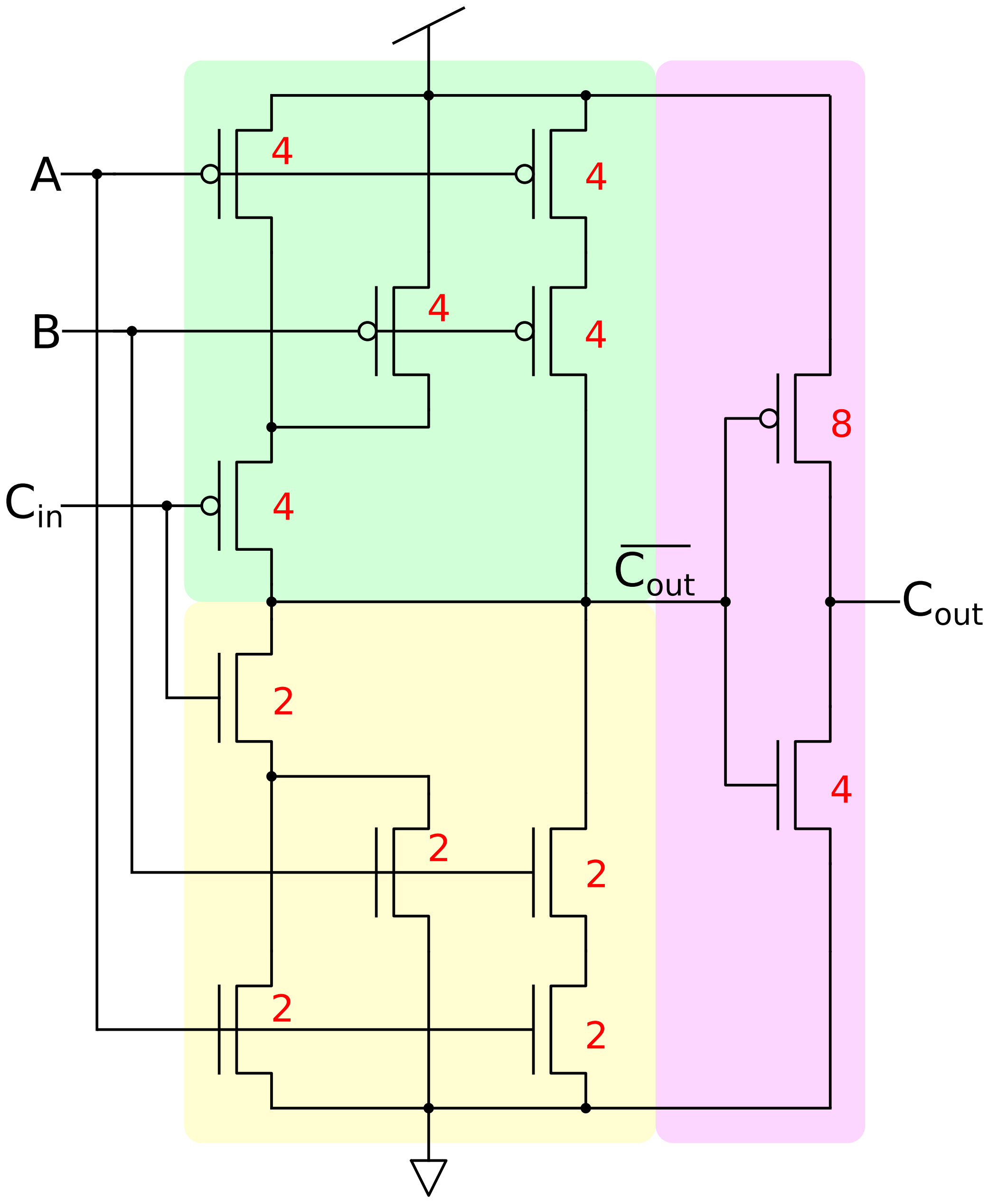

Figure 2: *Transistor-level schematic for a 1-bit ripple carry cell. Gate networks to the left, output inverter to the right. Relative scaling of transistors in red.*

4 Logical Effort and Parasitic Delay

Logical effort and parasitic delay are quantified relative to normalized input capacitance of a single CMOS inverter ie. 3C. The logical effort for the C_{in} input to the ripple-carry cell is

$$g_{\text{C}_{\text{in}}} = \frac{C_{\text{in}}}{C_{\text{inv}}} = \frac{4C + 2C}{3C} = 2$$

and the parasitic delay of the gate output is

$$P_{\text{C}_{\text{out}}} = \frac{4C + 4C + 2C + 2C}{3C} \cdot p_{\text{inv}} = 4p_{\text{inv}}$$

The inverter, of course, has $g_{\text{inv}} = 1$ and $p = p_{\text{inv}}$ since it is normalized relative to itself.

5 Worst-Case Propagation Delay

The schematic in fig 2 can be drawn as a block diagram below.

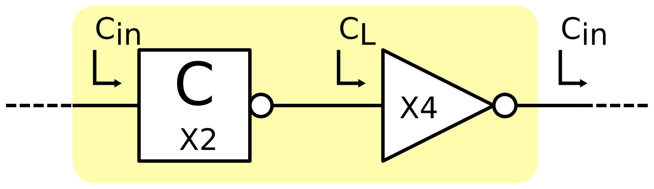

Figure 3: Block diagram showing a single link of a ripple carry adder. Input capacitances at each stage are labeled.

Due to the relative scaling of the stages $C_L = 2C_{\text{in}}$. Therefore the electrical effort h for the first stage is

$$h_1 = \frac{C_{\text{L}}}{C_{\text{in}}} = \frac{2C_{\text{in}}}{C_{\text{in}}} = 2$$

Due to the repeating nature of the ripple-carry adder the next load will be the input to the next carry-cell. The electrical effort for the second stage is therefore

$$h_2 = \frac{C_{\text{L}}}{C_{\text{in}}} = \frac{C_{\text{in}}}{2C_{\text{in}}} = \frac{1}{2}$$

The normalized delay for a single ripple-carry cell is thus

$$\begin{aligned} d &= g_1 \cdot h_1 + p_1 + g_2 \cdot h_2 + p_2 \\ &= 2 \cdot 2 + 4p_{\text{inv}} + 1 \cdot \frac{1}{2} + 1 \\ &= 4.5 + 5p_{\text{inv}} \end{aligned}$$

The time constant τ was determined in *lab prep 1* to be 5ps for this particular process. The worst case time-delay for a single carry cell is thus

$$t_d = d \cdot \tau = 5 \cdot (4.5 + 5p_{\text{inv}}) \approx 47.5ps$$

and for an 8-bit ripple-carry adder

$$8 * 47.5ps = 380ps$$

, where p_{inv} is assumed to be ≈ 1.

6 Summary

The ripple-carry cell (or *majority function* gate) can be designed using symmetrical pullup and pulldown networks. Alternatives where considered and shown to result in more area usage and higher output-drain capacitance. The fully realized carry logic uses 12 transistors: 10 for the inverting ripple-carry cell and 2 for output inverter. An 8-bit ripple carry circuit therefore uses 96 transistors.

References

[1] William James. Dally, R. Curtis. Harting, and Tom M. Aamodt. *Digital design using VHDL: a systems approach*. Cambridge University, 2016. Chap. 4.3.3, pp. 72–73.