Ripple Carry and Carry Lookahead Adders

1 Introduction

We will start by explaining the operation of one-bit full adder which will be the basis for constructing ripple carry and carry lookahead adders.

1.1 One-bit full adder

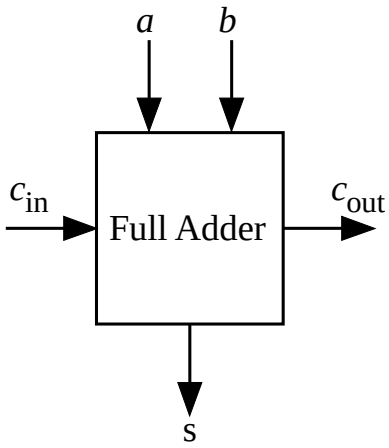

Figure 1: One-bit full adder.

A one-bit full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs $(a,\ b,$ and $c_{in})$ and two outputs $(s$ and $c_{out})$ as illustrated in Figure 1. The truth table of the full adder is listed in Table 1. The gate implementation of 1-bit full adder is shown in Figure 2.

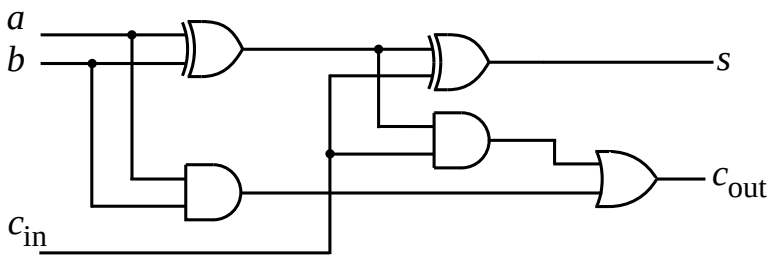

Figure 2: Gate implementation of full adder.

Table 1: Full adder truth table.

a	b	c_{in}	c_{out}	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

1.2 Ripple carry adder

A ripple carry adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascaded (see section 2.1), with the carry output from each full adder connected to the carry input of the next full adder in the chain. Figure 3 shows the interconnection of four full adder (FA) circuits to provide a 4-bit ripple carry adder. Notice from Figure 3 that the input is from the right side because the first cell traditionally represents the least significant bit (LSB). Bits a_0 and b_0 in the figure represent the least significant bits of the numbers to be added. The sum output is represented by the bits s_0–s_3.

1.3 Ripple carry adder delays

In the ripple carry adder, the output is known after the carry generated by the previous stage is produced. Thus, the sum of the most significant bit is only available after the carry signal has rippled through the adder from the least significant stage to the most significant stage. As a result, the final sum and carry bits will be valid after a considerable delay.

Table 2 shows the delays for several CMOS gates assuming all gates are equally loaded for simplicity. All delays are normalized relative to the delay of a simple inverter. The table also shows the corresponding gate areas normalized to a simple minimum-area inverter. Note from the table

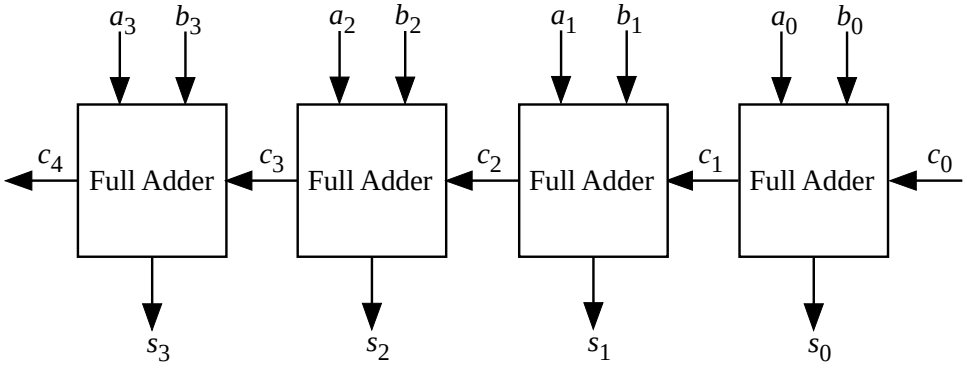

Figure 3: 4-bit full adder.

that multiple-input gates have to use a different circuit technique compared to simple 2-input gates.

Table 2: CMOS gate delays and areas normalized relative to an inverter.

Gate	**Delay**	**Area**	**Comment**
Inverter	1	1	Minimum delay
2-input NOR	1	3	More area to produce delay equal to that of an inverter
2-input NAND	1	3	More area to produce delay equal to that of an inverter
2-input AND	2	4	Composed of NAND followed by inverter
2-input OR	2	4	Composed of NOR followed by inverter
2-input XOR	3	11	Built using inverters and NAND gates
-input OR	2	$n/3+2$	Uses saturated load ($n > 2$).
-input AND	3	$n/3+2$	Uses -input OR preceded by inverters ($n > 2$).

Using Table 2 and the schematic of Figures 2 and 3, we can estimate the delays associated with the outputs of the ripple carry adder stages as indicated in Table 3. The delays are normalized relative to an inverter delay.

For an -bit ripple carry adder the sum and carry bits of the most significant bit (MSB) are obtained after a normalized delay of

$$\text{Sum } s_{n-1} \text{ delay} = 4n + 2 \quad (1)$$

$$\text{Carry } c_n \text{ delay} = 4n + 3 \quad (2)$$

For a 32-bit processor, the carry chain normalized delay would be 131. The ripple carry adder can get very slow when many bits need to be added. In fact, the carry chain propagation delay is the determining factor in most microprocessor speeds.

Table 3: Delays for the outputs of a 4-bit ripple carry adder normalized to an inverter delay.

Signal	Delay
s_0, c_1	6, 7
s_1, c_2	10, 11
s_2, c_3	14, 15
s_3, c_4	18, 19

1.4 Carry lookahead adder (CLA)

The carry lookahead adder (CLA) solves the carry delay problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases: (1) when both bits a and b_i are 1, or (2) when one of the two bits is 1 and the carry-in is 1. Thus, one can write,

$$c_{i+1} = a_i.b_i + (a_i \oplus b_i).c_i \quad (3)$$
$$s_i = (a_i \oplus b_i) \oplus c_i \quad (4)$$

The above two equations can be written in terms of two new signals and G , which are shown in Figure 4:

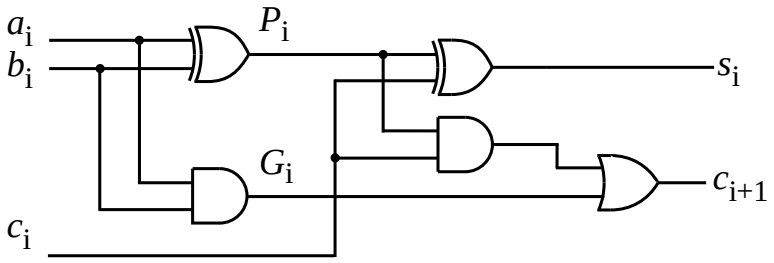

Figure 4: Full adder at stage i with and G shown.

$$c_{i+1} = G_i + P_i.c_i \quad (5)$$
$$s_i = P_i \oplus c_i \quad (6)$$

where

$$G = a_i.b_i \quad (7)$$
$$= a_i \oplus b_i \quad (8)$$
$$(9)$$

G_i and P_i are called the carry generate and carry propagate terms, respectively. Notice that the generate and propagate terms only depend on the input bits and thus will be valid after one and

two gate delay, respectively. If one uses the above expression to calculate the carry signals, one does not need to wait for the carry to ripple through all the previous stages to find its proper value. Let's apply this to a 4-bit adder to make it clear.

Putting $i = 0, 1, 2, 3$ in Equation 5, we get

$$c_1 = G_0 + {}_0.c_0 \tag{10}$$
$$c_2 = G_1 + {}_1.G_0 + {}_1.P_0.c_0 \tag{11}$$
$$c_3 = G_2 + {}_2.G_1 + P_2.P_1.G_0 + {}_2.P_1.P_0.c_0 \tag{12}$$
$$c_4 = G_3 + {}_3.G_2 + {}_3.P_2.G_1 + {}_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.c_0 \tag{13}$$

Notice that the carry-out bit, c_{i+1}, of the last stage will be available after four delays: two gate delays to calculate the propagate signals and two delays as a result of the gates required to implement Equation 13.

Figure 5 shows that a 4-bit CLA is built using gates to generate the P and G signals and a logic block to generate the carry out signals according to Equations 10–13.

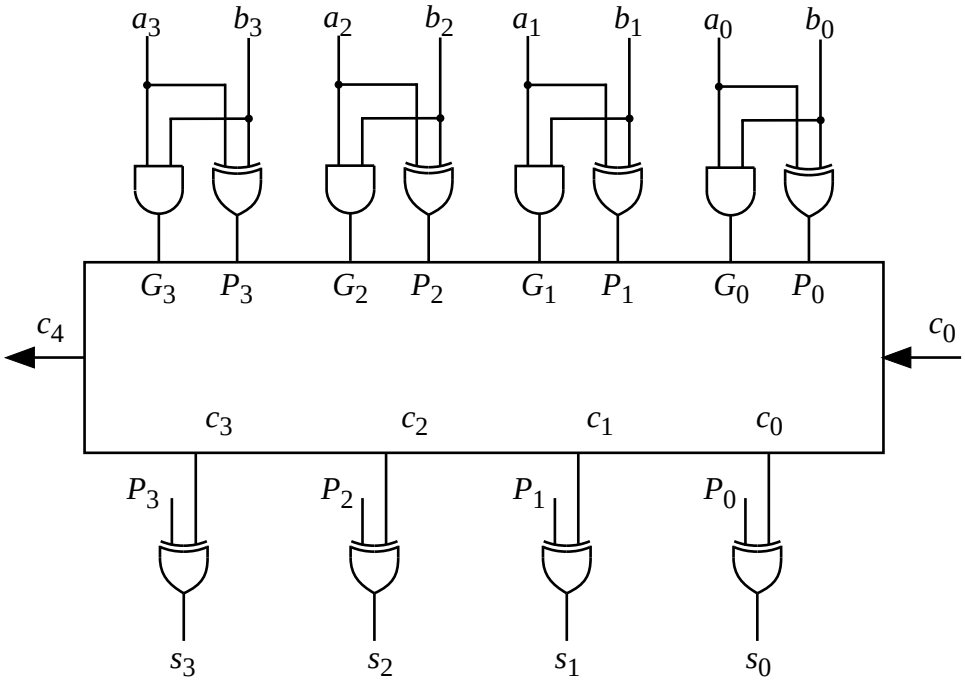

Figure 5: 4-Bit carry lookahead adder implementation detail.

The disadvantage of CLA is that the carry logic block gets very complicated for more than 4-bits. For that reason, CLAs are usually implemented as 4-bit modules and are used in a hierarchical structure to realize adders that have multiples of 4-bits.