Carry-Lookahead Adder In Math

Nakidai Perumenei

September 2025

1 Definitions

In this paper bits are counted from 0 (least significant) upwards; A is one number; B is another number; G is generate; P is propagate; C is carry; C_{in} is C_{i-1} — carry bit given to an adder's input; S is sum; i, j, k are bit indexes.

2 Introduction

To start with, CLA is a mechanism for calculating the carry bits independently of each other. It is used in hardware, because, unlike in software, here it is possible to do a lot of parallel calculations, which is more efficient than serial.

Before speaking about carry, it is important to know about generating and propagating it.

A pair generates a carry if both bits are 1. In math it is written as $G_i \equiv A_i \wedge B_i$,

A pair propagates a carry if at least one bit is 1, speaking math $P_i \equiv A_i \vee B_i$.

Now, carry bit is set if the current pair of bits generates it, or so does the previous one and the current propagates. Formula for the current carry is $C_i \equiv G_i \vee P_i \wedge C_{i-1}$.

Of course, it is possible to unwrap this formula: $C_i \equiv G_i \vee P_i \wedge C_{i-1} \equiv G_i \vee P_i \wedge (G_{i-1} \vee P_{i-1} \wedge C_{i-2})$. And also it is possible to simplify this to a very simple polynomial: $C_i \equiv G_i \vee P_i \wedge G_{i-1} \vee P_i \wedge P_{i-1} \wedge C_{i-2}$. So it is also correct to define that carry is produced if current pair generates it, or so does anyone before and all intermediate pairs propagate it.

Then, having a carry it is easy to calculate the sum: $S_i \equiv A_i \oplus B_i \oplus C_i$.

Though, it is not needed for P_i to be defined with OR as if $A_i = B_i = 1 \Rightarrow G_i \equiv A_i \wedge B_i = 1$. So we can define $P_i \equiv A_i \oplus B_i$, therefore sum is $S_i \equiv P_i \oplus C_i$. Also, as now G_i and P_i do not overlap, carry calculation can be written with XOR: $C_i \equiv G_i \oplus P_i \wedge C_{i-1}$.

If to expand sum, then $S_i \equiv P_i \oplus G_i \oplus P_i \wedge C_{i-1}$.

Obviously, as the formula is recursive it is needed to have an edge case. Stop condition there is $G_{i-1} \equiv C_{\text{in}}$.

i	A	B	P	G	C	S
0	1	1	0	1	0	0
1	1	0	1	0	1	0
2	0	1	1	0	1	0
3	0	0	0	0	1	1

Table 1: intermediate results

3 Adder Definition

With all that information, it is possible to define a CLA adder in math terms:

$$
\begin{aligned}
&P_i \equiv A_i \oplus B_i \\
&G_i \equiv \begin{cases} C_{\text{in}} & \text{if } i = -1 \\ A_i \wedge B_i & \text{otherwise} \end{cases} \\
&C_i \equiv \oplus_{j=-1}^{i-1} G_j \Pi_{k=j+1}^{i-1} P_k \\
&S(A,B) \equiv \textstyle\sum_{i=0}^{\lfloor \log_2 \max(A,B) \rfloor + 1} 2^i (P_i \oplus C_i)
\end{aligned}
$$

4 Example

Let us test this adder with a simple pair: 3 and 5. For ease of calculation, it is convenient to represent them in binary form: 11_2 and 101_2. Formula will iterate over more binary digits than present, so it is also convenient to align these numbers with zeroes: 0011_2 and 0101_2. Also let C_in be empty.

Table 1 shows all the results.

It's quite easy to calculate P and G: they are just 1 operation. Also it's important to keep in mind that $G_{-1} = 0$.

Next step is to calculate carries. Let us start from the C_0:

$$C_0 = \oplus_{j=-1}^{-1} G_j \Pi_{k=j+1}^{-1} P_k = G_{-1} = 0$$

After unwrapping all the scary operators we get that it just depends on $G_{-1} = 0$. Easy. Then:

$$C_1 = \oplus_{j=-1}^{0} G_j \Pi_{k=j+1}^{0} P_k = G_{-1} \wedge P_0 \oplus G_0 = 0 \wedge 0 \oplus 1 = 1$$

Now G_{-1} is in the same monomial as P_0, and G_0 is added. Then:

$$
\begin{aligned}
C_2 &= \oplus_{j=-1}^{1} G_j \Pi_{k=j+1}^{1} P_k \\
&= G_{-1} \wedge P_0 \wedge P_1 \oplus G_0 \wedge P_1 \oplus G_1 \\
&= 0 \oplus 1 \wedge 1 \oplus 0 = 1
\end{aligned}
$$

The tendency is clear. And the last carry:

$$
\begin{aligned}
C_3 &= \oplus_{j=-1}^{2} G_j \Pi_{k=j+1}^{2} P_k \\
&= G_{-1} \wedge P_0 \wedge P_1 \wedge P_2 \oplus G_0 \wedge P_1 \wedge P_2 \oplus G_1 \wedge P_2 \oplus G_2 \\
&= 0 \oplus 1 \oplus 0 \oplus 0 = 1
\end{aligned}
$$

The last thing to do is to XOR P and C to get the sum, which is $1000_2 = 8 = 5 + 3$. Adder works just as expected.