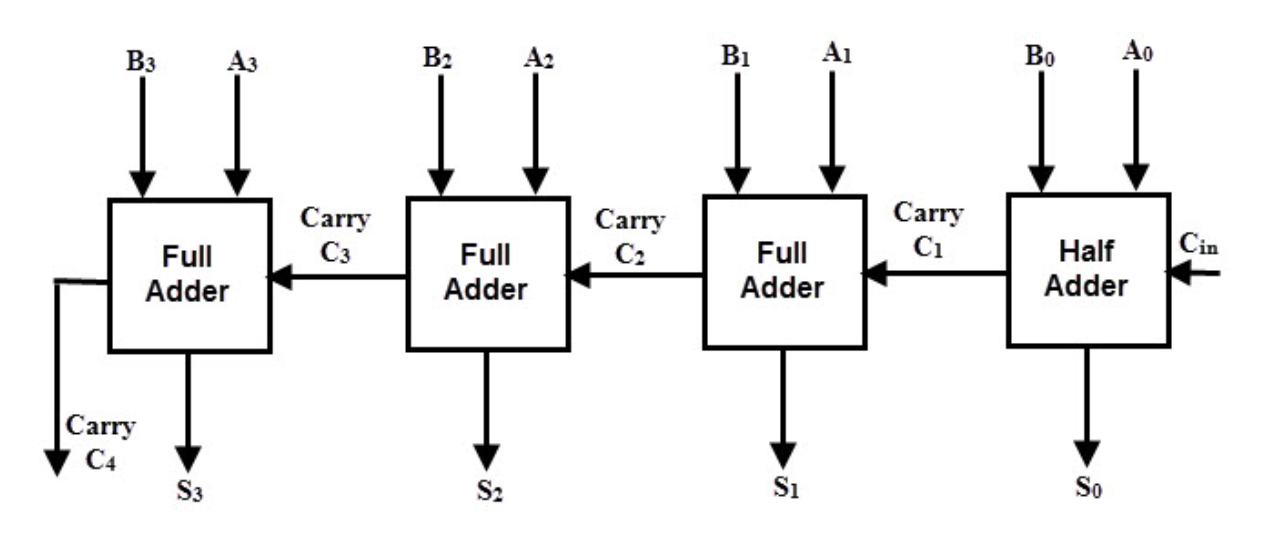
Research on Faster Adder and Divisor

# 1. Faster Adder: Carry Look-Ahead Adder

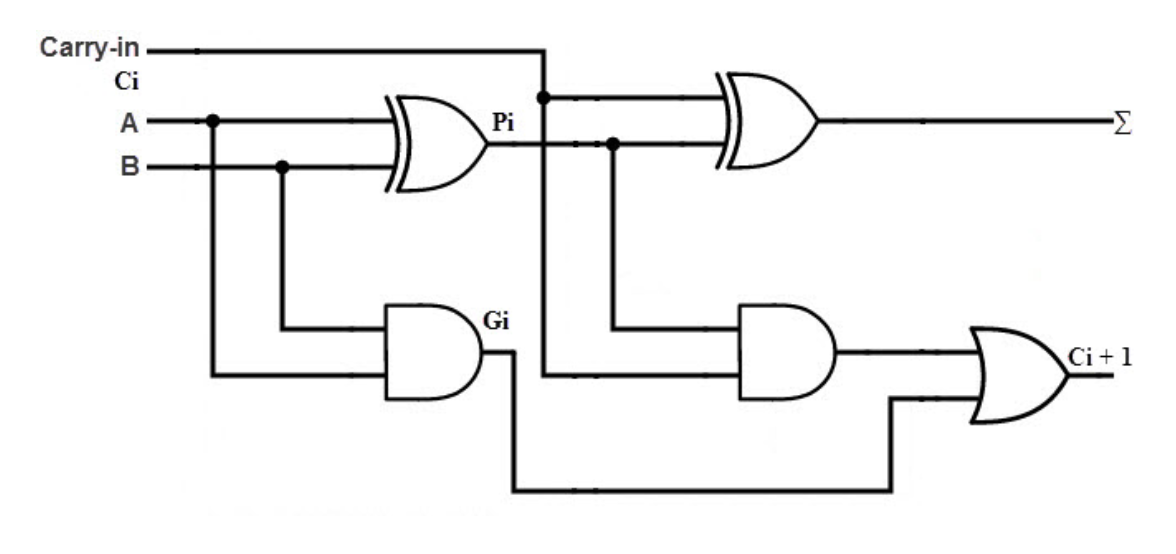
The key for carry look-ahead adder is using a complicated circuit to reduce the carry delay time.

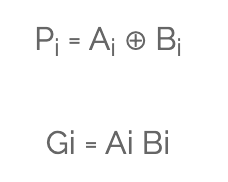
Details:

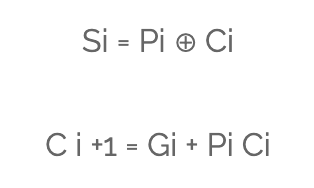
In case of parallel adders, the higher-order stage needs to wait for the lower-order stage’s carry to continue. It is impossible to produce the sum and output carry until the input carry occurs.



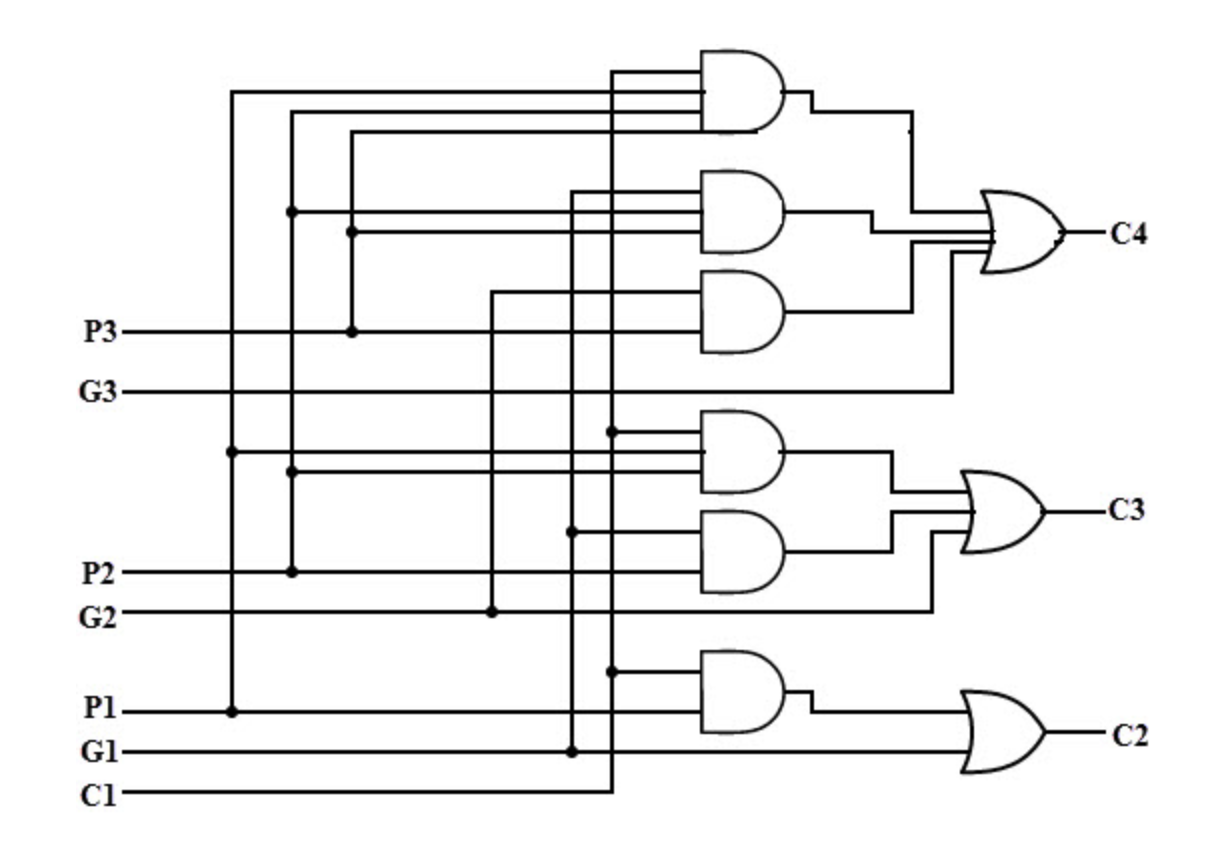
To add faster, we should consider a way to get the carry faster, and that makes carry look-ahead adder. We can use complicated circuit to reduce the carry delay time.



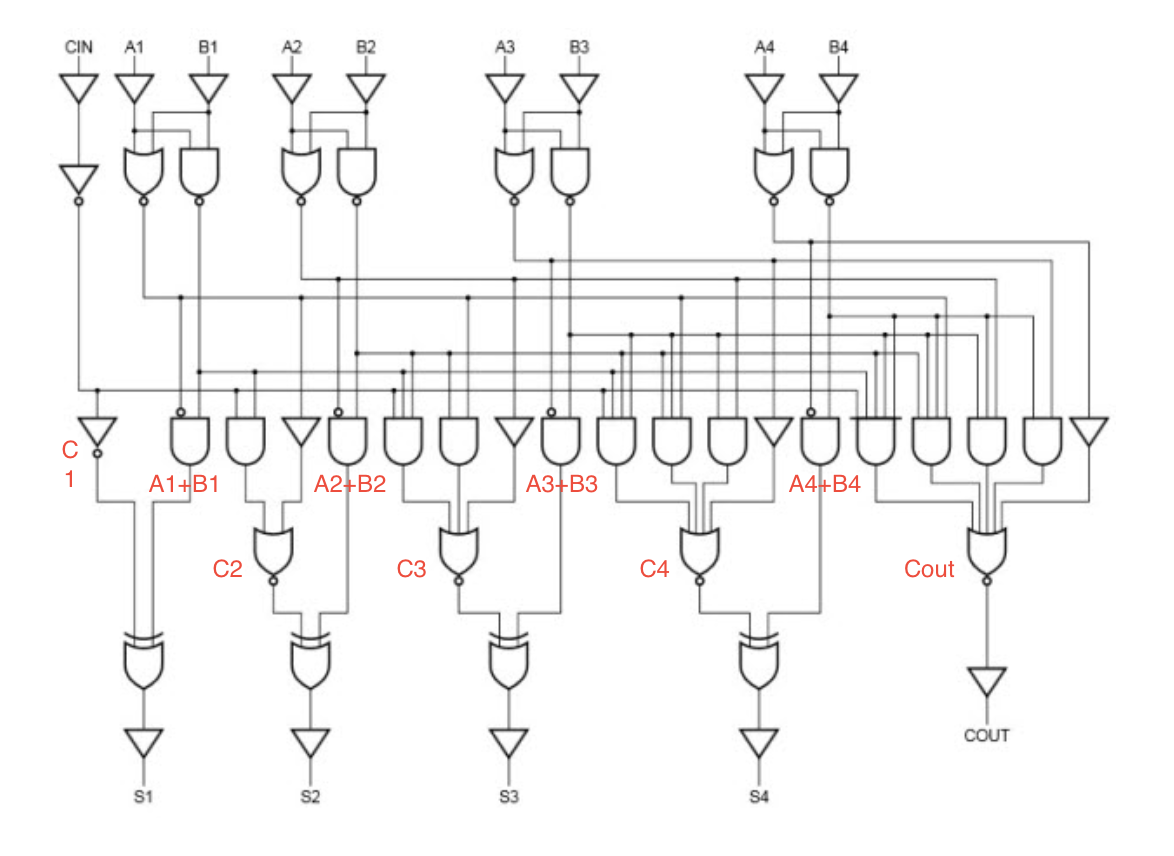




From this base, we can build a 4 stage carry look-ahead adder.



Once all input (C1, P1, G1, P2, G2, P3, G3) is set, we can get output carry C2, C3, C4 immediately. Then we add carry with the result of An plus Bn to calculate the sum.



I exacted the above content based on <https://www.electronicshub.org/carry-look-ahead-adder/>

# 2. Faster Divisor: SRT divisor

The key idea of SRT divisor is to guess the quotient digit based on a few of the most significant divisor and partial remainder bits, rather than computing it exactly.

Details:

Named for its creators (Sweeney, Robertson, and Tocher), SRT division is a popular method for division in many [microprocessor](https://en.wikipedia.org/wiki/Microprocessor) implementations. SRT division is similar to non-restoring division, but it uses a lookup table on the dividend and divisor to determine each quotient digit.

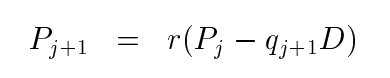
Quotient digit selection is faster because it only involves a few most significant bits. As long as the guess is close enough, the algorithm can correct on subsequent steps using redundant quotient digits. We can use a negative digit like -1 to correct on a later step for an incorrect guess in a previous step.

Radix needed to be chose with a balance between a minimal latency and a less complicated generation of all required divisor multiples. Once we decided the radix, we could have the digit set. For example, radix-2 for {-1, 0, 1}, radix-4 for {-2, -1, 0, 1, 2}(minimally redundant) or {-3, -2, -1, 0, 1, 2, 3}(maximally redundant).

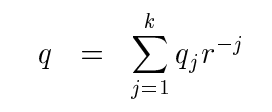
We set the initial partial remainder P0 to the dividend. On step j, we compute the next quotient digit qj+1 by comparing multiples of the divisor D to the current partial remainder:



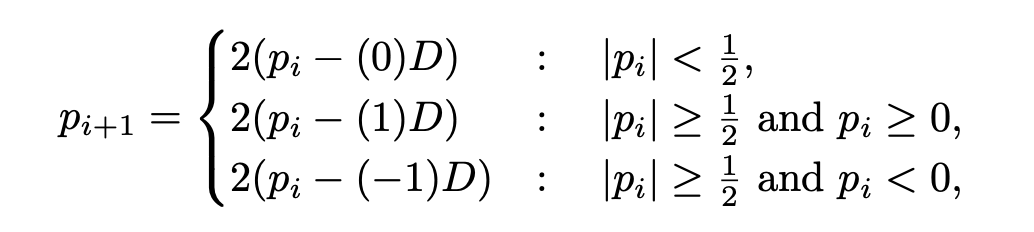
Then, we can get the next partial remainder Pj+1 by subtracting the selected divisor multiple from the current partial remainder and shifting the difference:



The final quotient after k iterations is the weighted sum of the quotient digits:



Now, I use a set {-1, 0, 1} of redundant digits to show 0.67/0.75.



1) p0 = 0.67 -> q0 = 1 ((|0.67| > 0.5) && (0.67 >= 0)) -> p1 = 2 \* (0.67 – 1 \* 0.75) = -0.16

2) p1 = -0.16 -> q1 = 0 -> p2 = 2 \* (-0.16 – 0 \* 0.75) = 2 \* p1 = -0.32

3) p2 = -0.32 -> q2 = 0 -> p3 = 2 \* p2 = -0.64

4) p3 = -0.64 -> q3 = -1 -> p4 = 2 \* (-0.64 – (-1) \* 0.75) = 0.22

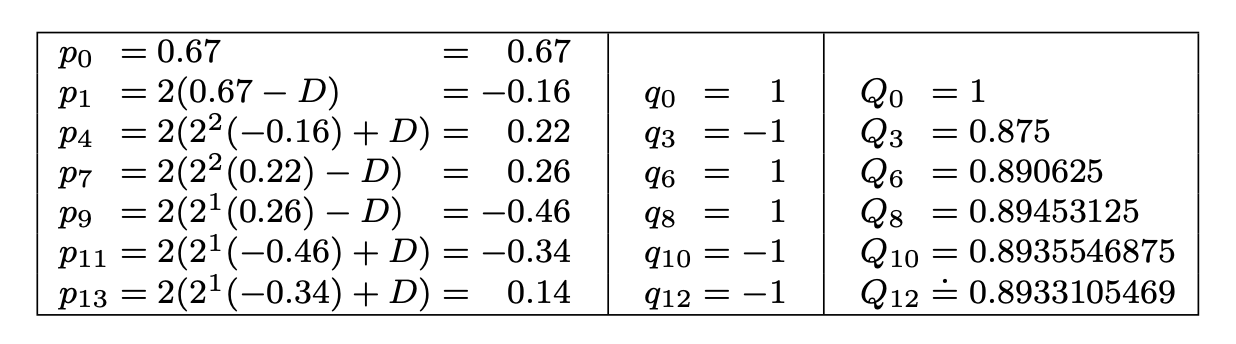
5) p4 = 0.22 -> q4 = 0 -> p5 = 2 \* p4 = 0.44

6) p5 = 0.44 -> q5 = 0 -> p6 = 2 \* p5 = 0.88

7) p6 = 0.88 -> q6 = 1 -> p7 = 2 \* (0.88 – 0.75) = 0.26

8) p7 = 0.26 -> q7 = 0 -> p8 = 2 \* p7 = 0.52

……



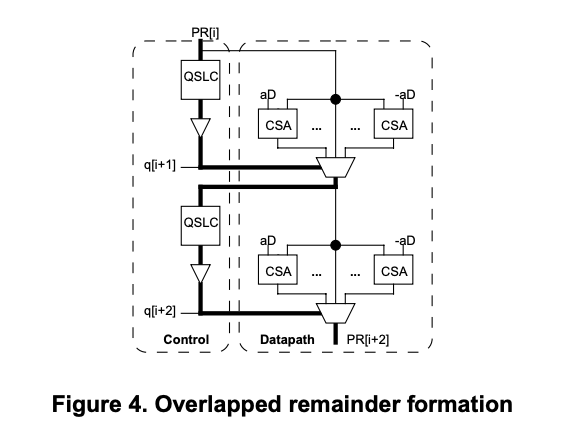
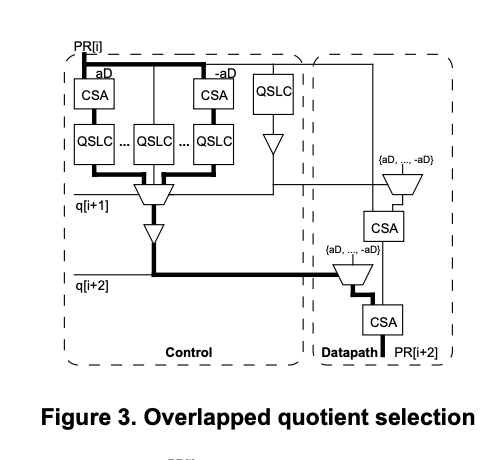
Q0 = 2^0

Q1 = 2^0 – 2^(-3) = 0.875

Q2 = 2^0 – 2^(-3) + 2(-6) = 0.890625

……

From above can we see the convenience if we use SRT division in binary. Engineers choose is non-overlapped and overlapped quotient selection design according to applications. A simple non-overlapped design is common in low-cost applications, like as the Intel Pentium Processor.



References:

1. Harris, David L.; Oberman, Stuart F.; Horowitz, Mark A. (9 September 1998). [*SRT Division: Architectures, Models, and Implementations*](http://pages.hmc.edu/harris/research/srtlong.pdf) (PDF) (Technical report). Stanford University.

2. McCann, Mark; Pippenger, Nicholas (2005). ["SRT Division Algorithms as Dynamical Systems"](http://scholarship.claremont.edu/cgi/viewcontent.cgi?article=1094&context=hmc_fac_pub). *SIAM Journal on Computing*. **34** (6): 1279–1301. [CiteSeerX](https://en.wikipedia.org/wiki/CiteSeerX) [10.1.1.72.6993](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.6993). [doi](https://en.wikipedia.org/wiki/Digital_object_identifier):[10.1137/S009753970444106X](https://doi.org/10.1137%2FS009753970444106X).

Last but not least, SRT divisor is still slow divisor. Fast division methods start with a close approximation to the final quotient and produce twice as many digits of the final quotient on each iteration. [Newton–Raphson](https://en.wikipedia.org/wiki/Division_algorithm#Newton%E2%80%93Raphson_division) and [Goldschmidt](https://en.wikipedia.org/wiki/Division_algorithm#Goldschmidt_division) algorithms fall into this category.