Semiconductor Technology from A to Z

Digital technology

Arithmetic Circuits: Adders and Multipliers

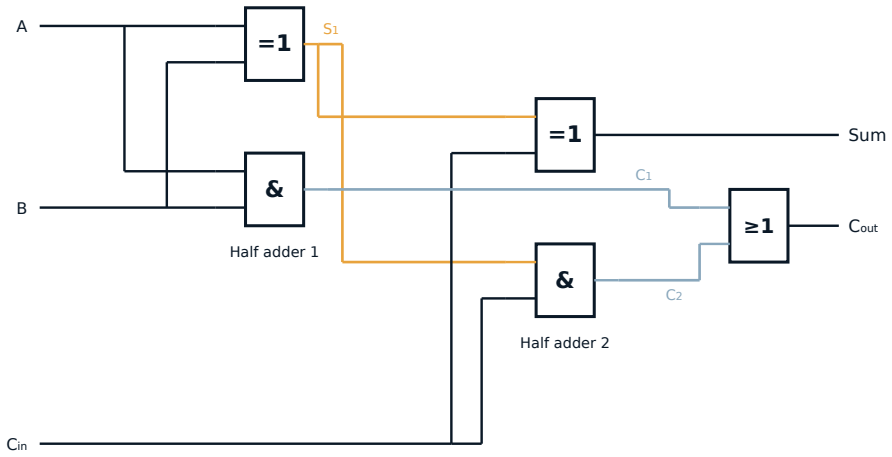
The Full Adder as a Basic Building Block

The simplest arithmetic building block is the half adder, which adds two individual bits and produces a sum (the XOR of the inputs) and a carry (the AND of the inputs). Since chaining multiple bit positions also requires accounting for a carry coming in from the previous position, in practice one uses the full adder, which combines three inputs (the two bits A and B to be added, plus the carry-in C_{in} from the previous position) into a sum and a carry-out C_{out}. A full adder can be built from two half adders plus an additional OR gate, and typically requires about 20 to 28 transistors in a standard CMOS cell.

The Boolean equation C_{out} = (A AND B) OR (C_{in} AND (A XOR B)) can be read intuitively as a majority vote: C_{out} becomes 1 exactly when at least two of the three inputs (A, B, C_{in}) are 1 – regardless of which two they are. For A=1, B=1, C_{in}=0, for instance, A AND B alone already produces the carry; for A=1, B=0, C_{in}=1, A and B differ (A XOR B = 1), and C_{in} AND (A XOR B) produces it instead. The sum, by contrast, is 1 when an odd number of the three inputs is 1 (1 or 3 out of 3) – exactly the behavior of a three-input XOR.

Full Adder Built from Two Half Adders

A, B, and Cin produce the sum and carry-out Cout



Sum = A ⊕ B ⊕ Cin – Cout = (A • B) + (Cin • (A ⊕ B))

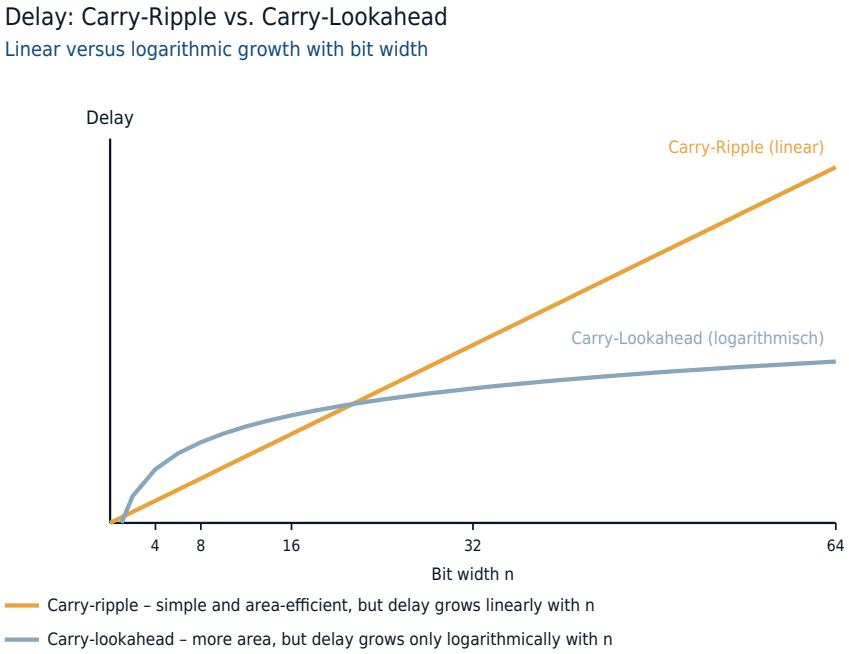
Symbols per IEC 60617: & = AND, =1 = XOR, ≥1 = OR

Chaining several full adders together, with the carry output of each stage feeding the carry input of the next higher stage, produces the simplest multi-bit

adder: the carry-ripple adder.

Carry-Ripple vs. Carry-Lookahead Adders

In a carry-ripple adder, the carry must "ripple" sequentially through every bit position (hence the name) before the final result is available. A concrete example illustrates this: in the 4-bit addition 0111 + 0001, position 0 first produces $C_{out,0}$=1 (since A_0=B_0=1); this carry must arrive at position 1 before Sum_1 and $C_{out,1}$ are determined there, which in turn must arrive at position 2, and so on – only once the carry has propagated through all four stages is the correct final result, 1000, fully available. In a 32-bit adder, the carry signal in the worst case propagates through all 32 full-adder stages one after another, so the delay grows linearly with the bit width.

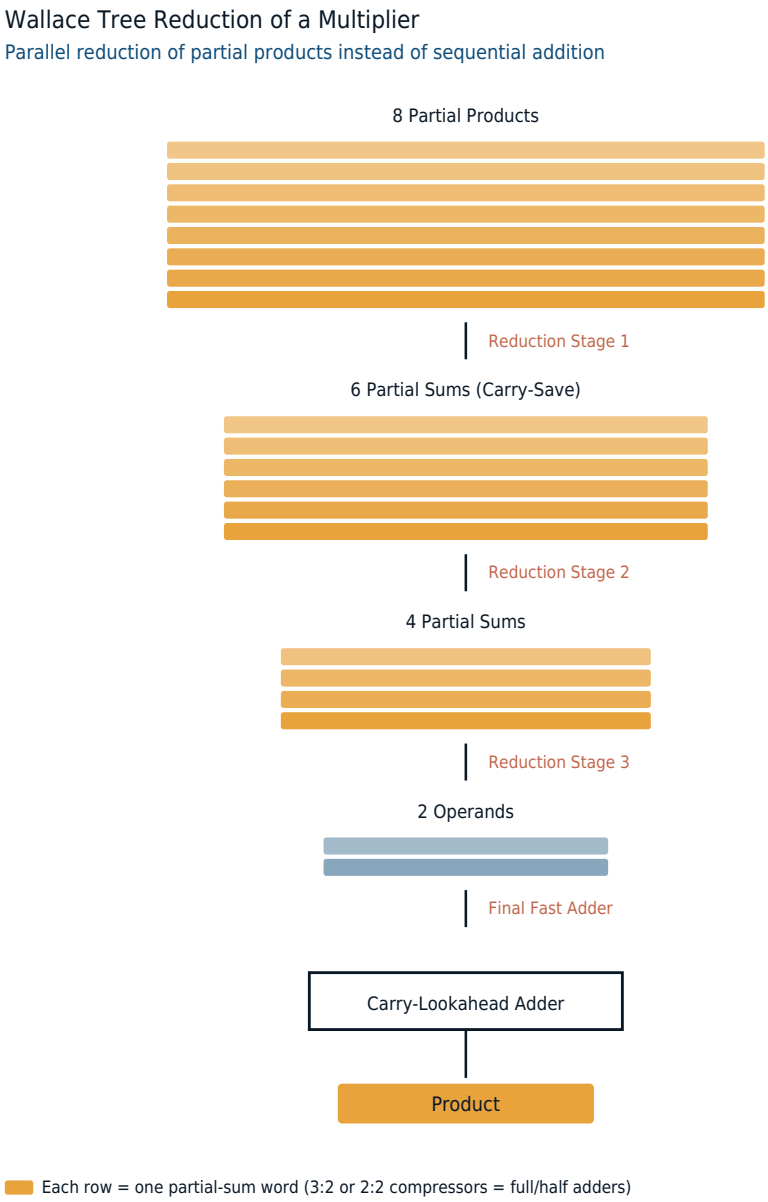


The carry-lookahead adder solves this problem by precomputing, for every bit position i, two auxiliary signals: generate G_i = A_i AND B_i (a carry is always generated here, regardless of the incoming carry) and propagate P_i = A_i XOR B_i (an incoming carry would be passed through). From these, every carry can be expressed directly as a sum of product terms, for example C_1 = G_0 + P_0C_0, C_2 = G_1 + P_1G_0 + $P_1P_0C_0$, C_3 = G_2 + P_2G_1 + $P_2P_1G_0$ + $P_2P_1P_0C_0$ – so each carry depends only on the input bits themselves, no longer on a sequentially computed predecessor carry, and can be computed in parallel for all positions with a fixed, small gate delay. Because the number of AND-gate inputs (fan-in) grows with each additional bit position, real lookahead adders are usually built in blocks of

4 bits, with the block carries combined at a second lookahead level to keep the fan-in bounded – a trade-off that explains the logarithmic rather than constant delay, but at the cost of significantly more area and wiring overhead, as seen in many areas of digital circuit design.

Multipliers: From Array to Wallace Tree

The simplest multiplier architecture is the array multiplier, which mimics classic long multiplication. For each bit of the multiplier, a partial product is first formed: bit i of the multiplier ANDed with every bit of the multiplicand, shifted left by i positions (exactly as in manual long multiplication). For two 4-bit operands, this produces four partial-product rows. These are added row by row: a row of full adders sums partial products 1 and 2 column by column, producing a sum row and a carry row; the carry row is shifted one position left and added together with the sum row to the next partial-product row, and so on in a staircase fashion through all rows. Since each row addition must complete before the next can begin, this sequential chaining dominates the delay, which grows roughly linearly with the bit width.



Faster multipliers instead use a Wallace tree or Dadda tree: instead of adding row by row in sequence, in every column three partial-product bits at a time – regardless of which original row they came from – are reduced by a full adder acting as a 3:2 compressor into two output bits (with only two bits remaining, a half adder acts as a 2:2 compressor instead). Because all columns are processed in parallel within the same reduction stage, rather than waiting on the result of the previous row, the number of rows shrinks from stage to stage by roughly a 3:2 ratio, until only two operands remain, which are finally combined with a single fast adder (often a carry-lookahead adder). Combined with Booth encoding, which reduces the number of required partial products by nearly half through clever grouping of multiple multiplier bits, Wallace-tree multipliers are

the standard architecture in modern processors, digital signal processors, and floating-point units.