Halbleitertechnologie von A bis Z

Digitaltechnik

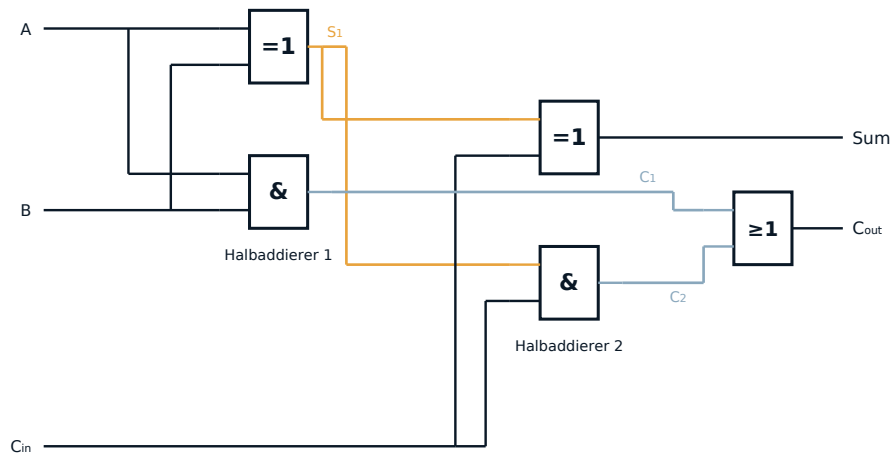
Arithmetik-Schaltungen: Addierer und Multiplizierer

Der Volladdierer als Grundbaustein

Der einfachste arithmetische Baustein ist der Halbaddierer, der zwei einzelne Bits addiert und dabei eine Summe (XOR der Eingänge) und einen Übertrag (AND der Eingänge) liefert. Da beim Verketten mehrerer Bitstellen jedoch auch ein Übertrag aus der vorherigen Stelle berücksichtigt werden muss, verwendet man in der Praxis den Volladdierer, der drei Eingänge (die beiden zu addierenden Bits A und B sowie den Übertrag C_{in} aus der vorherigen Stelle) zu einer Summe und einem Übertrag C_{out} verrechnet. Ein Volladdierer lässt sich aus zwei Halbaddierern und einem zusätzlichen OR-Gatter aufbauen und benötigt in einer typischen CMOS-Standardzelle etwa 20 bis 28 Transistoren.

Die Boolesche Gleichung C_{out} = (A AND B) OR (C_{in} AND (A XOR B)) lässt sich anschaulich als Mehrheitsentscheid lesen: C_{out} wird genau dann 1, wenn mindestens zwei der drei Eingänge (A, B, C_{in}) auf 1 liegen – ganz unabhängig davon, welche zwei das sind. Bei A=1, B=1, C_{in}=0 etwa liefert bereits A AND B den Übertrag; bei A=1, B=0, C_{in}=1 sind A und B unterschiedlich (A XOR B = 1), und C_{in} AND (A XOR B) liefert ihn stattdessen. Die Summe dagegen ist 1, wenn eine ungerade Anzahl der drei Eingänge 1 ist (1 oder 3 von 3) – exakt das Verhalten einer dreifachen XOR-Verknüpfung.

Volladdierer aus zwei Halbaddierern
A, B und Cin ergeben Summe und Übertrag Cout

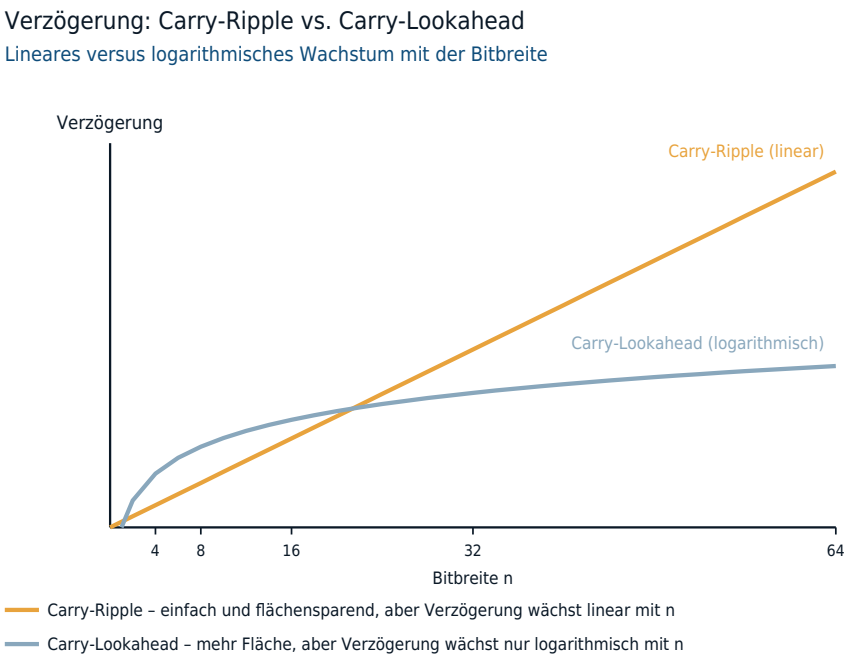


Sum = A ⊕ B ⊕ Cin – Cout = (A • B) + (Cin • (A ⊕ B))
Symbole nach IEC 60617: & = AND, =1 = XOR, ≥1 = OR

Reiht man mehrere Volladdierer aneinander, wobei der Übertragsausgang jeder Stelle zum Übertragseingang der nächsthöheren Stelle wird, entsteht der einfachste Mehrbit-Addierer: der Carry-Ripple-Addierer.

Carry-Ripple- vs. Carry-Lookahead-Addierer

Beim Carry-Ripple-Addierer muss der Übertrag nacheinander durch alle Bitstellen "durchsickern" (daher der Name), bevor das Endergebnis feststeht. Ein konkretes Beispiel verdeutlicht das: Bei der 4-Bit-Addition 0111 + 0001 entsteht an Stelle 0 zunächst $C_{out,0}=1$ (da $A_0=B_0=1$); dieser Übertrag muss an Stelle 1 ankommen, bevor dort Sum_1 und $C_{out,1}$ feststehen, dieser wiederum an Stelle 2, und so weiter – erst wenn der Übertrag alle vier Stufen durchlaufen hat, liegt das korrekte Endergebnis 1000 vollständig an. Bei einem 32-Bit-Addierer durchläuft das Übertragssignal im ungünstigsten Fall alle 32 Volladdiererstufen nacheinander, wodurch die Verzögerung linear mit der Bitbreite wächst.

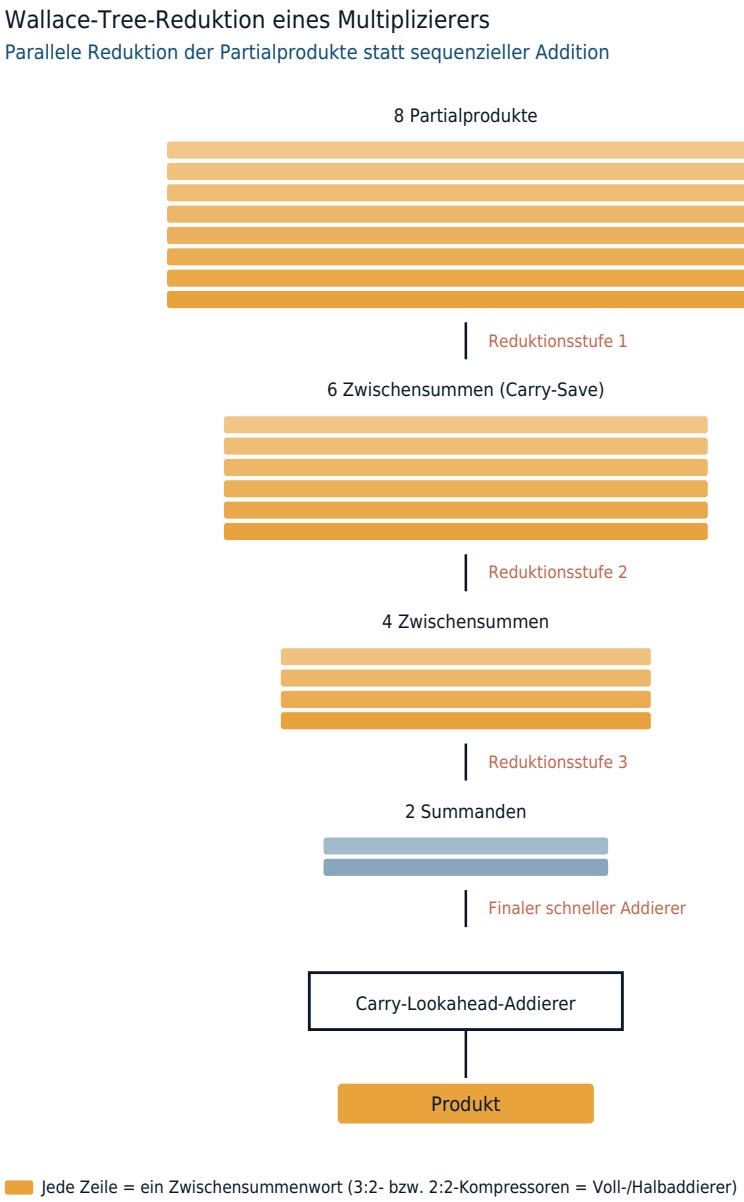


Der Carry-Lookahead-Addierer löst dieses Problem, indem er für jede Bitstelle i im Voraus zwei Hilfssignale berechnet: Generate $G_i = A_i$ AND B_i (an dieser Stelle entsteht immer ein Übertrag, unabhängig vom Eingangsübertrag) und Propagate $P_i = A_i$ XOR B_i (ein eingehender Übertrag würde durchgereicht). Daraus lässt sich jeder Übertrag direkt als Summe von Produkttermen ausdrücken, etwa $C_1 = G_0 + P_0C_0$, $C_2 = G_1 + P_1G_0 + P_1P_0C_0$, $C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$ – jeder Übertrag hängt also nur noch von den Eingangsbits

selbst ab, nicht mehr vom sequenziell berechneten Vorgänger-Übertrag, und lässt sich mit einer festen, kleinen Gatterlaufzeit parallel für alle Stellen berechnen. Da die Anzahl der UND-Eingänge (Fan-in) mit jeder weiteren Bitstelle wächst, werden reale Lookahead-Addierer meist in Blöcken von 4 Bit aufgebaut und die Blocküberträge auf einer zweiten Lookahead-Ebene kombiniert, um den Fan-in begrenzt zu halten – ein Kompromiss, der die logarithmische statt konstante Verzögerung erklärt, aber auf Kosten eines deutlich höheren Flächen- und Verdrahtungsaufwands, wie er in vielen Bereichen des digitalen Schaltungsentwurfs auftritt.

Multiplizierer: vom Array- zum Wallace-Tree

Die einfachste Multiplizierer-Architektur ist der Array-Multiplizierer, der die klassische schriftliche Multiplikation nachbildet. Für jedes Bit des Multiplikators wird zunächst ein Partialprodukt gebildet: Bit i des Multiplikators UND-verknüpft mit jedem Bit des Multiplikanden, um i Stellen nach links verschoben (genau wie beim Kopfrechnen). Bei zwei 4-Bit-Operanden entstehen so vier Partialprodukt-Zeilen. Diese werden zeilenweise addiert: Eine Reihe von Volladdierern summiert Partialprodukt 1 und 2 spaltenweise und erzeugt dabei eine Summen- und eine Übertragszeile; die Übertragszeile wird um eine Stelle nach links verschoben und zusammen mit der Summenzeile zur nächsten Partialprodukt-Zeile addiert, und so weiter treppenartig durch alle Zeilen. Da jede Zeilenaddition erst abgeschlossen sein muss, bevor die nächste beginnen kann, dominiert diese sequenzielle Verkettung die Verzögerung, die näherungsweise linear mit der Bitbreite wächst.



Schnellere Multiplizierer nutzen stattdessen einen Wallace-Tree oder Dadda-Tree: Anstatt zeilenweise nacheinander zu addieren, werden in jeder Spalte gleichzeitig jeweils drei Partialproduktbits – unabhängig davon, aus welcher ursprünglichen Zeile sie stammen – von einem Volladdierer als 3:2-Kompressor zu zwei Ausgangsbits reduziert (bei nur zwei verbleibenden Bits übernimmt ein Halbaddierer als 2:2-Kompressor diese Aufgabe). Da alle Spalten parallel in derselben Reduktionsstufe verarbeitet werden, statt auf das Ergebnis der vorherigen Zeile zu warten, sinkt die Anzahl der Zeilen von Stufe zu Stufe etwa im Verhältnis 3:2, bis nur noch zwei Summanden übrig bleiben, die abschließend mit einem einzigen schnellen Addierer (oft ein Carry-Lookahead-Addierer) verrechnet werden. In Kombination mit Booth-Kodierung, die die

Anzahl der benötigten Partialprodukte durch geschicktes Zusammenfassen mehrerer Multiplikator-Bits nahezu halbiert, sind Wallace-Tree-Multiplizierer die Standardarchitektur in modernen Prozessoren, digitalen Signalprozessoren und Fließkommaeinheiten.