Wyłączenie bramki znane jest jako *hi-Z* lub *stan wysokiej impedancji*. *Z* jest symbolem oznaczającym *impedancję*, skomplikowaną matematycznie wersję oporu. Można wyobrazić sobie bramkę trójstanową jako obwód z rysunku 2.35. Niezależne sterowanie każdą z baz daje nam cztery kombinacje: 0, 1, hi-Z i stopienie. Oczywiście projektanci obwodu muszą zadbać o to, by kombinacja powodująca jego stopienie nie mogła zostać wybrana.

Bramki trójstanowe pozwalają na łączenie dużej liczby urządzeń razem. Jedynym zastrzeżeniem jest to, że tylko jedno z tych urządzeń może być w danym czasie włączone.

Budowa bardziej skomplikowanych obwodów

Wprowadzenie bramek logicznych znacznie uprościło proces projektowania sprzętu komputerowego. Odtąd nie trzeba już składać każdej części od początku z oddzielnych elementów. Na przykład do budowy dwuwejściowej bramki NAND potrzeba było około 10 elementów. 7400 zawierał cztery takie bramki w jednej części, w układzie *małej skali integracji* (*small scale integration*, SSI). Tak więc jeden taki układ zastępował 40 części.

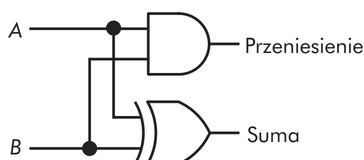
Projektanci sprzętu mogli budować systemy z układów SSI tak samo, jak wcześniej budowali je z użyciem oddzielnych komponentów. Sprzęt dzięki temu stał się tańszy i bardziej zwarty. A ponieważ pewne kombinacje układów SSI były używane częściej niż inne, wprowadzono układy *średniej skali integracji* (*medium scale integration*, MSI), w których te najczęstsze kombinacje były już wbudowane. Jeszcze bardziej zmniejszyło to liczbę części, którą trzeba było składać ze sobą, by uzyskać gotowy system. Jeszcze później wprowadzono układy *dużej skali integracji* (LSI), *bardzo dużej skali integracji* (VLSI) i tak dalej.

W punktach poniżej poznamy niektóre z kombinacji bramek, jednak to nie wszystko. Zobaczymy, jak można połączyć takie klocki budowlane wyższego poziomu, by uzyskać komponenty jeszcze wyższego poziomu. W podobny sposób bardziej złożone programy komputerowe powstają z kombinacji prostszych programów.

Budowa sumatora

Zbudujemy sumator uzupełnienia do dwóch. Być może nikt z was nigdy nie będzie naprawdę musiał budować czegoś takiego, ale ten przykład pokaże, w jaki sposób dobrze dobrane przekształcenia logiczne mogą znacznie poprawić wydajność. Reguła ta ma zastosowanie zarówno w dziedzinie sprzętu, jak i oprogramowania.

W rozdziale 1 przekonaliśmy się, że suma dwóch bitów jest równoważna ich XOR, przeniesienie zaś dwóch bitów wynosi tyle co ich AND. Rysunek 2.39 przedstawia realizację bramkową tego spostrzeżenia.



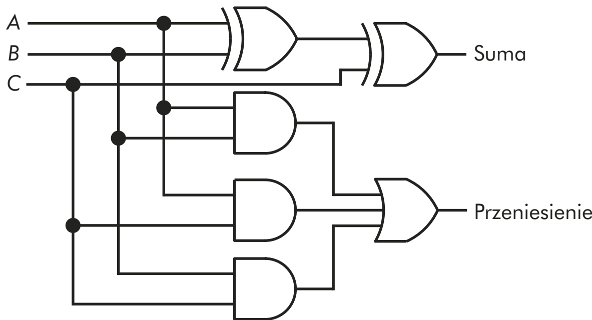
Rysunek 2.39. Sumator półpełny

Widzimy, że bramka XOR podaje sumę, a bramka AND podaje przeniesienia. Urządzenie, którego schemat widzimy na rysunku 2.39, nazywamy *sumatorem półpełnym*. „Półpełnym”, ponieważ czegoś tu brakuje. Wszystko działa dobrze, dopóki dodajemy dwa bity, ale potrzeba nam trzeciego wejścia, jeśli chcemy sygnalizować przeniesienie. To oznacza, że potrzebujemy dwóch sumatorów półpełnych, by uzyskać sumę dla każdego bitu. Przeniesienie powstaje, gdy przynajmniej dwa z wejść wynoszą 1. Tabela 2.1 zawiera tabelę prawdy dla tak powstałego *sumatora pełnego*.

Tabela 2.1. Tabela prawdy sumatora pełnego

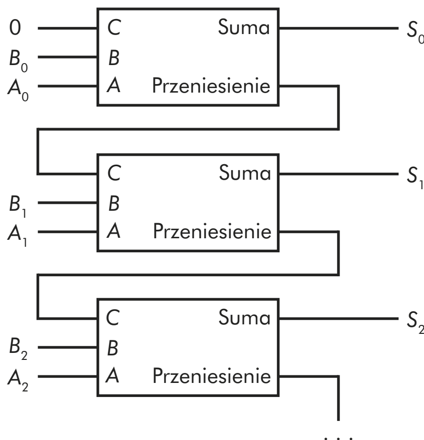
A	B	C	Suma	Przeniesienie
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sumator pełny jest nieco bardziej skomplikowany w budowie, a jego schemat widzimy na rysunku 2.40.



Rysunek 2.40. Sumator pełny

Jak widać, potrzeba na to nieco więcej bramek. Teraz, gdy już dysponujemy sumatorem pełnym, możemy użyć go do budowy sumatora dla więcej niż jednego bitu. Rysunek 2.41 pokazuje nam konfigurację nazywaną sumatorem *z przeniesieniami szeregowymi* (*ripple-carry adder*).



Rysunek 2.41. Sumator z przeniesieniami szeregowymi

Angielska nazwa tego typu sumatora (*ripples-carry adder*) pochodzi od fal na wodzie (*ripples*), które powstają, gdy ktoś np. rzuci kamieniem w sadzawkę. Odzwierciedla ona sposób, w jaki przeniesienie wędruje od jednego bitu do następnego – niczym właśnie fala na wodzie. To działa bardzo dobrze, ale widać, że powstają w ten sposób dwa opóźnienia na bramce na każdy bit. Takie opóźnienia kumulują się bardzo szybko, jeśli mamy zbudować 32- czy 64-bitowy sumator. Opóźnienia takie możemy wyeliminować za pomocą sumatora *z przeniesieniami równoległymi* (*carry look-ahead adder*). Prosta arytmetyka wskaże nam, jak zrobić taki sumator.

Na rysunku 2.40 widać, że przeniesienie z sumatora pełnego dla bitu i idzie do bitu $i+1$ jako wejście:

$$C_{i+1}=(A_i \text{ AND } B_i) \text{ OR } (A_i \text{ AND } C_i) \text{ OR } (B_i \text{ AND } C_i)$$

Ważną tutaj kwestią jest to, że potrzebujemy C_i do obliczenia C_{i+1}, co powoduje przesunięcie „fali". Widać to dobrze w następującym równaniu na C_{i+2}:

$$C_{i+2} = (A_{i+1} \text{ AND } B_{i+1}) \text{ OR } (A_{i+1} \text{ AND } C_{i+1}) \text{ OR } (B_{i+1} \text{ AND } C_{i+1})$$

Zależność tę możemy wyeliminować przez podstawienie w drugim równaniu wyrażenia C_{i+1} według pierwszego równania. Otrzymujemy co następuje:

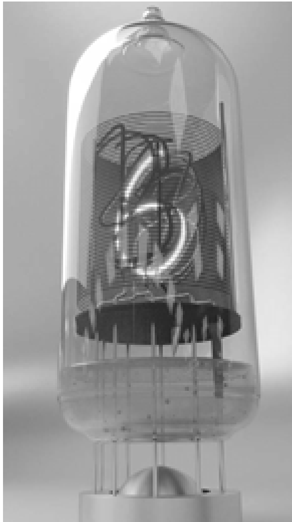
$$\begin{aligned} C_{i+2} = {} & (A_{i+1} \text{ AND } B_{i+1}) \\ & \text{OR } (A_{i+1} \text{ AND } ((A_{i+1} \text{ AND } B_{i+1}) \text{ OR } (A_{i+1} \text{ AND } C_{i+1}) \text{ OR } (B_{i+1} \text{ AND } C_{i+1}))) \\ & \text{OR } (B_{i+1} \text{ AND } ((A_{i+1} \text{ AND } B_{i+1}) \text{ OR } (A_{i+1} \text{ AND } C_{i+1}) \text{ OR } (B_{i+1} \text{ AND } C_{i+1}))) \end{aligned}$$

Zauważmy, że chociaż o wiele więcej w tym bramek AND i OR, to nadal czas propagacji całości równa się czasowi propagacji dwóch bramek. C_n zależy tylko od wartości wejść A i B, dzięki czemu czas przeniesienia, a więc i czas całego dodawania nie zależy już od liczby bitów. C_n zawsze może być wygenerowane z C_{n-1}, na co z kolei wraz ze wzrostem liczby dodawanych bitów

zużywamy coraz więcej bramek. Chociaż bramki są tanie, to jednak zużywają energię, jest to więc kompromis między szybkością a zużyciem energii.

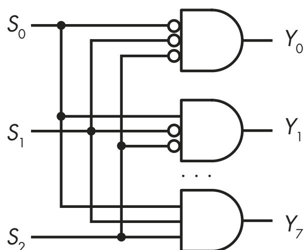
Budowa dekoderów

W punkcie „Przedstawianie liczb całkowitych za pomocą bitów” (s. 6), budowaliśmy albo *kodowaliśmy* liczby z użyciem bitów. *Dekoder* wykonuje operację odwrotną, tzn. zmienia zakodowaną liczbę z powrotem na zestaw poszczególnych bitów. Jednym z zastosowań dekoderów jest sterowanie wyświetlaczami. Ktoś z was mógł się zetknąć wcześniej z *lampą cyfrową* lub *digitronem* (*nixie tube*) – taką jak pokazana na rysunku 2.42. Spotyka się je zwłaszcza w starych filmach science-fiction; pozwalają na naprawdę fajne wyświetlanie cyfr. W zasadzie są one po prostu zestawem neonowych znaków, po jednym dla każdej cyfry. Każdy świecący drucik ma swoje własne połączenie, co zmusza nas do zamiany 4-bitowych liczb na 10 oddzielnych wyjść.



Rysunek 2.42. Digitron

Przypomnijmy sobie, że zapis ósemkowy bierze osiem różnych wartości i koduje je w trzech bitach. Rysunek 2.43 pokazuje nam taki dekoder 3:8, który konwertuje wartość ósemkową z powrotem na zestaw pojedynczych bitów.



Rysunek 2.43. Dekoder 3:8