Politechnika Wrocławska

Wydział Informatyki
i Telekomunikacji

Organizacja i architektura komputerów: projekt

Konwerter z nadmiarowej reprezentacji binarnej na standardową

Konrad Florczak 281110
Franciszek Wojnowski 281067

Prowadzący:
Dr hab. inż. S. Piestrak, prof. PWr

Wrocław 2025

Spis treści

1 Opis projektu **3**

2 Reprezentacje binarne stosowane w konwersji **3**

3 Podstawowa metoda **4**

4 Metoda A (tryb szeregowy) **6**

5 Metoda B (z antycypacją przeniesień) **9**

6 Wnioski **11**

Spis rysunków

1 Schemat układu wykorzystującego metodę podstawową 5
2 Schemat sumatora pełnego 6
3 Schemat sumatora pełnego dla LSB 6
4 Schemat układu wykorzystującego metodę szeregową 8
5 Schemat konwertera szeregowego dla pojedyńczego bitu 9
6 Schemat konwertera szeregowego dla LSB 9
7 Schemat konwertera CLA dla pojedyńczego bitu 10
8 Schemat konwertera CLA dla LSB 11

1 Opis projektu

Celem projektu jest analiza działania i porównanie efektywności na poziome układów logicznych trzech algorytmów konwersji: metody tradycyjnej oraz dwóch nowych podejść – metody A (tryb szeregowy) i metody B (z antycypacją przeniesień) – opisanych w publikacji [1]. W ramach projektu omówione zostaną:

- struktura i zasady działania każdej z metod konwersji,
- wymagana liczba bramek NAND dla każdej metody w zależności od długości wektora bitów.

Ponadto dokonano implementacji przedstawionych metod w języku C++.

2 Reprezentacje binarne stosowane w konwersji

W jednostkach arytmetycznych procesorów powszechnie stosuje się dwie następujące reprezentacje liczb binarnych:

- redundantna reprezentacja binarna ze zbioru cyfr $\{-1, 0, 1\}$ (oznaczaną dalej jako SD2), dla której wektor bitów o długości n:

  $$X_{\mathrm{sd2}} = [X_{n-1}, X_{n-2}, \ldots, X_0], \quad X_i \in \{-1, 0, 1\},$$

  ma wartość

  $$X_{\mathrm{sd2}} = \sum_{i=0}^{n-1} X_i \, 2^i.$$

  Reprezentacja ta nie wymaga propagacji przeniesienia przy dodawaniu, co znacząco przyspiesza realizację operacji arytmetycznych w układach scalonych.

- reprezentacja w kodzie *dwójkowym uzupełnieniowym* (oznaczaną dalej jako U2), w której (n+1)-bitowe słowo

  $$Y_{\mathrm{U2}} = [Y_n, Y_{n-1}, \ldots, Y_0], \quad Y_i \in \{0, 1\},$$

  odpowiada wartości

  $$Y_{\mathrm{U2}} = -\,Y_n \, 2^n \; + \; \sum_{i=0}^{n-1} Y_i \, 2^i.$$

  Jest to standardowy sposób kodowania liczb ze znakiem w cyfrowych jednostkach arytmetycznych.

Reprezentacja SD2 umożliwia szybkie wykonywanie dodawania bez kaskadowego przenoszenia, co bywa wykorzystywane w układach wykonujących szybko operacje arytmetyczne. Po zakończeniu obliczeń w formacie nadmiarowym wynik należy przekształcić na format U2, aby był zgodny z pozostałymi elementami układu cyfrowego.

Natomiast reprezentacja U2 jest szybsza przy operacjach logicznych oraz porównywaniu, ze względu na fakt, iż na zakodowanie znaku w reprezentacji SD2 potrzeba dwóch pozycji, co podwaja czas wykonywania tych operacji w porównaniu z U2.

Ze względu na fakt, że komputer operuje w systemie binarnym i rozumie jedynie zera oraz jedynki, a system SD2 wykorzystuje trzy wartości: -1, 0 i 1, konieczne było wprowadzenie specjalnego kodowania tych wartości na poziomie układów logicznych, aby umożliwić wykorzystanie tej reprezentacji w układach cyfrowych. W tym celu każda cyfra systemu SD2 jest reprezentowana za pomocą dwóch bitów: S (znak) oraz D (cyfra). Używane kodowanie w omawianej pracy wygląda następująco:

Tabela 1: Kodowanie cyfr systemu SD2 za pomocą bitów

Cyfra w SD2	**S (bit znaku)**	**D (bit cyfry)**
-1	1	1
0	0	0
1	0	1

3 Podstawowa metoda

Podstawowy sposób konwersji reprezentacji SD2 na U2 polega na rozdzieleniu bitów na część dodatnią i ujemną, a następnie obliczeniu różnicy pomiędzy tymi dwiema liczbami binarnymi.

Liczbę $\mathbf{X}_{SD2}$, składającą się z n-pozycji, rozdzielamy na składowe $\mathbf{X}^{+}_{SD2}$ oraz $\mathbf{X}^{-}_{SD2}$, gdzie:

$$\mathbf{X}^{+}_{SD2} = \sum_{i=0}^{n-1} x_i^{+}\, 2^i, \qquad x_i^{+} = \begin{cases} 1 & \text{gdy } x_i = 1, \\ 0 & \text{wpp.} \end{cases} \tag{1}$$

$$\mathbf{X}^{-}_{SD2} = \sum_{i=0}^{n-1} x_i^{-}\, 2^i, \qquad x_i^{-} = \begin{cases} 1 & \text{gdy } x_i = -1, \\ 0 & \text{wpp.} \end{cases} \tag{2}$$

Następnie odejmujemy te nowo powstałe liczby, aby otrzymać przekonwertowaną liczbę X_{SD2}

$$Y_{U2} = \mathbf{X}^{+}_{SD2} - \mathbf{X}^{-}_{SD2}, \tag{3}$$

które można zinterpretować jako dodawanie w kodzie U2: *dodajemy* do $\mathbf{X}^{+}_{SD2}$ *dopełnienie do dwóch* liczby $\mathbf{X}^{-}_{SD2}$ wraz z przeniesieniem początkowym równym 1.

Algorytm krok po kroku

1. Z liczby w redundantnej reprezentacji binarnej o n-długości wyodrębnij maski bitowe x_i^+ oraz x_i^-.
2. Zbuduj dwa n–bitowe słowa: X^+ (z bitami x_i^+) oraz X^- (z bitami x_i^-).
3. Oblicz $\overline{X^-} + 1$, czyli uzupełnienie liczby X^-.
4. Dodaj wynik z kroku 3 do X^+ za pomocą klasycznego dodawania binarnego. Wynik Y_2 jest reprezentacją U2.
5. Jeżeli w najstarszym bicie powstała kolizja znaku (tzn. wartość jest sprzeczna z oczekiwanym znakiem), roz-szerz słowo o jeden bit tak, aby zachować poprawność kodu U2.

Przykład 3.1. *Dla ciągu*

$$\mathbf{X}_{SD2} = [1\ 0\ 0\ 1\ 0\ \overline{1}\ \overline{1}\ 0\ 1\ \overline{1}\ 1]_{SD2}$$

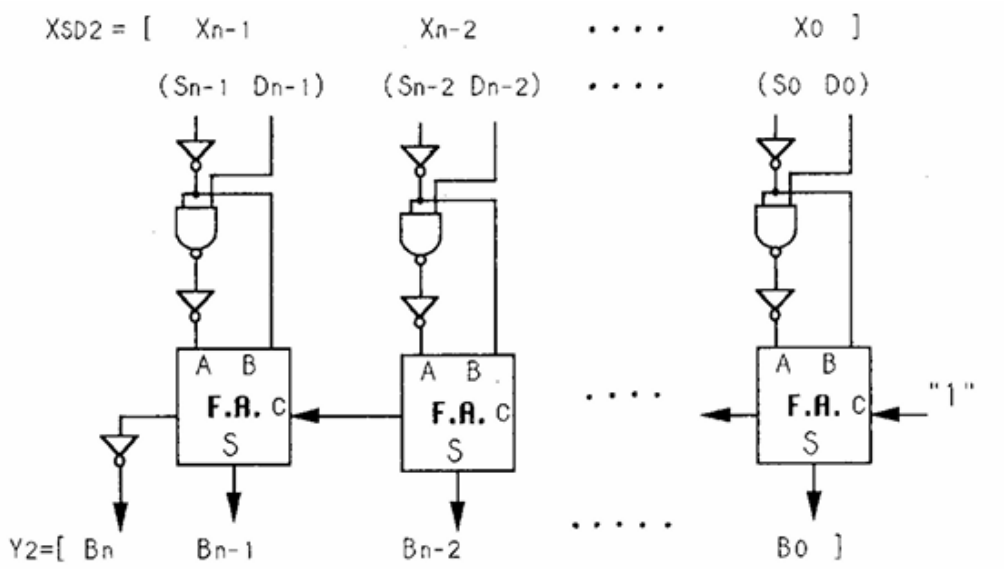
otrzymujemy

$$X^+ = [1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1]_{U2}, \qquad X^- = [0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0]_{U2}.$$

Po wykonaniu kroków 3-4 dostajemy

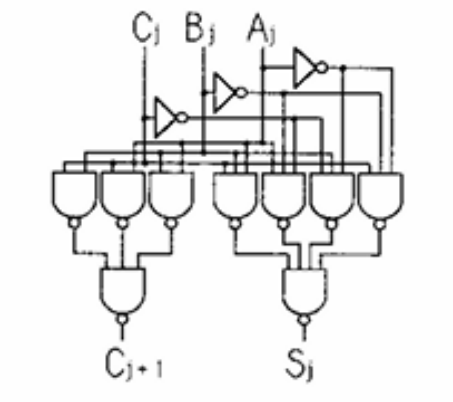
$$Y_{U2} = [0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1]_{U2},$$

Konwerter prezentuje się następująco:



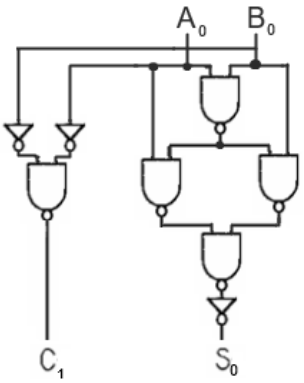
Rysunek 1: Schemat układu wykorzystującego metodę podstawową

gdzie sumator pełny (F.A.) ma postać:



Rysunek 2: Schemat sumatora pełnego

Ze względu na fakt, że C_0 ma zawsze wartość równą 1, możliwe jest uproszczenie sumatora wyznaczającego najmniej znaczący bit (LSB) do postaci:



Rysunek 3: Schemat sumatora pełnego dla LSB

4 Metoda A (tryb szeregowy)

Zaproponowana zoptymalizowana wersja konwertera w trybie szeregowym w celu zwiększenia efektywności konwersji liczb z SD2 do U2. W przeciwieństwie do metody podstawowej, która wymaga klasycznego odejmowania liczb binarnych, konwerter szeregowy bazuje na logicznym śledzeniu obecności cyfr ujemnych i dodatnich w liczbie w SD2, przy pomocy pomocniczej zmiennej sterującej. Dla każdej pozycji i w słowie SD2 definiujemy wartość logiczną C_i, która określa, czy dotychczas napotkano wartość -1 bez późniejszego wystąpienia $+1$ na prawo od tej pozycji. Zmienna ta steruje wartością bitu wyjściowego w reprezentacji binarnej U2.

Definicja zmiennej C_i:
$C_i = 1$, jeśli istnieje co najmniej jedno -1 po prawej stronie i nie ma żadnego $+1$ między tą pozycją a napotkanym -1.
$C_i = 0$ w przeciwnym przypadku.

Na tej podstawie wyznaczany jest bit wyjściowy B_i według zależności:

$$B_i = D_i \oplus C_i \tag{4}$$

oraz propagowana jest zmienna C_i do kolejnego etapu:

$$C_{i+1} = S_i + D_i \cdot C_i, \quad C_0 = 0 \tag{5}$$

gdzie:

- D_i i S_i to zakodowane bity cyfry SD2 (cyfra i znak),
- $+$ oznacza operację logiczną OR,
- $\cdot$ oznacza operację AND,
- $\oplus$ oznacza XOR.

Algorytm krok po kroku

1. Zakoduj każdą cyfrę RB jako parę bitów (S_i, D_i), np.:
   - $1 \to (0, 1)$
   - $0 \to (0, 0)$
   - $-1 \to (1, 1)$
2. Inicjalizuj zmienną $C_0 = 0$.
3. Dla każdego bitu od najmniej do najbardziej znaczącego wykonaj:
   - Wyznacz $B_i = D_i \oplus C_i$
   - Oblicz $C_{i+1} = S_i + D_i \cdot C_i$
4. Po przejściu przez wszystkie pozycje uzyskujemy wynik $Y_{U2} = [B_n, \ldots, B_0]$.

Przykład 4.1. *Dla ciągu*

$$X_{SD2} = [1\ 0\ 0\ 1\ 0\ 1\ \overline{1}\ 0\ 1\ \overline{1}\ 0\ 1]_{SD2}$$

Najpierw kodujemy każdą cyfrę za pomocą pary bitów (S_i, D_i):

$$(0,1),\ (0,0),\ (0,0),\ (0,1),\ (0,0),\ (0,1),\ (1,1),\ (0,0),\ (0,1),\ (1,1),\ (0,0),\ (0,1)$$

Inicjalizujemy zmienną pomocniczą $C_0 = 0$, a następnie propagujemy ją przez kolejne etapy według wzoru:

$$C_{i+1} = S_i + D_i \cdot C_i$$

Obliczamy kolejne wartości C_i:

$$C = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0]$$

Na podstawie relacji $B_i = D_i \oplus C_i$ otrzymujemy wynik konwersji binarnej:

$$Y_{U2} = [0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1]_{U2}$$

Należy zaznaczyć, że wynik zawiera dodatkowy bit rozszerzający długość słowa, co odpowiada specyfikacji konwersji z SD2 do reprezentacji binarnej U2 o długości $n + 1$.

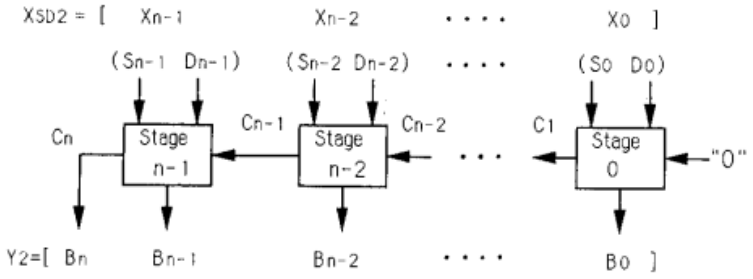
Zastosowany konwerter w trybie szeregowym realizuje funkcję logiczną w każdej pozycji i za pomocą dwóch prostych równań logicznych: XOR (dla wyjścia B_i) oraz OR i AND (dla zmiennej pomocniczej C_{i+1}). Taka konstrukcja pozwala na minimalizację liczby używanych bramek logicznych i uproszczenie układu względem klasycznego sumatora.

Zamiast pełnego sumatora binarnego (tzw. *full adder*), który byłby wymagany w metodzie podstawowej, każda pozycja konwersji może być zrealizowana za pomocą niewielkiej liczby bramek NAND. Autorzy wykazali, że taki konwerter:

- wymaga mniejszej liczby bramek logicznych,
- generuje mniejsze opóźnienia sygnału (tylko dwa poziomy bramek dla każdej pozycji),
- zajmuje mniejszy obszar w układzie scalonym.

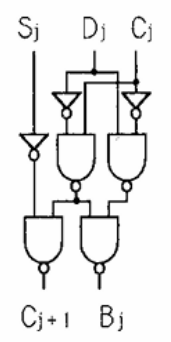
W rezultacie konwerter szeregowy stanowi wydajną i prostą alternatywę sprzętową dla klasycznego podejścia opartego na odejmowaniu binarnym.

Zaproponowany konwerter z $(n-1)$-bitowej liczby w $SD2$ na n-bitową liczbę w $U2$:



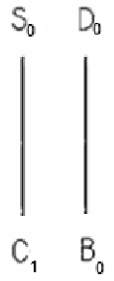
Rysunek 4: Schemat układu wykorzystującego metodę szeregową

gdzie bloki $Stage_{n-1}$ do $Stage_1$ stanowią układ konwertujący pojedyńczy znak w postaci:



Rysunek 5: Schemat konwertera szeregowego dla pojedyńczego bitu

Natomiast $Stage_0$ stanowi szczególny przypadek w układzie ze względu na C_0, które ma stałą wartość równą 0, co pozwala na uproszczenie do postaci:



Rysunek 6: Schemat konwertera szeregowego dla LSB

5 Metoda B (z antycypacją przeniesień)

Zoptymalizowany konwerter z reprezentacji SD2 na U2 może działać poprzez antycypację przeniesień (CLA), co znacząco redukuje czas propagacji sygnału „pożyczki" (analogicznie do propagacji przeniesienia w sumatorach lookahead). Metoda ta opiera się na wykorzystaniu sygnałów propagacji P_i oraz sumowania S_i, które umożliwiają wyprzedzające (lookahead) obliczenie zmiennej C_i dla każdej pozycji.

W przeciwieństwie do metody szeregowej, w której zmienna pomocnicza C_i propagowana jest po kolei od najmniej znaczącego bitu, tutaj wykorzystujemy strukturę blokową podobną do sumatorów typu carry lookahead.

Dla każdej pozycji i:

- definiujemy $P_i = \overline{D_i}$ (sygnał propagacji),
- zmienna pomocnicza C_i jest wyznaczana równolegle dla każdego bitu.

Pierwsze cztery wartości sygnału przeniesienia (carry) są następujące:

$$\begin{aligned} C_0 &= 0 \\ C_1 &= S_0 + P_0 C_0 \\ C_2 &= S_1 + P_1 S_0 + P_1 P_0 C_0 \\ C_3 &= S_2 + P_2 S_1 + P_2 P_1 S_0 + P_2 P_1 P_0 C_0 \end{aligned} \tag{6}$$

Dla bloku czterech bitów zmienna C_4 może być obliczona na podstawie sygnałów propagacji i generacji w następujący sposób:

$$C_4 = S'_0 + P'_0 C_0 \tag{7}$$

gdzie:

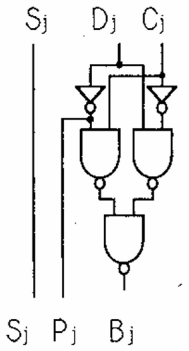
$$S'_0 = S_3 + P_3 S_2 + P_3 P_2 S_1 + P_3 P_2 P_1 S_0 \tag{8}$$

$$P'_0 = P_3 P_2 P_1 P_0 \tag{9}$$

W ujęciu ogólnym, dla kolejnych bloków 4-bitowych:

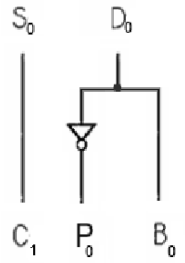
$$C_{4i+4} = S'_i + P'_i C_{4i}, \quad i = 0,\ 1,\ 2,\ \ldots \tag{10}$$

Konwerter CLA znacząco poprawia czas działania dzięki równoległemu obliczaniu zmiennych C_i w blokach, co pozwala uniknąć sekwencyjnego propagowania sygnału. Układ wygląda jak na rysunku (6) z tą różnicą, że same bloki $Stage_{n-1}$ do $Stage_1$ mają postać:



Rysunek 7: Schemat konwertera CLA dla pojedyńczego bitu

Podobnie jak w metodzie szeregowej i podstawowej, także w tym przypadku możliwe jest uproszczenie układu konwertującego LSB:



Rysunek 8: Schemat konwertera CLA dla LSB

6 Wnioski

Na podstawie implementacji i analizy trzech metod konwersji SD2 na U2 można stwierdzić, że zaawansowane podejścia znacząco poprawiają efektywność układów. Metoda szeregowa pozwala zredukować liczbę bramek NAND w przybliżeniu o 60% względem metody podstawowej, wykorzystując prostą propagację zmiennej sterującej. Metoda CLA idzie o krok dalej – równoległe wyznaczanie przeniesień umożliwia redukcję aż o około 70% przy jednoczesnym zwiększeniu szybkości działania.

W kontekście praktycznym metoda CLA wypada najlepiej – oferuje zarówno najwyższą wydajność czasową, jak i najmniejszą liczbę wykorzystywanych bramek NAND. Metoda szeregowa pozostaje dobrą alternatywą tam, gdzie kluczowa jest prostota implementacji. Klasyczne podejście, mimo swojej intuicyjności, ustępuje obu pozostałym pod względem złożoności i opłacalności.

Dodatkowo projekt ten nauczył nas krytycznego podejścia do analizy literatury naukowej. W pracy źródłowej natrafiliśmy na istotne niespójności – przykładowo, jedna z zaprezentowanych tabel (porównująca liczbę bramek i opóźnień czasowych) zawierała dane, których nie dało się zweryfikować na podstawie pozostałej części publikacji. Wskazuje to na potrzebę ostrożności przy bezpośrednim wykorzystywaniu cudzych wyników i podkreśla znaczenie samodzielnej weryfikacji oraz interpretacji prezentowanych treści.

Literatura

[1] Sung-Ming Yen, Chi-Sung Laih, Chin-Hsing Chen, and Jau-Yien Lee, “An Efficient Redundant-Binary Number to Binary Number Converter,” *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 109–112, Jan. 1992.