# Symulator układów kombinacyjnych

Błażej Łuczak

28.01.2026

# Treść zadania:

“Należy napisać program, który na podstawie opisu połączeń bramek logicznych (NAND, NOR, AND, OR etc.) dla zadanego pliku z wartościami na wejściach generuje plik opisujący stany wyjściowe. Dodatkową wyznaczaną informacją jest czas propagacji sygnałów".

Program, na podstawie zewnętrznych plików wejściowych, które zawierają opis struktury układu oraz wartości sygnałów wejściowych, ma generować plik wynikowy prezentujący stany wybranych wyjść oraz odpowiedni dla nich czas propagacji.

# Format danych wejściowych

## Plik opisujący układ logiczny:

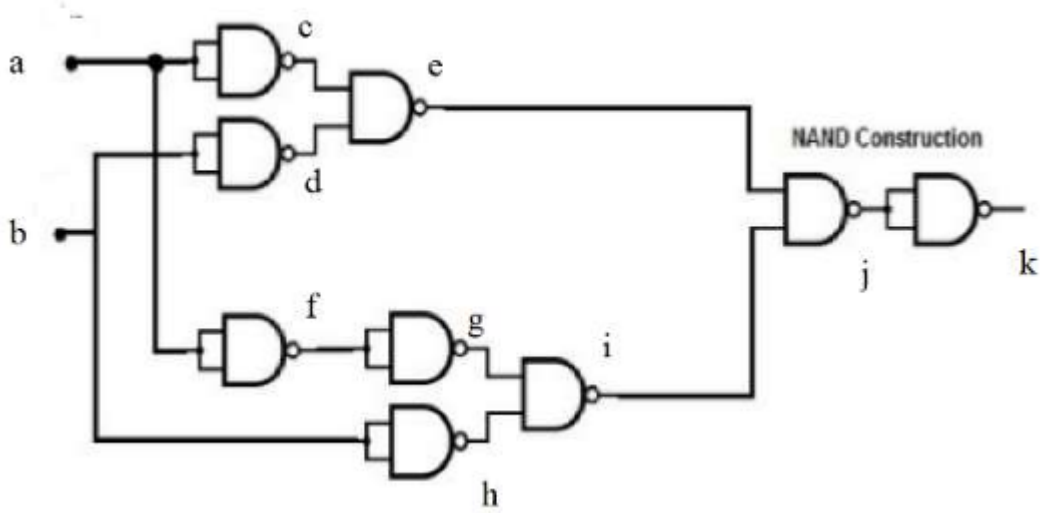
Układ jest definiowany poprzez listę bramek logicznych.
Każda bramka w pliku jest zapisana w formie:
**<typ_bramki>** <**wejście1> <wejście2> <wyjście> <czas_propagacji>**

| Pole | Typ | Opis | Przykład |
|---|---|---|---|
| **typ_bramki** | string | Rodzaj bramki logicznej | NAND |
| **wejście1**<br>**wejście2** | string | Nazwy sygnałów wejściowych | a,b |
| **wyjście** | string | Nazwa sygnału wyjściowego | c |
| **czas_propagacji** | int | Czas przejścia sygnału | 10 |

Przykład pojedynczej linii:
NAND a a c 10

Przykładowy plik:
NAND a a c 10
NAND b b d 10
NAND c d e 10
NAND a a f 10
NAND f f g 10
NAND b b h 10
NAND g h i 10
NAND e i j 10
NAND j j k 10

Opisuje układ:




Układ może zawierać wiele bramek korzystających z tych samych sygnałów oraz dowolną liczbę sygnałów pośrednich.

### Sygnały wejściowe układu:

Drugi plik wejściowy zawiera zestawy wartości logicznych przypisanych do sygnałów wejściowych układu.
Pierwsza linia określa nazwy wejść(np. a,b)
W kolejnych liniach znajdują się ich wartości dla poszczególnych symulacji.
Przykładowy plik:
a b
0 0
0 1
1 0
1 1
Dla każdej linii program przeprowadza osobną symulację działania całego układu logicznego.
Jeżeli chcemy znać wyniki tylko dla a=0 oraz b=0, plik będzie wyglądał następujaco:
a b
0 0

# Przetwarzanie danych oraz ich interpretacja

Program powinien rozpocząć działanie od wczytania pliku opisującego układ logiczny.

Każda bramka jest zapisywana jako instancja specjalnie przygotowanej klasy, która wygląda następująco:

```
//zawartosc pliku BramkaPrototyp.h bez komentarzy
class BramkaPrototyp {
public:
    std::string typ;
    std::string wejscie1;
    std::string wejscie2;
    std::string wynik;
    int czas_propagacji;
    int value;
    int czas_skumulowany;

    BramkaPrototyp(std::string typA, std::string in1,
std::string in2, std::string wynikA, int czasA);
};
```

Przechowuje:
jej typ,
nazwy sygnałów wejściowych,
nazwa sygnału wyjściowego,
przypisany czas propagacji,
bieżąca wartość wyjścia,
skumulowany czas dotarcia sygnału do wyjścia.

Posiada konstruktor.

Podczas wczytywania każda bramka jest zapisywana w wektorze jako samodzielny obiekt posiadający wszystkie potrzebne informacje do późniejszego przetwarzania:

```
  vector<BramkaPrototyp> bramki;
    string typ, in1, in2, out;
    int opoznienie;

    while (opis >> typ >> in1 >> in2 >> out >> opoznienie) {
        bramki.push_back(BramkaPrototyp(typ, in1, in2, out,
opoznienie));
    }
```

Przechowywanie bramek w ten sposób umożliwi, bądź ułatwi, iterowanie po całym układzie oraz dostęp do sygnałów pośrednich.

Następnie, wczyta i zapisze w kolejnej tablicy/wektorze drugi plik wejściowy zawierający wartości wejść.

```
    vector<pair<int,int>> zestawy;
    int aVal, bVal;
    while (wej >> aVal >> bVal) {
        zestawy.push_back({aVal,bVal});
    }
```

# Algorytm i obliczenia

Dla każdego zestawu wejść program przypisuje początkowe wejścia do odpowiednich wejść bramek i następnie wielokrotnie iteruje po liście bramek próbując obliczyć wyjścia tych bramek, których wejścia są już znane.
Bramki już policzone nie są brane pod uwagę w następnych iteracjach.

Gdy wartości obu wejść bramki są znane, program oblicza wartość wyjściową zgodnie z typem bramki i następnie ustala czas propagacji.

Iterowanie po układzie kończy się, gdy program nie może już obliczyć żadnej innej wartości bramki.

Obecna wersja programu jest więc modelem iteracyjnym. (Należy jednak zaznaczyć, że alternatywnym i bardziej wydajnym podejściem jest symulacja sekwencyjno-zdarzeniowa)

Główny algorytm z prostymi komentarzami:

```
ofstream wynikPlik("wynik.txt"); // otwarcie pliku wynikowego
    if (!wynikPlik.is_open()) {
        cout << "Nie mozna otworzyc wyniki.txt\n";
        return 1;
    }

    // petla dla kazdego zestawu a i b
    for (int z = 0; z < zestawy.size(); z++) {
        aVal = zestawy[z].first;
        bVal = zestawy[z].second;

        // resetujemy wartosci bramek
        for (auto& br : bramki) { //za kazda bramke w bramkach
            br.value = -1;
            br.czas_skumulowany = 0;
```

```
        }

        int a = aVal;
        int b = bVal;

// obliczenia: powtarzamy, az wszystkie bramki maja value
        bool zmiana = true; //w petli zmiana na true gdy
chociaz 1 bramka sie zmienila
        while (zmiana) {
            zmiana = false;
            for (auto& br : bramki) { //petla przechodzi przez
wszystkie bramki
                if (br.value != -1) continue; // jezeli value
nie jest rowne -1 to ta bramka ma juz wynik i nie trzeba jej
liczyc

                int val1 = 0, val2 = 0;
                int czas1 = 0, czas2 = 0;
                bool got1 = false, got2 = false; //czy
wartosc1 lub wartosc2 sa znane a nie jeszcze nie policzone

                // wejscie1
                if (br.wejscie1 == "a") { val1 = a; czas1 = 0;
got1 = true; }
                else if (br.wejscie1 == "b") { val1 = b; czas1
= 0; got1 = true; }
                else {
                    for (auto& prev : bramki) { //przechodzimy
znowu przez wszystkie bramki
                        if (prev.wynik == br.wejscie1 &&
prev.value != -1) { //jezeli wczesniejsza bramka ktora jest
jednym z potrzebnych sygnalow ma juz znana wartosc
                            val1 = prev.value;
                            czas1 = prev.czas_skumulowany;
                            got1 = true;
                            break;
                        }
                    }
                }

                // wejscie2 aka to samo co wejscie 1 ale dla
wejscia2
                if (br.wejscie2 == "a") { val2 = a; czas2 = 0;
got2 = true; }
```

```
                else if (br.wejscie2 == "b") { val2 = b; czas2
= 0; got2 = true; }
                else {
                    for (auto& prev : bramki) {
                        if (prev.wynik == br.wejscie2 &&
prev.value != -1) {
                            val2 = prev.value;
                            czas2 = prev.czas_skumulowany;
                            got2 = true;
                            break;
                        }
                    }
                }

if (got1 && got2) {
//jezeli znane sa wartosci obu wejsc to mozna policzyc wartosc
dla tego wejscia
                    br.value = obliczBramke(br, val1, val2);
//liczenie value
                    br.czas_skumulowany = (czas1 > czas2 ?
czas1 : czas2) + br.czas_propagacji;
                    zmiana = true;
                }
            }
        }
```

---

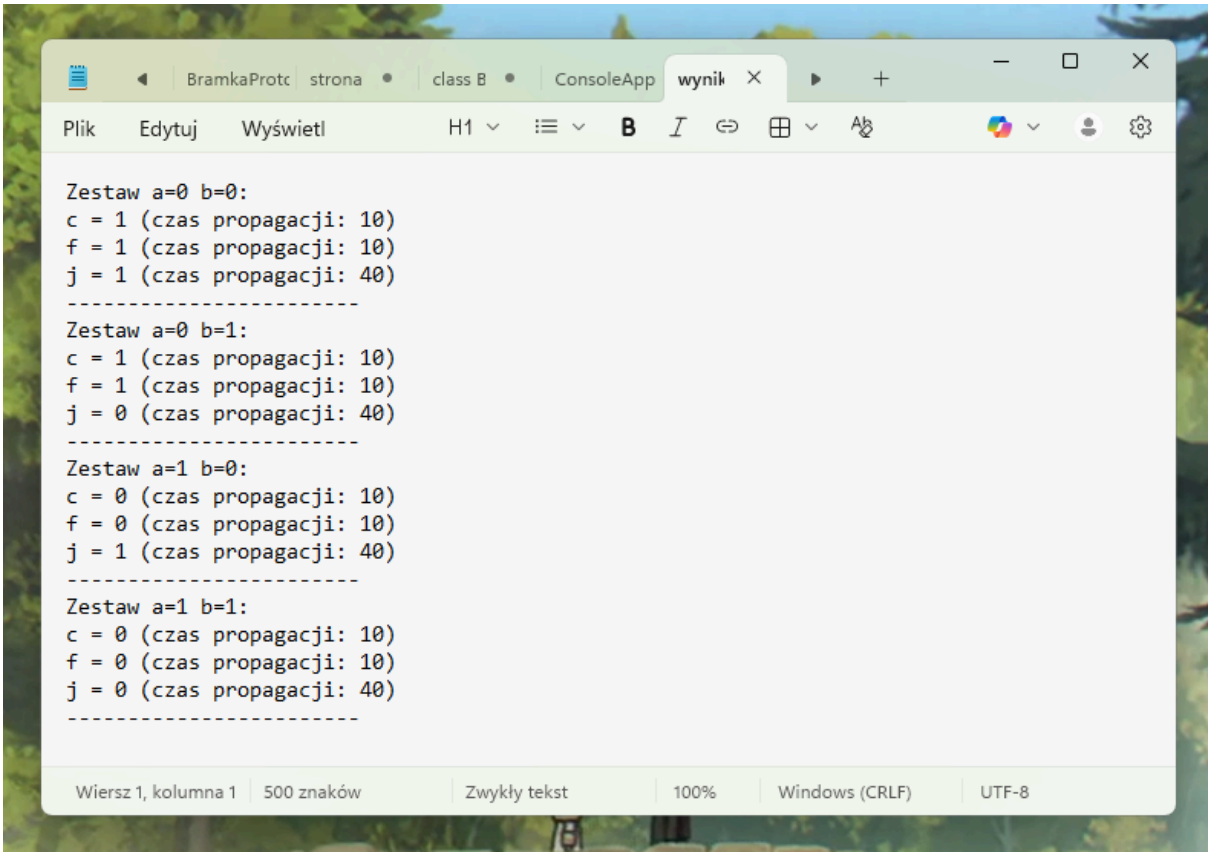
# Wynik działania programu

Dla każdego zestawu wartości wejściowych(z 2. pliku wejściowego) program powinien wyprowadzić wartości wybranych sygnałów wyjściowych oraz ich czas propagacji.
Wyniki będą zarówno wypisywane na ekran jak i zapisywane do pliku wynikowego.

# Przykład ilustrujący działanie:

```
Podaj nazwe pliku tekstowego(.txt)(bez rozszerzenia), ktory zawiera poprawnie opisany uklad kombinacyjny
opis_ukladu
Zestaw a=0 b=0:
c = 1 (czas propagacji: 10)
f = 1 (czas propagacji: 10)
j = 1 (czas propagacji: 40)
------------------------
Zestaw a=0 b=1:
c = 1 (czas propagacji: 10)
f = 1 (czas propagacji: 10)
j = 0 (czas propagacji: 40)
------------------------
Zestaw a=1 b=0:
c = 0 (czas propagacji: 10)
f = 0 (czas propagacji: 10)
j = 1 (czas propagacji: 40)
------------------------
Zestaw a=1 b=1:
c = 0 (czas propagacji: 10)
f = 0 (czas propagacji: 10)
j = 0 (czas propagacji: 40)
------------------------
```

(wypisanie obliczonych wartości dla podanego pliku opis_ukladu.txt)
(przy tym teście, zostały wypisane tylko wybrane wyjście by zająć mniej miejsca)




(wygenerowany plik wynikowy)

(Przykład został zrealizowany na podanym wcześniej pliku zawierającym opis układu kombinacyjnego na stronie 2.)

# Instrukcja obsługi

Przed uruchomieniem programu, należy utworzyć dowolnie nazwany plik tekstowy opisujący układ według wcześniej podanego standardu np.

---

opis2.txt:
NAND a b c 10
NOR b b d 10
OR c d e 10

---

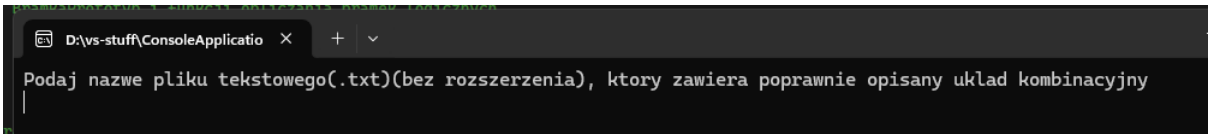
Plik ten musi znajdować się w tej samej ścieżce co reszta programu.
Tak samo, plik nazwany “wejscie.txt” musi się tam znaleźć.
Musi on wyglądać przynajmniej tak, by program działał:

---

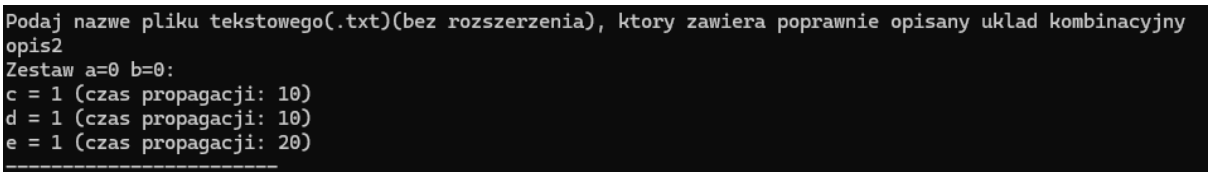
wejscie.txt:
a b
0 0

---

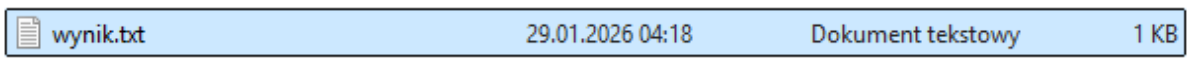
Program  będzie wtedy liczyć wyniki dla opisanych par (tutaj tylko 0,0)
Gdy oba te pliki istnieją, program powinien działać poprawnie. Można go więc uruchomić
Zapyta się on wtedy o plik, z którego ma pobrać opis układu.




Należy wtedy podać nazwę utworzonego pliku, a program policzy i wypisze wartości:




(są tu wypisane wszystkie wartości wynikowe, nie znane przed uruchomieniem programu)
Program utworzy również w tej ścieżce plik tekstowy o nazwie wynik.txt z wypisanym wynikiem.




```
Zestaw a=0 b=0:
c = 1 (czas propagacji: 10)
d = 1 (czas propagacji: 10)
e = 1 (czas propagacji: 20)
------------------------
```

# Założone cechy i ograniczenia

Układ nie zawiera sprzężeń zwrotnych
Jedynie operacje bramek wpływają na czas propagacji(kable idealne)
Kolejność bramek w pliku nie musi odpowiadać kolejności tych bramek w układzie - program sam ustali kolejność obliczeń
!!!Format danych wejściowych zakłada poprawność strukturalną(obsługa wyjątków oraz weryfikacja błędów składniowych w przygotowaniu)!!!
Każdy zestaw wartości wejściowych jest traktowany jako niezależna symulacja, wykonywana na tej samej strukturze układu podanej przez uzytkownika.
Na obecnym etapie, program zakłada korzystanie jedynie z bramek 2-wejściowych. NOT w opisie układu powinno się więc zastępować zwarciem NAND/NOR (np. NAND a a wyjście)
Program obecnie zakłada 2 wejścia do układu nazwane a i b(Zostanie to usprawnione)
Program ma być prosty do rozbudowy

# Podsumowanie, wnioski, dalszy rozwój

Program działa poprawnie.

## Możliwy dalszy rozwój, rozbudowa i/lub zmiana:

- Implementacja bramek z ilością wejść większą niż 2
- Implementacja obsługi większej ilości wejść początkowych.
- Obsługa wyjątków oraz weryfikacja błędów w składni danych wejściowych
- Obsługa stanu nieustalonego
- Wykrywanie sprzężeń
- Utworzenie biblioteki plików zawierających podstawowe układy(np. MUX) opisane w formacie danych wejściowych
- Przejście na symulację sekwencyjno-zdarzeniową