

Wydział Elektryczny
Katedra Automatyki i Robotyki

Materiały do wykładu z przedmiotu:

Układy cyfrowe i programowalne

Kod: **EMS1A1009**

Opracował: dr inż. Marian Gilewski

Białystok 2025



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Materiały przygotowane w ramach projektu „PB 5.0 – dostosowanie oferty dydaktycznej Politechniki Białostockiej do potrzeb nowoczesnej gospodarki oraz zielonej i cyfrowej transformacji” (nr umowy FERS.01.05-IP.08-0327/23-00) w ramach programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027 współfinansowanego ze środków Europejskiego Funduszu Społecznego Plus.

Publikacja jest udostępniona na licencji Creative Commons - Uznanie autorstwa 4.0 (CC BY 4.0). Pełna treść licencji dostępna na stronie creativecommons.org.

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego Politechniki Białostockiej

Program wykładu – Układy cyfrowe i programowalne, Elektromobilność, sem. 1 stud. stacjonarne I stopnia:

1. Historia układów cyfrowych oraz pojęcie sygnału cyfrowego.
2. Wybrane pozycyjne systemy liczbowe i kody cyfrowe.
3. Zaciskowe standardy sygnałowe współczesnych układów cyfrowych
4. Parametry współczesnych elementów logicznych i sekwencyjnych.
5. Klasyfikacja i wybrane architektury układów logicznych.
6. Ewolucja programowalnych struktur logicznych.
7. Elementy architektury układów FPGA .
8. Propagacja sygnału zegarowego i synteza częstotliwości w FPGA.
9. Wybrane funkcjonalności wybranej platformy projektowej CAD PLD/FPGA.
10. Elementy leksykalne języka HDL.
11. Instrukcje, podprogramy i biblioteki w VHDL.
12. Techniki definiowania systemów cyfrowych PLD – w tym do zastosowań automotive.
13. Metody syntezy kombinacyjnych i sekwencyjnych synchronicznych struktur w układach PLD.
14. Przetwarzanie współbieżne i sekwencyjne w PLD – technika FSM.
15. Zaliczenie wykładu i poprawa ocen.

Zasady zaliczania i kryteria oceny wykładu

Warunkiem uzyskania zaliczenia wykładu jest uzyskanie ponad 50% odpowiedzi pozytywnych z testu końcowego – lub 14 nieobowiązkowych testów składowych realizowanych po kolejnych wykładach.

- Student uzyskujący ocenę 3,0:
 - osiąga wszystkie efekty kształcenia wymienione w karcie przedmiotu na poziomie minimalnym (50%).
- Student uzyskujący ocenę 4,0 – oprócz efektów kształcenia na ocenę 3.0, dodatkowo:
 - osiąga efekt E2 na poziomie co najmniej 75%.
- Student uzyskujący ocenę 5.0 – oprócz efektów kształcenia na ocenę 4.0, dodatkowo:
 - osiąga efekty E2 i E3 na poziomie 90% oraz efekty „laboratoryjne” na poziomie co najmniej 60%, czyli ocenę laboratoryjną co najmniej 3,5.

Miarą procentowego spełnienia efektu jest wynik procentowy z testu w danej kategorii zadań.

Student uzyskujący ocenę 3,5 realizuje zakres wymagany 3,0 jednak mniejszy od wymagań na ocenę 4,0.

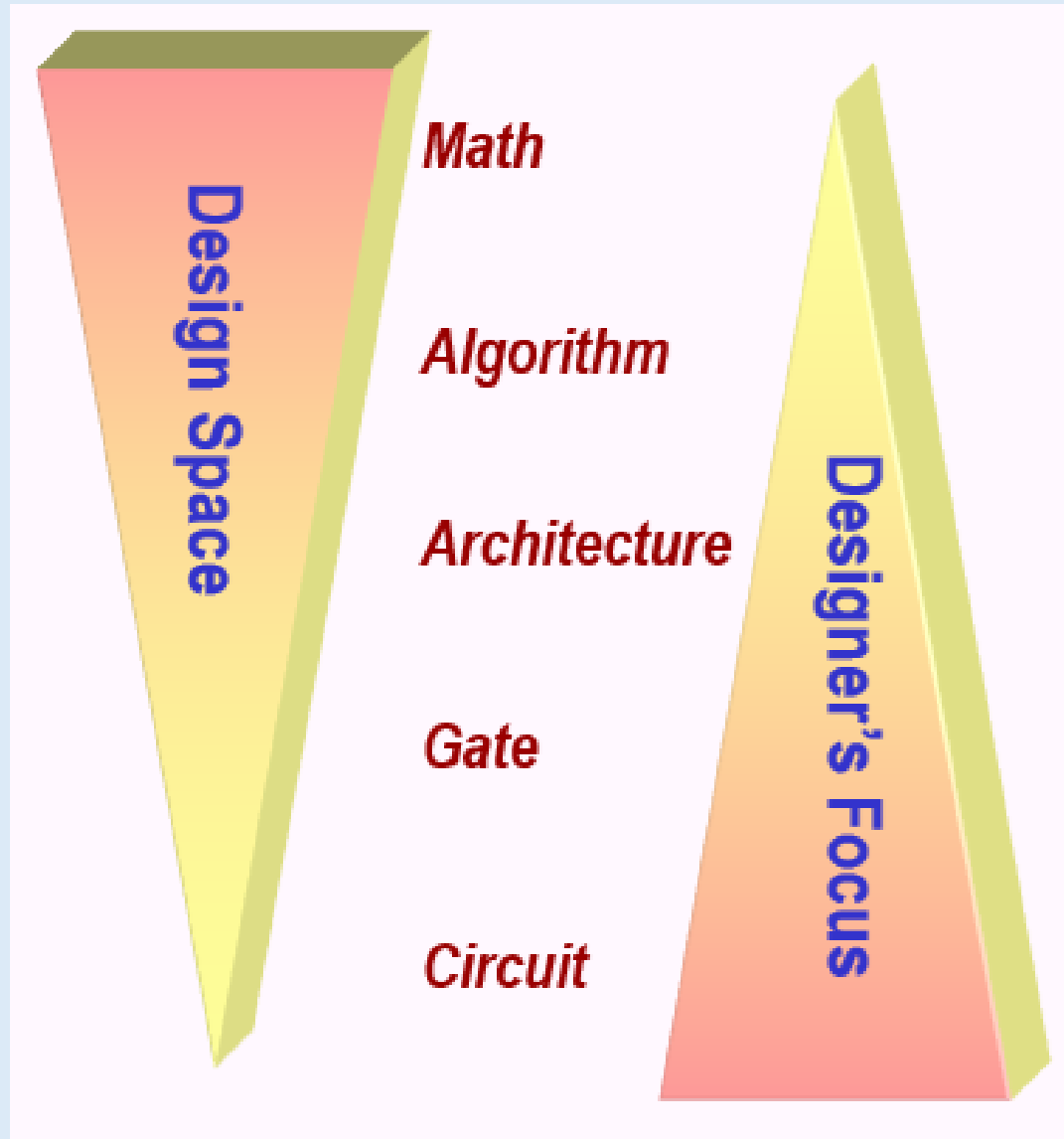
Student uzyskujący ocenę 4,5 realizuje zakres wymagany 4,0 jednak mniejszy od wymagań na ocenę 5,0.

Skala ocen: 91-100% → 5,0; 81-90% → 4,5; 71-80% → 4,0; 61-70% → 3,5; 51-60%. → 3,0 poniżej 51% → 2,0

Czym zajmują się inżynierowie elektromobilności?



Czym zajmują się inżynierowie techniki cyfrowej?



W czym specjalizują się inżynierowie techniki cyfrowej?

Formula example $DFT_8 = (F_2 \otimes I_4) \cdot D \cdot (I_2 \otimes (I_2 \otimes F_2 \cdots)) \cdot P$

A math guy



What I know:

Linear algebra
Digital signal processing
Adaptive filter theory ...

A hardware engineer

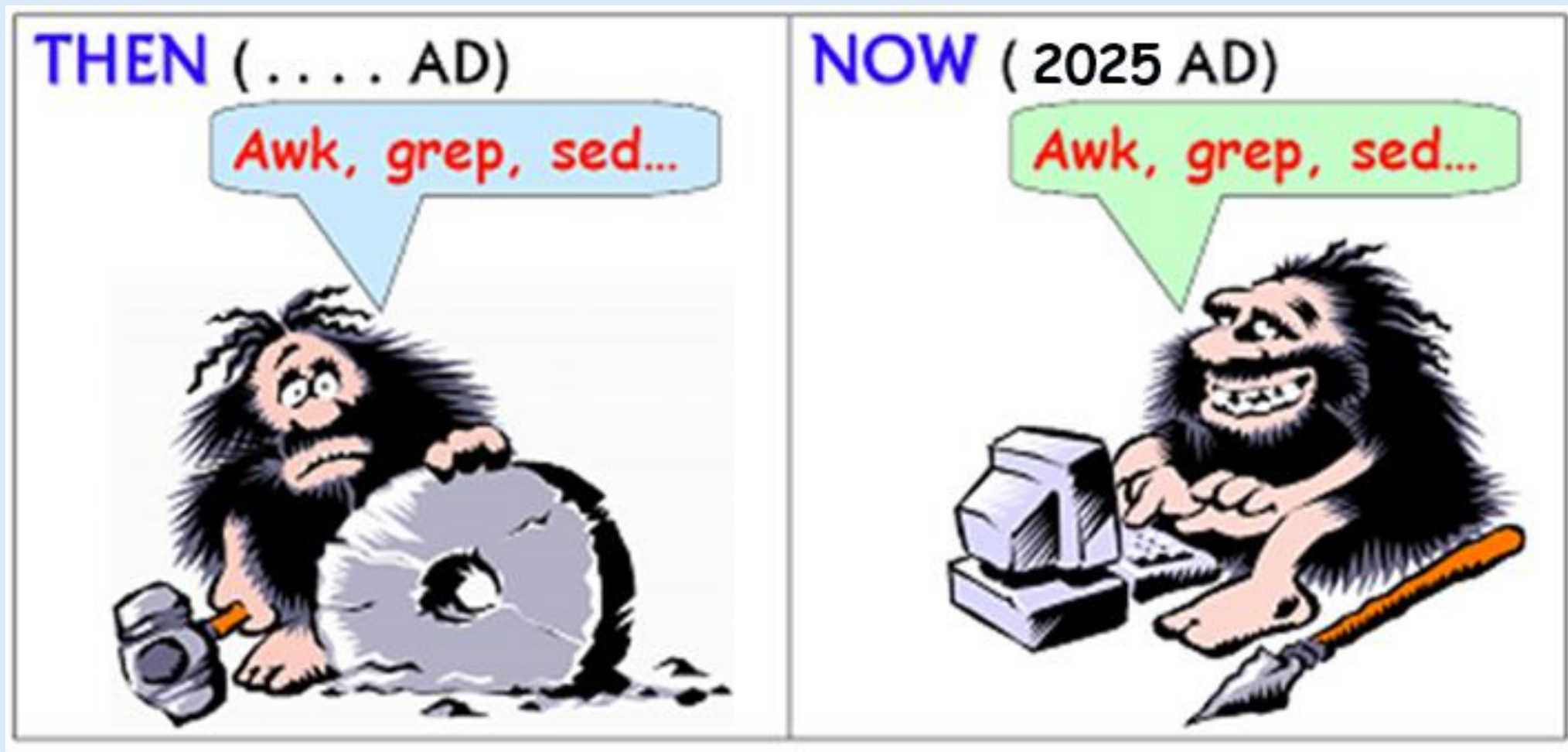


What I know:

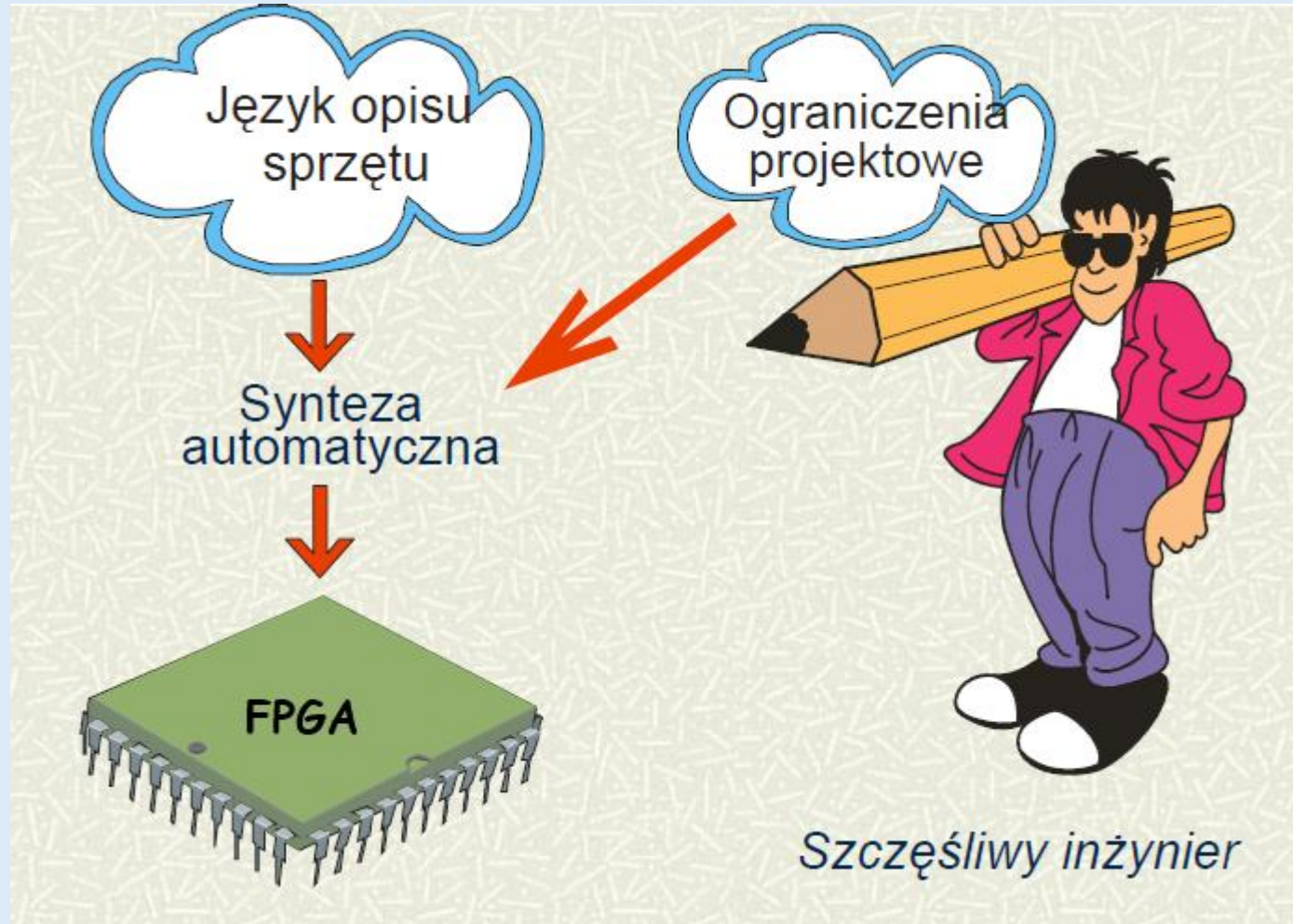
Finite state machine
Pipelining
Systolic array ...

**Formula
Manipulation
Mapping**

Czyżby historia zatoczyła koło?



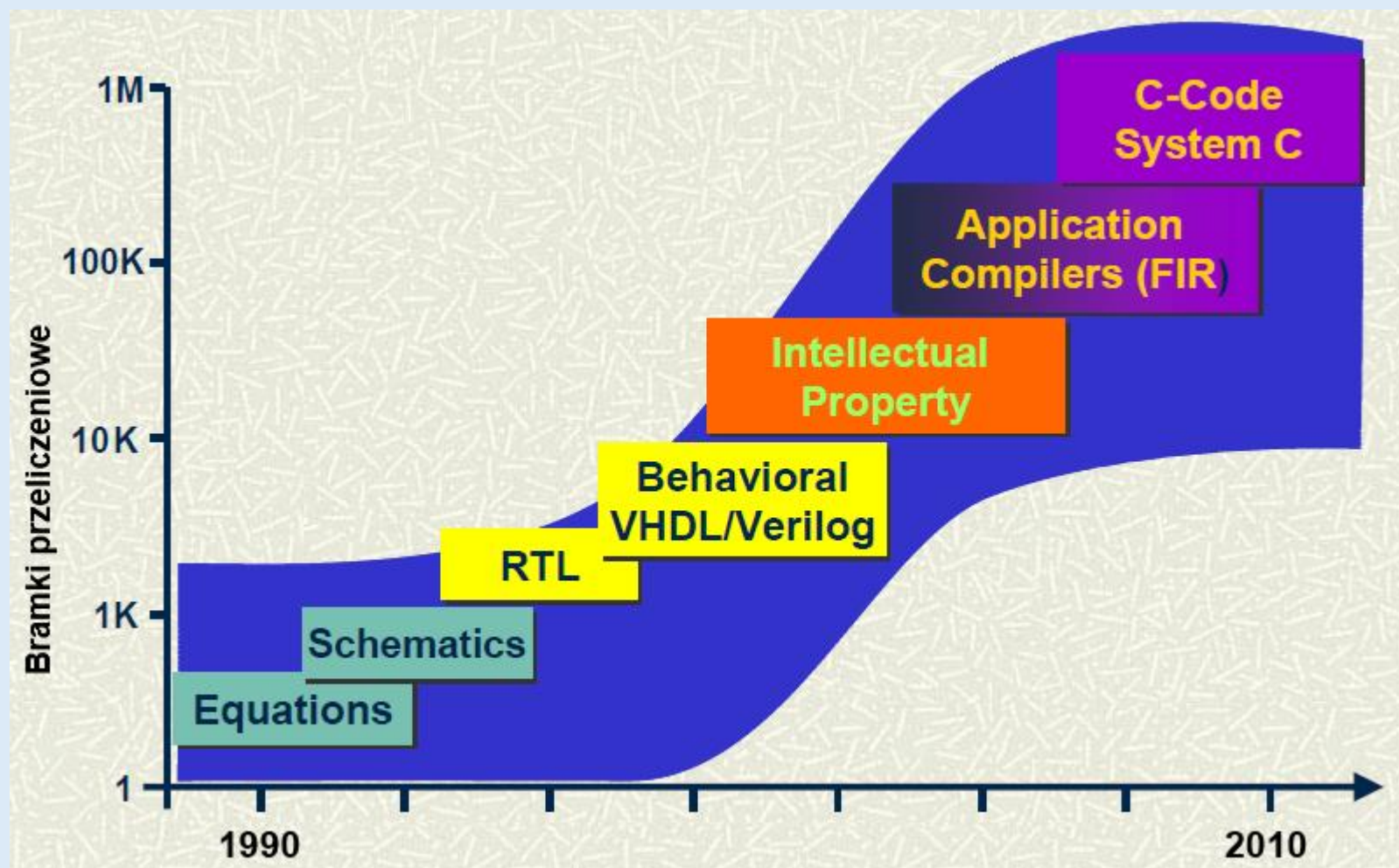
Czy projektowanie jest proste i czy konieczne trzeba zrozumieć FPGA?



Czy też może prowadzić do stanów depresyjnych – po co ta matematyka w technice cyfrowej?



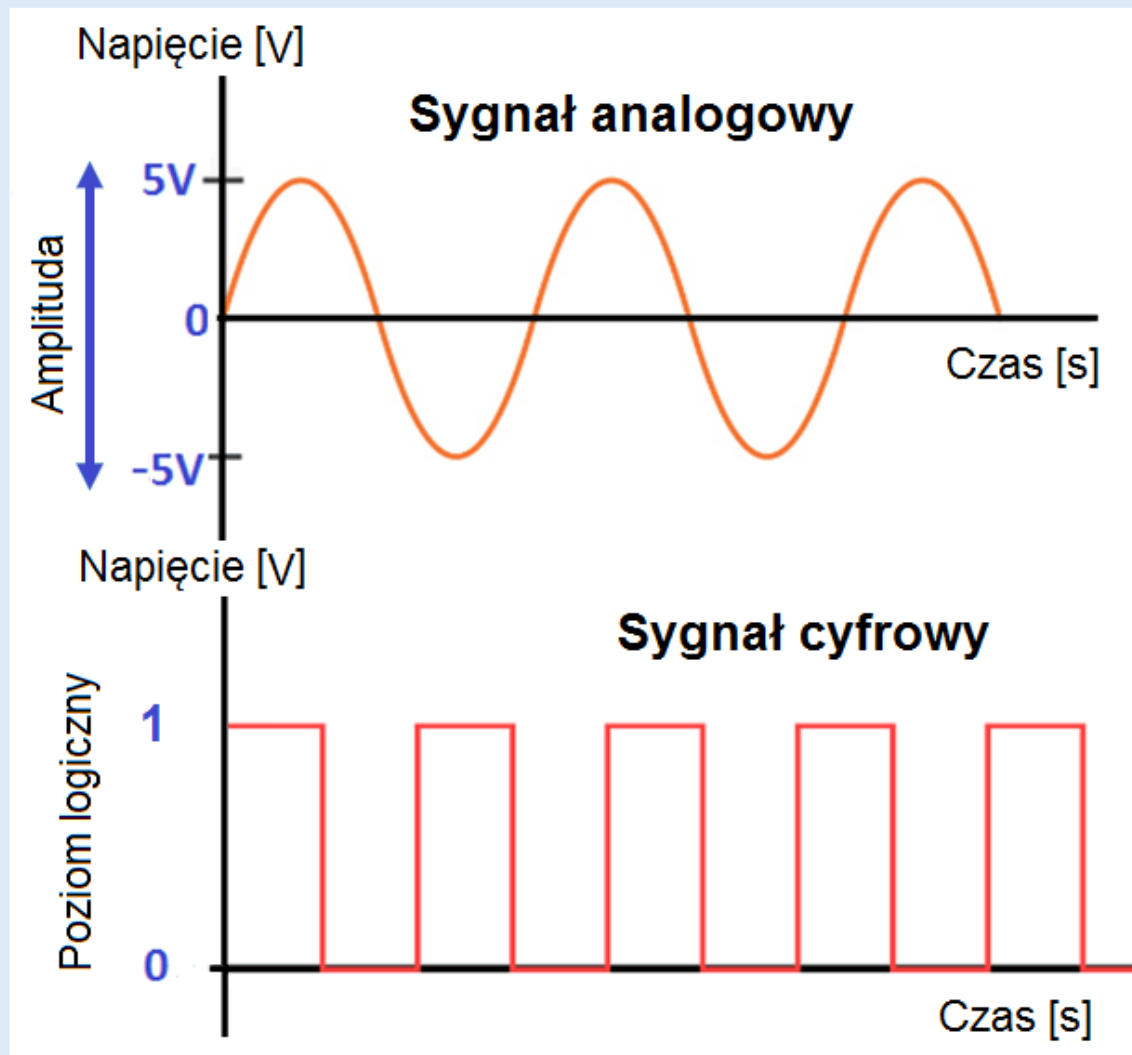
Różne poziomy abstrakcji i techniki w procesie projektowania układów cyfrowych:



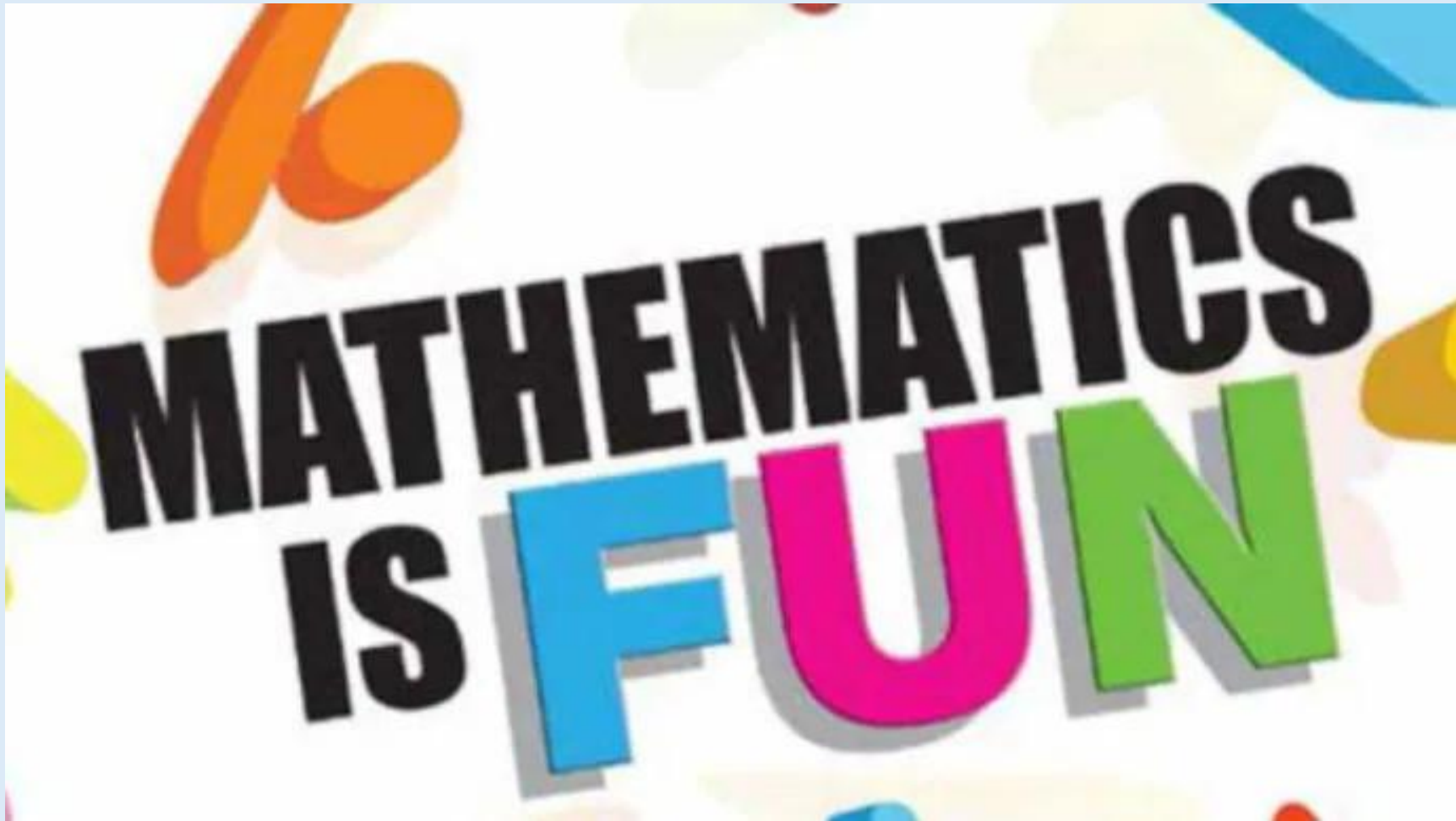
Używamy pojęć elektronika cyfrowa, technika cyfrowa – w jakim celu używa się terminu cyfrowa?



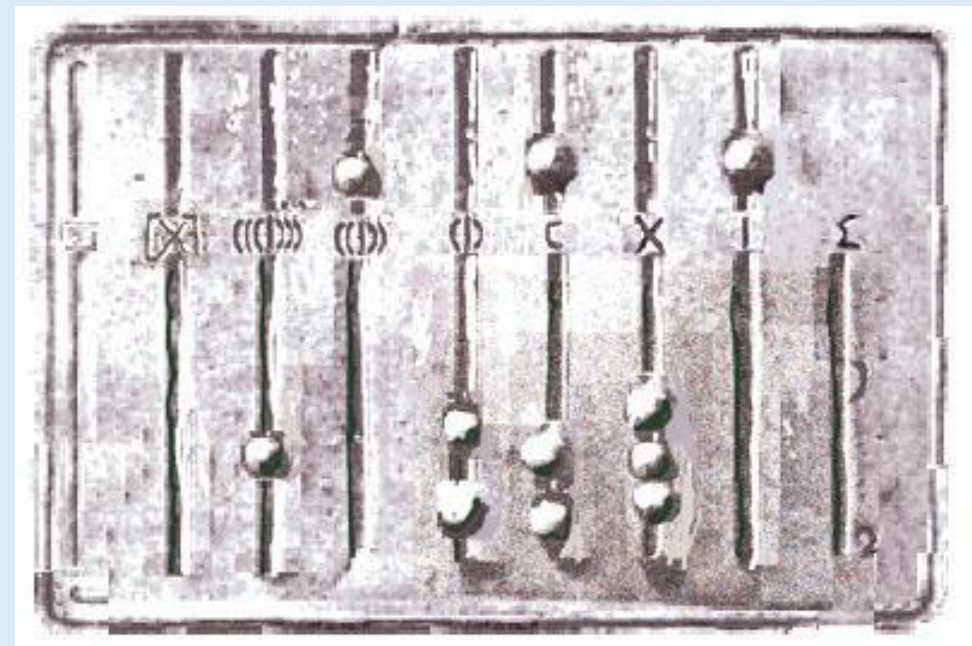
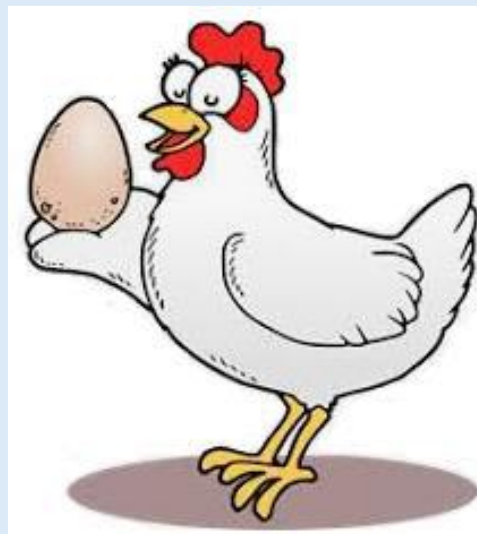
W elektronice/elektrotechnice przetwarzamy sygnały – czym różnią się sygnały: analogowy i cyfrowy?



Czy na pewno, a jeżeli już to jakie działy?



Który rodzaj matematyki, raczej który system liczbowy był opracowany jako pierwszy?



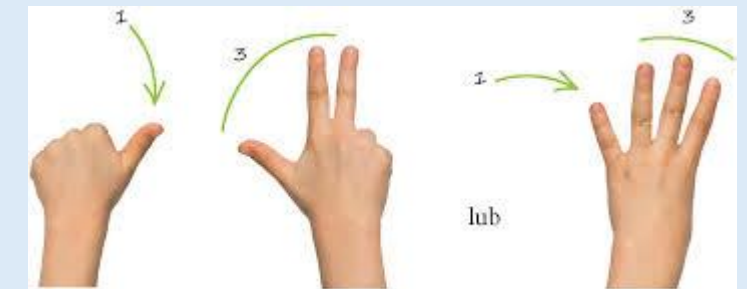
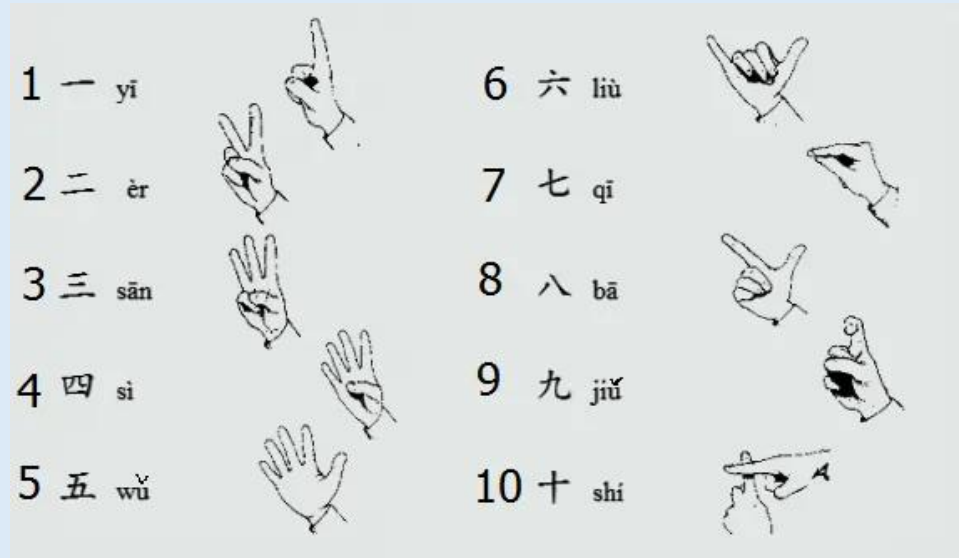
-20 000 ÷ -35 000 lat p.n.e. - nacinane rogi renifera, czy była to próba liczenia – czyli NARODZINY LICZB?



-10 000 p.n.e - kulki i żetony gliniane, Sumerowie zastosowali system umożliwiający przetwarzanie informacji (transakcje, zapamiętywanie, kontrola...) ?



-30 000 lat p.n.e. - liczenie na placach – Chińczycy używają jednej dłoni – wyrażają duże liczby rzędu mld, starożytny Rzym używał dwóch dłoni,

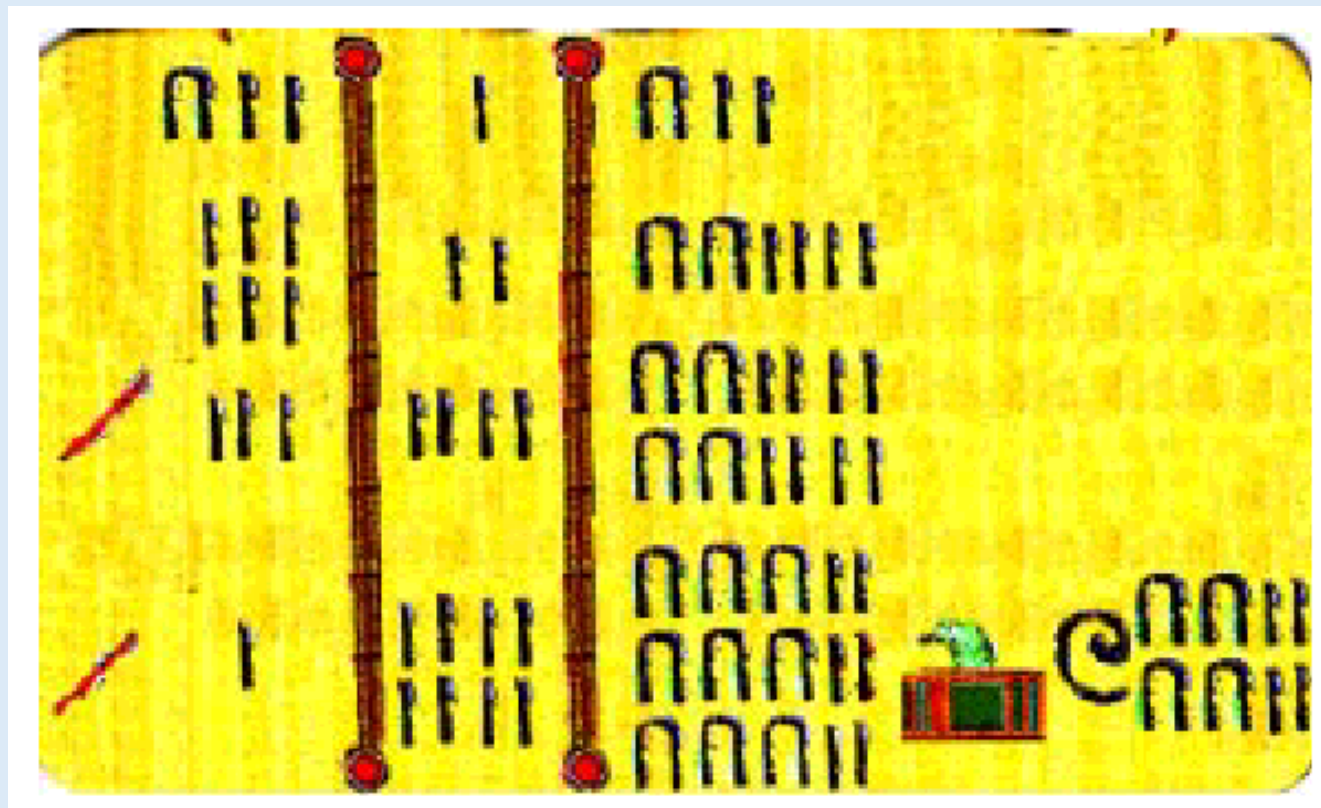


Bardzo szkodliwe jest zabranianie używania palców do liczenia dzieciom. Hamuje to rozwój matematyczny dziecka, uniemożliwia samodzielne zdobywanie i rozwijanie wiedzy matematycznej. Nawet gdy dziecko opanuje już liczenie w pamięci, w pewnych sytuacjach nadal może wspomagać się liczeniem na palcach. Badania dowodzą, że do tworzenia abstrakcyjnych pojęć wykorzystywane są te same obwody neuronalne, które pierwotnie służyły do wykonywania operacji percepcyjnych i motorycznych.

- 3 000 p.n.e.: chiński cesarz Fu Hi - ośmiokąt z trigramem - System binarny (chiński Ying i Yang). Pierwsze 8 cyfr w postaci binarnej, czyżby system binarny był wcześniej stosowany?



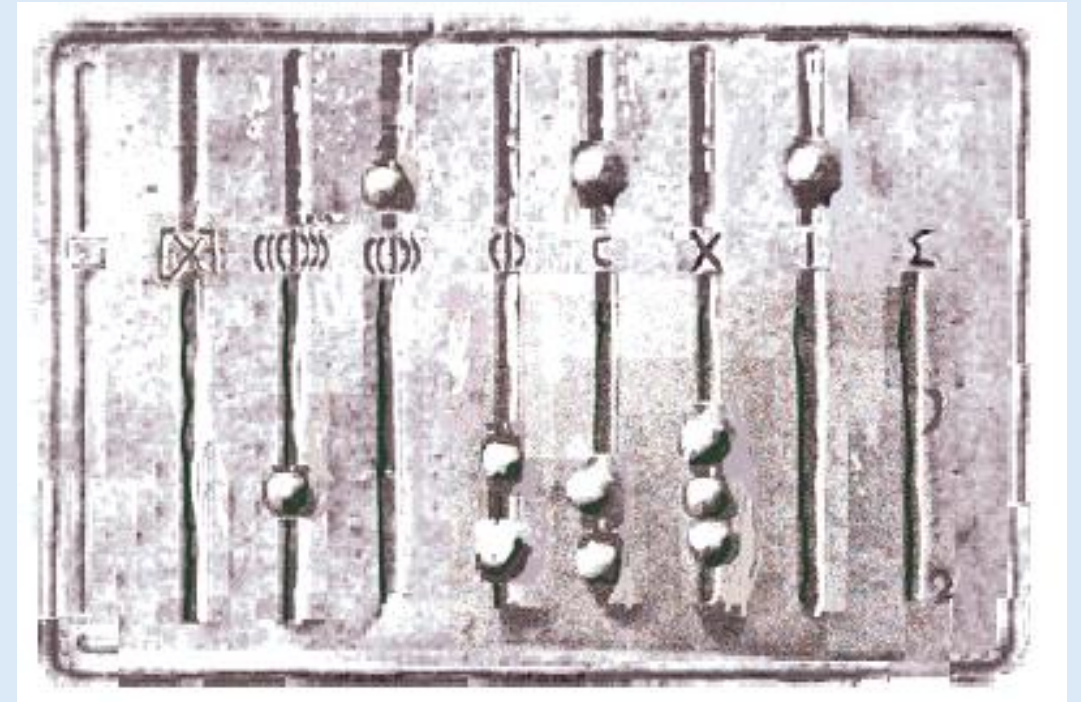
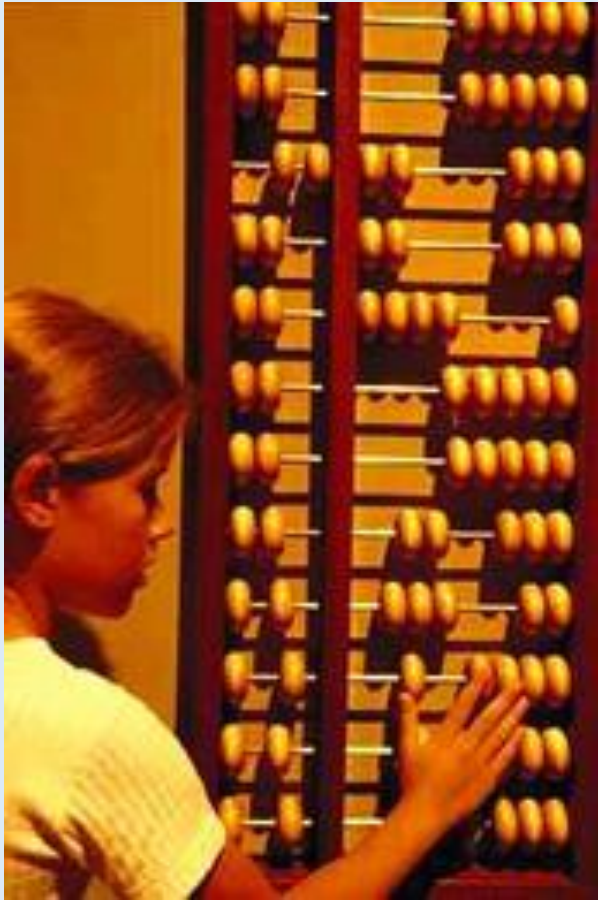
- 1 700 p.n.e. - egipskie papirus RIND, kupcy zapisywali transakcje – mnożąc binarnie ?



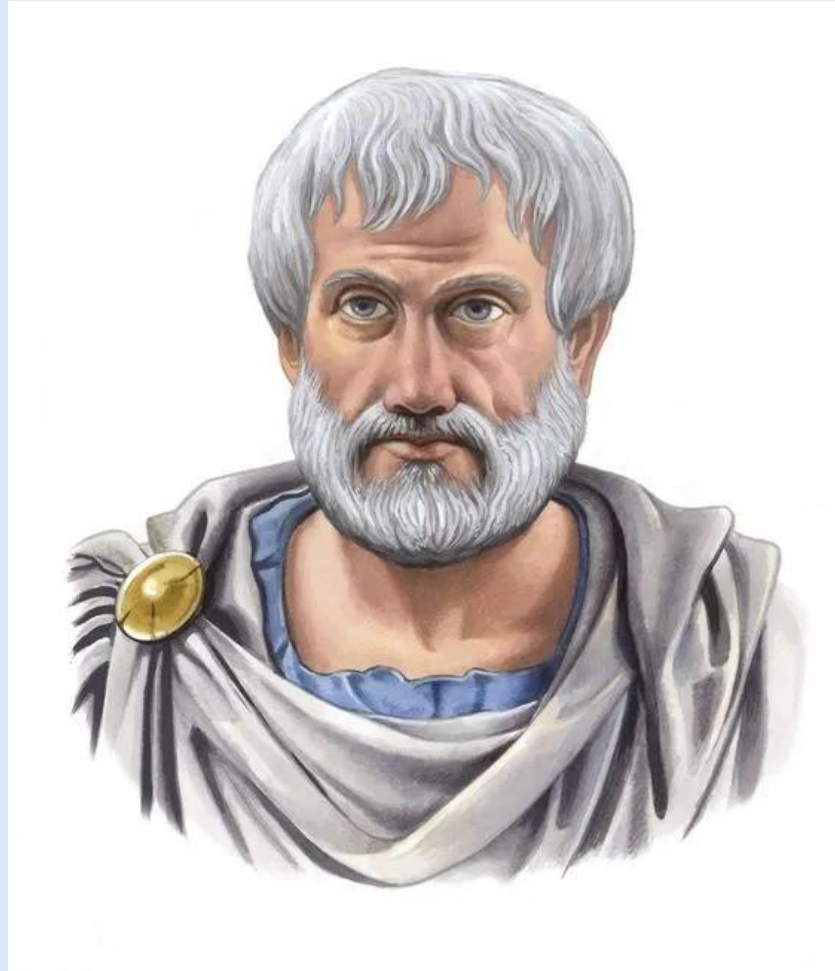


-1 750 p.n.e - Kodeks króla Mezopotamii
Hammurabiego - królewska sentencja
wnioskowania logicznego:
JEŚLI (osoba) I (czynność) WTEDY (wyrok)

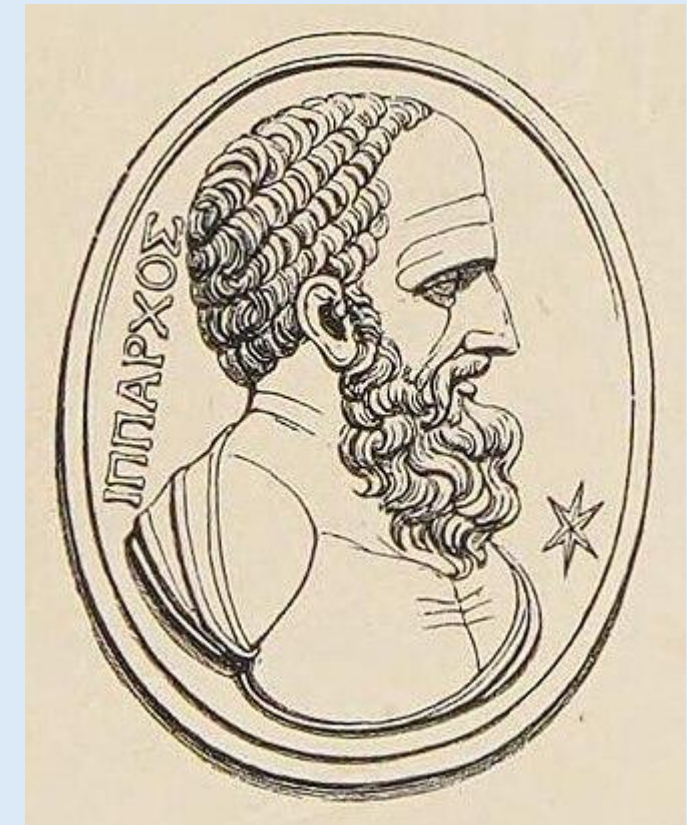
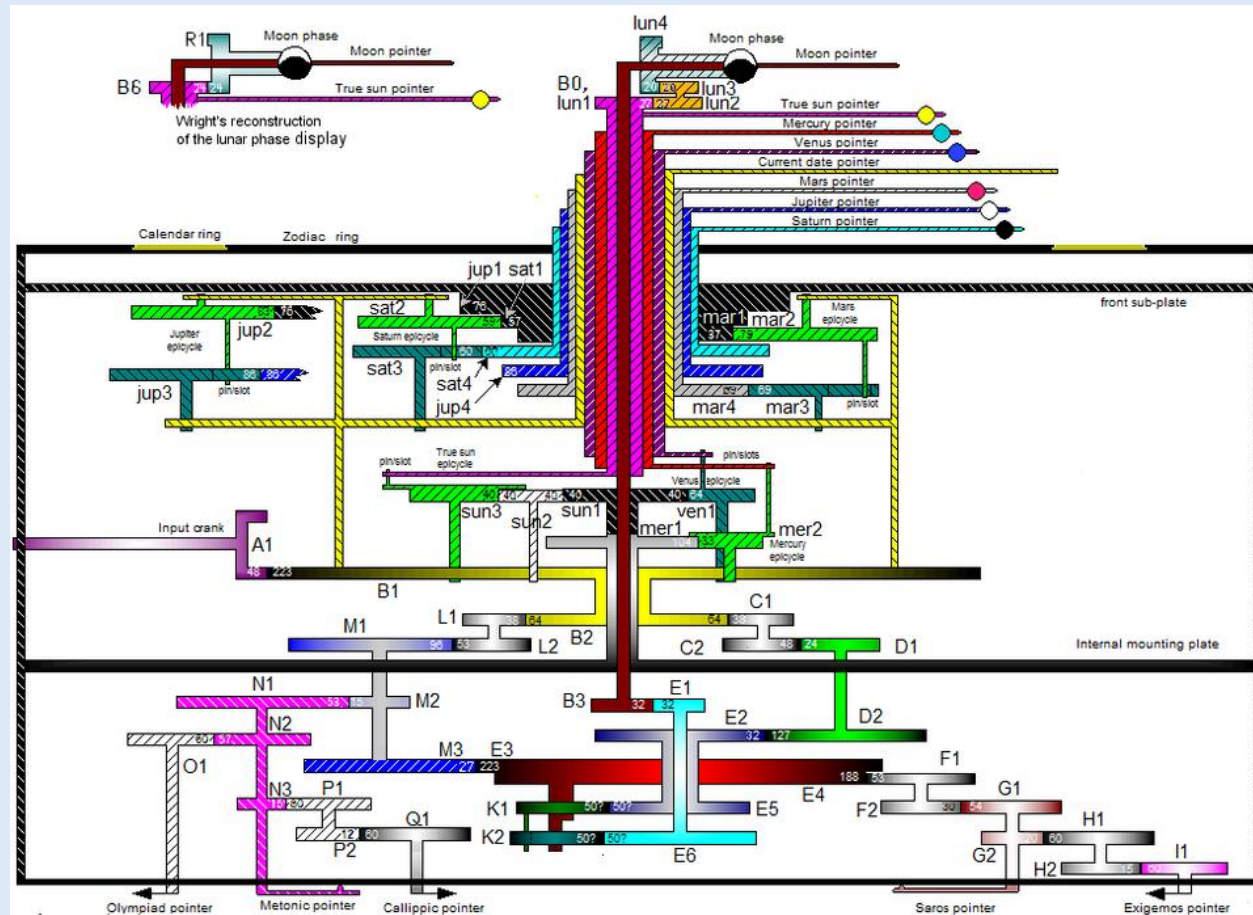
- 3000 (Chiny), -1 100 na Bliski Wschód, -500 Rzym - pierwsze „narzędzia” obliczeniowe – Abakus/liczydło. składający się z koralików przesuwanych po prętach, jest to już urządzenie obliczeniowe, które pozwala na przedstawienie liczby i wykonanie obliczeń na tej liczbie, czyżby początki Systemu dziesiętnego ?



-300 p.n.e. - **Arystoteles** w Organon rozwinął podstawy logiki: zasada tożsamości, sprzeczności i wyłączonego środka.



-150 ÷ -100 p.n.e. - **Hipparchos z Nikei**, uważany za prekursora komputerów, opracował mechanizm z Andikitiry, który służy do obliczania ruchu Słońca, Księżyca, Merkurego, Wenus, Marsa, Jowisza i Saturna, przewidywania zaćmień Księżyca i Słońca.



800 n.e. - **Muhammad ibn Musa al-Khwarizmi** (Bagdad) - promował pochodzący z Indii dziesiętny system liczenia i pozycyjnego systemu liczbowego. Wyjaśnił pojęcie zera, ułamków oraz funkcje trygonometryczne sinus i tangens. Przypisuje się dla niego również opracowanie pojęcia algorytmu – czyli podstaw przetwarzania sekwencyjnego – programowania współczesnych procesorów ?



1000 n.e. - **Gerbert z AURILLAC** (papież Sylwester II) – narzucił odkryte w Indiach pojęcie zera, które zostało zaakceptowane w świecie zachodnim dopiero w XIV wieku.





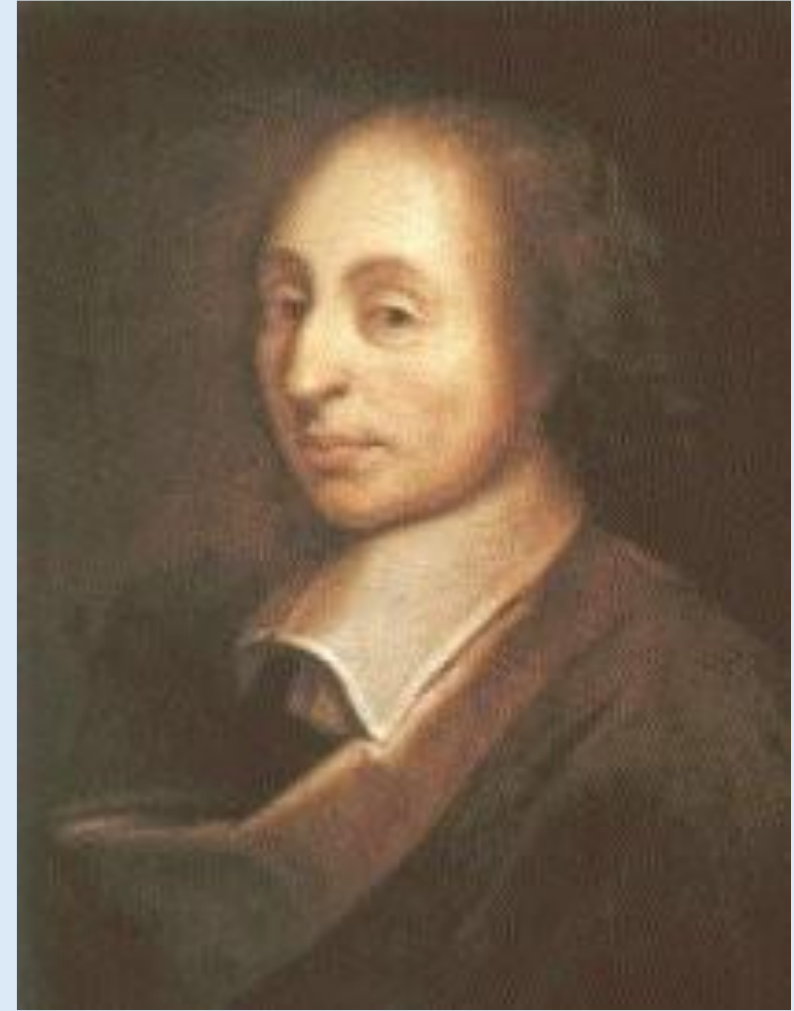
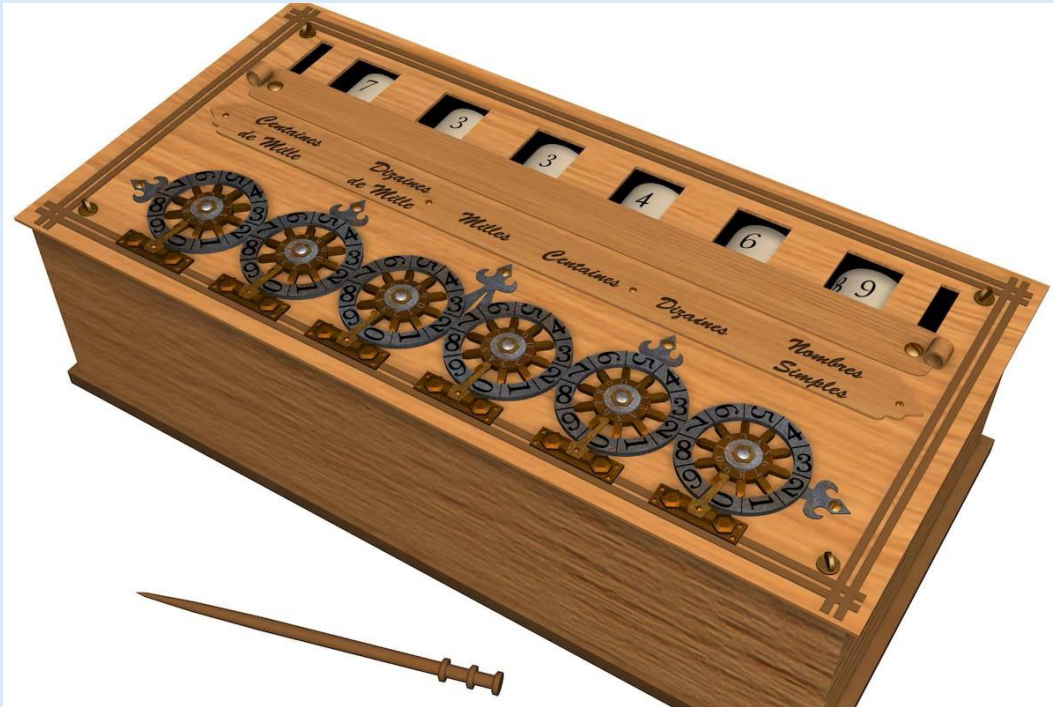
John NAPIER (1550-kwiecień 1617)

Szkocki matematyk, odkrył logarytmy i wynalazł metodę mechanicznego wykonywania operacji mnożenia, dzielenia i wyciągania pierwiastków kwadratowych. Dzięki opracowaniu tablic logarytmicznych – mnożenie i dzielenie można było zastąpić dodawaniem i odejmowaniem.

1623 - **Wilhelm Schickard** (pastor luterański)– niemiecki matematyk/konstruktor, opracował jedną z pierwszych na świecie maszyn liczących, wykonującą dodawanie, odejmowanie, mnożenie liczb całkowitych. Maszyna ta, złożona z elementów drewnianych, spłonęła w 1624 w czasie wojny trzydziestoletniej. Maszyna spłonęła w 1624.



1641 – **Blaise/Błażej PASCAL** - w wieku 11 lat napisał pracę dotyczącą dźwięków wydawanych przez wibrujące ciała, w wieku około 16 lat sformułował twierdzenie Pascala, w wieku 22 opracował maszynę dodającą i odejmującą liczby – Pascalinę. Zastosowanie zamiast kół zębatach kół z kołeczkami czyniły maszynę zawodną.



1632 n.e. - **William Oughtred** wynalazł suwak logarytmiczny do wykonywania takich działań jak: mnożenie, dzielenie, potęgowanie, pierwiastkowanie, wyznaczanie czwartych proporcjonalnych, średnich geometrycznych, a także wykonywanie obliczeń trygonometrycznych.

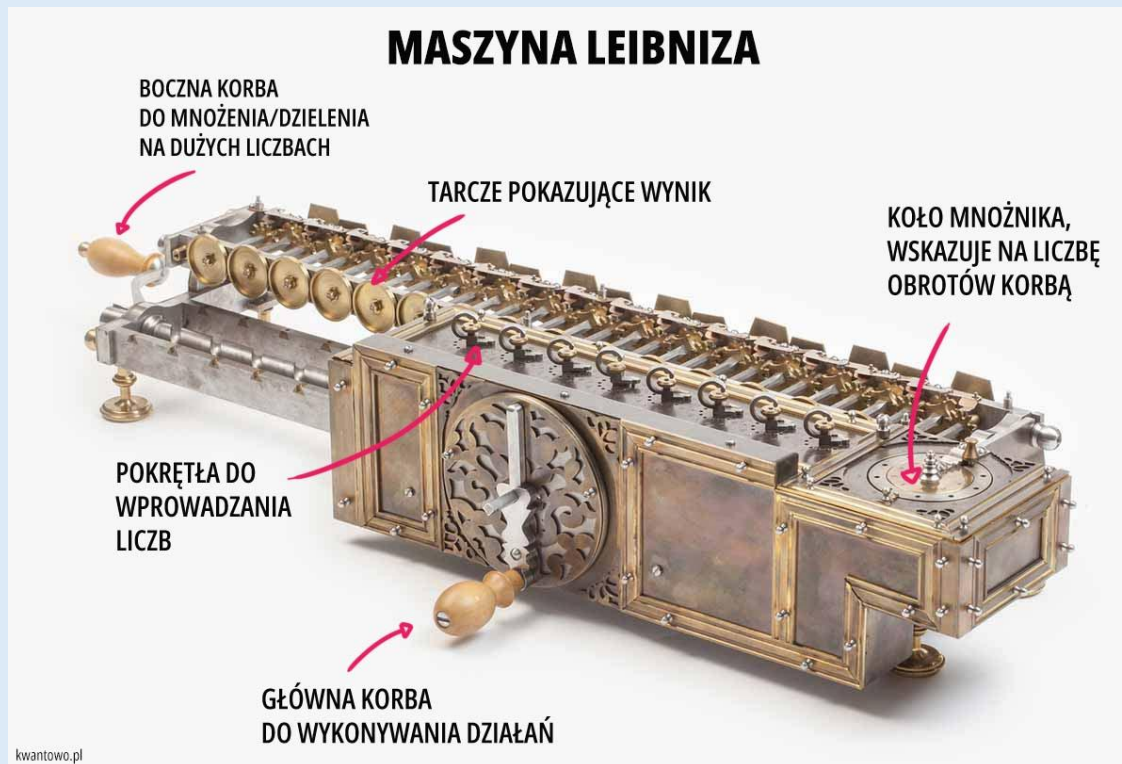
W praktyce inżynierskiej stosowano go do końca lat 70-tych – np. w Technikum Elektrycznym w Białymstoku.



Niegodnym jest, aby znakomici ludzie tracili godziny jak niewolnicy na pracę przy obliczeniach.

Gottfried Wilhelm Leibniz

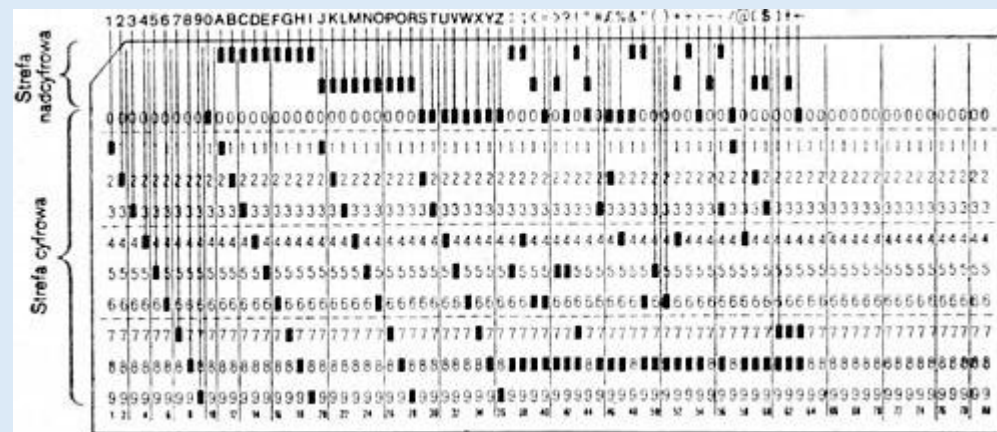
1671 n.e. – prawnik, filozof i bibliotekarz - został uznany za jednego z ważniejszych logików, przedstawił założenia koniunkcji, negacji i tożsamości. Opracował nowy rodzaj ruchomej przekładni – nazywany kołem Leibniza, w postaci wałka z zestawem dziewięciu zębów o stopniowo przyrastającej szerokości. Była to pierwsza w pełni mechaniczna maszyna do mnożenia - machina arithmetica. W mechanizmie złożonym z 650 drobnych elementów, co rusz dochodziło do awarii. Jego prace stały się podstawą późniejszych wynalazków.



1728 n.e. - **Jean-Baptiste Falcon** udoskonalił krosno tkackie, zamieniając wałowe wzornice na wzornice w formie perforowanych papierowych kart. Falcon zespolił je w sznur łączając je w ciąg kart.

1748 n.e. **Jacques Vaucanson** opracował mechanizm sterowania osnowami poprzez maszyny czytające drewniane perforowane karty – czyli pamięć programu np. Odra 1300 , stosowaną w latach późnego PRL-u.

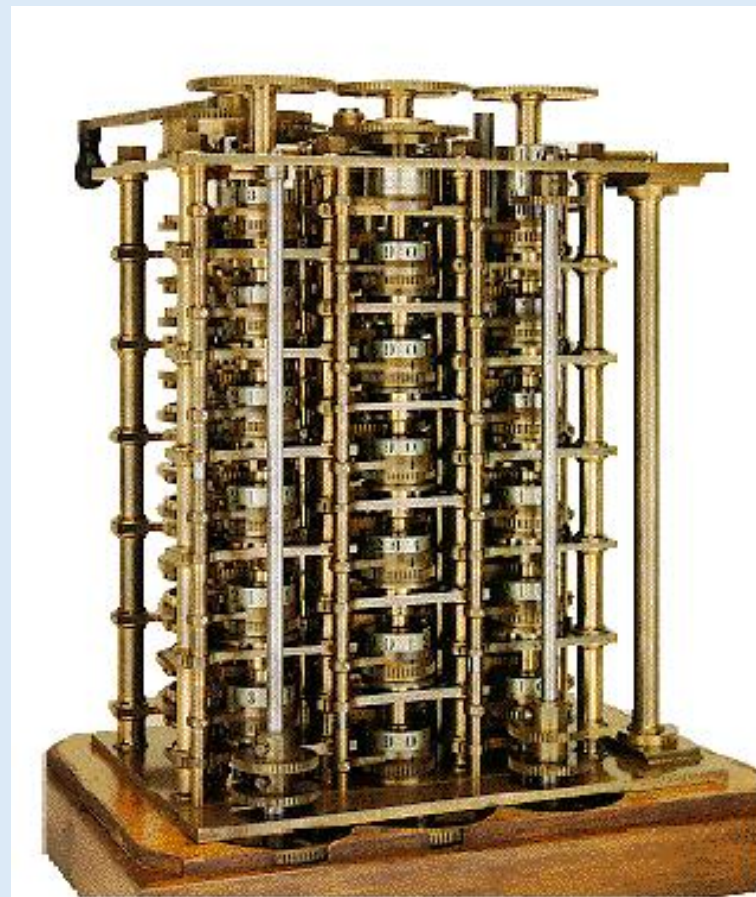
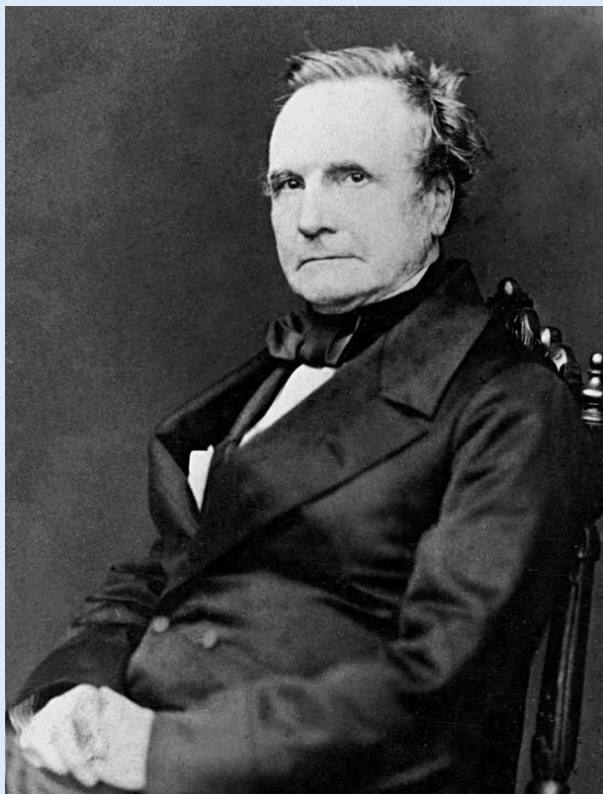
1775 n.e. **Philippe de Lasalle** wprowadził możliwość przenoszenia wzornic pomiędzy krosnami.



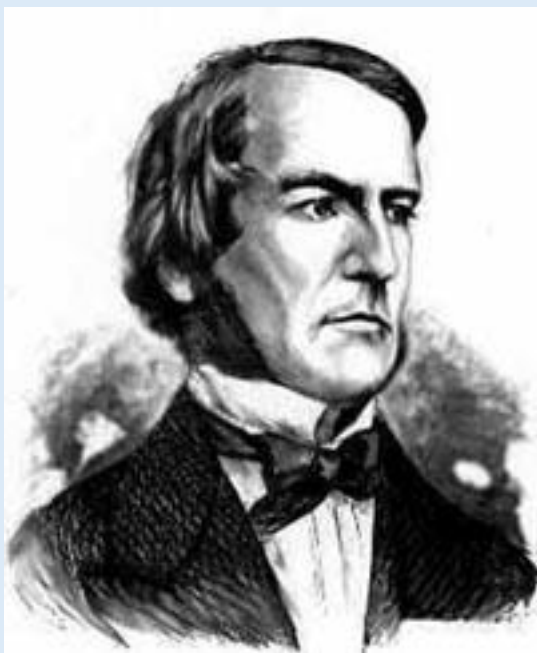
1805 n.e. – **Joseph-Marie Jacquard** udoskonalił metodę automatycznego sterowanie układem wątku i osnowy nici - kod na taśmie perforowanej sterował haczykami wybierającymi nici odpowiedniego koloru do wzorów na tkaninach. W roku 1812 r. we Francji pracowało około 11 000 tego typu maszyn.



1805 n.e. – **Charles BABBAGE** opracował maszynę obliczeniową, która posiadała cechy współczesnego komputera. Zawierała ona: jednostkę obliczeniową do wykonywania 4 operacji – odpowiednik procesora, odpowiednik pamięci ROM – kilka obracających się bębnow z modyfikacją położenia 200 kołków, pamięć masową z kilkoma czytnikami kart perforowanych - co odpowiadało pojemności 6,6 kB. 3 m wys. x 7 m dł.



1854 n.e. – **George BOOLE** opublikował "An Investigation of the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probabilities" - Badanie praw myślenia, na których oparte są matematyczne teorie logiki i prawdopodobieństwa, tworzy tzw. algebrę Boole'a dającej podstawy matematyczne współczesnej techniki cyfrowej i informatyki. Służy ona do rozwiązywania problemów związanych z teorią zbiorów, do algebraicznego przedstawiania logiki zdań oraz do algebraicznej manipulacji i opisu obwodów elektronicznych.



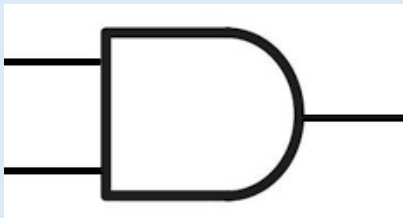
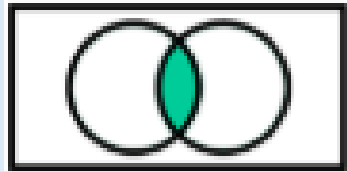
Algebra Boole'a jest oparta na 3 działaniach logicznych:

- alternatywa – OR
- koniunkcja) - AND
- negacji – NOT,

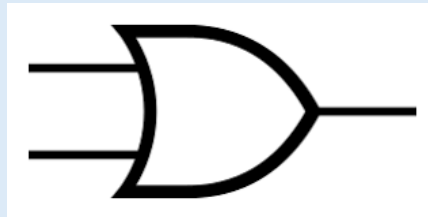
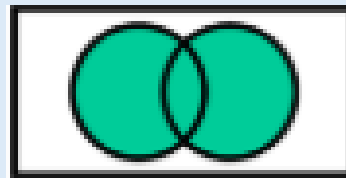
oraz wyróżnia dwie wartości: 0 - fałsz, 1 - prawda.

Różne reprezentacje podstaw algebry Boole'a :

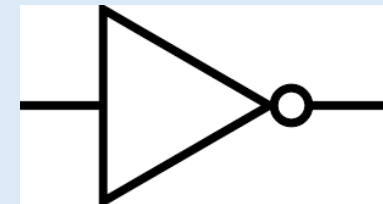
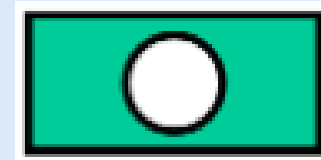
X AND Y		
X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1



X OR Y		
X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

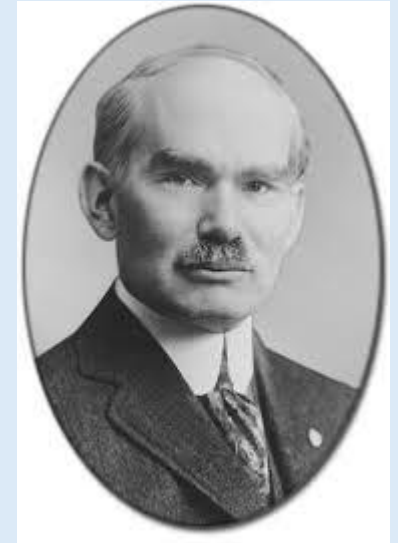
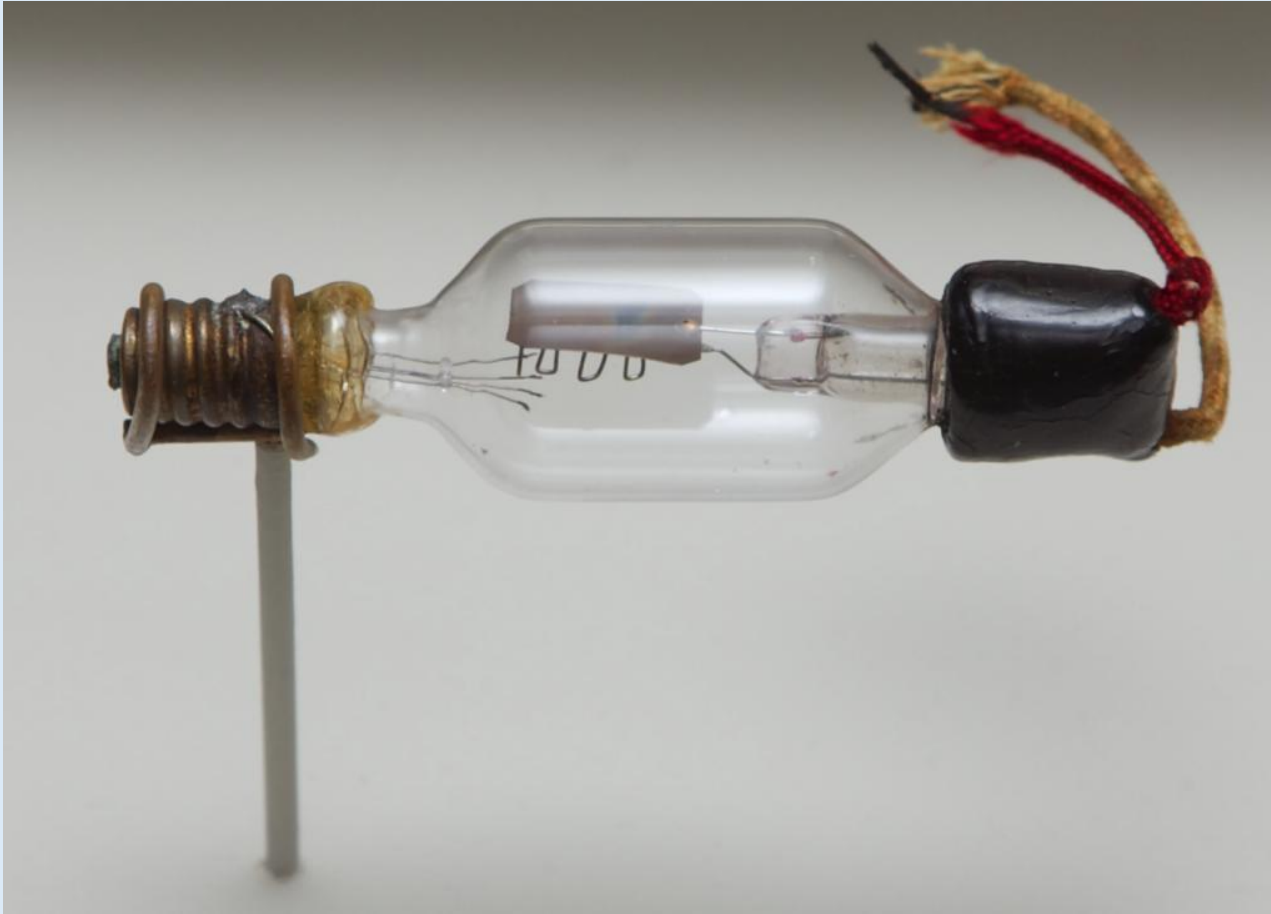


NOT X	
X	$\overline{X}$
0	1
1	0

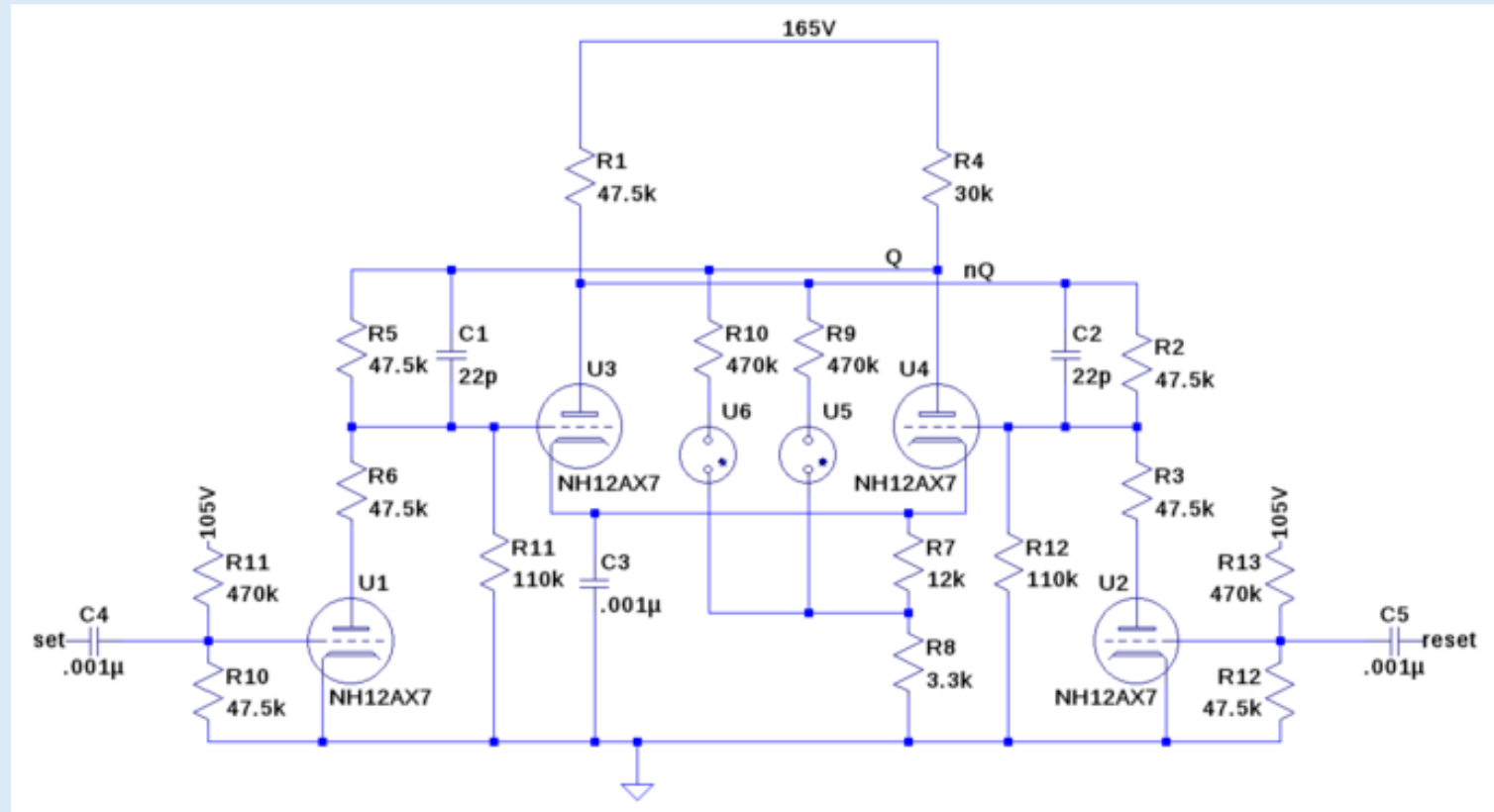


Mimo istnienia podstaw matematycznych, barierą rozwoju elektroniki cyfrowej była technologia: brak miniaturowych, niskoenergetycznych elementów przełączających i wzmacniających.

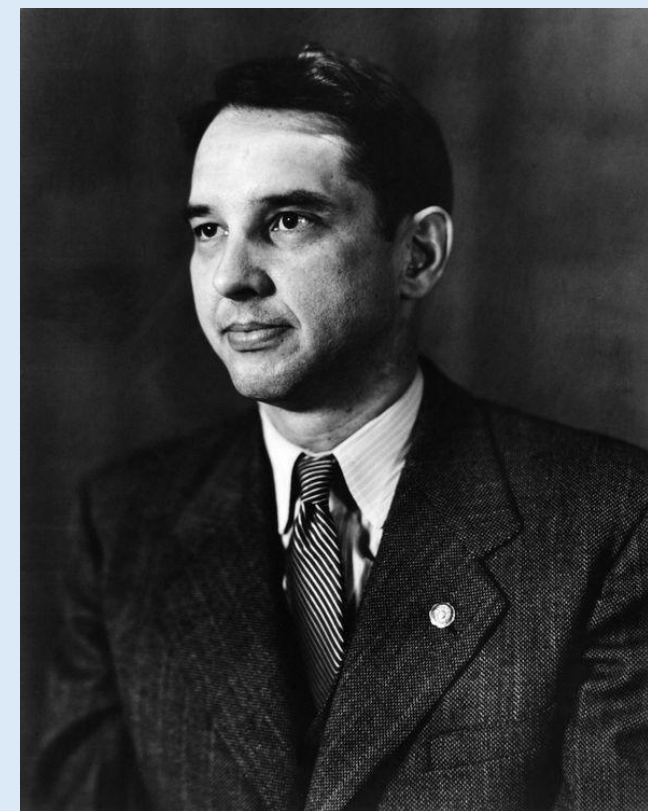
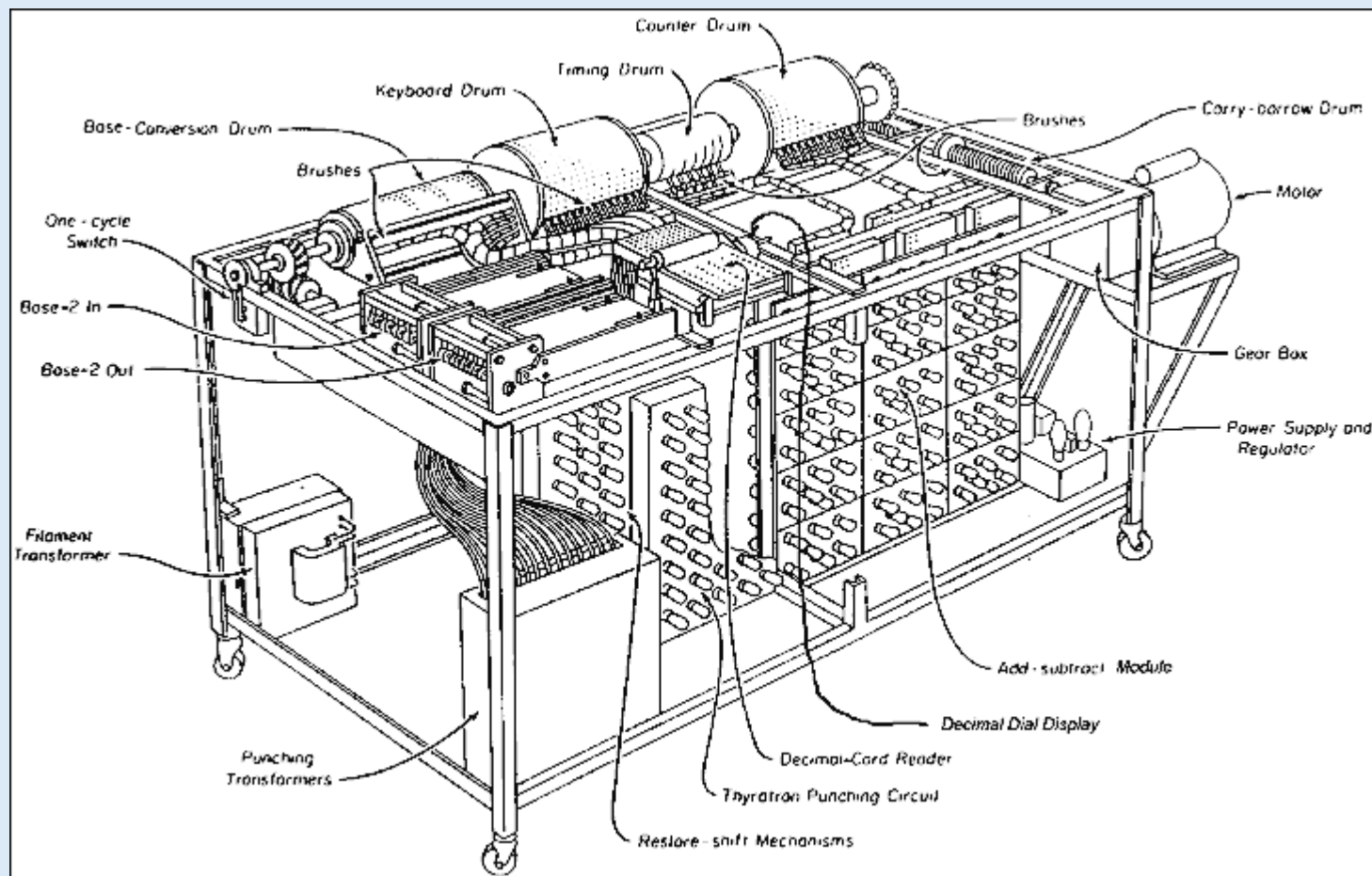
1906 - amerykański wynalazca **Lee de Forest** opracował lampę próżniową triodę. Mogła ona być używana zarówno jako wzmacniacz, jak i przełącznik.



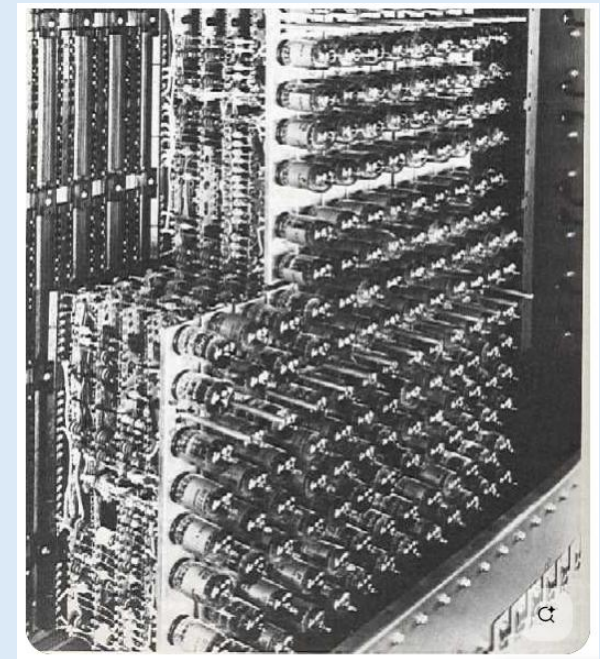
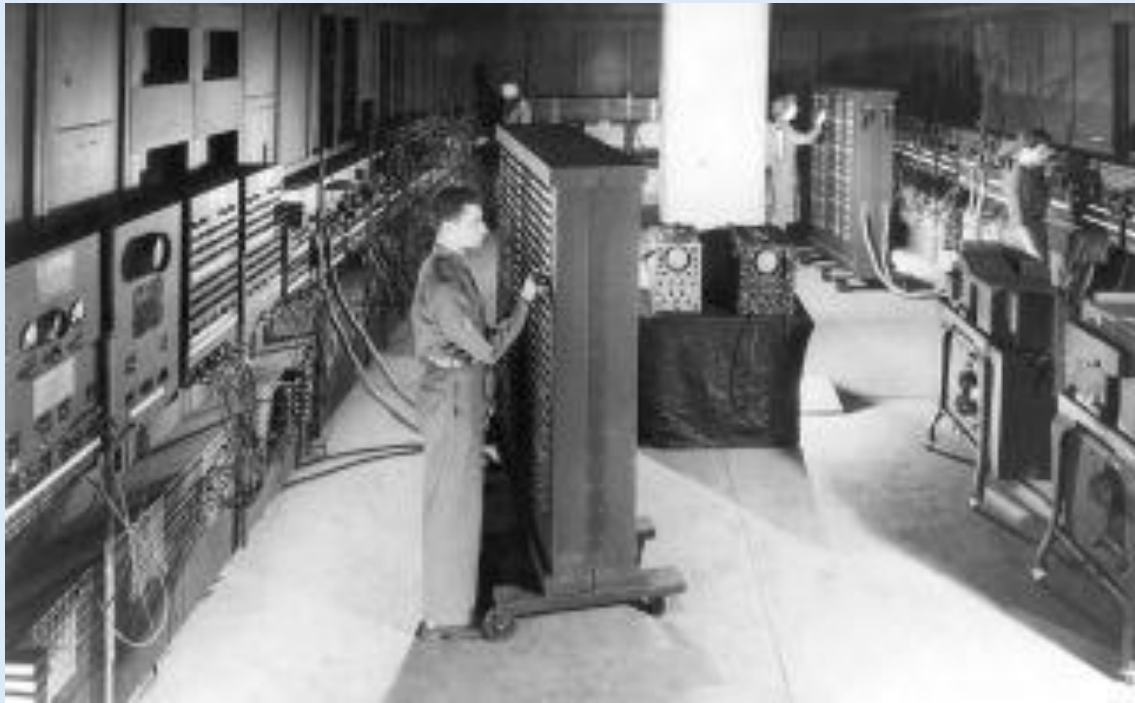
1912 – opracowanie lampowego przerzutnika RS – elementu pamiętającego komórki RAM



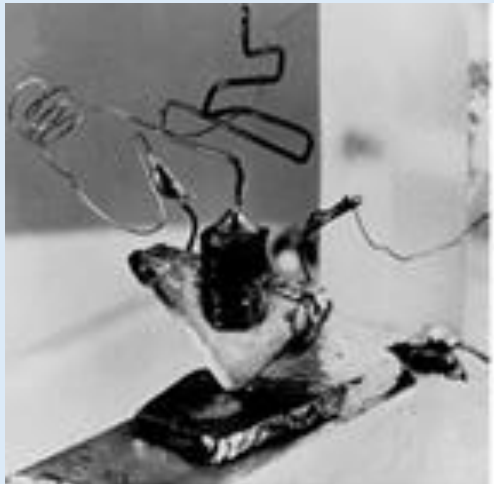
1937 - matematyk i fizyk **John Vincent Atanasoff** opracował koncepcję pierwszego lampowego komputera ABC - Atanasoff-Berry Computer. Maszyna była około 1000 razy szybsza od stosowanych wówczas urządzeń mechanicznych. Zawierała 270 lamp elektronowych.



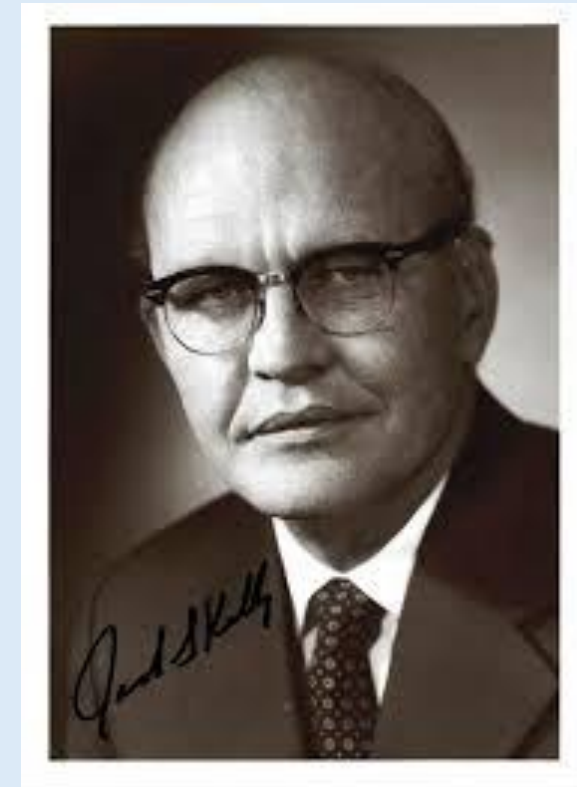
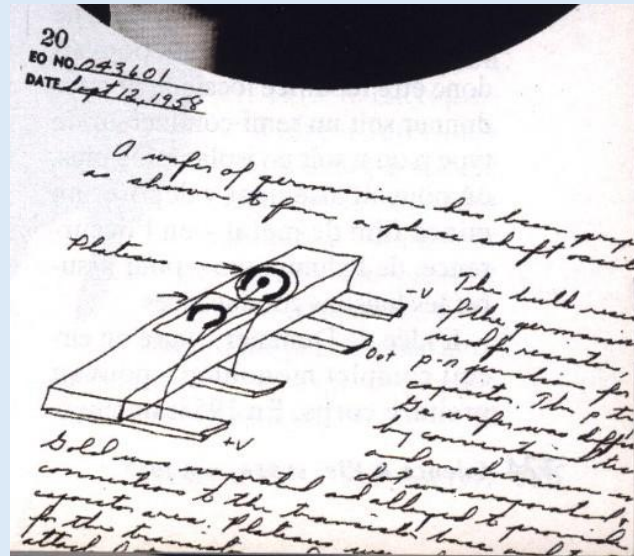
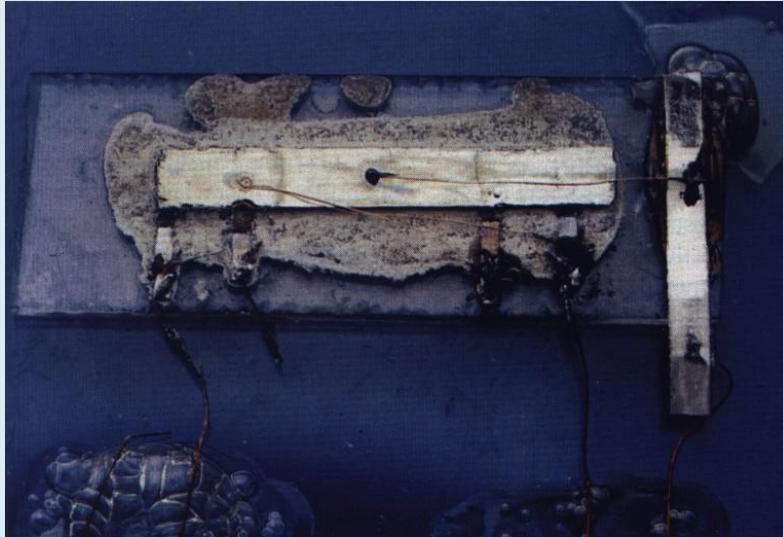
1946 - komputer ENIAC, programowany ręcznie poprzez ustawianie przełączników i wymianę specjalnych tablic programowych wymiary 15 metrów x 9 m, 30 ton, 18000 lamp elektronowych. Liczby były zapamiętywane w systemie dziesiętnym, a nie dwójkowym. Komputer wykonywał 5000 dodawań i do 360 dzieleni na sekundę, pamięć wewnętrzna 80 bajtów, pobór mocy do 200 kW.



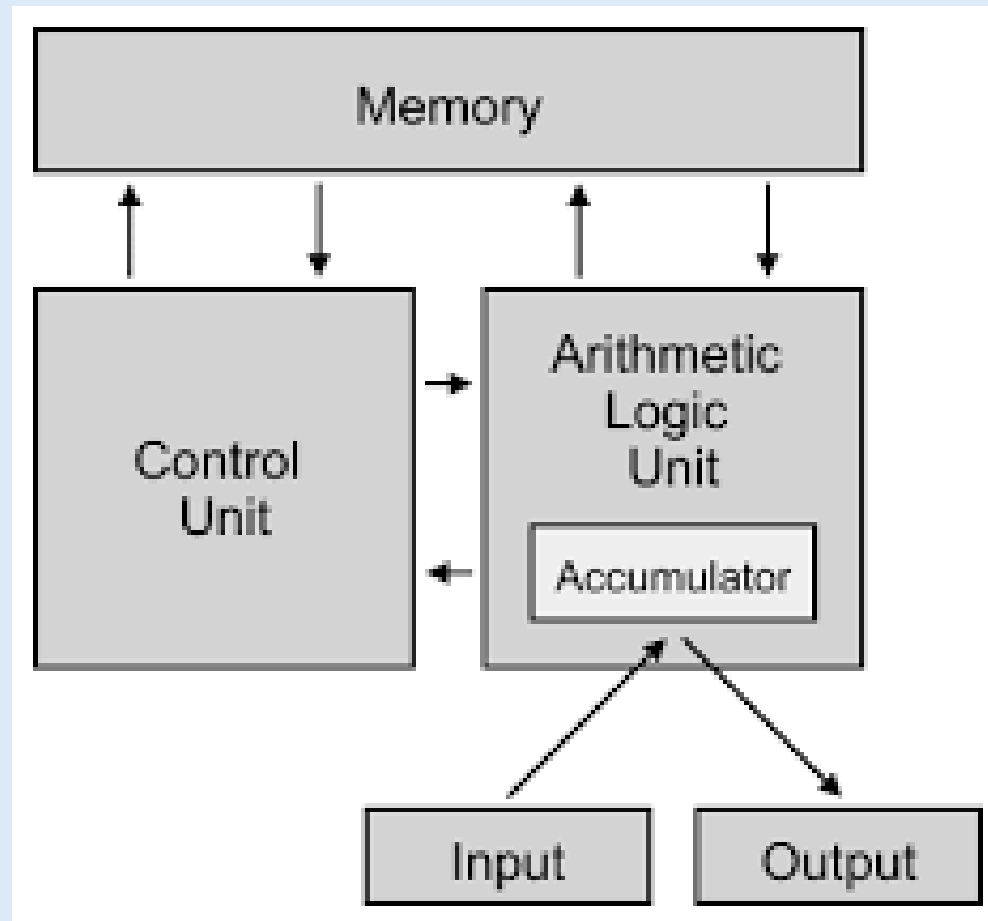
1947 – przełom w technologii - **John Bardeen**, **Walter Brattain** i **William Shockley** – opracowanie tranzystora. Nagroda Nobla w dziedzinie fizyki w 1956 roku, Patent No. 2502488.



1959 - **Jack St. Clair Kilby** - opracował pierwszy układ scalony. Nagroda Nobla w dziedzinie fizyki w 2000 r. W firmie Texas Instruments powstał na pojedynczym kawałku półprzewodnika oscylator ze zmianą fazy.



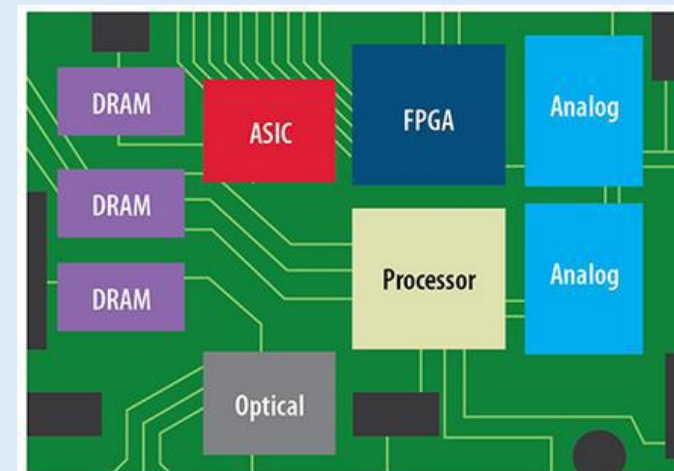
1943 - **János Lajos Neumann/John von Neumann** – urodzony w Budapeszcie, posiadał fotograficzną pamięć
- po krótkim spojrzeniu na stronę książki potrafił cytować dokładnie jej zawartość. Opracował architekturę pierwszego mikroprocesora.



... a potem poszło już szybko:

- 1961 FET,
- 1962 SUHL/9000/TTL (Texas)
- 1968 SRAM
- 1970 DRAM (INTEL)
- 1971 Procesor 4004 INTEL
- 1972 PROM
- 1973 Internet (25 machines)
- 1974 Intel 8080 – system CP/M
- 1975 PLD/FPGA/ASIC
- 1976 Apple I (1976)
- 1984 CD-ROM – PC/AT
- 1989 pierwszy Laptop
- 1992 Windows 3.1
- 2010 Intel Core i7
- od 1945 roku zaczęły powstawać języki programowania:
FORTRAN, ALGOL, COBOL, ADA, BASIC, PASCAL,
C, C++, Java – w sumie powstało około 120,
- ...
- elektronika kwantowa, AI, maszyny oparte na DNA i co dalej ?

MOEMS: elektronika, fluidyka,
mechanika, magnetyka, optyka,
chemia – w tym samym układzie.



bit czy kubit ?

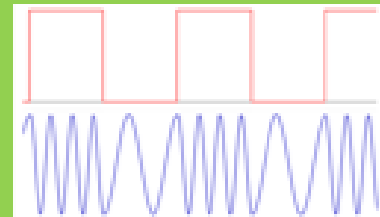


Bit fizycznie/technicznie może oznaczać:

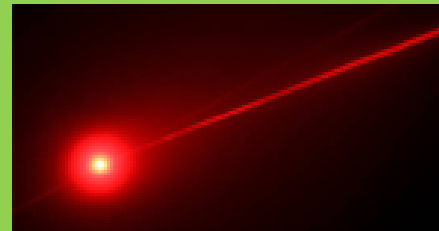
-- poziom napięcia w układzie/na przewodzie/ścieżce,



-- parametry sygnału radiowego w telekomunikacji,



-- parametry wiązki laserowej w otwartej przestrzeni,



Zaliczenie cząstkowe wykładu nr 1 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

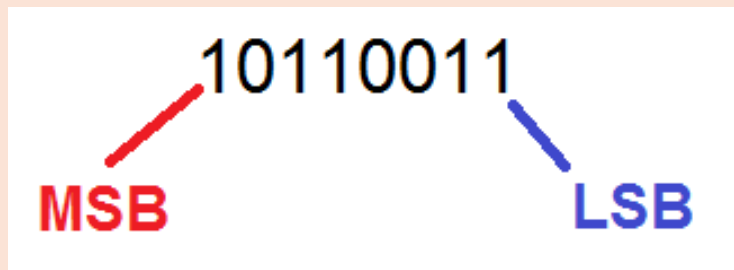
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

- **Bit** jest najbardziej elementarną jednostką informacji w systemach cyfrowych:
 - oznacza on stan włączenia (on) lub wyłączenia (off) w układach cyfrowych,
 - alternatywnie nazywa się: – **high/on** wysoki poziom napięcia, **low/off** niski poziom,
 - wartości bitów zapisujemy jako:
1 = high/on, 0 = low/off, a w VHDL odpowiednio '1' i '0'.
- **Bajt** (byte) jest ciągiem 8 bitów:
 - jest on najmniejszą jednostką informacji przetwarzaną w procesorach - co jest zaszłością historyczną,
 - jest też jednostką pojemności danych cyfrowych – mówi się 1 kilobajt (1 kB) a nie 8 tysięcy bitów (8 kb),
 - sposób zapisu wartości w bajtach jest łańcuchem – np.: 10110011 lub w VHDL "10110011",
 - bit leżący najbardziej **na lewo** jest bitem najbardziej znaczącym (**MSB** - Most Significant Bit) – a bit najbardziej **na prawo** jest bitem najmniej znaczącym (**LSB** – Least/Last? Significant Bit).
- **Słowo** (word) jest dłuższym ciągiem danych niż 1 bajt – 16/32/64 bity, ale też 5 czy 7 bitów.

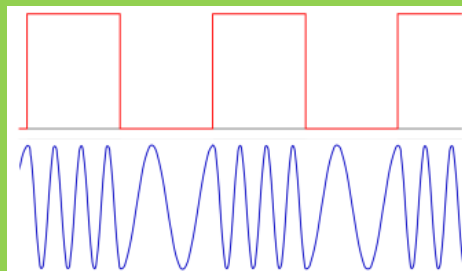


Bit fizycznie/technicznie może oznaczać:

-- poziom napięcia w układzie/na przewodzie/ścieżce,



-- parametry sygnału radiowego w telekomunikacji,



-- parametry wiązki laserowej w otwartej przestrzeni,



Powszechnie posługujemy się systemem dziesiętnym – w którym dowolną wartość wyrażamy za pomocą 10 cyfr: 0, 1, 2, 3, 4, 5, 6, 7, 8 i 9.

- Czyli wartość całkowitą 947 możemy zapisać jako:
9 setek + 4 dziesiątki + 7 jednostek – zatem w zapisie matematycznym mamy:

$$9 \times 100 + 4 \times 10 + 7 \times 1 = 9 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$$

- Podobnie możemy zapisać wartość ułamkową 5836.47 jako:
5 tysięcy + 8 setek + 3 dziesiątki + 6 jednostek + 4 dziesiąte + 7 setnych:

$$5 \times 1000 + 8 \times 100 + 3 \times 10 + 6 \times 1 + 4 \times 0.1 + 7 \times 0.01 = \\ 5 \times 10^3 + 8 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 + 4 \times 10^{-1} + 7 \times 10^{-2}$$

Zatem używamy systemu o podstawie 10 stosującego cyfry: 0, 1, 2, 3, 4, 5, 6, 7, 8 i 9.

Żeby wyrazić dowolną liczbę w tym systemie: na poszczególnych pozycjach wpisujemy cyfry odpowiednich potęg podstawy – zaczynając od kropki dziesiętnej w lewo dla części całkowitej i/lub w prawo dla części ułamkowej liczby.

Podobnie jak w systemie dziesiętnym, ta sama zasada obowiązuje w systemach cyfrowych – w tym w najpopularniejszym z systemów tej kategorii, czyli w **Naturalnym Systemie Dwójkowym** lub Naturalnym Kodzie Binarnym (skrót NSD / NKB).

Przy czym w tym przypadku – jest to system o podstawie 2 w którym używamy cyfr: 0 i 1.

- Przykładowo wartość 11001 w naturalnym systemie dwójkowym możemy wyrazić jako:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 16 + 8 + 0 + 0 + 1 = 25 \text{ (dziesiętnie)}$$

- Podobnie możemy wyrazić wartość ułamkową 10101.101 :

$$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 16 + 0 + 4 + 0 + 1 + 1/2 + 0 + 1/8 = 21.625 \text{ (dziesiętnie)}$$

Opanowanie istoty zapisu i konwersji danych pomiędzy różnymi systemami – **jest warunkiem zrozumienia działania układów cyfrowych i mikroprocesorowych.**

Dla rozstrzygnięcia niejednoznaczności zastosowanego systemu liczbowego – czasami z prawej strony zapisanej wartości podaje się symbol systemu zapisu – tzn. podstawę systemu i wtedy nie ma wątpliwości:

$(101.01)_{10}$ - oznacza wartość **dziesiętną**,

$(101.01)_2$ - oznacza wartość **dwójkową**

Uogólniając na możliwe przypadki – konwersji dziesiętnych liczb rzeczywistych na wartości naturalnego systemu dwójkowego przeprowadzamy w dwóch etapach:

- oddzielnie dokonujemy konwersji części całkowitej,
 - oddzielnie dokonujemy konwersji części ułamkowej,
- na końcu łączymy wyniki konwersji.

Przykład – liczbę dziesiętną 615.3125 zapisać w systemie dwójkowym:

- dokonujemy konwersji części całkowitej: $615 = 1001100111$,
- dokonujemy konwersji części ułamkowej: $0.3125 = 0.0101$,
- łączymy zapis: $(615.3125)_{10} = (1001100111.0101)_2$

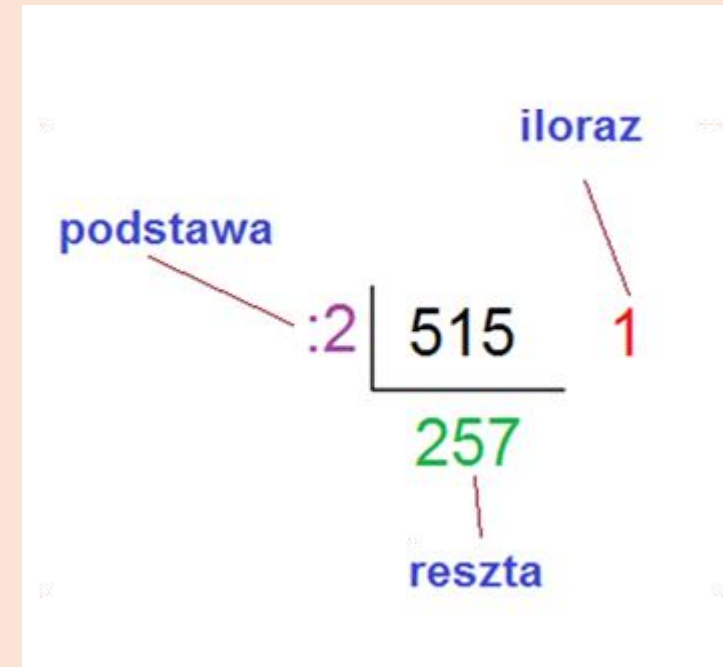
Wygodną, **mechaniczną** i łatwą w użyciu metodą konwersji liczb między systemem dziesiętnym a binarnym – jest **sekwencyjne dzielenie części całkowitej** przekształcaniej wartości przez podstawę docelowego systemu, oraz **mnożenie sekwencyjne części ułamkowej** przez podstawę nowego systemu.

Oczywiście dostępne są zasoby techniczne (w postaci komputerów czy kalkulatorów), które dokonują automatyczną konwersję danych pomiędzy różnymi systemami zapisu, zwłaszcza dla dużych wartości.

Przykład: chcemy przedstawić wartość dziesiętną 515.8125 w naturalnym kodzie binarnym.

Zaczynamy od konwersji części całkowitej :

- **Konwersja 515 do systemu NKB**
 - na początku bierzemy wartość poddawaną konwersji (515),
 - dzielimy tę wartość przez podstawę nowego systemu (2),
 - zapisujemy iloraz (257),
 - zapisujemy resztę z dzielenia (1)



Przykład c.d.: konwersja części całkowitej 515.8125 na NKB.

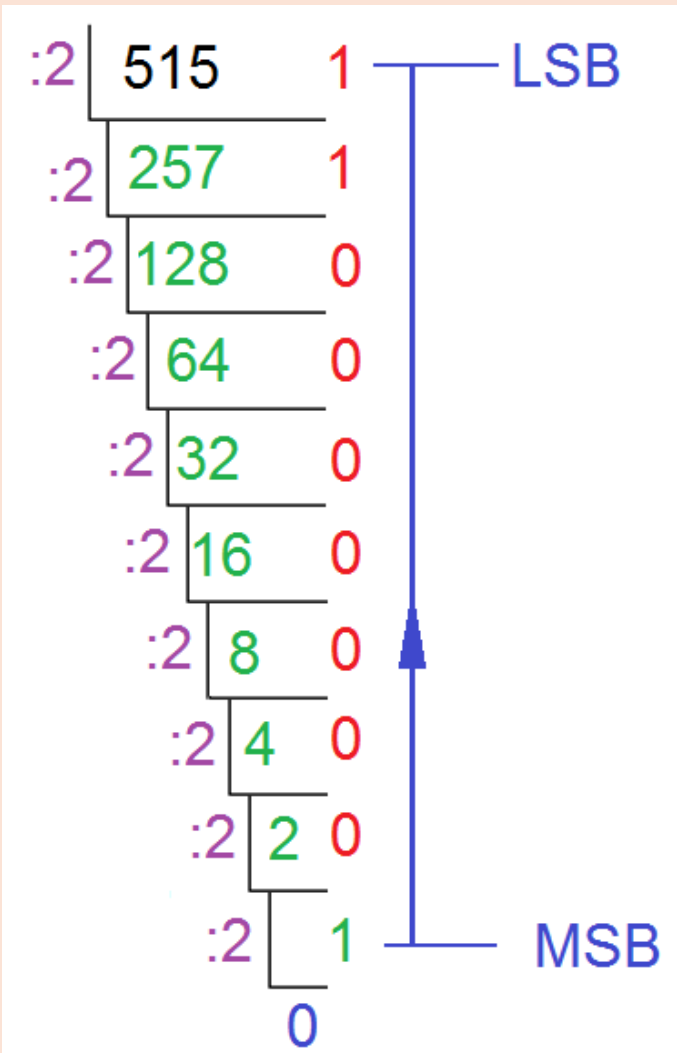
- **Konwersja 515 do systemu NKB**
 - otrzymany iloraz (257) ponownie dzielimy przez podstawę (2),
 - zapisujemy nowy iloraz (128),
 - zapisujemy nową resztę z dzielenia (1)

$$\begin{array}{r}
 :2 \overline{) 515} \quad 1 \\
 \underline{257} \\
 :2 \overline{) 257} \quad 1 \\
 \underline{128}
 \end{array}$$

Przykład c.d.: konwersja części całkowitej 515.8125 na NKB.

- **Konwersja 515 do systemu NKB**

- kontynuujemy podział przez 2 aż iloraz wyniesie zero,
- zapisujemy wynik od dołu do góry zaczynając o ostatniego niezerowego ilorazu i wpisując kolejne reszty,
- otrzymujemy wartość binarną: 1000000011.



Przykład c.d.: konwersja części ułamkowej wartości dziesiętnej 515.8125 na NKB.

- Konwersji dokonujemy mnożąc część ułamkową przez podstawę (2) oraz odrzucając części całkowite:
 - mnożymy przez 2 wyjściową część ułamkową (**0.8125**),
 - odrzucamy powstałą część całkowitą (**1**),
 - pozostałą część ułamkową iloczynu (**0.6250**) poddajemy dalszemu mnożeniu.

$$\begin{array}{r}
 0.8125 \\
 \times \quad 2 \\
 \hline
 1.6250
 \end{array}$$

odrzućana
wartość
całkowita — 1

mnożona
część
ułamkowa — 0.6250

Przykład c.d.: konwersja części ułamkowej wartości dziesiętnej 515.8125 na NKB.

- **Konwersji 0.8125 na NKB ...**
 - odrzucając pojawiające się wartości całkowite, kontynuujemy mnożenie każdej części ułamkowej przez podstawę.

$$\begin{array}{r}
 0.8125 \\
 \times \quad 2 \\
 \hline
 1 \quad .6250 \\
 \\
 0.6250 \\
 \times \quad 2 \\
 \hline
 1 \quad .2500 \\
 \\
 0.2500 \\
 \times \quad 2 \\
 \hline
 0 \quad .5000
 \end{array}$$

Przykład c.d.: konwersja dziesiętnej liczby 515.8125 na NKB – weryfikacja wyniku.

$$\begin{aligned}
 &(1000000011.1101)_2 = \\
 &= (1 \times 2^9 + 0 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \cdot (1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}) = \\
 &= 1 \times 512 + 0 \times 256 + 0 \times 128 + 0 \times 64 + 0 \times 32 + 0 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 \cdot 1 \times 1/2 + 1 \times 1/4 + 0 \times 1/8 + 1 \times 1/16 = \\
 &= (512 + 2 + 1) \cdot (0.5 + 0.25 + 0.0625) = \\
 &= (515.8125)_{10}
 \end{aligned}$$

- Naturalny system dwójkowy jest systemem podstawowym w układach cyfrowych.
- Aczkolwiek, trudno jest odczytywać długie ciągi liczb binarnych, gdyż nawet niewielka liczba dziesiętna staje się bardzo długą liczbą binarną:

$$11010100011011_2 = 13595_{10}$$

- Aby zapewnić zwartość i łatwość odczytu, wartości binarne są często wyrażane przy użyciu systemu **szesnastkowego** o podstawie 16 – czyli kodu **heksadecymalnego**.
- W tym systemie występuje 16 cyfr odpowiadających wartościom systemu dziesiętnego i 4-bitowego NKB w sposób następujący:

$$\mathbf{0}_{16} = 0_{10} = 0000_2, \mathbf{1}_{16} = 1_{10} = 0001_2, \mathbf{2}_{16} = 2_{10} = 0010_2, \mathbf{3}_{16} = 3_{10} = 0011_2,$$

$$\mathbf{4}_{16} = 4_{10} = 0100_2, \mathbf{5}_{16} = 5_{10} = 0101_2, \mathbf{6}_{16} = 6_{10} = 0110_2, \mathbf{7}_{16} = 7_{10} = 0111_2,$$

$$\mathbf{8}_{16} = 8_{10} = 1000_2, \mathbf{9}_{16} = 9_{10} = 1001_2, \mathbf{A}_{16} = 10_{10} = 1010_2, \mathbf{B}_{16} = 11_{10} = 1011_2,$$

$$\mathbf{C}_{16} = 12_{10} = 1100_2, \mathbf{D}_{16} = 13_{10} = 1101_2, \mathbf{E}_{16} = 14_{10} = 1110_2, \mathbf{F}_{16} = 15_{10} = 1111_2,$$

Konwersja z NKB na kod szesnastkowy / heksadecymalny polega na:

- pogrupowaniu liczby NKB w ciągi 4 bitowe zaczynając od LSB (prawej strony),

0011	0101	0001	1011
------	------	------	------

- zastąpieniu każdego ciągu odpowiadającą mu cyfrą heksadecymalną.

0011	0101	0001	1011
3	5	1	B

- zatem liczbie dziesiętnej **13595**₁₀ zapisanej kodem binarnym **0011010100011011**₂ odpowiada liczba szesnastkowa **351B**₁₆.

$$\mathbf{1}_{16} = 1_{10} = 0001_2, \mathbf{3}_{16} = 3_{10} = 0011_2, \mathbf{5}_{16} = 5_{10} = 0101_2, \mathbf{B}_{16} = 11_{10} = 1011_2,$$

Konwersja odwrotna – z kodu heksadecymalnego do kodu binarnego, polega na zastąpieniu każdej cyfry szesnastkowej jej 4-bitową reprezentacją w kodzie NKB. W przedstawiony sposób można zapisywać liczby rzeczywiste, dokonując oddzielnie konwersji części całkowitej i ułamkowej.

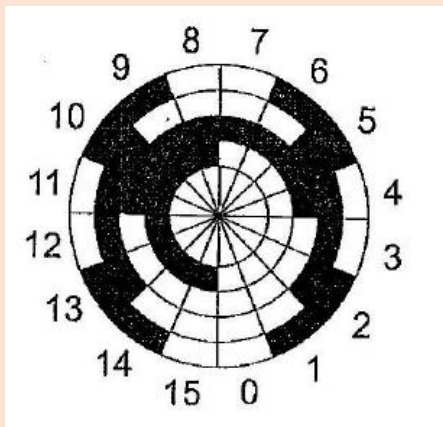
Kod Graya, dotyczy zapisu do liczb całkowitych, należy do **kodów bezpiecznych**, odpornych na zakłócenia. W tym kodzie, przy przechodzeniu do sąsiedniej wartości większej lub mniejszej – zmiana następuje tylko na jednej pozycji. Poniżej przedstawiono porównanie 4-bitowego kodu Graya z NKB i zapisem dziesiętnym.

System dziesiętny	Kod binarny	Kod Graya
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Konwersja wartości pomiędzy kodami Graya i NKB wykorzystuje jednobitową operację logiczną sumy modulo dwa. Tablicę działania tej funkcji dla jednobitowych danych wejściowych a i b przedstawia poniższa tabela:

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

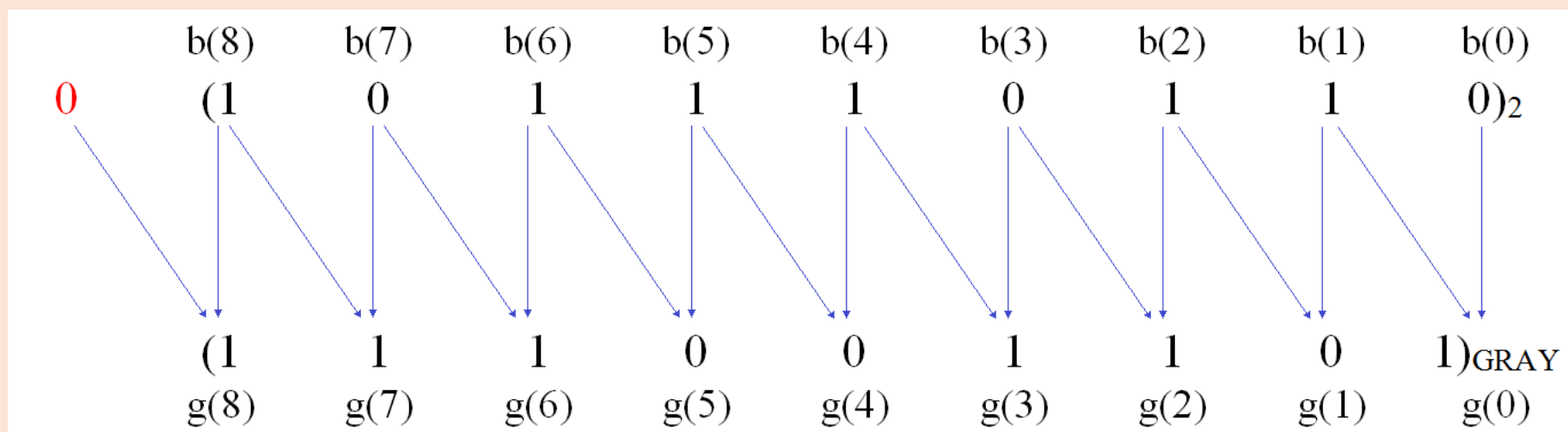
Funkcja przyjmuje wartość '1' dla nieparzystej liczby "jedynek" na wejściu.



Kod Graya ma zastosowanie w torach napędowych układów sterowania ze względu na odporność na zakłócenia, np. do kontroli kąta obrotu/położenia wału. - w postaci tarczy kodowej montowanej na osi wału.

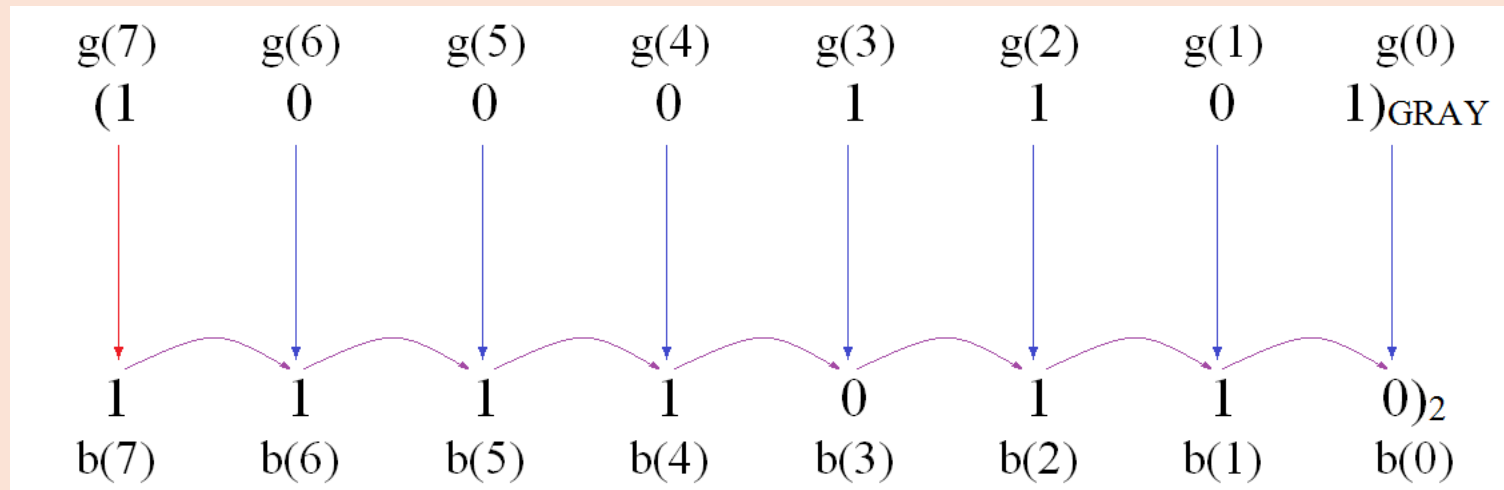
Konwersję wartości w NKB do wartości w kodzie Graya przeprowadzamy w następujący sposób:

- najmłodszy bit wyniku $g(0)$ wyznaczamy jako sumę modulo 2 bitów $b(0)$ i $b(1)$ liczby w NKB;
- kolejny bit $g(1)$ wyznaczamy jako sumę modulo 2 bitów $b(1)$ i $b(2)$ liczby NKB;
- kolejny bit wynikowy $g(2)$ wyznaczamy jako sumę modulo 2 bitów $b(2)$ i $b(3)$ liczby NKB;
- i.t.d. - czynność powtarzamy dla kolejnych bitów: $g(3)$, $g(4)$, ..., $g(7)$;
- najstarszy bit $g(8)$ wyznaczamy jako sumę modulo 2 najstarszego bitu NKB $b(8)$ z wartością **0**.



Konwersję wartości w kodzie Graya do wartości zapisanej w NKB przeprowadzamy w następujący sposób:

- najstarszy bit wynikowy $b(7)$ jest najstarszym bitem wejściowego kodu Graya $g(7)$;
- kolejny bit wynikowy $b(6)$ wyznaczamy jako sumę modulo 2 bitu $b(7)$ i $g(6)$;
- kolejny bit wynikowy $b(5)$ wyznaczamy jako sumę modulo 2 bitów $b(6)$ i $g(5)$;
- czynność powtarzamy dla kolejnych bitów: $b(4)$, $b(3)$, ..., $b(0)$.



Kody dwójkowo – dziesiętne powstają z systemu dziesiętnego w wyniku zastąpienia każdej cyfry dziesiętnej jej 4-bitowym odpowiednikiem w danym kodzie.

Konwersja kodu dwójkowo-dziesiętnego na system dziesiętny polega na wydzieleniu 4-bitowych ciągów w zapisie źródłowym i zastąpieniu ich odpowiednimi cyframi dziesiętnymi.

Popularnym przedstawicielem tej klasy kodów jest kod BCB (Binary Coded Decimal).

Przykład1:

liczbę dziesiętną $(2517)_{10}$ zastąpić kodem BCD:

$$(2517)_{10} = \quad 2 \quad \quad 5 \quad \quad 1 \quad \quad 7$$

$$(0010 \ 0101 \ 0001 \ 0111) = (0010010100010111)_{\text{BCD}}$$

Przykład2:

Liczbę $(10010111010100110001)_{\text{BCD}}$ przekształcić w zapis dziesiętny:

$$(1001 \ 0111 \ 0101 \ 0011 \ 0001)_{\text{BCD}}$$

$$= \quad 9 \quad \quad 7 \quad \quad 5 \quad \quad 3 \quad \quad 1 \quad = (97531)_{10}$$

Istnieje szereg innych kodów dwójkowo – dziesiętnych, znajdują one głównie zastosowanie w przekształcaniu informacji z postaci znakowej na informacyjną.

Przykładowe z nich przedstawiono w poniższej tabeli.

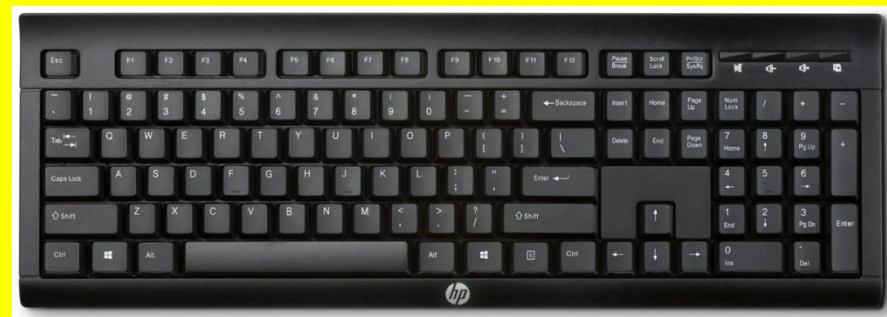
	BCD	Excess-3	BCD	BCD	IBM 1401
Wagi:	8421		2421	84-2-1	8421
0	0000	0011	0000	0000	1010
1	0001	0100	0001	0111	0001
2	0010	0101	0010	0110	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0100	0100
5	0101	1000	1011	1011	0101
6	0110	1001	1100	1010	0110
7	0111	1010	1101	1001	0111
8	1000	1011	1110	1000	1000
9	1001	1100	1111	1111	1001

Systemy cyfrowe operują na danych binarnych, człowiek historycznie przyzwyczał się do systemu dziesiętnego. Zatem, żeby ułatwić komunikację z urządzeniem cyfrowym (komputerem) opracowano tzw. kody znakowe.

Kody te w sposób binarny przedstawiają: cyfry, litery i inne symbole oraz specjalne znaki sterujące.

Jednym z takich kodów jest kod ASCII (American Standard Code for Information Interchange), znany również jako ISO-7. Jest to 7 bitowy zapis liter alfabetu łacińskiego oraz znaków specjalnych, jest on uzupełniany o bit parzystości na najstarszej pozycji – lub ta pozycja jest używana do wskazywania drugiej strony kodowej, która zawiera specyficzne znaki językowe. Znalazł on min. zastosowanie w kodowaniu klawiatury komputerowej, np.:

- klawisz Backspace (BS) => 0000 1000,
- klawisz Escape => 0001 1011,
- klawisz Spacja => 0010 0000,
- klawisz ‘#’ => 0010 0011,
- klawisz ‘6’ => 0011 0110,
- klawisz ‘K’ => 0100 1011,
- klawisz ‘m’ => 0110 1101,
- klawisz ‘+’ => 0010 1011, itd.



Dec Bin Hex Char				Dec Bin Hex Char				Dec Bin Hex Char				Dec Bin Hex Char			
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	\$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

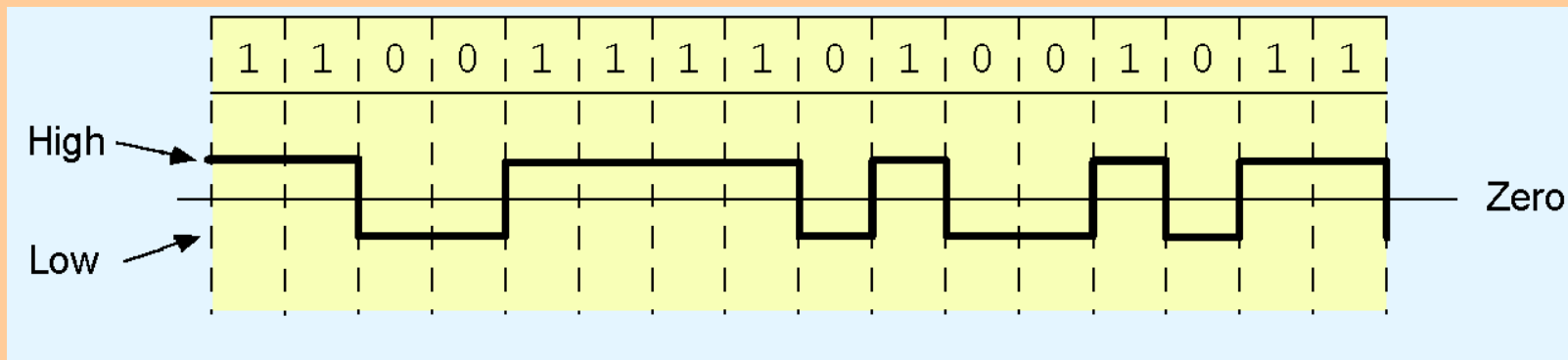
Współcześnie informacje cyfrowe przesyła się nie w sposób równoległy lecz szeregowy. Oznacza to, że np. 8 bitowa wartość binarna jest przekształcana w sygnał szeregowy i wysyłana bit po bicie, rozpoczynając od najstarszego bitu.

W trakcie szeregowej transmisji danych stosuje się specjalne kody znaków, bardziej odporne na zakłócenia i przekłamania. Przykładem takiego kodu jest kod NRZ (non-return-to-zero). W kodzie tym każdej zmianie na sąsiednich pozycjach wartości z '0' na '1' lub z '1' na '0' - odpowiada zmiana poziomu sygnału elektrycznego na przeciwny.

Np. szeregową transmisję słowa OK (w kodzie ASCII z bitem parzystości) wyglądałaby następująco:

0100 1111 (O)

0100 1011 (K)



Dotychczasowe nasze rozważania domyślnie koncentrowały się na zapisie liczb dodatnich w systemach dwójkowym lub szesnastkowym. W systemach cyfrowych operuje się zarówno na wartościach dodatnich jak i ujemnych - czyli liczbach ze znakiem. Do tego celu stosuje specjalne mechanizmy a długości przetwarzanych słów często uzupełnia się do długości jednego bajtu lub jego wielokrotności. Z samej wartości binarnej nie wynika znak tej liczby binarnej – jeżeli nie wiemy, w jakim systemie zapisu podana jest wartość.

Istnieją 3 systemy wyrażania liczb ze znakiem ze znakiem:

- system znak moduł – Signed Magnitude;
- system znak uzupełnienia do jedności (kod U1) – One's Complement,
- system znak uzupełnienia do dwóch (kod U2) - Two's Complement.



W systemie zapisu **znak moduł** wartości binarnych, **najstarszy bit** określa znak całej liczby:

- po bicie znaku występuje wartość bezwzględna liczby w NKB,
- jeżeli bit znaku wynosi '0' oznacza to liczbę dodatnią - jeżeli wartość '1' oznacza to liczbą ujemną,
- zatem liczby dodatnie posiadają identyczną interpretację jak w NKB. Przykłady:

"00000011" = $(+3)_{10}$,

- "10000011" = $(-3)_{10}$.

Jest to zrozumiały dla człowieka system lecz wymaga złożonych rozwiązań sprzętowych w obliczeniach komputerowych. Stąd chętniej stosuje się zapisy komplementarne, tzn. uzupełnienie do jedności / do dwóch.

Bit znaku	Bity danych						
1	0	0	0	1	1	0	1

W systemie zapisu **znak uzupełnienie do jedności**, zapis liczb dodatnich jest identyczny jak w systemie znak moduł. W przypadku liczb ujemnych – negujemy wszystkie bity komplementarnej liczby dodatniej zapisanej w systemie znak moduł, łącznie z bitem znaku.

Np. chcemy przedstawić liczbę $(-6)_{10}$ w postaci czterobitowej liczby binarnej systemu U1:

- komplementarną liczbą dodatnią jest $(+6)_{10}$ czyli $(0110)_2$,
- w liczbie komplementarnej bitem znaku jest wartość 0 na najstarszej pozycji,
- po zanegowaniu tej liczby otrzymujemy oczekiwaną wartość $(1001)_{U1}$.

Istnieją inne algorytmy konwersji ale nie będą rozpatrywane.

Kod U1	Wartość ₁₀	Kod U1	Wartość ₁₀	Kod U1	Wartość ₁₀	Kod U1	Wartość ₁₀
0000	0	0100	4	1000	-7	1100	-3
0001	1	0101	5	1001	-6	1101	-2
0010	2	0110	6	1010	-5	1110	-1
0011	3	0111	7	1011	-4	1111	0

W systemie zapisu **znak uzupełnienie do dwóch**, zapis liczb dodatnich jest identyczny jak w systemie znak moduł. Dla liczb ujemnych jest dopełnieniem do 2^n n-bitowego zapisu dodatniej wartości w systemie ZM:

- wyznaczamy binarną wartość 2^n – n jest liczbą bitów wartości dodatniej tej samej liczby,
- wyznaczamy dodatnią wartość przekształcanej liczby w systemie ZM,
- od wyznaczonej wartości 2^n odejmujemy wartość dodatnią tej samej liczby w systemie ZM,
- powstała różnica jest wartością ujemną konwertowanej liczby w systemie U2.

Np. chcemy przedstawić liczbę $(-5)_{10}$ w postaci liczby binarnej systemu U2:

- wartość ujemna $(-5)_{10}$ w systemie ZM wynosi 1101_{ZM} ,
- wartość dodatnia $(+5)_{10}$ w systemie ZM wynosi 0101_{ZM} ,
- liczba dodatnia w ZM posiada 4 bity – zatem $n = 4$,
- $2^n = 2^4 = 16_{10} = 10000_2$,
- odejmujemy: $10000_2 - 0101_{\text{ZM}} = 1011_{\text{U2}}$.

Kod U2	Wartość ₁₀	Kod U2	Wartość ₁₀	Kod U2	Wartość ₁₀	Kod U2	Wartość ₁₀
0000	0	0100	4	1000	-8	1100	-4
0001	1	0101	5	1001	-7	1101	-3
0010	2	0110	6	1010	-6	1110	-2
0011	3	0111	7	1011	-5	1111	-1

W systemie NKB dodawanie wykonujemy w podobny sposób jak w systemie dziesiętnym:

- zapisujemy obie liczby w taki sposób aby bity o tych samych wagach znajdowały się na identycznych pozycjach,
- dodajemy cyfry parami począwszy od najmniej znaczącej pozycji – od prawej strony,
- pojawiające się przeniesienia sumujemy przy dodawaniu następnej pary cyfr.

	1	1	1	1			1	1			
		1	0	1	1	1	.	0	1	1	0
+			1	0	1	0	.	0	1	1	1
	1	0	0	0	0	1	.	1	1	0	1

W systemie NKB odejmowanie wykonujemy w jak w systemie dziesiętnym:

- zapisujemy obie liczby w taki sposób aby bity o tych samych wagach znajdowały się na identycznych pozycjach,
- odejmujemy cyfry parami począwszy od najmniej znaczącej pozycji – od prawej strony,
- zaciągnięte pożyczki uwzględniamy przy odejmowaniu następnej pary cyfr.

			1	2	2	1	1	2		
1	1	0	1	1	0	0	0	.	1	1
-			1	1	0	0	1	.	1	0
1	0	1	1	1	1	1	1	.	0	1

W systemie NKB mnożenie wykonujemy w podobny sposób jak w systemie dziesiętnym:

- zapisujemy jedną liczbę pod drugą pomijając kropki dziesiętne,
- mnożymy jedną liczbę przez kolejne bity drugiej,
- sumujemy wyniki pośrednie,
- ustalamy położenie kropki dziesiętnej.

				1	0	1	0	0	.	0	1	
x						1	0	.	1	0	1	
				1	0	1	0	0	0	0	1	
						1	0	1	0	1		
				1	0	1	0	0	0	0	1	
			1	0	1	0	0	0	1			
+	1	0	1	0	0	0	1					
	1	1	0	1	0	1	0	0	1	0	1	
=	1	1	0	1	0	1	.	0	0	1	0	1

$$20,25 \times 2,625 = 53,15625$$

Warto zauważyć, że w szczególnym przypadku, mnożenie liczb binarnych przez potęgi wartości 2 – tzn. mnożenie przez 2 lub 4 lub 8 itd., polega na przesuwaniu mnożonej liczby o 1 lub 2 lub 3 itd. pozycje w lewo.

W systemie NKB dzielenie wykonujemy w prostszy sposób niż w systemie dziesiętnym:

- podobnie jak w systemie dziesiętnym zaczynamy od lewej strony,
- sprawdzamy czy dzielnik mieści się w odpowiednim ciągu dzielnej,
- jeżeli mieści się jako najstarszy bit ilorazu wpisujemy 1 jeżeli nie 0,
- do powstałej reszty dopisujemy kolejną cyfrę dzielnej,
- cykl powtarzamy do uzyskania zerowej reszty lub zakładanej dokładności (okresu reszty).

$$\begin{array}{r}
 \begin{array}{ccccccc}
 & & 1 & 0 & . & 1 & 1 & 0 & 0 & 1 & \dots \\
 \hline
 & 1 & 1 & 1 & 0 & : & 1 & 0 & 1 \\
 - & 1 & 0 & 1 & & & & & & & \\
 \hline
 & & 1 & 0 & 0 & & & & & & \\
 - & & 0 & 0 & 0 & & & & & & \\
 \hline
 & & 1 & 0 & 0 & 0 & & & & & \\
 - & & 1 & 0 & 1 & & & & & & \\
 \hline
 & & & 1 & 1 & 0 & & & & & \\
 - & & & 1 & 0 & 1 & & & & & \\
 \hline
 & & & & & 1 & 0 & & & & \\
 - & & & & & 0 & 0 & & & & \\
 \hline
 & & & & & 1 & 0 & 0 & & & \\
 - & & & & & 0 & 0 & 0 & & & \\
 \hline
 & & & & & 1 & 0 & 0 & 0 & & \\
 & & & & & & 1 & 0 & 1 & & \\
 \hline
 & & & & & & & 1 & 1 & &
 \end{array}
 \end{array}$$

Ze względu na znaczenie omówiony zostaną tylko operacje arytmetyczne tylko na danych w systemie U2. Opis operacji arytmetycznych na danych w zapisie ZM i U1 można znaleźć z przykładami w przesłanej literaturze. Operacje dodawania i odejmowania przeprowadza się na wszystkich bitach danych łącznie z bitem znaku – wynik otrzymuje się w zapisie U2. Jeżeli w wyniku operacji przed bitem znaku pojawi się bit przeniesienia – należy go zignorować.

Każde odejmowanie można sprowadzić do dodawania liczby ze znakiem przeciwnym – zatem, praktycznie wystarczy stosowanie tylko operacji dodawania.

Manualne operacje mnożenia i dzielenia na danych U2 są skomplikowane, ich opis można znaleźć w literaturze, najczęściej takie operacje są realizowane w mikroprocesorach lub specjalizowanych układach cyfrowych – dlatego nie są omawiane na tym wykładzie.

$$3_{10} + 3_{10}$$

$$\begin{array}{rcccc} & 0 & 0 & 1 & 1 \\ + & 0 & 0 & 1 & 1 \\ \hline & 0 & 1 & 1 & 0 \end{array}$$

$$6_{10}$$

$$5_{10} + (-4_{10})$$

$$\begin{array}{rcccc} & 0 & 1 & 0 & 1 \\ + & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 \end{array}$$

$$1_{10}$$

$$(-7_{10}) - (-6_{10})$$

$$\begin{array}{rcccc} & 1 & 0 & 0 & 1 \\ + & 0 & 1 & 1 & 0 \\ \hline & 1 & 1 & 1 & 1 \end{array}$$

$$(-1_{10})$$

Czym jest funkcja boolowska - jest wyrażeniem, które zawiera posiada:

- co najmniej jedną zmienną logiczną (A, B, ...) – może zawierać więcej,
- występuje w niej co najmniej jeden operator boolowski (NOT, AND, OR),
- co najmniej jedno wejście przyjmujące wartości 0 lub 1,
- wytwarza ona sygnał wyjściowy przyjmujący wartości 0 lub 1.

$$F(x, y, z) = x\bar{z} + y$$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z} + y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

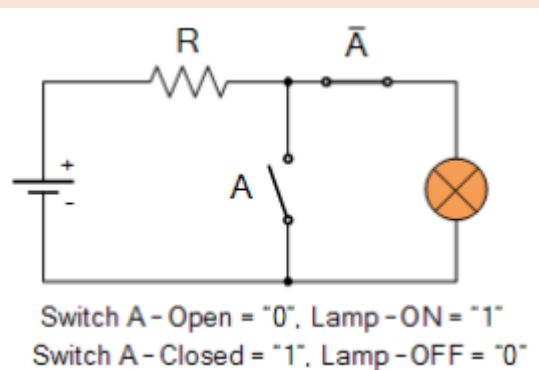
$$F(x, y, z) = x\bar{z} + y$$

Jest ona alternatywną formą opisu układu kombinacyjnego w postaci tablicy prawdy (Truth Table) – w której:

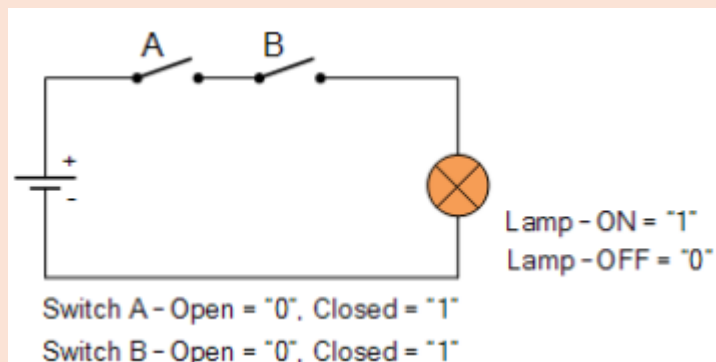
- podaje się wszystkie możliwe wartości sygnałów wejściowych (x,y,z),
- oraz odpowiadające im wartości sygnału/sygnałów wyjściowego,
- czasami podaje się wartości sygnałów cząstkowych, w celu ułatwienia opisu układu.

Alternatywną formą jest opis za pomocą połączeń kluczowych – tzw. logika przekaźnikowa:

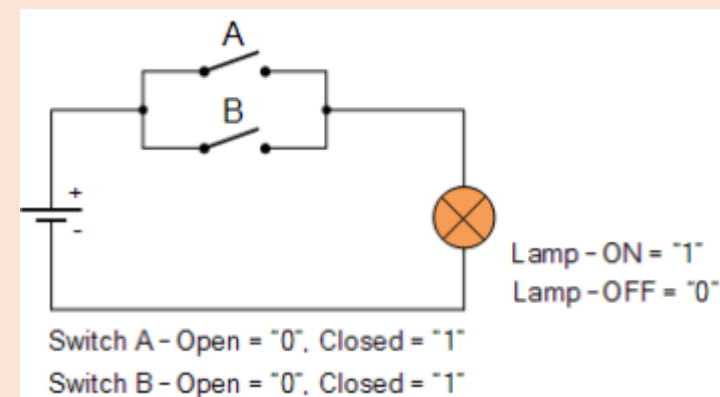
Logic NOT Function



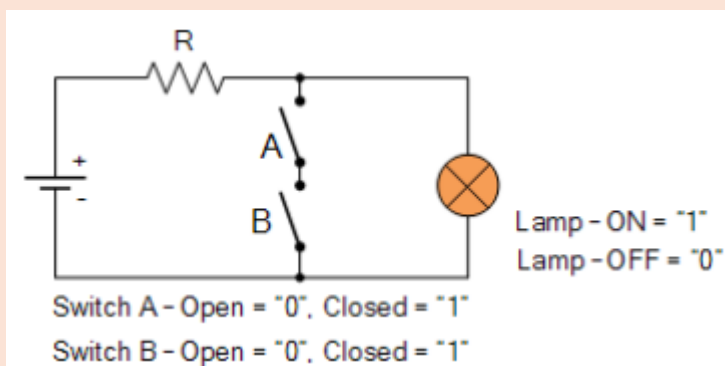
AND Function



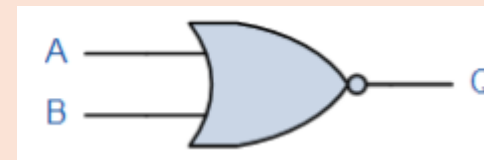
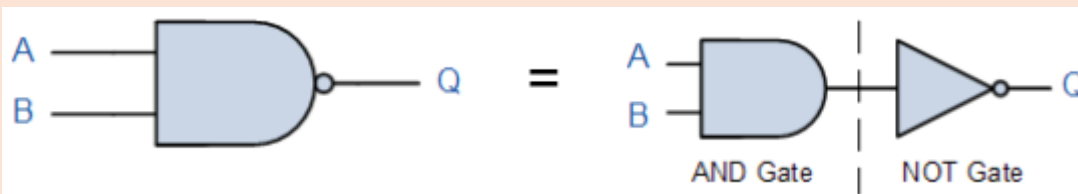
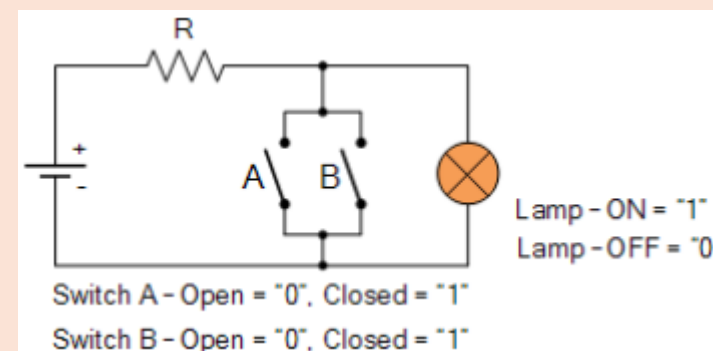
OR Function



NAND Function



NOR Function



Alternatywną formą operatorów logicznych jest opis za pomocą John Venna diagramów – teorii zbiorów:



Name		Low	Negation	Identity	High
Function		$f = 0$	$f = \bar{a}$	$f = a$	$f = 1$
Venn Diagram					
Graphic Symbol					Vcc

Name		0-Function	NOR		$\bar{b}$
Function		$f = 0$	$f = \overline{a+b}$	$f = a \cdot \bar{b}$	$f = \bar{b}$
Venn Diagram					
Graphic Symbol					

Name			$\bar{a}$	XOR	NAND
Function		$f = \bar{a} \cdot b$	$f = \bar{a}$	$f = a \oplus b$	$f = \overline{a \cdot b}$
Venn Diagram					
Graphic Symbol					

Name		AND	XNOR	a	
Function		$f = a \cdot b$	$f = \overline{a \oplus b}$	$f = a$	$f = a + \bar{b}$
Venn Diagram					
Graphic Symbol					

Name		b		OR	1-Function
Function		$f = b$	$f = \bar{a} + b$	$f = a + b$	$f = 1$
Venn Diagram					
Graphic Symbol					

T1. Prawo przemienności alternatywy i koniunkcji

- a) $a + b = b + a$
- b) $a \cdot b = b \cdot a$

T2. Prawo łączności alternatywy i koniunkcji

- a) $(a + b) + c = a + (b + c)$
- b) $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

T3. Rozdzielczość mnożenia względem dodawania

- a) $a + (b \cdot c) = (a + b) \cdot (a + c)$
- b) $a \cdot (b + c) = (a \cdot b) + (a \cdot c) = a \cdot b + a \cdot c$

T4. Prawa de Morgana – negacja sumy/iloczynu



$$\begin{aligned} \text{a)} \quad \overline{(a + b)} &= \bar{a} \cdot \bar{b} \\ \text{b)} \quad \overline{(a \cdot b)} &= \bar{a} + \bar{b} \end{aligned}$$

T5. Przekształcenia tożsamościowe

- a) $a + a = a$
- b) $a \cdot a = a$

T6. Prawo pochłaniania

- a) $a + (a \cdot b) = a$
- b) $a \cdot (a + b) = a$

T7. Prawo zerowania i ustawiania

- a) $0 + a = a, \quad 1 + a = 1$
- b) $0 \cdot a = 0, \quad 1 \cdot a = a$

T8. Prawo dopełniania

- a) $a + \backslash a = 1$
- b) $a \cdot \backslash a = 0$

Poznane prawa służą do upraszczania/minimalizacji funkcji logicznych – przykład:

$$\begin{aligned}
 &(X + Y) (X + \overline{Y}) (\overline{XZ}) \\
 &(X + Y) (X + \overline{Y}) (\overline{X} + Z) \\
 &(XX + X\overline{Y} + XY + Y\overline{Y}) (\overline{X} + Z) \\
 &((X + Y\overline{Y}) + X(Y + \overline{Y})) (\overline{X} + Z) \\
 &((X + 0) + X(1)) (\overline{X} + Z) \\
 &X(\overline{X} + Z) \\
 &X\overline{X} + XZ \\
 &0 + XZ \\
 &XZ
 \end{aligned}$$

Idempotent Law (Rewriting)

DeMorgan's Law

Distributive Law

Commutative & Distributive Laws

Inverse Law

Idempotent Law

Distributive Law

Inverse Law

Idempotent Law

Przykład: mamy system automatycznej klasyfikacji jakości piwa. Na stanowisku kontroli jakości mierzone są 3 parametry. Nazwijmy je:

A, B, C (% alkoholu w piwie, klarowność trunku, objętość).
Jeżeli określony parametr spełnia wymagania to przyjmuje wartość '1' – w przeciwnym przypadku wynosi '0'.

Wyrób jest kwalifikowany jako:

- bardzo dobry – jeżeli spełnia wszystkie parametry,
- jeżeli spełnia 2 parametry jest dobry,
- gdy osiąga 1 parametr – jest przeciętny,
- oraz jest wadliwy gdy nie spełnia żadnego.

Układ cyfrowy nie rozumie języka naturalnego: w tym słów: bardzo dobry, dobry, przeciętny lub wadliwy. Stąd trzeba te terminy zakodować najmniejszą liczbą bitów w sposób binarny np.: bardzo dobry = „11”, dobry = „10”, przeciętny „01” i wadliwy = „00”.

Zatem nasz system posiada trzy wejścia: A, B i C oraz dwubitowe wyjście Y1 i Y2 – określające jakość.

Tablica prawdy naszego układu jest następująca:

Wejścia			Wyjścia	
A	B	C	Y1	Y0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Istnieją dwie kanoniczne postacie zapisu funkcji boolowskich:

- kanoniczna postać sumacyjna – wyraża się ona w formie sum iloczynów operandów:

$$F(x, y, z) = xy + xz + yz$$

- kanoniczna postać iloczynowa – będąca iloczynem sum operandów:

$$F(x, y, z) = (x+y)(x+z)(y+z)$$

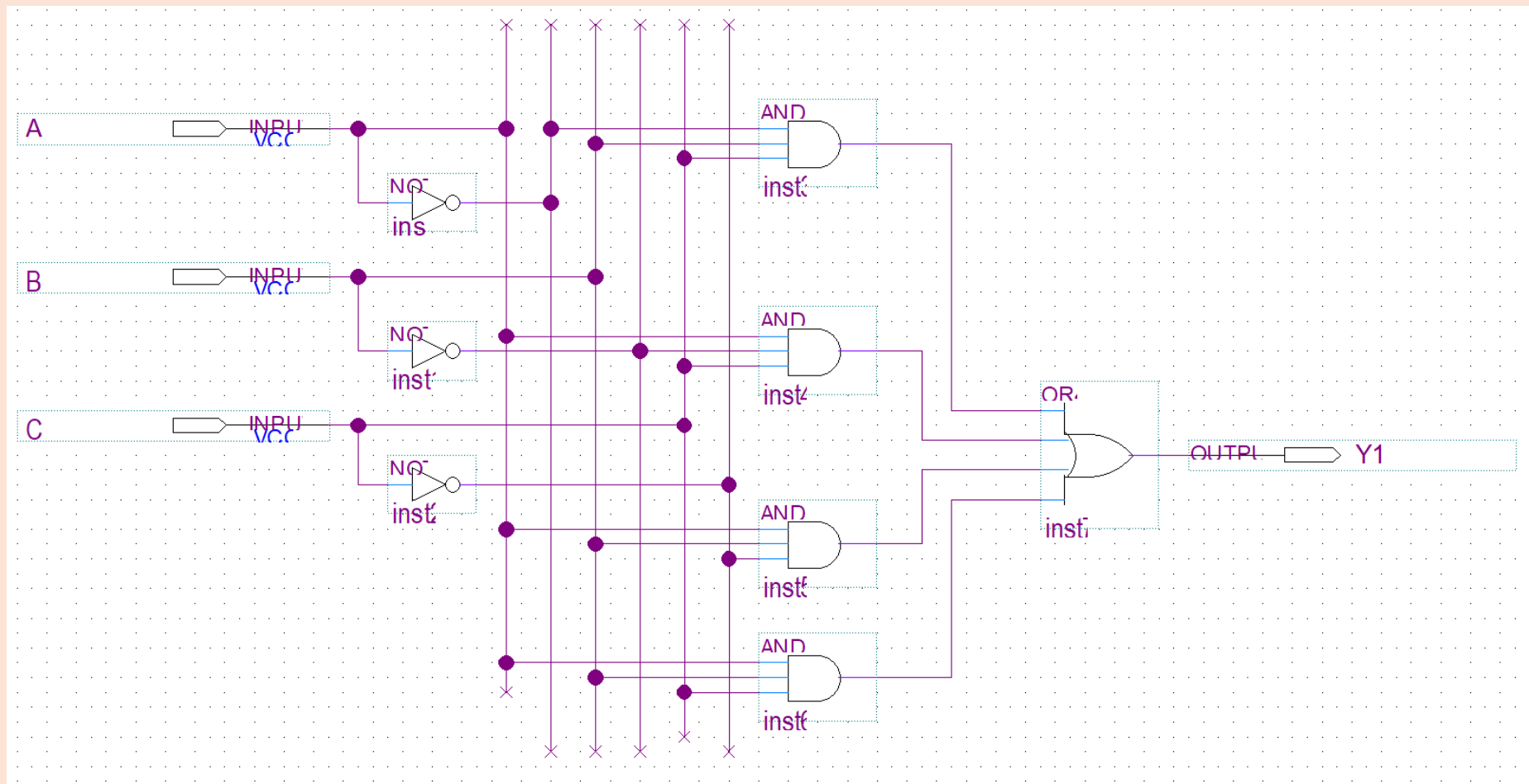
- obie postacie są równoważne, oznacza to, że układy zaprojektowane na podstawie postaci sumacyjnej i iloczynowej będą funkcjonowały identycznie.

Postacie kanoniczne stworzymy bezpośrednio z tabeli prawdy.

Utwórzmy kanoniczną **funkcję sumacyjną** dla wyjścia Y1:

- uwzględniamy tylko te przypadki gdy funkcja na wyjściu Y1 przyjmuje wartość 1,
- dla uwzględnianych przypadków tworzymy iloczyn wejść,
- jeżeli argument (A, B, C) przyjmuje wartość 0, to w funkcji wynikowej zapisujemy go w postaci zanegowanej,
- jeżeli dla rozpatrywanego przypadku, argument funkcji przyjmuje 1 – zapisujemy go w postaci prostej (np. B),
- tworzymy iloczyny składowe uwzględniając wszystkie argumenty (A, B, C),
- postać końcową funkcji przedstawiamy jako sumę iloczynów składowych.

Wejścia			Wyjścia	
A	B	C	Y1	Y0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$$Y1 = F(A, B, C) = \bar{A}BC + \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$$

Zaliczenie częściowe wykładu nr 2 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

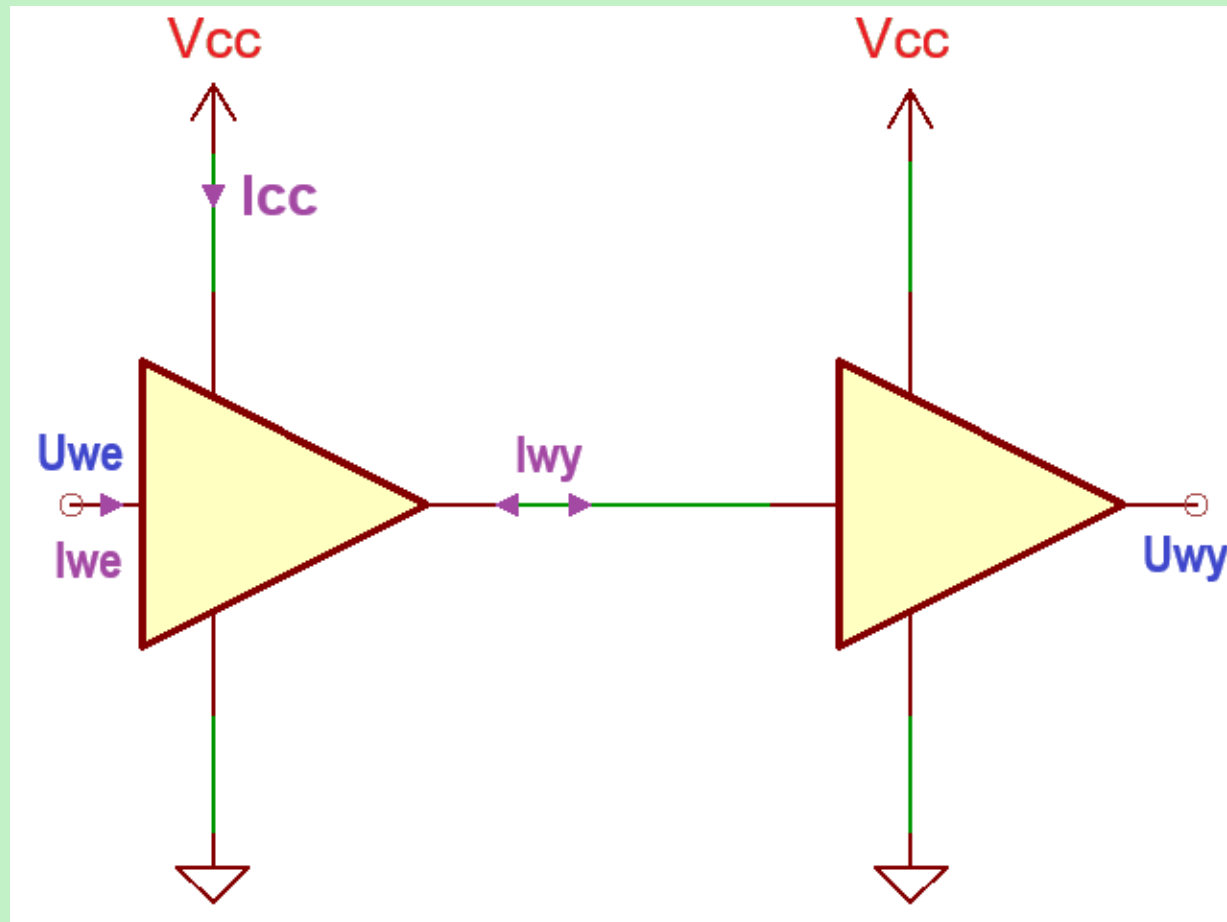
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

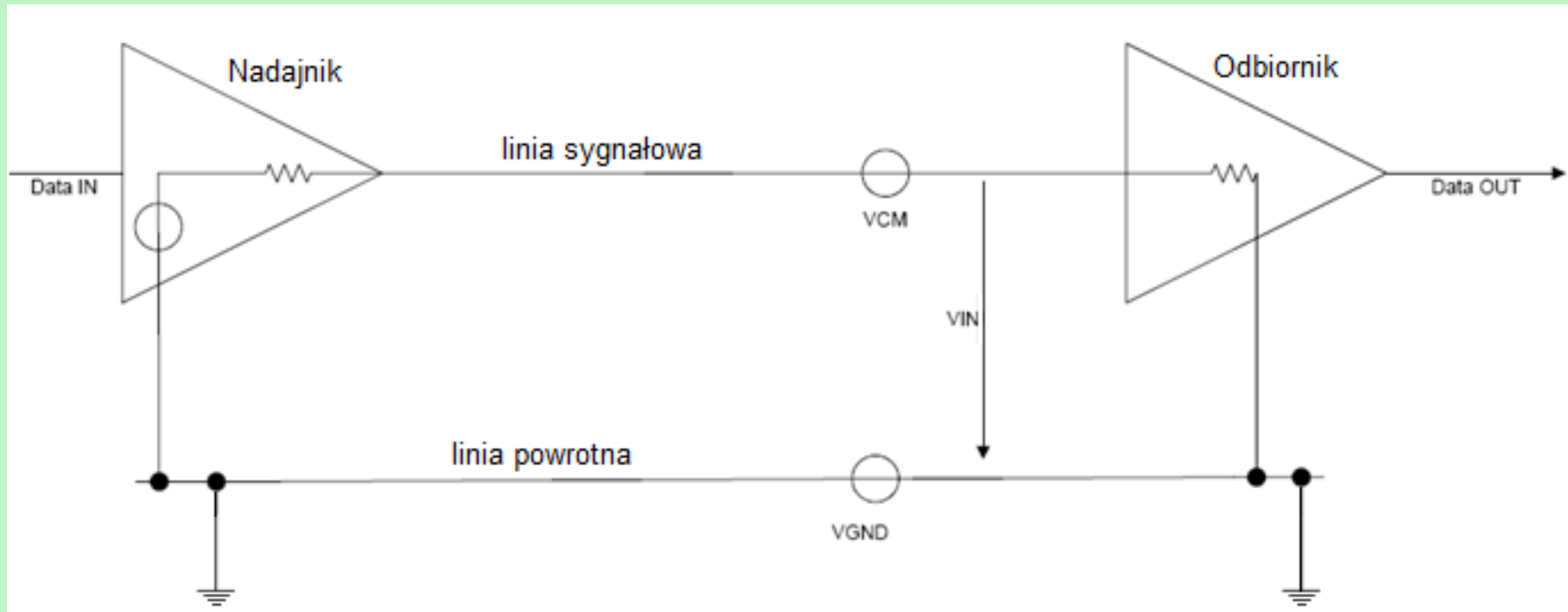
W systemach cyfrowych łączone są różne elementy i układy cyfrowe, które charakteryzują się różnymi wartościami napięć i prądów zasilających, różnymi wartościami napięć i prądów przetwarzanych sygnałów oraz różnymi charakterystykami czasowymi i częstotliwościowymi.



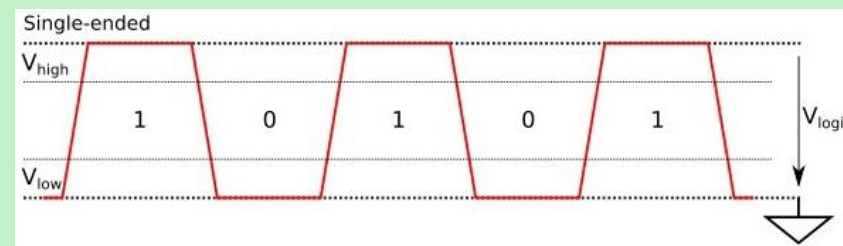
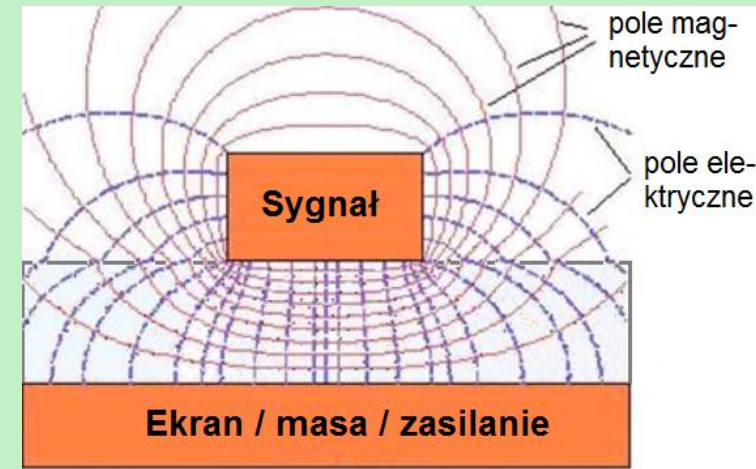
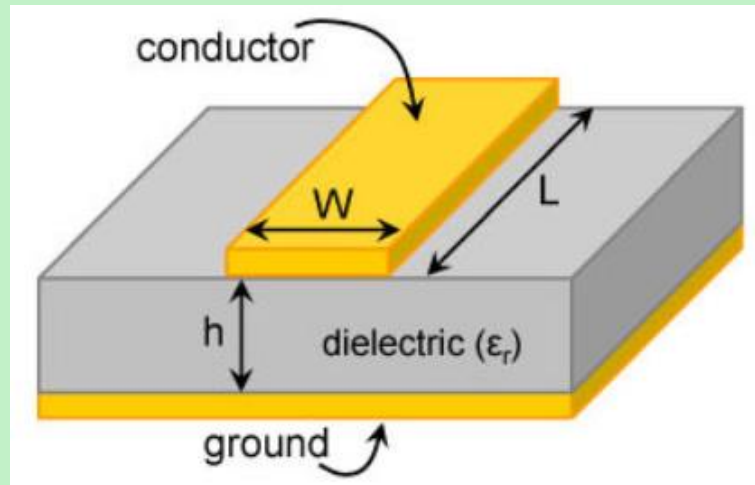
Podczas przesyłania przetwarzanych sygnałów stosowane są dwie metody:

- niesymetryczną (Single-Ended),
- symetryczną (differentia).

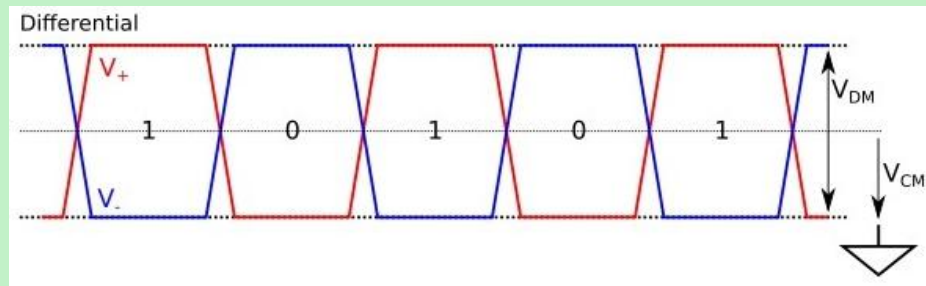
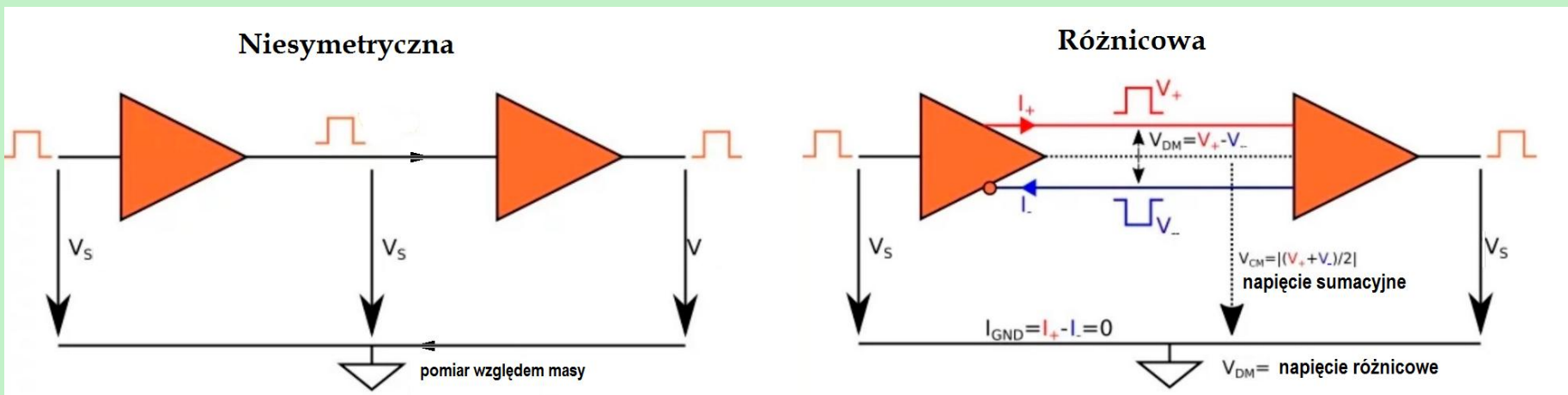
Jeżeli sygnał cyfrowy jest przesyłany jednym przewodem/linią/sygnałem radiowym, który jest mierzony względem sygnału odniesienia/masy – mówimy o transmisji niesymetrycznej.



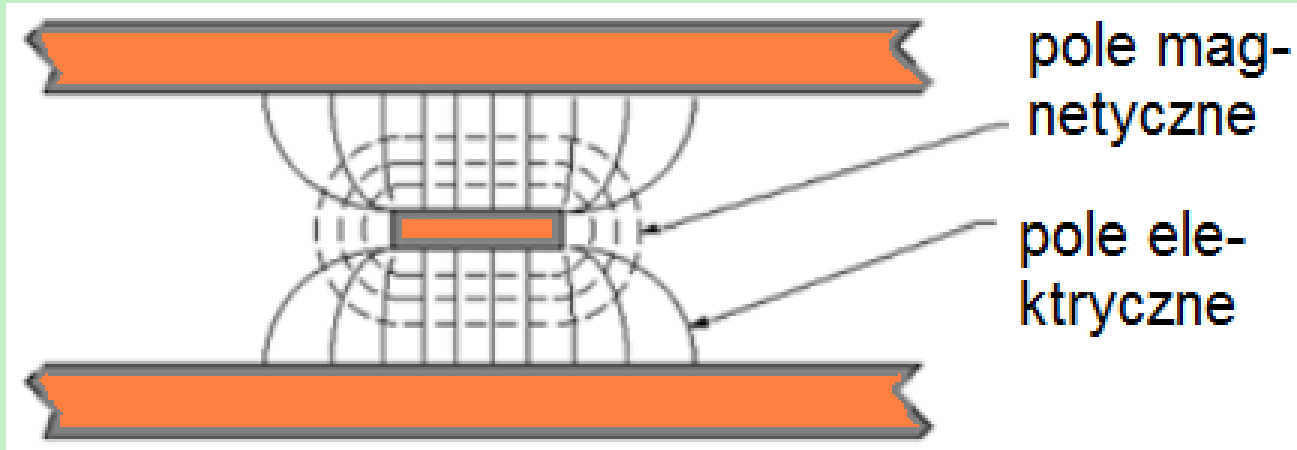
W przypadku transmisji niesymetrycznej, przesyłany sygnał nie jest ekranowany i wrażliwy na zakłócenia zewnętrzne. Typowym kanałem transmisyjnym jest linia mikropaskowa na płycie drukowanej/pcb.



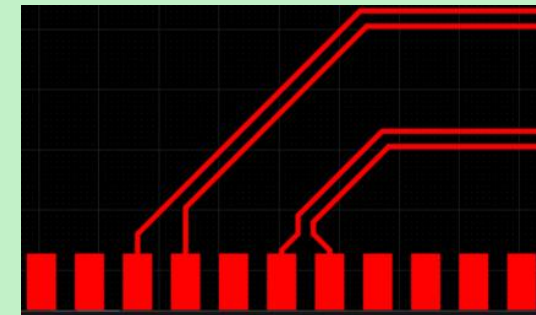
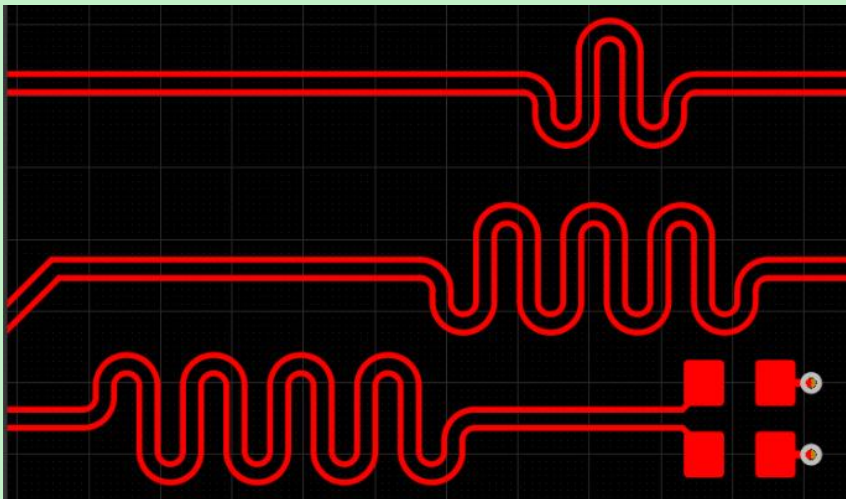
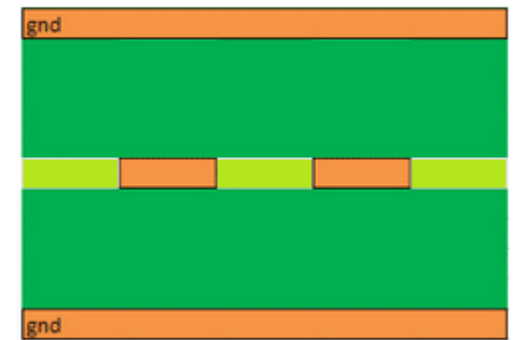
W transmisji różnicowej przetwarzany sygnał jest wysyłany dwoma kanałami/przewodami. W obu liniach jest przesyłany ten sam sygnał ale w przeciwnej fazie.



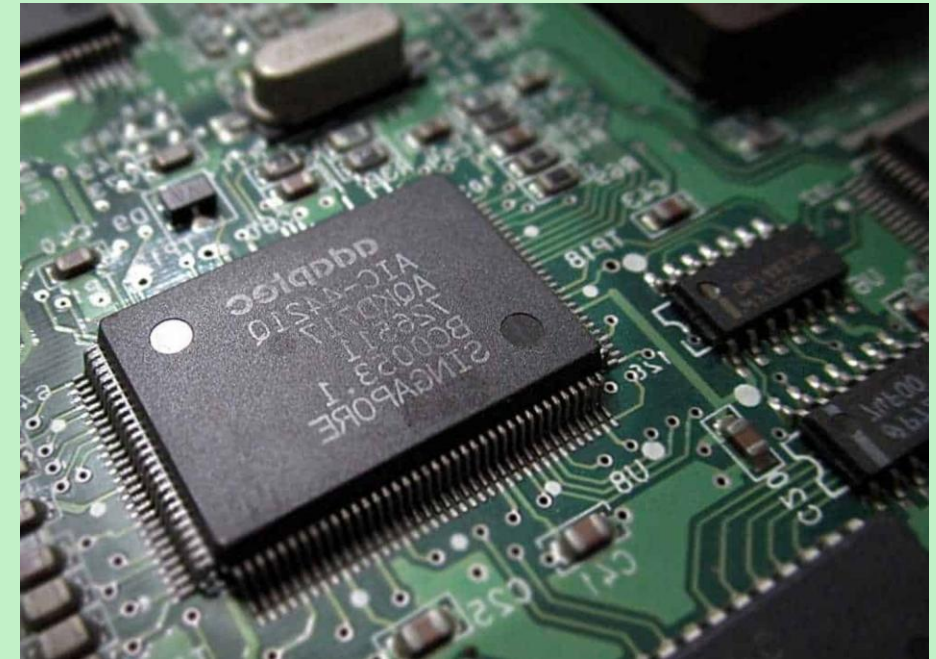
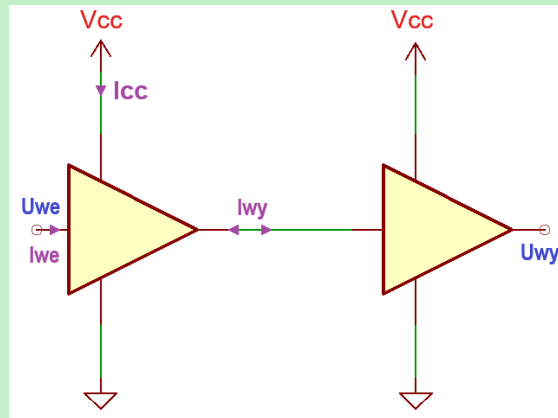
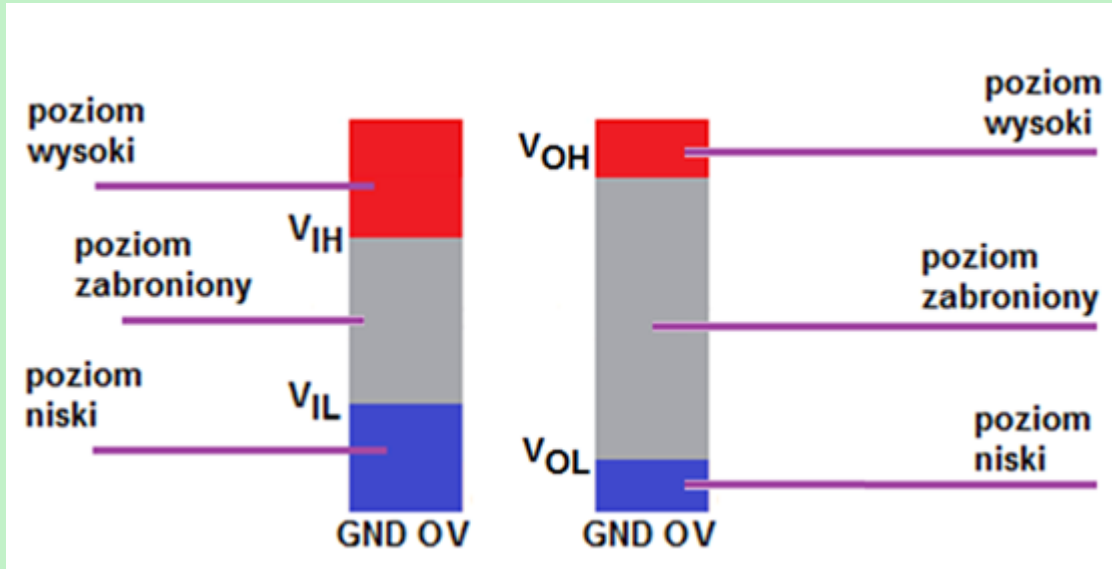
W transmisji różnicowej przetwarzany sygnał jest ekranowany i mniej wrażliwy na zakłócenia zewnętrzne. Typowym kanałem transmisyjnym sygnału różnicowego jest linia paskowa ekranowana.



Edge - Coupled Stripline

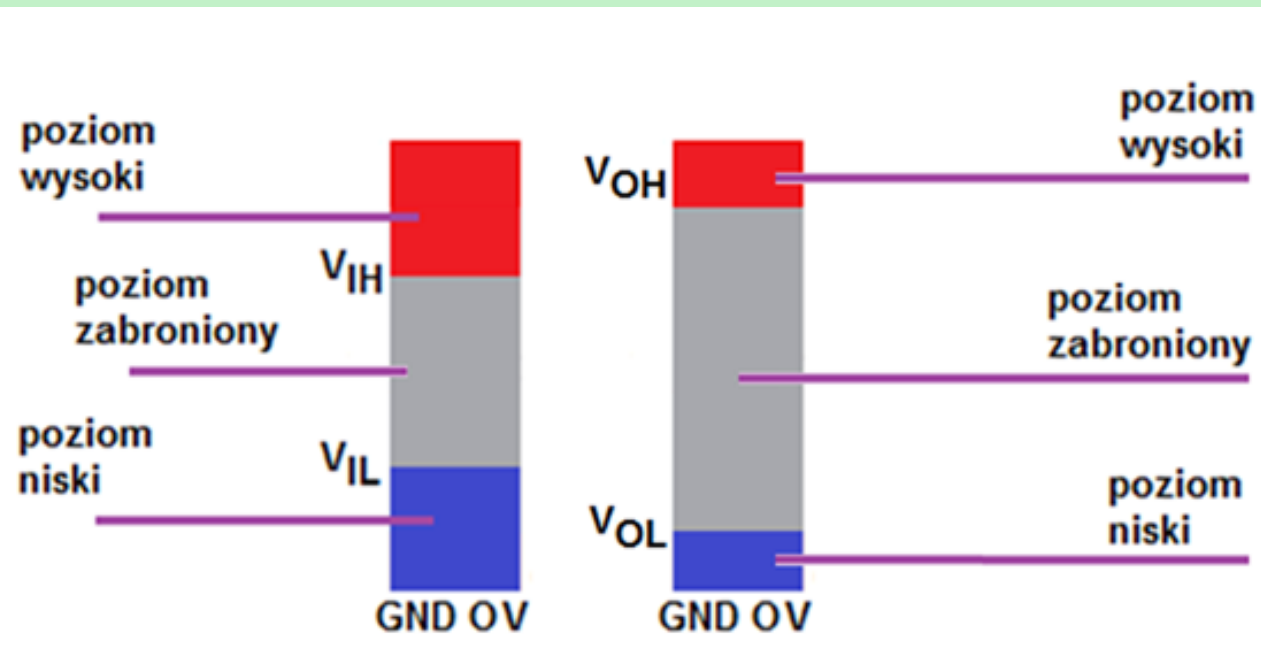


W cyfrowych układach elektronicznych pojęciom abstrakcyjnym: high/low, 0/1 lub on/off - przyporządkowane są konkretne wartości napięć, mówimy w tym przypadku o poziomach logicznych. Muszą one być zachowane, żeby przesyłanie sygnałów między dwoma układami przebiegało bez zakłóceń.



W transmisji niesymetrycznej zarówno poziomy sygnałów wejściowych i wyjściowych odnoszone są do masy układu, czyli poziomu 0V. Przy czym zakresy tych sygnałów na wejściu i wyjściu układu są różne. Interpretacja tych sygnałów jest następująca:

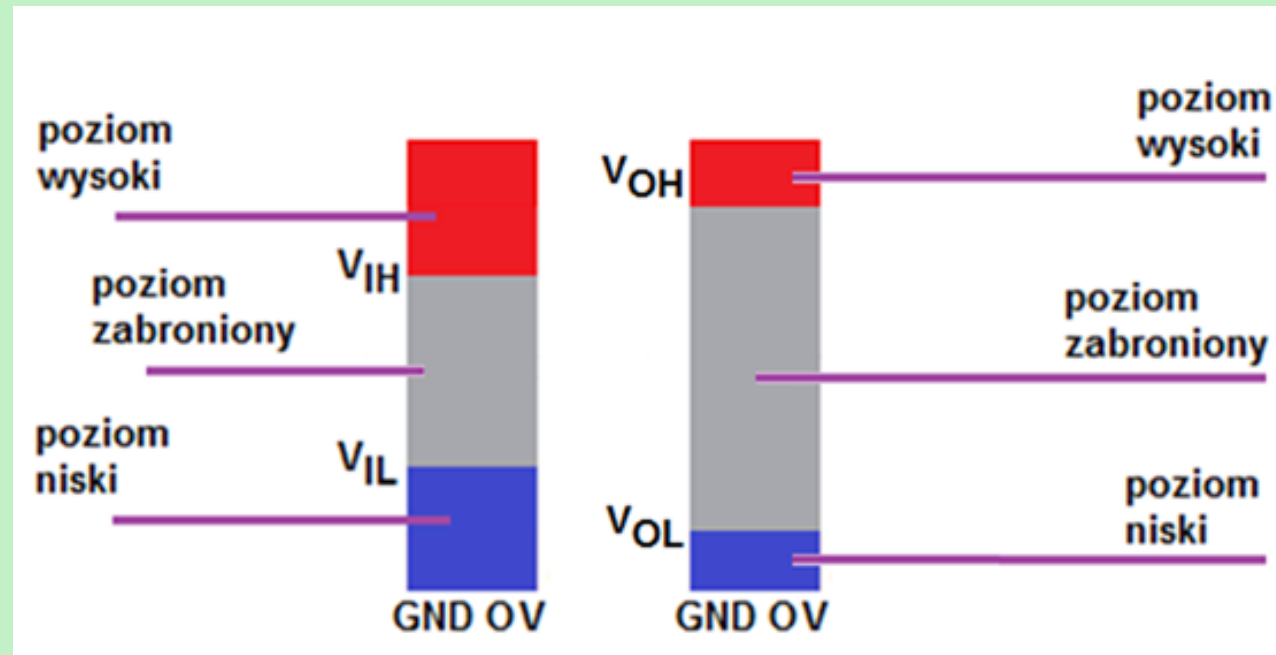
- jeżeli napięcie sygnału wejściowego mieści się w przedziale od 0V do V_{IL} (voltage input low) układ rozpoznaje ten poziom jako 0 na wejściu układu,
- jeżeli napięcie sygnału wejściowego mieści się w przedziale od V_{IH} (voltage input high) do wartości napięcia zasilania - układ rozpoznaje ten poziom jako 1 na wejściu układu,



Zabronione jest podawanie na wejście układu napięć z zakresu od V_{IL} do V_{IH} . Jest to tzw. zakres zabroniony, w którym układ cyfrowy nie będzie pracować poprawnie. Producent układu cyfrowego podaje w dokumentacji technicznej szczegółowe zakresy napięć wejściowych.

- jeżeli napięcie sygnału wyjściowego mieści się w przedziale od 0V do V_{OL} (voltage output low) jest to poziom 0 na wyjściu układu,
- jeżeli napięcie sygnału wyjściowego mieści się w przedziale od V_{OH} (voltage output high) do wartości napięcia zasilania – jest to 1 na wyjściu układu.

Logiczny poziom 0 na wyjściu układu mieści się w mniejszym przedziale napięć niż 0 na wejściu układu, podobnie 1 na wyjściu układu dotyczy wyższych napięć niż 1 na wejściu. Wynika to z konieczności dopasowania poziomów w procesie łączenia układów, gdyż linie łączące wprowadzają dodatkowe spadki napięć.

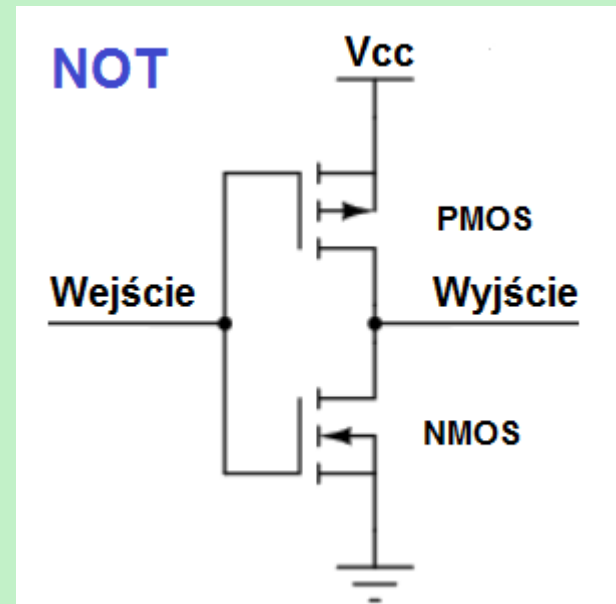


Ile woltów wynosi 1 i 0 na wejściu układu cyfrowego i ile wynosi 1 i 0 na wyjściu układu cyfrowego?

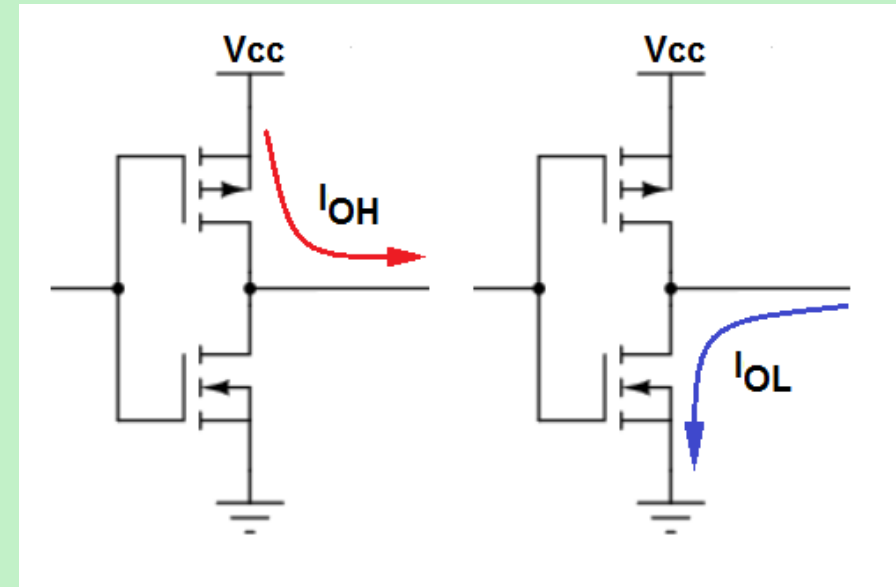
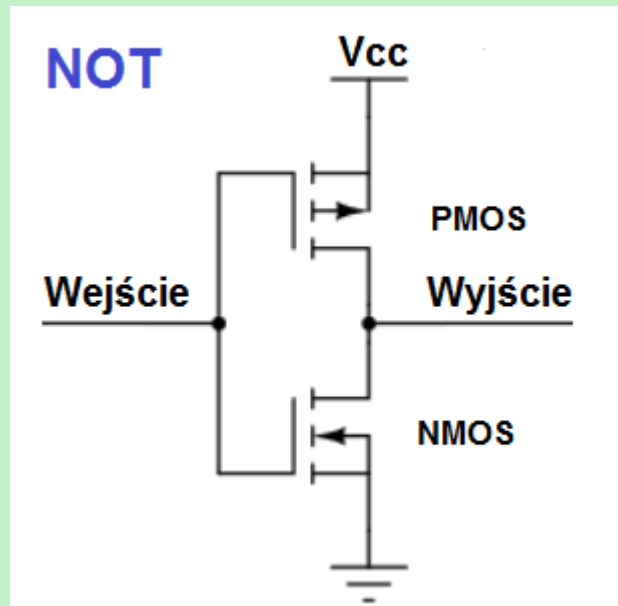
Na to pytanie nie istnieje jednoznaczna odpowiedź, gdyż w praktyce funkcjonuje wiele standardów napięciowych – o których będzie mowa w dalszej części wykładu.

Zanim przejdziemy do przeglądu typowych standardów sygnałowych, przyjrzyjmy się konstrukcji obwodów wejściowych i wyjściowych elementów i układów cyfrowych.

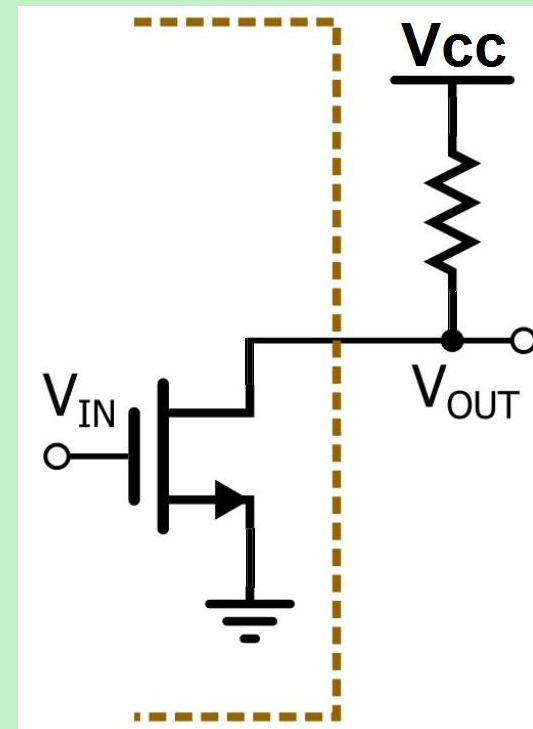
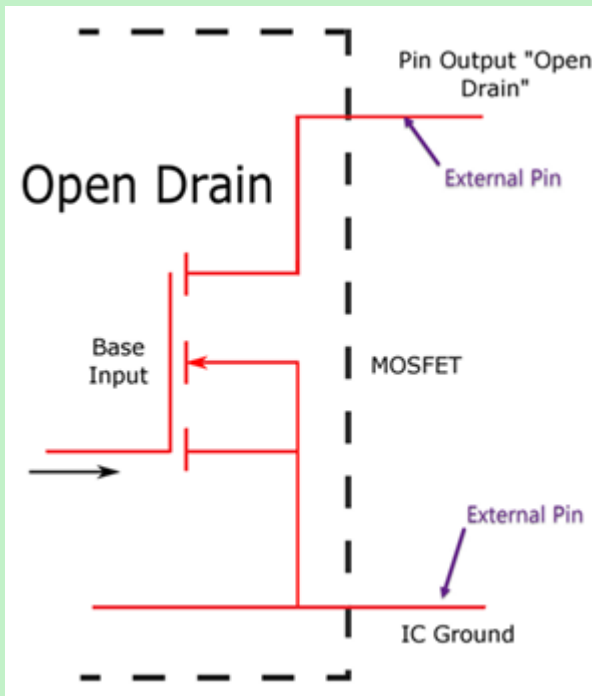
Przykładową konstrukcję typowej współczesnej bramki NOT pokazano na poniższym rysunku.



Jest to tzw. układ przeciwsobny (totem-pole) wyjścia układu cyfrowego, w którym uczestniczą dwa tranzystory przeciwnego typu. Należy zauważyć, że w danej sytuacji pracuje tylko jeden z tranzystorów. W stanie wysokim na wyjściu – prąd wypływa z układu I_{OH} , a w stanie niskim na wyjściu wpływa do układu I_{OL} . W przypadku tego typu wyjść, można je łączyć bezpośrednio z wejściami innych układów, które są wykonane w tym samym standardzie sygnałowym, w tej sytuacji nie są wymagane dodatkowe elementy dopasowujące.



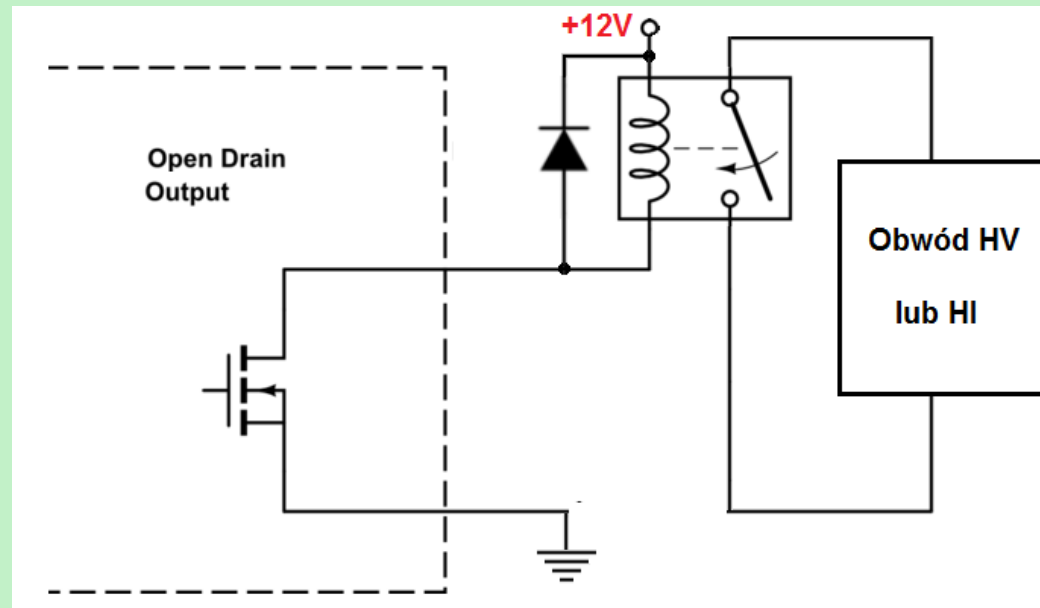
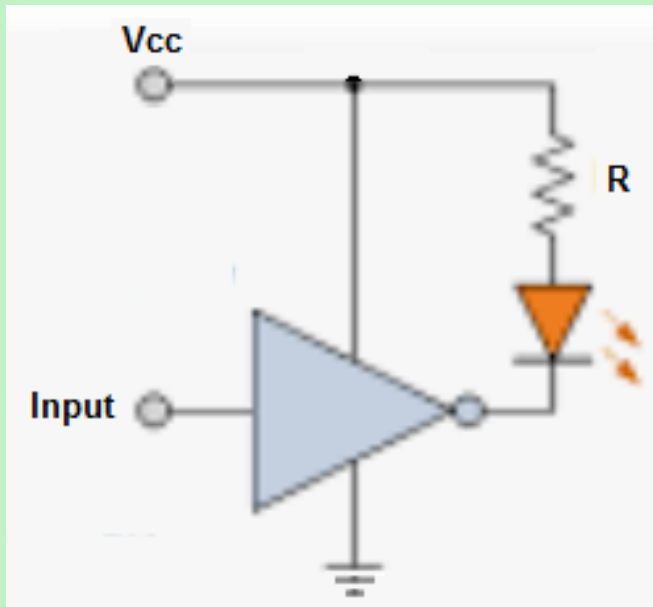
Innym rodzajem wyjścia układu cyfrowego, może być wyjście typu otwarty dren (open-drain). Cechą charakterystyczną tej konstrukcji, jest wyjście bezpośrednio z drenu tylko jednego tranzystora końcowego. Takiego układu nie można bezpośrednio połączyć z wejściem innego układu, z powodu braku zasilania drenu tranzystora. Dlatego pomiędzy wyjście a zasilanie V_{CC} układu należy włączyć element, który zapewni przepływ prądu – taki element może on stanowić jednocześnie obciążenie wyjścia układu cyfrowego.



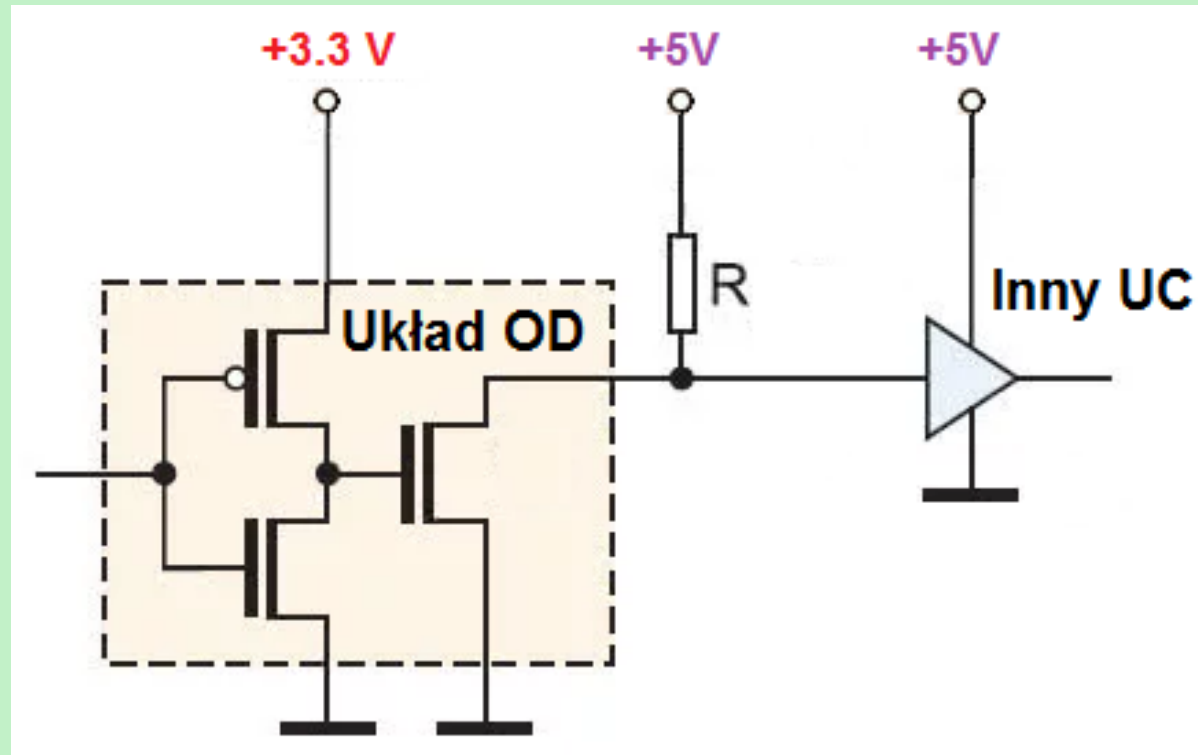
W układach z wyjściem typu otwarty dren - obciążenie można włączyć bezpośrednio w obwód wyjściowy:

- np. diodę LED – oczywiście ograniczając jej prąd rezystorem R,
- lub cewkę przekaźnika/stycznika obsługującego obwód wysokonapięciowy lub prądowy,
- w układzie otwarty dren napięcie polaryzujące może być wyższe niż napięcie zasilania układu cyfrowego.

Z samego symbolu układu cyfrowego (np. bramki NOT) nie wynika jego wyjście, co można znaleźć w katalogu.

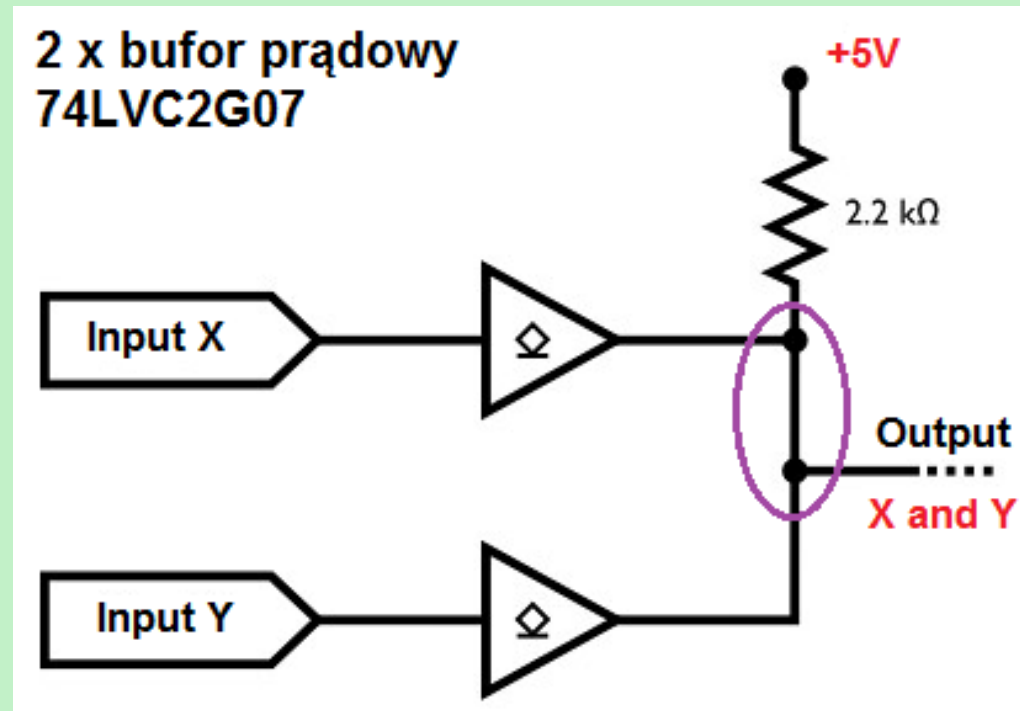


Tranzystor wyjściowy w układzie OD można zasilać napięciem wyższym niż zasilanie układu cyfrowego, w którym jest on wbudowany. Może to być np. napięcie +5V, co oznacza, że następne układy zasilane napięciem wyższym będą obsługiwane z układu o niższym napięciu (w innym standardzie sygnałowym), np. +3.3 V. Ten efekt pozwala płynnie dopasowywać sygnały cyfrowe między układami o różnych standardach.

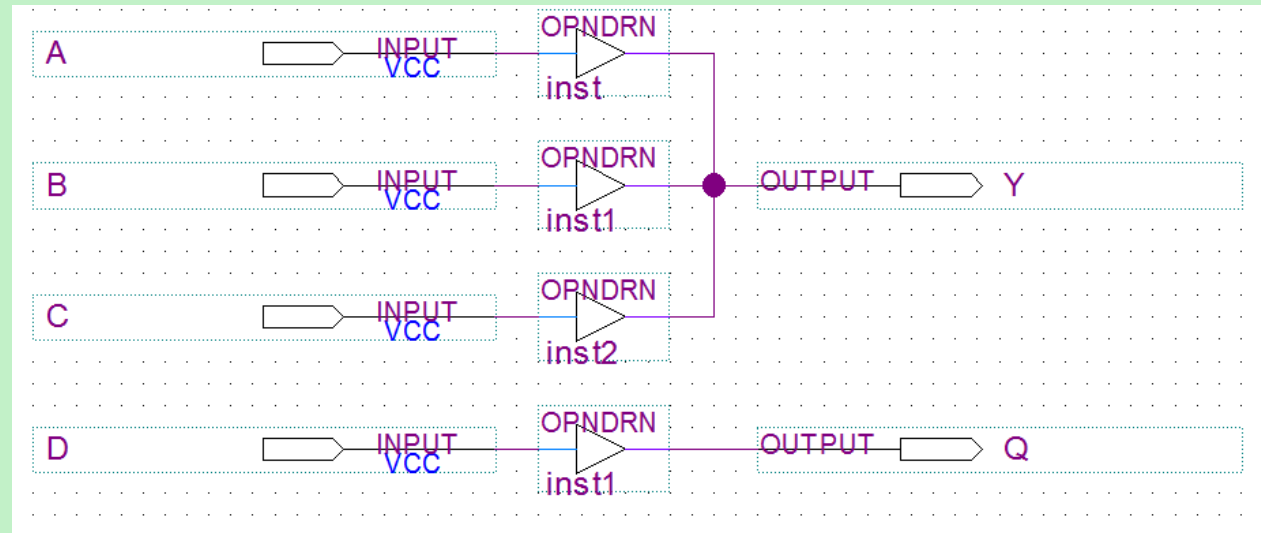


W odróżnieniu od wyjść totem pole, wyjścia układów typu otwarty dren łączyć ze sobą do wspólnego rezystora polaryzującego. W miejscu połączenia tworzy się tzw. iloczyn montażowy, tzn. funkcja AND dla sygnałów wszystkich połączonych ze sobą wyjść.

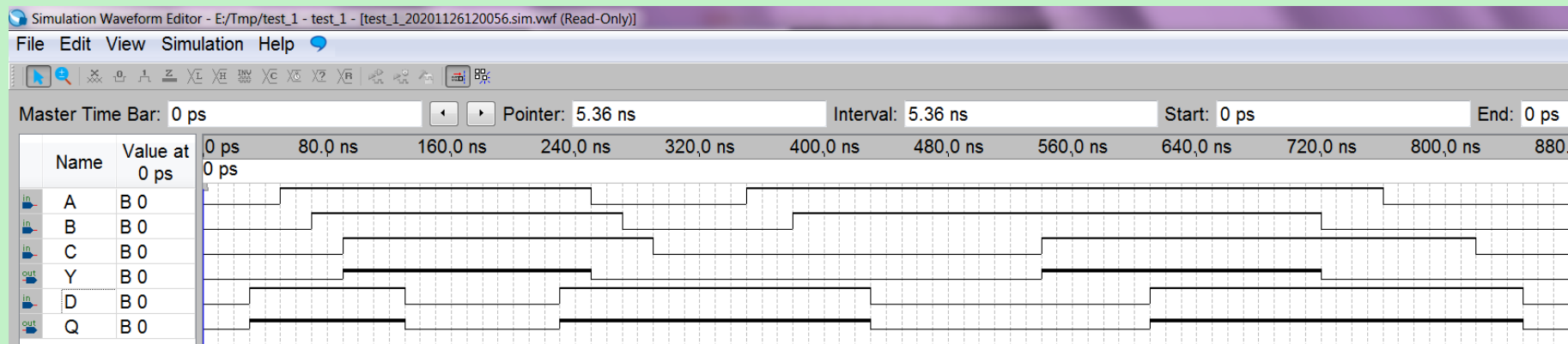
Poniżej pokazano przykład dwóch wzmacniaczy/buforów prądowych typu OC dołączonych do rezystora 2k2. Są to układy, które nie zmieniają poziomów sygnałów wejściowych ale zapewniają wyższe wartości prądów wyjściowych, w przedmiotowym przypadku do 32 mA – jest to tzw. wyjście mocy (nawet około 100 mA).



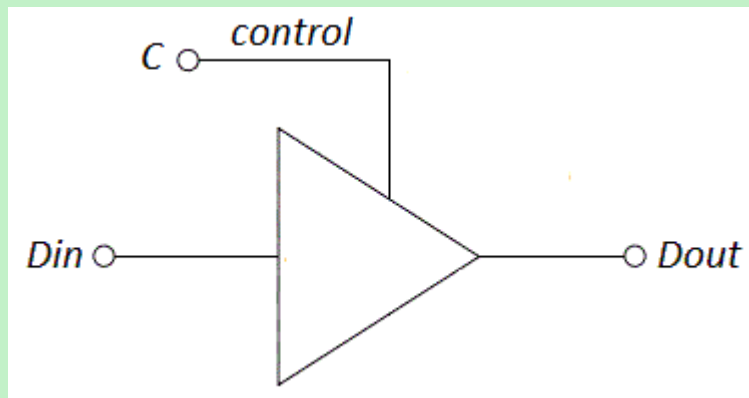
Wyjście „open drain” można ustawiać dyrektywą kompilatora **opndrn** na schemacie:



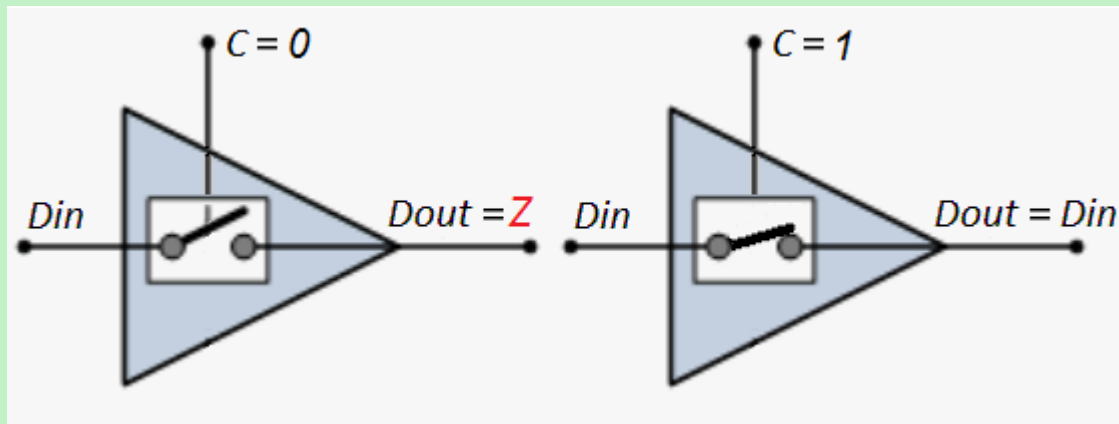
Dyrektywa musi być umieszczona na poziomie pliku top-level.



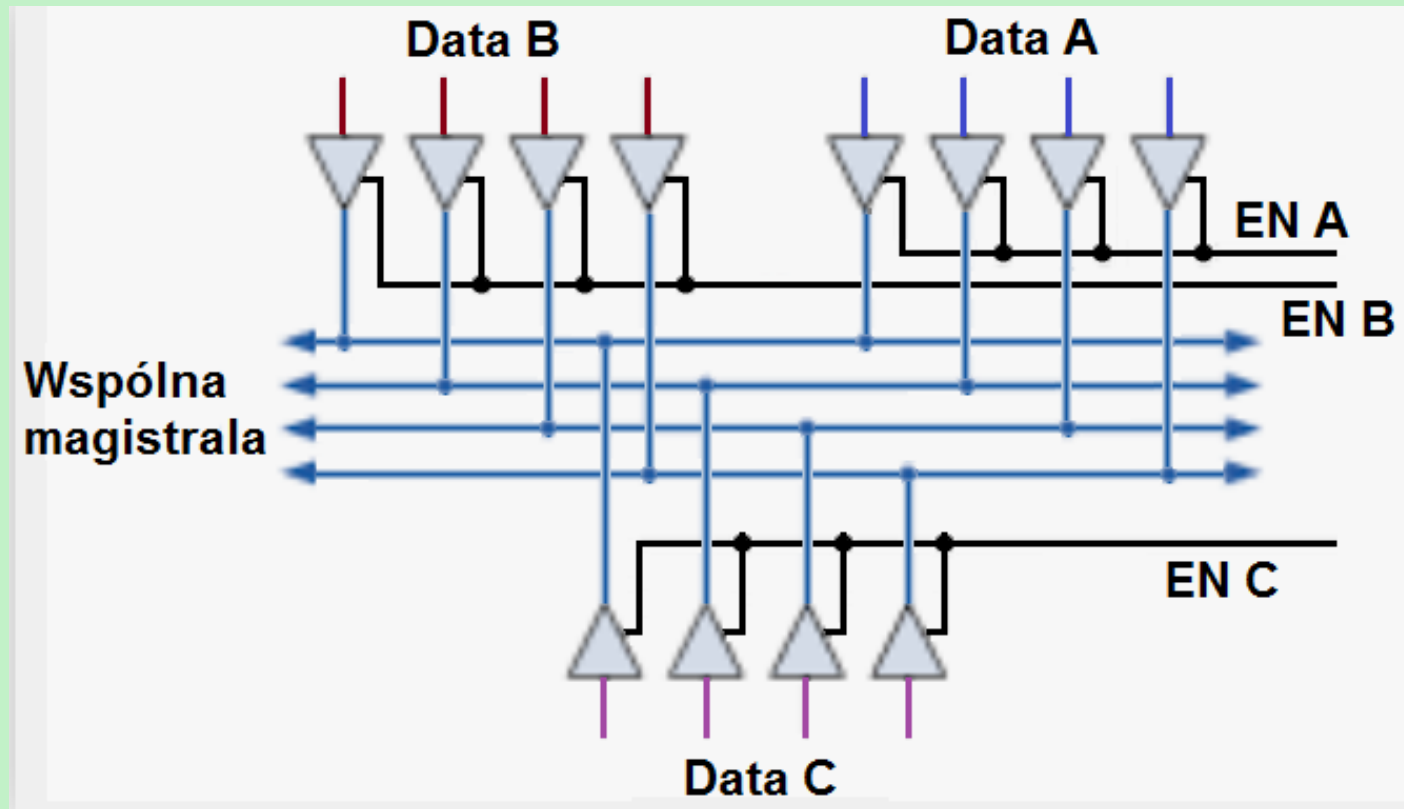
Jeszcze innym typem wyjścia jest wyjście trójstanowe (tri state), które może przyjmować 3 poziomy wyjściowe: wysoki, niski nietypowy oraz trzeci stan – tzw. stan wysokiej impedancji Z, który nie występuje w innych typach wyjść. W stanie Z – z układu praktycznie nie wypływa ani nie wpływa prąd, co pozwala łączyć wyjścia trójstanowe ze sobą – kontrolując za pomocą sygnału C ich włączanie.



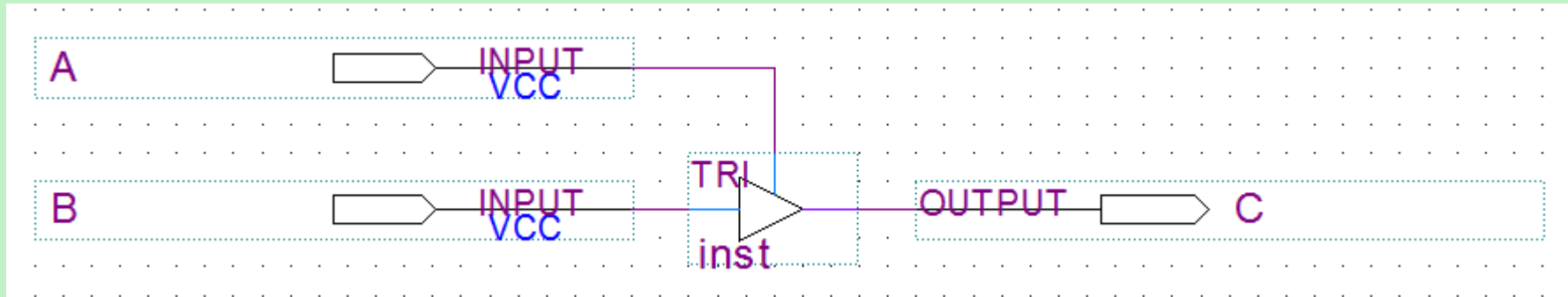
Truth table		
C	Din	$Dout$
0	0	High - Z
0	1	High - Z
1	0	0
1	1	1



Wyjścia trójstanowe wykorzystywane są min. do multipleksowania urządzeń korzystających ze wspólnej magistrali jak na rysunku. W danej chwili tylko jedno urządzenie (A, B lub C) przesyła dane z użyciem magistrali podczas gdy bufony innych pozostają w stanie wysokiej impedancji.



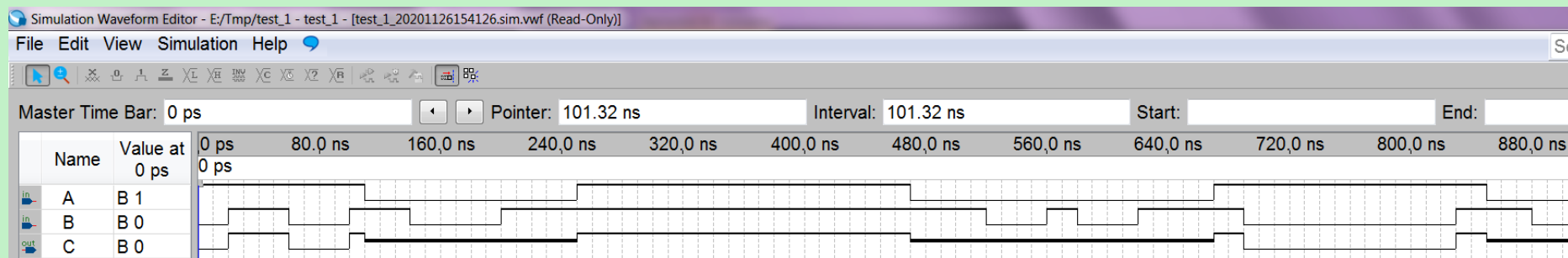
Wyjście tri state można ustawiać na poziomie schematu pliku top-level projektowanego układu:



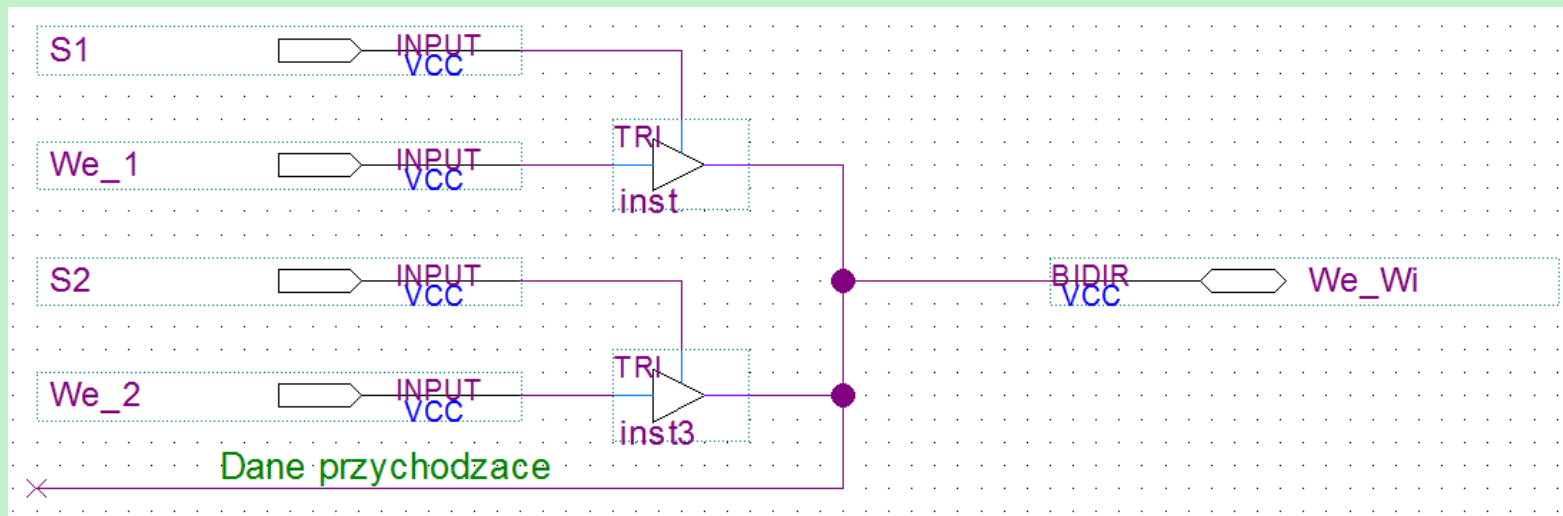
W przypadku układu opisanego w VHDL za pomocą np. instrukcji:

```
2 |  
3 C <= B when (A = '1') else 'Z';  
4
```

Niżej przedstawiono wyniki symulacji funkcjonalnej układu:



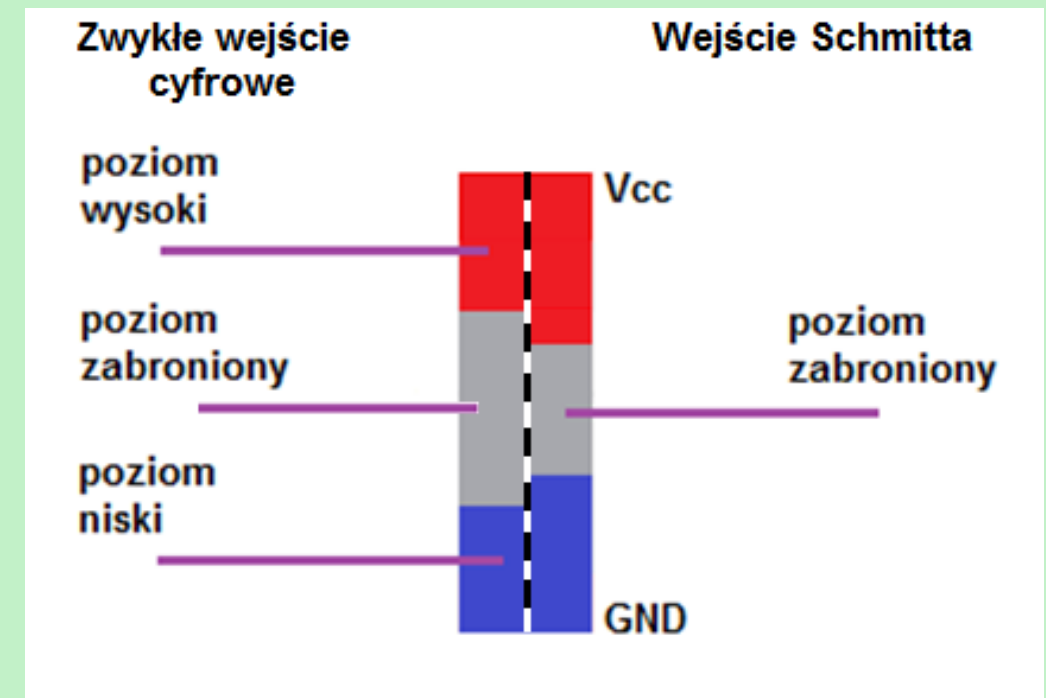
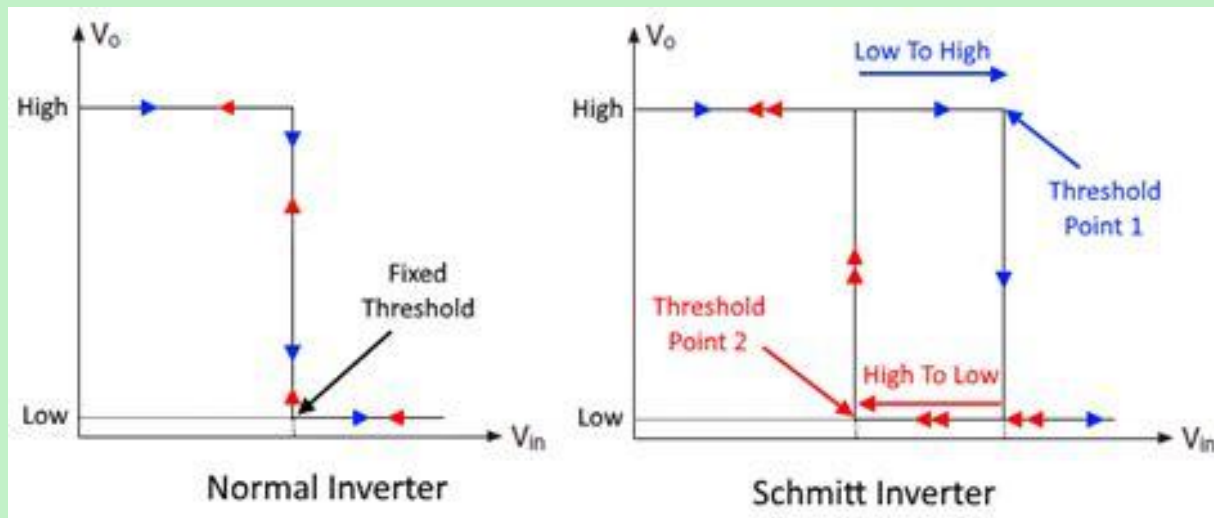
Wykorzystując elementy tri state możemy organizować obsługę transmisji dwukierunkowej:



W tym układzie:

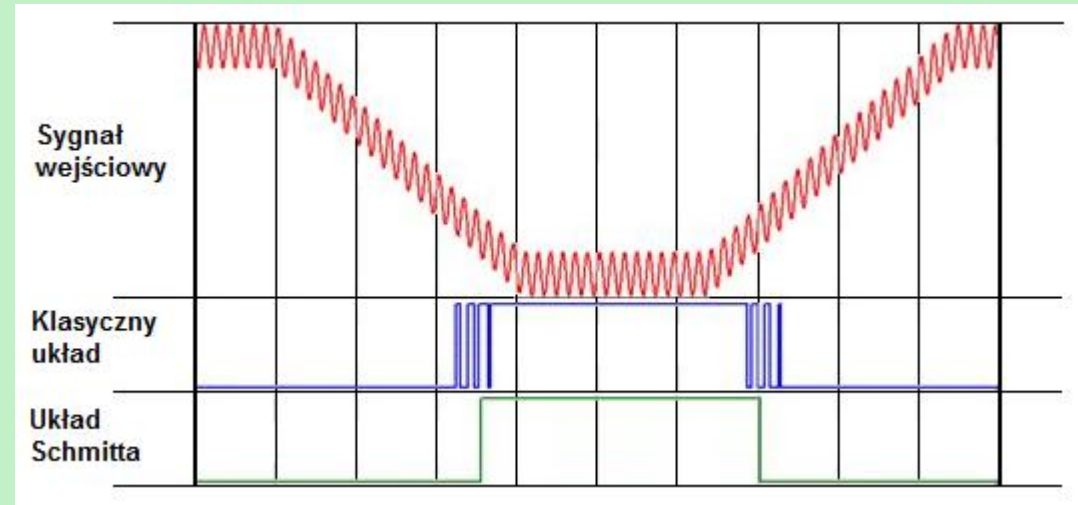
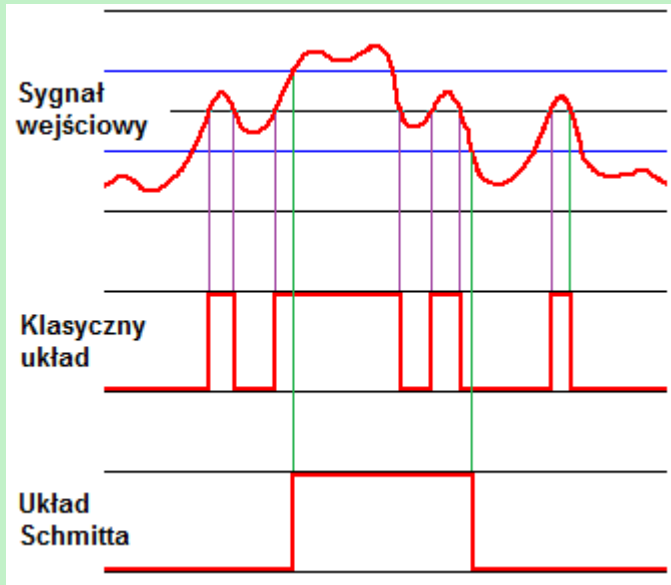
- dane są wysyłane z wejścia We_1 na linię dwukierunkową We_Wi gdy $S1 = 1$,
- dane są wysyłane z wejścia We_2 na linię We_Wi gdy $S2 = 1$,
- gdy $S1 = 0$ i $S2 = 0$ – możliwy jest odczyt z magistrali danych przychodzących,
- **NIE MOŻE** się zdarzyć sytuacja gdy jednocześnie $S1 = 1$ i $S2 = 1$ – oznaczałoby to konflikt danych na magistrali, czyli zwarcie wyjść aktywnych buforów trójstanowych a w konsekwencji uszkodzenie tych układów.

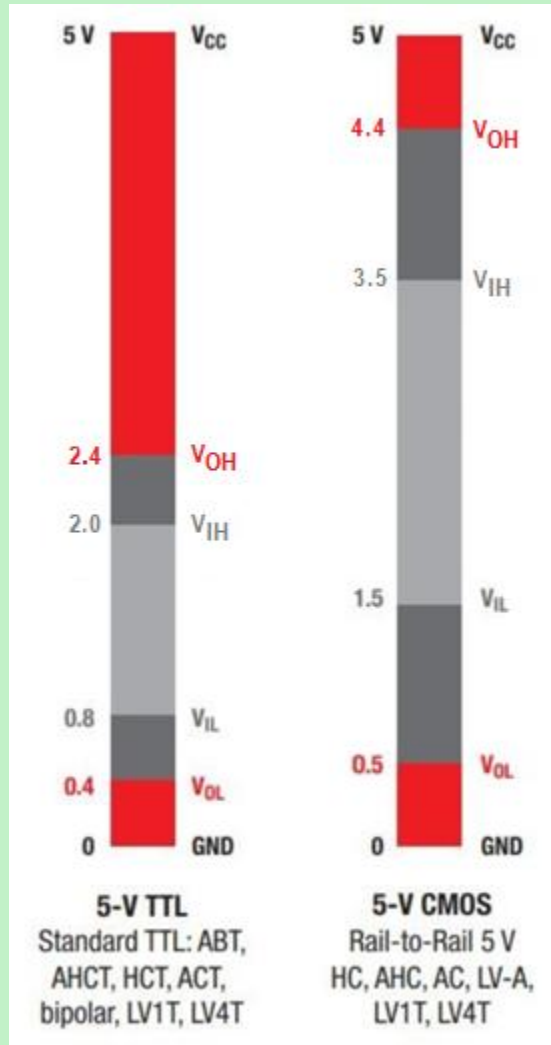
Niektóre układy posiadają na wejściach wbudowane układy Schmitta, co sygnalizuje charakterystyczny element symbolu. Są to układy wolniejsze, ale bardziej odporne na zakłócenia, posiadają rozszerzone i rozmyte zakresy sygnałów wejściowych low i high:



Układy Schmitta stosowane są:

- w sytuacji silnie zakłóconych sygnałów wejściowych - gdzie klasyczne układy wpadałyby w oscylacje,
- do regeneracji sygnałów wejściowych,
- do kształtowania sygnałów – np. przekształcania sygnału sinusoidalnego na prostokątny.





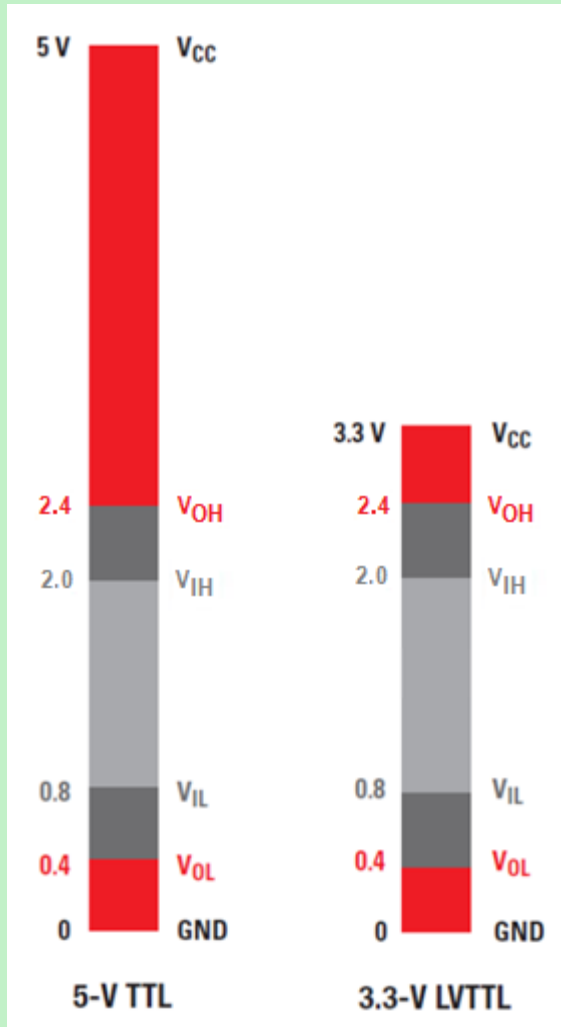
Historycznie utrwaliło się przekonanie, że wartość 0 to poziom napięcia około masy zasilania, zaś wartość logiczna 1 to poziom bliski zasilaniu, a nawet +5V.

Najstarszy standard cyfrowy TTL (Transistor - Transistor Logic), sprzed 60 lat, praktycznie wyszedł z zastosowań – mimo to jest on podawany w podręcznikach. Był on opracowany w technologii bipolarnej, pobierał dużą moc przy wysokim napięciu zasilającym 5V. Typowy prąd zasilania bramek TTL mieścił się w zakresie od kilku do około 20 mA, zaś czasy propagacji na poziomie 20 ns.

Doczekał się on wielu nowelizacji w tym w CMOS - min.:

- AHCT (*Advanced CMOS*),
- HCT (*High-Speed CMOS*),
- ALS (*Advanced Low-Power Schottky Logic*),
- LVC (*Low-Voltage CMOS*),
- F (*Fast Advanced Schottkey*),
- S (Schottky), itp.

Od około 2000 r. wchodziły standardy niskonapięciowe.

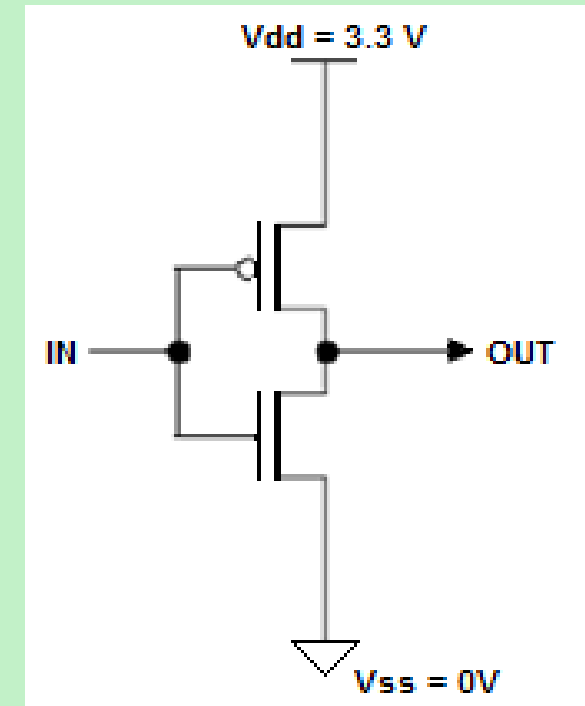


LVTTL (Low Voltage TTL) jest niskonapięciowym następcą standardu TTL w technologii unipolarnej. Przy niższym napięciu zasilającym zachował on wartości progowe napięć poziomów wejściowych i wyjściowych, co oznacza, że może on klasyczny TTL przy niższym zasilaniu.

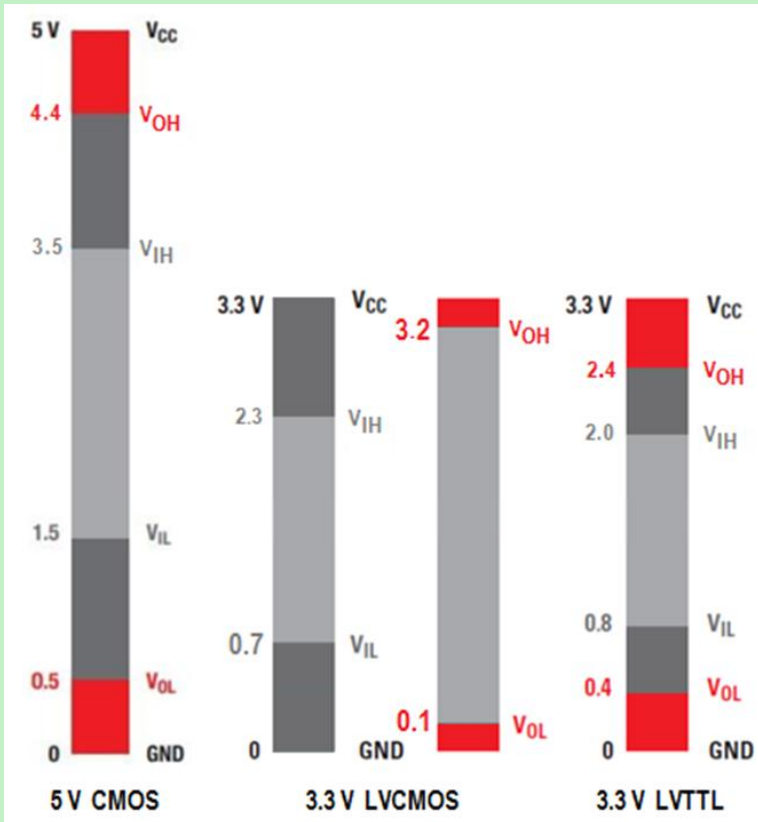
W standardzie LVTTL 3.3 V:

- maksymalna częstotliwość pracy wynosi 100 MHz,
- maksymalny prąd wyjściowy w stanie wysokim wynosi -4 mA,
- maksymalny prąd wyjściowy w stanie niskim wynosi 4 mA.

W klasycznym TTL wyjściowa obciążalność to około -1 mA/+16 mA.



LVC MOS (Low Voltage Complementary Metal Oxide Semiconductor - JESD8-B, 3.3 V) jest niskonapięciową wersją CMOS 5 V. Łącząc dwa układy tej samej serii zasilania, nie trzeba stosować dopasowania impedancyjnego. Schematy dopasowujące identyczne jak dla LVTTL.

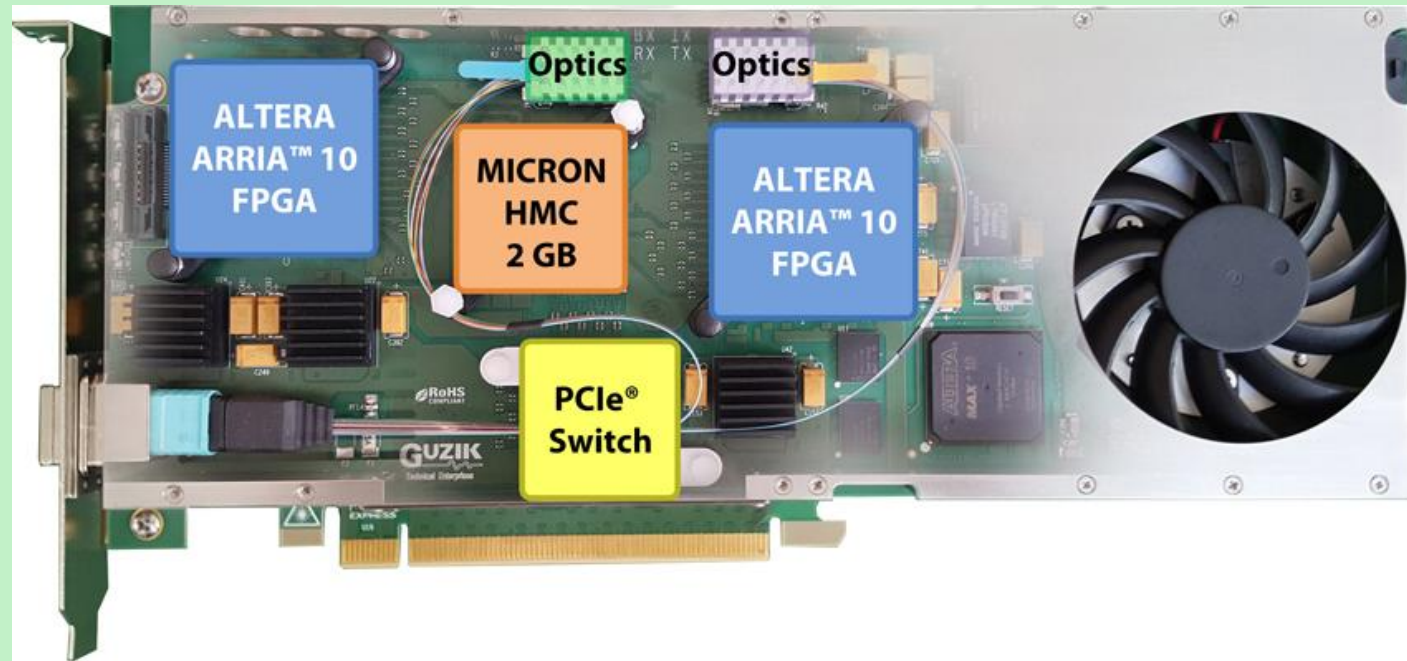


Oprócz wersji 3.3 V kompatybilnej z 2.5V i 3.0 V, istnieją szybsze wersje:

- 2.5V LVC MOS (**EIA/JESD8-5**)
obsługuje: LVC MOS 2.5V, 3.0V i 3.3V,
- 1.8V LVC MOS (**EIA/JESD8-7**)
obsługuje: LVC MOS 1.8V i 1.5V,
- 1.5V LVC MOS (**EIA/JESD8-11**)
obsługuje LVC MOS 1.8V i 1.5V,
- 1.2V LVC MOS akceptuje tylko LVC MOS 1.2V.

Graniczna częstotliwość pracy LVC MOS to około 160 MHz, obciążalności prądowe są różne, zależą od producenta i nie przekraczają 25 mA.

Standard PCI/PCIe (Peripheral Component Interconnect/Interconnect Express) jest zasilany napięciem 3.3 V, w kartach PC nie wymaga dopasowania impedancyjnego.



Przepustowość standardu to około 5 GB/s, jest on stosowany w równoległych magistralach 32 i 64 bitowych. W transmisji różnicowej w tym standardzie obowiązuje rezystor obciążający 100 Ω na końcu linii.

W tym standardzie zaleca się połączenia dłuższego niż 12 cali o szerokości ścieżek od 7 milsów (100 milsów = 2.54 mm = 0.1”).

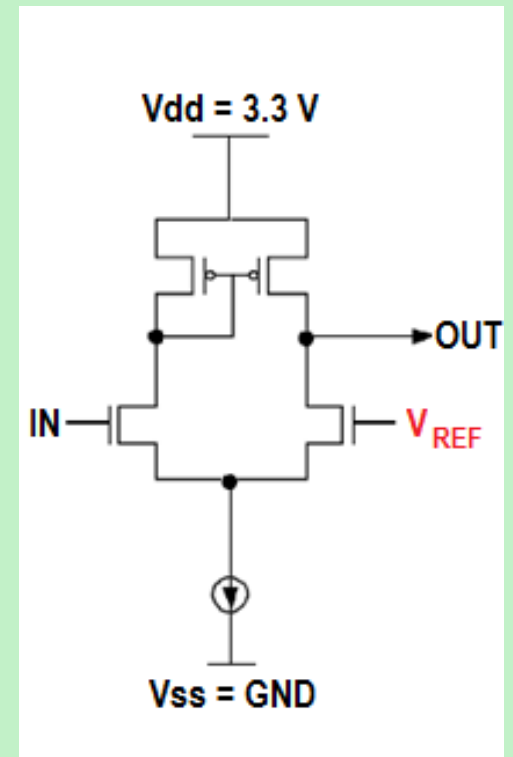
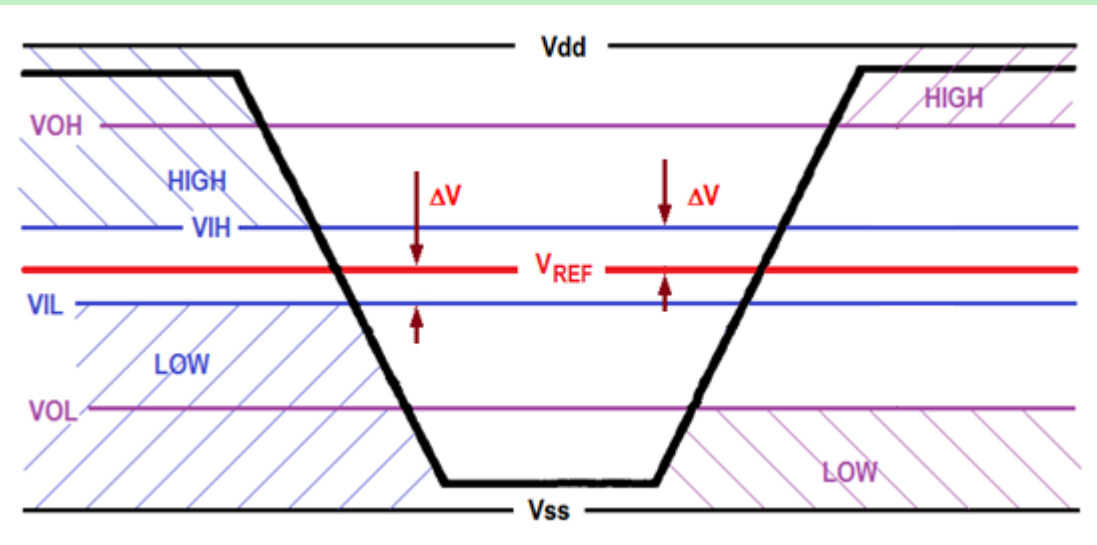
Niesymetryczny 3.3 V standard SSTL-3 (Stub Series Terminated Logic) był opracowany do obsługi pamięci dynamicznych SDRAM. Był on kompatybilny z 3.3 V LVTTTL. Po wycofaniu SDRAM, został on zastąpiony nowszymi wersjami, częstotliwość pracy sięgała 166 MHz.

W SSTL poziomy 1 i 0 na wejściu odnoszone są nie do potencjału masy (0 V) lecz do dołączanego z zewnątrz **napięcia referencyjnego** V_{REF} – połowy zasilania.

W wersji SSTL-3 poziomy logiczne są następujące:

- 0 gdy sygnał wejściowy jest mniejszy od V_{REF} o 200 mV,
- 1 gdy sygnał wejściowy jest większy od $V_{REF} + 200$ mV.

Margines 400 mV ($2 \times \Delta V$) zapewnia szybszą transmisję niż w standardzie LVTTTL (margines 1.2 V).

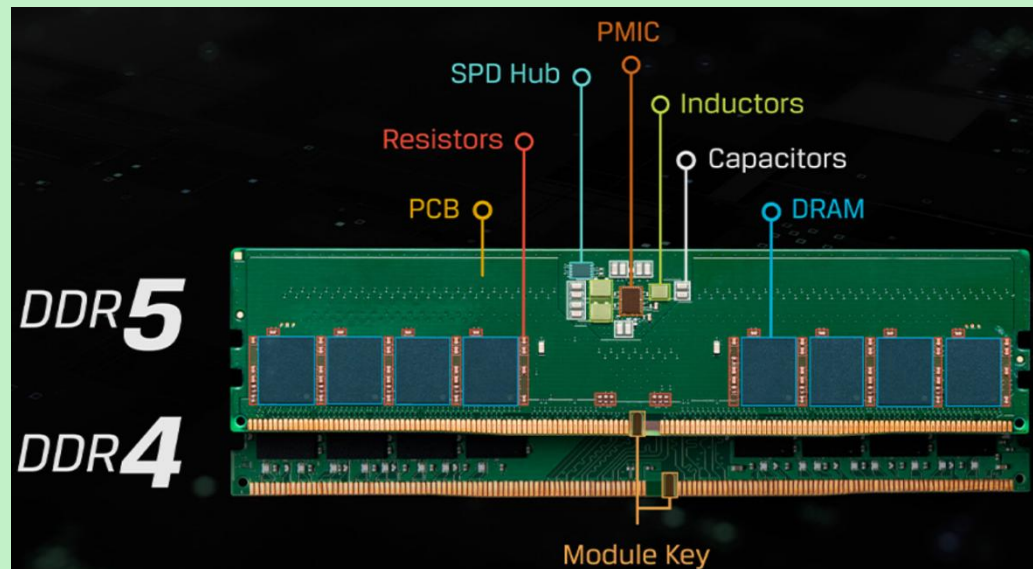


W praktyce obecnie stosowane są wyższe wersje niesymetrycznego SSTL:

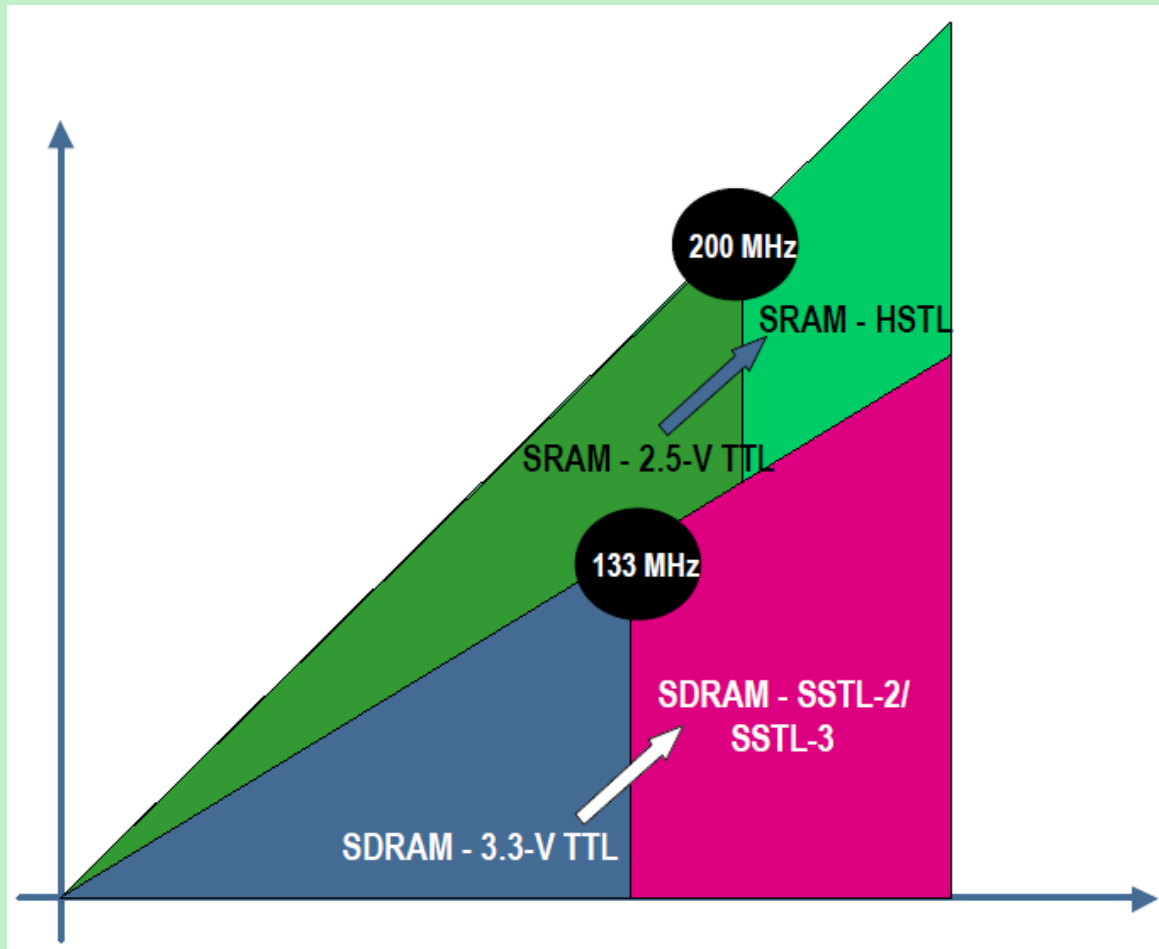
- standard SSTL-2 zasilany napięciem 2.5V do obsługi pamięci DDR (333 MHz),
- standard SSTL-18 zasilany napięciem 1.8V do obsługi pamięci DDR II (do 530 MHz Class I/II),
- standard SSTL-15 zasilany napięciem 1.5V do obsługi pamięci DDR III (do 1 GHz),
- standard SSTL-12 zasilany napięciem 1.2V do obsługi pamięci DDR IV (do 4 GHz).
- standard SSTL-11 zasilany 1.1V do obsługi DDR V (2021) (6.4 GHz do 10 GHz).

Zalety referencyjnych niskonapięciowych standardów:

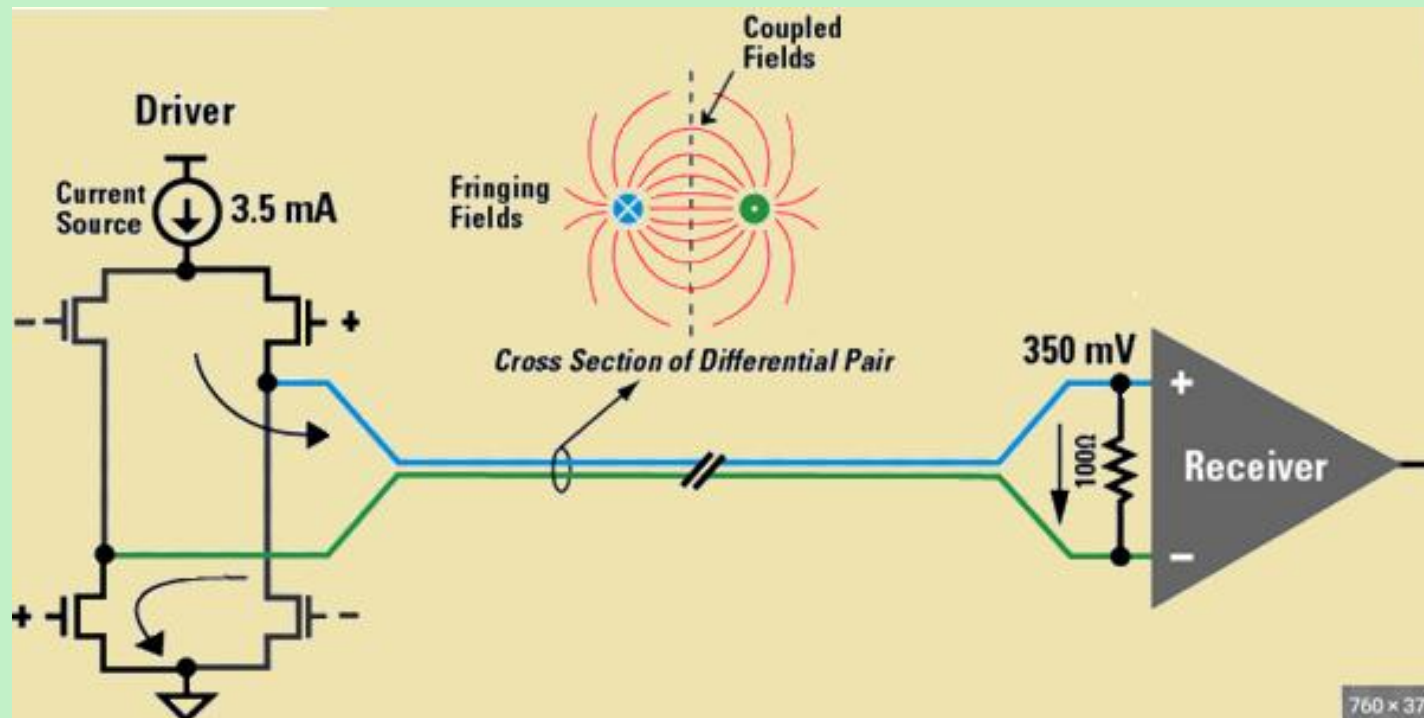
- redukcja zakłóceń elektromagnetycznych i mniejszy pobór mocy,
- mniejsze zakłócenia spowodowane odbiciami zewnętrznymi,
- szersze pasmo transmisyjne.



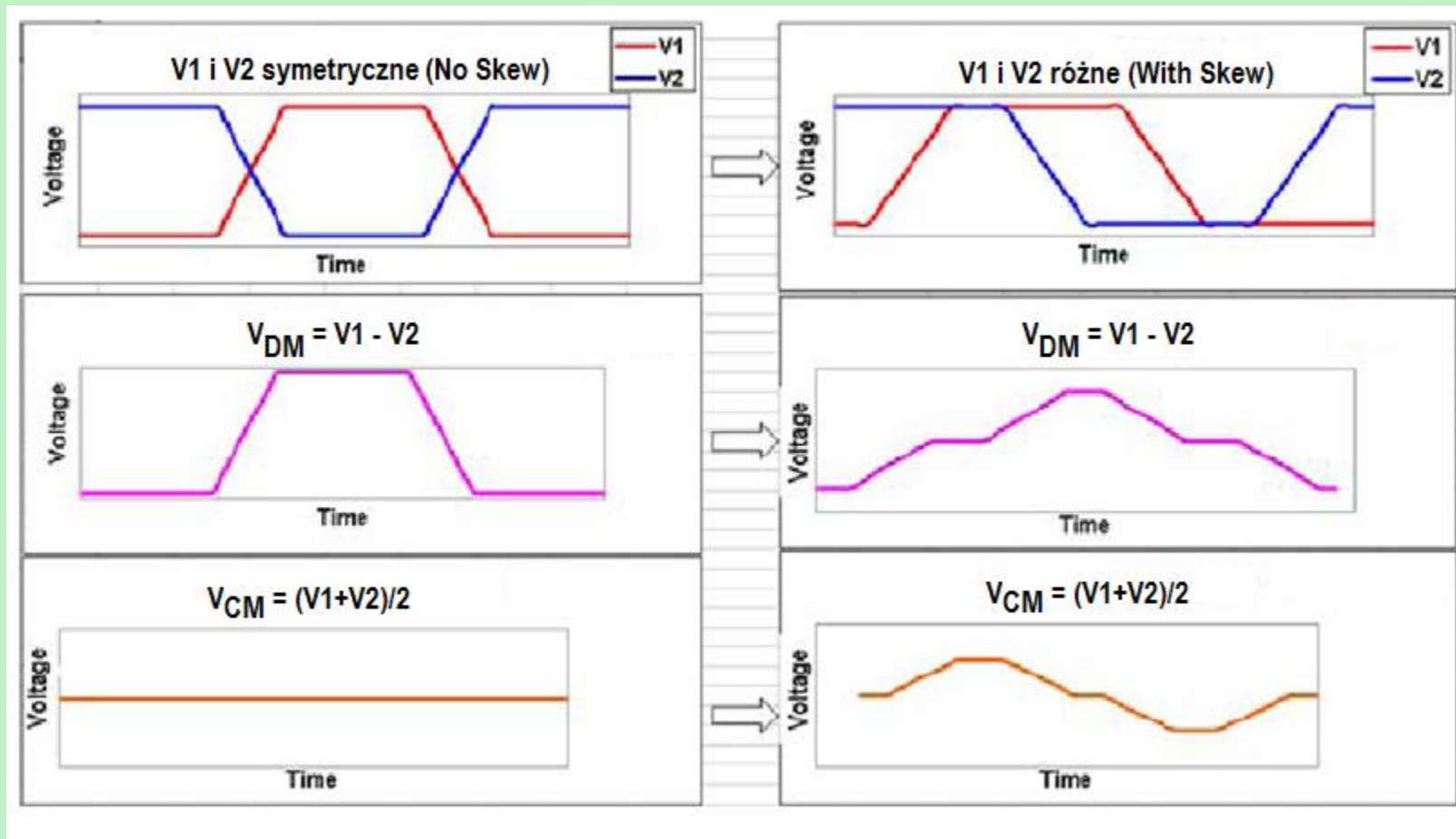
Niesymetryczny standard HSTL jest (High-speed transceiver logic JESD8-6) technologicznie niezależnym, skalowanym napięciowo standardem transmisyjnym. Standard przeznaczony do obsługi szybkich (powyżej 200 MHz) pamięci statycznych. Nie jest on związany z konkretnym napięciem zasilającym, aczkolwiek wyróżnia się klasy: 1.8V HSTL-18 Class I/II, 1.5V HSTL-15 Class I/II, 1.2V HSTL-12 Class I/II.



W transmisji symetrycznej/różnicowej jest przesyłany dwiema liniami, w każdej z nich przepływa ten sam sygnał ale w przeciw-fazie, tzn. przesunięty w fazie o 180 stopni względem drugiej linii. W odbiorniku, obie linie zamyka rezystor, na którym mierzony jest spadek napięcia przez wzmacniacz różnicowy odbiornika. Jeżeli w liniach występuje różnica sygnałów, która powoduje spadek napięcia na rezystorze zamykającym, oznacza to poziom wysoki. W przeciwnym przypadku jest to zero logiczne. Taki system posiada naturalny mechanizm redukcji zakłóceń, gdyż zakłócenia zewnętrzne w jednakowym stopniu indukują się w obu liniach i na wejściach odbiornika wytwarzają zerowe napięcie – tej cechy nie posiadają systemy niesymetryczne.



Efekt tłumienia sygnałów sumacyjnych i wzmacniania różnicowych słabnie gdy: zaburzona jest symetria geometryczna linii – tzn. w jednej linii wystąpi opóźnienie sygnału inne niż 180 stopni względem drugiej. Lewy rysunek pokazuje sytuację idealną i silne tłumienie sygnału zakłócającego/sumacyjnego, a prawy efekt wystąpienia niesymetrii.



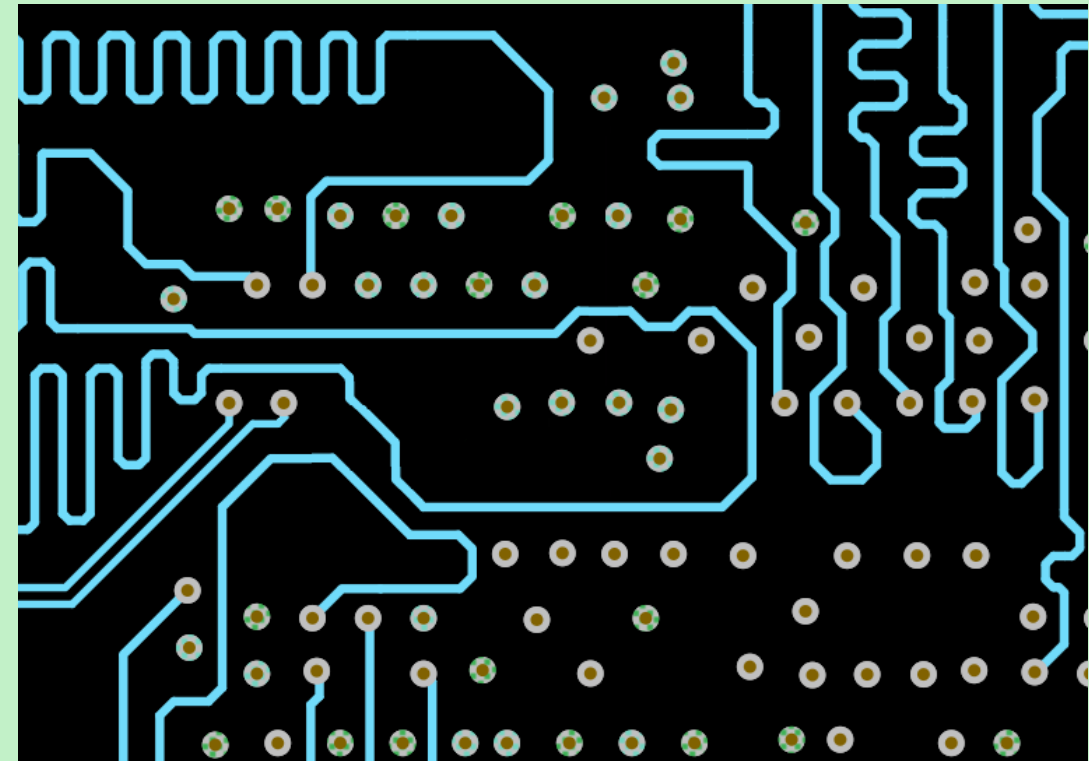
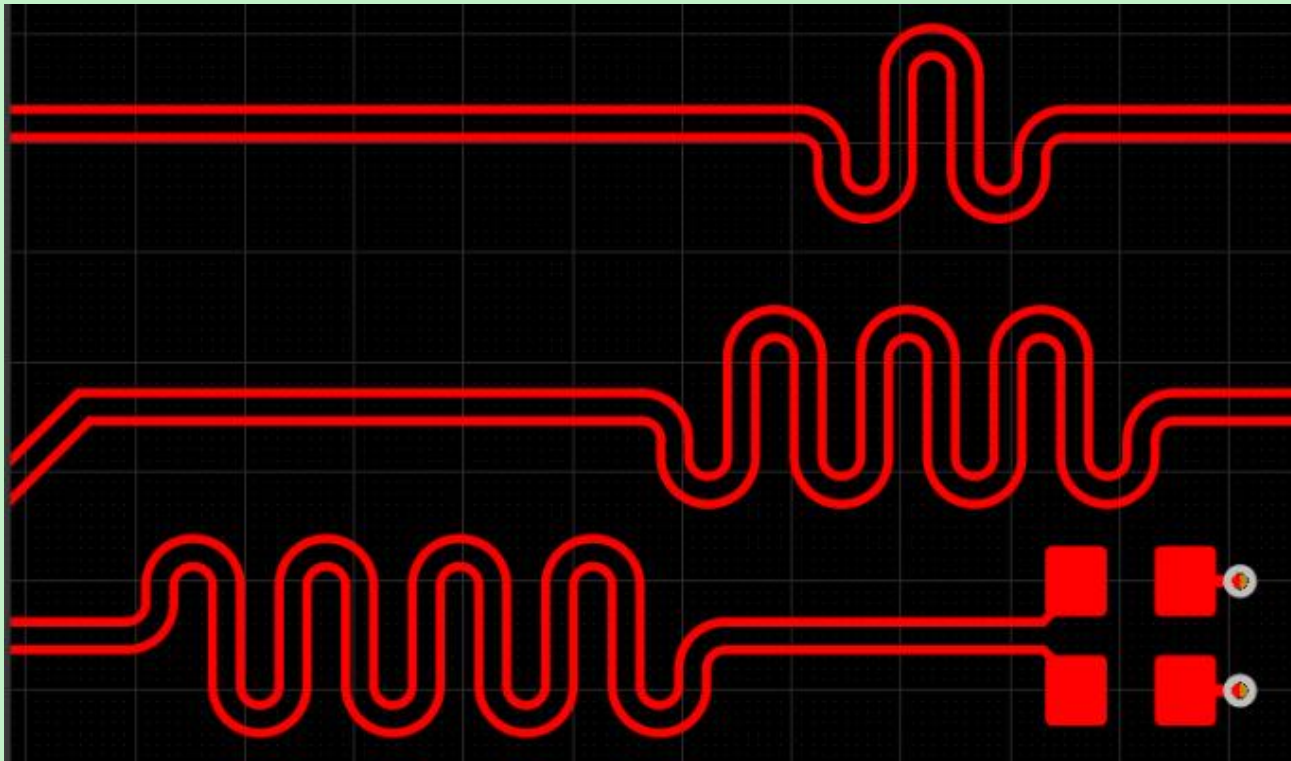
Stąd w transmisji różnicowej, kluczowym zagadnieniem jest symetria układu przesyłania sygnałów.

W transmisjach niesymetrycznych ważne są:

- przerwy w ciągłości linii,
- czasowe marginesy synchronizacyjne
- transmisji synchronicznych,
- strategie dopasowania impedancji

Z kolei W transmisji różnicowej istotne są:

- różnice opóźnień sygnałów w liniach,
- układ przelotek PCB,
- parametry dielektryka PCB,
- nieciągłości impedancyjne.

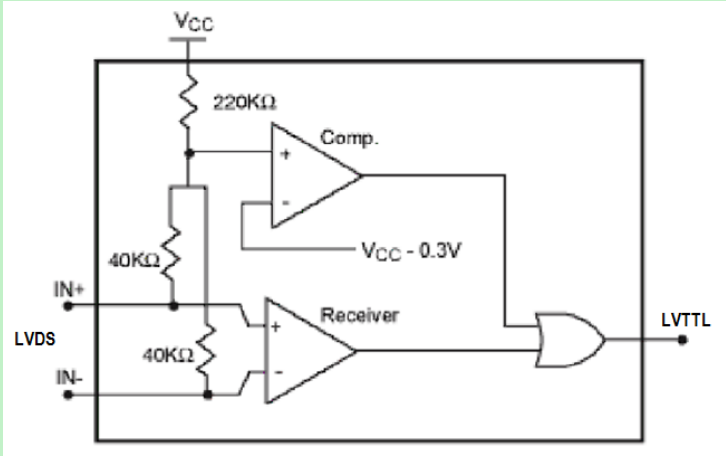
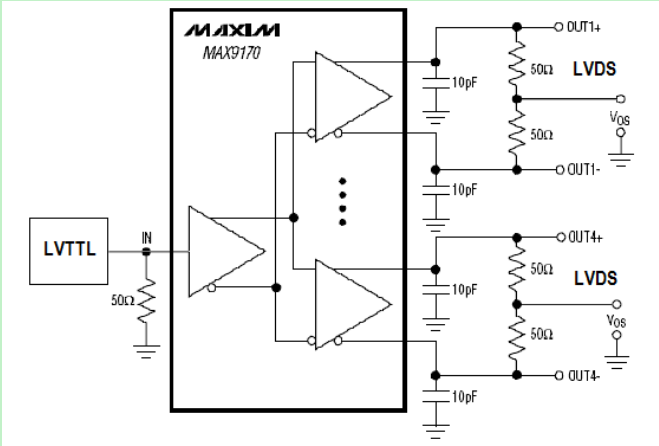


Wymagania PCB transmisji różnicowej. W celu zachowania różnicowych warunków transmisyjnych należy:

- umieszczać obciążenie linii symetrycznej na jej końcu,
- na całej długości linii należy zachować tę samą wartość impedancji różnicowej,
- należy w taki sposób wytwarzać sygnał różnicowy, żeby minimalizować różnicę opóźnień,
- należy minimalizować liczbę przelotek i innych nieciągłości w linii,
- elementy pasożytnicze takie jak np. pojemności powinny być identyczne w obu liniach,
- zaleca zmianę kierunku ścieżek pod kątem 45 stopni zamiast 90,
- sygnały różnicowe należy separować od sygnałów niesymetrycznych (LVTTTL, LVCMOS, ...),
- jeżeli to możliwe sygnały różnicowe należy prowadzić na oddzielnych warstwach PCB,
- zachować 12 mm między sygnałami różnicowymi i niesymetrycznymi tej samej warstwy,
- warstwy zasilające mogą pełnić rolę warstw separujących.

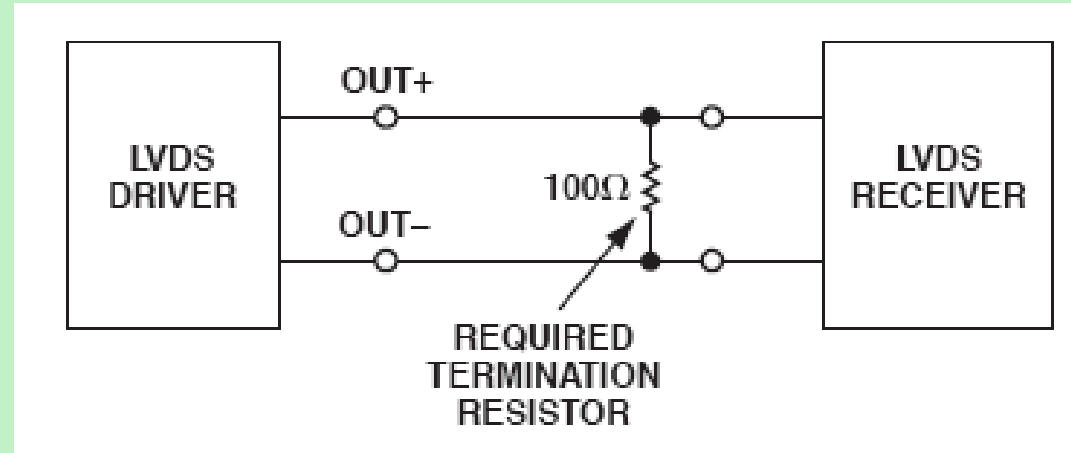


Nie należy manualnie przekształcać sygnałów niesymetrycznych w symetryczne i odwrotnie – nie stosować negatorów w jednej z linii. Istnieją specjalizowane układy scalone translatorów standardów (np. firmy Maxim).



Rx	Tx							
	ECL	PECL	LVECL	LVPECL	CMOS	LVC MOS	LVDS	LVTTL
PECL								
LVECL								
LVPECL								
CMOS								
LVC MOS								
LVDS								
TTL								
LVTTL								
HSTL								

Standard różnicowy LVDS:



Standard został opracowany do:

- szybkiej transmisji danych – 1350 MHz,

- transmisji telekomunikacyjnej,

- wymiany danych pomiędzy serwerami i urządzeniami peryferyjnymi,

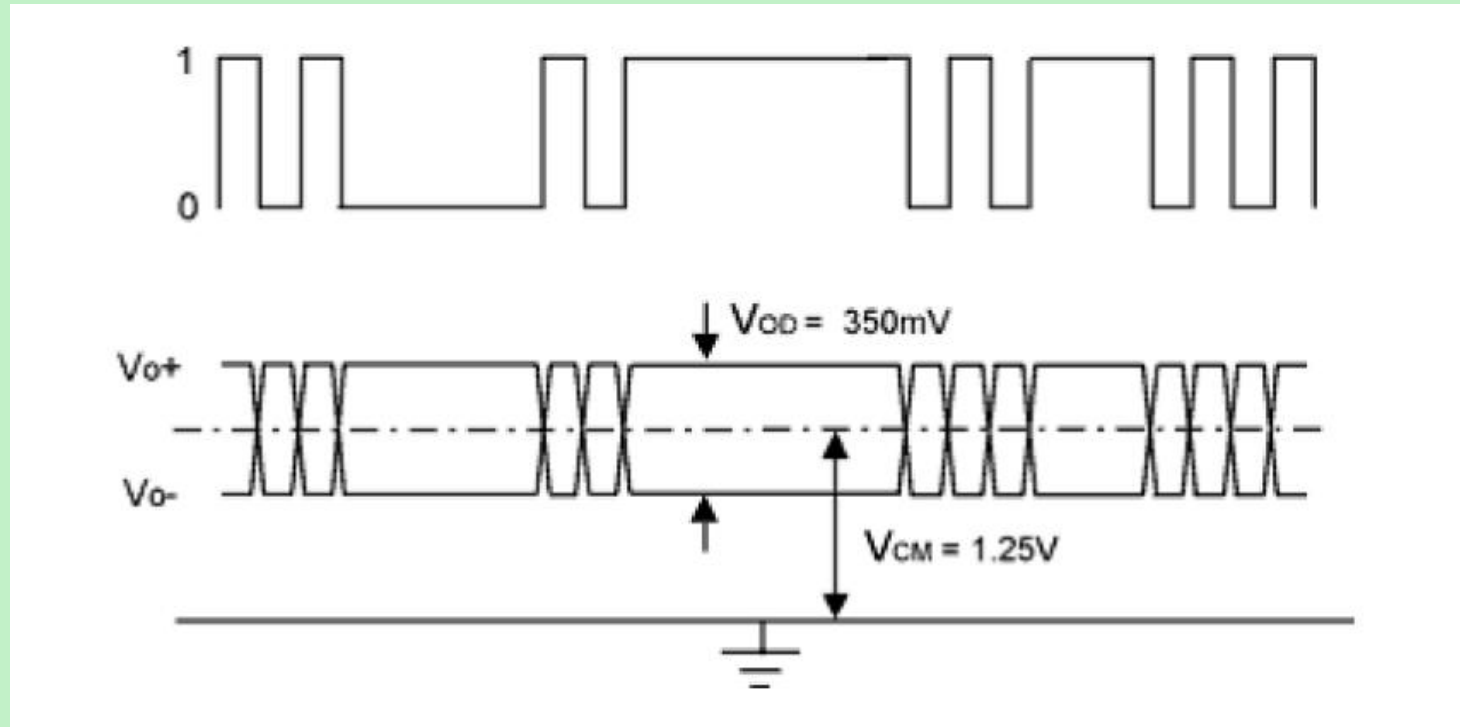
- w stosunku do innych standardów umożliwia dużo wyższą szybkość transmisji,

- charakteryzuje go mały pobór mocy – około 1,2 mW na kanał,

- jest odporny na zakłócenia.

Wyjściowy sygnał LVDS:

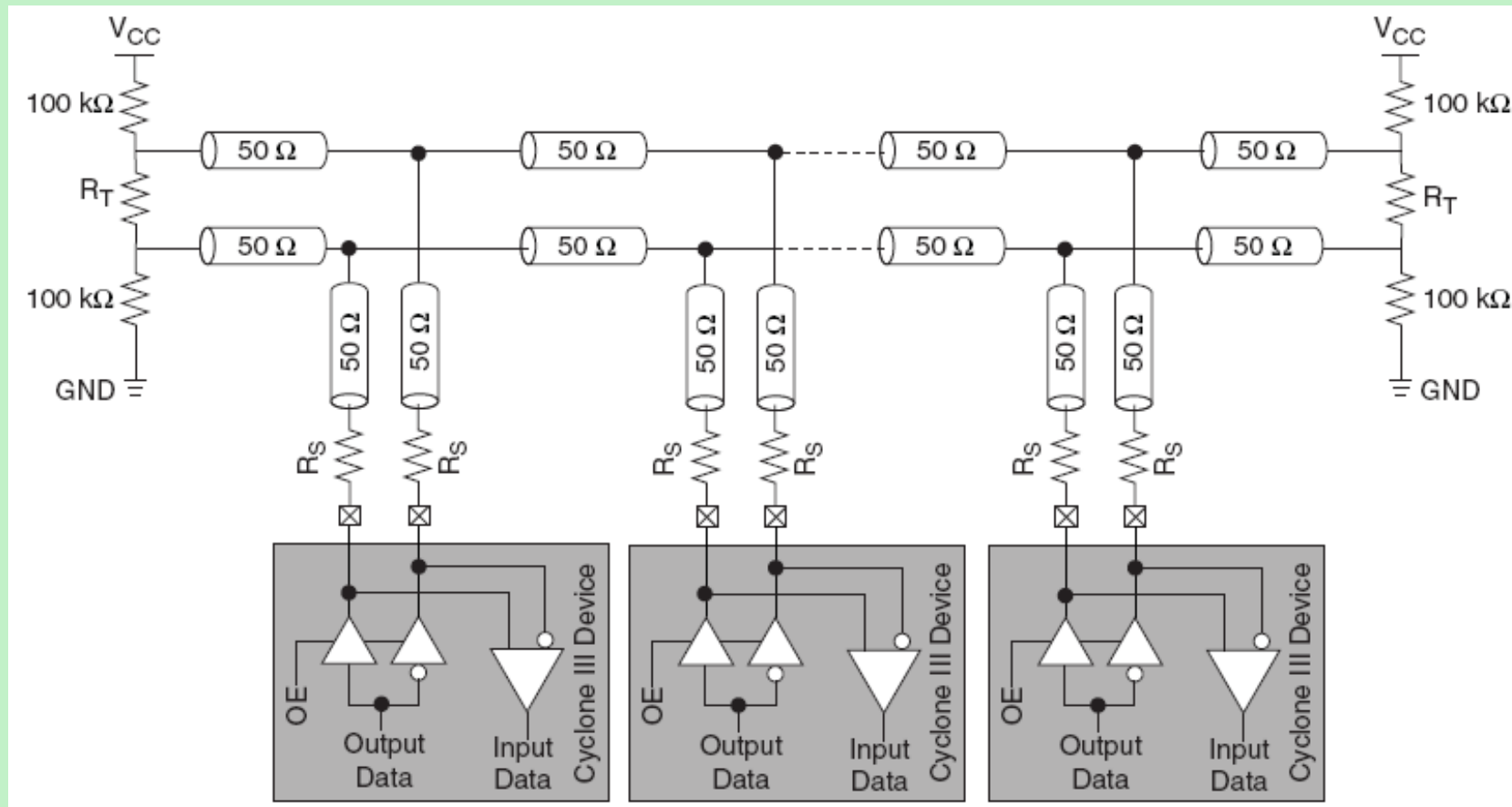
- koncentruje się wokół napięcia 1,25 V (masy pozornej),
- wartość międzyszczytowa składowej zmiennej wynosi 350 mV,
- standard nie jest ściśle związany z określoną wartością napięcia zasilającego.



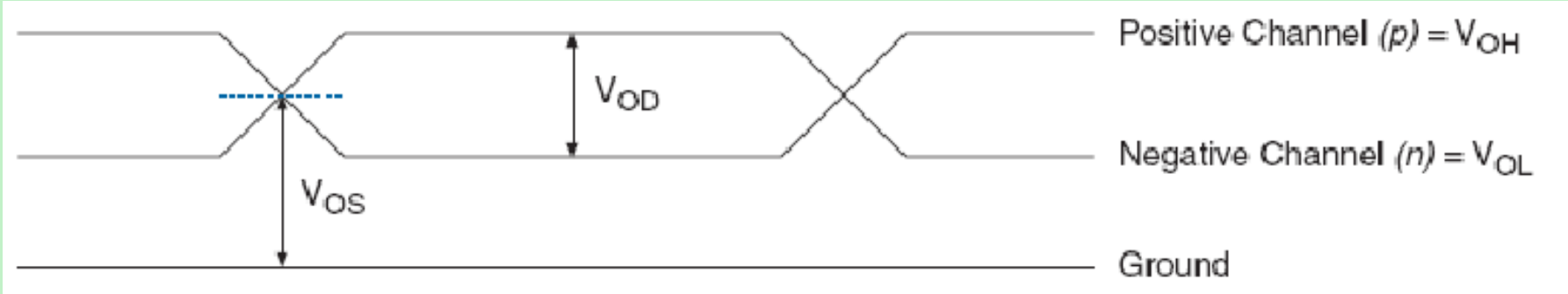
Przemysłowa konfiguracja Multi-Point Configuration standardu LVDS określana jest mianem BLVDS.

Wykorzystywana jest do szybkich transmisji w zakresie setek MHz.

Jest to dwukierunkowa transmisja po wspólnej szynie z transilwerami (nadajnik/odbiornik) dołączonymi do szyny, pracującymi z podziałem czasu.



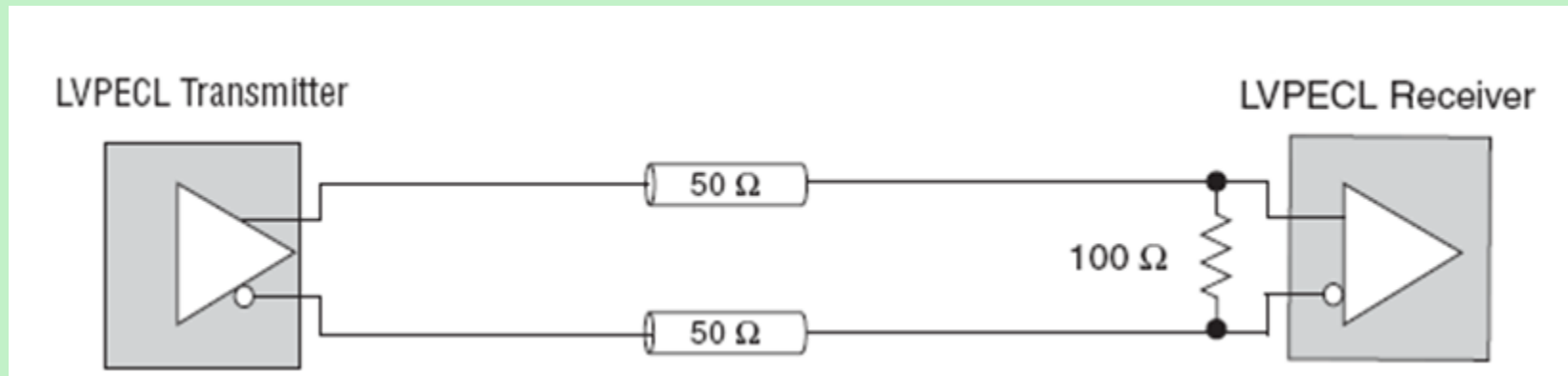
Standard RSDS. Transmisja wykorzystywana do obsługi panelowych wyświetlaczy. Szybkość transmisji wynosi do do 360 MBps. Parametry standardu:



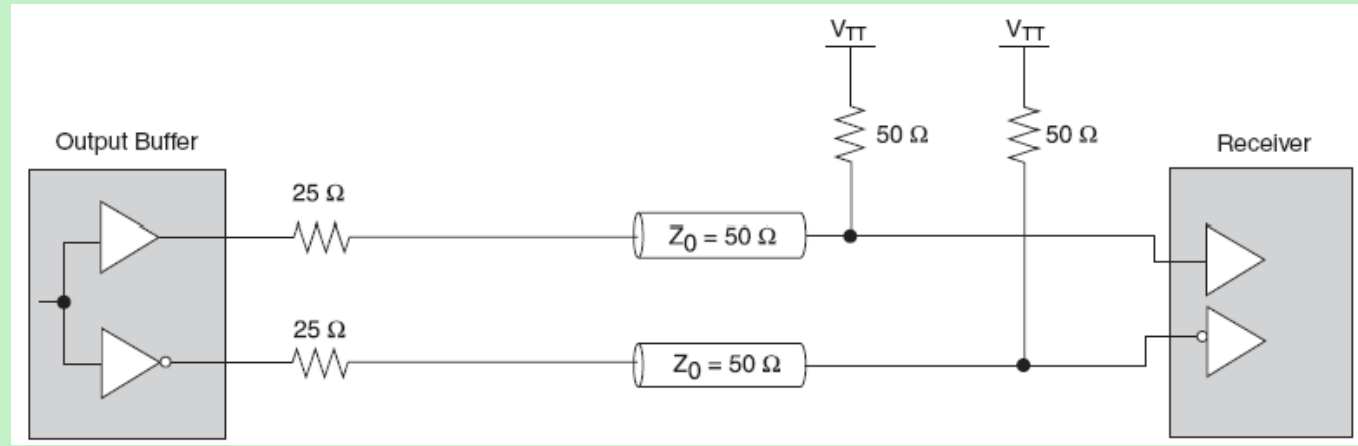
Symbol	Parameter	Min	Typical	Max	Units
V _{CCIO}	I/O supply voltage	2.375	2.5	2.625	V
V _{OD}	Differential output voltage	100	200	600	mV
V _{OS}	Output offset voltage	0.5	1.2	1.5	V
V _{TH}	Differential threshold			+/- 100	mV

Standard LVPECL. Standard jest standardem różnicowym, zasilanym napięciem 2,5 V. Stosowany on jest: w systemach grafiki video, telekomunikacji, transmisji danych, rozdzielaniu sygnałów zegarowych. Standard nie wymaga napięcia referencyjnego, niezbędny jest rezystor zamykający.

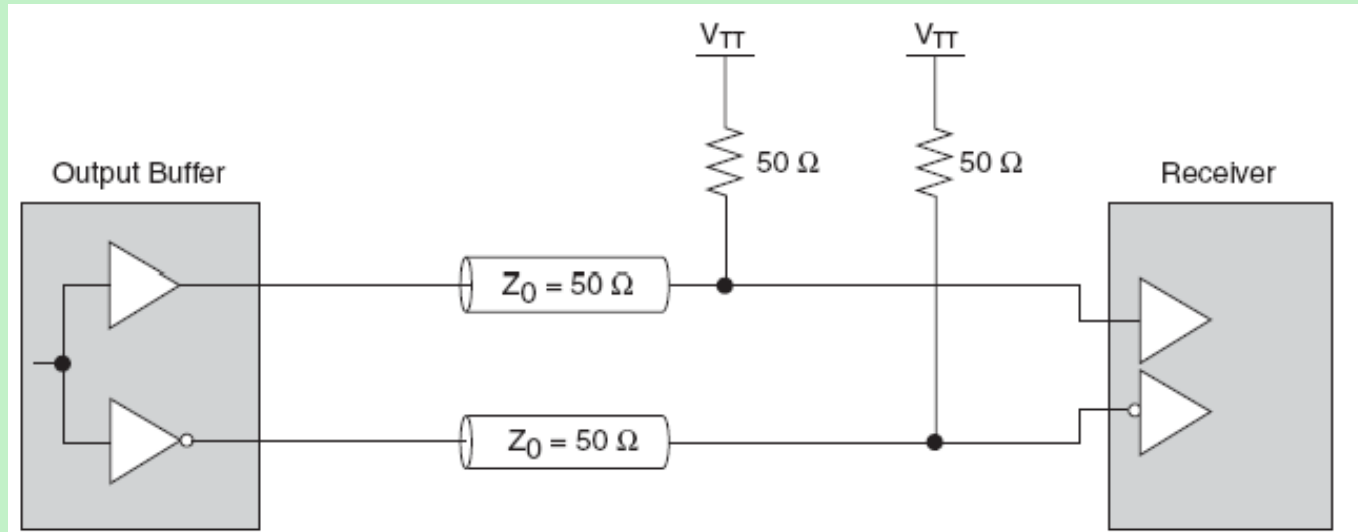
<i>Table 11–4. LVPECL Electrical Characteristics</i>						
Symbol	Parameters	Condition	Min	Typ	Max	Units
V_{CCIO}	Output supply voltage		3.135	3.3	3.465	V
V_{IH}	Input high voltage		2,100		2,880	mV
V_{IL}	Input low voltage		0		2,200	mV
V_{ID}	Differential input voltage	Peak to peak	100	600	950	mV



Różnicowy standard SSTL - stosowany jest do obsługi szybkich pamięci DDR, wymaga napięcia V_{REF} i napięcia polaryzującego V_{TT} , równych połowie V_{CCIO}



Różnicowy standard HSTL - stosowany jest do obsługi szybkich pamięci statycznych – przykładowy schemat:



Zaliczenie częściowe wykładu nr 3 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

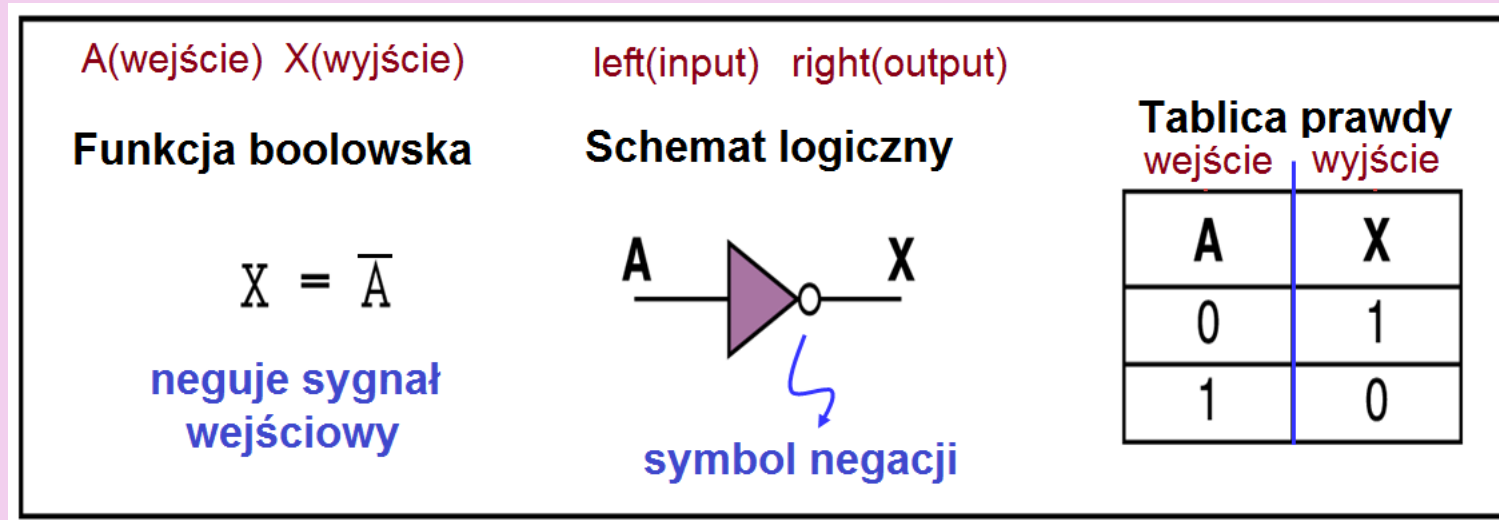
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

Bramka NOT posiada jedno wejście i jedno wyjście, sygnał wyjściowy jest zanegowaną wartością sygnału wejściowego.



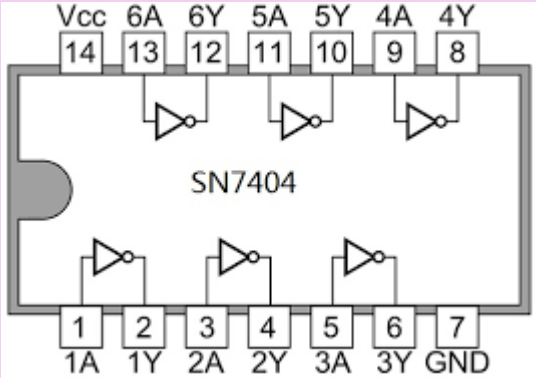
Opis bramki w kodzie VHDL – w przykładzie wejście jest magistralą, stąd funkcja NOT działa bitowo:

```
X <= not A;
```

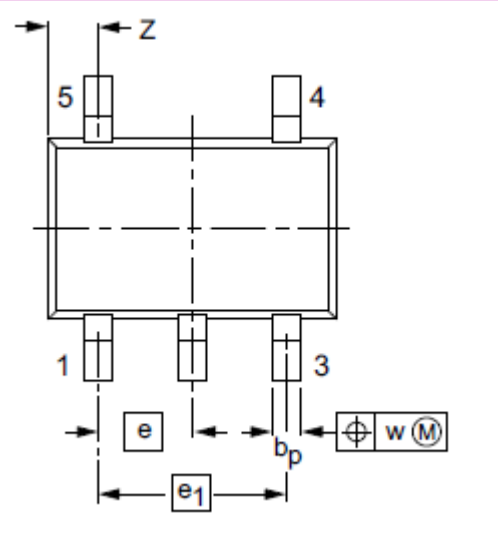
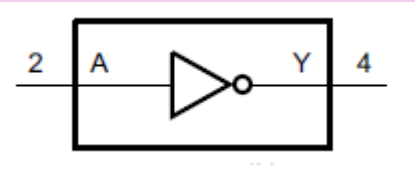
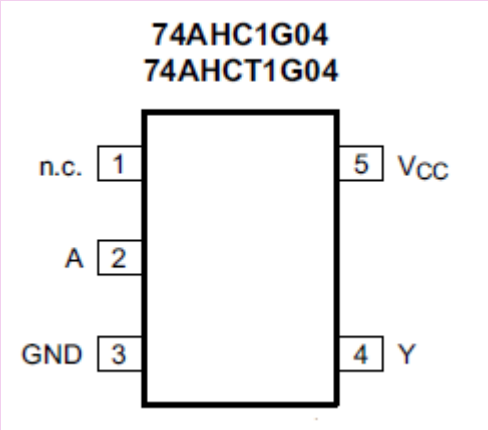
```
A = "0101"      x = "1010"
```

Parametry techniczne bramki NOT:

Historia



Współczesność



DIMENSIONS (mm are the original dimensions)

UNIT	A _{max.}	A ₁	A ₂	A ₃	b _p	c	D ⁽¹⁾	E ⁽¹⁾	e	e ₁	H _E	L	L _p	v	w	y	Z ⁽¹⁾	θ
mm	1.1	0.1 0	1.0 0.8	0.15	0.30 0.15	0.25 0.08	2.25 1.85	1.35 1.15	0.65	1.3	2.25 2.0	0.425	0.46 0.21	0.3	0.1	0.1	0.60 0.15	7° 0°

Parametry techniczne bramki NOT:

Table 5. Limiting values*In accordance with the Absolute Maximum Rating System (IEC 60134).*

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	supply voltage		-0.5	+7.0	V
V_I	input voltage		-0.5	+7.0	V
I_{IK}	input clamping current	$V_I < -0.5 \text{ V}$	-20	-	mA
I_{OK}	output clamping current	$V_O < -0.5 \text{ V}$ or $V_O > V_{CC} + 0.5 \text{ V}$ ^[1]	-	±20	mA
I_O	output current	$-0.5 \text{ V} < V_O < V_{CC} + 0.5 \text{ V}$	-	±25	mA
I_{CC}	supply current		-	75	mA
I_{GND}	ground current		-75	-	mA
T_{stg}	storage temperature		-65	+150	°C
P_{tot}	total power dissipation	$T_{amb} = -40 \text{ °C to } +125 \text{ °C}$ ^[2]	-	250	mW

Table 6. Recommended operating conditions*Voltages are referenced to GND (ground = 0 V).*

Symbol	Parameter	Conditions	74AHC1G04			74AHCT1G04			Unit
			Min	Typ	Max	Min	Typ	Max	
V_{CC}	supply voltage		2.0	5.0	5.5	4.5	5.0	5.5	V
V_I	input voltage		0	-	5.5	0	-	5.5	V
V_O	output voltage		0	-	V_{CC}	0	-	V_{CC}	V
T_{amb}	ambient temperature		-40	+25	+125	-40	+25	+125	°C
$\Delta t/\Delta V$	input transition rise and fall rate	$V_{CC} = 3.3 \text{ V} \pm 0.3 \text{ V}$	-	-	100	-	-	-	ns/V
		$V_{CC} = 5.0 \text{ V} \pm 0.5 \text{ V}$	-	-	20	-	-	20	ns/V

Parametry techniczne bramki NOT:

Table 7. Static characteristics										
Voltages are referenced to GND (ground = 0 V).										
Symbol	Parameter	Conditions	25 °C			−40 °C to +85 °C		−40 °C to +125 °C		Unit
			Min	Typ	Max	Min	Max	Min	Max	
74AHC1G04										
V _{IH}	HIGH-level input voltage	V _{CC} = 2.0 V	1.5	-	-	1.5	-	1.5	-	V
		V _{CC} = 3.0 V	2.1	-	-	2.1	-	2.1	-	V
		V _{CC} = 5.5 V	3.85	-	-	3.85	-	3.85	-	V
V _{IL}	LOW-level input voltage	V _{CC} = 2.0 V	-	-	0.5	-	0.5	-	0.5	V
		V _{CC} = 3.0 V	-	-	0.9	-	0.9	-	0.9	V
		V _{CC} = 5.5 V	-	-	1.65	-	1.65	-	1.65	V
V _{OH}	HIGH-level output voltage	V _I = V _{IH} or V _{IL}								
		I _O = −50 μA; V _{CC} = 2.0 V	1.9	2.0	-	1.9	-	1.9	-	V
		I _O = −50 μA; V _{CC} = 3.0 V	2.9	3.0	-	2.9	-	2.9	-	V
		I _O = −50 μA; V _{CC} = 4.5 V	4.4	4.5	-	4.4	-	4.4	-	V
		I _O = −4.0 mA; V _{CC} = 3.0 V	2.58	-	-	2.48	-	2.40	-	V
		I _O = −8.0 mA; V _{CC} = 4.5 V	3.94	-	-	3.8	-	3.70	-	V
V _{OL}	LOW-level output voltage	V _I = V _{IH} or V _{IL}								
		I _O = 50 μA; V _{CC} = 2.0 V	-	0	0.1	-	0.1	-	0.1	V
		I _O = 50 μA; V _{CC} = 3.0 V	-	0	0.1	-	0.1	-	0.1	V
		I _O = 50 μA; V _{CC} = 4.5 V	-	0	0.1	-	0.1	-	0.1	V
		I _O = 4.0 mA; V _{CC} = 3.0 V	-	-	0.36	-	0.44	-	0.55	V
		I _O = 8.0 mA; V _{CC} = 4.5 V	-	-	0.36	-	0.44	-	0.55	V
I _I	input leakage current	V _I = 5.5 V or GND; V _{CC} = 0 V to 5.5 V	-	-	0.1	-	1.0	-	2.0	μA
I _{CC}	supply current	V _I = V _{CC} or GND; I _O = 0 A; V _{CC} = 5.5 V	-	-	1.0	-	10	-	40	μA
C _I	input capacitance		-	1.5	10	-	10	-	10	pF

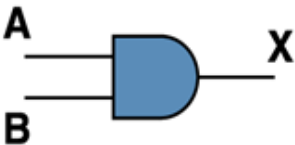
Bramka AND może posiadać jedno lub wiele wejść, jedno wyjście, na którym sygnał jest iloczynem logicznym sygnałów wejściowych. Zatem, na wyjściu bramki wystąpi stan 1 – jedynie gdy na wszystkich wejściach jest 1. Przypadek 2-wejściowej bramki AND :

Funkcja boolowska

$X = A \cdot B$

iloczyn logiczny
sygnałów wejściowych

Schemat logiczny



Tablica prawdy

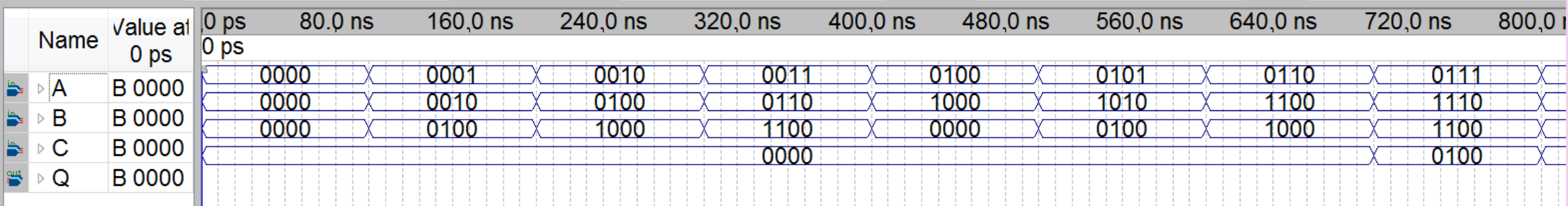
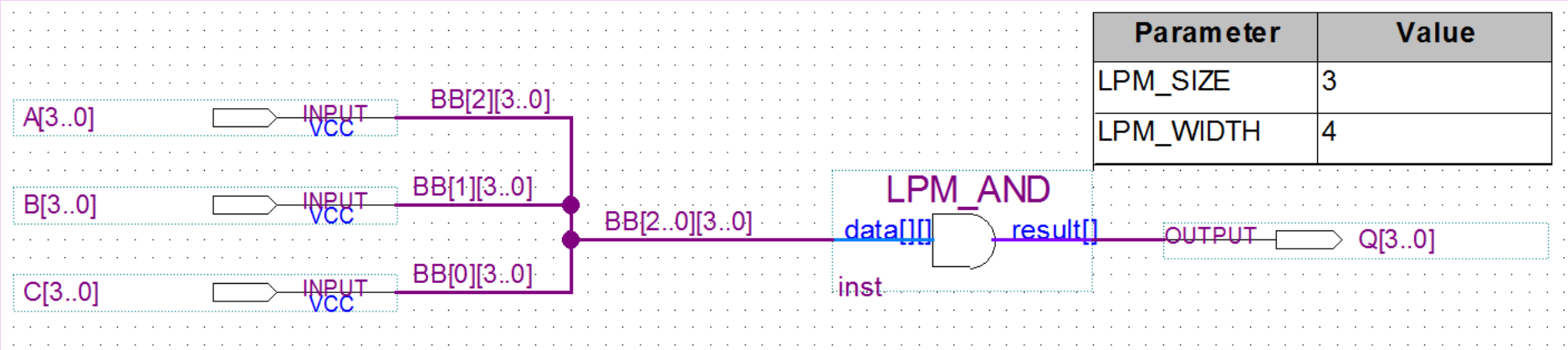
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Jeżeli wejściami bramki nie są pojedyncze bity lecz magistrale (słowa) – wówczas operator AND działa niezależnie na pozycjach o tej samej wadze danych wielobitowych (np. 8):

			00000000	00000001	00000010	00000011	00000100	00000101	00000110
▷ A	B 00...		00000000	00000010	00000100	00000110	00001000	00001010	00001100
▷ B	B 00...			00000000		00000010		00000000	00000100
▷ X	B 00...								

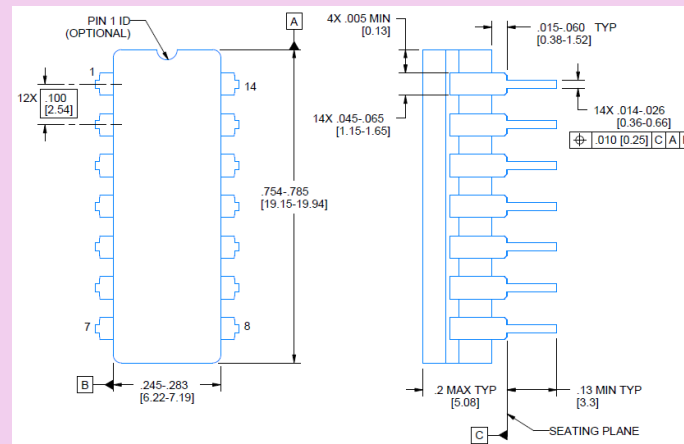
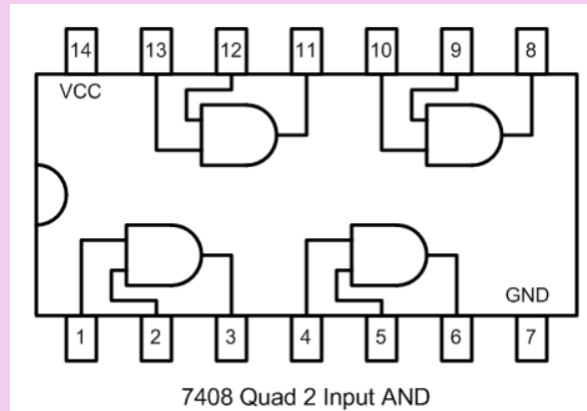
Alternatywą do opisu w VHDL bramek, których wejściami są magistrale, jest wykorzystanie edytora graficznego. W takiej sytuacji należy zwrócić uwagę na właściwe adresowanie i wymiar magistral oraz portów:

- wszystkie magistrale i porty powinny posiadać jednakową szerokość – np. 4 bity,
- w etykiecie typu A[3..0][7..0] – pierwszy wymiar ([3..0]) dotyczy liczby magistral zaś drugi [7..0] określa rozmiar tych magistral,
- zapis BB[1][7..0] oznacza magistralę nr 1 o szerokości 8 bitów.

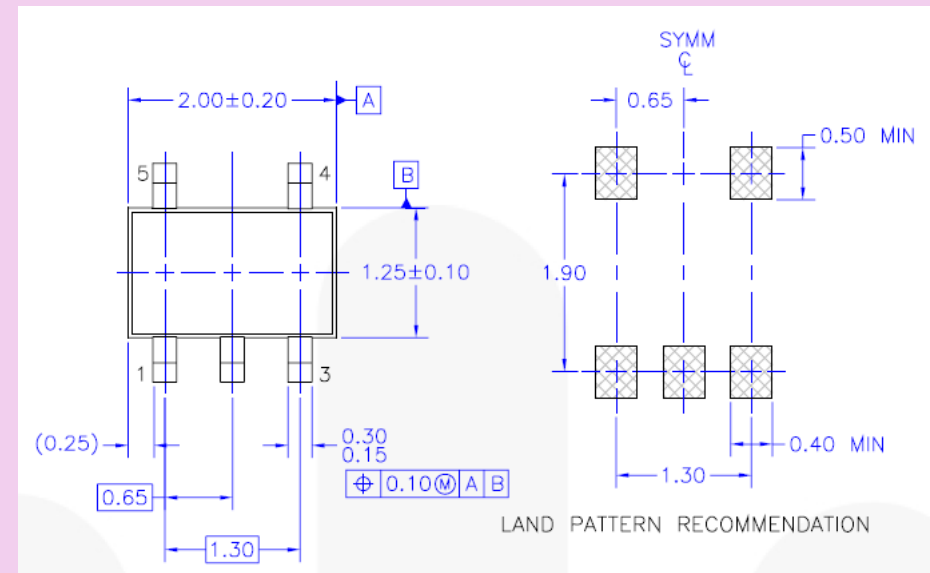
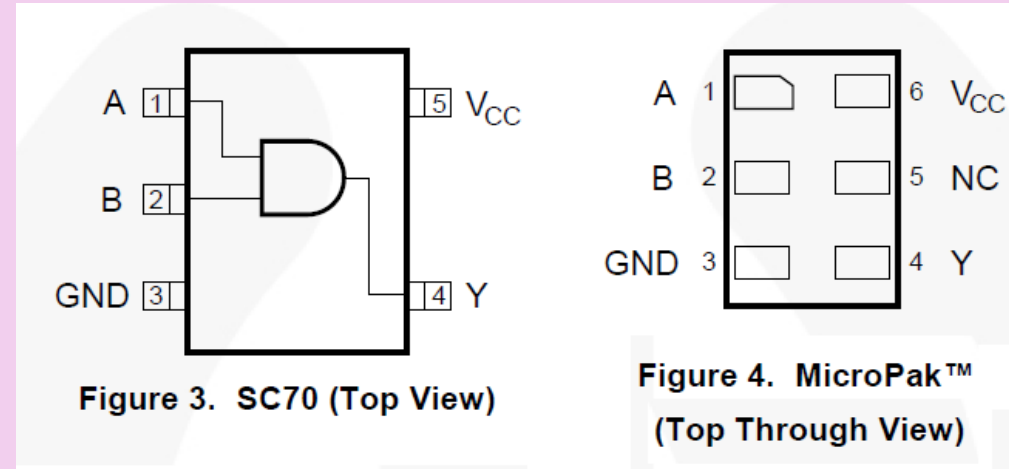


Parametry bramki AND:

Historia



Współczesność



Historia SN7408

recommended operating conditions								
		SN5408			SN7408			UNIT
		MIN	NOM	MAX	MIN	NOM	MAX	
V _{CC}	Supply voltage	4.5	5	5.5	4.75	5	5.25	V
V _{IH}	High-level input voltage	2			2			V
V _{IL}	Low-level input voltage			0.8			0.8	V
I _{OH}	High-level output current			−0.8			−0.8	mA
I _{OL}	Low-level output current			16			16	mA
T _A	Operating free-air temperature	−55		125	0		70	°C

Współczesność NC7SP08

Symbol	Parameter	Conditions	Min.	Max.	Unit
V _{CC}	Supply Voltage		0.9	3.6	V
V _{IN}	Input Voltage ⁽⁵⁾		0	3.6	V
V _{OUT}	Output Voltage	HIGH or LOW State	0	V _{CC}	V
		V _{CC} =0V	0	3.6	
I _{OH} / I _{OL}	Output Current in I _{OH} / I _{OL}	V _{CC} =3.0V to 3.6V		±2.6	mA
		V _{CC} =2.3V to 2.7V		±2.1	
		V _{CC} =1.65V to 1.95V		±1.5	
		V _{CC} =1.40V to 1.60V		±1.0	
		V _{CC} =1.10V to 1.30V		±0.5	
		V _{CC} =0.9V		20.0	μA
T _A	Free Air Operating Temperature		−40	+85	°C

Charakterystyki dynamiczne współczesnych bramek (NC7SP08) zależą od napięcia zasilania:

AC Electrical Characteristics										
Symbol	Parameter	V _{CC}	Conditions	T _A =+25°C			T _A =-40 to +85°C		Units	Figure
				Min.	Typ.	Max.	Min.	Max.		
t _{PHL} , t _{PLH}	Propagation Delay	0.90	C _L =10pF, R _L =1MΩ		27.0				ns	Figure 1, Figure 2
		1.10 ≤ V _{CC} ≤ 1.30		3.5	11.0	21.8	3.0	34.3		
		1.40 ≤ V _{CC} ≤ 1.60		2.5	7.0	14.8	2.0	15.0		
		1.65 ≤ V _{CC} ≤ 1.95		2.0	6.0	12.0	1.5	12.2		
		2.30 ≤ V _{CC} ≤ 2.70		1.5	5.0	9.4	1.0	9.9		
		3.00 ≤ V _{CC} ≤ 3.60		1.0	4.0	8.3	1.0	9.0		
		0.90	C _L =15pF, R _L =1MΩ		30.0					
		1.10 ≤ V _{CC} ≤ 1.30		4.0	11.0	22.8	3.5	37.3		
		1.40 ≤ V _{CC} ≤ 1.60		3.0	8.0	15.5	2.5	16.5		
		1.65 ≤ V _{CC} ≤ 1.95		2.5	6.0	12.6	2.0	13.6		
		2.30 ≤ V _{CC} ≤ 2.70		2.0	5.0	9.9	1.5	10.8		
		3.00 ≤ V _{CC} ≤ 3.60		1.5	4.0	8.7	1.0	9.5		
		0.90	C _L =30pF, R _L =1MΩ		32.0					
		1.10 ≤ V _{CC} ≤ 1.30		5.0	13.0	25.9	4.0	46.3		
		1.40 ≤ V _{CC} ≤ 1.60		4.0	9.0	17.8	3.5	18.2		
		1.65 ≤ V _{CC} ≤ 1.95		3.0	7.0	14.4	2.0	15.9		
		2.30 ≤ V _{CC} ≤ 2.70		2.0	6.0	11.3	1.5	12.8		
		3.00 ≤ V _{CC} ≤ 3.60		1.5	5.0	9.2	1.0	10.7		
C _{IN}	Input Capacitance	0			2				pF	
C _{PD}	Power Dissipation Capacitance	0.90 to 3.60	V _{IN} =0V or V _{CC} , f=10MHz		6				pF	

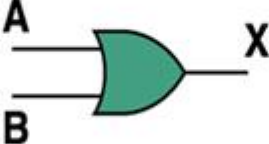
Bramka OR może posiadać jedno lub wiele wejść, jedno wyjście, na którym sygnał jest sumą logiczną sygnałów wejściowych. Zatem, że na wyjściu bramki wystąpi stan 0 – jedynie gdy na wszystkich wejściach jest również poziom niski 0. Przypadek 2-wejściowej bramki AND :

Funkcja boolowska

$X = A + B$

suma logiczna
sygnałów wejściowych

Schemat logiczny



Tablica prawdy

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

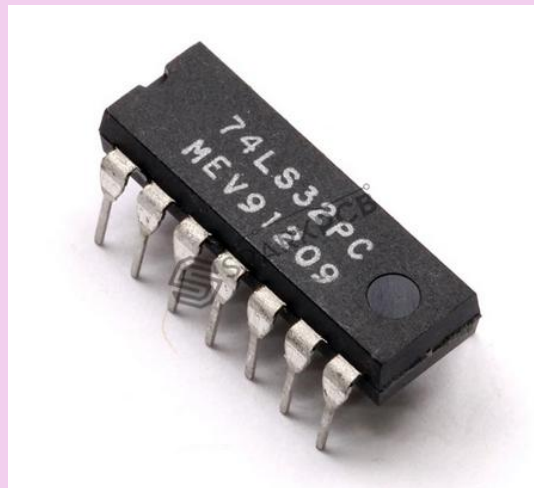
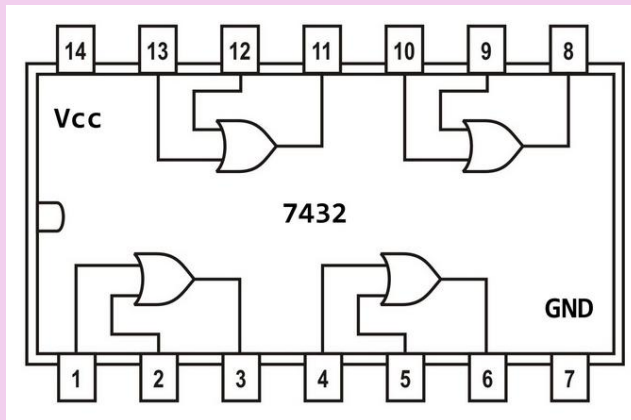
Szczególnym przypadkiem jest bramka szynowa, która wykonuje funkcję OR nie na pojedynczych bitach wejściowych – lecz na szynach danych – np. 2-wejściowa 4-bitowa OR:

	Name	Value at 0 ps	0 ps	80,0 ns	160,0 ns	240,0 ns	320,0 ns	400,0 ns	480,0 ns	560,0 ns	640,0 ns	700,0 ns
	▷ A	B 0000	0000	0000	0001	0010	0010	0011	0100	0101	0110	0110
	▷ B	B 0000	0000	0000	0010	0100	0100	0110	1000	1010	1100	1100
	▷ X	B 0000	0000	0000	0011	0110	0110	0111	1100	1111	1110	1110

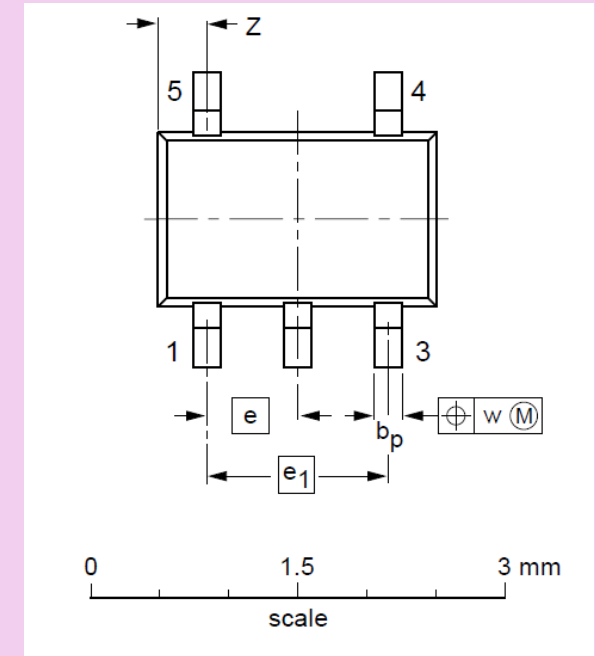
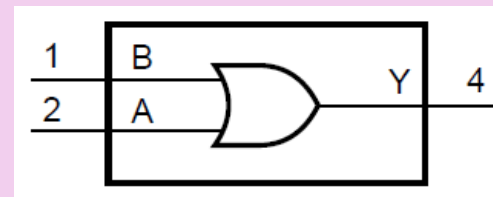
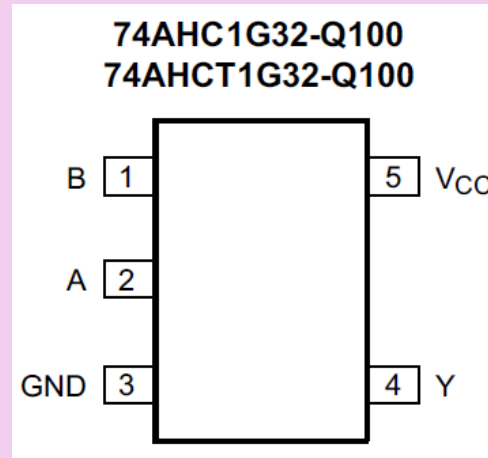
Poniższy listing przedstawia dwa kody VHDL :

- górny dotyczy dwuwejściowej jednobitowej bramki OR:
- dolny – bramki OR, której trzy wejścia stanowią magistrale czterobitowe.

Historia



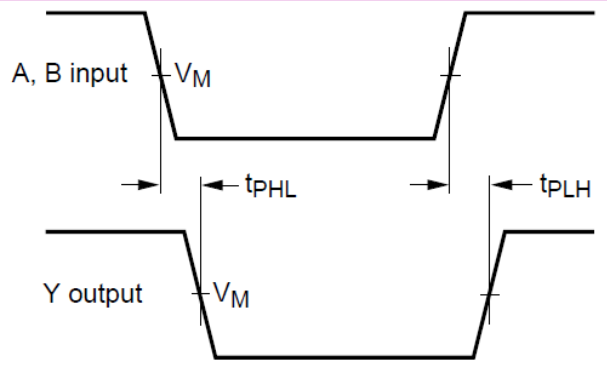
Współczesność



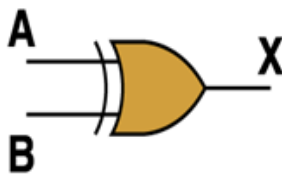
Wybrane parametry współczesnych bramek OR:

Table 6. Recommended operating conditions									
Symbol	Parameter	Conditions	74AHC1G32-Q100			74AHCT1G32-Q100			Unit
			Min	Typ	Max	Min	Typ	Max	
V _{CC}	supply voltage		2.0	5.0	5.5	4.5	5.0	5.5	V
V _I	input voltage		0	-	5.5	0	-	5.5	V
V _O	output voltage		0	-	V _{CC}	0	-	V _{CC}	V
T _{amb}	ambient temperature		-40	+25	+125	-40	+25	+125	°C

Table 8. Dynamic characteristics ...continued								
Symbol	Parameter	Conditions	25 °C			−40 °C to +85 °C		Unit
			Min	Typ	Max	Min	Max	
For type 74AHCT1G32-Q100								
t _{pd}	propagation delay	A and B to Y; see Figure 5 ; V _{CC} = 4.5 V to 5.5 V						
		C _L = 15 pF	-	3.3	6.9	1.0	8.0	ns
		C _L = 50 pF	-	4.8	7.9	1.0	9.0	ns



Bramka XOR może posiadać jedno lub wiele wejść, jedno wyjście, na którym sygnał jest sumą modulo 2 sygnałów wejściowych. Zatem, niezależnie od liczby wejść, obowiązuje zasada: na wyjściu bramki XOR pojawi się 1 jeżeli na nieparzystej liczbie wejść znajdują się 1.

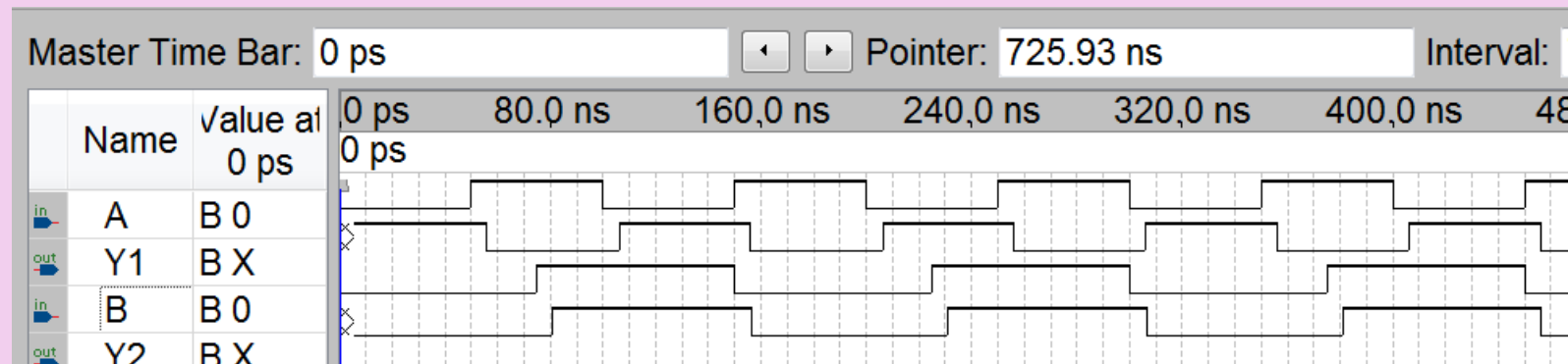
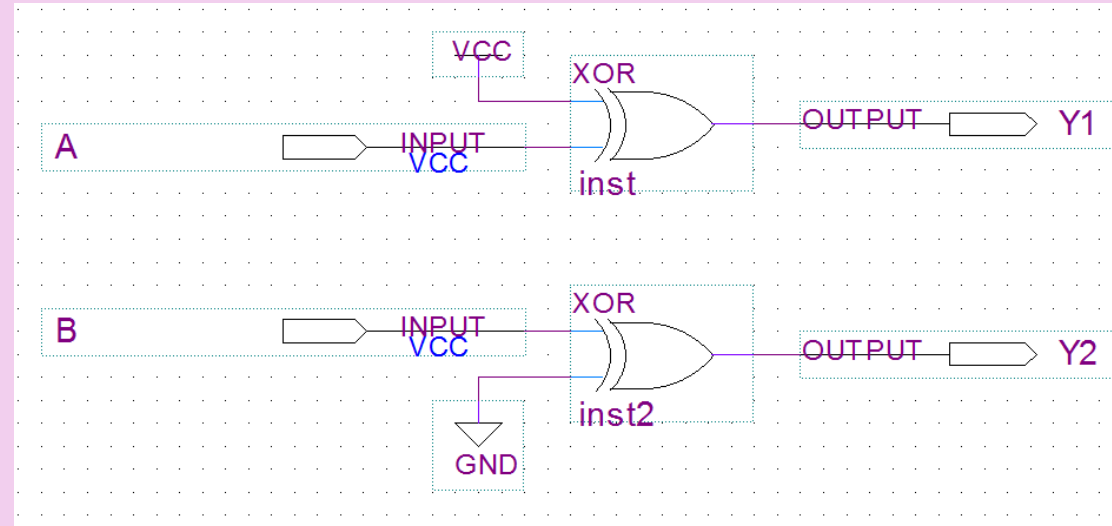
Funkcja boolowska	Schemat logiczny	Tablica prawdy															
$X = A \oplus B$ suma modulo dwa		<table> <tr> <th>A</th><th>B</th><th>X</th></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> </table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0
A	B	X															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

Opis bramki OR w kodzie VHDL:

```
X <= A xor B;
X <= A xor B xor C;
```

Jeżeli na jedno wejście dwuwejściowej bramki XOR podamy poziom wysoki (np. napięcie zasilania bramki VCC) – bramka pracuje jako negator względem drugiego wejścia.

Jeżeli na jedno wejście podany zostanie poziom niski (np. poziom masy) – względem drugiego wejścia układ zachowuje się jak bramka transmisyjna, tzn. niewpływająca na poziomy sygnał.



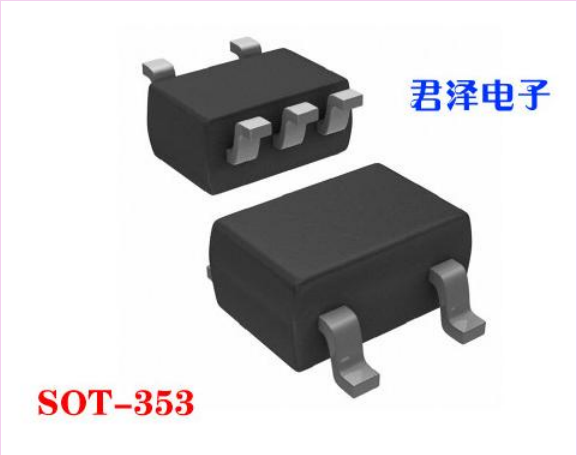
Poniższy listing przedstawia dwa kody VHDL :

- górny dotyczy dwuwejściowej jednobitowej bramki XOR:
- dolny – bramki XOR, której trzy wejścia stanowią magistrale czterobitowe.

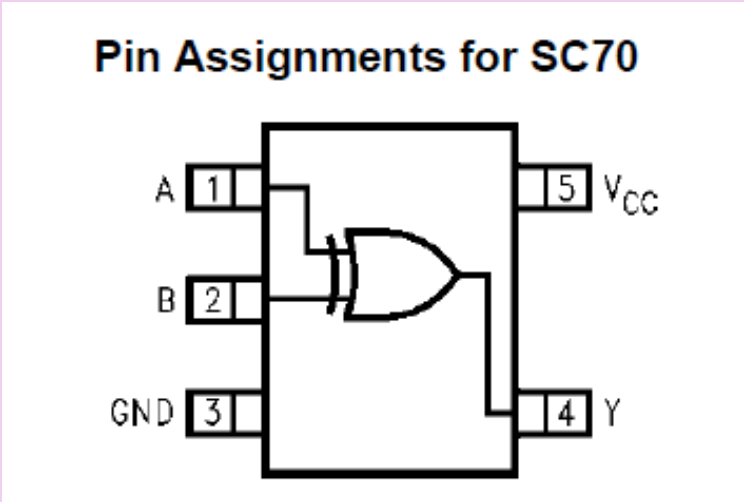
```
entity Bramka_jednobitowa is
|   port
|   (   A, B   : in  bit;
|       X       : out bit);
| end entity;
architecture rtl of Bramka_jednobitowa is
begin
|   X <= A xor B;
end rtl;

library ieee;           -- bramka o 3 wejściach szynowych
use ieee.numeric_std.all; --      4 - bitowych
entity Bramka_szynowa is
|   port
|   (   A, B, C : in  unsigned(3 downto 0);
|       X       : out unsigned(3 to 0));
| end entity;
architecture rtl of Bramka_szynowa is
begin
|   X <= A xor B xor C;
end rtl;
```


Parametry bramki XOR typu NC7SV865X

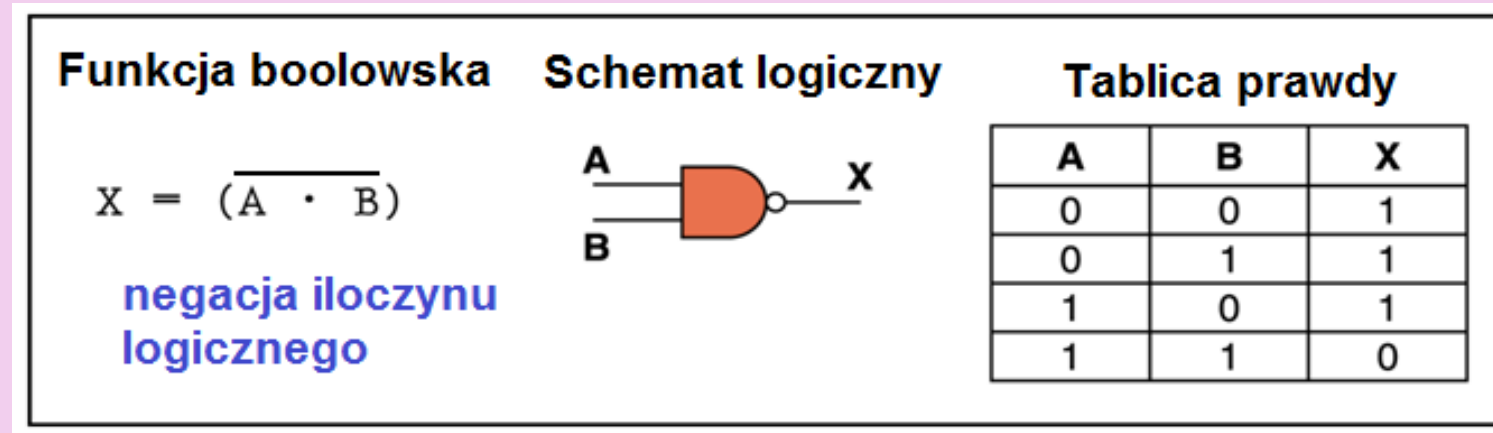


Supply Voltage	0.9V to 3.6V
Input Voltage (V_{IN})	0V to 3.6V
Output Voltage (V_{OUT})	
$V_{CC} = 0.0V$	0V to 3.6V
HIGH or LOW State	0V to V_{CC}

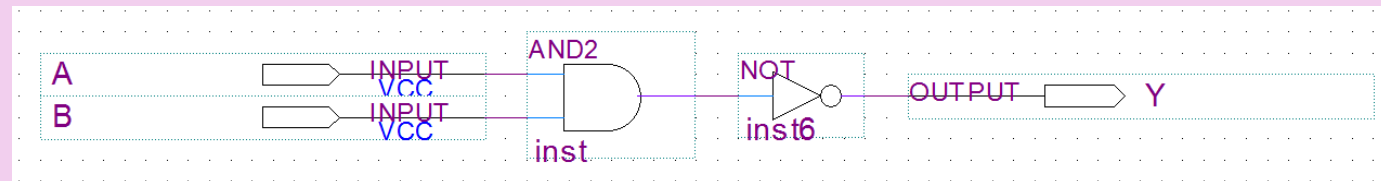


I_{IN}	Input Leakage Current	0.90 to 3.60	± 0.1	μA	$0 \leq V_I \leq 3.6V$
I_{OFF}	Power Off Leakage Current	0	0.5	μA	$0 \leq (V_I, V_O) \leq 3.6V$
I_{CC}	Quiescent Supply Current	0.90 to 3.60	0.9	μA	$V_I = V_{CC} \text{ or GND}$
		0.90 to 3.60			$V_{CC} \leq V_I \leq 3.6V$

Bramka NAND może posiadać jedno lub wiele wejść, jedno wyjście, na którym poziom sygnału jest negacją iloczynu logicznego sygnałów wejściowych. W tym sensie jest ona zaprzeczeniem bramki AND.



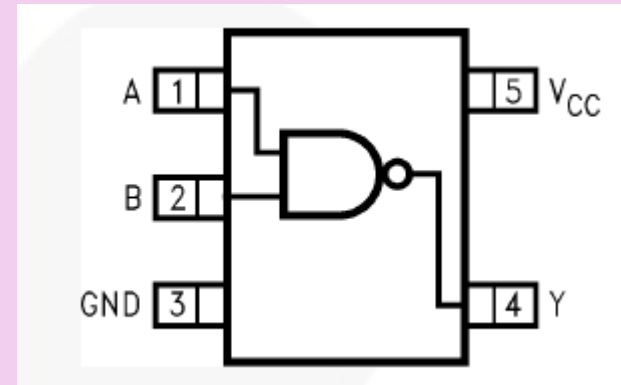
Równoważnym funkcjonalnie układem bramki NAND jest połączenie bramek AND i NOT:



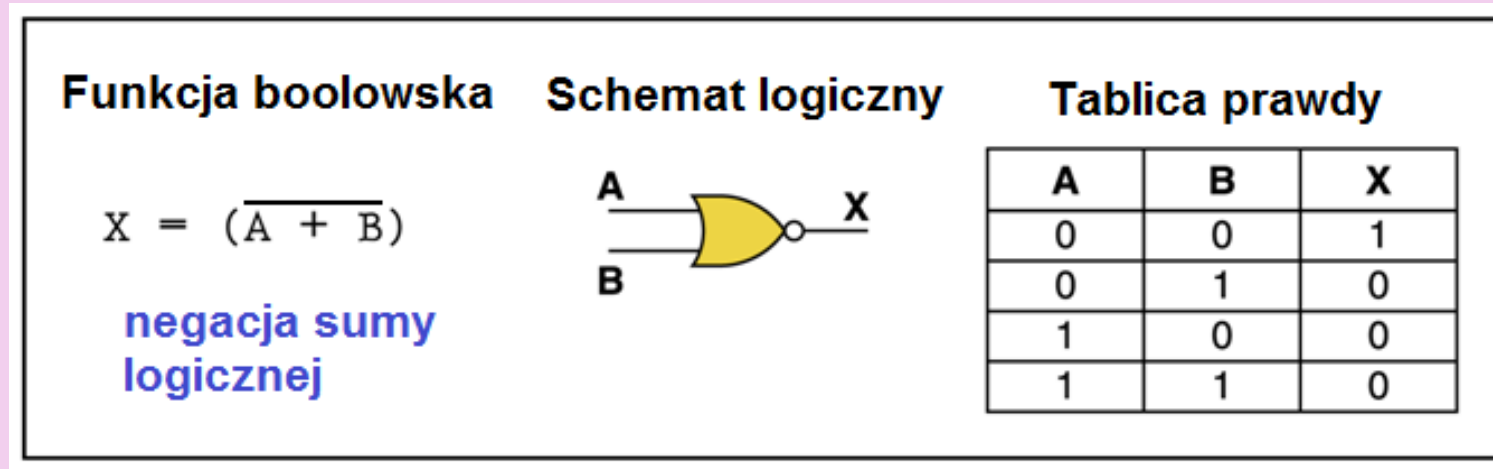
```
Y <= A nand B;
Y <= not (A and B);
```

Cechy bramek NAND w nowej technologii:

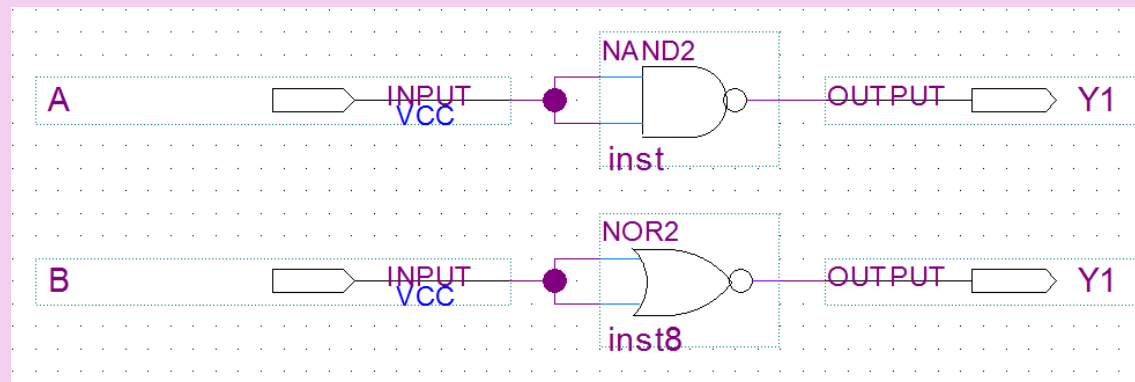
- bardzo krótkie czasy propagacji około 3 ns (tradycyjne np. SN74AS00N kilkanaście ns),
- wysoka obciążalność około 24 mA / pin (tradycyjne rzędu 10/1 mA),
- szeroki zakres napięć zasilających od 1.65 V do 5.5 V (tradycyjne od 4.5 V do 5.5 V),
- tolerancja przepięć wejściowych w umożliwiającą translację standardów 5V i 3 V,
- miniaturowe obudowy MicroPack, SOT23 i SC70,
- mały pobór prądu zasilania rzędu 20 uA (tradycyjne do około 20 mA).



Bramka NOR może posiadać jedno/wiele wejść, wyjście, którego poziom jest negacją sumy logicznej sygnałów wejściowych. W tym sensie jest ona zaprzeczeniem bramki OR.



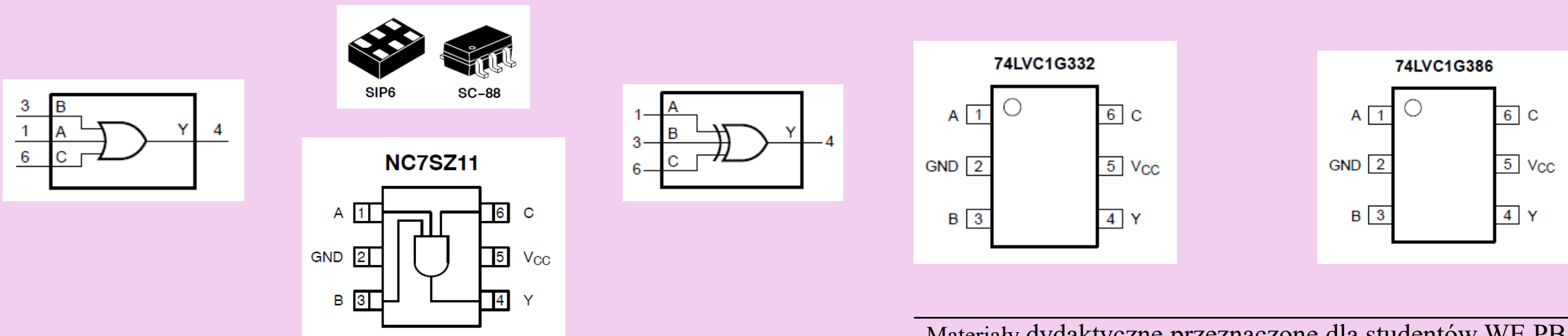
Wielowejściowe bramki negujące sygnały (NAND, NOR) mogą pełnić funkcje bramek negacji :



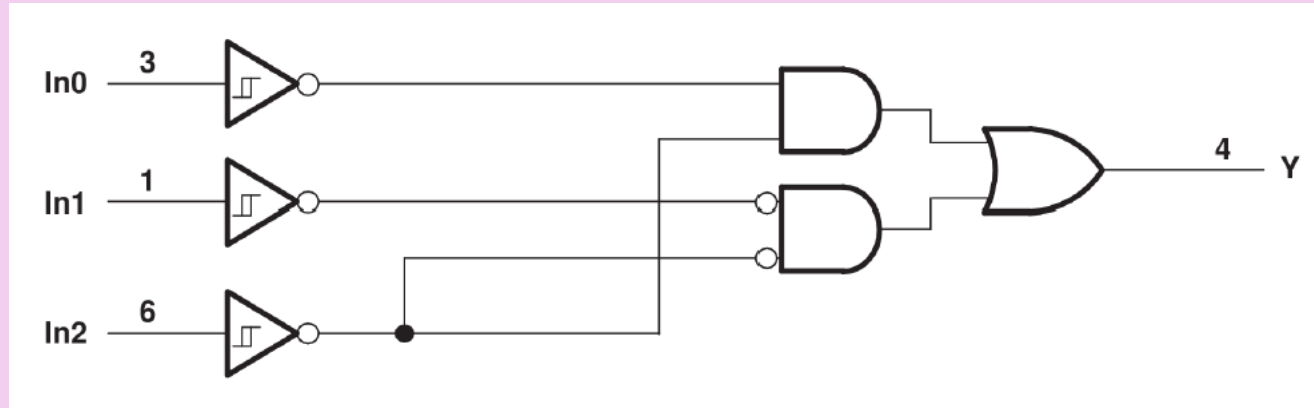
Przykład bramki NOR NC7SZ02 o ultra niskim prądzie zasilającym około 10 uA i szerokim zakresie napięć zasilania/pracy – od 1.65 V do 5.5 V. Pozostałe parametry są zbliżone do wartości charakterystycznych dla tej klasy układów.



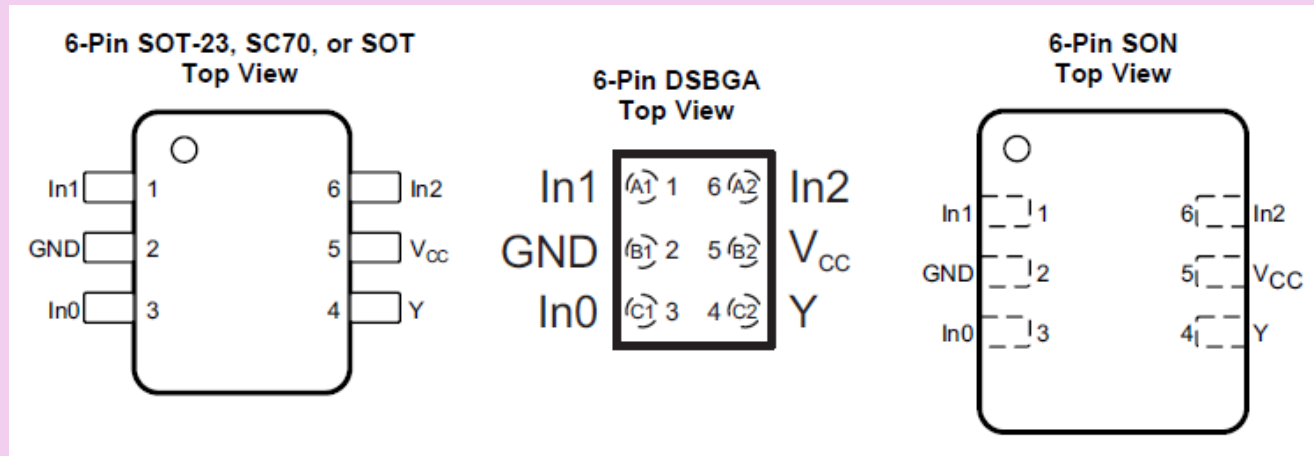
Oprócz wymienionych bramek dwuwejściowych – dostępne są aplikacje wielowejściowe:



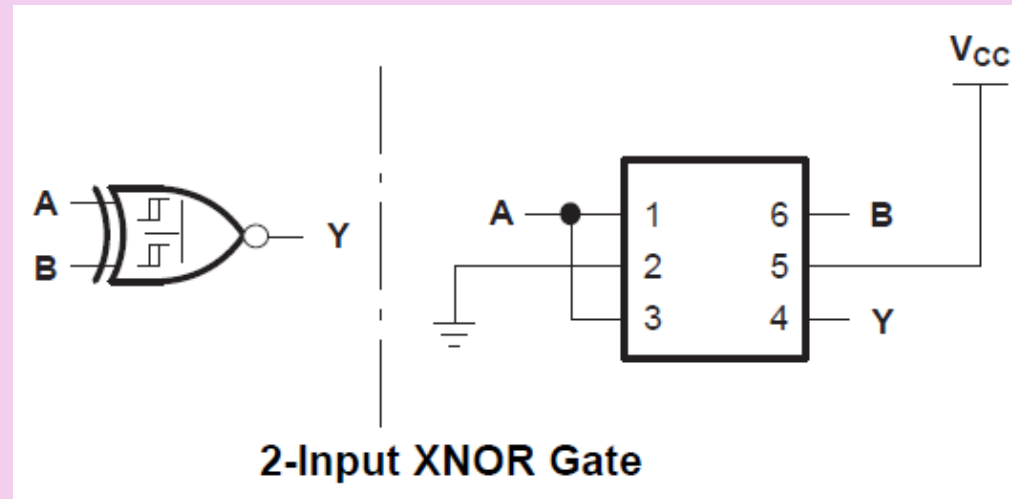
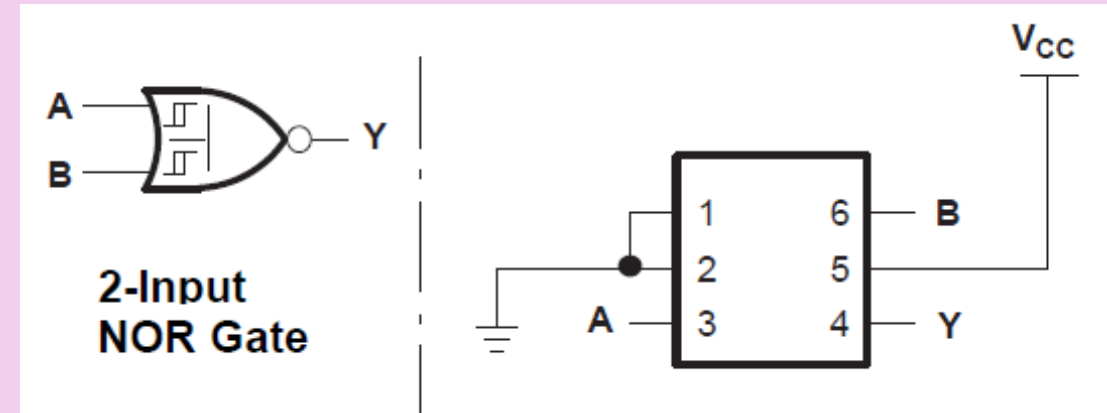
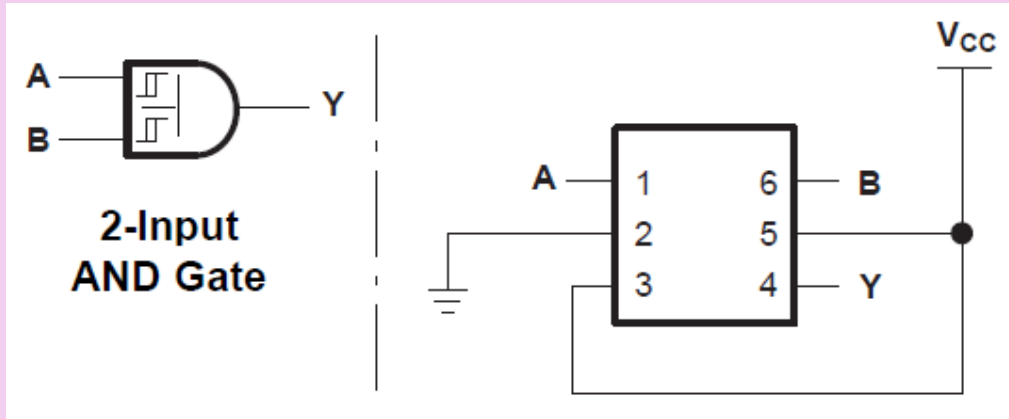
Istnieje kategoria współczesnych bramek, których funkcję można konfigurować. Przykładem takiego rozwiązania jest układ SN74LVC1G57.



Istnieje kategoria współczesnych bramek, których funkcję można konfigurować. Przykładem takiego rozwiązania jest układ SN74LVC1G57.



Przykładowe topologie połączeń bramki konfigurowalnej SN74LVC1G57.

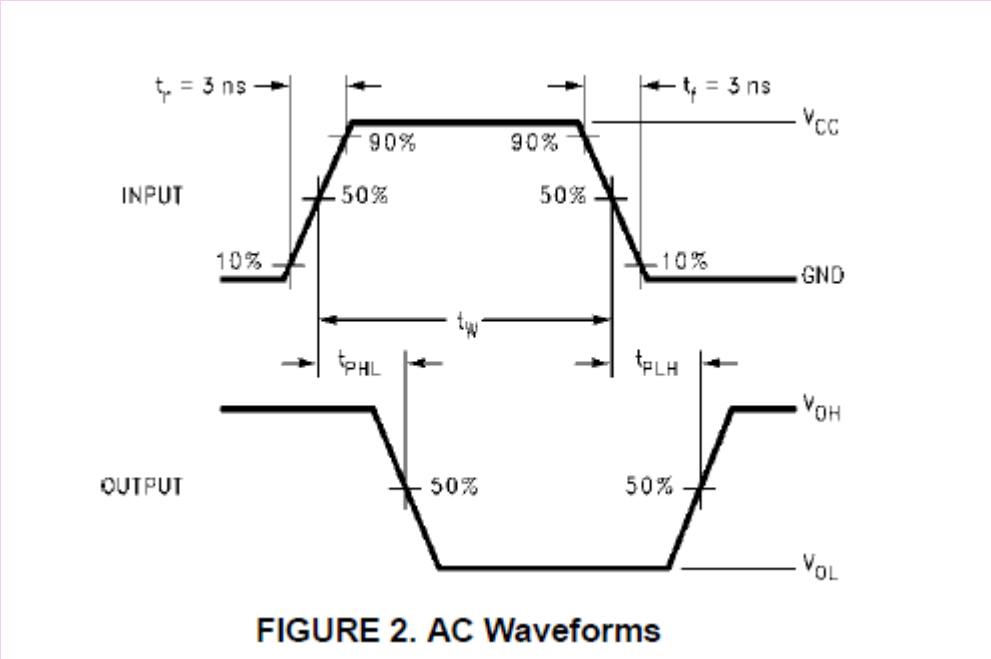


Oprócz zasilania i wartości napięć wejściowych, podczas projektowania układów cyfrowych ważne są ich wymagania dynamiczne podane w kartach katalogowych (ON Semi, Diodes Inc.)

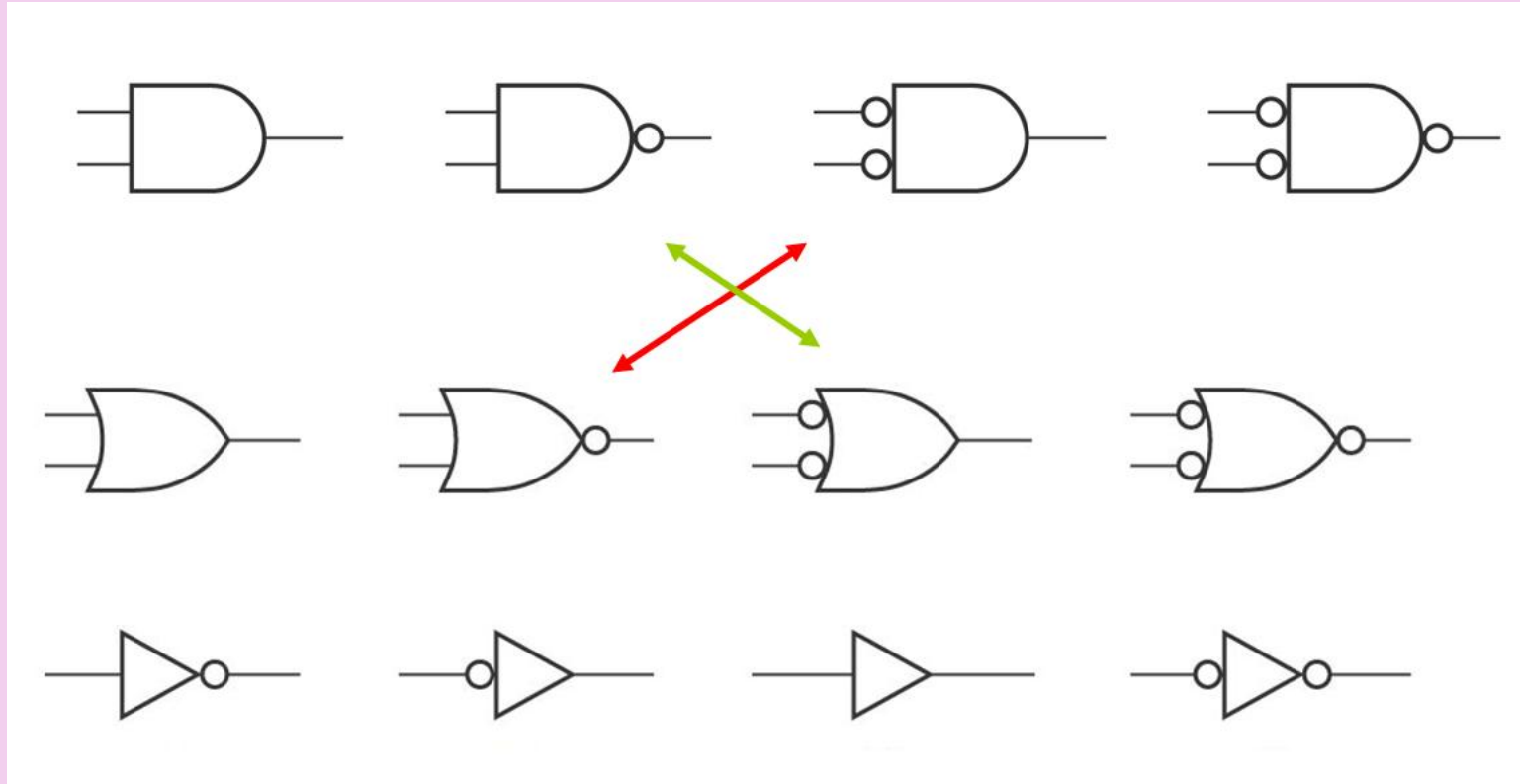
Minimum Input Edge Rate ($\Delta t/\Delta V$)
 $V_{IN} = 0.8V$ to $2.0V$, $V_{CC} = 3.0V$ 10 ns/V

t_r, t_f	Input Rise and Fall Times	V_{CC} at 1.8V, 2.5V $\pm$ 0.2V	0	20	ns/V
		V_{CC} at 3.3V $\pm$ 0.3V	0	10	
		V_{CC} at 5.0V $\pm$ 0.5V	0	5	

$\Delta t/\Delta V$	Input transition rise or fall rate	$V_{CC} = 1.8V \pm 0.15V, 2.5V \pm 0.2V$		20	ns/V
		$V_{CC} = 3.3V \pm 0.3V$		10	
		$V_{CC} = 5V \pm 0.5V$		5	



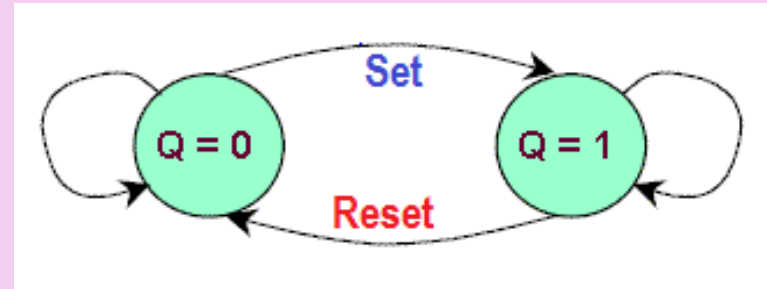
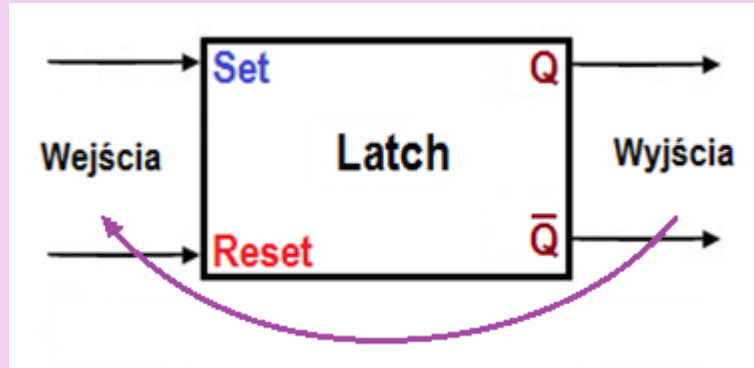
W dostępnej literaturze można spotkać dużą różnorodność (np. IEC/IEEE) w zakresie stosowanych symboli, nie przyjęła się w tym zakresie jednoznaczna standaryzacja.



Przerzutnik: który zapisuje określony stan/poziom na swoim wyjściu. Skoro przechowuje on wartość 0 i 1 - mówimy, że posiada 2 stany stabilne i może być użyty do pamiętania danych. Występują 2 typy przerzutników:

- przerzutniki asynchroniczne,
- przerzutniki synchroniczne.

Pierwsze przerzutniki asynchroniczne Set - Reset, opracowano na początku XX wieku:

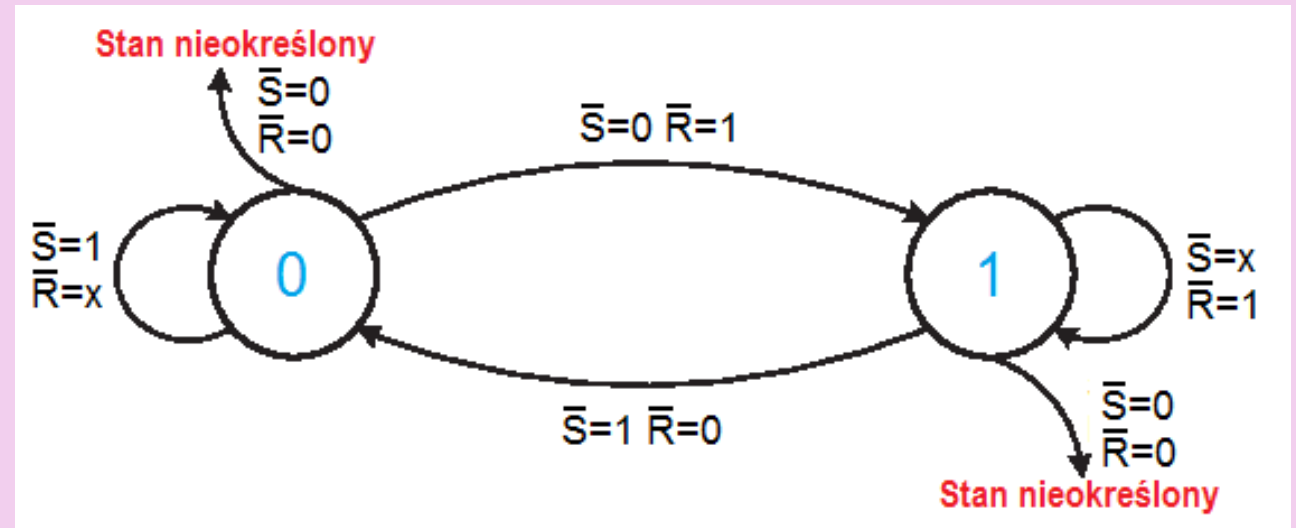
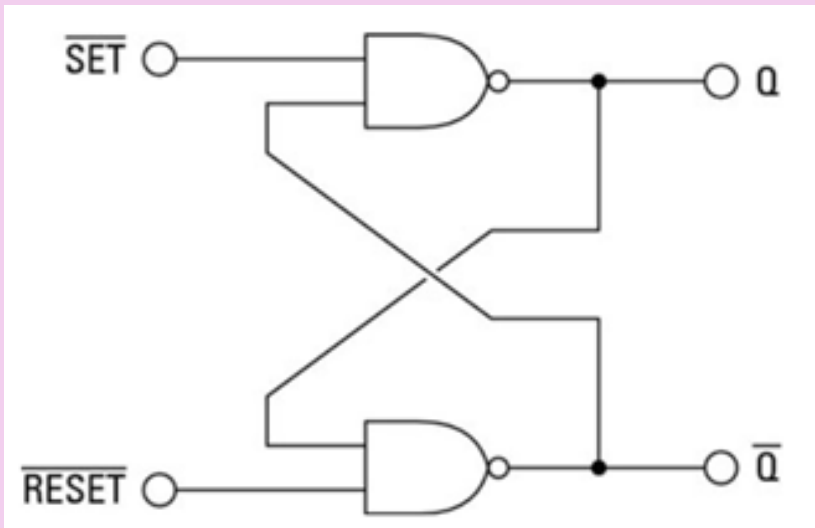


Działa on jak układ magazynujący 1-bitowe dane w torze sprzężenia zwrotnego, czasu zapisu nowych danych. Po włączeniu zasilania przerzutnik może zmienić stan, zanim nie ustabilizuje się sygnał wejściowy. Układ może być wyzwalany/ustawiany niskim albo wysokim poziomem – stąd występują dwie wersje tego układu oznaczane odpowiednio jako $\hat{S}\hat{R}$ lub RS - znak $\hat{}$ nad literą oznacza negację wartości ze względu na ułomność edytora, albo aktywny sygnał niski. Czasami zanegowany sygnał Q pisze się jako: **not Q** lub **Q-bar**.

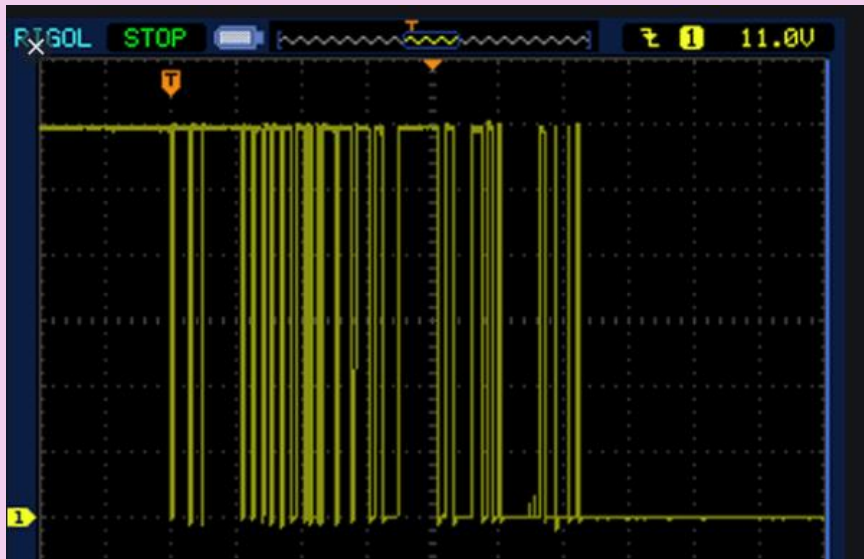
Przerzutnik $\hat{S}\hat{R}$ (nie S, nie R) wyzwalany jest poziomem niskim na wejściach, może być zbudowany z bramek NAND – stąd czasami nazywa się go SR NAND Latch, jest to nazwa nieprecyzyjna ale czasami stosowana.

Układ pracuje w następujący sposób:

- wyjście Q przechodzi z '0' do stanu wysokiego gdy wejścia: Set($\bar{S}$) = '0' i Reset($\bar{R}$) = '1',
 - wyjście Q przechodzi z '1' do stanu niskiego gdy wejścia: Set($\bar{S}$) = '1' i Reset = '0',
 - układ pozostaje w stanie niskim na wyjściu gdy wejście Set($\bar{S}$) = '1' - tabela,
 - układ pozostaje w stanie wysokim na wyjściu gdy wejście Reset($\bar{R}$) = '1' - tabela,
- zabronione jest jednoczesne ustawianie: Set($\bar{S}$) = Reset($\bar{R}$) = '0' - grozi niestabilnością pracy.

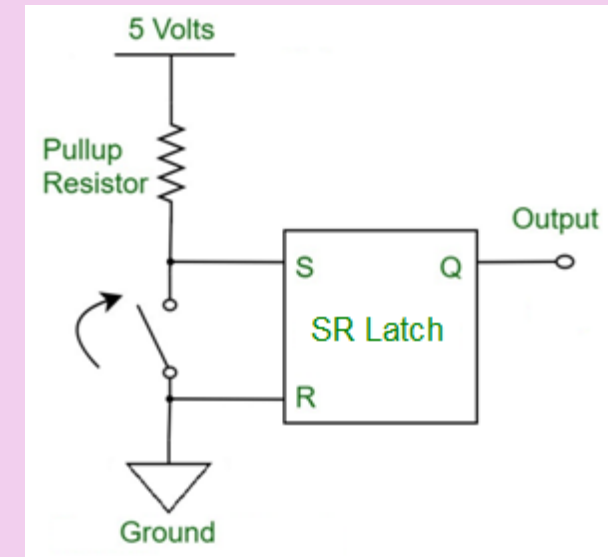


Przerzutnik SR NAND jest stosowany min. w układach eliminacji drgań styków. Każdy mechaniczny przycisk lub styki w czasie zwierania i rozwierania mają tendencję do drgań (bouncing), czyli przez pewien czas przełączają w obie pozycje – zanim ustabilizują. Zjawisko ma podłoże czysto mechaniczne i trwa od kilkudziesięciu mikrosekund do kilkudziesięciu milisekund, w zależności od konstrukcji styku i jest ono losowe.

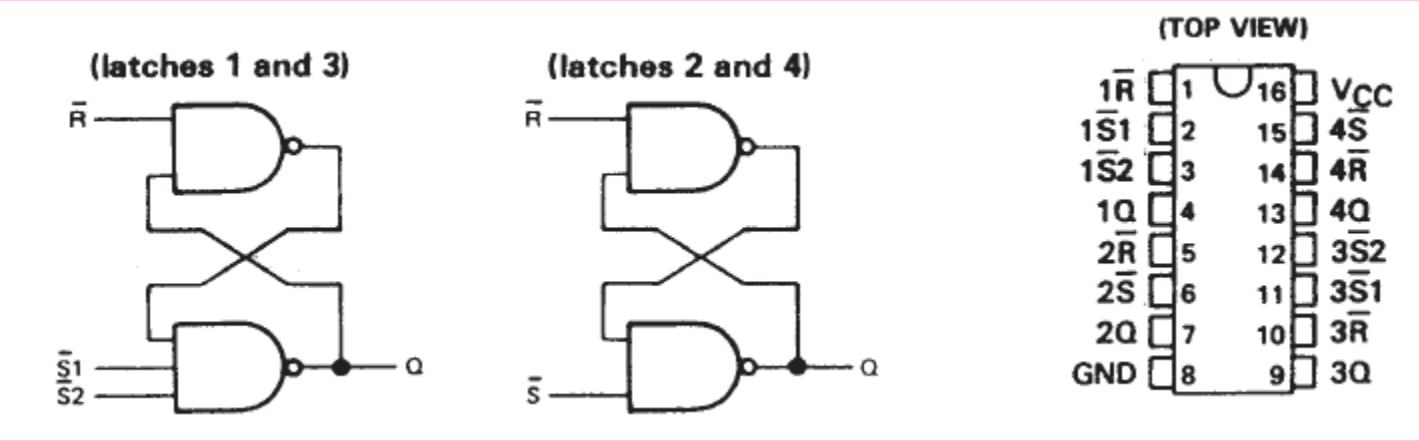


■ Ratings/Characteristics

Ratings	1 to 50 mA, 5 to 24 VDC
Ambient operating temperature	-25°C to +70°C at 60%RH max.
Ambient operating humidity	35% to 85% (at +5 to +35°C)
Contact form	SPST-NO
Contact resistance	100 mΩ max. (rated: 1 mA, 5 VDC)
Insulation resistance	100 MΩ min. (at 250 VDC)
Dielectric strength	500 VAC, 50/60 Hz for 1 min
Bounce time	5 ms max.
Durability	1,000,000 operations min (0.98 N) 300,000 operations min (1.47 N) 100,000 operations min (2.55 N) 50,000 operations min (4.9 N)



Dyskretne przerzutniki asynchroniczne $\hat{S}\hat{R}$ częściowo są już wycofywane ze sprzedaży – np. SN74LS279A.



recommended operating conditions

	SN54279			SN74279			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
V_{CC} Supply voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH} High-level input voltage	2			2			V
V_{IL} Low-level input voltage			0.8			0.8	V
I_{OH} High-level output current			-0.8			-0.8	mA
I_{OL} Low-level output current			16			16	mA
t_w Pulse duration, low	20			20			ns
T_A Operating free-air temperature	-55		125	0		70	°C

switching characteristics, $V_{CC} = 5\text{ V}$, $T_A = 25^\circ\text{C}$

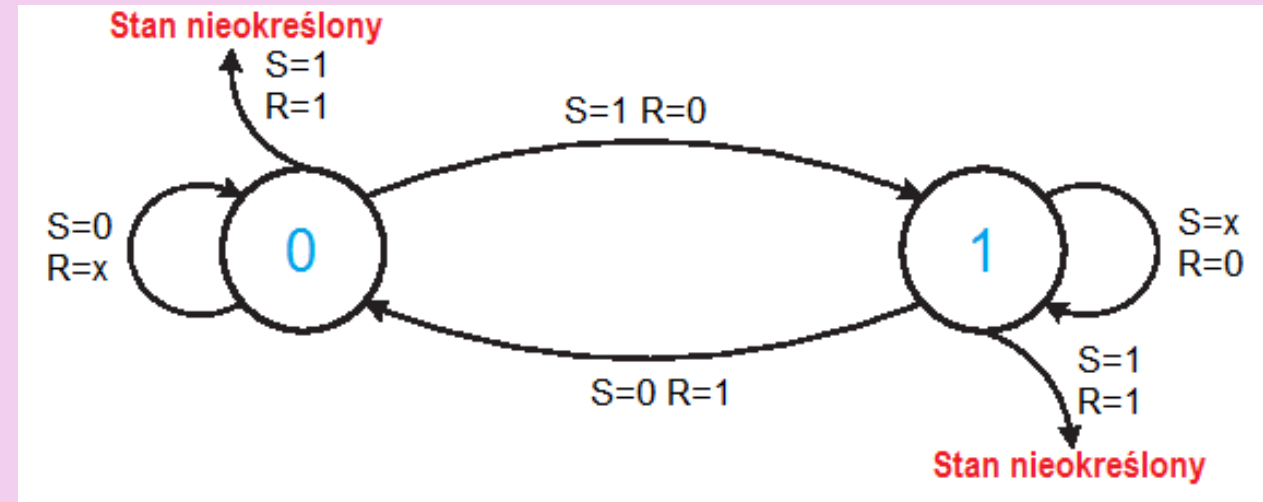
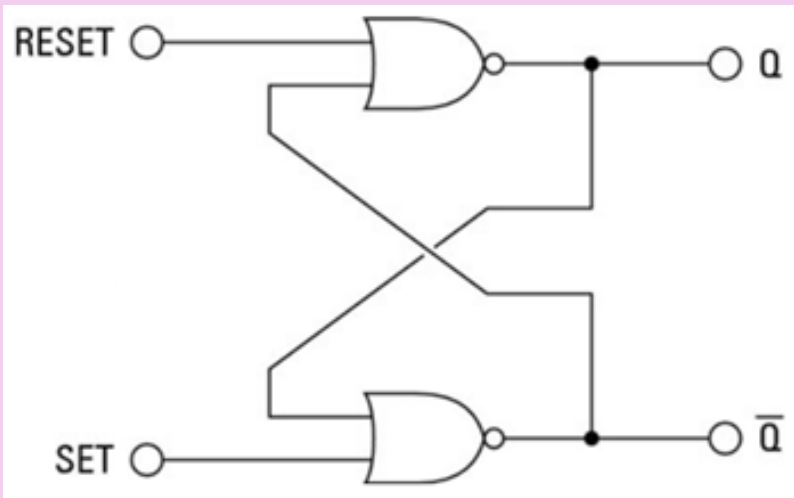
PARAMETER	FROM (INPUT)	TO (OUTPUT)	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	$\bar{S}$	Q	$R_L = 400\ \Omega$, $C_L = 15\text{ pF}$		12	22	ns
t_{PHL}					9	15	
t_{PHL}	$\bar{R}$	Q			15	27	ns

Drugą wersją przerzutnika asynchronicznego jest układ latch **SR**, w odróżnieniu od poprzednika, ten układ jest on wyzwalany poziomem wysokim, zbudowany jest z bramek NOR i nazywany SR NOR Latch).

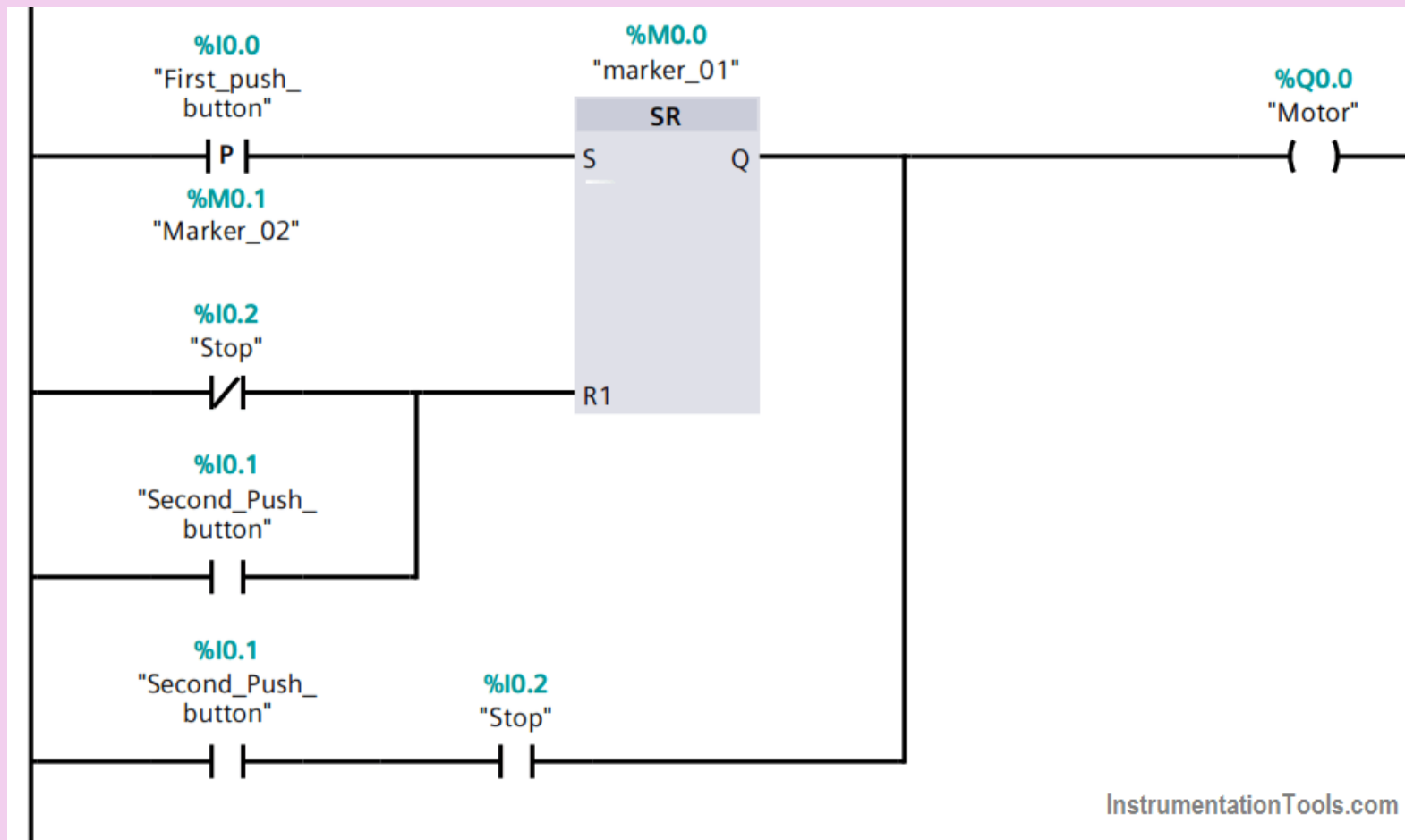
Pracuje on w następujący sposób:

- wyjście Q przechodzi do stanu wysokiego gdy wejścia: Set(S) = '1' i Reset(R) = '0',
- wyjście Q przechodzi do stanu niskiego gdy wejścia: Set = '0' i Reset = '1',
- układ pozostaje w stanie niskim na wyjściu gdy wejście Set = '0',
- układ pozostaje w stanie wysokim na wyjściu gdy wejście Reset = '0',

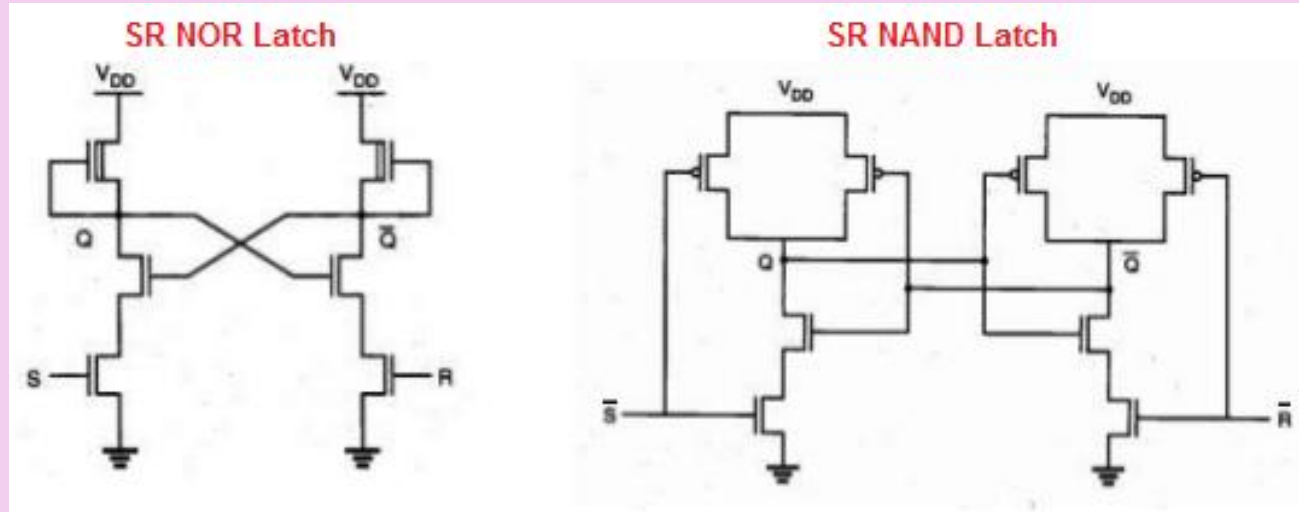
zabronione jest jednoczesne ustawianie: Set($\check{S}$) = Reset($\check{R}$) = '1' - grozi niestabilnością pracy.



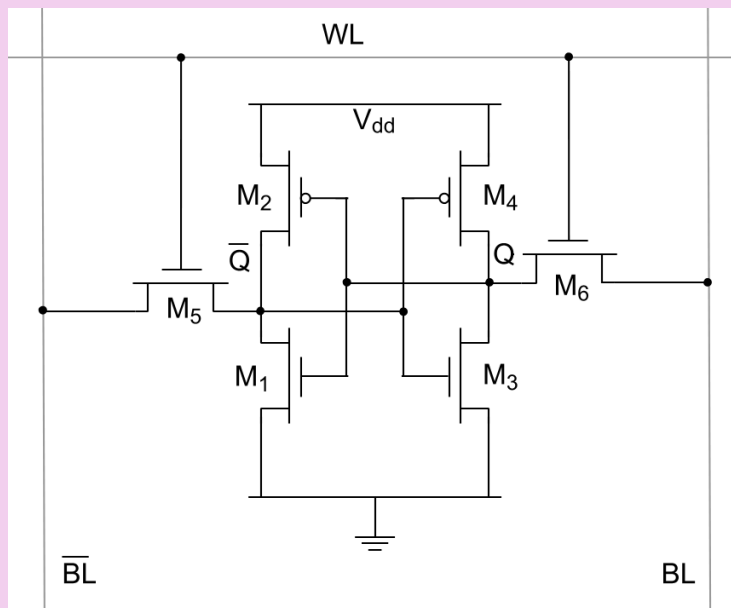
Zatrzaszaki RS miały zastosowanie min. w sterownikach, włączających i wyłączających silniki za pomocą jednego przycisku (<https://instrumentationtools.com/latched-and-momentary-operation-of-motor/>).



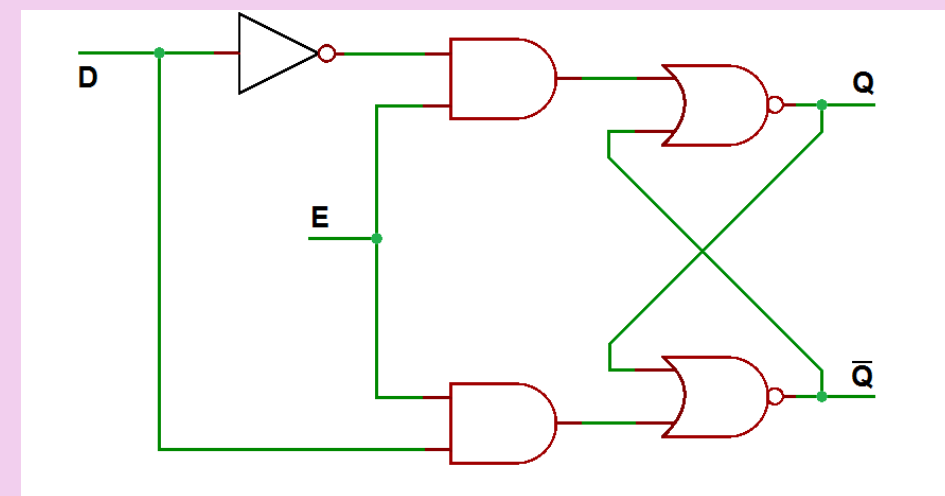
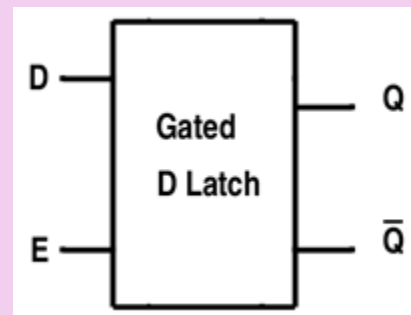
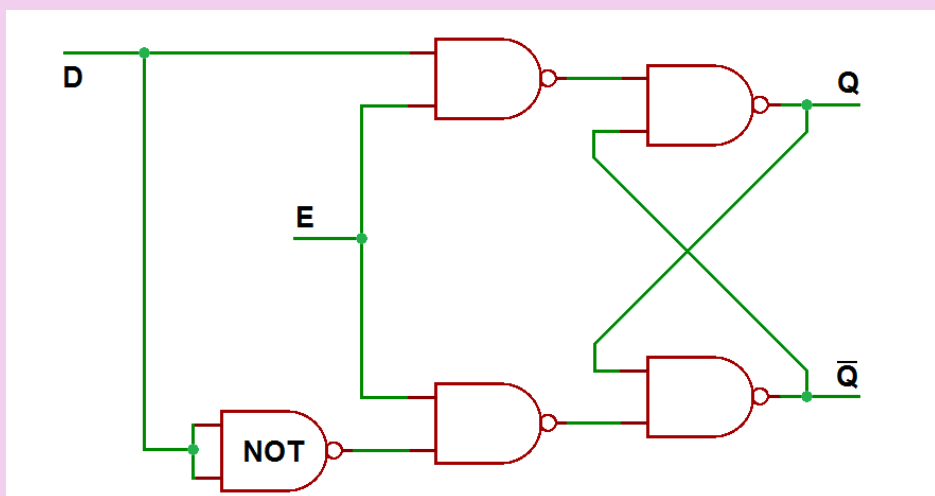
Tranzystorowe struktury przerzutników asynchronicznych stosowane są w komórkach SRAM.



Struktura części asynchronicznej komórki statycznej RAM:

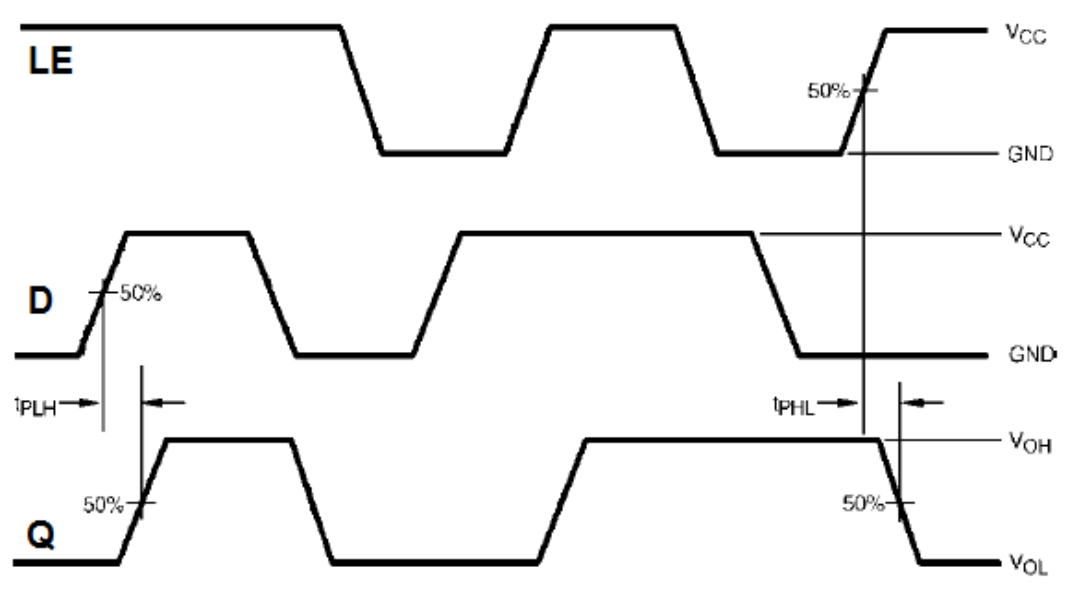
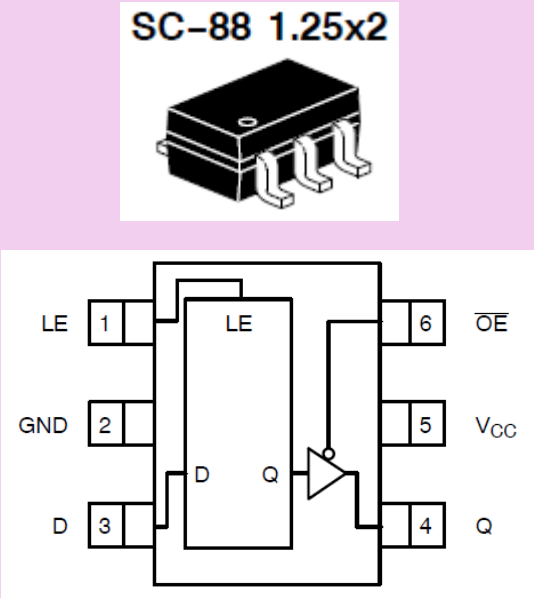


Przerzutniki asynchroniczne mogą być zawodne i nieprzewidywalne, jeżeli w zależności od konfiguracji oba wejścia przyjmą stan niski lub wysoki. Stąd układ zmodyfikowano, żeby jednoczesna zmiana sygnałów S i R była niemożliwa - w efekcie powstał układ o jednym wejściu sygnałowym typu D zatrzask (D Latch).



Możliwe są dwie topologie bramkowanego D zatrzask: NOR i NAND. Druga Topologia jest prostsza, wymaga tylko bramek NAND. Układ w stanie '1' na wejściu E jest "przezroczysty" dla sygnału D, zapamiętywany jest poziom D przy przejściu E z '1' do '0' - stąd nazwa zatrzask.

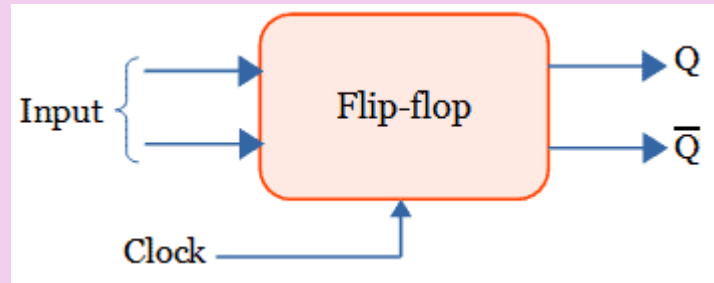
Przykładowa realizacja przerzutnika D-zatrzask typu NC7SZ373:



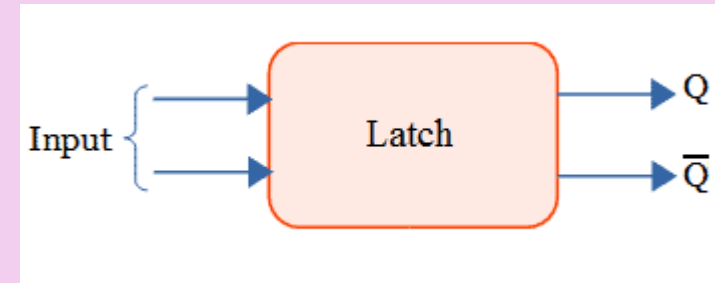
AC ELECTRICAL CHARACTERISTICS

Symbol	Parameter	VCC (V)	Conditions	TA = +25°C			TA = -40 to +85°C		Unit
				Min	Typ	Max	Min	Max	
tPLH tPHL	Propagation Delay D to Q	1.65	CL = 15 pF, RD = 1 MΩ,	–	9.0	15.0	–	16.0	ns
		1.8		–	6.1	10.0	–	10.5	

Przerzutnik synchroniczny (ang. Flip-Flop, Register) - jest elementem do przechowywania 1-bitowej informacji, który aktualizuje swój stan wyjściowy **pod wpływem sygnału zegarowego**.

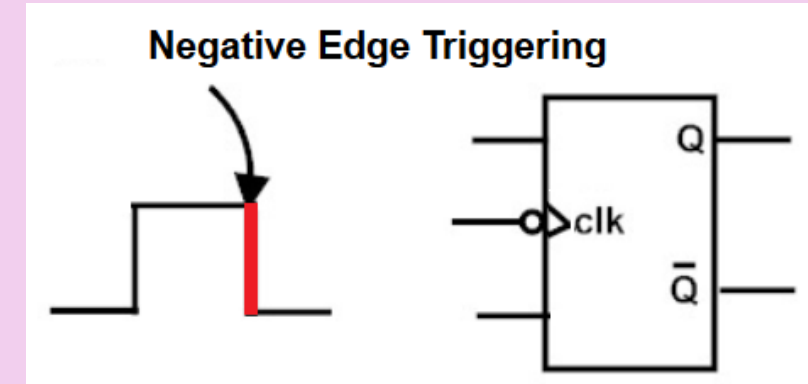
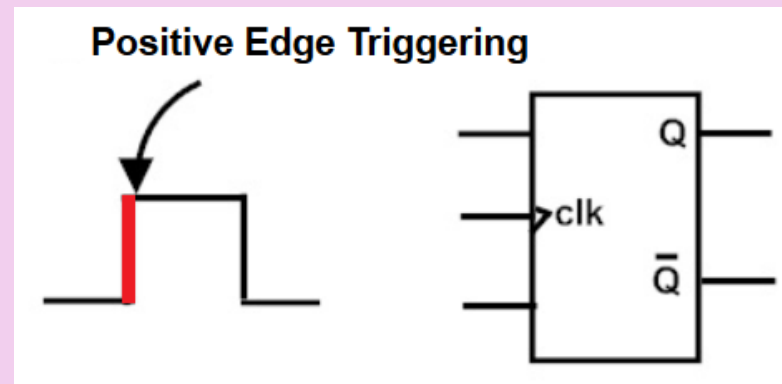
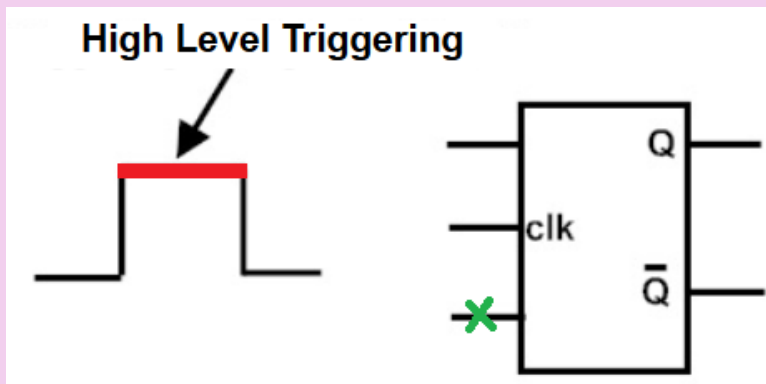


- dodatkowy sygnał zegarowy jest niezbędny,
- zmienia stan wyjścia tylko podczas zmiany sygnału zegarowego,
- reaguje na poziomy **wejść i sygnał zegarowy**,
- szybkość wyznaczają czasy propagacji oraz częstotliwość sygnału zegarowego,
- może być budowany z zatrzasków,
- miarą szybkości jest częstotliwość zegara,
- stosowane rodzaje: D, JK, SR, T.



- nie wymaga sygnału zegarowego,
- jest transparentny gdy sygnał zezwolenia = '1', zapisuje wartość w momencie powrotu do '0',
- reaguje na zmiany sygnałów wejściowych,
- szybkość pracy określa czas propagacji,
- kontrolowany jest tylko sygnałem zezwolenia,
- może być zbudowany z bramek,
- miarą szybkości są czasy opóźnień,
- stosowane rodzaje: **SR** i **SR**.

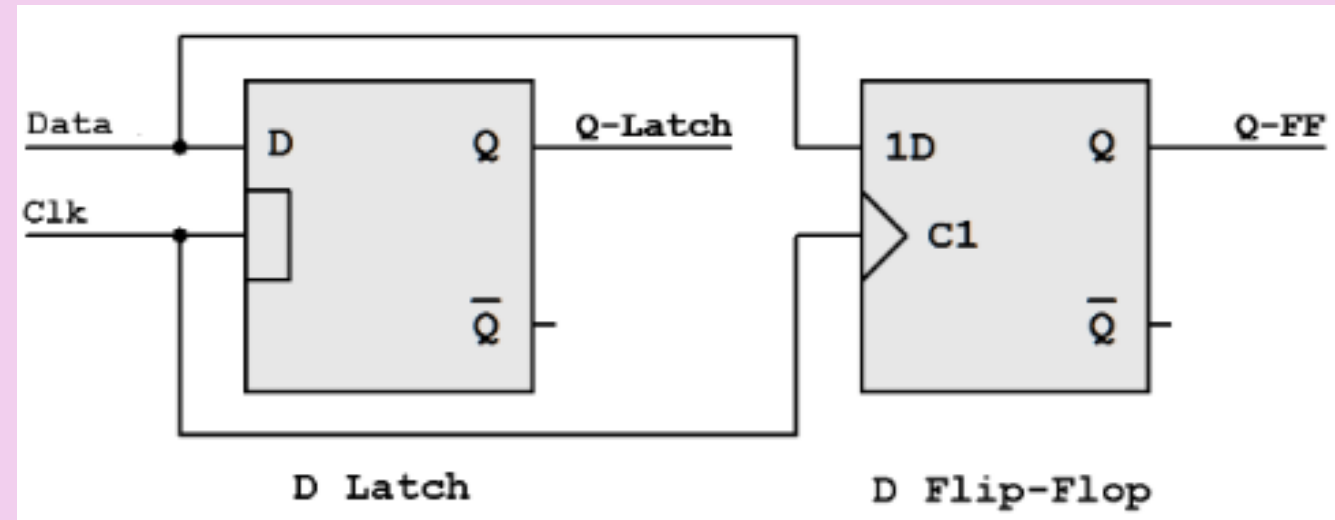
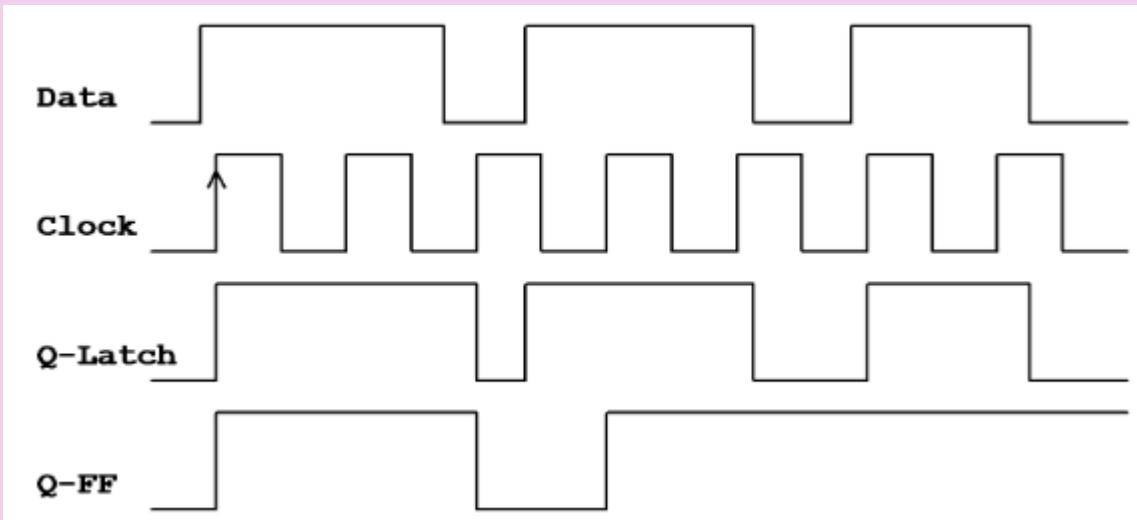
Zmianę stanu przerzutnika synchronicznego dokonuje się za pomocą impulsu zegarowego – czyli wpisanie nowej wartości lub podtrzymanie tego samego sygnału na wyjściu przerzutnika. Istnieją dwa rodzaje wyzwalania: wyzwalanie poziomem i wyzwalanie zboczem/zboczami. W wyzwalaniu poziomem zegar pełni podobną funkcję jak sygnał Enable w przerzutniku latch – często te rozwiązania są utożsamiane chociaż różnią je parametry. Wyzwalanie poziomem jest procesem w którym, zmiana stanu wyjściowego następuje podczas aktywnego/wysokiego poziomu sygnału zegarowego, w czasie którego sygnał z wejścia/wejść przepływa na wyjście – zapamiętywanie/zatrzaśnięcie następuje gdy impuls zegarowy przechodzi w stan niski:



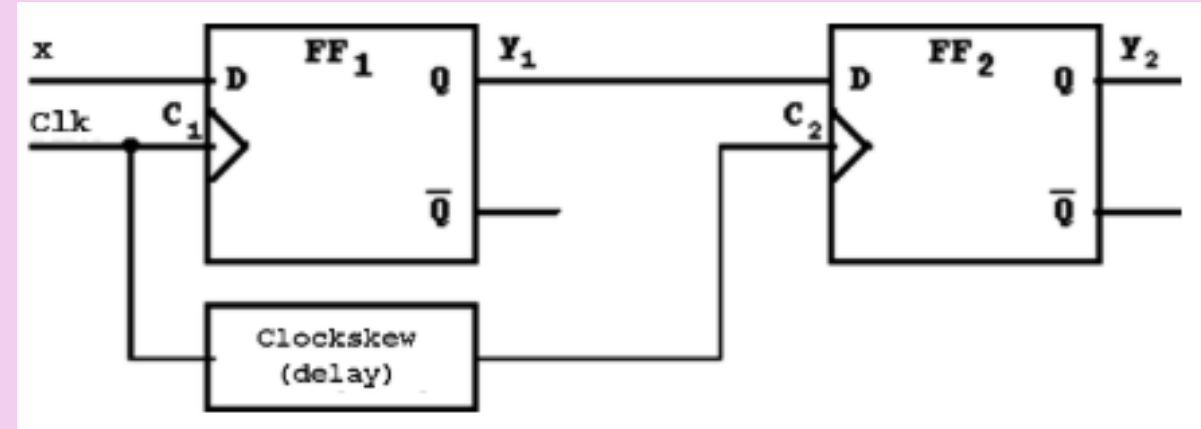
Wyzwalanie zboczem przerzutników synchronicznych jest najczęściej stosowane. Jest to odporny na zakłócenia sposób, gdyż proces zapisu nowych danych jest krótki – trwa tyle co aktywne zbocze sygnału zegarowego. Zboczem aktywnym/wyzwalającym może być zbocze narastające lub opadające. Najczęściej przerzutniki wyzwalają się zboczem narastającym (Positive/Rising Edge Triggering). Proszę zwrócić uwagę na symbole wejść.

Różnice między zatrzaskami a przerzutnikami (Latch/Flip-flop difference):

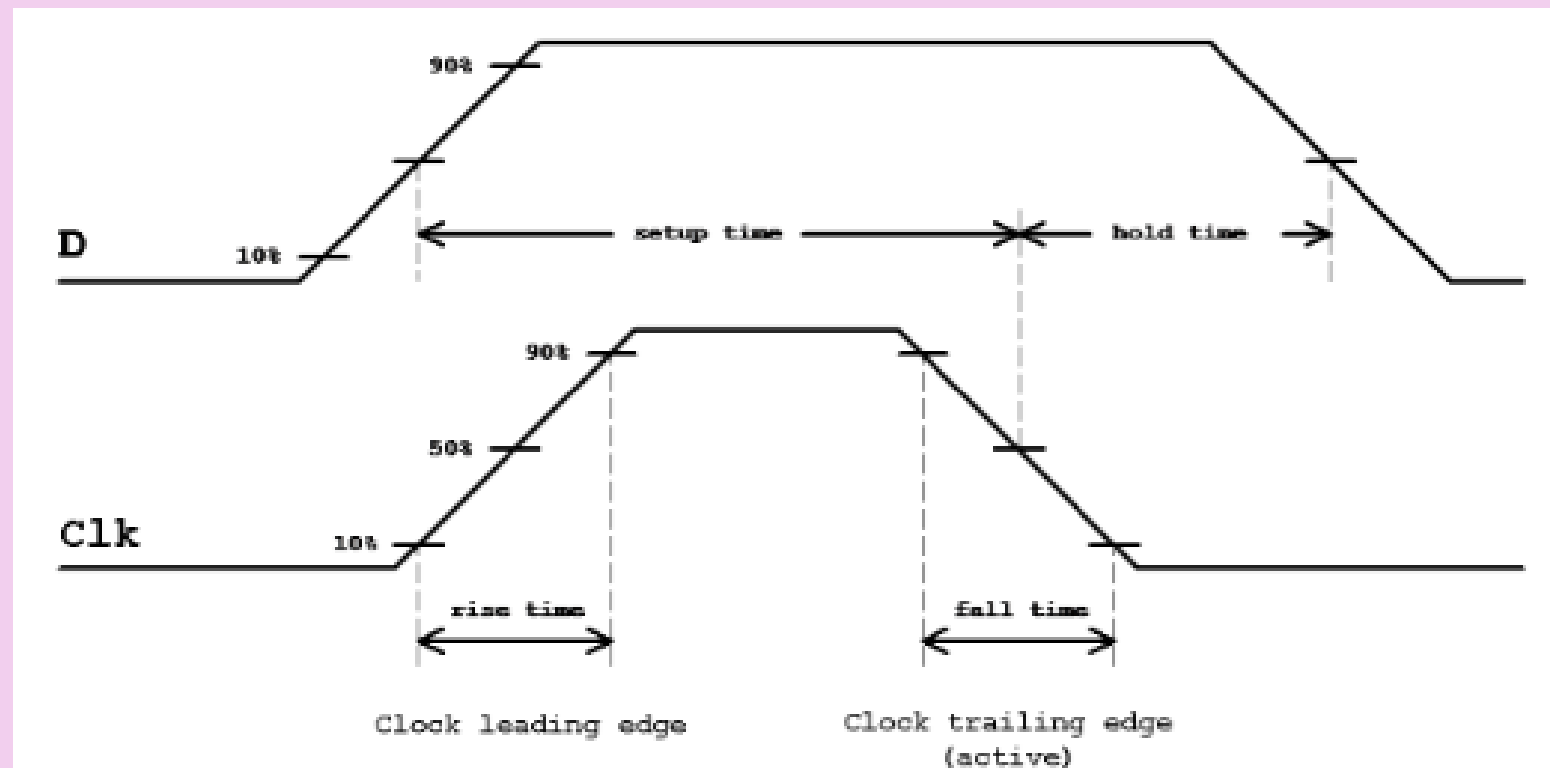
W literaturze angielskiej dokonuje się wyraźnego rozróżnienia między przerzutnikami czułymi na poziom i przerzutnikami wyzwalanymi zboczem, używając terminów zatrzask i przerzutnik. Choć są one bardzo podobne, oba urządzenia wykazują zupełnie inne zachowania. Różnicę wyjaśnia przykład porównawczy:



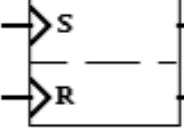
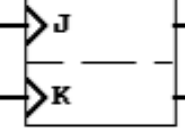
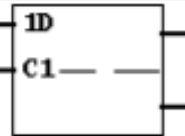
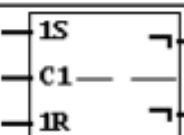
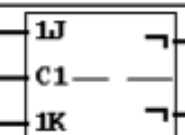
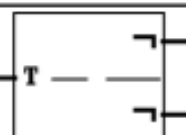
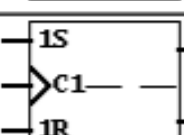
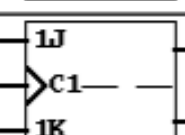
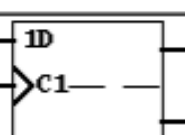
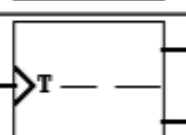
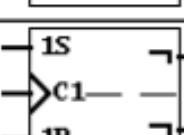
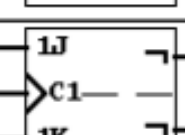
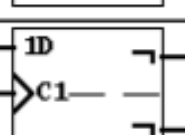
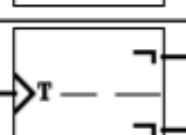
Opóźnienie taktowania (**Clock skew**), jest jednym z problemów w układach przerzutnikowych. Gdy obwody stają się bardziej złożone i mają długie linie zegara, należy wziąć pod uwagę opóźnienie propagacji sygnału. Mogą wystąpić błędy, ponieważ z powodu opóźnień propagacji sygnału nie wszystkie przerzutniki zostaną wyzwolone w tym samym czasie. To przesunięcie czasowe między sygnałami zegara nazywa się clock skew. Przykład:



Aby transfer informacji w przerzutnikach działał niezawodnie, urządzenia przerzutnikowe muszą bardzo precyzyjnie przestrzegać określonych (minimalnych) warunków czasowych. Ograniczenia te odnoszą się w szczególności do definicji czasowych znanych jako „czas ustawienia” i „czas utrzymania”. W połączeniach z niektórymi urządzeniami należy również uwzględnić tzw. „czas narastania” i „czas opadania” impulsu zegara. Żelazną zasadą przy pracy z układami synchronicznymi jest: przesyłany sygnał danych D, **nie może zmieniać się** w czasie trwania aktywnego zbocza/poziomu sygnału zegarowego



Podsumowanie: klasyfikacja przerzutników

		Flip-Flop Type			
Clock	Control	RS	JK	D	T (no data)
unclocked	level-sensitive				
unclocked	edge-sensitive				
clocked	single-state controlled				
clocked	dual-state controlled				
clocked	single-edge triggered				
clocked	dual-edge triggered				

Zaliczenie cząstkowe wykładu nr 4 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

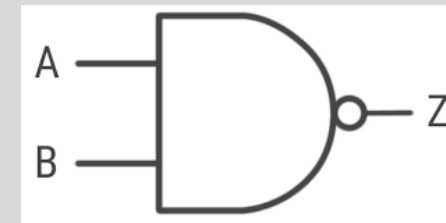
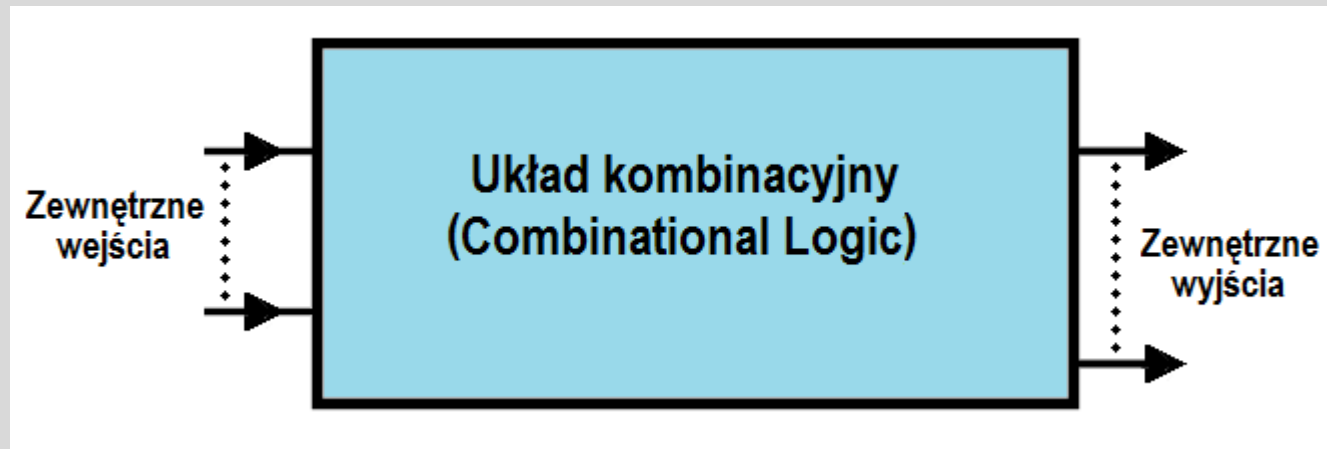
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

Układy cyfrowe dzielą się na dwie podstawowe kategorie: układy kombinacyjne i układy sekwencyjne, przy czym układy kombinacyjne mogą wchodzić w skład układów sekwencyjnych. Układy sekwencyjne dzielą się na kolejne subkategorie: układy sekwencyjne asynchroniczne i układy sekwencyjne synchroniczne. Projektowanie części kombinacyjnej takich układów wyraźnie różni się od projektowania części sekwencyjnej.

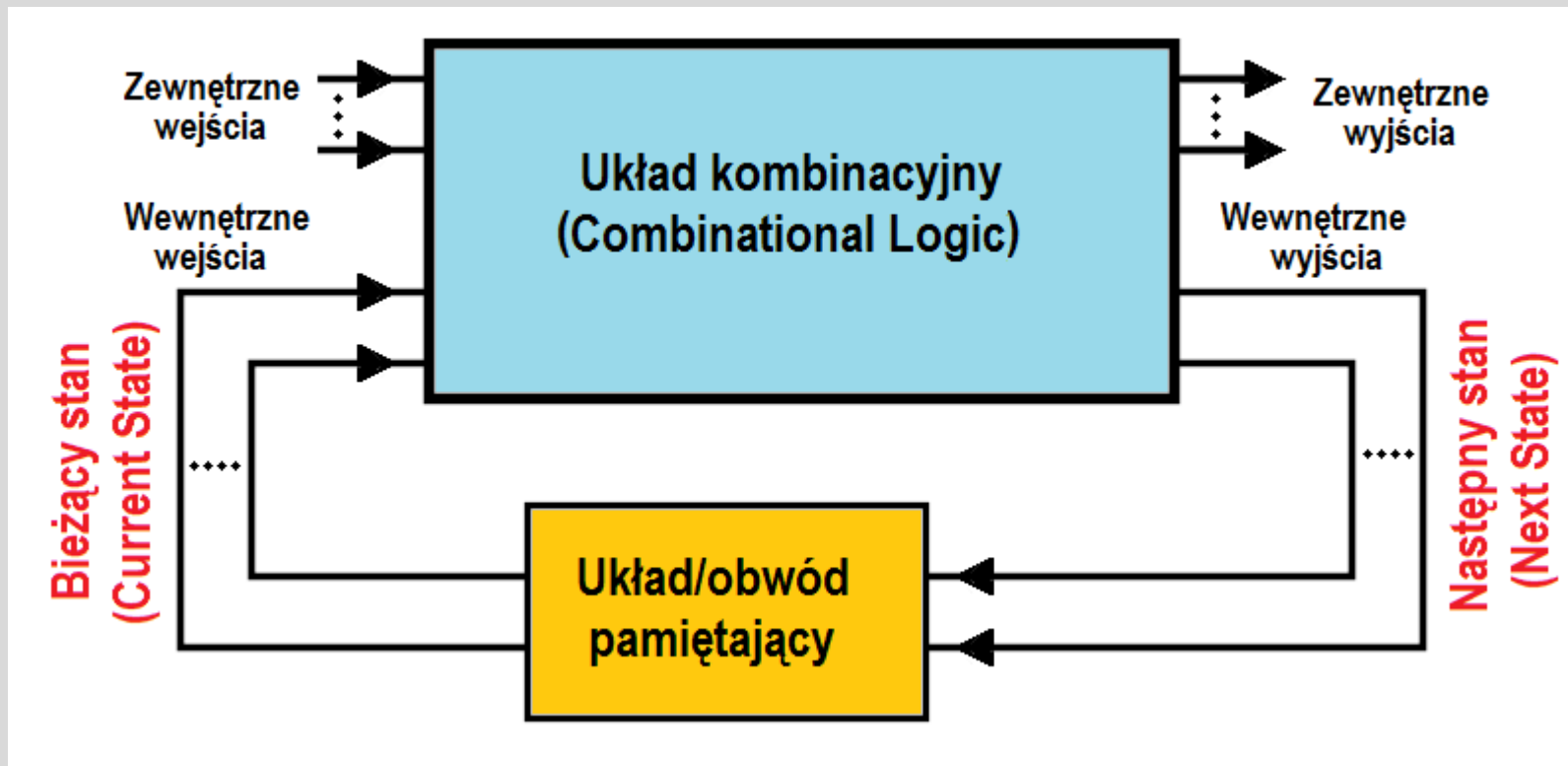


Układ kombinacyjny to taki w którym:

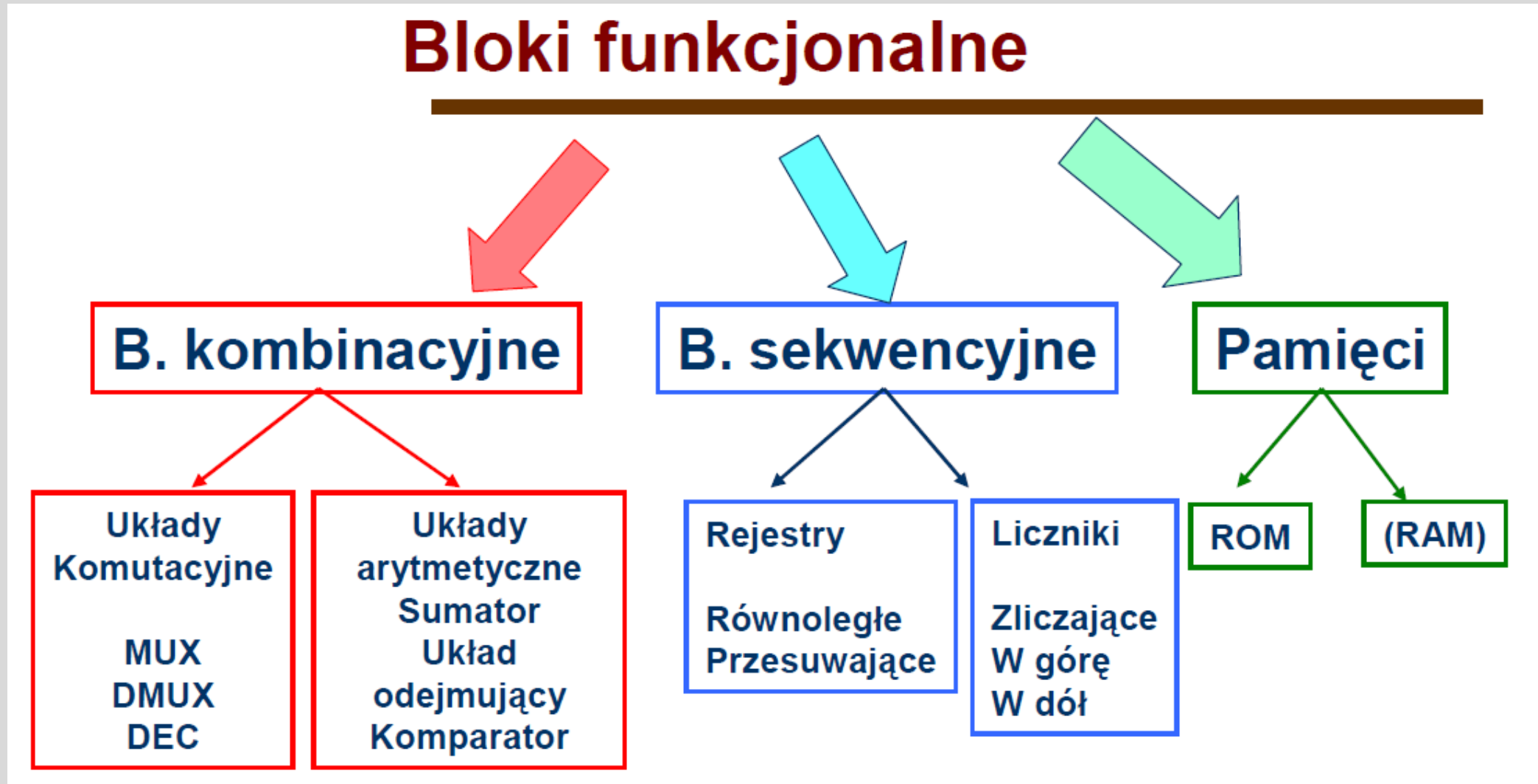
- aktualne wartości sygnałów wyjściowych zależą od aktualnych wartości sygnałów wejściowych,
- jeżeli zmieniają się sygnały wejściowe zmieniają się też sygnały wyjściowe (z opóźnieniem propagacji),
- układ kombinacyjny opisują równania boolowskie oraz tablice prawdy i instrukcje kodu HDL,
- układ może być zbudowany z bramek lub bardziej złożonych układów kombinacyjnych,
- typowym przykładem elementarnego układu kombinacyjnego jest bramka logiczna.

Sekwencyjne układy logiczne to takie, których zewnętrzne sygnały wyjściowe zależą nie tylko od aktualnej wartości zewnętrznych sygnałów wejściowych, ale także od poprzedniej sytuacji w której znajdował się układ – mówimy stanu. Układ sekwencyjny to układ kombinacyjny, który jest objęty pętlą sprzężenia zwrotnego. W pętli sprzężenia zwrotnego znajduje się obwód pamiętający poprzedni stan.

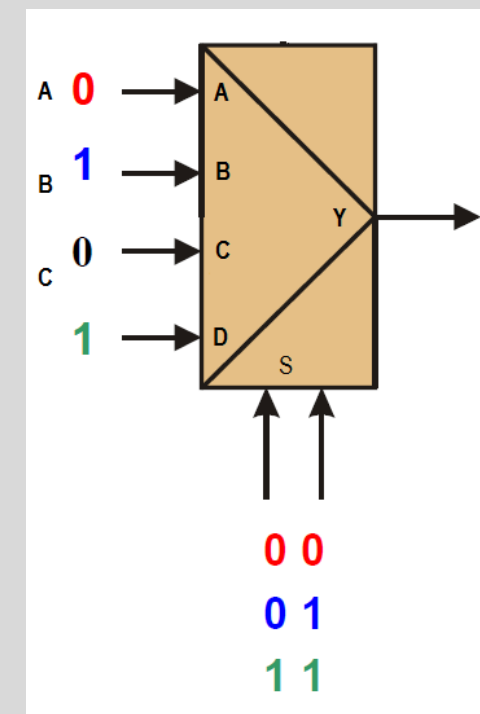
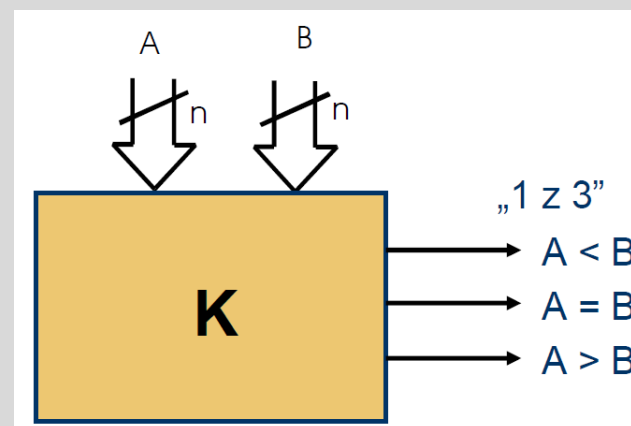
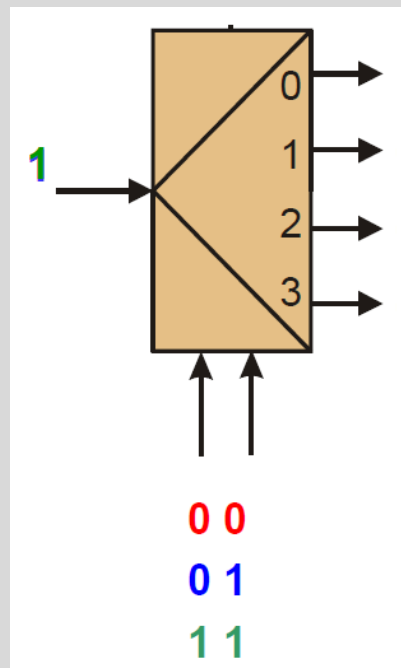
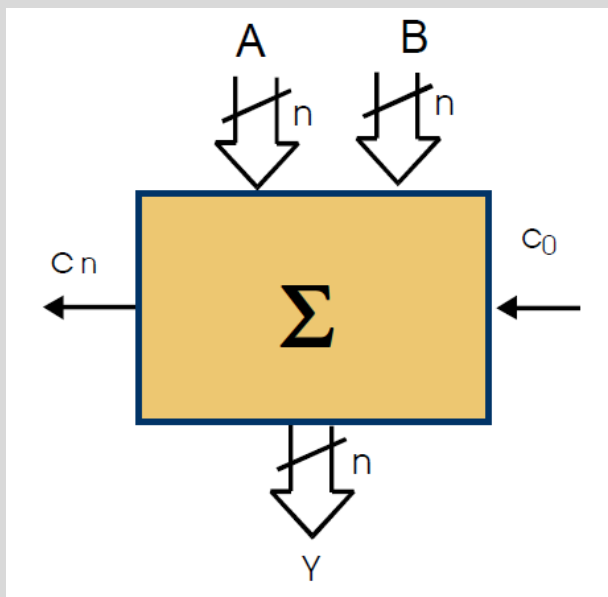
Stan układu to jego zachowanie w konkretnej ustabilizowanej sytuacji, to również zestaw wartości jego sygnałów wejściowych i wyjściowych w warunkach ustalonych.



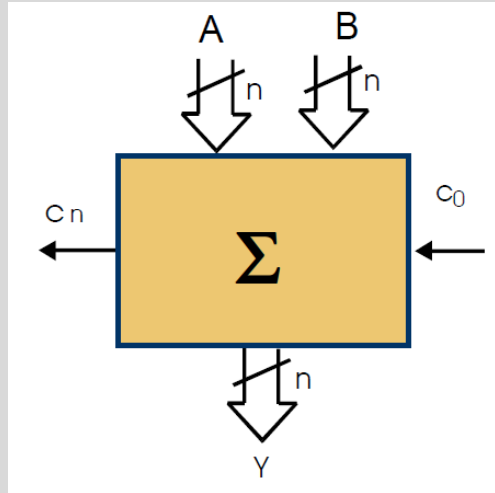
Na rysunku niżej przedstawiono ogólną klasyfikację cyfrowych układów kombinacyjnych:



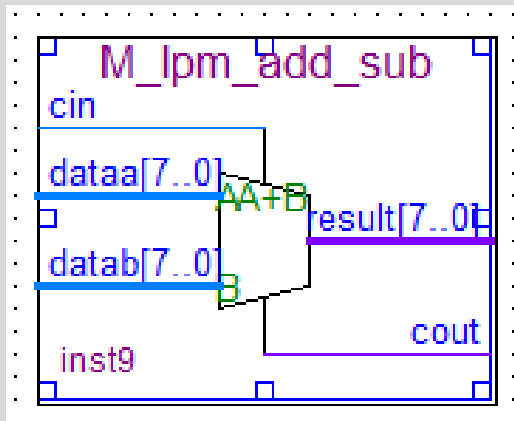
Subkategorię układów kombinacyjnych stanowią bloki kombinacyjne: multiplexery, dekodery, demultiplexery, sumatory, mnożniki i komparatory



Sumatory i układy odejmujące. W aktualnej ofercie handlowej – praktycznie zanikają układy dyskretne. Obecnie funkcję sumatorów (dodawania/odejmowania) realizują mikroprocesory albo układy FPGA.



W przypadku układów programowalnych sumator opisuje się kodem HDL lub makrofunkcją IP.

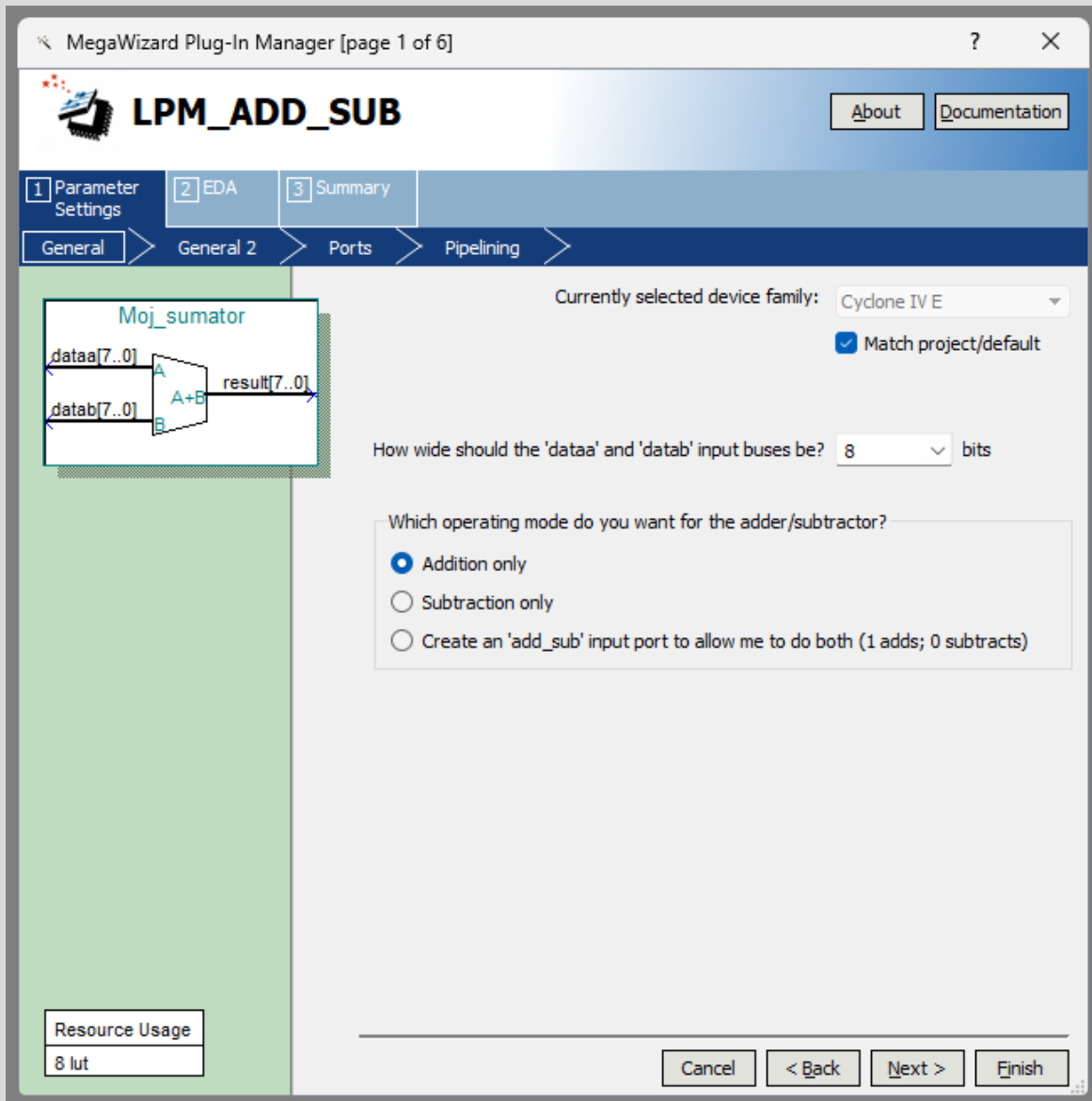


```

1  library ieee;
2  use ieee.numeric_std.all;
3
4  entity vhd11 is
5      Port
6      (
7          A, B      : in  unsigned  (7 downto 0);
8          Suma      : out unsigned  (8 downto 0)
9      );
10 end entity;
11
12 architecture rtl of vhd11 is
13 begin
14     Suma <= A + B;
15 end rtl;

```


Implementacja sumatora IP w systemie Quartus II przebiega następująco: wybieramy IP LPM_ADD_SUB



Na wstępie ustawiamy:

- szerokość pierwszego składnika,
- szerokość drugiego składnika,
- rodzaj operacji,
- możliwość przełączania trybu.

MegaWizard Plug-In Manager [page 2 of 6]

LPM_ADD_SUB [About](#) [Documentation](#)

1 Parameter Settings 2 EDA 3 Summary

General > General 2 > Ports > Pipelining >

Moj_sumator

dataa[7..0] A
datab[7..0] B
A+B
result[7..0]

Is the 'dataa' or 'datab' input bus value a constant?

☒ No, both values vary

☐ Yes, dataa = 0 Dec

☐ Yes, datab = 0 Dec

Which type of addition/subtraction do you want?

☒ Unsigned

☐ Signed

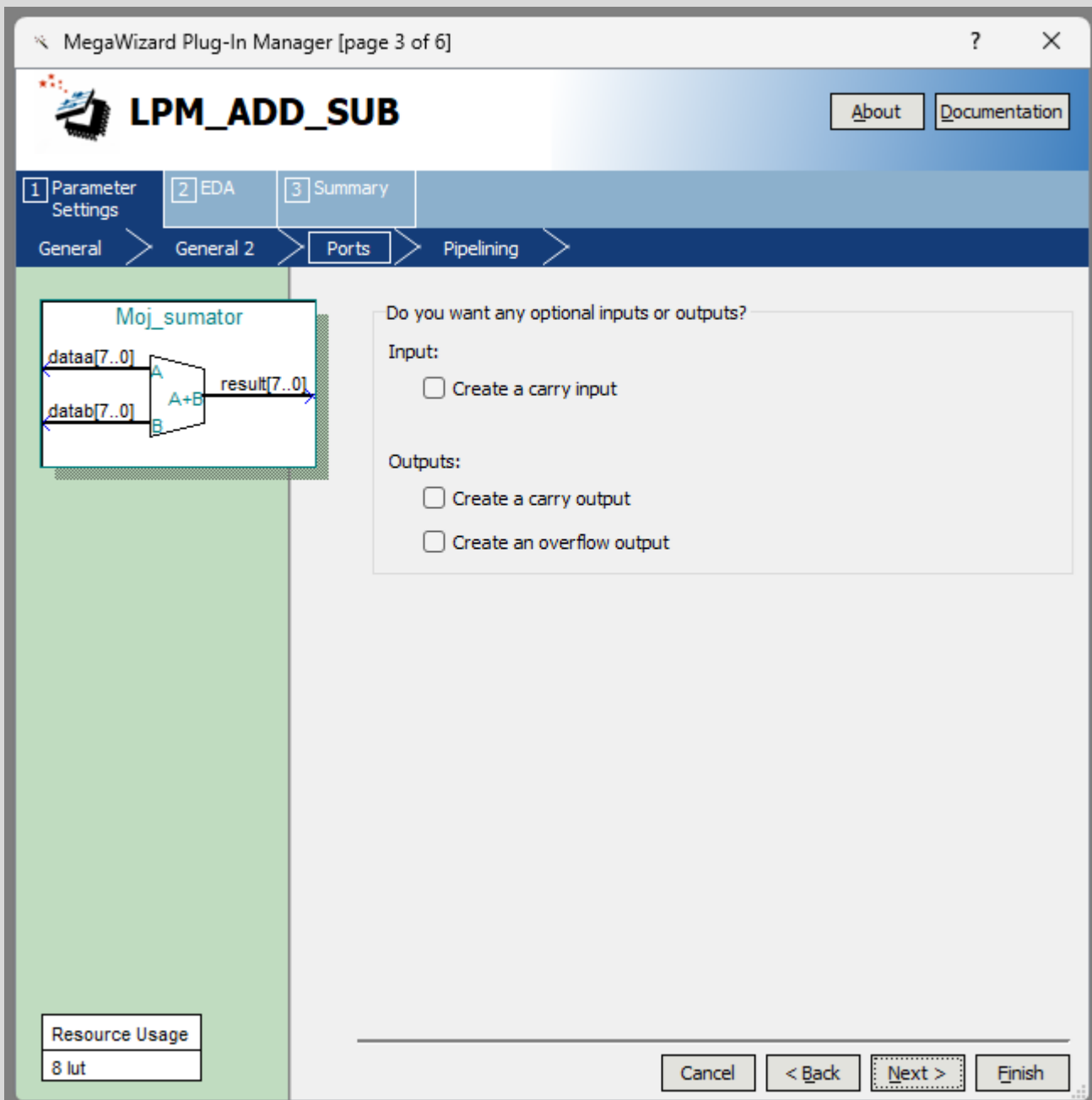
Resource Usage

8 lut

Cancel < Back Next > Finish

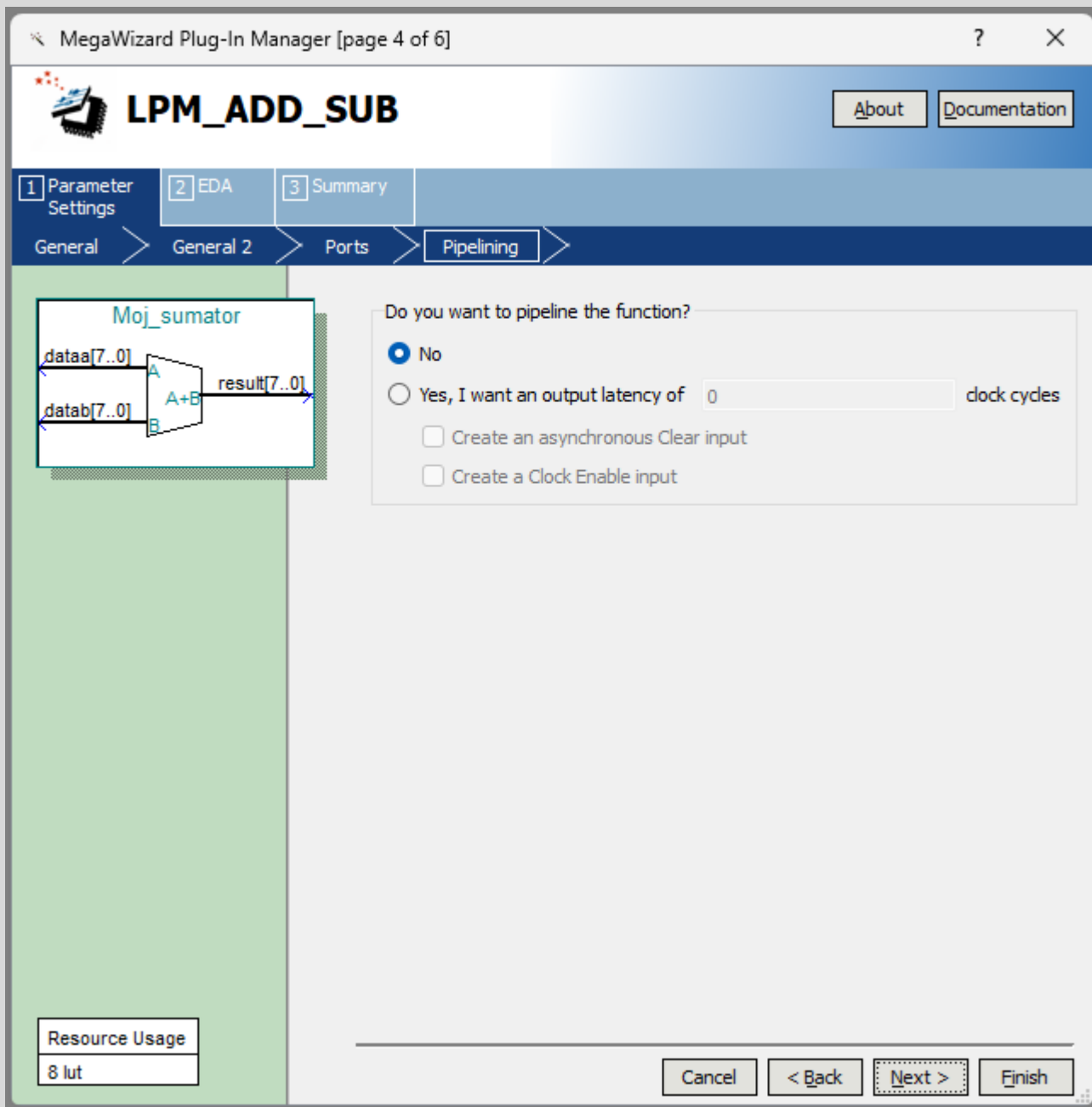
W oknie nr 2 ustawiamy:

- opcję sumowania z wartością stałą,
- system zapisu sumowanych danych,
- rodzaj operacji,
- możliwość przełączania trybu.



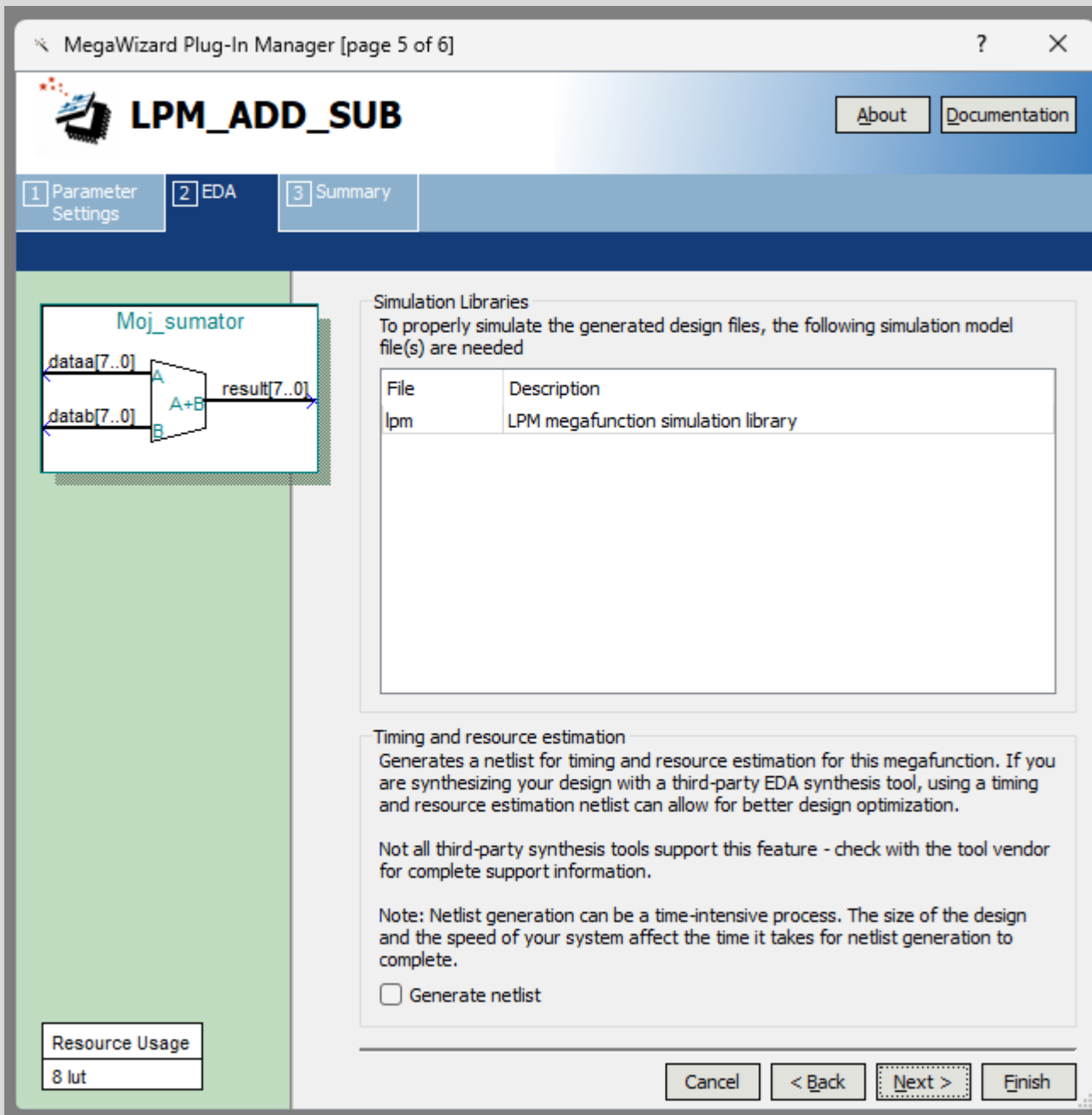
W oknie nr 3 dodajemy opcjonalnie:

- wyjściowy sygnał przeniesienia,
- wejściowy sygnał przeniesienia,
- wyjściowy sygnał przepełnienia.



W oknie nr 4 możemy ustawić tryb przetwarzania potokowego – w tym:

- liczbę taktów opóźniających ,
- ustawienie asynchronicznego zerowania,
- ustawienie sygnału zezwolenia zegara.



W oknie nr 5 akceptujemy podsumowanie:

MegaWizard Plug-In Manager [page 6 of 6]



LPM_ADD_SUB

About

Documentation

1Parameter Settings

2EDA

3Summary

Moj_sumator

dataa[7..0]

datab[7..0]

A

B

A+B

result[7..0]

Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:
E:\Soft\TMP\

File	Description
☒ Moj_sumator.vhd	Variation file
☐ Moj_sumator.inc	AHDL Include file
☒ Moj_sumator.cmp	VHDL component declaration file
☒ Moj_sumator.bsf	Quartus Prime symbol file
☐ Moj_sumator_inst.v...	Instantiation template file

Resource Usage

8 lut

Cancel

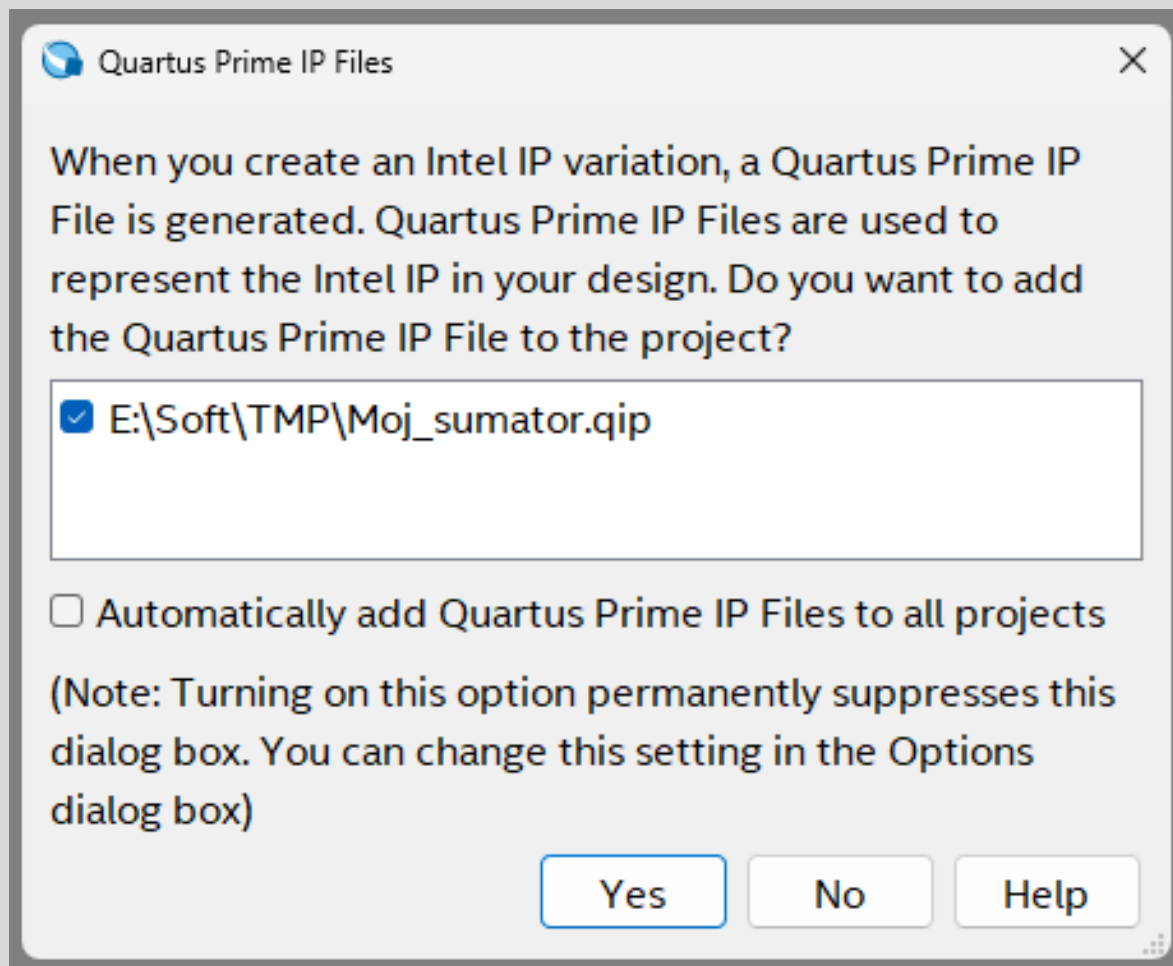
< Back

Next >

Finish

Ważne: w oknie podsumowującym należy ustawić na on klucz: Quartus Prime symbol file

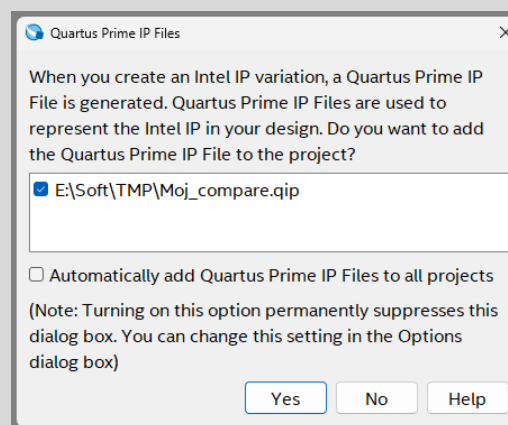
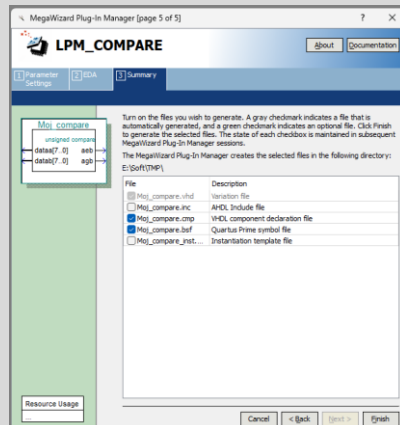
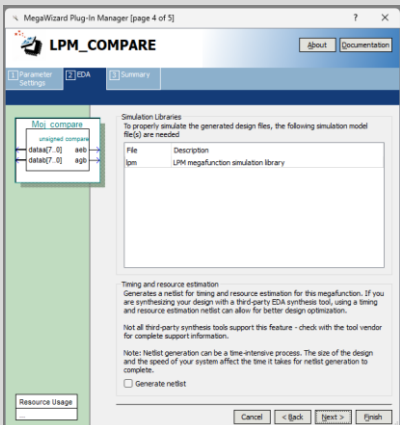
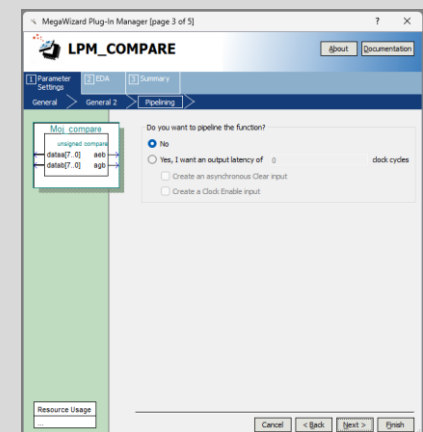
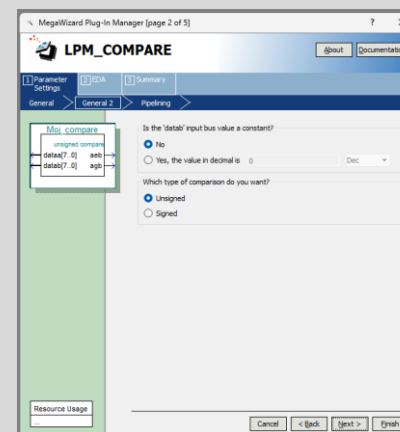
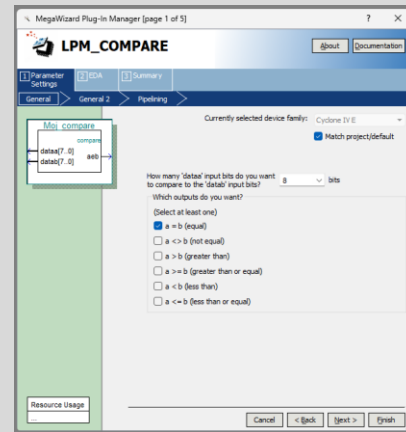
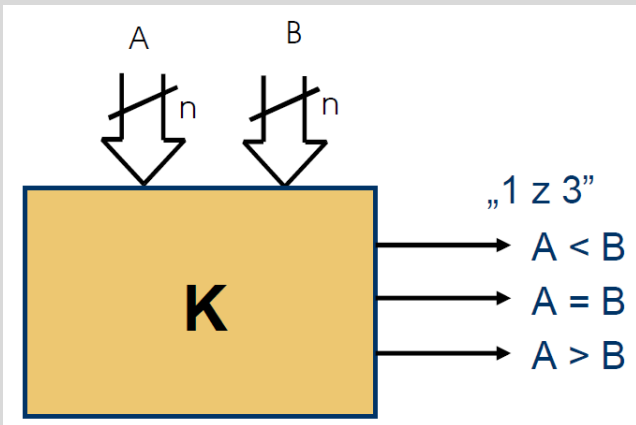
Materiały dydaktyczne przeznaczone dla studentów WE PB



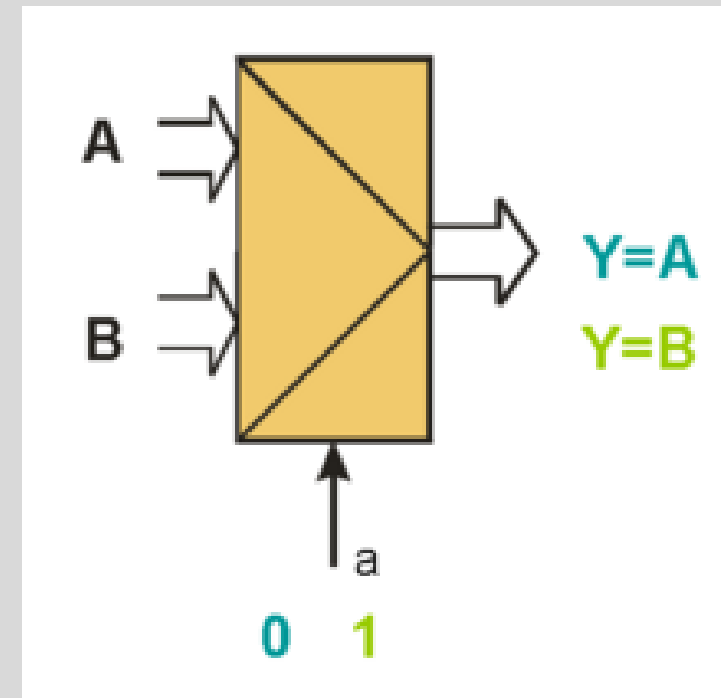
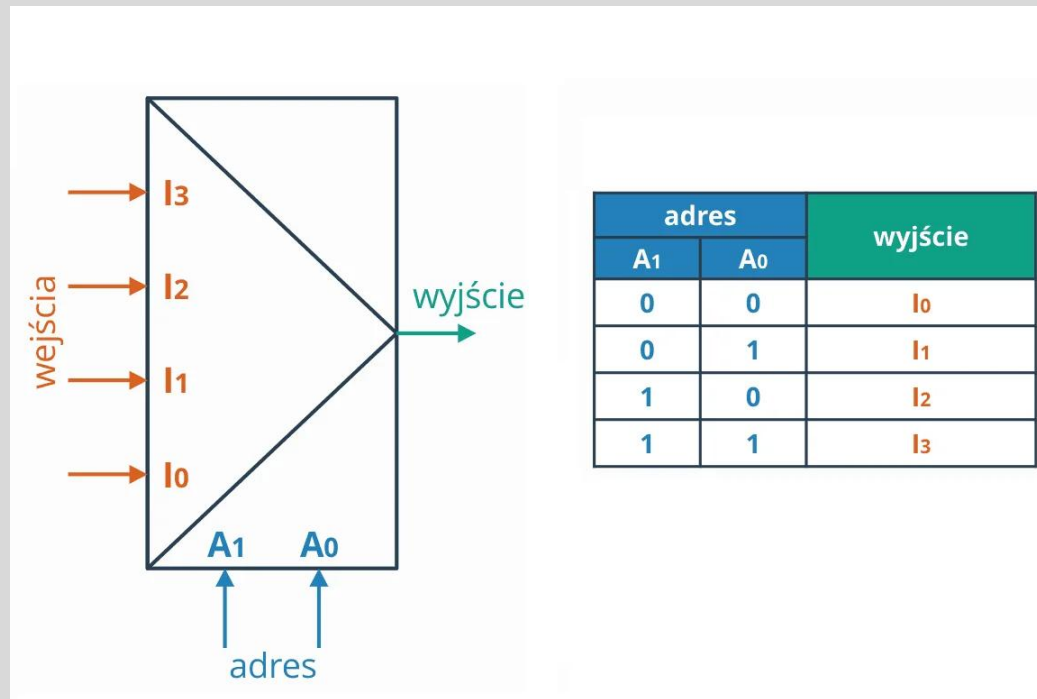
Na końcu akceptujemy (Yes) wygenerowanie makrofunkcji IP nowego sumatora.

Komparator cyfrowy jest układem porównujący dwie wartości wyrażone w sposób cyfrowy:

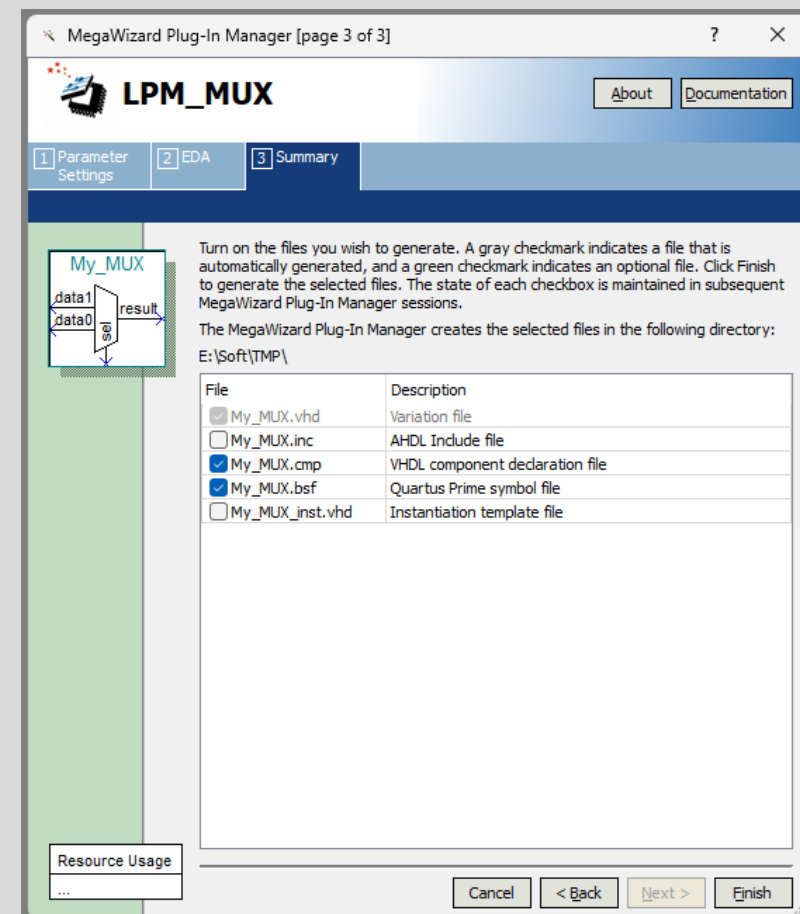
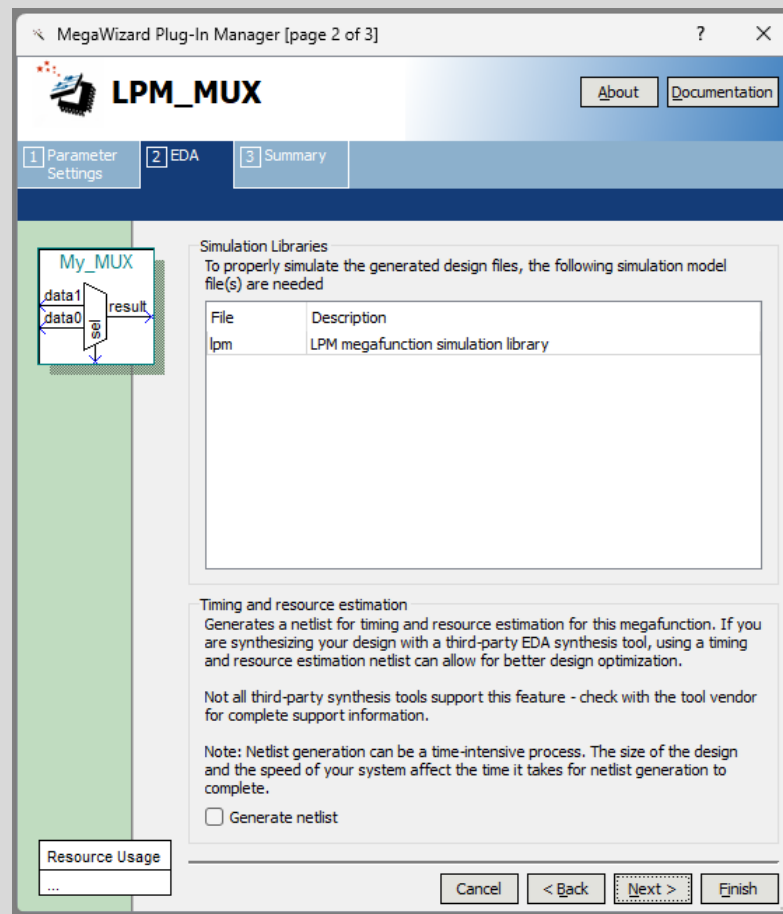
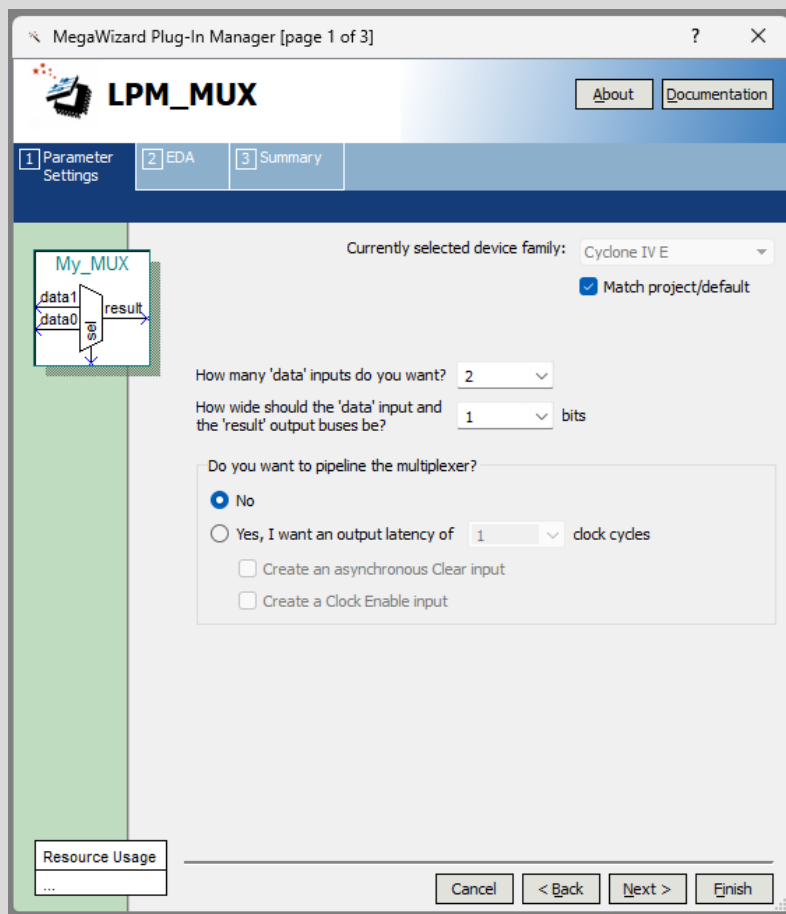
- układ porównuje dwie wejściowe liczby A i B,
- wynikowa wartość '1' ustawiana jest na tym wyjściu na którym spełniona jest relacja sygnałów wejściowych:
 $A = B$, $A < B$, $A > B$, itp.,
- w systemie Quartus II model komparatora realizuje funkcja LPM_COMPARE.



Multiplexer jest to cyfrowy klucz przełączający sygnały z wielu wejść na jedno wyjście w danej chwili. Umożliwia to przesyłanie wielu sygnałów z użyciem jednego kanału transmisyjnego, pracującego w trybie z podziałem czasu. Liczba bitów przełączanego wejścia jest zgodna z liczbą bitów wyjściowych, wejście łączone z wyjściem jest wybierane/wskazywane sygnałem sterującym (adresowym) – zgodnie z zasadą 2^n .



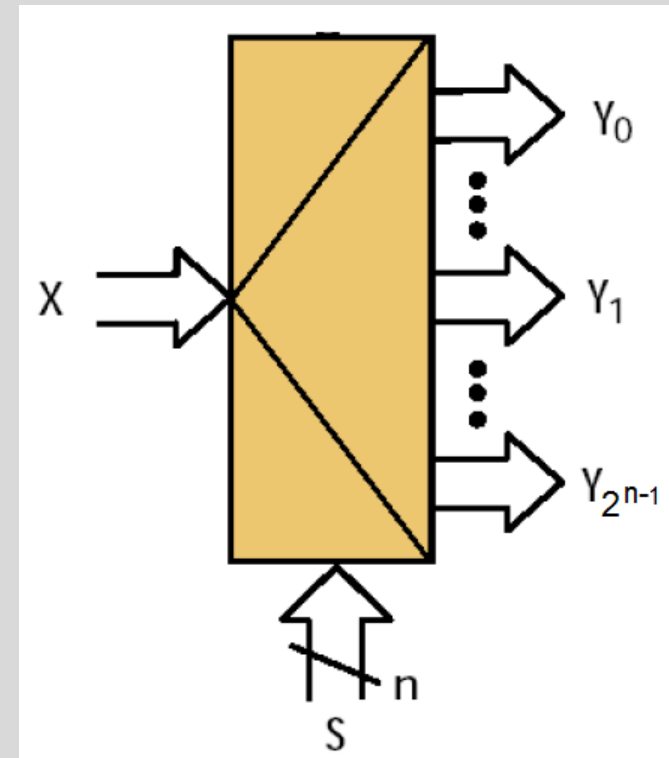
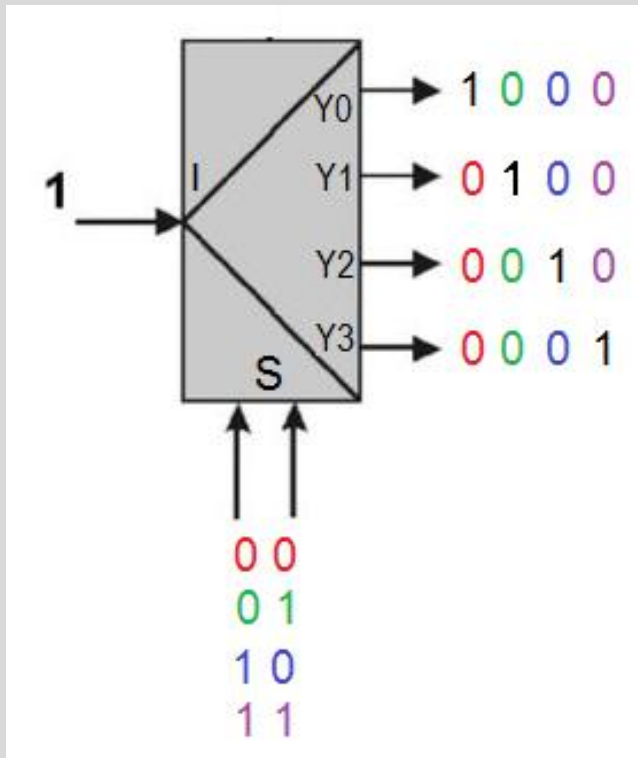
W systemie Quartus II model komparatora realizuje funkcja LPM_MUX:



Demultiplekser - układ o jednym wejściu i wielu wyjściach – realizuje on funkcję odwrotną do multipleksera:

- przełącza/rozdziela wejście jednobitowe lub wielobitowe (magistrale) na kilka wyjść,
- liczba bitów przełączanego wejścia jest zgodna z liczbą bitów sygnałów wyjściowych,
- wyjście łączone z wejściem jest wybierane sygnałem sterującym (adresowym),
- jeżeli układ posiada n-bitowy sygnał sterujący – może przełączać na 2^n wyjść.

W systemie Qurtus II nie istnieje biblioteczny model demultipleksera, który należy syntezować innymi metodami



W ofercie handlowej dostępny jest jednobitowy układ demultipleksa:

Liczba kanałów [2]

Wartość lub zakres...

☒

1 [2]

2

3

4

8

10

16

☒

odznacz wszystko

Liczba wejść [2]

Wprowadź wartość...

☒

1 [2]

2

3

4

5

6

8

☒

odznacz wszystko

Rodzaj układu scalonego [2]

Wprowadź tekst...

☐

dekoder [1]

☒

demultipleksa [2]

2bit

2 do 1 linii

2 do 4 linii

3 do 8 linii

3 do 16 linii

☒

odznacz wszystko

Producent [2]

Wprowadź tekst...

☐

ONSEMI [2]

☒

MICROCHIP (MICREL)

NEXPERIA

NTE Electronics

RENESAS

RENESAS (INTERSI)

☐

zaznacz wszystko

Napięcie zasilania [2]

Wprowadź wartość...

☐

1,65...5,5V DC [2]

0,8...2,7V DC

0,9...3,6V DC

1...3,6V DC

1...5,5V DC

1...6V DC

1,2...3,6V DC

☐

zaznacz wszystko

Obudowa [2]

Wprowadź tekst...

☐

SC70 [2]

DIP14

DIP16

DIP18

DIP24

DSBGA5

DSBGA7

☐

zaznacz wszystko

NC7SZ18

Pin Configurations

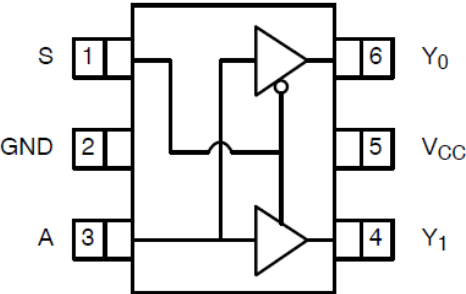


Figure 1. SC70 (Top View)

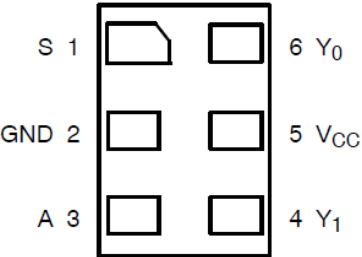
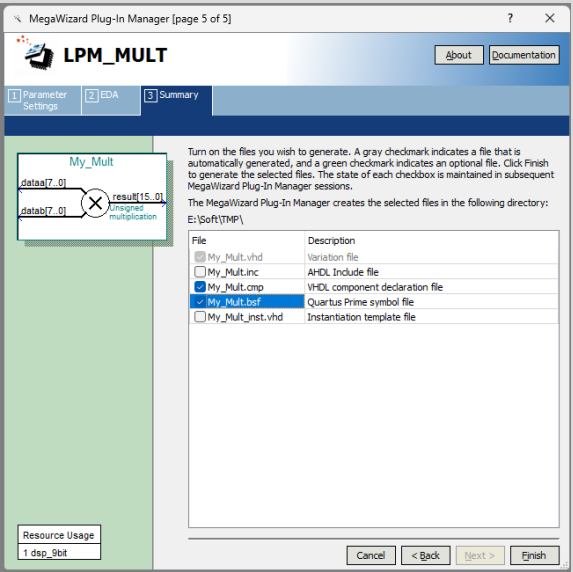
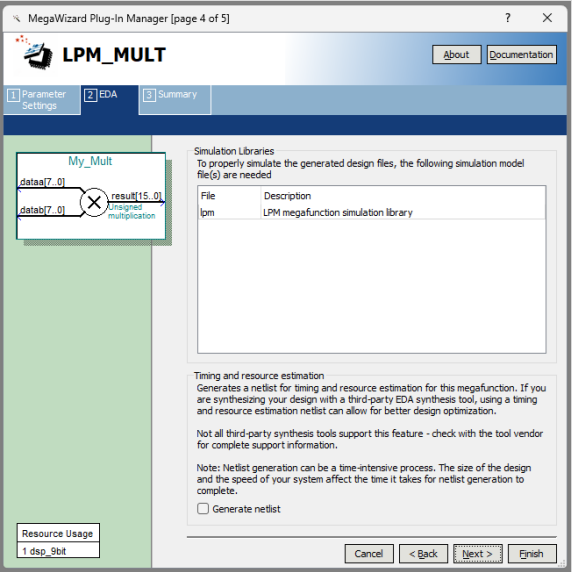
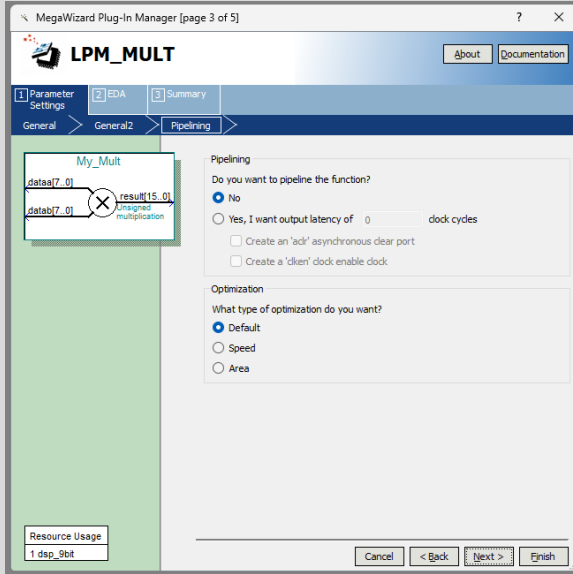
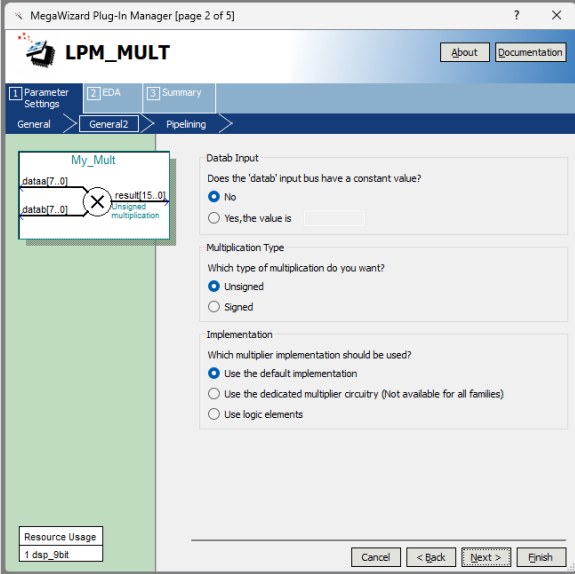
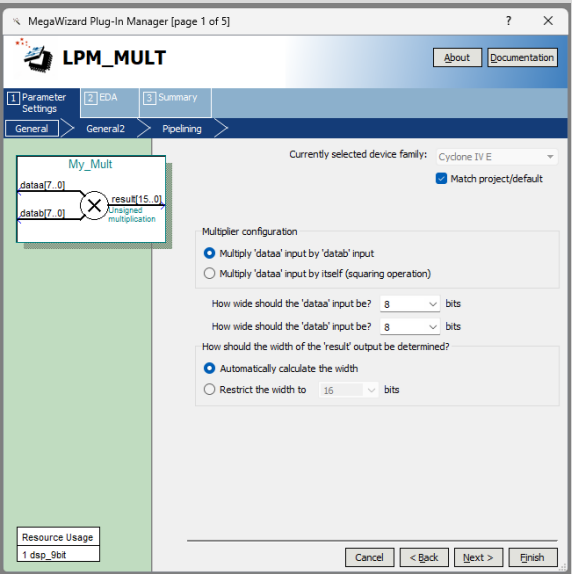


Figure 2. MicroPak (Top Through View)

FUNCTION TABLE

Inputs		Output	
S	A	Y ₀	Y ₁
L	L	L	Z
L	H	H	Z
H	L	Z	L
H	H	Z	H

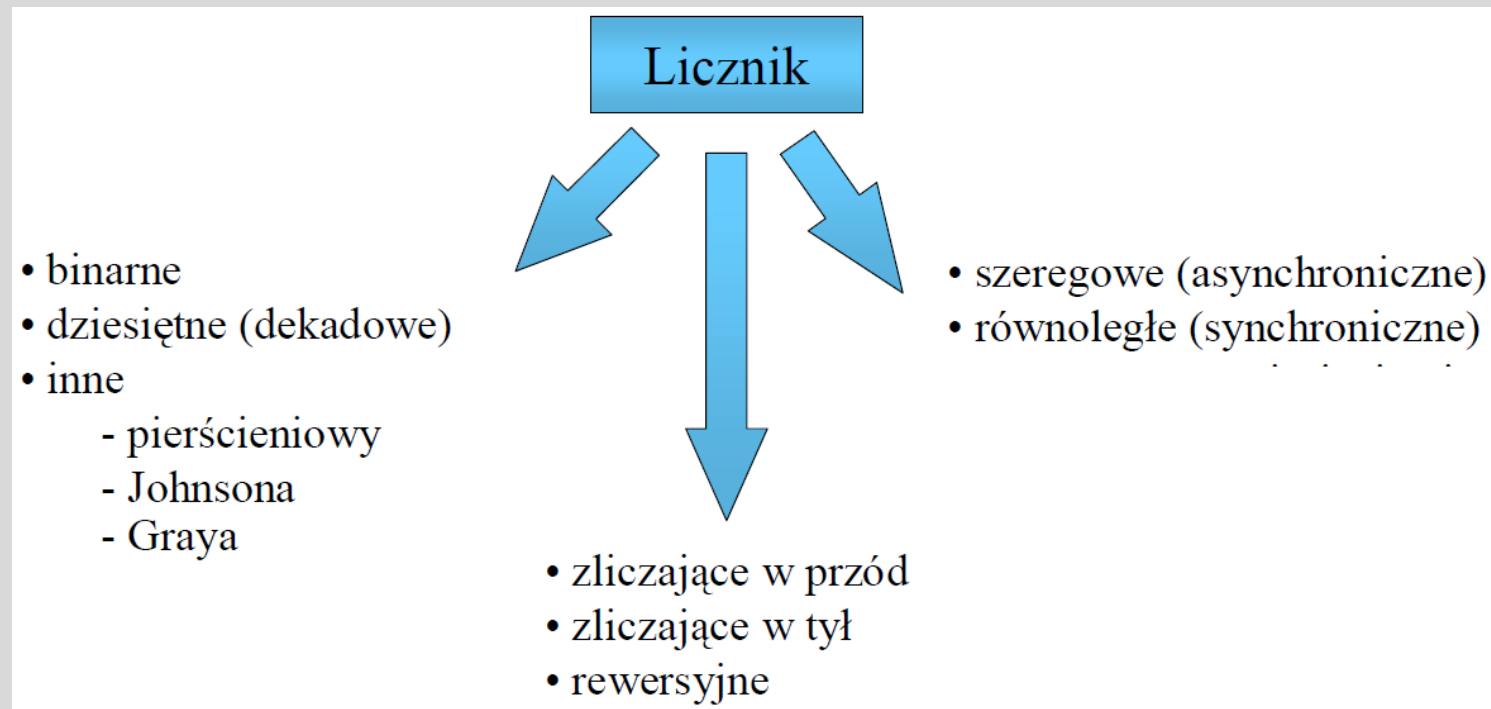
W systemie Quartus II model mnożnika realizuje funkcja LPM_MULT:



Przerzutniki synchroniczne znajdują zastosowania w:

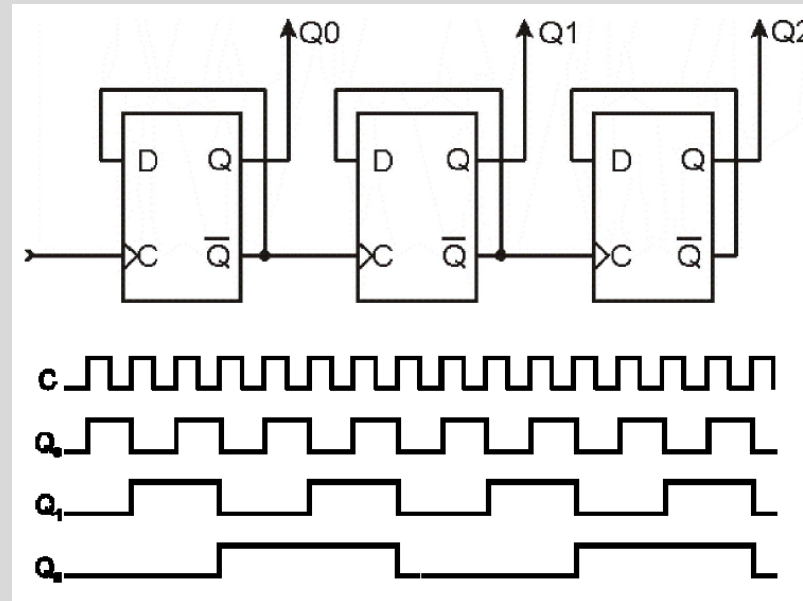
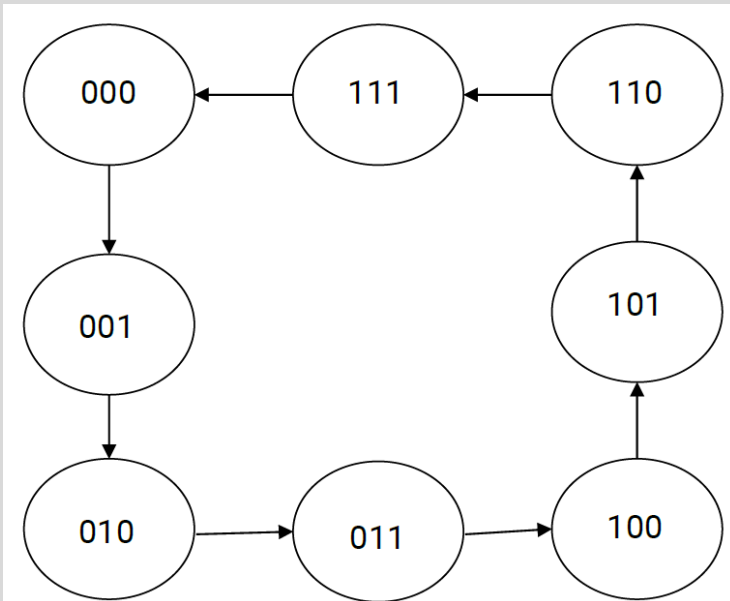
- licznikach.
- dzielnikach częstotliwości,
- rejestrach przesuwnych,
- rejestrach pamiętających,
- układach konwersji danych.

Moduły liczników możemy sklasyfikować jako:



Liczniki w technice cyfrowej służą do liczby impulsów wchodzących na wejście układu w określonym przedziale czasu. Są to układy sekwencyjne synchroniczne, ich podstawowymi parametrami są:

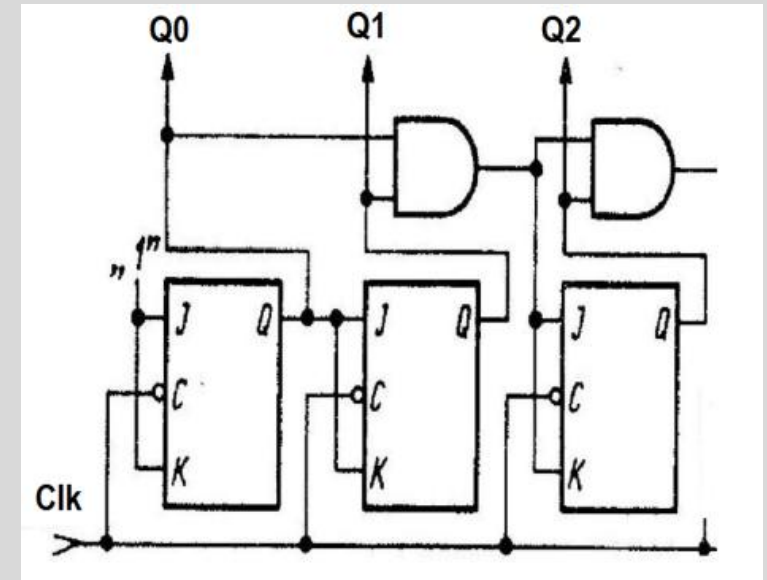
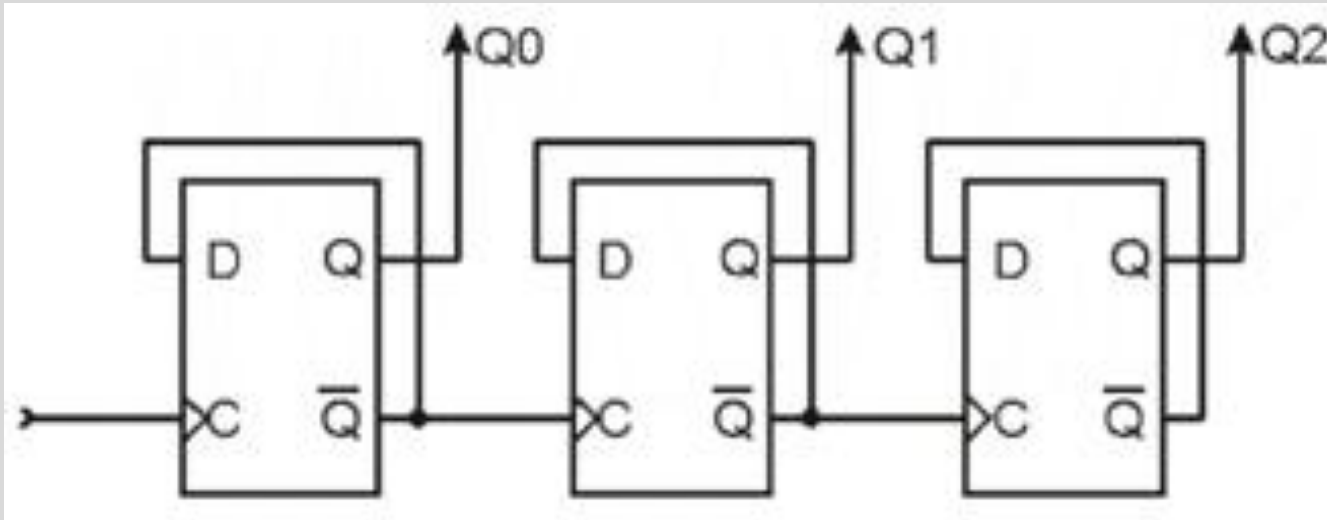
- maksymalna częstotliwość pracy,
- pojemność – liczba bitów słowa wyjściowego – liczba przerzutników,
- tryby pracy.



State	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Liczniki dzielimy na asynchroniczne i synchroniczne:

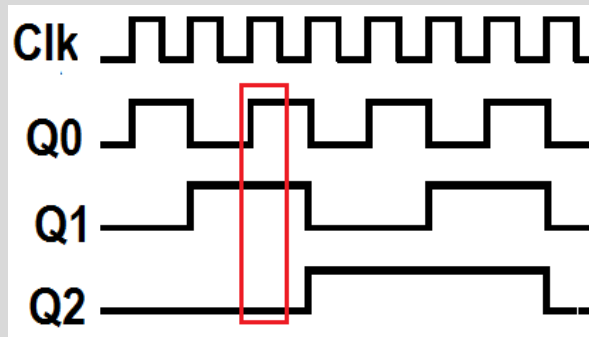
- w asynchronicznych wyjście poprzedniego przerzutnika łączy się z wejściem następnego;
- w synchronicznych wejścia zegarowe wszystkich przerzutników są zwarte;



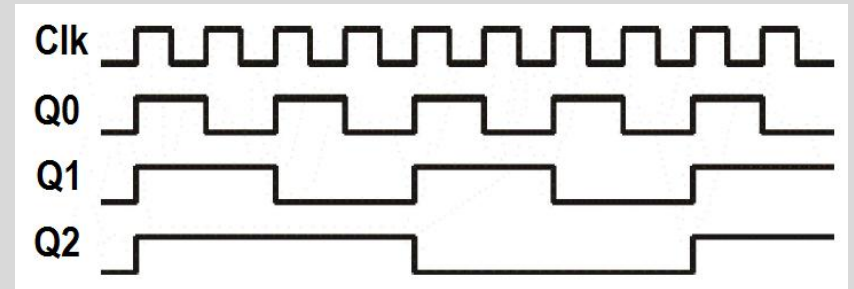
Liczniki klasyfikujemy również jako liczące w przód i wstecz:

- w zliczających w przód w każdym następnym następuje dodawanie kolejnych impulsów;
- w zliczających wstecz następuje odejmowanie impulsów od stanu wyjścia.

S_n	
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

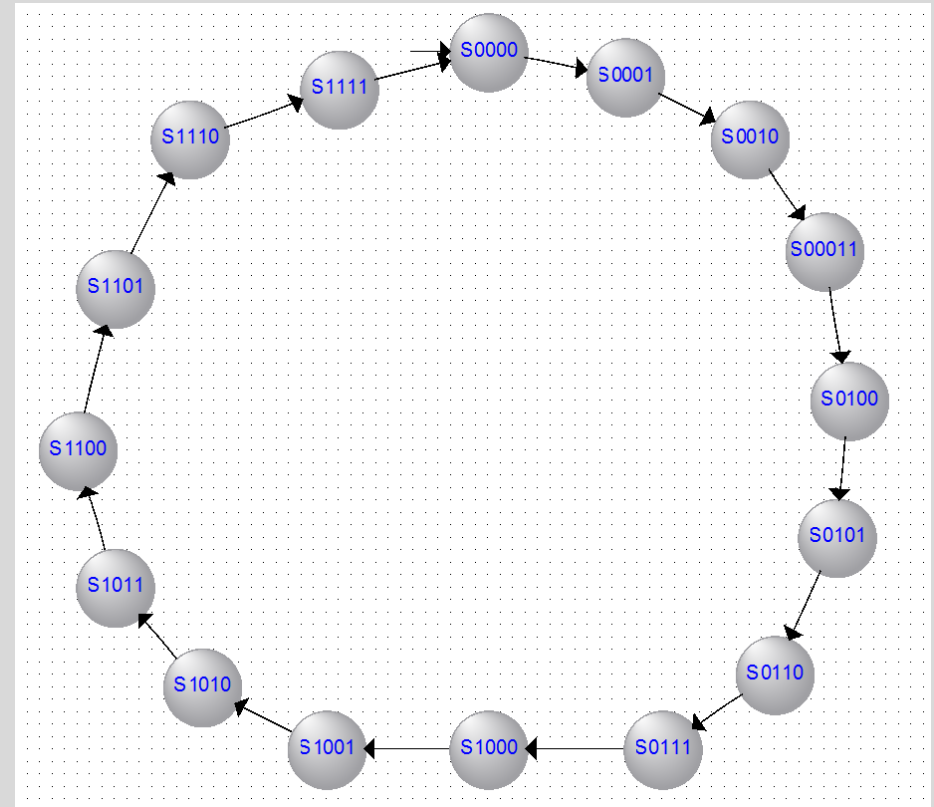
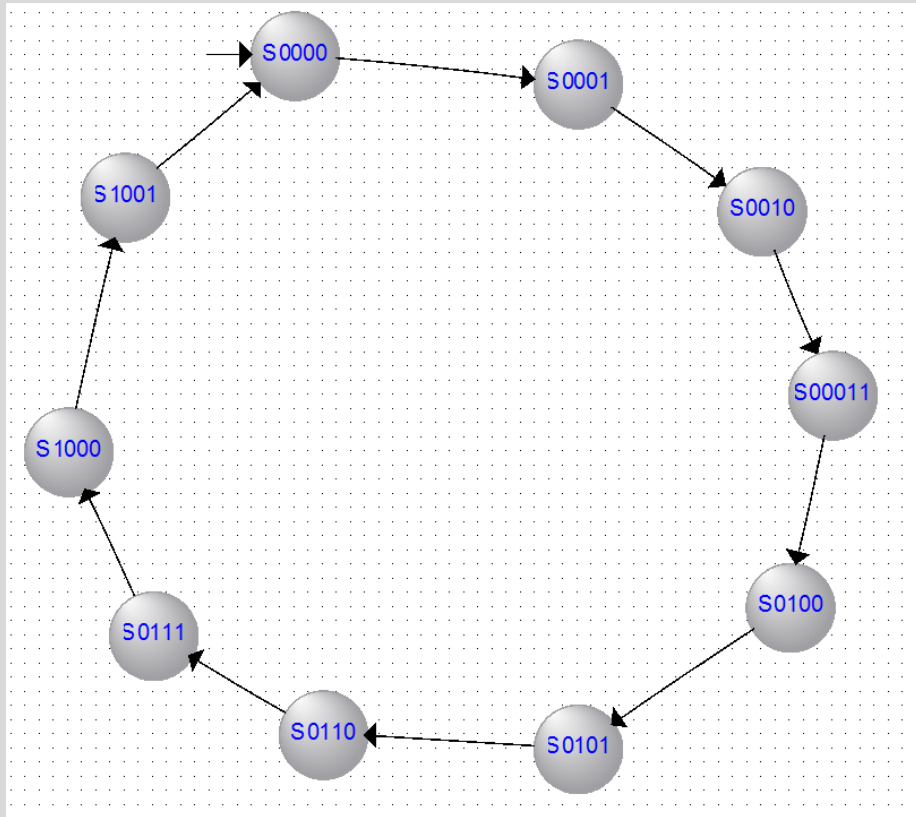


S_n	
0	000
1	111
2	110
3	101
4	100
5	011
6	010
7	001

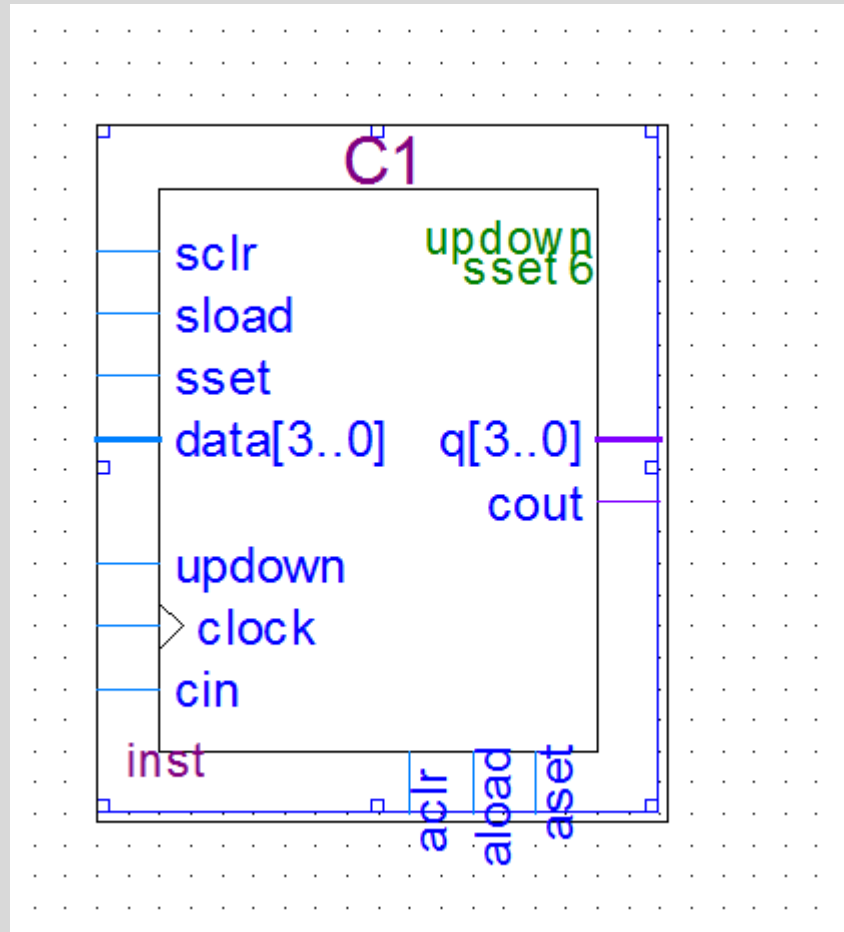


Liczniki klasyfikuje się również jako dziesiętne/dekadowe i binarne:

- dziesiętne/dekadowe przyjmujące tylko 10 stanów na wyjściu w NKB,
- binarne – przyjmujące wszystkie możliwe wartości wynikające z liczby wyjść/bitów.



Niektóre liczniki realizują szereg dodatkowych funkcji: takich jak: wpis równoległy, zerowanie i asynchroniczne i synchroniczne – uniwersalnym rozwiązaniem jest Quartus II **lpm_counter**:

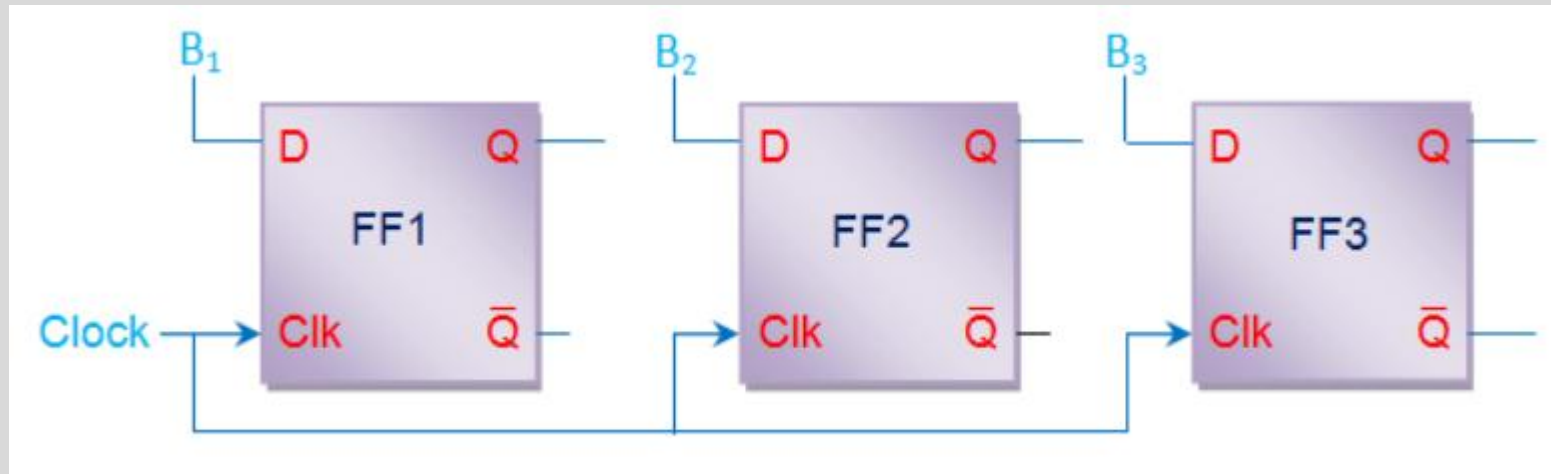


sload – synchroniczny wpis z data[],
 updown – sterowanie kierunkiem zliczania,
 cin – wejście przeniesienia,
 aclr – asynchroniczne zerowanie,
 aload – asynchroniczny wpis z data[],
 aset – asynchroniczne ustawienie "1111",
 sset – synchroniczne wpisanie "0110",
 cout – wyjście przeniesienia.

Rejestry są układami do przechowywania danych, opóźniania sygnałów i konwersji szeregowo-równoległej danych. Parametrami rejestru są:

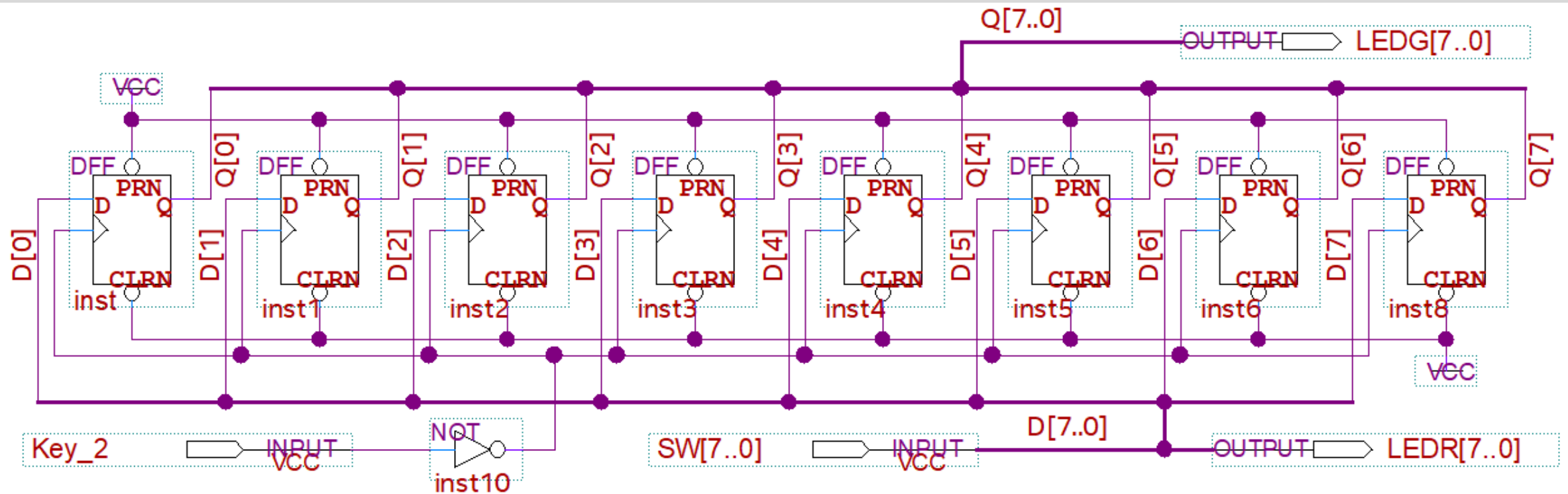
- częstotliwość pracy,
- długość słowa – liczba bitów danych równoległych,
- dodatkowe funkcje.

Rejestr jest zbudowany z zespołu przerzutników połączonych wspólnym wejściem zegarowym oraz często wejściem zerującym. Łącząc wejścia i wyjścia przerzutników otrzymujemy różne konfiguracje rejestrów.



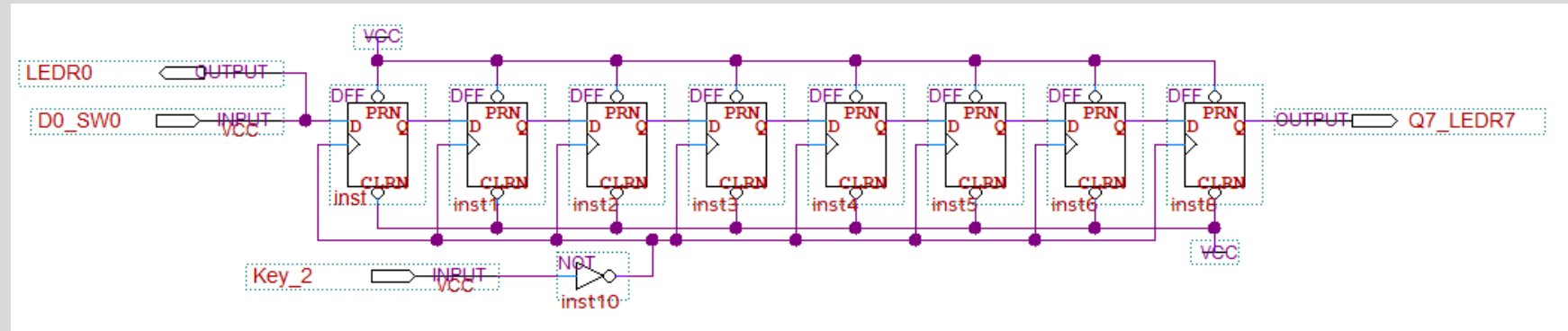
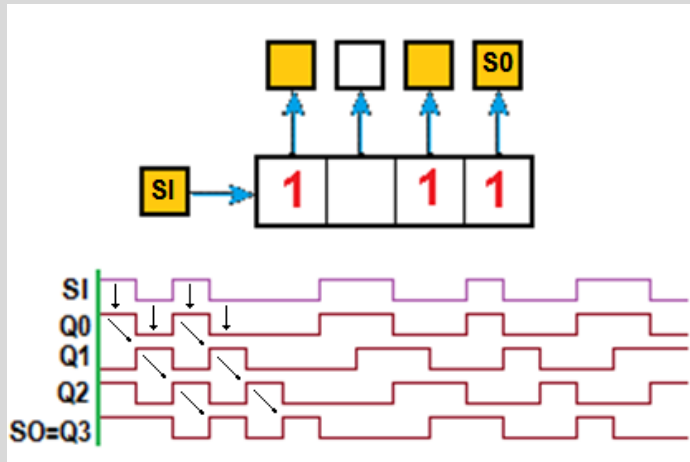
Rejestr równoległy służy do przechowywania danych wielobitowych – równoległych/słów binarnych.

Dane wejściowe są doprowadzane równolegle do wejść wszystkich przerzutników i zapisywane sygnałem zegarowym. Po zapisie danych są one pamiętane do wpisu nowych wartości i są widoczne na wyjściach przerzutników.

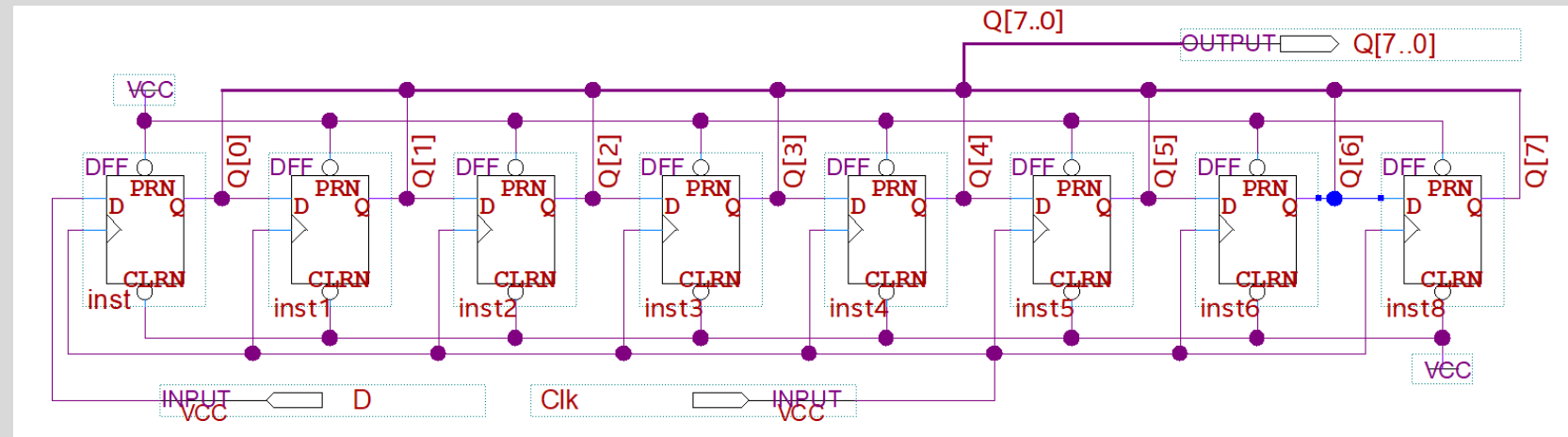
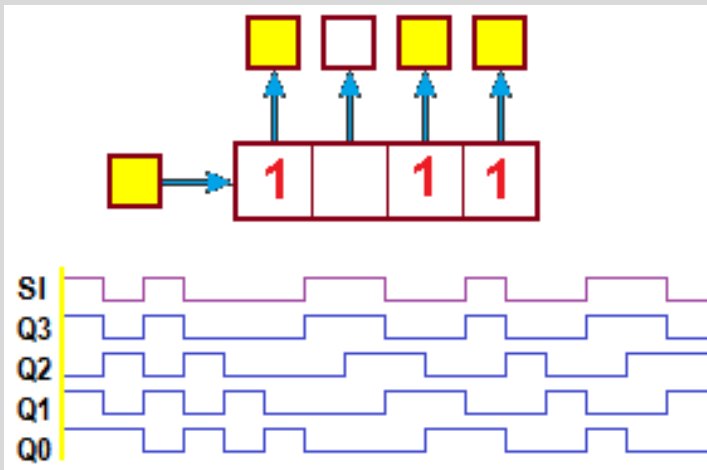


Rejestr szeregowy przesuwania/opóźniania danych szeregowych:

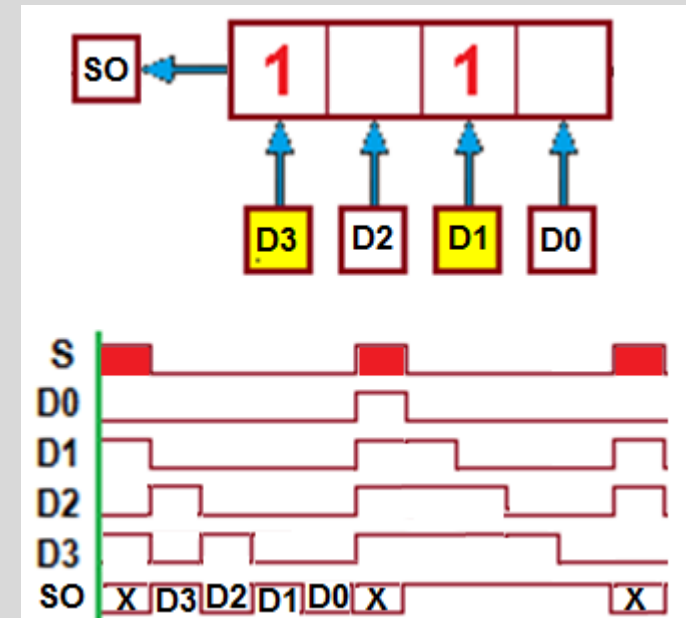
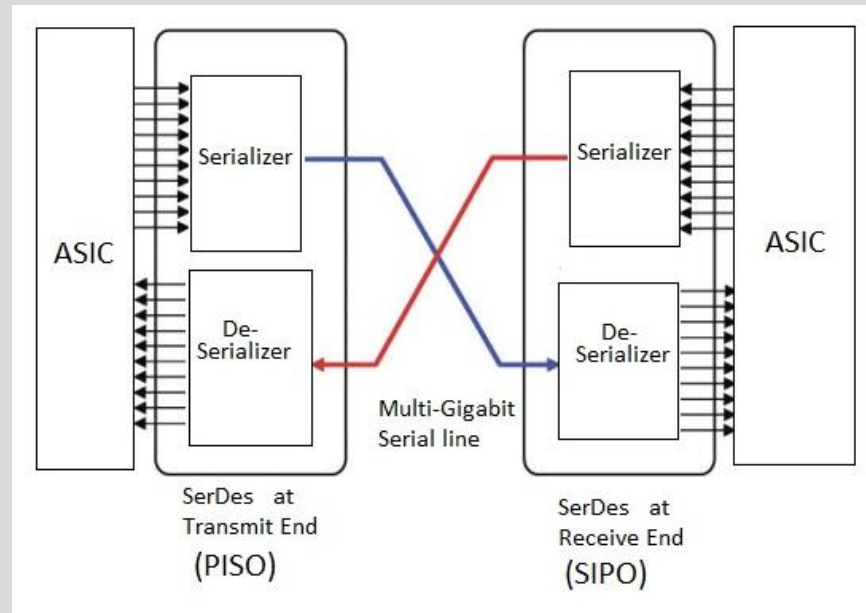
- w 1 takcie zegarowym dane z wejścia szeregowego SI wpisywane są do 1 przerzutnika,
- w 2 takcie dane z 1 przerzutnika są przepisywane do 2 a do 1 wpisywane nowe dane z SI,
- w 3 takcie przepisywane są dane z 2 FF do 3 FF oraz z 1 FF do 2 FF i z SI do 1 FF,
- itd. w kolejnych taktach zegarowych.



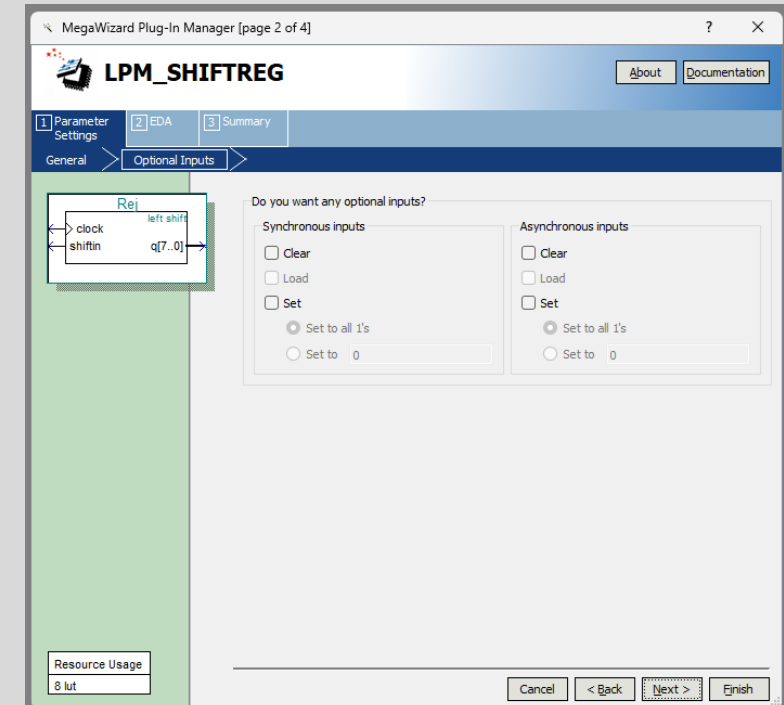
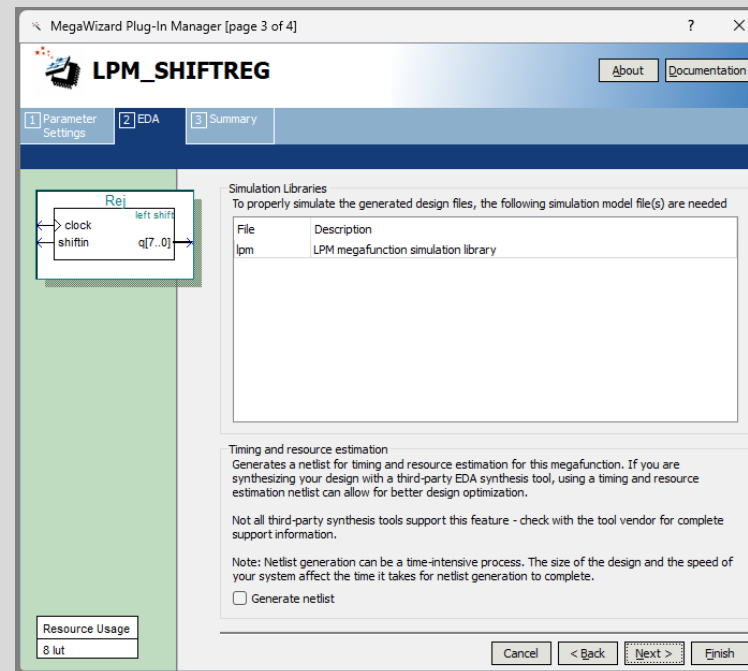
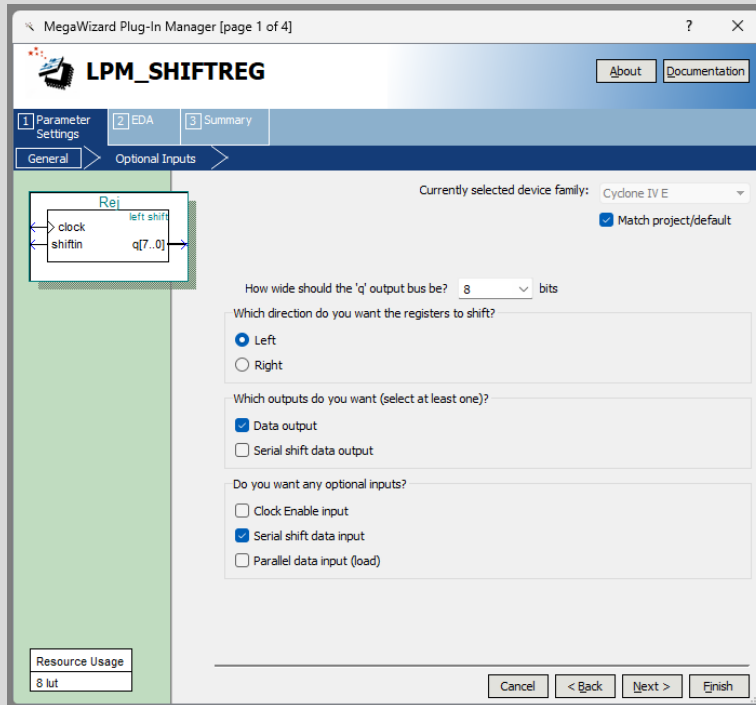
Rejestr szeregowo - równoległy (deserializer) służy do zamiany informacji szeregowej na równoległą. Dane wprowadzane szeregowo na wejście SI są przesuwane i widoczne na wyjściach QD[3..0] w takt sygnału zegarowego. Serializery i deserializery znajdują zastosowanie w systemach szybkiej transmisji szeregowej.



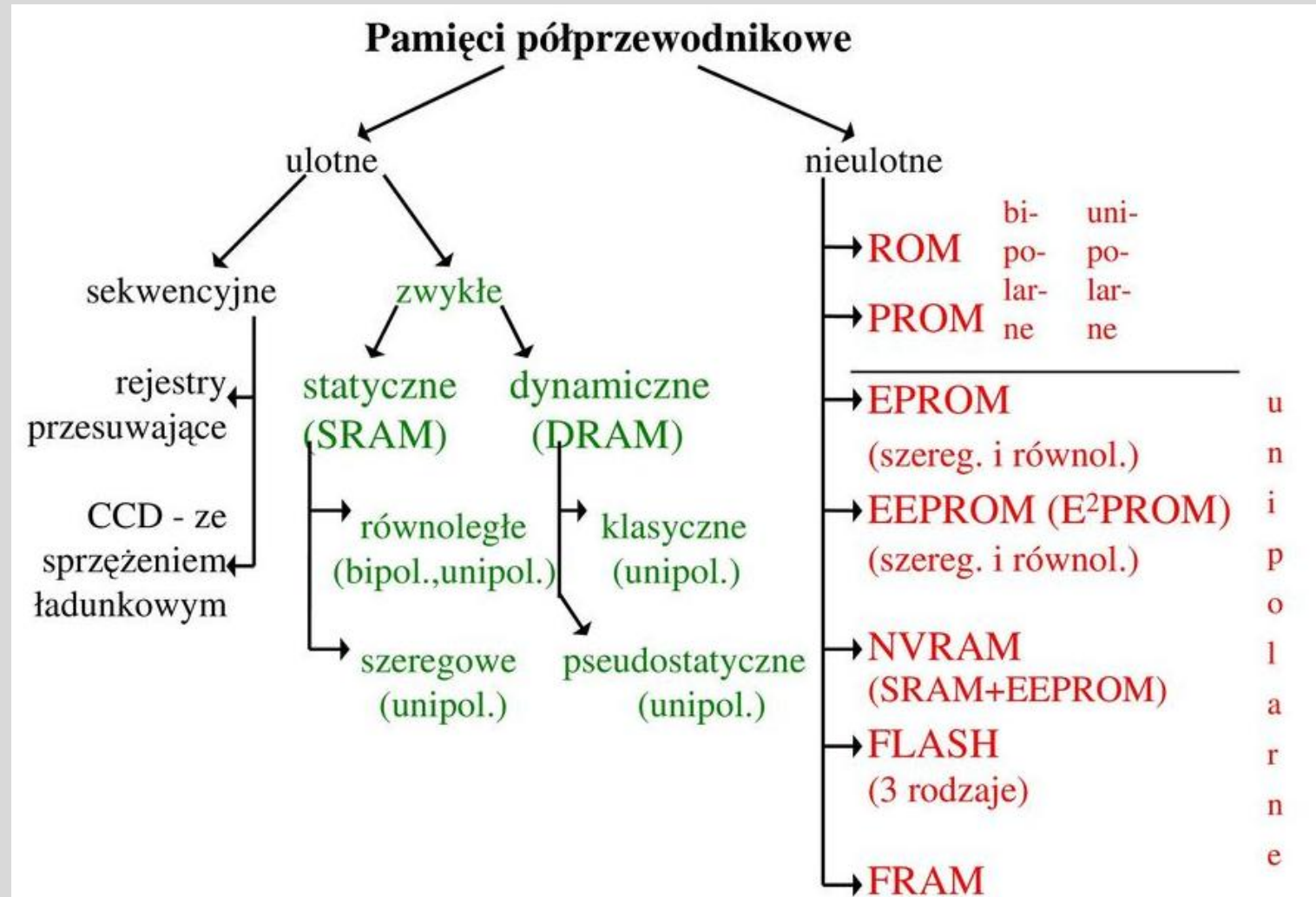
Rejestr równoległy - szeregowy (serializer) służy do zamiany informacji równoległej na szeregową. Konwertowane dane $D[3..0]$ są wprowadzane równoległe poprzez wejścia ABCD a następnie w kolejnych taktach zegarowych przesuwane w prawo i wyprowadzane przez wyjście szeregowe SO. Do wyboru trybu: wpis równoległy lub przesuwanie – służy wejście sterujące S.



Rejestr uniwersalny łączy w sobie funkcje wszystkich wymienionych rejestrów. W systemie Quartus II model tego rejestru realizuje funkcja LPM_SHIFTREG:



Pamięć półprzewodnikowa – rodzaj cyfrowego układu scalonego, przechowującego informacje binarne. Pojęcie pamięci ulotnej, nieulotnej i pamięci o dostępie swobodnym.



Najważniejsze parametry pamięci RAM – z **dostępem równoległym**, szyna adresowa i szyna danych jest wielobitowa, cechą jest duża liczba wyprowadzeń z układu scalonego – zaletą krótki czas dostępu.

Specyfikacja

Atrybuty produktu	Wartość atrybutu	Wyszukaj podobne
Producent:	GSI Technology	☐
Kategoria produktów:	SRAM	☒
RoHS:	 Szczegóły	
Wielkość pamięci:	288 Mbit	☐
Organizacja:	8 M x 36	☐
Czas dostępu:	4 ns	☐
Maksymalna częstotliwość zegara:	400 MHz	☐
Rodzaj interfejsu:	Parallel	☐
Napięcie zasilania – max.:	3.6 V	☐
Napięcie zasilania – min.:	2.3 V	☐
Prąd zasilania – max.:	560 mA, 840 mA	☐



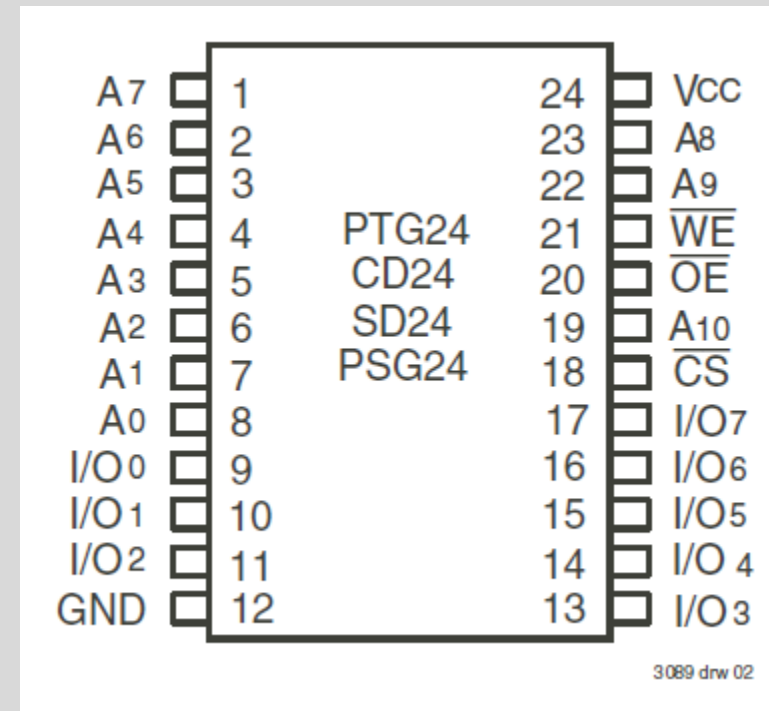
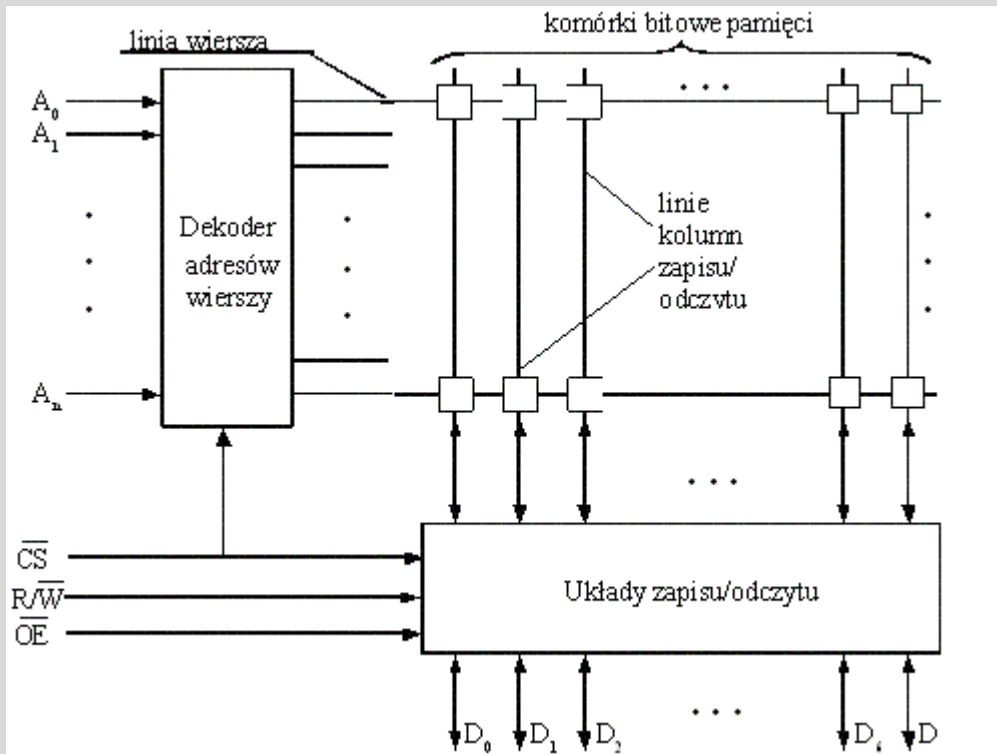
W pamięci z interfejsem szeregowym, zarówno adres jak i dane podawane są szeregowo bit po bicie w takt sygnału zegarowego. Stosowana specjalna magistrala sprzętowa SPI/I2C do przesyłu danych i adresu. Pamięć wolniejsza w stosunku do pamięci z interfejsem równoległym, ale prostsza w zastosowaniu.

Specyfikacja

Atrybuty produktu	Wartość atrybutu	Wyszukaj podobne
Producent:	Microchip	☐
Kategoria produktów:	SRAM	☒
RoHS:	 Szczegóły	
Wielkość pamięci:	1 Mbit	☐
Organizacja:	128 k x 8	☐
Czas dostępu:	25 ns	☐
Maksymalna częstotliwość zegara:	20 MHz	☐
Rodzaj interfejsu:	SDI, SPI	☐
Napięcie zasilania – max.:	5.5 V	☐
Napięcie zasilania – min.:	2.5 V	☐
Prąd zasilania – max.:	10 mA	☐
Minimalna temperatura robocza:	- 40 C	☐
Maksymalna temperatura robocza:	+ 85 C	☐
Styl mocowania:	Through Hole	☐
Opakowanie/obudowa:	PDIP-8	☐



Sygnały zewnętrzne pamięci równoległej SRAM asynchronicznej:



A_0 - A_{10} – magistrala adresowa – 11 bitów czyli można zaadresować $2^{11} = 2048$ słów,

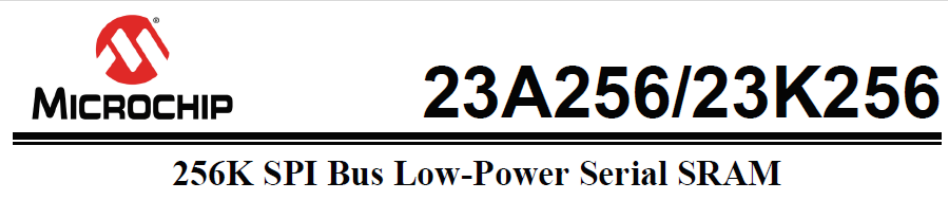
I/O_0 - I_7/Q_7 – magistrala danych 8 – bitowa, wspólna magistrala dla danych wejściowych i wyjściowych,

$\overline{CS}/\overline{CE}$ – Chip Select lub Chip Enable, sygnał włączający układ do pracy (aktywne zero – stąd negacja),

$\overline{OE}$ – Output Enable, sygnał włączający na magistralę dane wyjściowe (aktywne zero);

$\overline{R/W}$ – sygnał sterujący zapisem/odczytem (Read/Write) – $\overline{R/W} = 1$ odczyt, $\overline{R/W} = 0$ zapis (to samo $\overline{WE}$).

Sygnały zewnętrzne pamięci szeregowej SRAM – przykład z magistralą SPI:



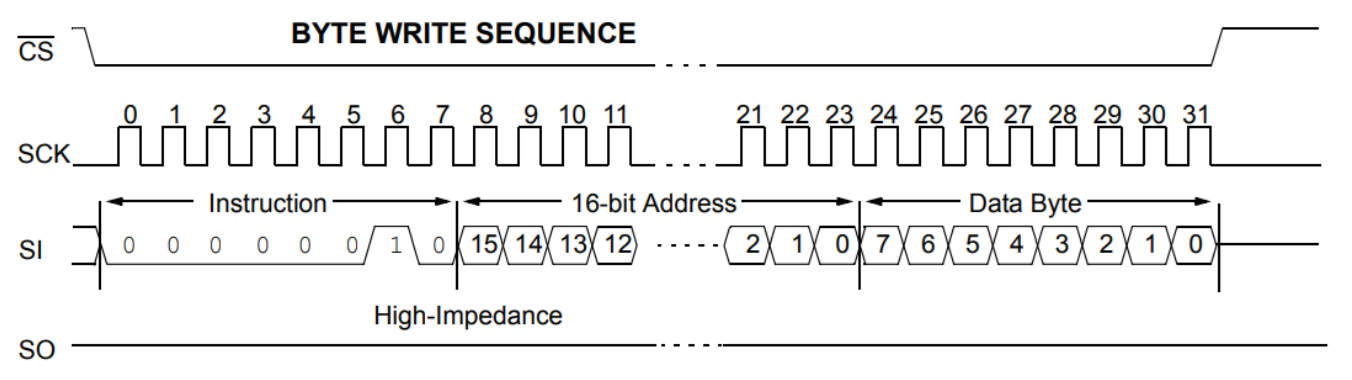
Pin Function Table

Name	Function
$\overline{\text{CS}}$	Chip Select Input
SO	Serial Data Output
Vss	Ground
SI	Serial Data Input
SCK	Serial Clock Input
$\overline{\text{HOLD}}$	Hold Input
Vcc	Supply Voltage

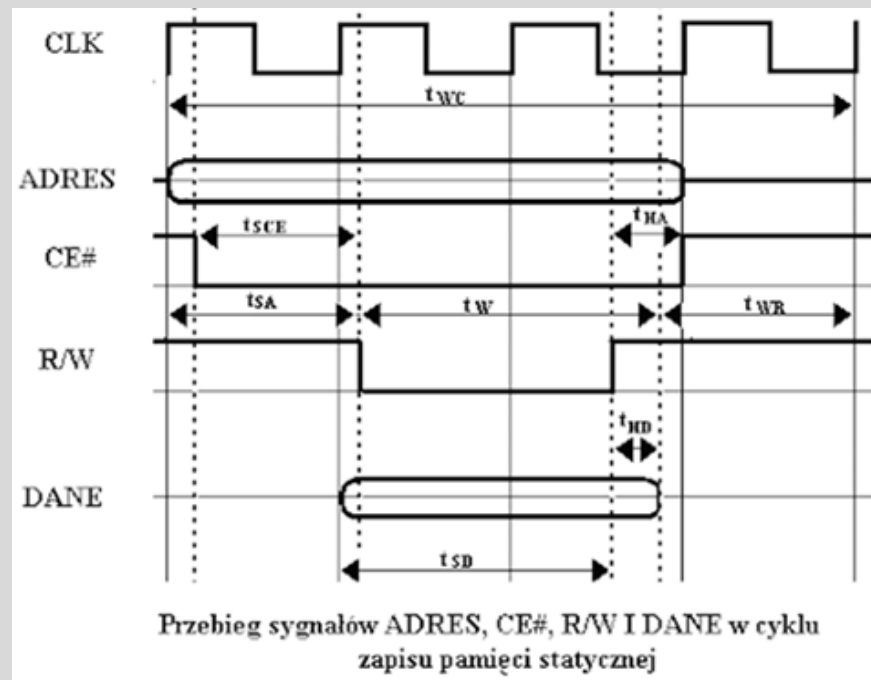
Package Types (not to scale)

PDIP/SOIC/TSSOP
(P, SN, ST)

The diagram shows a rectangular package with pins on both sides. The left side has pins 1 through 4, and the right side has pins 8 through 5. The functions are: 1: $\overline{\text{CS}}$, 2: SO, 3: NC, 4: Vss, 8: Vcc, 7: $\overline{\text{HOLD}}$, 6: SCK, 5: SI.

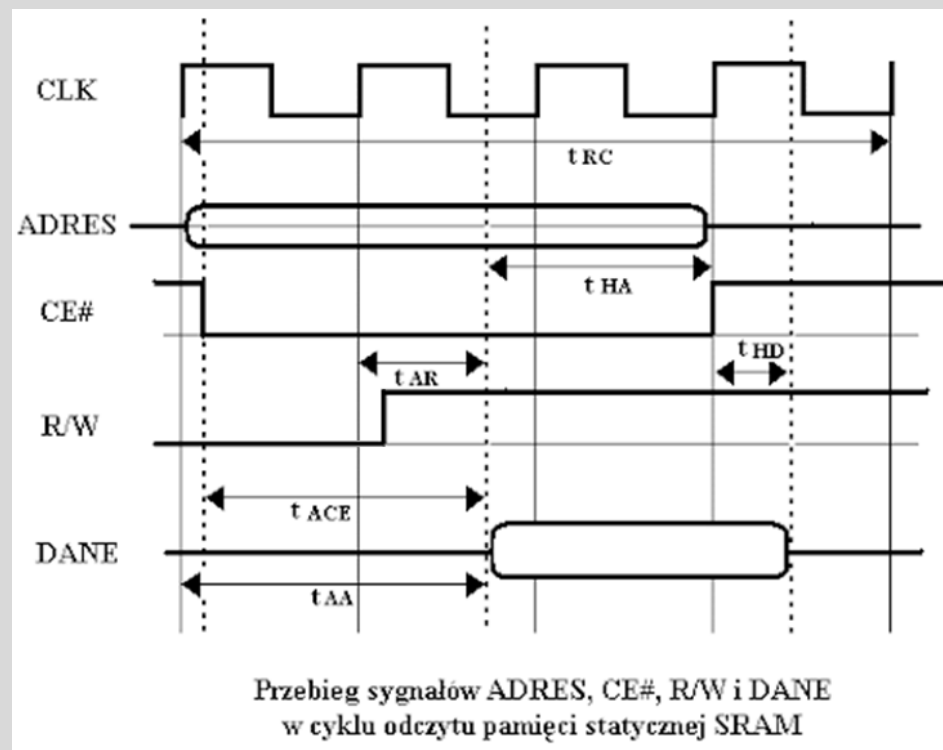


Cykl zapisu do pamięci równoległej SRAM synchronicznej – kolejność ustawiania sygnałów:



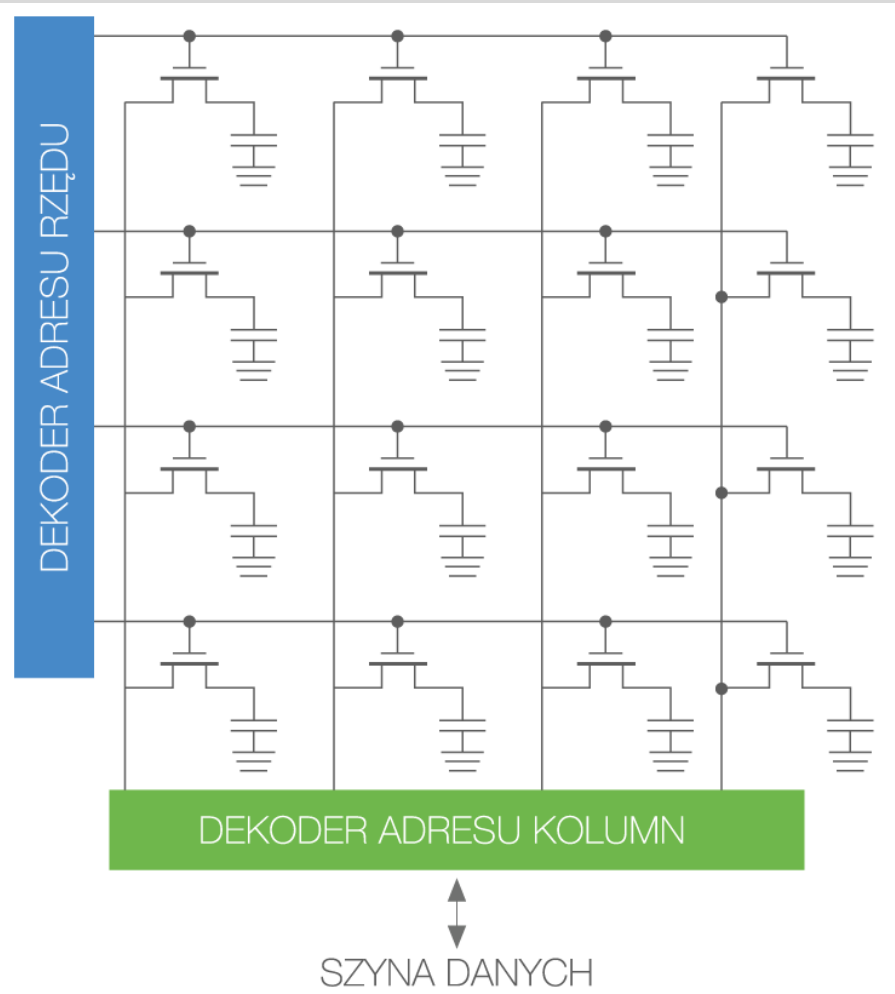
- sygnał zegarowy CLK doprowadzony w sposób ciągły,
- ustawiany jest adres komórki na magistrali adresowej – przy nieaktywnym CE i $R/W = \text{read}/\text{odczyt} = 1$,
- włączany jest logicznie układ – CE przyjmuje poziom aktywny czyli 0,
- ustawiane są dane na magistrali danych – sygnał R/W jak wyżej,
- przełączony zostaje sygnał R/W na write/zapis = 0,
- kolejny impuls zegarowy na zboczu narastającym – wpisuje dane z magistrali danych do pamięci,
- przestawiamy sygnał R/W na odczyt, zdejmujemy dane z magistrali, możemy wyłączyć CE,
- od tego punktu możemy powtórzyć cykl zapisu dla komórki o innym adresie – wracając na początek.

Cykl odczytu z pamięci równoległej SRAM synchronicznej – kolejność ustawiania sygnałów:



- sygnał zegarowy CLK doprowadzony w sposób ciągły,
- ustawiany jest adres komórki na magistrali adresowej – przy nieaktywnym CE i $R/W = \text{read}/\text{odczyt} = 1$,
- włączany jest logicznie układ – CE przyjmuje poziom aktywny czyli 0,
- ustawiane są dane na magistrali danych – sygnał R/W jak wyżej,
- przełączony zostaje sygnał R/W na write/zapis = 0,
- kolejny impuls zegarowy na zboczu narastającym – wpisuje dane z magistrali danych do pamięci,
- przestawiamy sygnał R/W na odczyt, zdejmujemy dane z magistrali, możemy wyłączyć CE.

Pamięć statyczna wymaga aż 6 tranzystorów do pamiętania 1 bitu, przez co zajmuje więcej powierzchni w strukturze. W pamięci dynamicznej 1 bit informacji jest przechowywany za pomocą wartości ładunku zgromadzonego w pojemności kondensatora – czyli w postaci napięcia na kondensatorze.



Zalety pamięci dynamicznych:

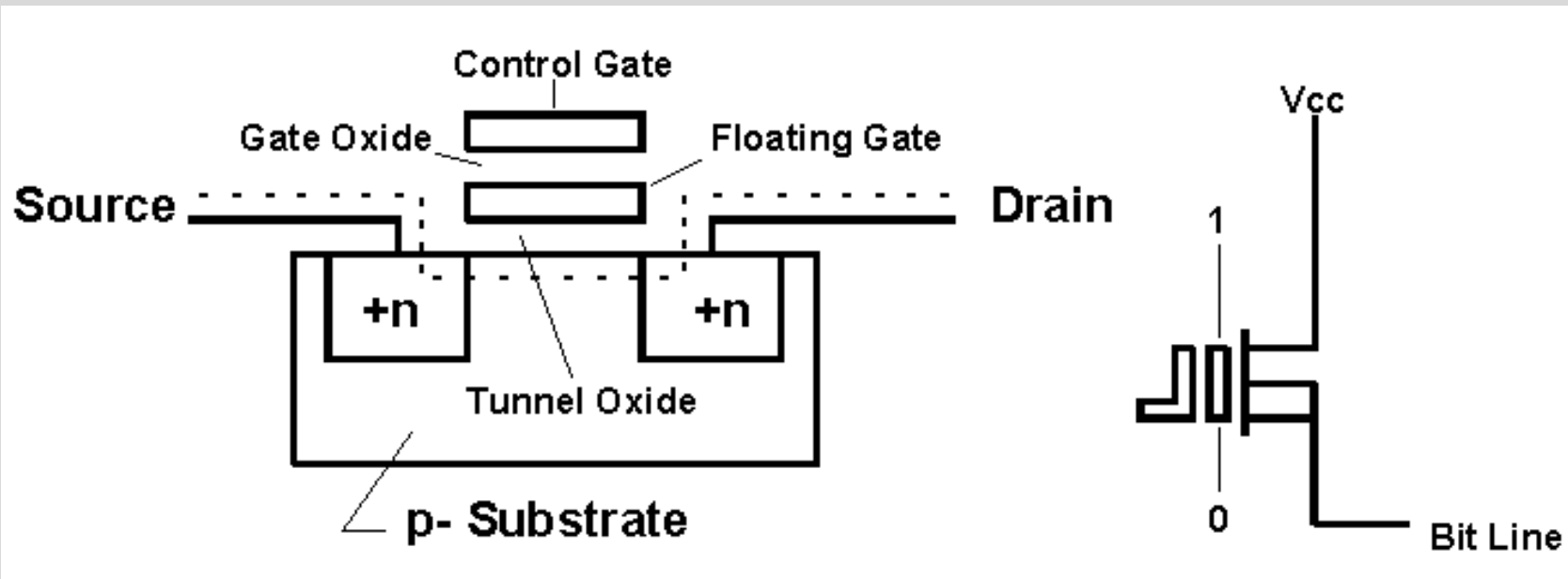
- większa gęstość upakowania,
- niższe koszty.
- większe pojemności (64 Gb DDR5).

Wady pamięci: mniejsza szybkość i większy pobór mocy.



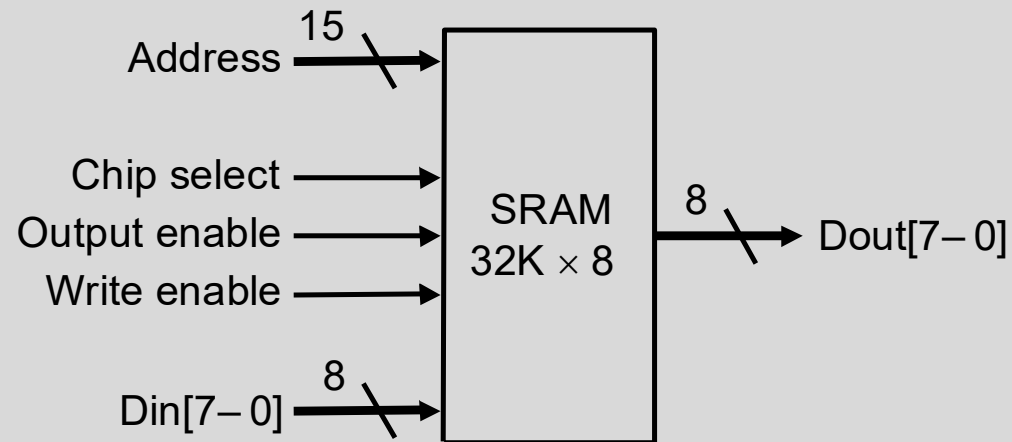
Electrically Erasable Programmable Read-Only (**EEPROM**)– ładunek gromadzony w bramce tranzystora:

- możliwość elektrycznego wielokrotnego kasowania i programowania pamięci,
- programowanie może odbywać się w układzie,
- liczba kasowań około 10 000 - 100 000 razy,
- interfejs szeregowy lub równoległy (pamięć szeregową lub równoległą),
- cykl zapisu jest długi i stosunkowo małe pojemności,
- pamięć wypierana przez pamięci typu Flash.



Pamięć Flash jest modyfikacją pamięci EEPROM, Jest to pamięć nieulotna (non-volatile). Posiada ograniczoną chociaż większą niż EEPROM żywotność. Dane zapisywane są w tablicy komórek opartych na tranzystorach floating-gate. Rozróżnia się dwa rodzaje pamięci Flash:

- NOR Flash - opracowane z myślą zastąpienia ROM: pełny dostęp do dowolnej komórki oraz wolniejszy dostęp sekwencyjny.
- NAND Flash - opracowane z myślą zastąpienia twardych dysków: szybszy dostęp sekwencyjny, dostęp tylko stronicowy/sektorowy, pamięci standaryzowane – różnych producentów.



Zaliczenie cząstkowe wykładu nr 5 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

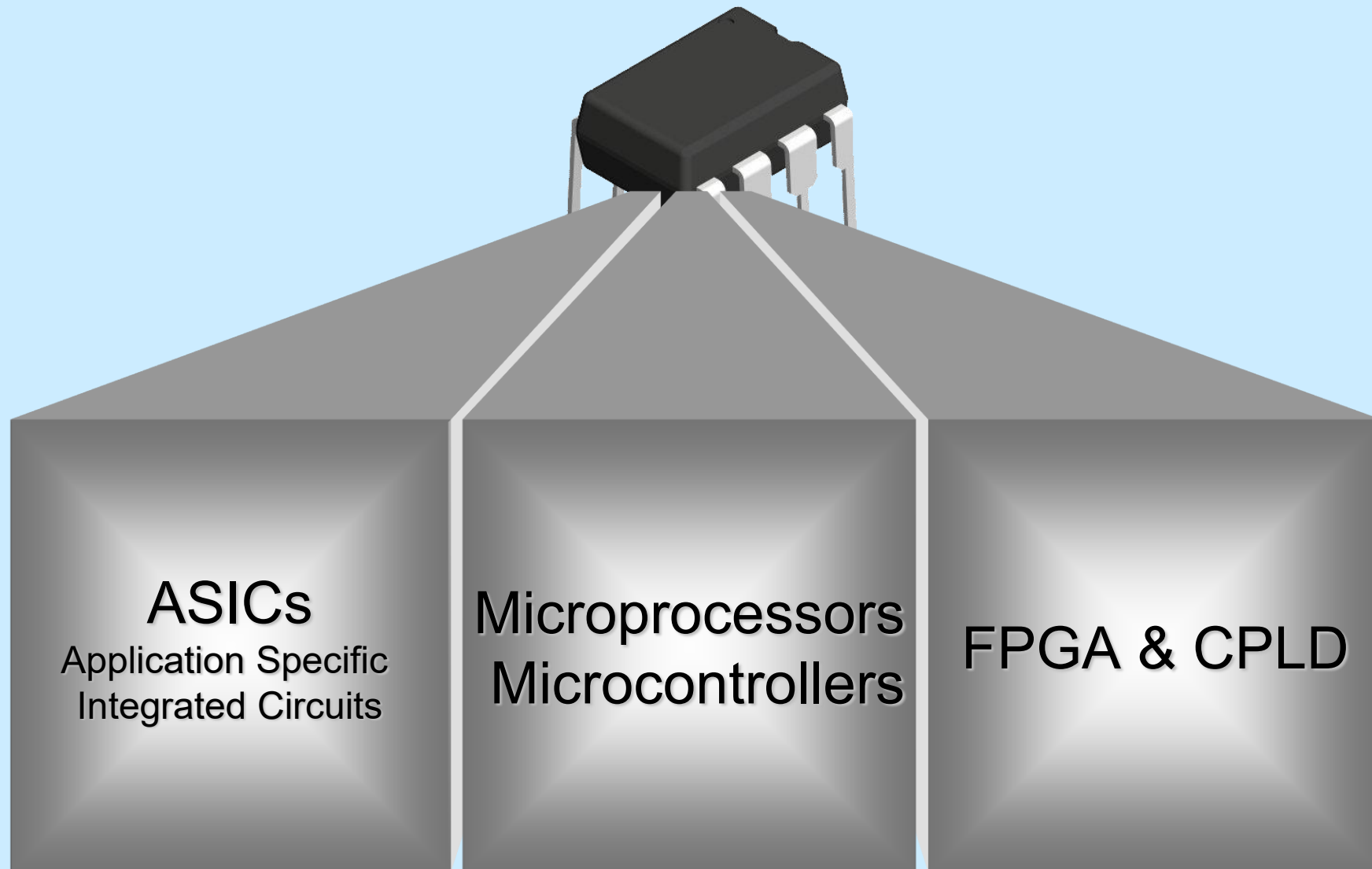
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

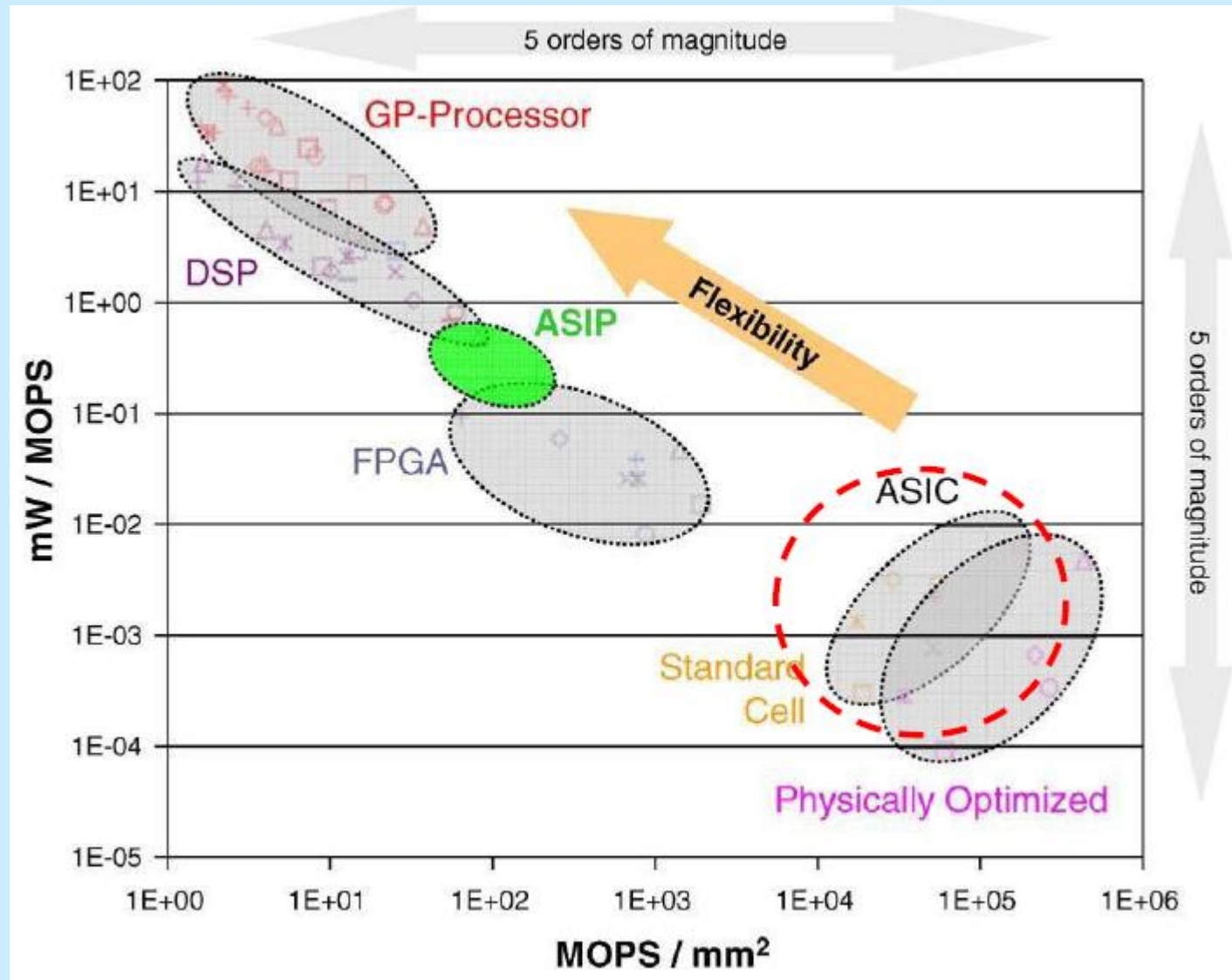
Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

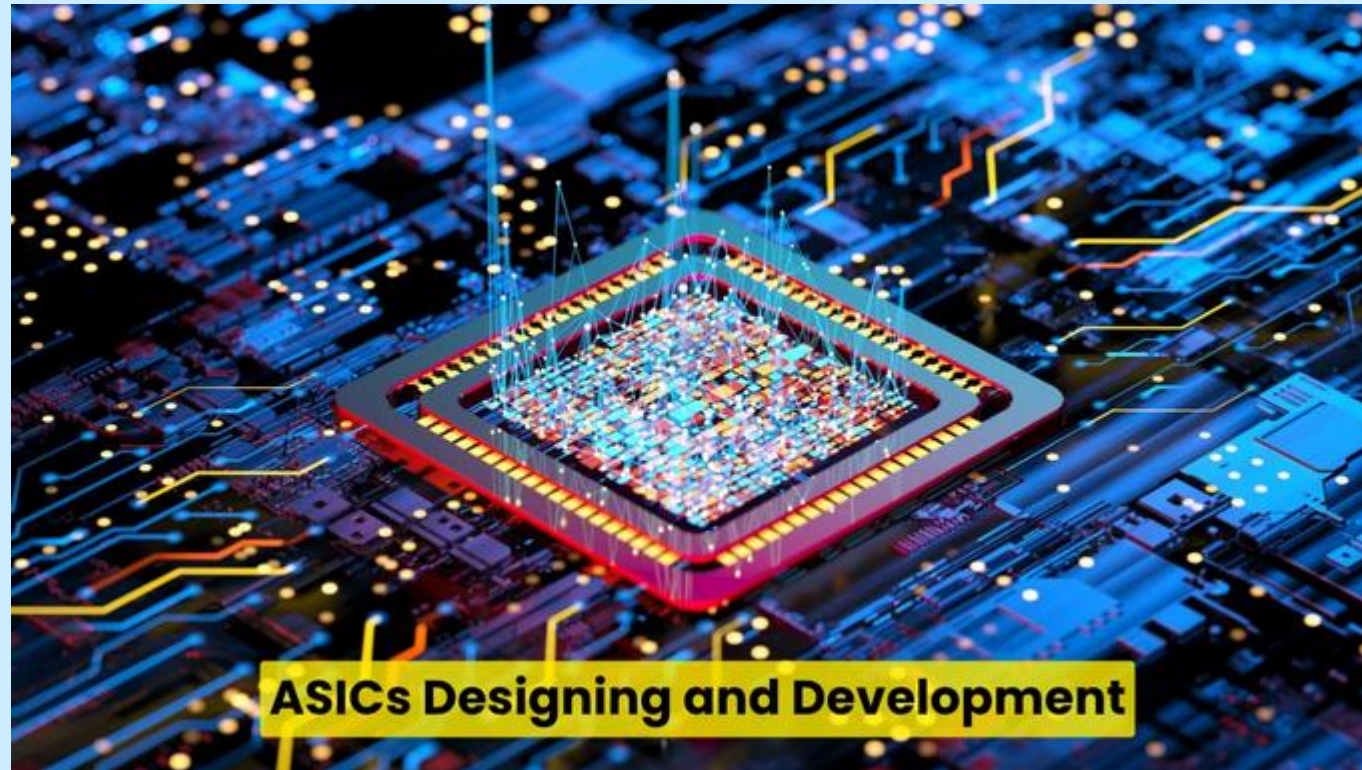
Klasyfikacja układów programowalnych:



Gęstość wydajności i mocy układów, MOPS/mm² - Million Operations Per Second per square millimeter



Układ ASIC - to specjalnie zaprojektowany układ scalony przeznaczony do konkretnego zastosowania. W przeciwieństwie do układów scalonych ogólnego przeznaczenia (takich jak procesory), które mogą być używane w różnych urządzeniach, są one zazwyczaj tworzone od podstaw, w oparciu o konkretne potrzeby zastosowania, do którego są przeznaczone. Ich zalety to: mniejszą objętość niż układy scalone ogólnego przeznaczenia, mniejsze rozmiary i zużycie energii, większa niezawodność, duża wydajność, większa poufność, niższy koszt.



Historia ASIC sięga lat 70 ubiegłego wieku. W 1974 roku Robert Lipp opracował pierwszą matrycę bramek CMOS, dynamiczny rozwój układów ASIC nastąpił w latach 80, gdy firmy VLSI Technology i LSI Logic opracowały narzędzia komputerowego wspomaganie projektowania (CAD) w celu uproszczenia projektowania układów scalonych do indywidualnych potrzeb. Niestandardowe układy ASIC zabezpieczają specyficzne potrzeby użytkowe, których nie są w stanie w pełni zaspokoić gotowe komponenty.

Główne obszary zastosowań układów ASIC to:

- szybkie wykonywanie algorytmów,
- elektronika użytkowa - w smartfonach oraz konsolach do gier i systemach identyfikacji tożsamości,
- sztuczna inteligencja i uczenie maszynowe,
- telekomunikacja i sieci komputerowe – zarządzanie ruchem sieciowym i szyfrowanie danych,
- systemy samochodowe - systemy wspomaganie kierowcy (ADAS) i zarządzania silnikiem/akumulatorami,
- urządzenia medyczne - urządzenia diagnostyczne i wszczepialne oraz monitorowania on-line,
- automatyka przemysłowa – robotyka i systemy kontroli procesów,
- obronność państwa oraz lotnictwo i zastosowania kosmiczne.

Cechą układów ASIC, jest lepsza ochrona własności intelektualnej. Ta właściwość jest wbudowana na poziomie sprzętowym, przez co jest ona z natury bardziej odporna na kopiowanie lub inżynierię odwrotną. Ma to szczególne znaczenie dla firm technologicznych, które postrzegają unikalne cechy swoich produktów jako podstawową przewagę konkurencyjną. Układy ASIC służą nie tylko jako narzędzie poprawiające wydajność produktów, ale także jako zabezpieczenie zastrzeżonych technologii i procesów.

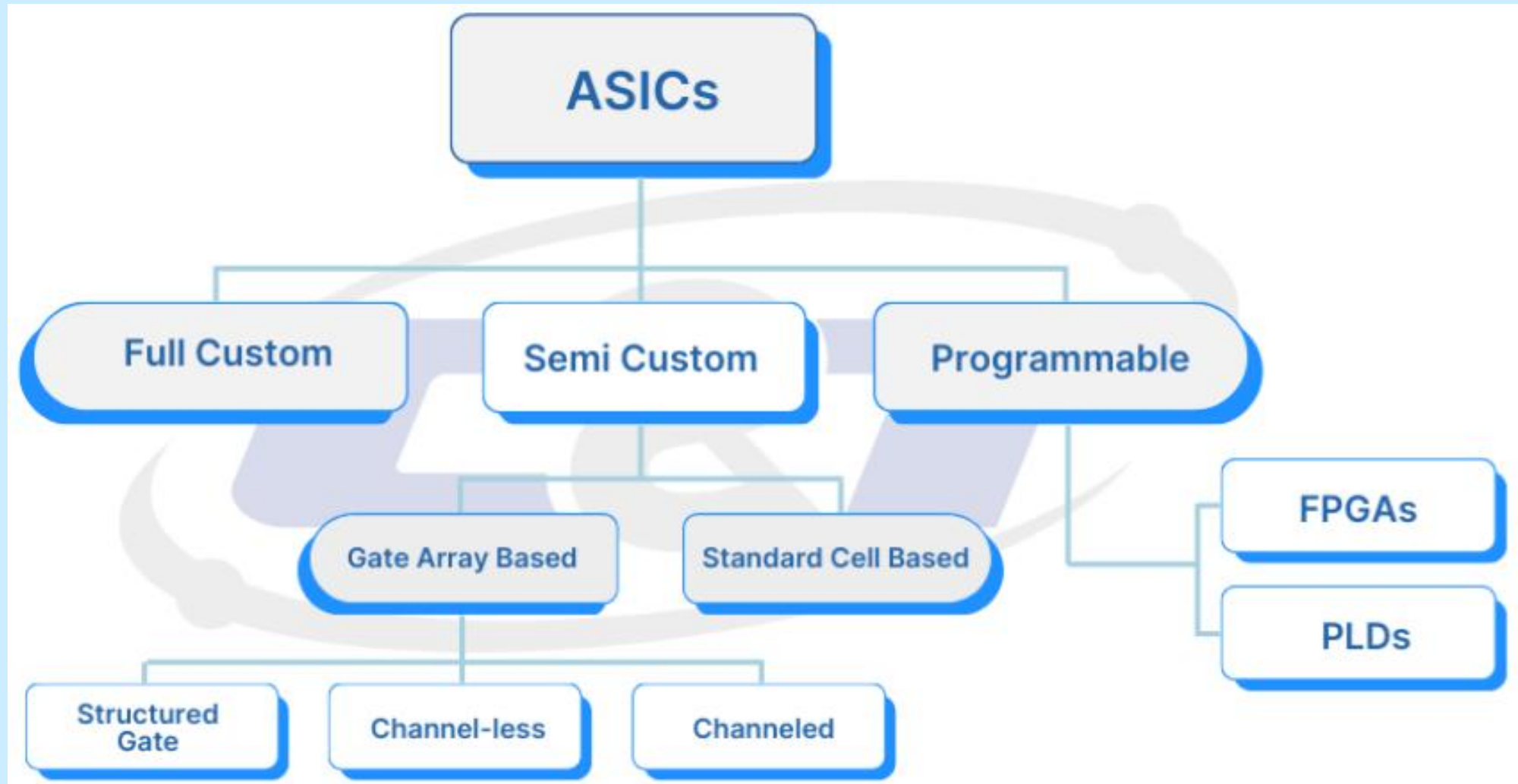
Zalety ASIC:

- wysoka wydajność: wyższa szybkość i wydajność,
- niższe zużycie energii: konstrukcja zmniejsza zużycie energii zaleta w zastosowaniach bateryjnych,
- integracja wielu funkcji w jednym układzie zmniejsza złożoność – technologia 3/5/7/14 nm,
- opłacalność dużych partii zastosowań.

Wady ASIC:

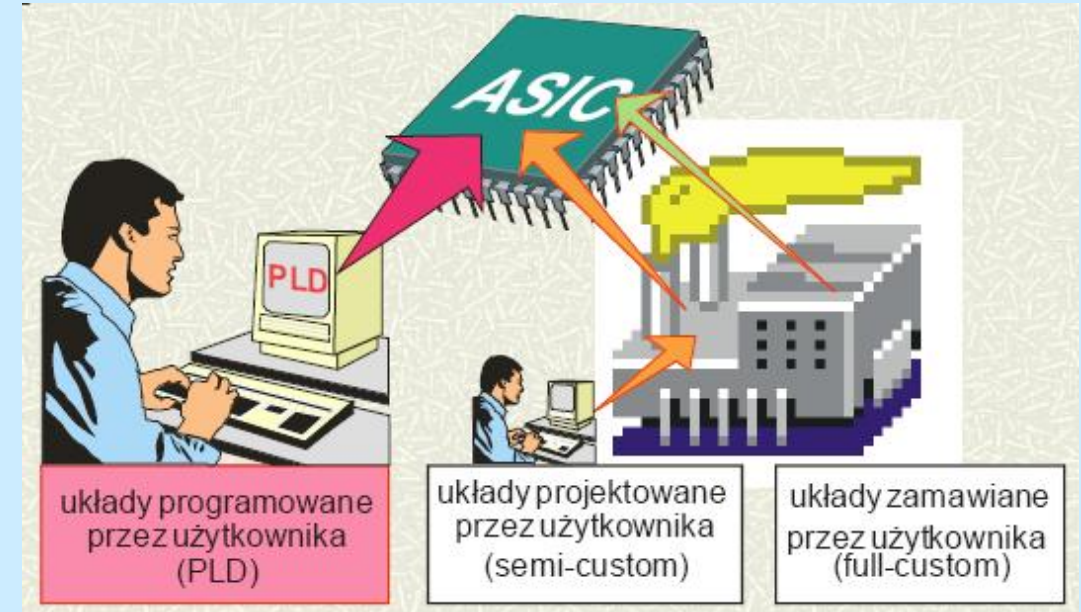
- brak elastyczności - układy ASIC nie mogą być przeprogramowywane,
- wysokie koszty opracowania technologii:
- długi czas rozwoju – miesiące,
- szybkie zmiany technologiczne będą powodować szybsze starzenie układów,
- ograniczona skalowalność - mogą nie dostosowywać się do przyszłych potrzeb.

Rodzaje układów ASIC:



Rodzaje układów ASIC:

Full-custom: w pełni niestandardowe/oryginalne opracowania realizowane od podstaw przez wyspecjalizowane firmy. Jest to rodzina układów scalonych o najwyższym stopniu dostosowania do wymagań. Firmy projektują struktury na bazie gotowych bibliotek obwodów analogowych, jednostki pamięci i struktury mechaniczne na platformie jednego układu scalonego. Dzięki niestandardowości są to najbardziej zoptymalizowane układy w zakresie wydajnością i zużycia energii. Średnia moc obliczeniowa full-custom układu ASIC jest około 8 razy większa niż w przypadku układu semi-custom. Najlepsza wydajność i efektywność tej rodziny układów, wiążą się z wysokimi kosztami rozwoju i dłuższym czasem projektowania. Są one stosowane w produktach, w których wydajność ma kluczowe znaczenie, takich jak układy AI lub szybkie urządzenia sieciowe.



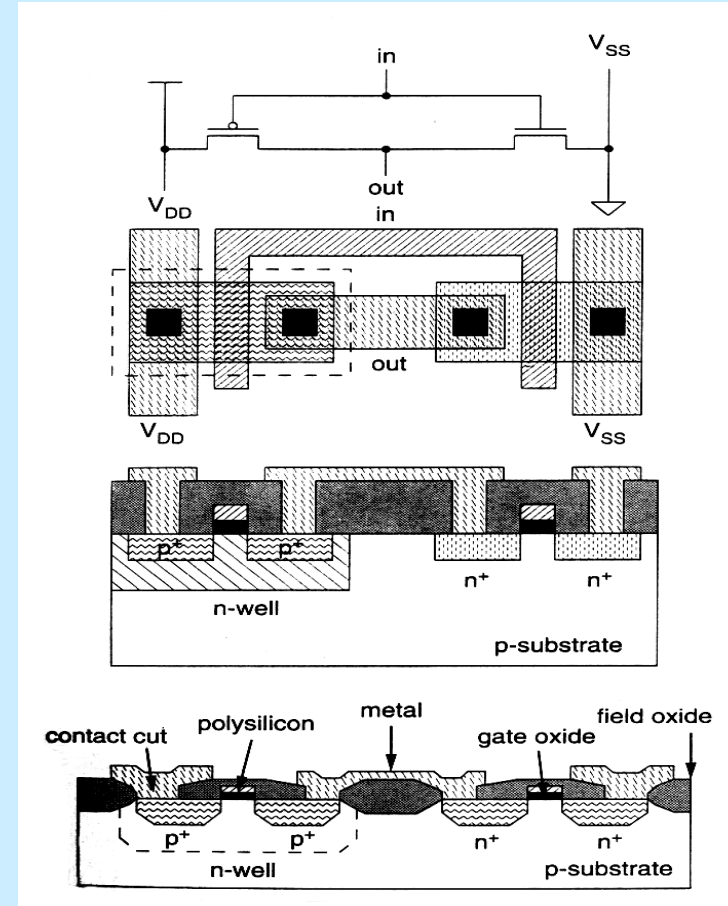
Niektóre z full-custom ASIC to: mikroprocesory, pamięci, procesory analogowe, urządzenia komunikacyjne analogowe/cyfrowe, czujniki, przetworniki, IC samochodów itp.

Cechy technologii full-custom ASIC:

- technika projektowania małych systemów,
- sposób optymalizacji na poziomie analogowym,
- długi cykl projektowy,
- nie występują struktury CPLD i FPGA.

Opracowanie układów ASIC obejmuje dwa kluczowe rodzaje kosztów – koszty NRE i kosztów produkcji. Koszty etapu projektowania NRE (Non-Recurring Engineering) są ponoszone jednorazowo i obejmują badania, projektowanie, prototypowanie i wstępne testy. Są to koszty niezależne od liczby wytwarzanych sztuk.

Drugą kategorię stanowią koszty produkcji, które powtarzają się z wytworzeniem każdego egzemplarza układu ASIC. Obejmują one koszt surowców, produkcji, pakowania, testowania i zapewnienia jakości. Koszty produkcji w pewnym przedziale maleją wraz ze wzrostem wielkości produkcji, dzięki efektowi skali.

CMOS Inverter in out

dekompozycja
i trasowanie
połączeń

maskowanie

synteza
struktury

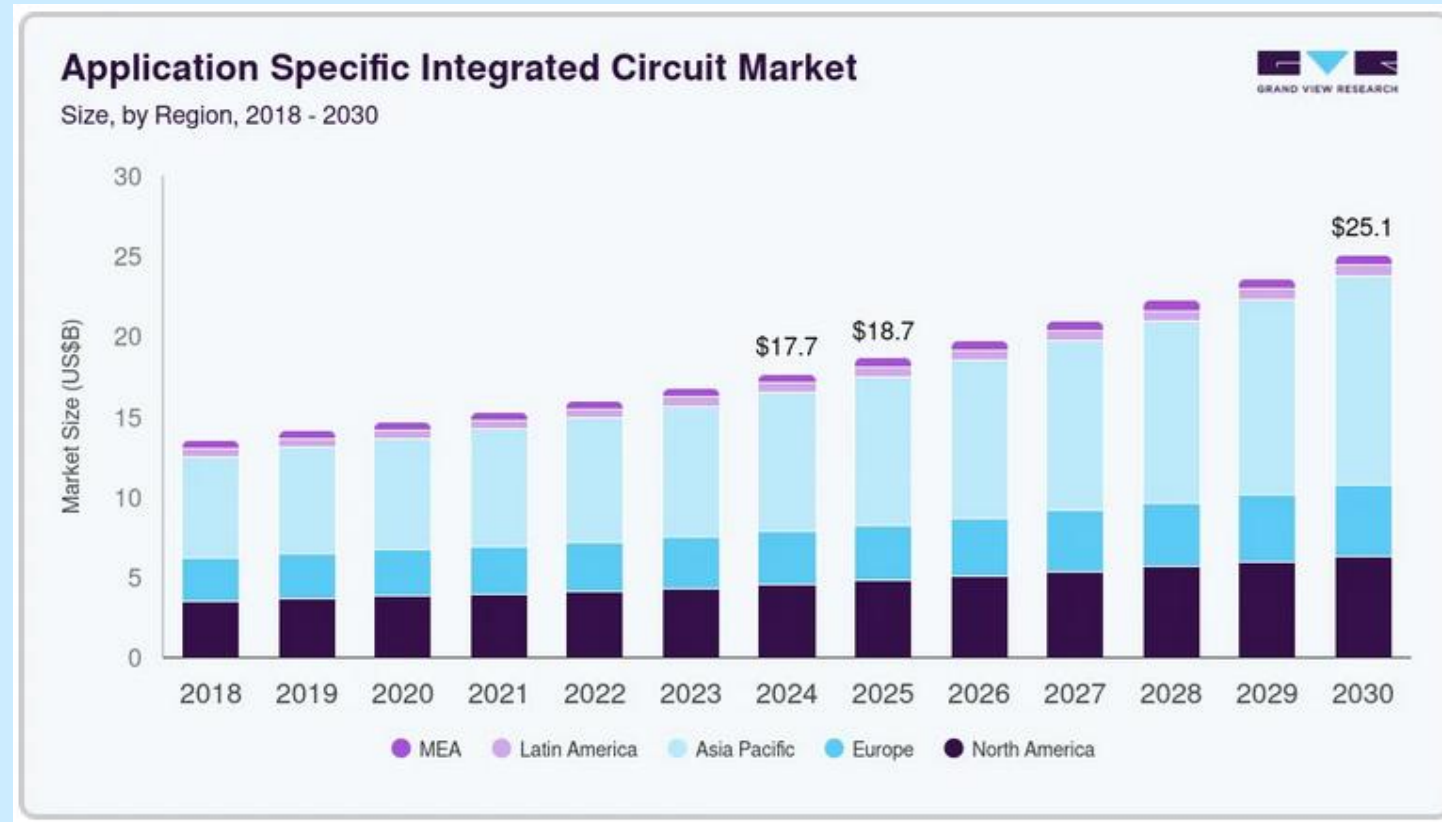
pakowanie, kontrola jakości

Rodzaje układów ASIC:

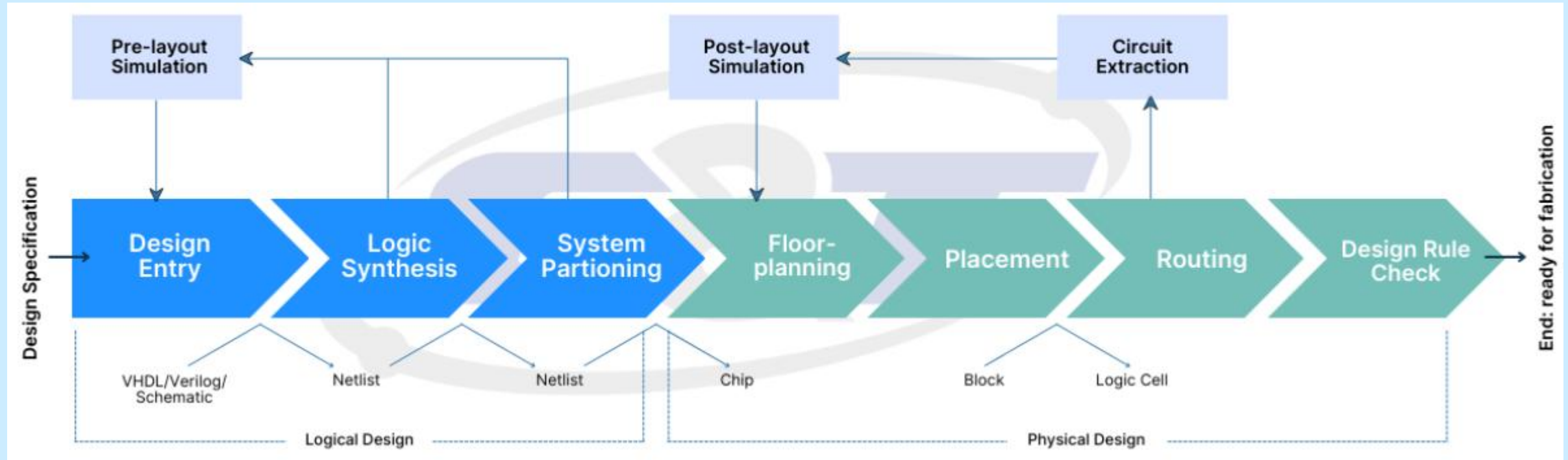
Semi-custom: powstały żeby skrócić czas projektowania i obniżyć koszty w układach full-custom ASIC. Zazwyczaj najniższym poziomem hierarchii w projektach semi-custom jest poziom logiczny lub poziom bramek. Odróżnia to od układów full-custom, w których projektowanie odbywa się na poziomie poszczególnych tranzystorów. Układy semi-custom ASIC dzielą się dalej na układy z komórkami standardowymi i układy ASIC z macierzą bramek.

W procesach projektowania tych układów używa się języków HDL (VHDL, Verilog).

W 2024 roku rynek wszystkich kategorii układów ASIC wyniósł \$17.7B.



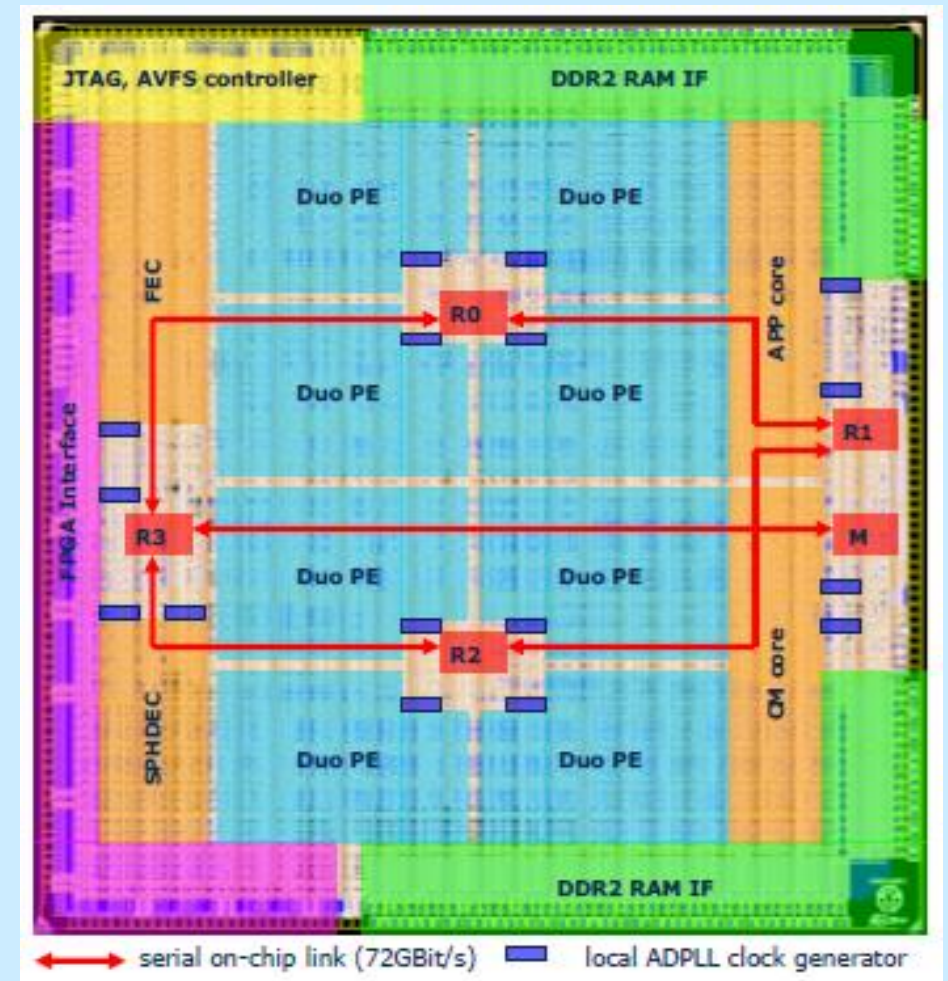
Np. układy scalone mieszane, zawierające komponenty analogowe i cyfrowe, można precyzyjnie dostosować w układach ASIC do konkretnych zadań przetwarzania sygnałów, co jest mniej skuteczne z oddzielnych standardowych układów.



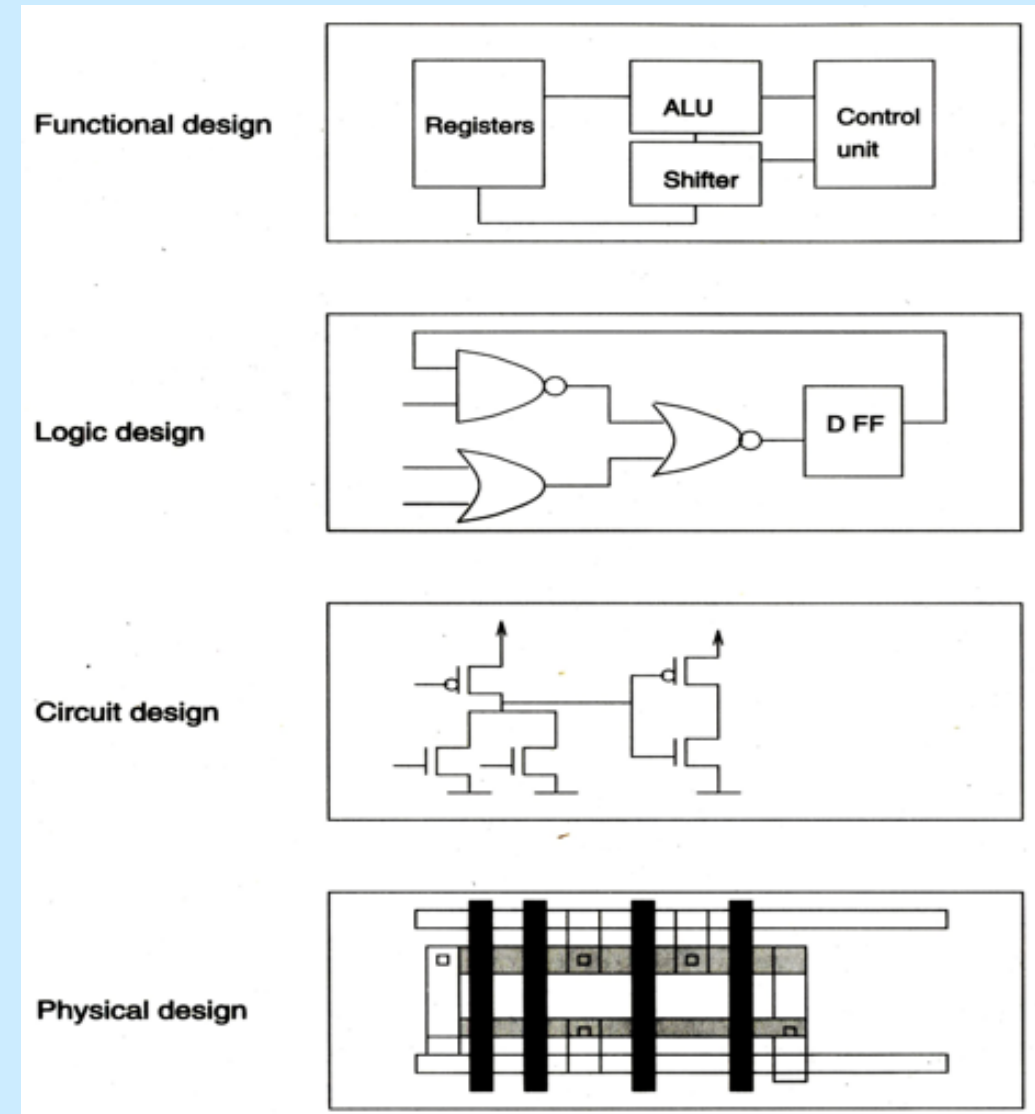
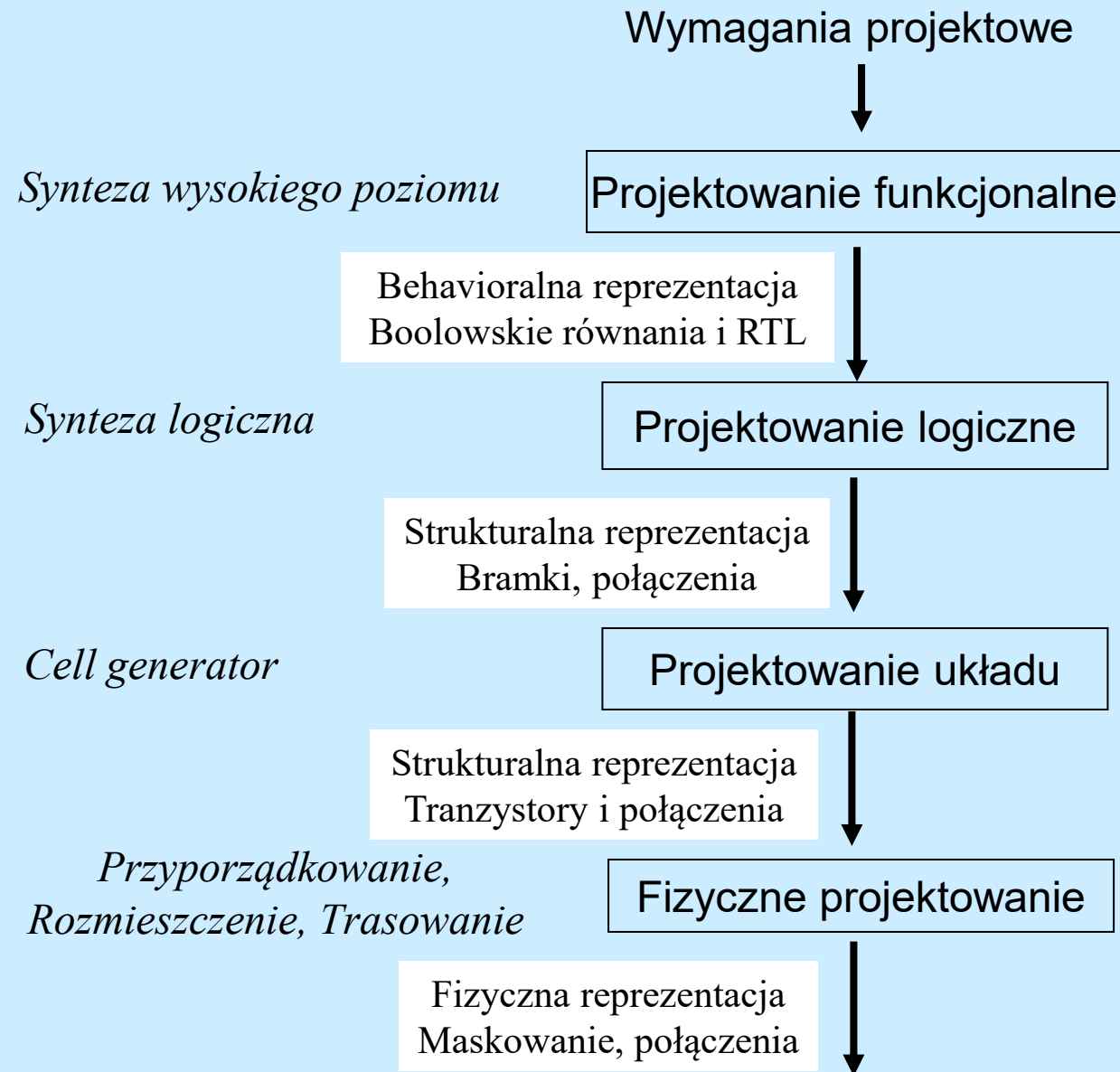
- Projektowanie wstępne - opracowanie mikro-architektury przy języków VHDL, Verilog i System Verilog.
- Synteza logiczna - tworzenie listy sieciowej komórek logicznych, połączeń, komponentów w HDL.
- Dekompozycja systemu - duży układ jest dzielony na bloki wydajniejsze w projektowaniu i przetwarzaniu.
- Symulacja przed rozmieszczeniem - symulacja w celu zidentyfikowania i skorygowania błędów w projekcie.
- Planowanie rozmieszczenia (Floor Planning) – wstępne rozmieszczenie elementów listy sieciowej w chipie.
- Rozmieszczenie - etap określenia dokładnych lokalizacji komórek w każdym bloku.
- Trasowanie połączeń i ocena właściwości elektrycznych połączeń jak rezystancja i pojemność.
- Symulacja po rozmieszczeniu - końcowe testowanie funkcjonalności całego systemu przed produkcją.

Tomahawk2 procesor programowo zdefiniowanego radia:

- powierzchnia chipa 36 mm²,
- 465 wyprowadzeń/pinów,
- 10,2 mln bramek przeliczeniowych,
- 750 kB pamięci SRAM,
- 20 rdzeni procesorowych,
- adaptacyjne i dynamiczne zarządzanie energią,
- 17 generatorów zegarowych,
- sieciowe złącze szeregowe 72 GBit/s,
- interfejs pamięci DDR,
- interfejs FPGA 10 GBit/s.

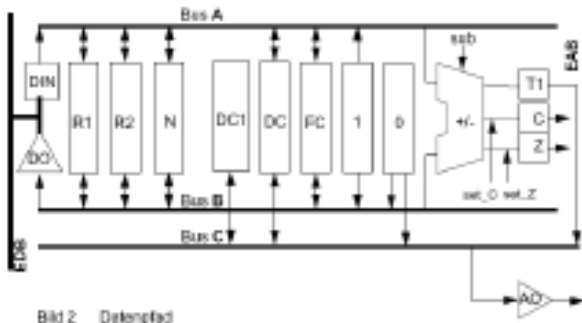


Zanim zostanie wyprodukowany chip musi nastąpić pełny cykl projektowy:



... a w nim opracowanie specyfikacji z użyciem języka opisu sprzętowego HDL (VHDL, Verilog)

Specification



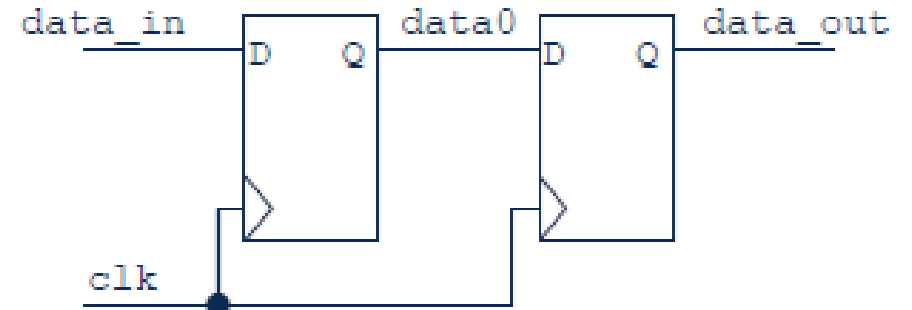
```
module fsm (  
  module decoder (  
    module adder_reg (clk_i, a_i, b_i, c_o);  
      input clk_i;  
      input [7:0] a_i, b_i;  
      output [8:0] c_o;  
      reg [8:0] c_r;  
      always @(posedge clk_i) begin  
        c_r<=a_i+b_i;  
      end  
      assign c_o=c_r;  
    endmodule  
  endmodule
```



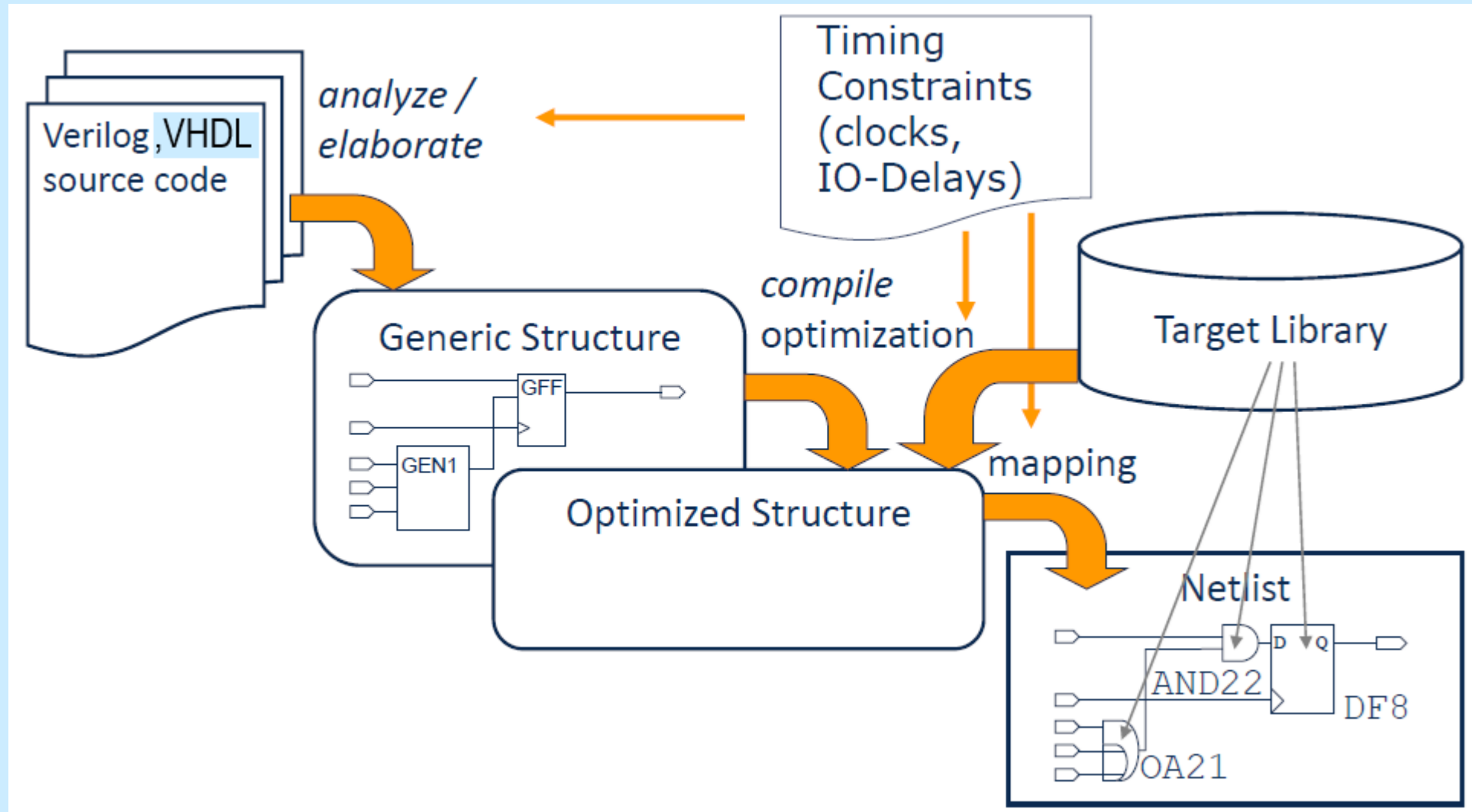

... i opracowanie schematu połączeń dla poszczególnych modułów kodu HDL na poziomie opisu RTL. Jest to lista/netlist bramek i przerzutników, które są oparte na standardowych bibliotekach makrokomórek. Netlist zawiera wszystkie informacje o funkcjonalności układu/systemu, zależnościach czasowych, wymaganej mocy, częstotliwości pracy, wejściach i wyjściach do układu.

```
always @(posedge clk)
begin
    data0 <= data_in;
end
always @(posedge clk)
begin
    data_out <= data0;
end
```

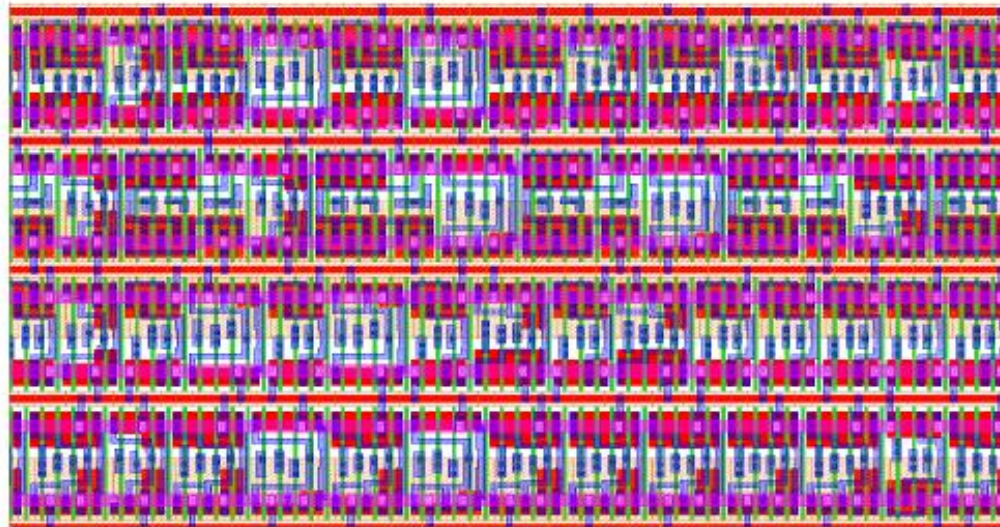
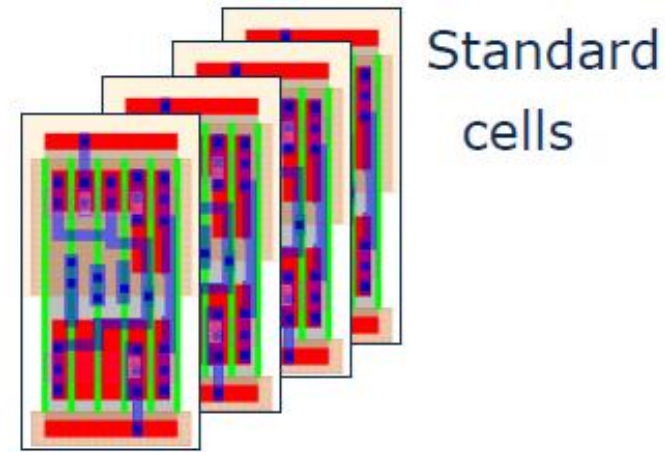
RTL Synthesis



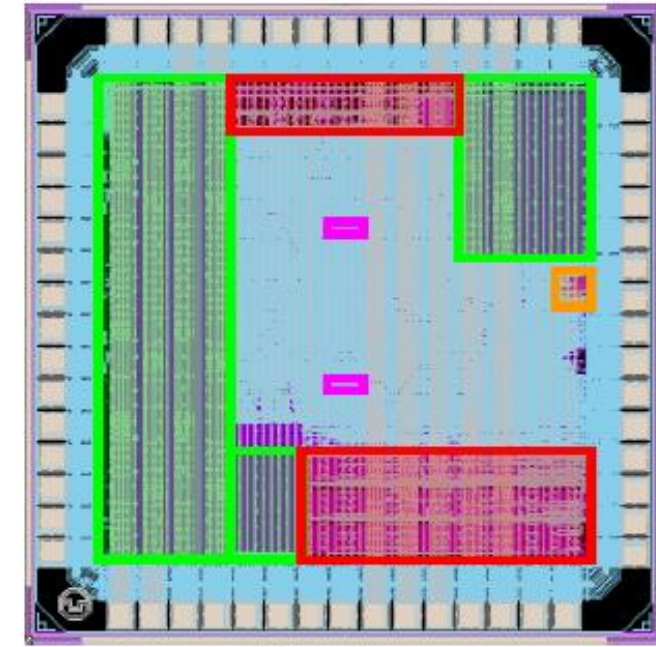
Podsumowując – synteza logiczna składa się z poniższych etapów, powiązań między etapami oraz uwarunkowań czasowych i bibliotecznych.



W fazie syntezy układu następuje automatyczne rozmieszczanie (Placement) oraz tworzenie połączeń (trasowanie – Routing) standardowych komórek. Proces kończy synteza sieci zegarowych i redukcja zbędnych elementów.

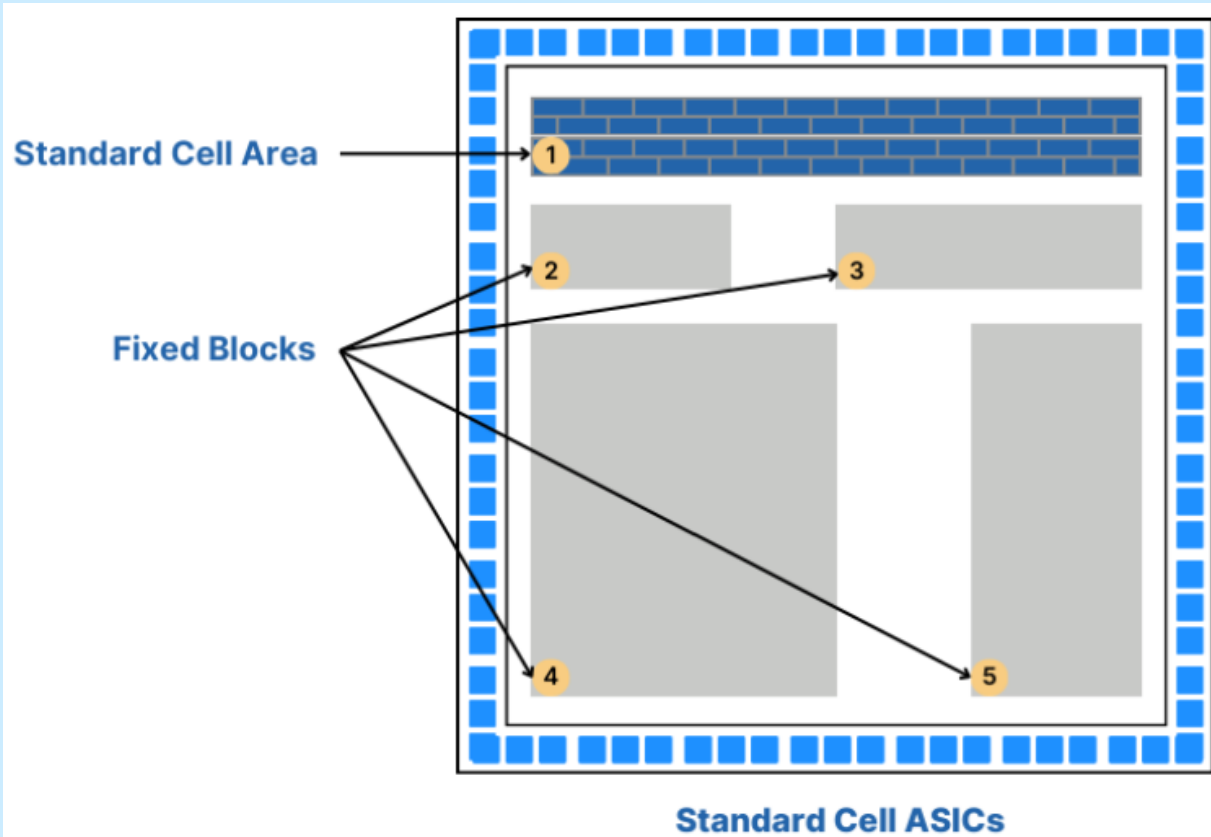


Placement



Routing

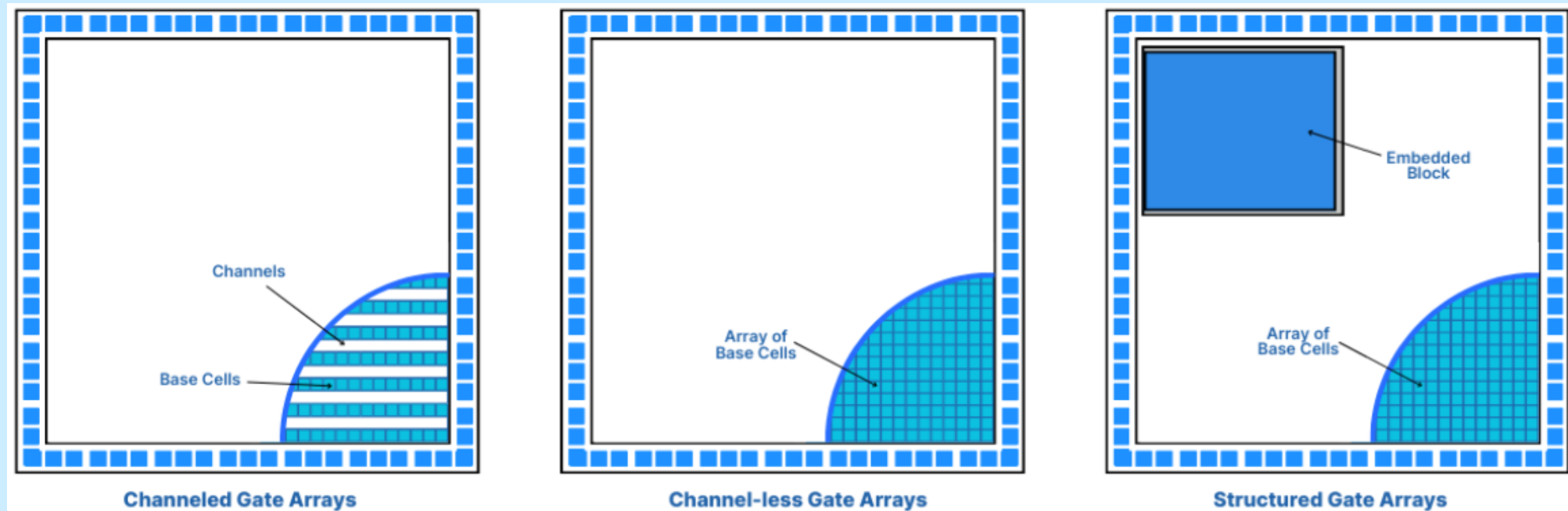
Układy rodziny **Full-custom - Standard Cell ASICs** – oparte są na bibliotece znormalizowanych komórek, które zawierają wstępnie zaprojektowane elementy logiczne, takie jak bramki AND i OR, multipleksery i przerzutniki. Układ ASIC zazwyczaj zawiera obszar komórek standardowych lub uniwersalny blok, złożony z tych komórek ułożonych w rzędach. Może on też zawierać **mega-komórki**, takie jak mikrokontrolery lub mikroprocesory. Warstwy maski standardowej komórki ASIC można dostosowywać efektywnie wykorzystując przestrzeń.



Koszty produkcji układów ASIC obejmują:

- Materiały - surowce potrzebne do budowy układu, w tym podłoże półprzewodnikowe i inne elementy;
- Produkcja - koszty ponoszone przez fabrykę półprzewodników zależne od złożoności chipa,
- Opakowanie - koszty ochrony chipa i przygotowania do integracji końcowej;
- Kontrola jakości i testowanie - koszty weryfikacji spełnienia specyfikacji i standardów wydajności;
- Logistyka - opłaty za wysyłkę i dystrybucję.

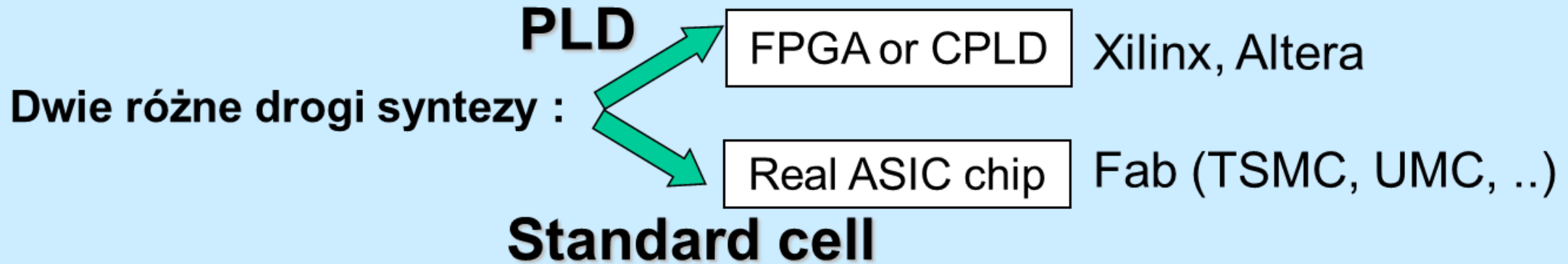
Układy rodziny **Full-custom - Gate Array ASICs** – są rodzajem semi-custom ASIC z predefiniowanymi tranzystorami na płycie krzemowej, gdzie projektant nie zmienia rozmieszczenia tranzystorów, ale połączenia między nimi. W projekcie wykorzystuje się bibliotekę bramek do konfiguracji - zwanych jako: Channeled, Channel-less lub Structured Gate Arrays. Kanałowe macierze bramek (Channeled) stosują predefiniowane kanały połączeniowe między komórkami, dedykowane dla typowych projektów z elastycznymi ścieżkami łączącymi. Bez-kanałowe macierze bramek (Channel-less): nie mają predefiniowanych kanałów routingu, co pozwala na tworzenie połączeń bezpośrednio nad komórkami, zwiększając gęstość upakowania chipów. Strukturalne macierze bramek (Structured): łączą predefiniowane bloki logiczne z konfigurowalnymi warstwami połączeń.



Programmable ASICs – do tej kategorii zaliczane są często Programowalne struktury logiczne (PLD) i układy FPGA. Układy PLD można zaprogramować do wykonywania szerokiego zakresu operacji logicznych. Z kolei układy FPGA można przeprogramować wielokrotnie do różnych zastosowań.

Najczęściej termin **Programmable ASICs** oznacza sytuację w której użytkownik częściowo projektuje układ w języku HDL i weryfikuje go w FPGA – a następnie wysyła do producenta, który optymalizuje i wytwarza chip.

elastyczna, krótki cykl projektowy, odpowiednia dla krótkich serii



sztywna, długi cykl projektowy, długie serie redukujące koszt jednostkowy

Różnice w przetwarzaniu sygnałów z użyciem mikroprocesora i programowalnej struktury logicznej PLD/FPGA:



```

mov BX,K           ; K - współczynnik podziału
mov AL,0B6H
out 43H,AL         ; ustalenie konfiguracji dzielnika
mov AL,BL          ; AL- mniej znaczący bajt K
out 42H,AL         ; wysłanie do portu 42H
mov AL,BH          ; AL- bardziej znaczący bajt K
out 42H,AL         ; wysłanie do portu 42H
in AL,61H          ; wczytanie zawartości portu 61H
or AL,11B          ; ustawienie dwa najniższe bity na 1
out 61H,AL         ; i wysłanie do portu 61H
...               ; dowolne operacje <dźwięk teraz gra>
in AL,61H          ; do AL wartość portu 61H
and AL,11111100B   ; wyzerowanie dwóch najniższych bitów
out 61H,AL         ; i wysłanie do portu 61H
    
```

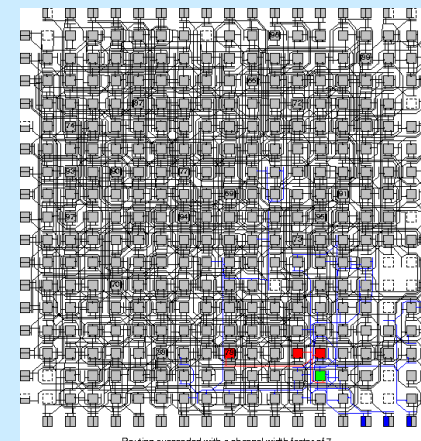
- wynikiem kompilacji jest ciąg instrukcji kodu maszynowego, wykonywanych w takt sygnału zegarowego



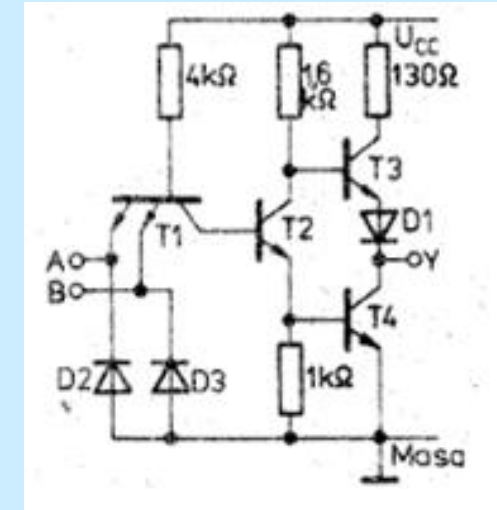
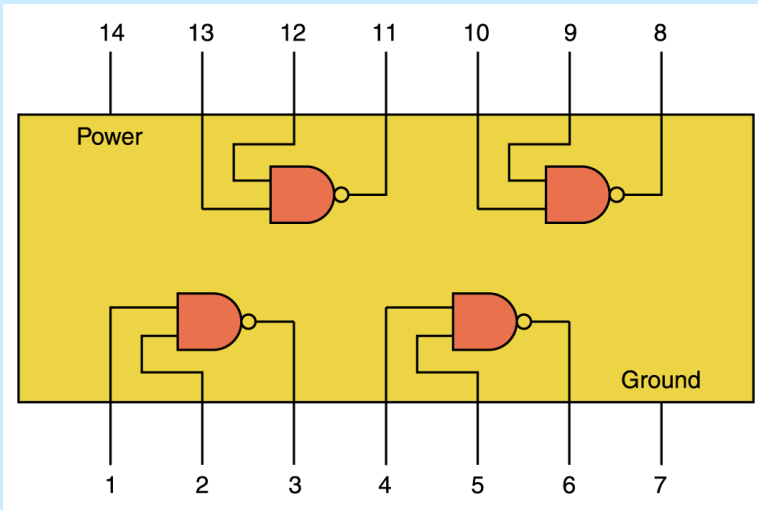
```

begin
  process (C,R)
    variable tq : bit;
    begin
      if R = '1' then
        tq := '0';

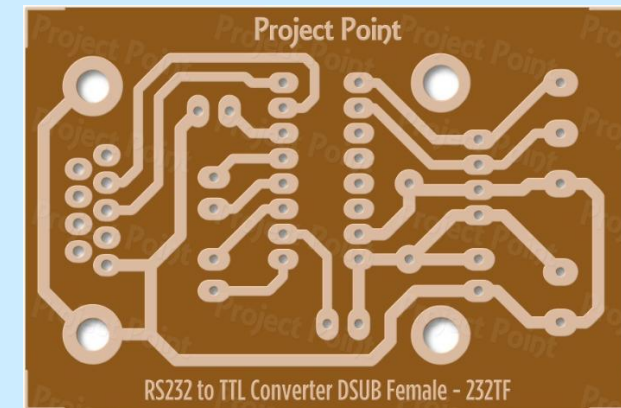
        elsif (C'event and C = '1' and T = '1') then
          tq := not tq;
        end if;
        Q <= tq;
      end process;
    end al;
  
```



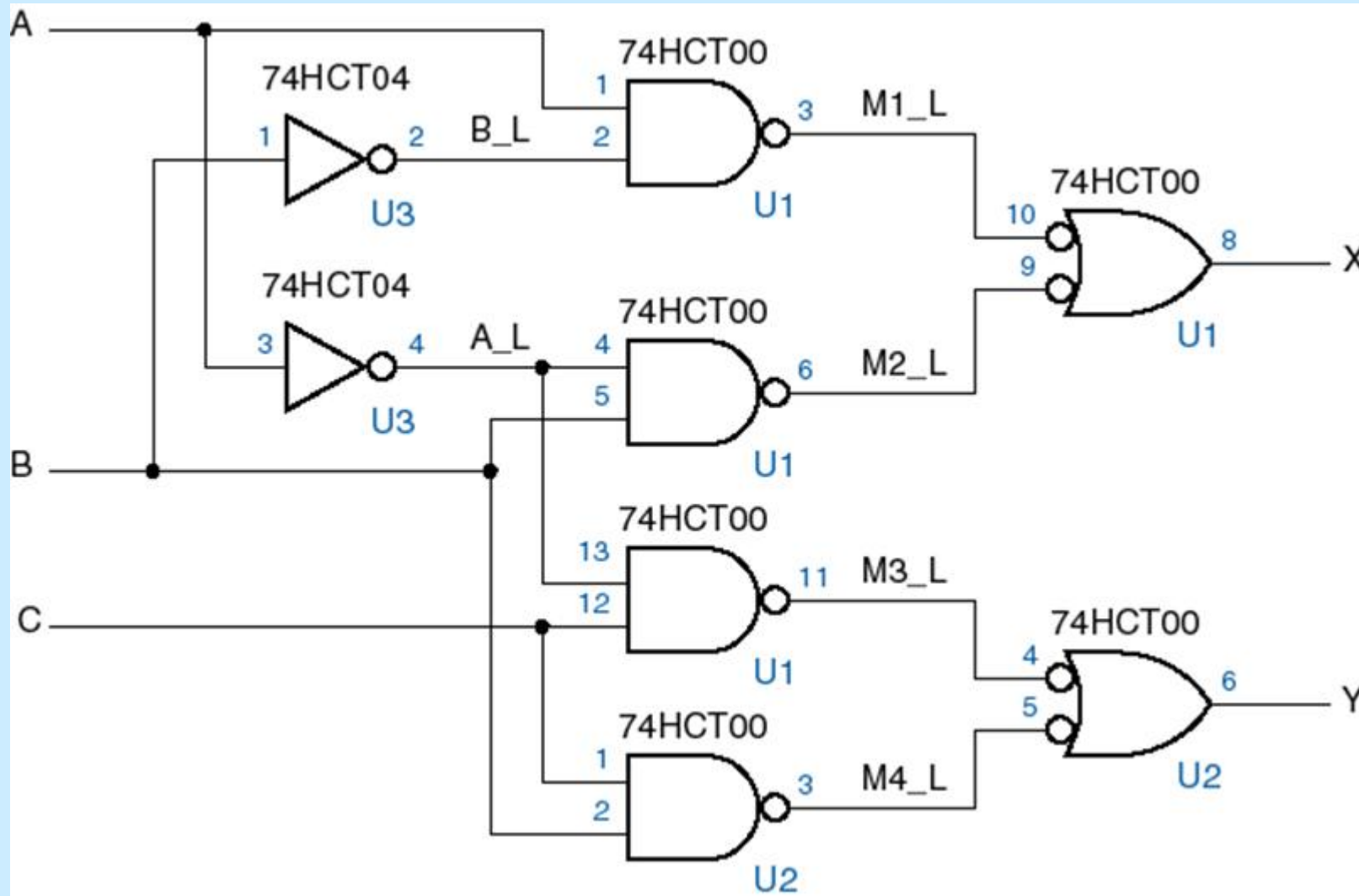
Pierwsze bramki TTL zostały opracowane w latach 60-tych ubiegłego wieku.



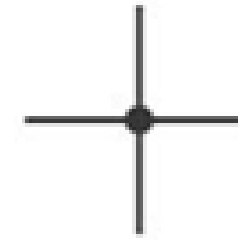
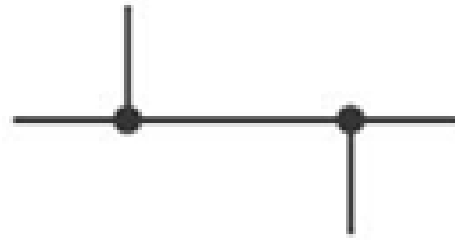
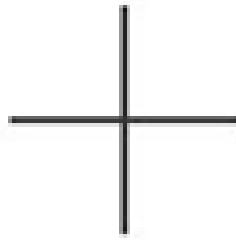
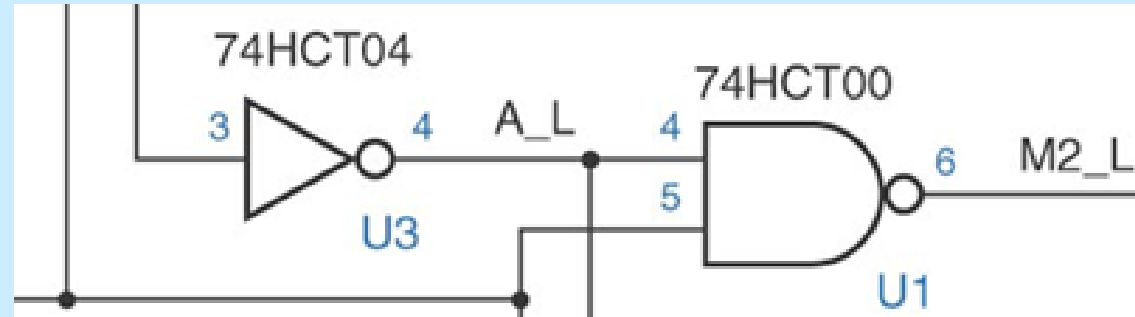
Wady technologii: mała gęstość koncentracji elementów/modułów w układzie scalonym, duże moce strat cieplnych, brak możliwości rekonfiguracji/modyfikacji połączeń.



Podstawą wykonania konstrukcji było opracowanie schematu:



Połączenia sygnałowe na schemacie - wires:

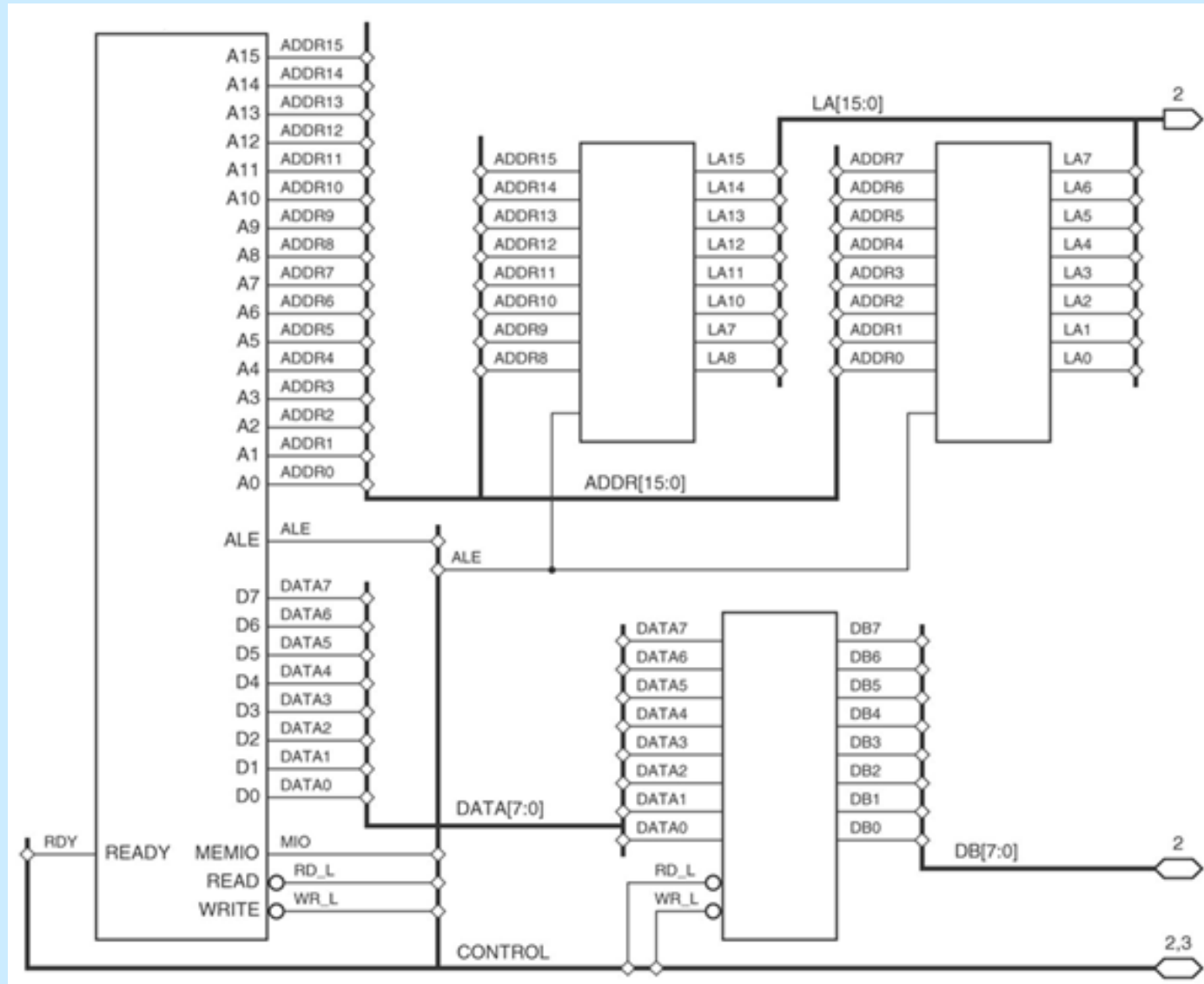


przejście
sygnałów

dołączenie
do linii

połączenie
krzyżowe

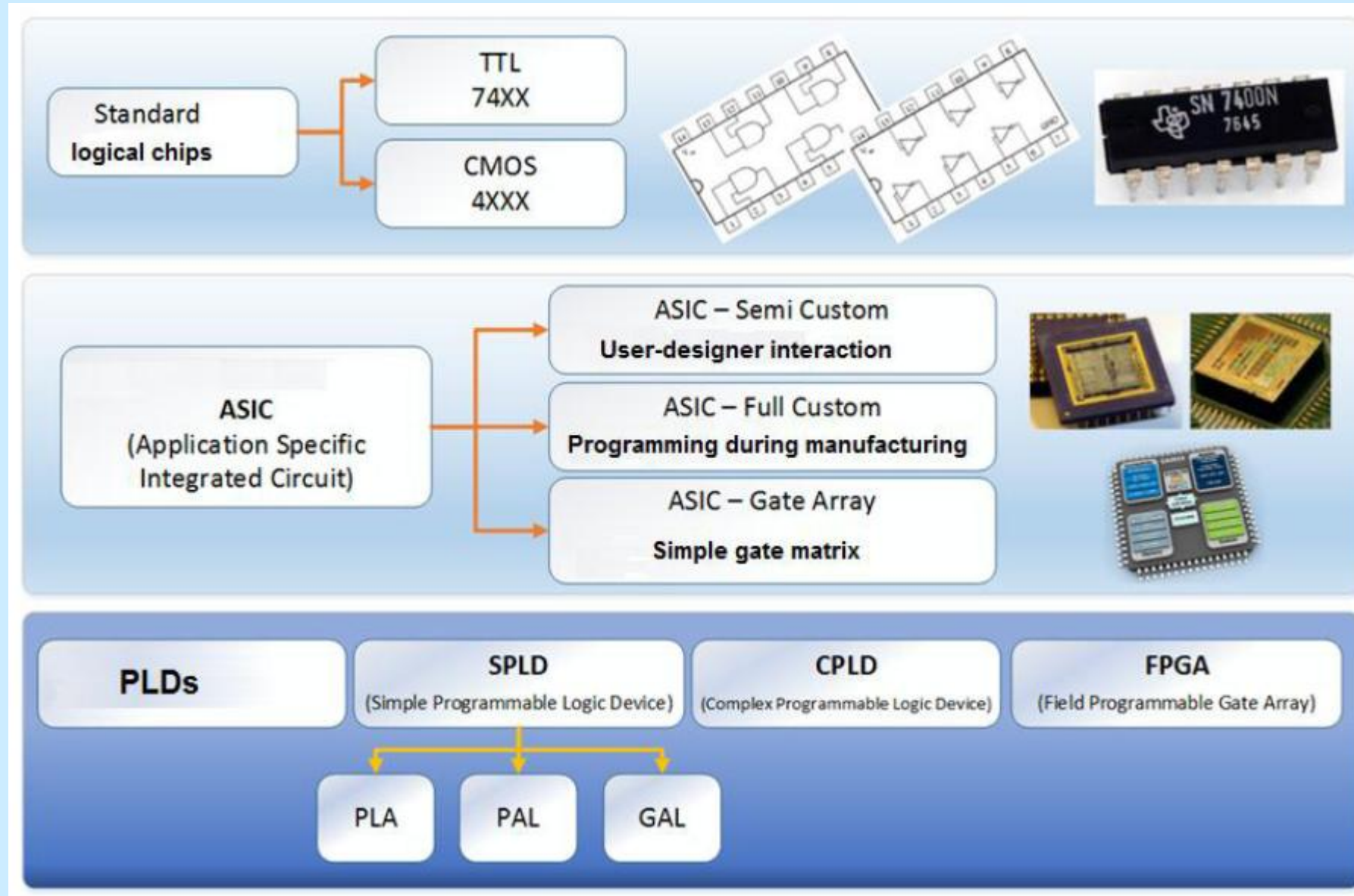
Pojęcie magistrali – bus:



Uciążliwość konstrukcyjna układów budowanych z elementów dyskretnych.

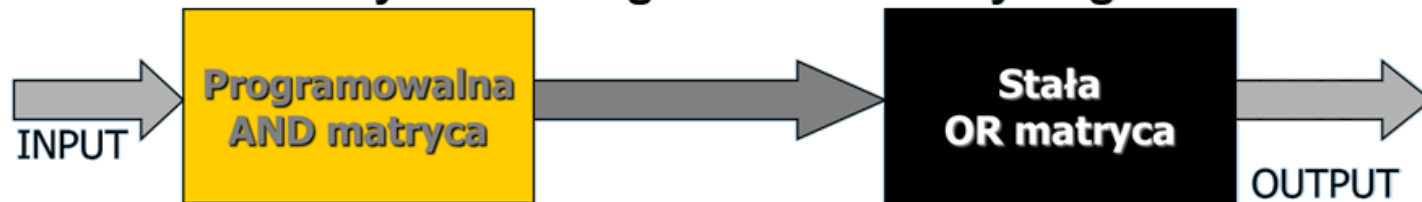


Pierwszą rodzinę układów programowalnych stanowią układy SPLD - Simple Programmable Logic Device

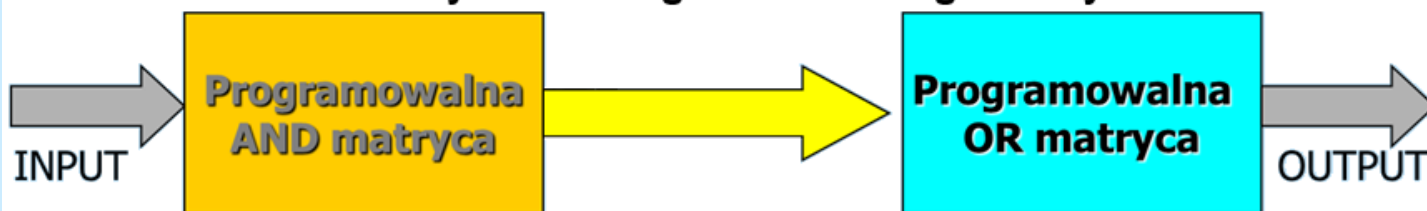


Do rodziny PLD/SPLD – zaliczane są następujące klasy układów:

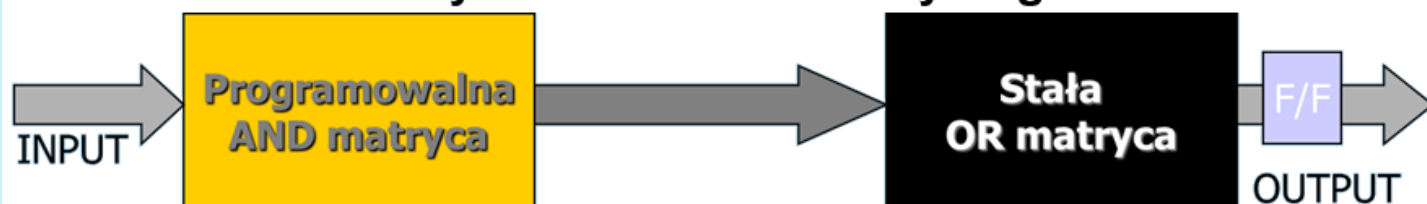
Układy PAL - Programmable Array Logic



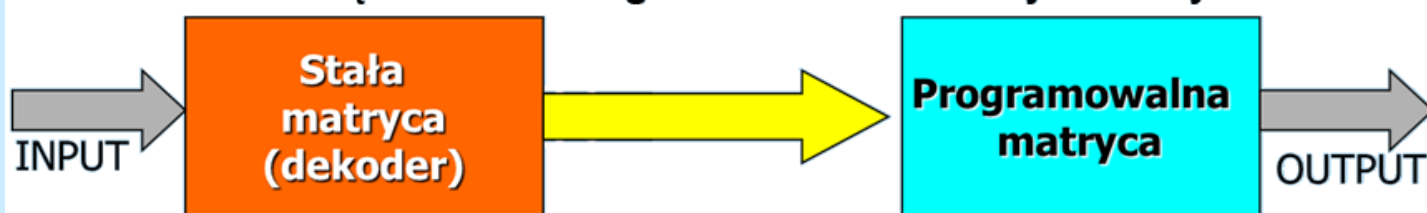
Układy PLA - Programmable Logic Array



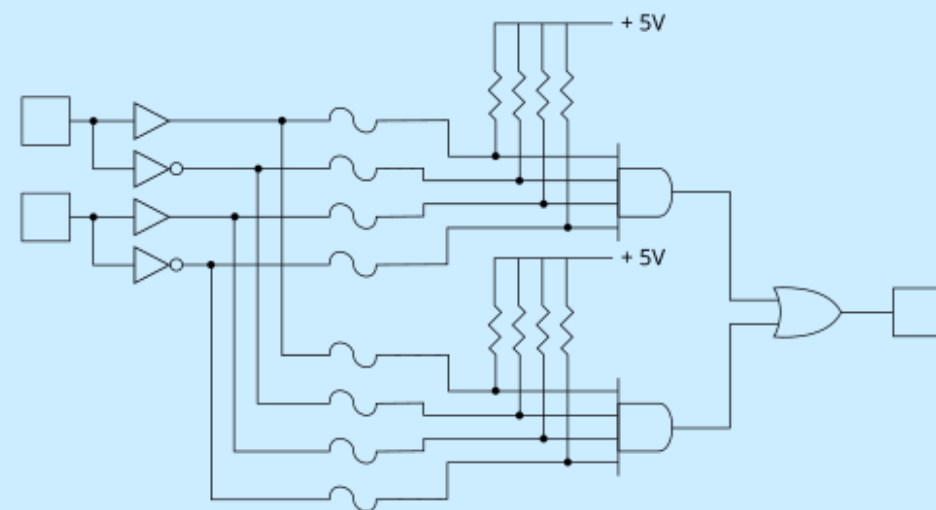
Układy GAL - Generic Array Logic



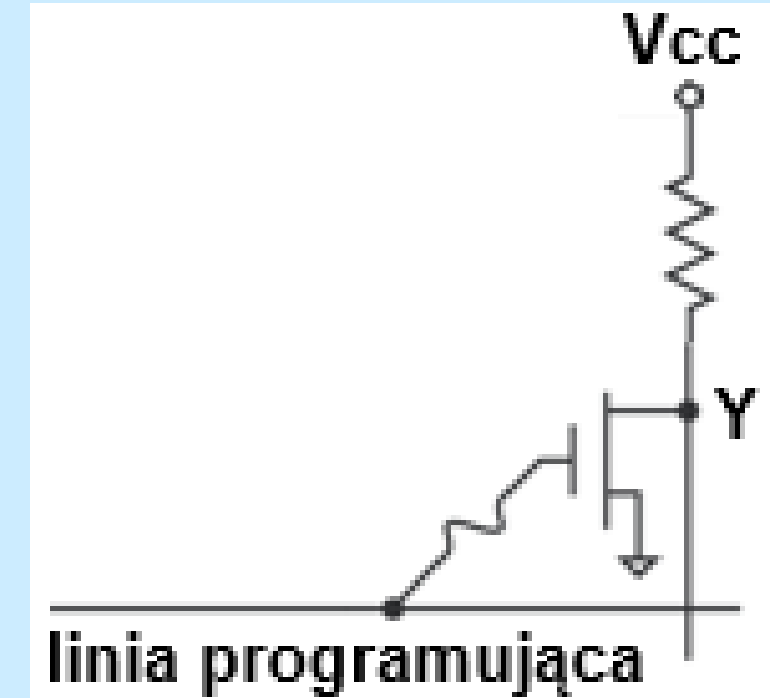
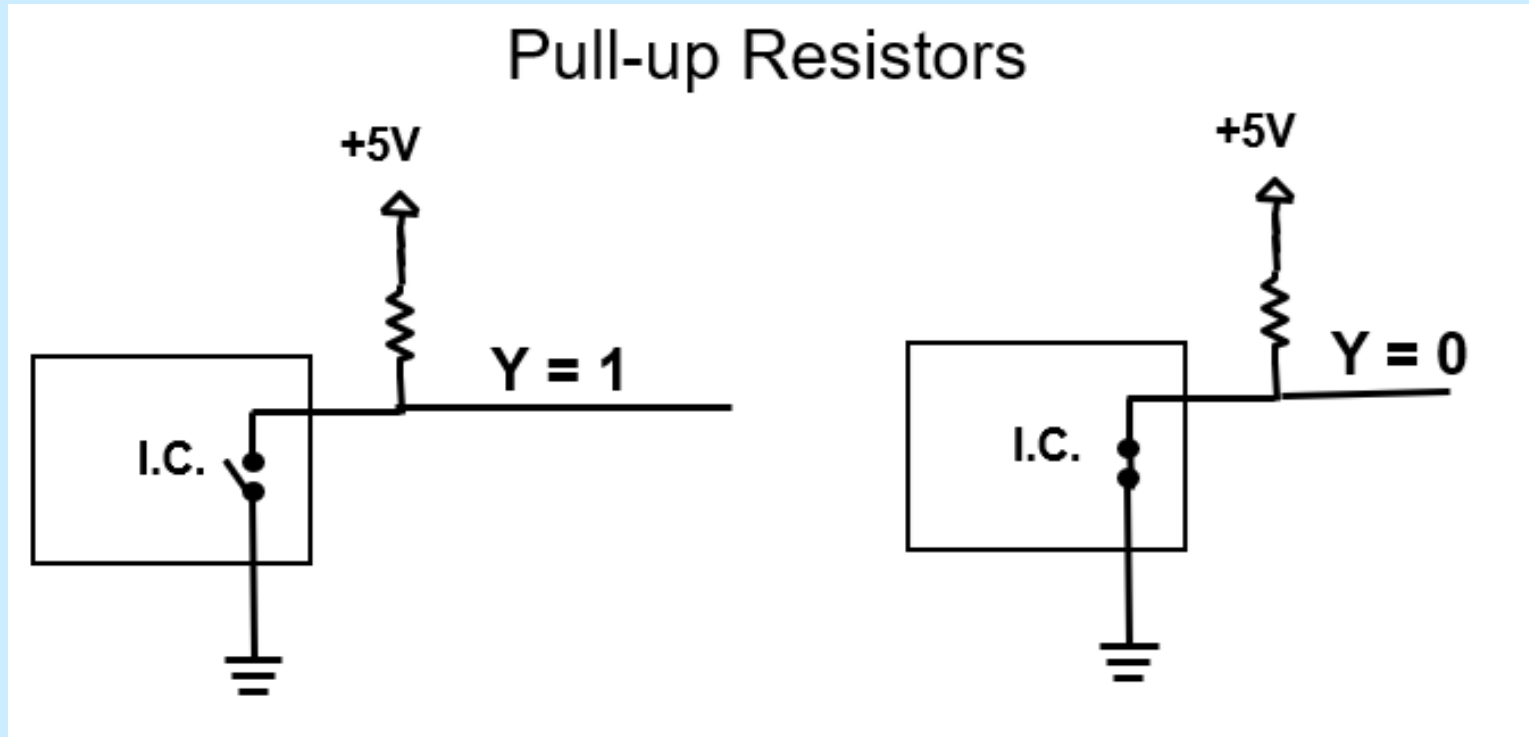
Pamięć PROM - Programmable Read-Only Memory



Koncepcja tworzenia układów programowalnych rozpoczęła się od rozwiązania problemu tworzenia przez użytkownika wewnętrznych połączeń w układzie scalonym. Mogą to być połączenia tworzone jednorazowo lub zmieniane wielokrotnie – stąd powstało pojęcie układy programowalne jednokrotnie i reprogramowalne.

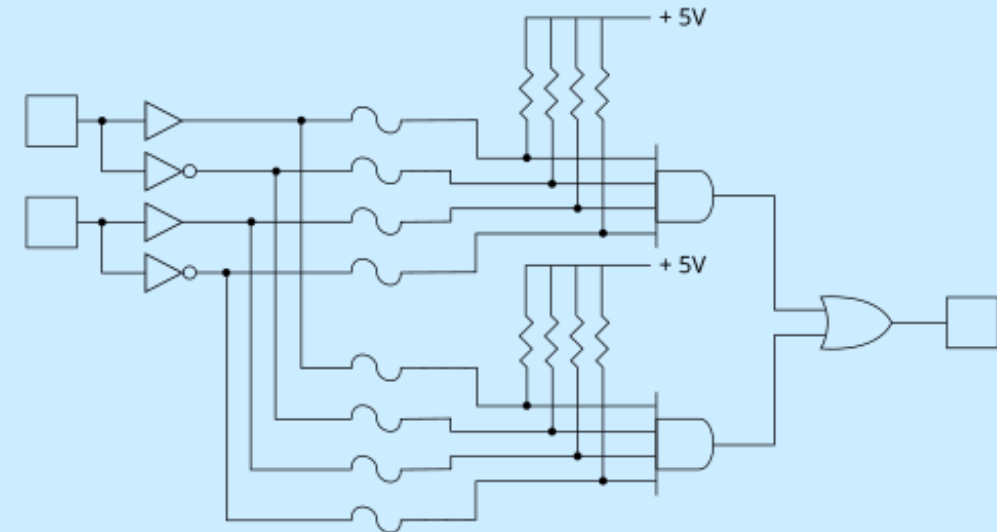
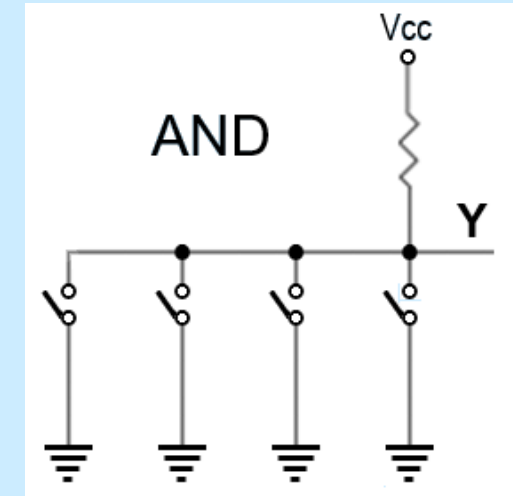
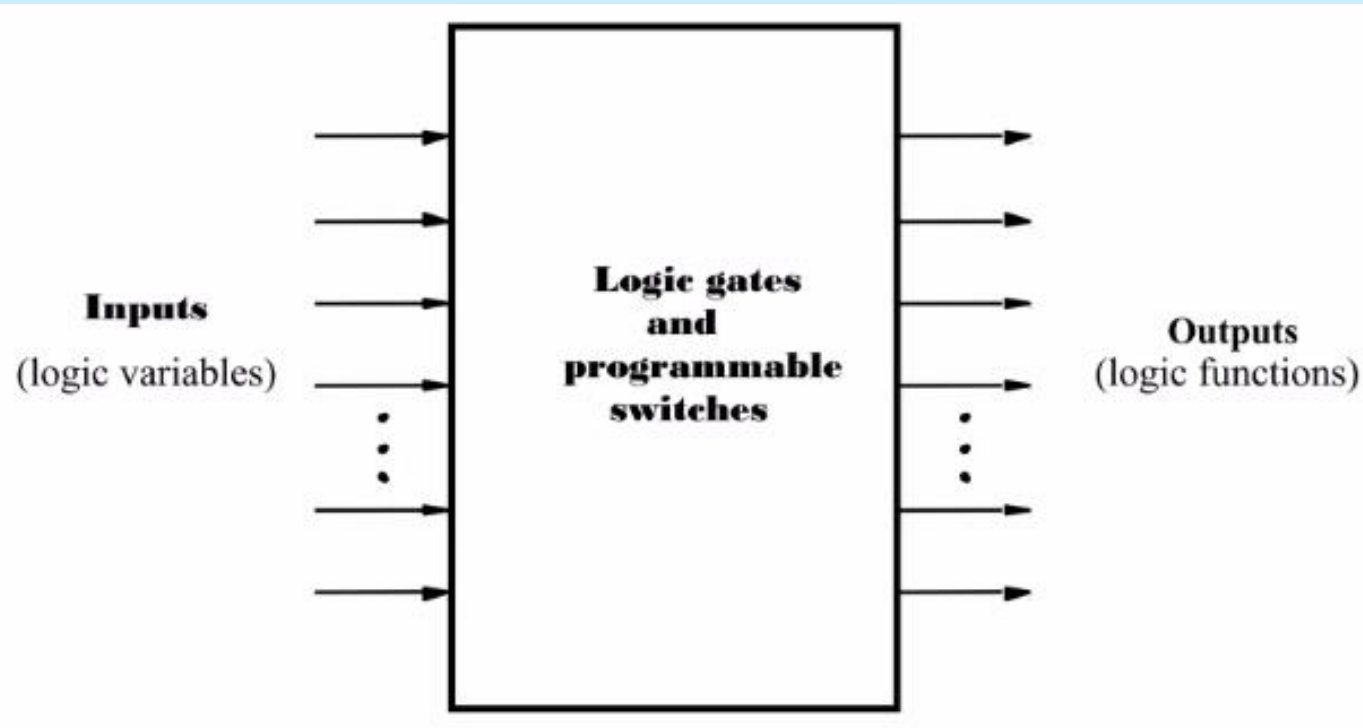


Rola rezystorów pull-up oraz kluczy tranzystorowych w układach PLD i FPGA:

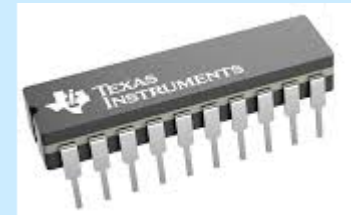
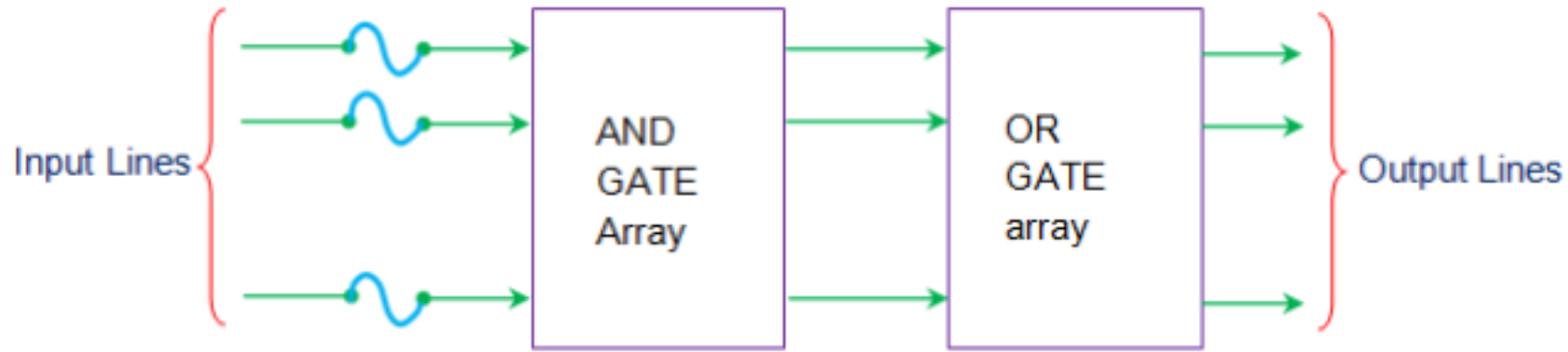


Posługując się symbolami schematu mamy pamiętać o realizacji tranzystorowej funkcyj logicznych.

Zaciskowo, z pozycji „czarnej skrzynki”, każdy układ PLD składa się z zestawu wejść i wyjść – nie wnikając w architekturę wewnętrzną.



Matryce programowalne **PAL** - Programmable Logic Array - to programowalne jednokrotnie przez użytkownika układy logiczne, które stanowią wydajny zamiennik konwencjonalnej logiki TTL. Oparte są na niskiej mocy technologii Schottky'ego, połączonej z bezpiecznikami tytanowo-wolframowymi. Ich łatwa programowalność pozwala na szybkie projektowanie funkcji niestandardowych, skutkującą kompaktową płytką drukowaną. Ich zastosowanie wymaga użycia zewnętrznego programatora oraz kompilatora języka HDL – np. ABEL. W podstawowej konfiguracji układ PAL składa się z programowalnej matrycy AND i stałej matrycy OR:



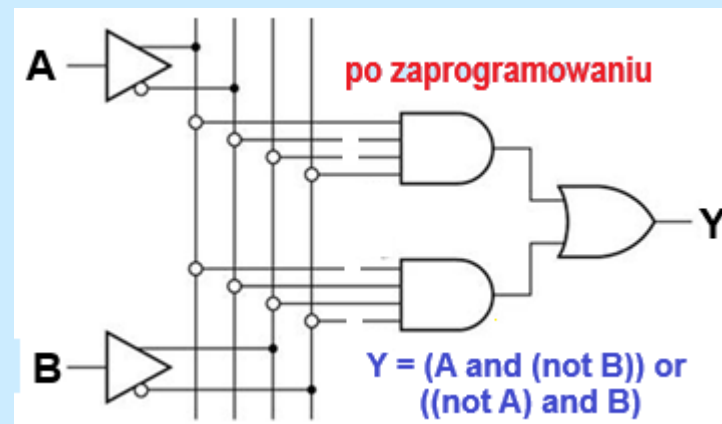
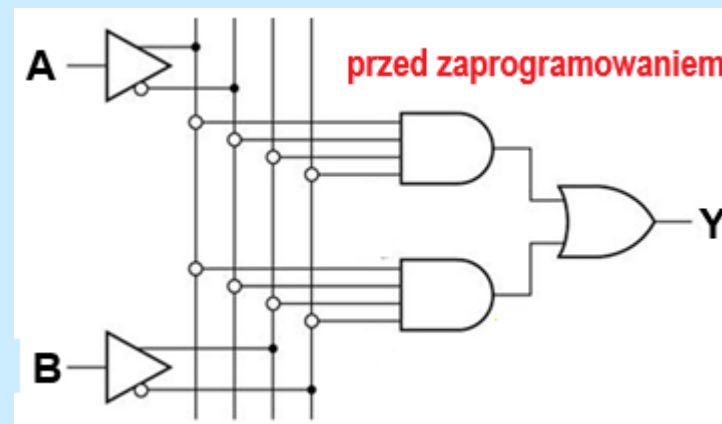
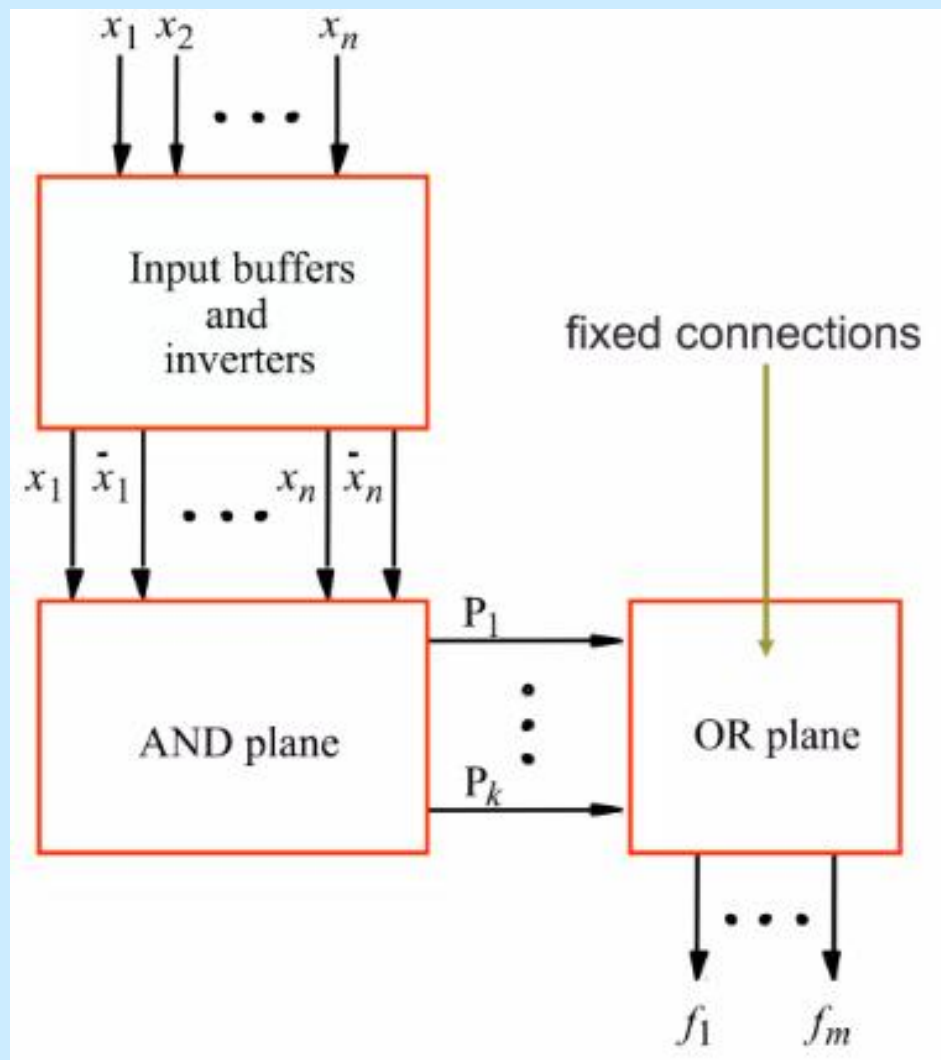
Zalety PAL:

- wysoka wydajność i niskie koszty,
- wysoki poziom bezpieczeństwa,
- niskie zapotrzebowanie na energię,
- duża elastyczność projektowania.

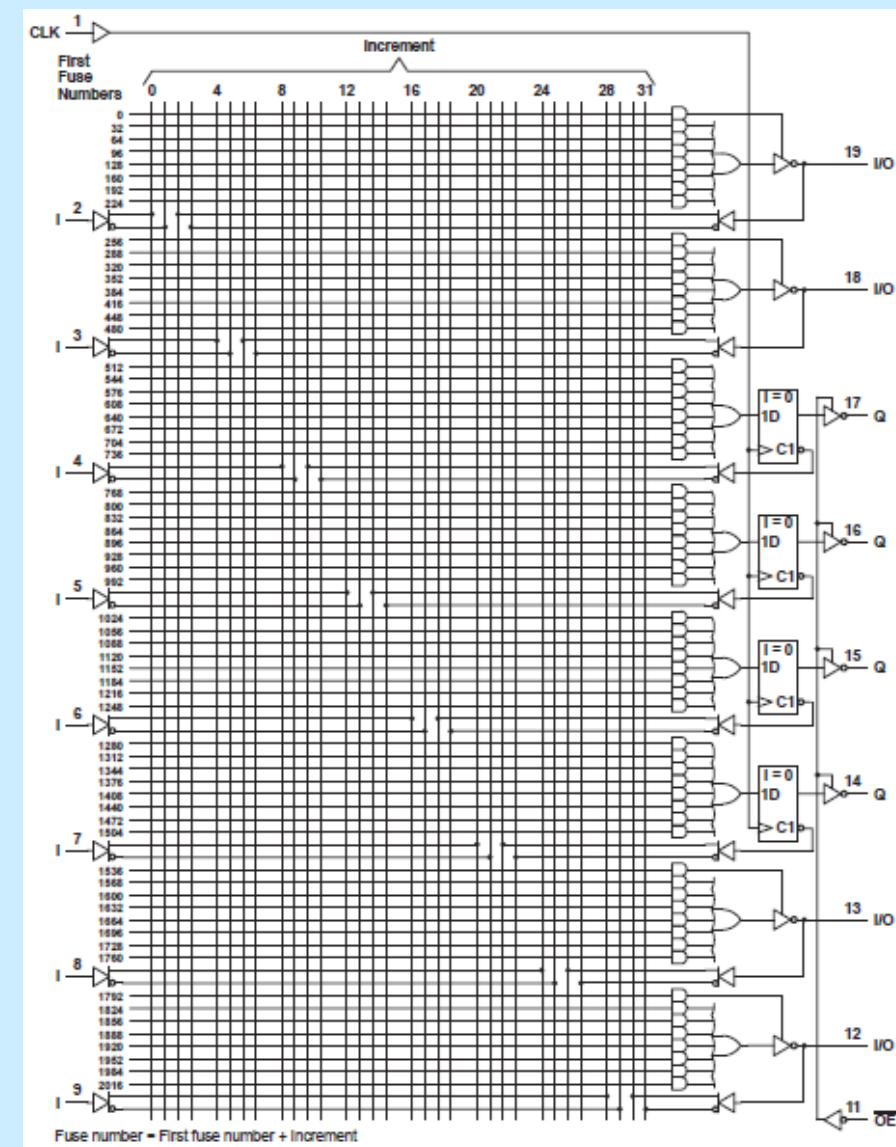
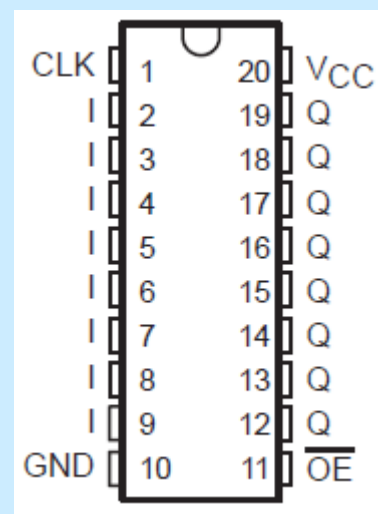
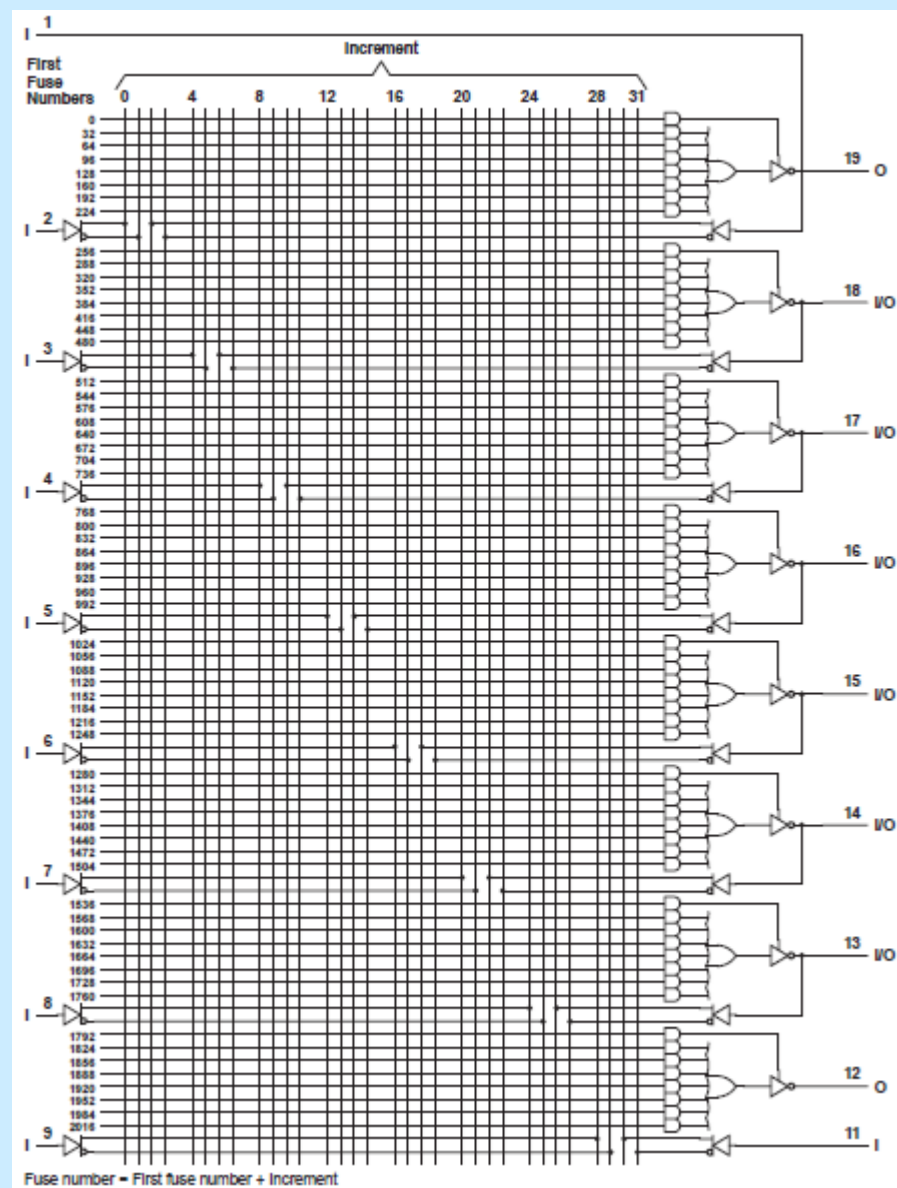
Wady PAL:

- jednokrotne programowanie,
- ograniczoną liczbą bramek/wejść,
- opóźniająca macierz OR.

W układach PAL sygnały wejściowe podawane są przez negatory wejściowe, dzięki czemu na wejście każdej bramki AND można doprowadzić dowolny sygnał wejściowy w postaci prostej i zanegowanej. W czasie programowania, podnosząc napięcie na wybranych liniach matrycy programowalnej – przerywa się połączenia.



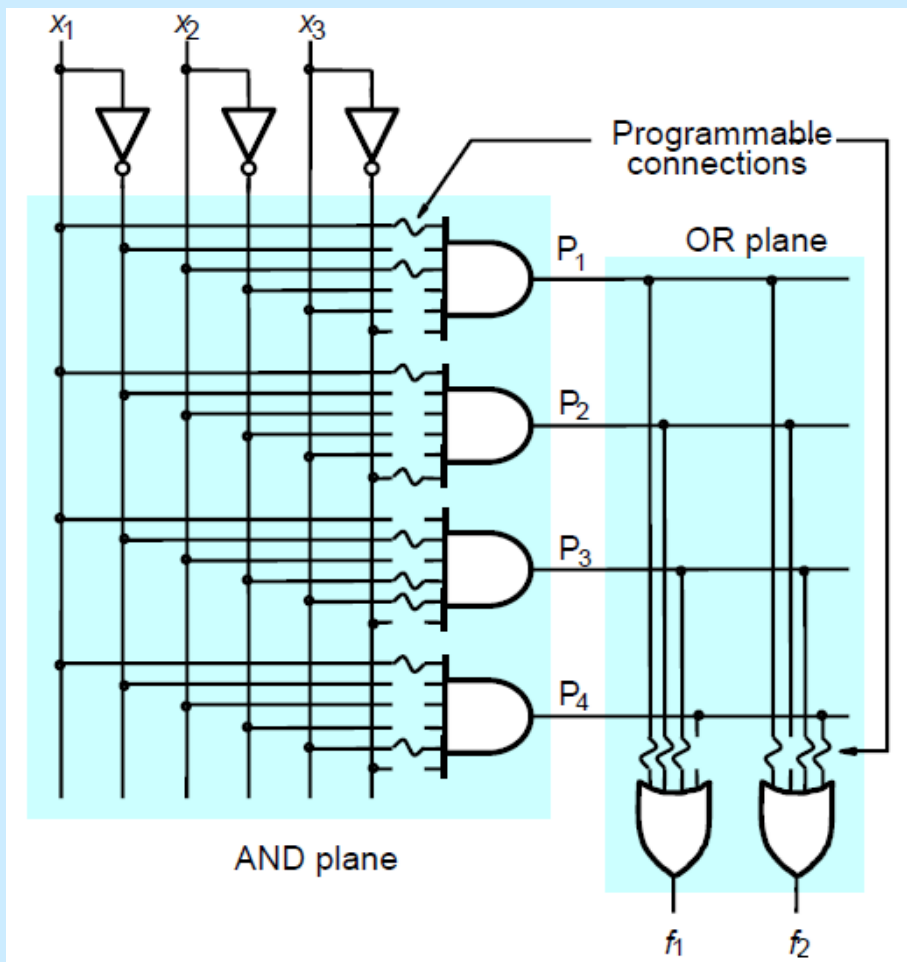
Przykładowe matryce PAL Texas Instruments – w wersji kombinacyjnej (16L8AN) i sekwencyjnej (16R4AM):



Programowalne matryce logiczne - **PLA** (Programmable Logic Array) – jest układem służącym do realizacji kombinacyjnych obwodów logicznych, podobnie jak PAL. Historycznie PLA była pierwszym układem PLD, poprzedzającym PAL. Układ PLD posiada dwa matryce programowalnych na wejściach bramek – programowalną matrycę bramek AND oraz programowalną matrycę bramek OR. Na wejściach bramek matryc znajdują się jednokrotnie tworzone/przerywane połączenia w postaci bezpieczników (fuse) lub anty-bezpieczników (antifuse). Bezpiecznik jest elementem o niskiej rezystancji, którego połączenie można przepalić aby uzyskać obwód rozwarty. W anty-bezpiecznikach w czasie programowania powstaje połączenie o niskiej rezystancji. Stosowanie PLA wymaga użycia zewnętrznego programatora i kompilatora PLD.



Realizacja przykładowych funkcji logicznych z użyciem układu PLA:



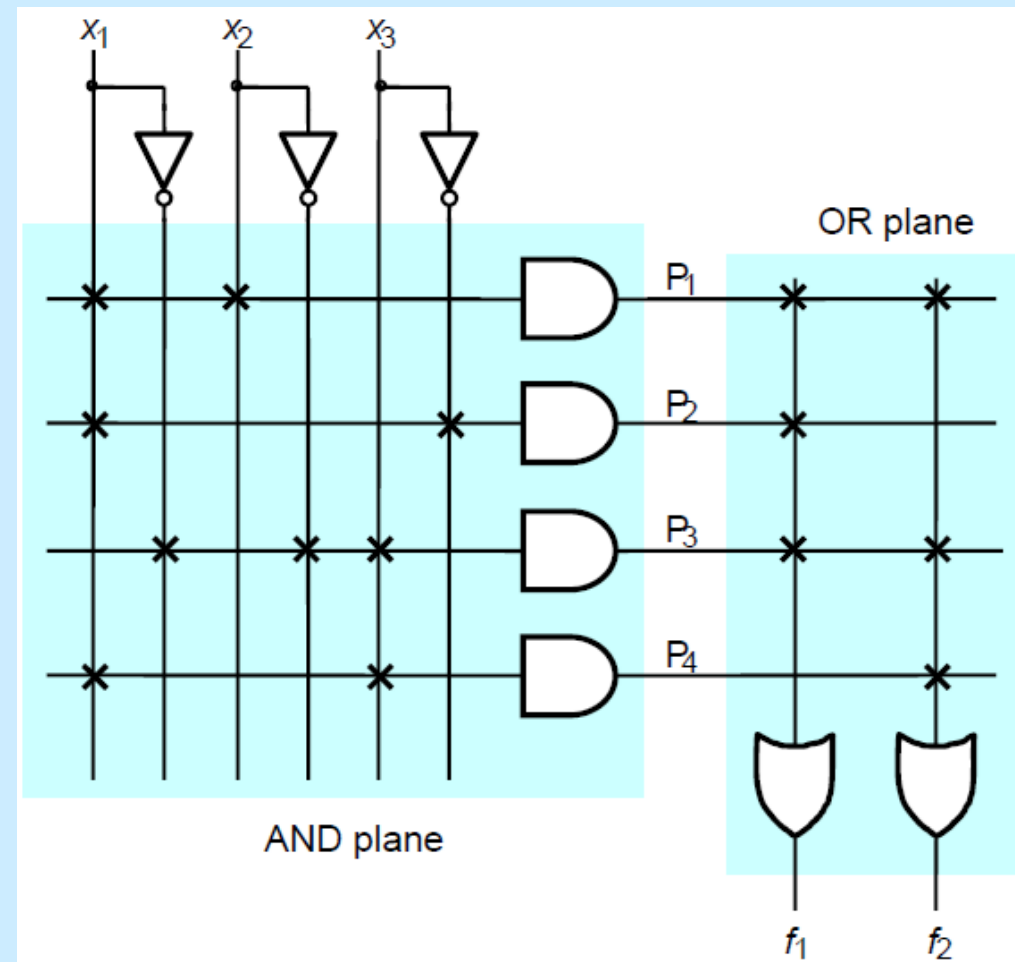
$$P_1 = x_1 x_2$$

$$P_2 = x_1 \bar{x}_3$$

$$P_3 = \bar{x}_1 \bar{x}_2 x_3$$

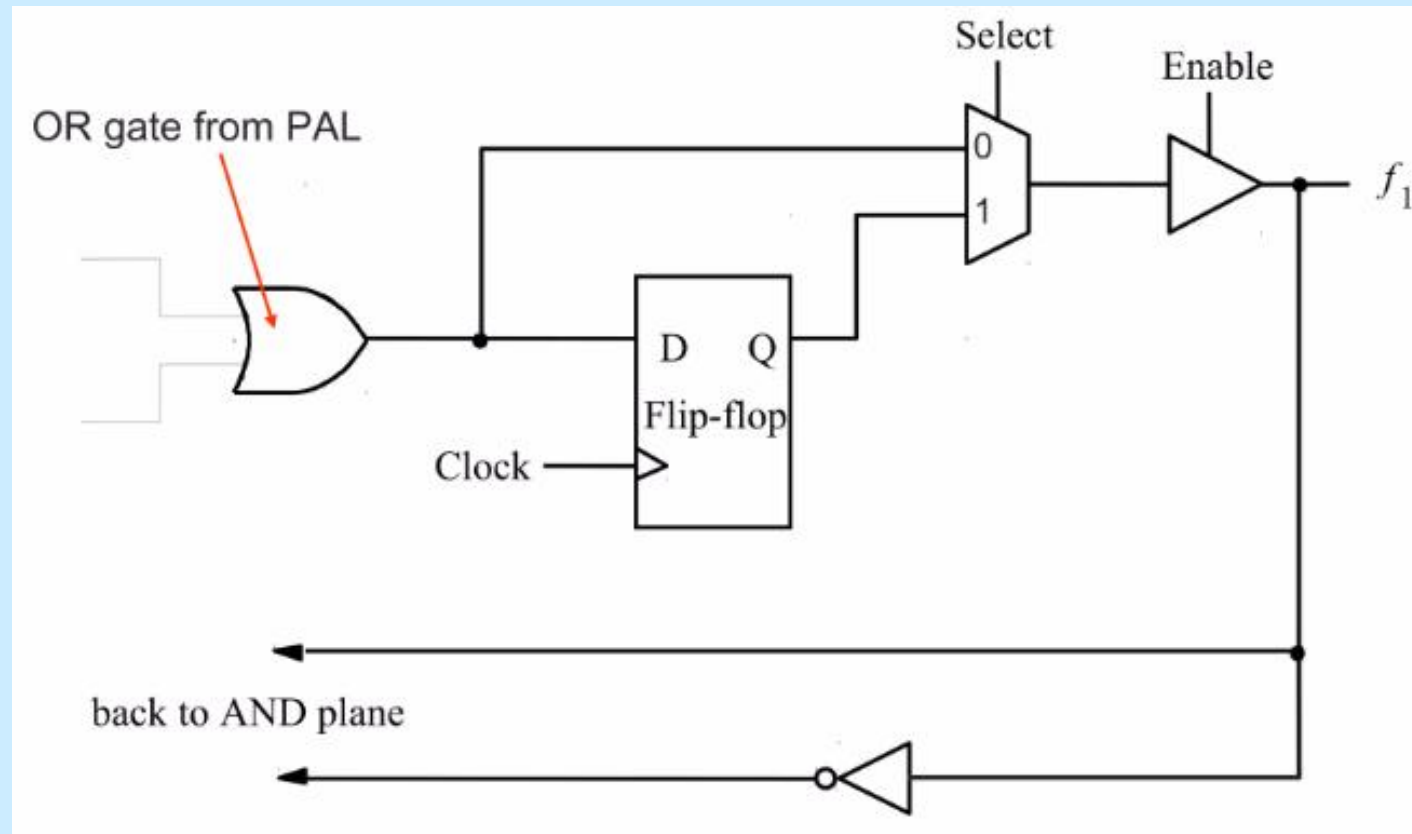
$$P_4 = x_1 x_3$$

$$f_1 = x_1 x_2 + x_1 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3$$

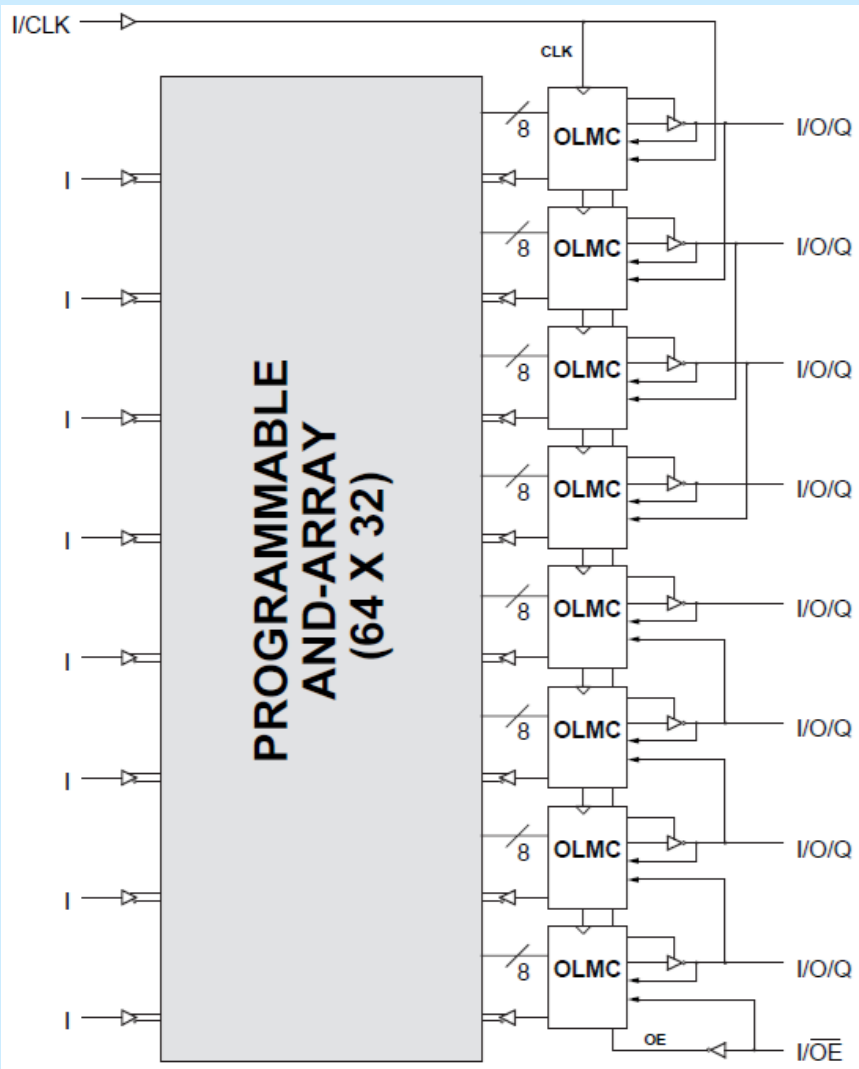


$$f_2 = x_1 x_2 + x_1 \bar{x}_2 \bar{x}_3 + x_1 x_3$$

Po dołączeniu na wyjściu bramki OR w układzie PAL dodatkowych elementów: przerzutnika D, multipleksera dwuwęściowego, bramki trójstanowej i dwóch linii sprzężenia zwrotnego powstała makrokomórka OLMC - Output Logic Macro Cell. Makrokomórki logiczne stały się podstawą architektury współczesnych układów programowalnych, konfiguracja połączeń jest podtrzymywana zawartością komórek EEPROM a nie fuse.



Po połączeniu struktury PAL z zespołami makrokomórek na wyjściu powstał układ GAL - Generic Array Logic. Był to pierwszy układ PLD **reprogramowalny**, tzn. układ scalony, który można wielokrotnie przeprogramować elektrycznie po uprzednim skasowaniu poprzedniej zawartości. GAL16V8 firmy Lattice pokazano na rysunku.



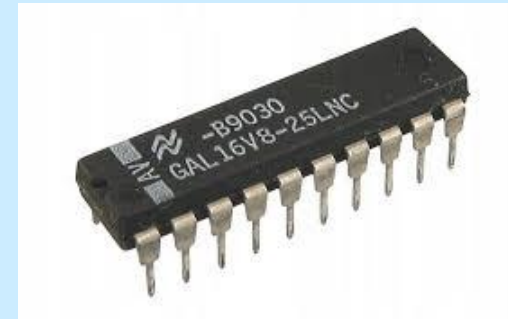
Zalety układów GAL:

- możliwość wielokrotnego programowania – około 100 cykli,
- możliwość łączenia wyjścia OLMC z dowolnym wejściem,
- możliwość syntezy układów sekwencyjnych.

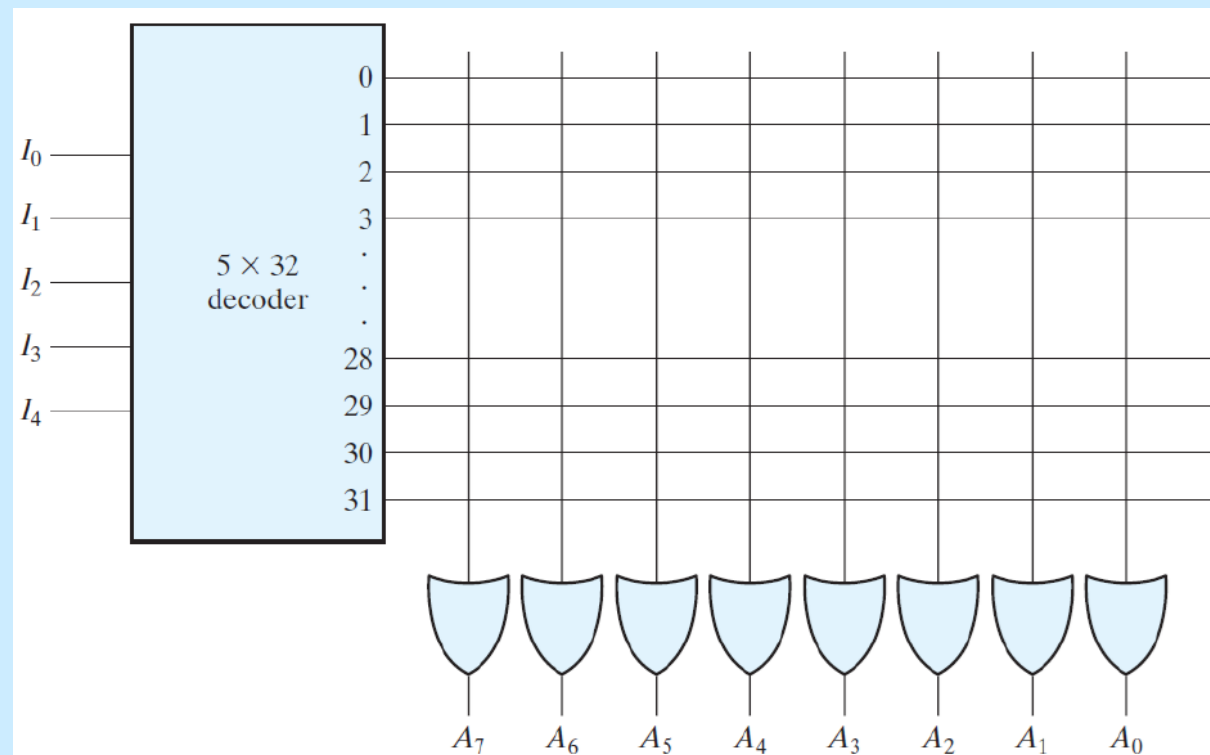
Wady:

- układ wymaga użycia programatora zewnętrznego,
- zasilanie 5V.

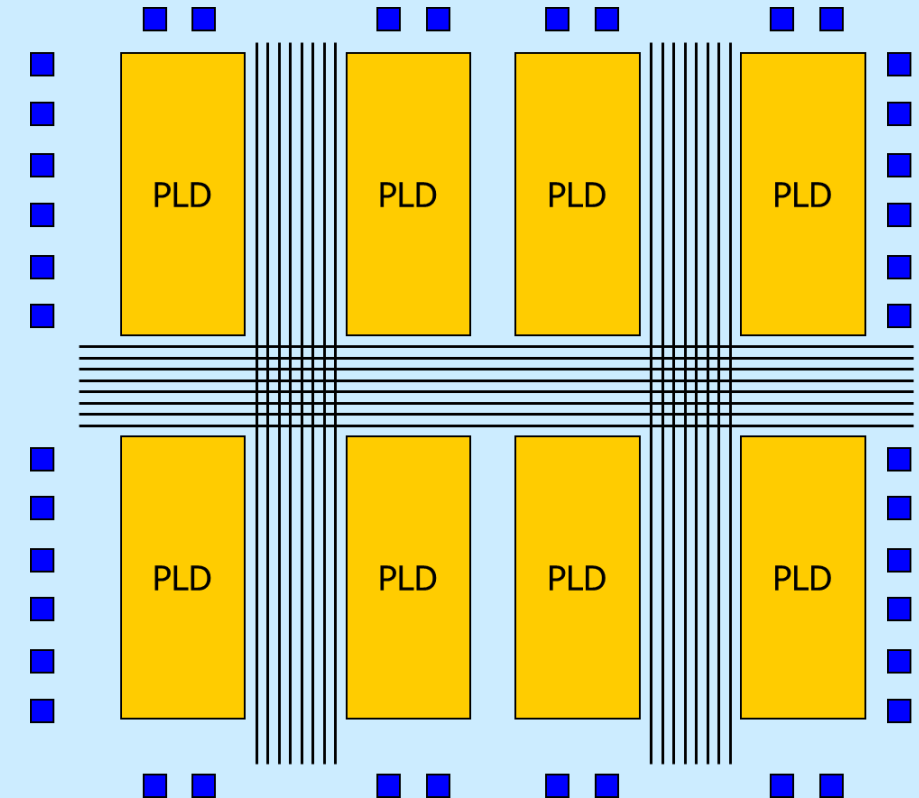
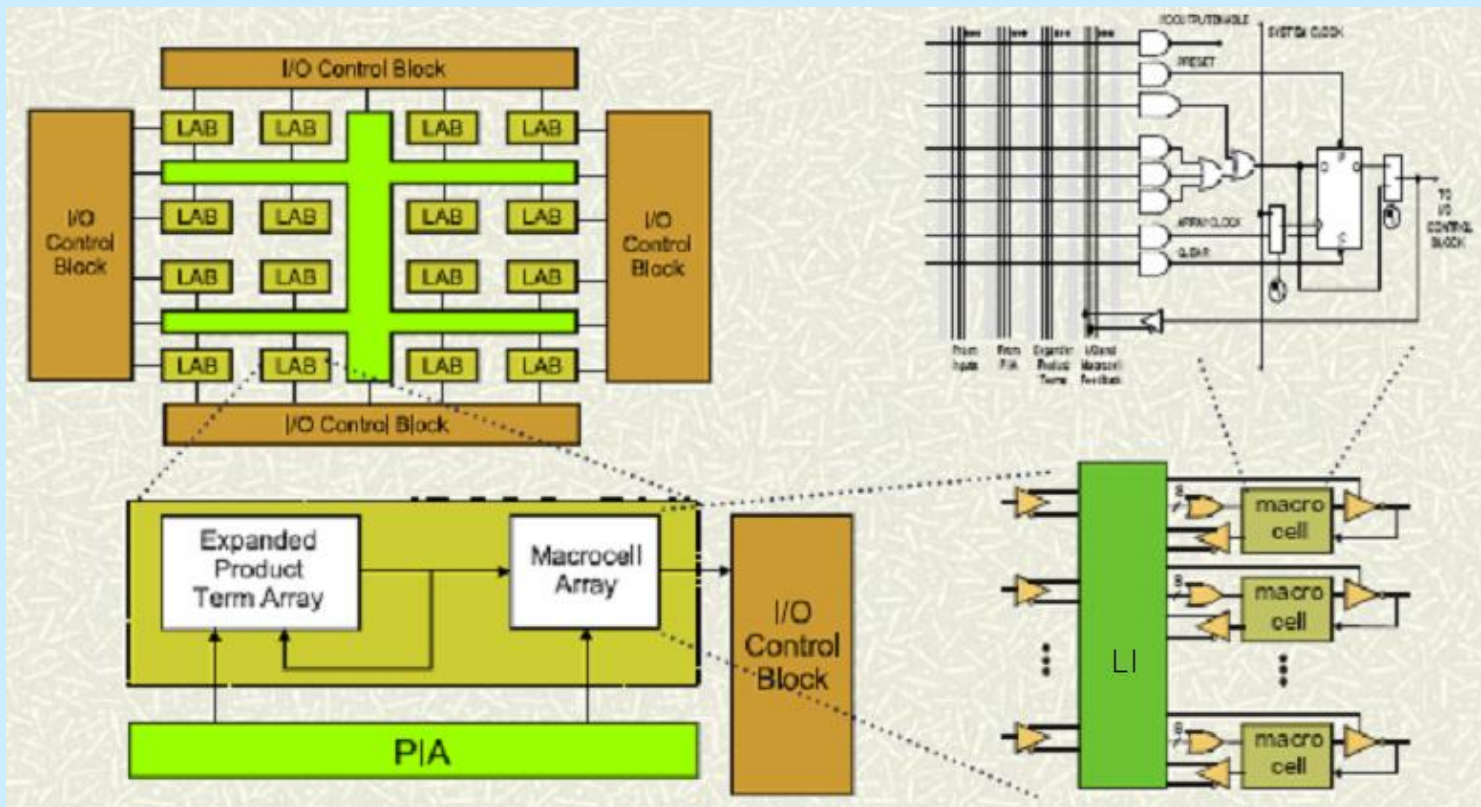
Kompilatory: WinCupl, ABEL, OrCAD-PLD, PLDesigner, PALasm.



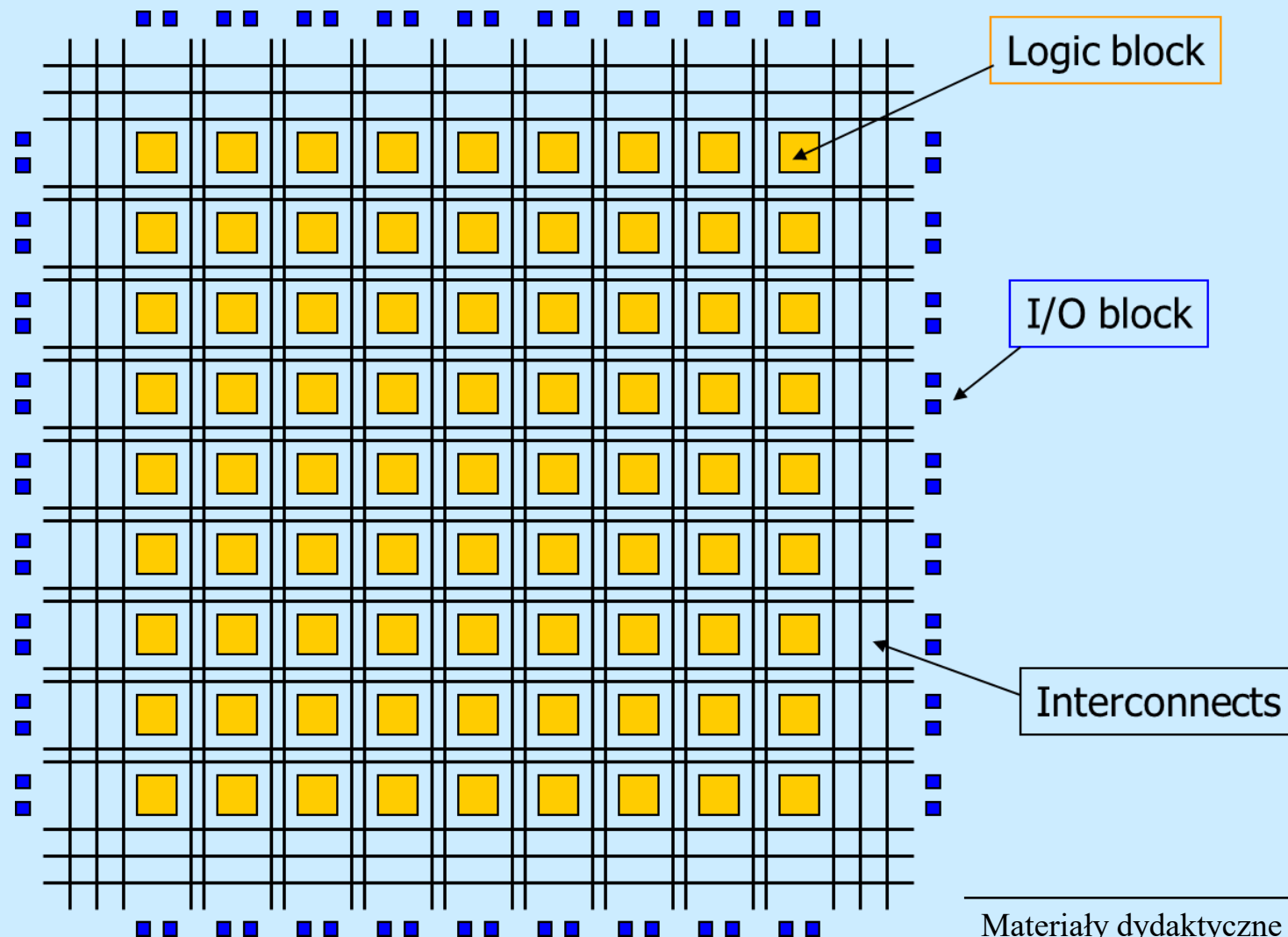
W przypadku krótkich partii produktów bardziej ekonomiczne może być użycie pamięci PROM (Programmable Read-Only Memory). Fabrycznie nowe PROM zawierają bezpieczniki w stanie nienaruszonym, co ustawia bity zapisanych słów na 1. Bezpieczniki PROM są przepalane poprzez podanie impulsów wysokiego napięcia do układu. Przepalony fuse ustawia stan 0, a nienaruszony daje stan 1. Pozwala to użytkownikowi zaprogramować PROM samodzielnie z użyciem programatora. Odczyt z PROM odróżnieniu od PAL/PLA odbywa się całymi słowami, które wskazuje adres wejściowy. Procedura programowania PROM jest nieodwracalna.



Przeniesienie matrycy bramki AND do struktury makrokomórki, połączenie kilku/kilkunastu makrokomórek w oraz wprowadzenie lokalnych połączeń LI (Local Interconnect), utworzyło blok LAB (Logic Array Block). Połączenie bloków LAB/PLD za pomocą programowalnej matrycy logicznej PIA (Programmable Interconnect Array) utworzyło układ CPLD (Complex PLD). Charakterystycznymi cechami układów CPLD są: **stała długość połączeń PIA**, **nieulotność konfiguracji** oraz **programowalność w układzie** (In System Programming).



Układy CPLD rozbudowano zwiększając liczbę makrokomórek, zrezygnowano ze stałych połączeń PIA na rzecz połączeń segmentowych różnej długości, zrezygnowano z nieulotności konfiguracji, dodano inne zasoby architektury – co doprowadziło do opracowania źródłowej wersji FPGA – Field Programmable Gate Array.



Zaliczenie częściowe wykładu nr 6 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

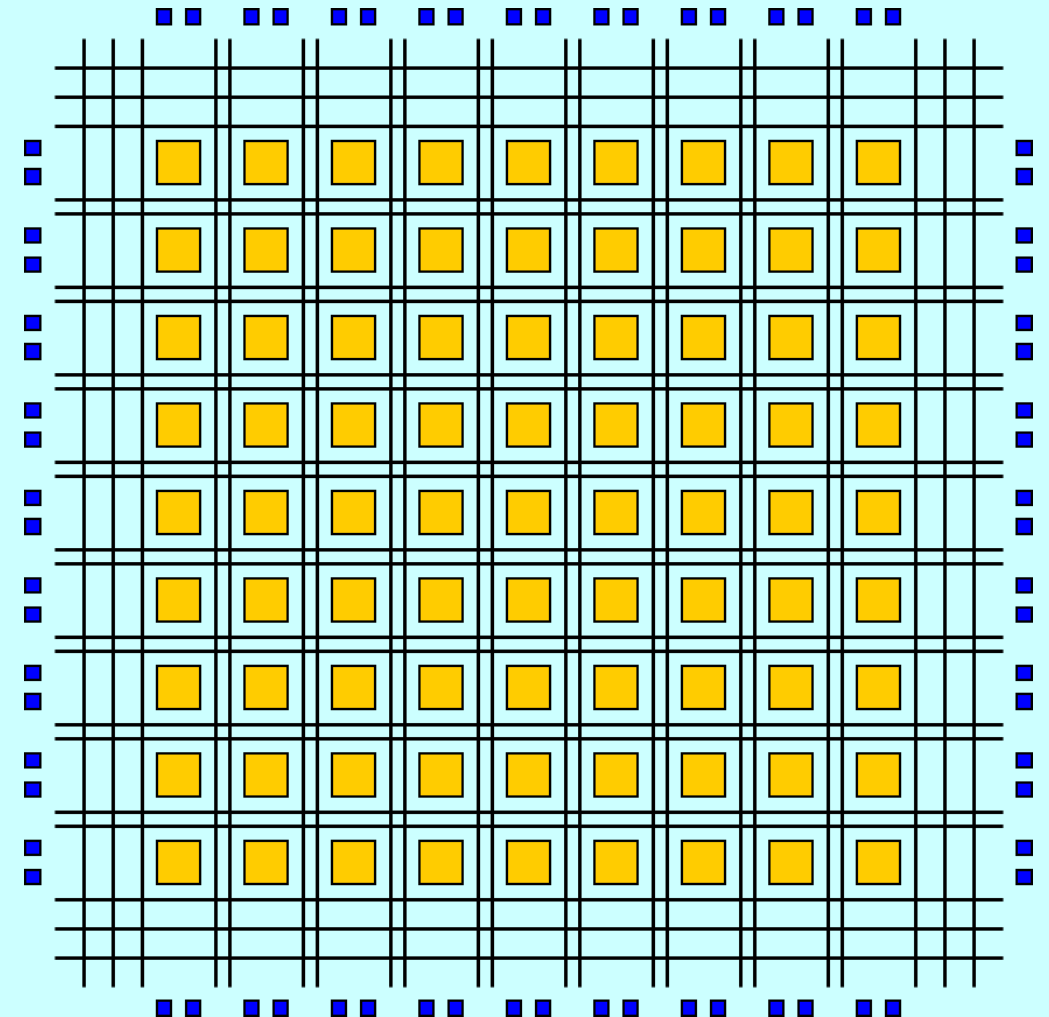
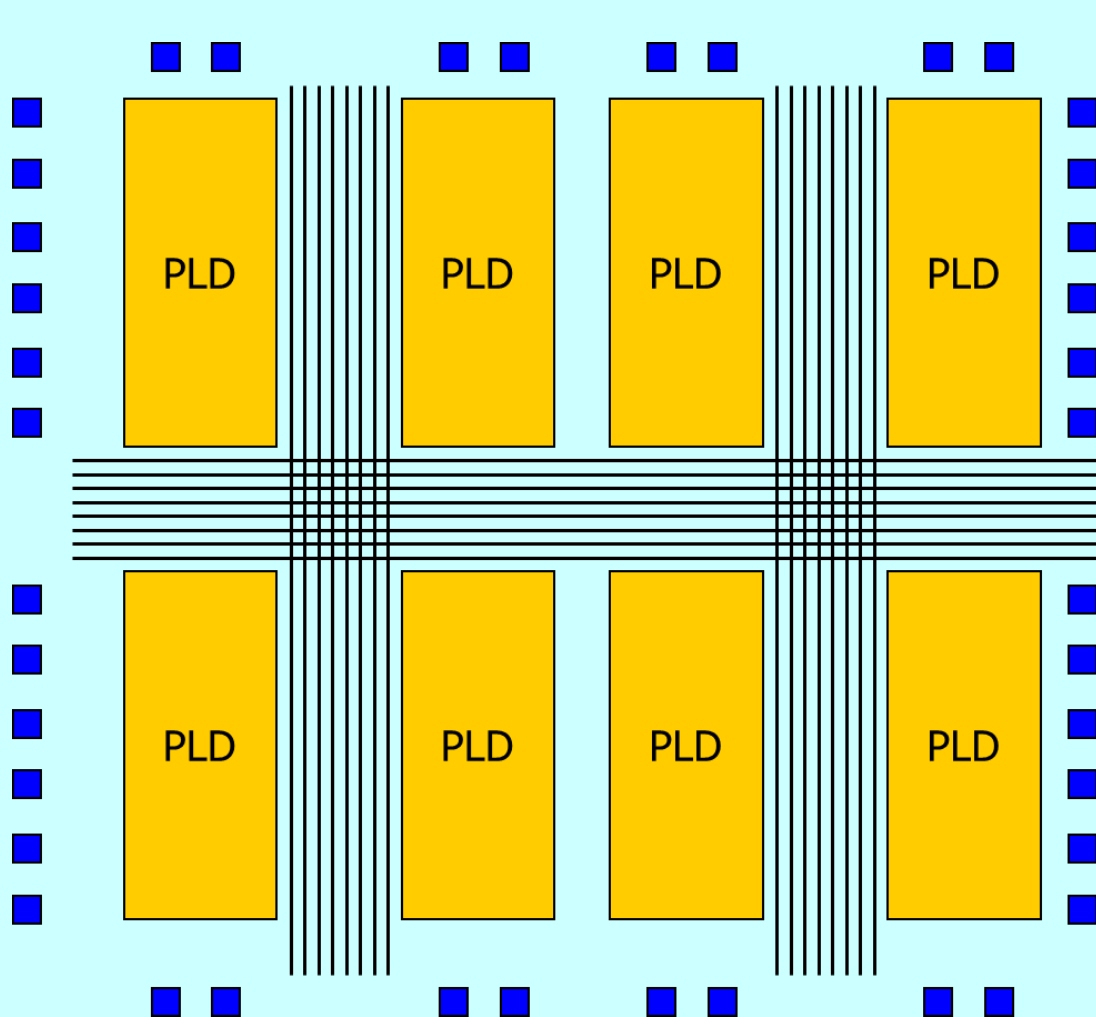
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

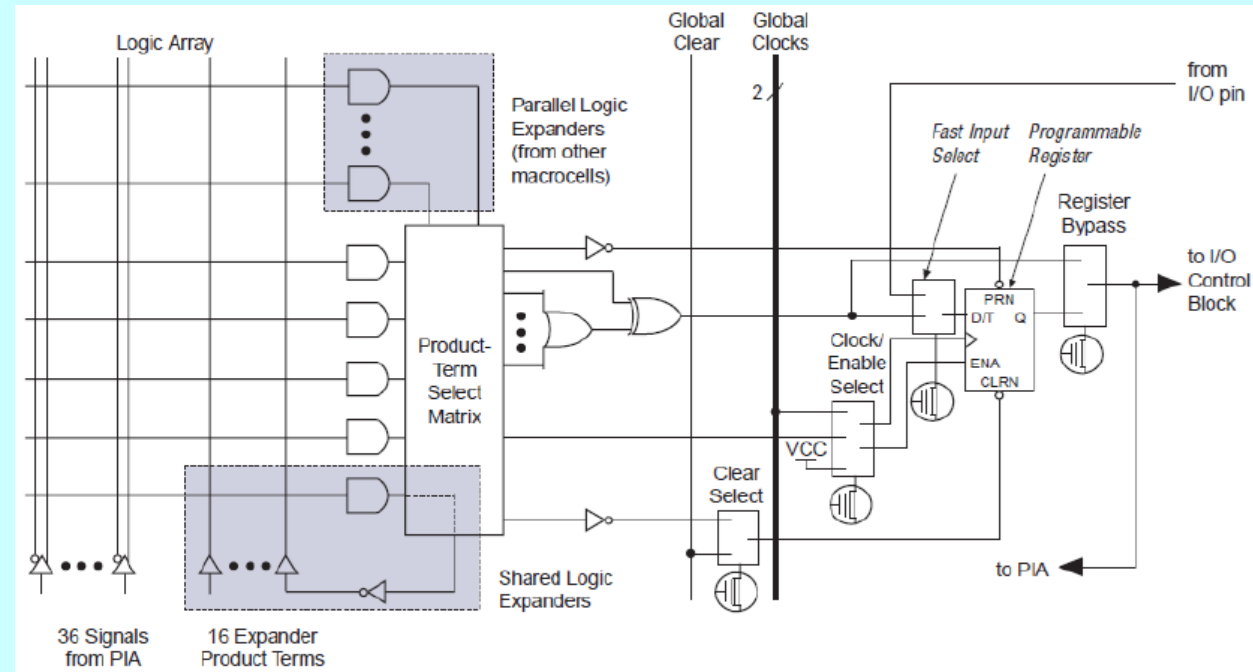
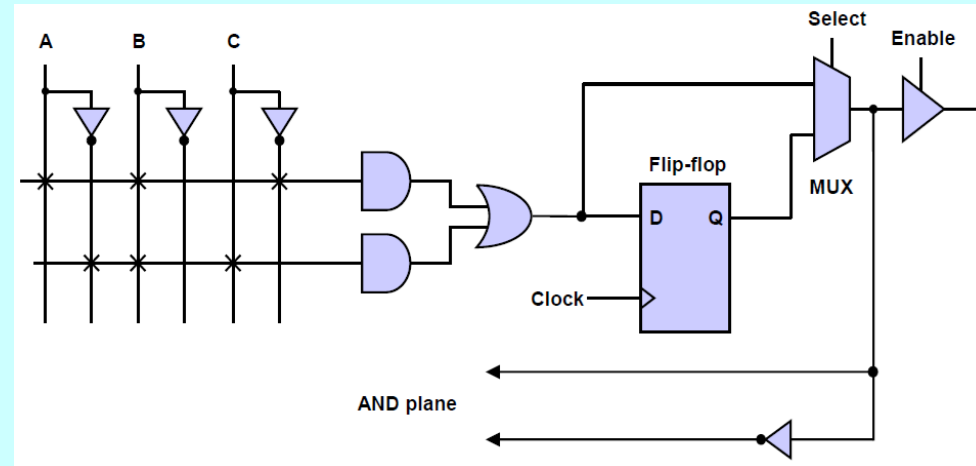
Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

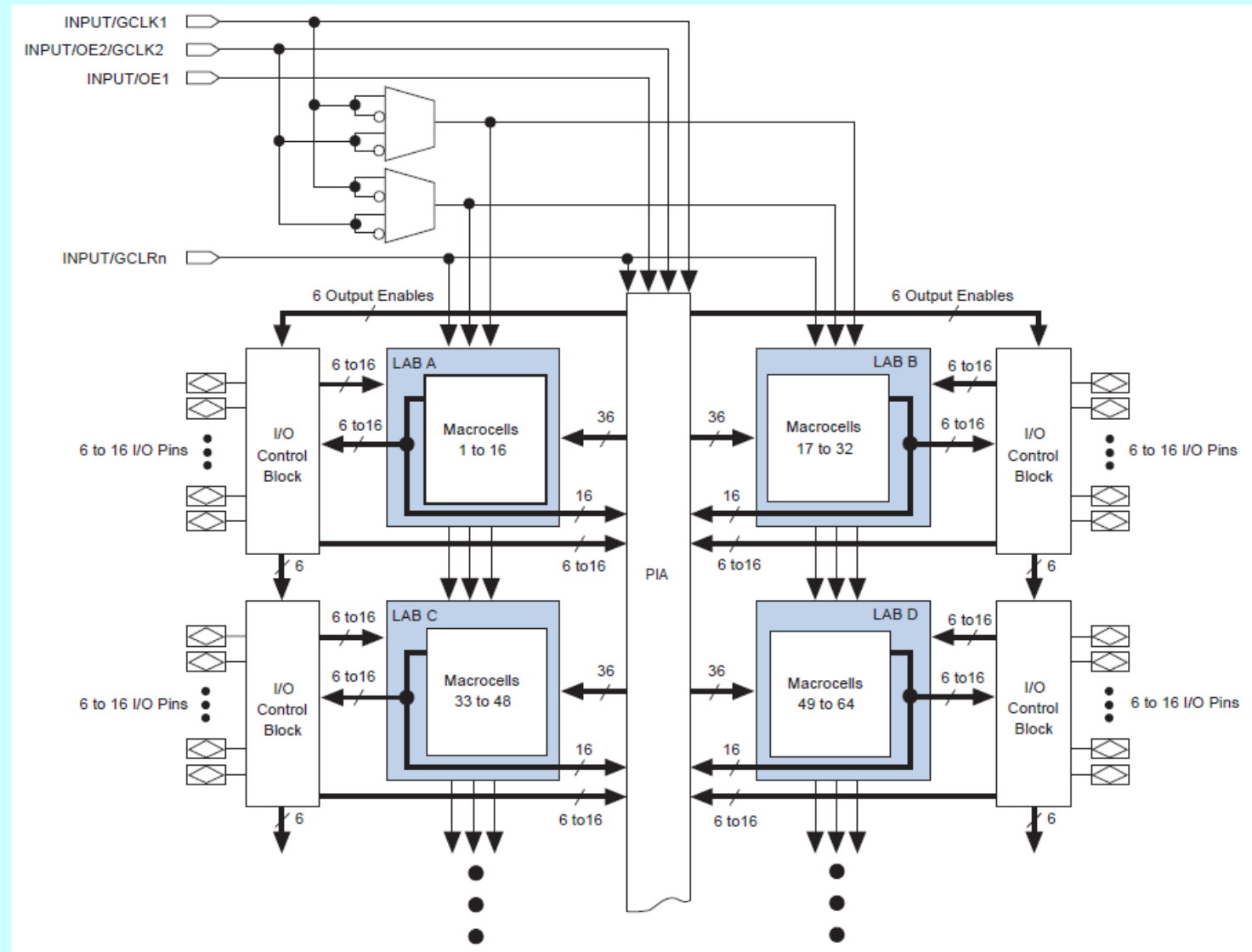
W poprzednim wykładzie przedstawiony był ogólny zarys koncepcji układów programowalnych CPLD i FPGA



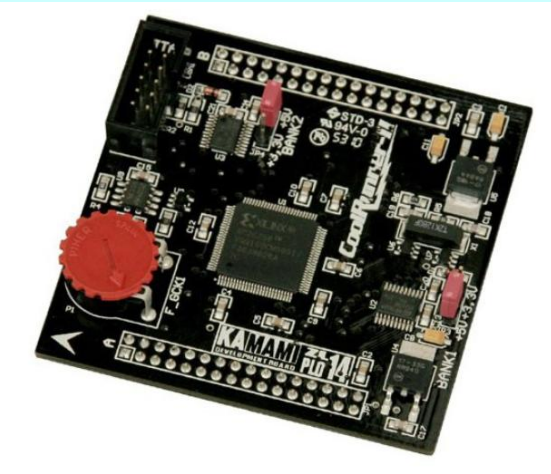
W rzeczywistości makrokomórka układu CPLD jest bardziej złożona w stosunku do OLMC z układów GAL:



Natomiast kompletny układ CPLD może posiadać architekturę podobną do poniższej:



Nie zawsze użytkownik musi montować układu CPLD w skomplikowanej obudowie – dostępne są moduły np.:

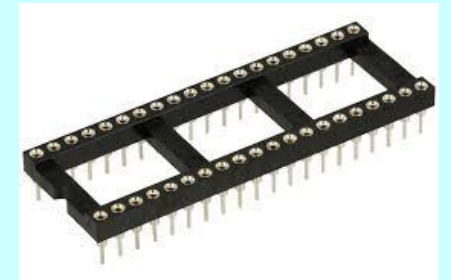


Układ XC2C256 (256 makrokomórek) z rodziny CoolRunner-II w obudowie VQFP100:

- wbudowany generator sygnału 32,768 kHz
- 10-pinowe gniazdo JTAG (ZL11PRG),
- sygnały wyprowadzone na złącza szpilkowe,
- zasilanie modułu 5 V DC,
- **160 zł.**



Układ XC2C64 (64 makrokomórki) z rodziny CoolRunner-II w obudowie PQFP40: przystosowany do podstawki DIP 40, **90 zł.**

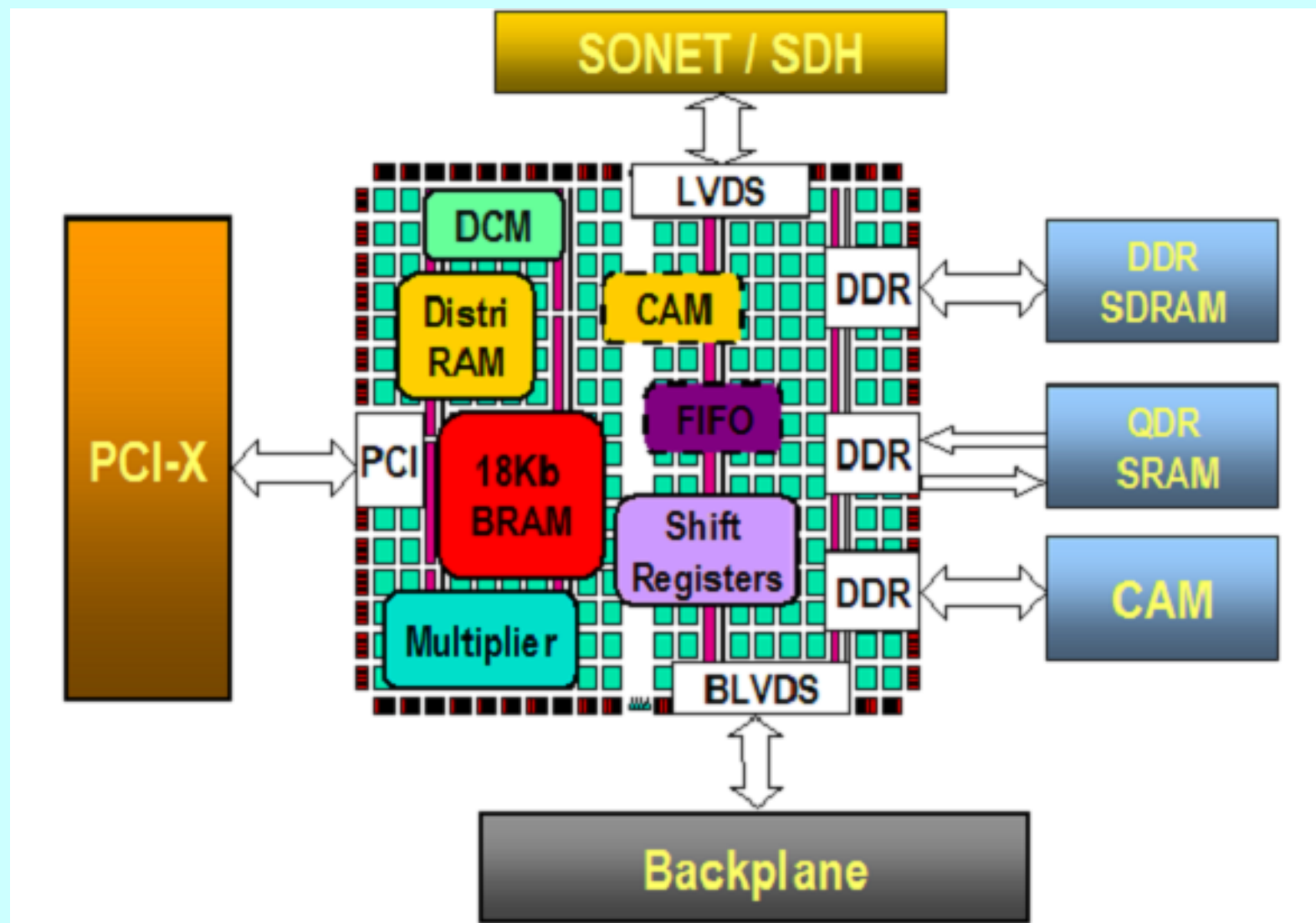
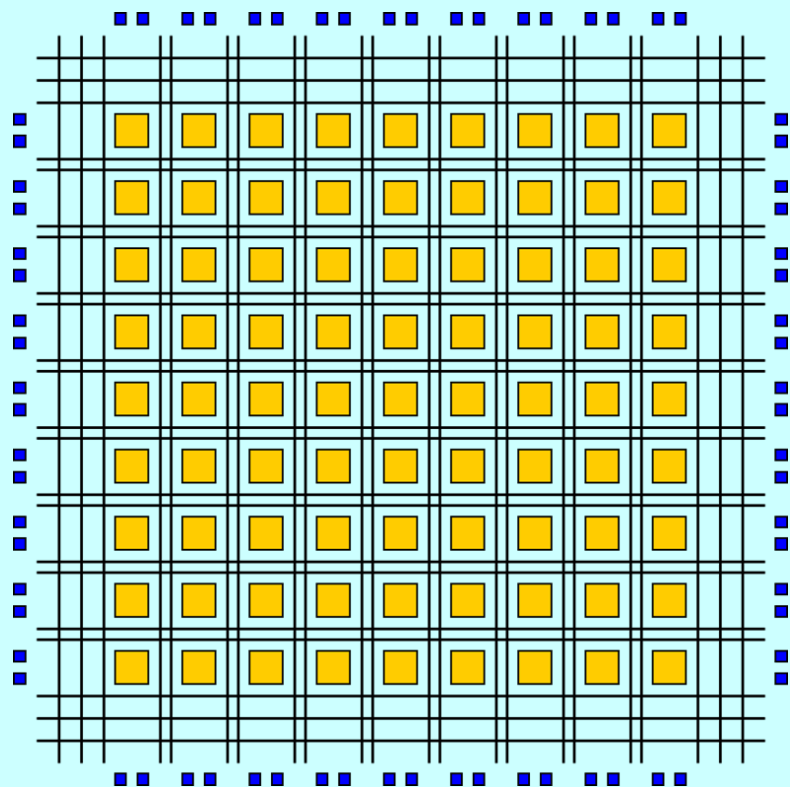


Max10 (4000 makrokomórek) Intel
przetwornik ADC 12 bit,
20kByte SRAM,
\$55.

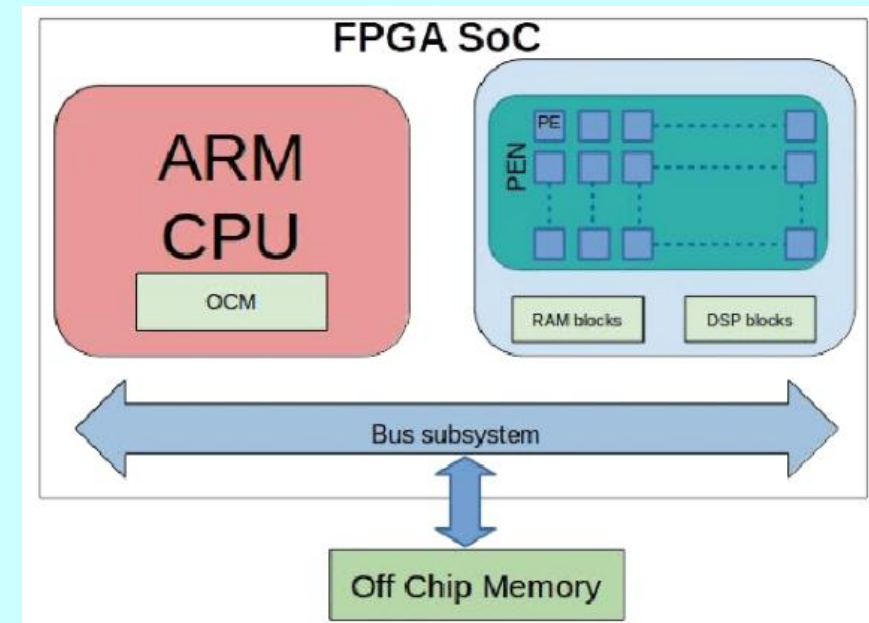
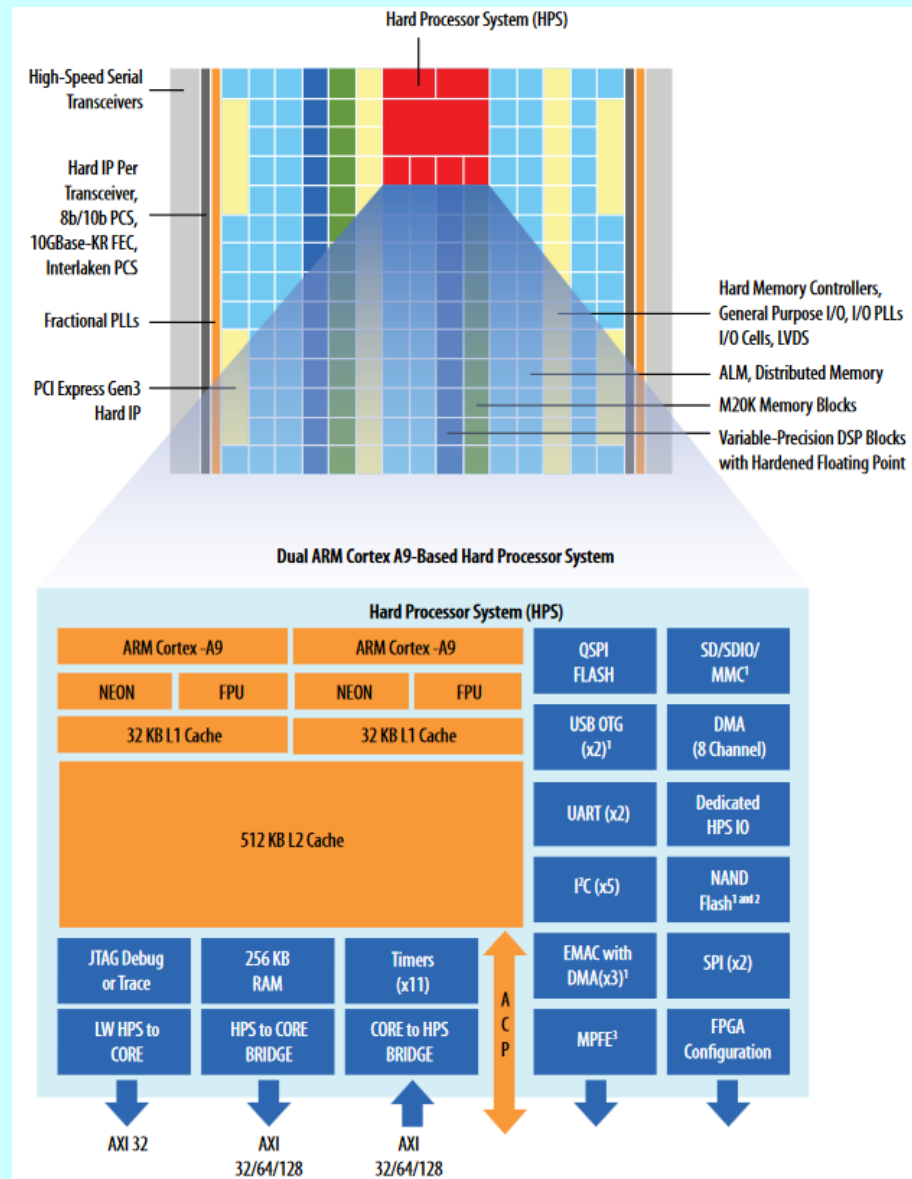
Porównanie podstawowych parametrów CPLD i FPGA:

	FPGA	CPLD
• liczba makrokomórek przeliczeniowych	>1.000.000	2.000
• konfiguracja struktury	ulotna	nieulotna
• liczba pinów użytkownika	3.000	300
• wielkość pamięci wbudowanej	>100 Mbit	8.000 kbit
• liczba wejść zegarowych	32	4
• liczba pętli PLL	32	1
• częstotliwość pracy	> 1 GHz	300 MHz
• bloki DSP	400	-
• mnożniki wbudowane	3000	-
• oscylator wewnętrzny	-	1
• interfejsy sprzętowe magistrali	tak	-
• twarde rdzenie procesorów	4	-
• cena	< 300.000 zł	30 - 1000 zł

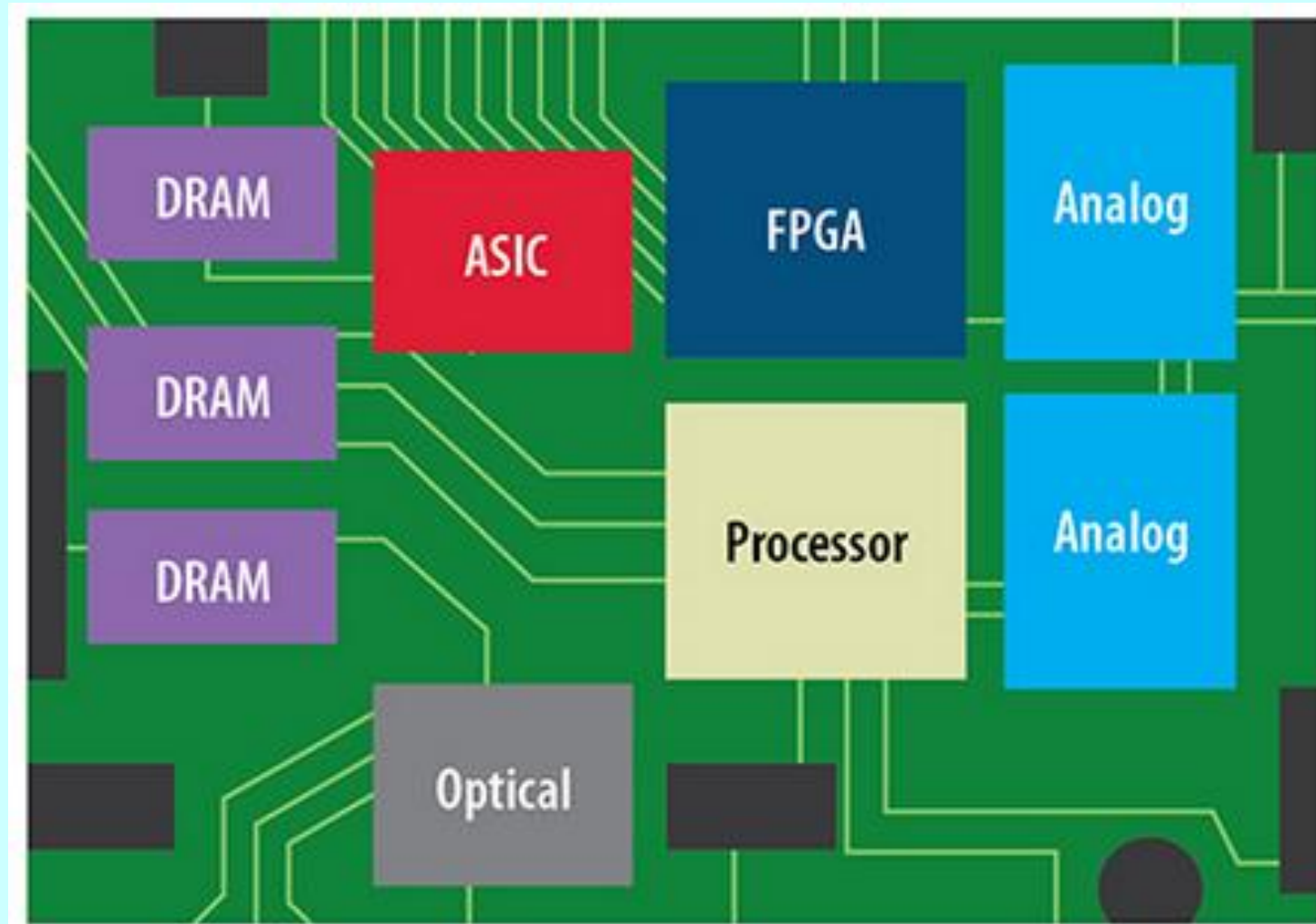
Współczesne układy FPGA to nie tylko matryce makrokomórek - ale programowalne złożone struktury cyfrowe.



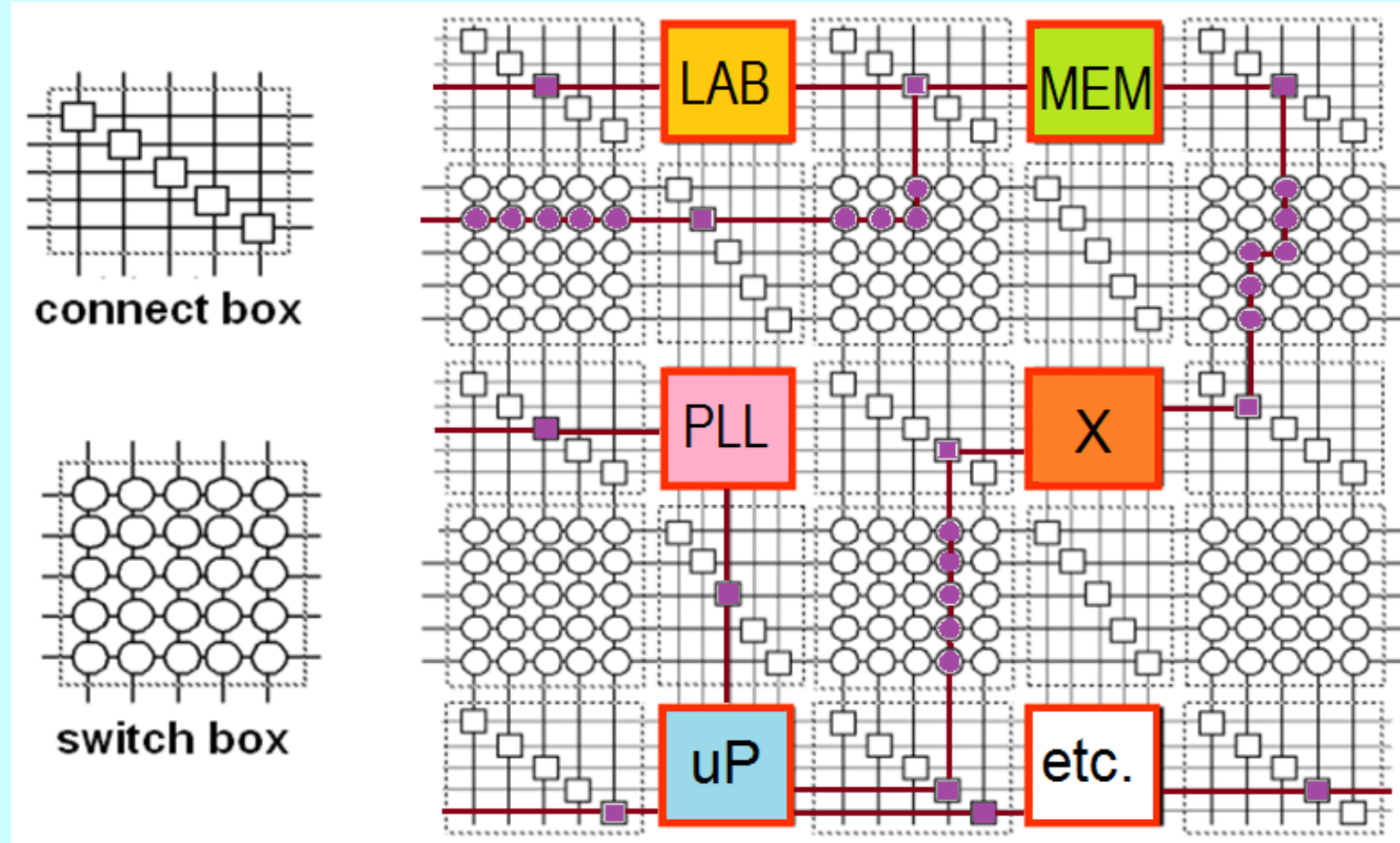
Jeżeli do struktury FPGA wtrącony jest chip mikroprocesora – mówimy o układzie SoC (System-on-Chip).



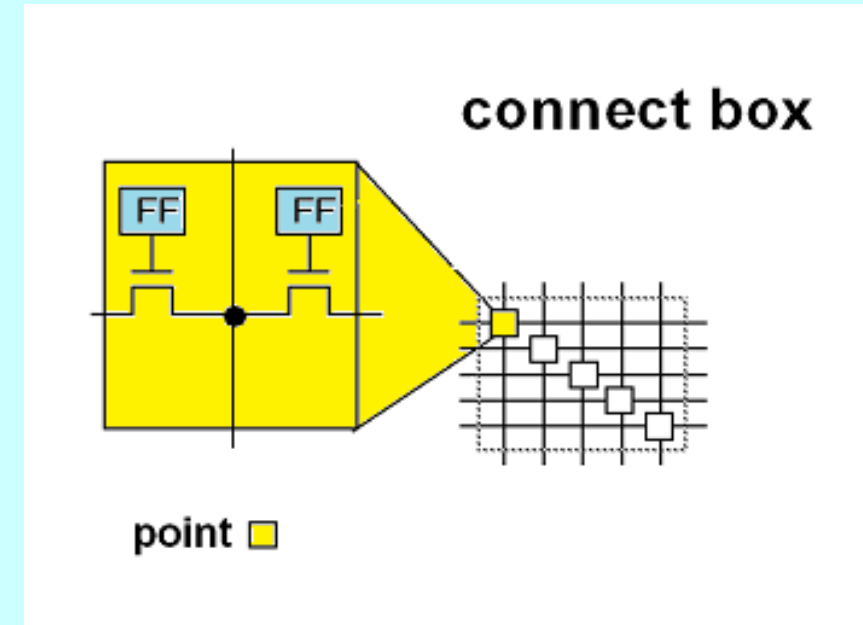
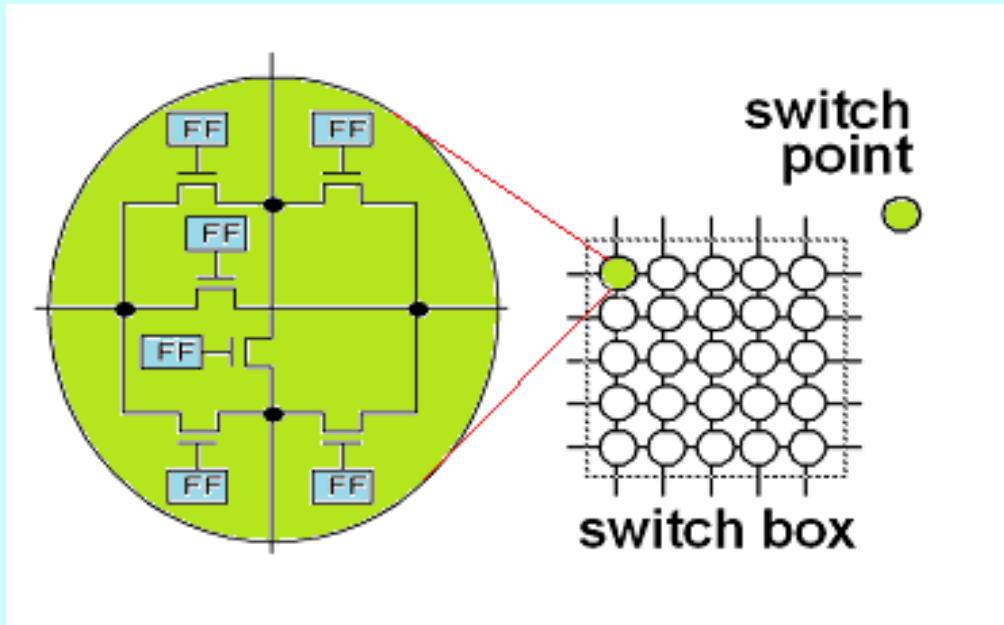
Jeżeli we wspólnej, hybrydowej obudowie zamknięte są różne układy cyfrowe różnej kategorii, optyka zintegrowana i wybrane układy analogowe - otrzymujemy SiP (System-in-Package).



Istota programowania/konfigurowania FPGA jest tworzenie w strukturze sieci sygnałowej z użyciem połączeń segmentowych i kluczy elektronicznych. W FPGA są to klucze elektroniczne podtrzymywane zawartością komórek RAM/przerzutników. Końce połączeń segmentowych stykają się w matrycach – switch/connect box.



Architektura pojedynczego węzła matrycy switch box i connect box jest następująca:



Węzeł stanowi zestaw kluczy tranzystorowych, których bramki są polaryzowane przez zawartość jednobitowej pamięci RAM/FF. W zależności od wartości napięcia na bramce – tranzystor pozostaje włączony lub nie.

W czasie konfigurowania układu FPGA przez komputer, do komórek tych pamięci są wpisywane:

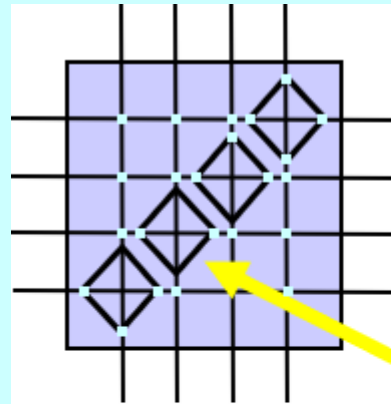
‘0’ – co tworzy połączenie, czyli włącza klucz;

‘1’ – pozostawia klucz w stanie otwartym.

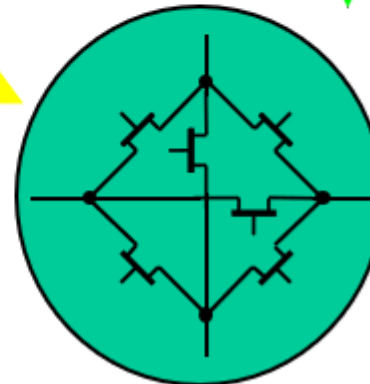
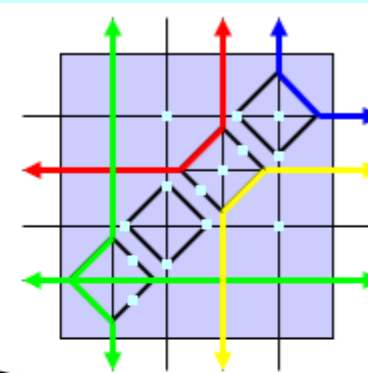
Przed zaprogramowaniem FPGA lub po wyłączeniu zasilania, wszystkie klucze pozostają otwarte.

Zatem włączenie odpowiednich zestawów kluczy powoduje czasowe wytworzenie w FPGA sieci łączącej moduły wewnętrzne układu. Połączenie różnych i w różnej konfiguracji modułów wewnętrznych powoduje powstanie wewnątrz FPGA zupełnie inaczey/nowego działającego układu/systemu – co obowiązuje jedynie w czasie trwania zasilania. Programując FPGA tworzymy zupełnie nowy funkcjonalnie układ cyfrowy wewnątrz struktury – unikając skomplikowanych i kłopotliwych połączeń zewnętrznych. Może nim być np. mikroprocesor.

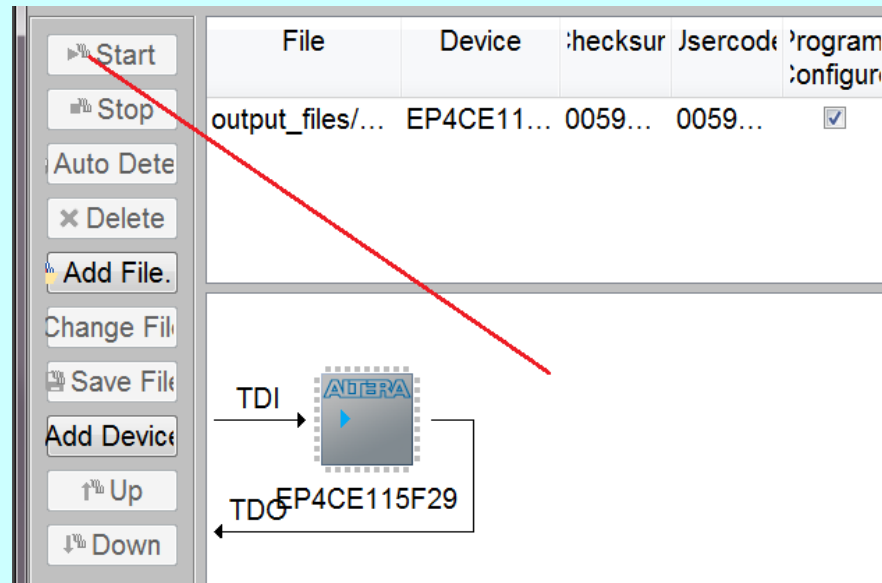
Układ przed zaprogramowaniem



Układ po zaprogramowaniu

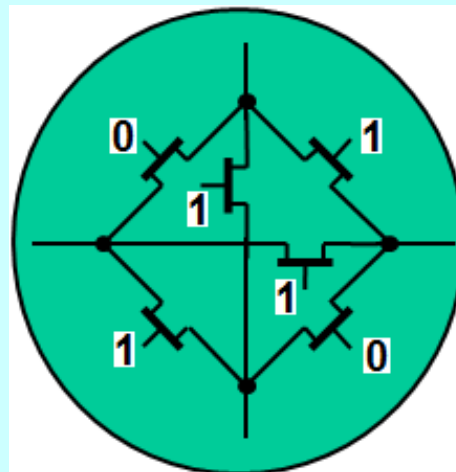
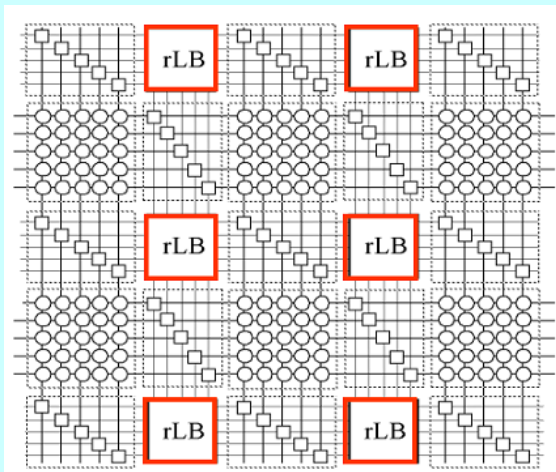


Po końcowej kompilacji - przyciskając **Start** w programatorze wysyłamy do FPGA ciąg 0 i 1, tzw. bitstream. Jest to bitowa mapa połączeń w FPGA, która dociera do bramek wszystkich tranzystorów konfigurujących.

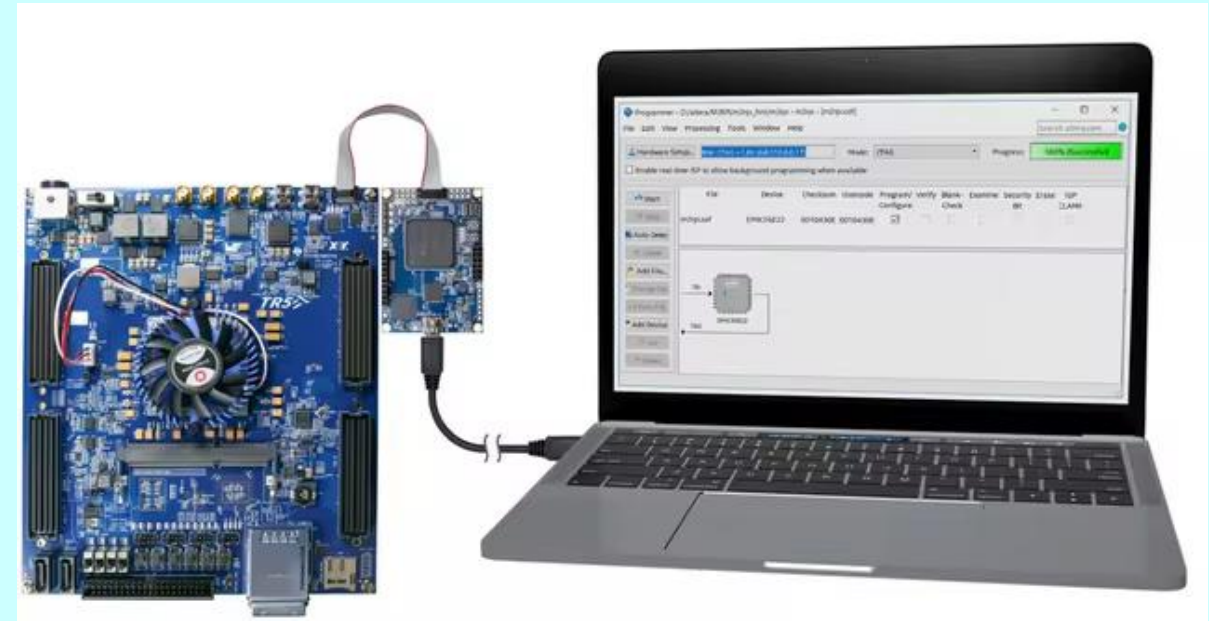
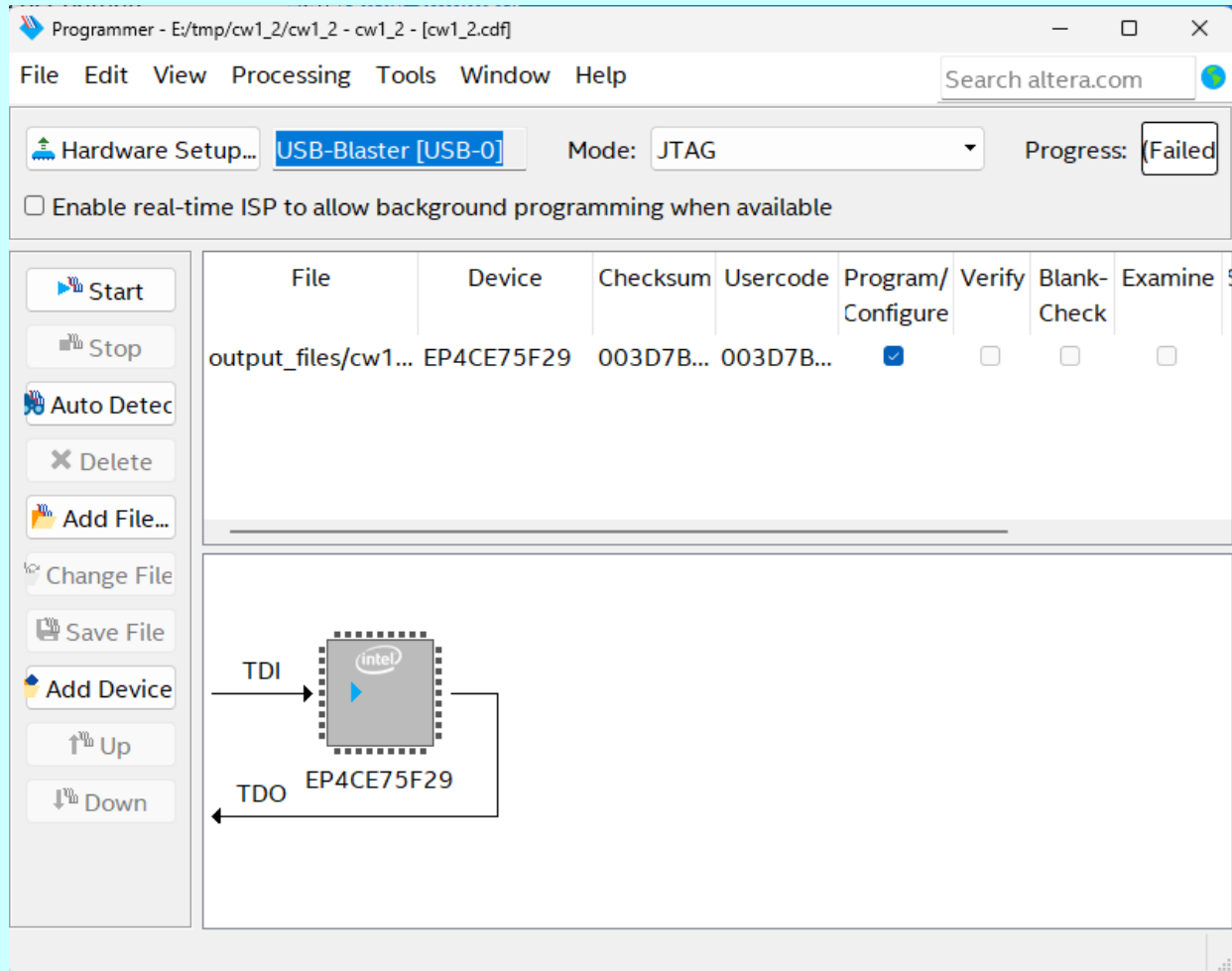


```
0000100000000010000000111000000.....1000000010100
0000100000001000010000001110.....000000101100001
00000000001000000000000000.....1110000100000010
00000110000000001110000.....111110000
* * * * *
```

```
0000100000011010000000111000000.....1000000110100
000010011001000010000001110.....0001001011
000010000110001000000000000000.....111001010
000010011001100000000001110000.....111110001.
```

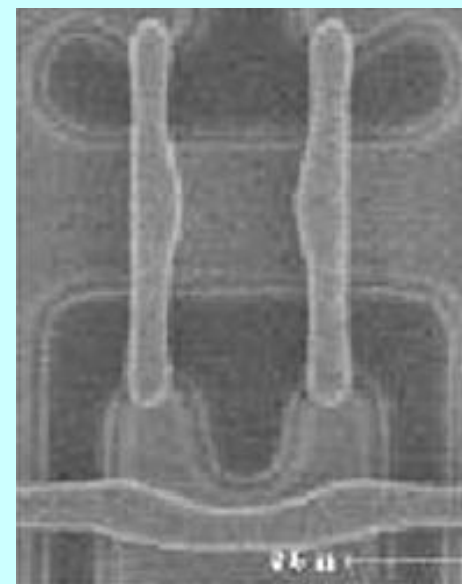
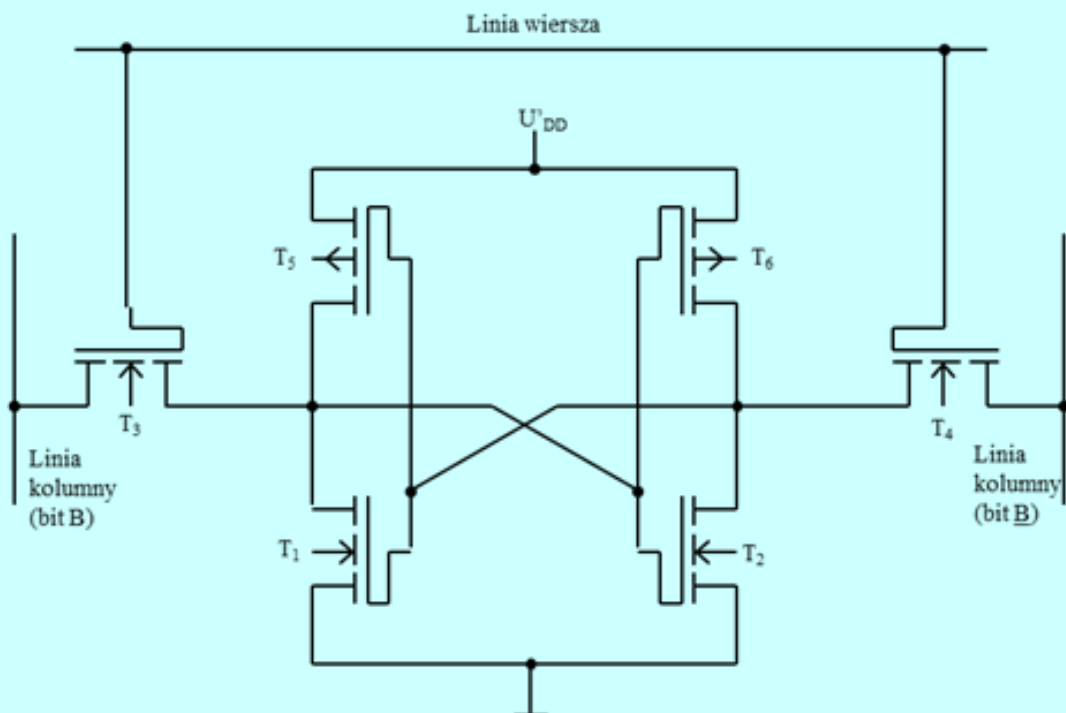


Uwzględniając około mln makrokomórek w FPGA, w każdej makrokomórce kilkanaście kluczy konfiguracyjnych, komórki połączone w LAB, między którymi przebiega 6 typów kanałów łączących z kilkaset segmentami – można oszacować ile mln/mld kluczy znajduje się w FPGA. W czasie szeregowej transmisji może dojść do przekłamania kilku bitów co sygnalizuje programator, wówczas proces konfiguracji należy powtórzyć.

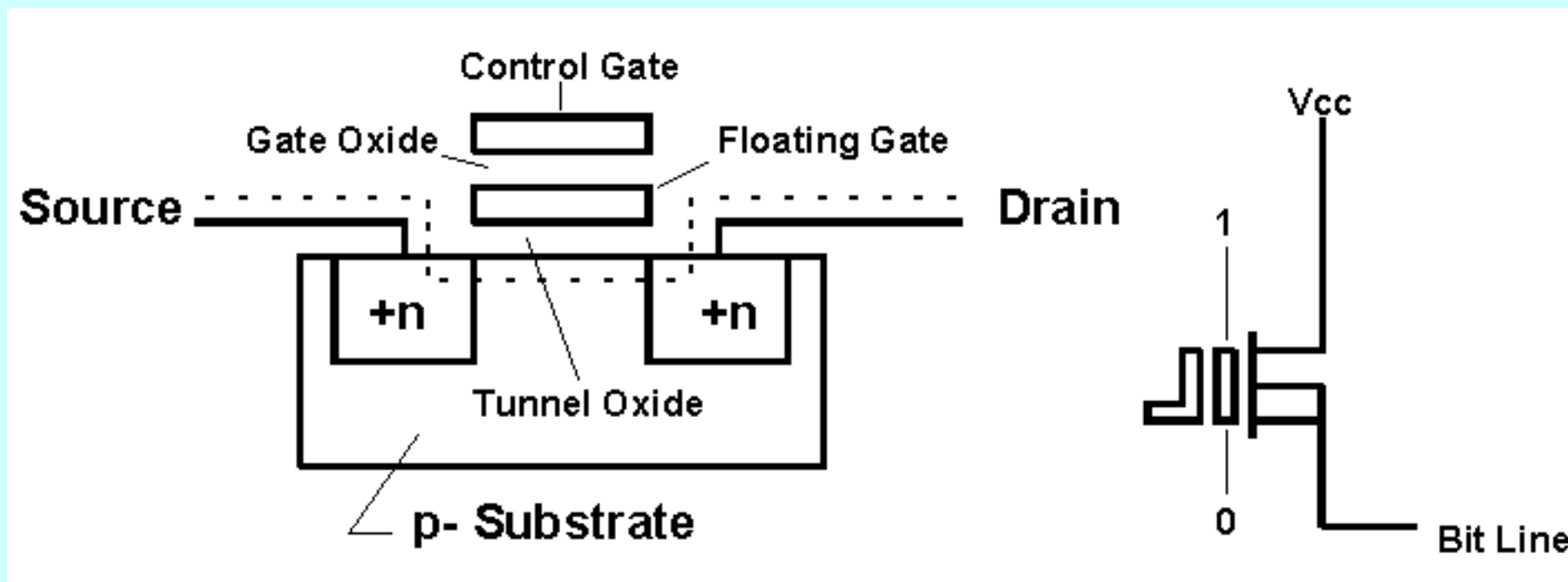


Stosuje się 3 techniki podtrzymania konfiguracji w PLD/FPGA: SRAM, EEPROM/Flash, Antifuse

Rozwiązania oparte na pamięci statycznej SRAM charakteryzują: małym rozmiarem komórki zapewniającym wysoką skalę integracji/upakowania układu, wadą jest ulotność stanu – volatile, nieograniczona liczba przeprogramowań i niskie napięcie zasilające. Komórka SRAM zawiera przerzutnik złożony z 6 tranzystorów. Dane są pamiętane/podtrzymywane dopóki układ jest zasilany, pamięć jest programowana w układzie.



Zastosowanie komórek pamięci Flash/EEPOM wymaga większej powierzchni elementów pamięciowych, bardziej są złożone warunki programowania, zaletą jest nieulotność konfiguracji, wadą ograniczona liczba programowań. Programowanie odbywa się w układzie, stan jest zapamiętywany w przestrzeni izolowanej bramki tranzystora, tzw. floating-gate. Rozwiązanie jest stosowane w układach CPLD i małych FPGA.



Technika antifuse stosowana jest w układach FPGA specjalnego zastosowania, gdzie wymagana jest niezawodność konfiguracji. Są to układy jednokrotnego programowania, czyli gęsto upakowane struktury PLD.

Tactical Missiles

AIM 9X Sidewinder
Hellfire
Brimstone
Patriot
PAC-3
GBI
THAAD
MLRS
Longbow
Harpoon



Commercial Aircraft

737
747
757
767
777
Dash-8 400
Airbus 319, 320, 321, 340, 380



Military Aircraft

F-22
KC-135
Apache
F-117
CH-60
B1B
Lamps
Nimrod
E2C
F-16
Comanche
JSF
F-14
A-10
F-15
F-18
C-130



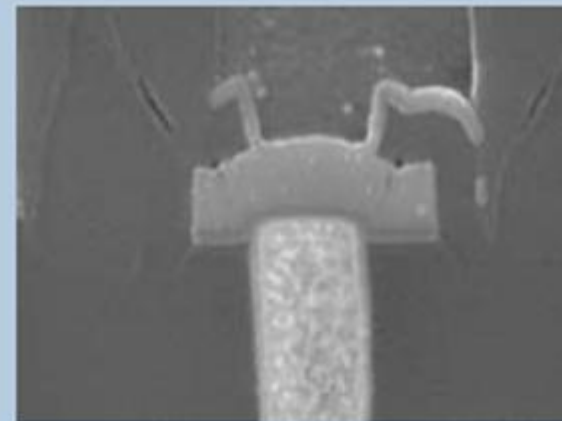
Military Ground Vehicles

Bradley Fighting Vehicle
M1 Abrams

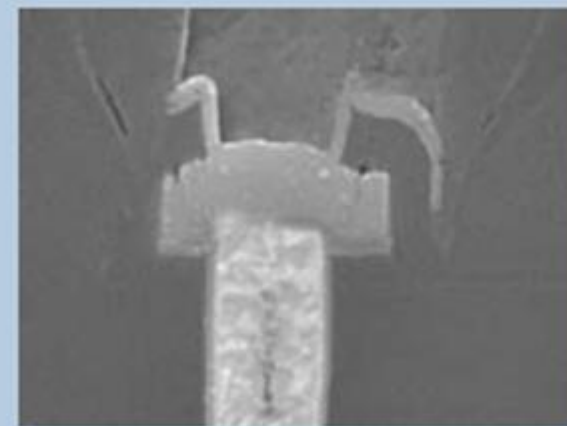


Military Systems

ATACMS
TACMS
TADS/PNVS
WCM
Lantim
IFCS
AITG
ESAF
ALQ135
AAQ24
ATIRCM
IDECM



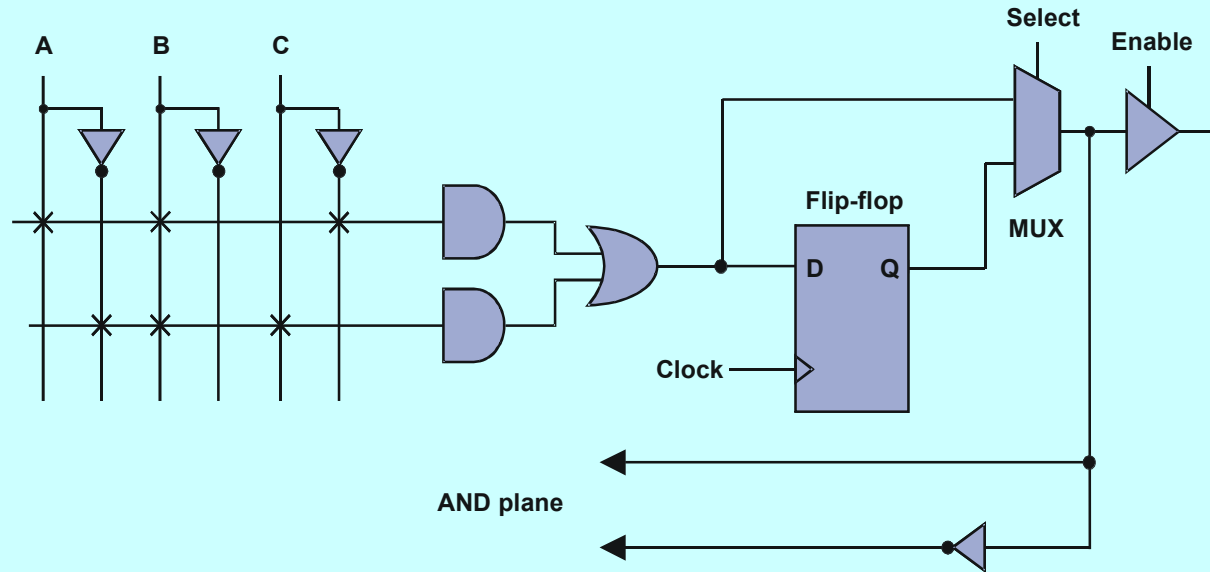
Unprogrammed Antifuse Element



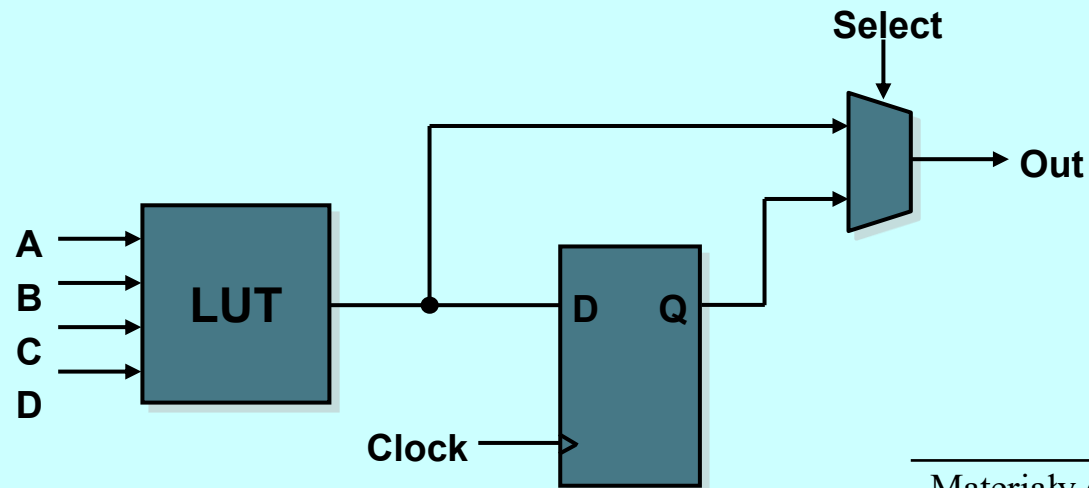
Programmed Antifuse Element

Różnica architektury makrokomórek CPLD i FPGA:

CPLD



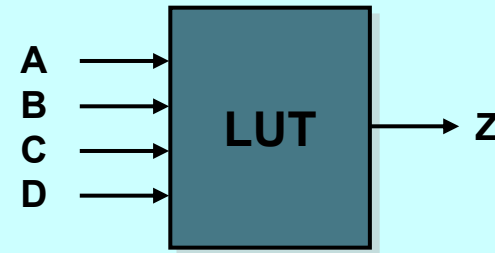
FPGA



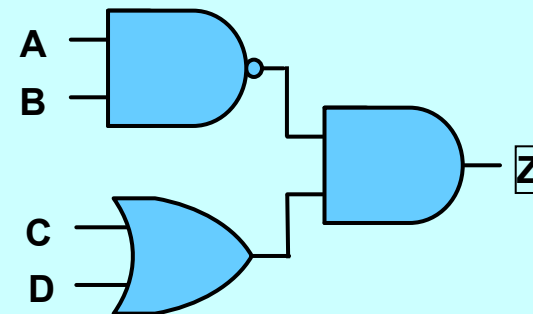
Architektura FPGA – czym jest tablica LUT (Look-Up Table)? Jest to szybka jednobitowa podręczna RAM dostępna w każdej makrokomórce. LUT realizuje jednowyjściową tablicę prawdy (Truth-Table), sygnałami wejściowymi są wejścia adresowe pamięci. Jej zawartość jest wpisywana w trakcie konfiguracji FPGA, taka konstrukcja pozwala syntezywać dowolną funkcję kombinacyjną o jednym wyjściu – np. bramkę logiczną. Jest to rozwiązanie szybsze i zajmuje mniej miejsca niż tablice programowalne w układach PLD.

A	B	C	D	Z
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0

Truth-table

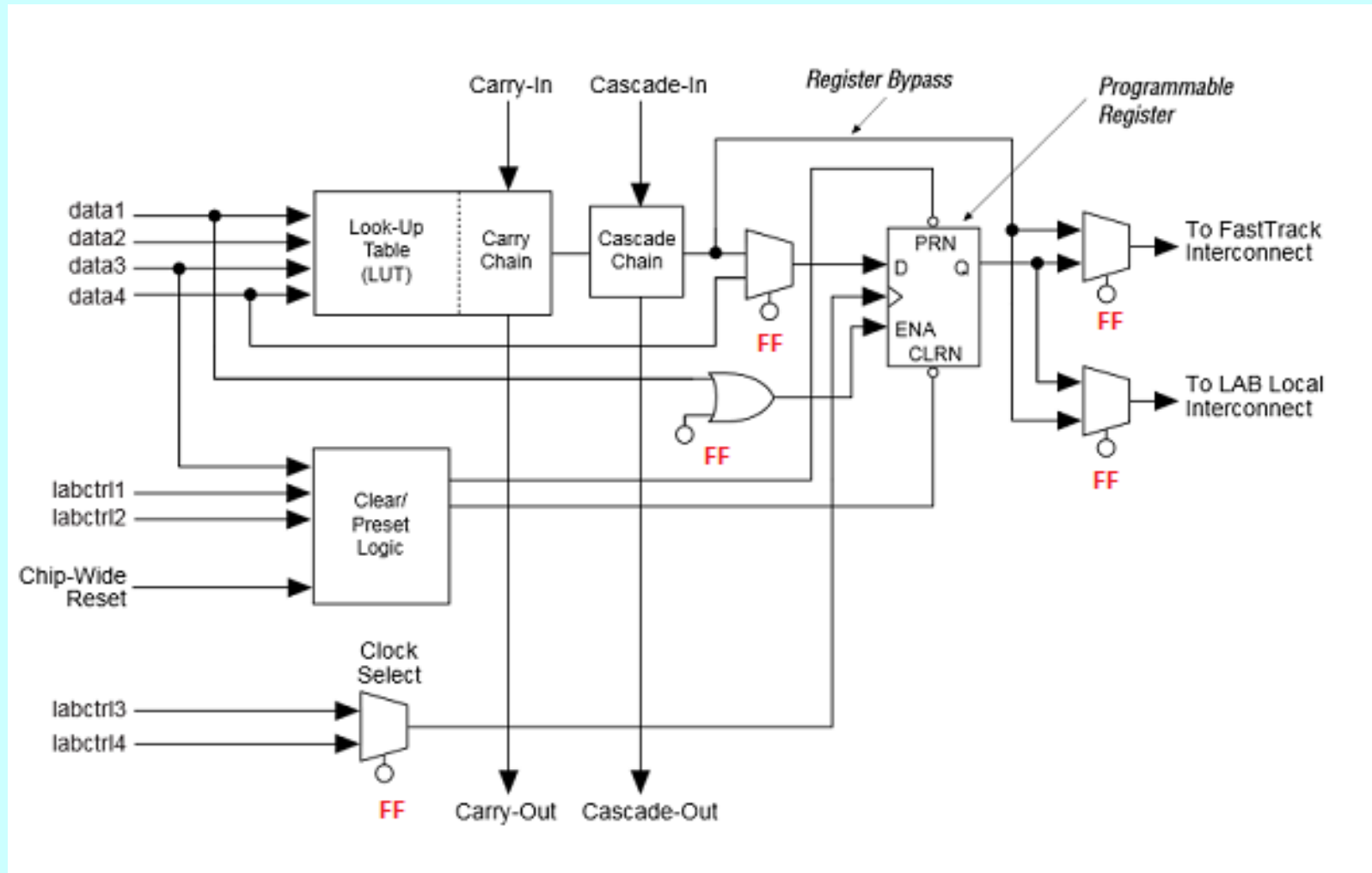


LUT implementacja

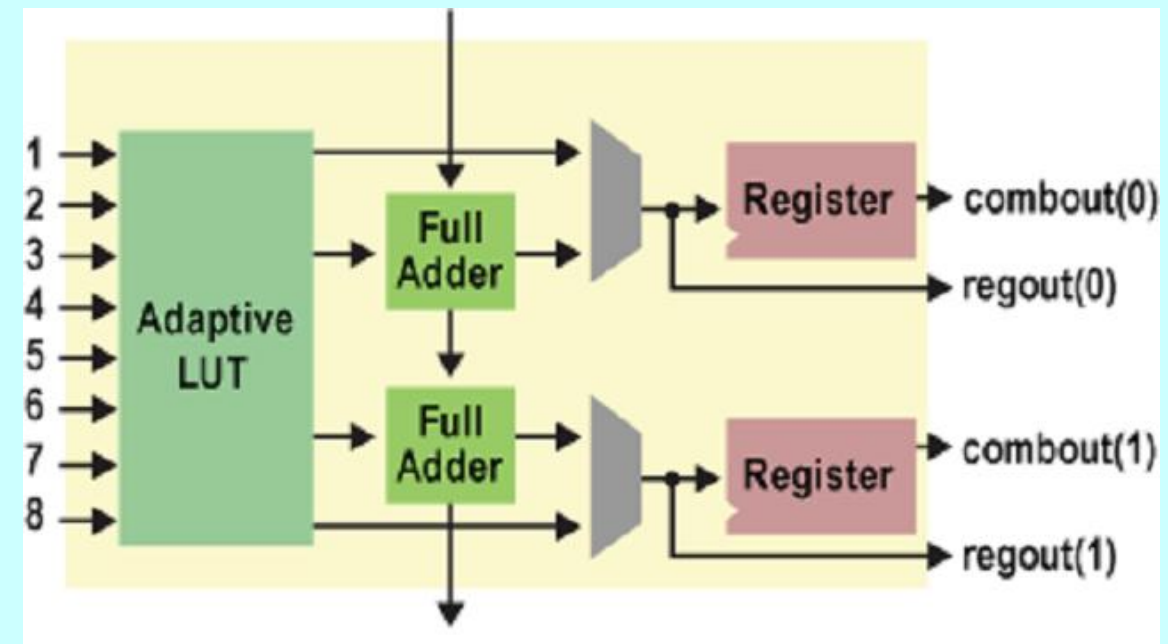
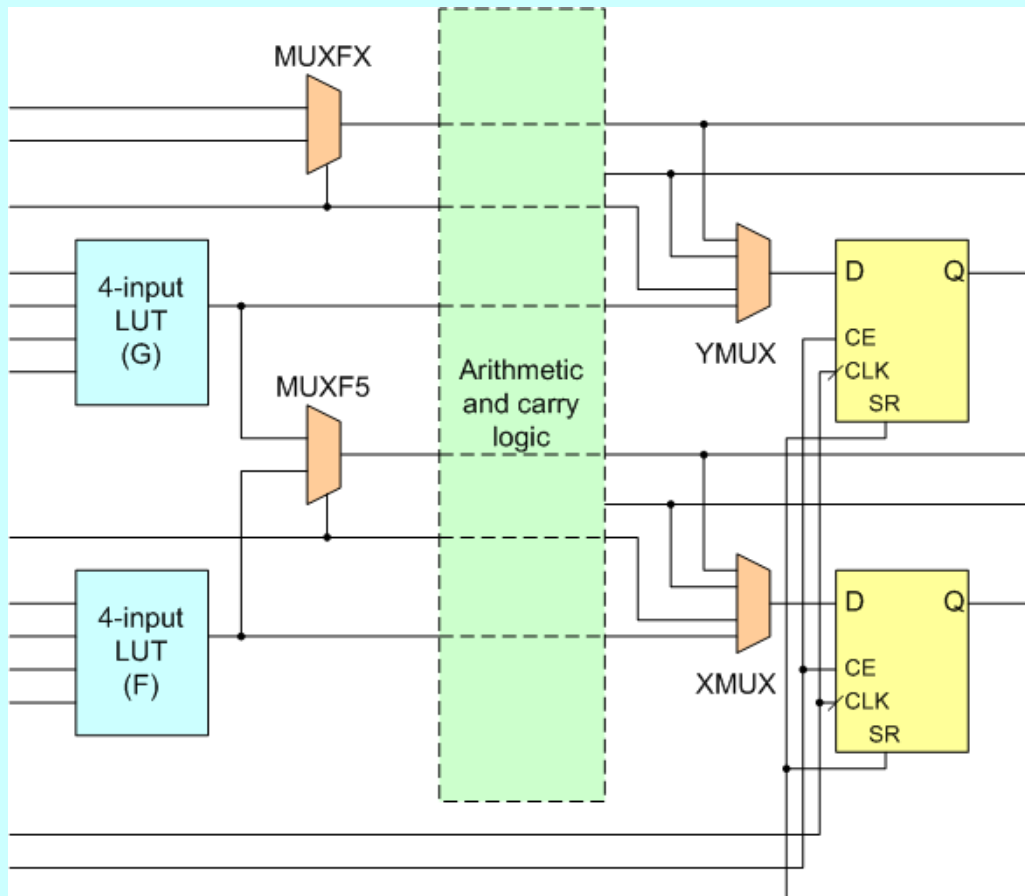


„Bramkowa” implementacja

W rzeczywistości makrokomórka FPGA jest bardziej złożonym i konfigurowalnym układem niż uproszczony schemat pokazany poprzednio:



Istnieje wielu producentów FPGA, każdy z nich oferuje kilka rodzin układów i stosuje nieco inne rozwiązania makrokomórek – nazywany logicznymi elementami LE (Logic Element) lub Logic Cell (LC, Slice). W rzeczywistości makrokomórka FPGA jest bardziej złożonym i konfigurowalnym układem niż uproszczony schemat pokazany poprzednio:



Miarą wielkości układu FPGA nie jest liczba bramek a zwłaszcza tranzystorów – jest nią liczba makrokomórek.



Duże FPGA w technologii procesowej 10 nm do aplikacji centrów danych i zastosowań sieciowych, zawiera 4-rdzeniowy ARM, do 3 mln LEs, 3 mln F-Fs, pamięć do 300 MB.



FPGA SoC o wysokiej wydajności i dużej przepustowości, zawiera do 2 mln LEs, stosowany do syntezy układów ASIC na bazie FPGA.



FPGA o niskim poborze mocy, przemysłowe rozwiązanie, do szybkiego prototypowania układów, zawiera do 120 000 LEs, 4 MB pamięci.

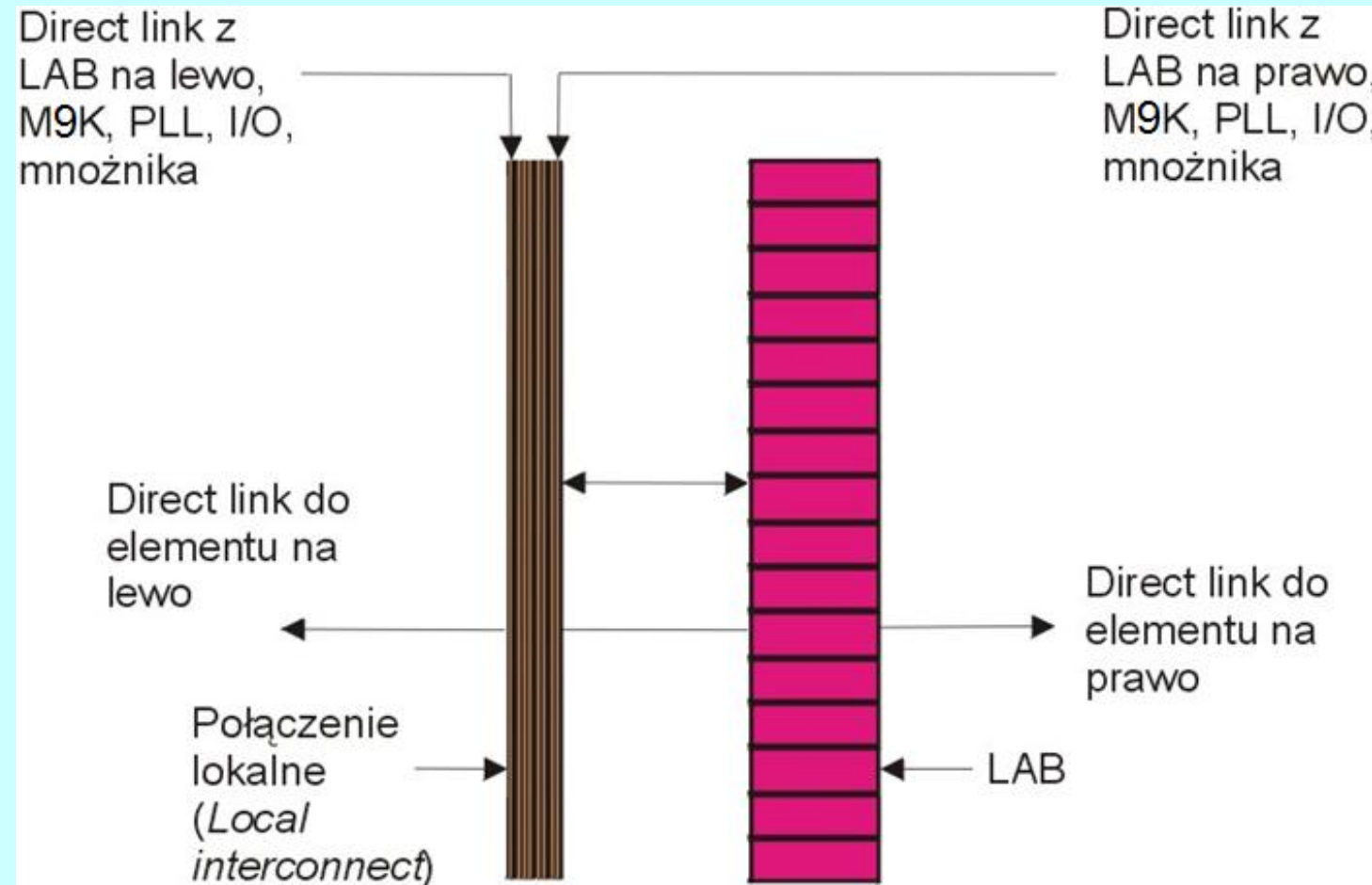


Współczesny układ CPLD o architekturze FPGA, zawiera do: 50.000 LEs, do 500 pinów I/O, nieulotną architekturę, zawiera wbudowaną pamięć Flash oraz przetwornik AC i oscylator.

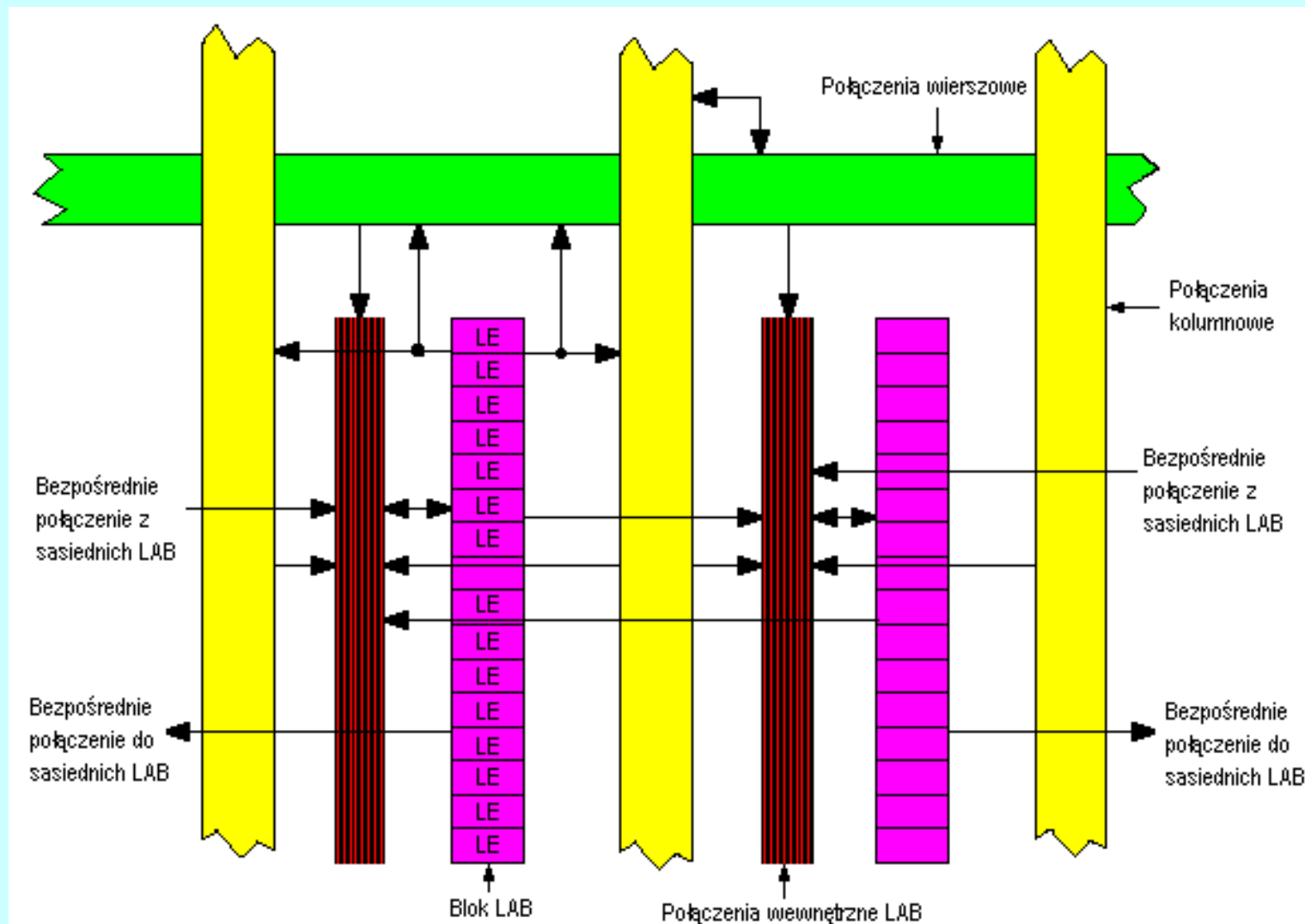


Układ telekomunikacyjny, do 50.000 LEs, zawiera boki pamięci, rozbudowane makrokomórki cyfrowego przetwarzania sygnałów (DSP), transceivery z szybkością transmisji do 25,78 Gb/s.

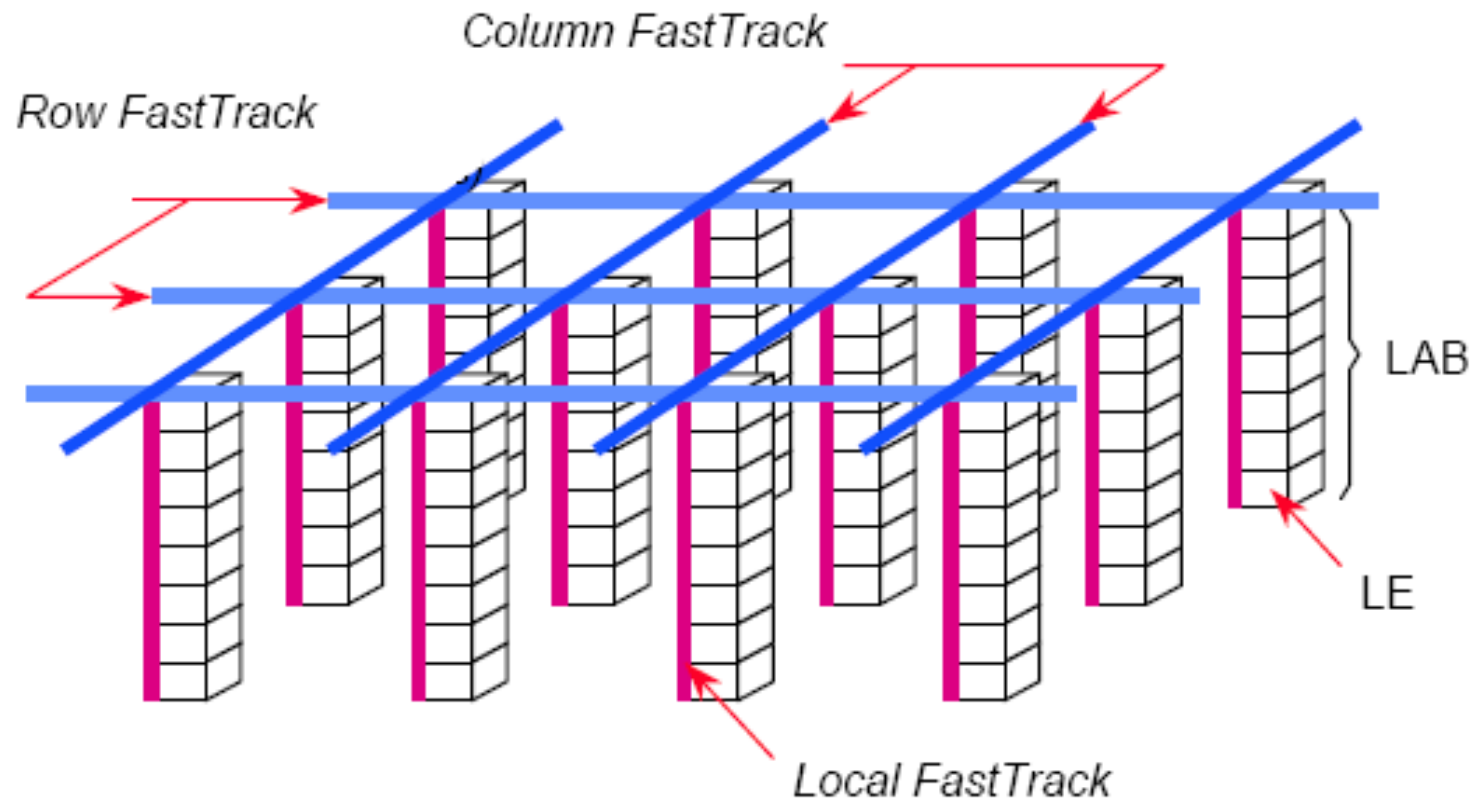
Więszym organizacyjnie modułem niż makrokomórka w układzie FPGA jest blok logiczny LAB (Logic Array Block). Moduł zawiera od 8 do 16 makrokomórek oraz wewnętrzną sieć szybkich połączeń między otoczeniem i makrokomórkami, tzw. Local Interconnect



Bloki LAB makrokomórek włączone są w globalną sieć sygnałową architektury FPGA. Poprzez segmentowe połączenia wierszowe i kolumnowe i matryce switch/connect box mogą się łączyć z odległymi modułami FPGA:

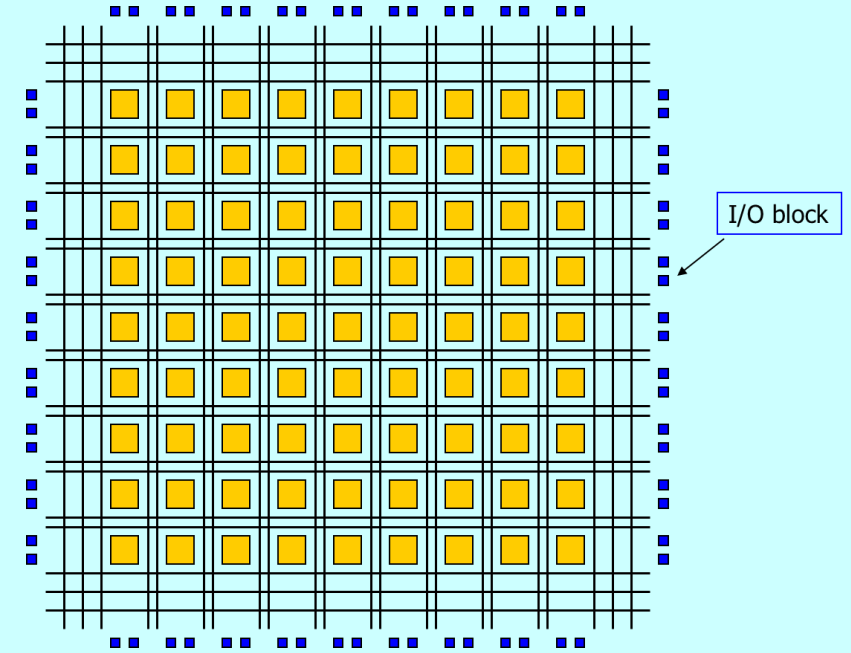
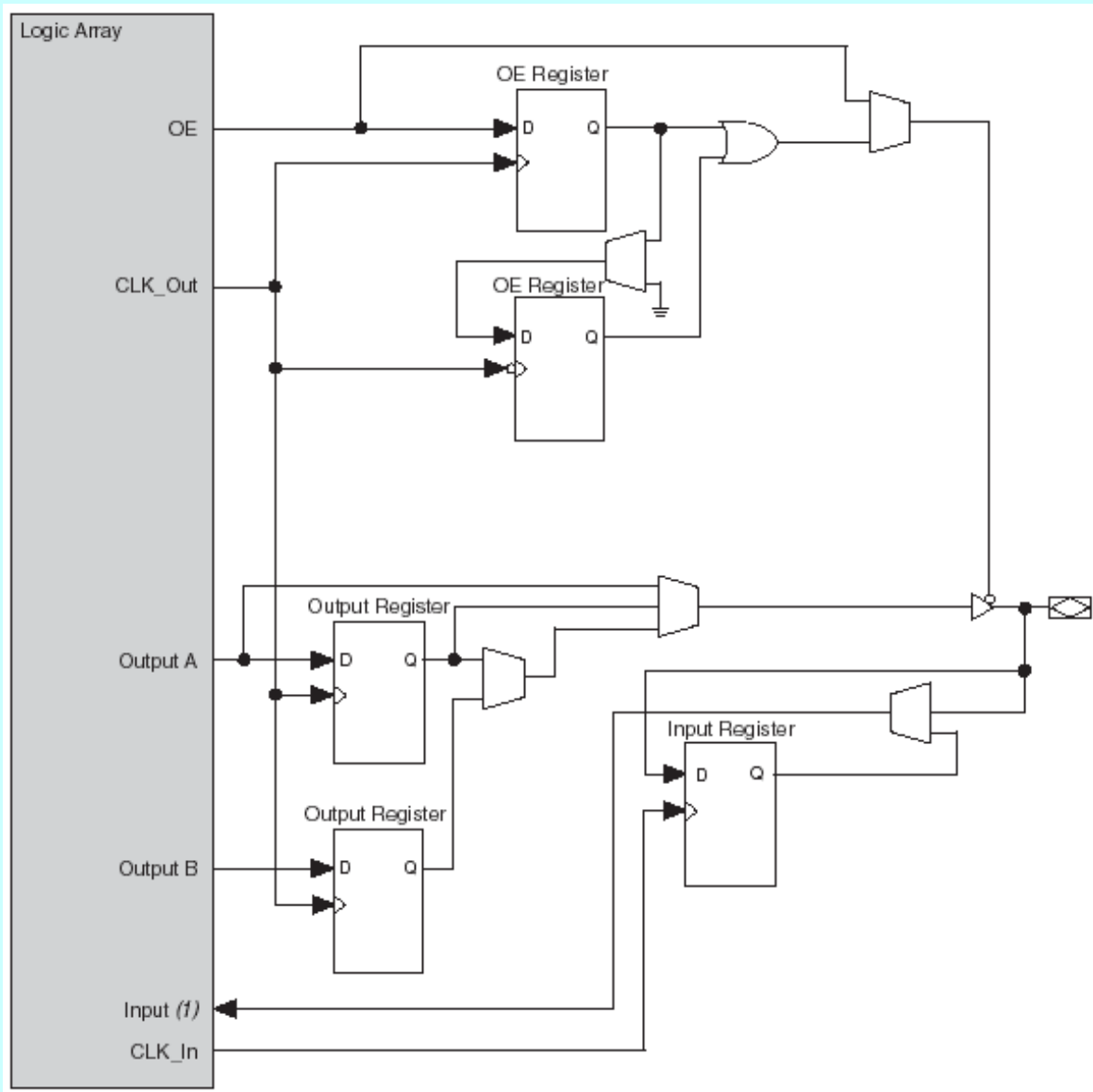


Architektura FPGA jest trójwymiarowa, w prezentacji przedstawiono tylko jej model. Faktyczna architektura FPGA stanowi know-how producenta i informacja o niej nie jest udostępniana.



W Cyclone IV ma 6 typów połączeń:
direct link łączące sąsiednie LAB
w układzie wierszowym,
R4 łączące 4 LABs z lewej/prawej,
R24 łączące przez szerokość układu,
register chain łączące przerzutniki
LE w LAB,
C4 łączące 4 LAB w górę/dół,
C16 łączące kolumnowo przez cały
układ.

Bazowa wersja architektury FPGA, oprócz bloków LAB i połączeń między nimi – zawiera bloki wyjściowe I/O:



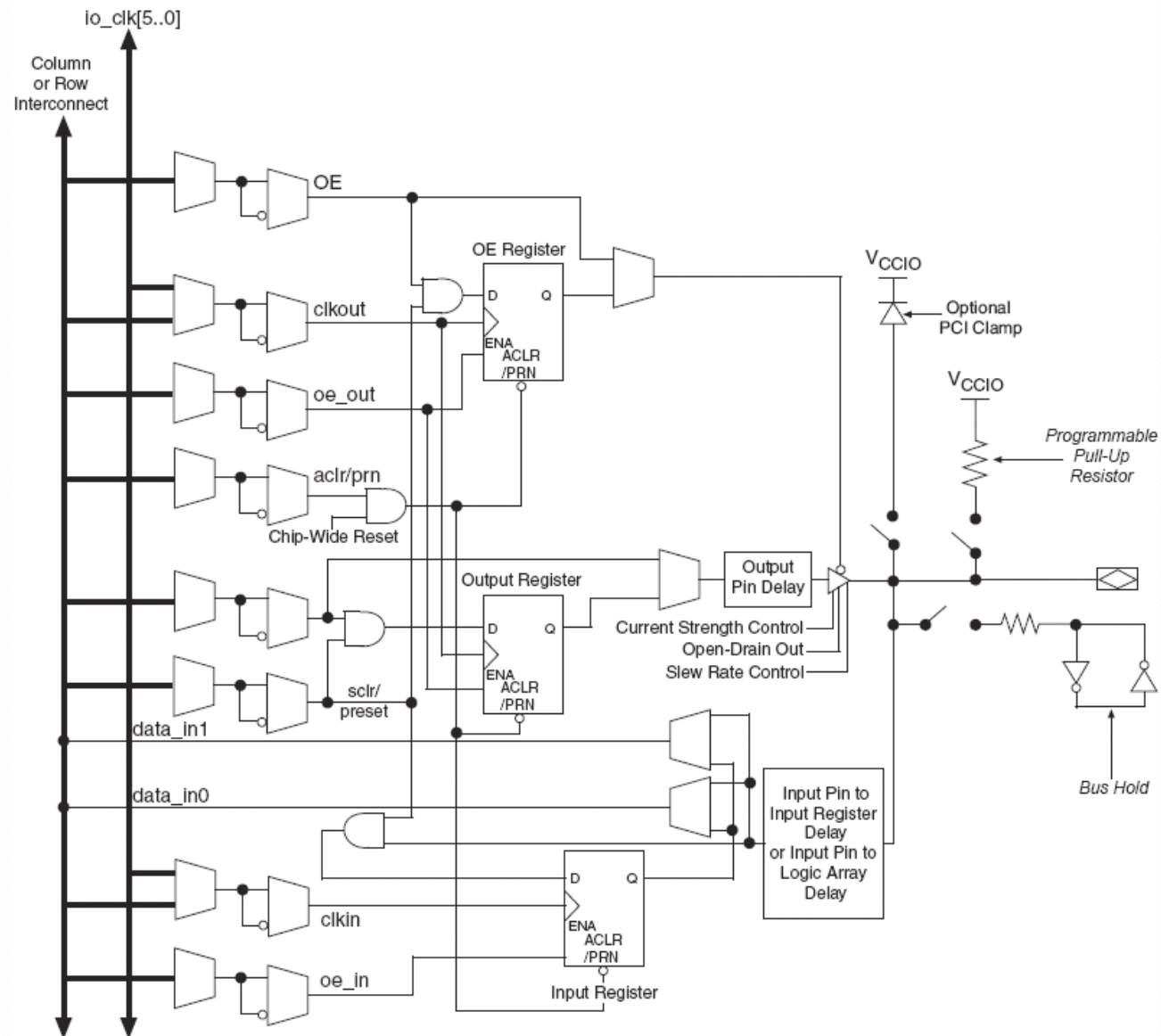
Z każdym pinem sygnałowym I/O FPGA związany jest element/blok I/O, który zawiera:

- przerzutnik wejściowy,
- dwa przerzutniki wyjściowe,
- dwa przerzutniki zezwolenia.

Element I/O może pracować jako:

- port wejściowy, wyjściowy lub dwukierunkowy.

W rzeczywistości blok I/O jest konfigurowalny i bardziej złożony:



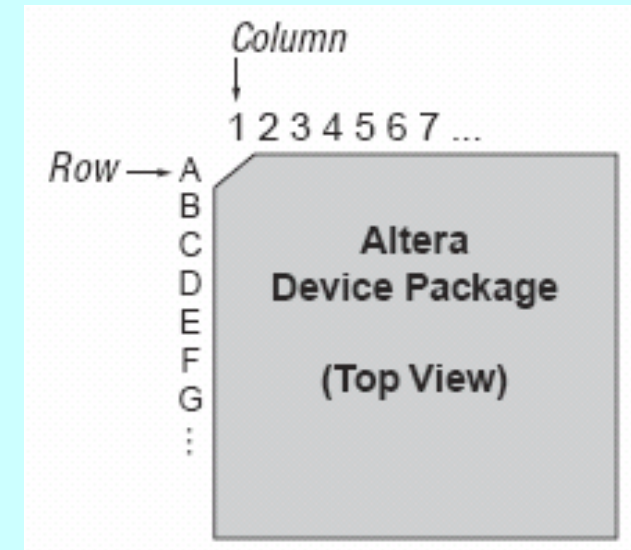
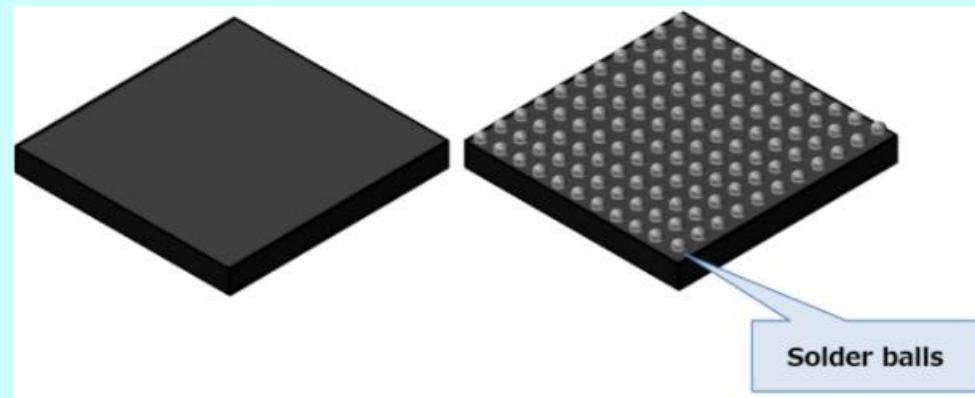
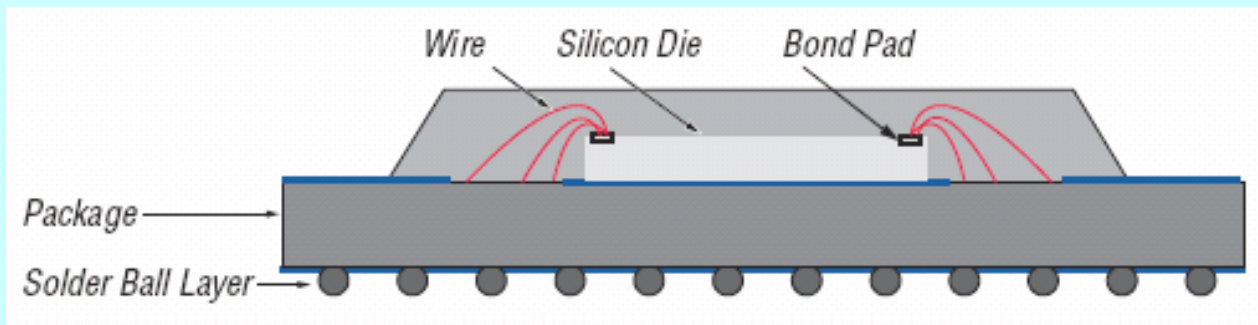
W bloku I/O można programować/ustawiać:

- transmisję symetryczną/niesymetryczną,
- standardy cyfrowe sygnałów,
- ograniczać natężenie prądu wyjściowego,
- programować współczynnika slew-rate,
- tryb wyjścia open-drain,
- mechanizm zatrzymywanie danych na szynie,
- dołączyć diodę zabezpieczającą - clamp diode,
- dołączyć rezystory pull-up/pull-down,
- opóźniać sygnały wejściowe i wyjściowe,
- stosować preemfazę transmisji,
- dopasowywać impedancję wyprowadzenia.

Układy FPGA najczęściej są pakowane w obudowach typu BGA/FBGA:

W takich obudowach obowiązuje współrzędnościowy układ numeracji wyprowadzeń ze względu na ich liczbę:

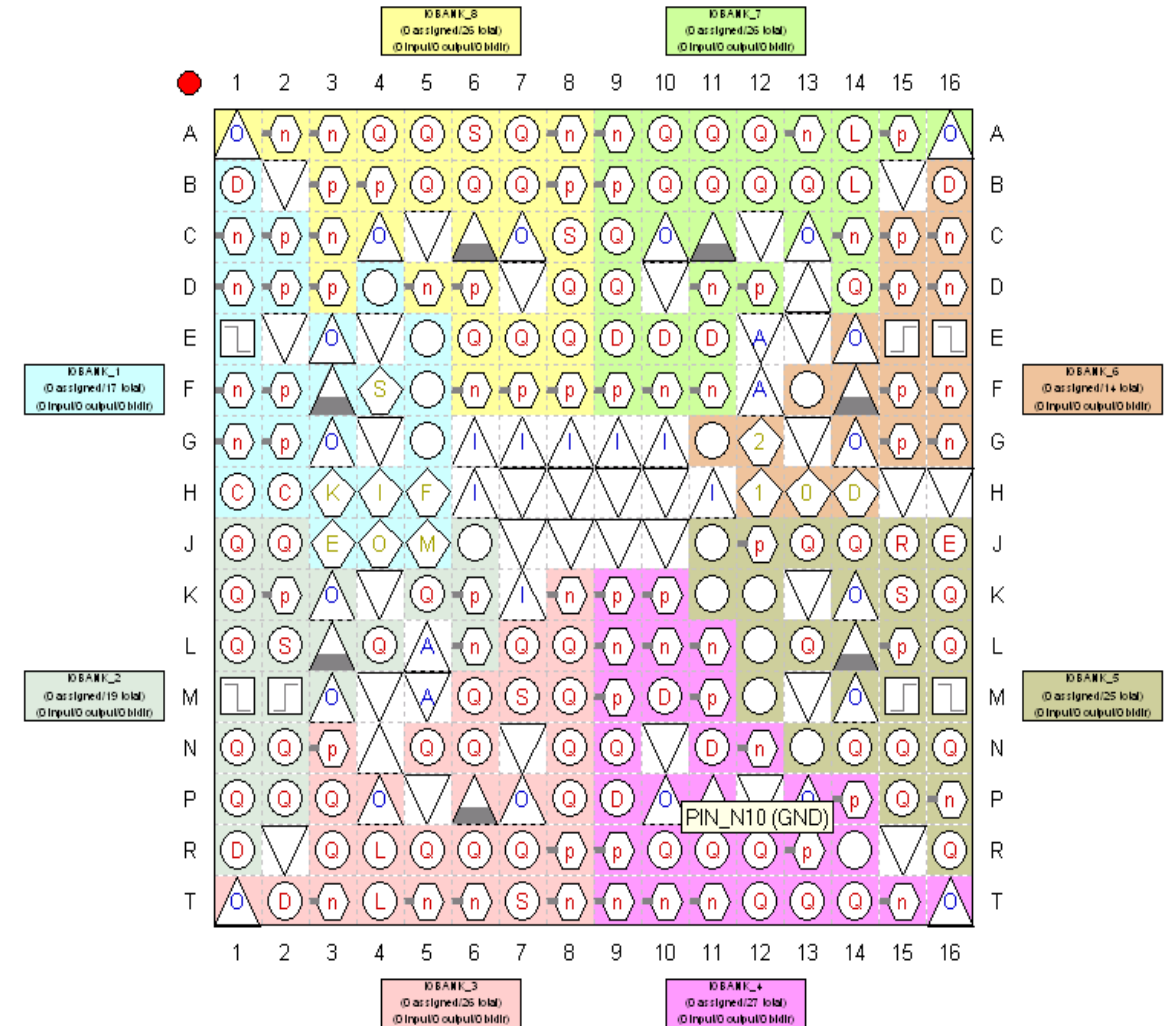
- wierszom nadane są indeksy literowe w kolejności alfabetycznej: A, B, ...,
- zabronione są nazwy wierszy zaczynające się od: I, O, Q, S, X i Z,
- kolumnom przyporządkowane są kolejne naturalne liczby dziesiętne,
- oznaczenie pinu składa się z nazwy wiersza i nr kolumny: np. **C3** oznacza pin w 3 kolumnie wiersza C,
- jeżeli „kończy się alfabet” - dodaje się kolejną literę w nazwie wiersza - np. AG11.



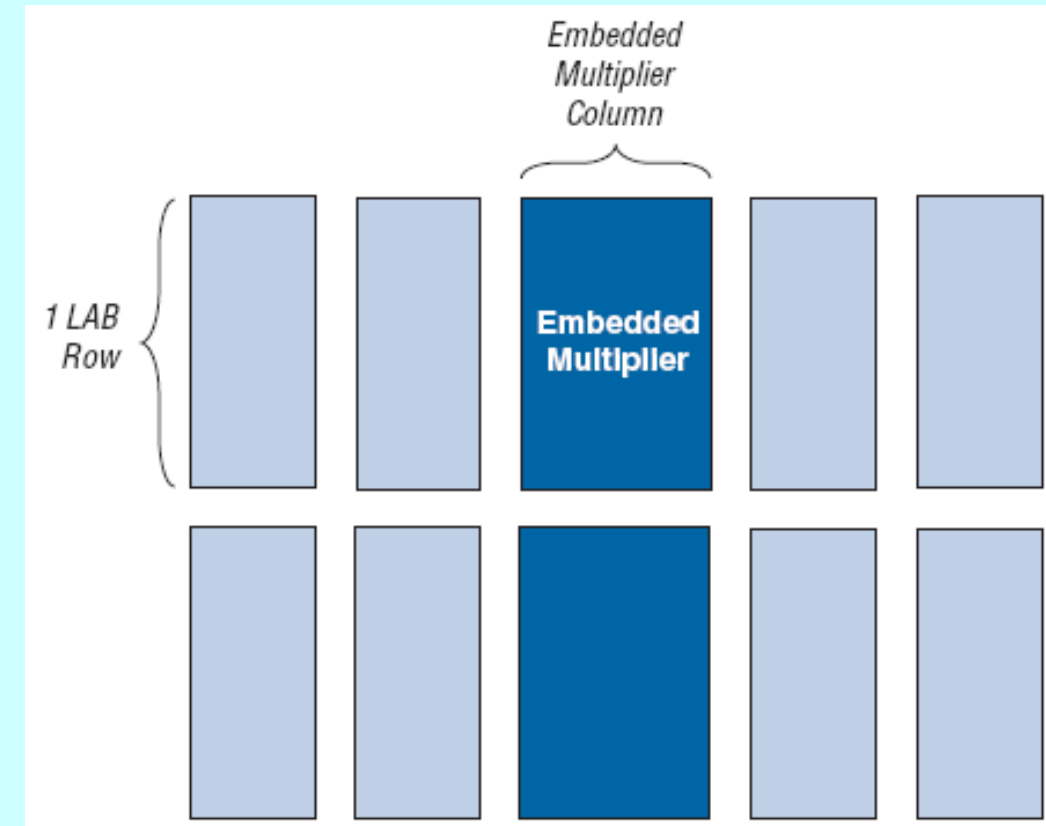
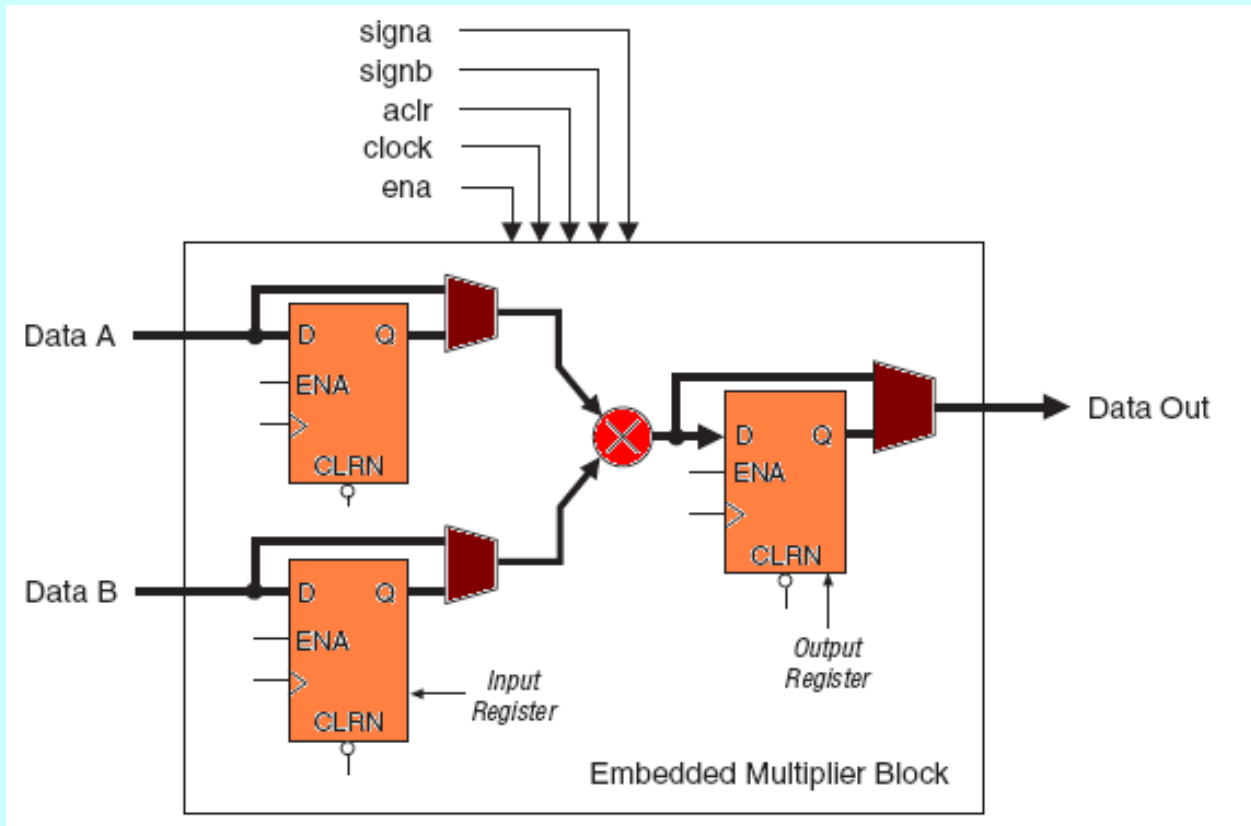
Układ wyprowadzeń układów FPGA:

Wyprowadzenia w FPGA zorganizowane są w bankach:

- każdy bank posiada oddzielny pin zasilający porty I/O tego banku,
- napięcie zasilające bank narzuca możliwe do ustawienia standardy sygnałów cyfrowych,
- banki posiadają piny napięć masy pozornej,
- bank zawiera własne piny dopasowania impedancyjnego,
- poszczególne banki zazwyczaj zawierają własne wejściowe piny sygnałów zegarowych.



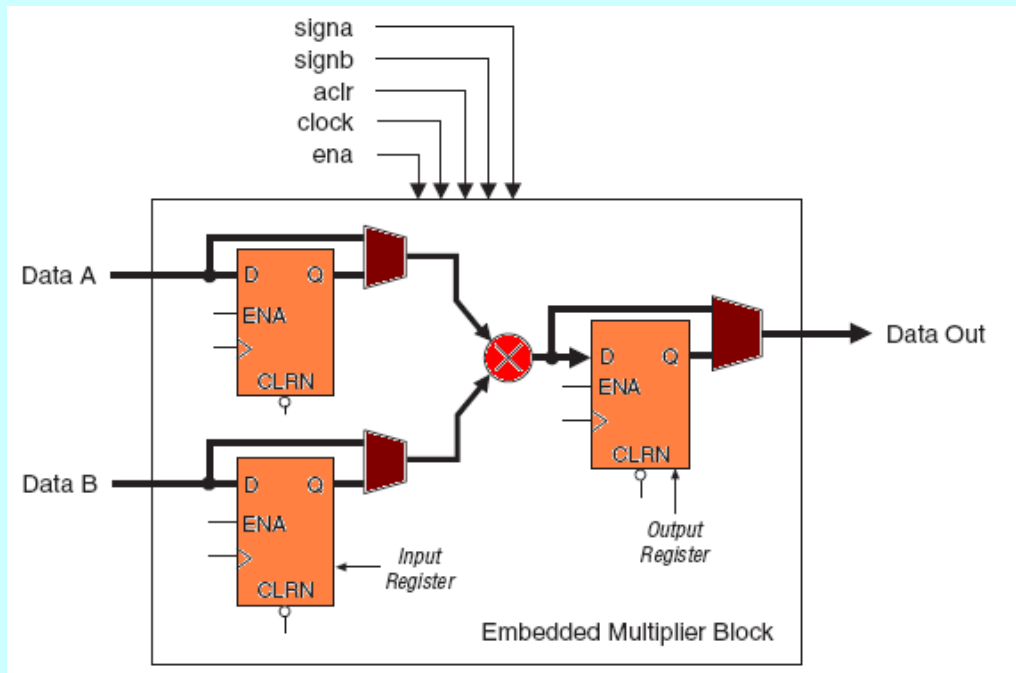
Niezależnie od zasobów tablicy logicznej (LABs, LEs), układ FPGA zawiera wbudowane struktury mnożników sprzętowych, co ułatwia złożone operacje obliczeniowe. W zależności od wersji FPGA, liczba mnożników wbudowanych waha się od kilkunastu sztuk do kilku tysięcy. Są one umieszczone kolumnowo (od 1 do 4 kolumn) i otoczone blokami LAB. Układ mnożnika zawiera: równoległy stopień mnożący, bufony/rejestry podtrzymujące dane wejściowe i wyjściowe, interfejsy wejściowe i wyjściowe.



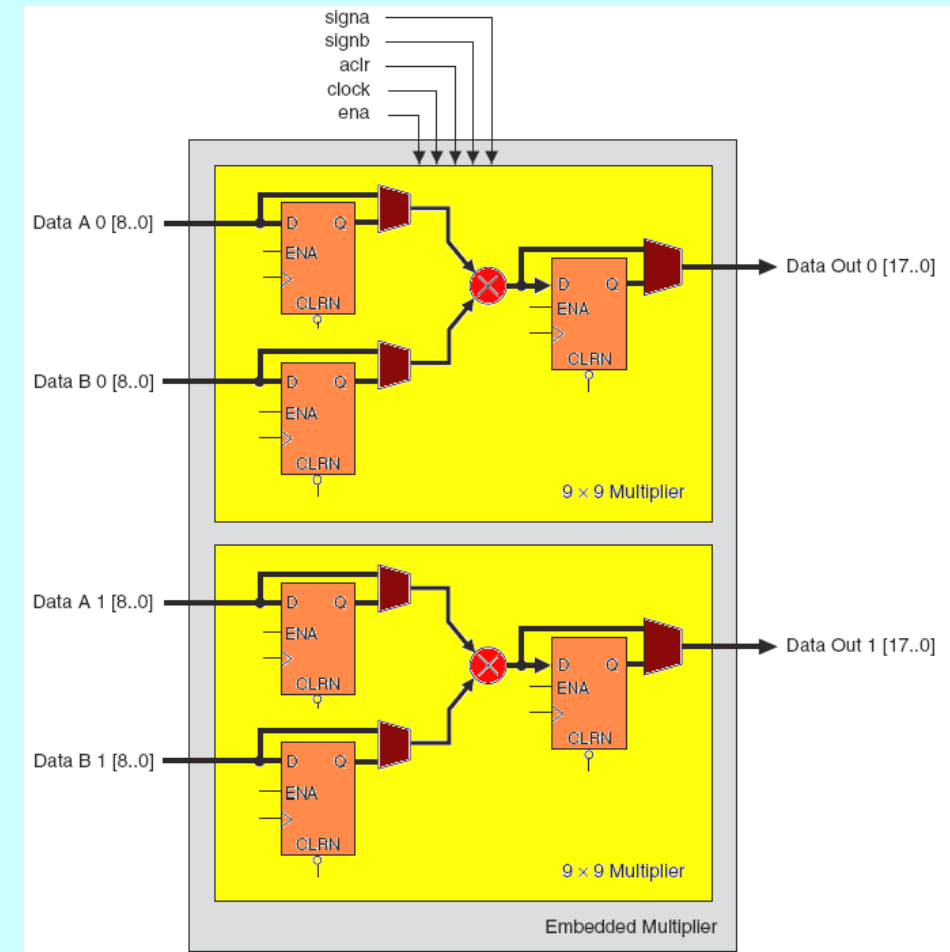
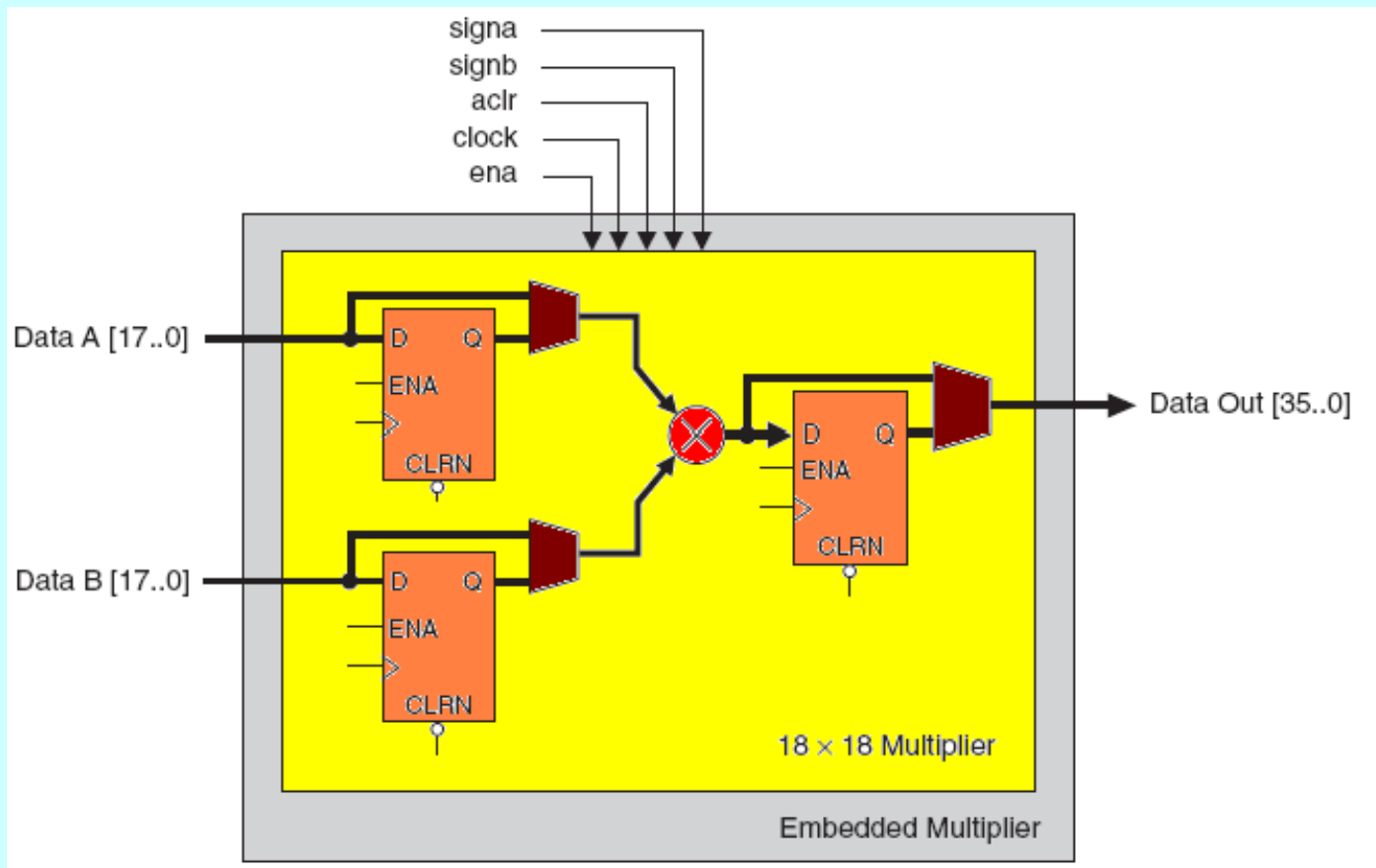
Do układu doprowadzone są następujące magistrale i sygnały sterujące:

- wejścia danych/czynników A i B
- bity znaku danych wejściowych signa i signb,
- szynę danych wyniku/iloczynu Data Out,
- sygnał zegarowy clock,
- wejście zezwolenia sygnałów zegarowych ena,
- asynchroniczny sygnał zerowania aclr buforów wejściowych i wyjściowych.

Operacja mnożenia mogą odbywać się na liczbach ze znakiem lub typu znak moduł

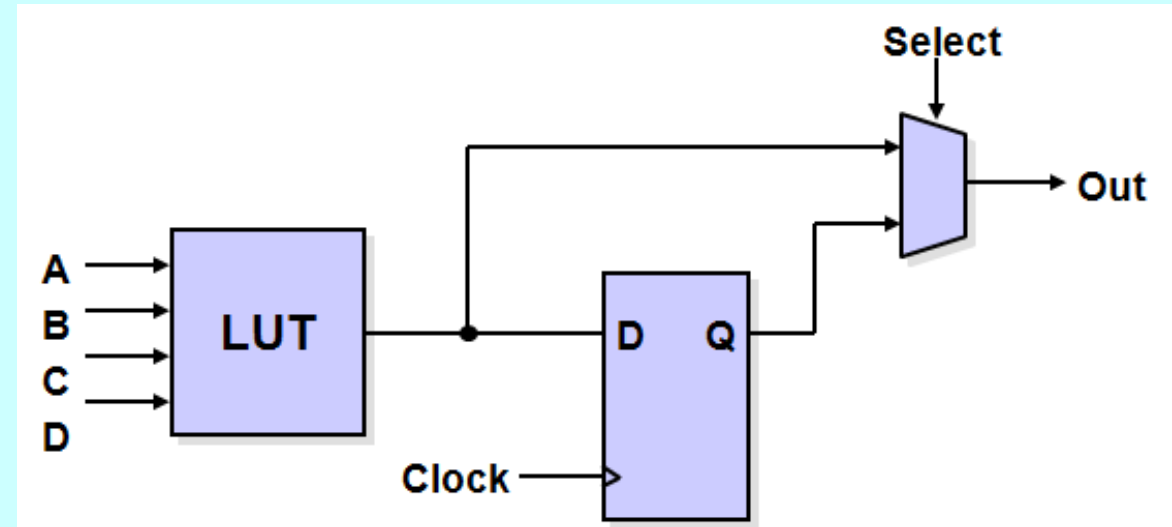
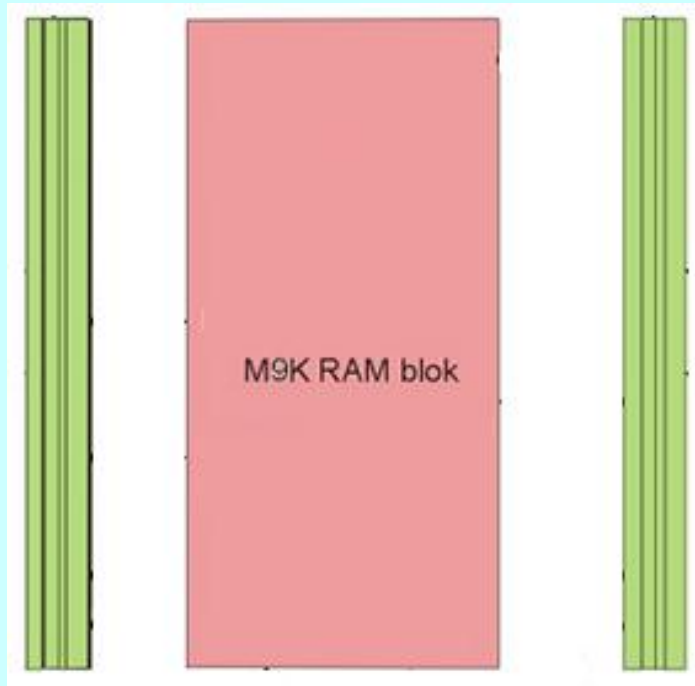


W cyclone IV moduł mnożnika wbudowanego można skonfigurować jako jednolity układ wykonujący operację mnożenia na dwóch czynnikach 18 - bitowych. Dane A i B mogą być zapisane w formacie ze znakiem, bez znaku lub kombinowanym – decydują o tym wartości sygnałów *signa* i *signb*. Znaki danych wejściowych można zmieniać dynamicznie w czasie pracy. Można też podzielić na 2 mnożniki 9-bitowe.



W używanym w czasie laboratorium układzie Cyclone IV, można syntezerować pamięci wewnętrzne. Pamięci układów programowalnych mogą implementowane w dwóch obszarach FPGA:

- w wewnętrznych dedykowanych blokach pamięci – np. M9K Cyclone IV (M4K, M514, itp.)
- w matrycy logicznej w przerzutnikach makrokomórek – tzw. pamięć rozproszona/podręczna.

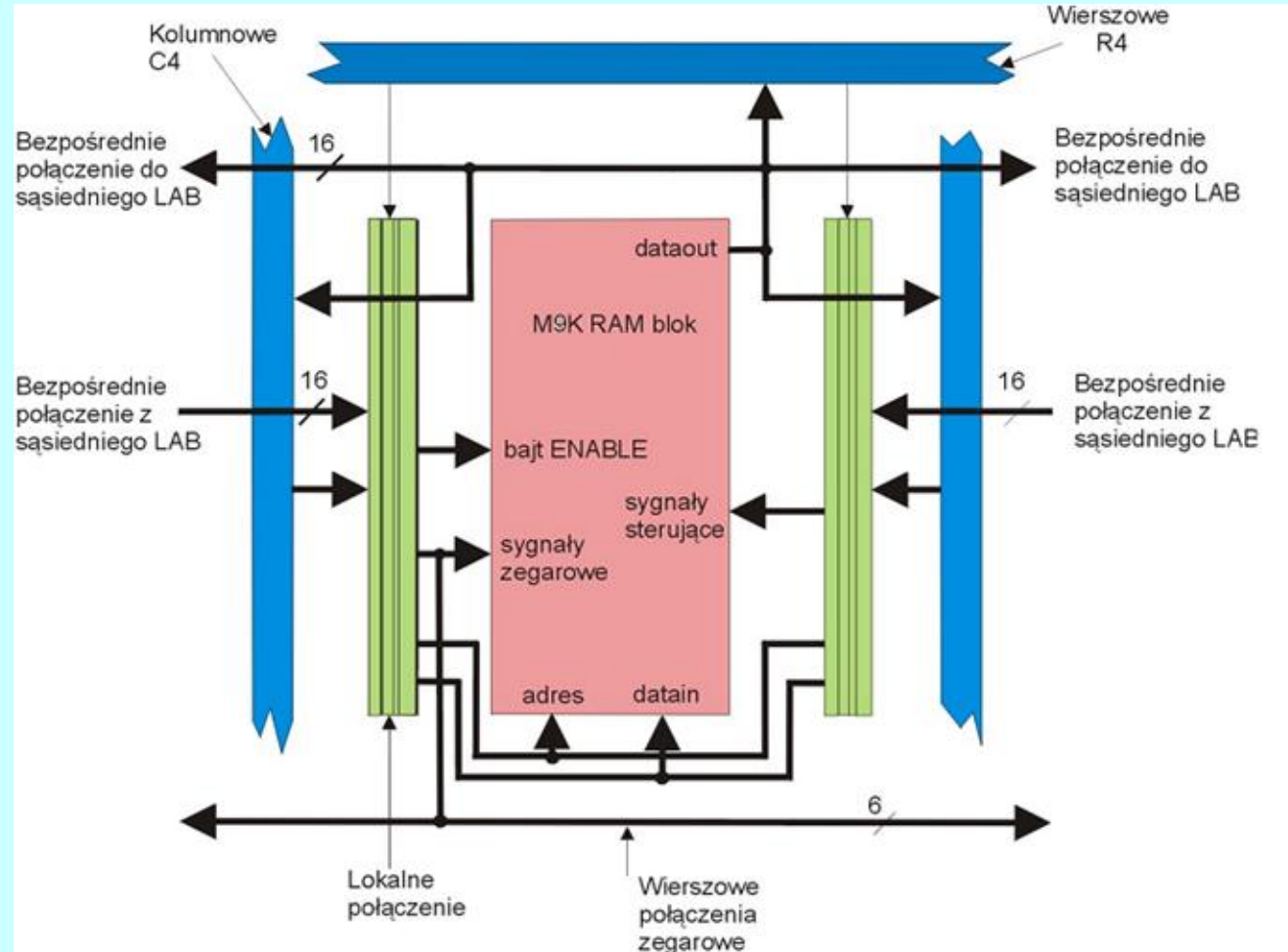


Pamięci rozproszone są szybkie, dopasowane pod względem wielkości do konkretnych potrzeb

- ich pojemność jest mniejsza niż w blokach – np. 115.000 bitów,
- łączna pojemność pamięci blokowej w Cyclone IV może wynieść nawet 3.800.000 bitów.

Blok pamięci wewnętrznej jest osadzony w sieci sygnałowej i w sieci zegarowej FPGA:

- posiada on dwie własne magistrale połączeń wewnętrznych (local interconnect),
- łączy się z globalną siecią połączeń wierszowych i kolumnowych,
- może on bezpośrednio łączyć się z blokami logicznymi LAB matrycy FPGA,



Wbudowane w Cyclone IV bloki M9K pozwalają budować rodzaje pamięci:

- RAM,
- FIFO,
- ROM,
- rejestry przesuwne.

Bloki M9K charakteryzują się następującymi właściwościami:

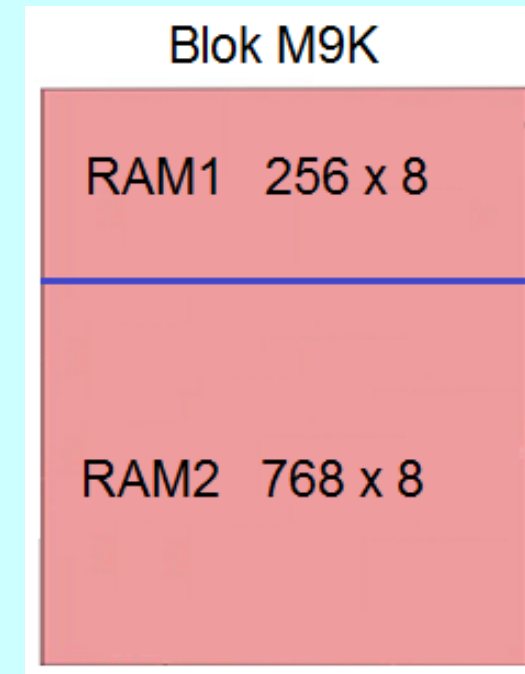
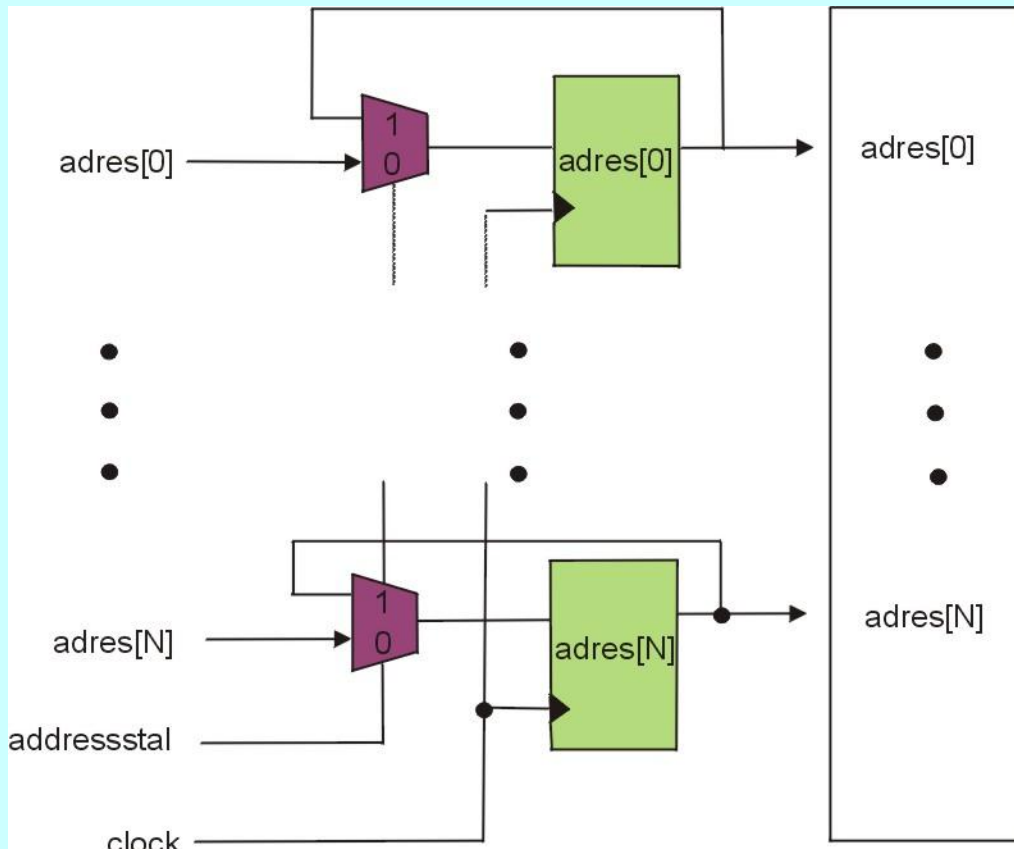
- umożliwiają syntezę 2 4.5 kbit RAM o częstotliwości pracy do 315 MHz,
- pojedynczy blok posiada pojemność 8.192 bitów (plus bity parzystości),
- możliwa jest pełna praca dwuportowa RAM i obsługa dwu-zegarowa,
- możliwy jest mechanizm plikowego inicjowania pamięci RAM i ROM,
- dostępne jest maskowanie części danych podczas zapisu do RAM,
- wszystkie rodzaje pamięci w M9K są synchroniczne – wymagają zegara.

Bloki posiadają bajty maskujące **byte enebles** – które przesłaniają zapis słowa danych. Domyślnie byteena przyjmuje wartość high i jego bity są skojarzone z fragmentami danych. Np. w przypadku pamięci 16-bitowej, gdy byteena = 01, wówczas tylko bity [7..0] słowa pamięci mogą być nadpisywane, zaś bity [15..8] będą niedostępne/maskowane.



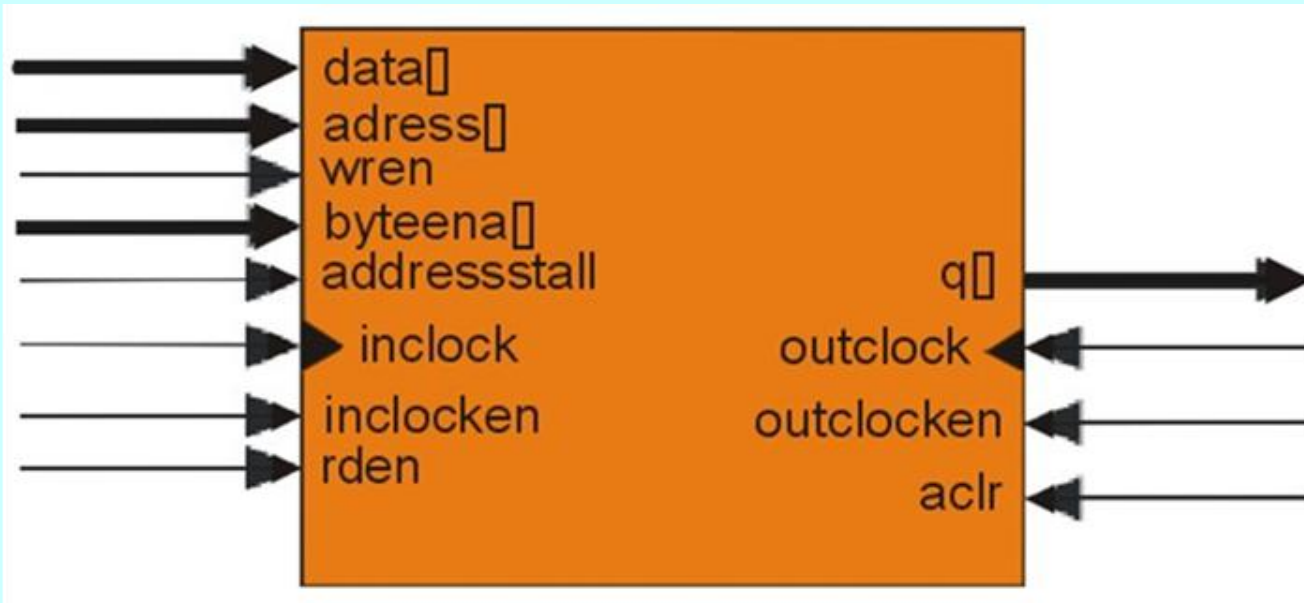
W trybie pakietowym można w pojedynczym bloku implementować dwie niezależne pamięci. Warunkami jest aby pojemności pamięci składowych nie przekraczały całkowitej pojemności M9K. Każda z pamięci musi pracować w trybie single clock a długość słowa danych w pamięciach składowych nie przekracza 18 bitów.

W FPGA można spowalniać ich obsługę wykorzystując mechanizm blokowania adresu. Bieżący adres można zablokować dopóki sygnał **addressstal** = 1, niezależnie od sygnału zegarowego. Realizuje to rejestr podtrzymujący szynę adresową w kolejnych taktach zegarowych, jeżeli addressstal jest aktywny.



Synteza pamięci RAM w FPGA obejmuje 3 konfiguracje: jednoportową, dwuportową prostą i dwuportową pełną. Pamięć RAM w trybie jednoportowym zawiera następujące sygnały:

- magistralę danych wejściowych **data[]**;
- magistralę adresową **adress[]**;
- sygnały sterujące (aktywny poziom wysoki) zapisem danych **wren** i odczytem danych **rden** ;
- sygnał zegarowy **inclock** inicjujący zboczem narastającym zapis danych z wejścia data[];
- magistralę danych wyjściowych **q[]**;
- sygnał zegarowy synchronizujący odczyt danych **outclock**.



Ponadto pamięć może zawierać:

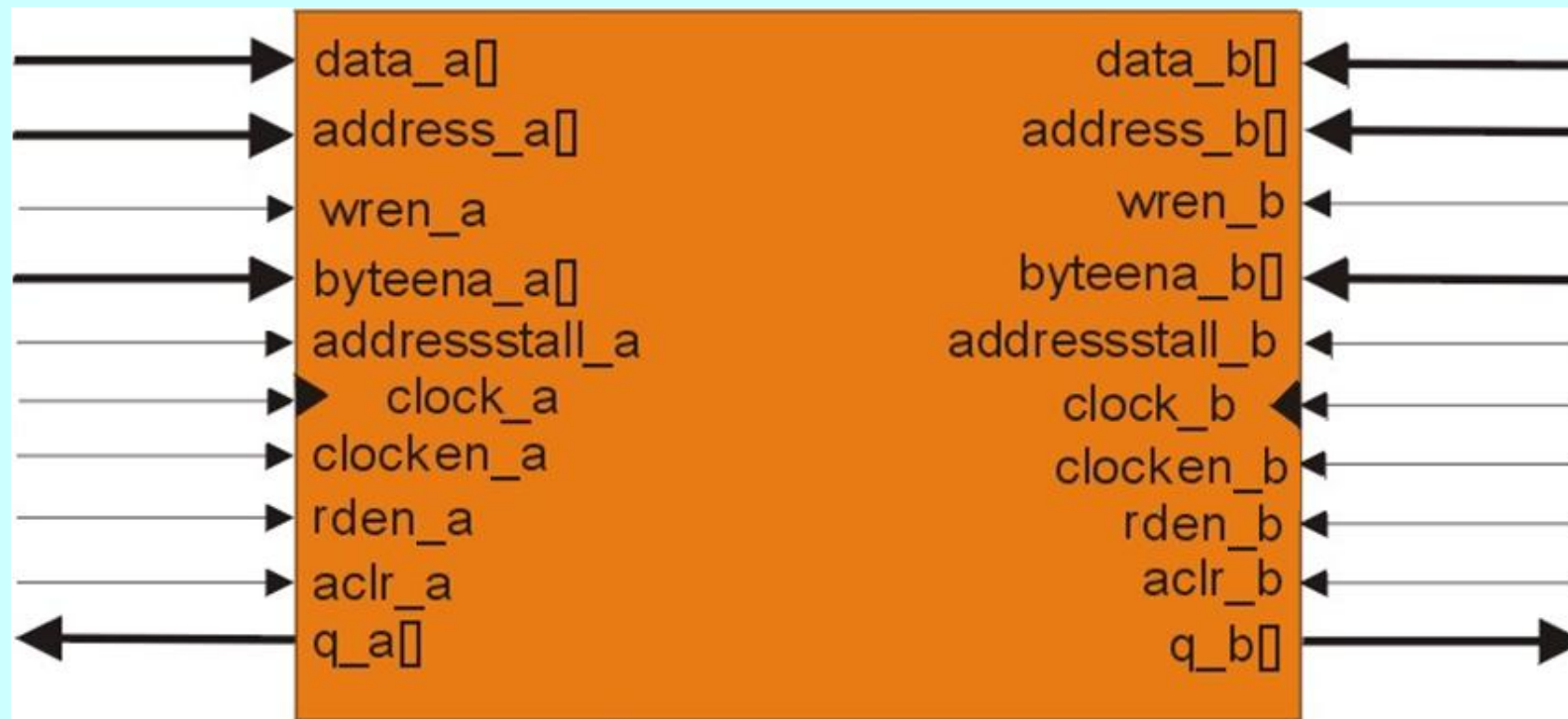
- sygnał maskujący zapisywane dane **byteena**;
- sygnał zatrzaśnięcia adresu **addressstall**;
- sygnały zezwolenia **inclocken/outclocken**;
- sygnał zerujący bufor wy **aclr**.

W trybie dwuportowym pełnym matryca M9K zachowuje jak obszar, na który nakładają się dwie niezależne pamięci (oznaczone jako a i b) jednoportowe, z których każda posiada własny komplet sygnałów.

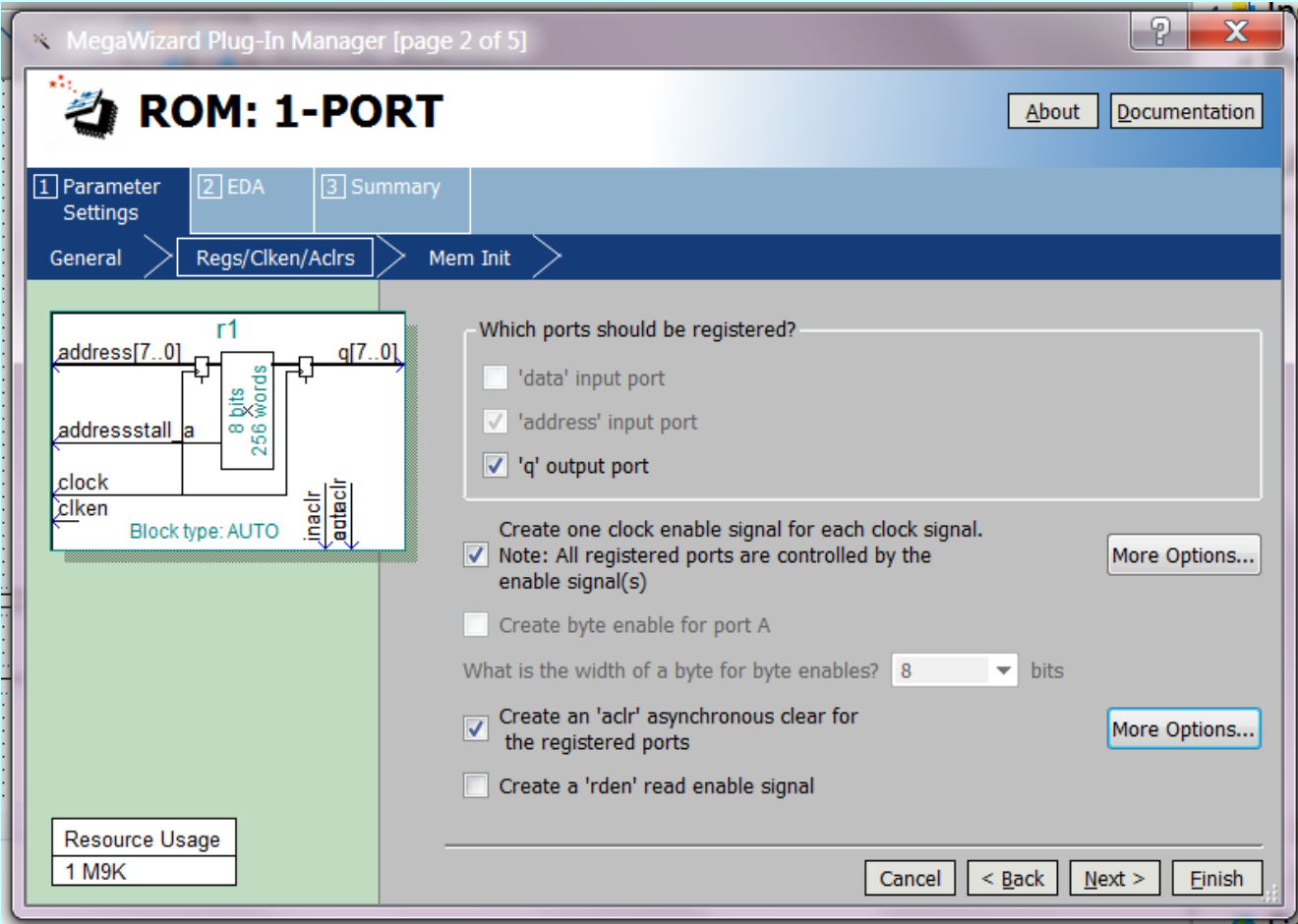
W szczególności tego samego obszaru matrycy M9K mogą odnosić się: dwa odczyty, dwa zapisy oraz zapis i odczyt – co więcej można je obsługiwać sygnałami zegarowymi o różnej częstotliwości.

Sama konstrukcja pamięci w tym trybie nie rozwiązuje konfliktu dostępu, który może się zdarzyć przy próbie zapisu do tej samej komórki obie pamięci, co może skutkować zapisem nieznanych danych.

Problem musi rozwiązać projektant systemowo, sterując w odpowiedni sposób pamięciami.



ROM jest to najprostszym trybem M9K. Operacja odczytu z ROMA polega na podaniu adresu w pamięci oraz impulsu zegarowego. W Quartus II modelem ROM jest IP **lpm_rom**, której zawartość jest inicjowana plikiem **.mif** lub **.hex** podczas konfiguracji FPGA. Na podobnych zasadach jak w pamięci RAM - można dodatkowo dodawać sygnały sterujące: **addressstall**, **clken**, **inaclr** oraz **outaclr** – zerowania rejestrów wyjściowych.



Mif1.mif*									
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	

Pamięć FIFO jest strukturą o dostępie kolejkowym, tzn. że o położeniu danych wewnątrz i na wyjściu pamięci nie decyduje adres lecz kolejność ich wprowadzania i kolejność odczytu. Ten rodzaj pamięci znajduje zastosowanie w pakietowej transmisji danych, gdy ze względu na szybkość trzeba zapisać lub wysłać cały blok danych.

Funkcje portów FIFO są następujące:

clock - sygnał zegarowy,

data[] - wejście danych do FIFO,

wrreq - żądanie zapisu – nie
ustawiać gdy aktywne full,

rdreq - żądanie odczytu - nie ustawiać
gdy aktywna flaga empty,

sclr - synchroniczne zerowanie stanu,

aclr - asynchroniczne zerowanie,

q[] - dane wyjściowe z FIFO,

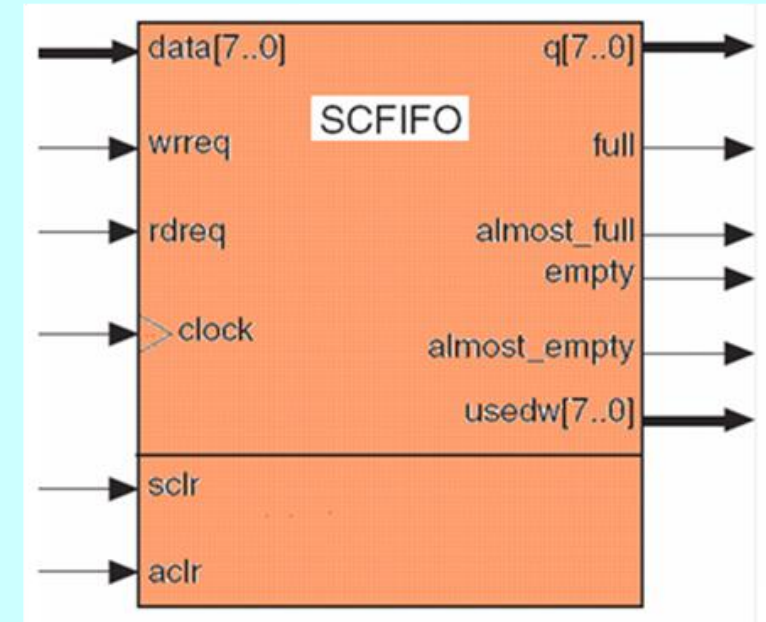
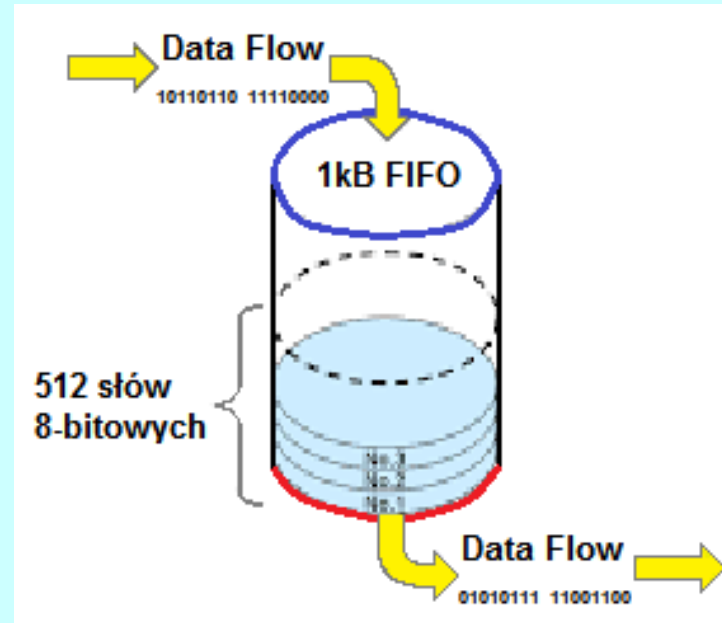
full - flaga zapelnienia FIFO – gdy
full = 1 nie zapisywać do pamięci,

empty - FIFO jest pusta - w czytać,

almost_full - flaga poprzedzająca full,

almost_empty - wyprzedza empty,

usedw[] - liczba słów w FIFO.



Zaliczenie cząstkowe wykładu nr 7 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

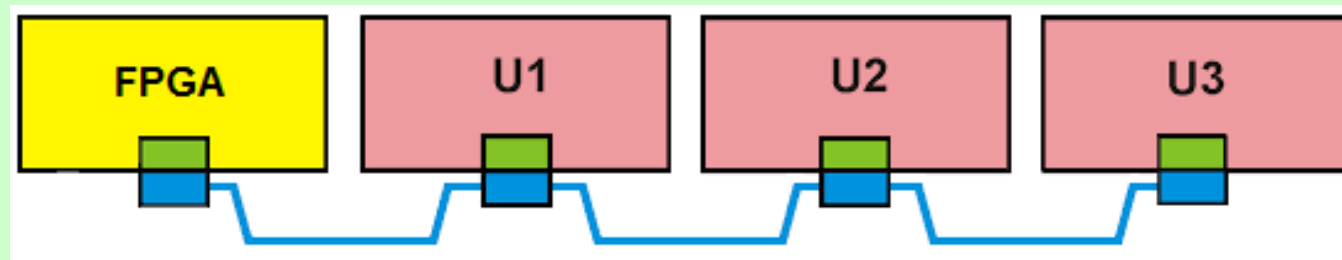
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

Transmisja – przesyłanie danych, między nadajnikiem a odbiornikiem w sposób zrozumiały dla obu stron za pomocą określonej drogi/kanалу transmisyjnej. Może istnieć więcej niż 1 odbiornik nadawanej informacji.

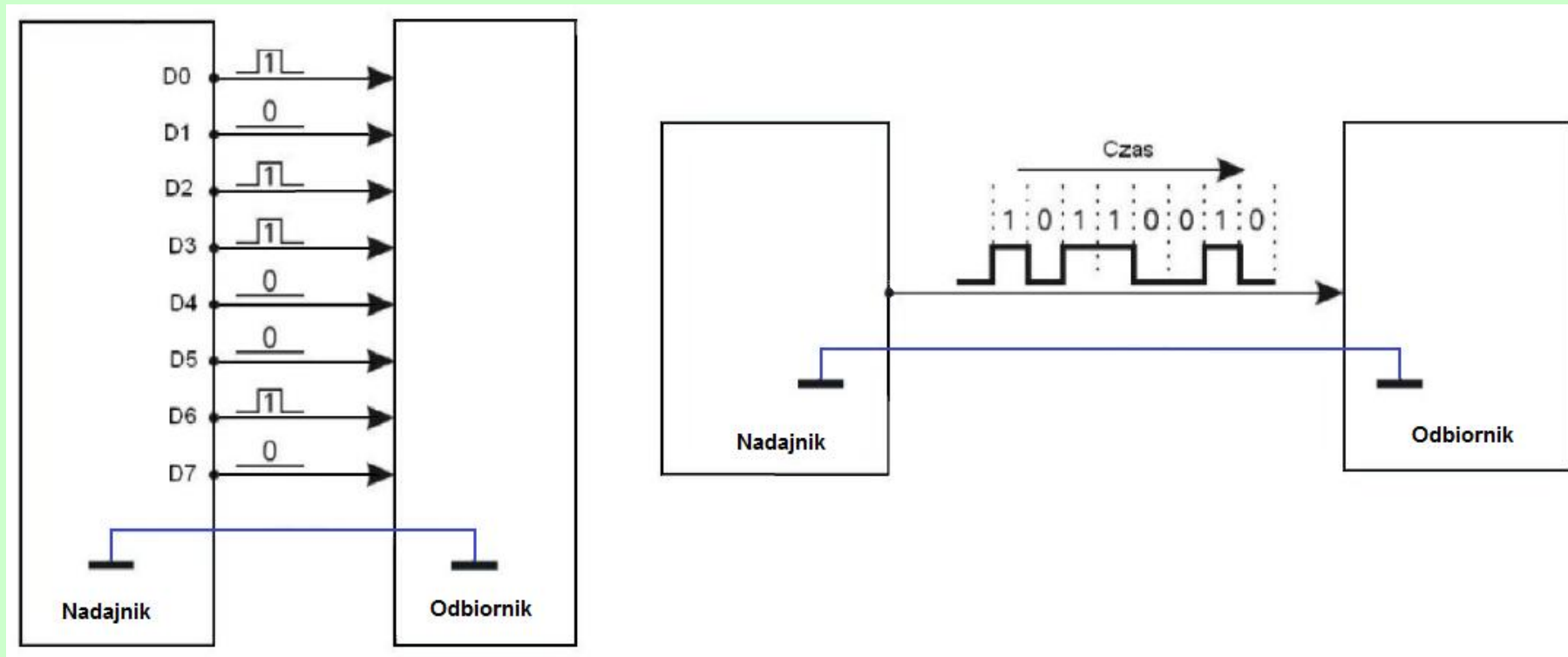


Interfejs – oznacza połączenie między stronami transmisji. Jest to zbiór elementów mechanicznych, elektrycznych i oprogramowania, koniecznych do wymiany informacji między urządzeniami. Stąd interfejs może sprzętowy i/lub programowy, z kolei magistrala stanowi warstwę sprzętową.

Ze względu na sposób przesyłania danych dzielimy rodzaj transmisji i interfejsy na:

- równoległe,
- szeregowe.

Porównanie transmisji równoległej i szeregowej:



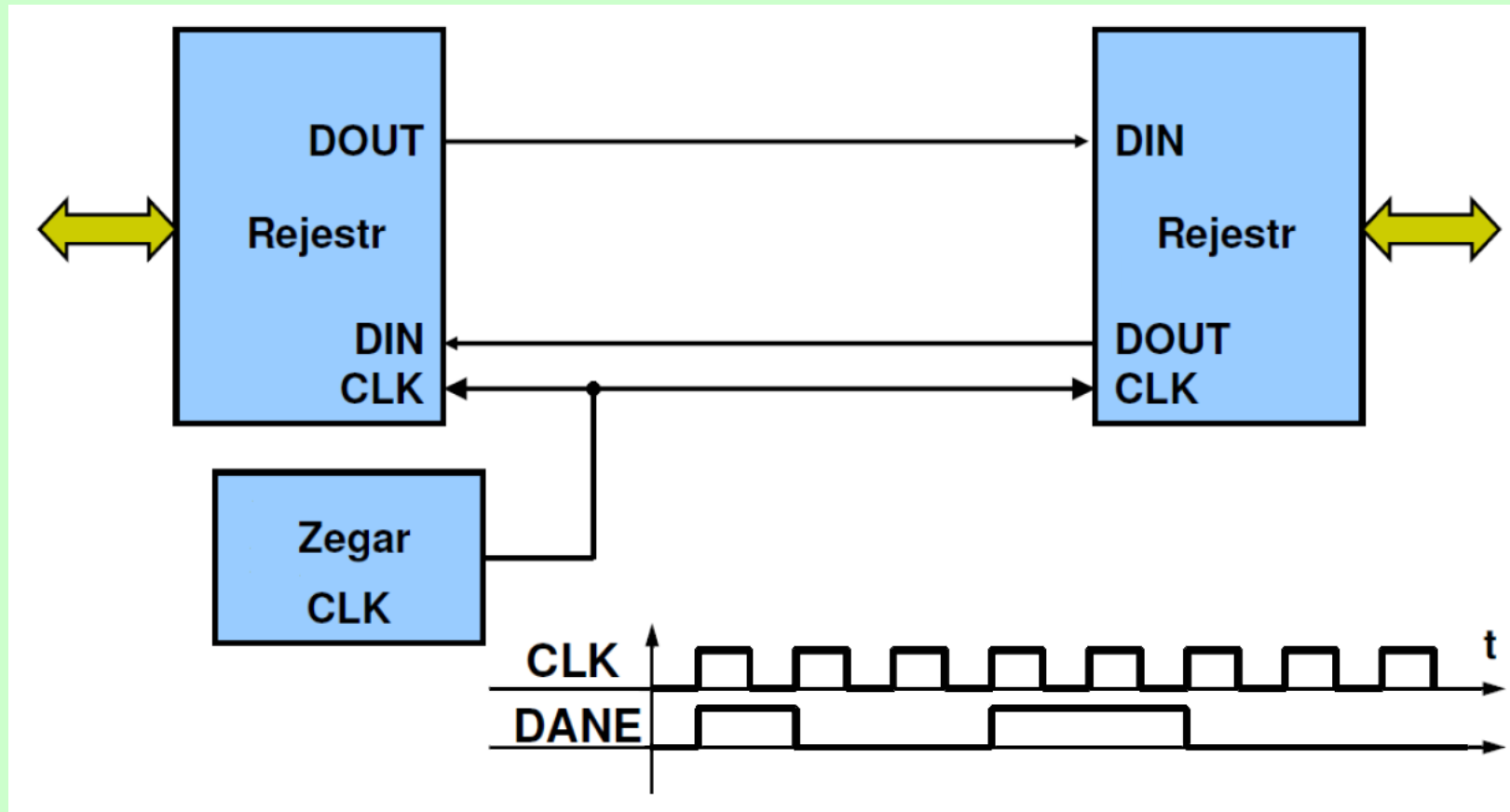
W transmisji równoległej w jednym takcie zegarowym przesyłamy cały bajt / słowo – a w nim wszystkie bity jednocześnie. Dzięki temu ten rodzaj transmisji posiada większą przepustowość danych, wadą jest większa liczba połączeń co komplikuje konstrukcję hardware.

W transmisji szeregowej dane są przesyłane bit po bicie, jeden bit w jednym takcie zegarowym. Najczęściej, najpierw przesyłany jest najstarszy bit danych (MSB) a potem kolejne. Wadą transmisji szeregowej jest mniejsza przepustowość, dla tej samej częstotliwości zegarowej. Interfejs szeregowy jest prostszy konstrukcyjnie.

Ze względu na sposób rozpoczynania/synchronizacji transmisji w czasie, rozróżniamy szeregową transmisję:

- synchroniczną,
- asynchroniczną.

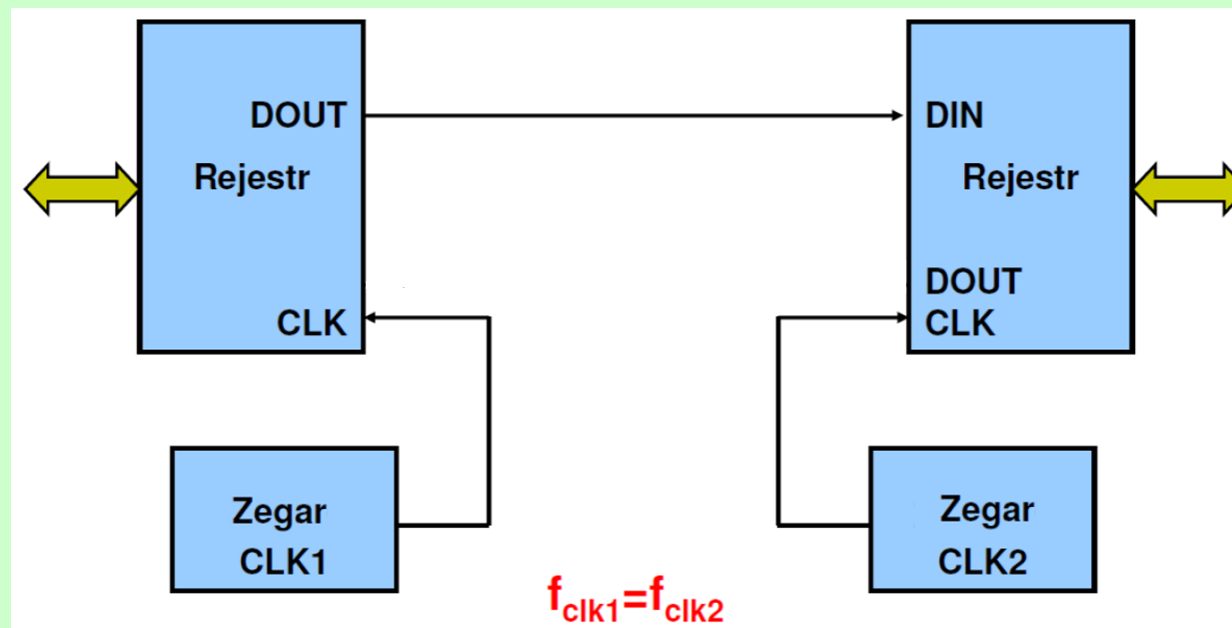
W transmisji synchronicznej momenty zapisu danych wyznacza sygnał zegarowy, ten sam sygnał doprowadzony zarówno do nadajnika jak i odbiornika.



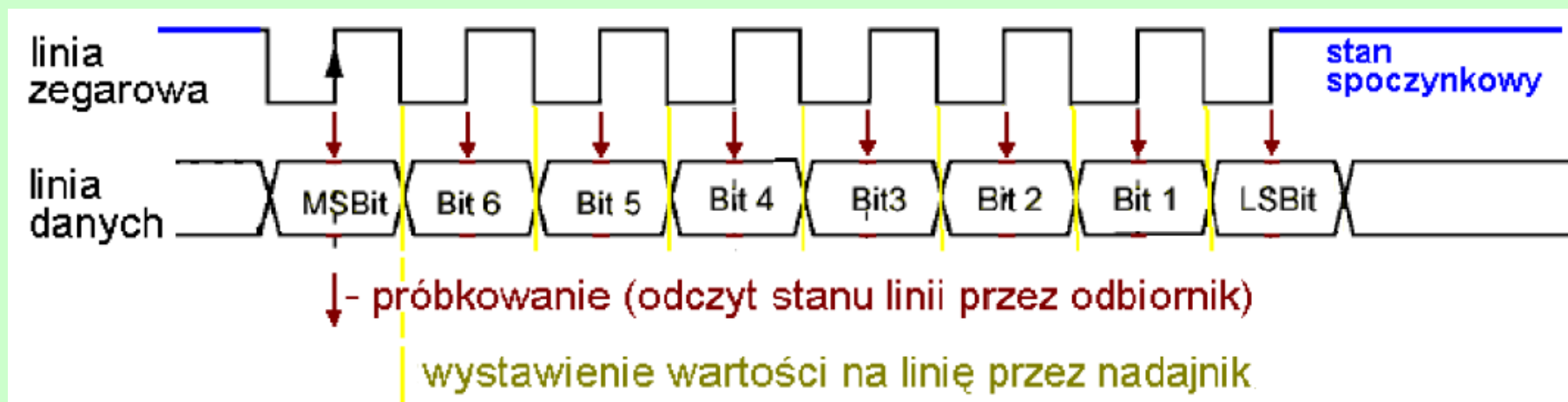
W transmisji asynchronicznej sygnał zegarowy nadajnika nie jest wprost przesyłany do odbiornika a jedynie dane szeregowo. Stąd sygnał zegarowy po stronie odbiorczej musi być odtworzony a początek transmisji rozpoznany na podstawie sygnałów na linii danych. Natomiast obie strony – nadajnik i odbiornik „wiedzą” z jaką częstotliwością są wysyłane dane, mają ustaloną wartość częstotliwości zegara.

Stąd odbiornik jest w stanie odtworzyć sygnał zegarowy bez konieczności dodatkowej linii transmisyjnej. Transmisja asynchroniczna może być jednokierunkowa (simplex) lub dwukierunkowa (duplex).

Problemem jest zsynchronizowanie dwóch niezależnych generatorów zegarowych, po stronie nadajnika i odbiornika, tak aby oba sygnały nie były przesunięte w fazie względem siebie – nie wystąpił efekt **Clock Skew**.



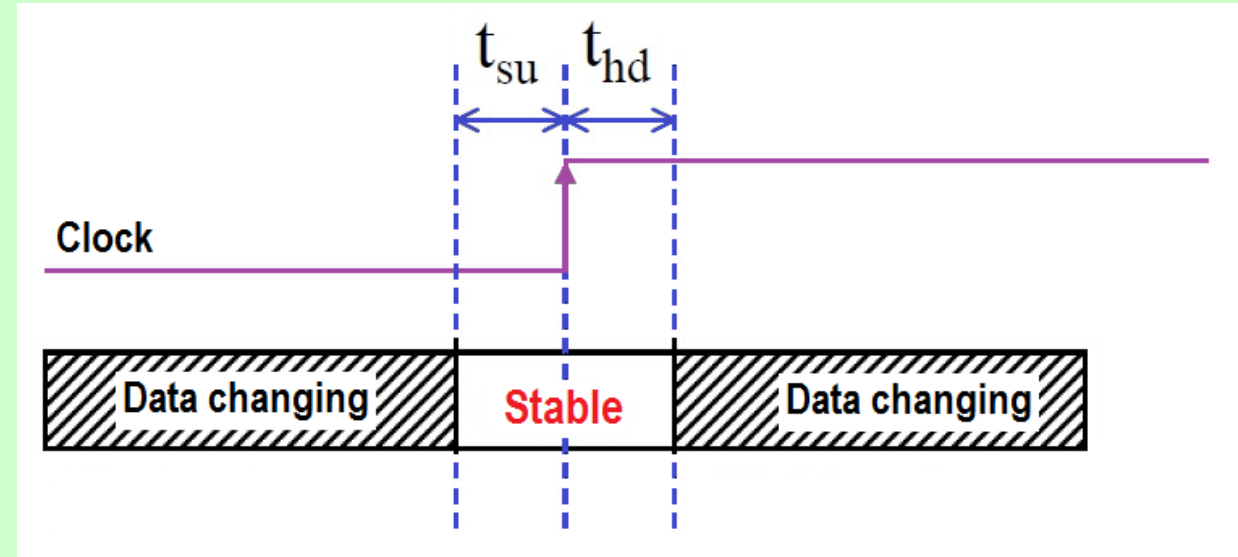
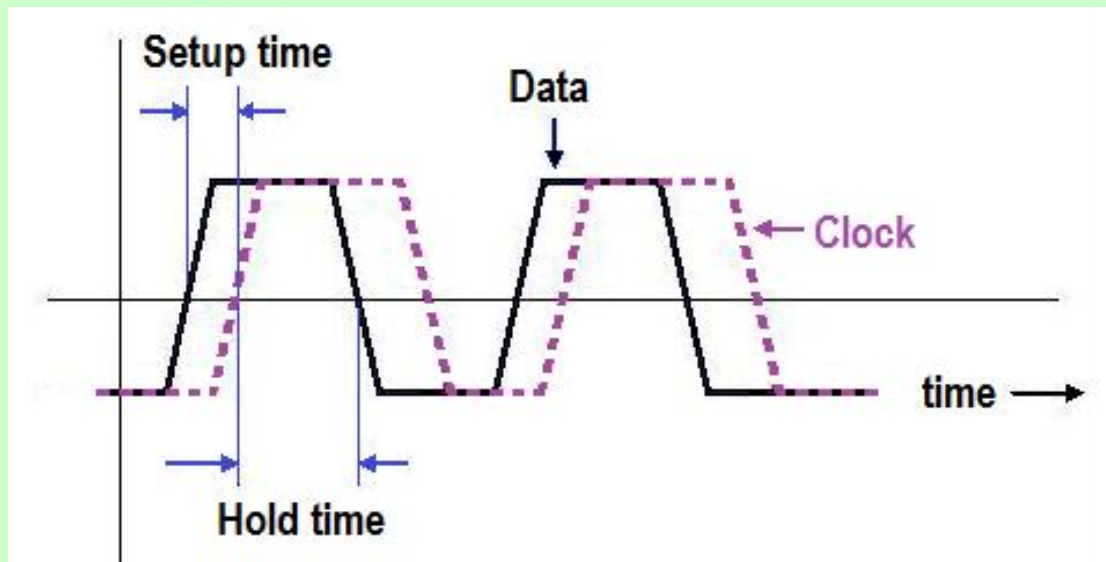
Jeżeli sygnał zegarowy odbiornika nie jest zsynchronizowany z zegarem nadajnika, niemożliwy jest prawidłowy odczyt przesyłanych danych – gdyż położenie zboczy sygnału zegarowego względem przesyłanych danych będzie niewłaściwe. Zapewnienie właściwego położenia/pozycjonowania impulsów zegarowych względem przesyłanych lub odbieranych danych jest warunkiem szeregowej transmisji synchronicznej i asynchronicznej.



Drugą ważną zasadą niezawodności transmisji szeregowej – jest stabilność danych w czasie występowania aktywnego zbocza sygnału zegarowego, tego zbocza, które zapisuje/zatrzaszkuje dane lub wysyła dane. Jest to zazwyczaj narastające zbocze sygnału zegarowego. Dlatego zmiana wartości/poziomu bitu na linii danych nie **może następować** w tym momencie.

Dlatego w technicznych danych katalogowych elementów i układów, podane są dwa parametry, które definiują zależności czasowe pomiędzy zmianą danych a zmianą sygnału zegarowego. Są nimi:

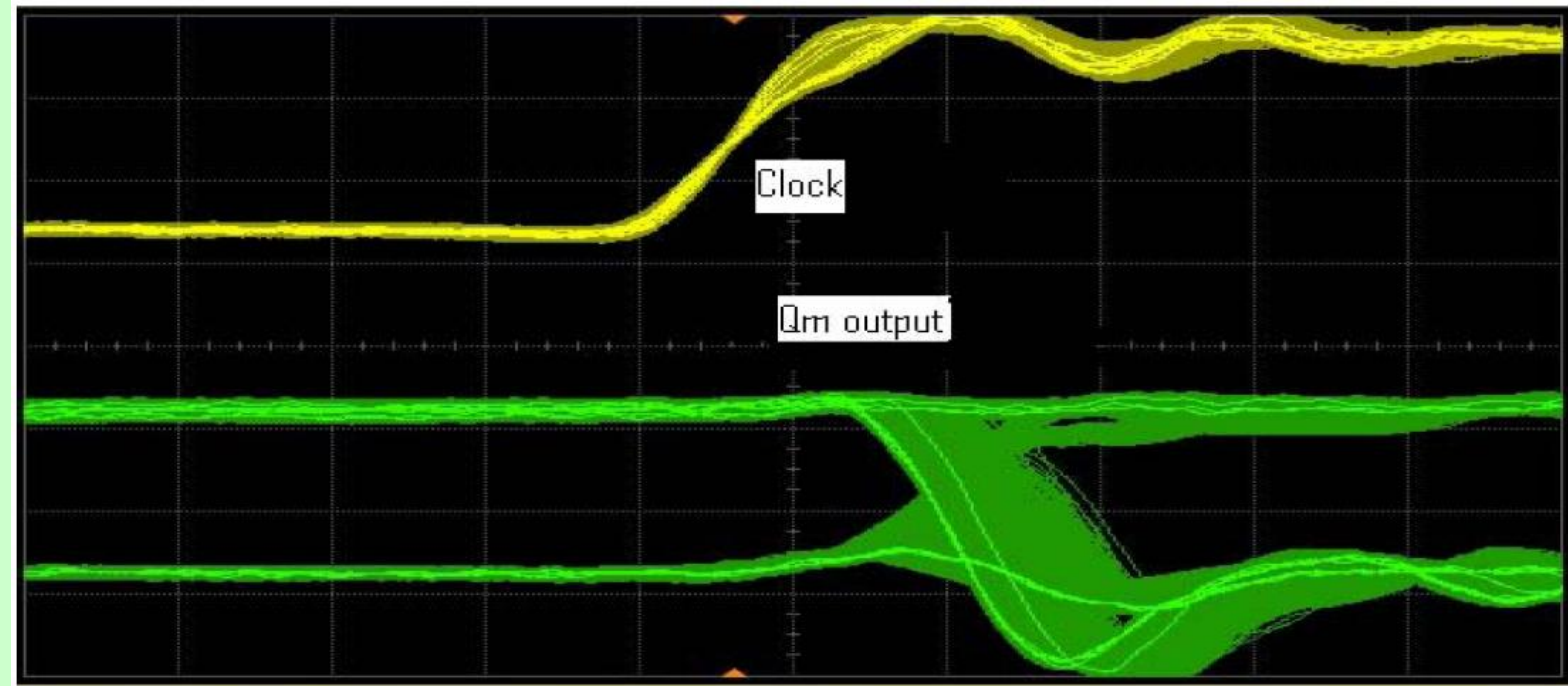
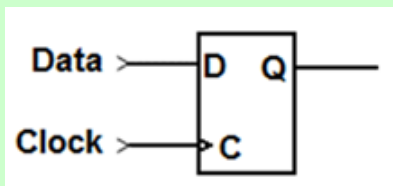
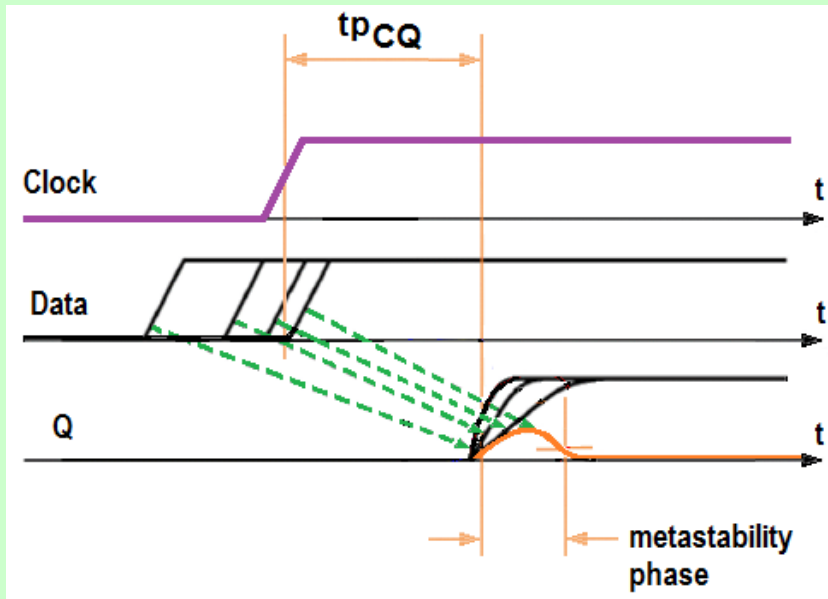
- czas ustawiania/wyprzedzenia/**Setup time** – oznaczany jako t_{su} , jest to odcinek czasu przed aktywnym zboczem sygnału zegarowego, kiedy dane muszą być pozostawać są stabilne - nie mogą zmieniać się;
- czas podtrzymania/utrzymania/**Hold time** – oznaczany jako t_{hd} , gdy po zaniknięciu aktywnego zbocza sygnału zegarowego dane jeszcze muszą pozostać stabilne – niezmiennie.



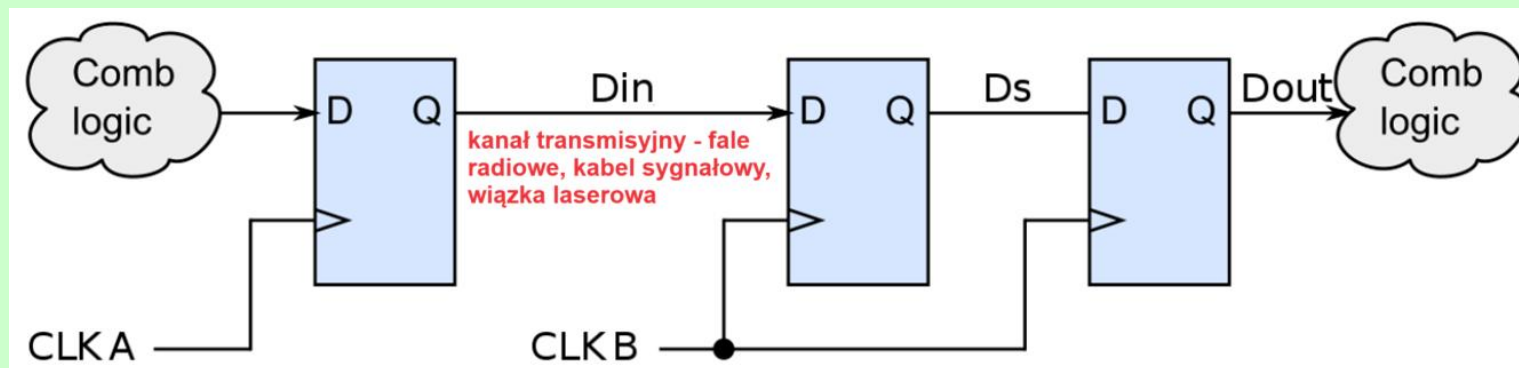
Niezachowanie właściwych relacji pomiędzy zmianą danych a zmianą sygnału zegarowego, może prowadzić do powstania **stanu metastabilnego**, w którym dane wejściowe zmieniają w czasie narastającego zbocza sygnału zegarowego, taki niestabilny stan przenosi się na wyjście układu.

W konsekwencji ustalanie końcowej wartości sygnału czytanego/wyjściowego może trwać długo (czas t_{pCQ}), a w skrajnym przypadku może prowadzić do zapamiętania innej/poprzedniej wartości niż bieżący bit wejściowy.

Poniższe przypadki pokazują sytuację gdy nie jest zachowany czas wyprzedzenia t_{su} :

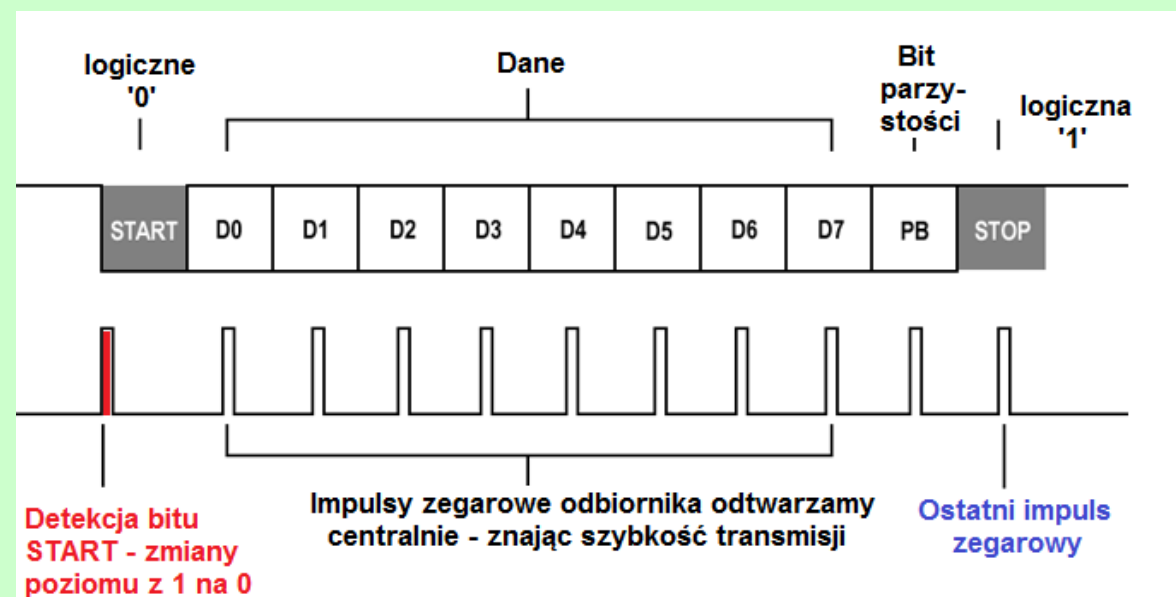


Efekt Clock Skew w transmisji asynchronicznej jest naturalnym źródłem metastabilności, gdzie ma miejsce taktowania układów niezależnymi i nie zsynchronizowanymi sygnałami zegarowymi.

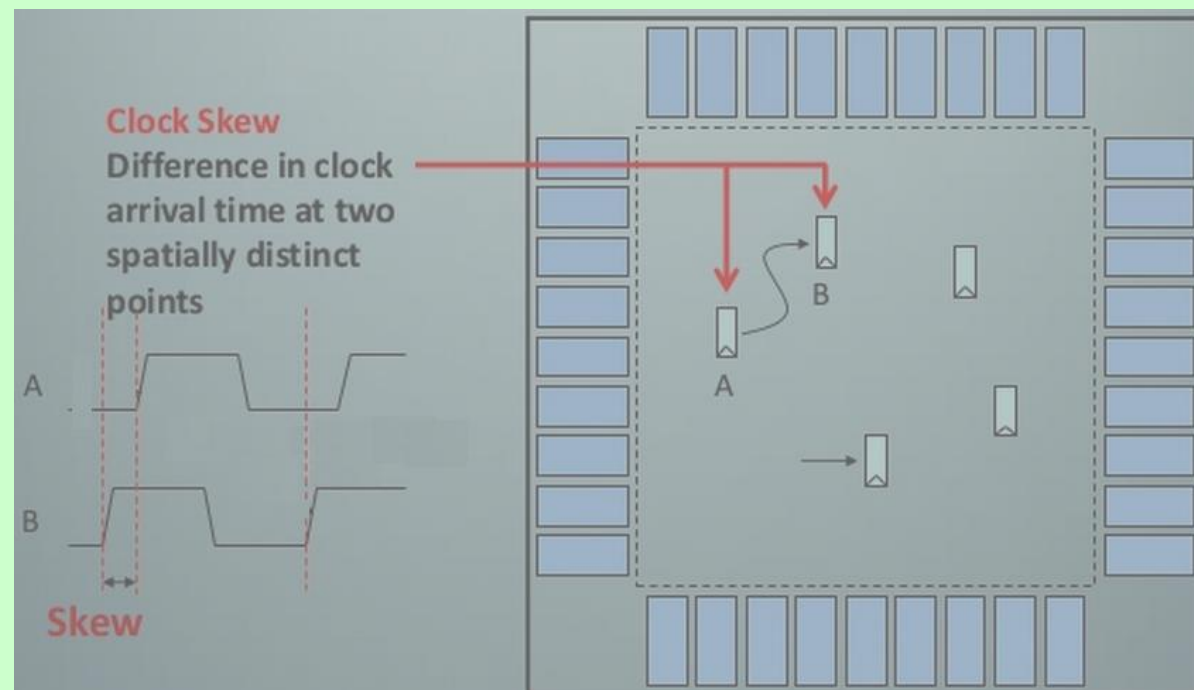
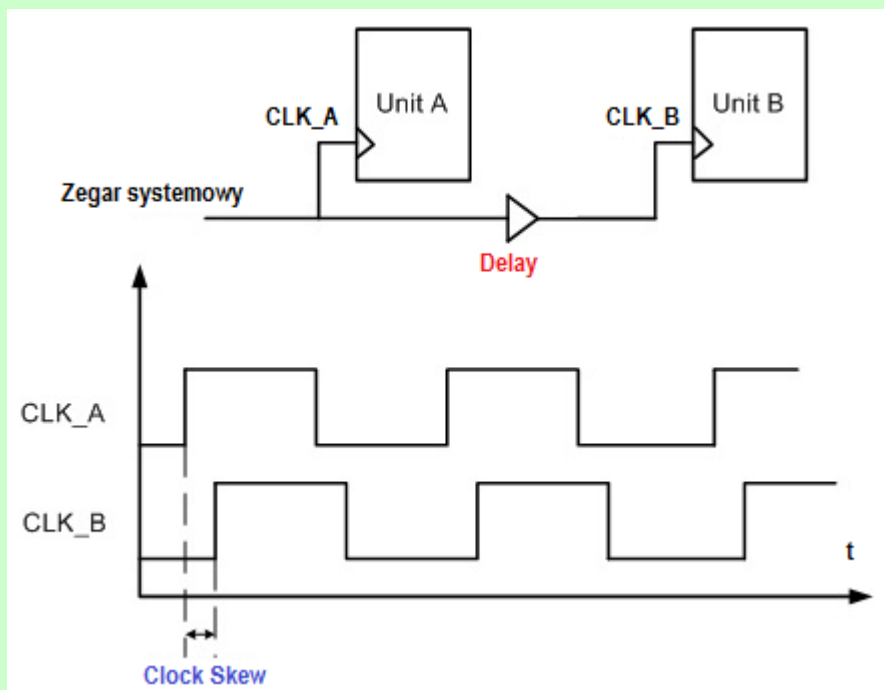


W transmisji asynchronicznej efekt jest łagodzony dzięki synchronizacji zegara w odbiorniku za pomocą sygnału START linii danych na początku każdej ramki.

Odbiornik po rozpoznaniu START koryguje moment uruchomienia generatora zegarowego po stronie odbiorczej. Po wykryciu poziomu niskiego na linii danych, układ sprawdza stabilność tego poziomu przez pół okresu zegara nadajnika, a następnie zaczyna generować impulsy zegarowe o znanej częstotliwości.



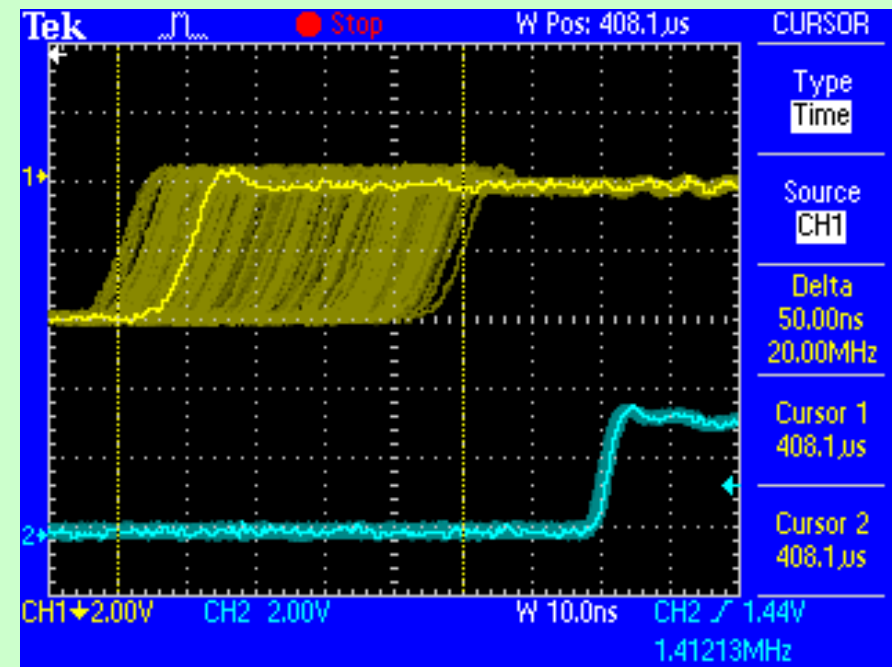
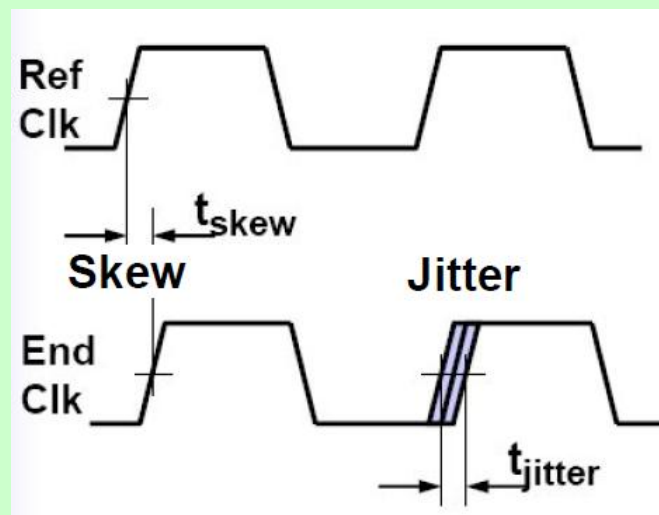
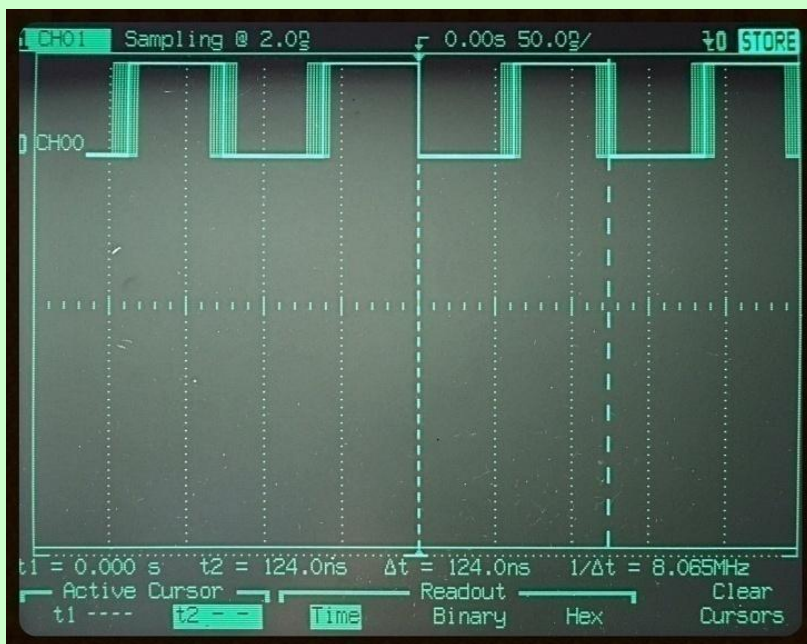
Z efektem Clock Skew spotykamy się też w układach synchronicznych, gdy sygnał zegarowy, wysyłany z tego samego źródła/punktu dociera do wejść zegarowych przerzutników układu cyfrowego w różnym czasie.



Zjawisko jest min. spowodowane:

- długością przewodów połączeniowych,
- wahaniami temperatury,
- sprzężeniami pojemnościowymi i różnicami w pojemności na wejściach zegarowych odbiorników,
- niedoskonałością struktury materiału. Efekt ten jest bardzo uciążliwy dla wysokich częstotliwości sygnału.

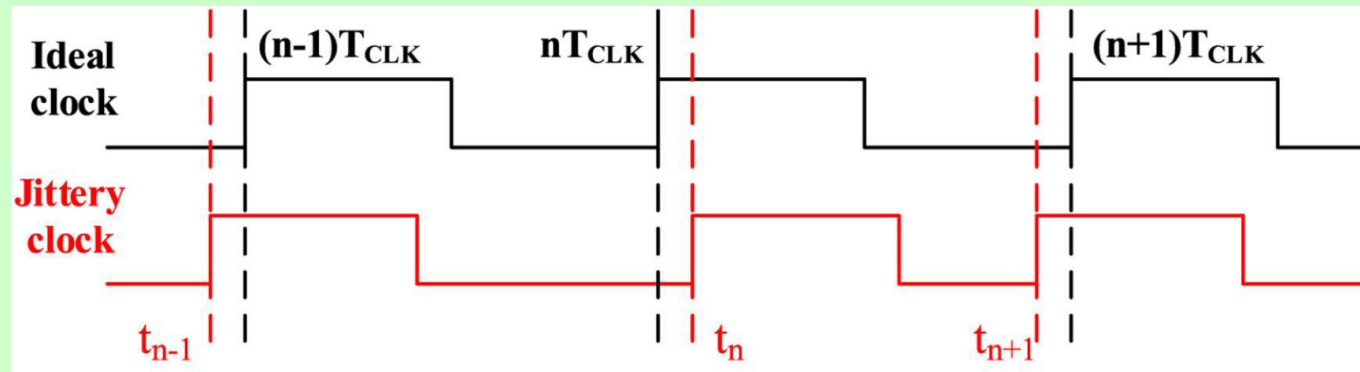
Podobny efekt jak niezachowanie czasów ustawiania i podtrzymania danych – może powodować **Clock Jitter**, czyli szumy fazy sygnału zegarowego. W wyniku tego zjawiska zbocze sygnału zegarowego nieznacznie się przemieszcza się w każdym okresie.



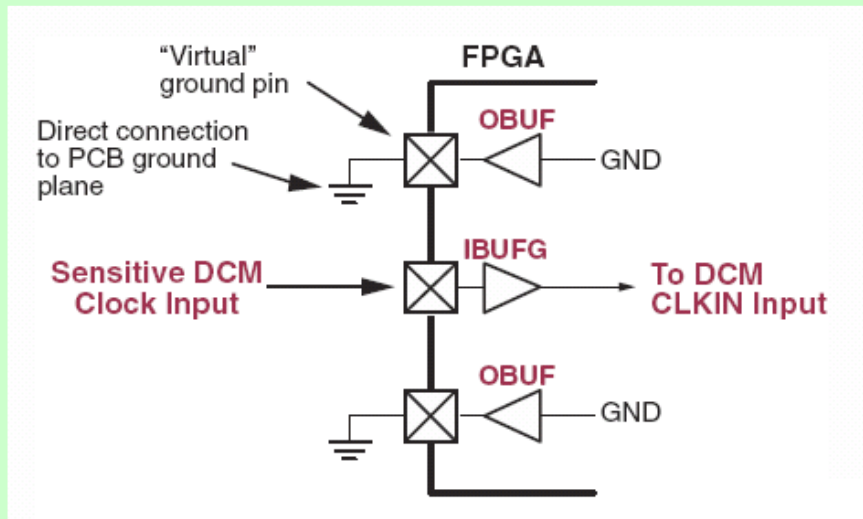
Szerokość przedziału drgań fazy zależy od: konstrukcji generatora, jakości ekranowania, jakości filtracji napięcia zasilającego, szumów własnych zastosowanych elementów, temperatury. Efekt jest dotkliwy i widoczny w zakresie wysokich częstotliwości.

Jitter zegara można dalej podzielić na jitter losowy i deterministyczny:

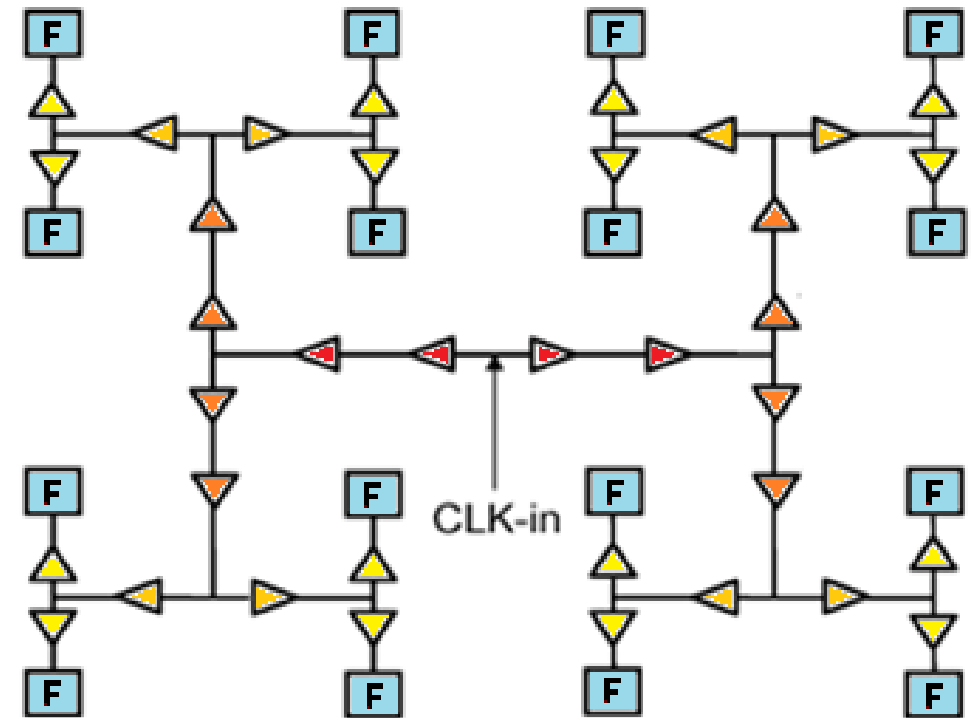
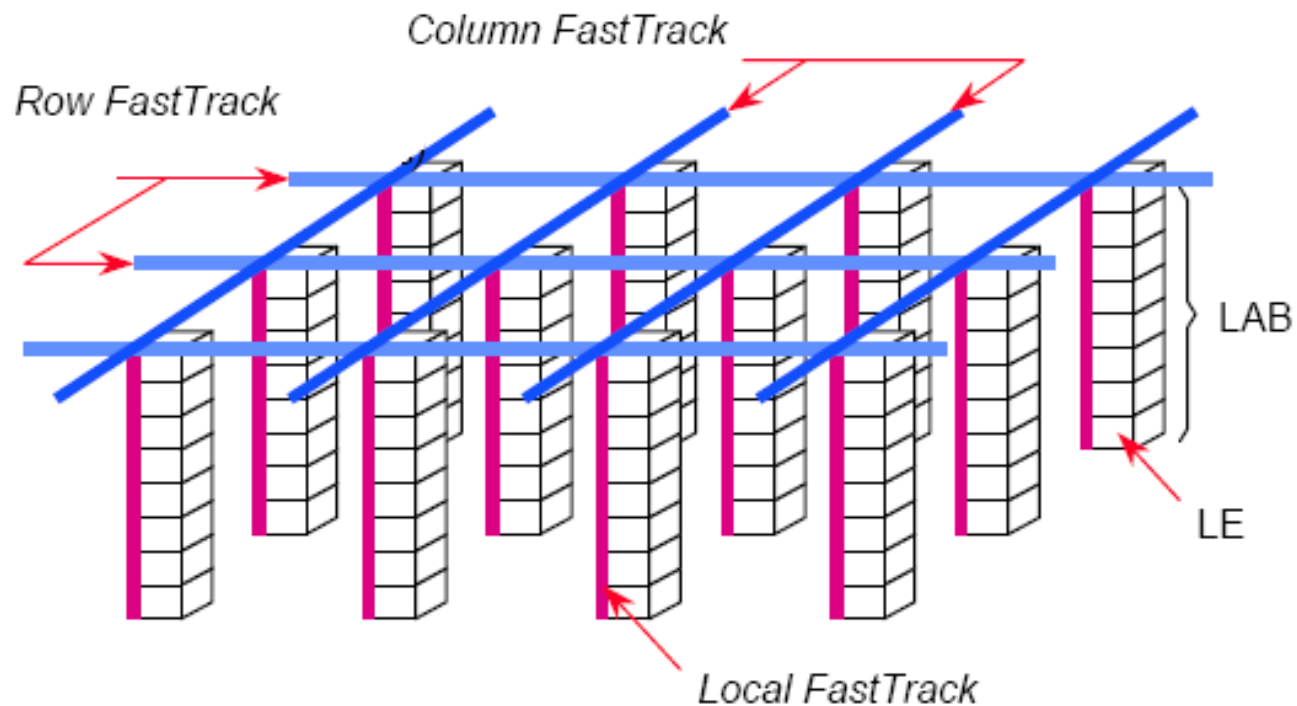
- losowy jitter pochodzi z wewnętrznego szumu termicznego obwodów dystrybucji zegara,
- deterministyczny jitter wynika ze zniekształceń cyklu pracy i szumu zasilania.



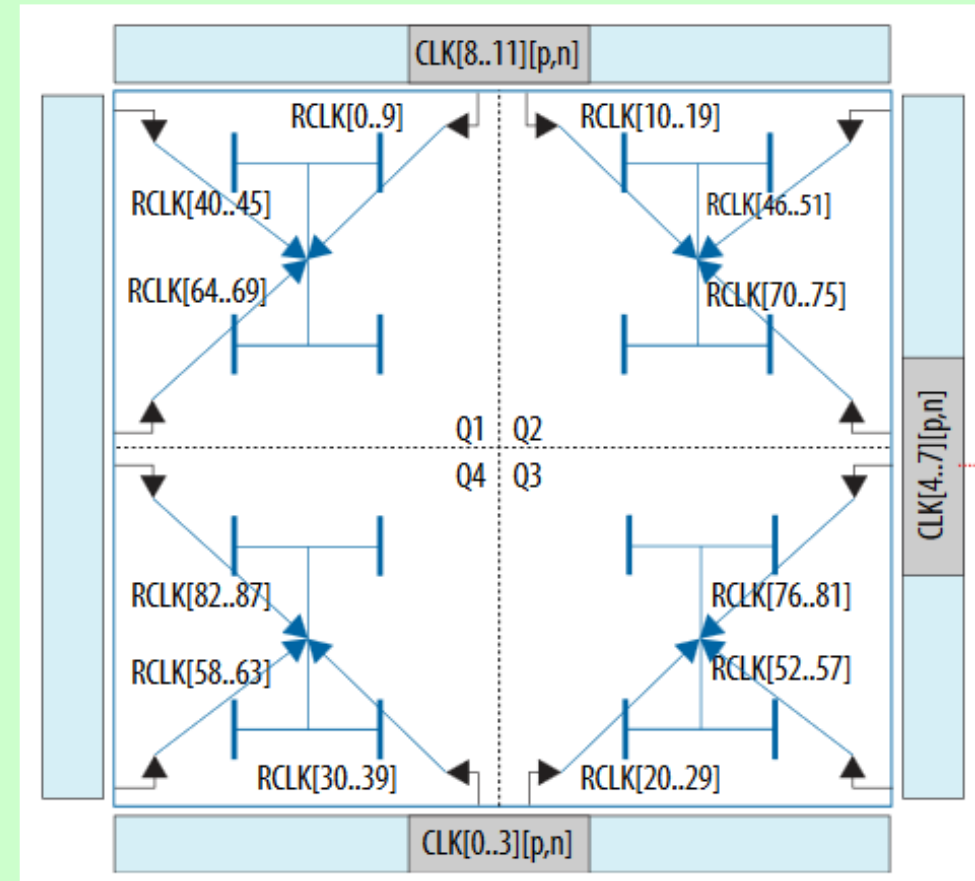
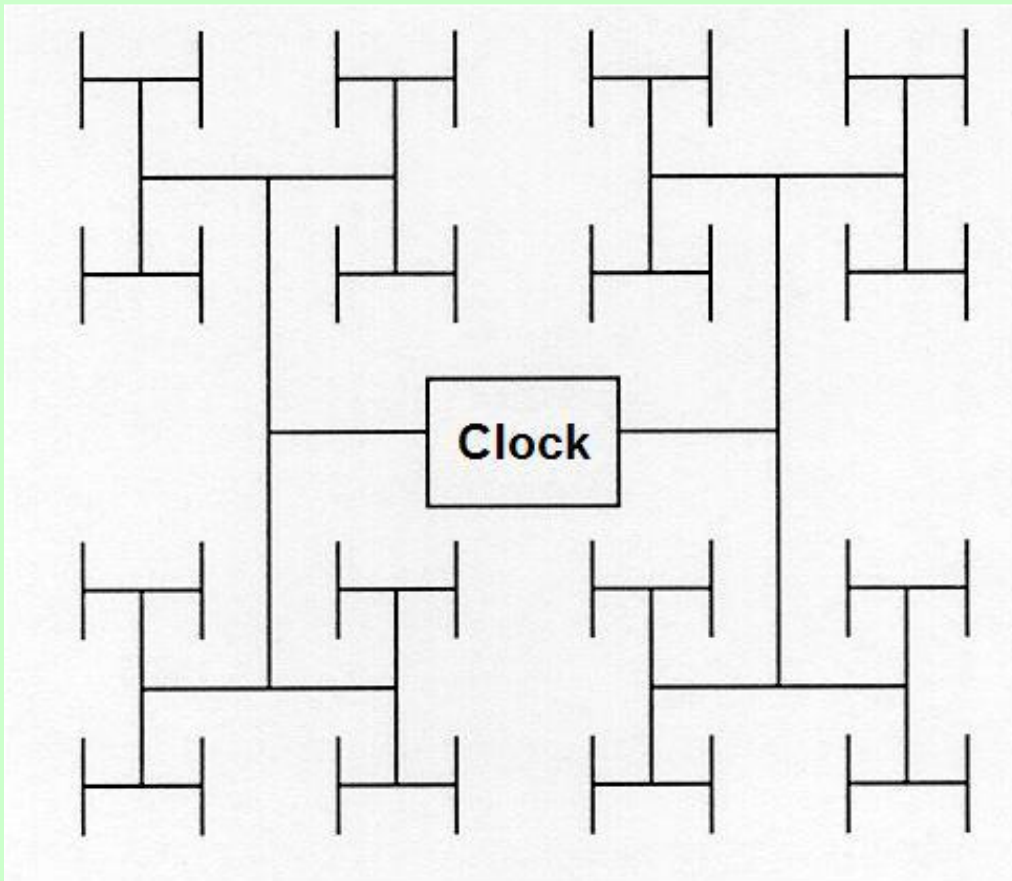
Ze względu na losowość zjawiska, jedyną skuteczną metodą ograniczenia jitter jest zastosowanie PLL w torze zegarowym i właściwe doprowadzenie sygnału do wnętrza.



Opisane zakłócenia sygnału zegarowego są szczególnie ważne w układach FPGA, gdzie znajdują się miliony elementów synchronizowanych sygnałem zegarowym. Z tego powodu w strukturach programowalnych rozdziela się ścieżki przepływu sygnałów przetwarzanych i sygnałów zegarowych. Sygnały danych przesyła się przez globalną sieć sygnałową (Global Multi Track Interconnect, rysunek z lewej strony), sygnały zegarowe przez globalną sieć zegarową (Global Clock Networks). Transmisja obu kategorii sygnałów następuje w niezależnych strukturach układu scalonego.

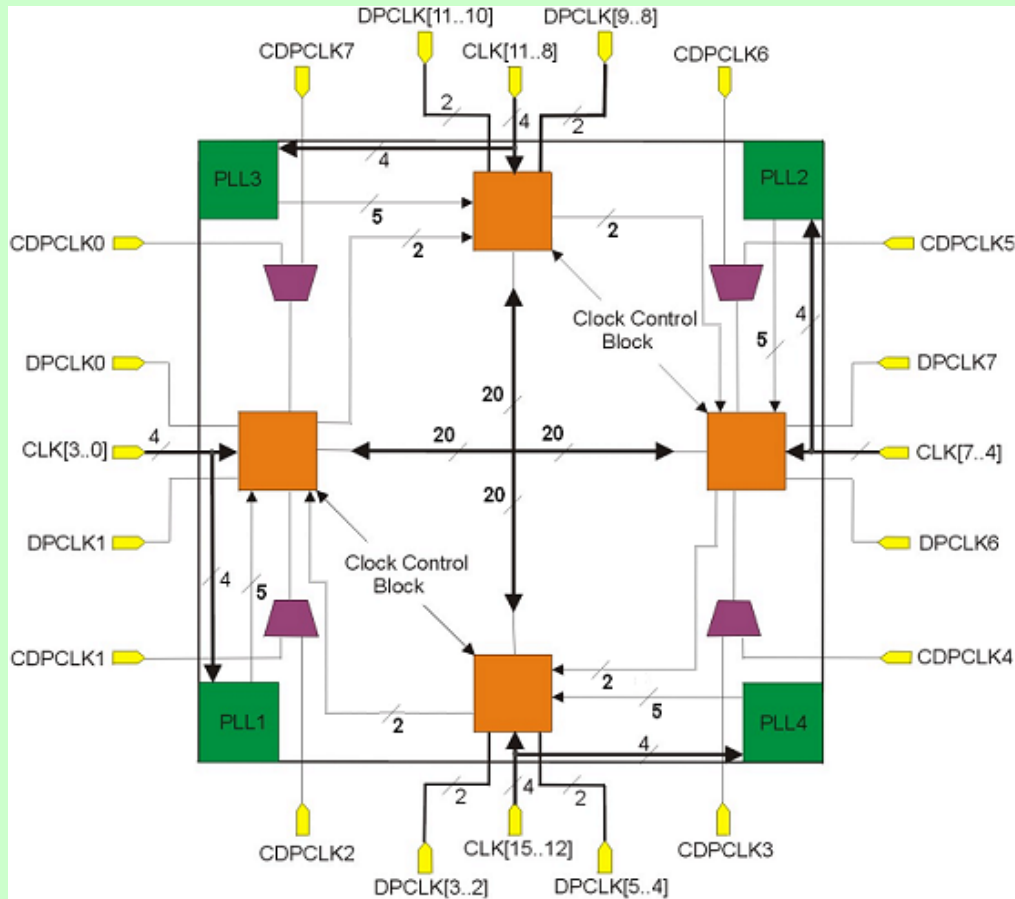


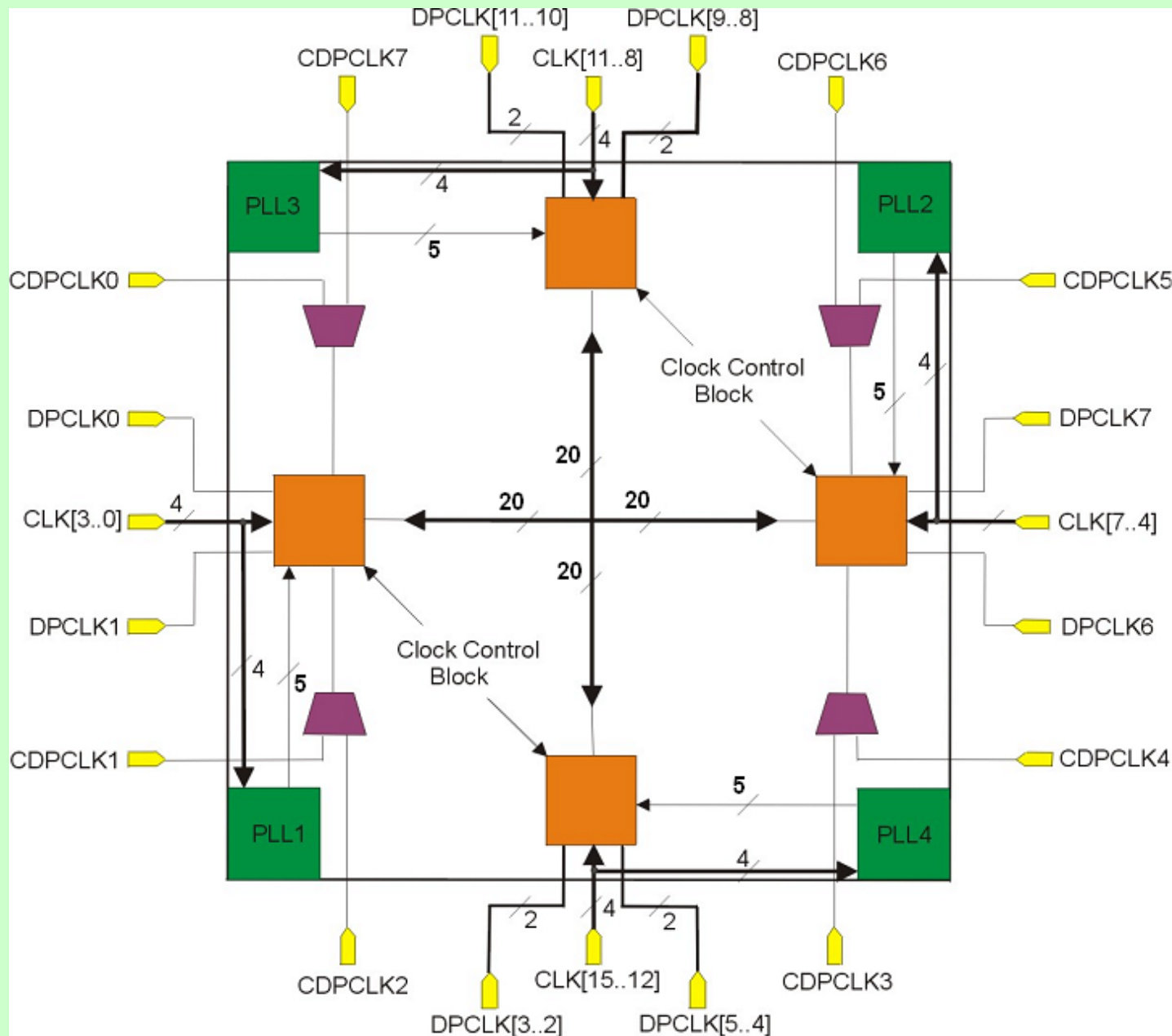
W zależności od wielkości struktury i producenta układów spotykane są dwie topologie rozsyłu sygnałów zegarowych. W mniejszych układach jest to sieć typu H tree (pokazana z lewej strony), gdzie pojedynczy sygnał może być rozsyłany w całej strukturze FPGA. W dużych układach matrycę logiczną FPGA dzieli się na segmenty, w których można rozsyłać niezależne sygnały zegarowe – czyli sieć typu X tree. W badanym w laboratorium układzie stosowana jest topologia typu H tree.



Sieć zegarowa FPGA Cyclone IV E zawiera 16 dedykowanych wejść zegarowych oznaczonych jako CLK[15..0]. Wewnątrz struktury znajduje się 20 globalnych sieci zegarowych GCLK, rozprawdzających przetworzone sygnały zegarowe. Każda GCLK z użyciem bloków selekcji (Control Block) pobiera sygnał zegarowy z:

- PLL,
- pinów CLK[] lub podwójnego znaczenia DPCLK[] lub CDCCLK[] lub z logiki wewnętrznej.

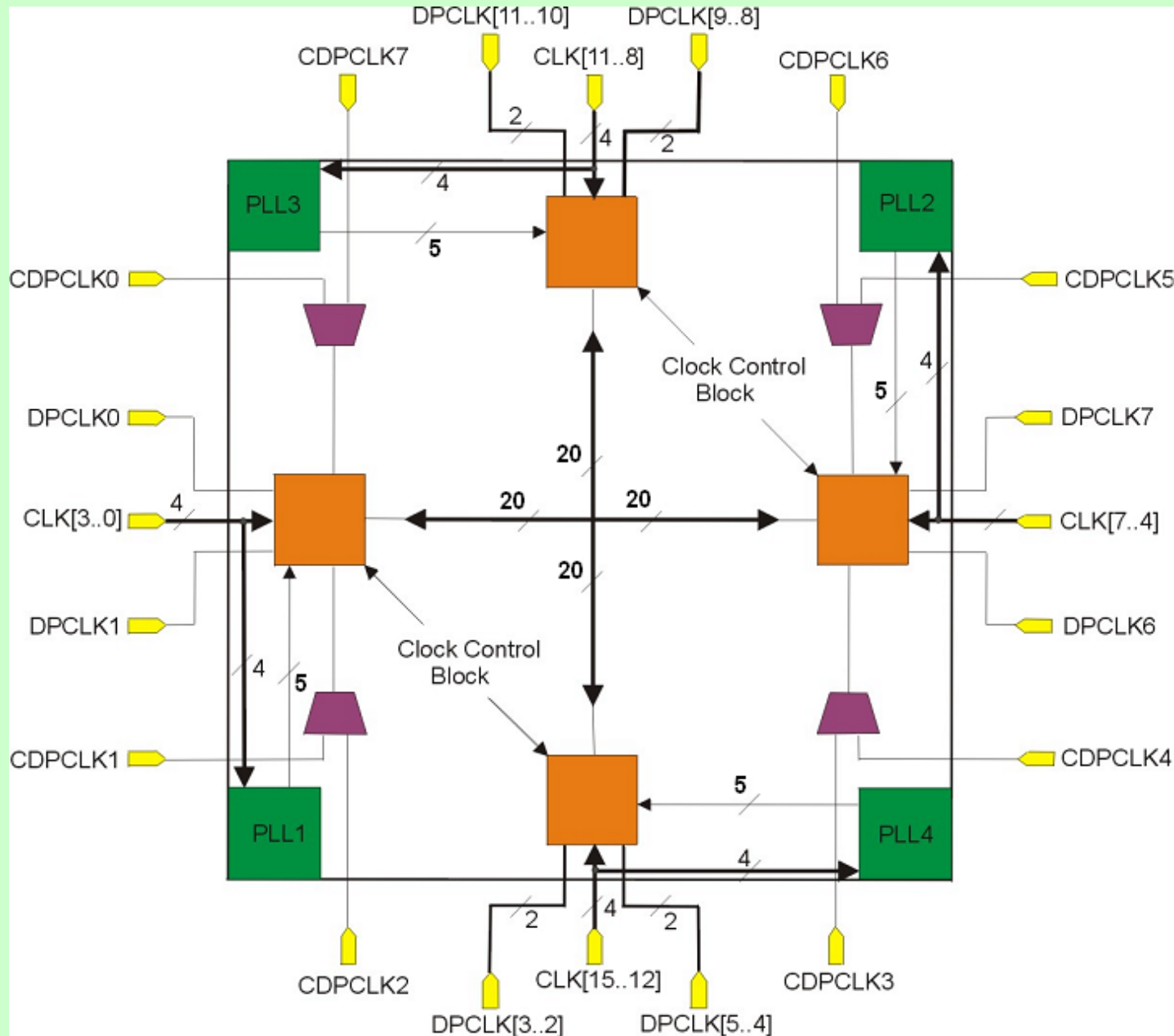




Skrajne piny podwójnego znaczenia CDPCLK[] są multipleksowane przed przyłączeniem do bloku sterującego. Sygnały z tych wejść są programowo opóźniane w stosunku do DPCLK.

Globalna sieć zegarowa doprowadza sygnały zegarowe i sterujące do:

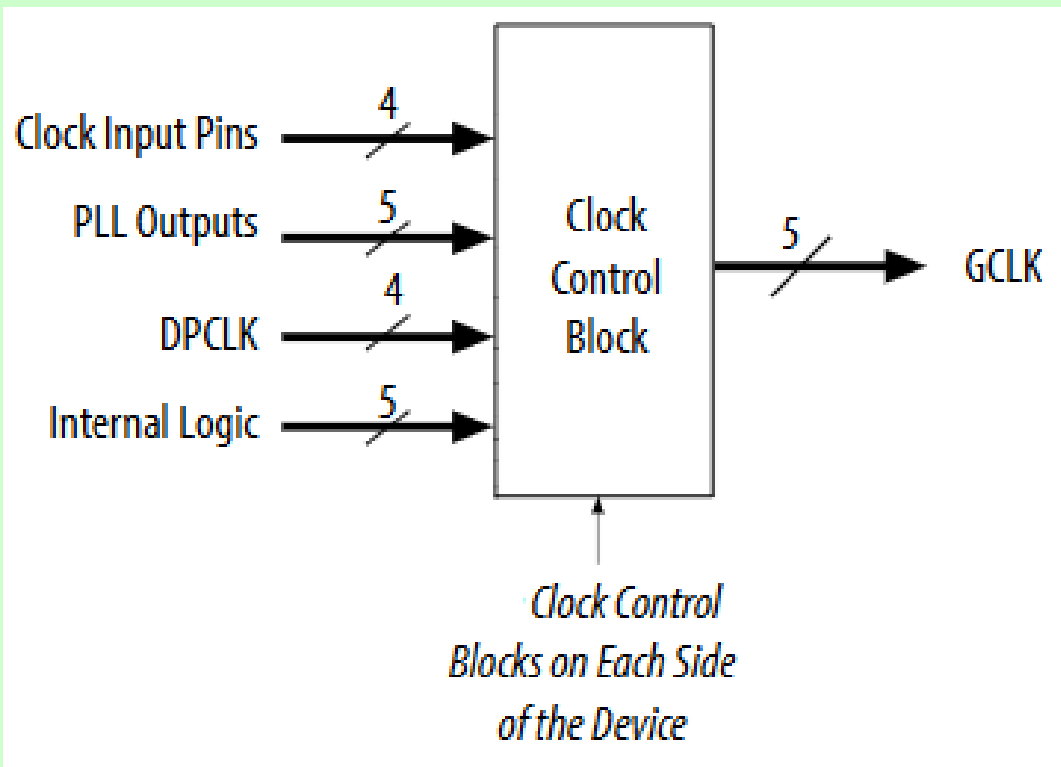
- pinów I/O,
- bloków LAB,
- mnożników wbudowanych,
- bloków pamięci.



Sygnaly sterujace takie jak:

- zezwolenie sygnału zegarowego,
 - synchroniczne zerowanie,
 - asynchroniczne zerowanie,
- doprowadzane są z zewnętrznych pinów lub generowane przez logikę wewnętrzną FPGA.

Zoptymalizowany pod względem poziomu zniekształceń i opóźnień blok sterujący (*CCB*) po przetworzeniu wyprowadza sygnały zegarowe do jednej z 20 GCLK.



Sygnały z wejść CLK/DPCLK mogą przepływać bezpośrednio na wejście CCB lub przez pętlę PLL.

Wyjście z bloku CCB jest przełączane do sieci GCLK.

Blok CCB dokonuje:

- dynamicznego wyboru źródła sygnału zegarowego FPGA,
- wyłącza zasilanie nieużywanej sieci GCLK (dynamiczne enable/disable).

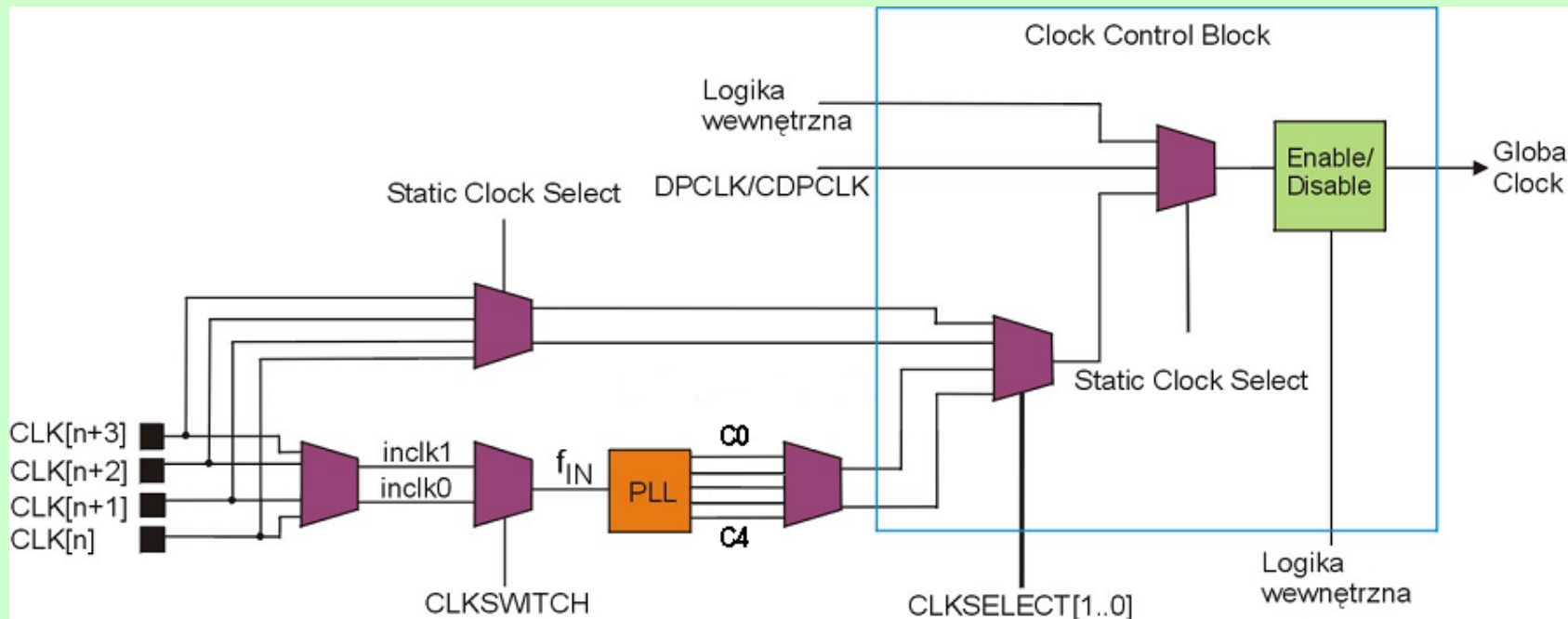
Sieć zegarowa, z powodu wysokiej częstotliwości pracy, jest znaczącym źródłem strat cieplnych.

Bity sterujące Static Clock Select służą do wyboru jednego/pary zewnętrznych źródeł sygnałów CLK - są one ustawiane w procesie konfiguracji FPGA. Źródła z wejść DPCLK/CDPCLK bezpośrednio łączą się z CCB.

CLKSWITCH może być ustawiany zarówno statycznie (podczas konfiguracji) jak również dynamicznie (w trakcie pracy FPGA). Ten bit wybiera zewnętrzny sygnał zegarowy podawany do PLL.

CLKSELECT[1..0] jest sterowany z logiki wewnętrznej (struktury FPGA), pozwala on na dynamiczne przełączanie źródła sygnału zegarowego z PLL lub CLK bezpośrednio do CCB.

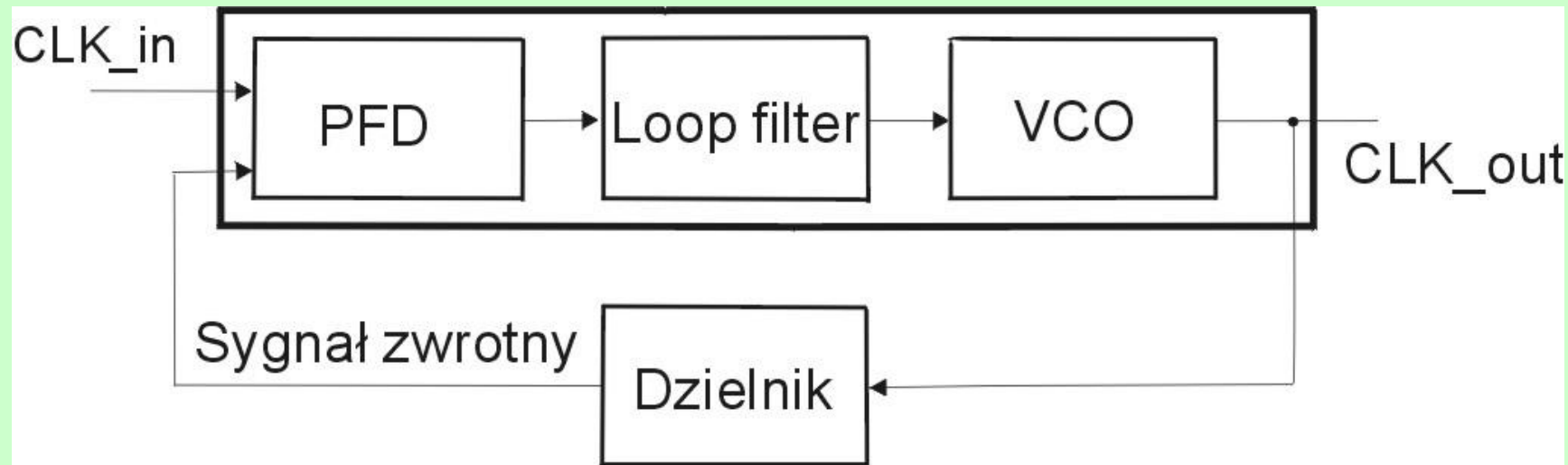
Wyłączenie lub włączenie sieci GCLK następuje sygnałem CLKENA z logiki wewnętrznej (proces wyłączania może odbywać się dynamicznie lub statycznie).



Do obsługi sygnałów zegarowych w Cyclone IV E przeznaczone są 4 pętle fazowe (PLL).

Funkcją PLL jest synteza sygnału zegarowego oraz jego kondycjonowanie - czyli:

- mnożenie częstotliwości sygnału zegarowego,
- dzielenie częstotliwości sygnału zegarowego,
- realizacja przesunięcia fazowego,
- regulacja współczynnika wypełnienia sygnału.
- regeneracja sygnału zegarowego, minimalizację opóźnień, regulację czasów propagacji.



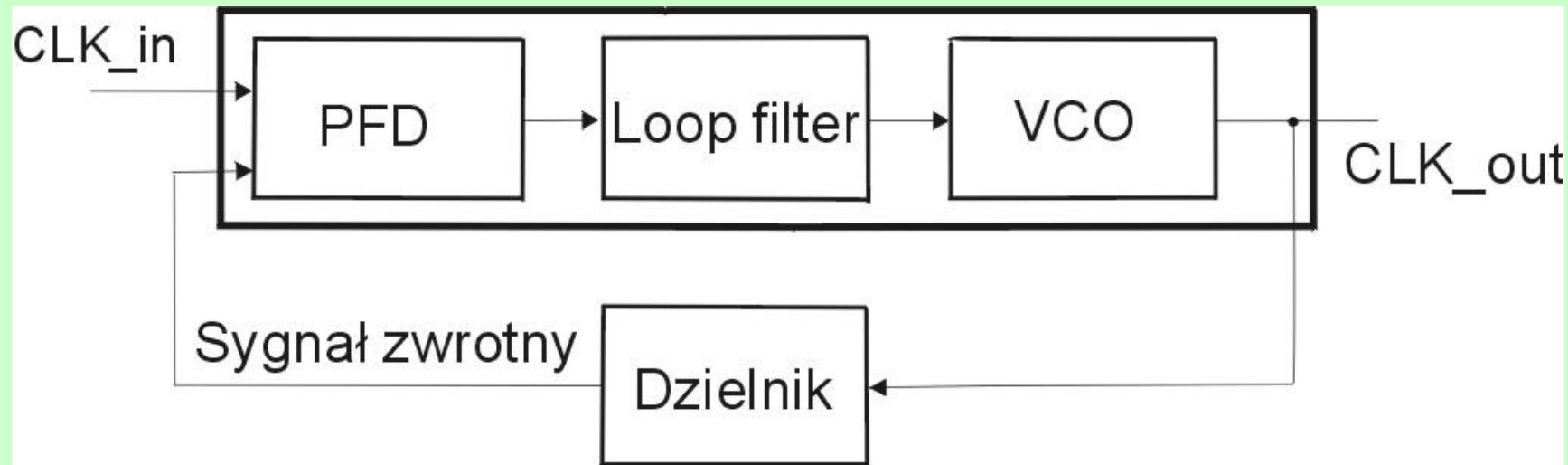
Pętla zawiera: detektor fazy, filtr pętlowy, generator przestrajany napięciem i dzielnik częstotliwości.

Sprzężenie zwrotne dopasowuje fazy/częstotliwości sygnału wyjściowego i wejściowego.

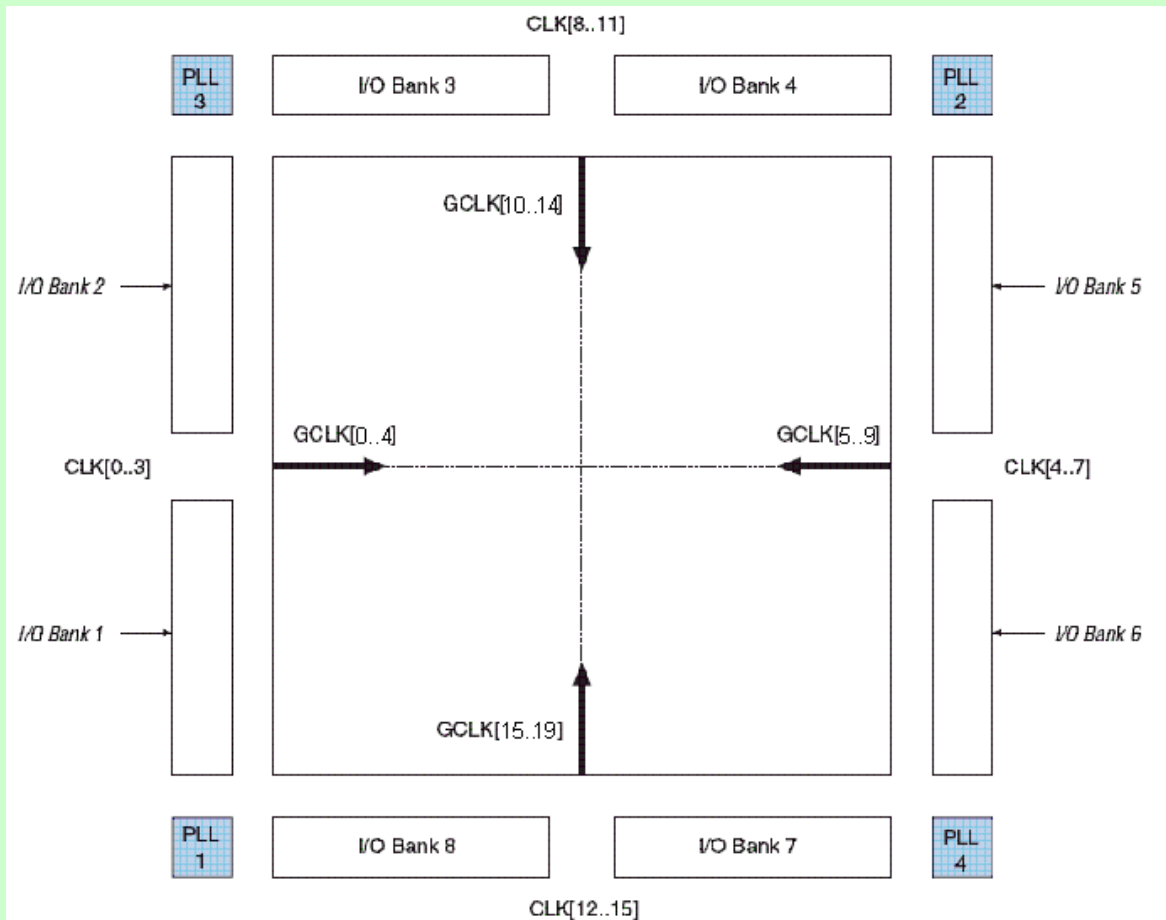
Występujący w sprzężeniu zwrotnym dzielnik powoduje, że częstotliwość sygnału wyjściowego może być wyższa od częstotliwości sygnału wejściowego.

Umieszczając dodatkowy dzielnik na wyjściu oraz moduły opóźniające - w pętli można regulować częstotliwość, fazę i opóźnienie sygnału wejściowego.

Pętla PLL jest analogowym blokiem wtrąconym w strukturę cyfrową FPGA.

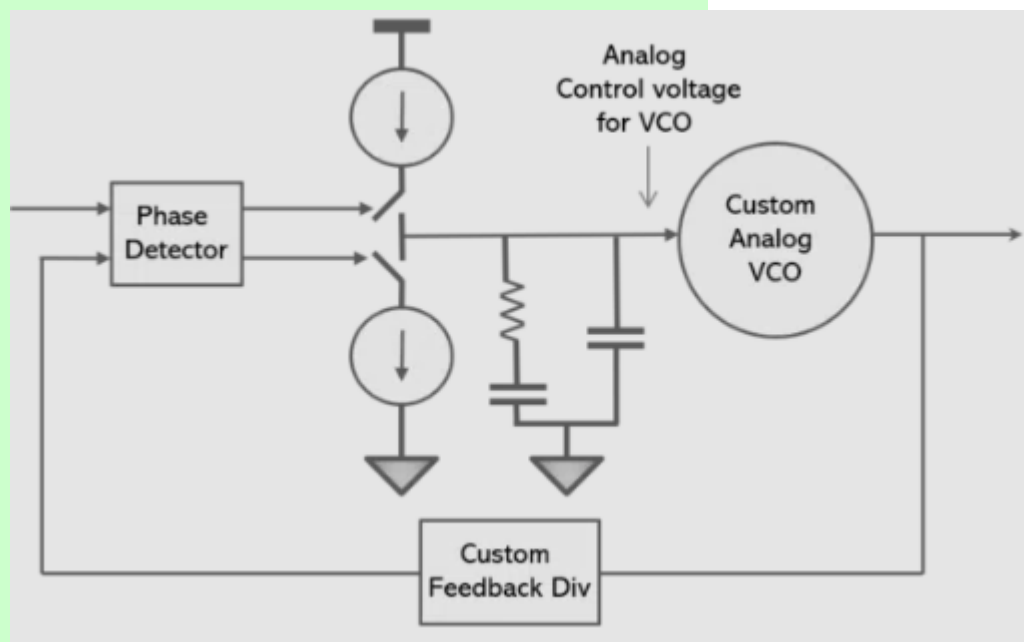
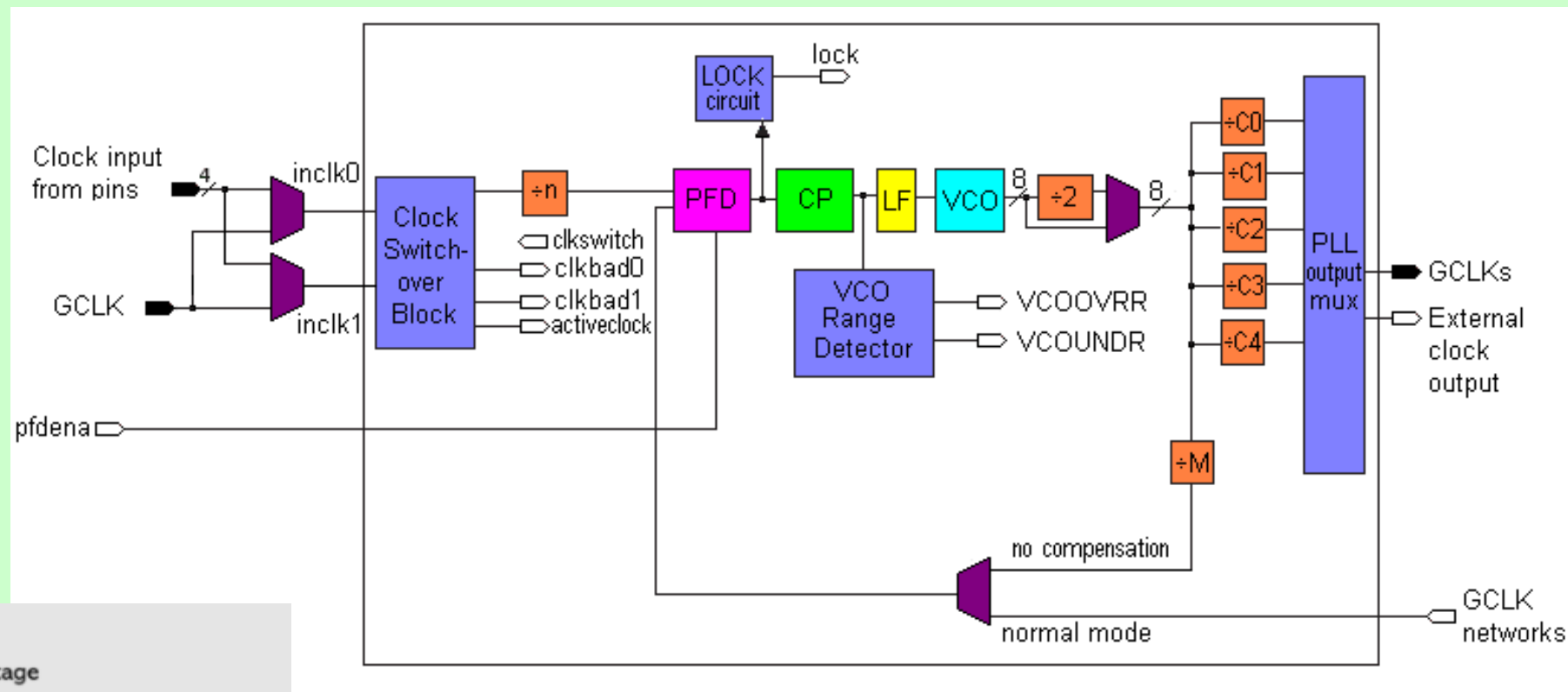


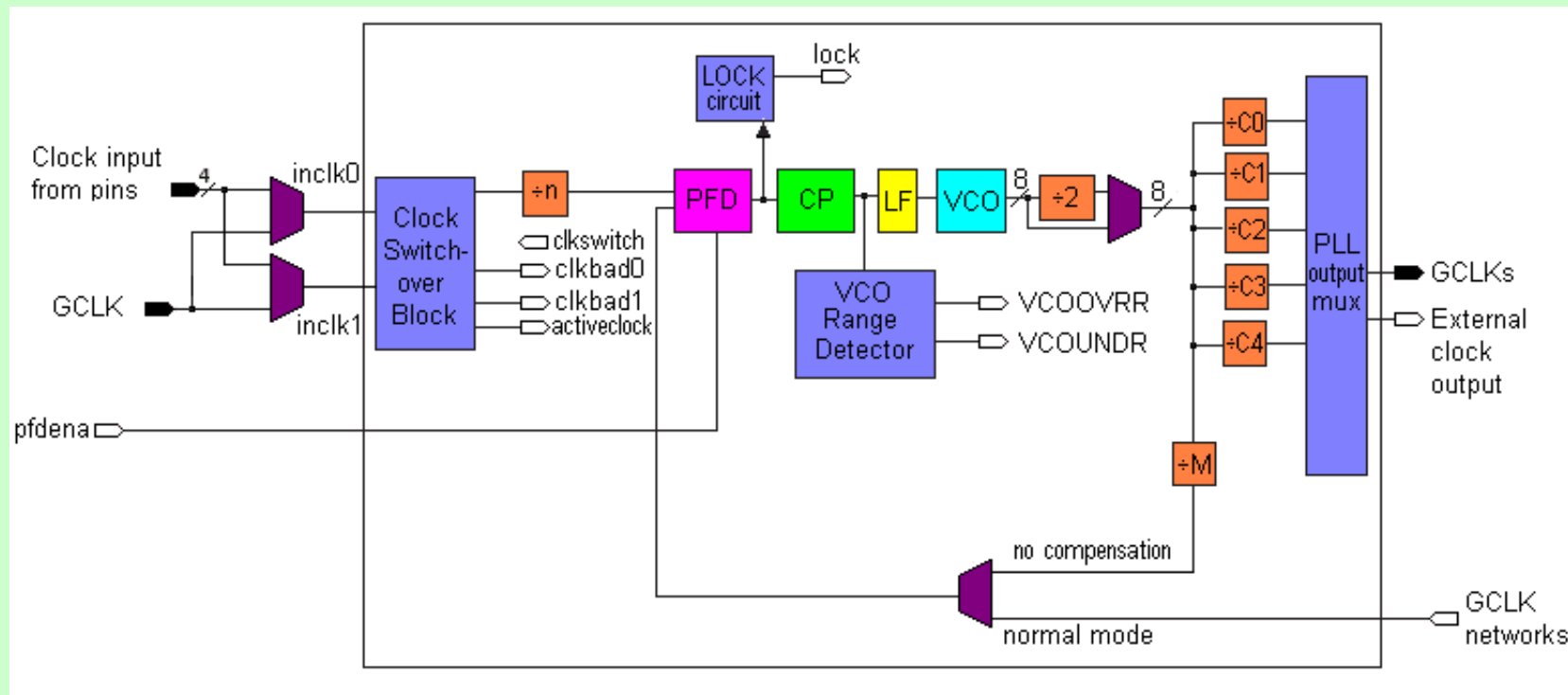
Pętle fazowe (w Cyclone 4 E) rozmieszczone są w czterech rogach układu FPGA z powodu dostępności niezależnego zasilania tych struktur. Realizują one:



- mnożenie częstotliwości przez $M = 1 \div 512$,
- dzielenie przez $n = 1 \div 512$,
- dzielenie przez $C = 1 \div 512$ sygnału I/O PLL,
- programowanie współczynnika wypełnienia,
- wytwarzanie do 5 sygnałów wewnętrznych,
- wytwarzanie zewnętrznego sygnału I/O,
- możliwość blokowania pracy pętli,
- możliwość wyboru sygnału zegarowego,,
- możliwość rekonfiguracji PLL,
- możliwość łączenia kilku PLL.

Architektura PLL
w FPGA jest
bardzo złożona:



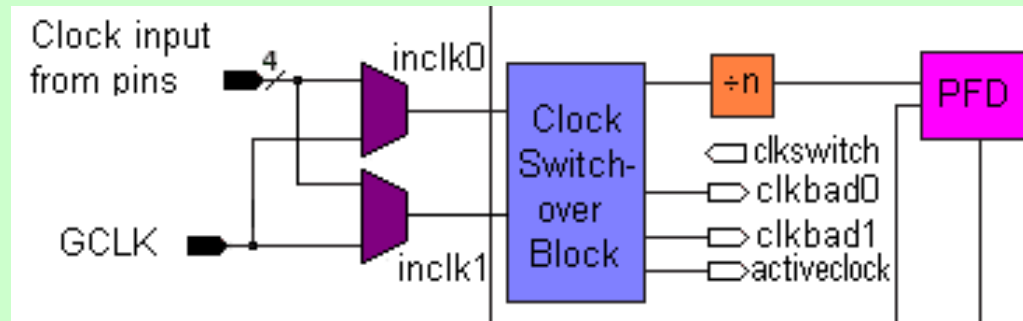


Sygnał do bloku CCB (na wyjściu) jest pobierany z: wejść CLK lub innej wewnętrznej GCLK.

Sygnały z CLK mogą być unipolarne lub bipolarne. W przypadku sygnałów bipolarnych uczestniczą dwa wejścia, np. CLK0 - LVDSCLK1p, CLK1 - LVDSCLK1n.

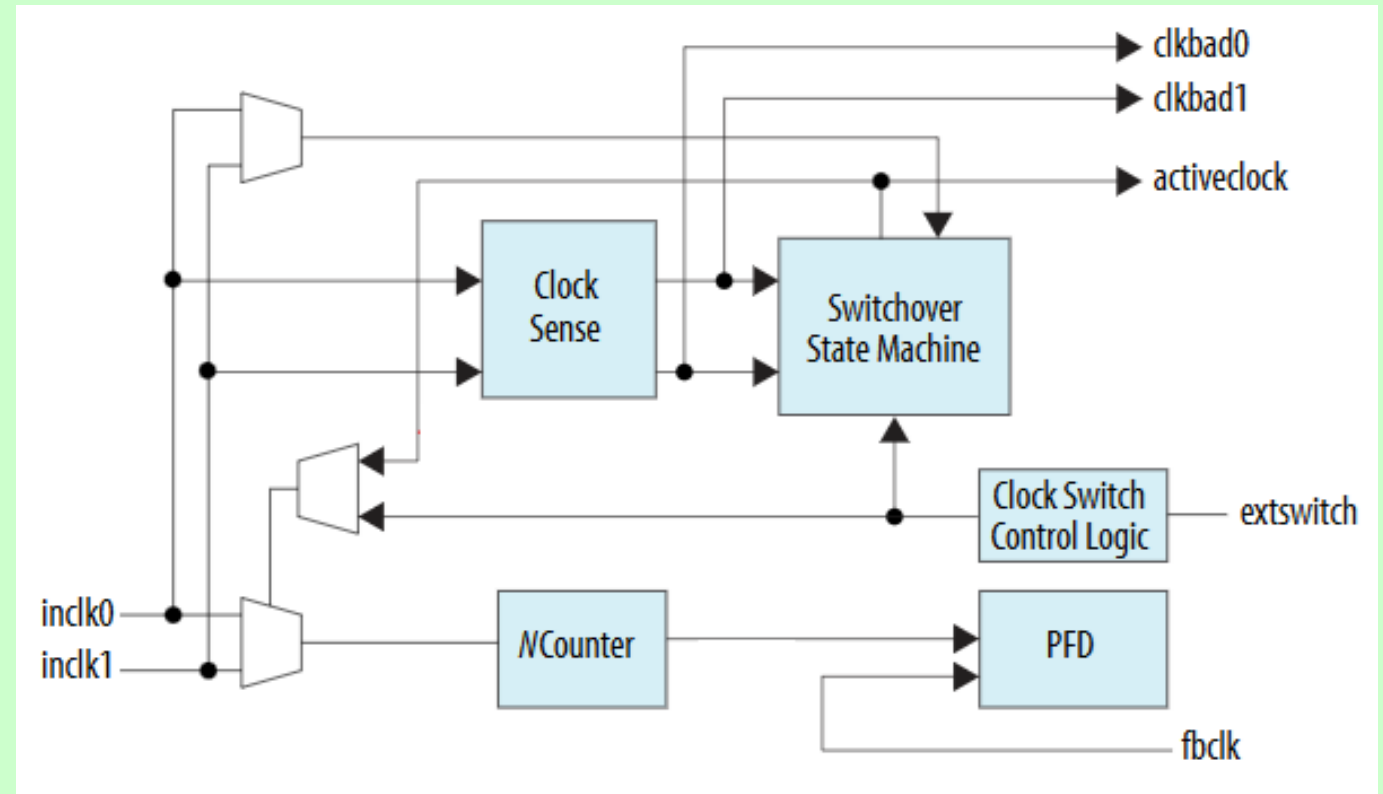
Wybrany przez multiplexer i automatyczny blok przełączający (CSB – Clock Switchover Block) sygnał jest dzielony przez n w preskalerze i podawany jako wejściowy sygnał odniesienia do PLL.

Preskaler pozwala na zmniejszenie częstotliwości sygnału na wejściu PLL – czyli pracę pętli z częstotliwością niższą niż wejściowa, realizując efekt **dzielenia częstotliwości** na wyjściu pętli.



Blok CSB (Clock Switchover Block) pozwala na przełączanie dwóch sygnałów zegarowych na wejściu PLL, w sytuacji gdy jeden z sygnałów zanika zachodzi potrzeba przełączania częstotliwości wyjściowej PLL. Na wyjściu bloku ustawiany jest jeden z sygnałów informujący o stanie sygnału wejściowego:

- clkbad0,
- clkbad1,
- activeclock.

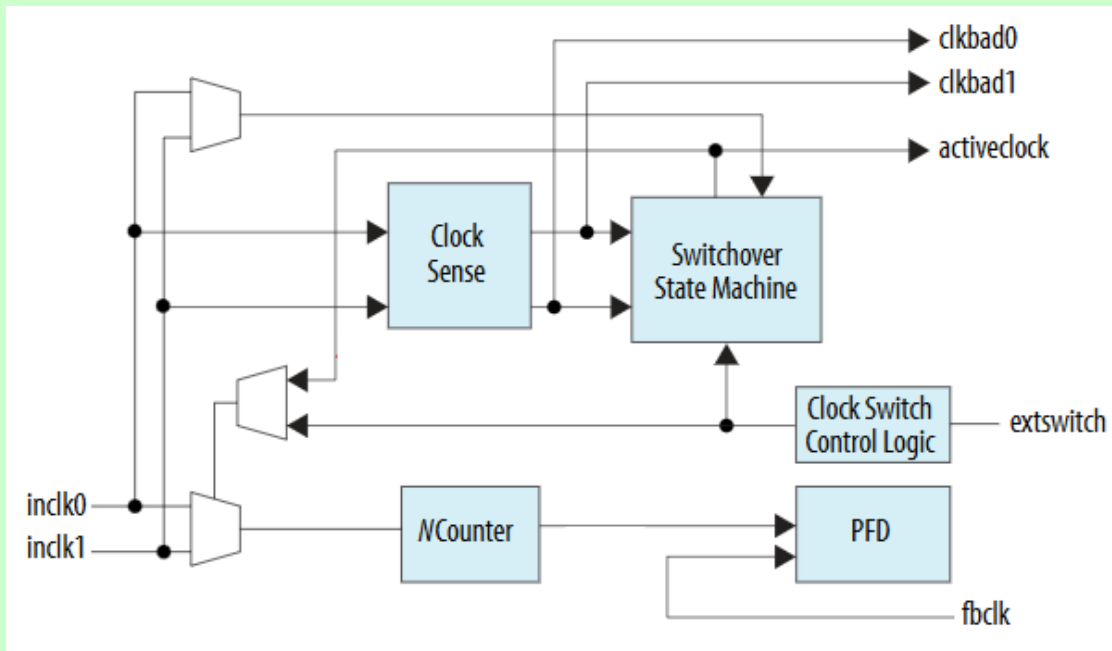


CSB może pracować w trybie automatycznym lub manualnym - sterowanym sygnałem clkswitch .

Mechanizm przełączania ma zastosowanie w dwóch przypadkach:

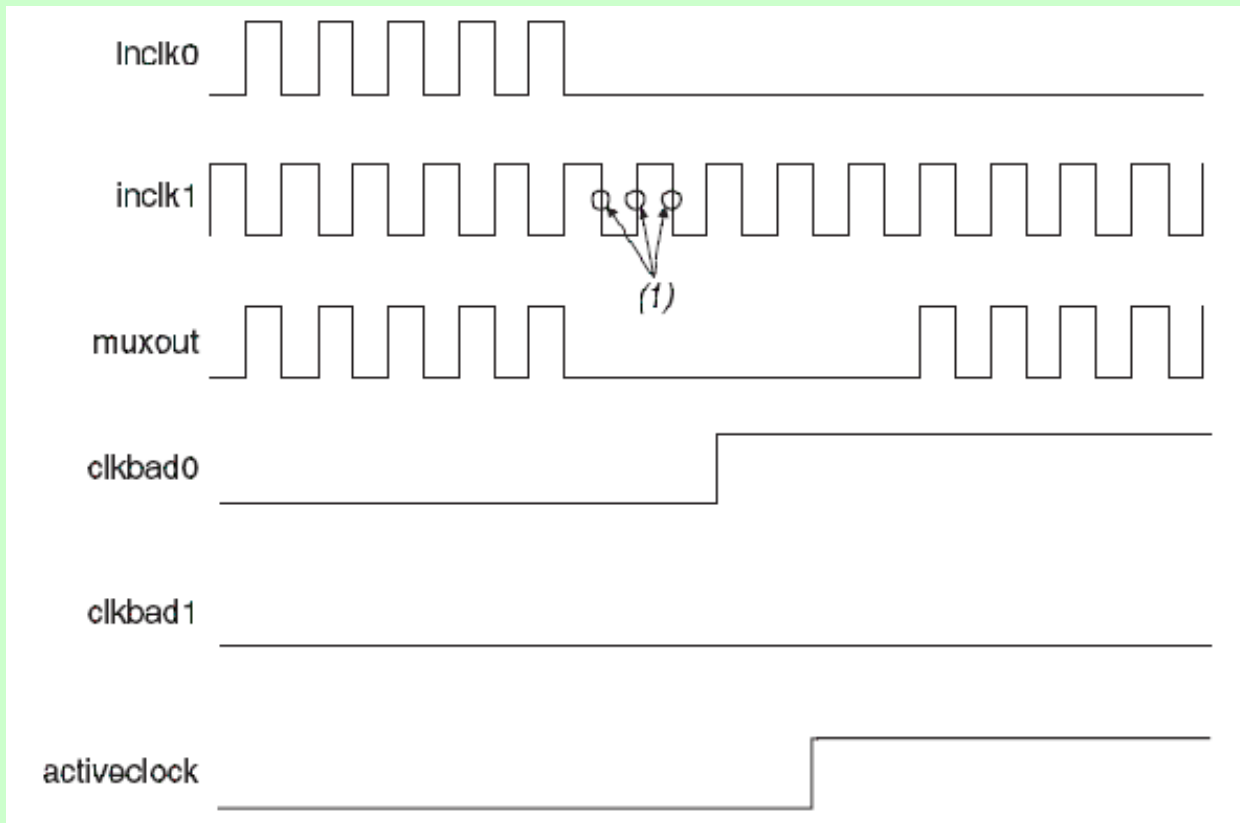
- do zwielokrotnienia źródła sygnału odniesienia – redundancji, w takiej sytuacji do obu wejść Inclk0 i Inclk1 jest doprowadzany ten sam sygnał i w momencie zaniku sygnału na jednym z nich automat stanu poprzez multiplexer automatycznie przełącza sygnał z drugiego wejścia,
- do manualnego, z zastosowaniem sygnału clkswitch, przełączania częstotliwości zegarowej urządzeń pracujących z różną szybkością, np. kart pracujących na 33 MHz i 200 MHz.

Automatyczny tryb nie obsługuje przełączania źródeł różniących się częstotliwościowo więcej niż o 20%.



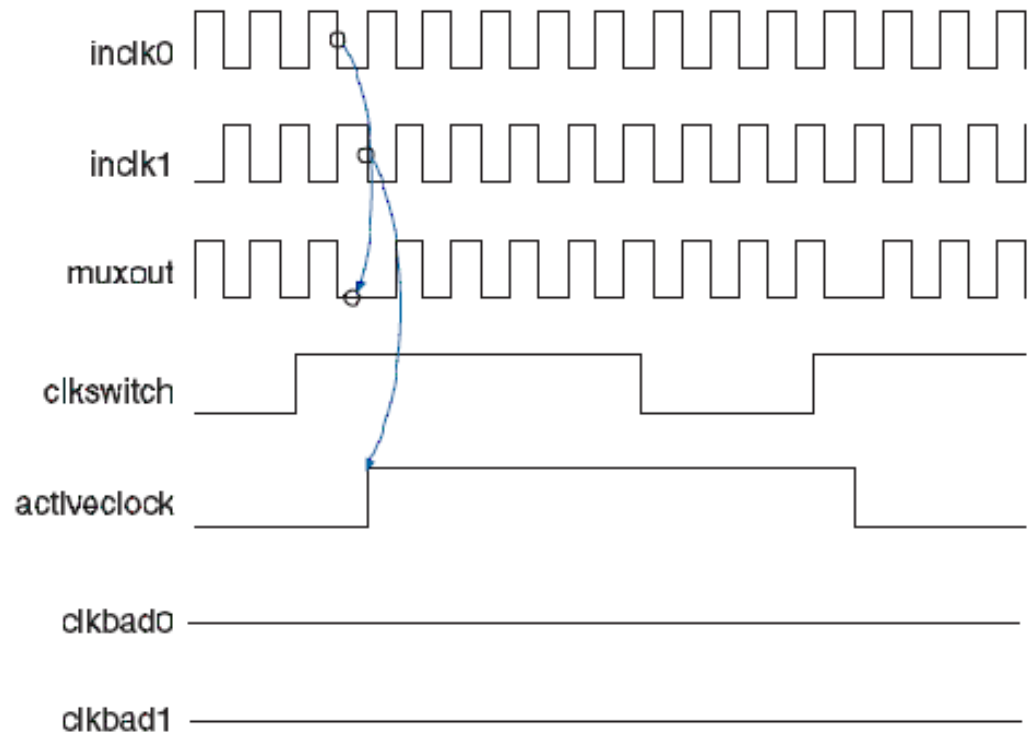
Prześledźmy czynność przełączania w trybie automatycznym:

- na wejściu odniesienia Inclk0 zanika sygnał,
- automat stanu sprawdza kolejne 3 zbocza sygnału Inclk1 i po 2 okresach ustawia clkbad0,
- następnie zmienia poziom sygnału activeclock, sygnalizujący przełączenie na Inclk1,
- przełącza na wyjście sygnał z Inclk1,
- automat wydłuża czas oczekiwania przy braku sygnału na obu wejściach - muxout = low.

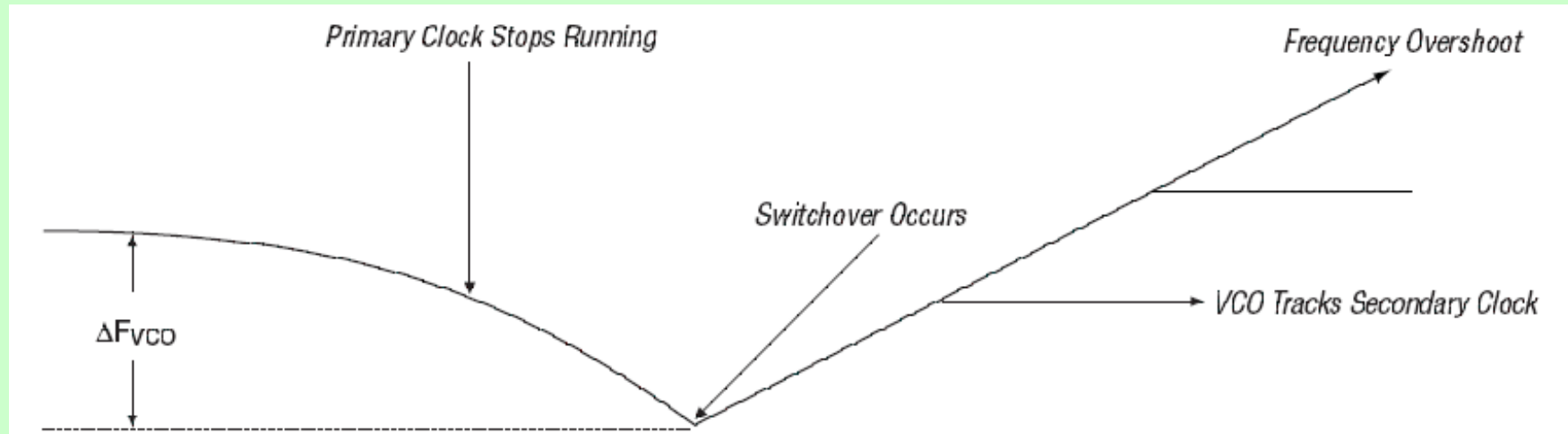


W trybie manualnym :

- na obu wejściach występuje stabilny sygnał,
- sygnał odniesienia jest pobierany z Inclk0,
- pojawia się żądanie zmiany wejścia – clkswitch przełączony zostaje do stanu wysokiego,
- na pierwszym opadającym zboczach Inclk0 zablokowany zostaje sygnał wyjściowy muxout,
- na zboczach Inclk1 zostaje przełączony jego sygnał na wyjście i ustawiony activeclock,
- sygnał clkswitch powinien pozostawać w stanie wysokim przez co najmniej trzy okresy.



- ponad 20% różnica częstotliwości wejściowych w trybie automatycznym generuje błędy clkbad0 i clkbad1,
- zbyt duże różnice częstotliwości i faz mogą zrywać synchronizację w trybie ręcznym,
- pasmo częstotliwościowe PLL należy dobierać do różnicy częstotliwości wejściowych,
- węższe pasmo filtruje „szpilki”, wolniej zrywa synchronizację i wydłuża czas chwytania pętli,
- przełączenie źródeł powoduje stan przejściowy w pracy PLL - brak sygnału na wejściu PFD,
- stan przejściowy może zmniejszać częstotliwość wyjściową lub zrywać synchronizację,

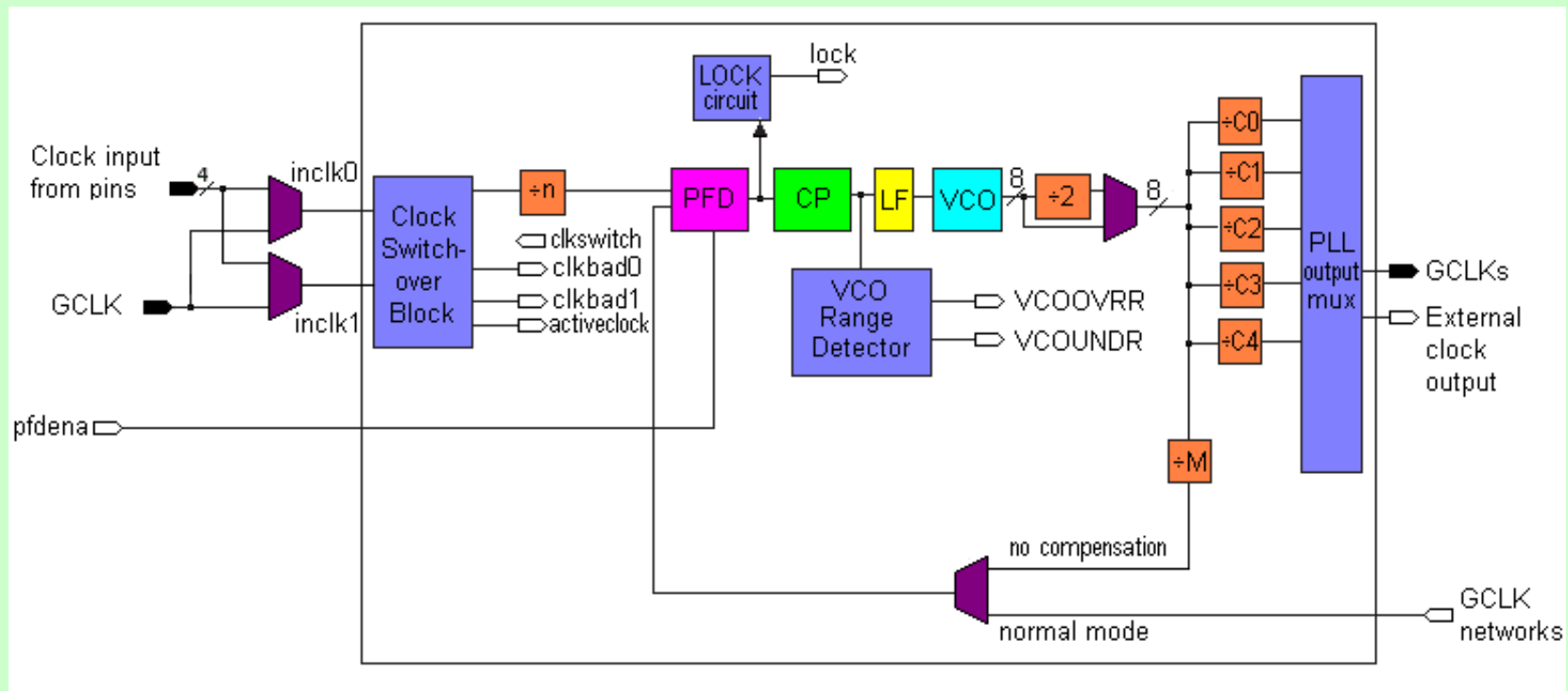


- w przypadkach zrywania synchronizacji można wyzerować pętlę sygnałem areset,
- sygnały clkbad0 i clkbad1 można wykorzystać do generacji sygnału pfdena, w celu zatrzaśnięcia pracy pętli w okresie przejściowym.

W zależności od znaku i wartości różnicy faz sygnałów na wejściach detektora fazy **PFD** wytwarza on, nie napięcie stałe jak w klasycznej PLL lecz, cyfrowe sygnały wyjściowe:

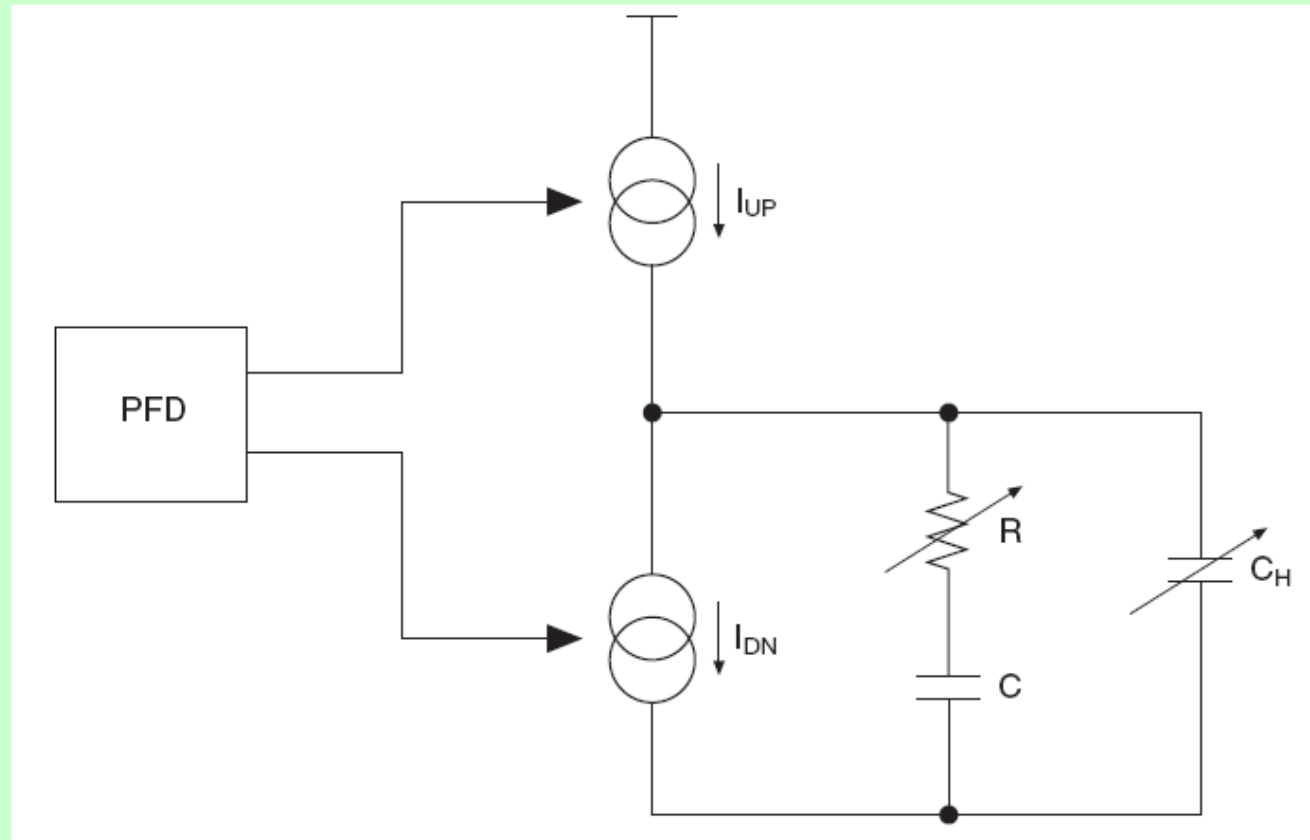
- up (żądanie zwiększenia częstotliwości),
- down (żądanie zmniejszenia częstotliwości).

Sygnały te sterują pompą ładunkową CP - czyli detektor fazy steruje VCO i wymusza zmianę częstotliwości w sposób cyfrowy.

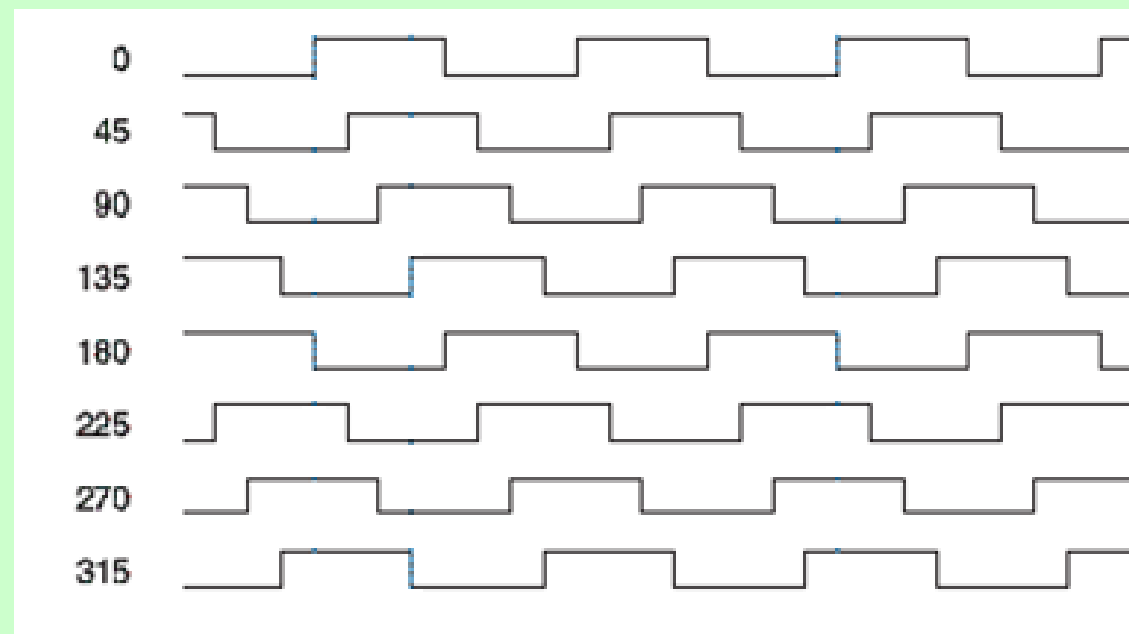
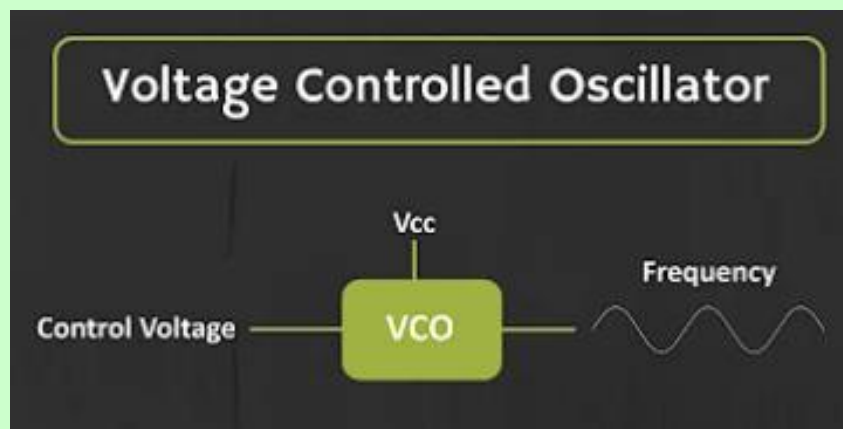


Pompą ładunkowa **CP** zmienia kierunek przepływu prądu w filtrze pętlowym **LF**, prąd wpływa do filtru jeżeli aktywny sygnał **up** lub wypływa przy aktywnym sygnale **down**.

Napięcie kondensatora pompy ładunkowej, po odfiltrowaniu w filtrze dolnoprzepustowym **LF**, steruje generatorem **VCO**, wymuszając zwiększenie lub zmniejszenie jego częstotliwości.



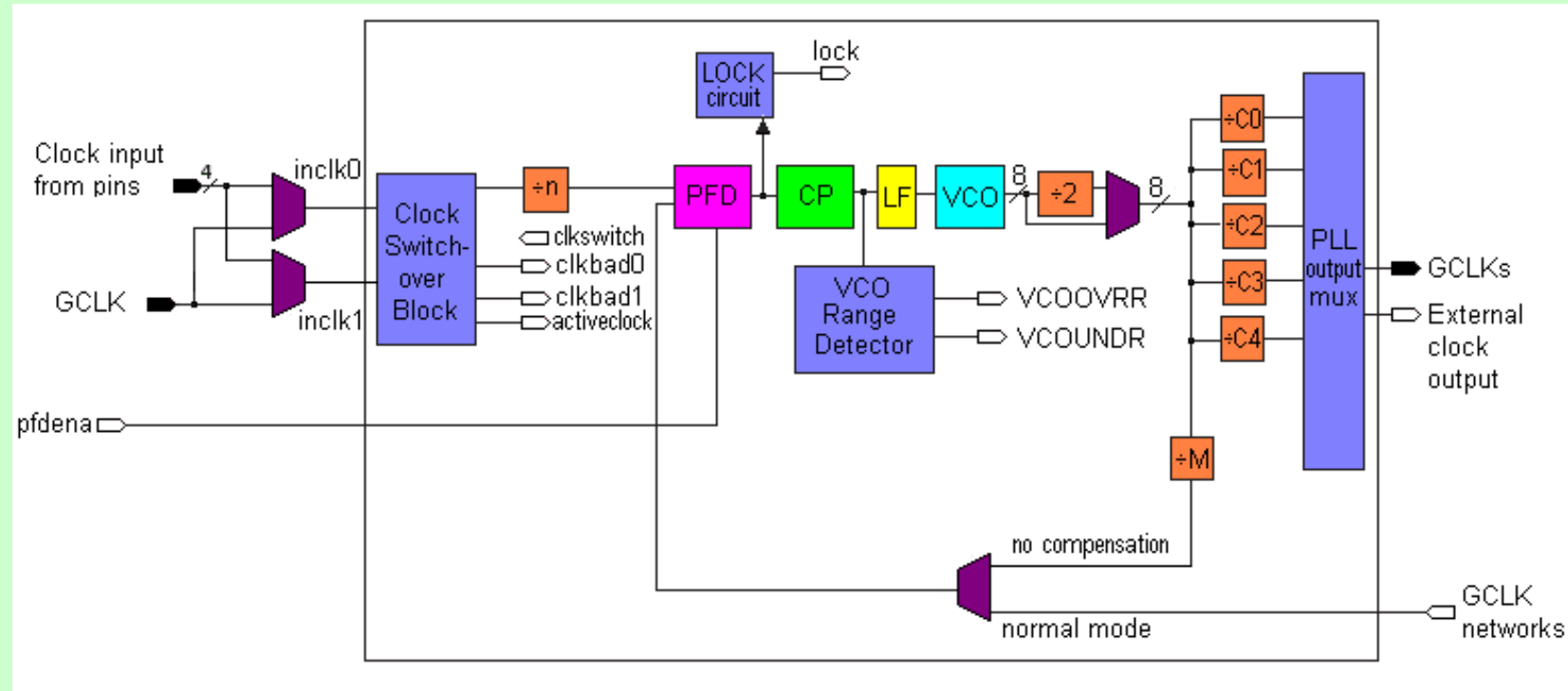
Generator VCO jest 8-bitowym generatorem pierścieniowym.



Pasmo częstotliwościowe pętli fazowej decyduje o dynamice pętli: jej właściwościach śledzących i zdolności do filtracji zakłóceń impulsowych.

Szersze pasmo szybciej wprowadza pętlę w stan synchronizacji oraz bardziej przenosi zakłócenia impulsowe z wejścia na wyjście.

Regulację pasma realizuje się programowo w wyniku zmiany nastaw wewnętrznych elementów RC filtra dolnoprzepustowego współpracującego z pompą ładunkową.

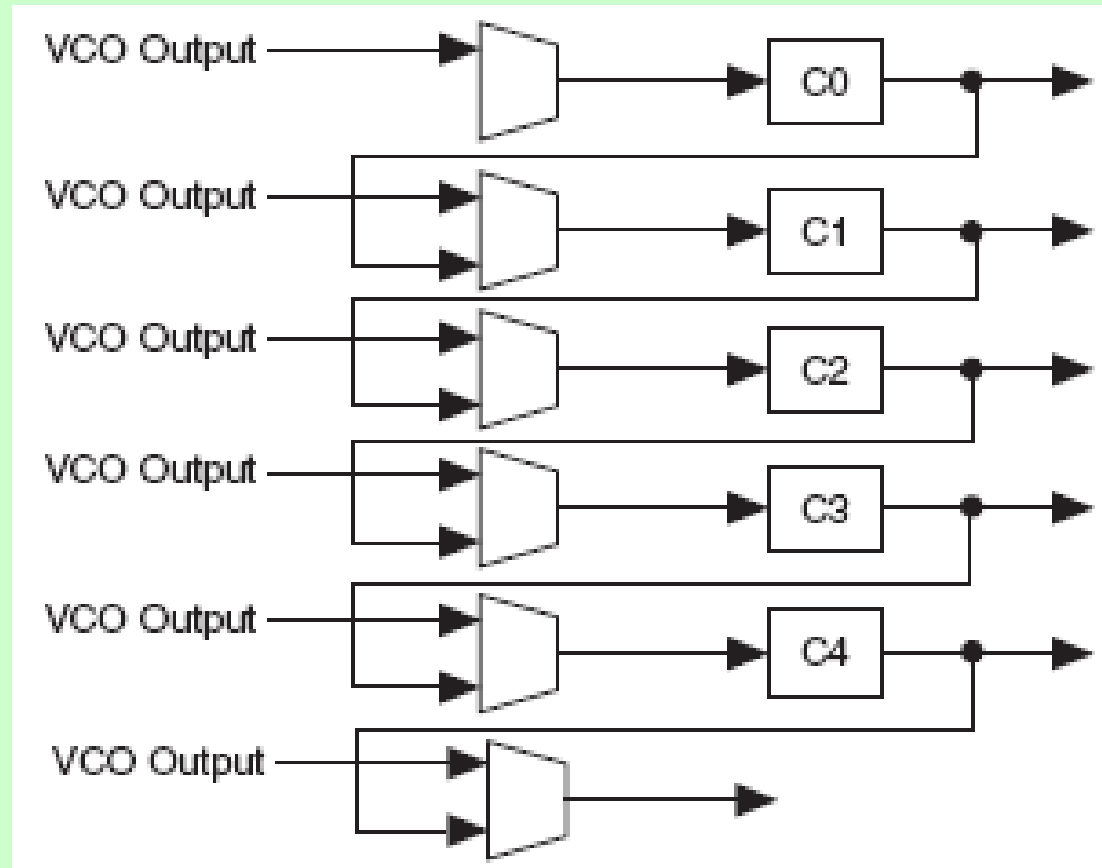


W pętli sprzężenia zwrotnego znajdują się postskalery częstotliwości wyjściowej z **VCO**, jeden dzielący **dwukrotnie** i drugi dzielący **M** częstotliwość sygnału. W ten sposób realizuje się pracę **VCO** z częstotliwością wyższą niż sygnał odniesienia – czyli **powielanie częstotliwości**.

Użytkownik nie ma wpływu na wartość kompensowanych opóźnień wprowadzanych przez pętlę. Częstotliwość wyjściowego multipleksa jest określona zależnością (w której C jest nastawą postskalera C0 .. C4):

$$f_{\text{global/external}} = f_{\text{IN}} \times (M / (n \times C))$$

Jeżeli podana zależność jest niewystarczająca do zapewnienia wymaganego zakresu dynamiki częstotliwości postskalery można połączyć kaskadowo. Wówczas podany wcześniej we wzorze parametr C jest iloczynem podziału postskalatorów składowych.

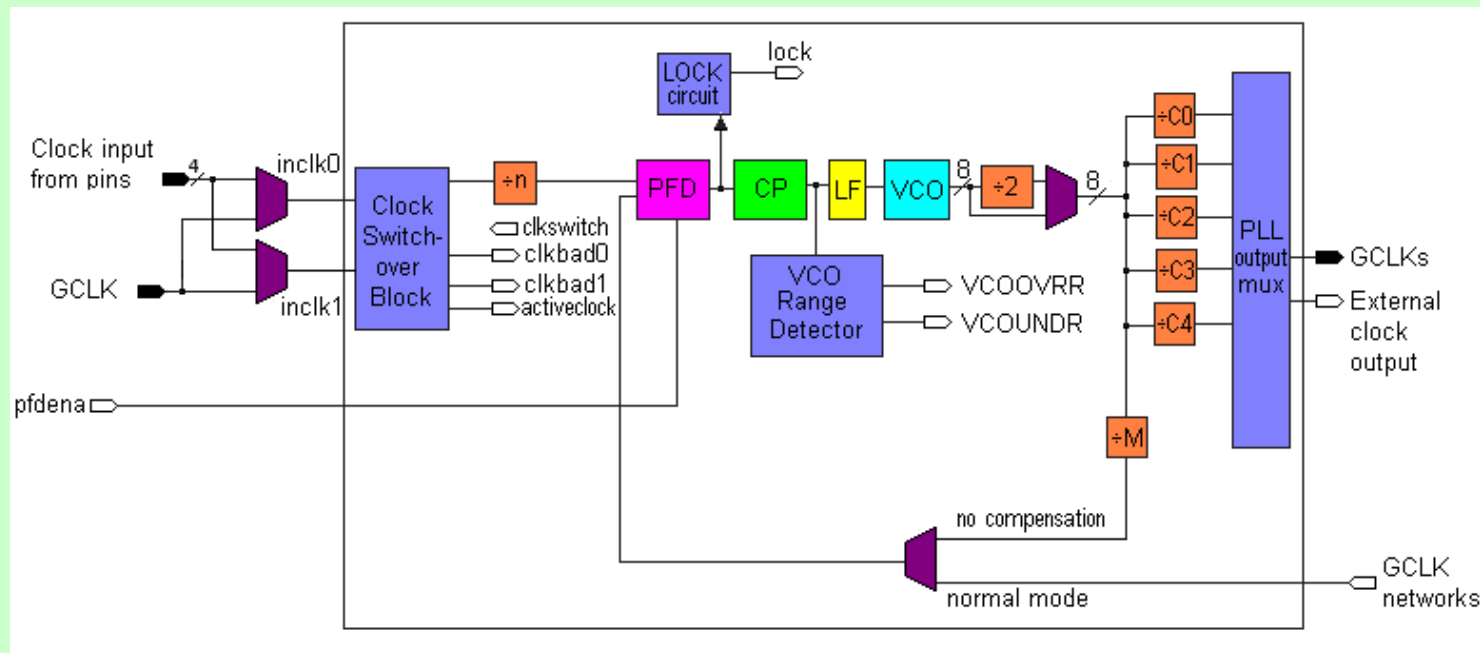


Sygnał **pfdena** służy do blokowania przestrajania pętli. Aktywny poziom sygnału wymusza pracę ze stałą, ostatecznie stabilną częstotliwością np. w sytuacji zanikania sygnału wejściowego.

Sygnał **areset** (niewidoczny na schemacie) podawany z pinu zewnętrznego lub struktury wewnętrznej wymusza ponowną synchronizację pętli: zeruje liczniki, wyprowadza pętlę ze stanu zatrzaśnięcia, ustawia tryb pracy na normalny i wymusza ponowną synchronizację.

Wyjściowy sygnał **locked** informuje logikę wewnętrzną o wejściu pętli w stan synchronizacji.

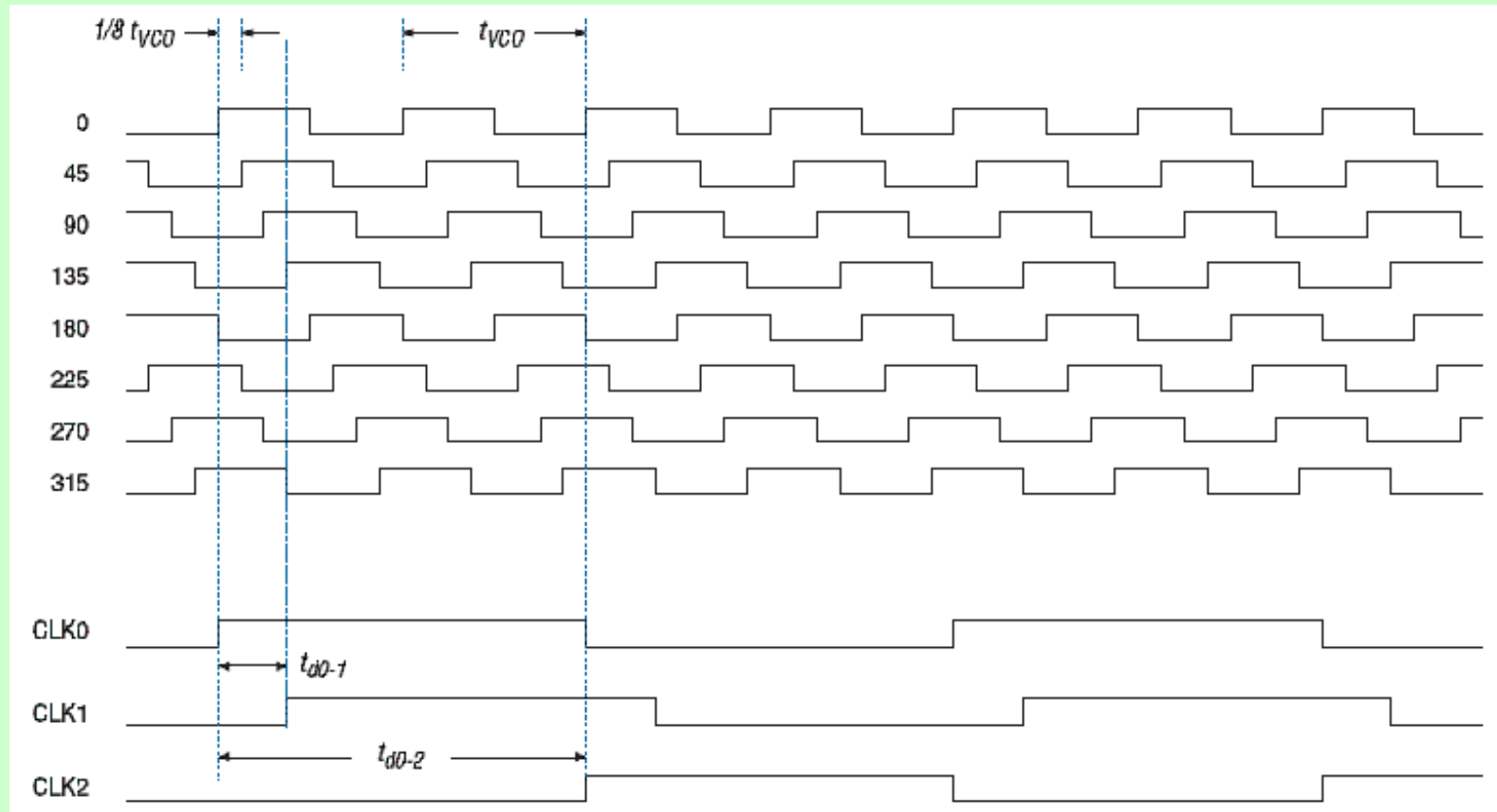
Sygnał ten może odczytywać przez logikę wewnętrzną FPGA.



Przesunięcie fazowe sygnału wyjściowego realizuje się poprzez zmianę fazy i momentu rozpoczęcia pracy liczników. Jest to metoda niezależna od parametrów elektrycznych czy termicznych struktury.

Stosuje się w tym celu dwie techniki:

- precyzyjną – wykorzystującą przesunięcia fazowe sygnałów wyjściowych VCO,
- zgrubną – opartą na regulacji czasu startu licznika.



Pętla fazowa Cyclone IV E może pracować w jednym z 4 trybów, w których realizowane są funkcje mnożenia, dzielenia, przesunięcia fazowego oraz regulacji współczynnika wypełnienia.

Kontrolowane opóźnienie sygnału zegarowego jest realizowane jedynie w przypadku, gdy wejściowy sygnał zegarowy doprowadzany jest do dedykowanego pinu wejściowego.

Tryb z **synchronizacją danych** jest używany w synchronicznej transmisji danych.

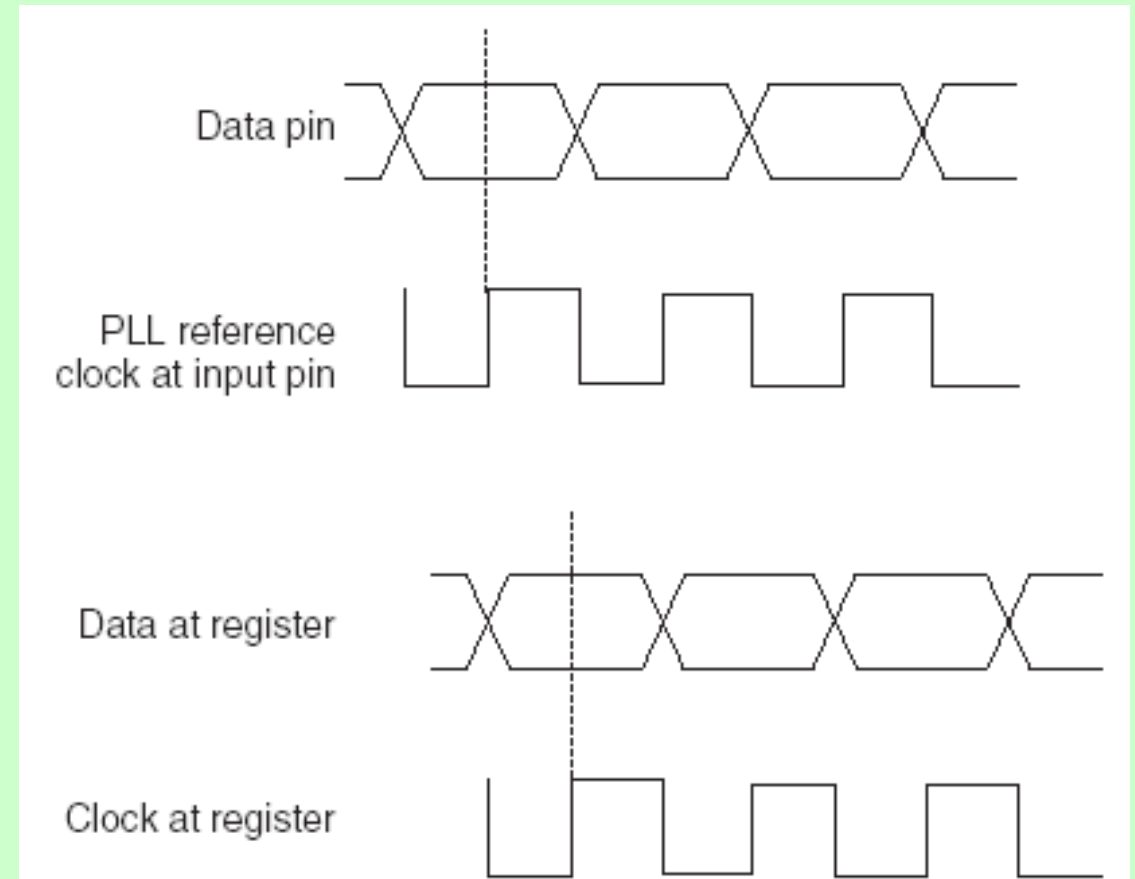
Sygnał zegarowy i dane są równocześnie dostarczane do portów wejściowych.

Kompensacja opóźnienia dotyczy dwóch ścieżek sygnałowych:

- ścieżki danych do wejść rejestrowych,
- sygnału zegarowego do detektora fazy.

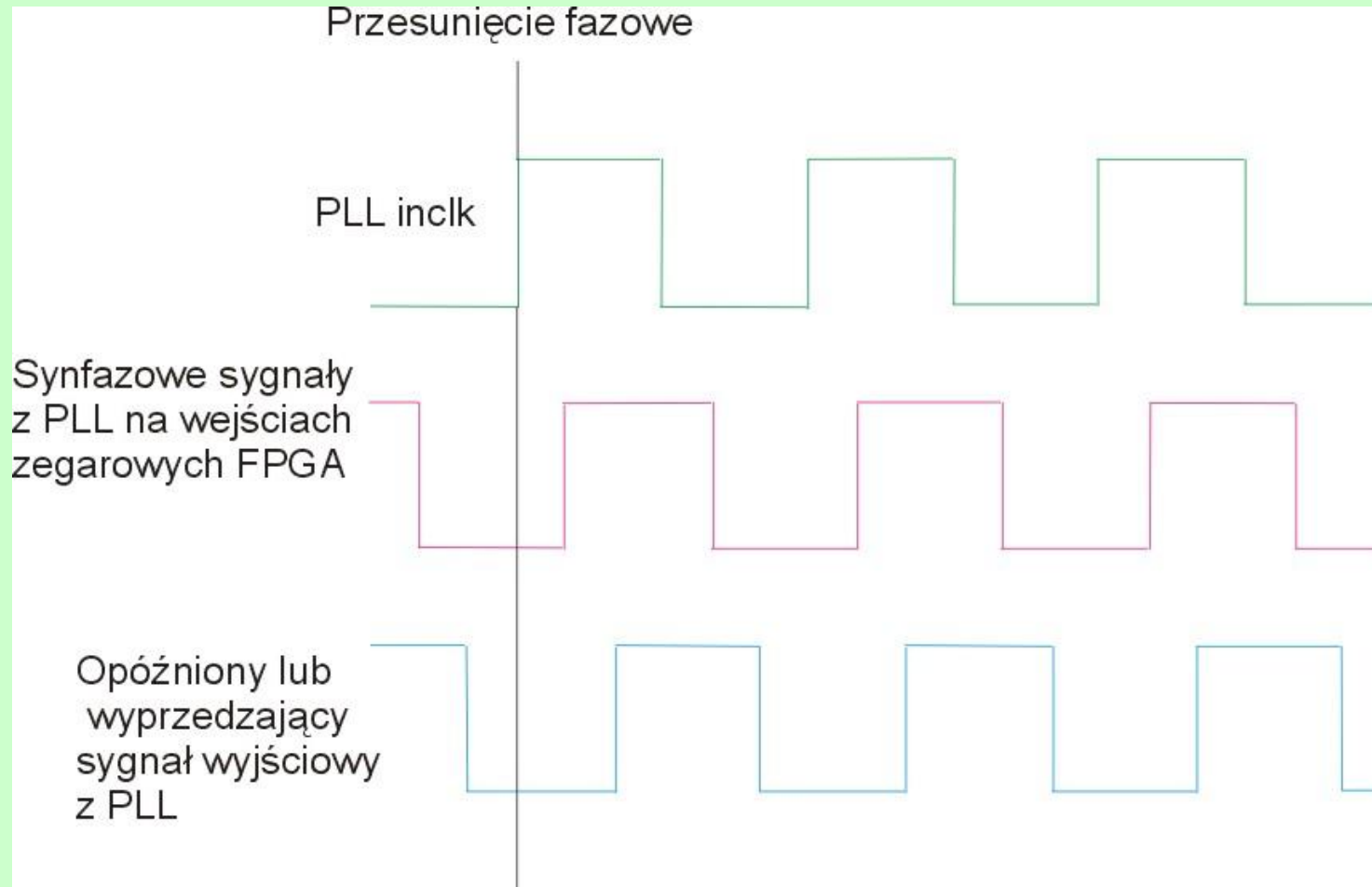
W tym trybie wejściowe sygnały danych należy ustawić w zerową wartość początkową.

Dodatkowo dla pinów danych wejściowych należy w Quartus II ustawić opcję **PLL COMPENSATED logic**.



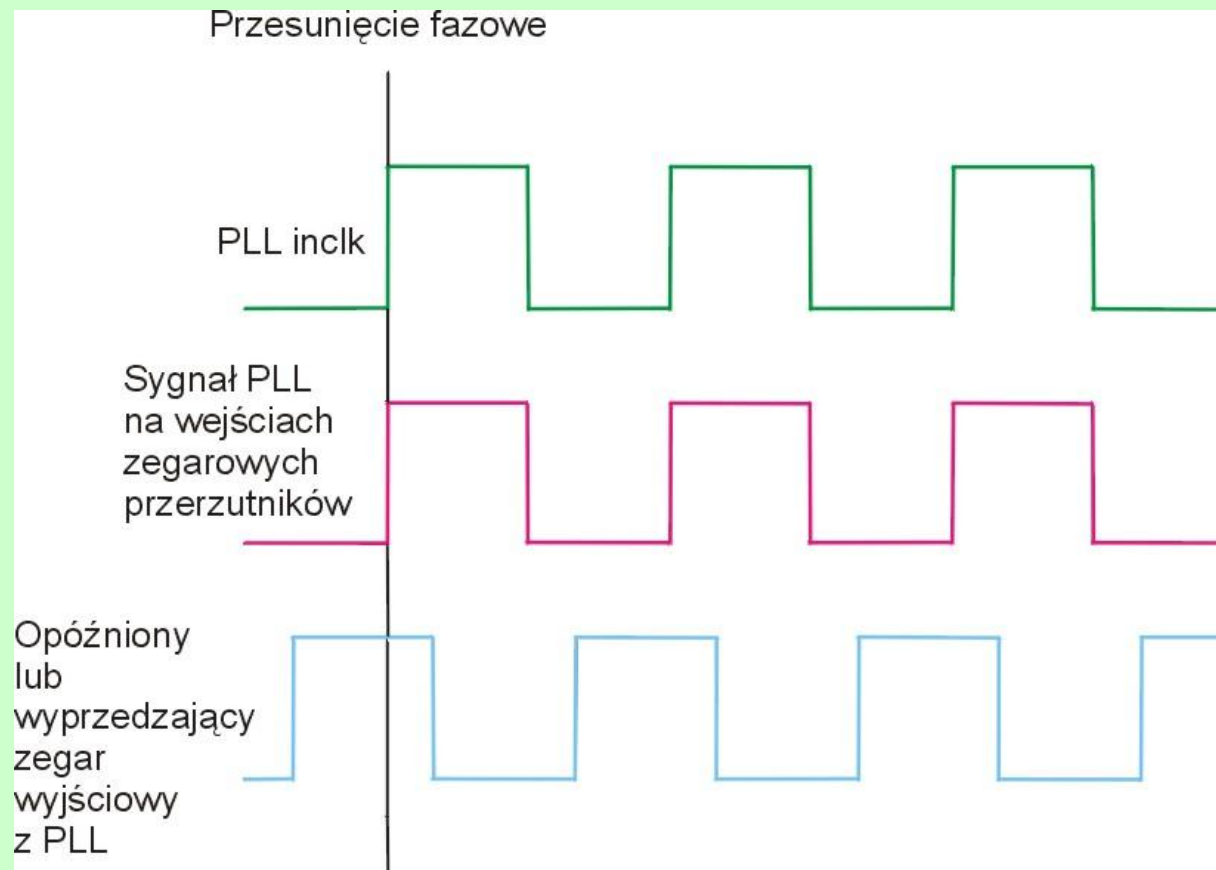
W trybie **bez kompensacji** występują różne przesunięcia fazowe sygnałów wyjściowych pętli - zarówno w stosunku do sygnału odniesienia na pinie FPGA jak i wewnątrz struktury.

Tryb wykazuje korzystniejsze charakterystyki w obecności zakłóceń impulsowych sygnału.



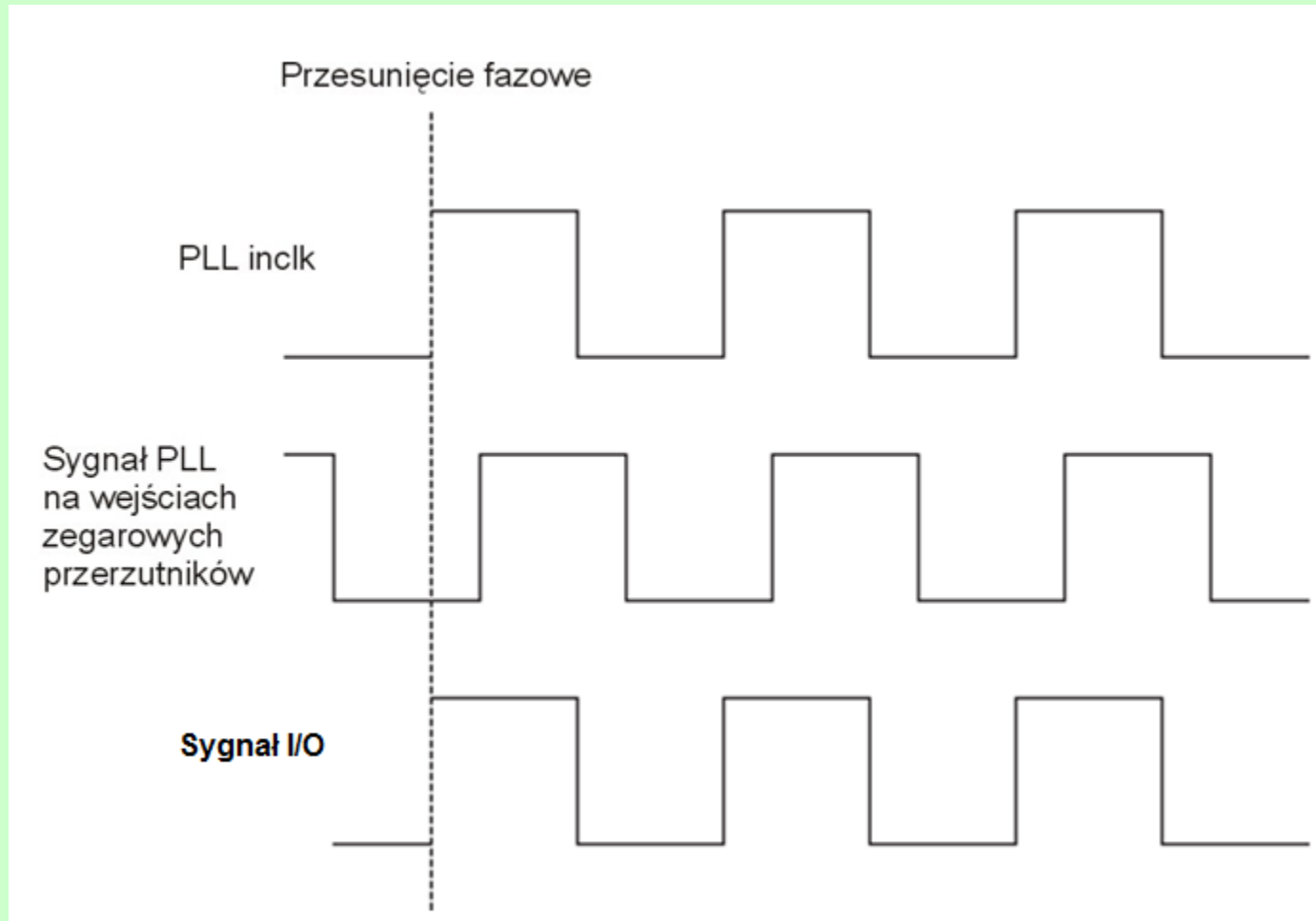
W trybie **normalnym** PLL ustawia zgodne (lub kontrolowane) fazy wejściowego sygnału odniesienia/zegarowego oraz sygnałów na wejściach zegarowych wewnętrznych przerzutników FPGA, kompensując wewnętrzne opóźnienia czasów propagacji. W programie Quartus II podczas definicji pętli należy wybrać, które wyjście pętli (C0 .. C4) ma być kompensowane.

Przetworzony wyjściowy sygnał z PLL na pinie FPGA jest przesunięty w fazie.



W trybie z **zerowym opóźnieniem** kompensowane są przesunięcia fazowe pomiędzy wejściowym sygnałem zegarowym oraz na pinie wyjściowym układu przetworzonego sygnału.

Jako piny wejściowe i wyjściowe mogą być użyte piny I/O ogólnego przeznaczenia.



Zaliczenie częściowe wykładu nr 8 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

Zajęcia laboratoryjne realizowane są w oparciu o platformę projektową CAD PLD firmy Intel – system Quartus II. Istnieją inne platformy, oferowane przez producentów PLD/FPGA – np. Vivado - AMD, Atmel Studio –Atmel, Libero SoC – Microchip, itp. Wybór Quartus II wynika z używanych w laboratorium płyt prototypowych tego producenta. Nie zaleca się stosowania tzw. kompilatorów uniwersalnych – tylko firmowe oprogramowanie gwarantuje pełne wykorzystanie modułów FPGA konkretnego producenta. Używane w czasie laboratorium oprogramowanie w wersji Lite Edition można pobrać bezpośrednio ze strony <https://www.intel.com>

Download Intel® Quartus® Prime Software

Three Intel® Quartus® Prime editions to meet your system design requirements

Pro Edition

The Intel® Quartus® Prime Pro Edition Software supports the advanced features in Intel's next-generation FPGAs and SoCs with the Intel® Agilex™, Intel® Stratix® 10, Intel® Arria® 10, and Intel® Cyclone® 10 GX device families.

Cyclone 10 GX devices supported for free in Intel Quartus Prime Pro Software Edition.

Download for Windows (paid license required)

Download for Linux (paid license required)

Standard Edition

The Intel® Quartus® Prime Standard Edition software includes extensive support for earlier device families in addition to the Intel® Cyclone® 10 LP device family.

Download for Windows (paid license required)

Download for Linux (paid license required)

Lite Edition

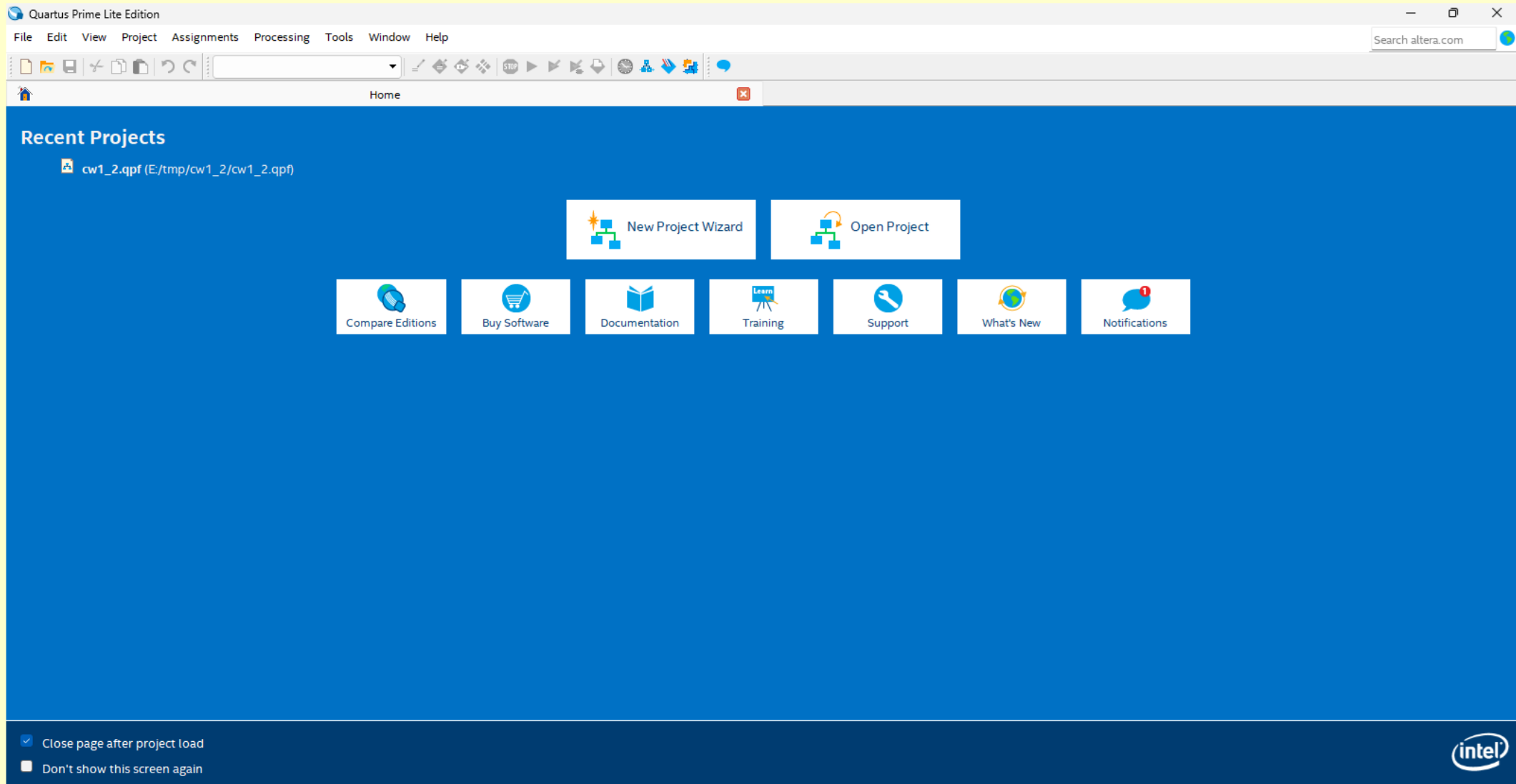
The Intel® Quartus® Prime Lite Edition software supports Intel's low-cost FPGA device families.

Refer to [Features](#) to review all devices supported by the Free Quartus Prime Software Lite Edition.

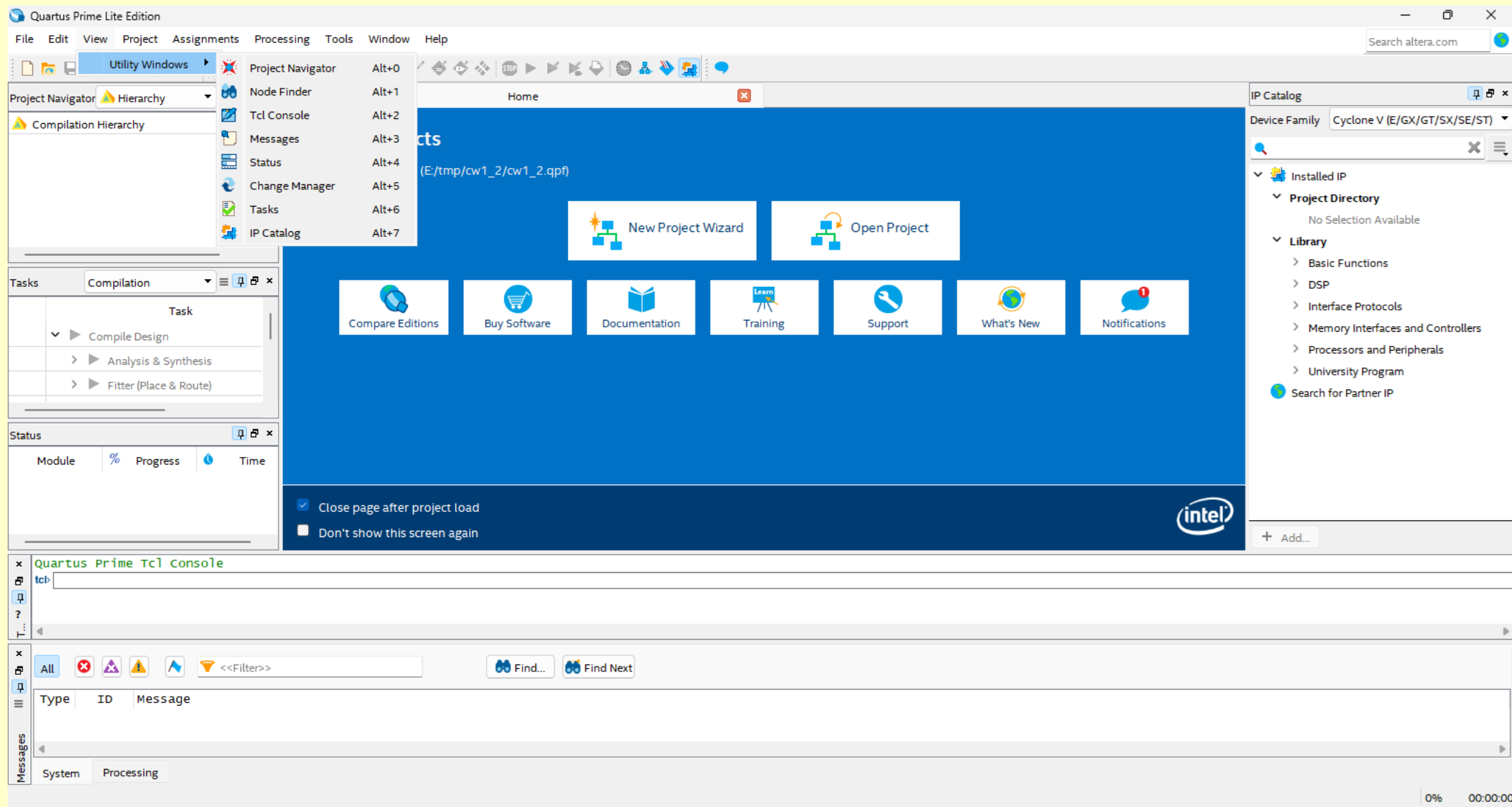
Download for Windows (free, no license required)

Download for Linux (free, no license required)

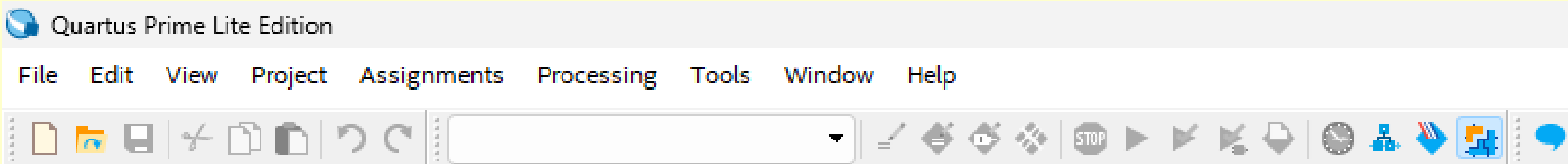
Po uruchomieniu oprogramowania możemy stosownie do potrzeb konfigurować widok monitora użytkownika, włączając lub wyłączając przydatne/niepotrzebne moduły.



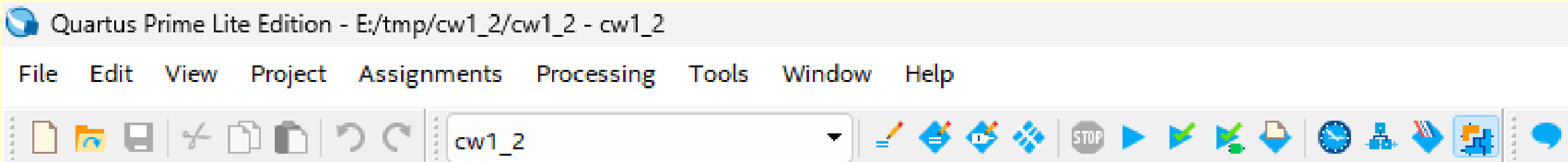
Widok monitora ustawiamy używając górnego menu: View → Utility Windows → klikając nazwę modułu:



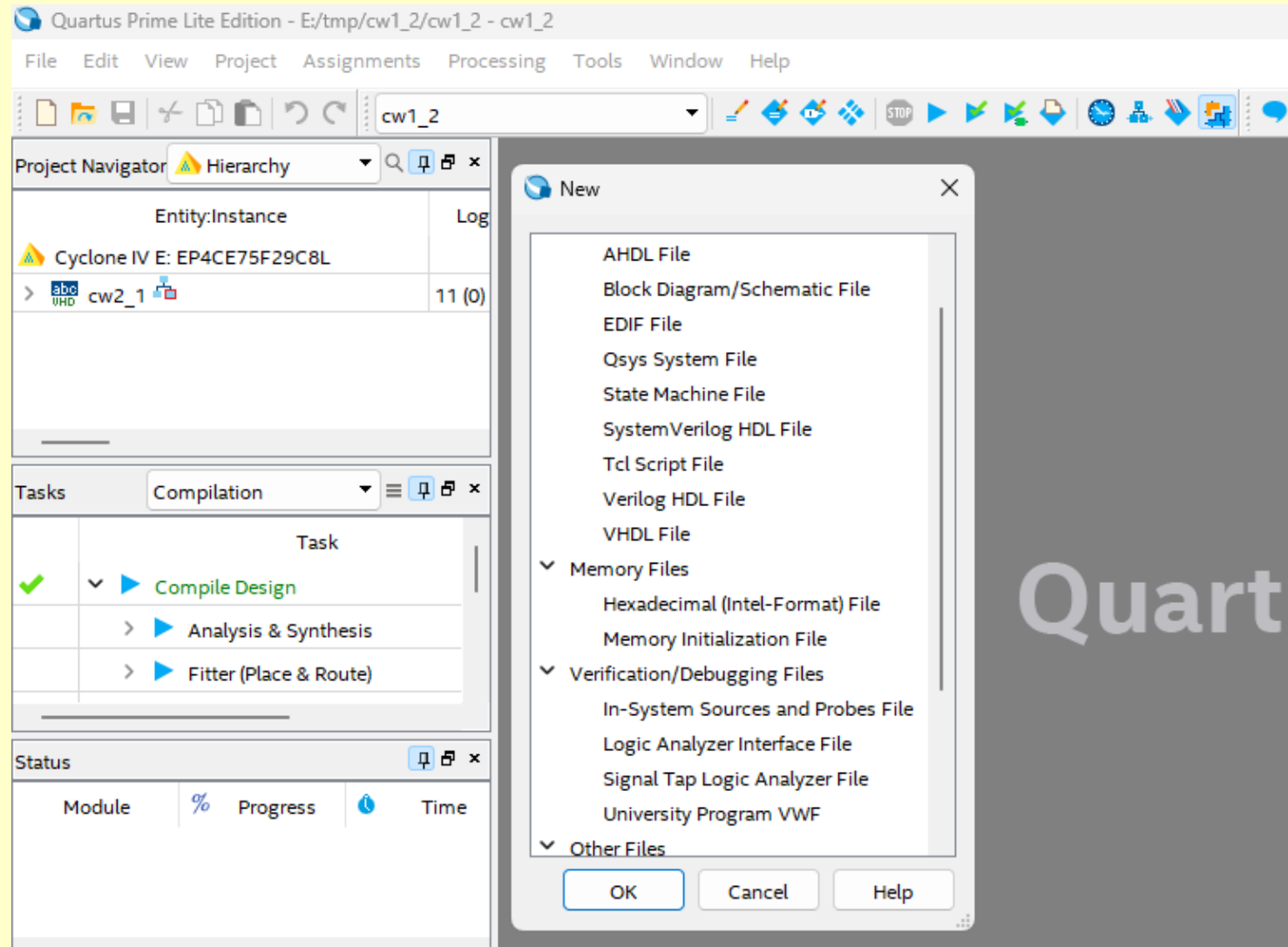
Pracę w Quartus II zawsze rozpoczynamy – inicjując projekt, może to być otwarcie istniejącego lub utworzenie nowego projektu. Nie rozpoczynamy pracy – otwierając nowy plik w Q II lub klikając plik pod systemem Windows. Brak nazwy projektu w pierwszym wierszu po tytule Quartus Prime Lite Edition informuje o braku inicjacji projektu – i dalsza praca bez inicjacji projektu jest nieefektywna:



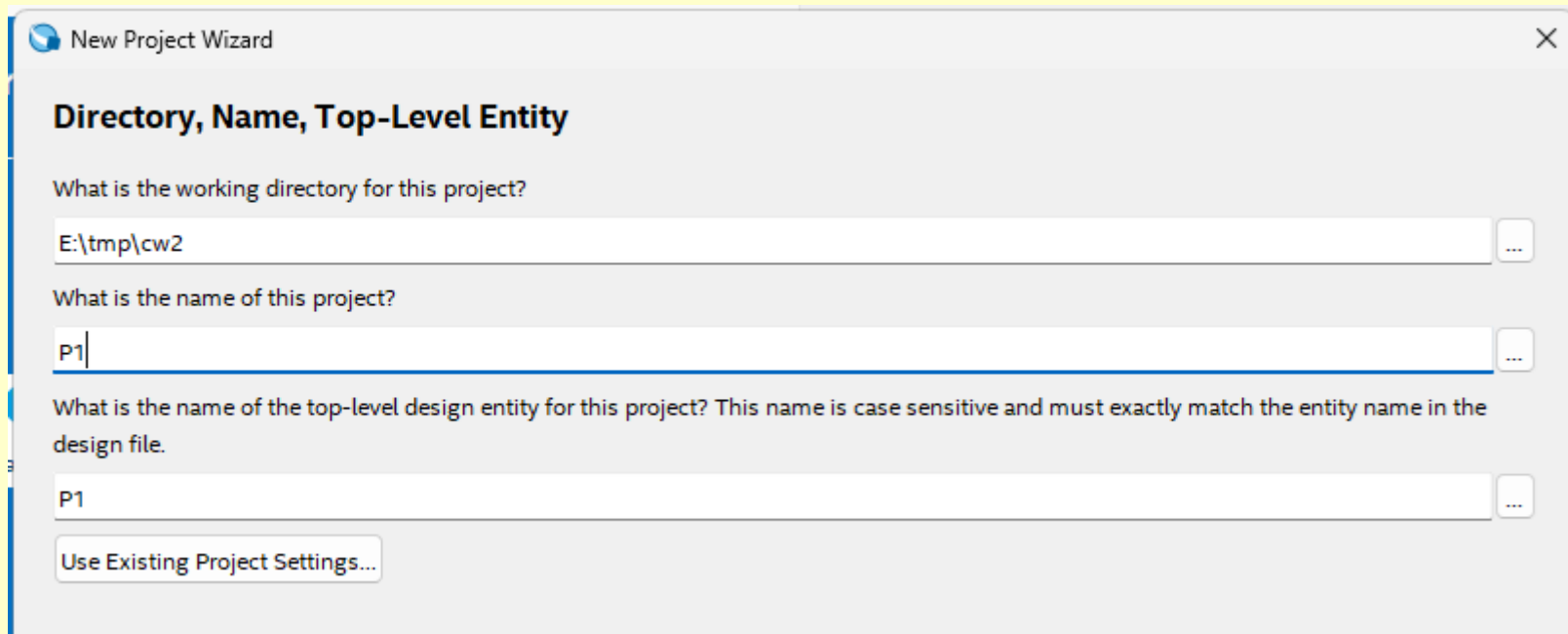
Prawidłowo zainicjowany projekt potwierdza w pierwszym wierszu informacje o: folderze roboczym, nazwie projektu i nazwie pliku top-level. Szczegółowe informacje dotyczące obsługi oprogramowania Qurtus II są podawane w trakcie zajęć laboratoryjnych – w ramach niniejszego wykładu będą przedstawione informacje uzupełniające. :



Dopiero po inicjacji projektu można otworzyć istniejący lub nowy plik źródłowy, np. plik schematu, kod źródłowy VHDL, plik inicjacji pamięci czy też plik wejściowy symulatora tzw. test bench w wersji graficznej.



W trakcie definicji nowego projektu należy podać szereg ustawień, min. należy podać nazwę pliku top-level, wskazać rodzaj docelowej FPGA, itp.



New Project Wizard

Directory, Name, Top-Level Entity

What is the working directory for this project?

E:\tmp\cw2

What is the name of this project?

P1

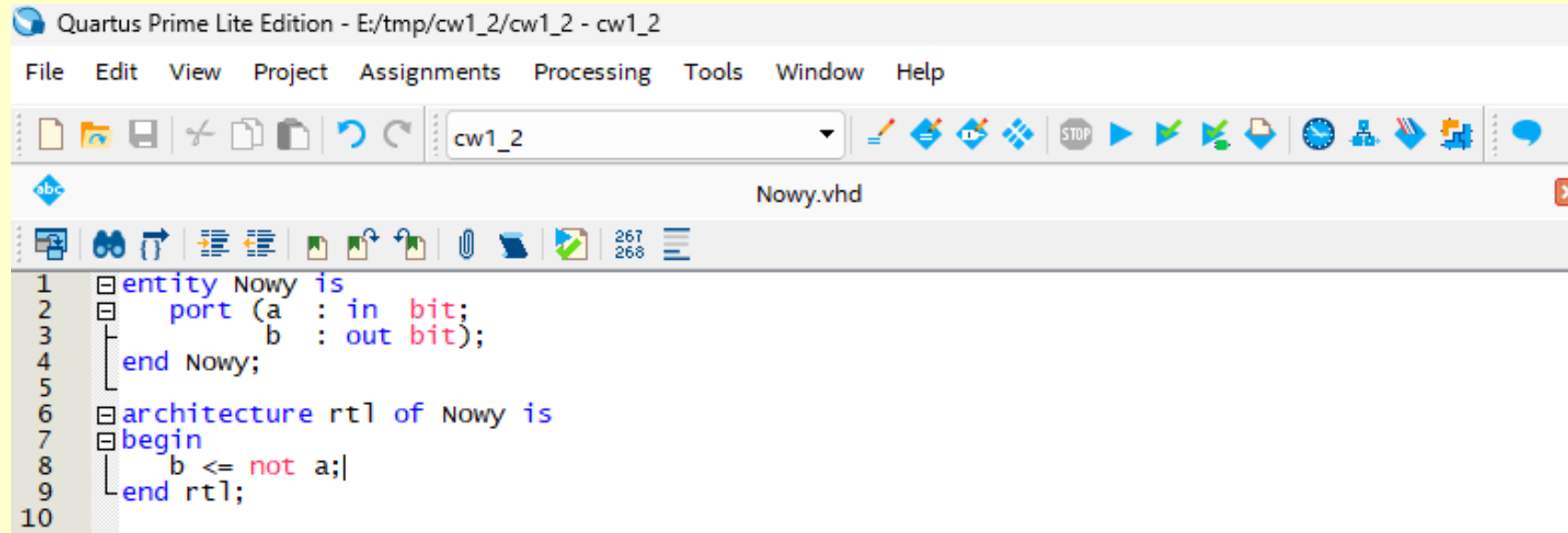
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

P1

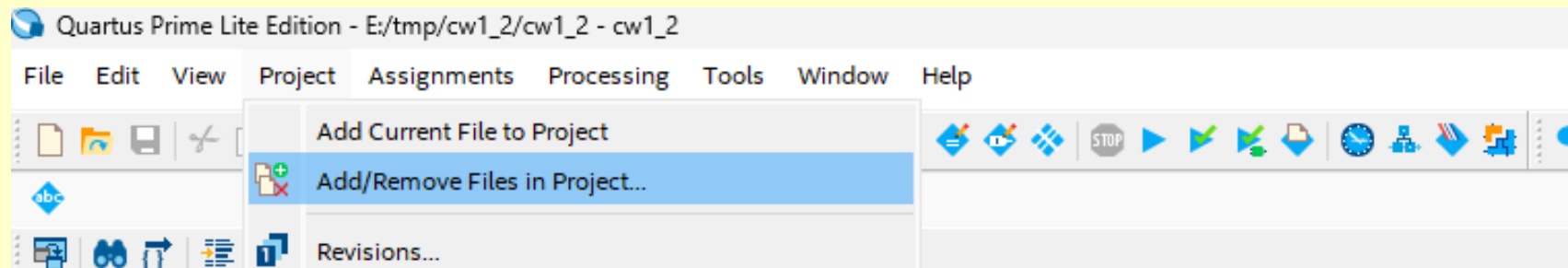
Use Existing Project Settings...

Plik top-level posiada szczególne znaczenie, to on „spina” wszystkie pliki projektu i do tego pliku odnoszą się wszystkie operacje projektowe – takie jak kompilacja, symulacja czy konfiguracja FPGA. Jeżeli plik top-level nie jest powiązany z bieżącym plikiem źródłowym, wszelkie zmiany w pliku bieżącym nie zostaną zauważone i kompilator będzie odwoływać poprzez plik top-level do związanych z nim plików. Plik top-level najczęściej jest plikiem graficznym, ale w tej roli może występować również plik VHDL. Zaleca się, żeby każdy projekt był inicjowany w oddzielnym katalogu, min. z uwagi na ustawienia symulacji.

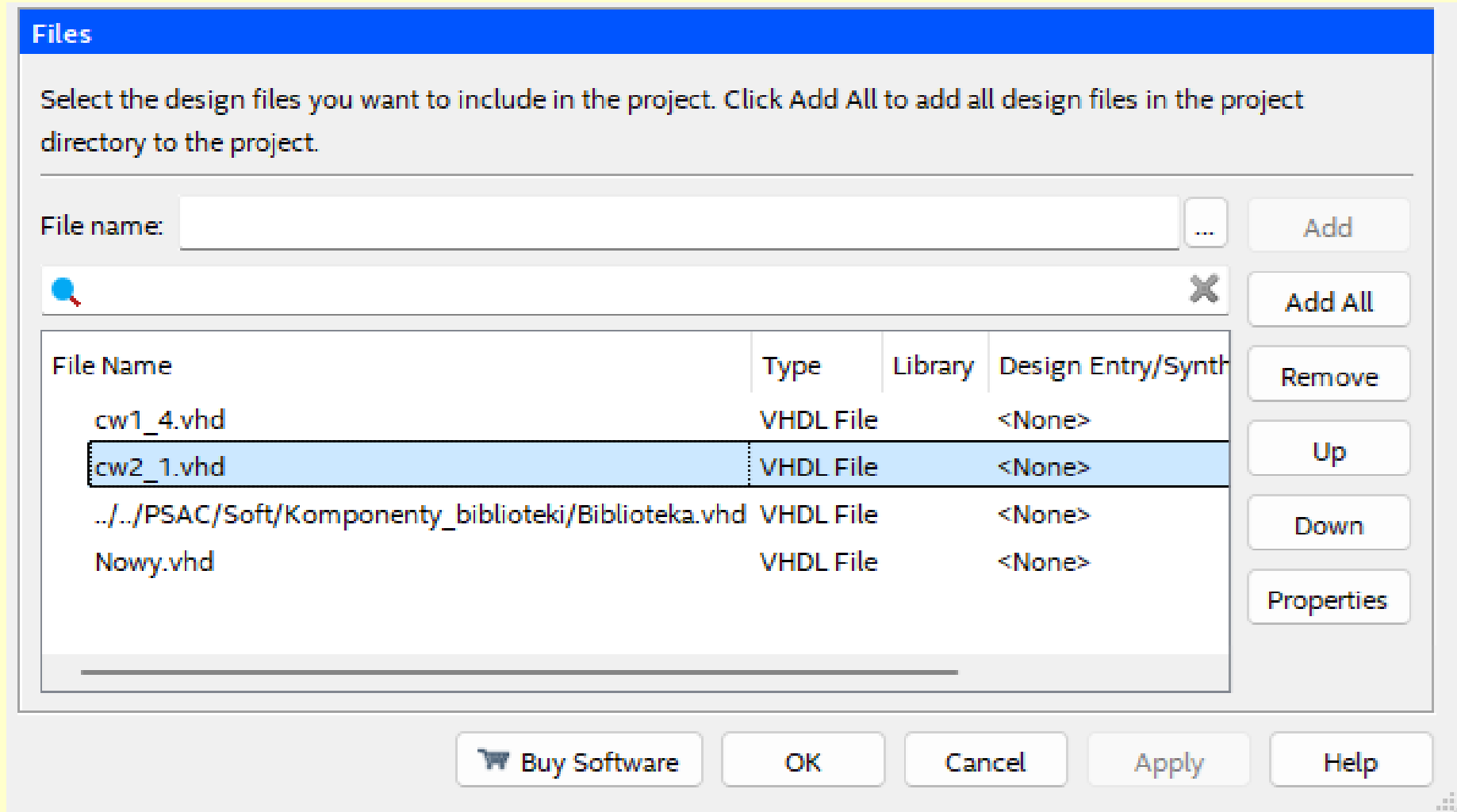
Czasami zachodzi jednak konieczność wymiany pliku top-level. W takiej sytuacji tworzymy nowy plik źródłowy. Np. chcemy zmienić plik top-level o nazwie **cw1_2** na plik **Nowy.vhd**.



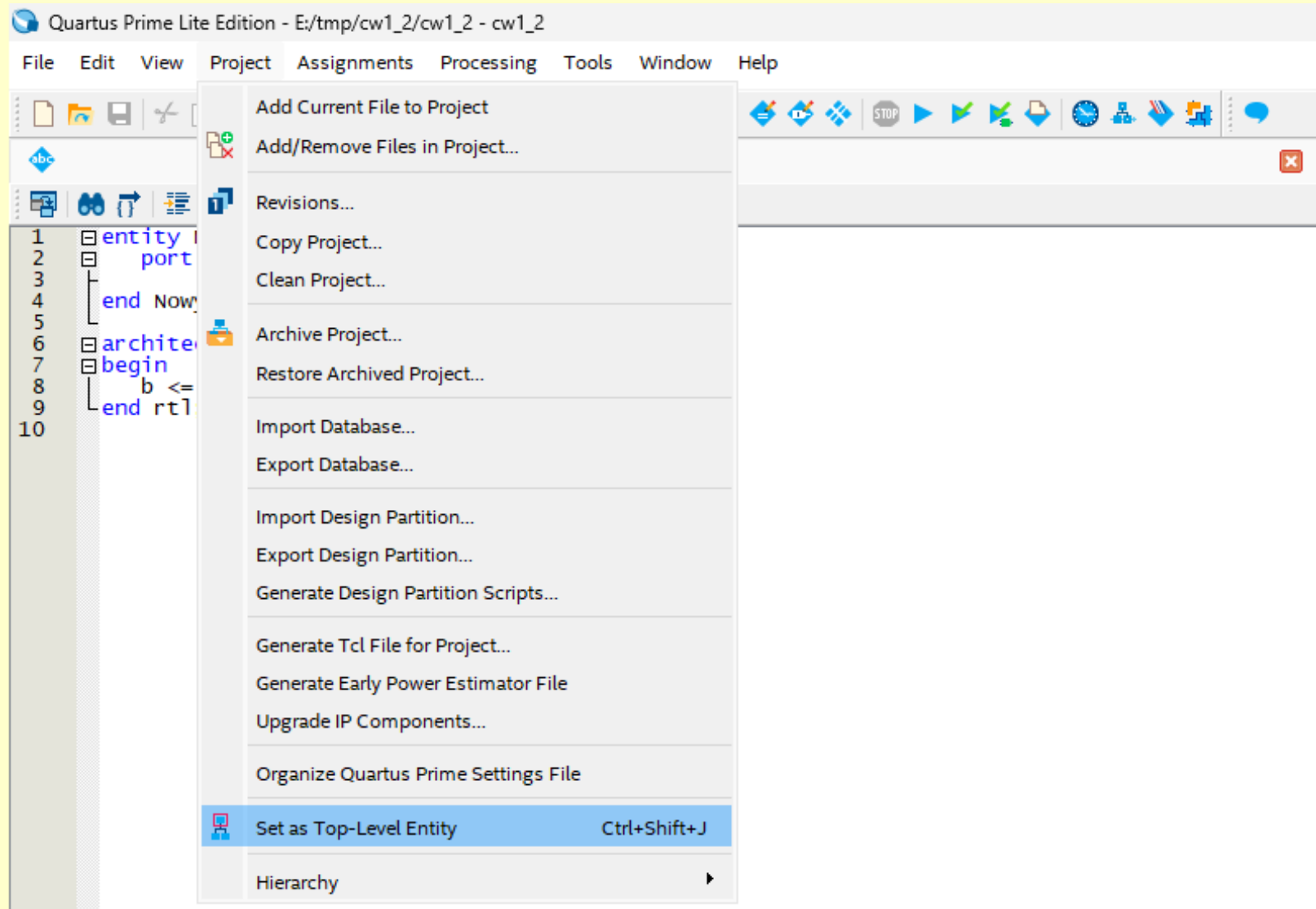
W tym celu otwieramy katalog projektu funkcją: Project → Add/Remove Files in Project... →



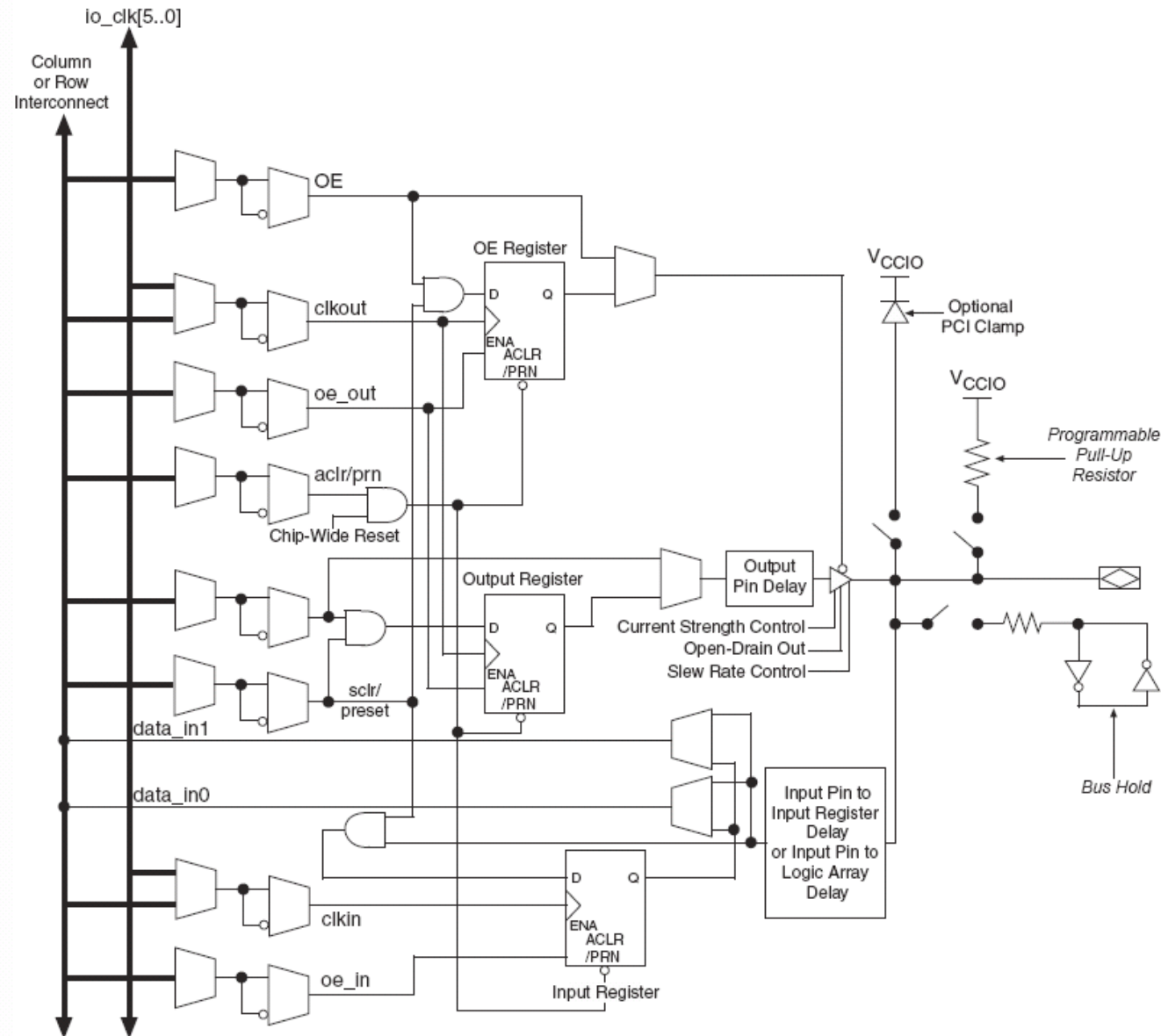
Następnie w katalogu projektu zaznaczamy stare pliki (cw1_4.vhd, cw2_1.vhd, Biblioteka.vhd) i usuwamy opcją Remove.



Na koniec wracamy do nowego pliku i z użyciem funkcji: Project → Set as Top-Level Entity, ustawiamy plik **Nowy** w roli nowego pliku top-level. Od tego momentu kompilacja będzie odnosić się do tego pliku.

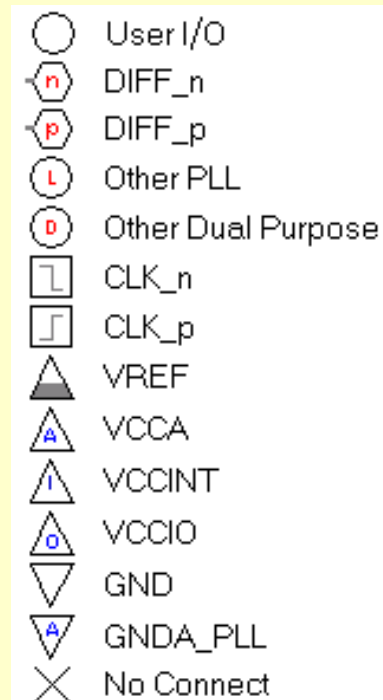
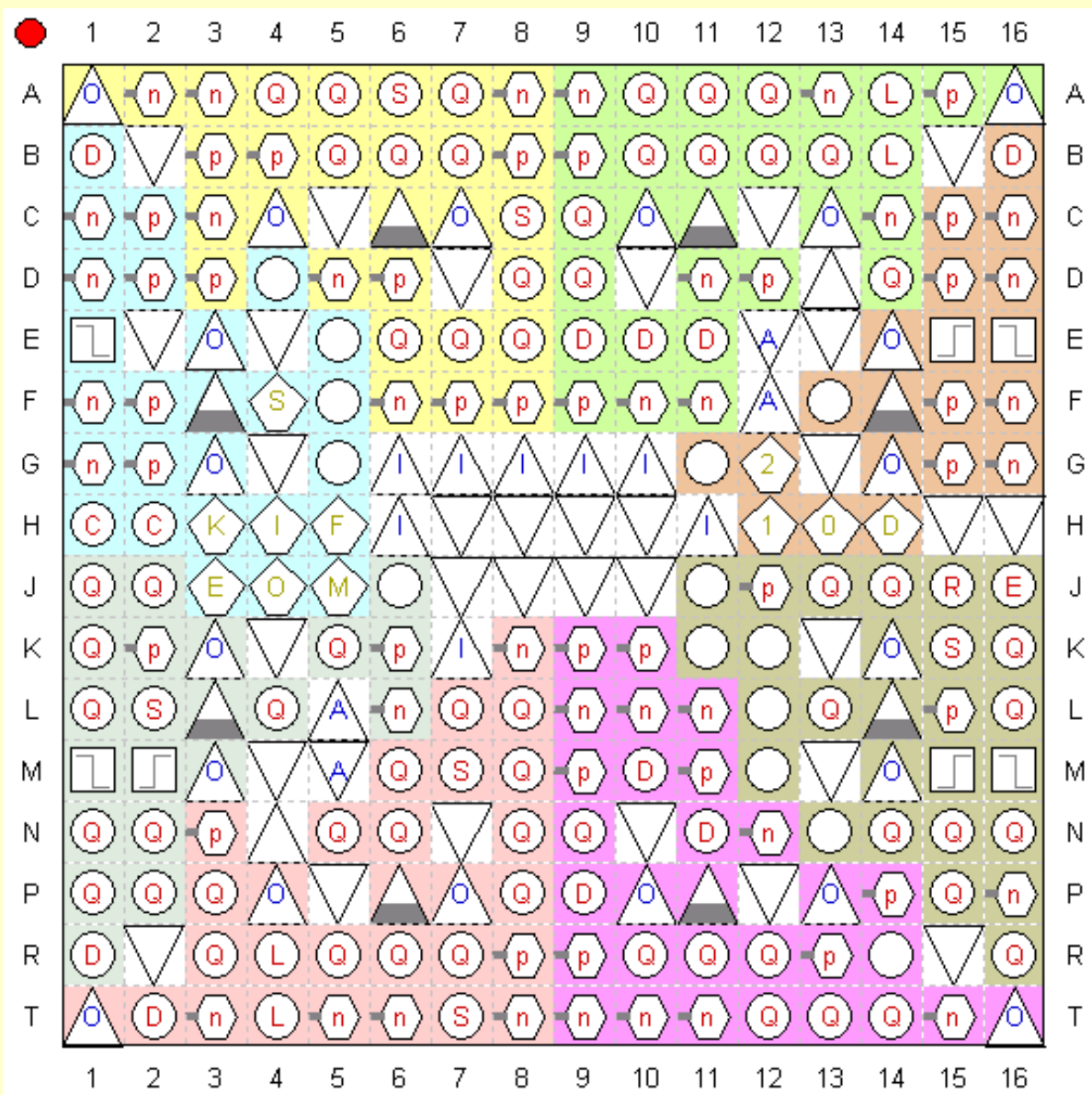


Efektywne wykorzystanie struktury FPGA wymaga odpowiedniej konfiguracji pinów I/O układu. Min.:



- obsługa standardów sygnałowych,
- programowe ograniczenie prądu,
- programowanie slew-rate impulsów,
- konfiguracja trybu open-drain,
- zatrzymywanie danych na szynie,
- włączenie diody zabezpieczającej,
- zastosowania rezystorów pull-up,
- opóźnianie sygnałów I/O,
- ustawienie preemfazy w.cz.,
- dopasowanie impedancyjne do linii.

Oznaczenia wyprowadzeń widoczne w Pin Planner:



piny użytkownika,
 pin „n” pary różnicowej,
 pin „p” pary różnicowej,
 pin wyjściowy PLL,
 pin podwójnego znaczenia,
 wejście „n” różnicowego CLK,
 wejście „p” różnicowego CLK,
 pin napięcia referencyjnego,
 + napięcia „analogowego” PLL,
 + napięcia zasilającego rdzeń,
 + napięcia zasilające bank,
 masa napięć cyfrowych,
 masa „analogowa” PLL.

Ustawiania dodatkowych parametrów pinów I/O realizujemy w dwóch modułach:

Pin Planner

Pin Planner - E:/Sprawozdania/PSL_EiT/Reklama3/Reklama - Reklama

File Edit View Processing Tools Window Help

Top View - Wire Bond
Cyclone IV E - EP4CE115F29C7

IO BANK 6
2 assigned/71 assignable

IO BANK 7
9 assigned/72 assignable (0 input/0 output/0 bi-dir)

All Pins
Named: * Edit: [X] [Y]

Node Name	Direction	Location	I/O Standard	Current Strength	Enable Bus-Hold Circuitry	Slew Rate	Output Termination
out K_Green[7]	Output	PIN_AF24	2.5 V	8mA (default)		2	Series 50 Ohm
out K_Green[6]	Output	PIN_AD19	2.5 V	8mA (default)	On	2 (default)	
out K_Green[5]	Output	PIN_AC19	2.5 V	8mA (default)		2 (default)	
out K_Green[4]	Output	PIN_AD15	2.5 V	8mA (default)		2 (default)	
out K_Green[3]	Output	PIN_AD21	2.5 V	8mA (default)		2 (default)	
out K_Green[2]	Output	PIN_AC21	2.5 V	8mA (default)		2 (default)	
out K_Green[1]	Output	PIN_AB21	2.5 V	8mA (default)		2 (default)	

Assignment Editor

Quartus Prime Lite Edition - E:/Sprawozdania/PSL_EiT/Reklama3/Reklama - Reklama

File Edit View Project Assignments Processing Tools Window Help

Reklama

Reklama.bdf Assignment Editor

<<new>> Filter on node names: *

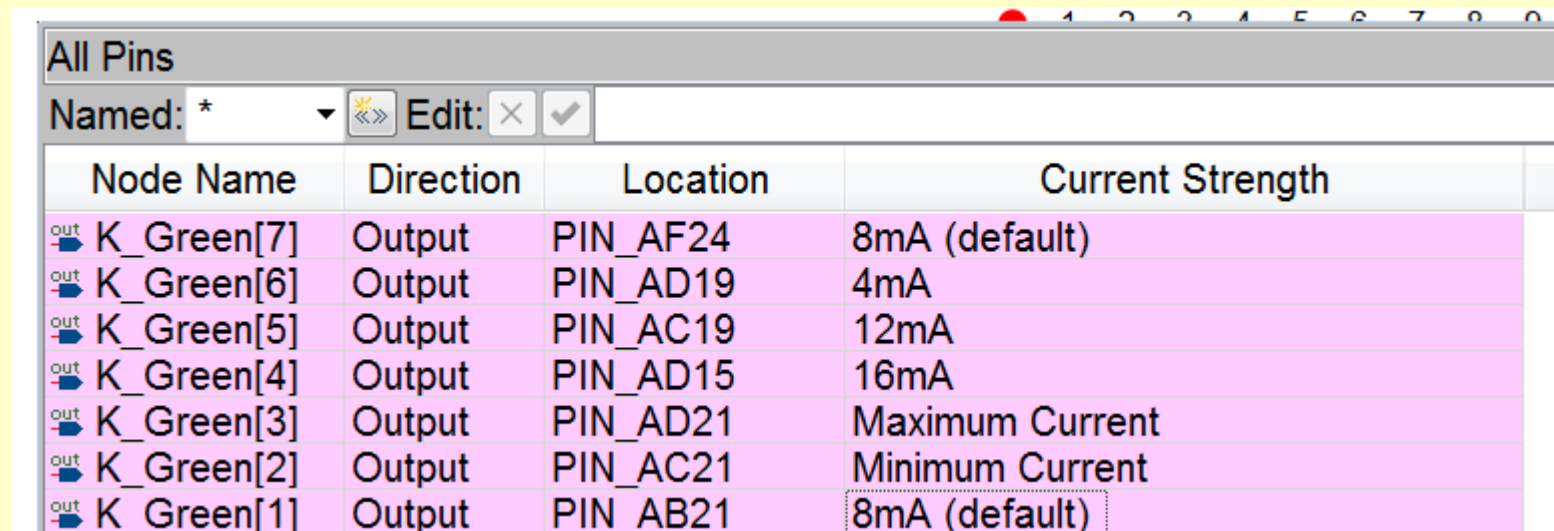
Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
44 ✓ Ok		out K_Green[7]	Slew Rate	2	Yes	Reklama		
45 ✓ Ok		out K_Green[7]	Output Termination	Series 50 Ohm	Yes	Reklama		
46 ✓ Ok		out K_Green[6]	Enable Bus-Hold Circ...	On	Yes	Reklama		
47 ✓ Ok		in Reset	Weak Pull-Up Resistor	On	Yes	Reklama		
48	<<new>>	<<new>>	<<new>>					

Ograniczanie prądu wyjściowego – **Current Strength**. Każdy z pinów wyjściowych posiada programowalne ograniczenie prądu wypływającego z wyprowadzenia, można ustalać maksymalną wartość prądu w celu:

- redukcji zakłóceń spowodowanych częstym przełączaniem wielu wyjść,
- kontroli przekroczenia maksymalnego prądu wyjściowego danego banku.

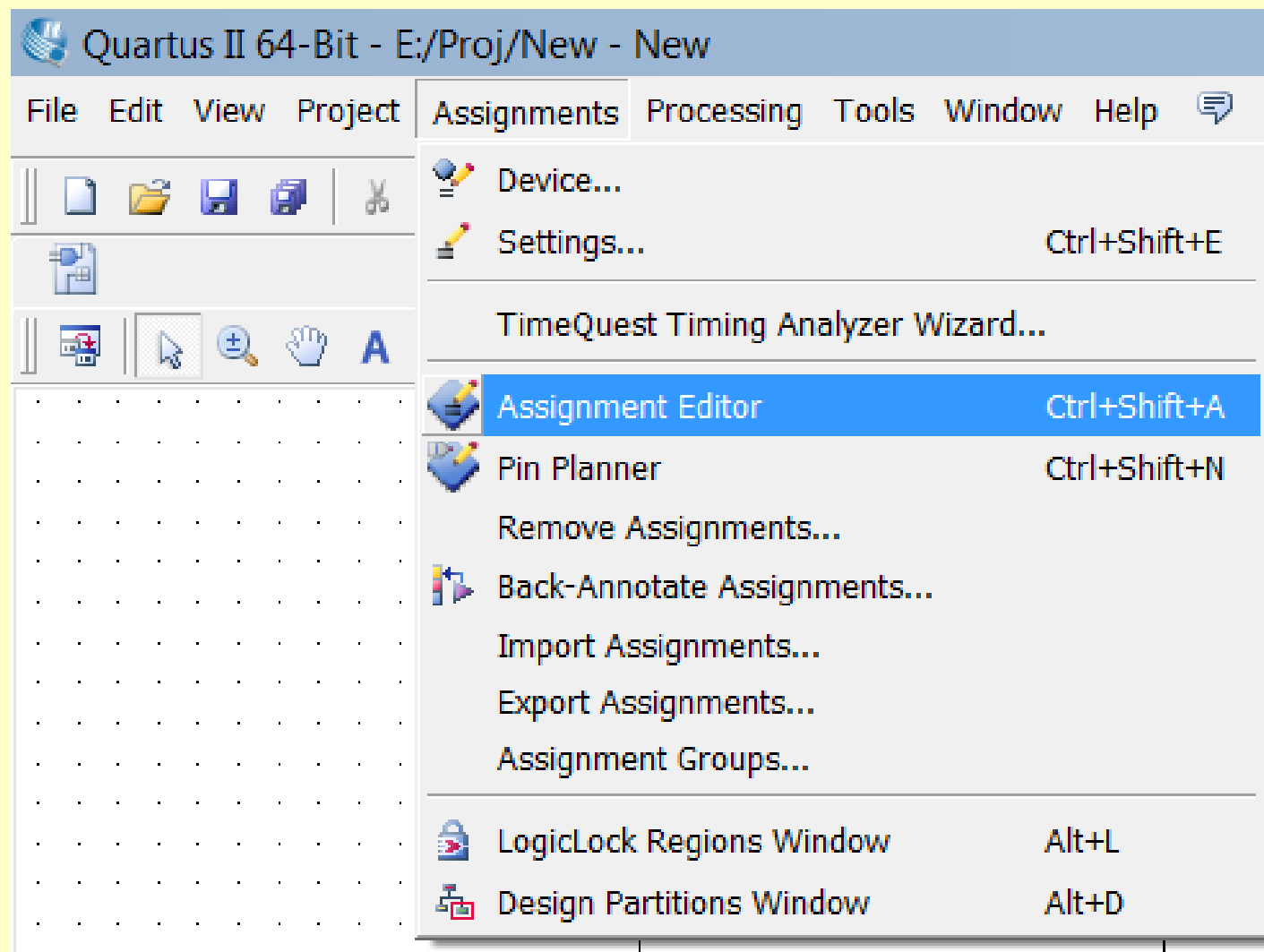
Wartość natężenia prądu wpływa na charakterystyki czasowe sygnałów impulsowych:

- częstotliwości pracy - f_{MAX} , - im wyższa częstotliwość pracy tym „uderzenie prądowe” musi być większe,
- wewnętrzne opóźnienia - t_{CO} (clock-to-output delay) i szybkość narastania/opadania zboczy,
- amplitudy przeregulowań – wybór między szybkością transmisji a przeregulowaniem,
- poziom szumów – pozwala kontrolować poziom sygnału na tle szumów,
- przeregulowanie lub niedoregulowanie sygnału – może naruszyć standardy sygnałowe.



Node Name	Direction	Location	Current Strength
out K_Green[7]	Output	PIN_AF24	8mA (default)
out K_Green[6]	Output	PIN_AD19	4mA
out K_Green[5]	Output	PIN_AC19	12mA
out K_Green[4]	Output	PIN_AD15	16mA
out K_Green[3]	Output	PIN_AD21	Maximum Current
out K_Green[2]	Output	PIN_AC21	Minimum Current
out K_Green[1]	Output	PIN_AB21	8mA (default)

Większą elastyczność ustawiania pinów niż Pin Planner posiada moduł Assignment Editor Prześledźmy jego obsługę na przykładzie parametru **Current Strength**. Wywołujemy funkcję: Assignment → Assignment Editor:



W menu modułu widzimy listę portów naszego projektu z aktualnymi ustawieniami:

Quartus II 64-Bit - E:/Proj/New - New

File Edit View Project Assignments Processing Tools Window Help

New.bdf* | Compilation Report - New

<<new>> | ☒ Filter on node names: *

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		A	Location	PIN_B22	Yes			
2	✓		Dioda_lewa	Location	PIN_E25	Yes			
3	✓		Diod...rawa	Location	PIN_J25	Yes			
4	✓		Reset	Location	PIN_P26	Yes			
5	✓		Select	Location	PIN_W26	Yes			
6	✓		Wyjscie_1	Location	PIN_AB26	Yes			
7	✓		Wyjscie_2	Location	PIN_AD26	Yes			
8	✓		Zegar	Location	PIN_U3	Yes			
9	✓		A	I/O Standard	3.3-V LVTTTL	Yes			
10	✓		Dioda_lewa	I/O Standard	3.0-V LVTTTL	Yes			
11	✓		Zegar	I/O Standard	1.5-V H...Class I	Yes			
12		<<new>>	<<new>>	<<new>>					

Na końcu listy dodajemy nowe ustawienie wybranego portu:

Quartus II 64-Bit - E:/Proj/New - New

File Edit View Project Assignments Processing Tools Window Help

New

New.bdf*

Filter on node names: *

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		in A	Location	PIN_B22	Yes			
2	✓		out Dioda_lewa	Location	PIN_E25	Yes			
3	✓		out Diod...rawa	Location	PIN_J25	Yes			
4	✓		in Reset	Location	PIN_P26	Yes			
5	✓		in Select	Location	PIN_W26	Yes			
6	✓		out Wyjscie_1	Location	PIN_AB26	Yes			
7	✓		out Wyjscie_2	Location	PIN_AD26	Yes			
8	✓		in Zegar	Location	PIN_U3	Yes			
9	✓		in A	I/O Standard	3.3-V LVTTTL	Yes			
10	✓		out Dioda_lewa	I/O Standard	3.0-V LVTTTL	Yes			
11	✓		in Zegar	I/O Standard	1.5-V H...Class I	Yes			
12		<<new>>							

Node Finder

Wybieramy

Port poddany dodatkowej konfiguracji wybieramy z listy istniejących portów (np. port A) lub wpisujemy nazwę:

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		A	Location	PIN_B22	Yes			
2	✓		Dioda_lewa	Location	PIN_E25	Yes			
3	✓		Diod...rawa	Location	PIN_J25	Yes			
4	✓		Reset	Location	PIN_P26	Yes			
5	✓		Select	Location	PIN_W26	Yes			
6	✓		Wyjscie_1	Location	PIN_AB26	Yes			
7	✓		Wyjscie_2	Location	PIN_AD26	Yes			
8	✓		Zegar	Location	PIN_U3	Yes			
9	✓		A	I/O Standard	3.3-V LVTTL	Yes			
10	✓		Dioda_lewa	I/O Standard	3.0-V LVTTL	Yes			
11	✓		Zegar	I/O Standard	1.5-V H...Class I	Yes			
12		<<new>>		<<new>>					

Node Finder

Named: *

List

Nodes Found:

Name	Assignments
New	
A	PIN_B22
Dio...ewa	PIN_E25
Dio...awa	PIN_J25
Reset	PIN_P26
Select	PIN_W26
Wyj...e_1	PIN_AB26
Wyj...e_2	PIN_AD26
Zegar	PIN_U3

Selected Nodes:

Name	Assignments
A	PIN_B22

OK

Cancel

Klikając pole „Assgnment Name” otwieramy listę możliwych ustawień i wybieramy parametr „Current Strength”:

Quartus II 64-Bit - E:/Proj/New - New

File Edit View Project Assignments Processing Tools Window Help

New

Compilation Report - New

<<new>> Filter on node names: *

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		A	Location	PIN_B22	Yes			
2	✓		Dioda_lewa	Location	PIN_E25	Yes			
3	✓		Diod...rawa	Location	PIN_J25	Yes			
4	✓		Reset	Location	PIN_P26	Yes			
5	✓		Select	Location	PIN_W26	Yes			
6	✓		Wyjscie_1	Location	PIN_AB26	Yes			
7	✓		Wyjscie_2	Location	PIN_AD26	Yes			
8	✓		Zegar	Location	PIN_U3	Yes			
9	✓		A	I/O Standard	3.3-V LVTTTL	Yes			
10	✓		Dioda_lewa	I/O Standard	3.0-V LVTTTL	Yes			
11	✓		Zegar	I/O Standard	1.5-V H...Class I	Yes			
12	!		A						
13		<<new>>	<<new>>						

Automatic Asynchronous Signal Pipelining Register Reach

Block Design Naming (Accepts wildcards/groups)

CKn/CK Pair (Accepts wildcards/groups)

CLKLOCKx1 Input Frequency

Carry Chain Length (Accepts wildcards/groups)

Current Strength (Accepts wildcards/groups)

D5 Output-Enable Delay (output-enable register to io buffer) (Accepts wildcards/groups)

DCLK Pin

DQ Group (Accepts wildcards/groups)

Sets the drive strength of a pin. Specify a number (in mA), MIN, or MAX for output or bidirectional pins that support programmable drive strength. Please refer to the family data sheet

rozwijamy listę parametrów

Rozwijając kolumnę „Value” wybieramy oczekiwaną wartość natężenia prądu wyjściowego:

Quartus II 64-Bit - E:/Proj/New - New

File Edit View Project Assignments Processing Tools Window Help

New

New.bdf* Compilation Report - New

<<new>> Filter on node names: *

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		in A	Location	PIN_B22	Yes			
2	✓		out Dioda_lewa	Location	PIN_E25	Yes			
3	✓		out Diod...rawa	Location	PIN_J25	Yes			
4	✓		in Reset	Location	PIN_P26	Yes			
5	✓		in Select	Location	PIN_W26	Yes			
6	✓		out Wyjscie_1	Location	PIN_AB26	Yes			
7	✓		out Wyjscie_2	Location	PIN_AD26	Yes			
8	✓		in Zegar	Location	PIN_U3	Yes			
9	✓		in A	I/O Standard	3.3-V LVTTL	Yes			
10	✓		out Dioda_lewa	I/O Standard	3.0-V LVTTL	Yes			
11	✓		in Zegar	I/O Standard	1.5-V H...Class I	Yes			
12	!		in A	Current Strength		Yes	New		
13		<<new>>	<<new>>	<<new>>					

10mA

12mA

14mA

16mA

2mA

4mA

6mA

8mA

Maxi...rent

Widzimy dodany nowy parametr dla portu A. Procedurę powtarzamy dodając inne parametry dla innych portów.

Quartus II 64-Bit - E:/Proj/New - New

File Edit View Project Assignments Processing Tools Window Help

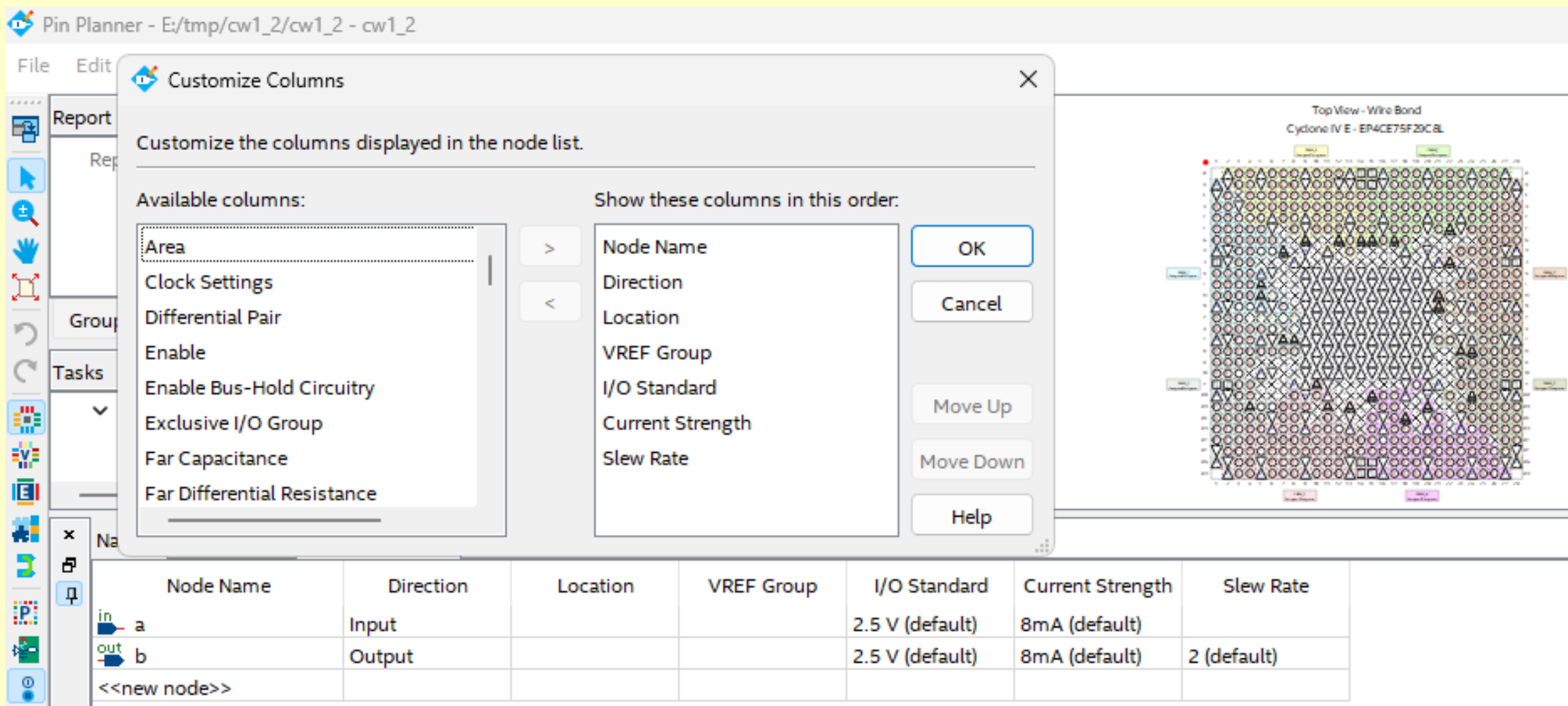
New

Compilation Report - New

<<new>> ☒ Filter on node names: *

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		 A	Location	PIN_B22	Yes			
2	✓		 Dioda_lewa	Location	PIN_E25	Yes			
3	✓		 Diod...rawa	Location	PIN_J25	Yes			
4	✓		 Reset	Location	PIN_P26	Yes			
5	✓		 Select	Location	PIN_W26	Yes			
6	✓		 Wyjscie_1	Location	PIN_AB26	Yes			
7	✓		 Wyjscie_2	Location	PIN_AD26	Yes			
8	✓		 Zegar	Location	PIN_U3	Yes			
9	✓		 A	I/O Standard	3.3-V LVTTL	Yes			
10	✓		 Dioda_lewa	I/O Standard	3.0-V LVTTL	Yes			
11	✓		 Zegar	I/O Standard	1.5-V H...Class I	Yes			
12	✓		 A	Current Strength	10mA	Yes	New		
13		<<new>>	<<new>>	<<new>>					

Tylko niektóre parametry można ustawiać w Pin Planner. Listę kolumn widocznych w module można rozszerzyć z pomocą funkcji Customize Columns, która jest aktywowana prawym kliknięciem na poziomie listy portów.



Parametr **Open Drain** ustawiamy z użyciem Assignment Editor:

Quartus Prime Lite Edition - E:/tmp/cw1_2/cw1_2 - cw1_2

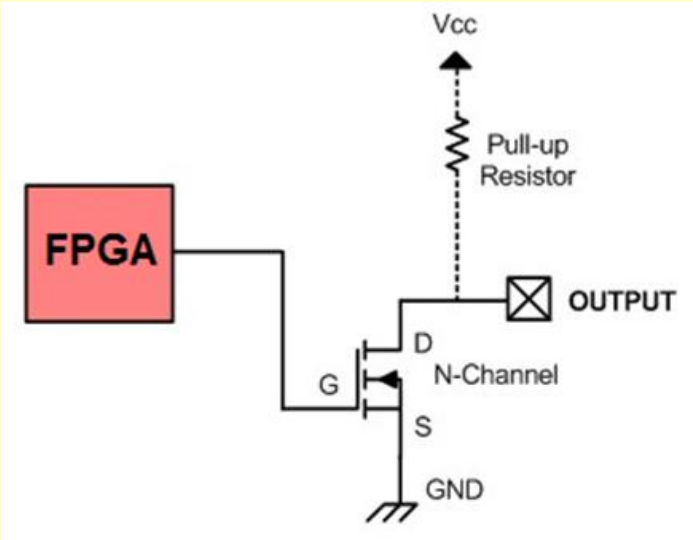
File Edit View Project Assignments Processing Tools Window Help

Nowy.vhd

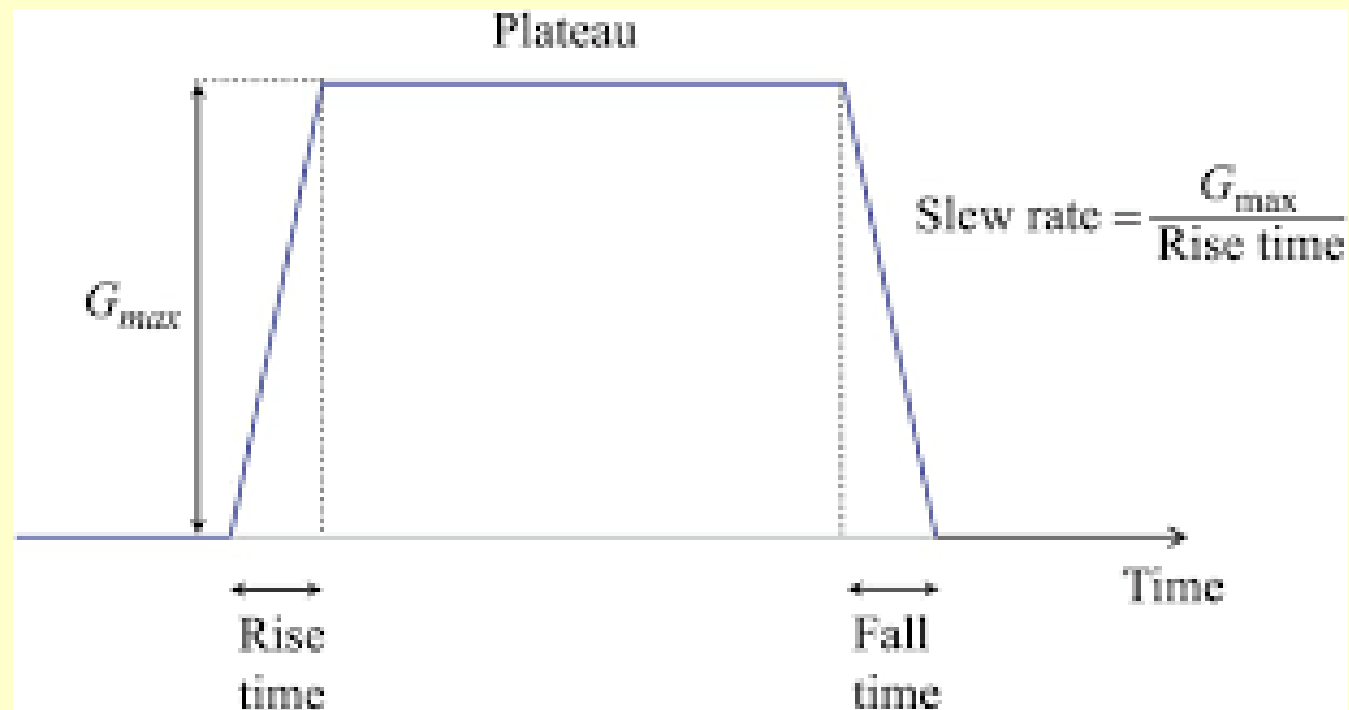
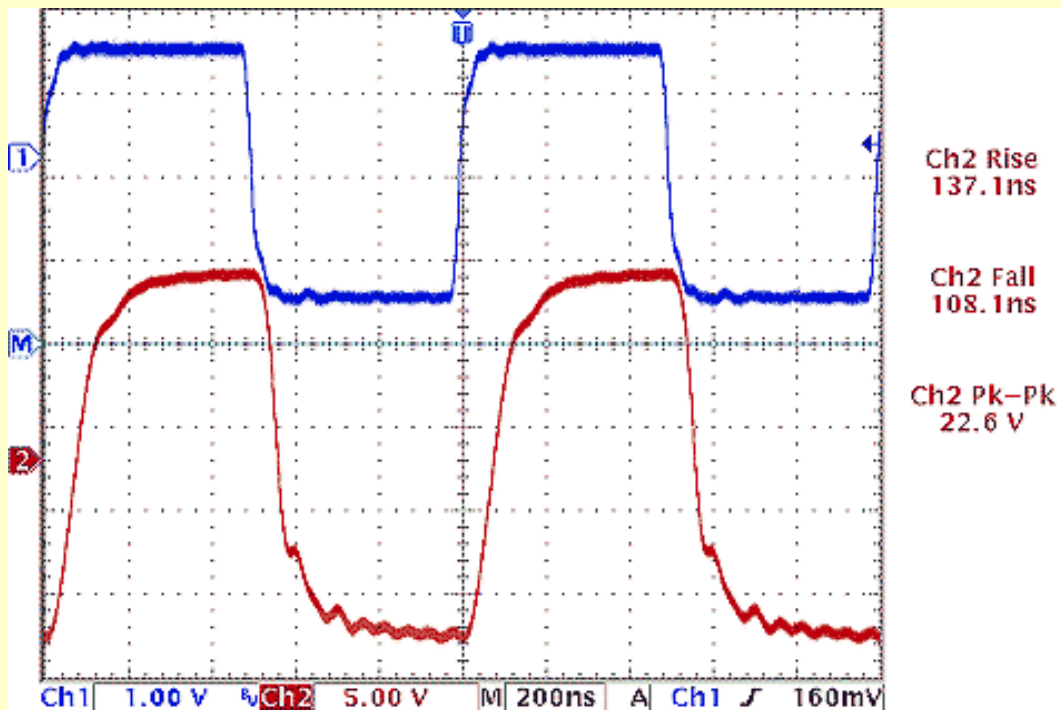
Compilation Report - cw1_2

<<new>> Filter on node names: *

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		in a	Location	PIN_A10	Yes			
2	✓		out b	Location	PIN_B21	Yes			
3	!		in a	Auto Open-Drain Pins	On	Yes	Nowy		
4		<<new>>	<<new>>	<<new>>					



Parametr **Slew Rate** ustawiamy z użyciem Assignment Editor. Jest to użyteczny parametr projektowania warunków pracy szybkiej lub niskoszumowej. Wyższe SR to krótszy przedział czasowy i zwiększenie przepustowości transmisji – niższe redukuje szumy, ale wydłuża czasy trwania zboczy impulsów. Szybkość narastania sygnału wyjściowego decyduje o kształcie sygnałów wyjściowych, w tym o: częstotliwości granicznej f_{MAX} , czasie narastania oraz opadania, poziomie szumów przełączania. Ustawiona wartość parametru oddziałuje na charakterystyki czasowe obu zboczy.



Quartus II domyślnie posiada ustawioną największą wartość współczynnika slew rate. Jeżeli współczynnik szumów jest parametrem krytycznym w układzie, należy wybierać niższą wartość współczynnika slew rate – spośród:

- 0 - slow,
- 1 - medium,
- 2 - fast.

Dokładne wartości parametru SR znajdują się w szczegółowych danych katalogowych konkretnego układu FPGA. Wartość parametru wybieramy w Assignments Editor.

Quartus Prime Lite Edition - E:/Tmp/test_1 - test_1

File Edit View Project Assignments Processing Tools Window Help

test_1

test_1.bdf Vhdl1.vhd Assignment Editor*

<<new>> Filter on node names: *

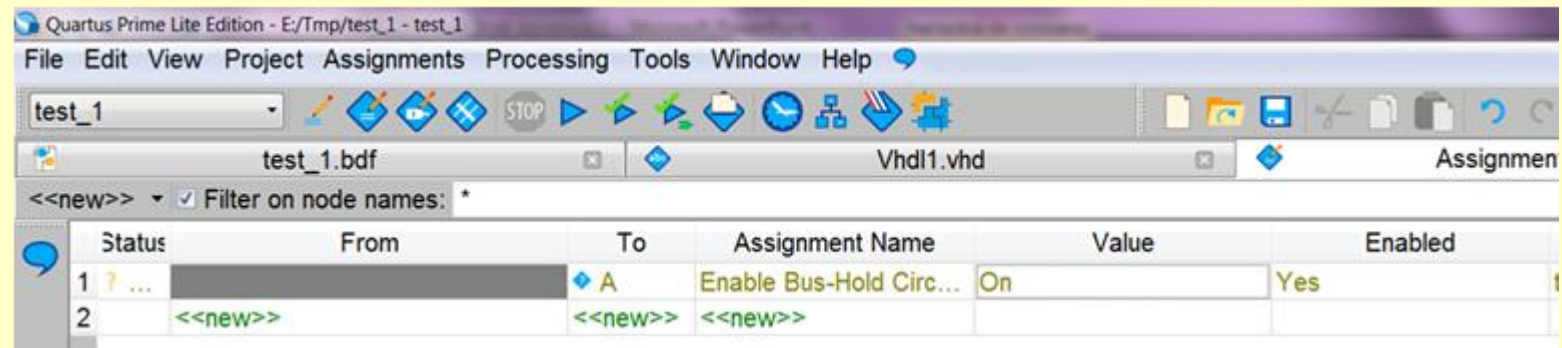
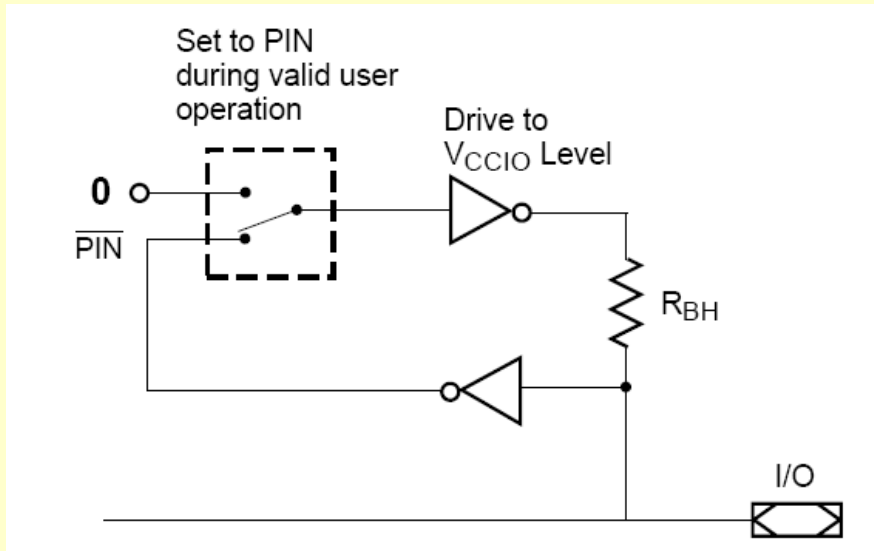
	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	! ...		A	Slew Rate	1	Yes	test_1		
2		<<new>>	<<new>>	<<new>>					

Mechanizm zatrzaśnięcia sygnału na szynie **Bus Hold** wykorzystywany jest w dwóch przypadkach:

- w celu podtrzymania danych aż pojawią się nowe ich wartości – zanim zostaną zapamiętane w układzie,
- pomocniczo, w celu utrzymania stałego poziomu napięcia w obecności silnych zakłóceń.

Mechanizm jest ustawiany selektywnie w stosunku do każdego pinu typu I/O, nie jest on jedynie możliwy:

- w przypadku pinów konfiguracyjnych - za pomocą których programujemy układ,
- w przypadku pinów zegarowych – sygnału nie wolno blokować,
- w przypadku standardów różnicowych – gdy występuje para linii przesyłających sygnał.



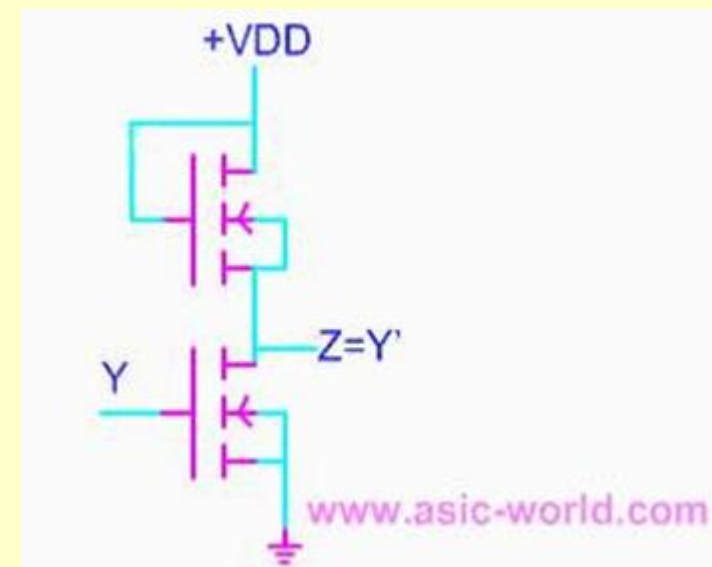
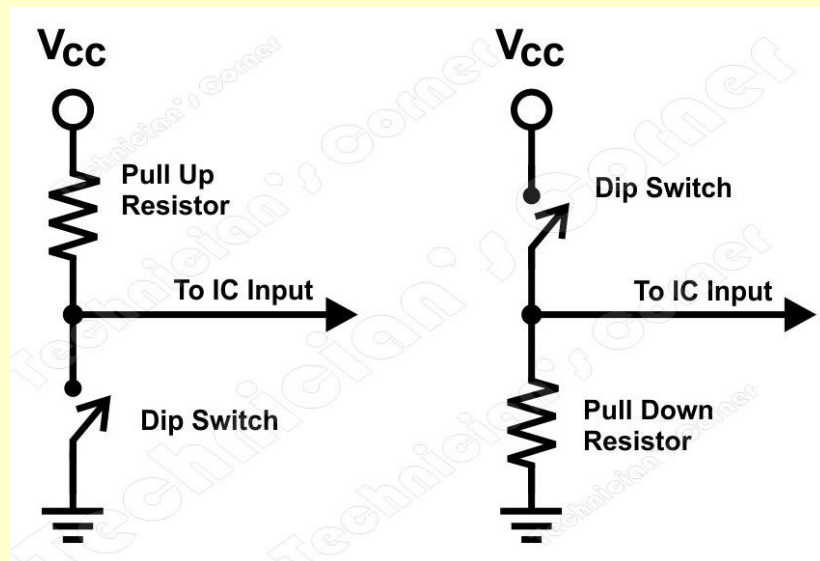
Mechanizm **Pull-Up/Down Resistor** polega na dołączeniu wewnętrznego rezystora do napięcia VCCIO lub masy.

Parametr jest ustawiany w Assignments Editor i stosowany:

- podczas pracy trójstanowej w celu podciągnięcia wyjścia do poziomu VCCIO,
- w celu zastąpienia zewnętrznego rezystora podciągającego do zasilania,
- w wyjściach typu open drain,
- w sytuacji silnych zakłóceń na wejściach układu scalonego.

Wewnętrzny rezystor podciągający wrażliwy jest na zmiany temperatury i napięć zasilających, jest to element półprzewodnikowy – rezystancja kanału tranzystora.

W układach wrażliwych na zmianę wartości rezystancji zaleca się element zewnętrzny.



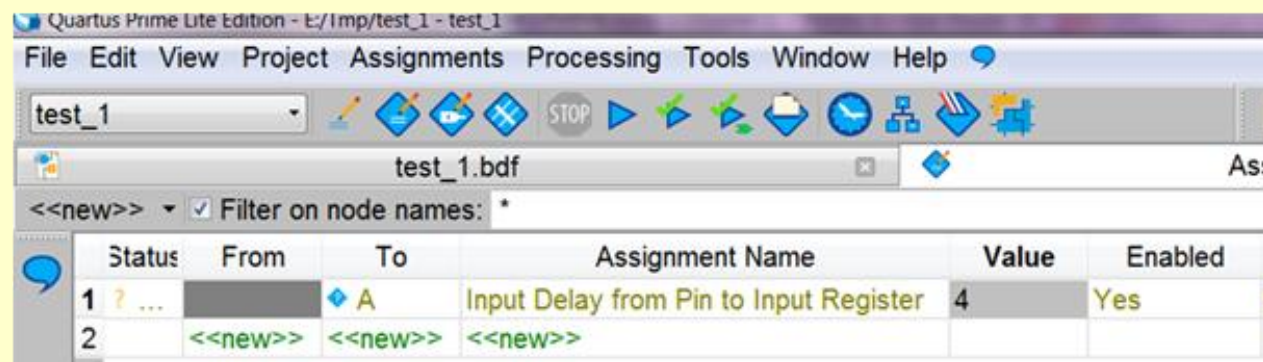
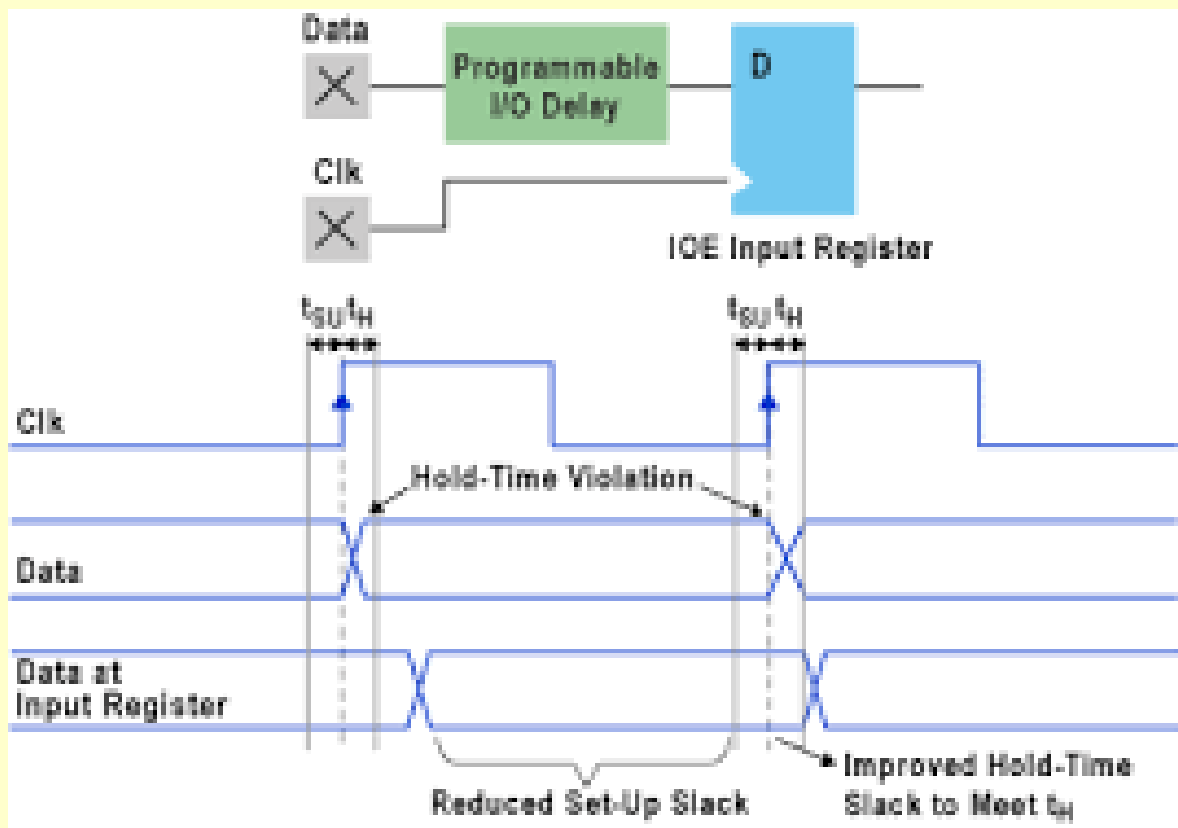
Wartości rezystorów wewnętrznych zależą od napięć zasilający banki V_{CCIO} – np. w Cyclone IV:

Table 1–12. Internal Weak Pull-Up and Weak Pull-Down Resistor Values for Cyclone IV Devices

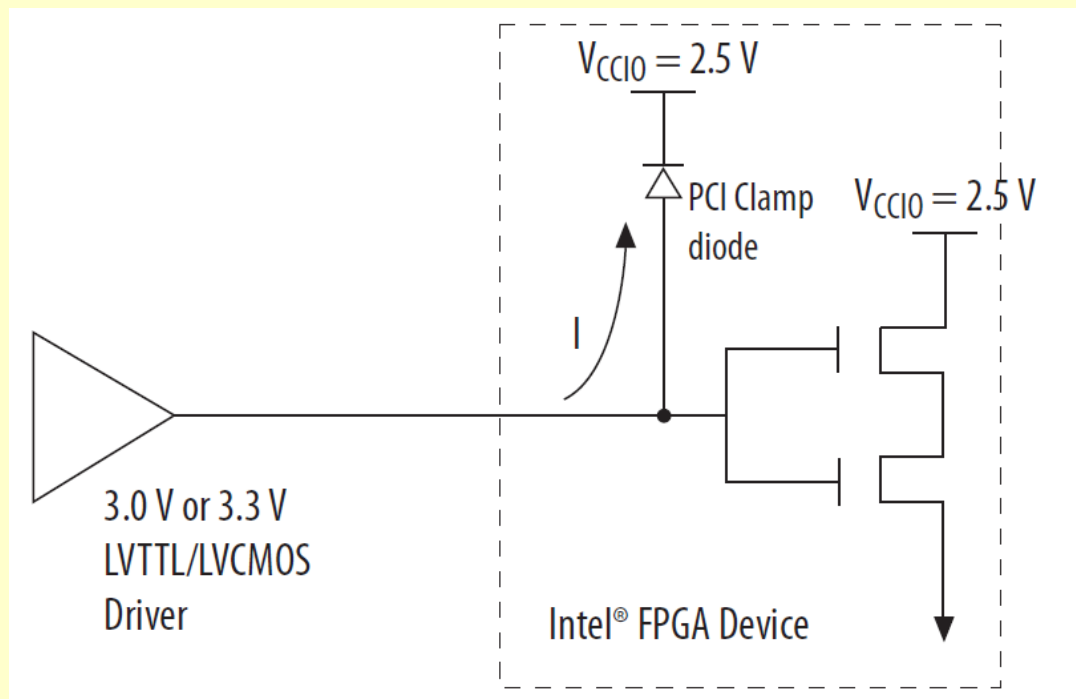
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
R _{PU}	Value of the I/O pin pull-up resistor before and during configuration, as well as user mode if you enable the programmable pull-up resistor option	$V_{CCIO} = 3.3\text{ V} \pm 5\%$	7	25	41	k Ω
		$V_{CCIO} = 3.0\text{ V} \pm 5\%$	7	28	47	k Ω
		$V_{CCIO} = 2.5\text{ V} \pm 5\%$	8	35	61	k Ω
		$V_{CCIO} = 1.8\text{ V} \pm 5\%$	10	57	108	k Ω
		$V_{CCIO} = 1.5\text{ V} \pm 5\%$	13	82	163	k Ω
		$V_{CCIO} = 1.2\text{ V} \pm 5\%$	19	143	351	k Ω
R _{PD}	Value of the I/O pin pull-down resistor before and during configuration	$V_{CCIO} = 3.3\text{ V} \pm 5\%$	6	19	30	k Ω
		$V_{CCIO} = 3.0\text{ V} \pm 5\%$	6	22	36	k Ω
		$V_{CCIO} = 2.5\text{ V} \pm 5\%$	6	25	43	k Ω
		$V_{CCIO} = 1.8\text{ V} \pm 5\%$	7	35	71	k Ω
		$V_{CCIO} = 1.5\text{ V} \pm 5\%$	8	50	112	k Ω

Układy FPGA posiadają możliwość programowania opóźnień sygnałów z użyciem Programmable Delay w celu:

- minimalizacji opóźnień czasu zatrzaśnięcia,
- minimalizacji opóźnień czasu ustawiania,
- regulacji opóźnienia sygnału zegarowego w stosunku do danych wejściowych,
- opóźniania wyjściowego sygnału zegarowego.



- Układy Cyclone posiadają możliwość programowalnego dołączenia diod wyrównujących **PCI Clamping Diode** dla każdego pinu wejściowego i wyjściowego I/O. Stosuje się je w celu:
 - „ucinania” napięć wyższych o około 0,7V od napięcia V_{CCIO} ,
 - zabezpieczenia pinów przed przekroczeniem granicznych wartości napięć AC i DC,
 - organizacji współpracy układów Cyclone z wyższymi standardami napięciowymi,
 - dioda działa skutecznie dla napięć zasilających banki 2.5 V i 3.0 V,
 - wejściowe sygnały są obcinane na poziomie V_{CCIO} zasilające bank plus spadek napięcia na diodzie.
- Mechanizm jest dostępny tylko dla niektórych standardów sygnałowych: LVTTTL, LVCMOS, PCI.



Zazwyczaj z układem *clamping diode* współpracuje rezystor szeregowy ograniczający prąd diody do jej maksymalnej wartości 10 mA.

Quartus Prime Lite Edition - E:/Tmp/test_1 - test_1

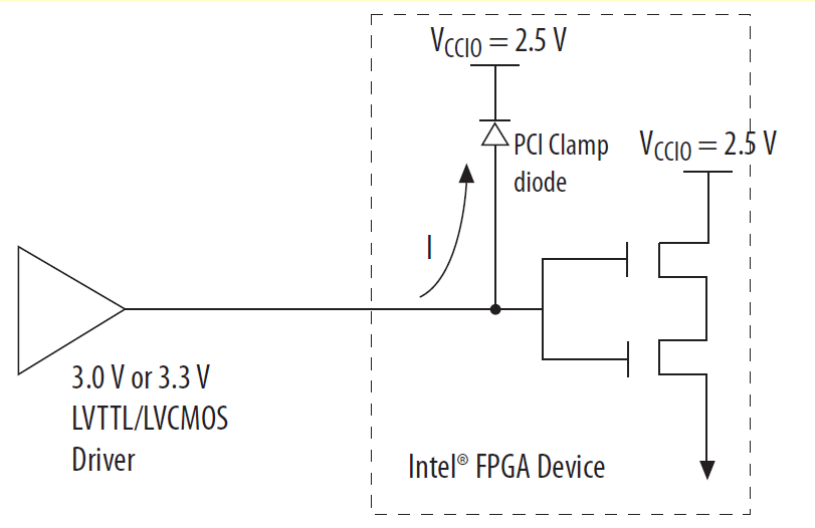
File Edit View Project Assignments Processing Tools Window Help

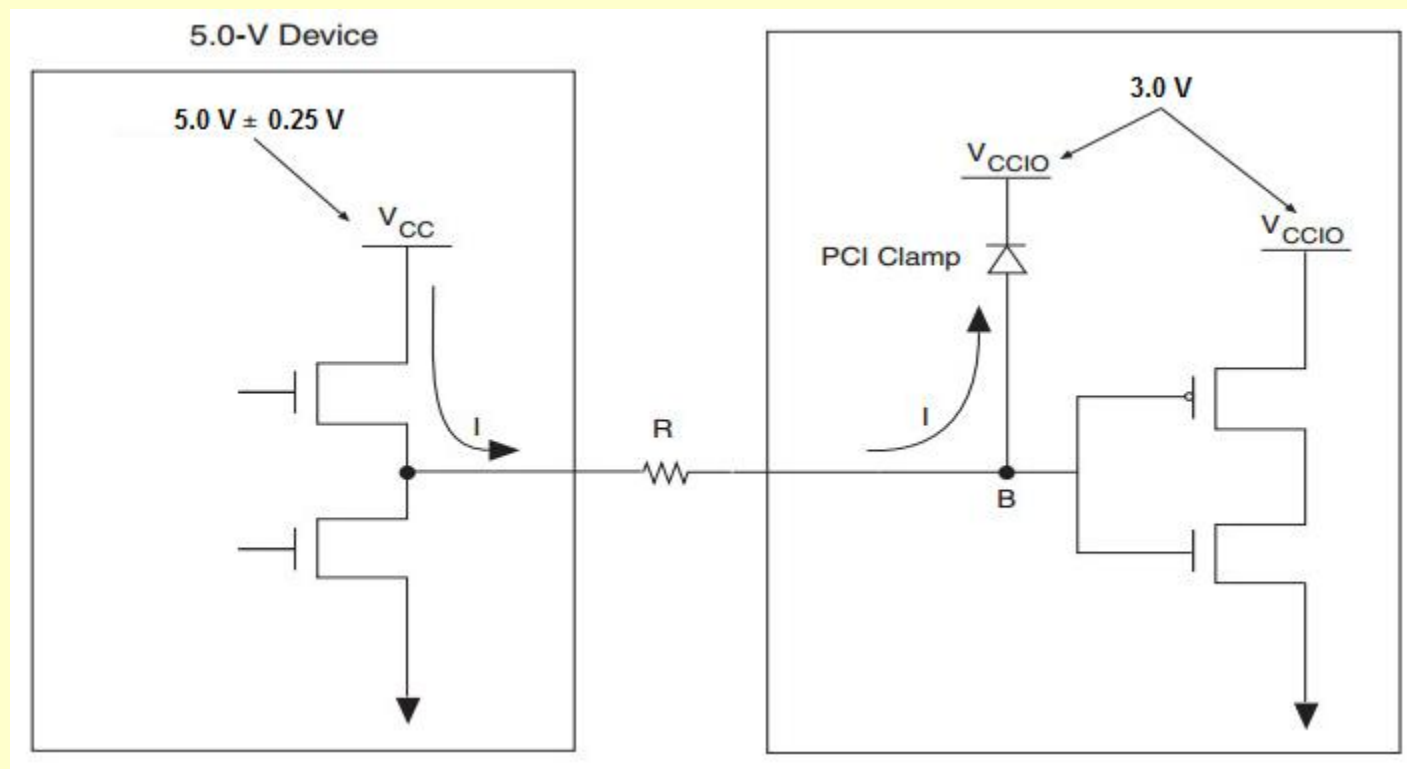
test_1

test_1.bdf Assignment Editor*

<<new>> Filter on node names: *

	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓ Ok		In	Location	PIN_A3	Yes			
2	✓ Ok		Out_1	Location	PIN_A4	Yes			
3	✓ Ok		Out_2	Location	PIN_A7	Yes			
4	✓ Ok		Out_3	Location	PIN_A8	Yes			
5	✓ Ok		Out_4	Location	PIN_A11	Yes			
6	✓ Ok		In	PCI I/O	On	Yes	test_1		
7		<<new>>	<<new>>	<<new>>					



Przykład:

$$U_{CC5V_{max}} = 5.0 \text{ V} + 0.25 \text{ V} = 5.25 \text{ V},$$

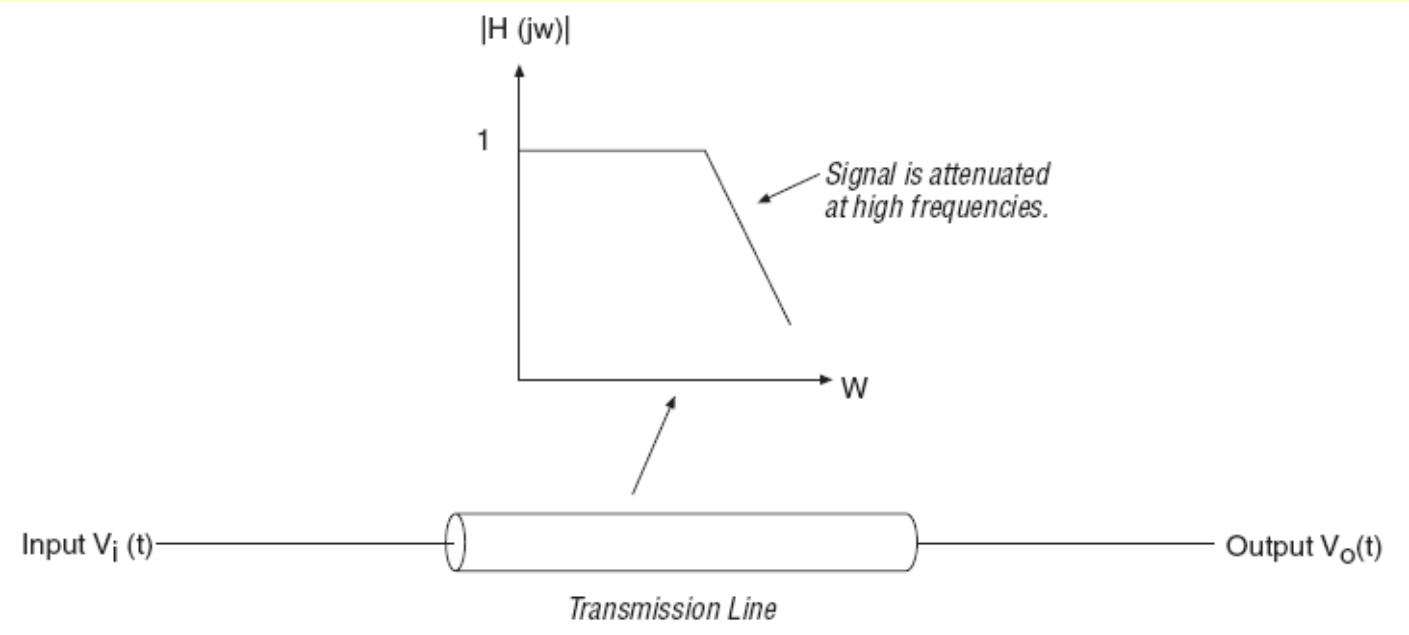
$$V_{IN} = V_{CCIO} + 0.7 \text{ V} = 3.7 \text{ V},$$

$$R_{OUT} = V_{CC5V} / I_{OH_{max}} = 5 \text{ V} / 135 \text{ mA} = 30 \Omega,$$

$$I_{OH} = 8 \text{ mA},$$

$$R = ((U_{CC5V_{max}} - V_{IN}) - (I_{OH} \cdot R_{OUT})) / I_{OH} \\ = 164 \Omega$$

Każdy tor transmisyjny ogranicza pasmo częstotliwościowe w zakresie w.cz. W standardach szybkiej transmisji – np. LVDS istnieje możliwość korekty/podbicia częstotliwościowej sygnału wewnątrz układu (preemfazy), tak aby po przejściu przez linię transmisyjną wpływ jej tłumienia na w.cz. był jak najmniejszy.



test_1.bdf

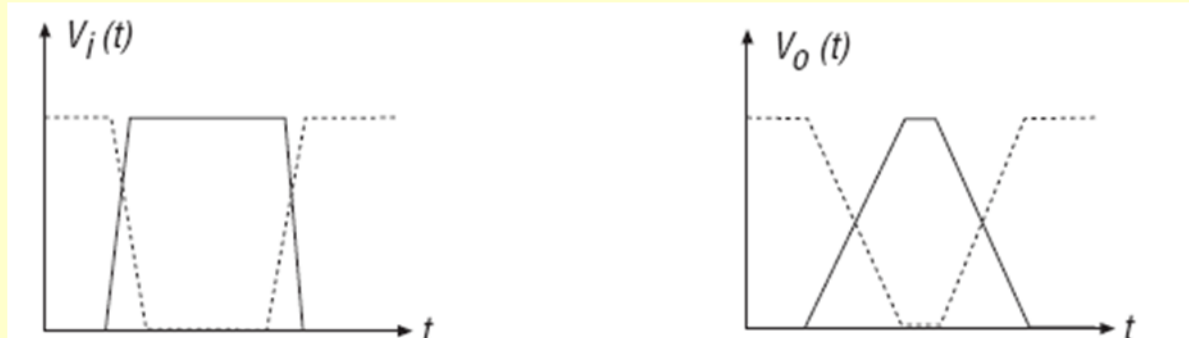
Assignment Editor*

<<new>> ☒ Filter on node names: *

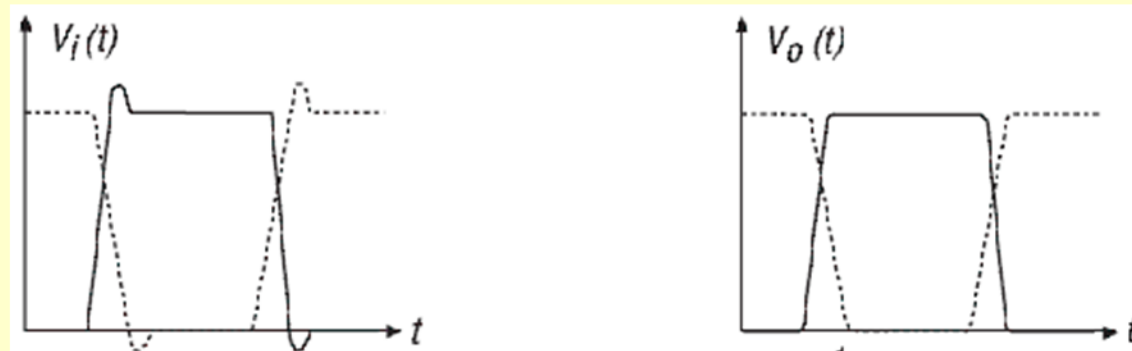
	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓ Ok		 In	Location	PIN_A3	Yes			
2	✓ Ok		 Out_1	Location	PIN_A4	Yes			
3	✓ Ok		 Out_2	Location	PIN_A7	Yes			
4	✓ Ok		 Out_3	Location	PIN_A8	Yes			
5	✓ Ok		 Out_4	Location	PIN_A11	Yes			
6	✓ Ok		 In	Programmable Pre-emphasis	1	Yes	test_1		
7		<<new>>	<<new>>	<<new>>					

Przypadek idealny:

Sygnał w torze bez preemfazy:

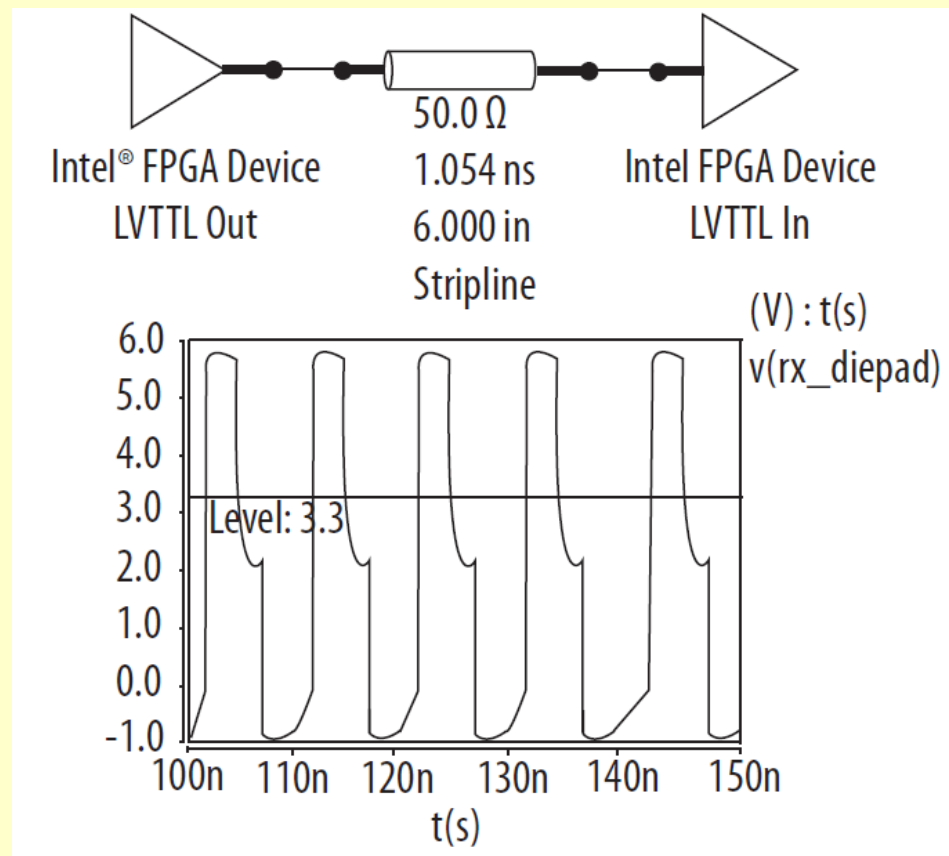


Sygnał w torze z preemfazą:



Korekcji preemfazy dokonuje się po zmontowaniu układu, gdy znane są warunki transmisyjne, niepotrzebnie włączona **preemfaza** tłumi niższe częstotliwości.

Podczas szybkich transmisji, wskutek niedopasowania impedancyjnego wyjść FPGA do parametrów linii transmisyjnej, może powstać sygnał odbity, przekraczający zabezpieczenia FPGA i uszkodzić go. Poniższy przykład pokazuje, gdy układ zasilany napięciem 3.3 V otrzymuje sygnał odbity na poziomie prawie 6 V.

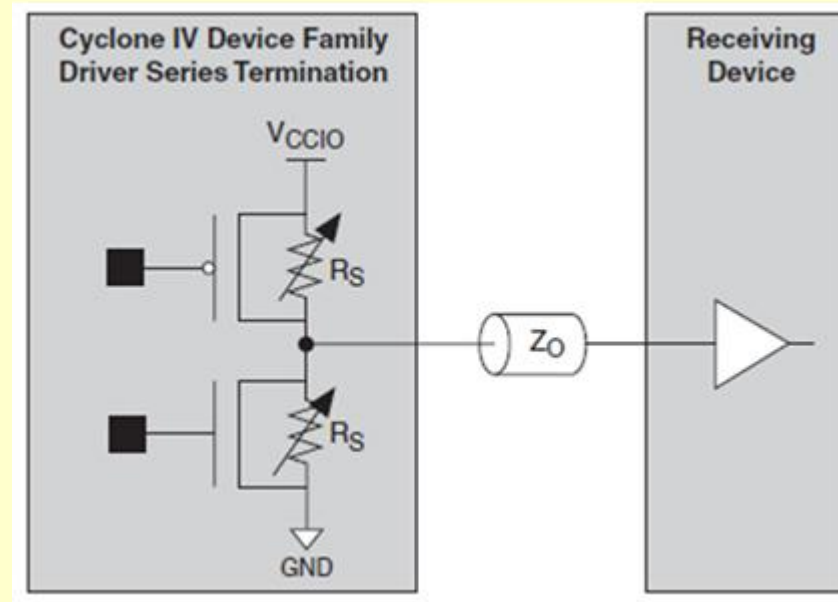
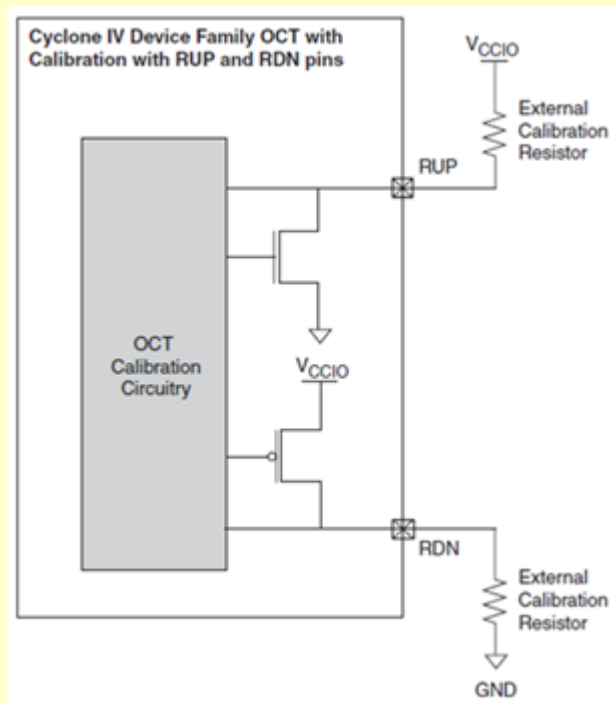


Przyczyną powstawania sygnału odbitego są niedopasowania w torze transmisyjnym – nieprawidłowo zaprojektowana płytko drukowana (PCB). Systemy CAD zapewniają mechanizmy kompensujące zjawisko.

W FPGA moduły I/O posiadają w ograniczonym zakresie możliwość dopasowania wejścia oraz wyjścia do impedancji linii transmisyjnej. Pozwala to redukować odbicia i zachować spójność sygnału, bez stosowania elementów zewnętrznych. Mechanizm dopasowania dostępny jest dla pinów wyjściowych i dwukierunkowych.

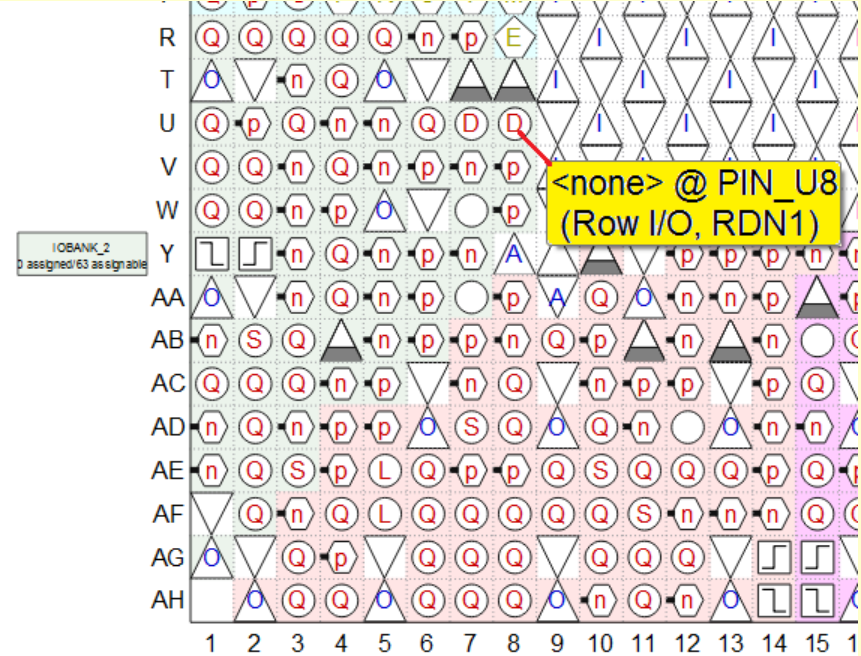
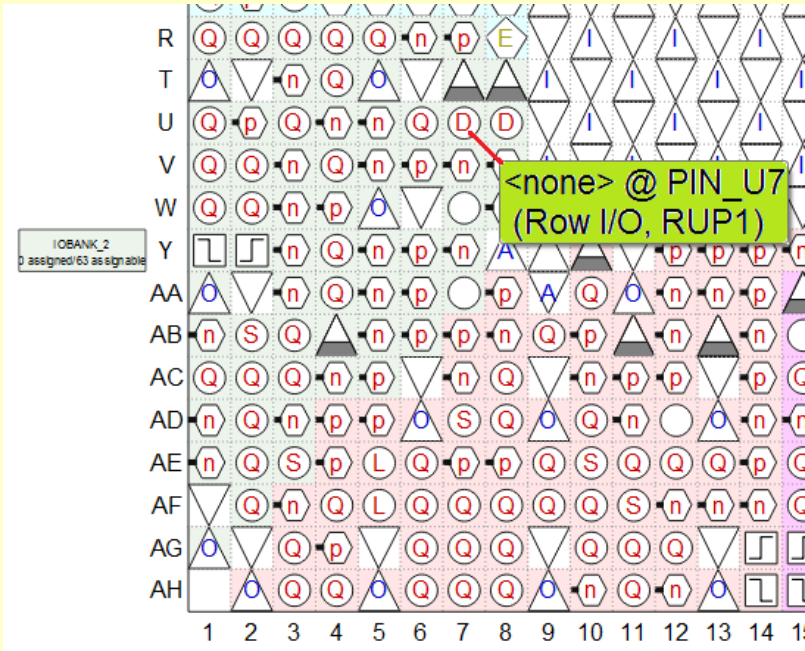
Zastosowanie tego mechanizmu, wyklucza opcję ograniczania prądu wyjściowego. Możliwe są dwie wersje konfiguracyjne OCT: OCT z kalibracją, OCT bez kalibracji.

Kalibracyjne OCT polega na szeregowym dopasowaniu w banku wyprowadzeń. Układ kalibrujący porównuje impedancję wyjściową I/O z zewnętrznymi 1% rezystorami odniesienia, najczęściej 25/50 omów, dołączonymi do wyprowadzeń RUP i RDN. Reguluje on impedancję wyjściową do równoległego połączenia RUP i RDN.

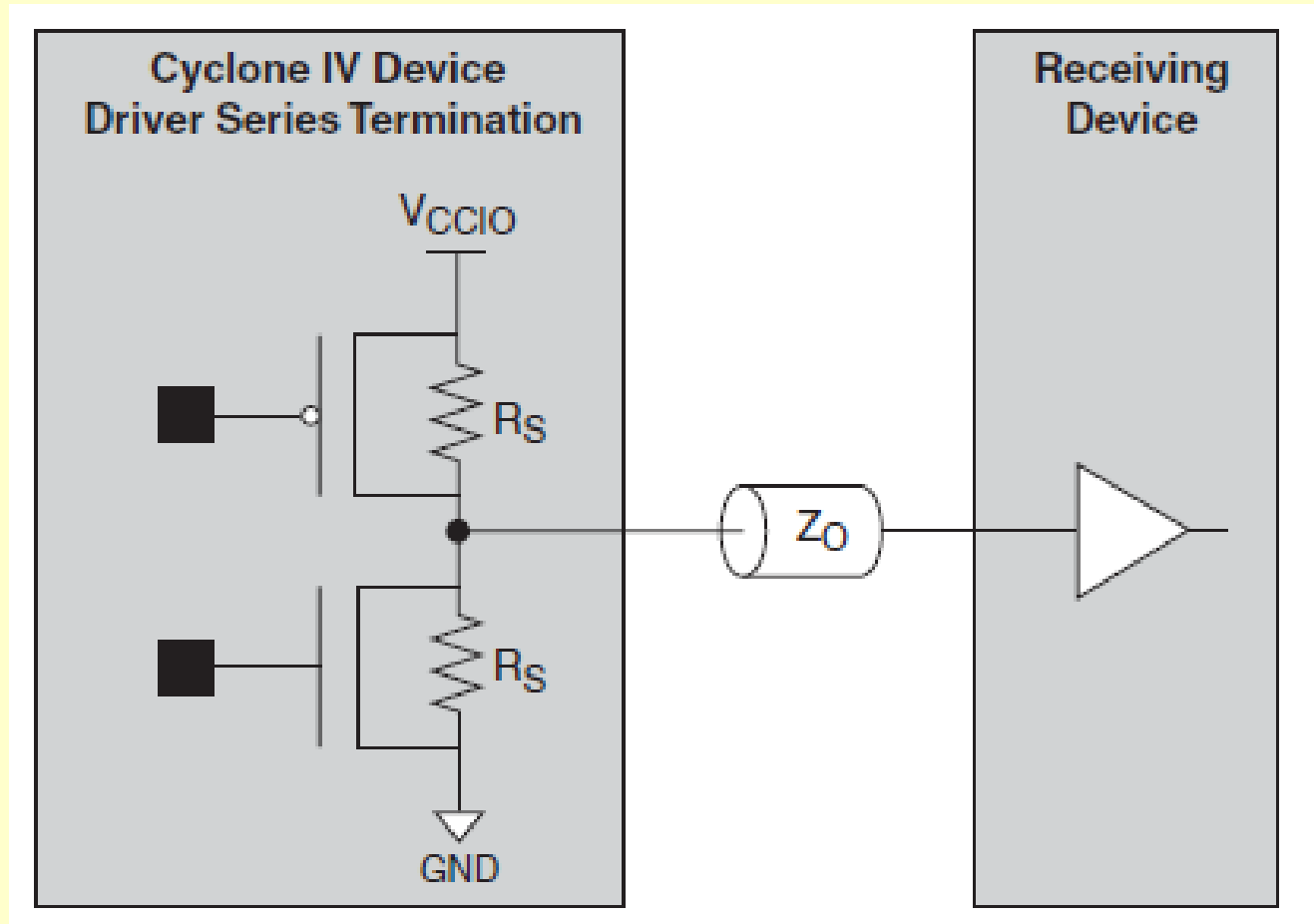


Piny RUP i RDN w FPGA są podwójnego znaczenia, nie są one bezpośrednio widoczne w Pin Planner. W związku z tym warto skorzystać z dokumentacji technicznej producenta w celu ich identyfikacji:

Pin Information for the Cyclone® IV EP4CE115 Device										
Version 1.1										
Notes (1), (2), (3)										
Bank Number	VREFB Group	Pin Name / Function	Optional Function(s)	Configuration Function	F780	F484	DQS for X8/X9 in 780 FBGA	DQS for X8/X9 in 484 FBGA	DQS for X16/X18 in 484 FBGA	DQS for X32/X36 in 484 FBGA
B2	VREFB2N0	IO	DIFFIO_L28n		R4	M3	DQ1L	DQ1L	DQ3L	DQ1L
B2	VREFB2N0	IO	DIFFIO_L29p		T4	N2	DQ1L	DQ1L	DQ3L	DQ1L
B2	VREFB2N1	IO	DIFFIO_L39n		AD1	Y1		DQ3L	DQ3L	DQ1L
B2	VREFB2N1	IO			AB3		DQ3L			
B2	VREFB2N1	IO	VREFB2N1		T8	T3				
B2	VREFB2N1	IO	DIFFIO_L40p		AA4		DQ3L			
B2	VREFB2N1	IO	DIFFIO_L40n		AA3	AA1		DQ3L	DQ3L	DQ1L
B2	VREFB2N1	IO	RUP1		U7	V4				
B2	VREFB2N1	IO	RDN1		U8	V3				
B2	VREFB2N1	IO	DIFFIO_L41p		AE2		DQ3L			
B2	VREFB2N1	IO	DIFFIO_L41n		AE1					



W OCT bez kalibracji, programowo wybierana jest wartość impedancji wyjściowej I/O spośród wartości 25 i 50 omów. Dopasowanie impedancji można przeprowadzać dla wszystkich pinów I/O. Ten rodzaj dopasowania jest podatny na wpływy temperatury i zasilania, gdyż wykorzystuje półprzewodnikowe struktury wewnętrzne. Poniższy rysunek przedstawia OCT bez kalibracji w przypadku standardu niesymetrycznego.



Wyboru konkretnego typu dopasowania OCT dedykowanego do określonych standardów sygnałowych wraz z podaniem jego parametrów dokonujemy w Assignment Editor. Możliwych kombinacji ustawień jest wiele, poniżej przedstawiono jeden szczegółowy przypadek.

Quartus Prime Lite Edition - E:/Tmp/test_1 - test_1

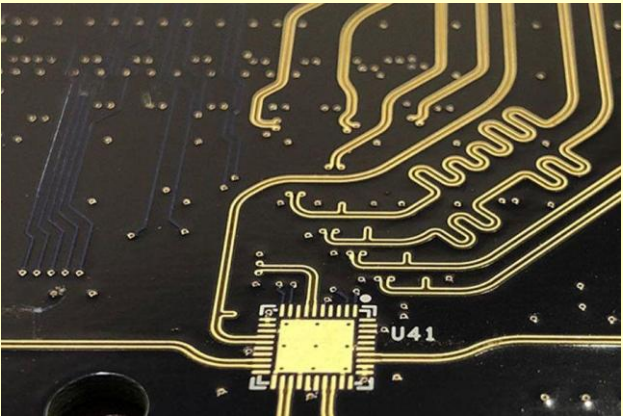
File Edit View Project Assignments Processing Tools Window Help

test_1

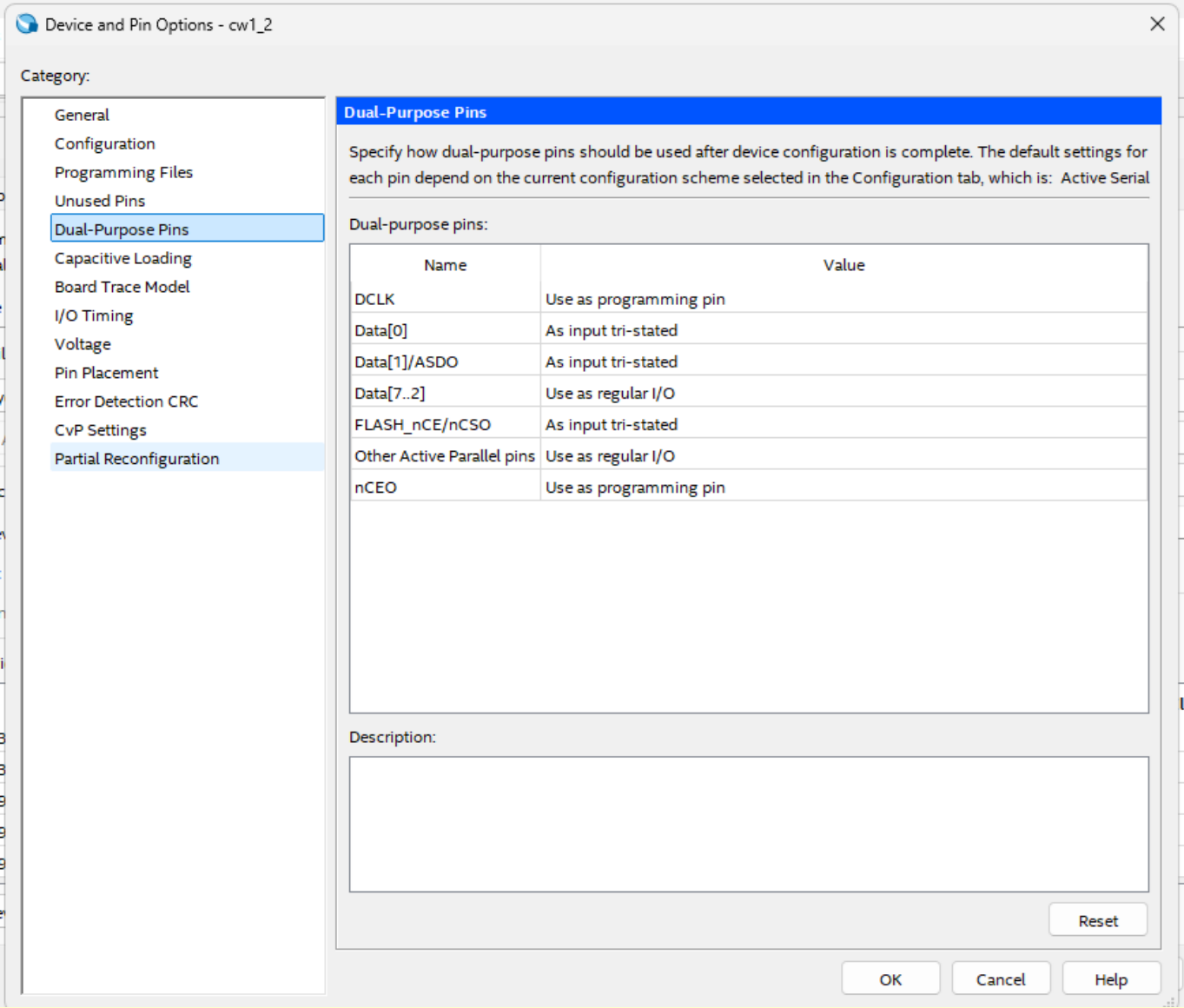
test_1.bdf Assignment Editor

<<new>> Filter on node names: *

	Status	From	To	Assignment Name	Value	Enabled	Entity
1	✓ Ok		in In	Location	PIN_A3	Yes	
2	✓ Ok		out Out_1	Location	PIN_A4	Yes	
3	✓ Ok		out Out_2	Location	PIN_A7	Yes	
4	✓ Ok		out Out_3	Location	PIN_A8	Yes	
5	✓ Ok		out Out_4	Location	PIN_A11	Yes	
6	✓ Ok		out Out_1	Output Termination	Parallel 50 Ohm with Calibration	Yes	test_1
7		<<new>>	<<new>>	<<new>>			



Ogólne warunki pracy kompilatora znajdują się w: Assingments → Device → Device and Pin Options →



Zaliczenie częściowe wykładu nr 9 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

V *Very High Speed Integrated Circuit*
H *Hardware*
D *Description*
L *Language*

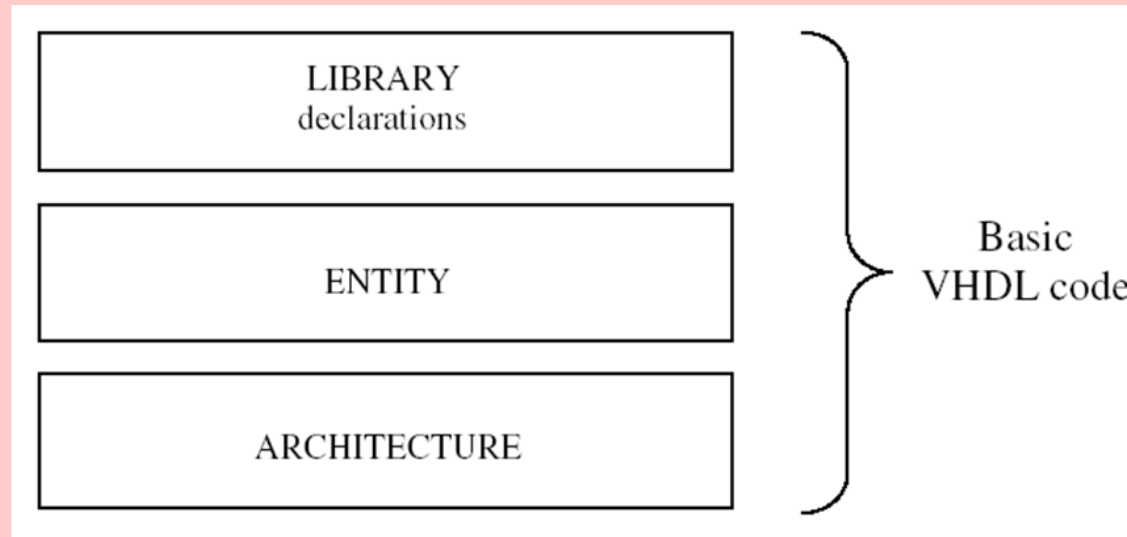
Opis w VHDL

Symulacja w VHDL

**Synteżowalność
w VHDL**

VHDL umożliwia opis układów:

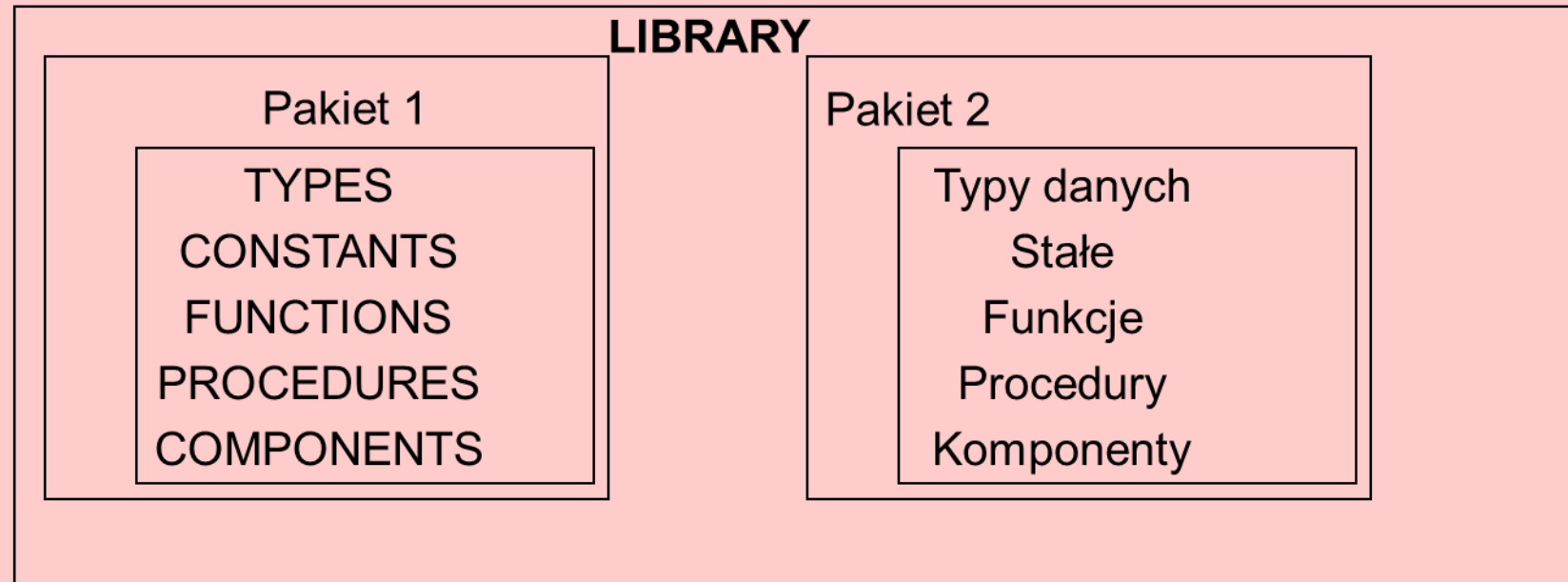
- przetwarzających równoległe,
- przetwarzających sekwencyjne,
- jest on oderwany od technologii,
- zapewnia hierarchiczność projektów,
- otwarty na: LabView, MathCAD,
- język zaakceptowany w środowisku,
- język opisu dużych układów,
- alternatywą jest język Verilog.



- LIBRARY: zawiera listę bibliotek używanych w projekcie. Np.: ieee, std, work, itd.
- ENTITY: definiuje porty I/O w układzie (połączenia z otoczeniem),
- ARCHITECTURE: zawiera właściwy opis działania układu.

Czasami można pominąć blok LIBRARY. Z jednym ENTITY może być związane kilka architektur, w niektórych przypadkach dodawane są inne dodatkowe bloki, np. konfiguracyjne.

Biblioteki – zbiory wspólnych i powtarzających się elementów kodu, zawierają obiekty wielokrotnego użycia, ich stosowanie skraca długość kodu i upraszcza go, biblioteki rozszerzają dostępne w języku narzędzia. W VHDL biblioteki pogrupowane są w pakietach/plikach – nazwa biblioteki wskazuje folder/katalog z pakietami.

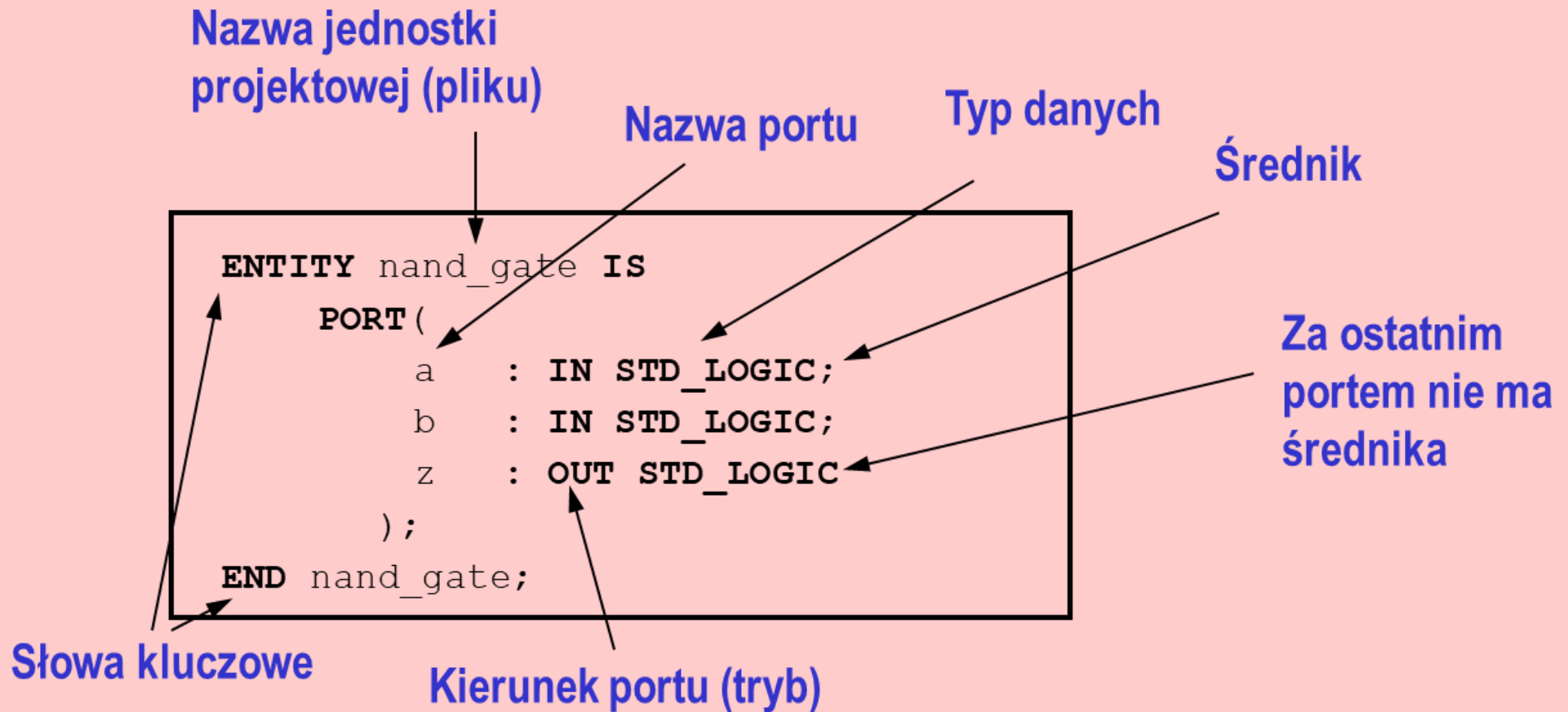


Biblioteki należy deklarować na początku kodu używając 2 poniższych instrukcji:

```
library library_name;  
use library_name.package_name.all;
```

z których pierwsza podaje nazwę folderu biblioteki, druga nazwę konkretnego pakietu oraz zakres – najczęściej stosujemy **.all** udostępniając wszystkie elementy

Podstawowa struktura ENTITY:



ENTITY zawiera wykaz portów, czyli połączeń projektowanego układu z otoczeniem – realizuje to wewnętrzna instrukcja PORT. Czasami w Entity występuje deklaracje stałych globalnych – instrukcja GENERIC.

- **porty** – to kanały przepływu danych pomiędzy jednostką projektową a otoczeniem;
- w dużych projektach występują porty wewnętrzne – łączące bloki/moduły;
- port można deklarować oddzielnie lub dla tego samego typu stosować listę:
a, b : IN STD_LOGIC;
- portom wejściowym (typu IN) można nadawać wartości początkowe, które ustawią się po włączeniu zasilania układu jeszcze przy braku sygnału – np. 0 lub 1;
- port jest **sygnałem** – port wejściowy może łączyć z innym portem wyjściowym;
- **port wejściowy może być argumentem wyrażenia logicznego** – po jego prawej stronie;
- każdy port musi być zadeklarowany – deklaracja portu **musi** zawierać:
 - **nazwę** (np.: A, b, ABC, J23, MaRuDa_34, ...),
 - **tryb** (kierunek: in, out, itp.),
 - **typ danych** przesyłanych (np.: bit, std_logic, integer, itp.).

W VHDL:

- wielkość liter nie ma znaczenia,
- zabronione jest używanie jako nazw słów **kluczowych**,
- nazwę stanowi litera lub litera plus kombinacja liter i/lub cyfr i podkreśleń w środku nazwy;
- instrukcja w VHDL zaczyna się słowem kluczowym/nazwą i kończy średnikiem.

Tryb portu określa kierunek przepływu danych przez port. W przypadku braku specyfikacji domyślnie przyjmowany jest tryb **In**.

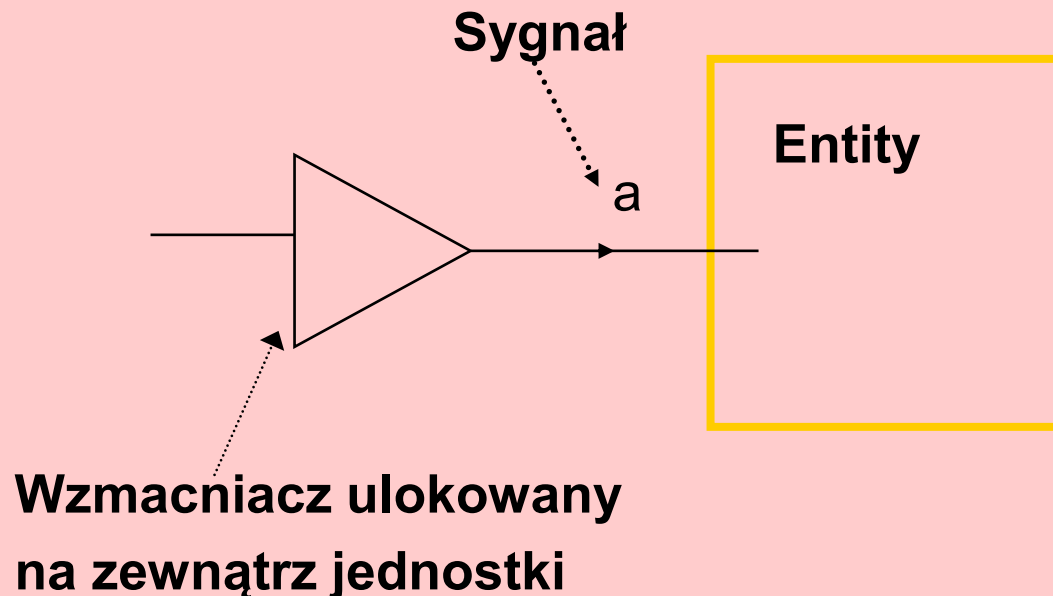
In – oznacza jednokierunkowy port wejściowy.

Łączy on wpływające do układu sygnały zegarowe, sterujące i dane.

Dane przechodzące przez ten port są czytane z zewnątrz jednostki projektowej.

Mogą one pojawiać się tylko po prawej stronie łączonych sygnałów/zmiennych, np.:

$Y \leq a;$



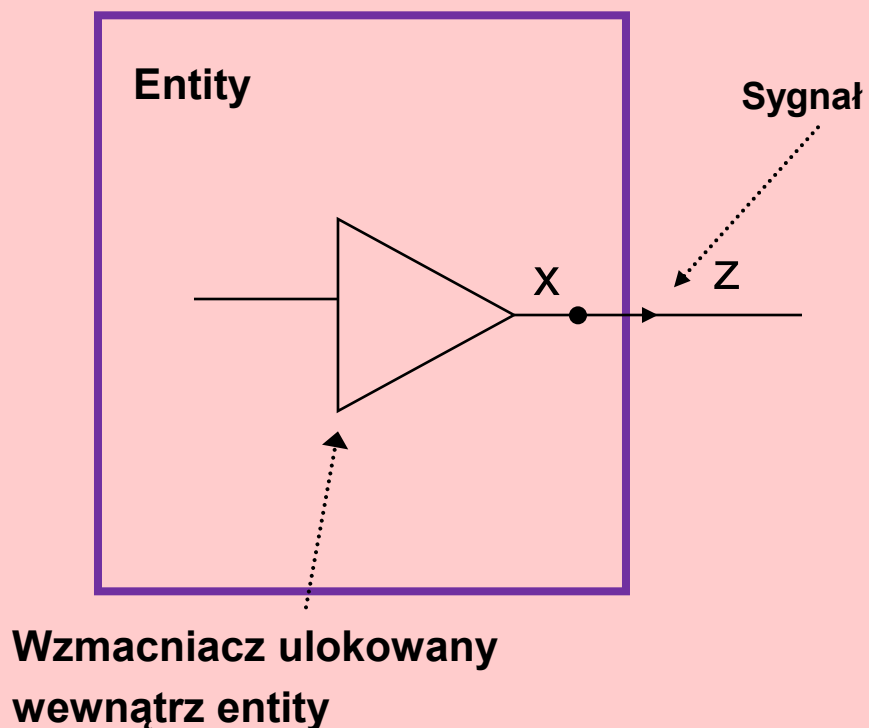
Out – jednokierunkowy port wyjściowy.

Jest on związany z sygnałami wypływającymi.

Nie realizuje sprzężeń zwrotnych – nie może być użyty jako argument wewnątrz.

Wartość danych wyjściowych może być aktualizowana tylko wewnątrz jednostki projektowej. Mogą one pojawiać się tylko po lewej stronie łączonych sygnałów, np.:

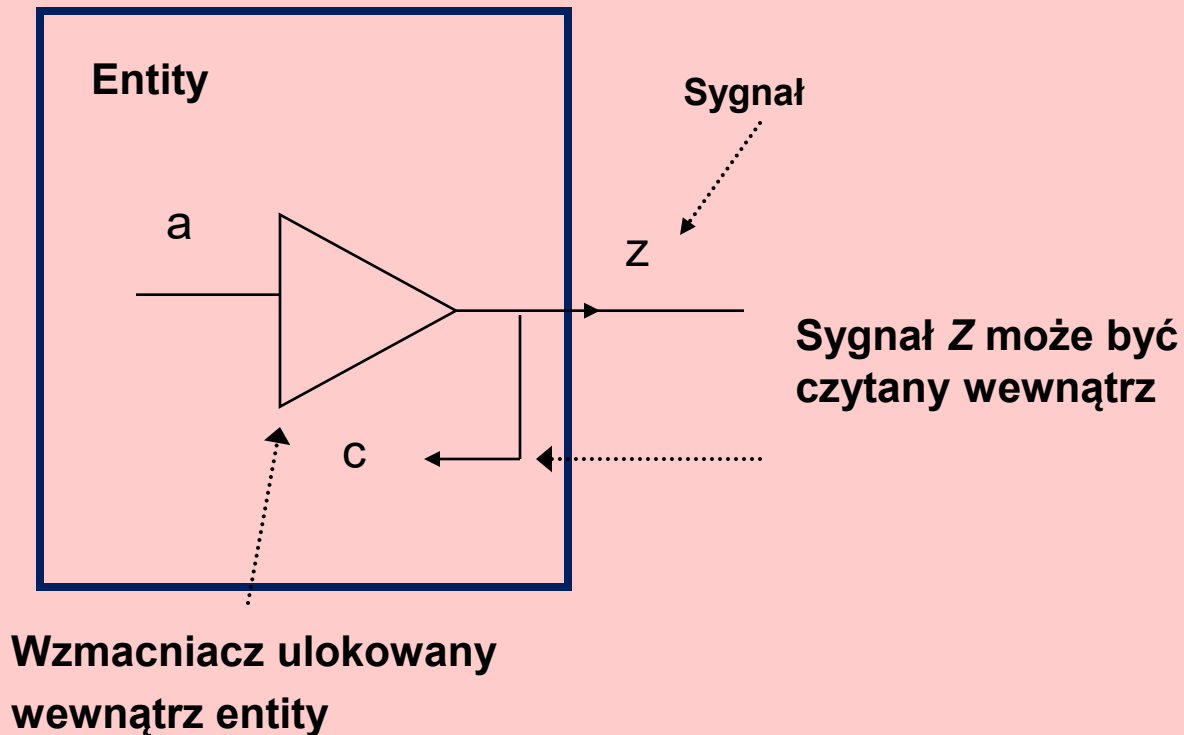
$z \leq x;$



Buffer – port wyjściowy z realizacją wewnętrznych sprzężeń zwrotnych, np. czytanie wyjść licznika w jednostce projektowej, nie jest portem dwukierunkowym.

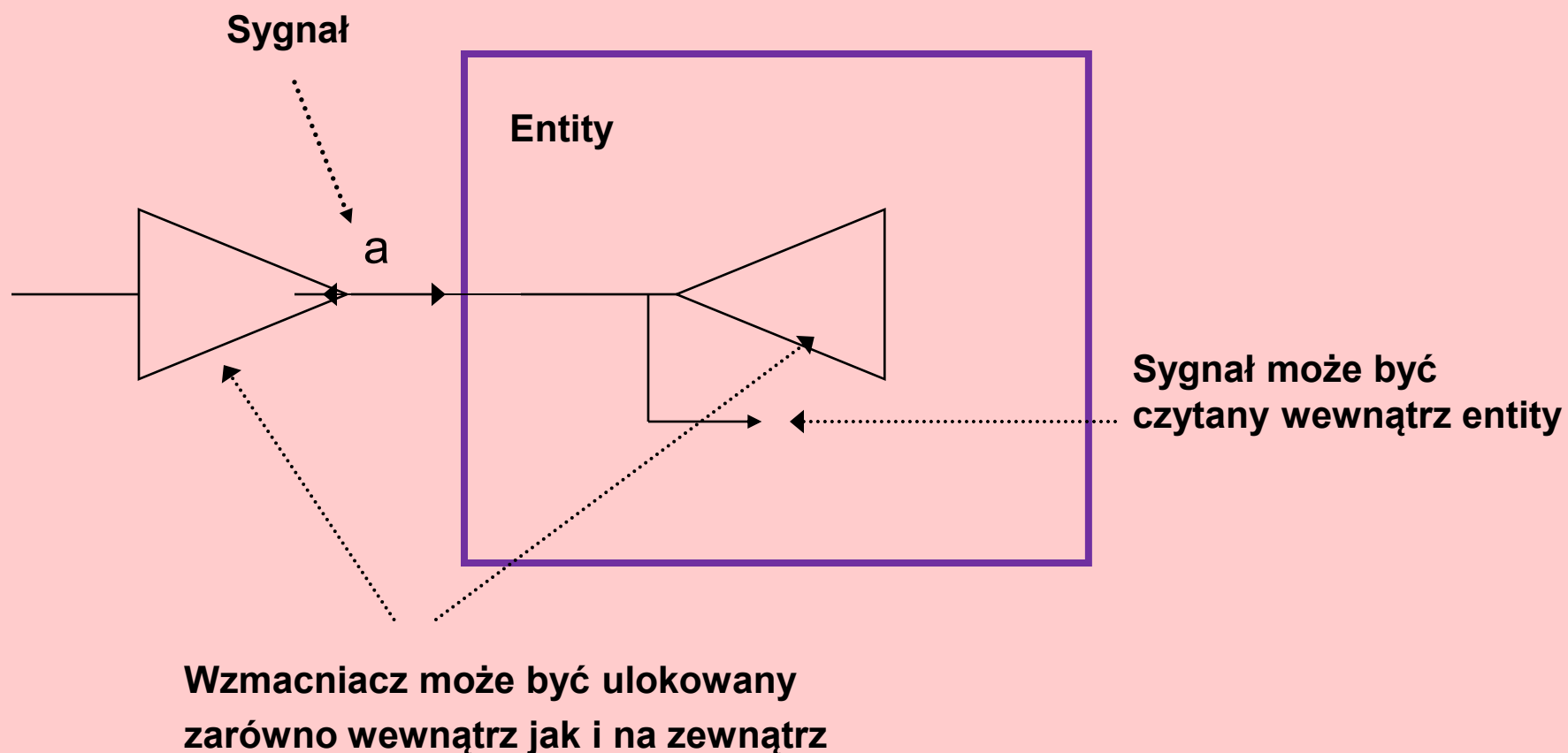
Wartość danych wyjściowych może być aktualizowana tylko wewnątrz jednostki projektowej. Ten port stosowany jest do sygnałów wychodzących z jednostki.

$z \leq a;$ $c \leq z;$

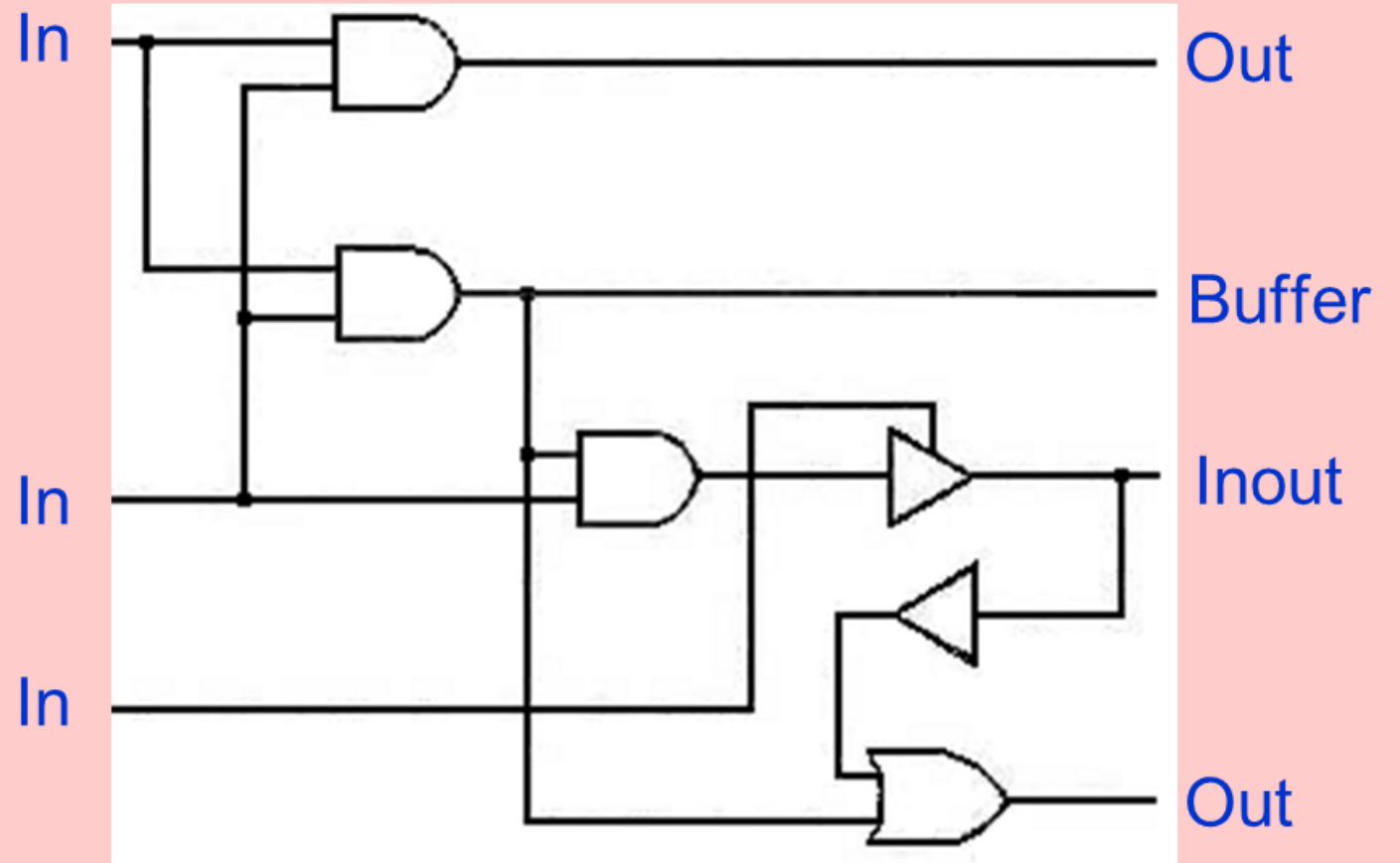
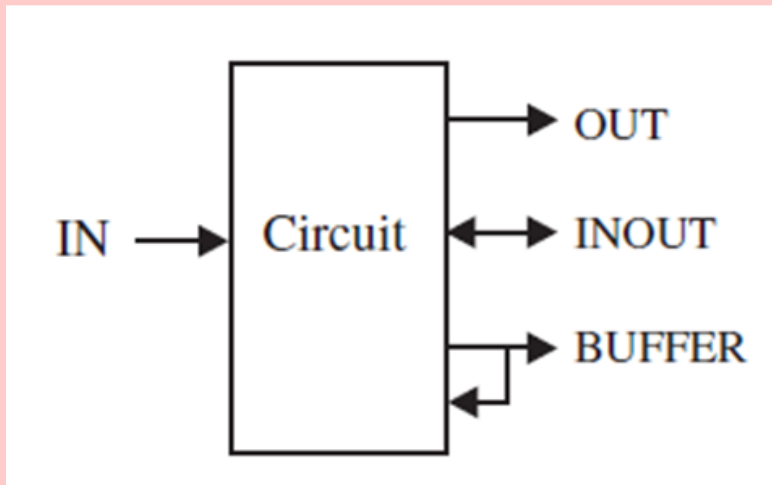


InOut – port dwukierunkowy, ze sprzężeniem zwrotnym, może zastępować każdy z poprzednio wymienionych portów.

Wartość danych wyjściowych może być czytana i aktualizowana wewnątrz jednostki projektowej. Mogą one pojawiać się po obu stronach łączonych sygnałów.



Podsumowanie/analogie:



Architektura (architecture body) - opisuje funkcjonowanie układu, w jaki sposób pracuje układ:

- jej nazwa własna **musi/powinna** odróżniać ją od nazwy interfejsu z którym jest związana,
- opisy interfejsu i architektury/architektur powinny znajdować się w jednym pliku,
- z tym samym interfejsem może być związane więcej niż jedna architektura w pliku VHDL.

architecture AB of CD is

- deklaracja sygnałów;
- deklaracja ...;

begin

Współbieżne instrukcje:

-,
-,
-,
-,
- **Instrukcja procesu**
- deklaracja zmiennych,

begin

Sekwencyjne
instrukcje:

.....

.....

end process;

end architecture;

Najważniejsze instrukcje współbieżne w VHDL:

- proste przypisanie sygnału -- **A <= B;**
- selektywne przypisanie -- **with select ...**
- warunkowe przypisanie -- **when ... else**
- instrukcja powielania -- **GENERATE,**
- konkretyzacja komponentów, -- **PORT MAP**
- instrukcja procesu -- **PROCESS ...**
- wywołanie procedury -- **PROCEDURE**

Najważniejsze instrukcje sekwencyjne:

- instrukcja oczekiwania -- **wait on/for/until ...**
- instrukcja warunkowa -- **if – then – else ...**
- instrukcja wyboru -- **case – when ...**
- instrukcja pętli -- **loop for/while ...**
- pomocnicze instrukcje pętli -- **exit / next ...**
- wywołanie funkcji/procedury -- **funkction.**

architecture NAZWA_ARCITEKTURY **of** NAZWA_ENTITY **is**

- tu można deklarować sygnały, stałe, funkcje, procedury
- można też deklarować komponenty
- **Nie wolno w tym miejscu deklarować zmiennych !!**

begin

Instrukcje **współbieżne**:

- przypisania do sygnału,
- instrukcja powielania,
- ...
- konkretyzacji komponentu - podstawienie,
- instrukcja procesu
 - deklaracja zmiennych - **nie wolno deklarować sygnałów**,
 - **begin**
 - w tym miejscu tylko **instrukcje** wykonywane **sekwencyjnie**,
 - end ...**

end NAZWA_ARCHITECTURE; - - lub **end architecture**;

Przykład projektu z brakiem bibliotek zewnętrznych – tylko STD

```
ENTITY example IS
```

```
    PORT ( a, b, clk: IN BIT;
```

```
           q: OUT BIT);
```

```
END example;
```

```
ARCHITECTURE example1 OF example IS
```

```
    SIGNAL temp : BIT;
```

```
    BEGIN
```

```
        temp <= a NAND b;
```

```
        PROCESS (clk)
```

```
        BEGIN
```

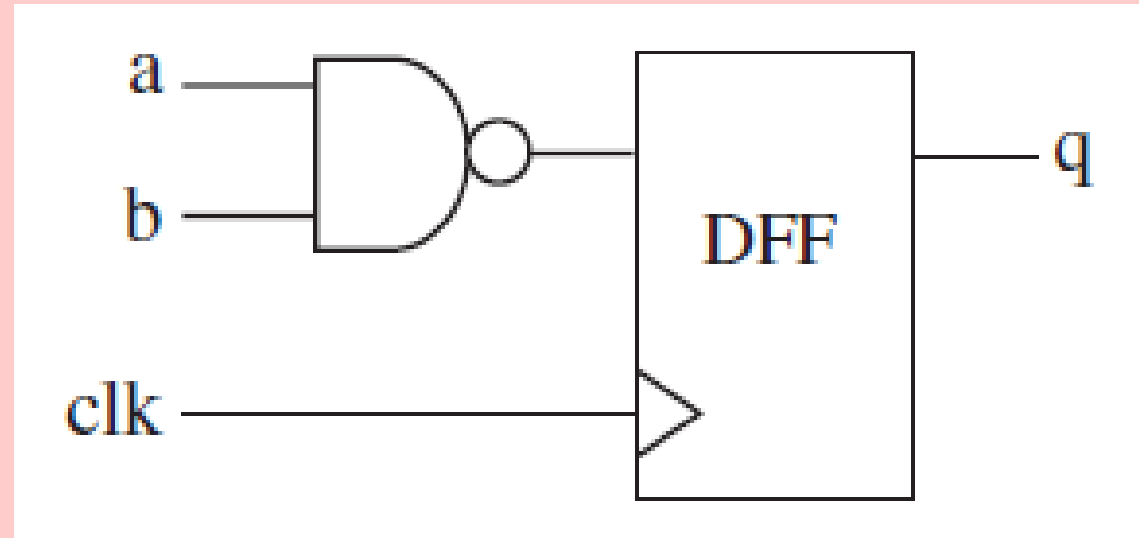
```
            IF (clk'EVENT AND clk='1') THEN
```

```
                q <= temp;
```

```
            END IF;
```

```
        END PROCESS;
```

```
    END example1;
```



Elementy leksykalne VHDL:

Do opisu i pamiętania przetwarzanych danych w VHDL służą obiekty, które używamy w kodzie – są nimi:

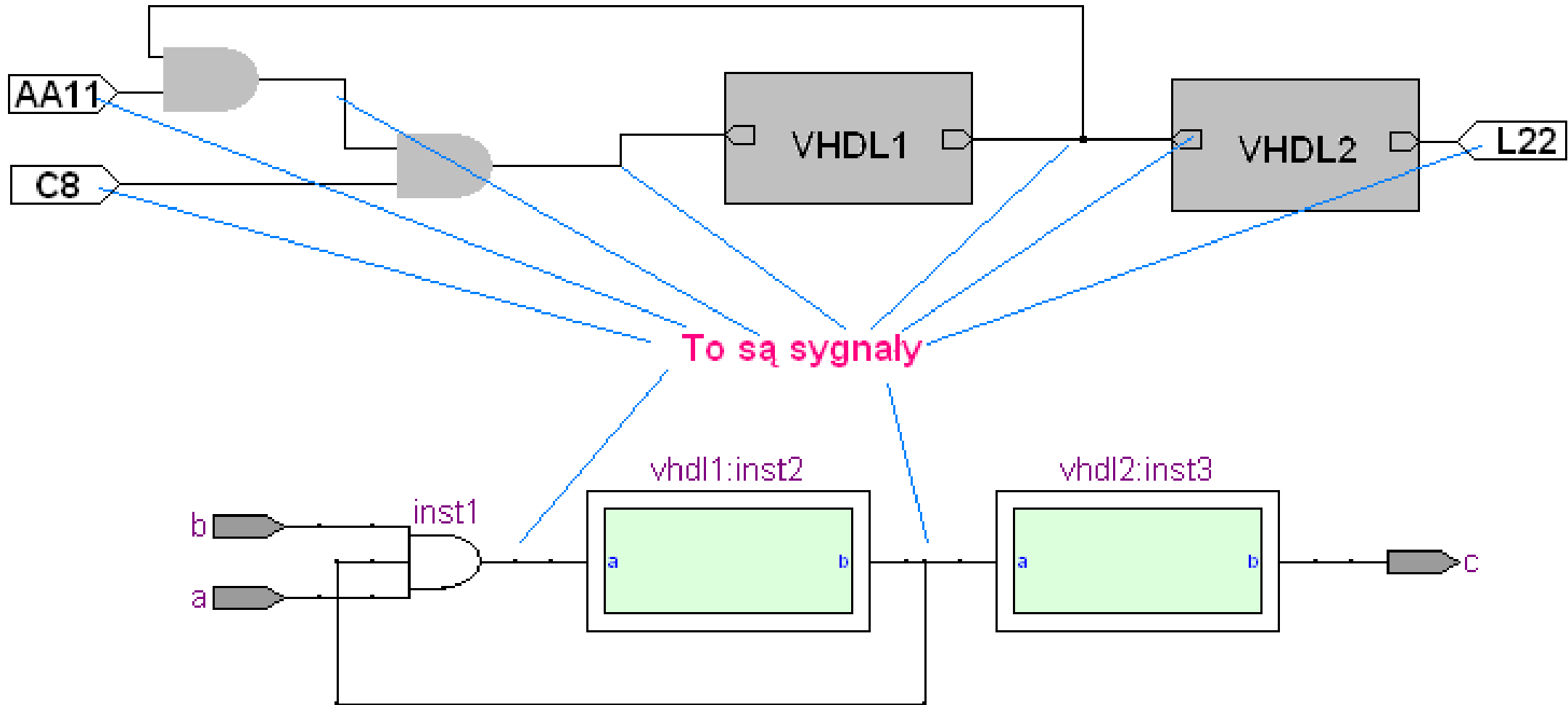
- sygnały,
- zmienne,
- stałe.

Sygnały reprezentują „przewody połączeniowe” między elementami.

Rozróżniamy sygnały wewnętrzne oraz porty – sygnałami są również porty:

- sygnały są współbieżnymi elementami VHDL – są to wartości napięć w FPGA,
- sygnały muszą być deklarowane wewnątrz architektury – wyjątek stanowią porty deklarowane w ENTITY,
- wartości sygnałów powstają z danych wpływających z portów lub wytworzonych wewnątrz struktury,
- przez porty zewnętrzne przepływają tylko wartości logiczne: bit, std_logic i unsigned – czyli poziomy/napięcia logiczne 0 i 1,
- przez porty wewnętrzne mogą przepływać sygnały dowolnego typu – np. Real.

Sygnałami są:



Wartości sygnałów powstają w wyniku:

- wprowadzenia poziomów logicznych do portów przez urządzenia zewnętrzne,
- wykonania instrukcji **przypisania/podstawienia do/- sygnału** dla sygnałów wewnętrznych,
- do sygnału można przypisać tylko wartość tego samego typu i rozmiaru.

Składnia/Syntax instrukcji prostego przypisania sygnału jest następująca:

signal-assignment-statement →
[label ':'] (*signal-name*) '<=' *value-expression*

Przykłady:

Y <= '1';

Z <= A;

S <= not (A and B);

skakaly_zajace <= niech_nam_zyje_2000;

Żeby sygnał wewnętrzny mógł zostać użyty – musi być zadeklarowany/nazwany/zdefiniowany.

Definiujemy je w części deklaracyjnej architektury – pomiędzy wierszami **architecture** a **begin**:

architecture AB of CD is

- tylko w tym miejscu deklarujemy sygnały wewnętrzne

begin

Instrukcje **współbieżne**:

- przypisania do sygnału,
- ...

Składnia instrukcji deklaracji sygnału jest następująca:

signal-declaration →

signal identifier { ‘,’ identifier } ‘:’ subtype-indication [‘:=’ expression] ‘;’

lub:

signal **nazwa_sygnału** : **typ**[ograniczenie_zakresu] [:= wyrażenie];

Przykłady deklaracji sygnałów:

signal zegar : bit ;	-- domyślna wartość '0';
signal reset : bit := '1';	-- inicjacja wartością '1';
signal x : integer ;	-- domyślnie -2.147.483.647
signal y : bit_vector (0 to 3) range 0 to 10;	-- ograniczenie zakresu
signal m13 : bit_vector (7 downto 0) := (others => '1'); -- ustawienie wszystkich bitów na 1	

Ograniczenie zakresu oznacza zawężenie przedziału przyjmowanych wartości – realizuje klauzula *range*.

Brak określenia wartości początkowej – **oznacza automatyczne ustawienie przez kompilator domyślnie najmniejszej/lewej skrajnej wartości dla danego typu.**

Np. jeżeli nie określimy zakresu dla sygnałów typu integer – kompilator przypisze wartość początkową sygnału równą -2 147 483 647, w przypadku std_logic będzie to '0'.

W przypadku wektorów najczęściej stosuje się konwencję "górny indeks **downto** dolny indeks", np.: 7 downto 0. Można stosować notację odrotną w układzie "dolny indeks **to** górny indeks", np.: 0 to 7.

Dolnym indeksem nie musi być wartość 0 – np.: **signal** tmp : std_logic_vector(7 **downto** 4);

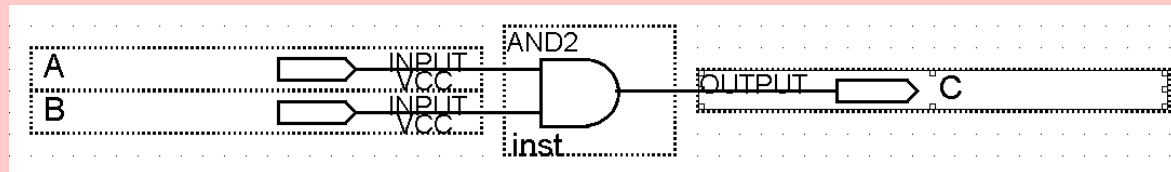
Instrukcja prostego przypisania sygnału:

architecture Opis1 of Plik2 is

- tu można deklarować sygnały, stałe, funkcje, procedury

beginInstrukcje **współbieżne**:

- przypisania do sygnału,
- instrukcja powielania,
- ...

end architecture;**LIBRARY** IEEE;

use ieee.std_logic_1164.all;

ENTITY example IS**PORT** (a, b : **IN** std_logic; c: **OUT** std_logic);**END** example;**ARCHITECTURE** example2 **OF** example IS**BEGIN****c** <= a **AND** b;**END** example2;

Rozróżniamy trzy rodzaje podstawień do sygnału:

- proste podstawienie do sygnału (simple signal assignment statement),
- warunkowe podstawienie do sygnału (conditional signal assignment statement),
- selektywne podstawienie do sygnału (selected signal assignment statement).

Wszystkie podstawienia są instrukcjami współbieżnymi.

architecture Opis1 **of** Plik2 **is**

- tu można deklarować sygnały, stałe, funkcje, procedury

begin

Instrukcje **współbieżne**:

- przypisania sygnału,
- instrukcja powielania,
- ...

end architecture;

Instrukcje te wpisujemy w części współbieżnej architektury w dowolnej kolejności.

Po włączeniu zasilania instrukcje współbieżne "startują" w tym samym czasie – jest to przetwarzanie równoległe.

Instrukcja selektywnego przypisania sygnału

with control_signal select

```

    result_signal <= val_a when control_signal_val1, -- first control_signal value
                    val_b when control_signal_val2, -- second control_signal value
                    val_c when control_signal_val3,
                    ...                               -- other control_signal values or
                    val_x when others;                -- closing clause

```

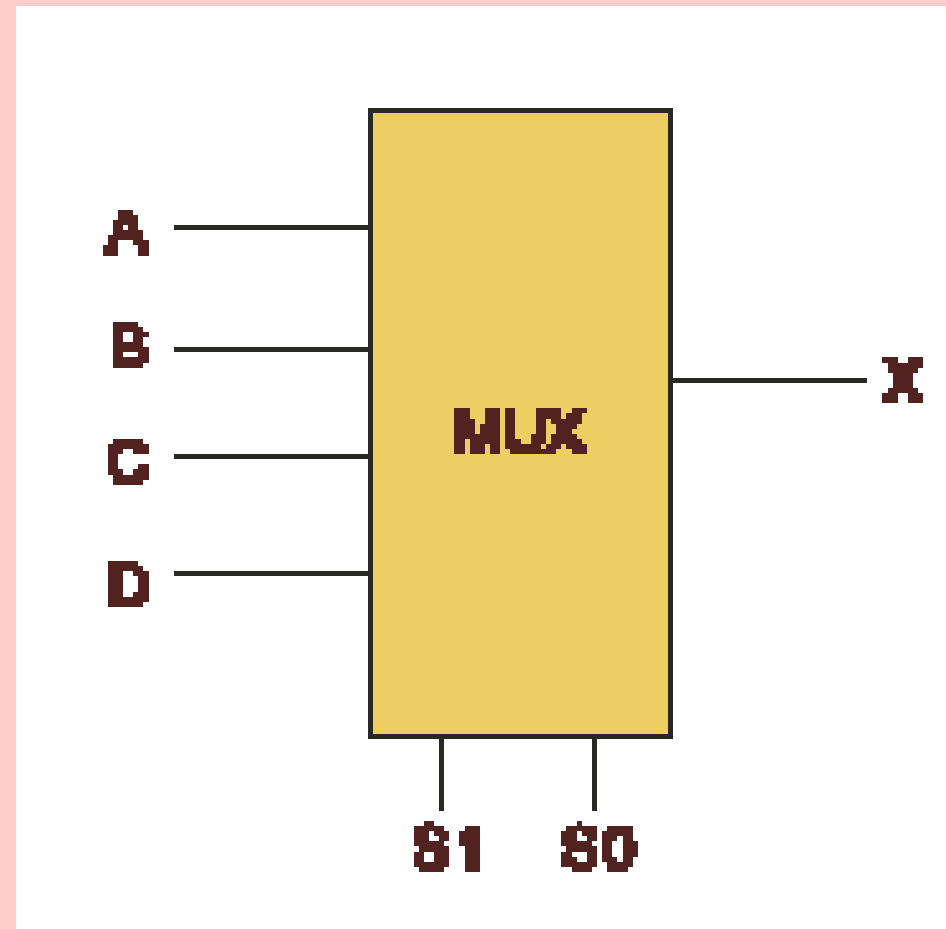
Instrukcja zawiera:

- sygnał sterujący jej wykonaniem **control_signal** często jest to sygnał wejściowy;
- **control_signal** może być bitem lub wektorem – nie może być kombinacją różnych sygnałów;
- instrukcja opisuje również tylko jedno wyjście **result_signal**, bitowe lub wektorowe;
- wyjściowe wartości (**val_a**, **val_b**, itd.) są stałymi i/lub sygnałami, które są określone;
- konkretna wartość wyjścia wynika z wartości (**control_signal_val1**, ...) sygnału sterującego;
- wartości (**control_signal_val1**, **control_signal_val2**, ...) mogą być złożonymi wyrażeniami;
- instrukcja **musi** określać sygnał wyjściowy dla wszystkich wartości sygnału sterującego;
- jeżeli występują powtarzające się wartości sygnału wyjściowego – na końcu instrukcji można użyć klauzuli **when others** ustalającej wartość sygnału dla tych przypadków.

Przykład:

with s select

```
x <= a when "00",  
      b when "01",  
      c when "10",  
      d when others;
```



Współbieżna instrukcja warunkowego przypisania do sygnału **WHEN-ELSE**

Syntax:

```
signal_name <= value_a when condition1 else  
    value_b when condition2 else  
    value_c when condition3 else ...  
    value_x;
```

Jeszcze inny sposób definiowania multipleksera, tym razem z użyciem instrukcji warunkowego przypisania do sygnału.

```
x <= a when s = "00" else  
    b when s = "01" else  
    c when s = "10" else  
    d ;
```

Typy danych w VHDL

- **BIT** (oraz **BIT_VECTOR**) – akceptuje wartości ‘0’ i ‘1’ i ich łańcuchy.

Przykłady deklaracji tego typu:

SIGNAL x: BIT;

VARIABLE y: BIT_VECTOR (3 DOWNT0 0);

Przykłady przypisania wartości:

X <= '0';

X <= '1';

Y := "0101";

----- które wyrażenie jest poprawne ? -----

x := Y(1);

X <= y(0);

y := (x,x,x,x);

Y <= x & x & x & '1';

Typ **STD_LOGIC** (oraz **STD_LOGIC_VECTOR**) – jest podtypem **STD_ULOGIC**,
nie obsługuje on wartości niezainicjowanej ‘U’,

- **STD_LOGIC** (oraz **STD_LOGIC_VECTOR**) – przyjmuje syntezywalne wartości: '0', '1', 'Z'

użycie tego typu danych wymaga dołączenia pakietu **ieee.std_logic_1164.all**

Przykłady przypisania wartości:

A <= '1';

B <= '0';

C := 'Z';

D := "1101";

E <= "1Z00";

F := "110ZZ";

- **INTEGER** – 32-bitowe wartości całkowite z przedziału: $-2.147.483.647 \div +2.147.483.647$. Ten typ danych umożliwia całkowite operacje matematyczne na danych.

Przykłady deklaracji sygnałów i zmiennych tego typu:

```
port ( A : in integer;  B : out integer);
```

```
signal D : integer;
```

```
variable E : integer;
```

oraz podstawiania wartości:

```
B <= A * 100;
```

```
D <= E / 2;
```

```
E := A + (D - 20);
```

- **SIGNED/UNSIGNED** – cyfrowy zapis danych typu integer. Ten typ danych jest synteżowany w portach zewnętrznych pliku top-level.

Ten typ danych wymaga dołączonej biblioteki:

```
use ieee.numeric_std.all;
```

Przykłady deklaracji typu i stosowania:

```
SIGNAL y: UNSIGNED (0 TO 3);
```

```
SIGNAL x: SIGNED (7 DOWNT0 0);
```

```
y <= "1011" -- integer value is 11,
```

```
x <= "1101" -- integer value is -3    -- wartość ujemna w systemie U2
```

Ten typ danych w pliku top-level może zastępować typy BIT oraz STD_LOGIC i ich pochodne. Można z jego użyciem wykonywać takie same operacje matematyczne jak na typie integer.

Często deklarowanym typem danych w układach sekwencyjnych jest typ wyliczeniowy.

Typy wyliczeniowe – służą do nazywania stanów.

Zbiór danych takiego typu stanowi listę nazw lub znaków podanych/wyliczonych w nawiasie.

Przykłady:

type **trzej_politycy** *is* (*Holownia*, *Kaczynski*, *Tusk*);

type **trojkowy** *is* (3, 6, 9, 12, 15);

żeby można było skorzystać z danych tak zdefiniowanego typu należy zadeklarować obiekt tego typu (sygnał lub zmienną) – czyli użyć instrukcji:

signal *Politycy* : **trzej_politycy**;

W jaki sposób kompilator rozpoznaje typ danych?

integer: +1, 862, -257, +123_456, 2#101011# (binarnie), 16#00FF# (heksadecymalnie),

real: 0.0, 0.5, 2.718_28, 12.4E-9, 16#D.1#,

literały wyliczeniowe: bit: '1' '0';

boolean: TRUE false;

literały łańcuchowe bitowe: bit_vector -- np. B"0010_0101" -- binarnie 00100101,

std_logic_vector -- np. "101Z0"

Co zrobić jeżeli trzeba przejść z jednego typu danych do innego?

Należy użyć bibliotecznych funkcji konwersji typów - np:

IEEE.std_logic_1164.all: std_logic_vector to bit_vector:

To_bitvector(<slv_sig>) -- syntax

A <= To_bitvector(B); -- użycie

IEEE.std_logic_unsigned.all: std_logic_vector to integer:

CONV_INTEGER(<slv_sig>)

C := CONV_INTEGER(D);

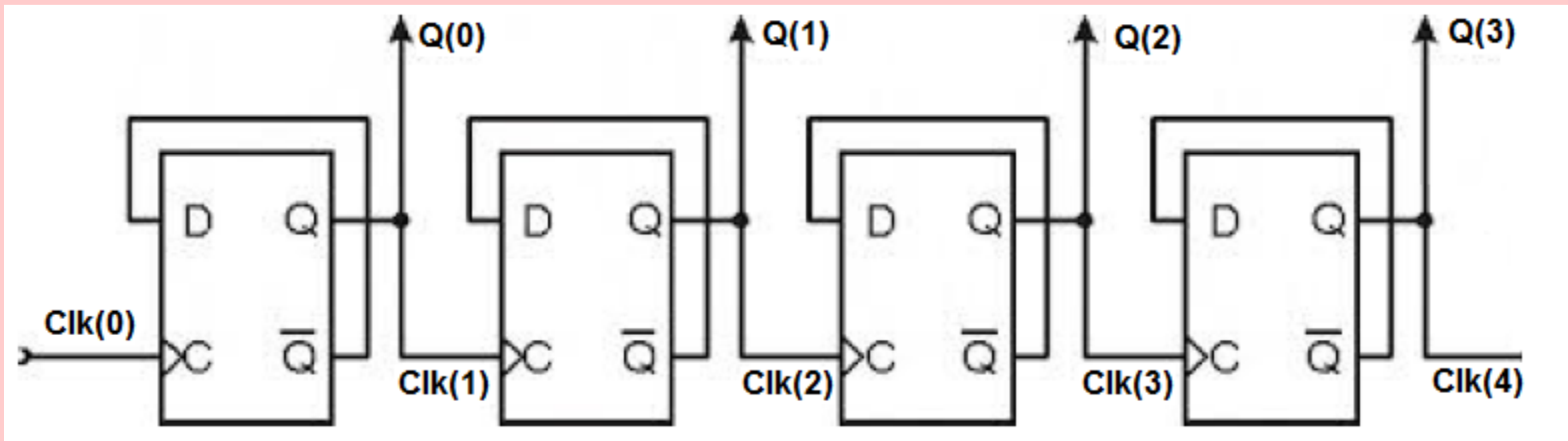
IEEE.std_logic_arith.all: integer to unsigned:

CONV_UNSIGNED(<int_sig>, <integer_size>)

E <= CONV_UNSIGNED(F, 8),

Współbieżna instrukcja powielania strukturalnego **Generate**. Współbieżna instrukcja powielania generate służy do tworzenia regularnie powtarzających się struktur, zawierających te same fragmenty funkcjonalne układu. Jest to instrukcja złożona, wewnątrz której zawarte są inne instrukcje współbieżne.

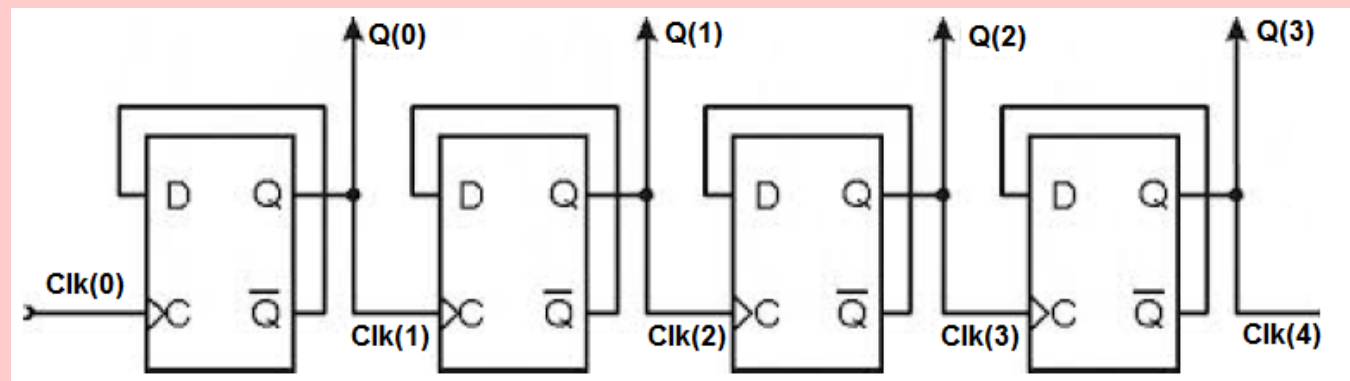
label: *FOR* *identifier* *IN* *range* **GENERATE** -- konstrukcja regularna instrukcji
(instrukcja współbieżna)
END GENERATE;



```

LIBRARY IEEE;
use ieee.std_logic_1164.all;
entity Dzielnik is
    port( CLK_in : in    STD_LOGIC;
          Q      : buffer STD_LOGIC_VECTOR(3
downto 0));
end Dzielnik;
architecture behavior of Dzielnik is
    signal Clk : std_logic_vector(4 downto 0);
begin
    Clk(0) <= CLK_in;
    GG : for i in 0 to 3 generate
        process (Clk)
        begin
            if rising_edge(Clk(i)) then
                Q(i) <= not Q(i);
            end if;
        end process;
        Clk(i+1) <= Q(i);
    end generate;
end behavior;

```



Kodowanie metodą ścieżki danych:

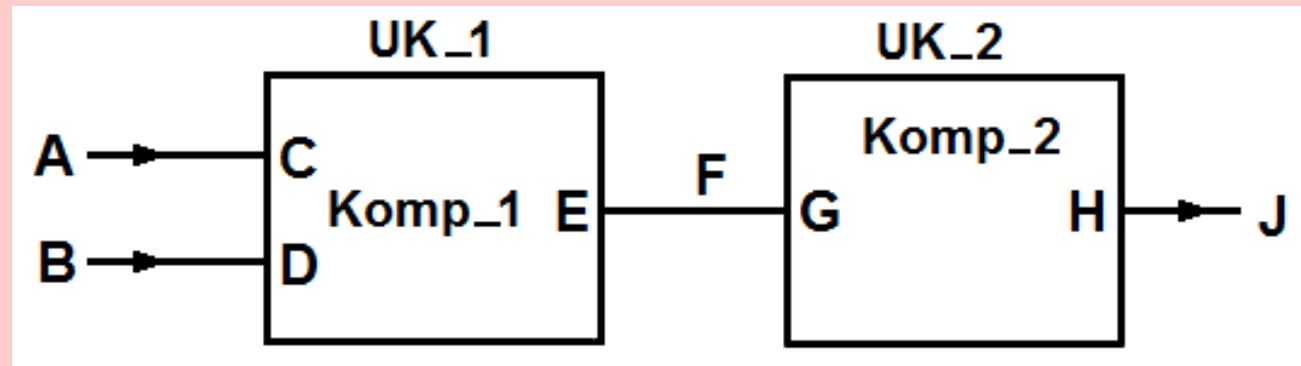
Mamy poniższy układ, który posiada 2 wejścia: **A** i **B**:

- sygnały wejściowe wpływają do wejść **C** i **D** układu **UK_1**,
- **UK_1** na ich podstawie wytwarza sygnał na wyjściu **E**,
- to wyjście jest połączone z sygnałem wewnętrznym **F**,
- **F** wraz wpływa na wejście **G** układu **UK_2**,
- **UK_2** na podstawie sygnału na wejściu **G** wytwarza sygnał na wyjściu **H**,
- z tego wyjścia wypływa sygnał wyjściowy **J**.

Opisaliśmy układ śledząc przepływ sygnałów od wejść do wyjścia – wystąpił **sygnał wewnętrzny**.

Ten styl opisu układu zwany jest opisem **ścieżki danych** – lub opisem **strukturalnym**.

Jest on stosowany na poziomie pliku top-level, nie interesuje nas struktura komponentów. Jest to „sposób zapisu schematu” w VHDL.



Opis strukturalny jest ekwiwalentem schematu układu zbudowanego z bloków (**komponentów**) i połączeń (**sygnałów**). Np. **Komp_1**, **Komp_2** i **F**

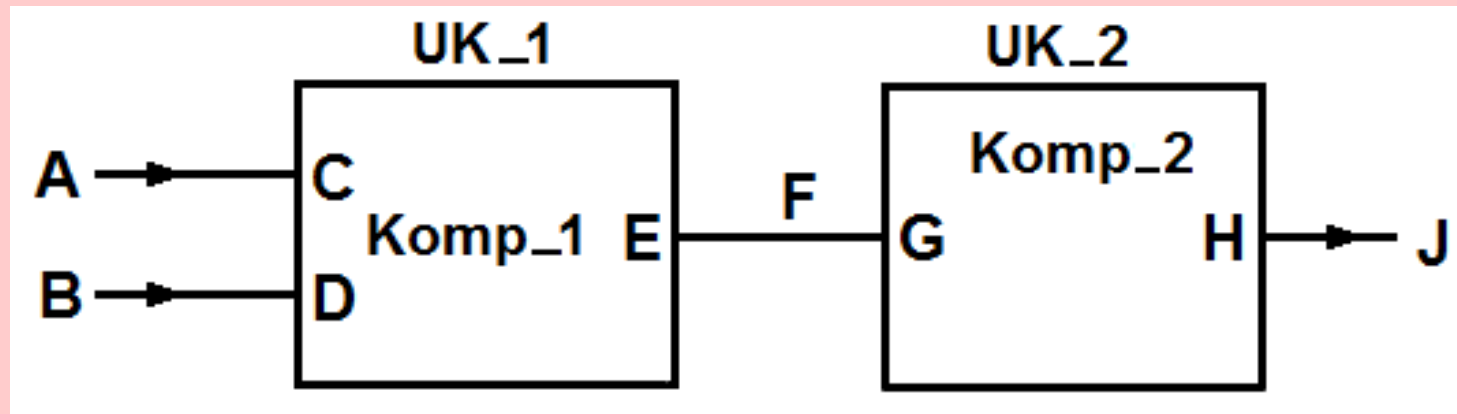
Każdy komponent jest zdefiniowany schematem lub kodem w oddzielnym pliku umieszczonym w katalogu projektu **work** i zadeklarowany w części deklaracyjnej architektury.

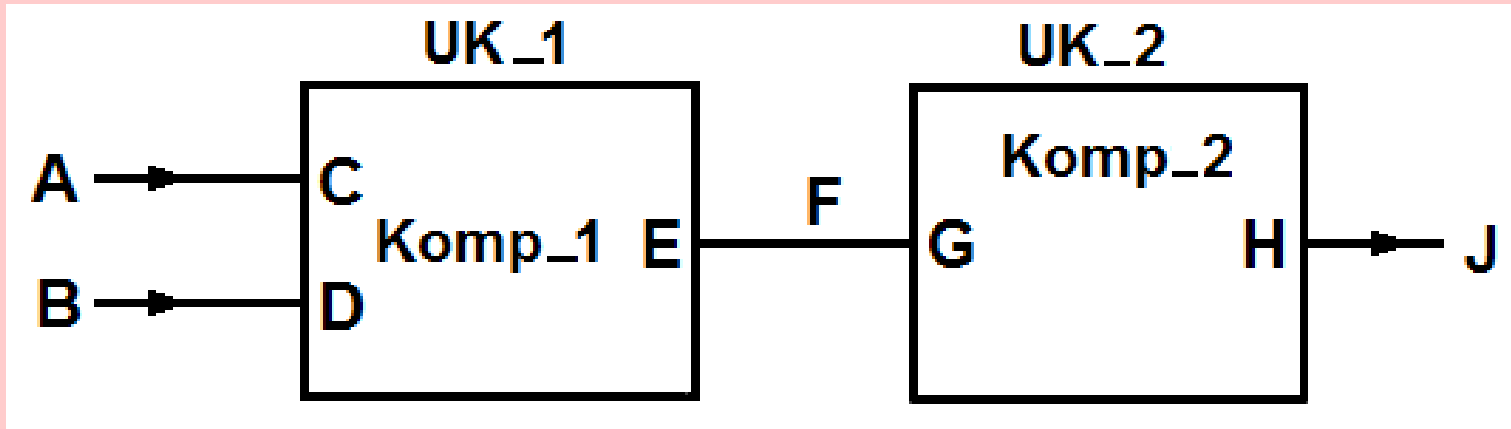
Do opisu połączenia komponentów służy instrukcja **port map** (**konkretyzacji komponentu**).

Składnia instrukcji jest następująca:

Etykieta : ***nazwa_komponentu** PORT MAP (lista połączeń sygnałów);*

Piszemy w kodzie tyle instrukcji ile jest komponentów w układzie





Istnieją dwa schematy listy połączeń:

Lista połączeń opisuje podłączenie portów w komponencie - w sposób jawny (**asocjacja nazwy**).

port_wewnętrzny_komponentu => **sygnał_zewnętrzny** lub **stała**;

Np.:

UK_1 : Komp_1 port map (C => A, D => B, E => F);

W tej konwencji po **lewej stronie** znaku "=>„ umieszczamy nazwę portu w komponencie, zaś po prawej stronie nazwę sygnału (**portu** lub wartość **stałej**) do którego łączymy komponent.

Lista połączeń może wystąpić w sposób niejawny (**asocjacja pozycji**), wtedy w nawiasie podajemy tylko listę nazw sygnałów/stałych zewnętrznych – zachowując kolejność elementów listy taką samą jak porty w pliku definiującym komponent.

Np.:

UK_2 : Komp_2 port map (F, J);

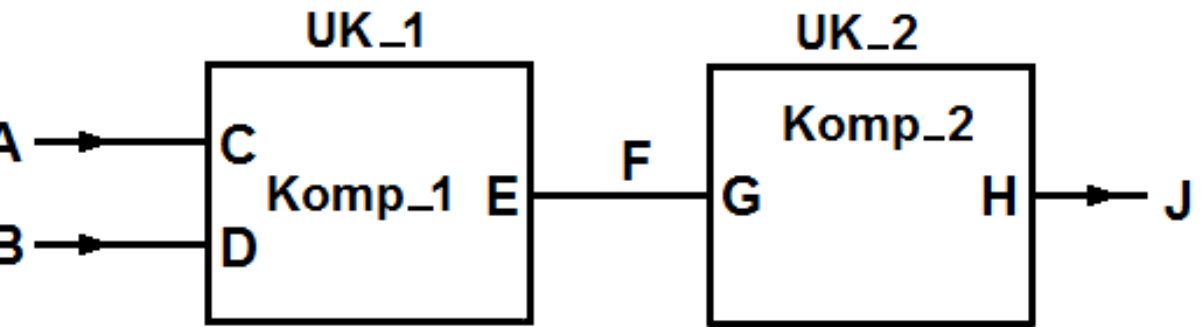

```

library ieee;
use ieee.std_logic_1164.all;
entity data_path is
    port (A, B : in std_logic;
          J : out std_logic);
end data_path;
architecture rtl of data_path is
    component Komp_1
        port (C , D : in std_logic;
              E  : out std_logic);
    end component;
    component Komp_2
        port (G : in std_logic;
              H : out std_logic);
    end component;
    signal F : std_logic;
begin
    UK_1 : Komp_1 port map ( C => A, D => B, E => F );
    UK_2 : Komp_2 port map (      F,      J );
end rtl;

```

-- deklaracja komponentu układu UK_1

-- deklaracja komponentu układu UK_2



Instrukcja procesu.

Jest to jedna z ważniejszych złożonych instrukcji VHDL.

Zawiera w sobie ciąg instrukcji wykonywanych sekwencyjnie - po kolei.

Na zewnątrz proces jako całość widoczny jest jako instrukcja współbieżna.

Większość instrukcji używanych w procesie należą do grupy instrukcji sekwencyjnych - oznacza to, że nie mogą wystąpić w żadnym innym miejscu.

Oprócz instrukcji sekwencyjnych wewnątrz procesu można też stosować również instrukcje współbieżne - jednak w tym przypadku ich wykonanie będzie sekwencyjne.

Instrukcja procesu składa się z trzech elementów:

- listy czułości / wrażliwości (sensitivity list),
- części deklarycyjnej (declarative part),
- części opisowej procesu (ciała procesu).

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY nand2 IS
    PORT( a, b : IN std_logic;
          c : OUT std_logic);
END nand2;
ARCHITECTURE nand2 OF nand2 IS
    BEGIN
        PROCESS ( a, b )
            VARIABLE temp : std_logic;
            BEGIN
                temp := NOT (a and b);
                IF (temp = '1') THEN
                    c <= temp AFTER 6 ns;
                ELSIF (temp = '0') THEN
                    c <= temp AFTER 5 ns;
                ELSE
                    c <= temp AFTER 6 ns;
                END IF;
            END PROCESS;
        END nand2;
```

-- lista czułości
-- część deklaracyjna
-- ciało procesu

Zmienne reprezentują symboliczny element pamięciowy, który może przechowywać i modyfikować wartości, służą one do przechowywania wyników pośrednich.

Nie ma bezpośredniego odwzorowania (w odróżnieniu od sygnałów) zmiennych w realizowanym układzie cyfrowym.

Zmienne można deklarować i wykorzystywać jedynie w procesach i podprogramach.

Zmienna przed użyciem musi być zadeklarowana w sposób następujący:

variable nazwa_zmiennej : typ [ograniczenie_zakresu][:= wartość_początkowa];

Ograniczenie zakresu i nadanie wartości początkowej obowiązuje na tych samych zasadach jak w przypadku sygnałów. Przykłady deklaracji zmiennych:

variable pp : **integer range** 256 **downto** 8 := 100;

Wartość zmiennej powstaje w wyniku operacji przypisania do zmiennej:

z1 := x + 1;

Operacja przypisania wartości posiada inną składnię niż w przypadku sygnałów.

Inne przykłady deklaracji zmiennych:

```
variable u    : integer           range 0 to 127 := 5;  
variable pp   : integer           range 256 downto 8    := 100;  
variable K, S : std_logic_vector(7 downto 0);  
signal A      : std_logic_vector(7 downto 0);  
variable B    : std_logic_vector(7 downto 0);
```

Zmienne można przypisywać do sygnałów a sygnały do zmiennych:

```
A <= B;  
K := A;  
pp := u + 9;  
A(7 downto 4) <= B(0 to 3);
```

Należy pamiętać, iż użycie zmiennych jest możliwe tylko w procesach.

Zaliczenie częściowe wykładu nr 10 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

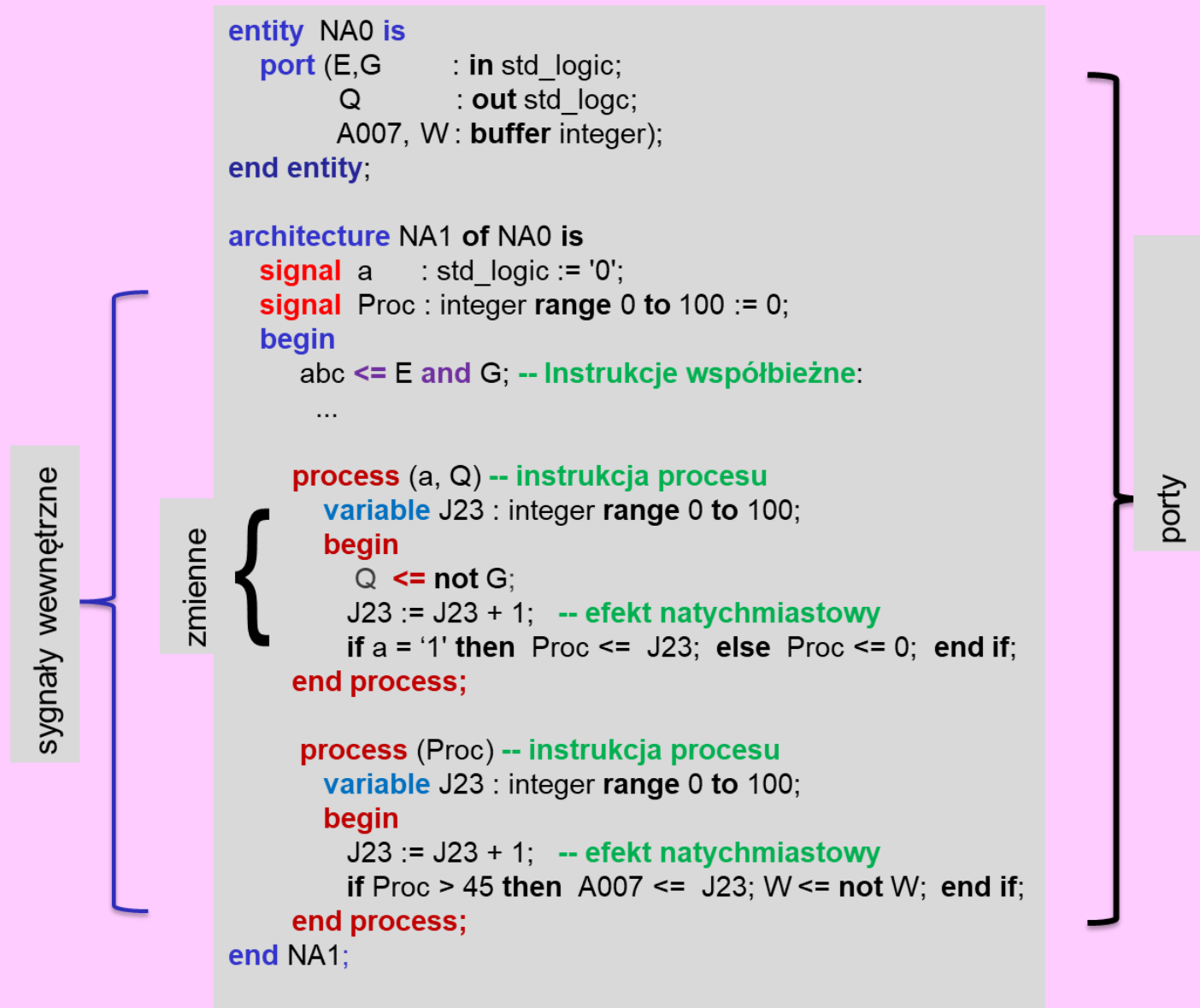
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

Zasięg obiektów w VHDL:



Instrukcja procesu c.d.:

Istnieją dwie struktury instrukcji procesu: z **listą czułości** w wierszu wywołującym i instrukcją **wait**.
Składnia pierwszej konstrukcji jest następująca:

```
[etykieta_procesu:] process (lista czułości)  
    część deklaracyjna  
    begin  
        instrukcje wykonywane sekwencyjnie  
    end process;
```

zaś struktura z instrukcją **wait** posiada układ:

```
[etykieta_procesu:] process  
    część deklaracyjna  
    begin  
        instrukcje wykonywane sekwencyjnie  
        wait on (lista czułości)  
        instrukcje wykonywane sekwencyjnie  
    end process;
```

Proces może znajdować się w dwóch stanach: **zawieszenia** lub **wykonywania**.

Zmiana sygnału oznacza wystąpienie zbocza narastającego lub opadającego.

W przypadku pierwszej konstrukcji jego wykonywanie przebiega następująco:

- | | | |
|----|--------------------------------|---|
| 1. | oczekiwanie na zmianę sygnału | <i>process (C,R)</i> |
| | | <i>variable tq : bit;</i> |
| | | <i>begin</i> |
| 2. | wykonanie pierwszej instrukcji | <i>if R = '1' then</i> |
| 3. | wykonanie kolejnej instrukcji | <i>tq := '0';</i> |
| | | <i>elsif (rising_edge(C) and T) then</i> |
| | ... jak wyżej ... | <i>tq := not tq;</i> |
| | | <i>end if;</i> |
| 4. | wykonanie ostatniej instrukcji | <i>Q <= tq;</i> |
| 5. | powrót do 1. | <i>end process;</i> |

Komunikacja między procesami:

proces jest
instrukcją
współbieżną

```

architecture NA1 of NA0 is
  signal a      : std_logic := '0';
  signal Proc : integer range 0 to 100 := 0;
  begin
    ...

    process (a, Q) -- instrukcja procesu
      variable J23 : integer range 0 to 100;
      begin
        Q <= not G;
        J23 := J23 + 1; -- efekt natychmiastowy
        if a = '1' then Proc <= J23; else Proc <= 0; end if;
      end process;

    process (Proc) -- instrukcja procesu
      variable J23 : integer range 0 to 100;
      begin
        J23 := J23 + 1; -- efekt natychmiastowy
        if Proc > 45 then A007 <= J23; W <= not W; end if;
      end process;
  end NA1;

```

Operatory w VHDL:

W VHDL stosuje się następujące rodzaje operatorów predefiniowanych:

- operatory przypisania (podstawienia),
- operatory logiczne,
- operatory arytmetyczne,
- operatory relacji,
- operatory przesunięcia i rotacji,
- operatory konkatencji i agregacji.

Operatory przypisania wykorzystywane są do nadawania wartości sygnałom, zmiennym i stałym:

- `<=` operator służący do nadawania wartości sygnałom,
- `:=` operator używany do nadawania wartości: stałym, zmiennym oraz początkowym zmiennym i sygnałom,
- `=>` operator używany do selektywnego ustawiania/zerowania elementów wektora - klauzula **OTHERS** / **ELSE**.

Przykłady:

```
SIGNAL a : STD_LOGIC; VARIABLE y : STD_LOGIC_VECTOR(3 DOWNTO 0); SIGNAL w : STD_LOGIC_VECTOR(0 TO 7);
a <= '1';           -- przypisanie bieżącej wartości '1' do sygnału a
y := "0000";        -- przypisanie bieżącej wartości "0000" do zmiennej y
w <= (0 => '1', OTHERS => '0'); -- LSB jest równe '1'.
```

Operatory logiczne stosowane są na danych typu BIT, STD_LOGIC, UNSIGNED oraz pochodne.

W przypadku danych wektorowych, operacje wykonują się na odpowiadających pozycyjnie bitach. Długości operandów powinny być identyczne.

W VHDL operatorami logicznymi, zapisanymi wg priorytetu, są:

- NOT, AND, OR, NAND, NOR, XOR, XNOR.

Przykłady: $y \leq \text{NOT } a \text{ AND } b$; $y \leq \text{NOT } (a \text{ AND } b)$; $z := "0110" \text{ and } "1010"$

Operatory relacyjne służą do realizacji porównań danych tego samego typu – są to operatory:

- = - równy,
- /= - różny,
- < - mniejszy,
- > - większy,
- <= - mniejszy lub równy,
- >= - większy lub równy.

W przypadku wektorów logicznych (np. std_logic_vector) porównanie następuje element po elemencie od lewej strony (w komparatorach może być inaczej). Jeżeli jeden z nich jest krótszy, jest uznawany za element mniejszy....

Operatory arytmetyczne stosowane są do danych typu: INTEGER, SIGNED, UNSIGNED i REAL.

Jeżeli używane są pakiety **std_logic_signed** lub **std_logic_unsigned**, wówczas możliwe są operacje sumowania i odejmowania na danych typu **STD_LOGIC_VECTOR**.

W tabeli zawarto wykaz obsługiwanych operatorów arytmetycznych.

Operator	Description	Example
+	Addition	out1 <= in1 + in2;
−	Subtraction	out1 <= in1 − in2;
*	Multiplication	out1 <= in1 * in2;
/	Division	out1 <= in1/in2;
abs	Absolute Value	absin1 <= abs(in1);
mod	Modulus	modin1 <= mod(in1);
rem	Remainder	remin1 <= rem(in1);
**	Exponent	out1 <= in1 ** 3;

Operatory używane do przesuwania wektorów logicznych. Składnia jest następująca:

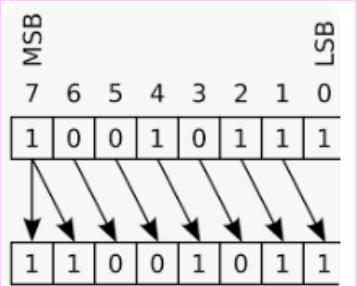
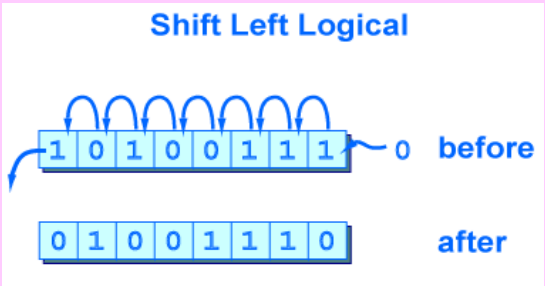
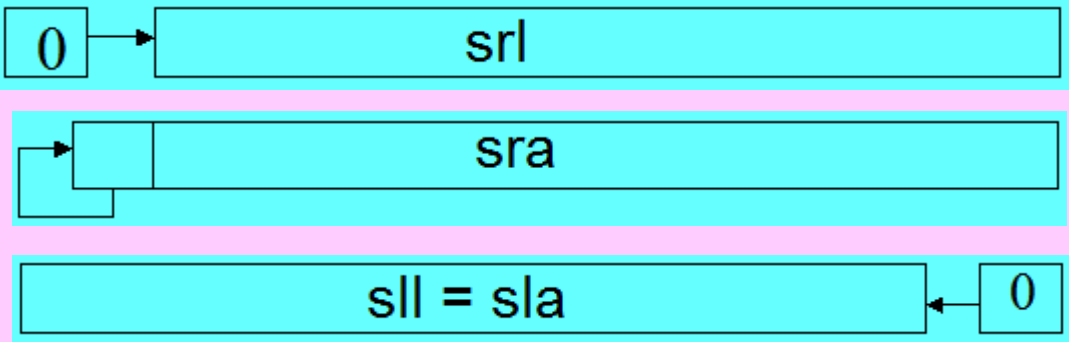
<lewy operand> <rodzaj operacji> <prawy operand>

lewy operand - typu BIT_VECTOR (uwaga STD_LOGIC_VECTOR) jest przesuwaną wartością,

prawy operand - musi być typu INTEGER, określa liczbę (pozycji) przesunięć,

rodzaj operacji - jeden z podanych w tabeli.

Operator	Description	Example
sll	Shift Left Logical	reg <= reg sll 2;
srl	Shift Right Logical	reg <= reg srl 2;
sla	Shift Left Arithmetic	reg <= reg sla 2;
sra	Shift Right Arithmetic	reg <= reg sra 2;
rol	Rotate Left	reg <= reg rol 2;
ror	Rotate Right	reg <= reg ror 2;



Sekwencyjna instrukcja **if-then-else** pozwala ona wybrać zestaw instrukcji, wykonywanych w zależności od podanych spełnienia warunków.

```

if (warunek 1) then           -- warunkiem jest wyrażenie dające wartość true/false
    ...                         -- wykonuje się pierwszy ciąg instrukcji
elsif (warunek 2) then       -- niespełniony pierwszy warunek
    ...                         -- wykonuje się inny zestaw instrukcji
elsif (warunek 3) then       -- niespełnione pierwszy i drugi warunek
    ...                         -- wykonuje się jeszcze inny zestaw instrukcji
    ...                         -- itd. - warunków elsif może być wiele
else                           -- niespełniony z żadnych poprzednich warunków
    ...                         -- wykonaj jeszcze inne instrukcje
end if;

```

Instrukcja jest typowo hierarchiczna:

- żeby warunek zaistnieć poprzedzające go nie mogą być spełnione,
- może wystąpić dowolna liczba warunków *elsif*,
- można pominąć tylko warunki *elsif* oraz można pominąć tylko warunek *else*,
- można pominąć warunek *else* i warunki *elsif*.

```
architecture deko of dekode is  
begin  
  process ( s )  
    begin  
      if s = "00" then  
        x <= "0001";  
      elseif s = "01" then  
        x <= "0010";  
      elseif s = "10" then  
        x <= "0100";  
      else  
        x <= "1000";  
      end if;  
    end process;  
end deko;
```

Sekwencyjna instrukcja **CASE** steruje wyborem zestawu instrukcji sekwencyjnych do wykonania (*pierwszy zestaw, drugi zestaw, ...*) za pomocą wartości (*value1_of_identifier, value2_of_identifier, ...*) klucza *identifier*. Składnia instrukcji jest następująca:

```
case identifier is  
  when value1_of_identifier =>  
    pierwszy zestaw instrukcji sekwencyjnych;  
  when value2_of_identifier =>  
    drugi zestaw instrukcji sekwencyjnych;  
  ... -- itd. kolejne warunki wyboru i zestawy realizowanych instrukcji  
  when others =>  
    wykonaj inne instrukcje; -- jeżeli nie spełniony żaden z wcześniejszych warunków  
end case;
```

Obiekt *identifier* musi przyjmować wartości dyskretne lub powinien być jednowymiarową tablicą.

Warunki *when* można skracać używając klauzuli *when others*, którą można pominąć.

```
architecture arch_1 of dekodery_priorytetowy is  
begin  
    process (r)  
        begin  
            case r is  
                when "1000" | "1001" | "1010" =>  
                    code <= "11";  
                when "0100" =>  
                    code <= "10";  
                when "0010" =>  
                    code <= "01";  
                when others =>  
                    code <= "00";  
            end case ;  
        end process ;  
        active <= r(3) or r(2) or r(1) or r(0);  
end arch_1;
```

Instrukcje case i if mogą wzajemnie zagnieżdżać się:

```

architecture arch_1 of dekodery_priorytetowy is
  type stany_USA is (Oregon, Nebraska);
  signal r : stany_USA;
  begin
    process (Clk, r, Reset)                                -- Clk = 1 kHz, czyli T = 1 ms
      variable n : integer := 0;
      begin
        if Reset = '1' then r <= Oregon;
        elsif rising_edge(Clk) then
          case r is
            when Oregon =>
              Pejzaz <= Gory;
              n := n + 1;
              if n = 10 then
                n := 0; r <= Nebraska;
              end if;
            when Nebraska =>
              Pejzaz <= Step; r <= Oregon;
          end case ;
        end if;
      end process ;
      active <= r(3) or r(2) or r(1) or r(0);
    end arch_1;

```

Instrukcja pętli jest używana, żeby wielokrotnie powtórzyć kilka instrukcji sekwencyjnych. Istnieją 2 wersje **loop**:
[*loop-label* ':'] [**while** *boolean-expression* | **for** *identifier in discrete-range*]

loop

{ sequential statement }

end loop [*loop-label*] ';'

- z określoną liczbą iteracji - z warunkiem **for**,
- z określonym warunkiem iteracji – z warunkiem **while**.

Jeżeli pętla powinna być wykonana znaną, z góry określoną liczbę razy, wówczas stosuje się schemat iteracji z **for**, którego składnia jest następująca:

Wskaźnik pętli wraz z jej każdym przebiegiem zmienia dyskretnie swoje wartości, zaczynając od wartości lewej, np.:

```
for count_value in 0 to 127 loop  
    count_out <= count_value;  
    ... ;  
end loop;
```

Instrukcja wyjścia z pętli. Pętla jest opuszczana gdy zostanie spełniony warunek w instrukcji **exit**, której składnia jest następująca:

```
[etykieta:] exit [etykieta_pętli] [ when wyrażenie_boolowskie];
```

Występujące w instrukcji *wyrażenie_boolowskie* jest najczęściej wyrażeniem relacyjnym wytwarzającym wartość *true* lub *false*.

Jeżeli instrukcja `exit` jest zagnieżdżona w konstrukcji `for`, wówczas po wyjściu z pętli wykonuje się kolejna instrukcja pod instrukcją pętli .

```
for i in 0 to max_limit loop  
  if (a(i) = 0) then exit;  
  end if;  
  g(i) := a(i);  
end loop;  
...
```

Natychmiastowe
opuszczenie
pętli



Jeżeli liczba iteracji pętli zależy od spełnienia warunku przed kolejnym powtórzeniem pętli, wówczas stosuje się konstrukcję **while - loop**:

```
[etykieta_pętli :] while [wyrażenie_boolowskie] loop  
    {instrukcje_sekwencyjne};  
end loop [etykieta_petli];
```

Warunek (wyrażenie_boolowskie) jest sprawdzany, przed każdym, **w tym pierwszym**, powtórzeniem pętli.

Przykład konstrukcji:

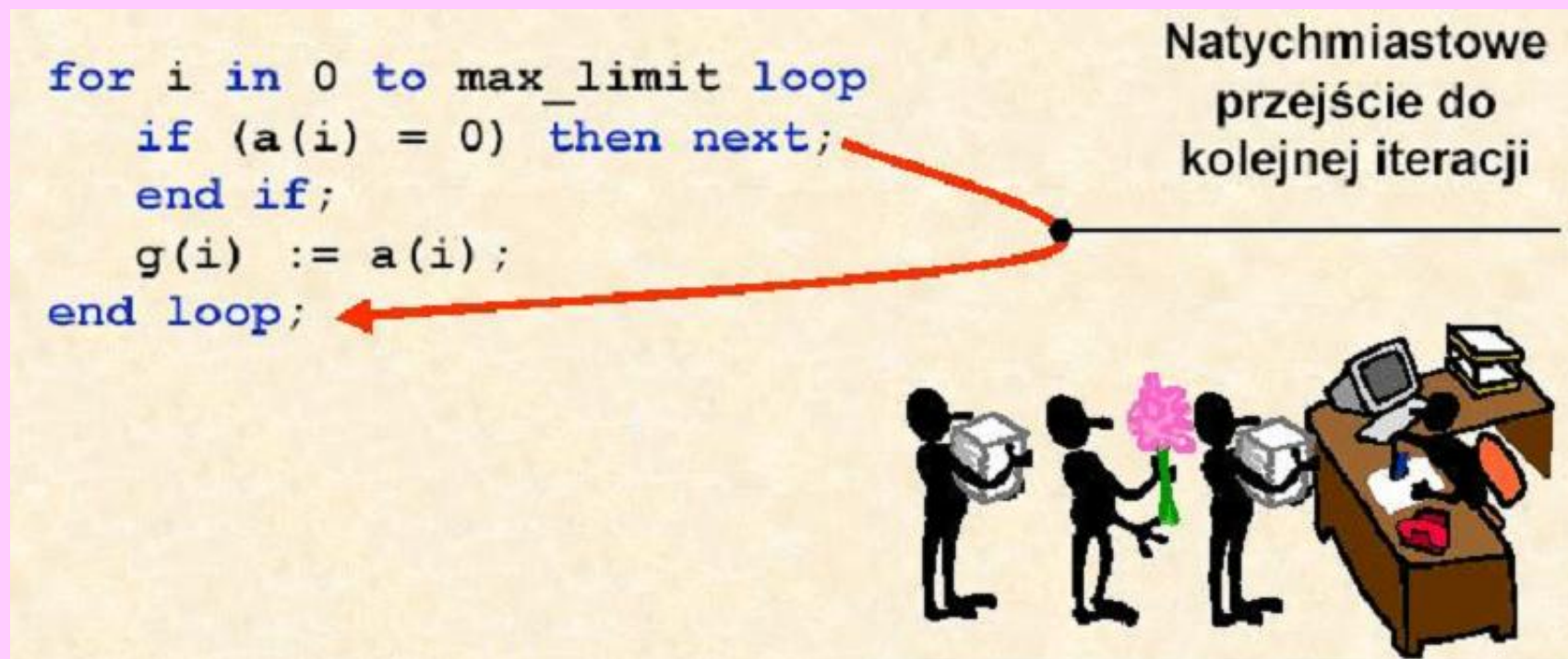
```
WHILE (i < 10) LOOP  
    WAIT UNTIL clk'EVENT AND clk = '1';  
    (other statements)  
END LOOP;
```

Instrukcja **next** wyjścia z pętli służy do zakończenia wykonania bieżącej iteracji pętli (pominięcie dalszych instrukcji) i przeskok jej końca, zależy to od spełnienia **warunku wyjścia**.

Jednocześnie indeks pętli zmienia się o jeden.

Składnia instrukcji jest następująca:

***next** [etykieta_pętli] [when wyrażenie_boolowskie];*

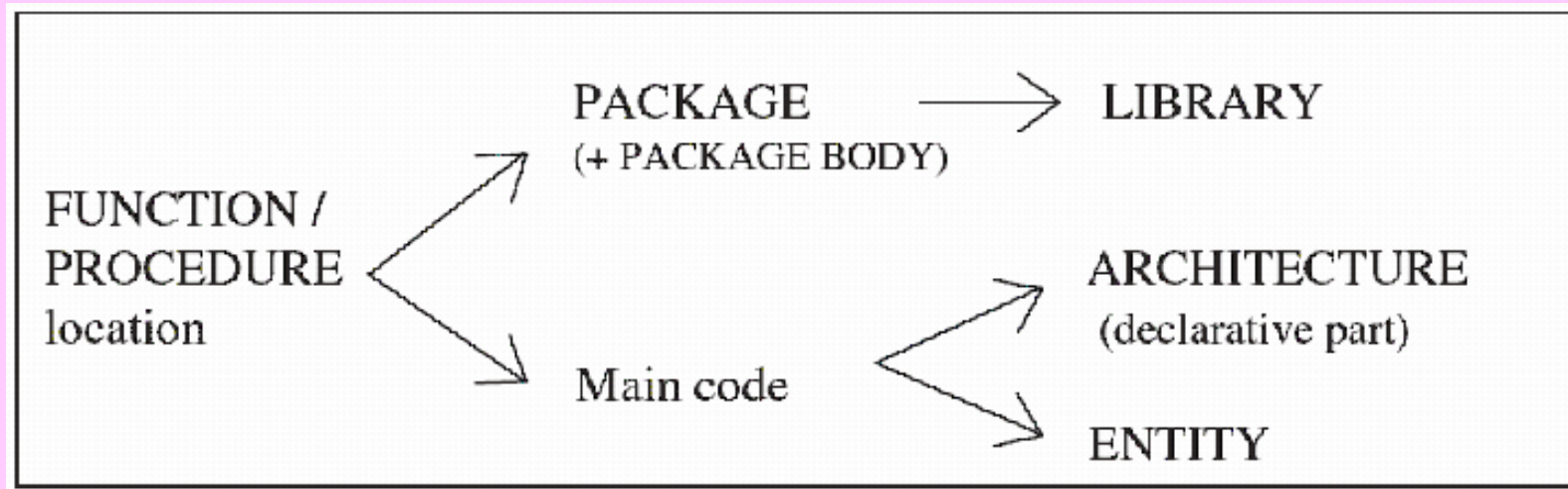


FUNKCJE i PROCEDURY należą do podprogramów. Pod względem składni podobne są do PROCESU. Różnią się jednak pod względem aplikacyjnym.

PROCES można implementować tylko w architekturze zaś podprogramy głównie w bibliotekach – chociaż możliwa jest również ich implementacja w architekturze.

Stanowią one zwartą funkcjonalnie część kodu VHDL, zawierającą instrukcje sekwencyjne – oprócz instrukcji WAIT.

Schematycznie, miejsce lokalizacji podprogramów przedstawiono na poniższym rysunku:



Funkcje:

- są częścią sekwencyjną kodu,
- służą do rozwiązywania prostych problemów – konwersja danych, obliczenia, itp.,
- mogą być wielokrotnie przywołane,
- skracają i upraszczają kod główny.

Wykonywane wewnątrz funkcji instrukcje sekwencyjne nie mogą zawierać:

- instrukcji WAIT,
- deklaracji SYGNAŁÓW,
- łączenia KOMPONENTÓW.

Użycie funkcji wymaga spełnienia następujących czynników:

- deklaracji funkcji (declaration),
- definicji funkcji (function body),
- wywołania funkcji (call to the function).

Składnia **definicji funkcji** jest następująca:

```
FUNCTION nazwa_funkcji [<lista_parametrów>] RETURN typ_zwracanych_danych IS  
[część_deklaracyjna_funkcji]  
BEGIN  
(instrukcje_sekwencyjne) -- opis działania funkcji  
END nazwa_funkcji;
```

Nazwa_funkcji może być identyfikatorem lub symbolem operatora („byte_to”, „+”, itp.).

Lista_parametrów (formalnych) w definicji funkcji może zawierać:

- stałe,
- sygnały,
- **nie może** zawierać zmiennych.

Parametry funkcji muszą być typu **in** (w przypadku portów). Nie wolno podawać zakresów dla:

- integer (RANGE),
- std_logic_vector (DOWNTO).

```
FUNCTION nazwa_funkcji [<lista_parametrów>] RETURN typ_zwracanych_danych IS  
[część_deklaracyjna_funkcji]  
BEGIN  
(instrukcje_sekwencyjne) -- opis działania funkcji  
END nazwa_funkcji;
```

W momencie wywołania funkcja wykonuje obliczenia na konkretnych wartościach (**parametrach aktualnych**), które podstawiane są w miejsce parametrów formalnych.

Wynik obliczeń zwracany jest przez obiekt w formacie *typ_zwracanych_danych*.

W części deklaracyjnej funkcji można deklarować:

- stałe
- zmienne,
- **nie wolno** deklarować sygnałów.

Deklaracja funkcji (występująca w pakietach):

```
FUNCTION AnyZeros (CONSTANT v : in bit_vector) RETURN boolean;
```

Definicja funkcji:

```
FUNCTION AnyZeros (CONSTANT v : in bit_vector) RETURN boolean is  
begin  
  for i in v'range loop  
    if v(i) = '1' then  
      RETURN true;  
    end if;  
  end loop;  
  RETURN false;  
end FUNCTION AnyZeros;
```

Opis działania funkcji **musi zawierać** chociaż jedną instrukcję **RETURN**

Instrukcja **return** używana jest do powrotu z podprogramu.

Procedura nie musi zawierać instrukcji **return**, jeżeli jej nie zawiera wykonanie procedury kończy się na ostatniej instrukcji.

Instrukcja **return** kończy wykonywanie funkcji, poprzez **wyrażenie** zwracając wynik funkcji (wartość zadeklarowanego typu):

```
return [ wyrażenie ] ;
```

Np.:

```
function byte_to_int(byte : word_8) return integer is  
    variable result : integer := 0;  
  
begin  
    for index in 0 to 7 loop  
        result := result*2 + bit'pos(byte(index));  
    end loop;  
    return result;  
end byte_to_int;
```

Funkcja sama w sobie **nie stanowi kompletnej instrukcji**.

Stąd wywołanie funkcji polega na umieszczeniu jej nazwy w kodzie innej instrukcji programu lub w części innego wyrażenia.

Wywołując funkcję należy podać parametry aktualne na których zostanie ona wykonana:

```
...  
q <= Func (p1 => v1, p2 => v2) ;    -- asocjacja nazwy – jawne zadanie parametrów  
q <= Func (v1, v2) ;                -- asocjacja pozycji – poprzez przyporządkowanie  
...
```

Wartość funkcji w momencie wywołania nie musi być przypisaniem do sygnału, może stanowić część składową instrukcji, np.:

```
...  
IF x > maximum(a, b) ...  
...
```

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
ENTITY dff IS  
    PORT ( d, clk, rst: IN STD_LOGIC;  
           q: OUT STD_LOGIC);  
END dff;  
ARCHITECTURE my_arch OF dff IS  
    FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS  
        BEGIN  
            RETURN s'EVENT AND s='1';  
        END positive_edge;  
  
    BEGIN  
        PROCESS (clk, rst)  
            BEGIN  
                IF (rst='1') THEN  
                    q <= '0';  
                ELSIF positive_edge(clk) THEN  
                    q <= d;  
                END IF;  
            END PROCESS;  
  
    END my_arch;
```

----- **Package:** ----- zdefiniowana przez nas biblioteka

LIBRARY ieee;

USE ieee.std_logic_1164.all;

PACKAGE my_package IS -- nazwa pliku biblioteki

FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN; -- deklaracja funkcji

END my_package;

PACKAGE BODY my_package IS

FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS – definicja funkcji

BEGIN

RETURN s'EVENT AND s='1';

END positive_edge;

END my_package;

----- **Main code:** ----- kod naszego projektu

LIBRARY ieee;

USE ieee.std_logic_1164.all; -- ieee jest nazwą katalogu na dysku

USE work.my_package.all; -- dołączenie naszej biblioteki z katalogu projektu

ENTITY dff IS

PORT (d, clk, rst: IN STD_LOGIC; q: OUT STD_LOGIC);

END dff;

ARCHITECTURE my_arch OF dff IS

BEGIN

PROCESS (clk, rst)

BEGIN

IF (rst='1') THEN q <= '0'; ELSIF positive_edge(clk) THEN q <= d; END IF;

END PROCESS;

END my_arch;

----- *Package:* -----

LIBRARY ieee;

USE ieee.std_logic_1164.all;

PACKAGE my_package ***IS***

FUNCTION "+" (a, b: STD_LOGIC_VECTOR) ***RETURN*** STD_LOGIC_VECTOR; -- deklaracja

END my_package;

PACKAGE BODY my_package ***IS***

FUNCTION "+" (a, b: STD_LOGIC_VECTOR) ***RETURN*** STD_LOGIC_VECTOR ***IS*** -- definicja

VARIABLE result: STD_LOGIC_VECTOR;

VARIABLE carry: STD_LOGIC;

BEGIN

 carry := '0';

FOR i ***IN*** a'REVERSE_RANGE ***LOOP***

 result(i) := a(i) XOR b(i) XOR carry;

 carry := (a(i) AND b(i)) OR (a(i) AND carry) OR (b(i) AND carry);

END LOOP;

RETURN result;

END "+";

END my_package;

----- *Main code:* -----

LIBRARY ieee;

USE ieee.std_logic_1164.all;

USE ***work.my_package.all;***

ENTITY add_bit ***IS***

PORT (a: ***IN*** STD_LOGIC_VECTOR(3 ***DOWNTO*** 0);

 y: ***OUT*** STD_LOGIC_VECTOR(3 ***DOWNTO*** 0));

END add_bit;

ARCHITECTURE my_arch ***OF*** add_bit ***IS***

CONSTANT b: STD_LOGIC_VECTOR(3 ***DOWNTO*** 0) := "0011";

CONSTANT c: STD_LOGIC_VECTOR(3 ***DOWNTO*** 0) := "0110";

BEGIN

 y <= a + b + c; -- ***overloaded*** "+" operator

END my_arch;

Procedura jest traktowana jako instrukcja współbieżna lub sekwencyjna w zależności od tego w której części kodu znajduje się (w części współbieżnej architektury czy w procesie).

Procedura może zawierać instrukcje sekwencyjne (z wykluczeniem jak funkcja).

Procedura opisuje zachowanie się projektu, operuje na wszelkiego typu obiektach.

W odróżnieniu od funkcji, procedura może zawierać dowolną liczbę parametrów wejściowych i wyjściowych – może zatem zwracać więcej niż jedną wartość wyniku.

Procedura podobnie jak funkcja składa się z deklaracji procedury i ciała procedury.

Deklaracja procedury jest następująca:

PROCEDURE nazwa_procedury [<lista_parametrów_formalnych>] ***IS***

[część_deklaracyjna_procedury]

BEGIN

(instrukcje sekwencyjne)

END nazwa_procedury ;

Na liście parametrów formalnych mogą się znaleźć następujące typy obiektów:

- constant – tylko rodzaju in,
- signal,
- variable,

Parametry formalne mogą być rodzaju:

- in,
- out,
- inout,
- buffer.

Właściwości procedury:

- do wprowadzania wartości parametrów do procedury służą rodzaje **in** i **inout**,
- do wyprowadzania danych z procedury służą **inout**, **out**, **buffer**,
- wykonanie procedury kończy się po napotkaniu dyrektywy **return** lub **end**,
- w procedurze **return** nie może zwracać wartości,
- wartości w procedurze zwracane są przez parametry formalne.

Procedurę można wywoływać w architekturze jako instrukcję współbieżną, **pod warunkiem**, że żaden z parametrów formalnych **nie jest** typu *variable* – gdyż zmiennych nie można używać poza procesem lub podprogramem,

Wykonanie procedury rozpoczyna się, gdy nastąpi zmiana stanu jednego z parametrów formalnych. Jeżeli procedura jest wywoływana wewnątrz procesu lub podprogramu jest ona traktowana jako instrukcja sekwencyjna. Przykład definicji procedury:

```
PROCEDURE my_procedure ( a: IN BIT; SIGNAL b, c: IN BIT;  
    SIGNAL x: OUT BIT_VECTOR(7 DOWNT0 0);  
    SIGNAL y: INOUT INTEGER RANGE 0 TO 99) IS  
BEGIN  
  
...  
END my_procedure;
```

gdzie:

- **a** - stała typu bit - można pominąć słowo *CONSTANT*,
- **b i c** - sygnały wejściowe,
- **x i y** - sygnały zwracane przez procedurę.

Podczas wywoływania procedury parametry formalne należy zastąpić parametrami aktualnymi. Może to być przyporządkowanie niejawne – asocjacja pozycji:

```
FFJK2(zegar, a, b, reset, set, Q4, Q4not);
```

w którym należy zwrócić uwagę aby liczba i kolejność nazw ściśle odpowiadała ich pozycji w definicji procedury.

Bardziej niezawodnym jest jawne przyporządkowanie parametrów, w którym po **lewej stronie** znaku => występuje **parametr formalny**, zaś po stronie **prawej parametr aktualny** – asocjacja nazwy:

```
FFJK2(C => zegar, J => a, K => b, R => reset, S => set, Q => Q4, QN => Q4not);
```

W takim przypadku nie jest istotna kolejność przyporządkowań parametrów.

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
ENTITY min_max IS  
  GENERIC (limit : INTEGER := 255);  
  PORT ( ena: IN BIT; inp1, inp2: IN INTEGER RANGE 0 TO limit;  
         min_out, max_out: OUT INTEGER RANGE 0 TO limit);  
END min_max;  
  
ARCHITECTURE my_architecture OF min_max IS  
  PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;  
                  SIGNAL min, max: OUT INTEGER RANGE 0 TO limit) IS  
    BEGIN  
      IF (in1 > in2) THEN max <= in1; min <= in2;  
      ELSE max <= in2; min <= in1;  
      END IF;  
    END sort;  
  
  BEGIN  
    PROCESS (ena)  
      BEGIN  
        IF (ena='1') THEN  
          sort(inp1, inp2, min_out, max_out);  
        END IF;  
      END PROCESS;  
  END my_architecture;
```

----- *Package:* -----

LIBRARY ieee;

USE ieee.std_logic_1164.all;

PACKAGE my_package ***IS***

CONSTANT limit: ***INTEGER*** := 255;

PROCEDURE sort (***SIGNAL*** in1, in2: ***IN INTEGER RANGE 0 TO limit***;

SIGNAL min, max: ***OUT INTEGER RANGE 0 TO limit***);

END my_package;

PACKAGE BODY my_package ***IS***

PROCEDURE sort (***SIGNAL*** in1, in2: ***IN INTEGER RANGE 0 TO limit***;

SIGNAL min, max: ***OUT INTEGER RANGE 0 TO limit***) ***IS***

BEGIN

IF (in1 > in2) ***THEN*** max <= in1; min <= in2;

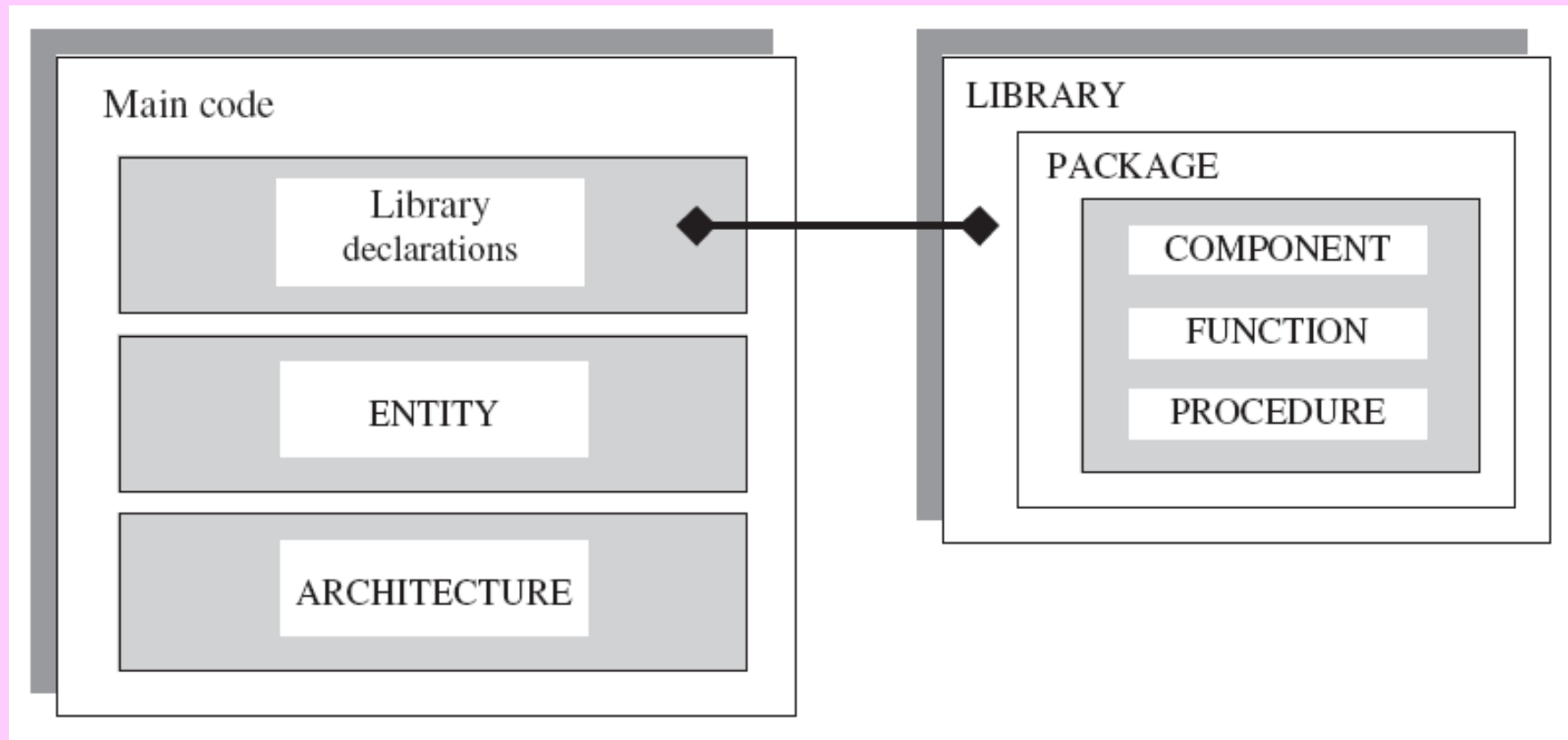
ELSE max <= in2; min <= in1;

END IF;

END sort;

END my_package;

```
----- Main code: -----  
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE work.my_package.all;  
-----  
ENTITY min_max IS  
  GENERIC (limit: INTEGER := 255);  
  PORT ( ena: IN BIT;  
        inp1, inp2: IN INTEGER RANGE 0 TO limit;  
        min_out, max_out: OUT INTEGER RANGE 0 TO limit);  
END min_max;  
-----  
ARCHITECTURE my_architecture OF min_max IS  
  BEGIN  
    PROCESS (ena)  
      BEGIN  
        IF (ena='1') THEN  
          sort (inp1, inp2, min_out, max_out);  
        END IF;  
      END PROCESS;  
END my_architecture;  
-----
```

Często używane fragmenty kodu mogą być wpisywane w struktury KOMPONENTÓW, FUNKCJI lub PROCEDUR, następnie umieszczone w PAKIECIE docelowo są skompilowane w BIBLIOTECE.

Takie podejście pozwala na fragmentaryzację kodu, jego współdzielenie i wielokrotne wykorzystanie.

Oprócz KOMPONENTÓW, FUNKCJI i PROCEDUR w pakietach mogą być zawarte definicje TYPÓW danych i STAŁYCH:

PACKAGE *package_name* **IS** -- część deklaracyjna
 (declarations)

END *package_name*;

[**PACKAGE BODY** *package_name* **IS** -- część opisowa
 (FUNCTION and PROCEDURE descriptions)

END *package_name*;

Jak widać składnia pakietu zawiera dwa elementy:

- część deklaracyjną PACKAGE
- część opisową PACKAGE BODY.

Pierwsza część jest obligatoryjna i zawiera wszystkie deklaracje, druga część jest niezbędna tylko wówczas gdy podprogramy części pierwszej wymagają definicji.

PACKAGE i PACKAGE BODY **muszą** posiadać tę samą nazwę.

Część deklaracyjna (*declarations*) może zawierać: KOMPONENTY, FUNKCJE, PROCEDURY, TYPY DANYCH, STAŁE, itp.

LIBRARY ieee;

USE ieee.std_logic_1164.all;

PACKAGE my_package **IS**

TYPE state **IS** (st1, st2, st3, st4);

TYPE color **IS** (red, green, blue);

CONSTANT vec: STD_LOGIC_VECTOR(7 **DOWNTO** 0) := "11111111";

FUNCTION positive_edge(**SIGNAL** s: STD_LOGIC) **RETURN** BOOLEAN;

END my_package;

PACKAGE BODY my_package **IS**

FUNCTION positive_edge(**SIGNAL** s: STD_LOGIC) **RETURN** BOOLEAN **IS**

BEGIN

RETURN (s'**EVENT** **AND** s='1');

END positive_edge;

END my_package;

Zaliczenie częściowe wykładu nr 11 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

Obszary zastosowań mobilnych układów cyfrowych, w tym programowalnych, jest szeroki.

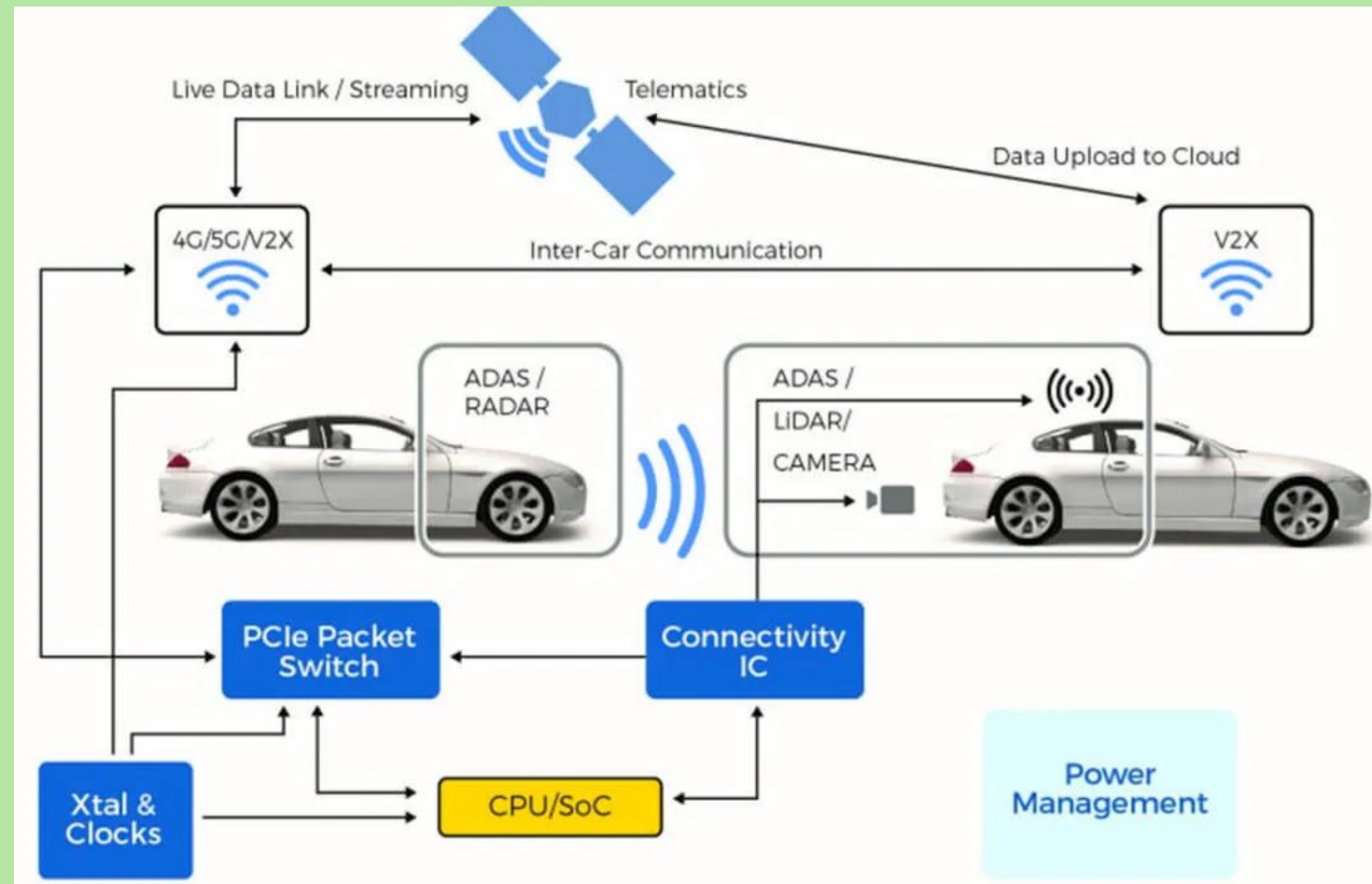


W przypadku samochodów, układy programowalne znajdują zastosowanie w systemach wspomagania kierowcy ADAS (Advanced Driver Assistance Systems). Samochody autonomiczne wykorzystują różne czujniki do wykrywania otoczenia, w tym kamery, radary, lidary, sonary, GPS, i jednostki pomiarowe. Zaawansowane systemy analizują dane z czujników, aby określić trasy podróży, a także przeszkody i niezbędne oznakowanie.

Zgodnie z metodologią opracowaną przez SAE J3016, autonomię pojazdów dzieli się na sześć etapów:

- Poziom 0 – brak automatyzacji,
- Poziom 1 – sterowanie ręczne/wspólne,
- Poziom 2 – bez użycia rąk,
- Poziom 3 – bez użycia wzroku,
- Poziom 4 – bez użycia umysłu,
- Poziom 5 – kierownica opcjonalna.

Pojazdy autonomiczne mogą zwiększać bezpieczeństwo na drogach, zmniejszyć liczbę obrażeń, dostępność dla osób starszych i niepełnosprawnych, eliminacja zmęczenia kierowcy i możliwość snu.



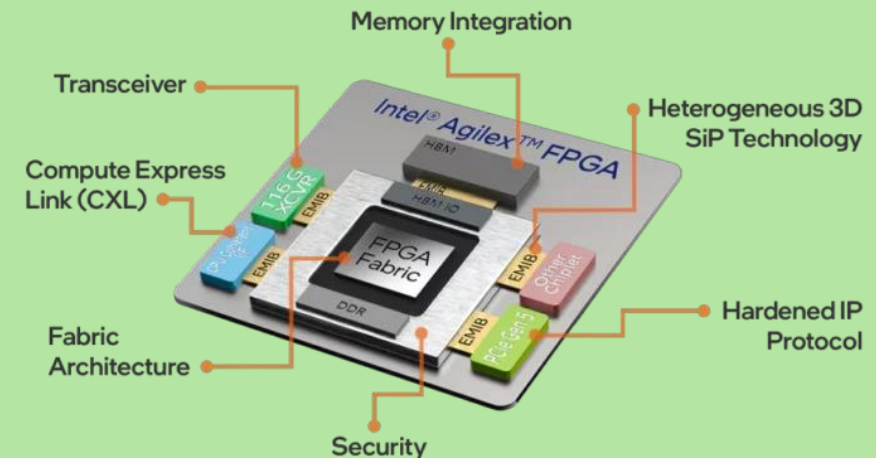
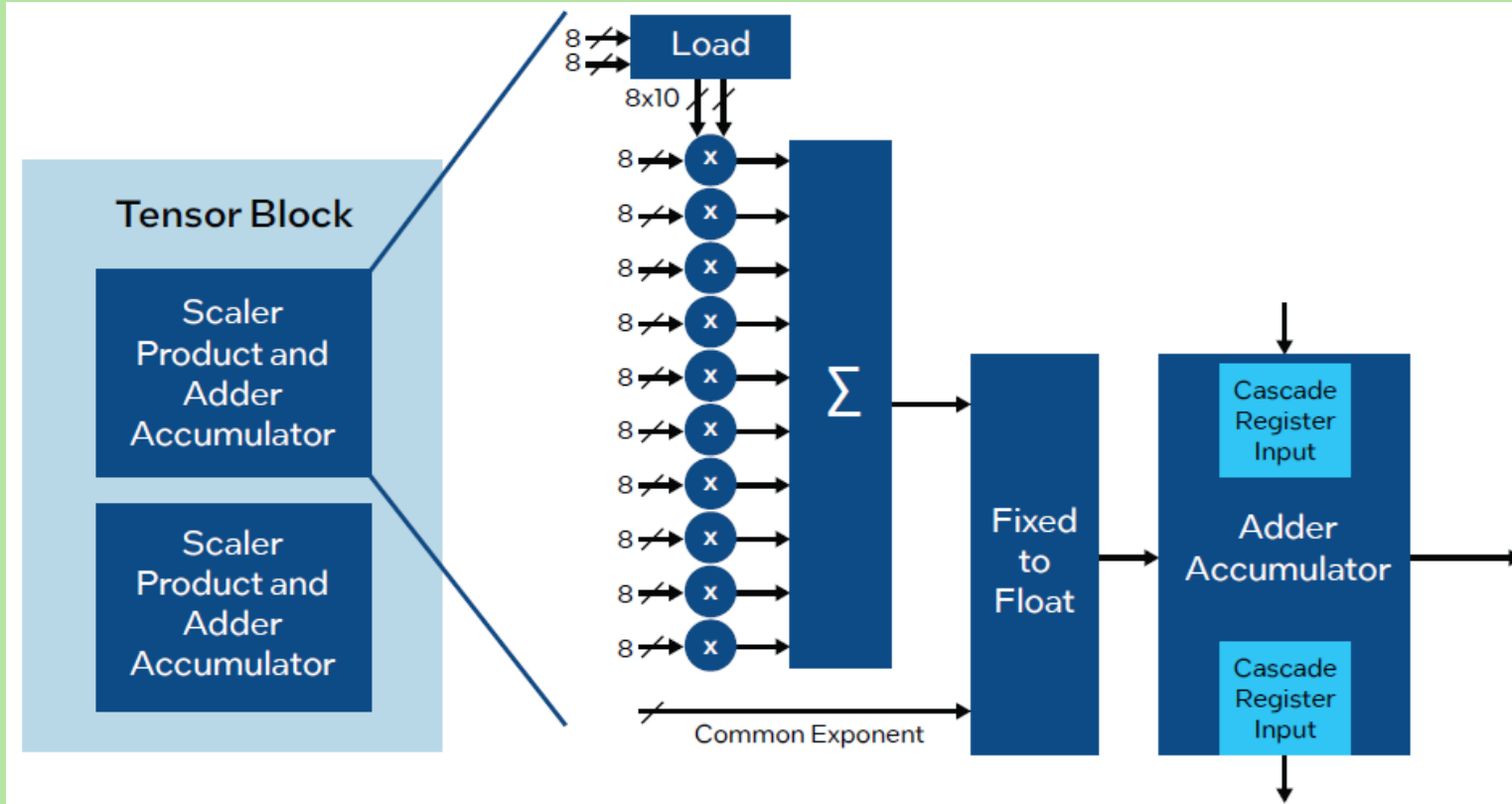
Układy FPGA służyły do projektowania tymczasowych rozwiązań, aż produkcja na tyle wzrośnie, aby obniżyć koszty FPGA i zastąpić je wersją ASIC. Chociaż badania nauki samodzielnego uczenia pojazdu i samodzielnej jazdy trwają od lat 50 - nie zostały jeszcze zakończone. Obecnie ma miejsce wiele badań nad samochodami autonomicznymi w kontrolowanym środowisku, pod nadzorem człowieka i w idealnych warunkach drogowych oraz środowiskowych, ale nadal opracowanie w pełni autonomicznego pojazdu wiąże się z kilkoma wyzwaniami. Nadal istnieją poważne przeszkody dla bezpiecznego wdrożenia w pełni autonomicznych pojazdów.

Zaletą układów FPGA jest to, że pozwalają tworzyć niestandardowe projekty za pomocą języka programowania i wdrażać w FPGA. Nie bez znaczenia jest elektryfikacja branży motoryzacyjnej. Przez lata w pojazdach mechanicznych niewiele znajdowało się układów logicznych. wiele układów logicznych. Jednym z celów projektowania samochodów jest ograniczenie liczby jednostek sterujących (ECUs) do minimum. FPGA pozwalają połączyć procesory CPU i GPU w jednym układzie.



Firma **Intel/Altera** prowadzi prace w zakresie automobilnych zastosowań FPFA w systemach ADAS ale również rozwija aplikacje związane z doświadczeniem w pojeździe na bazie sztucznej inteligencji

Układy FPGA i SoC FPGA są odpowiednie do przyspieszenia prac w zakresie AI, gdyż wydajność, elastyczność i energooszczędność. Układy FPGA firmy Intel/Altera mają specjalistyczne funkcje sztucznej inteligencji wbudowane w strukturę logiczną, min. przez dodanie AI do tradycyjnego bloku DSP FPGA, umożliwiając obsługę aplikacji AI z operacjami wektorowymi i macierzowymi. w skalowalnym układzie Agilex 3,5.



Układy FPGA Intel w zastosowaniach motoryzacyjnych są zorientowane na

Przetwarzanie i fuzję danych z czujników:

- kamer, LiDAR-ów, RADAR-ów;
- czujników ruchu montowanych wokół pojazdu i w kokpicie.

Monitorowanie kierowcy i pasażerów oraz sterowanie głosowe:

- poprzez systemy kamer i czujniki w kabinie pojazdu,
- w ramach działań zapobiegawczych w celu uniknięcia kolizji,
- realizację algorytmów rozpoznawania głosu w celu łatwej interakcji z pojazdem.

Konserwację predykcyjną, diagnostykę i zarządzanie akumulatorami:

- ciągłe monitorowanie czujników, układów elektrycznych i mechanicznych w pojeździe;
- ostrzegania właściciela pojazdu o konieczności przeprowadzenia konserwacji zapobiegawczej lub napraw;
- monitorować stan naładowania akumulatora.



Intel® Agilex™ F-Series
FPGAs and SoCs



Intel® Agilex™ I-Series SoC
FPGAs



Intel® Agilex™ M-Series
SoC FPGAs



Software-Defined Vehicle (SDV)



Advanced Driver-Assistance System (ADAS)



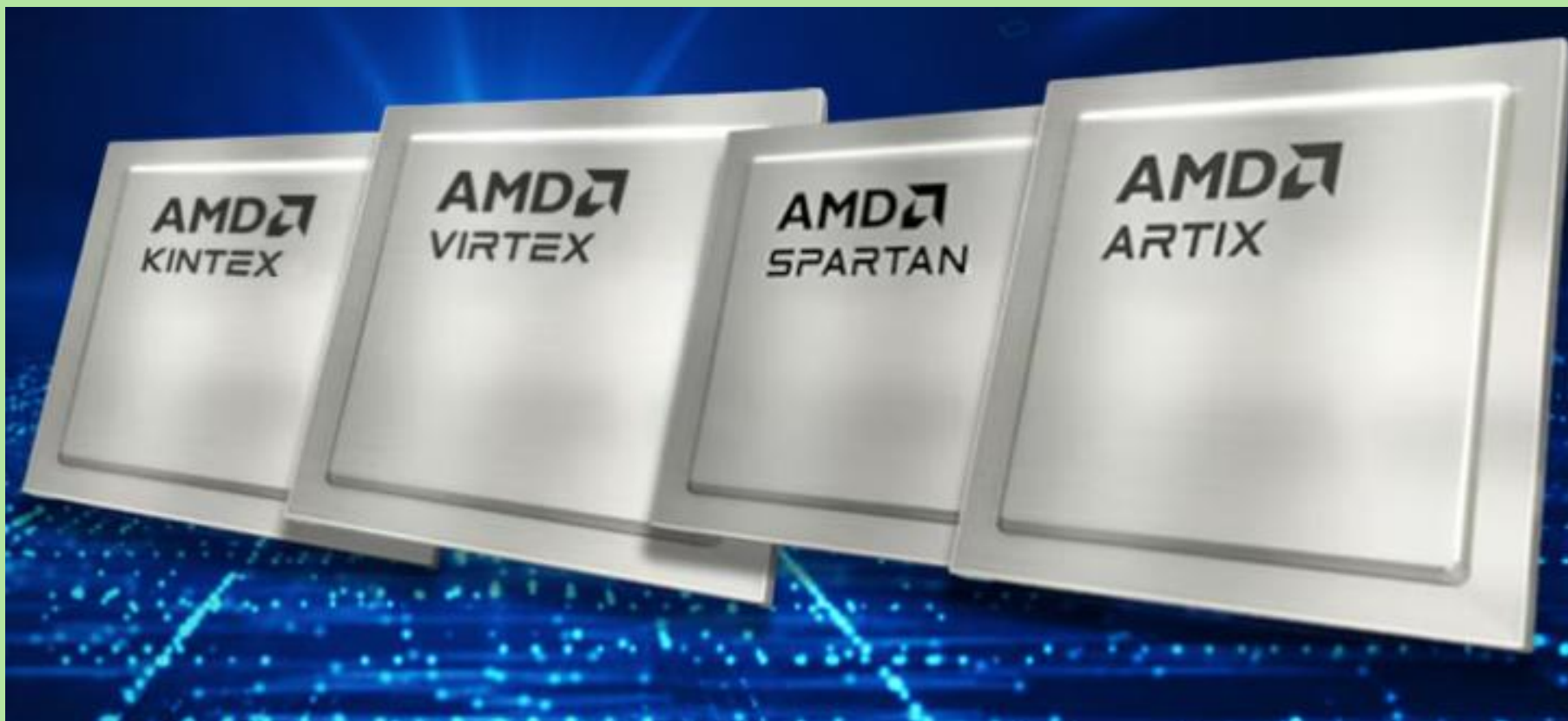
Electric Vehicle (EV) Powertrain

Przekształcenie w pojazd SDV opiera się na założeniu, że oprogramowanie kontroluje, zarządza i ulepsza każdy aspekt działania pojazdu. FPGAs zapewniają rekonfigurowalność, przetwarzanie, bezpieczeństwo i energooszczędność oparte na oprogramowaniu platformy motoryzacyjnej.

Aplikacje ADAS zwiększają bezpieczeństwo i wygodę jazdy poprzez integrację czujników, przetwarzanie danych w czasie rzeczywistym i automatyzację, aby pomóc kierowcom w podejmowaniu bardziej świadomych decyzji i zmniejszeniu ryzyka wypadków.

Układ napędowy pojazdu elektrycznego integruje technologie, które efektywnie przetwarzają energię elektryczną i zarządzają nią, zapewniając optymalną wydajność, zasięg i zrównoważony rozwój, oraz precyzyjną kontrolę pracy silnika i dystrybucji mocy. Układy FPGA zapewniają kontrolę w czasie rzeczywistym i zarządzanie mocą.

AMD/Xilinx oferuje kompleksową gamę produktów, które spełniają wymagania szerokiego zakresu zastosowań. Obszary zastosowań FPGA AMD to: wysokowydajne aplikacje sieciowe o najwyższej przepustowości i mocy obliczeniowej oraz wydajności, ale również tanie układy FPGA o niewielkich rozmiarach do konwersji technologii definiowanej programowo na wyższy poziom.



Układy FPGA automotive firmy AMD:



Automotive-Grade Artix™ 7 XA:

- skalowalna prędkość przetwarzania,
- przechwytywanie klatek wideo,
- korekcja zniekształceń i łączenie w 3D
- dynamiczna rekonfiguracja,
- softprocesor MicroBlaze V,
- 100 000 LESs i 264 DSPs,
- przepustowość 52 Gb/s,
- interfejsy DDR3/PCI Express/Ethernet,
- 2 12-bitowe ADCs 17-kanałowe.

A-G Artix™ UltraScale+™ XA:

- obsługa kamer/LIDARów oraz
- Optymalizacja mocy,
- szybkie protokoły 16 Gb/s,
- zoptymalizowany moduł DSP,
- różne formy obliczeń i obróbki obrazu i wideo,
- funkcje cyberbezpieczeństwa oraz ochrony przed nieuprawnionym dostępem..

Automotive-Grade Spartan™ 7 XA:

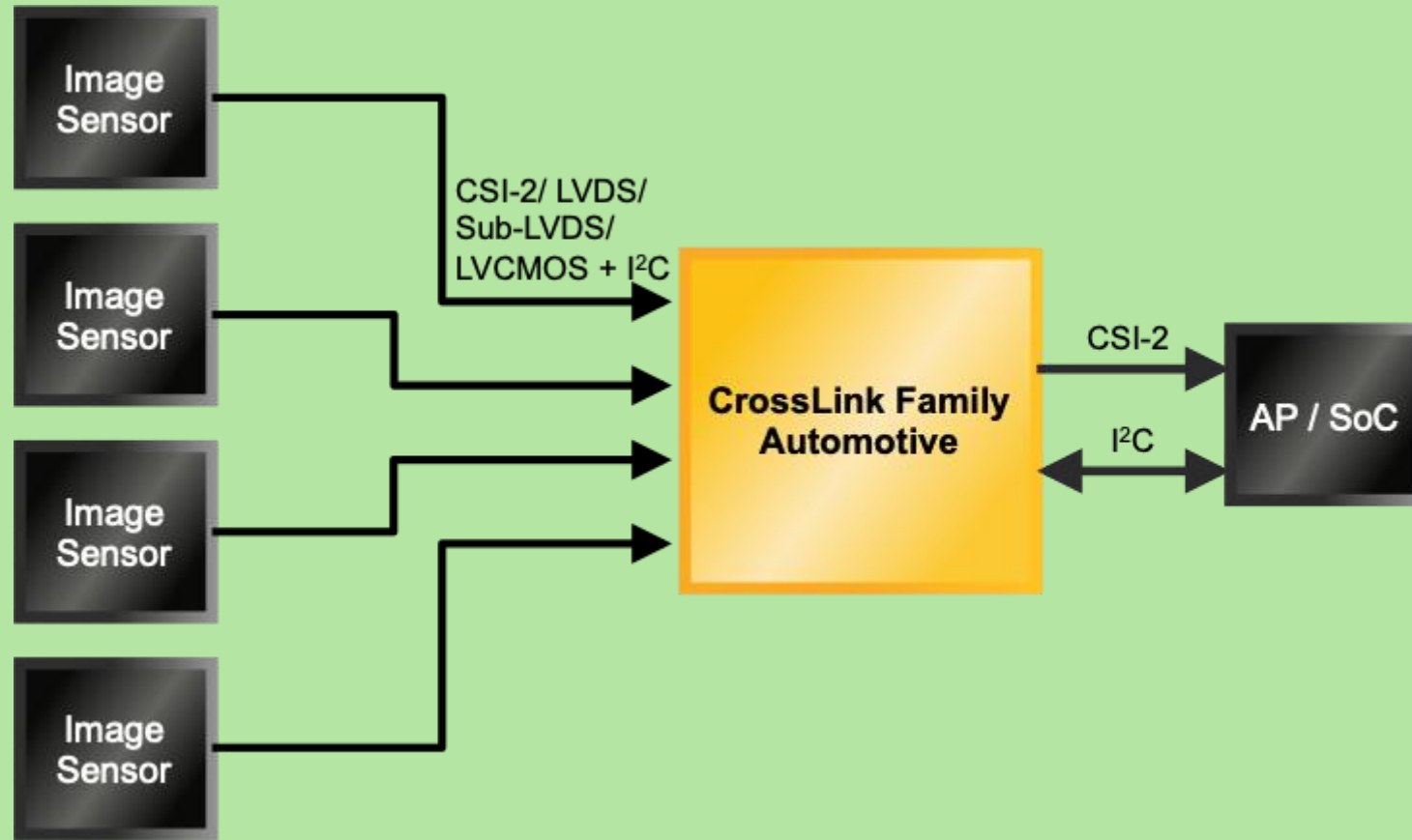
- softprocesor MicroBlaze V,
- zintegrowane zabezpieczenia,
- napięcie rdzenia 1,0 V lub 0,95 V,
- przepustowość DDR3-800 25,6 Gb/s,
- obsługa przez Vivado / WebPack,
- skalowalna architektura,
- łatwa integracja dyskretnych obwodów analogowych,
- optymalizacja kosztów wejść/wyjść.

Firma **Microchip** oferuje gamę produktów dla układów FPGA i SoC FPGA przeznaczonych dla motoryzacji. Są to energooszczędne, szybko uruchamiające się i niezawodne układy z certyfikatem AEC-Q100. Stosowane min. w sterowaniu falownikami i konwersji DC-DC w pojazdach elektrycznych i hybrydowych. Są one również wykorzystywane w inteligentnym systemie wizyjnym (SEV) dla ADAS do realizacji funkcji LiDAR/kamer.



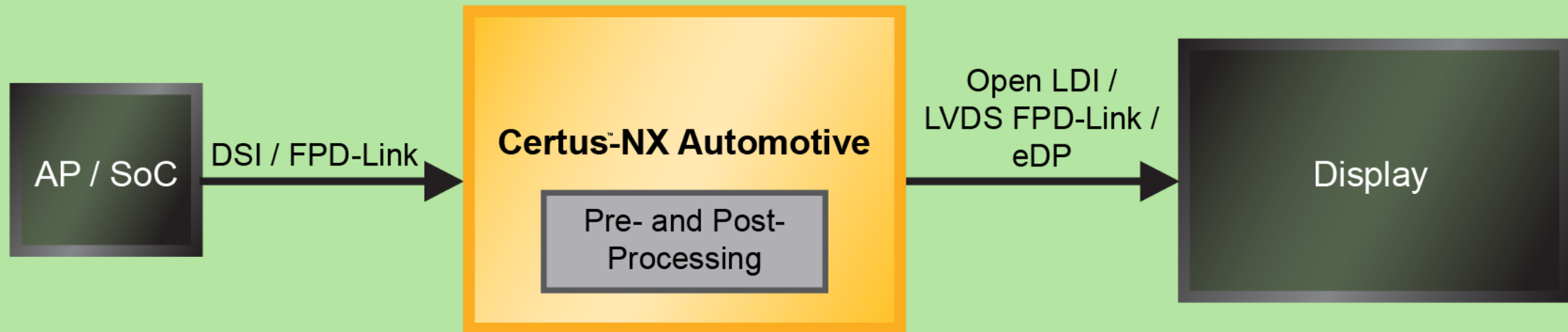
Features	PolarFire® SoC	PolarFire® FPGAs	IGLOO® 2 FPGAs	ProASIC® 3 FPGAs
Type	Mid-Range SoC FPGA	Mid-Range FPGA	Low-Density FPGA	CPLD Replacements
Logic Elements (LEs)	Up to 460K	100–480K	5–150K	100–30K
Transceiver Rate	250 Mbps–12.7 Gbps	250 Mbps–12.7 Gbps	1–5 Gbps	–
I/O Speeds	1600 Mbps LPDDR4 1.6 Gbps LVDS	1600 Mbps DDR4 1.6 Gbps LVDS	667 Mbps DDR3 750 Mbps LVDS	400 Mbps LVDS
Max RAM	33 Mb	33 Mb	5 Mb	144 Kb
Processor Options	Soft RISC-V Quad Core RISC-V Hard Crypto Process.	Soft RISC-V Soft Arm Cortex-M1 Hard Crypto Process.	Hard 166 MHz Arm Cortex-M3 Soft RISC-V	Soft 100 MHz Arm® Cortex®-M1

Firma **Lattice** oferuje szeroką gamę rozwiązań klasy motoryzacyjnej do różnych zastosowań w pojazdach. Aplikacje są zorientowane min na implementacje ADAS wykorzystujące układ FPGA CrossLink:



- łączenie wielu czujników obrazu CSI-2 w jedno wyjście MIPI CSI-2 do kamer 360 stopni,
- obsługa wejść kamer z interfejsów jak CSI-2, LVDS, Sub-LVDS i LVC MOS,
- obsługa wielu interfejsów kamer do połączenia z procesorem aplikacyjnym.

oraz aplikacje Infotainment z wykorzystaniem Certus-NX FPGA:



- komunikacja z wyświetlaczami MIPI DSI z kontrolą dla desek rozdzielczych,
- konwersja do/z MIPI DSI, Open-LDI/-LVDS i LVCMOS szybkiej łączności w samochodzie,
- wdrożenie funkcji PWM w celu sterowania oświetleniem i silnikiem,
- acentralizowany hub czujników do przetwarzania obrazu i wideo z powiadomieniami bezpieczeństwa.

oraz aplikacje Connectivity - HDMI/MHL Interface for Mobile devices:

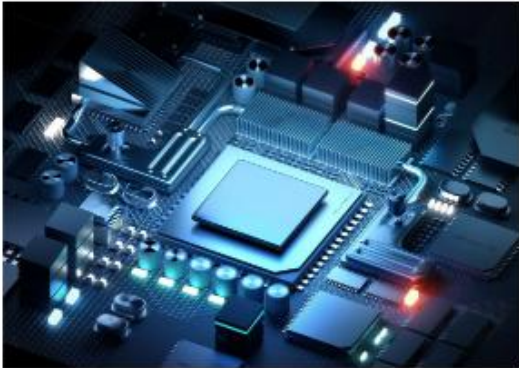


- łączenie łączności HDMI/MHL z interfejsem MIPI procesora samochodowego,
- łączność telefonu komórkowego z systemami informacyjno-rozrywkowymi,
- nieskompresowany mirroring ekranu telefonu komórkowego do celów rozrywkowych,
- dwukierunkowy kanał sterowania komunikacją między urządzeniem mobilnym a interfejsem samochodowym.

Problem stosowania FPGA w aplikacjach krytycznych.

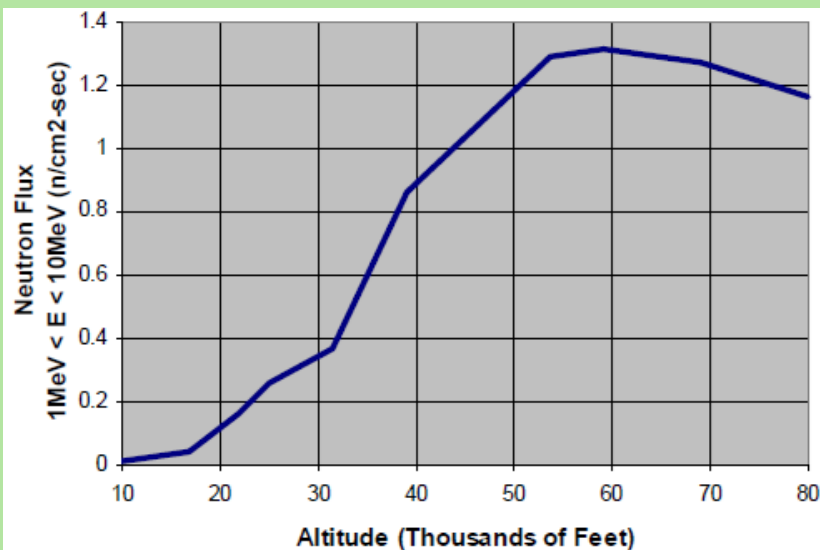
Actel Corporation były producentem nieulotnych macierzy bramek programowalnych i układów mieszanych. Firma opracowywała układy FPGA o znaczeniu krytycznym rodziny RTAX i ProASIC3, w tym układy FPGA o rozszerzonym zakresie temperatur dla motoryzacji, wojska i lotnictwa, a także szeroką gamę urządzeń odpornych na promieniowanie klasy kosmicznej. Układy FPGA typu flash i anti-fuse charakteryzują się wysokim poziomem niezawodności i odpornością na błędy stałe.

W 2010 r. firma Actel została przejęta przez Microsemi, a w 2018 r. firma Microchip przejęła Microsemi. Obecnie Microchip oferuje układy FPGA firmy Actel:

			
FPGAs	SoC FPGAs	Radiation-Tolerant FPGAs	FPGA Design Tools
• PolarFire® FPGAs • IGLOO® FPGAs	• PolarFire SoC FPGAs • SmartFusion® 2 FPGAs	• Radiation-tolerant PolarFire SoC FPGAs	• Libero® IDE • Libero SoC Design Suite

Układy FPGA są wrażliwe na promieniowanie kosmiczne. Elementy pamięci SRAM podatne na działanie neutronów są wykorzystywane do konfiguracji w układach FPGA. Stąd istnieje ryzyko awarii funkcjonalnej w układach FPGA. Wykrywanie i korygowanie tych zakłóceń konfiguracji nie zawsze następuje natychmiastowo. W rzeczywistości może upłynąć kilka tysięcy lub milionów cykli zegara, zanim wykryta zostanie awaria funkcjonalna. Ponadto schematy wykrywania/korygowania błędów zwiększają złożoność projektu i koszty.

Cząstki o wysokiej energii z przestrzeni kosmicznej i Słońca zderzają się z atomami azotu i tlenu w górnych warstwach atmosfery ziemskiej. Zderzenia te powodują zniszczenie atomów oraz powstanie wielu innych cząstek o wysokiej energii. Znaczna część produktów zderzeń atmosferycznych to neutrony. Neutrony te są emitowane z bardzo dużą częstotliwością i nie ulegają rekombinacji z innymi cząstkami, ponieważ nie są naładowane. Poruszają się z dużą prędkością, aż do zderzenia z gazami atmosferycznymi, obiektami na powierzchni Ziemi lub obiektami przemieszczającymi się przez atmosferę. Strumień neutronów zależy od wysokości - max 60 000 stóp.



Military and Avionics Flight Heritage: *Examples* Actel

Tactical Missiles

AIM 9X Sidewinder
Hellfire
Brimstone
Patriot
PAC-3
GBI
THAAD
MLRS
Longbow
Harpoon



Commercial Aircraft

737
747
757
767
777
Dash-8 400
Airbus 319, 320, 321, 340, 380



Military Aircraft

F-22
KC-135
Apache
F-117
CH-60
B1B
Lamps
Nimrod
E2C
F-16
Comanche
JSF
F-14
A-10
F-15
F-18
C-130



Military Ground Vehicles

Bradley Fighting Vehicle
M1 Abrams

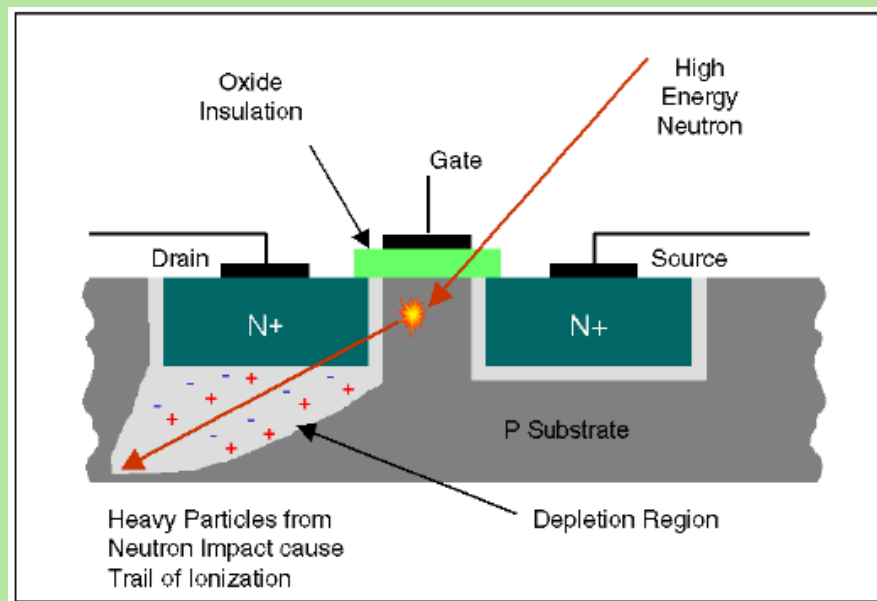


Military Systems

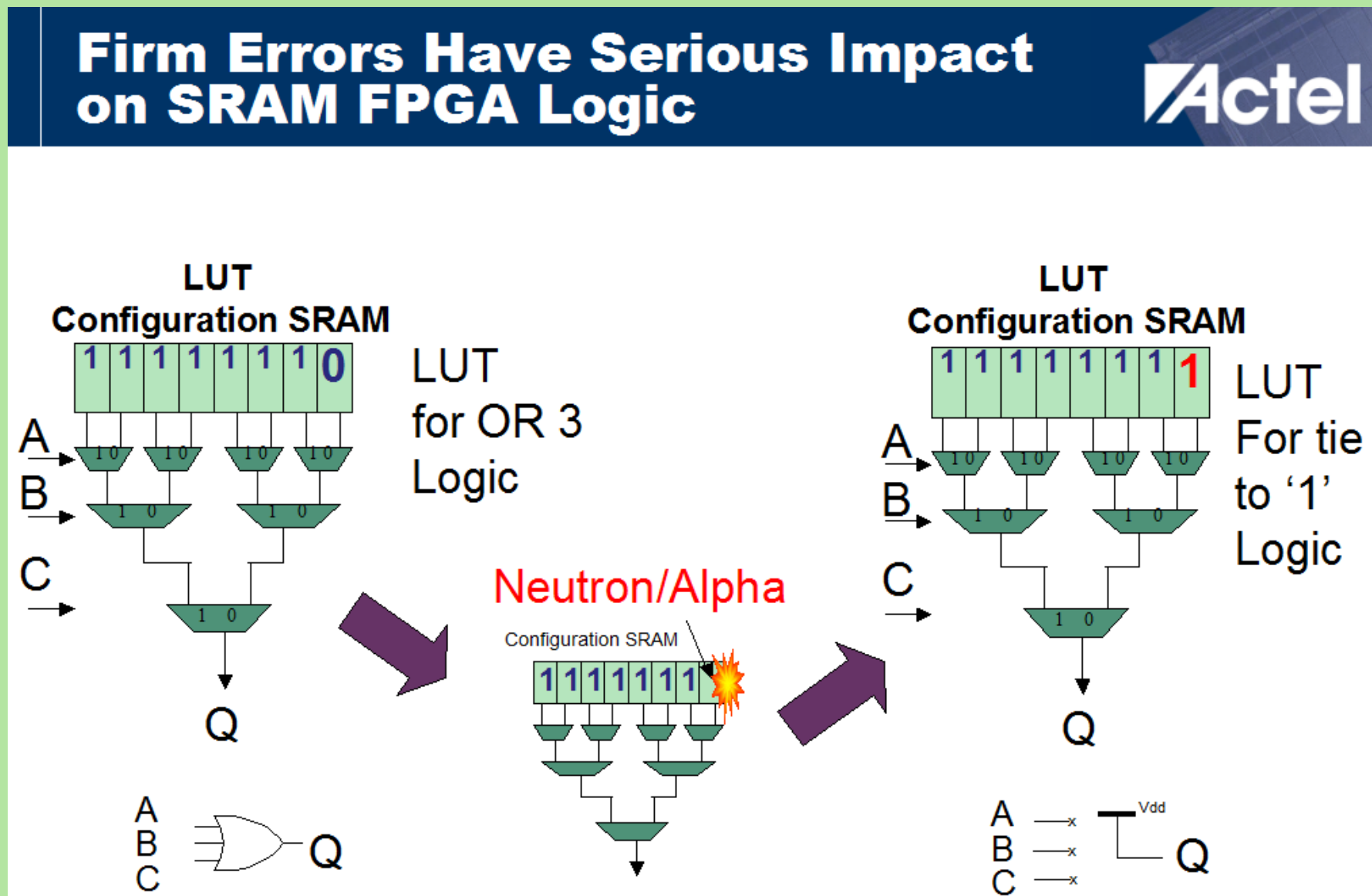
ATACMS
TACMS
TADS/PNVS
WCM
Lantim
IFCS
AITG
ESAF
ALQ135
AAQ24
ATIRCM
IDECM



Większość neutronów kosmicznych uderzających w układ scalony przechodzi przez niego bez oddziaływania z nim - ale czasami neutron przechodzi blisko atomu krzemu lub domieszki i zakłóca jego strukturę. Wynikiem tego oddziaływania jest powstanie cząstek wtórnych. Rysunek pokazuje tę interakcję. Jeśli dojdzie do tego w pobliżu złącza p-n z odwróconym napięciem w komórce pamięci lub przerzutniku, może wystąpić skok napięcia, powodujący zmianę stanu komórki pamięci lub przerzutnika. Gdy zawartość elementu pamięciowego ulega zmianie bez zakłócenia jego, taki stan nazywa się błędem miękkim. W takim przypadku urządzenie można pomyślnie nadpisać prawidłowymi danymi. Zmiana stanu komórki pamięciowej przechowującej informacje o konfiguracji FPGA może spowodować urządzenia lub systemu. W ten sposób zagrożenie związane z bombardowaniem neutronowym nie ogranicza się tylko do błędów miękkich. Uszkodzenie komórki konfiguracyjnej prowadzi do błędu trwałego – co może ostatecznie uszkodzić urządzenie powodując błąd twardy.



Błędy miękkie mogą występować również w obszarach tablic LUT makrokomórek – zakłócając pracę układu:



Problem wpływu promieniowania kosmicznego na pracę FPGA redukują dwie metody:

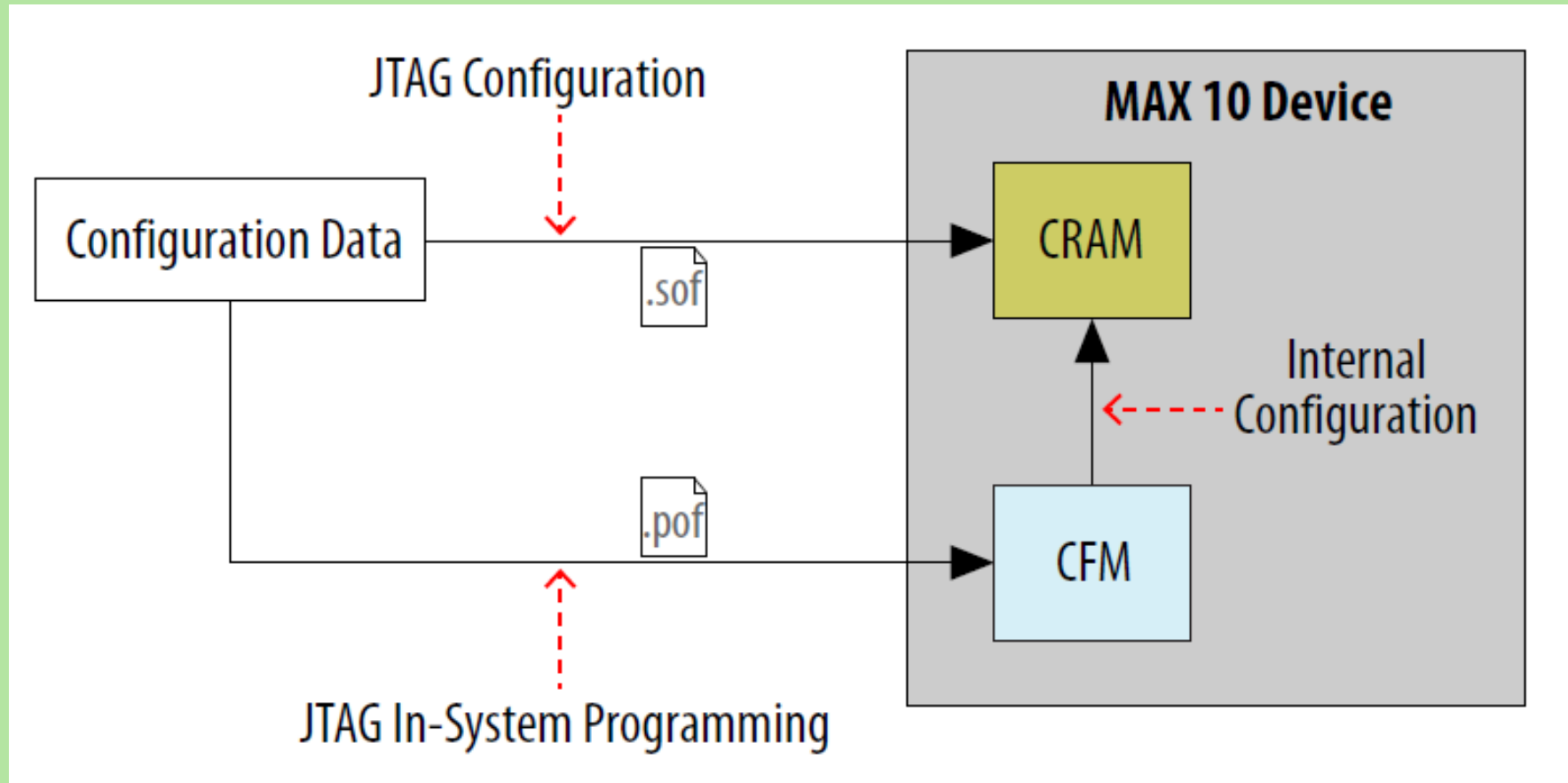
Jedną jest zwielokrotnienie układu – na płycie PCB montuje się trzy identyczne układy tego samego projektu. System jest uzupełniany obwodem większościowym, aby zapewnić wykrycie danych uszkodzonych przez pojedyncze zdarzenia zakłócające i podjąć decyzję o sygnałach wyjściowych przez dwa pozostałe układy.

Alternatywą jest zastosowanie technologii FPGA opartej konfiguracyjnie na antifuse. Układy FPGA Actel oparte na antifuse i Flash nie są podatne na utratę konfiguracji spowodowaną zakłóceniami wywołanymi przez neutrony.

Przykład 1: Gęstość strumienia neutronów zmierzono na wysokości 5000 stóp i na tej podstawie obliczono wskaźnik awaryjności FPGA $0,22 \mu$ na poziomie $1,05E-4$ awarii na układ FPGA o pojemności 1 mln bramek przeliczeniowych w ciągu 1 dnia. Na przykład przełącznik telekomunikacyjny może mieć 32 karty liniowe z 2 układami FPGA o pojemności 1 mln bramek na kartę, co daje łącznie 64 układy FPGA. Stąd po obliczeniach otrzymujemy $6,73E-3$ zakłóceń na dzień – czyli 3580 godzin między zakłóceniami systemu.

Przykład 2: Mamy płytkę drukowaną w systemie awioniki na pokładzie samolotu wojskowego lecącego 40 stopni na północ od równika na wysokości 60 000 stóp. Strumień neutronów w tych warunkach jest ponad czterokrotnie wyższy niż w cywilnych samolotach i powoduje to częstotliwość zakłóceń wynoszącą $8,33E-2$ zakłóceń dziennie dla układu FPGA $0,13\mu$ o pojemności 1 mln bramek przeliczeniowych. Zakładając, że płyta awioniki wykorzystuje cztery układy FPGA – daje to 72 godziny między zakłóceniami. Jeśli system awioniki wykorzystuje cztery płyty, czas między awariami wynosi 18 godzin.

Nie zawsze zachodzi potrzeba stosowania dużych układów FPGA – alternatywą pozostają CPLD-FPGA

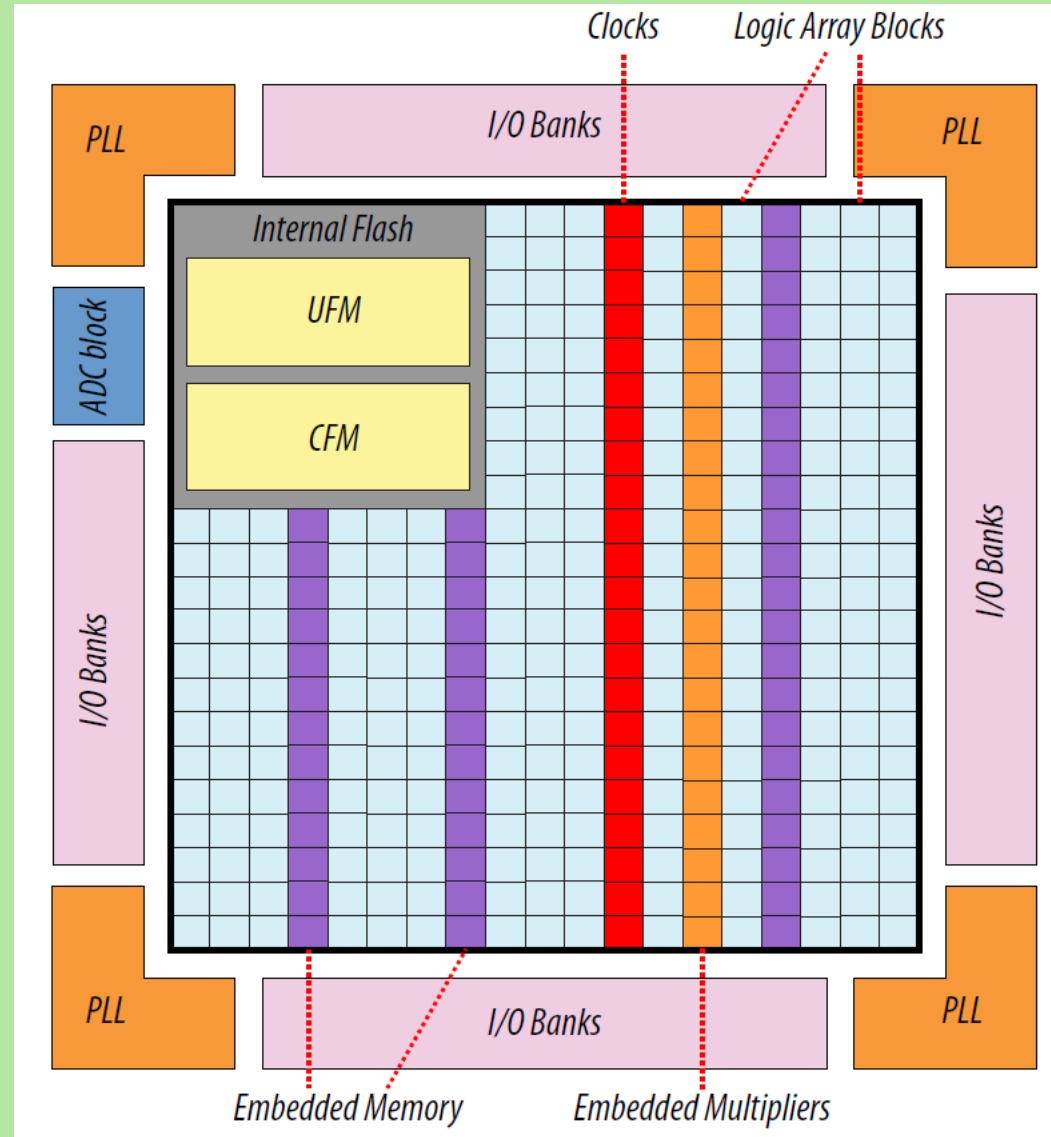


Np. układ Max 10 Intel:



Max 10 zawiera:

- klasyczne bloki LAB,
- przetwornik A/C,
- pamięć programująca Flash,
- pamięć Flash użytkownika,
- wbudowane mnożniki,
- wbudowane bloki M9K,
- PLL,
- standardowe i szybkie I/O,
- sterowniki pamięci zewnętrznych.



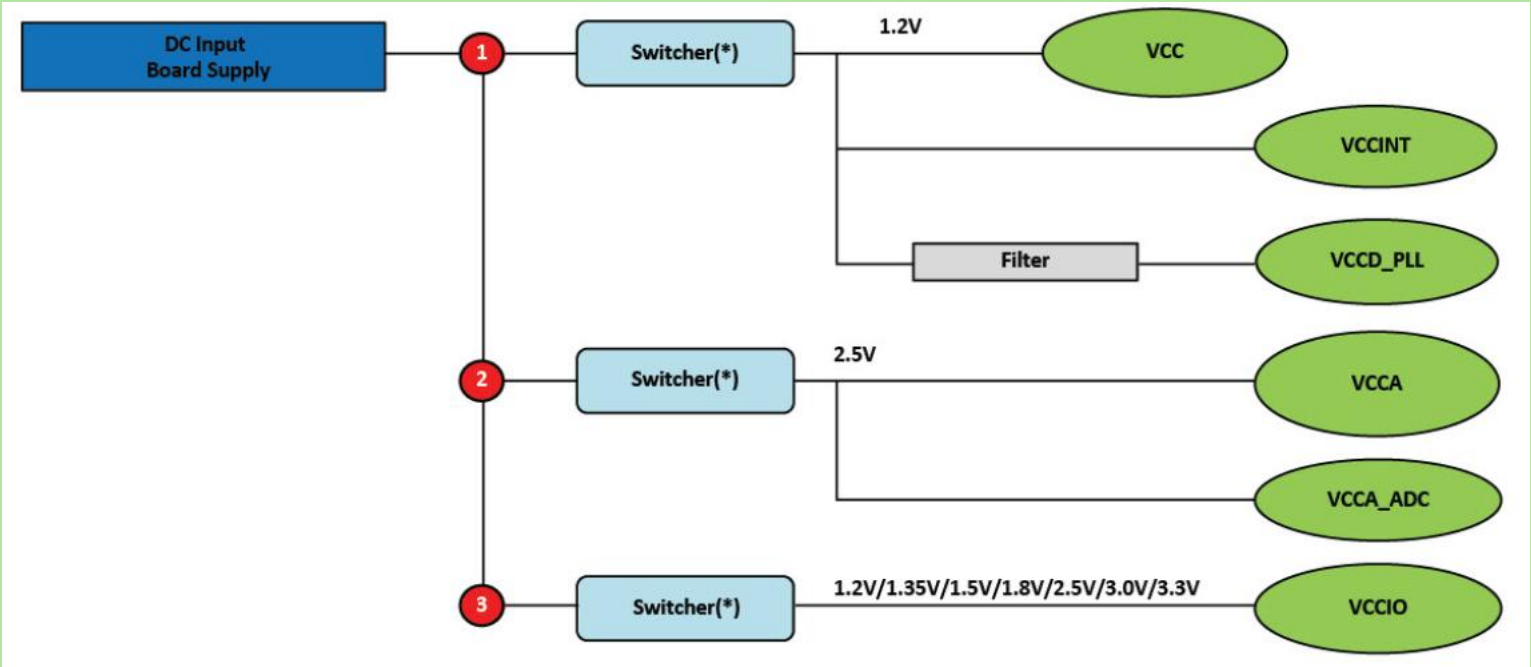
Parametry Max 10:

Resource		Device						
		10M02	10M04	10M08	10M16	10M25	10M40	10M50
Logic Elements (LE) (K)		2	4	8	16	25	40	50
M9K Memory (Kb)		108	189	378	549	675	1,260	1,638
User Flash Memory (Kb) ⁽¹⁾		96	1,248	1,376	2,368	3,200	5,888	5,888
18 × 18 Multiplier		16	20	24	45	55	125	144
PLL		2	2	2	4	4	4	4
GPIO		160	246	250	320	360	500	500
LVDS	Dedicated Transmitter	9	15	15	22	24	30	30
	Emulated Transmitter	73	114	116	151	171	241	241
	Dedicated Receiver	73	114	116	151	171	241	241
Internal Configuration Image		1	2	2	2	2	2	2
ADC		—	1	1	1	2	2	2

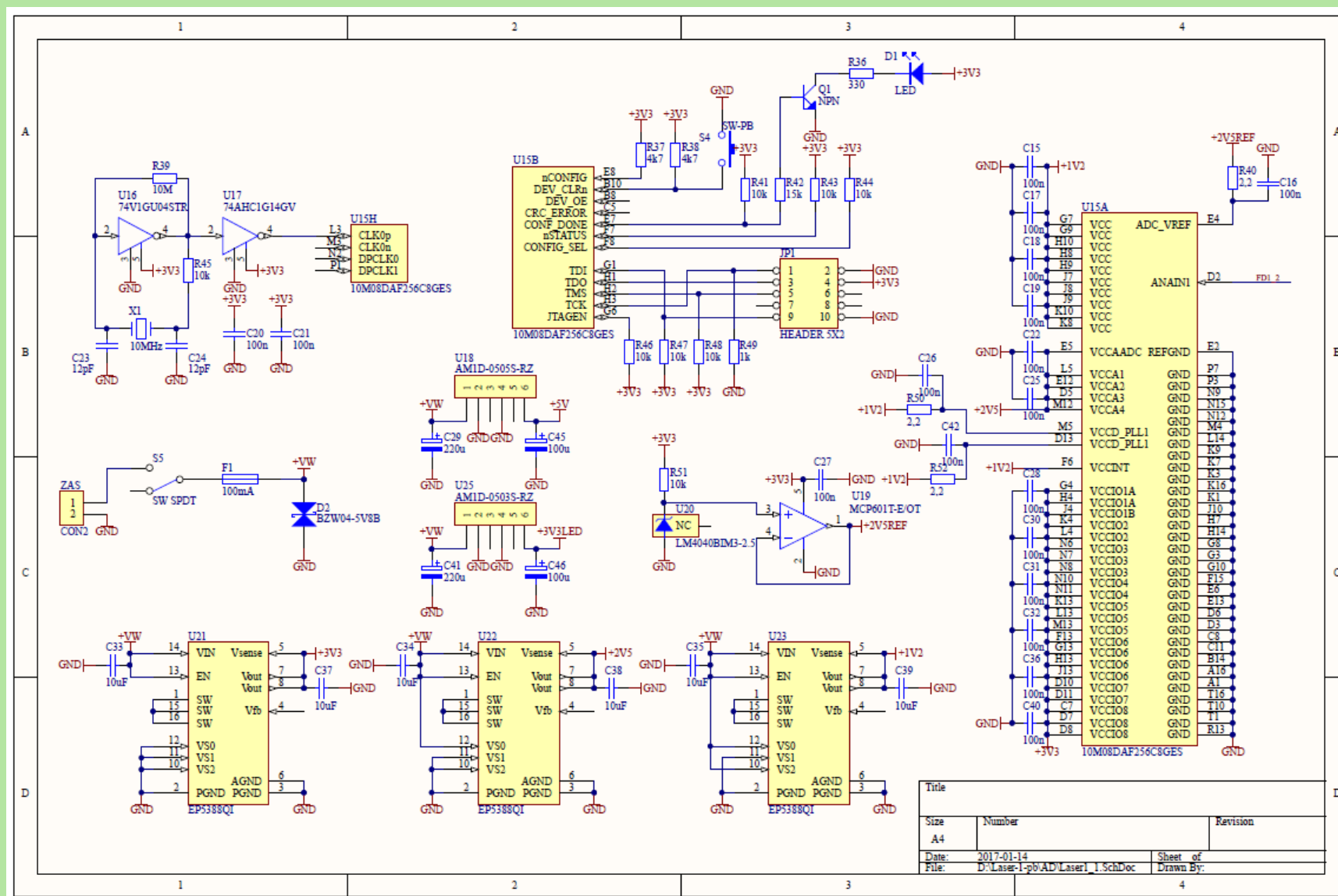
Układ FPGA, zwłaszcza duży, wymaga kilku napięć zasilających:

Example requiring 3 power regulators

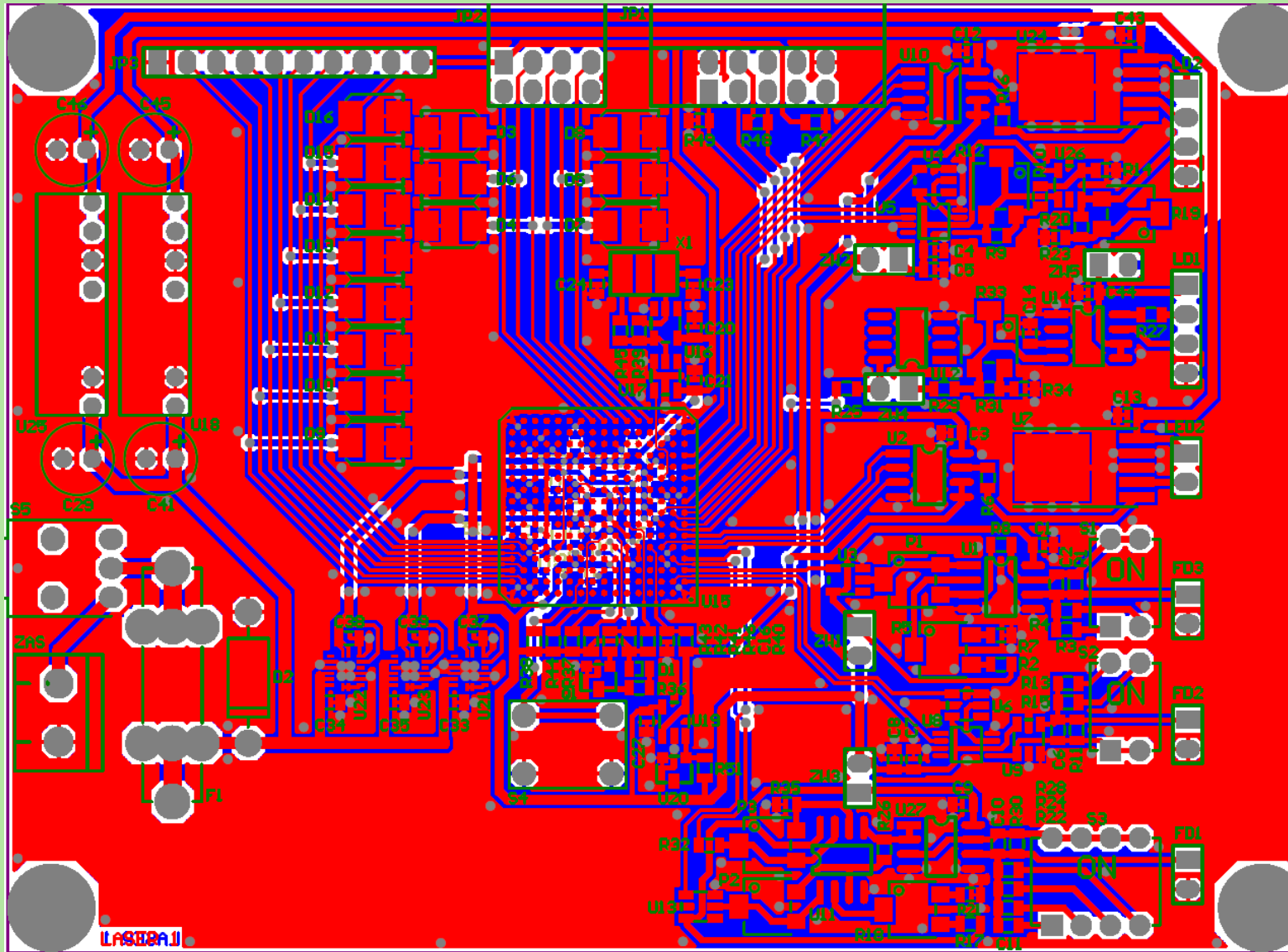
Power Pin Name	Regulator Count	Voltage Level (V)	Supply Tolerance	Power Source	Regulator Sharing	Notes
VCC	1	1.2	± 50mV	Switcher (*)	Share	You have the option to share VCCINT and VCCD_PLL with VCC using proper isolation filters.
VCCINT					Isolate	
VCCD_PLL					Isolate	
VCCA	2	2.5	± 5%	Switcher (*)	Share	You have the option to share VCCA_ADC with VCCA.
VCCA_ADC		2.5				
VCCIO	3	Varies	± 5%	Switcher (*)	Share	Individual power rail.



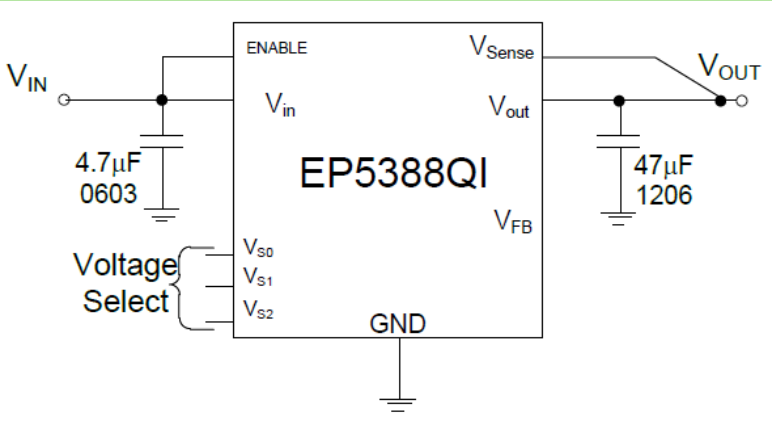
Przykładowy schemat z układem Max 10:



W takim przypadku płytka PCB jest wielowarstwowa – 6-warstwowa lub więcej:



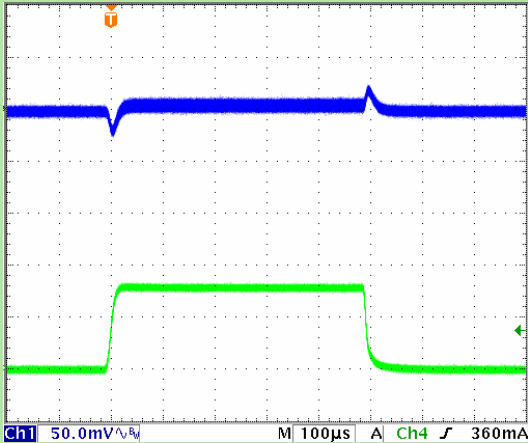
Do wytwarzania napięć zasilających zaleca się stosowanie dedykowanych układów:



VS2	VS1	VS0	V _{OUT}
0	0	0	3.3V
0	0	1	2.5V
0	1	0	1.8V
0	1	1	1.5V
1	0	0	1.25V
1	0	1	1.2V
1	1	0	0.8V
1	1	1	User Selectable

FEATURES

- 3mm x 3mm x 1.1mm QFN package
- Only two low cost MLCC caps required
- 4 MHz switching frequency
- High efficiency, up to 94%
- Up to 800mA continuous output current
- Wide 2.4V to 5.5V input range
- V_{OUT} Range: 0.6V to V_{IN} – 0.5V
- 3-Pin VID output voltage programming
- 100% duty cycle capable
- Less than 1 µA standby current
- Low V_{OUT} ripple for RF compatibility
- Short circuit and over current protection
- UVLO and thermal protection
- RoHS compliant; MSL 3 260°C reflow

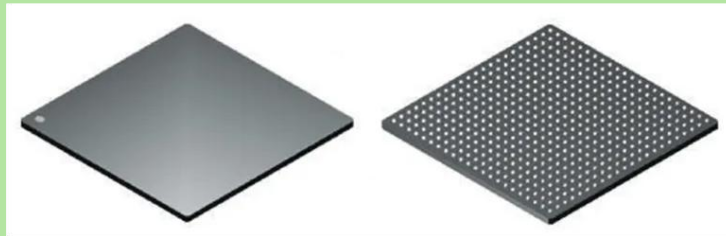


Równie ostrożnie należy podchodzić do projektowania płytki PCB pod układy FPGA:

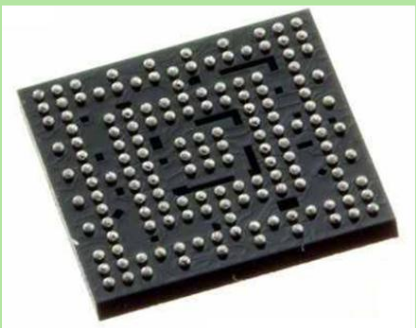
TQFP



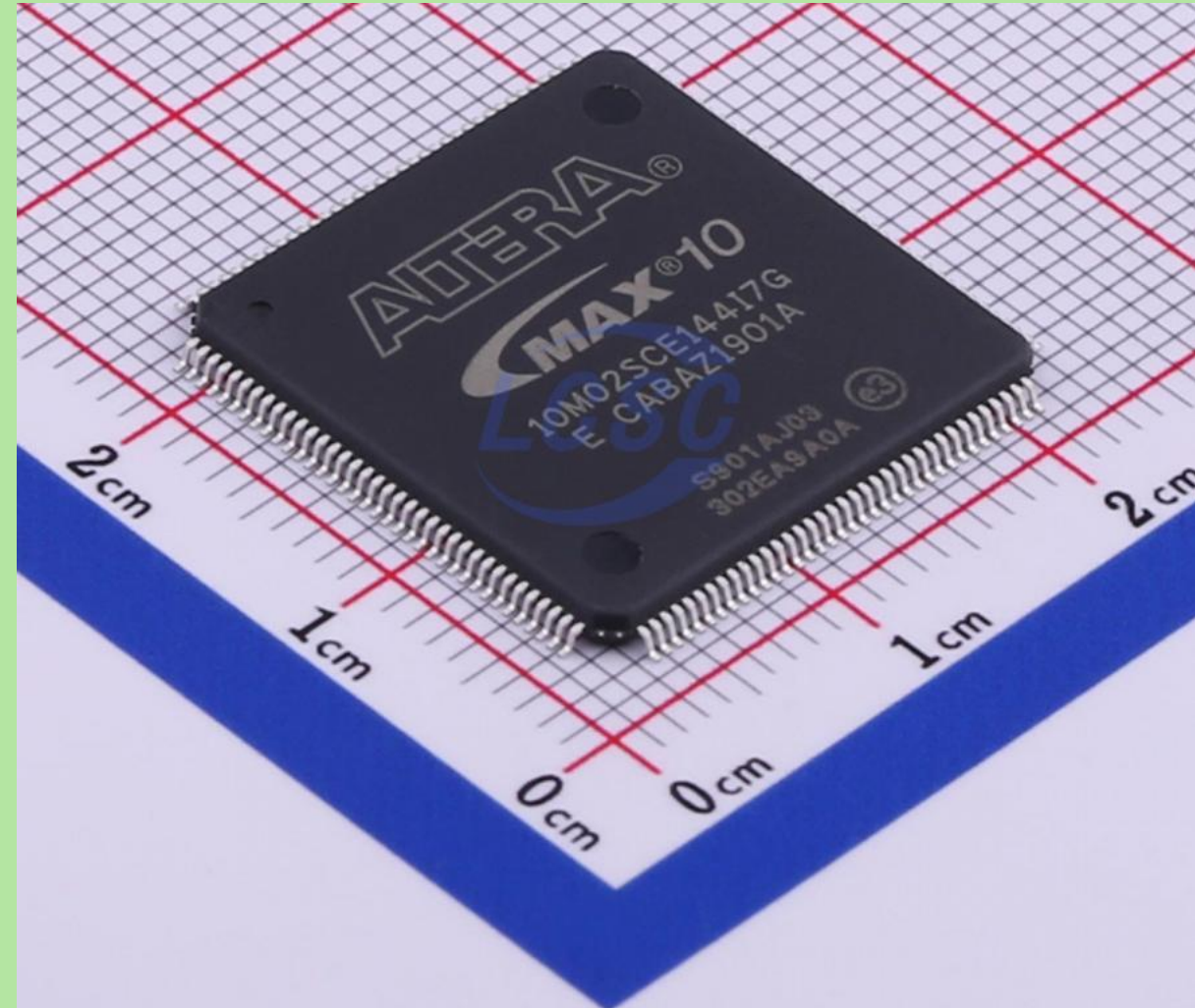
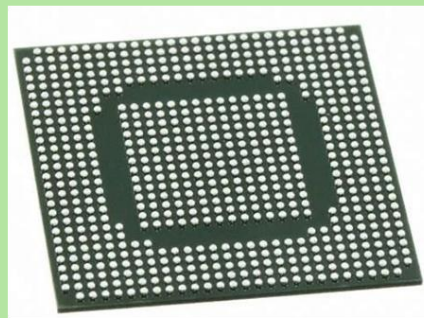
FBGA



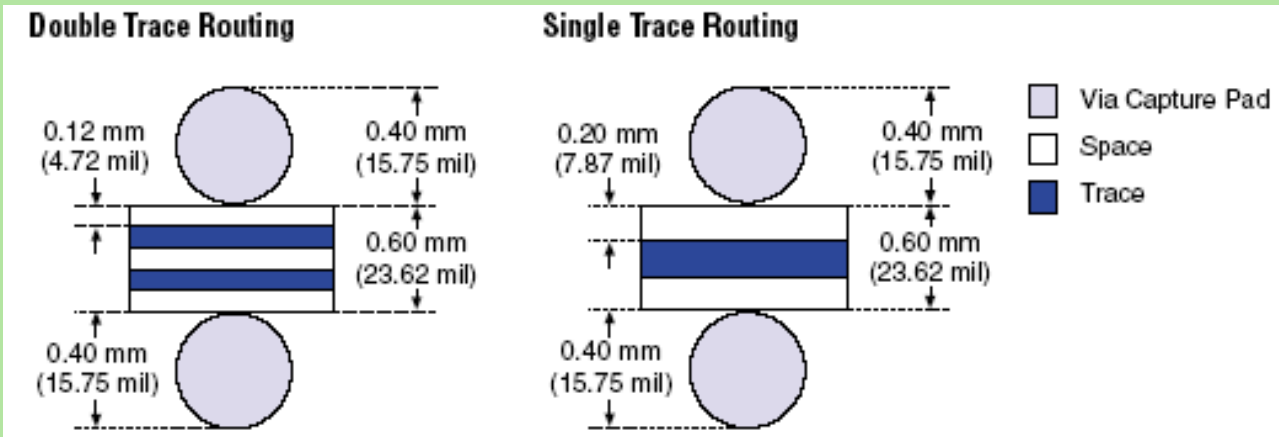
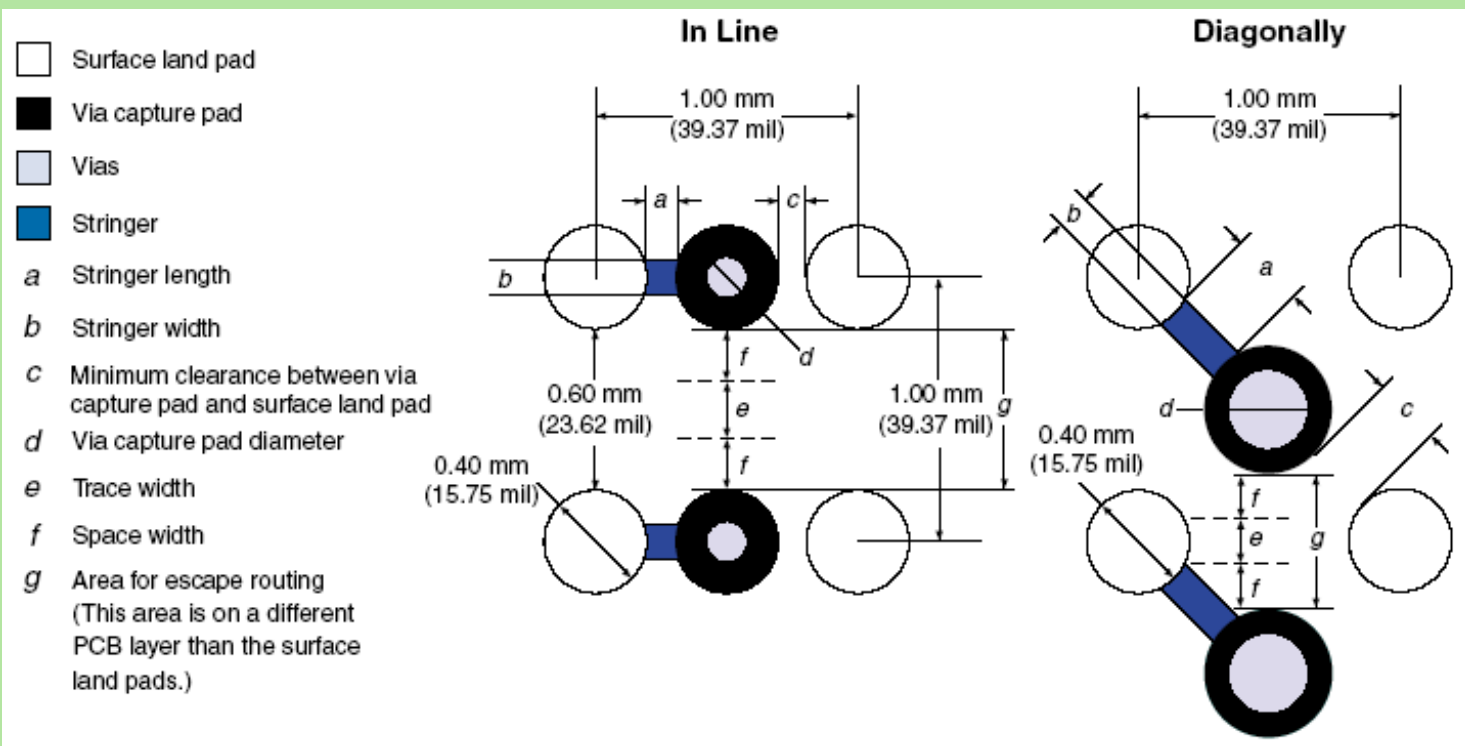
MBGA



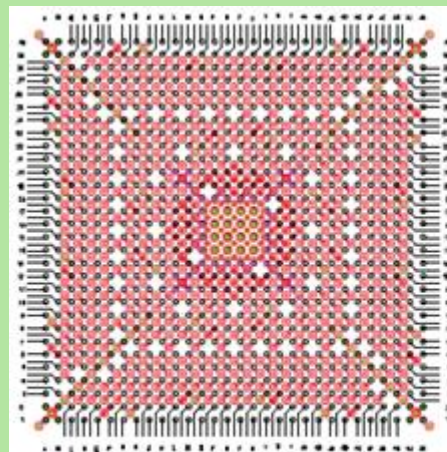
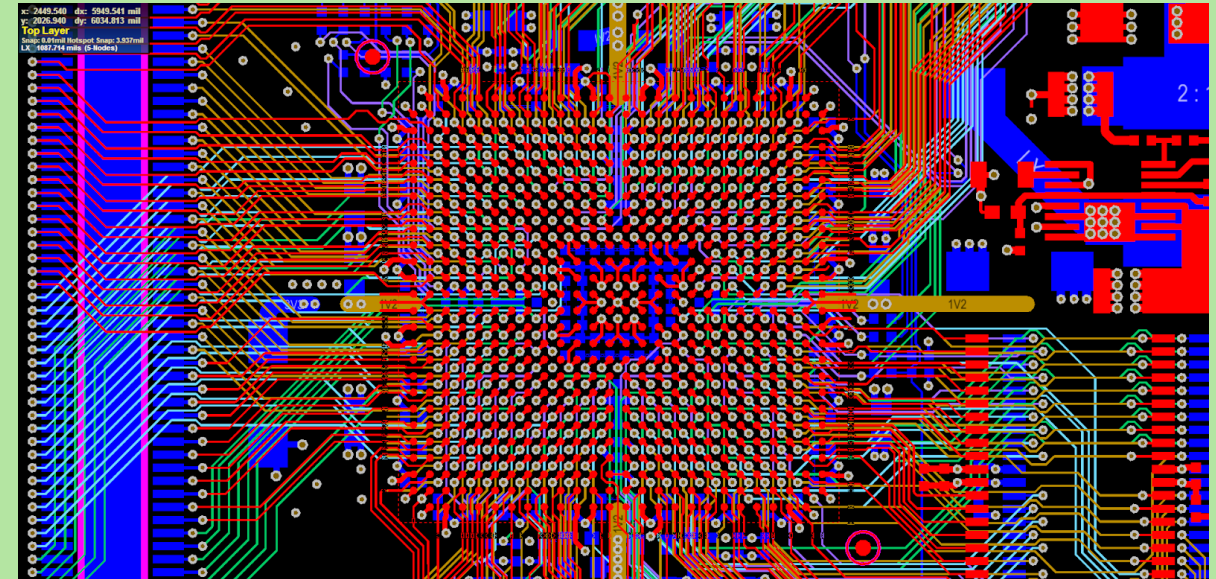
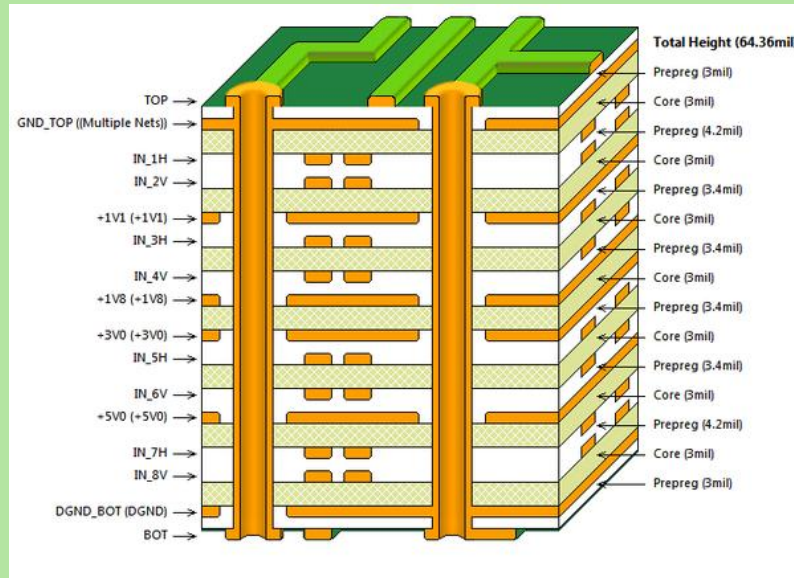
uBGA



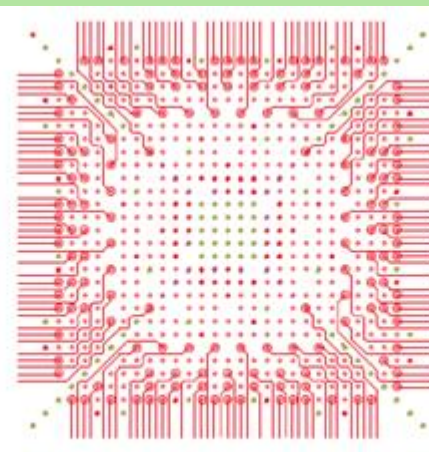
Chociaż w danych technicznych układów można znaleźć zalecenia projektowe PCB:



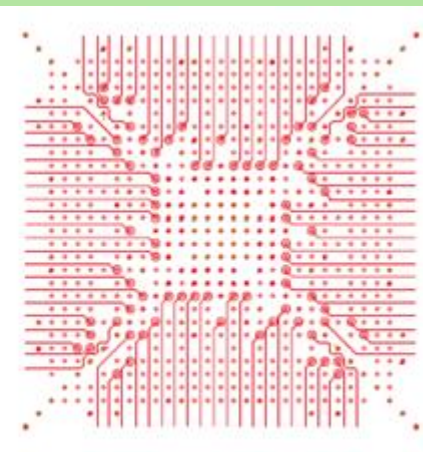
Oraz wzory masek dla poszczególnych warstw ... to samobójcy za \$1000 z żelazkiem są mile widziani:



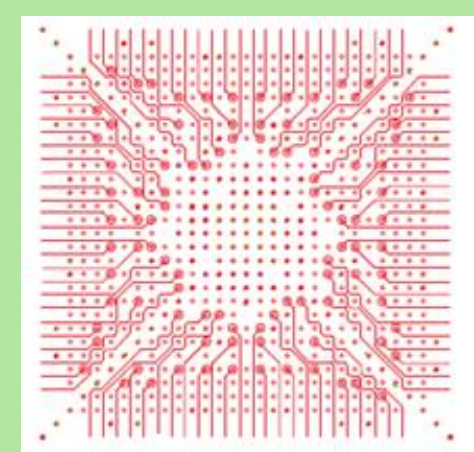
Top Signal Layer



Second Signal Layer



Third Signal Layer



Sixth Signal Layer

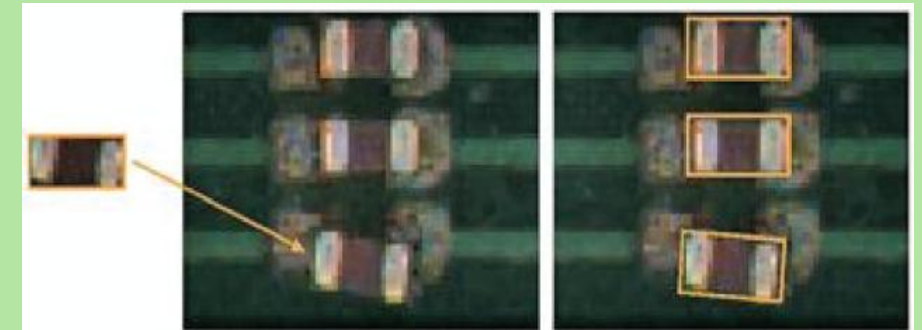
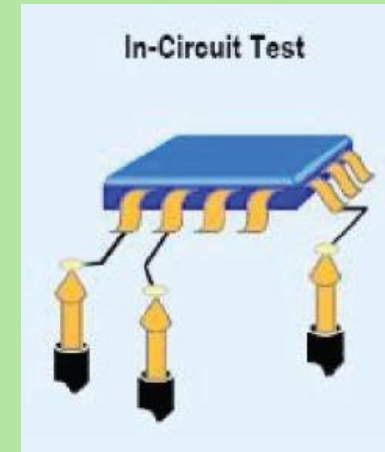
Jeżeli po zmontowaniu układ nie działa, to w jaki sposób ustalić kto popełnił błąd: chiński wykonawca i montażysta płytki czy polski projektant PCB? Czym jest system XJTAG?



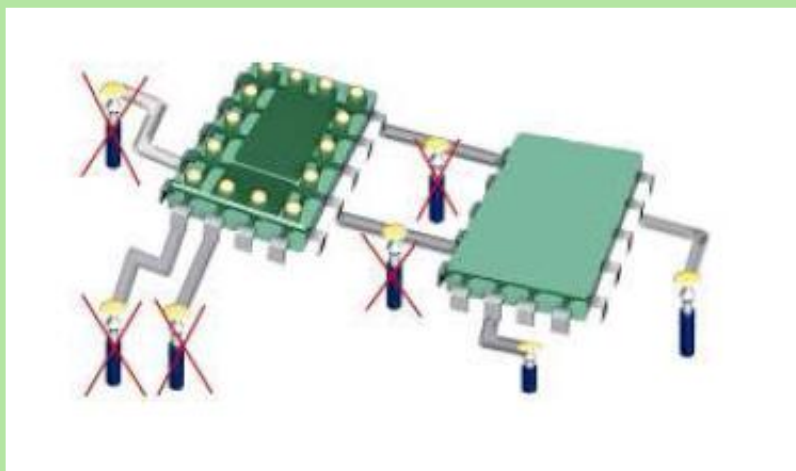
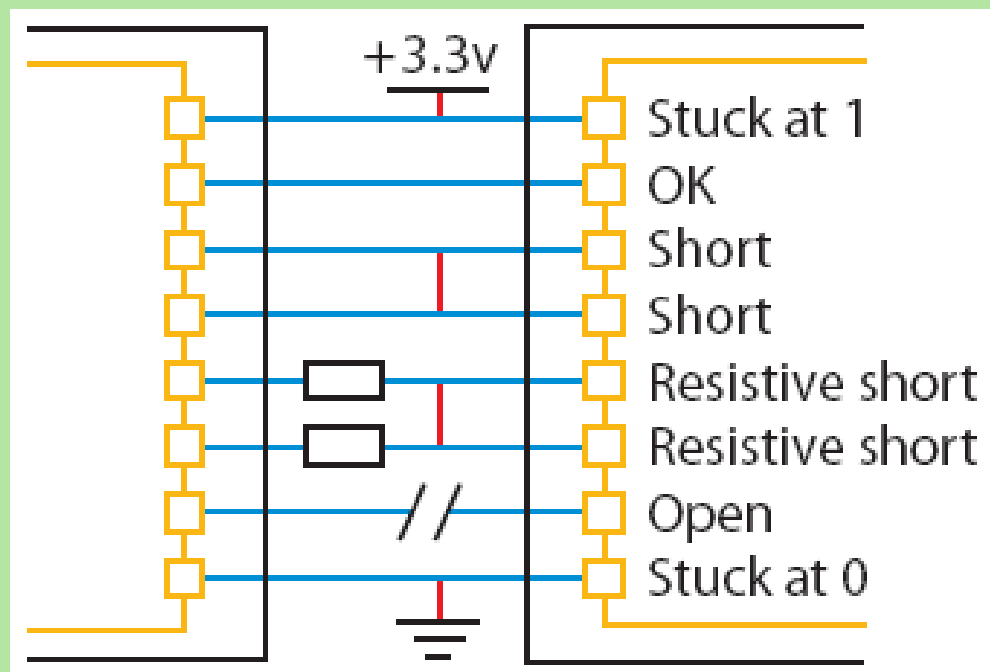
Konstrukcja zaawansowanych systemów elektronicznych wymusiła rozwój technik testowania:

Klasyfikacja stosowanych technik:

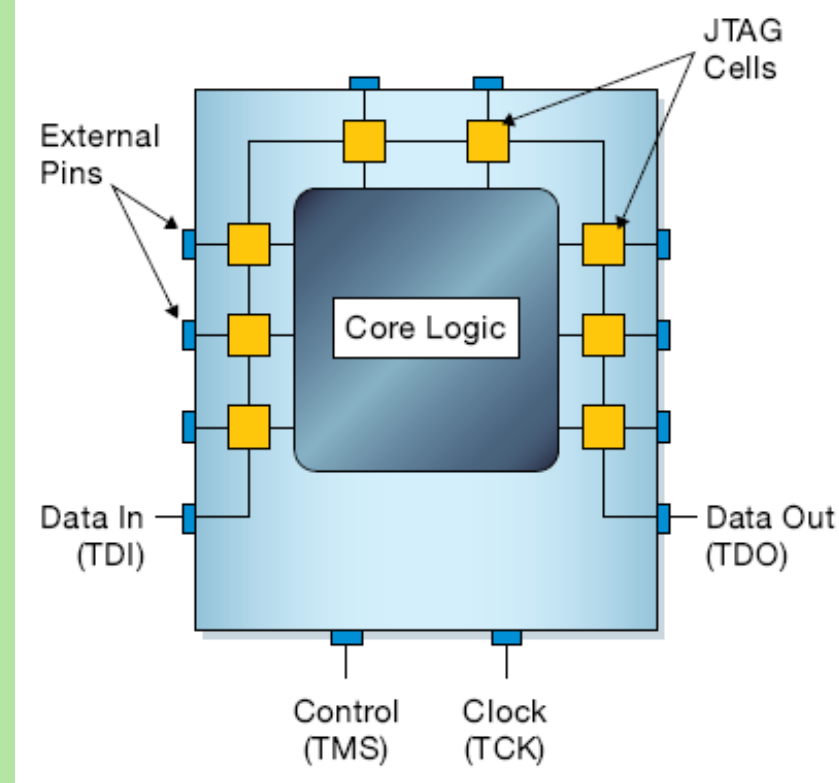
- kontaktowe:
 - ICT (in-circuit-test) - skomplikowana, droga z długim cyklem,
 - matryca szpilek dociskana do wyprowadzeń układów na płytce,
 - do tej grupy zalicza się metodę z użyciem generatora i oscyloskopu,
- bezdotykowej inspekcji optycznej:
 - AOI (Automated Optical Inspection) – wykrywa jakość połączeń,
 - zdjęcia wykonywane w specjalny sposób,
 - analiza zdjęć manualna lub automatyczna,
 - drogie, skomplikowane programy analizujące
- wbudowane:
 - kod źródłowy wykonywany przez mikrokontroler,
 - metoda tania i szybka,
 - metoda nie sprawdza linii sygnałowych.



Jakie uszkodzenia wykrywa XJTAG:



System opiera się na koncepcji ścieżki krawędziowej JTAG Boundary Scan



- wprowadzono pomiędzy rdzeń a wyprowadzenia makrokomórki BSC połączonych w łańcuch,
- dane testujące są szeregowo wprowadzane oraz szeregowo wyprowadzanie odpowiedzi,
- opracował zespół Joint Test Action Group (USA, Europa), technika stała się od 1990 r. standardem IEEE,
- pozwala ona na sprawdzanie połączeń, programowanie i uruchamianie,
- po roku 2000 nastąpił rozwój i coraz szersze zastosowanie tej techniki.

XJAnalyser jest:



- jest narzędziem do analizy/debugowania urządzeń w łańcuchu JTAG.
- prosta konfiguracja umożliwia natychmiastową weryfikację łańcucha,
- można błyskawicznie generować sygnały śledząc przerwy i zwarcia,

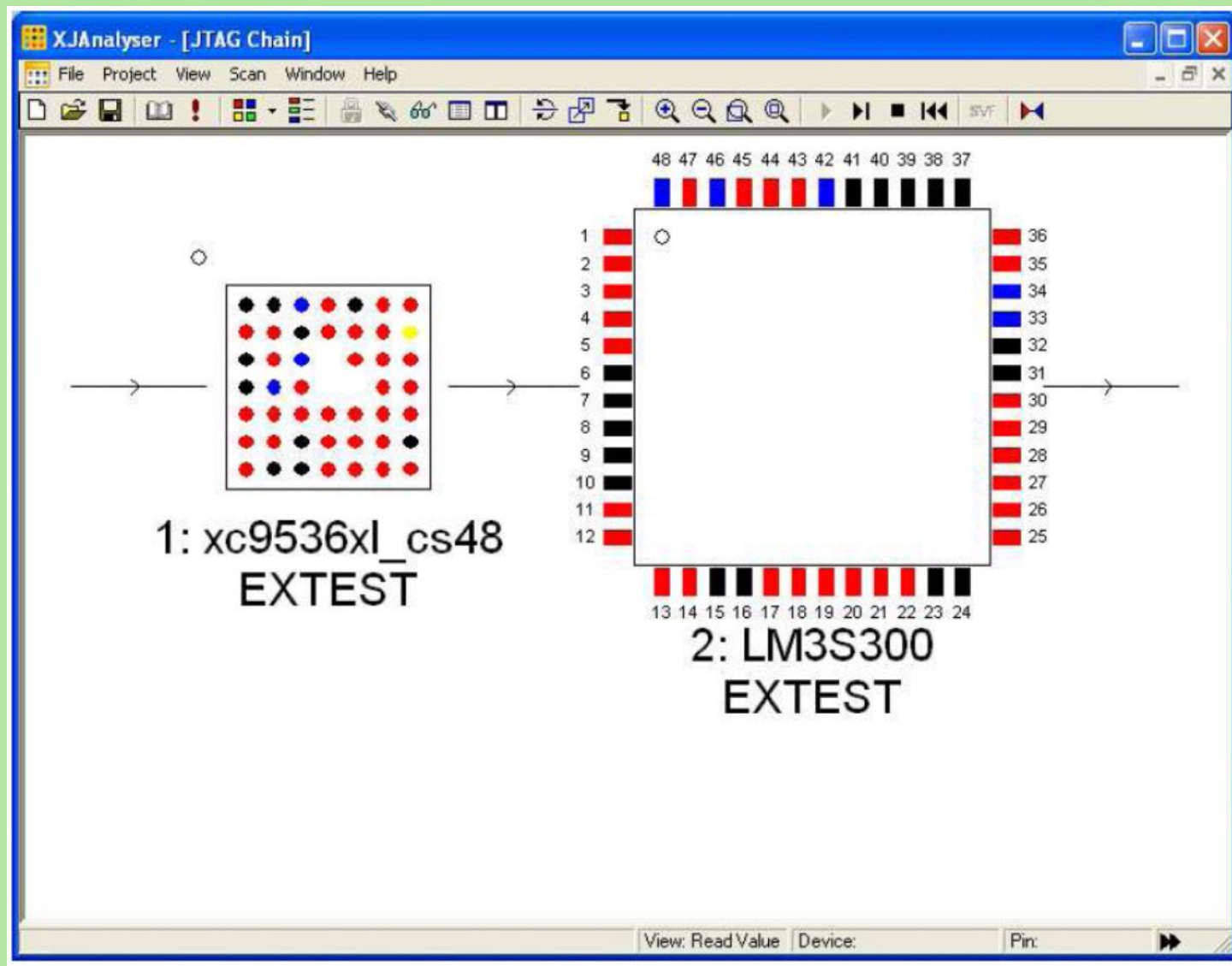
Kluczowe korzyści:

- sterowanie wartościami pinów oraz śledzenie sygnałów,
- skrócenie czasu tworzenia projektu od momentu prototypu,
- oszczędność budżetowa – eliminacja innych narzędzi, .

Właściwości

- konfiguracja typu Plug and Play i testowanie układów z obudową FBGA,
- debugowanie układu z użyciem jedynie plików BSDL,
- zmiana stanu pinów na wysoki, niski lub cykliczny,
- łączenie pinów w szyny,
- przejrzyste wyświetlanie pinów, ich numerów, nazw oraz stanów,
- szybkie wyszukiwanie pinów i śledzenie wartości w czasie rzeczywistym,
- możliwość programowania układów plikami SVF i STAPL.

XJAnalyser pozwala wykrywać:



XJAnalyser obsługa:

XJAnalyser - [Project1] - [JTAG Chain]

File Project View Scan Window Help

Please select a device: 1, xc9536xl_cs48

1: xc9536xl_cs48
EXTEST

2: EPM3032AT44
EXTEST

Nu...	Na...	Type	Re...	Wri...	Out...	C
A1	TCK	Lin...	N/A	N/A	N/A	N
A2	TMS	Lin...	N/A	N/A	N/A	N
A3	PB...	In...	0	N/A	No	N
A4	PB...	In...	0	N/A	No	N
A5	Vss...	Lin...	N/A	N/A	N/A	N
A6	PB...	In...	0	N/A	No	N
A7	PB...	In...	1	N/A	No	N
B1	PB...	In...	0	N/A	No	N
B2	PB...	In...	0	N/A	No	N
B3	TDI	Lin...	N/A	N/A	N/A	N
B4	PB...	In...	0	N/A	No	N
B5	PB...	In...	0	N/A	No	N
B6	PB...	In...	0s...	N/A	No	N
B7	PB...	In...	0	N/A	No	N
C1	Vc...	Lin...	N/A	N/A	N/A	N
C2	PB...	In...	0	N/A	No	N

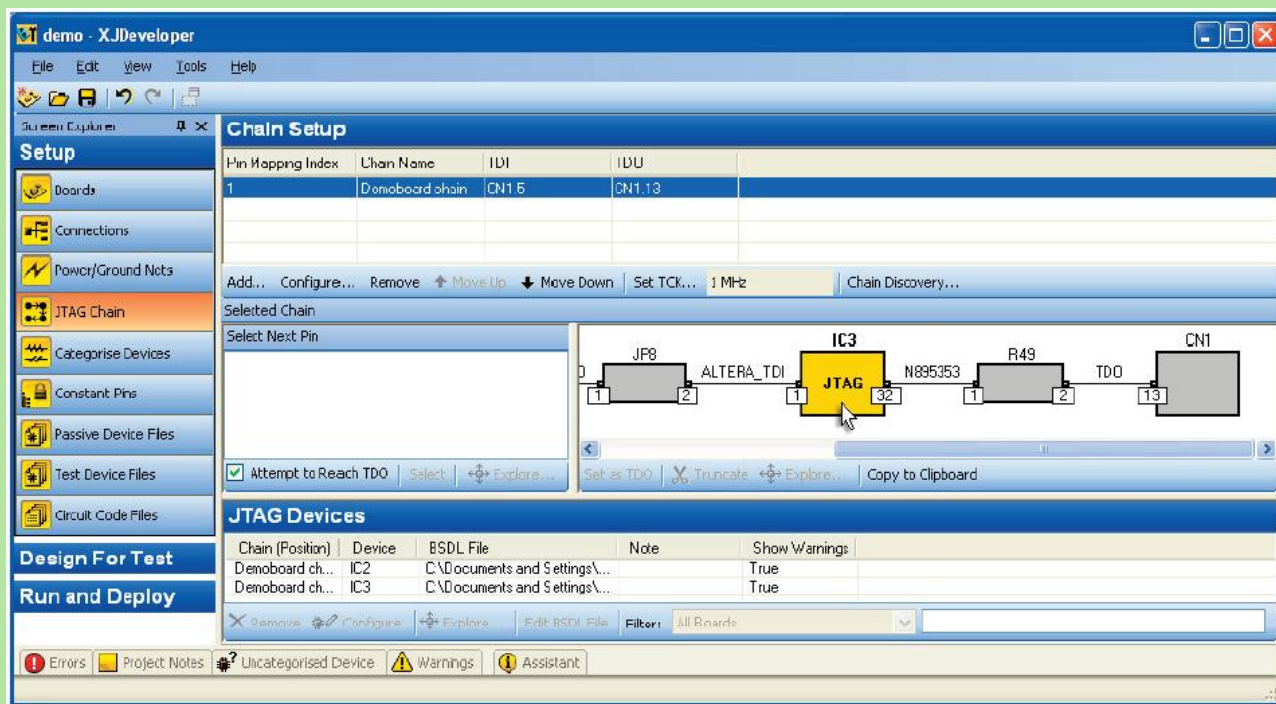
Watch Item	Device Number	Number	Name	Type	Read Value	Write Value	Outputting
Bus 1	1	MULTIPLE	Bus 1	In-Out	2	0	No
A6...	1	A6	PB00_05	In-Out	0	N/A	No
B1...	1	B1	PB00_13	In-Out	0	N/A	No
A7...	1	A7	PB00_06	In-Out	1	N/A	No
B2...	1	B2	PB00_12	In-Out	0	N/A	No

View: Read Value Device: Pin:

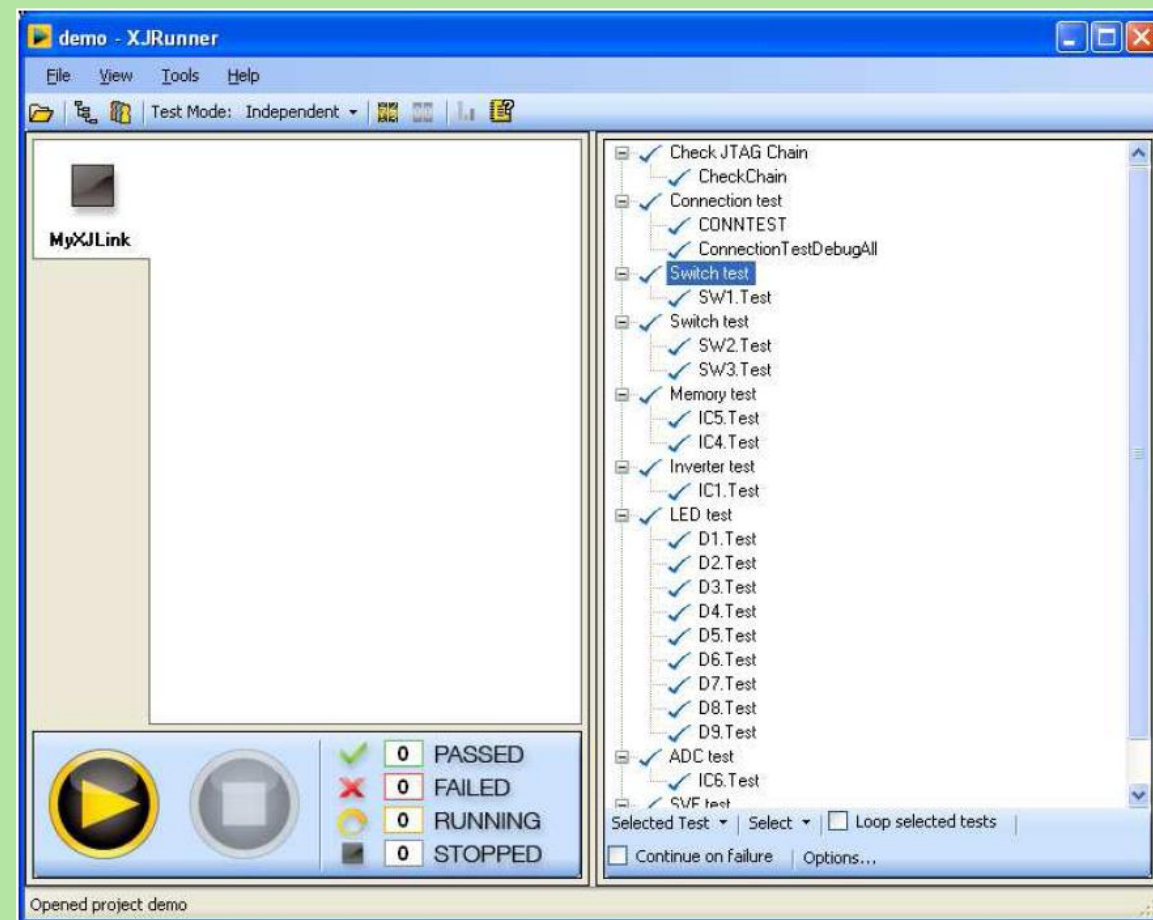
Zalecenia dla projektantów PCB ze ścieżkami testującymi:

- mimo, że JTAG umożliwia testowanie elementów non-JTAG wskazane jest jednak używanie układów ze ścieżkami krawędziowymi,
- należy zgromadzić pliki BSDL dla wszystkich układów z interfejsem JTAG,
- sygnały TDI, TMS, TDO podciągnąć rezystorami 10 kiloomów do zasilania,
- sygnał TCK zakończyć szeregowym rezystorem 68 kiloomów i kondensatorem 100 pF do masy,
- sygnał TDO z ostatniego układu dodatkowo zakończyć szeregowym rezystorem 22 omów,
- wolne piny badanych układów wykorzystać do testowania innych elementów,
- układ non-JTAG włączać poprzez piny układu JTAG w łańcuch,
- linie TCK i TMS powinny być symetryczne - równe długości i pojemności pasożytnicze (przesłuchy),
- elementy z układem watchdog podczas testowania powinny mieć możliwość jego wyłączenia,
- sygnały testujące układy analogowe mogą pochodzić z przetworników AC/DC,
- wykorzystywać możliwości języka JTErase.

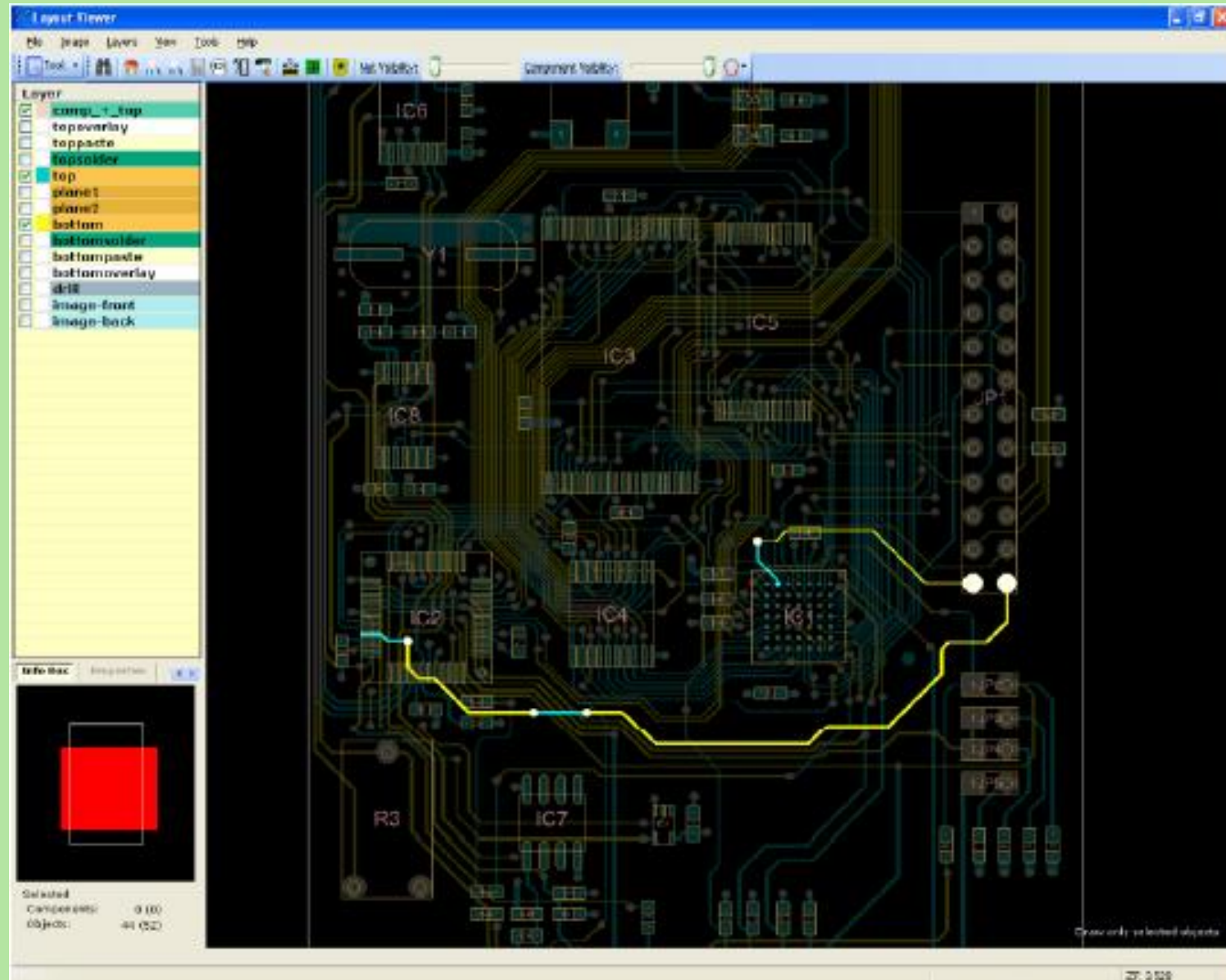
XJRunner jest wyspecjalizowanym środowiskiem uruchomieniowym dla testów automatycznych. Jest on skierowany na potrzeby producentów płytek oraz testowania mobilnego, testuje połączenia, programuje, zarządza numerami seryjnymi. Jest to prosty, w pełni kontrolowany przebieg testowania, generuje dziennik zdarzeń na potrzeby dalszej kontroli, ograniczony dostęp do wyników testów - wybrani użytkownicy.



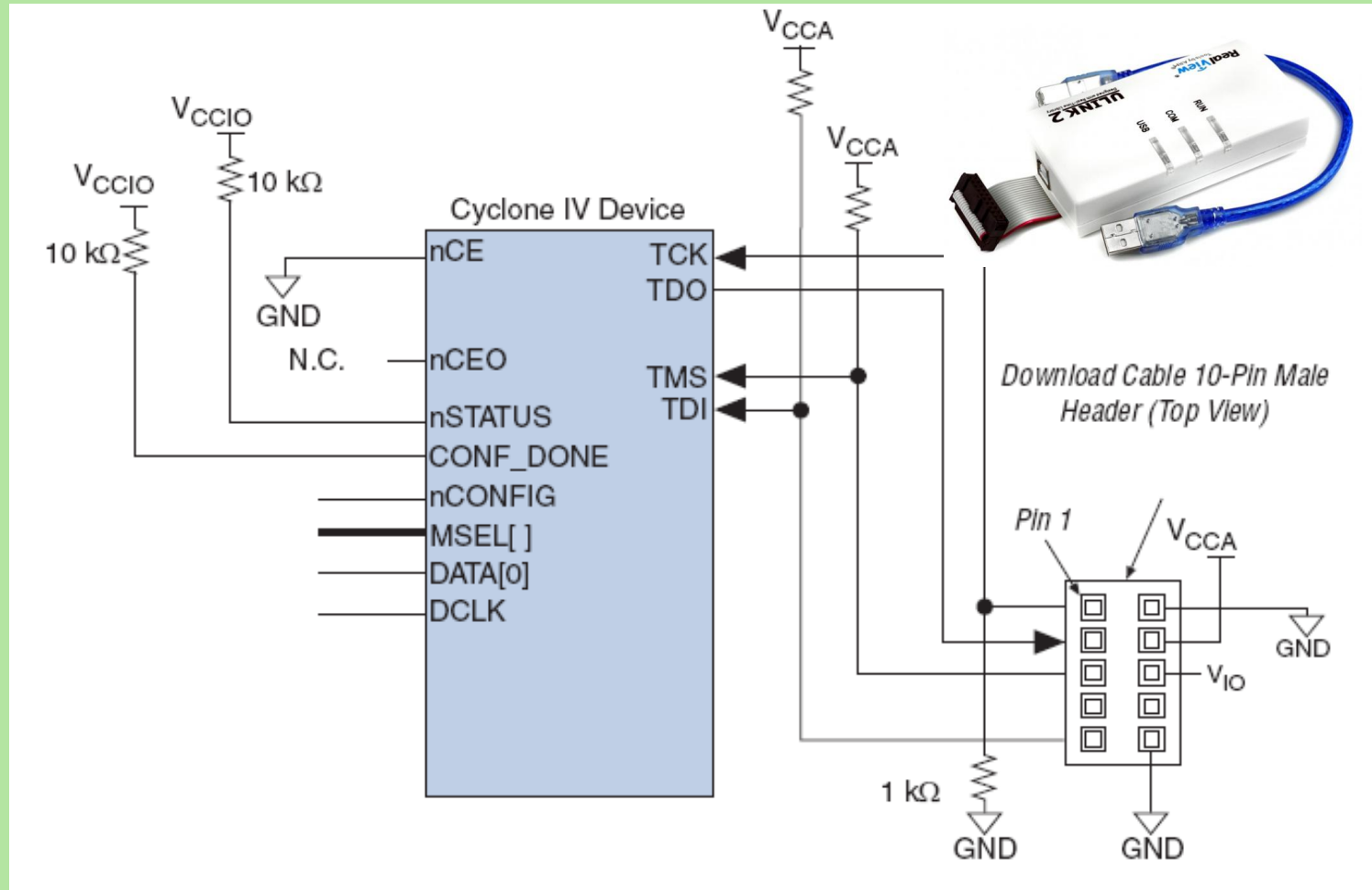
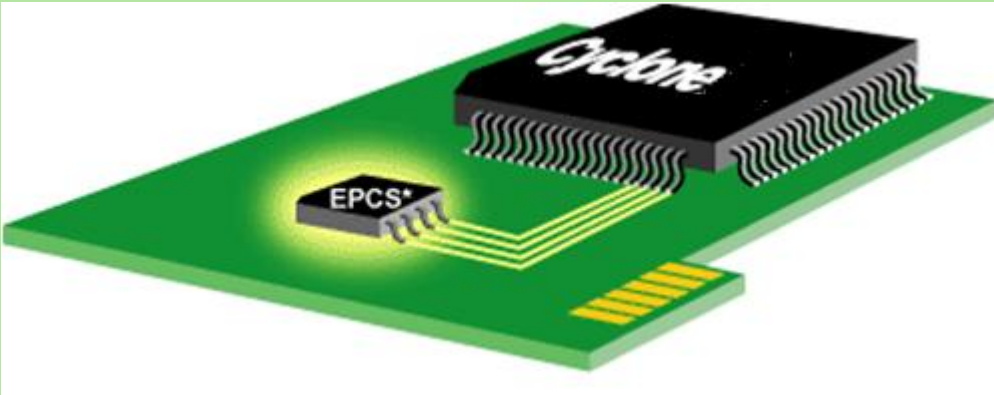
Niezbędny graficzny opis systemu na podstawie plików netlist i BSDL i języku XJEraser.



Efekt zastosowania XJTAG może być następujący:



Ścieżka krawędziowa JTAG to również mechanizm programowania FPGA:



Zaliczenie częściowe wykładu nr 12 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

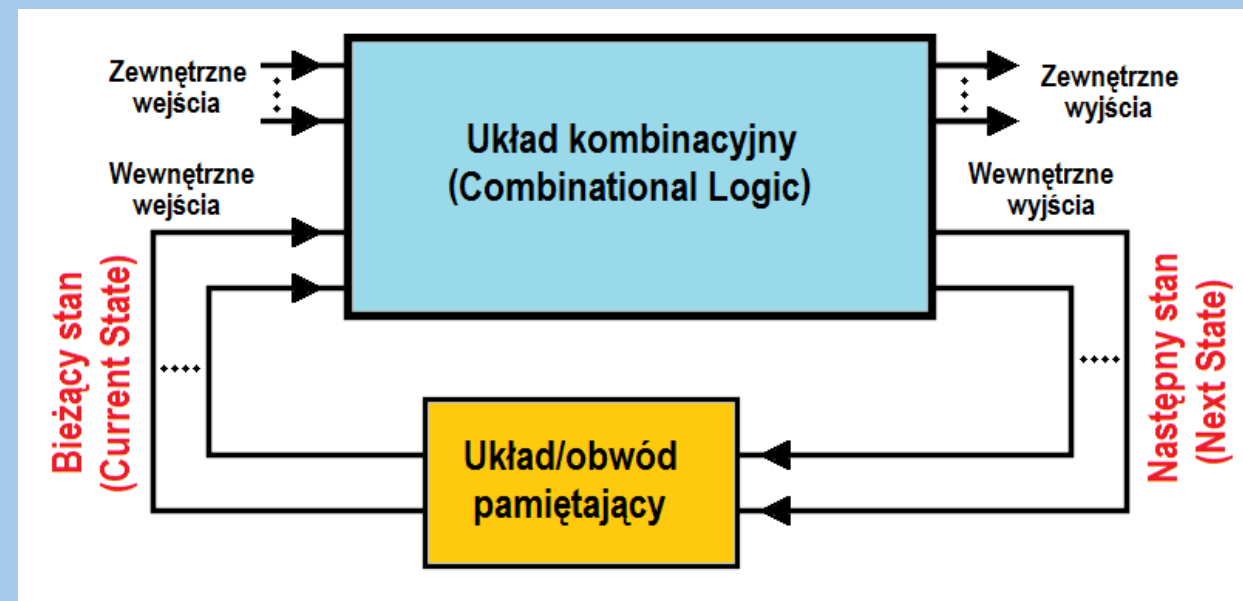
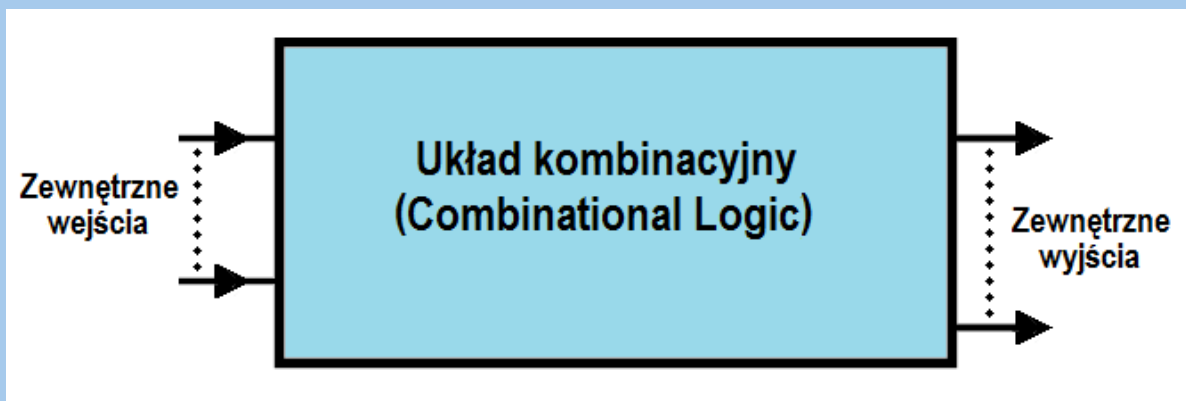
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

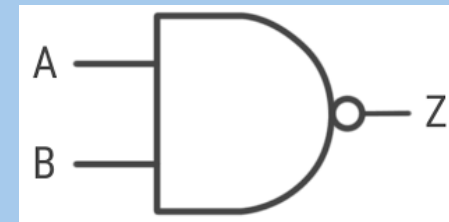
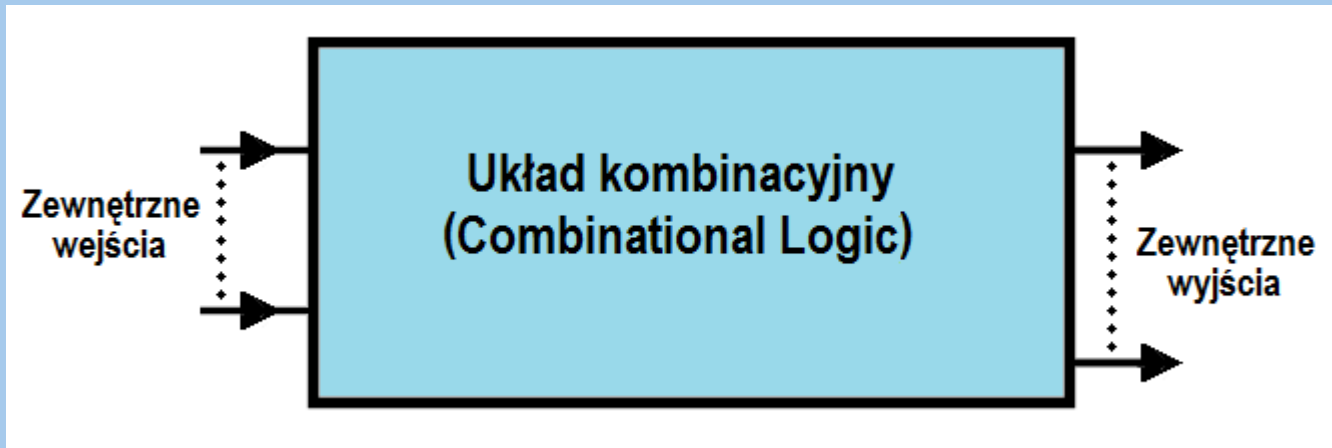
Powodzenia

Układy cyfrowe dzielą się na dwie podstawowe kategorie: układy kombinacyjne i układy sekwencyjne, przy czym układy kombinacyjne mogą wchodzić w skład układów sekwencyjnych. Układy sekwencyjne dzielą się na kolejne subkategorie: układy sekwencyjne asynchroniczne i układy sekwencyjne synchroniczne. Projektowanie części kombinacyjnej (Combinational Logic) wyraźnie różni się od projektowania części sekwencyjnej (Sequential Logic).



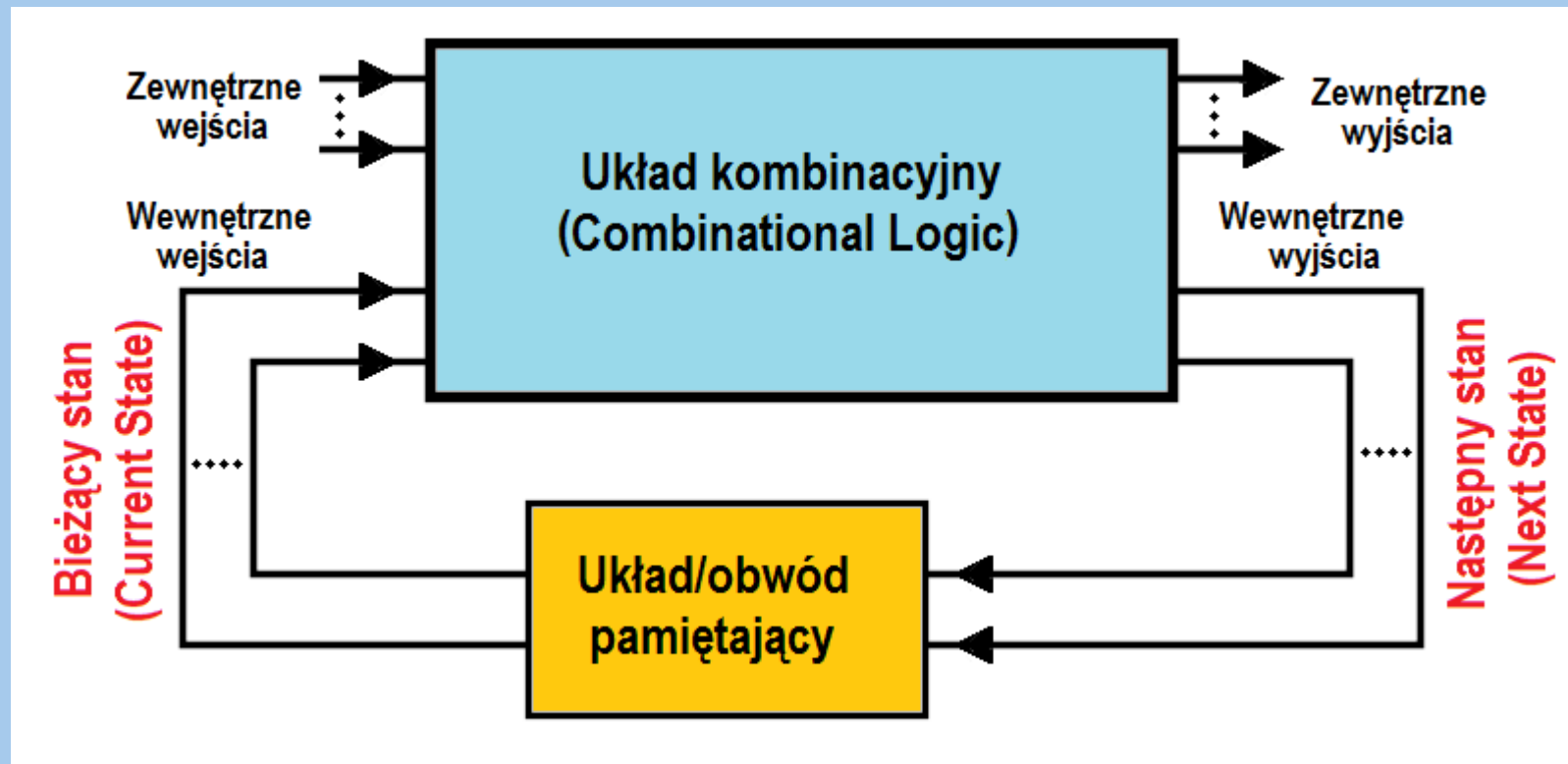
Układ kombinacyjny to taki w którym:

aktualne sygnały wyjściowe zależą od aktualnych wartości sygnałów wejściowych, jeżeli zmieniają się wartości sygnałów wejściowych mogą się zmienić sygnały wyjściowe, zmiana sygnałów wyjściowych może być opóźniona względem wejść o czas propagacji, układ opisują podstawowe równania Boolowskie lub współbieżne instrukcje kodu HDL, pracę układu definiuje tablica działania/prawdy (Truth Table), układ może być zbudowany z bramek lub bardziej złożonych układów kombinacyjnych, typowym przykładem elementarnego układu kombinacyjnego jest bramka logiczna.



Sekwencyjne układy logiczne to takie, których zewnętrzne sygnały wyjściowe zależą nie tylko od aktualnej wartości zewnętrznych sygnałów wejściowych, ale także od poprzedniej sytuacji w której znajdował się układ – mówimy stanu. Układ sekwencyjny to układ kombinacyjny, który jest objęty pętlą sprzężenia zwrotnego. W pętli sprzężenia zwrotnego znajduje się obwód pamiętający poprzedni stan.

Stan układu to jego zachowanie w konkretnej ustabilizowanej sytuacji, to również zestaw wartości jego sygnałów wejściowych i wyjściowych w warunkach ustalonych.

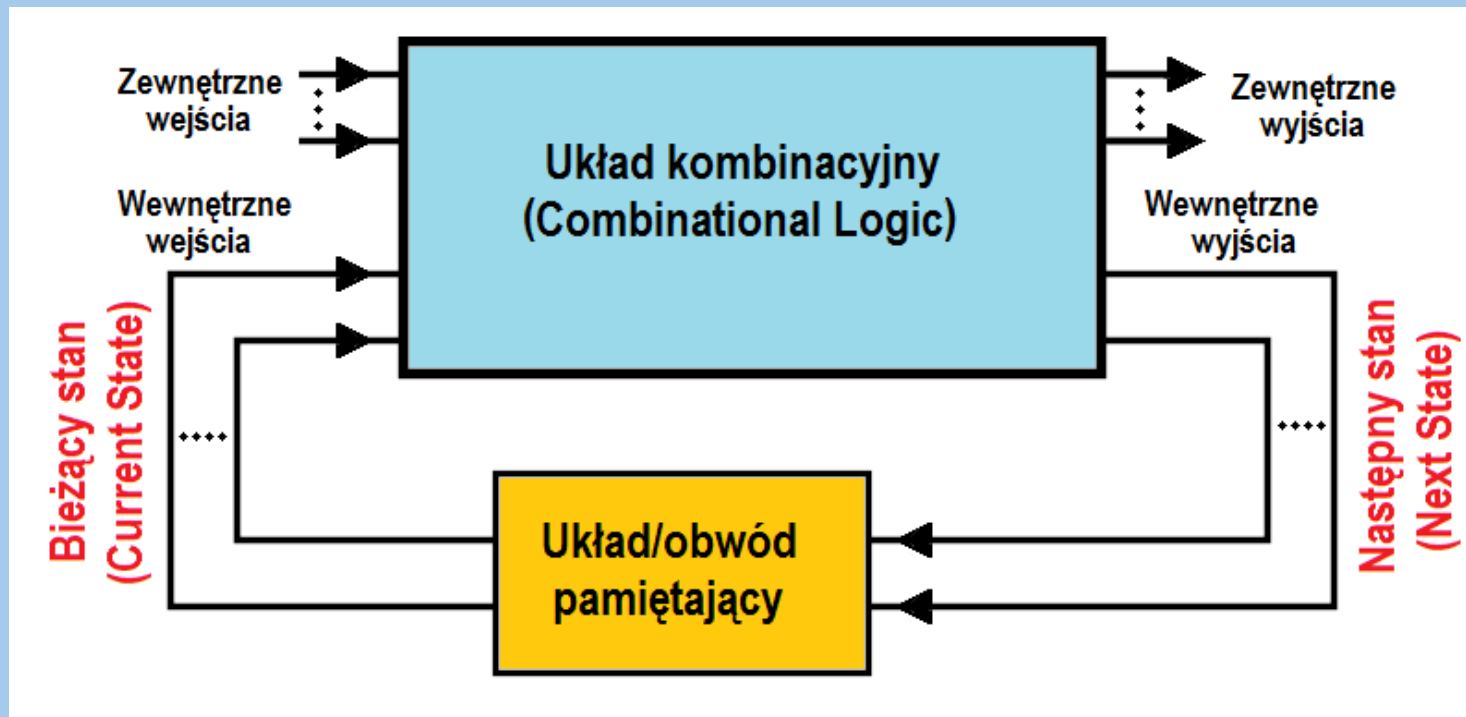


Rolą układu kombinacyjnego jest na podstawie aktualnych wartości wejściowych sygnałów zewnętrznych i informacji o bieżącym stanie wytworzyć:

- wartości zewnętrznych sygnałów wyjściowych,
- wytworzyć informację o stanie następnym.

Układ/obwód pamiętający ma za zadanie przechowywać informacje o bieżącym stanie i stanie następnym.

Istotną cechą, różnicującą układy sekwencyjne jest czynnik synchronizujący zmianę stanu – jest on podstawą klasyfikacji układów sekwencyjnych na: asynchroniczne i synchroniczne.

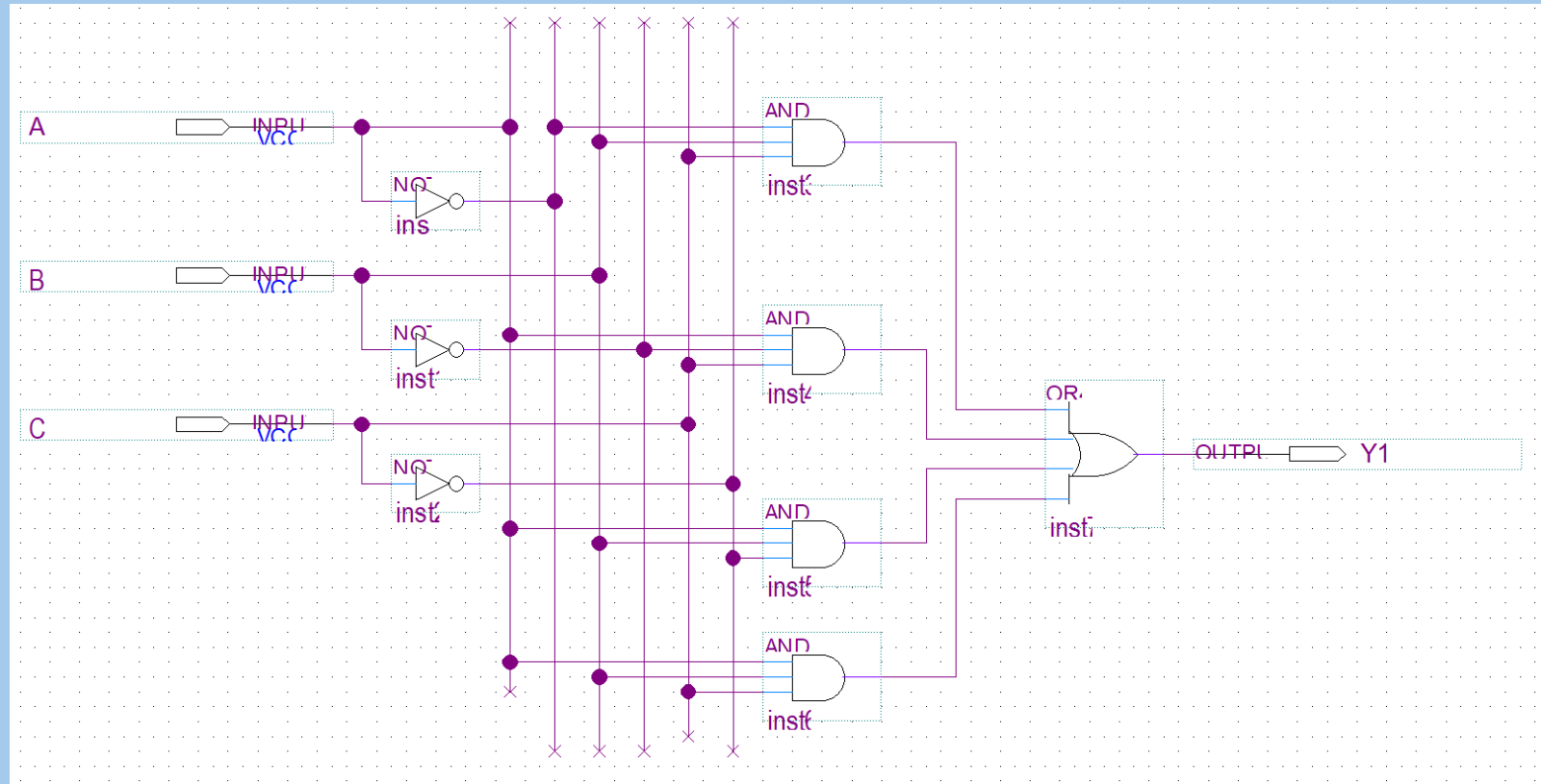


Tradycyjny przykład projektowania układów kombinacyjnych przedstawiony był w ramach wykładu nr 2.

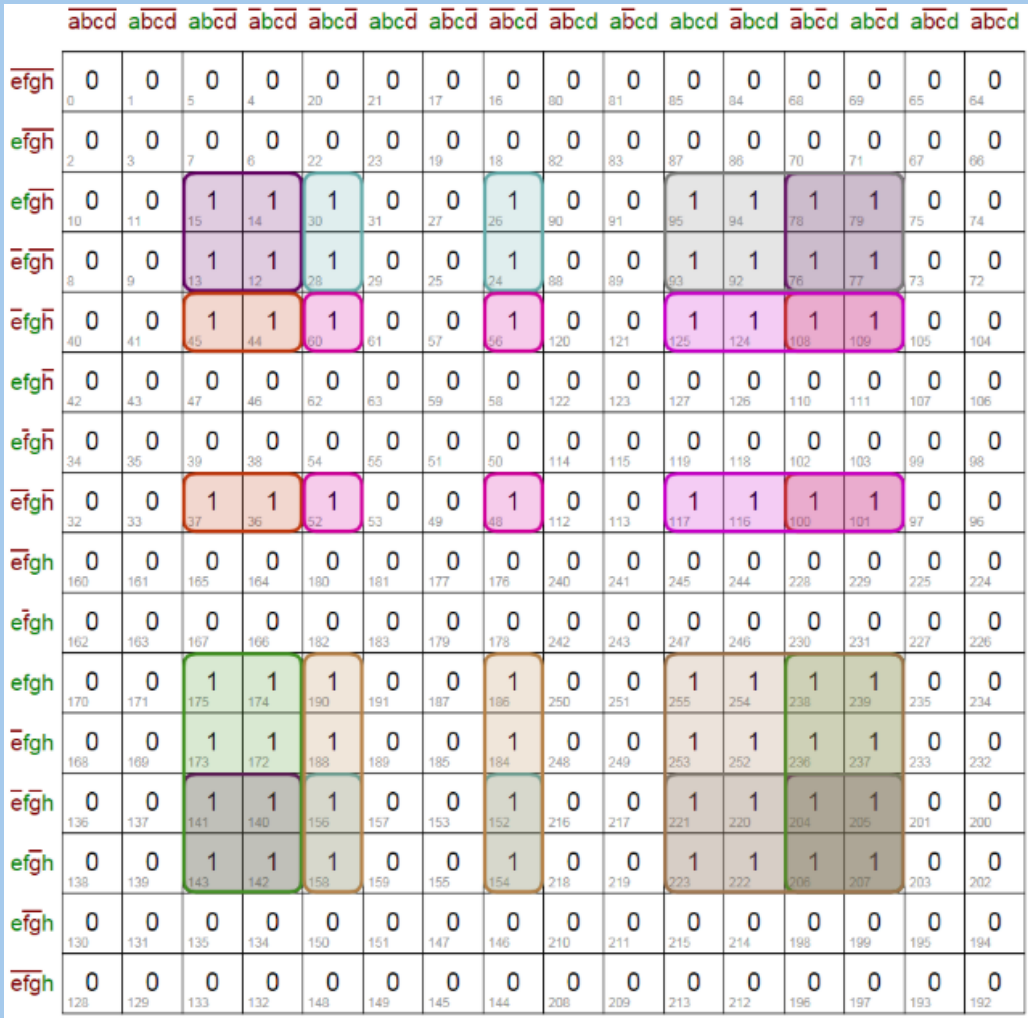
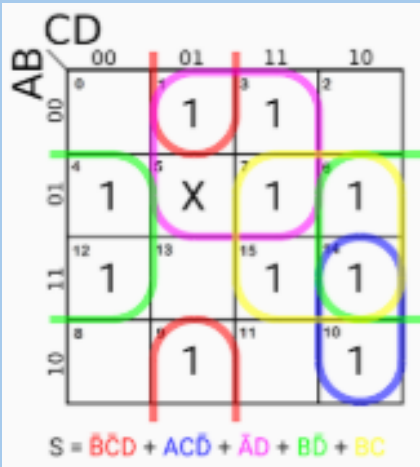
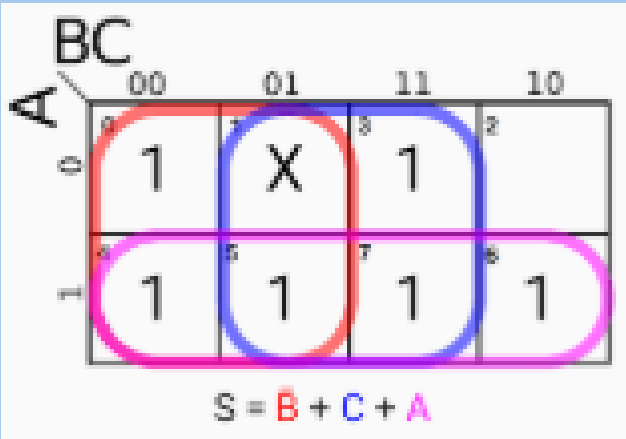
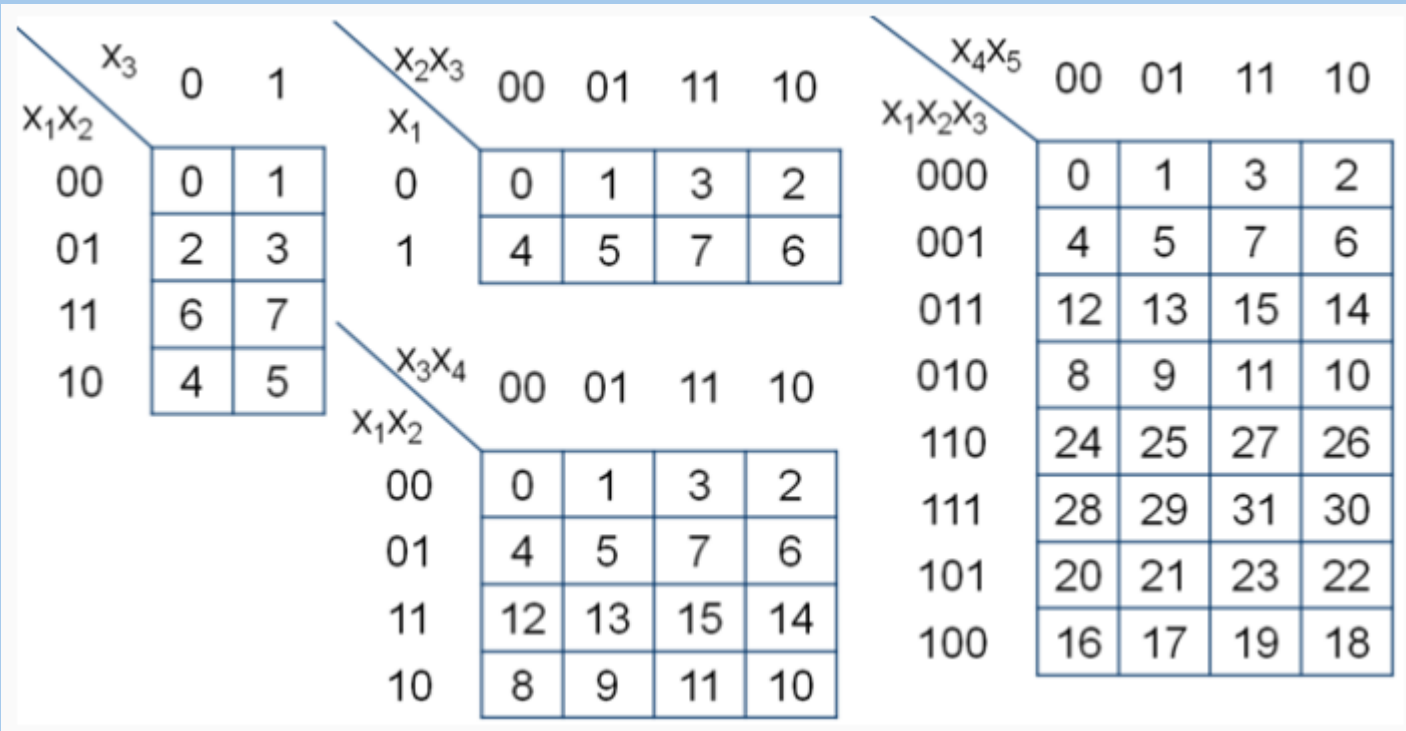
Składał się on z następujących etapów: utworzenia tablicy prawdy, napisania równania logicznego opisującego funkcjonowanie układu – na podstawie tablicy prawdy, minimalizacji równania logicznego, syntezy układu logicznego z użyciem funktorów logicznych.

Wejścia			Wyjścia	
A	B	C	Y1	Y0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$Y1 = F(A,B,C) = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

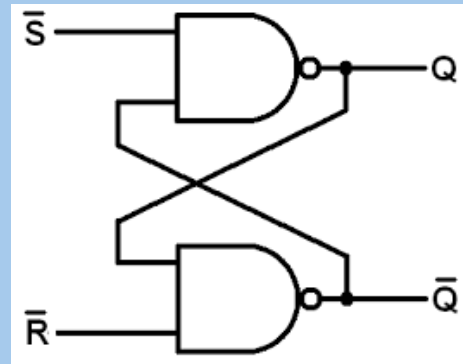


W przeszłości do minimalizacji funkcji stosowano metodę Karnaugh – która obecnie nie ma znaczenia



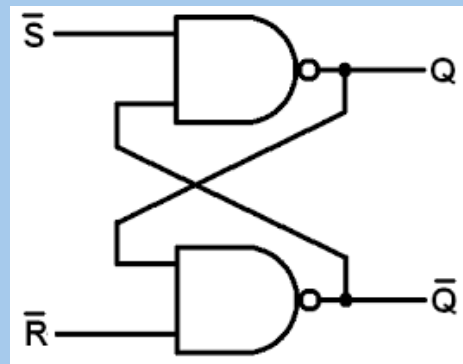
Układy sekwencyjne asynchroniczne to te, które nie zawierają sygnałów zegarowych, zmieniają one swój stan natychmiast po zmianie sygnału wejściowego. Zatem zachowanie układu w dowolnym momencie jest określone przez zmianę sygnałów wejściowych i kolejność w jakiej ta zmiana zachodzi. Sygnał wyjściowy ustala się po upływie czasów propagacji.

Przykładem elementarnego sekwencyjnego układu asynchronicznego jest zatrzask SR:

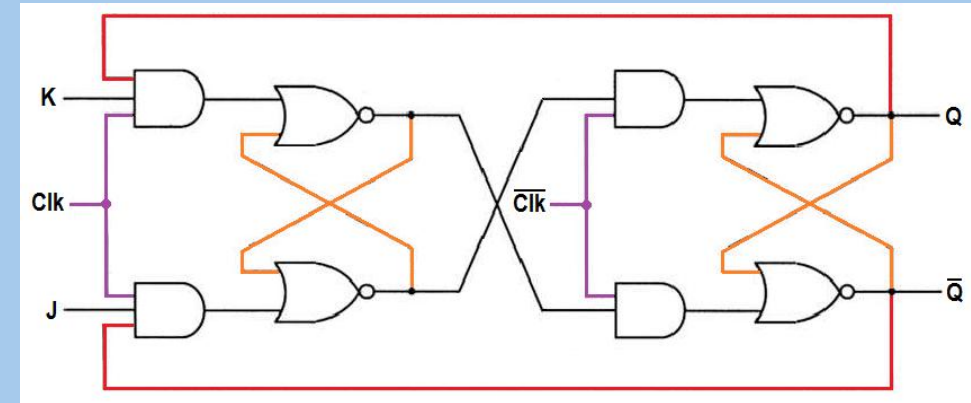
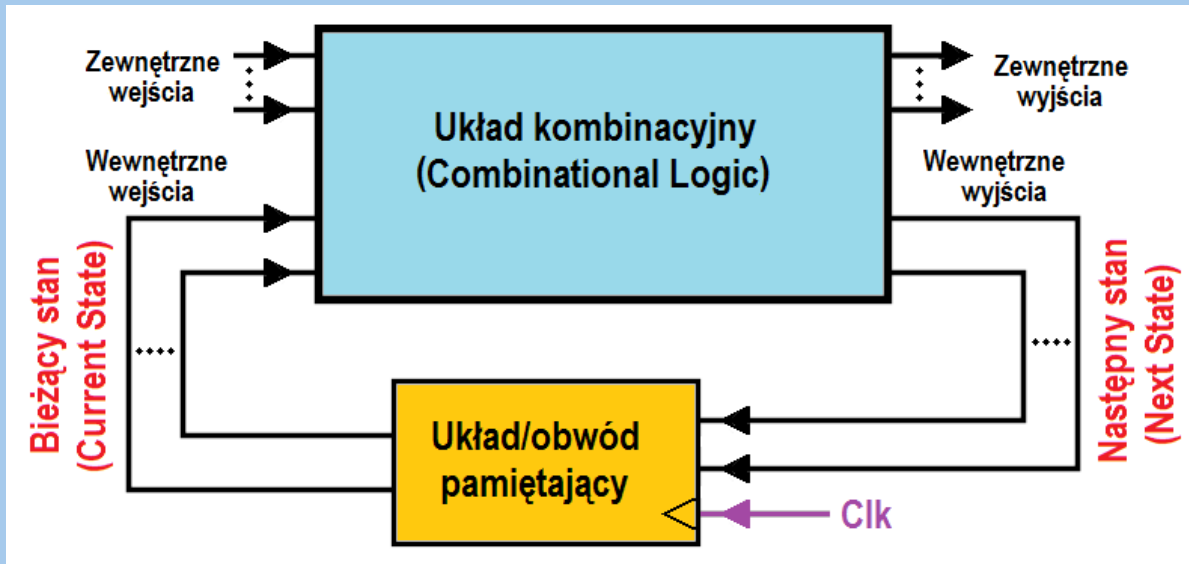


Mimo szybszej pracy i mniejszego poboru mocy, sekwencyjne układy asynchroniczne zostały wyparte przez szybkie sekwencyjne układy synchroniczne, ze względu na problemy z synchronizacją pierwszych. Układy asynchroniczne są trudniejsze w projektowaniu i uruchamianiu a ich praca może być niestabilna z powodu wielu czynników.

Złożone układy asynchroniczne projektuje się z użyciem bramek logicznych oraz przerzutników asynchronicznych, które pełnią rolę układu pamiętającego. W literaturze zagadnienie projektowania układów asynchronicznych zostało dogłębnie opracowane, jednak ta wiedza straciła na znaczeniu wraz z opracowaniem szybkich procesorów i układów programowalnych.



W odróżnieniu do układów asynchronicznych, w układach synchronicznych zmiana stanu następuje nie tylko w wyniku zmiany poziomów sygnałów wejściowych i kolejności ich zmiany – lecz przede wszystkim jest wyznaczana przez zewnętrzny sygnał zegarowy. Wprowadzając sygnał zegarowy do układów sekwencyjnych rozerwano łańcuch zależności od czasów propagacji, kontrolując ryzyko metastabilności. Przykładem układu jest JK Flip-Flop.



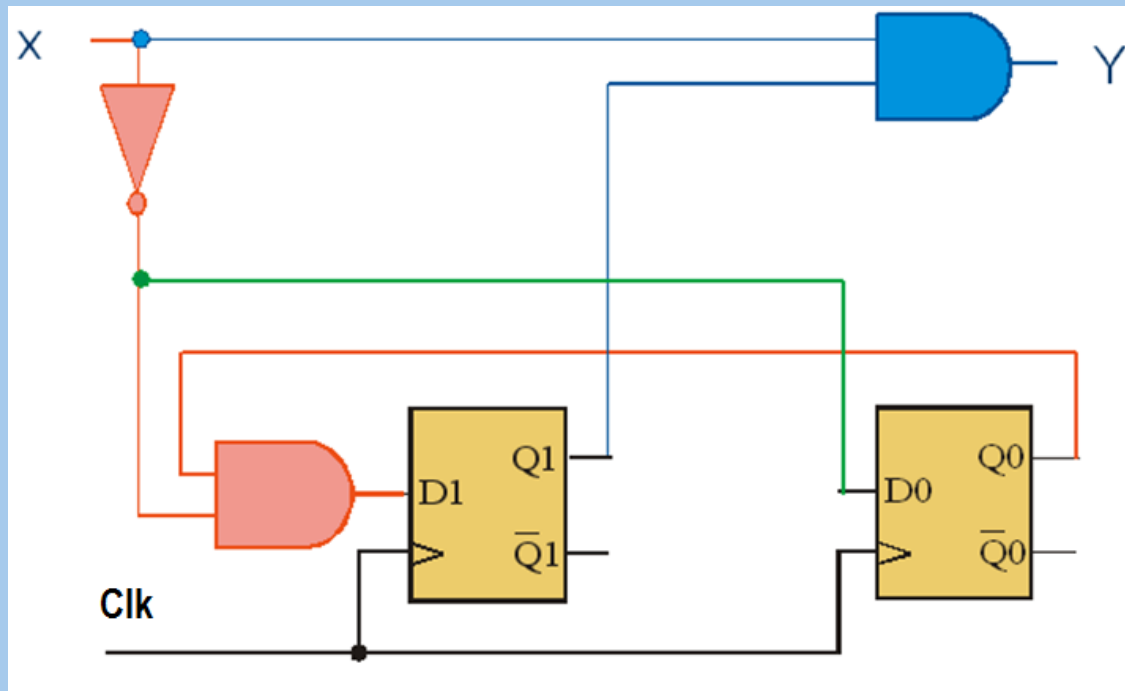
Klasyczna metoda projektowania sekwencyjnych układów synchronicznych obejmowała wykorzystanie bramek logicznych i przerzutników synchronicznych. W efekcie powstawał skomplikowany układ a sama metoda była zawodna w przypadku rozbudowanych układów.

Z tych samych względów, co w przypadku układów asynchronicznych, tradycyjna metoda projektowania układów sekwencyjnych synchronicznych straciła na znaczeniu.

Współczesne układy synchroniczne projektuje się:

schematycznie z wykorzystaniem gotowych podzespołów synchronicznych,

stosując technikę Finite State Machine w językach HDL układów programowalnych.



Do syntezy układów kombinacyjnych w strukturach programowalnych możemy zaprojektować układ metodą klasyczną i schemat narysować w systemie a następnie skompilować i zaimplementować w FPGA. Jest to jednak żmudna i nieefektywna metoda, gdyż część projektową można skrócić i zrealizować z wykorzystaniem kompilatora systemowego, który dokona minimalizacji nawet układów o wielu wejściach.

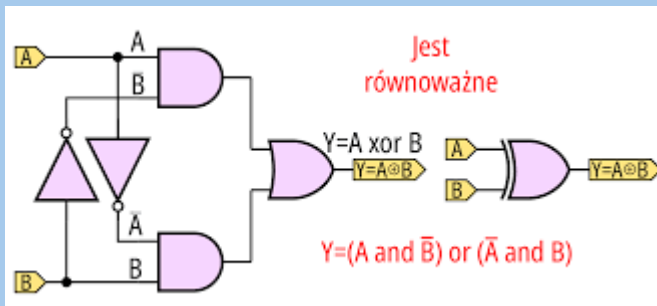
W takim układzie rolą projektanta pozostaje jedynie opis zachowania projektowanego układu, który został przygotowany w sposób zrozumiały dla systemu.

Zrozumiałą formą opisu układu jest kod z użyciem instrukcji języka HDL. W przypadku układów kombinacyjnych będzie to najczęściej na podstawie tablicy prawdy użycie instrukcji:

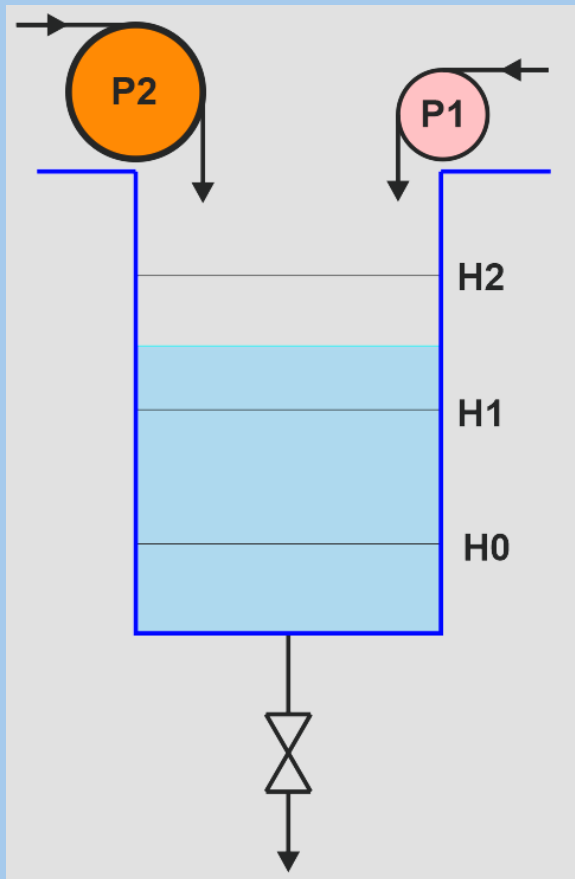
- prostego przypisania sygnału,
- selektywnego przypisania sygnału,
- warunkowego przypisania sygnału,

oraz w szczególnych przypadkach instrukcji powielania strukturalnego Generate.

Następne przykłady obrazują sposób zastosowania instrukcji przypisania sygnału w 3 wersjach.



Zbiornik z płynem napełniają 2 pompy: o większej wydajności P1 i mniejsza P2. W zbiorniku znajdują się 3 czujniki poziomów H0, H1 i H2. Pompy uruchamia się sygnałem w stanie wysokim, na wyjściu czujnika jest poziom wysoki jeżeli czujnik jest zanurzony w płynie. Algorytm sterowania pompami jest następujący: jeżeli poziom płynu jest poniżej H0 – pracują obie pompy, jeżeli poziom jest między H0 a H1 pracuje P1, jeżeli poziom jest między H1 i H2 pracuje P2, dla poziomu powyżej H2 – pompy wyłączane. Opracowujemy tablicę prawdy:



Wejścia			Wyjścia	
H2	H1	H0	P1	P2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Używając instrukcji prostego przypisania sygnału dla każdego z wyjść oddzielnie w VHDL opisujemy układ – **NIE MINIMALIZUJĄC**:

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Pompy0 is
5  port
6  (
7      H : in  std_logic_vector(2 downto 0);
8      P : out std_logic_vector(2 downto 1)
9  );
10 end entity;
11
12 architecture rtl of Pompy0 is
13 begin
14
15     P(1) <= ((not(H(2))) and (not(H(1))) and (not(H(0))))
16            or ((not(H(2))) and (not(H(1))) and H(0));
17
18     P(2) <= ((not(H(2))) and (not(H(1))) and (not(H(0))))
19            or ((not(H(2))) and H(1) and (H(0)));
20
21 end rtl;
22

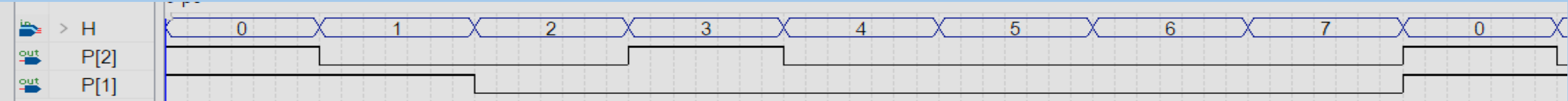
```

Wejścia			Wyjścia	
H2	H1	H0	P1	P2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

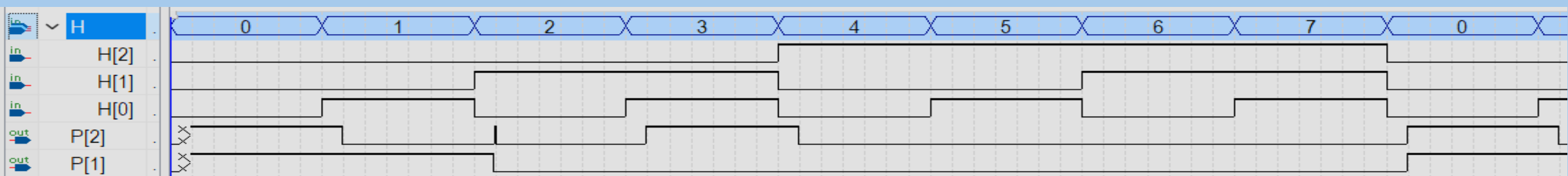
Wyniki symulacji weryfikują poprawność projektu:

Wejścia			Wyjścia	
H2	H1	H0	P1	P2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Symulacja funkcjonalna:



Symulacja czasowa – widoczne zjawisko hazardu:

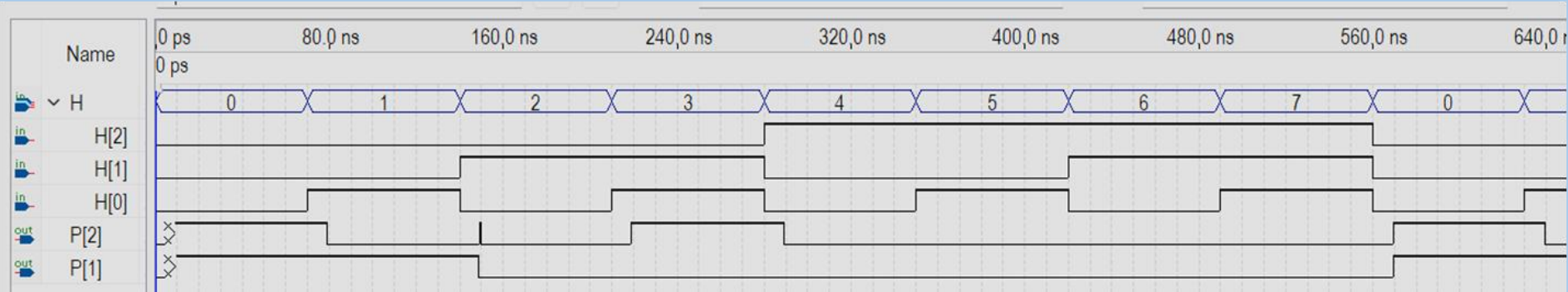


Instrukcja prostego przypisania sygnału może funkcjonować jako instrukcja sekwencyjna:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Pompy0 is
5  port
6  (
7      H : in  std_logic_vector(2 downto 0);
8      P : out std_logic_vector(2 downto 1)
9  );
10 end entity;
11
12 architecture rtl of Pompy0 is
13 begin
14     process(H)
15     begin
16         P(1) <= ((not(H(2))) and (not(H(1))) and (not(H(0))))
17                or ((not(H(2))) and (not(H(1))) and H(0));
18         P(2) <= ((not(H(2))) and (not(H(1))) and (not(H(0))))
19                or ((not(H(2))) and H(1) and (H(0)));
20     end process;
21 end rtl;
```

Wejścia			Wyjścia	
H2	H1	H0	P1	P2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Symulacja czasowa – zjawisko hazardu pozostało:



Gdy liczba bitów sygnału wejściowego układu kombinacyjnego zwiększa się – jego synteza z użyciem instrukcji prostego przypisania sygnału staje się co najmniej kłopotliwa:

Lp.	Wejścia						Wyjście
	S1	S0	A	B	C	D	X
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0
2	0	0	0	0	1	0	1
3	0	0	0	0	1	1	1
4	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...
62	1	1	1	1	1	0	0
63	1	1	1	1	1	1	1

W tym przypadku równanie logiczne składa się z 64 kombinacji sygnałów wejściowych:

$X \leq ((\text{not}(S(1)) \text{ and } (\text{not}(S(0)) \text{ and } (\text{not}(A)) \text{ and } (\text{not}(B)) \text{ and } (\text{not}(C)) \text{ and } (\text{not}(D))) \text{ or}$

$(S(1) \text{ and } S(0) \text{ and } A \text{ and } B \text{ and } C \text{ and } D;$

co komplikuje opis, ale jest on jeszcze możliwy.

W przypadku układów wielowyjściowych bardziej korzystne jest użycie instrukcji selektywnego przypisania sygnału – niż instrukcji prostego przypisania:

```
ARCHITECTURE example 2 OF example3 IS
  signal AA : std_logic_vector(2 downto 0);
  signal BB : std_logic_vector(1 downto 0);
BEGIN
  AA <= A2 & A1 & A0;
  with AA select
    BB <= „10” when "000",
        „11” when "001",
        „10” when "010",
        „01” when "011",
        „01” when "100",
        „01” when "101",
        „11” when "110",
        „11” when "111";

  B1 <= BB(1);
  B0 <= BB(0);
END example2;
```

Wejście A			Wyjście B	
A2	A1	A0	B1	B0
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Poprzedni kod można skrócić używając klauzuli **when others**. Należy zauważyć, że w 3 przypadkach w tabeli występują te same wartości sygnałów wyjściowych. Zatem można je opisać w wierszu końcowym stosując wspomnianą klauzulę. Warto zauważyć, że w tabeli występuje druga trójka przypadków o tych samych sygnałach wyjściowych.

ARCHITECTURE example 2 OF example3 IS

signal AA : std_logic_vector(2 downto 0);

signal BB : std_logic_vector(1 downto 0);

BEGIN

AA <= A2 & A1 & A0;

with AA select

*BB <= „10” when “000”,
 „11” when “001”,
 „10” when “010”,
 „01” when “011”,
 „01” when “100”,
 „01” when “101”,
 „11” when others ;*

B1 <= BB(1);

B0 <= BB(0);

END example2;

Wejście A			Wyjście B	
A2	A1	A0	B1	B0
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Kod źródłowy można jeszcze bardziej skrócić łącząc przypadki o tych samych sygnałach wejściowych, przy czym operator "|" oznacza "lub", a operator "to" zakres od – do.

```
ARCHITECTURE example 2 OF example3 IS  
  signal AA : std_logic_vector(2 downto 0);  
  signal BB : std_logic_vector(1 downto 0);  
BEGIN  
  AA <= A2 & A1 & A0;  
  with AA select  
    BB <= „10” when "000" | "010",  
        „01” when "011" to "101",  
        „11” when others ;  
  B1 <= BB(1);  
  B0 <= BB(0);  
END example2;
```

Wejście A			Wyjście B	
A2	A1	A0	B1	B0
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Ekwiwalentnym w stosunku do instrukcji selektywnego przypisania sygnału jest użycie instrukcji warunkowego przypisania sygnału:

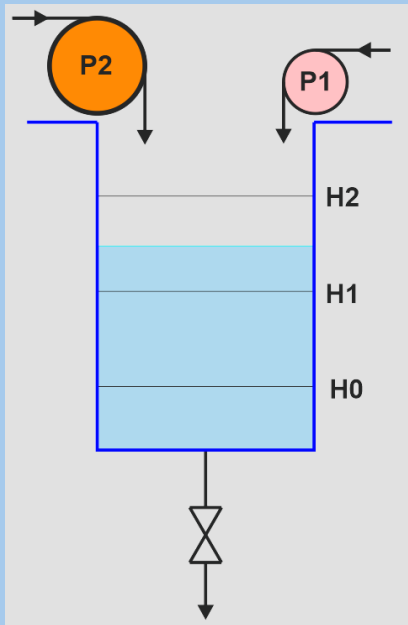
```
X7 <= „111110” when DCBA = „0000” else
    „0110000” when DCBA = „0001” else
    „1101101” when DCBA = „0010” else
    „1111001” when DCBA = „0011” else
    „0110011” when DCBA = „0100” else
    „1011011” when DCBA = „0101” else
    „0011111” when DCBA = „0110” else
    „1110000” when DCBA = „0111” else
    „1111111” when DCBA = „1000” else
    „1110011” when DCBA = „1001” else
    „0000000”;
```

Decimal	Binary DCBA	7 Segment Code						
		a	b	c	d	e	f	g
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	0	0	1	1	1	1	1
7	0111	1	1	1	0	0	0	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	0	0	1	1

Którą instrukcję zastosować:

- instrukcje selektywnego i warunkowego przypisania są równoważne pod względem funkcjonalnym,
- czasami jedna z nich pozwala skrócić kod opisujący,
- trzeba przeanalizować układ "zer" i "jedynek" w tablicy prawdy,
- jeżeli na wyjściu znajduje się wiele jedynek w kolejnych przypadkach korzystniejsza
- jest instrukcja warunkowego przypisania do sygnału,
- jeżeli jedynki na wyjściu są "rozstrzelane" - instrukcja selektywnego przypisania jest prostsza.

Sięgnijmy do poprzedniego przykładu:



Wejścia			Wyjścia	
H2	H1	H0	P1	P2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Ten układ można opisać na kilka sposobów:

with H select

P1 <= '1' when "000" | "001",
'0' when others;

with H select

P2 <= '1' when "000" | "011",
'0' when others;

P1 <= '1' when **H** < "010" else '0';

P2 <= '1' when (**H** = "000") or
(**H** = "011") else '0';

Wejścia			Wyjścia	
H2	H1	H0	P1	P2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

W odróżnieniu od instrukcji selektywnego przypisania sygnału – instrukcję warunkowego przypisania sygnału można sterować wieloma sygnałami wejściowymi bez potrzeby ich konsolidacji. Np.:

ARCHITECTURE Ex2 OF Ex3 IS

signal *H* : std_logic_vector(2 **downto** 0);

signal *P* : std_logic_vector(2 **downto** 1);

BEGIN

Z <= *H*2 & *H*1 & *H*0;

with *H* **select**

P <= "11" **when** "000",

"10" **when** "001",

"01" **when** "011",

"00" **when others**;

...

END Ex2;

Wejścia			Wyjścia	
H2	H1	H0	P1	P2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

ARCHITECTURE Ex2 OF Ex3 IS

signal *P* : std_logic_vector(2 **downto** 1);

BEGIN

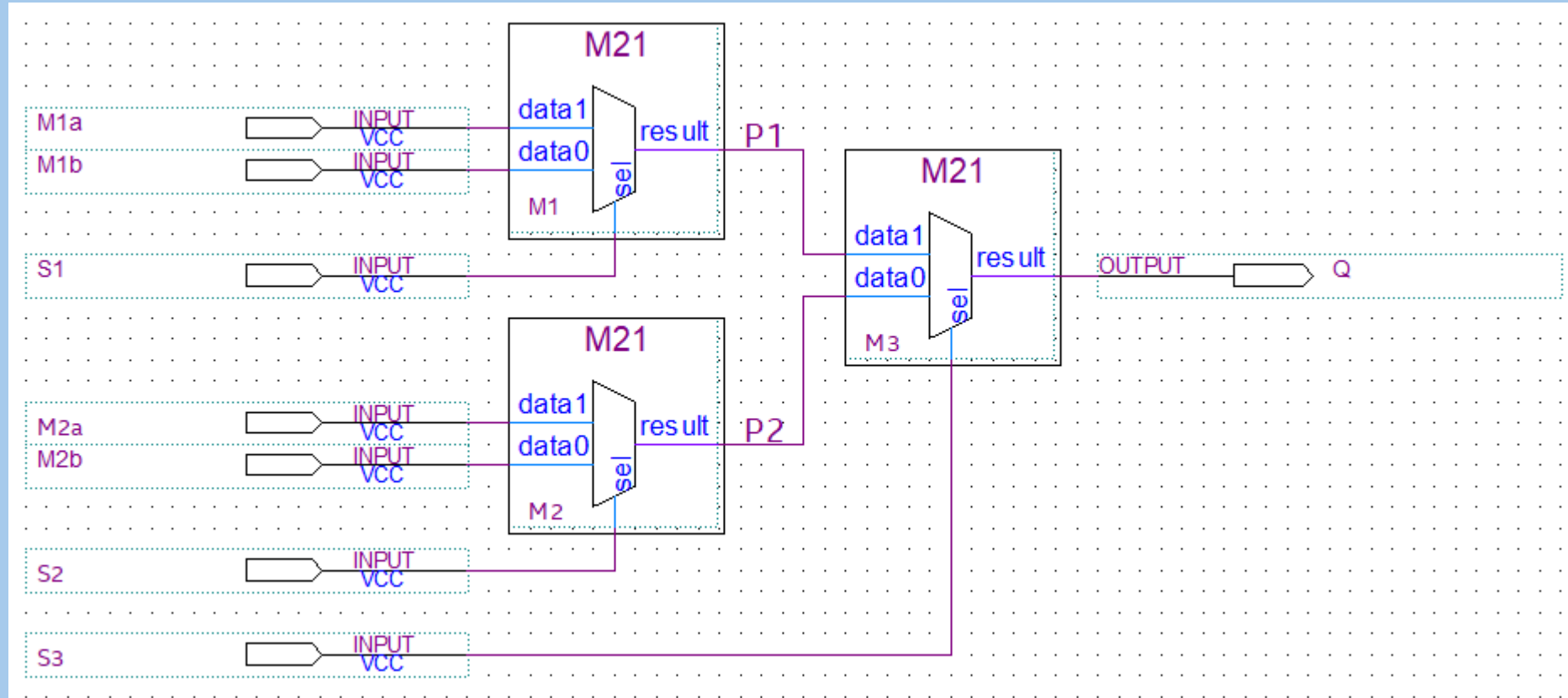
P <= "11" **when** ((*H*2 or *H*1 or *H*0) = '0') else "10" **when** ((*H*2 or *H*1) = '0') **and** (*H*0 = '1') else

"01" **when** ((*H*0 or *H*1) = '1') **and** (*H*2 = '0') else "00";

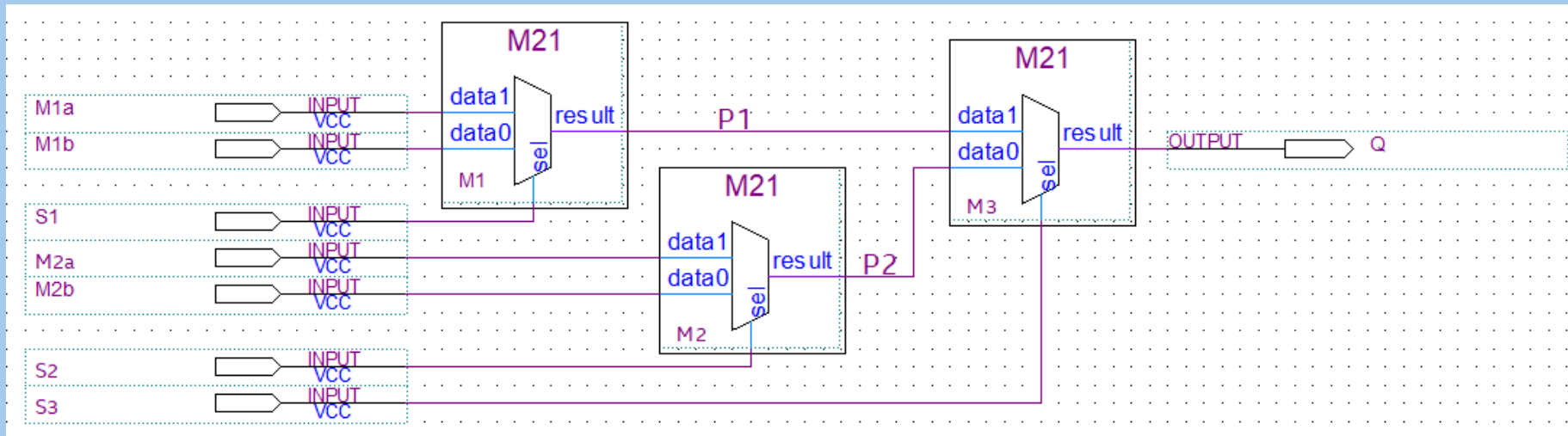
...

END Ex2;

Weźmy inny przykład: trzech multiplexerów połączonych ze sobą



Napişmy kod opisujący układ – tym razem nie mamy wyboru i musimy użyć sygnałów wewnętrznych P1 i P2.



entity BB is

port (M1a, M1b, M2a, M2b, S1, S2, S3 : in bit);

end entity;

architecture AA of BB is

signal P1, P2 : bit := '0';

begin

with S1 select P1 <= M1a when '1', M1b when others;

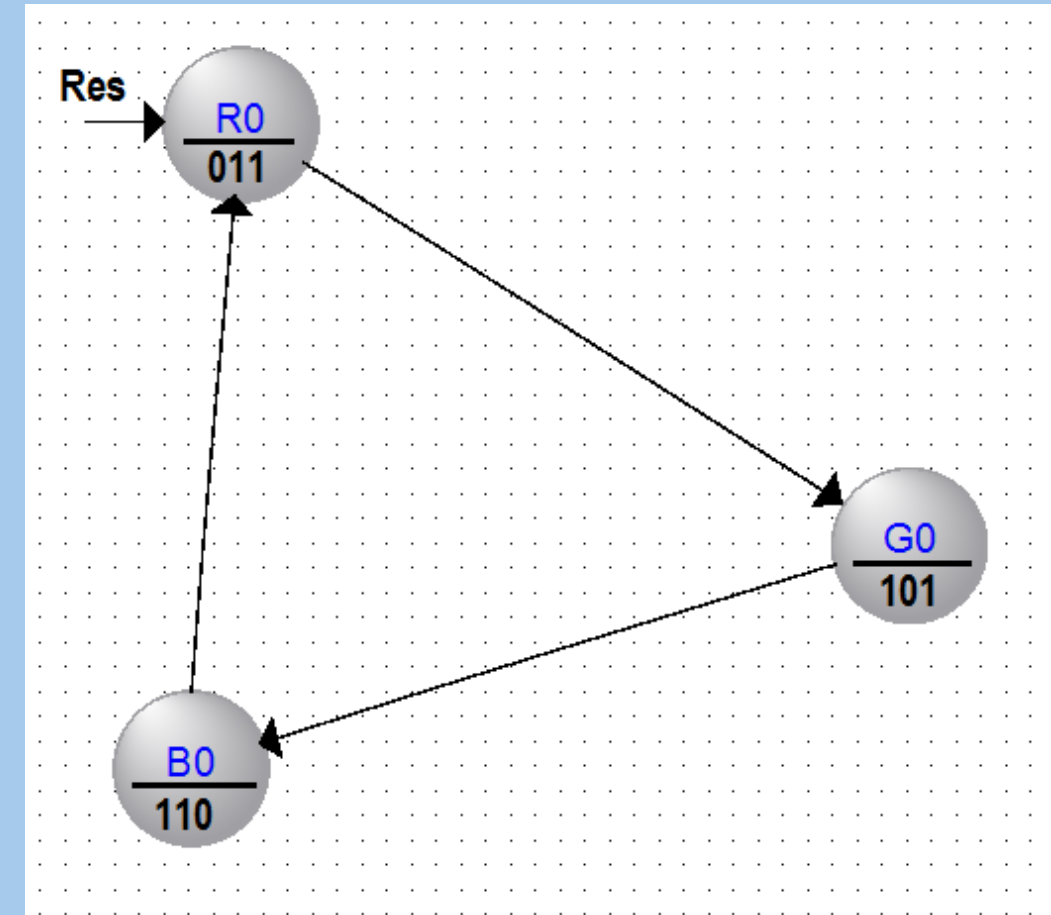
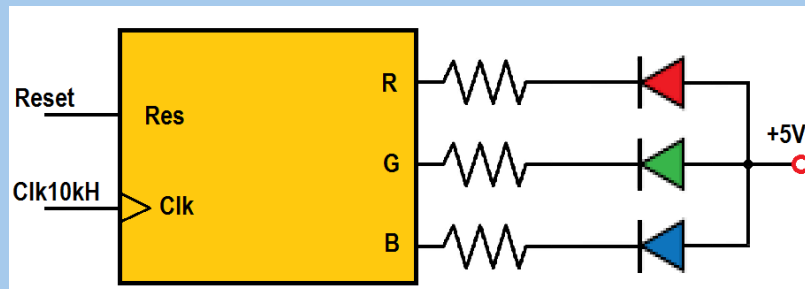
with S2 select P2 <= M2a when '1', M2b when others;

Q <= P1 when (S3 = '1') else P2;

end AA;

Układy sekwencyjne projektujemy stosując technikę FSM, która w sposób sformalizowany będzie omówiona w czasie następnego wykładu. W tym miejscu pokażemy sposób tworzenia opisu układu sekwencyjnego w sposób graficzny, tak jak tabela prawdy opisuje układ kombinacyjny. Technika definiowania układu jest jego graf, który pokazuje występujące stany oraz wartości wyjść w poszczególnych stanach i warunki przejścia między stanami.

Weźmy przykład: chcemy zaprojektować układ sterujący LED RGB, tak żeby: najpierw przez 10 us świeciła składowa czerwona (R), następnie zielona (G) a na koniec niebieska (B). Zatem układ zawiera wejście zegarowe Clk 10 kHz i wejście zerujące Res oraz trzy wyjścia R, G i B. W związku z tym, że składowe monochromatyczne połączone są w układzie wspólnej anody, świecenie określonej barwy wymaga podania stanu niskiego na jej katodę. Projektowany układ zawiera trzy stany: R0 z wyjściem "011", G0 z wyjściem "101" oraz B0 z wyjściem "110". Graf układu pokazano na rysunku obok.

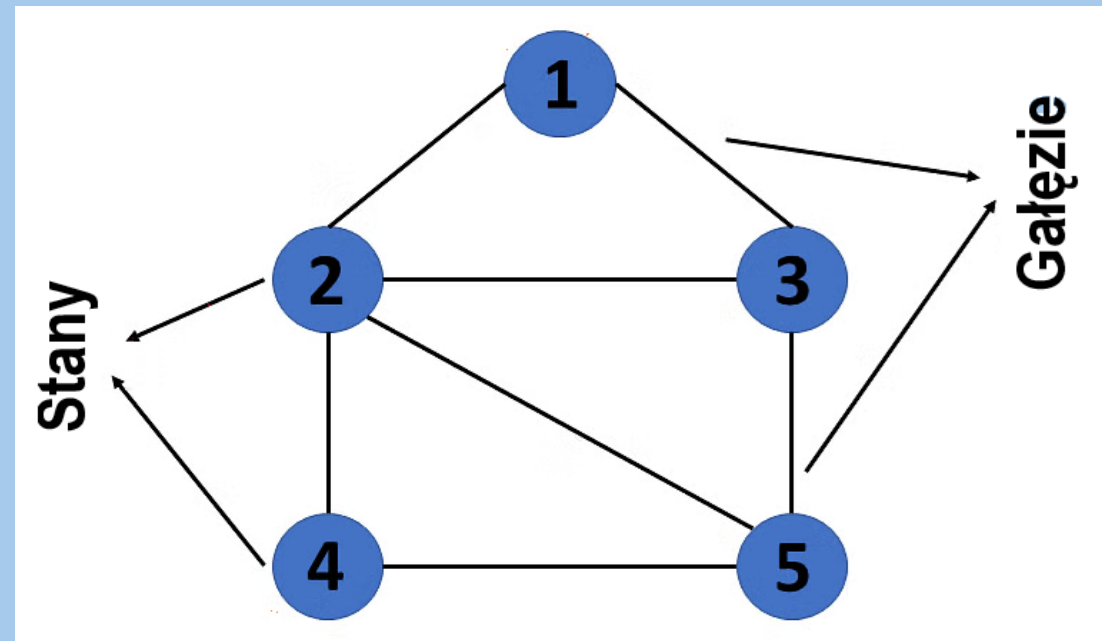
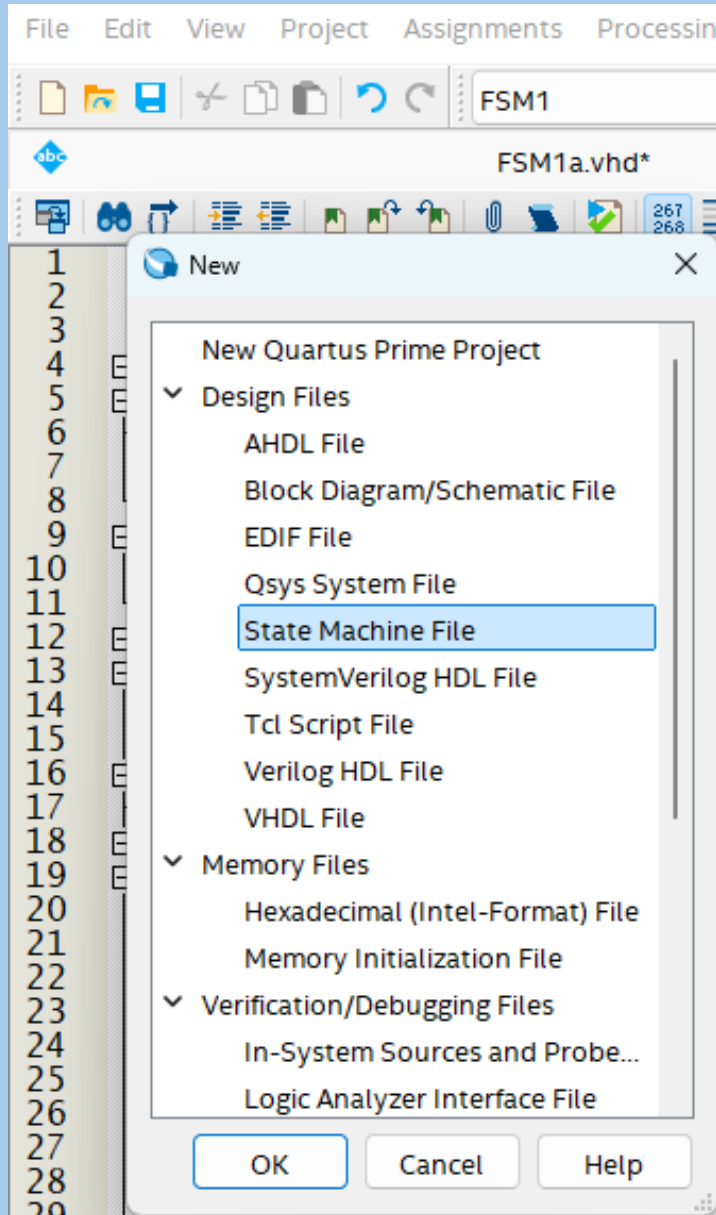


W pewnej klasie układów sekwencyjnych udział projektanta kończy się na utworzeniu grafu, a pozostałe czynności wykonuje system projektowy automatycznie. Jako przykład opracujmy graf 3-bitowego licznika sterowanego, który posiada 2 wejścia sterujące A i B wyznaczające tryb pracy. Zgodnie z nim jeżeli:

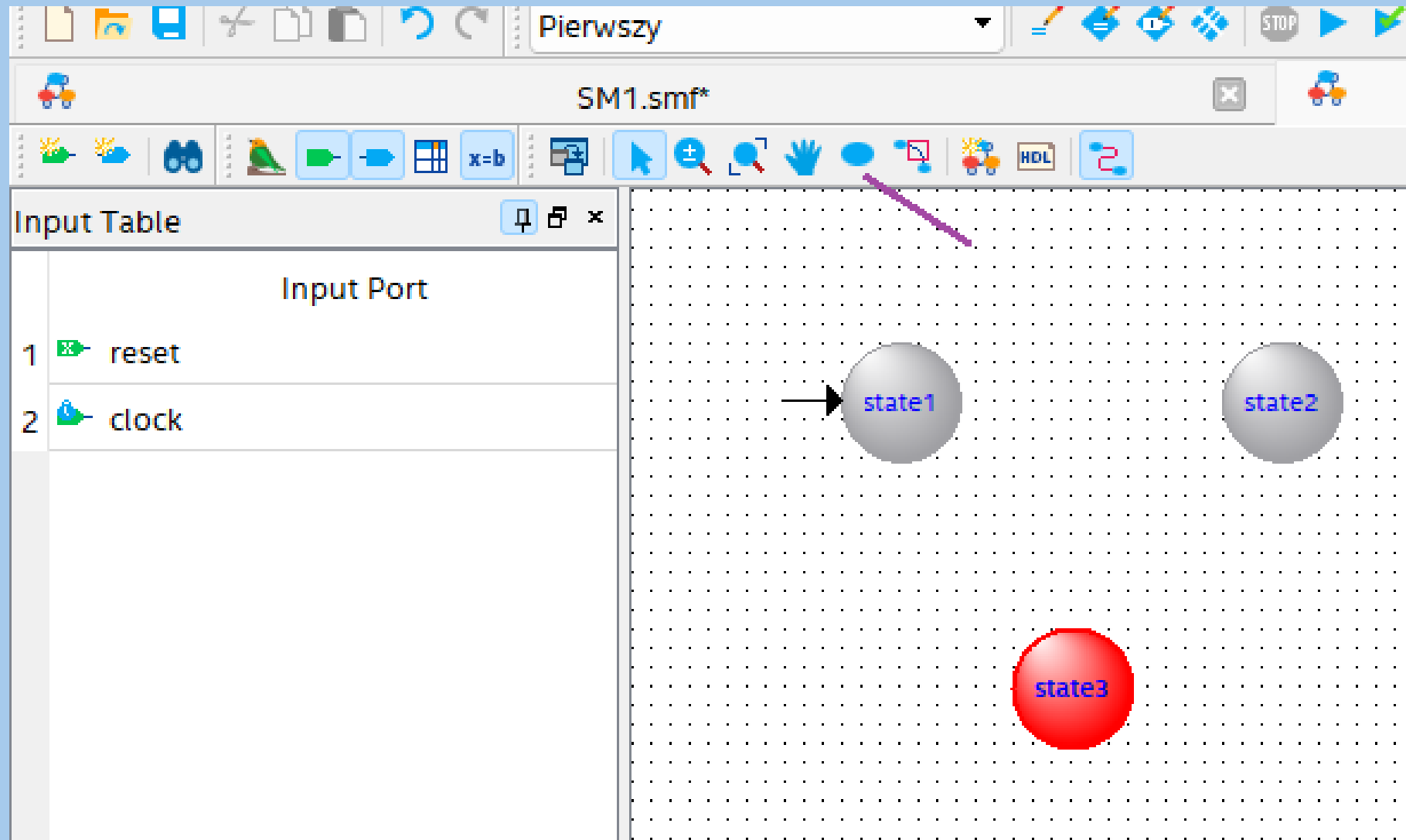
A = B = 0 licznik zlicza w przód w NKB, gdy A = 0 a B = 1 licznik liczy w tył w NKB, jeżeli A = 1 i B = 0 licznik zlicza w przód w kodzie Graya, oraz dla A = 1 i B = 1 licznik odlicza również w kodzie Graya.

AB - inputs	NKB	Rev_NKB	Gray	Rev_Gray
00 -> NKB	000	000	000	000
01 -> Rev_NKB	001	111	001	100
10 -> Gray	010	110	011	101
11 -> Rev_Gray	011	101	010	111
	100	100	110	110
	101	011	111	010
	110	010	101	011
	111	001	100	001

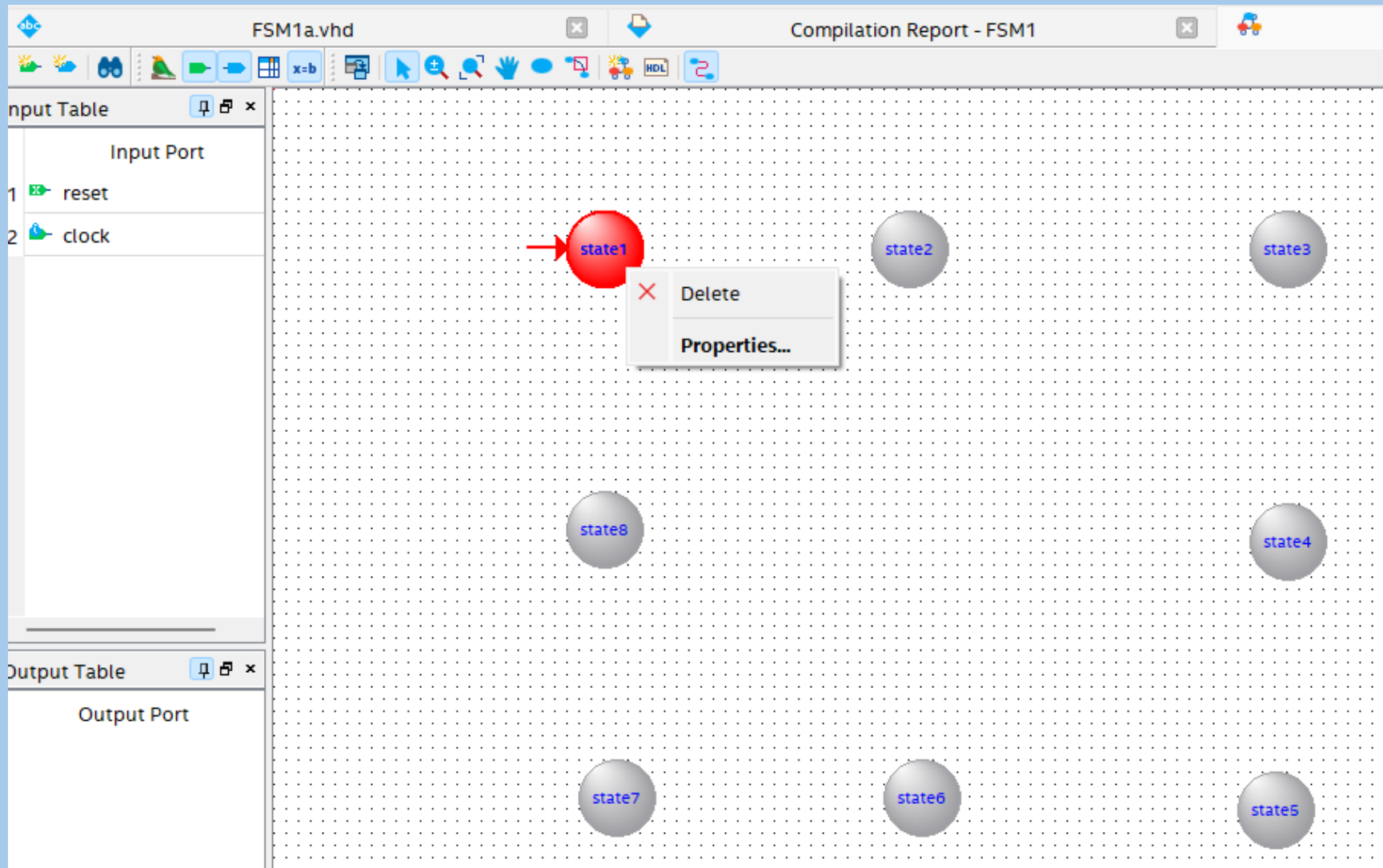
Żeby utworzyć graf układu w systemie Q II otwieramy nowy plik State Machine File:



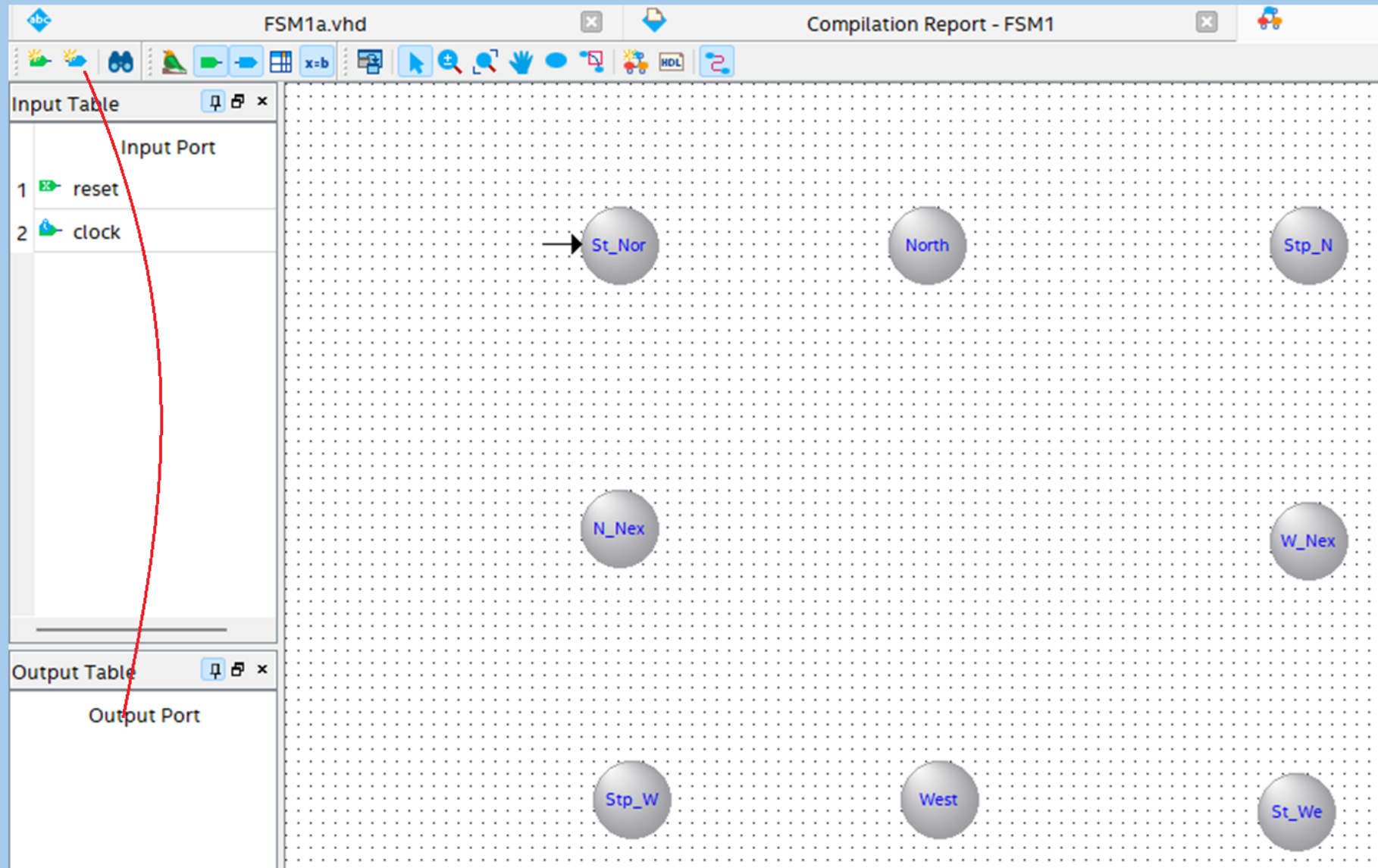
Używając górnego menu rysujemy węzły grafu:



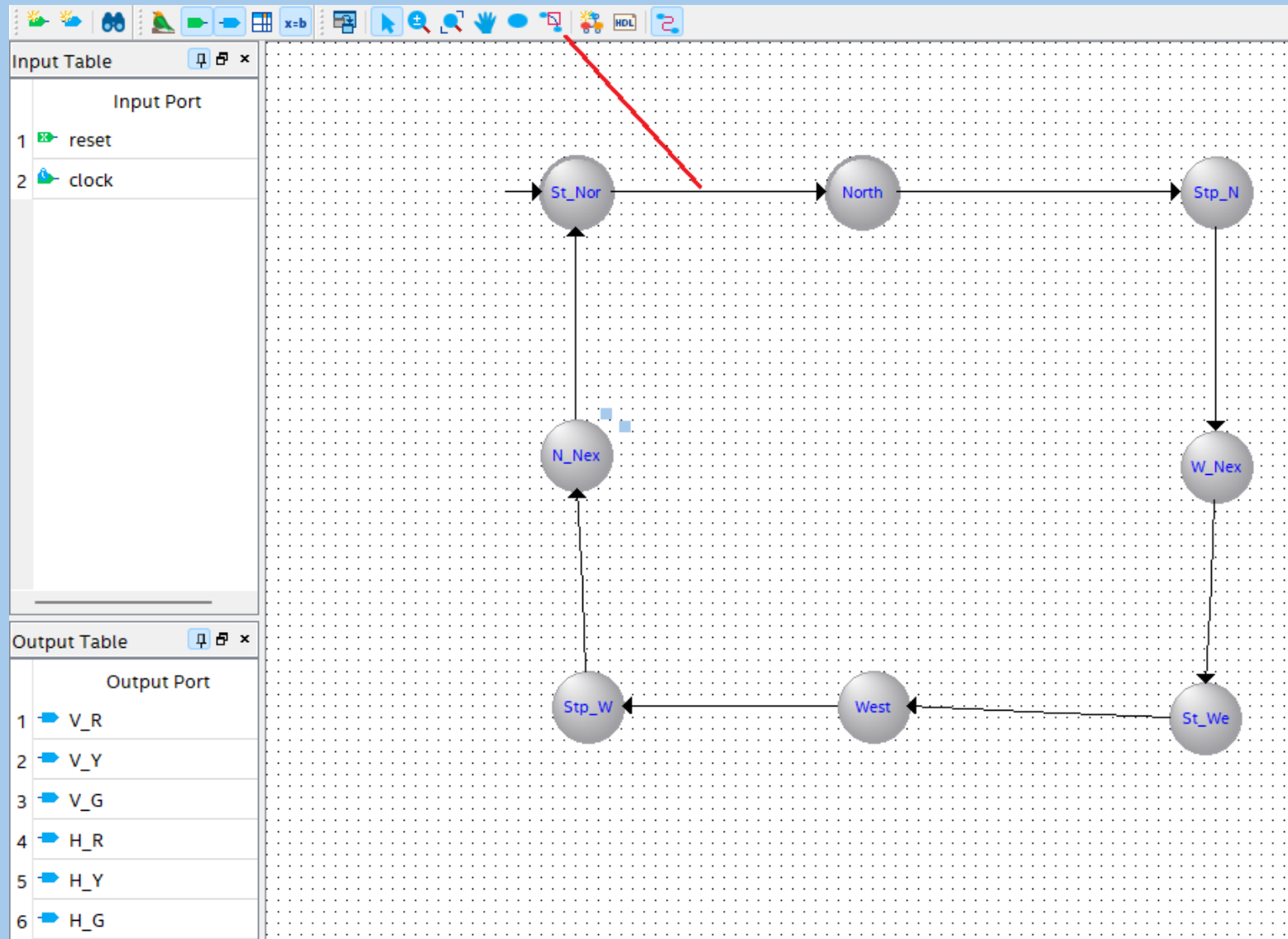
Automatycznie dodane zostaną 2 porty wejściowe: reset i clock. Zaznaczając wybrany stan (Right Click), funkcją Properties możemy zmieniać nazwy domyślne stanów/węzłów:



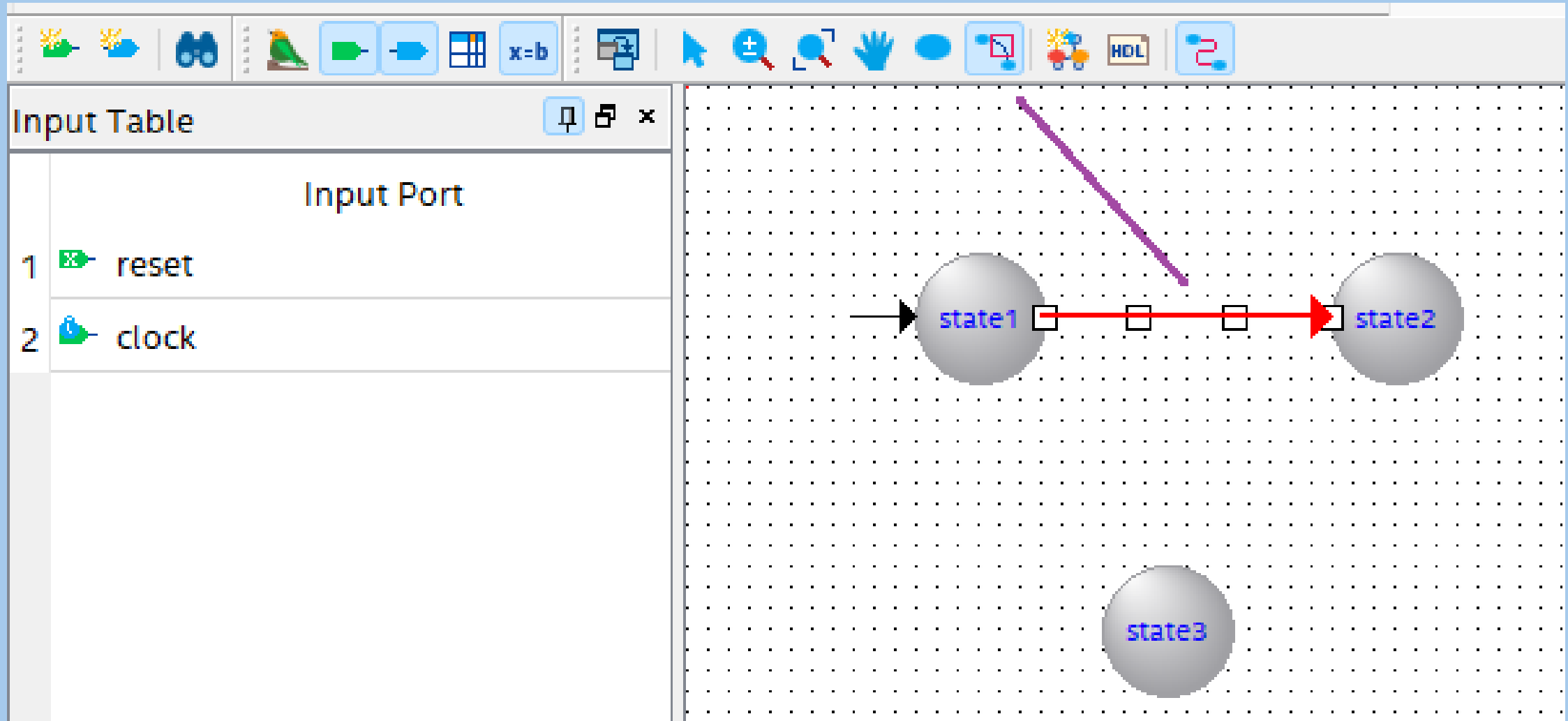
Używając górnego menu, dodajemy porty wyjściowe i wejściowe:



W podobny sposób rysujemy gałęzie w grafie (zaznaczając początek i koniec linii) :



Po ich narysowaniu, w ograniczonym stopniu wpływając na ich kształt i położenie, podajemy warunki przejścia:



Niestety warunki przejścia, ustawiane opcją Properties po Right Click wybranej gałęzi, musimy zapisać używając operatorów logicznych w języku Verilog a nie VHDL. Oznaczenie tych operatorów jest następujące:

Operatory logiczne Verilog

~ - NOT,

& - AND,

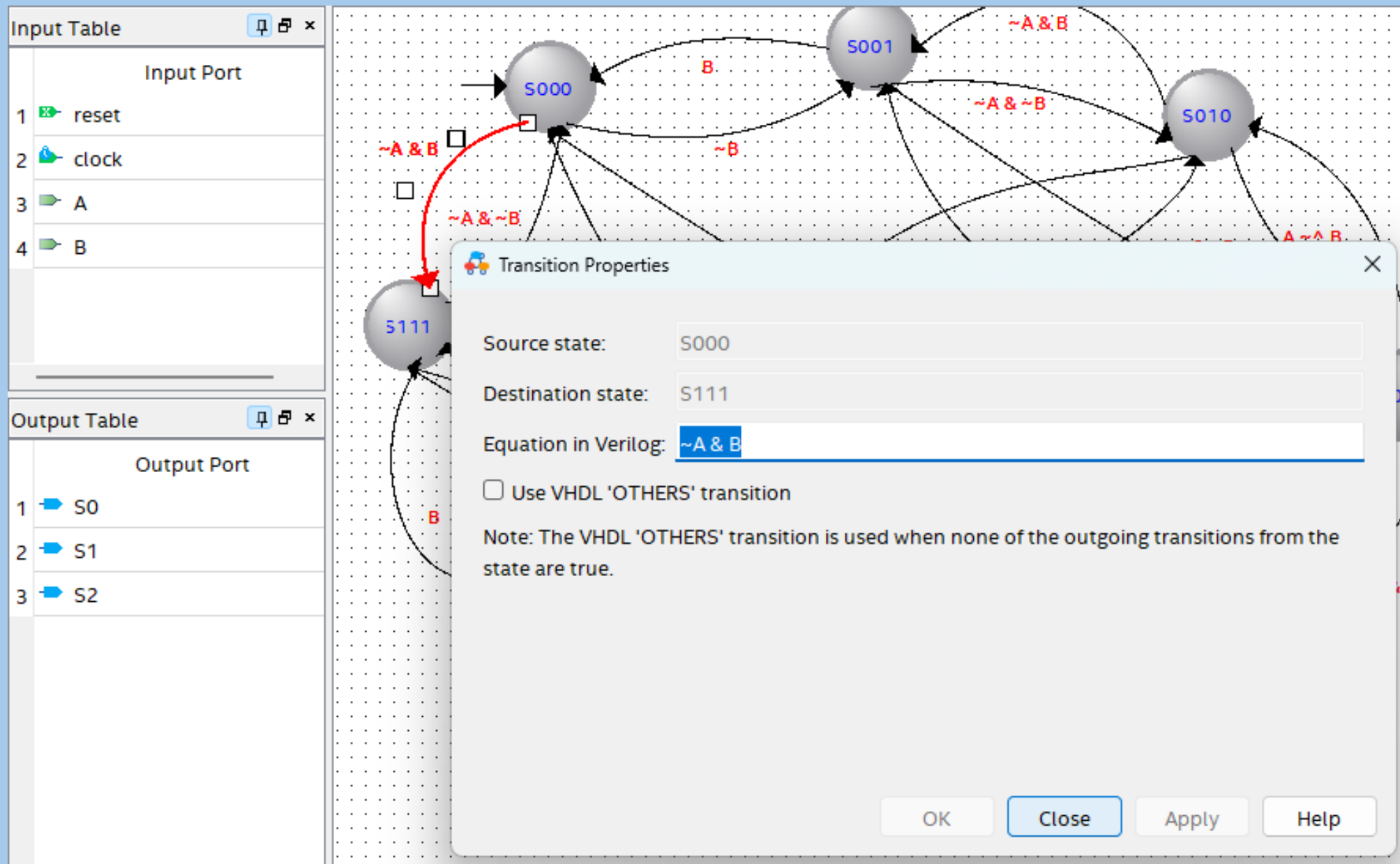
~& - NAND,

| - OR (to nie jest duże i),

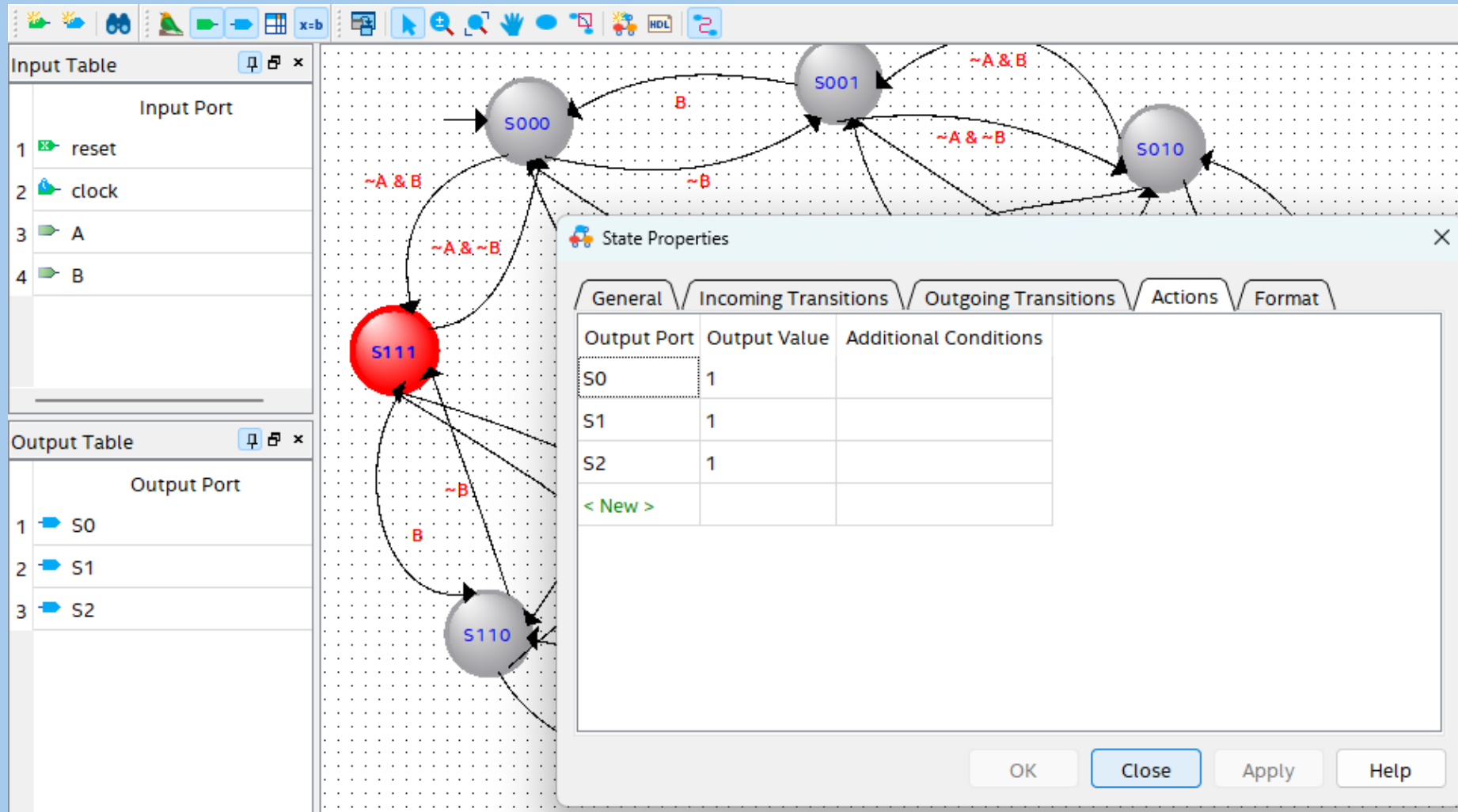
~| - NOR,

^ - XOR,

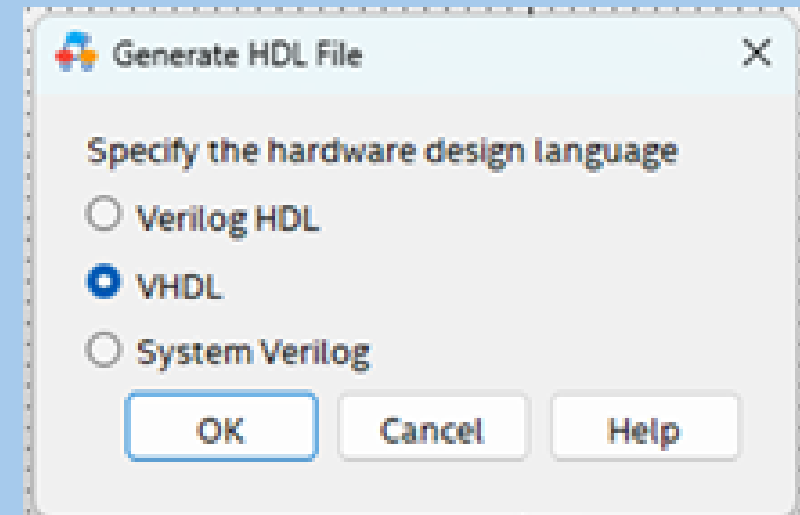
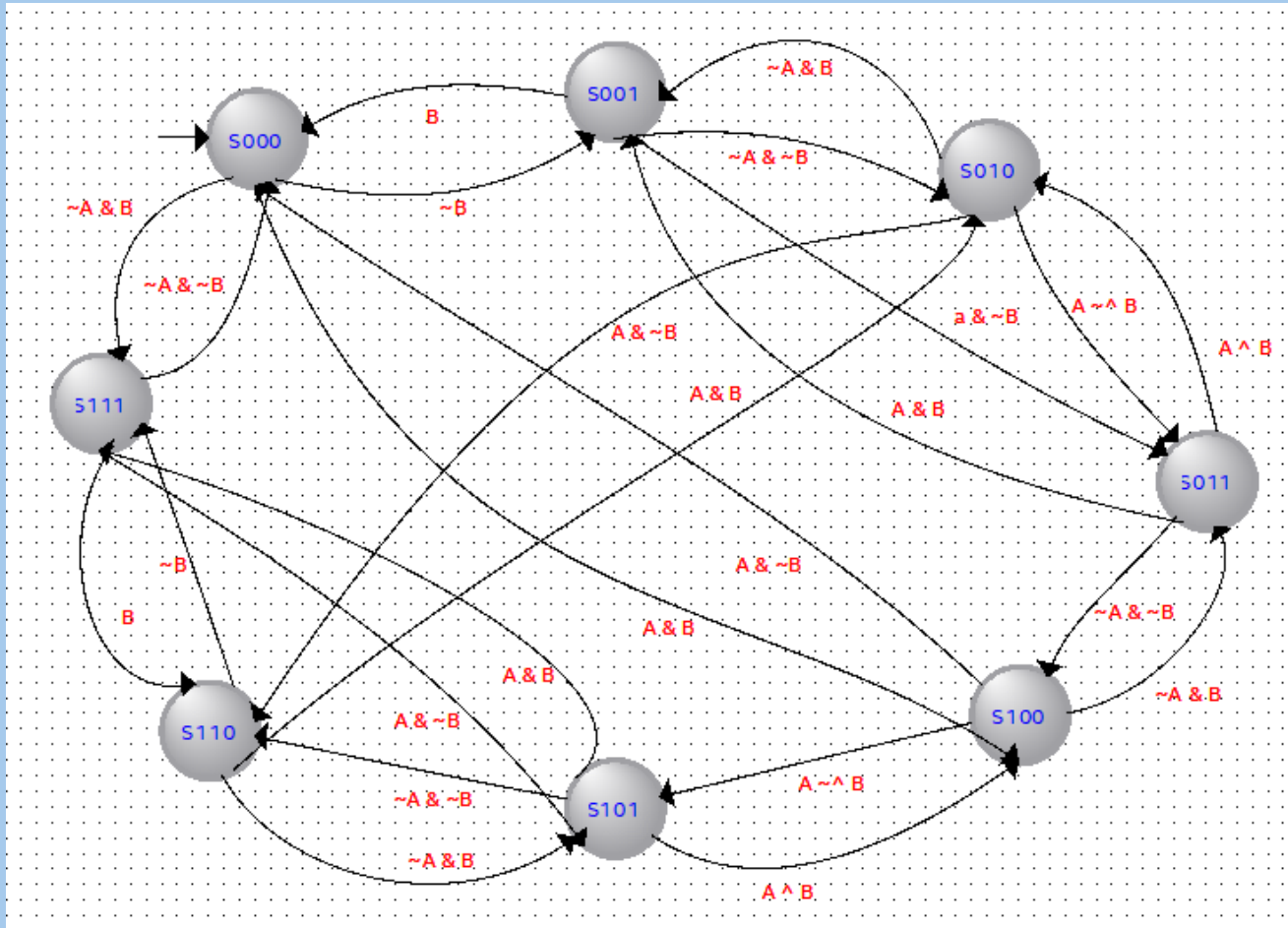
~^ - XNOR.



Po skompletowaniu grafu i wartości sygnałów wejściowych i warunków przejść - w każdym ze stanów, po jego zaznaczeniu ustawiamy wartości sygnałów wyjściowych:



Gdy graf jest kompletnie określony uruchamiamy górną funkcję HDL i automatycznie generujemy kompletny kod opisujący nasz układ. Wyniki automatycznego wytwarza kodu FSM pokazują dwa ostatnie slajdy.



```
1  -- Copyright (C) 2018 Intel Corporation. All rights reserved.
2  -- Your use of Intel Corporation's design tools, logic functions
3  -- and other software and tools, and its AMPP partner logic
4  -- functions, and any output files from any of the foregoing
5  -- (including device programming or simulation files), and any
6  -- associated documentation or information are expressly subject
7  -- to the terms and conditions of the Intel Program License
8  -- Subscription Agreement, the Intel Quartus Prime License Agreement,
9  -- the Intel FPGA IP License Agreement, or other applicable license
10 -- agreement, including, without limitation, that your use is for
11 -- the sole purpose of programming logic devices manufactured by
12 -- Intel and sold by Intel or its authorized distributors. Please
13 -- refer to the applicable agreement for further details.
14
15 -- Generated by Quartus Prime Version 18.1.0 Build 625 09/12/2018 SJ Lite Edition
16 -- Created on Sat Dec 14 23:00:58 2024
17
18 LIBRARY ieee;
19 USE ieee.std_logic_1164.all;
20
21 ENTITY SM3 IS
22 PORT (
23     reset : IN STD_LOGIC := '0';
24     clock  : IN STD_LOGIC;
25     A      : IN STD_LOGIC := '0';
26     B      : IN STD_LOGIC := '0';
27     S0     : OUT STD_LOGIC;
28     S1     : OUT STD_LOGIC;
29     S2     : OUT STD_LOGIC;
30 );
31 END SM3;
32
33 ARCHITECTURE BEHAVIOR OF SM3 IS
34     TYPE type_fstate IS (S000,S001,S010,S011,S100,S101,S110,S111);
35     SIGNAL fstate : type_fstate;
36     SIGNAL reg_fstate : type_fstate;
37 BEGIN
38     PROCESS (clock,reset,reg_fstate)
39     BEGIN
```



```
39 BEGIN
40     IF (reset='1') THEN
41         fstate <= s000;
42     ELSIF (clock='1' AND clock'event) THEN
43         fstate <= reg_fstate;
44     END IF;
45 END PROCESS;
46
47 PROCESS (fstate,A,B)
48 BEGIN
49     S0 <= '0';
50     S1 <= '0';
51     S2 <= '0';
52     CASE fstate IS
53     WHEN S000 =>
54         IF ((NOT((A = '1')) AND (B = '1')))) THEN
55             reg_fstate <= S111;
56         ELSIF (NOT((B = '1')))) THEN
57             reg_fstate <= S001;
58         ELSIF (((A = '1') AND (B = '1')))) THEN
59             reg_fstate <= S100;
60         -- Inserting 'else' block to prevent latch inference
61         ELSE
62             reg_fstate <= S000;
63         END IF;
64
65         S2 <= '0';
66
67         S1 <= '0';
68
69         S0 <= '0';
70     WHEN S001 =>
71         IF ((B = '1')) THEN
72             reg_fstate <= S000;
73         ELSIF ((NOT((A = '1')) AND NOT((B = '1')))) THEN
74             reg_fstate <= S010;
75         ELSIF (((A = '1') AND NOT((B = '1')))) THEN
76             reg_fstate <= S011;
77         -- Inserting 'else' block to prevent latch inference
78         ELSE
79             reg_fstate <= S001;
80         END IF;
```

Zaliczenie cząstkowe wykładu nr 13 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań.

Każde zadanie posiada 3 odpowiedzi.

Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c).

Zadania będą wyświetlane po kolei co 1 minutę.

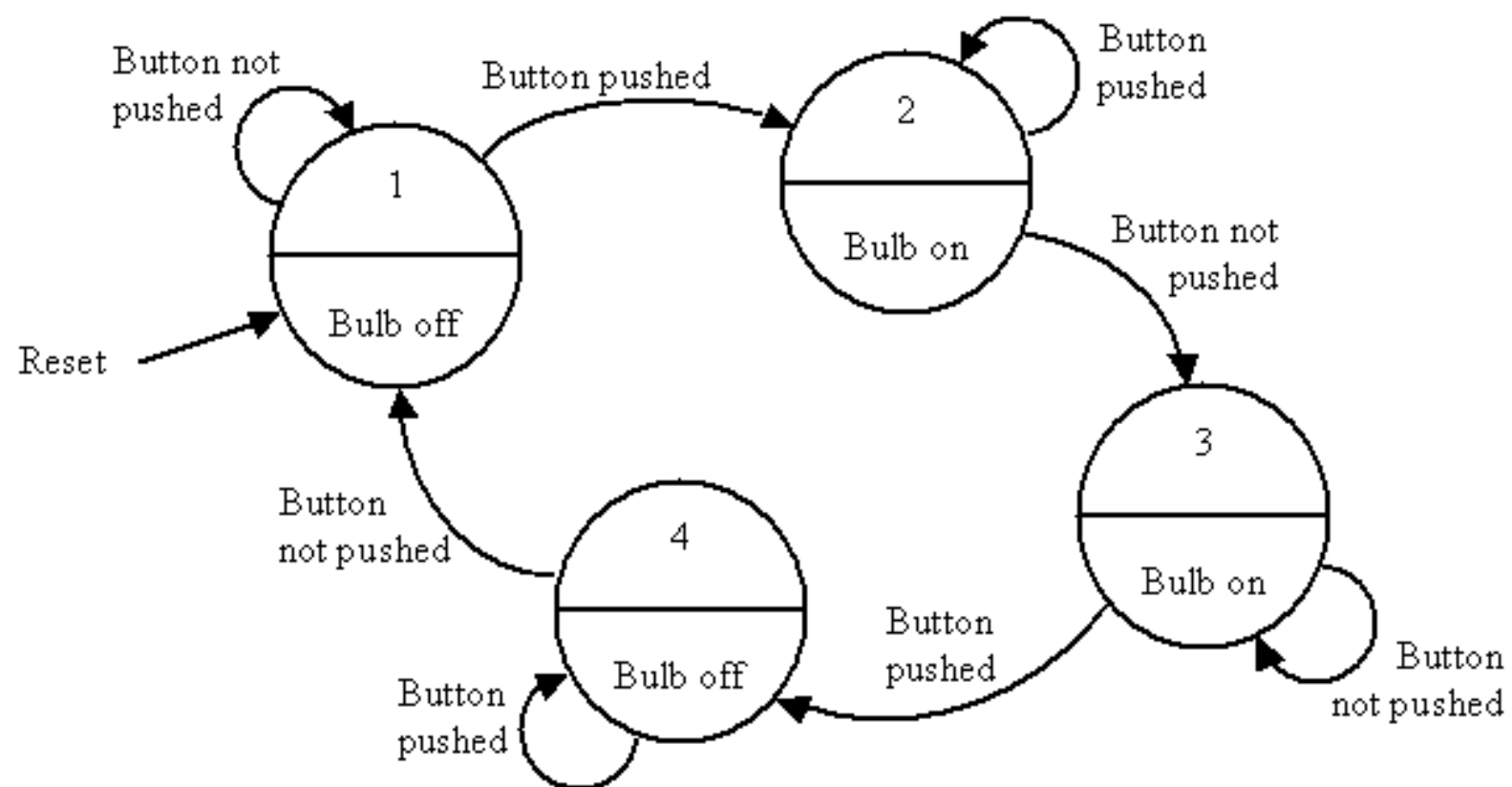
Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym.

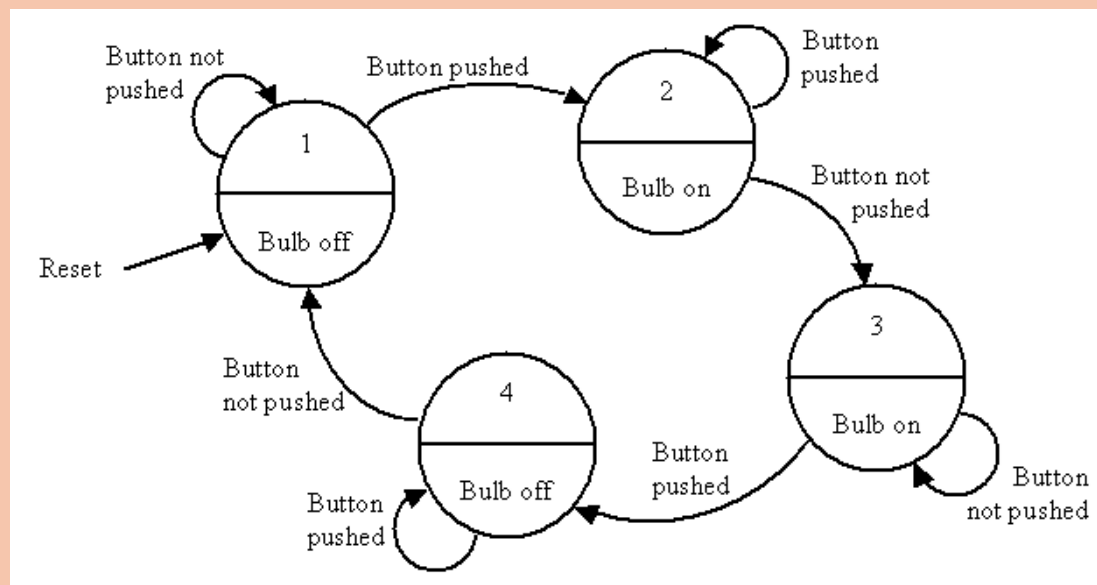
Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia

Wyobraźmy sobie obwód żarówki sterowany przyciskiem. Po naciśnięciu przycisku (Button pushed) żarówka zaświeca się (Bulb on). Po zwolnieniu przycisku żarówka pozostaje włączona (świeci). Jeżeli nacisnąć przycisk ponownie żarówka zgaśnie/wyłączy (Bulb off). Po zwolnieniu przycisku żarówka pozostaje wyłączona. Mamy zatem następstwo czynności: przyciśnij – zwolnij – przyciśnij – zwolnij = zaświeć – wyłączyć.





Mówimy, że mamy 4 stany:

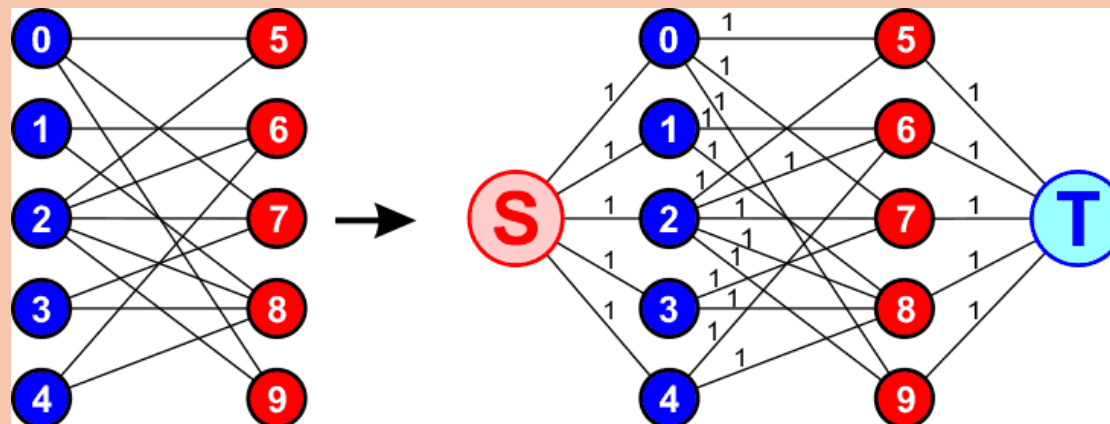
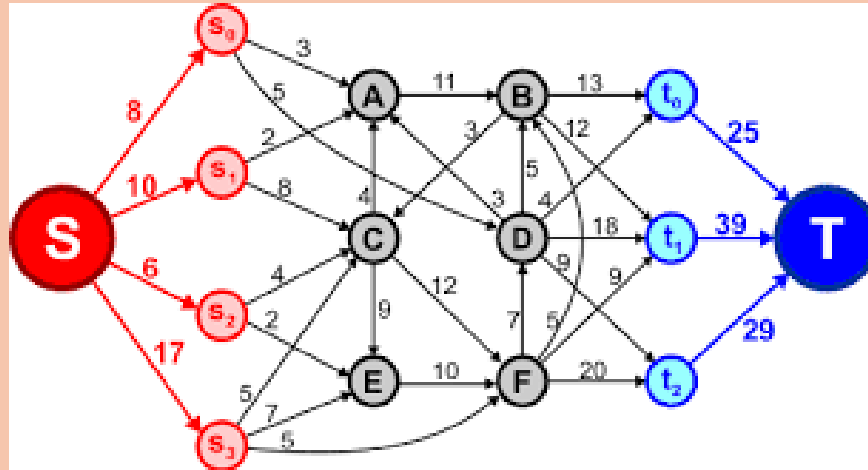
STAN 1 -- żarówka jest wyłączona i czeka na naciśnięcie przycisku w celu jej włączenia. Dopóki przycisk pozostaje zwolniony, system pozostaje w tym stanie. Po naciśnięciu przycisku system przechodzi do stanu 2.

STAN 2 -- Ten stan następuje, gdy użytkownik nacisnął przycisk w celu włączenia światła, światło zostało włączone, a system oczekuje na zwolnienie przycisku przez użytkownika. Zwolnienie przycisku powoduje przejście systemu do stanu 3.

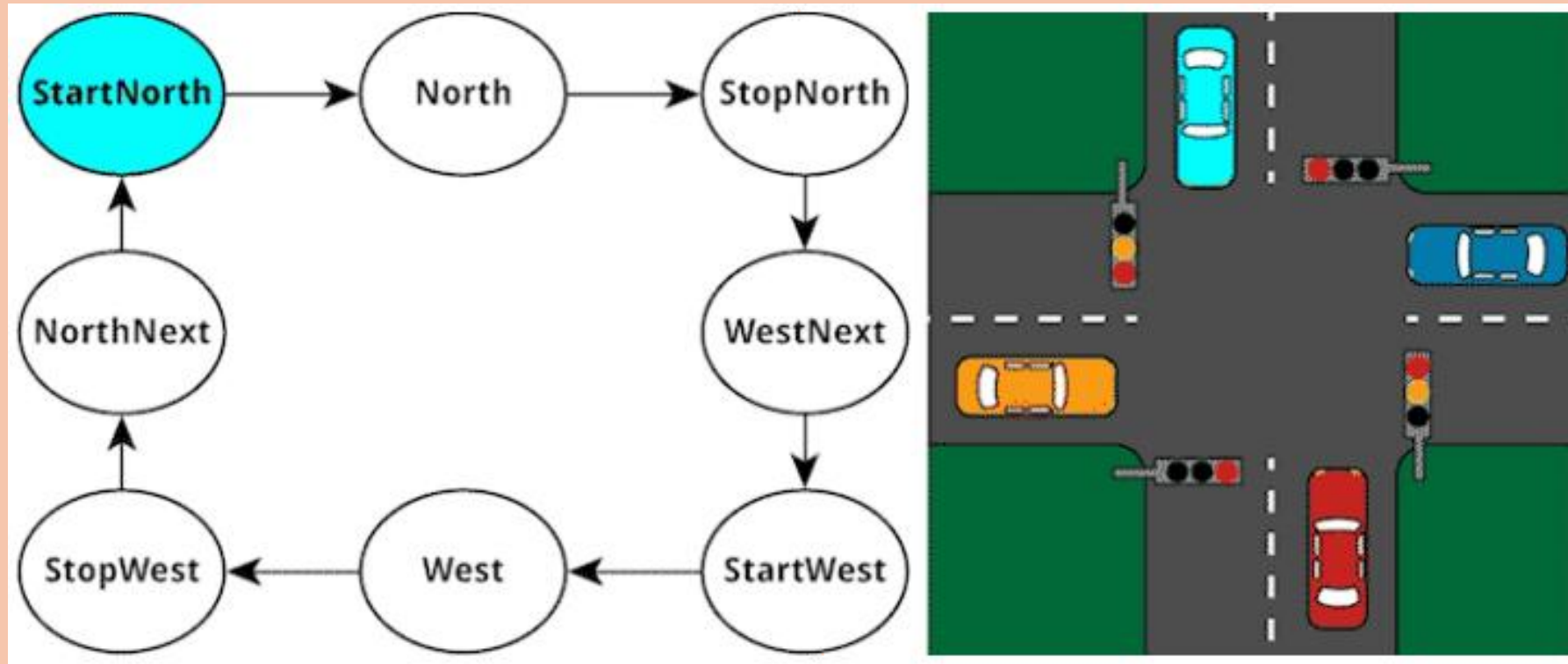
STAN 3 -- Ten stan jest pojawia się, gdy żarówka jest włączona, a przycisk jest zwolniony. Jedyną rzeczą, która wyprowadza nas z tego stanu, jest naciśnięcie przycisku przez użytkownika w celu wyłączenia żarówki. Po naciśnięciu przycisku system przechodzi do stanu 4.

STAN 4 -- Ten stan jest wprowadzany, gdy użytkownik nacisnął przycisk, aby wyłączyć światło, światło zgasło, a system czeka, aż użytkownik zwolni przycisk. Zwolnienie przycisku powoduje powrót systemu do stanu 1.

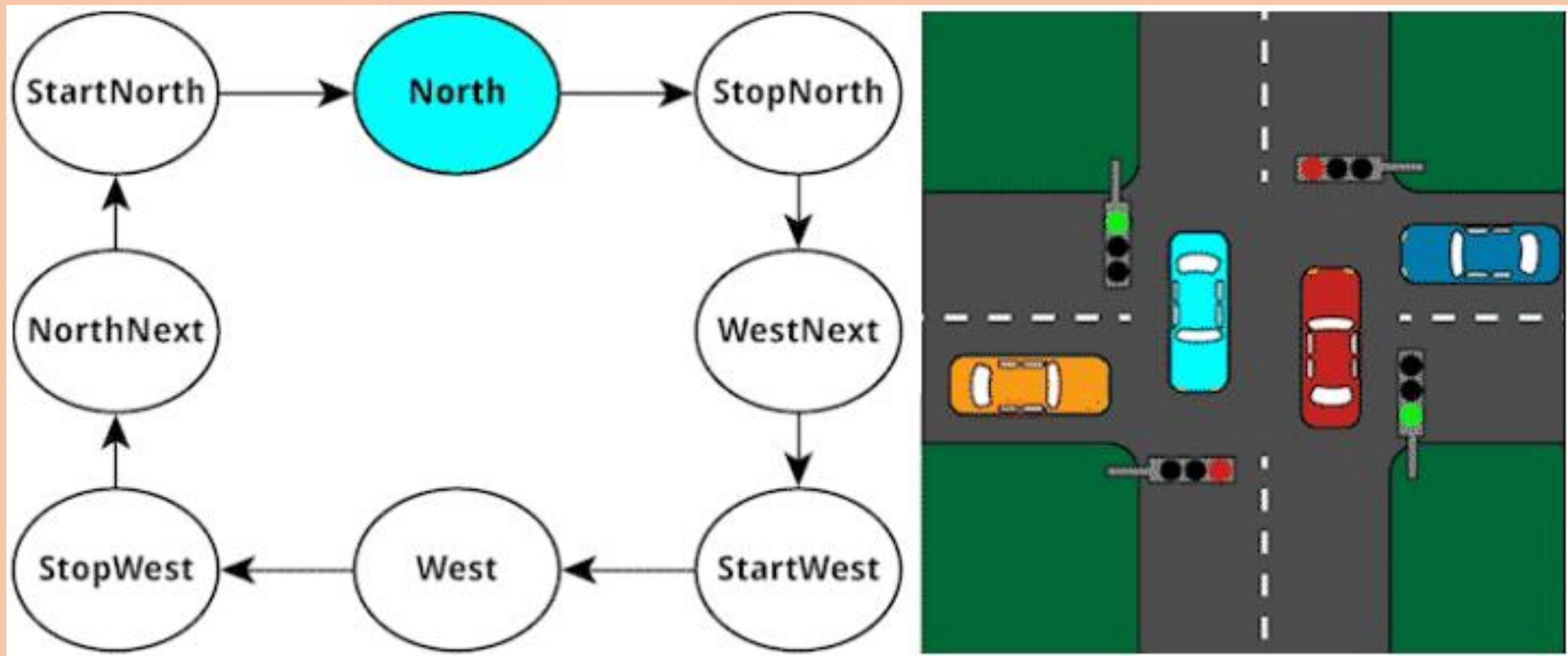
Graf opisujący może być bardzo złożony co nie oznacza, że projektowanie układów sekwencyjnych jest bardzo skomplikowane. Graf jest formą opisu pracy rzeczywistego układu – stąd pośredniczy w układzie projektant – system komputerowy.



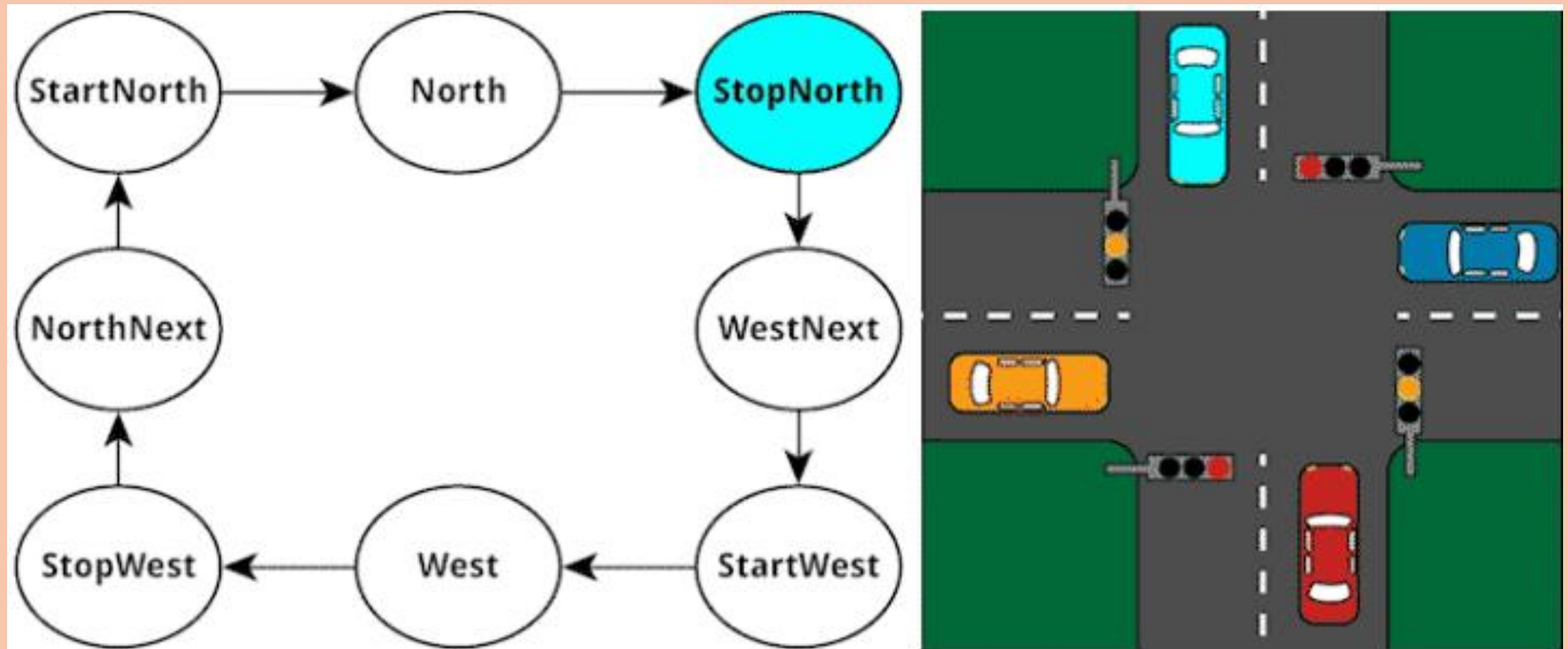
Przykład:



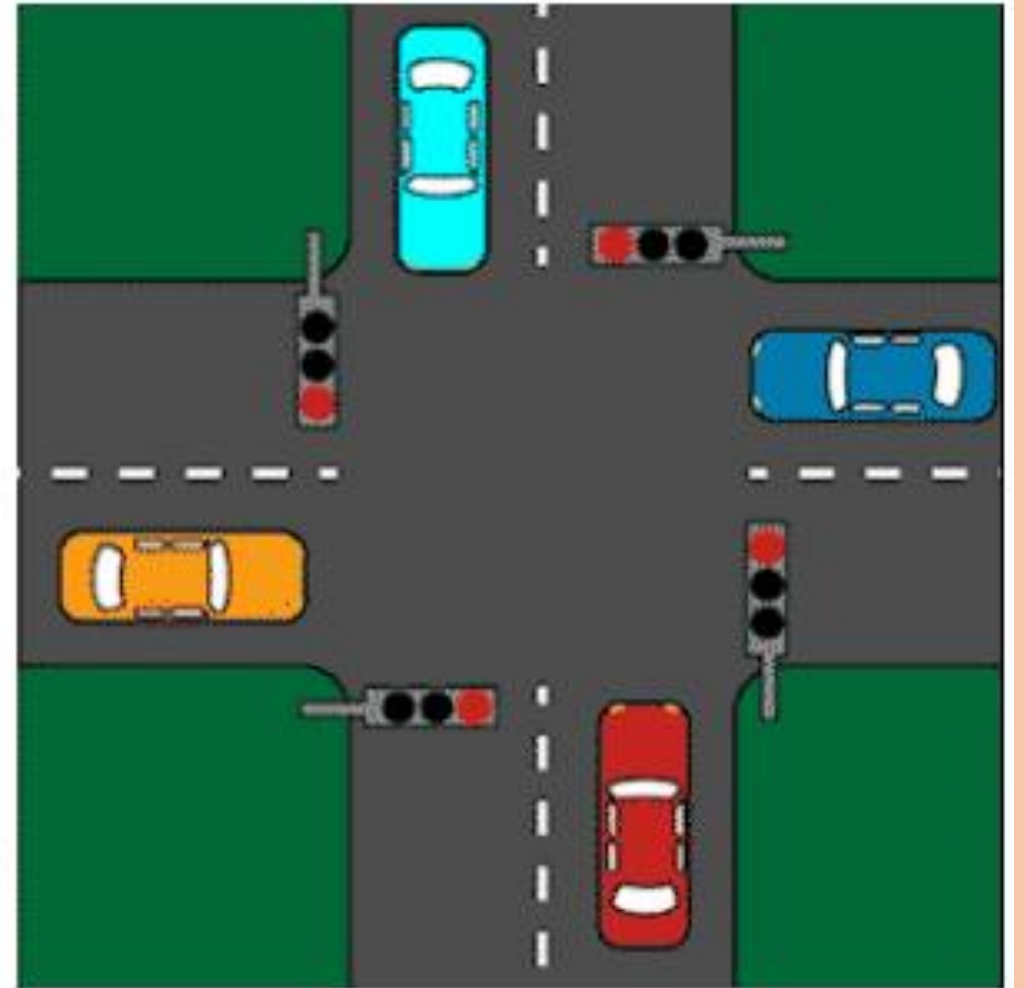
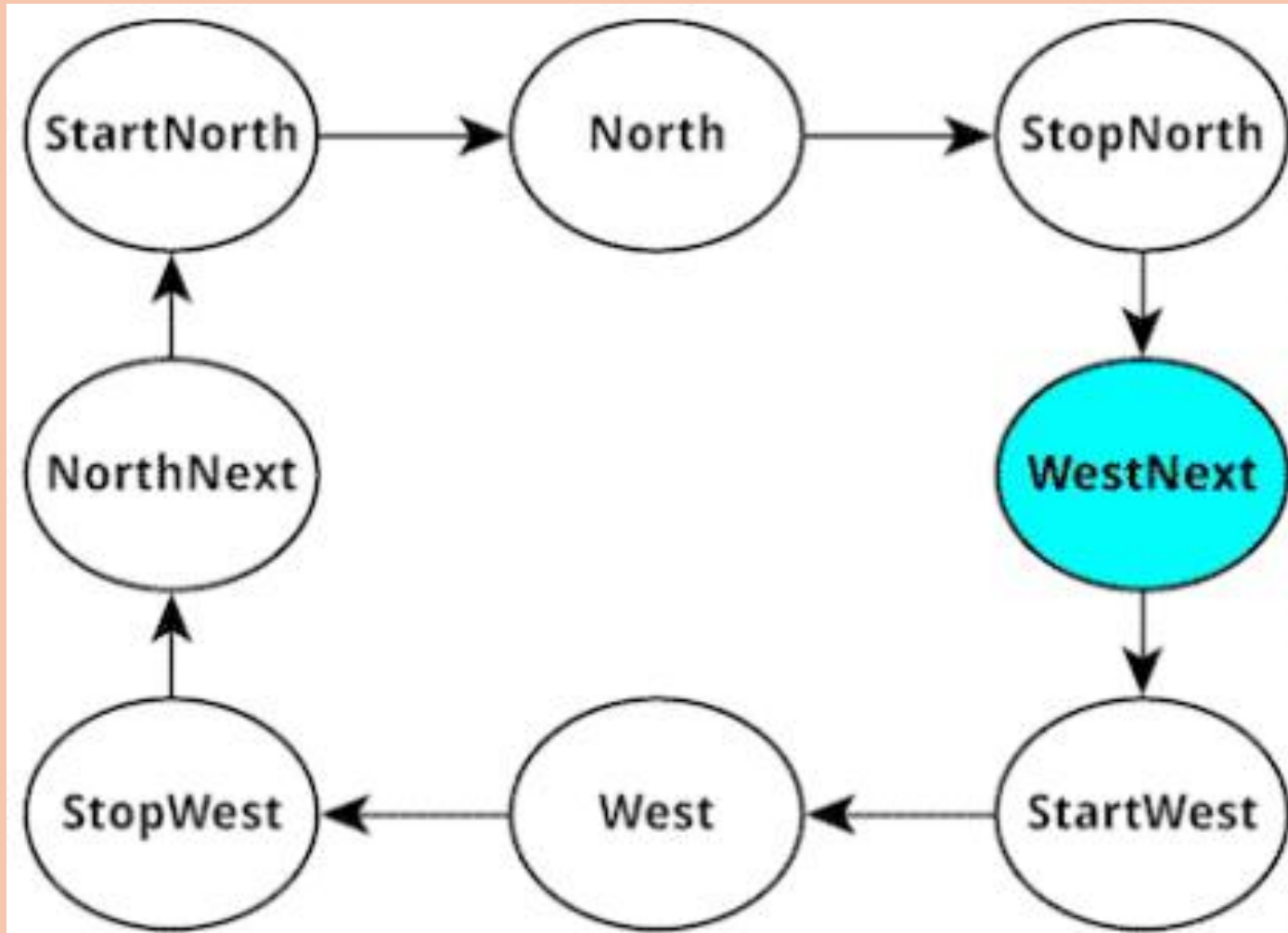
Sygnalizacja świetlna, którą opisujemy ma skończoną liczbę stanów równą 8, którym nadano rozpoznawalne nazwy. Ta przykładowa maszyna stanów nie ma wejść sterujących, wyjściami jest stan świateł w kierunkach północ/południe i zachód/wschód. Czynnikiem sterującym jest to czas, w tym przypadku okres sygnału zegarowego. Załóżmy, że stanem początkowym jest **StartNorth** przygotowanie przed rozpoczęciem jazdy na kierunku północnym.



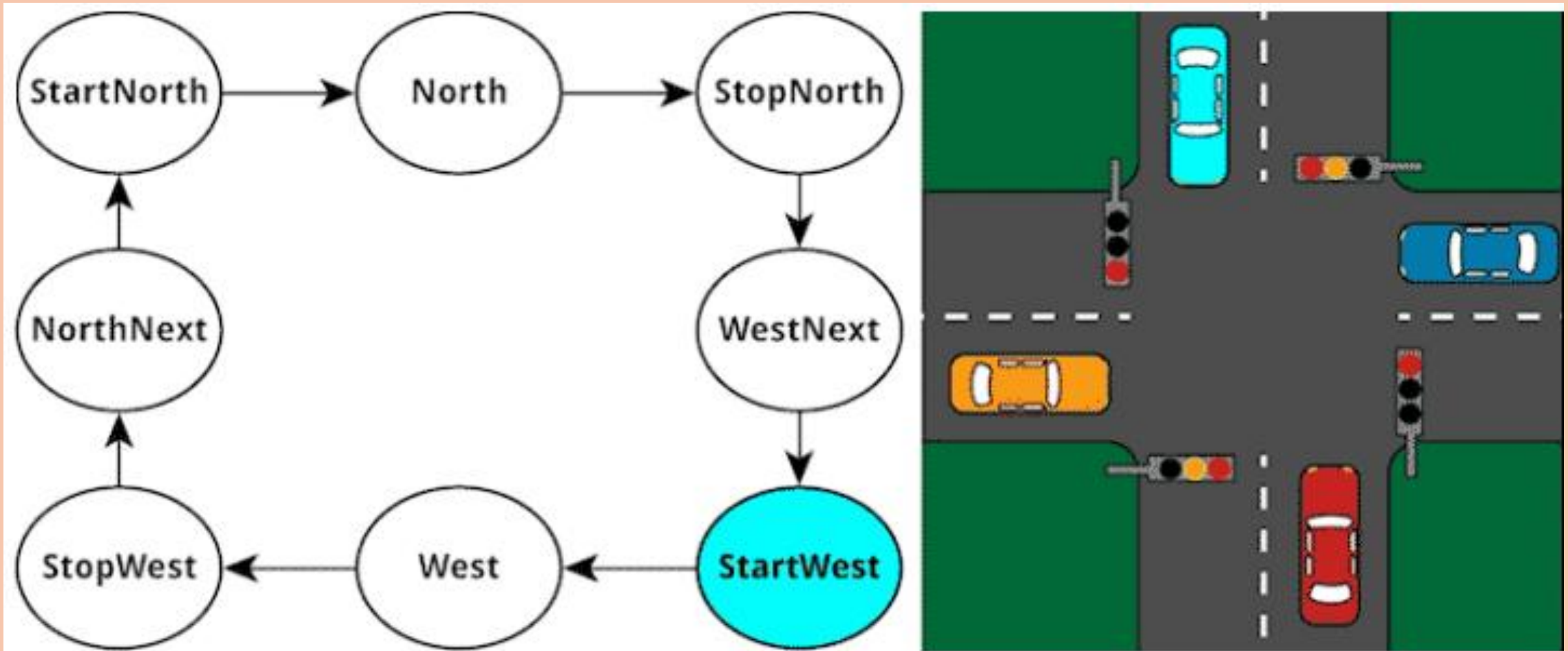
Ze stanu StartNorth układ przechodzi do stanu **North**, gdzie świecą światła zielone dla kierunku pionowego.



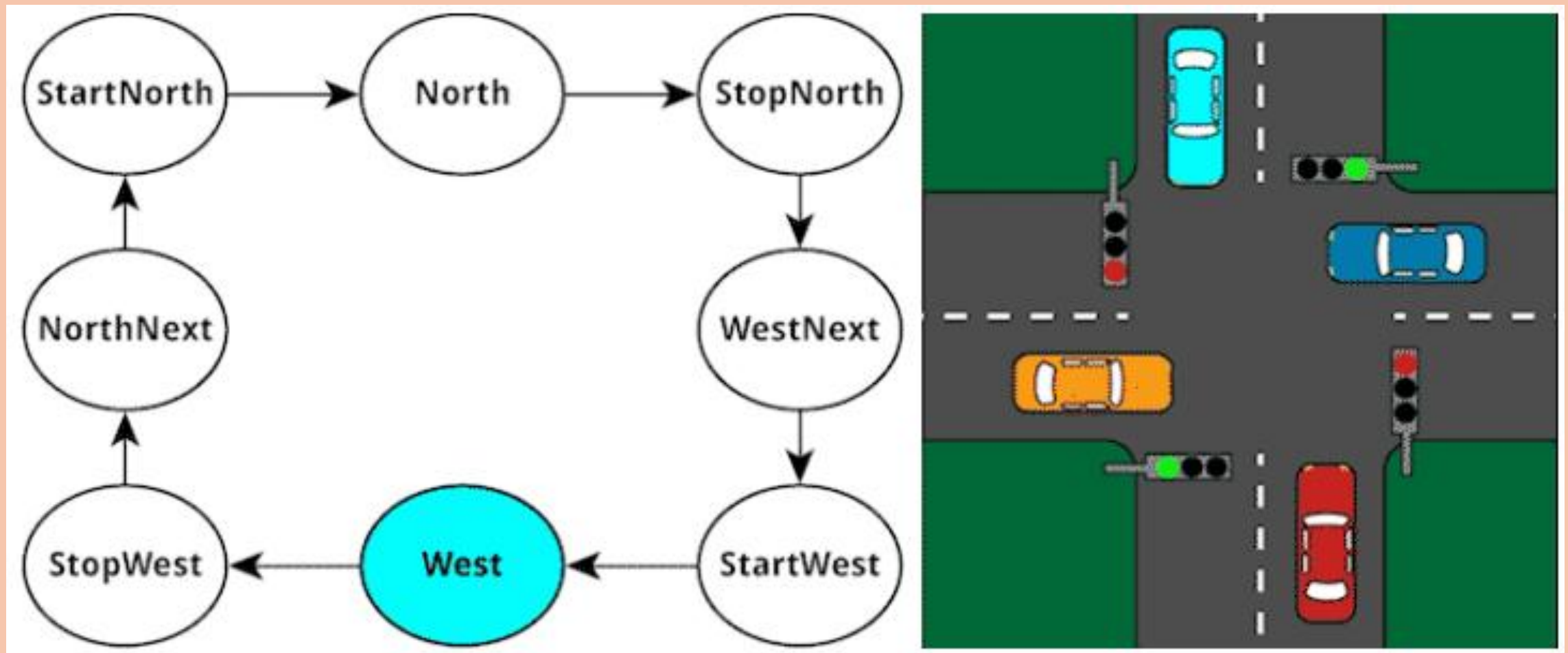
Po stanie Nort, układ przechodzi do stanu **StopNort**, gdzie świecą światła żółte dla kierunku pionowego.



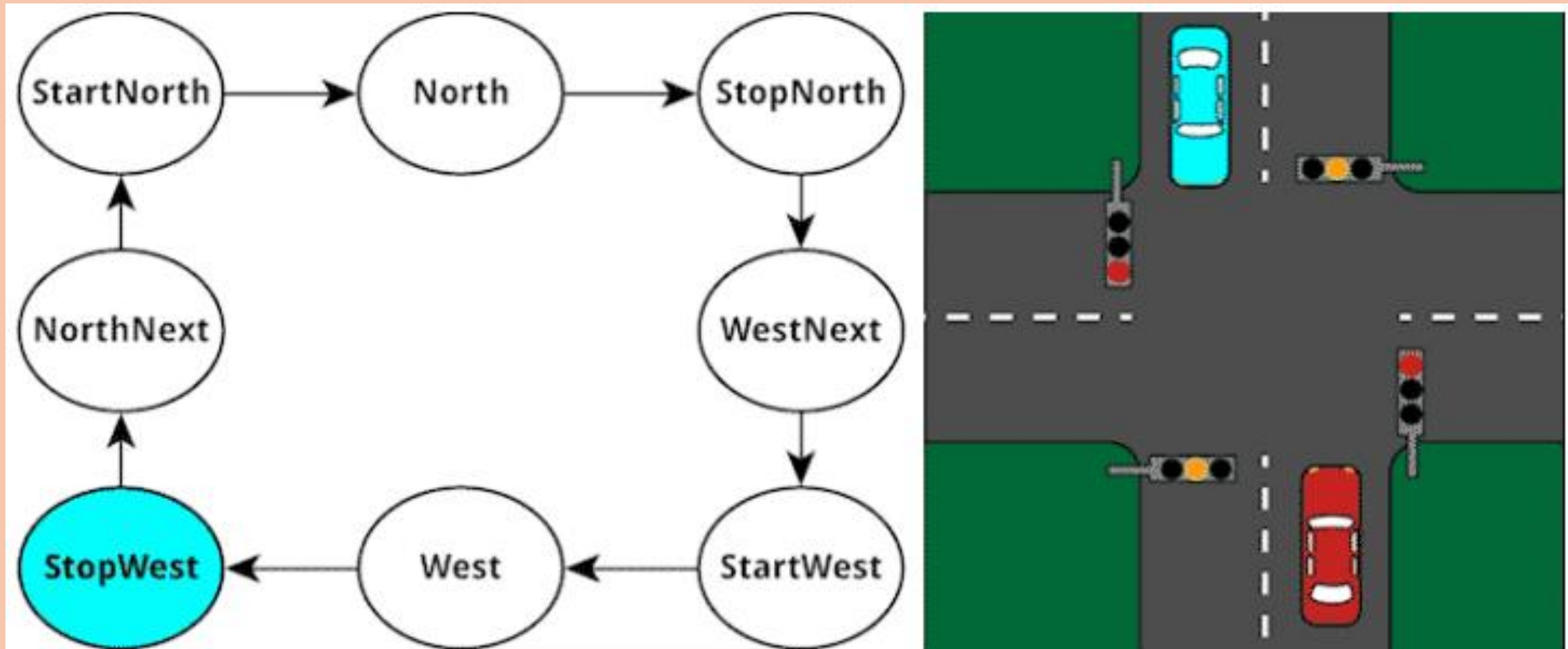
Po stanie StopNort, układ przez chwilę pozostaje w stanie **WestNext**, gdzie we wszystkich kierunkach świecą światła czerwone.



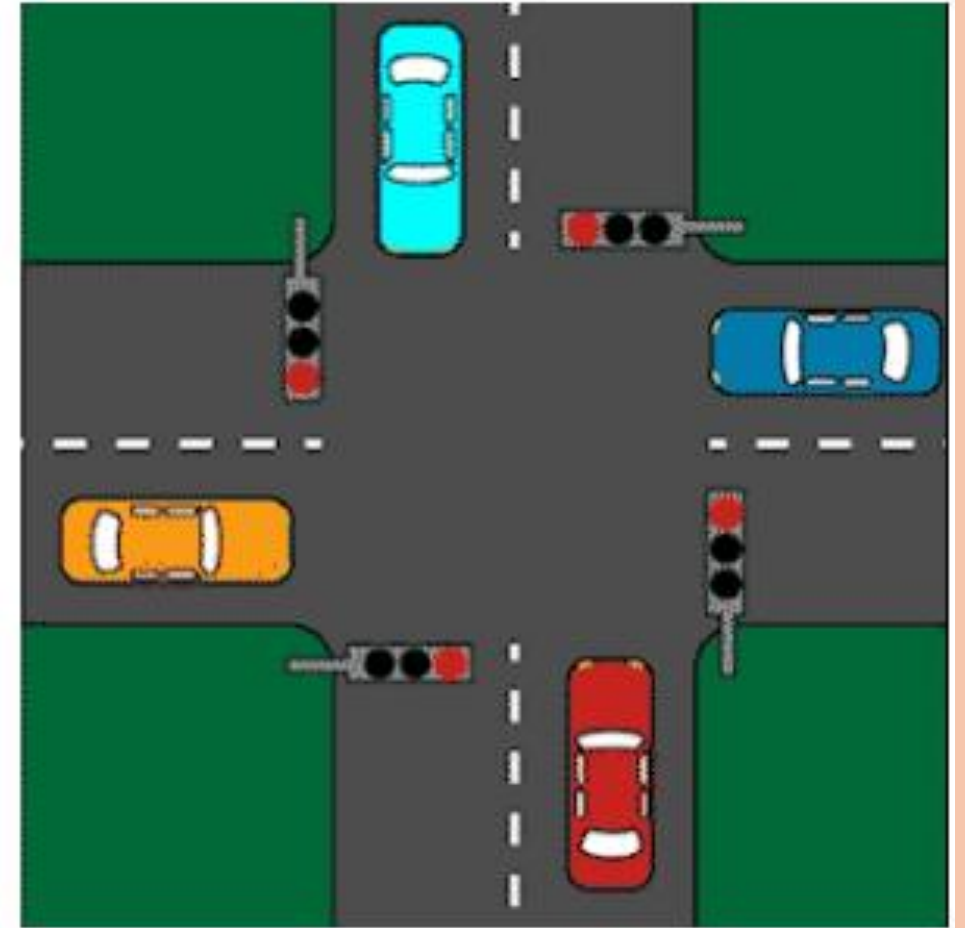
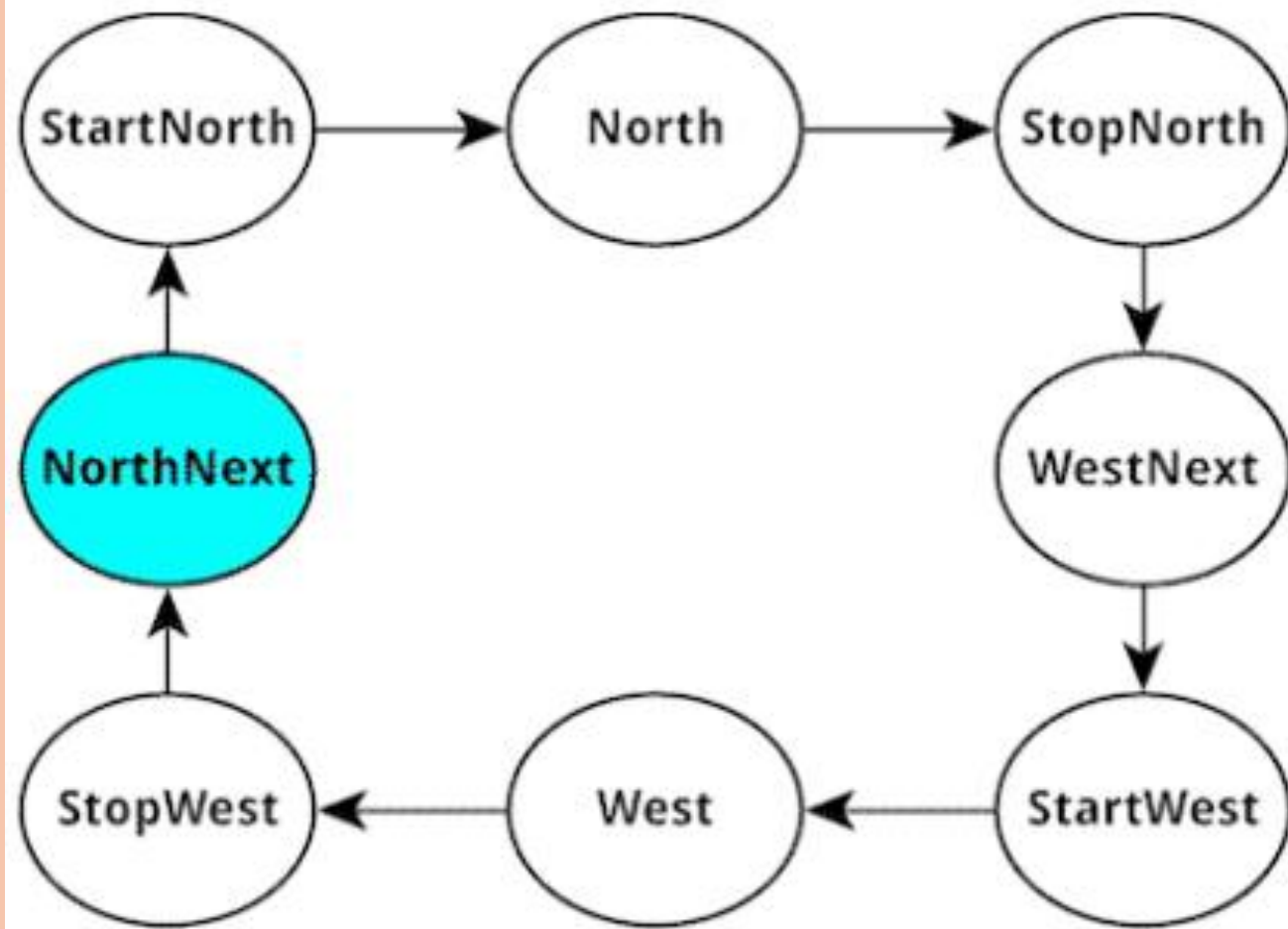
Po stanie WestNext, występuje stan **StartWest**, który przygotowuje do jazdy w kierunku poziomym.



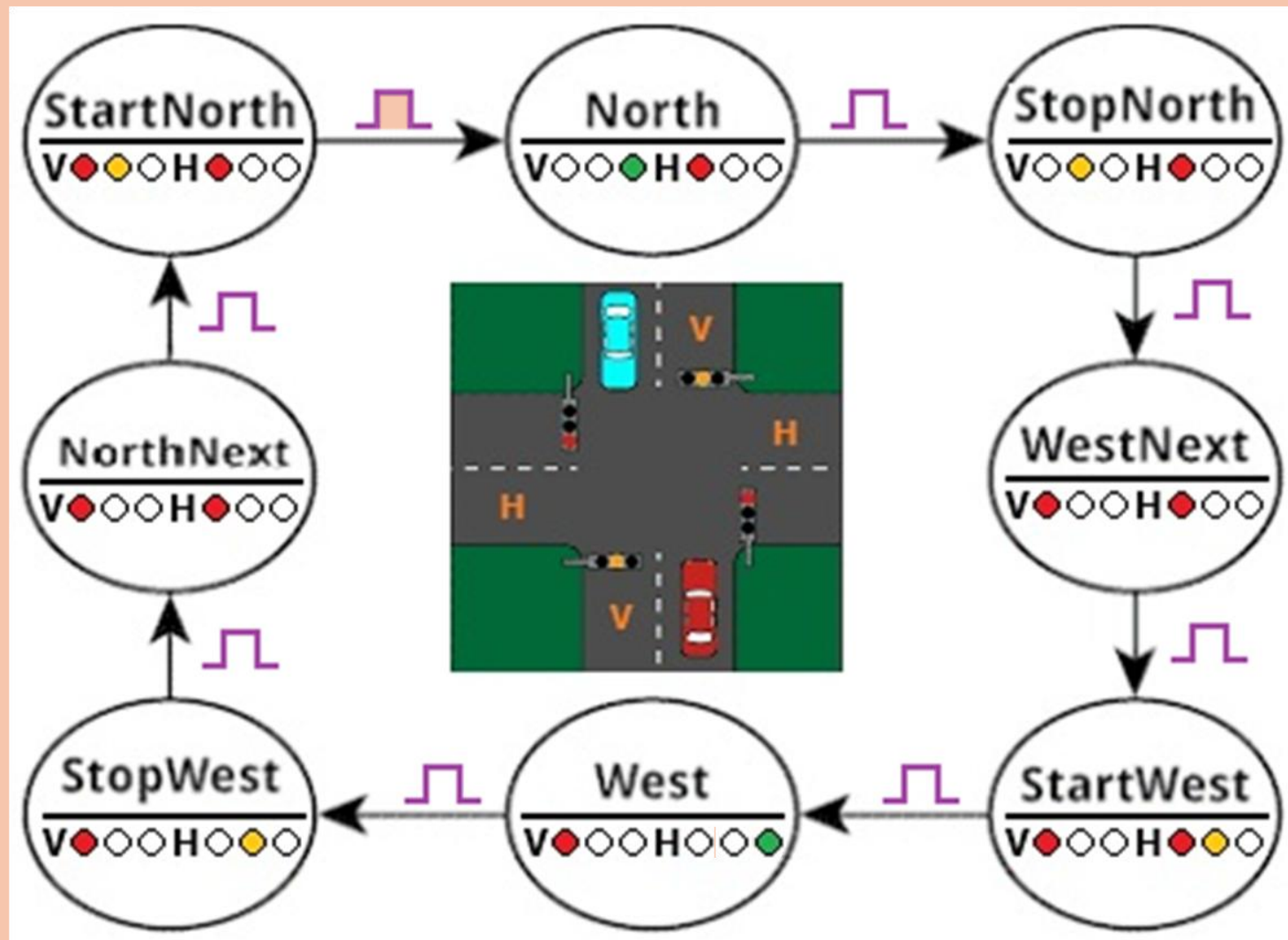
Po stanie StartWest następuje stan **West** z zielonymi światłami dla jazdy w kierunkach poziomych.



Jazdę w kierunku poziomym kończy żółte światło w stanie **StopWest**

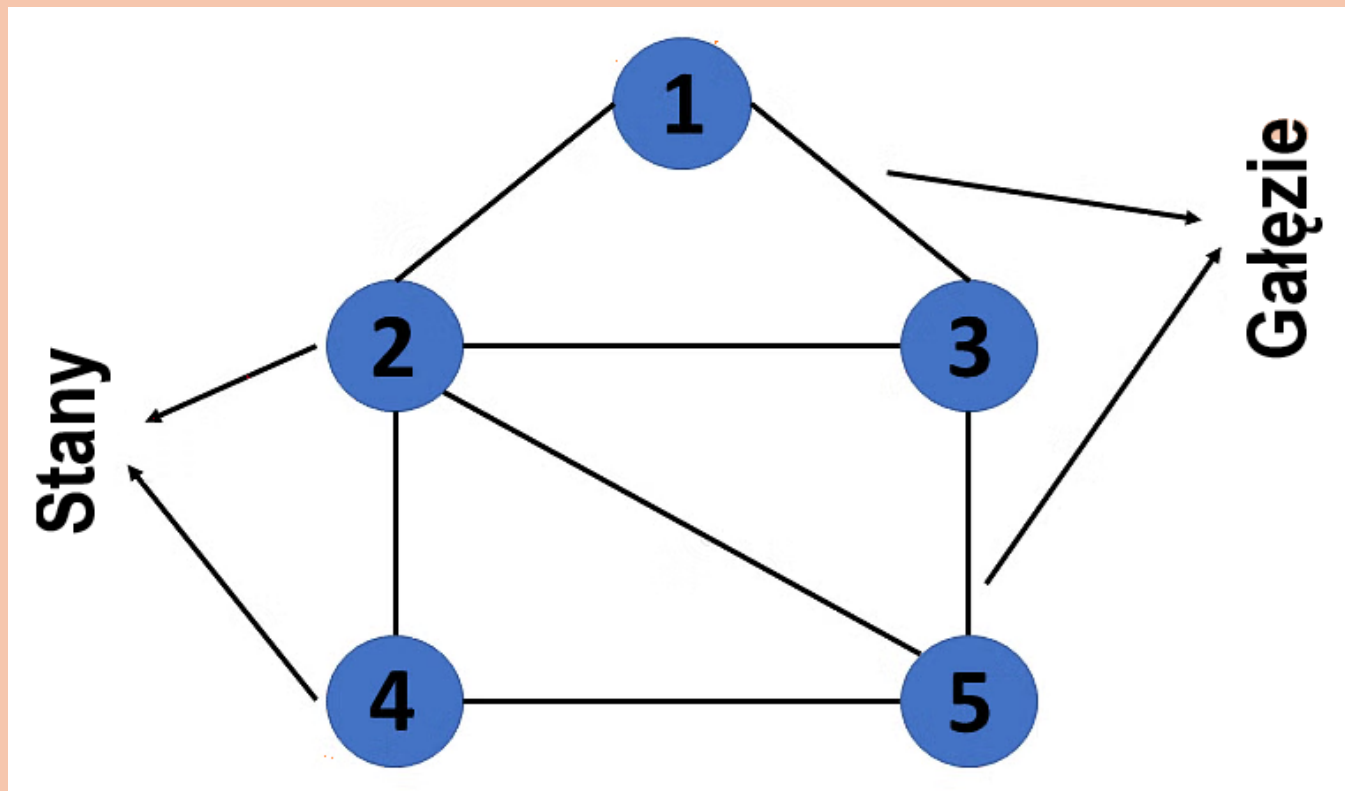


Po stanie StopWest następuje stan **NortWest** po czym układ przechodzi do stanu początkowego **StartNorth**.



Podsumowując: układ posiada 8 stanów, przejście pomiędzy stanami synchronizuje sygnał zegarowy, w układzie występuje 6 wyjść/światel – gdyż następuje zdublowanie lamp dla kierunków poziomych H i pionowych L.

Graf w układzie cyfrowym to struktura danych składająca się ze skończonej liczby **węzłów** / **wierzchołków** / **stanów** i łączących je **krawędzi** / **gałęzi**.

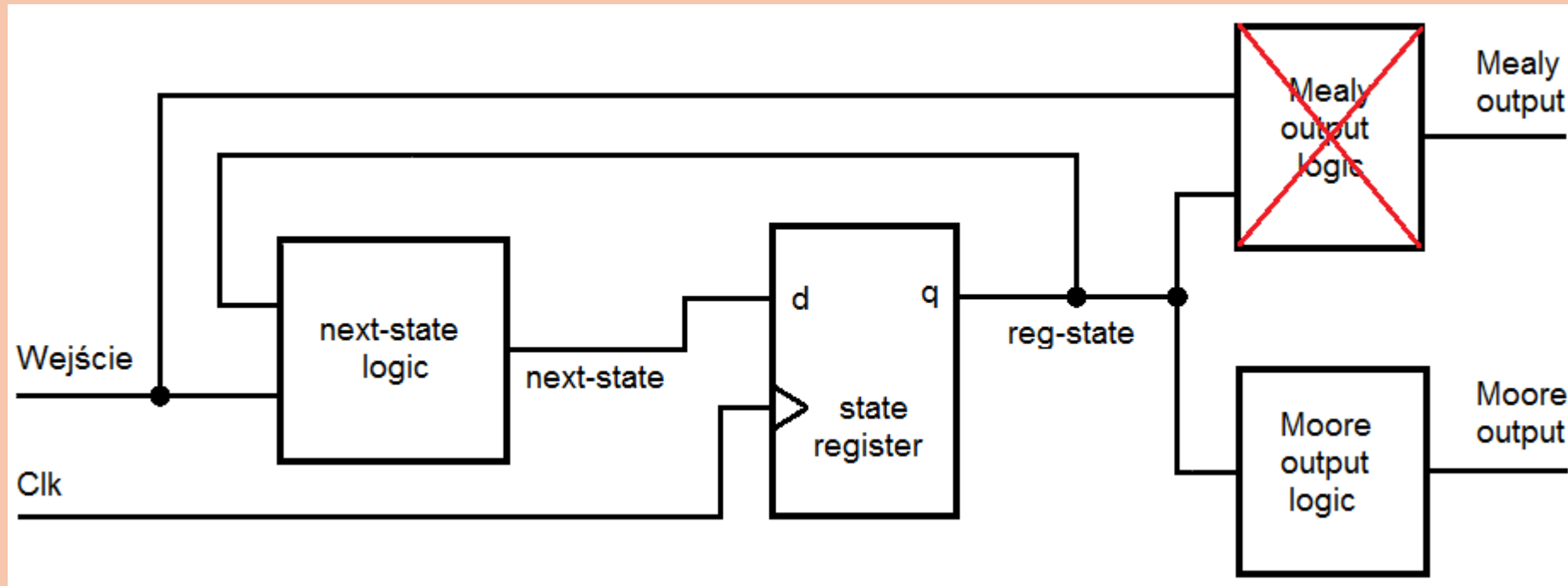


Wierzchołek grafu jest stanem układu cyfrowego, opisuje jego zachowanie w określonym momencie – w tym między innymi wartości sygnałów wyjściowych.

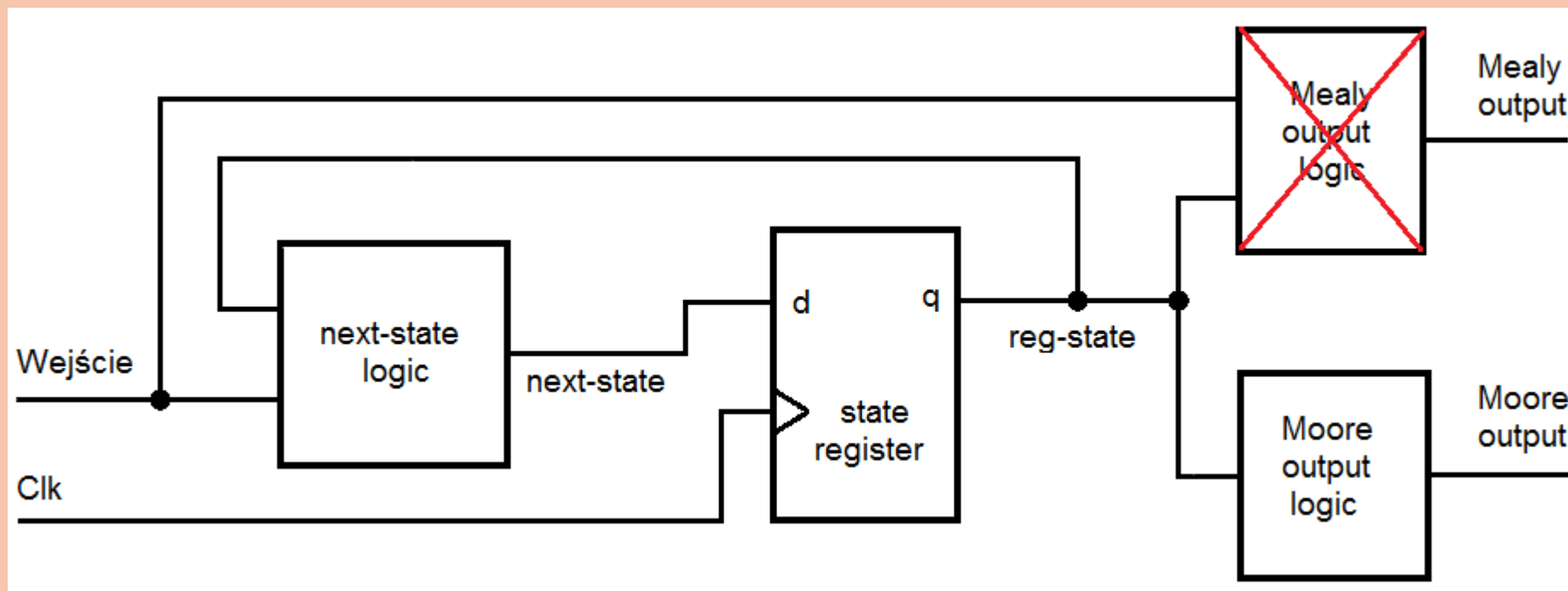
Krawędź grafu to powiązanie/połączenie/przejście pomiędzy dwoma wierzchołkami. Każda krawędź reprezentowana jest przez parę wierzchołków.

Technika FSM (Finite State Machine) jest to jedną z najczęściej stosowanych i najważniejszych technik opisu układów sekwencyjnych w VHDL. Chociaż ma ona również zastosowanie w technice mikroprocesorowej i np. programowaniu Python. Podręcznikowo, układ opisywany tą techniką zawiera 3 komponenty składowe - bloki:

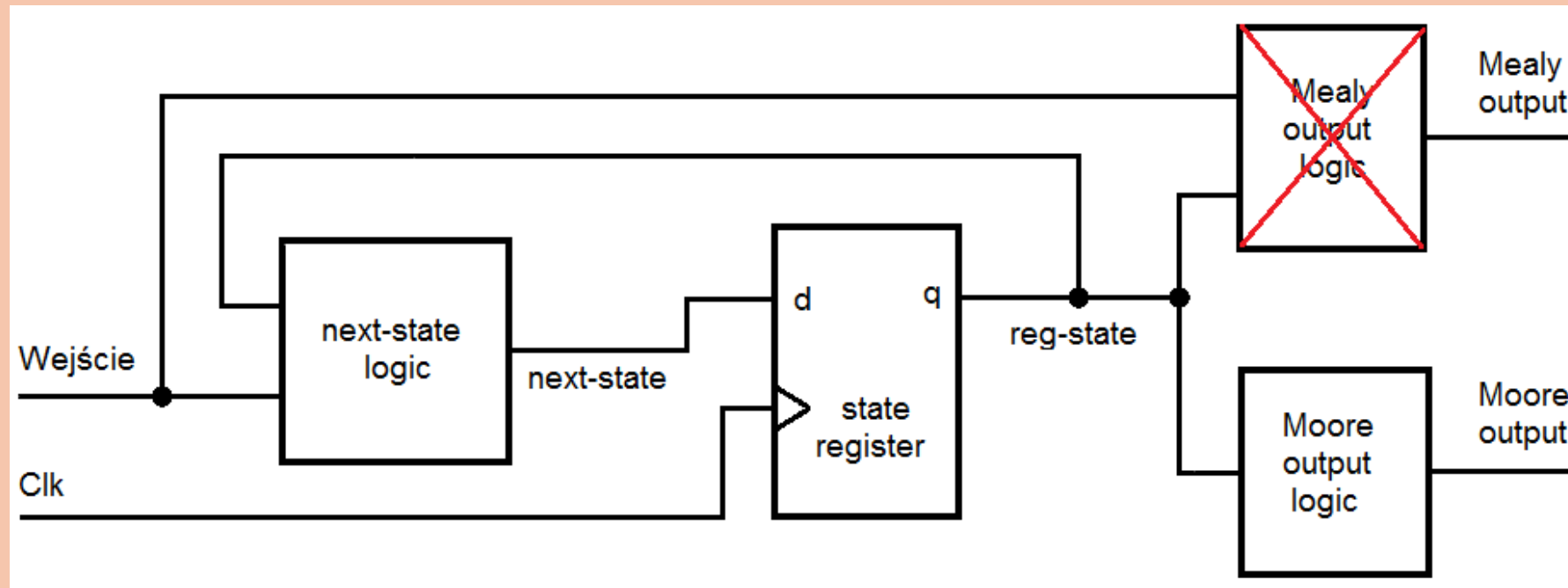
- next-state logic – wejściową część kombinacyjną,
- state register – pamięć stanów,
- output logic – wyjściową część kombinacyjną w wersji Moore'a i/lub Mealy'ego.



Układ typu Mealyego nie będzie analizowany z powodu mniejszego znaczenia w FPGA



- funkcją next-state logic – jest wyznaczenie następnego stanu na podstawie aktualnych sygnałów wejściowych i informacji o aktualnym stanie,
- rolą state register jest pamiętanie bieżącego stanu,
- funkcją output logic (w wersji Moore’a) jest wyznaczenie wartości sygnałów wyjściowych na podstawie nazwy/kodu aktualnego stanu, sygnał wyjściowy jest związany ze stanem.



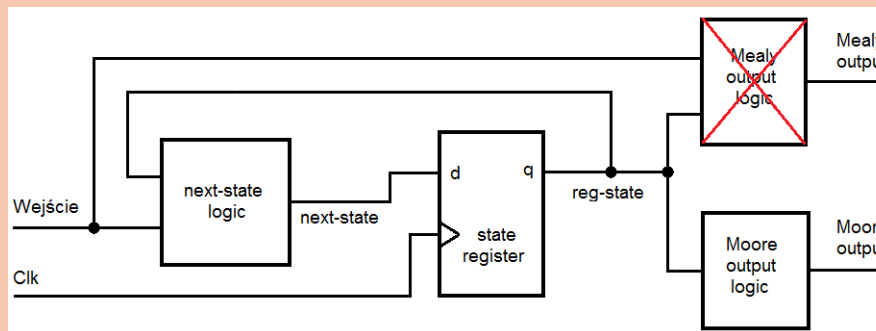
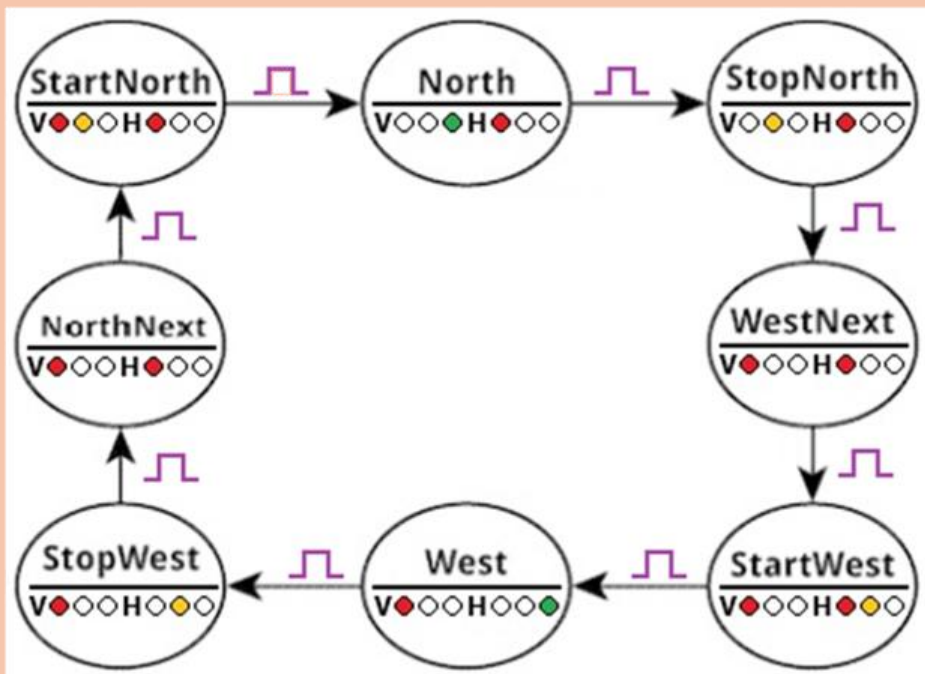
Sposób kodowania FSM jest typowy dla układów sekwencyjnych:

- na początku wydzielamy „rejestr stanu”,
- w dalszej części kodujemy część kombinacyjną (next-state logic),
- na końcu kodujemy kombinacyjną część wyjściową.

Opisując rejestr stanu używamy typu wyliczeniowego a następnie deklarujemy sygnały nowego typu:

type <type_name> **is** (<state_name1>, <state_name2>, ...); -- lista nazw stanów

signal <signal_name> : <type_name>; -- sygnały pomocnicze



stąd deklaracja rejestru stanu przyjmuje formę:

```
type FSM_1 is (StartNorth, North, StopNorth, WestNext, StartWest, West,
                StopWest, NorthNext); -- nowy wyliczeniowy typ danych
```

```
signal next_state, reg_state : FSM_1; -- mamy 2 sygnały wewnętrzne
```

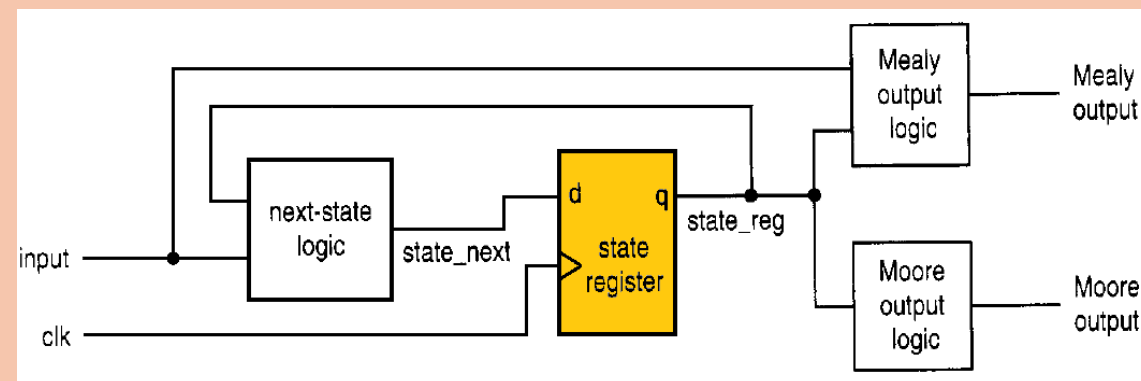
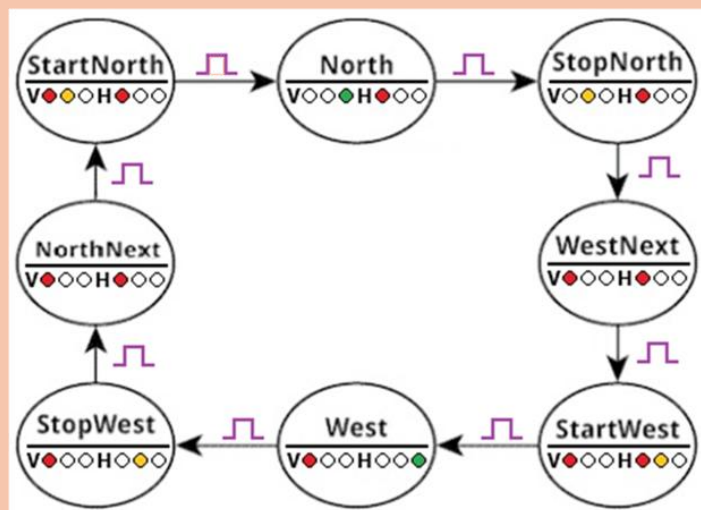
Układ posiada tylko 2 wejścia – wejście zegarowe i wejście zerujące, oraz 6 wyjść:

V_R (czerwona lampa wertykalna – pionowa red), V_Y (żółta – pionowa yellow), V_G (zielona – pionowa green), H_R (czerwona horyzontalna), H_Y (żółta) i H_G (zielona). Stąd początek kodu jest następujący:

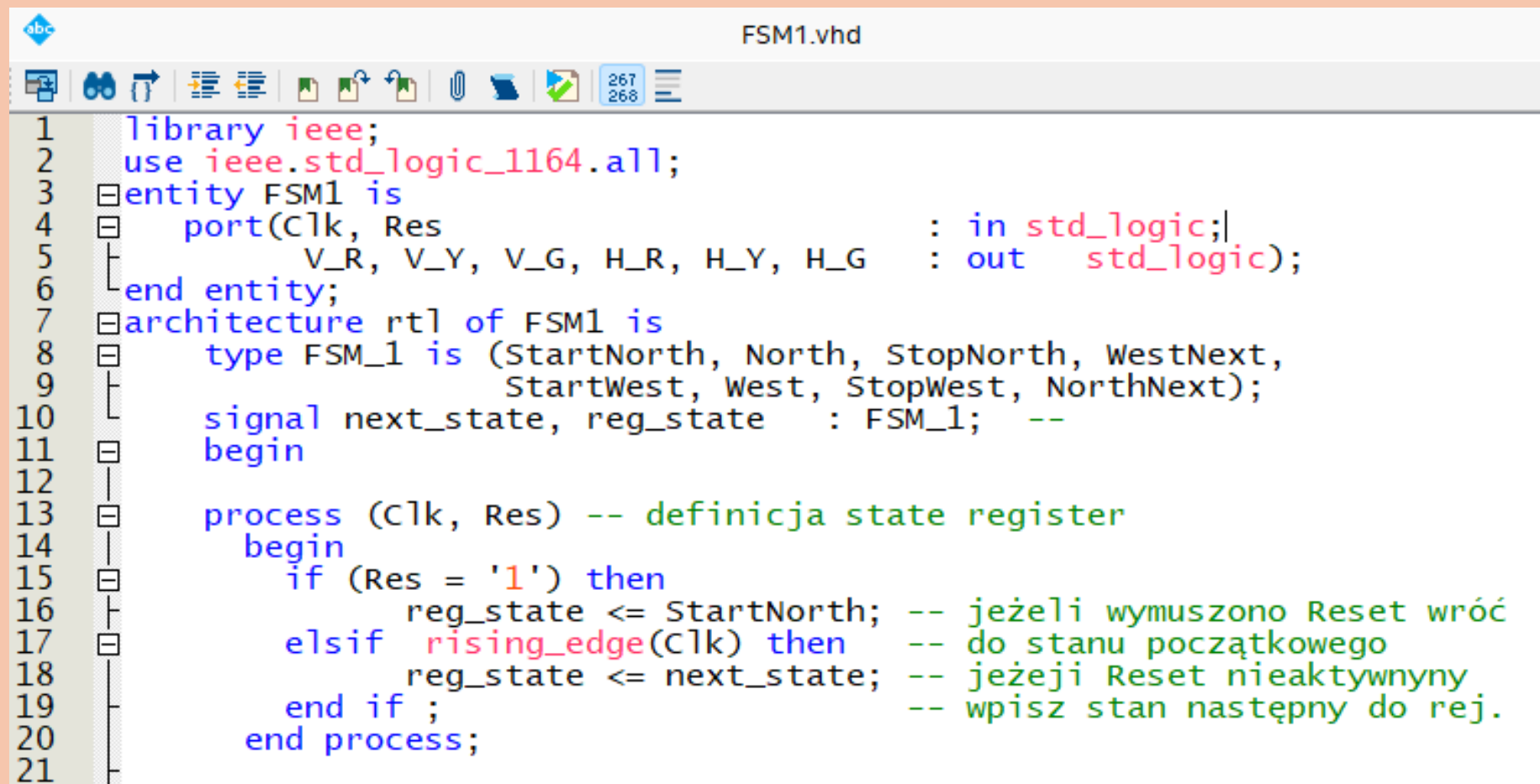
```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity FSM1 is
4  port(
5      Clk, Res : in  std_logic;
6      V_R, V_Y, V_G, H_R, H_Y, H_G : out std_logic
7  );
8  end entity;
9  architecture rtl of FSM1 is
10
11      type FSM_1 is (StartNorth, North, StopNorth, WestNext, StartWest, West, StopWest, NorthNext);
12
13      signal next_state, reg_state : FSM_1; --
14
15  begin

```



W następnej kolejności definiujemy rejestr/pamięć stanów – używając procesu:



```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity FSM1 is
4  port(Clk, Res
5      V_R, V_Y, V_G, H_R, H_Y, H_G : out std_logic);
6  end entity;
7  architecture rtl of FSM1 is
8  type FSM_1 is (StartNorth, North, StopNorth, WestNext,
9      StartWest, West, StopWest, NorthNext);
10 signal next_state, reg_state : FSM_1; --
11 begin
12
13     process (Clk, Res) -- definicja state register
14     begin
15         if (Res = '1') then
16             reg_state <= StartNorth; -- jeżeli wymuszono Reset wróć
17         elsif rising_edge(Clk) then -- do stanu początkowego
18             reg_state <= next_state; -- jeżeli Reset nieaktywny
19         end if ; -- wpisz stan następny do rej.
20     end process;
21

```

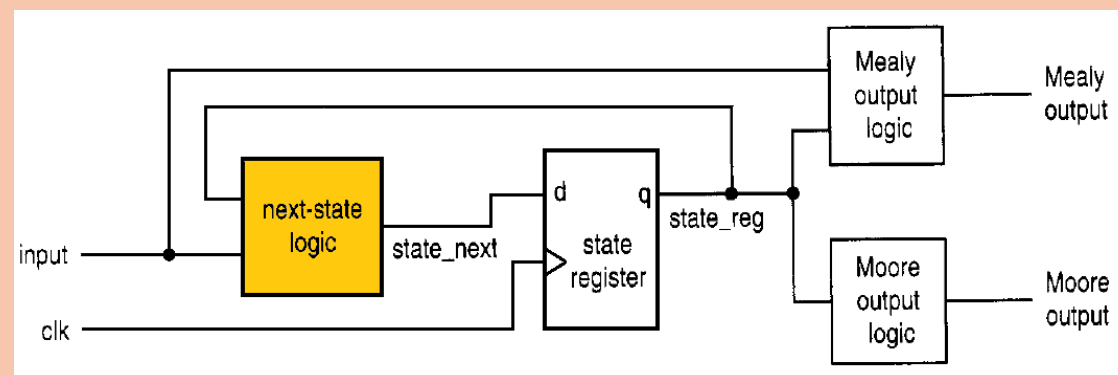
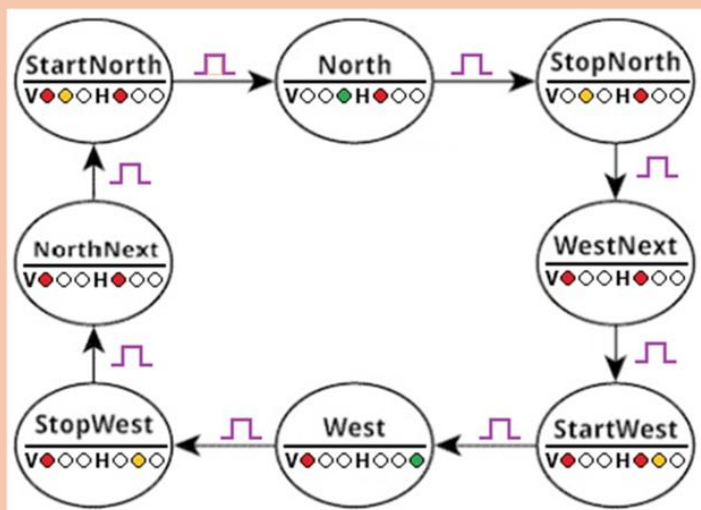
Wpisanie nowego stanu jako bieżącego następuje na narastającym zboczu sygnału zegarowego. To okres sygnału zegarowego wyznacza czas przebywania układu w określonym stanie.

Po zdefiniowaniu rejestru stanu, w kolejnym kroku definiujemy wejściowy układ kombinacyjny **next-state logic** – również używając procesu i najczęściej instrukcji case:

```

20      end process;
21
22      process (reg_state) -- kodowanie nex-state - przejść w grafie
23      begin
24          case reg_state is
25              when StartNorth => next_state <= North;
26              -- jeżeli jesteś w stanie StartNorth
27              -- następnym stanem będzie North
28              when North      => next_state <= StopNorth;
29              when StopNorth  => next_state <= WestNext;
30              when WestNext   => next_state <= StartWest;
31              when StartWest  => next_state <= West;
32              when West       => next_state <= StopWest;
33              when StopWest   => next_state <= NorthNext;
34              when NorthNext  => next_state <= StartNorth;
35              when others     => next_state <= StartNorth;
36              -- "zawór bezpieczeństwa" zawieszenia układu
37              -- w stanie nieokreślonym
38          end case;
39      end process;

```

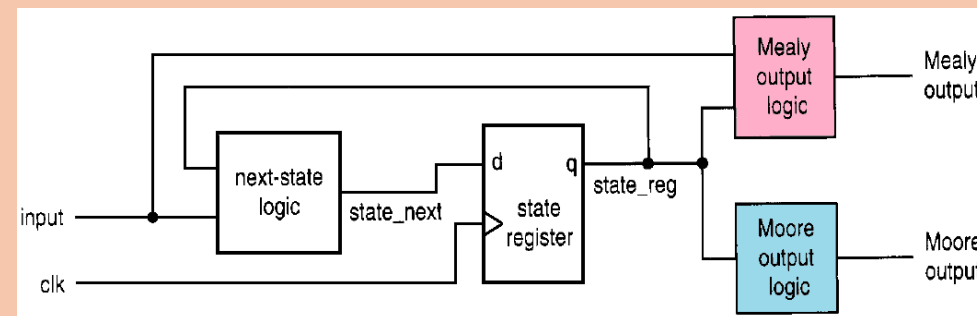
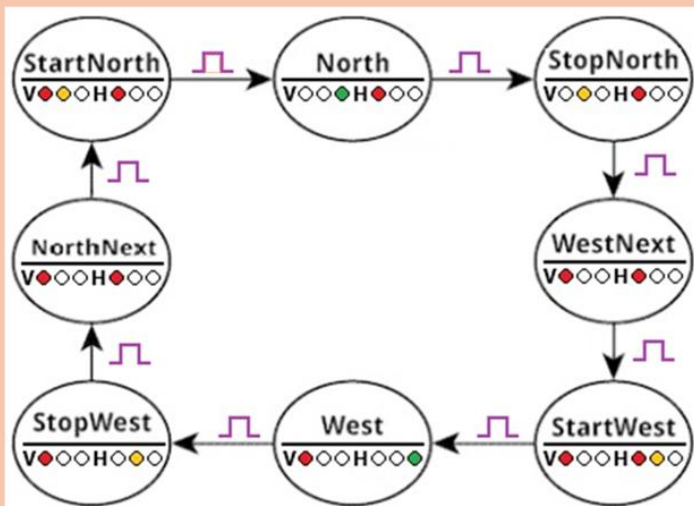


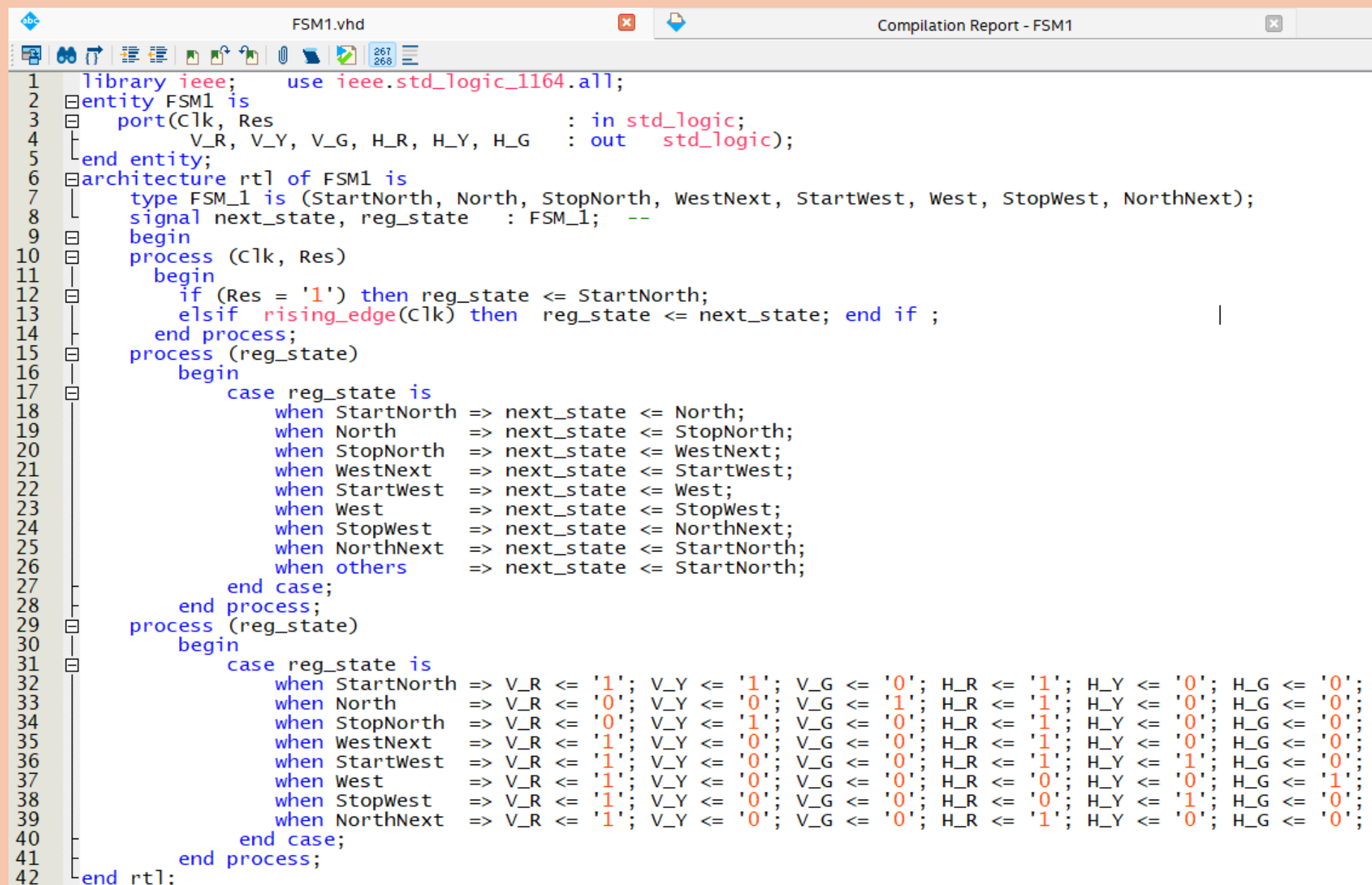
W ostatnim etapie kodowany jest kombinacyjny układ wyjściowy – za pomocą procesu:

```

39      end process;
40
41      process (reg_state) -- kodowanie wyjść układu - output logic
42      begin
43          case reg_state is
44              when StartNorth => V_R <= '1'; V_Y <= '1'; V_G <= '0';
45                               H_R <= '1'; H_Y <= '0'; H_G <= '0';
46              when North      => V_R <= '0'; V_Y <= '0'; V_G <= '1';
47                               H_R <= '1'; H_Y <= '0'; H_G <= '0';
48              when StopNorth  => V_R <= '0'; V_Y <= '1'; V_G <= '0';
49                               H_R <= '1'; H_Y <= '0'; H_G <= '0';
50              when WestNext   => V_R <= '1'; V_Y <= '0'; V_G <= '0';
51                               H_R <= '1'; H_Y <= '0'; H_G <= '0';
52              when StartWest  => V_R <= '1'; V_Y <= '0'; V_G <= '0';
53                               H_R <= '1'; H_Y <= '1'; H_G <= '0';
54              when West       => V_R <= '1'; V_Y <= '0'; V_G <= '0';
55                               H_R <= '0'; H_Y <= '0'; H_G <= '1';
56              when StopWest   => V_R <= '1'; V_Y <= '0'; V_G <= '0';
57                               H_R <= '0'; H_Y <= '1'; H_G <= '0';
58              when NorthNext  => V_R <= '1'; V_Y <= '0'; V_G <= '0';
59                               H_R <= '1'; H_Y <= '0'; H_G <= '0';
60          end case;
61      end process;
62  end rtl;

```



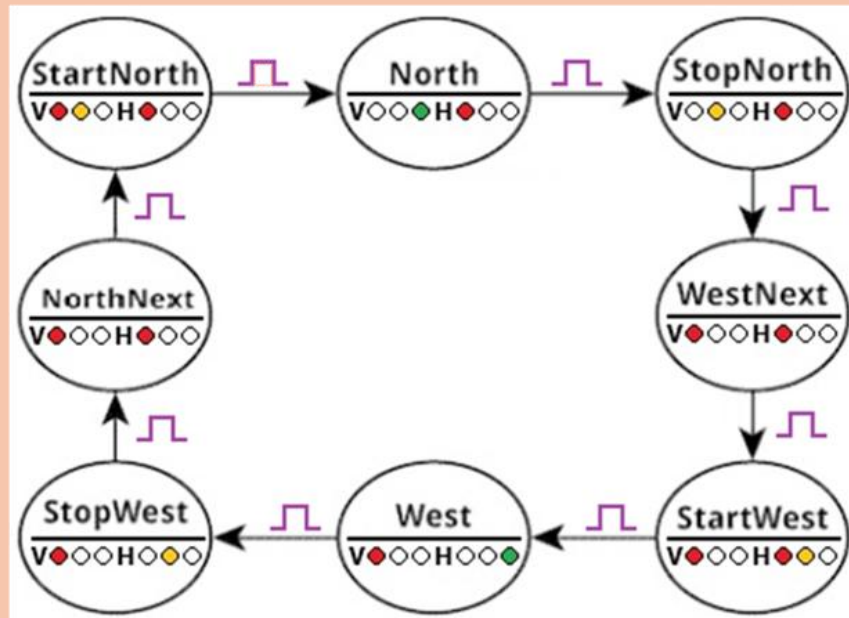
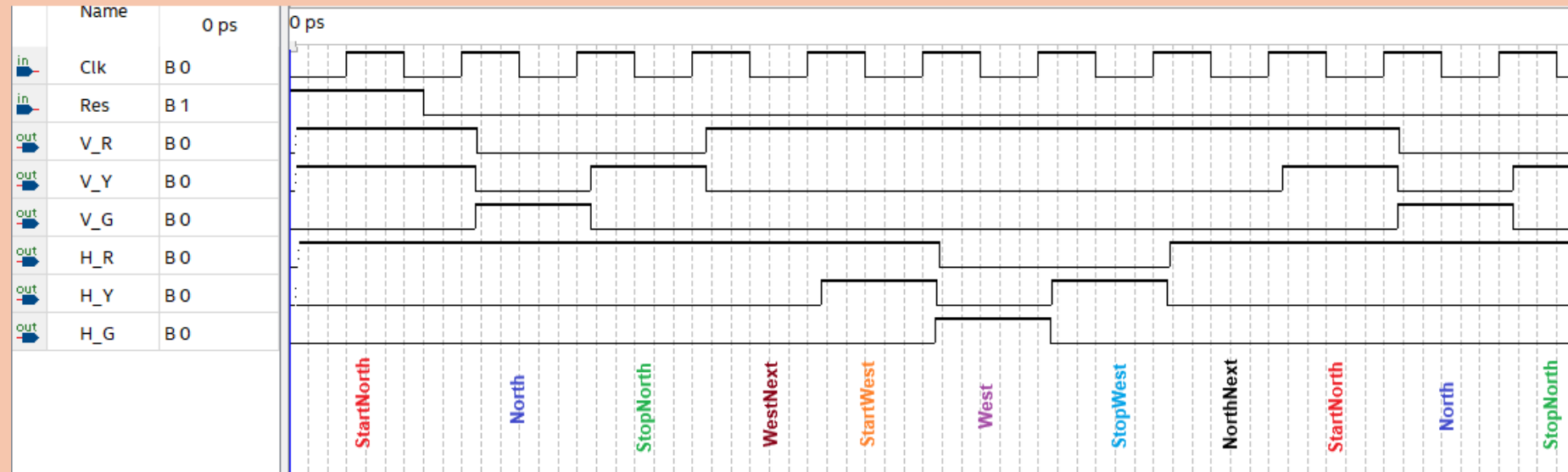


```

1  library ieee;      use ieee.std_logic_1164.all;
2  entity FSM1 is
3  port(Clk, Res       : in std_logic;
4       V_R, V_Y, V_G, H_R, H_Y, H_G : out std_logic);
5  end entity;
6  architecture rtl of FSM1 is
7      type FSM_1 is (StartNorth, North, StopNorth, WestNext, StartWest, West, StopWest, NorthNext);
8      signal next_state, reg_state : FSM_1; --
9  begin
10     process (Clk, Res)
11     begin
12         if (Res = '1') then reg_state <= StartNorth;
13         elsif rising_edge(Clk) then reg_state <= next_state; end if ;
14     end process;
15     process (reg_state)
16     begin
17         case reg_state is
18             when StartNorth => next_state <= North;
19             when North      => next_state <= StopNorth;
20             when StopNorth  => next_state <= WestNext;
21             when WestNext   => next_state <= StartWest;
22             when StartWest  => next_state <= West;
23             when West       => next_state <= StopWest;
24             when StopWest   => next_state <= NorthNext;
25             when NorthNext  => next_state <= StartNorth;
26             when others     => next_state <= StartNorth;
27         end case;
28     end process;
29     process (reg_state)
30     begin
31         case reg_state is
32             when StartNorth => V_R <= '1'; V_Y <= '1'; V_G <= '0'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
33             when North      => V_R <= '0'; V_Y <= '0'; V_G <= '1'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
34             when StopNorth  => V_R <= '0'; V_Y <= '1'; V_G <= '0'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
35             when WestNext   => V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
36             when StartWest  => V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '1'; H_Y <= '1'; H_G <= '0';
37             when West       => V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '0'; H_Y <= '0'; H_G <= '1';
38             when StopWest   => V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '0'; H_Y <= '1'; H_G <= '0';
39             when NorthNext  => V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
40         end case;
41     end process;
42 end rtl;

```

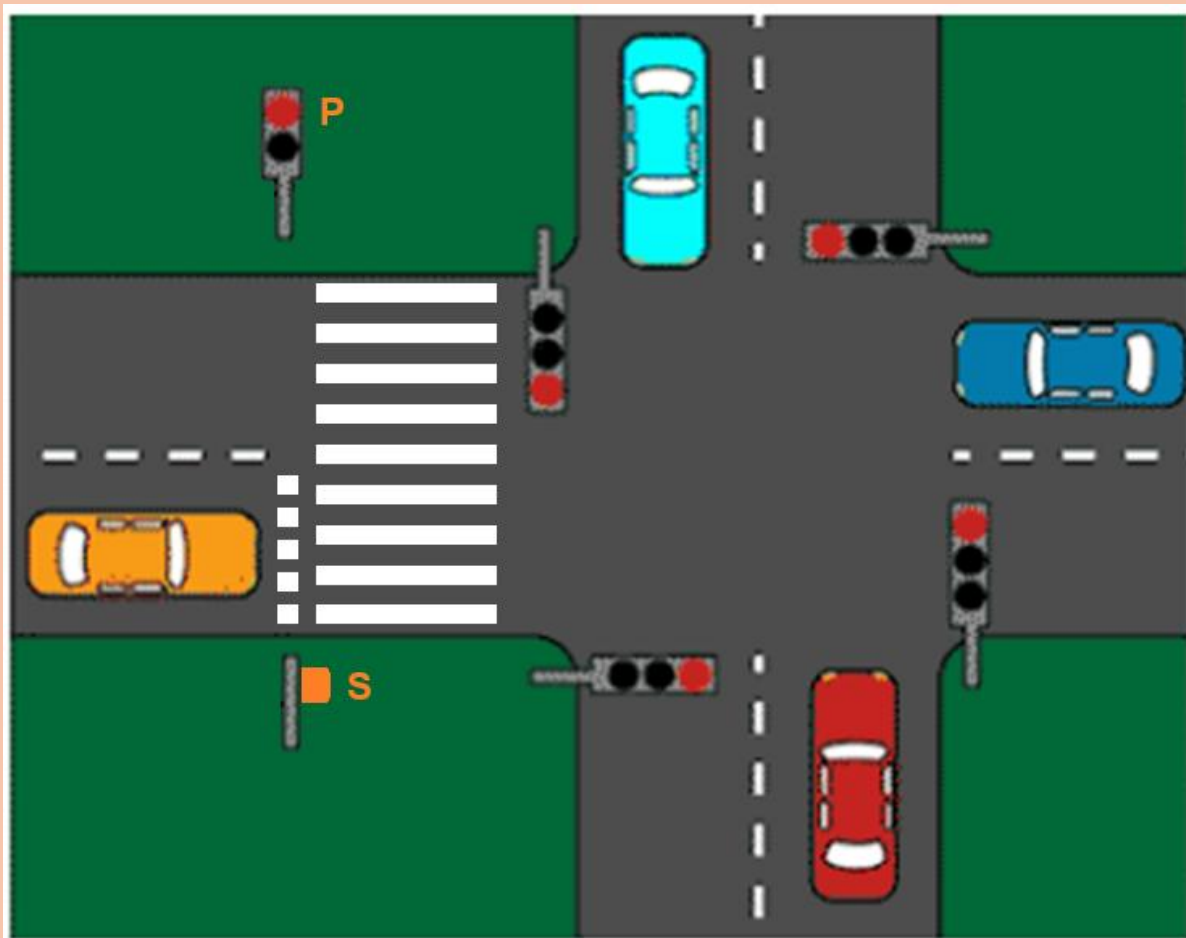
Wynik symulacji:



Dotychczas poznaliśmy 2 mechanizmy, które mogą modyfikować pracę FSM:

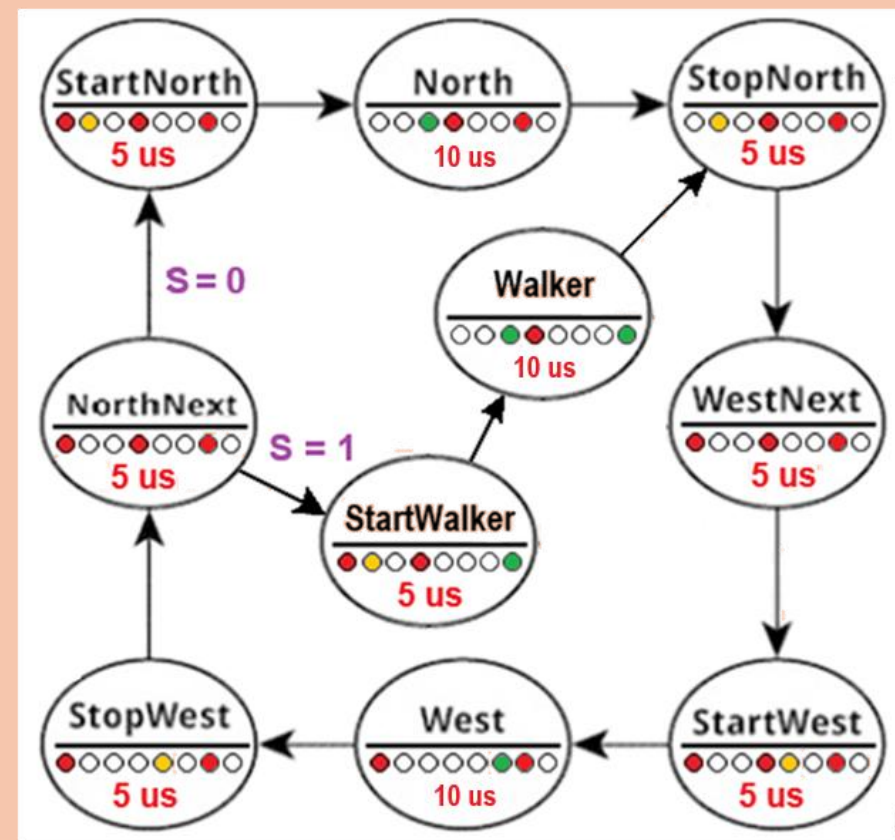
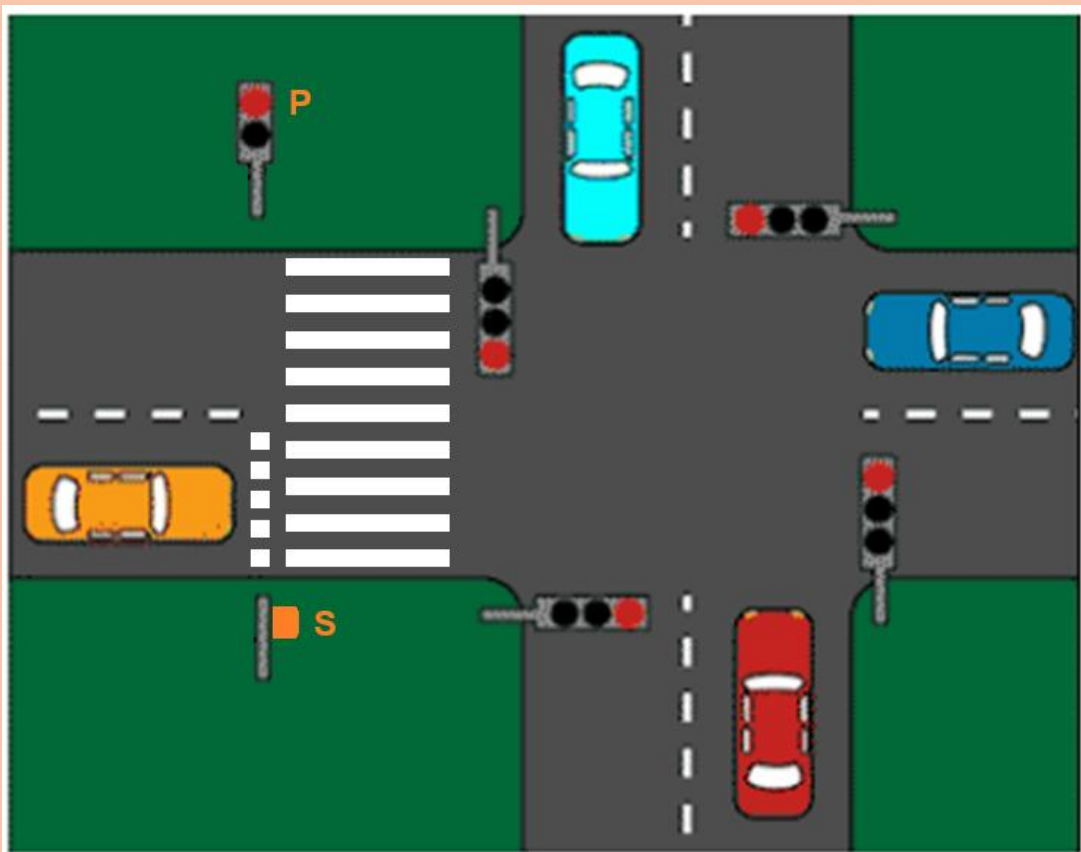
- częstotliwość/okres sygnału zegarowego - wyznacza trwania stanu układu,
- wydłużenie pozostawania układu w stanie poprzez zliczanie impulsów zegarowych.

Trzecim czynnikiem są sygnały/zmienne warunkujące zmianę stanu - wróćmy do układu:



Układ skrzyżowania został uzupełniony o sterowane przejście dla pieszych. W związku z tym, pojawiła się dodatkowa lampa P ze światłami P_R i P_G, oraz przycisk S włączający zielone światło dla pieszych na żądanie. W tej sytuacji sygnał S staje się zewnętrznym czynnikiem wpływającym na algorytm sterowanie światłami skrzyżowania.

Sygnal żądania zielonego światła dla pieszych może zostać zgłoszony przyciskiem S w dowolnym momencie/stanie układu. Nie oznacza to natychmiastowej zmiany sekwencji. Rozpoczęcie obsługi przejścia dla pieszych następuje w stanie NorthNext. W przypadku aktywnego sygnału $S = 1$, następuje przejście do stanu StartWalker – połączone z zaświeceniem zielonego światła lampy P na przejściu dla pieszych. Dla $S = 0$ układ pracuje w dotychczasowym domyślnym cyklu.



W zmodyfikowanym układzie występują 2 dodatkowe sygnały wyjściowe PR i PG świateł przejścia dla pieszych. Światła przejścia dla pieszych obsługują dwa dodatkowe stany: StartWalker obsługujący zielone światło przejścia oraz stan StartNorth dla ruchu pionowego, Walker obsługujący przejście i stan North. Wewnętrzny sygnał PP służy do pamiętania faktu użycia przycisku S.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity FSM1a is
5  port( Clk, Res : in std_logic; -- Clk 50MHz/20 ns, zerowanie poziomem 1
6        S : in std_logic := '0'; -- przycisk przejścia
7        VR : out std_logic; VY : out std_logic; -- czerwone / żółte światło pionowe
8        VG : out std_logic; HR : out std_logic; -- zielone pionowe / czerwone poziome
9        HY : out std_logic; HG : out std_logic; -- żółte poziome / zielone poziome
10       PR : out std_logic; PG : out std_logic; -- czerwone / zielone światło na przejściu
11       M : out integer range 0 to 9); -- pomocniczy do symulacji sygnał kodu stanów:
12       -- StartNorth = 0, North = 1, StopNorth = 2, WestNext = 3, StartWest = 4,
13       -- West = 5, StopWest = 6, NorthNext = 7, StartWalker = 8, Walker = 9
14  end entity;
15
16  architecture rtl of FSM1a is
17  type FSM_1 is (StartNorth, North, StopNorth, WestNext, StartWest, West,
18                StopWest, NorthNext, StartWalker, Walker);
19  signal Graf : FSM_1;
20  signal PP : std_logic := '0'; -- sygnał pamiętający stan przycisku S
21  begin
22  process (Clk, Res, S)
23
24  variable tmp : integer := 0;
25  begin

```

W celu prostszej interpretacji wyników symulacji, wprowadzono dodatkowe wyjście M, które wyświetla nr aktualnego stanu. To wyjście nie ma znaczenia praktycznego.

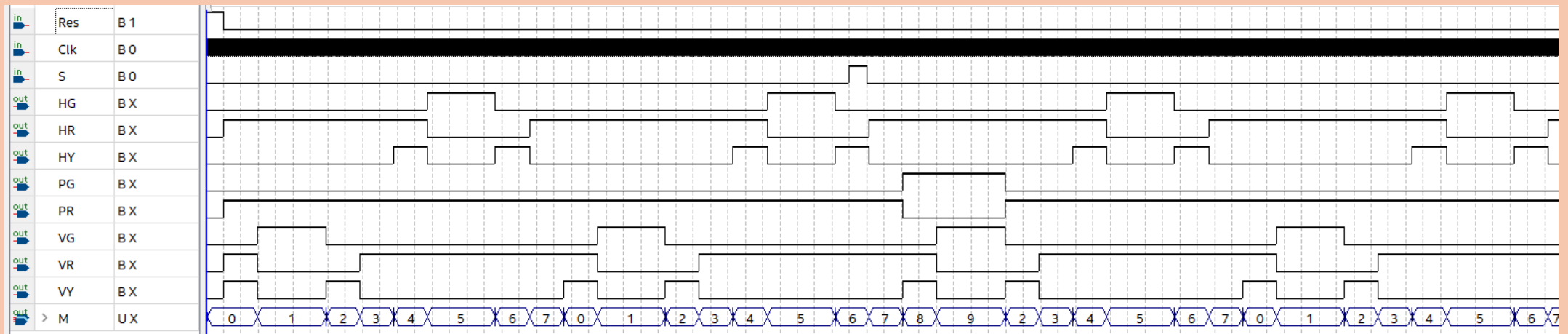
Pozostała część kodu wygląda następująco – skrócono czasy trwania stanów do 250 i 500 okresów zegarowych:

```

24 variable tmp : integer := 0;
25 begin
26   if (Res = '1') then Graf <= StartNorth;
27   elsif rising_edge(Clk) then
28     case Graf is
29       when StartNorth => if PP = '0' then PP <= S; M <= 0; end if; -- wczytanie S
30         VR <= '1'; VY <= '1'; VG <= '0'; HR <= '1'; HY <= '0'; HG <= '0'; PR <= '1'; PG <= '0';
31         tmp := tmp + 1; if tmp = 250 then tmp := 0; Graf <= North; end if;
32       when North => if PP = '0' then PP <= S; M <= 1; end if; -- wczytanie S
33         VR <= '0'; VY <= '0'; VG <= '1'; HR <= '1'; HY <= '0'; HG <= '0'; PR <= '1'; PG <= '0';
34         tmp := tmp + 1; if tmp = 500 then tmp := 0; Graf <= StopNorth; end if;
35       when StopNorth => if PP = '0' then PP <= S; M <= 2; end if; -- wczytanie S
36         VR <= '0'; VY <= '1'; VG <= '0'; HR <= '1'; HY <= '0'; HG <= '0'; PR <= '1'; PG <= '0';
37         tmp := tmp + 1; if tmp = 250 then tmp := 0; Graf <= WestNext; end if;
38       when WestNext => if PP = '0' then PP <= S; M <= 3; end if; -- wczytanie S
39         VR <= '1'; VY <= '0'; VG <= '0'; HR <= '1'; HY <= '0'; HG <= '0'; PR <= '1'; PG <= '0';
40         tmp := tmp + 1; if tmp = 250 then tmp := 0; Graf <= StartWest; end if;
41       when StartWest => if PP = '0' then PP <= S; M <= 4; end if; -- wczytanie S
42         VR <= '1'; VY <= '0'; VG <= '0'; HR <= '1'; HY <= '1'; HG <= '0'; PR <= '1'; PG <= '0';
43         tmp := tmp + 1; if tmp = 250 then tmp := 0; Graf <= West; end if;
44       when West => if PP = '0' then PP <= S; M <= 5; end if; -- wczytanie S
45         VR <= '1'; VY <= '0'; VG <= '0'; HR <= '0'; HY <= '0'; HG <= '1'; PR <= '1'; PG <= '0';
46         tmp := tmp + 1; if tmp = 500 then tmp := 0; Graf <= StopWest; end if;
47       when StopWest => if PP = '0' then PP <= S; M <= 6; end if; -- wczytanie S
48         VR <= '1'; VY <= '0'; VG <= '0'; HR <= '0'; HY <= '1'; HG <= '0'; PR <= '1'; PG <= '0';
49         tmp := tmp + 1; if tmp = 250 then tmp := 0; Graf <= NorthNext; end if;
50       when NorthNext => M <= 7; -- ignorujemy wczytanie S
51         VR <= '1'; VY <= '0'; VG <= '0'; HR <= '1'; HY <= '0'; HG <= '0'; PR <= '1'; PG <= '0';
52         tmp := tmp + 1; if tmp = 250 then tmp := 0;
53         if PP = '0' then Graf <= StartNorth; else Graf <= StartWalker; end if; end if;
54       when StartWalker=> M <= 8; -- ignorujemy wczytanie S
55         VR <= '1'; VY <= '1'; VG <= '0'; HR <= '1'; HY <= '0'; HG <= '0'; PR <= '0'; PG <= '1';
56         tmp := tmp + 1; if tmp = 250 then tmp := 0; Graf <= Walker; end if;
57       when Walker => PP <= '0'; M <= 9; -- zerujemy wskaźnik przycisku S
58         VR <= '0'; VY <= '0'; VG <= '1'; HR <= '1'; HY <= '0'; HG <= '0'; PR <= '0'; PG <= '1';
59         tmp := tmp + 1; if tmp = 500 then tmp := 0; Graf <= StopNorth; end if;
60       when others => Graf <= StartNorth;
61     end case;
62   end if;
63 end process;
64 end rtl;

```

Wyniki symulacji przedstawiono na poniższym rysunku. W przypadku barku sygnału z przycisku S układ przemieszcza się cyklicznie pomiędzy stanami 0 -7. Po „przyciśnięciu” S w stanie 6 układ dochodzi do stanu 7 w nim przechodzi do stanów 8 i 9 a następnie powraca do głównej pętli w stanie 2. Ze względu na czytelność, symulację przeprowadzono dla okresu sygnału zegarowego 10 ns, czyli częstotliwości 100 MHz.

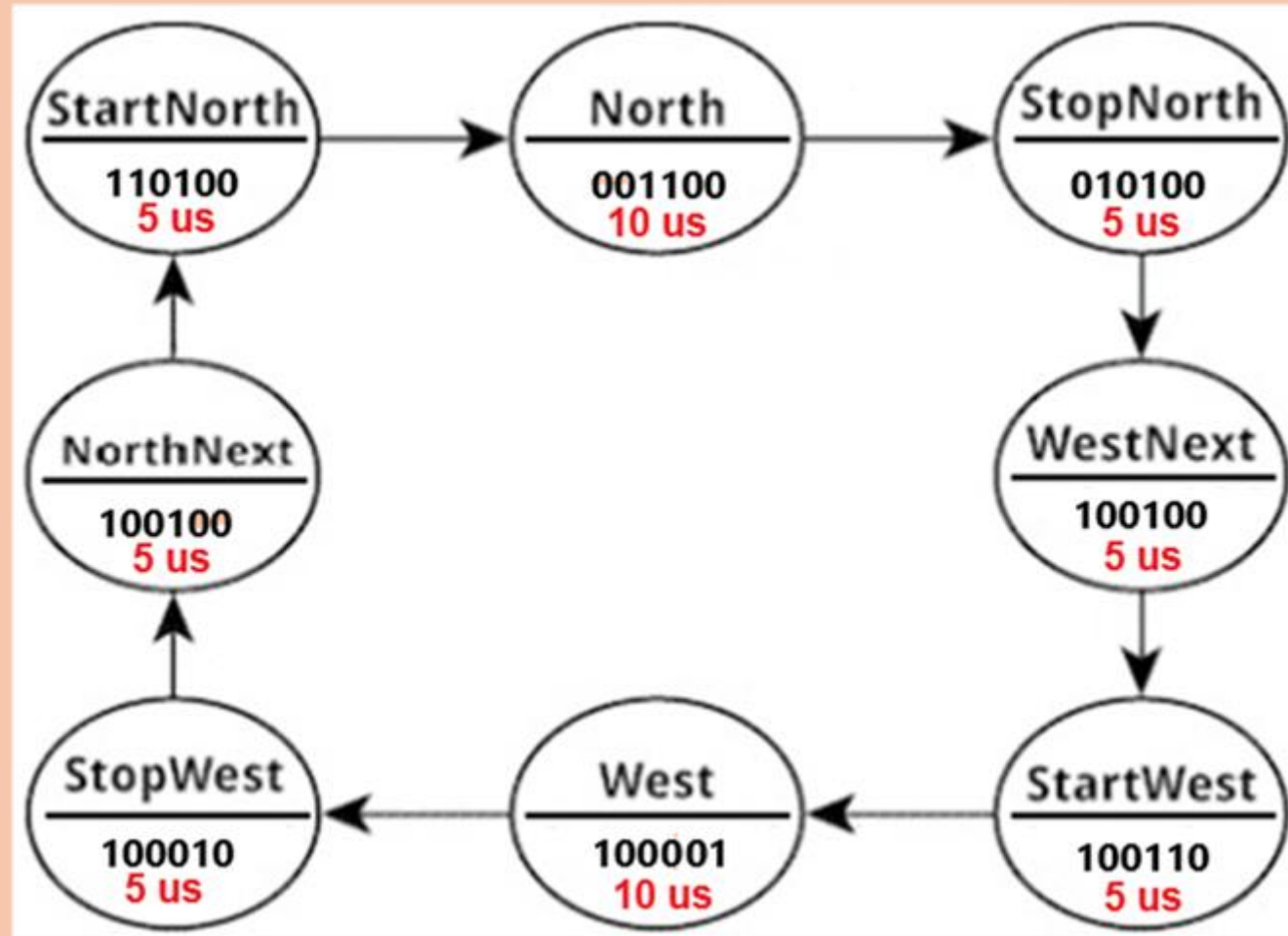


Interpretacja przebiegów czasowych FSM może być kłopotliwa, stąd warto wprowadzać dodatkowe sygnały/markery wyjściowe żeby jednoznacznie zweryfikować: przez które stany układ przechodzi albo gdzie zatrzymuje się w przypadku błędów.

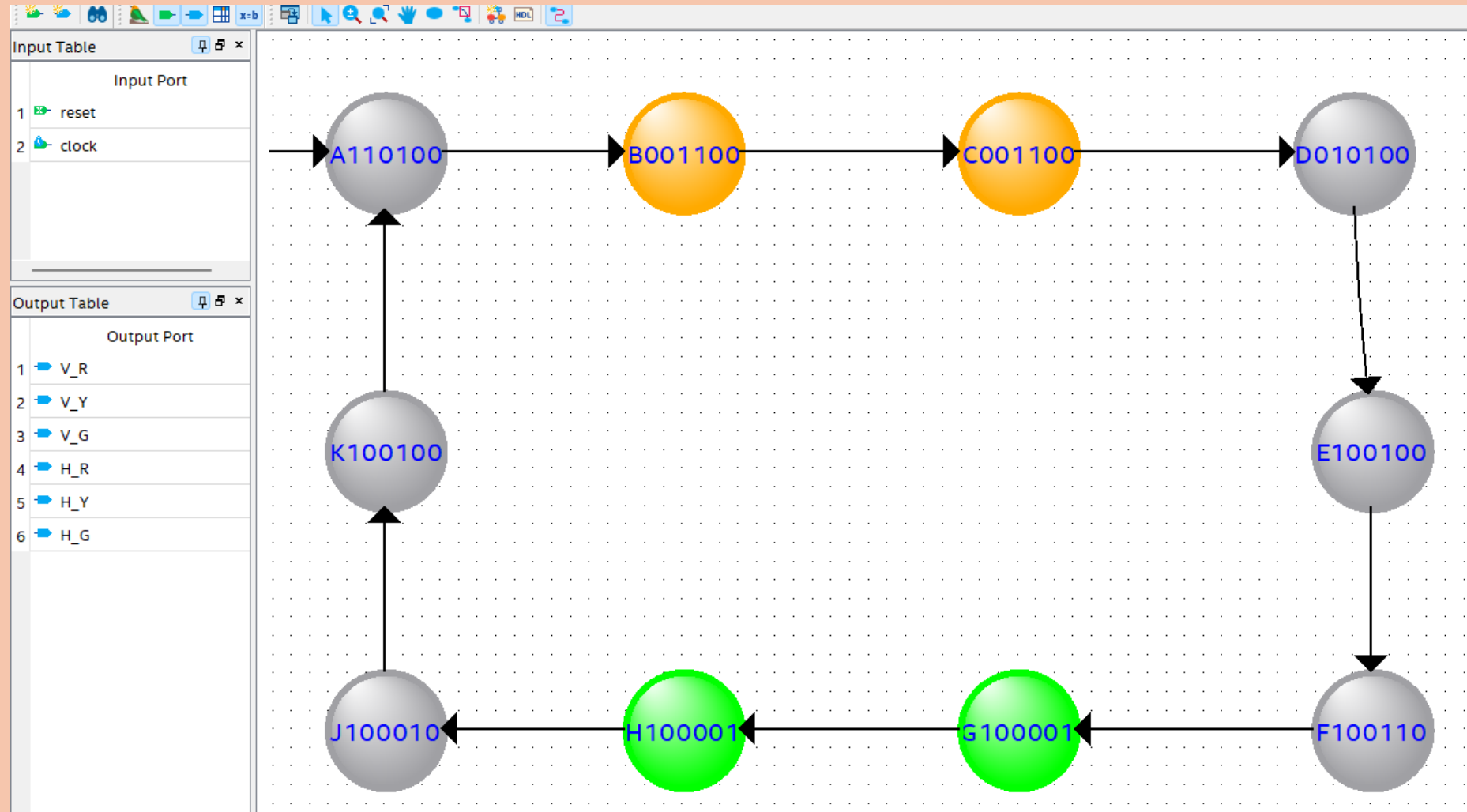
W układach rzeczywistych stosuje się często więcej niż jeden sygnał zewnętrzny a sterowanie sygnałami zewnętrznymi łączy się z pomiarem czasu.

Podczas pomiaru czasu, należy zwrócić uwagę na zerowanie w każdym stanie sygnałów/zmiennych pomocniczych, które służą do zliczania impulsów zegarowych.

W przypadku prostych układów FSM, takich jak zmodyfikowany niżej układ sterowania światłami drogowymi, nie zawsze musimy pomiar czasu realizować poprzez zliczanie impulsów zegarowych. W układzie na poniższym rysunku, czasy trwania stanów dłuższych są wielokrotnością czasu stanu najkrótszego



W takiej sytuacji przyjmujemy częstotliwość stanu najkrótszego – a wielokrotność jego czasu dla stanów dłuższych uzyskujemy powielając te same stany. Np. stan A110100 trwa 5 us a stany B001100 i C001100 są identyczne, wytwarzają te same sygnały wyjściowe – co daje sumaryczny czas tych stanów równy $2 \times 5 \text{ us} = 1 \text{ us}$.



Jak widać automatyczny kod jest krótszy niż w metodzie inkrementacji zmiennej impulsami zegarowymi. Częstotliwość sygnału zegarowego powinna być dobrana do wartości okresu najkrótszego stanu, czyli 5 us co daje częstotliwość 200 kHz. Przedstawiona metoda kontroli czasów stanów posiada swoje ograniczenia:

- jej rozdzielczość jest mniejsza niż w przypadku inkrementacji zmiennej – gdzie krok inkrementacji może wynosić nawet około 1 ns,
- może być użyta w wąskiej klasie układów – gdy czasy stanów są wielokrotnościami.

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  ENTITY SM1 IS
4  PORT ( reset : IN STD_LOGIC := '0'; clock : IN STD_LOGIC;
5        V_R : OUT STD_LOGIC; V_Y : OUT STD_LOGIC; V_G : OUT STD_LOGIC; H_R : OUT STD_LOGIC; H_Y : OUT STD_LOGIC; H_G : OUT STD_LOGIC);
6  END SM1;
7
8  ARCHITECTURE BEHAVIOR OF SM1 IS
9      TYPE type_fstate IS (A110100,D010100,E100100,F100110,J100010,K100100,B001100,C001100,G100001,H100001);
10     SIGNAL fstate : type_fstate;
11     SIGNAL reg_fstate : type_fstate;
12 BEGIN
13     PROCESS (clock,reg_fstate)
14     BEGIN
15         IF (clock='1' AND clock'event) THEN
16             fstate <= reg_fstate;
17         END IF;
18     END PROCESS;
19
20     PROCESS (fstate,reset)
21     BEGIN
22         IF (reset='1') THEN      reg_fstate <= A110100; V_R <= '0'; V_Y <= '0'; V_G <= '0'; H_R <= '0'; H_Y <= '0'; H_G <= '0';
23         ELSE                    V_R <= '0'; V_Y <= '0'; V_G <= '0'; H_R <= '0'; H_Y <= '0'; H_G <= '0';
24             CASE fstate IS
25             WHEN A110100 => reg_fstate <= B001100; V_R <= '1'; V_Y <= '1'; V_G <= '0'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
26             WHEN B001100 => reg_fstate <= C001100; V_R <= '0'; V_Y <= '0'; V_G <= '1'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
27             WHEN C001100 => reg_fstate <= D010100; V_R <= '0'; V_Y <= '0'; V_G <= '1'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
28             WHEN D010100 => reg_fstate <= E100100; V_R <= '0'; V_Y <= '1'; V_G <= '0'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
29             WHEN E100100 => reg_fstate <= F100110; V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
30             WHEN F100110 => reg_fstate <= G100001; V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '1'; H_Y <= '1'; H_G <= '0';
31             WHEN J100010 => reg_fstate <= K100100; V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '0'; H_Y <= '1'; H_G <= '0';
32             WHEN K100100 => reg_fstate <= A110100; V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '1'; H_Y <= '0'; H_G <= '0';
33             WHEN G100001 => reg_fstate <= H100001; V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '0'; H_Y <= '0'; H_G <= '1';
34             WHEN H100001 => reg_fstate <= J100010; V_R <= '1'; V_Y <= '0'; V_G <= '0'; H_R <= '0'; H_Y <= '0'; H_G <= '1';
35             WHEN OTHERS => reg_fstate <= J100010; V_R <= 'X'; V_Y <= 'X'; V_G <= 'X'; H_R <= 'X'; H_Y <= 'X'; H_G <= 'X';
36             report "Reach undefined state";
37         END CASE;
38     END IF;
39 END PROCESS;
40 END BEHAVIOR;
41

```

Wyniki symulacji przykładu z częstotliwością zegarową 200 kHz pokazano na poniższym rysunku. Widać na nim 3 przypadki stanów podwójnych – generujących te same sygnały wyjściowe.

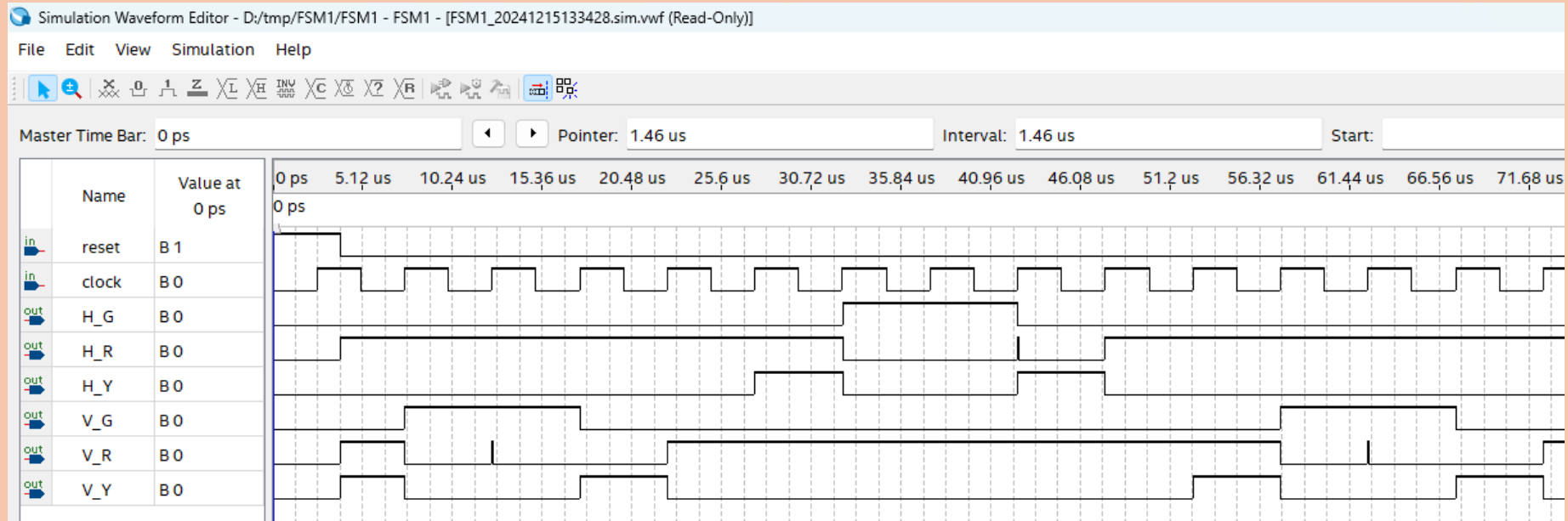


Table of Contents	Flow Summary																												
- Flow Summary - Flow Settings - Flow Non-Default Global Settings - Flow Elapsed Time - Flow OS Summary - Flow Log - Analysis & Synthesis - Fitter - Assembler - Timing Analyzer - EDA Netlist Writer - Flow Messages - Flow Suppressed Messages	<<Filter>> <table> <tr> <td>Flow Status</td><td>Successful - Sun Dec 15 14:05:05 2024</td></tr> <tr> <td>Quartus Prime Version</td><td>18.1.0 Build 625 09/12/2018 SJ Lite Edition</td></tr> <tr> <td>Revision Name</td><td>FSM1</td></tr> <tr> <td>Top-level Entity Name</td><td>SM1</td></tr> <tr> <td>Family</td><td>Cyclone IV E</td></tr> <tr> <td>Device</td><td>EP4CE115F29C7</td></tr> <tr> <td>Timing Models</td><td>Final</td></tr> <tr> <td>Total logic elements</td><td>16 / 114,480 (< 1 %)</td></tr> <tr> <td>Total registers</td><td>10</td></tr> <tr> <td>Total pins</td><td>8 / 529 (2 %)</td></tr> <tr> <td>Total virtual pins</td><td>0</td></tr> <tr> <td>Total memory bits</td><td>0 / 3,981,312 (0 %)</td></tr> <tr> <td>Embedded Multiplier 9-bit elements</td><td>0 / 532 (0 %)</td></tr> <tr> <td>Total PLLs</td><td>0 / 4 (0 %)</td></tr> </table>	Flow Status	Successful - Sun Dec 15 14:05:05 2024	Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition	Revision Name	FSM1	Top-level Entity Name	SM1	Family	Cyclone IV E	Device	EP4CE115F29C7	Timing Models	Final	Total logic elements	16 / 114,480 (< 1 %)	Total registers	10	Total pins	8 / 529 (2 %)	Total virtual pins	0	Total memory bits	0 / 3,981,312 (0 %)	Embedded Multiplier 9-bit elements	0 / 532 (0 %)	Total PLLs	0 / 4 (0 %)
Flow Status	Successful - Sun Dec 15 14:05:05 2024																												
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition																												
Revision Name	FSM1																												
Top-level Entity Name	SM1																												
Family	Cyclone IV E																												
Device	EP4CE115F29C7																												
Timing Models	Final																												
Total logic elements	16 / 114,480 (< 1 %)																												
Total registers	10																												
Total pins	8 / 529 (2 %)																												
Total virtual pins	0																												
Total memory bits	0 / 3,981,312 (0 %)																												
Embedded Multiplier 9-bit elements	0 / 532 (0 %)																												
Total PLLs	0 / 4 (0 %)																												

Ten sam układ opiszemy klasyczną techniką FSM – zawierającą 3 procesy:

- rejestru stanu,
- układu wejściowego,
- układu wyjściowego.

Kod jest nieco dłuższy, ale ...

```

1  LIBRARY ieee; USE ieee.std_logic_1164.all;
2  ENTITY SM1bb IS
3      PORT ( reset : IN STD_LOGIC := '0'; clock : IN STD_LOGIC; V_R, V_Y, V_G, H_R, H_Y, H_G : OUT STD_LOGIC);
4  END SM1bb;
5  ARCHITECTURE BEHAVIOR OF SM1bb IS
6      TYPE type_fstate IS (A110100,D010100,E100100,F100110,J100010,K100100,B001100,C001100,G100001,H100001); SIGNAL next_state, reg_state : type_fstate;
7  BEGIN
8      PROCESS (clock, reset)
9      begin
10         if reset = '1' then reg_state <= A110100; elsif rising_edge(Clock) then reg_state <= next_state; end if;
11     end process;
12     PROCESS (reg_state)
13     begin
14         CASE reg_state IS
15             WHEN A110100 => next_state <= B001100;
16             WHEN B001100 => next_state <= C001100;
17             WHEN C001100 => next_state <= D010100;
18             WHEN D010100 => next_state <= E100100;
19             WHEN E100100 => next_state <= F100110;
20             WHEN F100110 => next_state <= G100001;
21             WHEN G100001 => next_state <= H100001;
22             WHEN H100001 => next_state <= J100010;
23             WHEN J100010 => next_state <= K100100;
24             WHEN K100100 => next_state <= A110100;
25         END CASE;
26     END PROCESS;
27     PROCESS (reg_state)
28     BEGIN
29         CASE reg_state IS
30             WHEN A110100 => V_R <= '1', V_Y <= '1', V_G <= '0', H_R <= '1', H_Y <= '0', H_G <= '0';
31             WHEN B001100 => V_R <= '0', V_Y <= '0', V_G <= '1', H_R <= '1', H_Y <= '0', H_G <= '0';
32             WHEN C001100 => V_R <= '0', V_Y <= '0', V_G <= '1', H_R <= '1', H_Y <= '0', H_G <= '0';
33             WHEN D010100 => V_R <= '0', V_Y <= '1', V_G <= '0', H_R <= '1', H_Y <= '0', H_G <= '0';
34             WHEN E100100 => V_R <= '1', V_Y <= '0', V_G <= '0', H_R <= '1', H_Y <= '0', H_G <= '0';
35             WHEN F100110 => V_R <= '1', V_Y <= '0', V_G <= '0', H_R <= '1', H_Y <= '1', H_G <= '0';
36             WHEN J100010 => V_R <= '1', V_Y <= '0', V_G <= '0', H_R <= '0', H_Y <= '1', H_G <= '0';
37             WHEN K100100 => V_R <= '1', V_Y <= '0', V_G <= '0', H_R <= '1', H_Y <= '0', H_G <= '0';
38             WHEN G100001 => V_R <= '1', V_Y <= '0', V_G <= '0', H_R <= '0', H_Y <= '0', H_G <= '1';
39             WHEN H100001 => V_R <= '1', V_Y <= '0', V_G <= '0', H_R <= '0', H_Y <= '0', H_G <= '1';
40         END CASE;
41     END PROCESS;
42 END BEHAVIOR;

```


Czym różnią się te układy ?

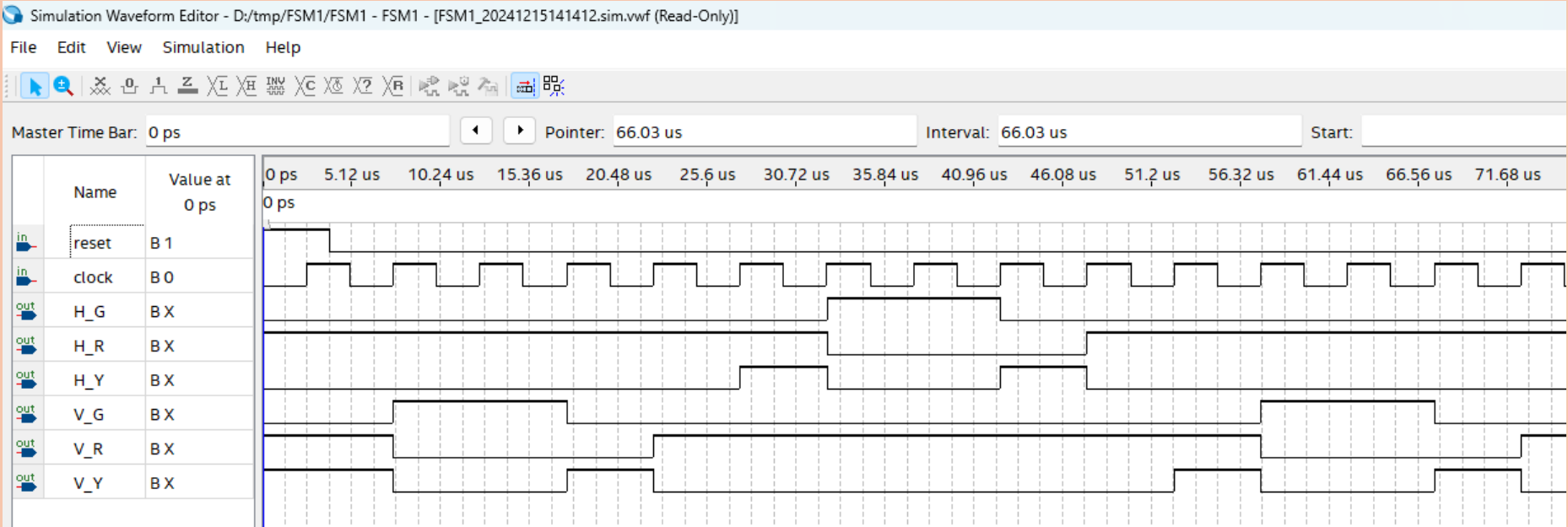
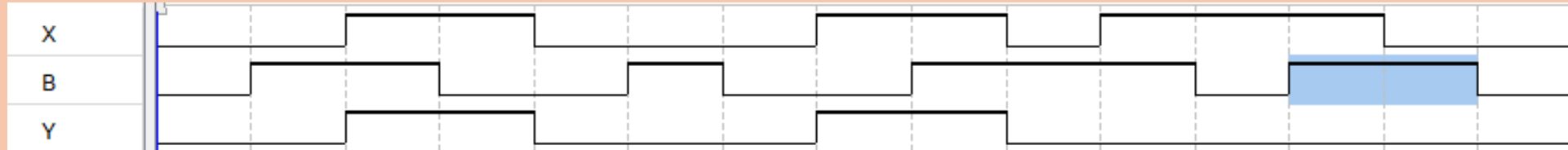


Table of Contents	Flow Summary																												
- Flow Summary - Flow Settings - Flow Non-Default Global Settings - Flow Elapsed Time - Flow OS Summary - Flow Log - Analysis & Synthesis - Fitter - Assembler - Timing Analyzer - EDA Netlist Writer - Flow Messages - Flow Suppressed Messages	<<Filter>> <table><tr><td>Flow Status</td><td>Successful - Sun Dec 15 14:13:33 2024</td></tr><tr><td>Quartus Prime Version</td><td>18.1.0 Build 625 09/12/2018 SJ Lite Edition</td></tr><tr><td>Revision Name</td><td>FSM1</td></tr><tr><td>Top-level Entity Name</td><td>SM1bb</td></tr><tr><td>Family</td><td>Cyclone IV E</td></tr><tr><td>Device</td><td>EP4CE115F29C7</td></tr><tr><td>Timing Models</td><td>Final</td></tr><tr><td>Total logic elements</td><td>14 / 114,480 (< 1 %)</td></tr><tr><td>Total registers</td><td>10</td></tr><tr><td>Total pins</td><td>8 / 529 (2 %)</td></tr><tr><td>Total virtual pins</td><td>0</td></tr><tr><td>Total memory bits</td><td>0 / 3,981,312 (0 %)</td></tr><tr><td>Embedded Multiplier 9-bit elements</td><td>0 / 532 (0 %)</td></tr><tr><td>Total PLLs</td><td>0 / 4 (0 %)</td></tr></table>	Flow Status	Successful - Sun Dec 15 14:13:33 2024	Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition	Revision Name	FSM1	Top-level Entity Name	SM1bb	Family	Cyclone IV E	Device	EP4CE115F29C7	Timing Models	Final	Total logic elements	14 / 114,480 (< 1 %)	Total registers	10	Total pins	8 / 529 (2 %)	Total virtual pins	0	Total memory bits	0 / 3,981,312 (0 %)	Embedded Multiplier 9-bit elements	0 / 532 (0 %)	Total PLLs	0 / 4 (0 %)
Flow Status	Successful - Sun Dec 15 14:13:33 2024																												
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition																												
Revision Name	FSM1																												
Top-level Entity Name	SM1bb																												
Family	Cyclone IV E																												
Device	EP4CE115F29C7																												
Timing Models	Final																												
Total logic elements	14 / 114,480 (< 1 %)																												
Total registers	10																												
Total pins	8 / 529 (2 %)																												
Total virtual pins	0																												
Total memory bits	0 / 3,981,312 (0 %)																												
Embedded Multiplier 9-bit elements	0 / 532 (0 %)																												
Total PLLs	0 / 4 (0 %)																												

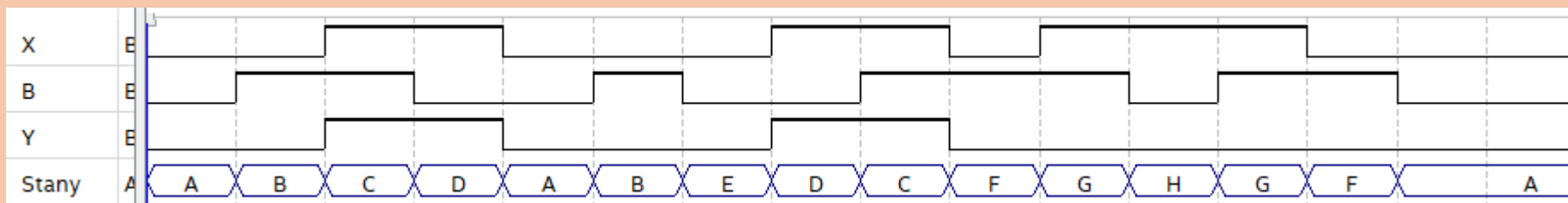
Przykład 2. Zaprojektować układ bramkujący impulsy X. Układ ma przepuszczać każdy impuls X, który jest poprzedzony narastającym zboczem sygnału bramkującego B, jak to pokazuje rysunek:



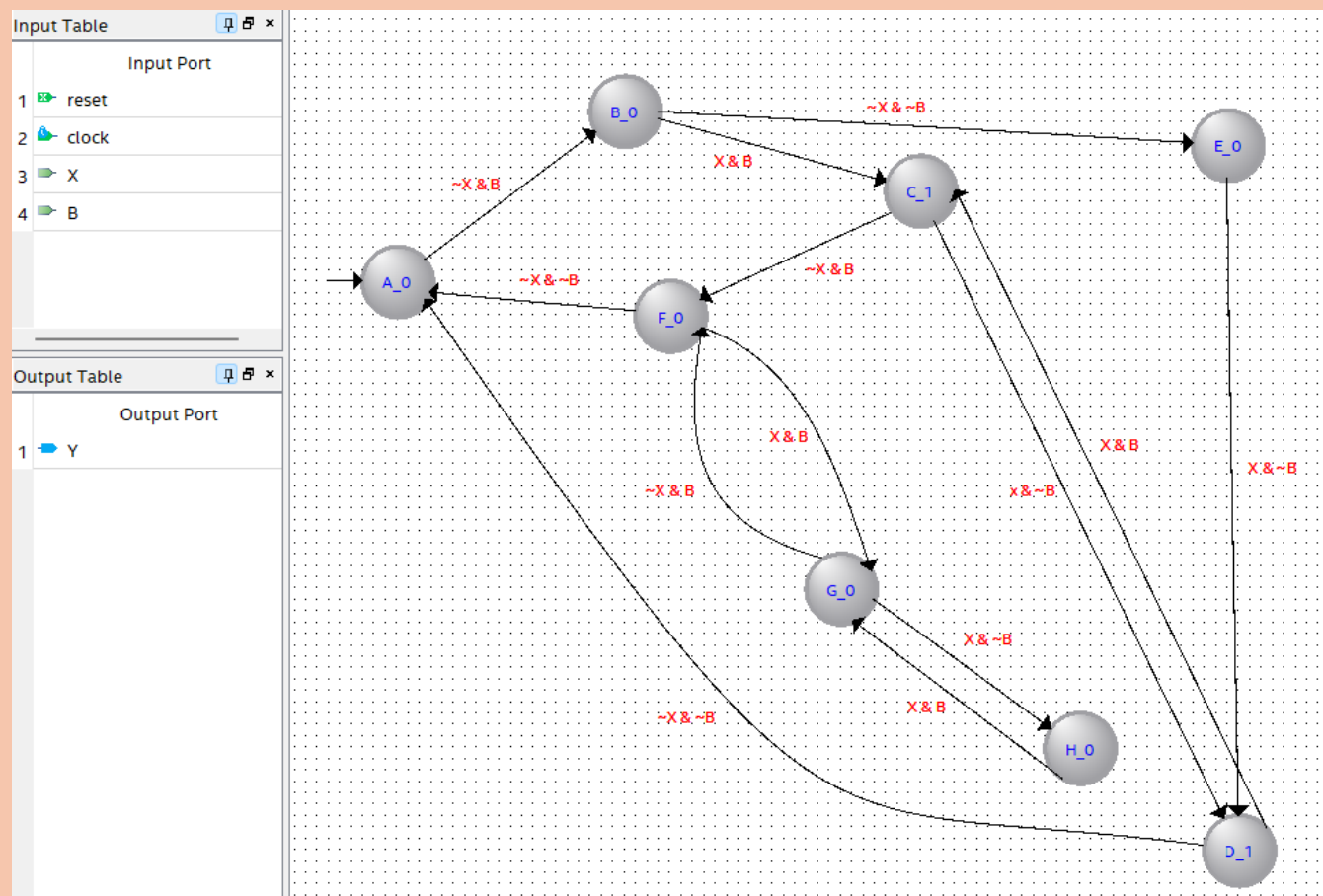
Wyodrębniając niezależne stany:

- zaczynamy od lewej strony,
- każdej kombinacji sygnałów wejściowych i wyjściowych nadajemy nową nazwę stanu,
- jeżeli kombinacja sygnałów powtarza się, ale występuje ona po innym stanie – taką sytuację traktujemy jako nowy stan,
- na podstawie wyznaczonego zestawu stanów i przebiegów czasowych tworzymy graf.

Nie przejmujemy się zbyt mocno, ryzykiem zdefiniowania stanów nadmiarowych. Kompilator zminimalizuje graf łącząc stany niesprzeczne.



każda kombinacja wejść./wyjść. stanowi stanu - powtarzające się po innym stanowią nowy

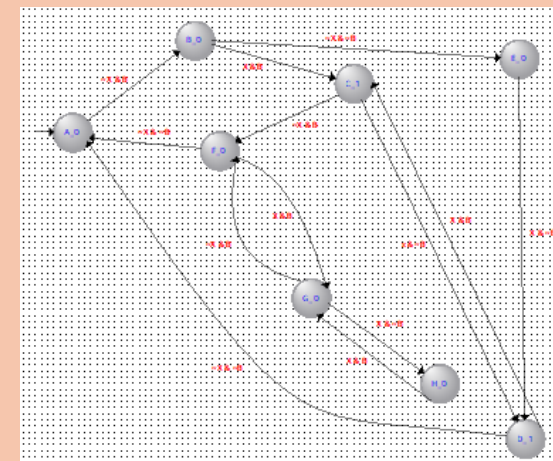


Kod układu wygenerowany automatycznie jest następujący:

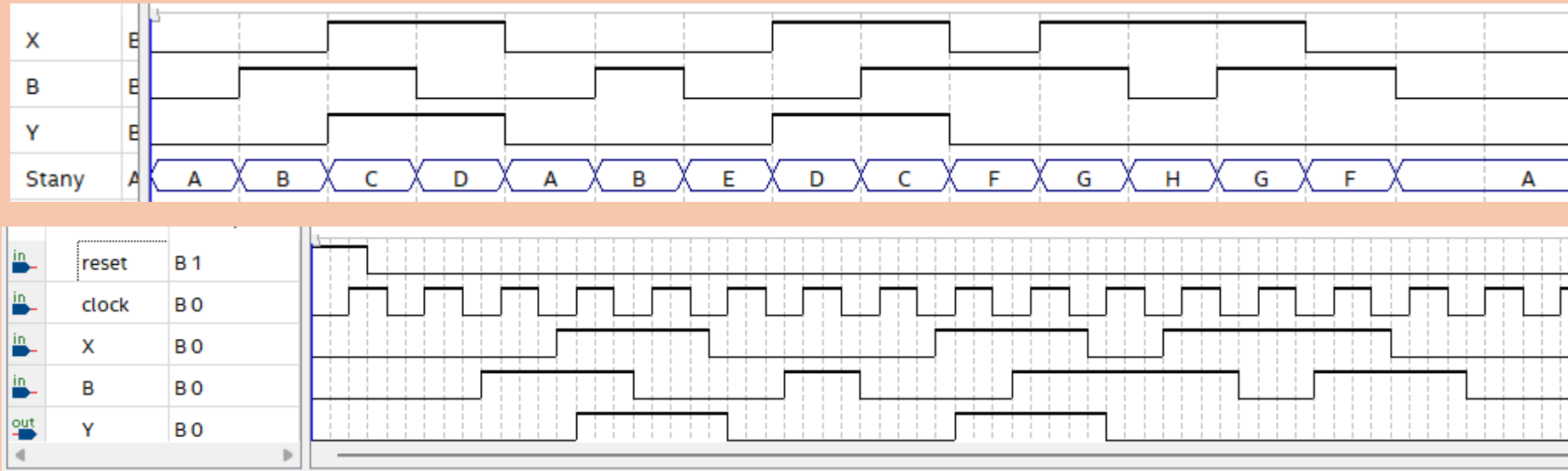
```

1  LIBRARY ieee; USE ieee.std_logic_1164.all;
2  ENTITY SM1b IS
3    PORT ( clock : IN STD_LOGIC; reset, X, B : IN STD_LOGIC := '0'; Y : OUT STD_LOGIC);
4  END SM1b;
5  ARCHITECTURE BEHAVIOR OF SM1b IS
6    TYPE type_fstate IS (B_0,A_0,C_1,H_0,E_0,F_0,G_0,D_1);    SIGNAL fstate : type_fstate; SIGNAL reg_fstate : type_fstate;
7  BEGIN
8    PROCESS (clock,reg_fstate)
9    BEGIN
10     IF rising_edge(clock) THEN fstate <= reg_fstate; END IF;
11   END PROCESS;
12   PROCESS (fstate,reset,X,B)
13   BEGIN
14     IF (reset='1') THEN reg_fstate <= A_0; Y <= '0';
15   ELSE Y <= '0';
16     CASE fstate IS
17       WHEN B_0 =>
18         IF ((X = '1') AND (B = '1')) THEN reg_fstate <= C_1; ELSIF ((NOT(X = '1')) AND (NOT(B = '1'))) THEN reg_fstate <= E_0;
19       ELSE reg_fstate <= B_0; END IF; Y <= '0'; -- podtrzymanie stanu
20       WHEN A_0 =>
21         IF ((NOT(X = '1')) AND (B = '1')) THEN reg_fstate <= B_0; ELSE reg_fstate <= A_0; END IF; Y <= '0';
22       WHEN C_1 =>
23         IF ((NOT(X = '1')) AND (B = '1')) THEN reg_fstate <= F_0; ELSIF ((X = '1') AND (NOT(B = '1'))) THEN reg_fstate <= D_1;
24       ELSE reg_fstate <= C_1; END IF; Y <= '1';
25       WHEN H_0 =>
26         IF ((X = '1') AND (B = '1')) THEN reg_fstate <= G_0; ELSE reg_fstate <= H_0; END IF; Y <= '0';
27       WHEN E_0 =>
28         IF ((X = '1') AND (NOT(B = '1'))) THEN reg_fstate <= D_1; ELSE reg_fstate <= E_0; END IF; Y <= '0';
29       WHEN F_0 =>
30         IF ((NOT(X = '1')) AND (NOT(B = '1'))) THEN reg_fstate <= A_0; ELSIF (((X = '1') AND (B = '1'))) THEN reg_fstate <= G_0;
31       ELSE reg_fstate <= F_0; END IF; Y <= '0';
32       WHEN G_0 =>
33         IF ((NOT(X = '1')) AND (B = '1')) THEN reg_fstate <= F_0; ELSIF (((X = '1') AND NOT((B = '1')))) THEN reg_fstate <= H_0;
34       ELSE reg_fstate <= G_0; END IF; Y <= '0';
35       WHEN D_1 =>
36         IF ((NOT(X = '1')) AND (NOT(B = '1'))) THEN reg_fstate <= A_0; ELSIF (((X = '1') AND (B = '1'))) THEN reg_fstate <= C_1;
37       ELSE reg_fstate <= D_1; END IF; Y <= '1';
38     WHEN OTHERS => Y <= 'X';
39   END CASE;
40   END IF;
41   END PROCESS;
42 END BEHAVIOR;

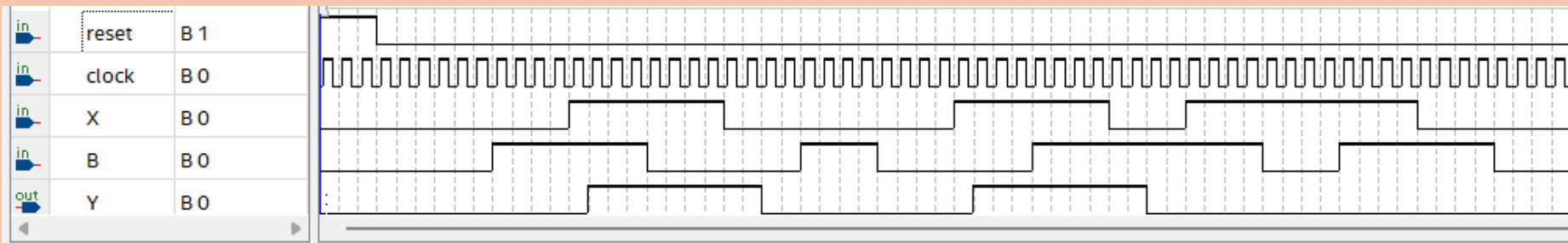
```



A wyniki symulacji potwierdzają pracę układu zgodnie z założeniami:

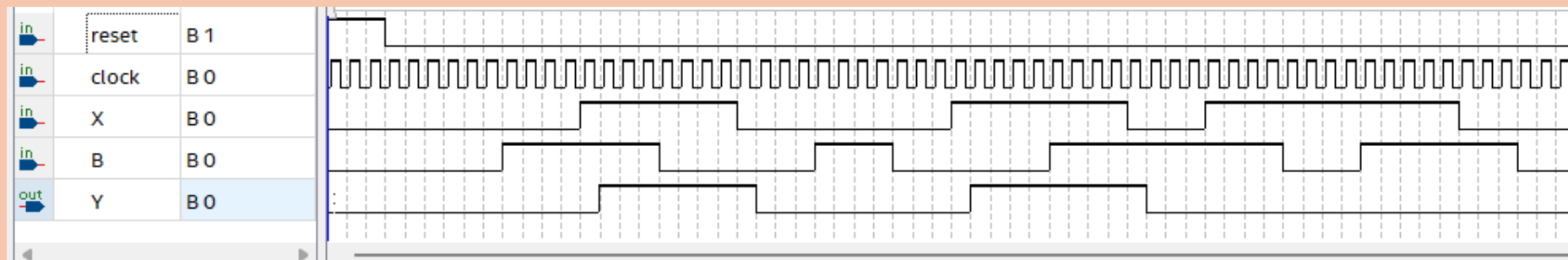
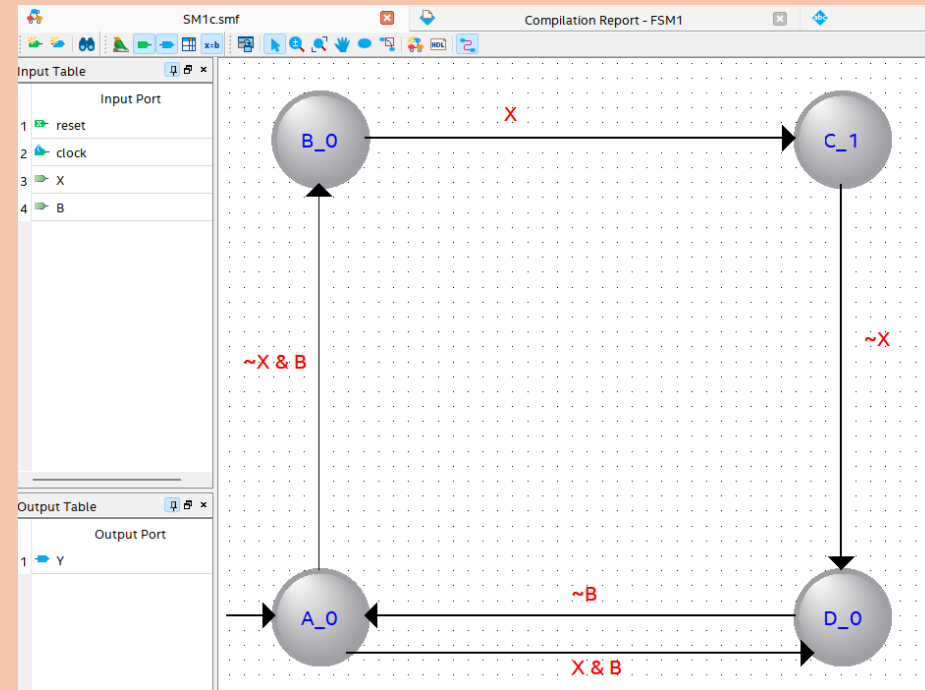
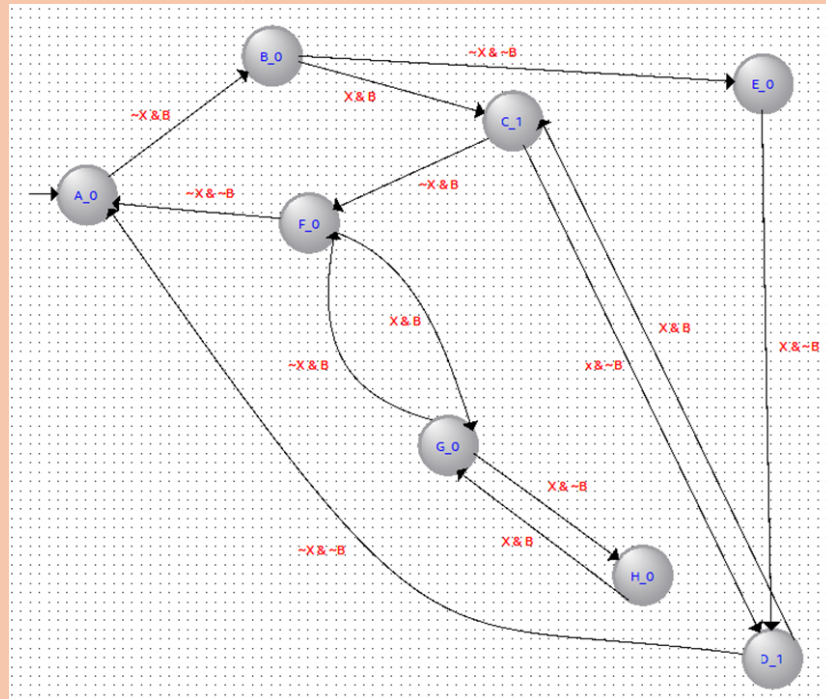


Zwiększenie częstotliwości sygnału zegarowego nie zmienia charakteru pracy układu ani nie zmniejsza opóźnień wnoszonych przez układ, których źródłem są głównie części kombinacyjne FSM:



Graf źródłowy można uprościć sprowadzając układ do 4 stanów.

Techniki minimalizacji grafów, są szeroko opisane w literaturze, w ramach wykładu nie będą one omawiane.



Zaliczenie cząstkowe wykładu nr 14 z UCiP sem.1, Elektromobilność

Test składa się z 10 zadań. Każde zadanie posiada 3 odpowiedzi. Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c). Zadania będą wyświetlane po kolei co 1 minutę. Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym. Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Zaliczenie końcowe wykładu z UCiP sem.1, Elektromobilność – terminy 1 i 2

Test składa się z 100 zadań. Każde zadanie posiada 3 odpowiedzi. Tylko jedna odpowiedź jest poprawna.

Na arkuszu odpowiedzi proszę zaznaczyć poprawną odpowiedź (a lub b lub c). Zadania będą wyświetlane po kolei co 1 minutę. Jeżeli w zadaniu znajduje się błąd lub niejednoznaczność – zadanie zalicza się wszystkim piszącym. Można korzystać z dowolnych materiałów – z wyjątkiem pomocy kolegów i urządzeń mobilnych.

Arkusz odpowiedzi proszę czytelnie podpisać.

Powodzenia