Logika cyfrowa

Wykład 1: wprowadzenie

Marek Materzok

24 lutego 2021

Bramki logiczne

Bramki logiczne

bufor

i	o
0	0
1	1

not

i	o
0	1
1	0

and

i_1	i_2	o
0	0	0
0	1	0
1	0	0
1	1	1

or

i_1	i_2	o
0	0	0
0	1	1
1	0	1
1	1	1

xor

i_1	i_2	o
0	0	0
0	1	1
1	0	1
1	1	0

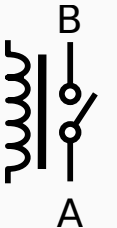
Budowa bramek

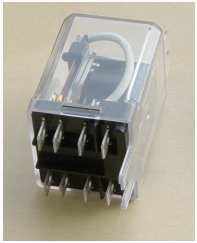
Sterowane elementy przełączające:

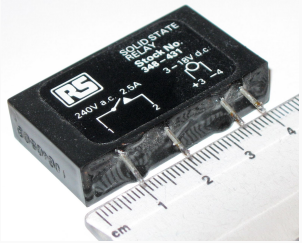
- Przekaźniki
  - powolne, ciężkie, nieefektywne, proste koncepcyjnie
- Lampy elektronowe
  - duże, wymagają wysokich napięć, wyszły z użytku
- Tranzystory bipolarne (RTL, DTL, TTL)
  - w XXI wieku praktycznie wyszły z użytku
- Tranzystory polowe (NMOS, CMOS)
  - NMOS: podobna do TTL, wyszła z użytku
  - CMOS: zużycie energii proporcjonalne do szybkości przełączeń, symetryczne poziomy napięć, duża impedancja wejściowa, mała wyjściowa

Przekaźniki

- prąd w cewce elektromagnesu przełącza przełącznik
- na obrazku przekaźnik SPST (jeden styk, jedna pozycja aktywna)
- wciąż używane do przełączania dużych prądów i napięć

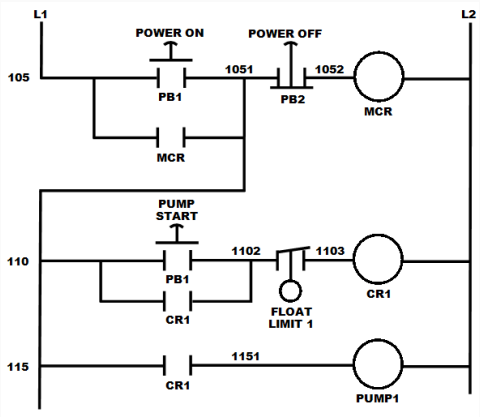






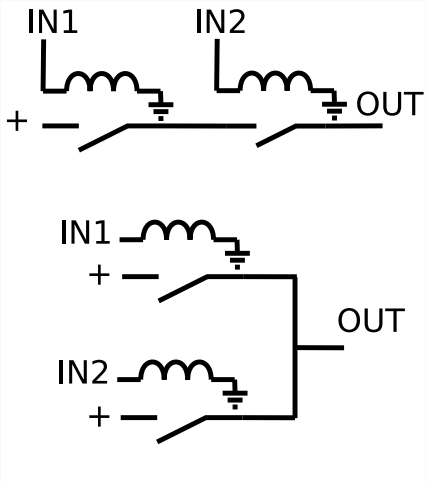
Diagramy drabinkowe, sterowniki PLC

Automatycy wciąż myślą „wirtualnymi przekaźnikami”:



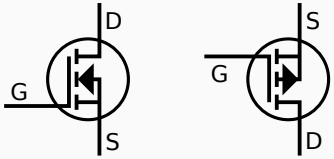
Bramki przekaźnikowe

- Bramki and i or
- Funkcja wzmacniająca: prąd wejść nie płynie do wyjść, tylko przełącza zasilanie do wyjścia
- Półprzewodnikowe bramki są analogiczne!

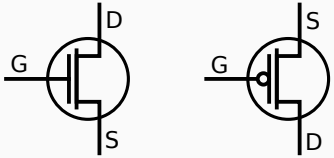


Tranzystory polowe z bramką izolowaną (MOSFET)

- Kanał między źródłem (S) i drenem (D)
- Kanał typu N lub P
  - Analogicznie jak w BJT
- Domyślnie wyłączone, napięcie GS włącza
- Bramką nie płynie prąd stały
  - Ale bramka ma pojemność

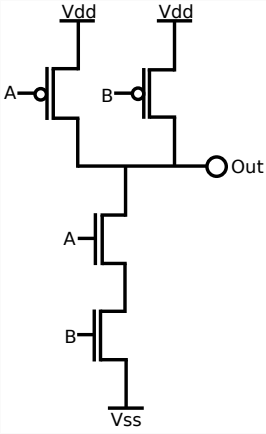


rysunek uproszczony:



Technologia CMOS

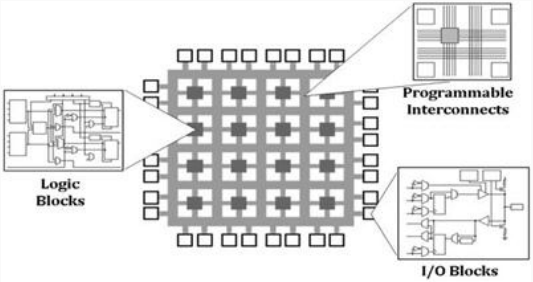
- Complementary MOS
- Tranzystory N-ch łączą z masą, P-ch z zasilaniem
- Pobiera istotny prąd tylko podczas przełączania
- Symetryczne poziomy napięć i prądy wyjść
- Najpowszechniejsza współczesna technologia
- Po prawej: bramka NAND



Układy programowalne (PLD), FPGA

Programmable Logic Devices

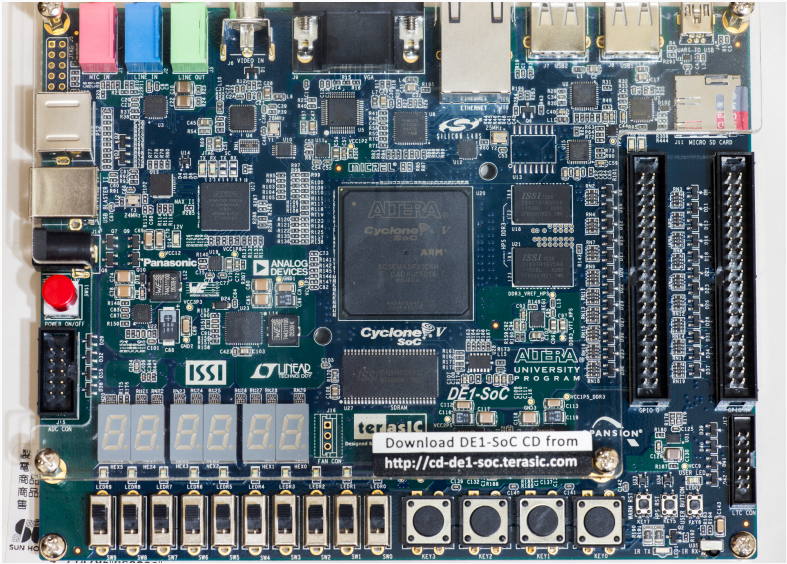
- PAL (programowanie funkcji logicznych w PROM)
- CPLD (funkcje logiczne + stan + interconnect)
- FPGA (siatka bloków logicznych z pamięcią)



Zalety FPGA

- Wydajność
  - Dla zadań równoległych wygrywa z CPU
  - Elastyczniejsze niż GPU
  - Szybki interfejs do zewnętrznych elementów
- Czas do wprowadzenia na rynek
  - Szybszy niż dla układów specjalizowanych (ASIC)
- Koszt
  - Dla niewielkiej produkcji tańszy niż ASIC
- Łatwość utrzymania
  - Reprogramowalne w locie
  - Można poprawiać błędy i dodawać funkcjonalność w istniejących produktach
- Niezawodność
  - Równoległe komponenty nie wpływają na siebie

Płytki deweloperskie FPGA



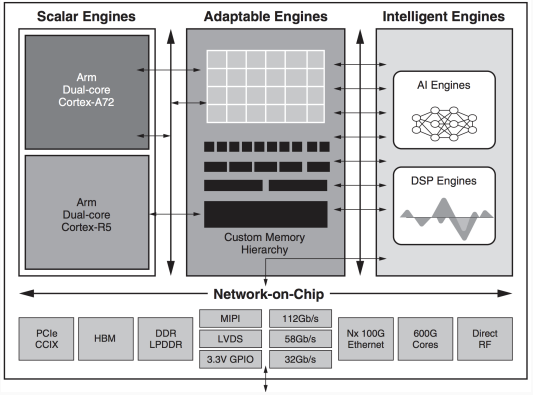
Zastosowania FPGA

Wszędzie, gdzie można by użyć specjalizowanego układu, ale koszt byłby zbyt duży:

- Prototypowanie układów
- „Glue logic" między układami specjalizowanymi
- Przetwarzanie sygnałów (badania medyczne, przemysł, multimedia)
- Telekomunikacja (bezprzewodowa, światłowodowa, routing)
- Obliczenia równoległe (badania naukowe, kryptowaluty)
- Sterowanie procesami przemysłowymi

Machine learning

- Sieci konwolucyjne, LSTM...
- 40% zgłoszonych artykułów na FPGA'19!
- Nadchodzą dedykowane układy do ML (Xilinx Versal)



Łaziki na Marsie

- Kontrola lądowania
- Sterowanie silnikami
- Przetwarzanie obrazu
- Tryb oszczędzania energii „dream mode”
- Układy Xilinx space-grade

JCREW – zakłócanie bomb

- Joint Counter RCIED (Radio Controlled Improvised Explosive Device) Electronic Warfare
- Wykrywanie i zakłócanie w czasie rzeczywistym sygnałów mogących wyzwolić improwizowaną bombę
  GSM, pilot od drzwi garażowych, itp.
- Układy Xilinx Virtex 6Q, defense-grade

Projektowanie układów cyfrowych

Budowa komputera

- zasilacz
- płyta główna
  zawiera mikroprocesor
- karty rozszerzeń
  np. karta sieciowa, graficzna
- urządzenia wejścia/wyjścia
  np. dysk twardy, klawiatura

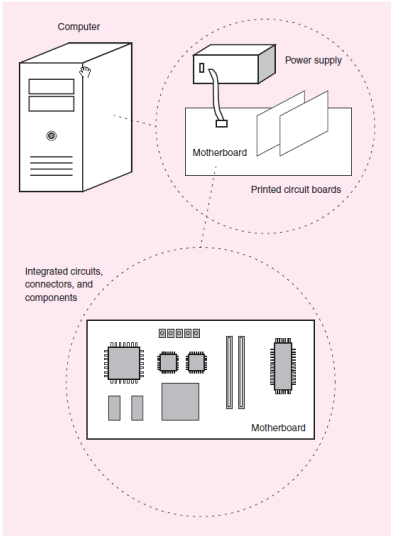


Figure 1.4 A digital hardware system (Part a).

Poziomy abstrakcji w systemie komputerowym

- język programowania (np. zmienne)
- system operacyjny (np. syscalle)
- architektura (np. instrukcje)
- mikroarchitektura (np. kontrolery)
- logika cyfrowa (np. sumatory)
- elementy cyfrowe (np. bramki)
- obwody analogowe (np. wzmacniacze)
- elementy elektron. (np. tranzystory)
- fizyka (np. pole elektromagnetyczne)

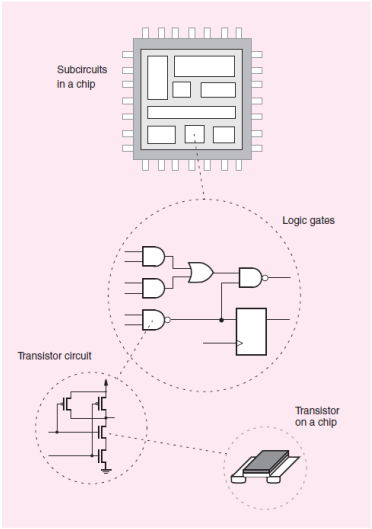


Figure 1.4 A digital hardware system (Part *b*).

Dyscyplina

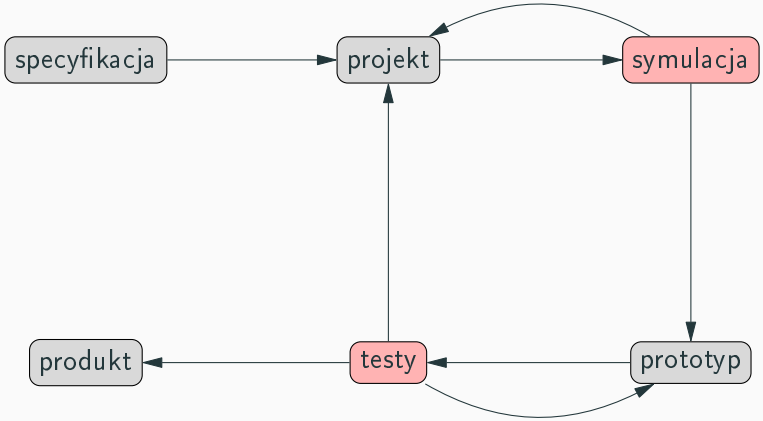
Ograniczenie możliwości projektowych celem podniesienia poziomu abstrakcji.

- Inżynieria: śruby i nakrętki o standardowych średnicach i gwintach
- Programowanie: warunki, funkcje i pętle zamiast instrukcji skoku warunkowego
- Układy cyfrowe: dwa poziomy napięć (niski i wysoki) zamiast napięć analogowych

Zarządzanie złożonością

- Hierarchia
  podział systemu na moduły
- Modularność
  dobrze określone funkcje modułów oraz połączenia między nimi
- Regularność
  wielokrotne użycie wspólnych modułów

Proces projektowania układu cyfrowego



Narzędzia CAD

- Computer Aided Design – projektowanie wspomagane komputerowo
- Projektowanie: języki opisu sprzętu
- Testowanie: symulatory, weryfikacja formalna
- Prototypowanie/produkcja: synteza sprzętu

Języki opisu sprzętu (HDL)

- Języki o semantyce odpowiedniej dla układów logicznych (jawny przepływ danych, obliczenia równoległe, moduły)
- Zastosowania: projektowanie, modelowanie, testowanie i weryfikacja sprzętu
- Popularne: Verilog, SystemVerilog, VHDL
- Inne: Bluespec, Lava, Cλash (Haskell), HardCaml (OCaml), Chisel, SpinalHDL (Scala), Migen, nMigen (Python)

Synteza wysokiego poziomu (HLS)

- Synteza układów z abstrakcyjnego, algorytmicznego kodu
- Języki źródłowe: podzbiór C, C++, SystemC
- OpenCL – wspólny język dla GPU i FPGA
- Szybsze projektowanie, ale niższa wydajność

Verilog – historia

- 1983 – Verilog powstaje w Gateway Design Automation
  - zastosowania: prototypowanie i modelowanie, projektowanie sprzętu odbywało się w oddzielnych programach CAD
- 1985-90 – synteza sprzętu z Veriloga
- 1995 – standaryzacja
  - IEEE 1364-1995 – początkowy standard
  - IEEE 1364-2001 – liczby ze znakiem, generowanie
  - IEEE 1364-2005 – doprecyzowanie semantyki, `uwire`
- 2005 – SystemVerilog jako rozszerzenie Veriloga
  - IEEE 1800-2005
- 2009 – SystemVerilog jako niezależny język
  - IEEE 1800-2009
  - IEEE 1800-2017

Narzędzia open source dla Veriloga

- Yosys – synteza sprzętu, weryfikacja formalna
- Icarus Verilog – symulator („interpreter”)
- Verilator – kompilacja do C++/SystemC, wysoka wydajność

DigitalJS

- `http://digitaljs.tilk.eu/`
- Kod: `https://github.com/tilk/digitaljs` (430⋆)
- Symulator obwodów dla SystemVeriloga
- Poszukiwani programiści, możliwe prace lic/inż

DigitalJS – nowości

Changelog od poprzedniej edycji przedmiotu:

- Panel wejść/wyjść
- Karty dla wielu plików źródłowych
- Skryptowanie w języku Lua
- Programowalne kodowanie/dekodowanie sygnałów
- Podświetlenie adresowanych komórek pamięci

Program wykładu

Program wykładu

Tercja 1: układy kombinacyjne

- Algebra Boole'a
- Reprezentacja liczb i arytmetyka
- Podstawowe układy kombinacyjne

Tercja 2: układy sekwencyjne

- Podstawowe układy sekwencyjne
- Synchroniczne układy sekwencyjne
- Projektowanie automatów skończonych

Tercja 3: procesor jako układ sekwencyjny

- Architektura RISC V (RV32I)
- Mikroarchitektury: jednocyklowa, wielocyklowa i potokowa

Literatura

- S. Brown, Z. Vranesic – Fundamentals of Digital Logic with Verilog Design
- D. Harris, S. Harris – Digital Design and Computer Architecture
- J. Bhasker, A Verilog HDL Primer
- S. Palnitkar, Verilog HDL – A Guide to Digital Design and Synthesis
- T. Kuphaldt, Lessons in Electric Circuits, Vol. IV – Digital
- Standard IEEE 1364-2005 (Verilog 2005)
- Standard IEEE 1800-2017 (SystemVerilog 2017)

Zasady zaliczania

Składowe ćwiczenio-pracowni:

- Listy zadań z teorii (dającej fundamenty praktyce!)
- Zadania praktyczne z implementacji układów w Verilogu

Wykład kończy się egzaminem.

Zasady – zadania ćwiczeniowe

- Obowiązuje system deklaracji znany z innych przedmiotów.
- Jeśli nie podano inaczej, to zadanie jest warte jeden punkt.
- W przypadku błędnego rozwiązania, zadanie zostaje skreślone z deklaracji.
- Jeśli student zadeklarował zadanie, którego w oczywisty sposób nie potrafi rozwiązać, przewiduje się karę w postaci wykreślenia pewnej ilości punktów z deklaracji.
- Jeśli dodatkowo student wykazuje brak znajomości podstawowych pojęć pojawiających się na liście, to przewidywana jest kara w postaci punktów ujemnych.
- Brak obecności w momencie losowania osoby prezentującej rozwiązanie zadania oznacza utratę punktów za to zadanie.

Zasady covidowe

- Deklaracje polegają na wypełnieniu formularza online, dostępnego przez Canvas, przed rozpoczęciem ćwiczeń.
- Ćwiczenia odbywają się online na platformie MS Teams, w ustalonych w planie zajęć godzinach.
- Student musi być przygotowany do prezentowania zadania w sposób interaktywny – np. przy użyciu tabletu graficznego, kamery i papieru, lub innej wybranej przez siebie metody.
- Przedłużające się problemy techniczne przy rozwiązywaniu zadania oznaczają skreślenie zadania z deklaracji.

Zasady – zadania praktyczne

- Językiem wykorzystywanym podczas zajęć jest SystemVerilog (a dokładniej: syntezowalny podzbiór SystemVerilog 2012 obsługiwany przez Yosys w wersji 0.9).
- Program musi syntezować się poprawnie w narzędziu DigitalJS.
- Student może pracować nad rozwiązaniem przy użyciu dowolnie wybranych przez siebie narzędzi, ale prowadzący będzie zakładać, że interpretacja programu przez DigitalJS jest zgodna z intencją studenta.
- Rozwiązania należy wysyłać w wyznaczonym terminie przez system Web-CAT (dostęp przez Canvas).
- Punktacja obejmuje poprawność i jakość rozwiązania (czytelność, rozmiar i efektywność układu, etc.)
- Obowiązuje limit 10 zgłoszeń – przed zgłoszeniem rozwiązania należy dołożyć starań, aby rozwiązanie było poprawne.

Zasady – samodzielność pracy

- Wspólne rozwiązywanie zadań ćwiczeniowych nie jest zabronione, jednak każdy student, który zadeklarował zadanie, musi być w stanie je samodzielnie zreferować.
- Współpraca przy zadaniach praktycznych jest dozwolona, jednak przesłane rozwiązania muszą być napisane całkowicie samodzielnie. Kopiowanie kodu od innych studentów oraz z Internetu jest niedopuszczalne.
- Sytuacje wątpliwe są rozstrzygane na korzyść studenta.
- Dodatkowe informacje w „Kodeksie samodzielnego studiowania” (w systemie Canvas).

Zasady – korespondencja

- Ogólne pytania dotyczące list / zajęć powinny być zadawane przez kanał przedmiotu na MS Teams, tak by wszyscy studenci mogli na tym skorzystać.
- Pytania o charakterze niepublicznym (np. korekcja liczby punktów za listę) proszę wysyłać przez e-mail lub prywatnie na MS Teams.
- Ogłoszenia związane z organizacją przedmiotu będą pojawiać się jednocześnie na MS Teams i w systemie Canvas.

Algebra Boole'a

Algebra dwóch wartości

Będziemy zajmować się operacjami (spójnikami) na dwuelementowym zbiorze $\mathbb{B}$:

- 1 – też: prawda (T, $\top$), stan wysoki (H)
- 0 – też: fałsz (F, $\bot$), stan niski (L)

Koniunkcja

Spójnik binarny (dwuargumentowy) $x \wedge y$
(czasem: $x\&y$, $x \cdot y$, xy, *iloczyn logiczny*):

- $0 \wedge 0 = 0$
- $0 \wedge 1 = 0$
- $1 \wedge 0 = 0$
- $1 \wedge 1 = 1$

Inaczej: $x \wedge y = 1$ wtw zarówno $x = 1$ oraz $y = 1$, w przeciwnym wypadku $x \wedge y = 0$.

Dysjunkcja

Spójnik binarny (dwuargumentowy) $x \vee y$
(czasem: $x \mid y$, $x + y$, *alternatywa*, *suma logiczna*):

- $0 \vee 0 = 0$
- $0 \vee 1 = 1$
- $1 \vee 0 = 1$
- $1 \vee 1 = 1$

Inaczej: $x \vee y = 0$ wtw zarówno $x = 0$ oraz $y = 0$, w przeciwnym wypadku $x \vee y = 1$.

Negacja

Spójnik unarny (jednoargumentowy) $\neg x$
(czasem: $\tilde{}x$, $!x$, $\bar{x}$):

- $\neg 0 = 1$
- $\neg 1 = 0$

Negacja jest *niemonotoniczna*, w przeciwieństwie do koniunkcji i dysjunkcji.

Wiązanie spójników

- Negacja wiąże najmocniej
- Słabiej wiąże koniunkcja
- Najsłabiej wiąże dysjunkcja

Przykłady:

Wiązanie spójników

- Negacja wiąże najmocniej
- Słabiej wiąże koniunkcja
- Najsłabiej wiąże dysjunkcja

Przykłady:

- $x \vee y \wedge z =$

Wiązanie spójników

- Negacja wiąże najmocniej
- Słabiej wiąże koniunkcja
- Najsłabiej wiąże dysjunkcja

Przykłady:

- $x \vee y \wedge z = x \vee (y \wedge z)$

Wiązanie spójników

- Negacja wiąże najmocniej
- Słabiej wiąże koniunkcja
- Najsłabiej wiąże dysjunkcja

Przykłady:

- $x \vee y \wedge z = x \vee (y \wedge z)$
- $x \wedge \neg y \vee z =$

Wiązanie spójników

- Negacja wiąże najmocniej
- Słabiej wiąże koniunkcja
- Najsłabiej wiąże dysjunkcja

Przykłady:

- $x \vee y \wedge z = x \vee (y \wedge z)$
- $x \wedge \neg y \vee z = (x \wedge (\neg y)) \vee z$

Wiązanie spójników

- Negacja wiąże najmocniej
- Słabiej wiąże koniunkcja
- Najsłabiej wiąże dysjunkcja

Przykłady:

- $x \vee y \wedge z = x \vee (y \wedge z)$
- $x \wedge \neg y \vee z = (x \wedge (\neg y)) \vee z$
- $x \wedge y \vee \neg z \wedge \neg q =$

Wiązanie spójników

- Negacja wiąże najmocniej
- Słabiej wiąże koniunkcja
- Najsłabiej wiąże dysjunkcja

Przykłady:

- $x \vee y \wedge z = x \vee (y \wedge z)$
- $x \wedge \neg y \vee z = (x \wedge (\neg y)) \vee z$
- $x \wedge y \vee \neg z \wedge \neg q = (x \wedge y) \vee ((\neg z) \wedge (\neg q))$

Funkcje logiczne wielu argumentów

- Funkcje $f(x_1, \ldots, x_n) = \Phi$, gdzie Φ jest wyrażeniem algebry Boole'a ze zmiennymi ze zbioru $\{x_1, \ldots, x_n\}$
- Piszemy: $f : \mathbb{B}^n \to \mathbb{B}$
- Funkcje w sensie „matematycznym", deklaratywnym (zależność wyniku od argumentów), nie „programistycznym", imperatywnym (ciąg instrukcji do wykonania)
- Przykłady:
  - $f(x) = x$
  - $f(x, y) = x \vee \neg y$
  - $f(x, y, z) = (x \wedge y) \vee (x \wedge z)$

Tabelki zero-jedynkowe

Funkcje logiczne można opisać za pomocą tabelki.

Przykład dla $f(x, y) = x \vee \neg y$:

x	y	$x \vee \neg y$
0	0	1
0	1	0
1	0	1
1	1	1

Tabelki zero-jedynkowe – konstrukcja

Tabelkę można konstruować dla kolejnych podwyrażeń.

Przykład dla $f(x, y, z) = (x \wedge y) \vee (x \wedge z)$:

x	y	z	$x \wedge y$	$x \wedge z$	$f(x, y, z)$
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Tabelki zero-jedynkowe – konstrukcja

Tabelkę można konstruować dla kolejnych podwyrażeń.

Przykład dla $f(x, y, z) = (x \wedge y) \vee (x \wedge z)$:

x	y	z	$x \wedge y$	$x \wedge z$	$f(x, y, z)$
0	0	0	0		
0	0	1	0		
0	1	0	0		
0	1	1	0		
1	0	0	0		
1	0	1	0		
1	1	0	1		
1	1	1	1		

Tabelki zero-jedynkowe – konstrukcja

Tabelkę można konstruować dla kolejnych podwyrażeń.

Przykład dla $f(x, y, z) = (x \wedge y) \vee (x \wedge z)$:

x	y	z	$x \wedge y$	$x \wedge z$	$f(x, y, z)$
0	0	0	0	0	
0	0	1	0	0	
0	1	0	0	0	
0	1	1	0	0	
1	0	0	0	0	
1	0	1	0	1	
1	1	0	1	0	
1	1	1	1	1	

Tabelki zero-jedynkowe – konstrukcja

Tabelkę można konstruować dla kolejnych podwyrażeń.

Przykład dla $f(x, y, z) = (x \wedge y) \vee (x \wedge z)$:

x	y	z	$x \wedge y$	$x \wedge z$	$f(x, y, z)$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

Równość ekstensjonalna funkcji

Funkcje logiczne n-arne f i g są *ekstensjonalnie równe*
wtw
dla każdego przypisania zmiennym x_1 do x_n wartości ze zbioru $\{0, 1\}$ zachodzi $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)$.

Gdy funkcje $f(x_1, \ldots, x_n) = \Phi_f$ i $g(x_1, \ldots, x_n) = \Phi_g$ są ekstensjonalnie równe, mówimy, że wyrażenia algebry Boole'a Φ_f i Φ_g są *logicznie równoważne*.

Sprawdzanie równości funkcji – tabelką

Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:

x	y	$\neg x$	$\neg y$	$x \vee y$	$f(x, y)$	$g(x, y)$
0	0					
0	1					
1	0					
1	1					

Sprawdzanie równości funkcji – tabelką

Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:

x	y	$\neg x$	$\neg y$	$x \vee y$	$f(x, y)$	$g(x, y)$
0	0	1				
0	1	1				
1	0	0				
1	1	0				

Sprawdzanie równości funkcji – tabelką

Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:

x	y	$\neg x$	$\neg y$	$x \vee y$	$f(x, y)$	$g(x, y)$
0	0	1	1			
0	1	1	0			
1	0	0	1			
1	1	0	0			

Sprawdzanie równości funkcji – tabelką

Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:

x	y	$\neg x$	$\neg y$	$x \vee y$	$f(x, y)$	$g(x, y)$
0	0	1	1	0		
0	1	1	0	1		
1	0	0	1	1		
1	1	0	0	1		

Sprawdzanie równości funkcji – tabelką

Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:

x	y	$\neg x$	$\neg y$	$x \vee y$	$f(x, y)$	$g(x, y)$
0	0	1	1	0	1	
0	1	1	0	1	0	
1	0	0	1	1	0	
1	1	0	0	1	0	

Sprawdzanie równości funkcji – tabelką

Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:

x	y	$\neg x$	$\neg y$	$x \vee y$	$f(x, y)$	$g(x, y)$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Diagramy Venna

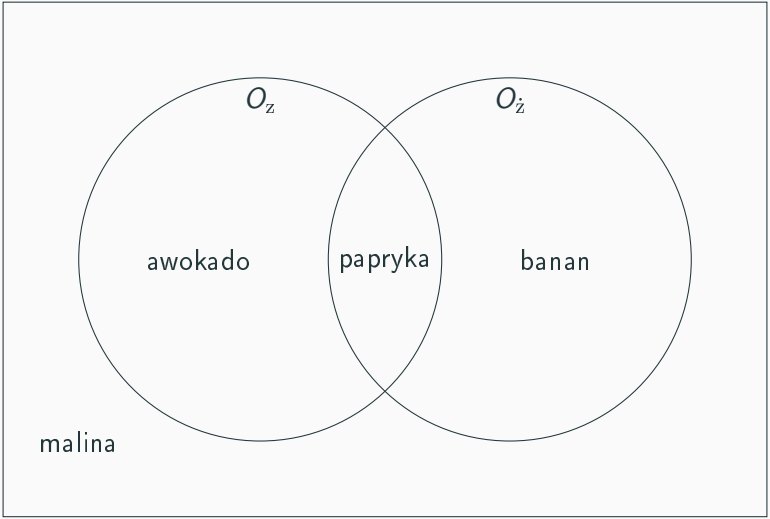
Diagramy Venna

- Diagramy przedstawiające wszystkie zależności pomiędzy skończoną liczbą zbiorów
- Nakładające się zamknięte krzywe (np. okręgi) reprezentują zbiory
- Obszary nakładania się krzywych reprezentują przecięcia zbiorów
- Obszar poza krzywymi reprezentuje dopełnienie sumy zbiorów (do większego zbioru „uniwersum”)

Diagramy Venna – przykład

- Uniwersum: zbiór nazw owoców
  $O = \{\text{awokado}, \text{papryka}, \text{banan}, \text{malina}, \dots\}$
- Zbiór nazw owoców zielonych
  $O_{\text{z}} = \{\text{awokado}, \text{papryka}, \dots\}$
- Zbiór nazw owoców żółtych
  $O_{\dot{\text{z}}} = \{\text{banan}, \text{papryka}, \dots\}$

Diagramy Venna – przykład

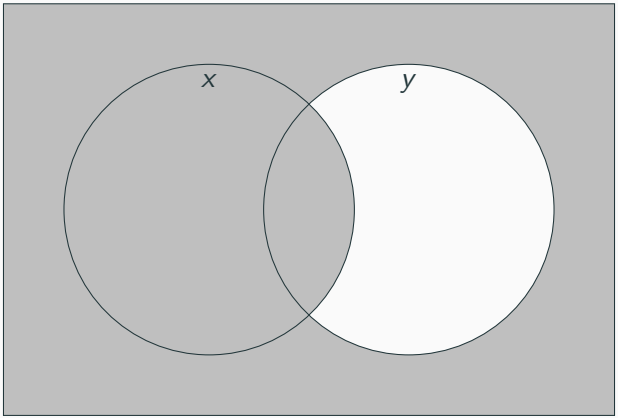


Diagramy Venna dla funkcji logicznych

- Uniwersum: zbiór wszystkich wartościowań argumentów
  $\mathbb{B}^n$
- Zbiory X_i: zbiory tych wartościowań, gdzie i-ty argument ma wartość 1
  $X_i = \{(x_1, \ldots, x_n) \in \mathbb{B}^n \mid x_i = 1\}$
- Dla zadanej funkcji $f : \mathbb{B}^n \to \mathbb{B}$ zamalowujemy wartościowania, dla których funkcja ma wartość 1

Diagram Venna – przykład

Funkcja $f(x, y) = x \vee \neg y$:

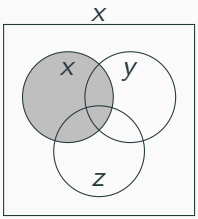


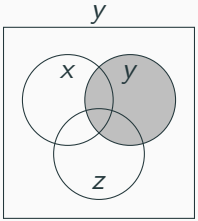
Konstrukcja diagramu Venna

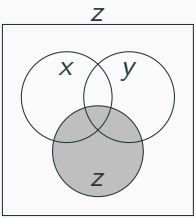
- Koniunkcja – przecięcie diagramów
- Dysjunkcja – suma diagramów
- Negacja – dopełnienie diagramu

Konstrukcja diagramu Venna – przykład

Funkcja $f(x, y, z) = (x \wedge y) \vee (x \wedge z)$:



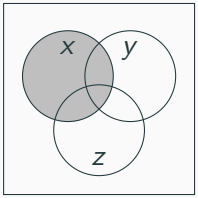




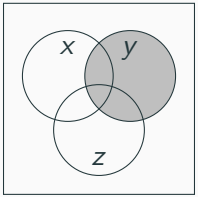
Konstrukcja diagramu Venna – przykład

Funkcja $f(x, y, z) = (x \wedge y) \vee (x \wedge z)$:

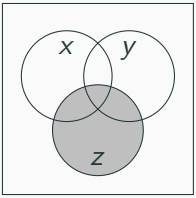
x



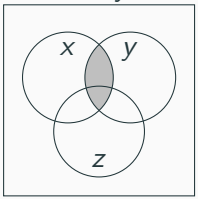
y



z

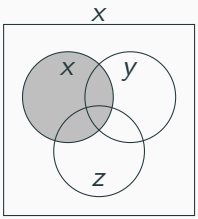


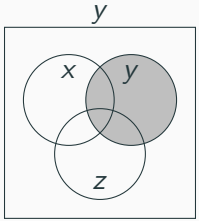
$x \wedge y$

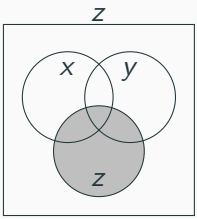


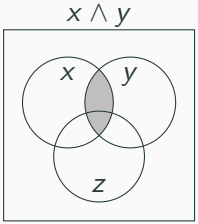
Konstrukcja diagramu Venna – przykład

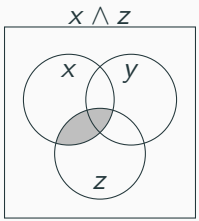
Funkcja $f(x, y, z) = (x \wedge y) \vee (x \wedge z)$:





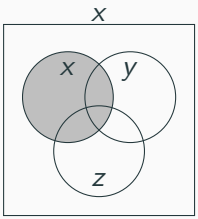


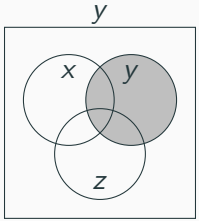


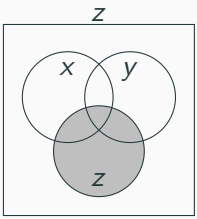


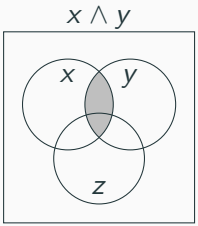
Konstrukcja diagramu Venna – przykład

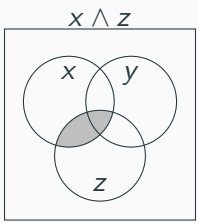
Funkcja $f(x, y, z) = (x \wedge y) \vee (x \wedge z)$:

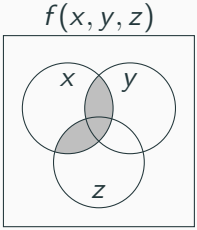






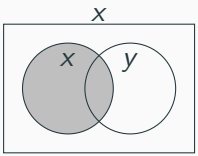


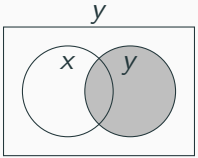




Sprawdzanie równości przy użyciu diagramów Venna

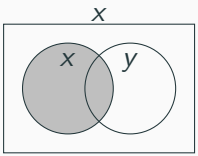
Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:

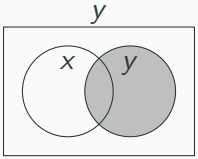


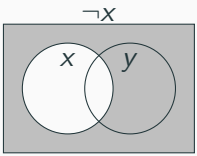


Sprawdzanie równości przy użyciu diagramów Venna

Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:

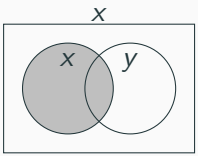


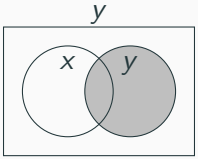


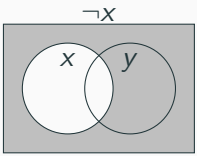


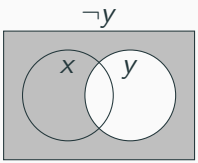
Sprawdzanie równości przy użyciu diagramów Venna

Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:



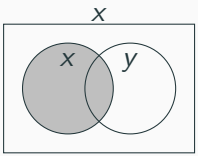


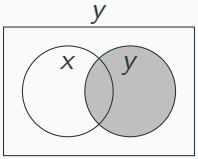


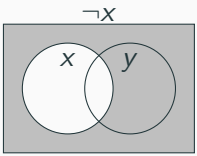


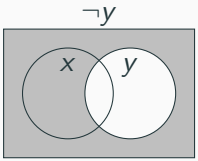
Sprawdzanie równości przy użyciu diagramów Venna

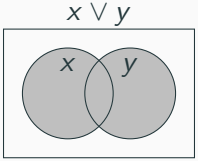
Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:







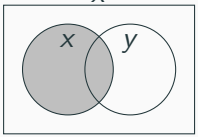




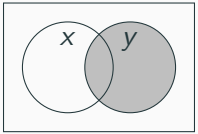
Sprawdzanie równości przy użyciu diagramów Venna

Przykład – funkcje $f(x, y) = \neg x \wedge \neg y$ oraz $g(x, y) = \neg(x \vee y)$:

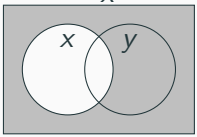
x



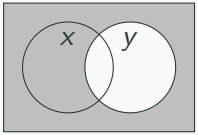
y



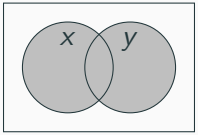
$\neg x$



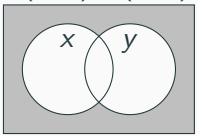
$\neg y$



$x \vee y$



$f(x, y), g(x, y)$



Prawa algebry Boole'a

Prawa algebry Boole'a

- Gdy formuły Φ_1 i Φ_2 są logicznie równoważne, to $\Phi_1 = \Phi_2$ nazywamy prawem algebry Boole'a.
- Jeśli pod dowolną zmienną x w logicznie równoważnych formułach Φ_1 i Φ_2 podstawimy inną formułę Ψ, to powstałe formuły też są logicznie równoważne. Inaczej: jeśli $\Phi_1 = \Phi_2$ jest prawem algebry Boole'a, to $\Phi_1\{x/\Psi\} = \Phi_2\{x/\Psi\}$ też jest prawem algebry Boole'a.
- Wniosek: formuły algebry Boole'a można przekształcać przez przepisywanie przy użyciu praw algebry Boole'a.

Prawa dotyczące stałych

Pozwalają na obliczanie wartości formuł bez zmiennych.

- $0 \vee 0 = 0$
- $1 \vee 0 = 0 \vee 1 = 1$
- $1 \vee 1 = 1$
- $\neg 0 = 1$
- $1 \wedge 1 = 1$
- $0 \wedge 1 = 1 \wedge 0 = 0$
- $0 \wedge 0 = 0$
- $\neg 1 = 0$

Prawa z 1 zmienną

Element neutralny

- $x \vee 0 = 0 \vee x = x$
- $x \wedge 1 = 1 \wedge x = x$

Element anihilujący

- $x \vee 1 = 1 \vee x = 1$
- $x \wedge 0 = 0 \wedge x = 0$

Idempotentność

- $x \vee x = x$
- $x \wedge x = x$

Dopełnienie

- $x \vee \neg x = \neg x \vee x = 1$
- $x \wedge \neg x = \neg x \wedge x = 0$

Podwójna negacja

- $\neg\neg x = x$

Prawa z 2 i 3 zmiennymi

Przemienność

- $x \vee y = y \vee x$
- $x \wedge y = y \wedge x$

Łączność

- $x \vee (y \vee z) = (x \vee y) \vee z$
- $x \wedge (y \wedge z) = (x \wedge y) \wedge z$

Rozdzielność

- $x \vee y \wedge z = (x \vee y) \wedge (x \vee z)$
- $x \wedge (y \vee z) = x \wedge y \vee x \wedge z$

Prawo de Morgana

- $\neg(x \vee y) = \neg x \wedge \neg y$
- $\neg(x \wedge y) = \neg x \vee \neg y$

Zasada dualności

Jeśli $\Phi_1 = \Phi_2$ jest prawem algebry Boole'a, to po zamianie wszystkich wystąpień w Φ_1 i Φ_2:

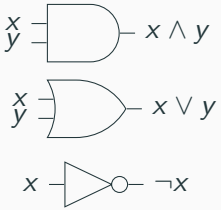
- 0 na 1, i odwrotnie,
- $\vee$ na $\wedge$, i odwrotnie,

otrzymujemy prawo algebry Boole'a.

Układy kombinacyjne

Bramka logiczna

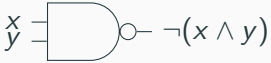
- Rysunkowe przedstawienie spójnika logicznego
- Bramka posiada:
  wejścia (z lewej)
  wyjścia (z prawej)



Bramka logiczna

Bramki AND i OR mogą mieć więcej niż dwa wejścia:

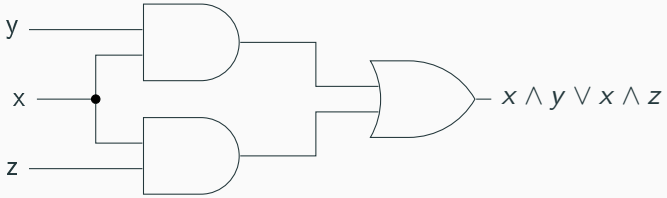
Negację wyjścia bramki można oznaczyć kółkiem („bąbelkiem”):



Układ logiczny

- Diagram, w którym do każdego wejścia bramki jest podłączone co najwyżej jedno wyjście (być może innej bramki).
- Wyrażeniom algebry Boole'a odpowiadają *drzewa* bramek.
  Bramki tworzą drzewo, gdy każde wyjście jest podłączone do co najwyżej jednego wejścia oraz diagram nie zawiera *cykli*.

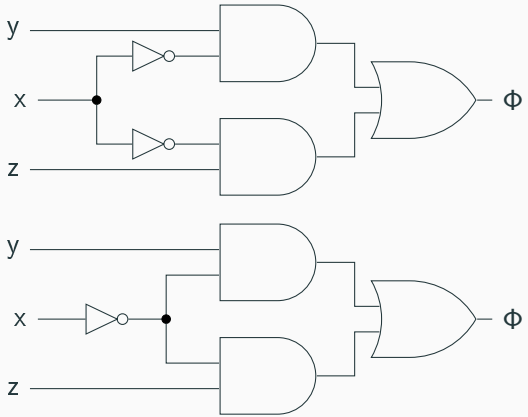
Przykład: $x \wedge y \vee x \wedge z$



Diagramy acykliczne

Można uwspólnić poddiagramy równoważne logicznie.

Przykład: $\Phi = \neg x \wedge y \vee \neg x \wedge z$



Układy kombinacyjne

- Acykliczne układy logiczne nazywamy *układami kombinacyjnymi*.
- Układy kombinacyjne odpowiadają funkcjom logicznym.

Wstęp do SystemVeriloga

Moduły

Podstawowa jednostka programów w Verilogu:

```
module modul(porty);

  // zawartość modułu...

endmodule
```

Porty

Mogą być wejściowe albo wyjściowe:

```
module polsumator(output s, output c, input a, input b);

  // zawartość modułu...

endmodule
```

Porty

Skrócony zapis typów portów:

```
module polsumator(output s, c, input a, b);

  // zawartość modułu...

endmodule
```

Instancje bramek

Należy podać wejścia i wyjścia bramki:

```
module polsumator(output s, c, input a, b);

  xor (s, a, b); // pierwszy port to wyjście
  and (c, a, b);

endmodule
```

Druty

Dodatkowe połączenia wewnątrz modułu:

```
module implikacja(output o, input a, b);

  logic na;

  not (na, a);
  or  (o, na, b);

endmodule
```

Tylko jedno wyjście na drucie

Drut musi być zapisywany przez tylko jedno wyjście!

```
module niedobry(output o, input a, b);

  or  (o, a, b);
  and (o, a, b); // nie zadziała

endmodule
```

Instancje modułów

Podobnie, jak instancje bramek:

```
module sumator(output s, c, input a, b, cin);

  logic ps1s, ps1c, ps2c;

  polsumator ps1(ps1s, ps1c, a, b);
  polsumator ps2(s, ps2c, ps1s, cin);
  or (c, ps1c, ps2c);

endmodule
```

Bloki `assign`

Przypisanie do drutu wartości wyrażenia logicznego:

```
module identycznosc(output o, input a);

  assign o = a;

endmodule
```

Operatory – bramki

- `&&` – koniunkcja, bramka AND
- `||` – dysjunkcja, bramka OR
- `!` – negacja, bramka NOT

```
module implikacja(output o, input a, b);

  assign o = !a || b;

endmodule
```

Stałe bitowe

Logika trójwartościowa – dodatkowa wartość `x` (don't know/don't care):

```
module stalebitowe(output t, f, x);

  assign t = 'b1;
  assign f = 'b0;
  assign x = 'bx;

endmodule
```

Bramki logiczne – logika trójwartościowa

bufor

i	o
0	0
x	x
1	1

not

i	o
0	1
x	x
1	0

and

i_1	i_2	o
0	0	0
0	x	0
0	1	0
x	0	0
x	x	x
x	1	x
1	0	0
1	x	x
1	1	1

or

i_1	i_2	o
0	0	0
0	x	x
0	1	1
x	0	x
x	x	x
x	1	1
1	0	1
1	x	1
1	1	1

xor

i_1	i_2	o
0	0	0
0	x	x
0	1	1
x	0	x
x	x	x
x	1	x
1	0	1
1	x	x
1	1	0

Bramki logiczne – logika trójwartościowa

and

	0	x	1
0	0	0	0
x	0	x	x
1	0	x	1

or

	0	x	1
0	0	x	1
x	x	x	1
1	1	1	1

xor

	0	x	1
0	0	x	1
x	x	x	x
1	1	x	0

Logika trójwartościowa – dla zainteresowanych

Inne spojrzenie – izomorfizm:

- $\{0, x, 1\} \cong \{-1, 0, 1\}$
- $\vee$ odpowiada $\max$ (maksimum)
- $\wedge$ odpowiada $\min$ (minimum)
- $\neg$ odpowiada $-$ (negacji)