Programowanie układów FPGA

Wykład 1: wstęp, logika cyfrowa

Marek Materzok

8 września 2019

FPGA

Scalone układy cyfrowe w użyciu

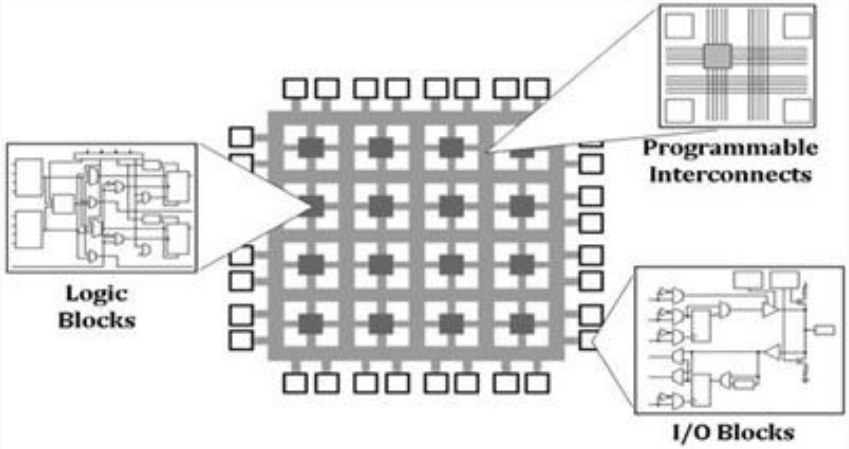
- Procesory – CPU (np. Intel, AMD)
- Procesory graficzne – GPU
- Mikrokontrolery – MCU (np. PIC, AVR, STM32)
- Procesory DSP (np. Analog Devices SHARC)
- System na układzie – SoC (np. Samsung Exynos)
- Układy cyfrowo-analogowe (np. przetworniki ADC/DAC, radio)
- Układy o niskiej integracji (bramki, przerzutniki, pamięci)
- Układy programowalne – PAL, PLA, CPLD, FPGA

Dlaczego układy programowalne?

- Wysoki koszt produkcji układów scalonych
- Możliwość przeprogramowania w zmontowanym urządzeniu
- Możliwość szybkiego prototypowania

Czym jest FPGA?

- Układ logiczny programowalny „w locie”
- Siatka bloków logicznych
- Blok logiczny – kilka bramek plus pamięć



Zalety

- 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

Zastosowania

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

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

Języki opisu sprzętu (HDL)

Języki opisu sprzętu (HDL)

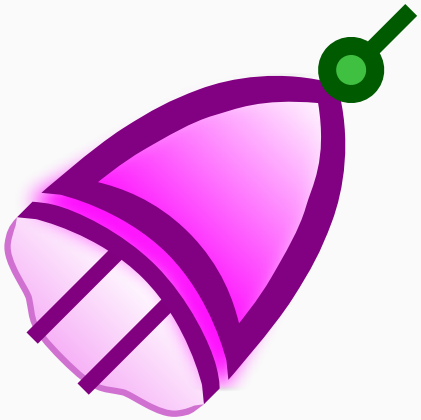
- Języki o semantyce odpowiedniej dla układów logicznych (jawny przepływ danych, obliczenia równoległe, moduły)
- Zastosowania: projektowanie, modelowanie, testowanie i weryfikacja sprzętu
- Popularne: Verilog, SystemVerilog, VHDL
- Inne: Bluespec, Lava, HardCaml, Clash, Chisel, SpinalHDL

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 1800-2012
  - IEEE 1800-2017

Verilog – proponowane narzędzia

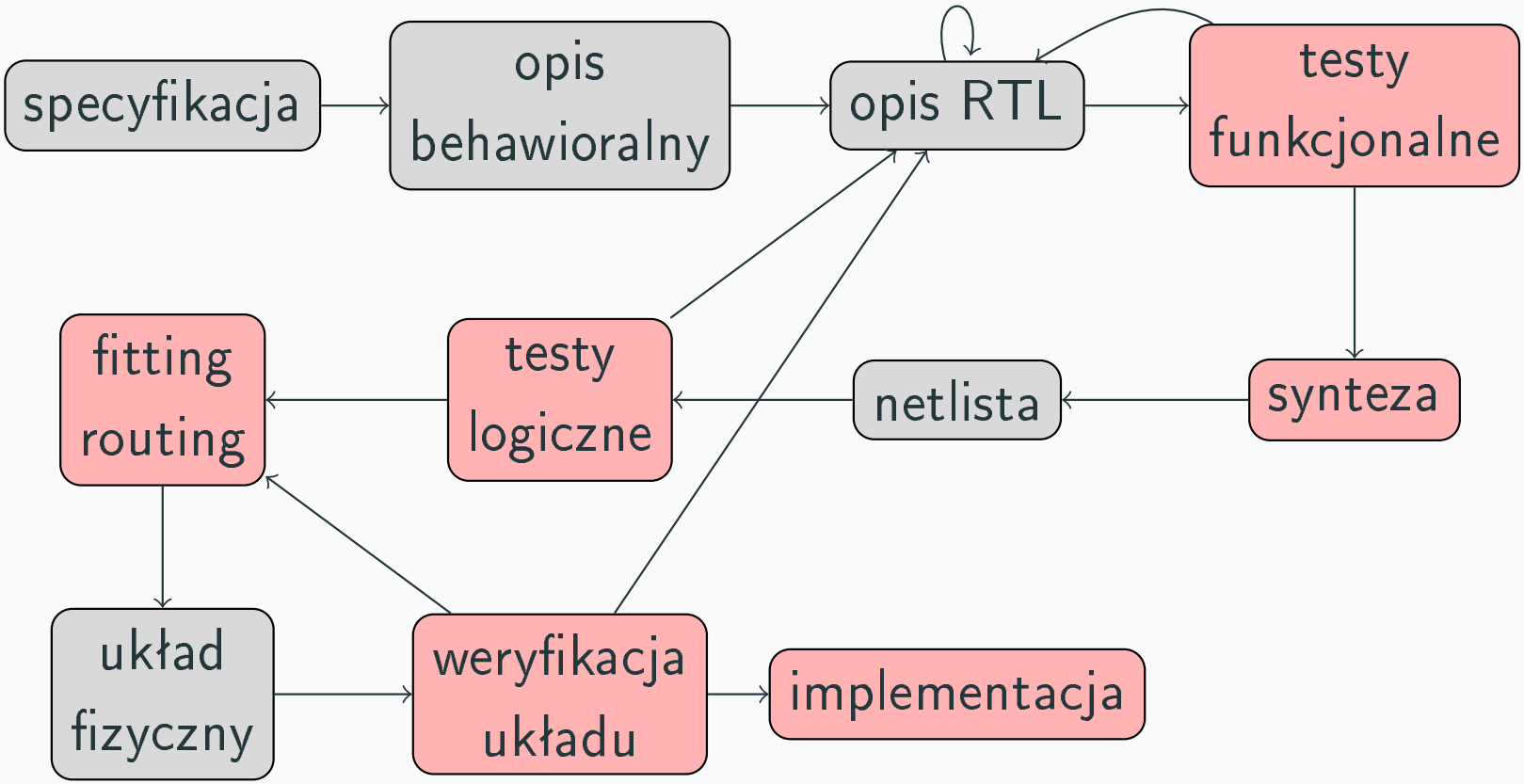
- ModelSim – symulacja, komercyjny
  do zastosowań niekomercyjnych i naukowych
- Quartus – synteza do FPGA, komercyjny
  IDE + narzędzia, wersja niekomercyjna
- Icarus Verilog – symulacja i synteza, GPL2
  obsługuje V2001 i elementy SV2012
- Verilator – symulacja, GPL3
  C++/SystemC z syntezowalnego V2001 i elementów SV2017
- Yosys – synteza i formalna weryfikacja, ISC
  obsługuje syntezowalny V2005 i elementy SV2012
- DigitalJS – interaktywny symulator po syntezie
  wykorzystuje Yosysa



DigitaJS

- `http://digitaljs.tilk.eu/`
- Kod: `https://github.com/tilk/digitaljs_online`
- Poszukiwani programiści, możliwe prace lic/inż

Typowy workflow



Poziomy abstrakcji

- Poziom przełączników

  indywidualne tranzystory, siły sygnału, opóźnienia
- Poziom bramek

  połączenia bramek
- Poziom przepływu danych

  wyrażenia logiczne, transfer między rejestrami (RTL)
- Poziom behawioralny (nie używany do syntezy)

  reakcja na zdarzenia, złożony stan wewnętrzny

Program wykładu

Program wykładu

- Logika cyfrowa (przypomnienie)
- Verilog jako język modelowania sprzętu
- Symulacja i testowanie modeli sprzętu
- Synteza sprzętu
- Budowa układów FPGA (na przykładzie Cyclone V)
- Praktyka programowania FPGA (Altera/Intel)
- Wykorzystywanie tzw. rdzeni IP (modułów sprzętowych)
- Język TCL
- Łączenie FPGA z sprzętowym procesorem
- Magistrale: Avalon, AMBA AXI, podstawy PCI Express (?)
- Języki opisu sprzętu wysokiego poziomu: Chisel i Clash
- Elementy formalnej weryfikacja sprzętu

Logika cyfrowa – literatura

Oczekiwana jest znajomość materiału prezentowanego na wykładzie „Logika cyfrowa”!

Literatura:

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

Literatura

- S. Palnitkar, Verilog HDL – A Guide to Digital Design and Synthesis
- J. Bhasker, A Verilog HDL Primer
- D. Thomas, Logic Design and Verification using SystemVerilog
- Dokumentacja układów serii Cyclone V

Zasady zaliczania

Składowe pracowni:

- Zadania praktyczne
- Projekt

Wykład kończy się egzaminem.

Pomysły na projekt

- Implementacja prostego mikroprocesora
  - na bazie istniejącej architektury (np. RISC-V, 6309, 68000)
  - pomysł własny (ambitne)
- Implementacja algorytmu równoległego
  - Przetwarzanie sygnałów (np. audio)
  - Przetwarzanie obrazów
  - Fraktale
  - Inteligencja obliczeniowa

Różne pomysły:

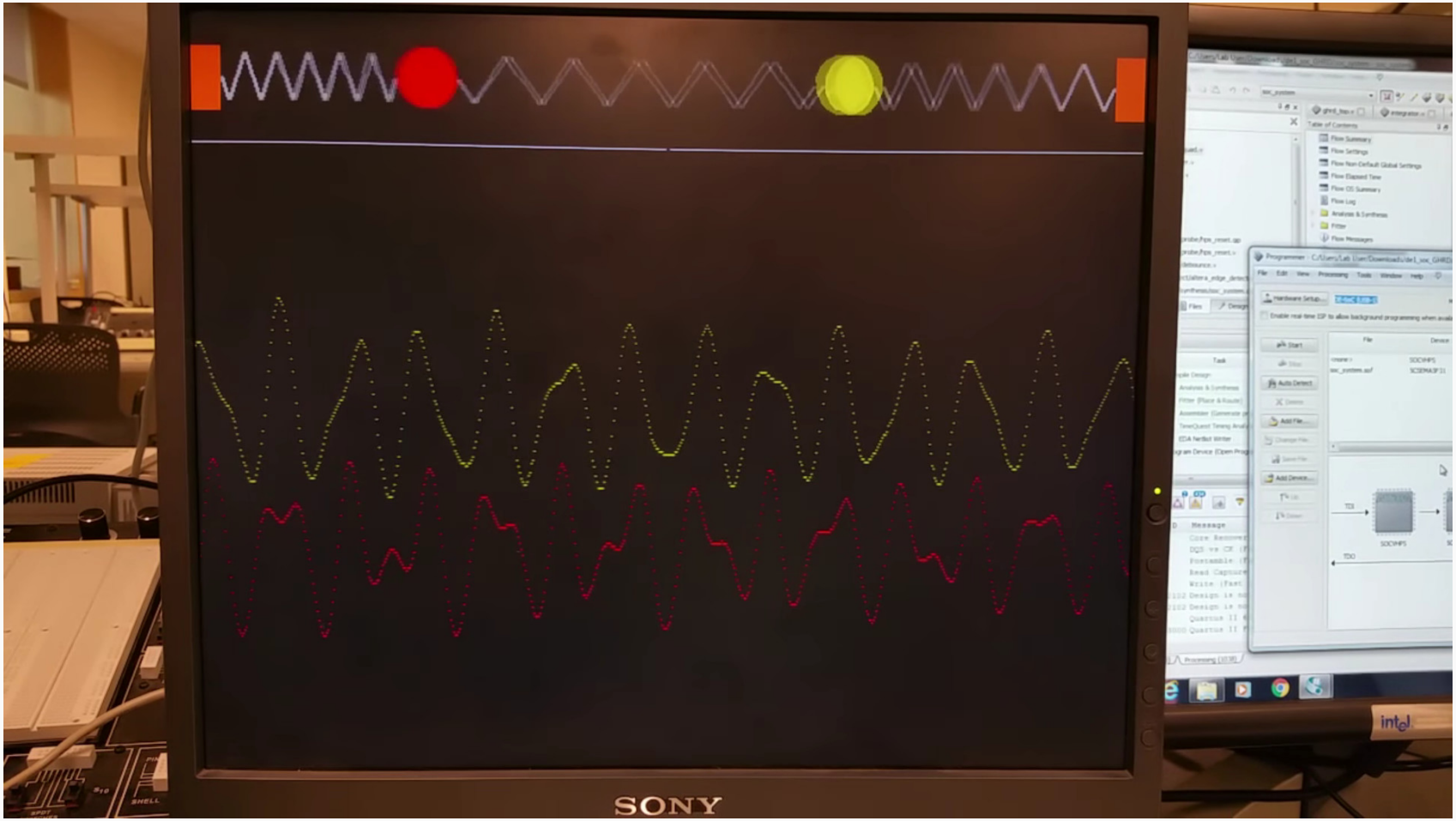
`https://www.youtube.com/playlist?list=PL2E0D05BEC0140F13`

`https://hackaday.io/projects?tag=FPGA`

Real-time cartoonifier

https://www.youtube.com/watch?v=_H2a1xnNvms&index=62&list=PL2E0D05BEC0140F13

Rozwiązywanie równań różniczkowych



https://www.youtube.com/watch?v=cq5JPZjgvKs&index=19&list=PL2E0D05BEC0140F13

Śledzenie twarzy

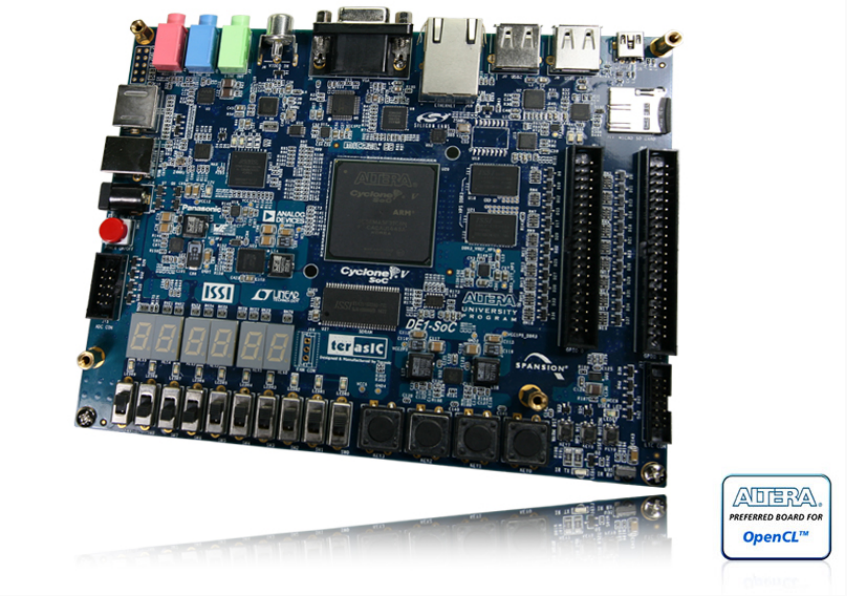
https://www.youtube.com/watch?v=Mys-iswAB88&index=58&list=PL2E0D05BEC0140F13

General game playing

- Projekt Cezarego Siwka z poprzedniej edycji przedmiotu
- Język GDL $\rightarrow$ „propnet” (układ bramkowy) $\rightarrow$ Verilog
- Przeszukiwanie drzewa gry metodą monte carlo, rozwiązanie CPU+FPGA
- Praca: „Implementing Propositional Networks on FPGA” `https://jakubkowalski.tech/Publications/Siwek2018ImplementingPropositional.pdf`

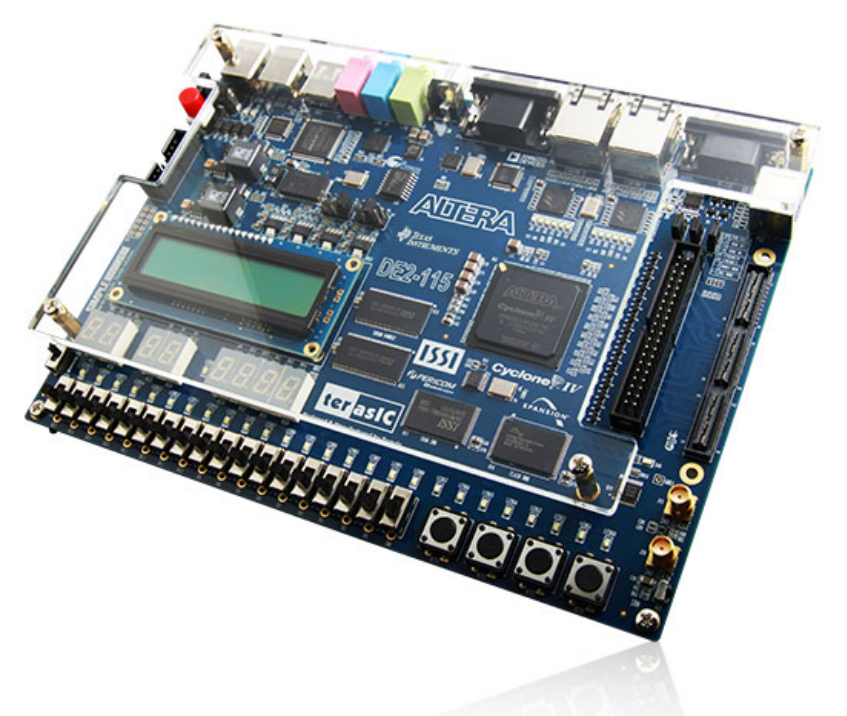
Sprzęt dostępny na zajęciach – DE1-SoC

- Intel Cyclone V
- SoC w architekturze ARMv7 (32-bit)
- dual-core 800 MHz
- FPGA z 32070 ALM (85k LE)
- zegar FPGA 50 MHz
- 5570 Kb RAM (wbudowane), 64 MB SDRAM (zewnętrzne), 1 GB SDRAM (dla SoC)
- 15 sztuk



Sprzęt dostępny na zajęciach – DE2-115

- Intel Cyclone IV
- FPGA z 115k LE
- zegar FPGA 50 MHz
- 3888 Kb RAM (wbudowane), 128 MB SDRAM (zewnętrzne), 2 MB SRAM (zewnętrzne)
- 3 sztuki



Dla projektów z potencjałem – DE10-Pro

- Intel Stratix 10 (GXHU1E1)
- FPGA z 933120 ALM (2800k LE)
- zegar FPGA 250 MHz
- 229 Mb RAM, 4 GB SDRAM (zewnętrzne)
- 1 sztuka

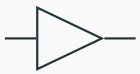
Sprzęt – porównanie

Model	ile	LE	zegar	wb. RAM	zew. DRAM
DE1-SoC	15	85k	50 MHz	5570 Kb	64 MB, 1 GB
DE2-115	3	115k	50 MHz	3888 Kb	128 MB
DE10-Pro	1	2800k	250 MHz	229 Mb	4 GB

Logika cyfrowa – przypomnienie

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

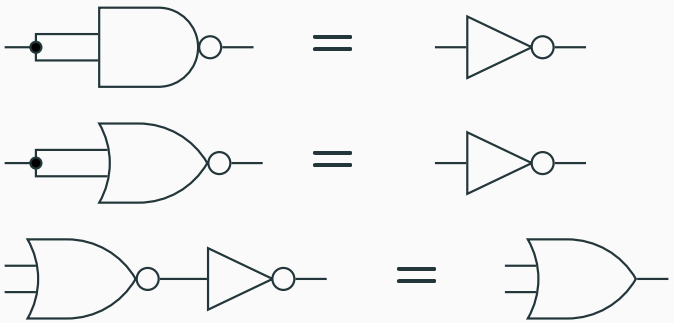
i_1	i_2	o
0	0	0
0	1	1
1	0	1
1	1	1

xor

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

Bramki uniwersalne

- Z NAND lub NOR można zbudować dowolną bramkę
  - przez prawa de Morgana i eliminację podwójnej negacji
- Prosta budowa półprzewodnikowa



nand

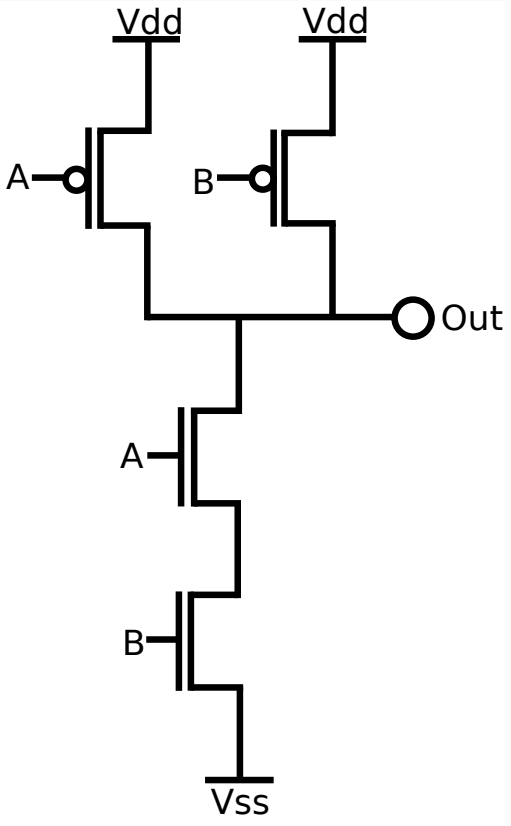
i_1	i_2	o
0	0	1
0	1	1
1	0	1
1	1	0

nor

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

Technologia CMOS

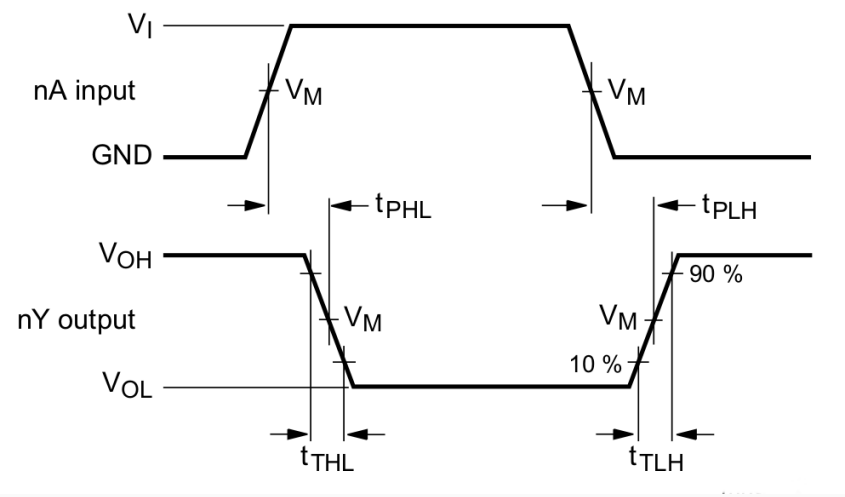
- Complementary MOS
- Tranzystory N-ch łączą z masą, P-ch z zasilaniem
- Pobiera duży prąd tylko podczas przełączania
- Symetryczne poziomy napięć i prądy wyjść
- Najpowszechniejsza współczesna technologia
- Po prawej: bramka NAND



Fizyczne cechy bramek

Zależne od technologii wykonania:

- Czas propagacji
- Impedancja wejściowa i wyjściowa
  - pojemność we/wy, prąd wejść, wydajność prądowa wyjścia
- Fan-in, fan-out
- Poziomy napięć



Cechy statyczne – przykład

Symbol	Parameter	Conditions	25 °C			−40 °C to +85 °C		−40 °C to +125 °C		Unit
			Min	Typ	Max	Min	Max	Min	Max	
74HC04										
V_{IH}	HIGH-level input voltage	V_{CC} = 2.0 V	1.5	1.2	-	1.5	-	1.5	-	V
		V_{CC} = 4.5 V	3.15	2.4	-	3.15	-	3.15	-	V
		V_{CC} = 6.0 V	4.2	3.2	-	4.2	-	4.2	-	V
V_{IL}	LOW-level input voltage	V_{CC} = 2.0 V	-	0.8	0.5	-	0.5	-	0.5	V
		V_{CC} = 4.5 V	-	2.1	1.35	-	1.35	-	1.35	V
		V_{CC} = 6.0 V	-	2.8	1.8	-	1.8	-	1.8	V
V_{OH}	HIGH-level output voltage	$V_I = V_{IH}$ or V_{IL}								
		I_O = −20 μA; V_{CC} = 2.0 V	1.9	2.0	-	1.9	-	1.9	-	V
		I_O = −20 μA; V_{CC} = 4.5 V	4.4	4.5	-	4.4	-	4.4	-	V
		I_O = −20 μA; V_{CC} = 6.0 V	5.9	6.0	-	5.9	-	5.9	-	V
		I_O = −4.0 mA; V_{CC} = 4.5 V	3.98	4.32	-	3.84	-	3.7	-	V
		I_O = −5.2 mA; V_{CC} = 6.0 V	5.48	5.81	-	5.34	-	5.2	-	V
V_{OL}	LOW-level output voltage	$V_I = V_{IH}$ or V_{IL}								
		I_O = 20 μA; V_{CC} = 2.0 V	-	0	0.1	-	0.1	-	0.1	V
		I_O = 20 μA; V_{CC} = 4.5 V	-	0	0.1	-	0.1	-	0.1	V
		I_O = 20 μA; V_{CC} = 6.0 V	-	0	0.1	-	0.1	-	0.1	V
		I_O = 4.0 mA; V_{CC} = 4.5 V	-	0.15	0.26	-	0.33	-	0.4	V
		I_O = 5.2 mA; V_{CC} = 6.0 V	-	0.16	0.26	-	0.33	-	0.4	V
I_I	input leakage current	$V_I = V_{CC}$ or GND; V_{CC} = 6.0 V	-	-	±0.1	-	±1	-	±1	μA
I_{CC}	supply current	$V_I = V_{CC}$ or GND; I_O = 0 A; V_{CC} = 6.0 V	-	-	2	-	20	-	40	μA
C_I	input capacitance		-	3.5	-	-	-	-	-	pF

Logika kombinacyjna

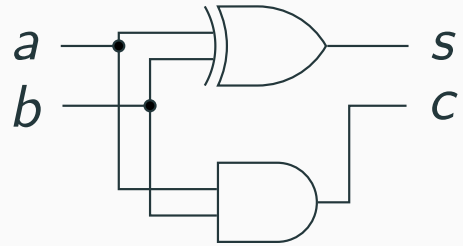
- Acykliczna sieć bramek (dowolny DAG)
- Oblicza funkcje boolowskie $\mathbb{B}^n \to \mathbb{B}^m$
- Asynchroniczna (bez zegara), ma czas propagacji (między zadanym wejściem i wyjściem), ścieżkę krytyczną
- Przykłady: sumator, multiplikator, komparator, koder, dekoder, multiplekser, demultiplekser

Półsumator, sumator

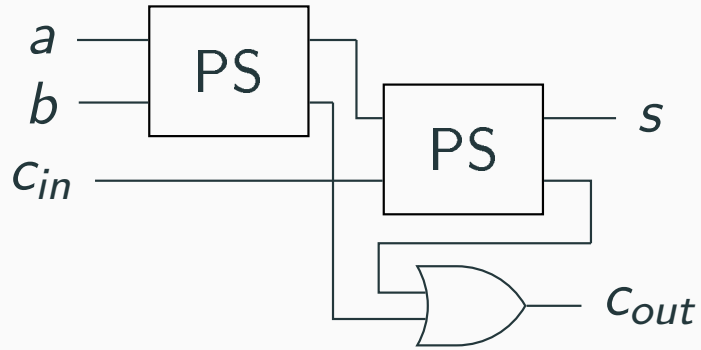
- Liczą sumę cyfr binarnych
- Produkują bit sumy i przeniesienia

a	b	c_{in}	s	c_{out}
0	0	0	0	0
1 jedynka			1	0
2 jedynki			0	1
1	1	1	1	1

Półsumator:

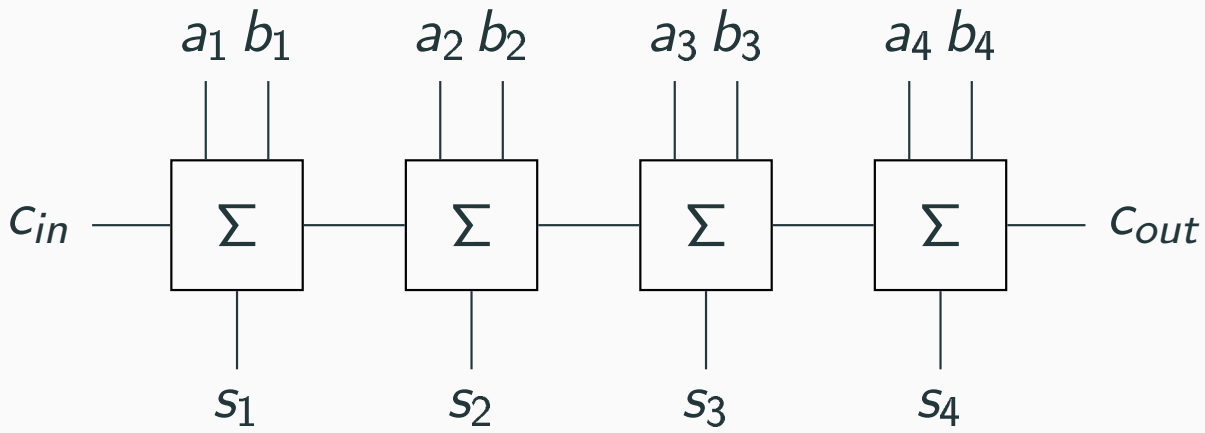


Sumator:



Sumator kaskadowy

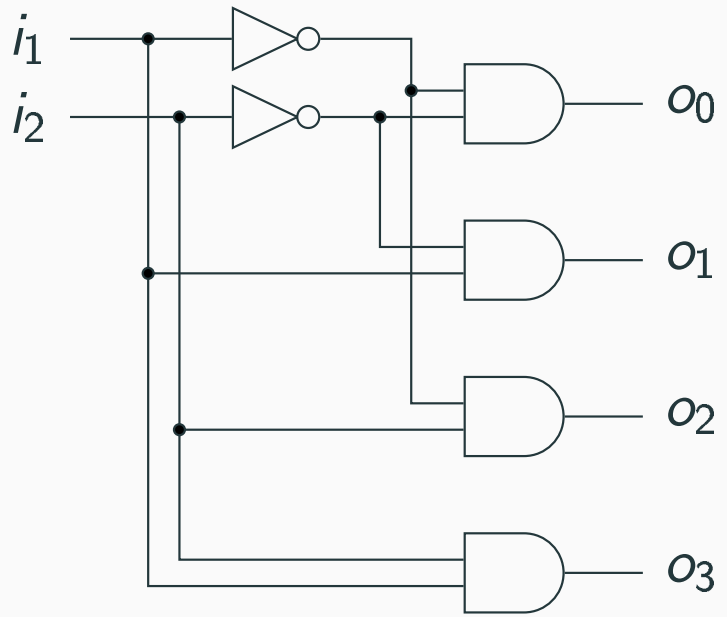
Sumuje dowolną liczbę bitów, z liniowym opóźnieniem:



Dekoder

- Zamienia kod na prostszy sygnał
- Np. przekształca liczbę binarną do postaci one-hot

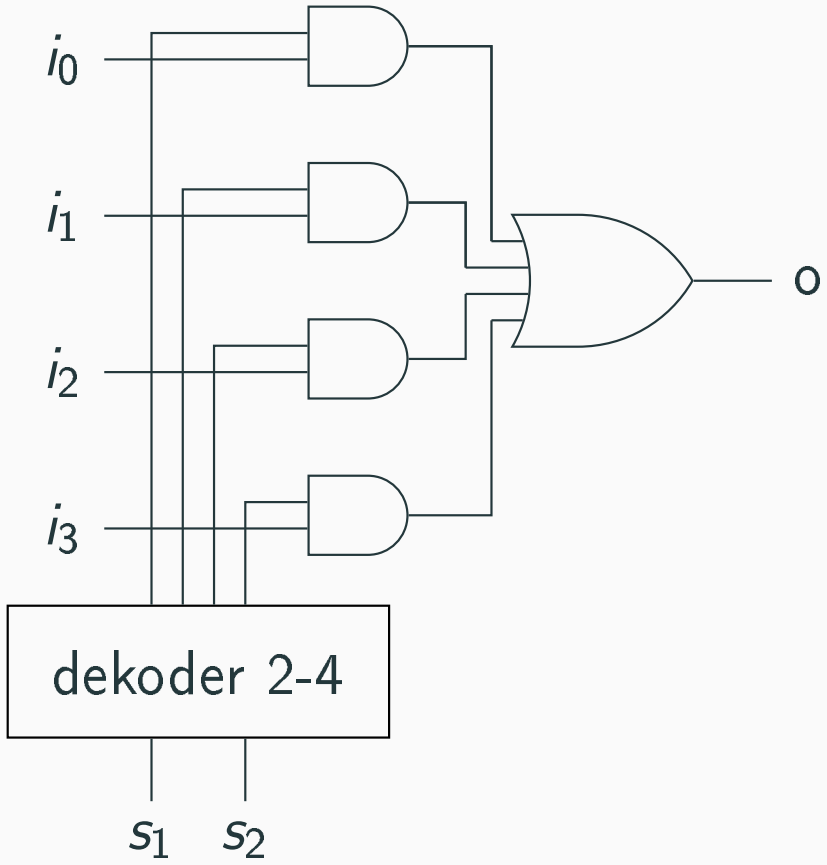
i_1	i_2	o_0	o_1	o_2	o_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Multiplekser

Dostarcza na wyjście jeden z sygnałów wyjściowych

i_0	i_1	i_2	i_3	s_1	s_2	o
x	*	*	*	0	0	x
*	x	*	*	0	1	x
*	*	x	*	1	0	x
*	*	*	x	1	1	x



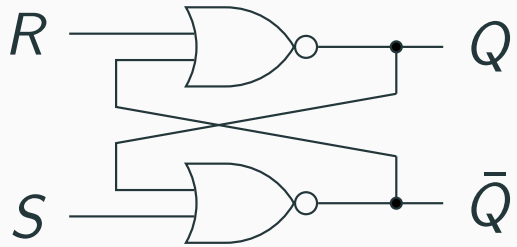
Logika sekwencyjna

- Graf bramek z cyklami
- Dodatnie sprzężenie zwrotne wprowadza stan (pamięć)
- Synchroniczna lub asynchroniczna
- Przykłady: przerzutniki (D, RS, JK), rejestry, liczniki

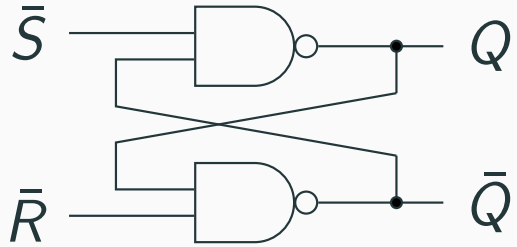
Przerzutnik typu RS (asynchroniczny)

- 1 na R kasuje Q
- 1 na S ustawia Q

R	S	Q	$\bar{Q}$
0	0	Q_0	$\bar{Q}_0$
0	1	1	0
1	0	0	1
1	1	0	0



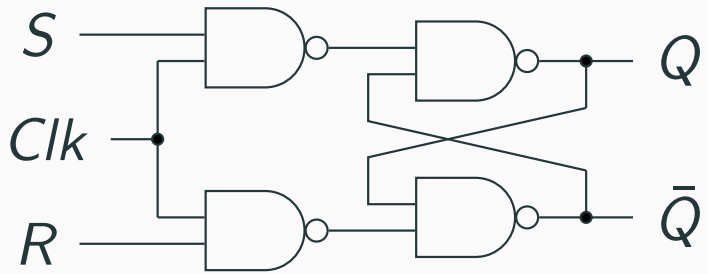
Alternatywnie:



Przerzutnik typu RS (wyzwalany poziomem)

- Kiedy zegar 0, wejście ignorowane
- Kiedy zegar 1, działa jak zwykły RS

Clk	R	S	Q	$\bar{Q}$
0	*	*	Q_0	$\bar{Q}_0$
1	0	0	Q_0	$\bar{Q}_0$
1	0	1	1	0
1	1	0	0	1
1	1	1	0	0

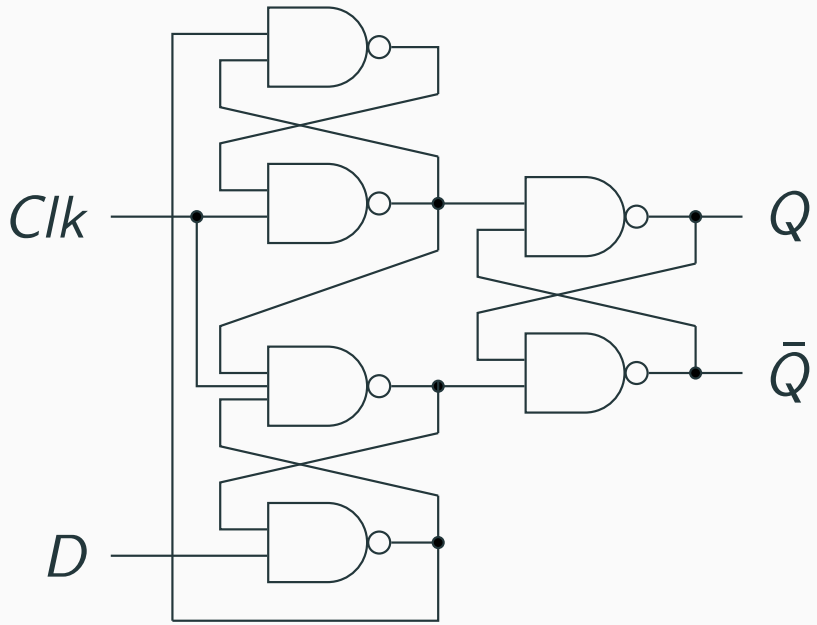


Przerzutnik typu D

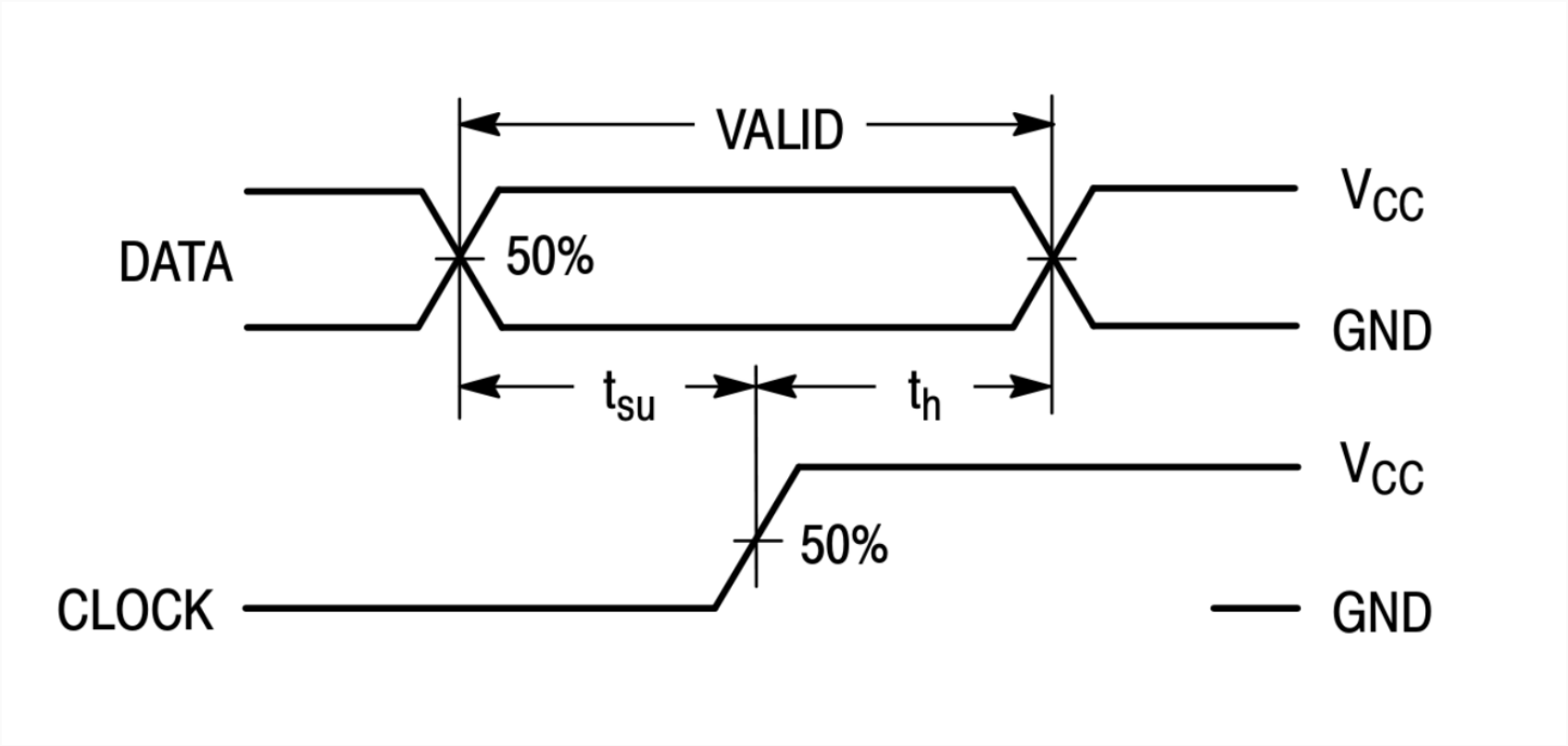
- Zbocze zegarowe przeładowuje stan
- Pojedyncze wejście danych

Clk	D	Q	$\bar{Q}$
$*$	$*$	Q_0	$\bar{Q}_0$
$\uparrow$	0	0	1
$\uparrow$	1	1	0

Jedna z możliwych realizacji:

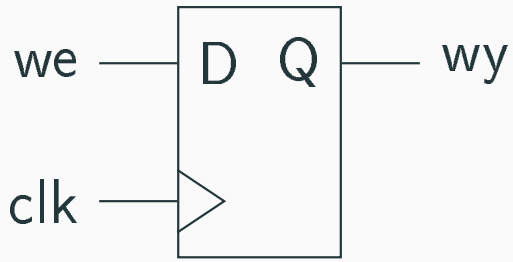


Czas ustalania (setup time), podtrzymania (hold time)



Rejestr

- Pamięć n-bitowa
- Przerzutniki D ze wspólnym zegarem
- Ładowane równolegle
  - Szeregowo: rejestr przesuwny



Automaty skończone

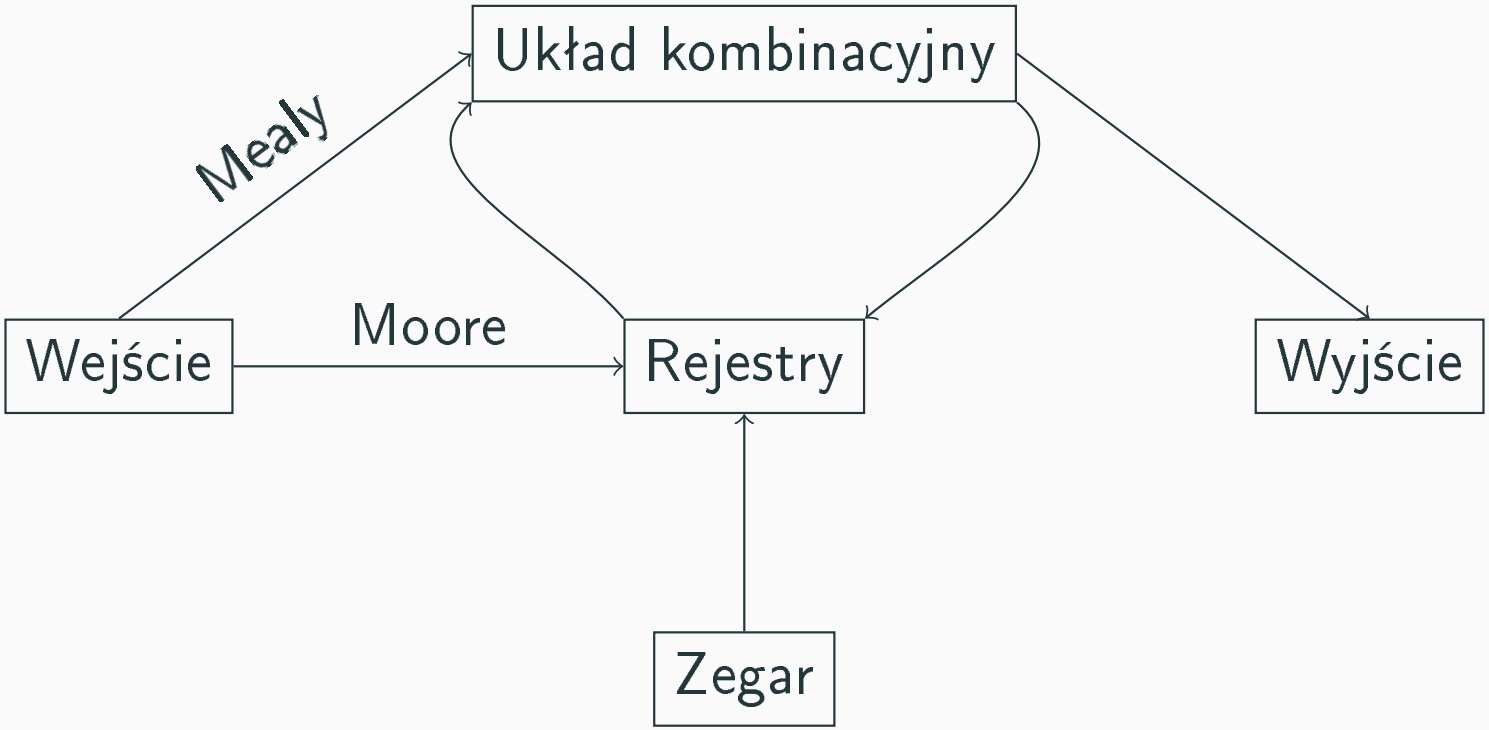
Deterministyczne automaty Moore'a i Mealy'ego:

- krotka $\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle$
- Q – skończony niepusty zbiór *stanów*,
- Σ – skończony niepusty *alfabet* wejściowy,
- Ω – skończony niepusty *alfabet* wyjściowy,
- $\delta : Q \times \Sigma \to Q$ – *funkcja przejścia*,
- $\chi : Q \to \Omega$ (Moore) lub
  $\chi : Q \times \Sigma \to \Omega$ (Mealy) – *funkcja wyjścia*,
- $q_0 \in Q$ – *stan początkowy*.

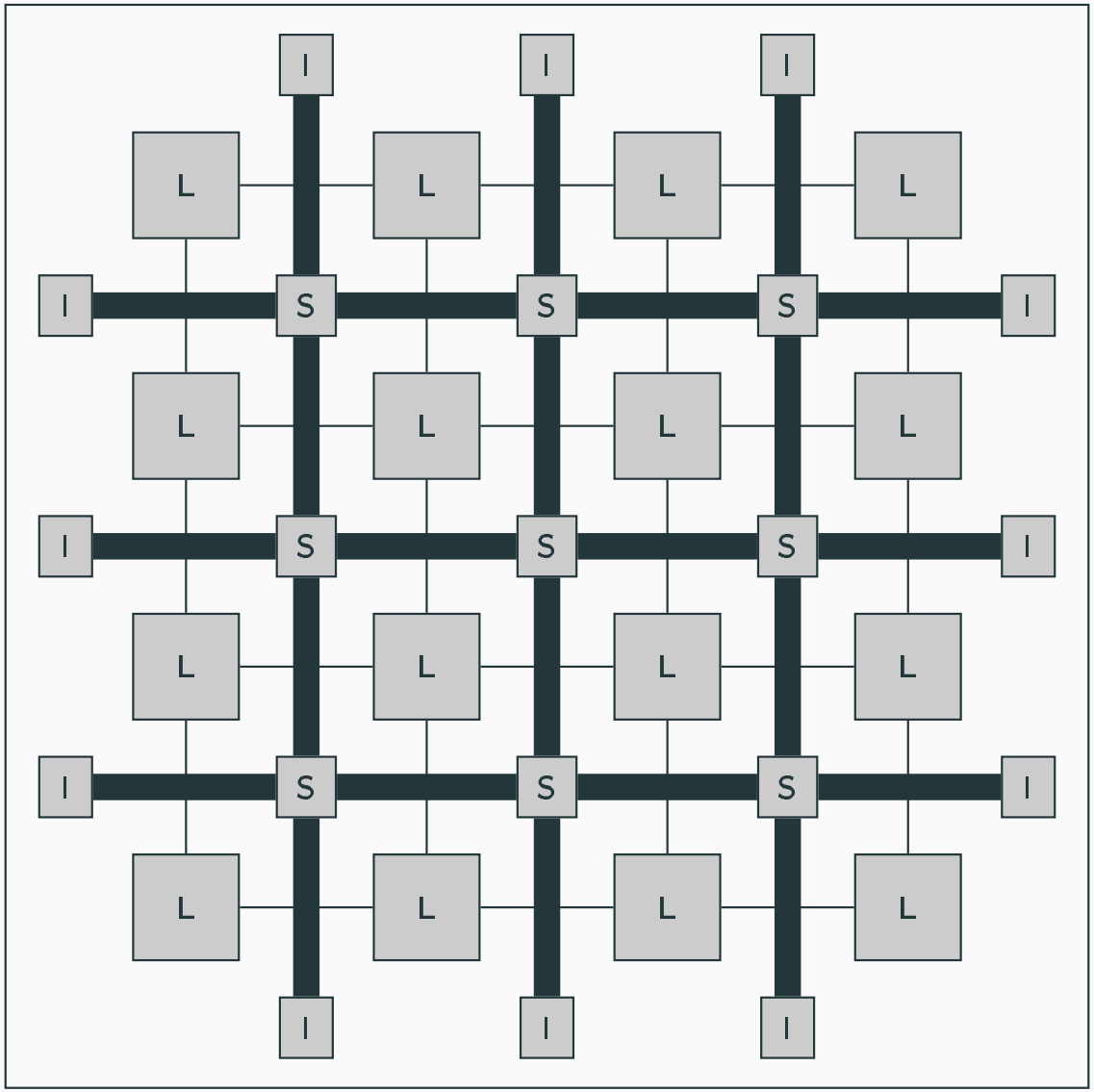
Automaty skończone – fakty

- Zapis: abstrakcyjny, tabele, graf
- Moore i Mealy równoważne (modulo timing)
- Istnieje efektywny algorytm minimalizacji automatu
- Automaty można składać, np. szeregowo i równolegle
- Automaty modelują układy synchroniczne
- Dobór binarnej reprezentacji stanów wpływa na rozmiar i efektywność układu

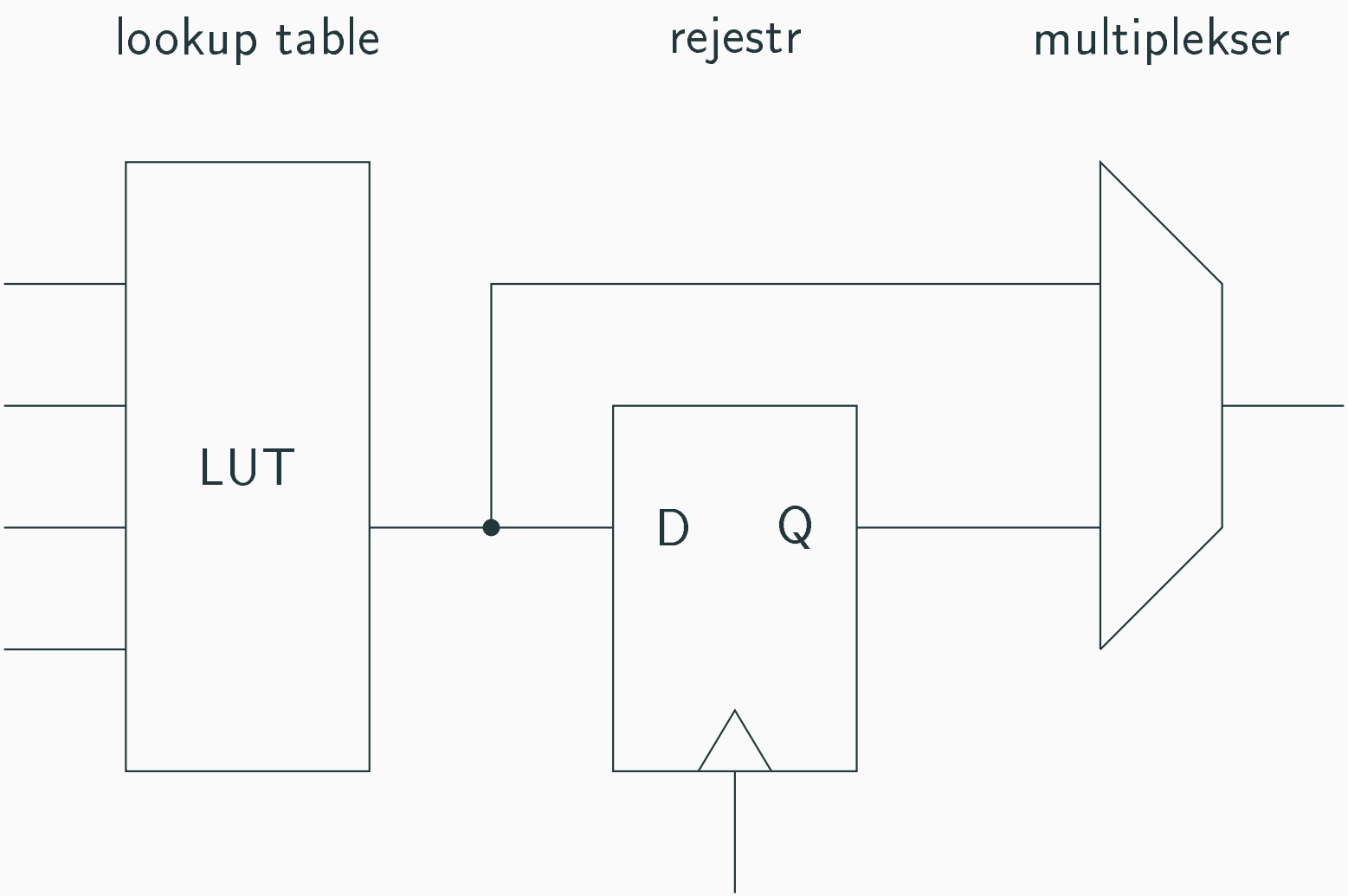
Architektura układów synchronicznych



Budowa układów FPGA



Bloczek logiczny (uproszczony)



Wprowadzenie do Veriloga

Verilog jako język modelowania

- Na wykładzie „Logika cyfrowa” (System)Verilog był przedstawiony jako język do **syntezy** sprzętu.
- Oryginalnym zastosowaniem Veriloga jest **modelowanie** sprzętu, z odrębną semantyką symulacji.
- Pułapki:
  - Nie każdy model sprzętu jest syntezowalny!
  - Synteza może wygenerować sprzęt o innym zachowaniu niż model!

Verilog a SystemVerilog

- SystemVerilog wprowadza rozszerzenia względem Veriloga:
  - Typ `logic`
  - Podział na `always_comb`, `always_latch` i `always_ff`
  - Tablice wielowymiarowe, struktury i unie
  - Inne... (np. klasy, weryfikacja formalna)
- Wsparcie narzędziowe dla SystemVeriloga niepełne (zwłaszcza w open source)
- Verilog powszechnie używany

Moduły

Podstawowa jednostka programów w Verilogu:

```
module modul(parametry);

  // zawartosc modułu...

endmodule
```

Parametry

Mogą być wejściowe albo wyjściowe:

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

  // zawartosc modułu...

endmodule
```

Instancje bramek

Należy podać wejścia i wyjścia bramki:

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

  xor xor1(s, a, b);
  and and1(c, a, b);

endmodule
```

Druty

Dodatkowe połączenia wewnątrz modułu:

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

  wire ps1s, ps1c, ps2c;

  // ciąg dalszy...

endmodule
```

Druty służą wyłącznie do łączenia elementów, **nie mają funkcji pamięciowej**. (Drut to nie zmienna!)

W SystemVerilogu `logic` może pełnić funkcję drutu.

Instancje modułów

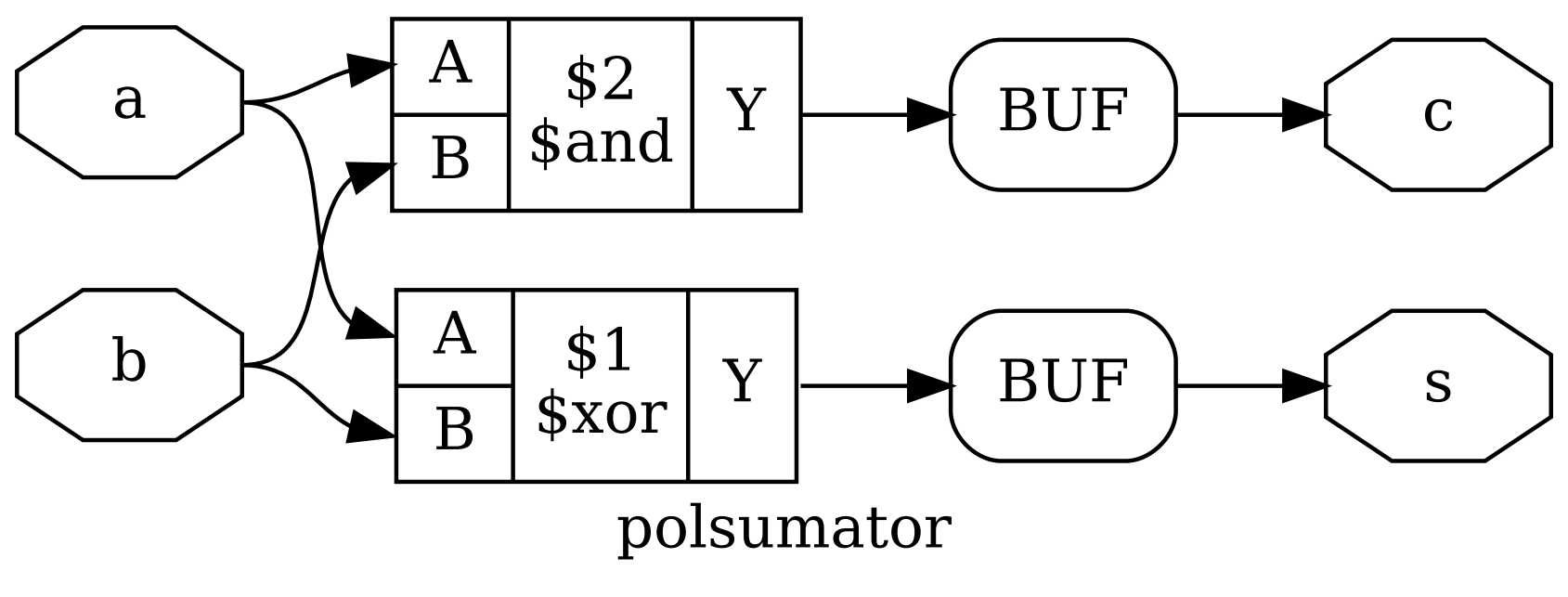
Podobnie, jak instancje bramek:

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

  wire ps1s, ps1c, ps2c;

  polsumator ps1(ps1s, ps1c, a, b);
  polsumator ps2(s, ps2c, ps1s, cin);
  or or1(c, ps1c, ps2c);

endmodule
```

Synteza – Yosys

```
$ yosys
yosys> read_verilog polsumator.v
yosys> show
```

Półsumator – wynik



Elementy proceduralne – sekwencje

Kod proceduralny w Verilogu (w **symulacji**) wykonywany jest sekwencyjnie.

```
begin
  instr1;
  instr2;
  instr3;
  instr4;
end
```

Elementy proceduralne – opóźnienia

Symulacja w Verilogu uwzględnia czas.

Sekwencje normalnie wykonują się „natychmiastowo”.

Wykonanie instrukcji można odroczyć do późniejszej chwili w czasie.

```
begin
  instr1;
  instr2;
  #5 instr3;
  #3 instr4;
end
```

Elementy proceduralne – blok initial

Instrukcje proceduralne muszą być użyte w **blokach**.

Blok `initial` wykonywany jest **raz**, w momencie rozpoczęcia symulacji.

```
initial begin
  instr1;
  instr2;
  #5 instr3;
  #3 instr4;
end
```

Elementy proceduralne – przypisania blokujące

Działają (w **symulacji**) analogicznie do przypisań np. w języku C.

Przypisywana zmienna w Verilogu musi być zdefiniowana jako `reg`.
W SystemVerilogu można użyć `logic`.

Do drutów (`wire`) nie można przypisywać.

```
reg a, b;

initial begin
  a = 0;
  b = 0;
  #5 b = 1;
  #3 a = 1;
end
```

Elementy proceduralne – kompletny moduł

```
module testbench;
  reg a, b;
  wire s, c;

  polsumator ps1(s, c, a, b);

  initial begin
    a = 0;
    b = 0;
    #5 b = 1;
    #3 a = 1;
  end
endmodule
```

Elementy proceduralne – finish

Instrukcja `$finish` kończy symulację.

```
module testbench;
  reg a, b;
  wire s, c;
  polsumator ps1(s, c, a, b);
  initial begin
    a = 0;
    b = 0;
    #5 b = 1;
    #3 a = 1;
    #5 $finish;
  end
endmodule
```

Elementy proceduralne – monitor

Instrukcja $monitor powoduje wypisywanie komunikatu **z końcem** każdej chwili czasu, w której nastąpiły zmiany wypisywanych wartości.

```
module testbench;
  reg a, b;
  wire s, c;
  polsumator ps1(s, c, a, b);
  initial begin
    // jak w poprzednim przykładzie
  end
  initial $monitor($time, " a=%d b=%d s=%d c=%d",
    a, b, s, c);
endmodule
```

Symulacja – iverilog

```
$ iverilog polsumator.v testbench.v -o testbench
$ ./testbench
                    0 a=0 b=0 s=0 c=0
                    5 a=0 b=1 s=1 c=0
                    8 a=1 b=1 s=0 c=1
$
```

Czas propagacji

Dla elementów podstawowych można określić czas propagacji.

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

  xor #(2) xor1(s, a, b);
  and #(2) and1(c, a, b);

endmodule
```

Czas propagacji - symulacja

```
$ iverilog polsumator.v testbench.v -o testbench
$ ./testbench
                   0 a=0 b=0 s=x c=x
                   2 a=0 b=0 s=0 c=0
                   5 a=0 b=1 s=0 c=0
                   7 a=0 b=1 s=1 c=0
                   8 a=1 b=1 s=1 c=0
                  10 a=1 b=1 s=0 c=1
$
```

Logika czterowartościowa

- Cztery wartości: 0, 1, x, z
- z – stan wysokiej impedancji („brak sygnału”)
- x – stan nieznany („sygnał nieznany/niepoprawny”)
- Tabela nadpisywania sygnałów (dla drutów `wire`):

	0	1	x	z
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
z	0	1	x	z

Czas propagacji – formy

```
xor #(2) xor1(s, a, b);
xor #(2,3) xor1(s, a, b);
xor #(2,3,4) xor1(s, a, b);
```

- Kolejne wartości: czas 0-1, czas 1-0, czas ?-z (czas wyłączenia)
- Jedna wartość: wszystkie są równe
- Dwie wartości: czas wyłączenia minimum z podanych dwóch

Czas propagacji – zróżnicowanie

```
xor #(2:3:5) xor1(s, a, b);
xor #(2:3:5, 3:4:5) xor1(s, a, b);
```

- Wartości minimalne, typowe i maksymalne
- Przy symulacji można wybrać, którą wartość chce się symulować

```
$ iverilog -Tmin polsumator.v testbench.v -o testbench_min
$ iverilog -Tmax polsumator.v testbench.v -o testbench_max
```

Czas propagacji – przykład

Symbol	Parameter	Conditions	25 °C			-40 °C to +125 °C		Unit
			Min	Typ	Max	Max (85 °C)	Max (125 °C)	
74HC08								
t_{pd}	propagation delay	nA, nB to nY; see Figure 6 [1]						
		V_{CC} = 2.0 V	-	25	90	115	135	ns
		V_{CC} = 4.5 V	-	9	18	23	27	ns
		V_{CC} = 5.0 V; C_L = 15 pF	-	7	-	-	-	ns
		V_{CC} = 6.0 V	-	7	15	20	23	ns
t_t	transition time	see Figure 6 [2]						
		V_{CC} = 2.0 V	-	19	75	95	110	ns
		V_{CC} = 4.5 V	-	7	15	19	22	ns
		V_{CC} = 6.0 V	-	6	13	16	19	ns

Specyfikowanie elementów podstawowych

```
primitive udp_and(output out, input a, b);
  table
    1 1 : 1;
    0 ? : 0;
    ? 0 : 0;
    1 x : x;
    x 1 : x;
  endtable
endprimitive
```

Znak ? oznacza dowolną wartość. Znak z nie jest dozwolony.
(Braku sygnału nie można testować!)

Elementy podstawowe sekwencyjne

```
primitive rs_flipflop(output reg out, input r, s);
  table
    0 0 : ? : -;
    1 0 : ? : 0;
    0 1 : ? : 1;
    1 1 : ? : x;
  endtable
endprimitive
```

Znak - oznacza brak zmian.

Elementy podstawowe wyzwalane zboczem

```
primitive d_flipflop(output reg out, input clk, d);
  table
    (01) 0    : ? : 0;
    (01) 1    : ? : 1;
    (1?) ?    : ? : -;
    (?0) ?    : ? : -;
    ?    (??) : ? : -;
  endtable
endprimitive
```