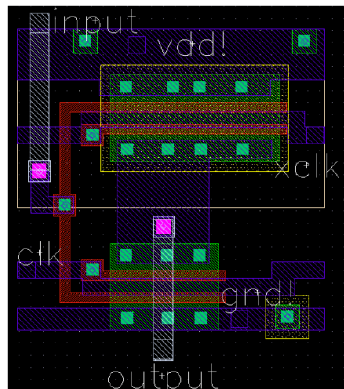

Układy analogowe, cyfrowe, mieszane

-



Analogowy cykl projektowy: cechy

Większość projektów analogowych **jest** silnie uzależniona od technologii.

Parametry technologiczne określają parametry elektryczne elementów (tranzystorów).

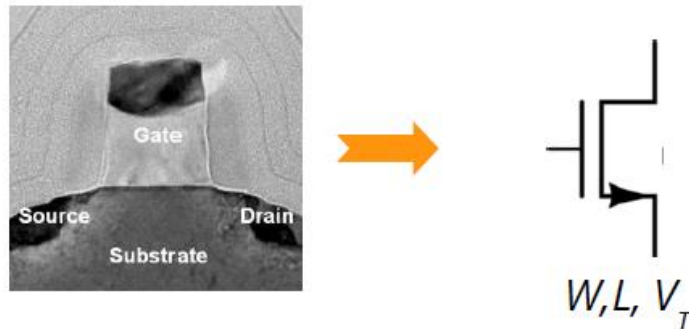
Parametry elektryczne elementów wpływają silnie na parametry wynikowe bloków (np. wzmacnienie wzmacniacza).

Technologia określa możliwe topografie układu. Dopiero weryfikacja układu po wykonaniu topografii daje możliwość znalezienia przewidywanych parametrów układu.

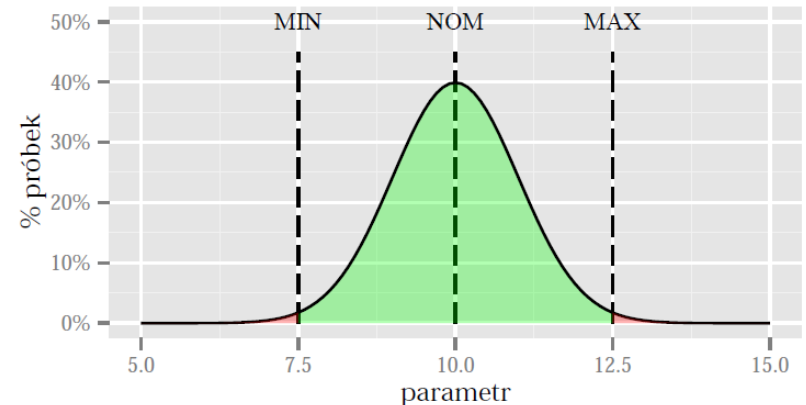
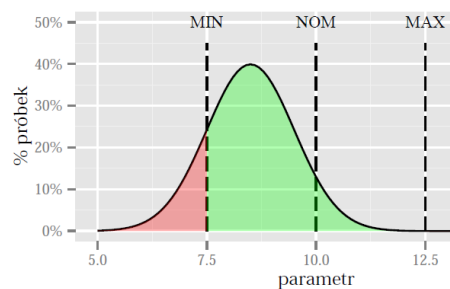
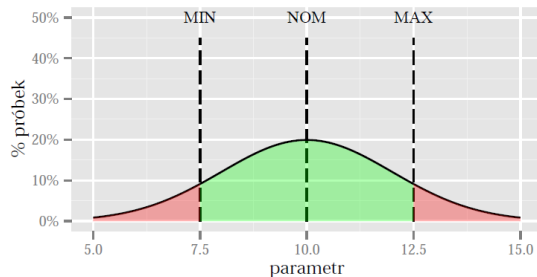
Na parametry mają dodatkowo silny wpływ rozrzuty technologiczne.

Analogowy cykl projektowy: ilustracje

- technologia a parametry elementów



- optymalizacja projektu ze względu na uzysk – dążenie do zmniejszenia wrażliwości na rozrzuty i „wycentrowania” wartości nominalnej

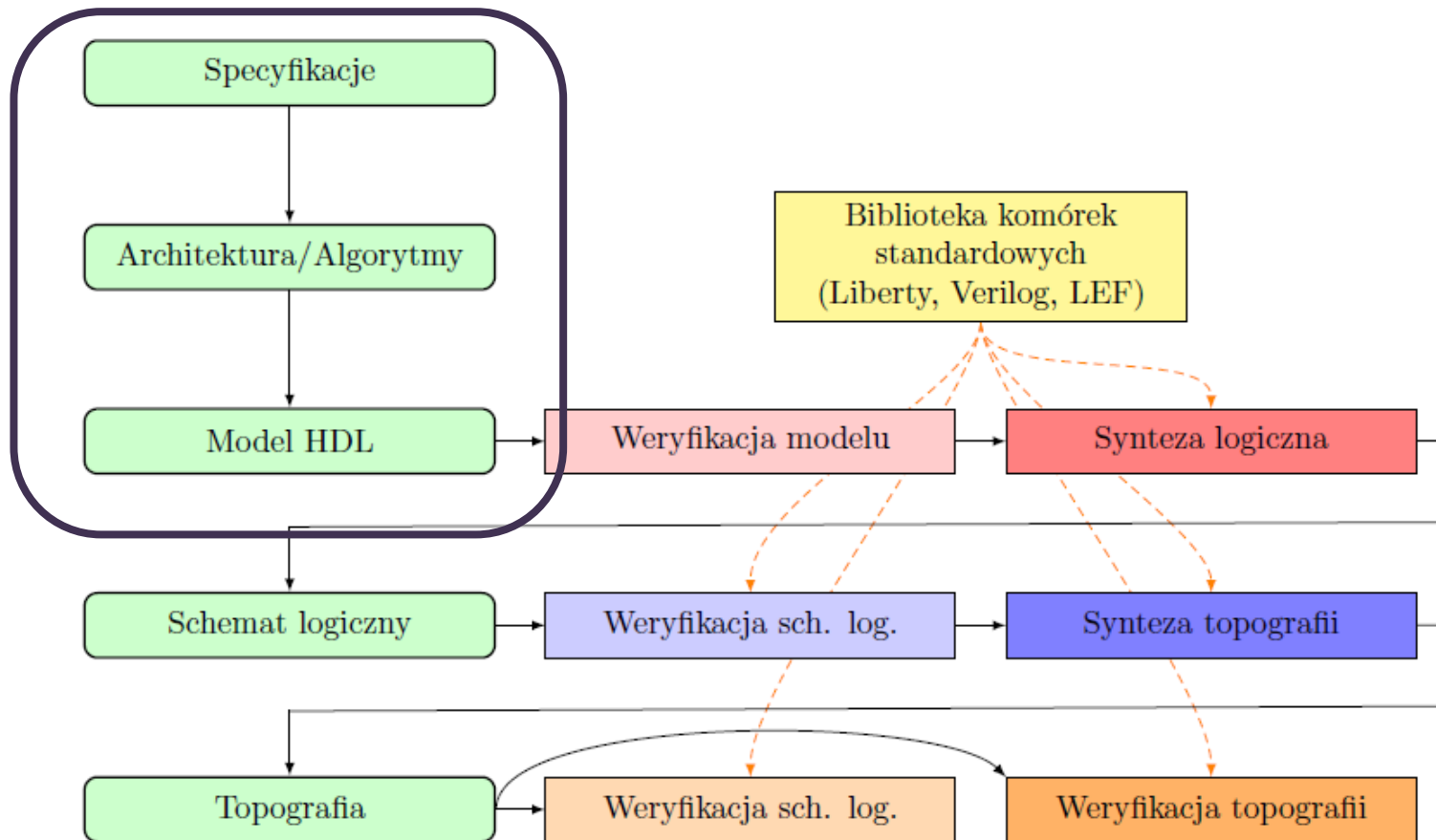


Cykl analogowy: automatyzacja

- Zmniejszenie czasu i kosztów projektu można uzyskać wykorzystując gotowe modele funkcjonalne bloków analogowych np.:
 - makromodele SPICE
 - modele AHDL (Verilog-A, Verilog-AMS, VHDL-AMS).
- Nie zwalnia to od precyzyjnych weryfikacji na kolejnych etapach projektowania, jeśli wymagania co do parametrów układów są duże.

Cyfrowy cykl projektowy

- Tylko część zaznaczona wymaga znaczącego nakładu pracy projektanta.
- Model HDL jest niezależny od technologii.
- Projektant nie tworzy schematu elektrycznego, ani topografii.



Układy mieszane

- Układy zawierające **w jednym chipie** zarówno część cyfrową jak i analogową są obecnie rozpowszechnione (telekomunikacja, motoryzacja, medycyna).
- Projektowanie takich układów wiąże się z kilkoma poważnymi problemami:
 - konieczność zintegrowania różnych cykli projektowych i modeli na różnym poziomie złożoności,
 - różne narzędzia do weryfikacji układów,
 - należy nie tylko odpowiedzieć na pytanie czy obie części działają samodzielnie, ale również czy wzajemnie się nie zakłócają (przesłuchy, sprzężenia zwrotne przez podłoże, szумы).
- Dlatego dla **typowych zastosowań** warto korzystać z rozwiązań predefiniowanych.

Programmable System on Chip, PSoC® rekonfigurowalny układ mieszany

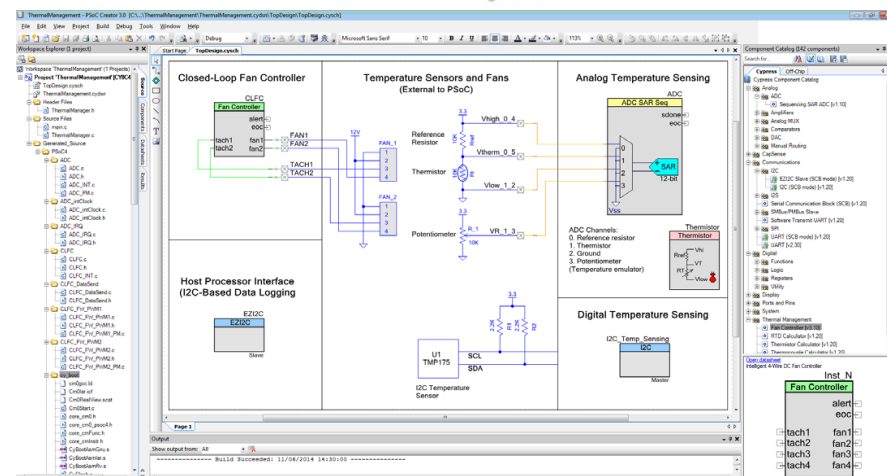
Programmable System on Chip

- PSoC firmy Cypress to udana próba zaradzenia problemom związanym z projektowaniem układów mieszanych.
- Jest to podobnie jak FPGA scalony **system rekonfigurowalny**, zawierający charakterystyczne dla danej wersji zasoby.
- W odróżnieniu od FPGA zasoby nie są ograniczone do bloków cyfrowych.
- Dostępne są również **programowane bloki analogowe**.
- Pozwala to na zastosowanie szybkiej ścieżki projektowej, w której **projektant uwolniony jest od skomplikowanego cyklu projektowania części analogowej i badania wpływu części cyfrowej na tę część.**

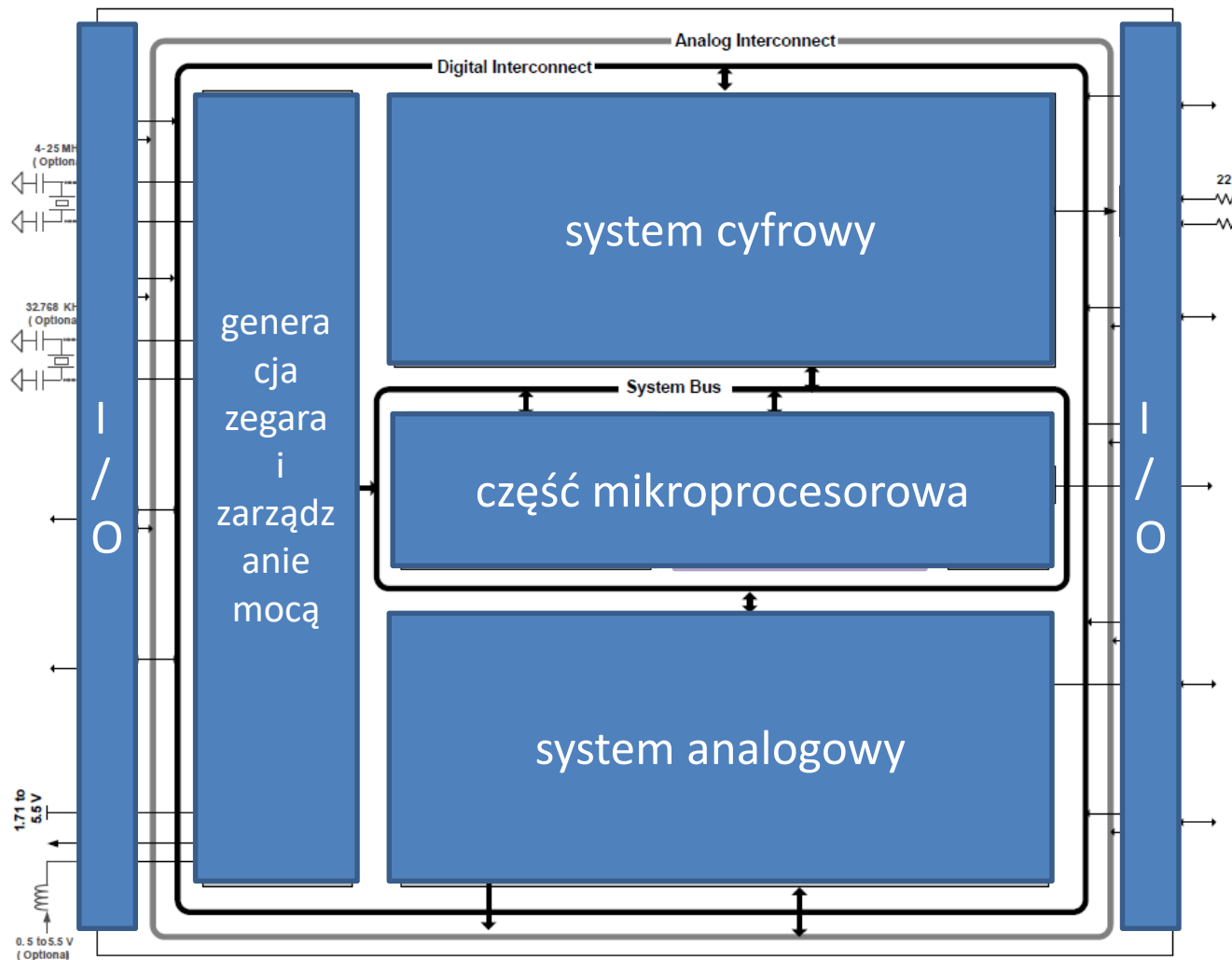
PSoC zawiera rekonfigurowalny system wbudowany z dodatkową częścią analogową i przetwarzaniem AC/DC.

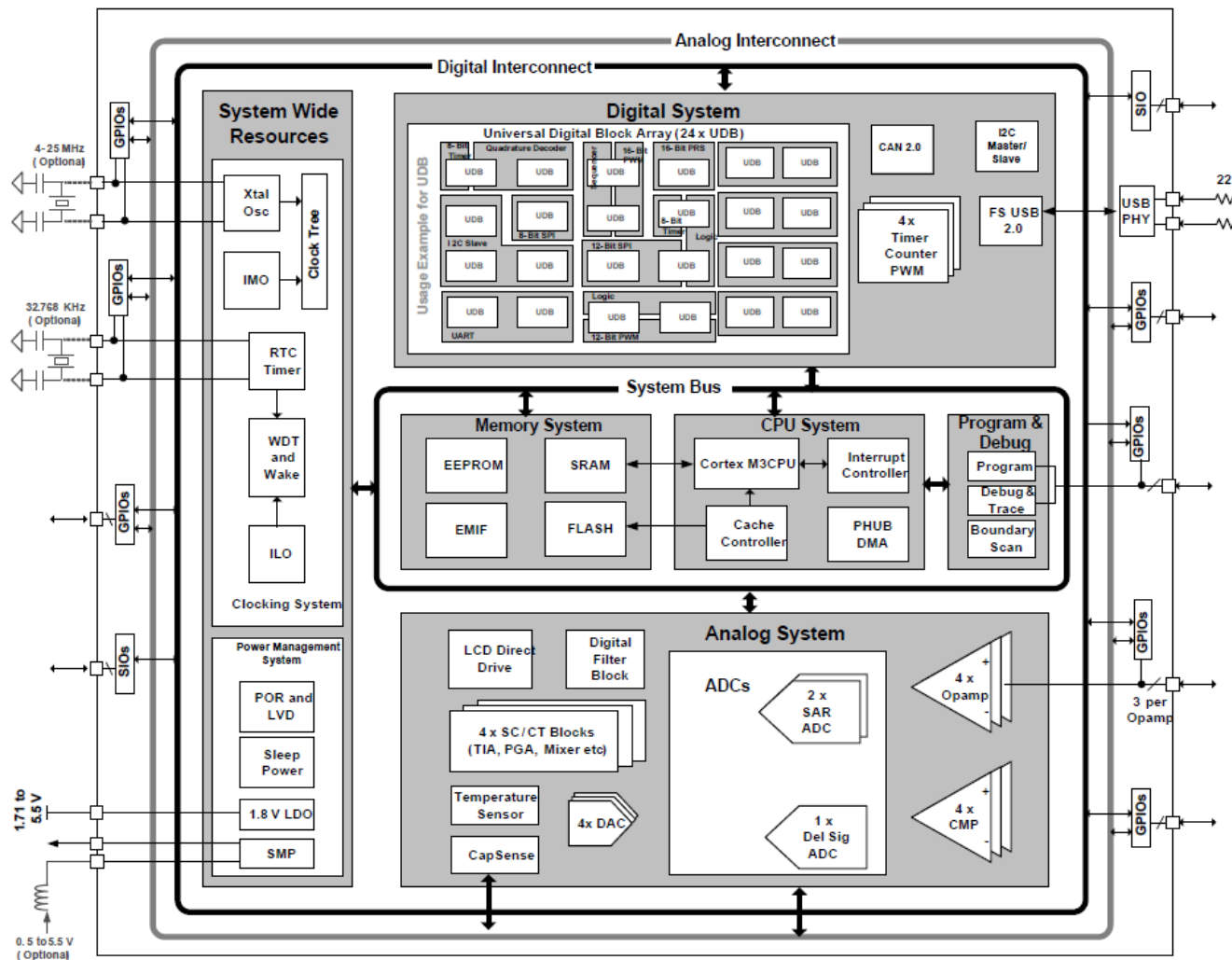
PSoC® - zestawy projektowe

- CY8CKIT-001: platforma uruchomieniowa dla mikrokontrolerów rodziny PSoC 1, PSoC 3, PSoC 4, or PSoC 5LP.
 - CY8CKIT-050 zestaw dla układów analogowych w zastosowaniach low-power i low-voltage na bazie PSoC 5LP.
- MiniProg3: urządzenie umożliwiające programowanie FLASH.
- PSoC Creator: darmowe zintegrowane środowisko uruchomieniowe IDE.



Architektura PSoC 5LP: CY8C58LP

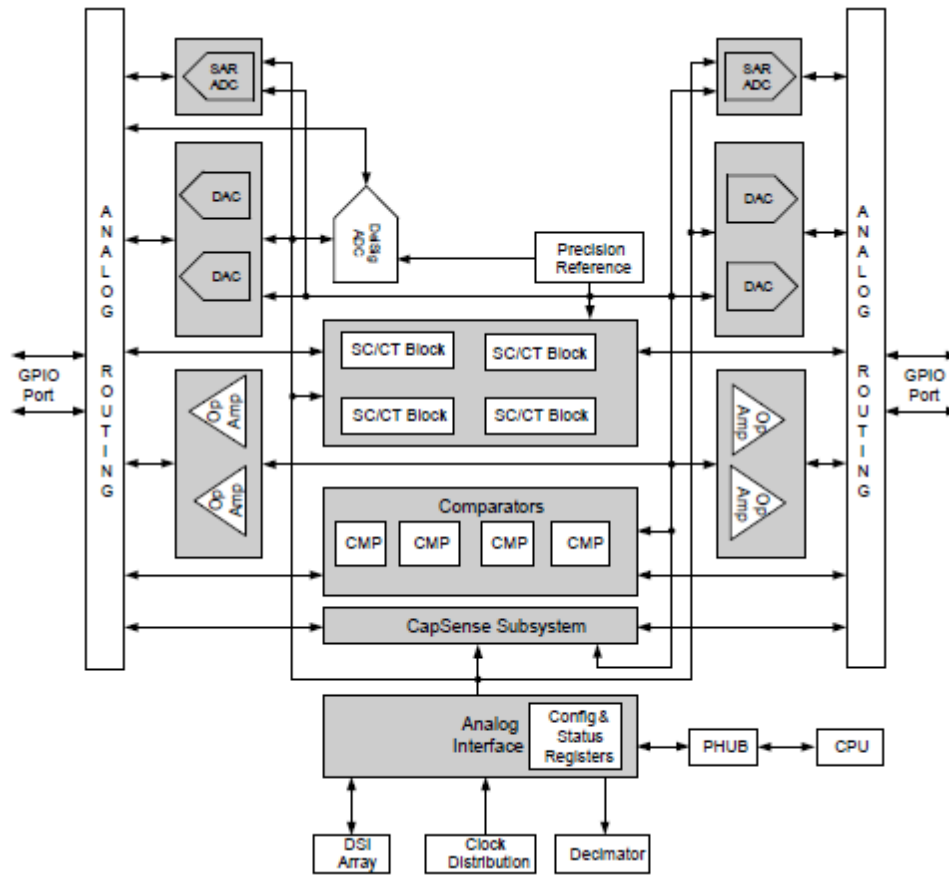




PSoC 5LP: podstawowe parametry

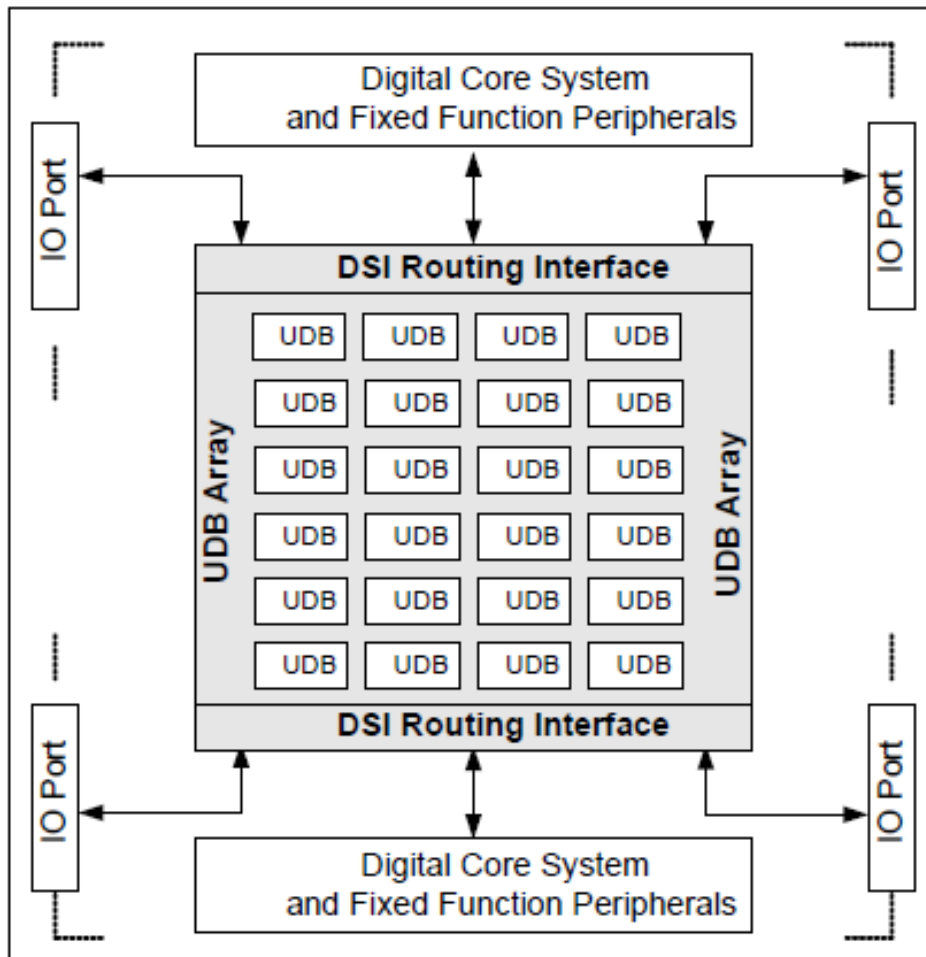
- Charakterystyki pracy:
 - napięcie: 1.71 to 5.5 V, 6 dziedzin zasilania
 - temperatura: -40 to 85°
 - częstotliwość: DC do 80-MHz
 - praca w różnych rodzajach poboru mocy: aktywny 3.1 mA dla 6 MHz, i 15.4 mA dla 48 MHz, 2- μ A sleep mode, 300-nA hibernacja z utrzymaniem RAM
 - regulacja napięcia wyjściowego 0.5-V do 5-V
- 32-bit ARM Cortex-M3 CPU
- Programowalny system zegarowy
 - wewnętrzny i zewnętrzny (krystaliczny) oscylator
 - generator PLL do 80 MHz
 - programowane dzielniki częstotliwości
- System I/O
 - 46 to 72 I/O pins – do 62 general-purpose I/Os (GPIOs)
 - 2 piny USBIO
 - bezpośrednie sterowanie LCD z dowolnego GPIO
 - wspomaganie CapSense (urządzenie dotykowe produkcji Cypress) z dowolnego GPIO

Część analogowa



- do dwóch 8- do 20-bitowych przetworników ADC delta-sigma
- do dwóch 12-bit SAR ADC
- cztery 8-bit DAC
- 4 komparatory
- 4 wzmacniacze operacyjne
- 4 programowalne bloki pozwalające na stworzenie:
 - wzmacniacza z programowanym wzmocnieniem (PGA)
 - wzmacniacza transimpedancyjnego(TIA)
 - mieszacza
 - układu Sample & Hold
 - układu obsługi CapSense®
 - źródła napięcia odniesienia: 1.024 V $\pm 0.1\%$

Część cyfrowa: matryca UDB



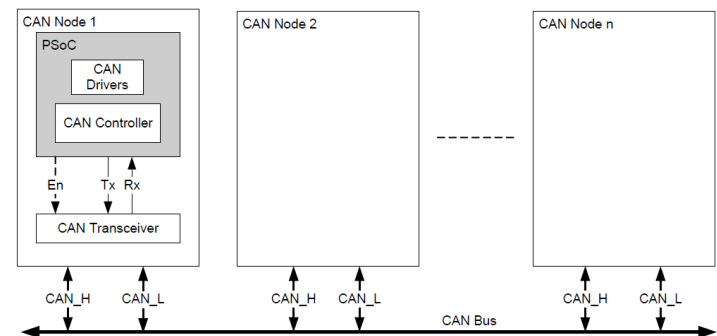
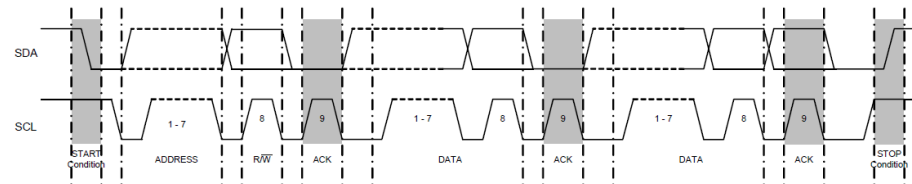
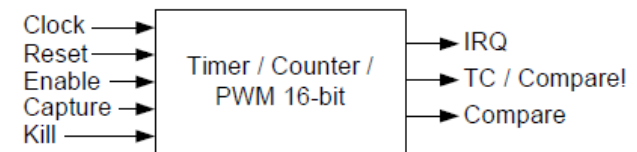
Część cyfrowa zbudowana jest z :

- matrycy uniwersalnych bloków cyfrowych (Universal Digital Block UDB)
- systemu połączeń bloków cyfrowych (Digital System Interconnect DSI) – bezpośrednie połączenie z blokami I/O, systemem przerwań, DMA lub wbudowanymi peryferiami cyfrowymi

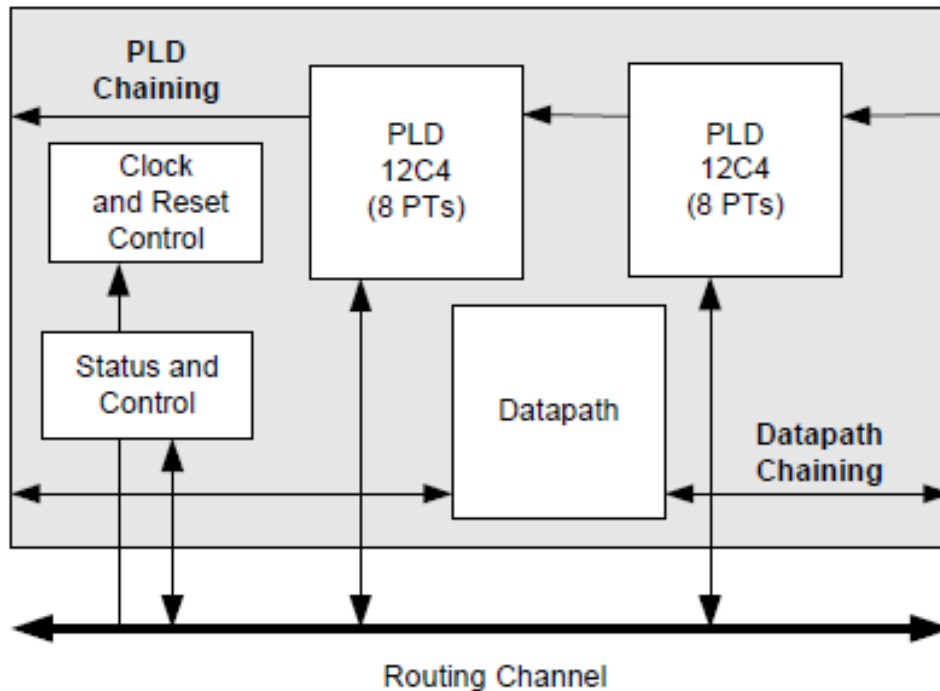
Część cyfrowa: pozostałe zasoby cyfrowe

Poza matrycą UDB i systemem połączeń dostępne są gotowe bloki (peryferia) cyfrowe takie jak:

- Cztery 16-bitowe bloki TCPWM, konfigurowane jako Timer/Counter, Pulse Width Modulator (PWM), dekodery kwadraturowe
- Kontroler magistrali I2C, 1 Mbps (szeregowy port z sygnałem w standardzie I2C)
- Interfejs USB 2.0 Full-Speed (FS) 12 Mbps
- Magistrala Controlled Area Network CAN 2.0b, 16 buforów odbiorczych, 8 nadawczych



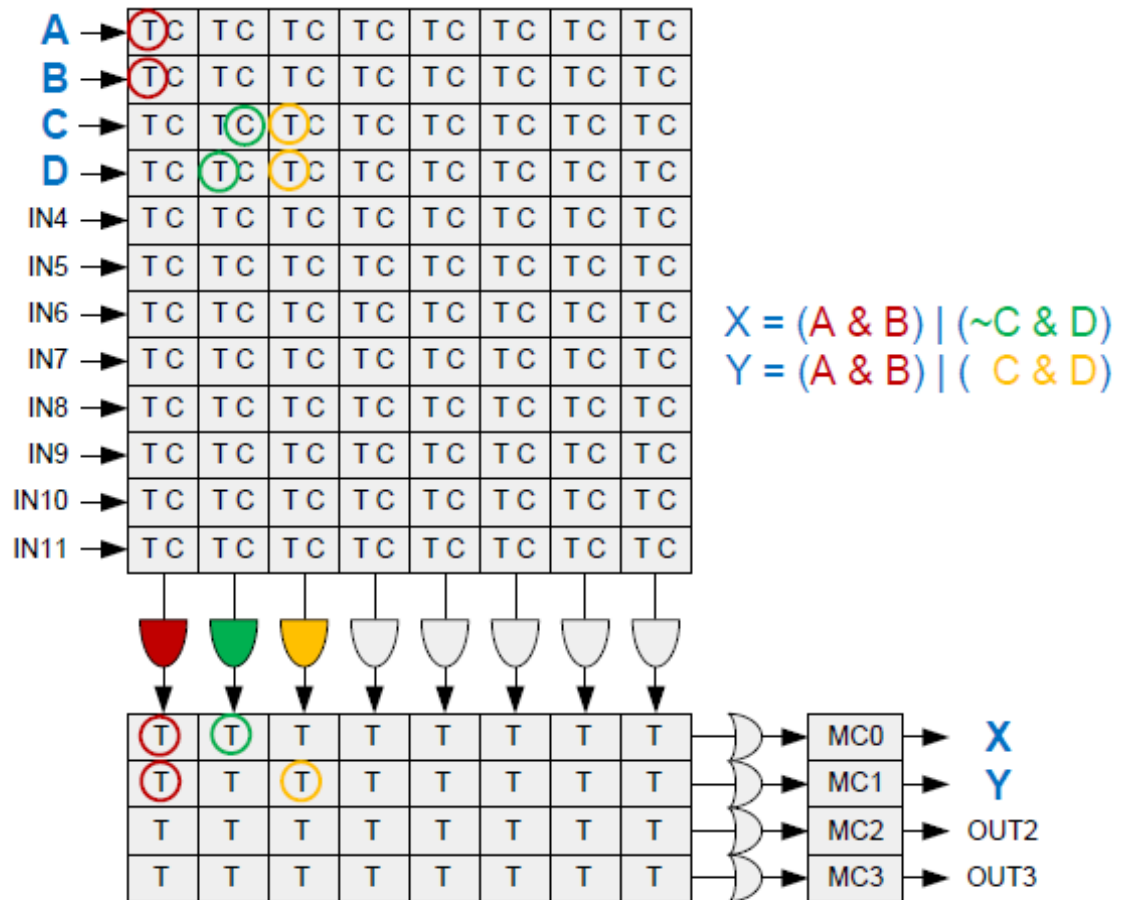
UDB Universal Digital Block



- UDB to podstawowy zasób do tworzenia logiki cyfrowej – funkcji logicznych. Zawiera 2 matryce PLD układ przetwarzania danych datapath oraz elastyczny system konfigurowania połączeń.
- Architektura UDB typu *slice* pozwala na łączenie z innymi blokami UDB i tworzenie w ten sposób np. funkcji dla argumentów o większych rozmiarach.

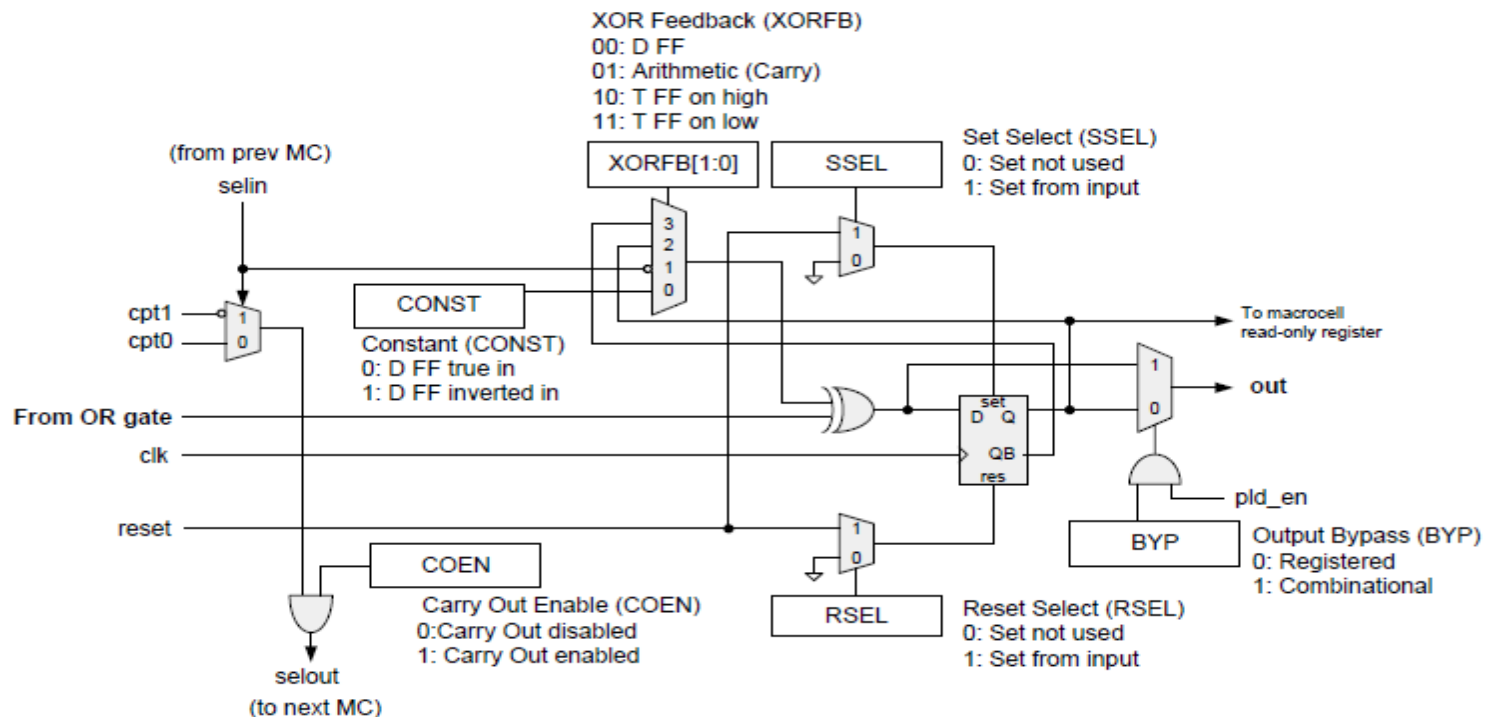
PLD

- PLD pobierają wejścia (12) z systemu połączeń i mapują funkcje boolowskie.
- Zwiera matrycę komórek AND i OR służącą do tworzenia logiki typu *sum of product*.
- Dla każdego argumentu dostępna jest wartość True (T) i jej negacja (Complement C).
- Wynik sumowania (or) jest podawany na wejście makrokomórki MC



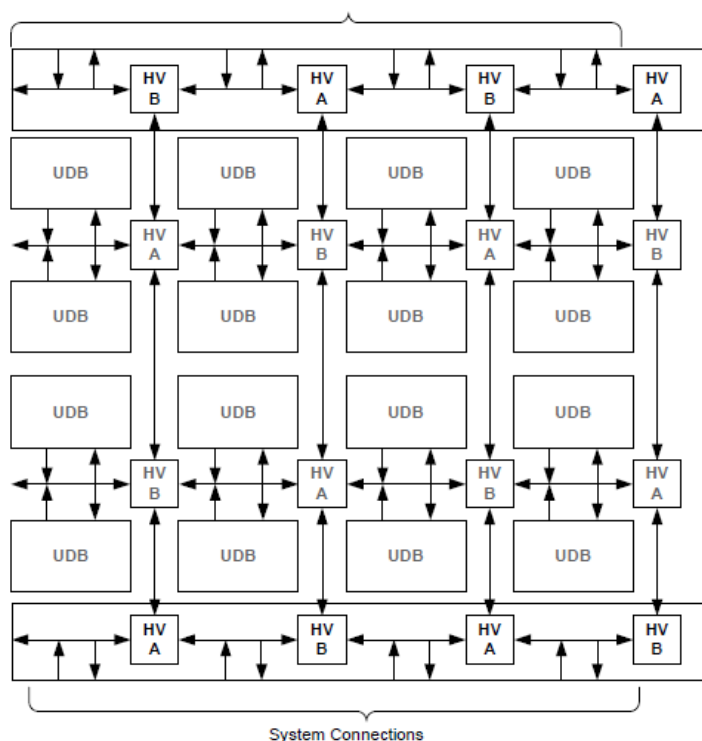
Struktura makromórki MC

- Makromórka to rejestr z dodatkową logiką kombinacyjną (przełączającą).
- Dzięki możliwości zapamiętywania wyników funkcji boolowskich w rejestrach makrokomórek PLD mogą być wykorzystywane do konfigurowania układów FSM.

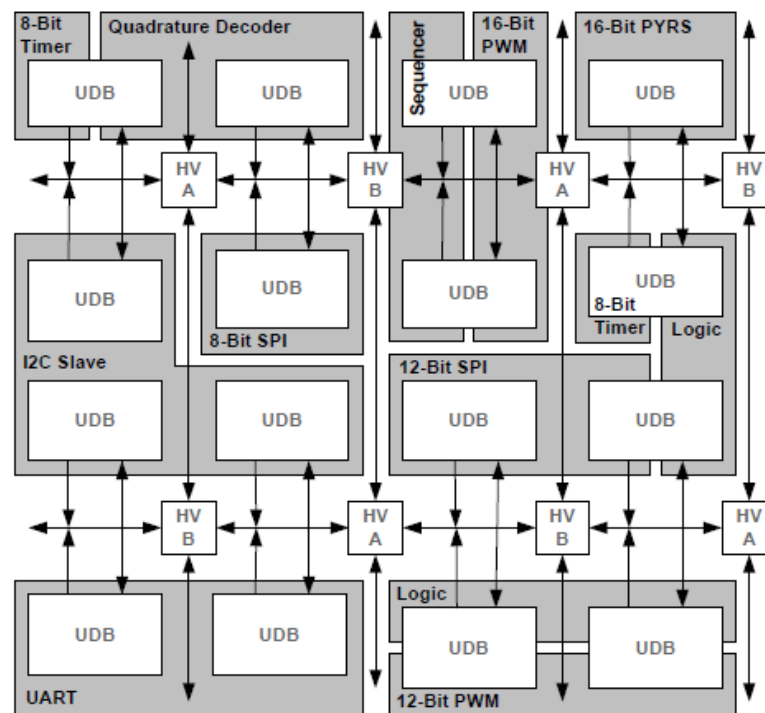


Matryca UDP

Matryce zawierają 16 – 24 UDB oraz system połączeń poziomych (horizontal H) i pionowych (vertical V) o rozmiarze 96 bitów każda. Rysunek przedstawia architekturę 16 UDB.

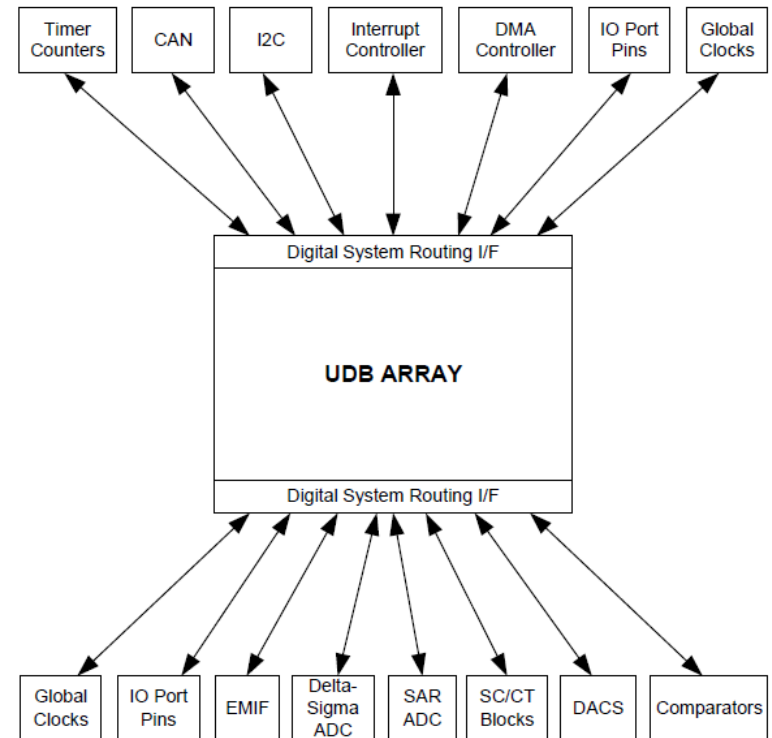


Przykład odwzorowania bloków cyfrowych w matrycy UDB.



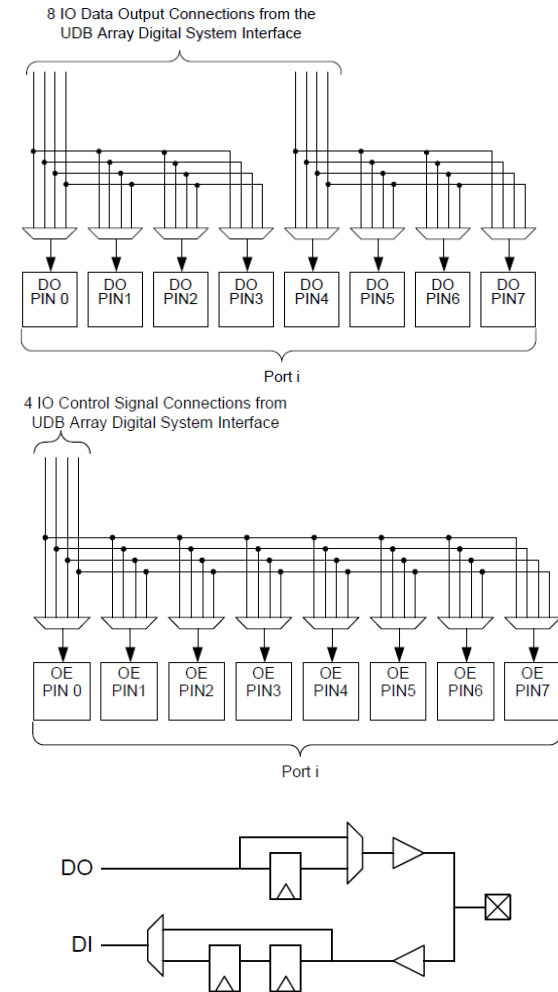
DSI *routing interface*

- Jest to system łączący globalne szyny matrycy UDB z innymi komponentami (peryferiami) układu PSoC.
- Wszystkie cyfrowe bloki konfigurowalne oraz o ustalonych funkcjach są dołączone do tego systemu.
- W szczególności następuje tu łączenie z komórkami I/O.



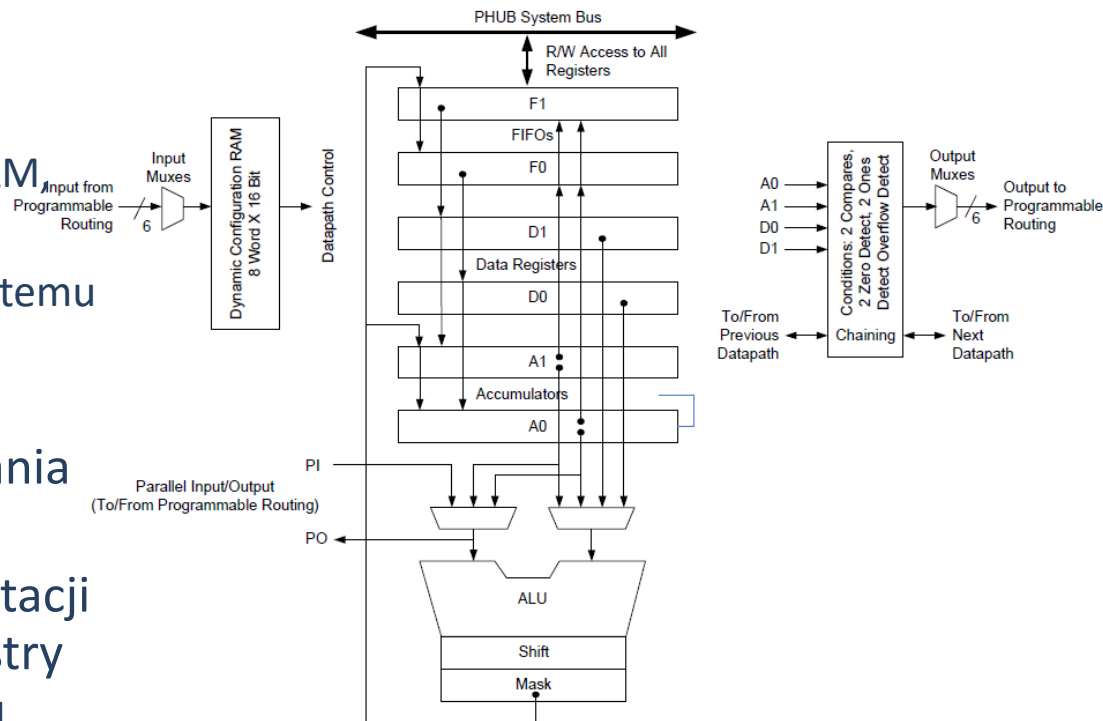
Obsługa cyfrowych portów I/O

- 16 8-bitowych kanałów DSI łączy dane z portami GPIO, mogącymi pracować jako wejściowe, wyjściowe lub dwukierunkowe.
- 4 kanały DSI (również 8-bitowe) zawierają sygnały sterujące wyznaczające tryb pracy poszczególnych GPIO (we czy wy)
- Sygnały I/O mogą być synchronizowane (zalecane): wyjścia jednym przerzutnikiem (one flip-flop synchronization), wejścia 2-ma ff.



Układ operacyjny (data path)

- Zawiera:
 - 8-bitowe ALU,
 - rejestry robocze,
 - rejestr statusu i rozkazu,
 - dynamicznie konfigurowaną RAM,
 - komparatory,
 - trasowanie poprzez FIFO do systemu rekonfigurowalnych połączeń,
 -
- Możliwość łączenia w rozwiązania 16-bitowe.
- Optymalizowany do implementacji bloków takich jak liczniki, rejestry przesuwające, różnego rodzaju generatory.

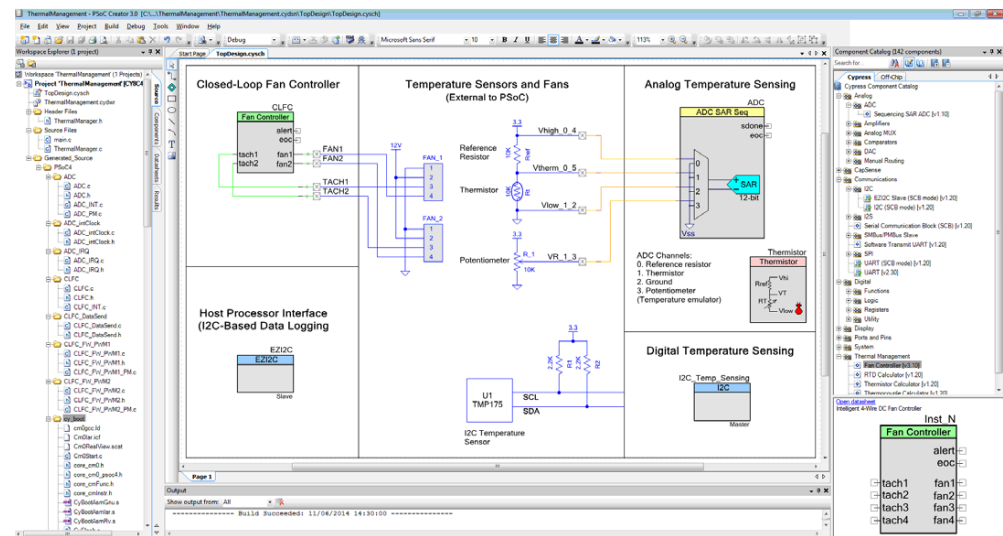


ALU: operacje

- 8 funkcji ogólnego zastosowania:
 - Increment
 - Decrement
 - Add
 - Subtract
 - Logical AND
 - Logical OR
 - Logical XOR
 - Pass, used to pass a value through the ALU to the shift register, mask, or another UDB register
- 4 niezależne operacje:
 - Shift left
 - Shift right
 - Nibble swap (zamiana półbajtów)
 - Bitwise OR mask

PSoC Creator

- IDE - narzędzie do projektowania układów PSoC
 - Bogata biblioteka konfigurowanych bloków analogowych i cyfrowych
 - Rozbudowany interfejs graficzny, możliwość graficznego projektowania struktury i parametryzowania bloków
 - Tworzenie SW (programowanie CPU) i HW
 - Prosty mechanizm rekonfigurowania PSoC (programowania)
- **Możliwość tworzenia własnych komponentów bibliotecznych z modelu w Verilog**
- Dobra dokumentacja
- Rozbudowane przykłady
- Niestety tylko Windows



Języki opisu sprzętu: standardy IEEE

- Verilog, IEEE 1364-1995
- Produkt komercyjny, rozpowszechniany od ok.1984 r., wspomagany przez wszystkich ważnych dostawców narzędzi EDA.
- Baza dla praktycznie wszystkich znaczących narzędzi syntezy.
- Dobre wspomaganie projektowania ale również **weryfikacji** układów specjalizowanych o średniej złożoności.
- Rozwinięty w standard SystemVerilog (IEEE 1800-2005).
- VHDL, IEEE 1076-1987
- Produkt publiczny, powstał na zamówienie rządu USA
- Charakteryzuje się bardziej precyzyjną semantyką.
- Nie ma możliwości modelowania na poziomie kluczy (dlatego synteza wykonywana jest do Verilog a nie VHDL).

Użycie VHDL czy Verilog zależy obecnie bardziej od przyzwyczajeń projektantów niż ma uzasadnienie merytoryczne. Po rewizji standardu Verilog w 2001 (Verilog2001 i Verilog2005) obserwuje się wzrost wykorzystania Verilog, również w Europie.

Standardy dotyczące Verilog i SystemVerilog

- Symulacja
 - IEEE 1364-1995: standard języka opisu sprzętu bazującego na Verilog Hardware Description Language
 - IEEE 1364-2001, IEEE 1364-2005: standard Verilog Hardware Description Language
 - IEEE 1800-2005: SystemVerilog standard języka opisu i weryfikacji sprzętu
 - IEEE 1800-2009: Verilog włączony do standardu SystemVerilog
 - najnowsza wersja: IEEE 1800-2017
- Synteza
 - 1364.1-2002 IEEE Standard for Verilog Register Transfer Level Synthesis
 - IEC 62142-2005 zastępuje standard IEEE Std 1364.1 Verilog Register Transfer Level Synthesis

Hierarchia w Verilog: moduły

- Jednostką hierarchii jest moduł (*module*).
- W Verilog nie ma podziału na interfejs i część wykonawczą – interfejs (wyprowadzenia) podawany jest bezpośrednio w nagłówku modułu.
- Tryb i typ są deklarowane wewnątrz modułu lub w nagłówku [od Verilog 2001].

```
module jk_flop_case (input j, k, clock, rst, output nq, output reg q);
```

← nagłówek

```
assign nq = ~q;
```

← przypisanie ciągłe

```
always @ (posedge clock or posedge rst)
```

```
begin
```

```
    if (rst == 1)
```

```
        q = 0;
```

```
    else
```

```
        case ({j, k})
```

```
            2'b00: q = q;
```

```
            2'b01: q = 0;
```

```
            2'b10: q = 1;
```

```
            2'b11: q = ~ q;
```

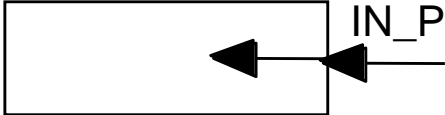
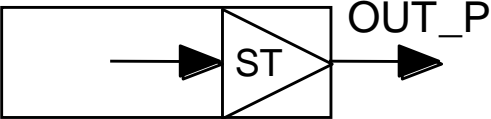
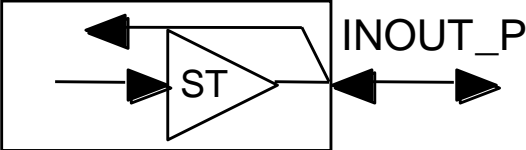
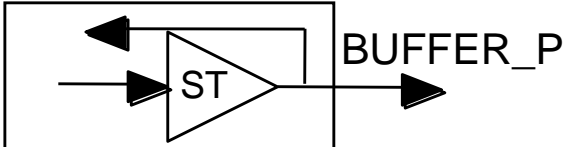
```
        endcase
```

```
    end
```

```
endmodule
```

← blok proceduralny

Wyprowadzenia – tryby (kierunki)

Verilog	VHDL		
input	in		wartość tylko czytana wewnątrz jednostki/modułu
output	out		wartość tylko zapisywana wewnątrz jednostki/modułu
inout	inout		wartość czytana i zapisywana wewnątrz jednostki/modułu
	buffer		port wyjściowy z jednym sterownikiem

Hierarchia: instancja modułu

- Tworzenie hierarchii w Verilog polega na wywołaniu modułu. Wszystkie moduły mogą znajdować się w jednym pliku.
- Dopiero Verilog 2001 wprowadza standard konfiguracji, ale koncepcja biblioteki różni się od VHDL .

nazwa_modułu #(odwzorowanie_parametrów)

nazwa_instancji(odwzorowanie_połączeń_modułu)

Przyporządkowanie niejawne

```
module ffnand (q, nq, set, clear);  
output q, nq;  
input  set, clear;  
  
nand    #(5,7) g1(q, nq, set),  
                g2(nq, q, clear);  
endmodule
```

Przyporządkowanie jawne

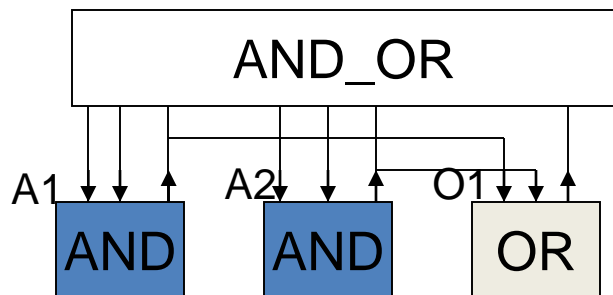
```
module ffnand (q, nq, set, clear);  
output q, nq;  
input  set, clear;  
  
nand    g1(.out(q), .a(nq), .b(set)),  
                g2(.out(nq), .a(q), .b(clear));  
endmodule
```

Odwzorowanie wyprowadzeń, Verilog

- Moduły na niższym poziomie hierarchii:

```
module AND (out, in1, in2);  
    output out;  
    input in1, in2;  
  
    assign out = in1 & in2;  
endmodule
```

```
module OR (out, in1, in2);  
    output out;  
    input in1, in2;  
  
    assign out = in1 | in2;  
endmodule
```



- Moduł na wyższym poziomie hierarchii:

- Niejawne mapowanie portów (przez zachowanie tej samej kolejności portów i odpowiadających im sygnałów)

```
module AND_OR (y, x1,x2, x3, x4);  
    output y;  
    input x1, x2, x3, x4;
```

```
    AND A1 (x12, x1, x2);  
    AND A2 (x34, x3, x4);  
    OR O1 (y, x12, x34);
```

```
endmodule
```

- Jawne mapowanie portów (przez określenie które porty mają być podłączone do których sygnałów)

```
module AND_OR (y, x1,x2, x3, x4);  
    output y;  
    input x1, x2, x3, x4;
```

```
    AND A1 (.in1(x1), .in2(x2), .out(x12));  
    AND A2 (.in1(x3), .in2(x4), .out(x34));  
    OR O1 (.in1(x12), .in2(x34), .out(y));
```

```
endmodule
```

Nazwy hierarchiczne

Nazwy hierarchiczne tworzy się poprzez złożenie nazw modułów.

W **Verilog** istnieje możliwość bezpośredniego, jednoznacznego odwołania się do obiektu wewnętrznego dowolnego nazwanego bloku (np. modułu, zadania, procesu). Pozwala to na nadanie lub pobranie wartości sygnału w module wewnętrznym.

Przykład:

```
module moduleA;  
  moduleB  wyst1(), wyst2();  
  initial  
  begin  
    wyst1.x = 1;  
    wyst2.x = 2;  
  end  
endmodule
```


Konfiguracja

- Do Verilog 2001 nie było możliwości konfiguracji. Wykorzystywane w hierarchii moduły musiały być zapisane w jednym pliku i każdorazowo w całości kompilowane.
- Konfiguracja została wprowadzona w Verilog 2001, aby zestandaryzować (uniezależnić od narzędzia) sposób dołączania modułów opisanych w różnych plikach. Pozwala na pokazanie położenia kodu modułu dla każdej instancji.
 - Blok konfiguracji jest opisany poza modułami, może być w oddzielnym pliku,
 - W kodzie Verilog specyfikowane są wirtualne biblioteki,
 - Fizyczne położenie kodu modułu podane jest w pliku „map”.

Verilog Design

```
module test;  
...  
    myChip dut (...);  
...  
endmodule
```

```
module myChip(...);  
...  
    adder a1 (...);  
    adder a2 (...);  
...  
endmodule
```

Library Map File

Configuration Block (part of Verilog source code)

```
/* define a name for this configuration */  
config cfg4  
  
/* specify where to find top level modules */  
design rtlLib.test  
  
/* set the default search order for finding  
   instantiated modules */  
default liblist rtlLib gateLib;  
  
/* explicitly specify which library to use  
   for the following module instance */  
instance test.dut.a2 liblist gateLib;  
endconfig
```

```
/* location of RTL models (current directory) */  
library rtlLib ./*.v;  
  
/* Location of synthesized models */  
library gateLib ./synth_out/*.v;
```

Parametry

- Są to stałe, które pozwalają na przepływ (statyczny) informacji pomiędzy modułami.
- Oznacza to, że w momencie tworzenia instancji ustawiane są wartości parametrów.
- **Parametry stanowią część interfejsu**, dlatego są deklarowane w nagłówku lub w części deklaracyjnej modułu.
- Verilog2001 zezwala na definicję parametrów w nagłówku, np.:

```
module register2001 #(parameter SIZE=8) (output reg [SIZE-1:0] q, input [SIZE-1:0] d, input clk, rst_n);
```

oraz redefinicję parametru w instancji jako tzw. *named parameter* np.:

```
register2001 #(.SIZE( 16)) u1 (.q(n1), .d (d), .clk(clk), .rst_n(rst_n));
```

Parametryzowanie

- Parametr definiuje się w części deklaracyjnej modułu, a zmiana wartości parametru dla komponentu może nastąpić:

1. w instrukcji powołania instancji poprzez powiązanie niejawne, np.:

- `nand #(5,7) g1(q, nq, set),`

- Moduł na niższym poziomie hierarchii z parametrami `size` i `delay`:

```
module vdff (out, in, clk);  
    parameter size = 1, delay = 1;  
    input [0:size-1] in;  
    input clk;  
    output [0:size-1] out;  
    reg [0:size-1] out;  
  
    always @(posedge clk)  
        # delay out = in;  
  
endmodule
```

- Moduł na wyższym poziomie hierarchii deklarujący komponenty z parametrami:

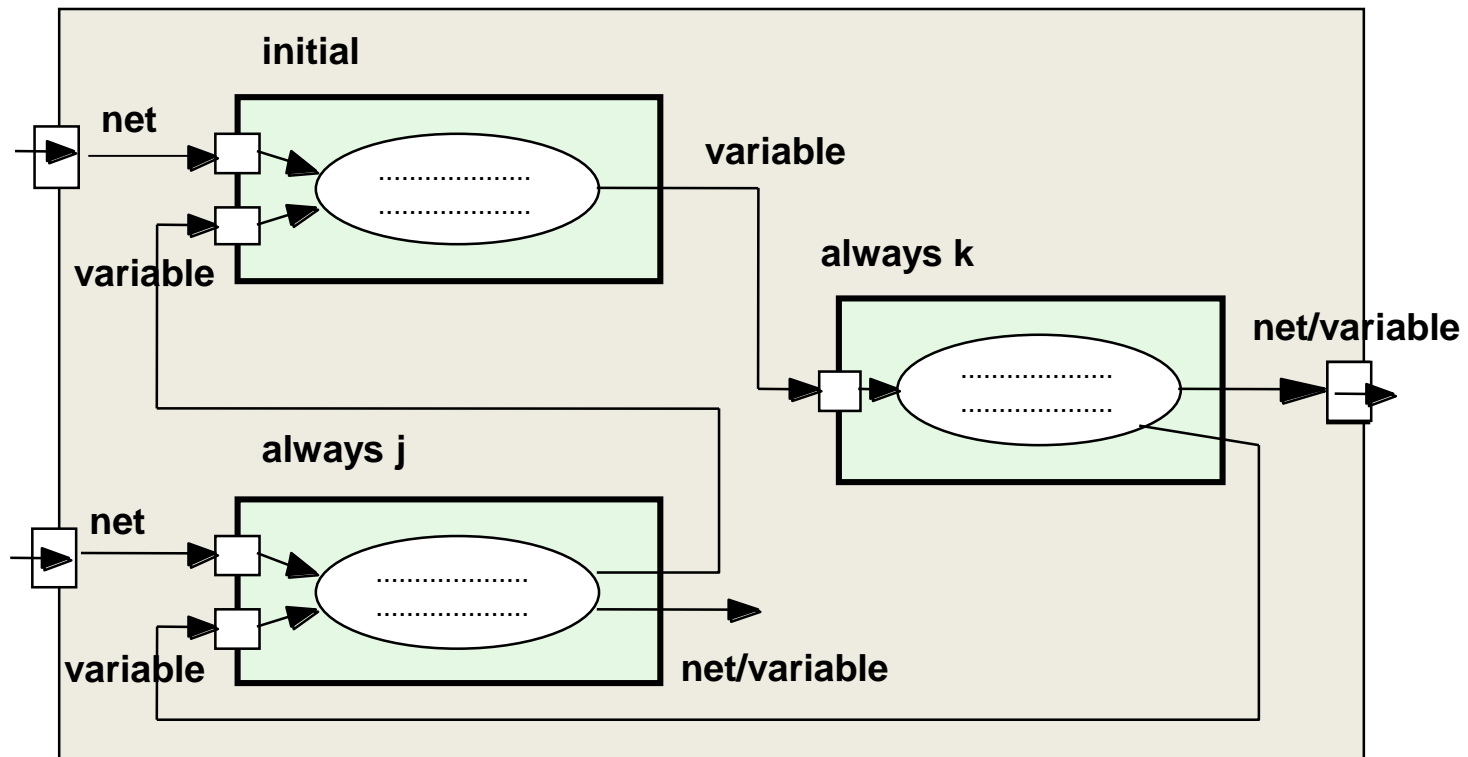
```
module m;  
    reg clk;  
    wire[1:10] out_a, in_a;  
    wire[1:5] out_b, in_b;  
  
    // create an instance and set  
    // parameters  
    vdff #(10,15) mod_a(out_a, in_a, clk);  
    // create an instance leaving default  
    // values  
    vdff mod_b(out_b, in_b, clk);  
  
endmodule
```

2. lub przy pomocy redefinicji **defparam** (działa od momentu wystąpienia), np.:

```
module ustal_parametry;  
    defparam top.mod1.size = 5, top.mod2.delay = 2;  
endmodule
```
3. lub poprzez makro definicję `'define`. To rozwiązanie nie jest zalecane, szczególnie przy złożonej hierarchii.

Modelowanie działania: procesy

- Do modelowania działania wykorzystywane są instrukcje wykonywane w sposób równoległy – przypisania ciągłe i bloki proceduralne.
- Zwyczajowo instrukcje/bloki równoległe nazywamy **procesami**.



Model działania: moduł

- Verilog nie posiada oddzielnego bloku na opis działania/struktury układu. Następuje to w module, po części deklarycyjnej.

module nazwa_Verilog (lista portów)

deklaracje:

(parametrów, portów, obiektów
wewnętrznych)

- instrukcje ciągłe
- bloki proceduralne
- powołania instancji

endmodule

Rodzaje instrukcji

- Do modelowania zachowania asynchronicznego (kombinacyjnego) wykorzystywane są instrukcje nazywane ciągłymi (czasem nieproceduralnymi).
- **Verilog** zachowuje się inaczej niż VHDL, który rozróżnia te instrukcje na podstawie miejsca ich wystąpienia. To czy instrukcja jest **ciągła** zależy głównie od **typu obiektu**, którego dotyczy, ale klasyfikacja jest znacznie bardziej elastyczna i skomplikowana. W pierwszym przybliżeniu można przyjąć, że typem charakterystycznym dla przypisania ciągłego jest **wire (net)**.
- Wewnątrz niektórych bloków instrukcje wykonywane są sekwencyjnie, ale nie oznacza to implementacji “wprost” takiego zapisu jako układ sekwencyjny. Sekwencyjność dotyczy działania symulatora a nie budowy modelowanego sprzętu.

Verilog: instrukcje równoległe

Przypisania:

- przypisanie ciągłe (*continuous assignment*)

Bloki:

- always
- initial
- fork-join

Budowanie struktury:

- powołanie instancji modułu (*module instantiation*)
- generacja (dopiero od 2001r.)

Instrukcje pomocnicze:

- wywołanie zadania (*task*) lub funkcji (*function*)

Verilog: przypisania

- Przypisanie ciągłe (również proceduralne ciągłe) oznacza, że lewa strona przypisywania jest ciągle sterowana wartością wyrażenia po stronie prawej.
- Przypisania proceduralne występują w blokach proceduralnych i mogą być blokujące lub nieblokujące. Blokujące zachowują się jak przypisania zmiennych w językach programowania, tzn. blokują wykonanie kolejnych instrukcji dopóki nie zostaną wykonane.

Rodzaj przypisania	Obiekt otrzymujący wartość
ciągłe	Sieć (<i>net</i>): obiekt typu net (wektor lub skalar) fragment wektora typu net lub sklejenie
proceduralne blokujące	Zmienne (<i>variables</i>): obiekt typu reg (wektor lub skalar) fragment wektora typu reg element pamięci obiekty typu real, integer i time sklejenie w/w
<i>proceduralne nieblokujące (ciągłe)</i>	<i>sieci i zmienne</i>

Verilog: instrukcje przypisania ciągłego

- Przypisanie ciągłe jest zawsze aktywne
- Następuje z opóźnieniem, które można wyspecyfikować
- **Możliwe jest przypisanie do sklejonych wektorów**

Continuous assignment statement:

assign [<opóźnienie>] <lista_przypisań>

Przykład1:

```
wire a;  
assign a = b | (c & d);
```

Przykład2:

```
assign #10 a = b | (c & d);
```

Przykład3:

```
wire carry_in, carry_out;  
wire [3:0] a, b, sum;  
  
assign {carry_out, sum} = a + b + carry_in;
```

Net declaration assignment :

Możliwe jest również przypisanie w deklaracji np. `wire a= x & y;`

Bloki proceduralne

W Verilog istnieją 4 bloki określone jako proceduralne:

- always
 - initial
 - task
 - function
-
- Instrukcje zawarte w blokach **initial** i **always** są wykonywane poczynając od czasu symulacji $t = 0$, blok initial jednokrotnie, blok always w nieskończonej pętli. Bloki initial i always są więc **procesami** wykonywanymi równolegle.
 - Zadania i funkcje to podprogramy - wykonywane są wtedy, gdy zostaną wywołane w innych blokach.

Verilog: always

- Instrukcja bloku **always** jest to, obok initial, podstawowy blok wykorzystywany do modelowania funkcjonalnego.
- Jak wszystkie bloki proceduralne, zawiera instrukcje sekwencyjne zamknięte pomiędzy begin i end. Wykonują się one w nieskończonej pętli. Bloki always wykonują się równolegle.
- Zawieszanie wykonania bloku always sterowane jest opóźnieniem (przykład 1), instrukcją wait lub zdarzeniem (przykład 2). Zdarzeniem może być wykrycie zbocza lub poziomu obiektu.
- W procesach kombinacyjnych sterowanie może być domyślne (przykład 3).

Przykład 1:

```
always #20 clock = ~clock;  
  
always clock = #20 ~clock;
```

Przykład 2:

```
always @(b or d)  
begin  
    a <= b;  
    c <= d;  
end
```

Przykład 3 (=2):

```
always (*)  
begin  
    a <= b;  
    c <= d;  
end
```

Instrukcje proceduralne

- Występują wewnątrz bloków proceduralnych
 - przypisanie (\leftarrow) proceduralne ciągle
 - przypisanie zmiennej ($=$)
 - wait
 - if-else
 - case, casex, casez
 - instrukcje pętli: repeat, forever, while
 - disable
 - return
- wywołanie task, function

Verilog: przypisania proceduralne

- Służą do modyfikowania wartości przechowywanych w obiektach określanych jako zmienne, tj. obiektach: typu reg, integer, real, time oraz pamięciach.
- Mogą występować tylko w **blokach proceduralnych**, jako przypisania **blokujące** lub **nieblokujące**.
- **blokujące**, =, wstrzymuje wykonanie kolejnych instrukcji sekwencyjnych do momentu wykonania danego przypisania (tak jak instrukcje w programowaniu)

```
rega = 0;  
rega[3] = 1; // a bit-select  
rega[3:5] = 7; // a part-select  
mema[address] = 8'hff; // assignment to a mem element  
{carry, acc} = rega + regb; // a concatenation
```

- **nieblokujące**, <=, pozwala na wykonywanie kolejnych instrukcji zanim nastąpi aktualizacja wartości (jak przypisanie sygnału w VHDL)

```
initial begin  
d <= 1; // d will be assigned 1 at time 0  
e <= #2 0; // e will be assigned 0 at time 2  
f <= #4 1; // f will be assigned 1 at time 4 (not 2+4)  
end
```

Przypisania w Verilog: analogie do VHDL

Verilog

- Przypisanie ciągłe (nieproceduralne):
`assign a = b;`
`assign #10 a = b;`
- Przypisanie blokujące:
`initial begin`
`b = 1;`
`a = b; // a = 1`
`end`
- Przypisanie nieblokujące (proceduralne ciągłe):
`always @(posedge c) begin`
`b <= 1;`
`a <= b; // a zmienia wartość na 1 w 2-gim takcie`
`end`

VHDL

- Przypisanie współbieżne:
`a <= b;`
`a <= b after 10 ns;`
- Przypisanie zmiennej:
`process (c)`
`variable a,b: integer;`
`begin`
`b := 1; a := b; -- a = 1`
`end`
- Sekwencyjne przypisanie sygnału:
`process (c) begin`
`if (clk'event and clk = 1) then`
`b <= 1;`
`a <= b; -- a zmienia wartość na 1 w 2-gim takcie`
`end if;`
`end`