

# UKTADY CYFROWE i SYSTEMY Wbudowane

3ECTS

WYKtAD

dr inż. Stanisław Sugier C-3/224

if (Ocena\_Kol == 2.0 || Ocena\_Lab == 2.0)

Ocena\_Indeks = 2.0;

else

Ocena\_Indeks = 0,6 \* Ocena\_Kol + 0,4 \* Ocena\_Lab;

kolokwium - 20.01.2012

LABORKI

dr inż. Jacek Majewski C-3/224

wkład CPLD

środowisko Xilinx

folder Xilinx\_XC9572

## JĘZYK VHDL

wystosowanie:

- specyfikacja / modelowanie
- symulacja
- syntez

## 1) Jednostki i architektury

- Każdy model = jednostka (entity) + architektura (-y) (architecture)

```
entity HalfAdder is
    port (A : in STD_LOGIC;
          B : in STD_LOGIC;
          S : out STD_LOGIC;
          C : out STD_LOGIC);
end entity HalfAdder;
```

- Nierozróżnialne wielkości liter

- Hasła:

- porty in/out
- typ STD\_LOGIC
- instancje komponentów, bibliotek, port map
- znaczenie operatorów "<=", operatory "and", "or"

library UNISIM;

Architecture Structural of HalfAdder is  
begin

XOR\_gate : XOR2 port map (A, B | S);

AND\_gate : AND2 port map (A, B | C);

end architecture Structural;

architecture Dataflow of HalfAdder is

begin

S &lt;= A &amp; B;

C &lt;= A &amp; B;

end architecture Dataflow;

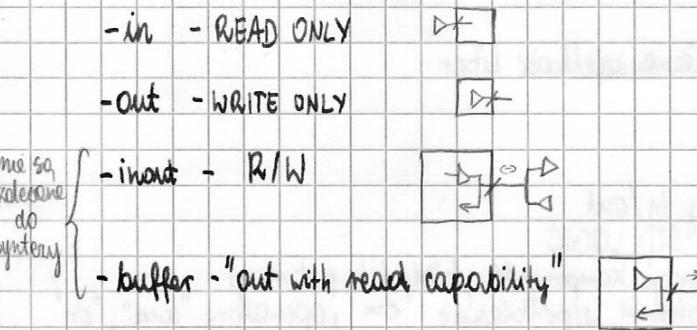
! - opisy sekwencyjne: w instrukcji 'process'

! - opisy użycięsne: treść architektury ('kolejnośc' miejstwa)

- Poziomy opisu
  - strukturalny (instancje komponentów)
  - preotypowy (dane od WE do WY)
  - behavioralny/skryptowy/proceduralny (opis działania jednostki)
- Opis na poziomie Register Transfer Level (RTL) połączenie opisu preotypowego i behavioralnego

### 1.1) Porty i sygnały

#### • Tryby portów



Identyfikator VHDL: ciąg liter, cyfr i znaku podkreślenia '-' zaczynający się od litery; po znaku '-' musi występować litera lub cyfra

```
entity AndNand is
  port (A : in STD_LOGIC;
        B : in STD_LOGIC;
        C : in STD_LOGIC;
        WY_And : out STD_LOGIC;
        WY_Nand : out STD_LOGIC);
end AndNand;
```

```
architecture DataflowBard of AndNand is
begin
```

```
  WY_And <= A and B and C;
  WY_Nand <= not WY_And;
end DataflowBard;
```

## 1.2) Obiekty jawnie

- Signal

- specjalny model wartości
- możliwe wiele przyjazów współbieżnych  $\Rightarrow$  sterowanie, funkcje rozstypowate, bufory z stanem, sygnały OE (output enable)...
- aspect czasu  $\Rightarrow$  translakcje
- specjalny model symulatora

```
architecture DataflowOK of AndNand is
    signal Int_And : STD_LOGIC;
begin
    Int_And <= A and B and C;
    WY_And <= Int_And;
    WY_Nand <= not Int_And;
end DataflowOK;
```

- Constant

- constant Pi : REAL := 3,141;

- Variable

- variable PropDelay : TIME := 0; (nierzeczywiste w opisach do symulatora)
- etc. np. file

## 1.3) Klawisze generic

## 2) Typy i operatory

### 2.1) Pakiet STANDARD

```

-- komentarz
ignorowane
przy kompilator

type INTEGER is range -- usually typical INTEGER;
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
type REAL is range -- usually double precision f.p.--;
type BOOLEAN is (FALSE, TRUE);
type CHARACTER is (-- 256 characters --);
type STRING is array (POSITIVE range <>) of CHARACTER;
type BIT is ('0', '1');
type TIME is range -- implementation defined --;

type
    timeframe
        units
            fs          -- femtosecond
            ps = 1000 fs; -- picosecond
            ns = 1000 ps; -- nanosecond
            us = 1000 ns; -- microsecond
            ms = 1000 us; -- millisecond
            sec = 1000 ms; -- second
            min = 60 sec; -- minute
            hr = 60 min; -- hour
end units;
subtype DELAY_LENGTH is TIME range 0 to TIME'HIGH;
type BIT_VECTOR is array (NATURAL range <>) of BIT;

```

### Wielotony i mopsy bitowe

signal DataBus : BIT\_VECTOR (7 downto 0);

```

toki sam efekt
DataBus <= "10000000";           ignorowane przy kompilator,
DataBus <= B"1000_0000";          dla wykrojonego zapisu
DataBus <= X"80";
DataBus <= ('1', '0', '0', '0', '0', '0', '0', '0');
DataBus <= ('1', others => '0');
DataBus <= (7 => '1', others => '0');

DataBus <= (others => '0');

HalfByte <= DataBus (7 downto 4);

MSB <= DataBus (7);

```

## Operatory

(r. - result)

**	exponentiation	numeric ** integer	r. numeric
abs	absolute value	abs numeric	r. numeric
not	complement	not logic or boolean	r. same
*	multiplication	numeric * numeric	r. numeric
/	division	numeric / numeric	r. numeric
mod	modulo	integer mod integer	r. integer
rem	remainder	integer rem integer	r. integer
sll	shift left logical	log.array sll integer	r. same
srl	shift right logical	log.array srl integer	r. same
sla	shift left arith	log.array sla integer	r. same
sra	shift left arith.	log.array sra integer	r. same
rotl	rotate left	log.array rotl integer	r. same
rotr	rotate right	log.array rotr integer	r. same
=	equality	r. boolean	
/=	inequality	r. boolean	

! Operator and (or /nand/... /xnor) w wyniesieniu zapisu m. Tzw. sig  
(czytaj, ze ten sam typ)

$Y \leftarrow A \text{ and } B \text{ and } C ; -- \text{OK}$

$Y \leftarrow A \underline{\text{and}} \ B \underline{\text{or}} \ C ; -- \boxed{B \neq A}$

$Y \leftarrow A \text{ and } (B \text{ or } C) ; -- \text{OK}$

## Operator &

signal ASCII : STD\_LOGIC\_VECTOR (7 downto 0)

signal Digit : STD\_LOGIC\_VECTOR (3 downto 0)

ASCII <= "0011" & Digit ; -- X"3" & Digit

ASCII <= X"3" & Digit(3) & Digit(2) & Digit(1) & Digit(0);

-- Shift right

ASCII <= '0' & ASCII (7 downto 1);

-- Arithmetic shift right

ASCII <= ASCII (7) & ASCII (7 downto 1);

-- Shift left:

ASCII <= ASCII (6 downto 0) & '0';

## 2.2) Pakiet std\_logic\_1164

- IEEE Standard 1164 Multivalue Logic System for VHDL Model Interoperability
- 9 wartości

library IEEE;  
use IEEE.STD\_LOGIC\_1164.all;

treba napisac  
pred kazdym uzyaniem  
- mozna tylko  
do konca bloku



Type STD\_ULOGIC is

'U'	-- Uninitialized
'X'	-- Forcing Unknown
'0'	-- Forcing 0
'1'	-- Forcing 1
'Z'	-- High Impedance
'W'	-- Weak Unknown
'L'	-- Weak 0
'H'	-- Weak 1
'-'	-- Don't care

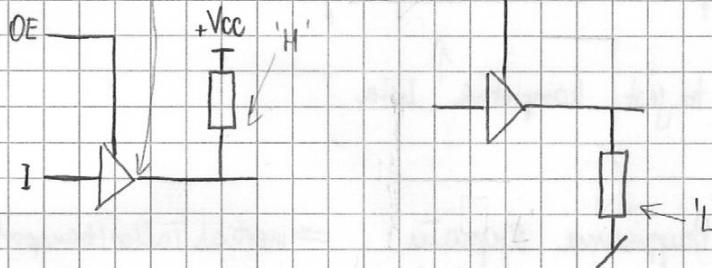
9 wartości!

microsimulatejacy

);

9 wartości : 3x "mocne" '0', '1', 'X'  
3x "słabe" 'L', 'H', 'W'

'0'/'1'/'Z'



3 wartości dodatkowe '0', 'Z', '-'

function resolved (s : STD\_ULOGIC\_VECTOR) return STD\_ULOGIC;

└ podjazd dolejip



## 3) Instrukcje wsparcia

- W treści architektury mogą wystąpić instrukcje wsparcia:

- wsparcie przypisanie  $\Leftarrow$
- instancja komponentu (component)
- instrukcja procesu (process)
- instrukcja generacji (generate)
- instrukcja bloku (block)
- wsparcie przygotowanie procedury
- wsparcie obliczanie asercji

W architekturze  
są tylko  
instrukcje wsparcia  
(nie mogą  
istnieć np.  
warunkowe)

$\equiv$  to jest kompletna ↑ lista

3.1) Przypisanie sygnatu  $\Rightarrow$  inertial / after / transport

$$y \Leftarrow x;$$

wspierająca zapis:  
me przypisuje  
impulsu skończyc  
na opóźnienie

$$y \Leftarrow (x_1 \text{ and } x_2) \text{ or } ((x_3 \text{ nand } x_4) \text{ xor } x_5);$$

$$y \Leftarrow \text{inertial } x \text{ after } 4 \text{ ms}$$

$$y \Leftarrow \text{transport } x \text{ after } 4 \text{ ms}$$

wszystko przypisuje

-- Domyslnie:

$$y \Leftarrow x; \Leftrightarrow y \Leftarrow \text{inertial } x \text{ after } 0 \text{ ms};$$

$$y \Leftarrow x \text{ after } 4 \text{ ms} \Leftrightarrow y \Leftarrow \text{inertial } x \text{ after } 4 \text{ ms};$$

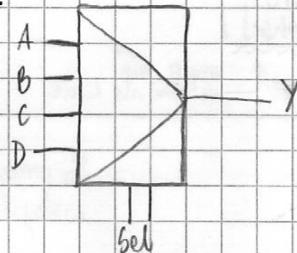
$$y \Leftarrow x \text{ after } 4 \text{ ms}, \text{ not } x \text{ after } 8 \text{ ms};$$

momenty czasu,  
w których następuje przypisanie

-- w opisach sekwencyjnych (np. powtarzanych w pętli)

$\text{Clk} \leftarrow '1', '0' \text{ after } \text{ClkPeriod}/2;$

multiplexer:



```
entity MUX_4 is
port (A,B,C,D : in STD-LOGIC;
      sel : in STD-LOGIC-VECTOR(1 downto 0);
      y : out STD-LOGIC);
end MUX_4;
```

```
architecture Dataflow of MUX_4 is
begin
```

```
    y <- A when Sel = "00" else
           B when Sel = "01" else
           C when Sel = "10" else
           D when Sel = "11" else
           'X';
end Dataflow;
```

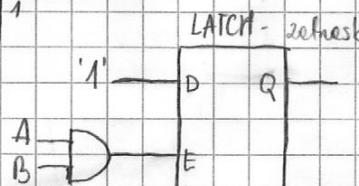
PRZYPISANIE WARUNKÓW  
when ... else

## ! UWAGA: ZATRZASKI

$Y \leftarrow '1' \text{ when } A = '1' \text{ and } B = '1';$

$\Leftrightarrow$  to nie jest jasne AND

A	B	Y
0	0	y
0	1	y
1	0	y
1	1	1



'else '0';  $\Leftrightarrow$  bramka AND poprawna

Niekompletna lista warunków

syntetyczne zatrasków (latches)

architecture Dataflow2 of MUX\_4 is  
begin

with Sel select

y <= A when "00",  
B when "01",  
C when "10",  
D when others;

end Dataflow2;

PRZYPISANIE SELEKTYWNE

with... select

musi być  
ostatni w liście

To pripisanie musi zawierać  
wszystkie opcje & nastęczne wartości

### 3.2) Instancje komponentów

⇒ konkretnie nazwa komponentu

```
entity XOR_2WE is
    generic( Tp : DELAY_LENGTH );
    port ( I1, I2 : in STD_LOGIC;
           O : out STD_LOGIC);
end XOR_2WE;
architecture A of XOR_2WE is
begin
    O <= I1 xor I2 after Tp;
end A;
```

```
entity AND_2WE is
    generic( Tp : DELAY_LENGTH );
    port ( I1, I2 : in STD_LOGIC;
           O : out STD_LOGIC);
end AND_2WE;
architecture A of AND_2WE is
begin
    O <= I1 and I2 after Tp;
end A;
```

```
entity HalfAdder is
(...)
architecture Structural of HalfAdder is
    component XOR_2WE is
        generic( Tp : DELAY_LENGTH );
        port ( I1, I2 : in STD_LOGIC; O : out STD_LOGIC);
    end component;
    component AND_2WE is
        generic( Tp : DELAY_LENGTH );
        port ( I1, I2 : in STD_LOGIC; O : out STD_LOGIC);
    end component;
begin
    XOR_gate : XOR_2WE generic map( 5 ns ) port map( A, B, S );
    AND_gate : AND_2WE generic map( Tp => 3 ns )
                port map( O=>C, I1=>A, I2=>B );
end architecture Structural;
```

### 3.3) Instrukcje procesu

```
[label:] process [(sensitivity-list)] [is]
  [declarative-items]
  begin
    sequential-statements
  end process [label];
```

w poziomie  
architektur  
tzw. SIGNATY

w poziomie  
procesów  
tzw. SIGNATY  
ZMIENNE

⇒ zmiana wartości dowolnego sygnału z listy wrażliwości powoduje wykonanie procesu

4) Instrukcje sekwencyjne — omówimy później

## 5) "VHDL CODING TECHNIQUES"

### 5.1) Metody kombinacyjne

- współbieżne przypisanie sygnałów

### 5.2) Opisy sygnałów synchronicznych

- Atrybut 'Event' = sygnał zmieniający wartości

<pre>entity DFF is   port ( D(T): in STD_LOGIC;         CLK : in STD_LOGIC;         Q : out STD_LOGIC); end DFF;</pre>	<p>(T/F)</p> <p>prezynik typu D - DFF lub typu T - TFF</p>
--	--

architecture RTL of DFF is

begin

process (Clk)

begin

if Clk'Event and Clk = '1' then

Q ← Di

end if;

end process;

end architecture;

opje z clock enable:

↳ if CE = '1' then

(dodac jez wtedy CE

do entity)

architecture RTL of TFF is

signal Q\_int: STD\_LOGIC := '0';

begin

Q ← Q\_int; opje z asynchronicznym clear:

process (Clk)

begin

if Clk = '1' then

Q ← 0;

elseif Clk'Event and Clk = '1' then

if T = '1' then

Q\_int ← not Q\_int;

end if;

end if;

end process;

end architecture;

asynchronicznego  
reset



Clk w losie  
wolnosci

synchronicznego  
reset



Clk tylko  
w jednym procesie

function rising-edge<sup>(1)</sup> (signal s: STD\_LOGIC) return BOOLEAN;  
function falling-edge<sup>(0)</sup> (signal s: STD\_LOGIC) return BOOLEAN;

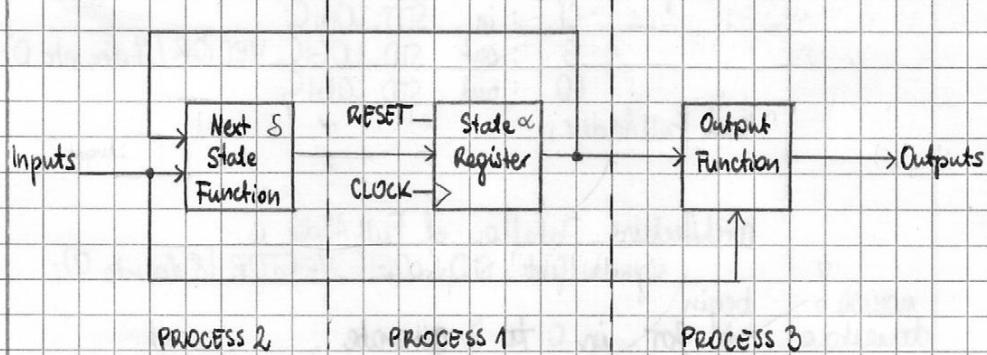
if Clk'Event and Clk = '1' then...



if rising-edge (Clk) then...

### 5.3) Elementy stanów

### 5.4) Maszyny stanów



Maszyny stanów:

```

library IEEE;
use IEEE.std_logic_1164.all;
entity fsm3 is
  port (clk,reset,x1 : IN std_logic;
        output : OUT std_logic);
end entity;
  
```

```

architecture beh of fsm3 is
  type state_type is (s1,s2,s3,s4);
  signal state,next_state:state_type;
begin
  
```

```

process1: process (clk,reset) -- Clock i Reset
begin
  if (reset = '1') then
    state <= s1;
  elsif (clk = '1' and clk'event) then
    state <= next_state;
  end if;
end process process1;
  
```

output <= '1' when state=s1 or state=s2  
else '0';

```

process2: process (state,x1) -- Kolejność stanów
begin
  next_state <= state;
  case state is
    when s1 => if x1='1' then
      next_state <= s2;
    else
      next_state <= s3;
    end if;
    when s2 => next_state <= s4;
    when s3 => next_state <= s4;
    when s4 => next_state <= s1;
  end case;
end process process2;
  
```

```

process3: process (state) -- funkcja wyjścia
begin
  case state is
    when s1 => output <= '1';
    when s2 => output <= '1';
    when s3 => output <= '0';
    when s4 => output <= '0';
  end case;
end process process3;
  
```

end beh;

### Rozdział 3) - c.d. (Instrukcje współbieżne)

#### 3.4) Instrukcje generacyjne:

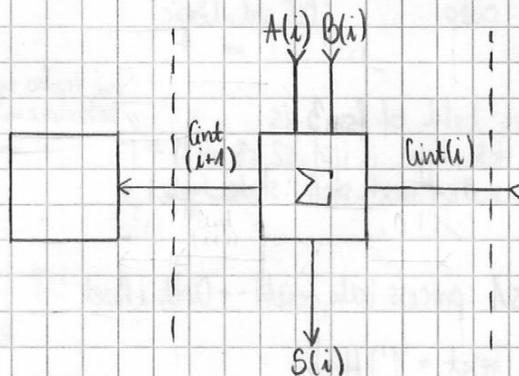
##### a) for... generate

```

entity FullAdder is
    port (A, B : in STD_LOGIC_VECTOR (7 downto 0);
          C1 : in STD_LOGIC;
          S : out STD_LOGIC_VECTOR (7 downto 0);
          CO : out STD_LOGIC);
end FullAdder;

architecture Dataflow of FullAdder is
    signal Cint : STD_LOGIC_VECTOR (8 downto 0);
begin
    process
        for i in 0 to 7 generate
            begin
                S(i) <= A(i) xor B(i) xor Cint(i);
                Cint(i+1) <= (A(i) and B(i)) or
                (A(i) and Cint(i)) or (B(i) and Cint(i));
            end;
        end generate;
        Cint(0) <= C1;
        CO <= Cint(8);
    end Dataflow;

```



##### b) if... generate

```

label: if condition generate      -- label required
      block-declarative items
      begin
          concurrent_statements
      end generate label;           -- optional

```

#### 4) Instrukcje sekwencyjne

##### 4.1) Przypisanie sygnału <=

- niemożliwe przypisanie when... else i with... select

##### 4.2) Przypisanie zmiennej :=

###### 4.3) Instrukcja wait

- wstrzymanie wykonania procesu

- wait for

- wait until ... - czeka do spełnienia warunku

Np. wait until clk = '1' then - czekaj aż do zdarzenia sygnału clk  
if clk = '1' then  
wait until clk = '1';  
end if;

- wait on

wait for 10 ms; -- timeout

wait until clk = '1'; -- warunek logiczny

wait until A > B and (S1 or S2);

wait on sig1, sig2; -- lista warunków

##### 4.4) Instrukcja if

[label:] if condition1 then

statements

elsif condition2 then > optional

statements

else

statements

end if [label];

> optional

## 4.5) Instanzijs case

[label:] case expression is  
when choice 1 =>  
statements  
when choice 2 =>  
statements  
when others =>  
statements  
end case [label];

} - opt.

} - opt. if all choices  
covered

```
architecture DF2 of MUX is
begin
    with Sel select
        Y <= A when "00",
        B when "01",
        C when "10",
        D when "11",
        'X' when others;
    end DF2;
```



```
architecture DF2_Eq of MUX_4 is
begin
    process ( Sel, A, B, C, D )
    begin
        case Sel is
            when "00" => Y <= A;
            when "01" => Y <= B;
            when "10" => Y <= C;
            when "11" => Y <= D;
            when others => Y <= 'X';
        end case;
    end process;
end DF2_Eq;
```

## 4.6) Iteracija

### [label:] loop

statements -- use exit to abort  
end loop [label];

- loop...  
- next (adv. C++: continue)  
- exit (" - : break)

[label:] for variable in range loop  
statements  
end loop [label];

[label:] while condition loop  
statements  
end loop [label];

#### 4.7) Inne instrukcje

- return
- wywołanie procedury
- instrukcja pusta: null ; np:

```
case ... is  
when ... =>  
when others =>  
    null;  
end case
```

- assertion (asercje)
- report (raportowanie błędów) ma potrzeby symulacji

#### 5) VHDL Coding... już omówiliśmy wcześniej

### 6) Model i symulacja

#### Model execution

#### I Elaboration of the design hierarchy

#### II Initialization

#### III Simulation cycles

#### → I Kompilacja

- utworzenie modelu do symulacji
- ustalenie sterowników (drivers) dla wszystkich sygnałów
- sterownik: lista transakcji (par wartości/czas);  
projected output waveform, POW

#### POJĘCIA:

- sygnał aktywny (active): jego sterownik ma transakcje w danym cyklu symulacji  
Atrybuty: S'Active (true/false), S'Quiet (signal) zostaną prypisane
- zdarzenie: zmiana wartości sygnału w danym cyklu  
Atrybuty: S'Event (true/false), S'State (signal) zostaną prypisane i zmienią wartość
- $T_c$  = biegący czas cyklu;  $T_{ij}$  = czas cyklu następującego (najbliższego)
- Przypisanie wsp. sygnału jest uwarunkowane procesem myślnego sygnałem strojnym po jego prawej stronie

\* na koto:

- jeśli jest zdarzenie, to sygnał aktywny
- jeśli sygnał aktywny, to nie musi to być zdarzenie

## I Sygnal: listy POV

S'Active  
S'Event

## II Inicjalizacje

- ustalenie wartości początkowych zadanego jawnie ("`:= '0'`") lub domyślnie `TYPE'left'`
- jednolotne wykonanie wszystkich procesów

$'0'$  wartości  
w typie

Np. w test bench  $x <= '0', '1' \text{ after } 100\text{ms} \dots \Rightarrow$  na liście POV sygnal  $x$  pojawi się odpowiednia transakcja

- $T_N :=$  moment najbliższej transakcji lub uruchomienie procesu

## III Cykl symulacji

$$- T_C := T_N$$

Dla sygnałów

Przypisanie wartości

- (a) Przypisanie wartości sygnałów aktywnych w danym cyklu

lubiące wartości

- (b) Wykarmianie procesów (tej przyp. współ. "`<=`") wartościami zadanymi, które

są konsekwencją (a)

- $T_N :=$  moment najbliższej transakcji lub uruchomienie procesu

$\Rightarrow$  Jeśli  $T_N = T_C$  to następny cykl symulacji jest tzw. "cyklem delta"

$\Rightarrow$  Wykonanie przypisania do sygnału = unieszczenie na jego liście POV odpowiedniej transakcji

```
entity Ex1 is
  port (
    A,B : in STD_LOGIC;
    Y   : out );
end entity;
```

```

architecture Dflow of Ex1 is
begin
    signal S : STD_LOGIC;
    S <= A or B;
    Y <= B xor S;
end architecture;

```

Cykl:	A	B	S	Y	Opis:
(...)	'0'	'0'	'0'	'0'	
10ms	'1'	'0'	'0'	'0'	(a) A
10ms + Δ	'1'	'0'	'1'	'0'	(a) S
10ms + 2Δ	'1'	'0'	'1'	'1'	(a) Y

```

process (A,B,S)
begin
    S <= A or B;
    Y <= B xor S;
end process,

```

Zmiany wortacj  
symetrie  
nestepluja  
dopiero po  
zakonczeniu  
procesu

#### 4) SPRAWCOWE REALIZACJE ALGORYTMÓW str. 40/41

```
process (A1c)
begin
    if rising_edge(I) and
        State=sReady then
        regD(←D1);
    end if;
end process;
```

jeżeli we we  
rising to down then,

Next State c = State;  
~default

## II PROGRAMOWALNE UKŁADY LOGICZNE

### Programmable Logic Devices - PLD

data 40:

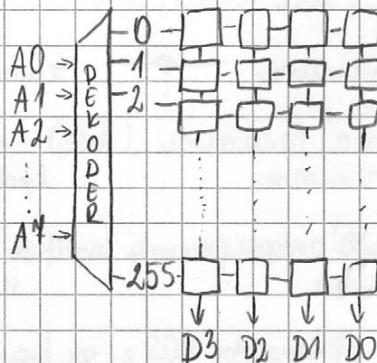
- pamięć PROM - Programmable ROM

następnie:

- Erasable PROM (EPROM) - kasowane promieniem UV

- Electrically EPROM (E<sup>2</sup>PROM) - po kasowaniu możliwe ponowne napisanie

PROM 256x4



### CYFROWE UKŁADY SCALONE

#### UKŁ. STANDARDOWE

- modern TTL, ECL
- μProc
- pamięci
- 

#### ASIC

Application Specific Int. Circ.

#### MASK PROGRAMMABLE

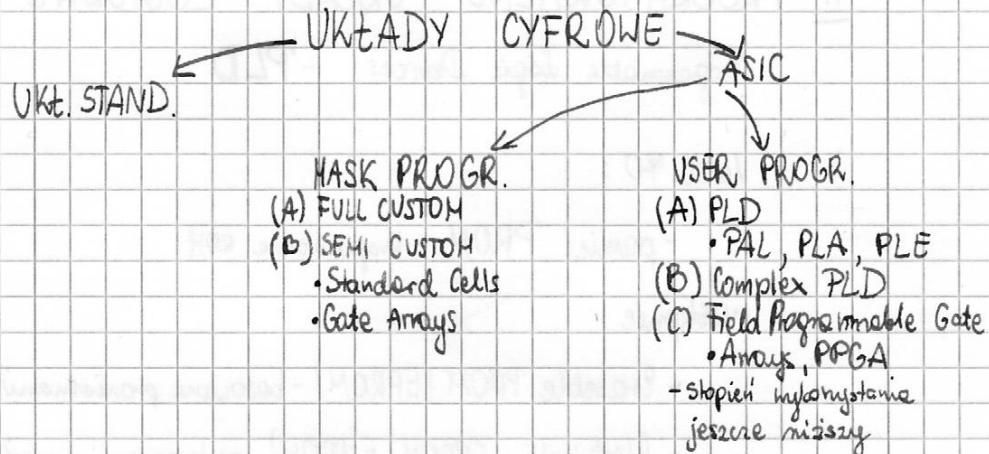
- (A) FULL CUSTOM
  - pełny cykl projektowy
  - długie, nieochotne projekty
- (B) SEMI CUSTOM

#### USER PROGRAMMABLE

- mo  
następny  
wykładek

- Gate Arrays
- gotowe matryce blok. log. w kremie
- wykonywanie projektu metodą montażową (płytka)
- stopień wykorzystania matrycy < 100%

- Standard Cells
- klocki gotowych komórek
- krótki cykl projektowy
- mega cells / cores



### 1) Modyfikowane programowalne PLD

Transistor  $\rightarrow$  Tr. MOS : OFF/ON

$G$   
Grenica stanów  
 $S$   
kierunek kroku  
 $D$

; tu: otwarte, gdy  $M_G = "0 \text{ log}"$

- Elementy : S, WE, linie sterowne, S, WY

- Punkty programowalne , technologie:

- Inne programowane a) bezpieczeniki (fuse) : normalnie : ON, tylko układy bipolarnie, niskie scalenie

- + trwałe, odporne, niezawodne  
- Inne programowane b) antybezpieczeniki (antifuse) : normalnie : OFF, dwoje lepsze scalenie, technologie CMOS

deformacjai  
otwory

c) transistor MOS z tzw. bramką swobodną : kasowanie prom WE

d) transistor MOS z tzw. bramką swobodną : kasowanie elektrycznie EEROM

e) transistor MOS sterowany 1bitem komórką pamięci S-RAM

Programowane jednobrotne vs. Kasowane vs. kasowane elektrycznie  
(a,b) (c-e) (d-e)

Inne (e) vs.剩能 (a-d)

$\Rightarrow \text{xc9500 XC : d , Spartan 3E : e}$

$\Rightarrow \text{In system Programming (ISP) : d + e}$   
 kabel programujący - interfejs JTAG

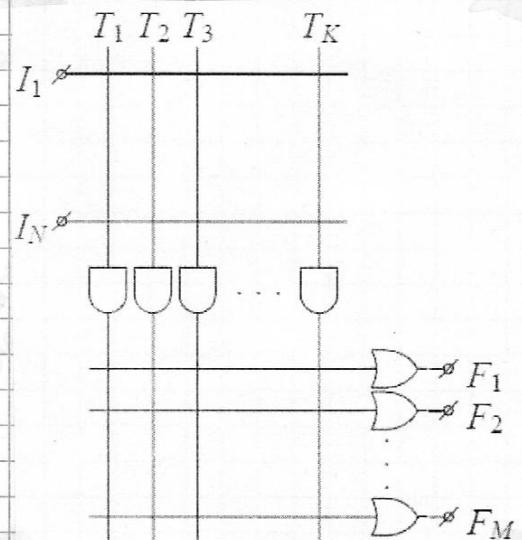
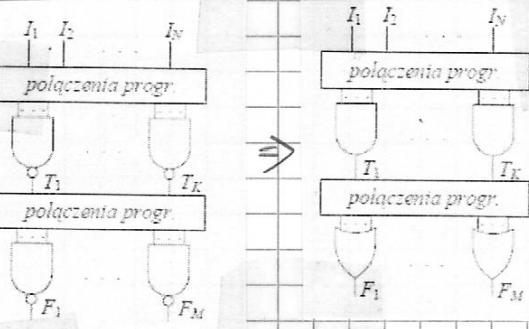
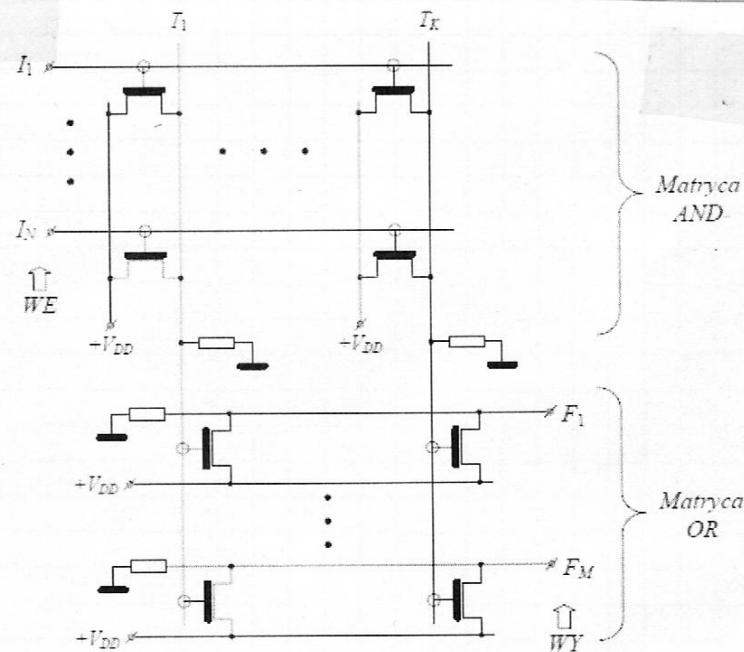
Analiza pracy:

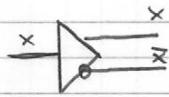
$$T_K = \emptyset \Leftrightarrow \text{dotarcie } WE \quad I = 1 : \quad T_K = \text{NAND}(\alpha_{K1} + T_1, \alpha_{K2} + T_2, \dots, \alpha_{Kn} + T_n)$$

$$\alpha_{Ki} = \begin{cases} 1 & - \text{we. niedotarcie} \\ \emptyset & - \text{we. jest dotarcie} \end{cases}$$

Analogicznie

$$F_m = \text{NAND}(\beta_{m1} + T_1, \beta_{m2} + T_2, \dots, \beta_{mk} + T_n)$$



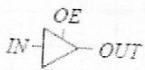
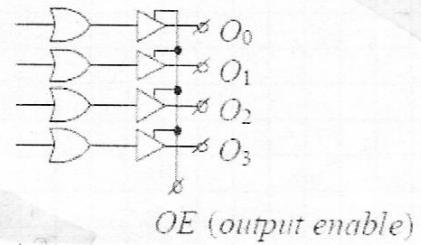
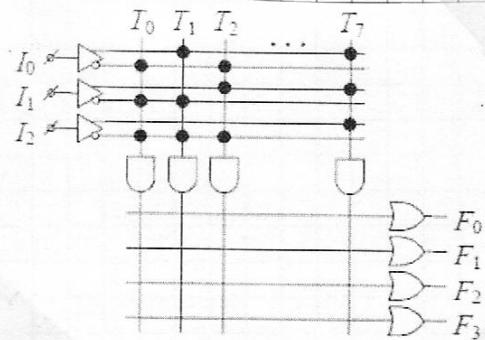


PAL Programmable Array Logic	PLE Program Logic Element	PLA Program Logic Array
Matrype AND	Prog.	Const.
Matrype OR	Const.	Prog.

### 1.1) Układy PLE

8x4:

⇒ pamięci PROM



$$OE = 1 \Rightarrow OUT = IN \quad (0/1)$$

$$OE = 0 \Rightarrow OUT = Z \quad (\text{HighZ})$$

### 2) Układy PAL

- suma programowalnych liczb binarnych: Sum Of Products SOP
- technologie: PROM, EPROM, EEPROM (np. PALCE, PAL CMOS Enable)
- podstawowe klasyfikacje
  - u. kombinacyjne - serie L/H
  - u. rejestrujące - R
  - u. z makrokomórkowym - seria V

## 2.1) Układy PAL kombinacyjne

### • Rodzina PAL16L8

- obudowa 20 pinowa
- 10 WE zewnętrznych
- 8 WY, 6 z spłaszczeniami zbrojnymi
- kieride WY: 4 terminów OR + 1 termin jako OE
- matryce  $32 \times 64$  (terminy)

## 2.2) Układy PAL rejestrowe

### • Np. PAL16R8

Podobne: PAL16R6 16R4

2WY komb.

4WY komb.

- obudowa j.w. matrycjiw
- 2 dedykowane WE: CLK, OE
- 8 WE zewnętrznych
- kieride bramki OR = 8 terminów
- 8+8 WE do matrycy programowej

6 rejestrów 4 rejestrów

Oznaczenie PAL, np.:

PAL 16 L 8

ilosc' WE matrycji  
ilosc' WY/WY rejestrów

Rodzina:

- L/H = akt. komb. aktywne "0" / "1"
- C = akt. komb. z WY komplementarnymi
- P = akt. komb. z pełniącego programowanie,
- R = akt. komb. z przesuniętym D (rejestrowe)
- V = akt. a tzw. mikrokontrolerami programowalnymi



## 2.3) PAL z mikrokontrolerem

### • Np. PALCE22 V10

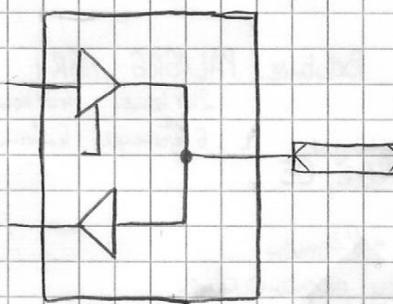
- matryce  $44 \times 132$
- 12 WE zew. + 10 spłaszczeń zbrojnych
- mikrokontroler 8 terminów / br OR ( $8 \div 16$ )
- dwa dodatkowe terminy wejściowe AR, SP
- po dwa punkty progr. S0, S1 w kieridze mikrokontrolerze

WYKŁAD 4 05.01.2012

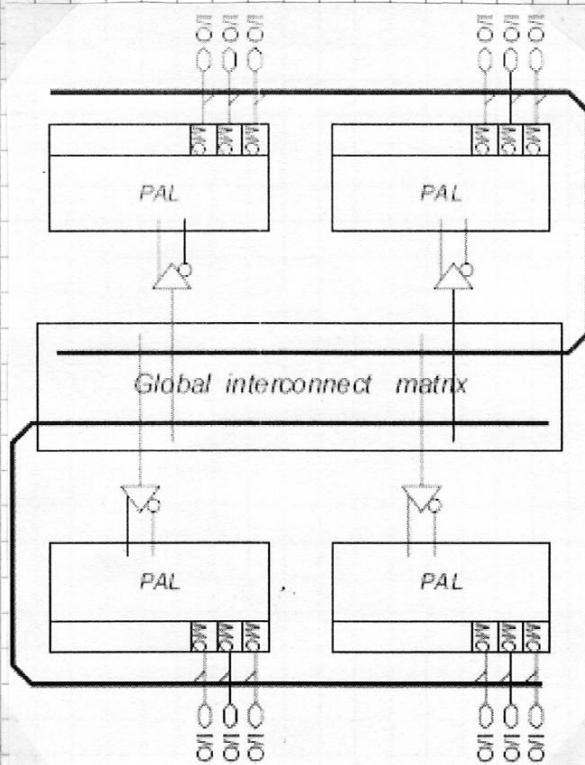
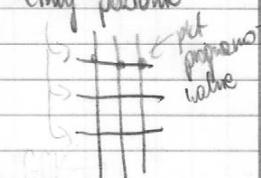
3) Układy CPLD modeliny XC9500XL  
Complex PLD

- 18 makrokomórek na kierdy modułu
- piny wejściowe - wyjściowe

10 Block

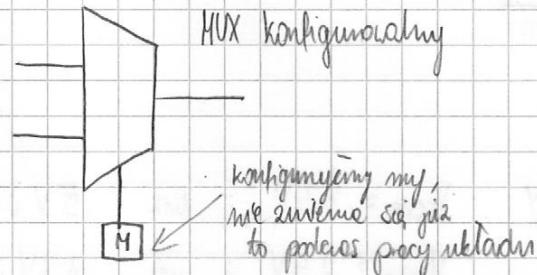


termy pośrednie

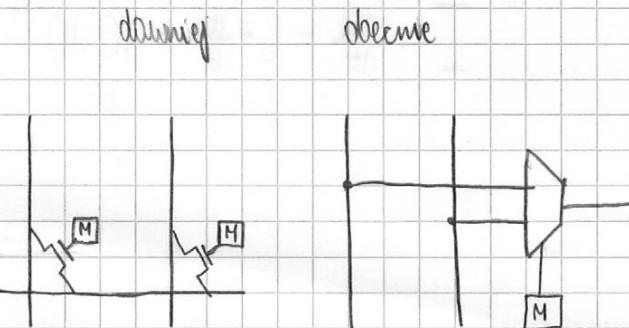


### 3.1) Architektura

- GCK = Global Clock ( $\times 3$ )
- GSR = Global SET/RESET
- GTS = Global Three State ( $\times 2 / \times 4$ ) Output Enable buf. 3-stanowy
- 54 WE do bloków funkcyjnych  
18 mukukombinacji / blok } = PALCE 54V18
- 5 terminów / MC, 90 / blok funkcyjny
- matryce  $108 \times 90$  (rozmiar)
- przekształtnik konfiguracyjny typ D/T



- Alokator terminów
- Matryca globalna



### 3.2) Bloki WE/WY

- osobne zasilanie  $V_{CCINT}$  vs.  $V_{CCIO}$

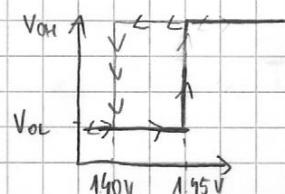
$$3,3V \quad 3,3V \text{ lub } 2,5V$$

- Bufor WE - zasilanie zewnętrzne 3,3V / prog. przeł.  $= 1,4V$

- wytrzymuje  $V_{in}$  do +5V

zdolność  $\approx$  TTL (+5V), CMOS:  $5V \div 2,5V$

- histerese charakterystyki WE



żeby nie było drgania na wyjściu  
filtruje fluktuacje

- Bufor WY  $V_{CCIO} = 3,3V$  : m.in. 5V i 3,3V

$$V_{CCIO} = 2,5V : \text{m.in. } 3,3V \text{ i } 2,5V$$

- ograniczenie szybkości narastania zbroty (Slow rate)

- przetaczanie buforów  $\Rightarrow$  pojemność obciążająca,  $C_L = 10\text{ pF}$

*Zmniejszać  
zbroby nie było problemu  
i problem*  $\Rightarrow$  szybkość narastania 1V/s

$$I = \frac{dQ}{dt} = C \cdot \frac{du}{dt} = 10 \cdot 10^{-12} \cdot \frac{1}{10^{-9}} = 10mA$$

do przetaczania  
w koridurze  
pomiędzy siebie 10mA  
(impulsy)