

VHDL, DE2, Quartus

Wprowadzenie do laboratorium

Rafał Walkowiak

IIIn PP

Wer 1.1

2.10.2012

VHDL

- **VHDL** (*ang. Very High Speed Integrated Circuits **H**ardware **D**escription **L**anguage*) jest bardzo popularnym językiem opisu sprzętu używanym w komputerowym projektowaniu układów cyfrowych typu FPGA i ASIC.
- Wersja VHDL z roku 1993 (IEEE Std 1076-1993) jest obecnie najpopularniejsza (większość narzędzi jest zgodna z tym standardem).

Charakterystyka VHDL

- Zawiera użyteczne konstrukcje semantyczne umożliwiające związłą specyfikację złożonych układów cyfrowych.
- Projekt może posiadać wielopoziomową hierarchię .
- Możliwe jest korzystanie z biblioteki gotowych elementów.
- Możliwe jest tworzenie podukładów wykorzystywanych jako tzw. komponenty w innych projektach.

Struktura projektu

- Definiowanie **jednostek projektowych**.
- Określenie **jednostki projektowej najwyższego poziomu**.
- Pozostałe jednostki projektowe możliwe do wykorzystania jako **komponenety** jednostki najwyższego poziomu.
- Wykorzystanie jednostek projektowych z **dołączonych bibliotek** jako komponentów innych jednostek projektowych.

Jednostka projektowa

Deklaracja –

wej-wy, parametry

```
library ieee; -- deklaracja biblioteki
-- import elementów pakietu biblioteki
use ieee.std_logic_1164.all;
entity sumator is
port(
a:      in std_logic;
b:      in std_logic;
cin:    in std_logic;
s:      out std_logic;
cout:   out std_logic);
end sumator ;
```

Opis działania

```
architecture pierwsza of sumator is
signal s1:std_logic;

begin
s1<= (a and not b) or (b and not a);
s <= (cin and not s1) or (not cin and s1);
cout<= (not s1 and b) or (s1 and cin);
end pierwsza;
```

Deklaracja jednostki projektowej

```
entity sumator is
port(
a:    in std_logic;
b:    in std_logic;
cin:  in std_logic;
s:    out std_logic;
cout: out std_logic);
end sumator ;
```

Tryby portów:

- **in** - wejściowy
- **out** - wyjściowy
- **inout** - dwukierunkowy
- **buffer** - wyjściowy z
możliwością odczytu

Opis działania jednostki

architecture nazwa_arch **of** nazwa_jednostki **is**

Definicje i deklaracje:
(typów, podtypów, stałych, sygnałów,
komponentów, konfiguracji)

begin

- instrukcje przypisania wartości do sygnałów
- procesy
- komponenty

end nazwa_arch;

Elementy strukturalne języka VHDL

»Słowa kluczowe

»Identyfikatory

»Obiekty danych

»Operatory

»Atrybuty

»Instrukcje

Identyfikatory

- Mogą się składać z liter, cyfr i znaków podkreślenia
- Wielkość liter nie ma znaczenia
- Nazwa musi się rozpoczynać od litery
- Nazwa nie powinna być dłuższa niż 16 znaków

Obiekty danych

- Obiekty danych służą do przechowywania wartości.
- Wyróżnia się trzy klasy obiektów:
 - sygnał – *signal*
 - zmienne – *variable*
 - stałe – *constant*
- Wartości zmiennych są przypisywane natychmiast. Wartości sygnałów są przypisywane zgodnie z regułami – np. z pewnym opóźnieniem.
- Odpowiednikiem sprzętowym sygnału jest ścieżka w układzie scalonym.
- Zmienna nie ma odpowiednika sprzętowego i służy do obliczeń na wyższym poziomie abstrakcji.

Obiekty danych (2)

Sygnały:

```
signal reset: std_logic;
```

```
signal dane: std_logic_vector(7 downto 0);
```

Stałe:

```
constant zero:
```

```
std_logic_vector(7 downto 0):="0000_0000";
```

Zmienne:

```
variable zmienna: bit;
```

Typy danych

- BIT, BIT_VECTOR,
- **STD LOGIC, STD LOGIC VECTOR,**
- STD_ULOGIC, STD_ULOGIC_VECTOR,
- INTEGER, BOOLEAN,
- TYP WYLICZENIOWY

STD_LOGIC

- STD_LOGIC, STD_LOGIC_VECTOR – *standardowy typ, rozstrzygalny (możliwych wiele nośników – przypisać równoległych), funkcja rozstrzygająca*
- Zastosowanie wymaga deklaracji:
 - **LIBRARY** IEEE;
 - **use** IEEE.Std_Logic_1164.all;

STD_LOGIC

- Wartości
 - ‘U’ – niezainicjowany,
 - ‘X’, ‘0’, ‘1’ – wymuszenie: stan nieokreślony, 0, 1
 - ‘Z’ – stan wysokiej impedancji
 - ‘W’, ‘L’, ‘H’ – słaby: stan nieokreślony, 0, 1
 - ‘-’ – stan dowolny (**wielkie litery obowiązkowo**)
- STD_LOGIC_VECTOR jest tablicą obiektów STD_LOGIC
- Operatory +, - dostępne dla tego typu (przeciążenie) po zastosowaniu dodatkowo:

use IEEE.std_logic_unsigned.all;

Typ *std_ulogic* – charakterystyka jak wyżej, lecz typ nierozstrzygalny

Inne typy danych

Typ **bit** - typ dwuwartościowy 0, 1;

Typ **integer** – np. - **variable** a: integer **range** –10 **to** 10;

Typ wyliczeniowy (definiowany przez użytkownika):

type wyliczeniowy **is** (zielony, żółty, czerwony);

Przykład użycia trybu wyliczeniowego:

```
signal stan: wyliczeniowy;
```

```
Case stan is
```

```
  when zielony =>
```

```
    stan <= żółty;
```

```
  when żółty =>
```

```
    stan <= czerwony;
```

```
end case;
```

Sygnały

- Sygnał może być zdefiniowany w bloku deklaracji jednostki oraz w części deklaracyjnej architektury.
- Przypisanie wartości do sygnału nie jest natychmiastowe lecz „harmonogramowane”
- Z sygnałem skojarzone są: typ i wartość
- Składnia deklaracji:

signal nazwa_sygnału:

typ_sygnału[:=wartość_początkowa];

- Przykłady: **signal** s1, s2 : bit;
signal liczba : integer := 7;

Stałe

- Stałe to obiekty danych, których wartości nie zmieniają się w trakcie symulacji.
- Stałe mogą być zadeklarowane w bloku deklaracji jednostki projektowej, architektury, pakiecie, procesie, funkcji i procedurze.
- Składnia: **constant nazwa : typ := wartość;**
- Przykłady użycia:

constant zero:

STD_LOGIC_VECTOR(3 DOWNT0 0):="0000";

Typ tablicowy

- Zakres typu tablicowego zmiennej lub stałej jest definiowany pod w ramach deklaracji typu lub zmiennej za pomocą słów kluczowych: „to” „downto”
- Przykłady:

```
TYPE Byte IS ARRAY (7 DOWNT0 0) OF STD_LOGIC;
```

```
SIGNAL X : Byte;
```

```
signal m1,m2: std_logic_vector(1 to 3);
```

```
m1(1 to 2) <= "00";
```

```
m2(3 downto 1) <= m1(3 downto 1);
```

Atrybuty sygnałów

- Dostarczają dodatkowych informacji o obiektach (np. sygnałach, zmiennych, typach lub komponentach).
- Składnia: **obiekt'atrybut[(parametr)];**
- Predefiniowane atrybuty:
 - 'left; 'right; 'high; 'low; - lewy, prawy, górna, dolna
wartość danego typu
 - 'length;'range; - liczba bitów, zakres
 - 'event; 'stable; - przyjmuje wartość true/false –
informacja, że sygnał zmienił lub nie wartość

Atrybuty sygnałów -przykłady

type count **is** integer **range** 127 **downto** 0;

type states **is** (idle, read, write);

count'left to 127

count'high to 127

states'right to write

states'high to write

If Clock'event **and** Clock = '1' **then** Q <= D;

Analogiczne do 'event informacje dostarczają funkcje określone
dla sygnałów - rising_edge(nazwa_sygnalu),
falling_edge(nazwa_sygnalu)

Operatory

- Specyfikacja wg malejącego priorytetu

Klasa	operator	Typ danych
negacja	NOT	INTEGER, BIT, STD_LOGIC
MNOŻENIA	<i>*, /, mod, rem</i>	INTEGER
ZNAKU	<i>+ -</i>	INTEGER
SUMOWANIA	<i>+ -</i>	INTEGER, BIT, STD_LOGIC
PRZESUWANIA	<i>sll, srl, sla, sra, rol, ror</i>	STD_LOGIC, BIT
RELACYJNE	<i>=, /=, <, <=, >, >=</i>	Argumenty – tego samego typu; wynik – BOOLEAN
LOGICZNE konieczność określenia kolejności realizacji	<i>and, or, nand, nor, xor, xnor</i>	BIT, BOOLEAN, STD_LOGIC

Instrukcje

Współbieżne

- Zachowanie układu jest niezależne od kolejności instrukcji przyporządkowania sygnałów.
- Stosowane w specyfikacji typu „przepływ danych” (ang. dataflow description).

Sekwencyjne

- Porządek zapisu instrukcji sekwencyjnych zmienia działanie układu.
- Instrukcje sekwencyjne są stosowane w specyfikacji behawioralnej (ang. behavioral description). Przede wszystkim w tzw. procesach (ang. process)

Instrukcja przypisania

- Instrukcja przypisania:
Nazwa_sygnalu <= **wyrażenie**;

- Przykład:

```
architecture a of test is
```

```
begin
```

```
error <= we(2) and (we(1) or we(0));
```

```
end a;
```

Instrukcje współbieżne: when-else, with-select-when

```
y <= j when a=b else  
    k when c=d else  
    m when others;
```

.....

```
with w select  
    x <=  a when v1 | v2,      -- gdy a=v1 lub v2  
          b when 1 to 3,  
          z when others;
```


Realizacja instrukcji współbieżnych

- Instrukcja współbieżnego przypisania wartości do sygnału ma **ukrytą listę czułości**, która zawiera wszystkie sygnały znajdujące się po prawej stronie symbolu \leq .
- Instrukcja współbieżnego przypisania jest wykonywana zawsze, gdy nastąpi **zmiana dowolnego sygnału** występującego na liście czułości.
- Dla typów rozstrzygalnych współbieżne przypisania wartości do sygnałów powodują utworzenie kolejnych nośników dla uaktualnianych sygnałów. W przypadku uaktualniania sygnału przez wiele instrukcji dla utworzonych nośników sygnału stosowana jest **funkcja rozstrzygająca** określająca stan sygnału. Wykorzystanie w przypadku magistral.
- W przypadku wielokrotnego przypisania w **instrukcjach sekwencyjnych** stosowany jest jeden nośnik sygnału.

Instrukcje sekwencyjne

- Do wykorzystania w *procesach*
- Przykłady:
 - If
 - Case
 - For

Proces (1)

Składnia:

```
[etykieta:] process [ ( lista_czułości ) ] [ is ]
```

```
--definicje i deklaracje
```

```
begin
```

```
--instrukcje_sekwencyjne;
```

```
end process [ etykieta ] ;
```

- Lista czułości - jest to lista sygnałów, których zmiana powoduje aktywację procesu.
- Przypisanie wartości sygnałów bazuje na bieżącej wartości sygnałów znajdujących się po prawej stronie `<=`. Modyfikowane sygnały zostaną uaktualnione dopiero po wykonaniu wszystkich instrukcji. Sygnały zostają uaktualnione wartością ostatniego przypisania.

Proces (2)

- Konstrukcja wykonywana równolegle z innymi procesami
- Występuje wewnątrz architektury
- Instrukcje wewnętrzne wykonywane są sekwencyjnie
- Procesy nie mogą być zagnieżdżane
- W części deklaracyjnej można definiować: typy, podtypy, stałe, atrybuty i zmienne
- Nie można deklarować sygnałów

Instrukcja warunkowa if – then - else

- Instrukcja sekwencyjna
- Część **elsif** może wystąpić dowolną ilość razy
- Część **else** nie musi wystąpić
- Może być zagnieżdżana
- Składnia:

```
if warunek then  
    sekwencja_instrukcji1  
    {elsif warunek then  
        sekwencja_instrukcji2 }  
    [else sekwencja_instrukcji3]  
end if ;
```

Instrukcja wyboru - case

- Instrukcja sekwencyjna
- Można określać kilka wartości lub zakres
- Wartości nie mogą się powtarzać
- Składnia: **case** zmienna **is**
when wybór1 => sekwencja1_instrukcji ;
when wybór2 => sekwencja2_instrukcji ;
[**when** others => sekwencja3_instrukcji ;]
end case;

Pętle for i while

Instrukcje sekwencyjne

```
lab1: for i in 3 downto 0 loop  
    if reset = '1' then  
        dane_wy(i) <= '0';  
    end if;  
End loop lab1;
```

```
i:=3;  
lab2: while ( i > 0 ) loop  
    if reset = '1' then  
        dane_wy(i) <= '0';  
    end if;  
    i:=i-1;  
End loop lab2;
```

--wyjście z pętli

```
exit etykieta2 when i > 14;  
if i>14 then exit etykieta2;
```

Komponent (1)

- Jednostka projektowa raz zdefiniowana i przeznaczona do wielokrotnego wykorzystania w projektach.
- Komponenty można tworzyć w ramach projektu w pliku głównym lub plikach dołączonych do projektu.
- W celu korzystania z komponentów konieczne są:
 - Definicja komponentu
 - Deklaracja komponentu w bloku deklaracji architektury jednostki projektowej wykorzystującej komponent
 - Konkretyzacja – specyfikacja wykorzystania komponentu
- Projekty mogą stanowić pakiety przechowywane w bibliotekach zawierające komponenty do wykorzystania w innych projektach.

Komponent (2)

Przykład wykorzystania komponentu:

```
architecture a of uklad1 is
```

```
component mux3_5to1
```

```
port( s,p1,p2,p3,p4,p5,p6,p7,p8: in std_logic_vector(2 downto 0);
```

```
m:out std_logic_vector(2 downto 0));
```

```
end component;
```

```
-- pozostałe deklaracje
```

```
begin
```

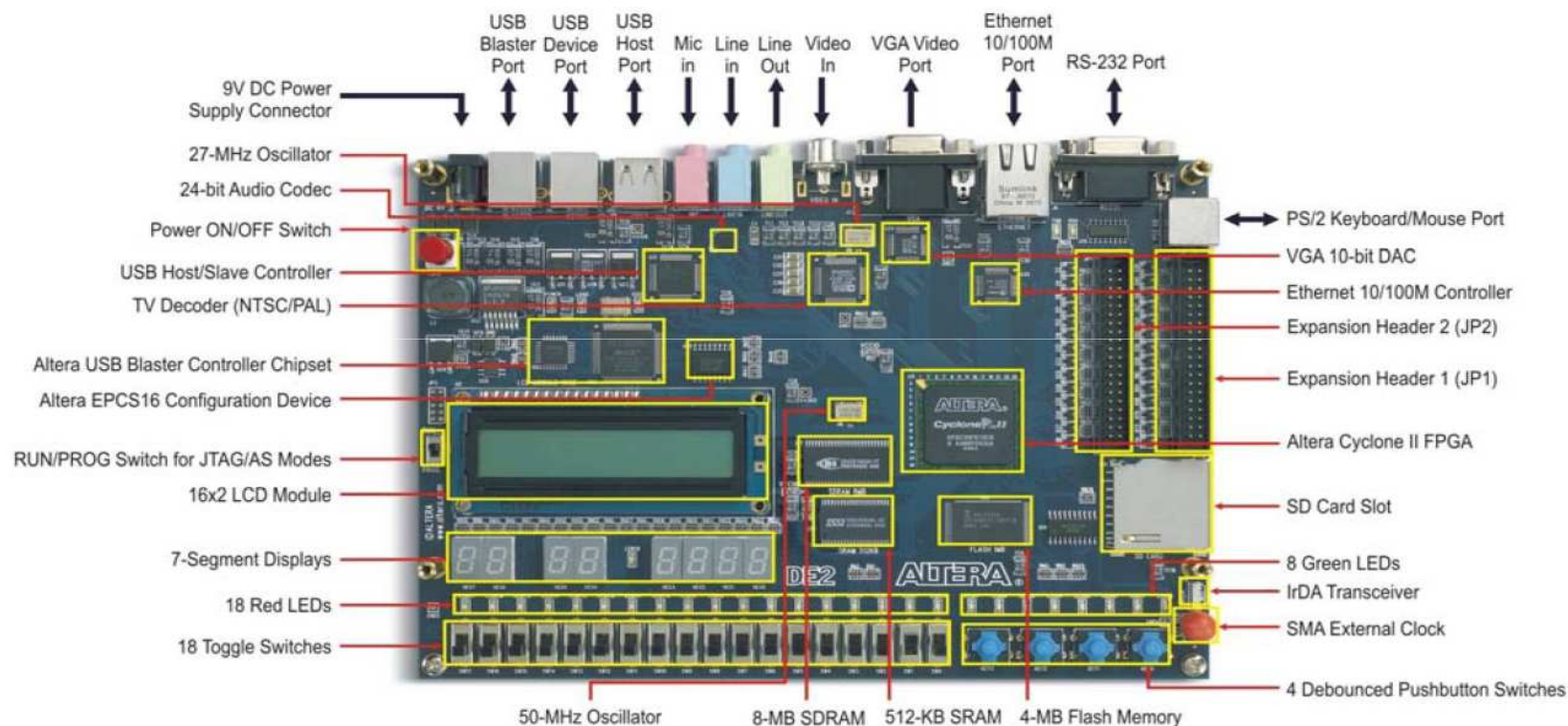
```
m1:mux3_5to1 port map (sw(17 downto 15),sw(14 downto 12),sw(14 downto  
12),sw(14 downto 12),sw(11 downto 9),sw(8 downto 6),
```

```
sw(8 downto 6),sw(5 downto 3),sw(2 downto 0),m1);
```

```
--reszta definicji układu1
```

```
end
```

DE2 Development and Education Board



Korzystamy również z DE2-70 – podobna struktura, ~2x większy FPGA

Elementy składowe DE2

- **Altera Cyclone® II 2C35 FPGA device**
- Altera Serial Configuration device - EPCS16
- USB Blaster (on board) for programming and user API control; both JTAG and Active Serial
- (AS) programming modes are supported
- 512-Kbyte SRAM
- 8-Mbyte SDRAM
- 4-Mbyte Flash memory (1 Mbyte on some boards)
- SD Card socket
- 4 pushbutton switches
- 18 toggle switches
- 18 red user LEDs
- 9 green user LEDs
- 50-MHz oscillator and 27-MHz oscillator for clock sources
- 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks
- VGA DAC (10-bit high-speed triple DACs) with VGA-out connector
- TV Decoder (NTSC/PAL) and TV-in connector
- 10/100 Ethernet Controller with a connector
- USB Host/Slave Controller with USB type A and type B connectors
- RS-232 transceiver and 9-pin connector
- PS/2 mouse/keyboard connector
- IrDA transceiver
- Two 40-pin Expansion Headers with diode protection

Schemat blokowy DE2

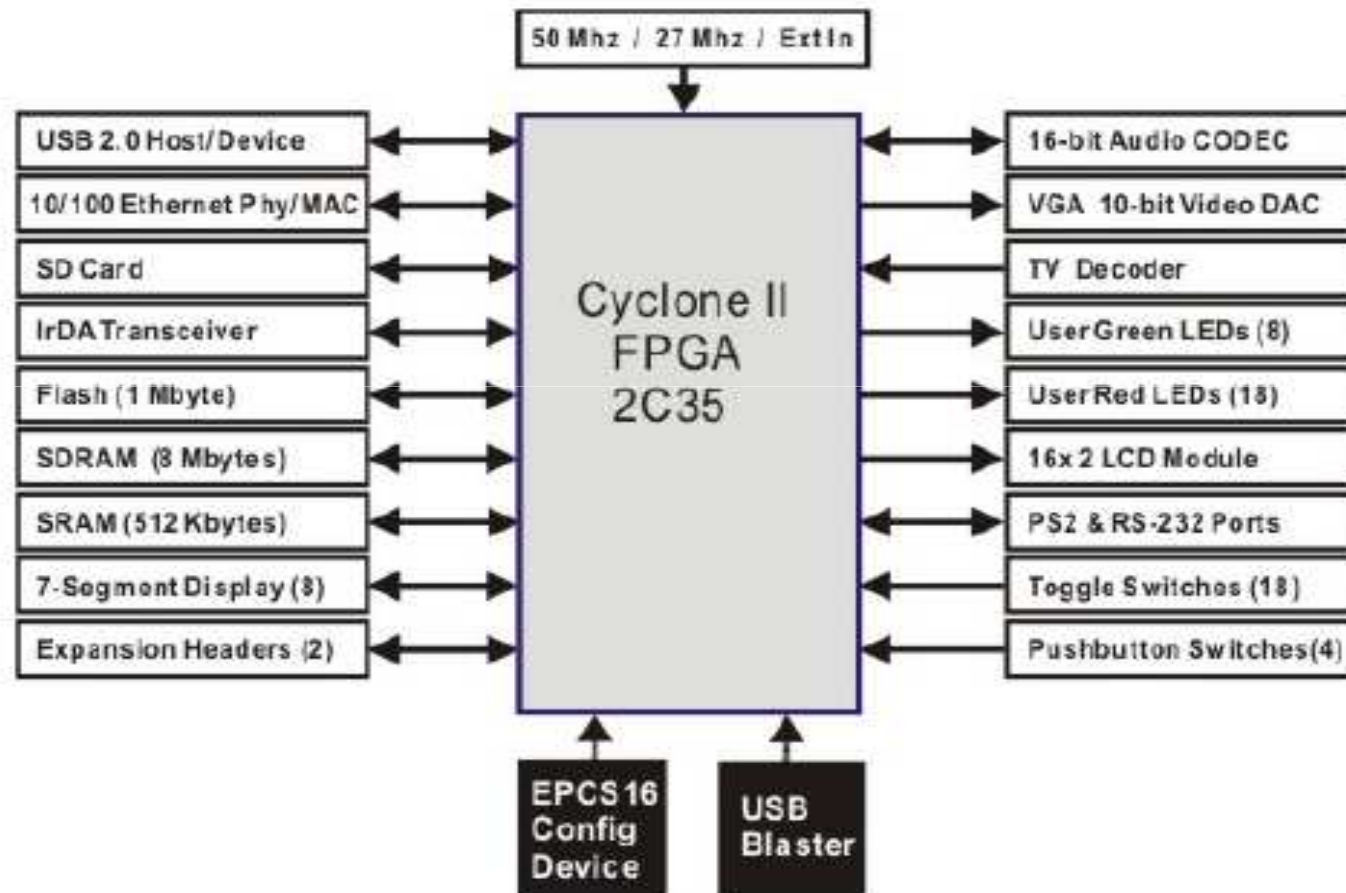


Figure 2.2. Block diagram of the DE2 board.

Parametry Cyclone II 2C35 FPGA

- 33,216 elementów logicznych/ 70 k LE dla 2C70
- 105 bloków pamięci RAM M4K
- 483,840 liczba bitów RAM
- 35 wbudowanych układów mnożących
- 4 pętle sprzężenia fazowego
- 475 wyprowadzeń użytkownika typu I/O
- obudowa z 672 wyprowadzeniami

Altera Quartus

- Narzędzie projektowe dla FPGA i CPLD
- Umożliwia:
 - wprowadzenie projektu,
 - syntezę logiczną i symulację funkcjonalną,
 - przydział do układów logicznych i ich łącznie,
 - symulację czasową,
 - analizę czasową,
 - zarządzanie użytkowaniem mocy i
 - programowanie układu FPGA

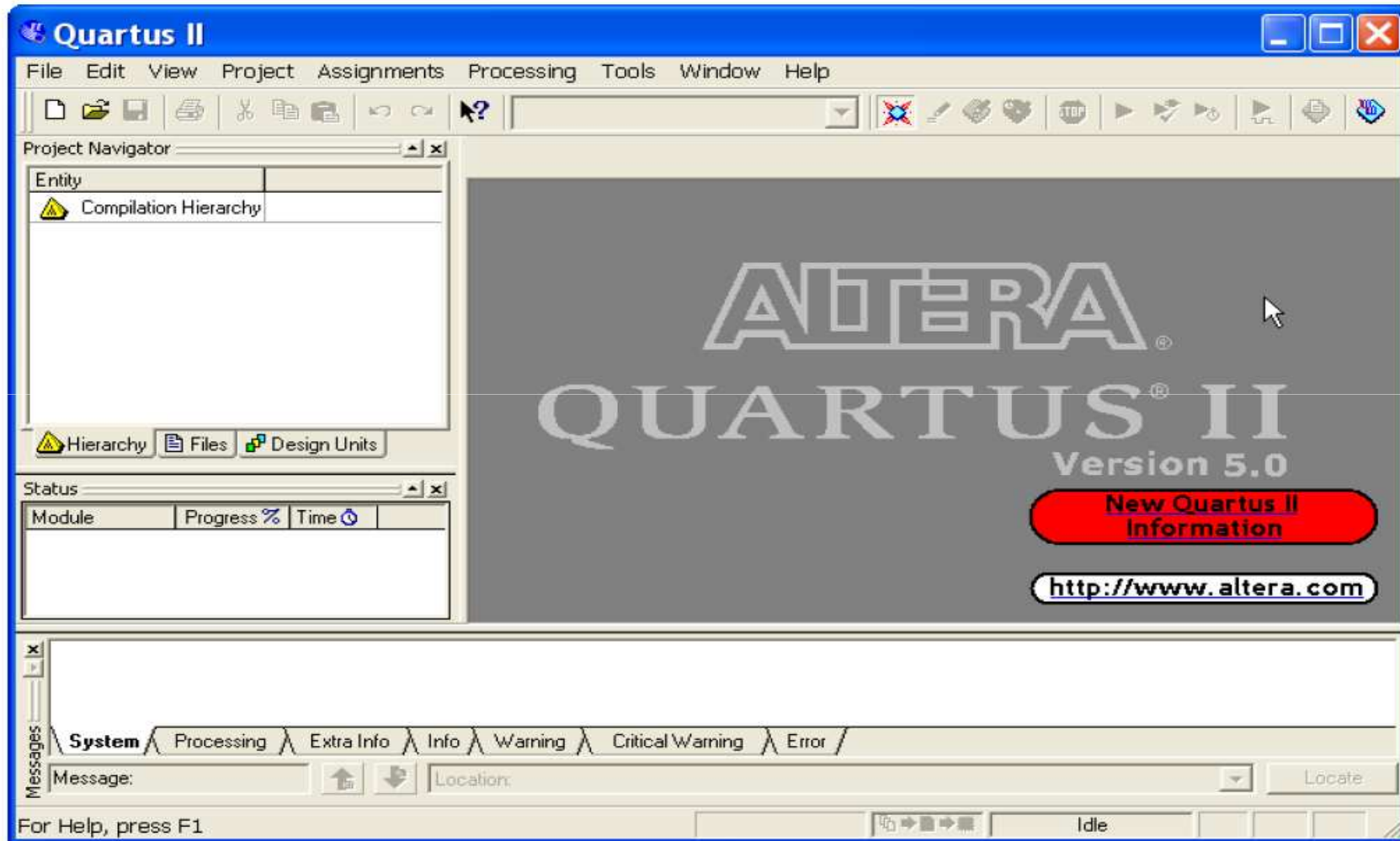
Altera Quartus

- **Używamy QUARTUS wersji 11.0 gdyż współpracuje z symulatorem Qsim z Altera University Program**
- **Możliwość darmowego pobrania programów (cele edukacyjne) ze strony altera.com:**
 - **11.0_quartus_free_windows.exe 2,70 GB**
 - **altera_upds_setup.exe 31,3 MB**
 - **symulator udaje się zainstalować w Windows XP (czerwiec 2011)**

Quartus interfejs użytkownika

- Okno główne: pliki, raporty , inne okna
- Projekt nawigator: hierarchia, pliki projektu, skróty komend
- Okno statusu: status przetwarzania zadań
- Okno komunikatów: informacje, błędy, ostrzeżenia
- Dynamicznie zmieniające się menu

Quartus okno główne



Projekt w Quartus

Praca zorganizowana w ramach projektów.

Projekt składa się z:

- plików projektowych i bibliotek,
- informacji o parametrach i ograniczeniach,
- hierarchii projektu i informacjach o wydaniach.

Etapy procesu projektowania w PLD

1. Wprowadzanie projektu

- Schemat – edytor blokowy
- Język definiowania sprzętu HDL
- Formaty reprezentacji projektu w systemach EDA

2. Symulacja na poziomie przestań międzyrejestrowych RTL (czy projekt poprawny logicznie?)

- weryfikacja logiczna modelu bez opóźnień

3. Synteza (przydział elementów logicznych modelu do standardowych elementów)

- minimalizacja modelu logicznego
- optymalizacja modelu logicznego
- przydział elementów logicznych do podstawowych elementów składowych układu cyfrowego

Etapy procesu projektowania dla PLD

4. **Umieszczanie i łączenie** (moduły ang. fitter router)
 - przydział elementów logicznych do jednostek logicznych sprzętu
 - uwzględnienie wprowadzonych ograniczeń na realizację projektu
 - realizacja połączeń
5. **Analiza czasowa** (czy spełnione są ograniczenia czasowe?):
 1. Badanie, optymalizacja i prezentacja efektywności czasowej
 2. sprawdzanie i informowanie o przekroczeniach ograniczeń czasowych
6. **Symulacja czasowa:**
 - Sprawdza poprawność czasową logiki
 - Korzysta z listy połączeń uwzględniających czas
 - Wymaga wektorów wymuszeń na wejściach układu
7. **Programowanie i badanie sprzętu**

Podstawowe działania (1)

TWORZENIE NOWEGO PROJEKTU

- File> New Project
 - katalog,
 - nazwa projektu,
 - nazwa elementu najwyższego poziomu w hierarchii projektu
 - dołączenie plików projektu (np. plików komponentów biblioteki)
 - określenie układu docelowego - EP2C35F672C6 dla DE2, EP2C70F896C6 dla DE2-70

Tworzenie nowego projektu za pomocą kreatora (katalog, nazwa, jednostka główna)

New Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

D:\projekty_quartus\RS

What is the name of this project?

RS

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

RS

Use Existing Project Settings ...

< Back Next > Finish Cancel

Tworzenie nowego projektu za pomocą kreatora projektu

New Project Wizard: Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family

Family: Cyclone II

Devices: All

Target device

☐ Auto device selected by the Filter

☒ Specific device selected in 'Available devices' list

Show in 'Available device' list

Package: Any

Pin count: Any

Speed grade: Any

☒ Show advanced devices

☐ HardCopy compatible only

Available devices:

Name	Core v...	LEs	User I/...	Memor...	Embed...	PLL
EP2C20F484C8	1.2V	18752	315	239616	52	4
EP2C20F484I8	1.2V	18752	315	239616	52	4
EP2C20Q240C8	1.2V	18752	142	239616	52	4
EP2C35F484C6	1.2V	33216	322	483840	70	4
EP2C35F484C7	1.2V	33216	322	483840	70	4
EP2C35F484C8	1.2V	33216	322	483840	70	4
EP2C35F484I8	1.2V	33216	322	483840	70	4
EP2C35F672C6	1.2V	33216	475	483840	70	4
EP2C35F672C7	1.2V	33216	475	483840	70	4

Companion device

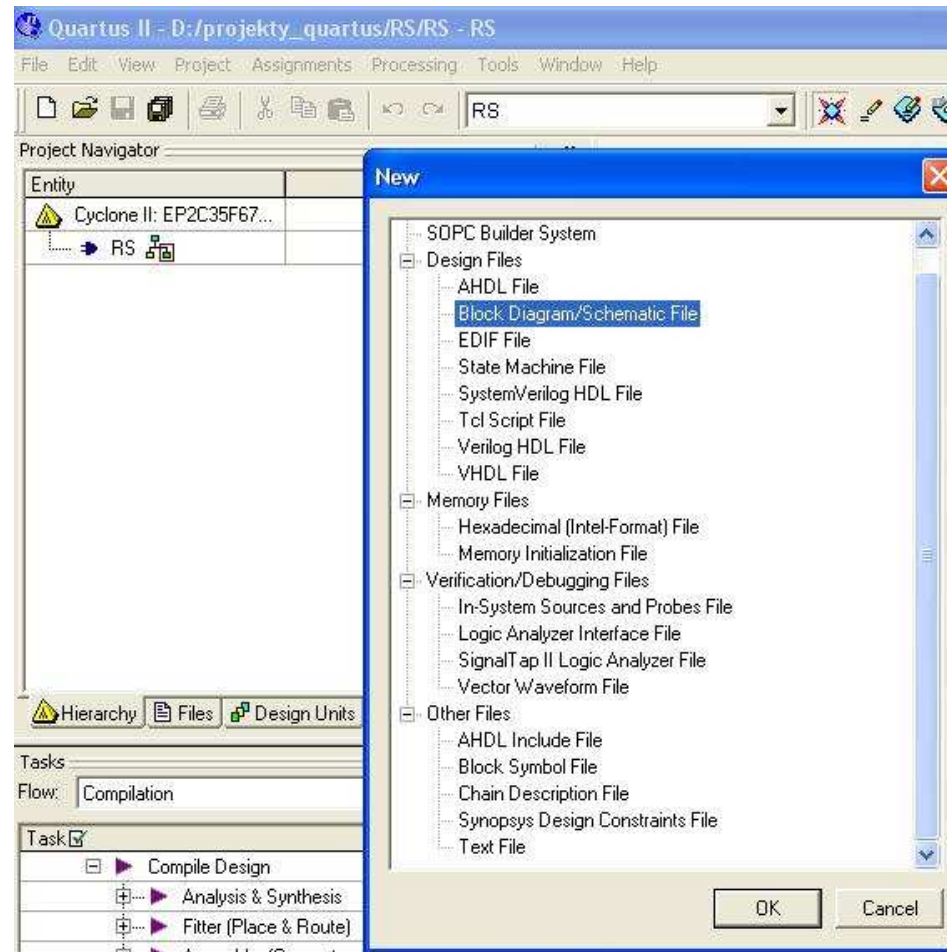
HardCopy:

☒ Limit DSP & RAM to HardCopy device resources.

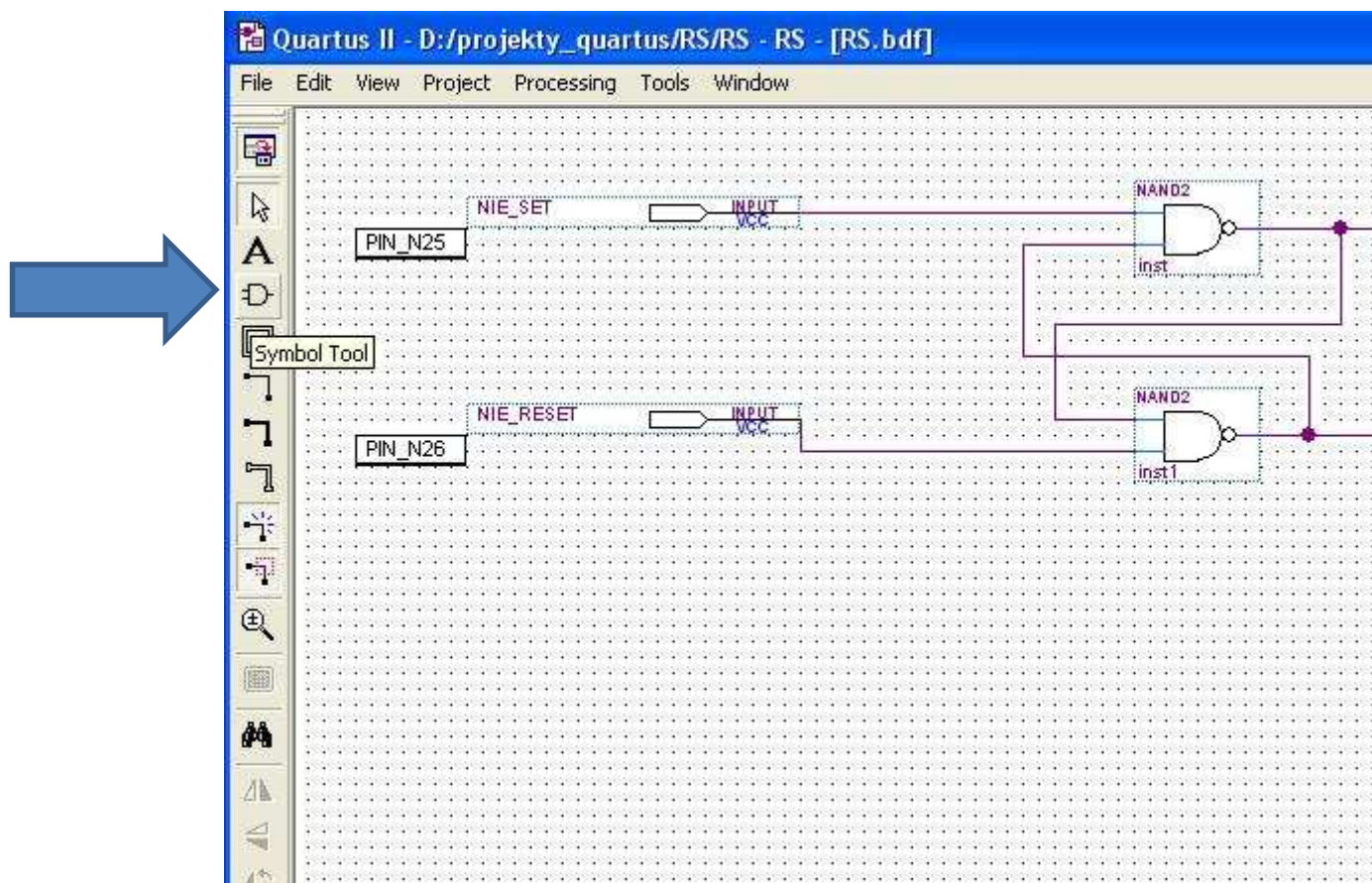
< Back Next > Finish Cancel

UWAGA: DLA DE2 EP2C35F672C6 , DLA DE2-70 EP2C70F896C6

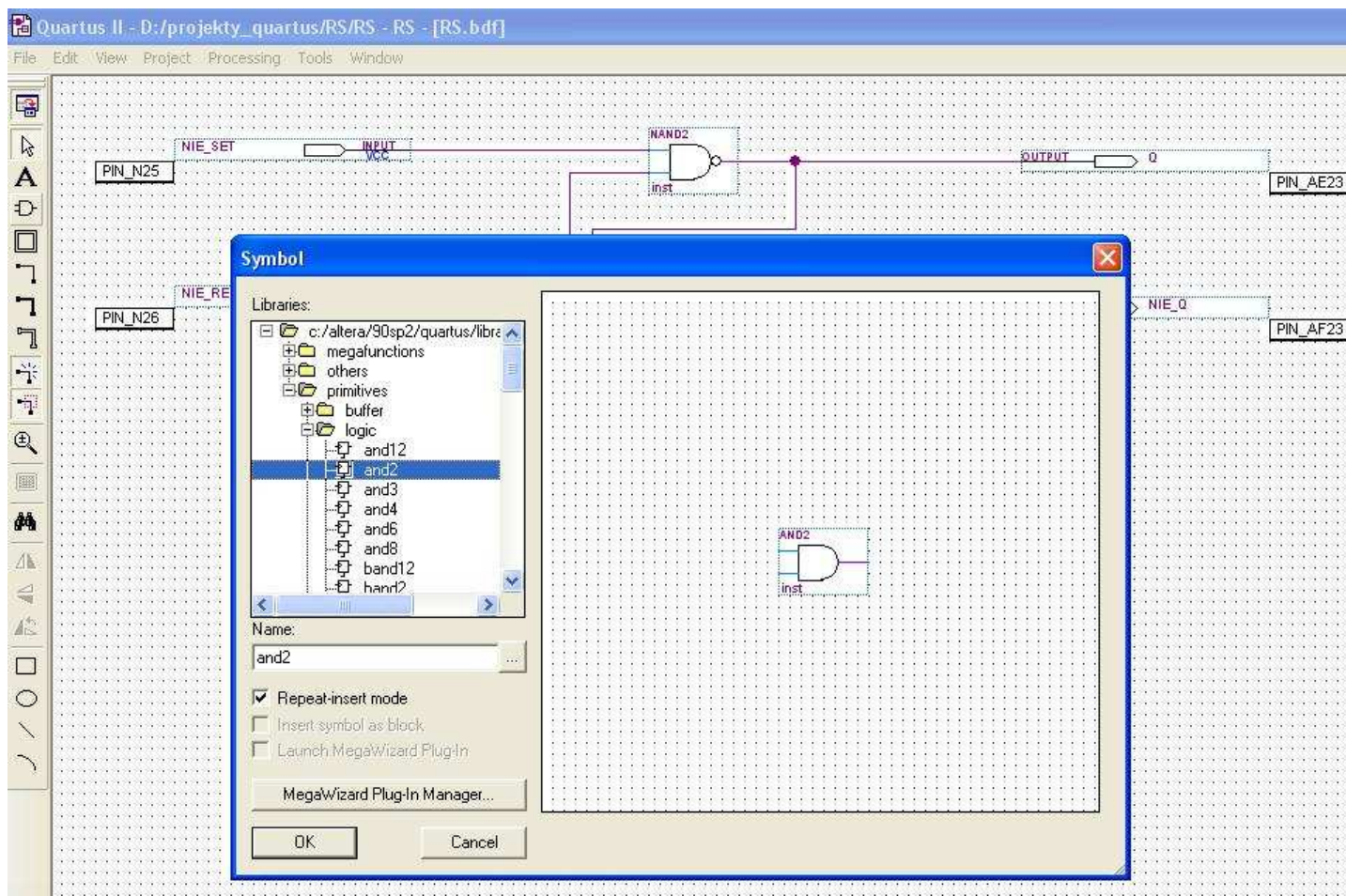
Tworzenie nowego projektu za pomocą kreatora pliki schematu blokowego



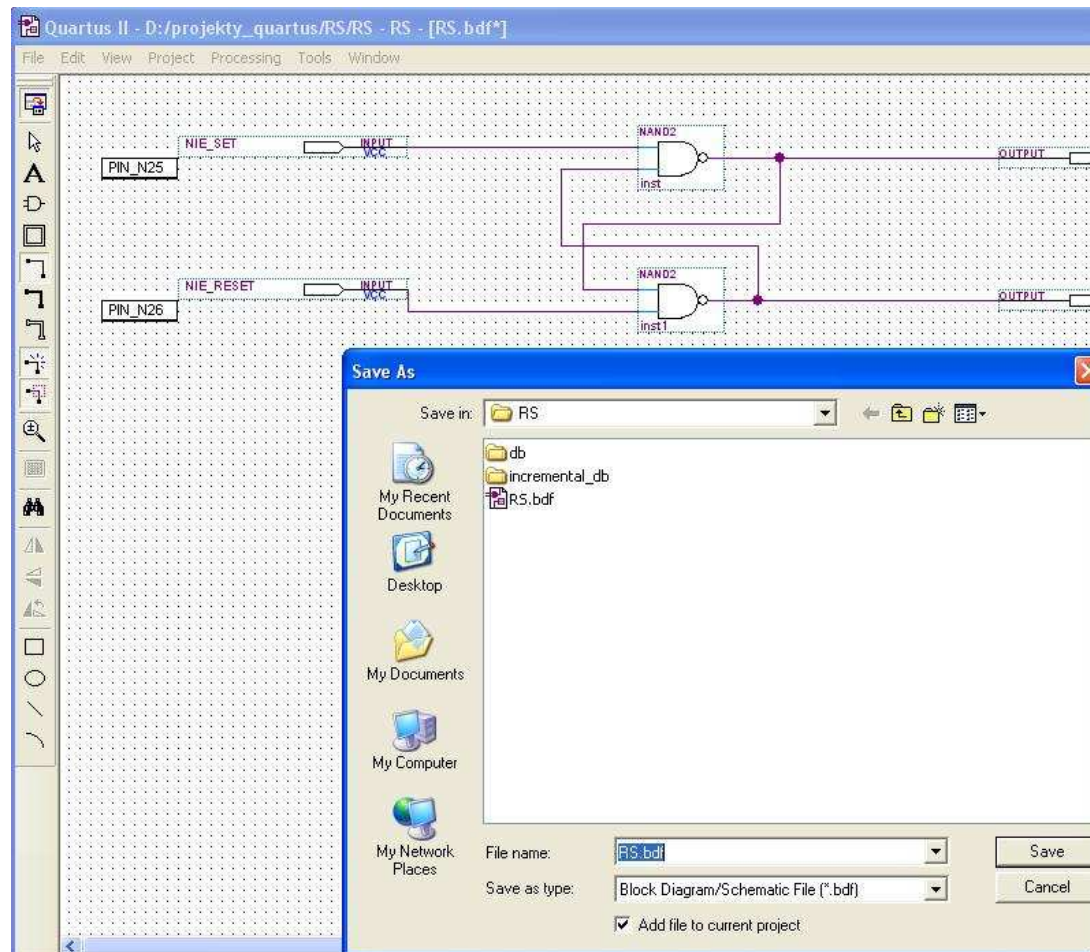
Schemat blokowy – umieszczanie podstawowych elementów logicznych 1



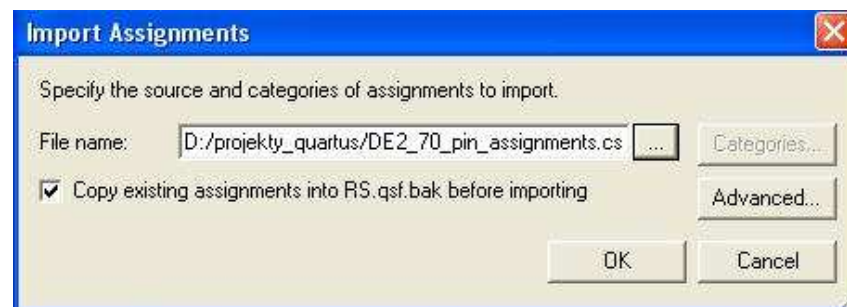
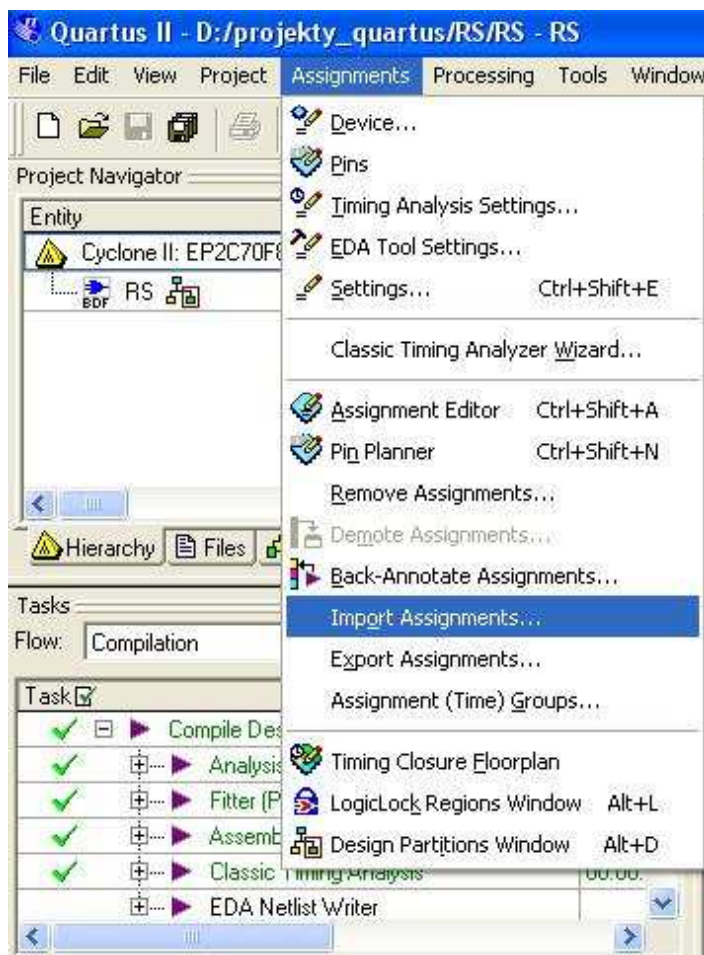
Schemat blokowy – umieszczanie podstawowych elementów logicznych 2



Schemat blokowy – zapis pliku w ramach projektu

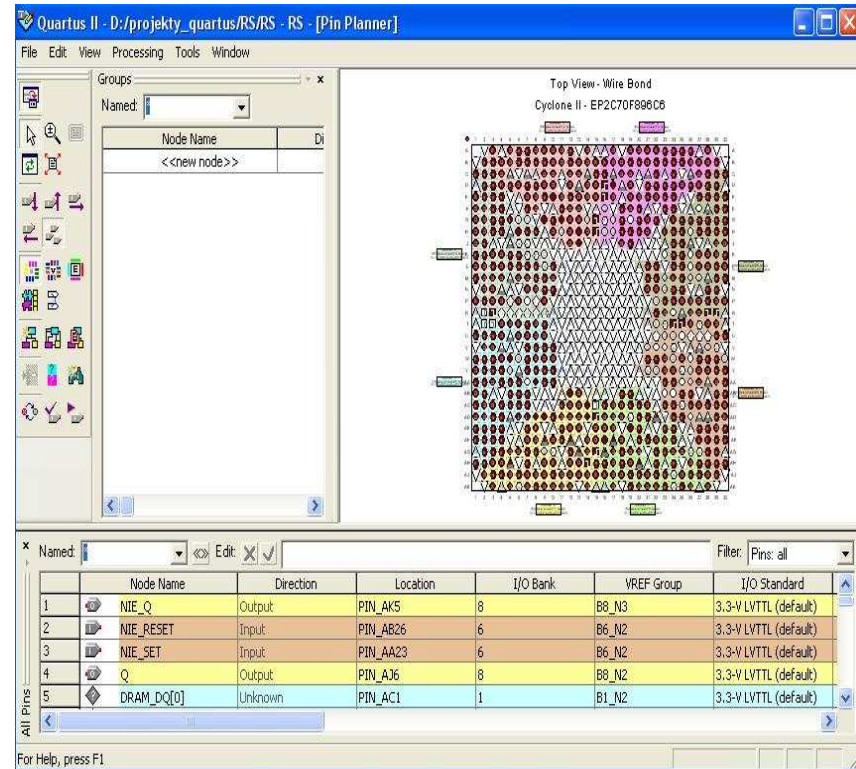


Wprowadzenie powiązań nazwa portu – pin układu FPGA



Edycja powiązań dla portów schematu – wykorzystanie Pin Planer

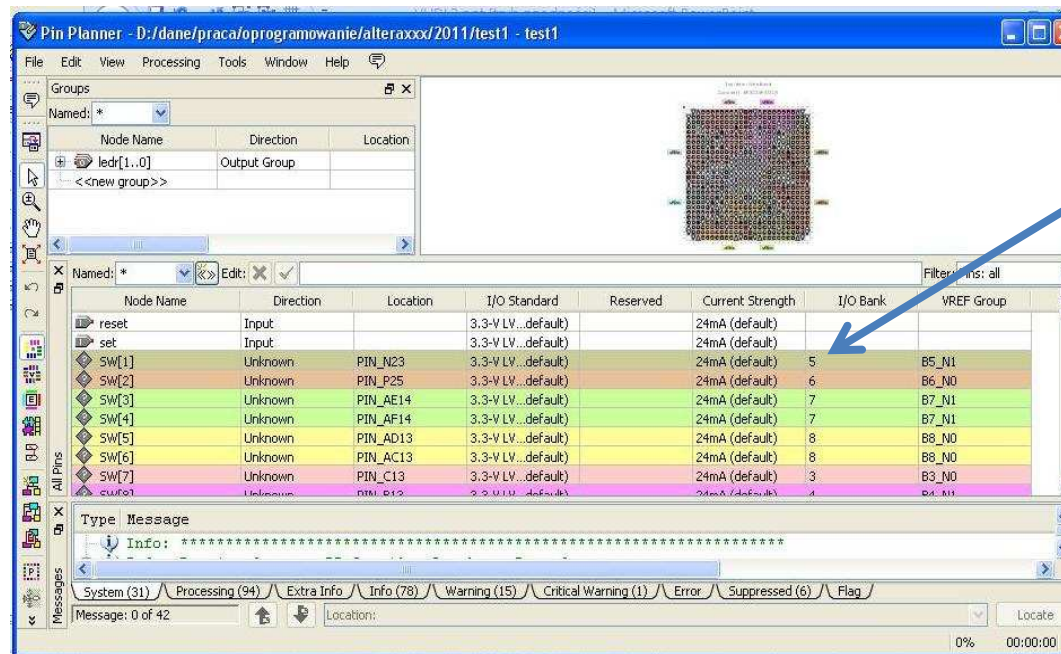
- Po kompilacji porty projektowanego układu pojawią się na liście węzłów w Pin Planer
- Dla łatwiejszego dopasowania portów do wyprowadzeń układu należy znaleźć grupy wyprowadzeń, do których przynależy wyprowadzenie, które chcemy wykorzystać.
- W naszym przypadku wybranie PIN_N26 (SW[1]) spowoduje możliwość sterowania przełącznikiem SW[1] wejścia układu SET
- Źródło:
- http://quartushelp.altera.com/9.1/mergedProjects/assign/ase/ase_pro_assignment_pins.htm



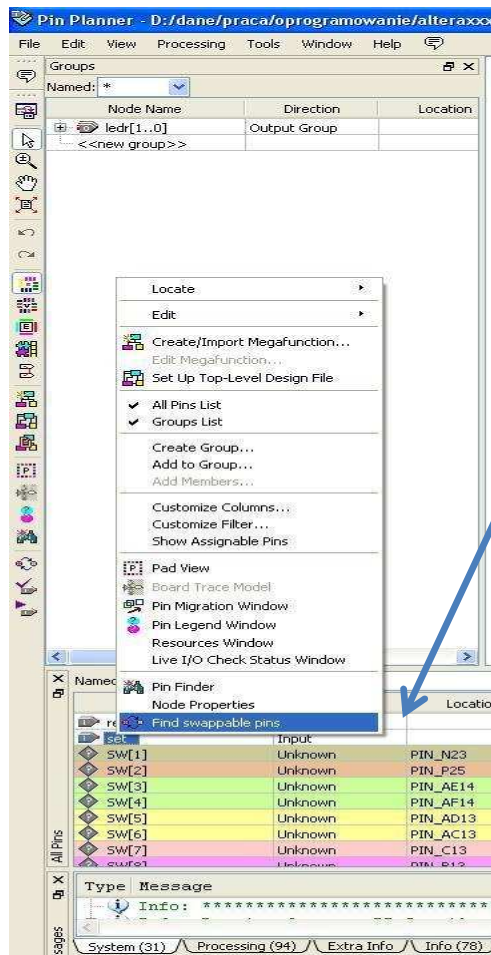
Przykład podłączenia wyprowadzenia logicznego układu do wyprowadzenia FPGA[1]

Przykład: sygnał układu SET przyporządkowujemy do tego wyprowadzenia FPGA, do którego podłączony jest przełącznik SW[1]

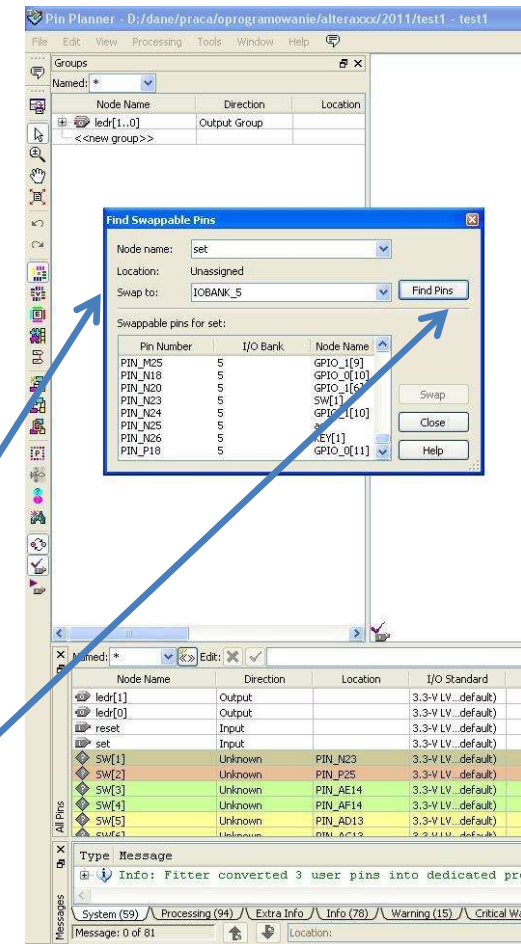
1. Sprawdzamy, do której grupy należy to wyprowadzenie układu (pin), do którego przyłączony jest przełącznik SW[1] (informację program pobiera z DE2_pin_assignments.csv) – jest to widoczne w oknie Pin Planner obok wężła SW[1] w kolumnie *location* grupa: *I/O bank 5*



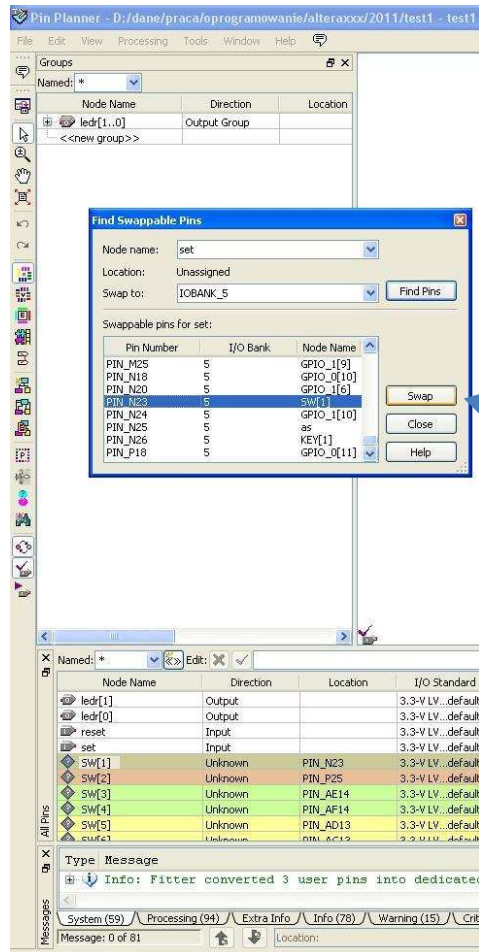
Przykład podłączenia wyprowadzenia logicznego układu do wyprowadzenia FPGA [2]



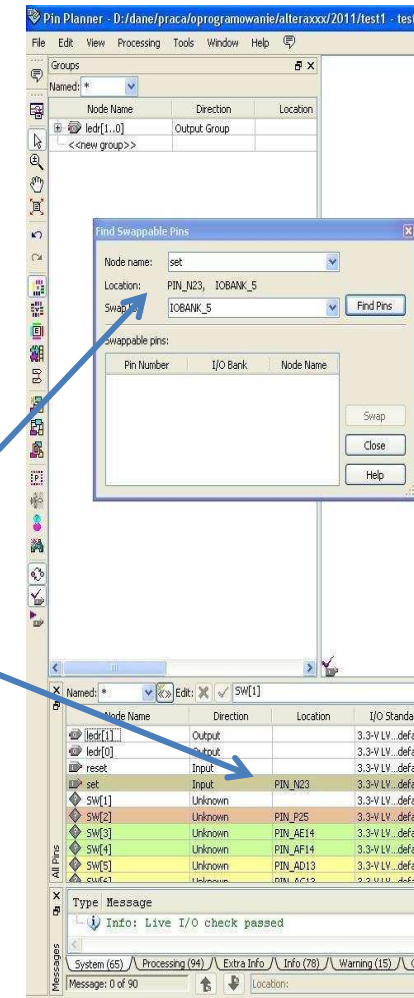
- Wybieramy sygnał SET prawym klawiszem myszy,
- dalej z pojawiającego się menu wybieramy opcję „find swappable pins”,
- a następnie w pojawiającym się oknie (pole „swap to”) określamy zakres przeszukiwanych wyprowadzeń w naszym przypadku: *I/O bank 5*
- Program znajduje możliwe wyprowadzenia do podmiany - opcja *Find pins*



Przykład podłączenia wyprowadzenia logicznego układu do wyprowadzenia FPGA [3]



- Po wybraniu wyprowadzeni a możliwa jest podmiana – *Swap*
- Widoczny efekt



Kompilacja projektu — analiza wejścia, synteza i optymalizacja modelu logicznego, przydział do sprzętu, asemblacja, analiza czasowa

Quartus II - D:/projekty_quartus/RS/RS - RS - [Compilation Report - Flow Summary]

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

Entity	Logic Cells	Dedicated Logic
Cyclone II: EP2C70F896C6	2 (2)	0 (0)

Tasks

Flow: Compilation

Task	Time
Compile Design	00:00
Analysis & Synthesis	00:00
Fitter (Place & Route)	00:00
Assembler (Generate programming files)	00:00
Classic Timing Analysis	00:00
EDA Netlist Writer	00:00

Compilation Report - Flow Summary

Flow Summary

Flow Status: Successful - Fri Oct 01 18:29:34 2010

Quartus II Version: 9.0 Build 235 06/17/2009 SP 2 SJ Web Edition

Revision Name: RS

Top-level Entity Name: RS

Family: Cyclone II

Device: EP2C70F896C6

Timing Models: Final

Met timing requirements: Yes

Total logic elements: 2 / 68,416 (< 1 %)

Total combinational functions: 2 / 68,416 (< 1 %)

Dedicated logic registers: 0 / 68,416 (0 %)

Total registers: 0

Total pins: 4 / 622 (< 1 %)

Total virtual pins: 0

Total memory bits: 0 / 1,152,000 (0 %)

Embedded Multiplier 9-bit elements: 0 / 300 (0 %)

Total PLLs: 0 / 4 (0 %)

Quartus II

Full Compilation was successful (536 warnings)

OK

Messages

Type	Message
Info	Fitter converted 3 user pins into dedicated programming pins
Info	Fitter is using the Classic Timing Analyzer
Info	Timing requirements not specified -- quality metrics such as performance and power consumption may be sacrificed to reduce compilation time
Info	Starting register packing
Info	Finished register packing
Warning	Ignored locations or region assignments to the following nodes

System (10) Processing (51) Extra Info Info (46) Warning (5) Critical Warning Error Suppressed (6) Flag

Message: 0 of 624

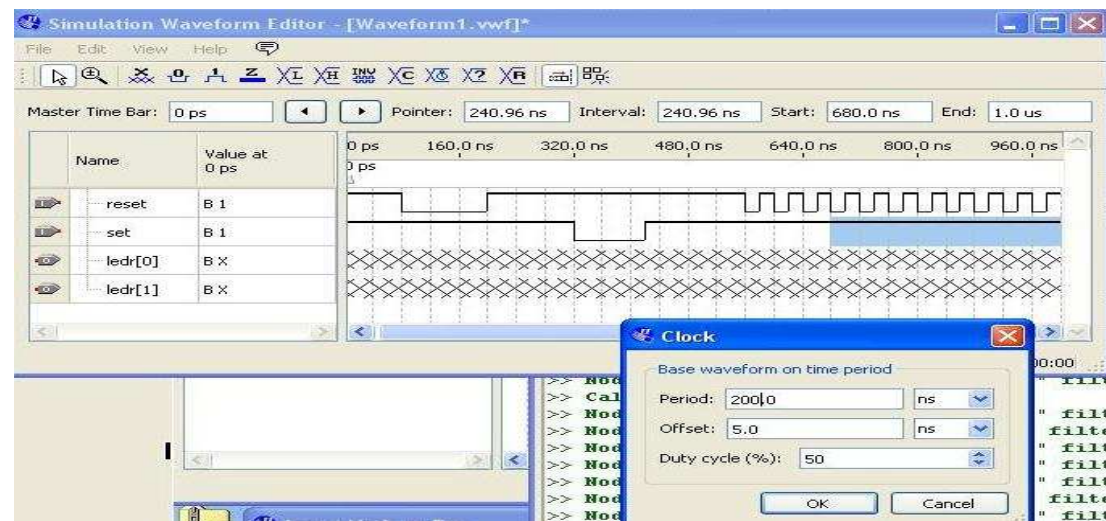
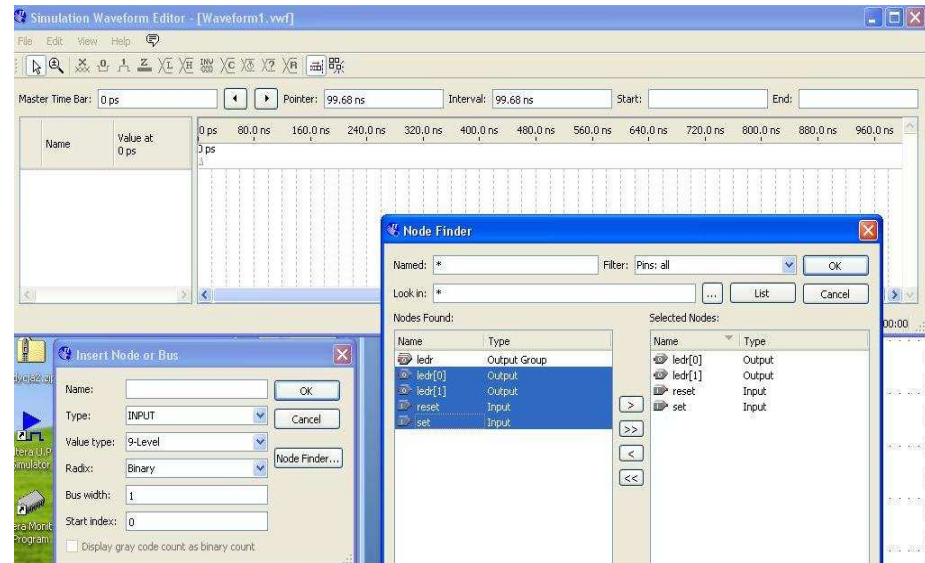
Location:

For Help, press F1

Idle

Symulacja - Altera U.P. Simulator

- File>Open Project
- File>New simulation Input File
- Edit >Insert > Insert Node or Bus
 - Node Finder



Podstawowe działania - symulator

Definiowanie wektora testowego: zbioru sygnałów, okresu symulacji i wymuszeń: Altera U.P. Simulator

- Przydatne opcje:
 - Edit> End time
 - View>Fit in window
 - Edit>Insert Node or Bus Node
 - View>Snap to grid
 - Selection Tool – ikona w kształcie strzałki
 - Waveform editing Tool – ikona ze zmieniającym się stanem magistrali
 - Edit>Value - edycja stanu wymuszenia

Symulacja - Altera U.P. Simulator

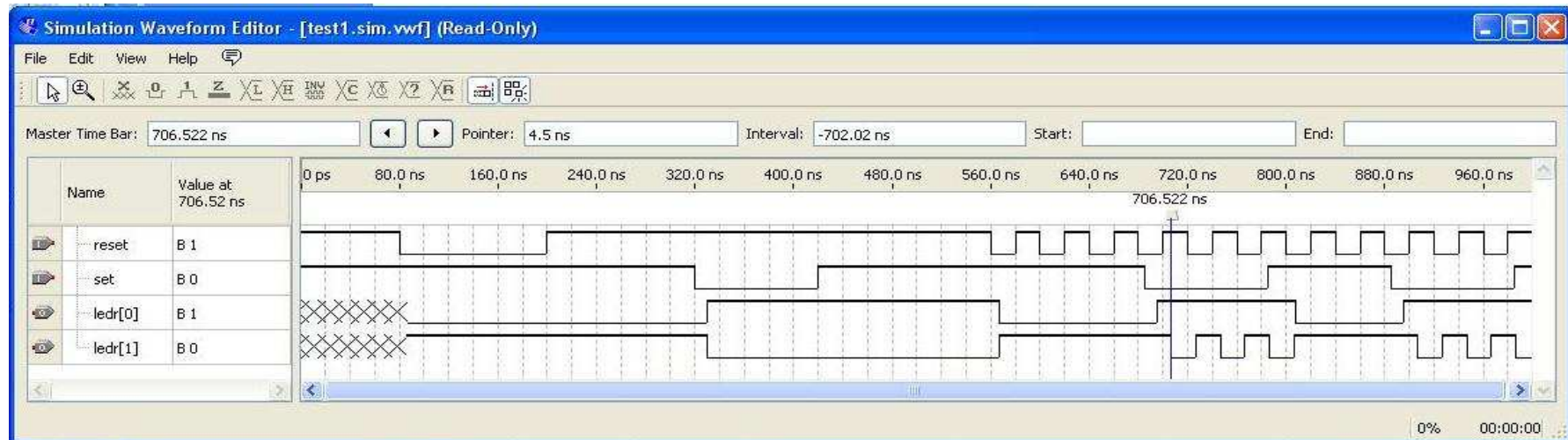
Użyteczne opcje:

- Edit > Set End Time
- Edit > Grid Size
- Edit > Snap to Grid
- Edit > Snap to Transition

Przygotowanie i kroki symulacji:

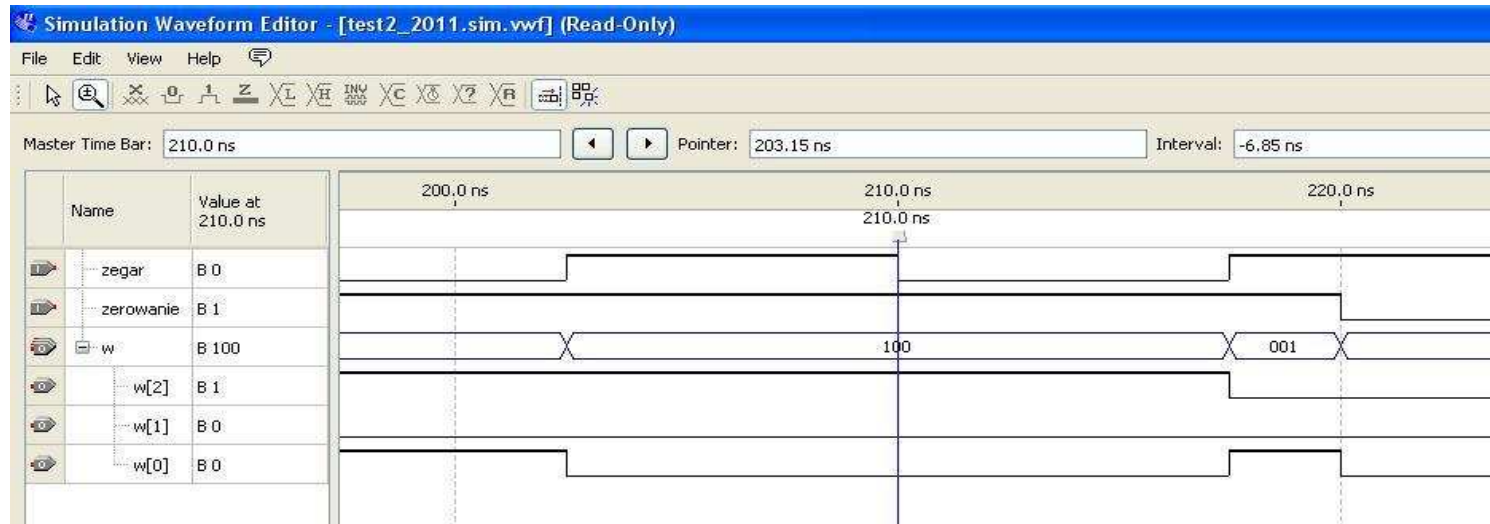
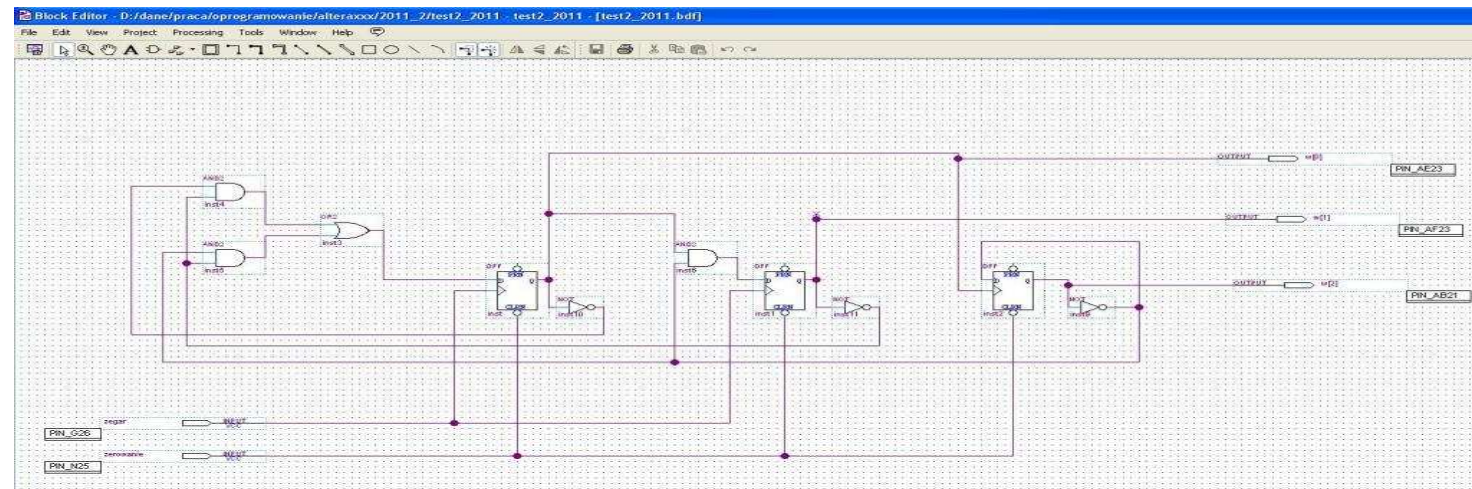
- Assign > Simulation Settings — określenie pliku wektorów testowych oraz typu symulacji
- Processing > Generate Simulation Netlist
- Processing > Start Simulation

Symulacja - Altera U.P. Simulator

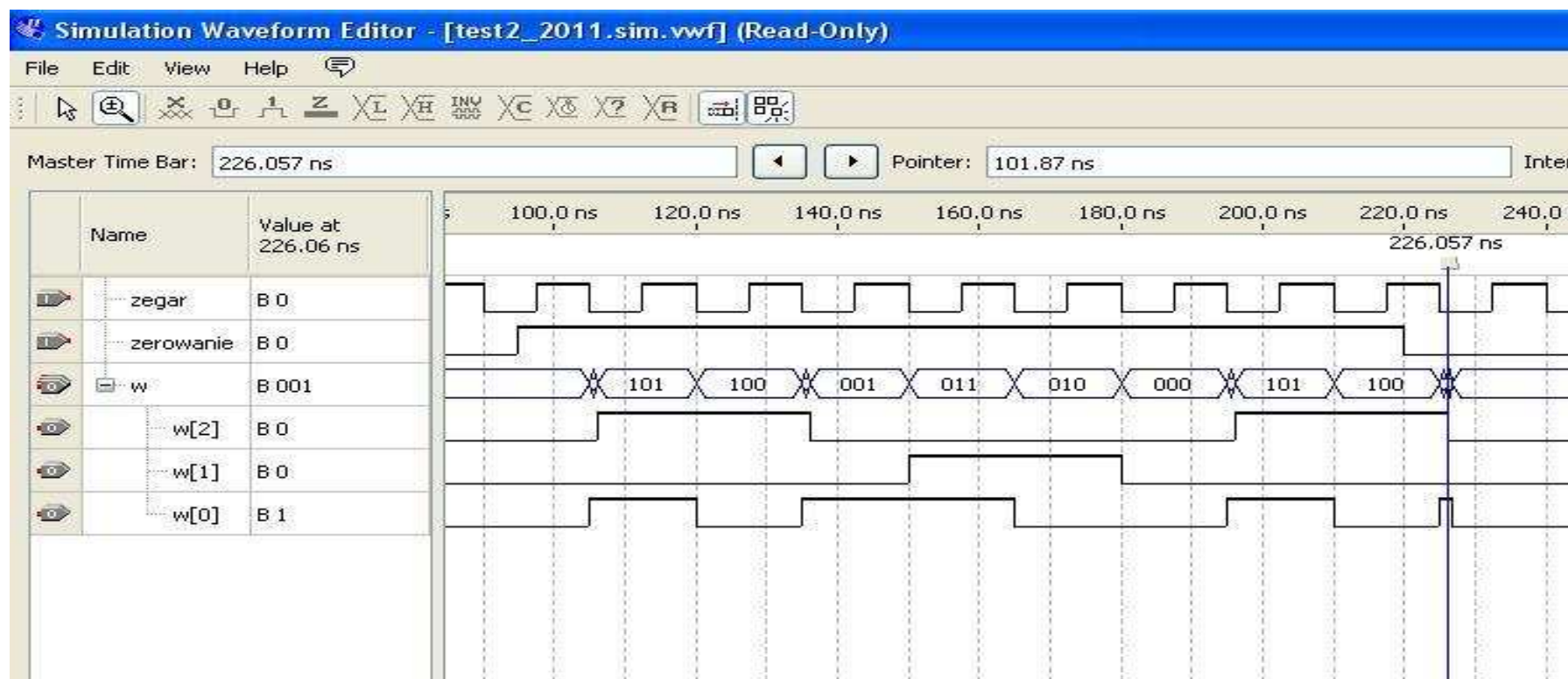


- Wyniki symulacji widoczne w Simulation Waveform Editor
- Szersze informacje na temat symulacji i edycji pliku wektorów testowych w:
Introduction to Simulation of VHDL Designs: Altera corporation University Program May 2011

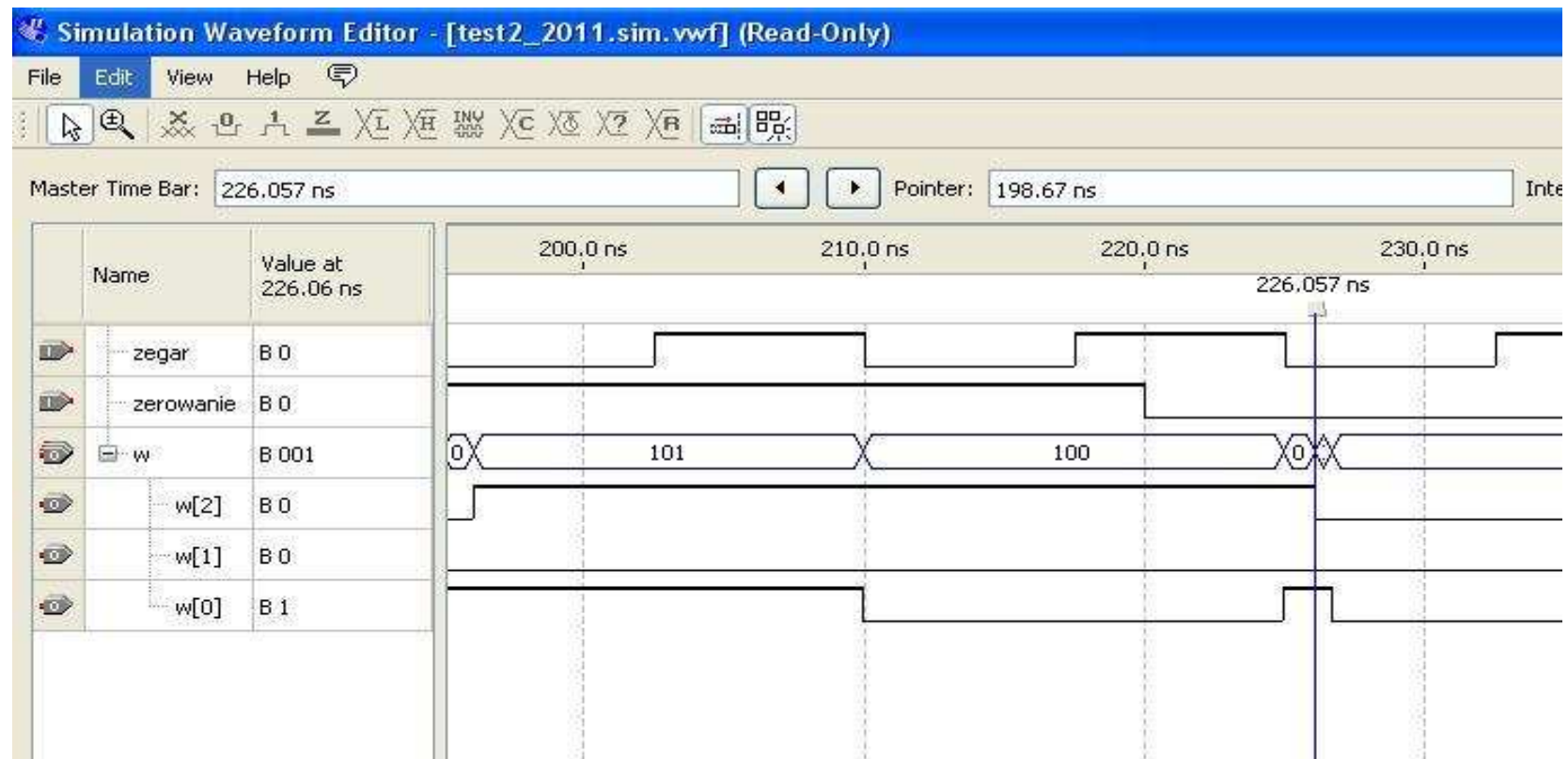
Przykłady symulacji – symulacja funkcjonalna logika działania



Przykłady symulacji – symulacja czasowa - realizacja w sprzęcie



Przykłady symulacji – symulacja czasowa (powiększenie)



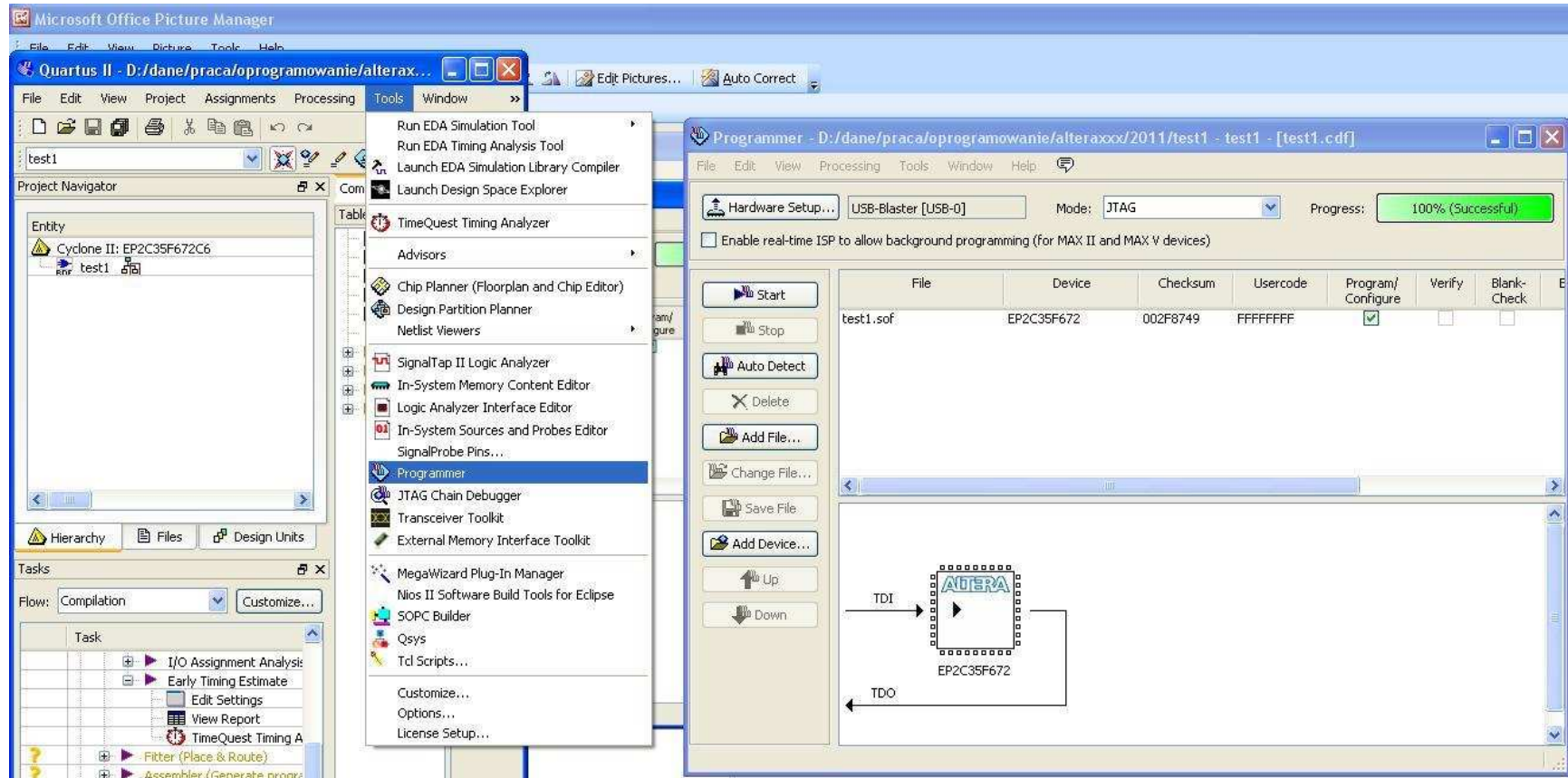
Podstawowe działania ALTERA

- **Programowanie i konfiguracja FPGA**

Tryb: JTAG – ładowanie danych konfiguracyjnych układu bezpośrednio do FPGA, pozostają do wyłączenia zasilania. Do programowania w tym trybie: przełącznik trybu programowania płyty DE2 w stan RUN

- Tools>Programmer
- Hardware Setup > USB-blaster
- Korzystamy z *.sof (SRAM object file) (Add file)
- Ustawienie przełącznika DE2 w stan RUN
- Program/Configure
- Start

Programowanie FPGA



Tools/Programmer ; Wykorzystujemy interfejs USB-Blaster

Podstawowe działania VHDL(1)

- **Nowy plik VHDL:**
 - File>New>VHDL
 - określenie nazwy: File>Save As > Box Add file to current project
- **Umieszczanie wzorców elementów w pliku VHDL:**
 - Edit>Insert Template>VHDL
- **Dodawanie/usuwanie plików do/z projektu:**
 - Assignments> Settings> Files
 - Project> Add/Remove Files in Project

Podstawowe działania VHDL (2)

- **Przydział sygnałów do wyprowadzeń układu:**
 - Assignments> Import Assignments
 - Wybrać plik *DE2_pin_assignments.csv*
 - oznaczenia sygnałów w projekcie zgodne z *DE2 user manual* lub powiązanie nazw portów z oznaczeniem wyprowadzeń
 - Pin Planner (SLAJDY 52-56)
- **Kompilacja:**
 - Processing>Start Compilation

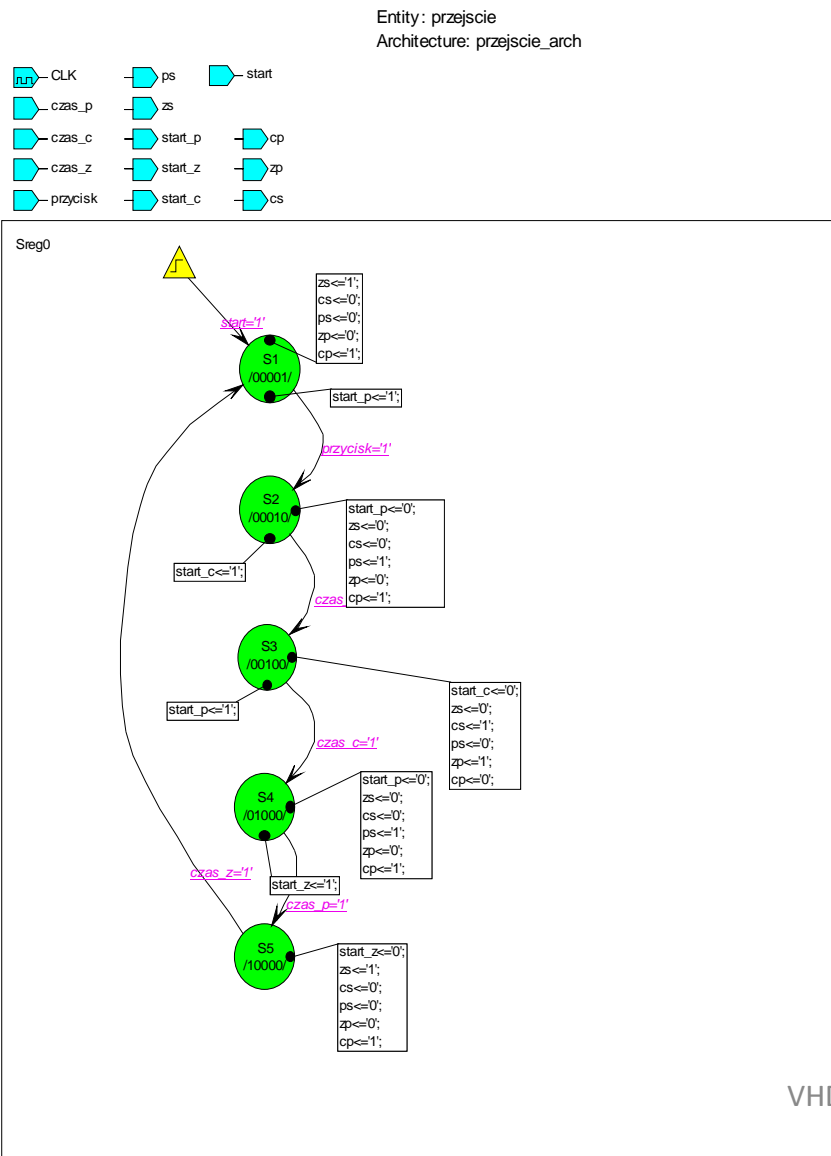
Polecane źródła informacji na temat DE2 i Quartus

- **Prezentacje na temat podstaw korzystania z Quartus**
 - *Quartus II Introduction Using VHDL Design*
 - *Introduction to Simulation of VHDL Designs: Altera corporation University Program*
May 2011

Oprogramowanie dostępne w celach edukacyjnych na
www.altera.com w sekcji Altera University Program Design
Software

- **Podręcznik użytkownika płyty DE2**
 - *DE2 Development and Education Board User Manual*

Przekształcenie grafu stanów w opis automatu w VHDL



```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

```

entity przejście is

port (

CLK: in STD_LOGIC;

czas_c: in STD_LOGIC;

...

end przejście;

architecture przejście_arch of przejście is

attribute enum_encoding: string;

type Sreg0_type is (

S1, S2, S3, S4, S5

);

attribute enum_encoding of Sreg0_type: type is

"00001 " &

-- S1

...

signal Sreg0: Sreg0_type;

begin

Sreg0_machine: process (CLK)

begin

if CLK'event and CLK = '1' then

if start='1' then

...

end process;

-- signal assignment statements for combinatorial outputs

zs_assignment:

zs <= '1' when (Sreg0 = S1) else

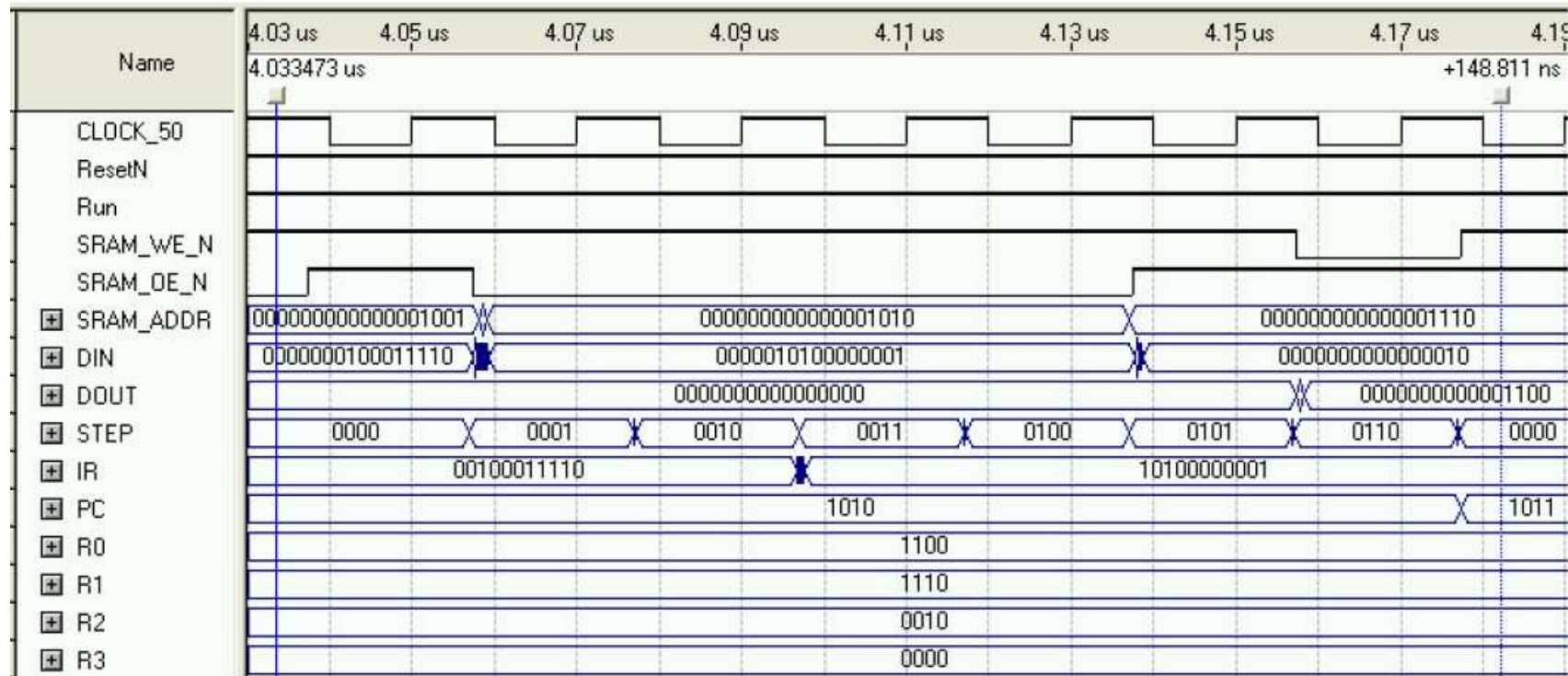
...

VHDL, DE2, Quartus; end przejście_arch;

Przykładowe wyniki pracy:

Symulacja pracy prostego procesora,

zapis zawartości rejestru do pamięci adresowanej zawartością rejestru



Implementacja VHDL i DE2 na laboratorium PTC, autor: student 3 roku informatyki PP 2010/2011