Aleksy Barcz
Marcin Harasimczuk

31 maj 2011

Dokumentacja etapu II

Projekt z przedmiotu PUCY - zadanie 11L-14D

1. Zadanie w etapie II.

Zrealizowaliśmy, za pomocą języka VHDL, program udostępniający funkcjonalność mikrokontrolera. Wykorzystaliśmy układy wejścia wyjścia opisane za pomocą języka AHDL w pierwszym etapie (*LPT_OUT* i *PS2_IN*). Na nasz projekt mikrokontrolera składają się dodatkowo takie moduły jak:

- CPU - moduł procesora.
- RAM - moduł pamięci RAM.
- ROM - moduł pamięci ROM.
- MUL - moduł mnożący.
- Microcontroller - moduł łączący powyższe składniki.

2. Opis realizacji modułów.

2.1. CPU

Moduł procesora został zaprojektowany za pomocą dwóch automatów. Do zadań automatu AUT_CPU należy:

- Pobieranie rozkazów z szyny w stanach AUT_CPU_FETCH i AUT_CPU_FETCH2.
- Dekodowanie pobranych kodów rozkazów według pięciu najstarszych bitów rejestru rozkazu REG_CMD.
- Pobieranie odpowiedniej ilości bajtów rozkazu (spis rozkazów w tabeli 1 "lista rozkazów") i inkrementowanie licznika rozkazów w stanach AUT_CPU_CMD, AUT_CPU_CMD1, AUT_CPU_CMD2, AUT_CPU_CMD3, AUT_CPU_CMD4, AUT_CPU_CMD5, AUT_CPU_CMD6.
- Wykonanie, lub skok do stanów, w których wykonanie instrukcji zostanie kontynuowane w stanie AUT_CPU_EXE.
- Wykonywanie instrukcji I/O w stanach AUT_CPU_IN1, AUT_CPU_IN2, AUT_CPU_IN3, AUT_CPU_OUT1, AUT_CPU_OUT2.
- Wykonanie instrukcji zapisu do pamięci RAM w stanach AUT_CPU_RAM_WR1, AUT_CPU_RAM_WR2, AUT_CPU_RAM_WR3 i AUT_CPU_RAM_WR4.
- Wykonanie instrukcji odczytu z pamięci RAM w stanach AUT_CPU_RAM_RD1, AUT_CPU_RAM_RD2, AUT_CPU_RAM_RD3, AUT_CPU_RAM_RD4.
- Wykorzystanie modułu mnożącego w stanie AUT_CPU_MUL.

Do zadań automatu AUT_BUS należy:

- Uruchamianie komunikacji z szyną systemu na życzenie automatu AUT_CPU za pomocą sygnału SIG_BUS_START ustawionego na "1".
- Wysterowanie sygnałów szyny MREQ, IORQ, RD i WR w zależności od cyklu jakiego żąda automat AUT_CPU.
- Wstrzymywanie pracy systemu w oczekiwaniu na zakończenie współpracy z modułami I/O za pomocą czytanego sygnału SIG_WAIT. Automat AUT_CPU wstrzymywany jest za

pomocą sygnału SIG_BUS_STOP ustawionego na "0".

Dodatkowo moduł CPU wyposażony jest w dwa moduły lpm_ram_dp implementujące funkcjonalność rejestrów uniwersalnych. Są to moduły adresowane za pomocą 3 bitów, o szerokości słowa 8-bit i mieszczące 8 słów tej szerokości. W wewnętrznej implementacji zastosowano 3 rejestry: REG_A, REG_B, oraz REG_ACC (akumulator), do którego wpisywane są wyniki. Uwagę należy zwrócić na komendę [09] Rd=>RAM(RaoRb), gdzie taki zestaw rejestrów jest zbyt mały i zastosowano dodatkowe 2 rejestery REG_A2 i REG_B2 do zapamiętania wartości potrzebnej do budowy adresu w pamięci RAM (przepisanie wartości jest dokonywane w stanie AUT_CPU_CMD4).

Ostatecznie moduł CPU podłączony jest do trójstanowej szyny danych za pomocą elementu e0 typu lpm_bustri. Moduł mnożący MUL został podłączony jako element mul0.

2.2. RAM

Blok pamięci RAM został ograniczony do 1024B ze względu na ilość dostępnych EAB. Przestrzeń adresowa zaczyna się od adresu 0x1000 i kończy na adresie 0x1399 (4096..5119). Moduł RAM został utworzony za pomocą elementu lpm_ram_dq o szerokości słowa 8-bit, 10 bitowym adresowaniu i 1024 słowach. Na wejście adresowe i wejście danych elementu lpm_ram_dq podawane są wartości z zatrzasków LA i LD, które zatrzaskują dane z szyny adresowej i danych. Wyjście modułu RAM podane jest na element realizujący szynę trójstanową lpm_bustri.

2.3. ROM

Zaprojektowano blok pamięci ROM o rozmiarze 90B i przestrzeni adresowej od 0x0000 do 0x0059 (0..89). Czas ustalenia się danych przy odczycie to 25ns, natomiast czas wyjścia z D po cofnięciu MREQ i RD został ustalony na 20ns. Ze względu na fakt, iż T(CPU) = 1/f = 50ns co jest zdecydowanie większe niż 25ns, ROM nie wystawia sygnału WT. Moduł ROM adresowany jest za pomocą 7 bitów.

Dodatkowo w pamięci ROM umieszczamy za pomocą właściwości LPM_FILE elementu lpm_rom kod programu maszynowego w postaci pliku ".mif" utworzonego za pomocą programu WinTim (kod programu użytego przy testowaniu opisany jest na listingu "program testowy").

2.3.1 Program

Program dla mikrokontrolera został napisany przy użyciu programu WinTim.
Dzięki wykorzystaniu jego możliwości, udało się zarówno stworzyć zestaw komend pozwalających na pisanie programu dla mikrokontrolera w assemblerze, jak i sformułować ich definicje w sposób przejrzysty.
Jedyny problem jaki się pojawił to zmienna długość komendy mikrokontrolera. Nie udało się skłonić WinTima do pracy z długością komendy 8 lub 16 lub 24 bity, okazało się że jedyną możliwością jest ustawienie długości słowa (WORD) na 24, a następnie ręczne edytowanie adresów w wynikowym pliku .mif, aby zgadzały się z długościami komend.
Kroki potrzebne do przerobienia oryginalnego mif'a aby współpracował z Quartusem:

- ustawienie width=8 i depth = 90
- zmiana adresów kolejnych komend w zależności od długości poprzednich
- zmiana adresów etykiet (dla skoków)
- zmiana wartości, którą nadpisywana jest reszta pamięci z 00 na STOP (0x78)

2.4. MUL

Zgodnie z treścią zadania zrealizowaliśmy moduł mnożący metodą podstawową. Automat AUT_CPU steruje modułem mnożącym wpisując dane do rejestrów REG_A i REG_B. Uruchomienie modułu następuje poprzez wysterowanie sygnału MUL_GO na stan "0". AUT_CPU czeka w stanie AUT_CPU_MUL na stan niski sygnału MUL_READY. Diagramy związane z

modułem mnożącym znajdują się na listingu 2 "Moduł mnożący".

2.5. Microcontroller

Jest to układ łączący wszystkie składniki systemu w całość (schemat poglądowy systemu widoczny jest na schemacie 3 "architektura systemu"). Do jego zadań należą:

- Propagacja sygnału WAIT (WT) między układami (zwłaszcza z zewnętrznych układów I/O).
- Propagacja sygnału RESET do wszystkich układów z przełącznika SW3B.
- Zatrzaskiwanie stanu linii danych przy każdej operacji I/O i MEM (wykrywanie błędów).
- Wyświetlanie na diodach kolejnych bajtów wczytywanych za pomocą klawiatury.
- Podłączenie układów LPT_OUT, PS2_IN, CPU, RAM i ROM do szyny systemowej.

3. Spis rysunków i tabel.

Tabela 1 - Lista rozkazów

Ilość bajtów rozkazu	Rozkaz	Kod	Opis
1	*[00] JMP 0*	"00000"	
1	*[0F] STOP*	"01111"	
2	*[01] JMP Rd=0,A*	"00001"	Jeśli Rd = 0 skok pod adres A
2	*[02] JMP A*	"00010"	
2	*[03] Rd<=Ra*	"00011"	
2	*[04] Rd<=Ra+Rb*	"00100"	
2	*[05] Rd<=Ra-Rb*	"00101"	
2	*[06] Rd<=Ra#Rb*	"00110"	
2	*[07] Rd<=Ra&Rb*	"00111"	
2	*[08] Rd<=RAM(RaoRb)*	"01000"	
2	*[09] Rd=>RAM(RaoRb)*	"01001"	
3	*[0A] Rd<=RAM(A)*	"01010"	
3	*[0B] Rd=>RAM(A)*	"01011"	
2	*[0C] Rd<=INP(A)*	"01100"	Wejście z klawiatury
2	*[0D] Rd=>OUT(A)*	"01101"	Wyjście na drukarkę
2	*[0E] Rd<=DI*	"01110"	
2	*[1E] Rd<=Ra op Rb*	"11110"	op = mnożenie metodą podstawową

Diagram 2 - moduł mnożący (metoda podstawowa)

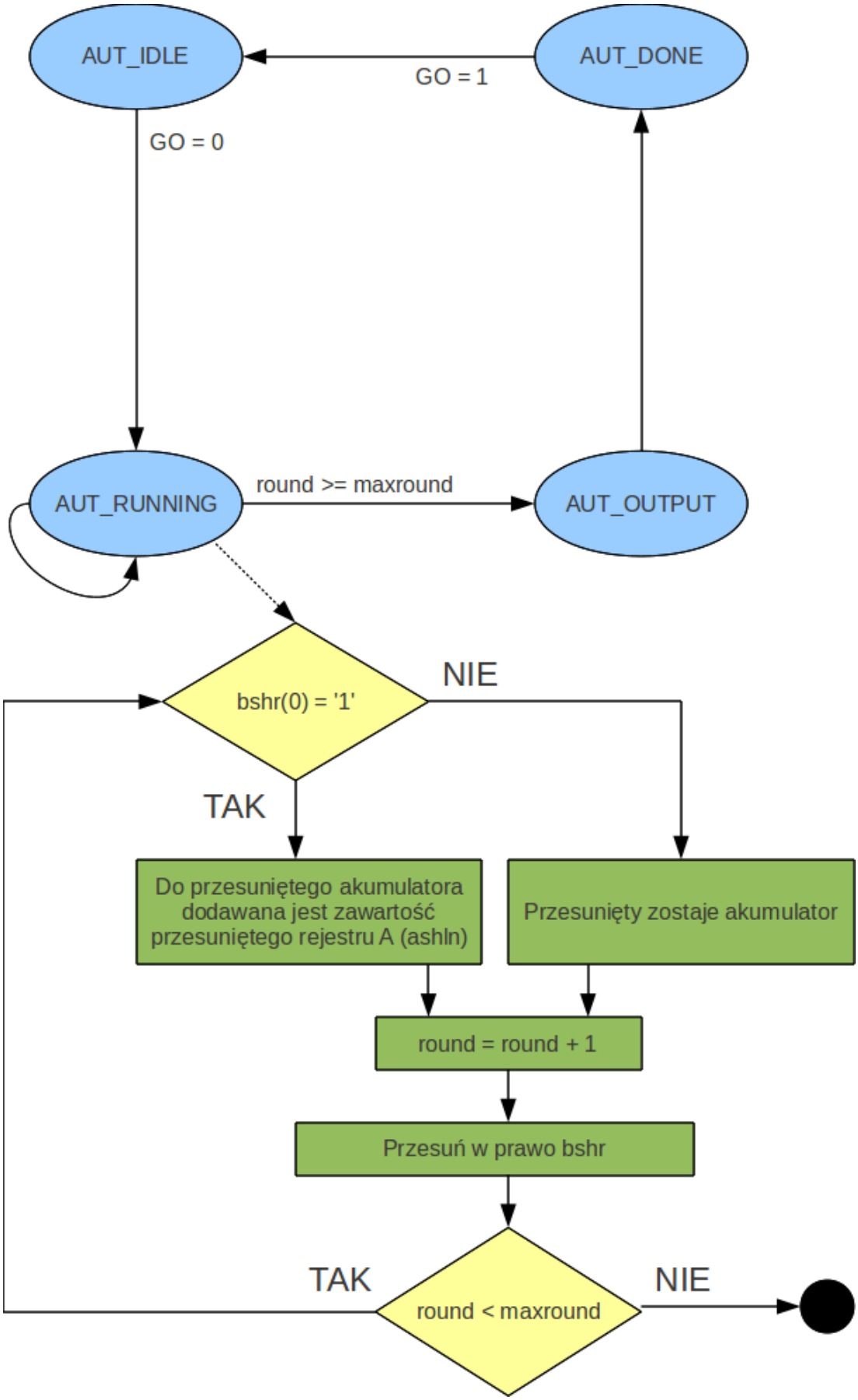
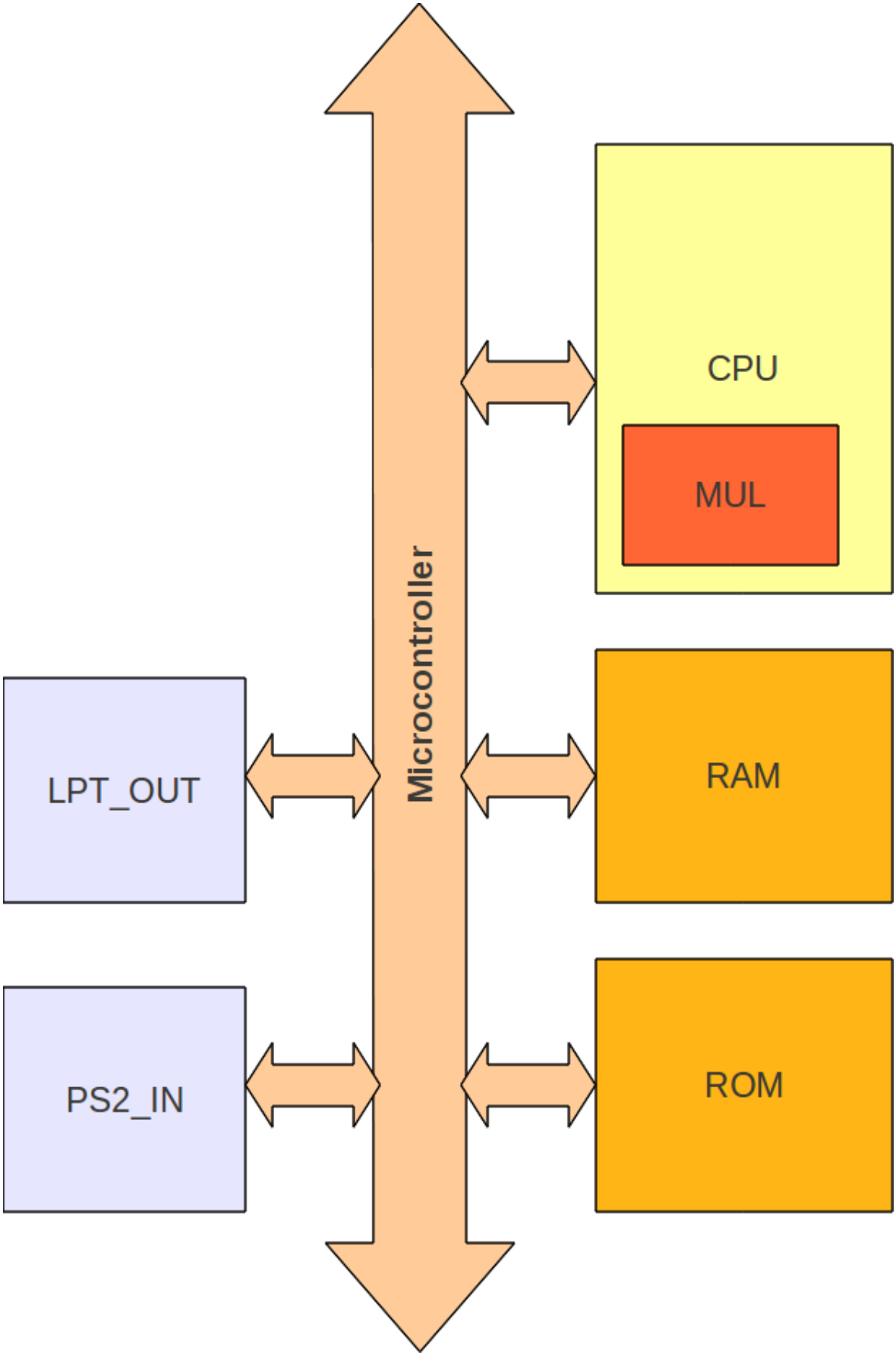


Diagram 3 - architektura systemu



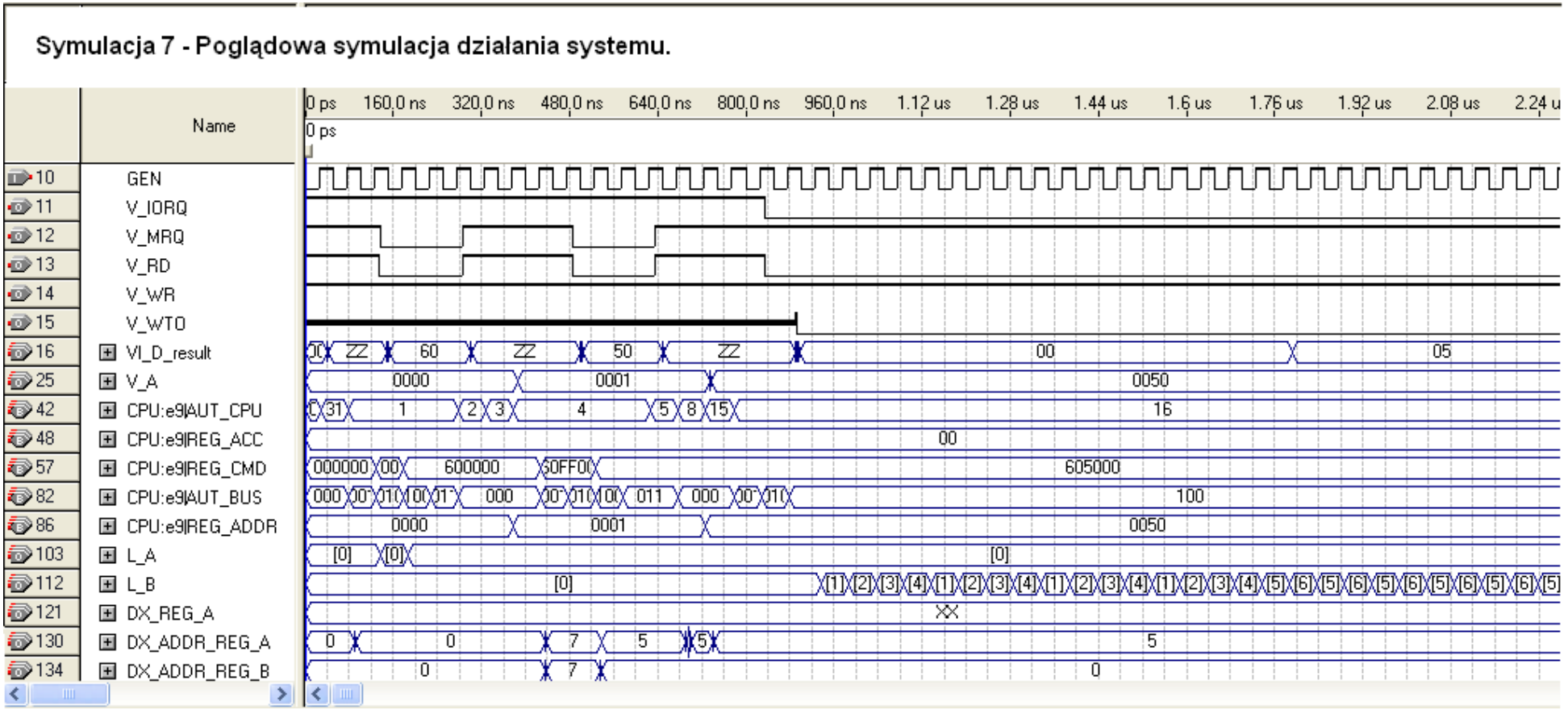
```
-- FPGA-CPU TEST PROGRAM
-- Altera Instruction Memory Initialization File
Depth = 90;
Width = 8;
Address_radix = HEX;
Data_radix = HEX;
Content
Begin
-- Use NOPS for default instruction memory values
    [40..59]: 78; -- STOP
-- Place MIPS Instructions here
-- Note: memory addresses are in words and not bytes
-- i.e. next location is +1 and not +4

-- TEST program for microcontroller
    00: 70 01;   -- START:  MOVI R0, H#01   ; constant '1'
    02: 61 50;   --     IN R1, H#50     ; command code: 0=arithmetics, 1=SWR(Ra,Rb,Rd),
    2=LWR&OUT(Ra,Rb)
    04: 09 1E;   --     JZ R1, ARITH
    06: 29 10;   --     SUBA R1, R1, R0 ; R1 := R1 - 1
    08: 09 13;   --     JZ R1, SAVE
    0A: 60 50;   -- LOADP:  IN R0, H#50     ; Ra
    0C: 61 50;   --     IN R1, H#50     ; Rb
    0E: 42 01;   --     LWR R2, R0, R1  ; R2 = RAM(R0.R1)
    10: 6A 10;   --     OUT R2, H#10    ; print R2
    12: 00;   --    JORG
    13: 60 50;   -- SAVE:   IN R0, H#50
    15: 61 50;   --     IN R1, H#50
    17: 62 50;   --     IN R2, H#50     ; RAM(R0.R1) = R2
    19: 4A 01;   --     SWR R2, R0, R1
    1B: 6A 10;   --     OUT R2, H#10    ; print R2
    1D: 00;   --    JORG

    1E: 60 50;   -- ARITH:  IN R0, H#50
    20: 61 50;   --     IN R1, H#50
    22: 22 01;   --     ADDA R2, R0, R1
    24: 6A 10;   --     OUT R2, H#10
    26: 2A 01;   --     SUBA R2, R0, R1
    28: 6A 10;   --     OUT R2, H#10
    2A: 3A 01;   --     AND R2, R0, R1
    2C: 6A 10;   --     OUT R2, H#10
    2E: 32 01;   --     OR R2, R0, R1
    30: 6A 10;   --     OUT R2, H#10
    32: 5A 00 10;   --  SWI R2, H#0010  ; RAM(0x1000) <= R2
    35: F2 01;   --     MUL R2, R0, R1
    37: 6A 10;   --     OUT R2, H#10
    39: 52 00 10;   --  LWI R2, H#0010  ; R2 <= RAM(0x1000)
    3C: 6A 10;   --     OUT R2, H#10
    3E: 10 00;   --     JMP H#00

End;
```

```
TITLE    FPGA-CPU TEST PROGRAM
LINES    50                                    ; program length
LIST     F, B, W                               ; parameters for .LST
FORM     11111B111B1111B1111B11111111    ; format for .LST binary code output (5-3-4-4-8)

;*********************************
; MACROS
;*********************************

;*********************************
; CONSTANTS
;*********************************

;*********************************
; PROGRAM AREA
;*********************************
         ORG H#00
START:   MOVI R0, H#01        ; constant '1'
         IN R1, H#50          ; command code: 0=arithmetics, 1=SWR(Ra,Rb,Rd), 2=LWR&OUT(Ra,Rb)
         JZ R1, ARITH
         SUBA R1, R1, R0      ; R1 := R1 - 1
         JZ R1, SAVE

LOADP:   IN R0, H#50          ; Ra
         IN R1, H#50          ; Rb
         LWR R2, R0, R1       ; R2 = RAM(R0.R1)
         OUT R2, H#10         ; print R2
         JORG

SAVE:    IN R0, H#50
         IN R1, H#50
         IN R2, H#50          ; RAM(R0.R1) = R2
         SWR R2, R0, R1
         OUT R2, H#10         ; print R2
         JORG

ARITH:   IN R0, H#50
         IN R1, H#50

         ADDA R2, R0, R1
         OUT R2, H#10

         SUBA R2, R0, R1
         OUT R2, H#10

         AND R2, R0, R1
         OUT R2, H#10

         OR R2, R0, R1
         OUT R2, H#10

         SWI R2, H#0010       ; RAM(0x1000) <= R2

         MUL R2, R0, R1
         OUT R2, H#10

         LWI R2, H#0010       ; R2 <= RAM(0x1000)
         OUT R2, H#10

         JMP H#00

;*********************************
; DATA FOR TEST PROGRAM
;*********************************
END
```

```
TITLE     ASSEMBLY LANGUAGE DEFINITION FILE FOR FPGA-CPU
WORD      24                        ; 24 || 16 || 8 bits
WIDTH     72
LINES     50

;*********************************************************************
; INSTRUCTION OPCODE LABELS - 1-Bit prefix + 1-Hex opcode
;*********************************************************************
LL:           EQU   B#0             ; prefix 0 (for all commands except Rd = Ra * Rb)
LH:           EQU   B#1             ; prefix 1 (for Rd = Ra * Rb)
LSTOP:        EQU   4H#F
LJORG:        EQU   4H#0            ; jump to 0
LJZ:          EQU   4H#1            ; jump to A if Rd == 0
LJMP:         EQU   4H#2
LMOVR:        EQU   4H#3            ; Rd = Ra
LADD:         EQU   4H#4
LSUB:         EQU   4H#5
LOR:          EQU   4H#6
LAND:         EQU   4H#7
LLWR:         EQU   4H#8            ; Rd = RAM(Ra.Rb)
LSWR:         EQU   4H#9            ; RAM(Ra.Rb) = Rd
LLWI:         EQU   4H#A            ; Rd = RAM(A)
LSWI:         EQU   4H#B            ; RAM(A) = Rd
LIN:          EQU   4H#C
LOUT:         EQU   4H#D
LMOVI:        EQU   4H#E            ; Rd = DI
LMUL:         EQU   4H#E            ; Rd = Ra * Rb

;*********************************************************************
; INSTRUCTION FIELDS DEFINITIONS
;*********************************************************************
Rd:           SUB   3VB#000         ; destination register
Ra:           SUB   3VB#000         ; first source register
Rb:           SUB   3VB#000         ; second source register
DI8:          SUB   8VH#00; 8-bit immediate
A8:           SUB   8VH#00; 8-bit address
A16:          SUB   16VH#0000       ; 16-bit address
Z1:           EQU   B#0             ; const 0
Z3:           EQU   B#000           ; const 000
Z4:           EQU   4H#0            ; const 0000

;*********************************************************************
; DATA PSEUDO OPS
;*********************************************************************
DB:           DEF   8VH#00; 8-BIT DATA DIRECTIVE
;*********************************************************************
; ASSEMBLY LANGUAGE INSTRUCTIONS
;*********************************************************************
STOP:         DEF   LL,LSTOP,     Z3                    ; STOP
JORG:         DEF   LL,LJORG,     Z3                    ; jump to 0
JZ:           DEF   LL,LJZ, Rd,A8                 ; jump to A[7:0] if Rd == 0
JMP:          DEF   LL,LJMP,      Z3,A8                 ; jump to A[7:0]
MOVR:         DEF   LL,LMOVR,     Rd,Z1,Ra,Z4           ; Rd = Ra                move register
ADDA:         DEF   LL,LADD,      Rd,Z1,Ra,Z1,Rb        ; Rd = Ra + Rb
SUBA:         DEF   LL,LSUB,      Rd,Z1,Ra,Z1,Rb        ; Rd = Ra - Rb           cannot be 'SUB' due to keyword
OR:           DEF   LL,LOR,       Rd,Z1,Ra,Z1,Rb        ; Rd = Ra # Rb
AND:          DEF   LL,LAND,      Rd,Z1,Ra,Z1,Rb        ; Rd = Ra & Rb
LWR:          DEF   LL,LLWR,      Rd,Z1,Ra,Z1,Rb        ; Rd = RAM(Ra.Rb)        load word register addressing
SWR:          DEF   LL,LSWR,      Rd,Z1,Ra,Z1,Rb        ; RAM(Ra.Rb) = Rd        store word register addressing
LWI:          DEF   LL,LLWI,      Rd,A16                ; Rd = RAM(A[15:0])      load word immediate addressing
SWI:          DEF   LL,LSWI,      Rd,A16                ; RAM(A[15:0]) = Rd      store word immediate addressing
IN:           DEF   LL,LIN,       Rd,A8                 ; Rd = INP(A[7:0])       read from IO
OUT:          DEF   LL,LOUT,      Rd,A8                 ; OUT(A[7:0]) = Rd       write to IO
MOVI:         DEF   LL,LMOVI,     Rd,DI8                ; Rd = DI                move immediate
MUL:          DEF   LH,LMUL,      Rd,Z1,Ra,Z1,Rb        ; Rd = Ra * Rb
```

```
;*****************************************************************
;
; CPU's REGISTERS CODES DEFINITION
;*****************************************************************
;
R0:             EQU     B#000
R1:             EQU     B#001
R2:             EQU     B#010
R3:             EQU     B#011
R4:             EQU     B#100
R5:             EQU     B#101
R6:             EQU     B#110
R7:             EQU     B#111

END
```

Symulacja 7 - Poglądowa symulacja dzialania systemu.



```
1   --  Mikrokontroler
2
3   library ieee;
4   use ieee.std_logic_1164.all;
5   use ieee.std_logic_arith.all;
6   library lpm;
7   use lpm.lpm_components.all;
8   library altera;
9   use altera.altera_primitives_components.all;
10
11  entity Microcontroller is
12      generic (
13                  ext_io_addr : natural := 4609       -- pierwszy adres zewn. IO
14      );
15      port (      GEN     : in std_logic;
16
17                  -- wyjscie na szyne zewnetrzna (:TODO: przypisanie pinow)
18                  V_A     : out std_logic_vector(15 downto 0);
19                  VI_D    : inout std_logic_vector(7 downto 0);
20                  V_MRQ, V_IORQ, V_RD, V_WR : out std_logic;
21                  V_WT    : in std_logic;
22                  V_WTO   : out std_logic;                    -- wyjscie WT
23
24                  -- sygnaly WE / WY
25                  L_A     : out std_logic_vector(7 downto 0); -- diody gorny rzad
26                  L_B     : out std_logic_vector(7 downto 0); -- diody dolny rzad
27                  P_1     : out std_logic_vector(7 downto 0); -- port danych do LPT (SV1)
28                  P_2     : in std_logic_vector(7 downto 0);  -- port stanu z LPT (SV2)
29                  P_3     : out std_logic_vector(7 downto 0); -- port sterowania do LPT (SV3)
30                  P_4     : in std_logic_vector(7 downto 0);  -- port wejsciowy z PS/2 (SV4)
31                  --P_5       : out std_logic_vector(7 downto 0); -- port SV5 (nieuzywany)
32                  --P_7       : out std_logic_vector(7 downto 0); -- port SV7 (nieuzywany)
33                  --SW1B      : in std_logic;                 -- przelacznik SW1B, nieuzywany
34                  --SW2B      : in std_logic;                 -- przelacznik SW2B, nieuzywany
35                  SW3B    : in std_logic;                     -- przelacznik RESET: GND=>RES
36
37                  DX_REG_ACC_EN       :   out std_logic;                      -- Accumulator wr. en.
```

```
38                  DX_ADDR_REG_A       :   out std_logic_vector (2 downto 0);      -- Register A address
39                  DX_ADDR_REG_B       :   out std_logic_vector (2 downto 0);      -- Register B address
40                  DX_ADDR_REG_ACC     :   out std_logic_vector (2 downto 0);      -- Accumulator address
41                  DX_REG_A            :   out std_logic_vector (7 downto 0)
42
43                  --DX_AUT_CPU            :   out std_logic_vector (4 downto 0);
44                  --DX_REGADDRn           :   out std_logic_vector (15 downto 0)
45                  --DX_REG_ADDRESS        :   out std_logic_vector (15 downto 0)
46                  );
47
48  end entity Microcontroller;
49
50  architecture arch_Microcontroller of Microcontroller is
51
52      -- sygnaly szyny wewnetrznej
53      signal B_A          : std_logic_vector(15 downto 0);
54      signal B_D          : std_logic_vector(7 downto 0);
55      signal B_MRQ, B_IORQ, B_RD, B_WR, B_WT : std_logic;
56      signal RESET        : std_logic;
57
58      signal WAIT_CPU     : std_logic;
59
60      -- sygnal z PS2 sygnalizujacy ilosc wczytanych kodow
61      signal PS2_KEYNUM   : std_logic_vector(7 downto 0);
62
63      -- sygnal z LPT sygnalizujacy gotowosc drukarki
64      signal LPT_READY    : std_logic;
65
66      -- sygnal zezwalajacy na wejscie sygnalu V_WT z zewnatrz
67      signal EXTERN_IO_SEL: std_logic;
68
69      -- zatrzasniety stan szyny danych z ostatniej operacji
70      signal D_LATCH      : std_logic_vector (7 downto 0);
71
72      signal DX_DATA      : std_logic_vector (7 downto 0);
73
74      component CPU is
```

```
 75          port (
 76              GEN         :   in std_logic;                           -- Clock
 77              RESET       :   in std_logic;                           -- Reset
 78              ADDR        :   out std_logic_vector (15 downto 0);     -- Address bus
 79              DATA        :   inout std_logic_vector (7 downto 0);    -- Data bus
 80              MREQ        :   out std_logic;                          -- Memory request
 81              IORQ        :   out std_logic;                          -- I/O request
 82              WR          :   out std_logic;                          -- Write enable
 83              RD          :   out std_logic;                          -- Read enable
 84              WT          :   inout std_logic;                        -- Wait bus
 85              WAIT_CPU    :   out std_logic;                          -- Wait cpu
 86              D_REG_ACC_EN        :   out std_logic;                      -- Accumulator wr. en.
 87              D_ADDR_REG_A        :   out std_logic_vector (2 downto 0);  -- Register A address
 88              D_ADDR_REG_B        :   out std_logic_vector (2 downto 0);  -- Register B address
 89              D_ADDR_REG_ACC      :   out std_logic_vector (2 downto 0);  -- Accumulator address
 90              D_REG_A             :   out std_logic_vector (7 downto 0)
 91              --D_DATA        :   out std_logic_vector (7 downto 0)
 92              --D_REGADDRn           :   out std_logic_vector (15 downto 0)
 93              );
 94      end component CPU;
 95
 96      component ROM is
 97          port (  A   : in std_logic_vector (15 downto 0);
 98                  D   : inout std_logic_vector (7 downto 0);
 99              MRQ, RD : in std_logic );
100      end component ROM;
101
102      component RAM is
103          port (  A   : in std_logic_vector (15 downto 0);
104                  D   : inout std_logic_vector (7 downto 0);
105          MRQ, RD, WR : in std_logic );
106      end component RAM;
107
108      component LPT_OUT is
109          port (  GEN : in std_logic;
110                  RESET:in std_logic;
111                  A   : in std_logic_vector (7 downto 0);
```

```
112                 D   : in std_logic_vector (7 downto 0);
113                 IORQ: in std_logic;
114                 WR  : in std_logic;
115                 WT  : out std_logic;
116                 -- sygnal powiadamiajacy o gotowosci drukarki, aktywny '1'
117                 PRN_READY   : out std_logic;
118                 -- sygnaly do komunikacji z laczem LPT
119                 LCTRL       : out std_logic_vector (7 downto 0); -- SV2: nSI,nI,nAF,nS do LPT
120                 LSTAT       : in std_logic_vector (7 downto 0);  -- SV3: BS,PE,nF,S z LPT
121                 LDATASYN    : out std_logic_vector (7 downto 0)  -- SV5: dane do LPT
122             );
123     end component LPT_OUT;
124
125     component PS2_IN is
126         port (  GEN     : in std_logic; -- 20MHz clock
127                 RESET   : in std_logic; -- Reset signal from uC
128                 -- Bus
129                 ADDR    : in std_logic_vector (7 downto 0); -- Address bus
130                 DATA    : out std_logic_vector (7 downto 0);-- Data bus
131                 WT      : out std_logic;                    -- Wait signal
132                 IORQ    : in std_logic;                     -- I/O request signal
133                 RD      : in std_logic;                     -- Read signal
134                 -- PS/2 debug
135                 PS2_DEBUG_PORT: out std_logic_vector (7 downto 0);  -- Output port for debugging
136                 -- PS/2 connection port
137                 PDATA_IN: in std_logic_vector (7 downto 0)  -- PS/2 bus input port
138             );
139     end component PS2_IN;
140
141     begin
142         -- przepisanie stanu wewnetrznej szyny kontrolera na wyjscie
143         V_IORQ <= B_IORQ;
144         V_MRQ <= B_MRQ;
145         V_RD <= B_RD;
146         V_WR <= B_WR;
147         V_A <= B_A;
148
```

```
149           -- sygnal resetujacy z przelacznika 3
150           RESET       <= SW3B;
151
152           -- sygnalizacja na diodach
153           L_A(7 downto 2) <= (others => '0');
154           L_A(1 downto 0) <= PS2_KEYNUM(7 downto 6);
155           L_B(7 downto 3) <= (others => '0');
156           L_B(2 downto 0) <= PS2_KEYNUM(2 downto 0);
157
158       -- zatrzasniecie stanu linii danych przy kazdej operacji IO / MEM (debug)
159       dl: process (VI_D, B_MRQ, B_IORQ, D_LATCH) is
160           begin
161               if ((B_MRQ = '0') OR (B_IORQ = '0')) then
162                   D_LATCH(7 downto 0) <= VI_D(7 downto 0);
163               else
164                   D_LATCH(7 downto 0) <= D_LATCH(7 downto 0);
165               end if;
166           end process;
167
168       -- wybranie wejscia WT z zewnetrznego IO
169       extern_io: process (B_IORQ, B_A) is
170           begin
171               if ((B_IORQ = '0') AND (unsigned(B_A) >= ext_io_addr)) then
172                   EXTERN_IO_SEL <= '1';
173               else
174                   EXTERN_IO_SEL <= '0';
175               end if;
176           end process;
177
178       -- wpuszczenie zewnetrznego sygnalu WAIT na szyne przy wyborze urzadzenia IO
179       t1: TRI port map (V_WT, EXTERN_IO_SEL, B_WT);
180
181       -- przepisanie wewnetrznego sygnalu WAIT na wyjscie
182       V_WTO <= B_WT;
183
184       e9: CPU port map (GEN, RESET, B_A, VI_D, B_MRQ, B_IORQ, B_WR, B_RD, B_WT, WAIT_CPU, DX_REG_ACC_EN,
      DX_ADDR_REG_A,DX_ADDR_REG_B,DX_ADDR_REG_ACC,DX_REG_A);
```

```
185
186       e0: ROM port map (B_A, VI_D, B_MRQ, B_RD);
187
188       e1: RAM port map (B_A, VI_D, B_MRQ, B_RD, B_WR);
189
190       e2: LPT_OUT port map (GEN, RESET, B_A(7 downto 0), VI_D, B_IORQ, B_WR, B_WT,LPT_READY, P_3, P_2, P_1);
191
192       e3: PS2_IN port map (GEN, RESET, B_A(7 downto 0), VI_D, B_WT, B_IORQ, B_RD, PS2_KEYNUM, P_4);
193
194   end architecture arch_Microcontroller;
```

```
 1   library ieee;
 2   use ieee.std_logic_1164.all;
 3   use ieee.std_logic_arith.all;
 4   library lpm;
 5   use lpm.lpm_components.all;
 6   library altera;
 7   use altera.altera_primitives_components.all;
 8
 9   entity CPU is
10       port (
11               GEN          :   in std_logic;                          -- Clock
12               RESET        :   in std_logic;                          -- Reset
13               ADDR         :   out std_logic_vector (15 downto 0);    -- Address bus
14               DATA         :   inout std_logic_vector (7 downto 0);   -- Data bus
15               MREQ         :   out std_logic;                         -- Memory request
16               IORQ         :   out std_logic;                         -- I/O request
17               WR           :   out std_logic;                         -- Write enable
18               RD           :   out std_logic;                         -- Read enable
19               WT           :   in std_logic;                          -- Wait bus
20               WAIT_CPU     :   out std_logic;                         -- Wait cpu
21
22               -- debug
23               D_REG_ACC_EN        :   out std_logic;                          -- Accumulator wr. en.
24               D_ADDR_REG_A        :   out std_logic_vector (2 downto 0);      -- Register A address
25               D_ADDR_REG_B        :   out std_logic_vector (2 downto 0);      -- Register B address
26               D_ADDR_REG_ACC      :   out std_logic_vector (2 downto 0);      -- Accumulator address
27               D_REG_A             :   out std_logic_vector (7 downto 0);
28
29               D_DATA              :   out std_logic_vector (7 downto 0);
30               D_REGADDRn          :   out std_logic_vector (15 downto 0);
31               D_REG_ADDRESS       :   out std_logic_vector (15 downto 0);
32               D_AUT_CPU           :   out std_logic_vector (4 downto 0);
33               D_AUT_BUS           :   out std_logic_vector (2 downto 0);
34               D_PC                :   out std_logic_vector (15 downto 0)
35
36               --D_REG_PC          :   out std_logic_vector (15 downto 0);
37               --D_REG_PCn         :   out std_logic_vector (15 downto 0);
```

```
38              --D_REG_PCx         :   out std_logic_vector (15 downto 0);
39              --D_REG_CMD         :   out std_logic_vector (23 downto 0);
40              --D_REG_CMDn            :   out std_logic_vector (23 downto 0)
41          );
42  end entity CPU;
43
44  architecture CPU_module of CPU is
45      -- Signals
46      signal SIG_DATA         :   std_logic_vector (7 downto 0);      -- Tri-state data output
47      signal SIG_IS_IO        :   std_logic;                          -- Is I/O request
48      signal SIG_IS_WR        :   std_logic;                          -- Is write
49      signal SIG_WAIT         :   std_logic;                          -- Tri-state wait output
50      signal SIG_BUS_START    :   std_logic;                          -- Start bus communication
51      signal SIG_BUS_STOP     :   std_logic;                          -- Stop bus communication
52      -- System registers
53      signal REG_ADDRESS      :   std_logic_vector (15 downto 0);     -- Current PC address
54      signal REG_PC           :   std_logic_vector (15 downto 0);     -- Program counter
55      signal REG_PCn          :   std_logic_vector (15 downto 0);     -- Next program counter
56      signal REG_PCx          :   std_logic_vector (15 downto 0);     -- Intermediate PC
57      signal REG_CMD          :   std_logic_vector (23 downto 0);     -- Current command
58      signal REG_CMDn         :   std_logic_vector (23 downto 0);     -- Next current command
59      -- Bus
60      signal REG_ADDR         :   std_logic_vector (15 downto 0);     -- Bus address
61      signal REG_ADDRn        :   std_logic_vector (15 downto 0);     -- Next bus address
62      signal REG_DATA         :   std_logic_vector (7 downto 0);      -- Bus data
63      signal REG_DATAn        :   std_logic_vector (7 downto 0);      -- Next bus data
64      signal SIG_MREQ         :   std_logic;                          -- Bus memory request
65      signal SIG_MREQn        :   std_logic;                          -- Next bus mem. req.
66      signal SIG_IORQ         :   std_logic;                          -- Bus I/O request
67      signal SIG_IORQn        :   std_logic;                          -- Next bus I/O req.
68      signal SIG_RD           :   std_logic;                          -- Bus read enable
69      signal SIG_RDn          :   std_logic;                          -- Next bus read en.
70      signal SIG_WR           :   std_logic;                          -- Bus write enable
71      signal SIG_WRn          :   std_logic;                          -- Next bus write en.
72      signal SIG_DOUT         :   std_logic;                          -- Enables DATA output to bus
73      signal SIG_DOUTn        :   std_logic;
74      -- State machines
```

```
 75       -- Main CPU state machine ( instruction fetch etc. )
 76       signal AUT_CPU          :   std_logic_vector (4 downto 0);
 77       signal AUT_CPUn         :   std_logic_vector (4 downto 0);
 78       ---- AUT_CPU states
 79       constant AUT_CPU_FETCH          :   std_logic_vector (4 downto 0) := "00000";
 80       constant AUT_CPU_CMD            :   std_logic_vector (4 downto 0) := "00001";
 81       constant AUT_CPU_CMD1           :   std_logic_vector (4 downto 0) := "00010";
 82       constant AUT_CPU_CMD2           :   std_logic_vector (4 downto 0) := "00011";
 83       constant AUT_CPU_CMD3           :   std_logic_vector (4 downto 0) := "00100";
 84       constant AUT_CPU_CMD4           :   std_logic_vector (4 downto 0) := "00101";
 85       constant AUT_CPU_CMD5           :   std_logic_vector (4 downto 0) := "00110";
 86       constant AUT_CPU_CMD6           :   std_logic_vector (4 downto 0) := "00111";
 87       constant AUT_CPU_EXE            :   std_logic_vector (4 downto 0) := "01000";
 88       constant AUT_CPU_RAM_WR1        :   std_logic_vector (4 downto 0) := "01001";
 89       constant AUT_CPU_RAM_WR2        :   std_logic_vector (4 downto 0) := "01010";
 90
 91       constant AUT_CPU_RAM_WR3        :   std_logic_vector (4 downto 0) := "11000";
 92       constant AUT_CPU_RAM_WR4        :   std_logic_vector (4 downto 0) := "11001";
 93
 94       constant AUT_CPU_OUT1           :   std_logic_vector (4 downto 0) := "01100";
 95       constant AUT_CPU_OUT2           :   std_logic_vector (4 downto 0) := "01101";
 96
 97       constant AUT_CPU_IN1            :   std_logic_vector (4 downto 0) := "01111";
 98       constant AUT_CPU_IN2            :   std_logic_vector (4 downto 0) := "10000";
 99       constant AUT_CPU_IN3            :   std_logic_vector (4 downto 0) := "10001";
100
101       constant AUT_CPU_RAM_RD1        :   std_logic_vector (4 downto 0) := "10011";
102       constant AUT_CPU_RAM_RD2        :   std_logic_vector (4 downto 0) := "10100";
103       constant AUT_CPU_MUL            :   std_logic_vector (4 downto 0) := "10101";
104
105       constant AUT_CPU_RAM_RD3        :   std_logic_vector (4 downto 0) := "11010";
106       constant AUT_CPU_RAM_RD4        :   std_logic_vector (4 downto 0) := "11011";
107
108       constant AUT_CPU_FETCH2         :   std_logic_vector (4 downto 0) := "11111";
109
110       -- Bus communication state machine ( mreq, iorq, rd, wr etc. )
111       signal AUT_BUS          :   std_logic_vector (2 downto 0);
```

```
112       signal AUT_BUSn         :   std_logic_vector (2 downto 0);
113       ---- AUT_BUS states
114       constant AUT_BUS_START      :   std_logic_vector (2 downto 0) := "000";
115       constant AUT_BUS_CYCLE1     :   std_logic_vector (2 downto 0) := "001";
116       constant AUT_BUS_CYCLE2     :   std_logic_vector (2 downto 0) := "010";
117       constant AUT_BUS_STOP       :   std_logic_vector (2 downto 0) := "011";
118       constant AUT_BUS_WAIT       :   std_logic_vector (2 downto 0) := "100";
119       constant AUT_BUS_WAIT2      :   std_logic_vector (2 downto 0) := "101";
120       constant AUT_BUS_WAIT3      :   std_logic_vector (2 downto 0) := "110";
121       -- Universal registers
122       signal REG_ACC_EN       :   std_logic;                          -- Accumulator wr. en.
123       signal REG_ACC_ENn      :   std_logic;
124       signal ADDR_REG_A       :   std_logic_vector (2 downto 0);      -- Register A address
125       signal ADDR_REG_B       :   std_logic_vector (2 downto 0);      -- Register B address
126       signal ADDR_REG_ACC     :   std_logic_vector (2 downto 0);      -- Accumulator address
127       signal REG_A            :   std_logic_vector (7 downto 0);      -- Register A
128       signal REG_B            :   std_logic_vector (7 downto 0);      -- Register B
129       signal REG_ACC          :   std_logic_vector (7 downto 0);      -- Accumulator Register
130       signal REG_ACCn         :   std_logic_vector (7 downto 0);      -- Accumulator Register
131       signal REG_ACCx         :   std_logic_vector (7 downto 0);      -- Accumulator Reg. inter.
132       signal MUL_RES          :   std_logic_vector (7 downto 0);      -- Multiplication result
133       signal MUL_GO           :   std_logic;                          -- 0 => starts MUL operation
134       signal MUL_READY        :   std_logic;                          -- 0 => MUL result ready
135       -- RegA o RegB
136       signal REG_A2           :   std_logic_vector (7 downto 0);      -- Register A2
137       signal REG_B2           :   std_logic_vector (7 downto 0);      -- Register B2
138
139       component MUL is
140           port (
141               GEN     : in std_logic;
142               RESET   : in std_logic;
143               A       : in std_logic_vector (7 downto 0);
144               B       : in std_logic_vector (7 downto 0);
145               RESULT  : out std_logic_vector (7 downto 0);-- wynik mnozenia
146               GO      : in std_logic;     -- uruchamia MUL, aktywne LOW
147               READY   : out std_logic     -- czy wynik gotowy? (LOW)
148           );
```

```
149       end component MUL;
150
151       begin
152
153       AUT_CPU_PROC:
154           process (
155                       AUT_CPU,
156                       SIG_BUS_STOP,
157                       REG_DATA,
158                       REG_CMD,
159                       REG_PC,
160                       REG_PCx,
161                       SIG_DATA,
162                       REG_A,
163                       REG_B,
164                       REG_A2,
165                       REG_B2,
166                       REG_ACCx,
167                       REG_ACC,
168                       ADDR_REG_ACC,
169                       MUL_READY,
170                       MUL_RES
171                       --
172                       --
173                       --
174                   ) is
175
176                   begin
177                       REG_DATAn<=REG_DATA;
178                       REG_CMDn<=REG_CMD;
179                       REG_PCn<=REG_PC;
180                       REG_ACC_ENn<='0';
181                       REG_ACCn<=REG_ACC;
182                       --
183                       REG_ADDRESS<=REG_PC;
184
185                       ADDR_REG_ACC <= REG_CMD(18 downto 16);
```

```
186                     ADDR_REG_A <= REG_CMD(14 downto 12);
187                     ADDR_REG_B <= REG_CMD(10 downto 8);
188                     --
189                     --
190                     SIG_BUS_START<='0';
191                     SIG_IS_IO<='0';
192                     SIG_IS_WR<='0';
193                     REG_PCx<=unsigned(REG_PC)+1;
194                     --
195                     MUL_GO <= '1';
196
197                     case AUT_CPU is
198                         when AUT_CPU_FETCH =>
199                             --
200                             --SIG_BUS_START<='1';
201                             --REG_CMDn<=(others => '0');
202
203                             AUT_CPUn<=AUT_CPU_FETCH2;
204
205                         when AUT_CPU_FETCH2 =>
206                             --
207                             SIG_BUS_START<='1';
208                             --REG_CMDn<=(others => '0');
209
210                             AUT_CPUn<=AUT_CPU_CMD;
211
212                         when AUT_CPU_CMD =>
213                             --
214                             REG_CMDn(23 downto 16)<=SIG_DATA;
215                             if SIG_BUS_STOP='0'then
216                                 AUT_CPUn<=AUT_CPU_CMD;
217                             else
218                                 AUT_CPUn<=AUT_CPU_CMD1;
219                             end if;
220
221                         when AUT_CPU_CMD1 =>
222                             REG_PCn<=std_logic_vector(REG_PCx);
```

```
223                         --
224                         -- Rozkazy 1-bajtowe
225                         if  (REG_CMD(23 downto 19)="00000") or
226                             (REG_CMD(23 downto 19)="01111") then
227                             AUT_CPUn<=AUT_CPU_EXE;
228                         else
229                             AUT_CPUn<=AUT_CPU_CMD2;
230                         end if;
231                         --
232                     when AUT_CPU_CMD2 =>
233                         AUT_CPUn<=AUT_CPU_CMD3;
234
235                     -- Rozkazy 2-bajtowe
236                     when AUT_CPU_CMD3 =>
237                         SIG_BUS_START<='1';
238                         REG_CMDn(15 downto 8)<=SIG_DATA;
239                         if SIG_BUS_STOP='0' then
240                             AUT_CPUn<=AUT_CPU_CMD3;
241                         else
242                             AUT_CPUn<=AUT_CPU_CMD4;
243                         end if;
244
245                     when AUT_CPU_CMD4 =>
246                         REG_PCn<=std_logic_vector(REG_PCx);
247                         --
248                         -- Rozkazy 3-bajtowe
249                         if  (REG_CMD(23 downto 19)="01010") or
250                             (REG_CMD(23 downto 19)="01011") then
251                             AUT_CPUn<=AUT_CPU_CMD5;
252                         else
253                             REG_A2<=REG_A;
254                             REG_B2<=REG_B;
255                             AUT_CPUn<=AUT_CPU_EXE;
256                         end if;
257
258                     when AUT_CPU_CMD5 =>
259                         AUT_CPUn<=AUT_CPU_CMD6;
```

```
260
261                         when AUT_CPU_CMD6 =>
262                             SIG_BUS_START<='1';
263                             REG_CMDn(7 downto 0)<=SIG_DATA;
264                             if SIG_BUS_STOP='0' then
265                                 AUT_CPUn<=AUT_CPU_CMD6;
266                             else
267                                 REG_PCn<=std_logic_vector(REG_PCx);
268                                 AUT_CPUn<=AUT_CPU_EXE;
269                             end if;
270
271                         -- 1 byte: 00000,01111
272                         -- 2 byte: 00001,00010,00011,00100,00101,00110,00111,01000,01001
273                         --         01100,01101,01110,11110
274                         -- 3 byte: 01010,01011
275                         when AUT_CPU_EXE =>
276                             --
277
278                             -- [00] JMP 0 (1 byte)
279                             if      REG_CMD(23 downto 19)="00000" then
280                                     REG_PCn<=(others => '0');
281
282                                     AUT_CPUn<=AUT_CPU_FETCH;
283
284                             -- [01] JMP Rd=0,A (2 byte)
285                             elsif   REG_CMD(23 downto 19)="00001" then
286                                     if signed(REG_ACC) = 0 then
287                                         REG_PCn(15 downto 8)<="00000000";
288                                         REG_PCn(7 downto 0)<=REG_CMD(15 downto 8);
289                                     end if;
290
291                                     AUT_CPUn<=AUT_CPU_FETCH;
292
293                             -- [02] JMP A (2 byte)
294                             elsif   REG_CMD(23 downto 19)="00010" then
295                                     REG_PCn(15 downto 8)<="00000000";
296                                     REG_PCn(7 downto 0)<=REG_CMD(15 downto 8);
```

```
297
298                         AUT_CPUn<=AUT_CPU_FETCH;
299
300                     -- [03] Rd<=Ra (2 byte)
301                     elsif   REG_CMD(23 downto 19)="00011" then
302                         REG_ACCn<=REG_A;
303                         REG_ACC_ENn<='1';
304
305                         AUT_CPUn<=AUT_CPU_FETCH;
306
307                     -- [04] Rd<=Ra+Rb (2 byte)
308                     elsif   REG_CMD(23 downto 19)="00100" then
309                         REG_ACCx<=signed(REG_A)+signed(REG_B);
310                         REG_ACCn<=std_logic_vector(REG_ACCx);
311                         REG_ACC_ENn<='1';
312
313                         AUT_CPUn<=AUT_CPU_FETCH;
314
315                     -- [05] Rd<=Ra-Rb (2 byte)
316                     elsif   REG_CMD(23 downto 19)="00101" then
317                         REG_ACCx<=signed(REG_A)-signed(REG_B);
318                         REG_ACCn<=std_logic_vector(REG_ACCx);
319                         REG_ACC_ENn<='1';
320
321                         AUT_CPUn<=AUT_CPU_FETCH;
322
323                     -- [06] Rd<=Ra#Rb (2 byte)
324                     elsif   REG_CMD(23 downto 19)="00110" then
325                         REG_ACCn<=REG_A or REG_B;
326                         REG_ACC_ENn<='1';
327
328                         AUT_CPUn<=AUT_CPU_FETCH;
329
330                     -- [07] Rd<=Ra&Rb (2 byte)
331                     elsif   REG_CMD(23 downto 19)="00111" then
332                         REG_ACCn<=REG_A and REG_B;
333                         REG_ACC_ENn<='1';
```

```
334
335                         AUT_CPUn<=AUT_CPU_FETCH;
336
337                     -- [08] Rd<=RAM(RaoRb) (2 byte)
338                     elsif   REG_CMD(23 downto 19)="01000" then
339                         REG_ADDRESS(15 downto 8)<=REG_A(7 downto 0);
340                         REG_ADDRESS(7 downto 0)<=REG_B(7 downto 0);
341
342
343
344                         AUT_CPUn<=AUT_CPU_RAM_RD3;
345
346                     -- [09] Rd=>RAM(RaoRb) (2 byte)
347                     elsif   REG_CMD(23 downto 19)="01001" then
348                     -- bedzie problem, bo chcemy czytac naraz z trzech rejestrow
349                     -- musimy spamietac na boku albo RaoRb albo Rd
350                     -- Rozwiązane: dodatkowe rejestry REG_A2 i REG_B2
351                         REG_ADDRESS(15 downto 8)<=REG_A2(7 downto 0);
352                         REG_ADDRESS(7 downto 0)<=REG_B2(7 downto 0);
353
354                         ADDR_REG_A <= ADDR_REG_ACC; -- Rd becomes source
355
356                         AUT_CPUn<=AUT_CPU_RAM_WR3;
357
358                     -- [0A] Rd<=RAM(A) (3 byte)
359                     elsif   REG_CMD(23 downto 19)="01010" then
360                         REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
361                         REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
362
363                         AUT_CPUn<=AUT_CPU_RAM_RD1;
364
365                     -- [0B] Rd=>RAM(A) (3 byte)
366                     elsif   REG_CMD(23 downto 19)="01011" then
367                         REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
368                         REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
369                         ADDR_REG_A <= ADDR_REG_ACC; -- Rd becomes source
370
```

```
371                     AUT_CPUn<=AUT_CPU_RAM_WR1;
372
373                 -- [0C] Rd<=INP(A) (2 byte)
374                 elsif   REG_CMD(23 downto 19)="01100" then
375                     REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
376
377                     AUT_CPUn<=AUT_CPU_IN1;
378
379                 -- [0D] Rd=>OUT(A) (2 byte)
380                 elsif   REG_CMD(23 downto 19)="01101" then
381                     --REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
382                     REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
383                     ADDR_REG_A <= ADDR_REG_ACC; -- Rd becomes source
384
385                     AUT_CPUn<=AUT_CPU_OUT1;
386
387                 -- [0E] Rd<=DI (2 byte)
388                 elsif   REG_CMD(23 downto 19)="01110" then
389                     REG_ACCn<=REG_CMD(15 downto 8);
390                     REG_ACC_ENn<='1';
391
392                     AUT_CPUn<=AUT_CPU_FETCH;
393
394                 -- [0F] STOP (1 byte)
395                 elsif   REG_CMD(23 downto 19)="01111" then
396
397                     AUT_CPUn<=AUT_CPU_EXE;
398
399                 -- [1E] Rd<=Ra op Rb (2 byte) op == multiply
400                 elsif   REG_CMD(23 downto 19)="11110" then
401                     MUL_GO <= '0';  -- start multiplication
402                     AUT_CPUn<=AUT_CPU_MUL;
403
404                 --
405                 -- Command unknown
406                 else
407                     AUT_CPUn<=AUT_CPU_FETCH;
```

```
408                         end if;
409
410                     when AUT_CPU_RAM_WR1 =>
411                         REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
412                         REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
413                         SIG_IS_WR<='1';
414
415                         ADDR_REG_A <= ADDR_REG_ACC;
416                         REG_DATAn<=REG_A;
417
418                         SIG_BUS_START<='1';
419                         AUT_CPUn<=AUT_CPU_RAM_WR2;
420
421                     when AUT_CPU_RAM_WR2 =>
422                         REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
423                         REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
424                         SIG_IS_WR<='1';
425
426                         ADDR_REG_A <= ADDR_REG_ACC;
427                         REG_DATAn<=REG_A;
428
429                         if SIG_BUS_STOP='0' then
430                             AUT_CPUn<=AUT_CPU_RAM_WR2;
431                         else
432                             AUT_CPUn<=AUT_CPU_FETCH;
433                         end if;
434
435                     when AUT_CPU_RAM_WR3 =>
436                         REG_ADDRESS(15 downto 8)<=REG_A2(7 downto 0);
437                         REG_ADDRESS(7 downto 0)<=REG_B2(7 downto 0);
438                         SIG_IS_WR<='1';
439
440                         ADDR_REG_A <= ADDR_REG_ACC;
441                         REG_DATAn<=REG_A;
442
443                         SIG_BUS_START<='1';
444                         AUT_CPUn<=AUT_CPU_RAM_WR4;
```

```
445
446                         when AUT_CPU_RAM_WR4 =>
447                             REG_ADDRESS(15 downto 8)<=REG_A2(7 downto 0);
448                             REG_ADDRESS(7 downto 0)<=REG_B2(7 downto 0);
449                             SIG_IS_WR<='1';
450
451                             ADDR_REG_A <= ADDR_REG_ACC;
452                             REG_DATAn<=REG_A;
453
454                             if SIG_BUS_STOP='0' then
455                                 AUT_CPUn<=AUT_CPU_RAM_WR4;
456                             else
457                                 AUT_CPUn<=AUT_CPU_FETCH;
458                             end if;
459
460
461                         when AUT_CPU_RAM_RD1 =>
462                             REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
463                             REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
464                             SIG_IS_WR<='0';
465
466                             SIG_BUS_START<='1';
467                             AUT_CPUn<=AUT_CPU_RAM_RD2;
468
469                         when AUT_CPU_RAM_RD2 =>
470                             REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
471                             REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
472                             SIG_IS_WR<='0';
473                             REG_DATAn<=SIG_DATA;
474                             --?
475                             if SIG_BUS_STOP='0' then
476                                 AUT_CPUn<=AUT_CPU_RAM_RD2;
477                             else
478                                 REG_ACCn<=SIG_DATA;
479                                 REG_ACC_ENn<='1';   -- now write data to Rd
480                                 AUT_CPUn<=AUT_CPU_FETCH;
481                             end if;
```

```
482
483                         when AUT_CPU_RAM_RD3 =>
484                             REG_ADDRESS(15 downto 8)<=REG_A(7 downto 0);
485                             REG_ADDRESS(7 downto 0)<=REG_B(7 downto 0);
486                             SIG_IS_WR<='0';
487
488                             SIG_BUS_START<='1';
489                             AUT_CPUn<=AUT_CPU_RAM_RD4;
490
491                         when AUT_CPU_RAM_RD4 =>
492                             REG_ADDRESS(15 downto 8)<=REG_A(7 downto 0);
493                             REG_ADDRESS(7 downto 0)<=REG_B(7 downto 0);
494                             SIG_IS_WR<='0';
495                             REG_DATAn<=SIG_DATA;
496                             --?
497                             if SIG_BUS_STOP='0' then
498                                 AUT_CPUn<=AUT_CPU_RAM_RD4;
499                             else
500                                 REG_ACCn<=SIG_DATA;
501                                 REG_ACC_ENn<='1';   -- now write data to Rd
502                                 AUT_CPUn<=AUT_CPU_FETCH;
503                             end if;
504
505                         when AUT_CPU_OUT1 =>
506                             REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
507                             ADDR_REG_A <= ADDR_REG_ACC; -- Rd becomes source
508                             SIG_IS_WR<='1';
509                             SIG_IS_IO<='1';
510
511                             REG_DATAn<=REG_A;
512                             SIG_BUS_START<='1';
513                             AUT_CPUn<=AUT_CPU_OUT2;
514
515                         when AUT_CPU_OUT2 =>
516                             REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
517                             ADDR_REG_A <= ADDR_REG_ACC; -- Rd becomes source
518                             SIG_IS_WR<='1';
```

```
519                     SIG_IS_IO<='1';
520
521                     REG_DATAn<=REG_A;
522                     if SIG_BUS_STOP='0' then
523                         AUT_CPUn<=AUT_CPU_OUT2;
524                     else
525                         AUT_CPUn<=AUT_CPU_FETCH;
526                     end if;
527
528                 when AUT_CPU_IN1 =>
529                     --REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
530                     REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
531                     SIG_IS_WR<='0';
532                     SIG_IS_IO<='1';
533
534                     SIG_BUS_START<='1';
535                     --SIG_BUS_START<='1';
536                     AUT_CPUn<=AUT_CPU_IN2;
537
538                 when AUT_CPU_IN2 =>
539                     --REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
540                     REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
541                     SIG_IS_WR<='0';
542                     SIG_IS_IO<='1';
543
544                     REG_DATAn<=SIG_DATA;
545                     --?
546                     if SIG_BUS_STOP='0' then
547                         AUT_CPUn<=AUT_CPU_IN2;
548                     else
549                         AUT_CPUn<=AUT_CPU_IN3;
550                     end if;
551
552                 when AUT_CPU_IN3 =>
553                     --REG_ADDRESS(15 downto 8)<=REG_CMD(7 downto 0);
554                     REG_ADDRESS(7 downto 0)<=REG_CMD(15 downto 8);
555                     SIG_IS_WR<='0';
```

```
556                             SIG_IS_IO<='1';
557
558                             REG_DATAn<=SIG_DATA;
559                             REG_ACCn<=REG_DATA;
560                             REG_ACC_ENn<='1';
561
562                             AUT_CPUn<=AUT_CPU_FETCH;
563
564                             MUL_GO <= '0';  -- start multiplication
565
566                         when AUT_CPU_MUL =>
567                             if MUL_READY = '0' then
568                                 REG_ACCn <= MUL_RES;
569                                 REG_ACC_ENn <= '1';
570                                 AUT_CPUn <= AUT_CPU_FETCH;
571                             else
572                                 AUT_CPUn <= AUT_CPU_MUL;
573                             end if;
574
575                         when others =>
576                             AUT_CPUn<=AUT_CPU;
577
578                     end case;
579                 end process AUT_CPU_PROC;
580
581     AUT_BUS_PROC:
582         process (
583                     AUT_BUS,
584                     SIG_BUS_START,
585                     REG_ADDR,
586                     SIG_WAIT,
587                     REG_ADDRESS,
588                     SIG_IS_IO,
589                     SIG_IS_WR,
590                     SIG_MREQ,
591                     SIG_IORQ,
592                     SIG_RD,
```

```
593                     SIG_WR,
594                     SIG_DOUT
595                 ) is
596
597             begin
598
599                 SIG_MREQn<='1';
600                 SIG_IORQn<='1';
601                 SIG_RDn<='1';
602                 SIG_WRn<='1';
603                 SIG_DOUTn<='1';
604                 SIG_BUS_STOP<='0';
605                 REG_ADDRn<=REG_ADDR;
606
607                 if SIG_WAIT='1' then
608                     WAIT_CPU<='0';
609                 else
610                     WAIT_CPU<='1';
611                 end if;
612
613                 case AUT_BUS is
614
615                     when AUT_BUS_START =>
616                         if SIG_BUS_START='0' then
617                             REG_ADDRn<=REG_ADDRESS;
618                             AUT_BUSn<=AUT_BUS_START;
619                         else
620                             AUT_BUSn<=AUT_BUS_CYCLE1;
621                         if SIG_IS_WR='1' then
622                             SIG_DOUTn<='0';
623                         end if;
624                     end if;
625
626                     when AUT_BUS_CYCLE1 =>
627                         SIG_DOUTn <= SIG_DOUT;
628                         if SIG_IS_IO='0' then
629                             SIG_MREQn<='0';
```

```
630                         else
631                             SIG_IORQn<='0';
632                         end if;
633
634                         if SIG_IS_WR='0' then
635                             SIG_RDn<='0';
636                         else
637                             SIG_WRn<='0';
638                         end if;
639                         AUT_BUSn<=AUT_BUS_CYCLE2;
640
641                     when AUT_BUS_CYCLE2 =>
642                         SIG_DOUTn <= SIG_DOUT;
643                         SIG_MREQn<=SIG_MREQ;
644                         SIG_IORQn<=SIG_IORQ;
645                         SIG_RDn<=SIG_RD;
646                         SIG_WRn<=SIG_WR;
647                         AUT_BUSn<=AUT_BUS_WAIT;
648
649                     when AUT_BUS_WAIT =>
650                         SIG_DOUTn <= SIG_DOUT;
651                         SIG_MREQn<=SIG_MREQ;
652                         SIG_IORQn<=SIG_IORQ;
653                         SIG_RDn<=SIG_RD;
654                         SIG_WRn<=SIG_WR;
655                         if SIG_WAIT='0' then
656                             AUT_BUSn<=AUT_BUS_WAIT;
657                         else
658                             if SIG_IS_IO = '1' then
659                                 AUT_BUSn<=AUT_BUS_WAIT2;
660                             else
661                                 AUT_BUSn<=AUT_BUS_STOP;
662                             end if;
663                         end if;
664
665                     when AUT_BUS_WAIT2 => -- for IO only
666                         SIG_DOUTn <= SIG_DOUT;
```

```
667                             SIG_MREQn<=SIG_MREQ;
668                             SIG_IORQn<=SIG_IORQ;
669                             SIG_RDn<=SIG_RD;
670                             SIG_WRn<=SIG_WR;
671                             if SIG_WAIT='0' then
672                                 AUT_BUSn<=AUT_BUS_WAIT2;
673                             else
674                                 AUT_BUSn<=AUT_BUS_STOP;
675                             end if;
676
677                         when AUT_BUS_STOP =>
678                             REG_ADDRn<=REG_ADDRESS;
679                             SIG_BUS_STOP<='1';
680
681                             if SIG_BUS_START='0' then
682                                 AUT_BUSn<=AUT_BUS_START;
683                             else
684                                 AUT_BUSn<=AUT_BUS_STOP;
685                             end if;
686
687                         when others =>
688                             REG_ADDRn<=REG_ADDRESS;
689                             SIG_BUS_STOP<='1';
690
691                             if SIG_BUS_START='0' then
692                                 AUT_BUSn<=AUT_BUS_START;
693                             else
694                                 AUT_BUSn<=AUT_BUS_STOP;
695                             end if;
696
697                     end case;
698                 end process AUT_BUS_PROC;
699
700         ADDR<=REG_ADDR;
701         MREQ<=SIG_MREQ;
702         IORQ<=SIG_IORQ;
703         RD<=SIG_RD;
```

```
704       WR<=SIG_WR;
705
706       D_REGADDRn <= REG_ADDRn;
707       D_REG_ADDRESS <= REG_ADDRESS;
708
709       mul0: MUL port map (GEN, RESET, REG_A, REG_B, MUL_RES, MUL_GO, MUL_READY);
710
711       e0: lpm_bustri
712           generic map
713           (
714               LPM_WIDTH =>8
715           )
716           port map
717           (
718               data=>REG_DATA,
719               result=>SIG_DATA,
720               tridata=>DATA,
721               enabledt=>not SIG_DOUT,
722               enabletr=>not SIG_RD
723           );
724
725   --  e1: tri
726   --      port map
727   --      (
728   --          a_in=>WT,
729   --          oe=>'1',
730   --          a_out=>SIG_WAIT
731   --      );
732   --
733       SIG_WAIT <= WT;
734
735       uni_reg_A: lpm_ram_dp
736           generic map
737           (
738               LPM_WIDTH=>8,
739               LPM_WIDTHAD=>3,
740               LPM_NUMWORDS=>8,
```

```
741             LPM_INDATA=>"REGISTERED",
742             LPM_OUTDATA=>"UNREGISTERED",
743             LPM_RDADDRESS_CONTROL=>"UNREGISTERED",
744             LPM_WRADDRESS_CONTROL=>"UNREGISTERED"
745         )
746         port map
747         (
748             rdaddress=> ADDR_REG_A,
749             wraddress=> ADDR_REG_ACC,
750             wren=> REG_ACC_EN,
751             wrclock=> GEN,
752             data=> REG_ACC,
753             q=> REG_A
754         );
755
756     uni_reg_B: lpm_ram_dp
757         generic map
758         (
759             LPM_WIDTH=>8,
760             LPM_WIDTHAD=>3,
761             LPM_NUMWORDS=>8,
762             LPM_INDATA=>"REGISTERED",
763             LPM_OUTDATA=>"UNREGISTERED",
764             LPM_RDADDRESS_CONTROL=>"UNREGISTERED",
765             LPM_WRADDRESS_CONTROL=>"UNREGISTERED"
766         )
767         port map
768         (
769             rdaddress=> ADDR_REG_B,
770             wraddress=> ADDR_REG_ACC,
771             wren=> REG_ACC_EN,
772             wrclock=> GEN,
773             data=> REG_ACC,
774             q=> REG_B
775         );
776
777     CLOCK_PROC:
```

```
778           process (
779                       GEN,
780                       RESET
781                   ) is
782
783                   begin
784                       if RESET='0' then
785                           AUT_CPU<=(others => '0');
786                           AUT_BUS<=(others => '0');
787                           REG_ADDR<=(others => '0');
788                           REG_DATA<=(others => '0');
789                           SIG_MREQ<='1';
790                           SIG_IORQ<='1';
791                           SIG_RD<='1';
792                           SIG_WR<='1';
793                           REG_PC<=(others => '0');
794                           REG_CMD<=(others => '0');
795
796                       elsif rising_edge(GEN) then
797                           AUT_CPU<=AUT_CPUn;
798                           AUT_BUS<=AUT_BUSn;
799                           REG_ADDR<=REG_ADDRn;
800                           REG_DATA<=REG_DATAn;
801                           SIG_MREQ<=SIG_MREQn;
802                           SIG_IORQ<=SIG_IORQn;
803                           SIG_RD<=SIG_RDn;
804                           SIG_WR<=SIG_WRn;
805                           SIG_DOUT<=SIG_DOUTn;
806                           REG_PC<=REG_PCn;
807                           REG_CMD<=REG_CMDn;
808                           REG_ACC_EN<=REG_ACC_ENn;
809                           REG_ACC<=REG_ACCn;
810                       end if;
811                   end process CLOCK_PROC;
812
813       -- debug
814       D_AUT_CPU<=AUT_CPU;
```

```
815       D_AUT_BUS<=AUT_BUS;
816       D_PC<=REG_PC;
817       --D_DATA <= REG_DATA;
818       D_DATA <= REG_ADDR(7 downto 0);
819
820       D_REG_ACC_EN    <= REG_ACC_EN;
821       D_ADDR_REG_A    <= ADDR_REG_A;
822       D_ADDR_REG_B    <= ADDR_REG_B;
823       D_ADDR_REG_ACC  <= ADDR_REG_ACC;
824       D_REG_A         <= REG_A;
825   end architecture CPU_module;
```

```
 1  -- Automat mnozacy metoda podstawowa
 2
 3  library ieee;
 4  use ieee.std_logic_1164.all;
 5  use ieee.std_logic_arith.all;
 6
 7  entity MUL is
 8      generic (
 9          wsize   : natural := 8;     -- rozmiar slowa
10          asize   : natural := 2 * 8 - 1; -- rozmiar akumulatora
11          csize   : natural := 3      -- rozmiar licznika rund (log2(wsize))
12      );
13      port (
14          GEN     : in std_logic;
15          RESET   : in std_logic;
16          A       : in std_logic_vector (wsize - 1 downto 0);
17          B       : in std_logic_vector (wsize - 1 downto 0);
18          RESULT  : out std_logic_vector (wsize - 1 downto 0);-- wynik mnozenia
19          GO      : in std_logic;     -- uruchamia MUL, aktywne LOW
20          READY   : out std_logic     -- czy wynik gotowy? (LOW)
21  --      D_ACC   : out std_logic_vector (asize - 1 downto 0);
22  --      D_ASHL  : out std_logic_vector (asize - 1 downto 0);
23  --      D_BSHR  : out std_logic_vector (wsize - 1 downto 0);
24  --      D_AUT   : out std_logic_vector (1 downto 0);
25  --      D_ROUND : out std_logic_vector (csize - 1 downto 0)
26      );
27  end entity MUL;
28
29  architecture arch_MUL of MUL is
30      signal ashl : unsigned (asize - 1 downto 0);-- SHL(Ra,n-1)
31      signal ashln: unsigned (asize - 1 downto 0);-- SHL(Ra,n-1)
32      signal bshr : unsigned (wsize - 1 downto 0);-- B
33      signal bshrn: unsigned (wsize - 1 downto 0);-- B
34      signal acc  : unsigned (asize - 1 downto 0);-- akumulator
35      signal accn : unsigned (asize - 1 downto 0);-- akumulator
36
37      signal round: unsigned (csize - 1 downto 0);-- licznik rund
```

```
38        signal roundn:unsigned (csize - 1 downto 0);-- licznik rund
39        constant maxround: unsigned (csize - 1 downto 0) := (others => '1');
40
41        signal RESULT_BUF   : std_logic_vector (wsize - 1 downto 0);
42        signal RESULT_BUFn  : std_logic_vector (wsize - 1 downto 0);
43
44        signal READY_BUF    : std_logic := '1';
45
46        signal AUT_MUL      : std_logic_vector (1 downto 0) := "00";
47        signal AUT_MULn     : std_logic_vector (1 downto 0);
48        constant AUT_IDLE   : std_logic_vector (1 downto 0) := "00";
49        constant AUT_RUNNING: std_logic_vector (1 downto 0) := "01";
50        constant AUT_OUTPUT : std_logic_vector (1 downto 0) := "11";
51        constant AUT_DONE   : std_logic_vector (1 downto 0) := "10";
52
53        begin
54    --  D_ACC <= std_logic_vector(acc);
55    --  D_ASHL <= std_logic_vector(ashl);
56    --  D_BSHR <= std_logic_vector(bshr);
57    --  D_AUT <= std_logic_vector(AUT_MUL);
58    --  D_ROUND <= std_logic_vector(round);
59        RESULT <= RESULT_BUF;
60
61        process_clock:
62        process (RESET, GEN, AUT_MULn, accn, bshrn,
63                 roundn, ashln, RESULT_BUFn, READY_BUF)
64        begin
65            if RESET = '0' then
66                AUT_MUL <= AUT_IDLE;
67                READY <= '1';
68                RESULT_BUF <= (others => '0');
69            else
70                if rising_edge(GEN) then
71                    AUT_MUL <= AUT_MULn;
72                    acc <= accn;
73                    bshr <= bshrn;
74                    ashl <= ashln;
```

```
75                  round <= roundn;
76                  RESULT_BUF <= RESULT_BUFn;
77                  READY <= READY_BUF;
78              end if;
79          end if;
80      end process;
81
82      multiply:
83      process (GEN, ashl, acc, bshr, round, RESULT_BUF, AUT_MUL, A, B, GO) is
84      begin
85          case AUT_MUL is
86              when AUT_IDLE =>
87                  READY_BUF <= '1';
88                  RESULT_BUFn <= RESULT_BUF;
89                  ashln(asize - 1 downto wsize - 1) <= unsigned(A(wsize - 1 downto 0));
90                  ashln(wsize - 2 downto 0) <= (others => '0');
91                  bshrn <= unsigned(B);
92                  roundn <= (others => '0');
93                  accn <= (others => '0');
94                  if GO = '0' then
95                      AUT_MULn <= AUT_RUNNING;
96                  else
97                      AUT_MULn <= AUT_IDLE;
98                  end if;
99              when AUT_RUNNING =>
100                 READY_BUF <= '1';
101                 RESULT_BUFn <= RESULT_BUF;
102                 ashln <= ashl;
103
104                 if bshr(0) = '1' then
105                     accn <= ('0' & acc(asize - 1 downto 1)) + ashl;
106                 else
107                     accn <= '0' & acc(asize - 1 downto 1);
108                 end if;
109
110                 roundn <= round + 1;
111                 bshrn(wsize - 1 downto 0) <= '0' & bshr(wsize - 1 downto 1);
```

```
112
113                     if round < maxround then
114                         AUT_MULn <= AUT_RUNNING;
115                     else
116                         AUT_MULn <= AUT_OUTPUT;
117                     end if;
118                 when AUT_OUTPUT =>
119                     READY_BUF <= '1';
120                     accn <= acc;
121                     ashln <= ashl;
122                     bshrn <= bshr;
123                     roundn <= round;
124                     RESULT_BUFn <= std_logic_vector(acc(wsize - 1 downto 0));
125                     AUT_MULn <= AUT_DONE;
126                 when AUT_DONE =>
127                     accn <= acc;
128                     ashln <= ashl;
129                     bshrn <= bshr;
130                     roundn <= round;
131                     RESULT_BUFn <= RESULT_BUF;
132                     READY_BUF <= '0';
133                     if GO = '1' then
134                         AUT_MULn <= AUT_IDLE;
135                     else
136                         AUT_MULn <= AUT_DONE;
137                     end if;
138             end case;
139         end process;
140
141     end architecture arch_MUL;
142
```

```
1   TITLE "LPT_OUT modul obslugi wyjsciowego lacza rownoleglego";
2   %
3   Modul podlaczony do ukladu SML: 235_DB25F,
4   odbierajacy z szyny danych 8-bitowa liczbe U2
5   i wysylajacy ja w postaci ZDDD przez port LPT do drukarki.
6   (Z - znak, D - cyfra dziesietna)
7   %
8   INCLUDE "U2_TO_ASCII.inc";
9
10  CONSTANT strobe_time = 10;  % 0,5us dla !STROBE                  %
11  % 0,5u / 0,05u = 10                                             %
12
13  CONSTANT dev_addr   = H"10";    % adres tego urzadzenia w IO     %
14  CONSTANT cr         = H"0D";    % ASCII dla CR                   %
15  CONSTANT lf         = H"0A";    % ASCII dla LF                   %
16
17  SUBDESIGN LPT_OUT
18  (
19      GEN         : input;    % zegar 20MHz                       %
20      RESET       : input;    % sygnal resetu mikrokontr.         %
21
22      % sygnaly szyny wewnetrznej mikrokontrolera                 %
23      A[7..0]     : input;    % linie adresowe szyny              %
24      D[7..0]     : input;    % linie danych szyny                %
25      IORQ        : input;
26      WR          : input;
27      WT          : output;
28
29      % sygnal powiadamiajacy o gotowosci drukarki, aktywny '1'   %
30      PRN_READY   : output;
31
32      % sygnaly do komunikacji z laczem LPT                       %
33      LCTRL[7..0] : output;       % SV2: nSI,nI,nAF,nS do LPT     %
34      LSTAT[7..0] : input;        % SV3: BS,PE,nF,S z LPT         %
35      LDATASYN[7..0]  : output;   % SV5: dane do LPT              %
36
37      % sygnaly debug do symulacji %
```

```
38        V_BPRINT    :output;
39        V_APRINT    :output;
40        V_CSTIME[5..0] :output;
41    )
42    VARIABLE
43        LDATA[7..0] : DFF;      % synchronizowany sygnal LPT::D     %
44        BPRINT      : NODE;     % komenda z szyny: drukuj           %
45        APRINT      : NODE;     % komenda od AUT_BUS: drukuj        %
46        WAIT_I      : NODE;     % wewnetrzny sygnal WAIT            %
47        nWAIT_SYN   : DFF;      % synchronizowany sygnal !WAIT      %
48        DIN[7..0]   : NODE;     % szyna danych wejsciowych          %
49        PRNT_OK     : NODE;     % czy drukarka ok?(!PE & nF & S)    %
50        BUSY        : NODE;     % sygnal BUSY od drukarki           %
51        STROBE      : NODE;     % wewnetrzny sygnal STROBE          %
52        nSTROBE_SYN : DFF;      % synchronizowany sygnal !STROBE    %
53        TRI_DI[7..0]: TRI;      % bufor 3stanowy wyjscia na szyne D %
54        CSTIME[5..0]: DFF;      % zlicza czas strobe                %
55        Z[7..0]     : NODE;     % kod ASCII znaku: 2D='-',2B='+'    %
56        D2[7..0]    : NODE;     % kod ASCII cyfry: 30..39           %
57        D1[7..0]    : NODE;
58        D0[7..0]    : NODE;
59        U2A         : U2_TO_ASCII;  % konwerter U2->ASCII(ZDDD)     %
60
61        % automat obslugujacy wydruk pojedynczego znaku             %
62        AUT_PRINT   : machine of bits (QP[1..0])
63                        with states (SP0=B"00", SP1=B"01",
64                                     SP2=B"10", SP3=B"11");
65
66        % automat obslugujacy szyne i sterujacy logika wydruku      %
67        AUT_BUS     : machine of bits (QB[3..0])
68                        with states (IDL=B"0000",
69                                     CRA=B"0001",
70                                     CRB=B"0010",
71                                     ZA =B"0011",
72                                     ZB =B"0100",
73                                     D2A=B"0101",
74                                     D2B=B"0110",
```

```
                                        D1A=B"0111",
                                        D1B=B"1000",
                                        D0A=B"1001",
                                        D0B=B"1010",
                                        LFA=B"1011",
                                        LFB=B"1100",
                                        WBU=B"1101");
BEGIN
    % DEBUG %
    V_BPRINT = BPRINT;
    V_APRINT = APRINT;
    V_CSTIME[] = CSTIME[];

    % podlaczenie przerzutnikow DFF i automatow                %
    nWAIT_SYN.clk   = GEN;
    nWAIT_SYN.clrn  = RESET;
    nWAIT_SYN       = !WAIT_I;
    LDATA[].clk     = GEN;
    LDATA[].clrn    = RESET;
    nSTROBE_SYN.clk = GEN;
    nSTROBE_SYN.clrn= RESET;
    CSTIME[].clk    = GEN;
    CSTIME[].clrn   = RESET;
    AUT_PRINT.clk   = GEN;
    AUT_PRINT.reset = !RESET;
    AUT_BUS.clk     = GEN;
    AUT_BUS.reset   = !RESET;

    % podlaczenie ukladu konwersji U2->ASCII(Z,D2,D1,D0)        %
    U2A.U2I[]       = DIN[];
    Z[]             = U2A.Z[];
    D2[]            = U2A.D2[];
    D1[]            = U2A.D1[];
    D0[]            = U2A.D0[];

    % podlaczenie linii komunikacji z laczem LPT               %
    LDATASYN[]      = LDATA[];
```

```
112       BUSY            = LSTAT[4];
113
114       % !PError & nFault & Select                                    %
115       PRNT_OK         = !LSTAT[2] & LSTAT[1] & LSTAT[0];
116       PRN_READY       = PRNT_OK;
117
118       % zapewnienie STROBE=VCC podczas startu i RESETu               %
119       nSTROBE_SYN     = !STROBE;
120
121       % nSelectIn=0, nInit=1, nAutoFd=1                              %
122       LCTRL[]         = (1,1,1,1,0,1,1,!nSTROBE_SYN);
123
124       % zdekodowanie rozkazu wyslania danych do LPT                  %
125       if (A[]==dev_addr & IORQ==GND & WR==GND)
126           then BPRINT = VCC;
127           else BPRINT = GND;  end if;
128
129       % wczytanie danych z szyny D[]                                 %
130       TRI_DI[].oe     = BPRINT;
131       TRI_DI[].in     = D[];
132       DIN[]           = TRI_DI[];
133
134       % wystawienie sygnalu WAIT na szyne                            %
135       --WT               = !nWAIT_SYN;
136       WT              = TRI(!nWAIT_SYN, BPRINT);   % (in, oe)     %
137
138       % automat obslugujacy szyne i sterujacy logika wydruku        %
139       case AUT_BUS is
140           % oczekiwanie na dane z szyny %
141           when IDL => if (BPRINT==VCC)then    WAIT_I=GND;
142                               APRINT=VCC; LDATA[]=cr;  AUT_BUS=CRA;
143                       else                WAIT_I=VCC;
144                               APRINT=GND; LDATA[]=cr;  AUT_BUS=IDL;
145                       end if;
146           % oczekiwanie na pojawienie sie !STROBE dla CR
147               - oznacza to, ze AUT_PRINT ruszyl, nie przeoczymy
148               tego sygnalu, bo trwa >1 takt zegara                 %
```

```
149           when CRA => if (STROBE==GND)then    WAIT_I=GND;
150                               APRINT=GND; LDATA[]=cr;  AUT_BUS=CRB;
151                           else                WAIT_I=GND;
152                               APRINT=VCC; LDATA[]=cr;  AUT_BUS=CRA;
153                           end if;
154           % oczekiwanie na koniec wydruku CR                      %
155           when CRB => if (AUT_PRINT==SP0)then WAIT_I=GND;
156                               APRINT=VCC; LDATA[]=Z[]; AUT_BUS=ZA;
157                           else                WAIT_I=GND;
158                               APRINT=GND; LDATA[]=cr;  AUT_BUS=CRB;
159                           end if;
160           % oczekiwanie na start wydruku Z %
161           when ZA => if (STROBE==GND)then     WAIT_I=GND;
162                               APRINT=GND; LDATA[]=Z[]; AUT_BUS=ZB;
163                           else                WAIT_I=GND;
164                               APRINT=VCC; LDATA[]=Z[]; AUT_BUS=ZA;
165                           end if;
166           % oczekiwanie na koniec wydruku Z                       %
167           when ZB => if (AUT_PRINT==SP0)then  WAIT_I=GND;
168                               APRINT=VCC; LDATA[]=D2[];AUT_BUS=D2A;
169                           else                WAIT_I=GND;
170                               APRINT=GND; LDATA[]=Z[]; AUT_BUS=ZB;
171                           end if;
172           % oczekiwanie na start druku D2                         %
173           when D2A => if (STROBE==GND)then    WAIT_I=GND;
174                               APRINT=GND; LDATA[]=D2[];AUT_BUS=D2B;
175                           else                WAIT_I=GND;
176                               APRINT=VCC; LDATA[]=D2[];AUT_BUS=D2A;
177                           end if;
178           % oczekiwanie na koniec wydruku D2                      %
179           when D2B => if (AUT_PRINT==SP0)then WAIT_I=GND;
180                               APRINT=VCC; LDATA[]=D1[];AUT_BUS=D1A;
181                           else                WAIT_I=GND;
182                               APRINT=GND; LDATA[]=D2[];AUT_BUS=D2B;
183                           end if;
184           % oczekiwanie na start druku D1                         %
185           when D1A => if (STROBE==GND)then    WAIT_I=GND;
```

```
186                             APRINT=GND; LDATA[]=D1[];AUT_BUS=D1B;
187                         else                WAIT_I=GND;
188                             APRINT=VCC; LDATA[]=D1[];AUT_BUS=D1A;
189                         end if;
190             % oczekiwanie na koniec wydruku D1                      %
191             when D1B => if (AUT_PRINT==SP0)then WAIT_I=GND;
192                             APRINT=VCC; LDATA[]=D0[];AUT_BUS=D0A;
193                         else                WAIT_I=GND;
194                             APRINT=GND; LDATA[]=D1[];AUT_BUS=D1B;
195                         end if;
196             % oczekiwanie na start druku D0                         %
197             when D0A => if (STROBE==GND)then    WAIT_I=GND;
198                             APRINT=GND; LDATA[]=D0[];AUT_BUS=D0B;
199                         else                WAIT_I=GND;
200                             APRINT=VCC; LDATA[]=D0[];AUT_BUS=D0A;
201                         end if;
202             % oczekiwanie na koniec wydruku D0                      %
203             when D0B => if (AUT_PRINT==SP0)then WAIT_I=GND;
204                             APRINT=VCC; LDATA[]=lf;  AUT_BUS=LFA;
205                         else                WAIT_I=GND;
206                             APRINT=GND; LDATA[]=D0[];AUT_BUS=D0B;
207                         end if;
208             % oczekiwanie na start druku LF                         %
209             when LFA => if (STROBE==GND)then    WAIT_I=GND;
210                             APRINT=GND; LDATA[]=lf;  AUT_BUS=LFB;
211                         else                WAIT_I=GND;
212                             APRINT=VCC; LDATA[]=lf;  AUT_BUS=LFA;
213                         end if;
214             % oczekiwanie na koniec wydruku LF                      %
215             when LFB => if (AUT_PRINT==SP0)then WAIT_I=VCC;
216                             APRINT=GND; LDATA[]=Z[]; AUT_BUS=WBU;
217                         else                WAIT_I=GND;
218                             APRINT=GND; LDATA[]=lf;  AUT_BUS=LFB;
219                         end if;
220             % oczekiwanie na wycofanie sie z szyny pol. druku       %
221             when WBU => if (BPRINT==GND)then    WAIT_I=VCC;
222                             APRINT=GND; LDATA[]=Z[]; AUT_BUS=IDL;
```

```
223                         else                WAIT_I=VCC;
224                             APRINT=GND; LDATA[]=Z[]; AUT_BUS=WBU;
225                         end if;
226       end case;
227
228       % automat obslugujacy wydruk pojedynczego znaku             %
229       case AUT_PRINT is
230           % oczekiwanie na polecenie druku                       %
231           when SP0 => if (APRINT==VCC) then
232                           STROBE=VCC; AUT_PRINT=SP1;
233                       else
234                           STROBE=VCC; AUT_PRINT=SP0; end if;
235           % sprawdzenie stanu linii BUSY                         %
236           when SP1 => if (BUSY==GND)  then
237                           STROBE=GND; AUT_PRINT=SP2;
238                       else
239                           STROBE=VCC; AUT_PRINT=SP1; end if;
240           % wystawienie !STROBE na czas strobe_time              %
241           when SP2 => if (CSTIME[]==strobe_time) then
242                           STROBE=VCC; AUT_PRINT=SP3;
243                       else
244                           STROBE=GND; AUT_PRINT=SP2;
245                           if (CSTIME[] < strobe_time) then
246                               CSTIME[]=CSTIME[]+1;
247                           else
248                               CSTIME[]=CSTIME[];
249                           end if; end if;
250           % zaczekanie na BUSY = L                               %
251           when SP3 => if (BUSY==GND)  then
252                           STROBE=VCC; AUT_PRINT=SP0;
253                       else
254                           STROBE=VCC; AUT_PRINT=SP3; end if;
255       end case;
256
257   END;
```

```
1   TITLE "PS/2 INPUT";
2   %-----------------------------------------------------------------------
3       PS/2 input module : Communication between PS/2 bus and System
4                           bus.
5   %-----------------------------------------------------------------------
6   INCLUDE "SC_TO_U2.inc";
7
8   CONSTANT ps2_addr = H"50";              -- Keyboard address in I/O
9
10  SUBDESIGN PS2_IN
11  (
12      GEN                     : input;    -- 20MHz clock
13      RESET                   : input;    -- Reset signal from uC
14
15      -- Bus
16      ADDR[7..0]              : input;    -- Address bus
17      DATA[7..0]              : output;   -- Data bus
18      WT                      : output;   -- Wait signal
19      IORQ                    : input;    -- I/O request signal
20      RD                      : input;    -- Read signal
21
22      -- PS/2 debug
23      PS2_DEBUG_PORT[7..0]    : output;   -- Output port for debugging
24
25      -- PS/2 connection port
26      PDATA_IN[7..0]          : input;    -- PS/2 bus input port
27  )
28  VARIABLE
29      -- Bus
30      TRI_D[7..0]             : TRI;      -- tristate buffer for D[]
31      CMD_READ                : DFF;      -- Read command from uC
32      KEY_Z[7..0]             : DFF;      -- Sign byte from keyboard
33      KEY_D0[7..0]            : DFF;      -- First number
34      KEY_D1[7..0]            : DFF;      -- Second number
35      KEY_D2[7..0]            : DFF;      -- Third number
36      SIG_WAIT                : DFF;      -- Wait signal for uC
37      DFF_DATA[7..0]          : DFF;      -- Register for final U2 number
```

```
38       CNT_DATA[7..0]          : DFF;      -- Wait extension counter
39       S2U                     : SC_TO_U2; -- Scancode to U2 decoder
40       -- Secure input dff's
41       DFF_1_PS2_CLK           : DFF;      -- Input dff #1 for clock
42       DFF_2_PS2_CLK           : DFF;      -- Input dff #2 for clock
43       DFF_1_PS2_DATA          : DFF;      -- Input dff #1 for data
44       DFF_2_PS2_DATA          : DFF;      -- Input dff #2 for data
45       -- Data recieving
46       PS2_FRAME[10..0]        : DFF;      -- Recieved frame
47       PS2_CNT_FRAME[3..0]     : DFF;      -- Frame bit's counter
48       PS2_CNT_BYTE[1..0]      : DFF;      -- ZDDD byte counter
49       PS2_BYTE[7..0]          : DFF;      -- Data from PS/2 frame - hardwired
50       PS2_BYTE_READY          : DFF;      -- Data byte is ready flag
51       -- Debug
52       D_PS2_DEBUG_PORT[7..0]  : DFF;      -- Debugging register
53
54       -- Machines
55
56       -- Machine for PS/2 bus communication
57       AUT_PS2     : machine of bits (Q[1..0])
58                     with states (S0=B"00", S1=B"01",
59                                  S2=B"10", S3=B"11");
60
61       -- Machine for system bus communication
62       AUT_PS2_BUS : machine of bits (QB[2..0])
63                     with states (SPB0=B"000", SPB1=B"001", SPB2=B"010",
64                                  SPB3=B"011", SPB4=B"100", SPB5=B"101",
65                                  SPB6=B"110", SPB7=B"111");
66
67
68   BEGIN
69       % assign global clock %
70       DFF_1_PS2_CLK.CLK       = GEN;
71       DFF_2_PS2_CLK.CLK       = GEN;
72       DFF_1_PS2_DATA.CLK      = GEN;
73       DFF_2_PS2_DATA.CLK      = GEN;
74       PS2_FRAME[].CLK         = GEN;
```

```
 75        PS2_CNT_FRAME[].CLK     = GEN;
 76        PS2_CNT_BYTE[].CLK      = GEN;
 77        D_PS2_DEBUG_PORT[].CLK  = GEN;
 78        PS2_BYTE[].CLK          = GEN;
 79        PS2_BYTE_READY.CLK      = GEN;
 80        CMD_READ.CLK            = GEN;
 81        KEY_Z[].CLK             = GEN;
 82        KEY_D0[].CLK            = GEN;
 83        KEY_D1[].CLK            = GEN;
 84        KEY_D2[].CLK            = GEN;
 85        DFF_DATA[].CLK          = GEN;
 86        SIG_WAIT.CLK            = GEN;
 87        CNT_DATA[].CLK          = GEN;
 88        S2U.GEN                 = GEN;
 89        AUT_PS2.CLK             = GEN;
 90        AUT_PS2_BUS.CLK         = GEN;
 91
 92        % assign global reset %
 93        S2U.RESET               = RESET;
 94        DFF_1_PS2_CLK.CLRN      = RESET;
 95        DFF_2_PS2_CLK.CLRN      = RESET;
 96        DFF_1_PS2_DATA.CLRN     = RESET;
 97        DFF_2_PS2_DATA.CLRN     = RESET;
 98        PS2_FRAME[].CLRN        = RESET;
 99        PS2_CNT_FRAME[].CLRN    = RESET;
100        PS2_CNT_BYTE[].CLRN     = RESET;
101        D_PS2_DEBUG_PORT[].CLRN = RESET;
102        PS2_BYTE[].CLRN         = RESET;
103        PS2_BYTE_READY.CLRN     = RESET;
104        CMD_READ.CLRN           = RESET;
105        KEY_Z[].CLRN            = RESET;
106        KEY_D0[].CLRN           = RESET;
107        KEY_D1[].CLRN           = RESET;
108        KEY_D2[].CLRN           = RESET;
109        DFF_DATA[].CLRN         = RESET;
110        SIG_WAIT.CLRN           = RESET;
111        CNT_DATA[].CLRN         = RESET;
```

```
112       AUT_PS2.RESET           = !RESET;
113       AUT_PS2_BUS.RESET       = !RESET;
114
115       % connect variables %
116       --WT                    = SIG_WAIT.Q;
117       WT                      = TRI(SIG_WAIT.Q, CMD_READ);     % (in, oe)     %
118
119       S2U.SC_Z[]              = KEY_Z[];
120       S2U.SC_D0[]             = KEY_D0[];
121       S2U.SC_D1[]             = KEY_D1[];
122       S2U.SC_D2[]             = KEY_D2[];
123
124
125   --  case PS2_CNT_BYTE[] is
126   --      when B"00"  => PS2_DEBUG_PORT[] = 0;
127   --      when B"01"  => PS2_DEBUG_PORT[] = 1;
128   --      when B"10"  => PS2_DEBUG_PORT[] = 3;
129   --      when B"11"  => PS2_DEBUG_PORT[] = 7;
130   --  end case;
131
132       PS2_DEBUG_PORT[] = (Q1, Q0, GND, GND, GND, QB2, QB1, QB0);
133
134       % tristate output data to D[] bus %
135       TRI_D[].oe              = CMD_READ;
136       TRI_D[].in              = DFF_DATA[];
137       DATA[]                  = TRI_D[];
138       -- DATA[] = DFF_DATA[];
139
140       % Hardwired byte from frame %
141       PS2_BYTE[] = (PS2_FRAME8,PS2_FRAME7,PS2_FRAME6,PS2_FRAME5,PS2_FRAME4,PS2_FRAME3,PS2_FRAME2,PS2_FRAME1);
142       % Secure input of ps2 clock %
143       DFF_1_PS2_CLK.D = PDATA_IN1;
144       DFF_2_PS2_CLK.D = DFF_1_PS2_CLK.Q;
145
146       % Secure input of ps2 data %
147       DFF_1_PS2_DATA.D = PDATA_IN0;
148       DFF_2_PS2_DATA.D = DFF_1_PS2_DATA.Q;
```

```
149
150       % Bus %
151       if(ADDR[] == ps2_addr & IORQ == GND & RD == GND) then
152               % Command read %
153               CMD_READ.D = VCC;
154       else
155               CMD_READ.D = GND;
156       end if;
157
158       case AUT_PS2_BUS is
159           % Wait for command from uC %
160           when SPB0 => CMD_READ.D = CMD_READ.Q;
161                        KEY_Z[] = B"00000000";
162                        KEY_D0[] = B"00000000";
163                        KEY_D1[] = B"00000000";
164                        KEY_D2[] = B"00000000";
165                        PS2_CNT_BYTE[] = B"00";
166                        CNT_DATA[] = B"00000000";
167                        DFF_DATA[] = 0;
168
169                        if(CMD_READ.Q == 1) then
170                           SIG_WAIT.D = GND;
171                           -- activate reciever
172                           AUT_PS2_BUS = SPB1;
173                        else
174                           -- unblock after reset
175                           SIG_WAIT.D = GND;
176                           AUT_PS2_BUS = SPB0;
177                        end if;
178           % Wait for frame %
179           when SPB1 => CMD_READ.D = CMD_READ.Q;
180                        KEY_Z[] = KEY_Z[];
181                        KEY_D0[] = KEY_D0[];
182                        KEY_D1[] = KEY_D1[];
183                        KEY_D2[] = KEY_D2[];
184                        CNT_DATA[] = CNT_DATA[];
185                        DFF_DATA[] = DFF_DATA[];
```

```
186                         PS2_CNT_BYTE[] = PS2_CNT_BYTE[];
187
188
189                         if(PS2_BYTE_READY.Q == 1) then
190                            AUT_PS2_BUS = SPB2;
191                         else
192                            AUT_PS2_BUS = SPB1;
193                         end if;
194
195               % Check if frame data is "F0" %
196               when SPB2 => CMD_READ.D = CMD_READ.Q;
197                         KEY_Z[] = KEY_Z[];
198                         KEY_D0[] = KEY_D0[];
199                         KEY_D1[] = KEY_D1[];
200                         KEY_D2[] = KEY_D2[];
201                         CNT_DATA[] = CNT_DATA[];
202                         DFF_DATA[] = DFF_DATA[];
203                         PS2_CNT_BYTE[] = PS2_CNT_BYTE[];
204
205                         if(PS2_BYTE[] == B"11110000") then
206                            AUT_PS2_BUS = SPB3;
207                         else
208                            AUT_PS2_BUS = SPB1;
209                         end if;
210
211               % Wait for data frame ( after "F0" comes scancode of key ) %
212               when SPB3 => CMD_READ.D = CMD_READ.Q;
213                         KEY_Z[] = KEY_Z[];
214                         KEY_D0[] = KEY_D0[];
215                         KEY_D1[] = KEY_D1[];
216                         KEY_D2[] = KEY_D2[];
217                         CNT_DATA[] = CNT_DATA[];
218                         DFF_DATA[] = DFF_DATA[];
219                         PS2_CNT_BYTE[] = PS2_CNT_BYTE[];
220
221                         if(PS2_BYTE_READY.Q == 1) then
222                            AUT_PS2_BUS = SPB4;
```

```
223                         else
224                            AUT_PS2_BUS = SPB3;
225                         end if;
226
227                 % Write key scancode into correct buffer %
228                 when SPB4 => CMD_READ.D = CMD_READ.Q;
229                              KEY_Z[] = KEY_Z[];
230                              KEY_D0[] = KEY_D0[];
231                              KEY_D1[] = KEY_D1[];
232                              KEY_D2[] = KEY_D2[];
233                              CNT_DATA[] = CNT_DATA[];
234                              DFF_DATA[] = DFF_DATA[];
235                              PS2_CNT_BYTE[] = PS2_CNT_BYTE[]+1;
236
237
238                              case PS2_CNT_BYTE[] is
239                                 when 0 => KEY_Z[] = PS2_BYTE[];
240                                           AUT_PS2_BUS = SPB1;
241                                 when 1 => KEY_D2[] = PS2_BYTE[];
242                                           AUT_PS2_BUS = SPB1;
243                                 when 2 => KEY_D1[] = PS2_BYTE[];
244                                           AUT_PS2_BUS = SPB1;
245                                 when 3 => KEY_D0[] = PS2_BYTE[];
246                                           AUT_PS2_BUS = SPB5;
247                              end case;
248
249                 % Convert ZDDD to U2 and put it on data bus %
250                 when SPB5 => CMD_READ.D = CMD_READ.Q;
251                              KEY_Z[] = KEY_Z[];
252                              KEY_D0[] = KEY_D0[];
253                              KEY_D1[] = KEY_D1[];
254                              KEY_D2[] = KEY_D2[];
255                              CNT_DATA[] = CNT_DATA[];
256                              PS2_CNT_BYTE[] = PS2_CNT_BYTE[];
257
258                              -- DEBUG
259                              DFF_DATA[] = S2U.U2_LS[];
```

```
260                         --DFF_DATA[] = B"00000101";
261
262                         if( CNT_DATA[] < 20 ) then
263                            AUT_PS2_BUS = SPB6;
264                         else
265                            AUT_PS2_BUS = SPB7;
266                         end if;
267
268               % Extend wait signal %
269               when SPB6 => CMD_READ.D = CMD_READ.Q;
270                            KEY_Z[] = KEY_Z[];
271                            KEY_D0[] = KEY_D0[];
272                            KEY_D1[] = KEY_D1[];
273                            KEY_D2[] = KEY_D2[];
274                            CNT_DATA[] = CNT_DATA[]+1;
275                            PS2_CNT_BYTE[] = PS2_CNT_BYTE[];
276                            DFF_DATA[] = DFF_DATA[];
277
278                            AUT_PS2_BUS = SPB5;
279               % Remove Wait, and wait for processor read %
280               when SPB7 => SIG_WAIT.D = VCC;
281                            CMD_READ.D = GND;
282                            KEY_Z[] = KEY_Z[];
283                            KEY_D0[] = KEY_D0[];
284                            KEY_D1[] = KEY_D1[];
285                            KEY_D2[] = KEY_D2[];
286                            CNT_DATA[] = CNT_DATA[];
287                            PS2_CNT_BYTE[] = PS2_CNT_BYTE[];
288
289                            DFF_DATA[] = DFF_DATA[];
290
291                            if(IORQ == VCC & RD == VCC)then
292                               AUT_PS2_BUS = SPB0;
293                            else
294                               AUT_PS2_BUS = SPB7;
295                            end if;
296
```

```
297       end case;
298
299       case AUT_PS2 is
300           % PS2 clock down - start bit %
301           when S0 =>  PS2_CNT_FRAME[] = B"0000";
302                       PS2_FRAME[] = B"00000000000";
303                       PS2_BYTE_READY.D = GND;
304
305                       if(DFF_1_PS2_CLK.Q == 0 & DFF_2_PS2_CLK.Q == 1) then
306                           AUT_PS2 = S2;
307                       else
308                           AUT_PS2 = S0;
309                       end if;
310           % Wait for clock falling edge %
311           when S1 =>  PS2_CNT_FRAME[] = PS2_CNT_FRAME[];
312                       PS2_FRAME[] = PS2_FRAME[];
313                       PS2_BYTE_READY.D = GND;
314
315                       if(DFF_1_PS2_CLK.Q == 0 & DFF_2_PS2_CLK.Q == 1) then
316                           AUT_PS2 = S2;
317                       else
318                           AUT_PS2 = S1;
319                       end if;
320           % Read bit into frame buffer %
321           when S2 =>  PS2_CNT_FRAME[] = PS2_CNT_FRAME[]+1;
322                       PS2_FRAME[] = (DFF_2_PS2_DATA.Q, PS2_FRAME[10..1]);
323                       PS2_BYTE_READY.D = GND;
324
325                       if(PS2_CNT_FRAME[] < 10) then
326                           AUT_PS2 = S1;
327                       else
328                           AUT_PS2 = S3;
329                       end if;
330           % 11 bit frame is ready %
331           when S3 =>  PS2_FRAME[] = PS2_FRAME[];
332                       PS2_CNT_FRAME[] = PS2_CNT_FRAME[];
333
```

```
334                     PS2_BYTE_READY.D = VCC;
335                     -- reciever off
336                     AUT_PS2 = S0;
337         end case;
338
339     END;
```

```
1   --  Blok pamieci RAM    : 1099B -> ograniczona do 1024B ze wzgledu na ilosc EAB
2   --  Przestrzen adresowa : 0x1000..0x144A (4096..5194)
3   --                      -> 0x1000..0x1399 (4096..5119)
4
5   --  Wersja z zatrzaskiwaniem:
6   --  Czas ustalenia sie D przy odczycie: 30ns
7   --      UWAGA: A i D musza byc ustalone na 10ns przed opuszczeniem MRQ i RD
8   --  Czas wyjscia z szyny D po odczycie: 15ns
9   --  Czas zapisu: 20ns
10  --      UWAGA: A i D musza byc ustalone na 20ns przed opuszczeniem MRQ i WR
11
12  library ieee;
13  use ieee.std_logic_1164.all;
14  use ieee.std_logic_arith.all;
15  library lpm;
16  use lpm.lpm_components.all;
17  library altera;
18  use altera.altera_primitives_components.all;
19
20  entity RAM is
21      generic (   base_addr   : natural := 4096;  -- adres bazowy w przestrzeni adr
22                  last_addr   : natural := 5119 );-- ostatni adres w przestrzeni adr
23
24      port (  A   : in std_logic_vector (15 downto 0);
25              D   : inout std_logic_vector (7 downto 0);
26      MRQ, RD, WR : in std_logic
27  --  V_CSR :out std_logic;
28  --  V_CSW : out std_logic;
29  --  V_LD : out std_logic_vector (7 downto 0);
30  --  V_LA : out std_logic_vector (10 downto 0);
31  --  V_DOUT : out std_logic_vector (7 downto 0)
32      );
33
34  end entity RAM;
35
36  architecture arch_RAM of RAM is
37      signal CSEL, CSW, CSR   : std_logic; -- zdekodowane sygnal wyboru pamieci RAM
```

```
38        signal LD   : std_logic_vector (7 downto 0);    -- zalatchowany sygnal D[]
39        signal LA   : std_logic_vector (10 downto 0);   -- zalatchowany sygnal A[]
40        signal DOUT : std_logic_vector (7 downto 0);    -- sygnal wyjsciowy D[]
41
42        begin
43    --      V_CSR <= CSR;
44    --      V_CSW <= CSW;
45    --      V_LA <= LA;
46    --      V_LD <= LD;
47    --      V_DOUT <= DOUT;
48
49            -- zatrzasnij D[] i A[] dla WR
50        l1: process (A, MRQ, WR, D, LA, LD) is
51            begin
52                if ((MRQ = '1') and (WR = '1')) then
53                    LD <= D;
54                    LA <= A(10 downto 0);
55                else
56                    LD <= LD;
57                    LA <= LA;
58                end if;
59            end process l1;
60
61        -- zdekodowanie sygnalu WR
62        sw: process (A, MRQ, WR) is
63            begin
64                if (((unsigned(A))>=base_addr) and ((unsigned(A))<=last_addr) and (MRQ='0') and (WR='0'))
65                    then CSW <= '1'; else CSW <= '0'; end if;
66            end process sw;
67
68        -- zdekodowanie sygnalu RD
69        sr: process (A, MRQ, RD) is
70            begin
71                if (((unsigned(A))>=base_addr) and ((unsigned(A))<=last_addr) and (MRQ='0') and (RD='0'))
72                    then CSR <= '1'; else CSR <= '0'; end if;
73            end process sr;
74
```

```
75       e0: lpm_ram_dq
76           generic map (LPM_WIDTH=>8, LPM_WIDTHAD=>10, LPM_NUMWORDS=>1024,
77                   LPM_INDATA => "UNREGISTERED", LPM_OUTDATA => "UNREGISTERED",
78                   LPM_ADDRESS_CONTROL => "UNREGISTERED")
79           port map (data => LD, address=> LA(9 downto 0), we => CSW, q => DOUT);
80
81       t1: lpm_bustri
82           generic map (LPM_WIDTH=>8)
83           port map (data => DOUT, enabledt => CSR, tridata => D);
84
85   end architecture arch_RAM;
```

```
1   --  Blok pamieci ROM    : 90B
2   --  Przestrzen adresowa : 0x0000..0x0059 (0..89)
3   --  Czas ustalenia sie D przy odczycie  : 25ns
4   --  Czas wyjscia z D po cofnieciu MREQ i RD : 20ns
5   --  Poniewaz T(CPU) = 1/f = 50ns > 25ns, ROM nie wystawia sygnalu WT.
6
7   library ieee;
8   use ieee.std_logic_1164.all;
9   use ieee.std_logic_arith.all;
10  library lpm;
11  use lpm.lpm_components.all;
12  library altera;
13  use altera.altera_primitives_components.all;
14
15  entity ROM is
16      generic ( last_addr : natural := 89 );  -- ostatni adres w przestrzeni adr
17
18      port (  A   : in std_logic_vector (15 downto 0);
19              D   : inout std_logic_vector (7 downto 0);
20          MRQ, RD : in std_logic );
21
22  end entity ROM;
23
24  architecture arch_ROM of ROM is
25      signal CSEL : std_logic;    -- zdekodowany sygnal wyboru pamieci ROM przez CPU
26      begin
27
28      cs: process (A, MRQ, RD) is
29          begin
30              if (((unsigned(A))<=last_addr) and (MRQ='0') and (RD='0'))
31                  then CSEL <= '1'; else CSEL <= '0'; end if; -- chip select
32          end process cs;
33
34      e0: lpm_rom
35          generic map (LPM_WIDTH=>8, LPM_WIDTHAD=>7, LPM_NUMWORDS=>90,
36                  LPM_FILE=>"cpu_asm_test.mif", LPM_OUTDATA => "UNREGISTERED",
37                  LPM_ADDRESS_CONTROL => "UNREGISTERED",
```

```
38                   LPM_HINT => "UNUSED")
39             port map (address => A(6 downto 0), q => D, memenab => CSEL);
40
41   end architecture arch_ROM;
```

```
TITLE "Scancode to BCD conversion";
%----------------------------------------------------------------------
    PS/2 input module : Scancode decoder
%----------------------------------------------------------------------
SUBDESIGN SC_TO_BCD
(
    SC[7..0]        : input;    -- Scancode input
    BCD[7..0]       : output;   -- BCD output
)
BEGIN
    TABLE   SC[]    =>      BCD[];
            H"45"   =>      B"00000000";
            H"16"   =>      B"00000001";
            H"1E"   =>      B"00000010";
            H"26"   =>      B"00000011";
            H"25"   =>      B"00000100";
            H"2E"   =>      B"00000101";
            H"36"   =>      B"00000110";
            H"3D"   =>      B"00000111";
            H"3E"   =>      B"00001000";
            H"46"   =>      B"00001001";
            -- debug
            H"0F"   =>      B"00000001";
            H"8F"   =>      B"00000010";
    END TABLE;
END;
```

```
1   TITLE "Scancode to U2 conversion";
2   %--------------------------------------------------------------------
3       PS/2 input module : ZDDD -> U2 converter.
4   %--------------------------------------------------------------------
5   INCLUDE "SC_TO_BCD.inc";
6
7   SUBDESIGN SC_TO_U2
8   (
9       GEN                 : input;        -- 20MHz clock
10      RESET               : input;        -- Reset signal from uC
11
12      SC_Z[7..0]          : input;        -- Z
13      SC_D0[7..0]         : input;        -- D
14      SC_D1[7..0]         : input;        -- D
15      SC_D2[7..0]         : input;        -- D
16
17      U2_MS[7..0]         : output;       -- Output most significant (not used)
18      U2_LS[7..0]         : output;       -- Output less significant
19  )
20  VARIABLE
21      SIGN                : DFF;          -- Sign of number
22      S2B0                : SC_TO_BCD;    -- Digit 0 decoder
23      S2B1                : SC_TO_BCD;    -- Digit 1 decoder
24      S2B2                : SC_TO_BCD;    -- Digit 2 decoder
25      BCD_D0[7..0]        : DFF;          -- Digit 0 in BCD
26      BCD_D1[7..0]        : DFF;          -- Digit 1 in BCD
27      BCD_D2[7..0]        : DFF;          -- Digit 2 in BCD
28      D0[7..0]            : DFF;          -- Digit 0 multiplied
29      D1[7..0]            : DFF;          -- Digit 1 multiplied
30      D2[7..0]            : DFF;          -- Digit 2 multiplied
31      BINARY[7..0]        : DFF;          -- Binary from ZDDD
32      NEGATOR[8..0]       : DFF;          -- For binary negation
33      NEGATED[8..0]       : DFF;          -- For binary negation
34      NEGATION[8..0]      : DFF;          -- For binary negation
35      RESULT[7..0]        : DFF;          -- Converter final output
36  BEGIN
37      % Clock %
```

```
38       SIGN.CLK            = GEN;
39       BCD_D0[].CLK        = GEN;
40       BCD_D1[].CLK        = GEN;
41       BCD_D2[].CLK        = GEN;
42       D0[].CLK            = GEN;
43       D1[].CLK            = GEN;
44       D2[].CLK            = GEN;
45       BINARY[].CLK        = GEN;
46       NEGATOR[].CLK       = GEN;
47       NEGATED[].CLK       = GEN;
48       NEGATION[].CLK      = GEN;
49       RESULT[].CLK        = GEN;
50
51       % Reset %
52       SIGN.CLRN           = RESET;
53       BCD_D0[].CLRN       = RESET;
54       BCD_D1[].CLRN       = RESET;
55       BCD_D2[].CLRN       = RESET;
56       D0[].CLRN           = RESET;
57       D1[].CLRN           = RESET;
58       D2[].CLRN           = RESET;
59       BINARY[].CLRN       = RESET;
60       NEGATOR[].CLRN      = RESET;
61       NEGATED[].CLRN      = RESET;
62       NEGATION[].CLRN     = RESET;
63       RESULT[].CLRN       = RESET;
64
65       % SC to BCD %
66       S2B0.SC[] = SC_D0[];
67       BCD_D0[] = S2B0.BCD[];
68
69       S2B1.SC[] = SC_D1[];
70       BCD_D1[] = S2B1.BCD[];
71
72       S2B2.SC[] = SC_D2[];
73       BCD_D2[] = S2B2.BCD[];
74
```

```
75        % Multiplication %
76        D0[] = BCD_D0[];
77        D1[] = BCD_D1[]*10;
78        D2[] = BCD_D2[]*100;
79
80        % Sum %
81        BINARY[] = D0[] + D1[] + D2[];
82
83        % Sign %
84        if( SC_Z[] == H"4E" ) then
85            -- Sign is "-" so convert to U2
86            NEGATOR[] = B"100000000";
87            NEGATED[] = (GND, BINARY[7..0]);
88            NEGATION[] = NEGATOR[] - NEGATED[];
89            RESULT[] = (NEGATION[7..0]);
90        else
91            RESULT[] = BINARY[];
92        end if;
93
94        % Return result %
95        U2_LS[] = RESULT[];
96        % Future extension? %
97        U2_MS[] = (GND, GND, GND, GND, GND, GND, GND, GND);
98    END;
```

```
1   TITLE "Konwerter liczby U2 (8bit) na 4 znaki ASCII: ZDDD";
2   %
3   Z - '+' lub '-'
4   D - 0..9
5   %
6   INCLUDE "bcd_sumator.inc";
7
8   SUBDESIGN U2_TO_ASCII
9   (
10      U2I[7..0]   : input;
11      Z[7..0]     : output;
12      D2[7..0]    : output;
13      D1[7..0]    : output;
14      D0[7..0]    : output;
15  )
16
17  VARIABLE
18      BCD[7..0]   : NODE;
19      U2[7..0]    : NODE;
20      S0          : bcd_sumator;
21      S1          : bcd_sumator;
22      S2          : bcd_sumator;
23      S3          : bcd_sumator;
24
25  BEGIN
26      S0.CIN  = U2I[7];   % jesli liczba ujemna, dodajemy 1 %
27      if (U2I[7]==1) then U2[] = !U2I[];  % negacja na potrzeby dodawania %
28          else            U2[] = U2I[];
29      end if;
30
31      S0.XX[] = (0,0,0,0,0,U2[2],U2[1],U2[0]);
32      S0.YY[] = (0,0,0,0,U2[3],0,0,0);
33      S1.XX[] = S0.ZZ[];
34      S1.CIN  = S0.NAD;
35      S2.CIN  = S1.NAD;
36      S3.CIN  = S2.NAD;
37
```

```
38       % 16 %
39       if (U2[4]==1) then  S1.YY[] = (0,0,0,1,0,1,1,0);
40           else            S1.YY[] = (0,0,0,0,0,0,0,0);
41       end if;
42       S2.XX[] = S1.ZZ[];
43
44       % 32 %
45       if (U2[5]==1) then  S2.YY[] = (0,0,1,1,0,0,1,0);
46           else            S2.YY[] = (0,0,0,0,0,0,0,0);
47       end if;
48       S3.XX[] = S2.ZZ[];
49
50       % 64 %
51       if (U2[6]==1) then  S3.YY[] = (0,1,1,0,0,1,0,0);
52           else            S3.YY[] = (0,0,0,0,0,0,0,0);
53       end if;
54       BCD[]   = S3.ZZ[];
55
56       % wyznaczenie wartosci znakow ASCII %
57       if (U2I[7]==1) then Z[] = H"2D";    % '-' %
58           else            Z[] = H"2B";    % '+' %
59       end if;
60       D2[] = (0,0,1,1,0,0,0,S3.NAD);
61       D1[] = (0,0,1,1,BCD[7],BCD[6],BCD[5],BCD[4]);
62       D0[] = (0,0,1,1,BCD[3],BCD[2],BCD[1],BCD[0]);
63   END;
```