Aleksy Barcz
Marcin Harasimczuk

Warszawa,
29.03.2011

Projektowanie urządzeń cyfrowych

Dokumentacja projektu - etap I

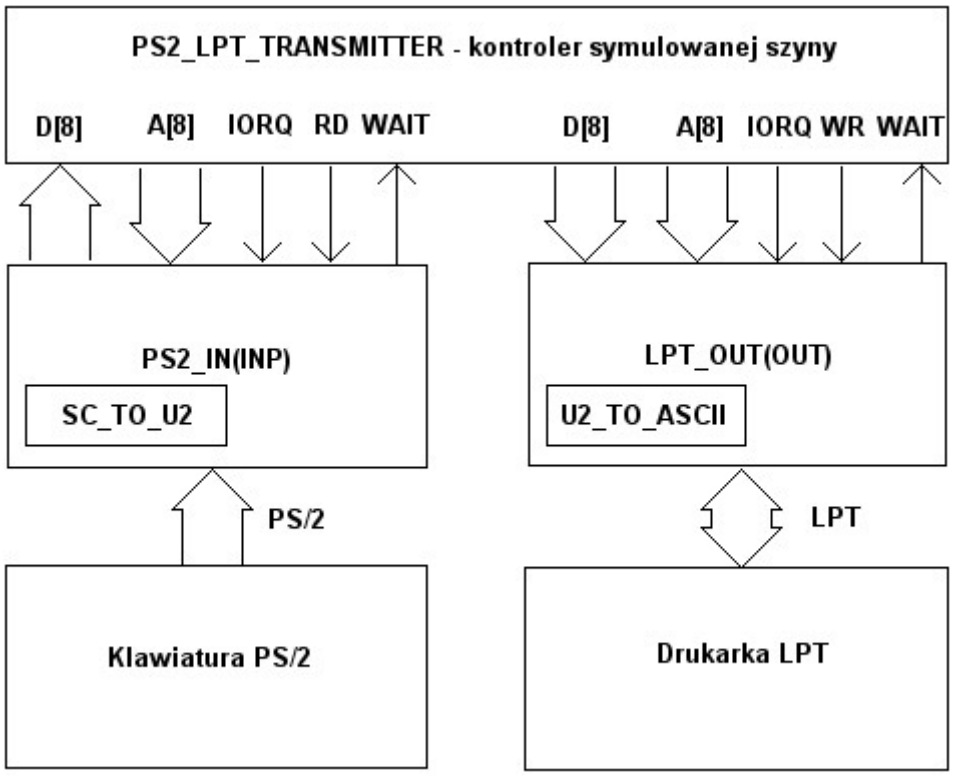
Zadanie 11L - 14D

Układ FPGA przesyłający liczby w formacie ZDDD z klawiatury PS/2 na drukarkę LPT.

1. **Treść zadania**
   Zaprojektować w języku AHDL moduły INP(0) (wejściowe łącze klawiatury PS/2 - klawisze znakowe) i OUT(0) (wyjściowe łącze równoległe LPT) plus moduł umożliwiający wyprowadzanie na łącze wyjściowe danych odbieranych z łącza wejściowego. Dane wejściowe i wyjściowe w postaci 4 znaków ASCII, ZDDD wyznaczających wartość dziesiętną DDD ze znakiem Z słowa 8-bitowego reprezentującego liczbę w kodzie U2 (bez błędów). Moduły powinny być zaprojektowane jako niezależne elementy współpracujące ze sobą za pośrednictwem wewnętrznej szyny systemowej.

2. **Założenia projektowe**
   a) sygnały szyny systemowej: WR, RD, IORQ, WAIT są aktywne poziomem niskim
   b) w związku z korzystaniem z AHDL'a na tym etapie projektu nie korzystaliśmy z linii trójstanowych
   c) moduł INP, po otrzymaniu rozkazu odczytu liczby, wystawia sygnał WAIT do czasu odebrania znaków ZDDD z klawiatury, dopiero wtedy wystawia pojedynczą liczbę U2 na linie danych szyny i zwalnia sygnał WAIT
   d) moduł OUT po otrzymaniu rozkazu wydruku liczby, wystawia sygnał WAIT, odczytuje z linii danych liczbę w kodzie U2, konwertuje ją ciąg ZDDD znaków ASCII i drukuje znak po znaku, po czym zwalnia sygnał WAIT
   e) ponieważ korzystamy tylko ze znaków 0..9, '+', '-' z klawiatury znakowej, uznajemy wciśnięcie klawisza '=/+' za znak '+', bez konieczności używania klawisza SHIFT
   f) kody klawiszy są wczytywane w momencie puszczenia klawisza (BREAK CODE), zakładamy że drukarka korzysta z zestawu 2 scan code'ów.

3. **Schemat blokowy**



4. **Moduł IN - łącze wejściowe PS/2**
   a) Poziom fizyczny komunikacji z łączem PS/2
      Do komunikacji z łączem wykorzystano moduł KB_PS2 systemu SML3.
      Kontroler PS/2 odbiera z portu modułu KB_PS2 2 sygnały - Clock i Data, są one wprowadzane na port 4 układu FPGA SML3.

   b) Moduł PS2_IN
      Moduł ten implementuje dwa automaty, które obsługują komunikację między szyną systemową a łączem PS/2, dodatkowo realizuje synchronizację sygnałów Clock i Data z łącza PS/2 z zegarem systemowym wprowadzając te sygnały na wejścia podwójnych zmiennych typu DFF rezydujących w tym module. W projekcie wstępnym przewidywano realizację odszumiania (eliminację krótkich zakłóceń na łączu PS/2), ale wstępne testy udowodniły, że nie jest ona potrzebna do prawidłowej pracy klawiatury.

      - Automat AUT_PS2 - Próbkuje zmienne DFF_1_PS2_CLK i DFF_2_PS2_CLK w celu wykrycia zboczy opadających zegara i buduje ramkę wprowadzając do niej kolejno odbierane bity pojawiające się na wyjściu zmiennej DFF_2_PS2_DATA. Odebranie ramki sygnalizuje sygnałem PS2_BYTE_READY w stanie wysokim.
      - Automat AUT_PS2_BUS - Oczekuje na rozkaz odczytu danych, następnie generuje sygnał wait obniżając wyjście zmiennej SIG_WAIT. Oczekuje na kolejne bajty danych odbierane przez automat AUT_PS2. Sprawdza czy odebrany bajt danych to Scancode 0xF0 co pozwala mu przypuszczać, że następny odebrany bajt będzie Scancodem puszczanego klawisza. Po otrzymaniu 4 bajtów (ZDDD) wydłuża sygnał sygnał SIG_WAIT przez 20 taktów zegara po czym zwalnia go i oczekuje na odczyt danych z szyny systemowej przez moduł nadrzędny.

   c) Moduł SC_TO_U2
      Moduł ten realizuje konwersje wejściowych bajtów ZDDD w postaci Scancode'a na wyjściowy bajt U2_LS w postaci kodu U2.

   d) Moduł SC_TO BCD
      Jest to moduł pomocniczy - dekoder Scancode-BCD, zrealizowany w postaci tablicowej, tłumaczy wprowadzony Scancode (klawiatura znakowa: 1,2,..,9,0,-,=) na odpowiedni kod BCD.
      Uwagi: Według założenia projektowego nie korzystamy z przycisku SHIFT, więc znak "+" to po prostu "=".

5. **Moduł OUT - łącze wyjściowe LPT**
   a) Format danych wyjściowych na drukarkę
      Pojedyncza zdekodowana liczba jest przesyłana na drukarkę jako ciąg znaków ASCII: CR, Z, D2, D1, D0, LF. Taka kolejność, gwarantująca przejście głowicy drukującej do następnej linii i cofnięcie jej do pozycji początkowej, została uzyskana metodą prób i błędów, po zauważeniu że kombinacje zawierające CR i LF lub LF i CR po znakach ZDDD nie dają właściwych rezultatów na drukarce. Zrezygnowano z opcji AutoFeed, gdyż nie dawała właściwych rezultatów.

   b) Poziom fizyczny komunikacji z drukarką
      Do komunikacji z drukarką wykorzystano moduł 235_DB25 SML.
      Kontroler drukarki komunikuje się z drukarką przy użyciu trzech portów:
      - LSTAT[8] (WE, SV4)- sygnały stanu drukarki:
        - BUSY - drukarka zajęta (sygnalizacja poziomem wysokim)
        - Select - poziom wysoki oznacza, że drukarka jest logicznie połączona z układem

- PaperError - poziom wysoki sygnalizuje wyczerpanie się papieru
- nFault - poziom niski sygnalizuje błąd drukarki

Stan sygnałów: (Select=1, PaperError=0, nFault=1) jest dekodowany jako sygnał gotowości drukarki. Sygnał nACK nie jest wykorzystywany w układzie.

- LCTRL[8] (WY, SV3) - sygnały sterujące pracą drukarki. Wykorzystywane sygnały:
  - STROBE - sygnał strobu wydruku
  - nSelIn - sygnał wyboru urządzenia, stale 0
  - nAutoFeed - automatyczne dołączanie znaku LF po CR, stale 1 (nieaktywne)
  - nInit - wykonanie procedury inicjalizującej drukarki, stale 1 (nieaktywne)
- LDATA[8] (WY, SV5) - linie danych portu LPT

Wydruk pojedynczego znaku przebiega następująco (*graf nr 3*):

- kontroler LPT wystawia na linie LDATA[] 8bitowy kod znaku ASCII
- kontroler opuszcza sygnał STROBE na czas 0,5µs (minimalny czas wymagany w standardzie, podczas testów okazał się wystarczający)
- drukarka podnosi sygnał BUSY na czas wydruku
- drukarka opuszcza sygnał BUSY, kontroler kończy cykl wydruku znaku, wraca do stanu początkowego

Sygnały LDATA oraz STROBE są synchronizowane, a w trakcie resetu urządzenia, sygnał STROBE ustawiany jest w stan wysoki, aby nie doszło do przypadkowego wydruku.

c) Komunikacja z szyną i ogólny schemat działania OUT (*graf nr 2*)
Po zdekodowaniu rozkazu wydruku, kontroler wystawia sygnał WAIT, po czym wczytuje liczbę w kodzie U2 z linii danych szyny systemowej, dekoduje ją do formatu ZDDD (ASCII) i rozpoczyna wydruk, zlecając wydruk kolejnych znaków wewnętrznemu automatowi kontrolującemu komunikację z drukarką. Po zakończeniu wydruku, sygnał WAIT jest wycofywany i kontroler czeka na wycofanie się rozkazu wydruku z szyny systemowej - gdy to nastąpi, wraca do stanu początkowego.
Sygnał WAIT jest synchronizowany, dla uniknięcia szpilenia, a w trakcie resetu ustawiany jest w stan wysoki.

d) Dekodowanie liczby w kodzie U2 do postaci kodów ASCII: ZDDD
Dekodowanie zostało zaimplementowane jako układ kombinacyjny, ustalający znak liczby na podstawie najbardziej znaczącego bitu oraz wyznaczający wartość kolejnych cyfr przy użyciu czterech 8bitowych (dwucyfrowych) sumatorów liczb w kodzie BCD (wykorzystano zmodyfikowany kod z wykładu). Kolejne sumowania to:

- S0 = bit3(8) + (bit2,bit1,bit0)
- S1 = bit4(16) + S0
- S2 = bit5(32) + S1
- S3 = bit6(64) + S2

e) Symulacja działania kontrolera OUT
Na dołączonym zrzucie ekranu znajduje się pełny przebieg symulacji pojedynczego cyklu działania kontrolera: zdekodowanie rozkazu wydruku, odebranie i wydruk liczby w kodzie BCD.

6. **Moduł zarządzający całością układu**

a) Sposób działania
Zaimplementowany układ sterujący szyną systemową cyklicznie wystawia rozkaz odczytu liczby z modułu PS/2, a po jej otrzymaniu (i zwolnieniu przez moduł sygnału WAIT), wycofuje sygnały z szyny i wystawia rozkaz wydruku odebranej liczby. Po

zakończeniu wydruku, wraca do stanu początkowego i rozpoczyna kolejny cykl pobranie->wydruk.
Możliwe jest rozpoczynanie kolejnych cykli pracy przy użyciu przełącznika SW1, aktualnie ta opcja jest wyłączona i układ po zakończeniu cyklu automatycznie rozpoczyna następny cykl (*graf nr 1*)
W związku z niekorzystaniem z sygnałów trójstanowych i dwukierunkowych, sygnały WAIT oraz D zostały rozdzielone dla modułu LPT i PS/2.

b) Sterowanie pracą układu
   - przełącznik SW3 służy do resetowania układu (ciągły reset gdy przełącznik jest w położeniu niskim)

c) Sygnalizacja na diodach
   Górny rząd diod:
   - dioda pierwsza od lewej (7) świecąc sygnalizuje gotowość drukarki do pracy (sygnał wyprowadzony z kontrolera OUT)
   - dioda druga od lewej sygnalizuje oczekiwanie układu na dane z klawiatury (sygnał WAIT)
   - diody 2,1,0 służą do wyświetlania stanu układu (*graf nr 1*)

   Dolny rząd diod:
   - świecenie diod 3,2,1,0 sygnalizuje odebranie kolejnych znaków przez moduł INP (sygnał wyprowadzony z kontrolera PS/2) - liczba świecących się diod oznacza ilość odebranych znaków.

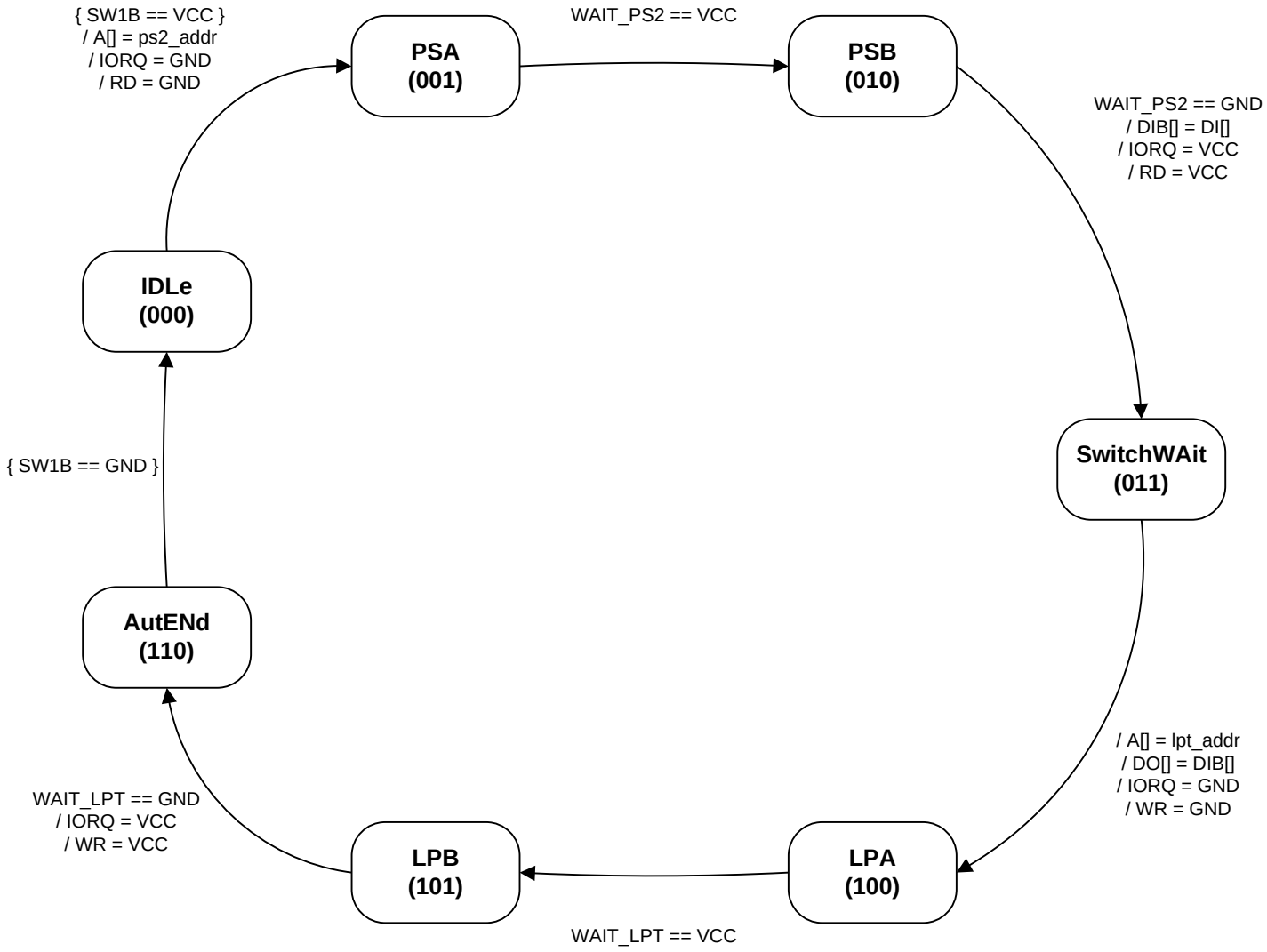
7. **Grafy**
   - graf nr 1: automat sterujący transmisją PS/2 -> LPT
   - graf nr 2: schemat komunikacji kontrolera LPT z szyną
   - graf nr 3: schemat komunikacji kontrolera LPT z drukarką

8. **Zrzuty ekranu z symulacji**
   - symulacje komunikacji z interfejsem PS/2
   - symulacja modułu LPT_OUT: cykl wydruku pojedynczej liczby ZDDD

9. **Listingi**
   Struktura programu:
   PS2_LPT_TRANSMITTER (moduł nadrzędny)
   - PS2_IN
     - SC_TO_U2
       - SC_TO_BCD
   - LPT_OUT
     - U2_TO_ASCII
       - bcd_sumator
         - bcd_blok

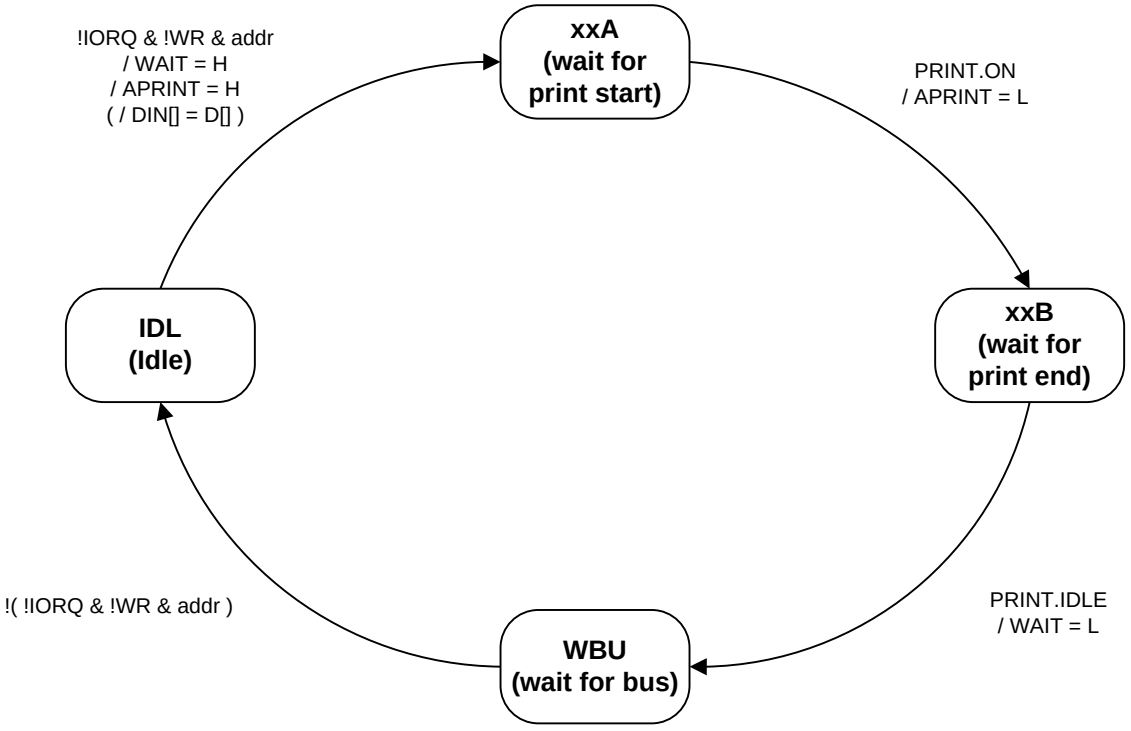
Graf 1: automat sterujący transmisją PS/2 -> LPT

SW1B - przełącznik sterujący pracą układu FPGA. Przejście L->H wyzwala jeden cykl przesłania danych PS/2->LPT
A[] - linie adresowe wewnętrznej szyny mikrokontrolera
IORQ - sygnał IORequest wewnętrznej szyny mikrokontrolera
RD - sygnał Read
WR - sygnał Write
WAIT_PS2 - sygnał WAIT pochodzący od modułu kontrolera PS/2
WAIT_LPT - sygnał WAIT pochodzący od modułu kontrolera LPT
DIB[] - wewnętrzny bufor do przechowywania danej wczytaniej z linii danych D
DI[] - wejściowa szyna danych od modułu PS/2
DO[] - wyjściowa szyna danych do modułu LPT
{ … } - funkcjonalność opcjonalna, możliwa praca automatyczna z pominięciem sygnału z przełącznika SW1B



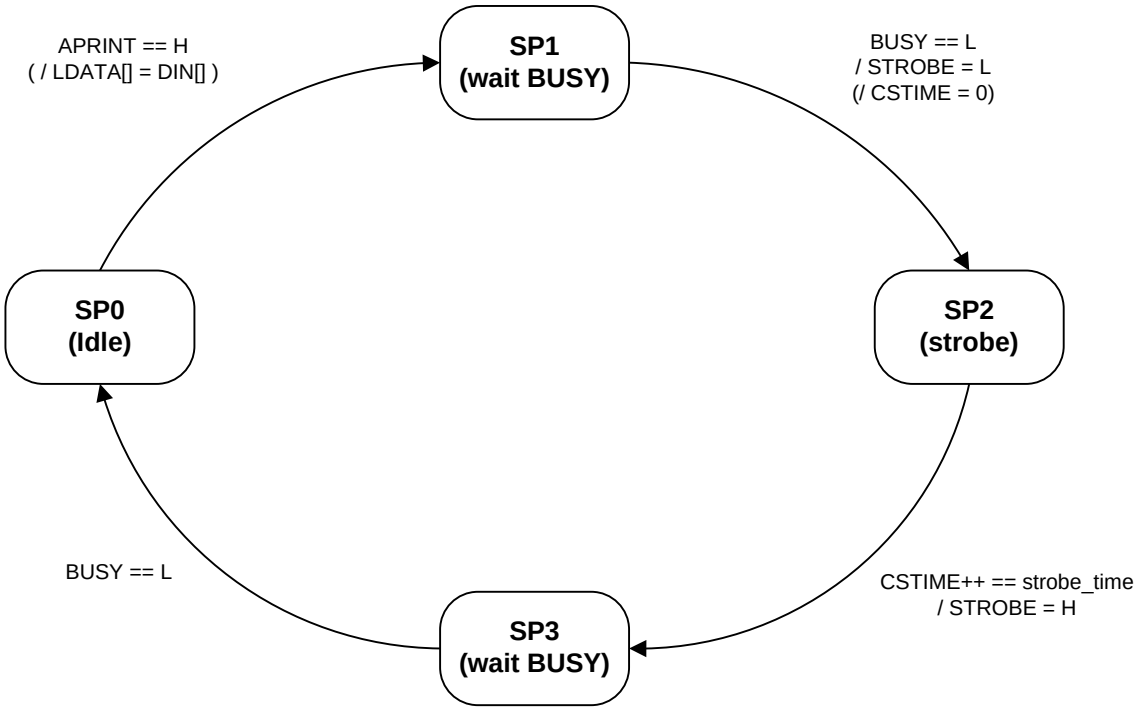
Graf 2: schemat komunikacji z szyną kontrolera LPT: OUT

IORQ, WR - sygnały szyny
addr - adres rejestru wejściowego OUT (do odbioru danych z szyny)
WAIT - sygnał szyny, ustawiany przez OUT na czas wydruku
APRINT - sygnał do automatu sterującego wydrukiem: „drukuj”
DIN[] - 8bit dane wejściowe z szyny zachowane w pamięci kontrolera
D[] - 8bit dane z szyny danych mikrokontrolera
PRINT.ON - automat drukujący rozpoczął drukowanie (!STROBE)
PRINT.IDLE – automat drukujący zakończył drukowanie (AUT_PRINT==SP0)
xxA, xxB - stany odpowiadające wydrukowi znaku ‘xx’, gdzie xx to kolejno: CR, Z, D2, D1, D0, LF
(dla uproszczenia grafu pokazano wydruk tylko jednego znaku)



Graf 3: schemat komunikacji z drukarką dla kontrolera LPT: OUT

APRINT - sygnał od automatu komunikującego się z szyną: „drukuj”
LDATA - 8bit dane wyjściowe na port równoległy
DIN - 8bit dane z szyny zbuforowane w pamięci kontrolera
BUSY - sygnał zajętości drukarki z portu równoległego
STROBE - sygnał strobu portu równoległego – dla drukarki
CSTIME - zmienna zliczająca cykle zegara, aby czas !STROBE > 0.5us
strobe_time - stała: czas trwania !STROBE w cyklach zegara

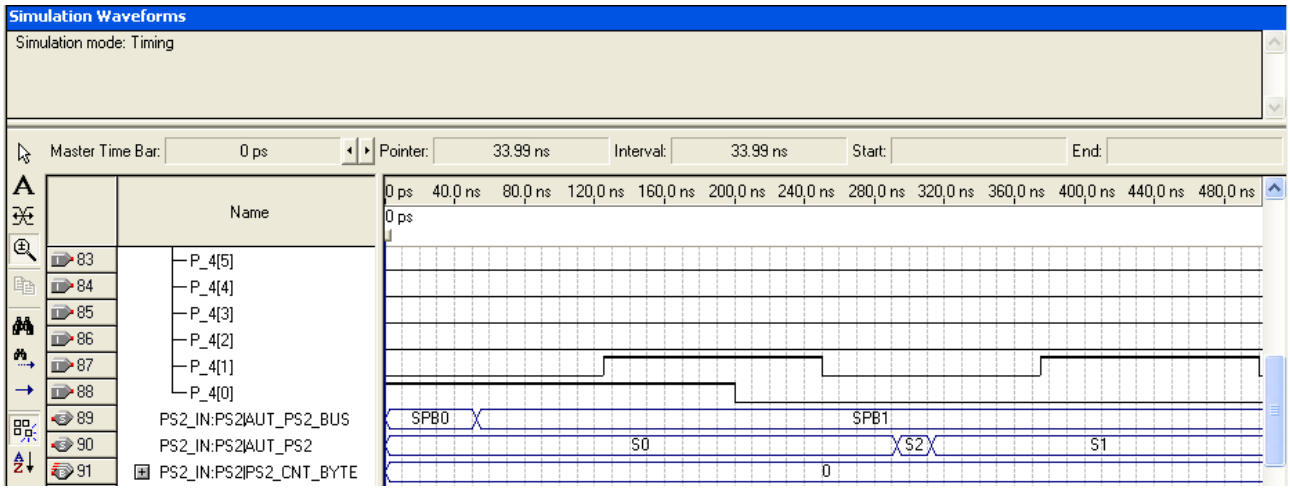


Symulacje komunikacji z interfejsem PS/2.

Oznaczenia:

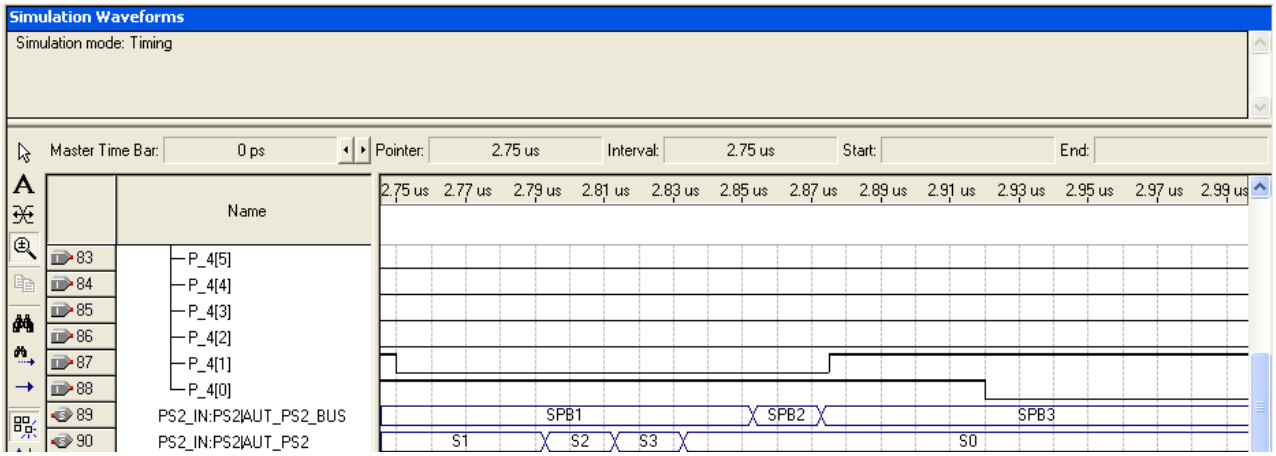
P4[1] - Sygnał clock z łącza PS/2.
P4[0] - Sygnał data z łącza PS/2.
AUT_PS2 - Automat operujący na bitach danych. Bada zmiany w sygnałach clock i data na łączu PS/2. Po zbudowaniu kompletnej ramki pozwala automatowi AUT_PS2_BUS na odebranie bajtu danych.
AUT_PS2_BUS - Automat operujący na bajtach danych odebranych przez automat PS2_BUS. Realizuje logikę komunikacji z szyną systemową.

Symulacja 1: Odebranie pierwszego bitu z ramki po otrzymaniu komendy czytania z szyny.



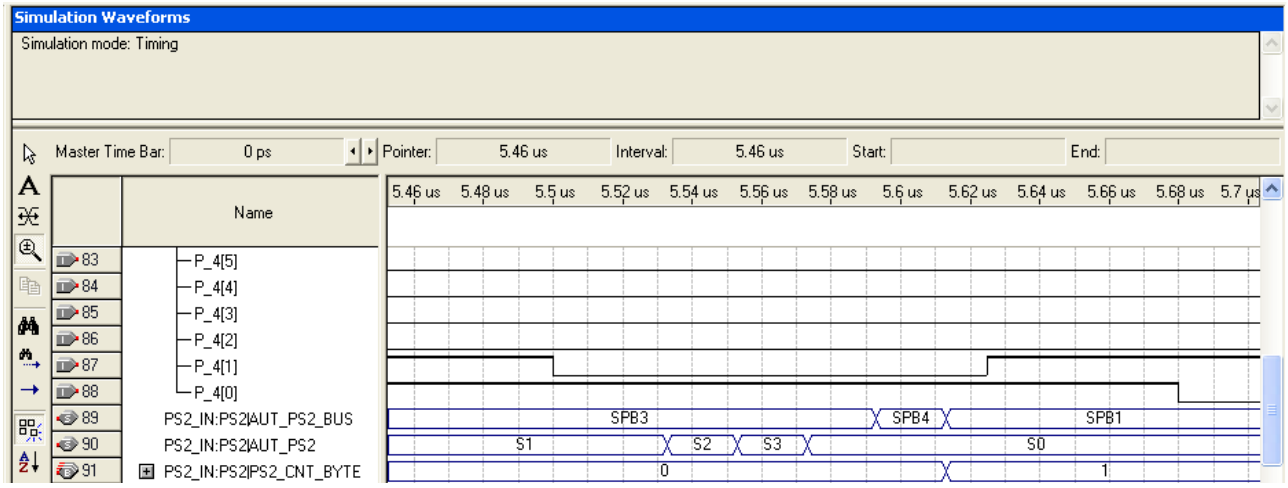
Automat AUT_PS2_BUS odbiera rozkaz czytania z portu PS/2 co powoduje, że przechodzi do stanu SPB1. Automat AUT_PS2 odczytuje bit startu (przejście S0->S2) a następnie przechodzi do stanu S1 (od tej pory przy odbieraniu bitów z łącza będzie przechodził między stanami S1 i S2).

Symulacja 2: Odebranie bajtu ze Scancode 0xF0.



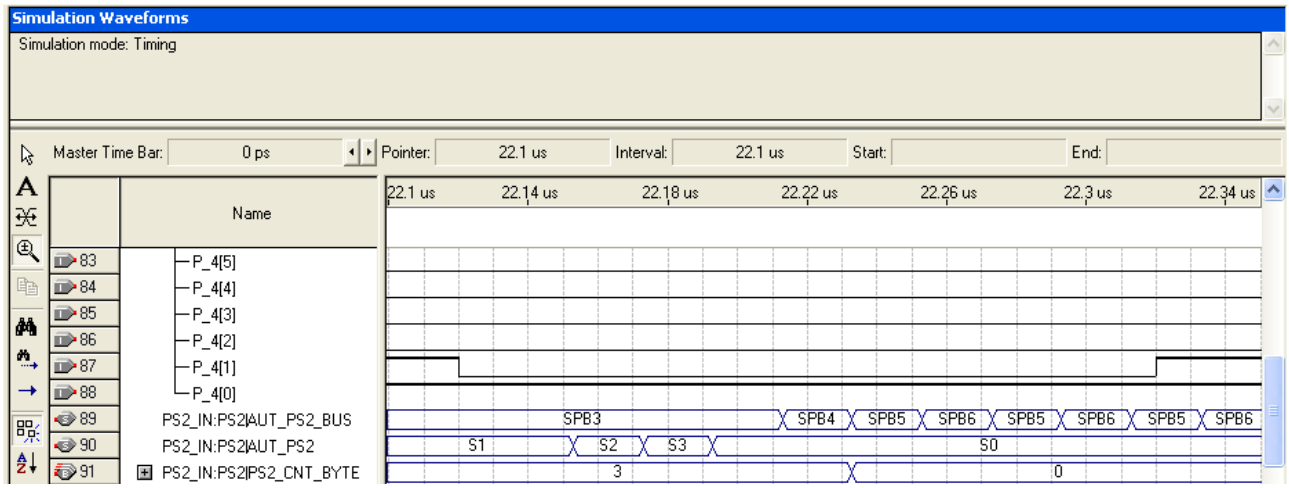
Po odebraniu ostatniego bitu ramki z łącza automat AUT_PS2 wystawia sygnał gotowości. Automat AUT_PS2_BUS przejmuje odebrany bajt danych i przechodzi ze stanu SPB1 do stanu SPB2, w którym sprawdza, czy odebrany bajt to 0xF0. W tym wypadku tak jest, więc automat AUT_PS2_BUS przechodzi do stanu SPB3, w którym oczekuje na następny bajt danych, który będzie Scancodem właśnie puszczonego klawisza.

<u>Symulacja 3:</u> Odebranie pierwszego Scancode'a.



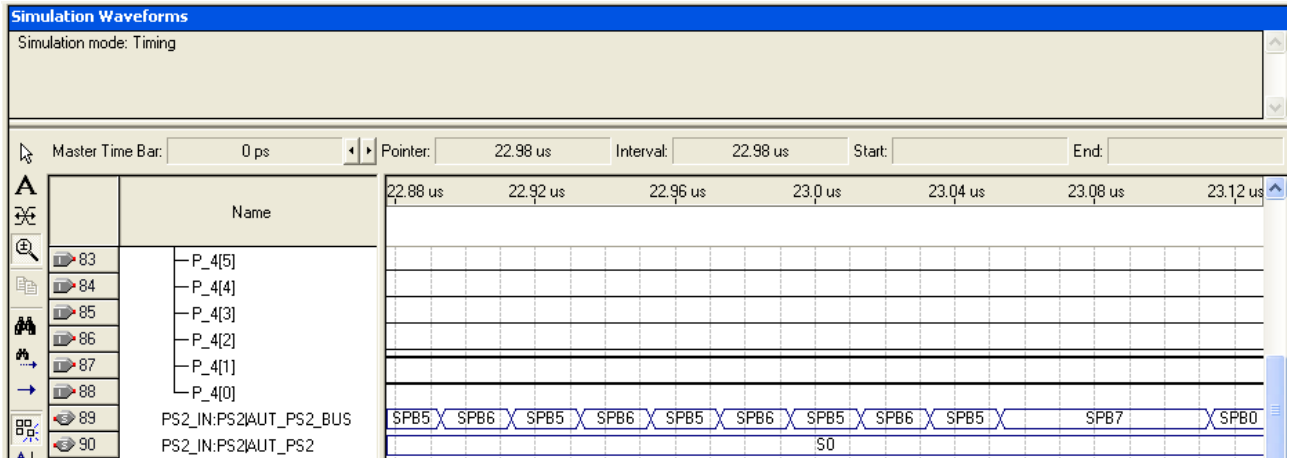
Automat AUT_PS2 przechodzi do stanu S3 i sygnalizuje odebranie ramki. Automat AUT_PS2_BUS oczekiwał na bajt ze Scancode'em puszczanego klawisza (stan SPB3), więc odbiera bajt i inkrementuje licznik PS2_CNT_BYTE. Jest to pierwszy odebrany bajt, więc odebraliśmy Z. Automat AUT_PS2_BUS wraca do stanu SPB1 i oczekuje na bajt 0xF0.

<u>Symulacja 4:</u> Odebranie ostatniego Scancode'a z serii 4 (ZDDD).



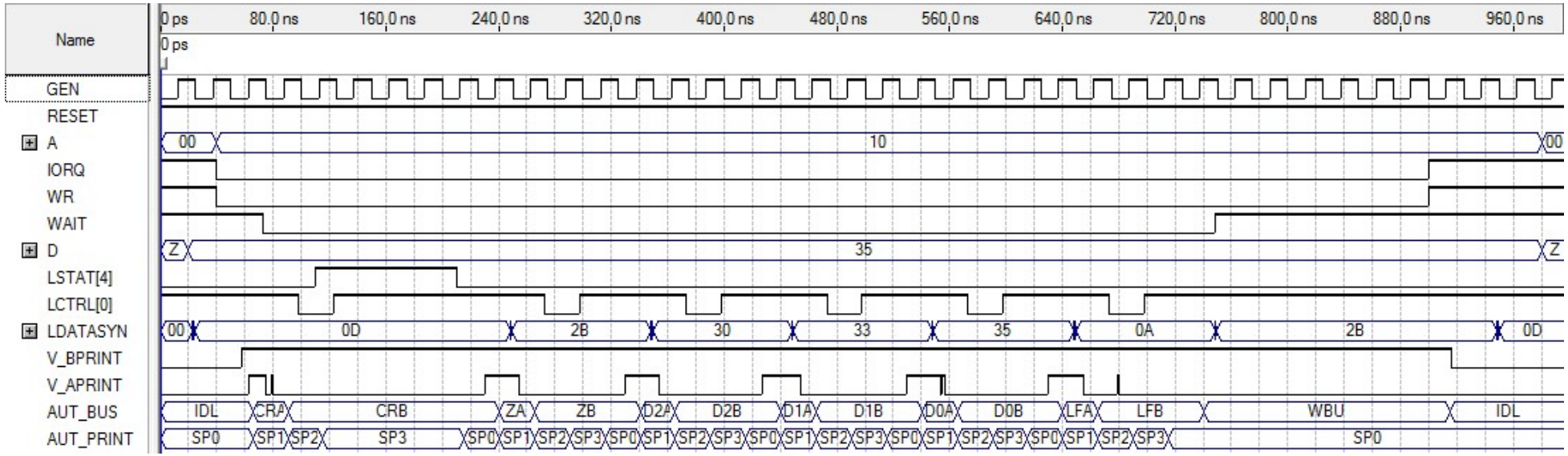
Automat AUT_PS2 sygnalizuje odebranie ramki. Automat AUT_PS2_BUS odbiera ostatni bajt danych z serii ZDDD i przechodzi do stanu SPB4, w którym sprawdza, czy odebrano już wszystkie bajty ZDDD. Odebrany ciąg ZDDD automatycznie trafia na wejście modułu dekodera U2, ale automat AUT_PS2_BUS wydłuża sygnał WAIT (przejścia między stanami SPB5 i SPB6 to odliczanie), aby zapewnić ustalenie się właściwych danych na wyjściu dekodera.

Symulacja 5: Wystawienie danych w formacie U2 na szynę i zwolnienie sygnału WAIT.



Automat AUT_PS2_BUS kończy odliczanie, zwalnia sygnał WAIT i czekuje na podniesienie przez "procesor" sygnałów IORQ i RD, po czym wraca do stanu SPB0 oczekując na koleną komendę odczytu.

Symulacja działania LPT_OUT: Cykl wydruku pojedynczej liczby (tu: 35)



Znaczenie sygnałów:

a) A == 10 - adres kontrolera OUT na linii adresowej
b) LSTAT[4] - sygnał BUSY z drukarki (widać w jaki sposób jego ustawienie wpływa na wydłużenie czasu trwania stanu SP3)
c) LCTRL[0] - synchronizowany sygnał STROBE (na potrzeby symulacji skrócony do jednego taktu użytego w symulacji zegara)
d) LDATASYN - synchronizowany sygnał danych do drukarki:
   - '0D' - kod ASCII znaku CR
   - '2B' / '2D' - kod ASCII znaku '+' / '-'
   - '3x' - kod ASCII cyfry 0..9
   - '0A' - kod ASCII znaku LF
e) V_BPRINT - wewnętrzny sygnał - zdekodowany rozkaz wydruku
f) V_APRINT - wewnętrzny sygnał uruchamiający wydruk pojedynczego znaku. (pokazany na grafie xx :TODO:) Szpilenie sygnału jest akceptowane, gdyż nie występuje w momencie gdy poziom APRINT jest sprawdzany (stan SP0 automatu drukującego znak).
g) AUT_BUS, AUT_PRINT - automaty opisane na grafie xx oraz yy (:TODO:)

```
 1  TITLE "Modul przesylajacy liczbe U2 z PS/2 na LPT";
 2  %
 3  Modul ma na celu zaprezentowanie dzialania:
 4  - kontrolera portu wejsciowego PS/2, odczytujacego
 5  liczbe U2 wpisana na klawiaturze jako ZDDD
 6  - kontrolera portu wyjsciowego LPT, wysylajacego
 7  na drukarke odebrana z PS/2 liczbe U2 (druk jako ZDDD)
 8  %
 9  INCLUDE "LPT_OUT.inc";
10  INCLUDE "PS2_IN.inc";
11
12  CONSTANT lpt_addr = H"10";  % adres drukarki LPT w IO       %
13  CONSTANT ps2_addr = H"50";  % adres klawiatury PS/2 w IO    %
14
15  SUBDESIGN PS2_LPT_TRANSMITTER
16  (
17      GEN         : input;    % zegar 20MHz                   %
18
19      % sygnaly WE / WY                                       %
20      L_A[7..0]   : output;   % diody gorny rzad              %
21      L_B[7..0]   : output;   % diody dolny rzad              %
22      P_1[7..0]   : output;   % port danych do LPT (SV1)      %
23      P_2[7..0]   : input;    % port stanu z LPT (SV2)        %
24      P_3[7..0]   : output;   % port sterowania do LPT (SV3)  %
25      P_4[7..0]   : input;    % port wejsciowy z PS/2 (SV4)   %
26      P_5[7..0]   : output;   % port SV5                      %
27      P_7[7..0]   : output;   % port SV7 (nieuzywany)         %
28      %SW1B       : input;%   % przelacznik RUN: VCC=>przeslij%
29      %SW2B       : input;%   % przelacznik SW2B, nieuzywany  %
30      SW3B        : input;    % przelacznik RESET: GND=>RES   %
31
32      % sygnaly debug na potrzeby symulacji                   %
33      V_A[7..0]   : output;
34      V_DIB[7..0] : output;
35      V_DI[7..0]  : output;
36      V_WR        : output;
37      V_RD        : output;
38      V_IORQ      : output;
39      V_WLPT      : output;
40      V_WPS2      : output;
41  )
42  VARIABLE
43      % sygnaly wewnetrznej szyny mikrokontrolera             %
44      A[7..0]     : DFF;      % linie adresowe szyny          %
45      nIORQ_SYN   : DFF;      % zsynchronizowane !IORQ        %
46      IORQ        : NODE;     % uzywane tylko wewnatrz modulu %
47
48      nWR_SYN     : DFF;      % zsynchronizowane !WR          %
49      WR          : NODE;     % uzywane tylko wewnatrz modulu %
50
51      nRD_SYN     : DFF;      % zsynchronizowane !RD          %
52      RD          : NODE;     % uzywane tylko wewnatrz modulu %
53
54      WAIT_PS2    : NODE;     % WAIT z PS/2                   %
55      WAIT_LPT    : NODE;     % WAIT z LPT                    %
```

```
56       % szyna D[] podzielona na czesc IN i OUT (dla AHDL)     %
57       DI[7..0]    : NODE;     % dane przesylane z PS/2 (U2)   %
58       DO[7..0]    : NODE;     % dane wysylane na LPT (U2)     %
59
60       % sygnal gotowosci drukarki, aktywny '1'                %
61       PRN_READY   : NODE;
62
63       % wewnetrzne zmienne modulu                              %
64       DIB[7..0]   : DFF;      % zbuforowane DI[]              %
65       RESET       : NODE;     % wewnetrzny sygnal RESET       %
66       LPT         : LPT_OUT;  % modul kontrolera LPT          %
67       PS2         : PS2_IN;   % modul kontrolera PS/2         %
68
69       % automat pobierajacy liczbe U2 z PS/2 i wysylajacy na LPT%
70       AUT_TEST    : machine of bits (QT[2..0])
71                       with states (   IDL=B"000", PSA=B"001",
72                                       PSB=B"010", SWA=B"011",
73                                       LPA=B"100", LPB=B"101",
74                                       AEN=B"110");
75   BEGIN
76       % debug %
77       V_A[]       = A[];
78       V_WR        = !nWR_SYN;
79       V_RD        = !nRD_SYN;
80       V_IORQ      = !nIORQ_SYN;
81       V_DIB[]     = DIB[];
82       V_DI[]      = DI[];
83       V_WLPT      = WAIT_LPT;
84       V_WPS2      = WAIT_PS2;
85       P_7[]       = 0;
86
87       % podlaczenie przerzutnikow DFF i automatow              %
88       DIB[].clk   = GEN;
89       DIB[].clrn  = RESET;
90       A[].clk     = GEN;
91       A[].clrn    = RESET;
92       AUT_TEST.clk    = GEN;
93       AUT_TEST.reset  = !RESET;
94       nIORQ_SYN.clk   = GEN;
95       nIORQ_SYN.clrn  = RESET;
96       nWR_SYN.clk     = GEN;
97       nWR_SYN.clrn    = RESET;
98       nRD_SYN.clk     = GEN;
99       nRD_SYN.clrn    = RESET;
100
101      % wymuszenie VCC na liniach przy starcie i podczas RESET%
102      nIORQ_SYN   = !IORQ;
103      nWR_SYN     = !WR;
104      nRD_SYN     = !RD;
105
106      % sygnal resetujacy z przelacznika 3                     %
107      RESET       = SW3B;
108
109      % sygnalizacja na diodach                                %
110      L_A[]       = (!PRN_READY,WAIT_PS2,WAIT_LPT,1,1,!QT[]);
```

```
111       L_B[]       = !PS2.PS2_DEBUG_PORT[];
112
113       % podlaczenie koncowek modulu PS/2                        %
114       %DI[]       = H"02";
115       WAIT_PS2    = SW2B;%
116       % WE/WY                                                   %
117       PS2.PDATA_IN[]  = P_4[];
118       P_5[]       = PS2.PS2_DEBUG_PORT[];
119       % sygnaly wewnetrznej szyny mikrokontrolera               %
120       WAIT_PS2    = PS2.WAIT;
121       DI[]        = PS2.DATA[];
122       PS2.ADDR[]  = A[];
123       PS2.IORQ    = !nIORQ_SYN;
124       PS2.RD      = !nRD_SYN;
125       % sygnaly kontrolne                                       %
126       PS2.RESET   = RESET;
127       PS2.GEN     = GEN;
128
129
130       % podlaczenie koncowek modulu LPT                         %
131       % WE/WY                                                   %
132       P_1[]       = LPT.LDATASYN[];
133       LPT.LSTAT[] = P_2[];
134       P_3[]       = LPT.LCTRL[];
135       % sygnaly wewnetrznej szyny mikrokontrolera               %
136       WAIT_LPT    = LPT.WAIT;
137       LPT.A[]     = A[];
138       LPT.IORQ    = !nIORQ_SYN;
139       LPT.WR      = !nWR_SYN;
140       LPT.D[]     = DO[];
141       % sygnaly kontrolne                                       %
142       LPT.GEN     = GEN;
143       LPT.RESET   = RESET;
144       PRN_READY   = LPT.PRN_READY;
145
146       % automat pobierajacy liczbe U2 z PS/2 i wysylajacy na LPT  %
147       case AUT_TEST is
148           % oczekiwanie na sygnal startu cyklu z przelacznika     %
149           when IDL => if (%SW1B==%VCC) then
150                   A[]=ps2_addr; IORQ=GND; RD=GND; WR=VCC;
151                                           AUT_TEST=PSA;
152               else
153                   A[]=ps2_addr; IORQ=VCC; RD=VCC; WR=VCC;
154                                           AUT_TEST=IDL; end if;
155
156           % wystawienie komendy odczytu z PS2/2, czeka na WAIT=H  %
157           when PSA => if (WAIT_PS2==GND) then
158                   A[]=ps2_addr; IORQ=GND; RD=GND; WR=VCC;
159                   DIB[]=DI[];             AUT_TEST=PSB;
160               else
161                   A[]=ps2_addr; IORQ=GND; RD=GND; WR=VCC;
162                                           AUT_TEST=PSA; end if;
163
164           % oczekiwanie na WAIT=L, aby wczytac dane do DIB        %
165           when PSB => if (WAIT_PS2==VCC) then
```

```
166                 A[]=lpt_addr; IORQ=VCC; RD=VCC; WR=VCC;
167                 DIB[]=DI[];             AUT_TEST=SWA;
168             else
169                 A[]=ps2_addr; IORQ=GND; RD=GND; WR=VCC;
170                 DIB[]=DI[];             AUT_TEST=PSB; end if;
171
172         % zmiana sygnalow na komende zapisu do LPT              %
173         when SWA =>
174                 A[]=lpt_addr; IORQ=GND; RD=VCC; WR=GND;
175                 DO[]=DIB[]; DIB[]=DIB[]; AUT_TEST=LPA;
176
177         % wystawienie komendy zapisu na LPT, czeka na WAIT=H    %
178         when LPA => if (WAIT_LPT==GND) then
179                 A[]=lpt_addr; IORQ=GND; RD=VCC; WR=GND;
180                 DO[]=DIB[]; DIB[]=DIB[]; AUT_TEST=LPB;
181             else
182                 A[]=lpt_addr; IORQ=GND; RD=VCC; WR=GND;
183                 DO[]=DIB[]; DIB[]=DIB[]; AUT_TEST=LPA; end if;
184
185         % oczekiwanie na WAIT=L (koniec wydruku)                %
186         when LPB => if (WAIT_LPT==VCC) then
187                 A[]=lpt_addr; IORQ=VCC; RD=VCC; WR=VCC;
188                 DIB[]=DIB[];            AUT_TEST=AEN;
189             else
190                 A[]=lpt_addr; IORQ=GND; RD=VCC; WR=GND;
191                 DO[]=DIB[]; DIB[]=DIB[]; AUT_TEST=LPB; end if;
192
193         % oczekiwanie na ustawienie przelacznika w pozycje GND  %
194         when AEN => if (%SW1B==GND%VCC) then
195                 A[]=ps2_addr; IORQ=VCC; RD=VCC; WR=VCC;
196                                         AUT_TEST=IDL;
197             else
198                 A[]=lpt_addr; IORQ=VCC; RD=VCC; WR=VCC;
199                 DIB[]=DIB[];            AUT_TEST=AEN; end if;
200       end case;
201
202   END;
```

```
TITLE "PS/2 INPUT";
%----------------------------------------------------------------------
    PS/2 input module : Communication between PS/2 bus and System
                        bus.
%----------------------------------------------------------------------
INCLUDE "SC_TO_U2.inc";

CONSTANT ps2_addr = H"50";              -- Keyboard address in I/O

SUBDESIGN PS2_IN
(
    GEN                     : input;    -- 20MHz clock
    RESET                   : input;    -- Reset signal from uC

    -- Bus
    ADDR[7..0]              : input;    -- Address bus
    DATA[7..0]              : output;   -- Data bus
    WAIT                    : output;   -- Wait signal
    IORQ                    : input;    -- I/O request signal
    RD                      : input;    -- Read signal

    -- PS/2 debug
    PS2_DEBUG_PORT[7..0]    : output;   -- Output port for debugging

    -- PS/2 connection port
    PDATA_IN[7..0]          : input;    -- PS/2 bus input port
)
VARIABLE
    -- Bus
    CMD_READ                : DFF;      -- Read command from uC
    KEY_Z[7..0]             : DFF;      -- Sign byte from keyboard
    KEY_D0[7..0]            : DFF;      -- First number
    KEY_D1[7..0]            : DFF;      -- Second number
    KEY_D2[7..0]            : DFF;      -- Third number
    SIG_WAIT                : DFF;      -- Wait signal for uC
    DFF_DATA[7..0]          : DFF;      -- Register for final U2 number
    CNT_DATA[7..0]          : DFF;      -- Wait extension counter
    S2U                     : SC_TO_U2; -- Scancode to U2 decoder
    -- Secure input dff's
    DFF_1_PS2_CLK           : DFF;      -- Input dff #1 for clock
    DFF_2_PS2_CLK           : DFF;      -- Input dff #2 for clock
    DFF_1_PS2_DATA          : DFF;      -- Input dff #1 for data
    DFF_2_PS2_DATA          : DFF;      -- Input dff #2 for data
    -- Data recieving
    PS2_FRAME[10..0]        : DFF;      -- Recieved frame
    PS2_CNT_FRAME[3..0]     : DFF;      -- Frame bit's counter
    PS2_CNT_BYTE[1..0]      : DFF;      -- ZDDD byte counter
    PS2_BYTE[7..0]          : DFF;      -- Data from PS/2 frame -
hardwired
    PS2_BYTE_READY          : DFF;      -- Data byte is ready flag
    -- Debug
    D_PS2_DEBUG_PORT[7..0]  : DFF;      -- Debugging register

    -- Machines
```

```
    -- Machine for PS/2 bus communication
    AUT_PS2     : machine of bits (Q[1..0])
                  with states (S0=B"00", S1=B"01",
                               S2=B"10", S3=B"11");

    -- Machine for system bus communication
    AUT_PS2_BUS : machine of bits (QB[2..0])
                  with states (SPB0=B"000", SPB1=B"001", SPB2=B"010",
                               SPB3=B"011", SPB4=B"100", SPB5=B"101",
                               SPB6=B"110", SPB7=B"111");


BEGIN
    % assign global clock %
    DFF_1_PS2_CLK.CLK       = GEN;
    DFF_2_PS2_CLK.CLK       = GEN;
    DFF_1_PS2_DATA.CLK      = GEN;
    DFF_2_PS2_DATA.CLK      = GEN;
    PS2_FRAME[].CLK         = GEN;
    PS2_CNT_FRAME[].CLK     = GEN;
    PS2_CNT_BYTE[].CLK      = GEN;
    D_PS2_DEBUG_PORT[].CLK  = GEN;
    PS2_BYTE[].CLK          = GEN;
    PS2_BYTE_READY.CLK      = GEN;
    CMD_READ.CLK            = GEN;
    KEY_Z[].CLK             = GEN;
    KEY_D0[].CLK            = GEN;
    KEY_D1[].CLK            = GEN;
    KEY_D2[].CLK            = GEN;
    DFF_DATA[].CLK          = GEN;
    SIG_WAIT.CLK            = GEN;
    CNT_DATA[].CLK          = GEN;
    S2U.GEN                 = GEN;
    AUT_PS2.CLK             = GEN;
    AUT_PS2_BUS.CLK         = GEN;

    % assign global reset %
    S2U.RESET               = RESET;
    DFF_1_PS2_CLK.CLRN      = RESET;
    DFF_2_PS2_CLK.CLRN      = RESET;
    DFF_1_PS2_DATA.CLRN     = RESET;
    DFF_2_PS2_DATA.CLRN     = RESET;
    PS2_FRAME[].CLRN        = RESET;
    PS2_CNT_FRAME[].CLRN    = RESET;
    PS2_CNT_BYTE[].CLRN     = RESET;
    D_PS2_DEBUG_PORT[].CLRN = RESET;
    PS2_BYTE[].CLRN         = RESET;
    PS2_BYTE_READY.CLRN     = RESET;
    CMD_READ.CLRN           = RESET;
    KEY_Z[].CLRN            = RESET;
    KEY_D0[].CLRN           = RESET;
    KEY_D1[].CLRN           = RESET;
    KEY_D2[].CLRN           = RESET;
    DFF_DATA[].CLRN         = RESET;
    SIG_WAIT.CLRN           = RESET;
```

```
    CNT_DATA[].CLRN          = RESET;
    AUT_PS2.RESET            = !RESET;
    AUT_PS2_BUS.RESET        = !RESET;

    % connect variables %
    WAIT                     = SIG_WAIT.Q;

    S2U.SC_Z[]               = KEY_Z[];
    S2U.SC_D0[]              = KEY_D0[];
    S2U.SC_D1[]              = KEY_D1[];
    S2U.SC_D2[]              = KEY_D2[];

    case PS2_CNT_BYTE[] is
        when B"00"  => PS2_DEBUG_PORT[] = 0;
        when B"01"  => PS2_DEBUG_PORT[] = 1;
        when B"10"  => PS2_DEBUG_PORT[] = 3;
        when B"11"  => PS2_DEBUG_PORT[] = 7;
    end case;

    DATA[] = DFF_DATA[];

    % Hardwired byte from frame %
    PS2_BYTE[] = (PS2_FRAME8,PS2_FRAME7,PS2_FRAME6,PS2_FRAME5,PS2_FRAME4,
PS2_FRAME3,PS2_FRAME2,PS2_FRAME1);
    % Secure input of ps2 clock %
    DFF_1_PS2_CLK.D = PDATA_IN1;
    DFF_2_PS2_CLK.D = DFF_1_PS2_CLK.Q;

    % Secure input of ps2 data %
    DFF_1_PS2_DATA.D = PDATA_IN0;
    DFF_2_PS2_DATA.D = DFF_1_PS2_DATA.Q;

    % Bus %
    if(ADDR[] == ps2_addr & IORQ == GND & RD == GND) then
            % Command read %
            CMD_READ.D = VCC;
    else
            CMD_READ.D = GND;
    end if;

    case AUT_PS2_BUS is
        % Wait for command from uC %
        when SPB0 => CMD_READ.D = CMD_READ.Q;
                     KEY_Z[] = B"00000000";
                     KEY_D0[] = B"00000000";
                     KEY_D1[] = B"00000000";
                     KEY_D2[] = B"00000000";
                     PS2_CNT_BYTE[] = B"00";
                     CNT_DATA[] = B"00000000";
                     DFF_DATA[] = 0;

                     if(CMD_READ.Q == 1) then
                        SIG_WAIT.D = GND;
                        -- activate reciever
                        AUT_PS2_BUS = SPB1;
```

```
                else
                   -- unblock after reset
                   SIG_WAIT.D = VCC;
                   AUT_PS2_BUS = SPB0;
                end if;
        % Wait for frame %
        when SPB1 => CMD_READ.D = CMD_READ.Q;
                KEY_Z[] = KEY_Z[];
                KEY_D0[] = KEY_D0[];
                KEY_D1[] = KEY_D1[];
                KEY_D2[] = KEY_D2[];
                CNT_DATA[] = CNT_DATA[];
                DFF_DATA[] = DFF_DATA[];
                PS2_CNT_BYTE[] = PS2_CNT_BYTE[];

                if(PS2_BYTE_READY.Q == 1) then
                   AUT_PS2_BUS = SPB2;
                else
                   AUT_PS2_BUS = SPB1;
                end if;
        % Check if frame data is "F0" %
        when SPB2 => CMD_READ.D = CMD_READ.Q;
                KEY_Z[] = KEY_Z[];
                KEY_D0[] = KEY_D0[];
                KEY_D1[] = KEY_D1[];
                KEY_D2[] = KEY_D2[];
                CNT_DATA[] = CNT_DATA[];
                DFF_DATA[] = DFF_DATA[];
                PS2_CNT_BYTE[] = PS2_CNT_BYTE[];

                if(PS2_BYTE[] == B"11110000") then
                   AUT_PS2_BUS = SPB3;
                else
                   AUT_PS2_BUS = SPB1;
                end if;
        % Wait for data frame ( after "F0" comes scancode of key ) %
        when SPB3 => CMD_READ.D = CMD_READ.Q;
                KEY_Z[] = KEY_Z[];
                KEY_D0[] = KEY_D0[];
                KEY_D1[] = KEY_D1[];
                KEY_D2[] = KEY_D2[];
                CNT_DATA[] = CNT_DATA[];
                DFF_DATA[] = DFF_DATA[];
                PS2_CNT_BYTE[] = PS2_CNT_BYTE[];

                if(PS2_BYTE_READY.Q == 1) then
                   AUT_PS2_BUS = SPB4;
                else
                   AUT_PS2_BUS = SPB3;
                end if;
        % Write key scancode into correct buffer %
        when SPB4 => CMD_READ.D = CMD_READ.Q;
                KEY_Z[] = KEY_Z[];
                KEY_D0[] = KEY_D0[];
                KEY_D1[] = KEY_D1[];
```

```
                KEY_D2[] = KEY_D2[];
                CNT_DATA[] = CNT_DATA[];
                DFF_DATA[] = DFF_DATA[];
                PS2_CNT_BYTE[] = PS2_CNT_BYTE[]+1;

                case PS2_CNT_BYTE[] is
                   when 0 => KEY_Z[] = PS2_BYTE[];
                             AUT_PS2_BUS = SPB1;
                   when 1 => KEY_D2[] = PS2_BYTE[];
                             AUT_PS2_BUS = SPB1;
                   when 2 => KEY_D1[] = PS2_BYTE[];
                             AUT_PS2_BUS = SPB1;
                   when 3 => KEY_D0[] = PS2_BYTE[];
                             AUT_PS2_BUS = SPB5;
                end case;
        % Convert ZDDD to U2 and put it on data bus %
        when SPB5 => CMD_READ.D = CMD_READ.Q;
                KEY_Z[] = KEY_Z[];
                KEY_D0[] = KEY_D0[];
                KEY_D1[] = KEY_D1[];
                KEY_D2[] = KEY_D2[];
                CNT_DATA[] = CNT_DATA[];
                PS2_CNT_BYTE[] = PS2_CNT_BYTE[];

                DFF_DATA[] = S2U.U2_LS[];

                if( CNT_DATA[] < 20 ) then
                   AUT_PS2_BUS = SPB6;
                else
                   AUT_PS2_BUS = SPB7;
                end if;

        % Extend wait signal %
        when SPB6 => CMD_READ.D = CMD_READ.Q;
                KEY_Z[] = KEY_Z[];
                KEY_D0[] = KEY_D0[];
                KEY_D1[] = KEY_D1[];
                KEY_D2[] = KEY_D2[];
                CNT_DATA[] = CNT_DATA[]+1;
                PS2_CNT_BYTE[] = PS2_CNT_BYTE[];
                DFF_DATA[] = DFF_DATA[];

                AUT_PS2_BUS = SPB5;
        % Remove Wait, and wait for processor read %
        when SPB7 => SIG_WAIT.D = VCC;
                CMD_READ.D = CMD_READ.Q;
                KEY_Z[] = KEY_Z[];
                KEY_D0[] = KEY_D0[];
                KEY_D1[] = KEY_D1[];
                KEY_D2[] = KEY_D2[];
                CNT_DATA[] = CNT_DATA[];
                PS2_CNT_BYTE[] = PS2_CNT_BYTE[];

                DFF_DATA[] = DFF_DATA[];
```

```
                    if(IORQ == VCC & RD == VCC)then
                        AUT_PS2_BUS = SPB0;
                    else
                        AUT_PS2_BUS = SPB7;
                    end if;

    end case;

    case AUT_PS2 is
        % PS2 clock down - start bit %
        when S0 =>  PS2_CNT_FRAME[] = B"0000";
                    PS2_FRAME[] = B"00000000000";
                    PS2_BYTE_READY.D = GND;

                    if(DFF_1_PS2_CLK.Q == 0 & DFF_2_PS2_CLK.Q == 1) then
                        AUT_PS2 = S2;
                    else
                        AUT_PS2 = S0;
                    end if;
        % Wait for clock falling edge %
        when S1 =>  PS2_CNT_FRAME[] = PS2_CNT_FRAME[];
                    PS2_FRAME[] = PS2_FRAME[];
                    PS2_BYTE_READY.D = GND;

                    if(DFF_1_PS2_CLK.Q == 0 & DFF_2_PS2_CLK.Q == 1) then
                        AUT_PS2 = S2;
                    else
                        AUT_PS2 = S1;
                    end if;
        % Read bit into frame buffer %
        when S2 =>  PS2_CNT_FRAME[] = PS2_CNT_FRAME[]+1;
                    PS2_FRAME[] = (DFF_2_PS2_DATA.Q, PS2_FRAME[10..1]);
                    PS2_BYTE_READY.D = GND;

                    if(PS2_CNT_FRAME[] < 10) then
                        AUT_PS2 = S1;
                    else
                        AUT_PS2 = S3;
                    end if;
        % 11 bit frame is ready %
        when S3 =>  PS2_FRAME[] = PS2_FRAME[];
                    PS2_CNT_FRAME[] = PS2_CNT_FRAME[];

                    PS2_BYTE_READY.D = VCC;
                    -- reciever off
                    AUT_PS2 = S0;
    end case;

END;
```

```
TITLE "Scancode to U2 conversion";
%--------------------------------------------------------------------
    PS/2 input module : ZDDD -> U2 converter.
%--------------------------------------------------------------------
INCLUDE "SC_TO_BCD.inc";

SUBDESIGN SC_TO_U2
(
    GEN                 : input;        -- 20MHz clock
    RESET               : input;        -- Reset signal from uC

    SC_Z[7..0]          : input;        -- Z
    SC_D0[7..0]         : input;        -- D
    SC_D1[7..0]         : input;        -- D
    SC_D2[7..0]         : input;        -- D

    U2_MS[7..0]         : output;       -- Output most significant (not
used)
    U2_LS[7..0]         : output;       -- Output less significant
)
VARIABLE
    SIGN                : DFF;          -- Sign of number
    S2B0                : SC_TO_BCD;    -- Digit 0 decoder
    S2B1                : SC_TO_BCD;    -- Digit 1 decoder
    S2B2                : SC_TO_BCD;    -- Digit 2 decoder
    BCD_D0[7..0]        : DFF;          -- Digit 0 in BCD
    BCD_D1[7..0]        : DFF;          -- Digit 1 in BCD
    BCD_D2[7..0]        : DFF;          -- Digit 2 in BCD
    D0[7..0]            : DFF;          -- Digit 0 multiplied
    D1[7..0]            : DFF;          -- Digit 1 multiplied
    D2[7..0]            : DFF;          -- Digit 2 multiplied
    BINARY[7..0]        : DFF;          -- Binary from ZDDD
    NEGATOR[8..0]       : DFF;          -- For binary negation
    NEGATED[8..0]       : DFF;          -- For binary negation
    NEGATION[8..0]      : DFF;          -- For binary negation
    RESULT[7..0]        : DFF;          -- Converter final output
BEGIN
    % Clock %
    SIGN.CLK            = GEN;
    BCD_D0[].CLK        = GEN;
    BCD_D1[].CLK        = GEN;
    BCD_D2[].CLK        = GEN;
    D0[].CLK            = GEN;
    D1[].CLK            = GEN;
    D2[].CLK            = GEN;
    BINARY[].CLK        = GEN;
    NEGATOR[].CLK       = GEN;
    NEGATED[].CLK       = GEN;
    NEGATION[].CLK      = GEN;
    RESULT[].CLK        = GEN;

    % Reset %
    SIGN.CLRN           = RESET;
    BCD_D0[].CLRN       = RESET;
    BCD_D1[].CLRN       = RESET;
```

```
    BCD_D2[].CLRN       = RESET;
    D0[].CLRN           = RESET;
    D1[].CLRN           = RESET;
    D2[].CLRN           = RESET;
    BINARY[].CLRN       = RESET;
    NEGATOR[].CLRN      = RESET;
    NEGATED[].CLRN      = RESET;
    NEGATION[].CLRN     = RESET;
    RESULT[].CLRN       = RESET;

    % SC to BCD %
    S2B0.SC[] = SC_D0[];
    BCD_D0[] = S2B0.BCD[];

    S2B1.SC[] = SC_D1[];
    BCD_D1[] = S2B1.BCD[];

    S2B2.SC[] = SC_D2[];
    BCD_D2[] = S2B2.BCD[];

    % Multiplication %
    D0[] = BCD_D0[];
    D1[] = BCD_D1[]*10;
    D2[] = BCD_D2[]*100;

    % Sum %
    BINARY[] = D0[] + D1[] + D2[];

    % Sign %
    if( SC_Z[] == H"4E" ) then
        -- Sign is "-" so convert to U2
        NEGATOR[] = B"100000000";
        NEGATED[] = (GND, BINARY[7..0]);
        NEGATION[] = NEGATOR[] - NEGATED[];
        RESULT[] = (NEGATION[7..0]);
    else
        RESULT[] = BINARY[];
    end if;

    % Return result %
    U2_LS[] = RESULT[];
    % Future extension? %
    U2_MS[] = (GND, GND, GND, GND, GND, GND, GND, GND);
END;
```

```
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 "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      WAIT        : 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",
75                                     D1A=B"0111",
76                                     D1B=B"1000",
77                                     D0A=B"1001",
78                                     D0B=B"1010",
79                                     LFA=B"1011",
80                                     LFB=B"1100",
81                                     WBU=B"1101");
82    BEGIN
83        % DEBUG %
84        V_BPRINT = BPRINT;
85        V_APRINT = APRINT;
86        V_CSTIME[] = CSTIME[];
87
88        % podlaczenie przerzutnikow DFF i automatow                 %
89        nWAIT_SYN.clk   = GEN;
90        nWAIT_SYN.clrn  = RESET;
91        nWAIT_SYN       = !WAIT_I;
92        LDATA[].clk     = GEN;
93        LDATA[].clrn    = RESET;
94        nSTROBE_SYN.clk = GEN;
95        nSTROBE_SYN.clrn= RESET;
96        CSTIME[].clk    = GEN;
97        CSTIME[].clrn   = RESET;
98        AUT_PRINT.clk   = GEN;
99        AUT_PRINT.reset = !RESET;
100       AUT_BUS.clk     = GEN;
101       AUT_BUS.reset   = !RESET;
102
103       % podlaczenie ukladu konwersji U2->ASCII(Z,D2,D1,D0)        %
104       U2A.U2I[]       = DIN[];
105       Z[]             = U2A.Z[];
106       D2[]            = U2A.D2[];
107       D1[]            = U2A.D1[];
108       D0[]            = U2A.D0[];
109
110       % podlaczenie linii komunikacji z laczem LPT                %
```

```
111       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       WAIT            = !nWAIT_SYN;
136       %WAIT           = 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 "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;
```

```
 1   TITLE "Sumator 2-cyfrowych liczb dziesiêtnych";
 2   INCLUDE "bcd_blok.inc";
 3   SUBDESIGN bcd_sumator
 4   (
 5       XX[7..0],YY[7..0],CIN : input;
 6       ZZ[7..0],NAD : output;
 7   )
 8   VARIABLE
 9       CYF_A,CYF_B : BCD_BLOK;
10   BEGIN
11       CYF_A.WA[]=XX[ 3.. 0]; CYF_A.WB[]=YY[ 3.. 0]; CYF_A.CI=CIN;
12       CYF_B.WA[]=XX[ 7.. 4]; CYF_B.WB[]=YY[ 7.. 4]; CYF_B.CI=CYF_A.
    CO;
13       ZZ[]=(CYF_B.WY[],CYF_A.WY[]);
14       NAD=CYF_B.CO;
15   END;
```

```
1   TITLE "BCD_BLOK";
2   SUBDESIGN bcd_blok
3   (
4       WA[3..0],WB[3..0],CI : input;
5       WY[3..0],CO : output;
6   )
7   VARIABLE
8       SUMA[4..0],WYNIK[4..0] : NODE;
9   BEGIN
10      SUMA[] = (0,WA[])+(0,WB[])+(0,0,0,0,CI);
11      if SUMA[]>9 then WYNIK[]=SUMA[]+6; % korekcja wyniku %
12                  else WYNIK[]=SUMA[]; % wynik bez korekcji %
13      end if;
14      CO=WYNIK[4]; WY[]=WYNIK[3..0];
15  END;
```