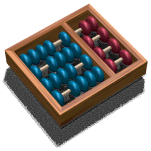

UNICAMP - Instituto de Computação

MC613 - Laboratório de Circuitos Lógicos

Tutorial - Controladores de teclado e de mouse

Monitores: Caio Hoffman & George Gondim
Professores: Mario Lúcio Côrtes & Guido Costa Souza de Araújo

Maio de 2012

Sumário

Lista de Tabelas

Conteúdo do Tutorial

1. Menção ao protocolo PS2 e a seu componente PS2_IOBASE
2. Utilização do mouse
   (a) Componente MOUSE_CTRL
   (b) Genéricos, sinais de entrada e de saída.
3. Utilização do teclado
   (a) Componente KBDEX_CTRL
   (b) Genéricos, sinais de entrada e de saída.
4. Exemplo de utilização
5. Exercícios propostos.

Materiais

- Quartus 9.1 sp2 da Altera.

1 Introdução

O objetivo desse tutorial é fazer uma descrição muito breve de como interpretar os sinais vindos de teclado e de mouse através da porta PS2 da DE1 da Altera, por meio dos componentes ***mouse_ctrl*** (controlador de mouse) e ***kbdex_ctrl*** (controlador de teclado). Primeiramente, serão apresentados os componentes, seus sinais de entrada e de saída. Em seguida, será exibido um pequeno exemplo para cada componente.

OBSERVAÇÃO: Recomendamos fortemente que todos os alunos realizem a atribuição de pinos da DE1 usando o arquivo ***DE1_pin_assignments.csv*** localizado na sessão "Material Complementar" do site da disciplina.

1.1 Componente PS2_IOBASE

Tanto o controlador de teclado quanto o controlador de mouse fazem uso do componente ***ps2_iobase***, portanto, tenha certeza de que o arquivo VHDL deste componente esteja incluso no seu projeto. Este componente implementa um módulo controlador do protocolo PS2. Explicar o que é este protocolo e como funciona o seu controlador foge do escopo deste tutorial, porém os alunos interessados são encorajados a estudá-los. É necessário, no entanto, que o aluno saiba que o protocolo PS2 faz uso de barramentos bidirecionais *i.e.* sinais tanto de entrada quanto de saída.

1.2 Componente MOUSE_CTRL

Explicaremos aqui o significado e o modo de uso de cada sinal e de cada genérico presente na entidade do componente. Observe que o protocolo de comunicação usado por esse módulo é o protocolo PS2 cujo controlador é o ***ps2_iobase***.

1.2.1 Genéricos

O componente faz uso de um único genérico o *clkfreq* que determina a frequência em kHz do *clock* de funcionamento do componente, recomendamos **fortemente** que seja usado $24000kHz$ que é sinal de atribuição CLOCK_24 da DE1, observe que este sinal

possui dois pinos, logo ele precisa ser declarado como um vetor.

1.2.2 Sinais Bidirecionais

Existem dois sinais bidirecionais: o sinal *ps2_data* são os bits transferidos serialmente para e da porta PS2, este sinal deve ser conectado aos pinos referidos pelo sinal de atribuição PS2_DATA; o *ps2_clk* é o *clock* de funcionamento do controlador PS2, deve ser conectado aos pinos do sinal de atribuição PS2_CLK.

Tabela 1: Associação de pinos bidirecionais do ***mouse_ctrl***.

Sinal bidirecionais	Nome do sinal de atribuição	Tipo do sinal
ps2_data	PS2_DATA	STD_LOGIC
ps2_clk	PS2_CLK	STD_LOGIC

1.2.3 Sinais de Entrada

São eles: o *clock* do sistema *clk* que deve possuir a mesma frequência atribuida ao *clkfreq*, como mencionado anteriormente, é recomendado que se use $24000kHz$; o sinal de habilitação *en* e o sinal de reinicialização *resetn* que é ativo baixo. O sinais *en* e *resetn* podem ser atribuidos a qualquer pino desejado.

Tabela 2: Associação de pinos dos sinais de entrada ***mouse_ctrl***.

Sinal de entrada	Nome do sinal de atribuição	Tipo do sinal de atribuição
clk	CLOCK_24	STD_LOGIC_VECTOR(1 downto 0)
en	-	-
resetn	-	-

1.2.4 Sinais de Saída

Não há necessidade de atribuição especial de pinos aos sinais de saída. São eles: o *newdata* que é alto quando há um novo pacote para ser recebido do mouse; o vetor *bt_on* corresponde aos botões do mouse – *bt_on(0)* é o botão esquerdo, *bt_on(1)* é o direito e *bt_on(2)* é o do meio – e, quando pressionados, temos um sinal alto na célula correspondente do vetor; Os sinais *dx* e *dy* determinam, em complemento de dois de 8 bits, a variação das coordenadas da posição do mouse; *ox* e *oy* são altos quando há ocorrência de

overflow em algumas das coordenadas de deslocamento *i.e.* o valor do deslocamento em x e y saiu do intervalo $[-2^8, 2^8 - 1]$; e, finalmente, o sinal *wheel* representa, em complemento de 2 de 4 bits, o valor do deslocamento da roda do mouse em relação à posição anterior. O código ***ps2_mouse_test*** será utilizado como exemplo para nos familiarizarmos com o controlador de mouse.

1.3 Componente KBDEX_CTRL

Neste sessão detalharemos o uso do controlador de teclado. Este módulo também faz uso do controlador PS2 ***ps2_iobase***.

1.3.1 Genéricos

Este componente possui um único genérico que é o mesmo explicado na sessão 1.2.1.

1.3.2 Sinais Bidirecionais

Os sinais bidirecionais que este componente usa são os mesmos explicados na sessão 1.2.2.

1.3.3 Sinais de Entrada

Três dos quatros sinais de entrada do componente são idênticos aos do controlador de mouse são eles: *clk*, *en* e *resetn* (ver sessão 1.2.3). O quarto sinal de entrada o *lights* é um vetor que determina o estado dos LEDs do teclado, *light(0)* é o do *scroll lock*, *light(1)* é o do *nunlock* e o *lights(2)* o do *capslock*, no entanto, a versão atual do componente **não suporta** que esses LEDs sejam ligados, portanto, mantenha-os em zero.

1.3.4 Sinais de Saída

Este controlador de teclado suporta até 3 teclas pressionadas ao mesmo tempo, o sinal de saída *key_on* é um vetor que indica se temos ou não teclas pressionadas, a célula 0 representa a primeira tecla pressinada, a 1 a segunda e a 2 a terceira. O sinal *key_code* é o vetor que indica o código de leitura (*scancode*)da tecla pressionada, os bits 15-0 são para

a primeira tecla pressionada, os 31-16 para a segunda e os 47-32 para a terceira. Para o correto mapeamento das teclas do teclado e para familizarição com seu controlador, utilizaremos o código de exemplo ***ps2_kbd_test***.

2 Exemplos

2.1 Mouse

Para exemplificar o funcionamento do mouse, temos o código 4. Seu funcionamento é bem simples: é mostrado nos displays de 7 segmentos o valor da posição absoluta do mouse, sendo HEX3 e HEX2 a abscissa e a ordenada são HEX1 e HEX0; nos LEDs verdes 7-5 da DE1 é mostrado o sinal de saída *bt_on*; no LED vermelho 9 *ox* e no 7 *7*; o valor do deslocamento da wheel é mostrado em binário nos LEDs verdes 3-0. Observe que a constante *SENSIBILITY* (linha 67) serve determinarmos a sensibilidade do mouse *i.e.* determinarmos o quanto será necessário mover o mouse para alterarmos suas coordenadas. Observando o processo entre as linhas 91 e 106, notamos que ele é síncrono com a chegada de um novo pacote – sinal *signewdata* que é o sinal *newdata* de saída do mouse. Veja a sessão 1.2.4 para relembrar os sinais de saída do mouse.

DICA: Aos alunos que vão utilizar o mouse, este código de exemplo pode ser utilizado com poucas modificações.

2.2 Teclado

O exemplo para o teclado está no código 5. Este também é um circuito bem simples que, ignorando o funcionamento dos LEDs do teclado (linhas 90-111), o circuito simplesmente mostra no display de 7 segmentos o *scancode* da primeira tecla pressionada no teclado e atribui aos LEDs verdes 7-5 da DE1 o valor do sinal *key_on* (explicado na sessão 1.3.4).

DICA: Aos alunos que vão utilizar o teclado, este código de exmplo pode ser utilizado para determinação do valor dos scancodes das teclas que vocês utilizarão.

3 Exercício Propostos

1. Verifique o funcionamento do exemplo do mouse na DE1. Aumente e diminua o valor da constante *SENSIBILITY* e observe o que acontece.

2. Qual a o *scancode* que codifica a letra minúscula 'ç'? E qual codifica a seta para baixo? Qual codifica a letra maíuscula 'A' (não use o fixa/caps para este exercício).

3. Altere o código do exemplo do teclado para que, usando os switchs da DE1, você possa verificar o valor de qualquer uma das 3 teclas pressionadas através do display de 7 segmentos.

A Códigos VHDL

Todos os códigos desta sessão estão na página da disciplina em "Material Complementar".

A.1 Componente PS2_IOBASE

Código VHDL 1: ps2_iobase.vhd

```
1  ------------------------------------------------------------------------------
2  -- Title      : MC613
3  -- Project    : PS2 Basic Protocol
4  -- Details    : www.ic.unicamp.br/~corte/mc613/
5  --              www.computer-engineering.org/ps2protocol/
6  ------------------------------------------------------------------------------
7  -- File       : ps2_base.vhd
8  -- Author     : Thiago Borges Abdnur
9  -- Company    : IC - UNICAMP
10 -- Last update: 2010/04/12
11 ------------------------------------------------------------------------------
12 -- Description:
13 -- PS2 basic control
14 ------------------------------------------------------------------------------
15
16 LIBRARY ieee;
17 USE ieee.std_logic_1164.all;
18 USE ieee.numeric_std.all;
19
20 entity ps2_iobase is
21   generic(
22     clkfreq : integer -- This is the system clock value in kHz
23   );
24   port(
25     ps2_data  : inout std_logic;  -- PS2 data pin
26     ps2_clk   : inout std_logic;  -- PS2 clock pin
27     clk       : in  std_logic;    -- system clock (same frequency as defined in
28                                   --  'clkfreq' generic)
29     en        : in  std_logic;    -- Enable
30     resetn    : in  std_logic;    -- Reset when '0'
31     idata_rdy : in  std_logic;    -- Rise this to signal data is ready to be sent
32                                   --  to device
33     idata     : in  std_logic_vector(7 downto 0); -- Data to be sent to device
34     send_rdy  : out std_logic;    -- '1' if data can be sent to device (wait for
35                                   --  this before rising 'idata_rdy'
36     odata_rdy : out std_logic;    -- '1' when data from device has arrived
37     odata     : out std_logic_vector(7 downto 0) -- Data from device
38   );
39 end;
40
41 architecture rtl of ps2_iobase is
42   constant CLKSSTABLE : integer := clkfreq / 150;
43
44   signal sdata, hdata : std_logic_vector(7 downto 0);
45   signal sigtrigger, parchecked, sigsending,
46          sigsendend, sigclkreleased, sigclkheld : std_logic;
47 begin
48   -- Trigger for state change to eliminate noise
49   process(clk, ps2_clk, en, resetn)
50     variable fcount, rcount : integer range CLKSSTABLE downto 0;
51   begin
52     if(rising_edge(clk) and en = '1') then
53       -- Falling edge noise
54       if ps2_clk = '0' then
```

```
55          rcount := 0;
56          if fcount >= CLKSSTABLE then
57            sigtrigger <= '1';
58          else
59            fcount := fcount + 1;
60          end if;
61        -- Rising edge noise
62        elsif ps2_clk = '1' then
63          fcount := 0;
64          if rcount >= CLKSSTABLE then
65            sigtrigger <= '0';
66          else
67            rcount := rcount + 1;
68          end if;
69        end if;
70      end if;
71      if resetn = '0' then
72        fcount := 0;
73        rcount := 0;
74        sigtrigger <= '0';
75      end if;
76    end process;
77
78    FROMPS2:
79    process(sigtrigger, sigsending, resetn)
80      variable count : integer range 0 to 11;
81    begin
82      if(rising_edge(sigtrigger) and sigsending = '0') then
83        if count > 0 and count < 9 then
84          sdata(count - 1) <= ps2_data;
85        end if;
86        if count = 9 then
87          if (not (sdata(0) xor sdata(1) xor sdata(2) xor sdata(3)
88           xor sdata(4) xor sdata(5) xor sdata(6) xor sdata(7))) = ps2_data then
89            parchecked <= '1';
90          else
91            parchecked <= '0';
92          end if;
93        end if;
94        count := count + 1;
95        if count = 11 then
96          count := 0;
97          parchecked <= '0';
98        end if;
99      end if;
100     if resetn = '0' or sigsending = '1' then
101       sdata <= (others => '0');
102       parchecked <= '0';
103       count := 0;
104     end if;
105   end process;
106
107   odata_rdy <= en and parchecked;
108
109   odata <= sdata;
110
111   -- Edge triggered send register
112   process(idata_rdy, sigsendend, resetn)
113   begin
114     if(rising_edge(idata_rdy)) then
115       sigsending <= '1';
116     end if;
117     if resetn = '0' or sigsendend = '1' then
118       sigsending <= '0';
119     end if;
120   end process;
121
122   -- Wait for at least 11ms before allowing to send again
123   process(clk, sigsending, resetn)
124     -- clkfreq is the number of clocks within a milisecond
```

```
125     variable countclk : integer range 0 to (12 * clkfreq);
126   begin
127     if(rising_edge(clk) and sigsending = '0') then
128       if countclk = (11 * clkfreq) then
129         send_rdy <= '1';
130       else
131         countclk := countclk + 1;
132       end if;
133     end if;
134     if sigsending = '1' then
135       send_rdy <= '0';
136       countclk := 0;
137     end if;
138     if resetn = '0' then
139       send_rdy <= '1';
140       countclk := 0;
141     end if;
142   end process;
143
144   -- Host input data register
145   process(idata_rdy, sigsendend, resetn)
146   begin
147     if(rising_edge(idata_rdy)) then
148       hdata <= idata;
149     end if;
150     if resetn = '0' or sigsendend = '1' then
151       hdata <= (others => '0');
152     end if;
153   end process;
154
155   -- PS2 clock control
156   process(clk, sigsendend, resetn)
157     constant US100CNT : integer := clkfreq / 10;
158
159     variable count : integer range 0 to US100CNT + 101;
160   begin
161     if(rising_edge(clk) and sigsending = '1') then
162       if count < US100CNT + 50 then
163         count := count + 1;
164         ps2_clk <= '0';
165         sigclkreleased <= '0';
166         sigclkheld <= '0';
167       elsif count < US100CNT + 100 then
168         count := count + 1;
169         ps2_clk <= '0';
170         sigclkreleased <= '0';
171         sigclkheld <= '1';
172       else
173         ps2_clk <= 'Z';
174         sigclkreleased <= '1';
175         sigclkheld <= '0';
176       end if;
177     end if;
178     if resetn = '0' or sigsendend = '1' then
179       ps2_clk <= 'Z';
180       sigclkreleased <= '1';
181       sigclkheld <= '0';
182       count := 0;
183     end if;
184   end process;
185
186   -- Sending control
187   TOPS2:
188   process(sigtrigger, sigsending, sigclkheld, sigclkreleased, resetn)
189     variable count : integer range 0 to 11;
190   begin
191     if(rising_edge(sigtrigger) and sigclkreleased = '1'
192      and sigsending = '1') then
193       if count >= 0 and count < 8 then
194         ps2_data <= hdata(count);
```

```
195         sigsendend <= '0';
196       end if;
197       if count = 8 then
198         ps2_data <= (not (hdata(0) xor hdata(1) xor hdata(2) xor hdata(3)
199          xor hdata(4) xor hdata(5) xor hdata(6) xor hdata(7)));
200         sigsendend <= '0';
201       end if;
202       if count = 9 then
203         ps2_data <= 'Z';
204         sigsendend <= '0';
205       end if;
206       if count = 10 then
207         ps2_data <= 'Z';
208         sigsendend <= '1';
209         count := 0;
210       end if;
211       count := count + 1;
212     end if;
213     if sigclkheld = '1' then
214       ps2_data <= '0';
215       sigsendend <= '0';
216       count := 0;
217     end if;
218     if resetn = '0' or sigsending = '0' then
219       ps2_data <= 'Z';
220       sigsendend <= '0';
221       count := 0;
222     end if;
223   end process;
224
225 end rtl;
```

A.2 Componente MOUSE_CTRL

Código VHDL 2: mouse_ctrl.vhd

```
1  ----------------------------------------------------------------------------------
2  -- Title      : MC613
3  -- Project    : Mouse Controller
4  -- Details    : www.ic.unicamp.br/~corte/mc613/
5  --              www.computer-engineering.org/ps2protocol/
6  ----------------------------------------------------------------------------------
7  -- File       : mouse_ctrl.vhd
8  -- Author     : Thiago Borges Abdnur
9  -- Company    : IC - UNICAMP
10 -- Last update: 2010/04/12
11 ----------------------------------------------------------------------------------
12 -- Description:
13 -- PS2 mouse basic I/O control
14 ----------------------------------------------------------------------------------
15
16 LIBRARY ieee;
17 USE ieee.std_logic_1164.all;
18 USE ieee.numeric_std.all;
19
20 entity mouse_ctrl is
21   generic(
22     -- This is the system clock value in kHz. Should be at least 1MHz.
23     --  Recommended value is 24000 kHz (CLOCK_24 (DE1 pins: PIN_A12 and
24     --  PIN_B12))
25     clkfreq : integer
26   );
27   port(
28     ps2_data  : inout std_logic;  -- PS2 data pin
29     ps2_clk   : inout std_logic;  -- PS2 clock pin
30     clk       : in  std_logic;    -- system clock (same frequency as defined in
31                                   --  'clkfreq' generic)
32     en        : in  std_logic;    -- Enable
```

```
33      resetn     : in  std_logic;     -- Reset when '0'
34      newdata    : out std_logic;     -- Rises when a new data package has arrived
35                                      --  from the mouse
36      -- Mouse buttons state ('1' when pressed):
37      --  bt_on(0): Left mouse button
38      --  bt_on(1): Right mouse button
39      --  bt_on(2): Middle mouse button (if it exists)
40      bt_on      : out std_logic_vector(2 downto 0);
41
42      -- Signals that an overflow occured on this package in one of the
43      --  coordinates
44      ox, oy     : out std_logic;
45
46      -- New position relative to the last package. Nine bit values in two's
47      --  complement for each coordinate.
48      dx, dy     : out std_logic_vector(8 downto 0);
49
50      -- Wheel movement relative to last package. 4 bit value in two's
51      --  complement. Wheel up is a negative value, down is positive.
52      wheel      : out std_logic_vector(3 downto 0)
53    );
54  end;
55
56  architecture rtl of mouse_ctrl is
57    component ps2_iobase
58      generic(
59        clkfreq : integer -- This is the system clock value in kHz
60      );
61      port(
62        ps2_data  : inout std_logic;
63        ps2_clk   : inout std_logic;
64        clk       : in  std_logic;
65        en        : in  std_logic;
66        resetn    : in  std_logic;
67        idata_rdy : in  std_logic;
68        idata     : in  std_logic_vector(7 downto 0);
69        send_rdy  : out std_logic;
70        odata_rdy : out std_logic;
71        odata     : out std_logic_vector(7 downto 0)
72      );
73    end component;
74
75    signal sigsend, sigsendrdy, signewdata,
76           sigreseting, xn, yn, sigwheel : std_logic;
77    signal hdata, ddata : std_logic_vector(7 downto 0);
78  begin
79    ps2io : ps2_iobase generic map(clkfreq) port map(
80      ps2_data, ps2_clk, clk, en, resetn, sigsend, hdata,
81      sigsendrdy, signewdata, ddata
82    );
83
84    -- Send Reset to mouse
85    process(clk, en, resetn)
86      type rststatename is (
87        SETCMD, SEND, WAITACK, NEXTCMD, CLEAR
88      );
89
90      constant ncmd : integer := 11; -- Total number of commands to send
91      type commands is array (0 to ncmd - 1) of integer;
92      constant cmd : commands := (
93        16#FF#, 16#F5#, -- Reset and disable reporting
94        16#F3#, 200, 16#F3#, 100, 16#F3#, 80, 16#F2#, -- Wheel enabling
95        16#F6#, 16#F4# -- Restore defaults and enable reporting
96      );
97
98      variable state : rststatename;
99      variable count : integer range 0 to ncmd := 0;
100   begin
101     if(rising_edge(clk)) then
102       hdata <= X"00";
```

```
103       sigsend <= '0';
104       sigreseting <= '1';
105
106       case state is
107         when SETCMD =>
108           hdata <= std_logic_vector(to_unsigned(cmd(count), 8));
109           if sigsendrdy = '1' then
110             state := SEND;
111           else
112             state := SETCMD;
113           end if;
114
115         when SEND =>
116           hdata <= std_logic_vector(to_unsigned(cmd(count), 8));
117           sigsend <= '1';
118           state := WAITACK;
119
120         when WAITACK =>
121           if signewdata = '1' then
122             -- Wheel detection
123             if cmd(count) = 16#F2# then
124               -- If device ID is 0x00, it has no wheel
125               if ddata = X"00" then
126                 sigwheel <= '0';
127                 state := NEXTCMD;
128               -- If device ID is 0x03, it has a wheel
129               elsif ddata = X"03" then
130                 sigwheel <= '1';
131                 state := NEXTCMD;
132               end if;
133             else
134               state := NEXTCMD;
135             end if;
136           end if;
137
138         when NEXTCMD =>
139           count := count + 1;
140           if count = ncmd then
141             state := CLEAR;
142           else
143             state := SETCMD;
144           end if;
145
146         when CLEAR =>
147           sigreseting <= '0';
148           count := 0;
149       end case;
150     end if;
151     if resetn = '0' or en = '0' then
152       state := SETCMD;
153       count := 0;
154       sigwheel <= '0';
155     end if;
156   end process;
157
158   -- Get update packages
159   process(signewdata, sigreseting, en, resetn)
160     variable count : integer range 0 to 4;
161   begin
162     if(rising_edge(signewdata) and sigreseting = '0'
163      and en = '1') then
164       newdata <= '0';
165
166       case count is
167         when 0 =>
168           bt_on <= ddata(2 downto 0);
169           xn <= ddata(4);
170           yn <= ddata(5);
171           ox <= ddata(6);
172           oy <= ddata(7);
```

```
173
174         when 1 =>
175           dx <= xn & ddata;
176
177         when 2 =>
178           dy <= yn & ddata;
179
180         when 3 =>
181           wheel <= ddata(3 downto 0);
182
183         when others =>
184           NULL;
185       end case;
186       count := count + 1;
187       if (sigwheel = '0' and count > 2) or count > 3 then
188         count := 0;
189         newdata <= '1';
190       end if;
191     end if;
192     if resetn = '0' or en = '0' then
193       bt_on <= (others => '0');
194       dx <= (others => '0');
195       dy <= (others => '0');
196       wheel <= (others => '0');
197       xn <= '0';
198       yn <= '0';
199       ox <= '0';
200       oy <= '0';
201       count := 0;
202       newdata <= '0';
203     end if;
204   end process;
205 end rtl;
```

A.3 Componente KBDEX_CTRL

Código VHDL 3: kbdex_ctrl.vhd

```
1  ----------------------------------------------------------------------
2  -- Title      : MC613
3  -- Project    : Keyboard Controller
4  -- Details    : www.ic.unicamp.br/~corte/mc613/
5  --              www.computer-engineering.org/ps2protocol/
6  ----------------------------------------------------------------------
7  -- File       : kbdext_ctrl.vhd
8  -- Author     : Thiago Borges Abdnur
9  -- Company    : IC - UNICAMP
10 -- Last update: 2010/03/29
11 ----------------------------------------------------------------------
12 -- Description:
13 -- The keyboard controller receives serial data input from the device and
14 -- signals through 'key_on' when a key is pressed. Up to 3 keys can be pressed
15 -- simultaneously. The key code data is written to 'key_code':
16 --    First key pressed:
17 --      . key_code(15 downto 0) is set with data
18 --      . key_on(0) rises
19 --    First key released:
20 --      . key_on(0) falls
21 --
22 --    Second key pressed:
23 --      . key_code(31 downto 16) is set with data
24 --      . key_on(1) rises
25 --    Second key released:
26 --      . key_on(1) falls
27 --
28 --    Third key pressed:
29 --      . key_code(47 downto 32) is set with data
30 --      . key_on(2) rises
```

```
31  --     Third key released:
32  --       . key_on(2) falls
33  --
34  -- Remarks:
35  --     .clk: system clock frequency needs to be at least 10 MHz
36  --     .PrintScreen key signals as if two keys were pressed (E012 and E07C)
37  --     .Pause/Break key signals as if two keys were pressed (14 and 77), so it
38  --       is the same as if LCTRL (14) and NUNLOCK(77) were pressed.
39  --     .Currently it's not possible to write to the keyboard, turning its lights
40  --       on.
41  ----------------------------------------------------------------------------------
42
43  LIBRARY ieee;
44  USE ieee.std_logic_1164.all;
45  USE ieee.numeric_std.all;
46
47  entity kbdex_ctrl is
48    generic(
49      clkfreq : integer
50    );
51    port(
52      ps2_data  : inout std_logic;
53      ps2_clk   : inout std_logic;
54      clk       : in  std_logic;
55      en        : in  std_logic;
56      resetn    : in  std_logic;
57      lights    : in  std_logic_vector(2 downto 0); -- lights(Caps, Num, Scroll)
58      key_on    : out std_logic_vector(2 downto 0);
59      key_code  : out std_logic_vector(47 downto 0)
60    );
61  end;
62
63  architecture rtl of kbdex_ctrl is
64    component ps2_iobase
65      generic(
66        clkfreq : integer -- This is the system clock value in kHz
67      );
68      port(
69        ps2_data  : inout std_logic;
70        ps2_clk   : inout std_logic;
71        clk       : in  std_logic;
72        en        : in  std_logic;
73        resetn    : in  std_logic;
74        idata_rdy : in  std_logic;
75        idata     : in  std_logic_vector(7 downto 0);
76        send_rdy  : out std_logic;
77        odata_rdy : out std_logic;
78        odata     : out std_logic_vector(7 downto 0)
79      );
80    end component;
81
82    type statename is (
83      IDLE, FETCH, DECODE, CODE,
84      RELEASE, EXT0, EXT1, CLRDP
85    );
86
87    -- State machine signals
88    signal state, nstate  : statename;
89    signal sigfetch, sigfetched, sigext0,
90           sigrelease, sigselect, sigclear :  std_logic;
91
92    -- Datapath signals
93    signal newdata, selE0, relbt, selbt,
94           key0en, key1en, key2en,
95           key0clearn, key1clearn, key2clearn : std_logic;
96    signal ps2_code     : unsigned( 7 downto 0);
97    signal fetchdata    : unsigned( 7 downto 0);
98    signal upperdata    : unsigned( 7 downto 0);
99    signal datacode     : unsigned(15 downto 0);
100   signal key0code     : unsigned(15 downto 0);
```

```
101   signal key1code     : unsigned(15 downto 0);
102   signal key2code     : unsigned(15 downto 0);
103
104   -- Lights control
105   signal hdata        : std_logic_vector( 7 downto 0);
106   signal sigsend, sigsendrdy, sigsending,
107          siguplights  : std_logic;
108
109   -- PS2 output signals
110   signal ps2_dataout  : std_logic_vector(7 downto 0);
111   signal ps2_datardy  : std_logic;
112
113 begin
114   ps2_ctrl : ps2_iobase generic map(clkfreq) port map(
115     ps2_data, ps2_clk, clk, en, resetn, sigsend, hdata,
116     sigsendrdy, ps2_datardy, ps2_dataout
117   );
118
119   ps2_code <= unsigned(ps2_dataout);
120
121   -- State cicle
122   process(clk, resetn, sigsending)
123   begin
124     -- Change state on falling edge of ps2_clk
125     if(rising_edge(clk) and en = '1') then
126       state <= nstate;
127     end if;
128     if resetn = '0' or sigsending = '1' then
129       state <= IDLE;
130     end if;
131   end process;
132
133   -- Select next state
134   process(state, newdata)
135   begin
136     case state is
137       when IDLE   =>
138         if newdata = '1' then
139           nstate <= FETCH;
140         else
141           nstate <= IDLE;
142         end if;
143
144       when FETCH  =>
145         nstate <= DECODE;
146
147       when DECODE =>
148         if fetchdata = X"F0" then
149           nstate <= RELEASE;
150         elsif fetchdata = X"E0" then
151           nstate <= EXT0;
152         elsif fetchdata = X"E1" then
153           nstate <= EXT1;
154         else
155           nstate <= CODE;
156         end if;
157
158       when CODE =>
159         nstate <= CLRDP;
160
161       when RELEASE | EXT0 | EXT1 | CLRDP =>
162         nstate <= IDLE;
163     end case;
164   end process;
165
166   -- Current state output
167   process(state)
168   begin
169     sigfetch <= '0';
170     sigfetched <= '0';
```

```
171     sigext0 <= '0';
172     sigrelease <= '0';
173     sigselect <= '0';
174     sigclear <= '0';
175
176     case state is
177       when IDLE | EXT1 =>
178         NULL;
179
180       when FETCH =>
181         sigfetch <= '1';
182
183       when DECODE =>
184         sigfetched <= '1';
185
186       when CODE =>
187         sigselect <= '1';
188
189       when RELEASE =>
190         sigrelease <= '1';
191
192       when EXT0 =>
193         sigext0 <= '1';
194
195       when CLRDP =>
196         sigclear <= '1';
197     end case;
198   end process;
199
200   -- Fetched signal register
201   process(clk, resetn)
202   begin
203     if(rising_edge(clk) and sigfetch = '1') then
204       fetchdata <= ps2_code;
205     end if;
206     if resetn = '0' then
207       fetchdata <= X"00";
208     end if;
209   end process;
210
211   -- EXT0 selection (SR Latch)
212   process(sigext0, sigclear, resetn)
213   begin
214     if sigclear = '1' or resetn = '0' then
215       selE0 <= '0';
216     elsif sigext0 = '1' then
217       selE0 <= '1';
218     end if;
219   end process;
220
221   -- Mux for upper value (E0 or 0)
222   process(selE0)
223   begin
224     if selE0 = '1' then
225       upperdata <= X"E0";
226     else
227       upperdata <= X"00";
228     end if;
229   end process;
230
231   -- datacode data set
232   datacode <= upperdata & fetchdata;
233
234   -- Keys registers
235   KEY0 : process(clk, key0clearn, resetn)
236   begin
237     if(rising_edge(clk) and key0en = '1') then
238       key0code <= datacode;
239     end if;
240     if key0clearn = '0' or resetn = '0' then
```

```
241       key0code <= X"0000";
242     end if;
243   end process;
244
245   KEY1 : process(clk, key1clearn, resetn)
246   begin
247     if(rising_edge(clk) and key1en = '1') then
248       key1code <= datacode;
249     end if;
250     if key1clearn = '0' or resetn = '0' then
251       key1code <= X"0000";
252     end if;
253   end process;
254
255   KEY2 : process(clk, key2clearn, resetn)
256   begin
257     if(rising_edge(clk) and key2en = '1') then
258       key2code <= datacode;
259     end if;
260     if key2clearn = '0' or resetn = '0' then
261       key2code <= X"0000";
262     end if;
263   end process;
264
265   -- Release command (SR Latch)
266   process(sigrelease, sigclear, resetn)
267   begin
268     if sigclear = '1' or resetn = '0' then
269       relbt <= '0';
270     elsif sigrelease = '1' then
271       relbt <= '1';
272     end if;
273   end process;
274
275   -- Release command (SR Latch)
276   process(sigselect, sigclear, resetn)
277   begin
278     if sigclear = '1' or resetn = '0' then
279       selbt <= '0';
280     elsif sigselect = '1' then
281       selbt <= '1';
282     end if;
283   end process;
284
285   -- Key replacement and clear selector
286   SELECTOR : process(relbt, selbt)
287   begin
288     key0en <= '0'; key1en <= '0'; key2en <= '0';
289     key0clearn <= '1'; key1clearn <= '1'; key2clearn <= '1';
290
291     -- Select an empty register to record fetched data
292     if relbt = '0' and selbt = '1' then
293       if datacode /= key0code and datacode /= key1code and
294        datacode /= key2code then
295         if key0code = X"0000" then
296           key0en <= '1';
297         elsif key1code = X"0000" then
298           key1en <= '1';
299         elsif key2code = X"0000" then
300           key2en <= '1';
301         end if;
302       end if;
303     -- Clear released key
304     elsif relbt = '1' and selbt = '1' then
305       -- Handle fake shifts
306       if datacode = X"0012" then
307         if key0code = X"E012" then
308           key0clearn <= '0';
309         elsif key1code = X"E012" then
310           key1clearn <= '0';
```

```
311         elsif key2code = X"E012" then
312           key2clearn <= '0';
313         end if;
314       elsif datacode = X"0059" then
315         if key0code = X"E059" then
316           key0clearn <= '0';
317         elsif key1code = X"E059" then
318           key1clearn <= '0';
319         elsif key2code = X"E059" then
320           key2clearn <= '0';
321         end if;
322       end if;
323       -- Handle normal release
324       if key0code = datacode then
325         key0clearn <= '0';
326       elsif key1code = datacode then
327         key1clearn <= '0';
328       elsif key2code = datacode then
329         key2clearn <= '0';
330       end if;
331     end if;
332   end process;
333
334   -- Out with key codes
335   key_code <= std_logic_vector(key2code & key1code & key0code);
336
337   -- Turn buttons on
338   process(clk, sigclear, resetn)
339   begin
340     if(rising_edge(clk) and sigclear = '1') then
341       if key0code /= X"0000" then
342         key_on(0) <= '1';
343       else
344         key_on(0) <= '0';
345       end if;
346
347       if key1code /= X"0000" then
348         key_on(1) <= '1';
349       else
350         key_on(1) <= '0';
351       end if;
352
353       if key2code /= X"0000" then
354         key_on(2) <= '1';
355       else
356         key_on(2) <= '0';
357       end if;
358     end if;
359     if resetn = '0' then
360       key_on <= "000";
361     end if;
362   end process;
363
364   -- Comparator for newdata signal
365   --   Signals on rising edge of ps2_datardy
366   process(ps2_datardy, sigfetched, sigsending, resetn)
367   begin
368     if(rising_edge(ps2_datardy)) then
369       newdata <= '1';
370     end if;
371     if resetn = '0' or sigfetched = '1' or sigsending = '1' then
372       newdata <= '0';
373     end if;
374   end process;
375
376   -- Keyboar lights control
377   -- Detect ligths state change
378   process(clk, lights, sigsending, resetn)
379     variable laststate : std_logic_vector(2 downto 0);
380   begin
```

```
381     if resetn = '0' or sigsending = '1' then
382       laststate := "XXX";
383       siguplights <= '0';
384     elsif(rising_edge(clk)) then
385       if laststate /= lights then
386         siguplights <= '1';
387         laststate := lights;
388       else
389         siguplights <= '0';
390       end if;
391     end if;
392   end process;
393
394   -- Send commands to keyboard
395   process(clk, siguplights, en, resetn)
396     type cmdstatename is (
397       SETCMD, SEND, WAITACK, SETLIGHTS, SENDVAL, WAITACK1, CLEAR
398     );
399     variable cmdstate : cmdstatename;
400   begin
401     if(rising_edge(clk)) then
402       sigsend <= '0';
403       sigsending <= '1';
404
405       case cmdstate is
406         when SETCMD =>
407           hdata <= X"ED";
408           if sigsendrdy = '1' then
409             cmdstate := SEND;
410           end if;
411
412         when SEND =>
413           sigsend <= '1';
414           cmdstate := WAITACK;
415
416         when WAITACK =>
417           if ps2_datardy = '1' then
418             if ps2_dataout = X"FE" then
419               cmdstate := SETCMD;
420             else
421               cmdstate := SETLIGHTS;
422             end if;
423           end if;
424
425         when SETLIGHTS =>
426           hdata <= "00000" & lights;
427           if sigsendrdy = '1' then
428               cmdstate := SENDVAL;
429           end if;
430
431         when SENDVAL =>
432           sigsend <= '1';
433           cmdstate := WAITACK1;
434
435         when WAITACK1 =>
436           if ps2_datardy = '1' then
437             if ps2_dataout = X"FE" then
438               cmdstate := SETLIGHTS;
439             else
440               cmdstate := CLEAR;
441             end if;
442           end if;
443
444         when CLEAR =>
445           sigsending <= '0';
446       end case;
447     end if;
448     if resetn = '0' or en = '0' or siguplights = '1' then
449       sigsending <= '1';
450       sigsend <= '0';
```

```
451       cmdstate := SETCMD;
452     end if;
453   end process;
454
455 end rtl;
```

A.4 Exemplos

A.4.1 Uso do mouse

Código VHDL 4: test_mouse_ctrl.vhd.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  entity ps2_mouse_test is
6    port
7    (
8      ---------- Clock Input ----------
9      CLOCK_24  :   in  STD_LOGIC_VECTOR (1 downto 0);  -- 24 MHz
10     CLOCK_27  :   in  STD_LOGIC_VECTOR (1 downto 0);  -- 27 MHz
11     CLOCK_50  :   in  STD_LOGIC;                      -- 50 MHz
12     CLOCK_300 :   out STD_LOGIC;                      -- 300 KHz
13
14     ---------- Push Button ----------
15     KEY   :   in  STD_LOGIC_VECTOR (3 downto 0);    -- Pushbutton[3:0]
16
17     ---------- DPDT Switch ----------
18     SW  :   in  STD_LOGIC_VECTOR (9 downto 0);      -- Toggle Switch[9:0]
19
20     ---------- 7-SEG Dispaly ----------
21     HEX0  :   out STD_LOGIC_VECTOR (6 downto 0);    -- Seven Segment Digit 0
22     HEX1  :   out STD_LOGIC_VECTOR (6 downto 0);    -- Seven Segment Digit 1
23     HEX2  :   out STD_LOGIC_VECTOR (6 downto 0);    -- Seven Segment Digit 2
24     HEX3  :   out STD_LOGIC_VECTOR (6 downto 0);    -- Seven Segment Digit 3
25
26     ---------- LED ----------
27     LEDG  :   out STD_LOGIC_VECTOR (7 downto 0);    -- LED Green[7:0]
28     LEDR  :   out STD_LOGIC_VECTOR (9 downto 0);    -- LED Red[9:0]
29
30     ---------- PS2 ----------
31     PS2_DAT   :   inout STD_LOGIC;  -- PS2 Data
32     PS2_CLK   :   inout STD_LOGIC   -- PS2 Clock
33   );
34 end;
35
36 architecture struct of ps2_mouse_test is
37   component conv_7seg
38     port(
39       digit   :   in STD_LOGIC_VECTOR (3 downto 0);
40       seg     :   out STD_LOGIC_VECTOR (6 downto 0)
41     );
42   end component;
43
44   component mouse_ctrl
45     generic(
46       clkfreq : integer
47     );
48     port(
49       ps2_data  : inout std_logic;
50       ps2_clk   : inout std_logic;
51       clk       : in  std_logic;
52       en        : in  std_logic;
53       resetn    : in  std_logic;
54       newdata   : out std_logic;
55       bt_on     : out std_logic_vector(2 downto 0);
```

```
56        ox, oy     : out std_logic;
57        dx, dy     : out std_logic_vector(8 downto 0);
58        wheel      : out std_logic_vector(3 downto 0)
59      );
60    end component;
61
62    signal CLOCK_100, CLOCKHZ, signewdata, resetn : std_logic;
63    signal dx, dy : std_logic_vector(8 downto 0);
64    signal x, y   : std_logic_vector(7 downto 0);
65    signal hexdata : std_logic_vector(15 downto 0);
66
67    constant SENSIBILITY : integer := 16; -- Rise to decrease sensibility
68  begin
69    -- KEY(0) Reset
70    resetn <= KEY(0);
71
72    mousectrl : mouse_ctrl generic map (24000) port map(
73      PS2_DAT, PS2_CLK, CLOCK_24(0), '1', KEY(0),
74      signewdata, LEDG(7 downto 5), LEDR(9), LEDR(7), dx, dy, LEDG(3 downto 0)
75    );
76
77    hexseg0: conv_7seg port map(
78      hexdata(3 downto 0), HEX0
79    );
80    hexseg1: conv_7seg port map(
81      hexdata(7 downto 4), HEX1
82    );
83    hexseg2: conv_7seg port map(
84      hexdata(11 downto 8), HEX2
85    );
86    hexseg3: conv_7seg port map(
87      hexdata(15 downto 12), HEX3
88    );
89
90    -- Read new mouse data
91    process(signewdata, resetn)
92      variable xacc, yacc : integer range -10000 to 10000;
93    begin
94      if(rising_edge(signewdata)) then
95        x <= std_logic_vector(to_signed(to_integer(signed(x)) + ((xacc + to_integer(signed(dx)
              )) / SENSIBILITY), 8));
96        y <= std_logic_vector(to_signed(to_integer(signed(y)) + ((yacc + to_integer(signed(dy)
              )) / SENSIBILITY), 8));
97        xacc := ((xacc + to_integer(signed(dx))) rem SENSIBILITY);
98        yacc := ((yacc + to_integer(signed(dy))) rem SENSIBILITY);
99      end if;
100     if resetn = '0' then
101       xacc := 0;
102       yacc := 0;
103       x <= (others => '0');
104       y <= (others => '0');
105     end if;
106   end process;
107
108   hexdata(3  downto  0) <= y(3 downto 0);
109   hexdata(7  downto  4) <= y(7 downto 4);
110   hexdata(11 downto  8) <= x(3 downto 0);
111   hexdata(15 downto 12) <= x(7 downto 4);
112
113   -- 100 KHz clock
114   process(CLOCK_24(0))
115     variable count : integer range 0 to 240 := 0;
116   begin
117     if(CLOCK_24(0)'event and CLOCK_24(0) = '1') then
118       if count < 240 / 2 then
119         CLOCK_100 <= '1';
120       else
121         CLOCK_100 <= '0';
122       end if;
123       if count = 240 then
```

```
124         count := 0;
125       end if;
126       count := count + 1;
127     end if;
128   end process;
129
130   -- 300 KHz clock
131   process(CLOCK_24(0))
132     variable count : integer range 0 to 80 := 0;
133   begin
134     if(CLOCK_24(0)'event and CLOCK_24(0) = '1') then
135       if count < 80 / 2 then
136         CLOCK_300 <= '1';
137       else
138         CLOCK_300 <= '0';
139       end if;
140       if count = 80 then
141         count := 0;
142       end if;
143       count := count + 1;
144     end if;
145   end process;
146
147   -- Hz clock
148   process(CLOCK_24(0))
149     constant F_HZ : integer := 1000000;
150
151     constant DIVIDER : integer := 24000000/F_HZ;
152     variable count : integer range 0 to DIVIDER := 0;
153   begin
154     if(rising_edge(CLOCK_24(0))) then
155       if count < DIVIDER / 2 then
156         CLOCKHZ <= '1';
157       else
158         CLOCKHZ <= '0';
159       end if;
160       if count = DIVIDER then
161         count := 0;
162       end if;
163       count := count + 1;
164     end if;
165   end process;
166 end struct;
```

A.4.2 Uso do teclado

Código VHDL 5: test_kbdex_ctrl.vhd.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  entity ps2_kbd_test is
5    port
6    (
7      ---------------------  Clock Input  ---------------------
8      CLOCK_24  :   in  STD_LOGIC_VECTOR (1 downto 0);  --  24 MHz
9      CLOCK_27  :   in  STD_LOGIC_VECTOR (1 downto 0);  --  27 MHz
10     CLOCK_50  :   in  STD_LOGIC;                      --  50 MHz
11     -- CLOCKTAP :   out STD_LOGIC;
12
13     ---------------------  Push Button  ---------------------
14     KEY   :   in  STD_LOGIC_VECTOR (3 downto 0);    --  Pushbutton[3:0]
15
16     ---------------------  DPDT Switch  ---------------------
17     SW  :   in  STD_LOGIC_VECTOR (9 downto 0);      --  Toggle Switch[9:0]
18
19     ---------------------  7-SEG Dispaly  ---------------------
20     HEX0  :   out STD_LOGIC_VECTOR (6 downto 0);    --  Seven Segment Digit 0
```

```
21      HEX1   :   out STD_LOGIC_VECTOR (6 downto 0);     -- Seven Segment Digit 1
22      HEX2   :   out STD_LOGIC_VECTOR (6 downto 0);     -- Seven Segment Digit 2
23      HEX3   :   out STD_LOGIC_VECTOR (6 downto 0);     -- Seven Segment Digit 3
24
25      ------------------------------ LED -------------------------------
26      LEDG   :   out STD_LOGIC_VECTOR (7 downto 0);     -- LED Green[7:0]
27      LEDR   :   out STD_LOGIC_VECTOR (9 downto 0);     -- LED Red[9:0]
28
29      ---------------------------- PS2 ---------------------------------
30      PS2_DAT    :   inout STD_LOGIC;  -- PS2 Data
31      PS2_CLK    :   inout STD_LOGIC   -- PS2 Clock
32    );
33  end;
34
35  architecture struct of ps2_kbd_test is
36    component conv_7seg
37      port(
38        digit    :   in STD_LOGIC_VECTOR (3 downto 0);
39        seg      :   out STD_LOGIC_VECTOR (6 downto 0)
40      );
41    end component;
42
43    component kbdex_ctrl
44      generic(
45        clkfreq : integer
46      );
47      port(
48        ps2_data  : inout std_logic;
49        ps2_clk   : inout std_logic;
50        clk       : in  std_logic;
51        en        : in  std_logic;
52        resetn    : in  std_logic;
53        lights    : in  std_logic_vector(2 downto 0); -- lights(Caps, Num, Scroll)
54        key_on    : out std_logic_vector(2 downto 0);
55        key_code  : out std_logic_vector(47 downto 0)
56      );
57    end component;
58
59    signal CLOCKHZ, resetn  : std_logic;
60    signal key0             : std_logic_vector(15 downto 0);
61    signal lights, key_on   : std_logic_vector( 2 downto 0);
62  begin
63    resetn <= KEY(0);
64
65    hexseg0: conv_7seg port map(
66      key0(3 downto 0), HEX0
67    );
68    hexseg1: conv_7seg port map(
69      key0(7 downto 4), HEX1
70    );
71    hexseg2: conv_7seg port map(
72      key0(11 downto 8), HEX2
73    );
74    hexseg3: conv_7seg port map(
75      key0(15 downto 12), HEX3
76    );
77
78    kbd_ctrl : kbdex_ctrl generic map(24000) port map(
79      PS2_DAT, PS2_CLK, CLOCK_24(0), KEY(1), resetn, lights(1) & lights(2) & lights(0),
80      key_on, key_code(15 downto 0) => key0
81    );
82
83    LEDG(7 downto 5) <= key_on;
84
85    -- lights <= SW(2 downto 0);
86
87    -- CLOCKTAP <= CLOCKHZ;
88
89    -- Playing with lights! xD
90    process(CLOCKHZ, resetn, key_on)
```

```
91      variable dir : boolean := false;
92    begin
93      if(rising_edge(CLOCKHZ)) then
94        if lights(2) = '1' then
95          dir := true;
96        elsif lights(0) = '1' then
97          dir := false;
98        end if;
99        if key_on = "000" then
100         if not dir then
101           lights <= lights(1 downto 0) & lights(2);
102         else
103           lights <= lights(0) & lights(2 downto 1);
104         end if;
105       end if;
106     end if;
107     if resetn = '0' then
108       dir := false;
109       lights <= "001";
110     end if;
111   end process;
112
113   -- Hz clock
114   process(CLOCK_24(0))
115     constant F_HZ : integer := 5;
116
117     constant DIVIDER : integer := 24000000/F_HZ;
118     variable count : integer range 0 to DIVIDER := 0;
119   begin
120     if(rising_edge(CLOCK_24(0))) then
121       if count < DIVIDER / 2 then
122         CLOCKHZ <= '1';
123       else
124         CLOCKHZ <= '0';
125       end if;
126       if count = DIVIDER then
127         count := 0;
128       end if;
129       count := count + 1;
130     end if;
131   end process;
132 end struct;
```