Example 41

Keyboard

In this example we will interface the PS/2 port to a PS/2 keyboard, also known as an AT keyboard. The example will not apply to the newer USB keyboards, or to the older, obsolete XT keyboard. Keyboards contain their own microprocessors that continually scan the keys and then send the resulting key pressings to the host – in our case through the PS/2 port.

For a PS/2 keyboard the key pressed is identified by a scan code. This code is associated with a physical key. Thus, the left shift key and the right shift key have different scan codes. When you press a key the *Make* scan code is sent to the PS/2 port. When you release the key the *Break* scan code is sent to the PS/2 port. The *Make* and *Break* scan codes for all the keys on a PS/2 keyboard are shown in Table 7.1.

For all of the letters and digits the *Make* scan code is a single byte and the *Break* scan code is the same byte preceded by the hex byte F0. Some of the other keys have a 2-byte *Make* scan code in which the first byte is hex E0. The corresponding *Break* scan code has three bytes starting with hex E0. Note that the *PrntScrn* and *Pause* keys are special with a 4-byte and 8-byte *Make* code respectively.

The scan codes have no relationship to the ASCII codes of the key characters. Recall that the ASCII code for an upper-case letter is different from the ASCII code of a lower-case letter. To tell if you are typing an upper-case or lower-case letter you would need to check if you are pressing the shift key (or if you have pressed the CapsLock key) and then press the letter key before you release the shift key. For example, if you want to type an upper-case A you would press the left shift key, press the key A, release the key A, and release the left shift key. From Table 7.1 this would send the following bytes to the PS/2 port.

```
12 1C F0 1C F0 12
```

When you hold a key down the *typematic* feature of the keyboard will, after a *typematic delay* of 0.25 – 1.00 seconds, continue to send out the *Make* scan code at a *typematic rate* of 2 – 30 characters per second. The typematic delay and rate can be changed by sending the 0xF3 command to the keyboard followed by a byte than encodes the new delay and rate.

In this example we will only read data from the keyboard and not send the keyboard any commands. Therefore, we don't need the tri-state buffers in Fig. 7.3. We will, however, need to filter the clock and data signals coming from the keyboard. The filtered data signal, *PS2Df*, will be shifted into two 11-bit words as shown in Fig. 7.5. Note that after shifting in these two words the first byte shifted in will be in *shift*2(8:1) and the second byte shifted in will be in *shift*1(8:1).

Listing 7.2 shows the VHDL code for interfacing to the keyboard. The output *xkey*(15:0) will contain the two bytes shifted in when a key is pressed on the keyboard.

Listing 7.3 is the VHDL program for the top-level design that will display the scan codes of the keys pressed on the 7-segment display.

Table 7.1 Keyboard Scan Codes

Key	Make	Break
A	1C	F0,1C
B	32	F0,32
C	21	F0,21
D	23	F0,23
E	24	F0,24
F	2B	F0,2B
G	34	F0,34
H	33	F0,33
I	43	F0,43
J	3B	F0,3B
K	42	F0,42
L	4B	F0,4B
M	3A	F0,3A
N	31	F0,31
O	44	F0,44
P	4D	F0,4D
Q	15	F0,15
R	2D	F0,2D
S	1B	F0,1B
T	2C	F0,2C
U	3C	F0,3C
V	2A	F0,2A
W	1D	F0,1D
X	22	F0,22
Y	35	F0,35
Z	1A	F0,1A
0	45	F0,45
1	16	F0,16
2	1E	F0,1E
3	26	F0,26
4	25	F0,25
5	2E	F0,2E
6	36	F0,36
7	3D	F0,3D
8	3E	F0,3E
9	46	F0,46

Key	Make	Break
`	0E	F0,0E
-	4E	F0,4E
=	55	F0,55
\	5D	F0,5D
BKSP	66	F0,66
SPACE	29	F0,29
TAB	0D	F0,0D
CAPS	58	F0,58
L Shift	12	F0,12
R Shift	59	F0,59
L Ctrl	14	F0,14
R Ctrl	E0,14	E0,F0,14
L Alt	11	F0,11
R Alt	E0,11	E0,F0,11
L GUI	E0,1F	E0,F0,1F
R GUI	E0,27	E0,F0,27
Apps	E0,2F	E0,F0,2F
Enter	5A	F0,5A
ESC	76	F0,76
Scroll	7E	F0,7E
Insert	E0,70	E0,F0,70
Home	E0,6C	E0,F0,6C
Page Up	E0,7D	E0,F0,7D
Page Dn	E0,7A	E0,F0,7A
Delete	E0,71	E0,F0,71
End	E0,69	E0,F0,69
[	54	F0,54
]	5B	F0,5B
;	4C	F0,4C
'	52	F0,52
,	41	F0,41
.	49	F0,49
/	4A	F0,4A
PrntScrn	E0,7C, E0,12	E0,F0,7C, E0,F0,12

Key	Make	Break
F1	05	F0,05
F2	06	F0,06
F3	04	F0,04
F4	0C	F0,0C
F5	03	F0,03
F6	0B	F0,0B
F7	83	F0,83
F8	0A	F0,0A
F9	01	F0,01
F10	09	F0,09
F11	78	F0,78
F12	07	F0,07
Num	77	F0,77
KP /	E0,4A	E0,F0,4A
KP *	7C	F0,7C
KP -	7B	F0,7B
KP +	79	F0,79
KP EN	E0,5A	E0,F0,5A
KP .	71	F0,71
KP 0	70	F0,70
KP 1	69	F0,69
KP 2	72	F0,72
KP 3	7A	F0,7A
KP 4	6B	F0,6B
KP 5	73	F0,73
KP 6	74	F0,74
KP 7	6C	F0,6C
KP 8	75	F0,75
KP 9	7D	F0,7D
U Arrow	E0,75	E0,F0,75
L Arrow	E0,6B	E0,F0,6B
D Arrow	E0,72	E0,F0,72
R Arrow	E0,74	E0,F0,74
Pause	E1,14, 77,E1, F0,14, F0,77,	None

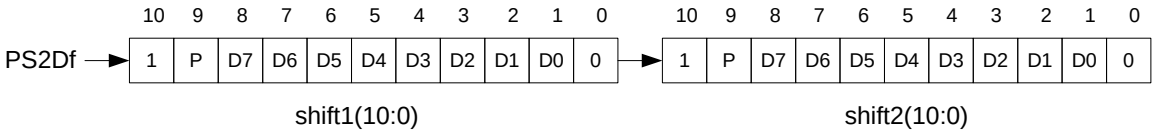

Figure 7.5 Shifting PS2Df into two bytes

Listing 7.2 keyboard.vhd

```
-- Example 41a: keyboard
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity keyboard is
    port(
          clk25 : in STD_LOGIC;
          clr : in STD_LOGIC;
          PS2C : in STD_LOGIC;
          PS2D : in STD_LOGIC;
          xkey : out STD_LOGIC_VECTOR(15 downto 0)
        );
end keyboard;


architecture keyboard of keyboard is
signal PS2Cf, PS2Df: std_logic;
signal ps2c_filter, ps2d_filter: std_logic_vector(7 downto 0);
signal shift1, shift2: std_logic_vector(10 downto 0);
begin

xkey <= shift2(8 downto 1) & shift1(8 downto 1);

-- filter for PS2 clock and data
filter: process(clk25, clr)
begin
   if clr = '1' then
         ps2c_filter <= (others => '0');
         ps2d_filter <= (others => '0');
         PS2Cf <= '1';
         PS2Df <= '1';
   elsif clk25'event and clk25 = '1' then
         ps2c_filter(7) <= PS2C;
         ps2c_filter(6 downto 0) <= ps2c_filter(7 downto 1);
         ps2d_filter(7) <= PS2D;
         ps2d_filter(6 downto 0) <= ps2d_filter(7 downto 1);
         if ps2c_filter = X"FF" then
               PS2Cf <= '1';
         elsif ps2c_filter = X"00" then
               PS2Cf <= '0';
         end if;
         if ps2d_filter = X"FF" then
               PS2Df <= '1';
         elsif ps2d_filter = X"00" then
               PS2Df <= '0';
         end if;
   end if;
end process filter;
```

Listing 7.2 (cont.) keyboard.vhd

```
--Shift Registers used to clock in scan codes from PS2--
shift: process(PS2Cf, clr)
begin
   if (clr = '1') then
         Shift1 <= (others => '0');
         Shift2 <= (others => '0');
   elsif (PS2Cf'event and PS2Cf = '0') then
         Shift1 <= PS2Df & Shift1(10 downto 1);
         Shift2 <= Shift1(0) & Shift2(10 downto 1);
   end if;
end process shift;

end keyboard;
```

Listing 7.3 keyboard_top.vhd

```
-- Example 41b: keyboard_top
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.ps2_components.all;

entity keyboard_top is
    port(
          mclk : in STD_LOGIC;
          PS2C: in STD_LOGIC;
          PS2D: in STD_LOGIC;
          btn : in STD_LOGIC_VECTOR(3 downto 0);
          a_to_g : out STD_LOGIC_VECTOR(6 downto 0);
          dp : out STD_LOGIC;
          an : out STD_LOGIC_VECTOR(3 downto 0)
        );
end keyboard_top;

architecture keyboard_top of keyboard_top is
signal pclk, clk25, clk190, clr: std_logic;
signal xkey: std_logic_vector(15 downto 0);
begin
   clr <= btn(3);
   dp <= '1';        -- decimal points off

U1 : clkdiv2
   port map(mclk => mclk, clr => clr, clk25 => clk25,
         clk190 => clk190);

U2 : keyboard
   port map(clk25 => clk25, clr => clr, PS2C => PS2C,
         PS2D => PS2D, xkey => xkey);

U3 : x7segb
   port map(x => xkey, cclk => clk190, clr => clr,
         a_to_g => a_to_g, an => an);

end keyboard_top
```