

Interfacing a PS/2 Keyboard Using SPI Slave

Authors: S. G. Sriharsha
H. N. Naveen

Associated Project: Yes
Associated Part Family: CY8C27xxx
PSoC Designer Version: 4.1

Abstract

This application note demonstrates the simplicity of interfacing a PS/2 Keyboard to a PSoC device. The SPI Slave user module in the PSoC device is used for the interfacing.

Introduction

A keyboard is a device which contains a large matrix of keys along with an on-board controller which takes care of reading the keys pressed and sending the data to the CPU of a personal computer. The initial computer keyboard was developed by IBM in 1980's which was called as IBM PC/XT keyboard. This had 83 keys, 5-pin DIN connector and it used uni-directional serial protocol. The IBM PC/XT keyboards were replaced by the IBM AT keyboards. These incorporated 84-101 keys, 5-pin DIN connector and they used bi-directional serial protocol. In the late 80's, these keyboards were succeeded by the IBM PS/2 keyboard. The PS/2 keyboards have 84-101 keys. They had two types of physical connectors, 5-pin DIN and 6-pin mini DIN socket which had a smaller form factor. These keyboards use bi-directional protocol with 17 host-to-keyboard commands.

The main intention of developing the keyboards was to connect it to a computer and to gather the user fed data/keystrokes. The latest microcontrollers in the market run at higher speeds have large internal memories and have many internal peripherals. These microcontrollers perform complex operations in variety of applications. These microcontrollers can use the PS/2 keyboard to get the user typed data and utilize this to perform other task.

This application note is intended to demonstrate the ability of the PSoC mixed signal array device

to communicate with an IBM PS/2 keyboard without using any external component.

We will go through the basics of PS/2 Keyboard, the physical interface, protocol and scan code used. Later we will see how the PSoC device can be interfaced with the PS/2 keyboard along with the schematics and the source code.

IBM PC Keyboard

Keyboards consist of a large matrix of keys, all of which are monitored by an on-board processor that monitor which key(s) are being pressed/released and send the appropriate data to the host. This processor takes care of all the debouncing. The motherboard inside the PC contains a "keyboard controller" that is in charge of decoding all of the data received from the keyboard and informing the software of what's going on. All communication between the host and the keyboard uses an IBM ps/2 keyboard protocol.

Physical Interface

The physical PS/2 port is available in one of two types of connectors: The 5-pin DIN or the 6-pin mini-DIN. Both connectors are completely (electrically) similar; the only difference between the two is the arrangement of pins.

Figure 1 below shows the pin-outs for the two types of connector.

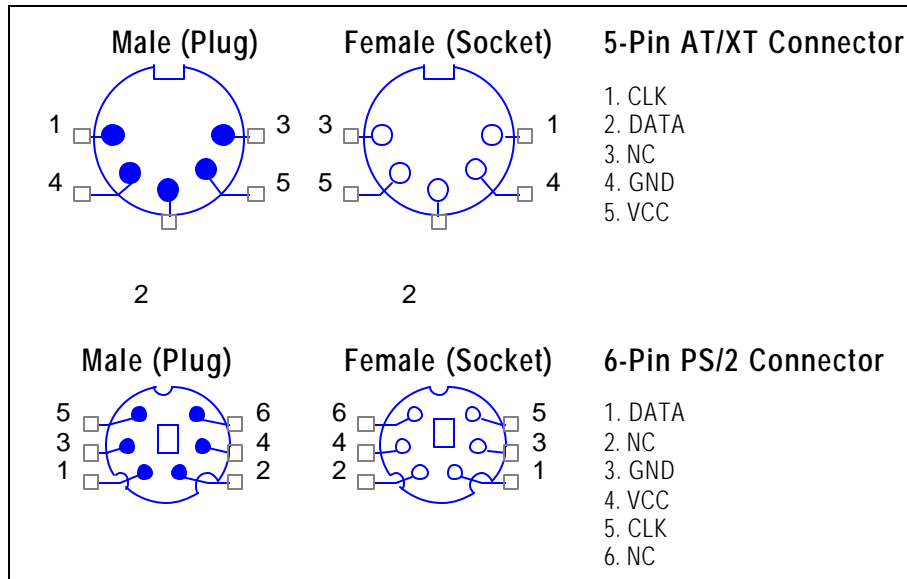


Figure 1. Pin-outs of two types of keyboard connectors

Electrical Interface

The power supply to the keyboard is provided through the VCC/GND pins. The supply is provided from the motherboard.

The voltage and current specification of the keyboard are given below:

$$\begin{aligned} \text{VCC} &= +4.5\text{V to } +5.5\text{V} \\ \text{Current} &= 275\text{mA} \end{aligned}$$

The DATA and CLK lines are open-collector type and have only a "Low" or "High-Z" states. Thus the two signal pins require external pull-up resistors to be connected to VCC for determining the "High" state.

Protocol

The PS/2 keyboards implement a bidirectional synchronous serial protocol. The bus is "idle" when both lines are high (open-collector). This is the only state where the keyboard/mouse is allowed begin transmitting data. The host has ultimate control over the bus and may inhibit communication at any time by pulling the Clock line low.

There are two modes of communication, first is the communication between Host-to-Device and the other is Device-to-host.

The device always generates the clock signal. All data is transmitted one byte at a time and each

byte is sent in a frame consisting of 11-12 bits. These bits are:

- 1 start bit. This is always 0.
- 8 data bits, least significant bit first.
- 1 parity bit (odd parity).
- 1 stop bit. This is always 1.
- 1 acknowledge bit (host-to-device communication only)

Data sent from the device to the host is read on the *falling* edge of the clock signal; data sent from the host to the device is read on the *rising* edge. The clock frequency must be in the range 10 - 16.7 kHz. The keyboard always generates the clock signal, but the host always has ultimate control over communication.

Device-to-Host Communication

When a keyboard wants to send information, it first checks the Clock line to make sure it's at a high logic level. If it's not, the host is inhibiting communication and the device must buffer any to-be-sent data until the host releases Clock. The Clock line must be continuously high for at least 50 microseconds before the device can begin to transmit its data.

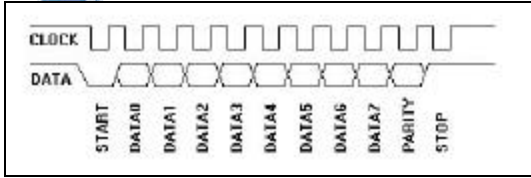


Figure 2. Device-to-Host communication

Host-to-Device Communication

The device always generates the clock signal. If the host wants to send data, it must first put the Clock and Data lines in a "Request-to-send" state as follows:

- Inhibit communication by pulling Clock low for at least 100 microseconds.
- Apply "Request-to-send" by pulling Data low, then release Clock.

Bus States

Data = high, Clock = high: *Idle state.*

Data = high, Clock = low: *Communication Inhibited.*

Data = low, Clock = high: *Host Request-to-Send*

The device should check for Request-to-send state at intervals not to exceed 10 milliseconds. When the device detects this state, it will begin generating Clock signals and clock in eight data bits and one stop bit. The host changes the Data line only when the Clock line is low, and data is read by the device when Clock is high. This is opposite of what occurs in device-to-host communication.

After the stop bit is received, the device will acknowledge the received byte by bringing the Data line low and generating one last clock pulse. If the host does not release the Data line after the 11th clock pulse, the device will continue to generate clock pulses until the Data line is released (the device will then generate an error.)

The host may abort transmission at time before the 11th clock pulse (acknowledge bit) by holding Clock low for at least 100 microseconds.

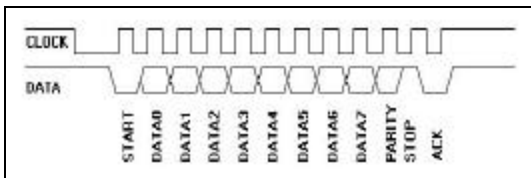


Figure 3. Host-to-Device communication

Reference 1 gives more information on the protocol.

Scan Codes

The PS/2 keyboard has a scan code associated with each key. When a key is pressed, this code is transmitted. If a key is held down for a while, it starts repeating. The repeat rate is typically 10 per second. When a key is released, a "break" code (\$F0) is transmitted followed by the key scan code. For most of the keys, the scan code is one byte.

Some keys like the Home, Insert and Delete keys have an extended scan code, from two to five bytes. The first byte is always \$E0. This is also true for the "break" sequence, e.g., E0 F0 xx...

PS/2 keyboards are capable of handling three sets of scan codes, where set two is default. The keyboard example will only use set two. The complete scan code (Set 2) is listed in Table 1 at the end of this application note.

PSoC and PS/2 Keyboard

In this example the PSoC device is interfaced with the PS/2 keyboard. An LCD is connected to the PSoC in-order to display the key presses. In this example, only the Device-to-Host mode of communication is used.

The PSoC device is configured for a SPI slave and a PWM8. The SPI slave is used for communicating with the keyboard and PWM8 is used to control the contrast of the LCD.

The schematic of the keyboard example is provided as Figure.6 at the end of the application note.

PSoC User Modules

The following user modules are used in the keyboard example:

- SPI Slave
- PWM8
- LCD Tool box

Figure 4 shows the placement of SPI slave and PWM8 user modules.

SPI Slave

The SPIS User Module is a Serial Peripheral Interconnect Slave. It performs full duplex synchronous 8-bit data transfers. SCLK phase, SCLK polarity, and LSB First can be specified to accommodate most SPI protocols.

The SPIS is used to communicate with the keyboard.

PWM8

PWM8 is an 8bit pulse width modulator with programmable period and pulse width. The clock and enable can be selected from several sources.

numbers to a common two- or four-line LCD module.

Port 1 of PSoC is used to drive the LCD in 4-bit interface mode to limit the number of I/O pins required.

LCD Tool box

The LCD Tool Box User Module is a set of library routines that writes text strings and formatted

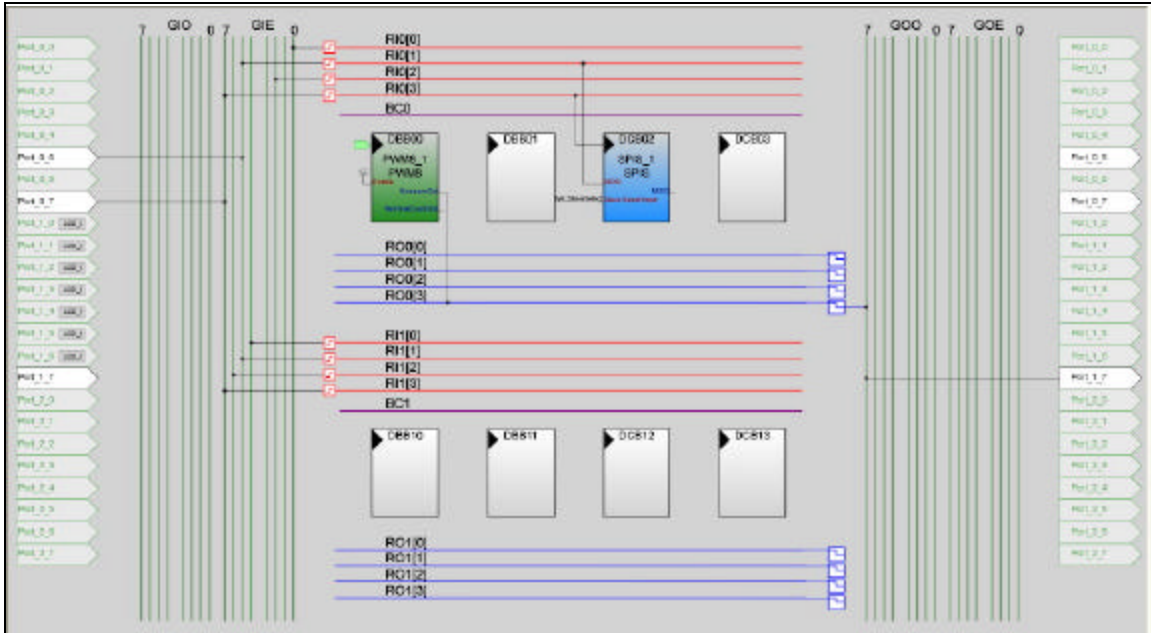


Figure 4. User Module Placement

Algorithm

Algorithm for the keyboard example is given below:

1. Initialize the SPI slave, PWM8, LCD.
2. Display the welcome message.
3. Wait for any key stroke.
4. If Key is pressed, then decode and store in buffer using look-up table (buffer size = 16)
5. Display content of buffer,
6. If buffer is full, need to clear by using Esc key, Backspace key clears only one character else goto step 7.
7. If F1 key is pressed, decrease LCD contrast by modifying PWM8 else goto step 8.
8. If F2 key is pressed, increase LCD contrast by modifying PWM8 else goto step 9.
9. Wait for further keyboard keystrokes .

Figure 5 shows the flow chart of the keyboard example project.

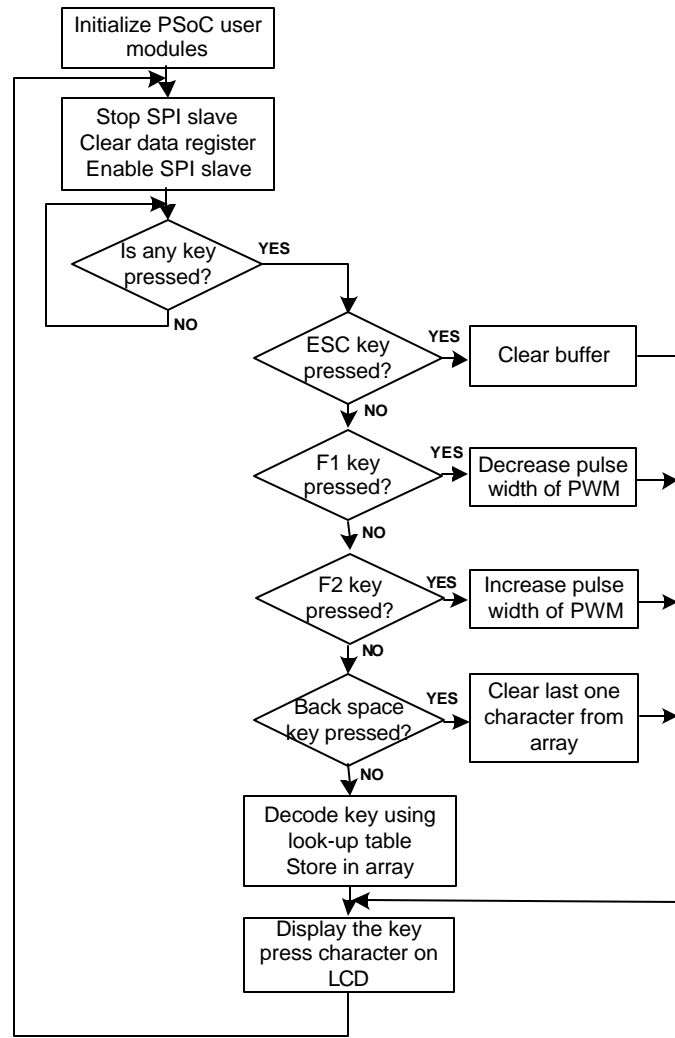
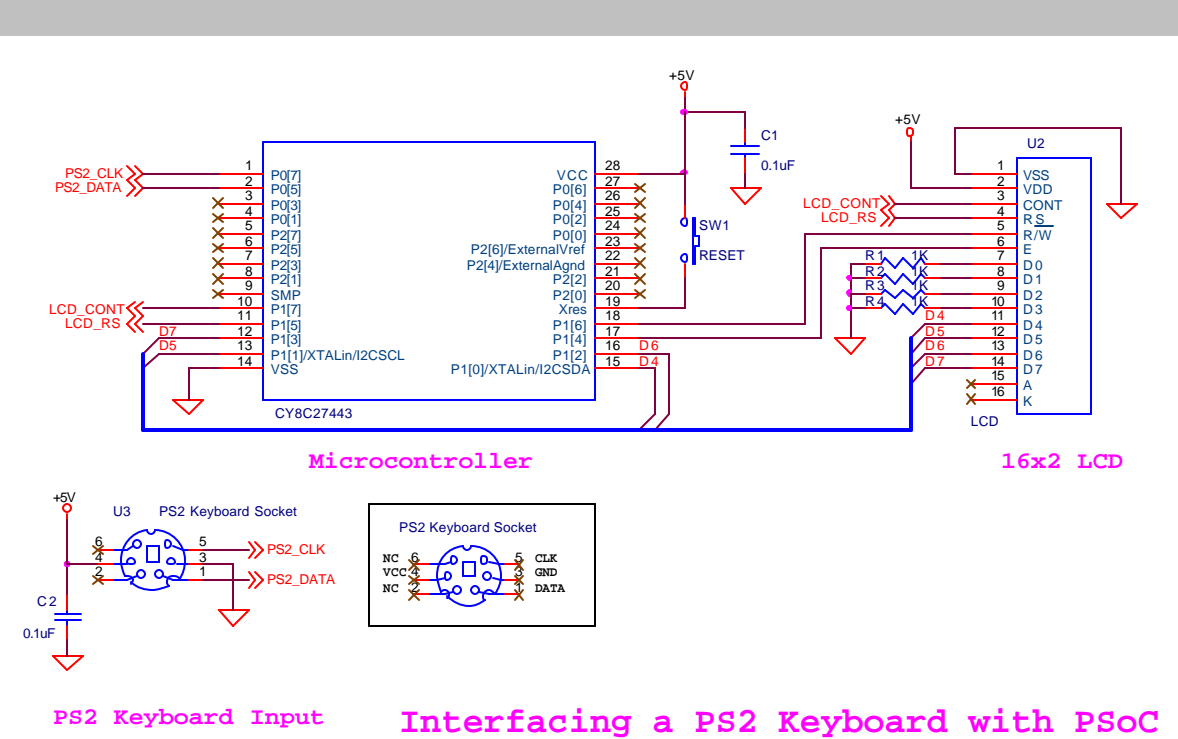


Figure 5. Flow chart

Schematics

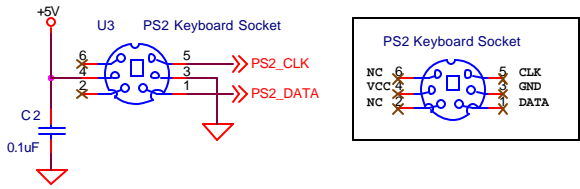
Figure 6 shows the schematic of the keyboard example. The PSoC port 1 is used to drive the LCD in 4-bit interface mode. The PS/2 keyboard is connected to the PSoC through pins P0[7] and P0[5]. In turn these pins are routed to the SPI Slave user module connections SCLK and SDA respectively of the PSoC. The output of PWM8 user module is connected to the LCD Pin 3 which

is contrast control. The contrast of the LCD is controlled through the keyboard keys F1 and F2. The F1 key press results in decreasing the contrast of the LCD and F1 key press increases the contrast.



Microcontroller

16x2 LCD



PS2 Keyboard Input

Interfacing a PS2 Keyboard with PSoC

Figure 6. Schematics

Source Code to interface PS/2 keyboard with PSoC Device

File: main.c

```

//-----
// Project Name      : Keyboard Interface using SPI Slave
// Project ID       : KB001
// Target Processor  : CY8C27443
// Author           : SG Sriharsha & HN Naveen
//-----

#include <m8c.h>           // part specific constants and macros
#include "lcd_1.h"
#include "pwm8_1.h"
#include "SPIS_1.h"
#include "scancodes.h"    // keyboard lookup table

/* main */
void main()
{
    /* Variable Declaration */

    unsigned char bData;
    unsigned char key_up=0, shift_key=0;
    unsigned char i;
    unsigned char buffer_cnt=0;
    unsigned char caps_lock=0;
    char kb_data[] = "                " /*16 white space to initialize array */
    char *ptr_kb_data; /* Limited to 16 because of 16x2 LCD module is used */
    ptr_kb_data = kb_data;

    /* Initialization of Blocks */
    PWM8_1_DisableInt();
    LCD_1_Init();

```

```

LCD_1_Start();
PWM8_1_Start();

LCD_1_Position(0,1);
LCD_1_PrCString("PSoc -Keyboard");

/* Keyboard Board Interface using SPI slave */
/* parity bit is ignored --- SPI block is 8 bit */
while(1)
{
    SPIS_1_Stop(); /*every time necessary to clear shift register and data
register */
    SPIS_1_ClearSS(); /* make sure in Slave mode */
    SPIS_1_Start(SPIS_1_SPIS_MODE_2 | SPIS_1_SPIS_LSB_FIRST ); /* Mode 2 -
data capture on trailing edge */

    while(!(SPIS_1_bReadStatus() & SPIS_1_SPIS_SPI_COMPLETE));
    bData = SPIS_1_bReadRxData(); /* Data only - Parity bit is ignored */

    if(!key_up) /* to avoid break code */
    {
        switch(bData)
        {
            case 0xAA: key_up = 0; break; /* Keyboard Basic Assurance Test -
OK */
            case 0xF0: key_up = 1; break; /* Break code */
            /* case 0x04: case 0x0C: case 0x03:
            case 0x0B: case 0x83: case 0x0A:
            case 0x01: case 0x09: case 0x78:
            case 0x07: break; future expansion */
            case 0x12: shift_key = 1; break; /* Left shift is pressed */
            case 0x59: shift_key = 1; break; /* Right shift is pressed */
            case 0x58: /* Uppercase alphabet */
            if(caps_lock)
                caps_lock = 0;
            else
                caps_lock = 1;
            break;

            case 0x76: /* ESC key to clear the display */
            ptr_kb_data = kb_data;
            for(i=0; i<16; i++)
            {
                *ptr_kb_data = ' ';
                ptr_kb_data++;
            }
            ptr_kb_data = kb_data;
            buffer_cnt = 0;
            break;

            case 0x05: /* F1 - Decrease LCD Contrast */
            PWM8_1_WritePulseWidth((PWM8_1_bReadPulseWidth()+2)&0x1F);
            break;
            case 0x06: /* F2 - Increase LCD Contrast */
            PWM8_1_WritePulseWidth((PWM8_1_bReadPulseWidth()-2)&0x1F);
            break;

            case 0x66: /* Backspace - delete last character from buffer */
            if (buffer_cnt == 0)
                ptr_kb_data = kb_data;
            else
            {
                ptr_kb_data--;

```

```

        buffer_cnt--;
        *ptr_kb_data = ' ';
    }
    break;

default:
if(buffer_cnt != 16)
{
    for(i=0; codes[i][0] != bData && codes[i][0] ; i++);
        if(codes[i][0] == bData)
        {
            if(shift_key)
                *ptr_kb_data = codes[i][1];
            else
                if(caps_lock & i < 26)
                    *ptr_kb_data = codes[i][1];
                else
                    *ptr_kb_data = codes[i][2];
                ptr_kb_data++;
                buffer_cnt++;
            } /* Data found */
        } /* if buffer_cnt = 16 than buffer_full */
        break;
    } /* switch */
} /* if key */
else
{
    key_up = 0;
    switch(bData)
    {
        case 0x12: shift_key = 0; break;
        case 0x59: shift_key = 0; break;
        default: break;
    }
}
LCD_1_Position(1,0);
LCD_1_PrString(kb_data);
} /* while */
} /* END of Main */

```

File: scancodes.h

```

unsigned char codes [ ][3] = {
0x1C, 'A', 'a',    /* Upper case, lower case */
0x32, 'B', 'b',
0x21, 'C', 'c',
0x23, 'D', 'd',
0x24, 'E', 'e',
0x2B, 'F', 'f',
0x34, 'G', 'g',
0x33, 'H', 'h',
0x43, 'I', 'i',
0x3B, 'J', 'j',
0x42, 'K', 'k',
0x4B, 'L', 'l',
0x3A, 'M', 'm',
0x31, 'N', 'n',
0x44, 'O', 'o',
0x4D, 'P', 'p',
0x15, 'Q', 'q',
0x2D, 'R', 'r',
0x1B, 'S', 's',
0x2C, 'T', 't',
0x3C, 'U', 'u',
0x2A, 'V', 'v',
0x1D, 'W', 'w',
0x22, 'X', 'x',
0x35, 'Y', 'y',
0x1A, 'Z', 'z',
0x45, ')', '0',    /* Numerical keys */
0x16, '!', '1',
0x1E, '@', '2',
0x26, '#', '3',
0x25, '$', '4',
0x2E, '%', '5',
0x36, '^', '6',
0x3D, '&', '7',
0x3E, '*', '8',
0x46, '(', '9',
0x29, ' ', ' ',    /* Space BAR */
0x41, '<', ',',    /* Less than, Comma */
0x49, '>', '.',    /* Greater than, Full stop (period) */
0x4E, '_', '-',    /* Underscore, Hyphen */
0x55, '+', '=',    /* Plus, Double Dash (Equal to) '=' */
0x4C, ':', ';',    /* Colon, Semi Colon */
0x4A, '?', '/',    /* Question, Fraction (slash) */
0x54, '{', '[',
0x5B, '}', ']',
0, 0, 0
};

```

References

[1] Web reference: Adam Chapwes ke, <http://www.computer-engineering.org/ps2protocol/>

[2] Keyboard Technical specifications .

Table 1. Keyboard Scan Codes: Set 2

KEY	MAKE	BREAK	---	KEY	MAKE	BREAK	---	KEY	MAKE	BREAK
			--				--			
A	1C	F0,1C		9	46	F0,46		[54	F0,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	F0,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	F0,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	F0,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	05	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	06	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	04	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	03	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	01	F0,01]	5B	F0,5B
3	26	F0,26		F10	09	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77					

About the Authors

Name: S. G. Sriharsha
Title: DVD Front-End Application Engineer

Background: Sriharsha is a Bachelor graduate in Telecommunication. Working with ST Microelectronics Asia Pacific Pte Ltd, Singapore. His interests are participating in hardware design contests.

Contact: sriharshag@gmail.com

Name: H. N. Naveen
Title: Senior Design Engineer

Background: Naveen is a Masters graduate in Industrial Electronics. He is working with Sasken Communication Technologies, Bangalore. He has experience in hardware modeling and designing microcontroller boards. His interests are product development and reference designs.

Contact: navmys@gmail.com or navmys@lycos.com

Web Site: <http://www.naveenmysore.com>

Cypress MicroSystems, Inc.
2700 162nd Street SW, Building D
Lynnwood, WA 98037
Phone: 800.669.0557
Fax: 425.787.4641

<http://www.cypress.com/> / <http://www.cypress.com/support/mysupport.cfm>

Copyright © 2004 Cypress MicroSystems, Inc. All rights reserved.

PSoC™, Programmable System-on-Chip™, and PSoC Designer™ are trademarks of Cypress MicroSystems, Inc.

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

The information contained herein is subject to change without notice. Made in the U.S.A.