Designing a USB Keyboard and PS/2® Mouse Combination Device Using the Cypress Semiconductor CY7C63413 USB Microcontroller

Introduction

The Universal Serial Bus (USB) is an industry standard serial interface designed to connect computers to their peripherals. Devices such as keyboards, mice, joysticks, scanners, printers, and others can all benefit from the connectivity provided by USB. This application note describes how to design a USB keyboard and PS/2 mouse combination device using the Cypress Semiconductor single-chip CY7C63413 USB microcontroller. The PS/2 mouse is a widely used, low-cost pointing device found in computers today. By implementing a legacy PS/2 interface on the CY7C63413 USB microcontroller, originally targeted for USB keyboards, we provide a simple, low-cost solution for keyboard and mouse combination devices. This document starts with the basic operations of a USB keyboard and PS/2 mouse transfer protocol, followed by an introduction to the CY7C63413 USB controller. A schematic of the USB keyboard and PS/2 mouse combination device with its connection details can be found in the Hardware Implementation Section.

The software section of this application note describes the architecture of the firmware required to implement the keyboard and PS/2 mouse interface functions. Several sample code segments are included to assist in the explanation. The source and binary code of the demonstration keyboard firmware is available free of charge from Cypress Semiconductor. Please contact your local Cypress sales office for details.

Since this design is an extension of the USB keyboard design, it is assumed that the reader is familiar with the USB keyboard implementation. Therefore, this application note will only focus on other relevant issues such as PS/2 interface implementation and keyboard/mouse integration. If the reader is not familiar with the USB keyboard design, the application note titled “Designing a USB Keyboard with the Cypress Semiconductor CY7C63413 USB Microcontroller” is highly recommended. The above application note is available at the Cypress web site at www.cypress.com.

This application note also assumes that the reader is familiar with the CY7C63413 USB controller and the Universal Serial Bus. The CY7C63413 data sheet is available from the Cypress web site. USB documentation can be found at the USB Implementers Forum web site at www.usb.org/.

USB Keyboard Basics

Key Switches and Scan Matrix

A PS/2 keyboard has between 101 and 104 keys that are uniquely positioned in a scan matrix. The scan matrix consists of M rows and N columns, all of which are electrically isolated from each other. Typically, the number of rows (M) is no greater than 8, and the number of columns (N) is no greater than 20. Each key sits over two isolated contacts of its corresponding row and column in the scan matrix. When a key is pressed, the two contacts are shorted together, and the row and column of the key are electrically connected.

USB Keyboard Controller

A single CY7C63413 USB microcontroller is used to perform a variety of tasks, all of which help to cut down on the overall system overhead. Besides USB interface functions, the essential task of the microcontroller is to monitor the keys and report to the host computer whenever a key is pressed or released. The microcontroller writes a scan pattern out to the column lines consisting of all 1s and one 0, which is shifted through each column. The result is then read at the row lines. If a 0 is propagated to a row line, then the key at the intersection of that column and row has been pressed. See *Figure 1*.

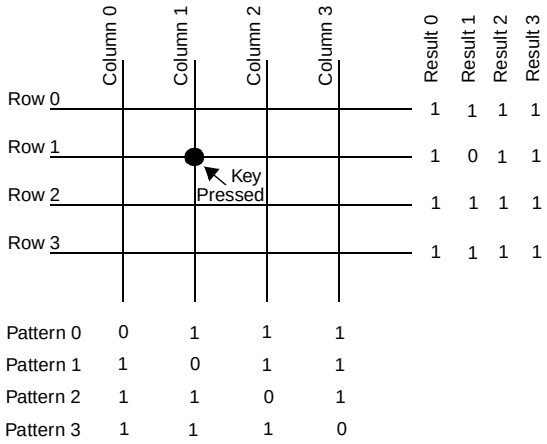

Figure 1. Scan Matrix with Key $_{1,1}$ Pressed

After scanning the key matrix and determining which key is pressed or released, the CY7C63413 microcontroller forms the keyboard data report and sends it to the host through a USB endpoint. For detailed implementation and operation of the USB keyboard, please refer to the application note “Designing a USB Keyboard with the Cypress Semiconductor CY7C63413 USB Microcontroller.”

PS/2 Mouse Basics

The IBM PS/2 mouse is a pointing device that uses a rubber track ball and two encoders to indicate X and Y movement to the host system. It also typically has two push-button switches. The mouse is connected to the host system via a 2.7 meter, shielded, 4-conductor cable, and a 6-pin connector. The four signals are: Data, Clock, +5Vdc and Ground. The signals and their pin assignments on the PS/2 connector are shown in the schematic in *Figure 6*.

Operation Modes

The PS/2 mouse can operate in four different modes, shown below:

- **Reset Mode**: in this mode a self-test is initiated during power-on or by a Reset command. Upon successful completion of the self-test, a completion code (AAh) and an ID code (00h) are transmitted to the host system. Any commands sent before the transmission of the completion code and the ID byte are ignored. If the self-test fails, the mouse responds with a error code (FCh) and an ID code (00h), after which the mouse is disabled and awaits further commands from the host system.
- **Stream Mode**: in this mode a data report is only transmitted to the host system if a button is pressed or released or if at least one count of movement has been detected. The maximum rate of transfer is the programmed sample rate.
- **Remote Mode**: also known as "polling" mode. In this mode, a data report is transmitted only in response to a Read Data command.
- **Wrap Mode**: in this mode, any byte of data sent by the host system, except hexadecimal values ECh and FFh, is returned by the mouse.

PS/2 Mouse Protocol and Commands

The PS/2 mouse communicates with the host system through a defined protocol consisting of commands and responses. The ACK byte (FAh) is always the first response to any valid input received from the host system, except for Resend command and in Wrap mode. *Table 1* shows a list of all the valid commands.

Table 1. PS/2 Mouse Commands

Hex Code	Command
FF	Reset
FE	Resend
F6	Set Default
F5	Disable
F4	Enable
F3	Set Sampling Rate
F2	Read Device Type
F0	Set Remote Mode
EE	Set Wrap Mode
EC	Reset Wrap Mode
EB	Read Data
EA	Set Stream Mode
E9	Status Request
E8	Set Resolution
E7	Set Scaling 2:1
E6	Reset Scaling

Only the mouse commands that are used in this design are discussed below. For complete description of the PS/2 mouse commands and operation, please see "IBM Personal System/2 Mouse Technical Reference." from IBM (Document Number S68X-2229-00).

The following are the commands used by this design and their brief descriptions:

- **Reset**: this command causes the mouse to enter the reset mode and do an internal self-test.
- **Resend**: any time the mouse receives an invalid command, it returns a Resend command to the host system. The host system, in turn, sends this command when it detects any error in any transmission from the mouse. When the mouse receives a Resend command, it retransmits the last packet of data sent.
- **Set Stream Mode**: this command sets the Stream mode.
- **Enable**: this command is used in the Stream mode to begin transmission.
- **Disable**: this command is used in the Stream mode to stop transmissions initiated by the mouse. The mouse responds to all other commands while disabled. If the mouse is in the Stream mode, it must be disabled before sending it any command that requires a response.
- **Read Data**: this command requests that all data defined in the data packet format be transmitted. This command is executed in either remote or stream mode. The data is transmitted even if there has been no movement since the last report or the switch status is unchanged.

Mouse Data Report

In the Stream mode, a data report is sent at the end of a sample interval if there has been a button event or mouse movement. The sampling rate is, by default, 100 reports per second, but it can be changed by the Set Sampling Rate command. On the other hand, in the Remote (or polling) mode, a data report is sent in response to a Read Data command.

The data report is three bytes long and its format is shown in *Table 2*.

Table 2. PS/2 Mouse Data Report Format

Byte	Bit	Description
1	7	Y Data Overflow 1= Overflow
	6	X Data Overflow 1= Overflow
	5	Y Data Sign 1= Negative
	4	X Data Sign 1= Negative
	3	Reserved
	2	Reserved
	1	Right Button Status 1= Pressed
	0	Left Button Status 1 = Pressed
2	7–0	X Data
3	7–0	Y Data

Data Transmission

Data transmission between the PS/2 mouse and the host system is carried out by 2 signals: **Clock** and **Data**. The Clock signal is used to clock serial data. The mouse generates the clocking signal when sending data to or receiving data from the host system. The host system requests the mouse to receive host data output by forcing the Data line LOW and allowing Clock to go HIGH. *Figure 2* illustrates a PS/2 data frame containing a single byte of data.

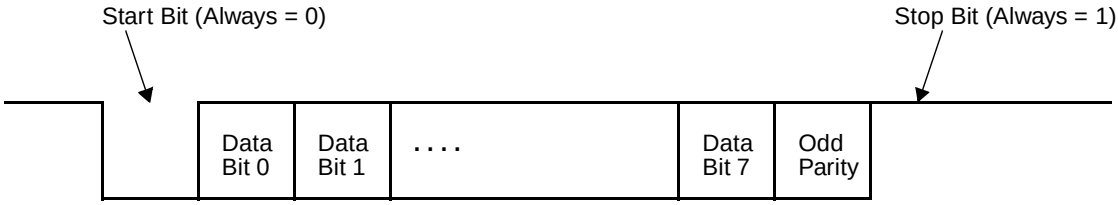

Figure 2. PS/2 Data Frame Format

Communication is bidirectional using the Clock and Data signal lines. The signal for each of these lines comes from open collector devices, allowing either the mouse or the host system to drive a line LOW. During a non-transmission state, both Clock and Data lines are held HIGH. For detailed information on signaling protocol for data transmission using Clock and Data lines, please refer to "IBM Personal System/2 Mouse Technical Reference."

Introduction to CY7C63413

The CY7C63413 is a high performance 8-bit RISC microcontroller with an integrated USB Serial Interface Engine (SIE). The architecture implements 37 instructions that are optimized for USB applications. The CY7C63413 has a built-in clock oscillator and timers, as well as general purpose I/O lines that can be configured as inputs with internal pull-ups, open-drain outputs, or traditional CMOS outputs. High performance, low-cost human-interface type computer peripherals such as a keypad or keyboard can be implemented with a minimum of external components and firmware effort.

Clock Circuit

The CY7C63413 has a built-in clock oscillator and PLL-based frequency doubler. This circuit allows a cost-effective 6-MHz ceramic resonator to be used externally while the on-chip RISC core runs at 12 MHz.

USB Serial Interface Engine (SIE)

The operation of the SIE is totally transparent to the user. In the receive mode, USB packet decode and data transfer to the endpoint FIFO are automatically done by the SIE. The SIE then generates an interrupt to invoke a service routine after a packet is unpacked.

In the transmit mode, data transfer from the endpoint and the assembly of the USB packet are handled automatically by the SIE.

General Purpose I/O

The CY7C63413 has 32 general purpose I/O lines divided into 4 ports: Port 0 through Port 3. One such I/O circuit is shown in *Figure 3*. Each port (8 bits) can be configured as inputs with internal pull-ups (7 Kohms), open drain outputs (high-impedance inputs), or traditional CMOS outputs. The port configuration is determined according to *Table 3* below**.**

Table 3. GPIO Configuration

Port Configuration Bits	Pin Interrupt Bit	Driver mode	Interrupt Polarity
11	X	Resistive	–
10	0	CMOS Output	disabled
10	1	CMOS Input	disabled
01	X	Open Drain	–
00	X	Open Drain	+

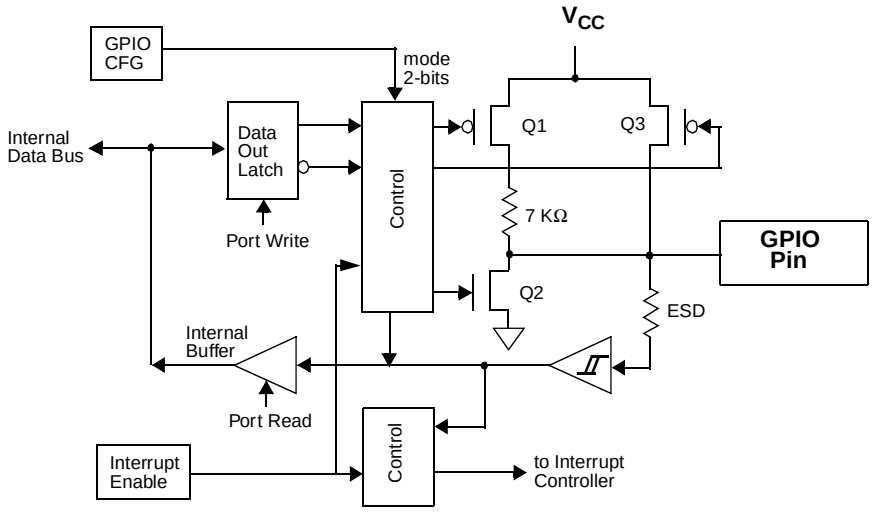

Figure 3. One General Purpose I/O Line

Ports 0 to 2 offer low current drive with a typical current sink capability of 7 mA. Port 3 offers a higher current drive, with a typical current sink of 12 mA which can be used to drive LEDs.

Each General Purpose I/O (GPIO) is capable of generating an interrupt to the RISC core. Interrupt polarity is selectable on a per port basis using the GPIO Configuration Register (see *Table 3* above.) Selecting a negative polarity ("–") will cause falling edges to trigger an interrupt, while a positive polarity ("+") selects rising edges as triggers. The interrupt triggered by a GPIO line is individually enabled by a dedicated bit in the Interrupt Enable Register. All GPIO interrupts are further masked by the Global GPIO Interrupt Enable Bit in the Global Interrupt Enable Register.

The GPIO Configuration Register is located at I/O address 0x08. The Data Registers are located at I/O addresses 0x00 to 0x03 for Port 0 to Port 3 respectively.

Power-up Mode

The CY7C63413 offers 2 modes of operation after a power-on-reset (POR) event: suspend-on-reset (typical for a USB application) and run-on-reset (typical for a non-USB application). The suspend-on-reset mode is selected by attaching a pull-up resistor (100 to 470 KΩ) to V_{CC} on Bit 7 of GPIO Port 3. The run-on-reset mode is selected by attaching a pull-down resistor (0 to 470 KΩ) to ground on Bit 7 of GPIO Port 3. See *Figure 4*. As the USB keyboard and PS/2 mouse combination device is a USB implementation, the CY7C63413 should be configured for suspend-on-reset.

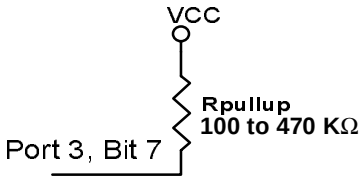

Figure 4. (a) Suspend-On-Reset Mode

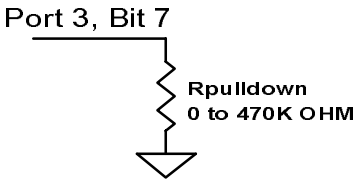

Figure 4. (b) Run-On-Reset Mode

Hardware Implementation

Figure 6 is the schematic for a USB keyboard and PS/2 mouse combination device. This schematic is very similar to the one in the USB keyboard-only design. The major difference is the addition of the PS/2 mouse Clock and Data signal lines to the GPIO Port 3 bit 6 and bit 7, respectively.

Port 2 is configured as resistive inputs (7 Kohm pull-ups to V_{CC}), and are connected to the M rows of the scan matrix (up to 8 rows are supported). Ports 0, 1, and 3 are configured as open drain outputs, and are used to connect to the N columns of the scan matrix, the 3 LEDs (Num Lock, Caps Lock, and Scroll Lock), and the two PS/2 mouse signals. Since the PS/2 signals occupy two bits of Port 3, we can only support keyboard scan matrices up to 18 columns. Most of the existing AT-101 or AT-104 type keyboards do not use more than 18 columns. For keyboards that use scan matrices with more than 18 columns, it is usually possible to relocate the keys from the columns 19 and greater to unassigned locations on columns 18 or below. The three LEDs are connected to the lower three bits of Port 3 as well (for high current drive). For scan matrices with less then 8 rows or 18 columns, the unused port bits should be left unconnected.

The PS/2 signals are connected to the PS/2 mouse through a 6-pin PS/2 receptacle. +5V and GND are also available on the receptacle to power the PS/2 mouse. The strap option pull-up resistor R8 (470 KΩ) selects the Suspend-On-Reset mode for the USB microcontroller, and the series termination resistor R7 (100Ω) is used to reduce crosstalk problems over the PS/2 lines.

During a keyboard scan test where no key is pressed, the row port data bits will be HIGH since all row lines are internally pulled up to V_{CC}. When a key is pressed, driving its column port line LOW (key scan pattern) will cause its row line to go LOW as well. See *Figure 5*.

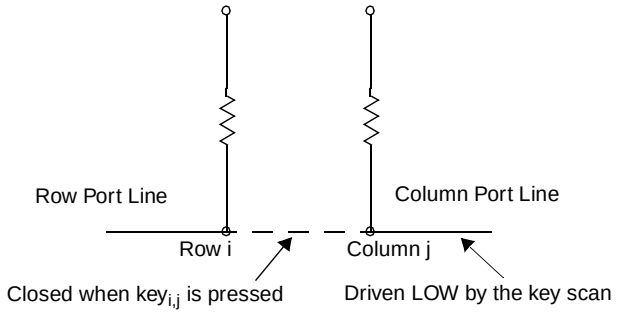

Figure 5. Row/Column Port Configuration

A 6 MHz ceramic resonator is connected to the clock inputs of the microcontroller. This component should be placed as close to the microcontroller as possible.

According to the USB specification, the USB D– line of a low-speed device (1.5 Mbps) should be tied to a voltage source between 3.0V and 3.6V with a 1.5 KΩ pull-up terminator. The CY7C63413 eliminates the need for a 3.3V regulator by specifying a 7.5-KΩ resistor connected between the USB D– line and the nominal 5V V_{CC}. This is compliant with "Device Working Group Review Request 135."

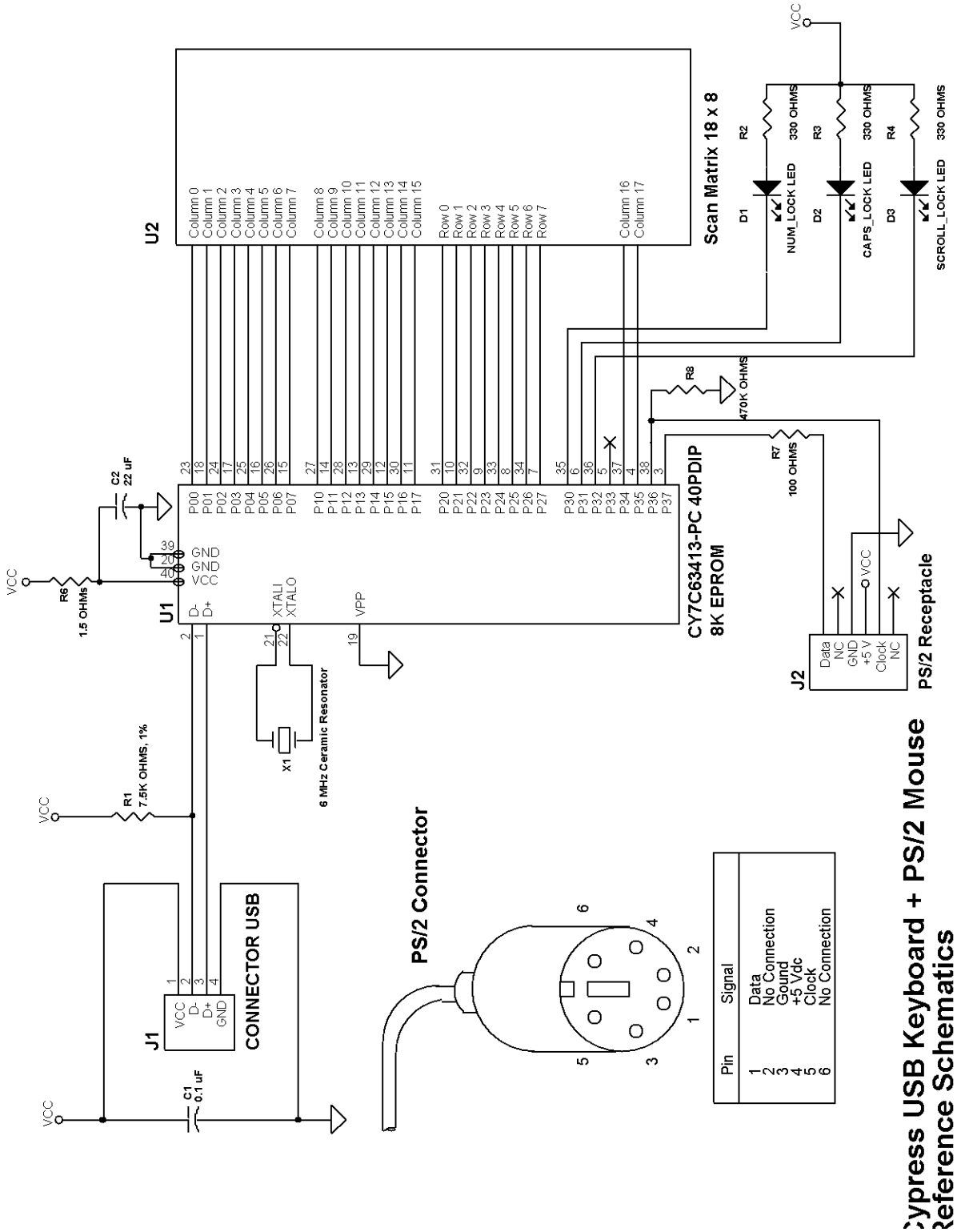

Figure 6. Hardware Implementation - Schematic

Firmware Implementation

USB Interface

Like the USB keyboard-only device, the USB keyboard and PS/2 mouse combination device is also a USB Human Interface Device (HID). All USB HID-class devices follow the same USB start-up procedure. The procedure is shown in *Figure 7*.

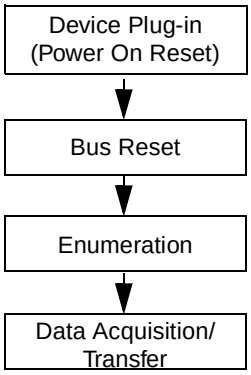

Figure 7. USB Start-Up Procedure

However, designing a USB HID combination device (keyboard and mouse functions) requires different approaches in phases such as USB enumeration and data acquisition/transfer. We will discuss these differences in the following sections. Device Plug-in and Bus Reset phases remain basically the same for the combination device as for the keyboard-only design.

Enumeration

A combination device follows the same usual USB enumeration process, where the device responds to the host with a number of USB descriptors. In this design, the combination device chooses to transmit both the keyboard and mouse data reports through a single interface, and over the same interrupt endpoint (endpoint 1). This means that the HID report descriptor needs to be modified to describe both the keyboard and mouse data reports. All the other USB descriptors (device, configuration, interface, class, and endpoint descriptors) can remain the same as in the keyboard-only design.

HID Report Descriptor

The HID report descriptor is basically divided into two application collections, corresponding to the keyboard and the mouse collections. To allow the host system to distinguish different reports arriving over the same interrupt endpoint, a Report ID tag is added to each of the two reports. The Report ID value indicates the prefix added to a particular report. For example, the report descriptor defines a 3-byte mouse report with a report ID of 02. Therefore, whenever the combination device needs to send a mouse report in response to an IN packet on endpoint 1, it will generate a 4-byte data report, in which the first byte is the report ID 02. The keyboard report is assigned a report ID of 01.

```
Example of HID Report Descriptor

Usage Page (Generic Desktop)
Usage (Keyboard)
Collection (Application)
   ReportID (0x01)
   Usage Page (Key Codes)
   Usage Minimum (234)
   Usage Maximum (231)
   Logical Minimum (0)
   Logical Maximum (1)
   Report Size (1)
   Report Count (8)
   Input (Data,Variable,Absolute)

   Report Count (1)
   Report Size (8)
   Input (Constant);reserved byte

   Report Count (5);LED report
   Report Size (1)
   Usage Page (LEDs)
   Usage Minimum (1)
   Usage Maximum (5)
   Output (Data,Variable,Absolute)
   Report Count (1)
   Report Size (3)
   Output (Constant);3-bit padding

   Report Count (6)
   Report Size (8)
   Logical Minimum (0)
   Logical Maximum (101)
   Usage Page (Key Codes)
   Usage Minimum (0)
   Usage Maximum (101)
   Input (Data, Array);key array(6)
End Collection

Usage Page (Generic Desktop)
Usage (Mouse)
Collection (Application)
   ReportID (0x02)
   Usage (Pointer)
   Collection (Linked)
      Usage Page (Buttons)
      Usage Minimum (1)
      Usage Maximum (3)
      Logical Minimum (0)
      Logical Maximum (1)
      Report Count (3);three buttons
      Report Size (1)
      Input (Data,Variable,Absolute)
      Report Count (1)
      Report Size (5)
      Input (Constant);5-bit padding

      Usage Page (Generic Desktop)
      Usage (X)
      Usage (Y)
      Logical Minimum (-127)
      Logical Maximum (127)
      Report Size (8);X and Y data
      Report Count (2)
      Input (Data,Variable,Variable)
   End Collection
End Collection
```

A more detailed description of HID and all the items used in an HID report descriptor can be found in the "Device Class Definition for Human Interface Devices (HID) Revision 1.0."

Data Acquisition/Transfer

The firmware alternately scans the keyboard scan matrix and polls the PS/2 mouse to determine whether data reports need to be sent to the host system. As mentioned earlier, both the keyboard and mouse data reports are sent to the host using endpoint 1. When the firmware determines there is a key or many keys pressed, it sets up the keyboard data report, loads it into the endpoint 1 FIFO, and enables the endpoint 1 transmission. Then the keyboard data report will be sent when the next IN packet is received on the endpoint 1 from the host. The PS/2 mouse follows the same steps to send its data report to the host.

The HID keyboard data report is 8 bytes long, and consists of one byte of modifiers plus 7 key scan code bytes. *Table 4* shows the format of the keyboard report. Usage codes for each key can be found in Appendix A.3 of the "Device Class Definition for Human Interface Devices (HID)."

Since the maximum data packet size of an interrupt pipe for a low-speed USB device is limited to 8 bytes, the device needs to send the complete report (1-byte report ID plus 8-byte data report) in two interrupt transactions, instead of only one in the keyboard-only design. In the first interrupt transaction, a report ID and the first 7 bytes of the data report are sent in response to a IN token. In the second transaction, a report ID and the last byte of the data report are sent in response to the following IN token.

When any LED keys (i.e., Num Lock, Caps Lock, Scroll Lock) are pressed or released, the host system issues a SETUP packet with a Set_Report request through the control pipe to End Point 0, followed by an OUT packet with 2 data bytes. The first byte corresponds to the report ID and the second byte indicates which LED should be on or off (see *Table 5*.)

Table 4. USB Keyboard Data Report Format

Byte	b7	b6	b5	b4	b3	b2	b1	b0
0 Modifiers	Right GUI	Right Alt	Right Shift	Right Ctrl	Left GUI	Left Alt	Left Shift	Left Ctrl
1	Reserved							
2	Key 1 scan code							
3	key 2 scan code							
4	Key 3 scan code							
5	Key 4 scan code							
6	Key 5 scan code							
7	Key 6 scan code							

Table 5. USB Keyboard Set_Report Data Format for LED

Byte	b7	b6	b5	b4	b3	b2	b1	b0
0 LED Report	Con-stant	Con-stant	Con-stant	Kana	Com-pose	Scroll Lock	Caps Lock	Num Lock

The HID mouse data report is 3 bytes long (see *Table 6*). The firmware polls the PS/2 mouse and obtains the PS/2 mouse data packet (see *Table 2*). Then it extracts the button and X, Y movement information from the PS/2 data packet to compose the HID mouse data report. Different from the keyboard report, the complete mouse report can be sent in a single USB interrupt transaction since it is only 4 bytes long (1 byte of report ID and 3 bytes of mouse data).

The byte order and bit field positions for both the keyboard and mouse reports are consistent with the *Boot Protocol* requirements for legacy systems.

Table 6. USB Mouse Data Report Format

Byte	b7	b6	b5	b4	b3	b2	b1	b0
0	Padding					Left Alt	Left Shift	Left Button
1	Horizontal (X) Displacement							
2	Vertical (Y) Displacement							

PS/2 Interface

State Machine Implementation

The USB keyboard and PS/2 mouse combination device firmware is an extension of the keyboard-only firmware, with the addition of PS/2 interface code and changes to allow functional integration. The PS/2 interface code consists of a number of subroutines which can be called by the main code to perform PS/2 related tasks. In this application note, we will mainly discuss the principle of operation and how to utilize these subroutines to operate a PS/2 mouse. For implementation details of the subroutines, please refer to the firmware source code.

The PS/2 mouse interface code is implemented by two state machines: *mouse_int* and *mouse_machine.* The former operates at the bit/byte transaction level and the latter operates at the byte/command transaction level. The *mouse_int* state machine manages the transmission and reception of data bytes over the PS/2 bus. Since the PS/2 mouse always drives the clock signal connected to a GPIO pin, the *mouse_int* state machine is driven by GPIO interrupts generated from the falling edge of the PS/2 clock. In the GPIO interrupt service routine, the microcontroller will either read from or write to the PS/2 data line, depending on whether it is receiving data from or sending data to the PS/2 mouse.

The states of *mouse_int* corresponds to bit states of a PS/2 byte transmission or reception (i.e., start bit, data bits, parity and stop bit). The *mouse_int* is initiated either by a byte transmission request from the high-level state machine, *mouse_machine*, or by the start of a new receive byte from the mouse. Upon completion of a byte transfer (normal or otherwise) by *mouse_int*, the PS/2 bus is inhibited (by holding

the clock line LOW) to allow the high-level state machine, *mouse_machine*, to act.

The high-level state machine (*mouse_machine*) is, in turn, driven by periodic calls from the application (the keyboard and PS/2 mouse firmware in this case). Whenever we initiate a mouse command, *mouse_machine* must be called regularly from the application in order to process its state sequences. Making these regular calls to *mouse_machine* is referred to as "operating" the *mouse_machine*. Every call to *mouse_machine* returns an idle/busy indication in the microcontroller's C-bit (carry). It returns C-bit = 0 if it is idle (i.e., no mouse commands are currently in process), and C-bit = 1 if a mouse command is currently being processed. No new mouse command should be initiated by the application unless *mouse_machine* is idle. Further, *mouse_machine* can only be made busy by a command initiated by the application or a power-up reset.

A sample code in *Figure 8* shows how to initiate a mouse command and "operate" *mouse_machine*. Call to *mouse_poll* initiates the Read Data command to the mouse, and sets the *mouse_machine* to the corresponding state. The subsequent loop calls *mouse_machine* to sequence through the states until the C-bit = 0, indicating the state machine is done and idle.

```
MouseTask:
      call mouse_poll      ;send poll cmd
tw3:
      call mouse_machine   ;wait until done
      jc tw3               ;jump back if C= 1
```

Figure 8. Operating *mouse_machine*

PS/2 Support Subroutines

A number of support subroutines are provided to facilitate the operation of the PS/2 mouse. The main firmware code calls these subroutines to initialize, set operation mode and read data from the PS/2 mouse. Their descriptions and usages are as follows:

- ***ps2_init**:* this subroutine initializes the PS/2 mouse port. It must be called from the main code within 500 ms of power-up, before the PS/2 mouse starts transmitting. Upon application of power and completion of its self-diagnostics, the mouse will automatically transmit a two-byte initialization pattern (0xAA and 0x00). Regular calls to *mouse_machine* must be made after the call to *ps2_init* until it returns idle, indicating the transmission of the initialization pattern is complete. The receipt of the pattern recognizes that the mouse is attached to the device.
- ***mouse_reset**:* this subroutine initiates a Reset command to the mouse, causing it to execute its self-diagnostics and return the initialization pattern. After a call to *mouse_reset*, *mouse_machine* must be operated until it returns idle. In this instance, the returned data from a properly functioning mouse will be a three-byte string (0xFA, 0xAA, and 0x00). The first byte is the ACK byte to the Reset command, and the next two bytes are the initialization pattern. Note that the initialization pattern will only be transmitted automatically by the mouse at power-up. Therefore, if the USB microcontroller gets reset by some reason, it is recommended that *mouse_reset* be called to duplicate the reset/initialization sequence, in order to make sure that the firmware recognizes the mouse is attached.
- ***ps2_status():*** this subroutine can be called once *mouse_machine* goes idle after the reset/initialization sequence to determine if a mouse is attached. It returns "0' in the accumulator (register A) if no mouse is attached; otherwise, "1" is returned.
- ***mouse_init**:* Once the mouse has gone through the reset/initialization phase and is detected as a result, *mouse_init* must be invoked by the main code to prepare the mouse for data polling. This is done by sending the Set Remote Mode and Enable commands to the mouse. *Mouse_init* operates the *mouse_machine* through the command sending transactions, and only returns when the mouse is properly configured. *Mouse_init* is also used to reinitialize the mouse upon leaving the suspend mode. Suspend/resume and remote wakeup issues will be discussed in the next section.
- ***mouse_poll**:* Upon successful completion of *mouse_init*, data may be polled from the mouse by calling the *mouse_poll* subroutine. Note that *mouse_poll* merely initiates the transaction, which is then driven to completion by operating *mouse_machine* until it returns idle. When the transaction is completed, the 3-byte PS/2 mouse data report will be available in RAM area at address *mouse_packetV*.
- ***mouse_suspend***: this subroutine is called before the microcontroller enters suspend state. It initiates a Set Stream Mode command and operates *mouse_machine* until the transaction is complete.

Keyboard and Mouse Firmware Integration

Main Loop

The main loop of the firmware (*Figure 9*) runs two tasks continuously: *ScanTask* and *MouseTask*. The *ScanTask* is the same keyboard task in the keyboard-only firmware. It scans the keyboard scan matrix to determine any key status change, loads the keyboard report to endpoint 1 FIFO and

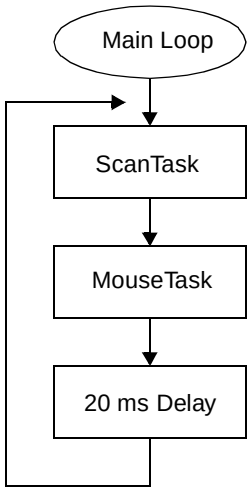

Figure 9. Main Loop

sends it to the host system if necessary. For more details on its operation please see the USB keyboard application note. Similarly, the *MouseTask* calls *mouse_poll* subroutine to read data from the PS/2 mouse, copies the mouse report from memory buffer to endpoint 1 FIFO and transmits it to the host system. An arbitrary delay of 20 ms was added before the code loops back. This is to make sure that the PS/2 mouse gets ready to respond to the next polling command.

USB Suspend/Resume and Remote Wakeup

According to the USB Specification 1.0, a device has to go into suspend mode after 3 ms of no bus activity. The 1 ms ISR determines when this condition occurs (by reading the Bus Activity bit, bit 3, of the USB Status and Control Register, 1Fh) and suspends the microcontroller. This is accomplished by setting bit 0 (Run bit) and bit 3 (Suspend bit) of the Processor Status and Control Register (FFh). The microcontroller will resume operation upon one of the following three events: USB bus activity, keyboard activity, or PS/2 mouse activity.

USB bus activity resume is carried out automatically by the microcontroller. On the other hand, keyboard and PS/2 mouse activity during suspend are detected through GPIO interrupts. For this reason, before entering suspend, GPIO interrupts for the keyboard and mouse must be properly enabled.

For the keyboard, prior to suspending the microcontroller, the 1 ms ISR pulls down all the column port lines and enable a falling edge interrupt on all Port 2 lines (Row port). If any key is pressed while in suspended state, one of the row lines will be pulled LOW and trigger a GPIO interrupt. The GPIO ISR will then send a resume signal (force a K state where D+ is HIGH and D– is LOW for 10 ms) to wake up the host system (remote wake up).

The PS/2 mouse normally operates in Remote (polling) mode, where the mouse only transmits a data report in response to a Read Data command. When the microcontroller enters suspend state, however, it no longer polls the mouse for data. Thus, it is desirable to allow the mouse to be capable of autonomously waking up the microcontroller if it has data to transmit (mouse movement or button press). This is accomplished by placing the mouse in Stream mode before entering suspend. In the Stream mode, the PS/2 mouse will transmit data at regular intervals any time it has new data available, generating GPIO interrupts and waking up the microcontroller.

Before entering suspend the main code must call function *mouse_suspend*, which initiates a Set Stream Mode command and operates *mouse_machine* until the transaction is complete. Then the main code will enable suspend, leaving the Port 3 GPIO interrupt enabled. During suspend state, any activity on the PS/2 port will cause an entry to the GPIO ISR, which will immediately inhibit the mouse transmission. During the wakeup sequence, the firmware must also call *mouse_init* to place the mouse back to Remote mode for normal data polling operations.

GPIO Interrupt Service Routine

In the keyboard-only firmware, the GPIO ISR was only used to handle the keyboard resume and the sending of the remote wakeup signal after leaving the suspend state. For the keyboard and mouse combination device, however, the GPIO interrupt is also used to drive the PS/2 low-level state machine (*mouse_int*) at each falling edge of the PS/2 clock signal.

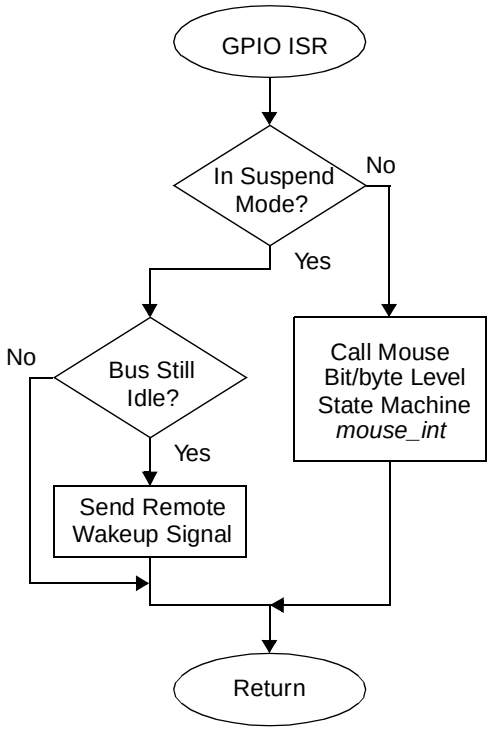

Figure 10. GPIO Interrupt Service Routine

Upon entering the GPIO ISR (See *Figure 10*), a test is done to determine whether the firmware is running in normal operation mode or has just left suspend mode. If it is in normal operation, the mouse_int state machine is called; otherwise, the remote wakeup sequence begins.

1-ms Interrupt Service Routine

For the keyboard and mouse combination firmware, a few changes were added to the keyboard-only 1 ms ISR (*Figure 11*):

- The keyboard scan task is no longer scheduled every 4 ms as in the keyboard-only firmware.
- Before entering suspend state, the PS/2 mouse is placed in Stream mode. Also, after coming out of suspend, the mouse is set back to Remote mode.
- The PS/2 mouse time-out counter increment and check is included. If the PS/2 is in the active state for more than 25 ms and no transfer happens, a time-out condition will occur, resetting the PS/2 bus and setting the state machines to idle.

PS/2 Signal Timing Issue

For reliable PS/2 bus operation, the GPIO interrupt service provided by the main code must ensure that the latency between a falling edge on the PS/2 clock line and resultant entry into the bit/byte level state machine (*mouse_int*) is no more than 17 µs. This requirement precludes the disabling of interrupts for long periods of time by other code modules, while *mouse_machine* is busy. Special attention should be given to the 1 ms ISR or other ISRs that may disable interrupt for long periods of time.

Conclusion

The two main enabling factors of the proliferation of the USB devices are cost and functionality. The CY7C63413 meets both requirements by integrating the USB SIE and multi-function I/Os with a USB optimized RISC core in a low-cost part. In this application note, the CY7C63413 USB microcontroller is used to implement a USB keyboard and PS/2 mouse combination device. The PS/2 interface, fully implemented in firmware, preserves the use of a legacy PS/2 mouse without any additional cost.

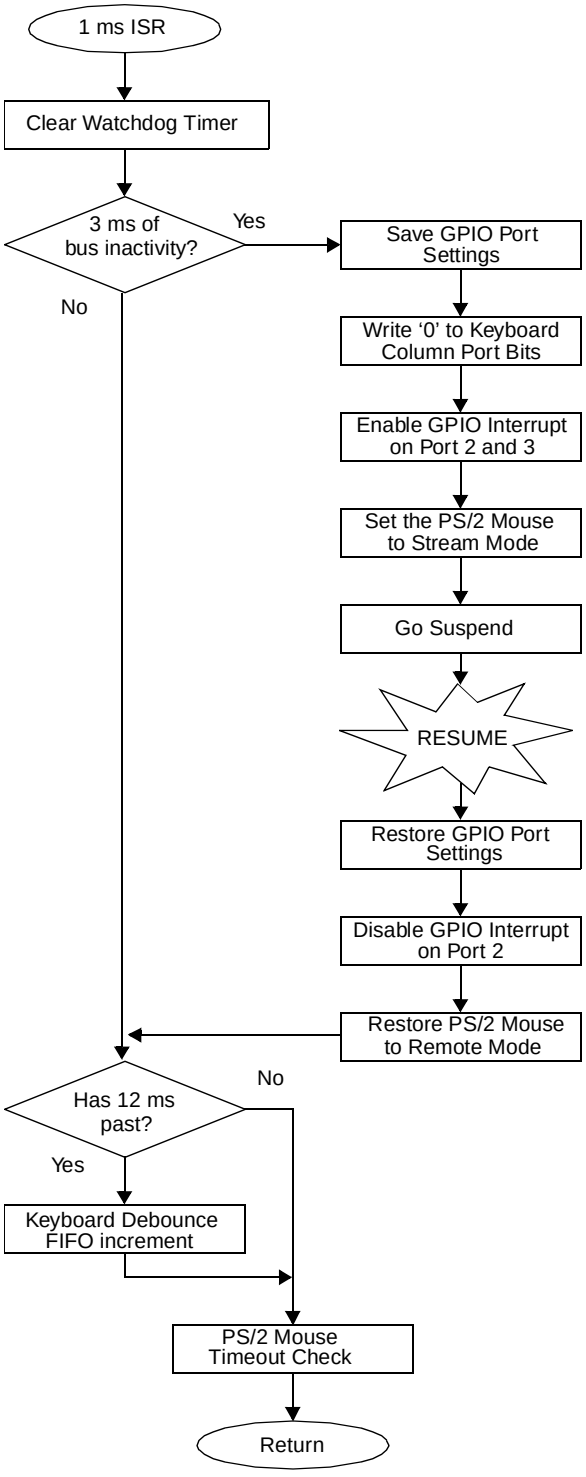

Figure 11. 1 ms Interrupt Service Routine

PS/2 is a registered trademark of the International Business Machines Corp.