Designing A Low Cost USB-PS/2 Combination Interface Mouse with the Cypress Semiconductor CY7C63723 enCoRe™ USB Microcontroller

Introduction

The Universal Serial Bus (USB) is an industry standard serial interface between a computer and peripherals such as a mouse, joystick, keyboard, UPS, etc. This application note describes how a cost-effective combination USB-PS/2 mouse can be built using the Cypress Semiconductor CY7C63723 USB microcontroller. The document starts with the basic operations of a computer mouse peripheral followed by an introduction to the CY7C63723 USB microcontroller. A schematic of the CY7C63723 USB microcontroller to the optics and buttons of a standard mouse 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 USB and PS/2 mouse functions. This application note assumes that the reader is familiar with the CY7C63723 USB microcontroller and the Universal Serial Bus standard. The CY7C63723 data sheet is available from the Cypress web site at www.cypress.com. USB documentation can be found at the USB Implementers Forum web site at www.usb.org.

Mouse Basics

A standard PC mouse has a switch for each button and a ball and rollers to measure movement. The rollers are connected to optics that output quadrature information, which can be deciphered into movement and direction information. This application note shows how to connect to and manage a standard configuration of mouse hardware, as well as handle the USB and PS/2 protocols. Each of these protocols provides a standard way of reporting mouse movement and button presses. For a better understanding of these protocols you should have four documents:

Universal Serial Bus Specification Revision 1.1, September 23, 1998
Available on-line from the USB Implementers Forum at http://www.usb.org.

Device Class Definition for HID Version 1.1, April 7, 1999
Available on-line from the USB Implementers Forum at http://www.usb.org.

IBM Personal System/2 Mouse Technical Reference
IBM Document #S68X-2229-00

IBM Personal System/2 Hardware Interface Technical Reference - Common Interfaces
IBM Document #S84F-9809-00

Introduction to the CY7C63723

The CY7C63723 is an 8-bit RISC microcontroller with an integrated USB Serial Interface Engine (SIE). The architecture executes general-purpose instructions that are optimized for USB applications. The CY7C63723 has a built-in clock oscillator and timers as well as programmable drive strength and pull-up resistors on each I/O line. High performance, low-cost human-interface type computer peripherals can be implemented with a minimum of external components and firmware effort. For a detailed look at the CY7C63723 please see the following document:

CY7C63723/23 CY7C63742/43 *enCoRe™* USB Combination Low-Speed USB & PS/2 Peripheral Controller
Available on-line from the Cypress Semiconductor web at http://www.cypress.com

Clock Circuit

The CY7C63723 has a crystal-less oscillator circuit that eliminates the need for an external resonator or crystal. This also frees up an additional input on the XTALIN line. At power up the oscillator will be within 5% of nominal 6MHz. Then as USB traffic starts up, the oscillator will self tune itself to within 1% of the 6MHz reference from low speed USB traffic. An external oscillator can be used by changing the value of the clock configuration register via firmware if your application demands stand alone clock accuracy.

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 request to invoke the 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 CY7C63723 has 16 general purpose I/O lines divided into 2 ports: Port 0 and Port 1, as well as four special input only pins.

A general purpose I/O pin can have two selectable drive strengths: CMOS and resistive. CMOS mode is a general-purpose mode used for driving logic and other relatively fast signaling. Resistive mode was designed to act as a pull up resistor with a value of about 14kΩ, which in this design will be used for pull up resistors on the button switches.

The CY7C63723 also has three sink modes including a super strong 50mA mode for driving, otherwise hard to drive, optics. It should be noted that this microcontroller has a current sink limit of 70mA total on all I/O pins at one time.

In addition to these output modes, this microcontroller has two user selectable input thresholds for interfacing with TTL and CMOS level signaling.

Four additional inputs are available on the CY7C63723. If the crystal-less oscillator is used, then the XTALIN pin can act as an input. When the VREG output is disabled, this pin may also act as an input. The other two inputs that are used for PS/2 communications are the D+/D- pins, normally used for the USB interface. A configuration register also allows you to select a number of configurations for these pins, meaning that USB and PS/2 can co-exist on the same signal pins.

Voltage Regulator

The CY7C63723 has a built in 3.3V regulator that is designed to drive the USB D- pull up resistor. This regulator supplies a small amount of current and should not be used as a general-purpose regulator. The output resistance of this regulator is 200Ω, which necessitates a 1.3kΩ resistor as a pull up on the D- line to create the 1.5kΩ resistor required by the USB specification.

The VREG pin is automatically turned off by the SIE (Serial Interface Engine) during USB data transmission, so this pin should not be used as a general-purpose voltage reference.

When disabled, the VREG pin can act as an input and is read via the port2 register.

Hardware Implementation

The standard hardware to implement a mouse is shown in figure 1.0. For each axis (x & y for the mouse ball, z for the wheel) there are a set of optics that output quadrature signals. For each button there is a switch that is pulled up internally by the built in pull up resistors. The D- line is pulled up via a 1.3kΩ resistor connected to the VREG pin.

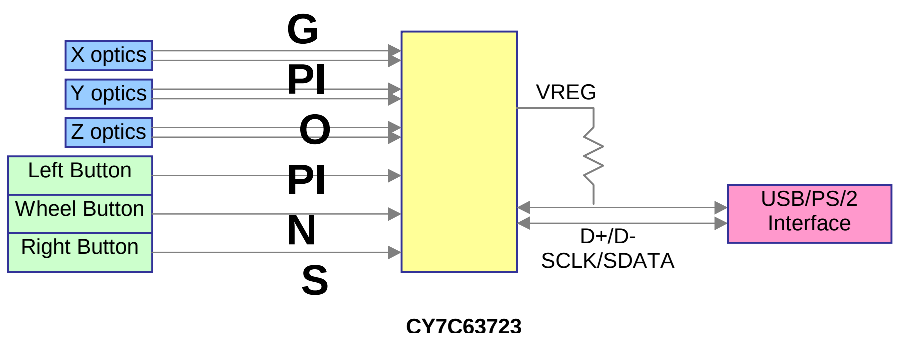


Figure 1.0 Standard Mouse Hardware Connected to a CY7C63723

Firmware Configurable GPIO

The reference firmware is configured to use the GPIO pins as shown on the schematic in appendix A. However, it may be more optimal for you to use a different IO configuration to meet the mechanical constraints of your particular PCB. The reference firmware is designed to be easily configured to another set of pin connections. This is accomplished through changes in the IO definitions at the beginning of the *combi.c* listing. The following statements are the pin definitions as they exist today. The firmware will use these definitions to read and configure the GPIO pins, without any other modifications.

```
#define OPTICS_PORT     PORT0
#define OPTICS_MASK     0x3f

#define GET_X_OPTICS(x) ((x >> 0)&0x3)
#define GET_Y_OPTICS(x) ((x >> 2)&0x3)
#define GET_Z_OPTICS(x) ((x >> 4)&0x3)

#define LEFT_SWITCH_PORT      PORT0
#define LEFT_SWITCH_MASK      BIT7
#define RIGHT_SWITCH_PORT     PORT1
#define RIGHT_SWITCH_MASK     BIT0
#define MIDDLE_SWITCH_PORT    PORT1
#define MIDDLE_SWITCH_MASK    BIT1

#define PORT0_OPTICS_MASK     0b00111111
#define PORT1_OPTICS_MASK     0b00000000
#define PORT0_LED_MASK        0b01000000
#define PORT1_LED_MASK        0b00000000
#define PORT0_SWITCH_MASK     0b10000000
#define PORT1_SWITCH_MASK     0b00000011
```

Mouse Optics

The standard way mouse optics are connected are via two phototransistors connected in a source follower configuration. An infrared LED shines, causing the phototransistors to turn on. In between the phototransistors and LED is a pinwheel that turns on the mouse ball rollers. The fan of this pinwheel is mechanically designed to block the infrared light such that the phototransistors are turned on and off in a quadrature output pattern. Every change in the phototransistor outputs represents a count of mouse movement. Comparing the last state of the optics to the current state derives direction information. As is shown in figure 2.0 below, travelling along the quadrature signal to the right produces a unique set of state transitions, and travelling to the left produces another set of unique state transitions. In this reference design, there are three sets of phototransistors. They are assigned to the x-axis, y-axis, and z-axis (wheel).

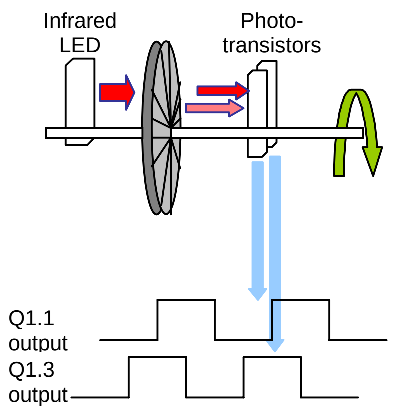


Figure 2.0 Optics Quadrature Signal Generation

Mouse Buttons

Mouse buttons are connected as standard switches. These switches are pulled up by the pull up resistors inside the microcontroller. Normally the switches are debounced in firmware for 15-20ms. In this reference design there are three switches: left, wheel, and right.

USB and PS/2 Connection

The CY7C63723 has a configuration register that switches control from the SIE to manual control on the D+ and D- pins. This allows the firmware to dynamically configure itself to operate on the bus you plug the mouse into. This way the signaling lines for USB and PS/2 can be shared without taking extra GPIO pins for PS/2 operation. The firmware for this reference design will automatically detect the host topology (USB or PS/2) at plug-in and will configure itself for operation on that bus.

If a USB host connection is detected then the firmware will enable the VREG pin, such that the 1.3kΩ resistor connected to the D- line can be pulled up to 3.3V. It is this action that causes the host to recognize that there is a low-speed USB peripheral attached.

The connections for the connectors are shown in figure 3.0 below.

USB type-A connector

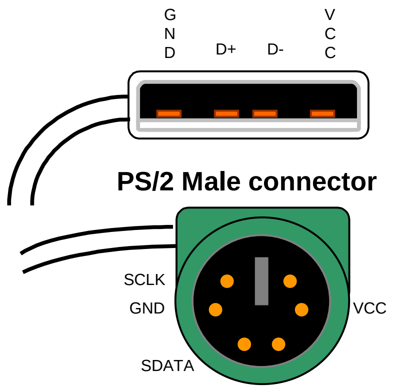


Figure 3.0 USB and PS/2 peripheral connectors

Overall circuit

A schematic of the overall circuit is shown in Appendix A of this document. Appendix B lists the bill of materials.

Firmware Implementation

The firmware for this reference design is written in the C language and requires the ByteCraft M8 C-compiler. The following files are required to compile the mouse firmware

Chip.h – interrupt table, register and port constants for the CY7C63723 microcontroller

Combi.h – main mouse firmware

Macros.h – general macros used with this design

Ps2defs.h – PS/2 interface constants

Usb_desc.h – USB descriptor tables

Usbdefs.h – USB interface constants

At power up, the firmware tests the host interface and automatically determines if the mouse if plugged into a USB or a PS/2 host connection. After interface detection, the host firmware configures itself to operate on the detected interface.

USB Interface

All USB Human Interface Device (HID) class applications follow the same USB start-up procedure. The procedure is as follows

1. Device Plug-in
When a USB device is first connected to the bus, it is powered and is running firmware, but communications on the USB remain non-functional until a USB bus reset by the host.

2. Bus Reset
The pull-up resistor on D– notifies the hub that a low-speed (1.5 Mbps) device has just been connected. The host recognizes the presence of a new USB device and initiates a bus reset to that device.

3. Enumeration
The host initiates SETUP transactions that reveal general and device specific information about the mouse. When the description is received, the host assigns a new and unique USB address to the mouse. The mouse begins responding to communication with the newly assigned address, while the host continues to ask for information about the device description, configuration description and HID report description. Using the information returned from the mouse, the host now knows the number of data endpoints supported by the mouse (2). At this point, the process of enumeration is completed. The USB descriptors used during enumeration are listed in Appendix C.

4. Post Enumeration Operation
Once the communications connection between the host and mouse is established, the peripheral now has the task of sending and receiving data on the control AND data endpoints. In this case, when the host configures endpoint 1, the mouse starts to transmit button and motion data back to the host when there is data to send. At any time the peripheral may be reset or reconfigured by the host.

USB Requests – Endpoint 0

Endpoint 0 acts as the control endpoint for the host. At power up endpoint 0 is the default communication channel for all USB devices. The host initiates Control-Read and Control-Write (see Chapter 8 of the USB specification) to determine the device type and how to configure communications with this device. In this particular design, only Control-Read transactions are required to enumerate a mouse. For a list of valid requests see Chapter 9 of the USBG specification. In addition to the standard "Chapter 9" requests, a mouse must also support all valid HID requests for a mouse. Appendix C of this document lists the valid requests that are recognized by the mouse firmware.

USB Requests – Endpoint 1

Endpoint 1 is the data transfer communications channel for mouse button, wheel, and movement information. Requests to this endpoint are not recognized until the host configures endpoint 1. Once this endpoint is enabled, then interrupt IN requests are sent from the host to the mouse to gather mouse data. When the mouse is left idle (i.e. no movement, no new button presses, no wheel movement) the firmware will NAK requests to this endpoint. Data is only reported when there is a status change with the mouse.

Two reporting formats are used in this design. The boot protocol, as defined by the HID specification, is the default reporting protocol that all USB enabled systems understand. The boot protocol has a three-byte format, and so does not report wheel information. The HID report descriptor defines the report protocol format. This format is four bytes and is the same as the report format with the exception of the fourth byte, which is the wheel information.

PS/2 Interface

The host driver determines the PS/2 mouse start up sequence. However, a few standard commands must be sent in order to enable all PS/2 mice.

The mouse is the clock master on this bus. The host must request the mouse to clock data into itself.

1. Device Plug-in
When a PS/2 mouse is first connected to the bus, it is powered and is running firmware. PS/2 communications generally begin with the host sending a RESET command to the mouse. The mouse will not report button, wheel, or movement back to the host until the ENABLE command is sent. Depending on the particular operating system the mouse is used with, the start up sequence will vary.

2. Device Configuration
During this time the host will set the standard PS/2 parameters such as scaling, resolution, stream mode, and eventually enabling stream mode for data reports. For a list of the valid PS/2 commands that this mouse recognizes see Appendix E.

3. Wheel Enable (optional)
Since the wheel is not part of the standard PS/2 specification, there is a sequence of commands that enable the wheel. Wheel-aware drivers, such as those for Microsoft and Linux operating systems will initiate this special sequence.

After the following sequence of commands, the wheel report format is enabled.

0xF3, 0xC8	Set Sampling Rate 200 per second
0xF3, 0x64	Set Sampling Rate 100 per second
0xF3, 0x80	Set Sampling Rate 50 per second
0xF2, 0x03	Read Device Type returns a value of 0x03

After the Read Device Type command returns 0x03 to indicate that this is a Microsoft compatible three button-wheel mouse, the wheel report format is enabled. See Appendix E for information on PS/2 standard and wheel reporting formats.

4. Post Start Up Operation
After the streaming mode is set and data reports are enabled, the mouse will send button, movement, and optionally wheel reports back to the host. Whenever the mouse has new data to send it will initiate a transfer to the host.

USB Firmware Description

A function call map for USB operation is shown in figure 4.0. The following are descriptions of the functions in *combi.c*.

USB Functions

void UsbReInitialize(void) - Wake up and delay 50mS. Initialize the PS2 BAT delay counter. For a period of 2mS, poll the SCLK and SDATA lines every 10uS. If we get 4 samples in a row with non-zero data on either line, detect PS2. If 2mS expires, enable the USB pull up resistor and delay 500uS. Poll the SCLK and SDATA lines indefinitely until a non-zero condition exists on either line. During this polling period, we begin to count down the PS2 BAT delay. If SCLK(D+) is sampled high, detect PS2. If SDATA(D-) sampled high, disable the USB connect resistor and Delay 100uS. If D+ and D- are both 0, detect a USB interface, else detect a PS2 interface.

void MouseTask(void) - This routine is called every 4 msec from the main loop. It maintains the idle counter, which determines the rate at which mouse packets are sent to the host in the absence of a state change in the mouse itself. It also sends a mouse packet if either of X, Y, or Z counts or the buttons have changed state.

void Suspend(void) - This routine handles the entrance/exit from suspend. If the mouse is configured for remote wakeup, the bus reset, wakeup, and GPIO interrupts are enabled. The optical inputs are sampled once. The code then enters a loop in which the chip is suspended, and will wake at least as often as the wakeup ISR, but perhaps due to a GPIO or a USB bus reset interrupt. Each time the chip wakes up, the LED drive is re-enabled, and the switches and optical inputs are sampled to see if a change occured, and bus activity is monitored. Any of these conditions will cause the firmware to exit the loop. If the device is not enabled for remote wakeup, all ports are put into the high-Z state, only the USB bus reset interrupt is enabled, and the part is suspended. If the resume was due to bus activity, the firmware returns to the main loop. If the resume was due to mouse movement or a button press, a K state is driven upstream for 14 milliseconds prior to returning to the main loop.

void usbmain(void) – This function spins in an infinite loop waiting for an event that needs servicing. Main is entered from either the power-on reset or the USB bus reset. Both of these reset routines insures that all USB variables have been initialized prior to calling *usbmain().*

void HandleSetup(void) - This routine is entered whenever an SETUP packet has come in on endpoint 0. It performs some preliminary validation on the packet, then parses the packet and calls the appropriate routine to handle the packet.

void HandleIn(void) - This routine is entered whenever an IN packet has come in on endpoint 0.

void USB_control_read(void) - This routine is called after a SETUP has been received that is requesting data response from the mouse. Upon entry, XmtBuff has been initialized to point to the data buffer that needs to be transmitted. This function adjusts the length of the data to be returned if the host requested less data than the actual length of the buffer. It then loads the FIFO with the first packet of data and prepares the SIE to ACK with the data.

char LoadEP0Fifo(void) - This routine loads the USB endpoint 0 FIFO with data pointed to by XmtBuff. It then returns the length of data actually loaded into the FIFO.

void SetConfiguration(void) - This routine is entered when a SET CONFIGURATION request has been received from the host.

void SetAddress(void) - This routine is entered whenever a SET ADDRESS request has been received. The address change cannot actually take place until after the status stage of this no-data control transaction, so we just save the address and set a flag to indicate that a new address was just received. The code that handles IN transactions will recognize this and set the address properly.

void GetDescriptor(void) - This routine is entered when a GET DESCRIPTOR request is received from the host. This function decodes the descriptor request and initializes XmtBuff to the proper descriptor, then initiates the transfer by calling *USB_control_read().*

void SetIdle(void) - This routine is entered whenever a SET IDLE request is received. See the HID spec for the rules on setting idle periods. This function sets the HID idle time. See the HID documentation for details on handling the idle timer.

void SetProtocol(void) - This routine is entered whenever an SET PROTOCOL request is received. This no-data control transaction enables boot or report protocol.

void GetReport(void) - This routine is entered whenever a GET REPORT request to get a report is received. The most recent mouse data report is sent to the host via a control-read transaction on endpoint 0, instead of endpoint 1.

void GetIdle(void) - This routine is entered whenever a GET IDLE request is received. This function then initiates a control-read transaction that returns the idle time. See the HID documentation for more details.

void GetProtocol(void) - This routine is entered whenever a GET PROTOCOL request is received. This request initiates a control-read transaction that tells the host if the mouse is configured for boot or report protocol. See the HID class documentation for more details.

void GetConfiguration(void) - This routine is entered whenever a GET CONFIGURATION Request is received. This function then starts a control read transaction that sends the configuration, interface, endpoint, and HID descriptors to the host.

void USB_Stall_In_Out(void) – This function sets endpoint 0 to stall IN and OUT tokens from the host. Unsupported or invalid descriptor requests will cause this firmware to STALL these transactions.

char BusInactive(void) - This routine should be called every millisecond from the main loop. It maintains an internal count of the successive samples of the USB status register in which no bus activity was recorded. When this count exceeds 3 milliseconds, this function returns 1, indicating that bus activity has suspended. When the bus activity suspends for more than 3 milliseconds, the mouse must enter a low power state until a wakeup even

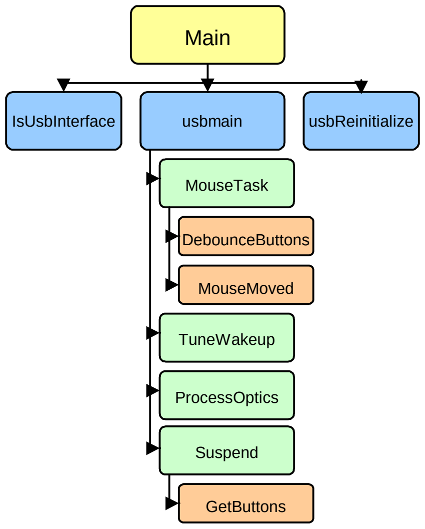


Figure 4.0 USB Operation Function Call Map

PS/2 Firmware Description

A function call map for PS/2 operation is shown in figure 5.0. The following are descriptions of the functions in *combi.c*.

PS/2 Functions

void ps2BAT(void) - delays for 500 milliseconds, then sends the AA 00 initialization string to the host for the PS/2 Basic Assurance Test.

void ps2_send(char data) - This routine sends a byte to the host according to the standard PS/2 protocol. This routine is written in assembler to precisely control the number of execution cycles required for the appropriate PS/2 timing.

char ps2_receive(void) - This routine receives a byte from the host according to the standard PS/2 protocol. This routine is written in assembler in order to control the receive timing.

void Reset(void) - This routine simply waits in a loop for the watchdog to perform a reset.

void Resend(void) - A copy of the last transmission is always left intact in the message buffer. To re-send it , this routine simply resets the message length.

void SetDefault(void) – This routine is called in response to a SET DEFAULT command from the host. It then sets the mouse parameters to the default settings.

void Disable(void) - Disables the mouse.

void Enable(void) - Enables the mouse.

char SetSampleRate(char p) - This routine is called in response to a SET SAMPLE RATE command from the host. It then verifies that the requested sample rate is valid and sets the sample rate for the mouse. Valid sample rates are defined in the PS/2 Mouse specification.

void ReadDeviceType(void) - This routine is called in response to a READ DEVICE TYPE request from the host. This mouse always sends a 0x00 in response to this request.

void SetRemoteMode(void) – This routine is called in response to a SET REMOTE MODE command from the host. The PS/2 mode is then set to remote.

void SetWrapMode(void) - This routine is called in response to a SET WRAP MODE command from the host. It then sets the mouse mode to wrap. See the PS/2 specification for more details on wrap mode.

void ResetWrapMode() -This routine is called in response to a RESET WRAP MODE command from the host. The mode is then reset to the previous mode. According to the IBM spec, if stream mode is enabled, the mouse is disabled when the wrap mode is reset.

void ReadData(void) - This routine is called in response to a READ DATA command from the host. This routine then sends a mouse packet in response to the command

void SetStreamMode(void) - This routine is called in response to a SET STREAM MODE command from the host. Stream mode is then enabled. See the PS/2 specification for more information about stream mode.

void StatusRequest(void) - This routine is called in response to a STATUS REQUEST command from the host. A three byte report is sent to the host in response to this request. See the PS/2 mouse specification for more details.

char SetResolution(char p) - This routine is called in response to a SET RESOLUTION command from the host. Set Resolution is a two byte command; the 2nd byte being the resolution itself. This routine is called after reception of the first byte, and so does nothing by itself

void SetScaling(void) - This routine is called in response to a SET SCALING command from the host. Scaling then changes to 2:1.

void ResetScaling(void) - This routine is called in response to a RESET SCALING command from the host. The scaling is then reset back to 1:1.

char GetByte(void) – This routine checks to see if the host is requesting to send data, and if so, it clocks in a data byte from the host. The function returns the received byte and implicitly sets the carry to 0 if the reception occurred without errors. *GetByte()* turns off the 1.024 millisecond interrupt during PS/2 clocking so as not to impose the potential for clock jitter due to that interrupt.

void ProcessCommand(char p) - This routine dispatches the received PS/2 command byte to the proper handler.

void SendMouseData(void) – This routine formats a mouse packet according to the PS/2 Mouse specification and sends it to the host.

void Put(char p) - *Put()* attempts to send a byte to the host. It is called to send an ACK, ERROR, or RESEND to the host. In accordance with the IBM specification, these responses are not held for retransmission in the event that the host requests to send a byte before the mouse can send the response itself. Thus, this routine waits for host inhibit to go away, then sends the byte. If the host requests-to-send before this happens, it simply returns without sending the byte

void ps2main(void) - *ps2main()* spins in an infinite loop waiting for an event that needs servicing.

void PutNextByte(void) - This routine sends the next byte of the message buffer to the host.

void ReInitialize(void) - This routine reinitializes PS/2 variables to their default states

void ResetInterval(void) - This routine resets the mouse report interval to the value last sent by the host. The report interval is counted down in the main loop to provide a time base for sending mouse data packets.

void Ps2MouseTask(void) - This routine handles ps2 mouse data packet management

void CheckZmouse(void) - This function is called whenever a set sample rate parameter is received. It maintains a sequence counter which advances each time a rate is received that matches the above sequence. If the sequence is completed, the z-wheel is enabled.

void ApplyResolution(void) - This routine scales the mouse output by right-shifting the mouse counts to achieve a /2 for each resolution factor.

void ApplyScaling(void) - This routine scales the mouse output according to the following to the PS/2 mouse specification, when scaling is enabled by the host.

send1 (assembler routine) – sends a PS/2 1 bit

send0 (assembler routine) – sends a PS/2 0 bit *getbit (assembler routine)* – receives a PS/2 bit from the host

SET RESOLUTION Command

The SET RESOLUTION command is conditionally enabled by the statement "#define ENABLE_RESOLUTION" at the beginning of the *combi.c* listing. On most systems this command is not supported. If you wish to disable this command in the firmware, comment out the aforementioned statement.

SET SCALING Command

The SET SCALING command is conditionally enabled by the statement "#define ENABLE_SCALING" at the beginning of the *combi.c* listing. On most systems this command is not supported. If you wish to disable this command in the firmware, comment out the aforementioned statement.

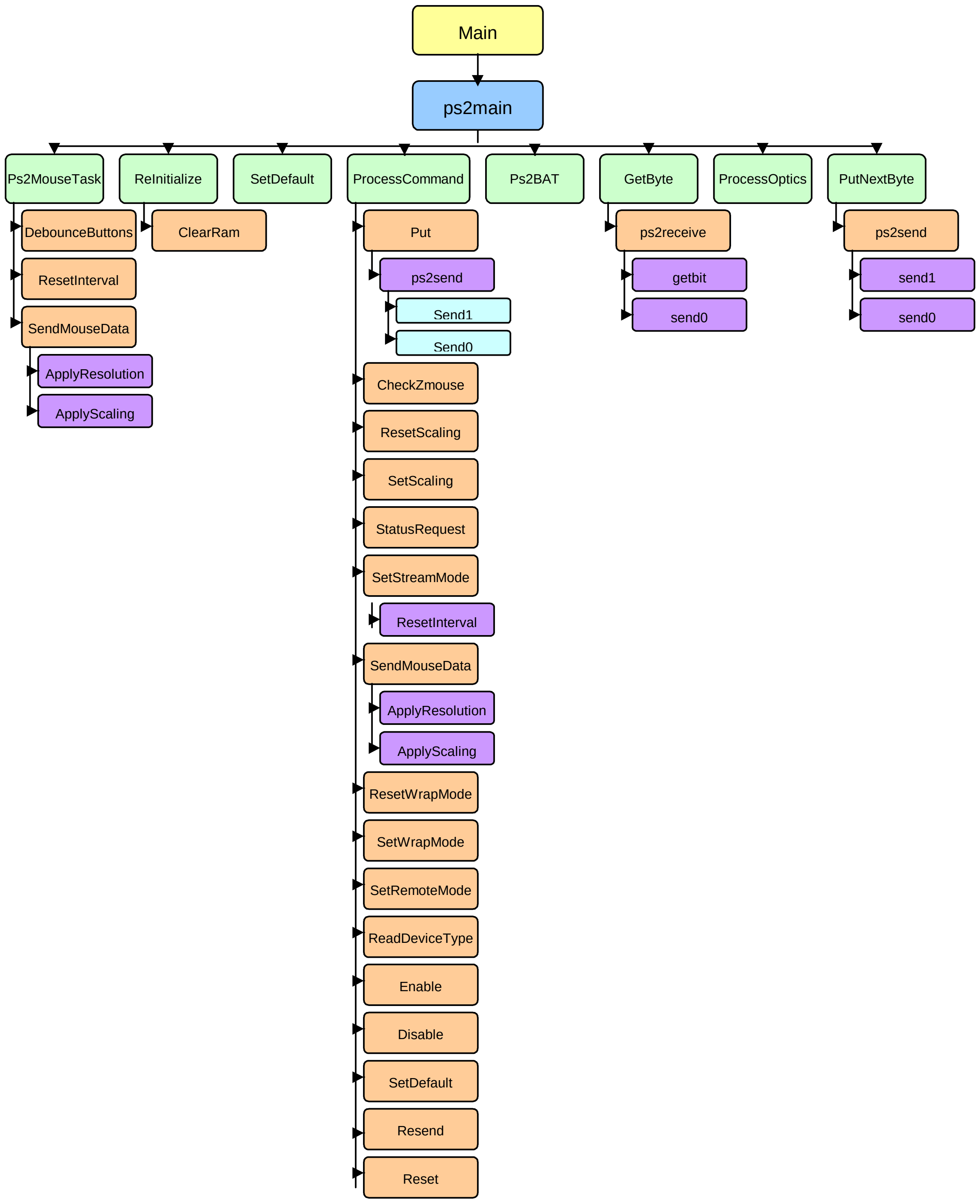


Figure 5.0 PS/2 Operation Function Call Map

Interrupt Jump Table

The interrupt jump table is defined in *chip.h*. This jump table must be defined in accordance with the documentation for the ByteCraft C-compiler. See section 10.3.10 in the ByteCraft C-compiler manual for more information about assigning interrupt vectors.

Interrupt Handlers

void MY_RESET_ISR(void) - This function initializes the data stack pointer, clears ram, initializes port I/O, enables the 1.024 millisecond interrupt, and jumps to main loop.

void USB_BUS_RESET_ISR(void) - The reset vector initializes the data stack pointer and program stack pointer, clears ram, makes a call to initialize system variables, and resets the USB address.

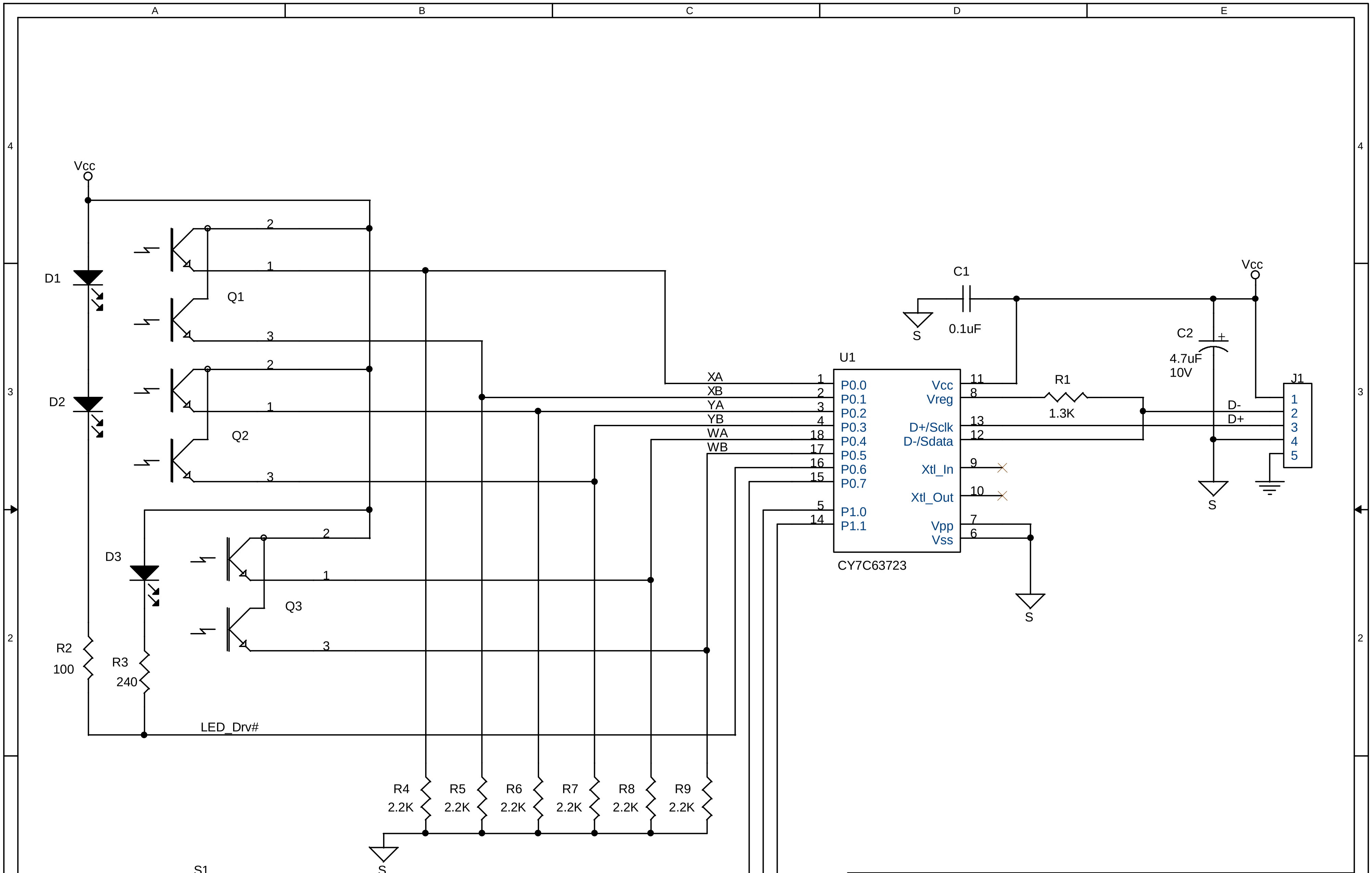
void MICROSECONDx128_ISR(void) - This ISR samples the mouse optics and places the sample in a queue for later processing by the main loop. This is done to minimize the execution time of this ISR, which can interrupted during the time PS/2 bits are being clocked to the host. The timing of the PS/2 clocking is critical (20 us margin), so the corresponding execution time of this ISR must be kept to less than 20 us.

void MILISECOND_ISR(void) - This ISR maintains the 1 millisecond counter variable, and sets a flag indicating a 1.024 millisecond interrupt has occurred. The main loop uses these variables for timing purposes.

void USB_A_EP0_ISR(void) - This ISR is entered upon receiving an endpoint 0 interrupt. Endpoint 0 interrupts occur during the Setup, data , and status phases of a control-read transaction. This ISR handler dispatches the proper routine to handle one of these phases. The interrupt will remain active until the phase of the transaction is complete.

void USB_A_EP1_ISR(void) - This ISR is entered upon receiving an endpoint 1 interrupt. If the ACK bit is sent, indicating that a valid mouse packet was just transmitted to the host, the SIE is set to NAK ins, and the data toggle bit is flipped for the next transaction.

void WAKEUP_ISR(void) - The wakeup ISR increments a global counter which is used to "tune" the frequency of the occurrence of the ISR itself (See routine TuneWakeup()).



CYPRESS SEMICONDUCTOR © 2000		
Title: USB/PS2 MOUSE REFERENCE DESIGN - Appendix A:CY7C63723		
Size: *A*	Document Number	Rev: 1.1
Date: *Friday, July 21, 2000*	Sheet *1* of *1*	

Appendix B: Bill of Materials for Components Shown on Schematic

Designator	Part Type
U1	Cypress CY7C63723 USB Microcontroller
D1	Infrared LED
D2	Infrared LED
D3	Infrared LED
Q1	Dual emitter output infrared phototransistor
Q2	Dual emitter output infrared phototransistor
Q3	Dual emitter output infrared phototransistor
S1	Momentary switch – normally open
S2	Momentary switch – normally open
S3	Momentary switch – normally open
C1	0.1uF ceramic capacitor, 10V or larger voltage rating
C2	4.7uF electrolytic capacitor, 10V or larger voltage rating
R1	1.3kΩ resistor, 5%*
R2	100Ω resistor
R3	240Ω resistor
R4	2.2kΩ resistor
R5	2.2kΩ resistor
R6	2.2kΩ resistor
R7	2.2kΩ resistor
R8	2.2kΩ resistor
R9	2.2kΩ resistor
J1	Mouse cable header

*** - tolerance specified by USB specification**

Appendix C. USB Descriptors

The following USB requests are supported. For details on these requests see the USB 1.1 and HID 1.0 specifications.

Request Name	Control Read	No-Data Control	Dev	Intf	Ep	Description
GET_STATUS	X		X	X	X	Return status for the specified recipient. The reply depends on recipient.
CLEAR_FEATURE		X	X		X	Clears or disables the specified feature.
SET_FEATURE		X	X		X	Sets or enables a EP stall and remote wakeup features
SET_ADDRESS		X	X			Sets the device address
GET_DESCRIPTOR	X		X			The returned data depends on the specified wValue. See table 9-4 in the USB 1.1 spec for a list of descriptor types. The entire table should be supported.
GET_CONFIGURATION	X		X			Returns the current device configuration
SET_CONFIGURATION		X	X			Sets the device configuration.
GET_INTERFACE	X			X		Returns the selected alternate setting for the specified interface.
SET_INTERFACE		X		X		Allows the host to select an alternate setting for the specified interface.
GET_REPORT	X			X		Allows the host to receive a report via the control pipe
GET_IDLE	X			X		Reads the current idle rate for the particular input report
GET_PROTOCOL	X			X		Reads which protocol is currently active (boot/report)
SET_REPORT		X		X		Allows the host to send a report to the device, possibly setting the state of input, output, or feature controls
SET_IDLE		X				Silences a particular report on the Interrupt In pipe until a new event occurs or the specified amount of time passes
SET_PROTOCOL		X		X		Switches between the boot protocol and the report protocol

Device Descriptor

Offset	Field	Size	Value	Description
0	Blength	1	0x12	Size of device descriptor
1	BdescriptorType	1	0x01	Device Descriptor Type
2	BcdUSB	2	0x0110	USB Specification Number
4	BdeviceClass	1	0x00	Not Supported
5	BdeviceSubClass	1	0x00	Not Supported
6	BdeviceProtocol	1	0x00	Protocol Depends on selected interface
7	BmaxPacketSize	1	0x08	Max Packet size on endpoint 0
8	IdVendor*1	2	0x04B4	Vendor ID – Cypress
10	IdProduct*2	2	0x6370	Product ID
12	BcdDevice	2	0x0100	Device version number
14	Imanufacturer	1	0x01	Index of manufacturer string descriptor
15	Iproduct	1	0x02	Index of Product string descriptor
16	IserialNumber	1	0x00	Not supported
17	BNumConfigurations	1	0x01	Number of configurations

Configuration Descriptor

Offset	Field	Size	Value	Description
0	Blength	1	0x09	Size of configuration descriptor
1	BdescriptorType	1	0x02	Configuration Descriptor Type
2	WtotalLength	2	0x0022	Total size of configuration descriptor including interface, hid and endpoint descriptors
4	BnumInterfaces	1	0x01	One interface
5	BconfigurationValue	1	0x01	Used by SET CONFIGURATION to select this configuration
6	Iconfiguration	1	0x04	USB HID Compliant Mouse
7	BmAttributes	1	0xA0	Bus powered and able to perform remote wakeup
8	MaxPower[*3]	1	0x19	Bus power draw is about 50mA

Interface Descriptor

Offset	Field	Size	Value	Description
0	Blength	1	0x09	Size of interface descriptor
1	BdescriptorType	1	0x04	Interface Descriptor Type
2	BinterfaceNumber	1	0x00	Interface Number 0
3	BalternateSetting	1	0x00	Alternate Setting 0
4	BnumEndpoints	1	0x01	1 Endpoint
5	BinterfaceClass	1	0x00	Interface Class assigned by USB spec
6	BinterfaceSubClass	1	0x01	Interface Sub Class assigned by USB spec
7	BinterfaceProtocol	1	0x02	Interface Protocol assigned by USB spec
8	Iinterface	1	0x00	Not supported

HID Descriptor

Offset	Field	Size	Value	Description
0	Blength	1	0x09	Size of HID descriptor
1	BdescriptorType	1	0x21	HID descriptor type
2	BcdHID	2	0x0110	Version of HID spec 1.1
4	BcountryCode	1	0x00	Country Code USA
5	BnumDescriptors	1	0x01	Number of report descriptors
6	BdescriptorType	1	0x22	Descriptor type
7	WdescriptorLength	2	0x0048	HID report Descriptor length

Endpoint Descriptor

Offset	Field	Size	Value	Description
0	Blength	1	0x07	Size of Endpoint descriptor
1	BdescriptorType	1	0x05	Endpoint descriptor type
2	BendpointAddress	1	0x81	IN endpoint on endpoint 1
3	BmAttributes	1	0x03	Interrupt endpoint
4	WmaxPacketSize	2	0x0004	Maximum packet size 4 bytes
6	Binterval	1	0x0A	Polling interval of 10ms

HID Report Descriptor

Offset	Field	Size	Value	Description
0	Usage page	2	0x05, 0x01	Generic Desktop Controls
2	Usage	2	0x09, 0x02	Mouse
4	Collection	2	0xA1, 0x01	Application
6	Usage page	2	0x05, 0x09	Buttons
8	Usage minimum	2	0x19, 0x01	1 bits for button data
10	Usage maximum	2	0x29, 0x03	3 bits for button data
12	Logical minimum	2	0x15, 0x00	0 – button open
14	Logical maximum	2	0x25, 0x01	1 – button closed
16	Report count	2	0x95, 0x03	3 reports, first byte of report
18	Report size	2	0x75, 0x01	Each button report is 1 bit, left = bit0, right = bit1, middle = bit2
20	Input	2	0x81, 0x02	Variable Data Bit Field with Absolute position
22	Report count	2	0x95, 0x01	1 report
24	Report size	2	0x75, 0x05	5 report bits for padding
26	Input	2	0x81, 0x03	Constant Variable Bit Field with Absolute position
28	Usage page	2	0x05, 0x01	Generic Desktop
30	Usage	2	0x09, 0x01	Pointer
32	Collection	2	0xA1, 0x00	Linked
34	Usage	2	0x09, 0x30	X
36	Usage	2	0x09, 0x31	Y
38	Logical minimum	2	0x15, 0x81	-127
40	Logical maximum	2	0x25, 0x7F	127
42	Report size	2	0x75, 0x08	The x and y reports are 8 bits
44	Report count	2	0x95, 0x02	2 reports, x is byte 1, y is byte 2
46	Input	2	0x81, 0x06	Variable Data Bit Field with Relative position
48	End collection	1	0xC0	End collection
49	Usage	2	0x09, 0x38	Wheel
51	Report size	2	0x95, 0x01	Wheel data size is 1 byte
53	Input	2	0x81, 0x06	Variable Data Bit Field with Relative position
55	Usage	2	0x09, 0x3C	Motion wakeup
57	Logical Minimum	2	0x15, 0x00	0 no movement
59	Logical Maximum	2	0x25, 0x01	1 movement
61	Report size	2	0x75, 0x01	Wheel report is 1 bit for movement, bit 0 of byte 3
63	Report Count	2	0x95, 0x01	1 report
65	Feature (Data, Ary, Abs)	2	0xB1, 0x22	Variable Data Bit Field with absolute positioning and no preferred state
67	Report Count	2	0x95, 0x07	7 reports for reversing, upper 7 bits of byte 3
69	Feature (Cnst, Ary, Abs)	2	0xB1, 0x01	Constant Array Bit Field with absolute positioning
71	End collection	1	0xC0	End collection

Language Descriptor

Offset	Field	Size	Value	Description
0	Blength	1	0x04	Language descriptor length
1	BdescriptorType	1	0x03	Language Descriptor Type
2	Language	1	0x09	English
3	Sub-Language	1	0x04	US

Manufacturer String[*4]

A request for the manufacturer string will return the following string.

"Cypress Semiconductor"

Product String[*5]

A request for the product string will return the following string.

"Cypress CY7C637xx USB Mouse v?.??"

Configuration String

A request for the configuration string will return the following string.

"HID Mouse"

Endpoint 1 String

A request for the endpoint string will return the following string.

"Endpoint 1 Interrupt Pipe"

Note 1: idVendor should be changed to the value supplied to you by the USBIF
Note 2: idProduct should be assigned by you for your specific product.
Note 3: MaxPower value should be changed as per your specific circuit's current draw.
Note 4: The Manufacturer String should be changed to the name of your company.
Note 5: The Product String should be changed to your product's name.

Appendix D: USB data reporting format

The USB report has two formats, depending on if boot or report protocol is enabled. The following format is the boot protocol and is understood by a USB aware BIOS.

	Bit 7							**Bit 0**
Byte 0	0	0	0	0	0	Middle	Right	Left
Byte 1	X	X	X	X	X	X	X	X
Byte 2	Y	Y	Y	Y	Y	Y	Y	Y

The following is the USB report protocol format and allows the additional wheel movement information in the fourth byte. When the wheel is moved forward the fourth byte reports a 0x01, and when moved backward the fourth byte reports 0xFF. When the wheel is idle, then this byte is assigned 0x00.

	Bit 7							**Bit 0**
Byte 0	0	0	0	0	0	Middle	Right	Left
Byte 1	X	X	X	X	X	X	X	X
Byte 2	Y	Y	Y	Y	Y	Y	Y	Y
Byte 3	R	R	R	R	R	R	R	F/R

Appendix E: PS/2 reporting format

The PS/2 portion of the firmware handles the following requests and commands listed in the table below.

Hex Code	Command	Action
0xFF	Reset	Resets mouse to default states
0xFE	Resend	Resends last data to host
0xF6	Set Default	Sets mouse to use default parameters
0xF5	Disable	Disables the mouse
0xF4	Enable	Enables the mouse
0xF3	Set Sampling Rate	Set sampling rate to 10,20,40,60,80,100,200/second
0xF2	Read Device Type	Returns 0x00 to host, indicating the device is a mouse
0xF0	Set Remote Mode	Sets remote mode so data values are only reported after a read data command
0xEE	Set Wrap Mode	Set wrap mode until 0xFF or 0xEC is received
0xEC	Reset Wrap Mode	Reset to previous mode of operation.
0xEB	Read Data	Responds by sending a mouse report packet to host
0xEA	Set Stream Mode	Sets stream mode
0xE9	Status Request	Returns current mode, en/disabled, scaling, button, resolution, and sampling rate information to the host.
0xE8	Set Resolution	Sets resolution to 1,2,4,8 counts/mm
0xE7	Set Scaling 2:1	Sets scaling to 2:1
0xE6	Reset Scaling	Resets scaling to 1:1
0xAA	Completion Code	Command completion code
0xFA	Peripheral ACK	Sent to acknowledge host requests

The PS/2 specification calls out the following default mouse report format. Byte 0 is the button data (1=pressed, 0=released), X and Y optics sign bits, and X and Y overflow bits. Byte 1 is the X optics data in 2's complement format. Byte 2 has the Y optics data in 2's complement format. At reset or power-on the standard PS/2 reporting format is enabled.

	Bit 7							Bit 0
Byte 0	Y overflow	X overflow	Y sign	X sign	Reserved 0	Reserved 0	Right button	Left button
Byte 1	X	X	X	X	X	X	X	X
Byte 2	Y	Y	Y	Y	Y	Y	Y	Y

After the following sequence of commands, the wheel report format is enabled.

0xF3, 0xC8	Set Sampling Rate 200 per second
0xF3, 0x64	Set Sampling Rate 100 per second
0xF3, 0x80	Set Sampling Rate 50 per second
0xF2, 0x03	Read Device Type returns a value of 0x03

After the Read Device Type command returns 0x03 to indicate that this is a Microsoft compatible three button-wheel mouse, the wheel report format is enabled. After this initialization sequence, the PS/2 wheel reporting format is enabled. The fourth byte represents the wheel data. This byte is assigned 0x01 for forward wheel movement and 0xFF for backward wheel movement. When the wheel is idle, this value is 0x00.

	Bit 7							**Bit 0**
Byte 0	Y overflow	X overflow	Y sign	X sign	Always 1	Middle Button	Right button	Left button
Byte 1	X	X	X	X	X	X	X	X
Byte 2	Y	Y	Y	Y	Y	Y	Y	Y
Byte 3	Wheel*	Wheel*	Wheel*	Wheel*	Wheel*	Wheel*	Wheel*	Wheel*

The PS2 data transmission according to the PS/2 Hardware Interface Technical Reference including eleven bits for each byte sent. The bits are sent in the following order with data valid on the falling edge of the clock. See the PS/2 Hardware Interface Technical Reference manual for timing information.

Start Bit (Always 0)	Data Bit 0	Data Bit 1	Data Bit 2	Data Bit 3	Data Bit 4	Data Bit 5	Data Bit 6	Data Bit 7	Odd Parity Bit	Stop Bit (Always 1)