# Designing a Low-Cost USB Mouse with the Cypress Semiconductor CY7C63000 USB Controller

## Introduction

The Universal Serial Bus (USB) is an industrial standard serial interface between a computer and peripherals such as a mouse, joystick, keyboard, etc. This application note describes how a cost-effective USB opto-mechanical mouse can be built quickly using the Cypress Semiconductor single-chip CY7C63000 USB controller. The document starts with the basic operations of an opto-mechanical mouse followed by an introduction to the CY7C63000 USB controller. A schematic of the USB mouse and 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 mouse function. Several sample code segments are included to assist in the explanation. The binary code of the complete mouse firmware is available free of charge from Cypress Semiconductor. Please contact your local Cypress sales office for details.

This application note assumes that the reader is familiar with the CY7C63000 USB controller and the Universal Serial Bus. The CY7C63000 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.

## USB Mouse Basics

USB has been gaining popularity due to it's simple connection, plug and play feature, and hot insertion capability. There are several kinds of USB pointing devices available in the market. The opto-mechanical mouse is the most popular type because it provides relatively high resolution and works on a wide range of surfaces.

Basically, an opto-mechanical mouse has a rubber track ball that is coupled to two roll bars as shown in *Figure 1*. The "stabilizer" is a roller that provides the third contact point for the mouse ball.

One roll bar keeps track of the X-axis movement while the other one keeps track of the Y-axis movement. There is a slotted wheel at one end of each roll bar. An LED is installed on one side of the wheel with two photo transistors positioned on the other side as shown in *Figure 2*.

The photo-transistor outputs allow the mouse to detect wheel motion and determine the motion direction. For example, from the starting position shown, wheel motion to the left would look like *Figure 3*.

From the starting position shown, slotted wheel motion to the right would look like *Figure 4*.

From the outputs of the photo-transistors, the mouse chip determines the direction and calculates the distance when the mouse is moved.

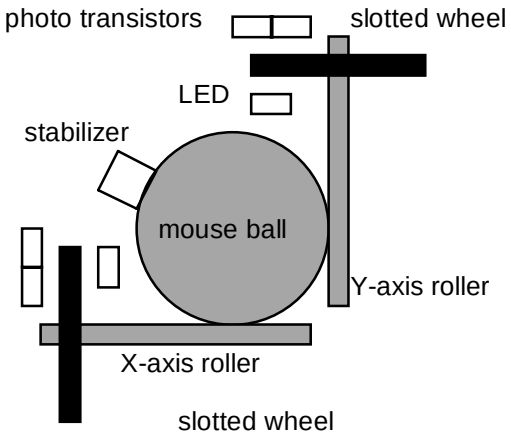


**Figure 1. Mechanical Hardware**

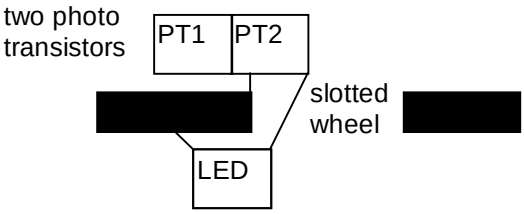


**Figure 2. Opto-Mechanical Detail**

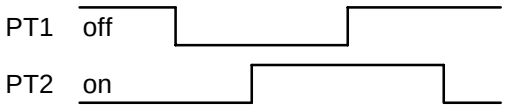


**Figure 3. Slotted Wheel Moves Left**

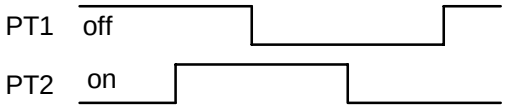


**Figure 4. Slotted Wheel Moves Right**

The resolution is the smallest motion the mouse can detect, measured in dots per inch (DPI). A typical opto-mechanical mouse has a resolution in the 200 to 400 DPI range. The mechanical dimensions of the mouse hardware limit the maximum achievable resolution.

USB provides the plug-and-play feature that is not supported in RS-232 and PS/2 interfaces. The USB interface uses a four-pin connector with positive retention. A 28 AWG twisted pair is used for differential signaling and two 20 to 30 AWG wires are used to supply power and ground. No cable shielding is necessary for a mouse application.

## Introduction to CY7C63000

The CY7C63000 is a high performance 8-bit RISC microcontroller with an integrated USB Serial Interface Engine (SIE). The architecture implements 34 commands that are optimized for USB applications. The CY7C63000 has built-in clock oscillator and timers as well as programmable current drivers, and pull-up resistors at each I/O line. High performance, low-cost human-interface type computer peripherals such as mouse, joystick, and gamepad can be implemented with minimum external components and firmware effort.

### Clock Circuit

The CY7C63000 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 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 CY7C63000 has 12 general purpose I/O lines divided into 2 ports: Port 0 and Port 1. One such I/O circuit is shown in *Figure 5*. The output state can be programmed according to *Table 1* below**.** Writing a “0” to the Data Register will drive the output Low and allow it to sink current.

**Table 1. Programmable Output State**

| Port Data bit | Port Pull-up bit | Output State |
|---|---|---|
| 0 | X | sink current “0” |
| 1 | 0 | pull-up resistor “1” |
| 1 | 1 | High-Z |

Instead of supporting a fixed output drive, the CY7C63000 allows the user to select an output current level for each I/O line. The sink current of each output is controlled by a dedicated 8-bit Isink Register. The lower 4-bits of this register contains a code selecting one of sixteen sink current levels. The upper 4-bits are reserved and must be written as zeros. The output sink current levels of the two I/O ports are different. For Port 0 outputs, the lowest drive strength (0000) is about 0.2 mA and the highest drive strength (1111) is about 1.0 mA. These levels are insufficient to drive the LEDs in a mouse.

Port 1 outputs are specially designed to drive high-current applications such as LEDs. Each Port 1 output is much stronger than their Port 0 counterparts at the same drive level setting. In other words, the lowest and highest drive for Port 1 lines are 3.2 mA and 16 mA respectively.

Each General Purpose I/O (GPIO) is capable of generating an interrupt to the RISC core. Interrupt polarity is selectable on a per bit basis using the Port Pull-up register. Setting a Port Pull-up register bit to “1” will select a rising edge trigger for the corresponding GPIO line. Conversely, setting a Port Pull-up Register bit to “0” will select a falling edge trigger. The interrupt triggered by a GPIO line is individually enabled by a dedicated bit in the Port Interrupt Enable Registers. All GPIO interrupts are further masked by the Global GPIO Interrupt Enable Bit in the Global Interrupt Enable Register.

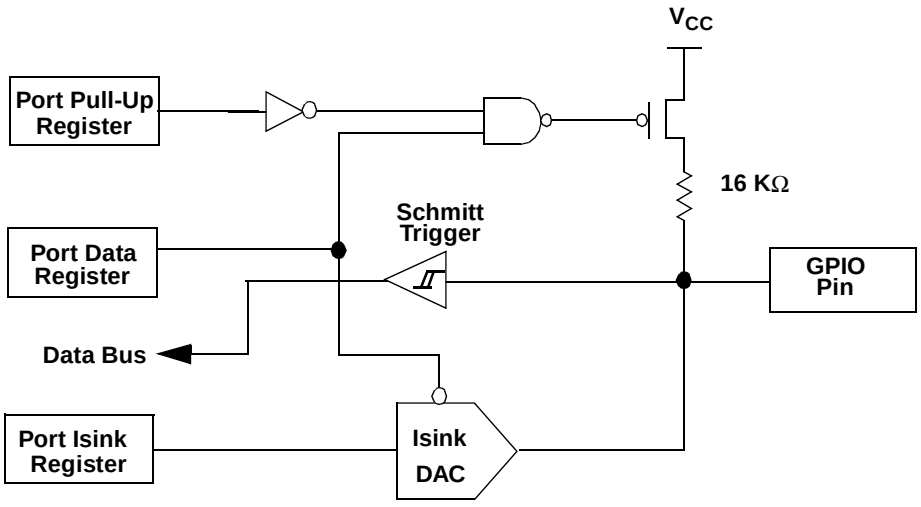


**Figure 5. One General Purpose I/O Line**

The Port Pull-up Registers are located at I/O address 0x08 and 0x09 for Port 0 and Port 1 respectively. The Data Registers are located at I/O address 0x00 and 0x01 for Port 0 and Port 1 respectively. The Port 0 and Port 1 Interrupt Enable Registers are at addresses 0x04 and 0x05 respectively.

**Wake-Up Interrupt**

Power management is paramount in many USB applications. To conserve power, the CY7C63000 supports an externally programmable interrupt input to wake up the microcontroller from the suspend mode when the mouse is moved or when a button is pressed. The suspend mode causes the microcontroller to shut down most of its functions such as the clock circuit, the RISC core, the timer, and part of the SIE. In the mouse application, a high percentage of the power is consumed by the LEDs. Therefore, the CY7C63000 should be programmed to turn off the LEDs before entering the suspend mode. With the LEDs off, the CY7C63000 can no longer detect any mouse movements although button closures are still recognized (because pressing a button causes an interrupt). This problem can be solved by using the wake-up interrupt that wakes up the microcontroller, checks for mouse movement, and then goes back to suspend mode.

The wake-up interrupt can be implemented by connecting the CEXT pin to VCC with a resistor and to GND with a capacitor. Before the firmware puts the microcontroller into the suspend mode, it writes a zero to the Cext register at address 0x22 to discharge the external capacitor. Then, to start timing a one is written to the Cext register to allow the RC circuit to begin charging. A wake-up interrupt is generated to the RISC core when the external capacitor is charged up to nominal 2.75V (45% to 65% of Vcc) by the external resistor. The duration between successive wake-ups is controlled by the RC constant of the external resistor and capacitor.

## Hardware Implementation

*Figure 6* is the schematic for a mouse application.

Photo transistor pins of Port 0 are programmed by writing a zero to the Data Registers which drives the output low. Then set the value of the Port Isink Register to the sink current value. One of sixteen sink current values could be selected. This is done to bias the photo transistors for correct operation.

Button pins of Port 0 are programmed to accept active-low inputs with internal pull-up resistors enabled. This is accomplished by setting all bits in the Port 0 Data Register to “1” and setting the contents of the Port 0 Pull-up Register to all “0”s.

Bits 4 to 6 of Port 0 are connected to the left, right, and middle buttons respectively. Bits 0 and 1 are connected to the left and right photo transistors of the horizontal axis respectively. Bits 2 and 3 are connected to left and right photo transistors of the vertical axis respectively.

The two LEDs are connected in series to bit 0 of Port 1. The LEDs are turned off in the suspend mode to conserve power. The LEDs are switched on only when the mouse wakes up. Because the sink current of each GPIO line can be set to one of sixteen levels, the user can adjust the light output of the LEDs to match the sensitivity of a wide range of photo transistors.

The CEXT pin of the CY7C63000 is connected to an external RC timing circuit formed by R2 and C1. The wake-up time is set to about 20 msec to achieve a good balance between wake-up response time and power savings.

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 1.5K ohms pull-up terminator. The CY7C63000 eliminates the need for a 3.3V regulator by specifying a 7.5 Kohm resistor connected between the USB D– line and the nominal 5V Vcc.

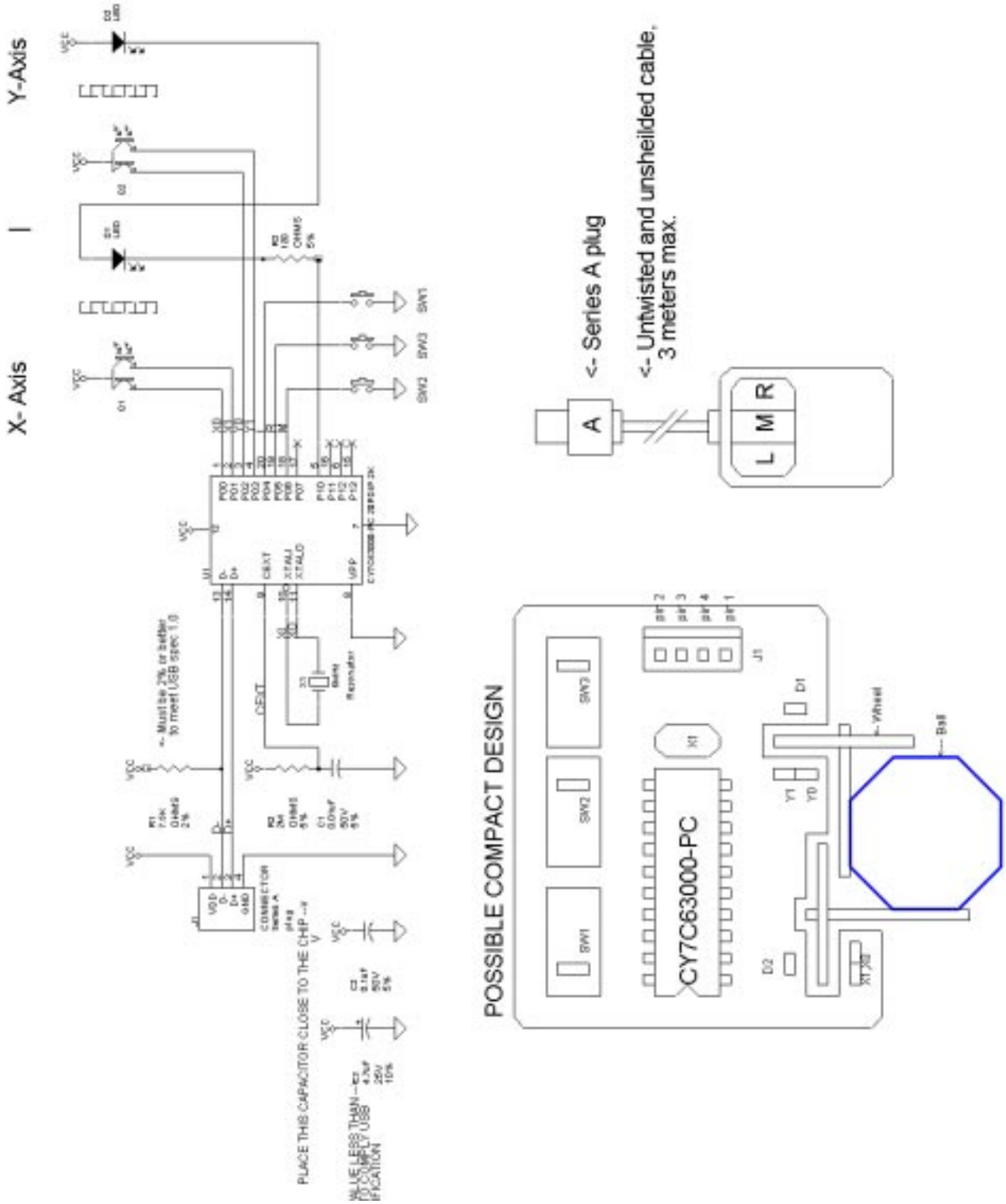


**Figure 6. Hardware Implementation**

## Firmware Implementation

### USB Interface

All USB Human Interface Device (HID) class applications such as a mouse, follow the same USB start-up procedure. The procedure is as follows (see *Figure 7*):

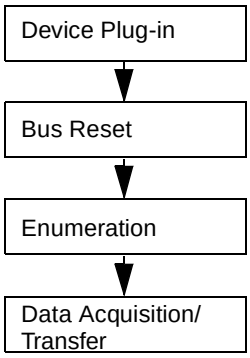


**Figure 7. USB Start-Up Procedure**

*Device Plug-in*

When a USB device is first connected to the bus, it is powered but remains non-functional waiting for a bus reset. The pull-up resistor on D– notifies the hub that a low-speed (1.5 Mbps) device has just been connected.

*Bus Reset*

The host recognizes the presence of a new USB device and resets it (see *Figure 8*).

*Enumeration*

The host sends a SETUP packet followed by IN packets to read the device description from default address 0. When the description is received, the host assigns a new USB address to the device. The device 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 device, the host now knows the number of data endpoints supported by the device (in a USB mouse, there is only one data endpoint). At this point, the process of enumeration is completed. See *Figures 9, 10* and *11*.

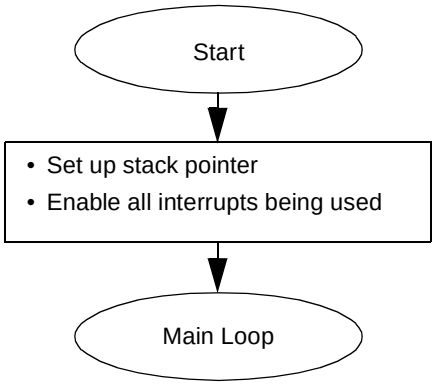


**Figure 8. Reset Interrupt Service Routine**

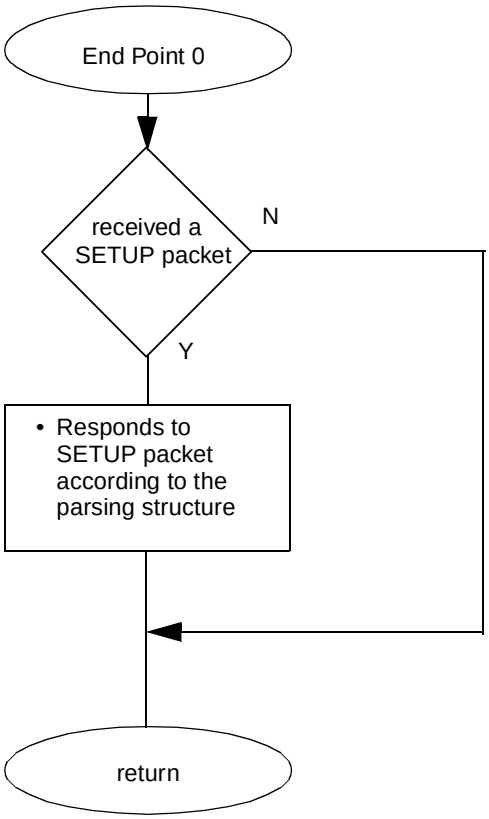


**Figure 9. Endpoint 0 ISR**

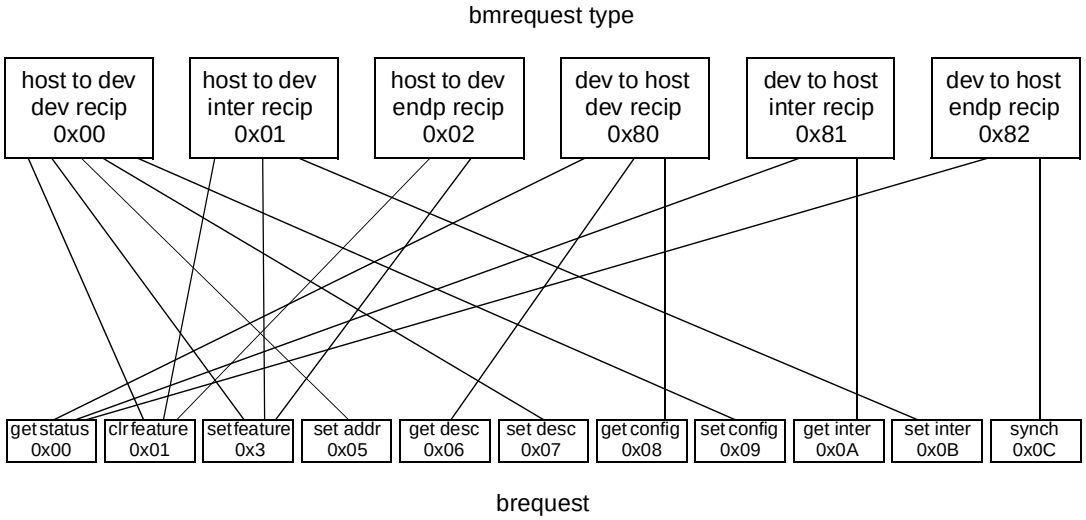


**Figure 10. USB Standard Request Parsing Structure**

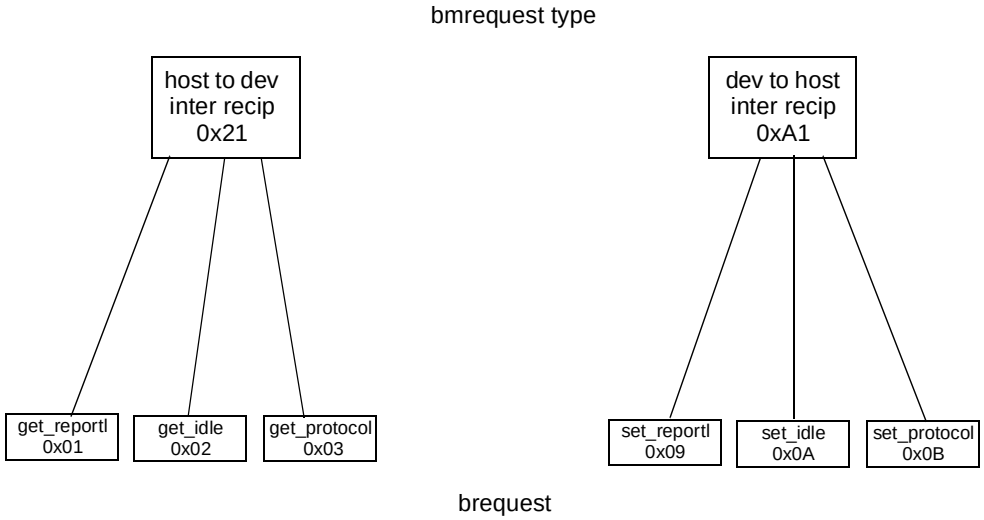


**Figure 11. USB HID Class Request Parsing Structure**

*Data Acquisition/Transfer*

The firmware polls the mouse buttons and the photo transistors. The status of the buttons as well as the horizontal and vertical displacements are sent to the host using endpoint 1. When the host issues IN packets to retrieve data from the device, the device returns three bytes of data as shown in *Figure 12. Figure 13 illustrates response to an e14,* and *15.*)

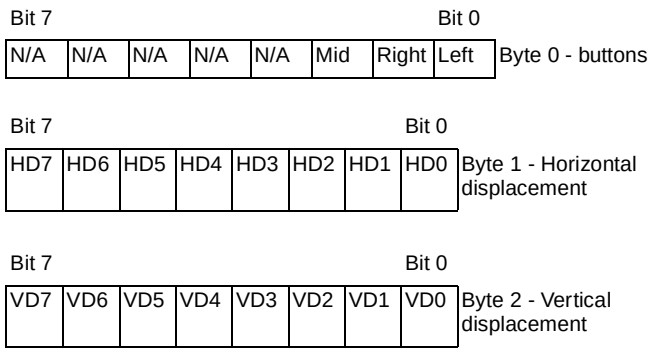


**Figure 12. Data Organization for USB Mouse**

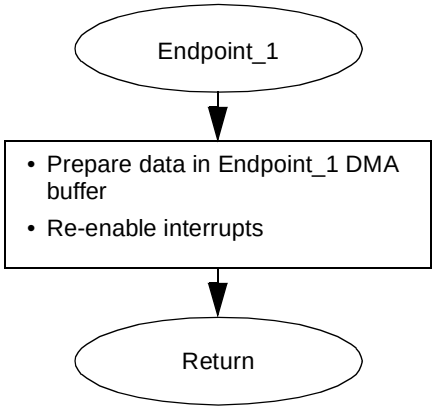


**Figure 13. Endpoint 1 Interrupt Service Routine**

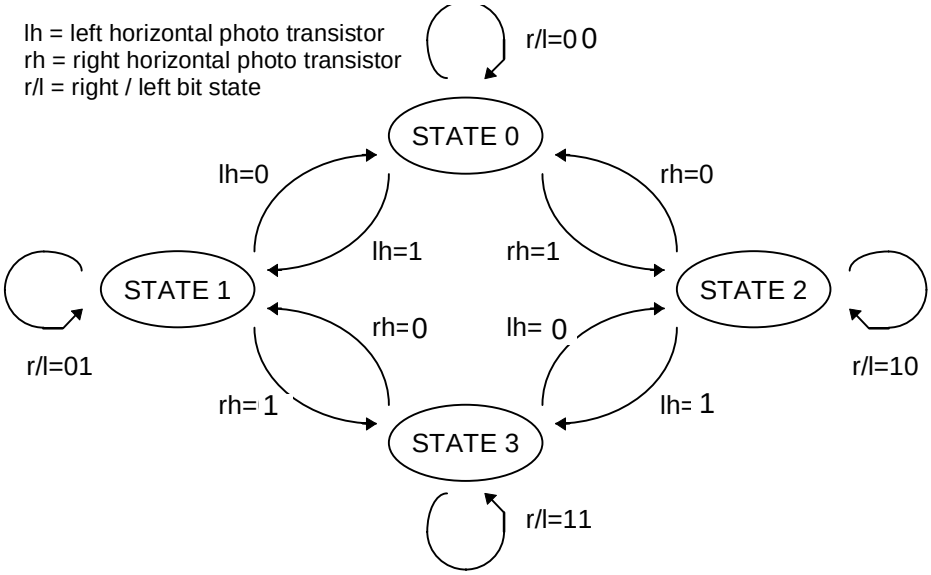


**Figure 14. Mouse State Diagram**

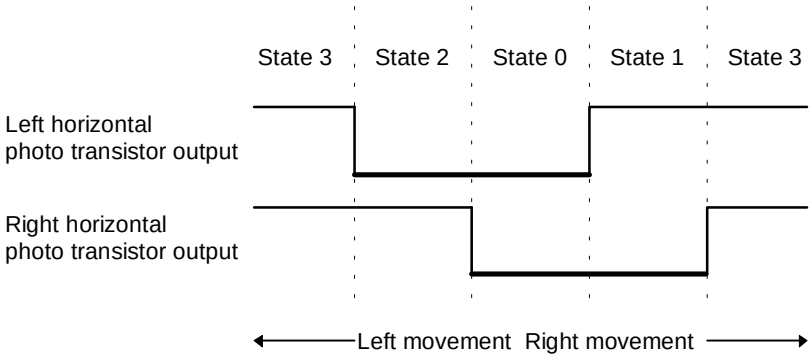


**Figure 15. State Definitions**

The byte order and bit field positions are defined by the USB HID specification.

**USB Descriptors**

As stated earlier, the USB descriptors hold information about the device. There are several types of descriptors, which will be discussed in detail below. All descriptors have certain characteristics in common. Byte 0 is always the descriptor length in bytes and Byte 1 is always the descriptor type. Discussion of these two bytes will be omitted from the following descriptions. The rest of the descriptor structure is dependent on the descriptor type. An example of each descriptor will be given. Descriptor types are device, configuration, interface, endpoint, string, report, and several different class descriptors.

*Device Descriptor*

This is the first descriptor the host requests from the device. It contains important information about the device. The size of this descriptor is 18 bytes. A list follows:

- USB Specification release number in binary-coded decimal (BCD) (2 bytes)
- Device class (1 byte)
- Device subclass (1 byte)
- Device protocol (1 byte)
- Max packet size for Endpoint 0 (1 byte)
- Vendor ID (2 bytes)
- Product ID (2 bytes)
- Device release number in BCD (2 bytes)
- Index of string describing Manufacturer (Optional) (1 byte)
- Index of string describing Product (Optional) (1 byte)
- Index of string containing serial number (Optional) (1 byte)
- Number of configurations for the device (1 byte)

```
Example of a device descriptor

Descriptor Length (18 bytes)
Descriptor Type (Device)
Complies to USB Spec Release (1.00)
Class Code (insert code)
Subclass Code (0)
Protocol (No specific protocol)
Max Packet Size for endpt 0 (8 bytes)
Vendor ID (Cypress)
Product ID (USB Joystick)
Device Release Number (1.03)
String Describing Vendor (None)
String Describing Product (None)
String for Serial Number (None)
Possible Configurations (1)
```

*Configuration Descriptor*

The configuration descriptor is 9 bytes in length and gives the configuration information for the device. It is possible to have more than one configuration for each device. When the host requests a configuration descriptor, it will continue to read these descriptors until all configurations have been received. A list of the structure follows:

- Total length of the data returned for this configuration (2 bytes)
- Number of interfaces for this configuration (1 byte)
- Value used to address this configuration (1 byte)
- Index of string describing this configuration (Optional) (1 byte)
- Attributes bitmap describing configuration characteristics (1 byte)
- Maximum power the device will consume from the bus (1 byte)

```
Example of configuration descriptor

Descriptor Length (9 bytes)
Descriptor Type (Configuration)
Total Data Length (34 bytes)
Interfaces Supported (1)
Configuration Value (1)
String Describing this Config (None)
Config Attributes (Bus powered)
Max Bus Power Consumption (100mA)
```

*Interface Descriptor*

The interface descriptor is 9 bytes long and describes the interface of each device. It is possible to have more than one interface for each device. This descriptor is set up as follows:

- Number of this interface (1 byte)
- Value used to select alternate setting for this interface (1 byte)
- Number of endpoints used by this interface. If this number is zero, only endpoint 0 is used by this interface (1 byte)
- Class code (1 byte)
- Subclass code (1 byte)
- Protocol code (1 byte)
- Index of string describing this interface (1 byte)

```
Example of interface descriptor

Descriptor Length (9 bytes)
Descriptor Type (Interface)
Interface Number (0)
Alternate Setting (0)
Number of Endpoints (1)
Class Code (insert code)
Subclass Code (0)
Protocol (No specific protocol)
String Describing Interface (None)
```

*Endpoint Descriptor*

The endpoint descriptor describes each endpoint, including the attributes and the address of each endpoint. It is possible to have more than one endpoint for each interface. This descriptor is 7 bytes long and is set up as follows:

- Endpoint address (1 byte)
- Endpoint attributes. Describes transfer type (1 byte)
- Maximum packet size this endpoint is capable of transferring (2 bytes)
- Time interval at which this endpoint will be polled for data (1 byte)

```
Example of endpoint descriptor

Descriptor Length (7 bytes)
Descriptor Type (Endpoint)
```

```
Endpoint Address (IN, Endpoint 1)
Attributes (Interrupt)
Maximum Packet Size (6 bytes)
Polling Interval (10 ms)
```

*HID (Class) Descriptor*

The class descriptor tells the host about the class of the device. In this case, the device falls in the human interface device (HID) class. This descriptor is 9 bytes in length and is set up as follows:

- Class release number in BCD (2 bytes)
- Localized country code (1 byte)
- Number of HID class descriptor to follow (1 byte)
- Report descriptor type (1 byte)
- Total length of report descriptor in bytes (2 bytes)

```
Example of HID class descriptor

Descriptor Length (9 bytes)
Descriptor Type (HID Class)
HID Class Release Number (1.00)
Localized Country Code (USA)
Number of Descriptors (1)
Report Descriptor Type (HID)
Report Descriptor Length (63 bytes)
```

*Report Descriptor*

This is the most complicated descriptor in USB. There is no set structure. It is more like a computer language that describes the format of the device's data in detail. This descriptor is used to define the structure of the data returned to the host as well as to tell the host what to do with that data. An example of a report descriptor can be found below.

A report descriptor must contain the following items: Input (or Output or Feature), Usage, Usage Page, Logical Minimum, Logical Maximum, Report size, and Report Count. These are all necessary to describe the device's data.

```
Example of report descriptor

Usage Page (Generic Desktop)
Usage (Mouse)
Collection (Application)
    Usage (Pointer)
    Collection (Physical)

         Usage Page (Buttons)
         Usage Minimum (01)
         Usage Maximum (03)
         Logical Minimum (0)
         Logical Maximum (1)
         Report Count (3)
         Report Size (1)
         Input (Data, Variable,
          Absolute)
         Report Count (1)
         Report Size (5)
         Input (Constant)
         Usage Page (Generic Desk-
top)
         Usage (X)
         Usage (Y)
         Logical Minimum (-127)
         Logical Maximum (127)
         Report Size (8)
         Report Count (2)
         Input (Data, Variable,
          Variable)
    End Collection
End Collection
```

Input items are used to tell the host what type of data will be returned as input to the host for interpretation. These items describe attributes such as data vs. constant, variable vs. array, absolute vs. relative, etc.

Usages are the part of the descriptor that defines what should be done with the data that is returned to the host. From the example descriptor, Usage (X) tells the host that the data is to be used as an X axis input. There is also another kind of Usage tag found in the example called a Usage Page. The reason for the Usage Page is that it is necessary to allow for more than 256 possible Usage tags. Usage Page tags are used as a second byte which allows for up to 65536 Usages.

Logical Minimum and Logical Maximum are used to bound the values that a device will return.

Report Size and Report Count define the structures that the data will be transferred in. Report Size gives the size of the structure in bits. Report Count defines how many structures will be used. In the example descriptor above, the lines Report Size (8) and Report Count (2) define the axes of the mouse. There are now two eight-bit fields defined, one for the X axis and one for the Y axis.

Collection items are used to show a relationship between two or more sets of data. End Collection items simply close the collection.

It is important to note that all examples given here are merely for clarification. They are not necessarily definitive solutions.

A more detailed description of all items discussed here as well as other descriptor issues can be found in the "Device Class Definition for Human Interface Devices (HID)" revision 1.0d and in the "Universal Serial Bus Specification" revision 1.0, chapter 9. Both of these documents can be found on the USB world wide web site at http://www.usb.org/.

**Power Management**

Power management on USB devices involves the issues described in *Figures 16* and *17*. The LEDs are turned off before the device goes into suspend and are turned on right after the device gets out of suspend.

**Displacement Calculation**

The outputs of the photo transistors for one axis transition through the states shown in *Figure 14*. A transition from one state to the next indicates mouse movement in that direction. Based on the position of the photo transistors, a counter-clockwise state change increments the mouse position counter and a clockwise state change decrements the position counter. The displacements are calculated based on the previous location of the mouse.

## Conclusion

The two main enabling factors of the proliferation of the USB devices are cost and functionality. The CY7C63000 meets both requirements by integrating the USB SIE and multi-function I/Os with a USB optimized RISC core.

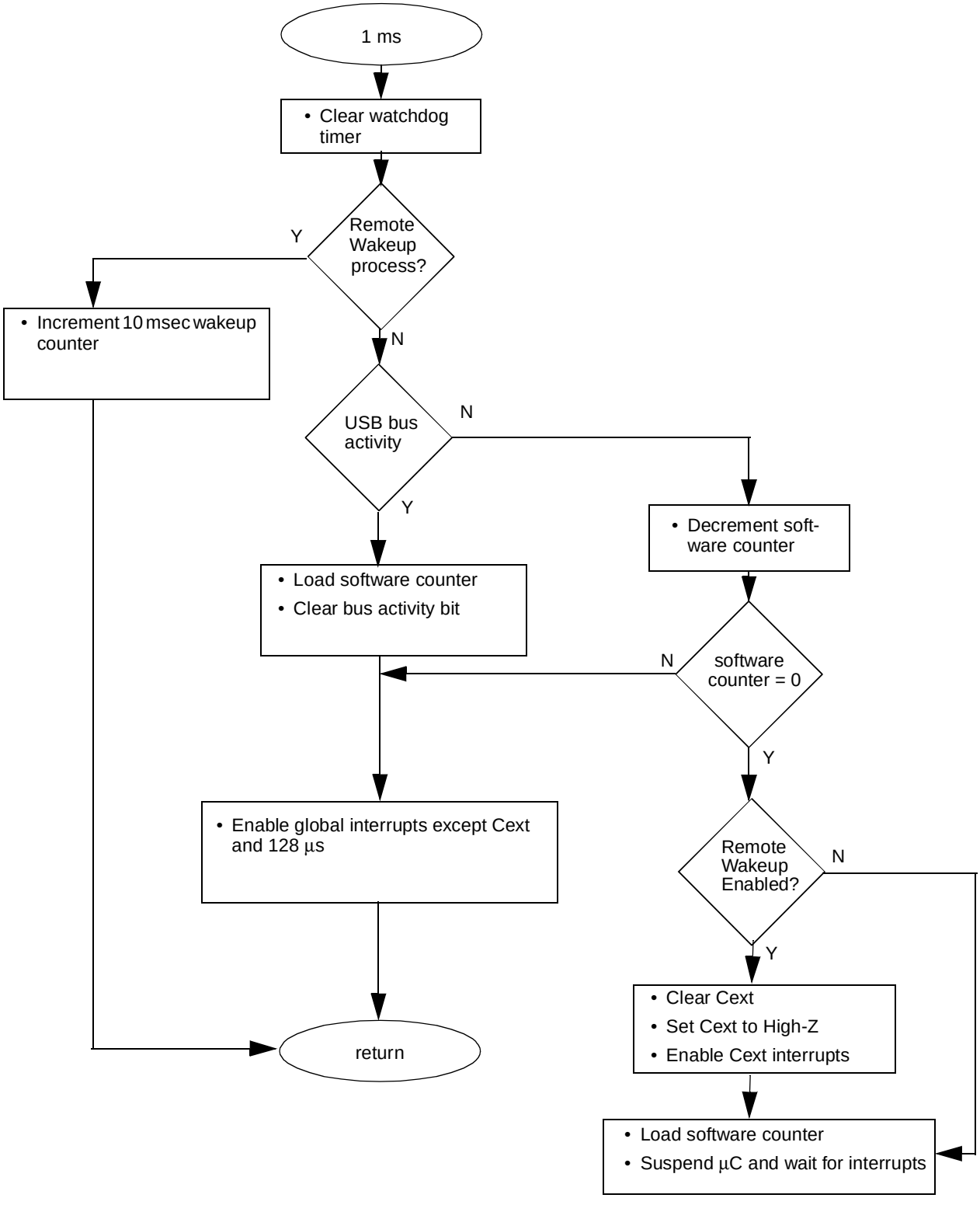


**Figure 16. One msec Interrupt Service Routine**

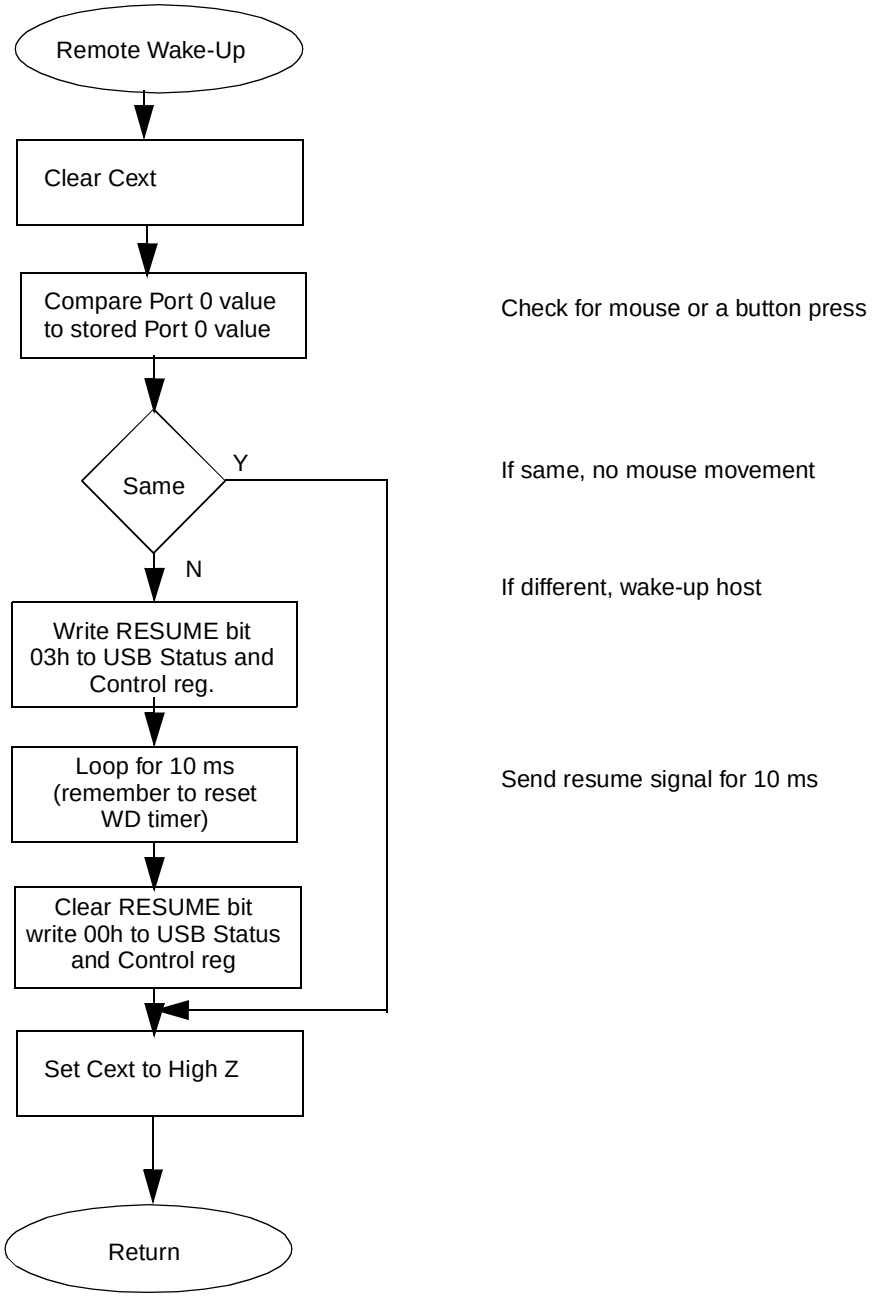


**Figure 17. Remote Wake-Up Interrupt**