# Designing a Low-Cost Analog USB Joystick with the Cypress Semiconductor CY7C63000 USB Microcontroller

## 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 analog USB joystick can be built quickly using the Cypress Semiconductor single-chip CY7C63000 USB microcontroller. The document starts with the basic operations of an analog joystick followed by an introduction to the CY7C63000 USB controller. A schematic of the analog USB joystick 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 joystick function. Several sample code segments are included to assist in the explanation. The binary code of the complete joystick 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 http://www.usb.org/

## Analog Joystick Basics

Basically, an analog joystick has a lever mechanically linked to a pair of 100 kohm potentiometers. One potentiometer varies resistance with X-axis movement (left, right) while the other one varies resistance with Y-axis movement (forward, back). IBM originally defined an interface for two joysticks, where each joystick had X and Y potentiometers and two “fire” buttons. Modern joysticks like the Thrustmaster™ Flight Control System actually use up all of these interfaces in a single joystick:

- One set of X and Y potentiometers for direction control
- One rocker switch that consists of four buttons in the four directions top, bottom, left and right for a “hat”. This is used by games to change the player’s viewing direction.
- Four “fire” buttons.

IBM defined a very simple hardware interface at I/O address 201H for joysticks and other analog devices (e.g. game pads). Each button has a pullup resistor to 5V. The signal from a button is HIGH if the button is released and LOW if the button is pressed. There are no interrupts from the game port. That means reading the port requires periodically polling address 201H and implementing a button debounce function in software.

Unfortunately, converting the X and Y potentiometer values to 8-bit digital values is not quite that simple. The traditional way to solve this problem [1] goes something like this:

1. Write anything to I/O address 201H to start a one-shot multivibrator. The “multivibrator” has a one-shot for each potentiometer. Starting the one-shots has the effect of causing all four of the position signals to go HIGH.
2. Read and save a starting count value from a system timer.
3. Each one-shot has a timing capacitor that is discharged to ground through a 2.2 kohm resistor in series with a potentiometer.
4. The timing capacitor for each one-shot will eventually discharge far enough to cross the one-shot threshold, which will cause the one-shot output to be driven LOW.
5. The software continuously polls the I/O port while any of the one-shot outputs are HIGH. Whenever a HIGH to LOW change is detected on the one-shot outputs, the system timer is read. The software calculates the potentiometer value from the difference between the new count and the start count.

The one-shot timing capacitors are selected to roughly obey the relationship:

time delay = 24.2 µs + 0.011 µs * resistance (ohms)

## 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 a 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 a mouse, joystick, or 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 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.

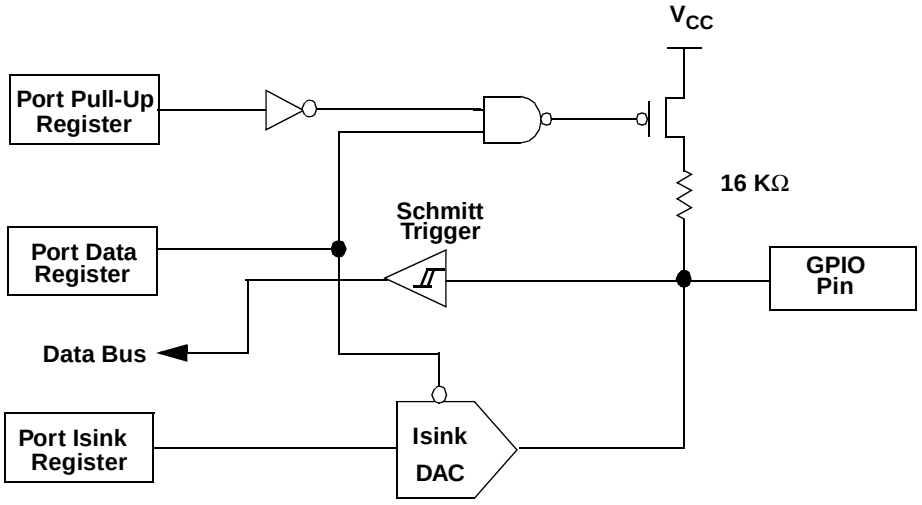


**Figure 1. One General Purpose I/O Line**

### 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 1*. 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 LEDs.

Port 1 outputs are specially designed to drive high-current applications such as LEDs. Each Port 1 output is much stronger than its Port 0 counterparts at the same drive level setting. In other words, the lowest and highest drive for Port 1 lines are about 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 Interrupt Enable Register. All GPIO interrupts are further masked by the Global GPIO Interrupt Enable Bit in the Global Interrupt Enable Register

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.

## Hardware Implementation

*Figure 2* is the schematic for a joystick application.

All I/O 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 0 to 3 of Port 0 are used to support up to four "fire" buttons. The only external components are the four buttons.

Bits 4 to 7 of Port 0 are used to support a "hat" function. The hat is limited to nine possible positions. In the left to right direction, the X value can be maximum left, centered, or maximum right. In the forward and back direction, the Y value can be maximum forward, centered, or maximum back. Again, there are no external components except the buttons.

Bits 0, 1, and 2 of Port 1 are used to read the X, Y, and THROTTLE potentiometers respectively. The only external components needed are a 1800 pF capacitor and a 2Kohm resistor for each axis.

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 $V_{CC}$ (USB Power).

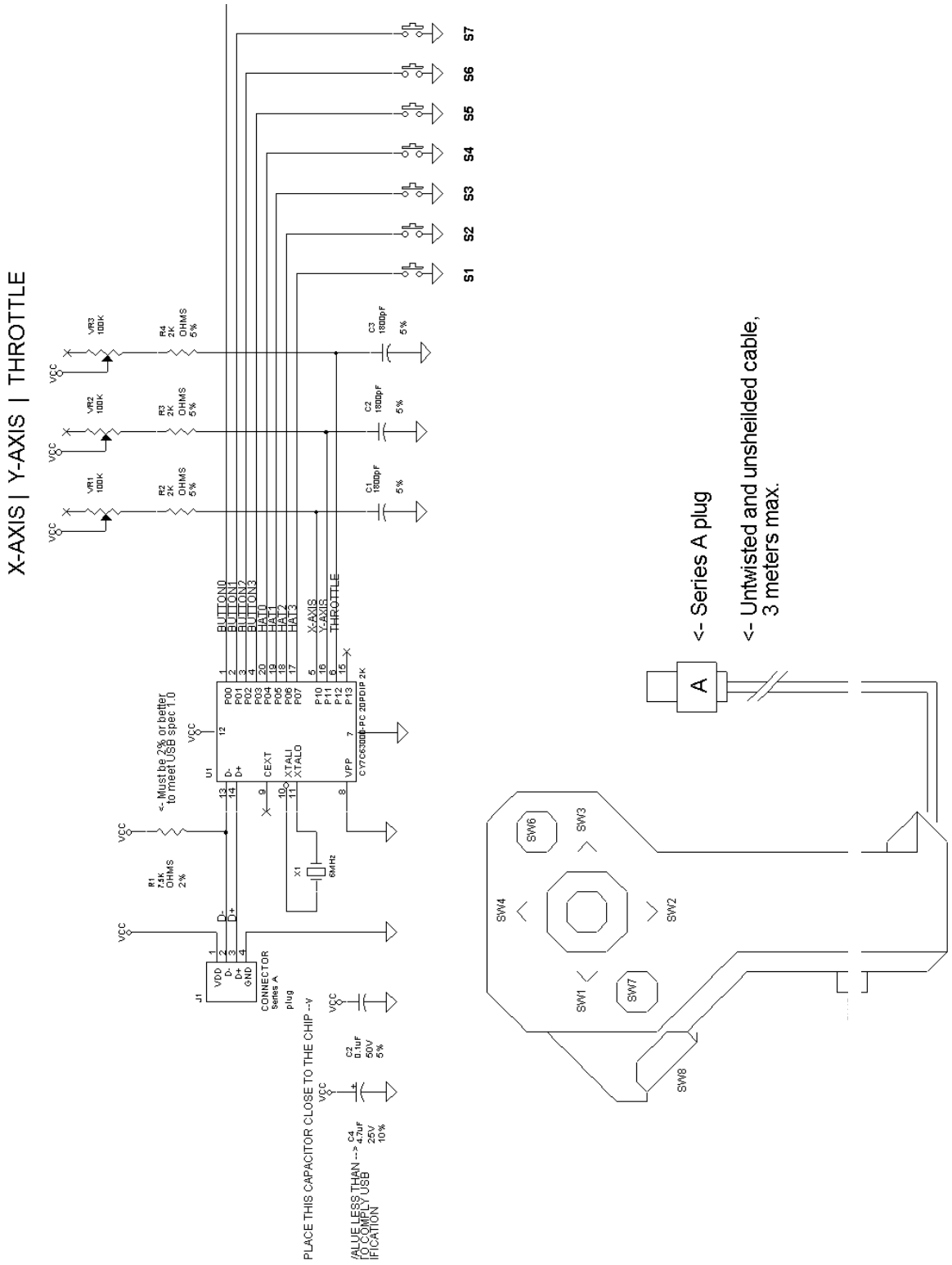


**Figure 2. Hardware Implementation**

## Firmware Implementation

### USB Interface

All USB Human Interface Device (HID) class applications such as a joystick, follow the same USB start-up procedure. The procedure is as follows (see *Figure 3*):

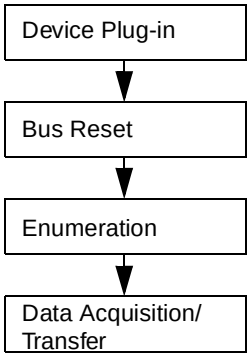


**Figure 3. USB Start-Up Procedure**

*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 4*).

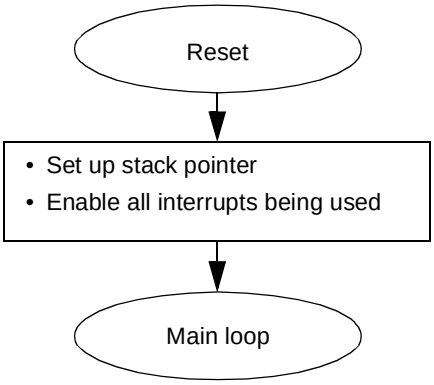


**Figure 4. Reset Interrupt Service Routine**

*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. The host then asks for the device descriptor, configuration descriptor and HID report descriptor. The descriptors hold the information about the device. They will be discussed in detail below. Using the information returned from the device, the host now knows the number of data endpoints supported by the device (in a USB joystick, there is only one data endpoint). At this point, the process of enumeration is completed. See *Figures 5*, 6 and *7*.

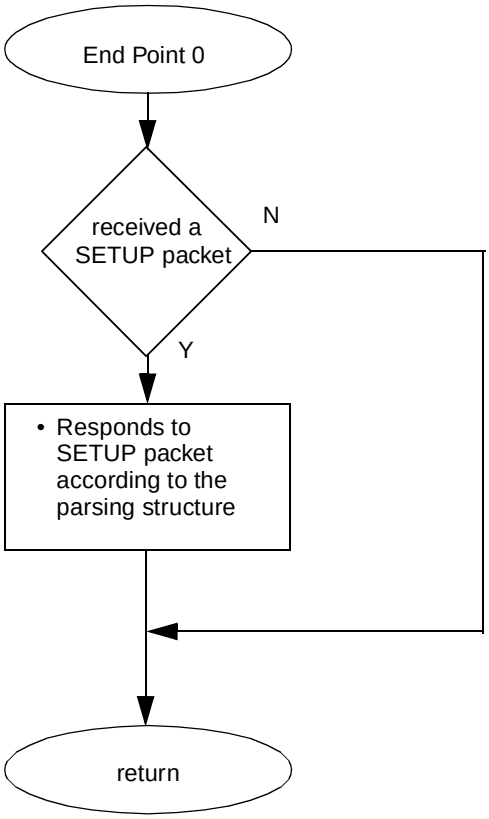


**Figure 5. Endpoint 0 ISR**

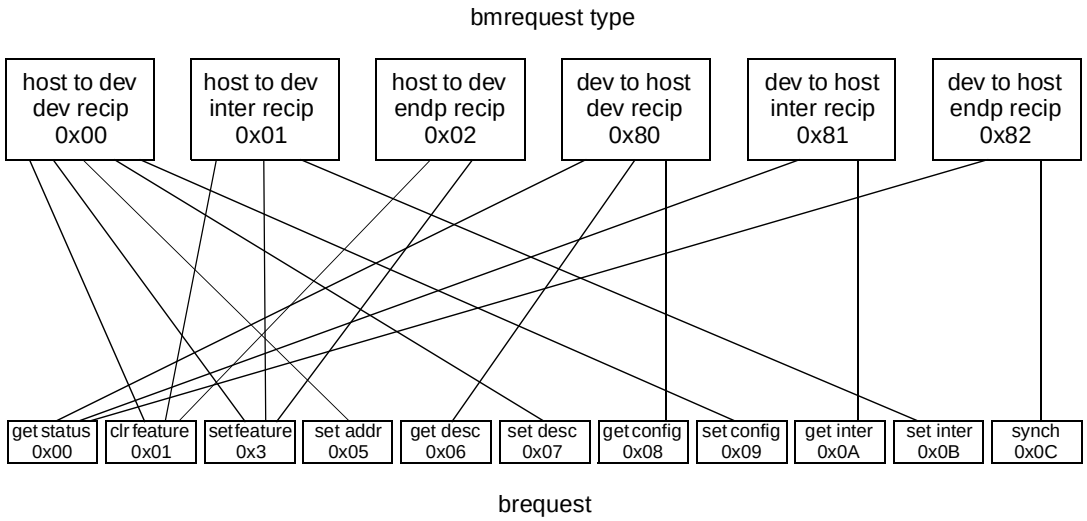


**Figure 6. USB Standard Request Parsing Structure**

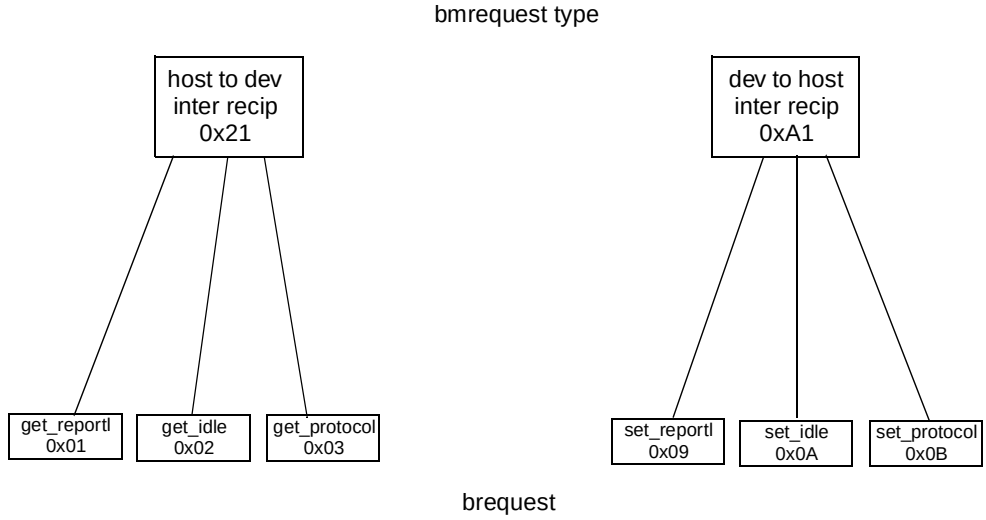


**Figure 7. USB HID Class Request Parsing Structure**

*Data Acquisition/Transfer*

The firmware polls the joystick buttons and the potentiometers for the X, Y, Z and THROTTLE axes. The status of the buttons as well as the displacements for each axis are sent to the host using endpoint 1 (see *Figure 8*). When the host issues IN packets to retrieve data from the device, the device returns six bytes of data. These six bytes hold the joystick control data (see *Figure 9*).

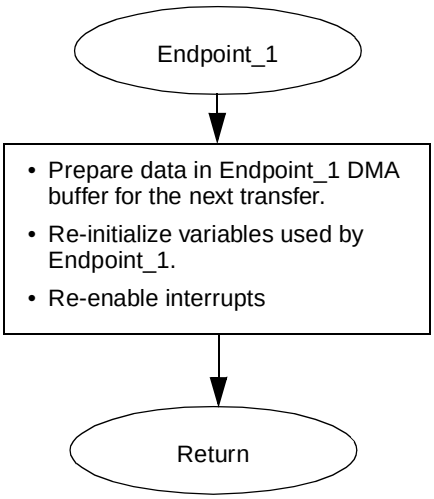


**Figure 8. Endpoint 1 Interrupt Service Routine**

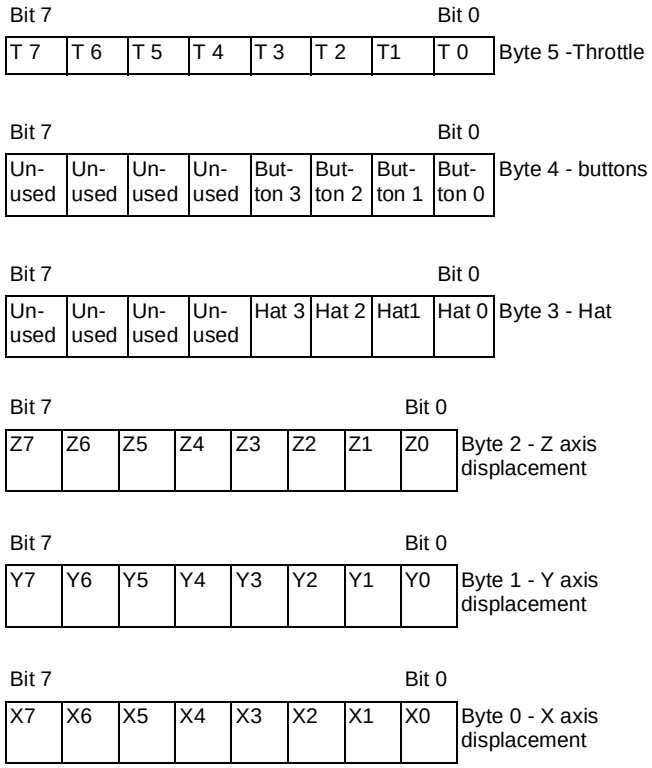


**Figure 9. Data Organization for USB Joystick**

The byte order and bit field positions are defined by the HID report descriptor (discussed below).

### 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 (Joystick)
Collection (Application)
    Usage (Pointer)
    Collection (Physical)

        Usage Page(Generic Desktop)
        Usage (X)
        Usage (Y)
        Logical Minimum (-127)
        Logical Maximum (127)
        Physical Minimum (0)
        Physical Maximum (255)
        Report Size (8)
        Report Count (2)
        Input (Data, Variable,
         Absolute)

        Usage (Rotation about
        z-axis)
        Logical Minimum (-64)
        Logical Maximum (63)
        Physical Minimum (0)
        Physical Maximum (255)
        Report Size (8)
        Report Count (1)
        Input (Data, Variable,
         Absolute)

        Usage (Hat switch)
        Logical Minimum (1)
        Logical Maximum (8)
        Physical Minimum (0)
        Physical Maximum (315)
        Unit (Degrees)
```

```
            Report Size (8)
            Report Count (1)
            Input (Data, Variable,
             Absolute)

            Usage (buttons)
            Usage Minimum (1)
            Usage Maximum (4)
            Logical Minimum (0)
            Logical Maximum (1)
            Report Size (1)
            Report Count (4)
            Input (Data, Variable,
             Absolute)

            Report Size (1)
            Report Count (4)
            Input (4 bit padding)
        End Collection

        Usage Page(Generic Desktop)
        Usage (Slider)
        Logical Minimum (0)
        Logical Maximum (255)
        Physical Minimum (0)
        Physical Maximum (255)
        Report Size (8)
        Report Count (1)
        Input (Data, Variable,
         Absolute)
    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. Note that Usage (Slider) refers to the joystick throttle. Usage (Rotation about z-axis) is implemented in the firmware, but is not depicted in *Figure 2*.

Logical Minimum and Logical Maximum are used to bound the values that a device will return. For example, a joystick that will return the values 0 to 255 for the x-axis input will have a Logical Minimum (0) and Logical Maximum (255). These are different from Physical Minimum and Physical Maximum. Physical boundaries give some meaning to the Logical boundaries. For example, a thermometer may have Logical boundaries of 0 to 999, but the Physical boundaries may be 32 to 212. In other words, the boundaries on the thermometer are 32 to 212 degrees Fahrenheit, but there are one thousand steps defined between the boundaries.

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 two axes of the joystick. 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. For example, a minimal joystick can be described as a collection of four data items (X axis, Y axis, and two buttons). 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/.

**Calibration**

Calibration of the joystick is performed by the host.

**Functionality**

The main infinite loop implemented in the firmware is a continuous polling scheme of the joystick fire and hat buttons. (See *Figure 10*). This loop is entered immediately following calibration of the joystick. First, all bits of Port 0 are polled in order to determine if any of the buttons have been pressed. The movement of the stick (and throttle if the joystick has this feature) is implemented using the GPIO interrupts and the displacement calculation described below is performed. The One_msec interrupt subroutine is used to discharge the capacitors and save the start count (See *Figure 12*). When the Port 1 bit for the axis being read transitions from LOW to HIGH a GPIO interrupt occurs and the end count is saved (See *Figure 11*).

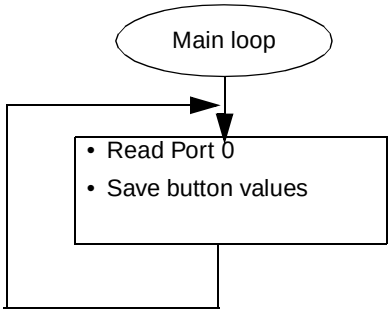


**Figure 10. Main Loop**

*Displacement Calculation*

The measurement algorithm will look like this:

For each axis defined...

1. Write a zero to Port 1 to discharge the 1800 pF timing capacitors.

2. Read and save the free running timer value as the start count.
3. When the Port 1 bit for the axis being read transitions from LOW to HIGH a GPIO interrupt occurs.
4. Read and save the free running timer value as the end count for that measurement (X or Y).
5. Subtract the start count from the end count.
6. Subtract new joystick position from previous joystick position (or vice versa depending on which position value is larger). If value less than 4h save previous position in endpoint 1 FIFO otherwise save new position in endpoint 1 FIFO. This is done to reduce joystick jitter.
7. The value in the endpoint 1 FIFO will be sent to the host on the next data transfer.

The host will then compare this value to a reference value that was found during calibration. From this, the host is able to determine how much and in which direction the joystick has moved and can act accordingly

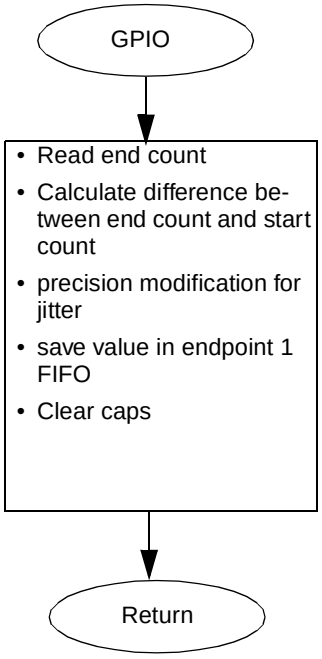


**Figure 11. GPIO Interrupt Subroutine**

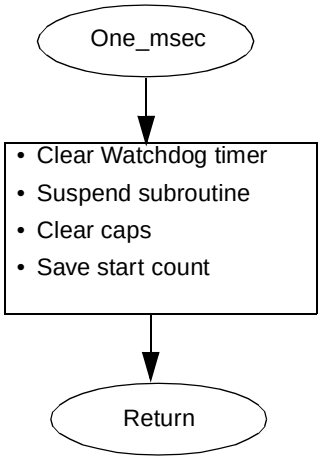


**Figure 12. One_msec Interrupt Subroutine**

## 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.

## References

[1] Second edition of “The Indispensable PC Hardware Book”