

**Designing a  
Universal Serial Bus (USB) Device  
Using the Cypress CY7C63001**

**A USB Thermometer**

## Introduction

### Purpose

The purpose of this application note is to:

- Provide a brief overview of the Universal Serial Bus (USB) system.
- Describe the design of an actual USB device using a CY7C63001 Cypress USB controller device.
- Illustrate how a Windows application communicates with a USB device through the Windows drivers.

It provides:

- A set of common assembly language routines for use by Cypress USB controllers to handle the interface to the USB system.
- A USB design guide for designing with the Cypress USB controller.
- A device driver which may be used to develop applications for non **HID** USB devices.
- Examples of communicating with drivers and USB devices for applications software.
- A glossary of common USB terms with which a user may be unfamiliar. We have defined **highlighted** terms in the glossary.

### Scope

It is assumed that the reader is familiar with (but not necessarily expert at) Windows programming using Microsoft Visual Basic (or C) and with assembly level programming of micro controllers.

This application note covers the basics of the USB bus architecture, but it is not intended to be a comprehensive reference to the USB. For more information on USB, please see "The Universal Serial Bus Specification." This document is available online for download (<http://www.usb.org>) and it is included in the USB Starter Kit CD-ROM.

It is beyond the scope of this application note to address any issues involving the actual writing of Windows device drivers.

### Operating Systems and Hardware Requirements for USB Support

There are several requirements that must be met in order to successfully use USB devices on your system:

- Your hardware must support the USB.  
Your system must contain a USB host controller and have USB connectors.
- Your **BIOS** must enable the USB host controller or provide the option to enable USB.
- Your operating system must support the USB.
- You must have the proper drivers for your USB device.

## Cypress USB Thermometer Application Note

### Hardware

- Integrated motherboard support

Most motherboards and systems produced today include integrated support for the USB. If your system does not have integrated support, you can use an add-in adapter (see below).

USB connectors are identified by the symbol:



- BIOS support

If your motherboard has integrated support for the USB, your motherboard must also have a BIOS that has support for the integrated hardware. Also, the BIOS Setup must enable the integrated USB hardware. Currently, some systems have USB ports and hardware, but do not have BIOS support for the ports (odd, but true).

If you are having problems getting your system to recognize your integrated USB Host Hub, you should check both to see if your BIOS has support for the USB Host Hub and to ensure that the host hub is enabled. The control of the USB hardware should show up as an "Enable/Disable" item in your **BIOS setup program**.

- Add-in adapter support

If your motherboard does not support the USB directly or you wish to add additional hubs to your system, adapters can be added to an existing computer to provide support. Usually, these adapters plug into the **PCI** bus.

### Operating Systems

At the time of this writing, there is only one released operating system that supports the USB and that is the Microsoft OSR 2.1 release of Windows95™. Microsoft OSR2.1 is an upgrade of Microsoft OSR2.0. OSR2.0 does not support USB and must be upgraded to OSR2.1 with the appropriate upgrade from Microsoft.

Currently OSR2.0 and OSR2.1 are only available from OEM's on newly purchased machines or on the Microsoft Developer's Network CD's. They are not available through retail channels.

There is currently no upgrade for Microsoft Windows 95 that provides USB support.

There is also currently no upgrade for Microsoft Windows NT that provides USB support.

Full support is available in Windows 98 and Windows NT5.0. These operating systems are expected to be generally released in 1998. Developers may obtain beta copies of Windows 98 for early development. To get more information about the Beta Developers Program, go to the Microsoft web site (<http://www.microsoft.com>).

### Drivers

The vendor of the USB device will supply drivers that may be required for your device.

## Cypress USB Thermometer Application Note

### Common Questions

**Q. "How do I determine if my operating system supports the USB?"**

- A. In order to use the USB with the Windows operating system, you need to have OSR2.1 or a more recent version of Windows such as Memphis (Windows98, currently in Beta test).

You may determine the version of Windows you have through the System Properties.

Information to help you determine which version of the Windows operating system you have is also available from Microsoft at:

<http://www.microsoft.com/kb/articles/q158/2/38.htm>

**Identifying your operating system as OSR2.0, OSR2.1 or Memphis:**

ORS 2.0 is Windows 95 version 4.00.950b.

OSR 2.1 is Windows 95 version 4.00.950b with the USB supplement installed.

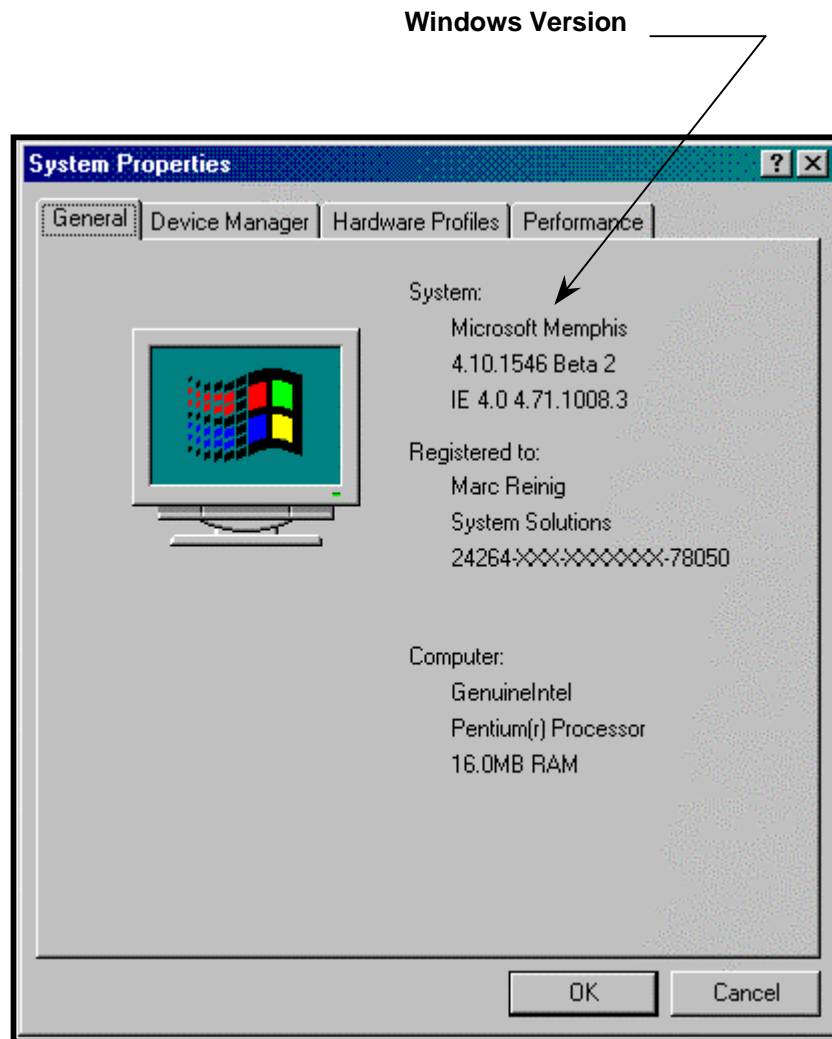
Memphis is Windows 98 version 4.10.1423 or later.

**System Properties**

The version of Windows you have installed can be found by clicking on the "System" icon in the Control Panel (See *Figure 1* and *Figure 2*).



**Figure 1 Microsoft Windows 95™ Control Panel**



**Figure 2 Windows System Properties**

You can determine whether the USB supplement has been installed by using the "Add/Remove Programs" application which is also found on the control panel. If the USB supplement is successfully installed, you should be able to find it in the list of software that can be added or removed in the "Install/Uninstall" options within the "Add/Remove Programs" screen.

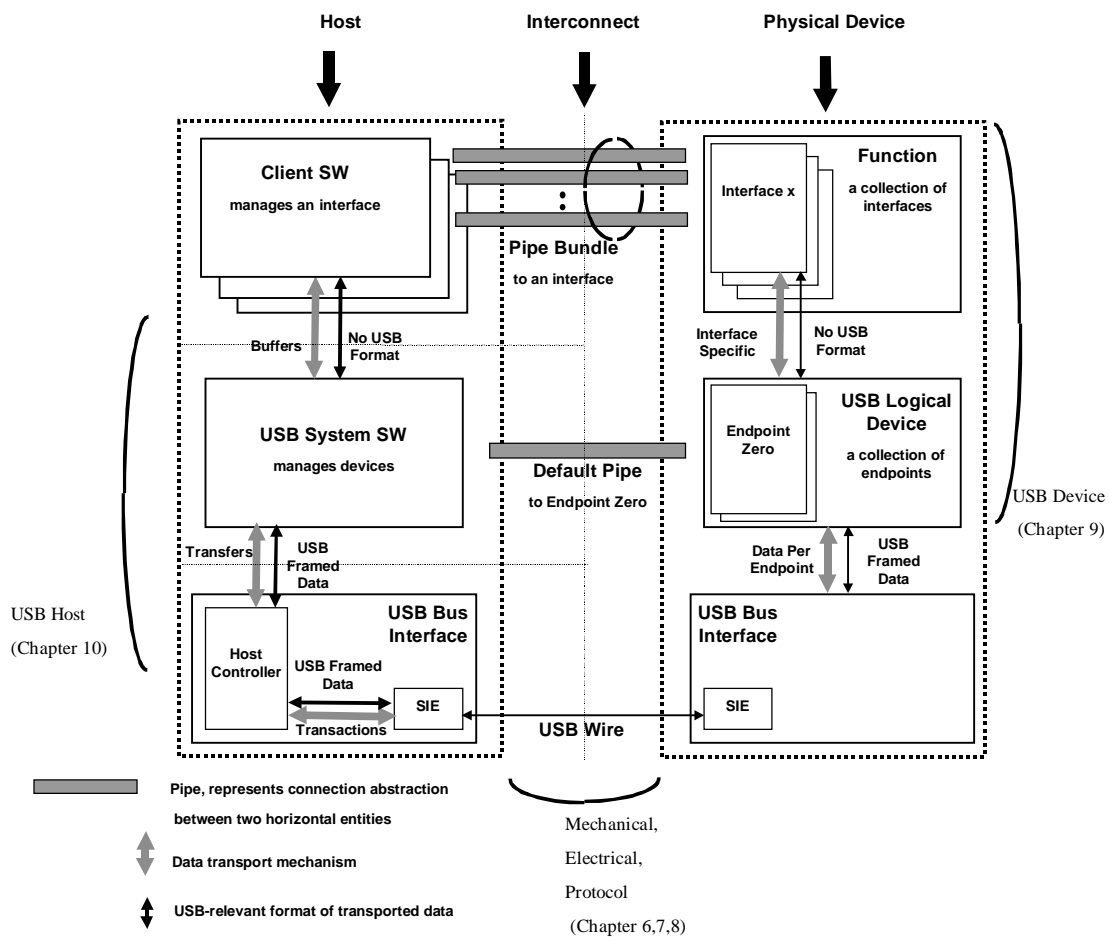
## USB Basics

This section and the following sections of the application note are meant to provide a brief overview of the USB system. They are not intended to provide a comprehensive treatment of the topic. Readers are directed to the various USB documents, including the USB specification and subsequent USB clarifications. You can obtain these documents at the USB web site (<http://www.usb.org>).

### USB Device Requirements

To provide a device or application that communicates on the USB requires adherence to a number of items in the USB specification.

The amount of requirements can seem daunting at first.



**Figure 3 Communications in the USB System**

## Cypress USB Thermometer Application Note

The minimum requirements for a USB device are:

- Respond to the minimum USB commands per the Spec.
- Observe the mechanical protocol for physically connecting to the USB.
- Observe electrical protocol for electrically connecting to the USB.
- Observe the USB specified data format and transport protocols.
- Handle Idle condition detection and enter and leave the Suspend condition per the USB Spec.
- Detect the USB reset condition and take appropriate action.
- Power management per the USB spec.

That is quite many things to keep track of. Fortunately, many of these issues are handled automatically by the Cypress USB Controller or are provided by Cypress in assembly code.

This leaves the designer free to concentrate primarily on the application specific aspects of their project.

Using the Cypress USB controller, there is minimal additional hardware and software that is required to interface to the USB bus. We have broken down the items into two groups: items are handled for the designer by the Cypress USB controller (or are provided by Cypress) and items that the designer must supply.

Items provided by Cypress:

- Responding to the minimum USB commands per the Spec.

Cypress supplies much of the needed assembly code that the controller can use to handle the standard minimum set of USB commands.

**ACK**, **NACK**, and **Stall** responses to **SETUP**'s, **IN**'s and **OUT**'s are handled based on the setting of only a few control bits.

The Cypress USB Controller automatically sends data buffers and receives data transfers independently on both Endpoint 0 and Endpoint 1 in response to **SETUP**'s, **IN**'s and **OUT**'s using its 8-byte **FIFO**'s.

The type of packet which was received (**SETUP**, **IN**, or **OUT**) is automatically detected by the Cypress controller and is reported in status bits. The assembly code can determine what action to take based on this information.

- Observing the electrical protocol for electrically connecting to the USB

The Cypress controller includes USB I/O drivers to connect to the bus, thus no external bus drivers are required.

## Cypress USB Thermometer Application Note

- Observing the USB specified data format and transport protocols.

The Cypress USB controller contains a Serial Interface Engine which performs all direct data handling on the bus (**bit stuffing**, **NRZI** encoding/decoding, etc.).

It also handles basic bus protocol decoding for USB features such as Reset, Idle Detection, **PID** decoding, etc.

- Handling Idle detection, entering and leaving the Suspend condition.

Bus Idle detection is handled automatically, making it easy to determine when the USB device should be put in Suspend mode.

When the Cypress USB Controller is placed in Suspend mode, its on-chip power is reduced per the USB specification, automatically.

The controller will automatically resume from the Suspend State on any USB activity. It may also be programmed to Resume after **GPIO** or **Cext** activity.

- Bus Reset and Power-On Reset conditions are automatically detected.

When a Reset is detected, the controller state is reset and program execution starts at a user-defined subroutine.

The firmware can determine which event caused the reset through status bits.

Items provided by the user:

Each USB device has unique hardware and software characteristics that are not generic to the USB system. The user must provide hardware and software development for these areas.

- Performing the correct application specific activity for each command.

While Cypress provides the majority of the assembly code for handling the interface to the USB and general controller activities for each command, the user must develop code for any activity that would be application specific.

- Observe the mechanical protocol for physically connecting to the USB

Each device must provide a means for their device to physically attach to the USB (USB Spec 1.0, 6.3). Attaching your device to the USB requires either:

- A USB cable with one end permanently attached to your device and a USB "A" type plug on the other end (to be connected to the host controller port).
- A USB "B" type receptacle on your device to which a USB cable with both an "A" and a "B" plug will be attached.

- Observing the electrical protocol for electrically connecting to the USB.

To allow the USB hub to identify your device as a low speed device, the user must supply a pull up resistor between the "D-" line and a positive supply voltage. This resistor can be either a 1.5KOhm, 5% resistor to connected to a 3.14 V - 3.47 V supply or a 7.5KOhm, 1% resistor connected directly to the USB Vbus. (USB Spec 1.0, 7.1.3 and Device Working Group Review Request 135, 3.3V Regulator Tolerance)

For more information on proper USB termination, please see the USB specification.



## **Cypress USB Thermometer Application Note**

- Power management per the USB spec.

The user must handle power shutdown and enable of external logic during Suspend and other stages of USB operation.

## **Communications Between Applications and Devices**

### **Device Drivers**

Applications cannot communicate directly with USB devices, and must do all communications through a **device driver**. In turn, the device driver will communicate with the USB device through the USB system drivers.

The device driver opens **pipes** to **endpoints** on the USB device. It uses these pipes for all communications with the device.

### **Addresses and Endpoints**

The Cypress CY7C63X0X family of USB Controllers support two endpoints (0 and 1). The USB device uses these to communicate with the USB system in a manner defined by the USB specification, its device driver, and its micro code.

Endpoint 0, the device default control endpoint, is always configured. The USB system uses this endpoint for command and control of the device and for retrieving its status. Communications to Endpoint 0 is always through a **message pipe** and is bi-directional.

The Cypress USB Controller supports an additional endpoint, Endpoint 1. This endpoint is unidirectional, can only be configured as an interrupt endpoint. It can be used to transmit data from the device to the host.

**Human Interface Devices (HID)** (mice, joysticks, and keyboards, etc.), frequently use Endpoint 1 to supply new information to the host on request.

The Cypress USB Controller also supports a programmable address register that holds the device logical address on the USB. The USB assigns this address to the device during **enumeration**.

### **Enumerating a USB Device**

For a host to use a USB device, the host must first enumerate it. This procedure allows the USB system to identify the device, load its driver if necessary, assign it a logical address, and configure it. This sequence of events is called enumeration. In order for the host to use any of the functions of a device, the host must first enumerate the device.

Each USB device has a pull-up resistor attached between a specific positive voltage and either D+ or D- signal line of the USB. This configuration indicates whether the device is a low-speed or full-speed device. D- pulled high indicates the device is a low-speed device while D+ pulled high indicates a full-speed device. The CY63XXX series of micro controllers are low-speed USB devices.

When a device is plugged into the USB, the USB system will detect the event. The USB system will determine whether the newly attached device is a full- or a low-speed device by determining which line (D+ or D-) is pulled high. It will also determine to which port the USB device is attached.

The USB system will then initialize the new device by sending it a USB reset. This reset will not affect any other device because the reset is only sent to the new devices' ports and not to the other ports.

The system will then request the device's **Device Descriptor** (see **Control Transfer** below) to Endpoint 0 at the default USB address, zero. The device will respond by sending its Device Descriptor as requested.

When the USB system has received enough of the device descriptor to determine what the newly attached device is, it will then attempt to locate an existing driver for the device. If a driver exists, the USB System will automatically load it.

If the operating system cannot locate an existing driver, it will automatically prompt the user to supply a driver, and will load the driver when the user has supplied it or its location.

## Cypress USB Thermometer Application Note

Once the driver has been loaded, the host will send the device a unique logical address that will be used for all future communications.

It will then again request the device descriptor at the new address. After it completes reading the device descriptor, it will then request the configuration descriptor.

Finally, the operating system and/or the driver will send the configuration information to the device.

This completes the process of enumeration and the host may now use any of the configured functions of the device.

### USB Device Communication (Packets and Transactions)

The host communicates with a USB device during enumeration and operation through a series of transactions on the USB. Most communication, including the process of enumeration, is accomplished using these transactions. At first, deciphering these transactions may seem complex, but it is really quite simple.

A transaction consists of several **packets** of information sent on the USB.

Each transaction consists of:

- A token packet

This identifies the type of transaction (**IN**, **OUT**, or **SETUP**) and the recipient's address and endpoint. The host is the only one that will issue a token packet. **INs** and **OUTs** correspond to **IN** and **OUT** transactions. A **SETUP** token is used for a Control Transfer transaction. These transactions are covered below in detail.

- Some amount of data

Either the host or the device will send data. This can range from zero to some number of bytes. The direction of the transfer will be indicated in the token. **IN** and **OUT** transactions transmit data from device to host or from host to device, respectively. The size of the transfer is determined by the context of the transaction. **SETUP** transactions always contain eight bytes of data sent from the host to the device. They are simply a special form of an **OUT** transaction.

- A handshake packet

This will indicate the status of a data transfer in a transaction.

The Cypress USB Controller automatically decodes these packets, and provides the information contained in easy-to-use status and data locations. The designer can then use this information in assembly code routines to determine what action to take. Since the designer does not need to handle these packets individually, we will not discuss them in detail here.

The designer does need to handle the transactions, which are made up of these packets.

### Transaction Types (INs, OUTs, and Control Transfers)

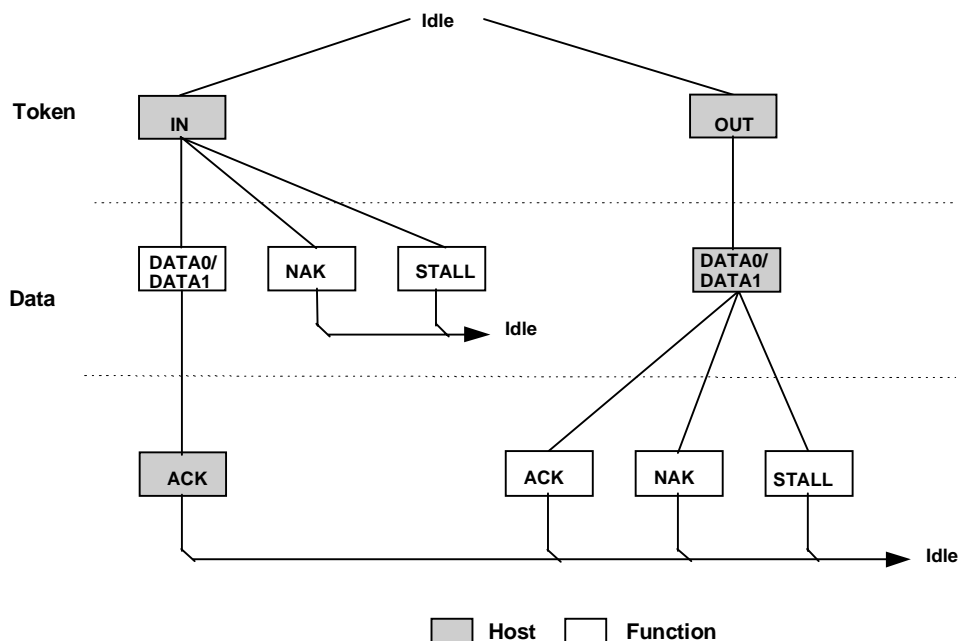
We need to be concerned with three types of transactions. They are **INs**, **OUTs**, and **Control Transfers**.

## IN's and OUT's

- An “**IN**” tells the device to send the data that the host is expecting.
- An “**OUT**” tells the device to take the data the host will send.

Figure 4 shows the structure of **INs** and **OUTs**. You can see that they consist of three packets:

- A token identifying the transaction as an **IN** or **OUT**.  
The host always sends this.
- The data  
Data either is sent by the device or host. The source of the data is dependent on whether the transaction is an **IN** or **OUT** respectively.
- A handshake  
The recipient of the data sends this to indicate whether or not the transfer was successful.



**Figure 4 Data Transfer Transaction**

**IN** and **OUT** transactions may occur singly or as part of a larger sequence.

When used singly, an **IN** transaction is used by a host to communicate with an interrupt endpoint (such as Endpoint 1 on the Cypress USB Controller) in order to retrieve data from the device. An example would be a host retrieving position information from a mouse on a regular basis.

When used this way, the host must first configure the device to respond to the **INs** on Endpoint 1 by communicating on Endpoint 0 with control transfers.

## Cypress USB Thermometer Application Note

An **IN** or **OUT** can also be used as part of a larger sequence of transactions called a Control Transfer. In a control transfer (see below), **INs** or **OUTs** are used both to transfer data and to return status to the host.

### Control Transfers

A host needs to be able to communicate with a device to control it or obtain its status. This is done through the device's Endpoint 0 using control transfers. A control transfer is simply a special series of transactions in a specific sequence between a host and a control endpoint (Endpoint 0).

**Control transfers** are used by a host to enumerate a device or change its state in any way, or obtain its status. Control transfers may also be used to send data to a device (although this is not the original intention of a control transfer).

A control transfer has at least two transaction stages (a **SETUP** stage and a Status stage). It may also have an optional data stage, which consists of one or more **IN** or **OUT** transactions.

The **SETUP** stage is illustrated in *Figure 5*.

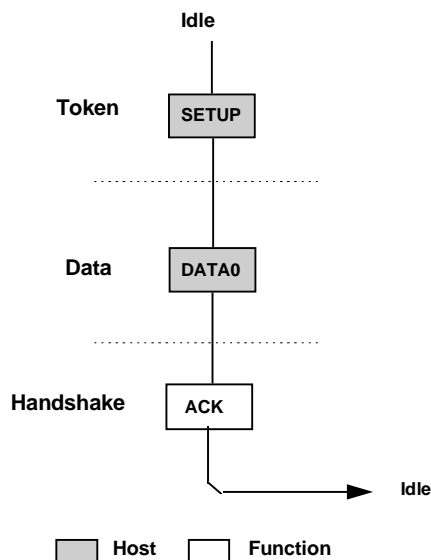
There are three types of Control Transfers: a Control Read, a Control Write, and a No-data Control.

The three types are illustrated in *Figure 6*.

### SETUP Stage

The **SETUP** stage (See *Figure 5*) consists of three packets: the **SETUP** token, eight bytes of data, and the handshake packet.

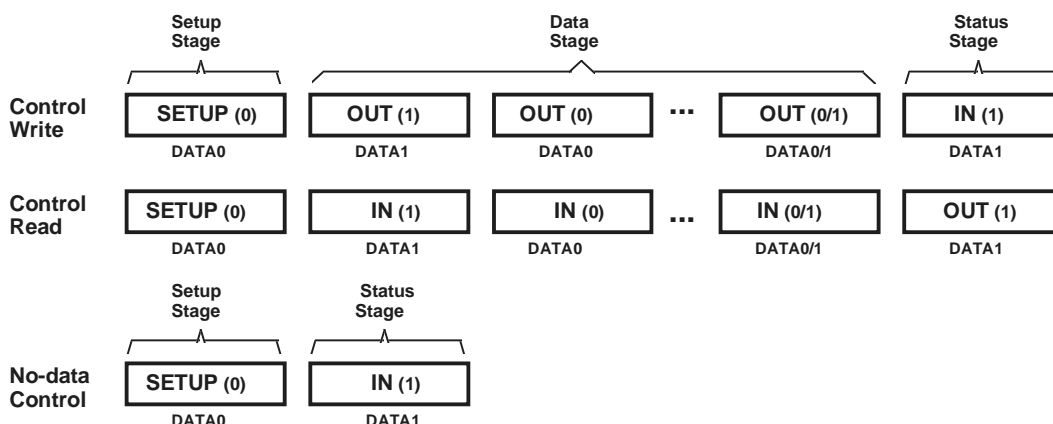
A “**SETUP**” is a special type of “**OUT**” that gives a specific command to the USB device to do something.



**Figure 5 Control SETUP Transaction**

## Data Stage

The data stage, if it exists, consists of one or more **IN** or **OUT** transactions (See Figure 6), the direction and number of which are specified in the **SETUP** stage. All transactions in the data stage must be in the same direction, i.e. either all **INs** or all **OUTs**.



**Figure 6 Control Read and Write Sequences**

## Status Stage

In all three types of Control Transfers, the Status stage is an **IN** or an **OUT** (See Figure 6). Additionally, the status stage has a data size of zero bytes and has a direction that is the opposite of the direction of the previous transfer. Since the **SETUP** is actually a special **OUT**, the Status stage for a No-data Control is an **IN**. The Status stage is also a zero byte transfer (an **IN** or an **OUT** with no data).

In a Control Read or Write, the **Setup** will specify how many bytes of data are to be read or written. However, if the Host enters a Status stage prior to completing the transfer of data, all further data transfers for that Control Transfer are canceled. The host can enter the Status stage by issuing an **IN** during a Control Write or an **OUT** with a zero byte data length during a Control Read.

## How Control Transfers are Used

Control transfers are like a command and response between the host and the USB device. They are simple and well defined in the USB Specification.

When a **SETUP** packet is received by the device, the device knows that it is receiving a Control Transfer. It decodes the **SETUP** to determine what to do with the following data (if any), and then proceeds accordingly.

Control Transfers handle most stages of the **enumeration** process: Get Device Descriptor, Set Address, Get Configuration Descriptor, and Set Configuration.

Since Control Transfers are used for fundamental control of the device, they are very critical. Cypress provides assembly language routines to handle the common control transfers the device will receive. The designer should be familiar with them in order to be able to add the device specific requirements of their design. These routines are located in USB.ASM on the Cypress USB CD-ROM.

The specifics of each of the Control Transfers can be found in the USB Spec, Chapter 9, "USB Device Framework".

## **Powering a USB Device**

The USB system limits both the total power, which the USB bus can supply for all devices, and the power consumed by a single device.

### **Bus Powered Devices**

A USB device may receive all or part of its power directly from the USB. However, the USB specification limits the amount of current a device can draw from the USB. This amount depends on:

1. The state of the device

- **Powered**

Prior to the device completing enumeration, power consumption is limited to one unit load (100 mA). This is a maximum value, and not an average value. (USB Spec 1.0, 7.2.1, USB Core Specification Clarifications by Section, rr97)

- **Configured**

Once configured, there are two classes of devices in terms of power consumption: low power devices and high power devices.

Low-power devices may draw no more than one unit load (100 mA) from the USB.

High-power devices may increase their power consumption to five unit loads (500 mA) or the maximum power they have specified during enumeration.

- **Operating Power Drain**

Once configured, a device may draw as much power as allowed by the USB specification.

- **Suspended**

When a device is in Suspend Mode its power drain is limited to 500  $\mu$ A. This includes the power consumed by the pull-up configuration resistor on the D+ or D- line.

2. The type of Hub to which it is attached

Hubs can supply different amounts of current to their ports depending on whether they are self-powered or bus-powered.

A device may draw up to 500 mA from the USB if it is attached to a self-powered hub.

If a device is attached to a bus-powered hub, it may only draw up to 100 mA from the USB.

The hub in the desktop PC is usually considered a self-powered hub. However, the hub in a portable PC might be considered a bus-powered hub. (USB Specification 1.0, 7.2.1)

If your device is a high powered device, it may exceed the available USB power from some hubs or in combination with other USB devices. If the power requirements are exceeded, it may become necessary to convert the design to "self-power" instead of "bus-power."

3. The maximum current that the device has told the USB system it will need

The maximum amount of power a device will draw from the USB is specified in the configuration descriptor.

This value is read the by the USB system during enumeration.

### 4. Supply voltage range

The voltage supplied by a USB hub at its “A” connector may be between 4.4Vdc and 5.25Vdc.

There may be additional loss in the cable between the “A” connector and the device itself.

Bus-powered low-power devices with detachable cables must be able to enumerate correctly with supply voltages between 4.15Vdc and 5.25Vdc at the device. Bus powered devices with an attached cable must be able to enumerate with a supply voltage between 4.4Vdc and 5.25Vdc at the “A” connector of the cable (USB Spec 1.0, 7.2.1.3).

All bus-powered devices must be able to continue normal operations when the supply voltage at the device momentarily drops to 4.0Vdc.

### Self-Powered Devices

Self-powered devices may use a combination of power from the USB and a local source. They must meet the same requirements as a bus-powered device in regards to the power they draw from the USB. However, the self-powered portion of the design is solely limited by the capabilities of the external power supply.



## **Designing the “Cypress USB Thermometer”**

### **Overview of the Cypress USB Thermometer system**

The goal of this project was to produce a low cost system that would display temperature from a remote device.

This application note documents the design of this system using the Cypress CY7C63001 USB controller chip to implement the USB thermometer.

It provides:

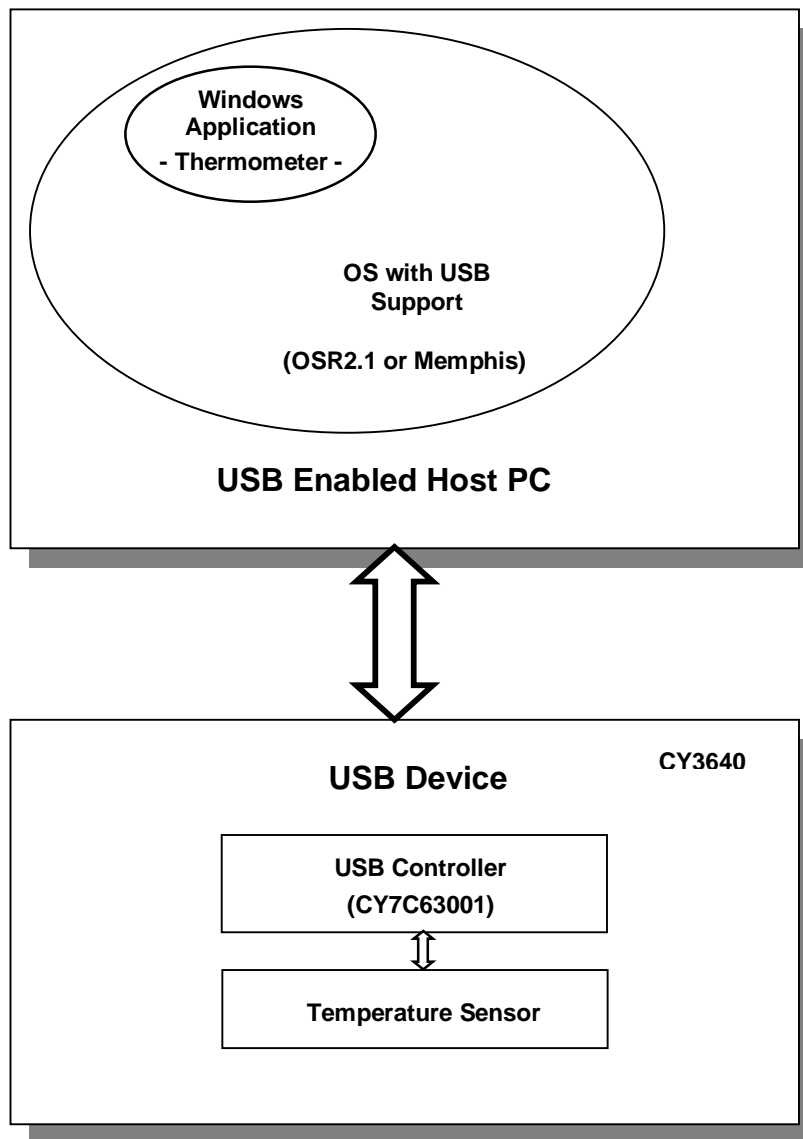
- A set of common USB assembly language routines which are usable by Cypress USB controllers to communicate with the USB system.
- A set of assembly language routines which are usable by Cypress USB controllers for performing serial I/O to external logic.
- USB hardware design guidelines for implementing a USB device using Cypress USB controllers.

### **Partitioning the elements of our USB based system**

The basic elements of the system are:

- A Windows application to acquire the temperature from the USB device and display it for the user.
- A USB enabled host PC and OS (with the appropriate drivers) to run the application and provide the USB hardware and software interface.
- A USB device which will connect to the USB, measure the temperature and communicate that measurement to the host.

*Figure 7* illustrates the elements of the temperature measurement and display system.



**Figure 7 Elements of a USB System**

A complete schematic of the Cypress USB Thermometer device is shown in *Figure 8*.

# Cypress USB Thermometer Application Note

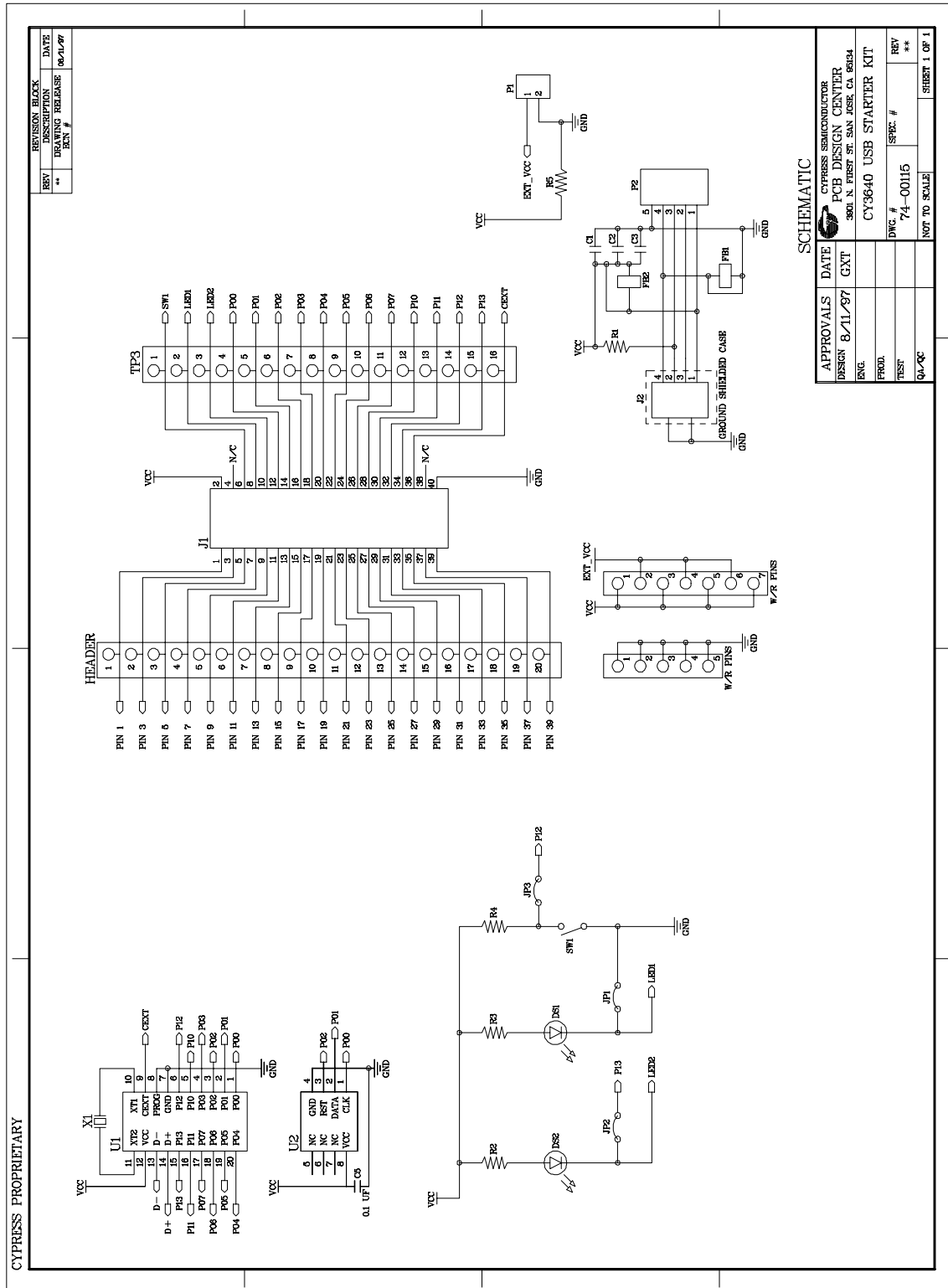


Figure 8 CY3640 Schematic

## Cypress USB Thermometer Application Note

### Partitioning the Cypress USB Thermometer device

There are four main sub-systems in the Cypress USB Thermometer device:

- The USB interface
- The temperature sensor interface
- The enumeration LED
- The Centigrade/Fahrenheit button

#### Overview

All USB communication between the host and the thermometer occurs on an interrupt basis in the Endpoint 0 interrupt service routine.

After a reset, the system is started in the main() routine in USB.ASM. This routine initializes the USB variables, the IO ports, the temperature sensor variables, and the data space. The system loops here, doing nothing except responding to USB Commands until the device has been **enumerated**.

Once the device is enumerated, the main routine also polls the temperature sensor approximately once every 10 ms to retrieve the new temperature reading, and updates the brightness of the enumeration LED if necessary.

The enumeration LED is controlled by the P13 pin of the USB controller and is turned on when a SetConfiguration command is received. This routine is decoded in USB.ASM.

An external sensor senses the temperature and all communication between the Cypress USB Controller and the sensor is through Port 0 using pins P00, P01, and P02. These routines are contained in DS1620.ASM.

The controller detects button presses using Port 1, pin P12. Routines which support button press detection are found in the GPIO interrupt servicing routine and the 1024  $\mu$ s interrupt service routine in USB.ASM.

#### The USB electrical and mechanical interface

- Identifying the device as a low-speed device

To allow the USB hub to identify our device as a low-speed device, we must supply a pull up resistor between the “D-“ line and a positive supply voltage. We chose a 7.5 KOhm, 1% resistor connected directly to the USB Vbus. (USB Spec 1.0, 7.1.3 and Device Working Group Review Request 135, 3.3V Regulator Tolerance)

It is important to insure that, if your device is self-powered, it cannot drive current into a floating upstream I/O driver. Supplying the power connected to the pull-up resistor from the USB Vbus is an easy way to insure this, and is independent of whether or not your device is self-powered. (USB Spec 1.0, 7.1.3 and USB Core Specification Clarifications rr90)

- Physically connecting the device to the USB

We chose to use a B type receptacle on our PC board. This allows us to use a detachable USB cable. However, the board also provides an in-line header footprint which the developer may use to hardwire a USB cable directly to the board.

#### The USB Protocol Interface

The routines for the functions that support this interface are located in USB.ASM.

## Cypress USB Thermometer Application Note

We communicate with one endpoint (Endpoint 0) by a pipe that supports both system and vendor specific communications.

Endpoint zero is required to support Setup oriented requests and other class or vendor requests. Each request interrupts the processor through the Endpoint 0 Interrupt service routine found in USB.ASM. This routine determines whether or not the request causing the interrupt was a **SETUP** command. If the request was a **SETUP**, the command is decoded and the subsequent routines service it.

When a token is received by Endpoint 0, the Cypress USB Controller's Serial Interface Engine (SIE) automatically determines if it is a **SETUP** packet. If the token is a Setup token, the SIE automatically places the associated data in Endpoint 0's **FIFO**.

It is only left to the designer to:

- Determine the type of Control Transfer that is indicated by the **SETUP** packet from data in the **FIFO**
- Respond appropriately to the subsequent **INs** or **OUTs** based on the type of **SETUP** packet received
- Respond correctly to the status transaction

Cypress supplies assembly language routines to accomplish this for the standard USB commands the device will receive. These are found in USB.ASM.

Two vendor-specific control transfers are supported by the assembly code: GetTemperature and SetBrightness.

When the controller receives a GetTemperature control transfer from the host, the last measured temperature reading is returned, along with a value, which indicates whether or not the button has been pushed since the last check.

When the controller receives a SetBrightness control transfer from the host, the brightness level of the enumeration LED is changed according to the value specified in the control transfer.

### The Temperature Sensing Logic

Since the Cypress USB Controller is a microcontroller with excellent control of its ports, no external logic was necessary to interface to the temperature-sensing chip.

The temperature-sensing logic required three signals: reset, clock, and data (IN/OUT). All signals required to operate the temperature-sensing device were controlled by micro code.

Port 0 is a low drive port and is suitable for use with low power devices such as CMOS and photo detectors. The temperature sensor is a CMOS device so Port 0 was used (P00 – P02).

The routines for these functions are primarily located in DS1620.ASM.

A set of routines for generating timing signals was developed. These routines are generic for generating any set of signals commonly needed for this type of purpose.

The assembly code for this interface can be found in files DS1620.ASM and DS1620.INC on the CDROM.

A simple 9-bit temperature value is read from the temperature sensor every 10 ms. After enumeration, the temperature sensor is initialized and placed into a continuous conversion mode. It stores the current temperature internally. Thereafter, the temperature is read every 10 ms and the value is placed into the USB endpoint one FIFO buffer for temporary storage.

When a "Read Temperature" command is decoded, it is copied from the EP1 FIFO along with the button status and placed into the EP0 FIFO where it will be returned to the host as part of the command.

## Cypress USB Thermometer Application Note

### The Enumeration LED

The LED is controlled by I/O Port P13. Port 1 has high drive capability and is capable of driving LEDs and other high current circuitry. The LED draws 20 mA and so Port 1 (P13) was chosen to drive it.

When P13 goes low, this turns the LED on. The LED indicates the status of the USB connection. Once this device has been enumerated, the LED is turned on. This occurs in the SetConfiguration routine in USB.ASM.

A Vendor Specific Control Transfer, Set Brightness, is supported to allow us to adjust the brightness of the LED. This passes the new brightness level to the controller.

To change the brightness of the LED we use a feature of the Cypress USB Controller that allows us to set the strength of the output buffer of each port. We adjust the LED's brightness by first setting the new brightness value (default: FFh = High) and then setting the brightness update field.

The routine MAIN in USB.ASM checks the update variable and sets the new brightness by loading the value of the brightness variable into the P13 port strength register if necessary.

### The Centigrade/Fahrenheit Button

A push button switch is used to indicate to the Windows application that the user wants the Centigrade/Fahrenheit display mode to be toggled.

The switch is a normally open momentary closed device. One side of the switch is connected to P12 and is also pulled high by a 10K resistor to Vcc. Thus, normally, P12 is held high.

Alternatively, it could be pulled high by using the selectable on chip pull up resistor on the controller. This would eliminate the need for an external component.

The other side of the switch is connected to Vss.

When the switch is pushed, P12 is grounded.

This is illustrated in *Figure 9*.

We have programmed the Cypress controller to give a GPIO interrupt on the Low-to-High transition of P12.

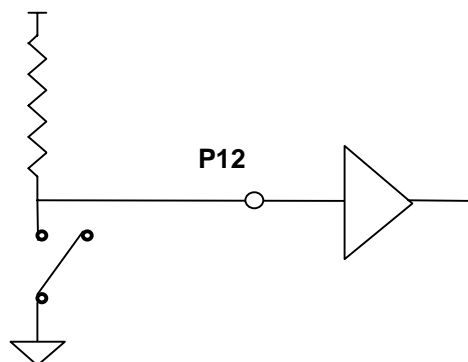
Any time this transition occurs, the GPIO interrupt routine sets a variable, *gbButtonDebounce*, to 100.

The 1024  $\mu$ s interrupt routine decrements this variable approximately each millisecond.

If the routine decrements the variable to zero, it checks to see if the port has returned to a high.

If P12 is now high, the routine determines that a valid button press and release has occurred.

If the port has not returned to a high state, the routine determines that the button has not yet been released or is bouncing and, resets the variable to 100. The process is then repeated.



**Figure 9 The Centigrade/Fahrenheit Button**

### The Cypress Watchdog Timer

The Cypress watchdog timer is not used in our application, but it deserves mention here.

This timer increments once each millisecond. If it reaches a count of eight, the Cypress USB Controller will be reset. It is cleared by any write to the Watchdog Timer Register at I/O address 21h.

The timer can be used to help determine if your device has gotten into an inappropriate state from which it cannot recover. An example would be if it were caught in a loop, unable to get out. After eight ms, the watchdog timer would time out and the device would be reset.

The watchdog timer cannot be disabled directly. However, placing a write to the Watchdog Timer Register at I/O address 21h in the 1024  $\mu$ s interrupt-handling loop will constantly clear the counter. This is an effective method of disabling the watchdog counter, since the only inappropriate event or condition that could cause it to reset the controller would be if the 1ms timer interrupt were disabled.

### The Windows Application

The Thermometer application queries the USB thermometer and displays the temperature. The application can display the temperature over time for the last 64 samples. The sample rate displayed varies from one sample per second to one sample every 30 hours.

#### Communicating with USB devices

A normal Windows application cannot communicate directly with a USB device. All communications with a device are through a USB device driver. This driver will be automatically loaded when a USB device is attached to the USB bus and automatically unloaded when the device is detached.

To communicate with a USB device, three Windows **APIs** are used: *OpenFile()*, *DeviceIoCommand()*, and *CloseHandle()*.

- **Starting communication with a USB device**

In order to communicate with a USB device, an application must first open a handle to its driver. This is done with the *OpenFile()* Windows API.

To use this API, you supply the name of the driver, and information about how you want to talk to the device (read, write, etc.).

If the device has been successfully attached to the USB, a device driver will have been automatically loaded, and the *OpenFile()* API will succeed and will return a handle to the driver. Otherwise, an error return will result.

## Cypress USB Thermometer Application Note

- Talking to the device

Once the application software has received a handle to the device driver, it can start communicating with the device with the DeviceIoCommand() API.

The device driver and various USB system drivers handle all actual communications with the device itself. The style of the communications is varied and is device and driver dependent.

In the case of the Thermometer driver, the application software simply passes messages to the driver and receives results back from the driver.

- Ending communication with a USB device

When the application is through communicating with the device, the handle to the device driver should be released with the CloseHandle() API.

- Example code:

### Visual Basic 5 Function Declares

```
Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName
    As String, ByVal dwDesiredAccess As Long, ByVal dwShareMode As Long,
    lpSecurityAttributes As SECURITY_ATTRIBUTES, ByVal dwCreationDisposition As
    Long, ByVal dwFlagsAndAttributes As Long, ByVal hTemplateFile As Long) As Long
```

```
Declare Function DeviceIoControl Lib "kernel32" (ByVal hDevice As Long, ByVal
    dwIoControlCode As Long, lpInBuffer As Any, ByVal nInBufferSize As Long,
    lpOutBuffer As Any, ByVal nOutBufferSize As Long, lpBytesReturned As Long,
    lpOverlapped As OVERLAPPED) As Long
```

```
Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
```

### Visual Basic 5 Sample Code

```
Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As Long
    bInheritHandle As Long
End Type
Type OVERLAPPED
    Internal As Long
    InternalHigh As Long
    offset As Long
    OffsetHigh As Long
    hEvent As Long
End Type
Public Security As SECURITY_ATTRIBUTES
Public gOverlapped As OVERLAPPED

Public hgDrvHnd As LONG
Public Const GENERIC_READ = &H80000000
Public Const GENERIC_WRITE = &H40000000
Public Const FILE_SHARE_WRITE = &H2
Public Const FILE_SHARE_READ = &H1
Public Const OPEN_EXISTING = &H3

Dim sFileName as STRING
Dim htemp As LONG
```



## Cypress USB Thermometer Application Note

```
Dim lIn as long, lInSize as long, lOut as long, lOutSize as long, lSize as long

sFileName = "\\.\Thermometer_0"

lIn = 11      ' Read Temperature Command
lSize = 0
lInSize = 2
lOutSize = 3
    ' Get a handle to the driver
    hgDrvrHnd = CreateFile(sFileName, GENERIC_WRITE Or GENERIC_READ,
        FILE_SHARE_WRITE Or FILE_SHARE_READ, Security, OPEN_EXISTING, 0, 0)
    ' Send the "GetTemperature Command"
    ltemp = DeviceIoControl(hgDrvrHnd, 4&, lIn, lInSize, lOut, lOutSize, lSize,
        gOverlapped)
    ' Close the Handle to the driver
    htemp = CloseHandle(hgDrvrHnd)
```

### The Cypress USB Thermometer Driver

Because the Cypress USB Thermometer does not fall into one of the currently supported classes of devices, Cypress wrote a vendor specific driver to accommodate the project. In the future, an official USB class will undoubtedly support a thermometer type of device, and the writing of a driver for this type of product would no longer be required. It is useful, in this application note, to illustrate communication with custom driver software. For many classes of devices, class drivers will be available with the operating system. However, for custom functionality or performance enhancement, a custom driver may be preferred over a generic "class driver."

See the "CY3640 User's Guide" for documentation on the calls to the driver using the DeviceIoControl() function from the above source code example.

## **General Hardware Design Considerations**

### **Connecting to the Bus**

#### Noise

Noise coupled to and from D+ and D- and Power lines can cause problems with both EMI and/or data errors.

D+ and D- should be kept away from signals that could couple noise to or from them such as clock lines or other high frequency signals. A good ground plane under these lines and/or shielding traces around them should be sufficient to minimize coupled noise to or from these lines.

Good bulk and high frequency bypassing techniques should be used where the USB Vbus connects to the USB cable. This is discussed below. If noise is a particular problem in your design or environment, ferrite beads can be placed on the power and ground lines between the bypass capacitors and the USB cable.

#### Cables and Connectors

Low-speed devices do not require shielded cables. However, some OEMs prefer the slight added cost of shielded cables to minimize the possibility of noise or EMI conformance problems.

The maximum cable length for a low speed device is 3 meters.

Care should be taken to insure that the wire gauge of the cable is chosen to be sufficiently large to insure that the minimum voltage (4.15Vdc) at the device is met under all conditions.

Only cables and connectors that are on the USB approved vendor list should be used.

### **Power Considerations**

#### Bypassing

A bulk bypass-capacitor should be used on Vbus. It should have a value of less than 10  $\mu\text{F}$  (per the USB Spec). This upper limit prevents large inrush currents during an attach event. This capacitor should be placed close to the power connection to the USB.

The bulk capacitor on the starter kit board, C1, value was chosen to be a 4.7  $\mu\text{F}$  tantalum capacitor.

Because the Cypress USB Controller uses so little current, it may take a while to discharge the bulk capacitor after a hot disconnect. This can be a problem if the device is plugged in again before the capacitor has discharged sufficiently to cause a power-on-reset.

A 50 KOhm resistor was placed between Vcc and Vss to bleed off the charge on the bulk bypass capacitor after a hot disconnect. This value allows the power to bleed off in about 1 second.

$50 \text{ KOhm} \times 4.7 \mu\text{F} \approx 250 \text{ ms}$  per time constant

5 time constants to discharge the capacitor =  $\sim 1 \text{ Sec}$

It also consumes only 100  $\mu\text{A}$  dc power, leaving 400  $\mu\text{A}$  for other device elements during Suspend. This allows the USB thermometer device to have more reliable and rapid unplug/plug operations.

A low ESL (Effective Series Inductance) capacitor should also be used near the USB connector to provide adequate high frequency bypass.

We chose a value of 0.1  $\mu\text{F}$  for C2.

If noise is a problem in your environment, ferrite beads can also be placed on both the Vcc and Vss lines of the USB connector for further isolation. Ferrite beads have the effect of filtering specific frequency noise while incurring an insignificant dc voltage drop.

Additional bypassing at individual circuit elements should follow standard guidelines for these elements.

### Testing the Thermometer (and Your Product)

#### ***OHCI*** and ***UHCI***

While there should be no user visible difference between using an ***OHCI*** and a ***UHCI*** device, we advise the designer to thoroughly check their product on each environment.

There may be aspects of the code that inadvertently depend on some sequence of events present in one type of controller and not in the other.

#### Multiple Language Environments

If your device includes support for multiple languages, ensure that you test it with each language for each operating system version. For example, Windows95 has versions specific for China, Japan, Germany, France, etc. The different language versions may behave in slightly different ways.

#### Voltage and Temperature Range

Insure that all logic is functional over the full voltage and temperature range covered by the USB spec. This is 0°C to 70°C and a bus voltage of 4.4Vdc to 5.25Vdc. In addition, the device must enumerate at a voltage of 4.15 Vdc.

#### Hot Connects and Disconnects

The device should be tested to ensure that it can be plugged and unplugged into the USB reliably under a variety of conditions and with both UHCI and OHCI hubs.

#### Startup and Shutdown

The device should be tested under both cold starts (no power applied) while connected to the hub and with warm starts (host restarted without being powered down).

#### OSR2.1 and Memphis

The device should be tested under both Windows 95 OSR2.1, Memphis (Windows 98), and any other supported OS to insure it functions properly.

### **Summary**

Low cost USB devices can be easily and rapidly designed using the Cypress USB Controller family of devices.

The Cypress USB Controllers handle most of the electrical issues you will encounter when designing a USB product. This includes USB I/O drivers and power handling for RESET and IDLE conditions.

The controller also automatically handles much of the low-level USB protocol issues for the designer. Additionally, most of the code to support the USB interface is supplied by Cypress to further ease the designer's task.

The architecture of the device provides many features that reduce the need to supply external logic to interface to other parts of the USB device.

These features include multiple I/O ports. These ports offer high and low current output drivers and individually selectable pull-up resistors, programmable output-buffer pull-down strength, the ability to wake-up the controller from a Suspend state, programmable interrupts on each pin, and more.

## Glossary

<b>ACK</b>	Acknowledgment. Handshake packet indicating a positive acknowledgment.
<b>Active Device</b>	A device that is powered and not in the suspend state.
<b>Application Programming Interface</b>	A defined interface to services provided by system software to an application.
<b>API</b>	See Application Programming Interface.
<b>Babble</b>	Unexpected bus activity that persists beyond a specified point in a frame.
<b>Bandwidth</b>	The amount of data transmitted per unit of time, typically bits per second (bps) or bytes per second (Bps).
<b>Basic IO System</b>	The code stored in ROM or EPROM, which provides the lowest level of interface to the computer and the basic configuration of the hardware. It is executed at startup.
<b>Big Endian</b>	A method of storing data that places the most significant byte of multiple byte values at a lower storage addresses. For example, a word stored in big endian format places the least significant byte at the higher address and the most significant byte at the lower address. See Little Endian.
<b>BIOS</b>	See Basic IO System.
<b>BIOS Setup Program</b>	A program which can be entered at startup to configure the low-level details of the hardware. This may include enabling or disabling USB hardware.
<b>Bit</b>	A unit of information used by digital computers. Represents the smallest piece of addressable memory within a computer. A bit expresses the choice between two possibilities and is typically represented by a logical one (1) or zero (0).
<b>Bit Stuffing</b>	Insertion of a “0” bit into a data stream to cause an electrical transition on the data wires allowing a PLL (Phase Locked Loop) to remain locked.
<b>Buffer</b>	Storage used to compensate for a difference in data rates or time of occurrence of events, when transmitting data from one device to another.
<b>Bus Enumeration</b>	Detecting and identifying Universal Serial Bus devices.
<b>Byte</b>	A data element that is eight bits in size.
<b>Capabilities</b>	Those attributes of a Universal Serial Bus device that a host can administrate.
<b>Cext</b>	A port on the Cypress USB Controller, which may be used the same as a GPIO port and as a special wakeup port for bringing a USB device out of a Suspended condition.
<b>Characteristics</b>	Those qualities of a Universal Serial Bus device that are unchangeable; for example, the device class is a device characteristic.
<b>Client</b>	Software resident on the host that interacts with host software to arrange data transfer between a function and the host. The client is often the data provider and consumer for transferred data.
<b>Configuring Software</b>	The host software responsible for configuring a Universal Serial Bus device. This may be a system configurator or software specific to the device.
<b>Control Pipe</b>	Same as a message pipe.

## Cypress USB Thermometer Application Note

<b>Control Transfer</b>	One of four Universal Serial Bus Transfer Types. Control transfers support configuration/command/status type communications between client and function.
<b>CRC</b>	See Cyclic Redundancy Check.
<b>Cyclic Redundancy Check</b>	A check performed on data to see if an error has occurred in transmitting, reading, or writing the data. The result of a CRC is typically stored or transmitted with the checked data. The stored or transmitted result is compared to a CRC calculated for the data to determine if an error has occurred.
<b>Default Address</b>	An address defined by the Universal Serial Bus Specification and used by a Universal Serial Bus device when it is first powered or reset. The default address is 00h.
<b>Default Pipe</b>	The message pipe created by Universal Serial Bus system software to pass control and status information between the host and a Universal Serial Bus device's Endpoint 0.
<b>Device</b>	<p>A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, device may refer to a single hardware component, as in a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a Universal Serial Bus interface device. At an even higher level, device may refer to the function performed by an entity attached to the Universal Serial Bus; for example, a data/FAX modem device. Devices may be physical, electrical, addressable, and logical.</p> <p>When used as a non-specific reference, a Universal Serial Bus device is either a hub or a function.</p>
<b>Device Address</b>	The address of a device on the Universal Serial Bus. The Device Address is the Default Address when the Universal Serial Bus device is first powered or reset. Hubs and functions are assigned a unique Device Address by Universal Serial Bus software.
<b>Device Descriptor</b>	A data structure with a defined format that USB devices use to report their attributes to the host.
<b>Device Driver</b>	Software which allows application software to talk to a hardware device without having to directly address it. Generally, it provides a logical interface to a physical configuration.
<b>Device Endpoint</b>	A uniquely identifiable portion of a Universal Serial Bus device that is the source or sink of information in a communication flow between the host and device.
<b>Device Enumeration</b>	See Bus Enumeration.
<b>Device Resources</b>	Resources provided by Universal Serial Bus devices, such as buffer space and endpoints. See Host Resources and Universal Serial Bus Resources.
<b>Device Software</b>	Software that is responsible for using a Universal Serial Bus device. This software may or may not also be responsible for configuring the device for use.
<b>Downstream</b>	The direction of data flow from the host or away from the host. A downstream port is the port on a hub electrically farthest from the host that generates downstream data traffic from the hub. Downstream ports receive upstream data traffic.
<b>Driver</b>	When referring to hardware, an I/O pad that drives an external load. When referring to software, a program responsible for interfacing to a hardware device; that is, a device driver.

## Cypress USB Thermometer Application Note

<b>DWORD</b>	Double word. A data element that is 2 words, 4 bytes, or 32 bits in size.
<b>Dynamic Insertion and Removal</b>	The ability to attach and remove devices while the host is in operation.
<b>End User</b>	The individual user of a host.
<b>Endpoint</b>	See Device Endpoint.
<b>Endpoint Address</b>	The combination of a Device Address and an Endpoint Number on a Universal Serial Bus device.
<b>Endpoint Number</b>	A unique pipe endpoint on a Universal Serial Bus device.
<b>Enumeration</b>	See Bus Enumeration
<b>EOP</b>	End of packet.
<b>FIFO</b>	A read/write memory element. On a FIFO read, data is presented in the same order that it was written and vice versa.
<b>Function</b>	A Universal Serial Bus device that provides a capability to the host. For example, an ISDN connection, a digital microphone, or speakers.
<b>General Purpose IO</b>	Read/write ports on the Cypress USB Controller family which can be used as required for general purpose by a designer of a USB device.
<b>GPIO</b>	See General Purpose IO.
<b>Handshake Packet</b>	A packet that acknowledges or rejects a specific condition. For examples, see ACK and NACK.
<b>HID</b>	See Human Interface Device.
<b>Host</b>	The host computer system where the Universal Serial Bus host controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.
<b>Host Controller</b>	The host's Universal Serial Bus interface.
<b>Host Controller Driver</b>	The Universal Serial Bus software layer that abstracts the host controller hardware. Host Controller Driver provides an SPI for interaction with a host controller. Host Controller Driver hides the specifics of the host controller hardware implementation.
<b>Host Resources</b>	Resources provided by the host, such as buffer space and interrupts. See Device Resources and Universal Serial Bus Resources.
<b>Hub</b>	A Universal Serial Bus device that provides additional connections to the Universal Serial Bus.
<b>Human Interface Device</b>	USB devices which are intended to be used primarily for human interface with the computer. Examples are: Keyboards, mice, joysticks, etc.
<b>Industry Standard Architecture</b>	The 8 and/or 16 bit expansion bus for IBM AT or XT compatible computers.
<b>In</b>	A transfer of information from the device to the host.
<b>Interrupt Request</b>	A hardware signal that allows a device to request attention from a host. The host typically invokes an interrupt service routine to handle the condition which caused the request.
<b>Interrupt Transfer</b>	One of four Universal Serial Bus Transfer Types. Interrupt transfer characteristics are small data, non periodic, low frequency, bounded latency, device initiated communication typically used to notify the host of device service needs.
<b>IRQ</b>	See Interrupt Request.
<b>ISA</b>	See Industry Standard Architecture.

## Cypress USB Thermometer Application Note

<b>Little Endian</b>	Method of storing data that places the least significant byte of multiple byte values at lower storage addresses. For example, a word stored in little endian format places the least significant byte at the lower address and the most significant byte at the next address. See Big Endian.
<b>LSB</b>	Least Significant Bit.
<b>Mbs</b>	Transmission rate expressed in megabits per second.
<b>MBs</b>	Transmission rate expressed in megabytes per second.
<b>Message Pipe</b>	A pipe that transfers data using a request/data/status paradigm. The data has an imposed structure which allows requests to be reliably identified and communicated.
<b>MSB</b>	Most Significant Bit.
<b>NACK</b>	Negative Acknowledgment. Handshake packet indicating that a device or endpoint is functional and will be able to respond in the future, but is not currently ready to respond.
<b>Non Return to Zero Invert</b>	A method of encoding serial data in which ones and zeroes are represented by opposite and alternating high and low voltages where there is no return to zero (reference) voltage between encoded bits. Eliminates the need for clock pulses.
<b>NRZI</b>	See Non Return to Zero Invert.
<b>Object</b>	Host software or data structure representing a Universal Serial Bus entity.
<b>OHCI</b>	Open Hardware Control Interface. Open Host Controller Interface. A specification for implementing a USB host controller. Sponsored by Microsoft and other companies.
<b>Out</b>	A transfer of information from the host to a device.
<b>Packet</b>	A bundle of data organized in a group for transmission. Packets typically contain three elements: control information (e.g., source, destination, and length), the data to be transferred, and error detection and correction bits.
<b>Packet Buffer</b>	The logical buffer used by a Universal Serial Bus device for sending or receiving a single packet. This determines the maximum packet size the device can send or receive.
<b>Packet ID</b>	A field in a Universal Serial Bus packet that indicates the type of packet, and by inference the format of the packet and the type of error detection applied to the packet.
<b>PCI</b>	See Peripheral Component Interconnect.
<b>Peripheral Component Interconnect</b>	A 32- or 64-bit, processor independent, expansion bus used on personal computers.
<b>Personal Computer Memory Card International Association</b>	The organization that standardizes and promotes PC Card technology.
<b>Phase</b>	A token, data, or handshake packet; a transaction has three phases.
<b>Physical Device</b>	A device that has a physical implementation; e.g., speakers, microphones, and CD players.
<b>PID</b>	See Packet ID.
<b>Pipe</b>	A logical abstraction representing the association between an endpoint on a device and software on the host. A pipe has several attributes; for example, a pipe may transfer data as streams (Stream Pipe) or messages (Message Pipe).
<b>Polling</b>	Asking multiple devices, one at a time, if they have any data to transmit.



## Cypress USB Thermometer Application Note

<b>POR</b>	See Power On Reset.
<b>Port</b>	Point of access to or from a system or circuit. For Universal Serial Bus, the point where a Universal Serial Bus device is attached.
<b>Power On Reset</b>	Restoring a storage device, register, or memory to a predetermined state when power is applied.
<b>Protocol</b>	A specific set of rules, procedures, or conventions relating to format and timing of data transmission between two devices.
<b>Request</b>	A request made to a Universal Serial Bus device contained within the data portion of a SETUP packet.
<b>Root Hub</b>	A Universal Serial Bus hub directly attached to the host controller. This hub is attached to the host; tier 0.
<b>Setup</b>	A packet that indicates that the transaction is a control transaction and what operation is expected of the device. It is only sent by the host.
<b>Root Port</b>	The upstream port on a hub.
<b>Service</b>	A procedure provided by an SPI.
<b>SPI</b>	See System Programming Interface.
<b>SRC</b>	See Sample Rate Conversion.
<b>Stage</b>	One part of the sequence composing a control transfer; i.e., the setup stage, the data stage, and the status stage.
<b>Stall</b>	A response on the USB that tells the host that the device responding is unable to complete the request and requires host intervention to proceed.
<b>System Programming Interface</b>	A defined interface to services provided by system software.
<b>Termination</b>	Passive components attached at the end of cables to prevent signals from being reflected or echoed.
<b>Time-out</b>	The detection of a lack of bus activity for some predetermined interval.
<b>Token Packet</b>	A type of packet that identifies what transaction is to be performed on the bus.
<b>Transaction</b>	The delivery of service to an endpoint; consists of a token packet, optional data packet, and optional handshake packet. Specific packets are allowed/required based on the transaction type.
<b>Transfer</b>	One or more bus transactions to move information between a software client and its function.
<b>Transfer Type</b>	Determines the characteristics of the data flow between a software client and its function. Four Transfer types are defined: control, interrupt, bulk, and isochronous.
<b>UHCI</b>	Universal Host Controller Interface. A specification for implementing a USB host controller. Sponsored by Intel.
<b>Universal Serial Bus</b>	A collection of Universal Serial Bus devices and the software and hardware that allow them to connect the capabilities provided by functions to the host.
<b>Universal Serial Bus Device</b>	Includes hubs and functions. See device.
<b>Universal Serial Bus Interface</b>	The hardware interface between the Universal Serial Bus cable and a Universal Serial Bus device. This includes the protocol engine required for all Universal Serial Bus devices to be able to receive and send packets.
<b>Universal Serial Bus Resources</b>	Resources provided by Universal Serial Bus, such as bandwidth and power. See Device Resources and Host Resources.
<b>Universal Serial Bus Software</b>	The host-based software responsible for managing the interactions between the host and the attached Universal Serial Bus devices.

## Cypress USB Thermometer Application Note

<b>USB</b>	See Universal Serial Bus.
<b>USBD</b>	See Universal Serial Bus Driver.
<b>Universal Serial Bus Driver</b>	The host resident software entity responsible for providing common services to clients that are manipulating one or more functions on one or more Host Controllers.
<b>Upstream</b>	The direction of data flow towards the host. An upstream port is the port on a device electrically closest to the host that generates upstream data traffic from the hub. Upstream ports receive downstream data traffic.
<b>Virtual Device</b>	A device that is represented by a software interface layer; e.g., a hard disk with its associated device driver and client software that makes it able to reproduce an audio .WAV file.
<b>Word</b>	A data element that is two bytes or 16 bits in size.

## **Links to Other USB Documents**

### **Datasheets:**

<a href="#">CY3650/CY3651</a>	USB Developer's Kit
<a href="#">CY7C63000/63001</a>	Universal Serial Bus Microcontroller
<a href="#">CY7C63100/63101</a>	Universal Serial Bus Microcontroller
<a href="#">CY7C63200/63201</a>	Universal Serial Bus Microcontroller
<a href="#">CY7C63410/63411</a>	Low Speed, High I/O 1.5 Mbps USB Controller
<a href="#">CY7C63412/63413</a>	Low Speed, High I/O 1.5 Mbps USB Controller
<a href="#">CY7C63510/63511</a>	Low Speed, High I/O 1.5 Mbps USB Controller
<a href="#">CY7C63512/63513</a>	Low Speed, High I/O 1.5 Mbps USB Controller
<a href="#">CY7C64011/64012/64013</a>	High Speed USB (12 Mbps) Peripheral Controller
<a href="#">CY7C64111/64112/64113</a>	High Speed USB (12 Mbps) Peripheral Controller
<a href="#">CY7C65013/65113</a>	4/8 Downstream Port USB Hub
<a href="#">CY7C66011/66012/66013</a>	High Speed USB (12 Mbps) Controller with Hub
<a href="#">CY7C66111/66112/66113</a>	High Speed USB (12 Mbps) Controller with Hub

### **Application Notes:**

[Designing a Low-Cost USB Mouse with the Cypress Semiconductor CY7C63000 USB Controller](#)

[Designing a Low-Cost Analog USB Joystick with the Cypress CY7C63200 USB Microcontroller](#)

### **USB Specification:**

[USB Specification](#)