Cypress CY3640

USB Starter Kit

User’s Guide

A. Contents of the CY3640 USB Starter Kit

1. Three CY7C63001 Cypress USB Controller devices (One pre-programmed windowed controller on the CY3640 USB Thermometer Demo board, and two spare devices (one windowed and one OTP)
2. One Cypress USB Programmer from HI-LO Systems with a wall power adapter, a serial cable and programming software on a floppy disk
3. One CY3640 USB Starter Kit printed circuit board
4. One low-speed unshielded USB "A to B" Cable
5. One Cypress USB Starter Kit CD-ROM:
    - Software:
        - CYASM: Cypress USB controller assembler
        - USB Thermometer device assembly source code
        - USB Thermometer Windows application program executable code
        - USB Thermometer device driver
    - Documentation:
        - Cypress USB controller family datasheets
        - USB Thermometer User's Guide and Application Note: Designing a USB Thermometer with the CY7C63001 USB Controller
        - USB Thermometer PC board layout and schematics
        - USB Specification v1.0
        - Cypress CYASM assembler manual
        - Cypress USB-related application notes
        - Cypress databook
6. Printed documentation:
    - USB Starter Kit User's Guide
    - USB Starter Kit Application Note
    - Registration card – Please fill it out and drop it in the mail

B. Getting started

Starting to use your Cypress USB thermometer is easy. Just follow these simple steps:

1. **Insert the Cypress USB CD-ROM**

   The Cypress CD-ROM contains the Cypress USB Thermometer application and driver files you will need. Place it in the CD-ROM drive you use to install your software.

2. **Run SETUP.EXE from the CD-ROM**

   This will install the Cypress USB Thermometer Windows application.

   The Cypress USB Thermometer application is the user interface program which displays the temperature measurement received from the Cypress USB Thermometer device on the USB Starter Kit PC board.

   The SETUP.EXE is located in the root directory of the Starter Kit CD-ROM. After you have run SETUP.EXE, the Cypress USB Thermometer Windows application will be properly installed in the default directory C:\Program Files\Thermometer or a directory selected by you.

   The setup program also installs an entry in the Programs section of your Start menu.

   The Cypress USB Thermometer application may be uninstalled by selecting it in the "Add/Remove Programs" section of the "Control Panel."

3. **Plug the A-end (flat connector) of the USB cable into the USB port of your PC**

4. **Plug the B-end (square connector) of the USB cable into the B-receptacle of the CY3640 USB Starter Kit**

   The computer will notify you that it has found new hardware and is looking for a driver. If it does not find a driver, it will then ask you to supply a driver.

5. **Insert the Cypress USB CD-ROM (if prompted to do so)**

   If your computer prompts you to supply a driver, make sure that your Cypress USB CD-ROM is inserted into your computer's CD-ROM drive.

   Your computer will now automatically install the Cypress USB driver from the CD-ROM.

6. **Run the Cypress USB Thermometer application**

   When you want to run the Cypress USB Thermometer application, use the "Programs" menu and select the Cypress USB Thermometer entry. The computer will display a graph showing the temperature reading over time. Use the options described in the following sections to control various aspects of the USB thermometer.

   NOTE: if you cannot get the thermometer to work, you may not be using the correct operating system. See *section K* for more details.

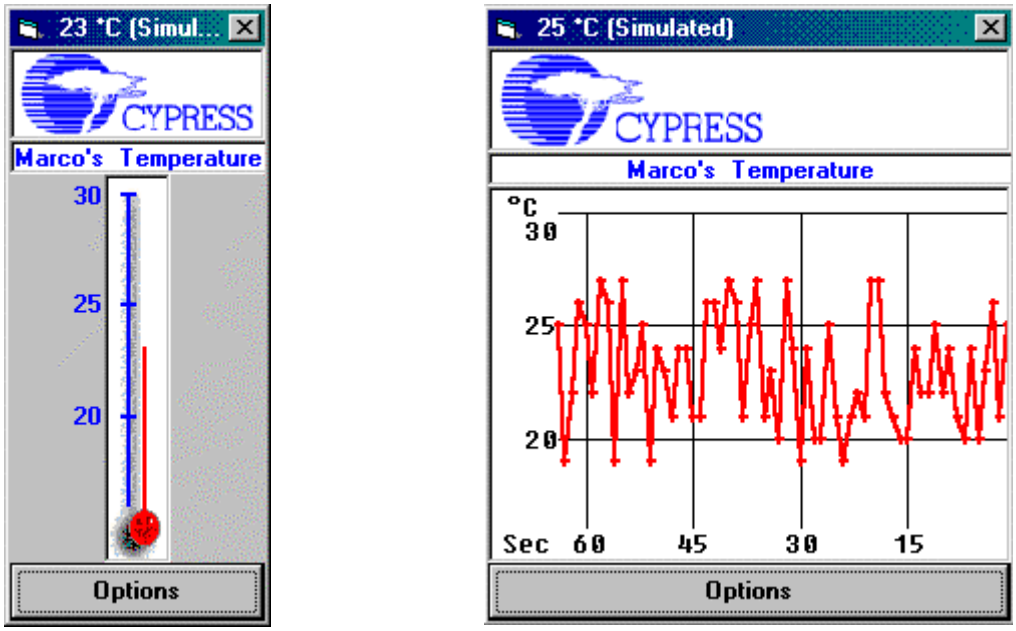
C. Uninstalling the Cypress USB Thermometer application

If you want to remove the Cypress USB Thermometer application from your system, it may be uninstalled by selecting Cypress USB Thermometer in the "Add/Remove Programs" section of the "Control Panel."

D. The Cypress USB Thermometer Application Options

Changing the display style from conventional thermometer display to a history of temperature

The Thermometer application will display either the current temperature using a conventional thermometer symbol or a history of the temperature recorded during the last 64 sample periods (See *Figure C1*).



Conventional Thermometer Display

Temperature History Display

Figure C1

To change the display style, click on the "Options" box in either the conventional thermometer display or the temperature history display. Checking the "Show Time Line" box will select the temperature history display. "Unchecking" the "Show Time Line" box will select the conventional thermometer display. Click on the "OK" box to complete your selection. (see *Figure C2*).

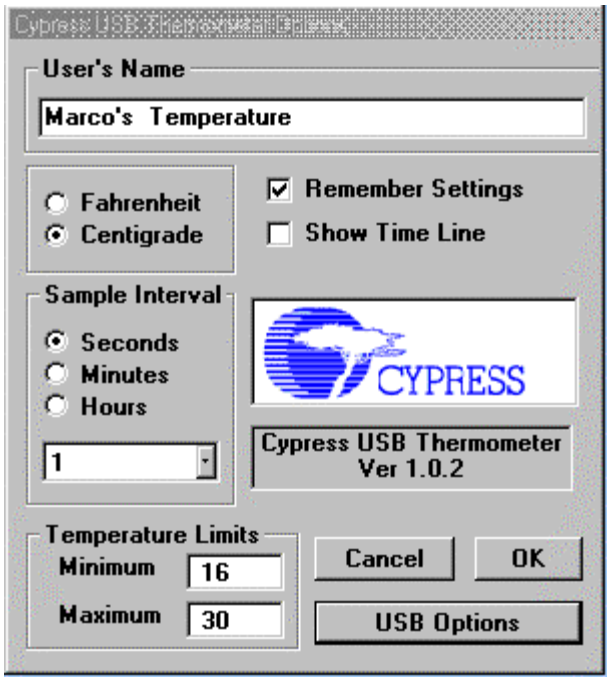


Figure C2

Changing the maximum and minimum temperature value displayed

The Thermometer application can display temperatures between 0ºC and 70ºC. (32ºF and 158ºF)

The user may set the temperature range display on the Options screen by entering the desired value in the appropriate "Temperature Limits" text box.

Changing the user's name

The user may change the heading, which appears at the top of the temperature display.

This is done on the Options screen by changing the text in the "User's Name" text box.

Changing the display mode from Centigrade to Fahrenheit

The Thermometer application can display temperature in either Centigrade or Fahrenheit.

The display mode is selected in the Options screen by selecting either the Fahrenheit or the Centigrade radio buttons box.

Changing the time between temperature sample periods

The Thermometer application can sample the temperature at a variable rate determined by the user.

The user sets the sample rate on the Options screen by selecting the appropriate options in the "Sample Interval" area. Sample intervals may be chosen between one per second to one every 30 hours.

Simulating temperature measurements

The Temperature display program allows you to simulate temperature readings instead of actually measuring them.

You can select this by clicking on the "USB Options" box. The "USB Options" screen will be displayed. (See *Figure C3*)

Checking the "Simulate Temperature" box will cause the thermometer to enter a test mode in which a series of random temperature readings will be displayed.

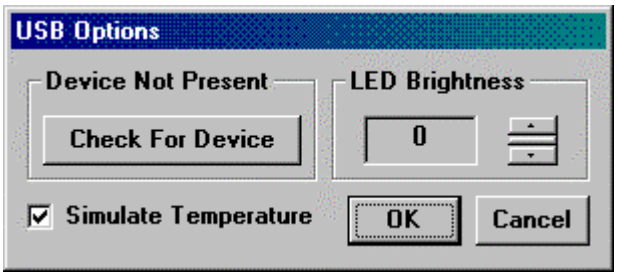


Figure C3

Testing to see if a Cypress USB Thermometer device is present

If your Cypress USB Thermometer device is plugged in to the USB and you cannot get temperature measurements, you may try to connect to it by clicking the "Check For Device" button on the USB Options screen (See *Figure C3*). This screen is available by clicking the "USB Options" button on the Options screen.

The text above the button will indicate whether the Thermometer application is connected to the Cypress USB Thermometer device through the device driver. If it is, the "Check For Device" button will be disabled.

Changing the brightness of the Enumerated LED on the Cypress USB Thermometer device

The brightness of the Enumeration LED on the Cypress USB Thermometer PC board may be set to one of sixteen levels.

The user can set the brightness level desired on the "USB Options" screen by changing the value of the "LED Brightness" box (See *Figure C3*). This screen is available by clicking the "USB Options" button on the Options screen.

The values may be set from zero to fifteen, with zero being the lowest setting.

Saving the current configuration of the Thermometer application

The user can choose to automatically save the current configuration of the Thermometer application when they exit the program by checking the "Remember Settings" box. The parameters saved include:

- Location on the desktop where the application screen will reside
- User Name text
- Centigrade or Fahrenheit mode
- Temperature history or conventional thermometer display selection
- Maximum and Minimum displayed temperature
- Temperature sample rate
- Simulate temperature selection status

E. About the Cypress CY3640 USB Starter Kit

Overview

The Cypress CY3640 USB Starter Kit includes a fully functional USB thermometer device, a Windows95 thermometer application, and a USB Thermometer device driver. Together, these components allow the user to measure and display temperatures between 0ºC and 70ºC to an accuracy of ±1ºC.

In addition to being a useful USB device, Cypress has designed the USB Starter Kit to serve as an easily customizable platform for USB device development using the Cypress CY7C63X0X family of USB controllers.

Features

The CY3640 has the following features:

The Cypress USB Controller is socketed in a 20-pin socket

> It may be removed, reprogrammed, and replaced for easy development of new assembly code.

Cypress has supplied a bread board area (sea of holes)

> This area will accept wire wrap pins and wire wrap sockets. This area will make it easy to develop additional logic on-board.

A connector for use with standard 40-pin flat-cable connectors has been provided.

> All signals from the Cypress USB Controller, which are useful for the development of external logic, are available at pins on the connector. The signal names are labeled on the top layer of the board. Pin numbers on the connector are labeled on both the top and bottom layers of the board to make it easy to wire to the connector.
>
> An identical set of signal vias is placed adjacent to the connector for easy wiring to the signals.

Connections to external power source

> There is a location for an external power supply connector, but the connector is not supplied with the kit. Individual labeled locations for Vss, USB Vbus (the positive supply of the USB), and External Vcc are provided on either side of the sea of holes.

LED indicators (Power and Enumeration)

> The CY3640 includes two LEDs: a red and a green one. As configured when the CY3640 is shipped from the factory, the LEDs are used to indicate that power is applied and that the USB host has enumerated the Cypress USB Thermometer device, respectively.
>
> The anode of each LED is connected to the USB Vbus (positive supply) through a current limiting resistor. The cathode of each device is connected to three places to allow you to reconfigure them for your own purposes: a jumper, a pin on the 40-pin connector, and a hole for installing a wire wrap pin.
>
> The Power LED
>
> The Power LED (as shipped from Cypress) is used to indicate that power (Vbus) from the USB has been applied. It is connected through a jumper (JP1) and a current limiting resistor (R4) directly between Vbus and Vss, and will be lit whenever power is applied through the USB. The user may remove JP1 if desired.

The cathode of this LED is also connected to a pin on the 40-pin connector (pin 8) and to an adjacent hole that may be used to install a wire wrap pin. This allows the user to remove the factory-installed jumper to Vss and use the LED for their own purposes.

Note: Although it may be convenient to know when the USB is applying power, during development. The fact that JP1 is left connected to Vss makes the device not strictly compatible with the USB specification, because it will draw approximately 20 mA whenever it is connected. The USB specification limits power prior to the device being configured to 500 µA.

The Enumeration LED

The Enumeration LED (as shipped from Cypress) is used to indicate that the USB Thermometer device has been enumerated. The cathode of the LED is connected to P13 through a jumper (JP2) which controls its path to Vss.

The cathode of this LED is also connected to a pin on the 40-pin connector (pin 10) and to an adjacent hole that may be used to install a wire wrap pin. This allows the user to remove the jumper to P13 and use the LED for their own purposes.

The switch (SW1)

SW1 (as shipped) on the Cypress USB Thermometer is used to select whether the Thermometer Windows application displays the temperature in Centigrade or Fahrenheit. It is a momentary SPST.

One pole of the switch is connected to Vss. The other pole is connected to Vcc (Vbus) through R4, to a pin on the 40-pin connector (pin 6), to an adjacent hole that may be used for installing a wire wrap pin, as well as to the GPIO P12 of the USB controller through a jumper JP3. This allows the user to remove the jumper to P12 and use SW1 for their own purposes.

The breadboard area

Sea of holes

Cypress has provided an area that will accept wire wrap pins and wire wrap sockets for development of logic and functionality on the board.

USB Vbus (Vcc) and Vss connections

We have provided locations for connecting power and ground from the USB to your bread board area. These locations will accept wire wrap pins.

External Vcc connections

If you need to power your breadboard with an external supply, a connector site (P1) is supplied for that purpose. Two capacitor locations are also provided adjacent to the connector for bypass. C6 is a bulk bypass and C7 is for high frequency. These capacitors are not populated with the CY3640 as shipped, and you should use components appropriate to your needs.

The External Vcc supply is brought to the bread board area and connected to three locations labeled (oddly enough) Ext. Vcc. You may insert wire wrap pins in these locations to route to your breadboard area.

The temperature sensor (U2)

The temperature sensor device is socketed so it can be removed if you need to use the associated pins on the Cypress USB Controller for other logic on your breadboard.

Ferrite bead locations

If noise is a problem in your environment, locations for two ferrite beads (FB1 and FB2) have been provided: one for the Vcc supply from the USB and one for the Ground return to the

USB. These beads should not be necessarily and are provided for exceptional noisy environments.

The locations are shorted by a trace on the bottom layer of the PC board. If you desire to use ferrite beads, you should cut the traces and install beads suitable to your needs.

Other items of interest

Bulk capacitor bleeder resistor (R5)

A resistor is provided to bleed off the charge stored on the bulk capacitor. This is provided to insure that charge is removed from the bulk capacitor and hence the board logic, within a short period after the device is unplugged from the USB.

USB connector

A "B" receptacle has been provided on the board so a detachable A/B cable can be used with the device.

A footprint for an in-line connector (P2) to a permanently attached USB cable has been provided if you need to have a non-removable USB cable.

F. Changing the functionality of the Cypress CY3640 USB Starter Kit

The CY3640 USB Starter Kit is designed to allow you to add or change its functionality in a variety of ways in order to meet your needs.
You may easily:

- Add logic to the board itself
- Change the functionality of the on-board LEDs and the switch
- Change the program stored inside the Cypress USB Controller
- Use an external power source
- Take signals off of or bring signals onto the board

Changing the program in your device

You can write your own code for your Cypress USB Controller.

The easiest way is to use the code provided as a base and change only those parts which are specific to your product.
The routines you will need to focus your changes on are in the file "USB.ASM". They are

- Main
- SetConfig
- 1024usec IRQ handling
- Vendor Specific Setup Commands

Assembling the code for your device

You can assemble your code for the Cypress USB Controller with CYASM, Cypress' assembler.

See the Cypress CYASM documentation. Both CYASM and its documentation are included in the USB Starter Kit CD-ROM.

Programming your device

After you have written and assembled your own USB controller firmware code, you can program a new USB controller using the device programmer contained in the Cypress USB Starter Kit. Please note that the device programmer only supports Cypress CY7C63X0X family of low-speed USB controllers, namely, CY7C63000, CY7C63001, CY7C63100, CY7C63101, CY7C63200 and CY7C63201. Its components include:

- a device programmer with a 32-pin DIP adapter
- a floppy disk with the programming software
- a 9-pin serial cable
- a wall power adapter

To program a new USB controller simply follow these steps:

- Connect the device programmer to your PC's serial port using the serial cable
- Turn it on by plugging the wall power adapter
- Copy the programming software executable on the floppy disk to your PC's hard disk and run it. At startup the software will detect the presence of the programmer connected to the serial port and perform self-test
- Insert your Cypress USB controller into the DIP adapter and choose the appropriate commands from the programmer software

The programming software is actually quite simple; however, it provides all the necessary functions to program a USB controller, such as blank check, read, program, verify and security fuse programming. The software is able to read and write .BIN and .HEX format files. To erase the code programmed in the windowed USB controller devices, use regular UV EPROM erasers available on the market.

Adding your own logic

See "Appendix D" for a description of the items that you may change to modify your design.

G. Cypress CY3640 USB Starter Kit Schematic

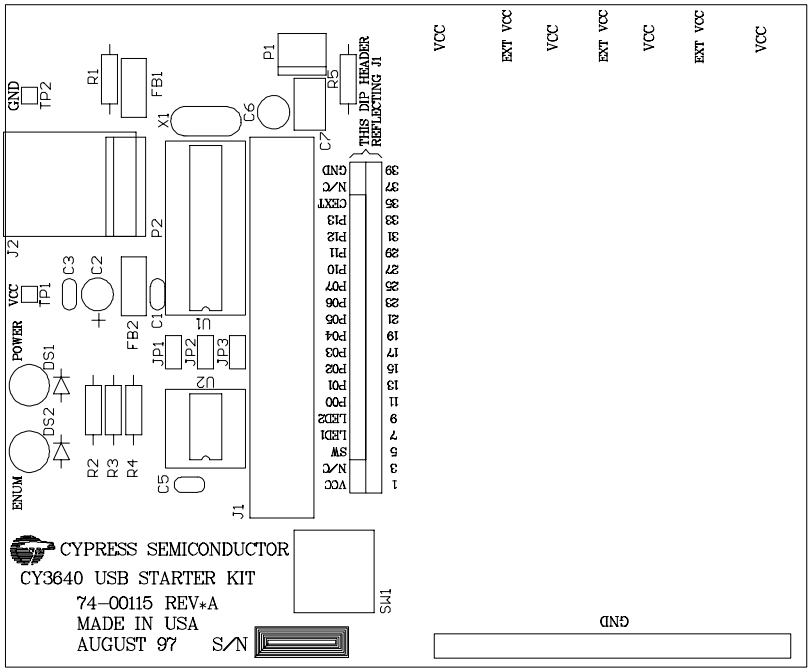
Bill of Materials

U1	Cypress CY7C63001 USB controller
U2	Dallas Semiconductor DS1623 temperature sensor
P1	2-pin header, polarized, 0.1" center, right angle. Not populated.
P2	5-pin header, polarized, 0.1" center, vertical. Not populated.
SW1	SPST, Momentary switch
DS1	20 mA, 2.2 Vdc Red LED, 0.1" center, 0.02 leads
DS2	20 mA, 2.2 Vdc Green LED, 0.1" center, 0.02 leads
X1	6 MHz ceramic resonator
JP1	2-pin header, 0.1" center, with shunt
JP2	2-pin header, 0.1" center, with shunt
JP3	3-pin header, 0.1" center, with shunt
R1	7.5K Ohm, 1%, 1/8 W, carbon film resistor
R2	150 Ohm, 10%, 1/8 W, carbon film resistor
R3	150 Ohm, 10%, 1/8 W, carbon film resistor
R4	10K Ohm, 10%, 1/8 W, carbon film resistor
R5	50K Ohm, 10%, 1/8 W, carbon film resistor
C1	0.1 µF, 5%, Low ESL, 50 Vdc capacitor
C2	4.7 µF, 10%, Tantalum, 25 Vdc capacitor
C3	0.1 µF, 5%, Low ESL, 50 Vdc capacitor
C5	0.1 µF, 5%, Low ESL, 50 Vdc capacitor
C6	External Vcc Bulk capacitor. Not populated
C7	External Vcc HF capacitor. Not populated
J1	40-pin, boxed header, straight, 0.1" x 0.1" center
J2	USB "B" connector
FB1	Ferrite bead. Not populated
FB2	Ferrite bead. Not populated

The USB Starter Kit demo board layout and schematic are shown on the next two pages.

CYPRESS PROPRIETARY

REVISION BLOCK		
REV	DESCRIPTION	DATE
*A	TOP SILKSCREEN REVISED	08/21/97



74-00115 | REV *A | TOP SILKSCREEN | GXT | AUGUST 1997

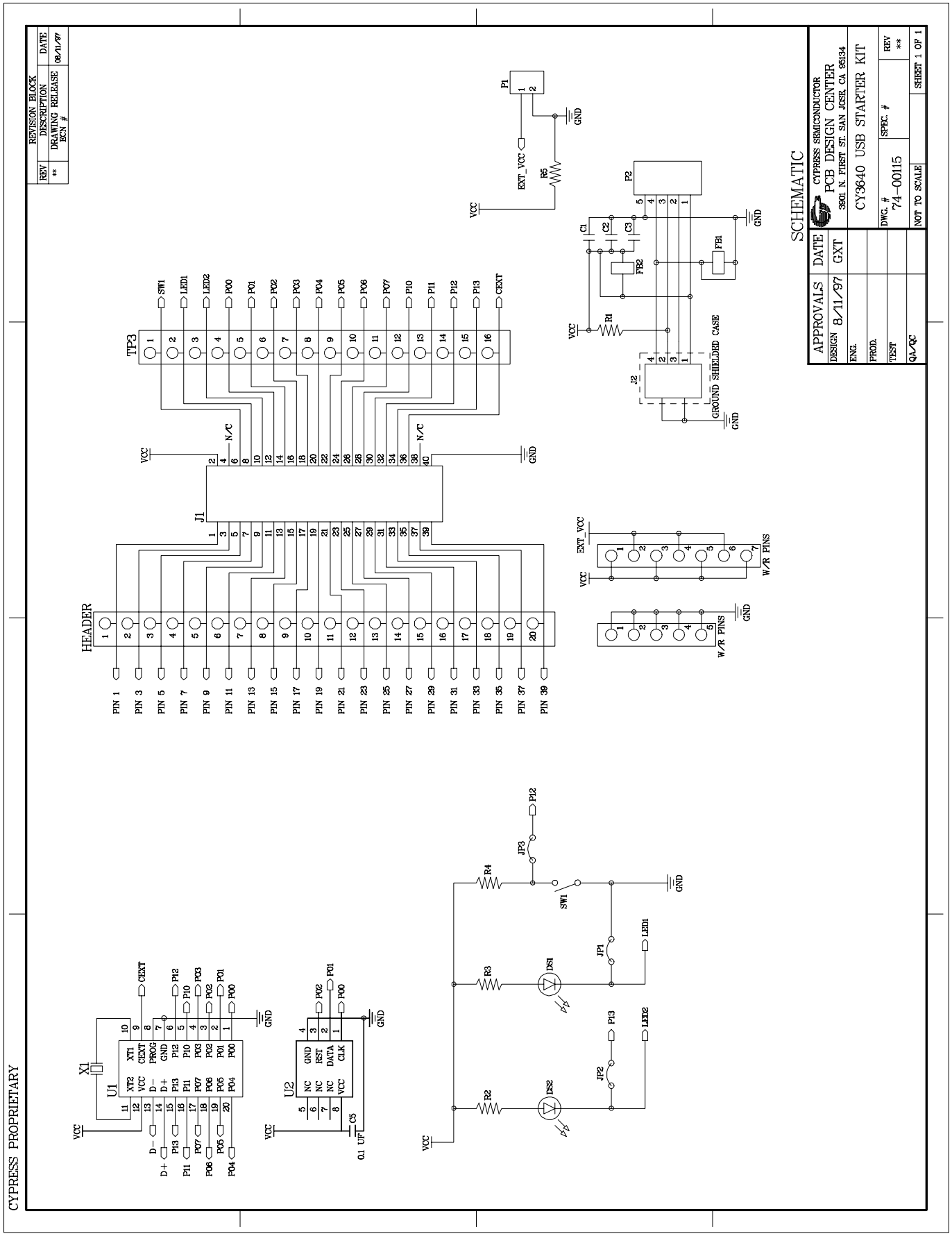
APPROVALS	DATE
DESIGN 8/1/97	GXT
ENG.	
PROD.	
TEST	
QA/QC	

CYPRESS SEMICONDUCTOR
PCB DESIGN CENTER
3901 N. FIRST ST. SAN JOSE, CA 95134

CY3640 USB STARTER KIT

DWG. #	SPEC. #	REV
74-00115		*A

NOT TO SCALE	74-00115.GTO	SHEET 1 OF 1



H. Cypress USB Controller micro code (Assembly)

USB.ASM

```
;; USB_20.ASM
;*****************************************************************************
;
; ***************************************************************************
;
; Target:
;       Cypress 7C63000 8bit RISC microcontroller with 1.5Mbps USB serial interface
;       Dallas 1623: High Resolution Temperature Measurement Sensor
; Overview
;  There are four main sub-systems: USB, Thermometer, LED, and Button.
;   The system is started in the main() routine at reset. This routine
;   initializes the USB variables, the IO ports, the Thermometer logic,
;   and the data space. All USB communication occurs on an interrupt basis.
;  First, Main() loops waiting for a USB reset.
;  After receiving a USB reset, Main() enables the Endpoint 0 interrupt and
;   loops waiting for Setups which ultimately will the result in the device
;   being enumerated.
;  Once the device has been enumerated on the USB, the main loop waits 10ms,
;   polls the thermometer, updates the LED, and initializes end point 1 if
;   appropriate.
; USB
;  Endpoint 0 is used to support Control Transfers and vendor specific
;   requests. End point 1 is also available for interrupt requests handling
;   small packets of data (good for mouse, joystick, keyboard, thermometer, etc.).
;   However, it is not used in this code.
;  Each control transfer interrupts the processor and the subsequent routines
;   services it.
; Thermometer
;  A simple 9-bit temperature value is read from the thermometer every 10ms. At
;   startup, the thermometer is initialized and placed into a continuous mode
;   storing internally the current temperature. Thereafter, the temperature is
;   read synchronously and returned into the USB end point one FIFO buffer.
; LED
;  The LED is controlled by P13. When P13 goes low, the LED is turned on.
;   The LED indicates the status of the USB connection. Once this device has
;   "logically" been enumerated and configured to run on the serial bus, the LED
;   is illuminated. The LED supports adjusting the brightness intensity by first
;   setting the new brightens value (default: FFh = High) and then setting the
;   brightness update field.
; Button
;  A momentary push button is used to indicate that the application's
;   Celsius/Fahrenheit display mode should be toggled.
;  With each GetTemperature request, a value is sent indicating whether the
;   button has been pushed.
;  The GPIO interrupt is triggered by pushing the button causing its
;   level to change from High to Low. A 100ms debounce was added to control the
;   erroneous re-occurrence of this logical state change for a period. The
;   1024ms timer decrements the debounce to zero, re-enabling the button if at the
;   end of the time out it has returned High.
;
; Port Usage
;  P0.0 - Thermometer Data  (input/output)
;    .1 - Thermometer Clock (output)
;    .2 - Thermometer Reset (output)
;    .3 -
;    .4 -
;    .5 -
;    .6 -
;    .7 -
;  P1.0 -
;    .1 -
```

```
;    .2 - Button (0=pushed) (input)
;    .3 - LED (0=on)        (output)
;
;-------------------------------------------------------------------------------

;//$PAGE
; Directives
FillROM 0

; Microprocessor definitions
include "63x0x.inc"

;-----------------------------------------------
; Data Segment (RAM)
;-----------------------------------------------

; Program Stack
gbSysProgramStack       :equ 00h    ; [00h-1Fh] Stack 0x20h
gbSysDataStack          :equ 50h    ; [50h-6Fh] Stack 0x20h
gbSysFIFO               :equ 70h    ; [70h-7Fh] EP0 and EP1 FIFO's

; Global Interrupt
gbSysInterruptMask      :equ 20h    ; Holds the current interrupt mask

; System tickers
gbSysTick1024us         :equ 22h    ; # of 1mSec ticks
gbSysTick1024usRoll     :equ 24h    ; # of 256mSec ticks

; USB management data
gbUSBValidRqsts         :equ 25h    ; Count of USB recognized requests
                                    ;   Used during debug
gbUSBSendSequence       :equ 26h    ; Buffer send data 0/1 line
gbUSBSendBytes          :equ 27h    ; Buffer bytes left to send
gbUSBSendBuffer         :equ 28h    ; Offset into current buffer

gbSuspendCount          :equ 30h    ; # of msec bus has been IDLE

; General
gbSysEnumerated         :equ 29h    ; Device is enumerated

; LED management
gbLEDBrightnessUpdate   :equ 2Bh    ; Semaphore to reset brightness
gbLEDBrightness         :equ 2Ch    ; Current brightness
LED_ON                  :equ 08h    ; P13 is used to indicate Enumeration

; Button management
gbButtonDebounce        :equ 2Dh    ; Debounce count down value
gbButtonPushed          :equ 7Ah    ; USBEndP1FIFO +2 (toggles if button was clicked)
Button_Pin              :equ 04h    ; Pin the switch is on, P12



;//$PAGE
;-----------------------------------------------
; Code Segment (ROM)
;-----------------------------------------------

; Vector Table
org 00h
    jmp main                ; Reset of some type
    jmp SysUnUsed           ; 128us timer (not used)
    jmp SysTimer1024usEvent ; 1024us timer
    jmp USBEndPoint0Event   ; EP0
    jmp SysUnUsed           ; EP1 (not used)
    jmp SysUnUsed           ; Reserved
    jmp SysGPIOEvent        ; Button
    jmp SysUnUsed           ; CExt (not used)

;-----------------------------------------------
; Unused event
; Do nothing, restore machine to prior state
```

```
;***********************************************
SysUnUsed:
    push  a
    mov   a,[gbSysInterruptMask]
    ipret SysInterrupt

;//$PAGE
;*****************************************************************************
; main()
; @func Entry point after PowerOn, WatchDog timeout or WakeUp from sleeping.
; @comm Never returns
;*****************************************************************************
main:
    ; This portion of Main is only executed after a RESET (Power-On or USB)
    ; Setup data stack in high order RAM, just below EP0 FIFO
    ; It will grow down from here
    mov  a,70h      ; USBEndP0FIFO
    swap a,dsp

    ; Initialize both Ports high
    mov  a,FFh
    iowr SysPort0   ; Port 0 Data reg
    iowr SysPort1   ; Port 1 Data reg
                    ;   1 on P13 is needed to make sure enumerate LED is off
                    ;   1 on any port that needs to be an input
    ; Enable Pullups (0=enable)
    mov  a,0
    iowr SysPort0PullUp
    mov  a,Button_Pin
    iowr SysPort1PullUp     ; 1 on P12 is needed to make sure GPIO interrupt
                            ;   occurs on LOW to HIGH transistion. This
                            ;   disables it's pull up

    ; Enable or disable interrupts on appropriate pins
    mov  a,0
    iowr SysPort0IntEnable     ; All pins irq's are disabled on Port 0
    mov  a,Button_Pin
    iowr SysPort1IntEnable     ; Enable P12, the button pin.
                               ; No interrupts will occur until the device
                               ;   is enumerated. Then GPIO's will be enabled and
                               ;   we will allow P12 to generate interrupts

    ; Initialize timers
    mov  a,0
    mov  [gbSysTick1024us],a
    mov  [gbSysTick1024usRoll],a

    ; Initialize USB variables
    mov  a,0
    mov  [gbUSBValidRqsts],a    ; No valid requests yet
    mov  [gbUSBSendSequence],a  ; Start with a 0
    mov  [gbSysEnumerated],a    ; Not enumerated

    ; Initialize LED
    mov  a,1     ; flag it for an update
    mov  [gbLEDBrightnessUpdate],a
    mov  a,FFh   ; set for maximum brightness
    mov  [gbLEDBrightness],a

    ; Initialized Button
    mov  a,0
    mov  [gbButtonPushed],a     ; Initial state of 0, no button pushed

    ; Initialize variables
    mov a,0
    mov [gbUSBSendBytes],a      ; No bytes to send in FIFO buffers
    mov [gbSuspendCount],a      ; Reset bus activity to 0
    mov [gbButtonDebounce],a    ; We are not debouncing

    ; Set interrupt mask
    mov  a,SysIntTimer1024us  | SysIntUSBEndP0
```

```
    mov  [gbSysInterruptMask],a

    ;******************************************
  MainLoop:
    ; Enable interrupts to current mask
    mov  a,[gbSysInterruptMask]
    iowr SysInterrupt

    ;************************
    ; do nothing until we are enumerated
    mov  a,0
    cmp  a,[gbSysEnumerated]
    jz   MainLoop                   ; Not enumerated, loop


    ; Ah! We're enumerated, lets do the rest of the loop
    ;************************

    ; Write a 0 to the LED on P13 to turn it on
    mov a,~(LED_ON)
    iowr SysPort1

    ; Wait 10 milliseconds
    mov  a,10
    call SysDelayMS

    ;************************
    ; Read temperature
    call ThermReadTemperature

    ;************************
    ; Update brightness?
    ; mov  a,0
    cmp  a,[gbLEDBrightnessUpdate]
    jz   MainLoopNoLEDUpdate        ; No, branch

    ; Yes, update the LED brightness
    ; Reset the LED update flag
    ; mov a,0h
    mov  [gbLEDBrightnessUpdate],a

    ; Set new brightness
    mov  a,[gbLEDBrightness]
    iowr SysPort1ISinkPin3

    ; Fall through to here in any case
  MainLoopNoLEDUpdate:

    ; Loop
    jmp  MainLoop
    ;******************************************

    halt
    ; Oops! We should never get here


;*****************************************************
; SysTimer1024usEvent()
; @func Timer interrupt event ocurring every 1.024 mSec
;       using 6Mhz crystal.
;*****************************************************
SysTimer1024usEvent:

    ; Save accumulator
    push a

    ; Clear watchdog timer
    ; Clearing it here effectively disables the timer
    iowr SysWatchDog

    ; Keep track of length of any IDLE conditions (No bus activity)
```

```
  iord USBControl                   ; Read the USB Status and Control Reg
  and a,01h                         ; Check bit 0
  cmp a,0h
  jz Inc_Counter                    ; Hmm! No activity. Branch and keep track of it.
  iord USBControl                   ; Ah! There was activity,
                                    ;  clear the bus activity bit
  and a,0feh
  iowr USBControl
  mov a,0                           ; Clear the suspend counter
  mov [gbSuspendCount],a
  jmp Suspend_End

Inc_Counter:                        ; Monitor the IDLE count
  mov a,[gbSuspendCount]            ; Get # of mSec we have been IDLE
  inc a                             ; Increment the count
  mov [gbSuspendCount],a
  cmp a,03h                         ; Has it been 3msec yet?
  jnz Suspend_End                   ; Not yet, branch
  mov a,0h                          ; Yes, clear the suspend counter
  mov [gbSuspendCount],a
  iord SysStatus
  or a,08h                          ; Set the suspend bit to cause a suspend
  iowr SysStatus                    ; We will enter the suspend state during
                                    ;   the next instruction.

Suspend_End:
 ; Increment the 1024 usec counter and check for rollover
  inc  [gbSysTick1024us]
  jnz  STimerNo1024usRoll     ; No

  ; Clear rollover
  mov  a,0
  mov  [gbSysTick1024usRoll],a

STimerNo1024usRoll:
  ; Are we counting down a button debounce
  mov  a,0
  cmp  a,[gbButtonDebounce]
  jz   STimerNoDebounce       ; Not debouncing, branch

  ; Yes, we're debouncing. Let's see if we are timed out.
  dec  [gbButtonDebounce]
  mov  a,0
  cmp a,[gbButtonDebounce]

  ; has debounce timed out?
  jnz STimerNoDebounce        ; No, still debouncing, branch.

  ; The debounce timer has timed out
  ; check if the button pin is at a 1. If not, the button is either still
  ;    bouncing or still pushed
  iord SysPort1              ; check the port the button is on
  and a,Button_Pin           ; check the pin
  jnz STimerDebounceOver     ; branch if it is not pushed ; mrr

  ; Reset debounce since the button is not yet released or is bouncing
  mov  a,100
  mov  [gbButtonDebounce],a
  jnz STimerNoDebounce       ; continue waiting for debounce to end

STimerDebounceOver:
  ; it's really ready!
  ; Toggle the button state flag to let the Windows app know that
  ;   the button has been pushed.
  mov  a,1
  xor  [gbButtonPushed],a

  ; Debounce must be over
STimerNoDebounce:
  ; Enable interrupts and return
  mov  a,[gbSysInterruptMask]
```

```
    ipret SysInterrupt

;//$PAGE
;*******************************************************
; SysGPIOEvent()
; @func General purpose port event
; @comm Which pin?
;*******************************************************
SysGPIOEvent:

    ; Save accumulator
    push a

    ; Reset debounce any time we are here
    mov  a,100
    mov  [gbButtonDebounce],a

  SysGPIOButtonDebouncing:
    ; Enable interrupts and return
    mov  a,[gbSysInterruptMask]
    ipret SysInterrupt


;*******************************************************************************
;
;    This section of code responds to activity on End Point 0 and determines
;    what needs to be done.
;
;*******************************************************************************

;//$PAGE
;*******************************************************
; USBEndPoint0Event()
; @func End Point zero USB event.
; @comm Default end point.
;*******************************************************
USBEndPoint0Event:

    ; This code checks to see what type of packet was received
    ;   (Setup, Out, or In) and jumps to the correct routine to decode the
    ;   specifics. After the code to which the jump points is through, it jumps
    ;   back to USBEventEP0End.

    ; Save accumulator
    push a

    ; Is this a SETUP packet?
    iord USBEndP0RxStatus
    and  a,USBEndP0RxSetup      ; Check the setup bit
    jnz  USBEventEP0_SETUP      ; Yes it's a setup, branch

    ; Not a setup, is it an OUT packet?
   ;iord USBEndP0RxStatus
   ;and  a,USBEndP0RxOut
   ;jnz  USBEventEP0_OUT

    ; Not an OUT packet, is it an IN packet?
   ;iord USBEndP0RxStatus
   ;and  a,USBEndP0RxIn
   ;jnz  USBEventEP0_IN

  USBEventEP0_IN:
  USBEventEP0_OUT:

  USBEventEP0End:
    ; OK. We're done with the packet.
    ;   Let's enable interrupts and return
    mov  a,[gbSysInterruptMask]
    ipret SysInterrupt    ; done with EP0 irq service routine

  USBEventEP0Stall:
```

```
    ; Stall any subsequent IN's or OUT's until the
    ;       stall bit (bit 5) is cleard by an I/O write to
    ;       the  USB End Point 0 TX Configuration Register (0x10)
    ;       or any SETUP is received.
    iord USBEndP0TxConfig
    or   a,USBEndP0TxStall
    iowr USBEndP0TxConfig

    ; OK. We've set the stall condition for Endpoint 0.
    ;   Now let's complete the routine.
    jmp  USBEventEP0End

;-------------------------------------------------------------------------------
;
;    We know we have received a Setup token. Now we need to parse it to
;    determine what command it is.
;
;-------------------------------------------------------------------------------


;//$PAGE
;-------------------------------------------------------------------------------
; USBEventEP0_SETUP()
; @func End point event SETUP packet handler.
; @devnote Runs in interrupt enabled context.
;-------------------------------------------------------
USBEventEP0_SETUP:
    ; Well, we have a SETUP packet. Let's find out what to do.
    mov A,[gbSysInterruptMask]
    iowr SysInterrupt

    ; If we are here and are and we are processing a previous Setup,
    ;  we need to abort the processing of the previous Setup
    mov  a,0    ; Clear any indication that we have bytes left to transfer
    mov [gbUSBSendBytes],a

    ; Clear EP0 RxReg (including the Setup flag)
    ; The Data toggle bit remains unchanged, however.
    mov  a,0
    iowr USBEndP0RxStatus

    ;-----------------------------------------
    ; Setup Event
    ;-----------------------------------------

    ; Check the request type and branch to the correct location to handle it.
    mov  a,[USBEndP0FIFO_0]

  USBEventEP0SetupTargetDeviceOUT:
    ; Target Device?
    cmp  a,USBRqstTargetDevice
    jz   USBEventEP0SetupIsSetAddress   ; Yes

  USBEventEP0SetupTargetInterfaceOUT:
    cmp  a,USBRqstTargetInterface
    jz   USBEventEP0Stall               ; Yes. Oops! We don't have an interface.

  USBEventEP0SetupTargetEndpointOUT:
    cmp  a,USBRqstTargetEndPoint
    jz   USBEventEP0Stall               ; Yes

  USBEventEP0SetupTargetDeviceIN:
    cmp  a,USBRqstTargetDevice | USBRqstTypeDirection
    jz   USBEventEP0SetupGetDescriptor  ; Yes

  USBEventEP0SetupTargetInterfaceIN:
    cmp  a,USBRqstTargetInterface | USBRqstTypeDirection
    jz   USBEventEP0Stall               ; Yes Oops! We don't have an interface.

  USBEventEP0SetupTargetEndpointIN:
    cmp  a,USBRqstTargetEndPoint | USBRqstTypeDirection
```

```
    jz   USBEventEP0Stall               ; Yes

    ; Vendor specific commands
  USBEventEP0SetupTargetVendorIN_OUT:
    ; Check request (IN packet OK, OUT packet ERR)
    mov  a,[USBEndP0FIFO_0]
    and  a,USBRqstTypeVendor | USBRqstTargetEndPoint | USBRqstTypeDirection
    cmp  a,USBRqstTypeVendor | USBRqstTargetEndPoint | USBRqstTypeDirection
    jz   USBEventEP0VendorRqst

    ; Unsupported request !!!
    jmp  USBEventEP0Stall               ; Oops! We don't support whatever
                                        ;   request was made.


;//$PAGE
;*******************************************************
; USBEventEP0SetupIsSet()
; @func End point event SETUP to set address.
; @devnote Runs in interrupt enabled context.
;*******************************************************
USBEventEP0SetupIsSetAddress:

    ; Set device address?
    mov  a,[USBRqstMessage]
    cmp  a,USBRqstSetAddress
    jz   USBEventEP0SetupSetAddress         ; Yes

  USBEventEP0SetupIsSetConfig:
    ; Set device configuration?
    mov  a,[USBEndP0FIFO_1]
    cmp  a,USBRqstSetConfiguration
    jz   USBEventEP0SetupSetConfig          ; Yes

    ; Unsupported set request !!!
    jmp  USBEventEP0Stall                   ; No. Stall

  ;USBEventEP0SetupIsGetDescriptor:
    mov  a,[USBRqstMessage]
    cmp  a,USBRqstGetDescriptor
    jz   USBEventEP0SetupGetDescriptor      ; Yes

    ; Unsupported get request !!!
    jmp  USBEventEP0Stall                   ; No

;*******************************************************
; USBEventEP0SetupSetAddress()
; @func End point zero event SETUP to set address.
; @devnote Runs in interrupt enabled context.
; @comm
; The status token of the SetAddress is an IN. So, we send status manually.
;*******************************************************
USBEventEP0SetupSetAddress:

    ; Send ACK
    call USBSendACK
    ; Now that we have been acknowleged, we actually set the address.
    ; This is different from all other commands which execute first
    ;   and then acknowlege (__________________)

    ; Remember this
    inc [gbUSBValidRqsts]

    ; Set Address
    mov  a,[USBRqstWordValueLo]
    iowr USBDeviceAddress

    ; Done
    jmp  USBEventEP0End

;*******************************************************
```

```
; USBEventEP0SetupSetConfig()
; @func End point zero event SETUP to Set Configuration.
; @devnote Runs in interrupt enabled context.
; 1
;  set enumerated (gbSysEnumerated) state,
;  enable GPIO (and EP1, if appropriate)
;  Enable P0 and P1
; 0
;  Reset enumerated (gbSysEnumerated) state,
;  Turn off LED
;  Reset variables
;  Disable GPIO and EP1
;  Disable dallas chip and P0 and P1
;*****************************************************
USBEventEP0SetupSetConfig:

    ; Enumerated !
    mov  a,01h
    mov  [gbSysEnumerated],a

    ; Initialize thermometer
    call ThermInitialize

    ; Enable button interrupt on port 1.
    ;   Actually, this has already been done in main().
    mov  a,04h
    iowr SysPort1IntEnable

    ; enable all appropriate irq's
    mov  a,SysIntTimer1024us | SysIntGPIO | SysIntUSBEndP0
    mov  [gbSysInterruptMask],a

    ; Send ACK
    call USBSendACK
    jmp  USBEventEP0End

;//$PAGE
;*****************************************************
; USBEventEP0SetupGetDescriptor()
; @func End point zero event SETUP to Get Descriptor.
; @devnote Runs in interrupt enabled context.
;*****************************************************
USBEventEP0SetupGetDescriptor:

    ; Get descriptor type
    mov  a,[USBRqstWordValueHi]

  USBEventEP0SetupGetDescriptorDevice:
    ; Device Descriptor?
    cmp  a,USBDescriptorTypeDevice
    jnz  USBEventEP0SetupGetDescriptorConfig    ; No

    ; Remember this
    inc [gbUSBValidRqsts]

    ;*****************************************
    ; Get Device Descriptor Event
    ;*****************************************
    ; Descriptor pointer
    mov  a,(USBDeviceDescription -USBSendROMBufferBase)
    mov  [gbUSBSendBuffer],a

    ; Descriptor size
    mov  a,12h                  ;[USBDeviceDescription]
    mov  [gbUSBSendBytes],a

    ; Check request size field
    call USBSendDescriptorCheckLength

    ; Send buffer
    call USBSendROMBuffer
```

```
  jmp  USBEventEP0End

USBEventEP0SetupGetDescriptorConfig:
  ; Configuration Descriptor?
  cmp  a,USBDescriptorTypeConfig
  jnz  USBEventEP0SetupGetDescriptorString    ; No

  ; Remember this
  inc [gbUSBValidRqsts]

  ;*****************************************
  ; Get Configuration Descriptor Event
  ;*****************************************
  ; Descriptor pointer
  mov  a,(USBConfigurationDescription -USBSendROMBufferBase)
  mov  [gbUSBSendBuffer],a

  ; Descriptor size
  mov  a,09h                  ;[USBConfigurationDescription]
  add  a,09h                  ;[USBInterfaceDescription]
  add  a,07h                  ;[USBEndPointDescriptionInt]
  mov  [gbUSBSendBytes],a

  ; Check request size field
  call USBSendDescriptorCheckLength

  ; Send buffer
  call USBSendROMBuffer
  jmp  USBEventEP0End

USBEventEP0SetupGetDescriptorString:
  ; Get String Descriptor?
  cmp  a,USBDescriptorTypeString
  jnz  USBEventEP0SetupGetDescriptorEnd       ; No

  ;*****************************************
  ; Get String Descriptor Event
  ;*****************************************

  ; Get string descriptor index
  mov  a,[USBRqstWordValueLo]

USBEventEP0SetupGetDescriptorString0:
  cmp  a,0h
  jnz  USBEventEP0SetupGetDescriptorString1   ; No

  ;*****************************************
  ; Get String Language(s) Descriptor Event
  ;*****************************************
  ; Descriptor pointer
  mov  a,(USBStringLanguageDescription -USBSendROMBufferBase)
  mov  [gbUSBSendBuffer],a

  ; Descriptor size
  mov  a,4h                   ;[USBStringLanguageDescription]
  mov  [gbUSBSendBytes],a

  ; Check request size field
  call USBSendDescriptorCheckLength

  ; Send buffer
  call USBSendROMBuffer
  jmp  USBEventEP0End

USBEventEP0SetupGetDescriptorString1:
  cmp  a,1
  jnz  USBEventEP0SetupGetDescriptorString2   ; No

  ;*****************************************
  ; Get String 1 Descriptor Event
  ;*****************************************
```

```
    ; Descriptor pointer
    mov  a,(USBStringDescription1 -USBSendROMBufferBase)
    mov  [gbUSBSendBuffer],a

    ; Descriptor size
    mov  a,10h                  ;[USBStringDescription1]
    mov  [gbUSBSendBytes],a

    ; Check request size field
    call USBSendDescriptorCheckLength

    ; Send buffer
    call USBSendROMBuffer
    jmp  USBEventEP0End

  USBEventEP0SetupGetDescriptorString2:
    cmp  a,2
    jnz  USBEventEP0SetupGetDescriptorString3   ; No

    ;*******************************************
    ; Get String 2 Descriptor Event
    ;*******************************************
    ; Descriptor pointer
    mov  a,(USBStringDescription2 -USBSendROMBufferBase)
    mov  [gbUSBSendBuffer],a

    ; Descriptor size
    mov  a,18h                  ;[USBStringDescription2]
    mov  [gbUSBSendBytes],a

    ; Check request size field
    call USBSendDescriptorCheckLength

    ; Send buffer
    call USBSendROMBuffer
    jmp  USBEventEP0End

  USBEventEP0SetupGetDescriptorString3:
    cmp  a,3
    jnz  USBEventEP0SetupGetDescriptorString4   ; No

    ;*******************************************
    ; Get String 3 Descriptor Event
    ;*******************************************
    ; Descriptor pointer
    mov  a,(USBStringDescription3 -USBSendROMBufferBase)
    mov  [gbUSBSendBuffer],a

    ; Descriptor size
    mov  a,24h                  ;[USBStringDescription3]
    mov  [gbUSBSendBytes],a

    ; Check request size field
    call USBSendDescriptorCheckLength

    ; Send buffer
    call USBSendROMBuffer
    jmp  USBEventEP0End

  USBEventEP0SetupGetDescriptorString4:
    cmp  a,4
    jnz  USBEventEP0SetupGetDescriptorString5   ; No

    ;*******************************************
    ; Get String 4 Descriptor Event
    ;*******************************************
    ; Descriptor pointer
    mov  a,(USBStringDescription4 -USBSendROMBufferBase)
    mov  [gbUSBSendBuffer],a

    ; Descriptor size
```

```
    mov  a,20h                  ;[USBStringDescription4]
    mov  [gbUSBSendBytes],a

    ; Check request size field
    call USBSendDescriptorCheckLength

    ; Send buffer
    call USBSendROMBuffer
    jmp  USBEventEP0End

  USBEventEP0SetupGetDescriptorString5:
    cmp  a,5
    jnz  USBEventEP0SetupGetDescriptorEnd       ; No

    ;*****************************************
    ; Get String 5 Descriptor Event
    ;*****************************************
    ; Descriptor pointer
    mov  a,(USBStringDescription5 -USBSendROMBufferBase)
    mov  [gbUSBSendBuffer],a

    ; Descriptor size
    mov  a,3Ch                  ;[USBStringDescription5]
    mov  [gbUSBSendBytes],a

    ; Check request size field
    call USBSendDescriptorCheckLength

    ; Send buffer
    call USBSendROMBuffer
    jmp  USBEventEP0End

  USBEventEP0SetupGetDescriptorEnd:
    ; Unsupported Get request !!!
    jmp  USBEventEP0Stall

;//$PAGE
;*******************************************************
; USBSendDescriptorCheckLength()
; @func Check and update send length for Get Descriptor
;       requests on end point 0.
; @parm BYTE | gbUSBSendBytes | Number of bytes to send.
;*******************************************************
USBSendDescriptorCheckLength:

    ; High byte set? (Assume <255 bytes)
    mov  a,[USBEndP0FIFO_7]
    cmp  a,0
    jnz  USBSendDescriptorCheckLengthEnd    ; Yes

    ; Check size
    mov  a,[USBEndP0FIFO_6]
    cmp  a,[gbUSBSendBytes]
    jz   USBSendDescriptorCheckLengthEnd    ; equal
    jnc  USBSendDescriptorCheckLengthEnd    ; greater than

    ; New size
    mov  [gbUSBSendBytes],a

  USBSendDescriptorCheckLengthEnd:
    ret

;//$PAGE
;*******************************************************
; USBSendROMBuffer()
; @func Send a number of ROM bytes on end point 0.
; @parm BYTE | gbUSBSendBytes | Number of bytes to send.
; @parm BYTE | gbUSBSendBuffer | Offset from ROM base
;       of data to send.
; @comm assumes IN packets are ignored in the interrupt routine
; @devnote Enables interrupts
```

```
;*****************************************************
USBSendROMBuffer:

    ; Clear flag
    mov  a,0h
    iowr USBEndP0RxStatus

    ; Enable interrupts
    mov  a,[gbSysInterruptMask]
    and  a,~SysIntUSBEndP0
    iowr SysInterrupt

    ; Auto ACK OUT packet (This would be a Status Out)
    mov  a,USBControlAckStatusData
    iowr USBControl

    ; Initialize sequence
    mov  a,0h
    mov  [gbUSBSendSequence],a

    ; Send count
    mov  a,[gbUSBSendBytes]

USendROMBufferLoop:
    ; One 8-byte chunk or less left?
    cmp  a,08h
    jz   USendROMBufferLoopDone     ; exactly 8 bytes left, branch
    jc   USendROMBufferLoopDone     ; less than 8 bytes left, branch

    ; more than 8 bytes left, fall through and loop
    ;    until there are 8 bytes or less.
    ; Save count
    push a

    ; Send 8 byte chunk
    mov  a,08h
    mov  [gbUSBSendBytes],a
    call _USBSendROMBuffer

    ; Check for OUT packet cancelling send
    iord USBEndP0RxStatus
    and  a,USBEndP0RxOut

    ; Restore count
    pop  a

    ; Handle exception: OUT packet cancel send
    jnz  USendROMBufferLoopExit     ; Cancelled

    ; Save bytes left
    sub  a,08h
    mov  [gbUSBSendBytes],a
    jmp  USendROMBufferLoop

  USendROMBufferLoopDone:
    ; Send last 8 or less bytes
    call _USBSendROMBuffer

USendROMBufferLoopExit:
    ret

;//$PAGE
;*****************************************************
; _USBSendROMBuffer()
; @func Buffer and inialize USB send of up
;               to 8 bytes of ROM data on end point 0.
; @comm affects gbUSBSendBytes & gbUSBSendBuffer
;*****************************************************
_USBSendROMBuffer:

    ; Save x
```

```
    push x

    ; Initialize
    mov  x,0h

  _USendROMBufferLoop:
    ; Any more?
    mov  a,0h
    cmp  a,[gbUSBSendBytes]
    jz  _USendROMBufferLoopDone         ; No more
    dec  [gbUSBSendBytes]

    ; Move bytes to FIFO
    mov  a,[gbUSBSendBuffer]
    index USBSendROMBufferBase
    mov  [x +USBEndP0FIFO],a
    inc  x

    ; Next byte
    inc  [gbUSBSendBuffer]
    jmp _USendROMBufferLoop

  _USendROMBufferLoopDone:
    ; Re-enable reception
    mov  a,0h
    iowr USBEndP0RxStatus

    ; Toggle sequence
    mov  a,USBEndP0TxSequence
    xor  [gbUSBSendSequence],a

    ; Send bytes
    push x
    pop  a
    or   a,[gbUSBSendSequence]
    or   a,USBEndP0TxRespond
    iowr USBEndP0TxConfig

    ; The FIFO is loaded, go and wait untill it's read
    call USBSendWaitForComplete

  _USendROMBufferEnd:
    ; Restore and exit
    pop  x
    ret

;//$PAGE
;******************************************************
; USBSendACK()
; func Respond to a "USB Status In" with a zero byte buffer with
;   Sequence field set) on end point 0.
; Called by SetAddress and SetConfig commands
;******************************************************
USBSendACK:

    ; Status response to Status In is to send a zero byte packet
    mov  a,USBEndP0TxRespond | USBEndP0TxSequence
    iowr USBEndP0TxConfig

    ; Enable interrupts
    mov  a,[gbSysInterruptMask]
    iowr SysInterrupt

    ; Wait for send complete
    jmp  USBSendWaitForComplete

;******************************************************
; USBSendWaitForComplete()
; @func Wait for send to complete on end point 0.
;******************************************************
```

```
; At some point, either the 0 data will be ACK'd or a SETUP
;    will come in.
; Either event will cause the "Enable Respond
;    to In Packets" to be reset, and we will fall out of the loop.
; In either case, an EP0 IRQ will be generated (5.9.2.2 in Cyp
;    device spec) if EP0 irq is enabled.

USBSendWaitForComplete:

    ; Poll the send complete bit
    ; This will be reset when the data has been sent to the host
    ;   and the host has ACK's, or the host has sent another SETUP
    ;   which should terminate this activity in any case.
    iord USBEndP0TxConfig
    and  a,USBEndP0TxRespond
    jz   USBSendWaitComplete

    ; Check for OUT packet cancelling send. A STATUS OUT should
    ;   terminate any pending IN's. A Setup could also set the Out bit.
    iord USBEndP0RxStatus
    and  a,USBEndP0RxOut
    jnz  USBSendWaitComplete    ; Cancelled

    ; Keep waiting
    jmp  USBSendWaitForComplete

  USBSendWaitComplete:
    ret

;//$PAGE
;-----------------------------------------------------
; USBEventEP0VendorRqst()
; @func Vendor request on end point zero.
; @devnote Runs in interrupt disabled context.
;-----------------------------------------------------
USBEventEP0VendorRqst:

    ; Save it
    push x

    ; Check Protocol
    mov  a,[USBEndP0FIFO_1]

  USBEventEP0VendorRqstPing:
    cmp  a,0h
    jnz  USBEventEP0VendorRqstReadROM       ; No

    ;-----------------------------------------
    ; Ping Event
    ;-----------------------------------------
    jmp  USBEventEP0VendorRqstFinish

  USBEventEP0VendorRqstReadROM:
    cmp  a,01h
    jnz  USBEventEP0VendorRqstReadRAM       ; No

    ;-----------------------------------------
    ; Read ROM Event
    ;-----------------------------------------
    mov  a,[USBEndP0FIFO_2]
    index USBSendROMBufferBase
    mov  [USBEndP0FIFO_1],a
    jmp  USBEventEP0VendorRqstFinish

  USBEventEP0VendorRqstReadRAM:
    cmp  a,02h
    jnz  USBEventEP0VendorRqstWriteRAM      ; No

    ;-----------------------------------------
    ; Read RAM Event
    ;-----------------------------------------
```

```
  mov  a,[USBEndP0FIFO_2]
  push a
  pop  x
  mov  a,[x +0]
  mov  [USBEndP0FIFO_1],a
  jmp  USBEventEP0VendorRqstFinish

USBEventEP0VendorRqstWriteRAM:
  cmp  a,3
  jnz  USBEventEP0VendorRqstReadPort      ; No

  ;*****************************************
  ; Write RAM Event
  ;*****************************************
  mov  a,[USBEndP0FIFO_2]
  push a
  pop  x
  mov  a,[USBEndP0FIFO_4]
  mov  [x +0],a
  jmp  USBEventEP0VendorRqstFinish

USBEventEP0VendorRqstReadPort:
  cmp  a,04h
  jnz  USBEventEP0VendorRqstWritePort     ; No

  ;*****************************************
  ; Read Port Event
  ;*****************************************
  mov  a,[USBEndP0FIFO_2]
  cmp  a,0h
  jnz  USBEventEP0VendorRqstReadPort1

USBEventEP0VendorRqstReadPort0:
  iord SysPort0
  jmp  USBEventEP0VendorRqstReadPortsDone

USBEventEP0VendorRqstReadPort1:
  iord SysPort1
 ;jmp  USBEventEP0VendorRqstReadPortsDone ; redundant, but good practice

USBEventEP0VendorRqstReadPortsDone:
  mov  [USBEndP0FIFO_1],a
  jmp  USBEventEP0VendorRqstFinish

USBEventEP0VendorRqstWritePort:
  cmp  a,05h
  jnz  USBEventEP0Stall                   ; No

  ;*****************************************
  ; Write Port Event
  ;*****************************************
  mov  a,[USBEndP0FIFO_2]
  cmp  a,0
  jnz  USBEventEP0VendorRqstReadPort1

USBEventEP0VendorRqstWritePort0:
  mov  a,[USBEndP0FIFO_4]
  iowr SysPort0
  jmp  USBEventEP0VendorRqstWritePortsDone

USBEventEP0VendorRqstWritePort1:
  mov  a,[USBEndP0FIFO_4]
  iowr SysPort1
 ;jmp  USBEventEP0VendorRqstWritePortsDone ; redundant, but good practice

USBEventEP0VendorRqstWritePortsDone:
 ;jmp  USBEventEP0VendorRqstFinish ; redundant, but good practice

USBEventEP0VendorRqstFinish:

  ; Protocol ACK
```

```
    mov  a,42h
    mov  [USBEndP0FIFO_0],a

    ; Auto ACK OUT packet
    mov  a,USBControlAckStatusData
    iowr USBControl

    ; Send bytes as Data1
    mov  a,8
    or   a,USBEndP0TxSequence
    or   a,USBEndP0TxRespond
    iowr USBEndP0TxConfig

   ;call USBSendWaitForComplete

    ; Restore it
    pop  x

    ; Return
    jmp  USBEventEP0End

;****************************
;//$PAGE
include "ds1620a.asm"
;****************************

;******************************************************
; SysDelayMS()
; @func Delay some number of milliseconds.
; @parm register | A | Number of milliseconds (0=65536).
; @comm Protects A and X registers.
;******************************************************
SysDelayMS:

    ; Save em'
    push a
    push x

  SysDelayMSLoop:
    ; Save count
    push a

    ; Delay 1ms = 10 * 100us
    mov  a,10

  SysDelayMSLoopDelay:
    ; Save it
    push a

    ; Delay 100us
    mov  a,100
    call SysDelay

    ; Done?
    pop  a
    dec  a
    jnz  SysDelayMSLoopDelay

    ; Done?
    pop  a
    dec  a
    jnz  SysDelayMSLoop

    ; Restore em'
    pop  x
    pop  a
    ret

;******************************************************
; SysDelay()
; @func Delay some number of microseconds.
```

```
; @parm register | A | Number of microseconds (0=65536).
; @comm Protects A and X registers.
;*****************************************************
SysDelay:

    ; Save em'
    push a
    push x

  SysDelayLoop:
    ; Save count
    push a

    ; Delay 1ms
    nop     ; 4 clock cycles (6Mhz or 166us cycle???)
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop

    ; Done?
    pop  a
    dec  a
    jnz  SysDelayLoop

    ; Restore em'
    pop  x
    pop  a
    ret


;*****************************************************
; Data Segment (ROM)
;*****************************************************
USBSendROMBufferBase:
USBDeviceDescription:
    db 12h          ; Length
    db 01h          ; Type (1=device)
    db 00h,01h      ; Complies to USB Spec. v1.00
    db 00h          ; Class code (0=??)
    db 00h          ; SubClass code (0=??)
    db 00h          ; Protocol (0=none)(9.6.1)
    db 08h          ; Max. packet size for port0
    db B4h,04h      ; Vendor ID: (0x4B4=Cypress)
    db 02h,00h      ; Product ID (0x02=USB Thermometer)
    db 09h,00h      ; Device release v0.90
    db 01h          ; Manufacturer string descriptor index (0=none)
    db 02h          ; Product string descriptor index (0=none)
    db 00h          ; Serial number string descriptor index (0=none)
    db 01h          ; Number of possible configurations
USBDeviceDescriptionEnd:

;*************************************************
;
USBConfigurationDescription:
    db 09h          ; Length
    db 02h          ; Type (2=config)
    db 19h,00h   ; Total data length (1 config,1 interface,1 endpoints)
    db 01h          ; Interface supported (1=???)
    db 01h          ; Configuration value (1=???)
    db 04h          ; Confituration string descriptor index (0=none)
    db 80h          ; Configuration (80h=Bus powered)
    db 32h          ; Maximum power consumption in 2mA units
USBConfigurationDescriptionEnd:

;*************************************************
```

```
;
USBInterfaceDescription:
    db 09h          ; Length
    db 04h          ; Type (4=interface)
    db 00h          ; Number of interfaces (0 based)
    db 00h          ; Alternate settings
    db 01h          ; Number of endpoints (1 based) (9.6.3)
    db 00h          ; Class code (0=non-specified,1=kb,2=mouse,3=joystick ???)
    db 00h          ; Subclass code (0=???)
    db 00h          ; Protocol code (0=non-specified)
    db 05h          ; Interface string index (0=non-specified, 1,2,3,...)
USBInterfaceDescriptionEnd:

;***********************************************
; Never used for EP0
USBEndPointDescriptionInt:
    db 07h          ; Length
    db 05h          ; Type (5=endpoint)
    db 81h          ; Address (EP#=1 | [0x80=IN, 0=OUT])
    db 03h          ; Attribute (0=control,1=isochronous,2=bulk,3=interrupt)
    db 08h,00h      ; Max packet size
    db 0Ah          ; Interval (10 ms)
USBEndPointDescriptionIntEnd:

;***********************************************
;
USBStringLanguageDescription:
    db 04h          ; Length
    db 03h          ; Type (3=string)
    db 09h          ; Language:  English
    db 01h          ; Sub-language: US

USBStringDescription1:
    db 10h          ; Length
    db 03h          ; Type (3=string)
    dsu "Cypress"

USBStringDescription2:
    db 18h          ; Length
    db 03h          ; Type (3=string)
    dsu "Thermometer"

USBStringDescription3:
                    ; If a SN is used, this must be unique
                    ;    for every device or the device may
                    ;    not enumerate properly

USBStringDescription4:
    db 20h          ; Length
    db 03h          ; Type (3=string)
    dsu "Get Temperature"

USBStringDescription5:
    db 3Ch          ; Length
    db 03h          ; Type (3=string)
    dsu "EndPoint1 10ms Interrupt Pipe"

USBSendROMBufferTail:

CopyrightStrings:
    ds "USB Thermometer Project, Version 1.01"
    ds "Copyright Slade Systems, Inc., July, 1997"
    ds "Copyright Marc Reinig, July, 1997"
    ds "Copyright Cypress Semiconductors, Inc., July, 1997"
```

CY6300X.INC

```
;;*****************************************************************************
;; C7C63x0x.h - Cypress Semiconductor Cy7C63x0x micrprocessor definitions
;; Copyright (c) Slade Systems, Inc, 1997
;;
;; Cypress Semiconductor Corp.
;; 12032 113th Ave NE, Kirkland, WA 98034
;; 206-821-9202 - 206-820-8959(f)
;;
;;*****************************************************************************


;*****************************************************************************
;
; M8 - 8bit microprocessor
; registers: accumulator     'acc'
;            index           'x'
;            stack pointer   'dsp'
;            program SP      'psp'
;            program counter 'pc'  16 bits (14 bit addressing)
;             PC low         'pcl'
;             PC high        'pch'
; When PC is pushed on stack
;  carry flag is stored in bit 14
;  zero  flag is stored in bit 15
;
; Program ROM 4096 bytes in 256 byte pages
; Program RAM  128 bytes
; Processor PORTs contain 16k-ohm resistor (pull-up and slew control)
;
; After reset:
;  Port 0 and Port 1 are set high
;
;
;*****************************************************************************

;***********************************************
;
; I/O ports defined
SysPort0                  :equ 00h     ; GPIO data port 0 (P00-P07)
SysPort1                  :equ 01h     ; GPIO data port 1 (P10-P13)
SysPort0IntEnable         :equ 04h     ; Port0 Interrupt Enable
SysPort1IntEnable         :equ 05h     ; Port1 Interrupt Enable
SysPort0PullUp            :equ 08h     ; Port0 PullUp Resistor Enable (0=active)
SysPort1PullUp            :equ 09h     ; Port1 PullUp Resistor Enable (0=active)

; General
SysStatus                 :equ FFh     ;
 SysStatusRun             :equ 01h     ;
 SysStatusReserved2       :equ 02h     ; nul
 SysStatusReserved3       :equ 04h     ; nul
 SysStatusSuspend         :equ 08h     ; write only (restart =256us)
 SysStatusPowerOn         :equ 10h     ;
 SysStatusUSBReset        :equ 20h     ;
 SysStatusWDReset         :equ 40h     ;
 SysStatusReserved7       :equ 80h     ; nul
SysWatchDog               :equ 21h     ; WatchDog controller
SysTimerExternal          :equ 22h     ; Timer also ???
SysTimer                  :equ 23h     ; Timer (read only) {6MHZ=1us resolution}
SysInterrupt              :equ 20h     ; Global interrupt
 SysIntReserved0          :equ 01h     ;
 SysIntTimer128us         :equ 02h     ;
 SysIntTimer1024us        :equ 04h     ;
 SysIntUSBEndP0           :equ 08h     ;
 SysIntUSBEndP1           :equ 10h     ;
 SysIntReserved5          :equ 20h     ;
 SysIntGPIO               :equ 40h     ;
 SysIntWakeUp             :equ 80h     ;
```

```
; Ouput ISink ???
SysPort0ISinkPin0          :equ 30h     ;
SysPort0ISinkPin1          :equ 31h     ;
SysPort0ISinkPin2          :equ 32h     ;
SysPort0ISinkPin3          :equ 33h     ;
SysPort0ISinkPin4          :equ 34h     ;
SysPort0ISinkPin5          :equ 35h     ;
SysPort0ISinkPin6          :equ 36h     ;
SysPort0ISinkPin7          :equ 37h     ;
SysPort1ISinkPin0          :equ 38h     ;
SysPort1ISinkPin1          :equ 39h     ;
SysPort1ISinkPin2          :equ 3Ah     ;
SysPort1ISinkPin3          :equ 3Bh     ;

; USB FIFOs
USBEndP0FIFO               :equ 70h
USBEndP0FIFO_0             :equ 70h     ; Will contain CRC if (rx count <8)
USBEndP0FIFO_1             :equ 71h
USBEndP0FIFO_2             :equ 72h
USBEndP0FIFO_3             :equ 73h
USBEndP0FIFO_4             :equ 74h
USBEndP0FIFO_5             :equ 75h
USBEndP0FIFO_6             :equ 76h
USBEndP0FIFO_7             :equ 77h
;
USBEndP1FIFO               :equ 78h
USBEndP1FIFO_0             :equ 78h
USBEndP1FIFO_1             :equ 79h
USBEndP1FIFO_2             :equ 7Ah
USBEndP1FIFO_3             :equ 7Bh
USBEndP1FIFO_4             :equ 7Ch
USBEndP1FIFO_5             :equ 7Dh
USBEndP1FIFO_6             :equ 7Eh
USBEndP1FIFO_7             :equ 7Fh

;***********************************************
;
USBDeviceAddress           :equ 12h     ; Assigned device address

; USB port control
USBControl                 :equ 13h     ; Status/Control register
 USBControlBusActive       :equ 01h     ; 1=active, write 0 and watch if bus dies
 USBControlResume          :equ 02h     ; set transmitter to k state sending resume to
    host ???
 USBControlReserve2        :equ 04h
 USBControlAckStatusData   :equ 08h     ; Auto ACK Data1 SETUP OUT data packets
 USBControlAckOutData      :equ 10h     ; Auto ACK Data1 OUT       data packets
 USBControlReserve5        :equ 20h
 USBControlReserve6        :equ 40h
 USBControlReserve7        :equ 80h

USBEndP0RxStatus           :equ 14h     ; Port0 receive status
 USBEndP0RxSetup           :equ 01h     ; 1=setup token received (must be cleared to
    write FIFOs ???)
 USBEndP0RxOut             :equ 02h     ; 1=out   token received
 USBEndP0RxIn              :equ 04h     ; 1=in    token received
 USBEndP0RxDataFlag        :equ 08h
 USBEndP0RxCount0          :equ 10h     ; size =count -2 (two bytes of CRC)
 USBEndP0RxCount1          :equ 20h
 USBEndP0RxCount2          :equ 40h
 USBEndP0RxCount3          :equ 80h

USBEndP0TxConfig           :equ 10h     ; Transmit configuration
 USBEndP0TxCount0          :equ 01h
 USBEndP0TxCount1          :equ 02h
 USBEndP0TxCount2          :equ 04h
 USBEndP0TxCount3          :equ 08h
 USBEndP0TxRxErr           :equ 10h     ; read and write
 USBEndP0TxStall           :equ 20h     ;
 USBEndP0TxSequence        :equ 40h     ;
```

```
 USBEndP0TxRespond         :equ 80h     ;

USBEndP1TxConfig           :equ 11h     ;
 USBEndP1TxCount0          :equ 01h
 USBEndP1TxCount1          :equ 02h
 USBEndP1TxCount2          :equ 04h
 USBEndP1TxCount3          :equ 08h
 USBEndP1TxEnable          :equ 10h     ;
 USBEndP1TxStall           :equ 20h     ;
 USBEndP1TxSequence        :equ 40h     ;
 USBEndP1TxRespond         :equ 80h

;***********************************************
; USB Protocol
;union USBRqst
;{   struct
;    {   BYTE bReceipient :5;    //
    0=Device,1=Interface,2=Endpoint,3=Other,4..31=Reserved
;        BYTE bType       :2;    // 1=Standard,1=Class,2=Vendor,3=Reserved
;        BYTE bDirection  :1;    // 0=Host to Device,1=Device to Host
;        BYTE bRqst;             //
;                                // 0x00,0x01,0x02 =Clear Feature
;                                // 0x00,0x01,0x02 =Set Feature
;                                // 0x80,0x81,0x82 =Get Status
;                                // 0x00 =Set Address
;                                // 0x80 =Get Descriptor
;                                // 0x00 =Set Descriptor
;                                // 0x80 =Get Configuration
;                                // 0x81 =Get Interface
;                                // 0x01 =Set Interface
;                                // 0x82 =Synch Frame
;        WORD wValue;            //
;        WORD wIndex;            //
;        WORD wLength;           //
;    }
;}

;***********************************************
; USB Protocol
USBRqstType                :equ USBEndP0FIFO_0  ;
 USBRqstTypeDirection      :equ 80h     ; 1=Device to Host,0=Host to Device

 USBRqstTypeMask           :equ 60h
 USBRqstTypeStandard       :equ 00h
 USBRqstTypeClass          :equ 20h
 USBRqstTypeVendor         :equ 40h
 USBRqstTypeReserved       :equ 60h

 USBRqstTargetDevice       :equ 00h
 USBRqstTargetInterface    :equ 01h
 USBRqstTargetEndPoint     :equ 02h
 USBRqstTargetOther        :equ 03h

USBRqstMessage             :equ USBEndP0FIFO_1  ;
 USBRqstGetStatus          :equ 00h     ; bit field: 0x01 =Self powered,0x02 =Remote
    wakeup
 USBRqstClearFeature       :equ 01h
 USBRqstReserved02         :equ 02h
 USBRqstSetFeature         :equ 03h
 USBRqstReserved04         :equ 04h
 USBRqstSetAddress         :equ 05h
 USBRqstGetDescriptor      :equ 06h
 USBRqstSetDescriptor      :equ 07h
 USBRqstGetConfiguration   :equ 08h
 USBRqstSetConfiguration   :equ 09h
 USBRqstGetInterface       :equ 0Ah
 USBRqstSetInterface       :equ 0Bh
 USBRqstSynchFrame         :equ 0Ch
 USBRqstReserved0D         :equ 0Dh

USBRqstWordValueLo         :equ USBEndP0FIFO_2  ;
```

```
USBRqstWordValueHi         :equ USBEndP0FIFO_3  ;

;***********************************************
;
;
USBDescriptorTypeDevice     :equ 01h
USBDescriptorTypeConfig     :equ 02h
USBDescriptorTypeString     :equ 03h
USBDescriptorTypeInterface  :equ 04h
USBDescriptorTypeEndPoint   :equ 05h
USBDescriptorTypeReserved06 :equ 06h

;***********************************************
;
;
USBRawProtocolSetup         :equ B4h
USBRawProtocolIn            :equ 96h
USBRawProtocolOut           :equ 87h
USBRawProtocolPort0         :equ C3h
USBRawProtocolPort1         :equ D2h
USBRawProtocolACK           :equ 4Bh
USBRawProtocolNAK           :equ 5Ah
```

DALLAS.ASM

```
;; DS1620a.asm - DS1620 High Reolution Temperature Measurement Sensor

include "ds1620a.inc"

;
ThermPort                   :equ 00h     ; SysPort0
ThermMaskBits               :equ 07h     ;
ThermData                   :equ 01h     ;
ThermClock                  :equ 02h     ;
ThermReset                  :equ 04h     ;

gbThermProtocol             :equ 30h     ;
gbThermPortValue            :equ 31h     ;
gbThermPortMirror           :equ 32h     ;
gbThermTempRead             :equ 33h     ;
gbThermTempRead2            :equ 34h     ;
gbThermTempLast             :equ 78h     ;USBEndP1FIFO
gbThermTempLast2            :equ 79h     ;USBEndP1FIFO +1

;//$PAGE
;*****************************************************************************
; ThermInitialize()
; @func Initialize the thermometer to continuous mode.
;*****************************************************************************
ThermInitialize:

    ; Standalone mode
    mov  a,ThermConfigRead
    call ThermPortRead

    ; Check mode
    mov  a,[gbThermPortValue]
    and  a,ThermControlOneShot | ThermControlCPUUse
    cmp  a,ThermControlCPUUse
    jz   ThermInitDone

    ; Set mode
    mov  a,[gbThermPortValue]
    and  a,~(ThermControlOneShot | ThermControlCPUUse)
    or   a,ThermControlCPUUse | 08h ; set reserved bit
    mov  [gbThermPortValue],a

    ; Write it out
    mov  a,ThermConfigWrite
    call ThermPortWrite

    ; Wait 10 milliseconds
    mov  a,10
    call SysDelayMS

  ThermInitDone:
    ; Start conversion
    mov  a,ThermConvertStart
    mov  [gbThermProtocol],a
    call ThermPortResetHigh
    call ThermPortProtocolWrite
    call ThermPortResetLow

    ret

;//$PAGE
;*****************************************************************************
; ThermReadTemperature()
; @func Read the current temperature.
;*****************************************************************************
ThermReadTemperature:
```

```
    ; Initialize results
    mov  a,0
    mov  [gbThermTempRead],a
    mov  [gbThermTempRead2],a

    ; Read Temperature request
    mov  a,ThermTempRead
    mov  [gbThermProtocol],a

    ; Get temperature
    call ThermPortResetHigh
    call ThermPortProtocolWrite
    call ThermPortReadTemperature
    call ThermPortResetLow

    ; Save results
    mov  a,[gbThermTempRead]
    mov  [gbThermTempLast],a
    mov  a,[gbThermTempRead2]
    mov  [gbThermTempLast2],a

    ret

;//$PAGE
;*****************************************************************************
; ThermPortResetHigh()
; @func .
;*****************************************************************************
ThermPortResetHigh:

    ; Initialize mirror
    iord [ThermPort]
    mov  [gbThermPortMirror],a

    ; ThermReset =1;
    or   a,ThermReset
    mov  [gbThermPortMirror],a
    iowr ThermPort
    ret

;*****************************************************************************
; ThermPortResetLow()
; @func .
;*****************************************************************************
ThermPortResetLow:
    ; ThermReset =0;
    mov  a,[gbThermPortMirror]
    and  a,~ThermReset
    iowr ThermPort
    ret

;*****************************************************************************
; ThermWaitForDone()
; @func Wait for indication of temperature conversion complete.
;*****************************************************************************
ThermWaitForDone:

    ; Read config
    mov  a,ThermConfigRead
    call ThermPortRead

    ; Test flag
    mov  a,[gbThermPortValue]
    and  a,ThermControlDone
    jz   ThermWaitForDone
    ret

;*****************************************************************************
; ThermPortWrite()
; @func Write protocol byte and value byte.
; @parm byte | A | Protocol to send.
```

```
;*****************************************************************************
ThermPortWrite:

    ; Save input
    mov  [gbThermProtocol],a

    call ThermPortResetHigh
    call ThermPortProtocolWrite
    call ThermPortWrite8Bits
    call ThermPortResetLow
    ret

;*****************************************************************************
; ThermPortRead()
; @func Write protocol byte and read value byte into gbThermPortValue.
; @parm byte | A | Protocol to send.
;*****************************************************************************
ThermPortRead:

    ; Save input
    mov  [gbThermProtocol],a

    ; Initialize results
    mov  a,0
    mov  [gbThermPortValue],a

    call ThermPortResetHigh
    call ThermPortProtocolWrite
    call ThermPortRead8Bits
    call ThermPortResetLow
    ret

;*****************************************************************************
; ThermPortProtocolWrite()
; @func .
; @parm BYTE | gbThermProtocol | Protocol value.
;*****************************************************************************
ThermPortProtocolWrite:

    mov  a,[gbThermPortValue]
    push a
    mov  a,[gbThermProtocol]
    mov  [gbThermPortValue],a
    call ThermPortWrite8Bits
    pop  a
    mov  [gbThermPortValue],a

    ret

;//$PAGE
;*****************************************************************************
; ThermPortReadTemperature()
; @func .
; @parm BYTE | gbThermTempRead | Returned read temperature (low bits).
; @parm BYTE | gbThermTempRead2 | Returned read temperature (high bit).
;*****************************************************************************
ThermPortReadTemperature:

    ; Setup bitmask
    mov  a,1
    push a
    pop  x

    mov  a,8
  ThermPortTempReadLoop:
    push a
    ;*****************************************

    ; Tri-state data pin for input
    mov  a,[gbThermPortMirror]
    or   a,ThermData
```

```
    iowr ThermPort

    ; ThermClock =0;
    mov  a,[gbThermPortMirror]
    and  a,~ThermClock
    mov  [gbThermPortMirror],a
    iowr ThermPort

    ; Read in data pin and check for 0 or 1
    iord ThermPort
    and  a,ThermData
    jnz  ThermPortTempReadValue1

  ThermPortTempReadValue0:
    jmp  ThermPortTempReadClock
  ThermPortTempReadValue1:
    ; Use bitmask
    push x
    pop  a
    or   [gbThermTempRead],a
   ;jmp  ThermPortTempReadClock ; redundant, but good practice

  ThermPortTempReadClock:
    ; ThermClock =1;
    mov  a,[gbThermPortMirror]
    or   a,ThermClock
    mov  [gbThermPortMirror],a
    iowr ThermPort

    ; Next bit in mask
    push x
    pop  a
    asl  a
    push a
    pop  x

    ;*****************************************
    ; Finished?
    pop  a
    dec  a
    jnz  ThermPortTempReadLoop

    ;*****************************************
    ; Last bit
    ;*****************************************
    ; Tri-state data pin for input
    mov  a,[gbThermPortMirror]
    or   a,ThermData
    iowr ThermPort

    ; ThermClock =0;
    mov  a,[gbThermPortMirror]
    and  a,~ThermClock
    mov  [gbThermPortMirror],a
    iowr ThermPort

    ; Read in data pin and check for 0 or 1
    iord ThermPort
    and  a,ThermData
    jnz  ThermPortTempReadLastValue1

  ThermPortTempReadLastValue0:
    jmp  ThermPortTempReadLastClock
  ThermPortTempReadLastValue1:
    ; Use bitmask
    mov  a,1
    mov  [gbThermTempRead2],a
   ;jmp  ThermPortTempReadLastClock ; redundant, but good practice

  ThermPortTempReadLastClock:
    ; ThermClock =1;
```

```
    mov  a,[gbThermPortMirror]
    or   a,ThermClock
    mov  [gbThermPortMirror],a
    iowr ThermPort

    ret

;//$PAGE
;***************************************************************************
; ThermPortRead8Bits()
; @func .
; @parm BYTE | gbThermPortValue | Return read byte.
;***************************************************************************
ThermPortRead8Bits:

    ; Setup bitmask
    mov  a,1
    push a
    pop  x

    mov  a,8
 ThermPortValueReadLoop:
    push a
    ;*****************************************

    ; Tri-state data pin for input
    mov  a,[gbThermPortMirror]
    or   a,ThermData
    iowr ThermPort

    ; ThermClock =0;
    mov  a,[gbThermPortMirror]
    and  a,~ThermClock
    mov  [gbThermPortMirror],a
    iowr ThermPort

    ; Read in data pin and check for 0 or 1
    iord ThermPort
    and  a,ThermData
    jnz  ThermPortReadValue1

  ThermPortReadValue0:
    jmp  ThermPortReadClock
  ThermPortReadValue1:
    ; Use bitmask
    push x
    pop  a
    or   [gbThermPortValue],a
   ;jmp  ThermPortReadClock ; redundant, but good practice

  ThermPortReadClock:
    ; ThermClock =1;
    mov  a,[gbThermPortMirror]
    or   a,ThermClock
    mov  [gbThermPortMirror],a
    iowr ThermPort

    ; Next bit in mask
    push x
    pop  a
    asl  a
    push a
    pop  x

    ;*****************************************
    ; Finished?
    pop  a
    dec  a
    jnz  ThermPortValueReadLoop

    ret
```

```
;//$PAGE
;*******************************************************************************
; ThermPortWrite8Bits()
; @func .
; @parm BYTE | gbThermPortValue | Value to write
;*******************************************************************************
ThermPortWrite8Bits:

    ; Setup bitmask
    mov  a,1
    push a
    pop  x

    mov  a,8
  ThermPortWriteLoop:
    push a
    ;*****************************************

    ; Get bitmask
    push x
    pop  a

    and  a,[gbThermPortValue]
    jnz  ThermPortWriteValue1

  ThermPortWriteValue0:
    ; ThermData =0;
    mov  a,[gbThermPortMirror]
    and  a,~ThermData
    mov  [gbThermPortMirror],a
    iowr ThermPort
    jmp  ThermPortWriteValueClock
  ThermPortWriteValue1:
    ; ThermData =1;
    mov  a,[gbThermPortMirror]
    or   a,ThermData
    mov  [gbThermPortMirror],a
    iowr ThermPort
   ;jmp  ThermPortWriteValueClock ; redundant, but good practice

  ThermPortWriteValueClock:
    ; ThermClock =0;
    mov  a,[gbThermPortMirror]
    and  a,~ThermClock
    mov  [gbThermPortMirror],a
    iowr ThermPort

    ; Next bit in mask
    push x
    pop  a
    asl  a
    push a
    pop  x

    ; ThermClock =1;
    mov  a,[gbThermPortMirror]
    or   a,ThermClock
    iowr ThermPort

    ;*****************************************
    ; Finished?
    pop  a
    dec  a
    jnz  ThermPortWriteLoop

    ret
```

DALLAS.INC

```
;; DS1620a.inc - DS1620 High Reolution Temperature Measurement Sensor
;;*****************************************************************************


; DS1620 Control register
ThermControlOneShot :equ 01h    ;
ThermControlCPUUse  :equ 02h    ; 1=data clock,0=Clock line signals start conversion
ThermControlNVB     :equ 10h    ; Nonvolatile Memory Busy flag (up to 10ms)
ThermControlDone    :equ 80h    ;
ThermControlFlags   :equ 60h    ; THF and TLF { Temperature High Flag (THF) and
    Temperature Low Flag (TLF) }

; DS1620 Protocol
ThermRead           :equ A0h

ThermTempRead       :equ AAh
ThermConvertStart   :equ EEh
ThermConvertStop    :equ 22h
ThermTempHighWrite  :equ 01h
ThermTempLowWrite   :equ 02h
ThermTempHighRead   :equ A1h    ;ThermTempHighWrite OR ThermRead
ThermTempLowRead    :equ A2h    ;ThermTempLowWrite  OR ThermRead
ThermConfigWrite    :equ 0Ch
ThermConfigRead     :equ ACh    ;ThermConfigWrite OR ThermRead
ThermCounterRead    :equ A0h    ;00h              OR ThermRead
ThermCounterLoad    :equ 41h    ;undocumented ???
ThermSlopeRead      :equ A9h
```

I. Thermometer driver reference

The Cypress driver is accessed through the Windows DeviceIoControl() API. The following code and table illustrates its use.

```
Type OVERLAPPED
  Internal As Long
  InternalHigh As Long
  offset As Long
  OffsetHigh As Long
  hEvent As Long
End Type

Public gOverlapped As OVERLAPPED
Public hgDrvrHnd As LONG
Dim lIn as long, lInSize as long, lOut as long, lOutSize as long, lSize as long
Dim ltemp as long

ltemp = DeviceIoControl(hgDrvrHnd, 4&, lIn, lInSize, lOut, lOutSize, lSize, gOverlapped)
```

Command				Command Value				Out Value			
		Size (Bytes)		lIn				lOut			
Function	**Value**	**lIn**	**lOut**	**MSB**			**LSB**	**MSB**			**LSB**
Set LED Brightness	0Eh	2	1	NA	NA	Brightness	0Eh	NA	NA	NA	Status
Read Thermometer	0Bh	1	3	NA	Button	Temp	0Bh	Button	Sign	Temp	Status
Read Port	014h	2	2	NA	NA	Port	014h	NA	NA	Value	Status
Write Port	015h	3	1	NA	Value	Port	015h	NA	NA	NA	Status
Read RAM	016h	2	2	NA	NA	Address	016h	NA	NA	Value	Status
Write RAM	017h	3	1	NA	Value	Address	017h	NA	NA	NA	Status
Read ROM	018h	3	2	NA	Index	NA	018h	NA	NA	Value	Status
Indexed to USBSendROMBufferBase											

J. References and Links

Obtaining the latest version of the USB specification

You may obtain the current version of the USB Specification (Revision 1.0) on the Cypress CD-ROM. You may also obtain updates to the USB specification and other USB information and documents from the USB web site (http://www.usb.org).

Obtaining the latest assembly code for the Cypress USB Thermometer

You may obtain the latest version of the assembly code for the Cypress USB Thermometer from the Cypress web site (http://www.cypress.com).

Obtaining the latest Cypress USB Thermometer driver

You may obtain the latest version of the Cypress USB Thermometer application for the Cypress USB Thermometer from the Cypress web site (http://www.cypress.com).

Obtaining the latest Cypress USB Thermometer application

You may obtain the latest version of the driver for the Cypress USB Thermometer from the Cypress web site (http://www.cypress.com).

K. Q&A, Errata and Gotchas

- **How can I tell if my system supports the USB**

  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 (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 K1* and *Figure K2*).

Figure K1 Windows 95 Control Panel

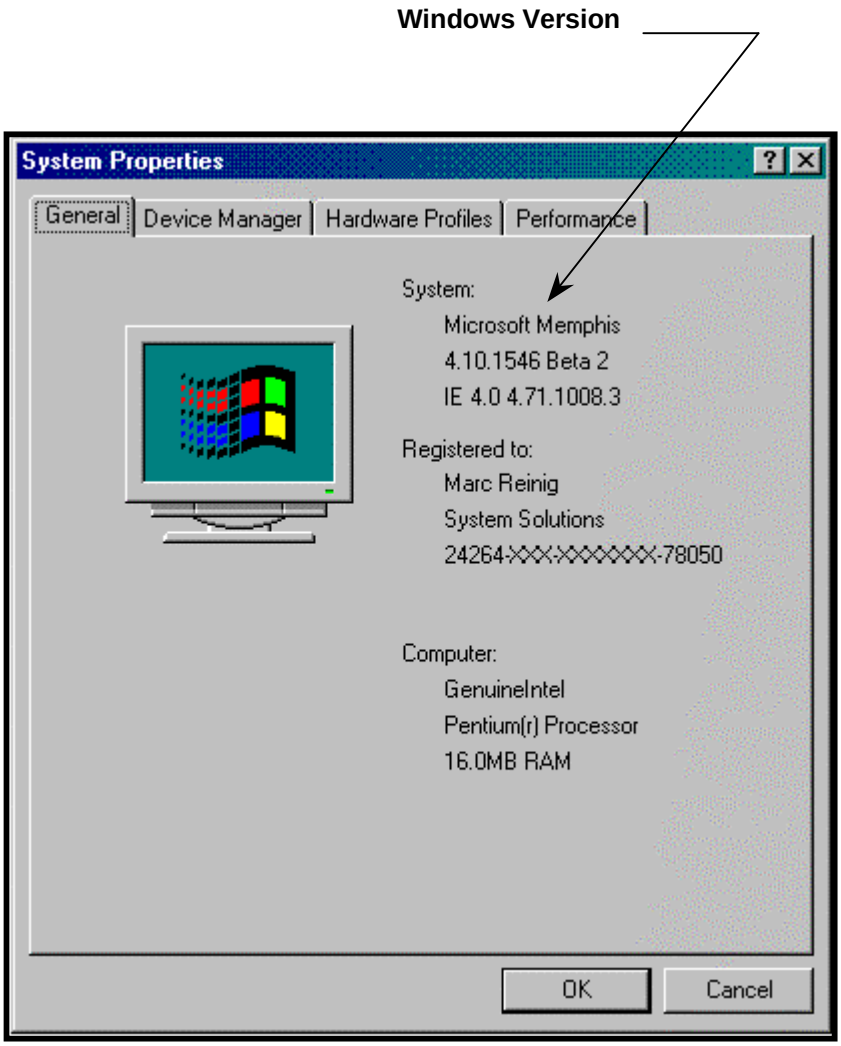


Figure K2 Windows System Properties

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 deleted from the "Install/Uninstall" option within the "Add/Remove Programs" screen.

- **Problem with system stability when a crystal is used with the Cypress CY7C63X0X family of USB controllers.**

  For system stability considerations, we highly recommend the use of ceramic resonator instead of crystal for the Cypress CY7C63X0X USB controllers. Crystals do not satisfy the startup and suspend/resume stability requirements of the CY7C63X0X USB controllers.

- **Memphis (Windows98 Beta X) is still a beta program**

  Because Memphis is still changing as it moves through its prerelease phase, releases subsequent to Beta 1 may not work well with the current product. If this occurs, Cypress will post new information, assembly code, drivers, or Windows applications (as appropriate).

- **Windows may ask for a USB device driver even if you have previously loaded it**

  If you attach a USB device to a USB *host* port to which you have not previously attached the device, Windows may ask for the USB device driver.

  This can be confusing if you have already attached the device to the other *host* port in the same system and loaded the driver. However, this is normal Windows behavior.

  Simply "Browse" to the Windows/System directory where the device driver is located and Windows will find it and not ask you again.

- **Hot Unplug problem with Windows98 (Memphis) Beta 2**

  If your system is running Memphis Beta 2, a hot unplug of the thermometer device will cause the operating system to crash ("blue screen"). Cypress is currently working on a solution to this problem. You can work around this problem by performing a Refresh in the Device Manager (under the Control Panel/System icon) prior to hot unplug. This will effectively unload the USB thermometer driver (please see next bullet).

- **Device Manager Refresh unloads USB thermometer driver**

  If you press the Refresh button on the Device Manager screen, the USB thermometer driver will unload (if it was loaded) or reload (if it was not loaded). Cypress is currently working on a solution to this problem. To work around this problem, do not refresh the Device Manager. If you must refresh the Device Manager, a second refresh will reload the thermometer driver.

- **A cold system boot will not automatically load the USB thermometer driver**

  If the system is rebooted, the USB thermometer driver will not automatically reload (even if the thermometer device is plugged into the USB). Cypress is currently working on a solution to this problem. There are two possible work-arounds. Once the system is up and running, either:

  1) Press the Refresh button under the Device Manager (please see previous bullet).
     -or-
  2) Hot unplug/replug the USB thermometer device. The driver will automatically reload.

Links to Other USB Documents

Datasheets:

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