CY4611B – *FX2LP* USB to ATA/CF Reference Design Notes

Introduction

Cypress has two USB 2.0 High Speed Mass Storage solutions. The AT2LP is a low-power fixed-function Mass Storage solution for ATA devices. The CY4611B is a flexible bridge solution that enables additional features to be added to a USB 2.0 bridge device. The Cypress EZ-USB *FX2LP* Mass Storage reference design connects the EZ-USB *FX2LP* to the following device types:

- IDE devices
  - 3.5", 2.5" Hard disk drives
- Compact Flash & micro drives
- ATAPI devices
  - ZIP drives
  - CD-ROM/R/RW drives
  - DVD-ROM/RAM/RW drives

Reference Design Contents

- FX2LP Mass Storage Reference Design PCBA
- Reference Design Schematic in both PDF and OrCAD source files
- Reference Design BOM
- Firmware source and object code
- Reference Design Notes (this document)
- UDMA White Paper
- Driver INF and .SYS files
- Operating Instructions
- Manufacturing tools
- Release Notes
- Errata

Background Information

You should be familiar with the USB Mass Storage Class specification and general operation of Cypress' EZ-USB *FX2LP* to get the most from this document. For more information please refer to these specifications or Cypress's EZ-USB *FX2LP* Technical Reference Manual.

Mass Storage Class Specification

The USB Mass Storage Class specification contains two subclasses, the CBI (Command, Bulk, Interrupt), and the newer Bulk Only Transport. This reference design complies with the Bulk Only subclass of the USB Mass Storage Specification. The Bulk Only subclass is supported by the Windows XP, 2000 and ME drivers as well as MacOS 9 and X. Cypress provides custom drivers for Windows and Macintosh operating systems to add support for security and SMART monitoring. The latest driver versions are available on the Cypress website.

Firmware Overview

Note: CBW, CSW, dataTransferLength, and "Persistent Stall" are defined in the "USB Mass Storage Class, Bulk Only Transport" document referenced below.

The firmware for the device is a straightforward implementation of a USB Bulk Only Mass Storage Device. After reset, it waits for a CBW packet, checks it and then executes the data phase of the command (if any). Once the data phase is complete, the firmware sends a CSW packet to the host. SETUP commands are handled in an ISR. A timer ISR is used to poll VBUS and GPIOs. The only commands that the firmware generates on its own are SCSI Identify Device (to get the device name) and ATA Identify Device (to get the device serial number).

The CY4611B firmware supports both high speed (480Mbps) and full speed (12Mbps) hosts.

Firmware Details

Refer to the flowchart on the following page for more details.

There are three main sections in the firmware:

- Initialization
- Command (CBW) processing
- ISRs

The initialization code sets up the hardware, reads the EEPROM configuration and detects the attached drives. Initialization routines include resetATAPIDevice, ATAInit, initUSB, TD_Init, detectSCSIvsATA, and ATAPIIdDevice. When the initialization code is complete, the hardware is set up, drives are fully enumerated, GPIF is loaded and the firmware is ready to accept CBW commands.

The CBW processing takes place in the TD_Poll() loop. This loop also polls the sleep flag to determine

if it is time for USB suspend. The final function of the main loop is to poll for new removable (CF) devices.

The ISRs have two main functions. They handle SETUP command processing and background polling for events like VBUS removal, GPIO changes and ATA_ENABLE changes.

Main()

This routine calls the TD_Init and ATAInit routines and then starts the master while(1) loop. The while(1) loop polls the sleep flag and calls TD_Poll, the main command processing routine.

ATAInit()

On a hard reset, TD_init() is called, which initializes the hardware using initUSB(). The resetATAPIDevice() routine is called to reset the drive. After the drive has been reset the drive discovery algorithm, detectSCSIvsATA(), is called once for the master device and again for the slave device.

detectSCSIvsATA()

The detectSCSIvsATA routine determines whether the attached device is IDE or ATAPI by reading the byte count registers. The scsi flag is set to 1 to indicate an ATAPI device, scsi is set to 0 on an IDE device. This routine sets the bDevicePresent flag when a drive is successfully detected.

ATAPIIdDevice()

This routine is called to collect information from the drive into internal data structures. This information includes the max PIO or UDMA speed supported and the serial number of the drive. If the device supports PIO-3, PIO-4 or UDMA, this routine will program the drive to run at the new speed.

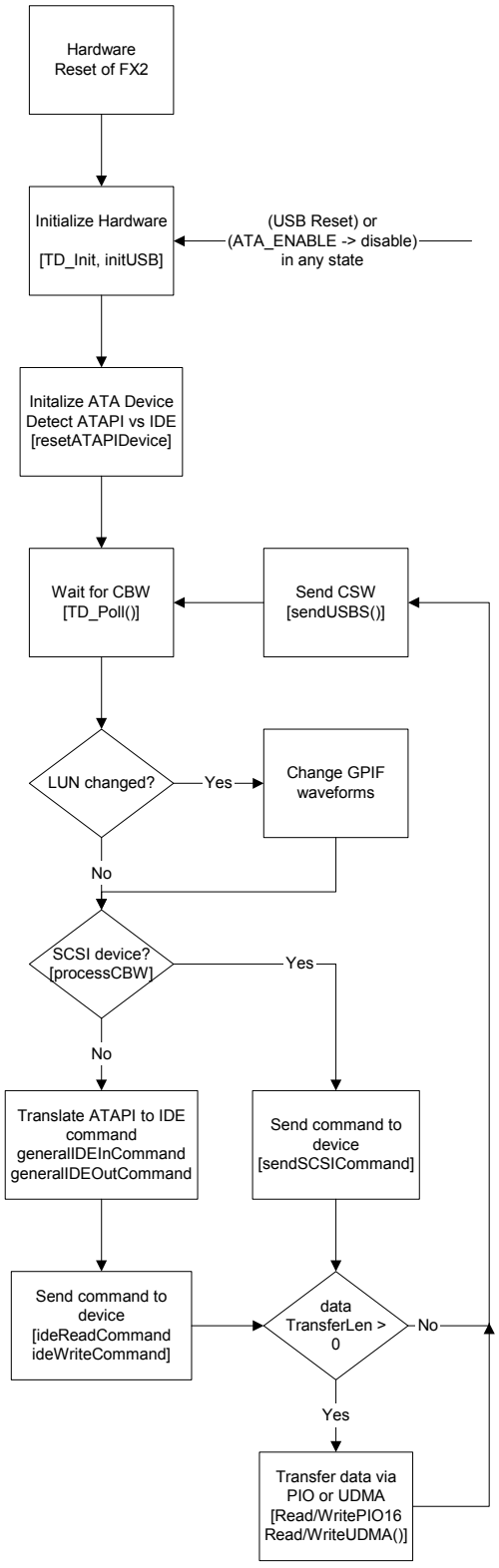


Figure 1: Overall program flow

TD_Poll()

As in all Cypress Frameworks based code, the main code loop is called TD_Poll(). This routine is called repeatedly until it detects a packet in the OUT buffer. TD_poll() checks the received packet for a valid CBW signature. If one is found, it calls processCBW(). If the packet is not a valid CBW, the device enters a "persistent stall" condition awaiting a device reset. ProcessCBW() checks to see if the LUN has changed from the previous command. If so, it reloads the GPIF with the proper waveforms for the new LUN. It then calls generalIDEInCommand() or generalIDEOutCommand() depending on the direction flag in the CBW. If the dataTransferLength is non-zero, the readPIO16() or writePIO16() routines are called to pass data directly from the USB buffers to the drive using the GPIF.

SETUP messages are handled in an ISR, so they may be received and responded to at any time. The entire SETUP message will be handled within the ISR, therefore long SETUP traffic will adversely affect disk performance. This is not expected to be an issue since Windows does not use SETUP packets after enumeration except to clear STALL conditions.

ReadPIO16(), WritePIO16(), ReadUDMA(), WriteUDMA()

These data transfer routines activate the GPIF to move data to/from the FIFO memory to/from the ATA bus. The data is read from the drive to the EP6 buffer. Write data moves from the host through the EP2 buffer.

Resets

The firmware performs a hard reset of the drive on a hard reset (power on). The firmware performs a soft reset of the 8051 and drives on a USB Reset or Mass Storage Class Reset.

File Descriptions

The FX2LP firmware is stored in its own directory. All of the FX2LP firmware is contained in the FX2LP source directory on the CD.

The purpose of the files in the **source** directory is shown in the following table:

Filename	Purpose
Dscr.a51	Descriptor table containing product/vendor ID, endpoint descriptions and other information reported to the host on startup.
memcmp.a51	Fast memory compare routine
reset.a51	Assembly routine used to branch to 0 on USB reset.
Startup.a51	Modified Keil startup file that does not initialize any variables.
USBjmptbl.a51	USB interrupt vector table and other fixed-address blocks including space allocation for EEPROM loader.
Atacb.c	Processes ATACB (ATA Command Block) requests. The optional ATACB facility allows IDE commands to be embedded within CBWs. This enables ANY command to be sent to the device. This facility is used to allow access to security features and SMART commands that do not have analogous SCSI commands.
atareset.c	Contains hard reset routine, selection of IDE vs ATAPI protocol. Identifies device characteristics, including serial number, capacity and transfer rate. Selects transfer rate by loading new GPIF waveforms. Contains timer ISR that checks VBUS and ATA_ENABLE.
fw.c	Frameworks based main routine. This fw.c has major differences from the fw.c released with the dev kit, since several implementation-specific functions have been merged with the general startup code in this file.
gpif.c	EZ-USB *FX2LP* low level i/o routines. Waveform descriptors. Routines for loading the GPIF memory with the waveform descriptors.
Globals.c, Globals2.c	Global variable definitions. The globals are split into two files to help the linker. The linker will place all of the variables in a .c file in a single block. Splitting the globals into two files allows some variables to be placed below the bit-addressable memory (at 0x20) and some to be placed above the bit-addressable memory.

Filename	Purpose
ide.c	Translates SCSI (ATAPI) commands sent by the host driver into IDE commands. Calls low-level transfer routines in gpif.c.
periph.c	TD_Init and TD_Poll(), misc init routines, misc util routines including our smaller version of memmove.
scsi.c	High level data transfer routines for ATAPI devices. (Named SCSI.c because ATAPI devices use the SCSI command set.) Calls low-level transfer routines in gpif.c.
Globals.h	Global variable references
atapi.h	Header file containing application specific items.
gpif.h	Header file containing hardware specific items.
scsi.h	SCSI command set
CY4611B.Opt	Options for UV2 project
CY4611B.hex	Output file from the linker. Combine.bat is used to merge this file with a configuration file like AT2_Legacy_pinout.iic to produce a full image like CY4611B_AT2_pinout.iic.
CY4611B.Uv2	UV2 project file
AT2_Legacy_pinout.iic	~200 byte configuration file produced by blaster.exe
cy4611b_AT2_PINOUT.iic	~16K EEPROM binary image

EEPROM configuration area

Many of the commonly changed items in the CY4611B configuration have been moved to a dedicated EEPROM configuration area. Locating these items in a dedicated area allows customization of many firmware attributes like Vendor ID and Product ID without compiling the firmware.
The format of this EEPROM configuration area is copied from the AT2LP and AT2. If the EEPROM configuration starts with 0x4d4d, the 56-pin package pinout will match the AT2 pinout. If the EEPROM configuration bytes 0 and 1 are 0x534b then the 56-pin pinout will match AT2LP. See the end of this document for a full description of the EEPROM config space format. A configuration tool is provided to assist you in creating and downloading your configuration file. This configuration utility (blaster.exe) can be found in the “manufacturing tools” directory on your CD. Blaster.exe can program or modify the EEPROM configuration on your CY4611B board.

The area allocated to EEPROM can be changed by modifying the value of CONFIG_SPACE_START in atapi.h and changing the –x argument passed to hex2bix. If your application needs more than 0x100 bytes of EEPROM configuration, several areas of the code will have to change. One area is sendDescriptor, which uses a BYTE offset within the EEPROM config space.

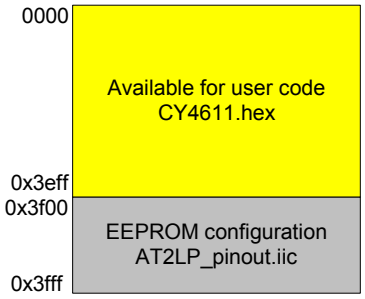


Figure 2: EEPROM memory map

The EEPROM settings can be programmed in two ways, interactively via blaster.exe or by creating a file. To program the EEPROM interactively, insert the “MFG mode” jumper and cycle power on your board. The board will bind to the manufacturing driver. You can then use the “write to device” button in blaster.exe to program your EEPROM.

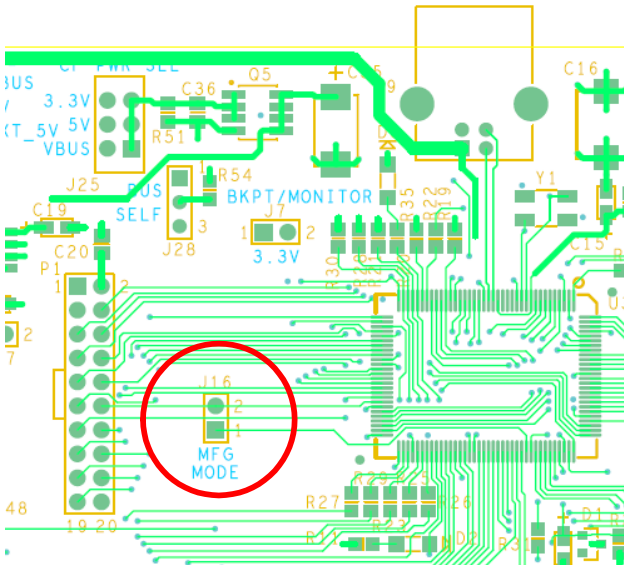


Figure 3: MFG Mode jumper location

The default build script will automatically build a unified image by combining cy4611b.hex with either AT2_legacy_pinout.iic or AT2LP_pinout.iic to produce CY4611B_AT2_PINOUT.iic or CY4611B_AT2LP_PINOUT.iic.

The configuration section of the EEPROM can also be modified without using the Keil tools. This enables you to modify some settings (like the Vendor ID and device name) without buying the Keil tools. Just use the combine.bat file on the CD to invoke hex2bix.exe.

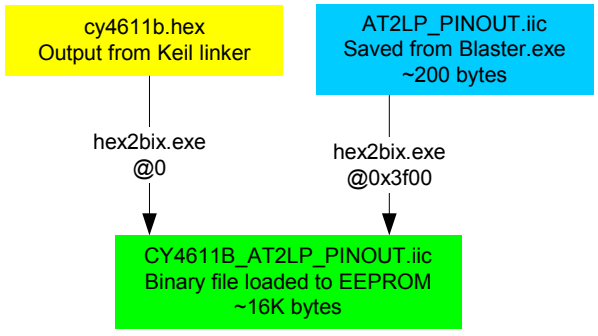


Figure 4: Output files

Compile Time Configuration Settings

The most common configuration settings are contained in the EEPROM configuration space described above. However, command line options and #defines control some of the compile time settings used by the code. The major ones are explained in the section below. To change some of these settings, right-click on the project name in uVision2 and select “options for target”, then select the C51 tab.

Atapi.h contains additional #defines that can be used to further customize the behavior of the firmware. These can be found in a section of atapi.h labeled “Configuration Settings”.

If you have one target platform, you will want to do your customization in atapi.h. Creating defines in the “options for target” tab allows you to create multiple targets with different #defines. This is useful if you have multiple targets (like debug vs production).

DEVICE_TYPE_IS_SCSI

Setting this flag to 0 will remove most of the ATAPI code from the EEPROM image. The resulting image will only work with hard drive type devices. Set this option to 0 to reduce code size.

Default Setting: 1

DEVICE_TYPE_IS_IDE

Setting this flag to 0 will remove most of the hard drive / CF code from the EEPROM image. The resulting image will only work with ATAPI devices. Set this option to 0 to reduce code size.

Default Setting: 1

REVC_4611_BOARD

The first 4611 board with compactFlash support changes several settings to make room for compactFlash. These changes include moving the interrupt input pin from PA0 to wakeup, and multiplexing the VBUS sense with ATA_RESET.

This pinout is not recommended for new designs.

Default Setting: 0

Serial numbers

The USB Mass Storage specification requires that each device have a unique serial number. Cypress provides manufacturing tools to program your device with a VID/PID and unique serial number. See the

"manufacturing tools" directory on the CD for more information.

Some customers want to read the serial number from the ATA or ATAPI device rather than from the EEPROM. The following flags enable this function in the firmware:

USE_ATA_DEVICE_SERIAL_NUMBER

For ATA devices, determines if the firmware uses the serial number reported by the device as the USB serial number. If TRUE, firmware uses the serial number reported by the device in response to the IDENTIFY_DEVICE command. If FALSE, the firmware sets the USB serial number index to 0 (i.e. no serial number string is reported in the device descriptor).

Default Setting: FALSE

USE_ATAPI_DEVICE_SERIAL_NUMBER

For ATAPI devices, determines if the firmware uses the serial number reported by the device as the USB serial number. If TRUE, firmware uses the serial number reported by the device in response to IDENTIFY_DEVICE command. If FALSE, the firmware sets the USB serial number index to 0 (i.e. no serial number string is reported in the device descriptor). Many ATAPI devices do not report a unique serial number. It is better to report no serial number than to report a non-unique serial number.

Default Setting: FALSE

NIBBLE_CONVERT_SERIAL_NUMBER
Determines if the firmware converts each nibble of the serial number reported by the device into a single character of the USB serial number. The Bulk-only mass storage class spec only allows HEX characters (0-9 and A-F) in the device serial number. Some devices report other ASCII characters. Converting each nibble into HEX assures spec compliance while maintaining the uniqueness of the serial number.

Default Setting: FALSE

Build Targets

There are two build targets for the CY4611B. They use exactly the same options for everything except one item. The CY4611B_AT2LP_PINOUT target includes the AT2LP_PINOUT.iic when it calls hex2bix.exe while the CY4611B_AT2_LEGACY_PINOUT Includes the AT2_LEGACY_PINOUT.iic configuration file.

Building the Software

Since the software is distributed on a CD, many operating systems will set the read-only flag when copying the data to your local directory. This flag must be turned off before uVision2 will properly build the .hex file. To do this, use "attrib –r *.*" at the DOS command line or select all of the files in Explorer, select "properties" and turn off the "read-only" checkbox in the "general" tab.

Once the files are no longer read-only, start the full uVision2 or uVision3 environment (available separately from www.keil.com) and click the "build all" button. This will generate an image that can be loaded with the control panel or the debugger. See below for more information on debugging.

The firmware in this Reference Design has only been tested with the release of the 3684 Dev Kit contained on the release CD. Please install the current Dev Kit before building.

The **hex2bix.exe** file in the software directory is newer than the version in the CY3684 install. Please use the version in the software directory.

You may have to delete the ezusb.lib file from your project and add it again to get the correct path in the Keil tools. The ezusb.lib file is installed at c:\cypress\usb\target\lib by default.

Note: This Reference Design is too large to compile with the 4K-demo version of the Keil tools that is shipped with Cypress' development kits.

Warnings

When the firmware is linked, it will generate three warnings. These warnings are expected. The linker may have to run several iterations to optimize the code and may generate this list two or three times (and report six or nine warnings).

EZUSB_Delay is called from the timer0 ISR and from the background code. This is not an issue because any calls to the EZUSB_Delay function in the ISR are followed by a soft reset.

The EEPROM read and write routines are only used by the ISR during manufacturing and debugging operations. The background code is not active during these operations.

```
*** WARNING L15: MULTIPLE CALL TO
FUNCTION
   NAME:    _EZUSB_DELAY/DELAY
   CALLER1: ?C_C51STARTUP
   CALLER2: ISRTIMER0/ATARESET
*** WARNING L15: MULTIPLE CALL TO
FUNCTION
   NAME:    _EEPROMWRITEBLOCK/EEPROM
   CALLER1: ?C_C51STARTUP
   CALLER2: ISR_SUDAV/PERIPH
*** WARNING L15: MULTIPLE CALL TO
FUNCTION
   NAME:    _EEPROMREAD/EEPROM
   CALLER1: ?C_C51STARTUP
   CALLER2: ISR_SUDAV/PERIPH
```

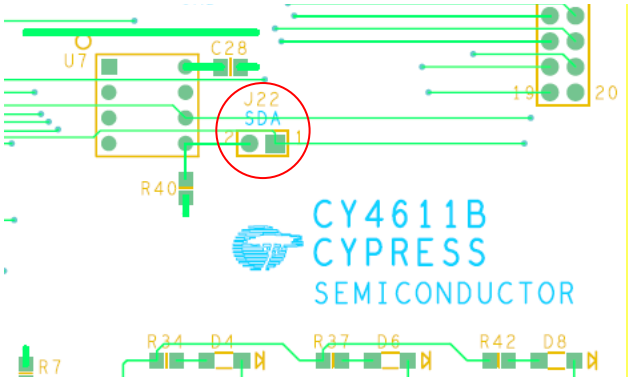
Using a CY3681 or CY3684 Board

The CY4611B software will run with the Keil debugger on the FX2LP development board (CY3684). This is a useful environment for debugging startup issues by single stepping the firmware. Note that the following changes to the 3684 board will be necessary:

1. Short solder points SP1, SP2 and SP3 to enable ATA pullups.
2. Remove JP2 to remove VBUS power to the board.
3. Connect VBUS (JP2 pin 2) to PA6 (P2 pin 13). Put a 10K pulldown on this signal. This gives the CPU the ability to sense VBUS.
4. Provide external 5v/ground to the board (JP2 pin 1 is a good 5v input). It will be very easy to use the board if you solder a disk drive connector to these pins.
5. Cut the Key pin (pin 20) on the ATA connector (P8)
6. Add a 10K pulldown on DD7 (pin 3 on the 40-pin connector).
7. Use the AT2_LEGACY configuration file.

Debugging without the Mass Storage Driver

Debugging specific commands requires a different approach because the Mass Storage driver will timeout while you are single stepping and may lock up or reboot the host machine. The CY4611B firmware can be bound to the Cyusb generic driver by following these steps:



1) Unplug the EEPROM jumper (J22)
2) Turn on power to your board. The board will enumerate and bind to the generic driver (CYUSB.sys).
3) Plug the EEPROM jumper back in so your code can access the config data in the EEPROM.
4) Open cyconsole and select options/EZ-USB interface
5) Hit the "Load Mon" button
6) Start the Keil debugger and download your firmware via the Keil debugger

Once the firmware is bound to the generic driver, commands can be sent to the device using the control panel. An easy way to do this is to construct a file containing the command and use the FileTrans button to send it.

1) Start the Keil debugger; download your firmware
2) Run the firmware, it will enumerate and bind to the general purpose driver
3) Start the control panel.
4) Do a "get pipes" on the control panel. This will fill in the pipe fields.
5) Select the OUT pipe and hit the FileTrans button.
6) Select your command file.
7) Manually transfer the IN or OUT data required by the command
8) Do a final IN to collect the CSW.

Difference between ATAPI and IDE devices

Although both ATAPI and IDE devices attach to the same 40 pin cable, they operate using different protocols, much like TCP/IP and NetBEUI share the same Ethernet wire, but cannot talk to each other. ATAPI commands are basically SCSI commands sent over an ATA interface.

This firmware will support **both** ATAPI and IDE task file commands. It will detect the type of device after reset. If the device is an IDE device, the ATAPI commands received over USB will be translated into IDE task file commands. One way to gain additional code space is to eliminate one of the supported protocols.

ATACB – ATA Command Block

The ATA Command Block (ATACB) feature enables the host to directly access the ATA register file on the device. This allows the host to send IDE commands that don't have direct SCSI translations. It also provides a powerful debug capability.

ATACB commands are transferred in the Command Block Wrapper Command Block (CBWCB) portion of the Command Block Wrapper (CBW) as shown below:

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11	*dCBWDataTransferLength*							
12	*bmCBWFlags*							
13	Reserved (0)				*bCBWLUN*			
14	Reserved (0)				*bCBWCBLength*			
15	*bVSCBSignature*							
16	*bVSCBSubCommand*							
17	*bmATACBActionSelect*							
18	*bmATACBRegisterSelect*							
19	*bATACBTransferBlockCount*							
20-27	*bATACBTaskFileWriteData*							
28-30	*Reserved*							

The ATACB is distinguished from other command blocks by having the first two bytes of the command block match the bVSCBSignature and bVSCBSubCommand values that are defined in the configuration area of the EEPROM. Only command blocks that have a valid bVSCBSignature and bVSCBSubCommand are interpreted as ATA Command Blocks. All other fields of the CBW and restrictions on the CBWCB remain as defined in the USB Mass Storage Class Bulk-Only Transport Specification. The ATACB must be 16 bytes in length. The following table and text defines the fields of the ATACB.

Windows Boot Support

The current level of boot functionality will allow you to boot to DOS or Win9x Safe Mode from a Hard Drive or CDROM. You cannot currently boot to Windows due to issues with the way Windows attempts to access a boot drive directly. Boot functionality has been tested with both Phoenix and AMI BIOS.

48 bit LBA Addressing

The ATA-6 spec contains support for large drives with 48-bit Logical Block Addresses (LBAs). This reference design supports the 48-bit addressing method. However, the SCSI commands passed by the Mass Storage Class Specification only support 32-bit LBAs, which limits support to 2^41 (2Tera) bytes on a 512-byte sectored device.

How this design uses GPIF

The *FX2LP* design takes advantage of its internal GPIF (General Programmable InterFace) to move data from the endpoint buffers to the mass storage device. For more details on the EZ-USB *FX2LP* and GPIF, see the EZ-USB *FX2LP* Technical Reference Manual and the UDMA white paper on this CD.

This design contains several GPIF waveforms:

- PIO-0
- PIO-3
- PIO-4
- Multi-word DMA
- UDMA/33
- UDMA/66

The firmware selects one of these waveforms based on the information returned by the device from the IDENTIFY DEVICE command. If there are two devices with different capabilities attached to the FX2LP, the firmware will reload the GPIF waveforms when the host addresses a different LUN.

References

USB Mass Storage Class – Bulk Only Transport, USB Mass Storage DWG. (www.usb.org)
USB Mass Storage Class – Overview Specification, USB Mass Storage DWG. (www.usb.org)
USB Specification – Revision 2.0, USB Implementers Forum. (www.usb.org)
EZ-USB *FX2LP* Technical Reference Manual, Revision 2.1, Cypress (www.cypress.com)
ATA/ATAPI-6 Specification, Proposed ANSI Standard (www.t13.org).
SCSI-3 Specification (www.t10.org)

ATACB format

Byte	Field Name	Field Description
0	bVSCBSignature	This field indicates to the CY7C68300B/CY7C68301B that the ATACB contains a vendor-specific command block. This value of this field must match the value in Config space offset 0x04 for this vendor-specific command to be recognized.
1	bVSCBSubCommand	This field must be set to 0x24 for ATACB commands.
2	bmATACBActionSelect	This field controls the execution of the ATACB according to the bitfield values:
		Bit 7 *IdentifyPacketDevice* This bit indicates that the data phase of the command will contain ATAPI (0xA1) or ATA (0xEC) IDENTIFY device data. Setting IdentifyPacketDevice when the data phase does not contain IDENTIFY device data will result in unspecified device behavior. 0 = Data phase does not contain IDENTIFY device data 1 = Data phase contains ATAPI or ATA IDENTIFY device data
		Bit 6 *UDMACommand* This bit enables supported UDMA device transfers. Setting this bit when a non-UDMA capable device is attached will result in undetermined behavior. 0 = Do not use UDMA device transfers (only use PIO mode) 1 = Use UDMA device transfers
		Bit 5 *DEVOverride* This bit determines whether the DEV bit value is taken from the value assigned to the LUN during start-up or from the ATACB. 0 = The DEV bit will be taken from the value assigned to the LUN during start-up 1 = The DEV bit will be taken from the ATACB field 0x0B, bit 4

		Bit 4 *DErrorOverride* This bit controls the device error override feature. This bit should not be set during a bmATACBActionSelect TaskFileRead. 0 = Data accesses are halted if a device error is detected 1 = Data accesses are not halted if a device error is detected
		Bit 3 *PErrorOverride* This bit controls the phase error override feature. This bit should not be set during a bmATACBActionSelect TaskFileRead. 0 = Data accesses are halted if a phase error is detected 1 = Data accesses are not halted if a phase error is detected
		Bit 2 *PollAltStatOverride* This bit determines whether or not the Alternate Status register will be polled and the BSY bit will be used to qualify the ATACB operation. 0 = The AltStat register will be polled until BSY=0 before proceeding with the ATACB operation 1 = The ATACB operation will be executed without polling the AltStat register.
		Bit 1 *DeviceSelectionOverride* This bit determines when the device selection will be performed in relation to the command register write accesses. 0 = Device selection will be performed prior to command register write accesses 1 = Device selection will be performed following command register write accesses
		Bit 0 *TaskFileRead* This bit determines whether or not the taskfile register data selected in bmATACBRegisterSelect is returned. If this bit is set, the dCBWDataTransferLength field must be set to 8. 0 = Execute ATACB command and data transfer (if any) 1 = Only read taskfile registers selected in bmATACBRegisterSelect and return 0x00h for all others. The format of the 12 bytes of returned data is as follows: • Address offset 0x00 (0x3F6) Alternate Status • Address offset 0x01 (0x1F1) Features / Error • Address offset 0x02 (0x1F2) Sector Count • Address offset 0x03 (0x1F3) Sector Number • Address offset 0x04 (0x1F4) Cylinder Low • Address offset 0x05 (0x1F5) Cylinder High • Address offset 0x06 (0x1F6) Device / Head • Address offset 0x07 (0x1F7) Command / Status
3	bmATACBRegisterSelect	This field controls which of the taskfile register read or write accesses occur. Taskfile read data will always be 8 bytes in length, and unselected register data will be returned as 0x00. Register accesses occur in sequential order as outlined below (0 to 7).
		Bit 0 *(0x3F6)* Device Control / Alternate Status
		Bit 1 *(0x1F1)* Features / Error
		Bit 2 *(0x1F2)* Sector Count
		Bit 3 *(0x1F3)* Sector Number
		Bit 4 *(0x1F4)* Cylinder Low
		Bit 5 *(0x1F5)* Cylinder High
		Bit 6 *(0x1F6)* Device / Head
		Bit 7 *(0x1F7)* Command / Status
4	bATACBTransferBlockCount	This value indicates the maximum requested block size in 512-byte incre ments. This value must be set to the last value used for the “Sectors per block” in the SET_MULTIPLE_MODE command. Legal values are 0, 1, 2, 4, 8, 16, 32, 64, and 128 where 0 indicates 256 sectors per block. A command failed status will be returned if an illegal value is used in the ATACB.

5-12	bATACBTaskFileWriteData	These bytes contain ATA register data used with ATA command or PIO write operations. Only registers selected in bmATACBRegisterSelect are required to hold valid data when accessed. The registers are as follows.
		ATACB Address Offset 0x05 *(0x3F6)* Device Control
		ATACB Address Offset 0x06 *(0x1F1)* Features
		ATACB Address Offset 0x07 *(0x1F2)* Sector Count
		ATACB Address Offset 0x08 *(0x1F3)* Sector Number
		ATACB Address Offset 0x09 *(0x1F4)* Cylinder Low
		ATACB Address Offset 0x0A *(0x1F5)* Cylinder High
		ATACB Address Offset 0x0B *(0x1F6)* Device
		ATACB Address Offset 0x0C *(0x1F7)* Command
13-15	Reserved	These bytes must be set to 0x00 for ATACB commands.

EEPROM configuration format

EEPROM Address	Field Name	Field Description	Required Contents	Suggested Contents
AT2LP Configuration				
0x00	I^2C EEPROM signature byte 0	I^2C EEPROM signature byte 0. This byte must be 0x53. For CY7C68300A compatibility mode, these bytes should be set to 0x4D4D.	0x53	
0x01	I^2C EEPROM signature byte 1	I^2C EEPROM signature byte 1. This byte must be 0x4B	0x4B	
0x02	APM Value	ATA Device Automatic Power Management Value. If an attached ATA device supports APM and this field contains other than 0x00, the AT2LP will issue a SET_FEATURES command to Enable APM with this value during the drive initialization process. Setting APM Value to 0x00 disables this functionality. This value is ignored with ATAPI devices.		0x00
0x03	Unused			0x80
0x04	bVSCBSignature Value	Value in the first byte of the CBW CB field that designates that the CB is to be decoded as vendor specific ATA commands instead of the ATAPI command block. See section 7 for more detail on how this byte is used.		0x24
0x05	Reserved	Bits (7:6)		0x07
	Enable mode page 8	Bit (5) Set to 1 to enable the write caching mode page (page 8). If this page is enabled, Windows will disable write caching by default which will limit write performance.		
	Disable wait for INTRQ	Bit (4) Set to 1 to poll status register rather than waiting for INTRQ. Setting this bit to 1 will improve USB BOT test results but may introduce compatibility problems with some devices.		
	BUSY Bit Delay	Bit (3) Enables a delay of up to 120 ms at each read of the DRQ bit where the device data length does not match the host data length. This allows the		

		CY7C68300B/CY7C68301B to work with most devices that incorrectly clear the BUSY bit before a valid status is present.		
	Short Packet Before Stall	Bit (2) Determines if a short packet is sent prior to the STALL of an IN endpoint. The USB *Mass Storage Class Bulk-Only Speci fication* allows a device to send a short or zero-length IN packet prior to returning a STALL handshake for certain cases. Certain host controller drivers may require a short packet prior to STALL. 1 = Force a short packet before STALL. 0 = Don't force a short packet before STALL.		
	SRST Enable	Bit (1) Determines if the firmware is to do an SRST reset during drive initialization. At least one reset must be enabled. Do not set SRST to 0 and Skip Pin Reset to 1 at the same time. 1 = Perform SRST during initialization. 0 = Don't perform SRST during initialization.		
	Skip Pin Reset	Bit (0) Skip ARESET# assertion. When this bit is set, the firmware will bypass ARESET# during any initialization other than power up. Do not set SRST to 0 and Skip Pin Reset to 1 at the same time. 0 = Allow ARESET# assertion for all resets. 1 = Disable ARESET# assertion except for power-on reset cycles.		
0x06	ATA UDMA Enable	Bit (7) Enable Ultra DMA data transfer support for ATAPI devices. If enabled, and if the ATAPI device reports UDMA support for the indicated modes, the firmware will utilize UDMA data transfers at the highest negotiated rate possible. 0 = Disable ATA device UDMA support. 1 = Enable ATA device UDMA support.		0xD4
	ATAPI UDMA Enable	Bit (6) Enable Ultra DMA data transfer support for ATAPI devices. If enabled, and if the ATAPI device reports UDMA support for the indicated modes, the FIRMWARE will utilize UDMA data transfers at the highest negotiated rate possible. 0 = Disable ATAPI device UDMA support. 1 = Enable ATAPI device UDMA support.		
	UDMA Modes	Bit (5:0) These bits select which UDMA modes, if supported, are enabled. Setting to 1 enables. Multiple bits may be set. The FIRMWARE will operate in the highest enabled UDMA mode supported by the device. The FIRMWARE supports UDMA modes 2, 3, and 4 only. Bit Descriptions 5 Reserved. Must be set to 0. 4 Enable UDMA mode 4. 3 Reserved. Must be set to 0. 2 Enable UDMA mode 2. 1 Reserved. Must be set to 0. 0 Reserved. Must be set to 0.		
0x07	Reserved	Bits(7:3) Must be set to 0.		0x07

	Multiword DMA mode PIO Modes	Bit (2) This bit selects multi-word DMA. If this bit is set and the drive supports it, multi-word DMA is used. Bits(1:0) These bits select which PIO modes, if supported, are enabled. Setting to 1 enables. Multiple bits may be set. The FIRMWARE will operate in the highest enabled PIO mode supported by the device. The FIRMWARE supports PIO modes 0, 3, and 4 only. PIO mode 0 is always enabled by internal logic. Bit Descriptions 1 Enable PIO mode 4. 0 Enable PIO mode 3.		
0x08	Pin Configurations BUTTON_MODE SEARCH_ATA_BUS	 Bit (7) Button mode. Set this bit to 1 to enable ATAPUEN, PWR500# and DRVPWRVLD to become button inputs returned on bits 2, 1, and 0 of EP1IN Bit (6) Enables a search performed at RESET to detect non-removable ATA and ATAPI devices. Systems with only a removable device (like CF readers) will set this bit to 0. Systems with one removable device and one non-removable device will set this bit to 1.		0x78
	BIG_PACKAGE	Bit (5) Package Select. Set this bit to 1 when using the 100-pin device.		
	ATA_EN DISKRDY Polarity HS Indicator Enable Drive Power Valid Polarity Drive Power Valid Enable	Bit (4) ATA sharing enable. Allows ATA bus sharing with other host devices. If ATA_EN=1 the ATA interface will be driven when VBUS_ATA_ENABLE is LOW. If ATA_EN=0 the ATA interface will be placed into Hi-Z state whenever VBUS_ATA_ENABLE is LOW. ‘0’ =ATA signals Hi-Z when VBUS_ATA_ENABLE is LOW. ‘1’ = ATA signals driven when VBUS_ATA_ENABLE is LOW. Bit (3) DISKRDY active polarity. ‘0’ = Active LOW polarity. ‘1’ =Active HIGH polarity. Bit (2) Enables GPIO2_nHS pin to indicate the current operating speed of the device (if output is enabled). ‘0’ = Normal GPIO operation. ‘1’ = High-speed indicator enable. Bit (1) Controls the polarity of DRVPWRVLD pin ‘0’ =Active LOW (“connector ground” indication) ‘1’ =Active HIGH (power indication from device) Bit (0) Enable for the DRVPWRVLD pin. When this pin is enabled, the FIRMWARE will enumerate a removable IDE device (normally CompactFlash) as the master device. ‘0’ =pin disabled (most systems) ‘1’= pin enabled (CompactFlash systems)		
0x09	Reserved	Bits (7:6)		0x00

	General Purpose IO Pin Output Enable	Must be set to zero. Bits (5:0) GPIO[5:0] Hi-Z control. '0' = Output enabled (GPIO pin is an output). '1' = Hi-Z (GPIO pin is an input).		
0x0A	Reserved General Purpose IO Pin Data	Bits (7:6) Must be set to zero. Bits (5:0) If the output enable bit is set, these bits select the value driven on the GPIO pins.		0x00
0x0B	Identify Device String Pointer LUN0	If this value is 00, the Identify Device data will be taken from the device. If this string is non-zero, it is used as a pointer to a 24 byte ASCII (non-Unicode) string in the EEPROM. This string will be used as the device identifier. This string is used by many operating systems as the user-visible name for the device.		0x00
0x0C	Identify Device String Pointer LUN1			0x00
0x0D	Delay after reset	Number of 20-ms ticks to wait between RESET and attempting to access the drive.		0x00
0x0E	Reserved Enable CF UDMA Fixed number of logical units = 2 Fixed number of logical units = 1 Search ATA on VBUS removed	Bits (7:4) Bit (3) '1' = Allow UDMA to be used with removable-media devices '0' = UDMA will not be used with removable-media devices Some CF devices will interfere with UDMA if the UDMA lines are connected to them. This bit tells the FIRMWARE if the UDMA lines are connected to the removable-media device. Bit (2) If bits 1 and 2 are both 0, the number of logical units will be determined by searching the ATA and CF buses for devices. Bit (1) If bits 1 and 2 are both 0, the number of logical units will be determined by searching the ATA and CF buses for devices. Bit (0) Search for ATA devices when VBUS returns. If this bit is set, the ATA bus will be searched for ATA devices every time FIRMWARE is plugged into a computer.		0x00
0x0F	Reserved	Must be set to 0x00.	0x00	
Device Descriptor				
0x10	bLength	Length of device descriptor in bytes.	0x12	
0x11	bDescriptor Type	Descriptor type.	0x01	
0x12	bcdUSB (LSB)	USB Specification release number in BCD.	0x00	
0x13	bcdUSB (MSB)		0x02	
0x14	bDeviceClass	Device class.	0x00	
0x15	bDeviceSubClass	Device subclass.	0x00	
0x16	bDeviceProtocol	Device protocol.	0x00	
0x17	bMaxPacketSize0	USB packet size supported for default pipe.	0x40	
0x18	idVendor (LSB)	Vendor ID. Cypress's Vendor ID may only be used for evalu ation purposes, and not in released products.		Your Vendor ID
0x19	idVendor (MSB)			
0x1A	idProduct (LSB)	Product ID.		Your

				Product ID
0x1B	idProduct (MSB)			
0x1C	bcdDevice (LSB)	Device release number in BCD LSB (product release number).		Your release number
0x1D	bcdDevice (MSB)	Device release number in BCD MSB (silicon release number).		
0x1E	iManufacturer	Index to manufacturer string. This entry must equal half of the address value where the string starts or 0x00 if the string does not exist.		0x53
0x1F	iProduct	Index to product string. This entry must equal half of the address value where the string starts or 0x00 if the string does not exist.		0x69
0x20	iSerialNumber	Index to serial number string. This entry must equal half of the address value where the string starts or 0x00 if the string does not exist. The USB *Mass Storage Class Bulk-Only Transport Specification* requires a unique serial number (in upper case, hexadecimal characters) for each device.		0x75
0x21	bNumConfigurations	Number of configurations supported. 1 for mass storage: 2 for HID: 3 for CSM		0x03
Device Qualifier				
0x22	bLength	Length of device descriptor in bytes.	0x0A	
0x23	bDescriptor	Type Descriptor type.	0x06	
0x24	bcdUSB (LSB)	USB Specification release number in BCD.	0x00	
0x25	bcdUSB (MSB)	USB Specification release number in BCD.	0x02	
0x26	bDeviceClass	Device class.	0x00	
0x27	bDeviceSubClass	Device subclass.	0x00	
0x28	bDeviceProtocol	Device protocol.	0x00	
0x29	bMaxPacketSize0	USB packet size supported for default pipe.	0x40	
0x2A	bNumConfigurations	Number of configurations supported.	0x01	
0x2B	bReserved	Reserved for future use. Must be set to zero.	0x00	
Configuration Descriptor				
0x2C	bLength	Length of configuration descriptor in bytes.	0x09	
0x2D	bDescriptorType	Descriptor type.	0x02	
0x2E	bTotalLength (LSB)	Number of bytes returned in this configuration. This includes the configuration descriptor plus all the interface and endpoint descriptors.	0x20	
0x2F	bTotalLength (MSB)		0x00	
0x30	bNumInterfaces	Number of interfaces supported.		0x01
0x31	bConfiguration Value	The value to use as an argument to Set Configuration to select the configuration. This value must be set to 0x01.	0x01	
0x32	iConfiguration	Index to the configuration string. This entry must equal half of the address value where the string starts, or 0x00 if the string does not exist.		0x00
0x33	bmAttributes	Device attributes for this configuration. Bit (7) Reserved. Must be set to 1. Bit (6) Self-powered. Must be set to 1. Bit (5) Remote wake-up. Must be set to 0. Bits (40) Reserved. Must be set to 0.	0xC0	
0x34	bMaxPower	Maximum power consumption for this configuration. Units used are mA*2 (i.e., 0x31 = 98 mA, 0xF9 = 498 mA). 0x00 reported for self-powered devices.		0x01

		Note: A value of 0x00 or 0x01 results in the 56-pin package configuring itself for self-powered mode, whereas a value greater than 0x01 results in the 56-pin package reporting itself as bus-powered. This is regardless of what address 0x33 is set to reflect in the 56-pin package.		
Interface and Endpoint Descriptors				
Interface Descriptor				
0x35	bLength	Length of interface descriptor in bytes.	0x09	
0x36	bDescriptorType	Descriptor type.	0x04	
0x37	bInterfaceNumber	Interface number.	0x00	
0x38	bAlternateSetting	Alternate setting.	0x00	
0x39	bNumEndpoints	Number of endpoints.	0x02	
0x3A	bInterfaceClass	Interface class.	0x08	
0x3B	bInterfaceSubClass	Interface subclass.		0x06
0x3C	bInterfaceProtocol	Interface protocol.	0x50	
0x3D	iInterface	Index to first interface string. This entry must equal half of the address value where the string starts or 0x00 if the string does not exist.		0x00
USB Bulk Out Endpoint				
0x3E	bLength	Length of this descriptor in bytes.	0x07	
0x3F	bDescriptorType	Endpoint descriptor type.	0x05	
0x40	bEndpointAddress	This is an Out endpoint, endpoint number 2.	0x02	
0x41	bmAttributes	This is a bulk endpoint.	0x02	
0x42	wMaxPacketSize (LSB)	Max data transfer size. To be set by speed (Full speed 0x0040; High speed 0x0200)		0x00
0x43	wMaxPacketSize (MSB)			0x02
0x44	bInterval	High-speed interval for polling (maximum NAK rate). Set to zero for full speed.	0x00	
USB Bulk In Endpoint				
0x45	bLength	Length of this descriptor in bytes.	0x07	
0x46	bDescriptorType	Endpoint descriptor type.	0x05	
0x47	bEndpointAddress	This is an In endpoint, endpoint number 8.	0x88	
0x48	bmAttributes	This is a bulk endpoint.	0x02	
0x49	wMaxPacketSize (LSB)	Max data transfer size. Automatically set by AT2 (Full speed 0x0040; High speed 0x0200)		0x00
0x4A	wMaxPacketSize (MSB)			0x02
0x4B	bInterval	High-speed interval for polling (maximum NAK rate). Set to zero for full speed.	0x00	
(Optional) HID Interface Descriptor				
0x4C	bLength	Length of HID interface descriptor	0x09	
0x4D	bDescriptorTypes	Interface descriptor type	0x04	
0x4E	bInterfaceNumber	Number of interfaces (2)	0x02	
0x4F	bAlternateSetting	Alternate setting	0x00	
0x50	bNumEndpoints	Number of endpoints used by this interface	0x01	
0x51	bInterfaceClass	Class code	0x03	
0x52	bInterfaceSubClass	Sub class	0x00	
0x53	bInterfaceSubSubClass	sub sub class	0x00	
0x54	iInterface	Index of string descriptor	0x00	

USB Interrupt In Endpoint				
0x5E	bLength	Length of this descriptor in bytes.	0x07	
0x5F	bDescriptorType	Endpoint descriptor type.	0x05	
0x60	bEndpointAddress	This is an In endpoint, endpoint number 1.	0x81	
0x61	bmAttributes	This is an interrupt endpoint.	0x03	
0x62	wMaxPacketSize (LSB)	Max data transfer size.	0x02	
0x63	wMaxPacketSize (MSB)		0x00	
0x64	bInterval	Interval for polling (max. NAK rate).		0x10
(Optional) HID Descriptor				
0x55	bLength	Length of HID descriptor	0x09	
0x56	bDescriptorType	Descriptor Type HID	0x21	
0x57	bcdHID (LSB)	HID Class Specification release number (1.10)	0x10	
0x58	bcdHID (MSB)		0x01	
0x59	bCountryCode	Country Code	0x00	
0x5A	bNumDescriptors	Number of class descriptors (1 report descriptor)	0x01	
0x5B	bDescriptorType	Descriptor Type	0x22	
0x5C	wDescriptorLength (LSB)	Length of HID report descriptor	0x22	
0x5D	wDescriptorLength (MSB)		0x00	
Terminator Descriptors				
0x65	Terminator		0x00	
(Optional) HID Report Descriptor				
0x66	Usage_Page	Vendor defined - FFA0	0x06	
0x67			0xA0	
0x68			0xFF	
0x69	Usage	Vendor defined	0x09	
0x6A			0xA5	
0x6B	Collection	Application	0xA1	
0x6C			0x01	
0x6D	Usage	Vendor defined	0x09	
0x6E			0xA6	
Input Report				
0x6F	Usage	Vendor defined	0x09	
0x70			0xA7	
0x71	Logical_Minimum	-128	0x15	
0x72			0x80	
0x73	Logical_Maximum	127	0x25	
0x74			0x7F	
0x75	Report_Size	8 bits	0x75	
0x76			0x08	
0x77	Report_Count	2 fields	0x95	
0x78			0x02	
0x79	Input	Input (Data, Variable, Absolute)	0x81	
0x7A			0x02	
Output Report				
0x7B	Usage	Usage - vendor defined	0x09	
0x7C			0xA9	
0x7D	Logical_Minimum	Logical Minimum (-128)	0x15	

0x7E			0x80	
0x7F	Logical_Maximum	Logical Maximum (127)	0x25	
0x80			0x7F	
0x81	Report_Size	Report Size 8 bits	0x75	
0x82			0x08	
0x83	Report_Count	Report Count 2 fields	0x95	
0x84			0x02	
0x85	Output	Output (Data, Variable, Absolute	0x91	
0x86			0x02	
0x87		End Collection	0xC0	
(optional) Standard Content Security Interface Descriptor				
0x88	bLength	Byte length of this descriptor	0x09	
0x89	bDescriptorType	Interface Descriptor type	0x0D	
0x8A	bInterfaceNumber	Number of interface.	0x02	
0x8B	bAlternateSetting	Value used to select an alternate setting for the interface identified in prior field		
0x8C	bNumEndpoints	Number of endpoints used by this interface (excluding endpoint 0) that are CSM dependent	0x02	
0x8D	bInterfaceClass			0x0D
0x8E	bInterfaceSubClass	Must be set to zero	0x00	
0x8F	bInterfaceProtocol	Must be set to zero	0x00	
0x90	iInterface	Index of a string descriptor that describes this Interface		
Channel Descriptor				
0x91	bLength	Byte length of this descriptor	0x09	
0x92	bDescriptorType	channel descriptor type	0x22	
0x93	bChannelID	Number of the channel, must be a zero based value that is unique across the device		
0x94	bmAttributes	Bits(7:5) Must be set to 0.		
0x95		Bit (4:0) 0 = Not used 1 = Interface 2 = Endpoint 3...31 = Reserved values		
0x96	bRecipient	Identifier of the target recipient If Recipient type field of bmAttributes = 1 then bRecipient field is the bInterfaceNumber If Recipient type field of bmAttributes = 2 then bRecipient field is an endpoint address, where: D7: Direction (0 = Out, 1 = IN) D6...D4: reserved and set to zero D3...D0: Endpoint number		
0x97	bRecipientAlt	alternate setting for the interface to which this channel applies	0x00	
0x98	bRecipientLogicalUnit	Recipient Logical Unit		
0x99	bMethod	Index of a class-specific CSM descriptor That describes		

		one of the Content Security Methods (CSM) offered by the device		
0x9A	bMethodVariant	CSM Variant descriptor		
CSM Descriptor				
0x9B	bLength	Byte length of this descriptor	0x06	
0x9C	bDescriptorType	CSM Descriptor type	0x23	
0x9D	bMethodID	Index of a class-specific CSM descriptor that describes on of the Content Security Methods offered by the device.	0x01	
0x9E	iCSMDescriptor	Index of string descriptor that describes the Content Security Method		
0x9F	bcdVersion (LSB)	CSM Descriptor Version number	0x10	
0xA0	bcsVersion (MSB)		0x02	
0xA1	Terminator		0x00	
USB String DescriptorIndex 0 (LANGID)				
0xA2	bLength	LANGID string descriptor length in bytes.	0x04	
0xA3	bDescriptorType	Descriptor type.	0x03	
0xA4	LANGID (LSB)	Language supported. The CY7C68300B supports one LANGID value.		0x09
0xA5	LANGID (MSB)			0x04
USB String DescriptorManufacturer				
0xA6	bLength	String descriptor length in bytes (including bLength).		0x2C
0xA7	bDescriptorType	Descriptor type.	0x03	
0xA8	bString	Unicode character LSB.		“C” 0x43
0xA9	bString	Unicode character MSB.		0x00
0xAA	bString	Unicode character LSB.		“y” 0x79
0xAB	bString	Unicode character MSB.		0x00
0xAC	bString	Unicode character LSB.		“p” 0x70
0xAD	bString	Unicode character MSB.		0x00
0xAE	bString	Unicode character LSB.		“r” 0x72
0xAF	bString	Unicode character MSB.		0x00
0xB0	bString	Unicode character LSB.		“e” 0x65
0xB1	bString	Unicode character MSB.		0x00
0xB2	bString	Unicode character LSB.		“s” 0x73
0xB3	bString	Unicode character MSB.		0x00
0xB4	bString	Unicode character LSB.		“s” 0x73
0xB5	bString	Unicode character MSB.		0x00
0xB6	bString	Unicode character LSB.		“ ” 0x20
0xB7	bString	Unicode character MSB.		0x00
0xB8	bString	Unicode character LSB.		“S” 0x53
0xB9	bString	Unicode character MSB.		0x00
0xBA	bString	Unicode character LSB.		“e” 0x65
0xBB	bString	Unicode character MSB.		0x00
0xBC	bString	Unicode character LSB.		“m” 0x6D
0xBD	bString	Unicode character MSB.		0x00
0xBE	bString	Unicode character LSB.		“i” 0x69
0xBF	bString	Unicode character MSB.		0x00
0xC0	bString	Unicode character LSB.		“c” 0x63

0xC1	bString	Unicode character MSB.		0x00
0xC2	bString	Unicode character LSB.		“o” 0x6F
0xC3	bString	Unicode character MSB.		0x00
0xC4	bString	Unicode character LSB.		“n” 0x6E
0xC5	bString	Unicode character MSB.		0x00
0xC6	bString	Unicode character LSB.		“d” 0x64
0xC7	bString	Unicode character MSB.		0x00
0xC8	bString	Unicode character LSB.		“u” 0x75
0xC9	bString	Unicode character MSB.		0x00
0xCA	bString	Unicode character LSB.		“c” 0x63
0xCB	bString	Unicode character MSB.		0x00
0xCC	bString	Unicode character LSB.		“t” 0x74
0xCD	bString	Unicode character MSB.		0x00
0xCE	bString	Unicode character LSB.		“o” 0x6F
0xCF	bString	Unicode character MSB.		0x00
0xD0	bString	Unicode character LSB.		“r” 0x72
0xD1	bString	Unicode character MSB.		0x00
USB String DescriptorProduct				
0xD2	bLength	String descriptor length in bytes (including bLength).		0x2C
0xD3	bDescriptorType	Descriptor type.	0x03	
0xD4	bString	Unicode character LSB.		“U” 0x55
0xD5	bString	Unicode character MSB.		0x00
0xD6	bString	Unicode character LSB.		“S” 0x53
0xD7	bString	Unicode character MSB.		0x00
0xD8	bString	Unicode character LSB.		“B” 0x42
0xD9	bString	Unicode character MSB.		0x00
0xDA	bString	Unicode character LSB.		“2” 0x32
0xDB	bString	Unicode character MSB.		0x00
0xDC	bString	Unicode character LSB.		“.” 0x2E
0xDD	bString	Unicode character MSB.		0x00
0xDE	bString	Unicode character LSB.		“0” 0x30
0xDF	bString	Unicode character MSB.		0x00
0xE0	bString	Unicode character LSB.		“ ” 0x20
0xE1	bString	Unicode character MSB.		0x00
0xE2	bString	Unicode character LSB.		“D” 0x53
0xE3	bString	Unicode character MSB.		0x00
0xE4	bString	Unicode character LSB.		“i” 0x74
0xE5	bString	Unicode character MSB.		0x00
0xE6	bString	Unicode character LSB.		“s” 0x6F
0xE7	bString	Unicode character MSB.		0x00
0xE8	bString	Unicode character LSB.		“k” 0x72
0xE9	bString	Unicode character MSB.		0x00
USB String DescriptorSerial Number (**Note**: The USB Mass Storage Class specification requires a unique serial number in each device. Not providing a unique serial number can cause the operating system to crash. The serial number must be at least 12 characters, but some USB hosts will only treat the last 12 characters of the serial number as unique.)				
0xEA	bLength	String descriptor length in bytes (including bLength).		0x22
0XEB	bDescriptor Type	Descriptor type.	0x03	

0XEC	bString	Unicode character LSB.		“1” 0x31
0XED	bString	Unicode character MSB.		0x00
0XEE	bString	Unicode character LSB.		“2” 0x32
0XEF	bString	Unicode character MSB.		0x00
0XF0	bString	Unicode character LSB.		“3” 0x33
0xF1	bString	Unicode character MSB.		0x00
0xF2	bString	Unicode character LSB.		“4” 0x34
0xF3	bString	Unicode character MSB.		0x00
0xF4	bString	Unicode character LSB.		“5” 0x35
0xF5	bString	Unicode character MSB.		0x00
0xF6	bString	Unicode character LSB.		“6” 0x36
0xF7	bString	Unicode character MSB.		0x00
0xF8	bString	Unicode character LSB.		“7” 0x37
0xF9	bString	Unicode character MSB.		0x00
0xFA	bString	Unicode character LSB.		“8” 0x38
0xFB	bString	Unicode character MSB.		0x00
0xFC	bString	Unicode character LSB.		“9” 0x39
0xFD	bString	Unicode character MSB.		0x00
0xFE	bString	Unicode character LSB.		“0” 0x30
0xFF	bString	Unicode character MSB.		0x00
0X100	bString	Unicode character LSB.		“A” 0x41
0X101	bString	Unicode character MSB.		0x00
0X102	bString	Unicode character LSB.		“B” 0x42
0X103	bString	Unicode character MSB.		0x00
Identify Device String (**Note**: This is not a Unicode string. It is the ASCII string returned by the device in the Identify Device information. It is a fixed length (24 bytes). Changing this string may cause CD authoring software to incorrectly identify the device.)				
0X104	Device name byte 1	ASCII Character		“C” 0x43
0X105	Device name byte 2	ASCII Character		“y” 0x79
0X106	Device name byte 3	ASCII Character		“p” 0x70
0X107	Device name byte 4	ASCII Character		“r” 0x72
0X108	Device name byte 5	ASCII Character		“e” 0x65
0X109	Device name byte 6	ASCII Character		“s” 0x73
0X10A	Device name byte 7	ASCII Character		“s” 0x73
0X10B	Device name byte 8	ASCII Character		“ “ 0x20
0X10C	Device name byte 9	ASCII Character		“C” 0x43
0X10D	Device name byte 10	ASCII Character		“u” 0x75
0X10E	Device name byte 11	ASCII Character		“s” 0x73
0X10F	Device name byte 12	ASCII Character		“t” 0x74
0X110	Device name byte 13	ASCII Character		“o” 0x6f
0X111	Device name byte 14	ASCII Character		“m” 0x6d
0X112	Device name byte 15	ASCII Character		“ ” 0x20
0X113	Device name byte 16	ASCII Character		“N“ 0x4e
0X114	Device name byte 17	ASCII Character		“a“ 0x61
0X115	Device name byte 18	ASCII Character		“m“ 0x6d
0X116	Device name byte 19	ASCII Character		“e“ 0x65
0X117	Device name byte 20	ASCII Character		“ ” 0x20

0X118	Device name byte 21	ASCII Character		"L" 0x4c
0X119	Device name byte 22	ASCII Character		"U" 0x55
0X11A	Device name byte 23	ASCII Character		"N" 0x4e
0X11B	Device name byte 24	ASCII Character		"0" 0x30
0x11C to 0x1FF	Unused ROM Space	Amount of unused ROM space will vary depending on strings.		0xFF

Note: More than 0X100 bytes of configuration are shown for example only. The firmware only supports 0X100 total bytes. This is controlled by the CONFIG_SPACE_START value in atapi.h. CONFIG_SPACE_START must match the –x address passed to hex2bix when the files are combined. If more than 0x100 bytes of config space are used, the routines that access config space must be changed to use a WORD offset rather than a BYTE offset.

Document Revision History

Revision #	Date	Comments
2.50	4/5/05	Updated for release 2.50. Removed multiple targets, moved many options to EEPROM. Added CY3684 references.
2.30	7/5/02	Updated for release 2.30. Added compact flash, multiple device support.
2.20	5/1/02	Updated for release 2.20
2.10	2/1/01	Updated for new board, added flowchart
2.09	12/1/01	Updated for final release.
2.0B8	8/15/01	Minor typographical and technical corrections.
2.0B7	7/1/01	Added information about unified code image.
2.0B5	5/20/01	Added DVD support info
2.0B1	3/26/01	Revised for Beta release. Added more file descriptions Added build instructions
2.0	11/29/00	Initial Release