Semiconductors

UM10042_1

ISP1582 PCI Mass Storage Kit Firmware

September 2003

User's Guide

Rev. 1.0

Revision History:

Version	Date	Description	Author
1.0	August 2003	First version.	Guo Yang Bin

We welcome your feedback. Send it to wired.support@philips.com.

Philips Semiconductors - Asia Product Innovation Centre
Visit www.semiconductors.philips.com/buses/usb or www.flexiusb.com

PHILIPS

CONTENTS

FIGURES

TABLES

1. Overview

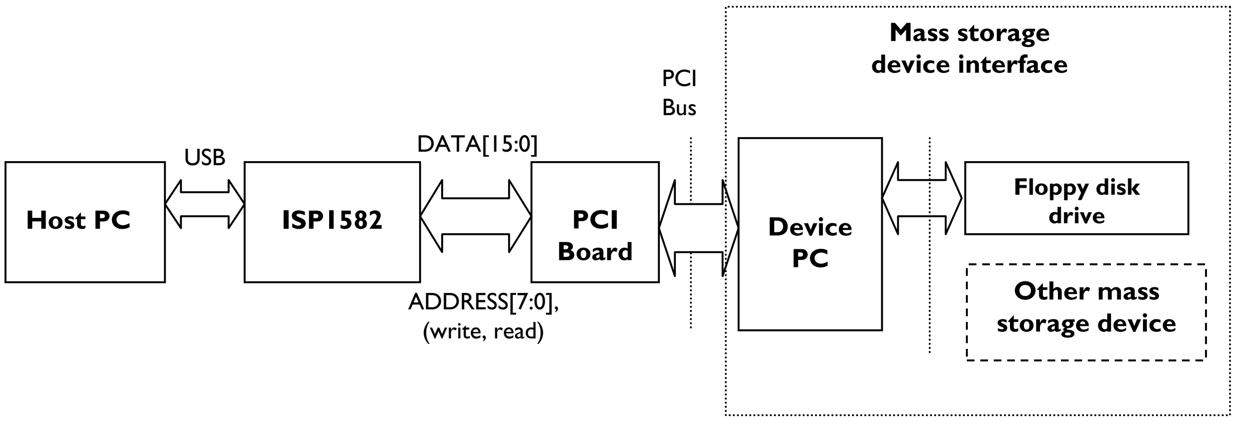
This application note will explain how to interface the ISP1582, a generic processor and a mass storage device to achieve a bridge solution. It will also illustrate the protocol and methodology of the bridge.

Note: This firmware is only for development purpose.

2. System structure

2.1. Block diagram of the bridge solution

The following figure illustrates the block diagram of the bridge solution.



Note: Philips Semiconductors will not fix any issues related to a floppy disk drive.

2.2. General description

2.2.1. ISP1582

In the bridge solution, the ISP1582 acts as a USB generic device chip. It is configured in the generic processor mode.

2.2.2. PCI board

The PCI board acts as a bridge from the ISP1582 to the device PC, by which the device PC can read and write data from the ISP1582 in the generic processor mode.

2.2.3. Device PC

In the bridge solution, the device PC acts as a generic processor and a bridge from the PCI to a PC mass storage device.

2.2.4. Mass storage device

The mass storage device is the end-level device in this solution. The current firmware can only operate a floppy disk drive. By changing the parameters, the firmware can support other standards PC mass storage devices.

2.2.5. Host PC

The role of the host PC is to act as a USB host.

3. Standard USB descriptor

The device supports the following standard USB descriptors:

- **Device**: Each USB device has one device descriptor (as specified in *Universal Serial Bus Specification Rev. 2.0*).
- **Configuration**: Each USB device has one default configuration descriptor, which supports at least one interface.
- **Interface**: The device supports at least one interface—the Bulk-Only data interface. Some devices may support additional interfaces to provide other capabilities.
- **Endpoint**: The device supports the following endpoints, in addition to the default pipe that is a requirement for all USB devices:
    - Bulk-In endpoint
    - Bulk-Out endpoint

  Some devices may support additional endpoints to provide other capabilities. The host uses the first reported Bulk-In and Bulk-Out endpoints for the selected interface.
- **String**: The device supplies a unique serial number.

This specification defines no class-specific descriptors.

The rest of this section describes the standard USB device, configuration, interface, endpoint and string descriptors for the device. For more information about these and other standard descriptors, refer to Chapter 9, "USB Device Framework," of the USB Specification.

Note: The following table notes appear as "Information in this table is from the *USB Specification version 1.1*". This is because they are extracted from *USB Mass Storage Class Bulk-only Transport Specification*, which still refers to version 1.1 of the USB Specification.

3.1. Device descriptor

Each USB device has one device descriptor (as set in the USB specification). The device specifies the device class and subclass codes in the interface descriptor, and not in the device descriptor.

Table 3-1: Device descriptor

Offset	Field	Size	Value	Description
0	*bLength*	Byte	12h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	01h	DEVICE descriptor type.
2	*bcdUSB*	Word	????h	*USB Specification* Release Number in Binary-Coded Decimal (i.e. 2.10 = 210h). This field identifies the release of the *USB Specification* with which the device and is descriptors are compliant.
4	*bDeviceClass*	Byte	00h	**Class is specified in the interface descriptor.**
5	*bDeviceSubClass*	Byte	00h	**Subclass is specified in the interface descriptor.**
6	*bDeviceProtocol*	Byte	00h	**Protocol is specified in the interface descriptor.**
7	*bMaxPacketSize0*	Byte	??h	Maximum packet size for endpoint zero. (only 8, 16, 32, or 64 are valid (**08h, 10h, 20h, 40h**)).
8	*idVendor*	Word	????h	Vendor ID (assigned by the USB-IF).
10	*idProduct*	Word	????h	Product ID (assigned by the manufacturer).
12	*bcdDevice*	Word	????h	Device release number in binary-coded decimal.
14	*iManufacturer*	Byte	??h	Index of string descriptor describing the manufacturer.
15	*iProduct*	Byte	??h	Index of string descriptor describing this product.
16	*iSerialNumber*	Byte	??h	Index of string descriptor describing the device's serial number. **(Details in 4.1.1 below)**
17	*bNumConfigurations*	Byte	??h	Number of possible configurations.

NOTE: Information in this table is from the *USB Specification version 1.1 table 9-7*. **Bold text** has been added for clarifications when using these descriptors with this specification.

3.2. Configuration descriptor

Table 3-2: Configuration descriptor

Offset	Field	Size	Value	Description
0	*bLength*	Byte	09h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	02h	CONFIGURATION Descriptor Type.
2	*wTotalLength*	Word	????h	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class- or vendor-specific) returned for this configuration.
4	*bNumInterfaces*	Byte	??h	Number of interfaces supported by this configuration. **The device shall support at least the Bulk-Only Data Interface.**
5	*bConfigurationValue*	Byte	??h	Value to use as an argument to the *SetConfiguration()* request to select this configuration.
6	*iConfiguration*	Byte	??h	Index of string descriptor describing this configuration.
7	*bmAttributes*	Byte	?0h	Configuration characteristics: **Bit** **Description** 7 Reserved (set to one) 6 Self-powered 5 Remote Wakeup 4..0 Reserved (reset to zero) **Bit** 7 is reserved and must be set to one for historical reasons. **For a full description of this *bmAttributes* bitmap, see the *USB 1.1 Specification*.**
8	*MaxPower*	Byte	??h	Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2mA units (i.e. 50 = 100mA)

NOTE: Information in this table is from the *USB Specification version 1.1 table 9-8*. **Bold text** has been added for clarifications when using these descriptors with this specification.

3.3. Interface descriptor

The device supports at least one interface—Bulk-Only Data Interface, which uses three endpoints.

Composite mass storage devices may support additional interfaces, to provide other features, such as audio or video capabilities. This specification does not define such interfaces.

The interface may have multiple alternate settings. The host examines each of the alternate settings to look for the bInterfaceProtocol and bInterfaceSubClass it optimally supports.

3.3.1. Endpoint descriptor

The device supports at least three endpoints: Control, Bulk-In and Bulk-Out. Each USB device defines a Control endpoint (endpoint 0), which is the default endpoint and it does not require a descriptor.

3.3.2. Bulk-In endpoint

The Bulk-In endpoint is used for transferring data and status from the device to the host.

Table 3-3: Bulk-In endpoint descriptor

Offset	Field	Size	Value	Description
0	*bLength*	Byte	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	05h	ENDPOINT Descriptor Type.
2	*bEndpointAddress*	Byte	8?h	The address of this endpoint on the USB device. The address is encoded as follows: **Bit** **Description** 3..0 The endpoint number 6..4 Reserved, set to 0 7 1 = In
3	*bmAttributes*	Byte	02h	This is a Bulk endpoint.
4	*wMaxPacketSize*	Word	00??h	Maximum packet size. Shall be 8, 16, 32 or 64 bytes (08h, 10h, 20h, 40h).
6	*bInterval*	Byte	00h	Does not apply to Bulk endpoints.

3.3.3. Bulk-Out endpoint

The Bulk-Out endpoint is used for transferring commands and data from the host to the device.

Table 3-4: Bulk-Out endpoint descriptor

Offset	Field	Size	Value	Description
0	*bLength*	Byte	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	05h	ENDPOINT descriptor type.
2	*bEndpointAddress*	Byte	0?h	The address of this endpoint on the USB device. This address is encoded as follows: **Bit** **Description** 3..0 Endpoint number 6..4 Reserved, set to 0 7 0 = Out
3	*bmAttributes*	Byte	02h	This is a Bulk endpoint.
4	*wMaxPacketSize*	Word	00??h	Maximum packet size. Shall be 8, 16, 32 or 64 bytes (08h, 10h, 20h, or 40h).
6	*bInterval*	Byte	00h	Does not apply to Bulk endpoints.

4. USB mass storage protocol

4.1. USB mass storage Bulk-Only transport

This specification addresses the Bulk-Only transport, that is, transport of command, data, and status occurring solely using the bulk endpoints. The specification uses only the default pipe to stall condition on bulk endpoints and to issue class-specific requests. It does not require the use of interrupt endpoint.

It also defines support for logical units that share common device characteristics. Although this feature provides support necessary to allow mass storage devices to share a common USB interface descriptor, it is not meant for implementing interface bridge devices.

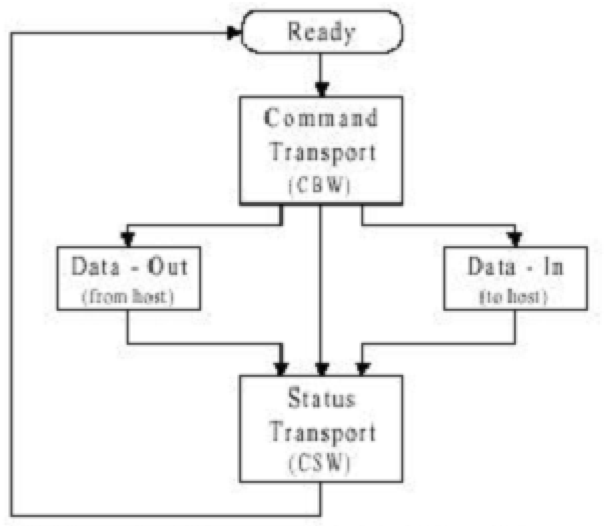


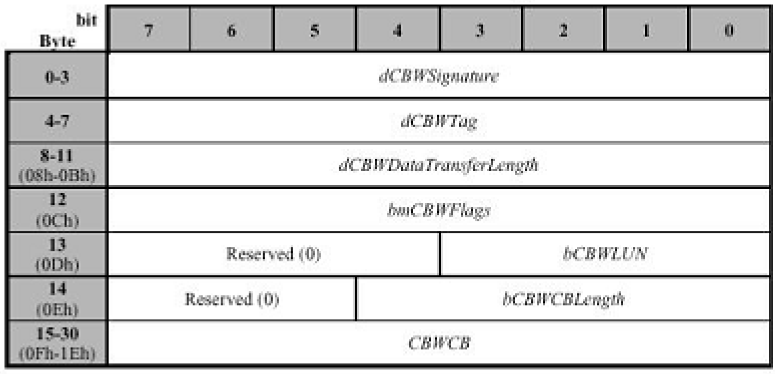
Figure 4-1: Command, data and status flow

Figure 4-1 shows the command, data and status flow of the Bulk-Only transport. All command, data and status stage transfers are done through the bulk endpoint. The ATAPI packet commands are embedded in the Command Block Wrapper (CBW) that the microcontroller or microprocessor has to decipher.

4.2. Command Block Wrapper

The Command Block Wrapper (CBW) starts on a packet boundary and ends as a short packet with exactly 31 (1FH) bytes transferred. Fields appear aligned to byte offsets equal to a multiple of their byte size. All subsequent data and Command Status Wrapper (CSW) start at a new packet boundary. All the CBW transfers are ordered with the LSB (byte 0) first (little endian).

Table 4-1: Command Block Wrapper



bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11 (08h-0Bh)	*dCBWDataTransferLength*							
12 (0Ch)	*bmCBWFlags*							
13 (0Dh)	Reserved (0)				*bCBWLUN*			
14 (0Eh)	Reserved (0)			*bCBWCBLength*				
15-30 (0Fh-1Eh)	*CBWCB*							

4.2.1. dCBWSignature

A signature that helps to identify this data packet as a CBW. The signature field contains the value 43425355H (little endian), indicating a CBW.

4.2.2. dCBWTag

A Command Block Tag sent by the host. The device echoes the contents of this field back to the host in the dCSWTag field of the associated CSW. The dCSWTag field positively associates a CSW with the corresponding CBW.

4.2.3. dCBWDataTransferLength

The number of data bytes that the host expects to transfer on the Bulk-In or Bulk-Out endpoint (as indicated by the Direction bit) during the execution of this command. If this field is zero, the device and the host do not transfer data between the CBW and the associated CSW, and the device ignores the value of the Direction bit in bmCBWFlags.

4.2.4. bmCBWFlags

The bits of this field are defined as follows:

Bit	Description
7	**Direction**: The device ignores this bit if the dCBWDataTransferLength field is zero, otherwise: **0**—Data-Out from the host to the device **1**—Data-In from the device to the host.
6	**Obsolete**: The host sets this bit to zero.
5 to 0	**Reserved**. The host sets these bits to zero.

4.2.5. bCBWLUN

The device Logical Unit Number (LUN) to which the command block is sent. For devices that support multiple LUNs, the host places into this field the LUN to which this command block is addressed. Otherwise, the host sets this field to zero.

4.2.6. bCBWCBLength

The valid length of CBWCB in bytes. This defines the valid length of the command block. The only legal values are 1 through 16 (01H through 10H). All other values are reserved.

4.2.7. CBWCB

The command block executed by the device. The device interprets the first bCBWCBLength bytes in this field as a command block as defined by the command set identified by bInterfaceSubClass. If the command set supported by the device uses command blocks of fewer than 16 (10H) bytes in length, the significant bytes are transferred first, beginning with the byte at offset 15 (FH). The device ignores the content of the CBWCB field past the byte at offset (15 + bCBWCBLength – 1).

4.3. Command Status Wrapper

The Command Status Wrapper (CSW) starts on a packet boundary and ends as a short packet with exactly 13 (0DH) bytes transferred. Fields appear aligned to byte offsets equal to a multiple of their byte size. All CSW transfers are ordered with the LSB (byte 0) first (little endian). For details, refer to the *Universal Serial Bus Specification Rev 2.0* Terms and Abbreviations.

Table 4-2: Command Status Wrapper

bit Byte	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11 (8-Bh)	*dCSWDataResidue*							
12 (Ch)	*bCSWStatus*							

4.3.1. dCSWSignature

A signature that helps identify this data packet as a CSW. The signature field contains the value 53425355H (little endian), indicating CSW.

4.3.2. DCSWTag

The device sets this field to the value received in the dCBWTag of the associated CBW.

4.3.3. dCSWDataResidue

For Data-Out, the device reports the difference between the amounts of data expected as stated in the dCBWDataTransferLength field and the actual amount of data processed by the device in dCSWDataResidue.

For Data-In, the device reports the difference between the amount of data expected as stated in the dCBWDataTransferLength field and the actual amount of relevant data sent by the device in dCSWDataResidue. The dCSWDataResidue field must not exceed the value sent in the dCBWDataTransferLength field.

4.3.4. bCSWStatus

bCSWStatus indicates the success or failure of the command. The device sets this byte to zero if the command is successfully completed. A non-zero value indicates a failure during command execution; see Table 4-3.

Table 4-3: Command Block Status Value

Value	Description
00h	Command Passed ("good status")
01h	Command Failed
02h	Phase Error
03h and 04h	Reserved (Obsolete)
05h to FFh	Reserved

4.4. Data transfer condition

This section describes how the host and the device remain synchronized.

The host indicates the expected transfer in the CBW by using the Direction bit and the dCBWDataTransferLength field. The device then determines the actual direction and data transfer length. The device responds by transferring data, STALLing endpoints when specified, and returning the appropriate CSW.

4.4.1. Command transport

The host sends each CBW, which contains a command block, to the device through the Bulk-Out endpoint. The CBW starts on a packet boundary and ends as a short packet with exactly 31 (1FH) bytes transferred.

The device indicates successful transport of a CBW by accepting (ACKing) the CBW. If the host detects a STALL of the Bulk-Out endpoint during command transport, the host responds with a Reset Recovery. To report error before data transport completes and to maximize data integrity, the device may terminate the command by STALLing the endpoint in use (the Bulk-In endpoint during data IN and the Bulk-Out endpoint during data OUT).

4.4.2. Data transport

All data transports begin on a packet boundary.

The host attempts to transfer the exact number of bytes to or from the device as specified by the dCBWDataTransferLength field and the Direction bit. The device responds as specified in *Chapter 6: Host/Device Data Transfers* of *Universal Serial Bus Mass Storage Class Bulk-Only Transport Rev 1.0*.

4.4.3. Status transport

The device sends each CSW to the host by using the Bulk-In endpoint. The CSW starts on a packet boundary and ends as a short packet with exactly 13 (DH) bytes transferred. Figure 4-2 defines the algorithm the host uses for any CSW transfer. The CSW indicates to the host the status of the execution of the command block from the corresponding CBW. The dCSWDataResidue field indicates how much of the data transferred is to be considered processed or relevant. The host ignores any data received beyond that, which is relevant.

The host performs a Reset Recovery when Phase Error status is returned in the CSW.

4.4.4. Reset recovery

For Reset Recovery, the host issues in the following order:

1. A Bulk-Only Mass Storage Reset
2. A Clear Feature HALT to the Bulk-In endpoint
3. A Clear Feature HALT to the Bulk-Out endpoint

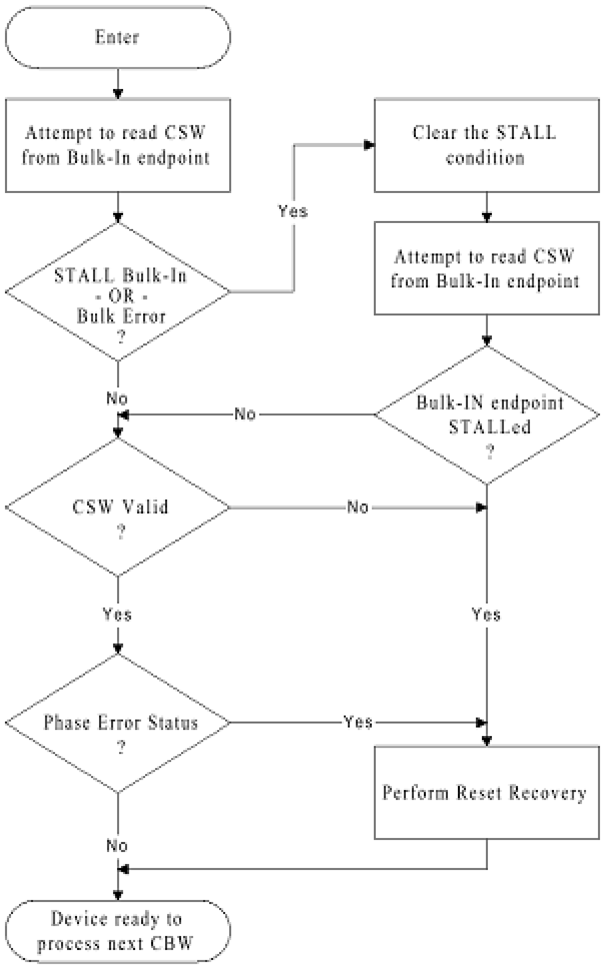

Enter
Attempt to read CSW from Bulk-In endpoint
STALL Bulk-In - OR - Bulk Error ?
Yes
No
Clear the STALL condition
Attempt to read CSW from Bulk-In endpoint
Bulk-IN endpoint STALLed ?
No
CSW Valid ?
No
Yes
Yes
Phase Error Status ?
Yes
No
Perform Reset Recovery
Device ready to process next CBW

Figure 4-2: Status transport flow

5. Error handling

The bridge solution implements the error-handling process according to the mass storage Bulk-Only transport specification. The error-handling process is classified under two categories: device and host.

5.1. Device error handling

The device may not be able to fully satisfy the host's request. When the device discovers that it cannot fully satisfy the request, there may be a Data-In or Data-Out transfer in progress on the bus, and the host may have other pending requests. The device may cause the host to terminate such transfers by STALLing the appropriate pipe.

The response of a device to a CBW that is not meaningful is not specified.

Note: Whether a STALL handshake actually appears on the bus depends on whether there is a transfer in progress when the device is ready to STALL the pipe.

5.2. Host error handling

If the host receives a CSW that is not valid, then the host performs a Reset Recovery. If the host receives a CSW that is not meaningful, then the host may perform a Reset Recovery.

5.3. Error classes

In every transaction between the host and the device, there are four possible classes of errors, which are not always independent of one another and may occur at any time during the transaction.

5.3.1. CBW Not Valid

If the CBW is not valid, the device STALLs the Bulk-In pipe. Also, the device either STALLs the Bulk-Out pipe, or the device accepts and discards any Bulk-Out data. The device maintains this state until a Reset Recovery.

5.3.2. Internal device error

The device may detect an internal error for which it has no reliable means of recovery other than a reset. The device responds to such conditions by:

either STALLing any data transfer in progress and returning a Phase Error status (bCSWStatus = 02H).

or STALLing all further requests on the Bulk-In and the Bulk-Out pipes until a Reset Recovery.

5.3.3. Host and device disagreements

After recognizing that a CBW is valid and meaningful, and in the absence of internal errors, the device may detect a condition in which it cannot meet the host's expectation for data transfer, as indicated by the Direction bit of the bmCBWFlags field and the dCBWDataTransferLength field of the CBW. In some of these cases, the device may require a reset to recover and so returns Phase Error status (bCSWStatus = 02H). Details on which cases result in Phase Error verses non-Phase Error status are given in the thirteen cases of the USB mass storage Bulk-Only transport specification.

5.3.4. Command failure

After recognizing that a CBW is valid and meaningful, the device may still fail in its attempt to satisfy the command. The device reports this condition by returning a Command Failed status (bCSWStatus = 01H).

6. Firmware flow

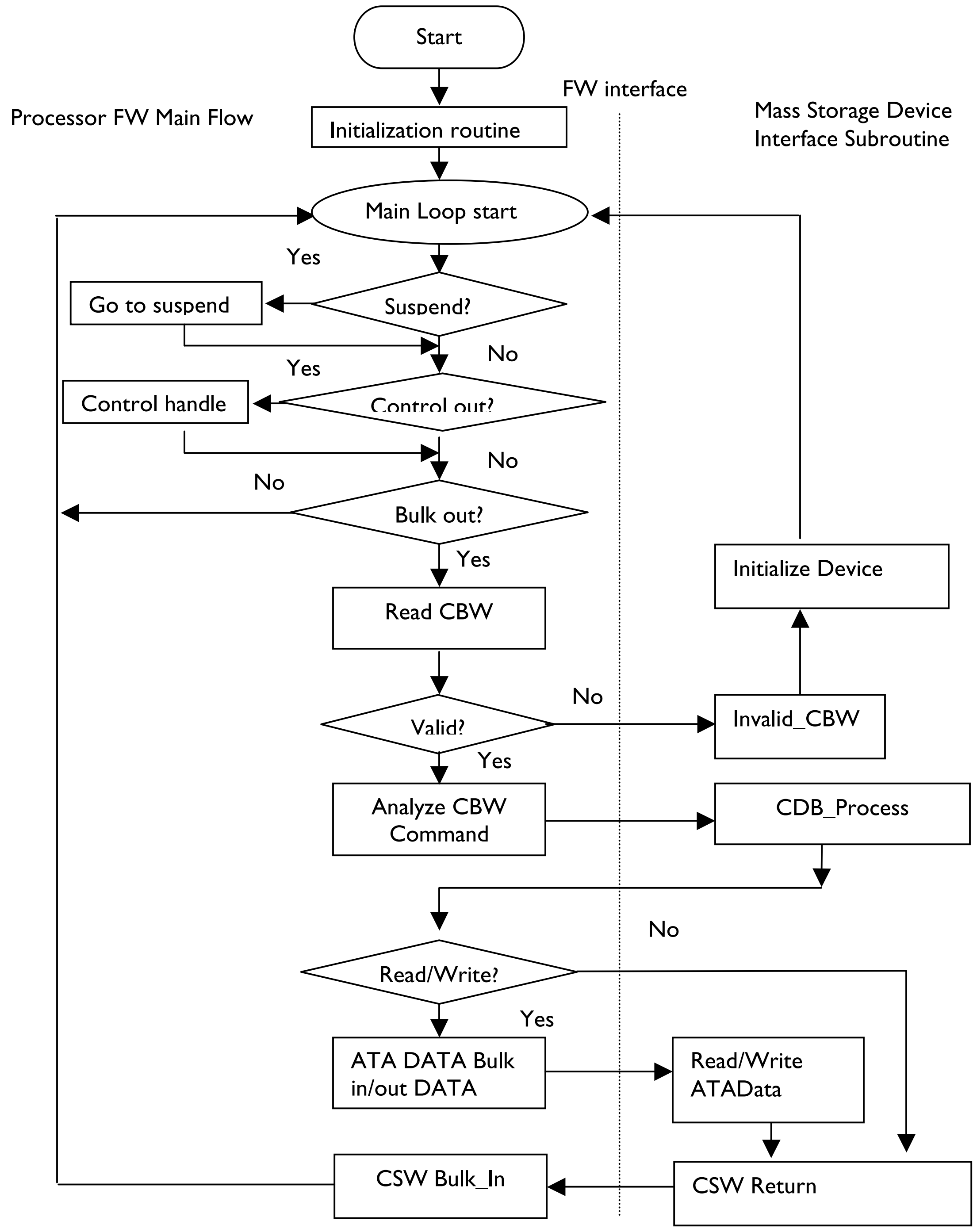


Figure 6-1: Processor FW Flow

The flowchart in Figure 6-1 illustrates how an ATA command sent by the host is interpreted and executed by the device.

For detailed explanation of the CDB process, refer to the ATA-5 specification or USB mass storage Class UFI Command specification.

7. Microsoft® Windows® 2000 and mass storage INF

The following is the usbstor.inf content, including the Vendor ID and the Product ID. You may want to include a string description to the INF. Once bus enumeration has passed, the USB mass storage driver boots by sending the CBW to the device attached. The device manager can be evoked to check whether the device has been correctly recognized.

```
[Version]
Signature="$CHICAGO$"
Class=USB
ClassGUID={36FC9E60-C465-11CF-8056-444553540000}
provider=%MSFT%
LayoutFile=LAYOUT.INF
DriverVer=11/14/1999,5.00.2183.1

[ControlFlags]
ExcludeFromSelect = *

[Manufacturer]
%MfgName%=Microsoft

[Microsoft]
%USB\VID_04cc&PID_1b53.DeviceDesc%=USBSTOR_BULK, USB\VID_04cc&PID_1b53

%GenericBulkOnly.DeviceDesc%=USBSTOR_BULK, USB\Class_08&SubClass_02&Prot_50
%GenericBulkOnly.DeviceDesc%=USBSTOR_BULK, USB\Class_08&SubClass_05&Prot_50
%GenericBulkOnly.DeviceDesc%=USBSTOR_BULK, USB\Class_08&SubClass_06&Prot_50

[PreCopySection]
HKR,,NoSetupUI,,1

[DestinationDirs]
USBSTOR.CopyList = 10, system32\drivers
NTMAP.CopyList = 10, system32\drivers

; Windows 98+ Install Sections
;
[USBSTOR_CBI]
CopyFiles=USBSTOR.CopyList, NTMAP.CopyList
AddReg=USBSTOR.AddReg, USBSTOR_CBI.AddReg

[USBSTOR_CB]
CopyFiles=USBSTOR.CopyList, NTMAP.CopyList
AddReg=USBSTOR.AddReg, USBSTOR_CB.AddReg

[USBSTOR_BULK]
CopyFiles=USBSTOR.CopyList, NTMAP.CopyList

AddReg=USBSTOR.AddReg, USBSTOR_BULK.AddReg
```

```
; Windows 2000 Install Sections
;
[USBSTOR_CBI.NT]
CopyFiles=USBSTOR.CopyList
AddReg=USBSTOR_CBI.AddReg

[USBSTOR_CB.NT]
CopyFiles=USBSTOR.CopyList
AddReg=USBSTOR_CB.AddReg

[USBSTOR_BULK.NT]
CopyFiles=USBSTOR.CopyList
AddReg=USBSTOR_BULK.AddReg

; Windows 98+ Registry Section
;
[USBSTOR.AddReg]
HKR,,DevLoader,,*NTKERN
HKR,,NTMPDriver,,USBSTOR.SYS

; Windows 98+ & Windows 2000 Registry Sections
;
[USBSTOR_CBI.AddReg]
HKR,,DriverFlags,0x00010001,0x00000002

[USBSTOR_CB.AddReg]
HKR,,DriverFlags,0x00010001,0x00000003

[USBSTOR_BULK.AddReg]
HKR,,DriverFlags,0x00010001,0x00000001

; Windows 98+ HW Install Sections
;
[USBSTOR_CBI.HW]
AddReg=USBSTOR.HW.AddReg

[USBSTOR_CB.HW]
AddReg=USBSTOR.HW.AddReg

[USBSTOR_BULK.HW]
AddReg=USBSTOR.HW.AddReg

[USBSTOR.HW.AddReg]
HKR,,upperfilters,0,"NTMAP.SYS"

; Windows 2000 Service Install Sections
;
[USBSTOR_CBI.NT.Services]
Addservice = USBSTOR, 0x00000002, USBSTOR.AddService

[USBSTOR_CB.NT.Services]
Addservice = USBSTOR, 0x00000002, USBSTOR.AddService
```

```
[USBSTOR_BULK.NT.Services]
Addservice = USBSTOR, 0x00000002, USBSTOR.AddService

[USBSTOR.AddService]
DisplayName    = %USBSTOR.SvcDesc%
ServiceType    = 1
StartType      = 3
ErrorControl   = 1
ServiceBinary  = %12%\USBSTOR.SYS

[USBSTOR.CopyList]
USBSTOR.SYS

[NTMAP.CopyList]
NTMAP.SYS

[Strings]
MSFT="Microsoft"
MfgName="Microsoft"

USB\VID_04cc&PID_1b53.DeviceDesc = "Philips ISP1582 Mass Storage Demo"

GenericBulkOnly.DeviceDesc = "USB Mass Storage Device"

USBSTOR.SvcDesc = "USB Mass Storage Driver"
```

8. References

- *Universal Serial Bus Specification Rev. 2.0*
- *Universal Serial Bus Specification Rev. 1.1*
- *ISP1582 Hi-Speed Universal Serial Bus interface device* data sheet
- *Universal Serial Bus Mass Storage Class Bulk-Only Transport Rev 1.0.*
- *USB Mass Storage Class UFI Command Specification*

Philips Semiconductors

www.semiconductors.philips.com

PHILIPS