Semiconductors

UM10043_1

ISP1183 Firmware Programming Guide

October 2003

User's Guide

Rev. 1.0

Revision History:

Version	Date	Description	Author
1.0	October 2003	First release.	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

This is a legal agreement between you (either an individual or an entity) and Philips Semiconductors. By accepting this product, you indicate your agreement to the disclaimer specified as follows:

DISCLAIMER

PRODUCT IS DEEMED ACCEPTED BY RECIPIENT. THE PRODUCT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, PHILIPS SEMICONDUCTORS FURTHER DISCLAIMS ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANT ABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE PRODUCT AND DOCUMENTATION REMAINS WITH THE RECIPIENT. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL PHILIPS SEMICONDUCTORS OR ITS SUPPLIERS BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, DIRECT, INDIRECT, SPECIAL, PUNITIVE, OR OTHER DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THIS AGREEMENT OR THE USE OF OR INABILITY TO USE THE PRODUCT, EVEN IF PHILIPS SEMICONDUCTORS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

CONTENTS

1. INTRODUCTION 5

2. ARCHITECTURE 6

2.1. FIRMWARE STRUCTURE 6
2.1.1. Hardware Abstraction Layer—HAL4SYS.C 6
2.1.2. Hardware Abstraction Layer—HAL4D13.C 6
2.1.3. Interrupt Service Routine—ISR.C 6
2.1.4. Protocol Layer—CHAP_9.C 7
2.1.5. Protocol Layer—D13bus.C 7
2.1.6. Main Loop—MAINLOOP.C 7
2.2. PORTING THE FIRMWARE TO OTHER CPU PLATFORM 7
2.3. USING THE FIRMWARE IN THE POLLING MODE 7

3. HARDWARE ABSTRACTION LAYER FOR A SYSTEM 7

4. HARDWARE ABSTRACTION LAYER FOR THE ISP1183 8

5. INTERRUPT SERVICE ROUTINE 10

5.1. BUS RESET 11
5.2. SUSPEND CHANGE 11
5.3. EOT HANDLER 12
5.4. CONTROL ENDPOINT HANDLER 12
5.5. CONTROL OUT HANDLER 12
5.6. CONTROL IN HANDLER 14
5.7. BULK ENDPOINT HANDLER 15
5.8. ISO ENDPOINT HANDLER 16

6. MAIN LOOP 18

7. STANDARD DEVICE REQUEST 19

7.1. CLEAR FEATURE REQUEST 19
7.2. GET STATUS REQUEST 20
7.3. SET ADDRESS REQUEST 20
7.4. GET CONFIG REQUEST 21
7.5. GET DESCRIPTOR REQUEST 21
7.6. SET CONFIG REQUEST 21
7.7. GET OR SET INTERFACE REQUEST 22
7.8. SET FEATURE REQUEST 22

8. VENDOR REQUEST 23

8.1. VENDOR REQUEST FOR BULK TRANSFER 23
8.2. HOST SIDE PROGRAMMING CONSIDERATIONS 24

9. REFERENCES 25

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

FIGURES

Figure 2-1: Firmware Structure of the ISP1183 6
Figure 5-1: Flowchart of the ISR 10
Figure 5-2: State Machine of a Control Transfer 12
Figure 5-3: Flowchart of a control OUT handler 13
Figure 5-4: Flowchart of a control IN handler 14
Figure 5-5: Flowchart of a bulk OUT handler 15
Figure 5-6: Flowchart of a bulk IN handler 16
Figure 5-7: Flowchart of an ISO OUT handler 17
Figure 5-8: Flowchart of an ISO IN handler 17
Figure 6-1: Flowchart of Main Loop 18
Figure 7-1: Flowchart of Clear Feature 19
Figure 7-2: Flowchart of Get Status 20
Figure 7-3: Flowchart of Set Address 20
Figure 7-4: Flowchart of Get Config 21
Figure 7-5: Flowchart of Set Config 22
Figure 7-6: Flowchart of Set Feature 23

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corp. in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners. All other names, products, and trademarks are the property of their respective owners.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

1. Introduction

The ISP1183 is a USB interface device, supporting data transfer at full-speed (12 Mbit/s). It has up to 14 configurable endpoints, and has a fast general-purpose parallel interface.

To a microcontroller or a microprocessor, the ISP1183 appears as a memory device with an 8-bit data bus and a 1-bit address bus. It has 2462 bytes of internal FIFO memory. The type and FIFO size of each endpoint can be individually configured, depending on the required packet size. For increased data throughput, isochronous and bulk endpoints are double-buffered.

The objective of the firmware is to achieve the fastest transfer rate over USB for the ISP1183. Peripherals such as mobile phones, printers, scanners, external mass storage (Zip® drive) devices, and digital still cameras can use the ISP1183 to transfer data over the USB. The CPUs in these devices are busy handling many tasks such as device control, data and image processing. The ISP1183 firmware is designed to be fully interrupt-driven. While the CPU is performing a foreground task, the USB transfer is being handled in the background. This assures the best transfer rate, better software structure, and also simplifies programming and debugging.

This firmware-programming guide aims to help you utilize the ISP1183 more efficiently.

The organization of this document is as follows:

- Section 2 describes architecture of the firmware. It also introduces how to port the firmware to other CPU platforms.
- Section 3 describes the hardware abstraction layer.
- Section 4 covers the ISP1183 command interface and implementations of the ISP1183 command sets.
- Section 5 focuses on Interrupt Service Routine (ISR). It gives details on individual endpoint handlers, including control endpoint, bulk endpoint and isochronous endpoint handlers.
- Section 6 describes the Main Loop that executes in the foreground.
- Section 7 describes the firmware implementation of USB device request based on chapter 9 of the *Universal Serial Bus Specification Revision 2.0*.
- Section 8 introduces the firmware implementation of vendor request.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

2. Architecture

2.1. Firmware Structure

The firmware for the evaluation (eval) board consists of six building blocks, as shown in Figure 2-1.

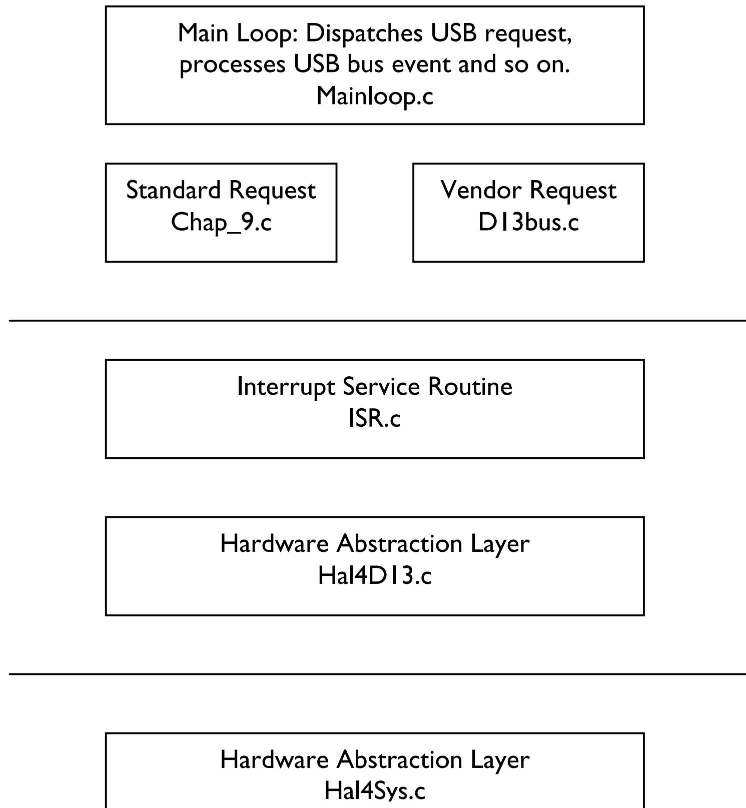


Figure 2-1: Firmware Structure of the ISP1183

2.1.1. Hardware Abstraction Layer—HAL4SYS.C

This is the lowest layer code in the firmware, which performs hardware-dependent I/O access to the ISP1183, as well as the eval board hardware. When porting the firmware to other CPU platforms, this part of the code always requires modifications or additions.

2.1.2. Hardware Abstraction Layer—HAL4D13.C

To simplify programming with the ISP1183, the firmware defines a set of command interfaces, which encapsulate all the functions used to access the ISP1183.

2.1.3. Interrupt Service Routine—ISR.C

This part of the code handles interrupt generated by the ISP1183. It retrieves data from ISP1183's internal FIFO to the CPU memory, and sets up proper event flags to inform the Main Loop program for processing.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

2.1.4. Protocol Layer—CHAP_9.C

This Protocol layer handles standard USB device request, which is defined in chapter 9 of the *Universal Serial Bus Specification Rev. 2.0*. The firmware implementation of the USB device request is described in detail in Section 7.

2.1.5. Protocol Layer—D13bus.C

This Protocol layer handles specific vendor requests. For example, bulk transfer and isochronous transfer.

2.1.6. Main Loop—MAINLOOP.C

The Main Loop checks the event flags and passes to the appropriate subroutine for further processing. It also contains the code for human interface, such as V_{BUS} detect and key scan.

2.2. Porting the Firmware to Other CPU Platform

The following table shows the modifications that are required on building blocks. There are two levels of porting. The first level is Standard Device Request, that is, USB Chapter 9 only, which is to enable the firmware to pass enumeration by supporting standard USB requests. The second level is full product development; this involves product-specific firmware code, that is, Vendor Request.

File Name	Chapter 9 Only	Product Level
HAL4SYS.C	Port to hardware specific.	Port to hardware specific.
HAL4D13.C	No change.	No change.
ISR.C	No change.	Add product-specific processing on Generic and Main endpoints.
CHAP_9.C	No change.	Product-specific USB descriptors.
D13BUS.C	No change.	Add vendor request supports, if necessary.
MAINLOOP.C	Depends on the CPU and system. Ports, timer and interrupt initialization need to be rewritten.	Add product-specific Main Loop processing.

2.3. Using the Firmware in the Polling Mode

It is very easy to use the firmware in the polling mode. Inside the Main Loop, add the following code:

```
if(interrupt_pin_low)
        fn_usb_isr();
```

Typically, the ISR is initiated by hardware. In the polling mode, the Main Loop detects the status of the interrupt pin, and invokes ISR, if necessary.

3. Hardware Abstraction Layer for a System

This layer contains the lowest-layer functions that need to be changed on different CPU platforms.

```
Hal4Sys_AcquireTimer0(void);
Hal4Sys_ReleaseTimer0(void);
interrupt Hal4Sys_Isr4Timer(void);
```


© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

```
void Hal4Sys_WaitinUS(IN OUT ULONG time);
void Hal4Sys_WaitinMS( IN OUT ULONG time);


void Hal4Sys_ControlD13Interrupt( BOOLEAN InterruptEN);
```

For example, the subroutine to acquire a system timer is as follows:

```
void Hal4Sys_AcquireTimer0(void)
{
        if(bD13flags.bits.verbose)
                printf("enter Hal4Sys_AcquireTimer0\n");

        Hal4Sys_OldIsr4Timer = getvect(0x8);
        setvect(0x8, Hal4Sys_Isr4Timer);

        if(bD13flags.bits.verbose)
                printf("exit Hal4Sys_AcquireTimer0\n");
}
```

4. Hardware Abstraction Layer for the ISP1183

The following functions are defined as the ISP1183 command interface to simplify device programming. They are the implementations of the ISP1183 command set, which is defined in the data sheet.

```
Hal4D13_SetEndpointConfig(UCHAR bEPConfig, UCHAR bEPIndex);
Hal4D13_GetEndpointConfig(UCHAR bEPIndex);

Hal4D13_SetAddressEnable(UCHAR bAddress, UCHAR bEnable);
Hal4D13_GetAddress(void);

Hal4D13_SetMode(UCHAR bMode);
Hal4D13_GetMode(void);

Hal4D13_SetDevConfig(USHORT wDevCnfg);
Hal4D13_GetDevConfig(void);

Hal4D13_SetIntEnable(ULONG dIntEn);
Hal4D13_GetIntEnable(void);

Hal4D13_SetDMAConfig(USHORT wDMAConfig);
Hal4D13_GetDMAConfig(void);
Hal4D13_SetDMACounter(USHORT wDMACounter);
Hal4D13_GetDMACounter(void);

Hal4D13_ResetDevice(void);

Hal4D13_WriteEndpoint(UCHAR bEPIndex, UCHAR * buf, USHORT len);
Hal4D13_ReadEndpoint(UCHAR bEPIndex, UCHAR * buf, USHORT len);

Hal4D13_Writedma(UCHAR  far * buf, USHORT len);
Hal4D13_Readdma(UCHAR  far * buf, USHORT len);

Hal4D13_SetEndpointStatus(UCHAR bEPIndex, UCHAR bStalled);
```


© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

```
Hal4D13_GetEndpointStatusWInteruptClear(UCHAR bEPIndex);


Hal4D13_ValidBuffer(UCHAR bEPIndex);
Hal4D13_ClearBuffer(UCHAR bEPIndex);

Hal4D13_AcknowledgeSETUP(void );

Hal4D13_GetErrorCode(UCHAR bEPIndex);
Hal4D13_LockDevice(UCHAR bTrue);

Hal4D13_ReadChipID(void);
Hal4D13_ReadCurrentFrameNumber(void);

Hal4D13_ReadInterruptRegister(void);
```

For example, the implementation of the Set Mode command is as follows:

```
void Hal4D13_SetMode(UCHAR bMode)
{
        outport(D13_COMMAND_PORT, D13CMD_DEV_WR_MODE);
        outport(D13_DATA_PORT, bMode);

}
```


© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

5. Interrupt Service Routine

The ISP1183 firmware is fully interrupt-driven. The flow of the ISR is shown in Figure 5-1.

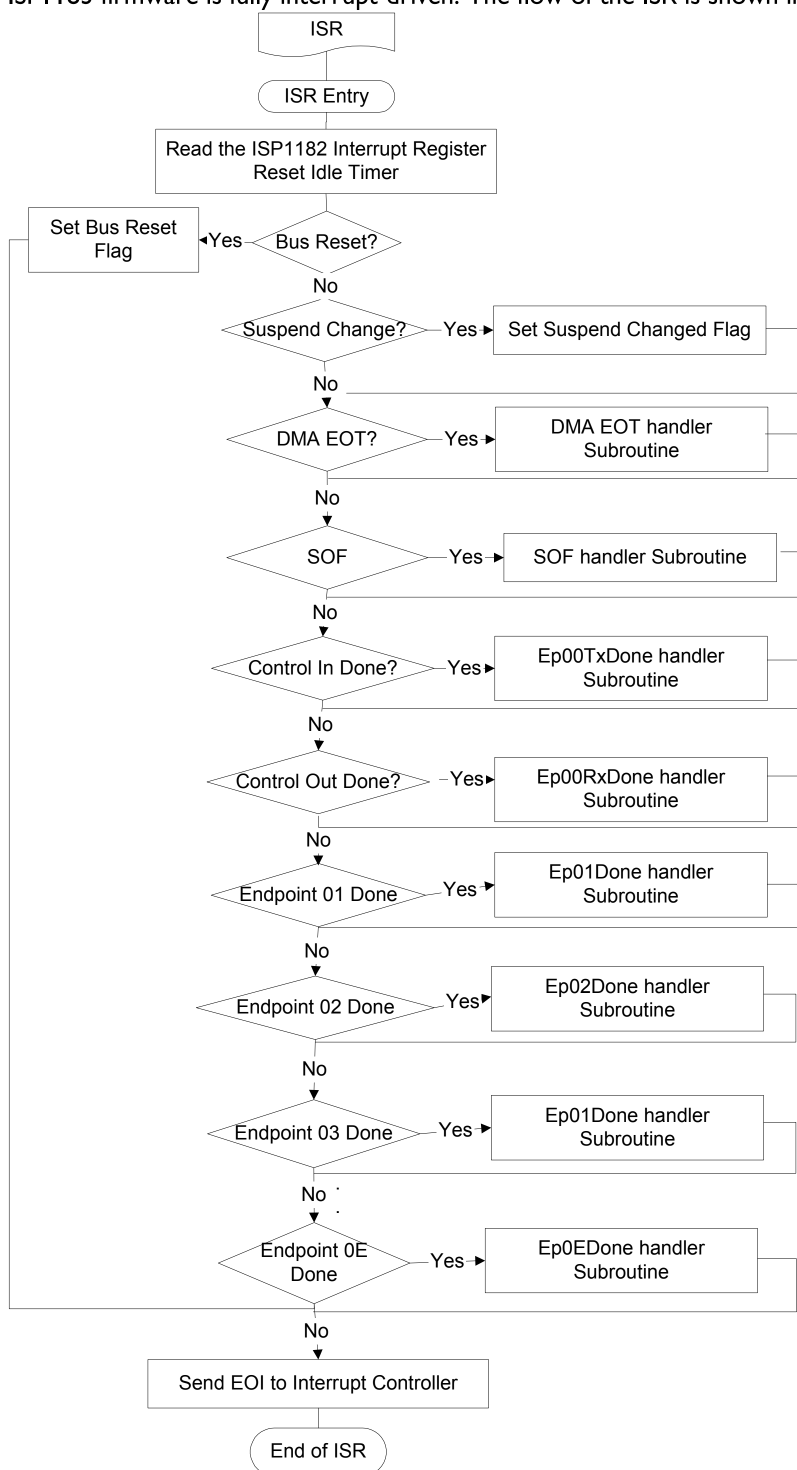


Figure 5-1: Flowchart of the ISR

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

At the entrance of the ISR, the firmware uses the Read Interrupt register to decide the source of an interrupt and then dispatches the interrupt to the appropriate subroutines for processing.

The ISR communicates with the foreground Main Loop through event flags 'D13FLAGS' and data buffers 'CONTROL_XFER'.

```
typedef union _D13FLAGS
{
        struct _D13FSM_FLAGS
        {

                IRQL_1 UCHAR        bus_reset       : 1;
                IRQL_1 UCHAR        suspend         : 1;
                IRQL_1 UCHAR        DCP_state       : 4;
                IRQL_1 UCHAR        setup_dma       : 1;
                IRQL_1 UCHAR        timer           : 1;
        } bits;
        ULONG value;
} D13FLAGS;

typedef struct _CONTROL_XFER
{
IRQL_1 DEVICE_REQUEST   DeviceRequest;
                IRQL_1 USHORT       wLength;
                IRQL_1 USHORT        wCount;
                IRQL_1 ADDRESS      Addr;
                IRQL_1 UCHAR        dataBuffer[MAX_CONTROLDATA_SIZE];

} CONTROL_XFER, * PCONTROL_XFER;

Where,

typedef struct _device_request
{
        UCHAR bmRequestType;
        UCHAR bRequest;
        USHORT wValue;
        USHORT wIndex;
        USHORT wLength;
} DEVICE_REQUEST;
```

Tasks are split between Main Loop and ISR as follows. The ISR collects data from the internal buffer of the ISP1183 and moves the data packet to a data buffer. When it has collected enough data, the ISR informs the Main Loop that data is ready for processing. The Main Loop then processes the data in the data buffer.

5.1. Bus Reset

Bus reset does not require special processing within the ISR. The ISR sets the 'bus_reset' flag in D13FLAGS and then exits.

5.2. Suspend Change

Suspend does not require special processing within the ISR. The ISR sets the suspend flag in D13FLAGS and then exits.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

5.3. EOT Handler

EOT does not require special processing. It indicates the last DMA transfer.

5.4. Control Endpoint Handler

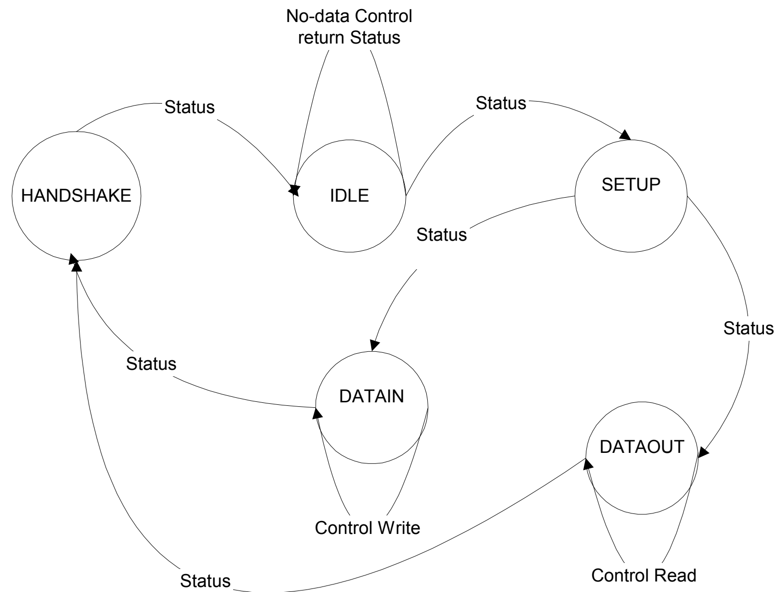


Figure 5-2: State Machine of a Control Transfer

The state machine of a control transfer is shown in Figure 5-2. A control transfer always begins with the setup stage, followed by an optional data stage. The data stage consists of one or more IN or OUT transactions. It ends with the status stage, that is, HANDSHAKE. The preceding diagram shows the various states of transitions on the control endpoints. The firmware uses these five states to correctly handle the control transfer.

5.5. Control OUT Handler

Figure 5-3 shows the flowchart of the control OUT handler. The microcontroller has to clear the control OUT interrupt bit of the ISP1183 and verify whether the endpoint is full. Then, the microcontroller extracts the content of the data OUT packet buffer by reading the control endpoint. Finally, the handler sets the ISP1183 to the proper status.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

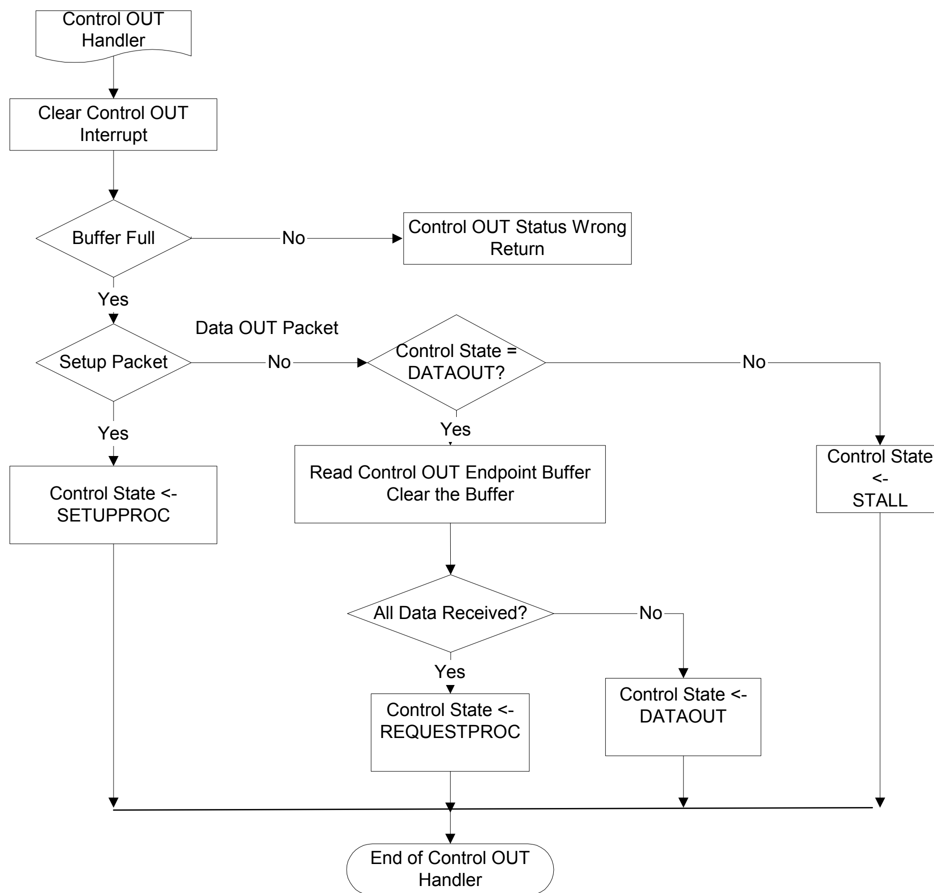


Figure 5-3: Flowchart of a control OUT handler

For example, if the setup packet is the Set_Descriptor request, then the control transfer in the setup packet indicates that it is a control OUT type. After executing the procedure for Set_Descriptor, the microcontroller waits for the data phase. The host sends an OUT token through the control OUT endpoint. Then, the microcontroller must extract data from the ISP1183 buffer.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

5.6. Control IN Handler

After the setup stage is complete, the host executes the data phase. If the ISP1183 receives a control IN packet, it goes to 'Control_ IN Handler'. Again, the microcontroller first needs to clear the control IN interrupt bit of the ISP1183 by reading its Read Endpoint Status. Then, the microcontroller proceeds to send the data packet after verifying that the ISP1183 is in the appropriate mode.

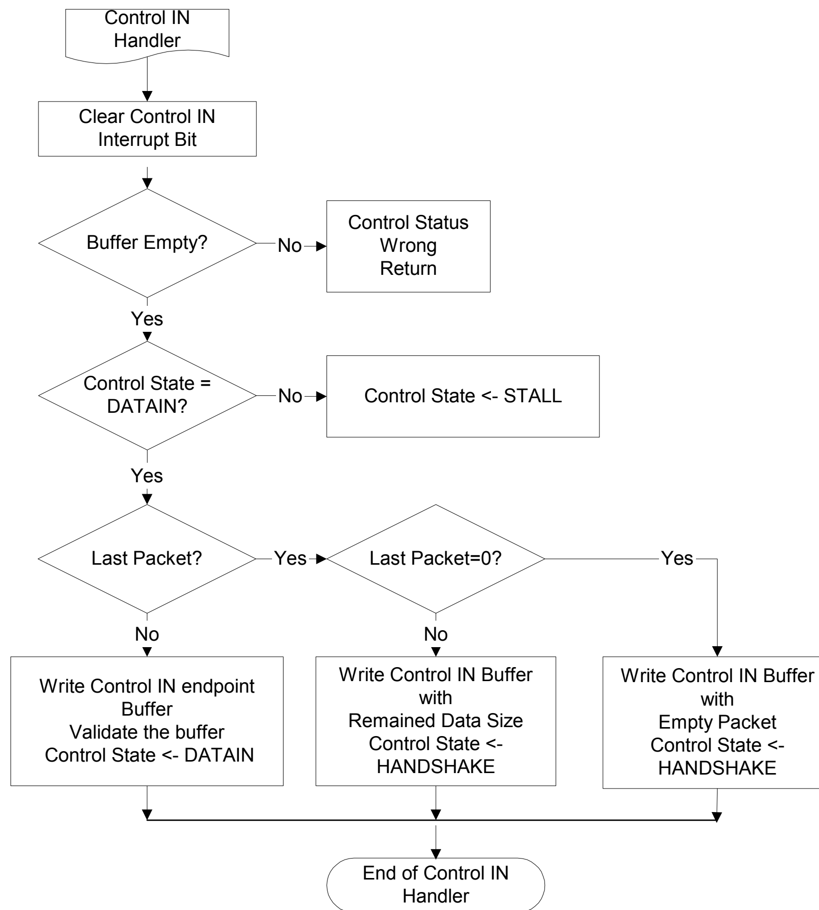


Figure 5-4: Flowchart of a control IN handler

Figure 5-4 shows the flowchart of a control IN handler. As an ISP1183 control endpoint has only 64 bytes FIFO, the microcontroller has to control the amount of data during the transmitting phase if the requested length is more than 64 bytes. As indicated in the flowchart, the microcontroller must check the current and remaining size of the data to be sent to the host. If the remaining bytes are greater than 64 bytes, the microcontroller sends the first 64 bytes and then subtract the reference length (requested length) by 64.

When the next control IN token comes, the microcontroller determines whether the remaining bytes are zero. If there is no more data to be sent, the microcontroller needs to send an empty packet to inform the host that there is no more data to be sent.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

5.7. Bulk Endpoint Handler

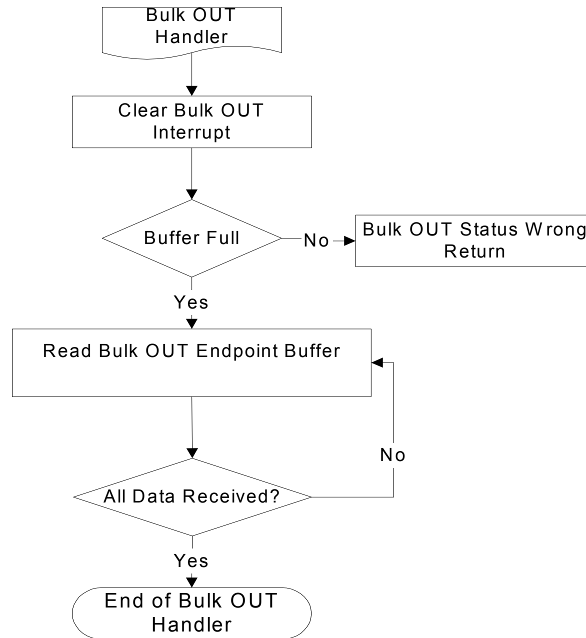
The ISP1183 has 16 endpoints: control IN and OUT plus 14 configurable endpoints. The 14 endpoints can be individually defined as interrupt, bulk or isochronous—IN or OUT. The FIFO size determines the maximum packet size that the hardware can support for a given endpoint.

The following table is the recommended register programming in the Endpoint Configuration register for a bulk endpoint.

Bit	Bit setting	Description
7	1	Endpoint enable bit
6	0 for OUT 1 for IN	Endpoint direction
5	1	Enable double buffering
4	0	Bulk endpoint
3 to 0	0011	Size bits of an enabled endpoint: 64 bytes

Figure 5-5: Flowchart of a bulk OUT handler

When the host is ready to transmit bulk data, it issues an OUT token packet followed by a data packet. The



ISP1183 generates an interrupt to inform the microcontroller, which needs to clear the ISP1183 interrupt bit and verify the data length. The flowchart of a bulk OUT handler is shown in Figure 5-5.

When the host is ready to receive bulk data, it issues an IN token. The ISP1183 generates an interrupt to inform the microcontroller, which must clear the ISP1183 interrupt bit and return the data packet to be sent. The flowchart of a bulk IN handler is shown in Figure 5-6.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

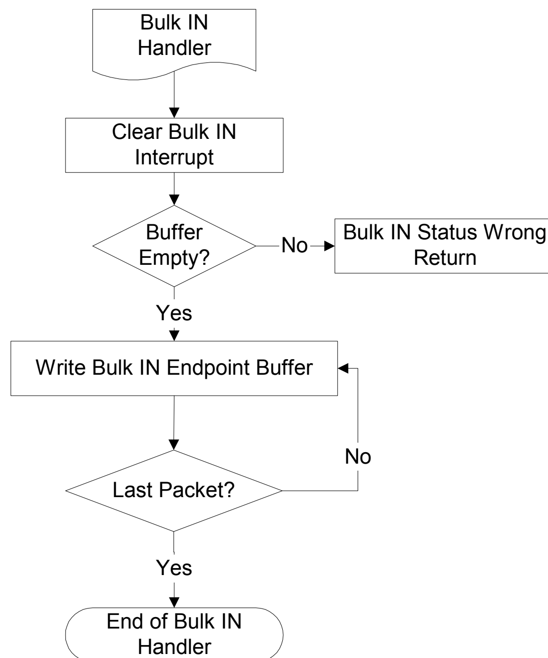


Figure 5-6: Flowchart of a bulk IN handler

5.8. ISO Endpoint Handler

The following table is the recommended register programming in the Endpoint Configuration register for an ISO endpoint.

The flowcharts of the ISO OUT handler and the ISO IN handler are shown in Figure 5-7 and Figure 5-8, respectively.

Bit	Bit setting	Description
7	1	Endpoint enable bit
6	0 for OUT 1 for IN	Endpoint direction
5	1	Enable double buffering
4	1	ISO endpoint
3 to 0	0011	Size bits of an enabled endpoint: 64 bytes

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

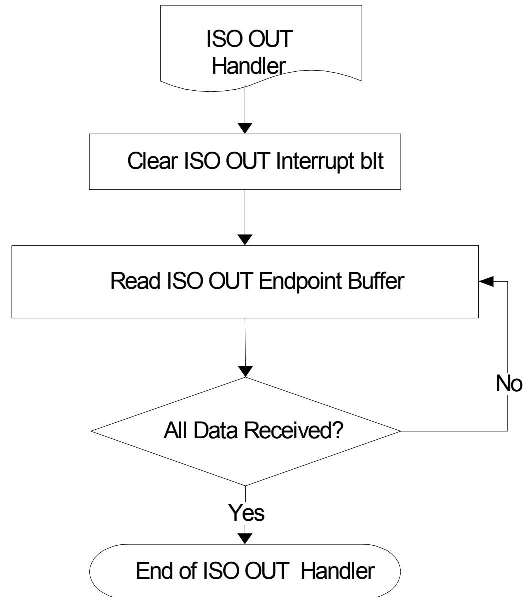


Figure 5-7: Flowchart of an ISO OUT handler

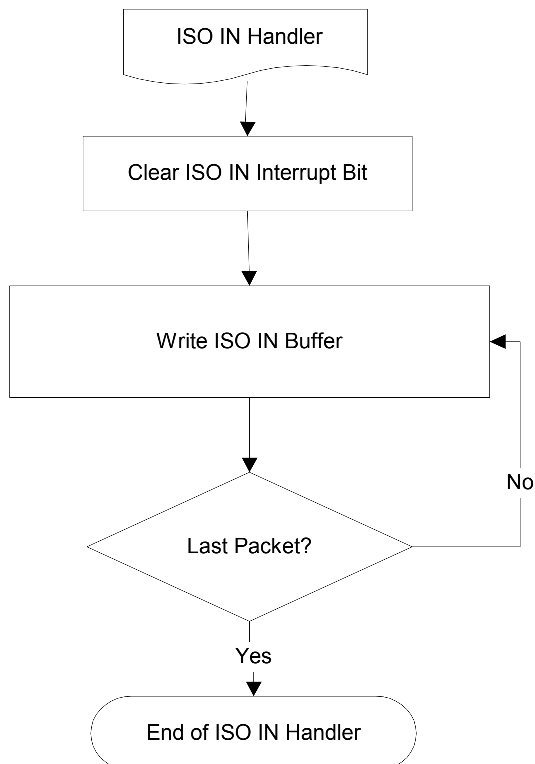


Figure 5-8: Flowchart of an ISO IN handler

Time is a key part of an isochronous transfer. A typical example of the isochronous data is voice. All isochronous pipes move exactly one data packet per frame, that is, every 1millisecond.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

6. Main Loop

Once powered, the microcontroller has to initialize its ports, memory, timer, and ISR routine handler. The microcontroller then reconnects USB, which involves setting bit SOFTCT in the Mode register. This procedure is important because it ensures that the ISP1183 will not operate before the microcontroller is ready to serve the ISP1183.

The flowchart of the Main Loop is shown in Figure 6-1. In the Main Loop routine, the microcontroller polls for any activity on the keyboard. If a key is pressed on the keyboard, the handle key commands execute the routine and then return to the Main Loop. This routine is added for debugging purposes only. A 1-ms timer is programmed to activate the routine to check for any keypress on the eval board.

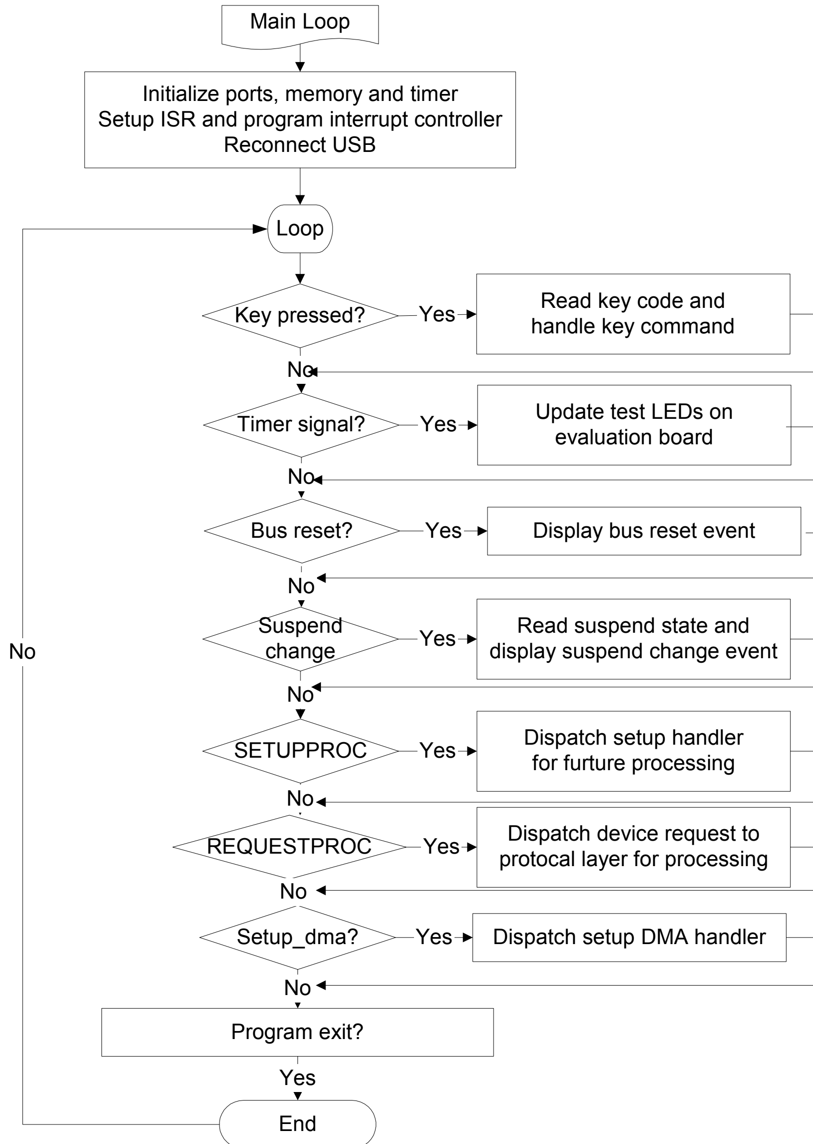


Figure 6-1: Flowchart of Main Loop

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

When polling reaches the check setup packet, the microcontroller verifies whether the current status is SETUPPROC. Then, it dispatches it to the setup handler subroutines for processing. When reaching REQUESTPROC, it dispatches device request to the protocol layer for processing.

7. Standard Device Request

All USB devices must respond to a variety of requests called 'standard' requests. These requests are used for configuring a device, controlling the state of its interface, along with other miscellaneous features. The host issues device requests by using the control transfer mechanism. For detailed information, refer to Chapter 9 of the *Universal Serial Bus Specification Rev. 2.0*.

7.1. Clear Feature Request

In the Clear feature request, the microcontroller has to clear or disable a specific feature of the device. The flowchart of Clear Feature is shown in Figure 7-1. The microcontroller determines whether the request is meant for the device, the interface, or endpoints. Note: If the recipient is an interface, there is no support. Feature selectors are used when enabling or setting features specific to the device or endpoint, such as remote wake-up. If the recipient is a device, the microcontroller has to disable the remote wake-up function, if this function is enabled. If the recipient is the endpoint, the microcontroller must unstall the specific endpoint through the Write Endpoint Status command.

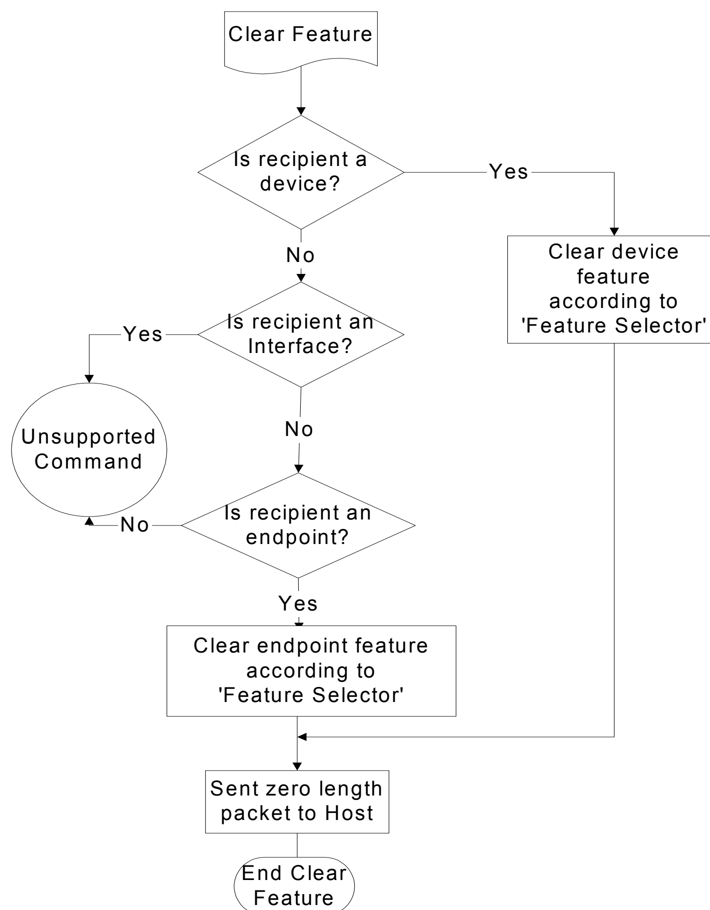


Figure 7-1: Flowchart of Clear Feature

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

7.2. Get Status Request

In the Get Status request (see Figure 7-2), the microcontroller must return the status for the specific recipient. The microcontroller has to determine again the recipient of the request. If the request is to the device, the microcontroller must return the status of the device to the host. For systems having remote wake-up and self-power capabilities, the returning data is 0x0003. If the recipient is an interface, the microcontroller should return 0x0000 to the host.

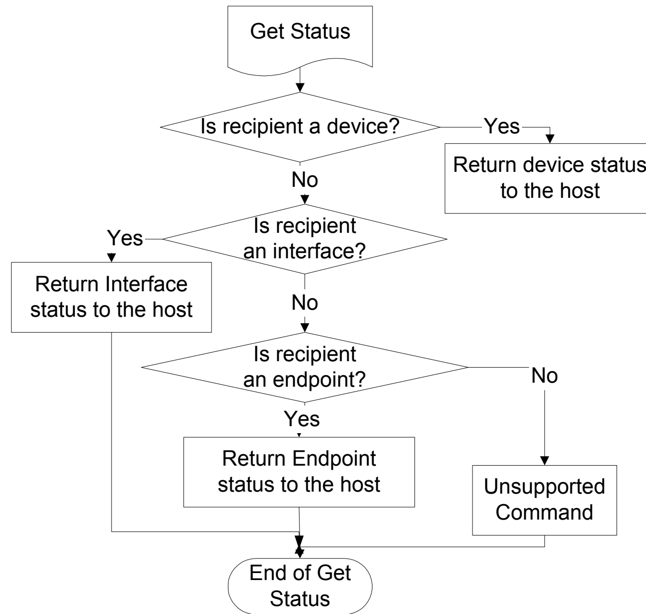


Figure 7-2: Flowchart of Get Status

7.3. Set Address Request

In the Set Address request, the device gets the new address from the content of the setup packet. The flowchart of Set Address is given in Figure 7-3.

Note: The Set Address request does not have a data phase. Therefore, the microcontroller has to write a zero-length data packet to the host at the acknowledgment phase.

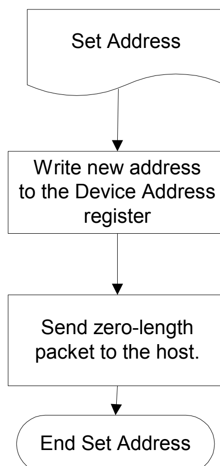


Figure 7-3: Flowchart of Set Address

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

7.4. Get Config Request

In the Get Config request, the microcontroller must return the current configuration value. The microcontroller determines whether the device is configured. If the device is not configured, it returns a zero to the host; otherwise, it returns a one. See Figure 7-4.

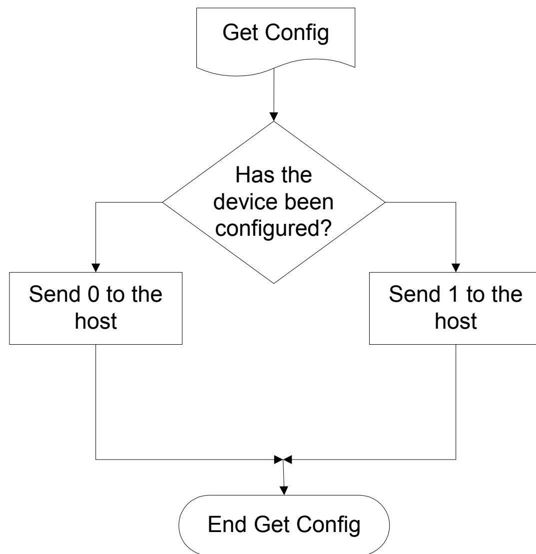


Figure 7-4: Flowchart of Get Config

7.5. Get Descriptor Request

For the Get Descriptor request, the microcontroller must return the specific descriptor, if the descriptor exists. First, the microcontroller determines whether the descriptor type request is for device or configuration. It then sends the first 64 bytes of device descriptor, if the descriptor type is for device. The size of the returning bytes must be controlled because the control buffer has only 64 bytes of memory. The microcontroller must set a register to indicate the location of the transmitted size.

7.6. Set Config Request

For the Set Config request, the microcontroller determines the configuration value from the setup packet. If the value is zero, then the microcontroller has to clear the configuration flag in its memory and disable the endpoint. If the value is one, then the microcontroller must set the configuration flag. Once the flag is set, the microcontroller also needs to send the zero data packet to the host at the acknowledgment phase.

The flowchart of Set Config is given in Figure 7-5.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

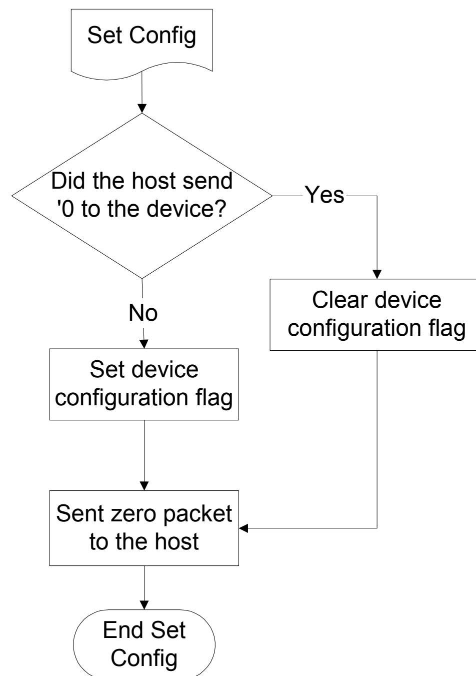


Figure 7-5: Flowchart of Set Config

7.7. Get or Set Interface Request

For the Get or Set Interface request, the microcontroller has to send a zero data packet to the host because the ISP1183 eval board supports only one type of interface. For the Set Interface request on the ISP1183 eval board, the microcontroller does not do anything except sending a zero data packet to the host as the acknowledgment phase.

7.8. Set Feature Request

The Set Feature request is the opposite of the Clear Feature request. If the recipient is a device, the microcontroller has to set the feature of the device according to the feature selector on the setup packet. There is no support for the Interface recipient. For example, if the feature selector is 0 (which means enabling endpoint), the ISP1183 specific endpoint must be stalled through the Write Endpoint status command. See Figure 7-6.

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

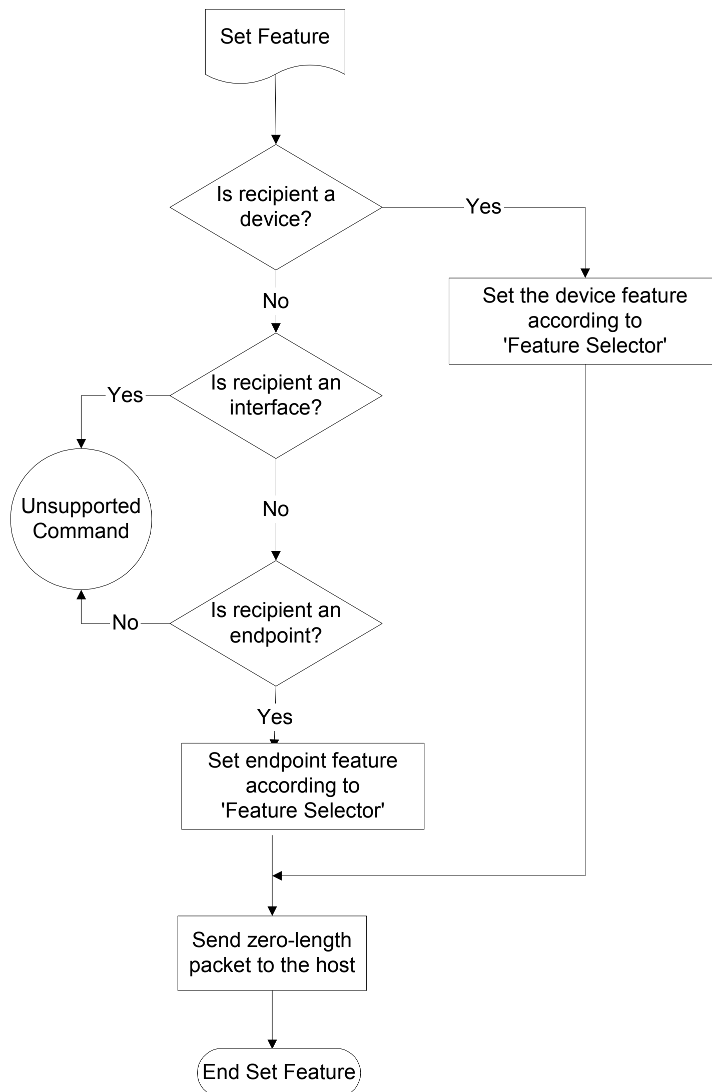


Figure 7-6: Flowchart of Set Feature

8. Vendor Request

In the ISP1183 sample firmware and applet, the bulk transfer or the isochronous transfer is set up by vendor request. This request is sent through control pipe (which is done using IOCTL_WRITE_REGISTER). IOCTL_WRITE_REGISTER is defined by Microsoft® Still Image USB Interface in Microsoft Windows 98 DDK. A device vendor may also define requests supported by the device.

8.1. Vendor Request for bulk transfer

The device request is defined as follows:

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

Offset	Field	Size	Value	Comments
0	BmRequestType	1	0x40	Vendor request, device to host
1	Brequest	1	0x0C	Fixed value for IOCTL_WRITE_REGISTER
2	Wvalue	2	0	Offset, set to zero
4	Windex	2	0x0471	Fixed value of the setup bulk transfer
6	Wlength	2	6	Data length of the setup bulk transfer

The details requested by the bulk transfer operation are sent in the data phase after the setup token phase of the device request. The sample firmware and applet use a proprietary definition as shown in the following table:

Offset	Field	Comments
0	Address [7:0]	The start address of the requested bulk transfer.
1	Address [15:8]	
2	Address [23:16]	
3	Size [7:0]	The size of transfer.
4	Size [15:8]	
5	Command	**Bit 7**: '1' starts the bulk transfer by DMA; '0' starts the bulk transfer by PIO. **Bit 0**: '1' IN token; '0' OUT token.

8.2. Host Side Programming Considerations

The USB device is not the only criterion that decides the transfer rate. The performance of the host side application plays a more important role in the overall system performance because the host always controls USB transactions.

In the sample firmware, either the bulk or ISO transfer is a sequential operation that involves both the control endpoint and the bulk or ISO endpoint. Cooperation is important because the next step of operation is determined by the result of the last operation. While multithreads can be used to access different pipes to increase system performance, it makes programming much easier to process Setup Vendor Request (IOCTL) and data transfer (WriteFile or ReadFile) operations on the main endpoints by using a single thread.

IOCTL_WRITE_REGISTER and IOCTL_READ_REGISTER use structure IO_BLOCK to exchange data with the device driver. The following structure definition is part of the Microsoft Still Image USB Interface:

```
typedef struct _IO_BLOCK {
    IN          unsigned        uOffset;
    IN          unsigned        uLength;
    IN          OUT  PUCHAR     pbyData;
    IN          unsigned        uIndex;
} IO_BLOCK, *PIO_BLOCK;
```

The IO_REQUEST structure is a proprietary definition that contains details of the Setup Vendor Request.

```
typedef struct _IO_REQUEST {
    unsigned short      uAddressL;
    unsigned char       bAddressH;
    unsigned short      uSize;
    unsigned char       bCommand;
} IO_REQUEST, *PIO_REQUEST;
```

See the following sample code:

```
ioRequest.uAddressL = 0;
ioRequest.bAddressH = 0;
```


© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

```
ioRequest.uSize = transfer_size;
ioRequest.bCommand = 0x80;        //start, write

ioBlock.uOffset = 0;
ioBlock.uLength = sizeof(IO_REQUEST);
ioBlock.pbyData = (PUCHAR)&ioRequest;
ioBlock.uIndex = 0x471;

bResult = DeviceIoControl(hDevice,
            IOCTL_WRITE_REGISTERS,
            (PVOID)&ioBlock,
            sizeof(IO_BLOCK),
            NULL,
            0,
            &nBytes,
            NULL);

if (bResult != TRUE) {
            testDlg->MessageBox("Setup DMA request failed!", "Test Error");
            return;
            }

bResult = WriteFile(hFile,
            pcIoBuffer,
            transfer_size,
            &nBytes,
            NULL);
```

9. References

- *Universal Serial Bus Specification Rev. 2.0*
- *ISP1183 low-power Universal Serial Bus interface device* data sheet

© Koninklijke Philips Electronics N.V. 2003. All rights reserved.

Philips Semiconductors

Philips Semiconductors is a worldwide company with over 100 sales offices in more than 50 countries. For a complete up-to-date list of our sales offices please e-mail
sales.addresses@www.semiconductors.philips.com.
A complete list will be sent to you automatically.
You can also visit our website
http://www.semiconductors.philips.com/sales/

www.semiconductors.philips.com

© Koninklijke Philips Electronics N.V. 2003
All rights reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey or imply any license under patent – or other industrial or intellectual property rights.