Semiconductors

AN10004_4

ISP1581 Programming Guide

Rev. 04 — 1 March 2004 Application note

Document information

Info	Content
Keywords	ISP1581, USB, programming guide
Abstract	This programming guide provides a brief introduction on how to implement the ISP1581 with hardware and firmware guidelines on interfacing to a generic processor with a 16-bit data bus width.

PHILIPS

Revision history

Rev	Date	Description
4.0	Mar 2004	• Updated Section 14. • Updated Table 3-1. • Applied the latest corporate template.
3.0	Aug 2003	• Added Section 7.
1.1	Nov 2002	• Section 11: changed the last sentence • Changed USB 2.0 to Hi-Speed USB and USB 1.1 to Original USB • Upgraded to the latest template
1.0	March 2002	First release.

Contact information

For additional information, please visit: ***http://www.semiconductors.philips.com/***

For sales office addresses, please send an email to: ***sales.addresses@www.semiconductors.philips.com***

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.

CONTENTS

FIGURES

TABLES

1. Introduction

The ISP1581 is a Hi-Speed Universal Serial Bus (USB) interface device that provides a flexible interface for various ranges of microcontrollers. The high-speed microcontroller interface increases system throughput and reduces processor utilization.

This document provides a brief introduction on how to implement ISP1581 with hardware and firmware guidelines on interfacing to a generic processor with a 16-bit data bus width. Note: Do not confuse the hardware-related descriptions in this document with those of the ISP1581's Split Bus mode.

2. ISP1581 Microcontroller Interface Signals

The ISP1581 provides a flexible configuration for the microcontroller interface. For most microprocessors, no glue logic is required.

The following tables provide the typical connections for ISP1581 pins.

Table 2-1: Microcontroller Interface Signal Connection

ISP1581 Signal	Microcontroller	Remarks
AD[0][1]	No connection	AD[0] must be connected to the ground on the ISP1581 side at 16-bit bus width
AD[7:1][2]	Address bus lines 7 to 1	—
$\overline{CS}$	System-decoded chip selector	Must be located in the 16-bit access bank
DATA[15:0][3]	16-bit data bus	—
$\overline{DS}$ / $\overline{WR}$	WR	Use write strobe
INT	Any interrupt input of the microcontroller	—
(R/ $\overline{W}$)/ $\overline{RD}$	RD	Use read strobe

Table 2-2: Direct Memory Access (DMA) Signal Connection

ISP1581 Signal	DMA Controller	Remarks
DREQ[4]	DMA request input	—
DACK[5]	DMA acknowledge output	—
DIOR	DMA read signal	Short to the ISP1581 $\overline{RD}$ pin when the DMA Controller uses the same read strobe as the microprocessor[6]
DIOW	DMA write signal	Short to the ISP1581 $\overline{WR}$ pin when the DMA Controller uses the same write strobe as the microprocessor[7]
EOT	DMA transfer end output	—

1 The 16-bit interface requires all address calls to be even. Therefore, AD[0] is not connected because some firmware compilers create confusing word alignment codes.
2 AD[7:0] in the Generic Processor mode is an address bus. In the Split Bus mode, it is used as a multiplexed address and data bus to control the ISP1581.
3 DATA[15:0] in the Generic Processor mode is used as a DMA bus as well as a system bus through which the ISP1581 is controlled. In the Split Bus mode, however, it is used only as a DMA bus.
4 The DMA core of the ISP1581 can be used as a DMA master or a DMA slave depending on the initiating opcode (DMA Command register, address: 30H). DREQ and DACK remain as high-Z until the DMA command is executed.
5 See note 4.
6 When ISP1581 is operating in the DMA ACK only mode, the read and the write signals must be connected to HIGH.
7 See note 6.

The ISP1581 evaluates the pin configuration level at power-on reset to determine the operation mode (see Table 2-3).

Table 2-3: Configuration Setting

ISP1581 Signal	Signal Level at Reset	Configuration
BUS_CONF	1	Generic microprocessor interface; 16-bit data bus and 8-bit address lines[1]
MODE0	1	ISP1581 detects $\overline{RD}$ for the read operation and $\overline{WR}$ for the write operation
ALE	1	If ALE is not used, pull it LOW

3. Initializing Registers

The firmware must initialize the ISP1581 registers to configure I/O signal levels to match their system setups. The following tables provide general initialization of the Mode, Interrupt Configuration, DMA Configuration and DMA Hardware registers of the ISP1581.

Table 3-1: Mode Register (0CH)

Register Bit (Hex)	Register Bit Symbol	ISP1581 Signal	Remarks
0C.7	CLKAON	No corresponding signal	***0***—turning off the clock reduces power consumption in the suspend state
0C.3	GLINTENA	INT	***1***—globe interrupt enable
0C.2	WKUPCS	$\overline{CS}$ and WAKEUP	***1***—activates $\overline{CS}$ and wakes up the ISP1581 from the suspend state (WAKEUP retains the same function for all settings)
0C.0	SOFTCT	RPU	***0***—1.5 kΩ resistor on the RPU pin is internally disconnected from DP ***1***—1.5 kΩ resistor on the RPU pin is internally connected to DP (performs Original USB full-speed function)

Table 3-2: Interrupt Configuration Register (10H)

Register Bit (Hex)	Register Bit Symbol	ISP1581 Signal	Remarks
10.1	INTLVL	INT	***1***—interrupt only generates a pulse on the INT pin ***0***—interrupt remains in the active state, if there is any interrupt event pending
10.0	INTPOL	INT	***1***—INT pin remains in or goes HIGH if there is an interrupt ***0***—INT pin remains in or goes LOW if there is an interrupt

1 Typically, A0 is connected to the ground and the microcontroller accesses the ISP1581 at the16-bit data alignment.

Table 3-3: DMA Configuration Register (38H)

Register Bit[1] (Hex)	Register Bit Symbol	ISP1581 Signal	Remarks
38.7	DIS_XFER_CNT	EOT	***1***—disabling the DMA counter will force the end of DMA transfer to fully depend on the assertion of the End-of-Transfer (EOT) signal
38.6 to 4	BURST[2:0]	DREQ and DACK	Determines the handshake of the DREQ signal
38.3 to 2	MODE[1:0]	DIOR, DIOW and DACK	Determines whether to use DIOR or DIOW, and how to work with DACK
38.0	WIDTH	DATA[15:0]	***1***—DATA[15:0] is used as a 16-bit data bus for DMA ***0***—only DATA[7:0] is used in DMA (DMA is in the 8-bit mode.)

Table 3-4: DMA Hardware Register (3CH)

Register Bit[2] (Hex)	Register Bit Symbol	ISP1581 Signal	Remarks
3C.7 to 6	ENDIAN[1:0]	DATA[15:0]	***00***—little endian; MSB on DATA[15:8] and LSB on DATA[7:0] ***01***—big endian; MSB on DATA[7:0] and LSB on DATA[15:8]
3C.5	EOT_POL	EOT	***1***—EOT active level is HIGH
3C.4	MASTER	DREQ, DACK, DIOR and DIOW	Signal direction changes at the DMA master or slave function
3C.3	ACK_POL	DACK	***1***—DACK remains HIGH when it is active
3C.2	DREQ_POL	DREQ	***1***—DREQ asserts HIGH when it is active
3C.1	WRITE_POL	DIOW[3]	***1***—DIOW active is at HIGH pulse and strobe is at the falling edge ***0***—DIOW active is at LOW pulse and strobe is at the rising edge
3C.0	READ_POL	DIOR[4]	***1***—DIOR active is at HIGH pulse and strobe is at the falling edge ***0***—DIOR active is at LOW pulse and strobe is at the rising edge

4. ISP1581 Firmware

USB is a master-to-slave structure. Figure 4-1 shows the firmware structure of the ISP1581, and Table 4-1 describes the various files.

The device does not initiate any transmission and responds only to requests from the host. In this architecture, the firmware always waits for the host command before branching to process routings.

The mainloop.c file keeps track of the USB events that come from an interrupt and dispatches them to the corresponding process routings.

1 DMA registers are reset at power-on reset, hardware reset, software reset and DMA reset.
2 DMA registers are reset at power-on reset, hardware reset, software reset and DMA reset.
3 Although, DIOR and DIOW can be set to different modes, $\overline{RD}$ and $\overline{WR}$ cannot be set to these modes. Therefore, the system must work with different hardware connections and may require other logic for DMA.
4 See note 2.

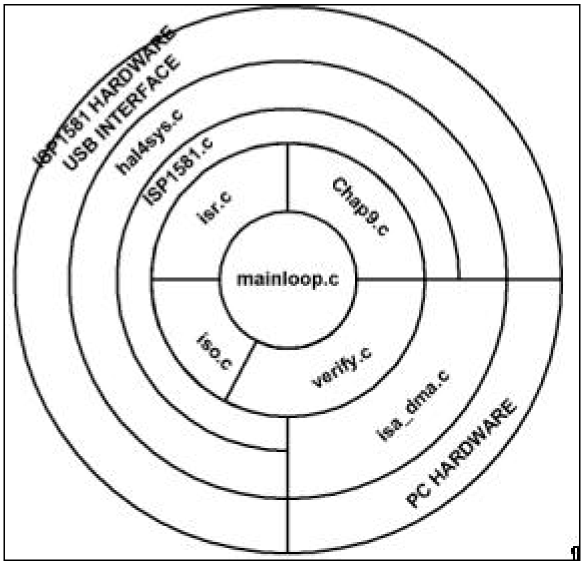

Figure 4-1: Firmware Structure

Table 4-1: Firmware Structure

File Name	***Function***
mainloop.c	Loops and scans USB events; initiates the device and the system
isr.c	Interrupt service routing; evaluates interrupt events and passes the event message to other processes
Chap9.c	Contains code for the standard USB command that is used to establish a basic connection between the device and the host
verify.c	Contains vendor-specified command process for the demo application that is used to verify data integrity
iso.c	Performs function similar to verify.c but sets up the transfer for isochronous (ISO) data
isa_dma.c	Sets up the DMA Controller for the DMA data path between the ISP1581 and the local memory (The file name depends on the microprocessor used.)
ISP1581.c	The ISP1581 command set; the low-level layer forms access to the ISP1581 registers and the data port
hal4sys.c	Hardware interface configuration for the break board between the ISP1581 board and the microprocessor main board. With direct connection to the microprocessor, this layer is not necessary.

4.1. Process Flow

There are many ways in which data flows between the USB host and device. The path of data flow is called a 'pipe'. In the ISP1581 sample-testing applet (see Figure 15-2), the following types of pipes are used:

- Control pipe (also called the default pipe) for USB control transfers
- Bulk-IN and Bulk-OUT data pipes
- ISO-IN and ISO-OUT data pipes.

The control pipe consists of the SETUP, control OUT and control IN endpoints. All other pipes contain an endpoint each at a unique direction.

The USB host sets up and controls data flow in the data pipe by using the Setup command through the control pipe. The command transfer is performed in three stages: Setup stage, Data stage (contents supplement command parameters or other information) and Status stage. An example of the data structure set up is shown in Figure 4-2, and the corresponding firmware routing is given in Figure 4-3.

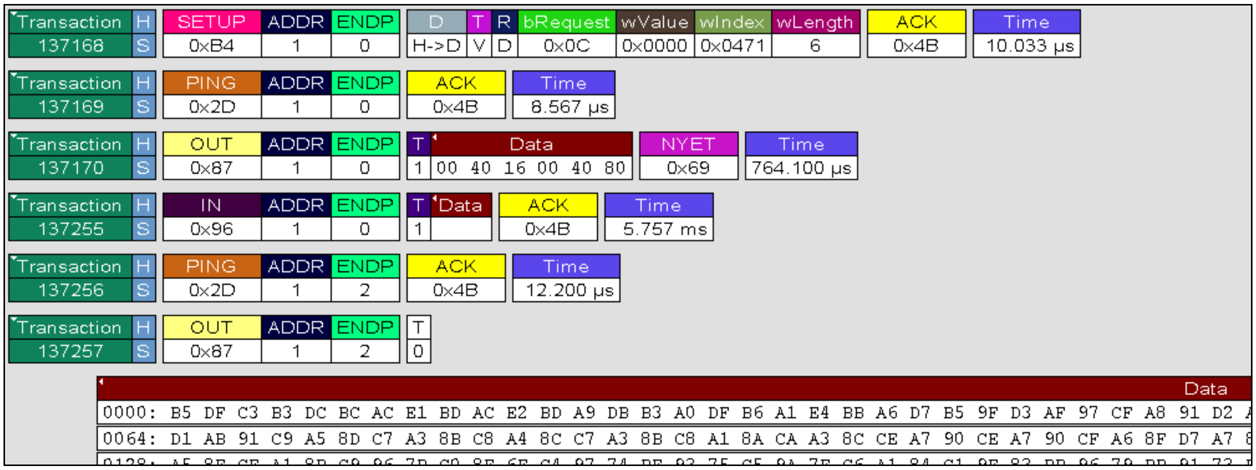

Figure 4-2: Example of a Data Structure Set Up

In this example, the SETUP packet indicates a vendor-specific command for the Bulk data transfer that is requested by the host. The host sends a control OUT packet and gives information on how many bytes of data will be sent. The device must facilitate DMA for the data transfer to the local memory. The control IN transfer completes the whole setup transfer. This indicates that both sides have received the command and also the status of the process of the command successfully. The Bulk-OUT transfer starts when the host and the device have made the appointment.

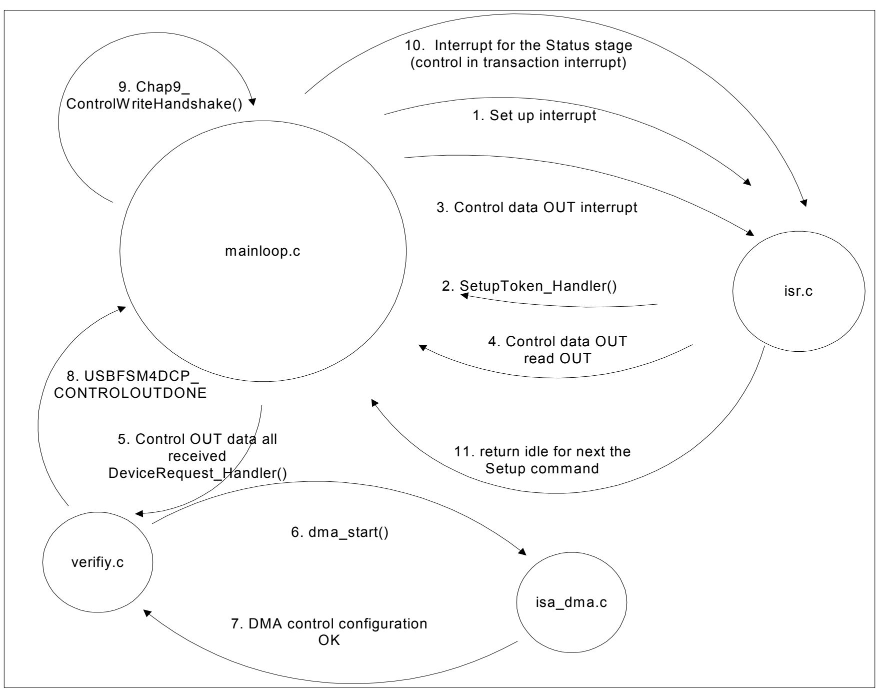

Figure 4-3: Firmware Routing for the Example

4.1.1. USB Command Process Flow

The USB host manipulates the device by using the control transfer in three stages: Setup, Data and Status. Figure 4-4 contains the flowchart of the USB command handling.

The device must follow the various states of the host and reply according to the request.

The Data stage is optional; if there is no Data stage, the control transfer is indicated as a data OUT command and the length of data is zero. The device must skip directly to the Status stage, and the direction of the Status stage is IN. The device sends a zero-length packet for the IN token to allow the host to acknowledge that the device has successfully received and executed the Setup request. In the ISP1581, the STATUS bit (bit 1) of the Control Function register must be set to logic 1 to acknowledge the generation of ACK or NAK during the Status stage of a SETUP transfer.

IDLE	Device in the idle mode, waiting for the host command;
SETUPPROC	Set up process, Setup command received and interpreted;
REQUESTPROC	Request process, command has been excused by the device;
DATA_IN	Data IN, device has prepared the data as per the host request;
DATA_OUT	Data OUT, device is ready to retrieve data sent by the host;
DATAOUTDONE	Data OUT has completed, the amount of data specified in setup has been received;
CTLRDHDSHAK	Control read handshake, device has completed the control read command, will acknowledge the Status stage;
CTLWRHDSHAK	Control write handshake, device has completed the control write command, will acknowledge the Status stage;
STALL	Device indicates that it cannot complete the host command.

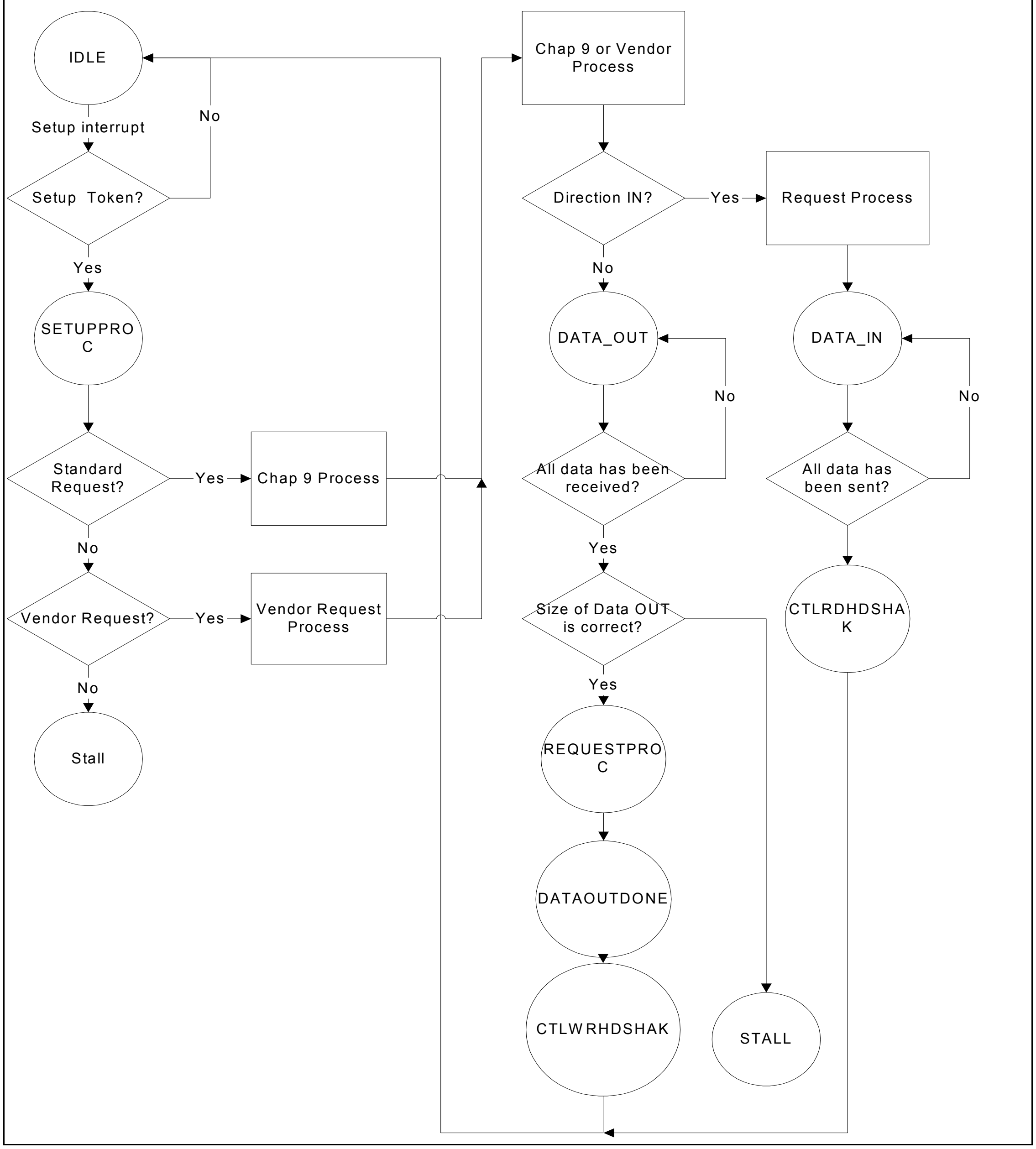

Figure 4-4: Machine State for Setup Processing

5. USB 2.0 Chapter 9 Commands

The host adds a new device when a set of commands that is described in ***Universal Serial Bus Specification Rev 2.0***—enumeration—is completed. This section provides details on the firmware flow of the command process.

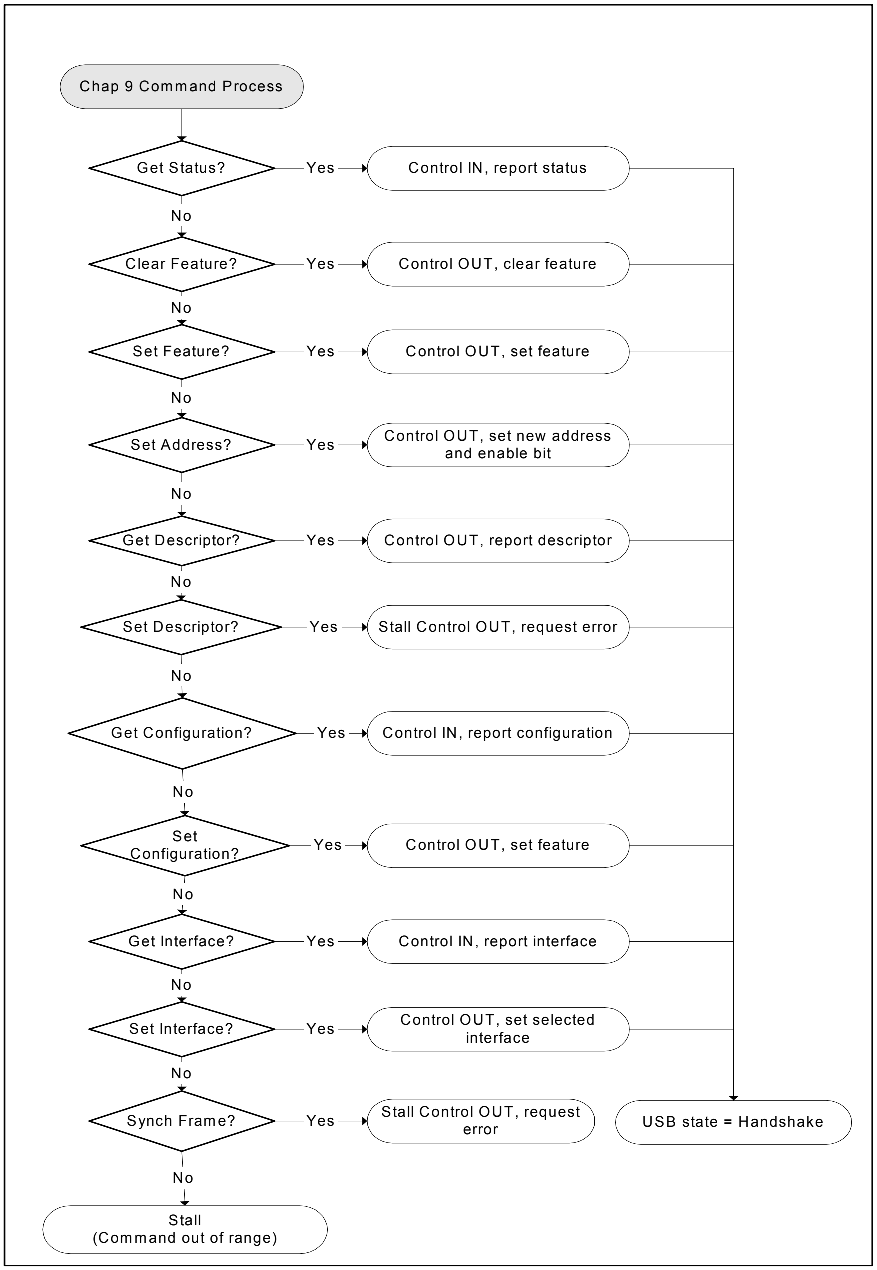

Figure 5-1: Chapter 9 USB 2.0 Standard Command Process Flowchart

5.1. Get Descriptor

USB devices report their attributes using descriptors. A descriptor is a data structure with a defined format. Each descriptor begins with a byte-wide field that contains the total number of bytes in the descriptor followed by a byte-wide field that identifies the descriptor type. The flowchart in Figure 5-2 provides a detailed process to decode the host command and return appropriate descriptor at the host request.

For the exact format of the device descriptor, refer to the ***Universal Serial Bus Specification Rev. 2.0***.

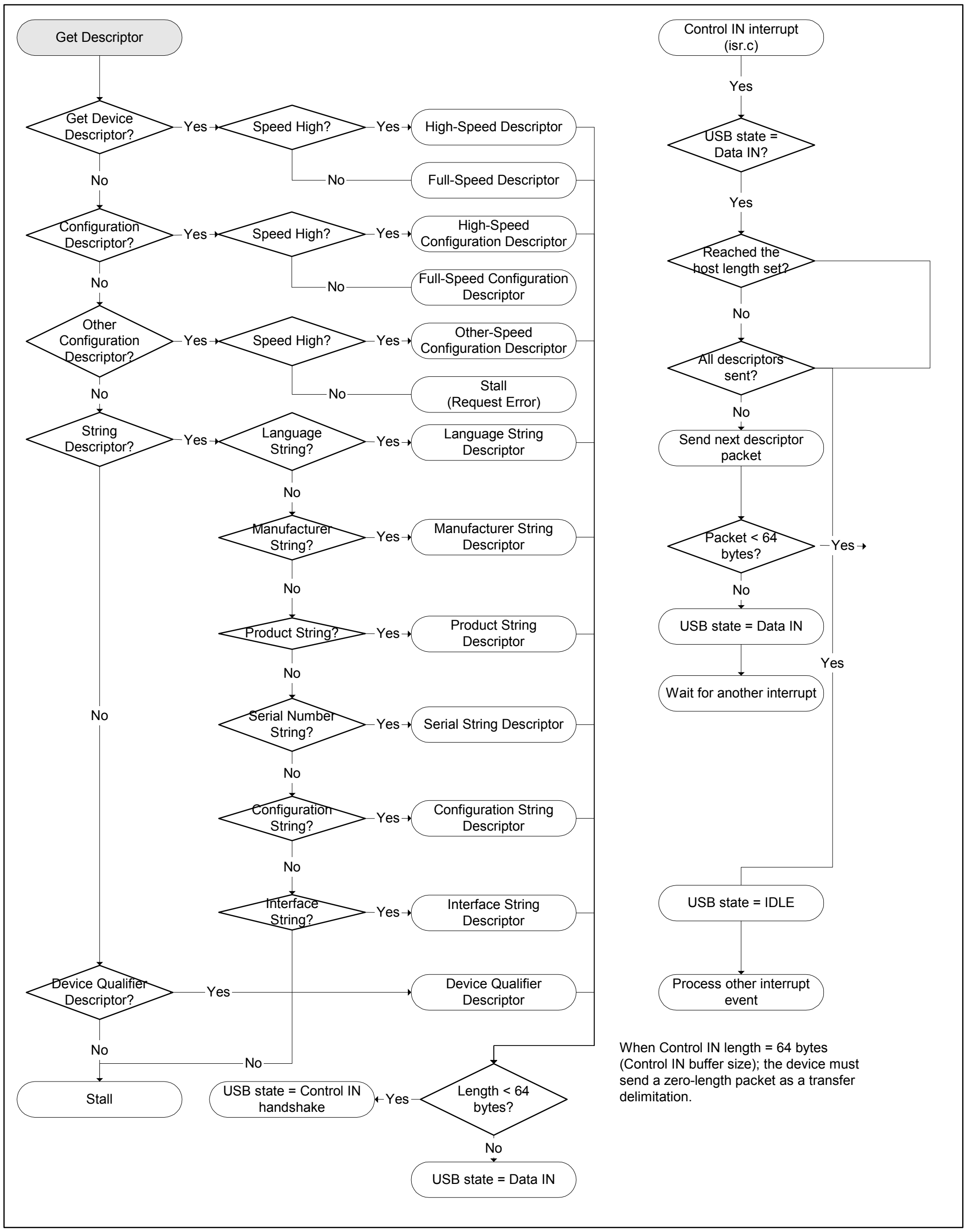

Figure 5-2: Get Descriptor Flowchart

5.1.1. Device Descriptor

The device descriptor provides general information on a USB device. This includes information that applies globally to devices and their configurations. A USB device has only one device descriptor.

The transfer in Figure 5-3 is for the Philips ISP1581 eval kit device descriptor. The device descriptor is high-speed capable and has a version number of 2.0 (0200H, third and fourth bytes in the reverse order).

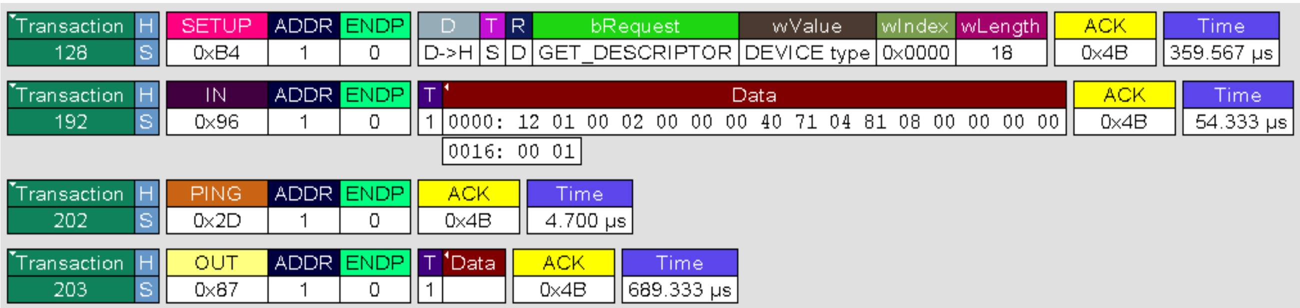

Figure 5-3: Device Descriptor

The following information can be derived from a typical device descriptor as given in Figure 5-3.

Device	USB 2.0 (that is, Hi-Speed USB device)
Vendor ID	0471 (Philips)
Product ID	0881 (Philips Demo Eval Device)

No class specified for this device.

5.1.2. Device_Qualifier Descriptor

The device_qualifier descriptor provides information about a high-speed capable device that would change if the device were operating at other speeds.

5.1.3. Configuration Descriptor (High Speed)

The configuration descriptor (see Figure 5-4) provides information on a specific device configuration. The ISP1581 eval kit consists of two interrupt endpoints and two ISO endpoints. The device with the ISO endpoint must support an alternative interface because of the restriction on bandwidth. In case low bandwidth is available, the USB host selects the default interface without the ISO interface or the interface that requests lower bandwidth.

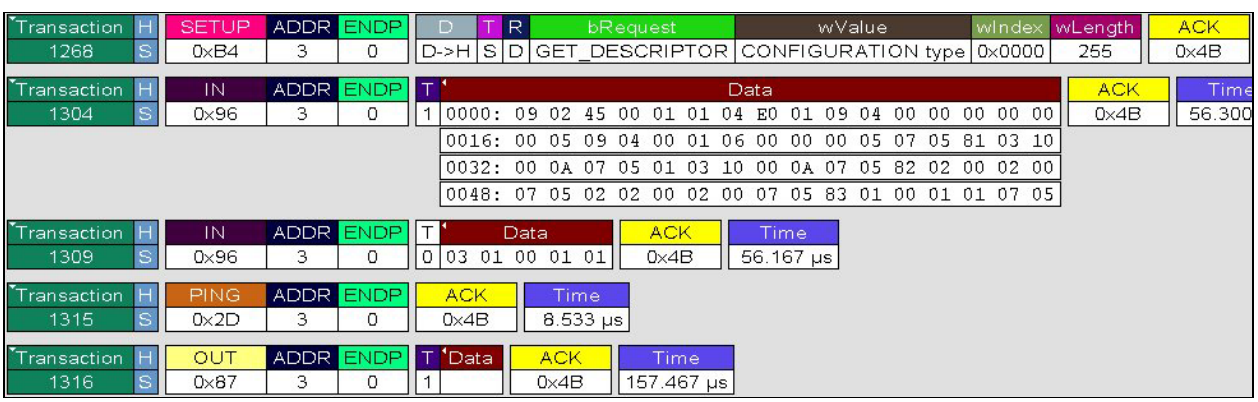

Figure 5-4: Configuration Descriptor

The configuration descriptor includes interfaces and endpoint descriptors. The ISP1581 eval kit has one configuration and one interface with an alternative setting. The default interface does not contain any endpoint. The alternative interface consists of six endpoints as listed in Table 5-1.

Configuration	1
Interface	1 (with alternative settings)
Endpoint number	6

Table 5-1: Endpoint Configuration (in the alternative setting)

Endpoint Descriptor	*Number*	*Direction*	*Size*
Interrupt endpoint	1	OUT	16 bytes
Interrupt endpoint	1	IN	16 bytes
Bulk endpoint	2	OUT	512 bytes
Bulk endpoint	2	IN	512 bytes
ISO endpoint	3	OUT	256 bytes
ISO endpoint	3	IN	256 bytes

5.2. Set Address

The device must acknowledge a new address when the command is completed. The validation of the new address is done by the ISP1581. Once the ACK (of the Status stage) of the IN token is received, the device will immediately respond to the new address (see Figure 5-5 for details).

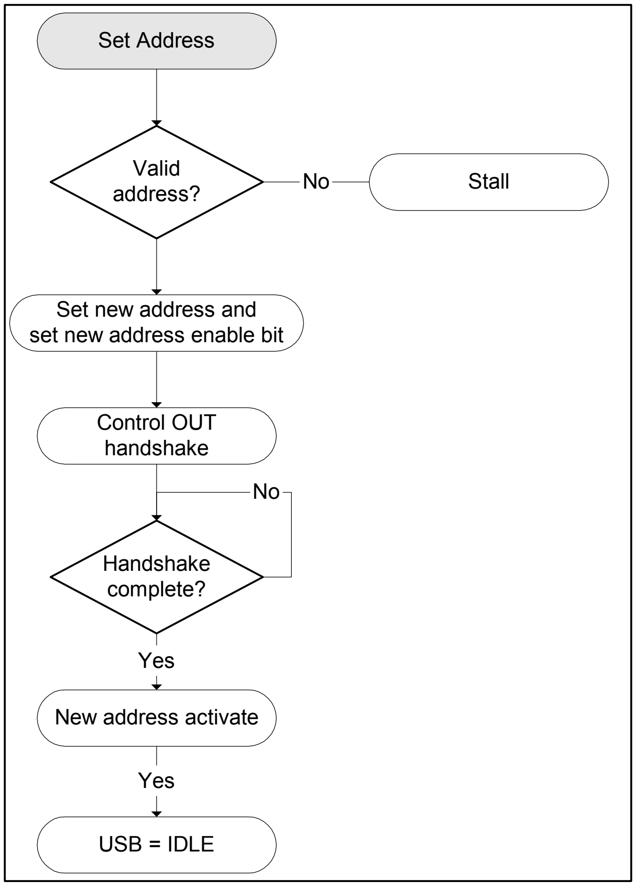

Figure 5-5: Set Address Flowchart

5.3. Set Feature

The recipients of the Set Feature could be a device, interfaces or endpoints. The interface feature of an eval kit is not supported here; and therefore, it is stalled. As a high-speed device, the eval kit fully implements TEST_MODE features, which are facilitated for the USB high-speed electrical test and other compliance tests (see Figure 5-6 for details).

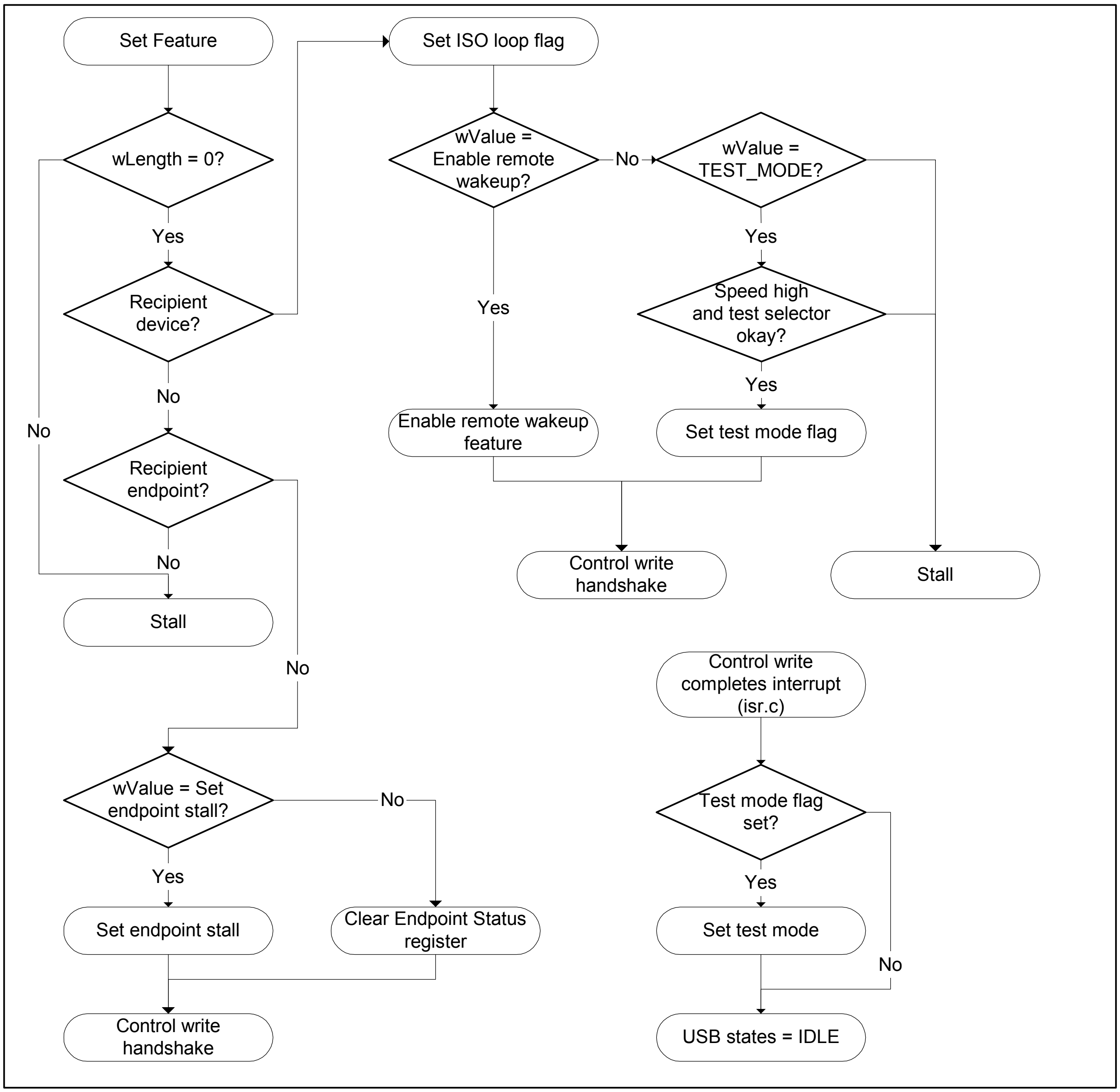

Figure 5-6: Set Feature Flowchart

6. Device Initiation and High-Speed Configuration

A device configuration is cleared by a hardware or software reset. A bus reset clears an endpoint configuration. Therefore, the ISP1581 must be completely initiated at every reset.

When there is a power-on reset or software reset, the device will be disconnected and then connected again (see Figure 6-1).

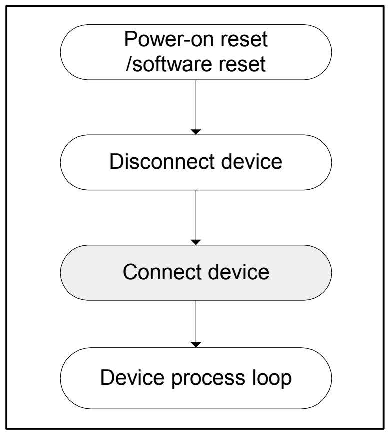

Figure 6-1: Reset Process Flowchart

Figure 6-2 shows the events that require initialization of the ISP1581.

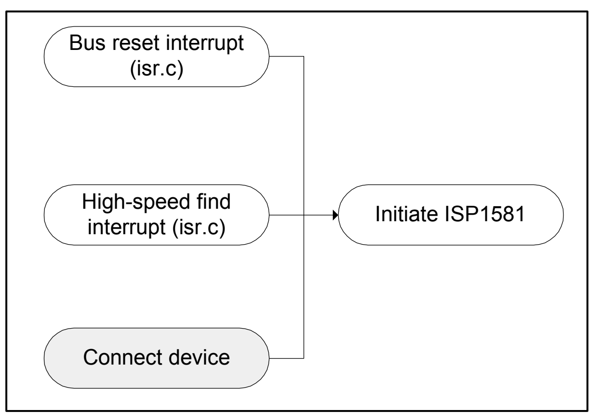

Figure 6-2: Initialization of the ISP1581 Flowchart

An example of the initialization that occurs after a bus reset is shown in Figure 6-3.

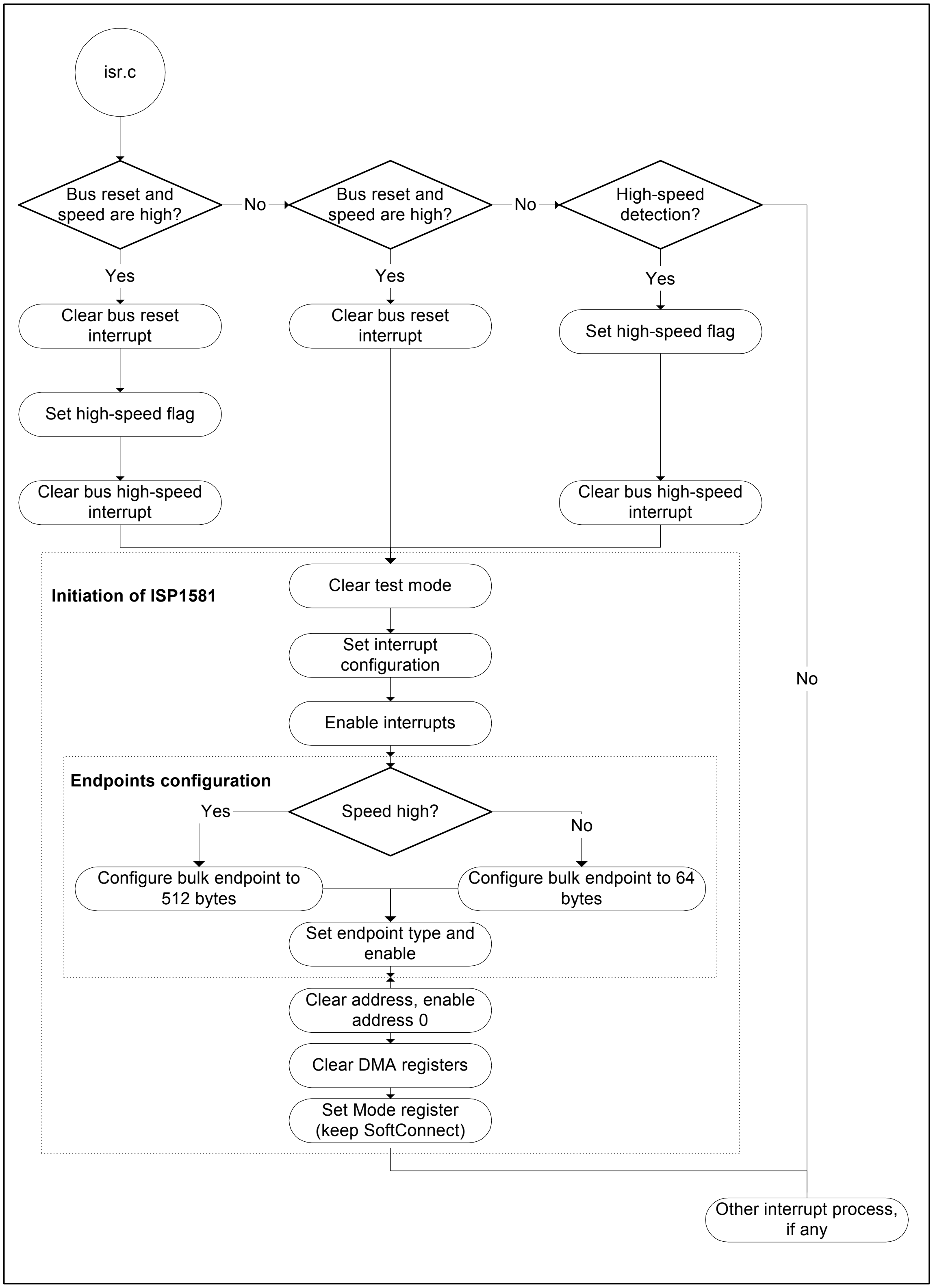

Figure 6-3: Device Configuration and High-Speed Configuration Flowchart

7. Firmware Flow for Setup Tokens

7.1. Setup Token with Data OUT Stage

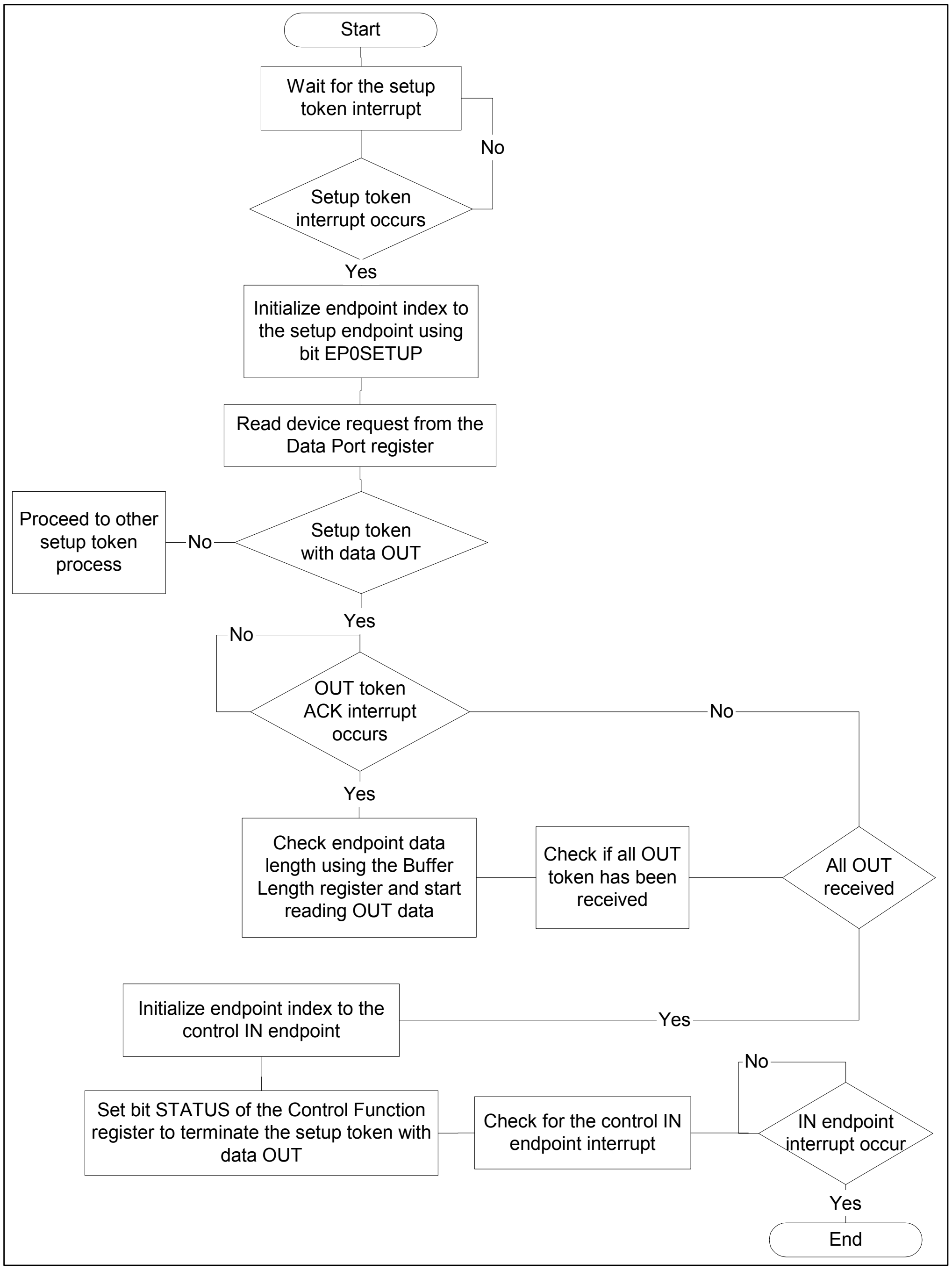

Figure 7-1: Setup Token with Data OUT Stage

7.2. Setup Token with Data IN Stage

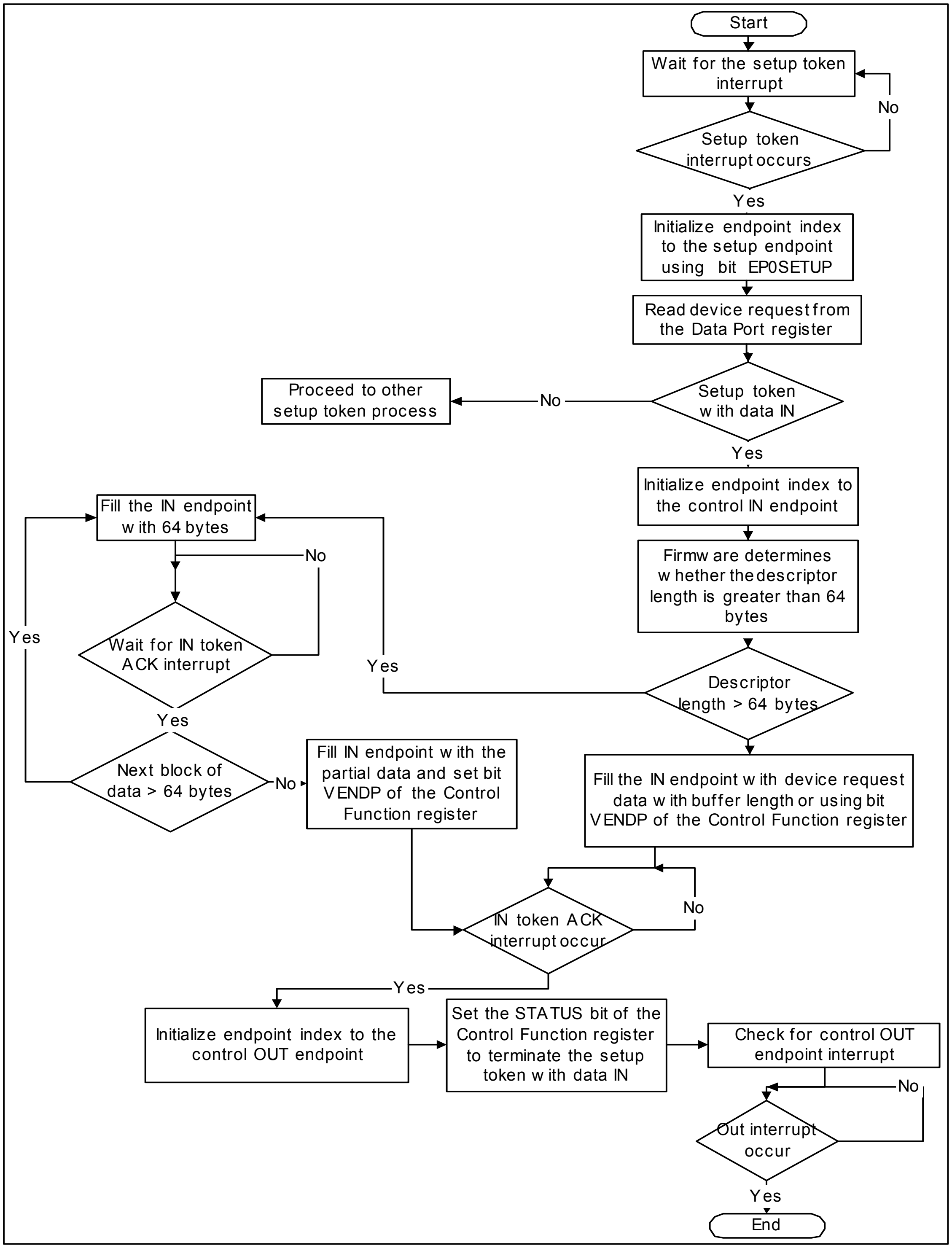

Figure 7-2: Setup Token with Data IN Stage

7.3. Setup Token without Data Stage

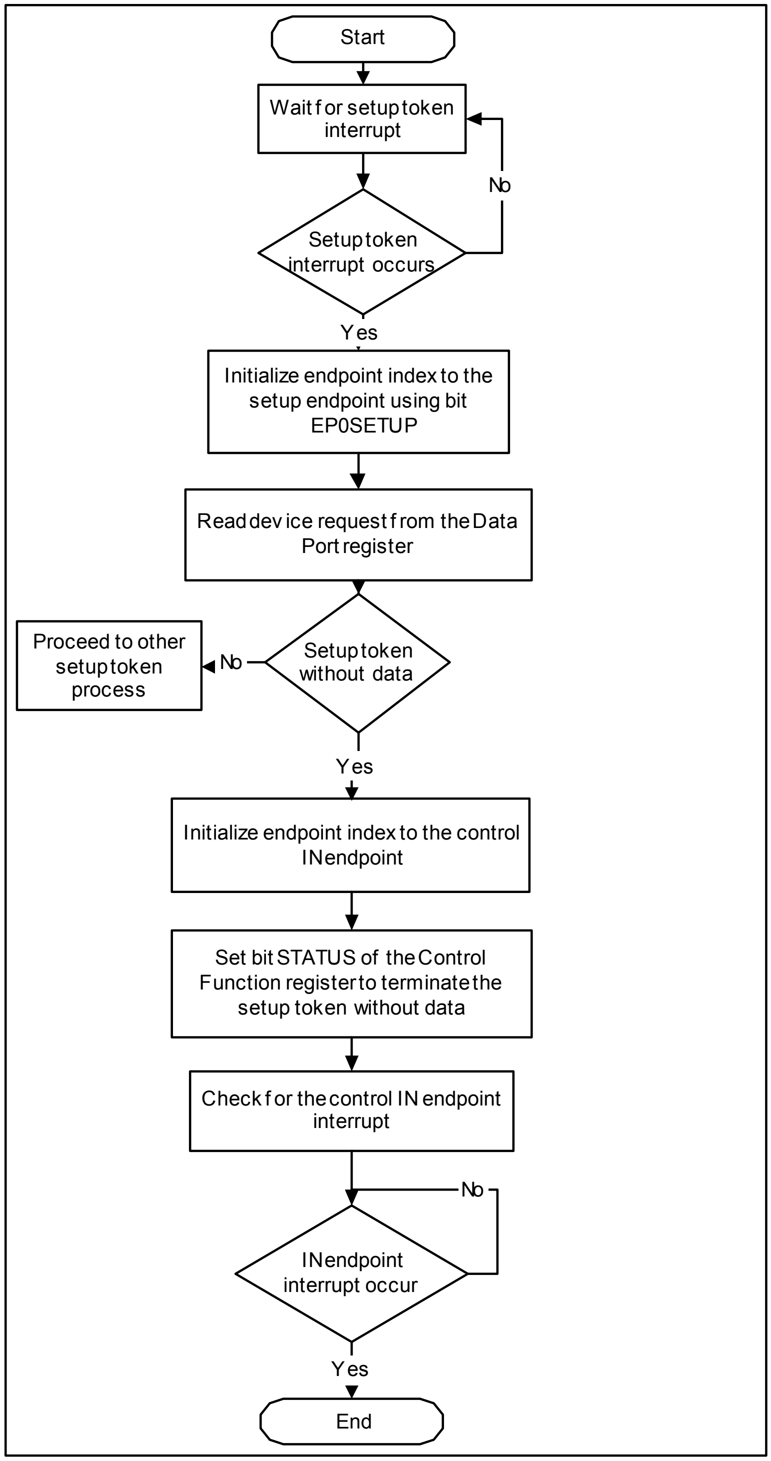

Figure 7-3: Setup Token without Data Stage

7.4. Firmware Flow for Data Transfer

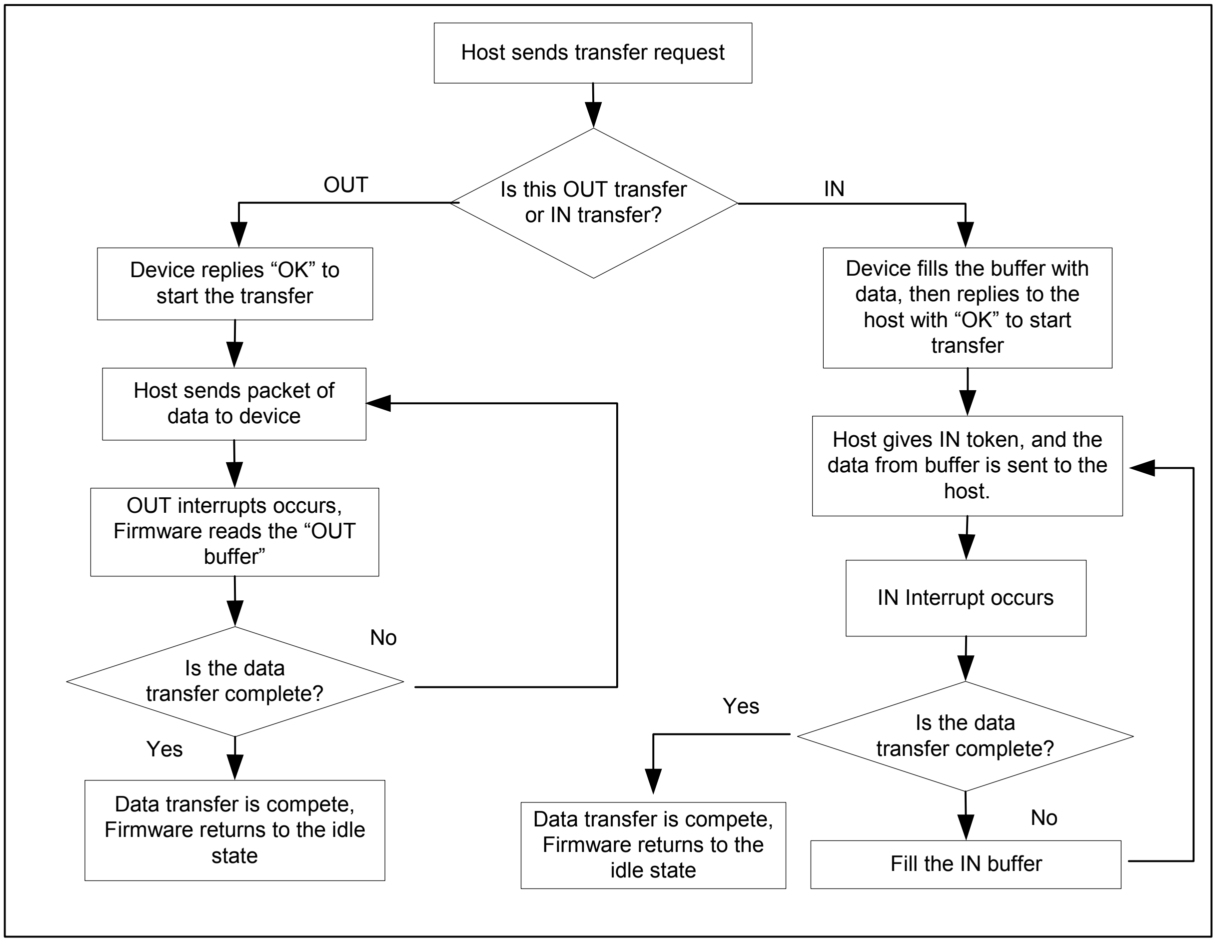

Figure 7-4: Firmware Flow for Data Transfer

8. DMA Transfer Setup

8.1. DMA Reset

The DMA reset clears an incomplete DMA transfer. This restores DMA to its default value, as when the power comes on. This is because there is no DMA stop command for generic DMA (GDMA). Therefore, some means to restart a new cycle of the DMA transfer is required.

When the DMA reset command is issued, already validated packets will remain in the buffer and will be sent to the host, if it is an IN endpoint. If it is an OUT endpoint, data will be cleared only when all the data in the packet are read. Otherwise, the data in the whole packet will be retained in the buffer.

The DMA Clear Buffer and Validate Buffer commands help to discard the last packet read halfway or validate the partially filled data in the buffer.

8.2. Continuous DMA Transfer with the Setup Command

Extra care is needed during the real-time tracking of data transfer between the host and the device because a nonblocking data read or write function on the host translates itself into data packets that can be held over several buffers, depending on the state of execution. For example, in the ISP1581 sample testing applet (see Figure 15-2), each DMA Bulk OUT is preceded by a SETUP token. The host application initiates a back-to-back operation of the Setup followed by the Bulk OUT command to mimic a continuous DMA flow.

From the device point of view, it may see a Setup command of a new cycle, although it is still processing the Bulk data in its buffer from the previous DMA command. Therefore, the firmware programmer must take precautions against such a condition.

8.3. ISP1581 DMA Interfacing with SH-3[1]

The ISP1581 implements the I/O-based access mode; and therefore, some modifications must be made on its signals to meet the requirements of SH-3 (see Table 8-1).

Table 8-1: SH-3 Signal Connection

Using DIOR and DIOW as DMA read and write strobe signals		
ISP1581 Signal	**SH-3**	**Remarks**
$\overline{CS}$	Must not be accessed at the DMA transfer	—
DIOR	Connect to the read strobe of SH-3	—
DIOW	Connect to the write strobe of SH-3	—
DREQ	Reduced to less than two CLKIO of SH-3	See Figure 8-1
Using DACK-only mode		
ISP1581 Signal	**SH-3**	**Remarks**
$\overline{CS}$	Must not be accessed at the DMA transfer	—
DIOR	Connected to V_{CC} 3.3 V	DMA direction depends on the DMA command
DIOW	Connected to V_{CC} 3.3 V	—
DREQ	Reduced to less than three CLKIO of SH-3	See Figure 8-1

8.3.1. DMA Mode Configuration

SH-3 performs memory-to-memory DMA. The chip select signal will always be set for the DMA source and destination. To avoid any confusion, you can map the ISP1581 into two memory locations: one for the PIO access,

1 If not indicated, SH-3 RISC in this document refers to 7709A.

and the other for the DMA access. The system-decoded circuit only asserts $\overline{CS}$ to the ISP1581 at the PIO memory. When performing DMA, set the address to the DMA location so that the $\overline{CS}$ signal is not asserted.

The ISP1581 DMA request signal lasts till the last access of the DMA transfer (see Figure 8-1). The ISP1581 DREQ must not be directly connected to SH-3 because there is a mismatch with SH-3 timing. SH-3 samples two times (that is, it starts two DMA cycles) at the beginning of a clock cycle, regardless of whether DMA has started or not. This occurs even if DREQ is deasserted at the first transfer. In the burst mode, there may even be an extra read/write cycle.

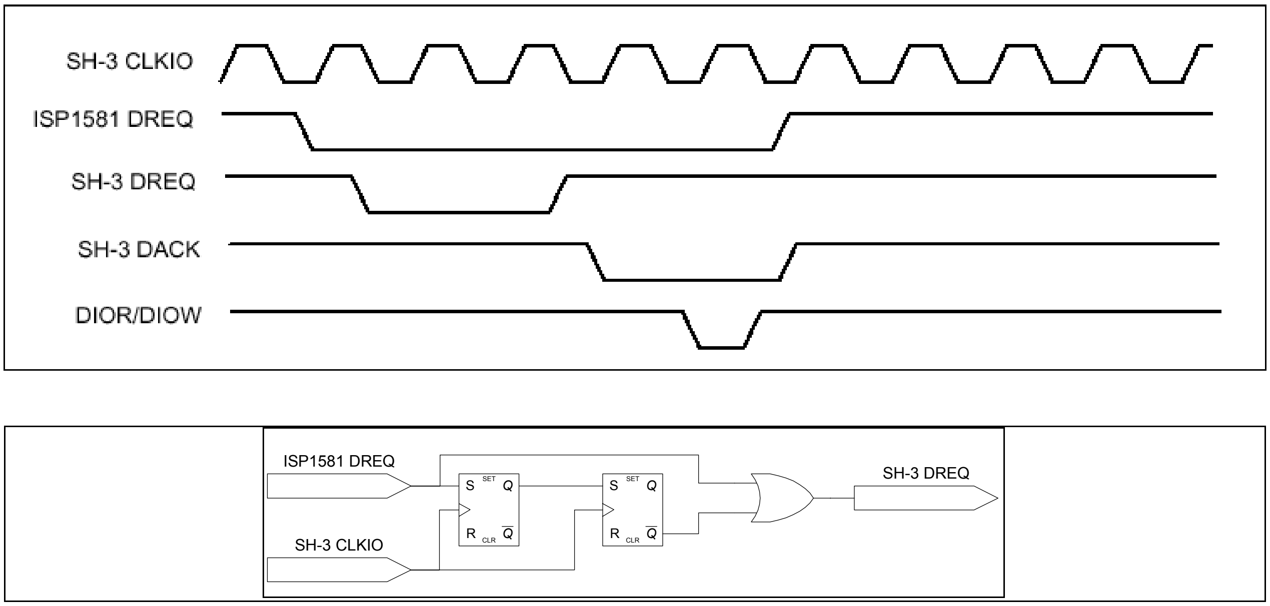

Figure 8-1: DMA Request Signal from ISP1581 to SH-3

SH-3 must be configured in the cycle-steal mode, in which the DMA transfer facilitates an idle cycle of the processor (either the edge or level mode because it presents no difference in the cycle-steal mode).

The ISP1581 must also be programmed in the single-cycle mode, in which DREQ is deasserted at the time of the DMA transfer, and asserted when the transfer is complete and a new cycle is started.

9. ISP1581 STALL Handling

9.1. Function STALL

The Bulk or interrupt endpoint supports the set and clear Halt (known as function STALL) feature. When an endpoint is stalled, it remains in the STALL condition until the host issues a clear Halt feature command through a control pipe. When the Halt feature of an endpoint is cleared (UNSTALL), it clears the buffer (buffers in the case of double buffer) and the data toggle bit is reset to 0 (PID DATA0). The ISP1581 accepts the packet with PID DATA0 for an OUT token and sends a packet with PID DATA0 for an IN token for the first transaction after UNSTALL.

9.1.1.1.Interrupt at Stall

When an endpoint is configured for the ACK only mode, there is no interrupt for a stalled transaction. This does not cause any problem for the firmware because the host issues the CLEAR_FEATURE command for the stalled endpoint.

In the Debug mode, you will get an interrupt. There is, however, no indication that the interrupt you are processing is the one you expected, unless you are sure there is not much traffic and the latency of your firmware is in service before there is any other traffic to the endpoint. The packet that is received later always overwrites the information received earlier.

9.1.2. Stalling an Endpoint and Exiting from a Stall

Stall: You can stall an endpoint at any time. To do this, set the STALL bit in the Endpoint Status register. The Set Endpoint Halt Feature command will also set an endpoint to a Stall condition.

Example: `ISP1581_SetEndpointStatus(bEPIndex, epctlfc_stall);`

The Stall function has the highest priority. Therefore, the host will receive a stall irrespective of whether there is any packet in buffers for the IN endpoint or they are empty for the OUT endpoint.

Exit (Clear): You must clear the STALL bit of the endpoint. Besides, you must disable the endpoint and enable it again to clear data remaining in the buffer, if any.

Example:

```
void Chap9_ClearFeature(void)
{
        unsigned char endp;
        unsigned char bRecipient = ControlData.DeviceRequest.bmRequestType & USB_RECIPIENT;
        unsigned short wFeature = ControlData.DeviceRequest.wValue;
        unsigned short wEPCFG;
        unsigned char dir = ControlData.DeviceRequest.bmRequestType & SB_REQUEST_DIR_MASK;
        if(dir)
                Chap9_StallEP0InControlRead();

        if( ControlData.DeviceRequest.wLength == 0 )
        {
          switch(bRecipient)
          {
              case USB_RECIPIENT_DEVICE:
                       //....
              case USB_RECIPIENT_ENDPOINT:
                 if( ControlData.DeviceRequest.wIndex & USB_ENDPOINT_DIRECTION_MASK)
                       endp = (ControlData.DeviceRequest.wIndex*2 + 1);
                else
                       endp = (ControlData.DeviceRequest.wIndex*2);

                if(wFeature == USB_FEATURE_ENDPOINT_STALL)
                {
                // Clear the data toggle bit to (set to 0) and clear buffers before clear stall of
                // the endpoint.
                       wEPCFG = ISP1581_GetEndpointConfig(endp);
                       ISP1581_SetEndpointConfig(endp, 0); // disable endpoint *
                // Enable endpoint, clear the buffer and set the data toggle bit to 0.
                       ISP1581_SetEndpointConfig(endp, wEPCFG);
                       ISP1581_SetEndpointStatus(endp, 0);   // clear the Stall condition of the
                                                             //endpoint.
                       Chap9_ControlWriteHandshake();
                }
                else
                {
                       Chap9_StallEP0InControlWrite();
                }
            break;

            default:
```

```
                        Chap9_StallEP0InControlWrite();
                        break;
                 }
            }
            else
            {
                 Chap9_StallEP0InControlWrite();
            }
}
```

Note: Disabling and re-enabling an endpoint does not affect the endpoint memory (buffer) allocation. All the other endpoints operate as usual.

9.2. Protocol Stall

Protocol Stall is applicable only to control pipes. The ISP1581 supports only one control pipe that consists of the SETUP, control OUT (endpoint index 0 OUT) and control IN (endpoint index 0 IN) endpoints. It is also the default control pipe.

Although both the control OUT and control IN endpoints in the Endpoint Status register have a STALL bit each, physically these bits refer to the same location. There is only one register bit inside the ISP1581 that can be accessed from different entries. This bit can be set either from control OUT or control IN. When it is set, the IN and OUT tokens from the host to the endpoint 0 (OUT or IN) will get a Stall reply. The Stall condition lasts till the setup transaction.

During a setup transaction, data in the control OUT and control IN endpoints are flushed and both the data toggle bits are set to 1.

The control pipe does not have the Halt feature (function Stall).

10. Validating Zero-Length Packet and Short-Length Packet

10.1. OUT Direction

For the OUT direction, the Data Counter register reflects the received packet length (in bytes). If the packet length is odd and the device is configured in the 16-bit mode, the last byte to be read is in the lower position and the higher byte is padded with unknown data.

10.2. IN Direction

For the IN direction, there are two ways to validate a zero-length packet or a short-length packet.

10.2.1. Using Data Counter

The data counter is a byte counter. In the data counter, write the number of bytes of the packet. Then, select the data port and fill it with the packet data. When the data length reaches the value stored in the data counter, the packet is automatically validated. When the value is odd, it means that the length of the packet to be sent is odd in bytes. The last write strobe of the packet validates the lower byte and discards the higher byte.

10.2.2. Using Validate Buffer Command

The Validate Buffer command applies only to the even byte short-length packets. When the Validate Buffer command is issued, the packet is validated and the length of the packet is recorded by the ISP1581.

Note: If the amount of data written is equal to the buffer length and is followed by a Validate Buffer command, an extra zero-length packet is validated.

10.2.3. Zero-Length Packet

Using Data Counter

Writing zero to the data counter validates a zero-length packet.

Using Validate Command

Validating the buffer without filling any data also validates a zero-length packet. If data written to the Data Port register reaches the length of the buffer and a Validate Buffer command is immediately issued, then a full-length packet and a zero-length packet are validated. In this case, the sequence of the packet is the full-length packet, followed by the zero-length packet.

11. Validating Buffer

If the IN endpoint is configured in the double-buffer mode, the only means for the firmware to know whether there is any empty space for new data is by reading the interrupt bit. Therefore, it is not recommended to fill more that one full-length packet data at a time. The firmware must maintain a counter to record the length of data written. If more than a full-length packet of data is written and another buffer is not empty, the ISP1581's behavior becomes unpredictable.

12. Configuring Endpoints

When configuring endpoints, the following sequence must be followed:

1. Disable all endpoints from 2 OUT to 7 IN.
2. Configure the buffer length of all endpoints from 2 OUT to 7 IN. (The endpoints that are not used can be filled with ‘0’s.)
3. Set the Endpoint Configuration register from 2 OUT to 7 IN. (The endpoints that are not used can be set to 0.)

If an endpoint is disabled but the buffer length of the endpoint is not zero, RAM is reserved for the endpoint. The amount of RAM instantiated for all endpoints must not exceed the total amount of RAM size. When the amount of RAM instantiated is more than the total RAM size, the final assignment of RAM to endpoints is not executed.

13. Pull-Up Resistor

The 1.5 kΩ pull-up resistor is connected to the $V_{CCA(3.3)}$ output so that the downstream port can detect a high-speed or full-speed device plug-in. The resistor must not provide current to the DP when V_{BUS} is not powered. For example, when the downstream port is powered down, the microcontroller must poll on V_{BUS} so that whenever power on V_{BUS} is removed it disconnects the pull-up resistor.

The pull-up resistor must be connected to the RPU pin at one end and to the 3.0 V to 3.6 V DC at the other end. When the ISP1581 switches to the high-speed mode, it removes this pull-up resistor. Therefore, it is not recommended to directly connect the resistor to V_{BUS}. Alternatively, you can connect the pull-up resistor to a voltage source that is derived or controlled by V_{BUS} so that it will be turned off when V_{BUS} is removed.

14. Hi-Speed USB Test Mode

For an IN token, a Hi-Speed USB device must support four test modes: Test_J, Test_K, Test_Packet and Test_SEO_NAK.

The Hi-Speed USB host initiates the test mode by using the SET_FEATURE command. The device must complete the Setup command and then go into the test mode within 3 ms.

For the test packet, the firmware must fulfill the test pattern of the control IN endpoint before it sets the test enable bit. Once the test mode is set, the device must be disconnected from the host and a proper termination for the signal measurement must be done manually. The flowchart in Figure 14-1 shows the details of the Hi-Speed USB test mode.

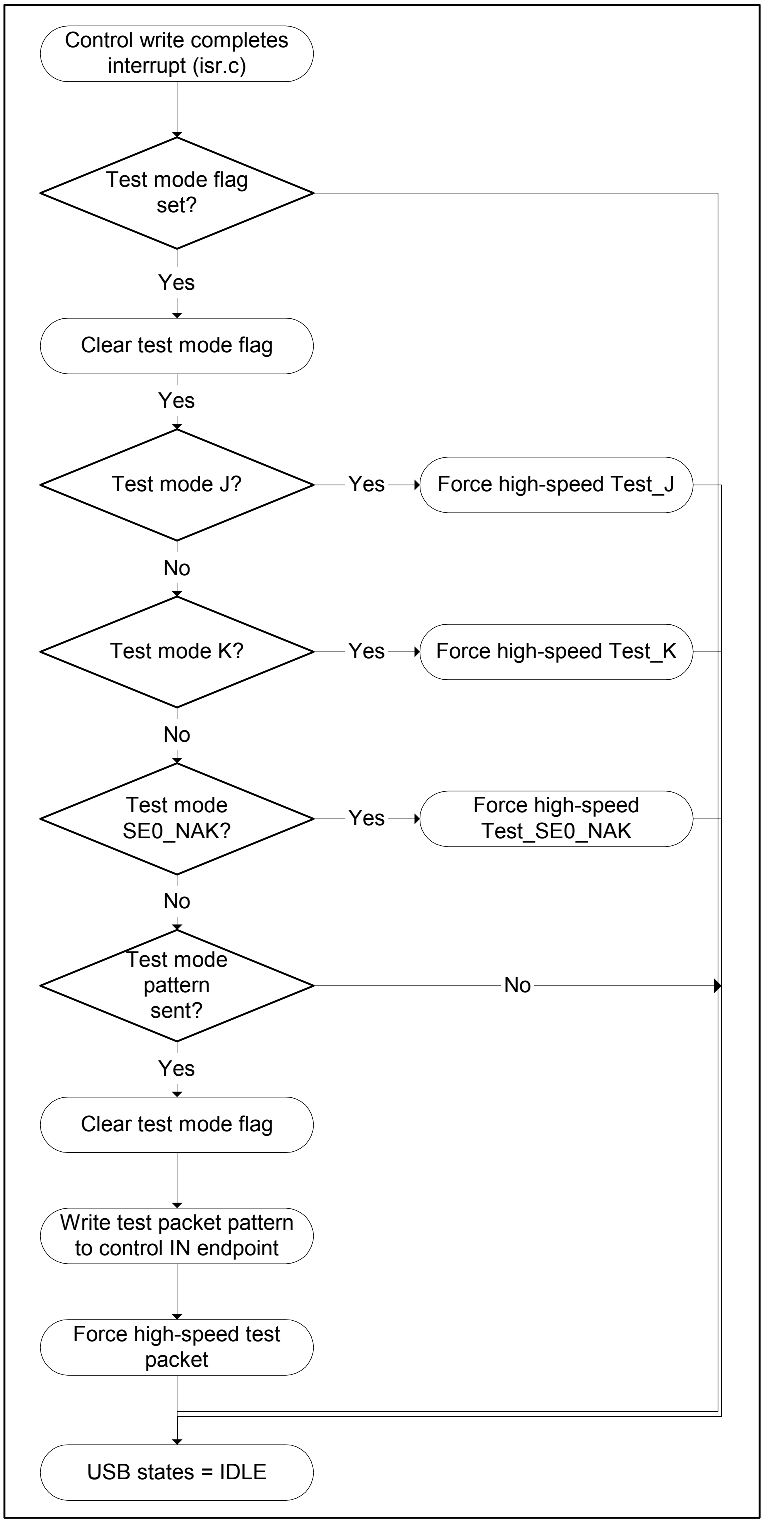

Figure 14-1: Test Mode Flowchart

Structure of the test pattern

```
USB_TESTPACKET  bTestPacket =
{
//      0x00, 0x00,
//      0x00, 0x80, // SYNC pattern will be added by ISP1581
        0xc3,
        0x00, 0x00,
        0x00, 0x00,
        0x00, 0x00,
        0x00, 0x00,
        0x00, 0xaa,
        0xaa, 0xaa,
        0xaa, 0xaa,
        0xaa, 0xaa,
        0xaa, 0xee,  //aa*4
        0xee, 0xee,
        0xee, 0xee,
        0xee, 0xee,
        0xee, 0xfe,  //ee*4
        0xff, 0xff,
        0xff, 0xff,
        0xff, 0xff,
        0xff, 0xff,
        0xff, 0xff,  //FF*11
        0xff, 0x7f,
        0xbf, 0xdf,
        0xef, 0xf7,
        0xfb, 0xfd,
        0xfc, 0x7e,
        0xbf, 0xdf,
        0xef, 0xf7,
        0xfb, 0xfd,
        0x7e,
        0xb6, 0xce   // CRC
//      0xff, 0xf7   // Bit stuff as end of the packet added by ISP1581.
```

15. Application Drivers and Applet Details

15.1. Loading Application Drivers

When the device reports a Philips testing device, the driver for the testing applet is loaded (see Figure 15-1). The driver establishes as many pipes as the device configuration descriptor reports.

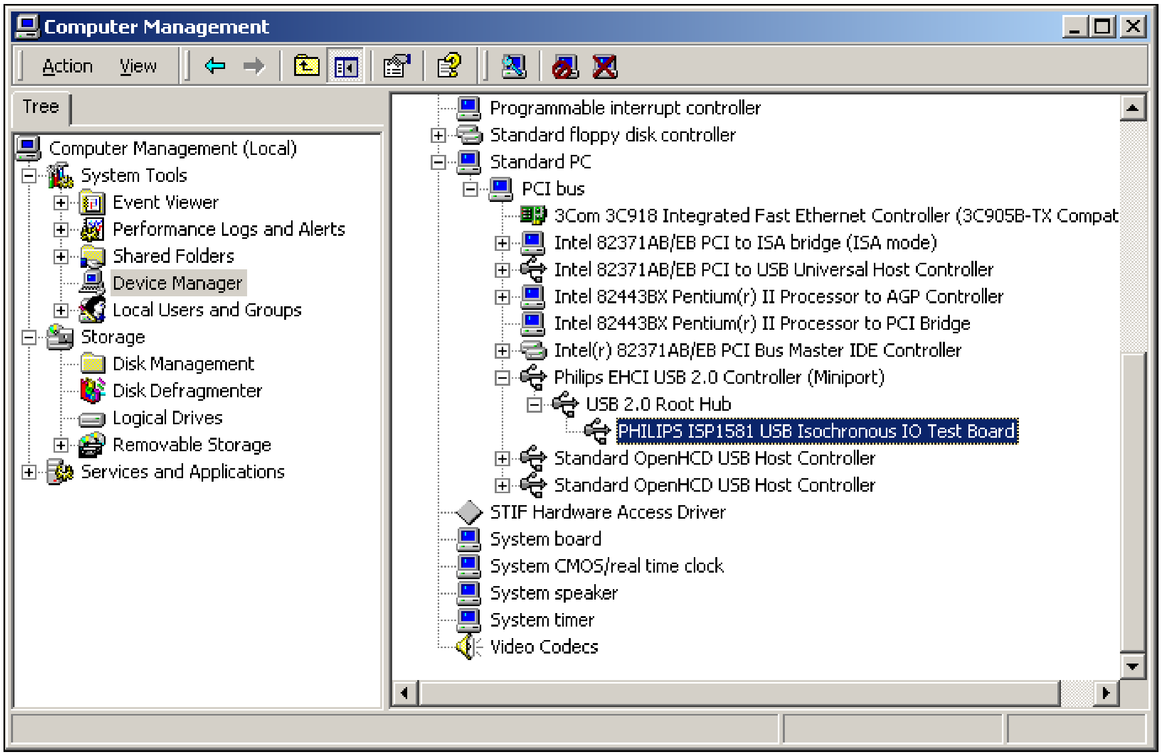

Figure 15-1: Device Driver Loaded

15.2. Testing Applet Application

For testing, a Philips device verification applet is used (see Figure 15-2). There are several vendor-specified commands in the firmware (see Section 16).

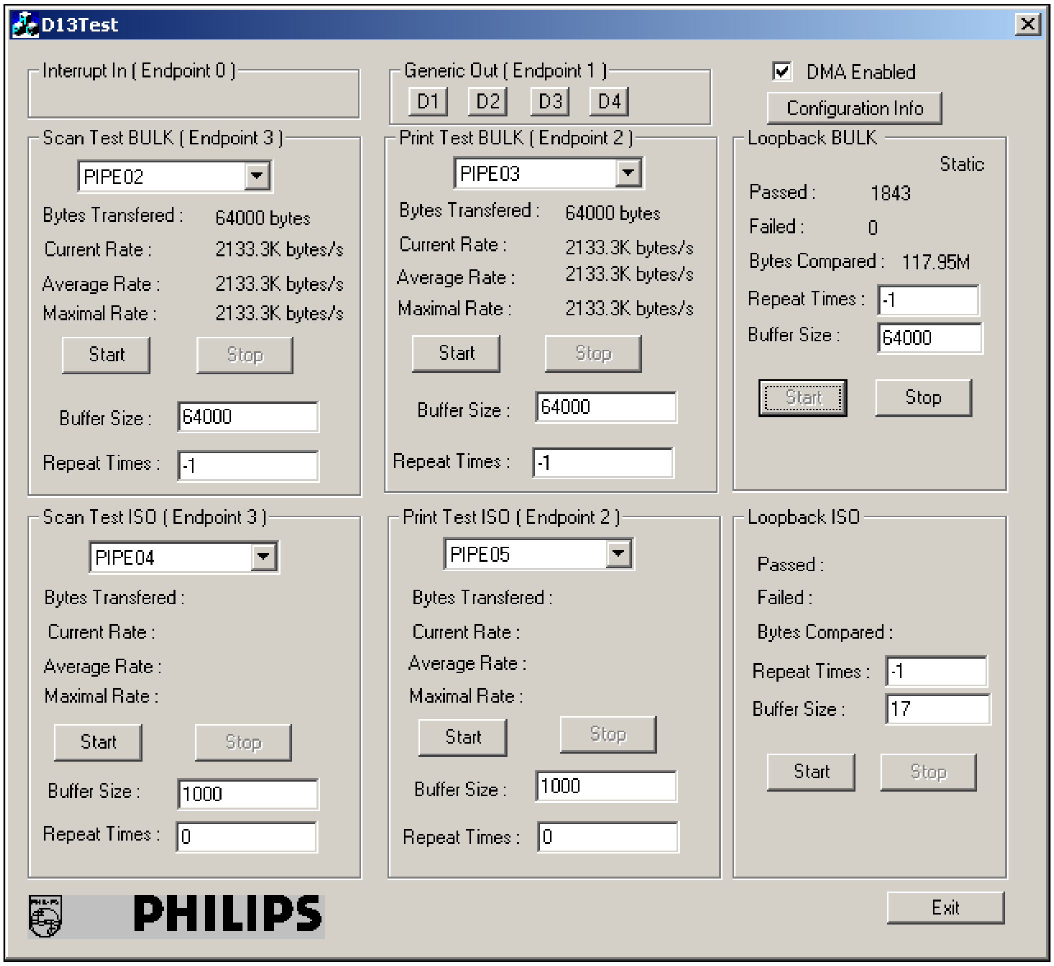

Figure 15-2: Testing Applet

16. Vendor-Specific Command Process Firmware Routines

Several vendor-specific commands are used for setting up the Bulk and ISO data. Figure 16-1 shows the flowchart for the vendor-specific command process. These commands are categorized as:

- Get firmware version (wIndex = GET_FIRMWARE_REQUEST)
- Set and Get TWIN configuration (wIndex = TWIN_CINFIGURATION)
- Write and read Bulk data to or from the device (wIndex = SETUP_DMA_REQUEST)
- Set up the ISO transfer (bmRequest = EnableIsoMode).

List of wIndex for the Bulk data transfer:

```
#define SETUP_DMA_REQUEST                      0x0471
#define GET_FIRMWARE_VERSION                   0x0472
#define TWIN_CONFIGURATION                     0x0473
#define TWIN_CLEAR_CURRENT_FILE                0x0006
#define TWIN_CURRENT_FILE_INDEX                0x0001
#define TWIN_CURRENT_FILE_SIZE                 0x0002
#define TWIN_CURRENT_FILE_INDEX_LENGTH         0x01
#define TWIN_CURRENT_FILE_SIZE_LENGTH          0x04
```

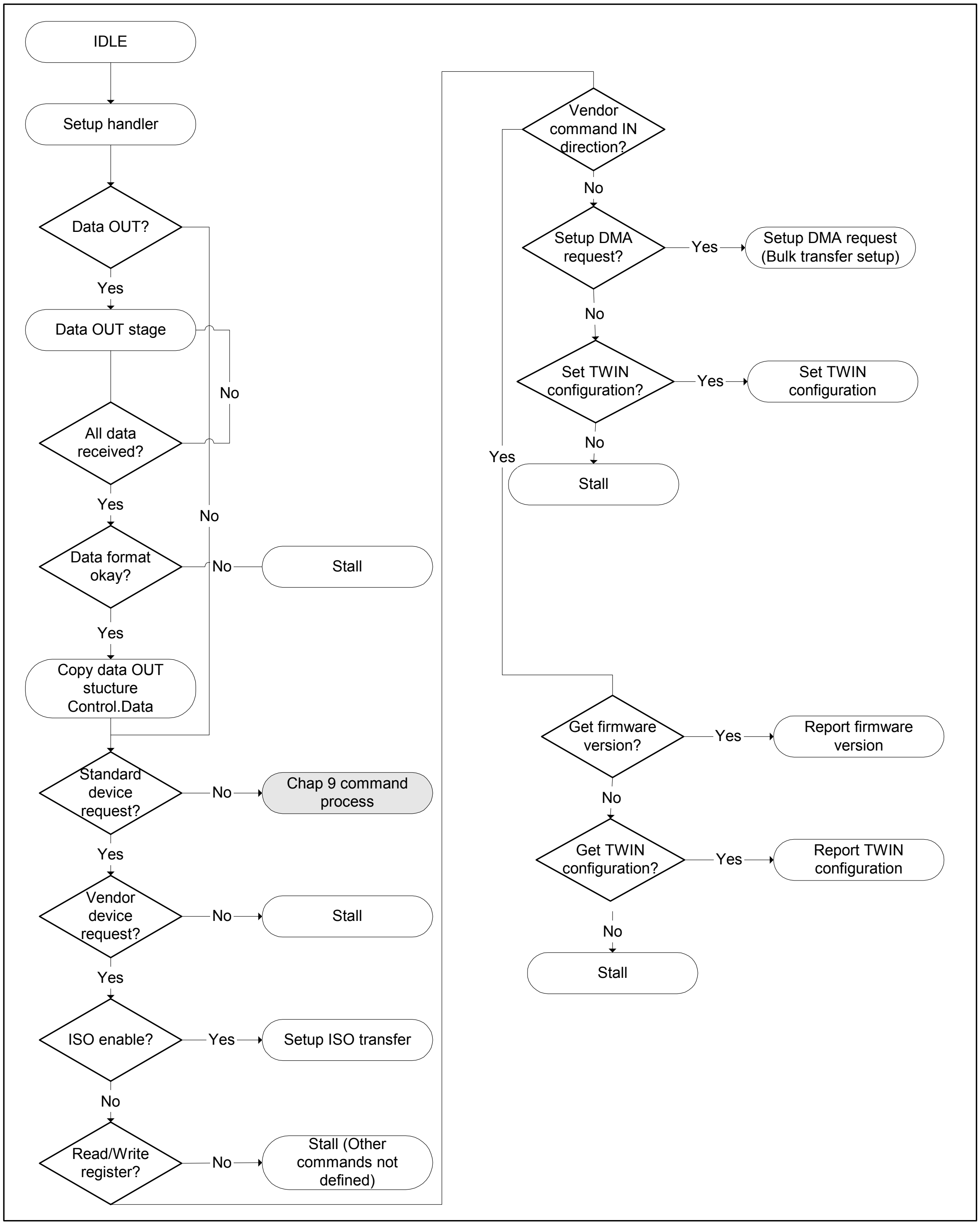

Figure 16-1: Vendor-Specific Command Process Flowchart

16.1. Get Firmware Version

When the test applet is started, it sends a vendor request for a device firmware version that is used to identify which eval kit is being used, and also to select the proper function to be tested. Figure 16-2 shows the flowchart of the Get firmware version command.

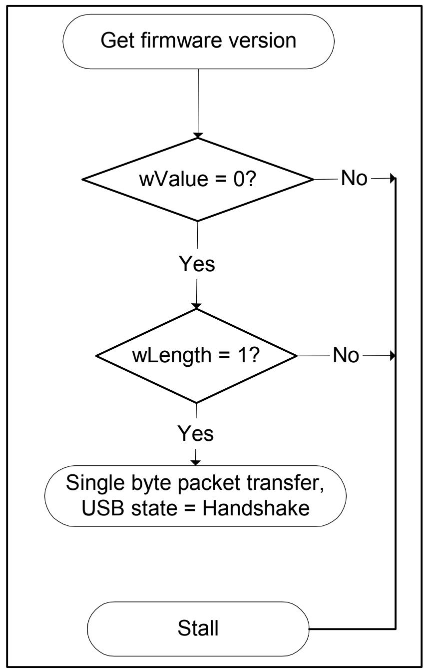

Figure 16-2: Get Firmware Version Command Flowchart

16.2. Get and Set TWIN Configuration

Before the Bulk data transfer, the test applet performs a sequence of configurations (similar to what a scanner does during setup). The complete data transfer is broken into several pages, each of which is up to 64 kbytes. The applet can also retrieve the configuration settings for verification.

The flowcharts of the Get TWIN Configuration and the Set TWIN Configuration are given in Figure 16-3 and Figure 16-4, respectively.

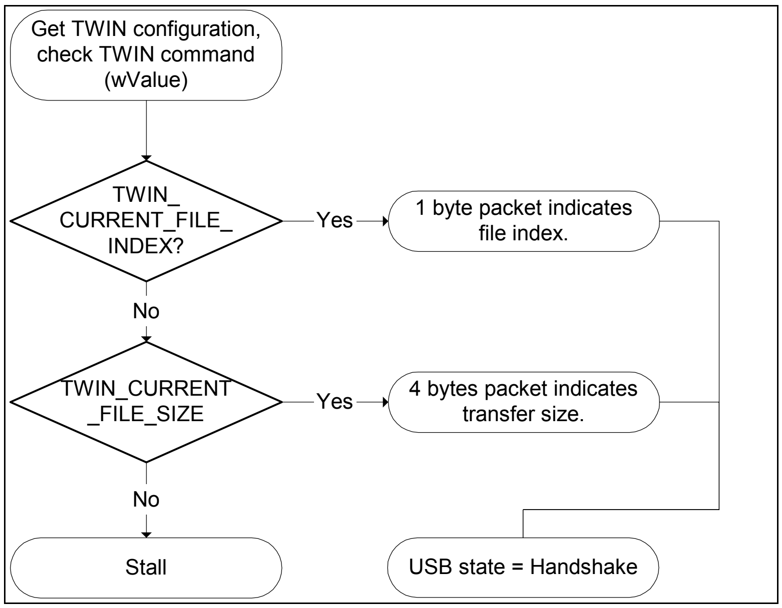

Figure 16-3: Get TWIN Configuration Flowchart

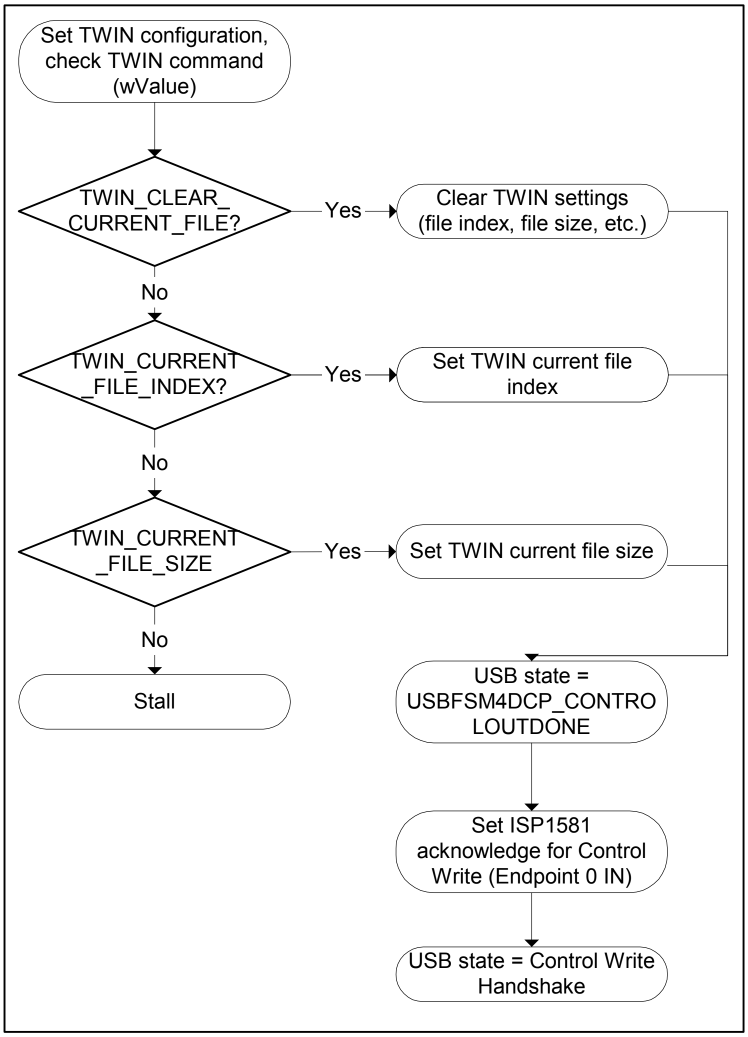

Figure 16-4: Set TWIN Configuration Flowchart

16.3. Setup DMA Request (Setup Bulk Transfer)

The Setup DMA request consists of the following information:

- Data size
- Direction of data transfer; either IN or OUT
- Page index; indicates the page sequence
- Data location; supports multiple page transfer, each has a file index.

Figure 16-5 shows the flowchart for the Setup DMA request.

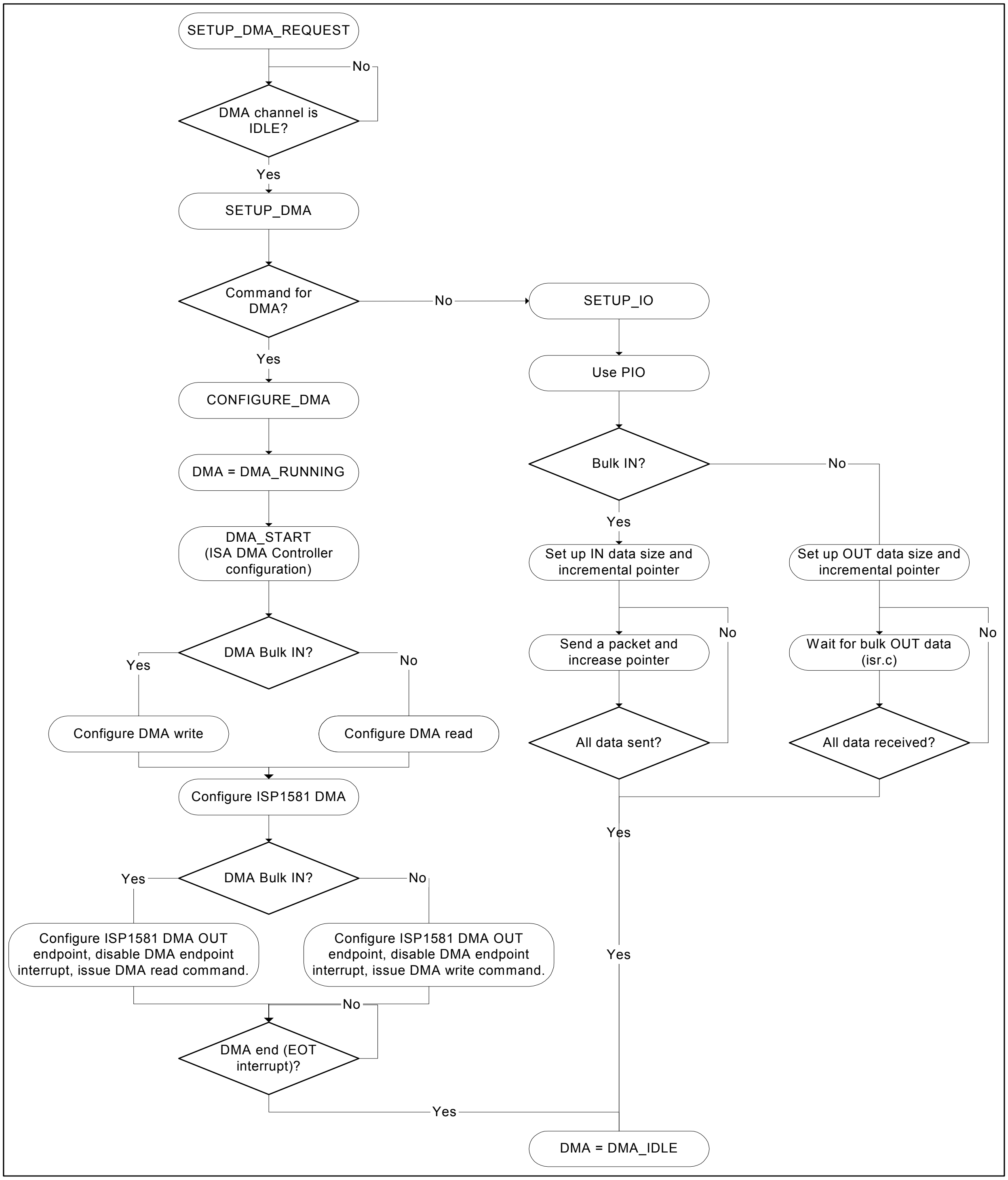

Figure 16-5: Setup DMA Request Flowchart

16.4. Setup ISO Transfer

The Setup ISO transfer details are given in the flowchart in Figure 16-6.

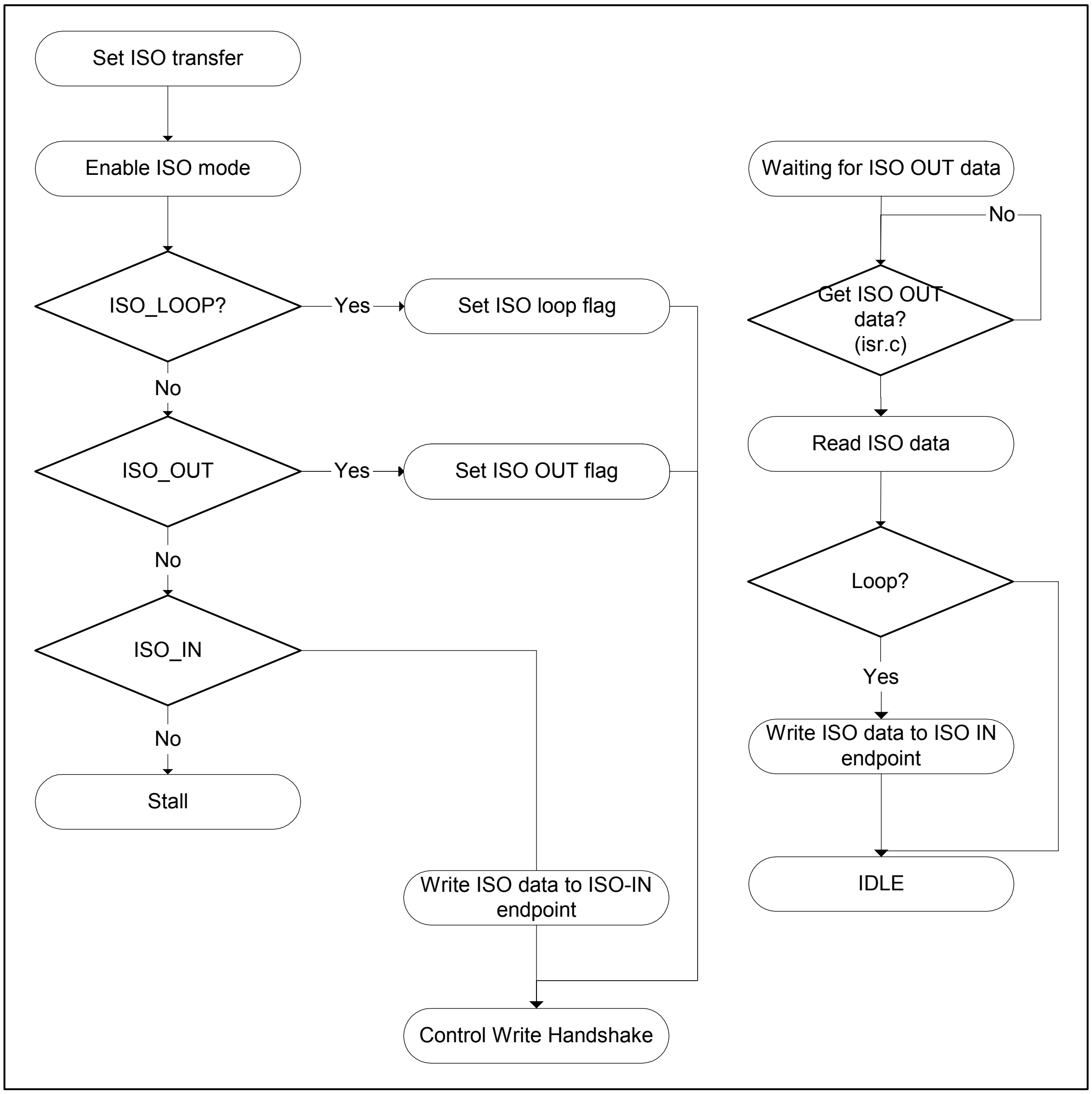

Figure 16-6: Setup ISO Transfer Flowchart

17. Application Details

Detailed information on the vendor-specified commands used for testing application is given in the following sections.

17.1. Get Firmware Version

The transfer illustrates the complete transaction of the Get firmware version command.

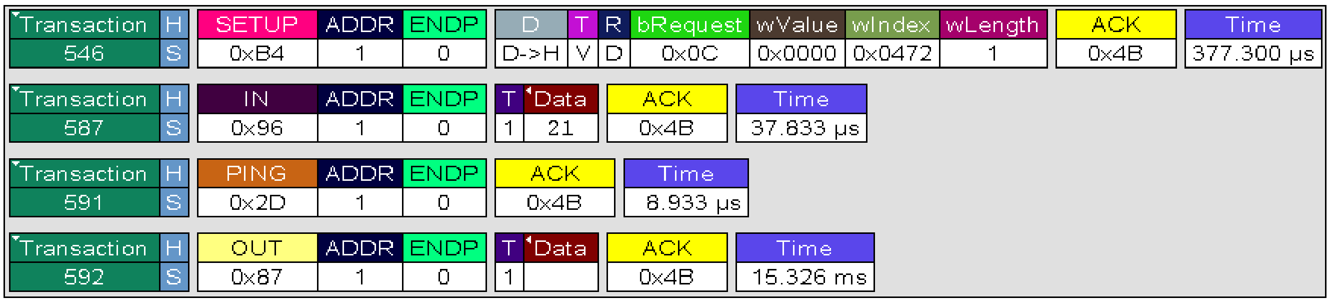

Figure 17-1: Get Firmware Version

The following information can be derived from Figure 17-1.

Request Index: 0472

Request Value: 0H

Requests firmware version of the device. The device reports any value.

17.2. Clear Current File

The transfer illustrates the complete transaction of the Clear Feature command.

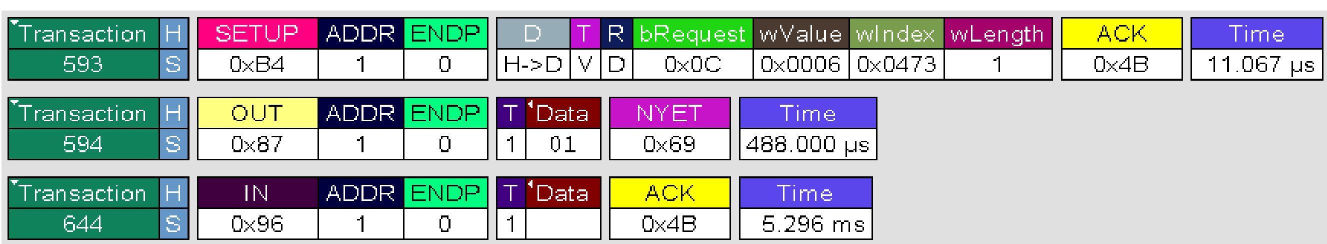

Figure 17-2: Clear Current File

Figure 17-2 provides the following information:

Request Index: 0473

Request Value: 0006H

This command requests the device to clear the stored image to free space for a new image.

17.3. Set File Index

The transfer illustrates the complete transaction of the Clear Feature command.

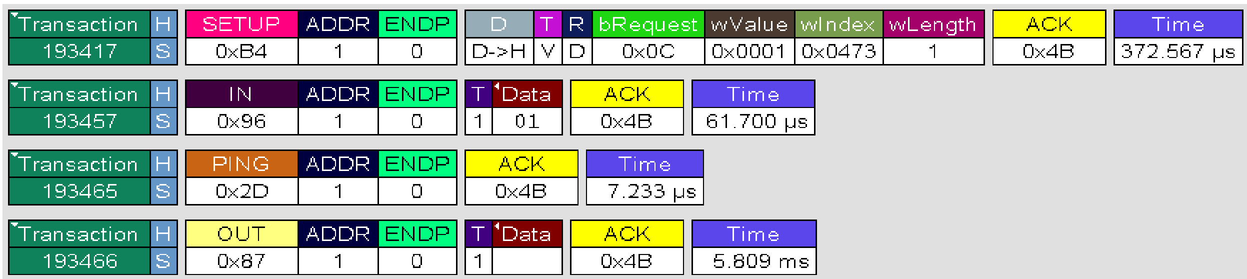

Figure 17-3: Set File Index

As can be seen in Figure 17-3:

Request Index: 0473

Request Value 0001H

Sets the device to prepare the file index of images to 1.

17.4. Bulk Transfer Setup

Get Image data command in the Philip Scan demo.

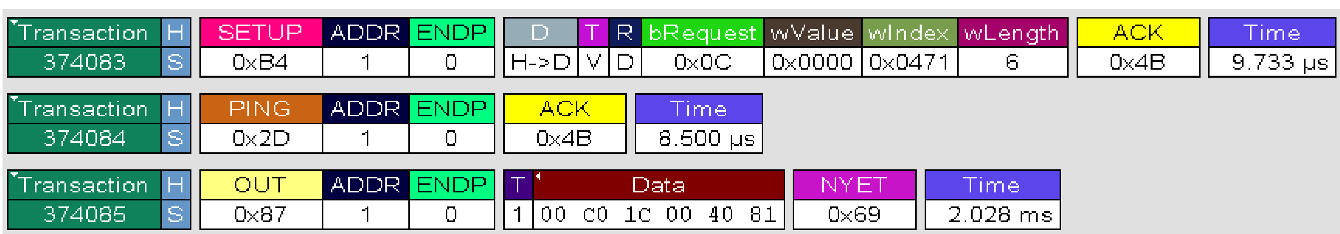

Figure 17-4: Bulk Transfer Setup

The following information can be derived from Figure 17-4.

Request Index: 0471

Request Value: 0H

The control write transfer to encapsulate data transfer.

The Data format in Figure 17-4 is explained in details in the following tables.

Table 17-1: Data Format

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
LSB	—	MSB	LSB	MSB	—
00	0c	1c	00	40	81
Address Offset of Image			Current Transfer Length		Operation Command (see Table 17-2)

Table 17-2: Operation Command

Bit 7	6	5	4	3	2	1	Bit 0
1—use DMA for transfer **0**—microprocessor operates on the buffer	Not used						**1**—scan image (Bulk data IN) **0**—store image data to device (Bulk data OUT)

18. References

- ***ISP1581 Hi-Speed Universal Serial Bus interface device*** data sheet
- ***ISP1581 PC Eval Kit User's Guide***
- ***ISP1581 Scanner Eval Kit User's Guide***
- ***Universal Serial Bus Specification Rev. 2.0.***