Freescale Semiconductor
Application Note

Document Number: AN3631
Rev. 0, 03/2008

Simplified Device Data Structures for the High-End ColdFire Family USB Modules

by: Melissa Hunter
Applications
Microcontroller Solutions Group

Some of the high-end ColdFire products (such as the MCF532x, MCF537x, MCF5253, and MCF5445x devices) contain an EHCI-compatible host or dual-role/OTG USB controller. The dual-role module can be used as a USB host, a USB device, or as an On-the-Go device. When the dual-role module is used as a device, it uses data structures to control USB data movement. These data structures are similar in some ways to the embedded host controller interface (EHCI) data structures that are used when operating in host mode.

The purpose of this application note is to discuss a simplified version of the device mode data structures used by the dual-role USB controller. This document also walks through a real-world example to show how the USB controller is programmed and to provide examples of how the data structures are used.

This document is intended as a guide for developing a simple driver for communicating with a USB host. The simplified version of the data structures discussed in this

Contents

freescale™ semiconductor

document do not support all of the capability of the USB controller. To simplify the data structures, isochronous transfers are not discussed. Transfer sizes are also restricted to smaller than 4 KB. When using the full data structures discussed in the device reference manual, both isochronous traffic and transfers larger than 4 KB are supported.

1 USB Device Overview

For device operations, the USB module uses two different types of data structures — endpoint queue heads (dQHs) and endpoint transfer descriptors (dTDs).

One dQH is defined for each endpoint and each direction. For example, there is one dQH used for endpoint 0 OUT transactions and a second dQH used for endpoint 0 IN transactions. These dQHs are stored in memory as an eight element array as shown in Figure 1.

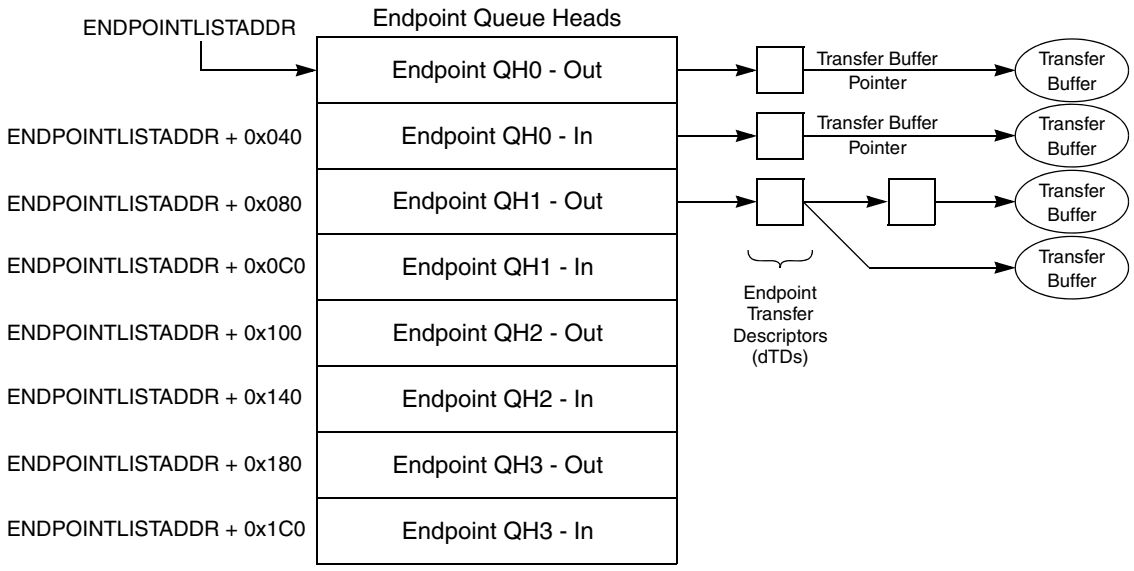

Figure 1. Endpoint Queue Head Diagram

1.1 Endpoint Queue Head (dQH)

There is one dQH used for each direction of each endpoint as shown in Figure 1. The dQH defines the max packet length along with other options for the endpoint. When a packet is transferred, the overlay area of the dQH acts as a staging area for the dTD so the device controller can access needed information with minimal latency.

Figure 2 shows a simplified dQH structure where the fields relating to isochronous transactions and large transfers (transfers that require more than one buffer pointer) have been removed or set to static values. This is a simplified version of the dQH structure specified in the device reference manual.

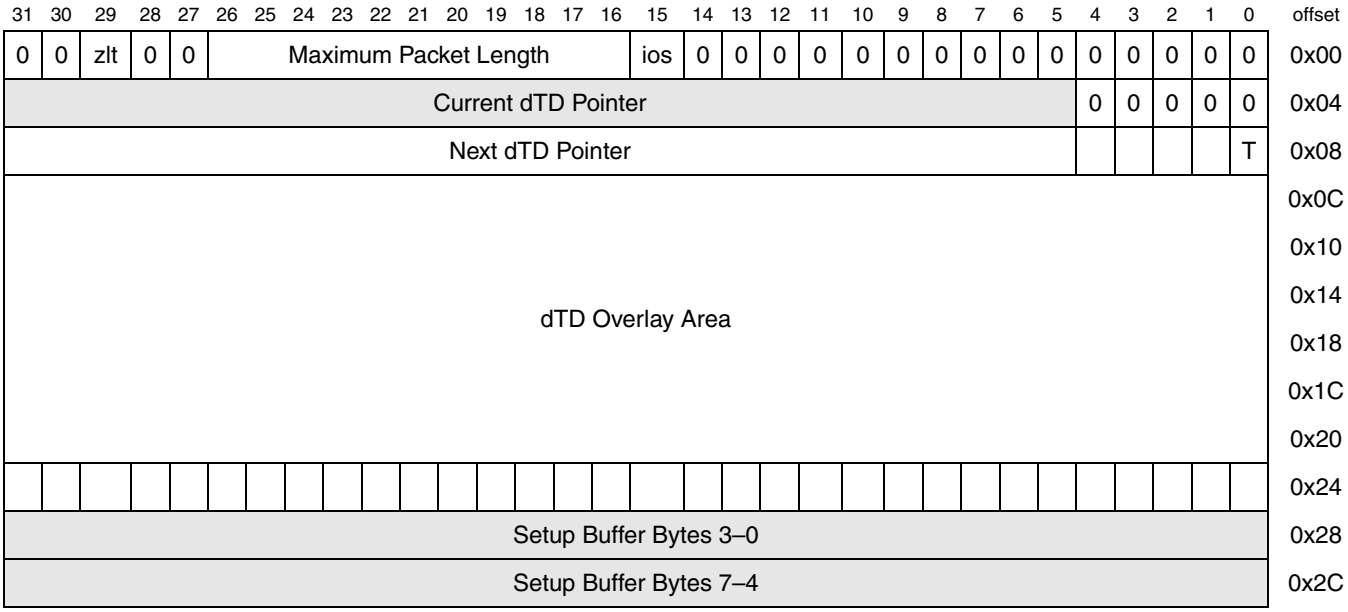

Figure 2. Endpoint Queue Head Layout (dQH)

1.1.1 Endpoint Capabilities/Characteristics (Offset = 0x0)

The first longword of the dQH specifies static information about the endpoint. In other words, this information does not change over the lifetime of the endpoint.

Table 1. Endpoint Capabilities/Characteristics

Bit	Name	Description
31–30	—	Write as 00.
29	zlt	Zero length termination select. Clearing this bit indicates that zero length packets will be used to terminate transfers that are a multiple of the max packet size as per the USB specification. For regular operation this bit must be cleared.
28–27	—	Write as 00.
26–16	Maximum Packet Length	Set to the maximum packet size in bytes of the associated endpoint. The maximum value this field may contain is 0x400 (1024).
15	ios	Interrupt on setup. If this bit is set, the device controller will issue an interrupt whenever a setup packet is received on this endpoint. This bit is only used for control endpoints.
14–0	—	Write as 0x000.

1.1.2 Current dTD Pointer (Offset = 0x4)

This longword is the address in memory of the dTD that is currently being processed. This field is written by the device controller when it reads in the dTD. Software does not need to initialize this longword when creating a new dQH.

1.1.3 dTD Overlay Area (Offset = 0x8–0x20)

The eight longwords in this area are a working copy of the dTD that is currently being processed or was last processed. When a transfer is in progress, the controller will write incremental status information to the dTD overlay area. When the transfer is complete, the results are written back to the original dTD location (the address pointed to by the current dTD pointer).

These values are initialized by the device controller when it copies in the current dTD, so software does not need to initialize these fields. The one exception is the next dTD pointer. This value should be initialized by software whenever a new chain of dTDs needs to be processed (the current linked list is complete or has been flushed). Software should initialize the next dTD pointer to the address of the first dTD to be processed for the endpoint (with the T bit cleared to indicate a valid pointer). After this is done the endpoint can be primed.

1.1.4 Setup Buffer (Offset = 0x28–0x2C)

The USB device controller manages setup packets differently than other packets. Whenever a setup packet is received, the incoming data is stored in a dedicated 8-byte setup buffer. The setup buffer is used only for OUT (host to device) control endpoints. Refer to the USB OTG controller chapter of the processor user's manual for more information on the procedure for reading the setup buffer.

1.2 Endpoint Transfer Descriptor (dTD)

A dTD is used to define an actual data movement for a single transfer. In particular, it provides the quantity of data to be sent or received and the location of the data. The dTDs are processed as a singly linked list. The next dTD pointer in the dQH should be initialized to point to the first dTD in the linked list, and then the endpoint can be primed to start processing of the first dTD. After the first dTD is processed, the controller will use the next dTD pointer in the first dTD to find the second dTD. The controller will continue traversing the linked list until a dTD with an invalid next dTD pointer is reached or the endpoint is flushed to abandon processing of the current dTD. dTDs must be aligned on 32-byte boundaries.

Figure 3 shows the simplified structure for a dTD. This is a simplified version of the dTD defined in the processor reference manual that can be used to transfer up to 4 KB of data.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Next dTD Pointer																											0	0	0	0	T	0x00
0	Total Bytes															ioc	0	0	0	0	0	0	0	Status								0x04
Buffer Pointer (Page 0)																																0x08
0000_0000_0000_0000_0000_0000_0000_0000																																0x0C
0000_0000_0000_0000_0000_0000_0000_0000																																0x10
0000_0000_0000_0000_0000_0000_0000_0000																																0x14
0000_0000_0000_0000_0000_0000_0000_0000																																0x18

Device controller read/write; all others read-only.

Figure 3. Endpoint Transfer Descriptor (dTD)

1.2.1 Next dTD Pointer (Offset = 0x0)

The first longword of a dTD is a pointer to the next valid dTD, if the T bit is zero. This pointer is used to create a singly linked list of dTDs.

Table 2. Next dTD Pointer

Bit	Name	Description
31–5	Next dTD Pointer	This field contains the physical memory address of the next dTD to be processed. The field corresponds to memory address signals[31:5], respectively.
4–1	—	Write as 0000.
0	T	Terminate. This bit indicates to the device controller that there are no more valid entries in the queue. 0 = Pointer is valid (points to a valid transfer element descriptor). 1 = Pointer is invalid.

1.2.2 dTD Token (Offset = 0x4)

The second longword of a dTD contains attributes for the transfer, including the number of bytes to read or write and the status of the transaction.

Table 3. dTD Token

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>31</td><td>—</td><td>Write as 0.</td></tr>
<tr><td>30–16</td><td>Total Bytes</td><td>Total Bytes. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction, but only if there is successful completion of the transaction.

The maximum value software may store in this field is 4K (0x1000). This is the maximum number of bytes a single page pointer can access. The device controller can accommodate larger transfers using multiple page pointers, but for the purposes of this application note the transfer size is limited to 4 KB, to help simplify the data structures.

If the value of this field is zero when the device controller fetches this transfer descriptor (and the active bit is set), the device controller executes a zero-length transaction and retires the transfer descriptor.

For OUT transfers the total bytes must be evenly divisible by the maximum packet length. For IN transfers it is not a requirement for total bytes to be an even multiple of the maximum packet length.</td></tr>
<tr><td>15</td><td>ioc</td><td>Interrupt on Complete. If this bit is set, it specifies that when this dTD is completed, the device controller should issue an interrupt.</td></tr>
<tr><td>14–8</td><td>—</td><td>Write as 0000000.</td></tr>
<tr><td>7–0</td><td>Status</td><td>This field is used by the device controller to communicate individual command execution states back to the device controller driver software. This field contains the status of the last transaction performed on the dTD. The bit encodings are:
<table>
<tr><th>Bit</th><th>Status Field Description</th></tr>
<tr><td>7</td><td>Active. Set by software to indicate that the dTD has been initialized and is ready to use. Enable the execution of transactions by the device controller.</td></tr>
<tr><td>6</td><td>Halted. Set by the device controller during status updates to indicate a serious error has occurred at the device/endpoint addressed by this dTD. Any time a transaction results in the halted bit being set, the active bit is also cleared.</td></tr>
<tr><td>5</td><td>Data Buffer Error. Set by the device controller during status update to indicate the device controller is unable to maintain the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (underrun).</td></tr>
<tr><td>4</td><td>Reserved.</td></tr>
<tr><td>3</td><td>Transaction Error. Set by the device controller during status update in case the device did not receive a valid response from the host (time-out, CRC, bad PID).</td></tr>
<tr><td>2–0</td><td>Reserved.</td></tr>
</table></td></tr>
</table>

1.2.3 dTD Buffer Page Pointer (Offset = 0x8)

The dTD buffer page pointer is used to specify the memory address of the data buffer for the transfer.

Table 4. dTD Buffer Pointer

Bit	Name	Description
31–0	Buffer Pointer	Buffer Pointer. Indicates the physical memory address for the data buffer to be used by the dTD. The device controller actually uses the first part of the address (bits 31–12) as a pointer to a 4 KB page, and the lower part of the address (bits 11–0) as an index into the page. The host controller will increment the index internally, but will not increment the page address. This is what determines the 4 KB transfer size limitation used for this application note. This also means that the data buffer cannot span the 4 KB page boundary. For applications where most of the transfers are small, then runtime buffer alignment can be avoided entirely by careful placement of the heap. If the heap space (used to allocate memory for dQHs, dTDs, and data buffers) starts at the beginning of a 4 KB page and the application will not require more than 4 KB worth of data structures and buffers at a time, then the data buffer alignment is not a concern. If more than 4 KB is needed for data structures and data buffers, then there are a couple of ways to avoid a data buffer crossing a 4 KB page boundary. • One option is to force all of the data buffers to a 4 KB page alignment; however, this does not make for a very efficient use of memory, unless most transfers are near to 4 KB. • Another option is to align each data buffer to its own size. For example, a 16 byte transfer would be aligned to a 16-byte line address. This will allow for much more efficient use of memory for most applications. This does add a small amount of code overhead to manage the alignment. This field contains the physical memory address of the next dTD to be processed. The field corresponds to memory address signals[31:5], respectively.

2 USB Device Example

The following subsections will walk through a working USB device software example. We will discuss the “m5329evb_usb_dev_mouse_test” demo code in the MCF532XSC.zip file available on the ColdFire website (http://www.freescale.com/coldfire). This example code is designed to configure the USB module for device mode, enumerate as USB mouse, and then send offset information to a PC host that will move the cursor in a rectangular pattern.

2.1 USB Device Initialization

The initialization below is used in the sample code. It is similar to the device initialization sequence in the reference manual, but some chip-level initialization has been added and optional steps have been omitted.

1. Poll for a valid VBUS.
2. Initialize the USB clock divider in the CCM. (This step is needed because the PLL speed was changed during the system initialization for the EVB.)
3. Set USBMODE register to enable device mode operation and big endian mode, and to disable setup lockouts (0x0000_000E).
4. Allocate memory to use for the device endpoint list, and initialize the EPLISTADDR to point to this memory.

5. Initialize EPCR0 to enable receive and transmit as a control endpoint (0x00800080). This is actually the default value for the register, so this step can be skipped.
6. Create two dQHs. One will be used for EP0-OUT, and one for EP0-IN.
7. Put the controller into a run state by setting USBCMD[RS].
8. Set UOCSR[BVLD] to indicate that a valid VBUS signal has been detected and the controller is ready to respond to enumeration by the host. This step is not needed for the MCF5253.

2.1.1 EP0-OUT dQH

Figure 4 shows the simplified dQH layout. Immediately below each line are the actual values used in the example software to initialize a QH for endpoint 0 OUT transactions. Only the first three longwords are shown, because the remaining fields do not need to be initialized by software when creating the dQH. The example software will clear the unshown fields to make reading dQH values easier when debugging.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
0	0	zlt	0	0	Maximum Packet Length											ios	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x00
0x0040_0000																																
Current dTD Pointer																											0	0	0	0	0	0x04
0x0000_0000																																
Next dTD Pointer																															T	0x08
0x0000_0001																																

Figure 4. Endpoint 0-OUT Queue Head Values

2.1.1.1 EP-OUT dQH Endpoint Capabilities/Characteristics (Offset = 0x0)

Table 5. EP0-OUT Example Endpoint Capabilities/Characteristics

Bit	Name	Description
31–30	—	Write as 00.
29	zlt	This bit is cleared to enable zero length packet termination.
28–27	—	Write as 00.
26–16	Maximum Packet Length	The maximum packet length chosen for the example software is 64 bytes (0x40).
15	ios	The example software will use polling to determine when a setup packet is received, so interrupt on setup is disabled by clearing this bit.
14–0	—	Write as 0x000.

2.1.1.2 EP0-OUT dQH Current dTD Pointer (Offset = 0x4)

Table 6. EP0-OUT Example Current dTD Pointer

Bit	Name	Description
31–5	Current dTD Pointer	The current dTD pointer field will be written by the USB controller as it traverses a linked list of dTDs. Software should not write this field, so it is cleared in this example.
4–0	—	Write as 00000.

2.1.1.3 EP0-OUT dQH Next dTD Pointer (Offset = 0x8)

Table 7. EP0-OUT Example Next dTD Pointer

Bit	Name	Description
31–5	Next dTD Pointer	At this point there are no dTDs, so the next dTD pointer is cleared for now. It will be initialized by software after dTDs are ready to be processed.
4–1	—	Write as 0000.
0	T	The T bit is set, indicating that the next dTD pointer value is not valid.

2.1.2 EP0-IN dQH

The dQH used for EP0-IN uses the same values as the dQH for EP0-OUT. The maximum packet length is set to 0x40 and the other settings are also the same.

2.2 Device Enumeration

After a connection to the host is established, the host will begin the process of enumerating the device. A request for the first eight bytes of the device descriptor should be the first USB packet that the host sends to the device.

The following subsections discuss in detail how the example code receives and responds to this first device descriptor request from the host. The process for handling the rest of the enumeration process, and then transmitting data on other endpoints after enumeration, is somewhat similar to that used for the initial device descriptor request. For more examples, please refer to the sample code.

2.2.1 Read the Setup Packet from the Host

The USB device controller manages setup packets differently than other USB packets. Instead of using a dTD that points to a buffer where the payload of the setup packet is stored, the packet is stored in a buffer at the end of the dQH data structure. This way the device controller does not need an active endpoint transfer descriptor (dTD) to accept a setup packet at any time.

So after a connection to the host is established, one of the first things the sample code does is poll the EPSETUPSR, waiting for indication that a setup packet has been received. After reception of a setup

packet is detected, a specific procedure must be used to read the packet and ensure that the data is not corrupted. This sequence is documented in the reference manual, but is also included below.

1. Write a 1 to clear the corresponding bit in the EPSETUPSR.
2. Set the setup tripwire bit (USBCMD[SUTW]).
3. Read the contents of the setup buffer in the dQH and copy into a local buffer.
4. Wait for the USBCMD[SUTW] bit to set.
5. Clear the USBCMD[SUTW] bit.
6. Wait for the corresponding EPSETUPSR bit to clear.

After the software has read the setup packet, it checks the value to ensure that it is a GET DESCRIPTOR request for the first eight bytes of the device descriptor, as expected.

2.2.2 Send the Device Descriptor to the Host

Now that the GET DESCRIPTOR request for the first eight bytes of the device descriptor has been received, the software needs to prepare some dTDs to respond to the host's request. To respond to the request the device needs to send the first eight bytes of the device descriptor, then receive a zero byte packet from the host. This means that two different dTDs are needed — one IN/transmit dTD and one OUT/receive dTD.

2.2.2.1 IN dTD

The IN dTD is used to transmit eight bytes of the device descriptor to the host. Figure 5 shows the simplified dTD layout along with the actual values used in the example software to initialize a dTD to transmit the device descriptor to the host.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Next dTD Pointer																											0	0	0	0	T	0x00
0xDEAD_0001																																
0	Total Bytes															ioc	0	0	0	0	0	0	0	Status								0x04
0x0008_0080																																
Buffer Pointer (Page 0)																																0x08
0x4001_BA08																																

Figure 5. Example IN Endpoint Transfer Descriptor (dTD)

2.2.2.1.1 IN dTD Next dTD Pointer (Offset = 0x0)

Table 8. IN dTD Example of Next dTD Pointer

Bit	Name	Description
31–5	Next dTD Pointer	This is the only IN dTD needed to complete the full GET DESCRIPTOR transaction, so a dummy value is used. This value was selected to make it easy to identify the last dTD in a linked list.
4–1	—	Write as 0000.
0	T	The T bit is set, indicating that the next dTD pointer value is not valid.

2.2.2.1.2 IN dTD Token (Offset = 0x4)

Table 9. IN dTD Example of dTD Token

Bit	Name	Description
31	—	Write as 0.
30–16	Total Bytes	The host has requested eight bytes of the device descriptor, so this field is set to 0x8.
15	ioc	This bit is cleared to disable a request for interrupt. Instead, the example software will request an interrupt at the end of the OUT packet, to indicate the completion of the full GET DESCRIPTOR transaction.
14–8	—	Write as 0000000.
7–0	Status	0x80 marks the dTD as active and ready for the device controller hardware to process.

2.2.2.1.3 IN dTD Buffer Page Pointer (Offset = 0x8)

Table 10. IN dTD Example of dTD Buffer Pointer

Bit	Name	Description
31–0	Buffer Pointer	The device descriptor is in a buffer at location 0x4001_BA08. This is the data that will be sent to the host during the next IN request.

2.2.2.2 OUT dTD

The OUT dTD is used to receive the host's response after transmitting the device descriptor. It is expected that the host will respond by sending a zero length packet. Figure 6 shows the simplified dTD layout, along with the actual values used in the example software to initialize a dTD to receive the host response.

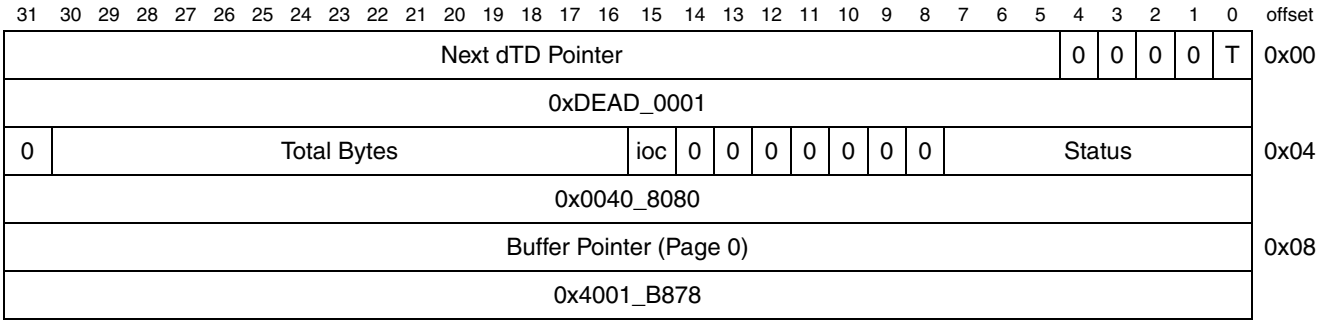

Figure 6. Example OUT Endpoint Transfer Descriptor (dTD)

2.2.2.2.1 OUT dTD Next dTD Pointer (Offset = 0x0)

Table 11. OUT dTD Example of Next dTD Pointer

Bit	Name	Description
31–5	Next dTD Pointer	This is the only OUT dTD needed to complete the full GET DESCRIPTOR transaction, so a dummy value is used. This value was selected to make it easy to identify the last dTD in a linked list.
4–1	—	Write as 0000.
0	T	The T bit is set, indicating that the next dTD pointer value is not valid.

2.2.2.2.2 OUT dTD Token (Offset = 0x4)

Table 12. OUT dTD Example of dTD Token

Bit	Name	Description
31	—	Write as 0.
30–16	Total Bytes	For OUT endpoints, the total bytes field must always be set to a multiple of the maximum packet length for the endpoint. Because a zero length packet is expected, the dTD needs to be able to accommodate at least one packet from the host, so the total bytes field is set equal to the maximum packet length (0x40).
15	ioc	This bit is set to enable a request for interrupt. The USBSTS[UI] bit will be set when the OUT completes, to allow software to detect when the full GET DESCRIPTOR transaction has finished.
14–8	—	Write as 0000000.
7–0	Status	0x80 marks the dTD as active and ready for the device controller hardware to process.

2.2.2.2.3 OUT dTD Buffer Page Pointer (Offset = 0x8)

Table 13. OUT dTD Example of dTD Buffer Pointer

Bit	Name	Description
31–0	Buffer Pointer	0x4001_B878 is the address of the buffer allocated to hold the data transmitted by the host.

2.2.2.3 Using the dTDs

After the dTDs have been created and initialized, there are two important steps needed to allow the device controller to begin processing them. First, the next dTD pointer field of the corresponding dQH must be written with the address of the appropriate dTD. The EP-IN dQH next dTD pointer is written with the address of the IN dTD, and the EP-OUT dQH next dTD pointer is written with the address of the OUT dTD.

The endpoints also need to be primed. The IN endpoint is primed by writing a 1 to the EPPRIME[PETB] field, and the OUT endpoint is primed by writing a 1 to the EPPRIME[PERB] field. When the endpoint is primed, the device controller will fetch the dTD pointed to by the next dTD field, and copy the contents into the dTD overlay area of the dQH. For transmit endpoints, priming will also cause the device controller to fetch the first part of the transmit buffer.

3 Additional Information

Here are some additional resources that can be used to find more information on the MCF532x processor and USB. The reference manuals are available at www.freescale.com. The USB specification is available at www.usb.org.

- MCF5329 Reference Manual
- MCF5373 Reference Manual
- MCF54455 Reference Manual
- MCF5253 Reference Manual
- Universal Serial Bus Specification

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3631
Rev. 0
03/2008

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

freescale™
semiconductor