

7 DMA

DMA in the ETRAX 100LX provides a high data transfer rate to and from the internal peripheral interfaces, or from one location to another location in the external memory. It consists of ten DMA channels with five in each direction. The ten DMA channels are served by a DMA controller, which takes care of the data flow between the channels and the external memory. In addition, there are two external DMA channels (See 7.8 *External DMA Channels*).

7.1 DMA Operation

7.1.1 Overview of the ETRAX 100LX DMA Architecture

A simplified architecture overview of DMA in the ETRAX 100LX, as well as a schematic flow of data is shown in figure 7-1 below.

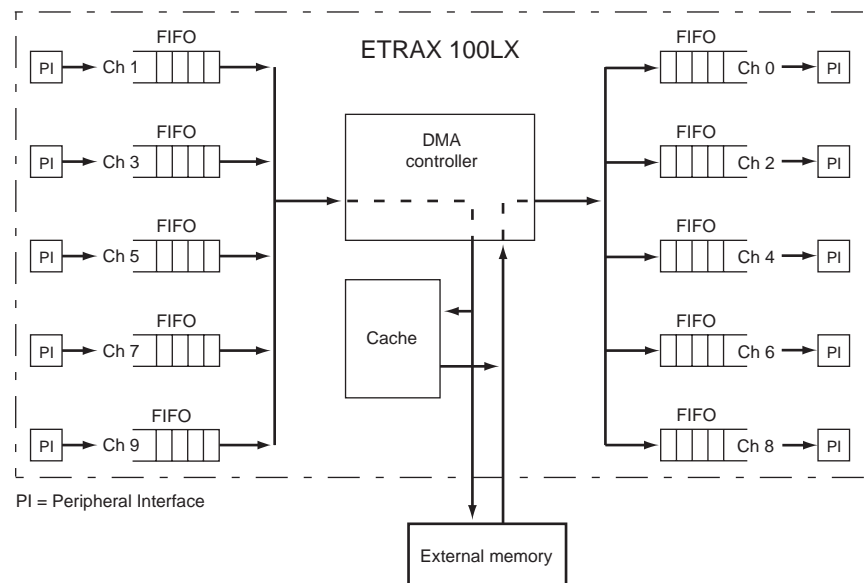


Figure 7-1 DMA Internal Overview

There are five input channels (1, 3, 5, 7 and 9), and five corresponding output channels (0, 2, 4, 6 and 8). All input and output channels have a FIFO buffer.

7.1.2 Data Transfer

There are three cases of data transference from:

- 1 A peripheral interface to external memory
- 2 External memory to a peripheral interface
- 3 One memory location to another memory location

Apart from these three cases of data transference, ETRAX 100LX DMA has two external DMA channels. See section *7.8 External DMA Channels*.

Data Transfer from a Peripheral Interface to External Memory

When data is to be transferred from one peripheral interface, it is first stored in a 64 byte FIFO buffer. When this buffer fills up to half its size (i.e. 32 bytes), the DMA controller is notified, and the data in the FIFO is transferred to external memory. If one of these memory addresses is present in the internal cache memory, the data is also stored in the internal cache memory. The transfer of data from a FIFO to the memory is done in bursts, and the length of these bursts is either 16 or 32 bytes as chosen by the software programmer in the register R_BUS_CONFIG.

Data Transfer from External Memory to a Peripheral Interface

When the DMA controller receives notice that a FIFO buffer is becoming half empty (i.e. less than or equal to 32 bytes), it begins a transfer of data from the external memory. When data is transferred from the external memory, it is read in bursts of 16 or 32 bytes as chosen by the software programmer in R_BUS_CONFIG. If the data in one of these memory addresses is present in the internal cache memory, the data is read from the internal cache memory.

Data Transfer from One Memory Location to Another Memory Location

In order to increase the performance of DMA for this type of data transfer, there are two channels specifically designed for this task: channels 6 and 7. The FIFO buffers for channels 6 and 7 can be set to connect directly to each other in the register R_GEN_CONFIG.

As with previous transfers, the data is read from and written to external memory when the FIFOs are either half empty or half full. If one of the memory addresses, where the data is located, is present in the internal cache memory, the data is read from or written to the internal cache memory. The data transfer to and from the FIFOs is done in bursts of 16 or 32 bytes as chosen by the software programmer in R_BUS_CONFIG.

7.2 The DMA Channels

There are only a limited number of combinations for the ten DMA channels which can be used for interconnection between the internal peripheral interfaces. The reason for this limitation is that some of the internal peripheral interfaces are multiplexed on the same package pins. Figures 7-2 and 7-3 below show how each peripheral interface is multiplexed on the DMA channels.

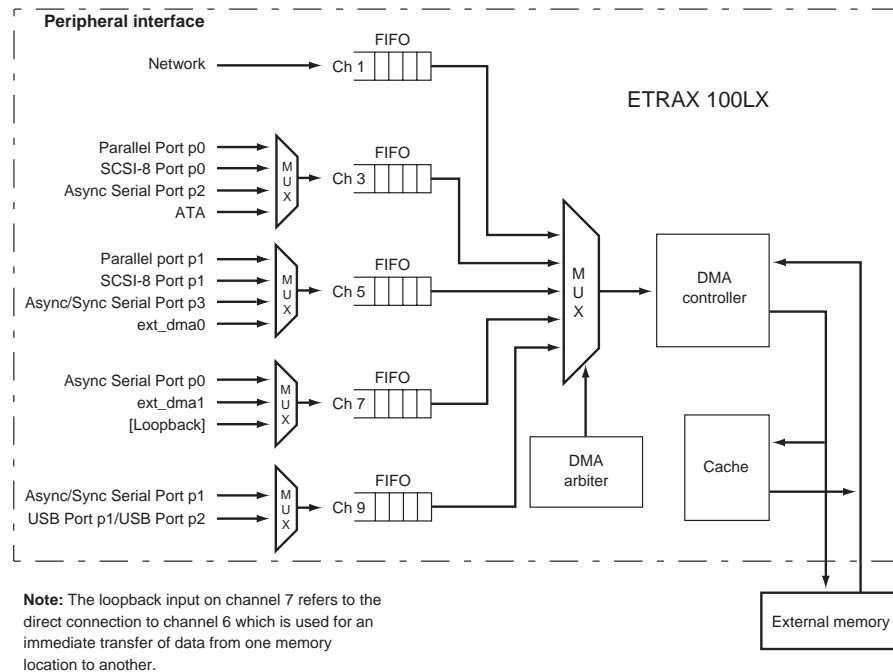


Figure 7-2 DMA Input Channels

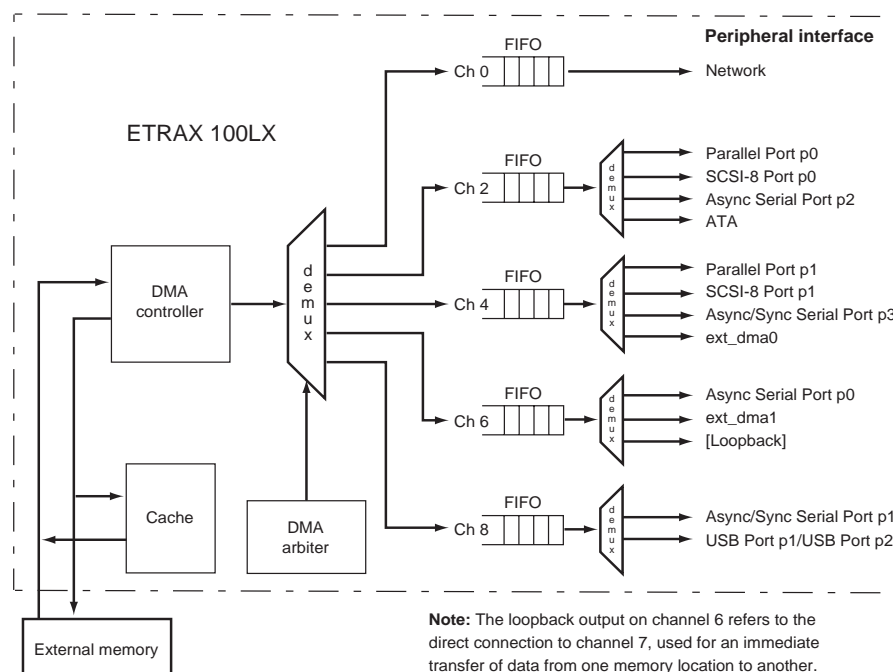


Figure 7-3 DMA Output Channels

The choice of which interface to be used is defined in the register R_GEN_CONFIG. The channels can be configured as shown in table 7-1 below:

DMA channel	I/O system(s) available				Direction	FIFO buffer (bytes)	Priority
0	Network				out	64	Highest
1	Network				in	64	
2	Parallel Port p0	SCSI-8 Port p0	Async serial Port p2	ATA	out	64	
3	Parallel Port p0	SCSI-8 Port p0	Async serial Port p2	ATA	in	64	
4	Parallel Port p1	SCSI-8 Port p1	Async/Sync serial Port p3	ext_dma0 (note 1)	out	64	
5	Parallel Port p1	SCSI-8 Port p1	Async/Sync serial Port p3	ext_dma0 (note 1)	in	64	
6	Async Serial Port p0	ext_dma1 (note 1)	Memory transfer (note 2)		out	64	
7	Async Serial Port p0	ext_dma1 (note 1)	Memory transfer (note 2)		in	64	
8	Async/Sync Serial Port p1	USB Port p1 & p2 (note 3)			out	64	
9	Async/Sync serial Port p1	USB Port p1 & p2 (note 3)			in	64	Lowest

Table 7-1 DMA Channel Configuration

- Note 1:** ext_dma0 and ext_dma1 are external DMA channels to be used between the external memory and an external I/O device, see 7.8 External DMA Channels.
- Note 2:** Memory-to-memory transfer. Channels 6 to channel 7 can be set for immediate connection, which provides an efficient way of transferring data from one memory location to another memory location.
- Note 3:** When Channels 8 and 9 are configured for USB, their priority becomes higher than the priority for channel 2 but lower than channel 1 (e.g. Highest: channel 0, 1, 8, 9, 2, 3...Lowest). See 7.4.2 DMA Linked Lists for USB

All 10 DMA channels (0 - 9) have a 64 byte FIFO buffer to allow efficient handling of burst mode DRAM access, and software can read the number of bytes in each FIFO buffer. For input channels, software can also flush the FIFO contents to the memory data buffer by forcing an **eop** (see table 7-2) in the data stream flowing into the FIFO by using the register R_SET_EOP.

7.3 DMA Registers, Linked Lists, and Descriptor Format

7.3.1 DMA Registers

There are a set of DMA registers, one set for each channel, which the DMA controller uses to handle the buffers and to store information about the buffers in the external memory. These registers and a little about of their functions are shown in figure 7-4 below:

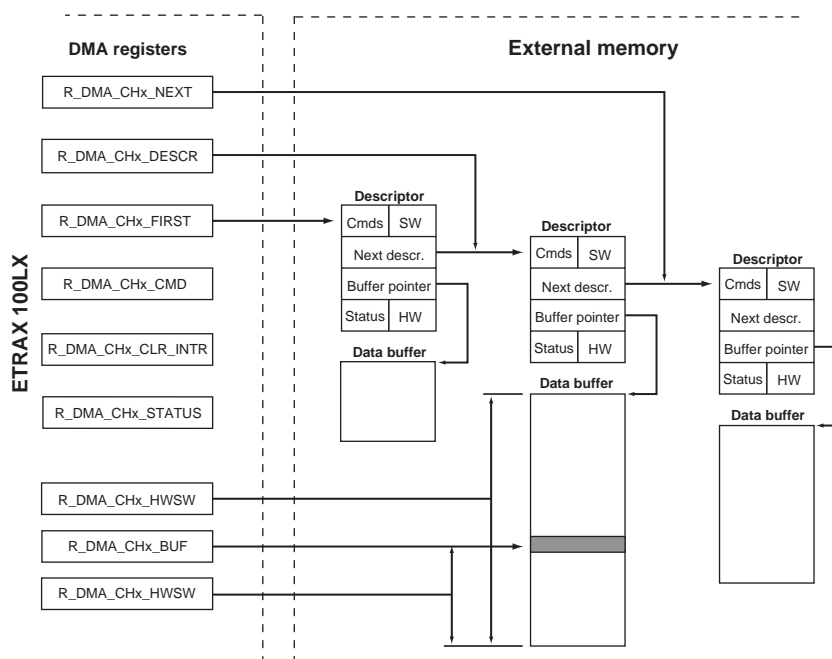


Figure 7-4 DMA Registers and the Structure of the Linked List

The figure above shows the DMA registers and a simplified linked list in the external memory. During normal operation, the only registers accessed by software are:

- R_DMA_CHx_FIRST
- R_DMA_CHx_CLR_INTR
- R_DMA_CHx_CMD

The R_DMA_CHx_FIRST register is used to locate the DMA list.

R_DMA_CHx_FIRST points to the first descriptor in the linked list as shown in figure 7-4 above. The R_DMA_CHx_DESCR register contains the current descriptor's address, and the R_DMA_CHx_NEXT contains the address to the next descriptor.

R_DMA_CHx_CLR_INTR is used to clear interrupts. For more details see 7.6 *DMA Transfer/Setup Examples*.

R_DMA_CHx_CMD is the command register to control DMA operation, and is used to reset and start DMA transfers. When a command is completed, the DMA channel sets R_DMA_CHx_CMD to **hold** (0) and stops.

The R_DMA_CHx_CMD register has five commands:

- Hold
- Start
- Restart
- Continue
- Reset

The **hold** command, which is completed immediately, holds the DMA channel in its current state. The **hold** command will fail, however, if the DMA channel has completed the previous command before the **hold** command is given. An unsuccessful **hold** command is indicated by the fact that the DMA channel reaches end-of-list so that R_DMA_CH0_FIRST is zero. A failed **hold** command is also indicated if the DMA channel receives stop-from-io, and the **stop** bit is set in the descriptor at R_DMA_CH0_DESCR.

When the **start** command is given, the DMA channel starts processing the list at R_DMA_CH0_FIRST. This command completes at end-of-list (1), stop-from-io (2), or a **reset** command (3). If the DMA channel is already running this command is ignored.

The **restart** command restarts the DMA channel after end-of-list has been reached. The DMA channel re-reads the descriptor at R_DMA_CH0_DESCR, and if the eol-bit is no longer set, it immediately follows the next link in the re-read descriptor ignoring the **wait**, **intr**, and **eop** bits. This command is completed at end-of-list (1), stop-from-io (2), or a **reset** command (3). If the DMA channel is already running this command is ignored (See note 4).

A **continue** command tells the DMA channel to continue after a successful **hold** command. If the **hold** command was unsuccessful, the continue command will be interpreted as a **restart** command. The **continue** command completes when the command held by the **hold** command has completed. (See note 4)

The **reset** command resets the DMA channel and its FIFOs. When the channel has won arbitration, the **reset** command takes 100 ns to complete.

Note 4: The **restart** and **continue** commands have the same value (3).

Three registers manage the actual storage of data in the buffers:

- The R_DMA_CHx_BUF register: pointer to the next (byte) position in the data buffer
- The R_DMA_CHx_HWSW register (**sw** field): gives the total length (in bytes) of the data buffer
- The R_DMA_CHx_HWSW register (**hw** field): gives the number of bytes left in the data buffer.

7.3.2 DMA Linked Lists

ETRAX 100LX DMA stores data in the external memory, in buffers linked together with the use of a list descriptor. Each list descriptor contains a number of data fields, which tell the DMA controller where to find the next descriptor in the list.

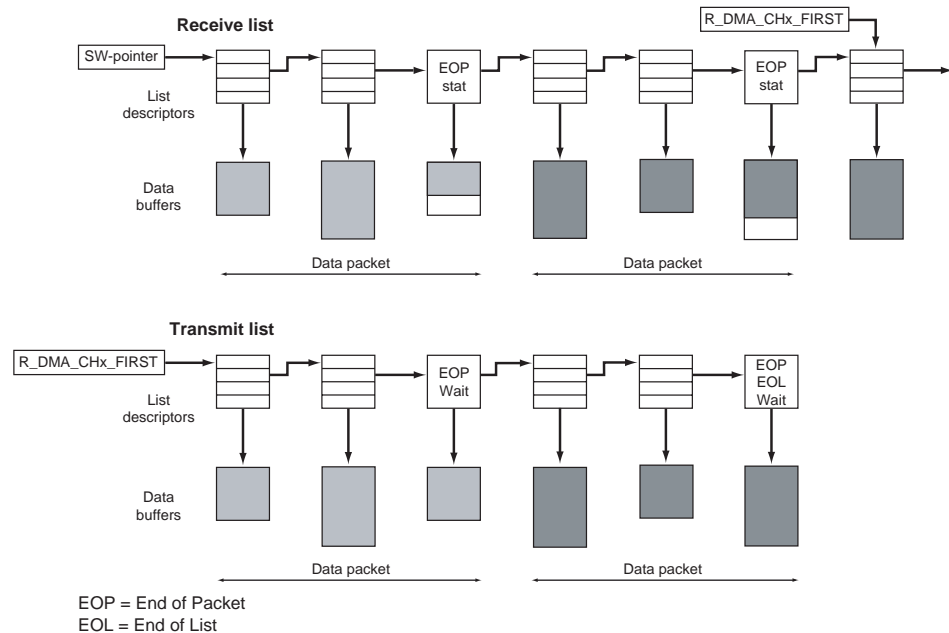


Figure 7-5 ETRAX 100LX Buffer Structure

The list descriptors also contain information about the location and size of the data buffer as well as other status information. This organization of data storage in memory has the advantage of having efficient memory management, and a very flexible structure.

DMA descriptors and data buffers do not have any alignment restrictions, except for the USB EP descriptor which must be 32-bit aligned (See section 7.4.3). However, performance improves if data and DMA descriptors are 32-bit and cacheline aligned.

For a more detailed overview of the list descriptor see *7.3.3 DMA Descriptor Format*.

7.3.3 DMA Descriptor Format

The construction of linked lists is done by defining DMA descriptors. The DMA descriptor consists of four 32-bit fields. The contents of a DMA descriptor is shown in figure 7-6 below:

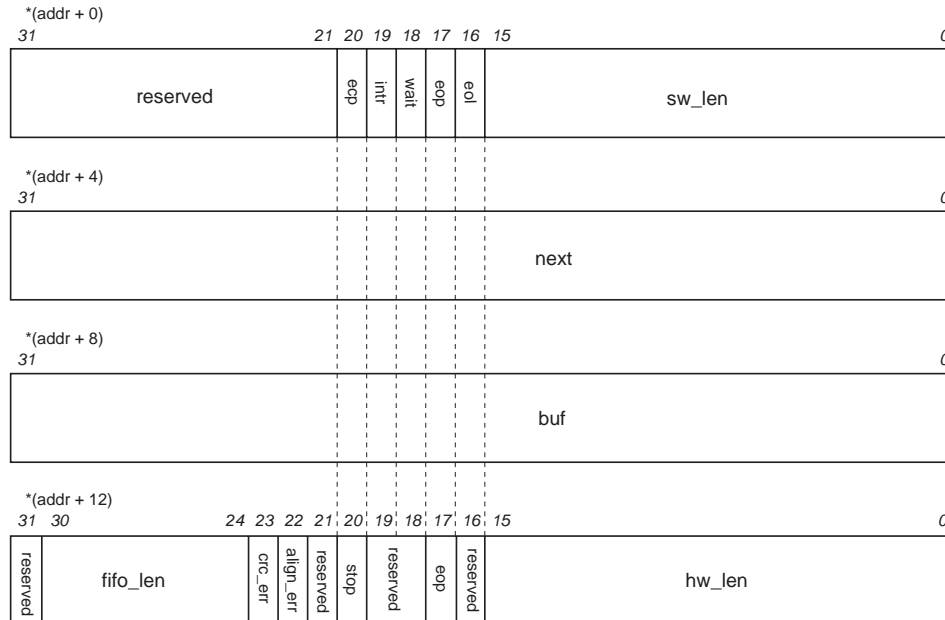


Figure 7-6 The DMA Descriptor Format

The first 32-bit field of the descriptor gives the length of the data buffer **sw_len** and a number of commands. The second 32-bit field gives the address to the next descriptor in the linked list, and the third 32-bit field gives the address to the data buffer **buf**. The last 32-bit field contains status information written by the DMA controller.

Table 7-2 describes the contents of the DMA descriptor in more detail:

Bit	Name	Explanation
(addr + 0):		
31-21	reserved	(note 6).
20	ecp_cmd	ECP command used by channel 2 and 4 when connected to a parallel port in ECP mode.
	tx_err	Force transmission error used by channel 0.
19	intr	Generate a descriptor interrupt when advancing to next descriptor.
18	wait	Wait until FIFO is empty before advancing to next descriptor, and also delay descriptor and end-of-packet interrupts until FIFO is empty (output channels only).
17	eop	Last descriptor in a packet (output channels only).
16	eol	Last descriptor in the list.
15-0	sw_len	Length in bytes of data buffer. (If all bits are 0, the length is 2^{16})
(addr + 4):		
31-0	next	Pointer to next descriptor in list (no alignment restrictions). If eol == 1 next is not used.
(addr + 8):		
31-0	buf	Pointer to first byte in data buffer (no alignment restrictions).
(addr + 12): (note 5)		
31	reserved	(note 6)
30-24	fifo_len	If stop == 1; Number of bytes in FIFO (output channels only).
23	crc_err	If eop == 1; Received packet has CRC error, used by channel 1.
22	align_err	If eop == 1; Received packet has alignment error, used by channel 1.
21	reserved	(note 6).
20	stop	Output channel was stopped by I/O interface. Bit 20 and bit 17 are mutually exclusive.
19-18	reserved	(note 6)
17	eop	Last descriptor in a received packet. Bit 20 and bit 17 are mutually exclusive.
16	reserved	(note 6).
15-0	hw_len	If eop == 1; Number of bytes written to the data buffer. (If all bits are 0, the length is 2^{16})
		If stop == 1; Number of bytes read from the data buffer. (If all bits are 0, the length is 2^{16})
		If eop == 0 && stop == 0; hw_len is not valid. (i.e you have to use sw_len to get the number of transferred bytes)

Table 7-2 Contents of the DMA Descriptor

Note 5: The 32-bit status field is only written if **eop** or **stop** is set to one. For software to be able to know that the status field has been written or not, the status field of the descriptor must be cleared to all zeros before the DMA channel is started. The **eop** bit is written in the last descriptor of a packet.

Note 6: For compatibility with future versions of ETRAX processors, reserved fields must be set to 0 before starting DMA. When read, no assumption can be made about their value.

7.4 DMA Registers, Linked Lists, and Descriptor Format for USB

7.4.1 DMA Registers for USB

Figure 7-7 below shows the DMA registers for USB support in the ETRAX 100LX:

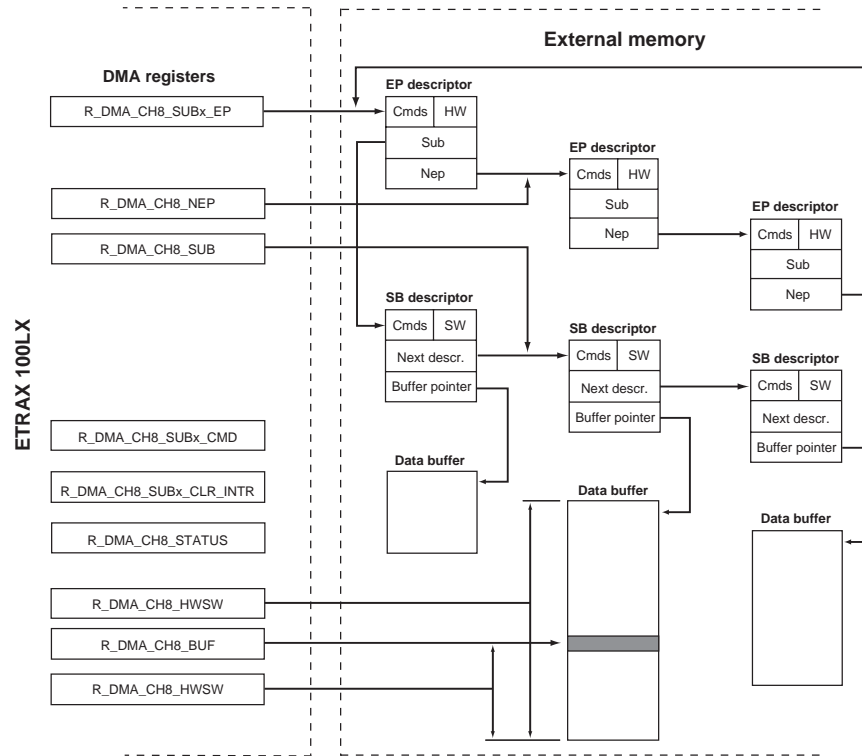


Figure 7-7 DMA Registers for USB

During normal operation, the only registers accessed by software are:

- R_DMA_CH8_SUBx_EP
- R_DMA_CH8_SUBx_CMD
- R_DMA_CH8_SUBx_CLR_INTR

R_DMA_CH8_SUBx_EP is used to locate the circular list of EP descriptors, R_DMA_CH8_SUBx_CMD is used to start the sub channel, and R_DMA_CH8_SUBx_CLR_INTR is used to clear interrupts. For more details see chapter 8 *USB*.

R_DMA_CH8_SUBx_EP points to the current *Endpoint* (EP) descriptor in the EP list. R_DMA_CH8_NEP points to the next EP descriptor in the EP list. R_DMA_CH8_SUB points to the current *Sublist* (SB) descriptor in the current sublist. The subchannel is either enabled or disabled with R_DMA_CH8_SUBx_CMD, and the operation of the sub channels are then controlled by the USB hardware. Interrupts are cleared by writing to R_DMA_CH8_SUBx_CLR_INTR.

7.4.2 DMA Linked Lists for USB

To handle USB in a practical way, DMA channel 8 has extended functionality to handle four sets of two dimensional lists. One set of lists is for each transfer type: control-, interrupt-, isochronous-, and bulk-transfers. When the two dimensional lists are used, the priorities of channels 8 and 9 are higher than channel 2, but lower than channel 1.

The construction of a linked list is done by defining DMA descriptors. There are three descriptor formats in the ETRAX 100LX. The first is the standard format and is used for all non USB communication and incoming USB communication. The other two formats are used only by out going USB communication. One forms a list of endpoints, and the other forms lists of data to the individual endpoints. See 7.4.3 *DMA Descriptor Format for USB*.

You can view the DMA list structure as a two dimensional list where the endpoint descriptors form one dimension, and by connecting a list of USB Sub-list descriptors to each endpoint descriptor, the USB Sublist forms the second dimension.

The same DMA list structure is used both when the USB controller is in Host mode, and when it is in Device mode. In Device mode, however, the endpoint list only contains one endpoint descriptor.

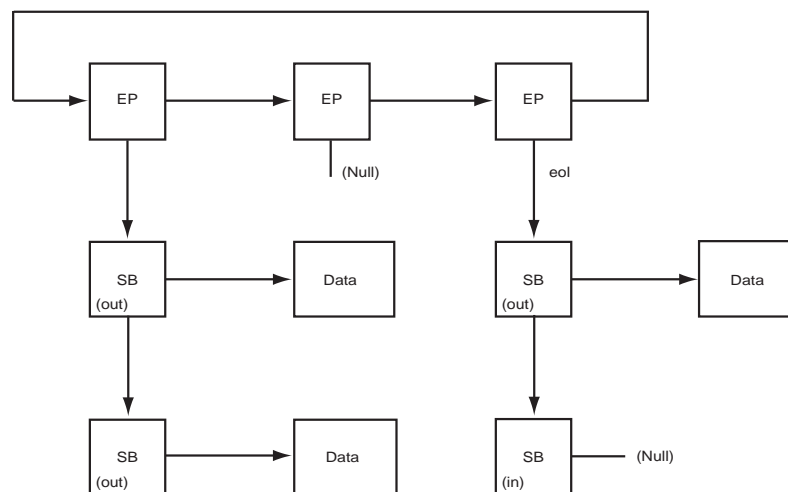


Figure 7-8 DMA List Structure for USB

There is no information in the descriptors telling which descriptor format a specific descriptor has. DMA interprets the descriptors based on how it is started, and on what internal state it is currently in.

7.4.3 DMA Descriptor Format for USB

The Standard Descriptor Format

The standard descriptor format in figure 7-9 below is similar to figure 7-6 with the following extensions of the 32-bit status field for channel 9 when connected to the USB interface.

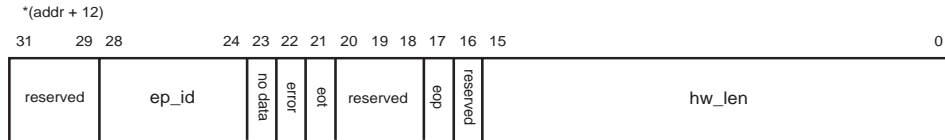


Figure 7-9 The DMA Descriptor Format for USB

This 32-bit field contains status information written by the DMA controller. Table 7-3 describes the contents of the standard DMA descriptor in more detail:

Bit	Name	Explanation
(addr + 12):		
31 - 29	reserved	(note 8)
28 - 24	ep_id	If eop == 1; ID of end point that is generating this data.
23	nodata	If eop == 1; There is no data in the buffer, and hw_len is not valid. Used when End Of USB Transfer was indicated by an empty USB packet.
22	error	If eop == 1; Received USB transaction has an error. Error status can be read in USB controller using ep_id as index. The corresponding endpoint is disabled to prevent status from being lost.
21	eot	If eop == 1; End Of USB Transfer.
20 - 18	reserved	(note 8)
17	eop	Last descriptor in a received packet.
16	reserved	(note 8)
15-0	hw_len	If (eop == 1 && !nodata); Number of received bytes in buffer. (If all bits are 0, the length is 2 ¹⁶)

Table 7-3 Contents of the Standard Descriptor Format

Note 7: The 32-bit status field is only written if **eop** is set to one. For software to be able to know that the status field has been written or not, the status field of the descriptor must be cleared to all zeros before the DMA channel is started. The **eop** bit is written in the last descriptor of a packet.

Note 8: For compatibility with future versions of ETRAX processors, reserved fields must be set to 0 before starting DMA. When read, no assumption can be made about their value.

USB EndPoint List Descriptor Format, EP

The DMA controller will only assume this format for sub channels of channel 8. Note that EndPoint descriptors must be 32-bit aligned.

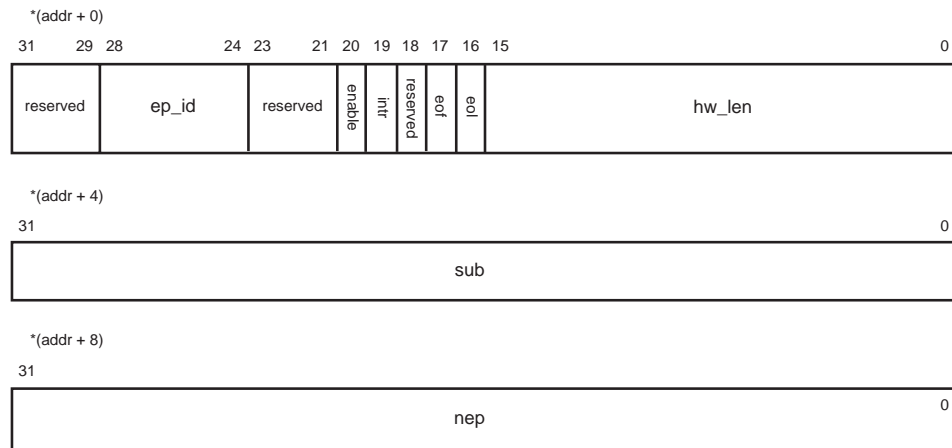


Figure 7-10 USB EndPoint List Descriptor Format, EP

The first 32-bit field of the descriptor gives a number of commands and an area **hw_len** for the DMA controller to save internal information. The second 32-bit field gives the address to the associate sub list. The last 32-bit field gives the address to the next EP descriptor in the EP list.

Table 7-4 describes the contents of the USB EndPoint list descriptor format, EP in more detail:

Bit	Name	Explanation
(addr + 0): (note 9)		
31 - 29	reserved	(note 10)
28 - 24	ep_id	Endpoint ID. Used by USB controller to associate a sublist with an endpoint. (note 11)
23 - 21	reserved	(note 10)
20	enable	Enable. 0: There is no sublist, advance to next endpoint. 1: There is a sublist, start processing the sublist. Note that software can only enable an endpoint, and that it has to be done according to the procedure in section 8.8.5 <i>Managing SB Descriptor Lists in Host Mode</i> in Chapter 8 <i>Universal Serial Bus</i> . To disable an endpoint, the EP descriptor must be removed from the endpoint list according to section 8.8.4 <i>Managing EP Descriptor Lists in Host Mode</i> .
19	intr	Generate a <code>dma8_subx_descr</code> interrupt when advancing to next descriptor. (note 11)
18	reserved	(note 10).
17	eof	This is the last endpoint descriptor in a frame. Note that this bit will not generate any DMA interrupt. This field is not used in Device mode. (note 11)
16	eol	This is the last descriptor in the list. The list is assumed to be circular, and the <code>nep</code> field is used even if eol is true. (note 11)
15-0	hw_len	Counter used by the DMA controller to remember bytes left in current sublist data buffer. It should be cleared by software before the endpoint is enabled, and then ignored by software.
(addr + 4):		
31-0	sub	Pointer to first sublist descriptor in sublist. If <code>enable == 0</code> sub is not used, there are no alignment restrictions on sublist descriptors.
(addr + 8):		
31-0	nep	Pointer to next endpoint descriptor in endpoint list. The two least significant bits in the nep field must be zero since it is a requirement that endpoint descriptors are 32-bit aligned. (note 9)

Table 7-4 Contents of the USB EndPoint List Descriptor Format, EP

- Note 9:** EndPoint descriptors must be 32-bit aligned, i.e. `addr<1:0> == 00`.
- Note 10:** For compatibility with future versions of ETRAX processors, reserved fields must be set to 0 before starting DMA. When read, no assumption can be made about their value.
- Note 11:** The `ep_id`, `intr`, `eof`, and `eol` fields must not be changed when the `enable` field is equal to 1. Also, note that there is a special procedure to clear the `enable` field according to Table 7-4, and section 8.8.4 of chapter 8 *Universal Serial Bus*.

USB Sublist Descriptor Format, SB

The DMA controller will only assume this format for sub channels of channel 8.

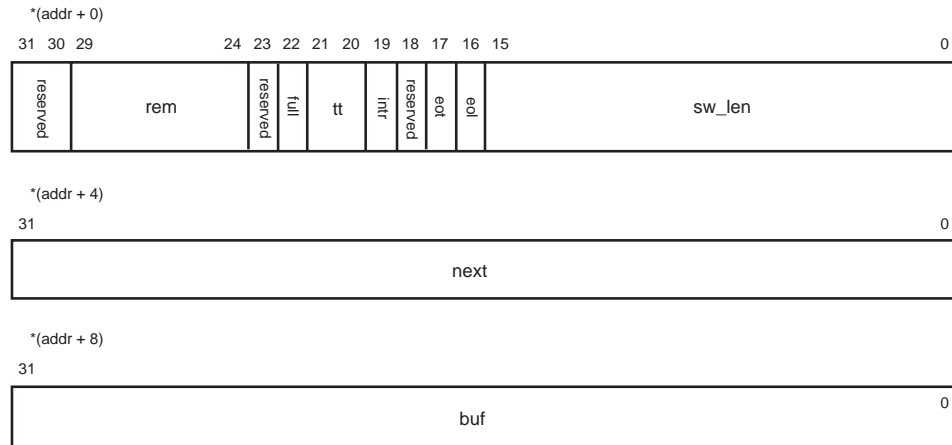


Figure 7-11 USB Sublist Descriptor Format, SB

The first 32-bit field of the descriptor gives a number of commands, and the length of the data buffer for outgoing transfers is given in **sw_len**. For Host mode IN transfers, **sw_len** gives the number of packets to be received by DMA channel 9. The second 32-bit field gives the address to the next descriptor in the linked list. The third 32-bit field gives the address to the data buffer **buf** for outgoing transfers. For Host mode IN transfers there is no data buffer appended, so **buf** should be set to NULL.

Table 7-5 describes the contents of the USB Sublist descriptor format, SB in more detail:

Bit	Name	Explanation
(addr + 0):		
31 - 30	reserved	(note 13)
29 - 24	rem	Expected number of bytes in the last packet of a Host mode IN transfer. Used to tell the USB controller the expected length of the last packet in a Host mode IN transfer. The expected number of packets in a Host mode IN transfer is the number of bytes in the data buffer at buf. This field is not used in Device mode.
23	reserved	(note 13).
22	full	If eot==1, full==1 tells the USB that an out transfer is a full length transfer. In the special case where the transfer-length is evenly divisible by the packetsize, this field prevents the USB sending an empty packet. In all other cases this field is don't care.
21 - 20	tt	USB Transfer Type: 00: zout (Zero length output transfer) 01: in (Input transfer) (not used in Device mode) 10: out (Output transfer) 11: setup (Setup transfer) (not used in Device mode)
19	intr	Generate a dma8_subx_descr interrupt when advancing to the next descriptor.
18	reserved	(note 13).
17	eot	This is the last descriptor in a USB transfer. Note that this bit will not generate any DMA interrupt.
16	eol	This is the last descriptor in the sublist. When eol is true (1), the next field is not used.
15-0	sw_len	Length in bytes of data buffer for outgoing transfers, and number of packets for Host mode IN transfers (If all bits are 0, the length is 2 ¹⁶).
(addr + 4):		
31-0	next	Pointer to next descriptor in list (no alignment restrictions). If eol == 1 next is not used.
(addr + 8):		
31-0	buf	Pointer to first byte in the data buffer (no alignment restrictions). (note 12)

Table 7-5 Contents of the USB Sublist Descriptor Format, SB

Note 12: For a Host mode IN transfer, DMA will not use the data referenced by buf.

Note 13: For compatibility with future versions of ETRAX processors, reserved fields must be set to 0 before starting DMA. When read, no assumption can be made about their value.

7.5 DMA Interrupt

There are two interrupts per channel: a descriptor interrupt, and an end-of-packet interrupt. Each channel has its own interrupt vector, and can be individually enabled or disabled. For a list of all interrupt vector numbers see chapter *18 Interrupts*.

For output channels, the interrupts are generated if the **intr** bit (descriptor interrupt) or **eop** bit (end-of-packet interrupt) in the descriptor are set. The interrupts are generated after DMA has read all data from the associated data buffer. If the **wait** bit is set in the descriptor, the interrupt is delayed until the DMA FIFO is emptied by the connected I/O interface.

For input channels the descriptor interrupt is generated if the **intr** bit is set in the descriptor, and the end-of-packet interrupt is generated when the peripheral interface signals end-of-packet to the DMA controller. The interrupts are generated after DMA has written all the data to the associated data buffer.

Interrupts are cleared with a write to the `R_DMA_CHx_CLR_INTR` internal register. For a more detailed information regarding enable, disable, and read interrupts, refer to chapter *18 Interrupts*, or chapter *19 Internal Registers*.

7.6 DMA Transfer/Setup Examples

In the following examples, a pseudo code notation is used to access bit fields in internal registers: *REG.field*.

Here *REG* is the name of the register and *field* is the bit field in the register. Also, symbolic constants for values are used. See chapter *19 Internal Registers 19.14* for a list of valid register, field, and constant names.

In addition, the following C structure is assumed for descriptors where a byte is an 8-bit unsigned integer, a word a 16-bit unsigned integer, and a dword a 32-bit unsigned integer:

```
struct descriptor
{
    word sw_len;
    word ctrl;
    dword next;
    dword buf;
    word hw_len;
    byte status;
    byte fifo_len;
}
```

For a information regarding USB to DMA transfer/setup examples, please refer to section *8.8 Procedures*, in chapter *8 Universal Serial Bus*.

7.6.1 Initiate and Setup a DMA Transfer

To initiate and setup a DMA transfer you typically:

- 1 Connect the I/O interface to the desired DMA channel in R_GEN_CONFIG.
- 2 Initiate the I/O interface.
- 3 Reset the DMA channel by writing the reset-cmd to R_DMA_CHx_CMD.
- 4 Initiate the linked list.
- 5 Set R_DMA_CHx_FIRST to the first descriptor.
- 6 Start DMA by writing start-cmd to R_DMA_CHx_CMD.
- 7 Start the I/O interface.
- 8 Wait for the DMA transfer to end.

This is the typical sequence of steps to initiate and set up a DMA transfer. However, for some I/O interfaces the sequence has to be altered. See each interface's respective chapter for a description of possible alterations to the procedure above.

7.6.2 Reset DMA Channel

To reset a DMA channel and its FIFO, the reset-cmd is written to R_DMA_CHx_CMD. The reset takes a while (>100ns), and when it is done DMA clears R_DMA_CHx_CMD.

Pseudo C-macros, and code for a DMA reset:

```
#define reset_dma( n ) \  
    R_DMA_CHn_CMD.cmd = reset  
  
#define wait_reset_dma( n ) \  
    while( R_DMA_CHn_CMD.cmd != hold )
```

Example:

```
/* Reset channel 0, 3, and 7. */  
reset_dma( 0 );  
reset_dma( 3 );  
reset_dma( 7 );  
  
/* Wait for reset do complete. */  
wait_reset_dma( 0 );  
wait_reset_dma( 3 );  
wait_reset_dma( 7 );
```

7.6.3 Initiating Linked List

The following is a C-macro to initiate a descriptor, and how to use it to form a linked list. The **next** field is Don't-Care at end of list.

```
#define descr( buf, len, next, flags ) \
    { len, flags, next, buf, 0, 0, 0 }
```

Example:

```
dma_descr olist[] = {
    descr( pattern0, sizeof pattern0, &olist[1], d_int | d_eop ),
    descr( pattern1, sizeof pattern1, &olist[2], 0 ),
    descr( pattern2, sizeof pattern2, 0xDC, d_eol )
};
```

7.6.4 Start a DMA Transfer

To start a DMA transfer `R_DMA_CHx_FIRST` and `R_DMA_CHx_CMD` have to be programmed. A sample C-macro for this is:

```
#define start_dma( n, desc ) \
    R_DMA_CHn_FIRST.first = (unsigned) desc; \
    R_DMA_CHn_CMD.cmd = start
```

Example:

```
start_dma( 0, olist ); /* Start channel 0 with olist. */
```

7.6.5 Restart a DMA Transfer

To append new data to a linked list the following steps should be taken. Note that the sequence of these steps is important:

- 1 Initiate the new list that shall be appended to the old list.
- 2 Set End-Of-List in the last descriptor of the new list.
- 3 Update the **next** field in the last descriptor of the old list.
- 4 Clear End-Of-List in the last descriptor of the old list.
- 5 Write the restart command to `R_DMA_CHx_CMD`.

The following code will work regardless of whether you give the restart command before or after DMA has finished the old list.

```
#define append_descr( n, old_last, new_first ) \
    old_last->next = new_first; \
    old_last->cmds =& ~d_eol; \
    R_DMA_CHn_CMD.cmd = restart
```

Example:

```
/* Initiate new list. */
dma_descr nlist[] = {
    descr( pattern3, sizeof pattern3, &nlist[1], d_int | d_eop ),
    descr( pattern4, sizeof pattern4, &nlist[2], 0 ),
    descr( pattern5, sizeof pattern5, 0xDC, d_eol )
};
append( 0, &nlist[2], nlist );
```

7.6.6 Hold DMA Temporarily and Continue Later

To temporarily hold DMA the following two C *macros* could be used.

```
#define hold_dma( n ) \
    R_DMA_CHn_CMD.cmd = hold

#define continue_dma( n ) \
    R_DMA_CHn_CMD.cmd = continue
```

7.7 Memory to Memory DMA

DMA channels 6 and 7 can be set up for memory to memory DMA, where the FIFO buffers of channels 6 and 7 connect directly to each other. This is configured in R_GEN_CONFIG.

To make sure the last part of the memory to memory transfer is written out from the destination FIFO to external memory in the destination list, either the **eop** bit has to be set in the last descriptor of the source list, or the software has to manually flush the destination FIFO by writing to the R_DMA_SET_EOP register when DMA channel 7 has read all data from the source list and stopped.

7.8 External DMA Channels

The external DMA channels `ext_dma0` and `ext_dma1` offer a DMA-like interface between external memory and external I/O devices. The external DMA channels operate in a “pseudo DMA” fashion, which differs from an ordinary DMA operation in that the data from the memory or the I/O device pass through the FIFO buffer of an internal DMA channel (see figure 7-12). This construction has the advantage of the bus width and timing for the I/O device, being different from the bus width and timing for memory since the accesses are performed in separate bus cycles.

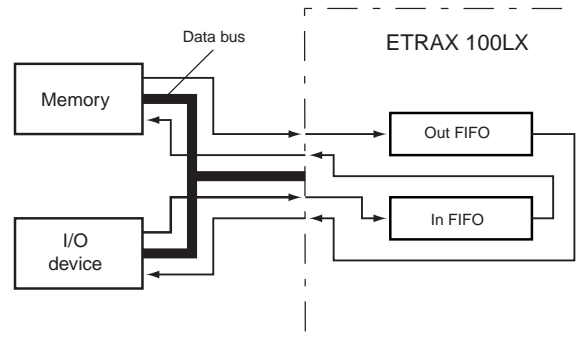


Figure 7-12 ETRAX 100LX External DMA Channels' Pseudo DMA Principle

Note 14: External DMA has the highest priority and might starve the other I/O interfaces such as Ethernet.

`ext_dma0` and `ext_dma1`, which are available for external I/O, are bidirectional:

- `ext_dma0` uses the internal DMA channels 4 and 5.
- `ext_dma1` uses the internal DMA channels 6 and 7.

The data bus width of the DMA transfers can be configured to either 8, 16, or 32 bits. Each external DMA channel uses two control signals to enable a handshake procedure:

- DMA request (`dreq0`, `dreq1`)
- DMA acknowledge (`dack0`, `dack1`)

7.8.4 Request/Acknowledge Signaling

Each external DMA channel has a pair of handshaking signals, **dreq** and **dack**. Both can be configured to be active high or active low. There are two modes for the request/acknowledge signalling: *handshake mode* and *burst mode*.

Handshake Mode

In handshake mode, a 4-phase handshaking scheme is used:

- 1 The I/O device sets **dreq** active.
- 2 The ETRAX 100LX sets **dack** active and performs a read or write operation to the I/O device (i.e. an external DMA bus cycle).
- 3 The I/O device negates **dreq**. This can be done immediately after **dack** gets active, and does not have to wait for the bus cycle to complete.
- 4 The ETRAX 100LX negates **dack**. The ETRAX 100LX will never negate **dack** before the external DMA bus cycle is completed.

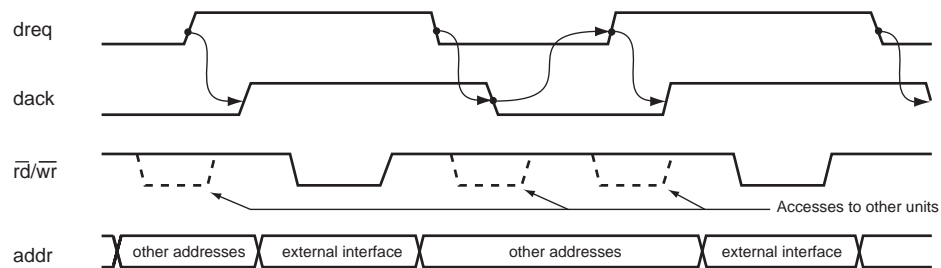


Figure 7-14 Timing Diagram for Handshaking Mode

Burst Mode

In burst mode, the ETRAX 100LX will always release the **dack** signal immediately after the completion of the external DMA bus cycle, and will continue to issue new external DMA bus cycles as long as the I/O device keeps the **dreq** active. Each cycle will be accompanied by a **dack** pulse. If the I/O device wants to stop or pause the transfers, it has to negate the **dreq** immediately after receiving the **dack** for the last cycle.

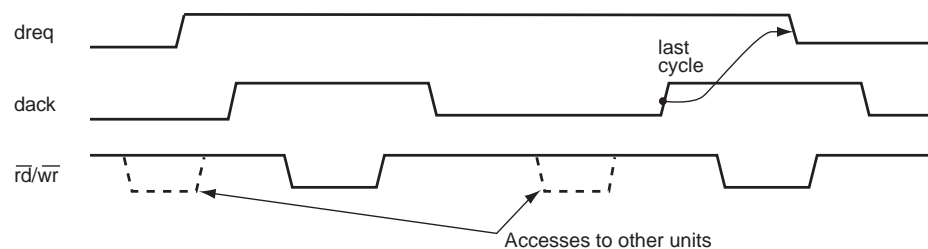


Figure 7-15 Timing Diagram for Burst Mode

7.8.5 Start and Stop of the Transfers

After initializing the external DMA (see section 7.8.3 *Initialization*), the external DMA interface is started by setting the **run** bit in the R_EXT_DMA_x_CMD register. When the external DMA channel is running, the external I/O device can start and stop the transfers by activating and deactivating the **dreq** signal.

Except for **dreq** going inactive, the transfers could stop for one of the following reasons:

- The external DMA channel is stopped by the software, by setting the **run** bit in R_EXT_DMA_x_CMD to **stop**.
- The transfer counter (see below) is in use and has expired.
- The descriptor list of the associated internal DMA channel contained an **eop** flag (only in the case of transfers from ETRAX 100LX to the I/O device).
- The data buffers of the associated internal DMA channel are exhausted.

If one of the first two cases occurs during input from the external I/O to the ETRAX 100LX, an **eop** status bit will be set in the corresponding DMA descriptor in the descriptor list for the associated internal DMA channel, and the internal DMA channel will advance to the next descriptor.

7.8.6 Transfer Counter

The number of transfers performed by the external DMA channel can be controlled by a transfer counter (**tfr_count** field in register R_EXT_DMA_x_CMD). The transfer counter is configured with the desired number of transfers (where a transfer is either 8, 16, or 32 bits), and will be decremented by one for each transfer. When the counter reaches zero, it will stop the transfers and clear the **run** bit in R_EXT_DMA_x_CMD. The current values of the **run** bit and the transfer counter can be read from the register R_EXT_DMA_x_STAT.

7.8.7 External DMA Interrupts

Each external DMA channel will generate an interrupt, **ext_dma0** and **ext_dma1** respectively, whenever the run bit in R_EXT_DMA_x_CMD is cleared. When not waiting for an external DMA transfer to complete, the interrupt should be masked off in R_IRQ_MASK0_SET. For further information about interrupts refer to chapter 18 *Interrupts*.