Notes:
- A transaction is a single transfer (up to 4 bytes) from source to destination and takes at least two bus cycles.
- A Cell transfer is the number of bytes transferred per event and may require several transactions.
- Events include software initiated transfers and interrupt flag initiated transfers.
- A Block transfer is the number of bytes transferred per DMA channel enable and may require several Cell transfers.
- The Block transfer size is the greater of DSIZ and SSIZ.
- If CSIZ is greater than the Block transfer size, only one Cell transfer takes place and it is limited to the Block transfer size.
- All addresses are physical addresses.
- Pointer registers are read only. Other registers should not be changed while the channel is enabled.
- Pointer registers are updated with each transfer and are reset when the associated portion of the transfer (i.e. cell, source or destination) is “done” (matches the size register). They are also reset as noted in 31.3.2.1.
**DCHxCON** DMA Channel Control Register

**CHEN:** Channel Enable
Channel Registers should not be modified with the channel enabled. Disabling the channel does not reset pointers. If a channel is disabled in the middle of a transaction, the transaction is completed.

**CHAEN:** Channel Automatic Enable
Channel pointers reset at the end of a block transfer. With auto enable set, the channel automatically re-enables (sets CHEN) as well, and a new Block transfer can begin.

**CHEDET:** Event detect
A read only status bit that indicates if an event occurred.

**CHAED:** Channel Allow Events when Disabled
Event detect normally operates only when the channel is enabled. Setting this bit allows events to be registered even when the channel is disabled. If the channel is later enabled, the transfer will begin immediately.

**CHCHN:** Channel Chain
Enables this channel for chaining. This channel will be enabled automatically (CHEN) when the channel it is chained to completes its block transfer.

**CHCHNS:** Channel Chain Selection
This channel chains to the next higher (1) or lower (0) numbered DMA channel. Note lower numbered channels are higher in natural priority.

**CHPRI<1:0>:** Channel Priority
Sets the priority of the channel. Higher priority channels transfer data before lower priority channels. 3 = highest, 0 = lowest. Applies only when multiple channels have simultaneous transfers pending.

**DCHxECON** DMA Channel Event Control Register

**CHSIRQ<7:0>:** Start IRQ
Selects the IRQ that will trigger the start of each DMA cell transfer. The DMA controller maintains and manages its own event flags, independent of IRQIF and IRQIE flags. Setting IRQIF will not start a DMA transfer.

**SIRQEN:** Start IRQ enable

**CHAIRQ<7:0>:** Abort IRQ
Selects the IRQ that can trigger a premature end of the DMA transfers (i.e. UART OERR)

**AIRQEN:** Abort IRQ enable

**CFORCE:** Channel Force
Setting this bit will start a DMA cell transfer. This bit always reads as 0.

**CABORT:** Channel Abort
Setting this bit will end the DMA transfers on the channel – completes current transaction, resets CHEN (in dependant of CHAEN), resets pointers and resets the IRQ event detector.

**PATEN:** Pattern Enable
Enable pattern matching to end DMA transfers. If one piece of data transferred in a cell matches the pattern, the unit functions as if the block is complete.
DCHxINT  DMA Channel Interrupt Control Register

CHSDIE, CHSDIF:  Source Done Interrupt enable and flag
CHSHIE, CHSHIF:  Source Half Done Interrupt enable and flag
CHDDIE, CHDDIF:  Destination Done Interrupt enable and flag
CHDHIE, CHDHIF:  Destination Half Done Interrupt enable and flag
CHBCIE, CHBCIF:  Block Transfer Complete Interrupt enable and flag
CHCCIE, CHCCIF:  Cell Transfer Complete Interrupt enable and flag
CHTAIE, CHTAIF:  Transfer Aborted Interrupt enable and flag
CHERIE, CHERIF:  Address Error Interrupt enable and flag
Illegal/Invalid Source or Destination address generated

All DMA interrupts for one channel go through one vector, with one group priority. The service routine can use DCHxINT to determine the cause of the interrupt. Note that both IFS1.DMAxIF flags and DCHxINT flags must be cleared to clear the interrupt.

DMACON  Overall DMA Module Control

ON:  DMA Module On. Off resets all pointers.
FRZ:  DMA Module freezes during Debug
SIDL:  DMA transfers are frozen during Sleep

SUSPEND:  Writing a 1 to this bit requests the DMA unit to stop transfers on all channels. The read or write in progress (not necessarily a full transfer) is completed. Poll the bit to see if the suspend request has taken effect. Writing a 0 re-starts the suspended transfer.

DMASTAT  Overall DMA Module Status

DMACH<1:0>:  Lists most recently active DMA channel

RDWR:  Last DMA access was Read (0) or Write (1)

DMAADDR  Last DMA address used
A summary of the status flags affected by channel transfer initiation or termination is provided in Table 31-5. Channel abort events are allowed if the channel is enabled, CHEN = 1, or if the user elects to allow events while the channel is disabled, CHAED = 1.

### Table 31-5: Channel Event Behavior

<table>
<thead>
<tr>
<th>Event</th>
<th>Description and Function</th>
<th>Registers Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Events Initiating Transfers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Interrupt Matching CHSIRO&lt;7:0&gt;(1,2)</td>
<td>The channel event detect will be set.</td>
<td>CHEDET = 1</td>
</tr>
<tr>
<td>Channel Chain Event</td>
<td>This will enable the channel if not already set. If an event detect is pending, a channel transfer will begin immediately.</td>
<td>CHEN = 1</td>
</tr>
<tr>
<td>User Writes the CFORCE Bit(1)</td>
<td>The channel event detect will be set.</td>
<td>CHEDET = 1</td>
</tr>
<tr>
<td><strong>Events Terminating Transfers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Interrupt Matching CHAIRO&lt;7:0&gt;(1,2)</td>
<td>The channel event detect will be reset and the channel turned off. The abort interrupt flag is set.</td>
<td>CHEDET = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHEN = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHAIF = 1</td>
</tr>
<tr>
<td>Pattern Match(1)</td>
<td>This occurs when any byte of data written in a transaction matches the data in CHPDAT. The channel event detect is reset. The channel is turned off if CHAEN = 0. This event is treated as a completed block transfer. Pointers are reset.</td>
<td>CHEDET = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHEN = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHBCIF = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHSPTR = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHDPTR = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHCPTR = 0</td>
</tr>
<tr>
<td>Cell Transfer is Complete</td>
<td>This occurs when CHCSIZ&lt;7:0&gt; bytes have been transferred. The transfer event detect is reset and the channel remains enabled pending the next event.</td>
<td>CHEDET = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHCCIF = 1</td>
</tr>
<tr>
<td>Block Transfer is Complete</td>
<td>The channel event detect is reset. The channel is turned off if CHAEN = 0. This event is treated as a completed transfer. Pointers are reset.</td>
<td>CHEDET = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHEN = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHBCIF = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHSPTR = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHDPTR = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHCPTR = 0</td>
</tr>
<tr>
<td>User Writes the CABORT bit</td>
<td>The channel is turned off and the channel event detect is reset. The pointers are reset.</td>
<td>CHEDET = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHEN = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHSPTR = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHDPTR = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHCPTR = 0</td>
</tr>
<tr>
<td>Address Error is Detected</td>
<td>The channel is turned off and the event detect is reset. The address error interrupt flag is set.</td>
<td>CHEDET = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHEN = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHERIF = 1</td>
</tr>
</tbody>
</table>

**Note 1:** Events are allowed only when the channel is enabled, or the user allows events while disabled (CHEN = 1 or CHAED = 1).

**Note 2:** The DMA Controller maintains its own flags for detecting start and abort IRQs in the system, and is completely independent of the INT Controller IES/IFS flags. Once the start or abort IRQ system events are triggered, they will be detected automatically by the DMA controller internal logic, without the need for user intervention.
31.3.2.1 Interrupt and Pointer Updates

The Source and Destination Pointers are updated after every transaction. Interrupts will also be set or cleared at this time. If a pointer passes the halfway point during a transaction, the interrupt will be updated accordingly.

Pointers are reset when any of the following occurs:

- On any device Reset
- When the DMA is turned off (ON bit (DMACON<15>) is 0’)
- A block transfer completes, regardless of the state of CHAEN (DCHxCON<4>)
- A pattern match terminates a transfer, regardless of the state of CHAEN (DCHxCON<4>)
- The CABORT (DCHxECON<6>) flag is written
- Source or destination start addresses are updated

Example 31-1: DMA Channel Initialization for Basic Transfer Mode Code Example

```c
/*
    The following code example illustrates the DMA channel 0 configuration for a data transfer.
*/
IEC1CLR=0x00010000; // disable DMA channel 0 interrupts
IFS1CLR=0x00010000; // clear existing DMA channel 0 interrupt flag
DMACONSET=0x00010000; // enable the DMA controller
DCH0CON=0x3; // channel off, pri 3, no chaining
CH0ECON=0; // no start or stop irq’s, no pattern match

// program the transfer
DCH0SSA=0xd1d010000; // transfer source physical address
DCH0DSA=0xd1d020000; // transfer destination physical address
DCH0SSIZ=0; // source size 256 bytes
DCH0DSIZ=0; // destination size 256 bytes
DCH0CSIZ=0; // 256 bytes transferred per event
DCH0INTCLR=0x00ff00ff; // clear existing events, disable all interrupts
DCH0CONSET=0x80; // turn channel on

// initiate a transfer
DCH0CONSET=0x00000080; // set CFORCE to 1

// do something else

// poll to see that the transfer was done

while(TRUE)
{
    register int pollCnt; // use a poll counter.
    // polling continuously the DMA controller in a tight
    // loop would affect the performance of the DMA transfer

    int dmaFlags=DCH0INT; // one of CHERIF (DCHxINT<0>), CHTAIF (DCHxINT<1>)
    if( (dmaFlags&0xb)) // or CHBCIF (DCHxINT<3>) flags set
    {
        break; // transfer completed
    }
    pollCnt=100; // use an adjusted value here
    while(pollCnt--); // wait before reading again the DMA controller
}
```

// check the transfer completion result
31.3.3 Pattern Match Termination Mode Operation

Pattern Match Termination mode allows the user to end a transfer if a byte of data written during a transaction matches a specific pattern, as defined by the DCHxDAT register. A pattern match is treated the same way as a block transfer complete, where the CHBCIF bit (DCHxINT<3>) is set and the CHEN bit (DCHxCON<7>) is cleared.

This feature is useful in applications where a variable data size is required and eases the set up of the DMA channel. UART is a good example of where this can be effectively used.

Assuming a system has a series of messages that are routinely transmitted to an external host and it has a maximum message size of 86 characters, the user would set the following parameters on the channel:

• DCHxSSIZ to 87 bytes:
  If something unexpected occurs the CPU program will be interrupted when the buffer overflows and can take the appropriate action.
• DCHxDSIZ set to 1 byte.
• The destination address is set to the UART TXREG.
• The DCHxDAT is set to 0x00, which will stop the transfer on a NULL character in any byte lane.
• The CHSIRQ (DCHxECON<15:8>) is set to the UART “transmit buffer empty” IRQ.
• The SIRQEN (DCHxECON<4>) is set to enable the channel to respond to the start interrupt event.
• The start address is set to the start address of the message to be transferred.
• The channel is enabled, CHEN = 1 (DCHxCON<7>).
• The user will then force a cell transfer through CFORCE (DCHxECON<7>) and the first byte transmission by the UART.
• Each time a byte is transmitted by the UART, the transmit buffer empty interrupt will initiate the following byte transfer from the source to the UART.
• When the DMA channel detects a NULL character in any of the byte lanes of the channel, the transaction will be completed and the channel disabled.

Pattern matching is independent of the byte lane of the source data. If ANY byte in the source buffer matches DCHxDAT, a pattern match is detected. The transaction will be completed and the data read from the source will be written to the destination.
31.3.8 Channel Priority and Selection

The DMA controller has a natural priority associated with each of the channels. Channel 0 has the highest natural priority. Each channel has two priority bits, CHPRI<1:0> (DCHxCON<1:0>). These bits identify the channel's priority. When multiple channels have transfers pending, the next channel to transmit data will be selected as follows:

- Channels with the highest priority will complete all cell transfers before moving onto channels with a lower priority (see behavior, “PRI3 xfers”, in Figure 31-4).
- If multiple channels have the same priority (identical CHPRI), the controller will cycle through all channels at that priority. Each channel with a cell transfer in progress at the highest priority will be allowed a single transaction of the active cell transfer before the controller allows a single transaction by the next channel at that priority level (see behavior, “PRI2 xfers” between markers “C” and “B”, in Figure 31-4).
- If a channel with a higher priority requests a transfer while another channel of lower priority has a transaction in process, the transaction will complete before moving to the channel with the higher priority (see events at markers “A” in Figure 31-4).

Figure 31-4: Channel Priority Behavior

Transition Legend:
A – Higher priority transfer request; suspend current and transfer next.
B – All highest priority transfers complete; drop to channels at lower priority.
C – Cycle through all channels at the current priority.
31.3.4 Channel Chaining Mode Operation

Channel chaining is an enhancement to the DMA channel operation. A channel (slave channel) can be chained to an adjacent channel (master channel). The slave channel will be enabled when a block transfer of the master channel completes, i.e., CHBCIF (DCHxINT<3>), is set. At this point, any event on the slave channel will initiate a cell transfer. If the channel has an event pending, a cell transfer will begin immediately.

The master channel will set its interrupt flags normally, CHBCIF (CHxINT<3>) and has no knowledge of the “chain” status of the slave channel. The master channel is still able to cause interrupts at the end of a DMA transfer if one of the CHSDIE/CHDDIE/CHBCIE (DCHxINT<23,21,19>) bits is set.

In the channels natural priority order, channel 0 has the highest priority and channel 4 the lowest. The channel higher or lower in natural priority, that can enable a specific channel, is selected by CHCHNS (DCHxCON<8>), provided that channel chaining is enabled, CHCHN = 1 (DCHxCON<5>).

A feature of the DMA module is the ability to allow events while the channel is disabled using CHAED (DCHxCON<6>). This bit is particularly useful in Chained mode, in which the slave channel needs to be ready to start a transfer as soon as the channel is enabled by the master channel.

The following examples demonstrate situations in which chaining may be useful:

1. Transferring data in one peripheral (e.g., from UART1, DMA channel 0, at 9600 baud, to SRAM) to another peripheral (e.g., from SRAM to UART2, DMA channel 1, at 19200 baud).
   In this example, CHAED will be set in both channels; with UART2 setting the event detect, CHEDET (DCHxCON<2>), on channel 1 when the last byte has been transmitted. As soon as channel 0 completes a transfer, channel 1 is enabled and the data is transferred immediately.

2. A/D converter transfers data to one buffer (connected to channel 0).
   When the destination buffer 0 is full (block transfer completes), channel 1 is enabled and further conversions are transferred to buffer 1. In this case, CHAED will not be enabled. If it were, the last word transferred by channel 0 would be transferred a second time by channel 1 (because the A/D converter interrupt event would have set the event detect flag CHEDET in both channels).
Example 31-3: DMA Channel Initialization in Chaining Mode Code Example

/*
The following code example illustrates the DMA channel 0 configuration for data transfer with
pattern match enabled. DMA channel 0 transfer from the UART1 to a RAM buffer while DMA channel 1
transfers data from the RAM buffer to UART2. Transferred strings are at most 256 characters
long. Transfer on UART2 will start as soon as the UART1 transfer is completed.
*/

unsigned char myBuff<256>; // transfer buffer
IEC1CLR=0x00010000;       // disable DMA channel 0 interrupts
IFS1CLR=0x00010000;       // clear any existing DMA channel 0 interrupt flag
DMACONSET=0x00008000;     // enable the DMA controller
DCH0CON=0x3;              // channel 0 off, priority 3, no chaining
DCH1CON=0x62; // channel 1 off, priority 2
             // chain to higher priority
             // (ch 0), enable events detection while disabled
DCH0ECON=(27 <<8)| 0x30; // start irq is UART1 RX, pattern enabled
DCH1ECON=(42 <<8)| 0x30; // start irq is UART1 TX, pattern enabled
DCH0DAT=DCH1DAT='\r';    // pattern value, carriage return

// program channel 0 transfer
DCH0SSA=VirtToPhys(&U1RXREG); // transfer source physical address
DCH0DSA=VirtToPhys(myBuff);   // transfer destination physical address
DCH0SSIZ=1;                   // source size is 1 byte
DCH0DSIZ=0;                   // dst size at most 256 bytes
DCH0CSIZ=1;                   // one byte per UART transfer request

// program channel 1 transfer
DCH1SSA=VirtToPhys(myBuff);   // transfer source physical address
DCH1DSA=VirtToPhys(&U2TXREG); // transfer destination physical address
DCH1SSIZ=0;                   // source size at most 256 bytes
DCH1DSIZ=0;                   // dst size is 1 byte
DCH1CSIZ=1;                   // one byte per UART transfer request

DCH0INTCLR=0x00ff00ff;       // DMA0: clear events, disable interrupts
DCH1INTCLR=0x00ff00ff;       // DMA1: clear events, disable interrupts
DCH1INTSET=0x00090000;       // DMA1: enable Block Complete and error interrupts

IPC9CLR=0x000001ff;         // clear the DMA channels 0 and 1 priority and
                            // sub-priority
IPC9SET=0x000000b16;        // set IPL 5, sub-priority 2 for DMA channel 0
                            // set IPL 2, sub-priority 3 for DMA channel 1
IEC1SET=0x00020000;         // enable DMA channel 1 interrupt

DCH0CONSET=0x80;            // turn channel on
                            // do something else

                            // the UART1 RX interrupts will initiate the DMA channel 0 transfer
                            // once this transfer is complete, the DMA channel 1 will start
                            // upon DMA channel 1 transfer completion will get an interrupt

while(!intCh1Ocurred);     // poll DMA channel 1 interrupt
### 31.3.9 Byte Alignment

The byte alignment feature of the DMA controller relieves the user from aligning the source and destination addresses. The read portion of a transaction will read the maximum number of bytes that are available to be read in a given word. For example, if the Source Pointer is N × 4 bytes from the source size, 4 bytes will be read if the Source Pointer points to byte 0, 3 bytes if the Source Pointer points to byte 1, etc. If the number of bytes remaining in the source is N < 4, only the first N bytes are read. This is important when the read includes registers that are updated on a read.

The Source Pointer and Destination Pointers are updated after every write, with the number of bytes that have been written. The user should note that in cases where a transfer is aborted, before a transaction is complete, the Source Pointer will not necessarily reflect the reads that have taken place.

An example of this behavior is given in Table 31-3. Example 1 demonstrates a simple transfer of 9 bytes between two large buffers, in which CHxSSA = 0x1000, CHxSSIZ = 100, CHxDISA = 0x43F9, CHxDISIZ = 100, and CHxXSIZ = 9.

#### Table 31-3: Source and Destination Pointer Updates – Example 1

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Operation</th>
<th>Source Pointer</th>
<th>Destination Pointer</th>
<th>Transfer Count/Size</th>
<th>Read Address</th>
<th>Write Address</th>
<th>Read Data(1)</th>
<th>Write Data(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>9</td>
<td>11</td>
<td>0/9</td>
<td>1009</td>
<td>xxxx</td>
<td>33_22_11_XX</td>
<td>XX_XX_XX_XX</td>
</tr>
<tr>
<td>1</td>
<td>Write1</td>
<td>9</td>
<td>11</td>
<td>0/9</td>
<td>1009</td>
<td>440A</td>
<td>33_22_11_XX</td>
<td>22_11_XX_XX</td>
</tr>
<tr>
<td>1</td>
<td>Ptr Update(3)</td>
<td>B</td>
<td>13</td>
<td>2/9</td>
<td>1009</td>
<td>440A</td>
<td>33_22_11_XX</td>
<td>XX_XX_XX_XX</td>
</tr>
<tr>
<td>1</td>
<td>Write2</td>
<td>B</td>
<td>13</td>
<td>2/9</td>
<td>1009</td>
<td>440C</td>
<td>33_22_11_XX</td>
<td>XX_XX_XX_33</td>
</tr>
<tr>
<td>1</td>
<td>Ptr Update(3)</td>
<td>C</td>
<td>14</td>
<td>3/9</td>
<td>1009</td>
<td>440C</td>
<td>33_22_11_XX</td>
<td>XX_XX_XX_XX</td>
</tr>
<tr>
<td>2</td>
<td>Read</td>
<td>C</td>
<td>14</td>
<td>3/9</td>
<td>100C</td>
<td>440C</td>
<td>77_66_55_44</td>
<td>XX_XX_XX_XX</td>
</tr>
<tr>
<td>2</td>
<td>Write1</td>
<td>C</td>
<td>14</td>
<td>3/9</td>
<td>100C</td>
<td>440D</td>
<td>77_66_55_44</td>
<td>66_55_44_XX</td>
</tr>
<tr>
<td>2</td>
<td>Ptr Update(3)</td>
<td>F</td>
<td>17</td>
<td>6/9</td>
<td>100C</td>
<td>440D</td>
<td>77_66_55_44</td>
<td>XX_XX_XX_XX</td>
</tr>
<tr>
<td>2</td>
<td>Write2</td>
<td>F</td>
<td>17</td>
<td>6/9</td>
<td>100C</td>
<td>4410</td>
<td>77_66_55_44</td>
<td>XX_XX_XX_77</td>
</tr>
<tr>
<td>2</td>
<td>Ptr Update(3)</td>
<td>10</td>
<td>18</td>
<td>7/9</td>
<td>100C</td>
<td>4410</td>
<td>77_66_55_44</td>
<td>XX_XX_XX_XX</td>
</tr>
<tr>
<td>3</td>
<td>Read</td>
<td>10</td>
<td>18</td>
<td>7/9</td>
<td>1010</td>
<td>4410</td>
<td>XX_XX_99_88</td>
<td>XX_XX_XX_XX</td>
</tr>
<tr>
<td>3</td>
<td>Write1</td>
<td>10</td>
<td>18</td>
<td>7/9</td>
<td>1010</td>
<td>4411</td>
<td>XX_XX_XX_88</td>
<td>XX_99_88_XX</td>
</tr>
<tr>
<td>3</td>
<td>Ptr Update(3)</td>
<td>12</td>
<td>1A</td>
<td>9/9</td>
<td>1010</td>
<td>4411</td>
<td>XX_XX_XX_88</td>
<td>XX_XX_XX_XX</td>
</tr>
</tbody>
</table>

**Note**

1: XX indicates that data read is discarded.
2: XX indicates that data that is NOT written.
3: Interrupts are updated when the pointers are updated as required.
Another example of this behavior is given in Table 31-4. Example 2 demonstrates worst-case bus utilization, i.e., unaligned buffers with destination buffer wrapping, in which CHxSSA = 0x1000, CHxSSIZ = 100, CHxDsa = 0x4402, CHxDSIZ = 4, and CHxXSIZ = 8.

### Table 31-4: Source and Destination Pointer Updates – Example 2

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Operation</th>
<th>Source Pointer</th>
<th>Destination Pointer</th>
<th>Transfer Count/Size</th>
<th>Read Address</th>
<th>Write Address</th>
<th>Read Data&lt;sup&gt;(1)&lt;/sup&gt;</th>
<th>Write Data&lt;sup&gt;(2)&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>9</td>
<td>0</td>
<td>0/8</td>
<td>1009</td>
<td>xxxx</td>
<td>33_22_11_XX</td>
<td>XX_XX_XX_XX</td>
</tr>
<tr>
<td>1</td>
<td>Write1</td>
<td>9</td>
<td>0</td>
<td>0/8</td>
<td>1009</td>
<td>4402</td>
<td>33_22_11_XX</td>
<td>22_11_XX_XX</td>
</tr>
<tr>
<td>1</td>
<td>Write2</td>
<td>B</td>
<td>2</td>
<td>2/8</td>
<td>1009</td>
<td>33_22_11_XX</td>
<td>XX_XX_XX_XX</td>
<td>XX_XX_XX_33</td>
</tr>
<tr>
<td>1</td>
<td>Write2</td>
<td>C</td>
<td>3</td>
<td>3/8</td>
<td>1009</td>
<td>4404</td>
<td>33_22_11_XX</td>
<td>XX_XX_XX_XX</td>
</tr>
</tbody>
</table>

| 2           | Read      | C              | 3                   | 3/8                | 100C         | 4404          | 77_66_55_44          | XX_XX_XX_XX           |
| 2           | Write1    | C              | 3                   | 3/8                | 100C         | 4405          | 77_66_55_44          | XX_XX_44_XX           |
| 2           | Write2    | D              | 0                   | 4/8                | 100C         | 4405          | 77_66_55_44          | XX_XX_XX_XX           |
| 2           | Write3    | F              | 2                   | 6/8                | 100C         | 4402          | 77_66_55_44          | XX_XX_XX_XX           |
| 2           | Write3    | 10             | 3                   | 7/8                | 100F         | 4404          | 77_66_55_44          | XX_XX_XX_77           |
| 3           | Write3    | 10             | 18                  | 7/8                | 1010         | 4404          | BB_AA_99_88          | XX_XX_XX_XX           |
| 3           | Write1    | 10             | 18                  | 7/8                | 1010         | 4405          | BB_AA_99_88          | XX_XX_88_XX           |
| 3           | Write3    | 11             | 1A                  | 8/8                | 1010         | 4405          | 77_66_55_44          | XX_XX_XX_XX           |

**Note:**
1: XX indicates that data read is discarded.
2: XX indicates that data that is NOT written.
3: Interrupts are updated when the pointers are updated as required.