AN2281
Rev. 1, 11/2004

Two-Dimensional DMA Buffer Examples for the MSC8101

By Flora P. McArthur and Iantha E. Scheiwe

The Freescale MSC8101 DSP device includes an efficient direct memory access (DMA) module to transfer data between various sections of the device as well as to external memory or peripherals. The DMA programming is flexible and allows you to configure many types of buffers according to the needs of an application. In the DSP56300 devices, the one-dimensional, two-dimensional, and three-dimensional modes of DMA transfers are available and specified by setting a bit in the DMA control register. The MSC8101 does not have a specific bit to enable two-dimensional transfers, but the flexible programming allows you to create this type of buffer. This application note describes the buffer planning and DMA programming required for a two-dimensional DMA transfer with two buffers. Each buffer is programmed with its size, address, and characteristics. Chaining two separate buffers to each other creates a two-dimensional transfer. This methodology can be applied to two-dimensional transfers for multiple buffers as well.[1]

In this particular application, data from two data channels is received into a single buffer. The data from each channel must be separated by channel before processing by the MSC8101 StarCore™ SC140 core and the Enhanced Filter Coprocessor (EFCOP). A dual-access DMA transfer accomplishes the task. The data transfers from a single buffer to the DMA FIFO and

CONTENTS

1. This application note assumes knowledge of the MSC8101 DMA. Refer to the DMA chapter of the *MSC8101 Reference Manual* and the *MSC8101 User's Guide* for details on DMA functionality.

then moves to the two chained buffers. After the first buffer receives the first group of data, the second buffer receives the second group of data. Next, the first buffer receives the third group of data, and the second buffer receives the fourth group. This process continues until the voice data is separated.

After the SC140 core and EFCOP process the two separate channels of data, this data must be combined in its original format. Two buffers are chained together so that data is transferred from each buffer to the DMA FIFO. After the first and second buffers transfer data to the DMA FIFO, the DMA FIFO receives data from the first buffer and then data from the second buffer. The DMA FIFO transfers the data in this format to a single buffer, and the data transfer process is complete.

1 DMA Buffer Basics

A two-dimensional DMA transfer requires a source or destination buffer with blocks of data predictably organized in a single buffer. The first dimension is the size of the data to be transferred by each user. The second dimension is the distance between consecutive datums of the same user. In **Figure 1**, two blocks of data are interspersed in a known pattern. There are two channel buffers, each with its own buffer size/addressing requirements. The next sections describe specific two-dimensional buffer examples and their data transfer implementation on the MSC8101.

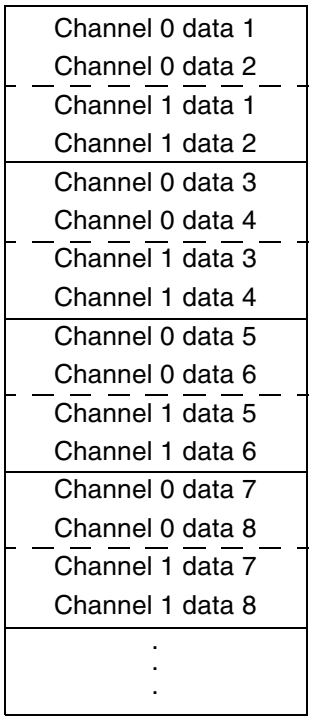

Figure 1. Two-Dimensional Buffer Example

1.1 Buffer 1 Example

In a given system, incoming voice data for two voice channels is placed into a memory buffer. The data in each channel has a real and an imaginary component. The data for each channel is placed immediately after the data for the preceding channel. That is, channel 0 real is placed into the buffer followed by channel 0 imaginary. Next, channel 1 real is placed into the buffer followed by channel 1 imaginary. Now, the next channel 0 data is received, and so on.

Once the data is received into the memory buffer, resident in external SDRAM, it can be transferred into buffers in internal memory for filter processing on the MSC8101. However, the application requires both the SC140 core and the EFCOP to filter process the data. The SC140 core processes the data from channel 0, and the EFCOP processes the data from channel 1. The data should be in separate buffers; that is, the channel 0 data should be in a completely separate buffer from the channel 1 data. **Figure 2** shows this transfer requirement.

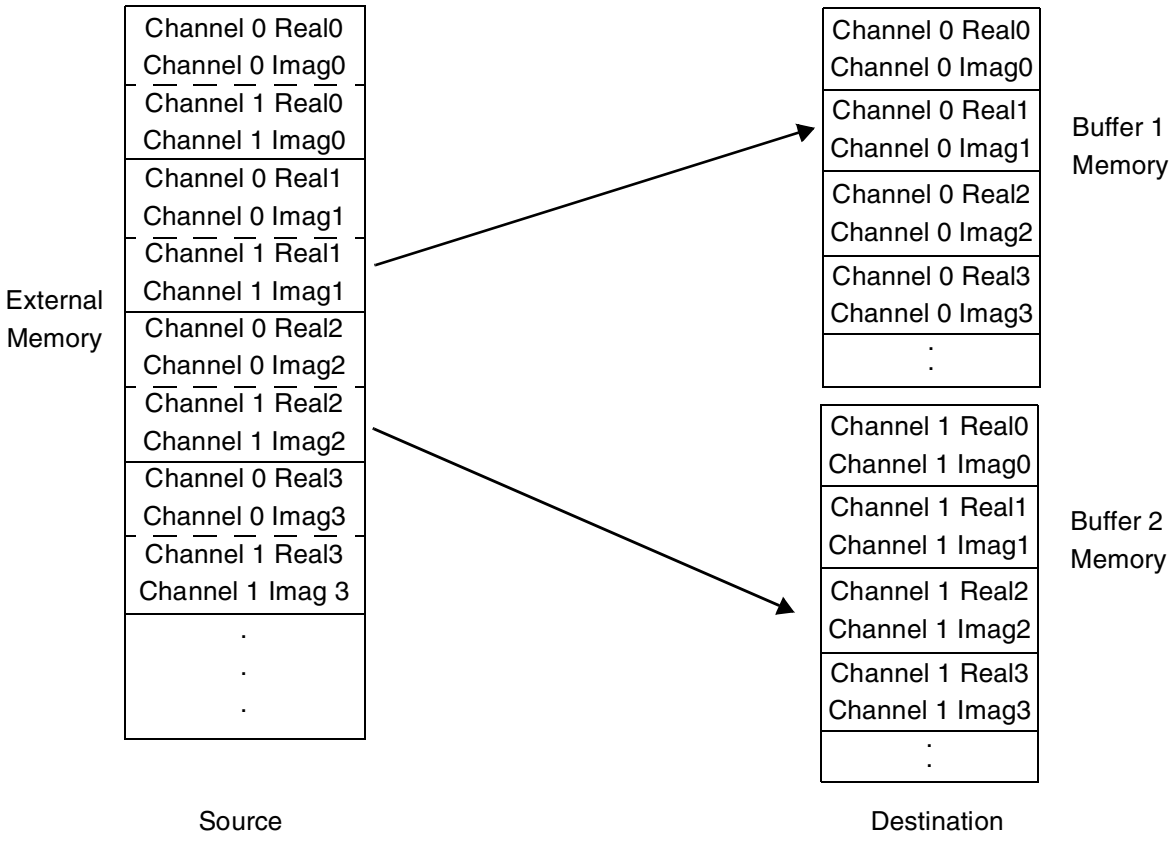

Figure 2. Transfer Combined Buffer Into Two Separate Buffers

1.2 Buffer 2 Example

After the SC140 core and the EFCOP process the data, the results of each channel reside in separate buffers in internal memory. The data must be in the original format in which it was received. Thus, the results must be transferred from the two separate buffers to a single buffer in external memory. **Figure 3** shows this transfer requirement. The data in channel 0 and channel 1 is combined in the same order as it is received in buffer example 1

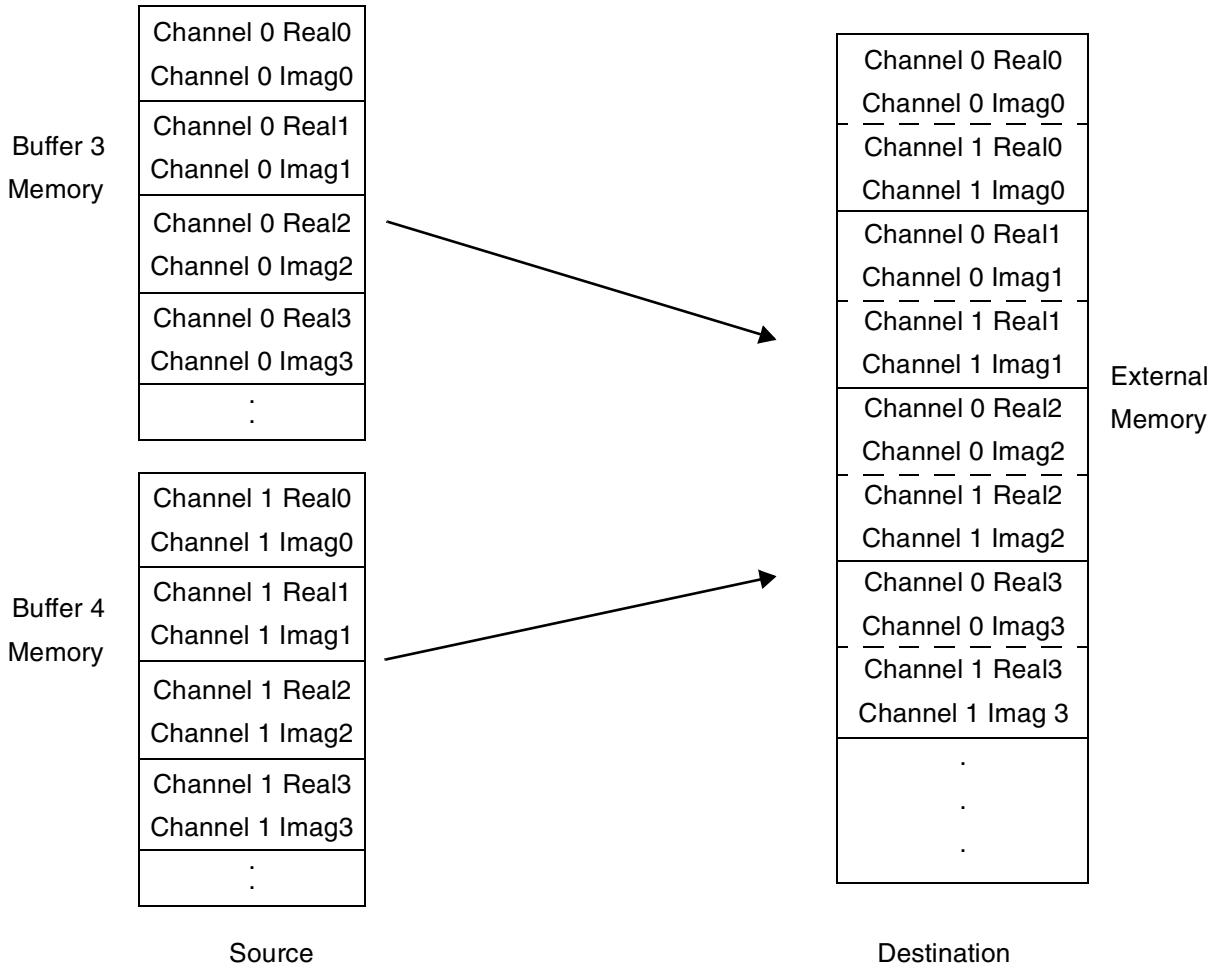

Figure 3. Transfer of Two Separate Buffers into a Combined Buffer

2 DMA Data Transfer Basics

The MSC8101 16-channel DMA controller enables data transfer between memory and peripherals with limited intervention of the SC140 core, freeing the SC140 core to perform other processes. The DMA controller is triggered by requests from internal peripherals, external peripherals, or the DMA FIFO.

The DMA controller can perform two types of transactions: normal (dual access) and flyby (single access) modes. In normal mode, data is transferred from the source to the DMA FIFO and from the DMA FIFO to the destination. The DMA FIFO connects to the 60x-compatible system bus and the local bus. If data is transferred between memory and a peripheral on the same bus and with the same port size, then access to the DMA FIFO is not needed, so a DMA transaction can occur in flyby mode. All other types of transfers are dual-access transactions.

Taking advantage of various DMA features, data can be transferred efficiently between memory and the EFCOP. Thus, the DMA controller can transfer data from external memory to two separate memory buffers (acting as two-dimensional mode) and then from the memory buffer to the EFCOP Filter Data Input Register (FDIR). After the EFCOP and the SC140 core finish processing the data, the DMA controller can transfer the results from the EFCOP Filter Data Output Register (FDOR) to a memory buffer and then from the memory buffer to external SDRAM, if desired.

When the buffer size in the chained simple buffers reaches zero, the same DMA channel triggers another transfer. The second transfer begins at the address to which the second buffer points. There is no limit to the number of DMA buffers chained together. An example of how two chained simple buffers work in a two-dimensional DMA data transaction is shown in **Figure 4**. The transfer size of buffer 1 and buffer 2 is two bytes. After bytes 0 and 1 from buffer 1 are read, bytes 0 and 1 from buffer 2 are read. Next, another two bytes from buffer 1 are read, followed by two bytes from buffer 2. This process continues until the DMA channel is no longer active.

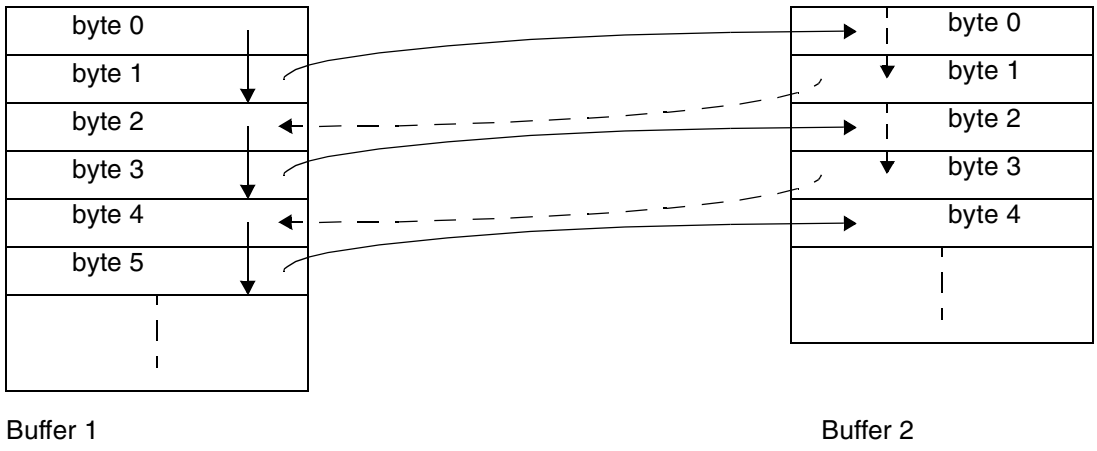

Figure 4. Chained DMA Buffers

3 Programming Registers

When two-dimensional DMA transfers are implemented on the MSC8101, there are two key things to remember:

- A dual-access DMA transfer is always required.
- One buffer descriptor is required for each buffer on the side of the FIFO where the multiple buffers are found (source or destination side).

3.1 Case 1

Buffer 1 requires a single source buffer to be transferred to two destination buffers (see **Figure 5**). The dual buffers are on the destination side of the DMA FIFO, so the source side may perform a single transfer to the FIFO. However, the destination side must break the transaction into two transfers, one to handle each buffer on the destination side.

The source side of the transfer moves the data to the DMA FIFO. The destination side begins removing data from the FIFO as soon as it detects that the FIFO is not empty. Two buffer descriptors are chained together on the destination side, and they work together until the FIFO is empty. Each buffer descriptor is programmed with the buffer attributes of its assigned destination. Once BD2 has transferred its designated size of data to its designated address, it transfers ownership of the DMA channel to BD3. BD3 then continues with the same DMA channel (and thus the same DMA FIFO buffer) and transfers data to its designated address for the number of pieces of data specified in its BD_SIZE field. This process continues until the entire buffer is transferred and the FIFO is empty.

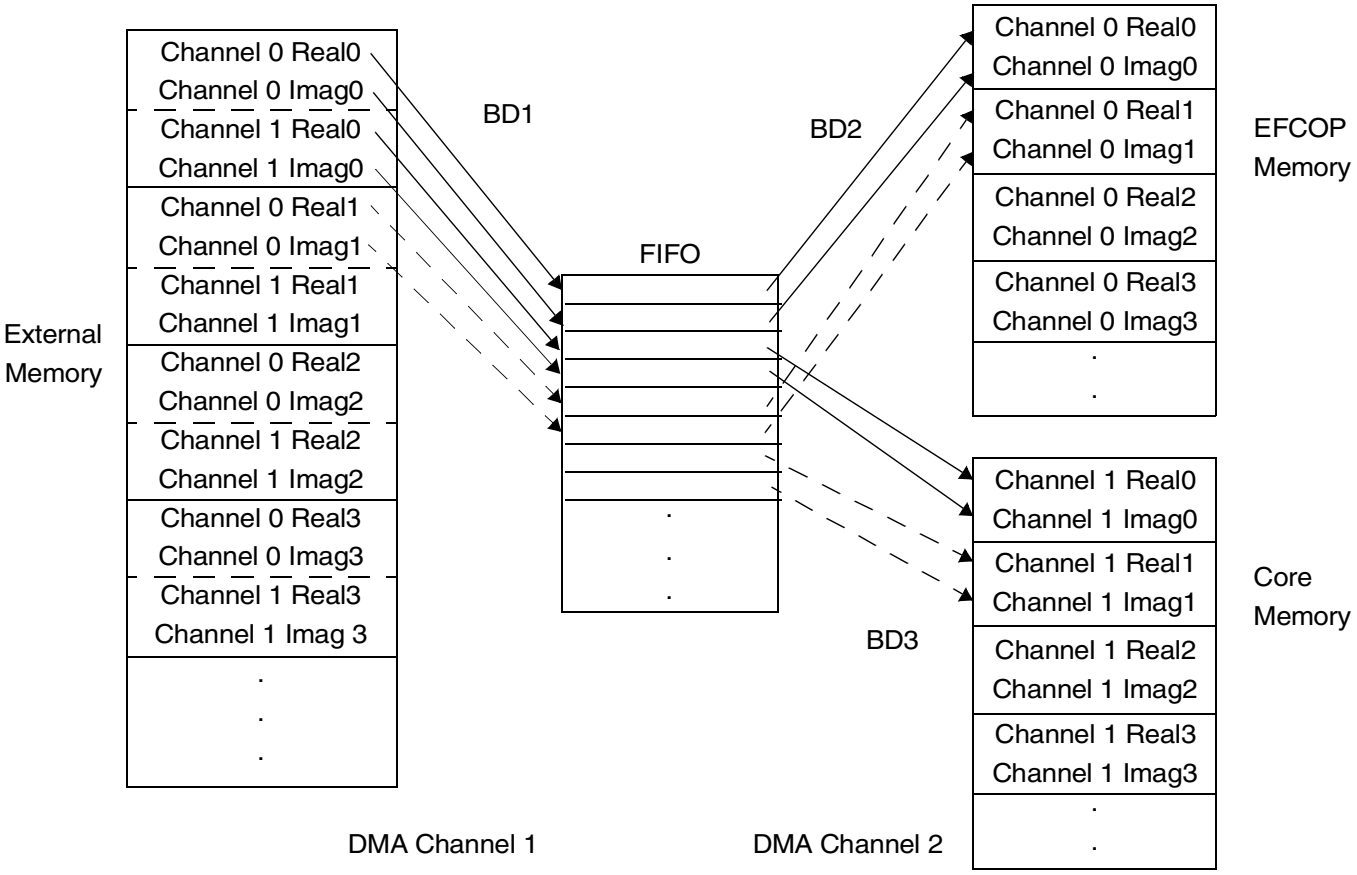

Figure 5. Buffer 1 Example with DMA Implementation

The steps in setting up the DMA registers for Case 1 are as follows:

1. The buffer descriptor attributes are initialized for the source-side transfer. This transfer uses Buffer Descriptor 0 attributes:
   - **a.** BD_ADDR0 = 0x20000000. External SDRAM source address.
   - **b.** BD_ATTR0 = 0x80000190. Generate interrupt, not continuous, 32-bit transfers, read transaction
   - **c.** BD_SIZE0 = 0x800. Transfer 2048 bytes.
2. The buffer descriptors for the destination-side are initialized.

   Since there are two buffers on the destination side, BD8 and BD9 are used. BD8 describes the EFCOP memory buffer, and BD9 describes the SC140 core memory buffer.
   - **a.** BD_ADDR8 = 0x02070000. EFCOP filter memory address on local bus.
   - **b.** BD_ATTR8 = 0x20090180. Continuous, chained to BD9, 32-bit transfers, write transaction.
   - **c.** BD_SIZE8 = 0x8. Transfer 8 bytes at a time.
   - **d.** BD_BSIZE8 = 0x8. Reset SIZE to 8 bytes when transaction complete.
   - **e.** BD_ADDR9 = 0x02060000. Data buffer in memory for core to process.
   - **f.** BD_ATTR9 = 0x20080180. Continuous, chained to BD8, 32-bit transfers, write transaction.
   - **g.** BD_SIZE9 = 0x8. Transfer 8 bytes at a time.
   - **h.** BD_BSIZE9 = 0x8. Reset SIZE to 8 bytes when transaction complete.

3. The memory buffers are initialized to a known state. The SC140 core and EFCOP memory buffers are cleared. The SDRAM buffer is initialized to the required data values.
4. The DSTR register is cleared, which clears any pending DMA status bits.
5. DMA channel 2 handles the source-side transaction and DMA channel 3 handles the destination-side transaction.
   a. DCHCR2 = 0xC0000048. Active DMA channel, system bus transaction, BD0, initiated by DMA, priority level 8.
   b. DCHCR3 = 0x80080048. Active DMA channel, local bus transaction, BD8, initiated by DMA, priority level 8.

Example 1. Buffer 1

```
#include <stdio.h>
#include "typedefs.h"
#include "msc8101.h"

#define SIU_BASE    0x14700000   /* 8101 PowerPC Bus Memory Map base     */
#define SDRAM_DATA  0x20000000   /* All channel data in external buffer  */
#define CORE_DATA   0x02060000   /* Filter data buffer for core          */
#define EFCOP_DATA  0x02070000   /* Filter data buffer for EFCOP         */
#define EFCOP_VAL1  0xAAAA5555   /* Channel 0 Real initialization value  */
#define EFCOP_VAL2  0xCCCC3333   /* Channel 0 Imag initialization value  */
#define CORE_VAL1   0xD2D2D2D2   /* Channel 1 Real initialization value  */
#define CORE_VAL2   0x1E1E1E1E   /* Channel 1 Imag initialization value  */
#define INTERNAL    0x00000000   /* Internal Memory initialization value */

t_8101IMM *IMM;

int main(void)
{
long *temp_ptr1;
int i;

       IMM =  (t_8101IMM *)(SIU_BASE);           /* MSC8101 int reg map     */

       //INIT_DMA1
       IMM->dma_dcpram[0].bd_addr = SDRAM_DATA;
       IMM->dma_dcpram[0].bd_attr = 0x80000190;  /* issue interrupt, non-
continuous,
       max xfer 32 bits,read */
       IMM->dma_dcpram[0].bd_size = 0x800;      /* transfer 2048 bytes       */


//INIT_DMA2
IMM->dma_dcpram[8].bd_addr = EFCOP_DATA;
IMM->dma_dcpram[8].bd_attr = 0x20090180;  /* max xfer 32 bits, write     */
IMM->dma_dcpram[8].bd_size = 0x8;         /* transfer 8 bytes at a time */
IMM->dma_dcpram[8].bd_bsize = 0x8;        /* transfer 8 bytes at a time */

IMM->dma_dcpram[9].bd_addr = CORE_DATA;
IMM->dma_dcpram[9].bd_attr = 0x20080180;  /* max xfer 32 bits, write    */
IMM->dma_dcpram[9].bd_size = 0x8;         /* transfer 8 bytes at a time */
IMM->dma_dcpram[9].bd_bsize = 0x8;        /* transfer 8 bytes at a time */
```

```
// init internal memory buffers
temp_ptr1 = (long *) CORE_DATA;
for (i=0; i<512; i++) {
        *temp_ptr1=INTERNAL;
        temp_ptr1++;
}
temp_ptr1 = (long *) EFCOP_DATA;
for (i=0; i<512; i++) {
        *temp_ptr1=INTERNAL;
        temp_ptr1++;
}
//INIT_SDRAM_DATA
temp_ptr1 = (long *) SDRAM_DATA;
for (i=0; i<512;i++) {
        *temp_ptr1 = EFCOP_VAL1;
        temp_ptr1++;
        *temp_ptr1 = EFCOP_VAL2;
        temp_ptr1++;
        *temp_ptr1 = CORE_VAL1;
        temp_ptr1++;
        *temp_ptr1 = CORE_VAL2;
        temp_ptr1++;
}

        //DMA_XFER
//Start DMA xfer from external mem to FIFO
        IMM->dma_dstr = 0x30000000; /* Clear DSTR bits by writing a 1    */
        IMM->dchcr[2] = 0xC0000048; /* activate,BD0,60x,int req,prio 8   */
        IMM->dchcr[3] = 0x80080048; /* activate,BD8,local,int req,prio 8 */

        while(1);
        // Stay here until the interrupt
        //while ((IMM->dma_dstr & 0x20000000) == 0) {};
        //while ((IMM->dma_dstr & 0x30000000) == 0) {};

        //END
        asm("debug");
}
```

3.2 Case 2

Once filter processing is complete, the results from the EFCOP and the SC140 core must be organized within a single buffer. Therefore, the DMA controller must transfer data from two source memory buffers in internal SRAM into one destination buffer in external memory (see **Figure 5**). On the source side, the transaction is two different transfers into the DMA FIFO, and on the destination side the transaction is a single transfer from the DMA FIFO.

Two buffer descriptors are chained together on the source side. Data from the two buffer descriptors are transferred to the DMA FIFO. BD14 is programmed with the attributes of the source memory buffer and has control of DMA channel 4. After BD14 has transferred its programmed size, BD15 has control of DMA channel 4. After BD15 has transferred its programmed size, control of the DMA is returned to BD14. This process continues until the DMA channel is no longer enabled. As soon as the DMA FIFO is not empty, the destination side removes data until it has received the number of programmed bytes.

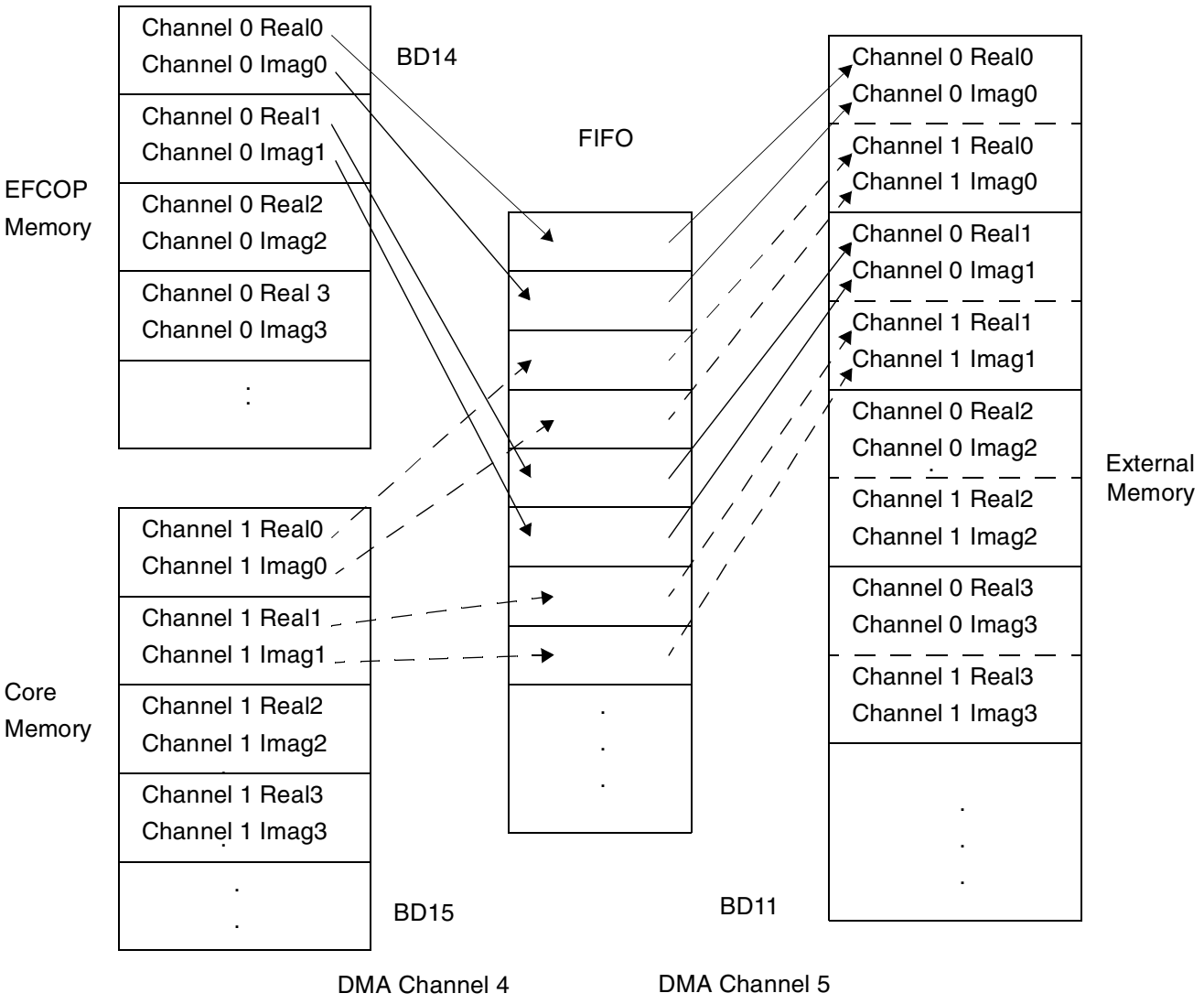

Figure 6. Buffer 2 With DMA Implementation

The steps in setting up the DMA registers for Case 2 are as follows:

1. Initialize the buffer descriptors for the source side of the transfer.

   Since the source side consists of two buffers, BD14 and BD15 are used. The EFCOP memory buffer is described by buffer descriptor 14, and the SC140 core memory buffer is described by BD15.

   a. BD_ADDR14 = 0x02070000. EFCOP filter memory address on local bus.

   b. BD_ATTR14 = 0x200F00190. Continuous, chained to BD15, 32-bit transfers, read transaction.

   c. BD_SIZE14 = 0x8. Transfer 8 bytes at a time.

   d. BD_BSIZE14 = 0x8. Reset SIZE to 8 bytes when transaction complete.

   e. BD_ADDR15 = 0x02060000. Data buffer in memory for core to process.

   f. BD_ATTR15 = 0x200E0190. Continuous, chained to BD14, 32-bit transfers, read transaction.

   g. BD_SIZE15 = 0x8. Transfer 8 bytes at a time.

   h. BD_BSIZE15 = 0x8. Reset SIZE to 8 bytes when transaction complete.

2. The code initializes the buffer descriptor for the destination side of the transfer. This transfer uses BD11.
   a. BD_ADDR11 = 0x20001000. External SDRAM destination address.
   b. BD_ATTR11 = 0x80000180. Generate interrupt when complete, not continuous, 32-bit transfers, write transaction.
   c. BD_SIZE11 = 0x800. Transfer 2048 bytes.
3. The DSTR register is cleared. This clears any pending DMA status bits.
4. DMA4 is used for the source-side transaction, and DMA5 is used for the destination-side transaction.
   a. DCHCR4 = 0x800E0048. Active DMA channel, local bus transaction, BD14, initiated by DMA, priority level 8.
   b. DCHCR5 = 0xC00B0048. Active DMA channel, system bus transaction, BD11, initiated by DMA, priority level 8.

Example 2. Buffer 2

```
#include <stdio.h>
#include "typedefs.h"
#include "msc8101.h"

#define SIU_BASE    0x14700000   /* 8101 PowerPC Bus Memory Map base     */
#define SDRAM_DATA  0x20001000   /* All channel data in external buffer  */
#define CORE_DATA   0x02060000   /* Filter data buffer for core          */
#define EFCOP_DATA  0x02070000   /* Filter data buffer for EFCOP         */
#define INTERNAL    0x00000000   /* Internal Memory initialization value */

t_8101IMM *IMM;

int main(void)

{
long *temp_ptr1;
int i;

       IMM =  (t_8101IMM *)(SIU_BASE);           /* MSC8101 int reg map     */

//INIT_DMA4
       IMM->dma_dcpram[14].bd_addr = EFCOP_DATA;
       IMM->dma_dcpram[14].bd_attr = 0x200F0190;  /* max xfer 32 bits, write
*/
       IMM->dma_dcpram[14].bd_size = 0x8;         /* transfer 8 bytes at a time
*/
       IMM->dma_dcpram[14].bd_bsize = 0x8;        /* transfer 8 bytes at a time
*/

       IMM->dma_dcpram[15].bd_addr = CORE_DATA;
       IMM->dma_dcpram[15].bd_attr = 0x200E0190;  /* max xfer 32 bits, write
*/
       IMM->dma_dcpram[15].bd_size = 0x8;         /* transfer 8 bytes at a time
*/
       IMM->dma_dcpram[15].bd_bsize = 0x8;        /* transfer 8 bytes at a time
*/
```

```
//INIT_DMA5
        IMM->dma_dcpram[11].bd_addr = SDRAM_DATA;
        IMM->dma_dcpram[11].bd_attr = 0x80000180;  /* issue interrupt, non-
continuous,
                                                    max xfer 32 bits,read */
        IMM->dma_dcpram[11].bd_size = 0x800;       /* transfer 2048 bytes
*/

//DMA_XFER
//Start DMA xfer from memory buffers to DMA FIFO to external memory
        IMM->dma_dstr = 0x0C000000; /* Clear DSTR bits by writing a 1    */
        IMM->dchcr[4] = 0x800E0048; /* activate,BD14,local,int req,prio 8   */
        IMM->dchcr[5] = 0xC00B0048; /* activate,BD11,60x,int req,prio 8 */

        while(1);
}
```

3.3 DMA Completion

There are two ways to determine when a DMA transfer completes: polling the DMA status register or generating a DMA complete interrupt. Unfortunately, in the buffer 1 example in which the destination side of the transfer has multiple buffers, it is difficult to determine when a transfer is complete and the interrupt should be generated. Normally, the interrupt is generated when BD_SIZE reaches 0. In the buffer 1 example, when BD_SIZE reaches 0 for the second channel (channel 3, BD8) the transfer is not complete. Instead, it is chained to a second buffer (channel 3, BD9) to complete, which is then chained back to the first buffer.

One way to determine when a transfer is complete is to generate an interrupt when the source-side transfer is complete (in this case, channel 2) and then wait some time for the destination-side to complete. Unfortunately, while this method works well in a system with few bus loads, it may not work in a fully-loaded system. Once the source-side has transferred all data to the DMA FIFO, loading on the local bus may cause the DMA FIFO to be almost completely full and still have quite a few transactions remaining before it is flushed.

The best way to ensure that all data transfers are complete is to generate an interrupt at the end of each destination-side final buffer (for example 1, each time BD9 size reaches 0). In the interrupt routine, increment a counter and check to see if the entire transfer has completed. You can thus be sure to receive all intended data.

The buffer 2 example does not have the same problem. The source side has multiple buffers that are chained to each other, making it difficult to know when the transfer is complete. However, on the destination side, the transfer size is known. When the channel closes, the DMA channel on the source side can close as well and an interrupt can be generated. The interrupt is reached only once, when the entire transfer is complete.

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 München, 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

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

AN2281
Rev. 1
11/2004