



## ADSP-BF70x Blackfin+™ Processor System Optimization Techniques

Contributed by Akash Agarwal, Mitesh Moonat, and Nabeel Shah

Rev 1 – January 5, 2016

### Introduction

The ADSP-BF70x family of Blackfin+™ processors (hereafter referred to as ADSP-BF707 processors) provides an optimized architecture supporting high system bandwidth and advanced peripherals. The purpose of this EE-Note is to discuss the processor's key architectural features affecting the overall system bandwidth and provide various bandwidth optimization techniques.

The ADSP-BF707 Blackfin+ processor architecture is similar in many ways to the ADSP-BF60x Blackfin® processor architecture and other older Blackfin processors. Thus, some of the system optimization techniques discussed in *System Optimization Techniques for Blackfin® Processors (EE-324)*<sup>[1]</sup> and *ADSP-BF60x Blackfin® Processor System Optimization Techniques (EE-362)*<sup>[2]</sup> are also applicable for these processors.

### ADSP-BF707 Processor Architecture

This section provides a holistic view of the ADSP-BF707 processor architectural features that play a crucial role in terms of system bandwidth and performance. For detailed information about these features, please refer to the *ADSP-BF70x Processor Hardware Reference Manual*<sup>[3]</sup>.

#### System Bus Slaves

The system bus slaves are shown on the right side in [Figure 1](#). They include on-chip and off-chip memory devices/controllers, such as L1 SRAM, L2 SRAM, the Dynamic Memory Controller (DMC) for DDR2 and LPDDR SDRAM devices, the Static Memory Controller (SMC) for SRAM and flash devices, SPI flash, the One-Time Programmable Memory Controller (OTPC), and the System Memory-Mapped Registers (MMRs).

Each system bus slave has its own latency characteristics, operating in a given clock domain. For example, L1 SRAM access is faster than L2 SRAM access, which is in turn faster than DMC and SMC memory accesses.

#### System Bus Masters

The system bus masters are shown on the left side in [Figure 1](#). They include peripheral Direct Memory Access (DMA) channels, such as the Enhanced Parallel Peripheral Interface (EPPI) and Serial Port (SPORT), among others. Also included are the Memory-to-Memory DMA channels (MDMA) and the core.

Each peripheral runs at a different clock speed, so each has individual bandwidth requirements. For example, parallel peripherals like the EPPI require higher bandwidth as compared to the slower serial peripherals (e.g., SPORT, SPI, TWI, etc.).

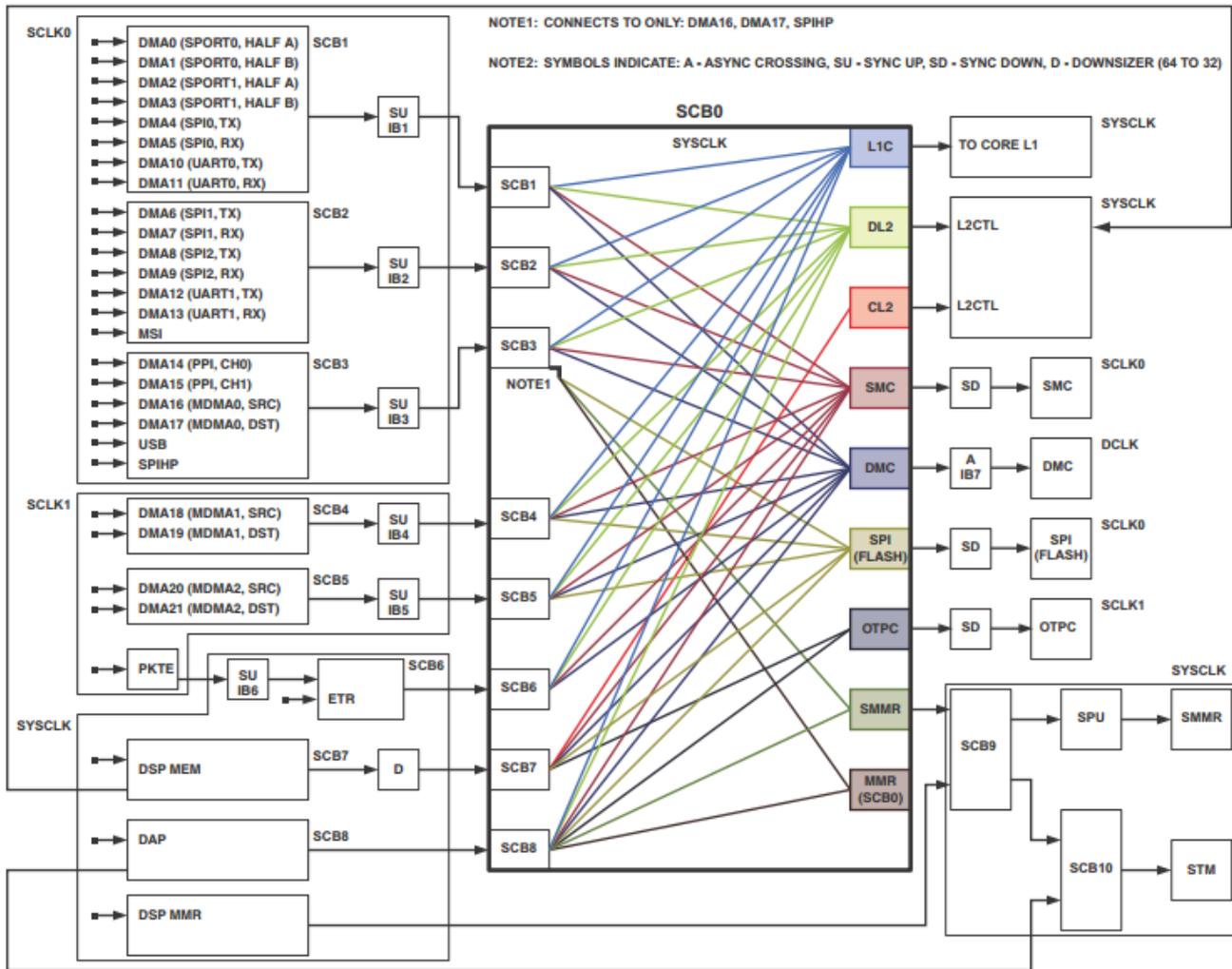


Figure 1. ADSP-BF707 Processor System Architecture

## System Crossbars

The System Crossbars (SCB) are the fundamental building blocks of the system bus interconnect. As shown in Figure 1, the SCB interconnect is built from multiple SCBs in a hierarchical model connecting system bus masters to system bus slaves. They allow concurrent data transfer between multiple bus masters and multiple bus slaves, providing flexibility and full-duplex operation. The SCBs also provide a programmable arbitration model for bandwidth and latency management. Also, as highlighted earlier, SCBs run on different clock domains (SCLK0, SCLK1, and SYSCLK), which introduces unique latencies to the system.

## System Latencies, Throughput, and Optimization Techniques

### Understanding the System Masters

#### *DMA Parameters*

Each DMA channel has two buses:

- one that connects to the SCB, which in turn is connected to the SCB slave (e.g., memories)
- one that connects to either a peripheral or another DMA channel

The SCB/memory bus width can vary between 8-, 16-, 32-, or 64-bit, as defined by the `DMA_STAT.MBWID` bit field, while the peripheral bus width is further configurable to 128-bit, as defined by the `DMA_STAT.PBWID` bit field. For ADSP-BF707 processors, the memory and peripheral bus widths are fixed to 32-bit (4-byte).

The `DMA_CFG.PSIZE` DMA parameter determines the width of the peripheral bus being used. It can be configured to 1-, 2-, 4-, or 8-byte. However, it cannot exceed the maximum possible bus width defined by `DMA_STAT.PBWID` because burst transactions are not supported on the peripheral bus.

The `DMA_CFG.MSIZE` DMA parameter determines the actual size of the SCB bus being used. It also determines the minimum number of bytes which will be transferred from/to memory corresponding to a single DMA request/grant. It can be configured to 1-, 2-, 4-, 8-, 16-, or 32-byte. If `DMA_CFG.MSIZE` is greater than `DMA_STAT.MBWID`, the SCB performs burst transfers with the width defined in `DMA_CFG.MSIZE`. Thus, it is important to understand how to choose the appropriate `DMA_CFG.MSIZE` value, from both a functionality and a performance perspective. The following points should be considered when setting `DMA_CFG.MSIZE`:

- The start address of the work unit should always align to the `DMA_CFG.MSIZE` value selected. Failing to do so will generate a DMA error interrupt.
- As a general rule, from a performance/throughput perspective, the largest possible `DMA_CFG.MSIZE` value (32-byte) should be used for better average throughput. This is because it results in a higher likelihood of uninterrupted sequential accesses to the slave (memory), which is most efficient for typical memory designs.
- From a performance perspective, in some cases, the minimum `DMA_CFG.MSIZE` value is determined by the burst length supported by the memory device. For example, for DDR2 accesses, the minimum `DMA_CFG.MSIZE` value is limited by the DDR2 burst length (4- or 8-byte). Any `DMA_CFG.MSIZE` value below this would lead to a significant throughput loss. For more details, please refer to the [L3/External Memory Throughput](#) section.

#### *Bandwidth Limiting and Monitoring*

MDMA channels are equipped with a bandwidth limit and monitor mechanism. The bandwidth limit feature can be used to reduce the number of DMA requests being sent by the corresponding masters to the SCB. The `DMA_BWLCNT` register can be programmed to configure the number of `SCLK` cycles between two DMA requests. This can be used to make sure that such DMA channels' requests do not occur more frequently than required. Programming a value of 0x0000 allows the DMA to request as often as possible. A value of 0xFFFF represents a special case and causes all requests to stop. The maximum throughput (in MB/s) is a function of `SCLK` (in MHz) and can be limited by the `DMA_BWLCNT` value and the `DMA_CFG.MSIZE` value (in bytes), calculated as follows:

$$\text{Bandwidth} = \min ((\text{SCLK} * 4), (\text{SCLK} * \text{DMA\_CFG.MSIZE} / \text{DMC\_BWLCNT}))$$

The example code provided in the *associated ZIP file* <sup>[4]</sup> utilizes a MDMA stream to perform data transfers from DDR2 to L1 memory. In this example, `DMC_BWLCNT` is programmed to different values to limit the throughput to an expected value for different `DMA_CFG.MSIZE` values. **Table 1** provides a summary of the theoretical and measured throughput values. As shown, the measured throughput is very close to the theoretical bandwidth, which it never exceeds. The `SCLK1` frequency used for testing was 200 MHz, and the DMA buffer size was 16384 words.

Sample No.	MSIZE	BWLCNT	Theoretical Throughput (MB/s) (SCLK * MSIZE / BWLCNT)	Measured Throughput (MB/s)
1	32	8	800	695
2	32	16	400	372
3	32	32	200	192
4	32	64	100	98
5	32	128	50	49
6	16	8	400	351
7	16	16	200	187
8	16	32	100	96
9	16	64	50	49
10	16	128	25	24
11	8	8	200	176
12	8	16	100	93
13	8	32	50	48
14	8	64	25	24
15	8	128	12.5	12

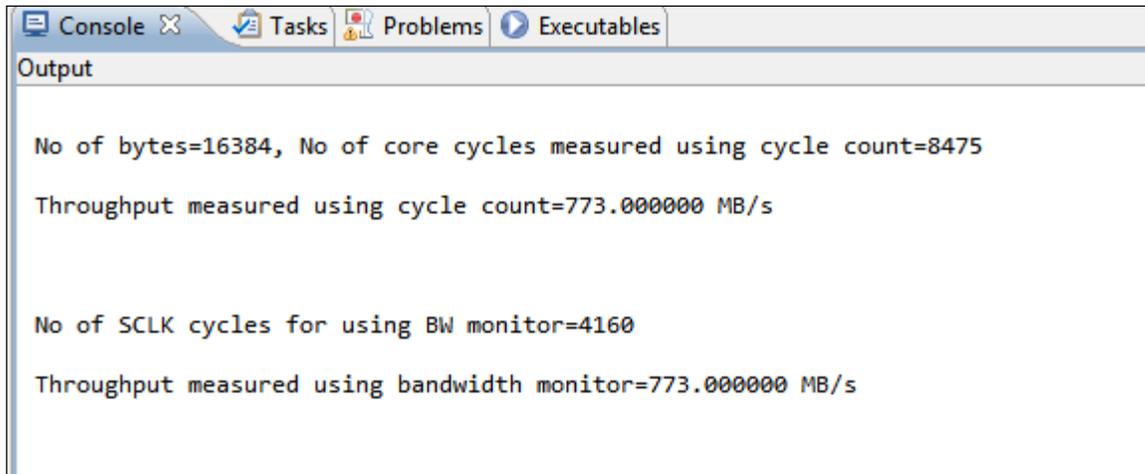
*Table 1. DMC MDMA Read Throughput at Different DMC\_BWLCNT Settings*

The bandwidth monitor feature can also be used to check for starving DMA channels. The `DMC_BMCNT` register can be programmed to the number of `SCLK` cycles within which the corresponding DMA should finish. Each time the `DMA_CFG` register is written to via an MMR access, a work unit ends, or an autobuffer wraps, the DMA loads the value from `DMA_BWMCNT` into `DMA_BWMCNT_CUR`. The DMA decrements `DMA_BWMCNT_CUR` every `SCLK` a work unit is active. If `DMA_BWMCNT_CUR` reaches 0x00000000 before the work unit finishes, the `DMA_STAT.IRQERR` bit is set, and the `DMA_STAT.ERRC` is set to 0x6. The `DMA_BWMCNT_CUR` remains at 0x00000000 until it is reloaded when the work unit completes.

Unlike other error sources, a bandwidth monitor error does not stop work unit processing. Programming 0x00000000 disables bandwidth monitor functionality. This feature can also be used to measure the actual throughput.

Example code is also provided to help explain this functionality. The example uses MDMA streams 18-19, configured to read from DDR2 memory to L1 memory. For an expected throughput of 400 MB/s, the `DMC_BWMCNT` register must be programmed to 8192 ( $\text{Buffer Size} * \text{SCLK} / \text{Bandwidth} = 16384 * 200 / 400$ ).

When this MDMA runs alone, the measured throughput (using both cycle count and `DMC_BWMCNT`) is as expected (773 MB/s), as shown in [Figure 2](#).



```

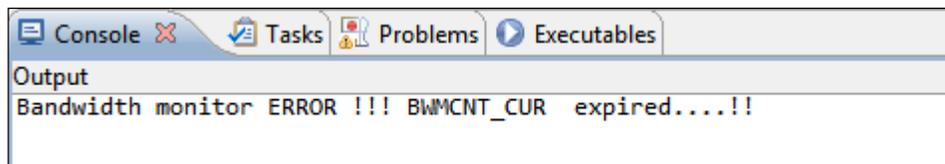
Console X Tasks Problems Executables
Output
No of bytes=16384, No of core cycles measured using cycle count=8475
Throughput measured using cycle count=773.000000 MB/s

No of SCLK cycles for using BW monitor=4160
Throughput measured using bandwidth monitor=773.000000 MB/s

```

*Figure 2. Using `DMC_BWMCNT` for Measuring Throughput*

Using the same example code, if another DMC read MDMA stream is initiated in parallel (`#define ADD_ANOTHER_MDMA`), the throughput drops to less than 400 MB/s, and a bandwidth monitor error is generated (as shown in [Figure 3](#)).



```

Console X Tasks Problems Executables
Output
Bandwidth monitor ERROR !!! BWMCNT_CUR expired....!!

```

*Figure 3. Bandwidth Monitor Expires Due to Less than Expected Throughput*

`SCLK0` runs up to `SYSCLK` and is the clock source for the MDMA1 and MDMA2 channels. Meanwhile, `SCLK1` runs up to `SYSCLK/2` and is the clock source for the MDMA0 channel. As such, using the MDMA1 and MDMA2 streams instead of the MDMA0 stream is recommended for higher efficiency.

### Understanding the System Crossbars

As shown in [Figure 1](#), the SCB interconnect consists of a hierarchical model connecting multiple SCB units. [Figure 4](#) shows the block diagram for a single SCB unit. It connects the system bus masters (M) to the system bus slaves (S) by using a Slave Interface (SI) and a Master Interface (MI). On each SCB unit, each S is connected to a fixed MI, and each M is connected to a fixed SI.

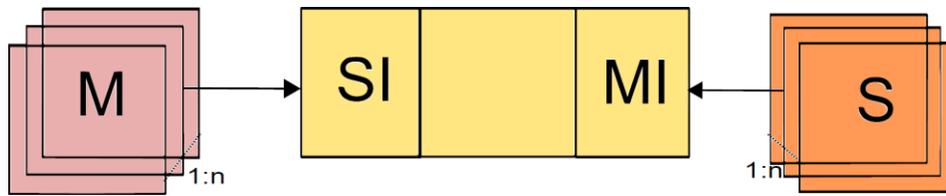


Figure 4. Single SCB Block Diagram

The slave interface of the crossbar (where the masters such as DDE connect to) performs two functions.

- **Arbitration:** the programmable Quality of Service (QoS) registers can be viewed as being associated with SCBx. For example, the QoS registers for DDE14-17, USB, and SPIHP can be viewed as residing in SCB3. Whenever a transaction is received at DDE14, the programmed QoS value is associated with that transaction and is arbitrated with the rest of the masters at SCB3.
- **Clock Domain Conversion:** IB1–3 perform clock domain crossing from `SCLK0` to `SYSCLK`. `SCLK0` is always assumed to be an integer ratio (1:n) to `SYSCLK` and passes through a one-stage synchronizer. Therefore, there are no programmable registers for IB1–3. IB4–6 perform clock domain crossing from `SCLK1` to `SYSCLK`. The clock domain crossing from `SCLK1` to `SYSCLK` is programmable via the sync mode register associated with IB4, IB5, and IB6. Similarly, the IB7 preset at the DDR boundary performs clock domain crossing from `SYSCLK` to `DCLK`. This crossing is also programmable via the sync mode register in IB7. Synchronization across clock domains affects the SCB performance. Interface blocks IB4 (MDMA1), IB5 (MDMA2), IB6 (Crypto), and IB7 (DDR) should be programmed depending upon the clock ratios of the two clock domains.

#### Programming the SCB QoS Registers

Consider a scenario where:

- At SCB1, masters  $M_1$ ,  $M_2$ , and  $M_3$  have RQoS values of 6, 4, and 2, respectively.
- At SCB2, masters  $M_4$ ,  $M_5$ , and  $M_6$  have RQoS values of 12, 13, and 1, respectively.

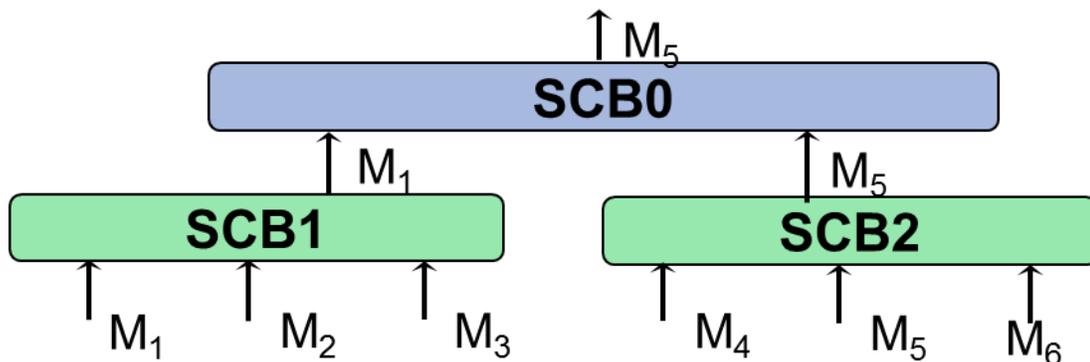


Figure 5 Arbitration among Various Masters

In this case:

- $M_1$  wins at SCB1, and  $M_5$  wins at SCB2, as each has the largest  $RQoS$  value for its respective group.
- In a perfect competition at SCB0,  $M_4$  and  $M_5$  would have the highest overall  $RQoS$  values and would fight for arbitration directly at SCB0; however, due to the mini-SCBs,  $M_1$  is able to win against  $M_4$  and make it all the way to SCB0, despite the much lower  $RQoS$  value.

*Programming Sync Mode in the IBx Registers*

To illustrate the effect of using the sync mode when programming the  $IB_x$  registers, DMC read throughput was measured using the MDMA1 stream with the core clock (CCLK) set to 400 MHz and SCLK1, SYSCLK, and DCLK each set to 200 MHz. [Figure 6](#) shows the measured throughput for the following four cases:

1.  $IB_4$  and  $IB_7$  in ASYNC mode
2.  $IB_4$  in SYNC(1:1) mode, and  $IB_7$  in ASYNC mode
3.  $IB_4$  in ASYNC mode, and  $IB_7$  in SYNC(1:1) mode
4.  $IB_4$  and  $IB_7$  in SYNC(1:1) mode

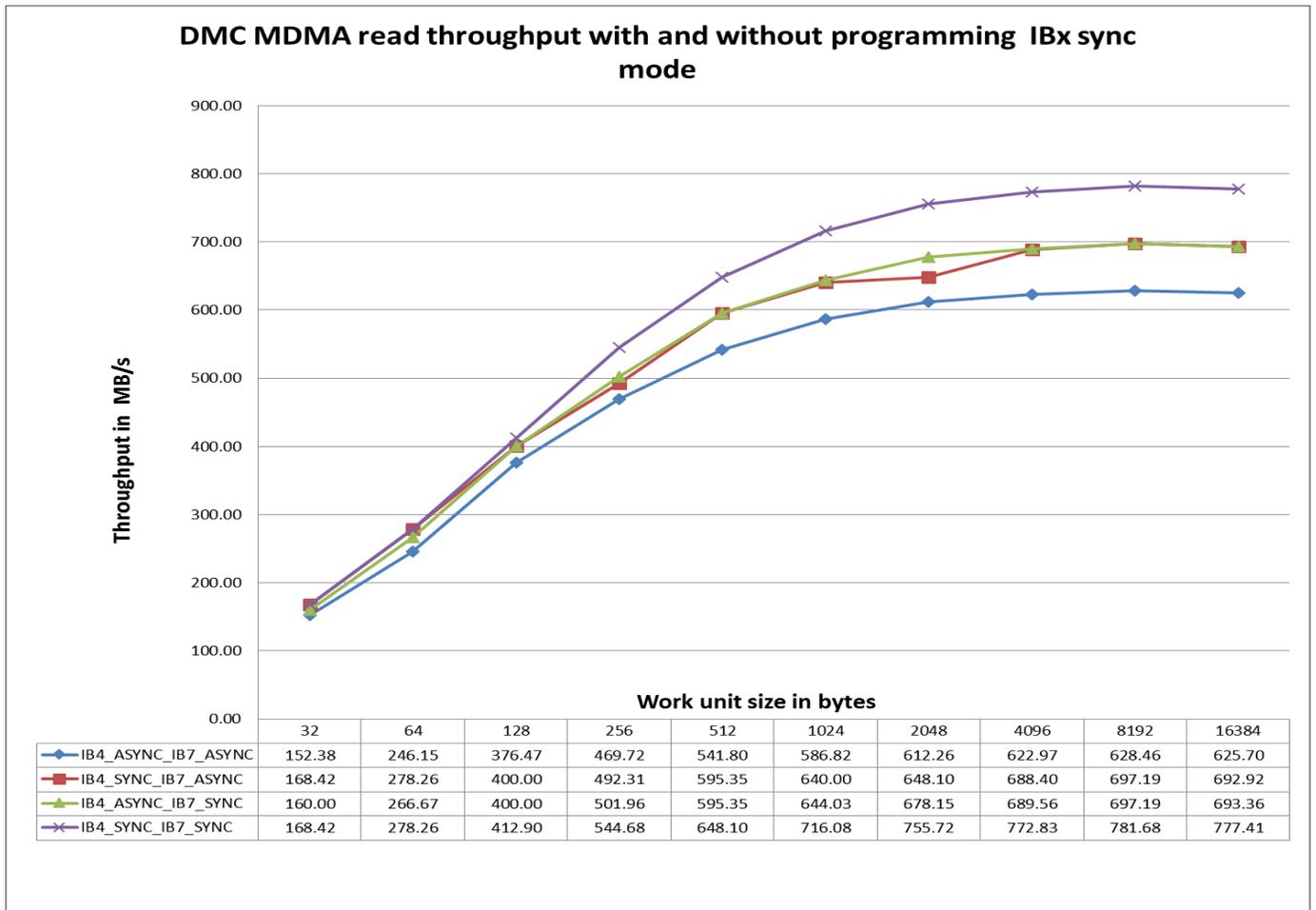


Figure 6. DMC MDMA Throughput with and without IBx Sync Mode

As can be seen, the throughput is best when both IB4 and IB7 are programmed in SYNC mode, and it is worst when both are programmed in ASYNC mode.

## Understanding the System Slaves

### Memory Hierarchy

ADSP-BF707 processors contain a hierarchical memory model (L1-L2-L3) similar to that of previous Blackfin devices. Thus, the memory-related optimization techniques discussed in *System Optimization Techniques for Blackfin® Processors (EE-324)* apply also to ADSP-BF707 processors. The following sections discuss access latencies and achievable throughput associated with each memory level.

L1 memory runs at the core clock (CCLK) speed and is the fastest accessible memory in the hierarchy. L2 memory access times are longer, as the maximum L2 clock frequency is the system clock (SYSCLK), which operates at CCLK/2. The L2 memory controller contains two ports to connect to the system crossbar. Port 0 is a 64-bit interface dedicated to core traffic, while port 1 is a 32-bit interface that connects to the DMA engine. Each port has a read and a write channel. For more details, refer to the *ADSP-BF70x Processor Hardware Reference*.

### L2 Memory Throughput

Since L2 runs at SYSCLK, it's capable of providing a maximum theoretical throughput of  $200 \text{ MHz} * 4 = 800 \text{ MB/s}$  in one direction. Since there are separate read and write channels, the total throughput in both directions is 1600 MB/s. In order to be able to operate L2 SRAM memory at its optimum throughput, both the core and DMA ports and separate read and write channels should be used in parallel.

All accesses to L2 memory are converted to 64-bit accesses (8-byte) by the L2 memory controller. Thus, in order to achieve optimum throughput for DMA access to L2 memory, the DMA channel's `DMA_CFG.MSIZE` field should be configured to 8 bytes or more.

Unlike L3 (DMC) memory accesses, L2 memory throughput for sequential and non-sequential accesses is the same.

ADSP-BF707 processors feature up to 1 MB of L2 SRAM, which is ECC-protected and organized into eight banks. A single 8- or 16-bit access, or a non-32-bit address-aligned 8-bit or 16-bit burst access to an ECC-enabled bank creates an additional latency of two SYSCLK cycles. This is because the ECC implementation is on the order of 32-bit, meaning that any writes that are less than 32-bit to an ECC-enabled SRAM bank will be implemented as a read-followed-by-write and will require three cycles to complete (two cycles for the read, and one cycle for the write).

When performing simultaneous core and DMA accesses to the same L2 memory bank, read and write priority control registers can be used to increase DMA throughput. If both the core and DMA access the same bank, the best access rate that the DMA can achieve is one 64-bit access every three SYSCLK cycles during the conflict period. This is achieved by programming the read and write priority count bits (`L2CTL_RPCR.RPC0` and `L2CTL_WPCR.WPC0`) to 0, while programming the `L2CTL_RPCR.RPC1` and `L2CTL_WPCR.WPC1` bits to 1.

Figure 7 shows the measured MDMA throughput for the case where the source and destination buffers are in different L2 memory banks.

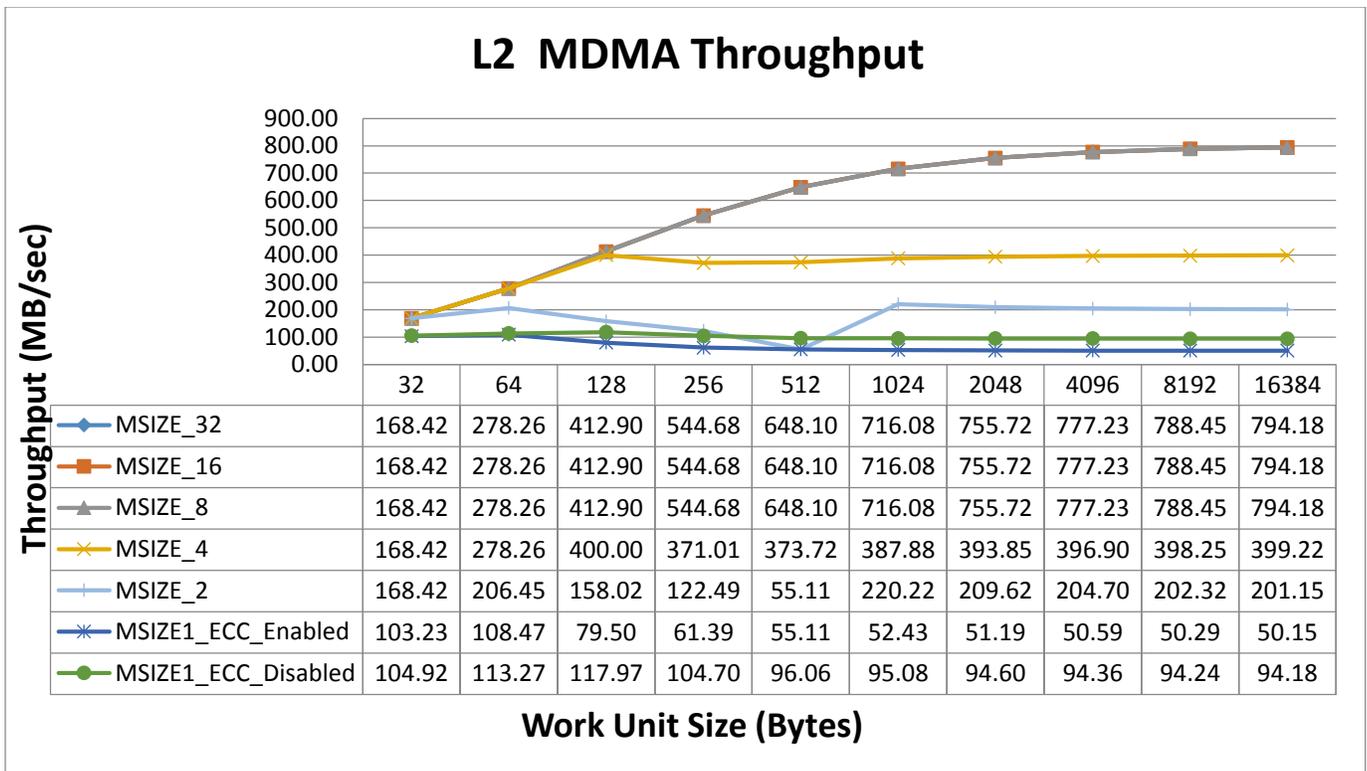


Figure 7. L2 MDMA Throughput for Different DMA\_CFG.MSIZE and Work Unit Size Values

As can be seen, the maximum throughput is very close to 800 MB/s in one direction (1600 MB/s in both directions) for all DMA\_CFG.MSIZE settings greater than or equal to 8 bytes, but it drops significantly for a DMA\_CFG.MSIZE of 4 bytes. L2 throughput drops drastically for 8-bit and 16-bit accesses as well. This is partly due to the non-utilization of the 32-bit wide DMA bus and the additional internal overhead. Throughput for a DMA\_CFG.MSIZE of 8 bytes improves by disabling ECC.

### L3/External Memory Throughput

ADSP-BF707 processors provide interfaces for connecting to different types of off-chip L3 memory devices such as parallel SRAM/Flash devices using the Static Memory Controller (SMC) and DRAM (DDR2/LPDDR) devices using the Dynamic Memory Controller (DMC).

The DMC interface operates at up to 200 MHz; therefore, for the 16-bit DDR2 interface, the maximum theoretical throughput which the DMC can deliver is 800 MB/s. However, the practical maximum DMC throughput is less because of the latencies introduced by the internal system interconnects, as well as the latencies introduced by the DRAM technology itself (access patterns, page hit to page miss ratio, etc.).

Although most of the throughput optimization concepts are illustrated using MDMA as an example, the same can be applied to other system masters as well.

Because they run in the SCLK1 domain, the MDMA1/2 channels can request DMC accesses faster than any other masters. The maximum SCLK1 is 200 MHz; therefore, for a 32-bit peripheral bus width, the maximum possible throughput using only MDMA channel(s) is  $200 \text{ MHz} * 4 = 800 \text{ MB/s}$ . That said, the practical DMC throughput possible using MDMA also depends upon a number of factors, such as whether the

accesses are sequential or non-sequential, the block size of the transfer, and other DMA parameters (e.g., DMA\_CFG.MSIZE).

Figure 8 and Figure 9 provide the DMC’s measured throughput between L1 and L3 memories for sequential read and write accesses using MDMA streams 18-19 and 20-21, respectively, for various DMA\_CFG.MSIZE values and for different buffer sizes. The lines with square markers are DMA streams 18-19, whereas the lines with round markers are DMA streams 20-21.

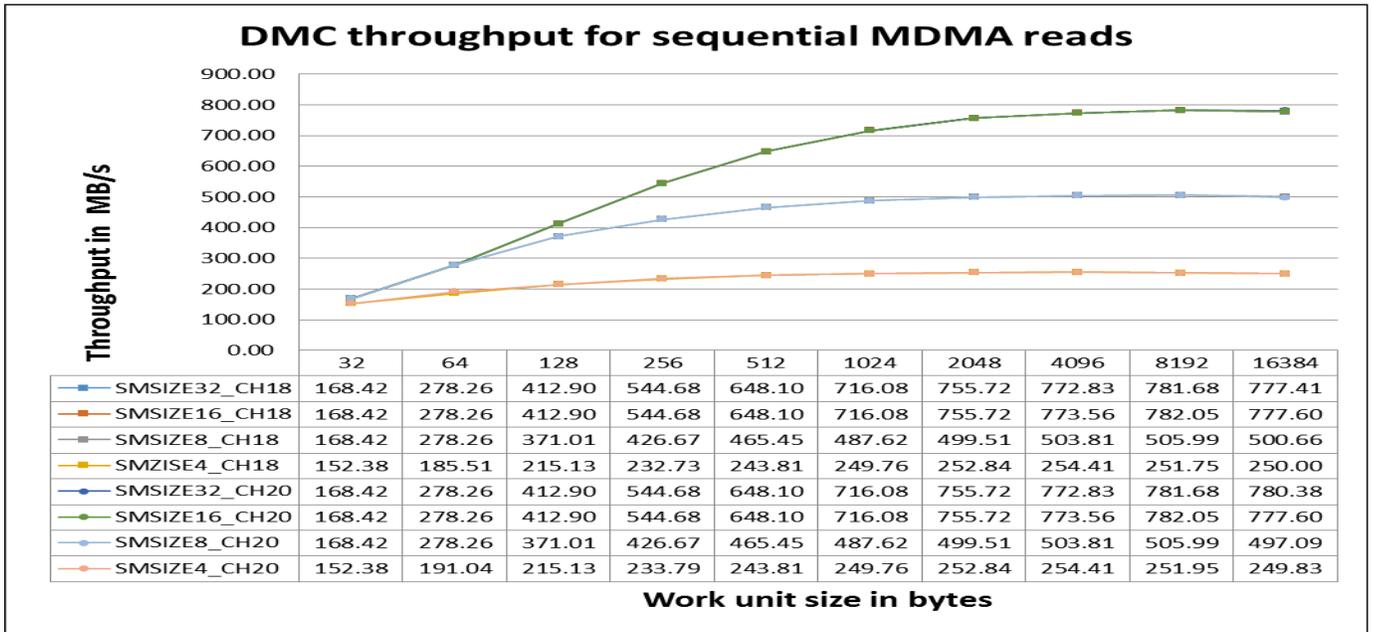


Figure 8. DMC Measured Throughput for Sequential MDMA Reads

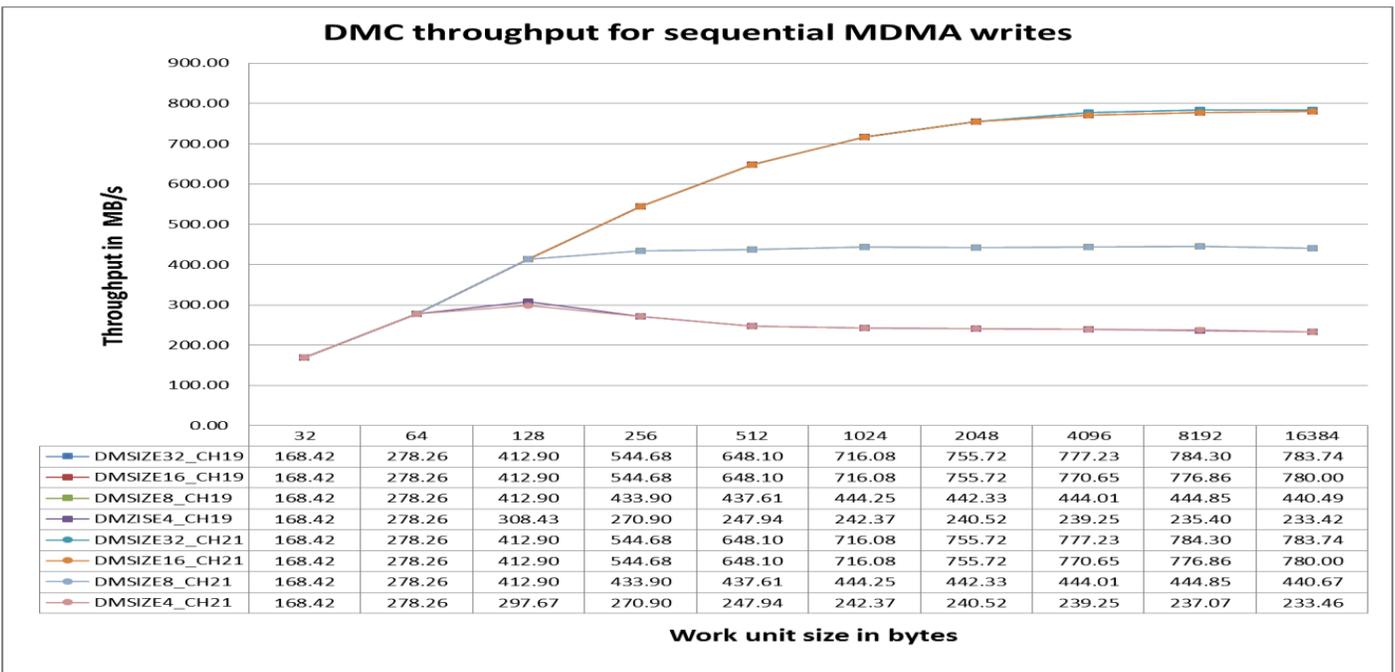


Figure 9. DMC Measured Throughput for Sequential MDMA Writes

The following important observations can be made from these plots:

- The throughput trends are very similar for reads and writes with regards to `DMA_CFG.MSIZE` and buffer sizes.
- The throughput depends largely upon the DMA buffer size. For smaller buffer sizes, the throughput is significantly lower, increasing to up to 780 MB/s with a larger buffer size. For instance, DMA channel 18 with `DMA_CFG.MSIZE` of 32 and a buffer size of 32 bytes yields a read throughput of 168.4 MB/s (21.05%), whereas the throughput reaches 780 MB/s (97.5%) for a 16 KB buffer size. This is largely due to the overhead incurred when programming the DMA registers, as well as the system latencies when sending the initial request from the DMA engine to the DMC controller.



Try to rearrange the DMC accesses such that the DMA count is as large as possible (i.e., better sustained throughput is obtained for continuous transfers over time).

- To some extent, throughput also depends upon the `DMA_CFG.MSIZE` value of the source MDMA channel for reads and the destination MDMA channel for writes. As shown in [Figure 8](#) and [Figure 9](#), in most cases, larger `DMA_CFG.MSIZE` values provide better results. Ideally, `DMA_CFG.MSIZE` should be at least equal to the DDR2 memory burst length. That is, for MDMA channel 18 with a buffer size of 16384 bytes, the read throughput is 780 MB/s (97.5%) for `DMA_CFG.MSIZE` of 32, but it reduces significantly to 152 MB/s (19%) for a `DMA_CFG.MSIZE` of 4. This is due to the fact that, for a `DMA_CFG.MSIZE` of 4, although all accesses are still sequential, the full DDR2 memory burst length of 8 bytes (four 16-bit words) is not used.

As explored, for sequential reads, it is easily possible to achieve optimum throughput, particularly for larger buffer sizes. This is because the DRAM memory page hit ratio is high, and the DMC controller does not need to close and open DDR2 device rows too many times. However, in the case of non-sequential accesses, throughput may drop slightly or significantly depending upon the page hit-to-miss ratio.

[Figure 10](#) provides a comparison of the DMC throughput numbers measured for sequential MDMA read accesses for a `DMA_CFG.MSIZE` of 8 bytes (equal to the DDR2 burst length) and `DMC_CTL.ADDRMODE` set to 0 (bank interleaving) versus non-sequential accesses with a modifier of 2048 bytes (equal to the DDR2 page size, thus leading to a worst-case scenario with maximum possible page misses).

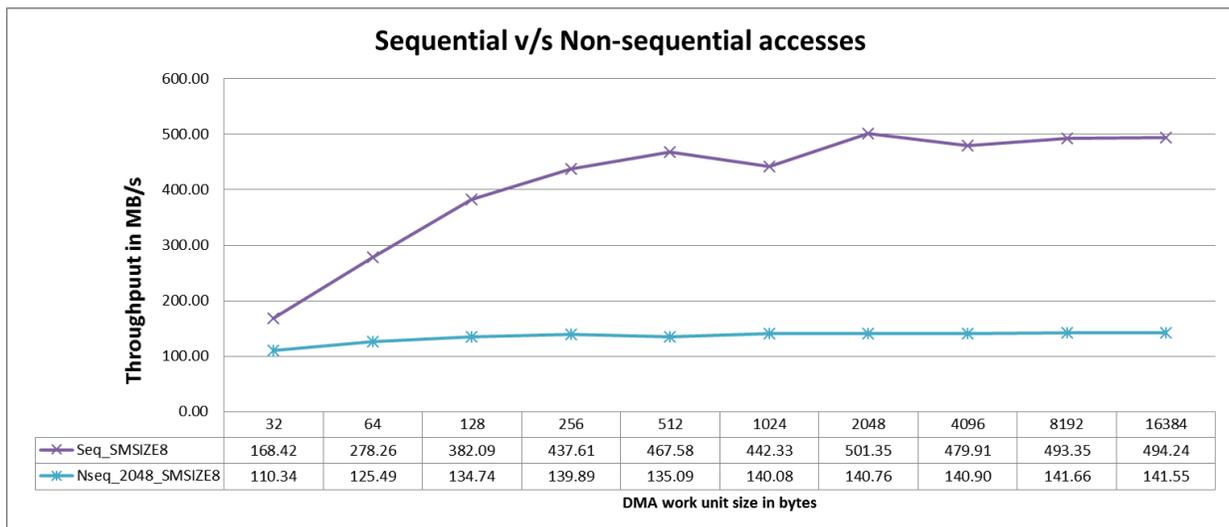


Figure 10. DMC Throughput for Non-Sequential and Sequential Read Accesses

As can be seen, for a buffer size of 16384 bytes, throughput drops significantly from 494.24 to 141.55 MB/s. DDR2 memory devices also support concurrent bank operation, allowing the DMC controller to activate a row in another bank without pre-charging the row of a particular bank. This feature is extremely helpful in cases where DDR2 access patterns incur page misses. By setting the `DMC_CTL.ADDRMODE` bit, throughput can be improved by ensuring that such accesses are to different banks. Figure 11 shows this in practice.

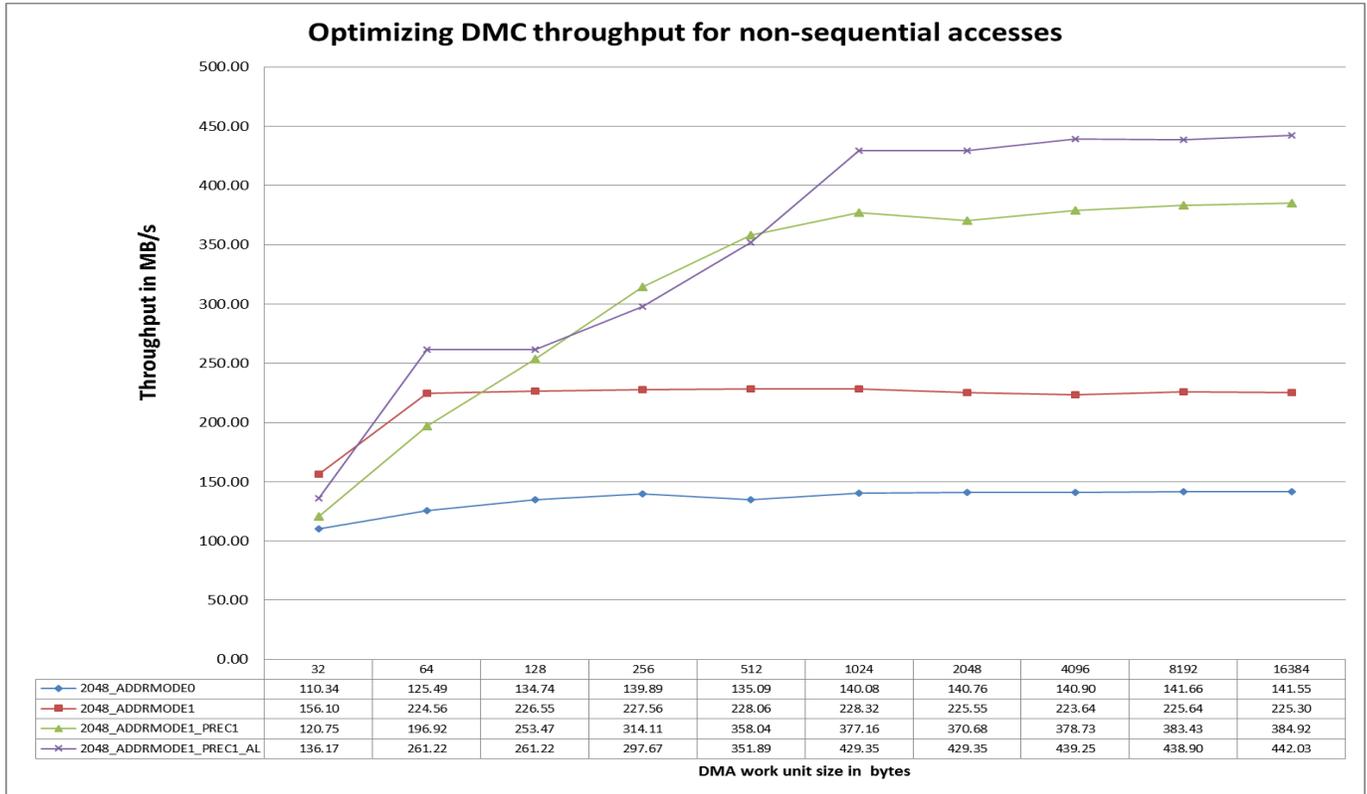


Figure 11. Optimizing Throughput for Non-Sequential Accesses

As can be seen, the DMC throughput increases from 141.55 MB/s to 225.30 MB/s (in red) simply by setting the `DMC_CTL.ADDRMODE` bit for the non-sequential access pattern shown in Figure 10.

The throughput can be further improved using the `DMC_CTL.PREC` bit, which forces the DMC to close the row automatically as soon as a DDR read burst is complete (using the *Read with Auto Pre-charge* command). This allows the row of a bank to proactively pre-charge after it has been accessed, helping improve the throughput by saving the latency involved in pre-charging the row at the time when the next row of the same bank has to be activated, as illustrated by the green line in Figure 11.

Note how the throughput increases from 225.30 MB/s to 384.92 MB/s by just setting the `DMC_CTL.PREC` bit. The same result can be achieved by setting the `DMC_EFFECTL.PRECBANK[7-0]` bits, which is used on a per bank basis. However, setting the `DMC_CTL.PREC` bit overrides the `DMC_EFFECTL.PRECBANK[7-0]` bits and results in pre-charging of the rows after every read burst, while setting the `DMC_EFFECTL.PRECBANK[7-0]` bits pre-charge the row after the last burst (corresponding to the respective `DMC_CFG.MSIZE` settings). This can provide an added throughput advantage for cases where `DMA_CFG.MSIZE` (e.g., 32 bytes) is greater than the DDR2 burst length (8 bytes).

For reads, the throughput can be further improved by using the additive latency feature supported by the DMC controller and DDR2 SDRAM devices, as shown in Figure 12 and Figure 13 (TN-47-02: DDR2 Offers New Features/Functionality Introduction [6]).

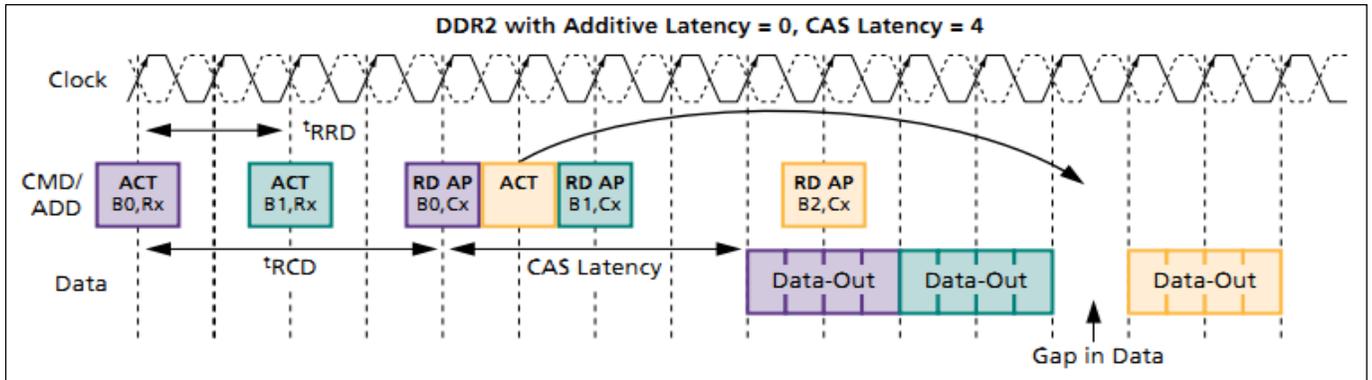


Figure 12. DDR2 Reads without Additive Latency

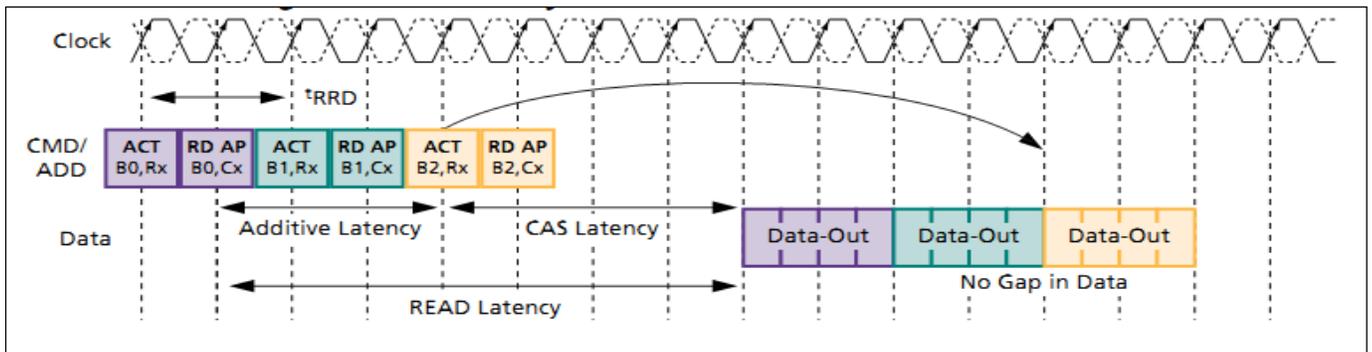


Figure 13. DDR2 Reads with Additive Latency

Programming the additive latency to  $t_{RCD}-1$  allows the DMC to send the *Read with Autoprecharge* command right after the *Activate* command, before the  $t_{RCD}$  duration. This enables the controller to schedule the *Activate* and *Read* commands for other banks, eliminating gaps in the data stream. The purple line in Figure 11 shows how the throughput improves from 384.92 MB/s to 442.03 MB/s by programming the additive latency (AL) in the `DMC_EMR1` register to  $t_{RCD}-1$  (in this case, it is set to three).

The DMC also allows elevating the priority of the accesses requested by a particular SCB master via the `DMC_PRI0` and `DMC_PRI0MSK` registers. The associated ZIP file provides example code in which two MDMA DMC read channels (18 and 20) run in parallel. Table 2 summarizes the measured throughput.

Test Case No.	Priority Channel (SCB ID)	MDMA1 (Ch.18) Throughput (MB/s)	MDMA2 (Ch.20) Throughput (MB/s)
1	None	328	326
2	MDMA1(0xC0)	317	363

Table 2. Measured DMC Throughput for Different DMC\_PRI0 Settings for MDMA Channels 20 and 21

For test case 1, when no priority is selected, the throughput for MDMA channel 18 is 328 MB/s, while channel 20 is at 326 MB/s. MDMA channel 20 throughput increases to 363 MB/s by setting the `DMC_PRIO` register to the corresponding SCB ID (0xC0) and the `DMC_PRIOMSK` register to 0xFFFF.

Furthermore, the *Postpone Autorefresh* command can be used to ensure that auto-refreshes do not interfere with any critical data transfers. Up to eight *Autorefresh* commands can be accumulated in the DMC. The exact number of *Autorefresh* commands can be programmed using the `DMC_EFFCTL.NUM_REF` bit.

After the first refresh command is accumulated, the DMC constantly looks for an opportunity to schedule a refresh command. When the SCB read and write command buffers become empty (which implies that no access is outstanding) for the programmed number of clock cycles (`IDLE_CYCLES`) in the `DMC_EFFCTL` register, the accumulated number of refresh commands are sent back-to-back to the DRAM memory.

After every refresh, the SCB command buffers are checked to ensure they stay empty. However, if the SCB command buffers are always full, once the programmed number of refresh commands gets accumulated, the refresh operation is elevated to urgent priority and one refresh command is sent immediately. After this, the DMC continues to wait for an opportunity to send out refresh commands. If self-refresh is enabled, all pending refresh commands are issued only after that DMC enters self-refresh mode.

Figure 14 shows the measured throughput for a MDMA stream reading via the DMC controller for different work unit sizes with and without using the *Postpone Autorefresh* feature. The `IDLE` cycles were programmed to its maximum supported value (15).

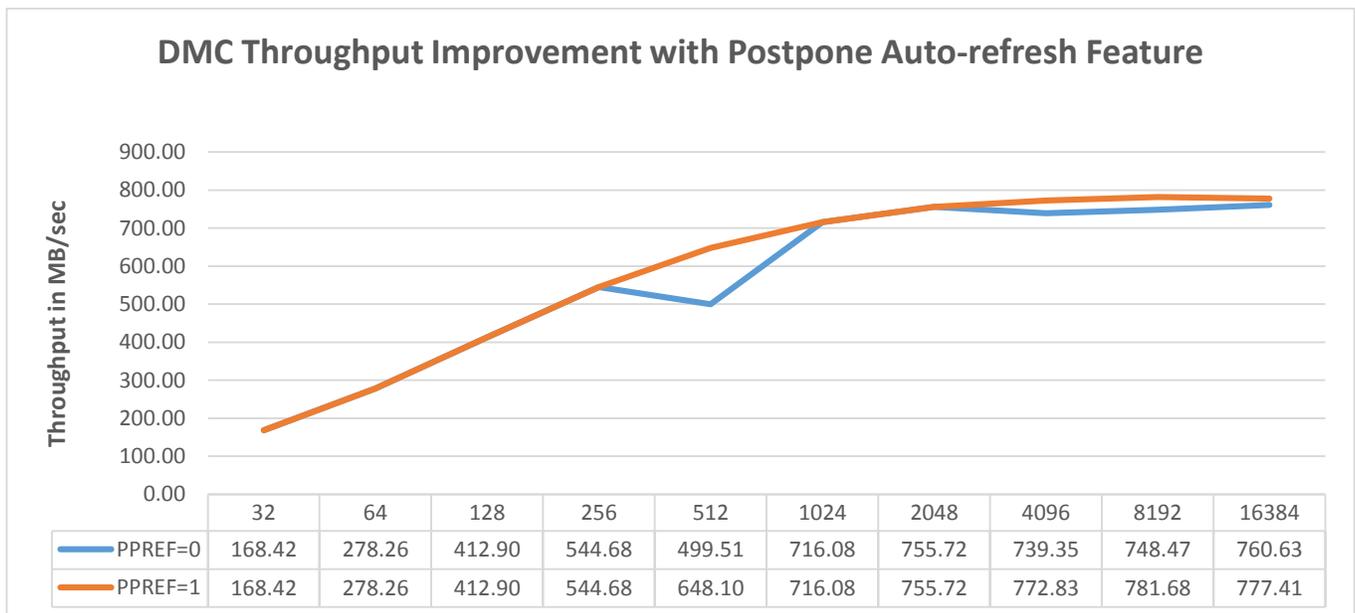


Figure 14. DMC Throughput Optimization with Postpone Autorefresh Feature

As can be seen, there is a slight throughput improvement, especially for the case where the block lengths are 512, 4096, 8192, and 16384 bytes.

#### System MMR Latencies

Unlike previous Blackfin processors, ADSP-BF70x processors may have higher MMR access latencies. This is mainly due to the interconnect fabric between the core and the MMR space and the number of

different clock domains (`SYSCLK`, `SCLK0` and `SCLK1`). With this, ADSP-BF70x processors have three groups of peripherals for which the MMR latency varies, as summarized in [Table 3](#).

Group 1	Group 2		Group 3
RCU	GPIO	COUNTER	CRC
TRU	PINT	HADC	CRYPTO
CGU	TMR	OTP	MDMA1/2
DPM	WDT	MSI	
SEC	TWI	SWU	
SPU	SPI	USB	
SMPU	UART	MDMA0	
L2CTL	PPI	DMA0/15	
DEBUG	CAN	RTC	

Table 3. Peripheral Groups Sorted by MMR Latency

[Table 4](#) shows the MMR latencies associated with the peripheral groups defined in [Table 3](#). Each peripheral in the group observes the same MMR access latencies.

MMR Access	Group1	Group2	Group3
Read (Single Read access)	36 CCLK	52 CCLK	40 CCLK
Write (8 sequential Posted Write)	91 CCLK	91 CCLK	91CCLK
Write (8 sequential Non- Posted Write)	187 CCLK	269 CCLK	211 CCLK

Table 4. Approximate MMR Access Latencies

On ADSP-BF70x processors, MMR write latency can be reduced by using the posted write feature supported by the core, enabled by setting the `SYSCFG.MPWEN` bit. When posted writes are enabled, the core does not wait for the write completion response before executing the next instruction, which could be another MMR write (thereby reducing the latency for multiple writes). However, the MMR access latencies may vary depending upon:

- **Clock ratios.** All MMR accesses are through SCB0, which is in the `SYSCLK` domain, while peripherals are in the `SCLK0/1` domains. Thus, the `CCLK::SYSCLK::SCLK0::SCLK1` ratio will affect the MMR latency. For example, a ratio of 4::2::1::2 is optimum for minimal MMR latency.
- **The number of concurrent MMR access requests in the system.** Although a single write incurs half the system latency as compared to back-to-back writes, the latency observed by the core will be shorter.

Similarly, the system latency incurred by a read followed by a write, or vice-versa, will differ from that observed by the core.

- **Posted write enabled.** The application may use an SSYNC instruction to force the core to wait for the responses from all the previous writes before executing the next instruction.



MMR write latencies are for eight back-to-back write accesses, as observed by the core, and have been calculated using C code (so numbers may vary slightly from case to case). A single MMR write latency is just two CCLKS.

## System Bandwidth Optimization Procedure

Although the optimization techniques may vary from one application to another, the general procedure involved in the overall system bandwidth optimization remains the same. Figure 15 provides a flow chart of a typical system bandwidth optimization procedure for ADSP-BF707-based applications.

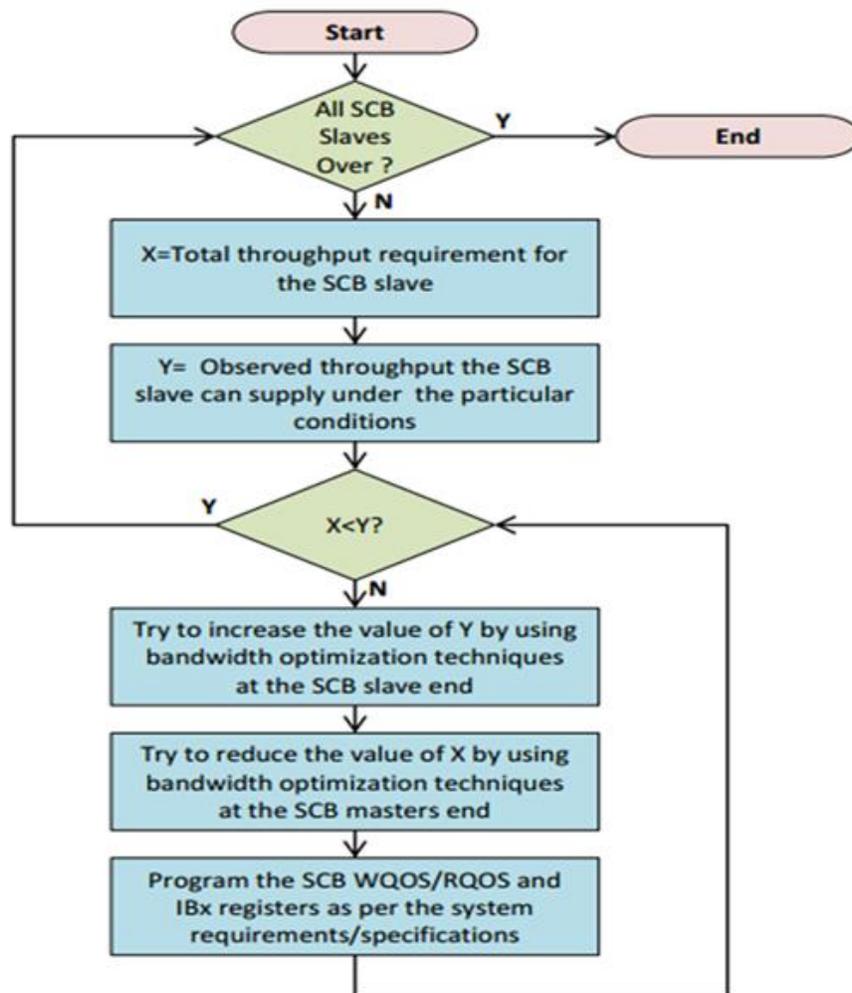


Figure 15. Typical System Bandwidth Optimization Procedure

For an SCB slave:

- Identify the individual and total throughput requirements for all the masters accessing the SCB slave in the system and allocate the corresponding read/write SCB arbitration slots accordingly. Consider the total throughput requirement to be value  $X$ .
- Calculate the observed throughput that the SCB slave(s) may be able to supply under the specific application conditions, and call this value  $Y$ .
- If  $X \leq Y$ , the bandwidth requirements are met. If  $X > Y$ , however, at least one peripheral is likely to encounter reduced throughput or an underflow condition. In this case, apply the bandwidth optimization techniques discussed earlier to either:
  - Increase the value of  $Y$  by applying the slave-specific optimization techniques (e.g.; using the DMC efficiency controller features).
  - Decrease the value of  $X$  by:
    - Reanalyzing whether a particular peripheral really needs to run that fast. If not, then reduce the peripheral clock to reduce the bandwidth requested by the peripheral.
    - Reanalyzing whether a particular MDMA can be slowed down (e.g., use `DMAx_BWLCNT` to limit the bandwidth of that particular DMA channel).

### Application Example

Figure 16 shows a block diagram of the example code in the *associated ZIP file*.



The supplied code is intended to stress the SCBs and the DMC controller for high throughput requirements to show various bandwidth optimization techniques. It does not consider pin multiplexing limitations.

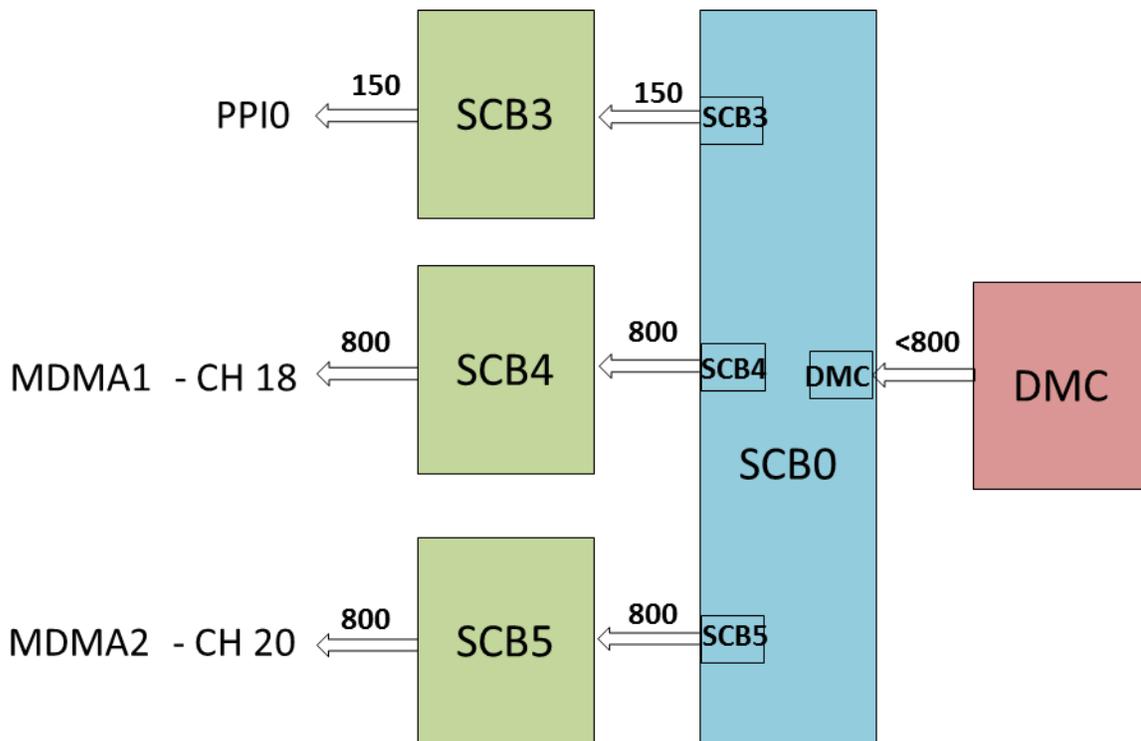


Figure 16. Example Application Showing DMC Throughput Distribution across Various SCBs

The example code defines the following application characteristics:

- $CCLK = 400\text{ MHz}$ ,  $DCLK = 200\text{ MHz}$ ,  $SYSCLK = 200\text{ MHz}$ ,  $SCLK0 = SCLK1 = 100\text{ MHz}$
- SCB3:
  - EPPI0 is configured for 24-bit data (3-byte) transmit with internal clock and frame syncs at  $EPPICLK = 50\text{ MHz}$ . The required throughput is therefore  $50\text{ MHz} * 3 = 150\text{ MB/s}$ .
- SCB4 and SCB5:
  - MDMA1 channels (18-19) and MDMA2 channels (20-21)
  - Throughput requirements depend upon the corresponding  $DMAx\_BWL CNT$  register values. If not programmed, the MDMA channels request for the bandwidth with full throttle (every  $SCLK1$  cycle). This means that both MDMA channels request from the SCB4 and SCB5, each with a throughput of  $200\text{ MB/s} * 4 = 800\text{ MB/s}$ .

As shown in [Figure 16](#), the total required throughput from the DMC controller is 1.75 GB/s ( $150 + 2 * 800$ ). Theoretically, with  $DCLK = 200\text{ MHz}$  and a 16-bit bus width, it should be possible to meet this requirement. However, the maximum throughput that can be supplied by the DMC controller and DDR2 SDRAM memory depends upon the page hit vs. page miss ratio when multiple masters are trying to concurrently access the DMC and the internal latencies involved when switching between masters accessing the DMC. For these reasons, it is possible for one or more masters to not get the required bandwidth, which could result in decreased throughput (for masters like a MDMA channel) or an underflow condition (for masters like the EPPI).

Bandwidth optimization strategies are provided in the example code, as defined in the following steps.

#### Step 1

In this step, all DMA channels run without applying any optimization techniques. To replicate the worst case scenario, the source buffers for all the DMA channels are placed in a single DDR2 SDRAM bank. The row corresponding to “No optimization” in [Table 5](#) shows the expected and measured throughput for all DMA channels and the corresponding SCBs for this case.

S.No.	Condition	SCB3		SCB4		SCB5		Total throughput Requirement X	Measured throughput Y'	PPI Underflow ?
		PPIO		MDMA1		MDMA2				
		Required	Measured	Required	Measured	Required	Measured			
1	No optimization	150	95.27	800	221.17	800	221	1750	537.47	YES
2	Optimization at the slave - T1	150	144.37	800	313	800	312.85	1750	770.22	YES
3	Optimization at the master - T2	150	150.2	300	290	300	289.8	750	729.89	NO
4	Optimization at the master - T3	150	150.44	500	490.32	100	98.35	750	739.11	NO

All units are expressed in MB/s.

Table 5. Example Application System Bandwidth Optimization Steps

As illustrated, the individual measured throughput of almost all channels is significantly less than expected, and the EPPI port encounters an underflow condition. The total expected throughput from the DMC (X) is 1750 MB/s, and the effective DMC throughput (Y) is 526 MB/s. As X is greater than Y, bandwidth optimization techniques should be applied to achieve better application performance.

### Step 2

As previously discussed, frequent DDR2 SDRAM page misses within the same bank can significantly degrade throughput. Although DMA channel accesses are sequential, multiple channels trying to access the DMC concurrently makes page misses more likely to occur. To work around this, the DMA channel source buffers can be reallocated to different DDR2 SDRAM banks, which will allow for parallel accesses to be made to multiple pages of different banks, thus helping to improve  $Y$ . The “*Optimization at the slave - T1*” row in [Table 5](#) provides the measured throughput numbers under this condition. Both the individual and overall throughput numbers increase significantly. The maximum throughput delivered by the DMC ( $Y$ ) increases to 766 MB/s. However, since  $X$  is still greater than  $Y$ , the measured throughput is still lower than expected, resulting in an underflow condition in EPPIO.

### Step 3

There is not much more room to significantly increase the value of  $Y$  on the slave end, but optimization strategy techniques can also be employed on the master end. One method is to try reducing the overall expected throughput to less than 766 MB/s. To meet this target, the bandwidth of the MDMA1 and MDMA2 streams can be limited to 300 MB/s each, thus decreasing the total MDMA required throughput from 1600 MB/s to 600 MB/s. The “*Optimization at the master - T2*” row in [Table 5](#) shows the measured throughput under this condition. As can be seen, each individual component’s bandwidth requirement (and the overall throughput requirement) are nearly met, and no underflow condition exists.

### Step 4

Let’s assume it’s desired to increase MDMA1 throughput to 500 MB/s. In this case, bandwidth can be reallocated from the MDMA2 stream by limiting its bandwidth to 100 MB/s in such a way that the overall expected throughput remains nearly the same (i.e.; 750 MB/s). The “*Optimization at the master - T3*” row in [Table 5](#) provides the measured throughput numbers under this condition. Again, each individual component’s bandwidth requirement (and the overall throughput requirement) are nearly met, and no underflow condition exists.

## System Optimization Checklist

This section summarizes the system bandwidth optimization techniques discussed in this EE-note, while also listing a few additional tips:

- Analyze the overall bandwidth requirements and make use of the bandwidth limit feature for memory pipe DMA channels to regulate the overall DMA traffic.
- Program the DMA channels’ `DMA_CFG.MSIZE` parameters to optimum values to maximize throughput and avoid any potential underflow/overflow conditions.
- Program the `SCB_MST_RQOS` and `SCB_MST_WQOS` registers to allocate priorities to various masters as per system requirements.
- Program the clock domain crossing (`SCB_IBx`) registers depending upon the clock ratios across SCBs.
- Make use of various optimization techniques at the SCB slave end, such as:
  - Efficient usage of the DMC controller
  - Usage of multiple L2/L1 sub-banks to avoid access conflicts
  - Usage of instruction/data caches
- Maintain the optimum clock ratios across different clock domains

- MMR write latencies can be reduced by enabling posted writes on the cores. If writes are done to different peripherals, use an `SSYNC` instruction in between them to avoid interrupt scenarios that would result in re-entering the interrupt routine.
- Since MMR latencies affect the interrupt service latency, ADSP-BF707 processors offer the Trigger Routing Unit (TRU) for bandwidth optimization and system synchronization. The TRU allows for synchronizing system events without processor core intervention. It maps the trigger masters (trigger generators) to trigger slaves (trigger receivers), thereby offloading processing from the core. For more details, refer to *Utilizing the Trigger Routing Unit for System Level Synchronization (EE-360)*<sup>[7]</sup>. Though it is written for the ADSP-BF60x Blackfin processor, the concepts can be used for the ADSP-BF707 processor as well.

## References

- [1] *System Optimization Techniques for Blackfin Processors (EE-324)*. Rev 1, July 2007. Analog Devices, Inc.
- [2] *ADSP-BF60x Blackfin® Processor System Optimization Techniques (EE-362)*. Rev 1, November 2013. Analog Devices, Inc.
- [3] *ADSP-BF70x Blackfin Processor Hardware Reference*. Rev 0.2, May 2014. Analog Devices, Inc.
- [4] *Associated ZIP File for EE-373*. Rev 1. July 2015. Analog Devices, Inc.
- [5] *ADSP-BF701/ADSP-BF702/ADSP-BF703/ADSP-BF704/ADSP-BF705/ADSP-BF706/ADSP-BF707 Blackfin Embedded Processors Data Sheet*. Rev PrD, November 2014. Analog Devices, Inc.
- [6] *ADSP-BF60x Blackfin® Processors System Optimization Techniques (EE-362)*. Rev 1, November 12, 2013. Analog Devices, Inc.
- [7] *Utilizing the Trigger Routing Unit for System Level Synchronization (EE-360)*. Rev 1, October 2013. Analog Devices Inc
- [8] *TN-47-02: DDR2 Offers New Features/Functionality Introduction*. Rev A, June 2006. Micron Technology Inc.

## Document History

Revision	Description
Rev 1 – January 5 <sup>th</sup> , 2016 by Akash Agarwal, Mitesh Moonat, and Nabeel Shah.	Initial release.