ADSP-BF60x Blackfin® Processors System Optimization Techniques

Contributed by Akash Agarwal, Mitesh Moonat, and Nabeel Shah

Rev 1 – November 12, 2013

Introduction

The ADSP-BF60x family of Blackfin® processors (hereafter referred to as ADSP-BF609 processors) provides an optimized architecture supporting high system bandwidth and advanced peripherals. The purpose of this application note is to discuss the processors key architectural features contributing to the overall system bandwidth, as well as various available bandwidth optimization techniques.

Despite having a fairly new architecture, these processors have a number of similarities (especially the core) to the older Blackfin processor architecture. Thus, some of the system optimization techniques discussed in *System Optimization Techniques for Blackfin® Processors (EE-324)*[3] are also applicable for these processors. Therefore, this EE-Note will mainly focus on the new features of the ADSP-BF60x processors.

ADSP-BF609 Processors Architecture

This section provides a holistic view of the ADSP-BF609 processors key 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-BF60x Processor Hardware Reference*[2].

System Bus Slaves

The System Bus Slaves are shown on the right hand side of Figure 1. They include on-chip and off-chip memory devices/controllers, such as L1 SRAM (Core0 L1, C0L1, and Core1 L1, C1L1), L2 SRAM, the Dynamic Memory Controller (DMC) for DDR2 and LPDDR SDRAM devices, the Static Memory Controller (SMC) for SRAM and Flash devices, and the System Memory Mapped Registers (MMRs).

Each System Bus Slave has its own latency characteristics, operating in a given clock domain as shown in Figure 1. 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 hand side of Figure 1. They include peripheral Direct Memory Access (DMA) channels, such as the Enhanced Parallel Peripheral Interface (EPPI), Link Port, and Serial Port (SPORT), among others. Also included are the Memory-to-Memory DMA channels (MDMA) and cores.

Note that each peripheral runs at a different clock speed, and thus have their individual bandwidth requirements. For example, parallel peripherals such as the EPPI and Link Port require higher bandwidth compared to the slower serial peripherals such as the SPORT, the Serial Peripheral Interface (SPI) or the Two Wire Interface (TWI).

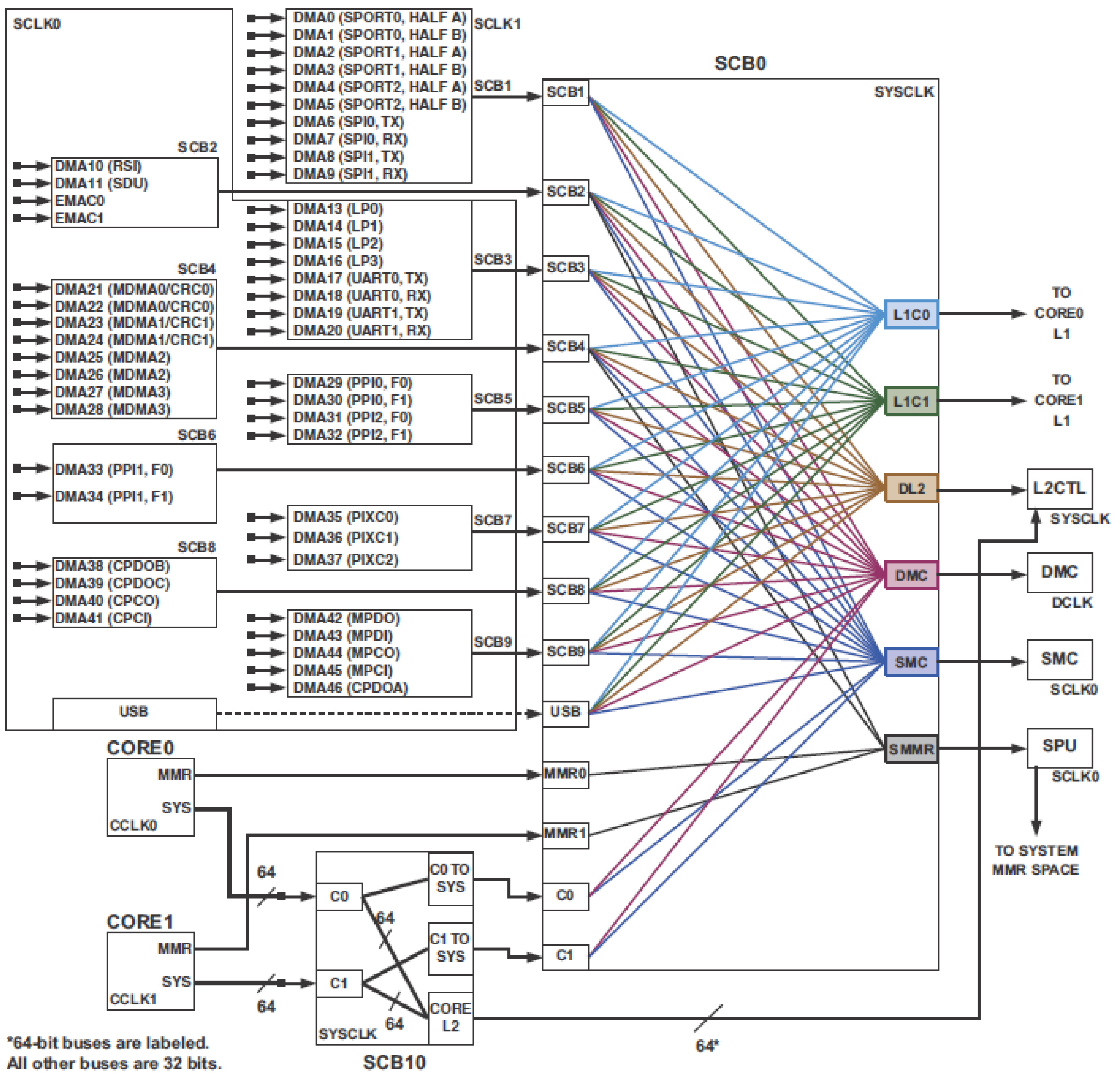


Figure 1. ADSP-BF609 Processors system architecture

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) introducing their own 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), and another bus that connects to either a peripheral or another DMA channel. The SCB/memory bus width can vary between 8, 16, 32, or 64 bits and is given by the `DMA_STAT.MBWID` bit field, while the peripheral bus width can vary between 8, 16, 32, 64, or 128 bits and is given by the `DMA_STAT.PBWID` bit field. For ADSP-BF609 processors, the memory and peripheral bus widths are fixed to 32 bits (4 bytes).

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

The DMA parameter `DMA_CFG.MSIZE` 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 bytes. If the `MSIZE` value is greater than `DMA_STAT. MBWID`, the SCB performs burst transfers to transfer the data equal to `MSIZE` value.

Thus, it is important to understand how to choose the appropriate `MSIZE` value, both from functionality and performance perspective. The following points should be kept in mind when choosing the `MSIZE` value:

- The start address of the work unit should always align to the `MSIZE` value selected. Failing to do so will generate a DMA error interrupt.
- As a general rule, from a performance/throughput perspective, the highest possible `MSIZE` value (32 bytes) 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 `MSIZE` value is determined by the burst length supported by the memory device. For example, for DDR2 accesses, the minimum `MSIZE` value is limited by the DDR2 burst length (4 or 8 bytes). Any `MSIZE` value below this would lead to a significant throughput loss. For more details, please refer to the L3/External Memory Throughput section.

- There are some cases in which a smaller `MSIZE` value might be useful. Let's assume a case where multiple DMA channels, such as EPPI, MDMA, or Pipelined Vision Processor (PVP) memory pipe DMA, are all active simultaneously. For memory pipe DMA channels such as MDMA or PVP, it is efficient to use the maximum `MSIZE` value (32 bytes) to improve the average throughput, as described above.

  However, for some real time peripherals, especially high speed parallel peripherals such as EPPI, in addition to maintaining the average throughput; it is also important to avoid underflows/overflows in real time. The DMA FIFO size of the peripheral DMA channels is limited (e.g., 64 bytes). Choosing a higher `MSIZE` value (e.g., 32 bytes) would mean that the peripheral can only request a new DMA transfer after 32 bytes have been emptied/filled by the peripheral.

  Thus, if there are other DMA channels (especially the memory pipe DMA channels requesting at a very high rate) running concurrently, they might delay the grant to the peripheral and thus chances of underflow/overflow would increase. On the other hand, choosing a smaller `MSIZE` value (e.g., 2 bytes) would make sure that the peripheral DMA is requesting the DMA transfer early enough and is also serviced faster, which might avoid underflow/overflow conditions.

- In some cases, the optimum `MSIZE` value selection may also depend upon the DMA FIFO depth of the master. For example, for MDMA channel 0 with a FIFO depth of 128 bytes, `MSIZE` set to 32 bytes will provide the optimum throughput. However, for MDMA channel 1 with FIFO depth of 64 bytes, `MSIZE` set to 16 bytes may provide better throughput. Whereas for FIFO depth of 64 bytes, and `MSIZE` of 32 bytes, it may only queue up to two DMA requests in the pipeline, for `MSIZE` of 16 bytes, it would be able to queue up more (4) DMA requests.

Furthermore, the FIFO depth of a DMA channel can also make a significant difference in terms of throughput. In general, DMA channels with larger FIFOs provide better throughput. This is because they can accumulate a greater number of DMA requests, which in turn reduces the effective latencies associated with the internal fabric.

For example, the DMA channels corresponding to the MDMA streams 0 and 2 have a FIFO depth of 128 bytes as compared to the FIFO depth of 64 bytes in case of the DMA channels corresponding to the MDMA streams 1 and 3. Thus, the DMA channels corresponding to the MDMA streams 0 and 2 (21-22 and 25-26) will typically provide better throughput, especially for accessing higher latency memories such as L2 SRAM or L3 DDR2 SDRAM.

Bandwidth Limiting and Monitoring

All DMA channels that operate in memory-to-memory mode (MDMA, PVP memory pipe DMA, Pixel Compositor – PIXC, DMA) 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 `SCLK0` 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 MBytes per second, will be determined by the `DMA_BWLCNT` register and the `MSIZE` value, and is calculated as follows:

*Bandwidth = SCLK frequency in MHz*MSIZE in bytes / DMC_BWLCNT*

The example code provided in the associated `.ZIP` file utilizes a MDMA stream to perform data transfers from DDR2 to L1 memory. In this example, `DMC_BWLCNT` register is programmed to different values to limit the throughput to an expected value for different `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 `SCLK0` frequency used for testing equals 125 MHz with a DMA buffer size of 16384 words.

Sample No.	**MSIZE**	**BWLCNT**	**Theoretical Throughput (MB/s) (SCLK*MSIZE/BWLCNT)**	**Measured Throughput (MB/s)**
1	32	8	500	435.33
2	32	16	250	232.90
3	32	32	125	120.58
4	32	64	62.5	61.37
5	32	128	31.25	30.97
6	16	8	250	220.19
7	16	16	125	117.07
8	16	32	62.5	60.45
9	16	64	31.25	30.73
10	16	128	15.625	15.49
11	8	8	125	110.61
12	8	16	62.5	58.68
13	8	32	31.25	30.27
14	8	64	15.625	15.38
15	8	128	7.8125	7.75

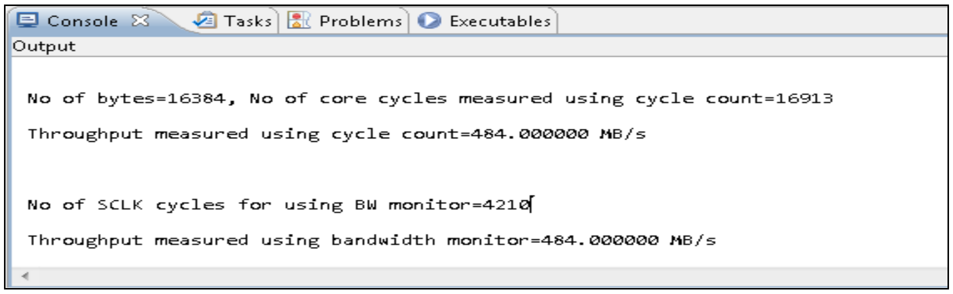
Table 1. DMC MDMA read throughput measured for different BWLCNT values

Furthermore, the bandwidth monitor feature can be used to check if such channels are starving for resources. 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 (MMR access only), a work unit ends, or an autobuffer wraps, the DMA loads the value in `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.

An example code is also provided in the associated `.ZIP` file to help explain this functionality. The example uses MDMA streams 21-22 configured to read from DDR2 memory to L1 memory. The `DMC_BWMCNT` register is programmed to `5120` for an expected throughput of 400 MBytes/s (Buffer Size*SCLK speed in MHz/Bandwidth = 16384*125/400).

When this MDMA runs alone, the measured throughput (both using cycle count and `DMC_BWMCNT`) results as expected (484 MBytes/s), as shown in Figure 2.

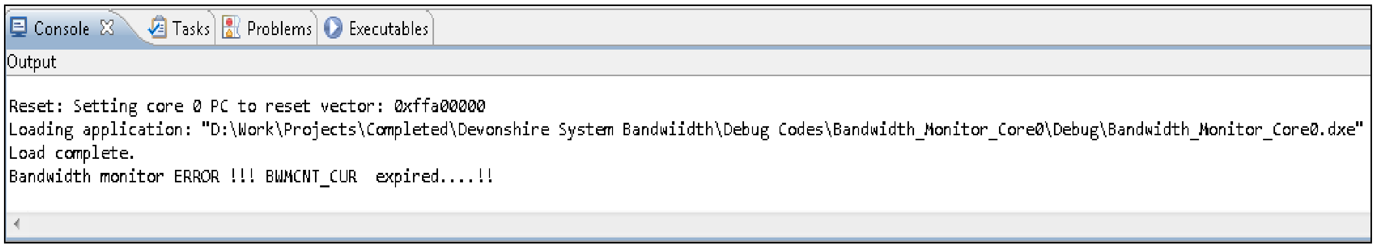


```
No of bytes=16384, No of core cycles measured using cycle count=16913
Throughput measured using cycle count=484.000000 MB/s


No of SCLK cycles for using BW monitor=4210
Throughput measured using bandwidth monitor=484.000000 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 MBytes/s and a bandwidth monitor error is generated as shown in Figure 3.



```
Reset: Setting core 0 PC to reset vector: 0xffa00000
Loading application: "D:\Work\Projects\Completed\Devonshire System Bandwiidth\Debug Codes\Bandwidth_Monitor_Core0\Debug\Bandwidth_Monitor_Core0.dxe"
Load complete.
Bandwidth monitor ERROR !!! BWMCNT_CUR  expired....!!
```

Figure 3. Bandwidth monitor expires due to less than expected throughput

Understanding the System Crossbars

As shown Figure 1, 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 Master Interface (MI). On each SCB unit, each S is connected to a fixed MI, and similarly each M is connected to a fixed SI. Each MI has a fixed number of programmable slots, and by default on each programmable slot, a particular SI is assigned. For a detailed list of the allocation of slots and Master Interfaces on each SCB, please refer to the SCB Arbitration Table in the ADSP-*BF60x Blackfin Processor Hardware Reference*[2].

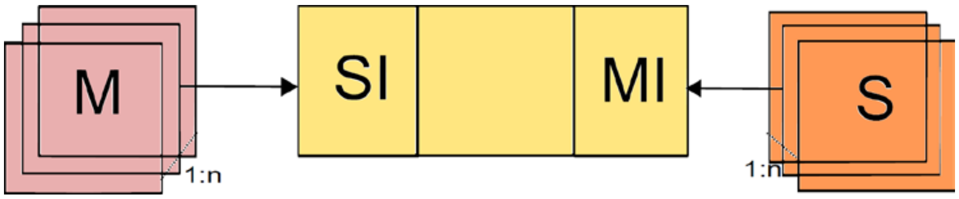


Figure 4. Single SCB block diagram

In SCB with multiple SI for each MI, round-robin arbitration is used to manage slot access on both the read and write channels concurrently. The SCB allows for assigning a programmable number of slots to each SI using the SCB read/write channel arbitration registers in order to adjust the DMA requests priorities as per the bandwidth requirements of the system.

Programming the SCB slots

The associated `.ZIP` file provides an SCB slot programming example showing how the SCB registers can be programmed to allocate different number of SCB arbitration slots to different masters. In this example, four MDMA streams (0-3) are used as masters, while L1, L2, or L3 memory can be used as slaves. On ADSP-BF609 processors, all four MDMA channels are connected on the same SCB4 block. SCB4 has a single Master Interface (MI0) and 16 programmable slots. By default, these 16 slots are divided equally among the 8 DMA channels (DMA21-DMA28).

The maximum observed throughput using the provided example for reading and writing from L1 and L2 memory is 914.38 MBytes/s, while for L3 memory equals 623.00 MBytes/s. This was calculated using a single active master only (MDMA0 stream) on SCB4. As the number of active master's increase, the throughput is divided among the MDMA channels on the SCB4.

In order to highlight the effect of slot re-programming on bandwidth allocation, three test cases can be selected (using the corresponding `TEST_CASE` macro).

- **Test Case 1:** Default configuration. 2 Slots for each DMA channel.
- **Test Case 2:** 3 slots for each MDMA stream 0 channels (21-22), 2 slots for each MDMA streams 1 and 2 channels (23-24 and 25-26), and 1 slot for each MDMA stream 3 channels (27-28).
- **Test Case 3:** 4 slots for each MDMA stream 0 channels (21-22), 2 slots for each MDMA stream 1 and 2 channels (23-24, 25-26), and no slot for MDMA stream 3 channels (27-28).

Table 2, Table 3, and Table 4 show the results obtained for the above test cases for buffers in L1, L2, and L3 memory respectively.

Test Case	**Throughput for Buffer in L1 Memory (MBytes/sec)**			
	MDMA0(21-22)	**MDMA1(23-24)**	**MDMA2(25-26)**	**MDMA3(27-28)**
1	317.89	158.95	317.89	158.95
2	322.53	166.82	319.74	144.58
3	381.47	190.47	381.47	0.00*

Table 2. SCB throughput for R/W from L1 memory

Test Case	**Throughput for Buffer in L2 Memory (MBytes/sec)**			
	MDMA0(21-22)	**MDMA1(23-24)**	**MDMA2(25-26)**	**MDMA3(27-28)**
1	317.90	158.94	317.90	158.94
2	317.90	158.94	317.90	158.94
3	378.86	195.96	378.86	0.00*

Table 3. SCB throughput for R/W from L2 memory

Test Case	Throughput for Buffer in L3 Memory (MBytes/sec)			
	MDMA0(21-22)	MDMA1(23-24)	MDMA2(25-26)	MDMA3(27-28)
1	251.30	125.66	251.30	125.66
2	251.30	125.66	251.30	125.66
3	301.58	150.78	301.56	0.00*

Table 4. SCB throughput for R/W from SDRAM memory

None of the DMA requests for these channels are granted, thus the transfer never completes.

The aim of the example is to show how SCB slot assignment can be altered and not to show how throughput can be changed proportional to the SCB slot assignments. As it can be seen from the above test cases, increasing the slots for a particular master does not lead to a proportional increase in throughput for that master.

Furthermore, it is also recommended to make sure that the SCB arbitration slots are programmed in the ratios as per the individual average throughput requirements. However, at the same time, the total throughput request to the SCB should be limited and should not exceed what the slave (L1/L2/L3) and the corresponding SCB can support.

Let's now consider a case in which two MDMA streams (MDMA0 and MDMA1 belonging to SCB4) and four other peripherals (P1, P2, P3, and P4), each from a different SCB, are running in parallel trying to access the DMC without programming the corresponding bandwidth limit registers.

The throughput requirements of all these DMA channels without any bandwidth limiting are 500, 500, 187.5, 125, 125, and 62.5 MBytes/sec respectively, which in total equal 1500 MBytes/sec. In this case, since SCB4 can support a maximum throughput of 500 MBytes/sec in one direction, the total read throughput which can be delivered to the two MDMA channels is limited to 500 MBytes/sec. Also, let's assume that the maximum practical throughput the DMC can deliver in this case is only 800 MBytes/sec.

Now, to meet these requirements, the bandwidth limit registers should be programmed in such a way that:

- The total read throughput request to SCB0-MI0 (DMC) does not exceed 800 MBytes/sec.
- The total read throughput request to SCB4 does not exceed 500 MBytes/sec.

One way to do this is to limit the bandwidth of both MDMA read channels to 125 MBytes/sec, which limits the total SCB4 throughput requirement to 250 MBytes/sec and the total throughput requirement to SCB0-MI0 to 750 MBytes/sec. In this case:

- The SCB4-MI0 read arbitration slots for the two MDMA channels should be programmed in 1:1 ratio, and
- The SCB0-MI0 read arbitration slots for SCB4, SCB-P1, SCB-P2, SCB-P3, and SCB-P4 should be programmed in the ratio 125:125:187.5:125:125:64.5, i.e.; 2:2:3:2:2:1.

In cases where the core is also accessing the external memory, make sure that an appropriate number of slots are allocated for the core as well.

Arranging the SCB slots

In addition to estimating the number of slots required for each peripheral/SCB, it is also important how the slots are arranged. In general, the SCB slot distribution across different masters should be uniform.

Let’s assume a case where one or more memory pipe DMAs and a few critical real-time peripheral DMAs (e.g.; EPPIs) are running concurrently. In such a case, even though the number of slots for each of these DMA channels is assigned appropriately as per the average throughput requirements, there might be a condition when the EPPI(s) can underflow/overflow. This might happen if the slots are arranged in such a way that all the memory pipe DMA slots are set consecutively and, similarly all EPPI slot(s) are set consecutively at the end. This might lead to a condition where the EPPI may not get the DMA grant for a long time and consequently, it might underflow/overflow. In such a situation, it might help to insert the EPPI slots in between two memory pipe DMA slots to avoid underflow/overflow. Also, for such cases, as mentioned in the earlier section, the `MSIZE` value of the EPPI should be as small as possible, just enough to meet the average throughput requirements (e.g.; 2, 4, 8, or 16 bytes instead of 32 bytes). This allows the EPPI to make an early DMA request which can help avoid underflow/overflow conditions.

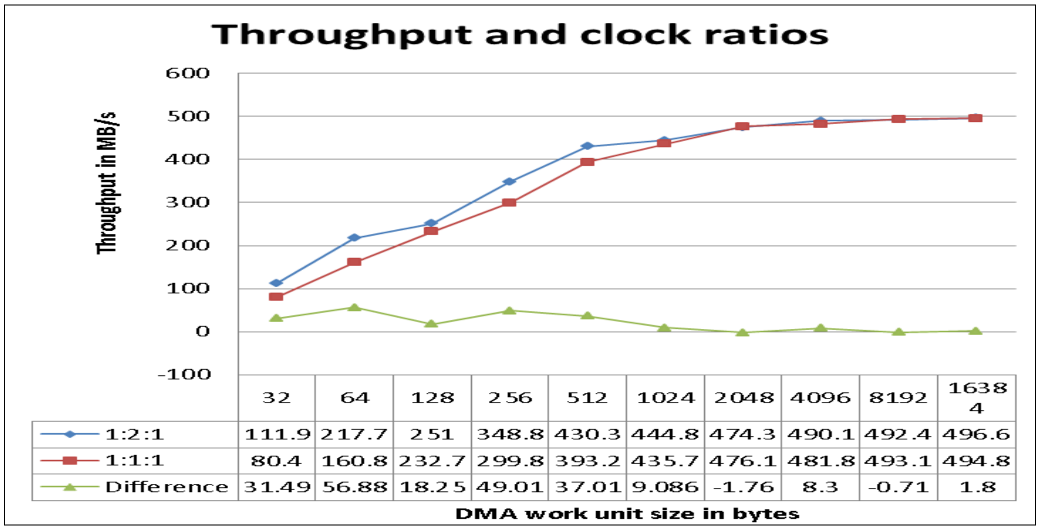
SCB slots arrangement may be an iterative process and is also application dependent.

SCB and Clock Domains

As shown in Figure 1, different SCB work at different clock domains. The ratio of the various clock domains (`DCLK, SYSCLK, SCLK0, SCLK1 and CCLK`) may also play an important role in the throughput, especially when multiple masters are active in the system.

Figure 5 shows the DMC MDMA read throughput measured for different DMA work unit sizes under the following two conditions:

- `CCLK` = 500 MHz, `DCLK` = 125 MHz, `SYSCLK` = 250 MHz, `SCLK0` = 125 MHz.
  Thus, `DCLK`:`SYSCLK`:`SCLK0` = 1:2:1.
- `CCLK` = 500 MHz, `DCLK` = 125 MHz, `SYSCLK` = 125 MHz, `SCLK0` = 125MHz.
  Thus, `DCLK`:`SYSCLK`:`SCLK0` = 1:1:1.



	32	64	128	256	512	1024	2048	4096	8192	16384
1:2:1	111.9	217.7	251	348.8	430.3	444.8	474.3	490.1	492.4	496.6
1:1:1	80.4	160.8	232.7	299.8	393.2	435.7	476.1	481.8	493.1	494.8
Difference	31.49	56.88	18.25	49.01	37.01	9.086	-1.76	8.3	-0.71	1.8

Figure 5. DMC throughput comparison for different clock ratios

As shown in Figure 5, the measured throughput for clock ratio 1:2:1 is almost always greater than that for clock ratio 1:1:1. The difference seen is significant, especially for smaller work unit sizes (e.g.; around 31 MBytes/sec in the case of a 32 bytes transfer).

In summary, the latency across the SCB varies:

- With the clock domain in which the SCB/master operates.
- With the number of active masters on a shared SCB.
- With the nature of master's connection (dedicated or shared) within the SCB.
- With the number and order of arbitration slots allocated to the master for both read and write channels.

Understanding the System Slaves

Memory Hierarchy

ADSP-BF609 dual core processors contain a hierarchical memory model (L1-L2-L3) similar to that in previous Blackfin devices. Thus, the memory related optimization techniques discussed in *System Optimization Techniques for Blackfin® Processors (EE-324)*[3] apply for ADSP-BF609 processors also. The following sections discuss the access latencies and achievable throughput associated to the different memory levels.

L1 memory runs at core clock speed (`CCLK`) and is the fastest accessible memory in the hierarchy. On the other hand, L2 memory access times are longer since the maximum L2 clock frequency is half the core clock, which equals the System Clock (`SYSCLK`). The L2 memory controller contains two ports to connect to the System Crossbar. Port 0 is a 64-bit wide interface dedicated to core traffic, while port 1 is a 32-bit wide interface that connects to the DMA engine. Each port has a read and a write channel. For more details, refer to the *ADSP-BF60x Processor Hardware Reference*[2].

L2 Memory Throughput

Since L2 runs at `SYSCLK` speed, it's capable of providing a maximum theoretical throughput of 250 MHz*4 = 1000 MBytes/sec in one direction. Since there are separate read and write channels, the total throughput in both directions equals 2000 MBytes/sec. However, in practice, the maximum achievable DMA throughput in one direction is less than 1000 MBytes/sec (around 880 MBytes/sec), because of the system interconnect and other internal overhead. Also, in order to be able to operate L2 SRAM memory at its optimum throughput, both 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 bytes) by the L2 memory controller. Thus, in order to achieve optimum throughput for DMA access to L2 memory, the DMA channel `MSIZE` should be configured to 8 bytes or higher. When performing a single DMA channel access to L2 memory configuring `MSIZE` to 8 bytes proves to be sufficient, however when multiple DMA accesses to L2 memory are performed, `MSIZE` of 16 or 32 bytes (depending upon the DMA FIFO depth) results in better throughput as it reduces the latency effect when arbitrating between different DMA channels.

One important point to note about accesses to L2 memory is that the throughput for sequential and non-sequential accesses is the same and does not vary, as it's the case when accessing L3 memories via the Dynamic Memory Controller (DMC). More details on this are provided later.

[Figure 6](#) provides details on the throughput achieved using MDMA0 and MDMA1 for transmitting data from L2 to L1 memory, using different `MSIZE` settings for work unit sizes of 2 KBytes and 16 KBytes.

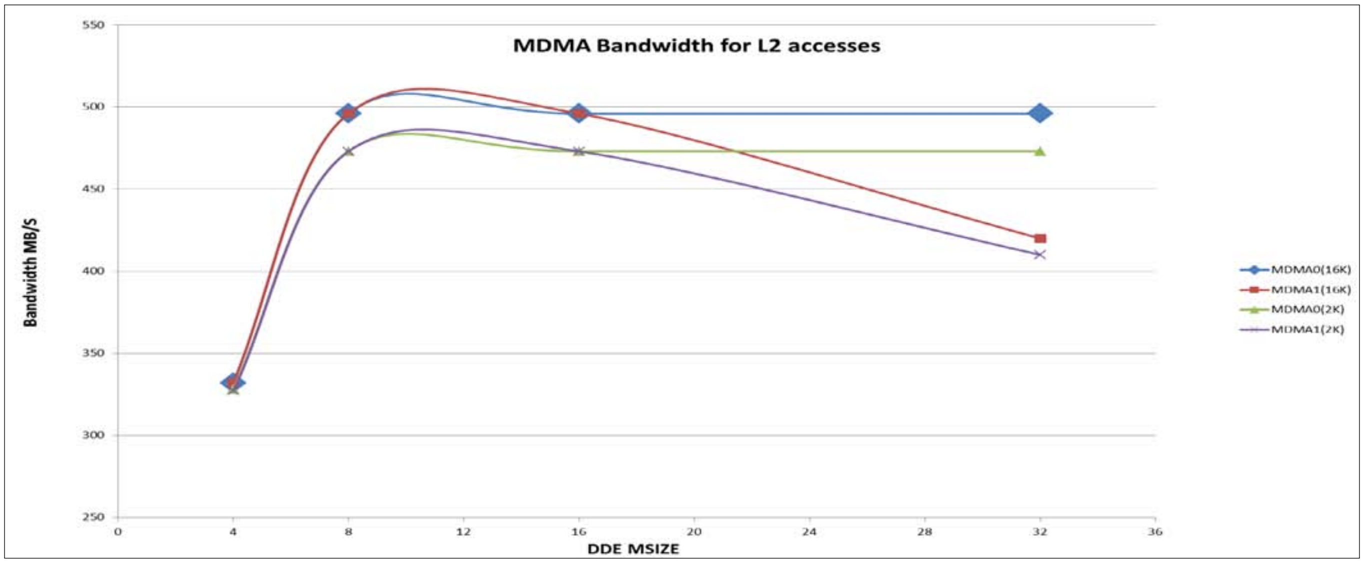


Figure 6. L2 MDMA read throughput for different MSIZE and work unit size values

As shown in Figure 6, the achieved throughput is very close to 500 Mbytes/sec for all `MSIZE` settings above 8 bytes, but drops significantly for `MSIZE` of 4 bytes. L2 throughput drops drastically for 8-bit and 16-bit accesses. This is partly due to the non-utilization of the 32-bit wide DMA bus and the additional internal overheads.

Furthermore, the graph shows a dip in throughput when setting MDMA1 `MSIZE` to 32 bytes. Remember, that as highlighted earlier in this document, MDMA1 provides a smaller FIFO depth, thus the reduced throughput.

Lastly, note that although not shown, the MDMA throughput for write accesses and its variation with work unit size and MDMA channel used is very similar to read accesses.

L2 write throughput with ECC enabled

ADSP-BF609 processors features up to 256 KBytes of L2 SRAM which is ECC-protected and organized in 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 in terms of 32 bits, such that any writes that are less than 32 bits wide to an ECC-enabled SRAM bank will be implemented as a read-followed by a write, and will require 3 cycles to complete (2 cycles for read, 1 cycle for write).

Table 5 shows L2 memory throughput for 8 and 16 bit accesses for both ECC-enabled and disabled.

Data Size (KBytes)	8-bit writes throughput (MBytes/sec)		16-bit writes throughput (MBytes/sec)	
	ECC disabled	ECC enabled	ECC disabled	ECC enabled
2	77.9	70.9	155.5	141.1
4	78.2	71.2	155.9	141.9
8	78.3	71.3	156.4	142.4
16	78.3	71.3	156.7	142.6

Table 5. Throughput for write accesses to L2 memory with ECC enabled and disabled

The measured throughput for 8 and 16-bit accesses with ECC enabled drops compared with ECC disabled. Although ECC-enabled accesses to L2 memory add latency (2 `SYSCLK` cycles), there is no equivalent drop in throughput. The measured drop is not large because the additional latency incurred due to ECC is masked to some extent behind the latency introduced by the system interconnect for DMA accesses of smaller `MSIZE`.

For instance, for the maximum theoretical throughput for an 8-bit DMA access with ECC disabled (125 MBytes/sec), the maximum achievable throughput is 78 MBytes/sec, due to the SCB inefficiencies involved in 8- and 16-bit accesses.

Note that ECC does not affect L2 memory read access throughput.

Furthermore, it is possible to access L2 memory at a higher throughput by combining DMA accesses from multiple SCBs. Table 6 shows bench measurements for the peak throughput achieved during L2 memory accesses by concurrently running multiple high bandwidth DMA channels. Tests were performed under following conditions:

- EPPI0 configured as receiver at 62.5 MHz in 24-bit mode with data packing enabled
- EPPI1 and EPPI2 configured as receiver at 62.5 MHz in 16-bit mode with data packing enabled
- MDMA0 stream reading data from L1 and writing to L2 memory
- Link port 0 and link port 1 running in loopback mode with link port 1 writing to L2 memory

Writes to L2 Memory Throughput (MBytes/sec)							
No of Bytes	**MDMA0**	**EPPI0**	**EPPI1**	**EPPI2**	**Link Port**	**Total**	**DMA Destination MSIZE**
16K	228	186	123	123	60	660	8 bytes
16K	303	186	123	123	60	795	16 bytes
16K	383	186	123	123	60	875	32 bytes

Table 6. Throughput for multiple DMA write accesses to L2 memory

Note that since the L2 memory controller provides separate ports for read and write DMA accesses, for data transfers to L2 memory in both directions, DMA throughputs of more than 1.5 GBytes/sec can be achieved.

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 core and DMA access the same bank, the best access rate that 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 L2CTL_RPCR.RPC1 and L2CTL_WPCR.WPC1 bits to 1.

L3/External Memory Throughput

ADSP-BF609 Processors provide interfaces for connecting to different types of off-chip L3 memory devices, such as the Static Memory Controller (SMC) for interfacing to parallel SRAM/Flash devices, and the Dynamic Memory Controller (DMC) for connecting to DRAM (DDR2/LPDDR) devices.

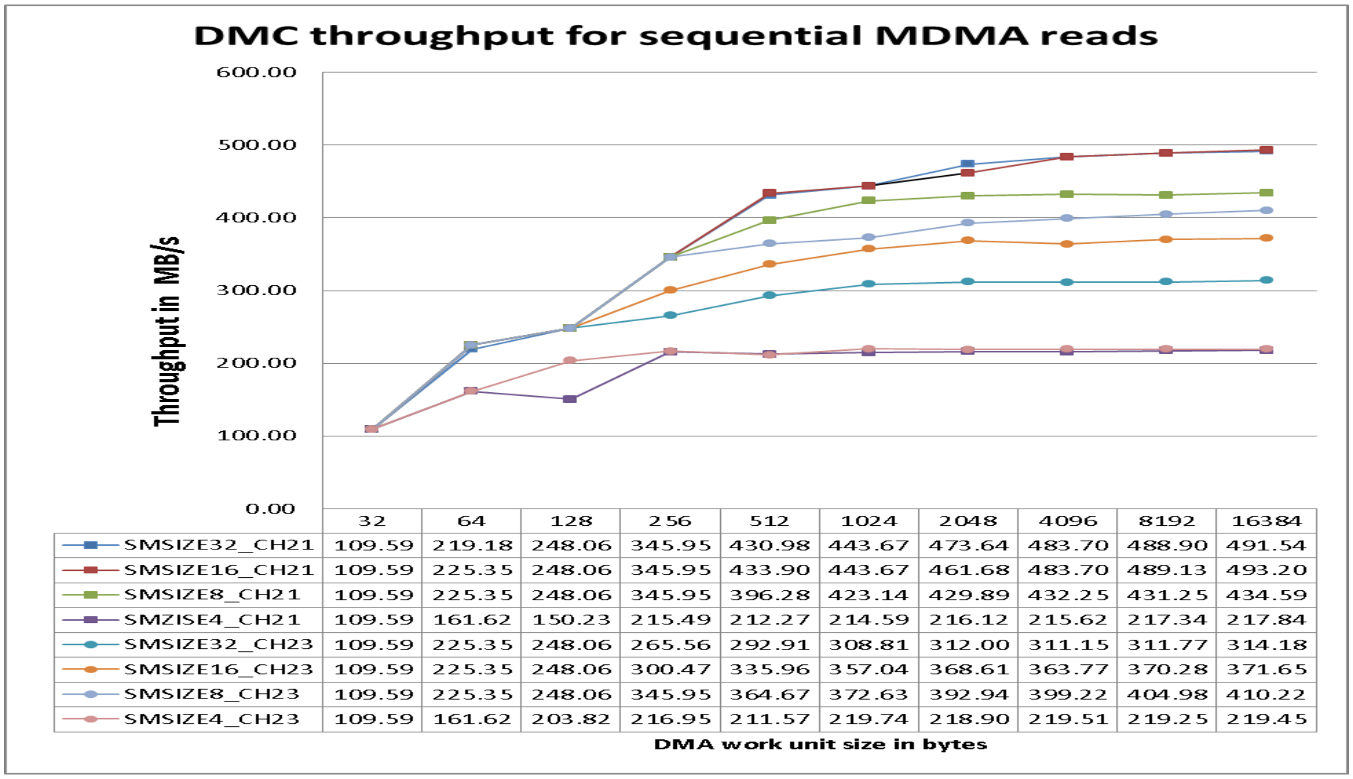
The DMC interface operates at speeds of up to 250 MHz. Thus, for the 16-bit DDR2 interface, the maximum theoretical throughput which the DMC can deliver equals 1 GByte/sec. However, the practical maximum DMC throughput is less because of the latencies introduced by the internal system interconnects as well as the latencies derived from the DRAM technology itself (access patterns, page hit to page miss ratio, etc.).

Techniques for achieving optimum throughput when accessing L3 memory via the DMC are discussed next. 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.

MDMA channels can request DMC accesses faster than any other masters. It has to be noted that all the MDMA channels belong to SCB4 which works on SCLK0 domain. The maximum SCLK0 value possible is 125 MHz. Thus, with 32-bit peripheral bus width, the maximum possible throughput using only MDMA channel(s) equals 125 MHz * 4 = 500 MBytes/sec. 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, FIFO depth of the MDMA channel being used, the block size of the transfer, and DMA parameters (such as MSIZE).

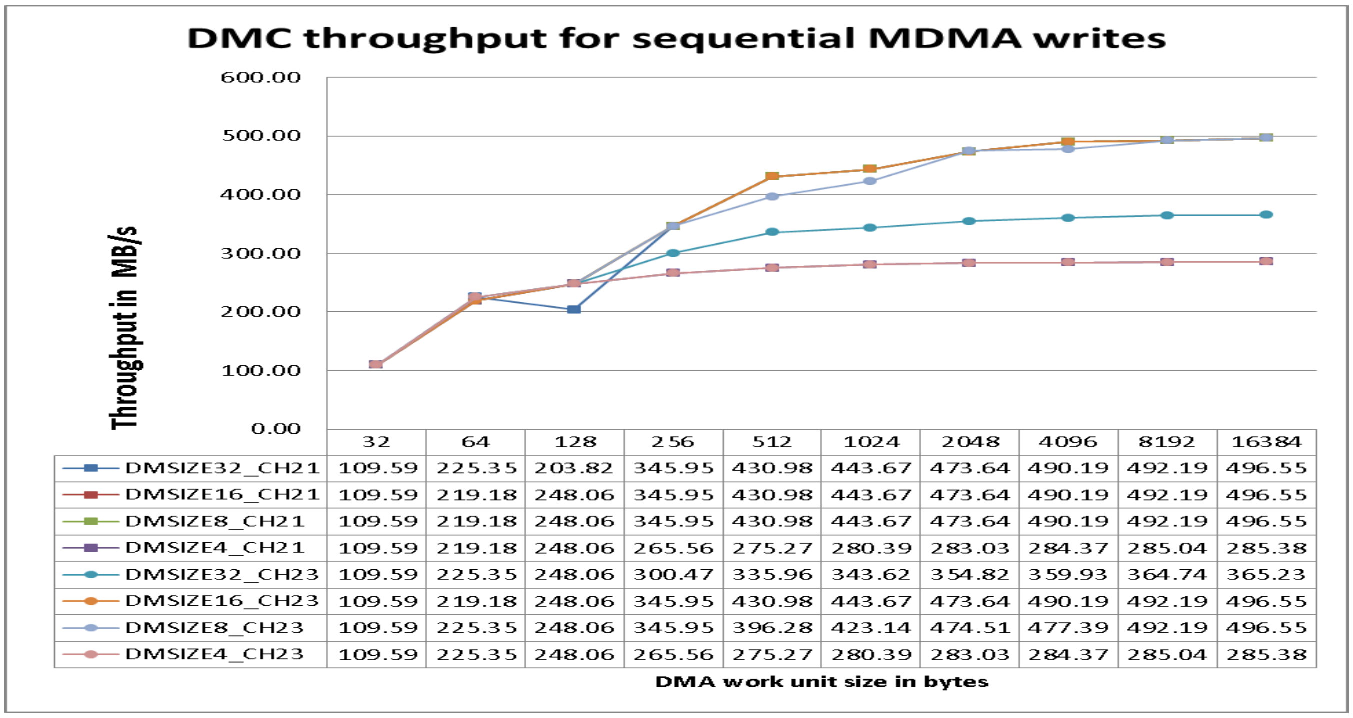
Figure 7 and Figure 8 provide the DMC measured throughput using MDMA channels 21 (FIFO depth of 128 Bytes) and 23 (FIFO depth of 64 Bytes) for various MSIZE values and for different buffer sizes, between L1 and L3 memories for sequential read and write accesses respectively.

Also, note that the lines with square markers belong to DMA channel 21, whereas the lines with round markers belong to DMA channel 23.



	32	64	128	256	512	1024	2048	4096	8192	16384
■ SMSIZE32_CH21	109.59	219.18	248.06	345.95	430.98	443.67	473.64	483.70	488.90	491.54
■ SMSIZE16_CH21	109.59	225.35	248.06	345.95	433.90	443.67	461.68	483.70	489.13	493.20
■ SMSIZE8_CH21	109.59	225.35	248.06	345.95	396.28	423.14	429.89	432.25	431.25	434.59
■ SMZISE4_CH21	109.59	161.62	150.23	215.49	212.27	214.59	216.12	215.62	217.34	217.84
● SMSIZE32_CH23	109.59	225.35	248.06	265.56	292.91	308.81	312.00	311.15	311.77	314.18
● SMSIZE16_CH23	109.59	225.35	248.06	300.47	335.96	357.04	368.61	363.77	370.28	371.65
● SMSIZE8_CH23	109.59	225.35	248.06	345.95	364.67	372.63	392.94	399.22	404.98	410.22
● SMSIZE4_CH23	109.59	161.62	203.82	216.95	211.57	219.74	218.90	219.51	219.25	219.45

Figure 7. DMC measured throughput for sequential MDMA reads



	32	64	128	256	512	1024	2048	4096	8192	16384
■ DMSIZE32_CH21	109.59	225.35	203.82	345.95	430.98	443.67	473.64	490.19	492.19	496.55
■ DMSIZE16_CH21	109.59	219.18	248.06	345.95	430.98	443.67	473.64	490.19	492.19	496.55
■ DMSIZE8_CH21	109.59	219.18	248.06	345.95	430.98	443.67	473.64	490.19	492.19	496.55
■ DMSIZE4_CH21	109.59	219.18	248.06	265.56	275.27	280.39	283.03	284.37	285.04	285.38
● DMSIZE32_CH23	109.59	225.35	248.06	300.47	335.96	343.62	354.82	359.93	364.74	365.23
● DMSIZE16_CH23	109.59	219.18	248.06	345.95	430.98	443.67	473.64	490.19	492.19	496.55
● DMSIZE8_CH23	109.59	225.35	248.06	345.95	396.28	423.14	474.51	477.39	492.19	496.55
● DMSIZE4_CH23	109.59	225.35	248.06	265.56	275.27	280.39	283.03	284.37	285.04	285.38

Figure 8. DMC measured throughput for sequential MDMA writes

The following important observations can be made from the above plots:

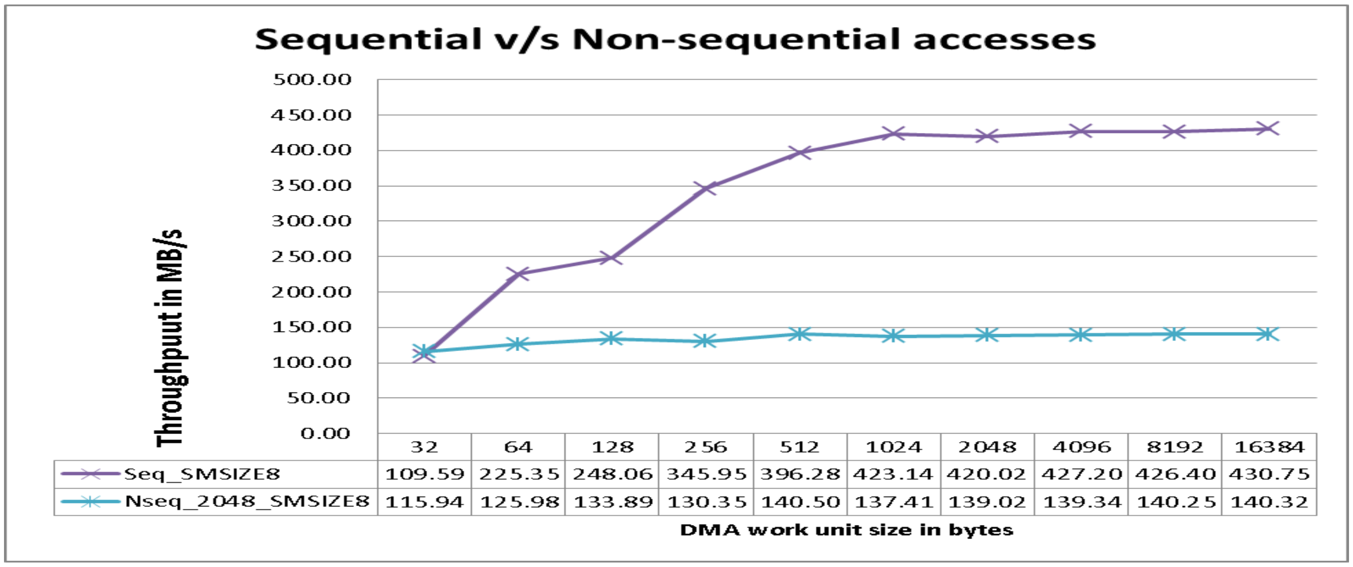
- The throughput trends are very similar for reads and writes with regards to `MSIZE` and buffer size values.
- The throughput depends largely upon the DMA buffer size. For smaller buffer sizes, the throughput is significantly lower, increasing up to 500 MBytes/sec with a larger buffer size. For instance, DMA channel 21 with `MSIZE` of 32, and buffer size of 32 Bytes, the read throughput is 109.6 MBytes/sec (22%), whereas it reaches 491.5 MBytes/sec (98.3%) for a 16 KBytes 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 DMC controller.

Try to rearrange the DMC accesses such that the DMA count is as large as possible. In other words, better sustained throughput is obtained for continuous transfers over time.

- The throughput also depends upon the FIFO depth of the MDMA channel being used. For example, for `MSIZE` of 32 and buffer size of 16384 Bytes, the read throughput for MDMA channel 21 (FIFO depth of 128 Bytes) is 491.54 MBytes/sec, while for MDMA channel 23 (FIFO depth of 64 Bytes), it equals 314.2 MBytes/sec. Clearly, the internal architecture is more efficient with larger FIFO sizes.
- To some extent, throughput also depends upon the `MSIZE` value of the source MDMA channel for reads, and destination MDMA channel for writes. As shown in Figure 7 and Figure 8, in most cases, greater `MSIZE` values provide better results. Ideally, `MSIZE` value should be at least equal to the DDR2 memory burst length. That is, for MDMA channel 21, buffer size of 16384 Bytes, the read throughput is 491.5 MBytes/sec (98.3%) for `MSIZE` of 32, while it reduces to almost half (217.8 MBytes/sec - 43.6%) for `MSIZE` of 4. This is due to the fact that, for `MSIZE` 4, although all accesses are still sequential, the full DDR2 memory burst length of 8 Bytes (4 16-bit words) is not used.

As explored, for sequential reads, it is easily possible to achieve optimum throughput, particularly for larger buffer size. This is because 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 case of non- sequential accesses, throughput may drop slightly or significantly depending upon the page hit-to-miss ratio.

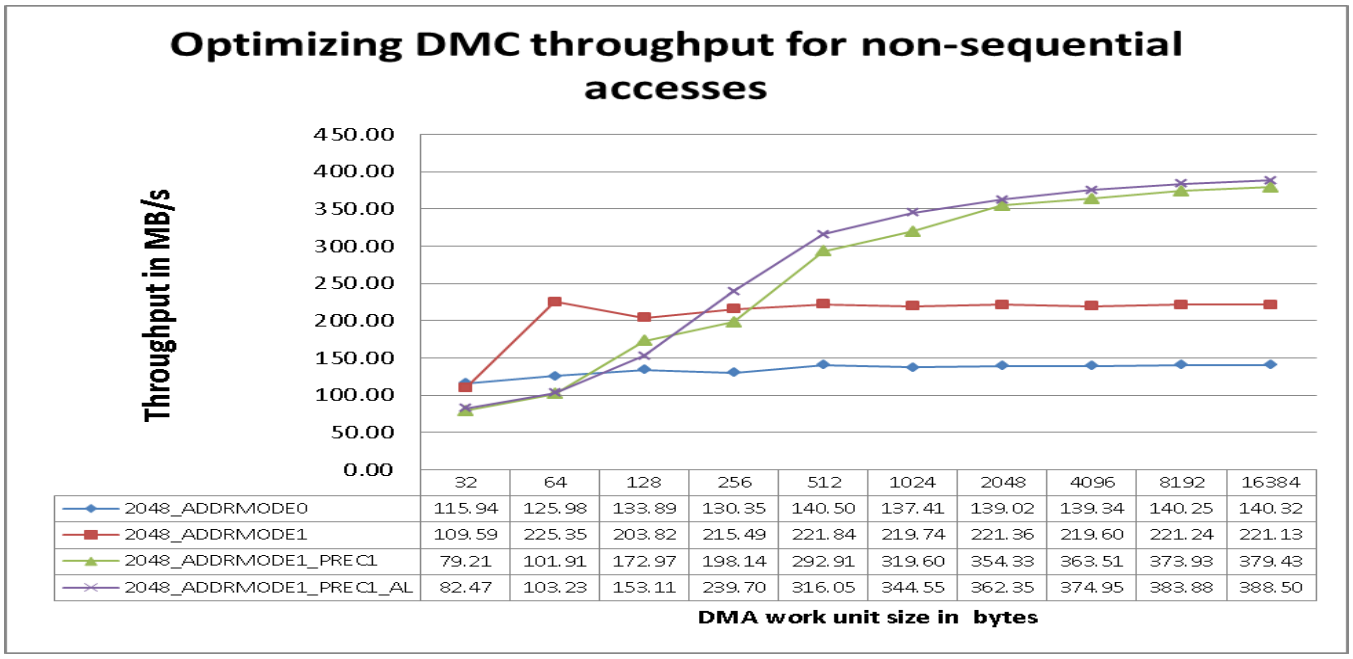
Figure 9 provides a comparison of the DMC throughput numbers measured for sequential MDMA read accesses for `MSIZE` 8 Bytes (equals DDR2 burst length), and `ADDRMODE` set to 0 (bank interleaving) versus non-sequential accesses with a modifier of 2048 Bytes (equals DDR2 page size, thus leading to a worst case scenario with maximum possible page misses). As shown, for buffer size of 16384 Bytes, throughput drops significantly from 430.75 to 140.3 MBytes/sec.



	32	64	128	256	512	1024	2048	4096	8192	16384
Seq_SMSIZE8	109.59	225.35	248.06	345.95	396.28	423.14	420.02	427.20	426.40	430.75
Nseq_2048_SMSIZE8	115.94	125.98	133.89	130.35	140.50	137.41	139.02	139.34	140.25	140.32

Figure 9. DMC throughput for non-sequential read accesses compared with sequential accesses

DDR2 memory devices 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 into page misses. By setting `DMC_CTL.1ADDRMODE` bit, throughput can be improved by making sure that such accesses fall into different banks. For instance, Figure 10 shows in red, how the DMC throughput increases from 140.32 MBytes/sec to 221.13 MBytes/sec by just setting this bit for the non-sequential access pattern shown in Figure 9.



	32	64	128	256	512	1024	2048	4096	8192	16384
2048_ADDRMODE0	115.94	125.98	133.89	130.35	140.50	137.41	139.02	139.34	140.25	140.32
2048_ADDRMODE1	109.59	225.35	203.82	215.49	221.84	219.74	221.36	219.60	221.24	221.13
2048_ADDRMODE1_PREC1	79.21	101.91	172.97	198.14	292.91	319.60	354.33	363.51	373.93	379.43
2048_ADDRMODE1_PREC1_AL	82.47	103.23	153.11	239.70	316.05	344.55	362.35	374.95	383.88	388.50

Figure 10. Optimizing throughput for non-sequential accesses

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 with the help of the *Read with Auto Precharge* command. This allows the row of a bank to proactively precharge after it has been accessed, helping improve the throughput by saving the latency involved in precharging the row at the time when the next row of the same bank has to be activated.

This is illustrated by the green line in Figure 10. Note how the throughput increases from 221.13 MBytes/sec to 379.43 MBytes/sec by just setting the `DMC_CTL.PREC` bit. The same result can be achieved by setting the `DMC_EFFCTL.PRECBANK[7-0]` bits. This feature can be used on per bank basis. However, it should be noted that setting the `DMC_CTL.PREC` bit overrides the `DMC_EFFCTL_PRECBANK[7-0]` bits. Also, setting the `DMC_CTL.PREC` bit results in precharging of the rows after every read burst, while setting the `DMC_EFFCTL_PRECBANK[7-0]` bits pre-charge the row after the last burst corresponding to the respective `MSIZE` settings. This can provide an added throughput advantage for cases where `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 11 and Figure 12 (TN4702[5]).

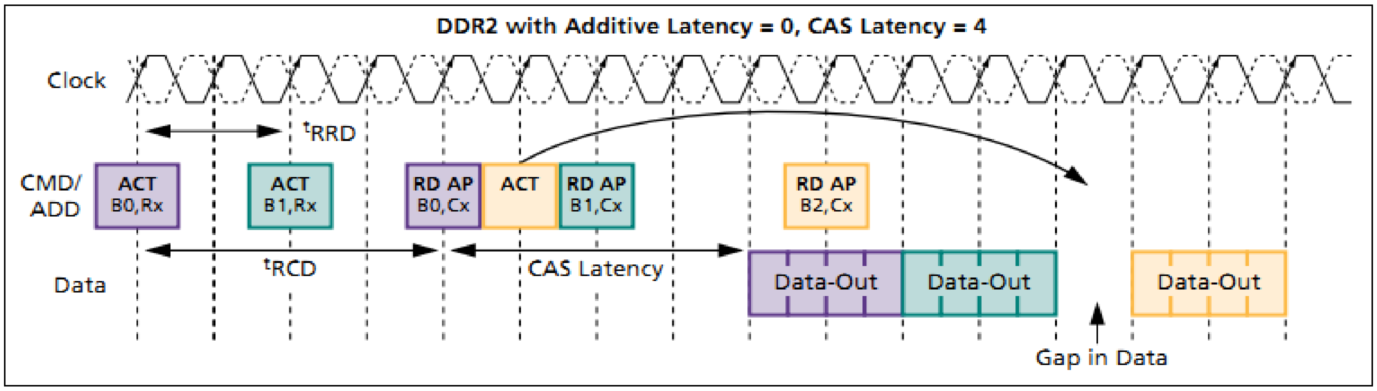


Figure 11. DDR2 reads without additive latency

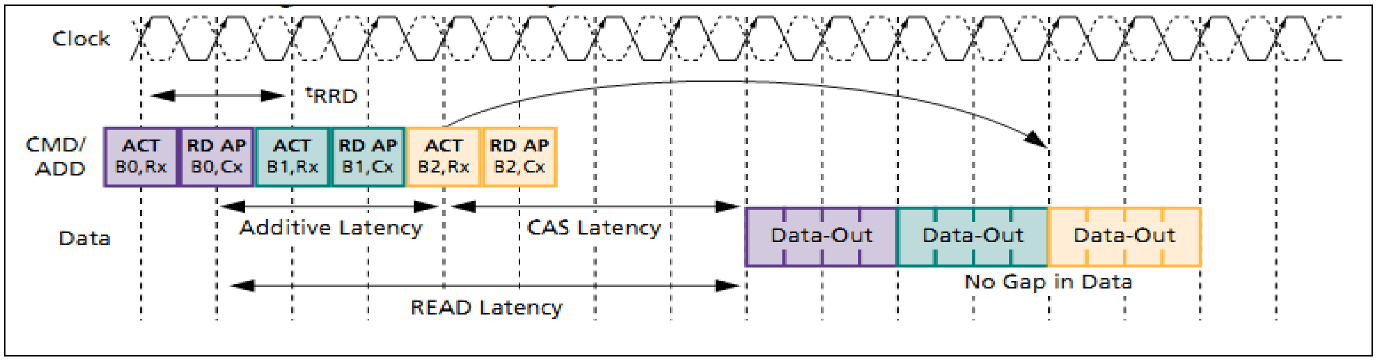


Figure 12. DDR2 reads with additive latency

Programming the additive latency to `tRCD-1` allows the DMC to send the *Read with Autoprecharge* command right after the *Activate* command, before `tRCD` completes. 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 10 shows how the throughput improves from 379.43 MBytes/sec to 388.5 MBytes/sec by programming the additive latency (*AL*) in the `DMC_EMR1` register to `tRCD-1` (equals 3 in this case).

The Dynamic Memory Controller also allows elevating the priority of the accesses requested by a particular SCB master with the help of `DMC_PRIO` and `DMC_PRIOMSK` registers. The associated `.ZIP` file provides an example code in which two MDMA DMC read channels (21 and 25) run in parallel. Table 7 summarizes the measured throughout.

For test case 1, when no priority is selected, the throughput for MDMA channel 21 is 245 MBytes/sec, while for channel 21 equals 228 MBytes/sec. MDMA channel 21 throughput increases to 270 MBytes/sec by setting the `DMC_PRIO` register to the corresponding SCB ID (`0x0C`) and the `DMC_PRIOMSK` register to `0xFFFF`. Similarly, MDMA channel 25 throughput increases to 260 MBytes/sec by setting the `DMC_PRIO` register to the corresponding SCB ID (`0x4C`).

Test Case Number	**Priority channel (SCB ID)**	**MDMA0 (Ch.21) Throughput (MBytes/sec)**	**MDMA2 (Ch.25) Throughput (MBytes/sec)**
1	None	244.68	228.21
2	MDMA0(0x0C)	269.97	236.54
3	MDMA2(0x4C)	236.11	259.57

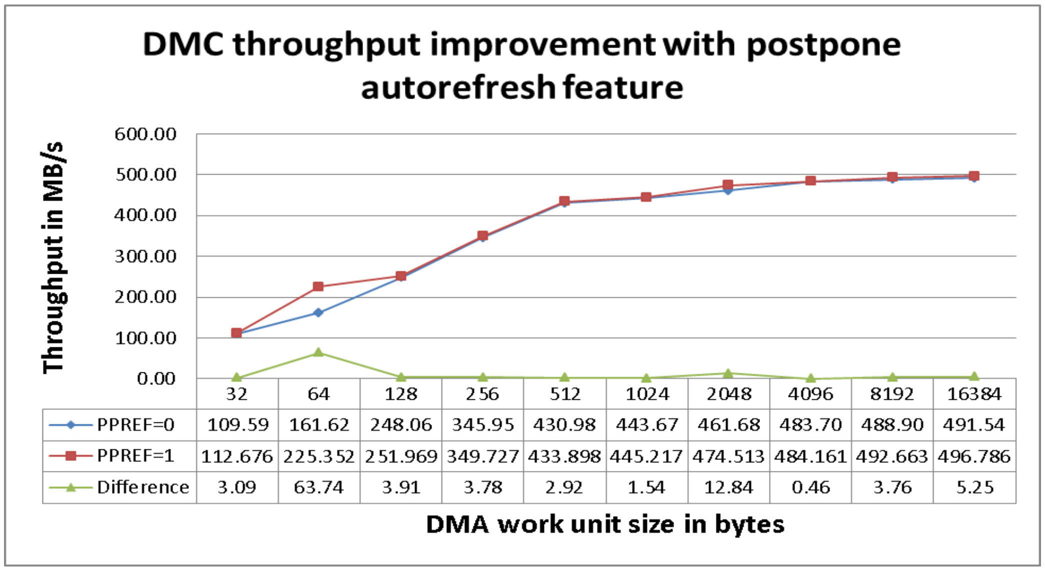
Table 7. DMC measured throughput for different DMC_PRIO settings for MDMA channels 21 and 25

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 `NUM_REF` bit in the `DMC_EFFCTL` register.

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 given out only after that DMC enters self-refresh mode.

Figure 13 shows the measured throughput for a MDMA stream reading from via 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). A slight throughput improvement of about 1-2% can be seen.



	32	64	128	256	512	1024	2048	4096	8192	16384
PPREF=0	109.59	161.62	248.06	345.95	430.98	443.67	461.68	483.70	488.90	491.54
PPREF=1	112.676	225.352	251.969	349.727	433.898	445.217	474.513	484.161	492.663	496.786
Difference	3.09	63.74	3.91	3.78	2.92	1.54	12.84	0.46	3.76	5.25

Figure 13. DMC throughput optimization with Postpone Autoreresh feature

As mentioned earlier, it is possible to run the DMC controller at throughput greater than 500 MBytes/sec by running more than one peripheral DMA or MDMA stream belonging to different SCBs in parallel. For instance, running a MDMA0 stream (SCB4) and two EPPIs (both in SCB5, with EEPI0 in 24-bit mode and EPPI1 in 16-bit mode at 62.5 MHz `EPPICLK` speed), the total throughput can be as high as 812.5 MBytes/sec (500+187.5+125).

System MMR Latencies

Unlike previous Blackfin processors, ADSP-BF609 processors may have higher MMR access latencies. This is mainly due to the interconnect fabric between the core and the MMR space, and also due to the number of different clock domains (SYSCLK, SCLK0 and SCLK1). With this, ADSP-BF609 processors have three groups of peripherals for which the MMR latency varies.

Group 1		Group 2	Group 3
PWM	EMAC	TIMER	TRU
WDOG	EPPI	CAN	SEC
PINT	PVP	TWI	SWU
GPORT	VID	ROT	RCU
Pin-mux	ACM	SPORT	DPM
SWU	SMC	SPI	SPU
UART		RSI	CGU
EPPI		LP	
SDU		CRC	

Table 8. Peripheral groups based on variable MMR latency

Table 9 shows the MMR latency numbers for the different group of peripherals as defined in Table 8 . Across the group, the peripherals should observe the same kind of MMR access latencies. However, the MMR access latencies may vary depending upon:

- Clock ratios. All MMR accesses are through SCB0 which is on the `SYSCLK` domain, while peripherals are on the `SCLK0/1` domain. Thus, the `CCLK:SYSCLK:SCLK0:SCLK1` ratio will affect the MMR latency. For example, a ratio of 4:2:1:1 (`CCLK:SYSCLK:SCLK0:SCLK1`) is optimum for minimal MMR latency
- Number of concurrent MMR access requests in the system. Although a single write incurs half the system latency compared to back-to-back writes, the latency observed on the core will be shorter. Similarly, the system latency incurred by a read followed by a write, or vice versa, will be different to that observed on the core.

MMR Access	**Group 1**	**Group 2**	**Group 3**
MMR Read	52 CCLK (104 ns)	72 CCLK (144 ns)	38 CCLK (76 ns)
MMR Write	49 CCLK (98 ns)	73 CCLK (146ns)	37 CCLK (74ns)

Table 9. MMR access latency numbers (approximate)

These MMR latency numbers are for back to back read/write accesses as observed on the Core and have been calculated using C code, so numbers may vary from case to case.

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 14 provides a flow chart of a typical system bandwidth optimization procedure for ADSP-BF609-based applications.

A typical procedure for an SCB slave includes the following steps:

- Identify the individual and total throughput requirements by all the masters accessing the corresponding SCB slave in the system and allocate the corresponding read/write SCB arbitration slots accordingly. Let’s consider the total throughput requirement as *X*.
- Calculate the observed throughput the corresponding SCB slave(s) may be able to supply under the specific conditions. Let’s refer to this as *Y*.
- For *X*<*Y*, bandwidth requirements are met. However, for *X*>*Y*, one or more peripherals are likely to hit reduced throughput or underflow condition. In such a case, try to apply the bandwidth optimization techniques as discussed in earlier sections 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 no, then slow down the peripheral to reduce the bandwidth requested by the peripheral.

- Reanalyzing whether a particular MDMA or any other memory pipe DMA (such as PVP or PIXC) can be slowed down. The corresponding `DMAx_BWLCNT` register can be used to limit the bandwidth of that particular DMA channel, as discussed earlier.

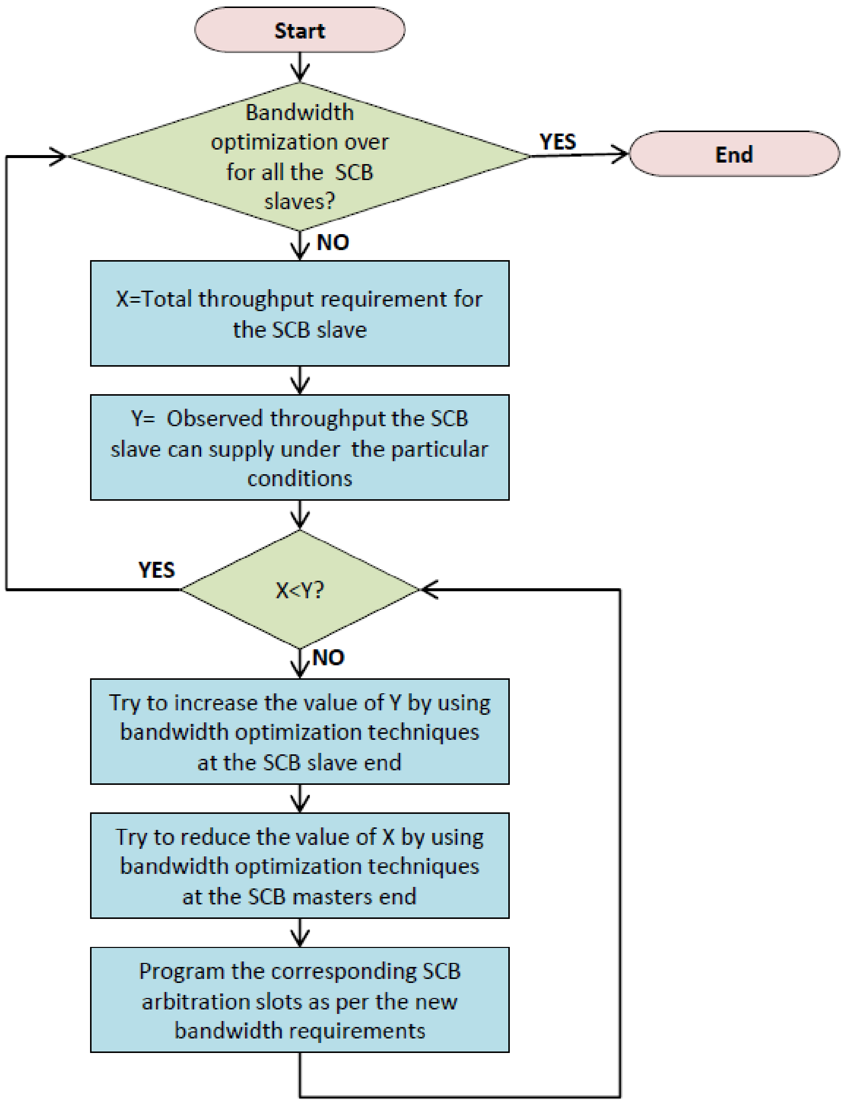

Start
Bandwidth optimization over for all the SCB slaves?
YES
End
NO
X=Total throughput requirement for the SCB slave
Y= Observed throughput the SCB slave can supply under the particular conditions
YES
X<Y?
NO
Try to increase the value of Y by using bandwidth optimization techniques at the SCB slave end
Try to reduce the value of X by using bandwidth optimization techniques at the SCB masters end
Program the corresponding SCB arbitration slots as per the new bandwidth requirements

Figure 14. Typical system bandwidth optimization procedure

Application Example

Figure 15 shows a block diagram of the example application code supplied with this EE-Note.

Note that the idea behind the code supplied is to stress the SCBs and the DMC controller for high throughput requirements to show various bandwidth optimization techniques. It does not depend on pin multiplexing limitations. For example, practically, it is not possible to run EPPI2 along with link ports 0 and 1 due to multiplexing, but the code enables them both at a time.

The example code involves the following conditions:

- `CCLK` = 500 MHz, `DCLK` = 250 MHz, `SYSCLK` = 250MHz, `SCLK0` = `SCLK1` = 125 MHz
- SCB5:
  - EPPI0: 24-bit transmit, internal clock and frame sync mode at `EPPICLK` = 62.5 MHz. Required throughput = 62.5 MHz * 3 = 187.5 MBytes/sec.
  - EPPI2: 16-bit transmit, internal clock and frame sync mode at `EPPICLK` = 62.5 MHz. Required throughput = 62.5 MHz * 2 = 125 MBytes/sec).
  - Total required throughput by SCB5 from SCB0 (MI0) = 187.5 + 125 = 312.5 MBytes/sec.
- SCB6:
  - EPPI1: 16-bit transmit, internal clock and frame sync mode at `EPPICLK` = 62.5 MHz. Required throughput by SCB6 from SCB0 (MI0) = 62.5 MHz * 2 = 125 MBytes/sec.
- SCB3:
  - Link Port 1 (transmitter) and Link Port 0 (receiver) operating in loopback mode with `LPCLK` = 62.5 MHz. Required throughput by SCB3 from SCB0(MI0) = 62.5MBytes/sec.
- SCB4:
  - MDMA0 channels (21-22) and MDMA1 channels (25-26).
  - Throughout requirements depend upon the corresponding `DMAx_BWLCNT` register values. If not programmed, the MDMA channels request for the bandwidth with full throttle (every `SCLK` cycle). This means that both MDMA channels request from the SCB4 an individual throughput of 125 MBytes/sec * 4 = 500MBytes/sec.
  - However, since SCB4 runs at `SCLK` itself, the total maximum theoretical bandwidth request by SCB4 from SCB0 (MI0) is limited to 125 MBytes/sec * 4 = 500MBytes/sec.

As shown in Figure 15, the total required throughput from the DMC controller equals 1GBytes/sec (312.5+125+62.5+500). Theoretically, with `DCLK` = 250 MHz and a bus width of 16 bits, it should be possible to meet this requirement. But, practically, 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.
- The internal latencies involved when switching between masters accessing the DMC.

For this reason, there is a possibility that one or more masters may not get the required bandwidth. For masters, such as MDMA and the link ports, this might result in decreased throughput. While for some other masters, such as EPPI, it might result in an underflow condition.

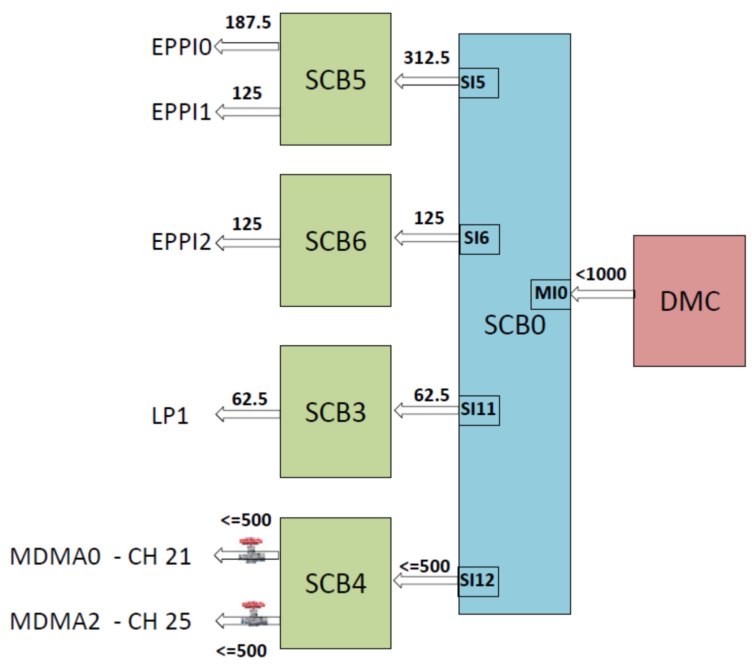


Figure 15. Example application showing DMC throughput distribution across various SCBs

Table 10 shows the expected and measured throughput for all DMA channels and the corresponding SCBs at various steps of bandwidth optimization, as discussed next.

Step 1

In this step, all DMA channels run without applying any optimization techniques. To replicate the worst case scenario, the source buffers of all DMA channels are placed in a single DDR2 SDRAM bank. The row corresponding to "No optimization" in Table 10 shows the measured throughput numbers under this condition. As illustrated, the individual measured throughput of almost all channels is significantly less than expected and all EPPIs show an underflow condition.

- Total expected throughput from the DMC (X) = 1000 MBytes/sec
- Effective DMC throughput (Y) = 485.78 MBytes/sec

Clearly, X is greater than Y, showing a definite need for implementing the corresponding bandwidth optimization techniques.

Step 2

As described earlier, in case of frequent DDR2 SDRAM page misses within the same bank, throughput significantly drops. Although DMA channel accesses are sequential, since multiple channels try to access the DMC concurrently, page misses are likely to occur. To work around this, source buffers of each DMA channels can be moved to different DDR2 SDRAM banks. This allows parallel accesses to multiple pages of different banks, helping improve Y. "*Optimization at the slave - T1*" in Table 10 provides the measured throughput numbers under this condition. Both, individual and overall throughput numbers increase significantly. The maximum throughput delivered by the DMC (Y) increases to 832.32 MBytes/sec (71.33 % improvement from step 1). However, since X is still greater than Y, the measured throughput is still lower than expected, resulting in an underflow condition in EPPI0.

Step 3

There is not much more room to significantly increase the value of *Y*. Alternatively, optimization techniques can be employed at the master end. One way could be by reducing the overall expected throughput to less than 832 MBytes/sec. Thus, for this, the bandwidth of MDMA0 and MDMA2 streams can be limited to 160 MBytes/sec, decreasing SCB4 total required throughput from 500 MBytes/sec to 320 MBytes/sec. *"Optimization at the master - T2"* in Table 10 shows the measured throughput under this condition. As it can be seen, both the individual and the overall and the expected throughput are very close to each other, and no underflow conditions exist.

Step 4

Let's assume it's desired to increase MDMA0 and MDMA2 throughput to 175 MBytes/sec. In this case, we can steal the required bandwidth from the link port DMA. For this, the link port clock speed could be lowered from 62.5 MHz to 31.25 MHz in such a way that the overall expected throughput remains almost the same (i.e.; 818.75 MBytes/sec). *"Optimization at the master - T3" in* Table 10 provides the measured throughput numbers under this condition. Both the individual and the overall and the expected throughput are very close to each other and no underflow conditions are observed.

S.No.	Condition	SCB5						SCB6			
		PPI0		PPI2		Total		PPI1		LP1	
		R	M	R	M	R	M	R	M	R	M
1	**No optimization**	187.5	150.2	125.0	125.2	312.5	275.4	125.0	74.3	62.5	44.1
2	**Optimization at slave - T1**	187.5	188.0	125.0	124.0	312.5	312.0	125.0	124.0	62.5	62.1
3	**Optimization at master - T2**	187.5	187.9	125.0	124.0	312.5	311.9	125.0	124.0	62.5	62.5
4	**Optimization at master - T3**	187.5	188.9	125.0	125.4	312.5	314.3	125.0	125.4	31.2	31.3

S.No.	Condition	SCB4						X	Y	PPI Underflow?
		MDMA0		MDMA1		Total				
		R	M	R	M	R	M			
1	**No optimization**	500.0	46.0	500.0	46.0	500.0	92.0	1000	485.8	YES PPI0,1,2
2	**Optimization at slave - T1**	500.0	167.2	500.0	167.0	500.0	334.3	1000	832.3	YES PPI0
3	**Optimization at master - T2**	160.0	152.6	160.0	152.6	320.0	305.2	820.0	803.6	NO
4	**Optimization at master - T3**	175.0	173.6	175.0	173.6	350.0	347.2	818.7	818.2	NO

All units are expressed in MBytes/sec.
X: Total Required Throughput, Y: Measured Throughput, R: Required, M: Measured.

Table 10. Example application system bandwidth optimization steps

System Optimization Techniques – Checklist

This section summarizes the optimization techniques discussed in this application note, while also listing a few additional tips for bandwidth optimization.

- 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 `MSIZE` parameters to optimum values to maximize throughput and avoid any potential underflow/overflow conditions.
- Calculate and program the appropriate number of SCB arbitration slots as per the throughput requirements.
- While assigning the slots to different masters, also take into consideration the ordering of the slot assignments.
- 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
- Use of peripheral DMA with MDMA instead of multiple MDMA's concurrently will affect the throughput as the access will be through different SCBs instead of the same SCB.
- Using the PVP in memory pipe mode to offer the same functionality as MDMA.
- Since MMR latencies affect the interrupt service latency, ADSP-BF609 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 (triggers receivers), thereby off-loading processing from the core. For a detailed discussion on this, refer to *Utilizing the Trigger Routing Unit for System Level Synchronization (EE-360)*[4].
- Utilize multiple cores to partition critical and non-critical activity to compensate the effect of MMR latency on the overall execution time.

References

[1] *ADSP-BF606/ADSP-BF607/ADSP-BF608/ADSP-BF609 Blackfin Dual Core Embedded Processors Data Sheet.* Rev 0, June 2013. Analog Devices, Inc.

[2] *ADSP-BF60x Blackfin Processor Hardware Reference.* Rev 0.5, February 2013. Analog Devices, Inc.

[3] *System Optimization Techniques for Blackfin Processors (EE-324)*. Rev 1, July 2007. Analog Devices, Inc.

[4] *Utilizing the Trigger Routing Unit for System Level Synchronization (EE-360).* Rev 1, October 2013. Analog Devices Inc

[5] *TN-47-02: DDR2 Offers New Features/Functionality Introduction.* Rev A, June 2006. Micron Technology Inc.

Document History

Revision	**Description**
Rev 1 – November 12, 2013 by Akash Agarwal, Mitesh Moonat, and Nabeel Shah.	Initial release.