Technical notes on using Analog Devices DSPs, processors and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

ADSP-2156x SHARC+ Processor System Optimization Techniques

Contributed by Mitesh Moonat *Rev 2 – September 10, 2020*

Introduction

The ADSP-2156x SHARC+ processor family provides an optimized architecture that supports high system bandwidth and advanced peripherals. This application note discusses the key architectural features of the processor that contribute to the overall system bandwidth, as well as various available bandwidth optimization techniques.

Most of the theoretical content of this application note is the same as in *ADSP-SC5xx/215xx SHARC+ Processor System Optimization Techniques (EE-401)*[1]. This application note includes the figures, tables, and data specific to the ADSP-2156x family of processors.

ADSP-2156x Processor Architecture

This section describes the ADSP-2156x processor's key architectural features that play a crucial role in system bandwidth and performance. For detailed information, refer to the *ADSP-2156x SHARC+ Processor Hardware Reference* [2].

The overall architecture of the ADSP-2156x processors consists of three main system components: system bus slaves, system bus masters, and system crossbars. Figure 1 and Figure 2 show how these components are interconnected to form the complete system.

System Bus Slaves

As shown in Figure 1 (top), system bus slaves include on-chip and off-chip memory devices/controllers, such as L1 SRAM, L2 SRAM, the Dynamic Memory Controller (DMC) for DDR3/DDR3L SDRAM devices, memory-mapped peripherals such as SPI FLASH, 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 runs at CCLK, L2 SRAM runs at SYSCLK, the DMC interface runs at DCLK, and so on.

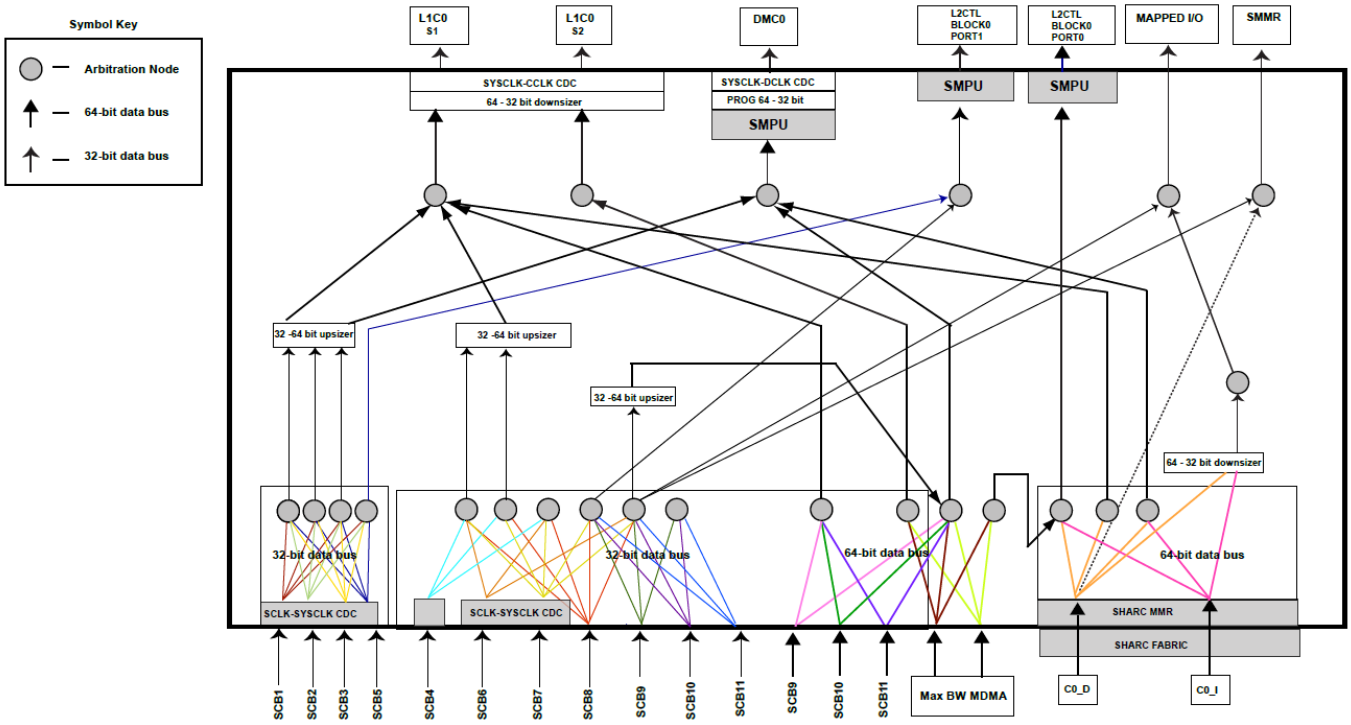


Figure 1: ADSP-2156x System Cross Bar (SCB) Block Diagram

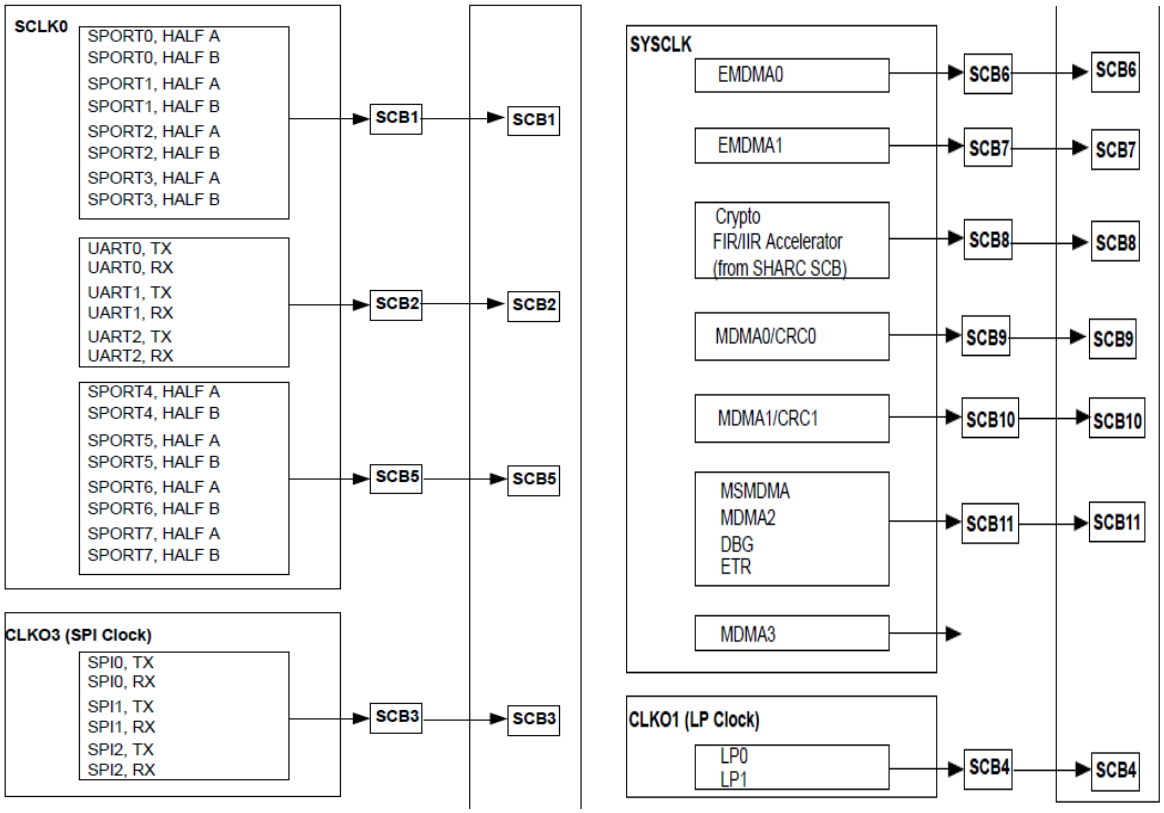


Figure 2: ADSP-2156x SCB Masters Groups

System Bus Masters

The system bus masters are shown at the bottom of Figure 1. They include peripheral Direct Memory Access (DMA) channels such as the Serial Port (SPORT) and Serial Peripheral Interface (SPI). Also included are the Memory-to-Memory DMA channels (MDMA) and the core. Note that each peripheral runs at a different clock speed, and thus has individual bandwidth requirements. For example, high speed peripherals such as the Link Port require higher bandwidth than slower peripherals such as the SPORT or UART.

System Crossbars

The System Crossbars (SCB) are the fundamental building blocks of the system bus interconnect. As shown in Figure 2, 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. SCBs run on different clock domains (SCLK0, SYSCLK, SPI clock, and LP clock) that introduce their own latencies to the system.

System Latencies, Throughput, and Optimization Techniques

The following sections describe various aspects related to latencies and throughput of system bus masters, system bus slaves, and the system cross bars. Various optimization techniques to reduce system latencies and improve throughput are also discussed.

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 (for example, memories), and another bus that connects to either a peripheral or another DMA channel. The SCB/memory bus width can vary among 8, 16, 32, or 64 bits and is defined by the `DMA_STAT.MBWID` bit field. The peripheral bus width can vary among 8, 16, 32, 64, or 128 bits and is defined by the `DMA_STAT.PBWID` bit field. For ADSP-2156x processors, the memory and peripheral bus widths for most of the DMA channels is 32 bits (4 bytes). However, for some channels, it is 64 bits (8 bytes).

The DMA parameter `DMA_CFG.PSIZE` determines the width of the peripheral bus in use. It can be configured to 1, 2, 4, or 8 bytes. However, it cannot be greater than the maximum possible bus width defined by the `DMA_STAT.PBWID` bit field. This restriction exists because burst transactions are not supported on the peripheral bus.

The DMA parameter `DMA_CFG.MSIZE` determines the actual size of the SCB bus in use. It also determines the minimum number of bytes that are 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 the `MSIZE` value.

It is important to understand how to choose the appropriate `MSIZE` value, both from a functionality and a performance perspective. When choosing the `MSIZE` value, consider the following points:

- The start address of the work unit must align to the `MSIZE` value. Failing to do so generates a DMA error interrupt.
- From a performance perspective, use the highest possible `MSIZE` value (32 bytes) for better average throughput. This results in a higher likelihood of uninterrupted sequential accesses to the slave (memory), which is the most efficient for typical memory designs.
- From a performance perspective, the minimum `MSIZE` value is determined by the burst length supported by the memory device in some cases. For example, for DDR3 accesses, the minimum `MSIZE` value is limited by the DDR3 burst length (16 bytes). Any `MSIZE` value below this length leads to a significant throughput loss. For details, refer to the L3/External Memory Throughput section.

Memory to Memory DMA (MDMA)

The ADSP-2156x processors support multiple MDMA streams (MDMA0/1/2/3) to transfer data from one memory to another (L1/L2/L3/memory-mapped peripherals such as SPI FLASH). Different MDMA streams can transfer the data at different bandwidths, as they run at different clock speeds and support different data bus widths. Table 1 shows the various MDMA streams and the corresponding maximum theoretical bandwidth supported by the ADSP-2156x processors. Table 2 shows the various memory slaves and the corresponding maximum theoretical bandwidth supported by the ADSP-2156x processors.

MDMA Stream No.	MDMA Type	Maximum CCLK/SYSCLK/SCLKx Speed (MHz)	MDMA Source Channel	MDMA Destination Channel	Clock Domain	Bus Width (bits)	Maximum Bandwidth (MB/s)
0	Enhanced Bandwidth or Medium Speed MDMA (MSMDMA)	1000/500/125	8	9	SYSCLK	32	2000
1	Enhanced Bandwidth or Medium Speed MDMA (MSMDMA)		18	19		32	2000
2	Enhanced Bandwidth or Medium Speed MDMA (MSMDMA)		39	40		32	2000
3	Maximum Bandwidth or High Speed MDMA (HSMDMA)		43	44		64	4000

Table 1: MDMA Streams and Maximum Theoretical Bandwidth

Memory Type (L1/L2/L3)	Maximum CCLK/SYSCLK/SCLKx/DCLK Frequency (MHz)	Clock Domain	Bus Width (Bits)	Data Rate/Clock Rate	Maximum Theoretical Bandwidth (MB/s)
L1	1000500/125/667	CCLK	32	1	4000
L2		SYSCLK	64	1	4000
L3		DCLK	16	2	2668

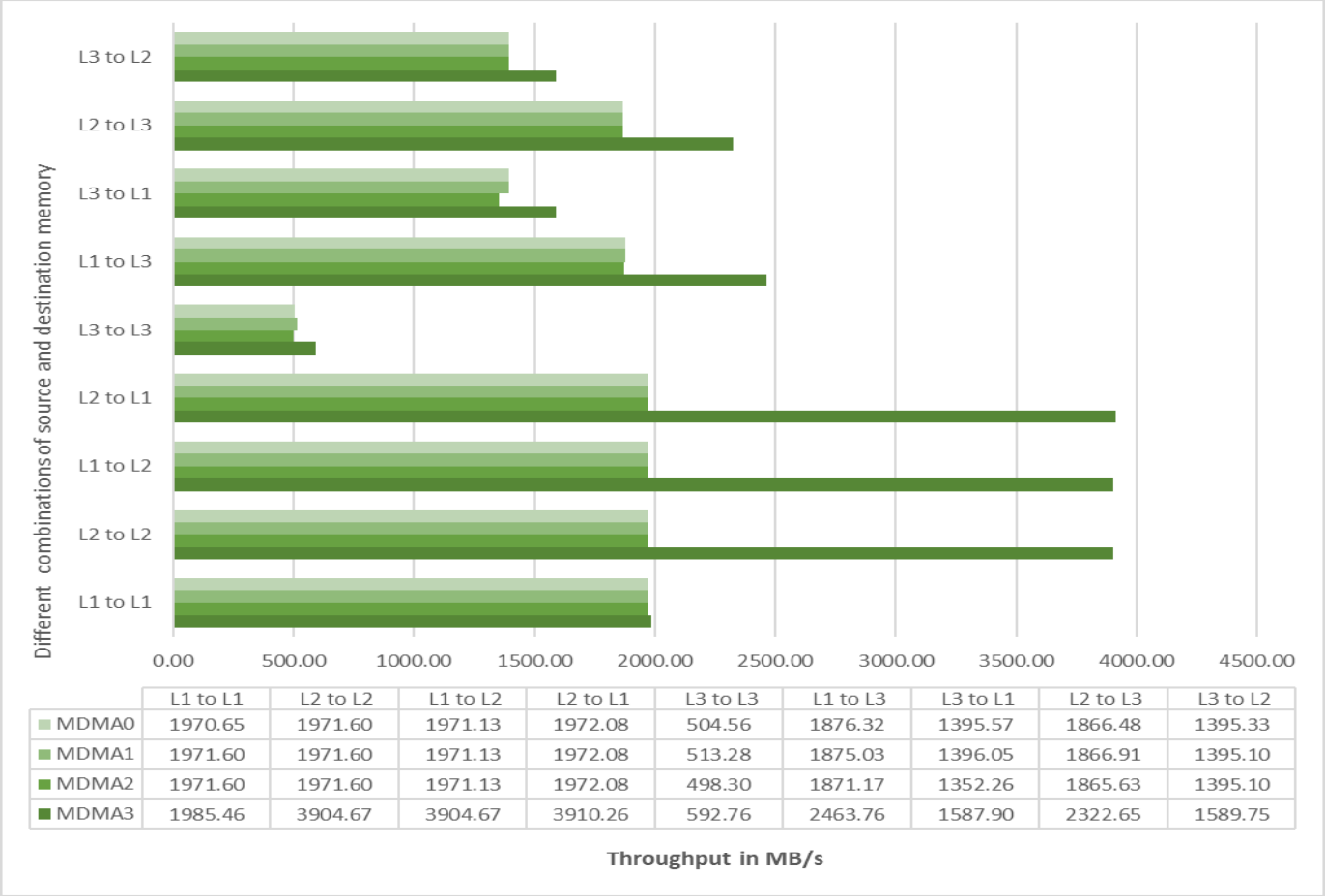
Table 2: Memory Slaves and Maximum Theoretical Bandwidth

The actual (measured) MDMA throughput is always less than or equal to the minimum of the maximum theoretical throughput supported by one of the three: MDMA, source memory, or destination memory. For example, the measured throughput of MDMA0 between L1 and L2 will be less than or equal to 2000 MB/s, which is limited by the maximum bandwidth of MDMA0. Similarly, the measured throughput of MDMA3 between L1 and L3 will be less than or equal to 2688 MB/s, which is limited by the maximum bandwidth of L3.

Figure 3 shows the actual throughput measured on the bench for various MDMA streams with different combinations of source and destination memories. The measurements were taken with:

- `MSIZE` = 32 bytes
- DMA count = 16384 bytes at CCLK = 1 GHz
- SYSCLK = 500 MHz
- DCLK = 667 MHz

The code `MDMA_Throughput` supplied with the application note[5] can be used to measure MDMA throughput.



	L1 to L1	L2 to L2	L1 to L2	L2 to L1	L3 to L3	L1 to L3	L3 to L1	L2 to L3	L3 to L2
MDMA0	1970.65	1971.60	1971.13	1972.08	504.56	1876.32	1395.57	1866.48	1395.33
MDMA1	1971.60	1971.60	1971.13	1972.08	513.28	1875.03	1396.05	1866.91	1395.10
MDMA2	1971.60	1971.60	1971.13	1972.08	498.30	1871.17	1352.26	1865.63	1395.10
MDMA3	1985.46	3904.67	3904.67	3910.26	592.76	2463.76	1587.90	2322.65	1589.75

Figure 3: MDMA Throughput on ADSP-2156x Processor

Optimizing Non-32-Byte-Aligned MDMA Transfers

In many cases, the start address and count of a MDMA transfer may not be aligned to a 32-byte address boundary. In such cases, the `MSIZE` value may need to be configured to less than 32 bytes. This configuration can affect the MDMA performance. One option to get better throughput for such cases is to split the single MDMA transfer into more than one transfer using a descriptor-based DMA. The first and last (if needed) MDMA transfers can use `MSIZE` < 32 bytes for non-32-byte-aligned address and count values. The second transfer can use `MSIZE` = 32 bytes for 32-byte-aligned address and count values.

The MDMA service available with CCES provides an additional API called `adi_mdma_Copy1DAuto`. It is compatible with the standard 1D-transfer API `adi_mdma_Copy1D` that is used for single-shot 1D transfers. As shown in Table 3, the MDMA performance of `adi_mdma_Copy1DAuto` is around 1.4 to 2.6 times better than `adi_mdma_Copy1D` for non-32-byte-aligned start addresses. The example code `MDMA_1DAuto` can be used to measure MDMA performance for both APIs for a given use case.

S. No.	Source Memory Address	Destination Memory Address	DMA Count	MSIZE (Bytes)	Copy1D MDMA Cycles	Copy1DAuto MDMA Cycles	Additional API Overhead	Effective Improvement Factor
1	0x2C0001	0x300000	256	1	1962	1276	152	1.37
2	0x2C0000	0x300001	256	1	1962	1169	148	1.49
3	0x2C0001	0x300000	1024	1	6182	2811	138	2.10
4	0x2C0000	0x300001	1024	1	6182	2704	144	2.17
5	0x2C0001	0x300000	4096	1	23082	8956	138	2.54
6	0x2C0000	0x300001	4096	1	23082	8849	144	2.57

Table 3: adi_mdma_Copy1D vs. adi_mdma_Copy1DAuto Performance

Bandwidth Limiting and Monitoring

MDMAs 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 SYSCLK cycles between two DMA requests. This configuration can be used to ensure 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 determined by the `DMA_BWLCNT` register and the `MSIZE` value and is calculated as follows:

*Bandwidth = min (SYSCLK frequency in MHz*DMA bus width in bytes, SYSCLK frequency in MHz*MSIZE in bytes / DMA_BWLCNT)*

The API `adi_mdma_BWLimit` can be used to program the `DMA_BWLCNT` register for a given target bandwidth and `MSIZE` value. The example code `MDMA_BWLimit` shows how to use this API. Figure 4 shows a sample result of this code with the target and measured bandwidth for different MDMA use cases.

```
Testing combination 1
MDMA Stream no = 0
MSIZE value = 32
Source channel = 1
Target BW = 400.000000 MB/s

Measured BW = 389.798250 MB/s

Test combination 1 passed


Testing combination 2
MDMA Stream no = 1
MSIZE value = 32
Source channel = 1
Target BW = 300.000000 MB/s

Measured BW = 296.039325 MB/s

Test combination 2 passed


Testing combination 3
MDMA Stream no = 2
MSIZE value = 32
Source channel = 1
Target BW = 700.000000 MB/s

Measured BW = 694.237300 MB/s

Test combination 3 passed
```

Figure 4: MDMA Bandwidth Limit Results

Furthermore, the bandwidth monitor feature can be used to check if such channels are starving for resources. The `DMA_BMCNT` register can be programmed to the number of SYSCLK 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 the `DMA_BWMCNT` register into the `DMA_BWMCNT_CUR` register. The DMA decrements `DMA_BWMCNT_CUR` every SYSCLK that a work unit is active. If the `DMA_BWMCNT_CUR` value reaches `0x00000000` before the work unit finishes, the `DMA_STAT.IRQERR` bit is set, and the `DMA_STAT.ERRC` bit is set to `0x6`. The `DMA_BWMCNT_CUR` value 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.

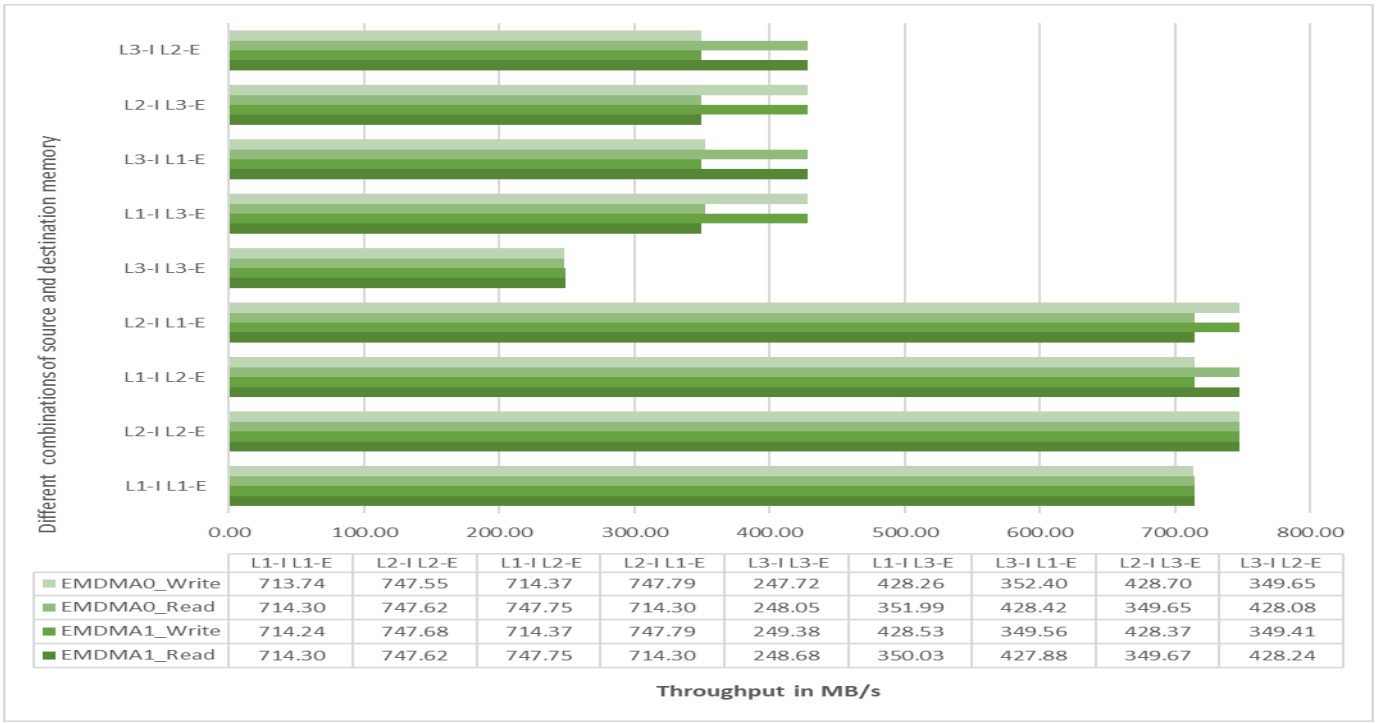
The API `adi_mdma_BWMonitor` can be used to program the `DMA_BWMCNT` register for a given target bandwidth and `MSIZE` value. The example code `MDMA_BWMonitor` shows how to use this API. Figure 5 shows a sample result of this code with a target bandwidth and bandwidth monitor expiration message for a given MDMA use case. The API `adi_mdma_BWMeasure` uses the `DMA_BMCNT` and `DMA_BWMCNT_CUR` registers to measure the MDMA bandwidth as shown in the example code `MDMA_BWLimit`.

```
Testing combination 1
MDMA Stream no = 0
MSIZE value = 32
Source channel = 1
Target BW = 400.000000 MB/s
BW Monitor Expired!! Test combination 1 passed

Testing combination 2
MDMA Stream no = 1
MSIZE value = 32
Source channel = 1
Target BW = 300.000000 MB/s
BW Monitor Expired!! Test combination 2 passed

Testing combination 3
MDMA Stream no = 2
MSIZE value = 32
Source channel = 1
Target BW = 700.000000 MB/s
BW Monitor Expired!! Test combination 3 passed
```

Figure 5: MDMA Bandwidth Monitor Results

Extended Memory DMA (EMDMA)

The ADSP-2156x processors also support Extended Memory DMA (EMDMA). The EMDMA engine is mainly used to transfer the data from one memory type to another in a non-sequential manner (such as circular, delay line, and scatter/gather). For details regarding EMDMA, refer to the *ADSP-2156x SHARC+ Processor Hardware Reference* [2]. The EMDMA on the ADSP-2156x processors has been enhanced to run at the SYSCLK speed instead of the SCLK speed. This enhancement results in improved EMDMA throughput. Figure 6 shows throughput measured on the bench for EMDMA0/EMDMA1 streams with a different combination of source and destination memories for sequential transfers of 4096 32-bit words at CCLK = 1 GHz, SYSCLK = 500 MHz, and DCLK = 667 MHz.



	L1-I L1-E	L2-I L2-E	L1-I L2-E	L2-I L1-E	L3-I L3-E	L1-I L3-E	L3-I L1-E	L2-I L3-E	L3-I L2-E
EMDMA0_Write	713.74	747.55	714.37	747.79	247.72	428.26	352.40	428.70	349.65
EMDMA0_Read	714.30	747.62	747.75	714.30	248.05	351.99	428.42	349.65	428.08
EMDMA1_Write	714.24	747.68	714.37	747.79	249.38	428.53	349.56	428.37	349.41
EMDMA1_Read	714.30	747.62	747.75	714.30	248.68	350.03	427.88	349.67	428.24

Figure 6: Measured EMDMA Throughput on ADSP-2156x Processors

Optimizing Non-Sequential EMDMA Transfers with MDMA

In some cases, the non-sequential transfer modes supported by EMDMA can be replaced by descriptor-based MDMA for better performance.

The example code `MDMA_Circular_Buffer` illustrates how a MDMA descriptor-based mode can be used to emulate a circular buffer memory-to-memory DMA transfer mode. The example code compares the core cycles measured (see Table 4) to write and read 4096 32-bit words to and from the DDR3 memory in circular buffer mode. The example uses a starting address offset of 1024 words for the following cases:

- EMDMA
- MDMA with `MSIZE` = 4 bytes (for 4-byte-aligned address and count)
- MDMA with `MSIZE` = 32 bytes (for 32-byte-aligned address and count)

Write/Read	Core Cycles		
	EMDMA	MDMA3 MSIZE = 4 bytes	MDMA3 MSIZE = 32 bytes
Write	38638	30327	7681
Read	49580	36281	10908

Table 4: MDMA Emulated Circular Buffer vs. EMDMA

As shown in Table 4, the MDMA emulated circular buffer (`MSIZE` = 4 bytes) is faster than EMDMA. The performance is further improved with `MSIZE` = 32 bytes when the addresses and counts are 32-byte-aligned.

Understanding the System Crossbars

As shown in Figure 2, the SCB interconnect consists of a hierarchical model connecting multiple SCB units. Figure 7 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. Similarly, each M is connected to a fixed SI.

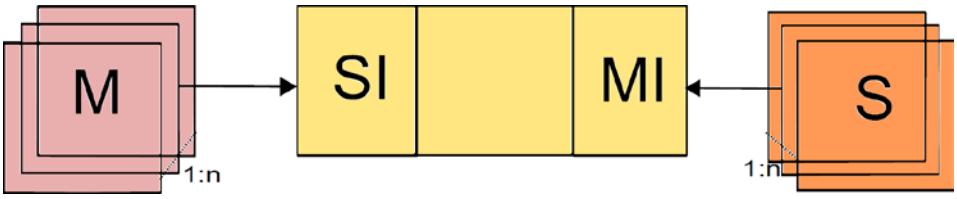


Figure 7: Single SCB Block Diagram

The slave interface of the crossbar (where masters such as DDE are connected) perform two functions, arbitration and clock domain conversion.

Arbitration

The programmable Quality of Service (QoS) registers can be viewed as being associated with SCBx. For example, the programmable QoS registers for SPORT0-3 and MDMA0 can be viewed as residing in

SCB1. Whenever a transaction is received at SPORT0 half A, the programmed QoS value is associated with that transaction and is arbitrated with the rest of the masters at SCB1.

Programming the SCB QOS Registers

Consider a scenario where:

- At SCB1, masters 1, 2, and 3 have RQOS values of 6, 4, and 2, respectively.
- At SCB2, masters 4, 5, and 6 have RQOS values of 12, 13, and 1, respectively.

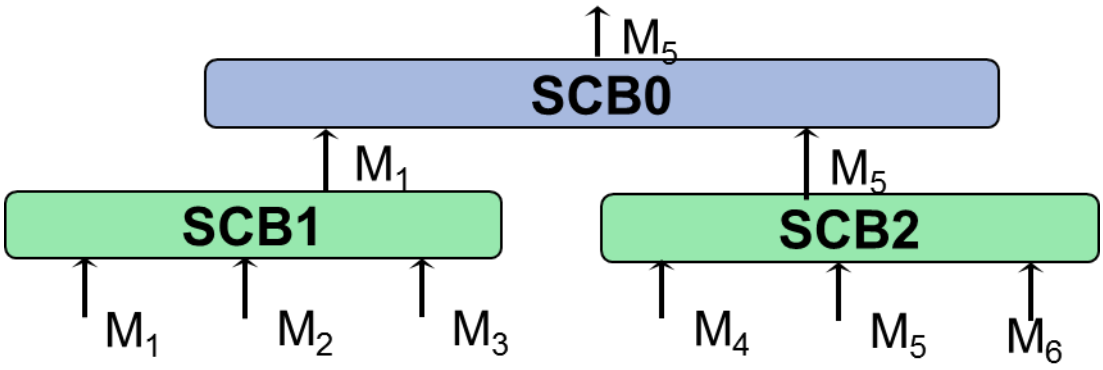


Figure 8: Arbitration Among Various Masters

As shown in Figure 8, in this case:

- Master 1 wins the arbitration at SCB1, and master 5 wins the arbitration at SCB2.
- In a perfect competition at SCB0, however, masters 4 and 5 had the highest overall RQOS values. So, the masters would have fought for arbitration directly at SCB0. Because of the mini-SCBs, however, master 1, at a much lower RQOS value, wins against master 4 and makes it all the way to SCB0.

Clock Domain Conversion

There are multiple Clock Domain Crossings (CDC) in the ADSP-2156x processor fabric:

- CCLK: SYSCLK is fixed to SYNC n:1
- SCLK0: SYSCLK is fixed to 1:n
- LPCLK: SYSCLK is fixed to m:n
- SPI CLK: SYSCLK is fixed to m:n

The SYSCLK to DCLK clock domain crossing is programmable, and it should be programmed depending upon the clock ratios of the two clock domains for optimum performance.

Programming Sync Mode in the IBx Registers

To illustrate the effect of the sync mode programming in the `IBx` registers, DDR3 read throughput was measured with an MDMA3 stream with MSIZE = 32 bytes at CCLK = 1 GHz, SYSCLK = 500 MHz, DCLK = 500 MHz (both SYNC 1:1 and ASYNC modes). Figure 9 shows throughput measured for the

two for different work unit size values. As can be seen, the throughput is better (for most of the work unit size values) when the CDC is programmed in SYNC mode with DCLK = 500 MHz as compared to the ASYNC mode.

	32	64	128	256	512	1024	2048	4096	8192	16384
ASYNC_DCLK500	161.62	344.09	598.13	820.51	1011.86	1147.98	1211.83	1137.15	1221.96	1233.73
SYNC_DCLK500	190.48	355.56	603.77	853.33	1075.63	1236.72	1036.44	1368.98	1309.88	1371.05

Figure 9: DMC MDMA Throughput with and Without IBx Sync Mode Programming

Understanding the System Slaves

Memory Hierarchy

As shown in Table 2, ADSP-SC5xx processors have a hierarchical memory model (L1/L2/L3). The following sections discuss the access latencies and achievable throughput associated with the different memory levels.

L1 Memory Throughput

L1 memory runs at CCLK and is the fastest accessible memory in the hierarchy. SHARC+ L1 memory is accessible by both the core and DMA (system). For system (DMA) accesses, L1 memory supports two ports: the S1 port and the S2 port. Two different banks of L1 memory can be accessed in parallel with these ports. Like the ADSP-SC57x processors, the system (DMA) accesses are hardwired to the L1 memory of the SHARC+ processer using the S1 port. One exception is the High-Speed MDMA

(HSMDMA), which is hardwired using the S2 port. From a programming perspective, when accessing the L1 memory of the SHARC+ processer, use a multiprocessor memory offset of 0x28000000 for all DMA accesses (including HSMDMA).

The maximum theoretical throughput of L1 memory (for system/DMA accesses) is 1000 * 4 = 4000 MB/s for 1 GHz CCLK operation. As shown in Figure 3, the maximum measured L1 throughput using MDMA3 is ~3910 MB/s.

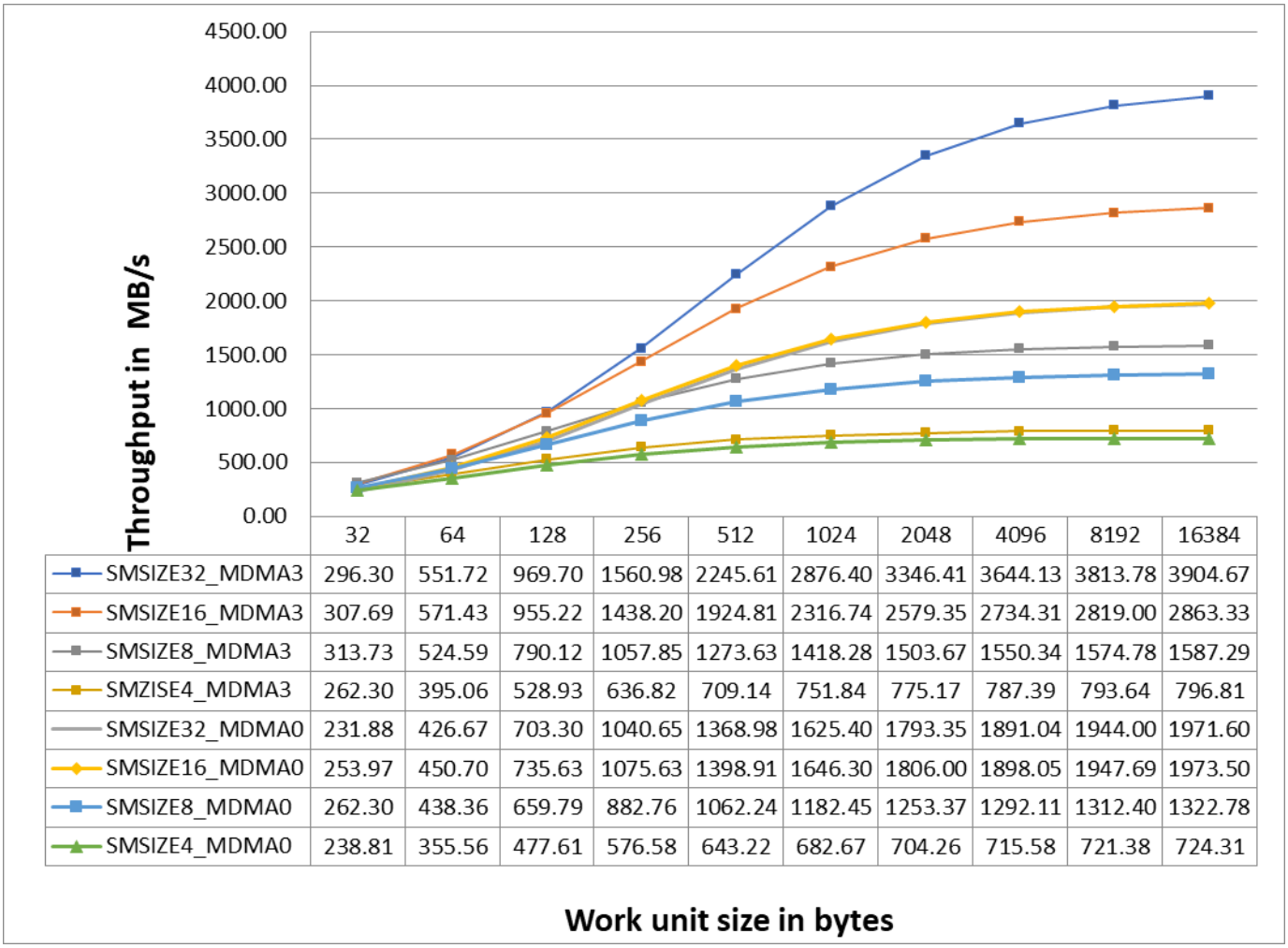
L2 Memory Throughput

L2 memory access times are longer than L1 because the maximum L2 clock frequency (SYSCLK) is half the CCLK. 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 (64-bit DMA bus is supported for HSMDMA accesses). Each port has a read and a write channel. For details, refer to the *ADSP-2156x SHARC+ Processor Hardware Reference*[2].

Consider the following important points regarding L2 memory throughput:

- Since L2 memory runs at the SYSCLK speed, it can provide a maximum theoretical throughput of 500 MHz * 4 = 2000 MB/s in one direction (for HSMDMA accesses, it can go up to 4000 MB/s). Since there are separate read and write channels, the total throughput in both directions equals 8000 MB/s. To operate L2 SRAM memory at its optimum throughput, use both the core and DMA ports and separate read and write channels in parallel. All of them should access different banks of L2.
- All accesses to L2 memory are converted to 64-bit accesses (8-byte) by the L2 memory controller. To achieve optimum throughput for DMA access to L2 memory, configure the DMA channel `MSIZE` to 8 bytes or higher.
- Unlike L3 (DMC) memory accesses, L2 memory throughput for sequential and non-sequential accesses is the same.
- L2 SRAM 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 latency is due to the ECC implementation. The implementation is in terms of 32-bit accesses. Any write that is less than 32-bit to an ECC-enabled SRAM bank is implemented as a read-followed-by-write and requires three cycles to complete (two cycles for the read, 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 the DMA engine 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 throughput 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 10 shows the measured MDMA throughput at CCLK = 1 GHz and SYSCLK = 500 MHz for a case where both source and destination buffers are in different L2 memory banks. As an example, for MDMA3, the maximum throughput is very close to 3905 MB/sec in one direction (7810 MB/s in both directions) for `MSIZE` = 32 bytes and drops significantly for smaller `MSIZE` values.



	32	64	128	256	512	1024	2048	4096	8192	16384
SMSIZE32_MDMA3	296.30	551.72	969.70	1560.98	2245.61	2876.40	3346.41	3644.13	3813.78	3904.67
SMSIZE16_MDMA3	307.69	571.43	955.22	1438.20	1924.81	2316.74	2579.35	2734.31	2819.00	2863.33
SMSIZE8_MDMA3	313.73	524.59	790.12	1057.85	1273.63	1418.28	1503.67	1550.34	1574.78	1587.29
SMZISE4_MDMA3	262.30	395.06	528.93	636.82	709.14	751.84	775.17	787.39	793.64	796.81
SMSIZE32_MDMA0	231.88	426.67	703.30	1040.65	1368.98	1625.40	1793.35	1891.04	1944.00	1971.60
SMSIZE16_MDMA0	253.97	450.70	735.63	1075.63	1398.91	1646.30	1806.00	1898.05	1947.69	1973.50
SMSIZE8_MDMA0	262.30	438.36	659.79	882.76	1062.24	1182.45	1253.37	1292.11	1312.40	1322.78
SMSIZE4_MDMA0	238.81	355.56	477.61	576.58	643.22	682.67	704.26	715.58	721.38	724.31

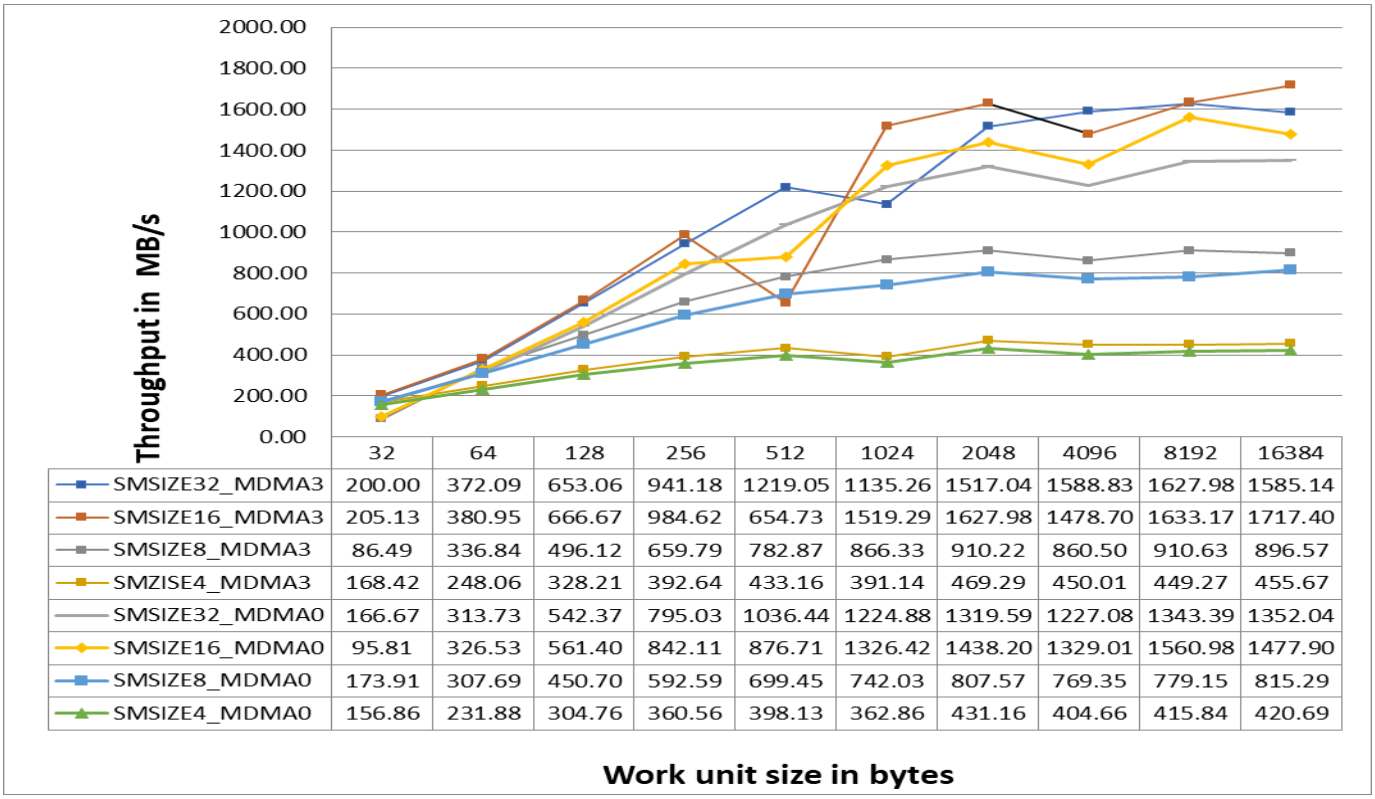
Figure 10: L2 MDMA Throughput for Different MSIZE and Work Unit Sizes

L3/External Memory Throughput

The ADSP-2156x processors provide interfaces for connecting to DDR3/DDR3L memory devices. The DMC interface operates at speeds of up to 667 MHz. For the 16-bit DDR3 interface, the maximum theoretical throughput that the DMC can deliver equals 2668 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 derived from the DRAM technology itself (access patterns, page hit to page miss ratio, and so forth).

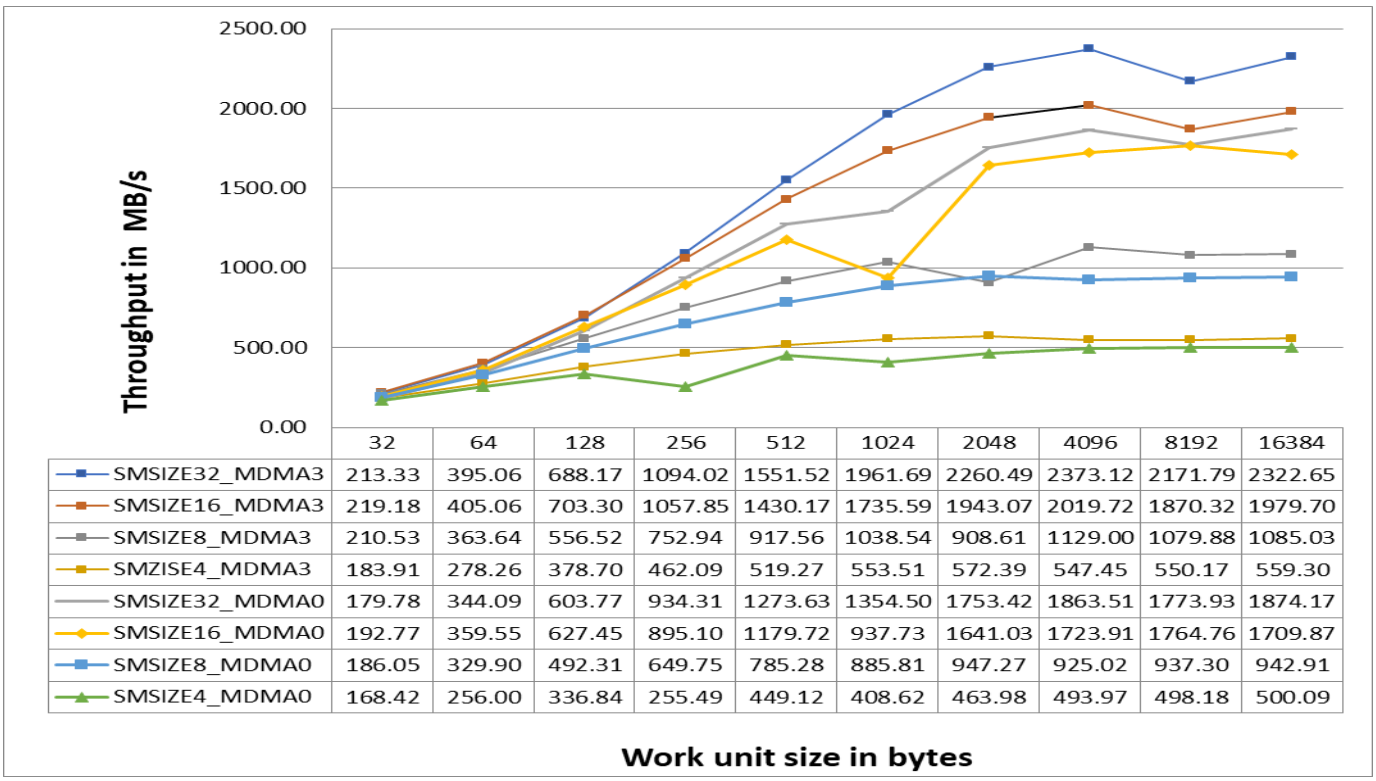
Although most of the throughput optimization concepts are illustrated using MDMA as an example, the same can be applied to other system masters as well.

The MDMA3 stream (HSMDMA) can request DMC accesses faster than any other masters (for example, 4000 MB/s). The practical DMC throughput possible using MDMA depends upon factors such as whether the accesses are sequential or non-sequential, the block size of the transfer, and DMA parameters (for example, `MSIZE`).



	32	64	128	256	512	1024	2048	4096	8192	16384
SMSIZE32_MDMA3	200.00	372.09	653.06	941.18	1219.05	1135.26	1517.04	1588.83	1627.98	1585.14
SMSIZE16_MDMA3	205.13	380.95	666.67	984.62	654.73	1519.29	1627.98	1478.70	1633.17	1717.40
SMSIZE8_MDMA3	86.49	336.84	496.12	659.79	782.87	866.33	910.22	860.50	910.63	896.57
SMZISE4_MDMA3	168.42	248.06	328.21	392.64	433.16	391.14	469.29	450.01	449.27	455.67
SMSIZE32_MDMA0	166.67	313.73	542.37	795.03	1036.44	1224.88	1319.59	1227.08	1343.39	1352.04
SMSIZE16_MDMA0	95.81	326.53	561.40	842.11	876.71	1326.42	1438.20	1329.01	1560.98	1477.90
SMSIZE8_MDMA0	173.91	307.69	450.70	592.59	699.45	742.03	807.57	769.35	779.15	815.29
SMSIZE4_MDMA0	156.86	231.88	304.76	360.56	398.13	362.86	431.16	404.66	415.84	420.69

Figure 11: DMC Measured Throughput for Sequential MDMA Reads



	32	64	128	256	512	1024	2048	4096	8192	16384
SMSIZE32_MDMA3	213.33	395.06	688.17	1094.02	1551.52	1961.69	2260.49	2373.12	2171.79	2322.65
SMSIZE16_MDMA3	219.18	405.06	703.30	1057.85	1430.17	1735.59	1943.07	2019.72	1870.32	1979.70
SMSIZE8_MDMA3	210.53	363.64	556.52	752.94	917.56	1038.54	908.61	1129.00	1079.88	1085.03
SMZISE4_MDMA3	183.91	278.26	378.70	462.09	519.27	553.51	572.39	547.45	550.17	559.30
SMSIZE32_MDMA0	179.78	344.09	603.77	934.31	1273.63	1354.50	1753.42	1863.51	1773.93	1874.17
SMSIZE16_MDMA0	192.77	359.55	627.45	895.10	1179.72	937.73	1641.03	1723.91	1764.76	1709.87
SMSIZE8_MDMA0	186.05	329.90	492.31	649.75	785.28	885.81	947.27	925.02	937.30	942.91
SMSIZE4_MDMA0	168.42	256.00	336.84	255.49	449.12	408.62	463.98	493.97	498.18	500.09

Figure 12: DMC Measured Throughput for Sequential MDMA Writes

Figure 11 and Figure 12 provide the DMC measured throughput at CCLK = 1 GHz and DCLK = 533 MHz using MDMA0 (channels 8/9) and MDMA3 (channels 43/44) streams for various MSIZE values and buffer sizes.

The following important observations can be made:

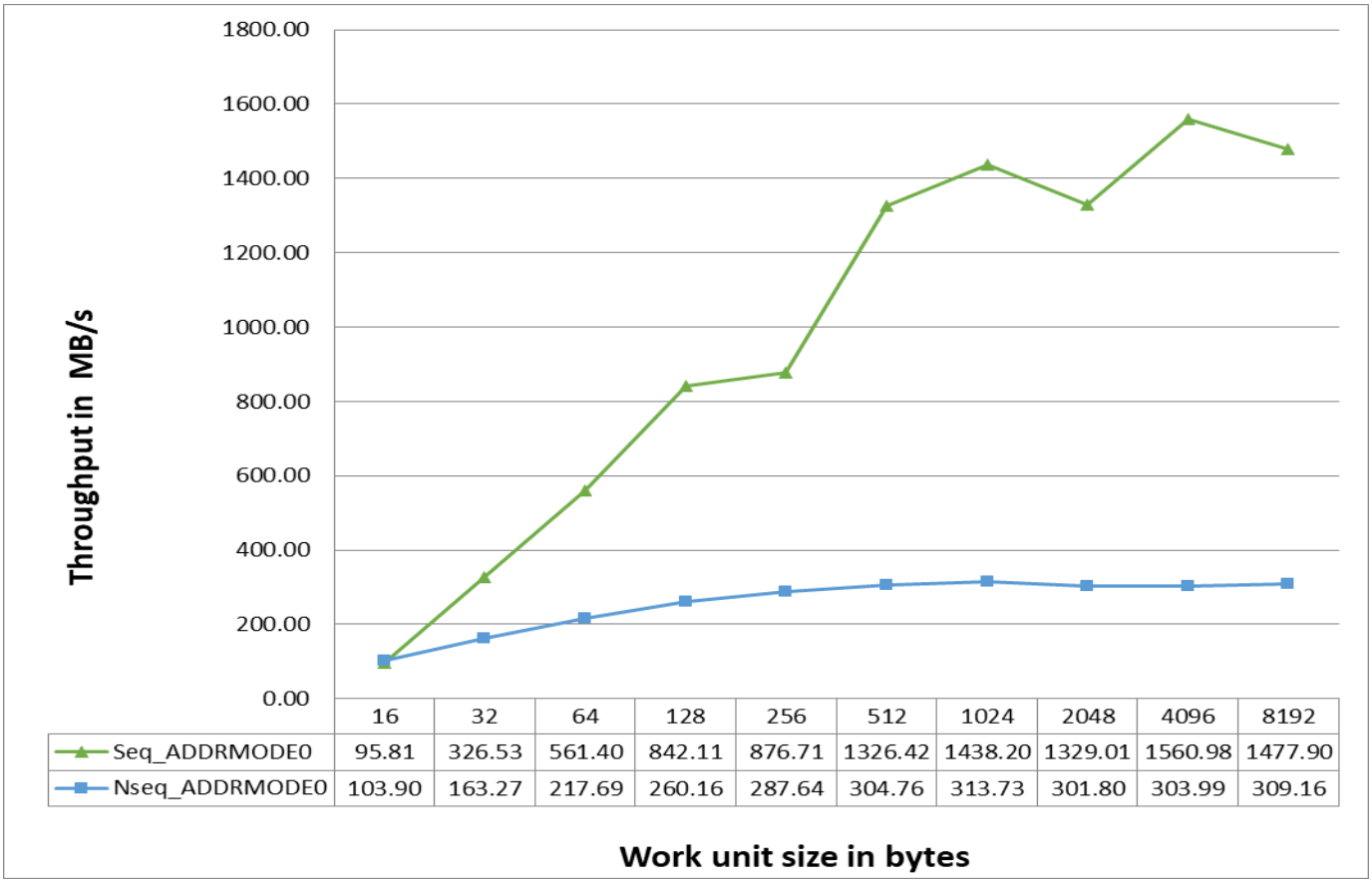
- The throughput trends are similar for reads and writes with regards to MSIZE and buffer size values.
- The peak measured read throughput is 1633 MB/s for MDMA3 with MSIZE = 16 bytes. The peak measured write throughput is 2373 MB/s for MDMA3 with MSIZE = 32 bytes.
- The throughput depends largely upon the DMA buffer size. For smaller buffer sizes, the throughput is significantly lower. The throughput increases significantly with a larger buffer size. For example, for MDMA3 with MSIZE = 32 bytes and a buffer size of 32 bytes, the read throughput is 200 MB/s, whereas it reaches 1585 MB/s for a 16 KB buffer size. This difference is 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. Better sustained throughput is obtained for continuous transfers over time.

- 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 11 and Figure 12, in most cases, greater MSIZE values provide better results. Ideally, the MSIZE value should be at least equal to the DDR memory burst length. For MDMA3 with a buffer size of 16384 bytes, the read throughput is 1585 MB/s for MSIZE = 32 bytes, while it reduces significantly to 559 MB/s for MSIZE = 4 bytes. For MSIZE = 4 bytes, although all accesses are still sequential, the full DDR3 memory burst length of 16 bytes (eight 16-bit words) is not used.

For sequential reads, it is easy to achieve optimum throughput, particularly for larger buffer sizes. The DRAM memory page hit ratio is high, and the DMC controller does not need to close and open DDR device rows frequently. However, in case of non-sequential accesses, throughput can drop slightly or significantly depending upon the page hit-to-miss ratio.

Figure 13 provides a comparison of the DMC throughput numbers measured for sequential MDMA read accesses for MSIZE = 16 bytes (equals DDR3 burst length) and ADDRMODE set to 0 (bank interleaving) versus non-sequential accesses with a modifier of 2048 bytes (equals DDR3 page size, thus leading to a worst-case scenario with maximum possible page misses). As shown, for a buffer size of 8192 bytes, throughput drops significantly from 1479 to 309 MB/s.



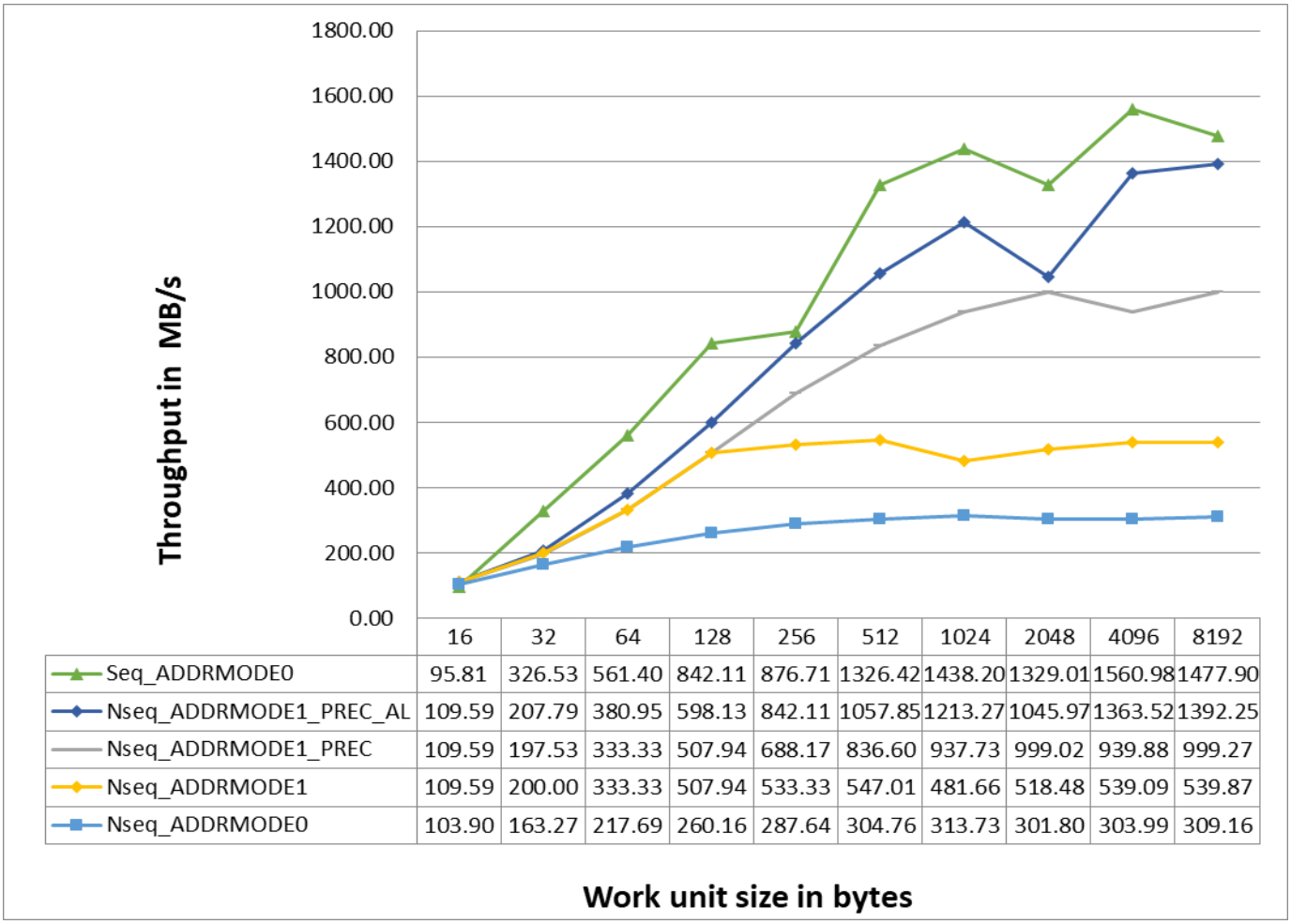
	16	32	64	128	256	512	1024	2048	4096	8192
Seq_ADDRMODE0	95.81	326.53	561.40	842.11	876.71	1326.42	1438.20	1329.01	1560.98	1477.90
Nseq_ADDRMODE0	103.90	163.27	217.69	260.16	287.64	304.76	313.73	301.80	303.99	309.16

Figure 13: DMC Throughput for Sequential vs. Non-Sequential Read Accesses

DDR memory devices support concurrent bank operations that allow the DMC controller to activate a row in another bank without pre-charging the row of a bank. This feature is extremely helpful in cases where DDR access patterns incur page misses. By setting the `DMC_CTL.ADDRMODE` bit, throughput can be improved by ensuring that such accesses fall into different banks. For instance, Figure 14 shows in orange how the DMC throughput increases from 309.16 MB/s to 539.87 MB/s by setting this bit for the non-sequential access pattern shown in Figure 13.

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 configuration allows the row of a bank to proactively precharge after it has been accessed. It improves the throughput by saving the latency involved in precharging the row at the time when the next row of the same bank must be activated.

The gray line in Figure 14 show the increase in throughput. Setting the `DMC_CTL.PREC` bit results in an increase from 540 MB/s to 999 MB/s. The same result can be achieved by setting the `DMC_EFFCTL.PRECBANK[7-0]` bits. This feature can be used on a per bank basis. However, note 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 configuration can provide an added throughput advantage for cases where `MSIZE` (for example, 32 bytes) is greater than the DDR3 burst length (16 bytes).



	16	32	64	128	256	512	1024	2048	4096	8192
Seq_ADDRMODE0	95.81	326.53	561.40	842.11	876.71	1326.42	1438.20	1329.01	1560.98	1477.90
Nseq_ADDRMODE1_PREC_AL	109.59	207.79	380.95	598.13	842.11	1057.85	1213.27	1045.97	1363.52	1392.25
Nseq_ADDRMODE1_PREC	109.59	197.53	333.33	507.94	688.17	836.60	937.73	999.02	939.88	999.27
Nseq_ADDRMODE1	109.59	200.00	333.33	507.94	533.33	547.01	481.66	518.48	539.09	539.87
Nseq_ADDRMODE0	103.90	163.27	217.69	260.16	287.64	304.76	313.73	301.80	303.99	309.16

Figure 14: Optimizing Throughput for Non-Sequential Accesses

For reads, the throughput can be further improved by using the additive latency feature. Figure 15 and Figure 16 illustrates how this feature helps to avoid gaps in a burst of data when accessing different banks for CAS Latency = 4 and Additive Latency = 3. Note that these figures were used to explain the concept from the reference TN4702[7] which refers to DDR2. The ADSP-2156x processors support DDR3/DDR3L devices only, but the same concept applies here as well.

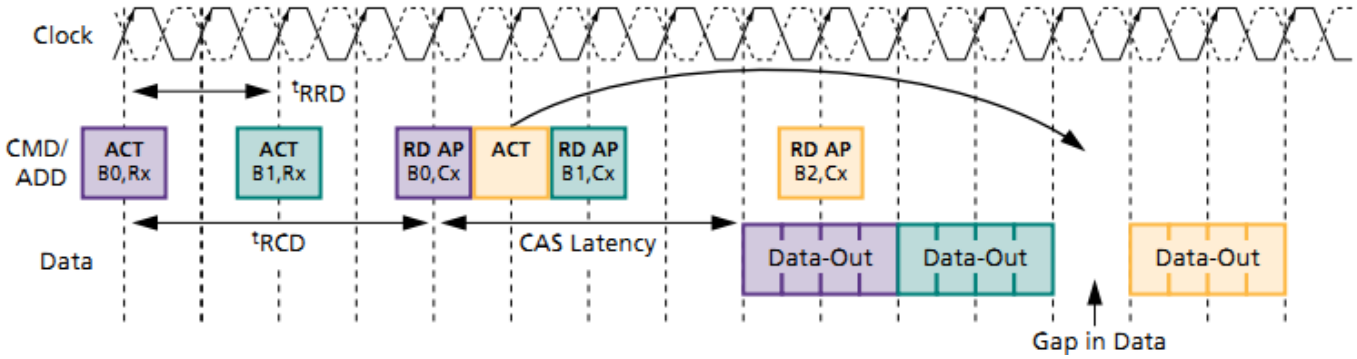


Figure 15: Reads Without Additive Latency

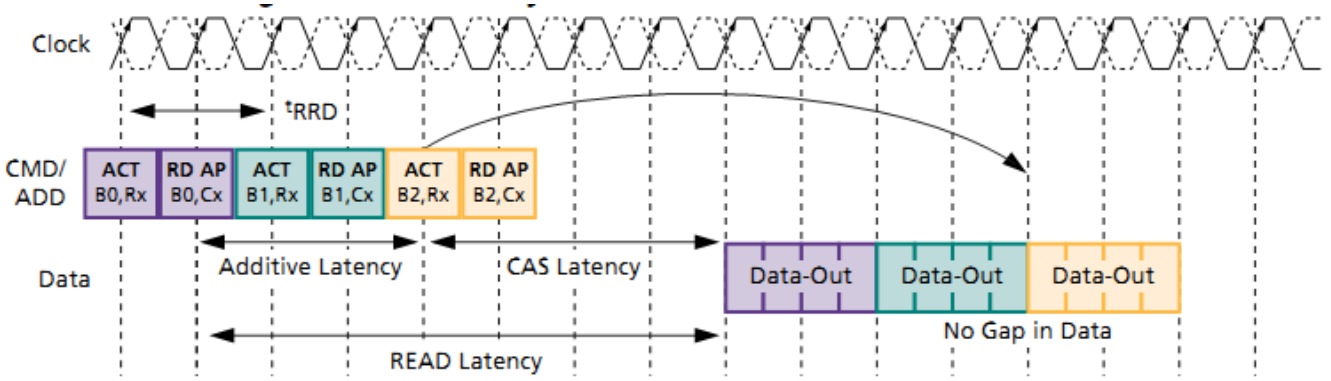


Figure 16: Reads with Additive Latency

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 sequence enables the controller to schedule the `Activate` and `Read` commands for other banks, eliminating gaps in the data stream. Figure 14 shows how the throughput improves from 999 MB/s to 1392 MB/s by programming the additive latency (`AL`) in the `DMC_EMR1` register to `tRCD-1` (equals 8 or CL-1 in this case).

The DMC also allows elevating the priority of the accesses requested by a SCB master using the `DMC_PRIO` and `DMC_PRIOMSK` registers. The associated `.ZIP` file provides example code `DMC_SCB_PRIO` in which two MDMA DMC read channels (8 and 18) run in parallel. Table 6 summarizes the measured throughout. As shown, programming the DMC SCB priority for a MDMA channel results in an increased throughput of the higher priority MDMA when compared with the other MDMA running in parallel.

Test Case Number	Priority Channel (SCB ID)	MDMA0 (Ch. 8) Throughput (MB/s)	MDMA1 (Ch. 18) Throughput (MB/s)
1	None	737	749
2	MDMA0 (0x008)	843	704
3	MDMA1 (0x208)	600	729

Table 5. DMC Measured Throughput for Different DMC_PRIO Settings

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 sequentially to the DRAM memory.

After every refresh, the SCB command buffers are checked to ensure that they stay empty. However, if the SCB command buffers are always full, once the programmed number of refresh commands accumulates 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 the DMC enters self-refresh mode.

System MMR Latencies

Table 6 shows the measured MMR latency in core cycles for different peripherals on the ADSP-2156x processor. The measurement was taken with CCLK = 1 GHz, SYSCLK = 500 MHz, and SCLK = 250 MHz. These numbers can be used to approximate the MMR access latency of the SHARC+ core for different peripherals.

S. No.	Register	Peripheral	Write Latency (Core Cycles)	Read Latency (Core Cycles)
1	FIR0_INIDX	FIR	28	27
2	IIR0_INIDX	IIR	28	27
3	MLB0_MDAT0	MLB	44	41
4	MEC0_PERR_IMASK0	MEC	44	41
5	CRC0_DCNT	CRC	44	43
6	CRC1_DCNT		44	43
7	EMDMA0_INDX1	EMDMA	44	43
8	EMDMA1_INDX1		44	43
9	TAPC_SDBGKEY0	TAPC	44	41
10	SMPU2_RADDR0	SMPU	46	43
11	SWU1_LA0	SWU	46	43
12	SWU2_LA0		46	43
13	L2CTL0_RPCR	L2CTL	46	43
14	SEC0_RAISE	SEC	46	45
15	TRU0_SSR0	TRU	46	45
16	SPU0_SECUREP10	SPU	46	45
17	RCU0_MSG	RCU	46	43
18	CGU0_OSCWDCTL	CGU	46	43
19	CDU0_CLKINSEL	CDU	46	43
20	DPM0_PER_DIS0	DPM	46	43
21	PKTE0_SA_ADDR	PKTE	48	47
22	TRNG0_OUTPUT0	TRNG0	52	51
23	PKA0_APTR	PKA	52	51
24	PKIC0_ACK	PKIC	52	51
25	DMC0_PRIO	DMC	60	57
26	UART0_CLK	UART	74	73

27	PORTA_DATA_SET	PORTA	74	73
28	WDOG1_WIN	WDOG	74	73
29	DMA1_XCNT	DMA	76	73
30	SPORT0_DIV_A	SPORT	76	81
31	PORTB_DATA_SET	PORTB	76	73
32	PINT0_ASSIGN	PINT	76	73
33	PINT1_ASSIGN		76	73
34	TIMER0_TMR0_WID	TIMER	76	81
35	DMA0_XCNT	DMA	78	73
36	PADS0_PORTA_PDE	PADS	78	73
37	WDOG0_WIN	WDOG	78	73
38	OSPI0_FCA	OPSI	78	73
39	SPI1_CLK	SPI	78	81
40	SPORT1_DIV_A	SPORT	80	81
41	UART1_CLK	UART	80	73
42	CNT0_CNTR	CNT	80	73
43	OTPC0_PMC_MODE0	OTPC	80	73
44	DAI0_IMSK_FE	DAI0	80	81
45	SPI0_CLK	SPI0	82	81
46	ASRC1_MUTE	ASRC	82	81
47	TMU0_FLT_LIM_HI	TMU	84	81
48	PCG0_PW1	PCG	84	81
49	DAI1_IMSK_FE	DAI	84	81
50	SPDIF0_TX_UBUFF_A0	SPDIF	86	81
51	SPDIF1_TX_UBUFF_A0		86	81
52	HADC0_CHAN_MSK	HADC	88	81
53	ASRC0_MUTE	ASRC	90	81
54	TWI1_CLKDIV	TWI	114	121
55	LP0_DIV	LP	117	121
56	TWI0_CLKDIV	TWI	118	121
57	LP1_DIV	LP	120	121
58	SCB0_SP0A_READ_QOS	SCB	780	777

Table 6: MMR Access Latency for ADSP-2156x Processors in CCLK cycles (approximate)

The MMR latency numbers are measured with the “sync” instruction after the write. This ensures that the write has taken affect. The SHARC+ core supports posted writes, which means that the core does not necessarily wait until the actual write is complete. This helps in avoiding unnecessary core stalls.

The MMR access latencies can vary based on the following factors:

- **Clock ratios:** All MMR accesses are through SCB0, which is in the SYSCLK domain, while peripherals are in the SCLK0/1, SYSCLK, and DCLK domains.
- **Number of concurrent MMR access requests in the system:** Although a single write incurs half the system latency when 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 than a latency observed on the core.
- Memory type (L1/L2/L3) from where the code is executed.

System Bandwidth Optimization Procedure

Although the optimization techniques can vary from one application to another, the general procedure involved in the overall system bandwidth optimization remains the same. Figure 17 provides a flow chart of a typical system bandwidth optimization procedure for ADSP-2156x processor-based applications.

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

- Identify the individual and total throughput requirements for all the masters accessing the corresponding SCB slave in the system. Consider the total throughput requirement as *X*.
- Calculate the observed throughput the corresponding SCB slave(s) are able to supply under the specific conditions. 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 an underflow condition. In this case, apply the bandwidth optimization techniques to either:
    - Increase the value of *Y* by applying the slave-specific optimization techniques (for example, using the DMC efficiency controller features).
    - Decrease the value of *X* by:
        - Reanalyzing whether a peripheral really needs to run that fast. If not, then slow down the peripheral to reduce the bandwidth requested by the peripheral.
        - Reanalyzing whether an MDMA can be slowed down. The corresponding `DMAx_BWLCNT` register can be used to limit the bandwidth of that DMA channel.

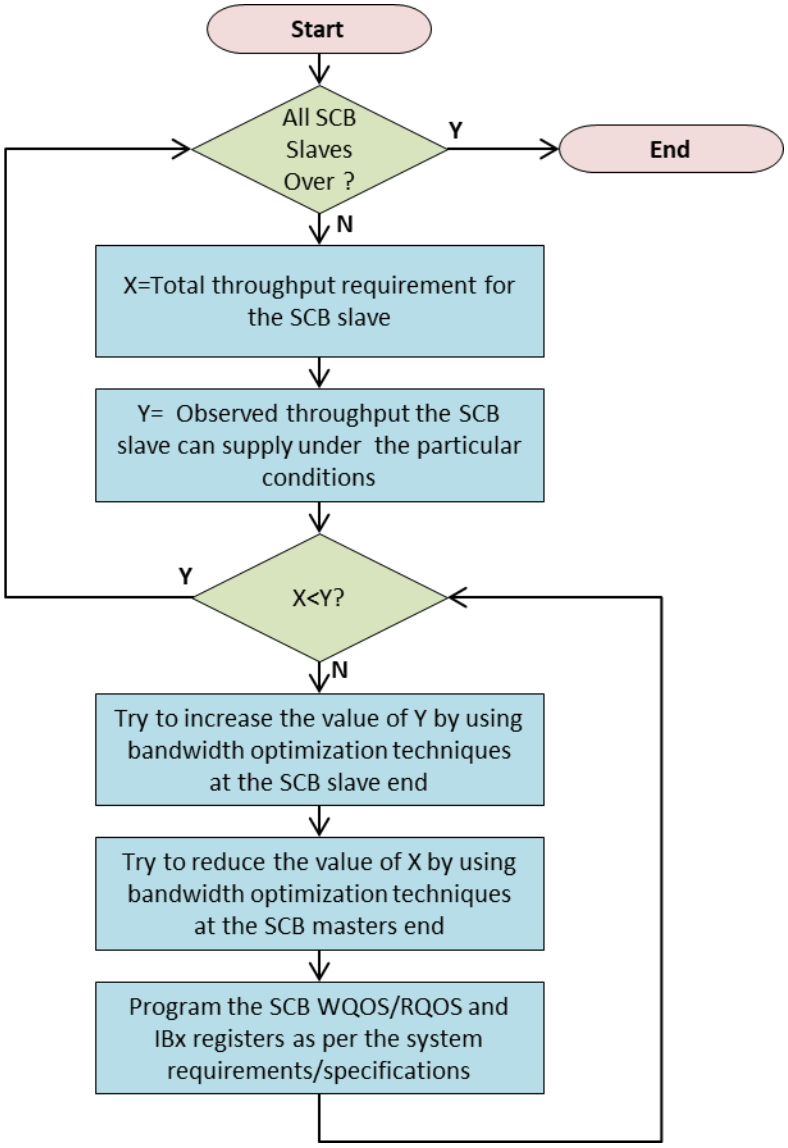


Figure 17: Typical System Bandwidth Optimization Procedure

Application Example

Figure 18 shows a block diagram of the example application code `Multiple_DMAs` supplied with this EE-Note. The example code has the following conditions:

- CCLK = 1 GHz, DCLK = 667 MHz, SYSCLK = 500 MHz, and SCLK0 = 125 MHz.
- MDMA0 channel 8: required throughput = 500 MHz * 4 ≤ 2000 MB/s.
- MDMA1 channel 18: required throughput = 500 MHz * 4 ≤ 2000 MB/s.
- MDMA2 channel 39: required throughput = 500 MHz * 4 ≤ 2000 MB/s.
- MDMA3 channel 43: required throughput = 500 MHz * 8 ≤ 4000 MB/s.

Throughput requirements for MDMA channels depend on the corresponding `DMAx_BWLCNT` register values. If not programmed, the MDMA channels request for the bandwidth with full throttle.

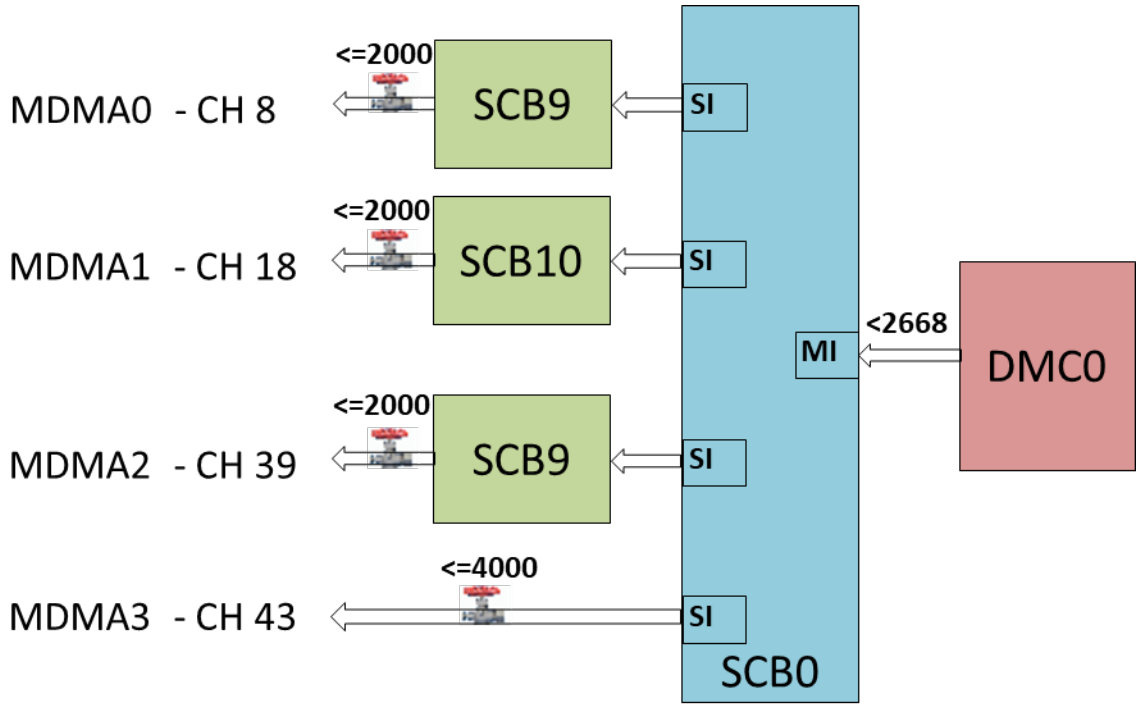


Figure 18: Example Application - DMC Throughput Distribution Across SCBs

As shown in Figure 18, the total required throughput from the DMC controller equals 10000 MB/s. Theoretically, with `DCLK` = 667 MHz and a bus width of 16 bits, the maximum possible throughput is only 2668 MB/s. For this reason, there is a possibility that one or more masters may not get the required bandwidth. For masters such as MDMA, this can result in decreased throughput.

S.No.	Condition	SCB9		SCB10		SCB11		MDMA3		Total throughput Requirement X	Measured throughput Y'
		MDMA0		MDMA1		MDMA2					
		Required	Measured	Required	Measured	Required	Measured	Required	Measured		
1	No optimization	2000	213.8	2000	213.8	2000	213.8	4000	213.7	10000	855.1
2	Optimization at the slave - T1	2000	329.7	2000	329.3	2000	329.2	4000	329.6	10000	1317.8
3	Optimization at the master - T2	200	196.8	200	196.6	300	288.1	400	359.1	1100	1040.6
4	Optimization at the master - T3	100	99.2	100	99.2	400	379.5	500	465.2	1100	1043.1

All units are expressed in MB/s.

Table 7: Example Application - System Bandwidth Optimization Steps

Table 7 shows the expected and measured throughput for all DMA channels and the corresponding SCBs at various steps of bandwidth optimization.

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 DDR3 SDRAM bank. The row corresponding to "No optimization" in Table 7 shows the measured throughput numbers under this condition. As illustrated, the individual measured throughput of almost all channels is significantly less than expected.

- Total expected throughput from the DMC (*X*) = 10000 MB/s
- Effective DMC throughput (*Y*) = 855.1 MB/s

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

Step 2

When there are frequent DDR3 SDRAM page misses within the same bank, throughput significantly drops. Although DMA channel accesses are sequential, multiple channels try to access the DMC concurrently, resulting in page misses. To work around this, move the source buffers for each DMA channel to different DDR3 SDRAM banks. This configuration allows parallel accesses to multiple pages of different banks, helping improve *Y*. *"Optimization at the slave - T1"* in Table 7 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 1317.8 MB/sec. However, since *X* is still greater than *Y*, the measured throughput is still lower than expected.

Steps 3 and 4

There is not much more room to significantly increase the value of *Y*. Alternatively, optimization techniques can be employed at the master end. Depending on the application requirements, the bandwidth limit feature of the MDMAs can be used to reduce the overall bandwidth requirement and get a predictable bandwidth at various MDMA channels. *"Optimization at the master - T2"* and *"Optimization at the master – T3"* in Table 7 show the measured throughput under two such conditions with different bandwidth limit values for the various MDMA channels. As shown, both the individual and the overall and the expected throughput are very close to each other.

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 use the bandwidth limit feature for memory pipe DMA channels to regulate the overall DMA traffic.
- Program the DMA channels' `MSIZE` parameters to optimal values to maximize throughput and avoid any potential underflow/overflow conditions.
- If required/possible, split single MDMA of a smaller `MSIZE` value into multiple descriptor-based MDMA transfers to maximize the usage of a larger `MSIZE` values for better performance.
- Use MDMA instead of EMDMA for sequential data transfers to improve performance. If possible, emulate EMDMA non-sequential transfer modes with MDMA.

- Program the SCB `RQOS` and `WQOS` registers to allocate priorities to various masters as per system requirements.
- Program the clock domain crossing (`IBx`) registers depending upon the clock ratios across SCBs.
- Use 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
- Since MMR latencies affect the interrupt service latency, ADSP-2156x 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 offloading processing from the core. For a detailed discussion on this topic, refer to application note *Utilizing the Trigger Routing Unit for System Level Synchronization (EE-360)*[6]. Although the note is written for the ADSP-BF60x processor, the concepts can be used for ADSP-2156x processors as well.

References

[1] *ADSP-SC5xx/215xx SHARC+ Processor System Optimization Techniques (EE-401)*, Rev 1, February, 2018.Analog Devices, Inc.

[2] *ADSP-2156x SHARC+ Processor Hardware Reference*, Rev 0.3, March 2020. Analog Devices, Inc.

[3] *ADSP-21562/21563/21565/21566/21567/21569 SHARC+ Single core High Performance DSP Data Sheet.* Rev 0, March 2020. Analog Devices, Inc.

[4] *ADSP-2156x SSDD API Reference Manual for SHARC+ Core*, Version 2.0, CrossCore® Embedded Studio Help.

[5] *Associated ZIP file for EE-412: ADSP-2156x SHARC+ Processors System Optimization Techniques*, August 2020. Analog Devices, Inc.

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

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

Document History

Revision	Description
Rev 2 – September 3, 2020 *by Mitesh Moonat*	Revised DCLK frequency from 533 MHz to 667 MHz.
Rev 1 – October 30, 2019 *by Mitesh Moonat*	Initial release.