SHARC-FX Processor System Optimization

Submitted by: Tejaswi Chitneedi *Revision 1.0 – Sept 18, 2024*

Summary

The ADSP-2183x/ADSP-SC83x SHARC-FX 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, and available bandwidth optimization techniques.

Most of the theoretical content of this application note is the same as in ADSP-SC59x SHARC+ Processor System Optimization Techniques (EE-445) [6]. This application note includes the figures, tables, and data specific to the ADSP-2183x/SC83x family of processors

Customer Takeaways

This application note provides the following customer information:

- Optimization techniques to improve ADSP-2183x/ADSP-SC83x system throughput
- Identifies measured MMR latencies in core cycles
- Provides insight into different L1 and L2 memory throughput methods

ADSP-2183x/SC83x Processor Architecture

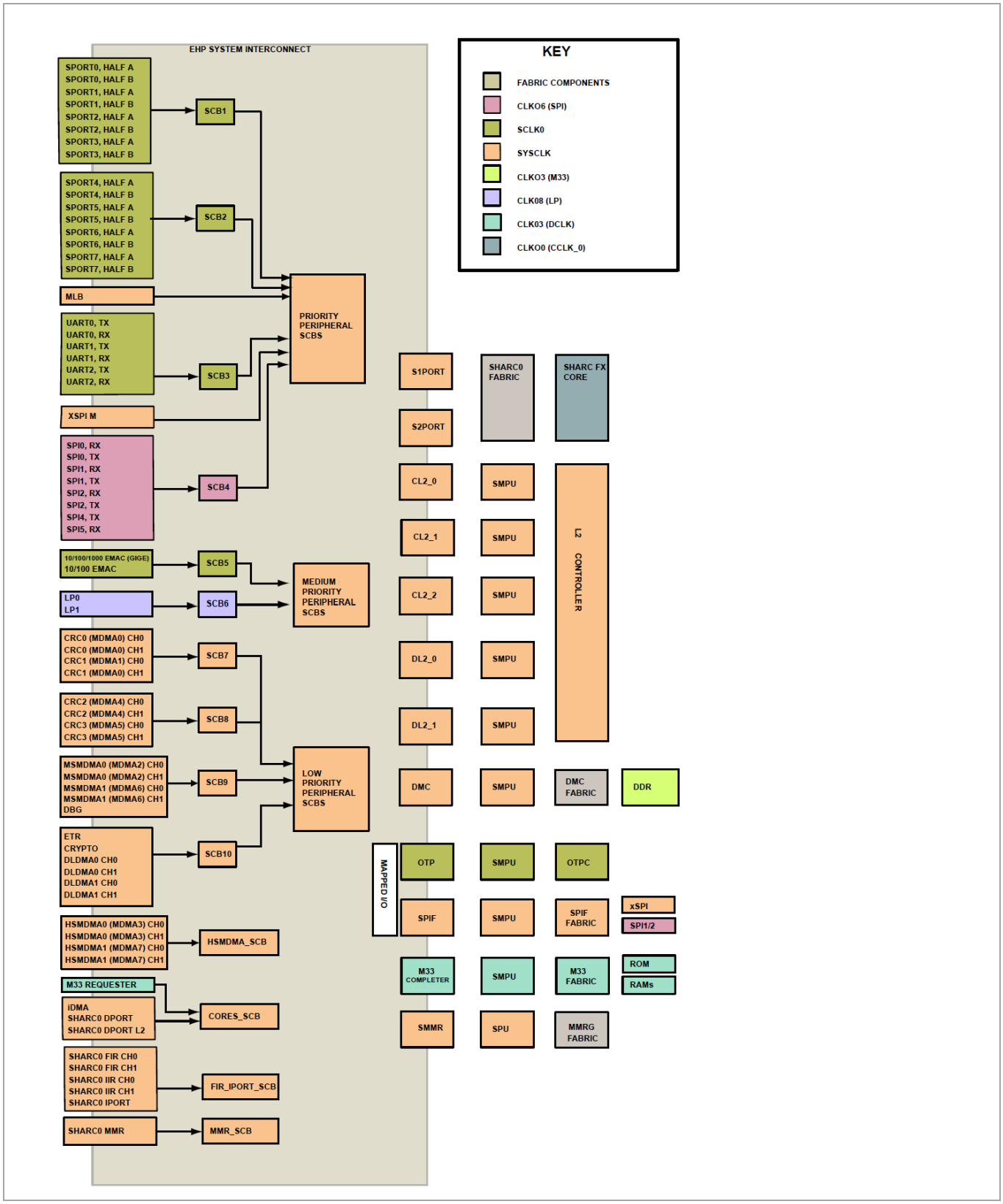
This section describes the ADSP-2183x/SC83x processor key architectural features that play a crucial role in system bandwidth and performance. For detailed information, refer to the *ADSP-2183x/ADSP-SC83x SHARC-FX Processor Hardware Reference*[1].

The overall architecture of the ADSP-2183x/ADSP-SC83x SHARC-FX processor consists of three main system components: system bus targets, system bus controllers, and system crossbars. Figure 1 and Figure 2 show how these components are interconnected to form the complete system.

System Bus Targets

As shown in Figure 1 system bus targets include on-chip and off-chip memory devices/controllers, such as L1 SRAM, L2 SRAM, memory-mapped peripherals (for example, SPI FLASH), and the System Memory Mapped Registers (MMRs). Each system bus target has its own latency characteristics, operating in a specific clock domain. For example, L1 SRAM runs at CCLK, L2 SRAM runs at SYSCLK, and so forth.

Figure 1: System Cross Bar (SCB) Block Diagram



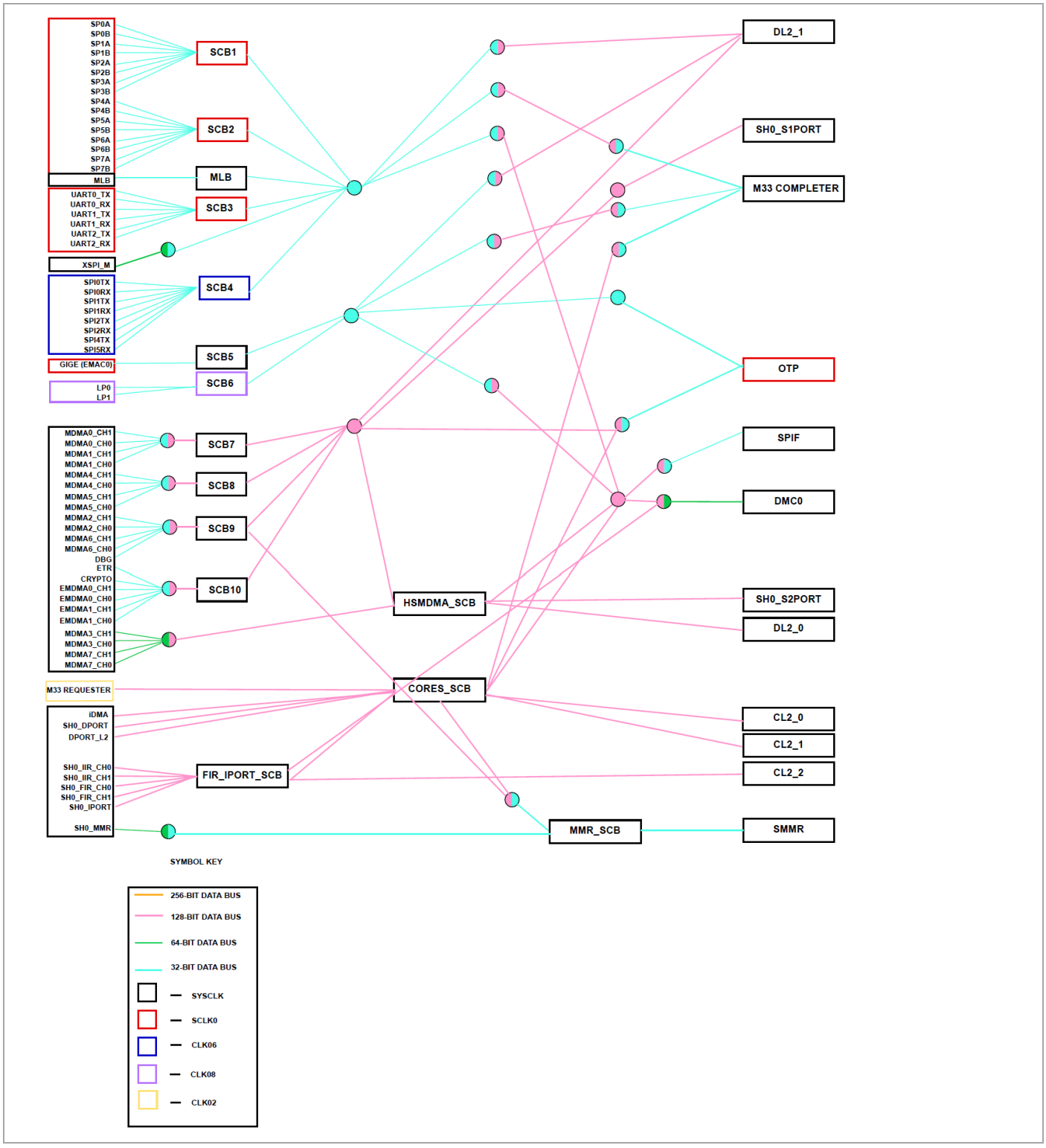
System Bus Controllers

The bottom of Figure 1 shows the system bus controllers. The controllers 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 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 controllers to system bus targets. They provide concurrent data transfer between multiple bus controllers and multiple bus targets, providing flexibility and full-duplex operation. The SCBs also provide a programmable arbitration model for bandwidth and latency management. The SCBs run on different clock domains (SCLK0, SYSCLK, SPI clock and LP clock) that introduce their own latencies to the system.

Figure 2: SCB Interconnections



System Latencies, Throughput, and Optimization Techniques

The following sections describe distinct aspects related to latencies and throughput of system bus controllers, system bus targets, and the system cross bars. The EE-note also discusses various optimization techniques to reduce system latencies and improve throughput.

Understanding the System Controllers

DMA Parameters

Each DMA channel has two buses: one that connects to the SCB, which, in turn, is connected to the SCB completer (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-2183x/ADSP-SC83x 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 completer (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-2183x/ADSP-SC83x processors support multiple MDMA streams (MDMA0/1/2/3/4/5/6/7) 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-2183x/ADSP-SC83x processors. Table 2 shows the various memory completers and the corresponding maximum theoretical bandwidth supported by the ADSP-2183x/ADSP-SC83x processors.

Table 1: DMA Streams and Maximum Theoretical Bandwidth

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)	CCLK-1000 SYSCLK-500 SCLKx-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
4	CRC2		45	46		32	2000
5	CRC3		47	48		32	2000
6	Enhanced Bandwidth or Medium Speed MDMA (MSMDMA1)		49	50		32	2000
7	Maximum Bandwidth or High Speed MDMA (HSMDMA1)		51	52		64	4000

Table 2 shows the various memory targets and the corresponding maximum theoretical bandwidth supported by the ADSP-2183x/SC83x processor.

Table 2: Memory Targets and Maximum Theoretical Bandwidth

Memory Type (L1/L2/L3)	Maximum CCLK/SYSCLK/SCLKx/ Frequency (MHz)	Clock Domain	Bus Width (Bits)	Data Rate Clock Rate	Maximum Theoretical Bandwidth (MB/s)
L1	CCLK-1000 SYSCLK-500 SCLKx-125 DCLKx-125	CCLK	32	1	4000
L2		SYSCLK	128	1	8000
L3		DCLK	16	2	3600

The actual (measured) MDMA throughput is always less than or equal to the minimum of the maximum theoretical throughput supported by: MDMA, source memory, or destination memory. For example, the measured throughput of MDMA0 between L1 and L2 is less than or equal to 2000 MB/s, which is limited by the maximum bandwidth of MDMA0. Figure 3 shows the actual throughput measured on the bench for various MDMA streams with different combinations of source and destination memories.

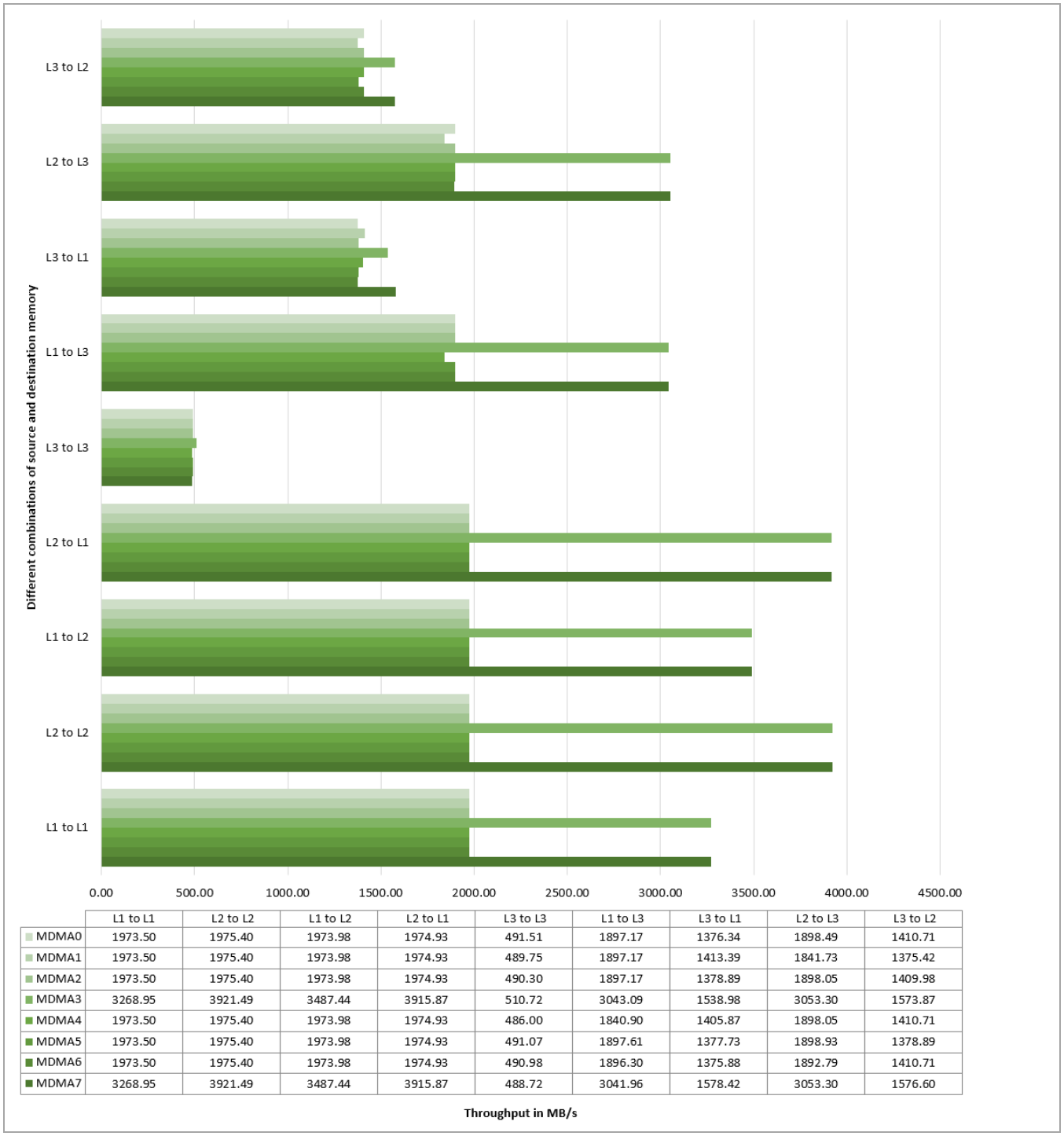
The measurements were taken using the following parameters:

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

- DCLK = 900 MHz

The code `Test1_MDMA_Throughput` supplied with this application note[7] can be used to measure MDMA throughput.

Figure 3: Measured MDMA Throughput on ADSP-2183x/SC83x Processor



	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	1973.50	1975.40	1973.98	1974.93	491.51	1897.17	1376.34	1898.49	1410.71
MDMA1	1973.50	1975.40	1973.98	1974.93	489.75	1897.17	1413.39	1841.73	1375.42
MDMA2	1973.50	1975.40	1973.98	1974.93	490.30	1897.17	1378.89	1898.05	1409.98
MDMA3	3268.95	3921.49	3487.44	3915.87	510.72	3043.09	1538.98	3053.30	1573.87
MDMA4	1973.50	1975.40	1973.98	1974.93	486.00	1840.90	1405.87	1898.05	1410.71
MDMA5	1973.50	1975.40	1973.98	1974.93	491.07	1897.61	1377.73	1898.93	1378.89
MDMA6	1973.50	1975.40	1973.98	1974.93	490.98	1896.30	1375.88	1892.79	1410.71
MDMA7	3268.95	3921.49	3487.44	3915.87	488.72	3041.96	1578.42	3053.30	1576.60

Optimizing Non-32-Byte-Aligned MDMA Transfers

In many cases, the start address and count of a MDMA transfer cannot be aligned to a 32-byte address boundary. In such cases, the `MSIZE` value needs 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 (when 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 CrossCore Embedded Studio (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 approximately 2.19 to 5.34 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.

Table 3: `adi_mdma_Copy1D` vs. `adi_mdma_Copy1DAuto` Performance

S. No.	Source Memory Address	Destination Memory Address	DMA Count	MSIZE (Bytes)	Copy1D MDMA Cycles	Copy1DAuto MDMA Cycles	Added API Overhead	Effective Improvement Factor
1	0x28240001	0x282B0000	256	1	3499	1203	392	2.19
2	0x28240000	0x282B0001	256	1	3499	1204	398	2.18
3	0x28240001	0x282B0000	1024	1	12721	2743	392	4.06
4	0x28240000	0x282B0001	1024	1	12721	2731	398	4.07
5	0x28240001	0x282B0000	4096	1	49579	8883	392	5.35
6	0x28240000	0x282B0001	4096	1	49579	8884	398	5.34

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 controllers 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 an example result of this code with the calculated and measured bandwidth for different MDMA use cases.

Figure 4: MDMA Bandwidth Limit Results

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

Measured BW = 389.816800 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 = 296.050016 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 = 694.296128 MB/s

Test combination 3 passed
```

The bandwidth monitor feature can be used to check whether 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 (MMR access only), a work unit ends, or an auto buffer 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. When 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 configured 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 an example result of this code with a calculated 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`.

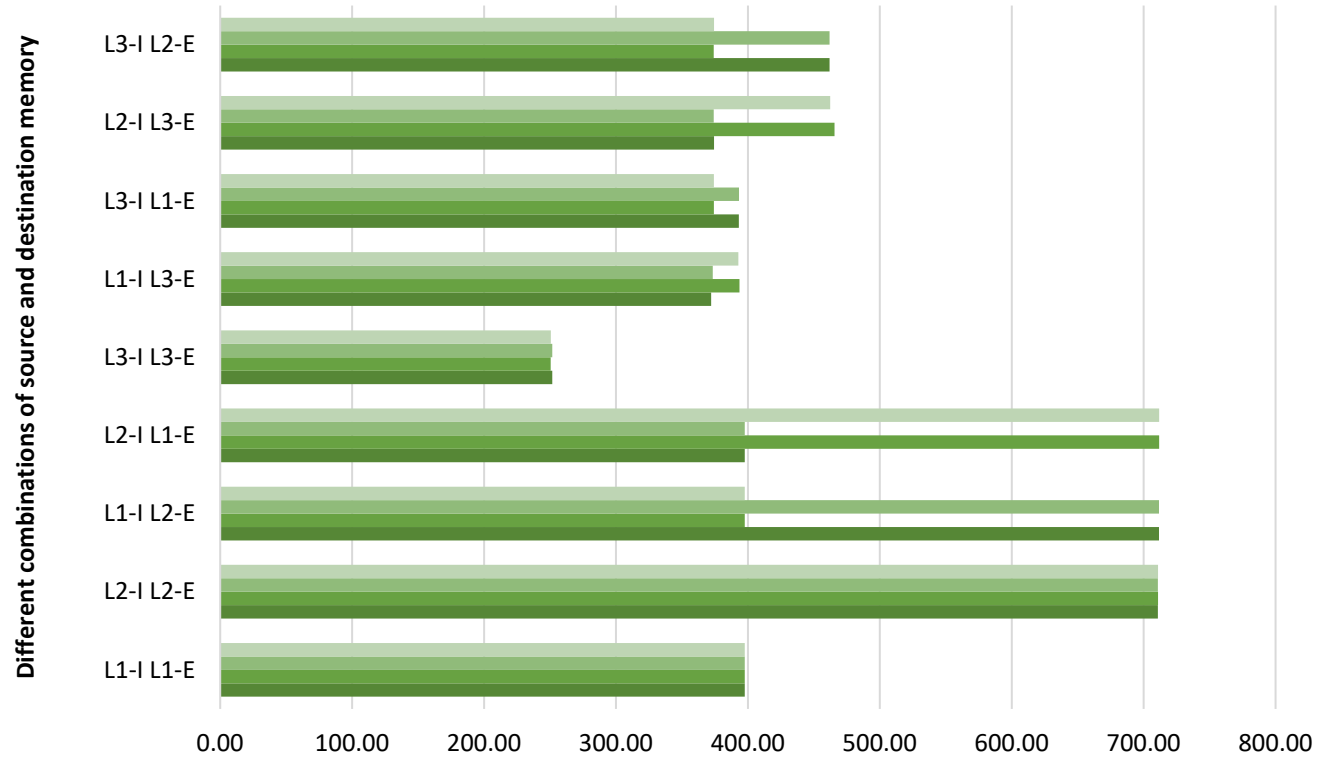
Figure 5: MDMA Bandwidth Monitor Results

```
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


Testing combination 4
MDMA Stream no = 3
MSIZE value = 32
Source channel = 1
Target BW = 600.000000 MB/s
BW Monitor Expired!! Test combination 4 passed
```

Extended Memory DMA (EMDMA)

The ADSP-2183x/SC83x processor also supports Extended Memory DMA (EMDMA). The EMDMA engine is 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 about the EMDMA, refer to the *ADSP-2183x/ADSP-SC83x SHARC-FX Processor Hardware Reference* [1]. The EMDMA on the processor is 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 and SYSCLK = 500 MHz and DCLK = 900MHz.

Figure 6: Measured EMDMA Processor Throughput



	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	397.63	710.86	397.67	711.73	250.62	392.79	374.22	462.35	374.32
EMDMA0_Read	397.61	710.80	711.67	397.65	251.73	373.40	393.26	374.08	461.96
EMDMA1_Write	397.63	710.86	397.67	711.73	250.55	393.64	374.23	465.64	374.12
EMDMA1_Read	397.61	710.80	711.67	397.65	251.71	372.14	393.13	374.34	461.94

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 `Test6_MDMA_circularbuffer` 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 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)

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.

Table 4: MDMA Emulated Circular Buffer vs. EMDMA

Write/Read	Core Cycles		
	EMDMA	MDMA3 MSIZE = 4 bytes	MDMA3 MSIZE = 32 bytes
Write	36120	29824	6264
Read	46494	36004	11514

Understanding the System Crossbars

For more information on system crossbars, refer to the System Crossbars (SCB) chapter in the Hardware Reference Manual.

Understanding the System Targets

Memory Hierarchy

As shown in Table 2, the ADSP-2183x/SC83x processors have a hierarchical memory model (L1/L2/L3). The following sections discuss the access latencies and the achieved throughput associated with the different memory levels.

L1 Memory Throughput

L1 memory runs at CCLK and is the fastest accessible memory in the hierarchy. From a programming perspective, when accessing the L1 memory of the SHARC-FX processer, use a multiprocessor memory offset of `0x28000000` for all DMA accesses.

The maximum theoretical throughput of L1 memory (for system/DMA accesses) is 1000 * 4 = 4000 MB/s for 1GHz CCLK operation. As shown in Figure 3, the maximum measured L1 throughput using MDMA3 is approximately ~3921 MB/s.

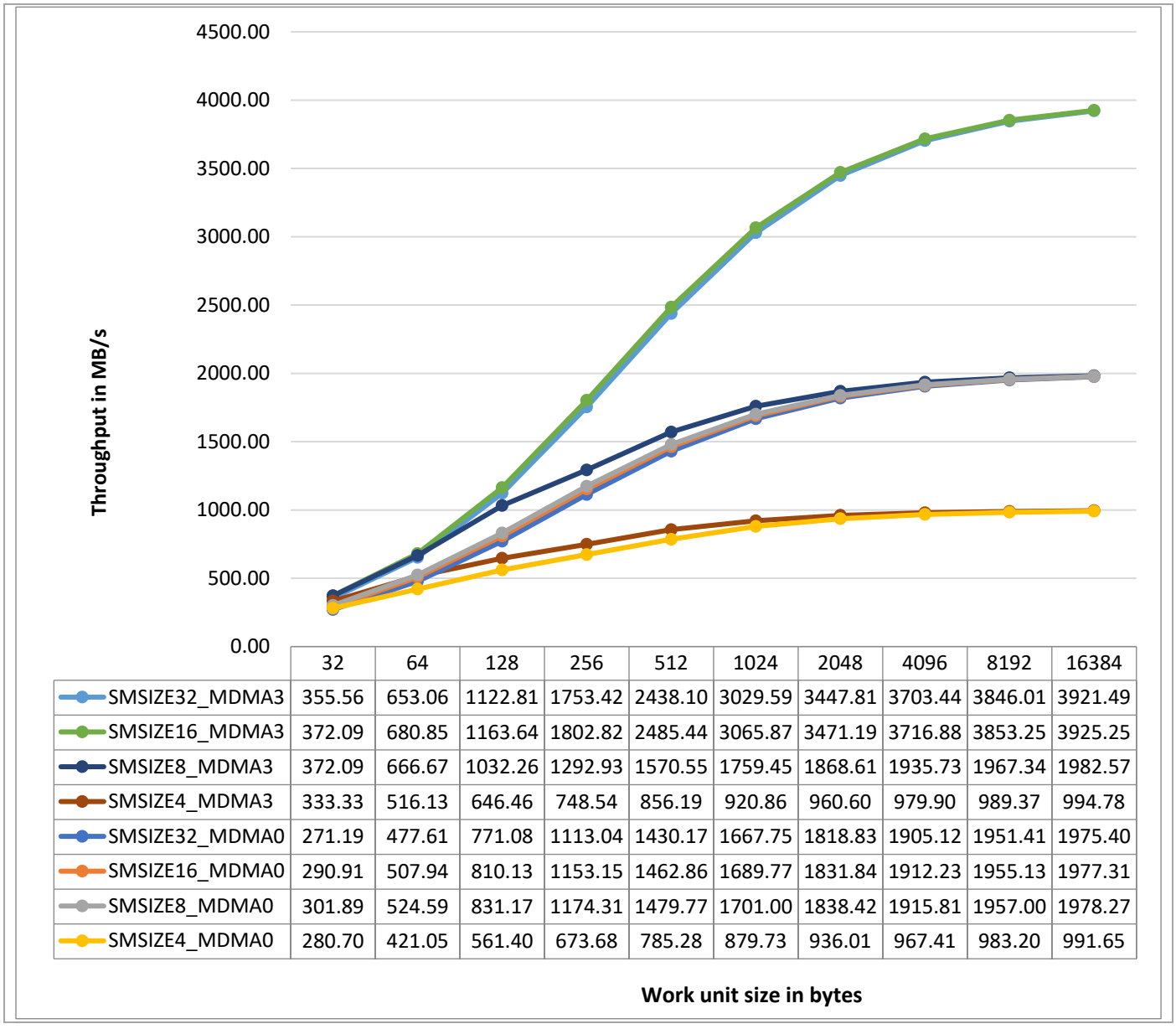
Consider the following important points regarding L2 memory throughput:

- L2 memory access times are longer than L1 because the maximum L2 clock frequency (SYSCLK) is half the CCLK. The L2 memory controller contains five ports to connect to the system crossbar. These are 128-bit interfaces that connect through DMA. Each port has a read and a write channel. For details, refer to the *ADSP-2183x/ADSP-SC83x SHARC-FX Processor Hardware Reference* [1].
- Because 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 has a maximum speed of 4000 MB/s). Because 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 128-bit accesses (16-byte) by the L2 memory controller. To achieve optimum throughput for DMA access to L2 memory, configure the DMA channel `MSIZE` to 16 bytes or larger.
- L2 memory throughput for sequential and non-sequential accesses is the same.
- L2 SRAM is ECC-protected.
- 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. When the core and the DMA engine access the same bank, the best access rate that DMA can achieve is one 128-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 zero, while programming the `L2CTL_RPCR.RPC1` and `L2CTL_WPCR.WPC1` bits to one.

Figure 7 shows the measured MDMA throughput at CCLK = 1 GHz and SYSCLK = 500 MHz for an example where both source and destination buffers are in different L2 memory banks. As an example, for MDMA3, the maximum throughput approximates 2000 MB/sec in one direction (4000 MB/s in both directions) for `MSIZE` = 32 bytes and drops significantly for smaller `MSIZE` values.

Figure 7: L2 MDMA Throughput for Different MSIZE and Work Unit Sizes



	32	64	128	256	512	1024	2048	4096	8192	16384
SMSIZE32_MDMA3	355.56	653.06	1122.81	1753.42	2438.10	3029.59	3447.81	3703.44	3846.01	3921.49
SMSIZE16_MDMA3	372.09	680.85	1163.64	1802.82	2485.44	3065.87	3471.19	3716.88	3853.25	3925.25
SMSIZE8_MDMA3	372.09	666.67	1032.26	1292.93	1570.55	1759.45	1868.61	1935.73	1967.34	1982.57
SMSIZE4_MDMA3	333.33	516.13	646.46	748.54	856.19	920.86	960.60	979.90	989.37	994.78
SMSIZE32_MDMA0	271.19	477.61	771.08	1113.04	1430.17	1667.75	1818.83	1905.12	1951.41	1975.40
SMSIZE16_MDMA0	290.91	507.94	810.13	1153.15	1462.86	1689.77	1831.84	1912.23	1955.13	1977.31
SMSIZE8_MDMA0	301.89	524.59	831.17	1174.31	1479.77	1701.00	1838.42	1915.81	1957.00	1978.27
SMSIZE4_MDMA0	280.70	421.05	561.40	673.68	785.28	879.73	936.01	967.41	983.20	991.65

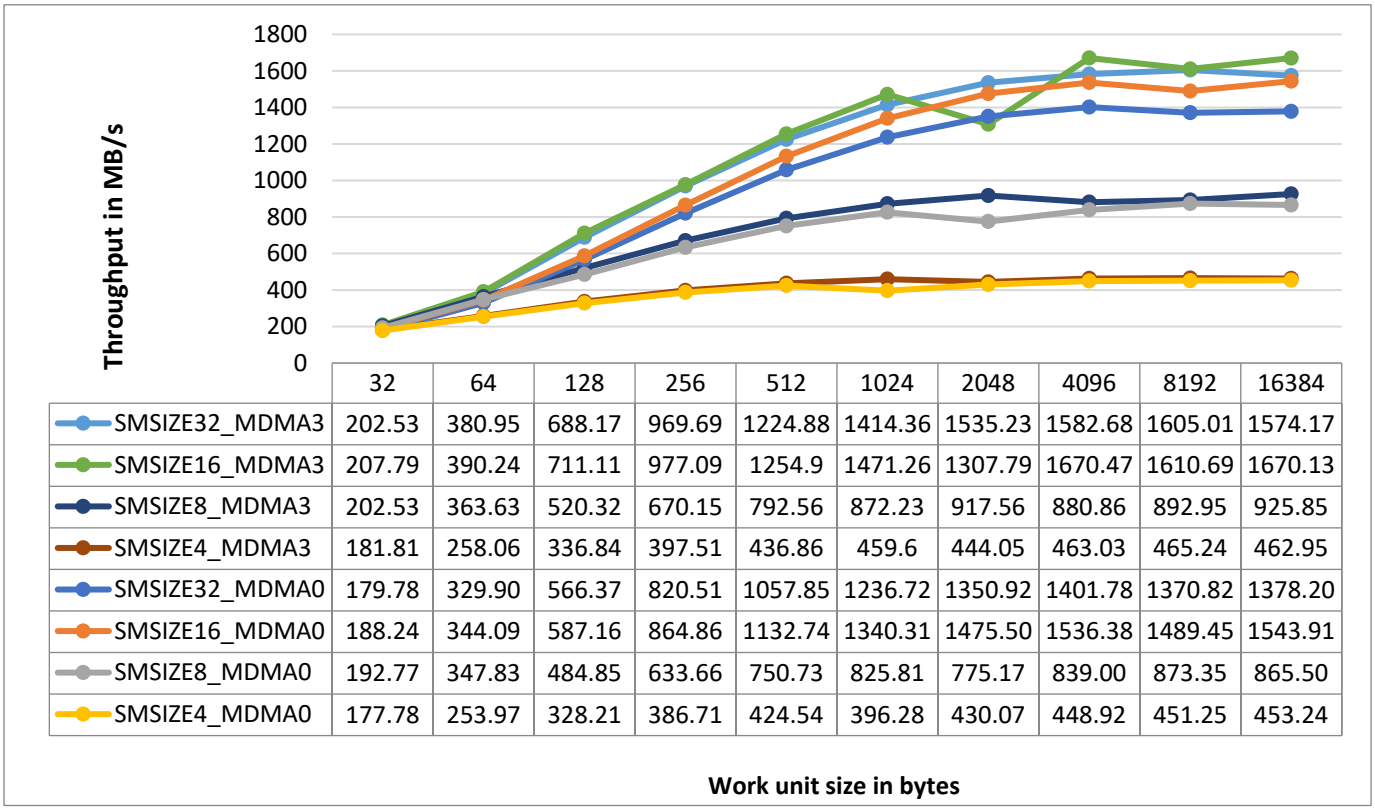
L3 Memory and External Memory Throughput

The ADSP-2183x/SC83x processors provide interfaces for connecting to DDR3/DDR3L memory devices. The DMC interface operates at speeds of up to 900 MHz. For the 16-bit DDR3 interface, the maximum theoretical throughput that the DMC can deliver equals 3600 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 requesters as well.

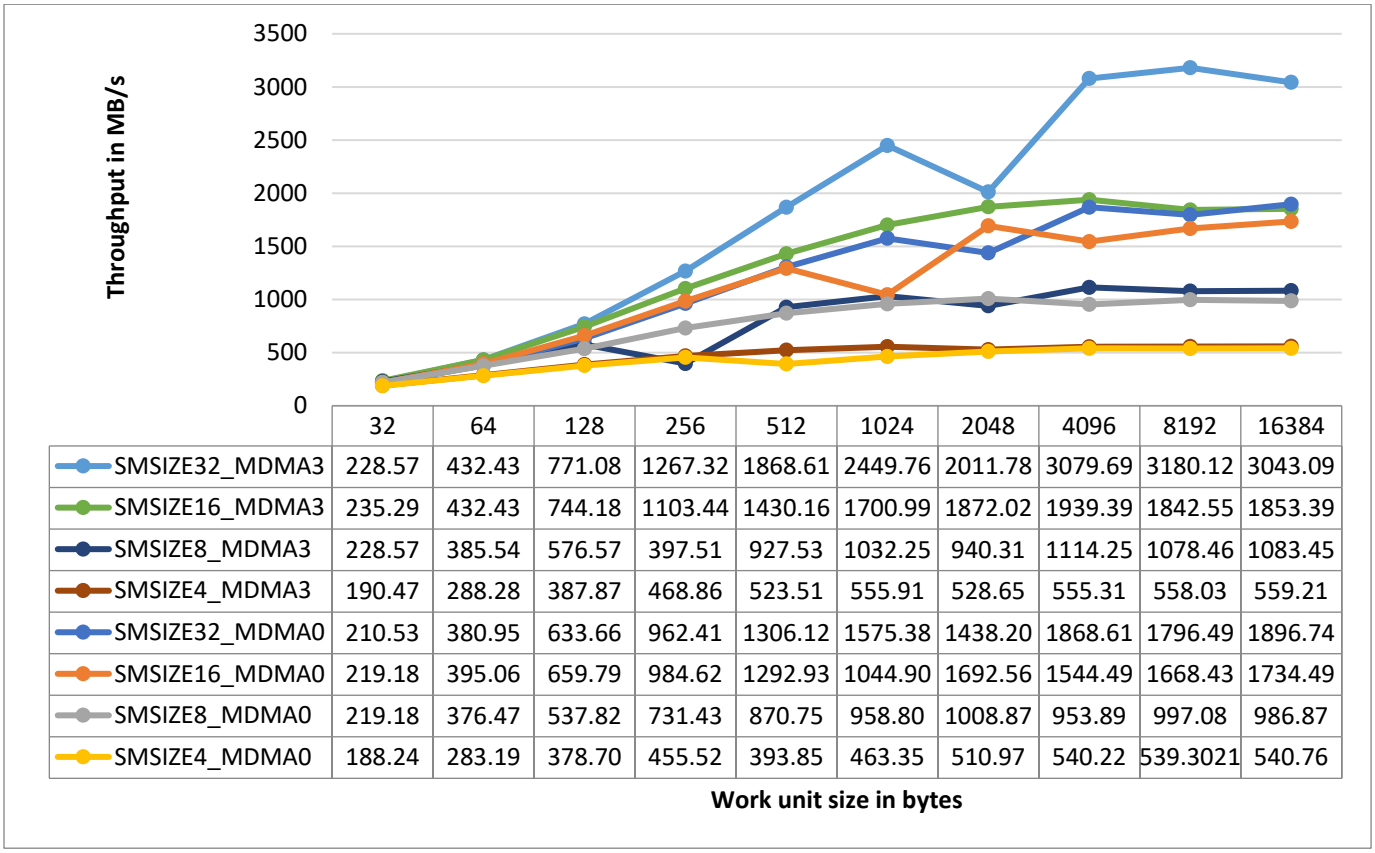
The MDMA3 stream (HSMDMA) can request DMC accesses faster than any other requesters (for example, 4000 MB/s). The practical DMC throughput possible using MDMA depends upon factors such as whether the accesses are sequential or non-sequential, the block size of the transfer, and DMA parameters (for example, `MSIZE`). Figure 8 and Figure 9 provide the DMC measured throughput at CCLK = 1 GHz and DCLK = 900 MHz using MDMA0 (channels 8 and 9) and MDMA3 (channels 43 and44) streams for various `MSIZE` values and buffer sizes.

Figure 8: DMC Measured Throughput for Sequential MDMA Reads



	32	64	128	256	512	1024	2048	4096	8192	16384
SMSIZE32_MDMA3	202.53	380.95	688.17	969.69	1224.88	1414.36	1535.23	1582.68	1605.01	1574.17
SMSIZE16_MDMA3	207.79	390.24	711.11	977.09	1254.9	1471.26	1307.79	1670.47	1610.69	1670.13
SMSIZE8_MDMA3	202.53	363.63	520.32	670.15	792.56	872.23	917.56	880.86	892.95	925.85
SMSIZE4_MDMA3	181.81	258.06	336.84	397.51	436.86	459.6	444.05	463.03	465.24	462.95
SMSIZE32_MDMA0	179.78	329.90	566.37	820.51	1057.85	1236.72	1350.92	1401.78	1370.82	1378.20
SMSIZE16_MDMA0	188.24	344.09	587.16	864.86	1132.74	1340.31	1475.50	1536.38	1489.45	1543.91
SMSIZE8_MDMA0	192.77	347.83	484.85	633.66	750.73	825.81	775.17	839.00	873.35	865.50
SMSIZE4_MDMA0	177.78	253.97	328.21	386.71	424.54	396.28	430.07	448.92	451.25	453.24

Figure 9: DMC Measured Throughput for Sequential MDMA Writes



	32	64	128	256	512	1024	2048	4096	8192	16384
SMSIZE32_MDMA3	228.57	432.43	771.08	1267.32	1868.61	2449.76	2011.78	3079.69	3180.12	3043.09
SMSIZE16_MDMA3	235.29	432.43	744.18	1103.44	1430.16	1700.99	1872.02	1939.39	1842.55	1853.39
SMSIZE8_MDMA3	228.57	385.54	576.57	397.51	927.53	1032.25	940.31	1114.25	1078.46	1083.45
SMSIZE4_MDMA3	190.47	288.28	387.87	468.86	523.51	555.91	528.65	555.31	558.03	559.21
SMSIZE32_MDMA0	210.53	380.95	633.66	962.41	1306.12	1575.38	1438.20	1868.61	1796.49	1896.74
SMSIZE16_MDMA0	219.18	395.06	659.79	984.62	1292.93	1044.90	1692.56	1544.49	1668.43	1734.49
SMSIZE8_MDMA0	219.18	376.47	537.82	731.43	870.75	958.80	1008.87	953.89	997.08	986.87
SMSIZE4_MDMA0	188.24	283.19	378.70	455.52	393.85	463.35	510.97	540.22	539.3021	540.76

The following important observations can be made for the ADSP-2183x/SC83x processor:

- The throughput trends are similar for reads and writes with regards to `MSIZE` and buffer size values.
- The peak measured read throughput is 1670.47 MB/s for MDMA3 with `MSIZE` = 16 bytes. The peak measured write throughput is 3079.69 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 202 MB/s, whereas it reaches 1574 MB/s for a 16 KB buffer size. This difference is affected by 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.

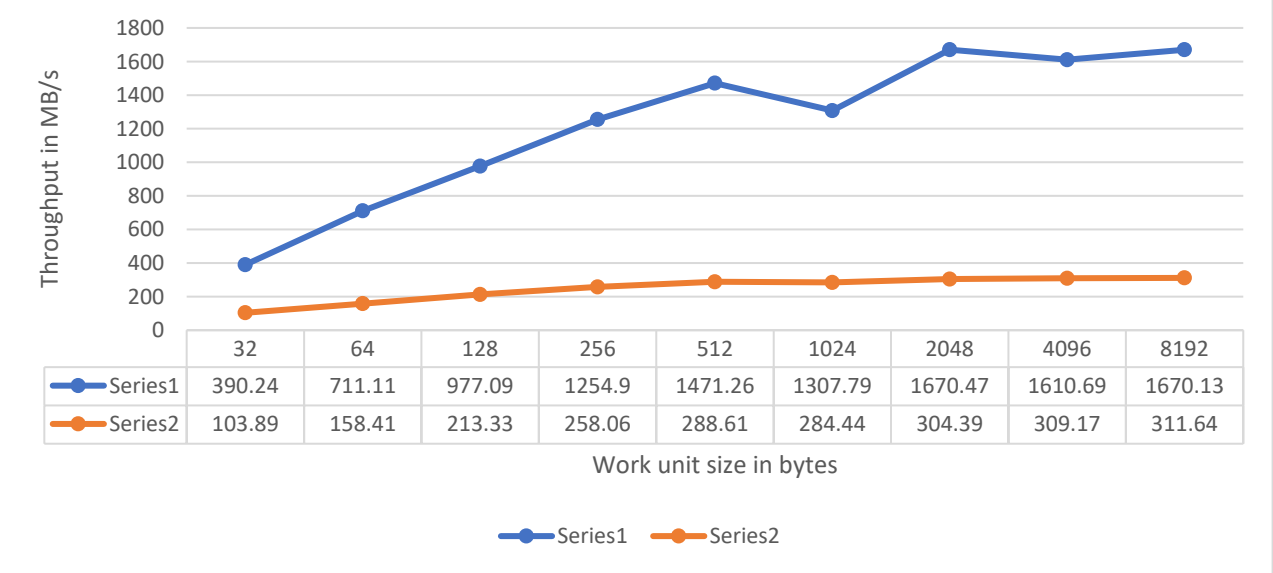
Try to rearrange the DMC accesses such that the DMA count is as large as possible. Better sustained throughput is obtained for continuous transfers over time.

To some extent, throughput also depends upon the `MSIZE` value of the source MDMA channel for reads and destination MDMA channel for writes. For Figure 8 and Figure 9, 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 1574 MB/s for `MSIZE` = 32 bytes, while it reduces significantly to 462 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 10 provides a comparison of the DMC throughput numbers measured for sequential MDMA read accesses for `MSIZE` = 16 bytes (equals DDR3 burst length) and `ADDRMODE`= 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 1670 to 311 MB/s.

Figure 10: DMC Throughput for Sequential vs. Non-Sequential Read Accesses

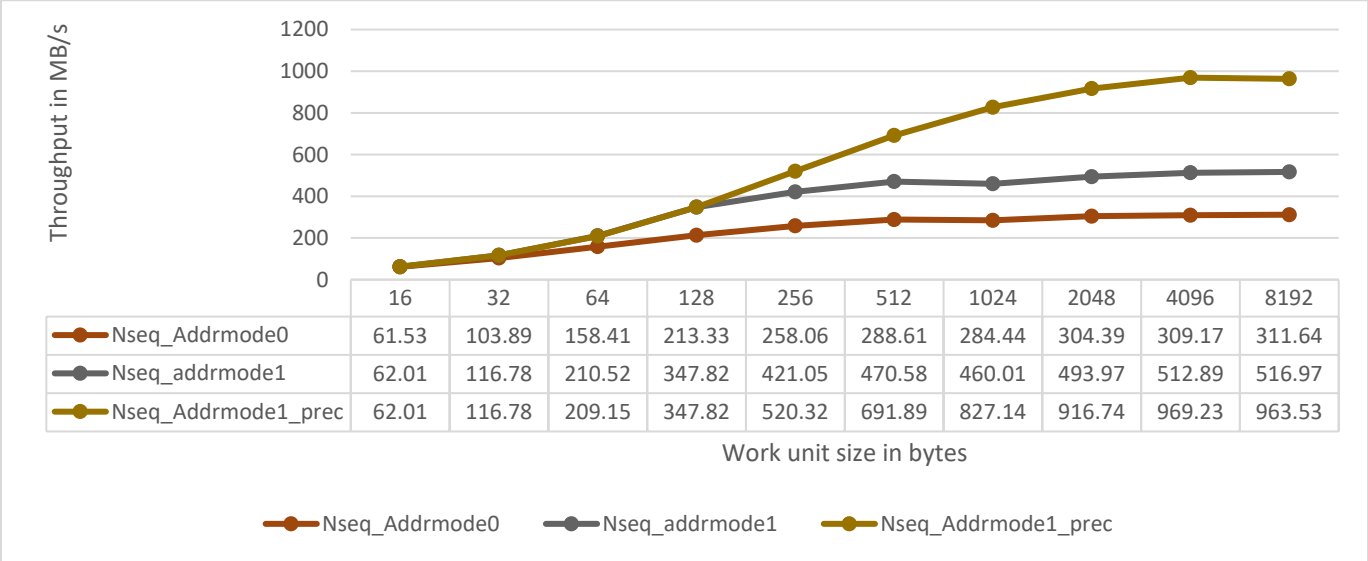


DDR memory devices support concurrent bank operations that provide the DMC controller the feature to activate a row in another bank without pre-charging the row of a bank. This feature is extremely helpful in cases where DDR access patterns incur page misses. By setting the `DMC_CTL.ADDRMODE` bit, throughput can be improved by ensuring that such accesses fall into different banks. Figure 11 shows how the DMC throughput increases from 311 MB/s to 516 MB/s by setting this bit.

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 pre-charge after it has been accessed. It improves the throughput by saving the latency involved in pre-charging the row at the time when the next row of the same bank must be activated.

Figure 11 shows the increase in throughput. Setting the `DMC_CTL.PREC` bit results in an increase from 516 MB/s to 963 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 pre-charging 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.

Figure 11 Optimizing Throughput for Non-Sequential Accesses



The DMC also provides elevating the priority of the accesses requested by a SCB Requester using the `DMC_PRIO` and `DMC_PRIOMSK` registers. The example source code found in the `Test10_DMC_SCB_PRIO` subfolder of the associated zip file, describes how two MDMA DMC read channels (8 and 18) run in parallel. Table 5 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.

Table 5 DMC Measured Throughput for Different DMC_PRIO Settings

Test Case Number	Priority Channel (SCB ID)	MDMA0 (Ch. 8) Throughput (MB/s)	MDMA1 (Ch. 18) Throughput (MB/s)
1	None	1051	1053
2	MDMA0 (0x0031)	1071	1237
3	MDMA1 (0x0011)	1297	1108

IDMA Throughput Measurement

The integrated DMA (iDMA) engine is in the Xtensa processor. It allows fast data movement between the AXI port and the local data memories, data movement between external memory and data RAM, and between data RAM and data RAM. All iDMA operations are controlled by the Xtensa processor through DMA registers and

DMA descriptors, which are data structures residing in data RAM. The typical way to operate the iDMA is to prepare descriptors in data RAM and instruct the iDMA to fetch and run a set of descriptors under the control of the iDMA registers. Software can check the iDMA status by reading the iDMA status registers. The iDMA registers provide control and status reporting. Use the iDMA control registers to specify the parameters of the data movement, for example, how many descriptors to run, what the allowed AXI access burst length are, how many outstanding bus requests are allowed, etc. The iDMA can also be programmed with interrupts that notify the Xtensa processor if a particular descriptor finishes or when an error occurs.

The iDMA does not support external-to-external memory transfers, data movement to/from any device that has read side effects (for example, a FIFO), control registers, core register, cache, or other non-memory devices. The data transfer direction should be consistent throughout each DMA command. Data transfers should not cross the data RAM boundary. The iDMA executes the DMA commands in the format of DMA descriptors. The descriptors are stored in the data RAM. A 1D transfer is described by a 128-bit descriptor. A 2D transfer is described by a 256-

bit descriptor. The 2D-predicated row transfers are only supported by a 512-bit descriptor. Note that the 512-bit descriptor is available only for the 3 cycle DSP configurations.

The iDMA hardware receives a copy request in the form of an iDMA descriptor. The hardware fetches descriptors consecutively, one after another, as long as its control registers indicate there are more descriptors to process. There are four types of descriptors:

- One-dimensional (1D) descriptors contain the source address, the destination address, and the transfer size.
- Two-dimensional descriptors (2D) are used to copy matrices (tiles) and contain parameters such as row size, source pitch, and destination pitch, which are used to specify a matrix row size and the distance between rows.
- A special JUMP command (descriptor) redirects the flow to another consecutive list of descriptors.
- All the descriptor types are of different sizes. The iDMA hardware knows the location of the next descriptor to fetch after decoding the currently executing descriptor. For more information, please refer to Xtensa Microprocessor Data Book and Xtensa System Software Manual .

Figure 12 - Figure 15 shows the actual throughput measured on the bench for IDMA with different combinations of source and destination memories by placing either of source or destination residing in L1 memory.

The code `IDMA_throughput_test` supplied with this application note[7] can be used to measure IDMA throughput.

Figure 12 Measured IDMA Throughput for transfer between L1-L2 on ADSP-2183x/SC83x Processor

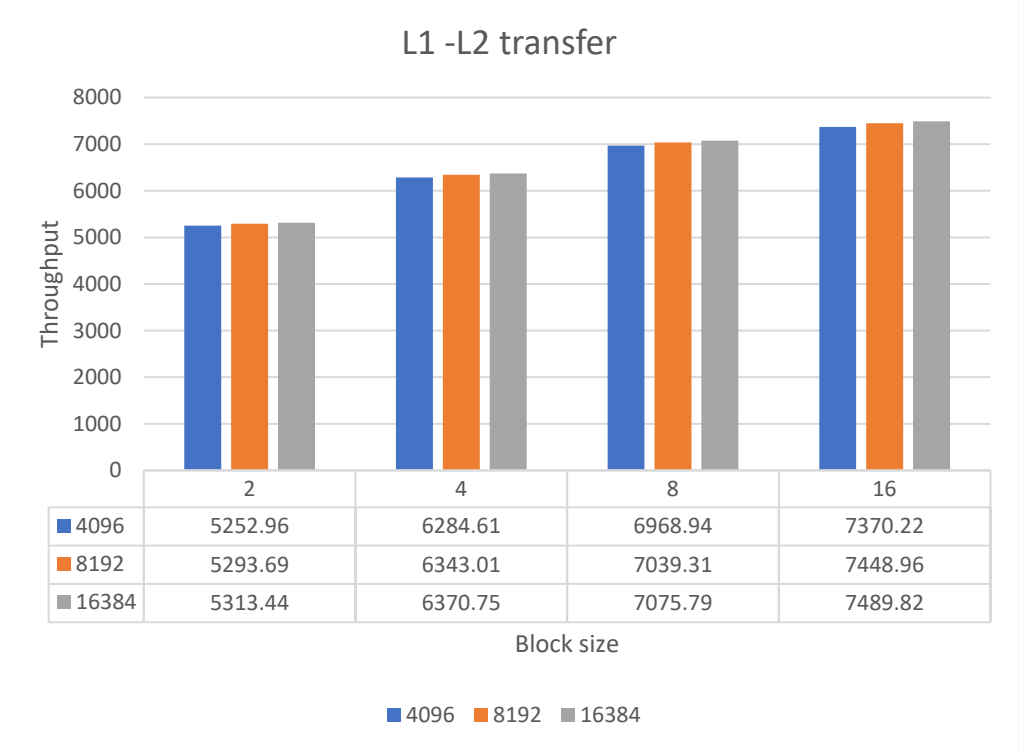
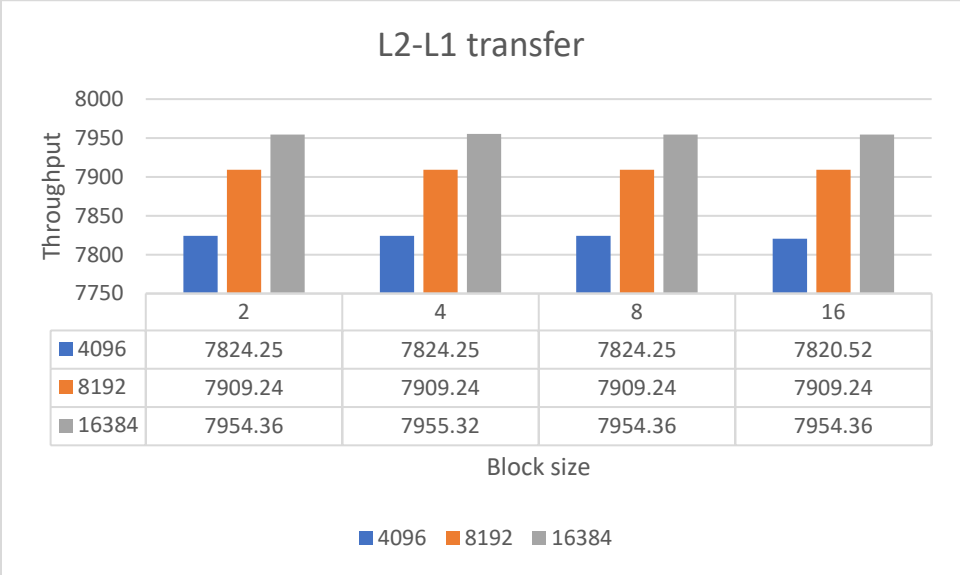
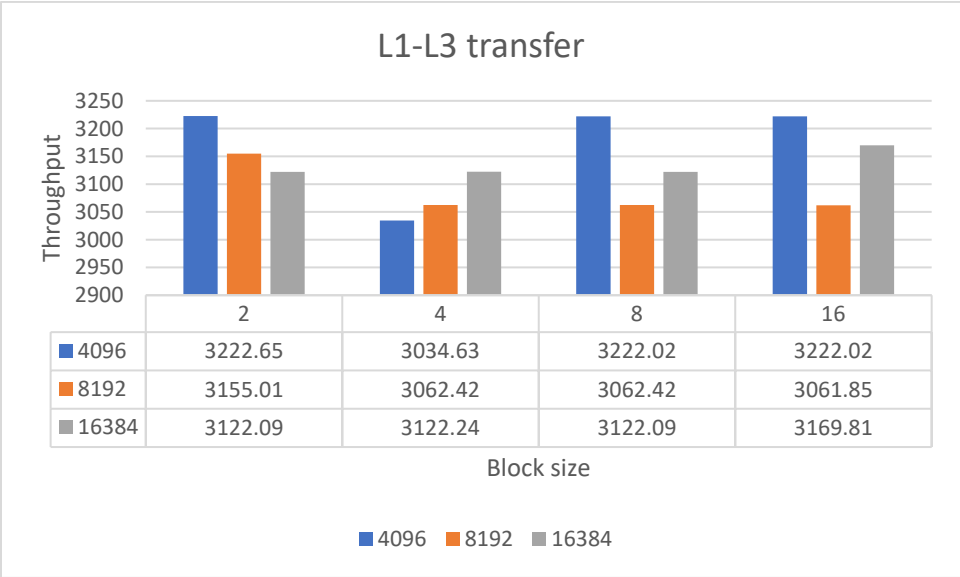


Figure 13 Measured IDMA Throughput for transfer between L2-L1 on ADSP-2183x/SC83x Processor



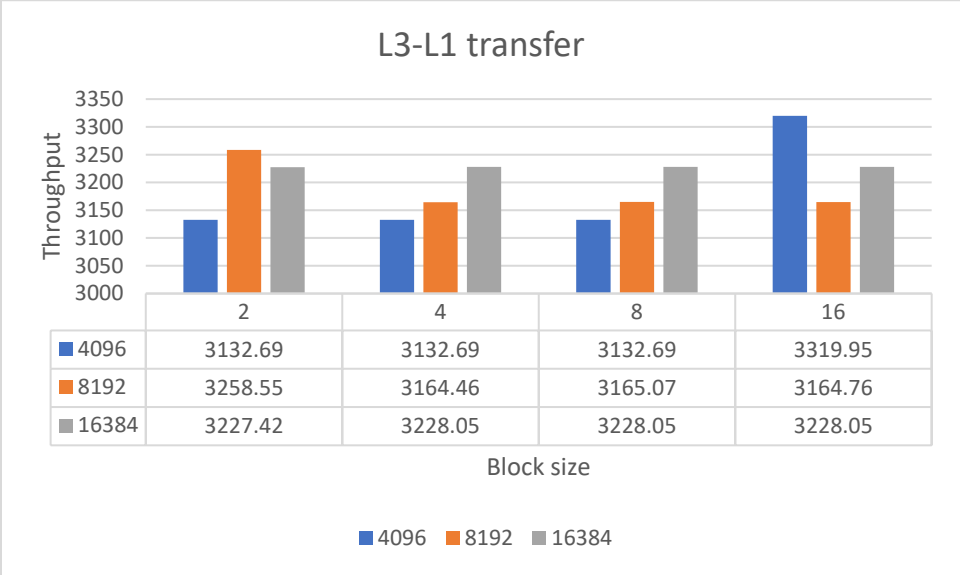
	2	4	8	16
4096	7824.25	7824.25	7824.25	7820.52
8192	7909.24	7909.24	7909.24	7909.24
16384	7954.36	7955.32	7954.36	7954.36

Figure 14 Measured IDMA Throughput for transfer between L1-L3 on ADSP-2183x/SC83x Processor



	2	4	8	16
4096	3222.65	3034.63	3222.02	3222.02
8192	3155.01	3062.42	3062.42	3061.85
16384	3122.09	3122.24	3122.09	3169.81

Figure 15 Measured IDMA Throughput for transfer between L3-L1 on ADSP-2183x/SC83x Processor



System MMR Latencies

Table 6 shows the measured MMR latency in core cycles for different peripherals on the ADSP-2183x/SC83x processor. The measurement was taken with CCLK = 1GHz, SYSCLK = 500 MHz, and SCLK = 125 MHz. These numbers can be used to approximate the MMR access latency of the SHARC-FX core for different peripherals.

Table 6: MMR Access Latency Processors in Approximate CCLK Cycles

S. No.	Register	Peripheral	Write Latency (Core Cycles)	Read Latency (Core Cycles)
1	FIR0_INIDX	FIR	16	16
2	IIR0_INIDX	IIR	16	16
3	MEC1_PERR_IMASK0	MEC	16	16
4	CRC0_DCNT	CRC	16	16
5	CRC1_DCNT		16	16
6	EMDMA0_INDX1	EMDMA	56	56
7	EMDMA1_INDX1		16	16
8	TAPC_SDBGKEY0	TAPC	16	16
9	L2CTL0_RPCR0	L2CTL	16	16
10	SEC0_RAISE	SEC	16	16
11	TRU0_SSR0	TRU	16	16
12	SPU0_SECUREP10	SPU	16	16
13	RCU0_MSG	RCU	16	16
14	CDU0_CLKINSEL	CDU	16	16
15	DPM0_PER_DIS0	DPM	16	16
16	PKTE0_SA_ADDR	PKTE	16	16
17	TRNG0_OUTPUT0	TRNG	16	16

S. No.	Register	Peripheral	Write Latency (Core Cycles)	Read Latency (Core Cycles)
18	PKA0_APTR	PKA	16	16
19	PKIC0_ACK	PKIC	16	16
20	SPORT1_DIV_A	SPORT	16	16
21	PINT1_ASSIGN	PINT	16	16
22	OTPC_PMC_MODE0	OTPC	31	57
23	DMA0_XCNT	DMA	16	16
24	PORTB_DATA_SET	PORT	16	16
25	WDOG1_WIN	WDOG	16	16
26	DMA1_XCNT	DMA	8	8
27	SPORT0_DIV_A	SPORT	16	16
28	UART0_CLK	UART	16	16
29	UART1_CLK	UART	16	16
30	PINT0_ASSIGN	PINT	16	16
31	WDOG0_WIN	WDOG	16	16
32	SPI1_CLK	SPI	16	16
33	PORTA_DATA_SET	PORTA	16	16
34	PADS0_PORTA_PDE	PADS	16	16
35	CNT0_CNTR	CNT	16	16
36	TIMER0_TMR0_WID	TIMER	16	16
37	SPI0_CLK	SPI	16	16
38	PCG0_PW1	PCG	16	16
39	SPDIF0_TX_UBUFF_A0	SPDIF	16	16
40	ASRC1_MUTE	ASRC	16	16
41	DAI1_IMSK_FE	DAI	16	16
42	SPDIF1_TX_UBUFF_A0	SPDIF	16	16
43	DAI0_IMSK_FE	DAI	16	16
44	ASRC0_MUTE	ASRC	16	16
45	SMPU2_RADDR0	SMPU	16	16
46	TWI0_CLKDIV	TWI	17	17
47	TWI1_CLKDIV	TWI	11	11
48	SMPU11_CTL	SMPU	16	16
49	HADC0_CHAN_MSK	HADC	16	16
50	TMU0_FLT_LIM_HI	TMU	16	16
51	DMC0_PRIO	DMC	16	16
52	CGU0_OSCWDCTL	CGU	16	16
53	SWU1_CTL0	SWU	16	16
54	PDM0_CTL0	PDM	16	16

S. No.	Register	Peripheral	Write Latency (Core Cycles)	Read Latency (Core Cycles)
55	PDM1_CTL0	PDM	16	16
56	MLB0_MDAT0	MLB	19	19
57	MISCREG_CAN_SYSCTL	MISC	19	19
58	LP0_DIV	LP	19	19
59	LP1_DIV	LP	19	19

Note: The MMR latency numbers are measured with the "d*sync* "instruction after the write. This ensures that the write has taken affect.

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 system MMR accesses**–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)**–where the code is executed (L1/L2/L3)

System Bandwidth Optimization Procedure

Although the optimization techniques can vary from one application to another, the general procedure for bandwidth optimization remains the same. Figure 16 provides a flow chart of typical steps used in a system bandwidth optimization procedure for ADSP-2183x/SC83x processor-based applications.

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

1. Identify the individual and total throughput requirements for all the requesters accessing the corresponding SCB completer in the system. Consider the total throughput requirement as *X*.
2. Calculate the observed throughput the corresponding SCB completer(s) can supply under the specific conditions. Refer to this calculated value as *Y*.
3. 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.

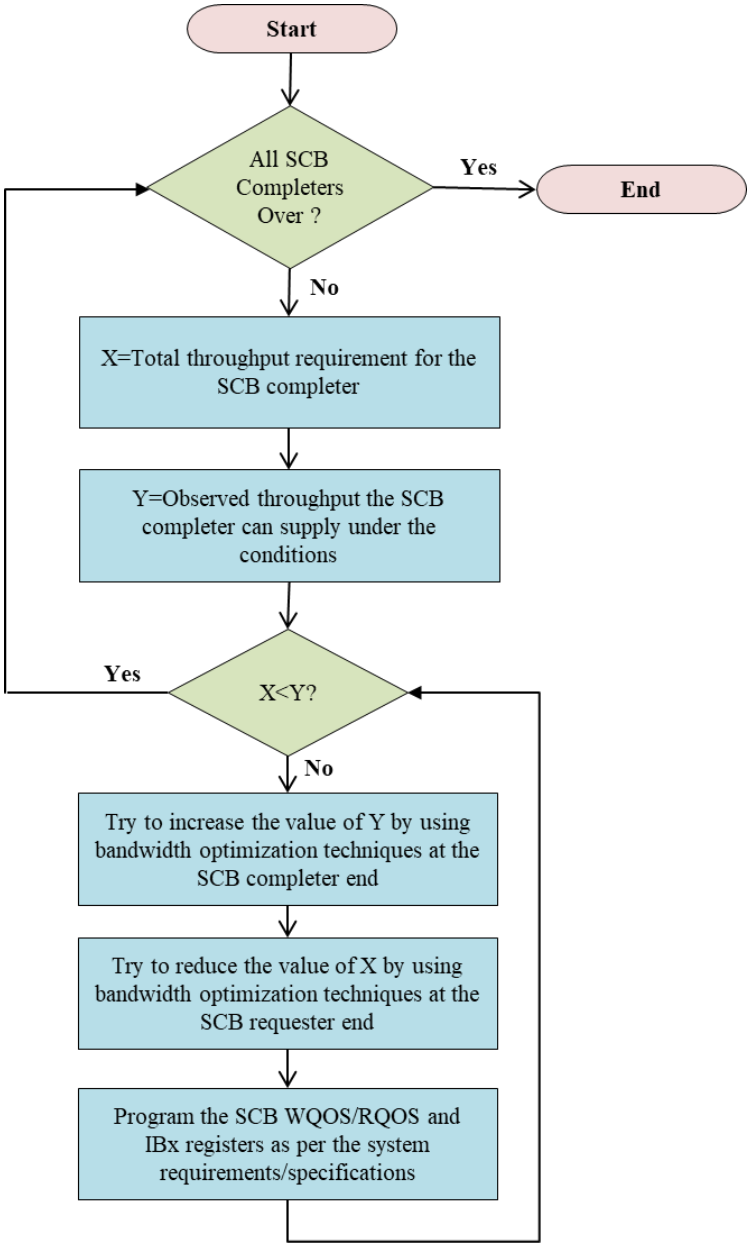
In this case, apply the bandwidth optimization techniques to:

- *Increase* the value of *Y* by applying the completer-specific optimization techniques (for example, using the DMC efficiency controller features).
- *Decrease* the value of *X*: Reanalyze whether a peripheral needs to run that fast. If not, then slow down the peripheral to reduce the bandwidth requested by the peripheral.
- Reanalyze whether an MDMA can be slowed down. The corresponding `DMAx_BWLCNT` register can be used to limit the bandwidth of that DMA channel.

Application Example

Figure 17 shows a block diagram of the example application code found in the `Test12_Multiple_DMAs` subfolder of the `zip` file supplied with this EE-note[7].

Figure 16 Typical System Bandwidth Optimization Procedure

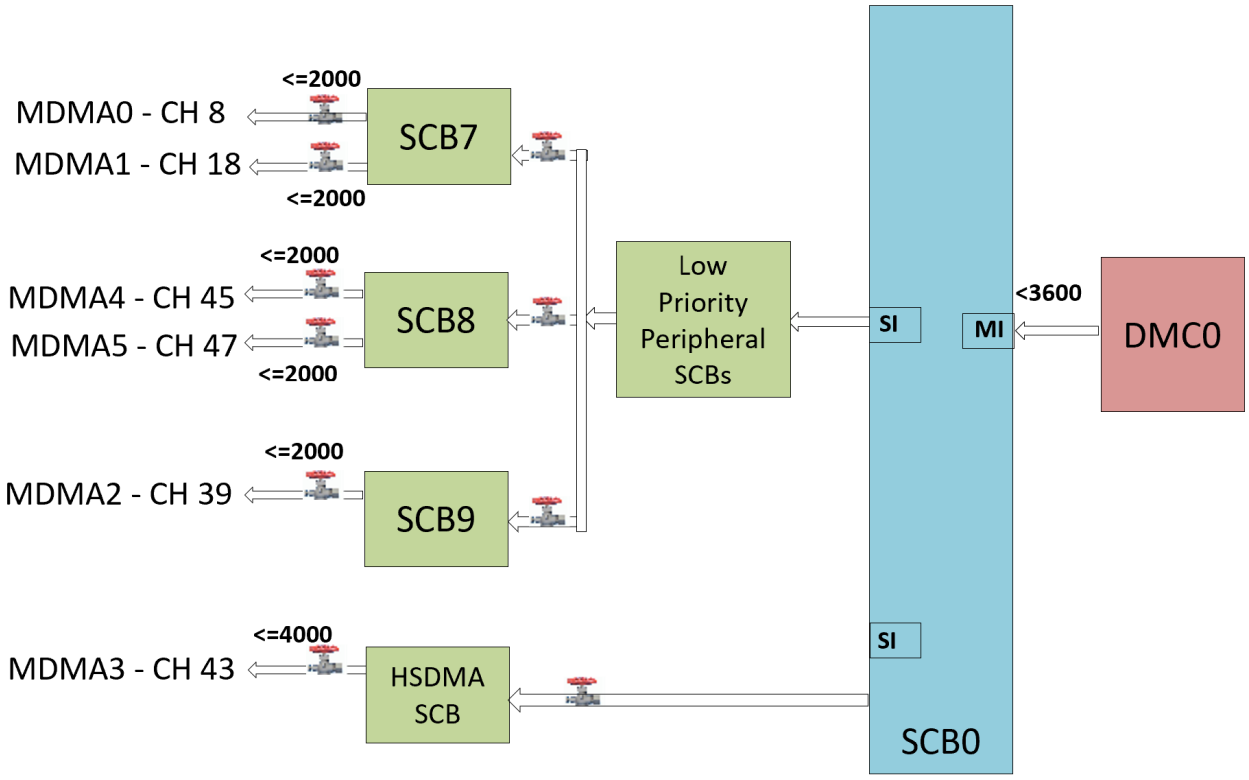


The procedures in Figure 16 generate the following throughput for the Figure 17 example application:

- CCLK = 1 GHz, DCLK = 900 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.
- MDMA4 channel 45, required throughput = 500 MHz * 4 <= 2000 MB/s.
- MDMA5 channel 47, required throughput = 500 MHz * 4 <= 2000 MB/s.

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

Figure 17 Example Application–DMC Throughput Distribution Across SCBs



As shown in Table 7 and Figure 17, the total required throughput from the DMC controller equals 14000 MB/s. Theoretically, with DCLK = 900 MHz and a bus width of 16 bits, the maximum possible throughput is only 3600 MB/s. For this reason, there is a possibility that one or more requesters may not get the required bandwidth. For requesters such as MDMA, this can result in decreased throughput.

Table 7 Example Application - System Bandwidth Optimization Steps on ADSP-2183x/SC83x Processor

S.No.	Condition	SCB7				SCB9		HSMDMA		SCB8				Total (X)	Measured (Y)
		MDMA0		MDMA1		MDMA2		MDMA3		MDMA4		MDMA5			
		Required	Measured	Required	Measured	Required	Measured	Required	Measured	Required	Measured	Required	Measured		
1	No optimization	2000	82.3	2000	82.3	2000	163.7	4000	240.4	2000	82.39	2000	82.37	14000	733.77
2	Optimization at slave-T1	2000	181.2	2000	181.2	2000	363.4	4000	468	2000	181.1	2000	181.1	14000	1556.3
3	Optimization at master-T2	200	198.9	100	99.6	200	199.1	300	300.2	150	150.2	150	99.6	1100	1047.9
4	Optimization at master-T3	150	150.2	200	198.7	100	99.6	200	198.7	200	198.8	200	199.8	1050	1045.18
5	Optimization at master-T4	100	99.6	75	74.9	300	300.8	400	398.7	100	99.6	75	74.9	1050	1048.8

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*) = 14000 MB/s
- Effective DMC throughput (*Y*) = 733.77 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 1556.3MB/sec. However, since *X* is still greater than *Y*, the measured throughput is still lower than expected.

Steps 3, 4 and 5

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", "Optimization at the master – T3" and "Optimization at the master – T4"* 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

Table 8 summarizes the optimization techniques discussed in this application note, while also listing a few additional tips for bandwidth optimization.

Table 8: System Optimization Techniques Checklist

	Optimization Tip Description
☐	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 channel `MSIZE` parameters to optimal values to maximize throughput and avoid any potential underflow/overflow conditions.
☐	When 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. When possible, emulate EMDMA non-sequential transfer modes with MDMA.
☐	Program the SCB RQOS and WQOS registers to allocate priorities to various controllers as per system requirements.
☐	Use optimization techniques at the SCB target end, such as: • Multiple L2/L1 sub-banks to avoid access conflicts • Instruction/data caches
☐	Maintain the optimum clock ratios across different clock domains.
☐	Because MMR latencies affect the interrupt service latency, ADSP-2183x/SC83x 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 controllers (trigger generators) to trigger targets (triggers receivers), thereby offloading processing from the core.

Note: For a detailed discussion on this topic, refer to application note *Utilizing the Trigger Routing Unit for System Level Synchronization (EE-360)*[5].

Although the EE-360 note was written for the ADSP-215xx processor, the concepts can also be used for the ADSP-2183x/SC83x processors.

References

[1] *ADSP-2183x/ADSP-SC58x SHARC-FX Processor Hardware Reference,* Rev 0.1, March 2024. Analog Devices, Inc. *https://www.analog.com/media/en/dsp-documentation/processor-manuals/adsp-2183x-adsp-sc83x-hrm.pdf*

[2] *ADSP-21834/21835/21836/21837/ADSP-SC834/SC835 High Performance SHARC-FX DSP Core With Arm-Based Connectivity Data Sheet*. Rev PrE, May 2024. Analog Devices, Inc.

[3] *Associated zip file for EE-412: ADSP-2156x SHARC+ Processors System Optimization.* Rev. 2, September 2019. Analog Devices, Inc.

[4] *Associated zip file for EE-461: ADSP-21568 SHARC+ Processors System Optimization*. Rev 2.0, June 2024. Analog Devices, Inc.

[5] *Utilizing the Trigger Routing Unit for System Level Synchronization (EE-360)*. Rev 1.0, October 2013. Analog Devices.

[6] *ADSP-SC59x SHARC+ Processor System Optimization Techniques* (EE-445). Rev 1.0, December 2022, Analog Devices, Inc.

[7] *Associated zip file for EE-464: ADSP-2183x/ADSP-SC83x SHARC-FX Processors System Optimization*. Rev 1.0, September 2024. Analog Devices, Inc.

Document History

Date	Author(s)	Description of EE-Note Changes
Sep 18, 2024	Tejaswi Chitneedi	Initial Release