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

ADSP-SC84x SHARC-FX Processor Optimization

Contributed by Tejaswi Chitneedi *Rev 1 – July 23, 2026*

Introduction

The ADSP-SC84x SHARC-FX® processor family provides an optimized architecture that supports high system bandwidth and advanced peripherals. This EE 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 EE note is the same as in *SHARC-FX Processor System Optimization (EE-464)*[1], which is specific for the ADSP-SC83x family of processors. This EE note includes figures, tables, and data specific to the ADSP-SC84x family of processors.

Customer Takeaways

This EE note provides the following customer information:

- Optimization techniques to improve ADSP-SC84x system throughput
- Identifies measured MMR latencies in core cycles
- Provides insight into different L1 and L2 memory throughput methods

ADSP-SC84x Processor Architecture

This section describes the ADSP-SC84x processor's key architectural features that play a crucial role in system bandwidth and performance. For detailed information, see the *ADSP-2184x/ADSP-SC84x SHARC-FX Processor Hardware Reference*[2].

The overall architecture of the ADSP-SC84x 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.

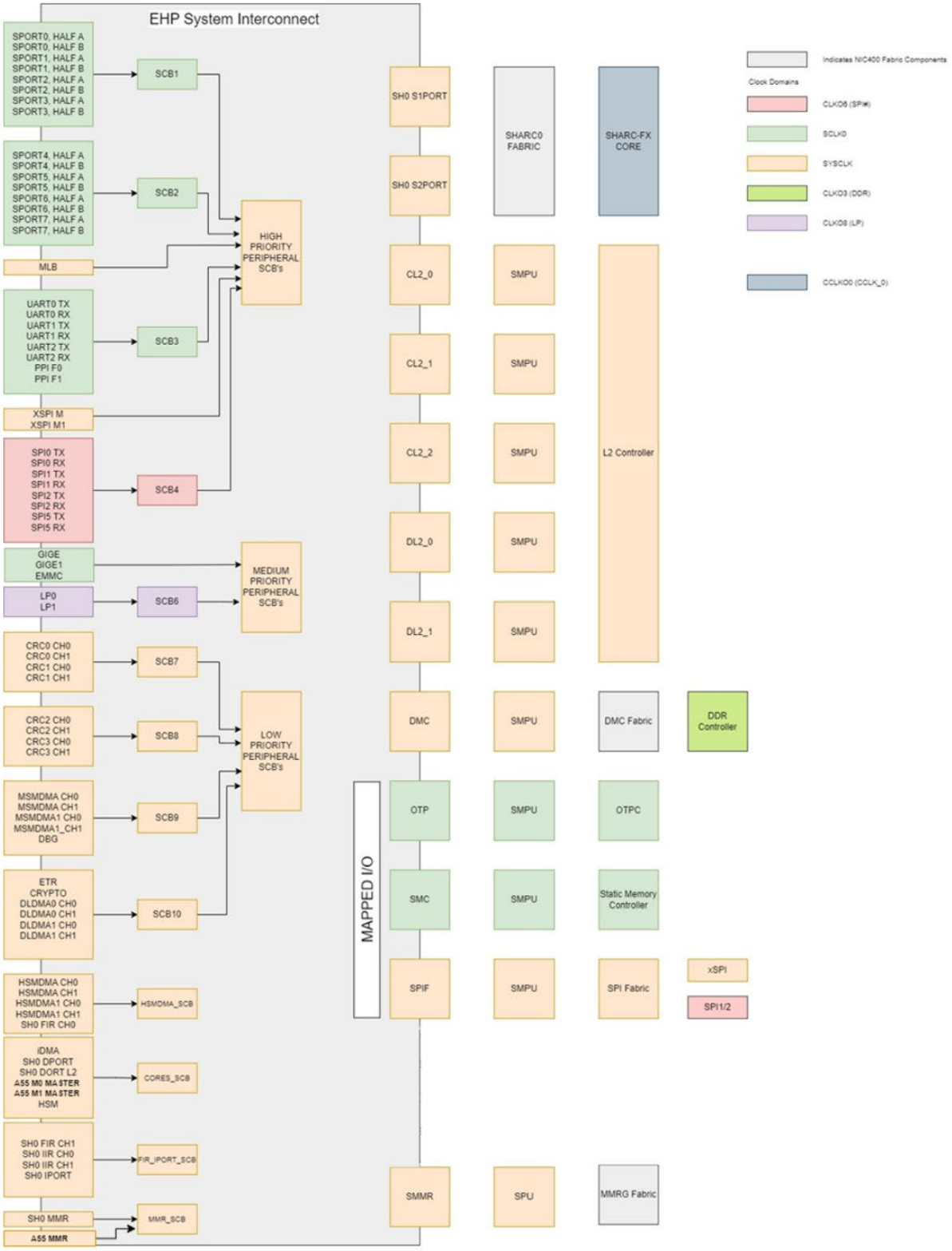


Figure 1: System Cross Bar (SCB) Block Diagram

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.

System Crossbars

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.

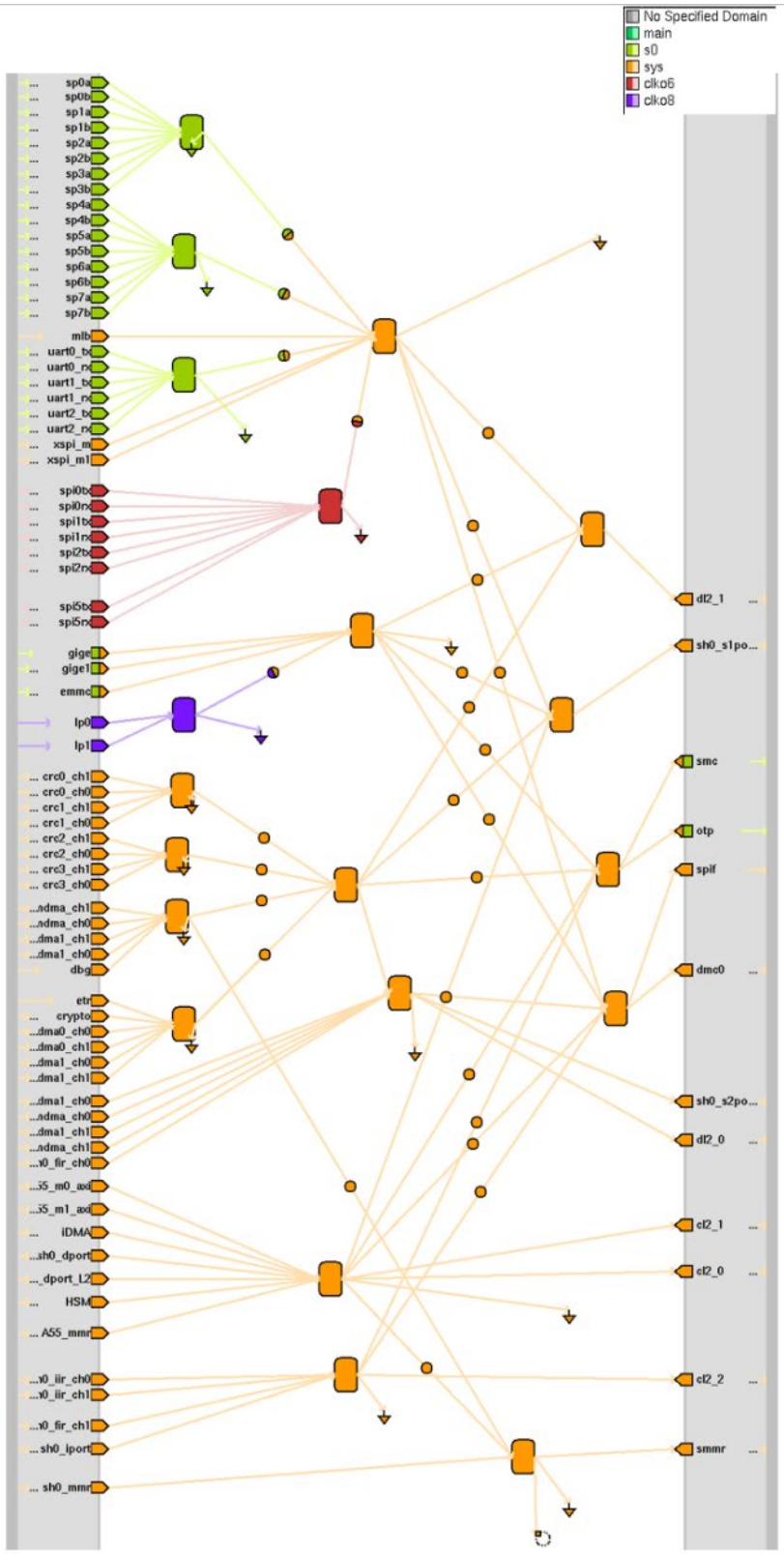


Figure 2: SCB Interconnections

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 crossbars. 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-SC84x processors, the memory and peripheral bus width 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.

Memory to Memory DMA (MDMA)

The ADSP-SC84x 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-SC84x 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-1200 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 show the various memory completers and the corresponding maximum theoretical bandwidth supported by the ADSP-SC84x processors.

Table 2: Memory Targets and Maximum Theoretical Bandwidth

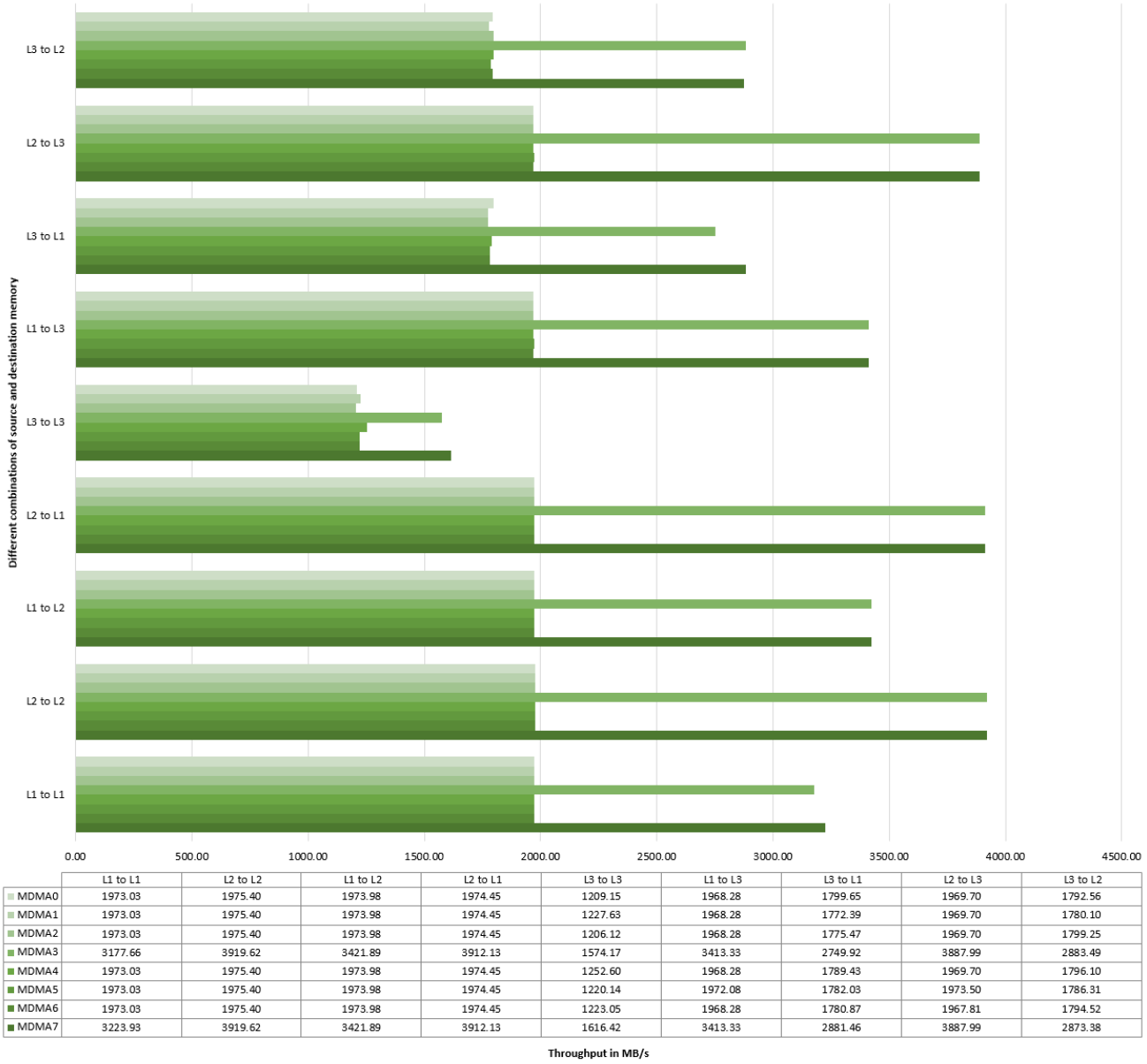
Memory Type (L1/L2/L3)	Maximum CCLK/SYSCLK/SCLKx/ Frequency (MHz)	Clock Domain	Bus Width (Bits)	Data Rate Clock Rate	Maximum Theoretical Bandwidth (MB/s)
L1	CCLK-1200	CCLK	32	1	4000
L2	SYSCLK-500	SYSCLK	128	1	8000
L3	SCLKx-125 DCLKx-1000	DCLK	32	2	8000

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 = 1.2 GHz
- SYSCLK = 500 MHz
- DCLK = 1000 MHz

The `Test1_MDMA_Throughput` code supplied with this EE note[3] 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	1973.03	1975.40	1973.98	1974.45	1209.15	1968.28	1799.65	1969.70	1792.56
MDMA1	1973.03	1975.40	1973.98	1974.45	1227.63	1968.28	1772.39	1969.70	1780.10
MDMA2	1973.03	1975.40	1973.98	1974.45	1206.12	1968.28	1775.47	1969.70	1799.25
MDMA3	3177.66	3919.62	3421.89	3912.13	1574.17	3413.33	2749.92	3887.99	2883.49
MDMA4	1973.03	1975.40	1973.98	1974.45	1252.60	1968.28	1789.43	1969.70	1796.10
MDMA5	1973.03	1975.40	1973.98	1974.45	1220.14	1972.08	1782.03	1973.50	1786.31
MDMA6	1973.03	1975.40	1973.98	1974.45	1223.05	1968.28	1780.87	1967.81	1794.52
MDMA7	3223.93	3919.62	3421.89	3912.13	1616.42	3413.33	2881.46	3887.99	2873.38

Figure 3: Measured MDMA Throughput on ADSP-SC84x Processor

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 must 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 2.41 to 4.65 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	3555	1361	117	2.41
2	0x28240000	0x282B0001	256	1	3555	1358	117	2.41
3	0x28240001	0x282B0000	1024	1	12762	3213	116	3.83
4	0x28240000	0x282B0001	1024	1	12775	3198	118	3.85
5	0x28240001	0x282B0000	4096	1	49635	10579	119	4.64
6	0x28240000	0x282B0001	4096	1	49635	10559	121	4.65

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.

```
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 = 694.237312 MB/s

Test combination 3 passed
```

Figure 4: MDMA Bandwidth Limit Results

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.

```
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
```

Figure 5: MDMA Bandwidth Monitor Results

Extended Memory DMA (EMDMA)

The ADSP-SC84x processor also supports extended memory DMA (EMDMA). The EMDMA engine is used to transfer 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-2184x/ADSP-SC84x SHARC-FX Processor Hardware Reference*[2]. 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 = 1.2 GHz and SYSCLK = 500 MHz and DCLK = 1000 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	358.15	629.26	358.37	574.88	487.59	358.30	498.40	529.23	500.00
EMDMA0_Read	358.38	629.28	574.90	358.43	487.49	499.80	358.31	497.74	529.11
EMDMA1_Write	358.35	629.33	358.44	574.96	483.16	358.39	502.61	529.30	498.69
EMDMA1_Read	358.42	629.19	575.00	358.40	486.66	500.81	358.36	501.30	529.27

Figure 6: Measured EMDMA Processor Throughput

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` supplied with this EE note[3], 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	31523	29311	6360
Read	35411	23910	8157

Understanding the System Crossbars

For more information on system crossbars, refer to the System Crossbars (SCB) chapter in the *ADSP-2184x/ADSP-SC84x SHARC-FX Processor Hardware Reference*[2].

Understanding the System Targets

Memory Hierarchy

As shown in Table 2, the ADSP-SC84x 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

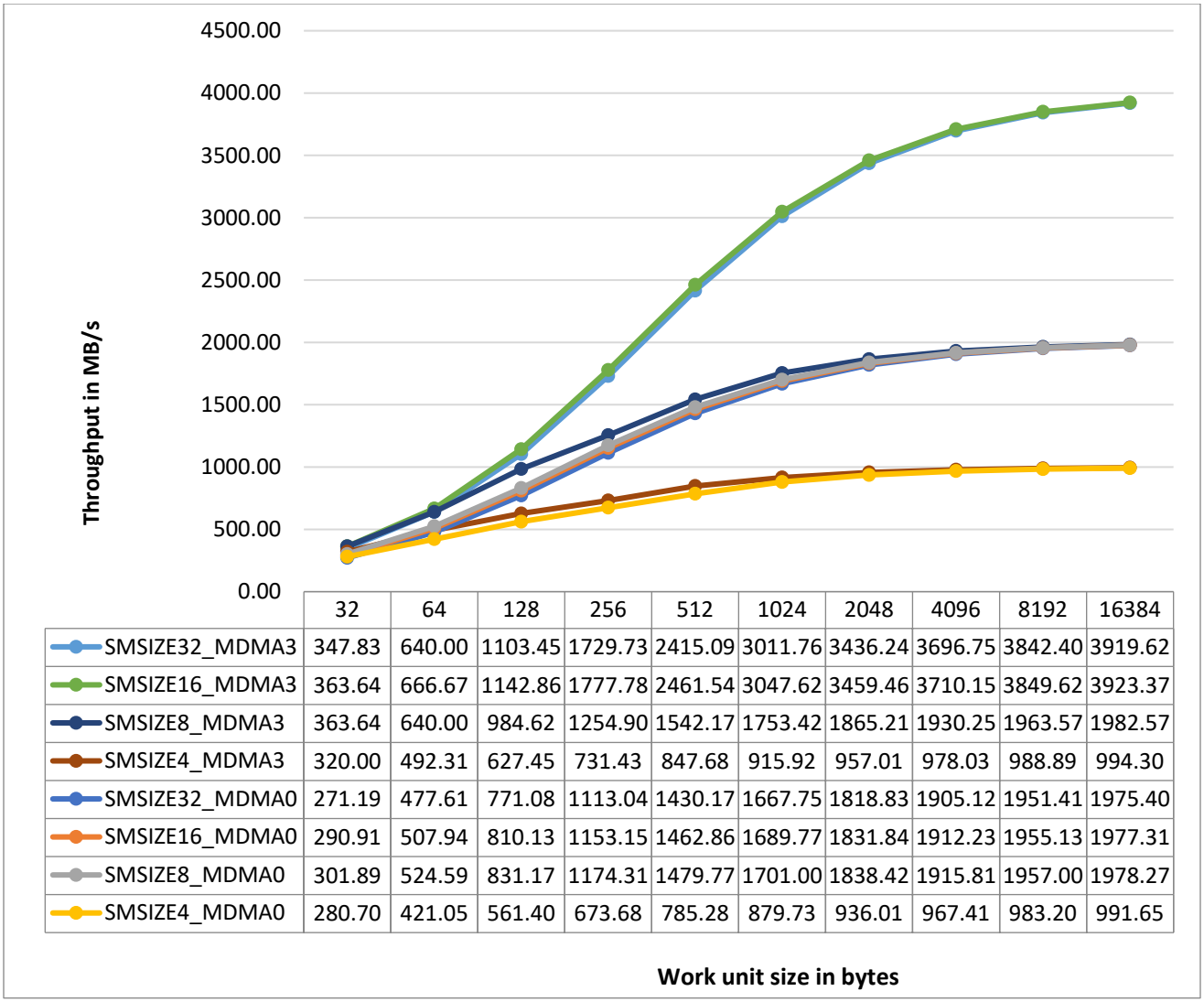
L1 memory runs at CCLK and is the fastest accessible memory in the hierarchy. From a programming perspective, when accessing the L1 memory of the SHARC-FX processer, use a multiprocessor memory offset of `0x28000000` for all DMA accesses.

The maximum theoretical throughput of L1 memory (for system/DMA accesses) is 1000 × 4 = 4000 MB/s for 1.2 GHz CCLK operation. As shown in Figure 3, the maximum measured L1 throughput using MDMA3 is approximately ~3919 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 almost 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-2184x/ADSP-SC84x SHARC-FX Processor Hardware Reference*[2].
- 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 = 1.2 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.



	32	64	128	256	512	1024	2048	4096	8192	16384
SMSIZE32_MDMA3	347.83	640.00	1103.45	1729.73	2415.09	3011.76	3436.24	3696.75	3842.40	3919.62
SMSIZE16_MDMA3	363.64	666.67	1142.86	1777.78	2461.54	3047.62	3459.46	3710.15	3849.62	3923.37
SMSIZE8_MDMA3	363.64	640.00	984.62	1254.90	1542.17	1753.42	1865.21	1930.25	1963.57	1982.57
SMSIZE4_MDMA3	320.00	492.31	627.45	731.43	847.68	915.92	957.01	978.03	988.89	994.30
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

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

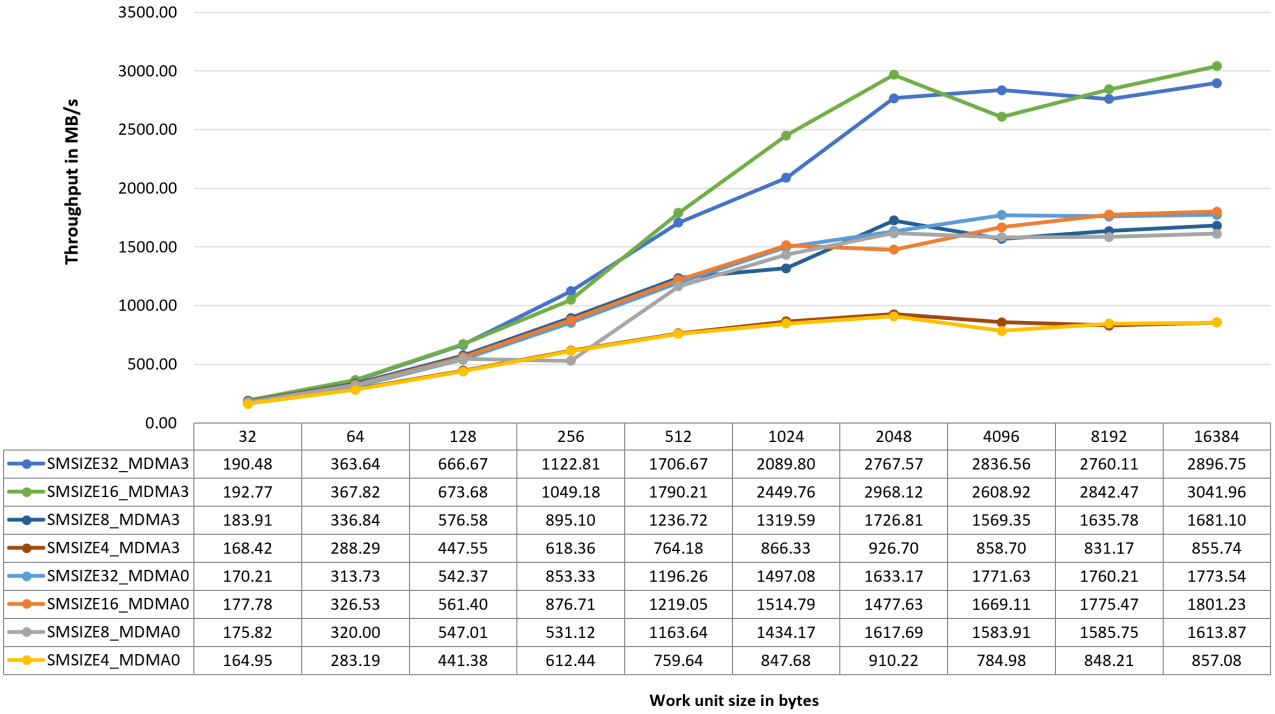
L3 Memory and External Memory Throughput

The ADSP-SC84x processors provide interfaces for connecting to DDR4 memory devices. The DMC interface operates at speeds of up to 1000 MHz. For the 32-bit DDR4 interface, the maximum theoretical throughput that the DMC can deliver equals 8000 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.

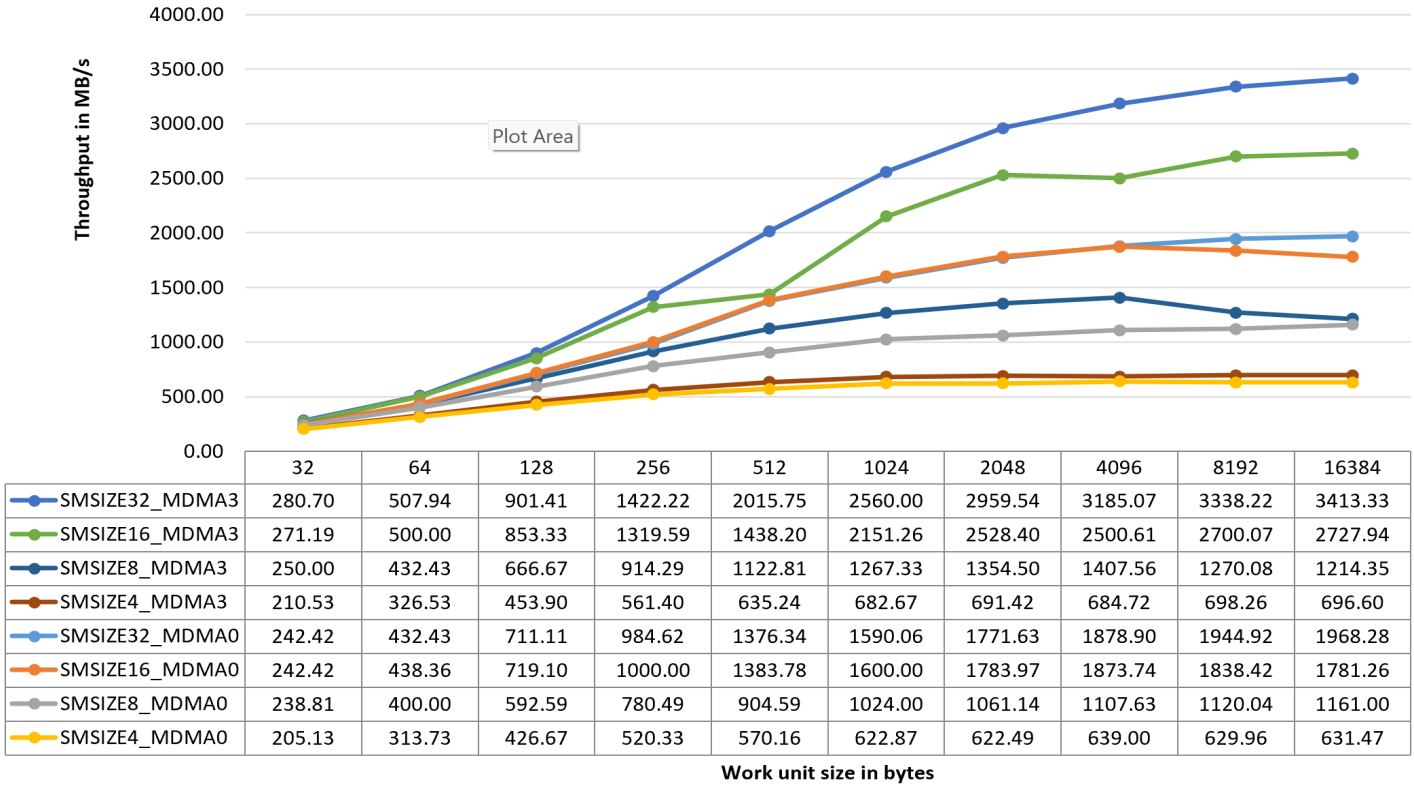
The MDMA3 stream (HSMDMA) can request DMC accesses faster than any other requesters (for example, 4000 MB/s). The practical DMC throughput possible using MDMA depends upon factors such as whether the accesses are sequential or non-sequential, the block size of the transfer, and DMA

parameters (for example, `MSIZE`). Figure 8 and Figure 9 provide the DMC measured throughput at CCLK = 1.2 GHz and DCLK = 1000 MHz using MDMA0 (channels 8 and 9) and MDMA3 (channels 43 and 44) streams for various `MSIZE` values and buffer sizes.



	32	64	128	256	512	1024	2048	4096	8192	16384
SMSIZE32_MDMA3	190.48	363.64	666.67	1122.81	1706.67	2089.80	2767.57	2836.56	2760.11	2896.75
SMSIZE16_MDMA3	192.77	367.82	673.68	1049.18	1790.21	2449.76	2968.12	2608.92	2842.47	3041.96
SMSIZE8_MDMA3	183.91	336.84	576.58	895.10	1236.72	1319.59	1726.81	1569.35	1635.78	1681.10
SMSIZE4_MDMA3	168.42	288.29	447.55	618.36	764.18	866.33	926.70	858.70	831.17	855.74
SMSIZE32_MDMA0	170.21	313.73	542.37	853.33	1196.26	1497.08	1633.17	1771.63	1760.21	1773.54
SMSIZE16_MDMA0	177.78	326.53	561.40	876.71	1219.05	1514.79	1477.63	1669.11	1775.47	1801.23
SMSIZE8_MDMA0	175.82	320.00	547.01	531.12	1163.64	1434.17	1617.69	1583.91	1585.75	1613.87
SMSIZE4_MDMA0	164.95	283.19	441.38	612.44	759.64	847.68	910.22	784.98	848.21	857.08

Figure 8: DMC Measured Throughput for Sequential MDMA Reads



	32	64	128	256	512	1024	2048	4096	8192	16384
SMSIZE32_MDMA3	280.70	507.94	901.41	1422.22	2015.75	2560.00	2959.54	3185.07	3338.22	3413.33
SMSIZE16_MDMA3	271.19	500.00	853.33	1319.59	1438.20	2151.26	2528.40	2500.61	2700.07	2727.94
SMSIZE8_MDMA3	250.00	432.43	666.67	914.29	1122.81	1267.33	1354.50	1407.56	1270.08	1214.35
SMSIZE4_MDMA3	210.53	326.53	453.90	561.40	635.24	682.67	691.42	684.72	698.26	696.60
SMSIZE32_MDMA0	242.42	432.43	711.11	984.62	1376.34	1590.06	1771.63	1878.90	1944.92	1968.28
SMSIZE16_MDMA0	242.42	438.36	719.10	1000.00	1383.78	1600.00	1783.97	1873.74	1838.42	1781.26
SMSIZE8_MDMA0	238.81	400.00	592.59	780.49	904.59	1024.00	1061.14	1107.63	1120.04	1161.00
SMSIZE4_MDMA0	205.13	313.73	426.67	520.33	570.16	622.87	622.49	639.00	629.96	631.47

Figure 9: DMC Measured Throughput for Sequential MDMA Writes

The following important observations can be made for the ADSP-SC84xprocessor:

- The throughput trends are similar for reads and writes with regards to `MSIZE` and buffer size values.
- The peak measured read throughput is 3041.96 MB/s for MDMA3 with `MSIZE` = 16 bytes. The peak measured write throughput is 3413.33 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 190.48 MB/s, whereas it reaches 2896.75 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 the 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 2896.75 MB/s for `MSIZE` = 32 bytes, while it reduces significantly to 855.74 MB/s for `MSIZE` = 4 bytes.

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, and so on. 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 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, if 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 (2D) descriptors 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, refer to *Xtensa Microprocessor Data Book and Xtensa System Software Manual*.

Figure 10 through Figure 13 show 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 EE note[3] can be used to measure IDMA throughput.

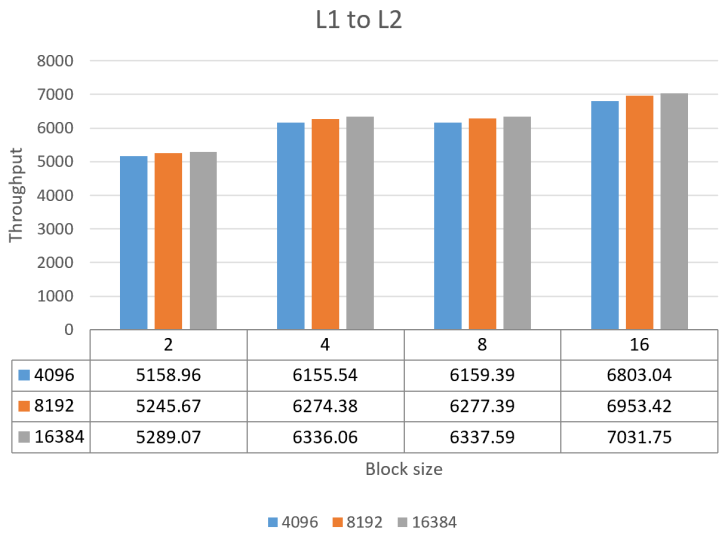


Figure 10: Measured IDMA Throughput for Transfer Between L1-L2 on the ADSP-SC84x Processor

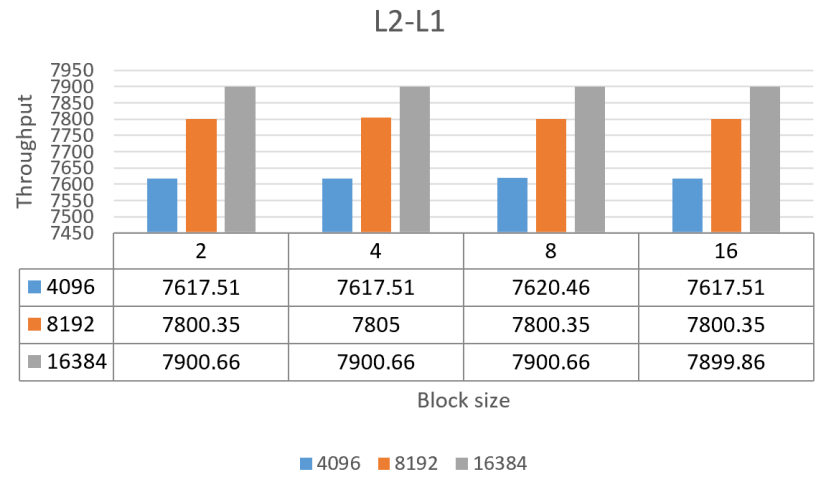


Figure 11: Measured IDMA Throughput for Transfer Between L2-L1 on the ADSP-SC84x Processor

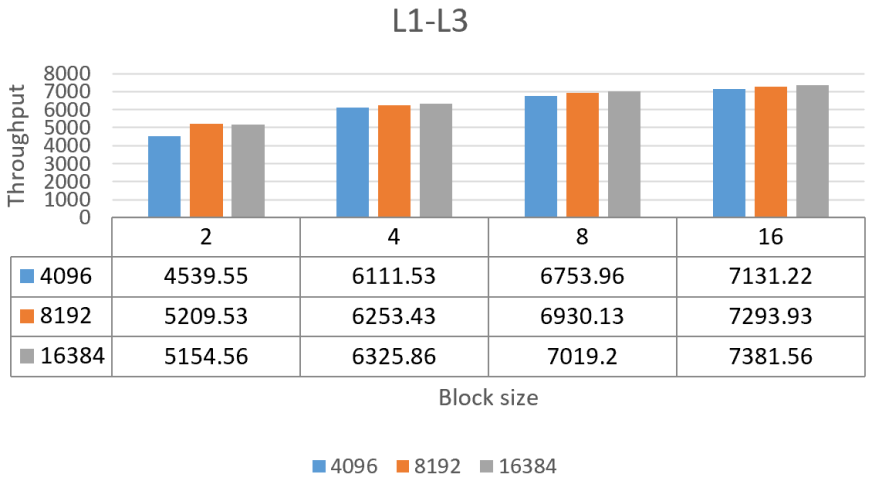


Figure 12: Measured IDMA Throughput for Transfer Between L1-L3 on the ADSP-SC84x Processor

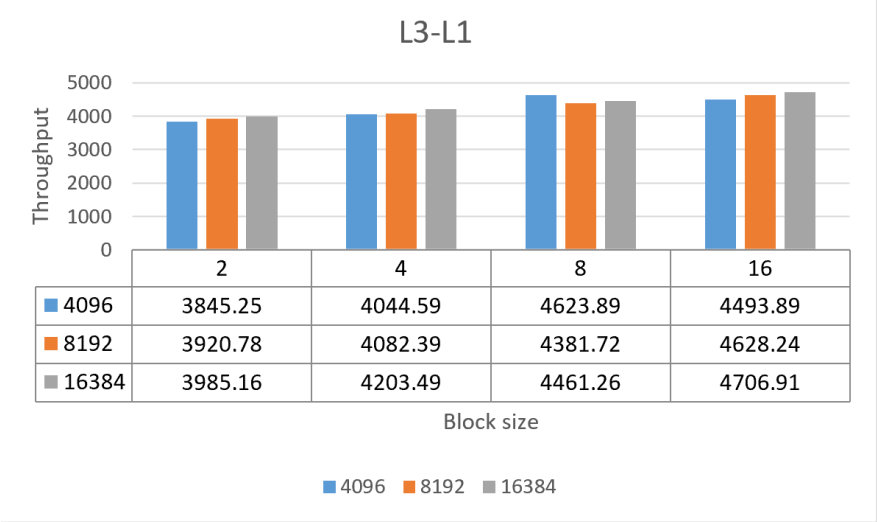


Figure 13: Measured IDMA Throughput for Transfer Between L3-L1 on the ADSP-SC84x Processor

System MMR Latencies

Table 5 shows the measured MMR latency in core cycles for different peripherals on the ADSP-SC84x processor. The measurement was taken with CCLK = 1.2 GHz, 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 5: MMR Access Latency Processors in Approximate CCLK Cycles

Register	Peripheral	Register Name	Write Cycles	Read Cycles
0	LP0	pREG_LP0_DIV	184	184
1	TWI0	pREG_TWI0_CLKDIV	119	119
2	TWI1	pREG_TWI1_CLKDIV	119	119
3	SPORT0	pREG_SPORT0_DIV_A	100	100
4	SPORT1	pREG_SPORT1_DIV_A	104	104
5	UART0	pREG_UART0_CLK	104	104
6	UART1	pREG_UART1_CLK	104	104
7	PORTA	pREG_PORTA_DATA_SET	104	104
8	PORTB	pREG_PORTB_DATA_SET	103	103
9	PINT1	pREG_PINT1_ASSIGN	104	104
10	SWU1	pREG_SWU1_LA0	68	68
11	SWU2	pREG_SWU2_LA0	67	67
12	HADC0	pREG_HADC0_CHAN_MSK	110	110
13	TMU0	pREG_TMU0_FLT_LIM_HI	111	111
14	TIMER0	pREG_TIMER0_TMR0_WID	101	101
15	OTPC0	pREG_OTPC0_CTL	216	216
16	SPI0	pREG_SPI0_CLK	100	100
17	SPI1	pREG_SPI1_CLK	100	100
18	LPDDR4	pREG_LPDDR4_CTLR	86	86
19	L2CTL0	pREG_L2CTL0_CTL	65	65
20	SEC0	pREG_SEC0_RAISE	576	576
21	TRU0	pREG_TRU0_RSR0	65	65
22	SPU0	pREG_SPU0_SECUREP10	65	65
23	RCU0	pREG_RCU0_MSG	64	64
24	CGU0	pREG_CGU0_CTL	65	65
25	CDU0	pREG_CDU0_CFG0	65	65
26	DPM0	pREG_DPM0_PER_DIS0	65	65
27	MLB0	pREG_MLB0_MDAT0	63	63
28	MEC0	pREG_MEC0_PERR_IMASK0	62	62
29	CRC0	pREG_CRC0_DCNT	62	62
30	CRC1	pREG_CRC1_DCNT	62	62
31	FIR0	pREG_FIR0_INIDX	38	38
32	IIR0	pREG_IIR0_INIDX	38	38

Register	Peripheral	Register Name	Write Cycles	Read Cycles
33	SPDIF0	pREG_SPDIF0_TX_UBUFF_A0	110	110
34	SPDIF1	pREG_SPDIF1_TX_UBUFF_A0	110	110
35	PCG0	pREG_PCG0_PW1	110	110
36	DAI0	pREG_DAI0_IMSK_FE	110	110
37	DAI1	pREG_DAI1_IMSK_FE	110	110
38	ASRC0	pREG_ASRC0_MUTE	111	111
39	ASRC1	pREG_ASRC1_MUTE	120	120
40	PKTE0	pREG_PKTE0_SA_ADDR	67	67
41	PKA0	pREG_PKA0_APTR	71	71
42	PKIC0	pREG_PKIC0_ACK	72	72
43	EMDMA0	pREG_EMDMA0_INDX1	63	63
44	EMDMA1	pREG_EMDMA1_INDX1	62	62
45	TAPC	pREG_TAPC_SDBGKEY0	62	62

The MMR latency numbers are measured with the *"sync"* instruction after the write. This ensures 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 is shorter. Similarly, the system latency incurred by a read followed by a write, or vice versa, is 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 14 provides a flow chart of typical steps used in a system bandwidth optimization procedure for ADSP-SC84x processor-based applications.

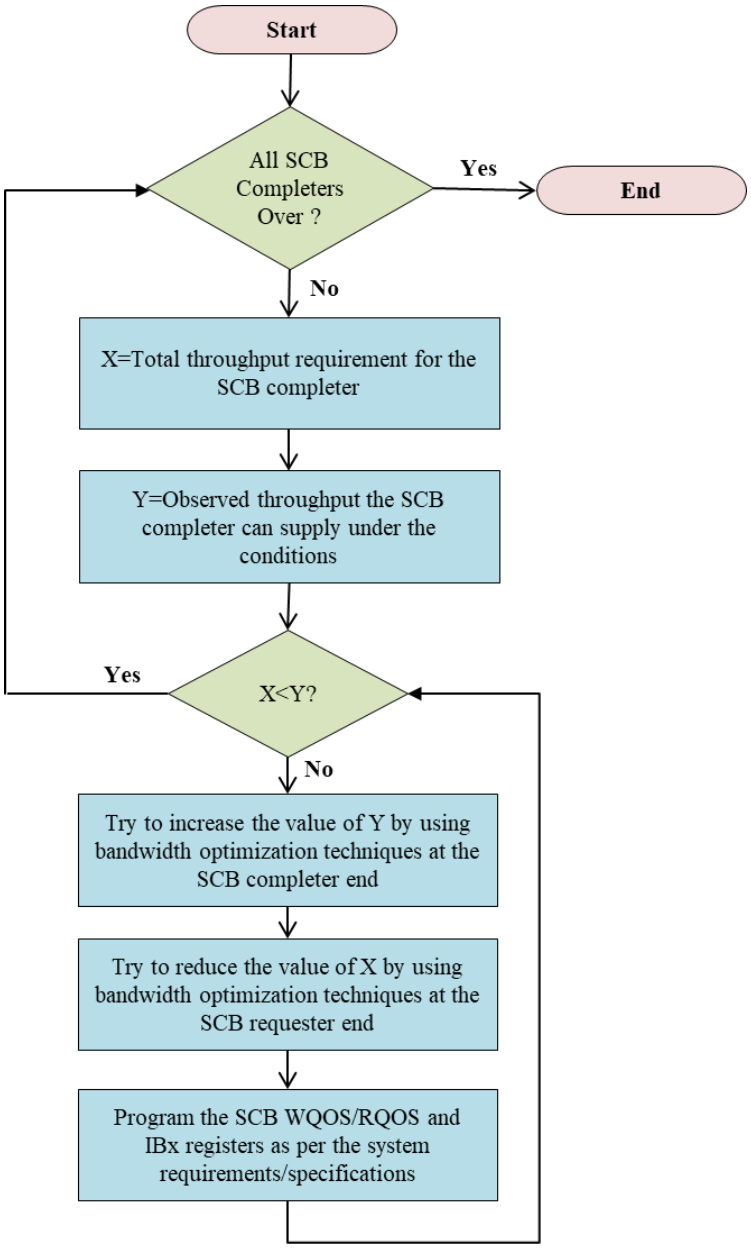


Figure 14: Typical System Bandwidth Optimization Procedure

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 15 shows a block diagram of the example application code found in the `Bandwidth_optimization` subfolder of the `Codes_appnote` folder of the `zip file` supplied with this EE-note[3].

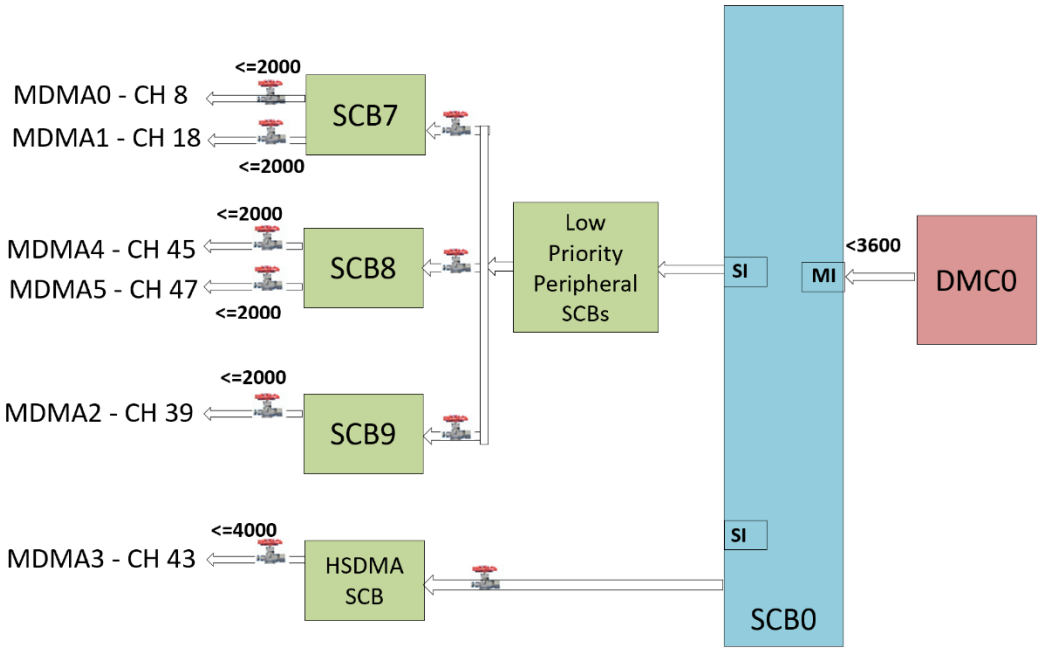


Figure 15: Example Application: DMC Throughput Distribution Across SCBs

The procedures in Figure 14 generate the following throughput for the Figure 15 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.

As shown in Figure 15 and Table 6, the total required throughput from the DMC controller equals 14000 MB/s. Theoretically, with `DCLK` = 1000 MHz and a bus width of 32 bits, the maximum possible throughput is only 8000 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 6: Example Application: System Bandwidth Optimization Steps on the ADSP-SC84x Processor

S. No.	Condition	SCB7				SCB9		HSDMA SCB		SCB8				Total throughput (X)	Measured throughput (Y)
		MDMA0		MDMA1		MDMA2		MDMA3		MDMA4		MDMA5			
		Required	Measured	Required	Measured	Required	Measured	Required	Measured	Required	Measured	Required	Measured		
1	No optimization	2000	105.96	2000	105.90	2000	209.91	4000	324.19	2000	105.92	2000	105.90	14000.00	957.78
2	Optimization at the target - T1	2000	209.51	2000	209.35	2000	416.11	4000	642.61	2000	209.37	2000	209.34	14000.00	1896.29
3	Optimization at the controller - T2	200	164.04	100	82.77	200	164.55	300	246.47	150	124.40	100	82.75	1050.00	865.00
4	Optimization at the controller - T3	150	124.30	200	163.31	100	82.74	200	164.34	200	163.20	200	163.22	1050.00	861.11
5	Optimization at the controller - T4	100	82.74	75	62.20	300	247.57	400	325.88	100	82.70	75	62.26	1050.00	863.35

Table 6 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 DDR4 SDRAM bank. The row corresponding to "No optimization" in Table 6 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) = 957.81 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 DDR4 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 DDR4 SDRAM banks. This configuration allows parallel accesses to multiple pages of different banks, helping improve *Y*. *"Optimization at the target - T1"* in Table 6 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 1896.31 MB/sec. However, since *X* is still greater than *Y*, the measured throughput is still lower than expected.

Step 3, 4, and 5

There is not much additional room to significantly increase the value of *Y*. Alternatively, optimization techniques can be employed at the controller 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 controller - T2", "Optimization at the controller – T3", and "Optimization at the controller – T4"* in Table 6 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 7 summarizes the optimization techniques discussed in this EE note, while also listing a few additional tips for bandwidth optimization.

Table 7: 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.

	Optimization Tip Description
☐	Because MMR latencies affect the interrupt service latency, ADSP-SC84x processors offer the trigger routing unit (TRU)[1] for bandwidth optimization and system synchronization. The TRU allows 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.

References

[1] *SHARC-FX Processor System Optimization (EE-464), ADSP-SC83x. Analog Devices, Inc.*

[2] *ADSP-2184x/ADSP-SC84x SHARC-FX Processor Hardware Reference. Analog Devices, Inc.*

[3] *Associated zip file for EE-485: ADSP-SC84x SHARC-FX Processor Optimization. Analog Devices, Inc.*

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

[5] *ADSP-21844/ADSP-21846/ADSP-SC844/ADSP-SC846 High Performance SHARC-FX DSP Core With Arm-Based Connectivity/Security Data Sheet. Analog Devices, Inc.*

Document History

Revision	Description
Rev 1 – July 23, 2026 *by Tejaswi Chitneedi*	Initial Release

[1] For a detailed discussion on this topic, see *Utilizing the Trigger Routing Unit for System Level Synchronization (EE-360)* [4]. Although EE-360 was written for the ADSP-215xx processor, the concepts can also be used for the ADSP-2184x/SC84x processors.