



Technical notes on using Analog Devices products and development tools. Visit our Web resources [EE Application Notes](#) and [Processors and DSP](#) or e-mail processor.support@analog.com and processor.tools.support@analog.com for customer technical support.

SHARC+ Processor System Optimization

Submitted by: Tejaswi Chitneedi

Revision 2.0 – June 2024

Summary

The ADSP-21568 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, plus various available bandwidth optimization techniques.

Applicable Processors

Most of the theoretical content of this application note is the same as in [ADSP-SC5xx/215xx SHARC+ Processor System Optimization Techniques](#) (EE-401)^[6]. This EE-461 application note includes the figures, tables, and data specific to the ADSP-21568 processor.

Customer Takeaways

This application note provides the following customer information:

- Optimization techniques to improve ADSP-21568 system throughput
- Identifies measured MMR latencies in core cycles
- Provides insight into different L1 and L2 memory throughput methods

ADSP-21568 Processor Architecture

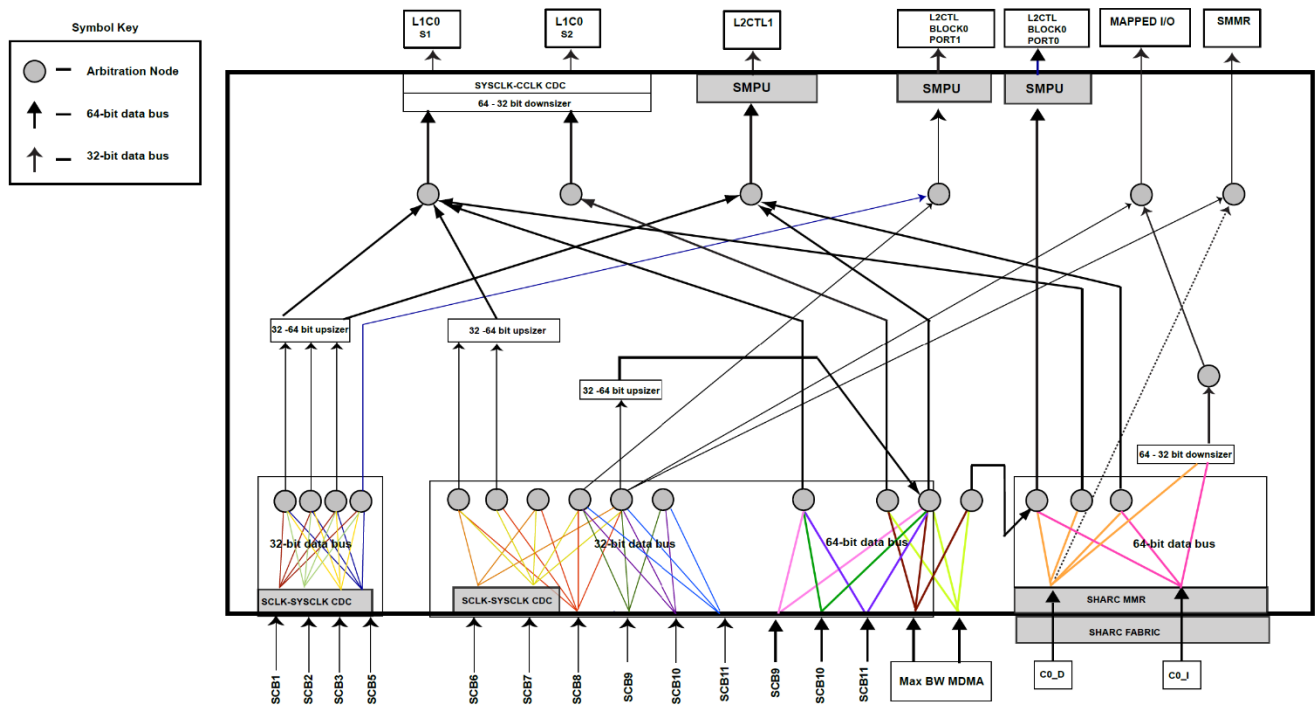
This section describes the ADSP-21568 processor key architectural features that play a crucial role in system bandwidth and performance. For detailed information, refer to the [ADSP-21568 SHARC+ Processor Hardware Reference](#)^[1].

The overall architecture of the ADSP-21568 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](#) (top), system bus targets (S) 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



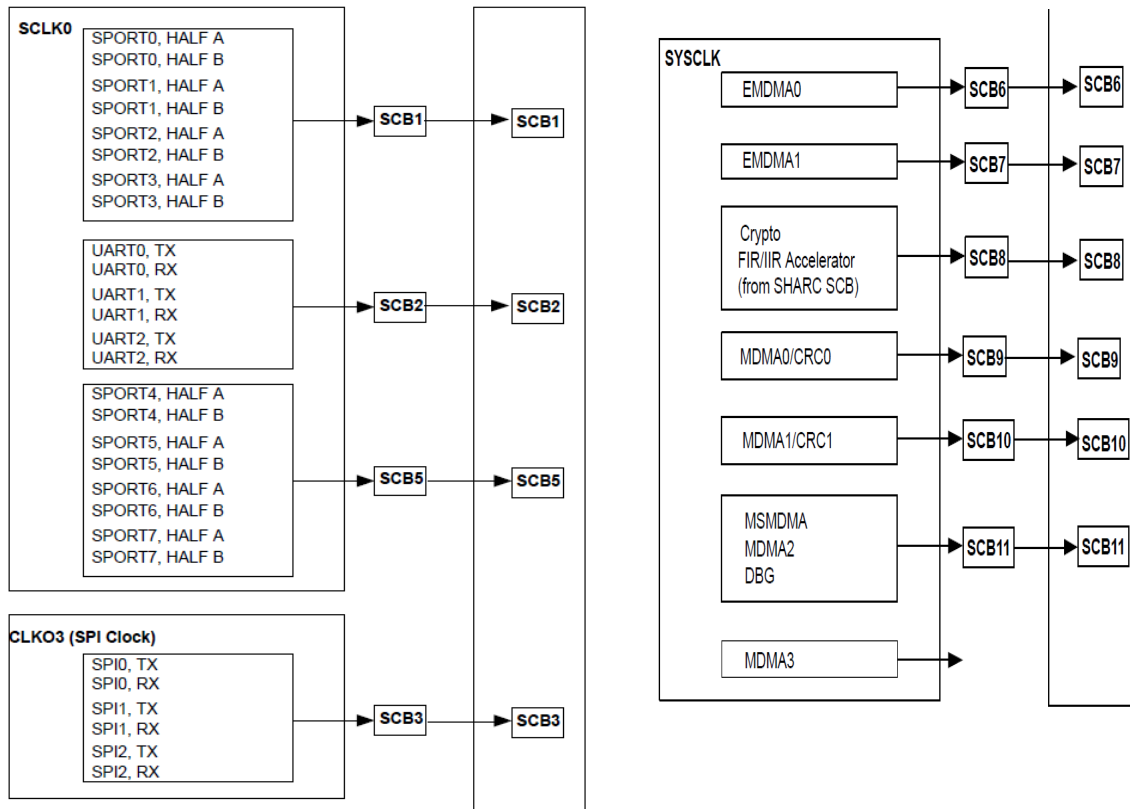
System Bus Controllers

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) that introduce their own latencies to the system.

Figure 2: SCB Controllers Groups



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 target (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 ADS-21568 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.

SHARC+ Processor System Optimization

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. When the MSIZE value is greater than `DMA_STAT.MBWID`, the SCB performs burst transfers to transfer the data equal to the MSIZE value.

Note: It is important to choose the appropriate MSIZE value, both from a functionality and a performance perspective.

When choosing the MSIZE value, consider the following needs:

- The start address of the work unit must align to the MSIZE value. When the start address does not align, it 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 target (memory), which is the most efficient for typical memory designs.

Memory to Memory DMA (MDMA)

The processor supports multiple MDMA streams (MDMA0/1/2/3) to transfer data from one memory to another (L1/L2/memory-mapped peripherals (for example, 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-21568 processor.

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-933.6 SYSCLK-466.8 SCLKx-116.7 | 8 | 9 | SYSCLK | 32 | 1867.2 |
| 1 | Enhanced Bandwidth or Medium Speed MDMA (MSMDMA) | | 18 | 19 | | 32 | 1867.2 |
| 2 | Enhanced Bandwidth or Medium Speed MDMA (MSMDMA) | | 39 | 40 | | 32 | 1867.2 |
| 3 | Maximum Bandwidth or High Speed MDMA (HSMDMA) | | 43 | 44 | | 64 | 3734.4 |

[Table 2](#) shows the various memory targets and the corresponding maximum theoretical bandwidth supported by the ADSP-21568 processor.

SHARC+ Processor System Optimization

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-933.6 | CCLK | 32 | 1 | 3734.4 |
| L2 | SYSCLK-466.8 SCLKx-116.7 | SYSCLK | 64 | 1 | 3734.4 |

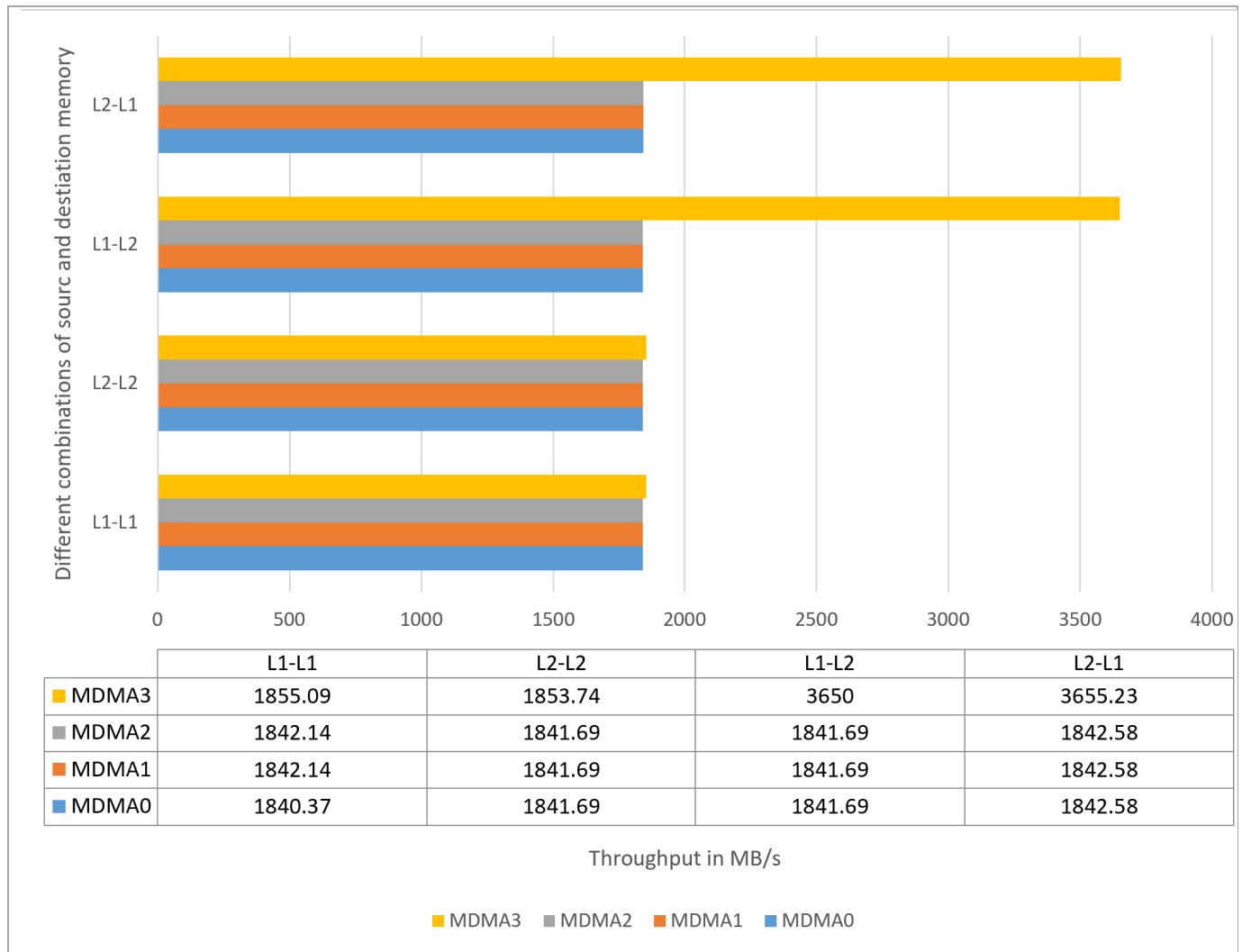
The actual (measured) MDMA throughput is always less than or equal to the minimum of the maximum theoretical throughput supported by one of the three: MDMA, source memory, or destination memory. For example, the measured throughput of MDMA0 between L1 and L2 is less than or equal to 1867.2 MB/s, which 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 = 933.6 MHz
- SYSCLK = 466.8 MHz

The code `MDMA_Throughput` supplied with [Source Files for EE-412: ADSP-2156x SHARC+ Processors System Optimization Techniques](#)^[5] can be used to measure MDMA throughput.

Figure 3: Measured MDMA Throughput on ADSP-21568 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 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 1.4 to 2.6 times better than `adi_mdma_Copy1D` for non-32-byte aligned start addresses. The example code `MDMA_1DAuto` (see [Source Files for EE-461: ADSP-2156x SHARC+ Processors System Optimization Techniques](#)^[4]) can be used to measure MDMA performance for both APIs for a given use case.

SHARC+ Processor System Optimization

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 | 0x2C0001 | 0x300000 | 256 | 1 | 2024 | 1362 | 524 | 1.07 |
| 2 | 0x2C0000 | 0x300001 | 256 | 1 | 2024 | 1254 | 520 | 1.14 |
| 3 | 0x2C0001 | 0x300000 | 1024 | 1 | 6248 | 2898 | 510 | 1.83 |
| 4 | 0x2C0000 | 0x300001 | 1024 | 1 | 6248 | 2790 | 522 | 1.89 |
| 5 | 0x2C0001 | 0x300000 | 4096 | 1 | 23144 | 9042 | 510 | 2.42 |
| 6 | 0x2C0000 | 0x300001 | 4096 | 1 | 23144 | 8934 | 522 | 2.45 |

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:

$$[1] \quad \text{Bandwidth} = \min(\text{SYSCLK frequency in MHz} * \text{DMA bus width in bytes}, \text{SYSCLK frequency in MHz} * \text{MSIZE in bytes} / \text{DMA_BWLCNT})$$

The API `adi_mdma_BWLimit` (see [Source Files for EE-461: ADSP-2156x SHARC+ Processors System Optimization Techniques](#)^[4]) 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 = 392.731525 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 = 298.564275 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 = 677.747200 MB/s

Test combination 3 passed
    
```

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 (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 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` (see [Source Files for EE-461: ADSP-2156x SHARC+ Processors System Optimization Techniques](#)^[4]) 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

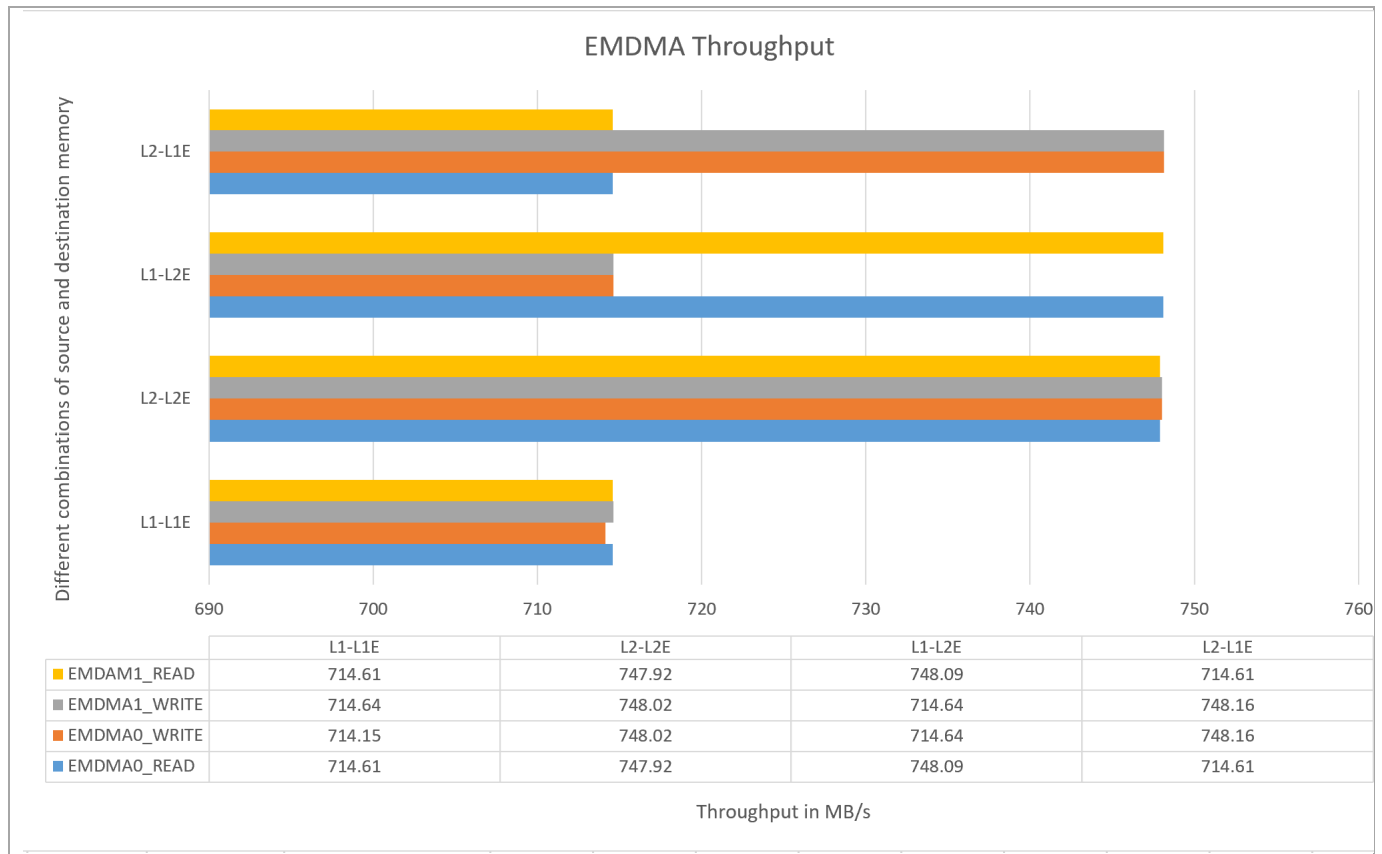
    Testing combination 2
    MDMA Stream no = 1
    MSIZE value = 32
    Source channel = 1
    Target BW = 300.000000 MB/s
    BW Monitor Expired!! Test combination 2 passed

        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
```

Extended Memory DMA (EMDMA)

The ADSP-21568 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-21568 SHARC+ 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 = 933.6 MHz and SYSCLK = 466.8 MHz.

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 `MDMA_Circular_Buffer` (see [Source Files for EE-461: ADSP-2156x SHARC+ Processors System Optimization Techniques](#)^[4]) 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 L2 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)

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.

SHARC+ Processor System Optimization

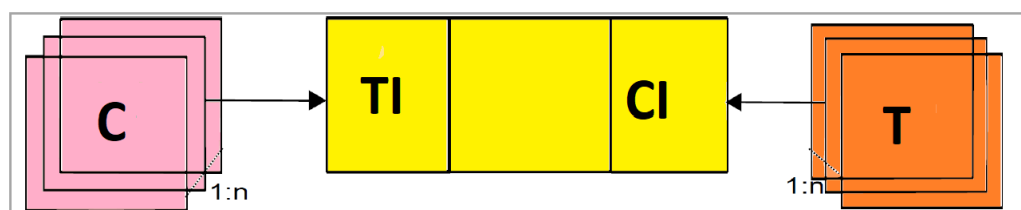
Table 4: MDMA Emulated Circular Buffer vs. EMDMA

| Write/Read | Core Cycles | | |
|------------|-------------|--------------------------|---------------------------|
| | EMDMA | MDMA3 MSIZE = 4 bytes | MDMA3 MSIZE = 32 bytes |
| Write | 23312 | 21212 | 5524 |
| Read | 23019 | 21189 | 5491 |

Understanding the System Crossbars

As shown in [Figure 2: SCB Controllers Groups](#) on page 3, 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 Controllers (M) to the System Bus Targets (T) by using a Target Interface (TI) and Controller Interface (CI). 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 target interface of the crossbar (where controllers such as DDE are connected) perform two functions, arbitration and clock domain conversion.

Arbitration

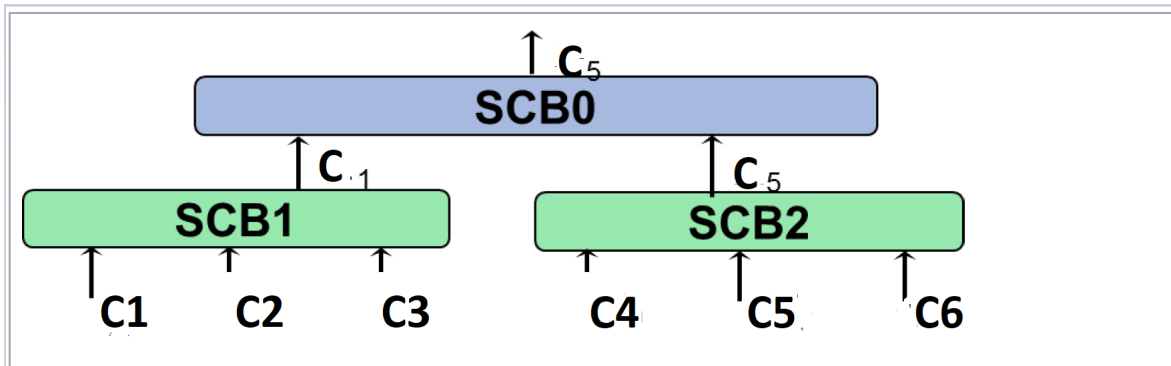
The programmable Quality of Service (QoS) registers are associated with SCBx. For example, the programmable QoS registers for SPORT0-3 and MDMA0 can be viewed as residing in SCB1. Whenever a transaction is received at SPORT0 half A, the programmed QoS value is associated with that transaction and is arbitrated with the rest of the controllers at SCB1.

Programming the SCB QoS Registers

Consider a scenario where:

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

Figure 8: Arbitration Among Various Controllers



As shown in [Figure 8](#), in this example:

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

Clock Domain Conversion

There are multiple Clock Domain Crossings (CDC) in the ADSP-21568 processor fabric:

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

Understanding the System Targets

Memory Hierarchy

As shown in [Table 2: Memory Targets and Maximum Theoretical Bandwidth](#) on page 5, ADSP-SC5xx processors have a hierarchical memory model (L1/L2/L3). The following sections discuss the access latencies and 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. 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+ processor 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+ processor, use a multiprocessor memory offset of 0×28000000 for all DMA accesses (including HSMDMA).

The maximum theoretical throughput of L1 memory (for system/DMA accesses) is $1000 * 4 = 4000$ MB/s for 933.6 MHz CCLK operation. As shown in [Figure 3: Measured MDMA Throughput on ADSP-21568 Processor](#) on page 6, the maximum measured L1 throughput using MDMA3 is approximately 3732.5 MB/s.

SHARC+ Processor System Optimization

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-21568 SHARC+ Processor Hardware Reference](#)^[1].

Consider the following important points regarding L2 memory throughput:

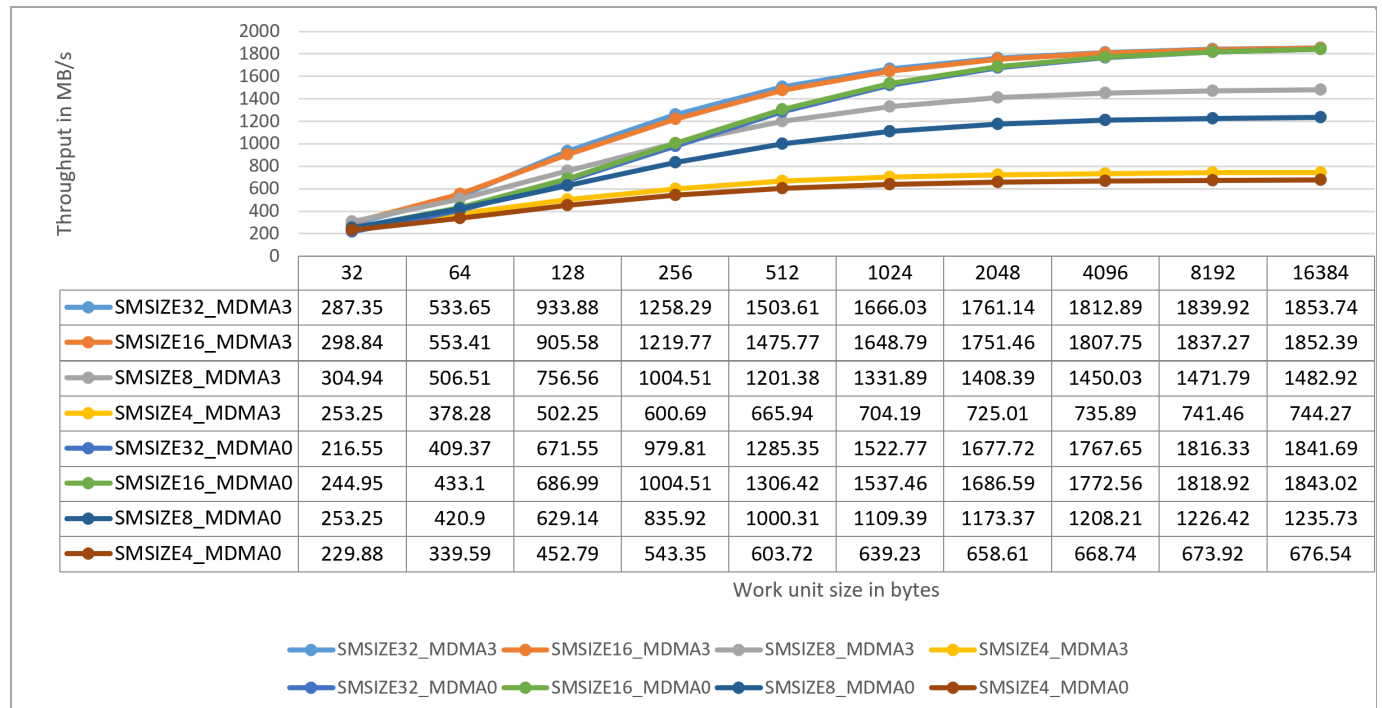
- Because L2 memory runs at the SYSCLK speed, it can provide a maximum theoretical throughput of $466.8 \text{ MHz} * 4 = 1867.2 \text{ MB/s}$ in one direction (for HSMDMA accesses, it has a maximum speed of 3734.4 MB/s). Because there are separate read and write channels, the total throughput in both directions equals 7468.8 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 larger.
- L2 memory throughput for sequential and non-sequential accesses is the same.
- L2 SRAM is parity-protected and organized into eight banks. The parity implementation is in terms of 32 bits. Therefore, for any writes less than 32-bit wide (one byte and half word) to the parity enabled RAM bank, the operation is implemented as a read-followed by a write. It requires two extra read cycles. However, all writes to parity-disabled banks and all writes that are 32/64 bit (with addresses aligned to 32-bit boundaries) do not have a read cycle in between.
- 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 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 zero, while programming the `L2CTL_RPCR.RPC1` and `L2CTL_WPCR.WPC1` bits to one.

[Figure 9](#) shows the measured MDMA throughput at CCLK = 933.6 MHz and SYSCLK = 466.8 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 1853 MB/sec in one direction (3706 MB/s in both directions) for MSIZE = 32 bytes and drops significantly for smaller MSIZE values.

SHARC+ Processor System Optimization



Figure 9: L2 MDMA Throughput for Different MSIZE and Work Unit Sizes



SHARC+ Processor System Optimization

System MMR Latencies

Table 5 shows the measured MMR latency in core cycles for different peripherals on the ADSP-21568 processor. The measurement was taken with CCLK = 933.6 MHz, SYCLK = 466.8 MHz, and SCLK = 116.7 MHz. These numbers can be used to approximate the MMR access latency of the SHARC+ core for different peripherals.

Table 5: MMR Access Latency Processors in Approximate CCLK Cycles

| S. No. | Register | Peripheral | Write Latency (Core Cycles) | Read Latency (Core Cycles) |
|--------|------------------|------------|-----------------------------|----------------------------|
| 1 | FIRO_INIDX | FIR | 40 | 39 |
| 2 | IIRO_INIDX | IIR | 40 | 39 |
| 3 | MECO_PERR_IMASK0 | MEC | 56 | 54 |
| 4 | CRC0_DCNT | CRC | 56 | 56 |
| 5 | CRC1_DCNT | | 56 | 56 |
| 6 | EMDMA0_INDX1 | EMDMA | 56 | 56 |
| 7 | EMDMA1_INDX1 | | 56 | 56 |
| 8 | TAPC_SDBGKEY0 | TAPC | 56 | 54 |
| 9 | L2CTL0_RPCR | L2CTL | 58 | 56 |
| 10 | SECO_RAISE | SEC | 58 | 58 |
| 11 | TRUO_SSRO | TRU | 58 | 58 |
| 12 | SPU0_SECUREP10 | SPU | 58 | 58 |
| 13 | RCU0_MSG | RCU | 58 | 56 |
| 14 | CDU0_CLKINSEL | CDU | 58 | 56 |
| 15 | DPM0_PER_DIS0 | DPM | 58 | 56 |
| 16 | PKTE0_SA_ADDR | PKTE | 60 | 60 |
| 17 | TRNG0_OUTPUT0 | TRNG | 64 | 64 |
| 18 | PKA0_APTR | PKA | 64 | 64 |
| 19 | PKIC0_ACK | PKIC | 64 | 64 |
| 20 | SPORT1_DIV_A | SPORT | 90 | 96 |
| 21 | PINT1_ASSIGN | PINT | 86 | 88 |
| 22 | OTPC_PMC_MODE0 | OTPC | 188 | 184 |
| 23 | DMA0_XCNT | DMA | 91 | 88 |
| 24 | PORTB_DATA_SET | PORT | 90 | 88 |
| 25 | WDOG1_WIN | WDOG | 88 | 88 |
| 26 | DMA1_XCNT | DMA | 88 | 88 |
| 27 | SPORT0_DIV_A | SPORT | 90 | 96 |
| 28 | UART0_CLK | UART | 86 | 88 |
| 29 | UART1_CLK | UART | 86 | 88 |
| 30 | PINT0_ASSIGN | PINT | 92 | 88 |
| 31 | WDOG0_WIN | WDOG | 92 | 88 |

SHARC+ Processor System Optimization

| S. No. | Register | Peripheral | Write Latency (Core Cycles) | Read Latency (Core Cycles) |
|--------|--------------------|------------|-----------------------------|----------------------------|
| 32 | SPI1_CLK | SPI | 88 | 96 |
| 33 | PORTA_DATA_SET | PORTA | 92 | 88 |
| 34 | PADS0_PORTA_PDE | PADS | 79 | 80 |
| 35 | CNT0_CNTR | CNT | 86 | 88 |
| 36 | TIMER0_TMRO_WID | TIMER | 86 | 96 |
| 37 | SPIO_CLK | SPI | 88 | 96 |
| 38 | PCG0_PW1 | PCG | 96 | 96 |
| 39 | SPDIFO_TX_UBUFF_A0 | SPDIF | 96 | 96 |
| 40 | ASRC1_MUTE | ASRC | 106 | 96 |
| 41 | DAI1_IMSK_FE | DAI | 94 | 96 |
| 42 | SPDIF1_TX_UBUFF_A0 | SPDIF | 98 | 96 |
| 43 | DAI0_IMSK_FE | DAI | 94 | 96 |
| 44 | ASRC0_MUTE | ASRC | 96 | 96 |
| 45 | SMPU2_RADDR0 | SMPU | 58 | 56 |
| 46 | TWI0_CLKDIV | TWI | 128 | 136 |
| 47 | TWI1_CLKDIV | TWI | 128 | 136 |

Note: 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 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

System Optimization Techniques

[Table 6](#) summarizes the optimization techniques discussed in this application note, while also listing a few additional tips for bandwidth optimization.

Table 6: System Optimization Techniques Checklist

| | Optimization Tip Description |
|--------------------------|--|
| <input type="checkbox"/> | Analyze the overall bandwidth requirements and use the bandwidth limit feature for memory pipe DMA channels to regulate the overall DMA traffic. |
| <input type="checkbox"/> | Program the DMA channel MSIZE parameters to optimal values to maximize throughput and avoid any potential underflow/overflow conditions. |
| <input type="checkbox"/> | 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. |
| <input type="checkbox"/> | Use MDMA instead of EMDMA for sequential data transfers to improve performance. When possible, emulate EMDMA non-sequential transfer modes with MDMA. |
| <input type="checkbox"/> | Program the SCB RQOS and WQOS registers to allocate priorities to various controllers as per system requirements. |
| <input type="checkbox"/> | Use optimization techniques at the SCB target end, such as: <ul style="list-style-type: none"> • Multiple L2/L1 sub-banks to avoid access conflicts • Instruction/data caches |
| <input type="checkbox"/> | Maintain the optimum clock ratios across different clock domains. |
| <input type="checkbox"/> | Because MMR latencies affect the interrupt service latency, ADSP-21568 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-21568 processor.

References

- [1] *ADSP-21568 SHARC+ Processor Hardware Reference*. Revision 1.1, October 2022. Analog Devices, Inc. https://www.analog.com/media/en/dsp-documentation/processor-manuals/adsp-2156x_hwr.pdf
- [2] *ADSP-21568 SHARC+ Single Core High Performance DSP Data Sheet*. Rev. D, June 2023. Analog Devices, Inc. <https://www.analog.com/media/en/technical-documentation/data-sheets/adsp-21562-21563-21565-21566-21567-21569.pdf>
- [3] *Source Files for EE-412: ADSP-2156x SHARC+ Processors System Optimization Techniques*. Rev. 2, September 2019. Analog Devices, Inc. <https://www.analog.com/media/en/technical-documentation/application-notes/ee412v02.zip>
- [4] *Source Files for EE-461: ADSP-2156x SHARC+ Processors System Optimization Techniques*. Rev 2.0, June 2024. Analog Devices, Inc. <https://www.analog.com/media/en/technical-documentation/application-notes/ee461v01.zip>
- [5] *Utilizing the Trigger Routing Unit for System Level Synchronization (EE-360)*. Rev 1.0, October 2013. Analog Devices. <https://www.analog.com/media/en/technical-documentation/application-notes/EE360v01.pdf>
- [6] *ADSP-SC5xx/215xx SHARC+ Processor System Optimization Techniques (EE-401)*. Rev 1.0, February 2018, Analog Devices, Inc. <https://www.analog.com/media/en/technical-documentation/application-notes/ee-401.pdf>

Document History

| Date | Author(s) | Description of EE-Note Changes |
|-----------|-------------------|--|
| June 2024 | Tejaswi Chitneedi | Initial Release |
| June 2024 | Tejaswi Chitneedi | Corrected frequency typo, replacing 933.6 GHz with 933.6 MHz |