Application Report
Lit. Number – April 2011

TMS320C6678 Memory Access Performance

Brighton Feng Communication Infrastructure

ABSTRACT

The TMS320C6678 has eight C66x cores, runs at 1GHz, each of them has 32KB L1D SRAM, 32KB L1P SRAM and 512KB LL2 SRAM; all DSP cores share 4MB SL2 SRAM. A 64-bit 1333M DDR3 SDRAM interface is provided on the DSP to support up to 8GB external memory.

Memory access performance is very critical for software running on the DSP. On C6678 DSP all the memories can be accessed by DSP cores and multiple DMA masters.

Each DSP core is capable of performing up to 128 bits of load/store operations per cycle. When accessing L1D SRAM the DSP core can access the memory at up to 16GB/second at a 1GHz core clock frequency.

The TeraNet switch fabric, which provides the interconnection between the C66x cores (and their local memories), external memory, the EDMA controllers, and on-chip peripherals, has access port to each end point. There are ten EDMA transfer controllers that can be programmed to move data, concurrently, between any memory endpoints on the device.

This document gives designers a basis for estimating memory access performance, provides measured performance data achieved under various operating conditions. Some factors affecting memory access performance are discussed.

Contents

Figures

Tables

Introduction

The TMS320C6678 has eight C66x cores, each of them has:

- 32KB L1D (Level 1 Data) SRAM, which runs at the DSP Core speed, can be used as normal data memory or cache.
- 32KB L1P (Level 1 Program) SRAM, which runs at the DSP Core speed, can be used as normal program memory or cache
- 512KB LL2 (Local Level 2) SRAM, which runs at the DSP Core speed divided by two, can be used as normal RAM or cache for both data and program.

All DSP cores share 4MB SL2 (Shared Level 2) SRAM, which runs at the DSP Core speed divided by two, can be used as data or code memory.

A 64-bit 1333M DDR SDRAM interface is provided on the DSP to support up to 8GB external memory, which can be used as data or program memory. The interface can also be configured to only use 32 bits or 16 bits data bus.

Memory access performance is very critical for software running on the DSP. On C6678 DSP, all the memories can be accessed by DSP cores and multiple DMA masters.

Each TMS320C66x core has the capability of sustaining up to 128 bits of load/store operations per cycle to the level-one data memory (L1D), capable of handling up to 16GB/second at 1GHz DSP core speed. When accessing data in the level-two (L2) unified memory or external memory, the access rate will depend on the memory access pattern and cache.

There is an internal DMA (IDMA) engine that can move data at a rate of the DSP Core speed divided by two, capable of handling up to 8GB/second at a 1GHz core clock frequency, in the background of DSP core activity (i.e. data can be brought in to buffer A while the DSP core is accessing buffer B). The IDMA can only transfer data between level-one (L1), local level-two (LL2) and peripheral configuration port, it can not access external memory.

The TeraNet switch fabric, which provides the interconnection between the C66x cores (and their local memories), external memory, the enhanced DMA v3 (EDMA3) controllers, and on-chip peripherals, there are two main TeraNet switch fabrics, one has 128 bit access bus to each end point, runs at DSP core frequency divided by three, so, in theory, capable of sustaining up to 5.333GB/second at 1GHz core clock frequency; the other TeraNet switch fabric has 256 bit access bus to each end point, runs at DSP core frequency divided by two, so, in theory, capable of sustaining up to 16GB/second at 1GHz core clock frequency.

There are ten EDMA transfer controllers that can be programmed to move data, concurrently, in the background of DSP core activity, between the on-chip level-one (L1), level-two (L2) memory, external memory, and the peripherals on the device, two of them connect to the 256-bit TeraNet switch fabric at DSP core clock divided by 2, the other eight connect to the 128-bit TeraNet switch fabric at DSP core clock divided by 3. The EDMA3 architecture has many features designed to facilitate simultaneous multiple high-speed data transfers. With a working knowledge of this architecture and the way in which data transfers interact and are performed, it is possible to create an efficient system and maximize the bandwidth utilization of the EDMA3.

Following figure shows the memory system of TMS320C6678. The number on the line is the bus width. Most modules run at CoreClock/n, the DDR typically runs at 1333M.

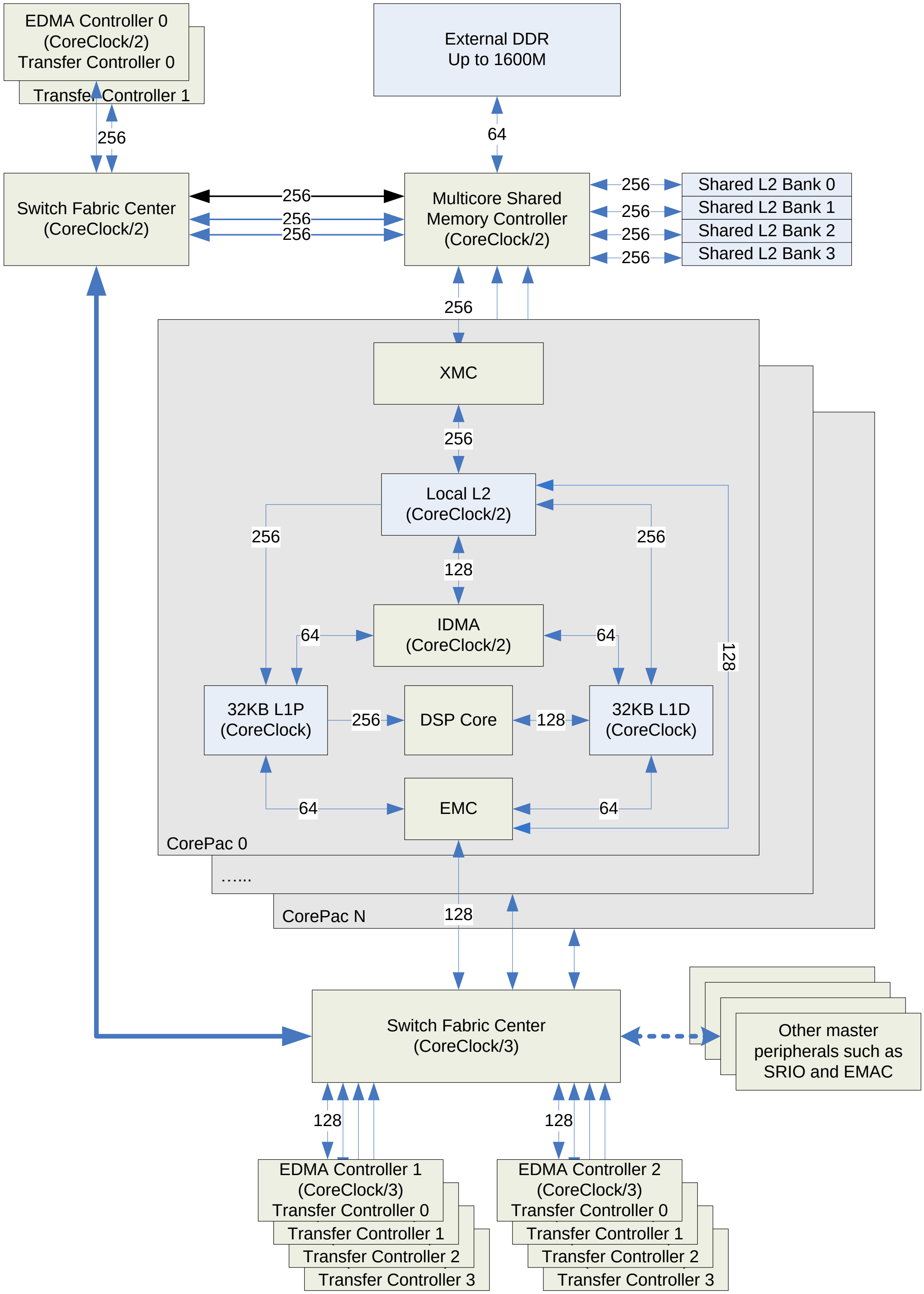


Figure 1. TMS320C6678 Memory System

This document gives designers a basis for estimating memory access performance, provides measured performance data achieved under various operating conditions. Most of the tests operate under best-case situations to estimate maximum throughput that can be obtained. The transfers described in this document serve as a sample of interesting or typical performance conditions.

Some factors affecting memory access performance are discussed in this document, such as access stride, index and conflict, etc.

The document should be helpful for analyzing following common questions:

1. Should I use DSP core or DMA for data copy?
2. How many cycles will be consumed for my function with many memory accesses?
3. How much degradation will be caused by multiple masters sharing memory?

Most of the performance data in this document is examined on the C6678 EVM (EValuation Module) with 64-bit 1333M DDR memory.

DSP Core vs EDMA3 and IDMA for memory copy

The bandwidth of memory copy is limited by the worst of following three factors:

1. Bus bandwidth
2. source throughput
3. destination throughput

Following tables summarizes the theoretical bandwidth of the C66x core, IDMA and EDMA on C6678.

Table 1. Theoretical bus bandwidth of DSP core, IDMA and EDMA on 1GHz C6678

Master	Maximum bandwidth MB/s	Comments
C66x core	16,000	(128 bits)/ (8 bit/byte)*1000M= 16000MB/s
IDMA	8,000	(64 bits)/ (8 bit/byte)*1000M = 8000MB/s
EDMA CC0 (TC0, 1)	16,000	(256 bits)/(8 bit/byte)*(1000M/2)=16000MB/s
EDMA CC1, 2 (TC0~3)	5,333	(128 bits)/(8 bit/byte)*(1000M/3)=5333MB/s

Following tables summarizes the theoretical throughput of different memories on C6678 EVM with 64-bit 1333M DDR external memory.

Table 2. Maximum Throughput of Different Memory Endpoints on 1GHz C6678

Memory	**Maximum Bandwidth MB/s**	**Comments**
L1D	32,000	(256 bits)/ (8 bit/byte)*(1000M) = 32000MB/s
L1P	32,000	(256 bits)/ (8 bit/byte)*(1000M) = 32000MB/s
LL2	16,000	(256 bits)/ (8 bit/byte)*(1000M/2) = 16000MB/s
SL2	64,000	(4*256 bits)/ (8 bit/byte)*(1000M/2) = 64000MB/s
DDR	10,666	(64 bits)/(8 bit/byte)*(1333M)=10666MB/s

Following table shows the transfer bandwidth measured for large linear memory block copy with EDMA, IDMA and DSP Core for different scenarios on 1GHz C6678 EVM with 64-bit 1333M DDR.

In these tests, the memory block size for L1 copy is 8KB; the memory block size for other DSP core copy is 64KB, the memory block size for other EDMA copy is 128KB; IDMA LL2->LL2 block size is 32KB.

The bandwidth is measured by taking total bytes copied and dividing it by the time it used.

Table 3. Transfer bandwidth comparison between DSP core, EDMA and IDMA

Bandwidth(MB/s) for Src-> Dst	**1GHz DSP**		
	DSP core	EDMA	IDMA
LL2 -> LL2 (32KB L1D cache)	2557	4939	2356
LL2-> L1D (16KB L1D cache)	3927	4303	3343
L1D-> LL2 (16KB L1D cache)	7713	4303	4156
LL2-> SL2 (32KB L1D cache, prefetchable default SL2 memory space)	3756		
LL2-> SL2 (noncacheable, nonprefetchable remapped SL2 memory space)	2264	5266	N/A
LL2-> SL2 (32KB L1D cache, prefetchable remapped SL2 memory space)	3362		
SL2-> LL2 (32KB L1D cache, prefetchable default SL2 memory space)	3270		
SL2-> LL2 (noncacheable, nonprefetchable remapped SL2 memory space)	591	5266	N/A
SL2-> LL2 (32KB L1D cache, prefetchable remapped SL2 memory space)	2606		

SL2-> SL2 (32KB L1D cache, prefetchable default SL2 memory space)	2660	7878	N/A
SL2-> SL2 (noncacheable, nonprefetchable remapped SL2 memory space)	484		
SL2-> SL2 (32KB L1D cache, prefetchable remapped SL2 memory space)	2454		
LL2-> LL2 of another core (non-cacheable, nonprefetchable)	1642	5253	N/A
LL2-> LL2 of another core (32KB L1D cache, prefetchable)	2675		
LL2-> LL2 of another core (32KB L1D, 256KB L2 cache, prefetchable)	1842		
LL2 of another core-> LL2 (non-cacheable, nonprefetchable)	183		
LL2 of another core-> LL2 (32KB L1D cache, prefetchable)	1182		
LL2 of another core-> LL2 (32KB L1D, 256KB L2 cache, prefetchable)	1838		
LL2 -> 64-bit DDR (non-cacheable, nonprefetchable)	1336	5267	N/A
LL2 -> 64-bit DDR (32KB L1D cache, prefetchable)	2677		
LL2 -> 64-bit DDR (32KB L1D, 256KB L2 cache, prefetchable)	2109		
64-bit DDR -> LL2 (non-cacheable, nonprefetchable)	188	5253	N/A
64-bit DDR -> LL2 (32KB L1D cache, prefetchable)	1321		
64-bit DDR -> LL2 (32KB L1D, 256KB L2 cache, prefetchable)	2025		
64-bit DDR -> 64-bit DDR (non-cacheable, nonprefetchable)	150	3878	N/A
64-bit DDR -> 64-bit DDR (32KB L1D cache, prefetchable)	976		
64-bit DDR -> 64-bit DDR (32KB L1D, 256KB L2 cache, prefetchable)	1847		

Generally speaking, DSP core accesses internal memory efficiently, while using the DSP core to access external data is a bad use of resources and should be avoided; The IDMA is good at linearly moving a block of data in internal memory (L1D, L1P, LL2), it can not access external memory; The EDMA3 should be given the task of transferring data to/from external memory.

The cache configurations dramatically affect the DSP core performance, but it does not affect EDMA and IDMA performance. All test data for DSP core in this application note are based on cold cache, i.e, all the caches are flushed before the test.

For DSP core, SL2 can be accessed through default space at 0x0C000000, normally that is for cacheable and prefetchable access. SL2 can be remapped to other memory space through XMC (eXtended Memory Controller), normally that is for non-cacheable nonprefetchable access, and it can also can be configured as cacheable and prefetchable. Access through default space is a little bit faster than access through remapped space. The XMC prefetch buffer improves the performance of read from the SL2 dramatically.

Above EDMA throughput data is measured on TC0 (Transfer Controller 0) of EDMA CC0 (Channel Controller 0), while the throughput of EDMA CC1 and EDMA CC2 is not as good as EDMA CC0, see more details in following section for the comparison between ten DMA transfer controllers.

Performance of DSP core access memory

L1 runs at the same speed as DSP core, so DSP core can access L1 memory one time per cycle. For some special application which requires accessing a small data block very quickly, part of the L1 can be used as normal RAM to store the small data block.

Normally, L1 is used as cached, if cache hit happens, DSP core can access data in one cycle; if cache miss happens, the DSP core stalls until the data coming into the cache.

The following sections examine the access performance for DSP Core accesses internal memory and external DDR memory. The pseudo codes for this test are like following:

```
flushCache();
preCycle= getTimeStampCount();
for(i=0; i< accessTimes; i++)
{
        Access Memory at address;
        address+= stride;
}
cycles = getTimeStampCount()-preCycle;
cycles/Access= cycles/accessTimes;
```

Performance of DSP core access LL2

Following figure shows data collected from 1GHz C6678 EVM. The time required for 512 consecutive LDDW (LoaD Double Word) or STDW (STore Double Word) instructions was measured, and the average time for each instruction is reported. 32KB L1D cache is used for this test. The cycles for LDB/STB, and LDW/STW are same as LDDW/STDW.

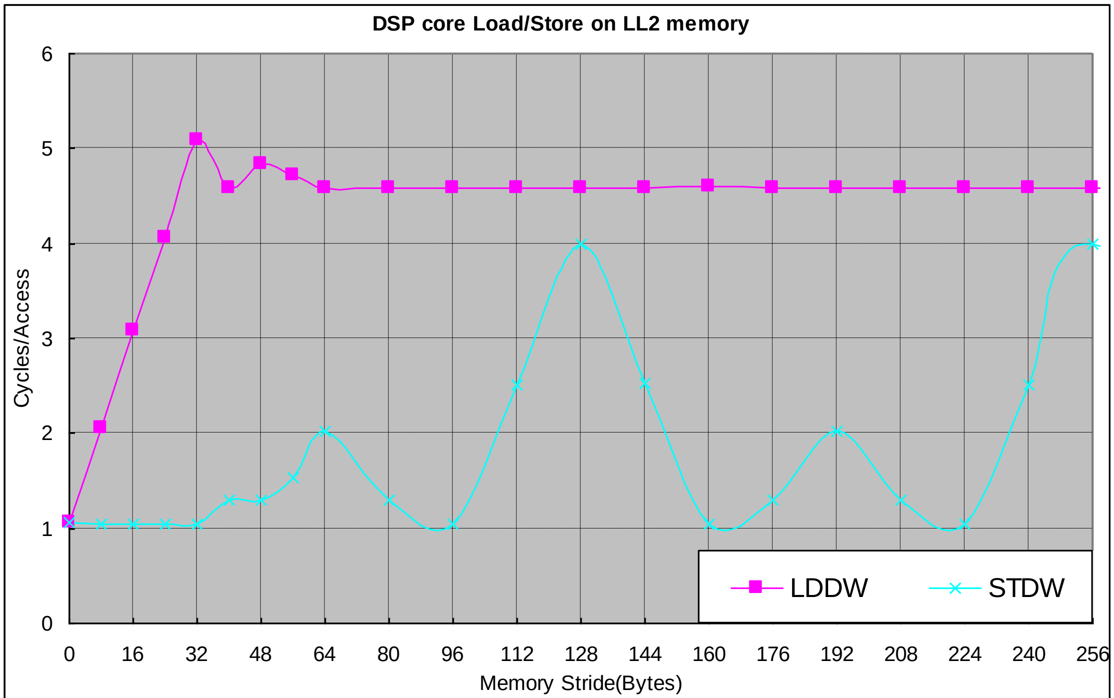


Figure 2. DSP Core access LL2

Since the L1D is a read-allocate cache, DSP core read LL2 should always go through L1D cache. So, DSP core access LL2 highly depends on the cache. The address increment (or memory stride) affects cache utilization. Contiguous accesses utilize cache to the fullest. A memory stride of 64 bytes or more causes every access to miss in the L1 cache because the L1D cache line size is 64 bytes.

Since the L1D is not a write-allocate cache, and the cache is flushed before the test, any write to the LL2 goes through L1D write buffer (4x16bytes). For write operation, if stride is less than 16 bytes, several writes may be merged into one write to the LL2 in the L1D write buffer, thus achieves the efficiency close to 1 cycle/write. When the stride is multiple of 128 bytes, every write always access to the same sub-bank of LL2 (because the LL2 is organized as 2 banks, each with 4 16-byte sub-banks), which requires 4 cycles. For other strides, the Consecutive writes access to different banks of LL2, they may be overlapped with pipeline, which requires less cycle.

C66x core is an improved core from C64x+ core, the L2 memory controller of C66x runs at DSP core speed, while C64x+ L2 memory controller runs at DSP core clock divided by two. Following figure shows the DSP core Load/Store on LL2 memory, C66x vs C64x+.

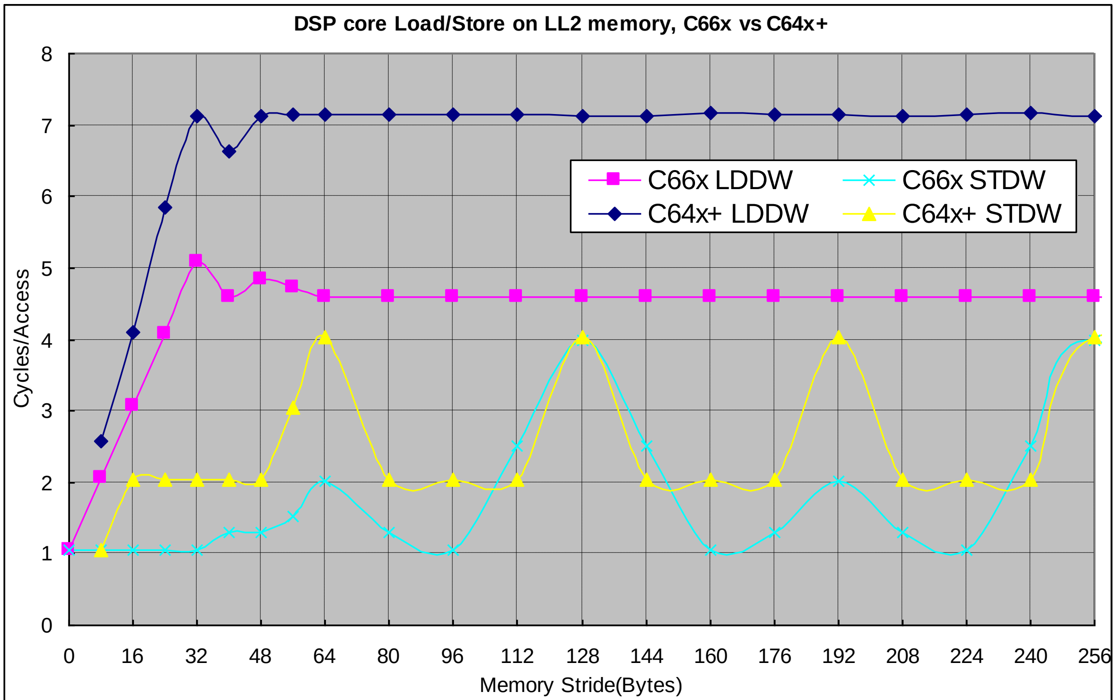


Figure 3. DSP core Load/Store on LL2 memory, C66x vs C64x+

Performance of DSP core access SL2

Following figure shows data collected from 1GHz C6678 EVM. The time required for 512 LDDW (LoaD Double Word) or STDW (STore Double Word) instructions was measured, and the average time for each instruction is reported. 32KB L1D cache is used for this test. The cycles for LDB/STB, and LDW/STW are same as LDDW/STDW.

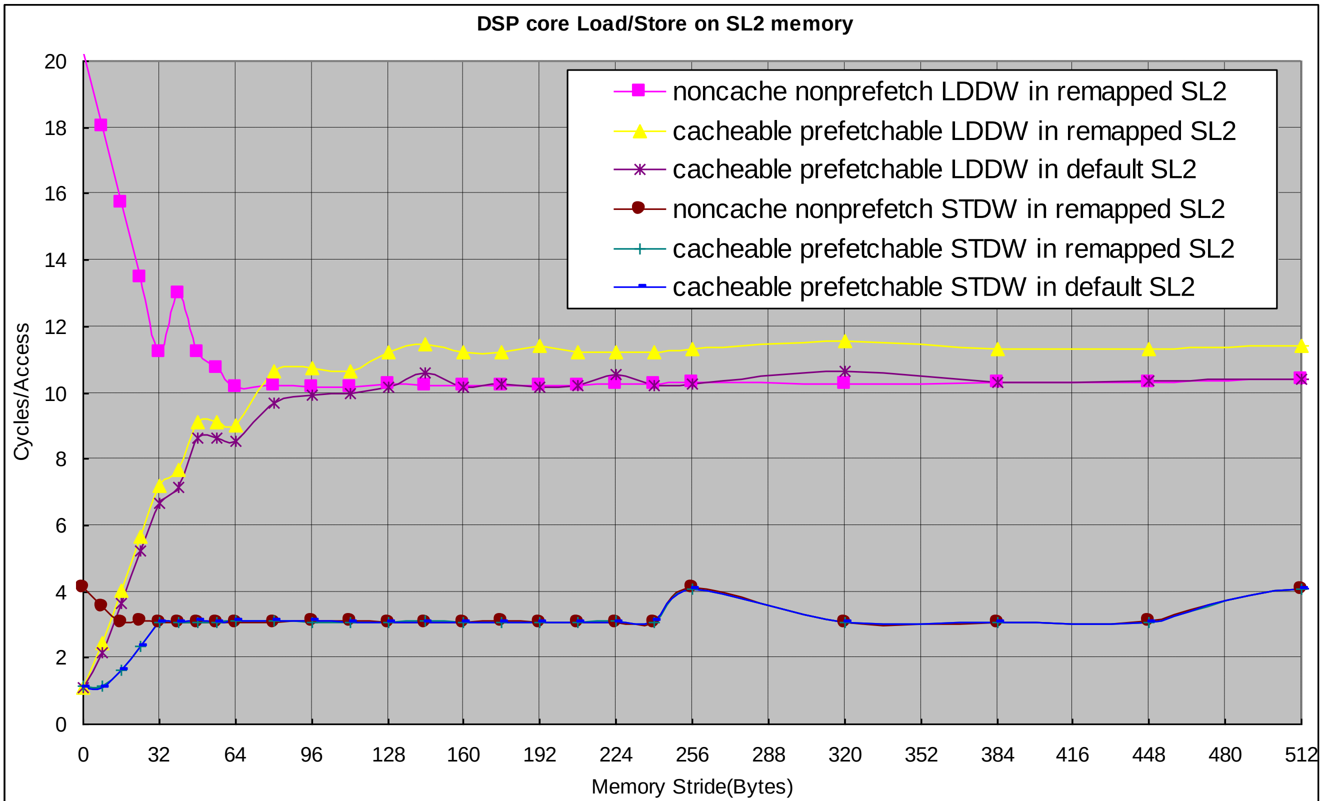


Figure 4. DSP Core access SL2

DSP core read SL2 should normally goes through L1D cache, so, DSP core access SL2 highly depends on the cache just like LL2. There is an additional data prefetch buffer (8x128bytes) inside XMC, which can be look as an additional cache for read only, which is configurable by software through PFX (PreFetchable eXternally) bit of MAR (Memory Attribute Register), enabling it will benefit mulit-core access, it improves the performance of consecutive read from the SL2 dramatically. But prefetch buffer does not help write operation.

SL2 can be accessed through default space at 0x0C000000, which is always cacheable, normally that is also be configured as prefetchable. SL2 can be remapped to other memory space through XMC, normally that is for non-cacheable nonprefetchable access, but it can also be configured as cacheable and prefetchable. Access through default space is a little bit faster than access through remapped space, because the address remapping consumes about one more cycle.

Since the L1D is not a write-allocate cache, any write may go through L1D write buffer (4x16bytes) to the SL2. For write operation, if stride is less than 16 bytes, several writes may be merged into one write to the SL2 in the L1D write buffer, thus achieves better efficiency. The XMC has similar write merge buffer, which can merger two write in 32 bytes, so for write operation with stride less than 32 bytes, the XMC write merge feature improves the performance.

When the stride is N*256 bytes, every write always access to the same bank of SL2 (the SL2 is organized as 4 bank x 2 sub-bank x 32 bytes), which may result in about 4 cycles per access. For other strides, the consecutive writes access to different banks of LL2, they may be overlapped with pipeline, which requires less cycle.

Following figure shows performance comparison of DSP core access SL2 vs LL2. For memory stride less than 16 bytes, the performance of SL2 access is almost same as LL2. For access with bigger memory stride, the performance of SL2 is worse than LL2. So, SL2 is suitable for linearly accessed codes or read only data.

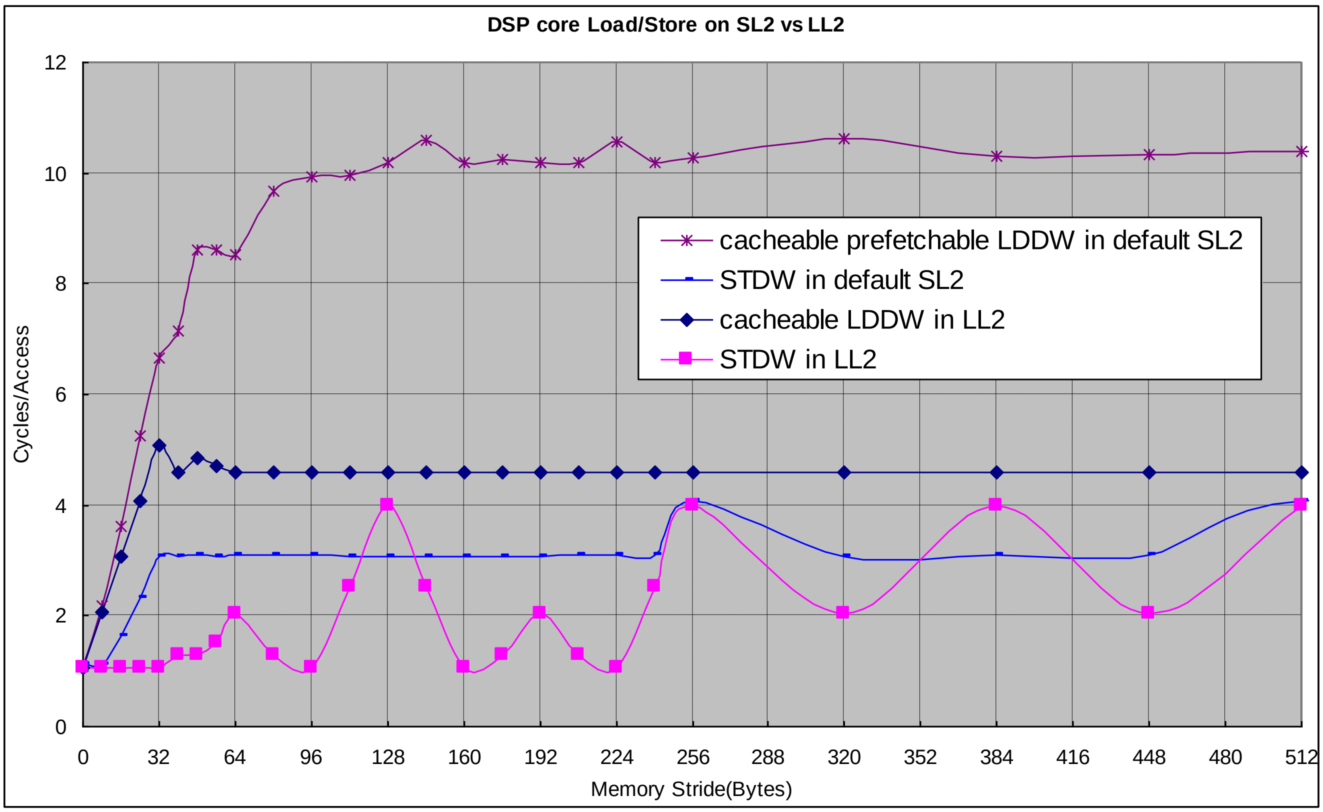


Figure 5. Performance of DSP core access SL2 vs LL2

Performance of DSP core access external DDR memory

DSP core access external DDR memory highly depends on the cache. When the DSP core accesses external memory spaces, a TR (transfer request) may be generated (depending on whether the data are cached and prefetchable) to the XMC. The TR will be for one of the following:

- a single element - if the memory space is non-cacheable, nonprefetchable
- a L1 cache line - if the memory space is cacheable and the L2 cache is disabled,
- a L2 cache line - if the memory space is cacheable and L2 cache is enabled.
- If the space is prefetchable, prefetch may be generated for a prefetch buffer slot.

No transfer request is generated in the case of an L1/L2 cache or prefetch hit.

An external memory can be cached by L1 cache, L2 cache, or neither. If the PC (Permit Copy) bit of appropriate MAR (Memory Attribute Register) for a memory space is not set, it is not cacheable. If the PC bit of MAR is set and L2 cache size is zero (all L2 is defined as SRAM), the external memory space is cached by L1. If the MAR bit is set and L2 cache size is greater than 0, the external memory space is cached by L2 and L1.

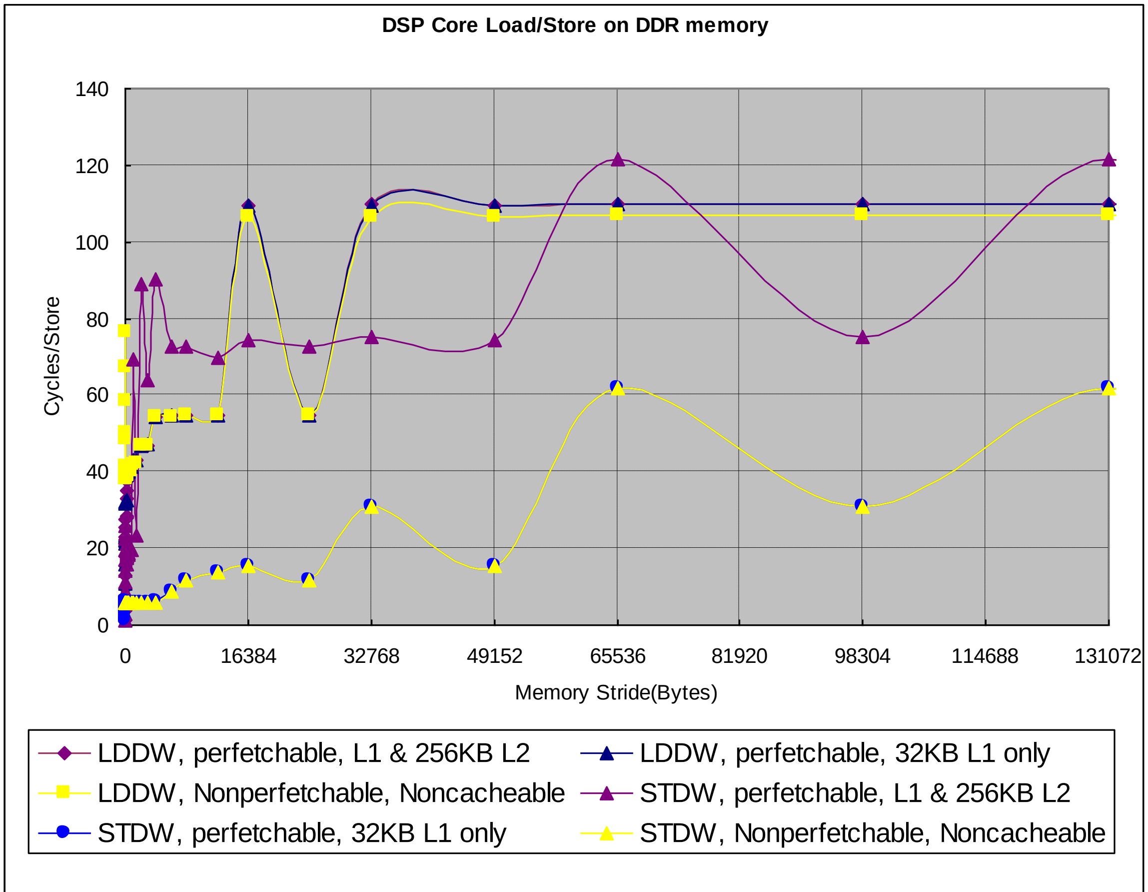
Read to external memory can also utilize the prefetch buffer in XMC, which is programmable by software through PFX (PreFetchable eXternally) bit of MAR (Memory Attribute Register).

The address increment (or memory stride) affects cache and prefetch buffer utilization. Contiguous accesses utilize cache and prefetch memory to the fullest. A memory stride of 64 bytes or more causes every access to miss in the L1 cache because the L1 line size is 64 bytes. A memory stride of 128 bytes causes every access to miss in L2 because the L2 line size is 128 bytes.

If cache miss happens, the DSP Core will stall, waiting for the return data. The length of the stall is equal to the sum of the transfer latency, transfer duration, data return time, and some small cache request overhead.

Following figures show data collected from 1GHz C6678 EVM with 64-bit 1333M DDR. The time required for 512 LDDW (LoaD Double Word) or STDW (STore Double Word) instructions was measured, and the average time for each instruction is reported. 32KB L1D cache and 256KB L2 cache are used for this test. The cycles for LDB/STB, and LDW/STW are same as LDDW/STDW.

Note, the second and third figures are the zoom in version of the first figure;


DSP Core Load/Store on DDR memory
Cycles/Store
0
20
40
60
80
100
120
140
0
16384
32768
49152
65536
81920
98304
114688
131072
Memory Stride(Bytes)
LDDW, perfetchable, L1 & 256KB L2
LDDW, perfetchable, 32KB L1 only
LDDW, Nonperfetchable, Noncacheable
STDW, perfetchable, L1 & 256KB L2
STDW, perfetchable, 32KB L1 only
STDW, Nonperfetchable, Noncacheable

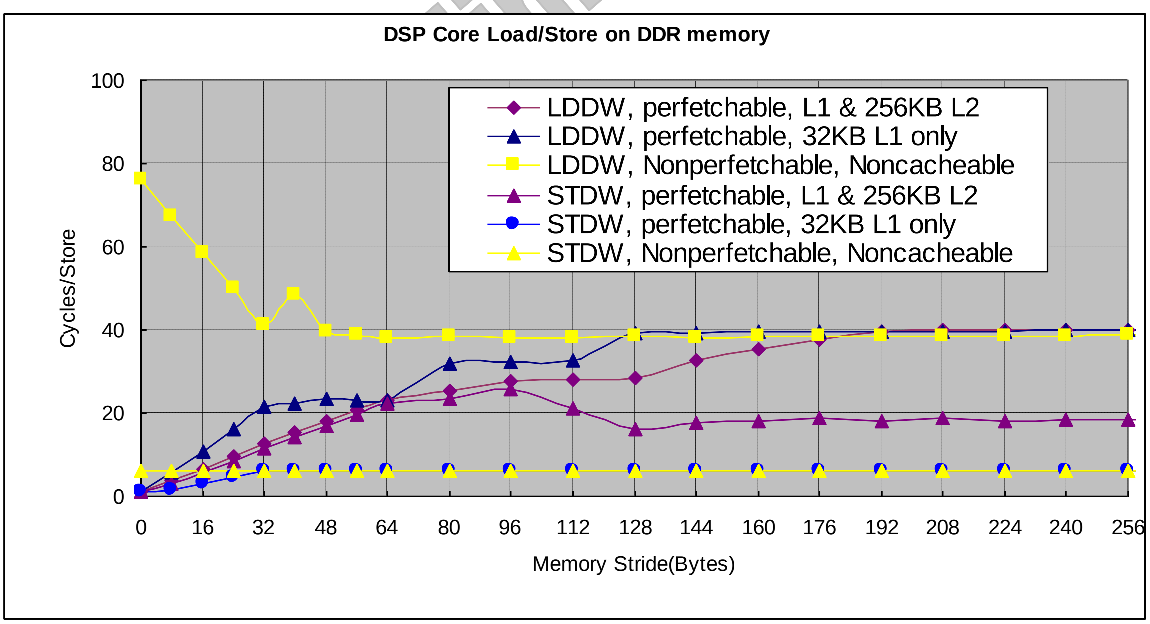


Figure 6. DSP Core Load/Store on DDR

For memory stride less than 128 bytes, the performance is dominated by cache as discussed above.

L2 cache is a write-allocate cache, for any write operation, it always read the 128 bytes including the accessed data into a cache line firstly, and then modify the data in the L2 cache. This data will be written back to real external memory if cache conflict happens or by manual writeback. When the memory stride equals to multiple of 1024 bytes, the cycles for L2 cacheable write operation increases dramatically, because the conflict happens frequently for big memory stride, thus every write operation may result in a cache line write back (for conflict) and a cache line read (for write-allocate).

For memory stride larger than 512 bytes, DDR row switch overhead becomes a main factor of performance degrading. The DDR row size or bank width on the C6678 EVM is 8KB, the DDR is organized as 8 banks. For access with memory stride equal to multiple of 64KB, the performance becomes worst because every read or write access to a new row in same bank, the row switch result in about 40 extra cycles. Please note, the DDR SDRAM row switch overhead may be different for different DDR SDRAM.

Performance of DMA access memory

The EDMA3 architecture has many features designed to facilitate simultaneous multiple high-speed data transfers. Its performance affected by the memory type and many other factors discussed in following sections.

DMA Transfer overhead

Initial latency is defined as the time between DMA event happen to real data transfer begin. Since initial latency is hard to measure. We measured transfer overhead instead; it is defined as the sum of the Latency, and the time to transfer smallest element. The values vary based on the type of source/destination endpoints. Following tables show the average cycles measured between EDMA trigger (write ESR) and EDMA completion (read IPR=1) for smallest transfer (1 word) between different ports on 1GHz C6678 EVM with 64-bit 1333M DDR.

Table 4. EDMA CC0 Transfer Overhead

destination / source	L1D	LL2	SL2	DDR
L1D	331	333	300	333
LL2	333	331	300	331
SL2	300	267	267	268
DDR	334	331	334	399

Table 5. EDMA CC1 and EDMA CC2 Transfer Overhead

destination / source	L1D	LL2	SL2	DDR
L1D	271	271	322	322
LL2	271	271	322	322
SL2	322	322	322	373
DDR	373	373	373	475

Since EDMA CC0 is connected to TeraNet switch fabric close to SL2 and DDR, so it's overhead to access SL2 and DDR is smaller. While EDMA CC1 and CC2 are connected to TeraNet switch fabric close to CorePac which including L1 and LL2, so their overhead to access L1 and LL2 is smaller.

The average measured IDMA transfer overhead is about 61 cycles.

Transfer overhead is a big concern for short transfers and need to be included when scheduling DMA traffic in a system. Single-element transfer performance will be latency-dominated. So, for small transfer, you should make the trade off between DMA and DSP core.

EDMA performance Difference between 10 transfer engines

EDMA3 on C6678 includes 10 TC (transfer controller). The 10 transfer engines are not exactly same. Table 6 is a summary of the difference.

Table 6. Difference between TCs

Name	TC0_0	TC0_1	TC1_0	TC1_1	TC1_2	TC1_3	TC2_0	TC2_1	TC2_2	TC2_3
Bus Width (bits)	256	256	128	128	128	128	128	128	128	128
Speed ratio to DSP core	1/2	1/2	1/3	1/3	1/3	1/3	1/3	1/3	1/3	1/3
FIFO Size (bytes)	1024	1024	1024	512	1024	512	1024	512	512	1024
Default Burst Size (bytes)	128	128	128	64	128	64	128	64	64	128

Following tables compare the maximum throughput measured for different TCs on 1GHz C6678 EVM with 64-bit 1333M DDR. AB_Sync is used, ACNT=1024, BCNT=128 is different for different case.

Table 7. Throughput comparison between TCs on 1GHz C6678

MB/s	TC0_0	TC0_1	TC1_0	TC1_1	TC1_2	TC1_3	TC2_0	TC2_1	TC2_2	TC2_3
LL2 -> LL2	5252	5252	5036	5036	5036	5036	5036	5036	5036	5036
LL2 -> SL2	5267	5267	5036	5036	5036	5036	5036	5036	5036	5036
SL2 -> LL2	5267	5267	5036	5036	5036	5036	5036	5036	5036	5036
SL2 -> SL2	7880	7880	5026	5026	5026	5026	5026	5026	5026	5026
LL2 -> DDR	5267	5267	5026	4987	5026	4987	5026	4997	4987	5026
DDR -> LL2	5252	5211	4948	3597	4968	3602	4977	3592	3602	4968
DDR -> DDR	3870	3878	3091	2976	3091	3043	3091	2980	2976	3091
SL2->DDR	10396	10396	5026	5026	5026	5026	5026	5026	5026	5026
DDR->SL2	9780	9926	5016	3519	5016	3553	5016	3523	3562	5016

For data transfer between SL2 and DDR, TC0_0 and TC0_1 achieve about two times bandwidth as other TCs. Without special note, all performance data in this application report is measured on TC0_0.

EDMA Bandwidth vs Transfer Flexibility

EDMA3 channel parameters allow many different transfer configurations. Most typical transfers burst properly, and memory bandwidth is fully utilized. However, in some less common configurations, transfers are unable to burst, reducing performance. To properly design a system, it is important to know which configurations offer the best performance for high speed operations, and which must trade throughput for flexibility.

First Dimension Size (ACNT) Considerations, Burst width

To make full utilization of bandwidth in the transfer engine, it is important to fully utilize the bus width available and allow for data bursting.

ACNT size should be multiple of 16 bytes to fully utilize 128bit or 256bit bus width; ACNT should be multiple of 64 bytes to fully utilize default burst size; ACNT should be multiple of 512 bytes to fully utilize the FIFO.

Figure 7 shows performance data from 1GHz C6678 EVM with 64-bit 1333M DDR, transferring 1~24K bytes from SL2 to DDR using an EDMA3 channel.

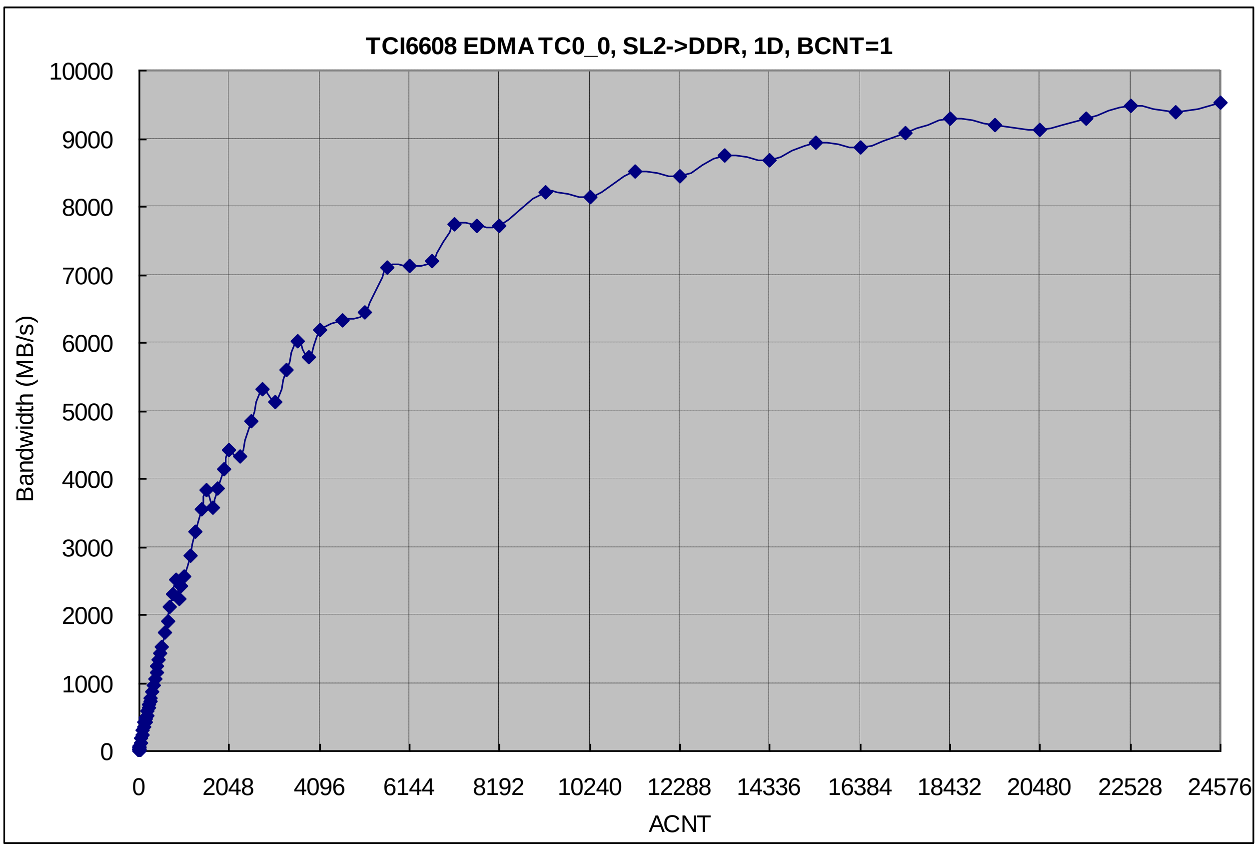


Figure 7. effect of ACNT size on EDMA bandwidth

As conclusion, the bigger ACNT, the more bandwidth can be utilized.

Two Dimension Considerations, Transfer Optimization

If 2D transfer (AB_Sync) is linear (BIDX=ACNT), and the ACNT value is a power of 2, the 2D transfer will be optimized as 1D transfer. Various ACNT and BCNT combinations were investigated; however, the overall transfer size (ACNT * BCNT) was proved to have more bearing than the particular combination settings. Figure 8 is linear 2D transfer test result on 1GHz C6678 EVM with 64-bit 1333M DDR, it shows, no matter what BCNT, the bandwidth are similar as long as ACNTxBCNT are same.

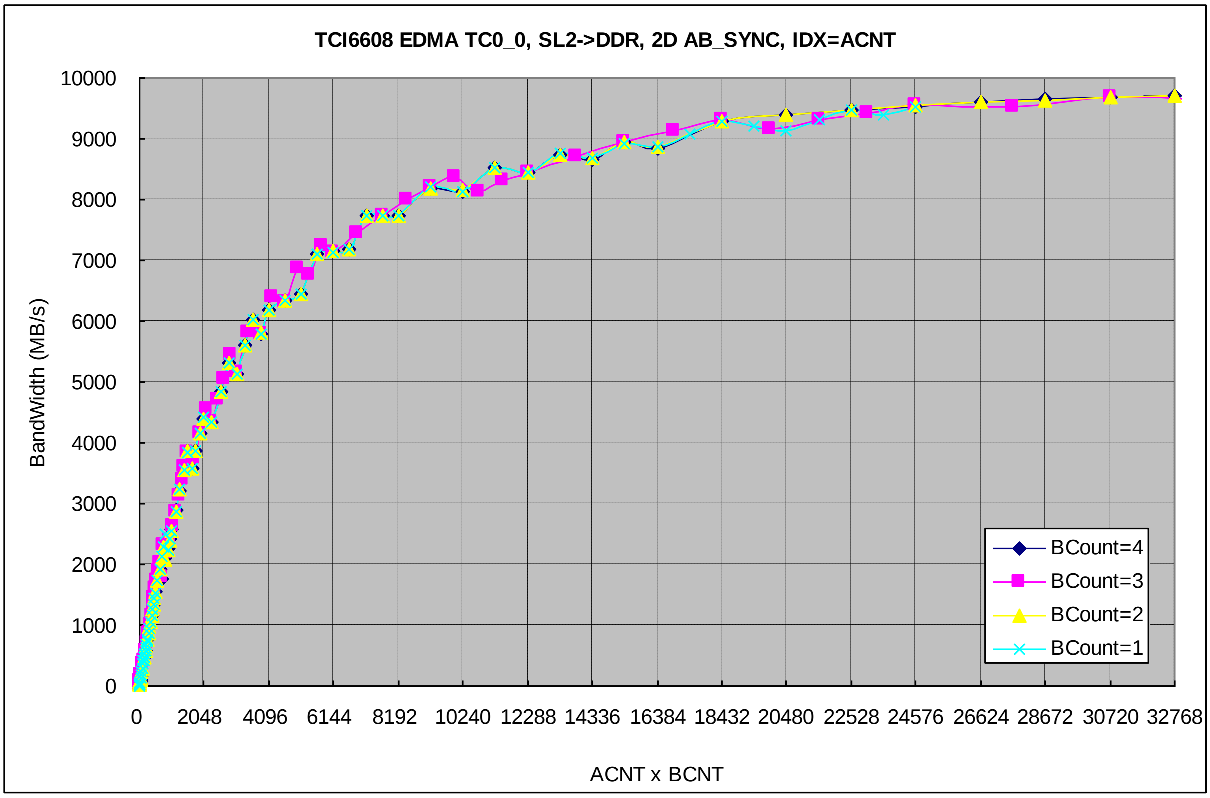


Figure 8. Linear 2D transfer

If 2D transfer is not linear, the bandwidth utilization is determined by the ACNT as showed in Figure 7.

Index Consideration

Index dramatically affects the EDMA throughput. Linear transfer (Index= ACNT) fully utilizes bandwidth; Other index modes may lower the EDMA performance. Odd index has worst performance. If index is power of 2, and it is larger than 8, the performance degradation is very small.

Figure 9 shows the index effect on EDMA throughput, transferring 1024 rows (BCNT= 1024) of 2D data form SL2 to DDR, with different index on 1GHz C6678 EVM with 64-bit 1333M DDR.

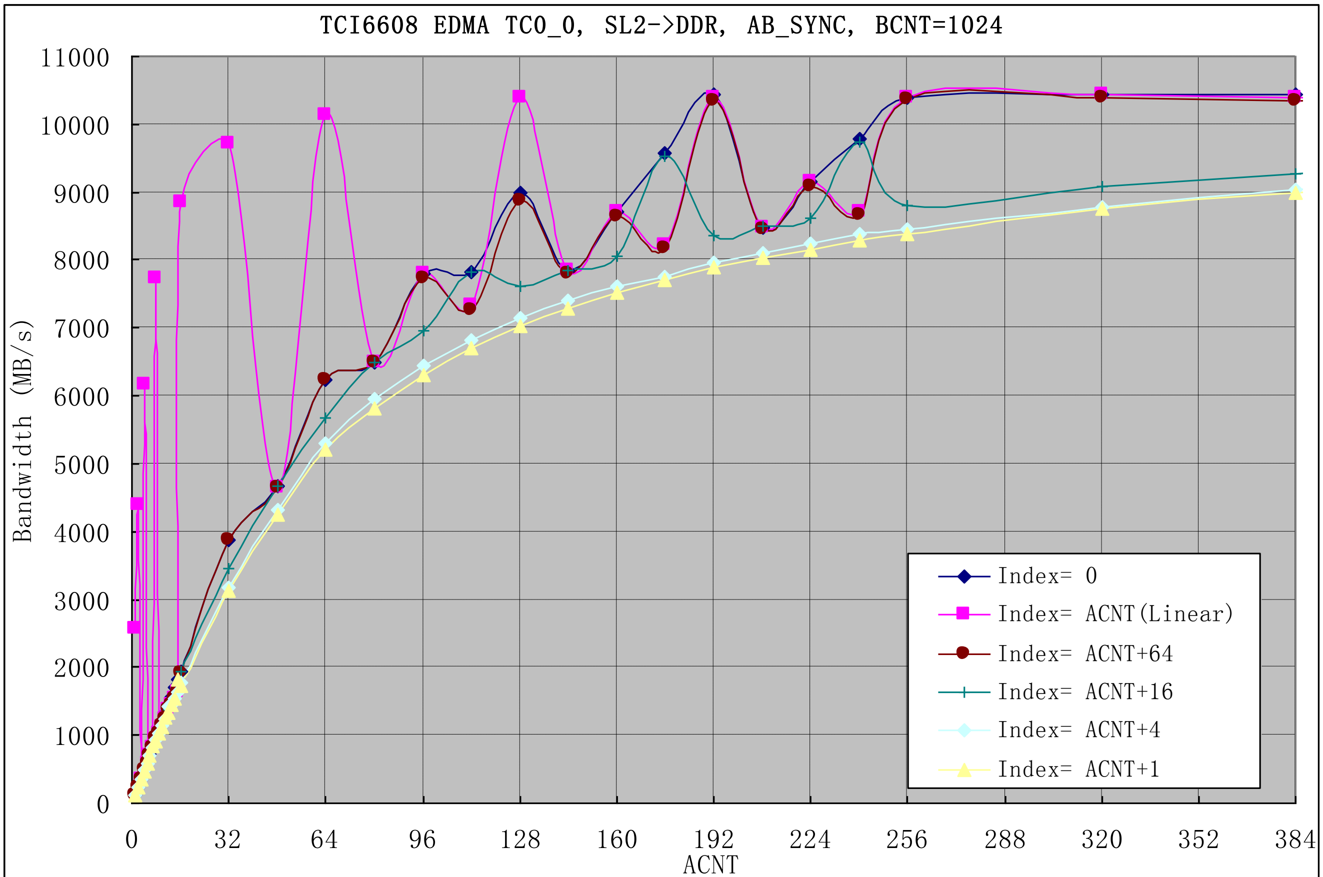


Figure 9. Index effect on EDMA bandwidth

Please note, for Index= ACNT, and ACNT is a power of 2, the 2D transfer will is optimized as 1D transfer, thus achieve much better performance.

Without special note, all performance data in this application report is measured with Index= ACNT.

Address Alignment

Address alignment may slightly impact the performance. The default burst size of EDMA3 is 64bytes or 128 bytes, if the transfer across the 64 bytes boundary, then the EDMA3 TC breaks the ACNT array into 64-bytes burst to the source/destination addresses. So, if the source or destination address is not align to 64 bytes boundary, and the transfer across 64 bytes boundary, extra burst will be generated to handle the unaligned head and tail data.

For big transfer this overhead may be ignored. All data presented in this document are based on the address aligned transfer.

Performance of Multiple masters sharing memory

Since the C6678 include 8 cores and multiple DMA masters, they may access memory in parallel. This section discusses the performance of multiple masters sharing memory.

Performance of Multiple masters sharing SL2

The data on SL2 is organized as following.

Bank 0			Bank 1			Bank 2			Bank 3		
byte 0	…	byte 31	byte 64	…	byte 95	byte 128	…	byte 159	byte 192	…	byte 223
byte 32	…	byte 63	byte 96	…	byte 127	byte 160	…	byte 191	byte 224	…	byte 255
byte 256	…	byte 287	byte 320	…	byte 351						
byte 288	…	byte 319	byte 352	…	byte 383						
…	…	…	…	…	…	…	…	…	…	…	…

Figure 10. Data organization on SL2 banks

All masters can access the 4 SL2 banks through MSMC (Multicore Shared Memory Controller), which is an X-BAR switch. Multiple masters can access different banks in parallel; if multiple masters access same bank, it is arbitrated based on priority.

Following tables show the performance data measured when multiple masters access SL2 simultaneously on 1GHz C6678. Every master accesses its own data buffer on the SL2 repeatedly, total data bytes transferred by each master in same time period (about 2 seconds) are counted; the bandwidth each master achieved is calculated by taking total bytes copied and dividing it by the time period.

L1D cache size is set to 32KB for every core, L2 cached is not used, prefetch buffer is enabled.

In following tables, each column represents a test case, different test case use different number of masters to access memory simultaneously, the blank cell in a column means corresponding master is not used for that case, and the number in a cell means the bandwidth the corresponding master achieved in that case. The last row is the total bandwidth achieved by all masters for that case.

Table 8. Performance of Multiple DSP cores sharing SL2

SL2->LL2 bandwidth(MB/s), same priority											
Core0	2249	2249	2249	2249	2249	2249					
Core1		2243	2243	2243	2242	2242					
Core2			2249	2249	2249	2249	2249				
Core3				2242	2242	2241	2242				
Core4					2249	2248	2249	2250			
Core5					2241	2241	2242	2243	2243		
Core6						2248	2249	2249	2249	2250	
Core7						2240	2241	2242	2243	2243	2243
Total	**2265**	**4508**	**6757**	**8999**	**13488**	**17974**	**13472**	**8984**	**6735**	**4493**	**2243**
LL2->SL2 bandwidth(MB/s), same priority											
Core0	3636	3636	3636	3636	3636	3636					
Core1		3635	3635	3635	3633	3615					
Core2			3635	3635	3633	3615	3634				
Core3				3635	3633	3615	3634				
Core4					3633	3615	3634	3635			
Core5					3633	3615	3634	3635	3635		
Core6						3615	3634	3635	3635	3636	
Core7						3615	3634	3635	3635	3636	3636
Total	**3677**	**7312**	**10947**	**14582**	**21839**	**28964**	**21804**	**14540**	**10905**	**7272**	**3636**

Above data proves the SL2 is NOT the bottle neck for multiple core access SL2 simultaneously. The SL2 has enough bandwidth (500M x 32 x 4 = 64000MB/s) to support multiple cores access simultaneously. The throughput limitation is on DSP core itself.

Since the SL2 bandwidth is enough for multiple core access, the priority of different core is not important for these cases.

Table 9. Performance of Multiple EDMA sharing SL2

Bandwidth(MB/s), DDR->SL2 by DMA0, LL2->SL2 by other DMA, same priority												
DMA0 TC0		9942	5245	4011	2670	2003	1996					
DMA0 TC1			5245	4011	2670	2003	1996					
DMA1 TC0				3963	2645	1989	1336	1788				
DMA1 TC1				3963	2645	1989	692	938				
DMA1 TC2					2645	1989	1983	2642	2672			
DMA1 TC3					2645	1989	1982	2639	2672			
DMA2 TC0						1983	1977	2629	2644	3973		
DMA2 TC1						1983	1976	2628	2644	3973		
DMA2 TC2							673	904	2644	3973	5064	
DMA2 TC3							1317	1755	2644	3973	5064	5064
total		**9942**	**10490**	**15948**	**15920**	**15928**	**15928**	**15923**	**15920**	**15892**	**10128**	**5064**
Bandwidth(MB/s), SL2->DDR by DMA0, SL2->LL2 by other DMA, same priority												
DMA0 TC0		10388	5247	4526	3184	2436	2431					
DMA0 TC1			5247	4526	3184	2436	2431					
DMA1 TC0				4526	3184	2455	1245	1804				
DMA1 TC1				2263	1592	1231	628	916				
DMA1 TC2					3184	2454	2451	3513	3554			
DMA1 TC3					1592	1229	1230	1770	1787			
DMA2 TC0						2436	2436	3489	3502	5086		
DMA2 TC1						1226	1229	1763	1791	2719		
DMA2 TC2							623	903	1789	2714	4885	
DMA2 TC3							1214	1749	3440	5027	4885	5066
total		**10388**	**10494**	**15841**	**15920**	**15903**	**15918**	**15907**	**15863**	**15546**	**9770**	**5066**
Bandwidth(MB/s), DDR->SL2 by DMA0, LL2->SL2 by other DMA, different priority												
DMA0 TC0	0	9942	9175	7628	7701	7701	6893					
DMA0 TC1	1		1320	2793	2728	2728	1910					
DMA1 TC0	2			4023	4013	4012	3449	3527				
DMA1 TC1	3			1412	1378	1379	626	1881				
DMA1 TC2	4				102	102	2	5069	5098			
DMA1 TC3	5				5	5	0	224	664			
DMA2 TC0	6					0	0	4	71	5045		
DMA2 TC1	7					0	0	0	4	720		
DMA2 TC2	0						1789	1776	5044	5045	5064	
DMA2 TC3	1						1231	3419	5044	5045	5064	5064
total		**9942**	**10495**	**15856**	**15927**	**15927**	**15900**	**15900**	**15925**	**15855**	**10128**	**5064**

Bandwidth(MB/s), SL2->DDR by DMA0, SL2->LL2 by other DMA, different priority												
DMA0 TC0	0	10388	5497	5496	5495	5495	7164					
DMA0 TC1	1		4997	4996	4996	4996	109					
DMA1 TC0	2			5075	5074	5074	3322	3367				
DMA1 TC1	3			170	150	150	30	951				
DMA1 TC2	4				45	45	0	21	2654			
DMA1 TC3	5				0	0	0	0	123			
DMA2 TC0	6					0	0	0	9	2644		
DMA2 TC1	7					0	0	0	0	137		
DMA2 TC2	0						1921	1901	3403	3407	4921	
DMA2 TC3	1						59	1681	4707	4707	4921	5066
total		**10388**	**10494**	**15737**	**15760**	**15760**	**12605**	**7921**	**10896**	**10895**	**9842**	**5066**

Since there are 10 TCs, but there are only 8 DSP cores, in these tests, eight TCs transfer data between SL2 and eight different core's LL2, the other two TCs transfer data between SL2 and DDR.

Though SL2 has very high bandwidth, but all EDMA access it through same port on TeraNet switch fabric, the port on the switch fabric becomes the bottleneck. The theoretic bandwidth of that port is 500MHz x 32 bytes = 16000MB/s. If the priority of the EDMA is same, the bandwidth is almost equally allocated between these EDMA. While the priority is different (different priority is shown in the second column of above tables), the low priority EDMA gets less bandwidth, for these very heavy load cases, some low priority EDMA may be starved, i.e. achieve 0 bandwidth.

According to the difference between TCs list in Table 6, DMA1 TC1 and TC3, DMA2 TC1 and TC2 may get less bandwidth than other TCs in some cases even the priority is same.

Performance of Multiple masters sharing DDR

If multiple masters access DDR at same time, it is arbitrated based on priority of the masters.

The bank number of different DDR device may be different. The DDR controller on C6678 can support DDR memory with 1, 2, 4, or 8 banks. The DDR on C6678 EVM has 8 banks; the data on it is organized as following. Please note, the row size for different DDR device may be different.

	Bank0	**Bank1**	**…**	**Bank7**
Row 0	byte 0~8191	byte 8192~8192*2-1	…	byte 8192*7~8192*8-1
Row 1	byte 8192*8~8192*9-1	byte 8192*9~8192*10-1	…	byte 8192*15~8192*16-1
…	…	…	…	…

Figure 11. Data organization on DDR banks

Though DDR has multiple banks, but there is NO multiple buses connect to each bank like SL2. So, the bank number does not directly improve the throughput.

The DDR SDRAM is accessed based on row or page. Before a master accesses data in a page, it must open the page firstly, and then it can randomly access any bytes in the page. If master wants to access data in a new row in same bank, the old row must be closed firstly, and then open the new row in the same bank for access. The row switch (row close/open) operations introduce extra delay cycles, which is called row switch overhead.

Every bank can have one open row, so the DDR with 8 banks can have 8 open rows at the same time, which reduces the probability of row switch. For example, after a master opens row 0 in bank 0 for access, it can open row 1 in bank 1 without closing the row 0 in bank 0, and then the master can access both row 0 in bank 0 and row 1 in bank 1 randomly without row switch overhead.

Two data structures for test are defined to verify the effect of DDR row switch (row close/open) overhead.

	Bank 0	**Bank 1**	**Bank2**	**…**	**Bank n**	**…**
Row 0	Master 0 Access Range					
Row 1	Master 1 Access Range					
Row 2	Master 2 Access Range					
…	…					
Row n	Master n Access Range					
…	…					

Figure 12. Multiple master access different rows on same DDR bank

Above case is the worst case with maximum row switch overhead. Every master access may result in row switch.

Following case is the best case without any row switch because every master always access an open row dedicated for it.

	Bank 0	**Bank 1**	**Bank2**	**…**	**Bank n**	**…**
Row 0	Master 0 Access Range					
Row 1		Master 1 Access Range				
Row 2			Master 2 Access Range			

…				…		
Row n					Master n Access Range	
…	…					…

Figure 13. Multiple master access different rows on different DDR banks

Performance of Multiple DSP cores sharing DDR

Following table shows the Performance of Multiple DSP cores sharing the 64-bit 1333M DDR on 1GHz C6678 EVM under different scenarios. Every master accesses its own data buffer on the DDR repeatedly. Total data bytes transferred by each master in same time period (about 2 seconds) are counted; the bandwidth each master achieved is calculated by taking total bytes copied and dividing it by the time period.

The DDR is cacheable and prefetchable for this test, and the L1D cache is set to 32KB, L2 cache size is set to 256KB, prefetch buffer is enabled. Non-cacheable case is not measured because it demands much less bandwidth than cacheable case.

In following tables, each column represents a test case, different test case use different number of masters to access memory simultaneously, the blank cell in a column means corresponding master is not used for that case, and the number in a cell means the bandwidth the corresponding master achieved in that case. The last row is the total bandwidth achieved by all masters for that case.

Table 10. Performance of Multiple DSP cores sharing DDR

DDR->LL2 bandwidth(MB/s), different row in different bank, same priority										
Core0		1631	1626	1606	1536	1288				
Core1			1602	1584	1524	1283				
Core2				1584	1524	1283	1529			
Core3				1584	1524	1283	1525			
Core4					1524	1283	1525	1597		
Core5					1524	1283	1525	1585		
Core6						1283	1525	1585	1619	
Core7						1283	1525	1585	1604	1626
Total		**1631**	**3228**	**6358**	**9156**	**10269**	**9154**	**6352**	**3223**	**1626**
DDR->LL2 bandwidth(MB/s), different row in same bank, same priority										
Core0		1632	1419	1060	720	539				
Core1			1412	1057	719	539				
Core2				1057	719	539	719			
Core3				1057	719	539	719			

Core4					719	539	719	1068		
Core5					719	539	719	1068		
Core6						539	719	1068	1357	
Core7						539	719	1068	1357	1626
Total		**1632**	**2831**	**4231**	**4315**	**4312**	**4314**	**4272**	**2714**	**1626**
LL2->DDR bandwidth(MB/s), different row in different bank, same priority										
Core0		1858	1844	1816	1643	1294				
Core1			1821	1796	1633	1290				
Core2				1796	1633	1290	1651			
Core3				1796	1633	1290	1650			
Core4					1633	1290	1650	1800		
Core5					1633	1290	1650	1791		
Core6						1290	1650	1791	1836	
Core7						1290	1650	1791	1828	1848
Total		**1858**	**3665**	**7204**	**9808**	**10324**	**9901**	**7173**	**3664**	**1848**
LL2->DDR bandwidth(MB/s), different row in same bank, same priority										
Core0		1858	1602	1079	719	539				
Core1			1593	1079	719	539				
Core2				1079	719	539	719			
Core3				1079	719	539	719			
Core4					719	539	719	1078		
Core5					719	539	719	1078		
Core6						539	719	1078	1516	
Core7						539	719	1078	1516	1848
Total		**1858**	**3195**	**4316**	**4314**	**4312**	**4314**	**4312**	**3032**	**1848**
DDR->LL2, different row in different bank, different priority										
Core0	0	1631	1628	1623	1611	1571				
Core1	1		1600	1594	1580	1535				
Core2	2			1582	1562	1479	1602			
Core3	3			1551	1521	1394	1580			
Core4	4				1433	1247	1561	1614		
Core5	5				1272	1076	1518	1594		
Core6	6					943	1426	1582	1621	
Core7	7					827	1266	1550	1600	1626
Total		**1631**	**3228**	**6350**	**8979**	**10072**	**8953**	**6340**	**3221**	**1626**
LL2->DDR, different row in different bank, different priority										
Core0	0	1858	1851	1844	1825	1731				
Core1	1		1811	1804	1778	1681				

Core2	2			1792	1741	1588	1813			
Core3	3			1707	1620	1397	1779			
Core4	4				1428	1159	1737	1833		
Core5	5				1180	983	1617	1806		
Core6	6					867	1427	1792	1841	
Core7	7					798	1181	1705	1813	1848
Total		**1858**	**3662**	**7147**	**9572**	**10204**	**9554**	**7136**	**3654**	**1848**

The performance of multiple cores access different rows on ***same*** DDR bank is worse than the performance of multiple cores access different rows on ***different*** DDR banks, the reason is the DDR row switch overhead.

Above table shows the DDR bandwidth (1333 x 8 = 10666MB/s) is NOT enough for all DSP cores access simultaneously, the priority of different core affects the bandwidth allocation between these cores. When the priority is same, the bandwidth is allocated between cores almost equally; while priority is different (different priority is shown in the second column of above tables), the low priority core gets lower bandwidth.

The DDR controller has the feature to momentarily raise the priority of oldest request to avoid starvation. There are configurable counters, which specifies Number of DDR3CLKOUT cycles after which DDR3 controller momentarily raises the priority of oldest request. Without special note, all tests for this application report are done with this counter to be 4x16=64. In 64 DDR3CLKOUT cycles, 64x2x8=1024 bytes can be transferred, that is, after transfer of 1024 bytes, DDR controller may raise the priority of oldest request.

Following tables show the performance of Multicore sharing the 64-bit 1333M DDR on 1GHz C6678 EVM with different priority raise count. Different priority is shown in the second column of the tables.

Table 11. Effect of DDR priority raise count

LL2->DDR, different bank, different priority, priority raise count = 0								
Core0	0	1709	1614	1433	1253			
Core1	1	1694	1604	1428	1249	1434		
Core2	2	1694	1604	1428	1249	1434	1615	
Core3	3	1694	1604	1428	1249	1434	1615	1709
Core4	4	1694	1604	1428	1249	1434	1615	1709
Core5	5		1604	1428	1249	1434	1615	1709
Core6	6			1428	1249	1434	1615	1709
Core7	7				1249	1434	1615	1709
Total		**8485**	**9634**	**10001**	**9996**	**10038**	**9690**	**8545**
LL2->DDR, different bank, different priority, priority raise count = 64								
Core0	0	1838	1825	1802	1731			
Core1	1	1795	1778	1748	1681	1789		
Core2	2	1777	1741	1685	1588	1746	1813	

Core3	3	1684	1620	1529	1397	1668	1779	1827
Core4	4	1512	1428	1298	1159	1516	1737	1797
Core5	5		1180	1085	983	1294	1617	1775
Core6	6			914	867	1089	1427	1675
Core7	7				798	924	1181	1503
Total		**8606**	**9572**	**10061**	**10204**	**10026**	**9554**	**8577**
LL2->DDR, different bank, different priority, priority raise count = 4080								
Core0	0	1843	1847	1848	1821			
Core1	1	1803	1806	1808	1792	1837		
Core2	2	1787	1788	1789	1773	1811	1836	
Core3	3	1706	1699	1699	1719	1786	1810	1833
Core4	4	1456	1430	1418	1442	1699	1786	1805
Core5	5		968	962	970	1420	1697	1786
Core6	6			516	520	967	1424	1700
Core7	7				214	518	969	1435
Total		**8595**	**9538**	**10040**	**10251**	**10038**	**9522**	**8559**

According to above result, priority raise count=0 actually disable the priority scheme. The bigger the count, the priority scheme makes more difference. So, for real application, designer may chose a value between 0 and the maximum (4080) for the count according to his application requirement.

Performance of Multiple EDMA sharing DDR

Following table shows the Performance of Multiple EDMA TCs sharing 64-bit 1333M DDR on 1GHz C6678 EVM under different conditions.

Table 12. Performance of Multiple EDMA sharing DDR

SL2->DDR by DMA0, LL2->DDR by other DMA, different row different bank, same priority												
DMA0 TC0		10383	5247	2631	1757	1317	1123					
DMA0 TC1			5247	2630	1757	1317	1123					
DMA1 TC0				2616	1745	1311	751	1174				
DMA1 TC1				2616	1745	1311	382	601				
DMA1 TC2					1745	1311	1119	1743	1757			
DMA1 TC3					1745	1311	1119	1742	1757			
DMA2 TC0						1308	1117	1739	1745	2623		
DMA2 TC1						1308	1117	1737	1745	2623		
DMA2 TC2							377	592	1745	2623	5019	
DMA2 TC3							746	1164	1745	2623	5019	5060

total		**10383**	**10494**	**10493**	**10494**	**10494**	**8974**	**10492**	**10494**	**10492**	**10038**	**5060**
SL2->DDR by DMA0, LL2->DDR by other DMA, different row same bank, same priority												
DMA0 TC0		10388	3693	1280	652	398	356					
DMA0 TC1			3693	1280	652	397	356					
DMA1 TC0				1275	650	397	237	360				
DMA1 TC1				1275	650	397	119	181				
DMA1 TC2					650	397	355	538	653			
DMA1 TC3					650	397	355	538	653			
DMA2 TC0						397	355	538	651	1283		
DMA2 TC1						397	355	538	651	1283		
DMA2 TC2							119	181	651	1283	2728	
DMA2 TC3							237	359	651	1283	3895	5063
total		**10388**	**7386**	**5110**	**3904**	**3177**	**2844**	**3233**	**3910**	**5132**	**6623**	**5063**
DDR->SL2 by DMA0, DDR->LL2 by other DMA, different row different bank, same priority												
DMA0 TC0		9942	5245	3161	2106	1621	1610					
DMA0 TC1			5245	3160	2105	1621	1610					
DMA1 TC0				2607	2097	1613	816	1183				
DMA1 TC1				1574	1048	809	410	596				
DMA1 TC2					2097	1612	1602	2312	2328			
DMA1 TC3					1048	808	803	1172	1188			
DMA2 TC0						1607	1597	2303	2312	3226		
DMA2 TC1						809	803	1170	1185	2016		
DMA2 TC2							402	591	1184	2016	3414	
DMA2 TC3							799	1172	2304	3225	4990	5005
total		**9942**	**10490**	**10502**	**10501**	**10500**	**10452**	**10499**	**10501**	**10483**	**8404**	**5005**
DDR->SL2 by DMA0, DDR->LL2 by other DMA, different row same bank, same priority												
DMA0 TC0		9942	3926	2264	1032	640	475					
DMA0 TC1			3926	2264	1032	640	475					
DMA1 TC0				1422	1022	638	238	558				
DMA1 TC1				991	517	320	119	280				
DMA1 TC2					1020	638	474	1034	1129			
DMA1 TC3					516	320	237	563	648			
DMA2 TC0						637	473	1029	1116	1811		
DMA2 TC1						320	237	562	647	1451		
DMA2 TC2							119	279	647	1450	2710	
DMA2 TC3							237	555	1130	1803	3610	5005
total		**9942**	**7852**	**6941**	**5139**	**4153**	**3084**	**4860**	**5317**	**6515**	**6320**	**5005**
SL2->DDR by DMA0, LL2->DDR by other DMA, different row different bank, different priority												

DMA0 TC0	0	10383	5496	4147	4140	4140	3551					
DMA0 TC1	1		4997	4147	4140	4140	1499					
DMA1 TC0	2			2122	2104	2104	2050	2572				
DMA1 TC1	3			78	90	90	435	1322				
DMA1 TC2	4				18	18	0	2630	2670			
DMA1 TC3	5				0	0	0	83	220			
DMA2 TC0	6					0	0	1	23	2697		
DMA2 TC1	7					0	0	0	0	261		
DMA2 TC2	0						1046	1316	3767	3769	5019	
DMA2 TC3	1						870	2506	3754	3708	5019	5060
total		**10383**	**10493**	**10494**	**10492**	**10492**	**9451**	**10430**	**10434**	**10435**	**10038**	**5060**
DDR->SL2 by DMA0, DDR->LL2 by other DMA, different row different bank, different priority												
DMA0 TC0	0	9942	9175	5785	5726	5726	5051					
DMA0 TC1	1		1320	4338	4288	4288	1904					
DMA1 TC0	2			311	347	347	1368	2346				
DMA1 TC1	3			68	80	80	321	1012				
DMA1 TC2	4				57	57	26	1814	1879			
DMA1 TC3	5				2	2	0	1047	1195			
DMA2 TC0	6					0	0	565	1348	2175		
DMA2 TC1	7					0	0	9	87	1394		
DMA2 TC2	0						844	1310	2422	2903	3544	
DMA2 TC3	1						979	2359	3523	3993	4962	5005
total		**9942**	**10495**	**10502**	**10500**	**10500**	**10493**	**10462**	**10454**	**10465**	**8506**	**5005**

Since there are 10 TCs, but there are only 8 DSP cores, in these tests, eight TCs transfer data between DDR and eight different core's LL2, the other two TCs transfer data between DDR and SL2.

Above table shows the DDR has NO enough bandwidth to support all EDMA TCs access it simultaneously. So, priority affects the bandwidth allocation between EDMAs. The low priority EDMA gets less bandwidth, for these very heavy load cases, some low priority EDMA may be starved, i.e. achieve 0 bandwidth.

According to the difference between TCs list in Table 6, DMA1 TC1 and TC3, DMA2 TC1 and TC2 may get less bandwidth than other TCs in some cases even the priority is same.

The performance of multiple EDMA TCs access different rows on ***same*** DDR bank is much worse than the performance of multiple EDMA TCs access different rows on ***different*** DDR banks, the reason is the DDR row switch overhead. The result becomes worse when DDR load becomes heavy. The worst case is multiple EDMA TCs write to different rows on same DDR bank, which is almost dominated by the row switch overhead because every write burst result in row switch.

The probability of row switch, i.e. multiple master access ***same*** DDR bank depends on the master number and DDR bank number. For example, if 4 EDMA randomly access DDR memory, the probability of at least two TCs access same DDR bank is:

$$1 - C_8^4 P_4^4 / 8^4 = 59\%$$

Following table list the probability for different combinations of master number and bank number.

Table 13. Probability of multiple masters access same DDR bank

	2 masters	**4 masters**	**6 masters**	**8 masters**	**10 masters**
8 banks	12.5%	59%	92.3%	99.7%	100%

The DDR controller on C6678 is optimized to alleviate the row switch overhead. An access to an open row may be given priority over the access to a closed row.

References

1. *TMS320C66x DSP CorePac User Guide* (SPRUGW0)
2. *KeyStone Architecture Multicore Shared Memory Controller (MSMC) User Guide (*SPRUGW7*)*
3. *KeyStone Architecture DDR3 Memory Controller User Guide (*SPRUGV8*)*
4. *KeyStone Architecture Enhanced Direct Memory Access (EDMA3) Controller User Guide (*SPRUGS5*)*
5. *TMS320C6678 data manual (*SPRS623*)*