Application Report
SPRAC21–June 2016

TDA2xx and TDA2ex Performance

ABSTRACT

This application report provides information on the TDA2xx and TDA2ex device throughput performances and describes the TDA2xx and TDA2ex System-on-Chip (SoC) architecture, data path infrastructure, and constraints that affect the throughput and different optimization techniques for optimum system performance. This document also provides information on the maximum possible throughput performance of different peripherals on the SoC.

Contents

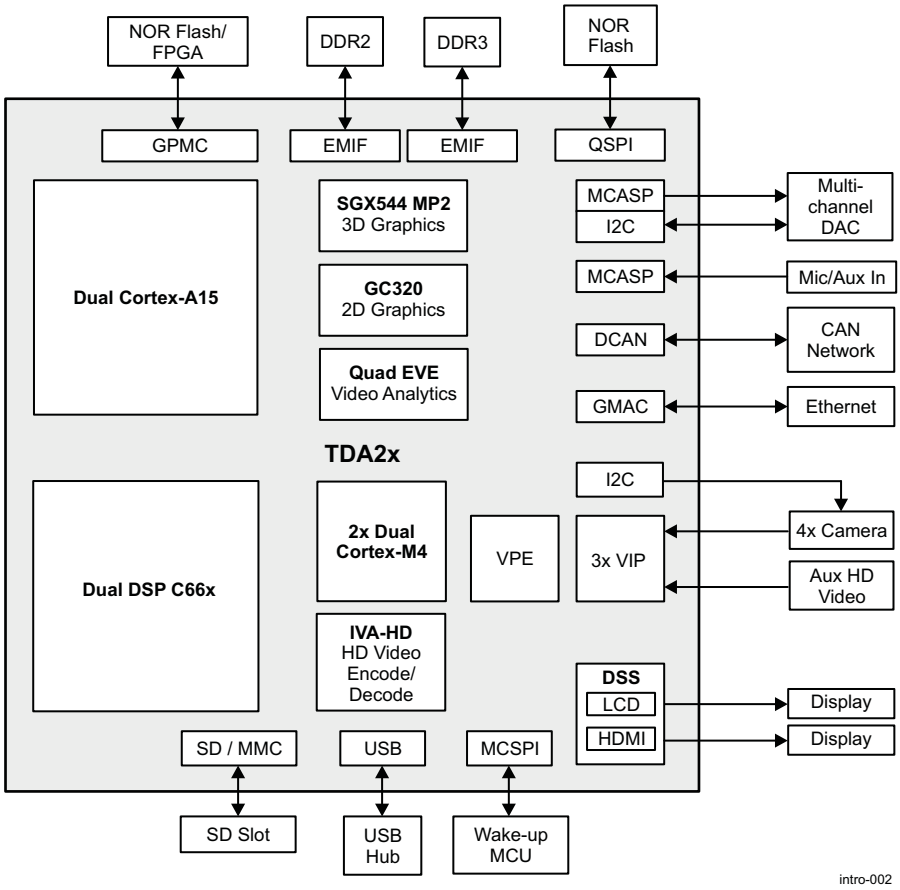
List of Figures

List of Tables

1 SoC Overview

1.1 Introduction

TDA2xx and TDA2ex are high-performance, infotainment-application devices, based on the enhanced OMAP™ architecture integrated on a 28-nm technology. The architecture is designed for advanced graphical HMI and Navigation, Digital and Analog Radio, Rear Seat Entertainment and Multimedia playback, providing Advanced Driver Assistance integration capabilities with Video analytics support, and best-in-class CPU performance, video, image, and graphics processing sufficient to support, among others. Figure 1 shows high-level block diagrams of the processors, interfaces, and peripherals and their various operating frequencies.



A The speeds shown are the highest design targeted, non-binned OPP.

Figure 1. TDA2xx Device Block Diagram

1.2 Acronyms and Definitions

Table 1. Acronyms and Definitions

Term	Definition
AESCTR	Advanced Encryption Standard Counter Mode
AESECB	Advanced Encryption Standard Electronic Code Book
BL	Bandwidth Limiter (in L3)
BW	Bandwidth (measured in megabytes per second (MB/s))
BWR	Bandwidth Regulator (in L3)
CBC-MAC	Cipher Block Chaining-Message Authentication Code
CCM	CBC-MAC + CTR
CPU/MPU	Cortex®-A15
DDR	Double Data Rate
DVFS	Dynamic Voltage and Frequency Scaling
EVE	Embedded Vision Engine
GCM	Galois Counter Mode
GMAC	Galois Message Authentication Code
LPDDR	Low-Power Double Data Rate
NVM	Non-Volatile Memory
OCP	Open Core Protocol
OTFA	On-The-Fly Advanced Encryption Standard
PHY	Hard macro that converts single data rate signals to double data rate
QSPI	Quad Serial Peripheral Interface
ROM	On-Chip ROM Bootloader
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System On-Chip
Stat Coll	Statistics collector in interconnect

1.3 TDA2xx and TDA2ex System Interconnect

The system's interconnect and master to slave connection in the TDA2xx and TDA2ex devices is shown in Figure 2. For those initiators and peripherals not supported by TDA2ex, the slave and master ports to L3 are tied to a default value resulting in errors when trying to access a peripheral not present in the device.

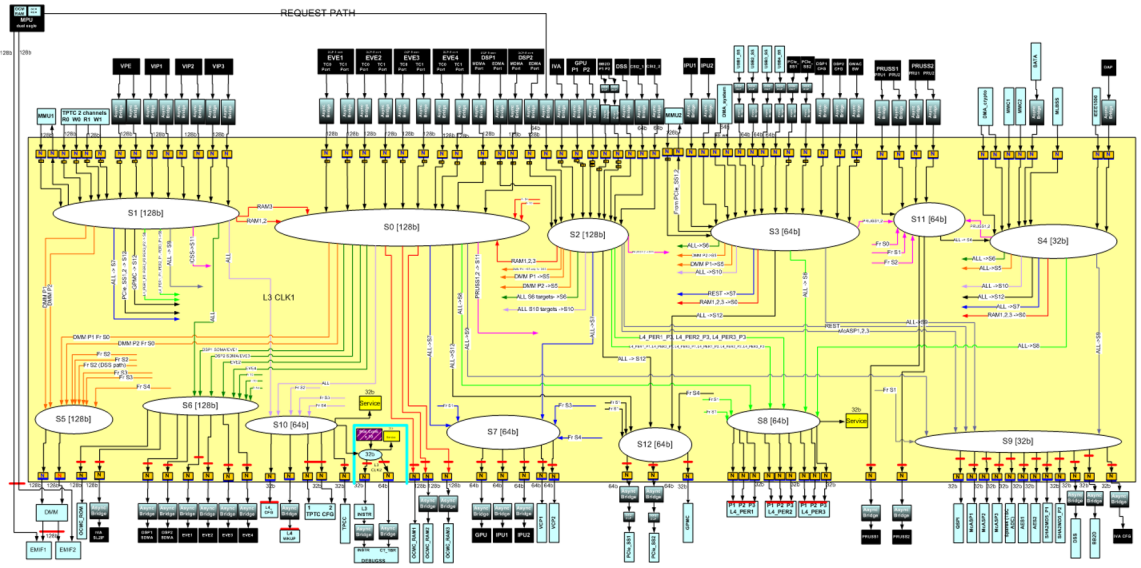


Figure 2. TDA2xx and TDA2ex SoC Interconnect Diagram

For more information on the L3 interconnect, see the *Interconnect* chapter in the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5) and the *TDA2Ex SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.0 Technical Reference Manual* (SPRUI00). Broadly, the list of masters and slaves in the system are listed in Table 2 and Table 3.

Table 2. List of Master Ports in TDA2xx and TDA2ex

Master	Supported Maximum Tag Number	Maximum Burst Size (Bytes)	Type
MPU	32	120	RW
CS_DAP	1	4	RW
IEEE1500_2_OCP	1	4	RW
DMA_SYSTEM RD	4	128	RO
MMU1	33	128	RW
DMA_CRYPTO RD	4	124	RO
DMA_CRYPTO WR	2	124	WR
TC1_EDMA RD	32	128	RO
TC2_EDMA RD	32	128	RO
TC1_EDMA WR	32	128	WR
TC2_EDMA WR	32	128	WR
DMA_SYSTEM WR	2	128	WR
DSP1 CFG	33	128	RW
DSP1 DMA	33	128	RW
DSP1 MDMA	33	128	RW
DSP2 CFG (1)	33	128	RW
DSP2 DMA (1)	33	128	RW
DSP2 MDMA (1)	33	128	RW

Table 2. List of Master Ports in TDA2xx and TDA2ex (continued)

Master	Supported Maximum Tag Number	Maximum Burst Size (Bytes)	Type
CSI2_1	1	128	WR
IPU1	8	56	RW
IPU2	8	56	RW
EVE1 P1 (1)	17	128	RW
EVE1 P2 (1)	17	128	RW
EVE2 P1 (1)	17	128	RW
EVE2 P2 (1)	17	128	RW
EVE3 P1 (1)	17	128	RW
EVE3 P2 (1)	17	128	RW
EVE4 P1 (1)	17	128	RW
EVE4 P2 (1)	17	128	RW
PRUSS1 PRU1	2	128	RW
PRUSS1 PRU2	2	128	RW
PRUSS2 PRU1	2	128	RW
PRUSS2 PRU2	2	128	RW
GMAC_SW	2	128	RW
SATA	2	128	RW
MMC1	1	124	RW
MMC2	1	124	RW
USB3_SS	32	128	RW
USB2_SS	32	128	RW
USB2_ULPI_SS1	32	128	RW
USB2_ULPI_SS2	32	128	RW
GPU P1	16	128	RW
MLB	1	124	RW
PCIe_SS1	16	128	RW
PCIe_SS2	16	128	RW
MMU2	33	128	RW
VIP1 P1	16	128	RW
VIP1 P2	16	128	RW
VIP2 P1	16	128	RW
VIP2 P2	16	128	RW
VIP3 P1	16	128	RW
VIP3 P2	16	128	RW
DSS	16	128	RW
GPU P2	16	128	RW
GRPX2D P1	32	128	RW
GRPX2D P2	32	128	RW
VPE P1	16	128	RW
VPE P2	16	128	RW
IVA	16	128	RW

1. Not present in TDA2ex.

Table 3. List of L3 Slaves in TDA2xx and TDA2ex

Slave	Tag Number	Maximum Burst Size (Bytes)
GPMC	1	124
GPU	1	8
IVA1 SL2IF	1	16
OCMC_RAM1	1	128
DSS	1	124
IVA1 CONFIG	1	124
IPU1	1	56
AES1	1	4
AES2	1	4
SHA2MD5_1	1	4
DMM P1	32	128
DMM P2	32	128
L4_WKUP	1	124
IPU2	1	56
OCMC_RAM2 (1)	1	128
OCMC_RAM3 (1)	1	128
DSP1 SDMA	1	128
DSP2 SDMA (1)	1	128
OCMC_ROM	1	16
TPCC_EDMA	1	128
PCIe SS1	1	120
VCP1	1	128
L3_INSTR	1	128
DEBUGSS CT_TBR	1	128
QSPI	256	128
VCP2	1	128
TC1_EDMA	1	128
TC2_EDMA	1	128
McASP1	1	128
McASP2	1	128
McASP3	1	128
PCIe SS2	1	120
SPARE_TSC_ADC	1	128
GRPX2D	1	4
EVE1 (1)	16	128
EVE2 (1)	16	128
EVE3 (1)	16	128
EVE4 (1)	16	128
PRUSS1	1	128
PRUSS2	1	128
MMU 1	32	128
MMU 2	32	128
SHA2MD5_2	1	4
L4_CFG	1	124
L4_PER1 P1	1	124
L4_PER1 P2	1	124
L4_PER1 P3	1	124

Table 3. List of L3 Slaves in TDA2xx and TDA2ex (continued)

Slave	Tag Number	Maximum Burst Size (Bytes)
L4_PER2 P1	1	124
L4_PER2 P2	1	124
L4_PER2 P3	1	124
L4_PER3 P1	1	124
L4_PER3 P2	1	124
L4_PER3 P3	1	124

1. Not present in TDA2ex.

The L3 high-performance interconnect is based on a Network-On-Chip (NoC) interconnect infrastructure. The NoC uses an internal packet-based protocol for forward (read command, write command with data payload) and backward (read response with data payload, write response) transactions. All exposed interfaces of this NoC interconnect, both for Targets and Initiators; comply with the OCP IP2.x reference standard.

1.4 Traffic Regulation Within the Interconnect

The interconnect has internal components that can aid in the traffic regulation from a specific initiator to a specific target. The components are called Bandwidth Regulators and Bandwidth limiters. Additionally, the initiator IPs can set their respective MFLAG or MREQPRIORITY signals, which is understood by the interconnect and subsequently DMM/EMIF to give priority to a given initiator.

The default value of the various traffic regulator within the interconnect is set to a default that allows most use case to work without any tweaking. However, if there is any customization needed for a given use case, it is possible by various programmable parameters explained in subsequent sections.

1.4.1 Bandwidth Regulators

The bandwidth regulators prevent master NIUs from consuming too much bandwidth of a link or a slave NIU that is shared between several data flows: packets are then transported at a slower rate. The value of a bandwidth can be programmed in the bandwidth regulator. When the bandwidth is below the programmed value, the pressure bit is set to 1, giving priority to this master. When the bandwidth is above the programmed value, the pressure bit is set to 0 and the concerned master has the same weight as others.

Bandwidth regulators are by default enabled in interconnect with the default configuration such that the expected average bandwidth is set to zero. With any amount of traffic, the actual average bandwidth seen will be greater than zero and, hence, lower pressure bits (00b) will be enabled by default. You need to program the regulators to achieve the desired regulation in case of concurrences. If set, the bandwidth regulator discards the L3 initiator priority using the device control module.

Bandwidth regulators:

- Regulates the traffic through priority; does not really stop traffic.
- Sets low-priority, by default. On setting a valid bandwidth other than 0, the initiator is given higher priority when the bandwidth is lower than the threshold.
- Does not set an upper limit, If there is no contention, the initiator can claim full share as well.

Bandwidth regulator available for the following IPs:

- MMU2
- EVE1, EVE2, EVE3, EVE4 – both TC0 and TC1
- DSP1, DSP2 MDMA (CPU access port)
- DSP1, DSP2 EDMA
- IVA
- GPU

- GMAC
- PCIe

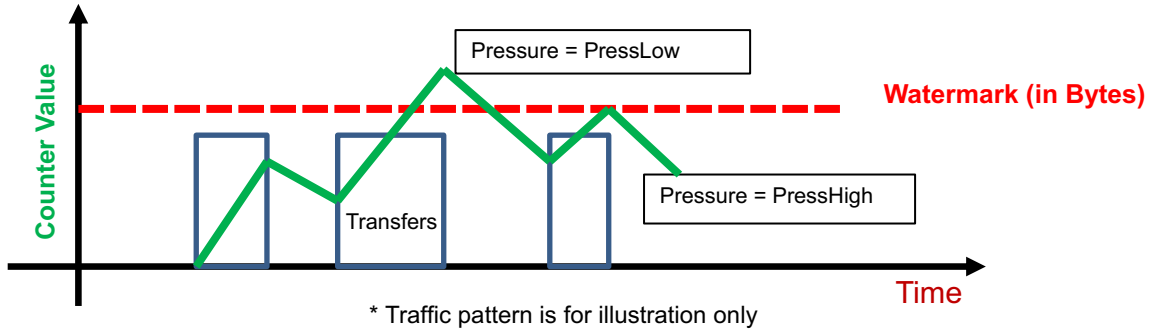


Figure 3. TDA2xx and TDA2ex Bandwidth Regulator Mechanism Illustration

Programming API for the bandwidth regulator is:

```
set_bw_regulator(int port, unsigned int average_bw, unsigned int time_in_us
    {
     unsigned int base_address = get_bw_reg_base_address(port);
     WR_REG32(base_address+0x8,(int)(ceil(average_bw/8.3125));
     WR_REG32(base_address+0xC,(time_in_us*average_bw));
     WR(REG32(base_address+0x14,0x1);
     }

get_bw_reg_base_address(port) {
     if ( port == "EVE1_TC0" ) { return
L3_NOC_AVATAR__DEBUGSS_CS_DAP_INIT_OCP_L3_NOC_AVATAR_CLK1_EVE1_TC0_BW_REGULATOR;
     }
...
    }
```

For more details on the programming of bandwidth regulators and limiters, see the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

1.4.2 Bandwidth Limiters

The bandwidth limiter regulates the packet flow in the L3_MAIN interconnect by applying flow control when a user-defined bandwidth limit is reached. The next packet is served only after an internal timer expires, thus ensuring that traffic does not exceed the allocated bandwidth. The bandwidth limiter can be used with a watermark mechanism that allows traffic to temporarily exceed the peak bandwidth.

Bandwidth limiter:

- Limits the bandwidth by flow control (actually stops traffic)
- Enabled by default not to cap the maximum bandwidth of any initiator. Hence, by default, it allows the maximum bandwidth that the CPU can generate without any other initiators. The upper limit can be set by programming the Bandwidth Limiter (BL) appropriately to the desired maximum bandwidth.
- Actually sets an upper limit allowing for some maximum water mark.

Programming API for the bandwidth limiter is:

```
set_bw_limiter(port,limit-bw) {
    base_address = get_bw_limiter_base_address(port);
    bandwidth = int(limit_bw / 8.3125);
    bandwidth_int = ( bandwidth & 0xFFFFFFE0 ) >> 5;
    bandwidth_frac = ( bandwidth & 0x1F );
    WR_REG32(base_address+0x8,bandwidth_frac);
    WR_REG32(base_address+0xC,bandwidth_int);
    WR_REG32(base_address+0x10,0x0);
    WR_REG32(base_address+0x14,0x1);
    }
get_bw_limiter_base_address(port) {
    if ( port == "VPE_P2" ) { return
L3_NOC_AVATAR__DEBUGSS_CS_DAP_INIT_OCP_L3_NOC_AVATAR_CLK1_VPE_P2_BW_LIMITER;
    }
...

   }
```

For more details on the programming of bandwidth regulators and limiters, see the *TDA2x Technical Reference Manual* (SPRUHK5).

1.4.3 Initiator Priority

Certain initiators in the system can generate MFLAG signals that provide higher priority to the data traffic initiated by them. The modules that can generate the MFLAG dynamically are VIP, DSS, EVE, and DSP. Following is a brief discussion of the DSS MFLAG.

- DSS MFLAG
  - DSS has four display read pipes (Graphics , Vid1, Vid2, and Vid3) and one write pipe (WB).
  - DSS drives MFLAG if any of the read pipes are made high priority and FIFO levels are below low threshold for high-priority display pipe.
  - VIDx have 32 KB FIFO and GFX has 16 KB FIFO.
  - FIFO threshold is measured in terms of 16-byte word.
  - Recommended settings for high and low threshold are 75% and 50%, respectively.
  - MFLAG can be driven high permanently through a force MFLAG configuration of the DISPC_GLOBAL_MFLAG_ATTRIBUTE register.

The behavior of setting the MFLAG dynamically can be realized using Figure 4.

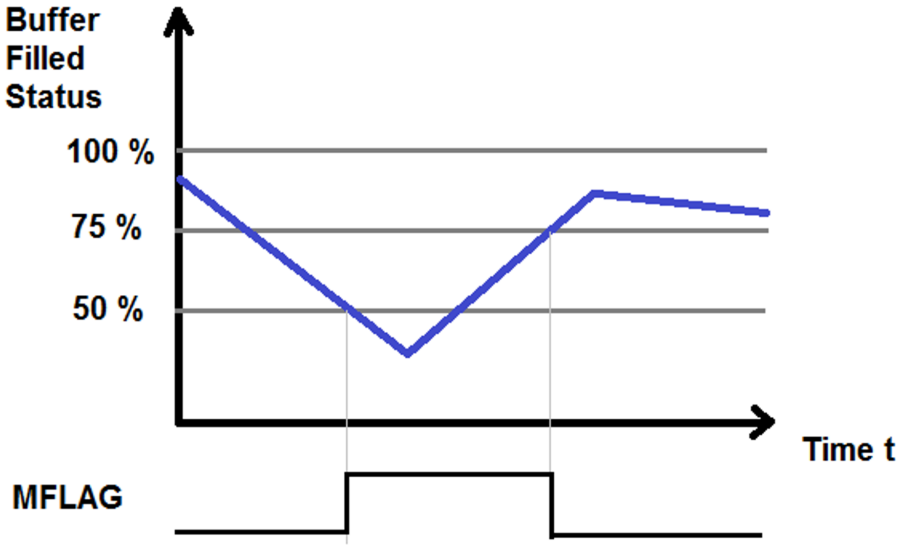


Figure 4. TDA2xx and TDA2ex DSS Adaptive MFLAG Illustration

The programming model used to enable dynamic MFLAG is:

```
Enable MFlag Generation DISPC_GLOBAL_MFLAG_ATTRIBUTE
DISPC_GLOBAL_MFLAG_ATTRIBUTE = 0x2;
Set Video Pipe as High Priority DISPC_VIDx_ATTRIBUTES
DISPC_VID1_ATTRIBUTES | = (1<<23);
DISPC_VID2_ATTRIBUTES | = (1<<23);
DISPC_VID3_ATTRIBUTES | = (1<<23);
Set Graphics Pipe as High Priority DISPC_GFX_ATTRIBUTES
DISPC_GFX_ATTRIBUTES | = (1<<14);

GFX threshold 75 % HT , 50 % LT
DISPC_GFX_MFLAG_THRESHOLD =  0x03000200;
VIDx threshold 75 % HT , 50 % LT
DISPC_VID1_MFLAG_THRESHOLD =  0x06000400;
DISPC_VID2_MFLAG_THRESHOLD =  0x06000400;
DISPC_VID3_MFLAG_THRESHOLD =  0x06000400;
```

- DSP EDMA + MDMA
  - EVTOUT[31] and EVTOUT[30] are used for generation of MFLAGs dedicated to the DSP MDMA and EDMA ports, respectively.
  - EVTOUT[31/30] = 1 → Corresponding MFLAG is high.
- EVE TC0/TC1
  - For EVE port 1 and port 2 (EVE TC0 and TC1), MFlag is driven by evex_gpout[63] and evex_gpout[62], respectively.
  - evex_gpout[63] is connected to DMM_P1 and EMIF.
  - evex_gpout[62] is connected to DMM_P2 and EMIF.
- VIP/VPE
  - In the VIP/VPE Data Packet Descriptor Word 3, can set the priority in [11:9] bits.
  - This value is mapped to OCP Reqinfo bits.
  - 0x0 = Highest Priority, 0x7 = Lowest Priority.
  - VIP Has Dynamic MFLAG specific scheme based on internal FIFO status
    - Based on HW set margins to overflow/underflow
    - Enabled by default, no MMR control

Many other IPs have their MFLAG driving mechanism via the control module registers.

The CTRL_CORE_L3_INITIATOR_PRESSURE_1 to CTRL_CORE_L3_INITIATOR_PRESSURE_4 registers are used for controlling the priority of certain initiators on the L3_MAIN.

- 0x3 = Highest Priority/Pressure
- 0x0 = Lowest Priority/Pressure
- Valid for MPU, DSP1, DSP2, IPU1, PRUSS1, GPU P1, GPU P2

There are SDRAM initiator priorities that control the priority of each initiator accessing two EMIFs. The CTRL_CORE_EMIF_INITIATOR_PRIORITY_1 to CTRL_CORE_EMIF_INITIATOR_PRIORITY_6 registers are intended to control the priority of each initiator accessing the two EMIFs. Each 3-bit field in these registers is associated only with one initiator. Setting this bit field to 0x0 means that the corresponding initiator has a highest priority over the others and setting the bit field to 0x7 is for lowest priority. This feature is useful in case of concurrent access to the external SDRAM from several initiators. For more information regarding the SDRAM Initiator priority registers, see the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

In the context of TDA2xx and TDA2ex, the CTRL_CORE_EMIF_INITIATOR_PRIORITY_1 to CTRL_CORE_EMIF_INITIATOR_PRIORITY_6 are overridden by the DMM PEG Priority and, hence, it is recommended to set the DMM PEG priority instead of the Control module EMIF_INITIATOR_PRIORITY registers.

The MFLAG influences the priority of the Traffic packets at multiple stages:

- At the interconnect level, the NTTP packet is configured with one bit of pressure. This bit, when set to 1, gives priority to the concerned packet across all arbitration points. This bit is set to 0 for all masters. The pressure bit can be set to 1 either using the bandwidth regulators (within L3) or can be directly driven by masters using OCP MFlag. MFLAG asserted pressure is embedded in the packet while pressure from the BW regulator is a handshake signal b/w the BW regulator and the switch.
- At the DMM level, the MFLAG is used to drive the DMM Emergency mechanism. At the DMM, the initiators with MFLAG set will be classified as higher priority. A weighted round-robin algorithm is used for arbitration between high priority and other initiators. Set DMM_EMERGENCY[0] to run this arbitration scheme. The weight is set in the DMM_EMERGENCY[20:16] WEIGHT field.
- At the EMIF level, the MFLAG from all of the system initiators are ORed to have higher priority to the system traffic versus the MPU traffic when any system initiator has the MFLAG set.

1.5 TDA2xx and TDA2ex Memory Subsystem

The different memory subsystems in TDA2xx and TDA2ex and their interconnection is as shown in Figure 5.

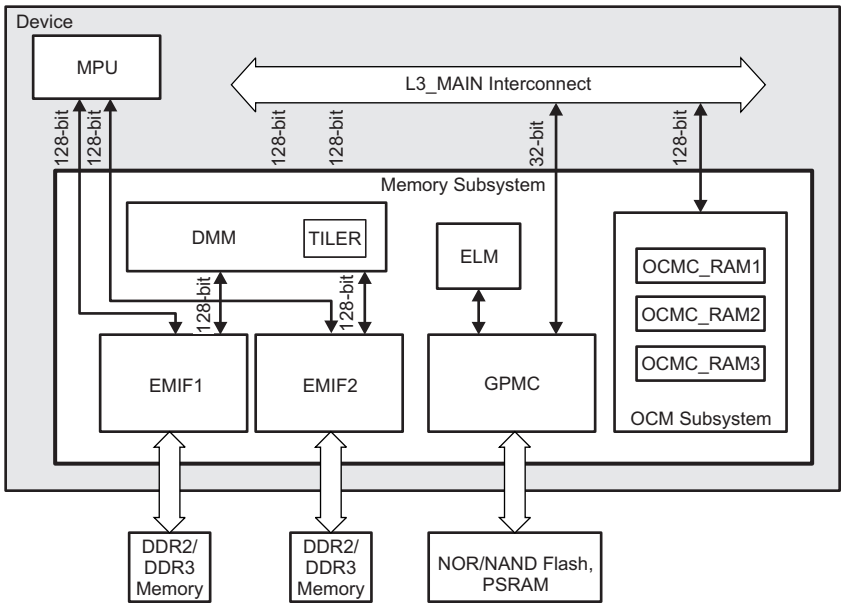
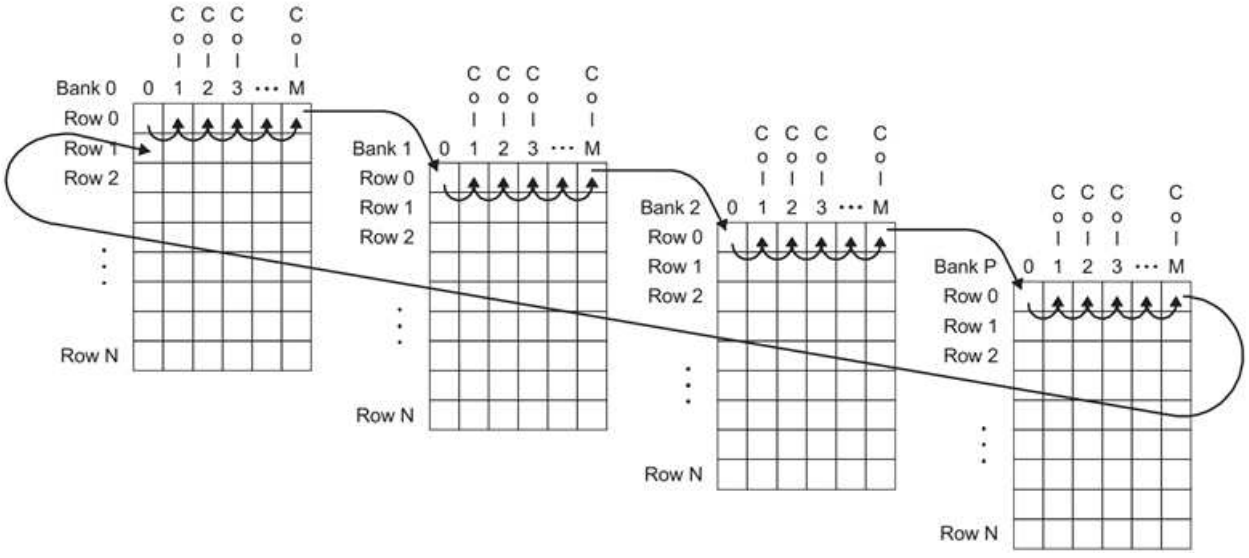


Figure 5. TDA2xx and TDA2ex Memory Subsystem Interconnection

NOTE: TDA2ex does not contain EMIF2, OCMC RAM 2 and 3.

The EMIF is the memory controller of the system's main memory (DRAM). Supported DRAM types are DDR2 (400 MHz), DDR3 (532 MHz), TDA2ex specifically supports DDR3 (666 MHz) and so on. This external memory controller module supports 32-bit, 16-bit mode (narrow mode), up to 2KB page size, up to 8 banks, 128 byte burst. The bank distribution and the row, column and bank access pattern is as shown in Figure 6.



A M is the number of columns (as determined by PAGESIZE) minus 1, P is the number of banks (as determined by IBANK) minus 1, and N is the number of rows (as determined by both PAGESIZE and IBANK) minus 1.

Figure 6. DDR Row, Column and Bank Access

All the measurements done in this report use the following configurations, unless mentioned otherwise.

- DDR3 Memory Part Number: MT41K128M16 16 Meg × 16 × 8 banks
- CAS write latency = 6
- CAS Latency = 7
- SDRAM Data Bus width: 32
- DDR in non-interleaved mode.
- Number of banks (P + 1) = 8
- Number of Columns (M + 1) = 2^{10}
- Number of Rows (N + 1) = 2^{14}
- Size of each DDR cell = 16 bits
- Page size is 1024 cells. This makes the effective page size = (1024 × 16 bits) × 2 = 32768 bits = 4096 Bytes = 4KB.

1.5.1 Controller/PHY Timing Parameters

Based on the DRAM data sheet, the respective timing values specified in time need to be translated into clock cycles and programmed into the MEMSS MMR. Also, parameters like refresh rate DRAM topology, CAS latencies are also programmed in these MMR. The MEMSS PHY interface also has a MMR space that needs to be configured to enable the PHY and program parameters like read latency.

1.5.2 Class of Service

For priority escalation, there is one more level of escalation that can be enabled inside MEMSS using class of service with eight priority levels. There are two classes of service (CoS) implemented in MEMSS. Each class of service can support up to three master connection IDs. Each of the three masters can be associated with the same or different priority levels. Each class of service has a priority raise counter that can be set to N, which implies N times 16 clocks. This counter value specifies the number of clocks after which MEMSS momentarily raises the priority of the class of service command (master connections are arbitrated using priority levels with in a class of service). Using masks along with master connection IDs, a maximum of 144 master connection IDs can participate in this class of service. For details on Class of Service, see the *TDA2x Technical Reference Manual* (SPRUHK5).

NOTE:

- Priority raise counter is also available to momentarily raise the priority of the oldest command in the command FIFO. This counter can be set to N, which implies N times 16 clocks.
- Read and write execution thresholds register can be programmed to a maximum burst size after which the MEMSS arbitration will switch to executing the other type of commands (set write threshold to provide a chance for read commands to execute).
- DRAM page open entails cost; so if you avoid multiple page closes and opens, you can obtain better performance.

1.5.3 Prioritization Between DMM/SYS PORT or MPU Port to EMIF

Each EMIF has two ports in the TDA2xx and TDA2ex devices. EMIF OCP configuration by default gives equal priority to values set in the system and MPU cmd FIFO. In case the priority of these cmd FIFO needs to be changed, use EMIF_OCP_CONFIG register settings as explained in the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

1.6 TDA2xx and TDA2ex Measurement Operating Frequencies

The sections that follow report the system bandwidth for the various system initiators based on the subsystem operating frequencies in Table 4, unless mentioned otherwise.

Table 4. TDA2xx and TDA2ex Subsystem Operating Frequencies

IP/Core/Memory	TDA2xx and TDA2ex Si Frequency (MHz)	Bus Width (bits)
Cortex-A15	1000	128
EDMA TPTC+TPTC	266	128
L3	266	128
OCMC RAM	266	128
EMIF	266	128
DDR	532	32
DSP SS	600	128
IVAHD SL2	388.33	128
ICSS RAM	200	32
M4 - L2 RAM	212.8	128
EVE	535	128
Timer	20	-

1.7 System Instrumentation and Measurement Methodology

1.7.1 GP Timers

The GP Timers are good tools to have a quick measurement of performance of different initiators in the system. Though this provides a quick way to measure throughput, it should be kept in mind that there are some inherent errors involved with this technique of measurement. To make sure the measurement error is relatively small the transfer size should be kept larger. The typical pseudo code for the GP Timer-based throughput measurement is as follows:

```
/*Get the Timer Start Stamp*/
timerStartStamp = timerRead(TIMER_NUM);

/*Start the transfer*/
transferStart();

/*Wait for Transfer Completion*/
waitForTransferCompletion();

/*Get the Time Stamp*/
timerEndStamp = timerRead(TIMER_NUM);

/* Calculate the bandwidth - Note that one needs to multiply x2 from the calculated value to get
* Bandwidth when the source and destination memory are the same       */
if (timerEndStamp > timerStartStamp)
    BW = (((float) (TRANSFER_SIZE)/ (float) (timerEndStamp - timerStartStamp)) * TIMER_FREQ);
else
    BW = (((float) (TRANSFER_SIZE)/ (float) (timerEndStamp -
 timerStartStamp + 0xFFFFFFFF)) * TIMER_FREQ);
```

The formulas used for the throughput calculations are:

Actual Throughput = (Transfer Size/Time Taken)

Ideal Throughput = Frequency of Limiting Port × Data Bus Width in Bytes

Utilization or Efficiency = (Actual Throughput/Ideal Throughput) × 100

Consider the data is transferred from one memory M1 to another memory M2.

The interconnect signals that implement OCP protocol used to transfer the data between the two memories, are as shown in Figure 7.

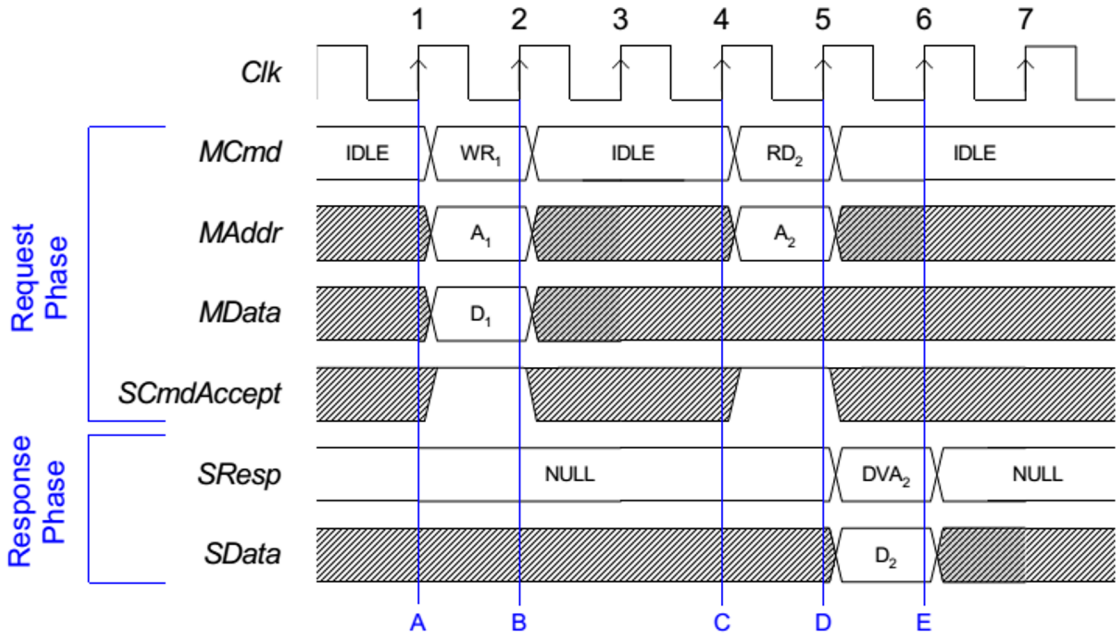


Figure 7. OCP Interconnect Signals

There are two data lanes:

- MData that contains the data to be written
- SData that contains the data to be read

For further details on the exact operation of the OCP signals, see the OCP specifications.

Due to the presence of two data lanes, the case when the theoretical maximum throughput is obtained is when the MData contains valid data to be written and SData contains valid data to be read on every clock cycle of the transaction.

Mathematically stating this:

if Source Mem = Destination Mem, $Ideal\ Throughput_{mem2mem} = 2 \times Frequency \times Bus\ Width$

if Source Mem ≠ Destination Mem, $Ideal\ Throughput_{mem2mem} = Frequency \times Bus\ Width$

Hence, when the source and destination memories are the same, the measured bandwidth should be multiplied into 2.

1.7.2 L3 Statistic Collectors

The interconnect in TDA2xx and TDA2ex has a special component called statistics collector, see Figure 8, that computes traffic statistics within a user-defined window and periodically reports the results in a memory-mapped register or DEBUGSS interface. TDA2xx and TDA2ex has 10 statistic collectors and are dedicated for load/latency monitoring.

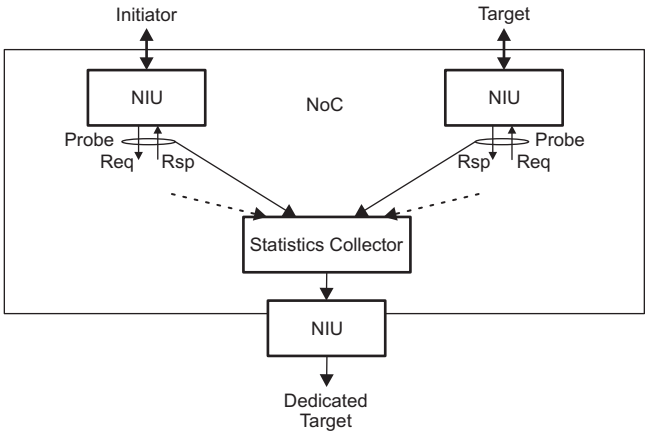


Figure 8. L3 Statistic Collectors Basic Infrastructure

The key features of the statistic collector are:

- Nonintrusive monitoring
- Programmable filters and counters
- Collects results at a programmable time interval

Event detectors are programmed through the L3_STCOL_REQEVT and L3_STCOL_RSPEVT configuration registers for request and response ports, respectively. The following events can be identified:

- Word transfer
- WAIT cycles
- Flow control
- Payload transfers
- Latency measurements

Performance monitoring is enabled through the L3_STCOL_EN register. The L3_STCOL_SOFTEN register enables software to monitor the performance. Event muxes are programmed through the L3_STCOL_EVTMUX_SEL0 configuration register that determines which port will be monitored by a filter configured by the filter registers.

Filters are programmed through the L3_STCOL_FILTER_i_GLOBALEN configuration register, along with additional selection criteria programmed through the mask and match registers. A filter can be configured to accept or reject:

- Read operations
- Write operations
- Errors
- Addresses

Filter operation is programmed through the L3_STCOL_OP registers.

There are ten statistic collectors used to monitor the traffic on DRAM (EMIF1, EMIF2, MA_MPU_P1 and MA_MPU_P2), MPU, MMU, TPTC, VIP, VPE, EVE Subsystem, DSP MDMA/EDMA, IVA, GPU, BB2D, DSS, IPU, OCMC RAM, USB, PCIe Subsystem, DSP CFG, MMC, SATA, VCP, GPMC, and McASP ports.

For more details, see the *L3* and *On-Chip Debug Support* sections in the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

The L3 statistic collectors provide a more accurate measurement of the bandwidth of different initiators. This provides a very good visualization of peaks in the traffic profile of different initiators in the system.

The formulas and concepts for bandwidth measurement when source and destination memories are the same is described in Section 1.7.1.

Bandwidth profiles can be captured using L3 statistic collectors through software executing on the target by using the following steps:

1. Capture the number of bytes transferred every ‘x’ µs at the initiator ports (read + write) or the destination memory.
2. Use a timer to maintain ‘x’ µs time gap.
3. The number of bytes obtained is divided by ‘x’ µs to get the average BW within the ‘x’ µs.
4. The process is repeated multiple times to generate a bandwidth profile.

As an example of this measurement technique, the DSS bandwidth profile is captured using x = 100 µs as shown in Figure 9.

As can be seen in Figure 9, with the bandwidth profile you can clearly see the peak bandwidth that corresponds to the DSS pre-fetch period at the beginning of the frame, the gaps in the profile that correspond to the period DSS is displaying data from internal buffers and not requesting for new data. The steady-state bandwidth corresponds to the real-time traffic.

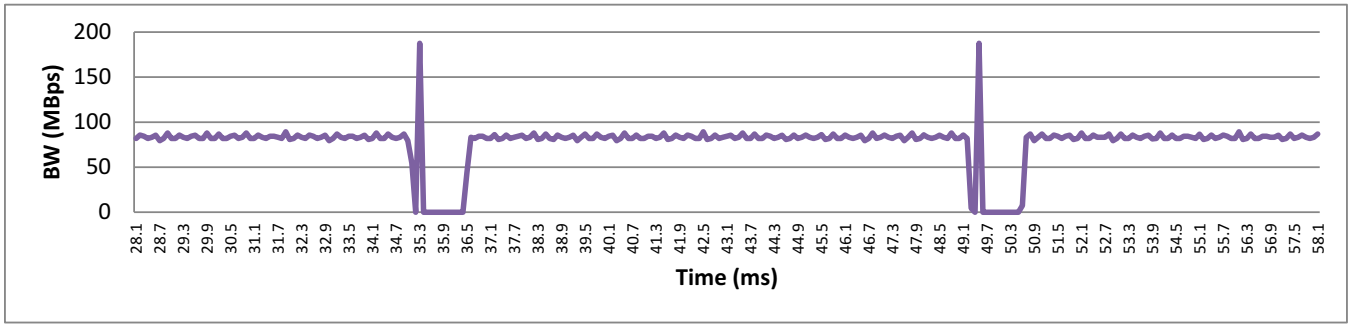


Figure 9. DSS Single VID-Single VENC 720×480 70 fps RGB

2 Cortex-A15

The ARM® Cortex®-A15 (also referred to as CPU/MPU) is a 32-bit RISC microprocessor with NEON™ SIMD coprocessor and can be clocked up to 1.5 GHz. The Cortex-A15 (ARMv7 architecture based) has an MMU, 32KB of level1 instruction and data cache, and up to 4MB configurable level2 cache. For more details on the Cortex-A15, see the Cortex®-A15 Technical Reference Manual: r2p0.

2.1 Level1 and Level2 Cache

The Level1 instruction and data cache are of 32KB each with 4-way set associativity and a cache line size of 32 bytes.

The L2 memory system consists of a tightly-coupled L2 cache and an integrated Snoop Control Unit (SCU), connecting up to four processors within a Cortex-A15 MP Core device. The L2 memory system has the following features:

- Configurable L2 cache size of 512KB, 1MB, 2MB, and 4MB.
- Fixed line length of 64 bytes.
- Physically indexed and tagged cache.
- 16-way set-associative cache structure.

NOTE: Using the level1 and level2 cache significantly improves the performance of A15 by many folds as compared to when the caches are disabled.

For more details on A15 level1 and level2 cache, see the Cortex®-A15 Technical Reference Manual: r2p0 and the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

2.2 MMU

MMU is a memory management unit that works with level1 and level2 cache memories to translate virtual addresses to physical addresses. The translation applies for any accesses to and from the main memory. The MMU in Cortex-A15 supports page table entries of 4 KB, 64 KB, 1 MB, and 16 MB. MMU supports the following features:

- 32-entry fully-associative L1 instruction TLB
- Two separate 32-entry fully associative L1 TLBs for data load and store pipelines
- 4-way set-associative 512-entry L2 TLB in each processor
- Intermediate table walk caches
- The TLB entries contain a global indicator or an Address Space Identifier (ASID) to permit context switches without TLB flushes.
- The TLB entries contain a Virtual Machine Identifier (VMID) to permit virtual machine switches without TLB flushes.
- The default replacement policy is pseudo round-robin

For more details on A15 MMU, see the Cortex®-A15 Technical Reference Manual: r2p0.

2.3 Performance Control Mechanisms

The Cortex A15 subsystem has various performance control mechanisms at various levels of the code and data access path to the main memory (DRAM). Some of the key control mechanisms are explained in detail in this section.

2.3.1 Cortex-A15 Knobs

Table 5 shows a few of the Cortex-A15 knobs available via CP15 programming and other special registers.

Table 5. Cortex-A15 CPU Settings

Number	Cortex-A15 CPU Settings
1	L1 I Cache enabled
2	L1 D Cache enabled
3	L2 Cache enabled
4	Branch Prediction Enabled
5	MMU Enabled
6	L2 Prefetch Enabled. Prefetch offset set to maximum (3 cache lines for instruction and 8 for data).
7	Write Streaming thresholds

2.3.2 MMU Page Table Knobs

MMU supports memory accesses based on memory sections or pages that are essentially formulated as the page table. The translation tables held in memory could have two levels. The Page table has descriptors that define the attributes and characteristics of the memory (for example, 4KB size page) for all of the pages in the 4 GB map for a 32-bit address range. TEX [2:0], C, B are key bit fields in the page descriptor that can be programmed to tweak the following knobs.

- Bufferable (B), Cacheable (C), and Type Extension (TEX) settings
- Outer cache policy (L2) – defined by TEX[1:0]
- Write back Write allocate
- Write back Write no allocate
- Write through Write no allocate
- Non-cacheable
- Inner cache policy (L1)
- Write back Write allocate
- Write back Write no allocate
- Write through Write no allocate
- Non-cacheable
- Bufferable and cacheable

For more details on the MMU Page attributes, see chapter B3 of the *ARM Architectural Reference Manual - ARMv7-A and ARMv7-R edition* http://infocenter.arm.com/help/topic/com.arm.doc.ddi0406c/index.html.

2.4 Cortex-A15 CPU Read and Write Performance

This section describes the Cortex-A15 CPU Read and Write performance obtained on the TDA2xx and TDA2ex platform. The results described correspond to OPP100. Write back Write Allocate (WB-WA) is the default configuration used for all performance measurement. Linux operating system has moved from WB-WA to WB-Write no allocate after ARMv7A. A default L3 initiator priority is used (00b) for all initiators. MMU page table is 16KB size with 1MB section granularity.

2.4.1 Cortex-A15 Functions

Following are the three functions used for Cortex-A15 operations:

The C code for the write function is:

```
void memWrite (UWORD32 DstAddr, UWORD32 transSize) {
register UWORD32 wrData=0xA5B5C5D5;
register UWORD32 i_wr;
register volatile UWORD32 *wrAddr;
wrAddr = (UWORD32*)DstAddr;
    for(i_wr=0; i_wr<transSize; i_wr+=128) {
            /*128 Words Increment*/
            *wrAddr++ =wrData;    /*Word 1*/
            *wrAddr++ =wrData;    /*Word 2*/
            *wrAddr++ =wrData;    /*Word 3*/
            *wrAddr++ =wrData;    /*Word 4*/
            *wrAddr++ =wrData;    /*Word 5*/
            ...
            ...
            ...
            *wrAddr++ =wrData;    /*Word 127*/
            *wrAddr++ =wrData;    /*Word 128*/
    }
}
```

The C code for the read function is:

```
void memRead (UWORD32 SrcAddr, UWORD32 transSize) {
register UWORD32 rdData;
register UWORD32 i_rd;
register volatile UWORD32 *rdAddr;
rdAddr = (UWORD32*)SrcAddr;
    for(i_rd=0; i_rd<transSize; i_rd+=128) {
            /*128 Words Increment*/
            *rdAddr++ =rdData;    /*Word 1*/
            *rdAddr++ =rdData;    /*Word 2*/
            *rdAddr++ =rdData;    /*Word 3*/
            *rdAddr++ =rdData;    /*Word 4*/
            *rdAddr++ =rdData;    /*Word 5*/
            ...
            ...
            ...
            *rdAddr++ =rdData;    /*Word 127*/
            *rdAddr++ =rdData;    /*Word 128*/
    }
}
```

The C code for the copy function is:

```
void memCopy (UWORD32 SrcAddr, UWORD32 DstAddr, UWORD32 trasSize) {
register volatile UWORD32 *rdAddr, *wrAddr;
register UWORD32 i;
rdAddr = (UWORD32*)SrcAddr;
wrAddr = (UWORD32*)DstAddr;
    for(i=0; i<transSize; i=i+32) {
            *wrAddr++ = *rdAddr++; /*Word 1*/
            *wrAddr++ = *rdAddr++; /*Word 2*/
            *wrAddr++ = *rdAddr++; /*Word 3*/
            *wrAddr++ = *rdAddr++; /*Word 4*/
            *wrAddr++ = *rdAddr++; /*Word 5*/
            ...
            ...
            ...
            *wrAddr++ = *rdAddr++; /*Word 30*/
            *wrAddr++ = *rdAddr++; /*Word 31*/
            *wrAddr++ = *rdAddr++; /*Word 32*/
     }
```

```
}
```

Additionally, an optimized asm copy is used that is found to have the highest memory copy performance. The parameters passed to the function are:

- R0 – Source Buffer Address
- R1 – Destination Buffer Address
- R2 – Number of Bytes to transfer

```
void memcpy_arm( UWORD32 srcBuffer, UWORD32 destBuffer, UWORD32 numBytes );

.text
.global memcpy_arm
memcpy_arm:
    CMP      r2,#3
    BLS      _my_memcpy_lastbytes
    ANDS     r12,r0,#3
    BEQ      l1
    LDRB     r3,[r1],#1
    CMP      r12,#2
    ADD      r2,r2,r12
    LDRLSB   r12,[r1],#1
    STRB     r3,[r0],#1
    LDRCCB   r3,[r1],#1
    STRLSB   r12,[r0],#1
    SUB      r2,r2,#4
    STRCCB   r3,[r0],#1
l1:
    ANDS     r3,r1,#3
    BEQ      __my_aeabi_memcpy4
l3:
    SUBS     r2,r2,#8
    BCC      l2
    LDR      r3,[r1],#4
    LDR      r12,[r1],#4
    STR      r3,[r0],#4
    STR      r12,[r0],#4
    B        l3
l2:
    ADDS     r2,r2,#4
    LDRPL    r3,[r1],#4
    STRPL    r3,[r0],#4
    MOV      r0,r0
_my_memcpy_lastbytes:
    LSLS     r2,r2,#31
    LDRCSB   r3,[r1],#1
    LDRCSB   r12,[r1],#1
    LDRMIB   r2,[r1],#1
    STRCSB   r3,[r0],#1
    STRCSB   r12,[r0],#1
    STRMIB   r2,[r0],#1
    BX       lr
__my_aeabi_memcpy4:
__my_aeabi_memcpy8:
__my_rt_memcpy_w:
    PUSH     {r4-r8,lr}
    SUBS     r2,r2,#0x20
    BCC      l4
    DSB
    PLD     [r1, #0x20]
    PLD     [r1, #0x40]
    PLD     [r1, #0x60]
    PLD     [r1, #0x80]
    PLD     [r1, #0xa0]
    PLD     [r1, #0xc0]
```

```
    PLD     [r1, #0xe0]
l5:
    PLD      [r1,#0x100]
    LDMCS    r1!,{r3-r8,r12,lr}
    SUBCSS   r2,r2,#0x20
    STMCS    r0!,{r3-r8,r12,lr}
    BCS      l5
l4:
    LSLS     r12,r2,#28
    LDMCS    r1!,{r3,r4,r12,lr}
    STMCS    r0!,{r3,r4,r12,lr}
    LDMMI    r1!,{r3,r4}
    STMMI    r0!,{r3,r4}
    POP      {r4-r8,lr}
    LSLS     r12,r2,#30
    LDRCS    r3,[r1],#4
    STRCS    r3,[r0],#4
    BXEQ     lr
_my_memcpy_lastbytes_aligned:
    LSLS     r2,r2,#31
    LDRCSH   r3,[r1],#2
    LDRMIB   r2,[r1],#1
    STRCSH   r3,[r0],#2
    STRMIB   r2,[r0],#1
    BX       lr
.end
```

2.4.2 Setup Limitations

L1 and L2 caches along with the Cortex-A15 MMU are enabled in all the measurements. Based on the write back write allocate cache policy, the net amount of reads and writes to the main memory (DDR3) are greater or lesser than the intended data size. In this case, the performance measurement is mostly based on the time taken for the intended size read, write, and copy, and not the actual data size. The GP Timer 3 is easy to use and widely used for profiling; however, this timer runs only at 20 MHz so there will be a minor difference in the accuracy. For large transfer sizes like 4MB, the difference is negligible.

2.4.3 System Performance

This section lists the standalone Cortex-A15 performance with SDRAM (DDR3) and OCMC RAM. All of the performance data is calibrated for Megabytes per second (MB/s).

2.4.3.1 Cortex-A15 Stand-Alone Memory Read, Write, Copy

Cortex-A15 settings:

- L1 cache enabled
- L2 cache enabled
- MMU enabled
- Branch Prediction enabled
- L2 Prefetch distance: Icache = 3, Dcache = 8
- Compiler Options: -O3,--opt_for_speed = 5

2.4.3.2 Results

As observed in Table 6 and Table 7, MEM copy throughput numbers are better with the GCC compiler as compared to the TI compiler flow for DDR-to-DDR transfers.

Table 6. TI Compiler Timer Based Results

Operations for Cortex-A15	Source	Destination	Transfer Size (KB)	Bandwidth with Cache Policy - WB WA (MBps)
CPU_WR	CPU Register	DDR	8192	3642
	CPU Register	OCMC	256	1392
CPU_RD	DDR	CPU Register	8192	2203
	OCMC	CPU Register	256	2173
COPY	DDR	DDR	8192	1907
	OCMC	OCMC	256	3935

Table 7. GCC Compiler with ASM Optimized Copy Results

Initiator/Operation	Source	Destination	Size (KB)	Bandwidth (MB/s)
Cortex-A15 MEM COPY	DDR	DDR	8192	2467.36

3 System Enhanced Direct Memory Access (System EDMA)

This section provides a throughput analysis of the EDMA module. The enhanced direct memory access module, also called EDMA, performs high-performance data transfers between two slave points, memories and peripheral devices without microprocessor unit (MPU) or digital signal processor (DSP) support during transfer. EDMA transfer is programmed through a logical EDMA channel, which allows the transfer to be optimally tailored to the requirements of the application. The EDMA can also perform transfers between external memories and between Device subsystems internal memories, with some performance loss caused by resource sharing between the read and write ports.

The EDMA controller block diagram is shown in Figure 10. The EDMA controller is based on two major principal blocks:

- EDMA third-party channel controller (EDMA_TPCC)
- EDMA third-party transfer controller (EDMA_TPTC)

The EDMA controller's primary purpose is to service user programmed data transfers between internal or external memory-mapped slave endpoints. It can also be configured for servicing event driven peripherals (such as serial ports), as well. There are 64 direct memory access (DMA) channels and 8 QDMA channels serviced by two concurrent physical channels.

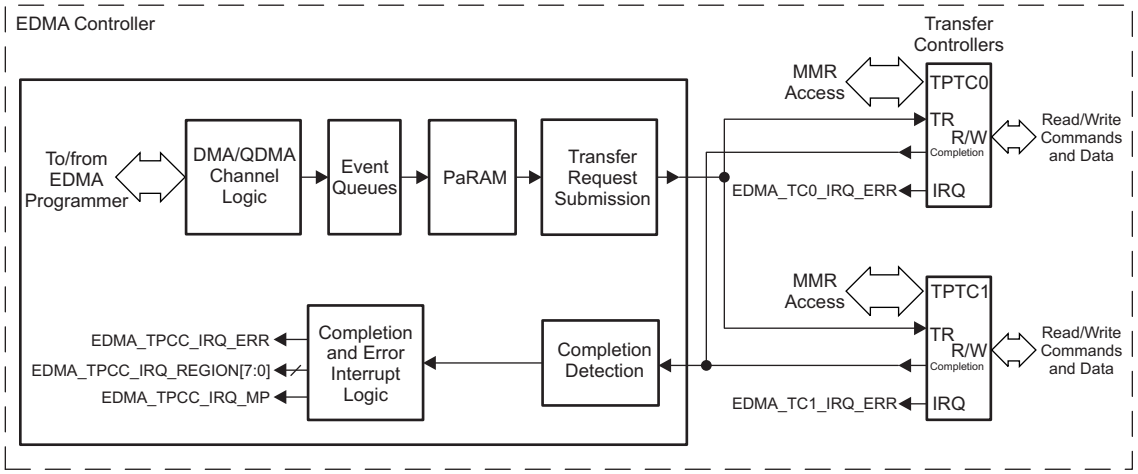


Figure 10. EDMA Controller Block Diagram

DMA channels are triggered by external event, manual write to event set register (ESR), or chained event. QDMA are auto-triggered when write is performed to the user-programmable trigger word. Once a trigger event is recognized, the event is queued in the programmed event queue. If two events are detected simultaneously, then the lowest-numbered channel has highest priority. Each event in the event queue is processed in the order it was queued. On reaching the head of the queue, the PaRAM associated with that event is read to determine the transfer details. The transfer request (TR) submission logic evaluates the validity of the TR and is submits a valid transfer request to the appropriate transfer controller. The maximum theoretical bandwidth for a given transfer can be found by multiplying the width of the interface and the frequency at which it transfers data.

The maximum speed the transfer can achieve is equal to the bandwidth of the limiting port. In general, a given transfer scenario will never achieve maximum theoretical band width due to several factors, like transfer overheads, access latency of source/destination memories, finite number of cycles taken by EDMA CC and EDMA TC between the time the transfer event is registered to the time the first read command is issued to EDMA TC. These overheads can be calibrated by looking at the time taken to do a 1 byte transfer. These factors are not excluded in these throughput measurements.

The frequency and bus widths of the different memory and slave end points are summarized in Table 8.

Table 8. Frequency and Bus Widths of Modules

Module Name	Frequency (MHz)	Bus Width (bits)
EDMA TPTC+TPTC	266	128
L3	266	128
OCMC RAM	266	128
EMIF	266	128
DSPL2 RAM	600	128
IVAHD SL2	388.33	128
ICSS RAM	200	32

3.1 System EDMA Performance

3.1.1 System EDMA Read and Write

The common system setup for the EDMA throughput measurement is:

- EMIF configuration
- CAS write latency – 6
- CAS latency - 7
- SDRAM Data Bus width : 32
- DDR in non-interleaved mode

The data presented is for stand-alone transfers with no other ongoing or competing traffic. All profiling is done with the TIMER 2 counter running at SYS_CLK2 = 22.57 MHz.

3.1.2 System EDMA Results

NOTE: When using multiple TCs transferring data to DDR it is important to understand the impact of DDR page open and close on the overall DDR efficiency.

Table 9. System EDMA 1 TC Bandwidth for Different Source Destination Combinations With GP Timer (Single TC)

No.	Source	Destination	ACNT	BCNT	Size (KB)	Ideal Throughput (MB/s)	Measured Bandwidth (MB/s)	TC Utilization (%)
1	EMIF 0 DDR	EMIF 0 DDR	65535	128	8192	4256	3274.08	76.93
2	OCMC RAM0	EMIF 0 DDR	65535	4	256	4256	3186.1	74.86
3	EMIF 0 DDR	OCMC RAM0	65535	4	256	4256	3071.96	72.18
4	OCMC RAM2	OCMC RAM2	65535	8	512	4256	3942	92.6
5	DSP L2	EMIF 0 DDR	65535	4	256	4256	3015.59	70.86
6	EMIF 0 DDR	DSP L2	65535	4	256	4256	2956.82	69.47
7	IVA SL2	EMIF 0 DDR	65535	4	256	4256	1486.21	34.92
8	EMIF 0 DDR	IVA SL2	65535	4	256	4256	1486.77	34.93
9	IVA SL2	DSP L2	65535	4	256	4256	1486.21	34.92
10	DSP L2	IVA SL2	65535	4	256	4256	1486.21	34.92
11	DSP L2	OCMC RAM	65535	4	256	4256	3017.9	70.91
12	OCMC RAM	DSP L2	65535	4	256	4256	3020.22	70.96
13	PRUSS1 IRAM	EMIF 0 DDR	1024	12	12	800	287.55	35.94
14	EMIF 0 DDR	PRUSS1 IRAM	1024	12	12	800	287.55	35.94
15	PRUSS1 DRAM0	EMIF 0 DDR	1024	8	8	800	287.33	35.92
16	EMIF 0 DDR	PRUSS1 DRAM0	1024	8	8	800	285.99	35.75
17	PRUSS1 DRAM1	EMIF 0 DDR	1024	8	8	800	286.66	35.83
18	EMIF 0 DDR	PRUSS1 DRAM1	1024	8	8	800	285.99	35.75
19	PRUSS2 IRAM	EMIF 0 DDR	1024	8	8	800	283.58	35.45
20	EMIF 0 DDR	PRUSS2 IRAM	1024	8	8	800	281.86	35.23
21	PRUSS2 DRAM0	EMIF 0 DDR	1024	12	12	800	275.76	34.47
22	EMIF 0 DDR	PRUSS2 DRAM0	1024	12	12	800	272.73	34.09
23	PRUSS2 DRAM1	EMIF 0 DDR	1024	8	8	800	313.64	39.21
24	EMIF 0 DDR	PRUSS2 DRAM1	1024	8	8	800	314.04	39.26
25	PRUSS2 DRAM2	EMIF 0 DDR	1024	32	32	800	313.84	39.23
26	EMIF 0 DDR	PRUSS2 DRAM2	1024	32	32	800	313.84	39.23

Table 10. System EDMA 1 TC Bandwidth for Different Source Destination Combinations With L3 Statistic Collectors (Single TC)

No.	Source	Destination	ACNT	BCNT	Size (KB)	Ideal Throughput (MB/s)	Measured Bandwidth (MB/s)	TC Utilization (%)
1	EMIF 0 DDR	EMIF 0 DDR	65535	128	8192	4256	3456	81
2	OCMC RAM0	EMIF 0 DDR	65535	4	256	4256	3300	78
3	EMIF 0 DDR	OCMC RAM0	65535	4	256	4256	3400	80
4	OCMC RAM2	OCMC RAM2	65535	8	512	4256	3900	92
5	DSP L2	EMIF 0 DDR	65535	4	256	4256	3070	72
6	EMIF 0 DDR	DSP L2	65535	4	256	4256	3070	72
7	IVA SL2	EMIF 0 DDR	65535	4	256	4256	1530	36
8	EMIF 0 DDR	IVA SL2	65535	4	256	4256	1530	36
9	IVA SL2	DSP L2	65535	4	256	4256	1530	36
10	DSP L2	IVA SL2	65535	4	256	4256	1530	36
11	DSP L2	OCMC RAM	65535	4	256	4256	3070	72
12	OCMC RAM	DSP L2	65535	4	256	4256	3070	72
13	PRUSS1 IRAM	EMIF 0 DDR	1024	12	12	800	380	48
14	EMIF 0 DDR	PRUSS1 IRAM	1024	12	12	800	380	48
15	PRUSS1 DRAM0	EMIF 0 DDR	1024	8	8	800	380	48
16	EMIF 0 DDR	PRUSS1 DRAM0	1024	8	8	800	380	48
17	PRUSS1 DRAM1	EMIF 0 DDR	1024	8	8	800	380	48
18	EMIF 0 DDR	PRUSS1 DRAM1	1024	8	8	800	380	48
19	PRUSS2 IRAM	EMIF 0 DDR	1024	8	8	800	380	48
20	EMIF 0 DDR	PRUSS2 IRAM	1024	8	8	800	380	48
21	PRUSS2 DRAM0	EMIF 0 DDR	1024	12	12	800	380	48
22	EMIF 0 DDR	PRUSS2 DRAM0	1024	12	12	800	380	48
23	PRUSS2 DRAM1	EMIF 0 DDR	1024	8	8	800	380	48
24	EMIF 0 DDR	PRUSS2 DRAM1	1024	8	8	800	380	48
25	PRUSS2 DRAM2	EMIF 0 DDR	1024	32	32	800	380	48
26	EMIF 0 DDR	PRUSS2 DRAM2	1024	32	32	800	380	48

Table 11. System EDMA 2 TC Bandwidth for Multiple Source Destination Combinations With GP Timer

No.	Source	Destination	ACNT	BCNT	Size (KB)	Ideal Throughput (MB/s)	Measured Bandwidth (MB/s)	TC Utilization (%)
1	EMIF 0 DDR	EMIF 0 DDR	65535	128	8192	4256	2594.2	60.95
2	OCMC RAM	EMIF 0 DDR	65535	4	256	4256	2502.79	58.81
3	EMIF 0 DDR	OCMC RAM	65535	4	256			
4	DSP L2	EMIF 0 DDR	65535	4	256	4256	2501.2	58.77
5	EMIF 0 DDR	DSP L2	65535	4	256			
6	IVA SL2	EMIF 0 DDR	65535	4	256	4256	1516.78	35.64
7	EMIF 0 DDR	IVA SL2	65535	4	256			
8	IVA SL2	DSP L2	65535	4	256	4256	1515.03	3.56
9	DSP L2	IVA SL2	65535	4	256			
10	DSP L2	OCMC RAM	65535	4	256	4256	3132.96	73.61
11	OCMC RAM	DSP L2	65535	4	256			
12	PRUSS1 IRAM	EMIF 0 DDR	1024	12	12	800	311.53	38.94
13	EMIF 0 DDR	PRUSS1 IRAM	1024	12	12			

Table 11. System EDMA 2 TC Bandwidth for Multiple Source Destination Combinations With GP Timer (continued)

No.	Source	Destination	ACNT	BCNT	Size (KB)	Ideal Throughput (MB/s)	Measured Bandwidth (MB/s)	TC Utilization (%)
14	PRUSS1 DRAM0	EMIF 0 DDR	1024	8	8	800	321.84	40.23
15	EMIF 0 DDR	PRUSS1 DRAM0	1024	8	8			
16	PRUSS1 DRAM1	EMIF 0 DDR	1024	8	8	800	321.42	40.18
17	EMIF 0 DDR	PRUSS1 DRAM1	1024	8	8			
18	PRUSS2 IRAM	EMIF 0 DDR	1024	8	8	800	305.61	38.2
19	EMIF 0 DDR	PRUSS2 IRAM	1024	8	8			
20	PRUSS2 DRAM0	EMIF 0 DDR	1024	12	12	800	307.77	38.47
21	EMIF 0 DDR	PRUSS2 DRAM0	1024	12	12			
22	PRUSS2 DRAM1	EMIF 0 DDR	1024	8	8	800	323.31	40.41
23	EMIF 0 DDR	PRUSS2 DRAM1	1024	8	8			
24	PRUSS2 DRAM2	EMIF 0 DDR	1024	32	32	800	323.21	40.4
25	EMIF 0 DDR	PRUSS2 DRAM2	1024	32	32			

Table 12. System EDMA 2 TC Bandwidth for Multiple Source Destination Combinations With L3 Statistic Collectors

No.	Source	Destination	ACNT	BCNT	Size (KB)	Ideal Throughput (MB/s)	Measured Bandwidth (MB/s)	TC Utilization (%)
1	EMIF 0 DDR	EMIF 0 DDR	65535	128	8192	4256	3050	71.66
2	OCMC RAM	EMIF 0 DDR	65535	4	256	4256	3143	73.85
3	EMIF 0 DDR	OCMC RAM	65535	4	256			
4	DSP L2	EMIF 0 DDR	65535	4	256	4256	3250	76.36
5	EMIF 0 DDR	DSP L2	65535	4	256			
6	IVA SL2	EMIF 0 DDR	65535	4	256	4256	1560	36.65
7	EMIF 0 DDR	IVA SL2	65535	4	256			
8	IVA SL2	DSP L2	65535	4	256	4256	1560	36.65
9	DSP L2	IVA SL2	65535	4	256			
10	DSP L2	OCMC RAM	65535	4	256	4256	3250	76.36
11	OCMC RAM	DSP L2	65535	4	256			
12	PRUSS1 IRAM	EMIF 0 DDR	1024	12	12	800	330	41.25
13	EMIF 0 DDR	PRUSS1 IRAM	1024	12	12			
14	PRUSS1 DRAM0	EMIF 0 DDR	1024	8	8	800	330	41.25
15	EMIF 0 DDR	PRUSS1 DRAM0	1024	8	8			
16	PRUSS1 DRAM1	EMIF 0 DDR	1024	8	8	800	330	41.25
17	EMIF 0 DDR	PRUSS1 DRAM1	1024	8	8			
18	PRUSS2 IRAM	EMIF 0 DDR	1024	8	8	800	330	41.25
19	EMIF 0 DDR	PRUSS2 IRAM	1024	8	8			
20	PRUSS2 DRAM0	EMIF 0 DDR	1024	12	12	800	330	41.25
21	EMIF 0 DDR	PRUSS2 DRAM0	1024	12	12			
22	PRUSS2 DRAM1	EMIF 0 DDR	1024	8	8	800	330	41.25
23	EMIF 0 DDR	PRUSS2 DRAM1	1024	8	8			
24	PRUSS2 DRAM2	EMIF 0 DDR	1024	32	32	800	330	41.25
25	EMIF 0 DDR	PRUSS2 DRAM2	1024	32	32			

3.2 System EDMA Observations

NOTE: On the TDA2xx and TDA2ex device, all transfer controllers yield identical performance for all transfer scenarios because both TCs have the same configuration, and most importantly the same FIFOSIZE for a given burst size.

EDMA channel parameters allow many different transfer configurations. Typical transfer configurations result in transfer controllers bursting the read write data in default burst size chunks, thereby, keeping the busses fully utilized. However, in some configurations, the TC issues less than optimally sized read and write commands (less than default burst size), reducing performance. To properly design a system, it is important to know which configurations offer the best performance for high-speed operations.

On TDA2xx and TDA2ex, there are two transfer controllers to move data between slave end points. The default configuration for the transfer controllers is shown in Table 13.

Table 13. Default Configuration for the Transfer Controllers

Name	Description	TC0	TC1
TCCFG[2:0] FIFOSIZE	Channel FIFO Size	1024 Bytes	1024 Bytes
TCCFG[5:4] BUSWIDTH	Data Transfer Bus Width	16 Bytes	16 Bytes
TCCFG[9:8] DSTREGDEPTH	Destination Register Depth	4 entries	4 entries
DBS (Default Burst Size)	Size of each data burst	Configurable	Configurable

The individual TC performance for paging and memory-to-memory transfers is essentially dictated by the TC configuration. In most scenarios, the FIFOSIZE and default burst size configuration for the TC have the most significant impact on the TC performance; the BUSWIDTH configuration is dependent on the device architecture and the DSTREGDEPTH values impact the number of in-flight transfers.

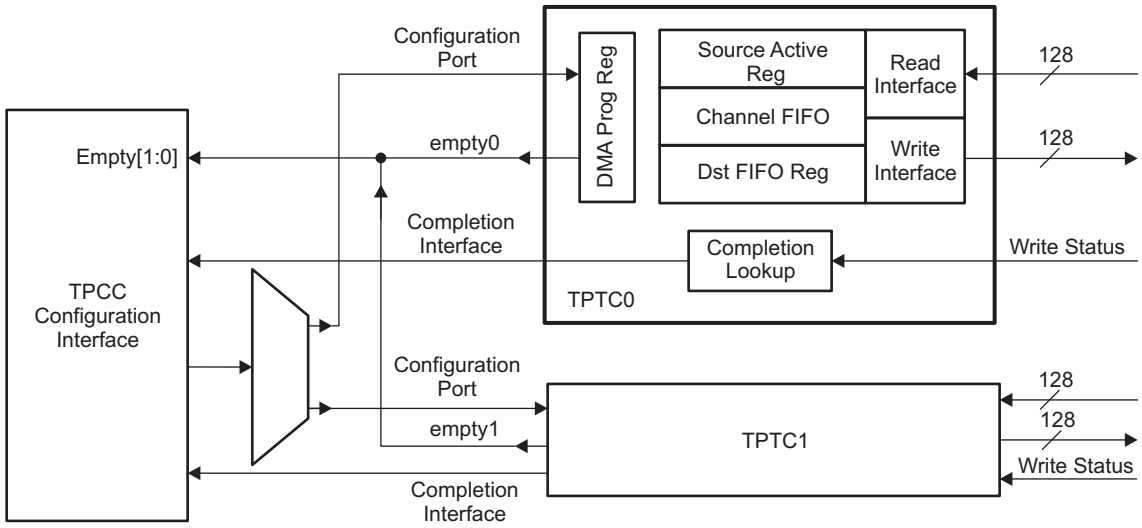


Figure 11. EDMA Third-party Transfer Controller (EDMA_TPTC) Block Diagram

The default burst size (DBS) can be controlled with the CTRL_CORE_CONTROL_IO_1 register in the TDA2xx and TDA2ex Control Module Registers, as shown in Table 14.

Table 14. CTRL_CORE_CONTROL_IO_1

Address offset	0x0000 0554		
Physical Address	0x4A00 2554	**Instance**	CTRL_MODULE_CORE
Description	Register to configure some IP level signals		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED											MMU2_DISABLE	RESERVED			MMU1_DISABLE	RESERVED		TC1_DEFAULT_BURST_SIZE		RESERVED		TC0_DEFAULT_BURST_SIZE		RESERVED		GMII2_SEL	RESERVED		GMII1_SEL	

Bits	Field Name	Description	Type	Reset
31:21	RESERVED	Reserved	R	0x0
20	MMU2_DISABLE	MMU2 DISABLE setting	RW	0x0
19:17	RESERVED	Reserved	R	0x0
16	MMU1_DISABLE	MMU1 DISABLE setting	RW	0x0
15:14	RESERVED	Reserved	R	0x0
13:12	TC1_DEFAULT_BURST_SIZE	EDMA TC1 DEFAULT BURST SIZE setting	RW	0x3
11:10	RESERVED	Reserved	R	0x0
9:8	TC0_DEFAULT_BURST_SIZE	EDMA TC0 DEFAULT BURST SIZE setting	RW	0x3
7:6	RESERVED	Reserved	R	0x0
5:4	GMII2_SEL	GMII2 selection setting	RW	0x0
3:2	RESERVED	Reserved	R	0x0
1:0	GMII1_SEL	GMII1 selection setting	RW	0x0

The effect of the DBS setting on the TC% utilization can be realized with the graph in Figure 12. The data transfer used to obtain these results is a DDR-to-DDR transfer of 16 MB. It is observed that the default setting of 128 bytes per burst generates the maximum TC utilization.

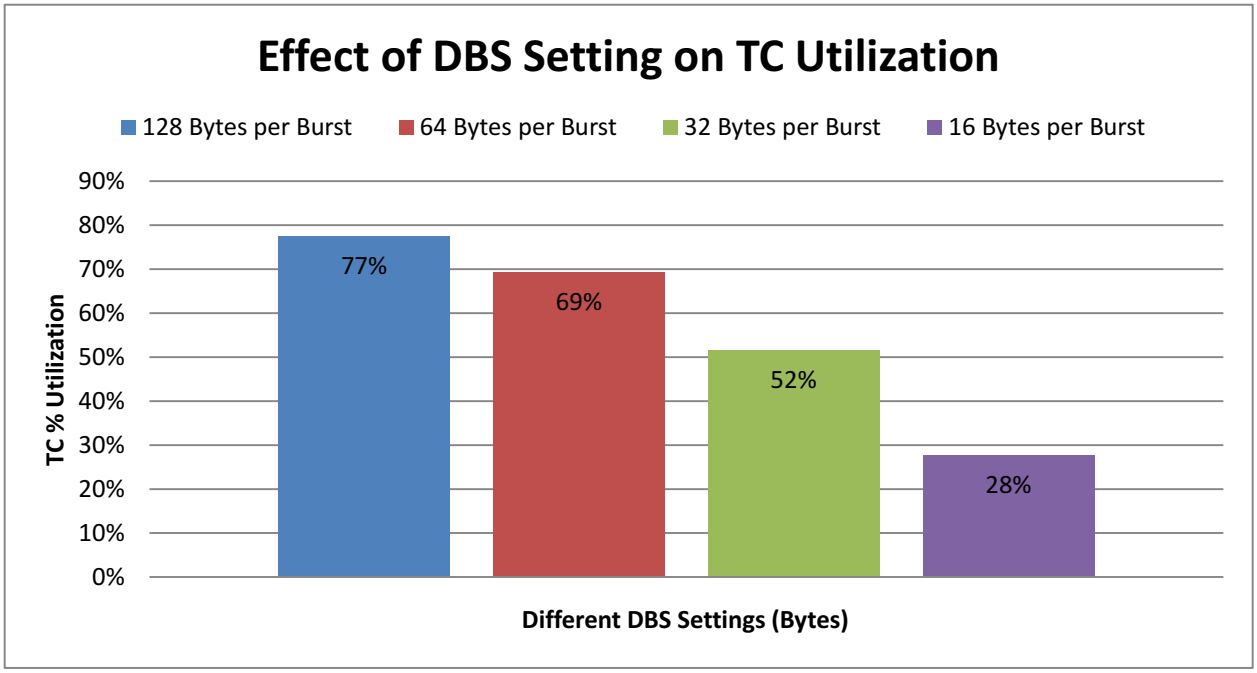


Figure 12. Effect of DBS on System EDMA TC Utilization

The TC read and write controllers in conjunction with the source and destination register sets are responsible for issuing optimally-sized reads and writes to the slave endpoints. An optimally-sized command is defined by the transfer controller default burst size (DBS).

The EDMA_TPTC attempts to issue the largest possible command size as limited by the DBS value or the ABCNT_n[15:0] ACNT and ABCNT_n[31:16] BCNT value of the TR. The EDMA_TPTC obeys the following rules:

- The read and write controllers always issue commands less than or equal to the DBS value.
- The first command of a 1D transfer command always aligns the address of subsequent commands to the DBS value.

Table 15 lists the TR segmentation rules that are followed by the EDMA_TPTC. In summary, if the ABCNT_n[15:0] ACNT value is larger than the DBS value, then the EDMA_TPTC breaks the ABCNT_n[15:0] ACNT array into DBS-sized commands to the source and destination addresses. Each ABCNT_n[31:16] BCNT number of arrays are then serviced in succession.

For BCNT arrays of ACNT bytes (that is, a 2D transfer), if the ABCNT_n[15:0] ACNT value is less than or equal to the DBS value, then the TR may be optimized into a 1D-transfer in order to maximize efficiency. The optimization takes place if the EDMA_TPTC recognizes that the 2D-transfer is organized as a single dimension (ABCNT_n[15:0] ACNT == BIDX_n) and the ACNT value is a power of 2.

Table 15. System EDMA TC Optimization Rules

ACNT ≤ DBS	ACNT is Power of 2	BIDX = ACNT	BCNT ≤ 1023	SAM/DAM = Increment	Description
Yes	Yes	Yes	Yes	Yes	Optimized
No	X	X	X	X	Not Optimized
X	No	X	X	X	Not Optimized
X	X	No	X	X	Not Optimized
X	X	X	No	X	Not Optimized
X	X	X	X	No	Not Optimized

In summary, Table 16 lists the factors that affect the EDMA performance.

Table 16. Factors Affecting System EDMA Performance

Factors	Impact	General Recommendation
Source/Destination Memory	The transfer speed depends on SRC/DST memory bandwidth.	Know the nature of the source and destination memory, specifically the frequency of operation and the bus width.
Transfer Size	Throughput is less for small transfers due to transfer overhead and latency.	Configure EDMA for larger transfer size as throughput, small transfer size is dominated by transfer overhead.
A-Sync/AB-Sync	Performance depends on the number of TRs (Transfer Requests). More TRs would mean more overhead.	Using AB-Sync transfers gives better performance than chaining A-Sync transfers.
Source/Destination Bidx	Optimization will not be done if BIDX is not equal to ACNT value optimization guidelines.	Whenever possible, follow the EDMA TC optimization guidelines. See the TPTC spec for optimization details.
Queue TC Usage	Performance is the same for both TCs.	Both TCs have the same configuration and show the same performance.
Burst Size	Decides the largest possible read/write command submission by TC.	The default burst size for all transfer controllers is 128 bytes. This also results in most efficient transfers/throughput in most memory-to-memory transfer scenarios.
Source/Destination Alignment	Slight performance degradation if source/destination are not aligned to Default Burst Size (DBS) boundaries.	For smaller transfers, as much as possible, source and destination addresses should be aligned across DBS boundaries.

4 DSP Subsystem EDMA

The DSP subsystem is built around the high-performance TI's standard TMS320C66x™ DSP CorePac core. The subsystem includes EDMA and L2 interconnect to facilitate high-bandwidth transfers between chip-level resources/memory and DSP memory.

The DSP subsystem inputs a primary /1 (dsp_clk) and internally generates the /2 (clk2) or /3 (clk3) or /4 (clk4) clock rates. The division is defined upon device boot time through a signal level externally applied on the device sysboot15 input. The actual bit configuration is latched upon power-on reset by the CTRL_CORE_BOOTSTRAP[15] SYS_BOOT_15_CLOCK_DIVIDER boot status bit in the TDA2xx and TDA2ex Control Module Registers. Only DSP_CLK3 clock is supported on TDA2xx and TDA2ex. Sysboot[15] must be pulled to vdd for proper device operation (SR1.x). For SR2.0, it is used to permanently disable the internal PU/PD resistors on pads gpmc_a[27:19]. For SR2.0 the DSP EDMA always operates at clk3.

The different portions of the DSP subsystem run at the DSP_CLK3 and DSP_CLK, as shown in Figure 13.

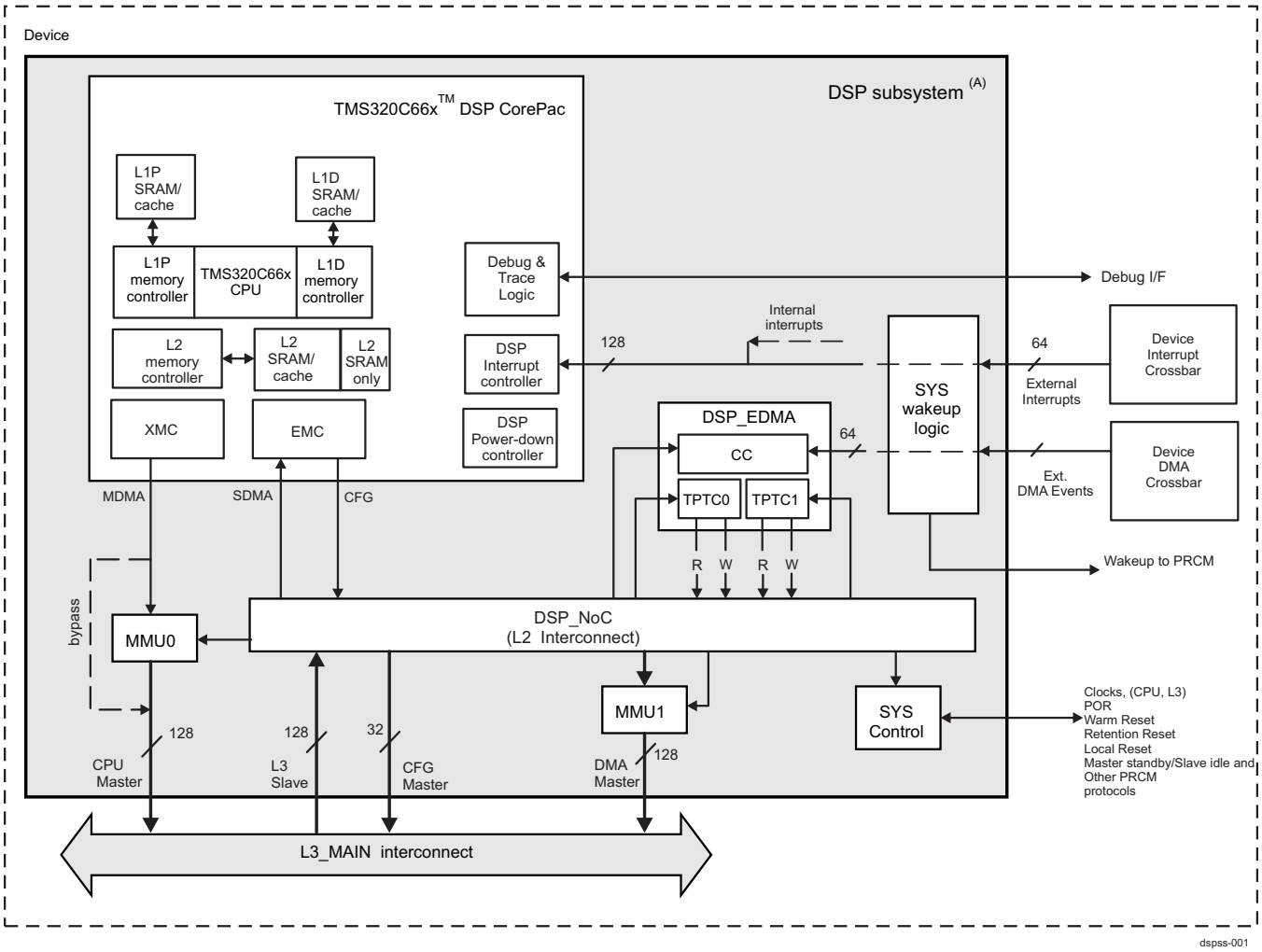


Figure 13. DSP Subsystem Block Diagram and Clocking Scheme

This section provides a throughput analysis of the DSP SS EDMA module. The enhanced direct memory access module, also referred to as EDMA, performs high-performance data transfers between two slave points, memories and peripheral devices without the digital signal processor (DSP) support during transfer. EDMA transfer is programmed through a logical EDMA channel, which allows the transfer to be optimally tailored to the requirements of the application. The EDMA can also perform transfers between external memories and between device subsystems internal memories, with some performance loss caused by resource sharing between the read and write ports.

The EDMA controller block diagram is shown in Figure 14. The EDMA controller is based on two major principal blocks:

- EDMA third-party channel controller (EDMA_TPCC)
- EDMA third-party transfer controller (EDMA_TPTC)

The EDMA controller's primary purpose is to service user-programmed data transfers between internal or external memory-mapped slave endpoints. It can also be configured for servicing event driven peripherals (such as serial ports), as well. There are 64 direct memory access (DMA) channels and 8 QDMA channels serviced by two concurrent physical channels.

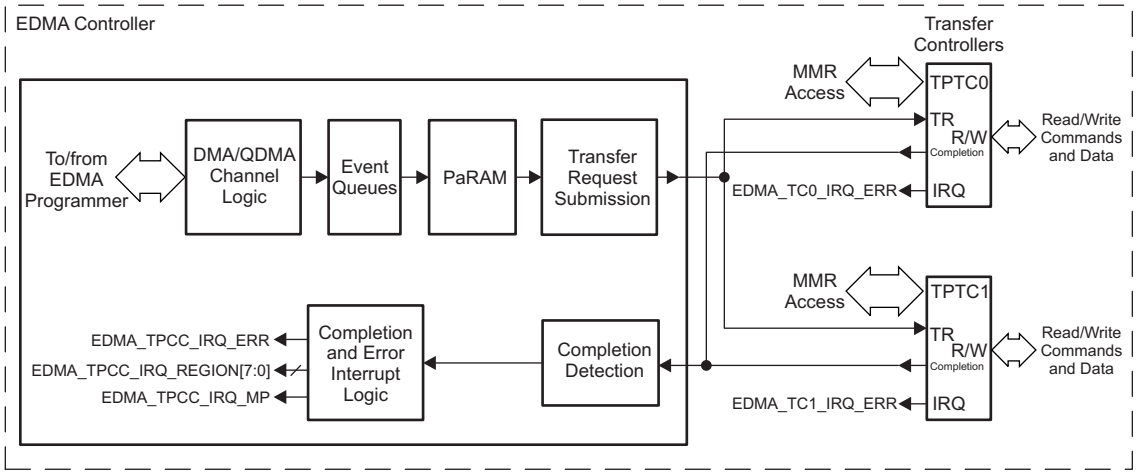


Figure 14. DSP Subsystem EDMA Block Diagram

DMA channels are triggered by external event, manual write to event set register (ESR), or chained event. QDMA are auto-triggered when write is performed to the user-programmable trigger word. Once a trigger event is recognized, the event is queued in the programmed event queue. If two events are detected simultaneously, then the lowest-numbered channel has highest priority. Each event in the event queue is processed in the order it was queued. On reaching the head of the queue, the PaRAM associated with that event is read to determine the transfer details. The transfer request (TR) submission logic evaluates the validity of the TR and is submits a valid transfer request to the appropriate transfer controller.

The maximum theoretical bandwidth for a given transfer can be found by multiplying the width of the interface and the frequency at which it transfers data. The maximum speed the transfer can achieve is equal to the bandwidth of the limiting port. In general, a given transfer scenario will never achieve maximum theoretical band width due to several factors, like transfer overheads, access latency of source/destination memories, finite number of cycles taken by EDMA CC and EDMA TC between the time the transfer event is registered to the time the first read command is issued to EDMA TC. These overheads can be calibrated by looking at the time taken to do a 1 byte transfer. These factors are not excluded in these throughput measurements.

4.1 DSP Subsystem EDMA Performance

The formulas used for the throughput calculations are:

Actual Throughput = (Transfer Size/Time Taken)

Ideal Throughput = Frequency of Limiting Port × Data Bus Width in Bytes

TC Utilization = (Actual Throughput/Ideal Throughput) × 100

4.1.1 DSP Subsystem EDMA Read and Write

The common system setup for the EDMA throughput measurement is:

- EMIF configuration
- CAS write latency – 6
- CAS latency - 7
- SDRAM Data Bus width : 32
- DDR in non-interleaved mode

The data presented is for stand-alone transfers with no other ongoing or competing traffic. All profiling is done with the C66x CorePac Timer operating at 600 MHz.

4.1.2 DSP Subsystem EDMA Results

Table 17. DSP Subsystem EDMA 1 TC Read and Write Throughput With CorePac Timer

No.	Source	Destination	ACNT	BCNT	Size (KB)	Ideal Throughput (MB/s)	Bandwidth (MB/s)	TC Utilization (%)
1	EMIF 0 DDR	EMIF 0 DDR	65535	128	8192	3200	2998.91	94
2	OCMC RAM0	EMIF 0 DDR	65535	4	256	3200	2540.81	79
3	EMIF 0 DDR	OCMC RAM	65535	4	256	3200	2540.81	79
4	DSP L2	EMIF 0 DDR	65535	8	256	3200	2537.86	79
5	EMIF 0 DDR	DSP L2	65535	4	256	3200	2585.93	81
6	IVA SL2	EMIF 0 DDR	65535	4	256	3200	1524.68	48
7	EMIF 0 DDR	IVA SL2	65535	4	256	3200	1524.68	48
8	IVA SL2	DSP L2	65535	4	256	3200	1526.82	48
9	DSP L2	IVA SL2	65535	4	256	3200	1525.75	48
10	DSP L2	OCMC RAM	65535	4	256	3200	2537.86	79
11	OCMC RAM	DSP L2	65535	4	256	3200	2592.06	81
12	PRUSS1 IRAM	EMIF 0 DDR	1024	12	12	800	319.22	40
13	EMIF 0 DDR	PRUSS1 IRAM	1024	12	12	800	318.23	40
14	PRUSS1 DRAM0	EMIF 0 DDR	1024	8	8	800	333.37	42
15	EMIF 0 DDR	PRUSS1 DRAM0	1024	8	8	800	332.56	42
16	PRUSS1 DRAM1	EMIF 0 DDR	1024	8	8	800	333.37	42
17	EMIF 0 DDR	PRUSS1 DRAM1	1024	8	8	800	333.37	42
18	PRUSS2 IRAM	EMIF 0 DDR	1024	8	8	800	313.36	39
19	EMIF 0 DDR	PRUSS2 IRAM	1024	8	8	800	312.41	39
20	PRUSS2 DRAM0	EMIF 0 DDR	1024	12	12	800	319.33	40
21	EMIF 0 DDR	PRUSS2 DRAM0	1024	12	12	800	317.85	40
22	PRUSS2 DRAM1	EMIF 0 DDR	1024	8	8	800	326.72	41
23	EMIF 0 DDR	PRUSS2 DRAM1	1024	8	8	800	326.72	41
24	PRUSS2 DRAM2	EMIF 0 DDR	1024	32	32	800	326.92	41
25	EMIF 0 DDR	PRUSS2 DRAM2	1024	32	32	800	326.33	41

Table 18. DSP Subsystem EDMA 1 TC Read and Write Throughput With L3 Statistic Collectors

No.	Source	Destination	ACNT	BCNT	Size (KB)	Ideal Throughput (MB/s)	Bandwidth (MB/s)	TC Utilization (%)
1	EMIF 0 DDR	EMIF 0 DDR	65535	128	8192	3200	3100	97
2	OCMC RAM	EMIF 0 DDR	65535	4	256	3200	2700	84
3	EMIF 0 DDR	OCMC RAM	65535	4	256	3200	2900	91
4	DSP L2	EMIF 0 DDR	65535	8	256	3200	2760	86
5	EMIF 0 DDR	DSP L2	65535	4	256	3200	2950	92
6	IVA SL2	EMIF 0 DDR	65535	4	256	3200	1610	50
7	EMIF 0 DDR	IVA SL2	65535	4	256	3200	1600	50
8	IVA SL2	DSP L2	65535	4	256	3200	1600	50
9	DSP L2	IVA SL2	65535	4	256	3200	1610	50
10	DSP L2	OCMC RAM	65535	4	256	3200	2660	83
11	OCMC RAM	DSP L2	65535	4	256	3200	2710	85
12	PRUSS1 IRAM	EMIF 0 DDR	1024	12	12	800	320	40
13	EMIF 0 DDR	PRUSS1 IRAM	1024	12	12	800	320	40
14	PRUSS1 DRAM0	EMIF 0 DDR	1024	8	8	800	320	40
15	EMIF 0 DDR	PRUSS1 DRAM0	1024	8	8	800	320	40
16	PRUSS1 DRAM1	EMIF 0 DDR	1024	8	8	800	320	40
17	EMIF 0 DDR	PRUSS1 DRAM1	1024	8	8	800	320	40
18	PRUSS2 IRAM	EMIF 0 DDR	1024	8	8	800	320	40
19	EMIF 0 DDR	PRUSS2 IRAM	1024	8	8	800	320	40
20	PRUSS2 DRAM0	EMIF 0 DDR	1024	12	12	800	320	40
21	EMIF 0 DDR	PRUSS2 DRAM0	1024	12	12	800	320	40
22	PRUSS2 DRAM1	EMIF 0 DDR	1024	8	8	800	320	40
23	EMIF 0 DDR	PRUSS2 DRAM1	1024	8	8	800	320	40
24	PRUSS2 DRAM2	EMIF 0 DDR	1024	32	32	800	320	40
25	EMIF 0 DDR	PRUSS2 DRAM2	1024	32	32	800	320	40

Table 19. DSP Subsystem EDMA 2 TC Read and Write Throughput With CorePac Timer

No.	Source	Destination	ACNT	BCNT	Size (KB)	Ideal Throughput (MB/s)	Bandwidth (MB/s)	TC Utilization (%)
1	EMIF 0 DDR	EMIF 0 DDR	65535	128	8192	3200	2728.92	85
2	OCMC RAM	EMIF 0 DDR	65535	4	256	3200	2678.6	84
3	EMIF 0 DDR	OCMC RAM	65535	4	256			
4	DSP L2	EMIF 0 DDR	65535	4	256	3200	3006	94
5	EMIF 0 DDR	DSP L2	65535	4	256			
6	IVA SL2	EMIF 0 DDR	65535	4	256	3200	1527.81	48
7	EMIF 0 DDR	IVA SL2	65535	4	256			
8	IVA SL2	DSP L2	65535	4	256	3200	1529.86	48
9	DSP L2	IVA SL2	65535	4	256			
10	DSP L2	OCMC RAM	65535	4	256	3200	3009.75	94
11	OCMC RAM	DSP L2	65535	4	256			
12	PRUSS1 IRAM	EMIF 0 DDR	1024	12	12	800	322.56	40
13	EMIF 0 DDR	PRUSS1 IRAM	1024	12	12			
14	PRUSS1 DRAM0	EMIF 0 DDR	1024	8	8	800	338.8	42
15	EMIF 0 DDR	PRUSS1 DRAM0	1024	8	8			
16	PRUSS1 DRAM1	EMIF 0 DDR	1024	8	8	800	338.8	42
17	EMIF 0 DDR	PRUSS1 DRAM1	1024	8	8			
18	PRUSS2 IRAM	EMIF 0 DDR	1024	8	8	800	316.12	40
19	EMIF 0 DDR	PRUSS2 IRAM	1024	8	8			
20	PRUSS2 DRAM0	EMIF 0 DDR	1024	12	12	800	322.69	40
21	EMIF 0 DDR	PRUSS2 DRAM0	1024	12	12			
22	PRUSS2 DRAM1	EMIF 0 DDR	1024	8	8	800	327.8	41
23	EMIF 0 DDR	PRUSS2 DRAM1	1024	8	8			
24	PRUSS2 DRAM2	EMIF 0 DDR	1024	32	32	800	327.89	41
25	EMIF 0 DDR	PRUSS2 DRAM2	1024	32	32			

Table 20. DSP Subsystem EDMA 2 TC Read and Write Throughput With L3 Statistic Collectors

No.	Source	Destination	ACNT	BCNT	Size (KB)	Ideal Throughput (MB/s)	Bandwidth (MB/s)	TC Utilization (%)
1	EMIF 0 DDR	EMIF 0 DDR	65535	128	8192	3200	2690	84
2	OCMC RAM	EMIF 0 DDR	65535	4	256	3200	2870	90
3	EMIF 0 DDR	OCMC RAM	65535	4	256			
4	DSP L2	EMIF 0 DDR	65535	4	256	3200	3050	95
5	EMIF 0 DDR	DSP L2	65535	4	256			
6	IVA SL2	EMIF 0 DDR	65535	4	256	3200	1600	50
7	EMIF 0 DDR	IVA SL2	65535	4	256			
8	IVA SL2	DSP L2	65535	4	256	3200	1610	50
9	DSP L2	IVA SL2	65535	4	256			
10	DSP L2	OCMC RAM	65535	4	256	3200	3050	95
11	OCMC RAM	DSP L2	65535	4	256			
12	PRUSS1 IRAM	EMIF 0 DDR	1024	12	12	800	340	43
13	EMIF 0 DDR	PRUSS1 IRAM	1024	12	12			
14	PRUSS1 DRAM0	EMIF 0 DDR	1024	8	8	800	340	43
15	EMIF 0 DDR	PRUSS1 DRAM0	1024	8	8			
16	PRUSS1 DRAM1	EMIF 0 DDR	1024	8	8	800	340	43
17	EMIF 0 DDR	PRUSS1 DRAM1	1024	8	8			
18	PRUSS2 IRAM	EMIF 0 DDR	1024	8	8	800	340	43
19	EMIF 0 DDR	PRUSS2 IRAM	1024	8	8			
20	PRUSS2 DRAM0	EMIF 0 DDR	1024	12	12	800	340	43
21	EMIF 0 DDR	PRUSS2 DRAM0	1024	12	12			
22	PRUSS2 DRAM1	EMIF 0 DDR	1024	8	8	800	340	43
23	EMIF 0 DDR	PRUSS2 DRAM1	1024	8	8			
24	PRUSS2 DRAM2	EMIF 0 DDR	1024	32	32	800	340	43
25	EMIF 0 DDR	PRUSS2 DRAM2	1024	32	32			

4.2 DSP Subsystem EDMA Observations

NOTE: On the TDA2xx and TDA2ex device, all DSP subsystem transfer controllers yield identical performance for all transfer scenarios because both TC have the same configuration, and most importantly the same FIFOSIZE for a given burst size.

EDMA channel parameters allow many different transfer configurations. Typical transfer configurations result in transfer controllers bursting the read write data in default burst size chunks, thereby, keeping the busses fully utilized. However, in some configurations, the TC issues less than optimally sized read/write commands (less than default burst size), reducing performance. To properly design a system, it is important to know which configurations offer the best performance for high-speed operations.

On TDA2xx and TDA2ex, there are two transfer controllers to move data between slave end points. The default configuration for the transfer controllers is shown in Table 21.

Table 21. Default Configuration for the Transfer Controllers

Name	Description	TC0	TC1
TCCFG[2:0] FIFOSIZE	Channel FIFO Size	1024 Bytes	1024 Bytes
TCCFG[5:4] BUSWIDTH	Data Transfer Bus Width	16 Bytes	16 Bytes
TCCFG[9:8] DSTREGDEPTH	Destination Register Depth	4 entries	4 entries
DBS (Default Burst Size)	Size of each data burst	Configurable	Configurable

The individual TC performance for paging/memory to memory transfers is essentially dictated by the TC configuration. In most scenarios, the FIFOSIZE and default burst size configuration for the TC have the most significant impact on the TC performance; the BUSWIDTH configuration is dependent on the device architecture and the DSTREGDEPTH values impact the number of in-flight transfers.

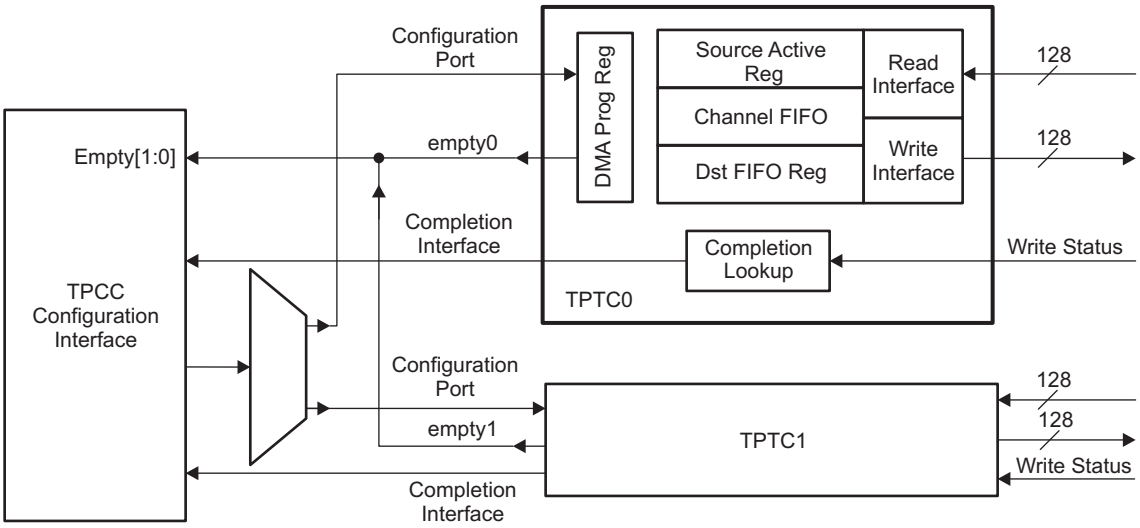


Figure 15. DSP EDMA Third-party Transfer Controller (EDMA_TPTC) Block Diagram

The default burst size (DBS) can be controlled with the C66x_OSS_BUS_CONFIG register in the TDA2xx and TDA2ex DSP Subsystem OCP Registers, as shown in Table 22.

Table 22. C66x_OSS_BUS_CONFIG

Address offset	0x0000 0014		
Physical Address	0x1D0 0014 (DSP View)	**Instance**	C66x_OCP_REGISTERS
Description	Bus Configuration		
Type	RW		

31	30 29 28	27 26 25	24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
RESERVED	SDMA_PRI	RESERVED	NOPOSTOVERRIDE	RESERVED	SDMA_L2PRES	RESERVED	CFG_L2PRES	RESERVED	TC1_L2PRES	RESERVED	TC0_L2PRES	RESERVED	TC1_DBS	RESERVED	TC0_DBS

Bits	Field Name	Description	Type	Reset
31	RESERVED	Reserved	R	0x0
30:28	SDMA_PRI	Sets the CBA/VBusM Priority for the CGEM SDMA port. Can typically be left at default value.	R	0x4
27:25	RESERVED	Reserved	R	0x0
24	NOPOSTOVERRIDE	Non-Posted writes setting	RW	0x1
23:22	RESERVED	Reserved	R	0x0
21:20	SDMA_L2PRES	OCP Slave port L2 interconnect pressure driven on ocp mflag to control arbitration within the L2 interconnect	RW	0x0
19:18	RESERVED	Reserved	R	0x0
17:16	CFG_L2PRES	CGEM CFG L2 interconnect pressure driven on ocp mflag to control arbitration within the L2 interconnect	RW	0x0
15:14	RESERVED	Reserved	R	0x0
13:12	TC1_L2PRES	TC1 L2 interconnect pressure driven on ocp mflag to control arbitration within the L2 interconnect	RW	0x0
11:10	RESERVED	Reserved	R	0x0
9:8	TC0_L2PRES	TC0 L2 interconnect pressure driven on ocp mflag to control arbitration within the L2 interconnect	RW	0x0
7:6	RESERVED	Reserved	R	0x0
5:4	TC1_DBS	TC1 Default Burst size setting	RW	0x3
3:2	RESERVED	Reserved	R	0x0
1:0	TC0_DBS	TC0 Default Burst size setting	RW	0x3

The TC read and write controllers in conjunction with the source and destination register sets are responsible for issuing optimally-sized reads and writes to the slave endpoints. An optimally-sized command is defined by the transfer controller default burst size (DBS).

The EDMA_TPTC attempts to issue the largest possible command size as limited by the DBS value or the ABCNT_n[15:0] ACNT and ABCNT_n[31:16] BCNT value of the TR. EDMA_TPTC obeys the following rules: The read/write controllers always issue commands less than or equal to the DBS value. The first command of a 1D transfer command always aligns the address of subsequent commands to the DBS value.

Table 23 lists the TR segmentation rules that are followed by the EDMA_TPTC. In summary, if the ABCNT_n[15:0] ACNT value is larger than the DBS value, then the EDMA_TPTC breaks the ABCNT_n[15:0] ACNT array into DBS-sized commands to the source/destination addresses. Each ABCNT_n[31:16] BCNT number of arrays are then serviced in succession.

For BCNT arrays of ACNT bytes (that is, a 2D transfer), if the ABCNT_n[15:0] ACNT value is less than or equal to the DBS value, then the TR may be optimized into a 1D-transfer in order to maximize efficiency. The optimization takes place if the EDMA_TPTC recognizes that the 2D-transfer is organized as a single dimension (ABCNT_n[15:0] ACNT == BIDX_n) and the ACNT value is a power of 2.

Table 23. DSP EDMA TC Optimization Rules

ACNT ≤ DBS	ACNT is Power of 2	BIDX = ACNT	BCNT ≤ 1023	SAM/DAM = Increment	Description
Yes	Yes	Yes	Yes	Yes	Optimized
No	X	X	X	X	Not Optimized
X	No	X	X	X	Not Optimized
X	X	No	X	X	Not Optimized
X	X	X	No	X	Not Optimized
X	X	X	X	No	Not Optimized

In summary, Table 24 lists the factors that affect the EDMA performance.

Table 24. Factors Affecting System EDMA Performance

Factors	Impact	General Recommendation
Source/Destination Memory	The transfer speed depends on SRC/DST memory bandwidth.	Know the nature of the source and destination memory, specifically the frequency of operation and the bus width.
Transfer Size	Throughput is less for small transfers due to transfer overhead/latency.	Configure EDMA for larger transfer size as throughput, small transfer size is dominated by transfer overhead.
A-Sync/AB-Sync	Performance depends on the number of TRs (Transfer Requests). More TRs would mean more overhead.	Using AB-Sync transfers gives better performance than chaining A-Sync transfers.
Source/Destination Bidx	Optimization will not be done if BIDX is not equal to ACNT value optimization guidelines.	Whenever possible, follow the EDMA TC optimization guidelines. See the TPTC spec for optimization details.
Queue TC Usage	Performance is the same for both TCs.	Both TCs have the same configuration and show the same performance.
Burst Size	Decides the largest possible read/write command submission by TC.	The default burst size for all transfer controllers is 128 bytes. This also results in most efficient transfers/throughput in most memory-to-memory transfer scenarios.
Source/Destination Alignment	Slight performance degradation if source/destination are not aligned to Default Burst Size (DBS) boundaries.	For smaller transfers, as much as possible, source and destination addresses should be aligned across DBS boundaries.

5 Embedded Vision Engine (EVE) Subsystem EDMA

NOTE: This section is not applicable to TDA2ex.

The Embedded Vision Engine (EVE) module, Figure 16, is a programmable imaging and vision processing engine, intended to be used in devices that serves consumer electronics imaging and vision applications. The EVE Module consists of an ARP32 scalar core and a VCOP vector core and DMA controller.

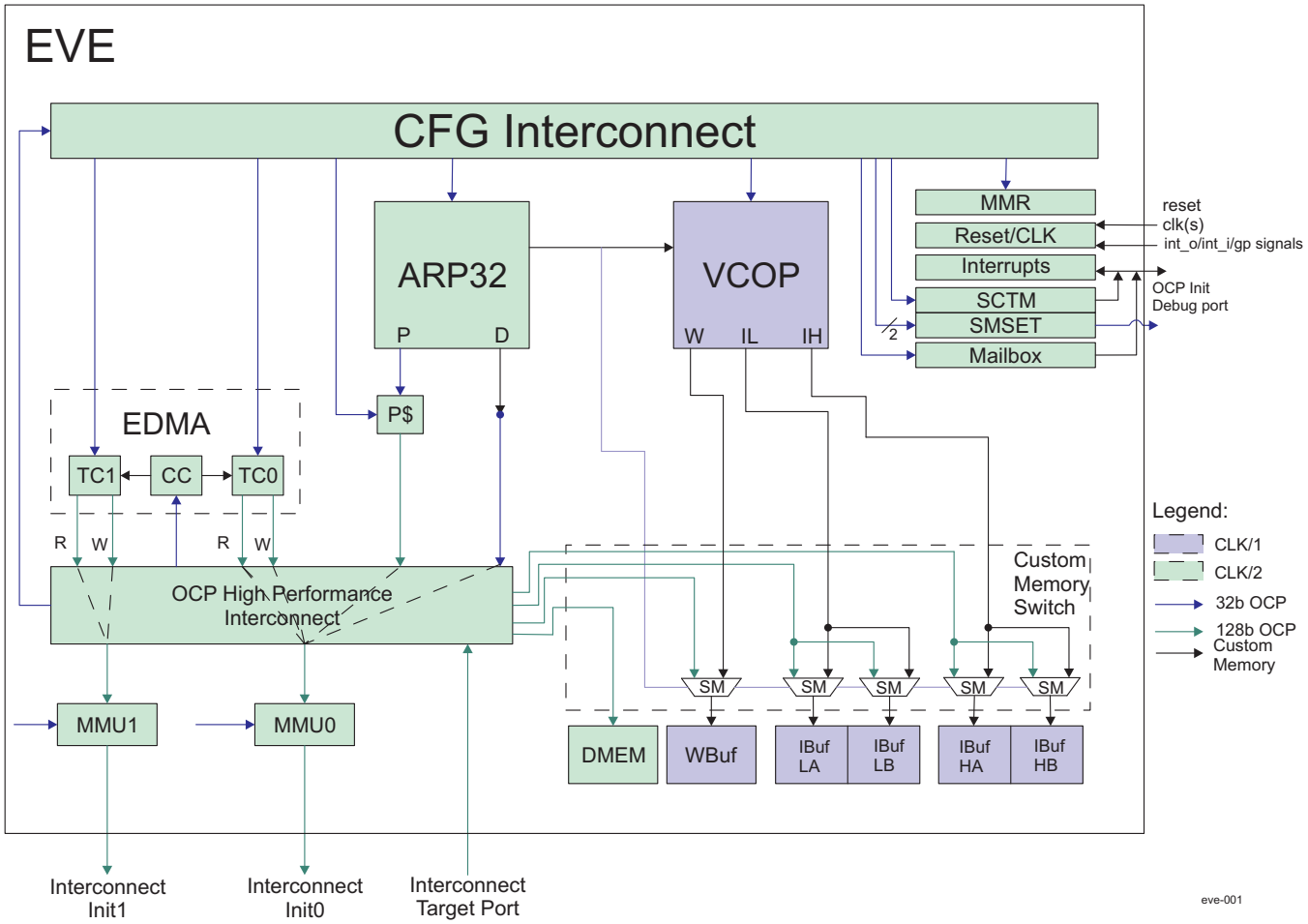


Figure 16. EVE Subsystem Block Diagram and Subsystem Clocking Architecture

The ARP32 scalar core plays the role of the subsystem controller, coordinating internal EVE interaction (VCOP, EDMA), as well as interaction with the host processor (ARM typically) and DSP (CGEM typically). The ARP32 program memory is serviced via a dedicated direct-mapped program cache. Data memory accesses are typically serviced by tightly coupled DMEM block though ARP32 is able to access other memory blocks as well as both internal and external MMRs. The Vector Coprocessor (VCOP) is a SIMD engine with built-in loop control and address generation. The EDMA block is the local DMA, and is used to transfer data between system memories (typically SDRAM, and/or L3 SRAM) and internal EVE memories.

The OCP interconnect is conceptually broken into two categories: OCP high performance interconnect and OCP configuration interconnect. The OCP high performance interconnect serves as the primary high bandwidth (partial) crossbar connection between the EDMA, ARP32, DMA OCP Init/Target busses and the EVE Memories. The OCP CFG interconnect provides connectivity to the various MMRs located within EVE.

This section provides a throughput analysis of the EVE EDMA. The enhanced direct memory access module, also called EDMA, performs high-performance data transfers between two slave points, memories and peripheral devices without ARP32 support during transfer. EDMA transfer is programmed through a logical EDMA channel, which allows the transfer to be optimally tailored to the requirements of the application. The EDMA can also perform transfers between external memories and between device subsystems internal memories, with some performance loss caused by resource sharing between the read and write ports.

The EDMA controller block diagram is shown in Figure 17. The EDMA controller is based on two major principal blocks:

- EDMA third-party channel controller (EDMA_TPCC)
- EDMA third-party transfer controller (EDMA_TPTC)

The EDMA controller's primary purpose is to service user programmed data transfers between internal or external memory-mapped slave endpoints. It can also be configured for servicing event driven peripherals (such as serial ports), as well. There are 64 direct memory access (DMA) channels and 8 QDMA channels serviced by two concurrent physical channels.

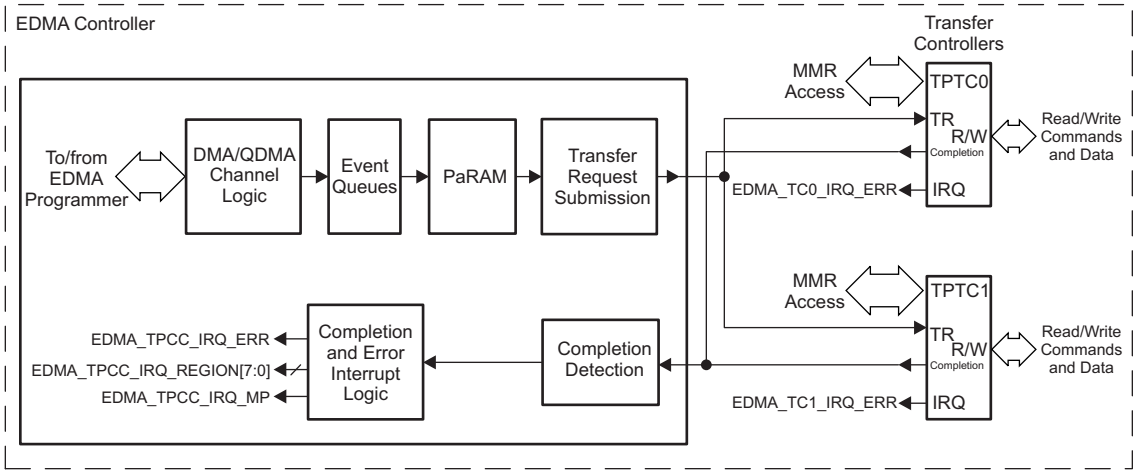


Figure 17. EVE Subsystem EDMA Controller Block Diagram

DMA channels are triggered by external event, manual write to event set register (ESR), or chained event. QDMA are auto-triggered when write is performed to the user-programmable trigger word. Once a trigger event is recognized, the event is queued in the programmed event queue. If two events are detected simultaneously, then the lowest-numbered channel has highest priority. Each event in the event queue is processed in the order it was queued. On reaching the head of the queue, the PaRAM associated with that event is read to determine the transfer details. The transfer request (TR) submission logic evaluates the validity of the TR and is submits a valid transfer request to the appropriate transfer controller. The maximum theoretical bandwidth for a given transfer can be found by multiplying the width of the interface and the frequency at which it transfers data.

The maximum speed the transfer can achieve is equal to the bandwidth of the limiting port. In general, a given transfer scenario will never achieve maximum theoretical band width due to several factors, like transfer overheads, access latency of source/destination memories, finite number of cycles taken by EDMA CC and EDMA TC between the time the transfer event is registered to the time the first read command is issued to EDMA TC. These overheads can be calibrated by looking at the time taken to do a 1 byte transfer. These factors are not excluded in these throughput measurements.

5.1 EVE EDMA Performance

The formulas used for the throughput calculations are:

Actual Throughput = (Transfer Size/Time Taken)

Ideal Throughput = Frequency of Limiting Port × Data Bus Width in Bytes

TC Utilization = (Actual Throughput/Ideal Throughput) × 100

5.1.1 EVE EDMA Read and Write

The common system setup for the EDMA throughput measurement is:

- EVE ARP32 clock: 266 MHz (unless specified)
- DDR clock: 532 MHz (unless specified)
- EMIF configuration
- CAS write latency – 6
- CAS latency - 7
- SDRAM Data Bus width : 32
- DDR in non-interleaved mode

The data presented is for stand-alone transfers with no other ongoing or competing traffic. All profiling has been done with ARP32 CPU counters.

5.1.2 EVE EDMA Results

Table 25. EVE EDMA Single TC Read and Write Performance With ARP32 Counters

No.	Source	Destination	ACNT	BCNT	CCNT	Transfer Size (KB)	Ideal Through-put (MB/s)	Through-put (MB/s)	TC Utilization (%)
1	EMIF 0	EMIF 0	65535	128	1	8192	4256	2984.86	70.13
2	EMIF 0	EMIF 1	65535	128	1	8192	4256	3298.52	77.5
3	EMIF 1	EMIF 0	65535	128	1	8192	4256	3299.09	77.52
4	EMIF 1	EMIF 1	65535	128	1	8192	4256	2986.49	70.17
5	OCMC RAM	EMIF 0	65535	4	1	256	4256	3268.26	76.79
6	EMIF 0	OCMC RAM	65535	4	1	256	4256	3283.01	77.14
7	DSP L2	EMIF 0	65535	4	1	256	4256	3114.47	73.18
8	EMIF 0	DSP L2	65535	4	1	256	4256	3096	72.74
9	IVA SL2	EMIF 0	65535	4	1	256	4256	1519.59	35.7
10	EMIF 0	IVA SL2	65535	4	1	256	4256	1518.73	35.68
11	IVA SL2	DSP L2	65535	4	1	256	4256	1517.86	35.66
12	DSP L2	IVA SL2	65535	4	1	256	4256	1519	35.69
13	IBUFLA	EMIF 0	8192	1	1	8	4256	2019.77	47.46
14	EMIF 0	IBUFLA	8192	1	1	8	4256	1953.18	45.89
15	IBUFHA	EMIF 0	8192	1	1	8	4256	2023.79	47.55
16	EMIF 0	IBUFHA	8192	1	1	8	4256	1932.66	45.41
17	IBUFLB	EMIF 0	8192	1	1	8	4256	2022.17	47.51
18	EMIF 0	IBUFLB	8192	1	1	8	4256	1932.4	45.4
19	IBUFHB	EMIF 0	8192	1	1	8	4256	1994.79	46.87
20	EMIF 0	IBUFHB	8192	1	1	8	4256	1930.02	45.35
21	IBUFLA	OCMC RAM	8192	1	1	8	4256	2149.98	50.52
22	OCMC RAM	IBUFLA	8192	1	1	8	4256	2035.46	47.83
23	IBUFLB	OCMC RAM	8192	1	1	8	4256	2140.57	50.3
24	OCMC RAM	IBUFLB	8192	1	1	8	4256	2045.61	48.06

Table 25. EVE EDMA Single TC Read and Write Performance With ARP32 Counters (continued)

No.	Source	Destination	ACNT	BCNT	CCNT	Transfer Size (KB)	Ideal Through-put (MB/s)	Through-put (MB/s)	TC Utilization (%)
25	IBUFHA	OCMC RAM	8192	1	1	8	4256	2113.56	49.66
26	OCMC RAM	IBUFHA	8192	1	1	8	4256	2051.92	48.21
27	IBUFHB	OCMC RAM	8192	1	1	8	4256	2130.14	50.05
28	OCMC RAM	IBUFHB	8192	1	1	8	4256	2067.09	48.57
29	EVE1 IBUFHB	EVE2 IBUFHB	8192	1	1	8	4256	2107.1	49.51
30	EVE2 IBUFHB	EVE1 IBUFHB	8192	1	1	8	4256	2067.33	48.57
31	EVE1 IBUFHB	EVE3 IBUFHB	8192	1	1	8	4256	2142.66	50.34
32	EVE3 IBUFHB	EVE1 IBUFHB	8192	1	1	8	4256	2071.76	48.68
33	EVE1 IBUFHB	EVE4 IBUFHB	8192	1	1	8	4256	2133.89	50.14
34	EVE4 IBUFHB	EVE1 IBUFHB	8192	1	1	8	4256	2076.25	48.78

Table 26. EVE EDMA Single TC Read and Write Performance With L3 Statistic Collectors

No.	Source	Destination	ACNT	BCNT	CCNT	Transfer Size (KB)	Ideal Through-put (MB/s)	Through-put (MB/s)	TC Utilization (%)
1	EMIF 0	EMIF 0	65535	128	1	8192	4256	3070	72
2	EMIF 0	EMIF 1	65535	128	1	8192	4256	3320	78
3	EMIF 1	EMIF 0	65535	128	1	8192	4256	3330	78
4	EMIF 1	EMIF 1	65535	128	1	8192	4256	3010	71
5	OCMC RAM	EMIF 0	65535	4	1	256	4256	3280	77
6	EMIF 0	OCMC RAM	65535	4	1	256	4256	3280	77
7	DSP L2	EMIF 0	65535	4	1	256	4256	3150	74
8	EMIF 0	DSP L2	65535	4	1	256	4256	3260	77
9	IVA SL2	EMIF 0	65535	4	1	256	4256	3200	75
10	EMIF 0	IVA SL2	65535	4	1	256	4256	3200	75
11	IVA SL2	DSP L2	65535	4	1	256	4256	3200	75
12	DSP L2	IVA SL2	65535	4	1	256	4256	3200	75
13	IBUFLA	EMIF 0	8192	1	1	8	4256	2750	65
14	EMIF 0	IBUFLA	8192	1	1	8	4256	2720	64
15	IBUFHA	EMIF 0	8192	1	1	8	4256	2750	65
16	EMIF 0	IBUFHA	8192	1	1	8	4256	2720	64
17	IBUFLB	EMIF 0	8192	1	1	8	4256	2720	64
18	EMIF 0	IBUFLB	8192	1	1	8	4256	2720	64
19	IBUFHB	EMIF 0	8192	1	1	8	4256	2750	65
20	EMIF 0	IBUFHB	8192	1	1	8	4256	2720	64
21	IBUFLA	OCMC RAM	8192	1	1	8	4256	2720	64
22	OCMC RAM	IBUFLA	8192	1	1	8	4256	2790	66
23	IBUFLB	OCMC RAM	8192	1	1	8	4256	2720	64
24	OCMC RAM	IBUFLB	8192	1	1	8	4256	2790	66
25	IBUFHA	OCMC RAM	8192	1	1	8	4256	2720	64
26	OCMC RAM	IBUFHA	8192	1	1	8	4256	2800	66
27	IBUFHB	OCMC RAM	8192	1	1	8	4256	2730	64
28	OCMC RAM	IBUFHB	8192	1	1	8	4256	2800	66

5.2 EVE EDMA Observations

NOTE: On the TDA2xx devices, both EVE transfer controllers yield identical performance for all transfer scenarios because both TC have the same configuration, and most importantly the same FIFOSIZE for a given burst size. The performance of the transfer controllers across multiple instantiation of EVEs in TDA2xx is also the same.

EDMA channel parameters allow many different transfer configurations. Typical transfer configurations result in transfer controllers bursting the read write data in default burst size chunks, thereby, keeping the busses fully utilized. However, in some configurations, the TC issues less than optimally sized read/write commands (less than default burst size), reducing performance. To properly design a system, it is important to know which configurations offer the best performance for high-speed operations.

On TDA2xx, there are two transfer controllers to move data between slave end points. The default configuration for the transfer controllers is shown in Table 27.

Table 27. EVE EDMA Configuration for the Transfer Controllers

Name	Description	TC0	TC1
TCCFG[2:0] FIFOSIZE	Channel FIFO Size	1024 Bytes	1024 Bytes
TCCFG[5:4] BUSWIDTH	Data Transfer Bus Width	16 Bytes	16 Bytes
TCCFG[9:8] DSTREGDEPTH	Destination Register Depth	4 entries	4 entries
DBS (Default Burst Size)	Size of each data burst	Configurable	Configurable

The individual TC performance for paging/memory to memory transfers is essentially dictated by the TC configuration. In most scenarios, the FIFOSIZE and default burst size configuration for the TC have the most significant impact on the TC performance; the BUSWIDTH configuration is dependent on the device architecture and the DSTREGDEPTH values impact the number of in-flight transfers.

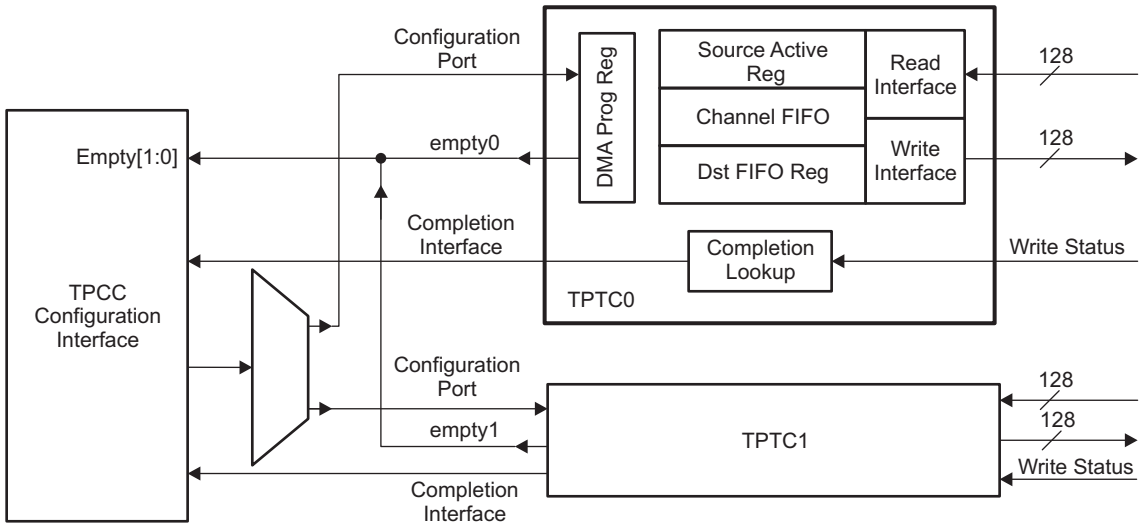


Figure 18. EVE EDMA Third-party Transfer Controller (EDMA_TPTC) Block Diagram

The DBS can be controlled with the register EVE_BUS_CONFIG. The default burst size of 128 bytes per burst generates the maximum TC throughput, as shown in Table 28.

Table 28. EVE_BUS_CONFIG

EVE System Address offset	0x0000 0014		
Physical Address EVE Internal	0x4008 0014	**Instance**	EVE_SYSTEM
Description	EVE Bus Configuration		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																		TC1_DBS		RESERVED		TC0_DBS		RESERVED				DBP_ENABLE	MAX_IN_FLIGHT		

Bits	Field Name	Description	Reset	Access When UNLOCKED	Access When LOCKED
31:14	RESERVED	Reserved	0x0	R	R
13:12	TC1_DBS	TC1 Default Burst size setting	0x3	RW	R
11:10	RESERVED	Reserved	0x0	R	R
9:8	TC0_DBS	TC0 Default Burst size setting	0x3	RW	R
7:5	RESERVED	Reserved	0x0	R	R
4	DBP_ENABLE	Program Cache Demand Based Prefetch enable	0x1	RW	R
3:0	MAX_IN_FLIGHT	Defines maximum number of OCP requests in flight. Can be reduced to limit the peak bandwidth for Software Directed Preload, which in turn may provide advantage to other EVE level (for example, EDMA) or system level initiators.	0x4	RW	R

The TC read and write controllers in conjunction with the source and destination register sets are responsible for issuing optimally-sized reads and writes to the slave endpoints. An optimally-sized command is defined by the transfer controller default burst size (DBS).

The EDMA_TPTC attempts to issue the largest possible command size as limited by the DBS value or the ABCNT_n[15:0] ACNT and ABCNT_n[31:16] BCNT value of the TR. EDMA_TPTC obeys the following rules: The read/write controllers always issue commands less than or equal to the DBS value. The first command of a 1D transfer command always aligns the address of subsequent commands to the DBS value.

Table 29 lists the TR segmentation rules that are followed by the EDMA_TPTC. In summary, if the ABCNT_n[15:0] ACNT value is larger than the DBS value, then the EDMA_TPTC breaks the ABCNT_n[15:0] ACNT array into DBS-sized commands to the source/destination addresses. Each ABCNT_n[31:16] BCNT number of arrays are then serviced in succession.

For BCNT arrays of ACNT bytes (that is, a 2D transfer), if the ABCNT_n[15:0] ACNT value is less than or equal to the DBS value, then the TR may be optimized into a 1D-transfer in order to maximize efficiency. The optimization takes place if the EDMA_TPTC recognizes that the 2D-transfer is organized as a single dimension (ABCNT_n[15:0] ACNT == BIDX_n) and the ACNT value is a power of 2.

Table 29. EVE EDMA TC Optimization Rules

ACNT ≤ DBS	ACNT is Power of 2	BIDX = ACNT	BCNT ≤ 1023	SAM/DAM = Increment	Description
Yes	Yes	Yes	Yes	Yes	Optimized
No	X	X	X	X	Not Optimized
X	No	X	X	X	Not Optimized
X	X	No	X	X	Not Optimized
X	X	X	No	X	Not Optimized
X	X	X	X	No	Not Optimized

In summary, Table 30 lists the factors that affect the EDMA performance.

Table 30. Factors Affecting EVE EDMA Performance

Factors	Impact	General Recommendation
Source/Destination Memory	The transfer speed depends on SRC/DST memory bandwidth.	Know the nature of the source and destination memory, specifically the frequency of operation and the bus width.
Transfer Size	Throughput is less for small transfers due to transfer overhead/latency.	Configure EDMA for larger transfer size as throughput, small transfer size is dominated by transfer overhead.
A-Sync/AB-Sync	Performance depends on the number of TRs (Transfer Requests). More TRs would mean more overhead.	Using AB-Sync transfers gives better performance than chaining A-Sync transfers.
Source/Destination Bidx	Optimization will not be done if BIDX is not equal to ACNT value optimization guidelines.	Whenever possible, follow the EDMA TC optimization guidelines. See the TPTC spec for optimization details.
Queue TC Usage	Performance is the same for both TCs.	Both TCs have the same configuration and show the same performance.
Burst Size	Decides the largest possible read/write command submission by TC.	The default burst size for all transfer controllers is 128 bytes. This also results in most efficient transfers/throughput in most memory-to-memory transfer scenarios.
Source/Destination Alignment	Slight performance degradation if source/destination are not aligned to Default Burst Size (DBS) boundaries.	For smaller transfers, as much as possible, source and destination addresses should be aligned across DBS boundaries.

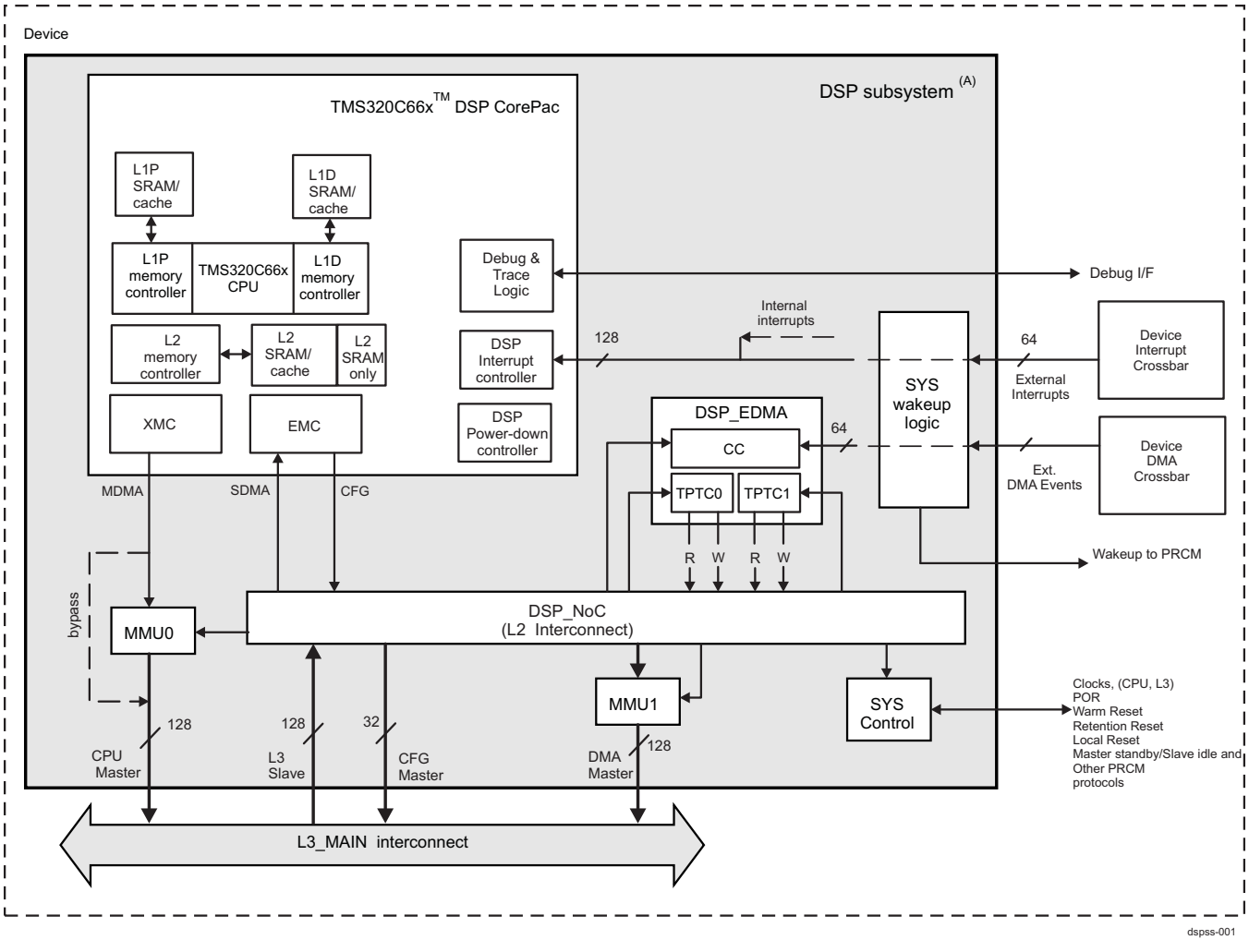
6 DSP CPU

C66x CorePac is the name used to designate the hardware that includes the following components: TMS320C66x DSP, Level 1 program (L1P) memory controller, Level 1 data (L1D) memory controller, Level 2 (L2) memory controller, Internal DMA (IDMA), external memory controller (EMC), extended memory controller (XMC), bandwidth management (BWM), interrupt controller (INTC) and power down controller (PDC).

The TMS320C66x DSP CorePac memory components include:

- A 32-KiB L1 program memory (L1P) configurable as cache and / or SRAM. When configured as a cache, the L1P is a 1-way set-associative cache with a 32-byte cache line
- A 32-KiB L1 data memory (L1D) configurable as cache and / or SRAM. When configured as a cache, the L1D is a 2-way set-associative cache with a 64-byte cache line
- A 288-KiB (program and data) L2 memory, only part of which is cacheable. When configured as a cache, the L2 memory is a 4-way set associative cache with a 128-byte cache line. Only 256 KiB of L2 memory can be configured as cache or SRAM. 32 KiB of the L2 memory is always mapped as SRAM.

The C66x DSP CorePac block in the DSP subsystem is shown in Figure 19.



A This diagram shows a single DSP instance. Each device may have one or two identical DSP instances. For more information, see the device-specific data manual.

Figure 19. DSP Subsystem Block Diagram and Clocking Structure

This section describes the DSP CPU read-write performance with no other traffic in the system. The three operations that the bandwidth is measured for are: a pipelined copy from source to destination buffer, pipeline read from the source buffer and pipelined write to the destination buffer. The CPU data path in the DSP Subsystem is from the CorePac XMC to the WC, optionally through the MMU0 and out of the DSP subsystem through the MDMA port.

6.1 DSP CPU Performance

The formulas used for the throughput calculations are:

Actual Throughput = (Transfer Size/Time Taken)

Ideal Throughput = Frequency of Limiting Port × Data Bus Width in Bytes

TC Utilization = (Actual Throughput/Ideal Throughput) × 100

6.1.1 DSP CPU Read and Write

The common system setup for the DSP CPU Read and Write throughput measurement is:

- DSP C66x clock: 600 MHz (unless specified)
- DDR clock: 532 MHz (unless specified)
- EMIF configuration
- CAS write latency – 6
- CAS latency - 7
- SDRAM Data Bus width : 32
- DDR in non-interleaved mode

The data presented is for stand-alone transfers with no other ongoing or competing traffic. All profiling has been done with C66x CorePac Timer operating at 600 MHz.

The theoretical bandwidth is calculated with the limiting port as the MDMA operating at 200 MHz. With this in mind, the theoretical bandwidth is calculated as 16 Bytes × 200 MHz = 3200 MB/s.

6.1.2 Code Setup

The source and destination buffers from and to which the DSP CPU read and write throughput would be measured are placed in the system memory (DDR/OCMC RAM). Rest of the code, stack, global variables, constants, and so on, are placed in the L2 RAM to avoid any other traffic in the system other than the system buffer access.

The compiler version used during the measurements is TI Code Gen Tool v7.4.4. Target processor version is set to 6600 (-mv6600).

The pipeline copy, read and write functions are optimized with the code optimization level 3 setting (–O3) and the space optimization of level 3 (-ms3). Additionally, all debug symbols are suppressed by setting --symdebug:none.

In order to statically debug the loops for the iterative copy/read/write, the following compiler option was used:

- --debug_software_pipeline or –mw: To generate verbose software pipelining information.
- --keep_asm or –k: Keep the generated assembly language (.asm) file.

The aim of the optimized code is to ensure that the two load/store engines in the C66x CPU is occupied every cycle of the loop performing 64-bit loads or stores.

With this in mind, the following sections provide some sample codes and their disassembly to help understand the pipeline copy, read and write functions used in the throughput measurements.

The following sections analyze the copy, read and write loops as scheduled iterations and pipelined loops. You are encouraged to go through the following link to understand the basics of software pipelined loops on the C6000 architecture: C6000 Compiler: Tuning Software Pipelined Loops.

6.1.2.1 Pipeline Copy

The C code for the pipeline copy function is:

```
extern volatile float wrDuration;
void pipeline_copy(int byte_cnt)
{
     long long *restrict dst = (long long *)ext_buf[1];
     long long *restrict src = (long long *)ext_buf[0];
     unsigned int wrStartTime, wrStopTime;
     int i;
     _nassert((int)dst == 0);
     _nassert((int)src == 0);
     wrStartTime = CSL_tscRead();
     for (i=0; i<byte_cnt/8; i++) {
    dst[i] = src[i];
    }
    WBINVALIDATE
    wrStopTime = CSL_tscRead();
    wrDuration = (float)(wrStopTime-wrStartTime)/(DSP_FREQ/1000);
}
```

The analysis of the scheduled iteration is given out by the compiler as:

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../pipeline_loop.c
;*      Loop source line                : 37
;*      Loop opening brace source line  : 44
;*      Loop closing brace source line  : 46
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound    : 1
;*      Partitioned Resource Bound(*)   : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     0        0
;*      .D units                     1*       1*
;*      .M units                     0        0
;*      .X cross paths               0        1*
;*      .T address paths             1*       1*
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        1     (.L or .S unit)
;*      Addition ops (.LSD)          0        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             0        1*
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 1  Schedule found with 7 iterations in parallel
;*----------------------------------------------------------------------------*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C330:
;*   0              LDDW    .D1T1   *A3++,A5:A4       ; |45|
;*   1              NOP             4
;*   5              DADD    .L2X    0,A5:A4,B5:B4     ; |45| Define a twin register
;*   6              STDW    .D2T2   B5:B4,*B6++       ; |45|
;*     ||           SPBR            $C$C330
;*   7              ; BRANCHCC OCCURS {$C$C330}       ; |37|
;*----------------------------------------------------------------------------*
```

The pipeline can be viewed as in Figure 20. It can be observed that once the loop prologue (pipe up) completes, the code would keep the two 64-bit load and store engines occupied every cycle until the loop begins to pipe down.

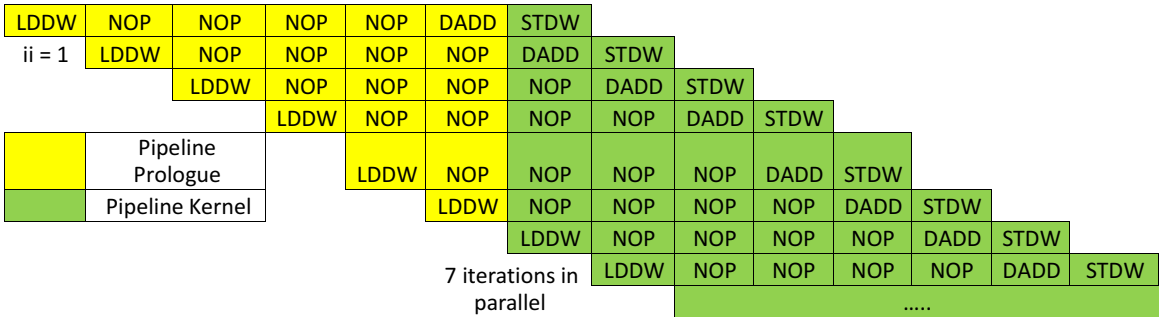


Figure 20. DSP CPU Pipeline Copy Software Pipelining

6.1.2.2 Pipeline Read

The C code for the pipeline read function is:

```
extern volatile float wrDuration;
long long temp1, temp2, temp3, temp4;
void pipeline_read(unsigned byte_cnt)
{
    long long *restrict src = (long long *)ext_buf[0];
    unsigned int wrStartTime, wrStopTime;
    int i;
    wrStartTime = CSL_tscRead();
    for (i=0; i<byte_cnt/8; i+=4)
    {
      temp1 = src[i];
      temp2 = src[i+1];
      temp3 = src[i+2];
      temp4 = src[i+3];
    }
    wrStopTime = CSL_tscRead();
    WBINVALIDATE
    wrDuration = (float)(wrStopTime-wrStartTime)/(DSP_FREQ/1000);
}
```

The analysis of the scheduled iteration is given out by the compiler as:

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../pipeline_loop.c
;*      Loop source line                : 59
;*      Loop opening brace source line  : 66
;*      Loop closing brace source line  : 75
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 1
;*      Unpartitioned Resource Bound    : 2
;*      Partitioned Resource Bound(*)   : 2
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     0        0
;*      .D units                     2*       2*
;*      .M units                     0        0
;*      .X cross paths               0        1
;*      .T address paths             2*       2*
;*      Long read paths              0        0
```

```
;*      Long write paths             0        0
;*      Logical  ops (.LS)           1        0     (.L or .S unit)
;*      Addition ops (.LSD)          0        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1        0
;*      Bound(.L .S .D .LS .LSD)     1        1
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Schedule found with 2 iterations in parallel
;*----------------------------------------------------------------------------*
;*       SETUP CODE
;*
;*                  MV              A8,B8
;*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C201:
;*   0              LDDW    .D1T1   *+A8(16),A7:A6    ; |72|  ^
;*   1              LDDW    .D1T1   *+A8(24),A5:A4    ; |73|  ^
;*     ||           ADD     .L1     A3,A8,A8          ;  ^
;*   2              LDDW    .D2T2   *B8,B7:B6         ; |67|  ^
;*   3              LDDW    .D2T2   *+B8(8),B5:B4     ; |71|  ^
;*     ||           MV      .L2X    A8,B8             ;  ^ Define a twin register
;*     ||           SPBR            $C$C201
;*   4              ; BRANCHCC OCCURS {$C$C201}       ; |59|
;*----------------------------------------------------------------------------*
```

The pipeline can be viewed as in Figure 21. It can be observed that once the loop prologue (pipe up) completes, the code would keep the two 64-bit load engines occupied every cycle until the loop begins to pipe down.

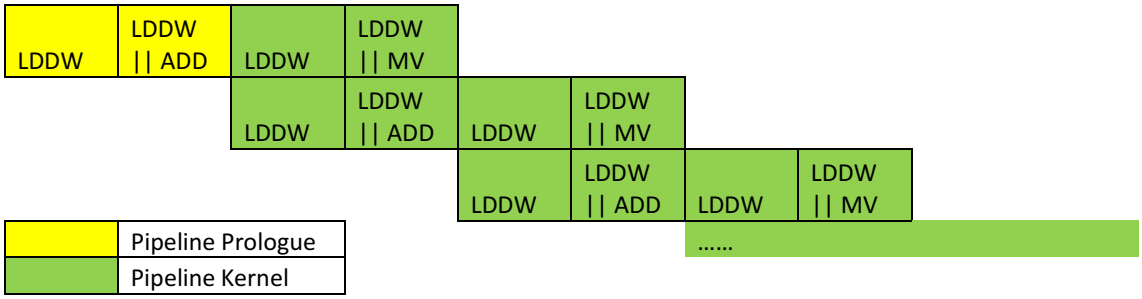


Figure 21. DSP CPU Pipeline Read Software Pipelining

6.1.2.3 *Pipeline Write*

The C code for the pipeline write function is:

```
void pipeline_write(unsigned byte_cnt)
{
    long long *restrict dst = (long long *)ext_buf[1];
    unsigned int wrStartTime, wrStopTime;
    int i;
    _nassert((int)dst == 0);
    wrStartTime = CSL_tscRead();
    #pragma UNROLL(2)
    for (i=0; i<byte_cnt/8; i++)
    {
      dst[i] = 0xDEADDEAD;
    }
    wrStopTime = CSL_tscRead();
    WBINVALIDATE
    wrDuration = (float)(wrStopTime-wrStartTime)/(DSP_FREQ/1000);
}
```

The analysis of the scheduled iteration is given out by the compiler as:

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../pipeline_loop.c
;*      Loop source line                 : 89
;*      Loop opening brace source line   : 96
;*      Loop closing brace source line   : 98
;*      Loop Unroll Multiple             : 2x
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 1
;*      Partitioned Resource Bound(*)    : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     0        0
;*      .D units                     1*       1*
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             1*       1*
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          0        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             0        0
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 1  Schedule found with 2 iterations in parallel
;*----------------------------------------------------------------------------*
;*       SETUP CODE
;*
;*                 MV              B6,A3
;*                 ADD             8,A3,A3
;*                 MV              A4,B4
;*                 MV              A5,B5
;*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C94:
;*   0              STDW    .D2T2   B5:B4,*B6++(16)   ; |97|
;*     ||           STDW    .D1T1   A5:A4,*A3++(16)   ; |97|
;*     ||           SPBR            $C$C94
;*   1              NOP             1
;*   2              ; BRANCHCC OCCURS {$C$C94}        ; |89|
;*----------------------------------------------------------------------------*
```

The pipeline can be viewed as in Figure 22. It can be observed that this code would make sure the two store engines are occupied every cycle of the pipeline.

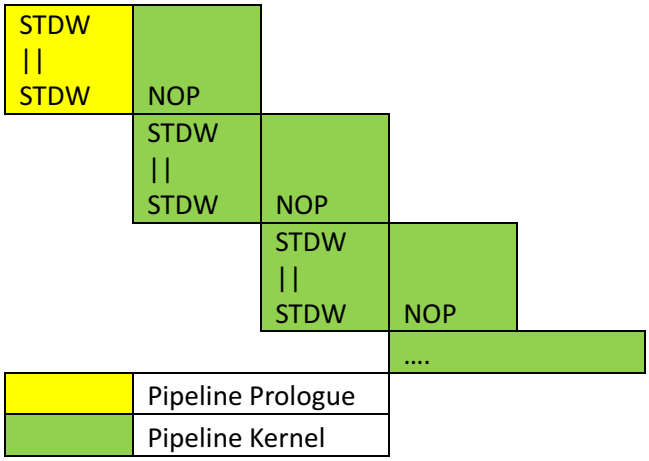


Figure 22. DSP CPU Pipeline Write Software Pipeline

The CGEM (C66x CorePac) L2 cache controller can get up to 4 L2 line allocations in flight. Each allocation brings in 128 bytes. The XMC can get an additional 8 pre-fetch requests (also for 128 bytes) in flight. In the best case, L2 + XMC can get 12 × 128 = 1.5K bytes worth of requests outstanding at once.

NOTE: In terms of bus requests, the L2 cache controller and XMC actually make 64 byte requests, so this is actually 24 64-byte requests in the best case.

In steady state, however, this should drop to 8 × 128 = 1K bytes total in-flight, because XMC only sends additional pre-fetches in response to pre-fetch hits. This is because, the XMC only sends new pre-fetches in two cases: (1) on recognizing a new stream, and (2) on getting hits to an existing stream. In steady state, there are no new streams, so you are only in case (2). In the "100% pre-fetch hit" steady state case, the L2 misses will all hit in the XMC pre-fetch buffer and stop there, and the only traffic leaving XMC will be additional pre-fetches. Thus, the total number of outstanding requests is limited to the total number of outstanding pre-fetches.

To get higher performance, you would need an optimized copy loop that can get 4 L2 misses pipelined up as much as possible, or if XMC pre-fetch is enabled, at least 4 pre-fetch streams active to maximize out the DSP Subsystem busses. In order to emulate this behavior, the following functions were defined that read or write the first 64-bit word of the L2 cache line of 128 bytes. Since they access only the first word of the L2 cache line, they are named L2 Stride-Jmp Copy, L2 Stride-Jmp Read, and L2 Stride-Jmp Write, respectively.

6.1.2.4 L2 Stride-Jmp Copy

Following is the C code for the L2 Stride-Jmp Copy function. The difference from the earlier pipeline copy is as highlighted.

```
void l2_stride_jmp_copy(int byte_cnt)
{
    long long *restrict dst = (long long *)ext_buf[1];
    long long *restrict src = (long long *)ext_buf[0];
    unsigned int wrStartTime, wrStopTime;
    int i;
    _nassert((int)dst == 0);
    _nassert((int)src == 0);
    wrStartTime = CSL_tscRead();
    for (i=0; i<byte_cnt/8;  i+=16 )
    {
      dst[i] = src[i];
    }
```

```
    wrStopTime = CSL_tscRead();
    WBINVALIDATE
    wrDuration = (float)(wrStopTime-wrStartTime)/(DSP_FREQ/1000);
}
```

Following is the analysis of the scheduled iteration given out by the compiler. The pipeline remains the same as with the pipeline copy with addition of the parallel add operations with the LDDW and STDW.

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../pipeline_loop.c
;*      Loop source line                : 37
;*      Loop opening brace source line  : 44
;*      Loop closing brace source line  : 46
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 1
;*      Unpartitioned Resource Bound    : 1
;*      Partitioned Resource Bound(*)   : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     0        0
;*      .D units                     1*       1*
;*      .M units                     0        0
;*      .X cross paths               1*       1*
;*      .T address paths             1*       1*
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           1        2     (.L or .S unit)
;*      Addition ops (.LSD)          0        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1*       1*
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 1  Schedule found with 7 iterations in parallel
;*----------------------------------------------------------------------------*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C325:
;*   0              LDDW    .D1T1   *A3,A5:A4         ; |45|  ^
;*     ||           ADD      .L1X     B6,A3,A3          ;  ^
;*   1              NOP             4
;*   5              DADD    .S2X    0,A5:A4,B5:B4     ; |45| Define a twin register
;*   6              STDW    .D2T2   B5:B4,*B7         ; |45|  ^
;*     ||           ADD      .L2     B6,B7,B7          ;  ^
;*     ||           SPBR            $C$C325
;*   7              ; BRANCHCC OCCURS {$C$C325}       ; |37|
;*----------------------------------------------------------------------------*
```

6.1.2.5 L2 Stride-Jmp Read

Following is the C code for the L2 Stride-Jmp Read function. The difference from the earlier pipeline read is as highlighted.

```
long long temp1, temp2, temp3, temp4;
void l2_stride_jmp_read(unsigned byte_cnt)
{
    long long *restrict src = (long long *)ext_buf[0];
    unsigned int wrStartTime, wrStopTime;
    int i;
    wrStartTime = CSL_tscRead();
    for (i=0; i<byte_cnt/8;  i+=32  )
    {
      temp1 = src[i];
      temp2 = src[i+16];
    }
    wrStopTime = CSL_tscRead();
    WBINVALIDATE
    wrDuration = (float)(wrStopTime-wrStartTime)/(DSP_FREQ/1000);
}
```

The pipeline can be viewed as in Figure 23. It can be observed that this code would make sure the two load engines are occupied every cycle of the pipeline.

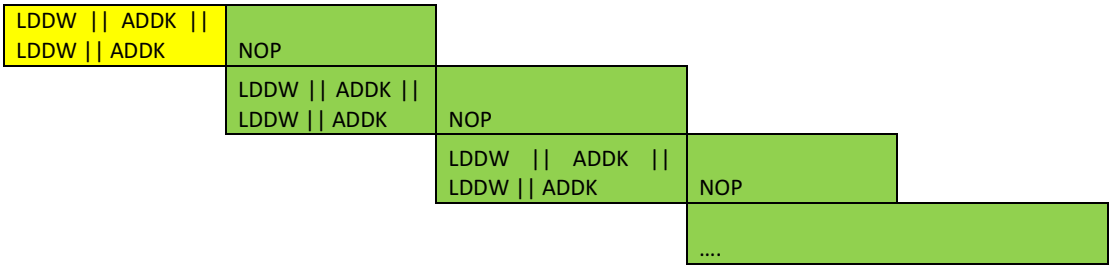


Figure 23. DSP CPU Pipeline L2 Stride-Jmp Read Software Pipelining

6.1.2.6 L2 Stride-Jmp Write

Following is the C code for the L2 Stride-Jmp Write function. The difference from the earlier pipeline read is as highlighted.

```
void l2_stride_jmp_write(unsigned byte_cnt)
{
    long long *restrict dst = (long long *)ext_buf[1];
    unsigned int wrStartTime, wrStopTime;
    int i;
    _nassert((int)dst == 0);
    wrStartTime = CSL_tscRead();
    #pragma UNROLL(2)
    for (i=0; i<byte_cnt/8; i+=16 )
    {
      dst[i] = 0xDEADDEAD;
    }
    wrStopTime = CSL_tscRead();
    WBINVALIDATE
    wrDuration = (float)(wrStopTime-wrStartTime)/(DSP_FREQ/1000);
}
```

Following is the analysis of the scheduled iteration is given out by the compiler. The pipeline is the same as the pipeline write function with the difference of additional ADD operations happening in parallel with the two store operations.

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../pipeline_loop.c
;*      Loop source line                : 88
;*      Loop opening brace source line  : 95
;*      Loop closing brace source line  : 97
;*      Loop Unroll Multiple            : 2x
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 1
;*      Unpartitioned Resource Bound    : 1
;*      Partitioned Resource Bound(*)   : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     1*       1*
;*      .D units                     1*       1*
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             1*       1*
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          0        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1*       1*
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 1  Schedule found with 2 iterations in parallel
;*----------------------------------------------------------------------------*
;*       SETUP CODE
;*
;*                 MV              A4,B4
;*                 MV              A5,B5
;*
;*        SINGLE SCHEDULED ITERATION
;*
;*        $C$C94:
;*   0              STDW    .D2T2   B5:B4,*B6         ; |96|  ^
;*     ||           ADDK     .S2     256,B6           ; |96|  ^
;*     ||           STDW    .D1T1   A5:A4,*A3         ; |96|  ^
;*     ||           ADDK     .S1     256,A3           ; |96|  ^
;*     ||           SPBR            $C$C94
;*   1              NOP             1
;*   2              ; BRANCHCC OCCURS {$C$C94}        ; |88|
;*----------------------------------------------------------------------------*
```

6.2 DSP CPU Observations

Figure 24 gives the average DSP bandwidth in MBps (y-axis) measured for the different functions introduced above, for different data sizes of 16 KiB, 64 KiB, 128 KiB, 256 KiB, and 8 MiB (x-axis) for the L1D and L2 cache size of 32 K and 128 K, respectively. The given bandwidth was measured with prefetch enable, MMU off, and L1D write back policy enabled. Each cache line fetch for 128 bytes is actually two VBUS commands for 64 bytes.

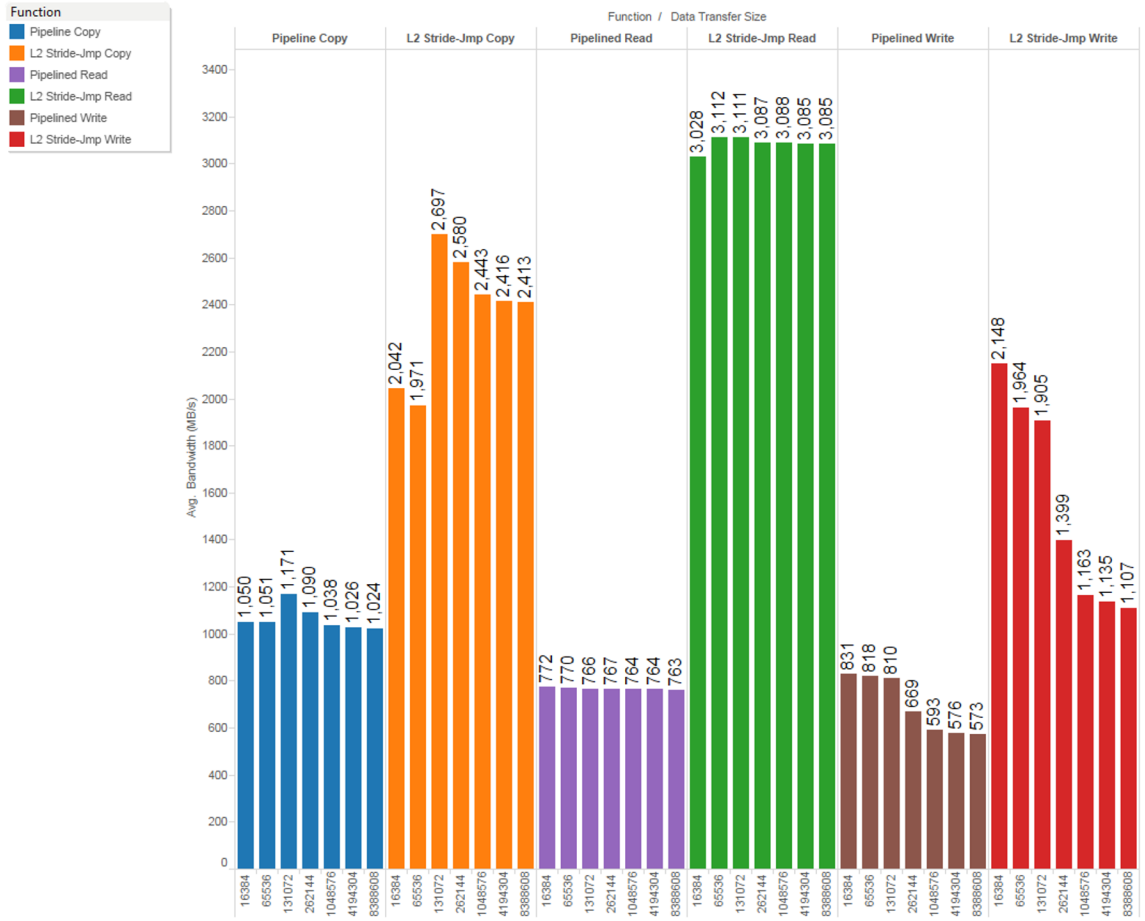


Figure 24. DSP CPU Read and Write Performance With Different Data Sizes to DDR

For a memcpy() type operation, there are both reads and writes. Furthermore, the L2 cache write-allocates. For buffer sizes that fit entirely within L2, the traffic at the MDMA boundary will look like two streams of reads. For buffer sizes larger than L2, there is a third stream consisting of victim writes. That's why the numbers start falling off as the data sizes get above 128K. The reads do not show this trend as the cache lines do not become dirty and the cache would not perform a write back of the cache line when the data sizes are larger than the cache line.

The L2 pipeline functions generate more L2 cache line fetches and write backs in a shorter time span leading to a higher throughput.

The L2 memory controller conveys to the XMC whether a given address range is pre-fetchable. This information comes directly from the “PFX” field in the corresponding MAR register. Figure 25 shows the effect of pre-fetch ON versus OFF for DDR transfers with MMU off, MDMA Posted writes and L1D write back policy enabled. The XMC pre-fetcher does not distinguish read-allocate from write-allocate; it will try to pre-fetch for either to speed things up as seen by the ~2x performance increase with pre-fetch ON versus OFF for both read and write streams.

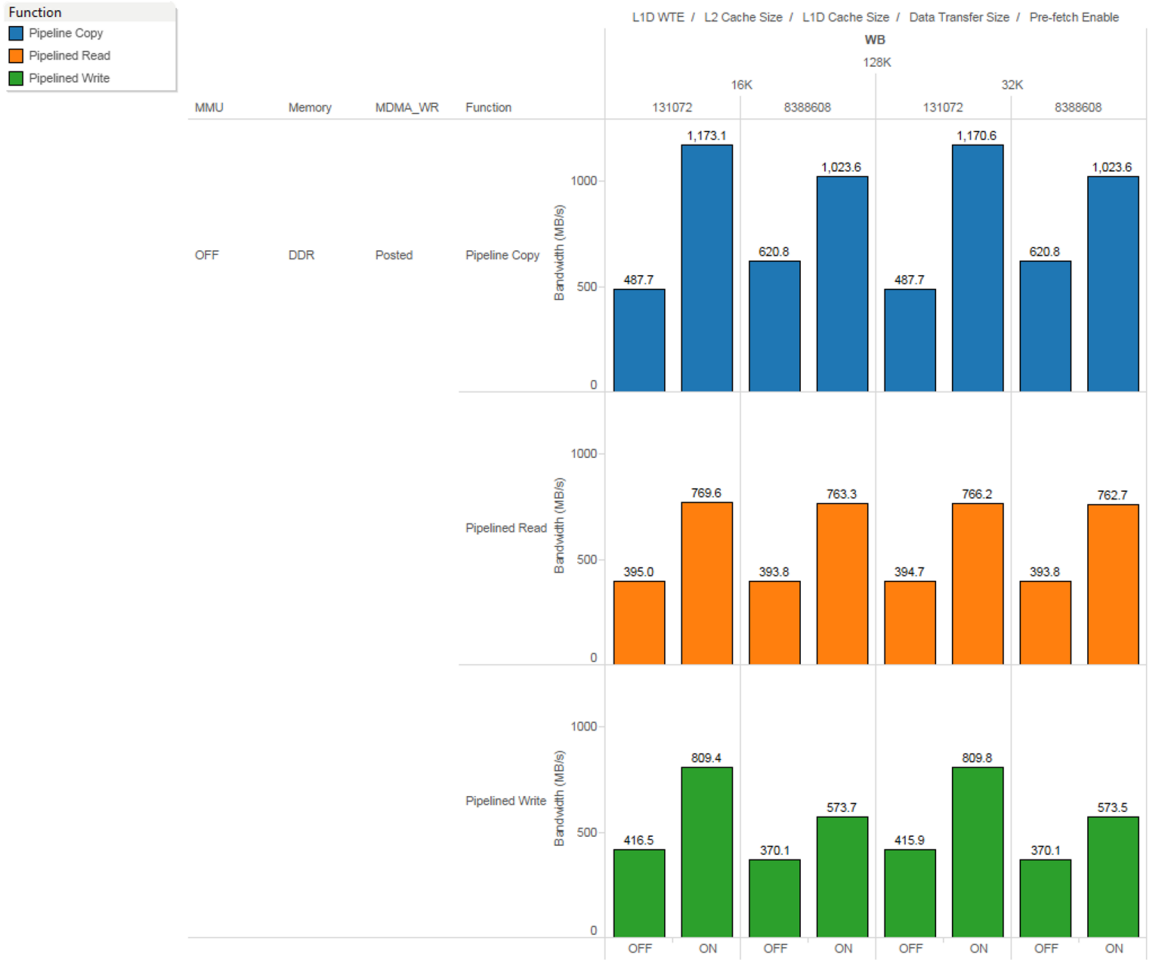


Figure 25. Impact on Prefetch Enable versus Disable on CPU Performance

The DSP CPU read and writes throughput varies with the source and the destination of the buffer. Figure 26 shows the difference in bandwidth obtained when the data is transferred from DDR-to-DDR versus OCMC RAM-to-OCMC RAM for different data transfer sizes, with pre-fetch enabled, L2 cache size of 128K, and L1D of 32K with L1D write back policy enabled, MMU off and non-posted writes at the MDMA boundary for cached data.

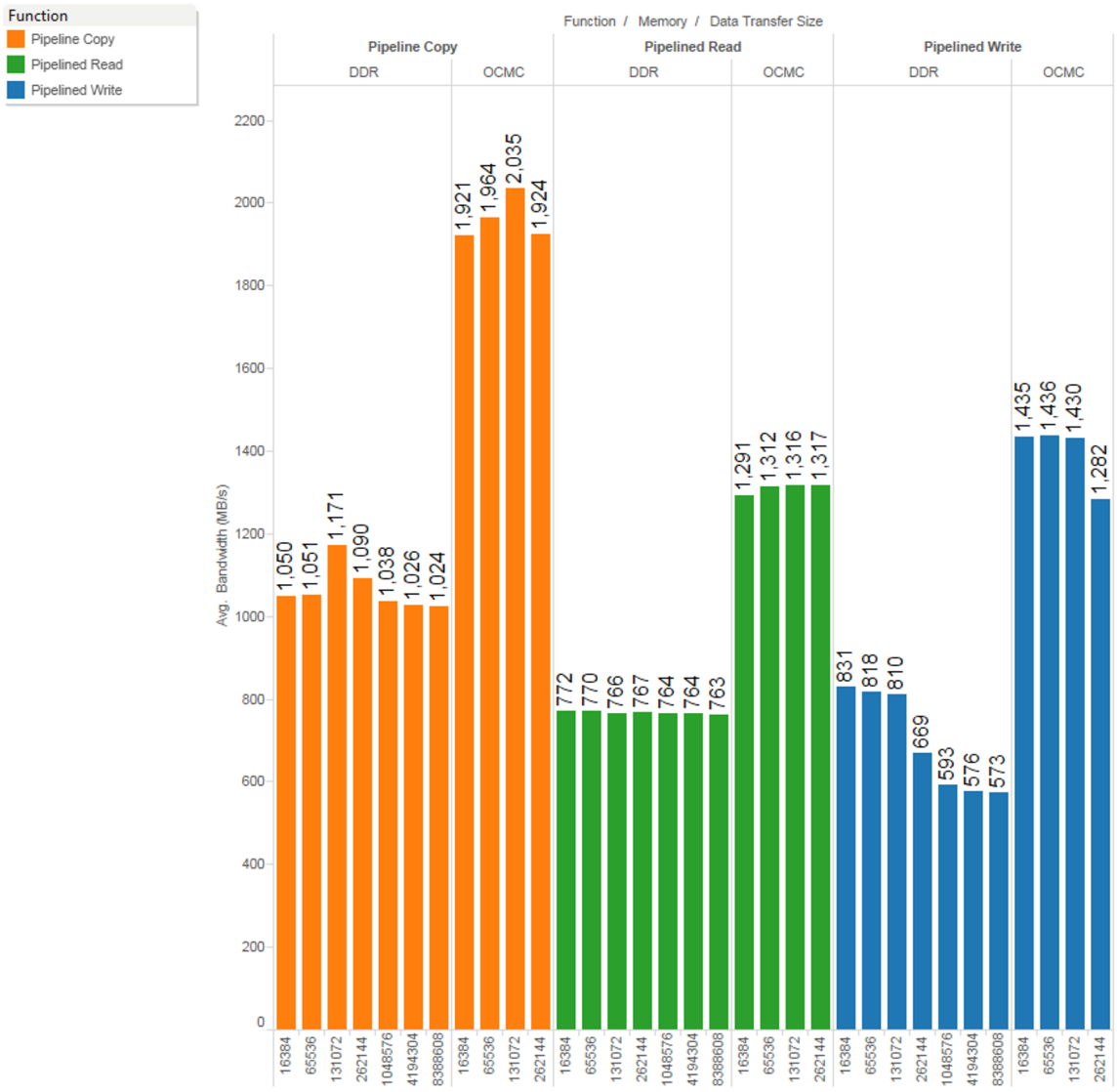


Figure 26. Impact of Source and Destination Memory on DSP CPU RD-WR Performance

A standalone memory management unit (DSP_MMU0) is included within the DSP1 (DSP1_MMU0) and DSP2 (DSP2_MMU0) subsystems boundaries. The DSP_MMU0 is integrated on the C66x CPU MDMA path to the device L3_MAIN interconnect. This provides several benefits including protection of the system memories from corruption by DSP1 and DSP2 accidental accesses. Figure 27 shows the effect of MMU off versus MMU on, with pre-fetch enabled, L2 cache size of 128K, and L1D of 32K with L1D write back policy enabled and posted writes at the MDMA boundary for cached data. The MMU adds to the latency in the path leading to slight drop in the throughput. (16MB Page size in TLB)

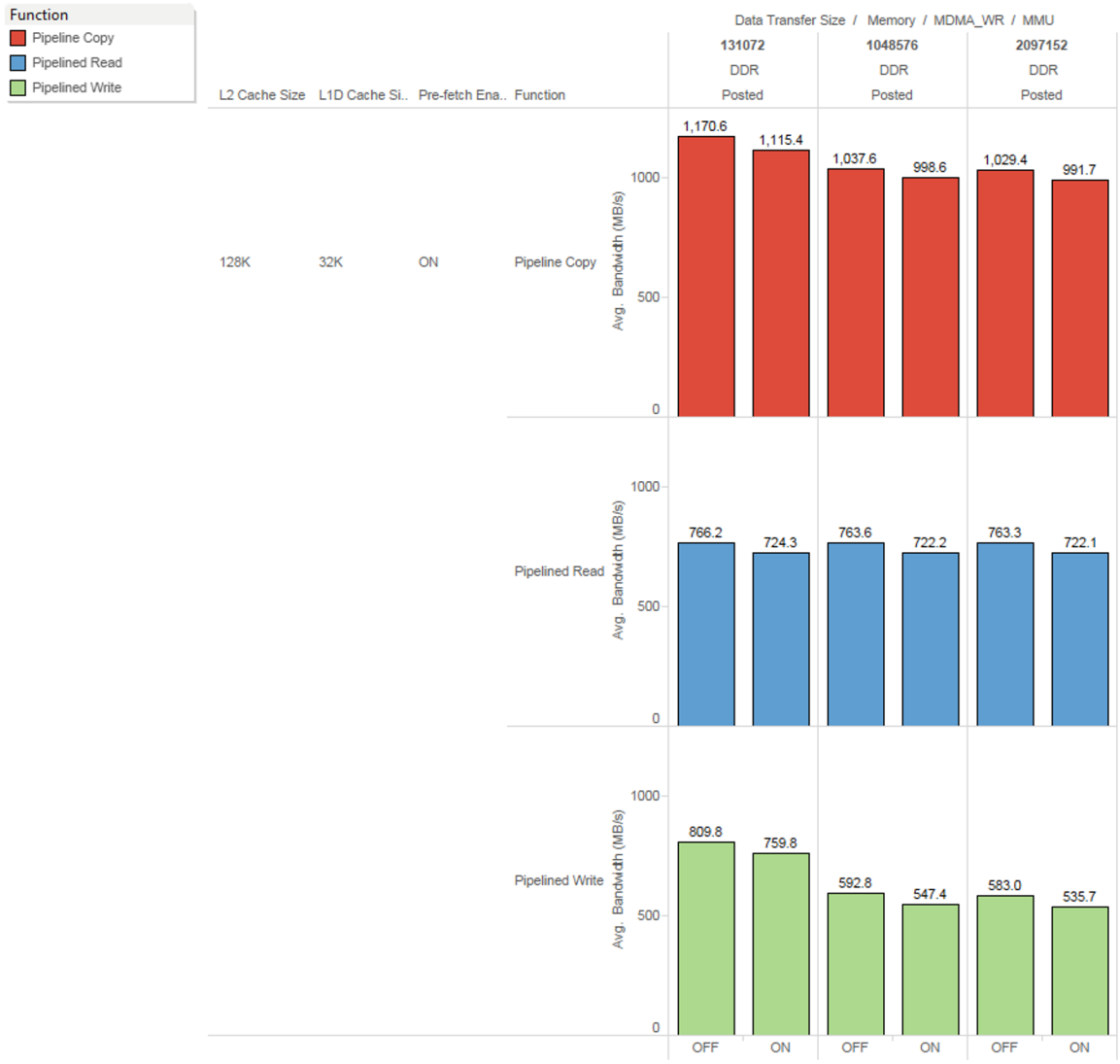


Figure 27. Impact of MMU Enable on DSP RD-WR Performance

The C66x CorePac submits writes denoted as either “cacheable” or non-cacheable. Write accesses that are non-cacheable will be submitted as interconnect (L3_MAIN) non-posted writes; whereas, write accesses that are cacheable are submitted as interconnect posted writes based on the configuration of the C66xOSS_BUS_CONFIG. Figure 28 gives the comparison of the posted versus non-posted writes when measuring bandwidth of the cache flush operation while transferring data to DDR with pre-fetch enabled, L2 cache size of 128K, and L1D of 32K with L1D write back policy enabled.

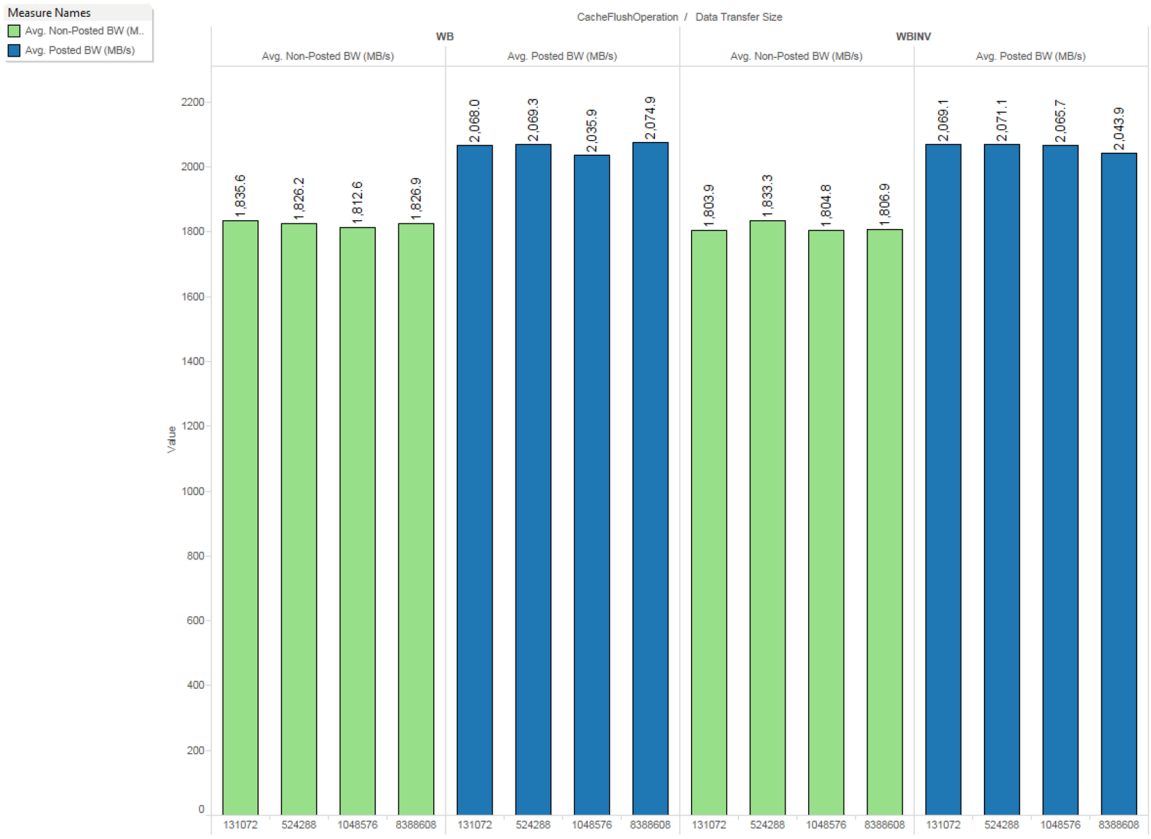


Figure 28. Impact of Posted and Non-Posted Writes on DSP Cache Flush

6.3 Summary

Based on the observations made in Section 6.2, Table 31 lists the factors that affect the DSP CPU RD WR performance.

Table 31. Factors Affecting DSP CPU RD-WR Performance

Factors	Impact	General Recommendation
Source/Destination Memory	The transfer speed depends on SRC/ DST memory bandwidth.	Know the nature of the source and destination memory, specifically the frequency of operation and the bus width.
Transfer Size versus Cache Size	Larger data buffers written to than the cache size introduces a L2 cache line write back along with the L2 cache line reads at write allocate at the MDMA port.	Expect drop in performance when the data buffer size written to is larger than the L2 cache size.
Code Optimization	The more the load and store units in the DSP core are occupied the better will be the CPU read and write performance.	You would need an optimized copy loop that can get 4 L2 misses pipelined up as much as possible, or if XMC pre-fetch is enabled, at least 4 pre-fetch streams active to max out the DSP Subsystem MDMA bus. Use the compiler options and pipelined loops to achieve this.
MAR Register Pre-fetch Enable	Improves the CPU RD-WR throughput.	Enable pre-fetch in the MAR register for better CPU RD-WR throughput.
C66xOSS_BUS_CONFIG: MDMA posted versus non-posted writes	Posted writes give better performance than the non-posted writes.	Enable posted writes whenever you do not expect race conditions when the data would be read even before the memory gets updated.
MMU Enable	Enabling MMU leads to slight drop in CPU RD-WR throughput.	
MAR Register Cache ability	Improves the CPU RD-WR performance when regions are made cacheable.	Set the MAR cacheable bit for regions accessed by the DSP CPU.
Maximizing cache line reuse	Improves the CPU RD-WR performance.	The same memory locations within a cached line should be reused as often as possible. Either the same data can be reread or new data written to already cache locations so that subsequent reads will hit.
Eviction of a line	Avoiding eviction of a line as long as it is being reused improves the CPU RD-WR performance.	
Stall cycles per miss	Reducing the number of stall cycles per miss improves the CPU RD-WR performance.	This can be achieved by exploiting miss pipelining.

7 Cortex-M4 (IPU)

IPU Subsystem, Figure 29, is the name used to designate the hardware that includes the following components: Dual ARM Cortex-M4 CPUs, Level 1 Unicache, L1 MMU, L2 MMU, L2 ROM, L2 RAM, and interconnect.

Main features of IPU subsystem:

- Dual Cortex-M4 cores
- Interrupt controller integrated per Cortex-M4 routing up to 80 interrupt events (external and internal) including NMI and reset
- 32 KB L1 cache (Shared cache/Unicache) with internal AMMU for attribute and internal address translation.
- 16 KB L2 ROM
- 64 KB L2 RAM
- L2 MMU (32 entries) at IPU L3 Master Port with Table Walking Logic
- One OCP Initiator and One OCP Target port to L3 interconnect

NOTE: The 32-bit Master ISS OCP port and ISS bridge in Figure 29 are not available in the TDA2xx and TDA2ex class of devices.

7.1 Cortex-M4 CPU Performance

7.1.1 Cortex-M4 CPU Read and Write

The common system setup for the Cortex-M4 CPU Read and Write throughput measurement is:

- Cortex-M4 clock: 212.8 MHz
- GP timer clock: 20 MHz
- Program code running from L2 RAM
- Optimization enabled (-O3)
- Word size 32-bit/64-bit

7.1.2 Code Setup

The source and destination buffers from and to which the Cortex-M4 read and write throughput would be measured are placed in the system memory (DDR/OCMC RAM). Rest of the code, stack, global variables, constants, and so on, are placed in the L2 RAM to avoid any other traffic in the system other than system buffer access.

The compiler version used during the measurements is TI Code Gen Tool v5.0.4. Target processor version is set to Generic Cortex-M4 (-mv7M4).

The copy, read and write functions are optimized with the code optimization level 3 setting (–O3) and the space optimization of level 3 (-ms3).

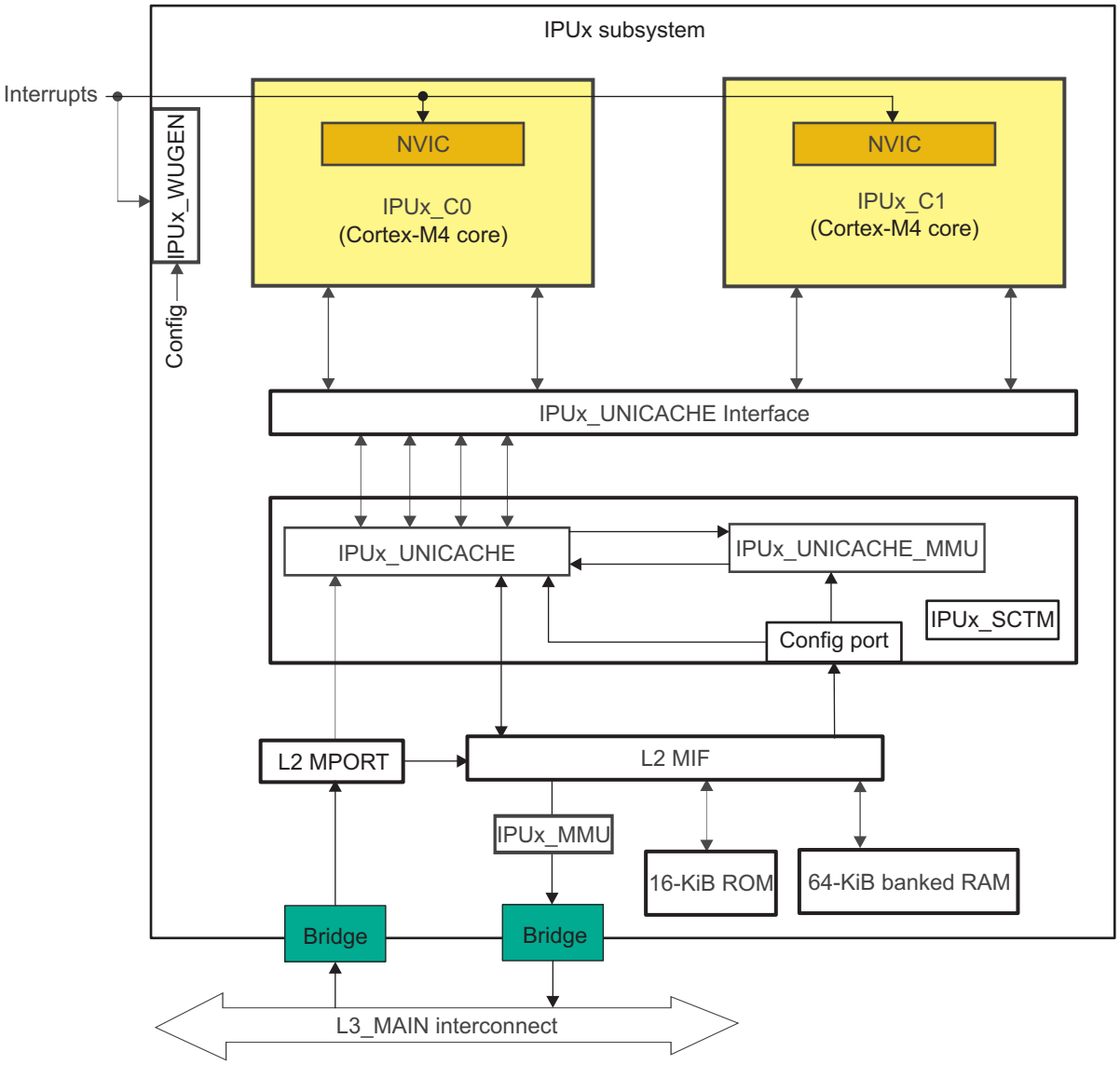


Figure 29. IPU Block Diagram

7.1.3 Cortex-M4 Functions

Following are the three functions used for Cortex-M4 operations.

NOTE: The read, write, and copy functions use unrolled loops as it will generate more optimized code. With loop unrolled, it will check for loop condition after every 128 words (read or write) or 32 words (copy) transfer. Without loop unrolled, it will check for loop condition after each word transfer so it will generate less optimized code.

The C code for the write function is:

```
void memWrite(UWORD32 DstAddr, UWORD32 transSize) {
    register UWORD32 wrData = 0xA5B5C5D5;
    register UWORD32 i_wr;
    register volatile UWORD32* wrAddr;
    wrAddr = (UWORD32 *)DstAddr;
    for(i_wr=0; i_wr<transSize; i_wr+=128){
        /*128 words increment */
        *wrAddr++ = wrData     /*word 1*/
        *wrAddr++ = wrData     /*word 2*/
        *wrAddr++ = wrData     /*word 3*/
        *wrAddr++ = wrData     /*word 4*/
        *wrAddr++ = wrData     /*word 5*/
        ...
        ...
        ...
        *wrAddr++ = wrData     /*word 127*/
        *wrAddr++ = wrData     /*word 128*/
    }
}
```

The C code for the read function is:

```
void memRead(UWORD32 SrcAddr, UWORD32 transSize) {
    register UWORD32 rdData;
    register UWORD32 i_rd;
    register volatile UWORD32* rdAddr;
    rdAddr = (UWORD32 *)SrcAddr;
    for(i_rd=0; i_rd<transSize; i_rd+=128){
        /*128 words increment */
        rdData = *rdAddr++      /*word 1*/
        rdData = *rdAddr++      /*word 2*/
        rdData = *rdAddr++      /*word 3*/
        rdData = *rdAddr++      /*word 4*/
        rdData = *rdAddr++      /*word 5*/
        ...
        ...
        ...
        rdData = *rdAddr++      /*word 127*/
        rdData = *rdAddr++      /*word 128*/
    }
}
```

The C code for the copy function is:

```
void memCopy(UWORD32 SrcAddr, UWORD32 DstAddr, UWORD32 transSize) {
    register volatile UWORD32* rdAddr, *wrAddr;
    register UWORD32 i;
    rdAddr = (UWORD32 *)SrcAddr;
    wrAddr = (UWORD32 *)DstAddr;
    for(i=0; i<transSize; i=32){
        /*32 words increment */
        *wrAddr++ = *rdAddr++      /*word 1*/
        *wrAddr++ = *rdAddr++      /*word 2*/
        *wrAddr++ = *rdAddr++      /*word 3*/
        *wrAddr++ = *rdAddr++      /*word 4*/
        *wrAddr++ = *rdAddr++      /*word 5*/
        ...
        ...
        ...
        *wrAddr++ = *rdAddr++      /*word 31*/
        *wrAddr++ = *rdAddr++      /*word 32*/
    }
}
```

7.1.4 Setup Limitations

In case of cache enabled and write back write allocate cache policy, the net amount of reads/writes to the main memory (DDR3) will be greater or lesser than the intended data size; In this case, the performance measurement is mostly based on the time taken for the intended size read/write/copy and not the actual data size. GP Timer 3 is easy to use and widely used for profiling; however, this timer runs only at 20 MHz so there will be a minor difference in the accuracy. For large transfer sizes like 4MB, the difference is negligible.

7.2 Cortex-M4 CPU Observations

Cortex-M4 CPU write, read, and copy performance varies with the source and the destination of the buffer. Table 32 and Table 33 show the difference between the bandwidth obtained when the data is transferred from DDR-to-DDR versus OCMC RAM-to-OCMC RAM. OCMC RAM gives better bandwidth performance than DDR memory.

7.2.1 Cache Disable

Table 32. IPU RD, WR, COPY Performance With Cache Disabled

Initiator/Operation	Source	Destination	Size (KB)	Bandwidth (MB/s)
M4 WR	CPU Register	DDR	4096	175.68
M4 RD	DDR	CPU Register	4096	13.98
M4 COPY	DDR	DDR	4096	22.38
M4 WR	CPU Register	OCMC	128	175.18
M4 RD	OCMC	CPU Register	128	28.87
M4 COPY	OCMC	OCMC	128	100.4

7.2.2 Cache Enable

Table 33. IPU RD, WR, COPY Performance With Cache Enabled (Policy: Write-Back, No Write-Allocate), 32-bit Word Size

Initiator/Operation	Source	Destination	Size (KB)	Bandwidth (MB/s)
M4 WR	CPU Register	DDR	4096	268.03
M4 RD	DDR	CPU Register	4096	93.59
M4 COPY	DDR	DDR	4096	111.39
M4 WR	CPU Register	OCMC	128	276.12
M4 RD	OCMC	CPU Register	128	158.76
M4 COPY	OCMC	OCMC	128	177.34

Table 34. Impact of Different Cache Policies on IPU CPU Performance, 32-bit Word Size

Initiator/ Operation	Source	Destination	Size (KB)	Write-Back, Write Allocate	Write-Back, No-Write Allocate	Write-Through, Write Allocate	Write-Through, No-Write Allocate
M4 WR	CPU Register	DDR	4096	75.2	268.03	267.98	267.03
M4 RD	DDR	CPU Register	4096	93.48	93.59	93.15	93.61
M4 COPY	DDR	DDR	4096	80.56	111.39	110.44	111.42
M4 WR	CPU Register	OCMC	128	148.78	276.12	269.51	269.6
M4 RD	OCMC	CPU Register	128	158.73	158.76	155.07	158.73
M4 COPY	OCMC	OCMC	128	141.7	177.34	174.21	177.78

- Transfer speed depends on Cache allocation policy.
- No-Write allocate policy gives better transfer performance for write operation than Write allocate policy. In case of write allocate policy, write miss causes cache line size written back to next level memory and new line read into the cache. For larger data transfer, due to frequent misses it will add extra latency and reduces transfer speed.
- Both Write-back and Write-through policies can use either of Write allocate or No-write allocate policy. But generally paired in this way: a Write-back policy uses Write allocate, hoping for subsequent writes to the same location, which is now cached; a Write-through policy uses No-write allocate policy, here subsequent writes have no advantage as they still need to be written to backing store.

Table 35. Impact of Word Size Used on IPU CPU Performance (Cache Policy: Write-Back, No-Write Allocate)

Initiator/Operation	Word Size	Source	Destination	Size (KB)	Bandwidth (MB/s)
M4 WR	32 bit	CPU Register	DDR	4096	267.95
M4 RD	32 bit	DDR	CPU Register	4096	93.17
M4 COPY	32 bit	DDR	DDR	4096	110.42
M4 WR	64 bit	CPU Register	DDR	8192	179.88
M4 RD	64 bit	DDR	CPU Register	8192	78.09
M4 COPY	64 bit	DDR	DDR	8192	100.2

Table 36 shows the performance difference between with and without loop unroll. In case of without loop unroll (when there is only one read, write, and copy operation in the loop) gives inferior performance compared to with loop unroll as it checks for loop condition after every word transfer in case of without loop unroll. In case of with loop unroll, it checks for loop condition only after a 128 words (write or read) or a 32 words (copy) transfer.

Table 36. IPU RD, WR, COPY Performance With Cache Enabled (Policy: Write-Back, No-Write Allocate), 32-bit Word Size

Initiator/Operation	Source	Destination	Size (KB)	Bandwidth (MB/s) (with loop unroll)	Bandwidth (MB/s) (without loop unroll)
M4 WR	CPU Register	DDR	4096	267.76	203.08
M4 RD	DDR	CPU Register	4096	93.19	69.2
M4 COPY	DDR	DDR	4096	110.6	104.24

7.3 Summary

Based on the observations made in Section 7.2, Table 37 lists the factors that affect the IPU Cortex-M4 CPU performance.

Table 37. Factors Affecting IPU Cortex-M4 CPU Performance

Factors	Impact	General Recommendation
Source/Destination Memory	The transfer speed depends on SRC/DST memory bandwidth.	Know the nature of the source and destination memory, specifically the frequency of operation and the bus width.
Transfer Size versus Cache Size	Larger data buffers written to than the cache size introduces a cache line write back along with the cache line reads for write allocate cache policy.	Expect drop in performance when the data buffer size written to is larger than the cache size.
Cache ability	Transfer speed depends on Unicache enable/disable.	Enable Unicache to get performance boost.
Cache Policy	Transfer speed depends on cache policy; write-back versus write-through and write-allocate versus no-write allocate.	Use write-through, no-write allocate policy to get more transfer speed.
Maximizing cache line reuse	Improves the CPU RD-WR performance.	The same memory locations within a cached line should be reused as often as possible. Either the same data can be reread or new data written to already cached locations so that subsequent reads will hit.
Eviction of a line	Avoiding eviction of a line as long as it is being reused improves the CPU RD-WR performance.	

8 USB IP

8.1 Overview

SuperSpeed USB DRD subsystem has four instances in the TDA2xx and TDA2ex device:

- USB1: SuperSpeed (SS) USB 3.0 Dual-Role-Device (DRD) subsystem with integrated SS (USB3.0) PHY and HS/FS (USB2.0) PHY
- USB2: High-Speed (HS) USB 2.0 Dual-Role-Device (DRD) subsystem with integrated HS/FS PHY
- USB3: High-Speed (HS) USB 2.0 Dual-Role-Device (DRD) subsystem with ULPI (SDR) interface to external HS/FS PHYs
- USB4 (not available in TDA2ex): High-Speed (HS) USB 2.0 Dual-Role-Device (DRD) subsystem with ULPI (SDR) interface to external HS/FS PHYs

SuperSpeed USB DRD subsystem has the following features:

- Dual-role-device (DRD) capability:
  - Supports USB Peripheral (or Device) mode at speeds SS (5 Gbps) (USB1 only), HS (480 Mbps), and FS (12 Mbps)
  - Supports USB Host mode at speeds SS (5 Gbps) (USB1 only), HS (480 Mbps), FS (12 Mbps), and LS (1.5 Mbps)
  - USB static peripheral operation
  - USB static host operation
  - Flexible stream allocation
  - Stream priority
  - External Buffer Control
- Each USB instance contains a single xHCI controller with the following features:
  - Internal DMA controller
  - Descriptor caching and data prefetching
  - Interrupt moderation and blocking
  - Power management USB3.0 states for U0, U1, U2, and U3
  - Dynamic FIFO memory allocation for all endpoints
  - Supports all modes of transfers (control, bulk, interrupt, and isochronous)
  - Supports high-bandwidth ISO mode
- Connects to an external charge pump for VBUS 5 V generation
- USB-HS PHY (USB2PHY1 for USB1 and USB2PHY2 for USB2): contains the USB functions, drivers, receivers, and pads for correct D+/D– signaling
- USB3PHY. The USB3PHY is embedded in the USB1 subsystem and contains:
  - USB3RX_PHY deserializer to receive data at SuperSpeed mode
  - USB3TX_PHY serializer to transmit data at SuperSpeed mode
  - Power sequencer that contains a power control state machine, generating the sequences to power up/down the USB3RX_PHY/USB3TX_PHY

8.2 USB IP Performance

8.2.1 Test Setup

Configuration used for the performance numbers are:

- Data Size used for calculation: Data transfer rates are 1Mbytes of data sent out 100 times in each of the observations.
- USB frequency:
  - SuperSpeed (SS):
    - Data transfer/differential Port: 5 GHz
    - Configuration Port : 133 MHz
  - High-Speed (HS):
    - Data transfer/differential Port: 480 Mbps
    - Configuration Port : 133 MHz
- L3 frequency : 133 MHz of L3 clock is used to configure the USB registers.

The USB can be used in 2 different modes: Peripheral and Host mode. For these two modes, the configuration of the test used is:

- Peripheral mode tests:
  - Host PC
    - Dell Precision T3600 - Xubuntu 13.10
    - 2 XHCI host cards, one from NEC and one from TI
    - On-board EHCI controller
  - TDA2xx and TDA2ex-evm
    - TDA2xx-evm running mass storage gadget with RAM backed storage for Bulk I/O tests. (Mass storage gadget are pen drives/TDA2xx and TDA2ex-evm as Storage device in case of Peripheral mode.)
    - TDA2xx-evm running g_zero gadget for raw ISO I/O tests. (g_zero gadget is a linux usb test peripheral driver that has source/sink capabilities for bulk and ISO transfers. Only the ISO endpoints for the ISO throughput test are used.)
    - CPU locked at 1 GHz
- Host mode tests:
  - TDA2xx and TDA2ex-evm used as host
    - CPU locked at 1 GHz
  - Peripheral gadget
    - OMAP5 µEVM running mass storage gadget with RAM backed storage for Bulk I/O tests. (Mass storage gadget are pen drives/TDA2xx and TDA2ex-evm as Storage device in case of Peripheral mode.)
    - OMAP5 µEVM running g_zero gadget for raw ISO I/O tests. (g_zero gadget is a linux usb test peripheral driver that has source/sink capabilities for bulk and ISO transfers. Only the ISO endpoints for the ISO throughput test are used.)
    - CPU locked at 1 GHz

      For host bulk tests, uses the OMAP5 µEVM as a mass storage device. For host ISO tests, uses the µEVM as a g_zero gadget.

Commands used for the transactions are:

- IN: dd if=/dev/sda of=/dev/null bs=1M count=100 iflag=direct
- OUT: dd if=/dev/zero of=/dev/sda bs=1M count=100 oflag=direct

8.2.2 Results and Observations

NOTE: Data transfer rates are 1 Mbytes of data sent out 100 times in each of the observations.

Table 38. USB IP Performance

Role of (TDA2xx and TDA2ex)	Device Type	Speed	Protocol	Conditions	TDA2xx and TDA2ex (MB/s)		Comment
					IN	OUT	
Host	Mass storage with RAM backed storage connected	HS	XHCI	TDA2xx and TDA2ex-evm high-speed host port (USB2)	35.6	33.6	
		SS	XHCI	TDA2xx and TDA2ex-evm SuperSpeed host port (USB1)	118.2	111	
Peripheral	As Mass storage with RAM backed storage	HS	EHCI	PC	32.8	27.6	
			XHCI	PC + NEC Card	33.3	31.3	
			XHCI	PC + TI Card	32.2	31.3	
		SS	XHCI	PC + NEC Card	104	71	
			XHCI	PC + TI Card	90.2	79.1	
	Raw ISO transfers	HS	EHCI	PC	24	24	24MB/s is maximum theoretical possible bandwidth for ISO transfers in HS mode

8.2.3 Summary

TDA2xx and TDA2ex performs 3x better in the SuperSpeed (SS) mode than in the High-Speed (HS) mode, when operating as a host or as a peripheral.

9 PCIe IP

9.1 Overview

The peripheral component interconnect express (PCIe) module is a multi-lane I/O interconnect that provides low pin count, high reliability, and high-speed data transfer at rates of up to 5.0 Gbps per lane per direction, for serial links on backplanes and printed wiring boards. The device has two PCIe subsystems (PCIe_SS1 and PCIe_SS2) each of which is built on a Synopsys DesignWare® core (DWC) PCIe controller. Each of the two PCIe controllers is capable to operate either in Root Complex (RC) or in End Point (EP) PCIe mode. PCIe_SS1 can be configured to operate in either 2-Lane (dual lane) or 1-Lane (single lane) mode. PCIe_SS2 can only operate in 1-Lane mode.

9.2 PCIe IP Performance

9.2.1 Test Setup

The test setup consists of two TDA2xx and TDA2ex EVM (for gen1, x1 mode) PCIe ports connected by way of a PCIe cable. 64kB of data is transferred from DDR of one TDA2xx and TDA2ex board to DDR of the other TDA2xx and TDA2ex board (EVM-to-EVM for x1) by way of a PCIe slot/cable. The TDA2xx and TDA2ex GP Timer of the initiator board (board initiating read/write transfer on the PCIe through the EDMA) is used to measure the time taken for the transfer for read/write transfer and arrive at the bandwidth measurement. The EDMA is used for the data transfer.

Configuration for the different modules is:

- PCIe configuration:
  - Packet sizes: 128, 64, 32, 16 bytes
  - RC and EP configurations
- EDMA configuration:
  - AB cnt: a_cnt = 16k, b_cnt = 4, tptc0
  - Burst sizes: 128, 64, 32, 16 bytes
- Timer Configuration:
  - GP Timer 2, Clock Frequency = 22.5 MHz
  - L3 clock frequency of 266 MHz

9.2.2 Results and Observations

Table 39. GEN1, X1 PCIe Performance

Operation (WR/RD)	Data Direction	Burst Size	Size (KB)	Bandwidth (Mbits/s)
WR	RC writing Into EP	128 bytes	65534	1510
WR	RC writing Into EP	64 bytes	65534	1510
WR	RC writing Into EP	32 bytes	65534	1223.06
WR	RC writing Into EP	16 bytes	65534	884.02
RD	RC reading from EP	128 bytes	65534	1430.35
RD	RC reading from EP	64 bytes	65534	1409.83
RD	RC reading from EP	32 bytes	65534	1141.04
RD	RC reading from EP	16 bytes	65534	821.05
WR	EP writing Into RC	128 bytes	65534	1512.32
WR	EP writing Into RC	64 bytes	65534	1512.32
WR	EP writing Into RC	32 bytes	65534	1223.03
WR	EP writing Into RC	16 bytes	65534	883.8
RD	EP reading from RC	128 bytes	65534	1430.39
RD	EP reading from RC	64 bytes	65534	1409.88
RD	EP reading from RC	32 bytes	65534	1140.41
RD	EP reading from RC	16 bytes	65534	819.88

Table 40. GEN1, X2 PCIe Performance

Operation (WR/RD)	Data Direction	Burst Size	Size (KB)	Bandwidth (Mbits/s)
WR	RC writing Into EP	128 bytes	65534	3006.14
WR	RC writing Into EP	64 bytes	65534	3006.14
WR	RC writing Into EP	32 bytes	65534	2427.18
WR	RC writing Into EP	16 bytes	65534	1719.3
RD	RC reading from EP	128 bytes	65534	2839.01
RD	RC reading from EP	64 bytes	65534	2808.6
RD	RC reading from EP	32 bytes	65534	2136.97
RD	RC reading from EP	16 bytes	65534	1250.25
WR	EP writing Into RC	128 bytes	65534	3010.75
WR	EP writing Into RC	64 bytes	65534	3006.14
WR	EP writing Into RC	32 bytes	65534	2431.68
WR	EP writing Into RC	16 bytes	65534	1763.24
RD	EP reading from RC	128 bytes	65534	2841.15
RD	EP reading from RC	64 bytes	65534	2814.71
RD	EP reading from RC	32 bytes	65534	2133.56
RD	EP reading from RC	16 bytes	65534	1255.47

Table 41. GEN2, X1 PCIe Performance

Operation (WR/RD)	Data Direction	Burst Size	Size (KB)	Bandwidth (Mbits/s)
WR	RC writing Into EP	128 bytes	65534	3013.15
WR	RC writing Into EP	64 bytes	65534	3013.15
WR	RC writing Into EP	32 bytes	65534	2440.82
WR	RC writing Into EP	16 bytes	65534	1765.68

Table 41. GEN2, X1 PCIe Performance (continued)

Operation (WR/RD)	Data Direction	Burst Size	Size (KB)	Bandwidth (Mbits/s)
RD	RC reading from EP	128 bytes	65534	2849.3
RD	RC reading from EP	64 bytes	65534	2826.77
RD	RC reading from EP	32 bytes	65534	2278.12
RD	RC reading from EP	16 bytes	65534	1639.72
WR	EP writing Into RC	128 bytes	65534	3013.15
WR	EP writing Into RC	64 bytes	65534	3015.46
WR	EP writing Into RC	32 bytes	65534	2439.31
WR	EP writing Into RC	16 bytes	65534	1764.88
RD	EP reading from RC	128 bytes	65534	2849.3
RD	EP reading from RC	64 bytes	65534	2828.81
RD	EP reading from RC	32 bytes	65534	2280.77
RD	EP reading from RC	16 bytes	65534	1645.21

Table 42. GEN2, X2 PCIe Performance

Operation (WR/RD)	Data Direction	Burst Size	Size (KB)	Bandwidth (Mbits/s)
WR	RC writing Into EP	128 bytes	65534	5966.86
WR	RC writing Into EP	64 bytes	65534	5957.82
WR	RC writing Into EP	32 bytes	65534	4824.74
WR	RC writing Into EP	16 bytes	65534	3507.73
RD	RC reading from EP	128 bytes	65534	5476.38
RD	RC reading from EP	64 bytes	65534	5530.3
RD	RC reading from EP	32 bytes	65534	3357.85
RD	RC reading from EP	16 bytes	65534	1904.14
WR	EP writing Into RC	128 bytes	65534	5957.82
WR	EP writing Into RC	64 bytes	65534	5957.82
WR	EP writing Into RC	32 bytes	65534	4830.66
WR	EP writing Into RC	16 bytes	65534	3510.86
RD	EP reading from RC	128 bytes	65534	5514.78
RD	EP reading from RC	64 bytes	65534	5522.53
RD	EP reading from RC	32 bytes	65534	3389.69
RD	EP reading from RC	16 bytes	65534	1943.67

10 IVA-HD IP

10.1 Overview

HDVICP2/IVAHD is TIs second generation Video and Imaging co-processor designed to accelerate the HD Video encoding and decoding. It is a successor to the HDVICP 1.0. Both generations support 4:2:0 Chroma formats only. HDVICP2 is sometimes referred to as IVAHD 1.0. The DM46x devices are some of the SoCs that have HDVICP 1.0.

DM816x, DM814x, DM813x, OMAP4, and OMAP5 are some of the SoCs that have HDVICP 2.0. These different SoCs have varying number of instances of HDVICP2 and operating frequency. For more details, see the device-specific data sheet.

The block diagram of IVAHD is shown in Figure 30.

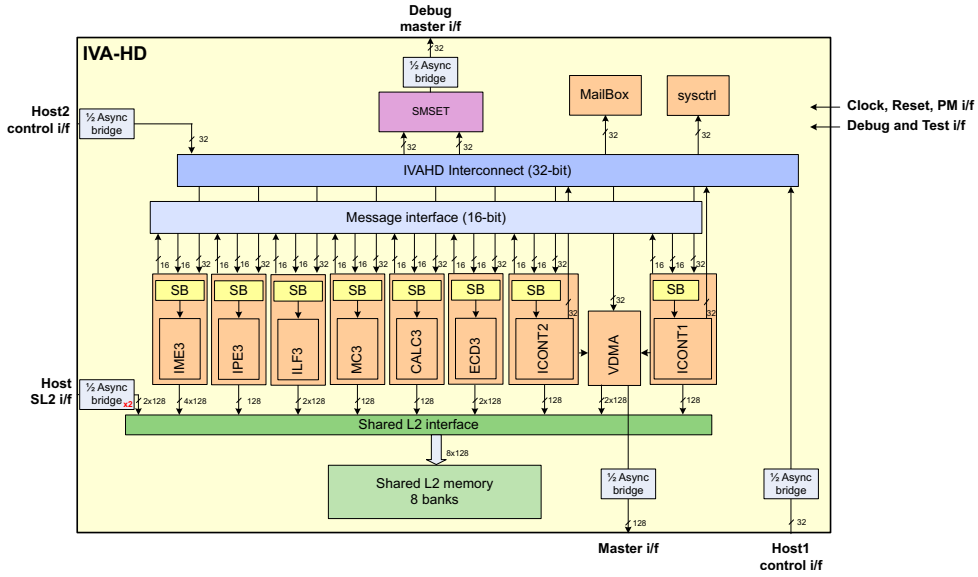


Figure 30. IVAHD Block Diagram

The typical process call structure for the IVAHD codecs that encodes or decodes a single frame of a single channel of encode or decode is shown in Figure 31. The major performance probe points are:

- Process Call Start – Begin the CODEC operation.
- HDVICP2 Resource Acquire – Acquire IVAHD resource to start IVAHD operations.
- HDVICP2 Wait – M4 pre-processing completes. M4 can perform a thread switch to perform some other operation while IVAHD performs encode or decode operations.
- HDVICP2 Done – M4 receives a completion interrupt from IVAHD and switches back to the codec thread.
- HDVICP2 Release – M4 post processing thread releases the IVAHD resource.
- Process Call End – Completion of the frame encodes or decode operation.

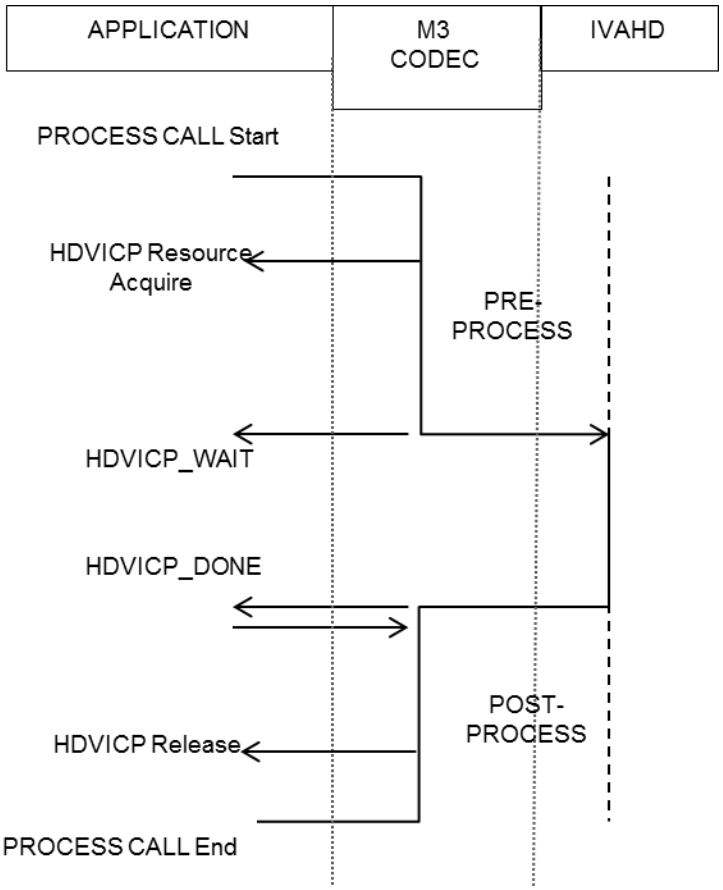


Figure 31. IVAHD Software Performance Probe Points

10.2 H.264 Decoder

10.2.1 Description

H.264 is a video compression standard from the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group. This H.264 Decoder is validated on IVAHD (also known as, HDVICP2) and Media Controller (M4) based platform with code generation tools version 4.5.1.

10.2.2 Test Setup

Codec Version Used: REL.500.V.H264AVC.D.HP.IVAHDSRC.01.00.00.12

Frequencies of operation for each of the modules involved is listed in Table 43.

Table 43. IVAHD Measurement Frequency Table

IP	TDA2xx and TDA2ex Si Frequency (MHz)
L3	266
EMIF	266
DDR	532
IVAHD	388.33
M4	212.8

The configuration used to measure the performance is for H.264 High Profile Universal Decoder: Resolutions up to 1920 × 1080.

10.2.3 Test Results

Table 44. H.264 Decoder Performance Data

Stream Name	Stream Properties	IVAHD (Mega Cycles per Second)	
		Average	Peak
cabac_mot_fld0_full.264	D1 (720 × 480) Resolution, Interlaced, I, P, and B Fields	31.2122	31.76064
bigships_p1280x720_60fps_420pl_600fr.yuv	720p Resolution, Progressive, I, P, and B Frames	70.45614	73.5405
Shields_i1920x1080_30fps_14mbps.264	1080p Resolution, Interlaced, I, P, and B Fields	166.9958	230.9215

For information on the slice/sub frame level performance of H.264 decoder on the IVAHD, see the Performance Summary of HDVICP2 Video Code.

10.3 MJPEG Decoder

10.3.1 Description

JPEG is an international standard for color image compression. This standard is defined in the ISO 10918-1 JPEG Draft International Standard CCITT Recommendation T.81. It is a widely used Image compression algorithm that uses Inverse Quantization, Inverse Discrete Cosine Transform (IDCT) coding of the residual data and Huffman entropy coding.

10.3.2 Test Setup

The test setup used to measure frame level switch overheads leading to three categories of measurements:

- Same Codec: When frames of the same codec (MJPEG) and same stream are decoded back to back. The M4 MHz consumed will be the least in this case as the codec does not save and restore channel persistent data and codec IVAHD ICONT code.
- Same Codec Type: When frames of the same codec (MJPEG) and different streams are decoded back to back. The overheads for channel specific persistent data save and restore come into picture for this case.
- Different Codec: When frames from different codecs are run back to back. Here, the overheads include both channel specific persistent data save and restore and Codec ICONT code load.

The overhead in all of the cases can be observed in the M4 MHz for the Codec as shown in Section 10.3.3.

10.3.3 Test Results

Table 45. MJPEG Decoder Performance Data for Same Codec

Test Cases	Output Format	M3 Cycles / MHz				Bitrate (Mb/s)	IVAHD (MHz)
		Process Entry to HDVICP Wait	HDVICPwait to HDVICPdone	HDVICPdone to Process End	Total MHz (M3 Overhead)		
crowdRun_1280x720_420.mjpeg	420SP	7196	1094578	2798	0.30	96	64
city_1280x720_420p.mjpeg	420SP	7173	1048932	2789	0.30	60	61
crowdRun_1280x720_422.mjpeg	420SP	7443	1434446	2890	0.31	109.2	83
city_1280x720_422.mjpeg	420SP	7401	1288815	2765	0.30	56.65	75

Table 46. MJPEG Decoder Performance Data for Same Codec Type

Test Cases	Output Format	M3 Cycles / MHz				Bitrate (Mb/s)	IVAHD (MHz)
		Process Entry to HDVICP Wait	HDVICPwait to HDVICPdone	HDVICPdone to Process End	Total MHz (M3 Overhead)		
crowdRun_1280x720_420.mjpeg	420SP	16254	1094656	12901	0.87	96	63
city_1280x720_420p.mjpeg	420SP	16674	1048568	12821	0.88	60	57
crowdRun_1280x720_422.mjpeg	420SP	16932	1434460	12798	0.89	109.2	81
city_1280x720_422.mjpeg	420SP	17108	1288789	13010	0.90	56.65	75

Table 47. MJPEG Decoder Performance Data for Different Codec

Test Cases	Output Format	M3 Cycles / MHz				Bitrate (Mb/s)	IVAHD (MHz)
		Process Entry to HDVICP Wait	HDVICPwait to HDVICPdone	HDVICPdone to Process End	Total MHz (M3 Overhead)		
crowdRun_1280x720_420.mjpeg	420SP	34126	1094590	14107	1.45	96	63
city_1280x720_420p.mjpeg	420SP	33921	1048978	14239	1.45	60	57
crowdRun_1280x720_422.mjpeg	420SP	34201	1434501	13997	1.45	109.2	81
city_1280x720_422.mjpeg	420SP	34153	1288954	14076	1.45	56.65	75

11 MMC IP

11.1 MMC Read and Write Performance

11.1.1 Test Description

MMC1: Full SD Extreme Pro card from SanDisk [SanDisk Extreme Pro SDHC I U1 C10 8GB (SDR104 capable)]

MMC2: eMMC capable of HS200 [Device Micron MTFC8GLWDM-3M AIT Z (HS-200 capable eMMC device)]

MMC1: always 4-bit mode by design

MMC2: could be 4-bit or 8-bit

SDR: Single Data Rate

DDR: Double Data Rate

The MMC, in general, has ideal READ throughput only; for WRITE, the throughput would be much less than READ as write is limited by flash write performance of card.

The test used would be to WRITE and READ 100 MB of data from MMC1 to Full SD card (MMC2 to eMMC). MMC ADMA has been used to perform the transfers in the data shown in Table 48. It has been found that using the MMC SDMA does not give any advantage in terms of bare metal MMC throughput but does give an advantage in the programming of DMA due to the number of descriptors that can be programmed and the chaining possible between them.

11.1.2 Test Results

Table 48. MMC Read and Write Throughput

Operation	Frequency (MHz)	Mode	Throughput (MB/s) Ideal RD	Throughput (MB/s) Actual RD	Throughput (MB/s) Actual WR
MMC2	48	DDR mode 8-bit	96	81.2	36.1
MMC2	48	DDR mode 4-bit	48	43.1	30.1
MMC2	48	SDR mode 8-bit	48	43.1	30.2
MMC2	48	SDR mode 4-bit	24	22.2	19.7
MMC1	96	SDR mode 4-bit	48	44.4	38
MMC2	192	SDR mode 4-bit	96	87	37 (1)
MMC2	192	SDR mode 8-bit	192	162	40 (1)
MMC2	96	SDR mode 4-bit	48	44	31 (1)
MMC2	96	SDR mode 8-bit	96	85	37 (1)
MMC1	192	SDR mode 4-bit	96	89	77 (1)

(1) The speeds of write operation are applicable only to TDA2xx SR2.0. For more information, see the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Silicon Errata* (SPRZ397).

11.2 Summary

eMMC sequential WRITE throughput from the data sheet is 15 MB/s in 192-MHz 8-bit mode, whereas, you can see approximately 40 MB/s, which means that more than nominal throughput (mentioned in the data sheet) was achieved.

eMMC sequential READ throughput from the data sheet is 160 MB/s in 192-MHz 8-bit mode, whereas, you can see approximately162 MB/s, which means that more than nominal throughput (mentioned in the data sheet) was achieved.

From the results, the READ and WRITE throughputs are matching to the expectations.

12 SATA IP

The SATA controller complies to the Serial ATA Standard Specification (revision 2.6) and Serial ATA Advanced Host Controller Interface Specification (AHCI) revision 1.1. The AHCI-based SATA host controller supports both Gen1/2 speeds: 1.5 Gbps (SATA-1) and 3 Gbps (SATA-2).

12.1 SATA Read and Write Performance

12.1.1 Test Setup

SATA performance measurements were done in both RAW mode and using the file system in Linux.

12.1.2 Observations

12.1.2.1 RAW Performance

With SATA in GEN2 mode, the standalone RAM performance for Reads and Writes to a SanDisk iSSD device are:

- Read performance data: 273 MB/s
- Write performance data: 105 MB/s

12.1.2.2 SDK Performance

Seagate ST3500514NB SATA-II drive is used to measure the performance data in Table 49.

Table 49. SATA - File System Performance

File System	Total Bytes Transferred (MBytes)	TDA2xx and TDA2ex	
		Read (MB/s)	Write (MB/s)
Ext2	1000	133	100
Ext4	1000	129	117
VFat	1000	134	90

12.2 Summary

Lower write performance using SATA iSSD is mainly attributed to the slower flash write speed of the iSSD device. There is no restriction from the TDA2xx and TDA2ex SoC to attain full write speed as specified in the SATA specification.

When using HDD and the file system, it is observed that there is marginal lower SATA write performance than read performance. This is as expected as the HDD drive write has higher seek time than reads and other file system write overheads.

13 GMAC IP

The three-port gigabit Ethernet switch subsystem (GMAC_SW) provides Ethernet packet communication and can be configured as an Ethernet switch. It provides the media independent interface (MII), reduced gigabit media independent interface (RGMII), reduced media independent interface (RMII) and the management data input output (MDIO) for physical layer device (PHY) management.

GMAC sub-system in TDA2xx and TDA2ex supports the following features:

- Two Ethernet ports (port 1 and port 2) plus internal CPPI on port 0
- Synchronous 10/100/1000 Mbit operation
- Wire rate switching (802.1d)
- Internal DMA controller
- Support for Audio/Video Bridging (P802.1Qav/D6.0)
- Energy Efficient Ethernet (EEE) support (802.3az)
- Reset isolation (switch function remains active even in case of all device resets except for POR pin reset and ICEPICK cold reset)
- Static Packet Filter(SPF)

For more details, see the *GMAC* section of the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

13.1 GMAC Receive/Transmit Performance

13.1.1 Test Setup

- All benchmarking tests run from Cortex M4 CPU (IPU) (Frequency: 212 MHz and unicache enabled).
- DDR3 Clock : 532 MHz
- L3 Frequency : 266 MHz
- Throughput is in Mbps and data size in bytes
- Throughput measured without any application stack running. Throughput numbers are collected with AVV bare metal tests without any application stack or actual TCP/IP network connected.
- GMAC subsystem is configured in RGMII 1000 Mbps mode
- Data Transfer path for performance measurements is as shown below:
  - Transmit and Receive throughput set up: This test set up involves GMAC port1 transmitting data that is looped back on GMAC port2. Channel 0 of 8-channel CPDMA used for receive and channel 7 used for transmit. Single EVM was used for test.

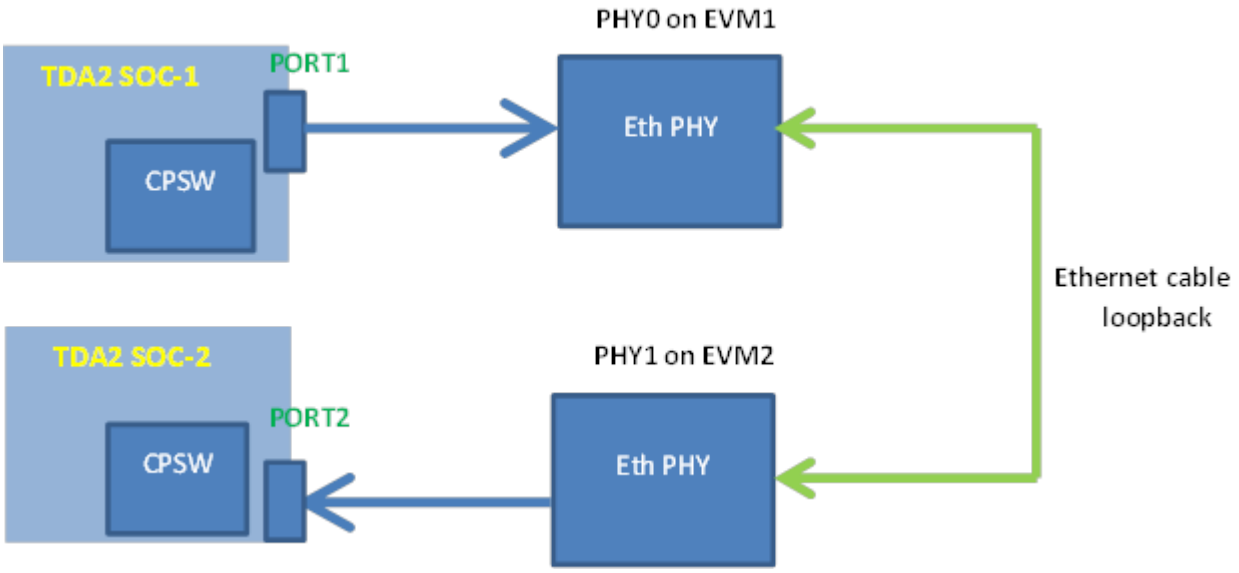


Figure 32. Tx/Rx Throughput Measurement Set Up

– Transmit only and Receive only performance set up: two TDA2xx EVMs were connected back-to-back to measure Tx only and Rx only performances. On one EVM transmit throughput was measured, while on other receive throughput was measured.

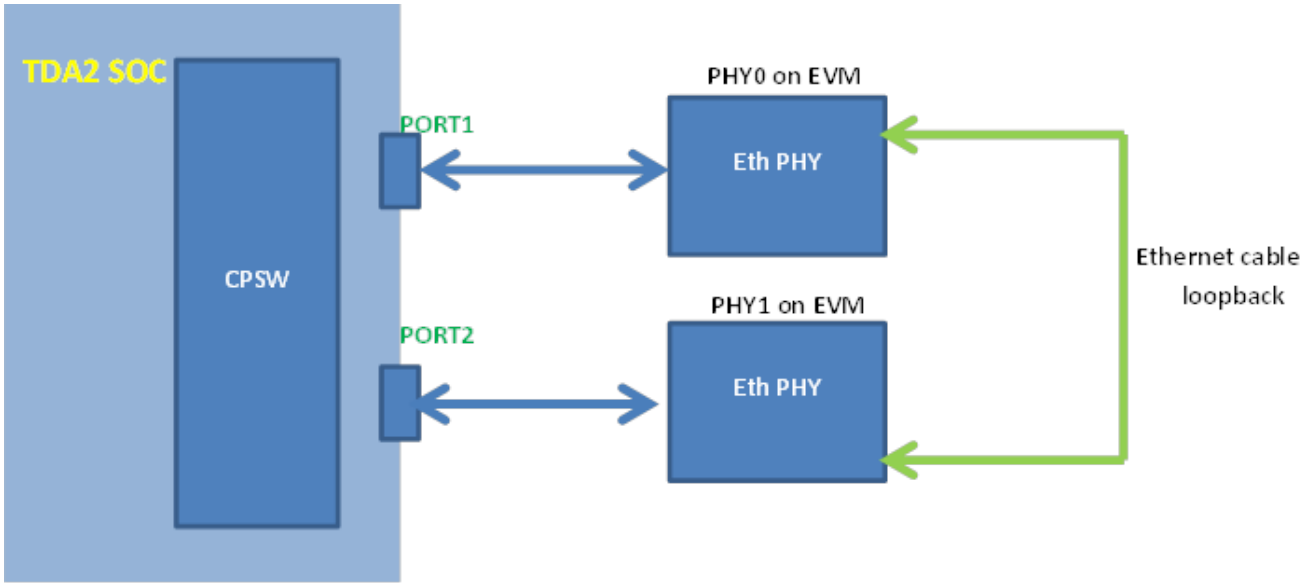


Figure 33. Tx only and Rx Only Throughput Measurement Set Up

13.1.2 Test Description

13.1.2.1 CPPI Buffer Descriptors

The buffer descriptor is a central part of the GMAC_SW Ethernet Subsystem and is how the application software describes Ethernet packets to be sent and empty buffers to be filled with incoming packet data. The TDA2xx and TDA2ex GMAC subsystem contains 8KB internal CPPI RAM for buffer descriptor storage, which can be shared for receive and transmit transfer however the application software intends. GMAC also supports using SRAM and DDR memory for buffer descriptors. In throughput measurement tests, all three memories have been used for storing descriptors.

13.1.3 Test Results

The tests results for GMAC in RGMII mode are as shown below.

13.1.3.1 Receive/Transmit Mode (see Table 50)

Table 50. GMAC RGMII Rx/Tx Transfer

Sr. No	RX/TX Descriptor Location	Source	Destination	Data Size (packet no x packet size (in bytes))	Throughput (Mbps)	Utilization/Efficiency (% to 2Gbps)
1	DDR	DDR	DDR	9000 x 1512	1833.916	91.6958
2	OCMC	DDR	DDR	9000 x 1512	1853.943	92.69715
3	DDR	DDR	DDR	125 x1512	1791.954	89.5977
4	OCMC	DDR	DDR	125 x1512	1801.491	90.07455
7	CPPI BUFFER	DDR	DDR	125 x1512	1817.703	90.88515

13.1.3.2 Receive Only Mode (see Table 51)

Table 51. GMAC RGMII Rx Only

Sr. No	RX/TX Descriptor Location	Source	Data Size (packet no x packet size (in bytes))	Throughput (Mbps)	Utilization/Efficiency (% to 2Gbps)
1	DDR	DDR	9000 x 1512	936.9456	93.69456
2	OCMC	DDR	9000 x 1512	937.0865	93.70865
3	DDR	DDR	1000 x 1512	938.2352	93.82352
4	OCMC	DDR	1000 x 1512	938.2467	93.82467
5	DDR	DDR	125 x1512	938.3187	93.83187
		OCMC	125 x1512	938.4111	93.84111
6	DDR	DDR	125 x1512	938.3187	93.83187
		OCMC	125 x1512	938.4111	93.84111
7	CPPI BUFFER	DDR	125 x1512	938.4111	93.84111
		OCMC	125 x1512	938.5034	93.85034

13.1.3.3 Transmit Only Mode (see Table 52)

Table 52. GMAC RGMII Tx Only

Sr. No	RX/TX Descriptor Location	Source	Data Size (packet no x packet size (in bytes))	Throughput (Mbps)	Utilization/Efficiency (% to 2Gbps)
1	DDR	DDR	9000 x 1512	928.6769	93.66769
2	OCMC	DDR	9000 x 1512	936.914	93.6914
3	DDR	DDR	1000 x 1512	936.4754	93.64754
4	OCMC	DDR	1000 x 1512	936.4754	93.67149
5	DDR	DDR	125 x1512	934.7566	93.47566
		OCMC	125 x1512	935.0294	93.50294
6	OCMC	DDR	125 x1512	934.9384	93.49384
		OCMC	125 x1512	935.3023	93.53023
7	CPPI BUFFER	DDR	125 x1512	935.2113	93.52113
		OCMC	125 x1512	935.3023	93.53023

13.2 Summary

GMAC CPPI internal memory should be used to get the optimized performance as latency to access CPPI buffer memory is low compared to DDR/SRAM for CPDMA. This is applicable only if the CPU is not accessing GMAC descriptors very frequently like in case of high bandwidth stream small size packets. In this case as latency to access CPPI RAM from CPU is more compared to the DDR keeping descriptors in DDR would give better performance. So care should be taken for selecting CPPI descriptor location depending on the use-case.

14 GPMC IP

The General-Purpose Memory Controller is the TDA2xx and TDA2ex unified memory controller dedicated to interface external memory devices like asynchronous SRAM-like memories, asynchronous, page mode and synchronous burst NOR Flash, NAND Flash and pseudo-SRAM devices. The GPMC data access engine provides a flexible programming model to interface all known standard memories. The access engine can support the following interfacing protocols:

- Asynchronous read/write access
- Asynchronous read page access (4-8-16-32 Word16, 4-8-16 Word32)
- Synchronous read/write access
- Synchronous read/write burst access without wrap capability (4-8-16-32 Word16, 4-8-16 Word32)
- Synchronous read/write burst access with wrap capability (4-8-16-32 Word16, 4-8-16 Word32)
- Address and Data multiplexed access
- Little-Endian and Big-Endian access

For more details, see the *GPMC* section in the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

This enables to interface a wide range of external devices like:

- External asynchronous or synchronous 8-bit width memory or device (non-burst device)
- External asynchronous or synchronous 16-bit width memory or device
- External asynchronous or synchronous 32-bit width memory or device
- External 16-bit non-multiplexed NOR Flash device
- External 16-bit and 32-bit address and data multiplexed NOR Flash device
- External 8-bit and 16-bit NAND flash device
- External 16-bit and 32-bit pSRAM device

For more details, see the *GPMC* section in the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

14.1 GPMC Read and Write Performance

14.1.1 Test Setup

14.1.1.1 NAND Flash

Configuration used for the performance numbers are:

- NAND Flash part number is MT29F2G08ABCWP
- 16-bit NAND device
- 1 Page size = 1024 words + 32 spare data words
- 1 block size = 64 pages
- Timing information about the NAND Flash:
  - The page read initiates the transfer of 2048 bytes of data from the flash array to the data register that requires a time of 25 µs
  - The random read command helps to access the data from any column address in a page with an initial access time of min 80 ns
  - 1 page write costs a time of about 220-300 µs
  - Sequential read time per byte is 50 ns

14.1.1.2 NOR Flash

Configuration used for the performance numbers are:

- NOR Flash part number is s29gl512s
- GPMC functional clock = L3_MAIN_CLK = 266 MHz
- 512-Mbyte density
- x16 data bus
- Asynchronous 32-byte Page read

14.1.2 Test Description

14.1.2.1 Asynchronous NAND Flash Read/Write Using CPU Prefetch Mode

This performance measurement is done using prefetch mode for read and post engine mode for write. This mode is more efficient than the normal polling method since in both these modes, a programmable FIFO threshold (maximum 64 bytes) is set. The FIFO input on the host OCP side is accessible at any address in the associated chip-select memory region. Using prefetch mode and post engine mode and polling of the FIFO status, the following results are obtained. For more details, see the *GPMC* section of the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

- DDR3 at 532 MHz
- L3 MAIN Frequency = GPMC functional clock frequency = 266 MHz
- GP Timer was used to time the number of clock cycles it took for data transfer. Timer Clock for profiling selected as 20 MHz.
- All throughput data collected “standalone”. No other ongoing traffic.
- 8-bit BCH correction algorithm is used.

GPMC timing parameters used:

- NAND_CSWROFFTIME: 13
- NAND_CSRDOFFTIME: 13
- NAND_RDCYCLETIME: 13
- NAND_WRCYCLETIME: 13
- NAND_RDACCESSTIME: 12

- NAND_WRACCESSTIME: 12
- NAND_OEOFFTIME: 11

14.1.2.2 Asynchronous NOR Flash Single Read

Address access time (tACC) is equal to the delay from stable addresses to valid output data The chip enable access time (tCE) is the delay from stable CE# to valid data at the outputs. In order for the read data to be driven on to the data outputs, the OE# signal must be low for at least the output enable time (tOE) before valid data is available.

Figure 34 shows the read operation timing diagram at the Flash.

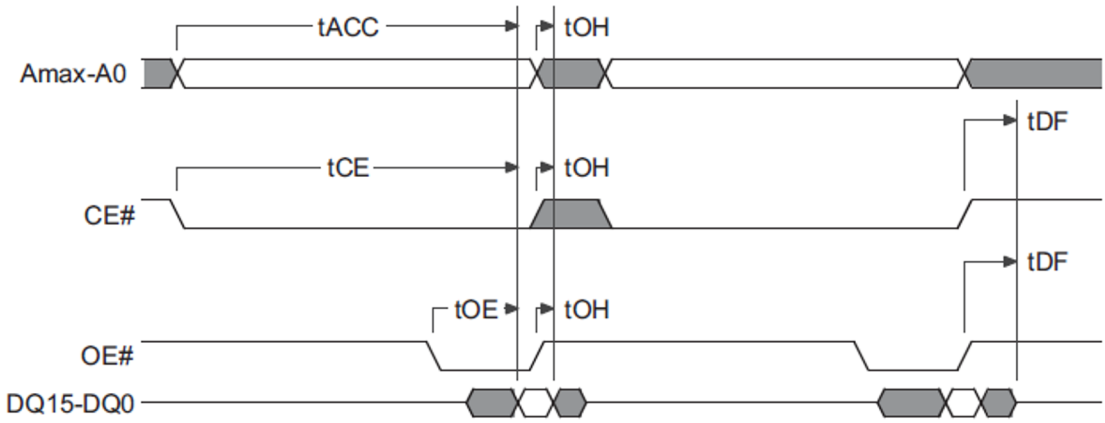


Figure 34. Back-to-Back Read (tACC) Operation Timing Diagram

Figure 35 shows the read operation timing diagram with GPMC signal parameters.

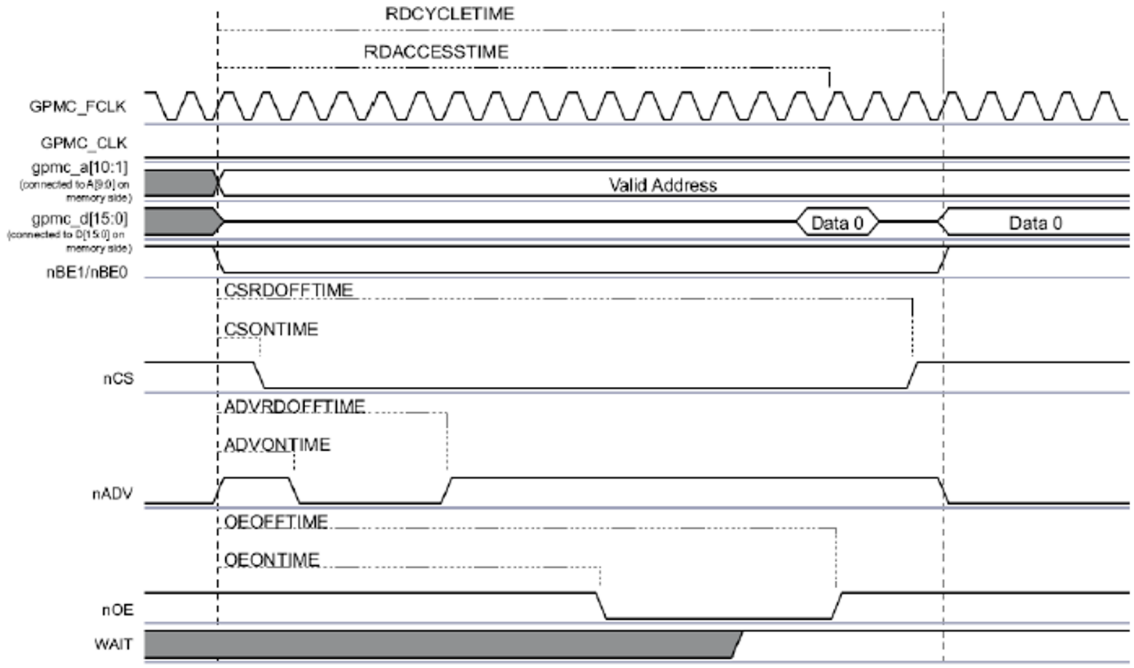


Figure 35. Asynchronous Single Read Timing Parameters

For the successful read operation to occur, GPMC timing parameters have to be set satisfying the Flash level timing values. Table 53 shows the optimum configuration for GPMC timing values for successful read operation. 1 GPMC clock =~ 3.7 ns. Here “Timeparagranularity” is set as 0x1, which will multiply the configured timing values by 2.

Table 53. Optimum Configuration for GPMC for Reads of NOR Flash Single Read

Signal	Parameter	Description	Value Programmed
Read op	RDACCESSTIME	Address latch + Initial access time = 0ns + 110ns = 30 GPMC clock cycles	0x0F
	RDCYCLETIME	RDACCESSTIME + Data holding + tDF = 30 + 4 + 4 = 38 GPMC clock cycles	0x13
nCS	CSONTIME	Assert after address latch	0x00
	CSRDOFFTIME	RDACCESSTIME + Data holding = 30 + 4 = 34 GPMC clock cycles	0x11
nADV	ADVONTIME	Immediate assert with read cycle	0x00
	ADVOFFTIME	Provide ADV assertion duration of 2 cycles	0x01
nOE	OEONTIME	Assert after address latch	0x00
	OEOFFTIME	RDACCESSTIME + Dataholding	0x11

14.1.2.3 Asynchronous NOR Flash Page Read

In Page mode read, each read can access a maximum of 32-byte pages in parallel. For page read to be successful, asynchronous page read access time (tPACC) should be satisfied.

Figure 36 shows the page read operation timing diagram at the Flash.

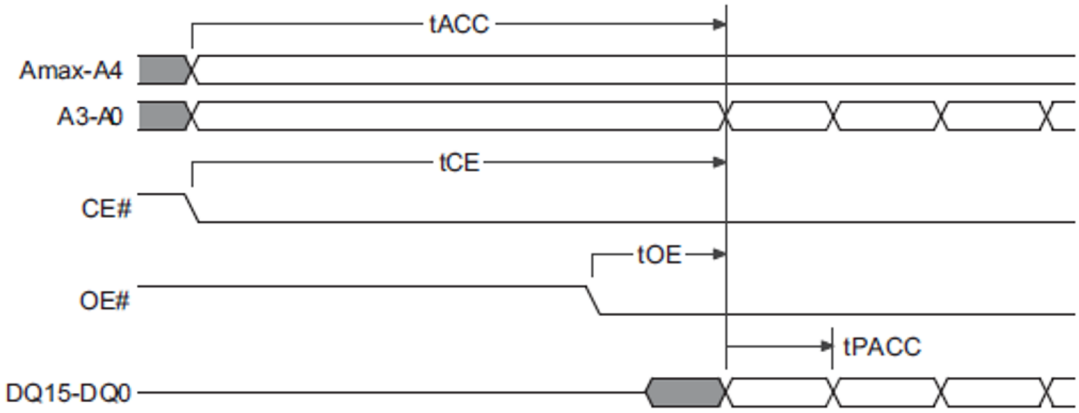


Figure 36. NOR Flash Page Read Timing Diagram

Figure 37 shows the page read operation timing diagram with GPMC signal parameters.

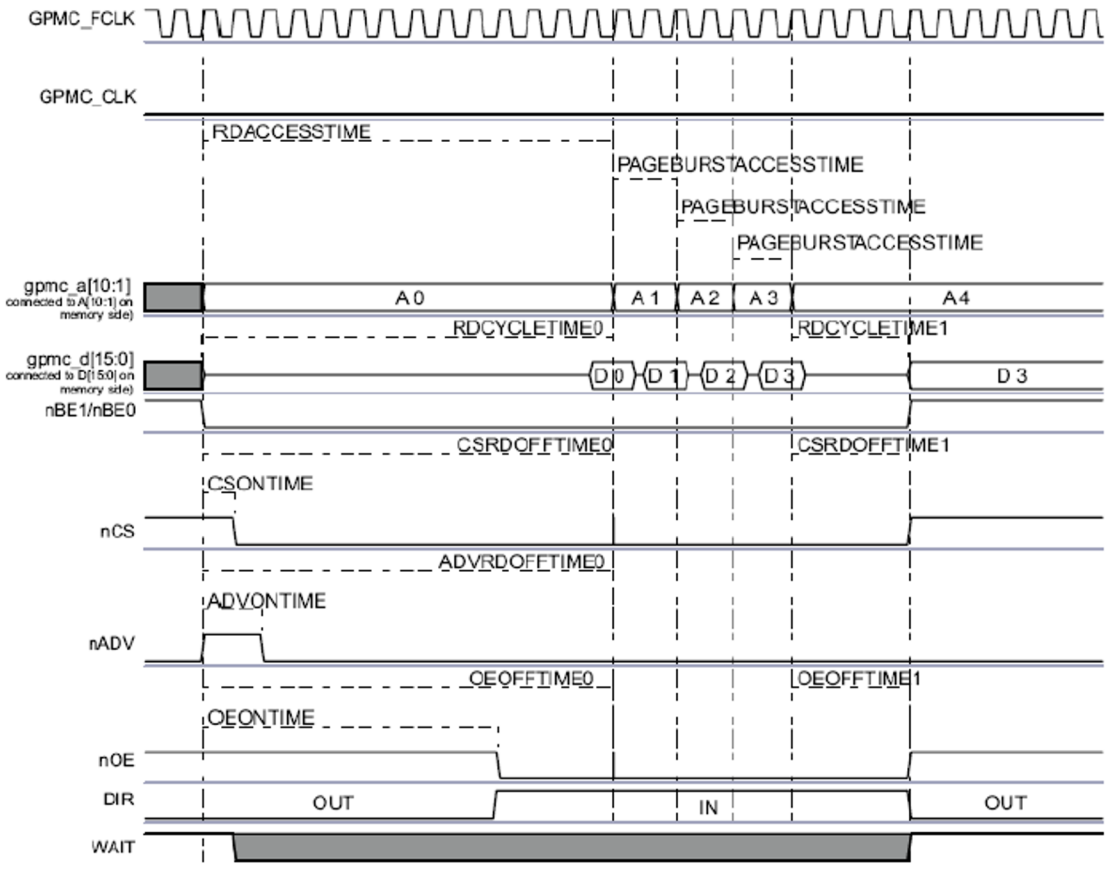


Figure 37. Asynchronous Page Read Timing Parameters

Table 54 shows the optimum configuration for GPMC timing values for successful page read operation. 1 GPMC clock = approximately 3.7 ns. Here “Timeparagranularity” is set as 0x1, which will multiply the configured timing values by 2.

Table 54. Optimum Configuration for GPMC Timing Values for Successful Page Read Operation

Signal	Parameter	Description	Value Programmed
Read op	RDACCESSTIME	Address latch + Initial access time = 0ns + 110ns = 30 GPMC clock cycles	0x0F
	RDCYCLETIME = RDCYCLETIME0 + RDCYCLETIME1	RDACCESSTIME + Data holding + tDF = 30 + 4+ 4 = 38 GPMC clock cycles	0x13
nCS	CSONTIME	Assert after address latch	0x00
	CSRDOFFTIME = CSRDOFFTIME0 + CSRDOFFTIME1	RDACCESSTIME + Data holding = 30 + 4 = 34 GPMC clock cycles	0x11
nADV	ADVONTIME	Immediate assert with read cycle	0x00
	ADVOFFTIME	Provide ADV assertion duration of 2 cycles	0x01
nOE	OEONTIME	Assert after address latch	0x00
	OEOFFTIME = OEOFFTIME0 + OEOFFTIME1	RDACCESSTIME + Data holding	0x11
Page Burst Access	PAGEBURSTACCESSTIME	tPACC = 25ns (min) ~ 7 GPMC clock cycles	0x04
Pagelength	ATTACHEDDEVICEPAGELENGTH	16 words burst size	0x10

14.1.2.4 Asynchronous NOR Flash Single Write

Figure 38 shows the write operation timing diagram at the Flash.

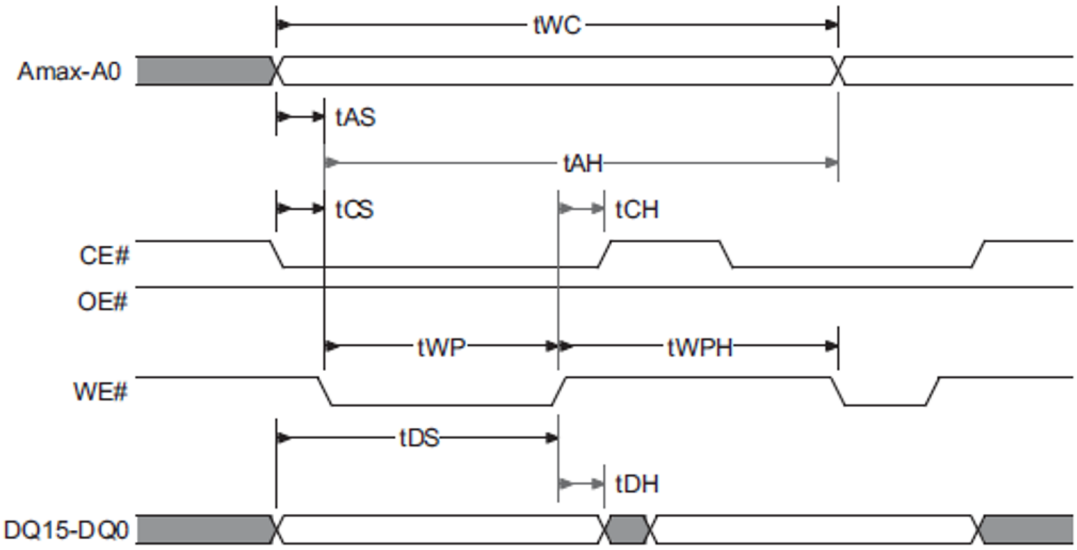


Figure 38. Back-to-Back Write Operation Timing Diagram

Figure 39 shows the asynchronous write operation timing diagram with GPMC signal parameters.

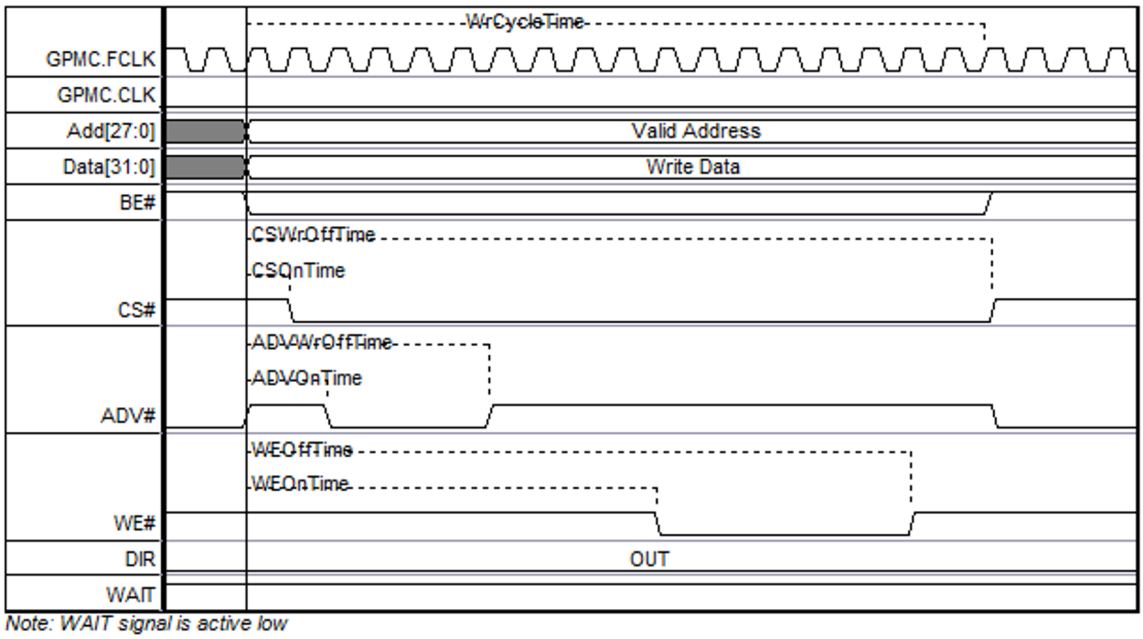


Figure 39. Asynchronous Single Write to a Non-Multiplexed Add/Data Device

Table 55 shows the optimum configuration for GPMC timing values for successful write operation. 1 GPMC clock = approximately 3.7 ns. Here “Timeparagranularity” is set as 0x1, which will multiply the configured timing values by 2.

Table 55. Optimum Configuration for GPMC Timing Values for Successful Page Write Operation

Signal	Parameter	Description	Value Programmed
Write op	WRACCESSTIME	Address latch + delay from Start access time to first data capture = 0ns + 60 ns = 16 GPMC clock cycles	0x08
	WRCYCLETIME	WRACCESSTIME + Data holding = 34 GPMC clock cycles	0x11
nCS	CSONTIME	Assert after address latch	0x00
	CSWROFFTIME	WRACCESSTIME + Data holding = 30 + 4 = 34 GPMC clock cycles	0x11
nADV	ADVONTIME	Immediate assert with read cycle	0x00
	ADVOFFTIME	Provide ADV assertion duration of 2 cycles	0x01
WE	WEONTIME	Assert after address latch	0x00
	WEOFFTIME	WRACCESSTIME + Data holding	0x11

14.1.3 Test Results

This section provides the test results for Asynchronous NAND Flash read/write using CPU Prefetch mode.

Table 56. Asynchronous NAND Flash Read/Write using CPU Prefetch Mode

Initiator/Operation	Source	Destination	Size (KB)	Bandwidth (MB/s) (with ECC)	Bandwidth (MB/s) (without ECC)
Flash Read	NAND Flash	DDR	4	8.63	10.76
	NAND Flash	DDR	16	8.64	10.77
	NAND Flash	DDR	128	8.64	10.77
Flash Write	DDR	NAND Flash	8	6.06	6.40
	DDR	NAND Flash	32	6.06	6.42
	DDR	NAND Flash	128	6.06	6.41

Table 57. Asynchronous NOR Flash Single Read by CPU

Transfer Type	Source	Destination	Transfer Size (KB)	Throughput
CPU Flash read	NOR Flash	DDR	16	7.434104
CPU Flash read	NOR Flash	DDR	32	7.429332
CPU Flash read	NOR Flash	DDR	64	7.426684
CPU Flash read	NOR Flash	DDR	128	7.424434

Table 58. Asynchronous NOR Flash Single Read by DMA

Transfer Type	Source	Destination	Transfer Size (KB)	Throughput
DMA Flash read	NOR Flash	DDR	16	13.2409
DMA Flash read	NOR Flash	DDR	32	13.2477
DMA Flash read	NOR Flash	DDR	64	13.2478
DMA Flash read	NOR Flash	DDR	128	13.2489

Table 59. NOR Flash Page Read by CPU (Page Length: 16 × 16 bit)

Transfer Type	Source	Destination	Transfer Size (KB)	Throughput
CPU Flash read	NOR Flash	DDR	16	10.17551
CPU Flash read	NOR Flash	DDR	32	10.17004
CPU Flash read	NOR Flash	DDR	64	10.17551
CPU Flash read	NOR Flash	DDR	128	10.17402

Table 60. Asynchronous NOR Flash Page Read by DMA (Page Length: 16 × 16 bit)

Transfer Type	Source	Destination	Transfer Size (KB)	Throughput
DMA Flash read	NOR Flash	DDR	16	49.6268
DMA Flash read	NOR Flash	DDR	32	49.7215
DMA Flash read	NOR Flash	DDR	64	49.7809
DMA Flash read	NOR Flash	DDR	128	49.7867

14.2 Summary

- GPMC timing parameters have to be set according to the minimum flash timing requirements to get the optimized performance.
- Different knobs in the controller such as prefetch mode, burst mode and different knobs supported by flash such as page mode improves the flash read throughput.
- DMA reads give improved performance compared to the CPU because of the 128-byte burst access.

15 QSPI IP

The QSPI is a Serial Peripheral Interface module that allows for single, dual, or quad read access to external memory devices. The Quad SPI protocol is a 6-pin interface that supports master mode only in 3-pin, 4-pin, or 6-pin configurations. Up to 4 chip selects are supported. This Serial Flash Interface does not directly support all details of writing to serial Flash devices and also does not support dual/quad write. For more details, see the *Quad SPI* section of the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).

This device supports a normal configuration port mode as well as a memory-mapped interface, which provides a direct memory interface for accessing data from the external SPI device, simplifying software requirements.

- Configuration Port Mode: In this mode, commands can be sent through the core SPI to communicate with a serial Flash device, but software must load the command into the SPI data transfer register along with additional configuration fields, perform the byte transfer, then place the data to be sent (or configure for receive) along with additional configuration fields, and perform that transfer. Reads and writes to serial Flash devices are more specific. First, the read or write command byte is sent, followed by 1 to 4 bytes of address (corresponding to the address to read/write), then followed by the data write/receive phase. Data is always sent in 8-bit chunks (byte oriented). Once the address has been loaded, data can be continuously read or written and internally to the serial Flash device, the address will automatically increment to each byte address.
- Memory-Mapped Mode: This mode supports long transfers through a frame style sequence. In its generic SPI use mode (referred to as core SPI), a word can be defined up to 32 bits and multiple words can be transferred during a single access. For each word, the processor will need to read or write the new data and then tell the SPI module to continue the current operation. Using this sequence, up to 4096 words (32-bit max) can be transferred in a single SPI read or write operation. While this allows for the greatest flexibility in terms of connecting to various types of devices, it does not lend itself to a memory-mapped type device such as a serial Flash device.

A block diagram representation of QSPI controller is shown in Figure 40.

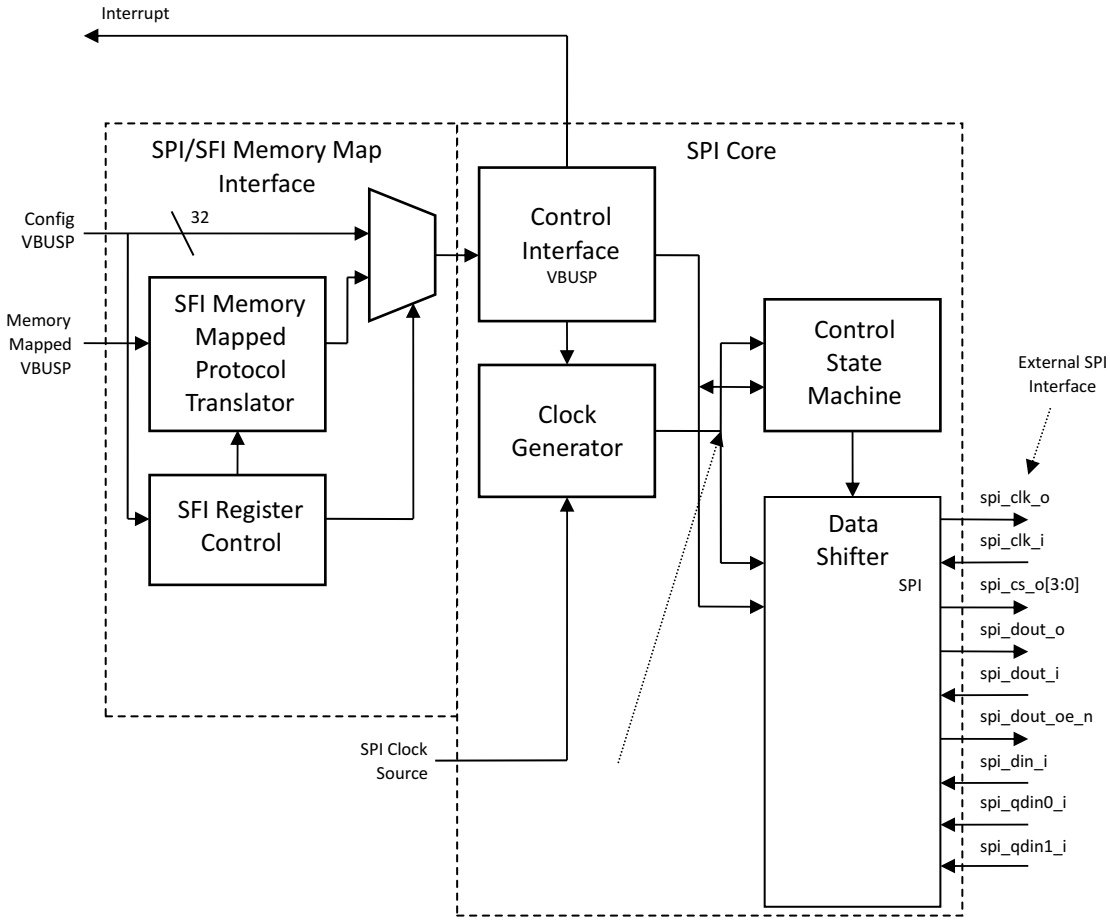


Figure 40. QSPI Block Diagram

15.1 QSPI Read and Write Performance

15.1.1 Test Setup

The tests are performed on the TDA2xx and TDA2ex Pre-silicon (Quick-turn) and Post-silicon platforms. Configuration used for the performance numbers are:

- Cortex-A15 Frequency: 1176 MHz
- DDR3 Clock: 532 MHz
- L3 Frequency: 266 MHz
- EDMA Burst length: 0x20
- CPU Burst length: 0x10
- QSPI OCP Profiling: 32 bits (Sdata and Mdata is 32 bits)
- QSPI mode: 0x3
- External Flash is configured in Quad Mode

Memory-mapped SETUP Register is programmed for:

- Read Command = 0x6B
- Number of Address Bytes = 0x2 (3-byte addressing for external flash)
- Number of Dummy Bytes = 0x1 (8 bits as dummy bytes)

- Read Type = 0x3 (Quad Read)

DM Timer was used to time the number of clock cycles it took for data transfer. The Timer Clock for profiling selected as 20 MHz. All throughput data collected “standalone”. No other ongoing traffic.

15.1.2 Test Results

Table 61. QSPI Throughput Using DMA

No.	Activity	Source	Destination	Transfer Size (KB)	SPI Clock (MHz)	Throughput (MB/s)
1	Flash read	Memory-Mapped Flash	OCMC1	4	48	16.3
2	Flash read	Memory-Mapped Flash	OCMC1	8	48	16.5
3	Flash read	Memory-Mapped Flash	OCMC1	16	48	16.5
4	Flash read	Memory-Mapped Flash	OCMC1	4	64	19.5
5	Flash read	Memory-Mapped Flash	OCMC1	8	64	19.5
6	Flash read	Memory-Mapped Flash	OCMC1	16	64	19.5

Table 62. QSPI Throughput Using CPU

No.	Activity	Source	Destination	Transfer Size (KB)	SPI Clock (MHz)	Throughput (MB/s)
1	Flash read	Memory-Mapped Flash	OCMC1	4	48	16.3
2	Flash read	Memory-Mapped Flash	OCMC1	8	48	16.5
3	Flash read	Memory-Mapped Flash	OCMC1	16	48	16.5

15.1.3 Analysis

15.1.3.1 Theoretical Calculations

Theoretical maximum bandwidth is 30.5 MB/s in quad mode, if the SPI clock is 64 MHz (4 bits per clock cycle in quad mode: $((64 \times 10^6) \times 4)/(8 \times 1024 \times 1024) = 30.5$ MB/s

15.1.3.2 % Efficiency

A typical Read transfer consists of:

- Total clks = CMD + ADDR + Dummy Byte(/s) + spi_clks between dummy byte tx to Read start + (reads spi_clks per trans × number of trans per CS# low) + (idle spi_clks between two reads × number of reads in CS# low - 1) + end of read to start of new read
- So, total spi_clks = 8 + 24 + 8 + 8 + (32 × 8) + (8 × 7) + 13 = 373
- Actual spi_clks in a transfer (CS# low) = 32 × 8 = 256
- Typical % efficiency = (256/373) × 100 = 68.63%
- A throughput of 19.5 MB/s (typically for a EDMA transfer). So the typical efficiency will be: (19.5/30.5) × 100 = 64%
- 4% efficiency delta could be because of 20-MHz timer that is used to measure count for transfers.

15.2 QSPI XIP Code Execution Performance

In order to understand the impact of executing code from QSPI flash in XIP mode, the following test setup was used on the TDA2xx device:

- QSPI configured to Mode 0 and 64 MHz operating frequency
- Vision SDK (version 2.9) IPU application modified to run out of QSPI. The TI Vision SDK is a multi-processor software development platform for TI's family of ADAS SoCs. For more information, see the *TI Vision SDK, Optimized Vision Libraries for ADAS Systems* (SPRY260). The software framework allows users to create different ADAS application data flows involving video capture, video pre-processing, video analytics algorithms and video display.
- Cortex M4 Unicache enabled
- Cortex M4 operating at 212 MHz

Table 63 provides a comparative analysis of the impact of QSPI XIP code execution versus DDR Cortex-M4-based Capture Display Vision SDK. Note Cortex-M4 frequency is 212 MHz. The FPS was found to match between DDR and QSPI XIP code execution. (30 FPS)

Table 63. M4_0 CPU Execution Time in QSPI XIP Mode

Scenario	M4_0 CPU Task Load (%) (load can vary by 2 % in different runs)
M4_0 code execution from DDR3 532 MHz	Approximately 6.2 %
QSPI XIP (64 MHz clock frequency, Mode 0)	Approximately 10.91 %

In order to understand the impact of the QSPI Code execution for a fully loaded M4 CPU, the networking usecase was also run with the capture display usecase. In order to run the networking threads on the M4_0 core, the Network Development Kit (NDK) (http://www.ti.com/lit/ug/spru524j/spru524j.pdf) windows application was run as shown below:

```
ndk_2_24_02_31\packages\ti\ndk\winapps>send <IPAddress> 2000
```

This tool prints out the number of megabytes of data that were sent by the tool and serviced by the TDA2xx device (M4_0) running the network stack. The M4_0 is 100% loaded in the following experiments. Table 64 provides the comparison of the achieved network throughput at different device conditions.

Table 64. M4_0 CPU Networking Bandwidth Performance

Networking Bandwidth Achieved (all numbers mega Bytes per second)	QSPI4 (64 MHz)	DDR3 532 MHz
M4 (212.8 MHz)	3.05	5.26

When there is a concurrent EDMA transfer from QSPI to DDR (possible application image copy from QSPI) while the M4_0 is executing code out of QSPI, there is a significant impact on the M4_0 code execution time. The impact on M4 code execution for varying EDMA ACNT parameter for AB_SYNC and ASYNC transfers is shown in Table 65.

Table 65. M4_0 CPU Networking Bandwidth Performance for Different EDMA ACNT Values

Networking Bandwidth Achieved With concurrent EDMA Traffic M4 @ 212.8 MHz QSPI @ 64 MHz	A_SYNC (Bandwidth in MBps)	AB_SYNC With BCNT = 512 (Bandwidth in MBps)
ACNT = 65535	0.0118	0.0046
ACNT = 16384	0.0379	0.0046
ACNT = 4096	0.233	0.0050
ACNT = 512	1.638	0.0088
ACNT = 256	2.048	0.0121
ACNT = 128	2.559	0.0122
ACNT = 64	2.730	0.0186
Without EDMA	2.935	2.935

The impact on M4 traffic can be controlled by using bandwidth limiter on the EDMA Read from QSPI. Table 66 provides the impact on performance for M4_0 code running the (1) network usecase + Capture and Display and (2) Capture-display usecase only in two independent runs.

Table 66. M4_0 CPU Networking Bandwidth Performance for Concurrent EDMA Traffic at Different EDMA Throughputs

BW Limited EDMA TPUT (ACNT = 65535, A SYNC)	M4 Networking Performance (MBps) QSPI4 (64 MHz)	M4 Capture Display Total CPU Load (%) QSPI4 (64 MHz)
22.46 MBps	0.0118 MBps	99.9 %
17.86 MBps	0.621 MBps	40.1 %
8.98 MBps	1.772 MBps	19.2 %
Without EDMA	2.935 MBps	13.2 %

The impact of the performance of the IPU code can also be understood by looking at the traffic profile using L3 statistic collectors. Figure 41 through Figure 45 shows how the IPU traffic is impacted with a concurrent EDMA traffic and how the performance of IPU can be recovered using BW limiters on the EDMA Read from QSPI. QSPI is operating at 64 MHz in all the below BW plots.

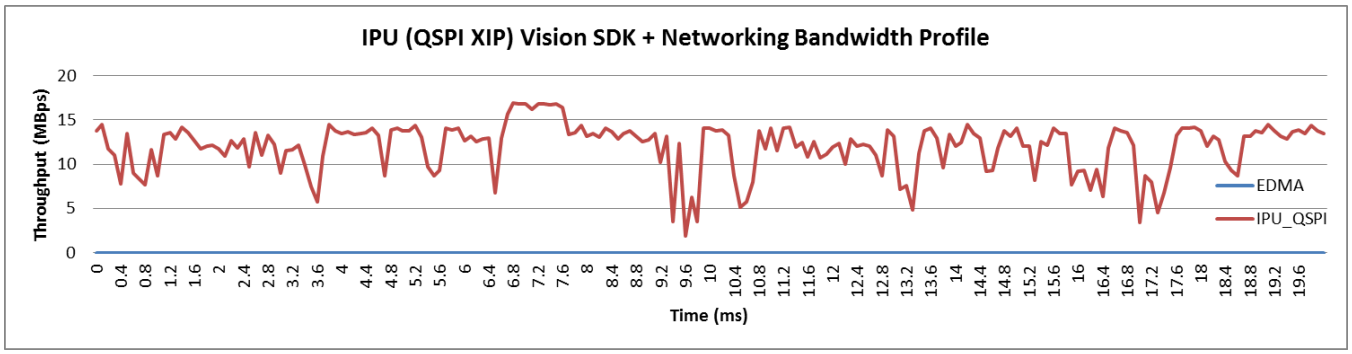


Figure 41. IPU (QSPI XIP) Vision SDK + Networking Bandwidth Profile

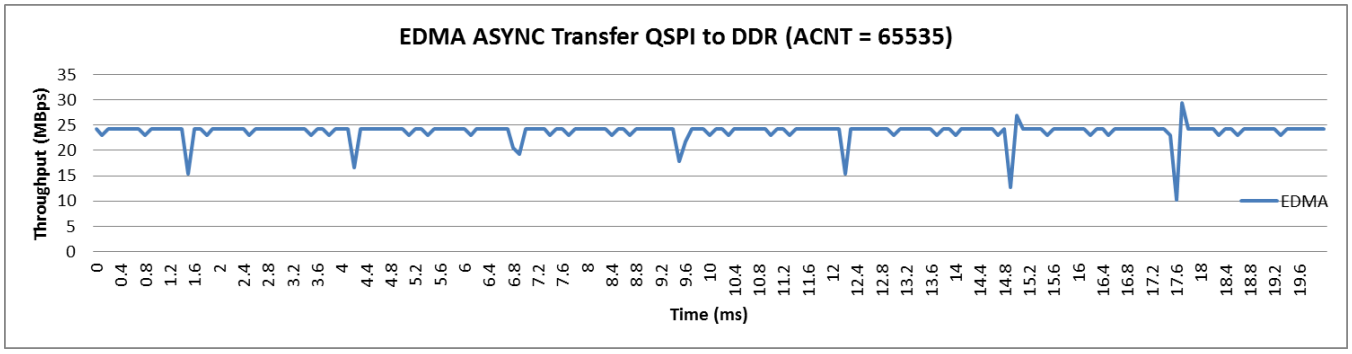


Figure 42. EDMA ASYNC Transfer QSPI to DDR (ACNT = 65535)

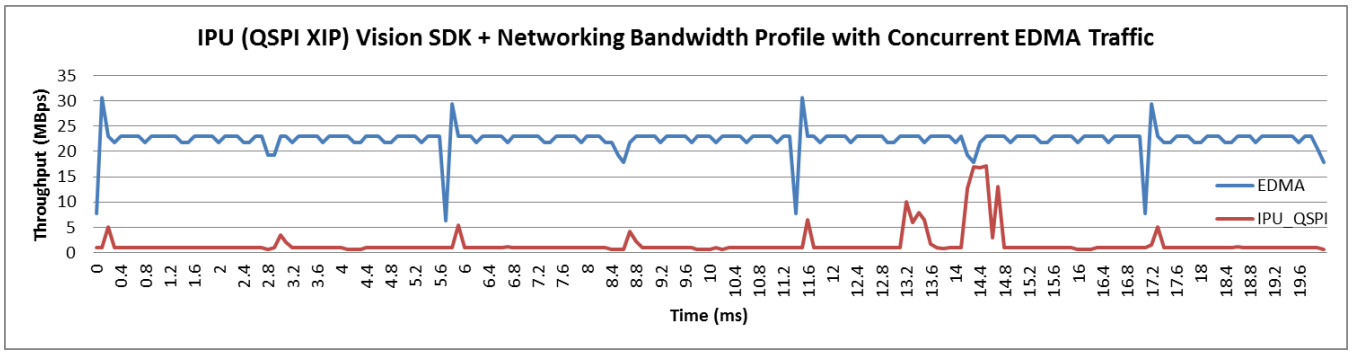


Figure 43. IPU (QSPI XIP) Vision SDK + Networking Bandwidth Profile With Concurrent EDMA Traffic

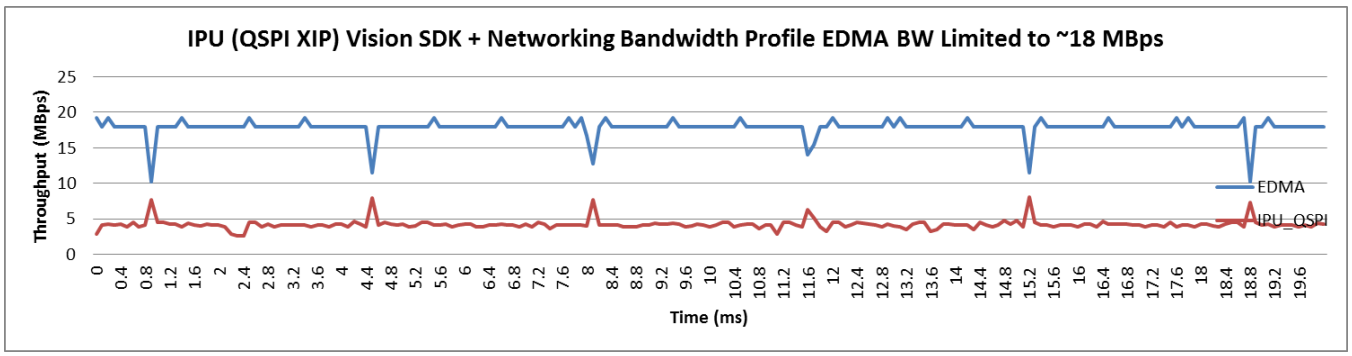


Figure 44. IPU (QSPI XIP) Vision SDK + Networking Bandwidth Profile EDMA BW Limited to Approximately 18 MBps

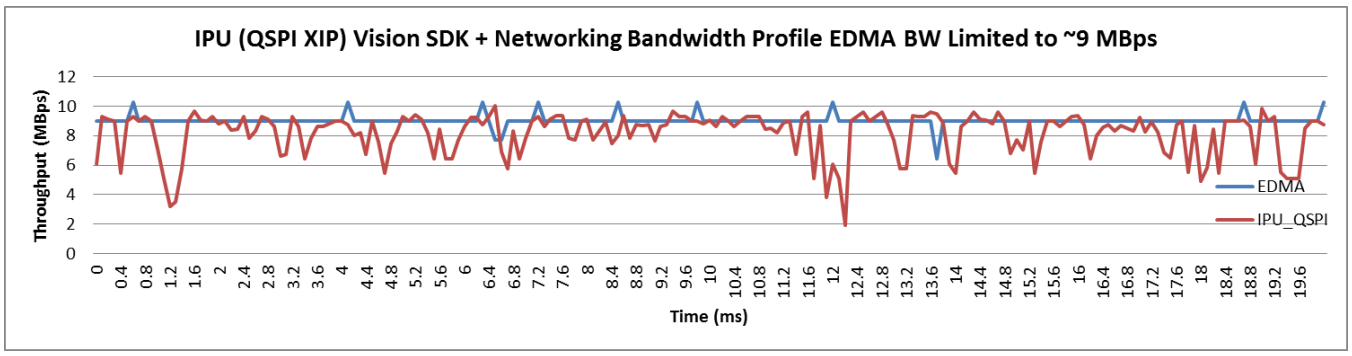


Figure 45. IPU (QSPI XIP) Vision SDK + Networking Bandwidth Profile EDMA BW Limtied to Approximately 9 MBps

15.3 Summary

- Current QSPI reads from external flash are not continuous.
- This is due to the asynchronous nature of SPI_clk (used to drive shift register) and ocp_clk. Your design needs to wait for “word done” and “next command issue” from design before communicating with external flash.
- This causes the design to wait for some clk cycles (typically 7-8 external spi_clk cycles) before receiving next “word” from external flash.
- Due to this wait time, QSPI throughput comes down to an efficiency of 82% than actually supported by the external device.
- QSPI XIP performance is comparable to 60 % of DDR performance when operating at 64 MHz.
- There is a significant impact on the CPU QSPI XIP performance when there is concurrent EDMA copy from QSPI. Application developers should use ASYNC EDMA transfers with lower ACNT or BW limiter to balance the share of the CPU and DMA traffic to the CPU for the CPU traffic to not get starved. The application developer can choose the BW limit/ACNT based on the priority of the application image load versus the IPU CPU performance.

16 Standard Benchmarks

This section covers standard benchmark results of the TDA2xx and TDA2ex devices, for example, Dhrystone, LMbench, STREAM, Whetstone, and Core mark.

16.1 Dhrystone

The Dhrystone benchmark was designed to test performance factors important in non-numeric systems programming (operating systems, compilers, word processors, and so on). Some important features of Dhrystone are:

- Contains no floating-point operations.
- A considerable percentage of time is spent in string functions making the test very dependent upon the way such operations are performed (for example, by in-line code or routines written in assembly language) making it susceptible to manufacturers 'tweaking' of critical routines.
- Contains hardly any tight loops so in the case of very small caches, the majority of instruction accesses will be misses; however, the situation changes radically as soon as the cache reaches a critical size and can hold the main measurement loop.
- Only a small amount of global data is manipulated (as opposed to Whetstone).
- The output is the number of Dhrystones per second (the number of iterations of the main code loop per second).
- The industry has adopted the VAX 11/780 as the reference 1 MIP machine. The VAX 11/780 achieves 1757 Dhrystones per second.

The DMIPS Calculation is shown in Equation 1.

$$\text{DMIPS / MHz} = \frac{(1/\text{Execution Time}) * \text{Number\_of\_Runs}}{1{,}757 * \text{MCU Frequency in MHz}} \quad (1)$$

- cycles_per_run = Clock_cycles/Number_Of_Runs
- User_Time = Clock_cycles/(CPU cycles per second)
- Dhrystones_Per_Second = Number_Of_Runs/User_Time
- DMIPS = Dhrystones_Per_Second/1757.0
- DMIPS/MHz = DMIPS/(CPU cycles per second)

16.1.1 Cortex-A15 Tests and Results

Test setup:

- Compile options used:

```
-mv7A8 --code_state=32 --abi=eabi -me -O3
--opt_for_speed=5
--include_path="C:/ti/ccsv5/tools/compiler/arm_5.1.1/include"
--include_path="..."
--diag_warning=225 --display_error_number --asm_listing
```

- Icache and Dcache enabled
- MMU enabled
- Branch prediction enabled

TDA2xx and TDA2ex Cortex-A15 has a score of 2.95 DMIPS/MHz for the first running 3 seconds.

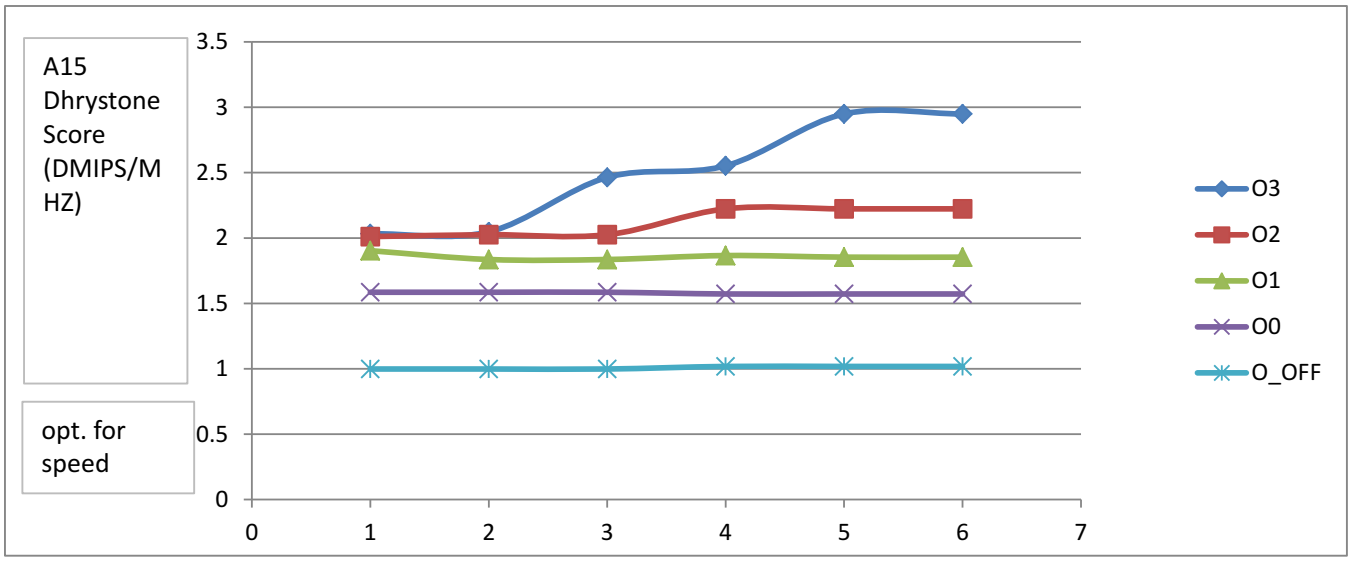


Figure 46. DMIPS Numbers Trend With Optimization Level (-O) Change and Speed Option (-opt_for_speed)

16.1.2 Cortex-M4 Tests and Results

Test setup:

- Compile options used:

```
-mv7M4 --code_state=32 --abi=eabi -me -O3
--opt_for_speed=5
--include_path="C:/ti/ccsv5/tools/compiler/arm_5.1.1/include"
--include_path="..."
--diag_warning=225 --display_error_number --asm_listing
```

- Unicache enabled
- MMU enabled
- Program code, data, variables, and constants stored in L2 RAM.

TDA2xx and TDA2ex Cortex-M4 has a score of 1.0649 DMIPS/MHz for the running 3 seconds. The score is close to the ARM reference of 1.25 DMIPS/MHz per core with difference due to the TI compiler optimization.

16.2 LMbench

LMbench is a widely-used memory benchmark in the embedded systems benchmarking. This section also covers a brief comparison with a predecessor device AM435x.

Some features of LMbench are:

- A kernel benchmark developed by Larry McVoy.
- LMbench is still active as of 2007, with Carl Staelin acting as maintainer.
- Suite of simple, portable, ANSI/C micro benchmarks for UNIX/POSIX.
- Compares different UNIX systems performance.
- Measure system latency and bandwidth of data movement among the processor and memory, network, file system, and disk.
- Focus on latency and bandwidth because performance issues are usually caused by latency problems, bandwidth problems, or some combination of the two.
- Version 3A15 includes a total of approximately 41 metrics divided under the categories of bandwidth, latency, and others.
  - Out of the 41 micro benchmarks, bandwidth and latency micro benchmarks are explored in the next sections.

16.2.1 LMbench Bandwidth

bw_mem is the bandwidth micro benchmark of LMbench. It allocates twice the specified amount of memory, zeroes it, and then times the copying of the first half to the second half. Results are reported in megabytes moved per second. The size specification may end with ‘‘KB’’ or ‘‘MB’’ to mean kilobytes or megabytes.

Following are the functions part of the bw_mem benchmark:

- Stride 4 read: this is a read of stride 4 (also referred to as a partial read, read first 4 bytes for every 16 bytes, skip the rest, and continue)
- Stride 4 write: this is a write of stride 4 (also referred to as a partial write, write first 4 bytes for every 16 bytes, skip the rest, and continue)
- Stride 4 read write: this is a read and write to the same memory address of stride 4
- Stride 4 copy: this is a read and write to the different memory address of stride 4
- Stride 1 write: this is a write of stride 1
- Stride 1 read: this is a read of stride 1
- Stride 1 copy: this is a copy (different memory address) of stride 1

Following are the functions not part of the bw_mem benchmark but present in libraries:

- Memset : bzero
- Memmove : bcopy

16.2.1.1 TDA2xx and TDA2ex Cortex-A15 LMbench Bandwidth Results

The results of TDA2xx and TDA2ex Cortex-A15 for the bandwidth benchmark are shown in Table 67.

Table 67. TDA2xx and TDA2ex Cortex-A15 LMbench Bandwidth Micro Benchmark Results

Benchmark Function	Function Name	Memory Block Size (kB)	TDA2xx and TDA2ex Cortex-A15 Bandwidth (MB/s)	AM437x Bandwidth (MB/s)
Stride 4 read	rd	4096	2026.34	194.73
Stride 4 write	wr	4096	1195.27	165.92
Stride 4 read write	rdwr	4096	852.81	165.93
Stride 4 copy	mcp	4096	431.05	114.78
Stride 1 write	fwr	4096	1138.87	1592.78
Stride 1 read	frd	4096	569.03	158.13
Stride 1 copy	fcp	4096	700.01	128.39

16.2.1.2 TDA2xx and TDA2ex Cortex-M4 LMBench Bandwidth Results

The results of TDA2xx and TDA2ex Cortex-M4 for the bandwidth benchmark are shown in Table 68.

Table 68. TDA2xx and TDA2ex Cortex-M4 LMbench Bandwidth Micro Benchmark Results

Benchmark Function	Function Name	Memory Block Size (kB)	TDA2xx and TDA2ex Cortex-M4 Bandwidth (MB/s)
Stride 4 read	rd	4096	97.70
Stride 4 write	wr	4096	1064.50
Stride 4 read write	rdwr	4096	87.62
Stride 4 copy	mcp	4096	74.32
Stride 1 write	fwr	4096	76.78
Stride 1 read	frd	4096	268.10
Stride 1 copy	fcp	4096	56.06

16.2.1.3 Analysis

- The AM435x device has a Cortex-A9 running at 500 Hz, uses a 16-bit DDR2, a different architecture with the same MMU and cache settings as in the TDA2xx and TDA2ex Cortex-A15.
- Cortex-A15 LMbench bandwidth micro benchmark Stride 1 read/write/copy bandwidth numbers are similar to what is obtained using the C-functions from the system performance test cases.
- A 16-bit DDR2 versus a 32-bit DDR2 difference is clearly seen with respect to Stride 1 and Stride 4 writes. The read traffic profile of the Cortex-A9 seems to not really take advantage of the DDR bus width; while in the case of the Cortex-A15, the difference between Stride 1 and Stride 4 performance is clearly seen.
- Cortex-M4 LMbench bandwidth micro benchmark Stride 1 read/write/copy performance numbers are matching with performance numbers obtained using the C-functions from the system performance test cases.

16.2.2 LMbench Latency

Lat_mem_rd is the bandwidth micro benchmark of LMbench.

- lat_mem_rd measures memory read latency for varying memory sizes and strides. The results are reported in nanoseconds per load.
- The entire memory hierarchy is measured, including on-board cache latency and size, external cache latency and size, main memory latency, and TLB miss latency.
- Only data accesses are measured; the instruction cache is not measured.
- The benchmark runs as two nested loops. The outer loop is the stride size. The inner loop is the array size. For each array size, the benchmark creates a ring of pointers that point backward one stride. Traversing the array is done by p = (char **)*p; in a for loop (the over head of the for loop is not significant; the loop is an unrolled loop 100 loads long).
- The size of the array varies from 512 bytes to (typically) 8 megabytes. For the small sizes, the cache will have an effect and the loads will be much faster. This becomes much more apparent when the data is plotted.
- Default stride length is 128.

The result of the latency micro benchmark plots are shown in the next sections. The index shown has the format Bare metal (BM) – device name (CPU Frequency : DDR frequency). Bare metal since it is done in a no-operating system environment.

16.2.2.1 TDA2xx and TDA2ex Cortex-A15 LMbench Latency Results

The results of the TDA2xx and TDA2ex Cortex-A15 for the LMbench latency benchmark are shown in Figure 47.

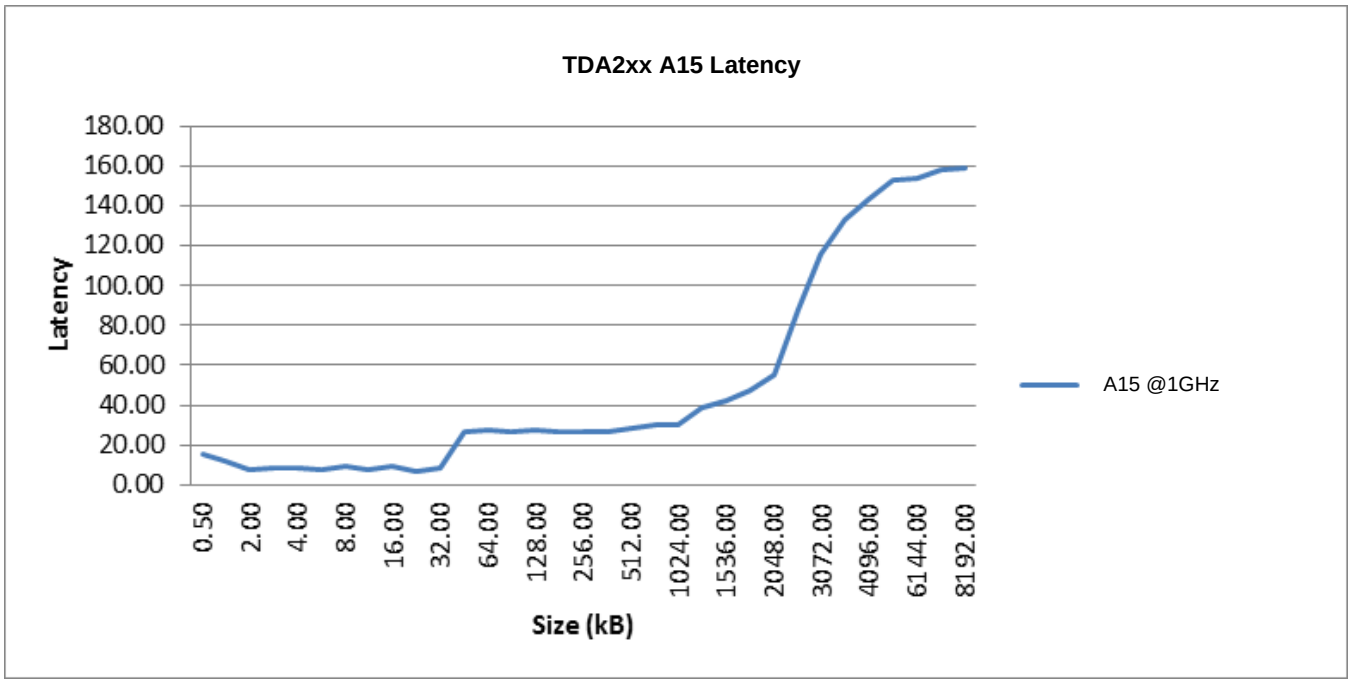


Figure 47. TDA2xx and TDA2ex Cortex-A15 LMbench Latency Results

16.2.2.2 TDA2xx and TDA2ex Cortex-M4 LMbench Latency Results

The results of the TDA2xx and TDA2ex Cortex-M4 for the LMbench latency benchmark are shown in Figure 48.

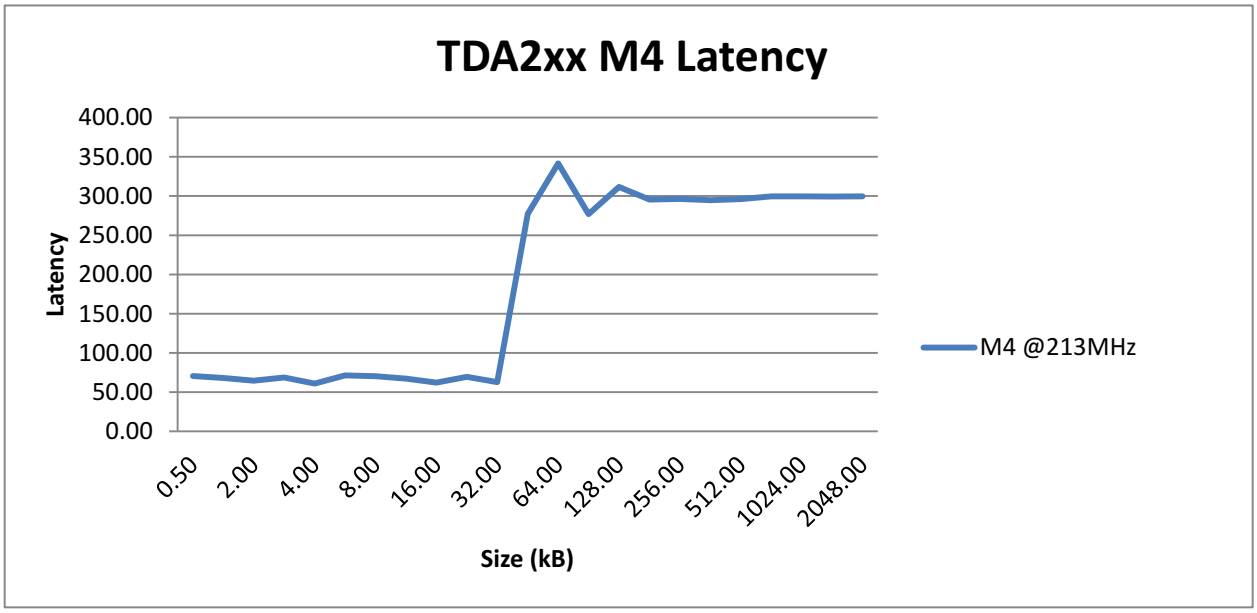


Figure 48. TDA2xx and TDA2ex Cortex-M4 LMbench Latency Results

16.2.2.3 Analysis

- The latency plot of any SoC device usually has n plateaus, if there are n levels to reach the external memory after all the cache boundaries.
- The Y-axis is in nano seconds (ns) and the X-axis is in KiloBytes (KB).
- In the TDA2xx and TDA2ex device, there are 2 levels of cache (L1, L2) and then the external memory is on the L3. From the latency plot, you can find out how many levels of cache an SoC has and what are their approximate cache sizes; from the plot, you can see that there are 3 plateaus.
- L1 plateau ends by ~32 KB (cache size)
- L2 plateau ends by ~2 MB (cache size)
- L3 plateau starts after 2MB (L2) and the peak of saturation at DDR2 latency.
- The DDR2 latency is very high due to the limitation mentioned in Section 2.4.2.

16.3 STREAM

The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. STREAM has 2 versions and each version performs 4 operations. We worked on version 1, which consists of the following 4 functions:

- Copy => a[i] = b[i]
- Scale => a[i] = k × b[i]
- Add => a[i] = b[i] + c[i]
- Triad => a[i] = b[i] + k × c[i]

Stream is integrated as a part of the latest LMbench suite.

Stream version 2 has the same copy function, zero, fill, sum, and daxpy (triad-like) functions.

16.3.1 TDA2xx and TDA2ex Cortex-A15 STREAM Benchmark Results

The results of the TDA2xx and TDA2ex Cortex-A15 for the STREAM benchmark are shown in Figure 49.

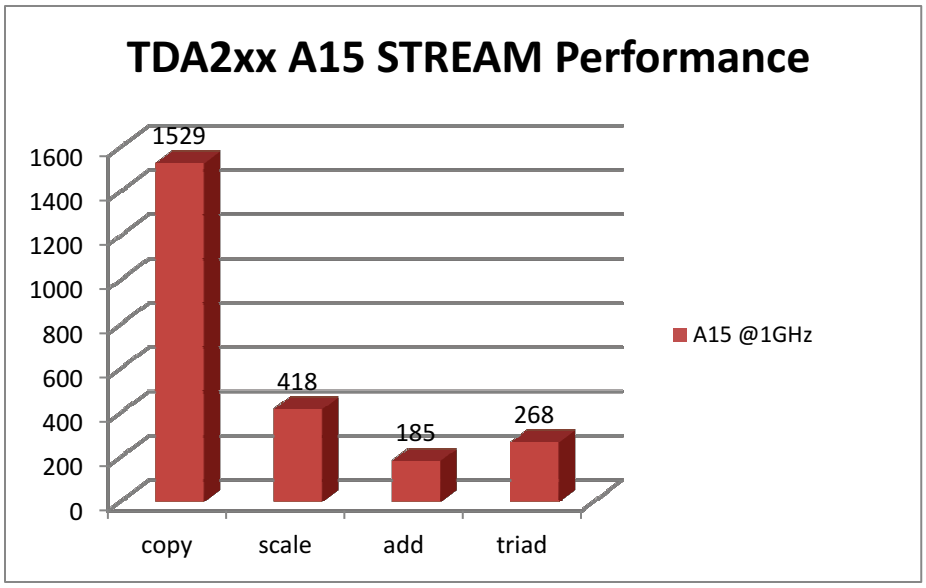


Figure 49. TDA2xx and TDA2ex Cortex-A15 STREAM Benchmark Results

16.3.2 TDA2xx and TDA2ex Cortex-M4 STREAM Benchmark Results

The results of the TDA2xx and TDA2ex Cortex-M4 for the STREAM benchmark are shown in Figure 50.

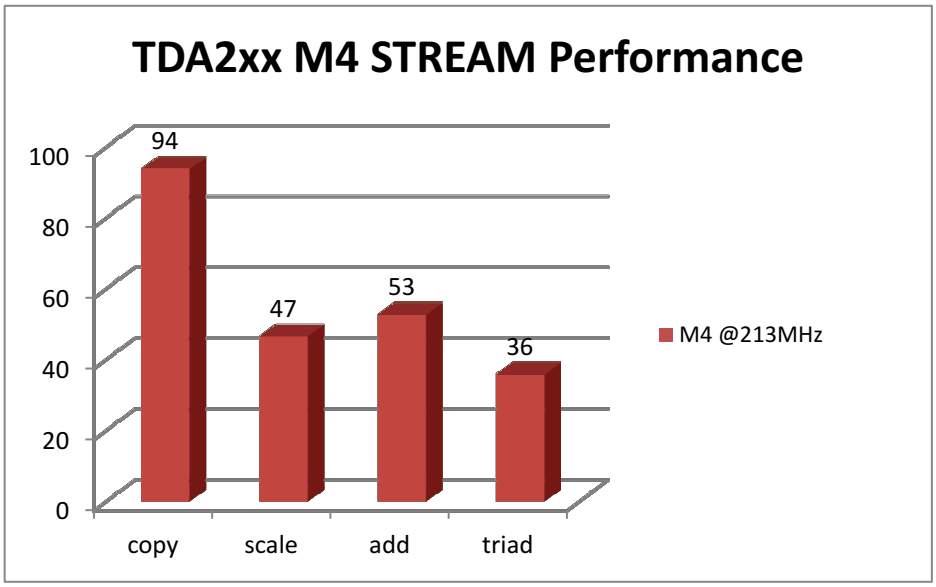


Figure 50. TDA2xx and TDA2ex Cortex-M4 STREAM Benchmark Results

17 Error Checking and Correction (ECC)

TDA2xx and TDA2ex Error Checking and Correction (ECC) implemented on the OCMC RAM 1, 2, and 3, and EMIF. The ECC memory wrappers can perform single error correction and double error detection.

Overview of the OCMC ECC features are:

- Error correction and detection: Single Error Correction and Dual Error Detection (SECDED)
  - 9-bit Hamming Error Correction Code (ECC) calculated on 128 data word concatenated with memory address bits
  - Hamming distance of 4 (single error detection/correction and double error detection; triple error detection (TED) is NOT supported).
  - Enable/Disable/Test-Suspend Mode Control through control register
  - Read transaction single bit error correction
  - Hardware Automated write back of correctable detected error
  - Exclude repeated addresses from correctable error address trace history
  - ECC valid for all write transactions to enabled region
  - 128-bit aligned/128-bit length writes have no additional overhead
  - Sub 128-bit writes supported by way of read-modify-write
- ECC Error Status Reporting features:
  - Corrected Error Address Trace History Buffer (FIFO): Depth of 4
  - Non-correctable error address trace history buffer (including DED): Depth of 4
  - Interrupt Generation for correctable/uncorrectable detected errors
- ECC Diagnostics Configuration:
  - SEC/DED/Addr Error Event Counters
  - Programmable SEC/DED/Addr Error Event Counter Exception Threshold registers
  - Corrected Single Error bit distribution history
  - Register control for enable and disabling of diagnostics
  - Configuration registers and ECC status accessible through OCP MMR interface (L4)

The EMIF ECC features are:

- ECC on SDRAM data bus.
  - 7-bit ECC over 32-bit quanta or 6-bit ECC over 16-bit quanta in Narrow mode.
  - 1-bit correction and 2-bit detection.
  - Programmable address ranges to define ECC protected region.
  - ECC calculated and stored on all writes to ECC protected address region.
  - ECC verified on all reads from ECC protected address region.
  - Statistics for 1-bit ECC and 2-bit ECC errors.
  - All DDRs must have the same data bus width. The ECC DDR must support the same data bus width as the normal DDR ICs for data. The total width of the ECC DDR data bus is 8 bits.

17.1 OCMC ECC Programming

The programming sequence to use the OCMC RAM ECC feature is:

1. Enable OCMC RAM ECC by configuring CFG_OCMC_ECC[2:0]:CFG_OCMC_MODE in the OCM subsystem, as shown in Table 69.

Table 69. CFG_OCMC_ECC

Bits	Field Name	Description	Type	Reset
31:6	RESERVED	Reserved	R	0x0
5	CFG_ECC_OPT_NON_ECC_READ	Optimize read latency for non-ECC read. Returns the data one cycle faster, if the read access is from a non-ECC enabled space.	RW	0x0
4	CFG_ECC_ERR_SRESP_EN	ECC non-correctable error SRESP enable. Enables ERR return on L3 OCP SRESP when a non-correctable data (DED) or address error is detected.	RW	0x0
3	CFG_ECC_SEC_AUTO_CORRECT	SEC error auto correction mode. Enables the OCMC_ECC to automatically update the error data word with the corrected word.	RW	0x0
2:0	CFG_OCMC_MODE	OCM Controller memory access modes. 000: Non-ECC mode (data access) 001: Non-ECC mode (code access) 010: Full ECC enabled mode 011: Block ECC enabled mode 1xx: Reserved (internally defaults to 000 mode)	RW	0x0

- When enabled, a 9-bit Hamming ECC is calculated and stored for each consecutive 128-bit block of the SRAM. The ECC is calculated based on a code word constructed by concatenating the 128 bits of data with the address bits A21 through A4 of the L3_MAIN. The ECC generated is Hamming(155,146) code and has a Hamming distance of 4. The OCM controller uses this code to validate the content of the SRAM, to correct a single bit error that occurs within the 128-bit boundary or to determine if a non-correctable error has occurred within the 128-bit boundary.
- When SECDED is enabled, for every 128-bit write transaction that starts at an ECC boundary, the ECC is calculated and stored in the ECC bit field for that boundary.
- A sub 128-bit or a non-aligned 128-bit write transaction (with at least one deasserted write-data byte-enable signal) is handled as a read-modify-write.
- When SECDED is enabled, on every read transaction within an ECC boundary, the ECC is regenerated on the memory address and the 128-bit data read out of the memory. The code is then compared against the ECC value stored at the address. If the two ECCs match, then the data is transferred to the requesting bus master without further exceptions.
- If the two ECCs do not match, then a check is made to determine if the error is correctable or not. If the error is not correctable (double-error), then the starting address of the 128-bit ECC boundary (128-bit MEMORY word address) is stored in the uncorrectable double error address FIFO and the uncorrectable error exception flag is asserted.
- If the error is correctable (single error), the controller first determines if the bit error is in the address, data, or code portion. If the error is located in the data itself, the bit in error is corrected and presented to the requesting bus master and the bit in error is corrected in the SRAM location by the auto re-write feature (if this feature is enabled in the cfg_ecc register). Also, the starting address of the 128-bit ECC boundary in which the detected error occurred is pushed to the correctable error event address FIFO and the correctable error event counter is incremented by 1. If the bit error is located in the address portion of the quanta, then an address error exception is generated. If the ECC code bit-error (which indicates a single bit error in the code word) is detected, the data is returned to the bus master unchanged but the single error counter is incremented.

2. Given that the ECC parity bits are uninitialized when TDA2xx and TDA2ex is first powered on and the parity is calculated every 128 bits, it is important for the software to first initialize every 128 bits that the code will read from after enabling the ECC.
   - This step would ensure that the parity bits are correctly set before the code access to ECC enabled regions.
   - This step would ensure that NC ECC error scenarios are not generated for uninitialized parity data.
   - This step can be done by either the CPU memset or the EDMA transfer to the memory region.
   - The value of the data with which the ECC enabled space is initialized does not matter. Initializing the full 128-bit line is important as parity will not be set correctly if only a portion of the 128 bits is written to. The following example illustrates this concept.
   - Example: Consider the state of the memory as shown in Table 70. Address 0x40300010 to 0x4030001F is initialized with some known byte pattern. The corresponding 9 bits of ECC corresponding to this is also initialized. When initializing only word 0x40300020, the OCMC ECC controller treats this as a read-modify-write and, hence, causes the ECC controller to read the uninitialized parity and data, check for ECC errors and then modify the parity based on the new word written and write the data and the partially initialized parity into the ECC parity memory. This sequence can lead to NC ECC errors being generated as the ECC parity bits are not in a known state.

Table 70. OCMC ECC Programming Example

OCMC Memory	0	4	8	C	ECC
0x40300000	32-bit word	32-bit word	32-bit word	32-bit word	9-bit Parity
0x40300010	Initialized	Initialized	Initialized	Initialized	Initialized
0x40300020	Initialized	Un-Initialized	Un-Initialized	Un-Initialized	Partially Initialized
...					

When using an EDMA to initialize the memory, the EDMA would access OCMC using a 128-bit access and thus initialize the whole 128 bits in one OCP write command. Care should be taken to ensure the start address and end address is 128-bit aligned when initializing.

NOTE: When un-cached CPU memset is used to initialize the memory, care should be taken to clear the error counts by writing to the CFG_OCMC_ECC_CLEAR_HIST register in the OCM subsystem, as shown in Table 71, as the NC errors may get set when initializing the memory word by word or byte by byte (sub 128 bit) as shown in the previous example.

NOTE: When cache policy of Write Back (WB)/ Write Allocate (WA) is enabled on the CPU, care should be taken to not have the cache lines being read from uninitialized ECC memory. Typically when the CPU cache is enabled in WB-WA mode and the CPU is trying to initialize the OCMC ECC enabled memory, the cache line would be first read into the cache leading to the ECC controller to start reporting ECC errors. Additionally, the initialization would reside in cache unless explicitly flushed to memory. The way to avoid errors from the ECC controller when using cache is to always initialize the ECC enabled memory before performing any read or write to cached ECC enabled memory region.

Table 71. CFG_OCMC_ECC_CLEAR_HIST

Bits	Field Name	Description	Type	Reset
31:4	RESERVED	Reserved	R	0x0
3	CLEAR_SEC_BIT_DISTR	Clear stored SEC bit distribution history. Write of 1 causes the STATUS_SEC_ERROR_DISTR registers to be cleared. Reads return 0.	R/W1C	0x0
2	CLEAR_ADDR_ERR_CNT	Clear stored ADDR error history. Write of 1 causes the ADDR_ERROR_CNT bit and ADDR_ERROR_ADDRESS_TRACE FIFO to be cleared. Reads return 0.	R/W1C	0x0
1	CLEAR_DED_ERR_CNT	Clear stored DED error history. Write of 1 causes the DED_ERROR_CNT bit and DED_ERROR_ADDRESS_TRACE FIFO to be cleared. Reads return 0.	R/W1C	0x0
0	CLEAR_SEC_ERR_CNT	Clear stored SEC error history. Write of 1 causes the SEC_ERROR_CNT bit and SEC_ERROR_ADDRESS_TRACE FIFO to be cleared. Reads return 0.	R/W1C	0x0

3. Set up the count of the errors for which the interrupt should be triggered using the CFG_OCMC_ECC_ERROR register in the OCM subsystem, as shown in Table 72, before enabling the interrupt to the CPU. Enable interrupts for ECC errors using the INTR0_ENABLE_SET/INTR1_ENABLE_SET registers in the OCM subsystem, as shown in Table 73.

Table 72. CFG_OCMC_ECC_ERROR

Bits	Field Name	Description	Type	Reset
31:25	RESERVED	Reserved	R	0x0
24	CFG_DISCARD_DUP_ADDR	Do not save duplicate error address. 0: Save the duplicated addresses. 1: Save only the unique addresses.	R/W	0x0
23:20	CFG_ADDR_ERR_CNT_MAX	Number of ADDR errors to trigger an interrupt (The value must be > 0 to generate an interrupt).	R/W	0x1
19:16	CFG_DED_CNT_MAX	Number of DED errors to trigger an interrupt (The value must be > 0 to generate an interrupt).	R/W	0x1
15:0	CFG_SEC_CNT_MAX	Number of SEC error to trigger an interrupt (The value must be > 0 to generate an interrupt).	R/W	0x1

Table 73. INTR0_ENABLE_SET/INTR1_ENABLE_SET

Bits	Field Name	Type	Reset
31:15	RESERVED	R	0x0
14	CBUF_SHORT_FRAME_DETECT_FOUND	R/W	0x0
13	CBUF_UNDERFLOW_ERR_FOUND	R/W	0x0
12	CBUF_OVERFLOW_WRAP_ERR_FOUND	R/W	0x0
11	CBUF_OVERFLOW_MID_ERR_FOUND	R/W	0x0
10	CBUF_READ_SEQUENCE_ERR_FOUND	R/W	0x0
9	CBUF_VBUF_READ_START_ERR_FOUND	R/W	0x0
8	CBUF_READ_OUT_OF_RANGE_ERR_FOUND	R/W	0x0
7	CBUF_WRITE_SEQUENCE_ERR_FOUND	R/W	0x0
6	CBUF_VBUF_WRITE_START_ERR_FOUND	R/W	0x0
5	CBUF_WR_OUT_OF_RANGE_ERR_FOUND	R/W	0x0
4	CBUF_VIRTUAL_ADDR_ERR_FOUND	R/W	0x0
3	OUT_OF_RANGE_ERR_FOUND	R/W	0x0
2	ADDR_ERR_FOUND	R/W	0x0
1	DED_ERR_FOUND	R/W	0x0
0	SEC_ERR_FOUND	R/W	0x0

4. Resume normal operation of the code.
5. If the ECC logic generates an error that indicates single error detection or double error detection, the following registers can be read to understand the state of the ECC controller:
   - STATUS_ERROR_CNT
   - STATUS_SEC_ERROR_TRACE
   - STATUS_DED_ERROR_TRACE
   - STATUS_ADDR_TRANSLATION_ERROR_TRACE
   - STATUS_SEC_ERROR_DISTR_0 through STATUS_SEC_ERROR_DISTR_4

17.2 EMIF ECC Programming

The programming sequence to use the OCMC RAM ECC feature is:

1. Enable EMIF ECC by configuring the EMIF1_EN_ECC bit in the CTRL_WKUP_EMIF1_SDRAM_CONFIG_EXT register in the TDA2xx and TDA2ex Control Module Registers, as shown in Table 74.

Table 74. CTRL_WKUP_EMIF1_SDRAM_CONFIG_EXT

Address Offset	0x0 0044		
Physical Address	0x4AE00 C144	**Instance**	CTRL_MODULE_WKUP__Core_Registers
Description			
Type	RW		

Bits	Field Name	Description	Type	Reset
31:18	Reserved		RO	0x0000
17	EMIF1_NARROW_ONLY	EMIF1 ECC can be enabled 0: ECC cannot be enabled 1: ECC can be enabled	RW	0
16	EMIF1_EN_ECC			
15:14	EMIF1_REG_PHY_NUM_OF_SAMPLES		RW	0x0
13	EMIF1_REG_PHY_SEL_LOGIC		RW	0
12	EMIF1_REG_PHY_ALL_DQ_MPR_RD_RESP		RW	0
11:9	EMIF1_REG_PHY_OUTPUT_STATUS_SELECT		RW	0x0
8	EMIF1_DYNAMIC_PWRDN_ENABLE		RW	1
7	EMIF1_SDRAM_DISABLE_RESET		RW	0
6:5	EMIF1_PHY_RD_LOCAL_ODT		RW	0x0
4	EMIF1_STATIC_CMOSEN_ENABLE		RW	0
3	EMIF1_DFI_CLOCK_PHASE_CTRL		RW	0
2	EMIF1_EN_SLICE_2		RW	1
1	EMIF1_EN_SLICE_1		RW	1
0	EMIF1_EN_SLICE_0		RW	1

7-bit ECC is calculated over 32-bit data when in 32-bit DDR mode (reg_narrow_mode = 0). 6-bit ECC is calculated over 16-bit data when in 16-bit DDR mode (reg_narrow_mode = 1). The ECC is calculated for all accesses that are within the address ranges protected by ECC.

NOTE: Note that only EMIF1 supports ECC on TDA2xx and TDA2ex.

2. Set the address range for ECC operation. The address ranges are specified in the ECC Address Range 1 and ECC Address Range 2 registers shown in Table 75 and Table 76.

Table 75. EMIF_ECC_ADDRESS_RANGE_1 (0x114)[1][2]

Bits	Field Name	Description	Type	Reset
31:16	REG_ECC_END_ADDR_1	End caddress [31:16] for ECC address range 1. The other 16 LSBs are always 0xFFFF.	R/W	0x0
15:0	REG_ECC_STRT_ADDR_1	Start caddress [31:16] for ECC address range 1. The other 16 LSBs are always 0xFFFF.	R/W	0x0

(1) This register can only be written if lock_config_ctrl port is cleared to 0.
(2) The range is inclusive of the start and end addresses.

Table 76. EMIF_ECC_ADDRESS_RANGE_2 (0x118)[1][2]

Bits	Field Name	Description	Type	Reset
31:16	REG_ECC_END_ADDR_2	End caddress [31:16] for ECC address range 2. The other 16 LSBs are always 0xFFFF.	R/W	0x0
15:0	REG_ECC_STRT_ADDR_2	Start caddress [31:16] for ECC address range 2.The other 16 LSBs are always 0xFFFF.	R/W	0x0

(1) This register can only be written if lock_config_ctrl port is cleared to 0.
(2) The range is inclusive of the start and end addresses.

Example:

```
//EMIF_ECC_ADDRESS_RANGE_1 - 0x80000000 to 0x90000000
WR_MEM_32(0x4C000114, 0x0FFF0000);
//EMIF_ECC_ADDRESS_RANGE_2 - 0x90000000 to 0xA0000000
WR_MEM_32(0x4C000118, 0x1FFF1000);
```

CAUTION

The EMIF ECC region should not overlap; this can lead to unexpected results.

3. Enable ECC on both ranges by writing to the EMIF_ECC_CTRL_REG register as shown in Table 77. This register needs to be programmed to enable ECC, enable the regions and to define whether the ECC needs to be done within the region or outside the region.
Table 77 explains the value of various bit fields.

Table 77. EMIF_ECC_CTRL_REG (0x110)[1]

Bits	Field Name	Description	Type	Reset
31	REG_ECC_EN	ECC enable 0: ECC is disabled. 1: ECC is enabled.	R/W	0x0
30	REG_ECC_ADDR_RGN_PROT	Setting this field to 1 and reg_ecc_en to a 1 will enable ECC calculation for accesses within the address ranges and disable ECC calculation for accesses outside the address ranges. The address ranges can be specified using the ECC Address Range 1 and 2 registers.	R/W	0x0
29:2	RESERVED	Reserved	R	0x0
1	REG_ECC_ADDR_RGN_2_EN	ECC address range 2 enable. 0: ECC address range 2 is disabled. 1: ECC address range 2 is enabled.	R/W	0x0

(1) This register can only be written if lock_config_ctrl port is cleared to 0.

Table 77. EMIF_ECC_CTRL_REG (0x110)[(1)] (continued)

Bits	Field Name	Description	Type	Reset
0	REG_ECC_ADDR_RGN_1_EN	ECC address range 1 enable. 0: ECC address range 1 is disabled. 1: ECC address range 1 is enabled.	R/W	0x0

NOTE: In DDR3 mode, software must trigger PHY initialization and full-leveling/calibration after enabling ECC for the first time. The ECC can then be enabled/disabled for test purposes without triggering full-leveling.

NOTE: ECC enable bit in CTRL_WKUP_EMIF_SDRAM_CONFIG_EXT register should be set even if ECC feature is not used.

Example:

```
if (ENABLE_ECC) //ECC Enabled
    {
        HW_WR_REG32(SOC_CTRL_MODULE_WKUP_CORE_REGISTERS_BASE +
                    CTRL_WKUP_EMIF1_SDRAM_CONFIG_EXT,
                    0x0001C127U); /* EMIF1_EN_ECC = 1 */
    }
```

NOTE: EMIF must be set to non-interleaving mode so that the lower 2-GiB memory can be properly mapped to EMIF1 with ECC support and EMIF2 that doesn't have ECC supported.

Example:

```
    if (MEMMAP_2GB_NON_INTL_EMIFX2)
    {
        printf("Two EMIFs in non interleaved mode (2GB total)\n");
        /* MA_LISA_MAP_i */
        WR_MEM_32(0x482AF040, 0x80600100);
        WR_MEM_32(0x482AF044, 0xC0600200);
        /* DMM_LISA_MAP_i */
        WR_MEM_32(0x4E000040, 0x80600100);
        WR_MEM_32(0x4E000044, 0xC0600200);
    }
```

4. Set the threshold for 1 bit error interrupt - optional.

   Single bit errors are corrected by ECC logic. So user need not get worried about that. But in cases, when there are many single bit errors, this can potentially mean that something in the environment or the memory is not correct. Software may choose to be informed of such condition. For that purpose, EMIF controller provides means to interrupt when single bit error crosses a desired threshold. For details of the register, see Table 78.

Table 78. 1B_ECC_ERR_THRSH – 1 Bit ECC Error Threshold Register (0x0134)

Bits	Name	Description
31:24	REG_1B_ECC_ERR_THRSH	1-bit ECC error threshold. The EMIF will generate an interrupt when the 1-bit ECC error count is greater than this threshold. A value of 0 will disable the generation of interrupt.
23:16	RESERVED	RESERVED
15:0	REG_1B_ECC_ERR_WIN	1-bit ECC error window in number of refresh periods. The EMIF will generate an interrupt when the 1-bit ECC error count is equal to or greater than the threshold within this window. A value of 0 will disable the window. Refresh period is defined by reg_refresh_rate in SDRAM Refresh Control register.

5. Clear the error status - optional.

   This is again an optional step. Before enabling/using ECC, it is a good programming practice to clear any stale ECC error status. Table 79 and Table 80 should be cleared by writing 0x1 to it.

Table 79. EMIF_1B_ECC_ERR_ADDR_LOG – 1 Bit ECC Error Address Log Register (0x013C)

Bits	Name	Description
31:0	REG_1B_ECC_ERR_ADDR	1-bit ECC error address. Most significant bits of the starting address(es) related to the SDRAM reads that had a 1-bit ECC error. This field displays up to two addresses logged in the 4 deep address logging FIFO. Writing a 0x1 will pop one element of the FIFO. Writing a 0x2 will pop all elements of the FIFO. Writing any other value has no effect.

Table 80. EMIF_2B_ECC_ERR_ADDR_LOG – 2 Bit ECC Error Address Log Register (0x0140)

Bits	Name	Description
31:0	REG_2B_ECC_ERR_ADDR	2-bit ECC error address. Most significant bits of the starting address of the first SDRAM burst that had the 2-bit ECC error. Writing a 0x1 will clear this field. Writing any other value has no effect.

6. Enable the interrupts

   In order to receive interrupts, one should set the EMIF interrupt enable in the EMIF configuration space. The register description is shown in Table 81.

Table 81. EMIF_SYSTEM_OCP_INTERRUPT_ENABLE_SET (0x00b4)

Bits	Name	Description
31:6	Reserved	Reserved - writes are ignored, always reads zeros.
5	ONEBIT_ECC_ERR_SYS	Enabled status of sysem ECC one bit error correction interrupt. Writing a 1 will enable the interrupt, and set this bit as well as the corresponding Interrupt Enable Clear Register. Writing a 0 has no effect.
4	TWOBIT_ECC_ERR_SYS	Enabled status of system ECC two bit error detection interrupt. Writing a 1 will enable the interrupt, and set this bit as well as the corresponding Interrupt Enable Clear Register. Writing a 0 has no effect.
3	WR_ECC_ERR_SYS	Enabled status of system ECC Error interrupt when a memory access is made to a non-quanta aligned location. Writing a 1 will enable the interrupt, and set this bit as well as the corresponding Interrupt Enable Clear Register. Writing a 0 has no effect.
2-1	Reserved	Reserved
0	EN_ERR_SYS	Enable set for system OCP interrupt for command and address error. Writing a 1 will enable the interrupt, and set this bit as well as the corresponding Interrupt Enable Clear Register. Writing a 0 has no effect.

   Also, the SoC crossbar should be configured properly to receive the EMIF interrupt to the desired CPU. For example, for A15, SYSBIOS, below code needs to be added in configuration file.

```
var IntXbar    = xdc.useModule('ti.sysbios.family.shared.vayu.IntXbar');
var Hwi        = xdc.useModule('ti.sysbios.family.arm.gic.Hwi');
/* IRQ_CROSSBAR_105 EMIF1_IRQ. Interrupt 57 on A15 */
IntXbar.connectIRQMeta(57, 105);
var hwi_param_0 = new Hwi.Params;
hwi_param_0.arg = 0;
Hwi.create(57+32, '&emifErrIrqIsr', hwi_param_0);
```

7. Initialize ECC enabled memory regions. DDR initialization is needed so that the ECC checksum memory gets initialized properly. Given that the DDR and ECC parity bits are uninitialized when the device is first powered on, it is important for the software to first initialize the whole memory after enabling the ECC.
   - This step ensures that the parity bits are correctly set before the code access to ECC enabled regions.
   - This step ensures that ECC error scenarios are not generated for uninitialized parity data.
   - This step can be done by either the CPU memset or the EDMA transfer to the memory region.

- The initialization requirement is multiple of 2 bytes for narrow_mode = 0, 4 bytes for narrow_mode = 1.
- When using an EDMA to initialize the memory, Care should be taken to ensure the start address and end address is 32- or 16-bit aligned (based on normal or narrow mode) when initializing.

8. ECC programming is now complete, resume normal operation of the code.

There are other steps related to the below that are not listed but can be used and present in software provided by TI:

- Disable ECC
- Clearing interrupts status
- Disabling interrupts
- Getting RAW status of error
- Getting ECC error info

For more description on how to do these, see the TI software. To know the register details, see the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5) and the *TDA2Ex SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.0 Technical Reference Manual* (SPRUI00).

NOTE: When cache policy of WB (Write Back)/ WA (Write Allocate) is enabled on the CPU care should be taken to not have the cache lines being read from uninitialized ECC memory. Typically when the CPU cache is enabled in WB-WA mode and the CPU is trying to initialize the EMIF ECC enabled memory the cache line would be first read into the cache leading to the ECC controller to start reporting ECC errors. Additionally the initialization would reside in cache unless explicitly flushed to memory. The way to avoid errors from the ECC controller when using cache is to always initialize the ECC enabled memory before performing any read or write to cached ECC enabled memory region.

It is recommended to set the ECC enabled DDR memory as non-cacheable and strongly ordered during initialization, and then re-configure the memory regions as cacheable, write-back and write-allocate normal memory.

Example:

```
void setDDRnonCacheable()
{
      UInt64 i;
      Mmu_DescriptorAttrs attrs;
      Mmu_initDescAttrs(&attrs);

      attrs.type = Mmu_DescriptorType_BLOCK;    // BLOCK descriptor
      attrs.shareable = 2;                      // sharerable
      attrs.attrIndx = 1;                       // Non-cache, strongly ordered

//    Mmu_setMAIR(1, 0x04);
      for (i=0x80000000; i < 0xA0000000; i = i + 0x00200000)
            Mmu_setSecondLevelDesc((Ptr)i, (UInt64)i, &attrs);
}
void setDDRcacheable()
{
      UInt64 i;
      Mmu_DescriptorAttrs attrs;
      Mmu_initDescAttrs(&attrs);

      attrs.type = Mmu_DescriptorType_BLOCK;    // BLOCK descriptor
      attrs.shareable = 2;                      // sharerable
      attrs.attrIndx = 2;                       // Cached, normal memory

//    Mmu_setMAIR(2, 0xFF);
      for (i=0x80000000; i < 0xA0000000; i = i + 0x00200000)
            Mmu_setSecondLevelDesc((Ptr)i, (UInt64)i, &attrs);
```

```
}
```

17.3 EMIF ECC Programming to Starterware Code Mapping

ADAS starterware gives example implementation of using EMIF ECC. The STW can be downloaded from Here.

The basic starterware API are : <baseDir>\starterware_xx_xx_xx_xx\drivers\emif.c.

The example file are: <baseDir>\\ starterware_xx_xx_xx_xx \examples\ecc_app.

Table 82 maps the function to the programming sequence.

Table 82. Mapping of Starterware Functions to ECC Programming Steps

Programming Step	Starterware Function
EMIF ECC example	Main.c: emifEccTest()
STEP 1: Enable EMIF ECC by configuring	This is taken care in GEL file or SBL
STEP 2: Set the address range for ECC operation	emif.c: EMIFConfigECCInitECCParams()
STEP 3: Enable ECC on both ranges by writing to the EMIF_ECC_CTRL_REG register	emif.c: EMIFConfigECCInitECCParams() emif.c: EMIFConfigECCEnableEcc
STEP 4: Optional - Set the threshold for 1 bit error interrupt	emif.c: EMIFConfigECCInitECCParams()
STEP 5: Optional – Clear the error status	emif.c: EMIFConfigECCClrAllEccErrInfo()
STEP 6: Enable the interrupts	emif.c: EMIFEnableIntr()
STEP 7: Initialize ECC enabled memory regions	Main.c: emifEccConfig()

17.4 Careabouts of Using EMIF ECC

There are few restrictions that need to be taken into account while using EMIF ECC. This is due to non-availability of read modify write support and silicon errata related to ECC.

17.4.1 Restrictions Due to Non-Availability of Read Modify Write ECC Support in EMIF

In normal mode, 7 bit ECC are computed for 32 bit word whereas in narrow mode, 6 bit ECC is computed for 16bit word. Non availability of read-modify-write support in ECC means that when a ECC sub-quanta (<32bit writes for normal and <16bit writes for narrow) writes happen to EMIF, the ECC does not get computed properly as EMIF does not read back the whole 32/16 bit to compute the ECC for the non-modified portion of the word. This results in below careabouts to be followed while programming.

17.4.1.1 Un-Cached CPU Access of EMIF

Any uncached CPU access to EMIF can result in sub-quanta EMIF writes if the datatype being chosen does not match the ECC quanta size. Table 83 and Table 84 summarize the constraints.

Table 83. ECC Correctness for 32-Bit EMIF for Uncached CPU Data Writes

Memory/Core	Byte	Halfword	Word	Double
A15	Not OK	Not OK	OK	OK
DSP	Not OK	Not OK	OK	OK
EVE	Not OK	Not OK	OK	OK
M4	Not OK	Not OK	OK	OK

Table 84. ECC Correctness for 16-Bit EMIF for Uncached CPU Data Writes

Memory/Core	Byte	Halfword	Word	Double
A15	Not OK	OK	OK	OK
DSP	Not OK	OK	OK	OK
EVE	Not OK	OK	OK	OK

Table 84. ECC Correctness for 16-Bit EMIF for Uncached CPU Data Writes (continued)

Memory/Core	Byte	Halfword	Word	Double
M4	Not OK	OK	OK	OK

Apart from above, even unaligned writes should be avoided. Unaligned writes can be produced by compiler even if datatypes corresponding to word or double is used. For details, see Section 17.4.2.

17.4.1.2 Cached CPU Access of EMIF

For cached CPU access to EMIF, the accesses are typically cache line aligned. Hence, the restriction of RMW is not applicable. But there are some points that need to be considered for below cores:

- A15:

  Cortex A15 (ARMV7-A arch) cache architecture is complicated and has multiple features to support performance and cache coherency.

  If the cache is enabled (SCTLR.C set) and the memory page is configured for write-back, then read-write-allocate (WB-RWA) is compatible with our 32-bit (or 16-bit) ECC quanta. Further the write streaming (ACTLR bit 24) enhancement, bypassing write allocate, is also compatible with our 32-bit (or 16-bit) ECC.

  So from software point of view, if user is using “normal” memory type with cache enabled and set in Write back , read write allocate, A15 by design should not produce any sub-quanta writes

  Other modes of cache can produce sub-quanta writes. They are summarized as below.

  – If the cache is set as write though mode, this can produce sub-quanta writes.
  – If MMU setting is strongly ordered or device type, this disables cache and it can produce sub-quanta writes

- EVE:

  Eve does not have data cache, it only has program cache. So the data access of EVE to the EMIF will totally depend on the C datatype being used. If the C code does a byte or a half word access, this violates the alignment constraint and ECC can be corrupted

17.4.1.3 Non CPU Access of EMIF Memory

DMA/Non CPU access to EMIF if not ECC quanta aligned can cause incorrect ECC computation. To avoid this, the recommendation below should be followed:

- DMA start pointer should be 32bit aligned
- DMA size should be 32 bit aligned
  There are multiple masters in TDA2x SoC that can issue DMA to EMIF. A non-comprehensive list is given below:
  – Masters where alignment can be made possible by system design software control:
    - EDMA (DSP, EVE, SYSTEM)
    - SDMA
    - VIP
    - VPE
    - DSS
  – Masters where alignment CANNOT be made possible by system design software control:
    - GPU/BB2D
    - MMC
    - GMAC
    - USB

For many of the above interface, it might not be possible to program the descriptor to always produce aligned writes. This is because the DMA size is determined by the incoming packet length or pixel location in frame that is not in user control.

17.4.1.4 Debugger Access of EMIF via the Memory Browser/Watch Window

Debugger access of EMIF is mostly read. In case any write is done via debugger watch window or memory browser, care should be taken similar to what is described in Section 17.4.1.1. Note that any debug access does not go via the CPU normal path but via the debug path and the EMIF content may not correlate with what CPU sees due to cache.

17.4.1.5 Software Breakpoints While Debugging

Software breakpoints are mostly supported by temporarily modifying the instruction and replacing the opcode with a breakpoint opcode. Since the opcode are like 8 bit in many cases, it becomes unaligned with ECC sub-quanta and can cause ECC errors while the breakpoint executes. Table 85 summarizes the CPU and the impact for software breakpoints.

Table 85. ECC Correctness for Software Breakpoints on ECC Enabled Regions

Memory/Core	Software Breakpoint
A15	Not OK
DSP	Not OK
EVE	Not OK
M4	Not OK

To overcome the above situations, the below approach is recommended:

- Keep ECC disabled in DEBUG mode
- Limit breakpoints to only HW breakpoints
- Keep the code section in ECC disabled region

17.4.2 Compiler Optimization

Compiler can optimize a C code to produce unaligned access even if the native datatype used in of word size.

Example:

```
void
sub (int *a, int *b)
{
*b = (0xFFFF00FF & *b) | (*a & 0x0000FF00);
}
```

Produces the code below:

```
ldrb r3, [r0, #1] @ zero_extendqisi2
strb r3, [r1, #1]
```

To ensure that there is no byte or halfword access in the code, the below steps are recommended:

1. Screen the generated object code to check for any such byte/halfword store instruction
   (a) Example: For ARM Disassemble the final executable (<filename>.out) file with armdis and search the output for STRB or STRH. The disassembler is part of the compiler release distribution.
2. Rewrite the C code to eliminate the usage of byte/halfword store instruction.
3. volatile Keyword will direct the compiler to use only 32-bit aligned access.

17.4.3 Restrictions Due to i882 Errata

As per the device specification of TDAx, EMIF had a mechanism of detecting illegal ECC sub-quanta writes to EMIF and though an error response over the internal bus and raise and interrupt. Due to the errata, this feature is not available.

For details, see i882 erratas: *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Silicon Errata* (SPRZ397) and *TDA2Ex SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 1.0 Silicon Errata* (SPRZ428).

The current workaround for , TDA2e and DRA72x SR1.0 is shown below:

1. Disable ECC, or
2. Enable ECC for desired ranges in EMIF1, and ensure that all DDR write accesses to EMIF1 (including ECC protected or unprotected ranges) from all initiators are a multiple of quanta size and are quanta aligned.

This bug is fixed in TDA2ex PG2.0 and in DRA72x with the restrictions shown in Table 86.

Table 86. Impact of EMIF ECC Errata i882 Hardware Fix

Parameter	Software Meaning	Before Fix	After Fix
EMIF Error Response	Error response from EMIF enables CPUs to give aborts upon sub-quanta writes	No	Yes
ERR_SYS and WR_ECC_ERR_SYS are set in EMIF_SYSTEM_OCP_INTERRUPT_STATUS	Sub-quanta write related error response in EMIF error registers	Yes	Yes
ECC errors is generated	ECC mismatch error in case of sub-quanta writes	Only reads generate errors	Only reads generate errors
ConnID reported in EMIF_OCP_ERROR_LOG register	Information about which master caused the Sub-quanta error	Yes	Yes
EMIF interrupt generated and Interrupt checked on M4 via crossbar id 105	Ability to route the EMIF error to CPU cores	Yes	Yes

After the fix, the feature of EMIF to send error response to a sub-quanta write over the internal bus to the respective master will be depreciated. But other methods like interrupts and error flags being set will be there and can be effectively used to find such illegal sub-quanta writes.

17.4.4 How to Find Who Caused the Unaligned Quanta Writes After the Interrupt

Assuming that the TDA2xx PG2.0 device is being used, an interrupt will be received from EMIF to a configured core where the interrupt is routed (via CROSSBAR). Note that there is no EMIF address log registered for this sub-quanta write.

1. Determine the reason of the interrupt by checking if WR_ECC_ERR_SYS is set in the EMIF_SYSTEM_OCP_INTERRUPT_STATUS register and verify it is because of sub-quanta write.
2. Check the MCONID in the EMIF_OCP_ERROR_LOG and establish the source of the error. The ConnID definition is present in the *EMIF_OCP_ERROR_LOG* table in the *Interconnect* chapter of the *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5).
3. The CONNID can help find the source of unaligned writes
4. Once the source is identified, use OCP watchpoint/MA watchpoint to check for sub-quanta writes
5. Other inaccurate way if the source of error is CPU is as below:
   (a) Route the EMIF interrupt to the same core. The call trace can give the locality of the code, which is triggered by the sub-quanta write.
   (b) For CPU causing such sub-quanta writes, one can run a check on the object/disassembly file to check for a store halfword or byte instruction.

17.5 Impact of ECC on Performance

Enabling ECC on the EMIF and OCMC has minimal impact on the throughput. Table 87 through Table 89 show the impact of ECC for different EDMA transfers.

Table 87. System EDMA Operation Throughput

System EDMA Operation	Throughput (MB/s)	
	Without ECC	Without ECC
OCMC → OCMC	4006.29	3996.68
DDR → DDR	3143.36	3141.14
DDR → OCMC	3436.32	3424.55
OCMC → DDR	3501.29	3496.4

Table 88. DSP EDMA Operation Throughput

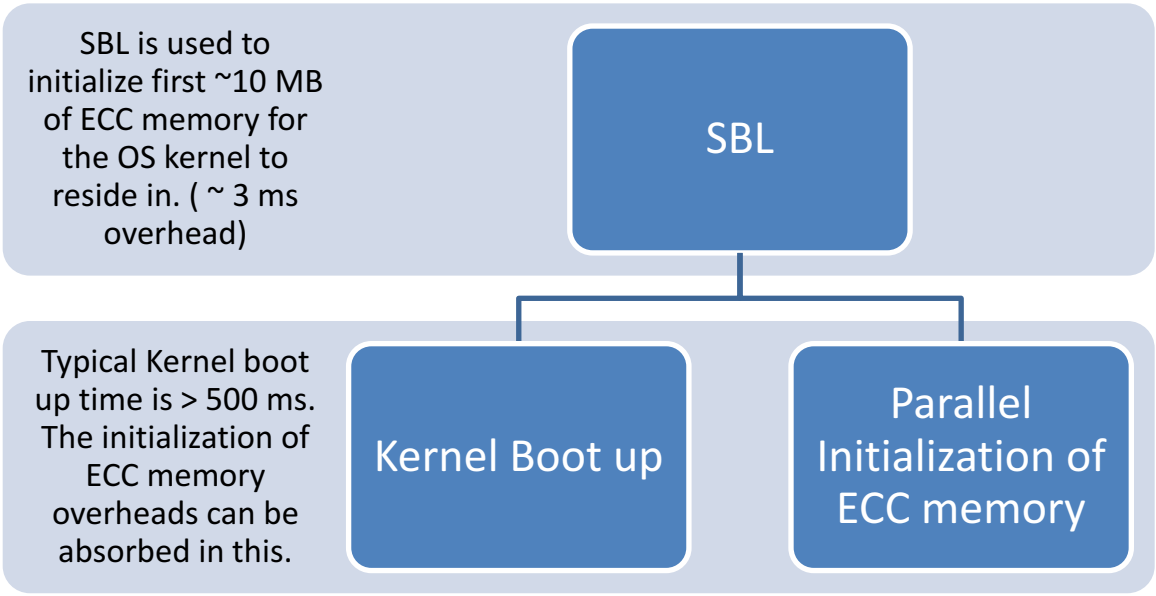
DSP EDMA Operation/Throughput (MBps)	Throughput (MB/s)	
	Without ECC	With ECC
OCMC → OCMC	4011.109	3996.681
DDR → DDR	2843.383	2843.838
DDR → OCMC	2658.079	2653.847
OCMC → DDR	2662.326	2656.667

Table 89. EVE EDMA Operation Throughput

EVE EDMA Operation/Throughput (MBps)	Throughput (MB/s)	
	Without ECC	With ECC
OCMC → OCMC	4011.109	4011.109
DDR → DDR	2845.051	2844.444
DDR → OCMC	3144.558	3150.502
OCMC → DDR	3150.502	3156.469

The impact of ECC initialization time depends on the amount of space the software decides to make ECC protected. Typically, CPU code and essential data structures are maintained in ECC enabled regions.

For example, if 256 MB of memory is used in ECC enabled region and the EDMA is used to initialize the memory the time taken to initialize memory given a system EDMA throughput of 3.14 GBps is ~82 ms. This time required can be hidden by a stage wise memory initialization to hide the initialization time.



18 DDR3 Interleaved vs Non-Interleaved

NOTE: This section is not valid for TDA2ex.

18.1 Interleaved versus Non-Interleaved Setup

TDA2xx supports two external memory controllers (EMIF). These EMIFs can be configured in interleaved and non-interleaved modes. In the non-interleaved mode of EMIF operation, the internal banks of 32-bit SDRAM can be accessed. In interleaved mode, the internal banks of two 32-bit SDRAMs can be accessed. Interleaving is configured to occur at 128-byte, 256-byte, or 512-byte granularity. For example, if 128-byte granularity interleaving is set it means the first 128-bytes are from EMIF1, the second 128-bytes are from EMIF2, the third 128-bytes are from EMIF1, and so on. Interleaving is controlled by the DMM_LISA_MAP registers, shown in Table 90.

Table 90. DMM_LISA_MAP_x

Bits	Field Name	Description	Type	Reset
31:24	SYS_ADDR	DMM system section address MSB.	R/W	0x0
23	RESERVED	Reserved	R	0x0
22:20	SYS_SIZE	DMM system section size. 0: 16-MiB section 1: 32-MiB section 2: 64-MiB section 3: 128-MiB section 4: 256-MiB section 5: 512-MiB section 6: 1-GiB section 7: 2-GiB section	R/W	0x0
19:18	SDRC_INTL	SDRAM controller interleaving mode. 0: No interleaving 1: 128-byte interleaving 2: 256-byte interleaving 3: 512-byte interleaving	R/W	0x0

Table 90. DMM_LISA_MAP_x (continued)

Bits	Field Name	Description	Type	Reset
17:16	SDRC_ADDRSPC	SDRAM controller address space	R/W	0x0
15:10	RESERVED	Reserved	R	0x0
9:8	SDRC_MAP	SDRAM controller mapping. 0: Unmapped 1: Mapped on SDRC 0 only (not interleaved). 2: Mapped on SDRC 1 only (not interleaved). 3: Mapped on SDRC 0 and SDRC 1 (interleaved). If this setting is used, SYS_SIZE must at least be equal to 32-MiB.	R/W	0x0
7:0	SDRC_ADDR	SDRAM controller address MSB.	R/W	0x0

- The configuration used for non-interleaved DDR3 is as follows: (512-MB, Non-Interleaved) DDR3 running at 532 MHz.

  Note that the value of the LISA map should be the same between the MA LISA map and the DMM Lisa Map.

```
//MA_LISA_MAP_i
WR_MEM_32(0x482AF040, 0x80500100);
WR_MEM_32(0x482AF044, 0xA0500200);
//DMM_LISA_MAP_i
WR_MEM_32(0x4E000040, 0x80500100);
WR_MEM_32(0x4E000044, 0xA0500200);
```

- The configuration used for interleaved DDR3 is as follows: (1-GB, 128-Byte Interleaved) DDR3 running at 532 MHz.

  Note that the value of the LISA map should be the same between the MA LISA map and the DMM Lisa Map.

```
//MA_LISA_MAP_i
WR_MEM_32(0x482AF040, 0x80640300);
WR_MEM_32(0x482AF044, 0x00000000);
//DMM_LISA_MAP_i
WR_MEM_32(0x4E000040, 0x80640300);
WR_MEM_32(0x4E000044, 0x00000000);
```

18.2 Impact of Interleaved vs Non-Interleaved DDR3 for a Single Initiator

The following bar graphs show the impact of the improvement in bandwidth of single transfer controller System EDMA, DSP EDMA, and EVE EDMA for interleaved EMIF versus non-interleaved EMIF. The interleaving leads to lesser DDR page opens and closes, which leads to improvement of the individual EMIF performances. Because two EMIFs are used for any data transfer greater than 128 Bytes, the total bandwidth available for the initiators also increases significantly. Note that in the following data (for the interleaved case, especially), the total utilization of the available EMIF bandwidth is less as the system is now initiator bottlenecked; that is, the initiator (EDMA TC) requests do not generate the fill of the EMIF command FIFOs.

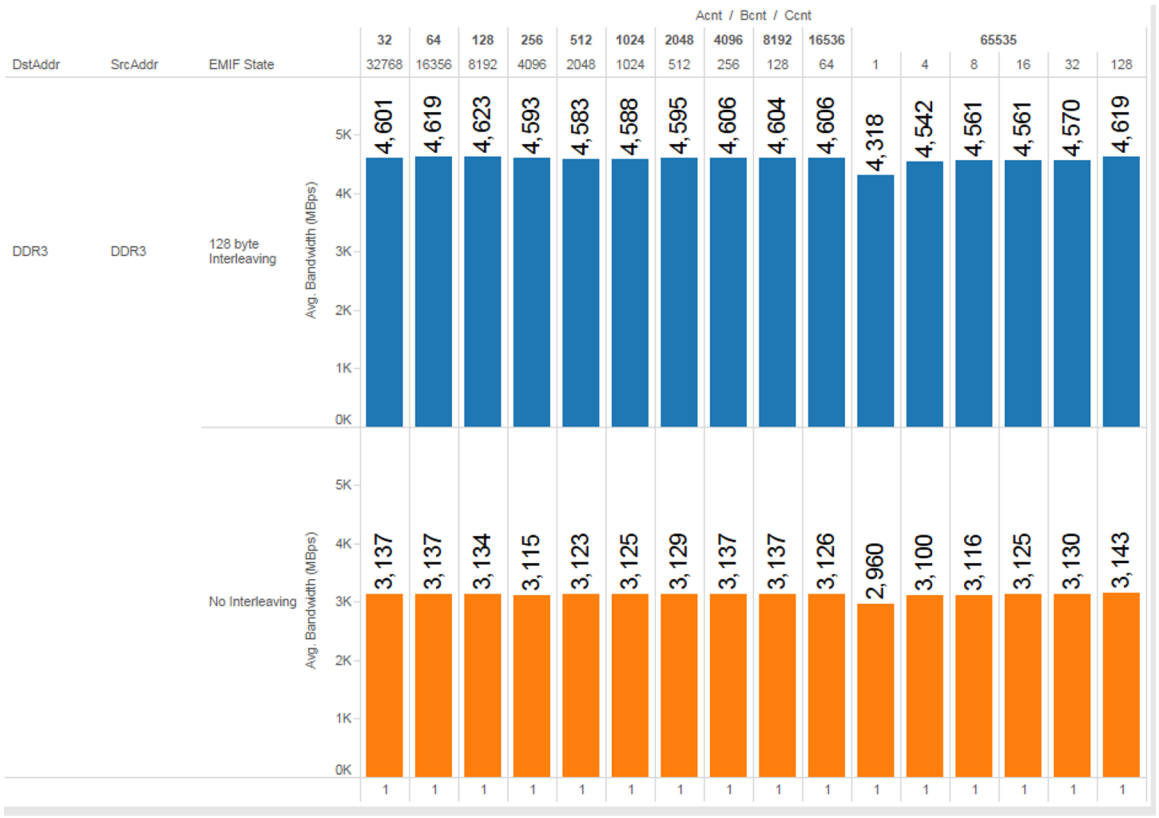


Figure 51. System EDMA Single Transfer Controller

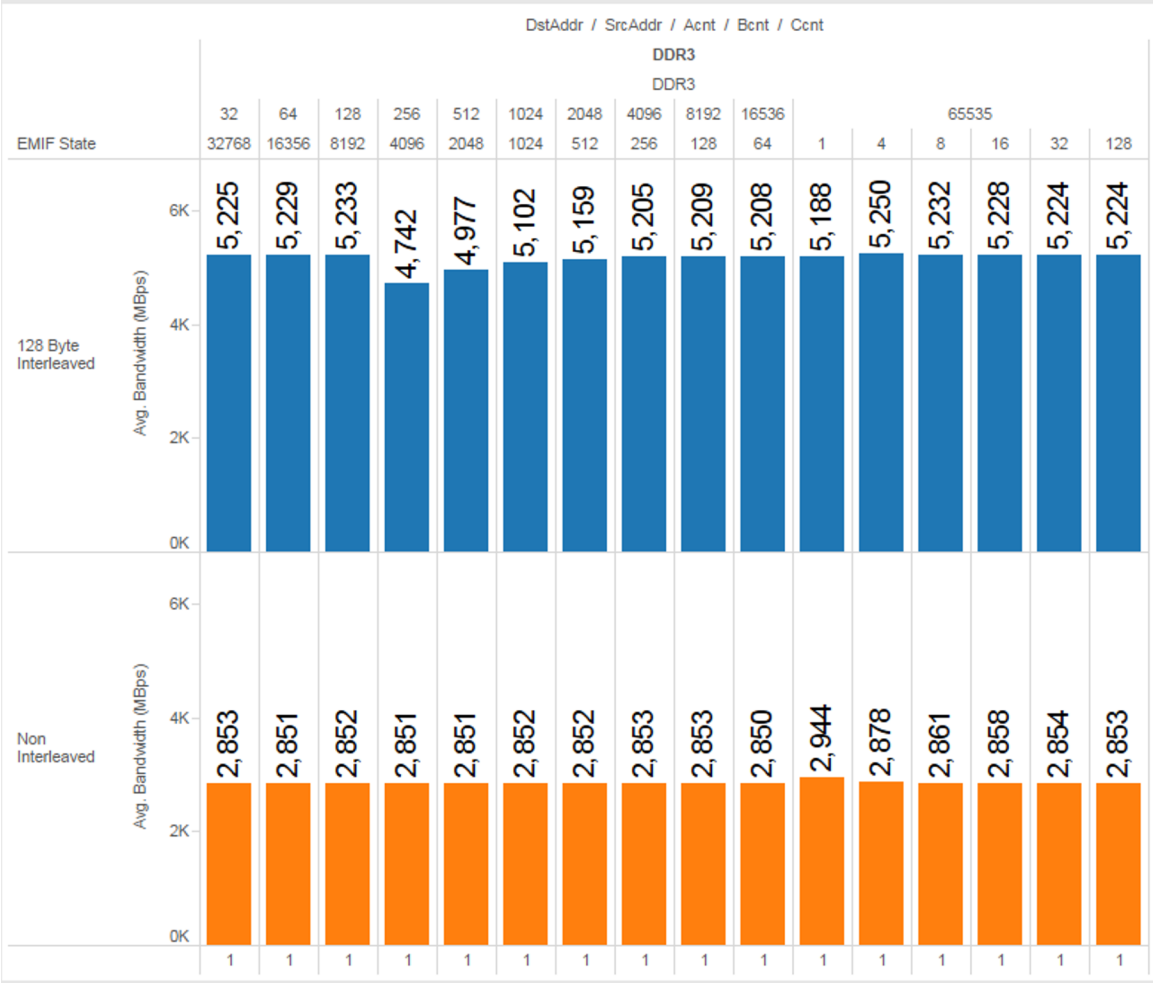


Figure 52. DSP EDMA Single Transfer Controller

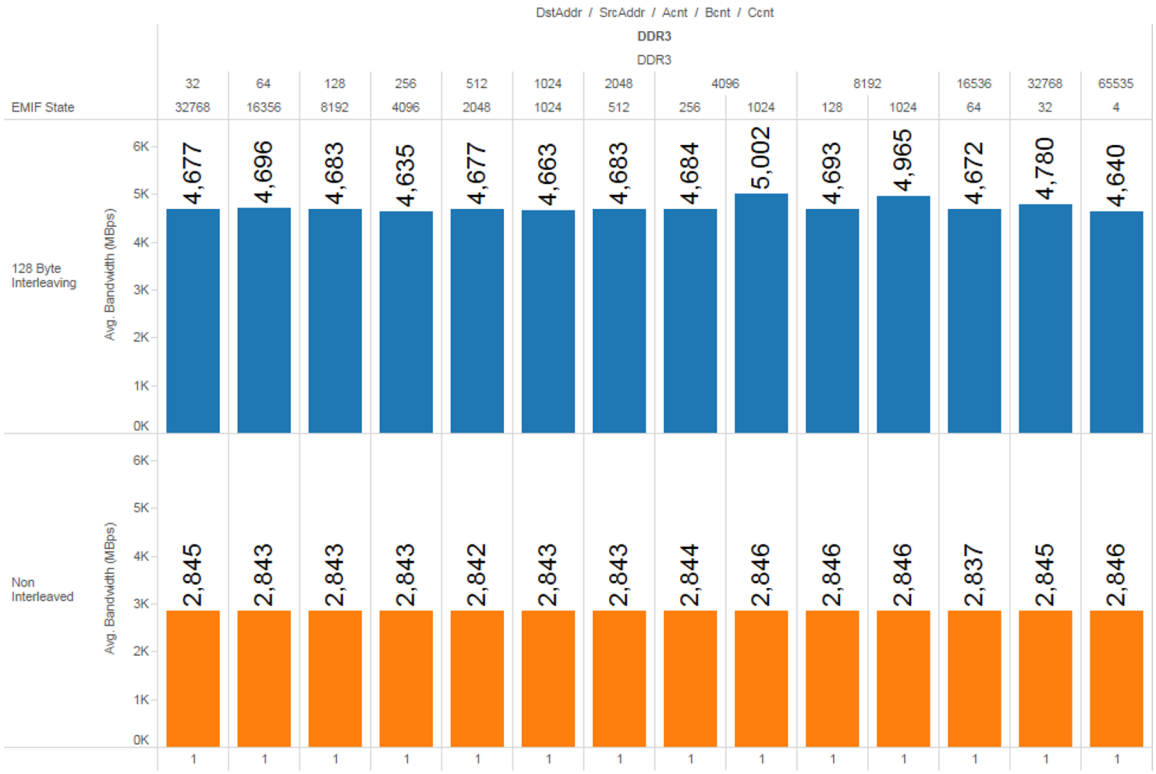


Figure 53. EVE EDMA Single Transfer Controller

18.3 Impact of Interleaved vs Non-Interleaved DDR3 for Multiple Initiators

To understand the total available DDR bandwidth of the system for the interleaved versus non-interleaved mode of operation, it is required to have a multi-initiator test. In this synthetic test, the following initiators are used for the interleaved versus non-interleaved case. The DSS reporting an Underflow error was used as a reference point to know when the DDR bandwidth is maximized out in the system.

NOTE: In the current experiment for the non-interleaved case, the data is not routed through EMIF2 and the system is assumed to be a single, 32-bit DDR system.

Non-Interleaved initiators:

- IVAHD: Executing 1080p60 decode
- BB2D: Executing 1280x720 NV12 3 frame overlay
- DSP1 EDMA single TC
- DSS 3 VID + 1 GFX pipe: 2 VID pipes 720p ARGB8888, 1VID + 1 GFX RGB888

Interleaved initiators:

- IVAHD: Executing 1080p60 decode
- BB2D: Executing 1280x720 NV12 3 frame overlay
- DSP1 EDMA single TC
- DSS 3 VID + 1 GFX pipe: 2 VID pipes 720p ARGB8888, 1VID + 1 GFX RGB888
- DSP2 EDMA two TC
- System EDMA two TC

The throughput is measured using statistic collectors available in the L3 at the EMIF1_probe and EMIF2_probe as shown in Figure 54. The statistic collector registers are read every 100 µs for 5 seconds and the bandwidth data is obtained by dividing the number of bytes transferred in 100 µs divided by 100 µs.

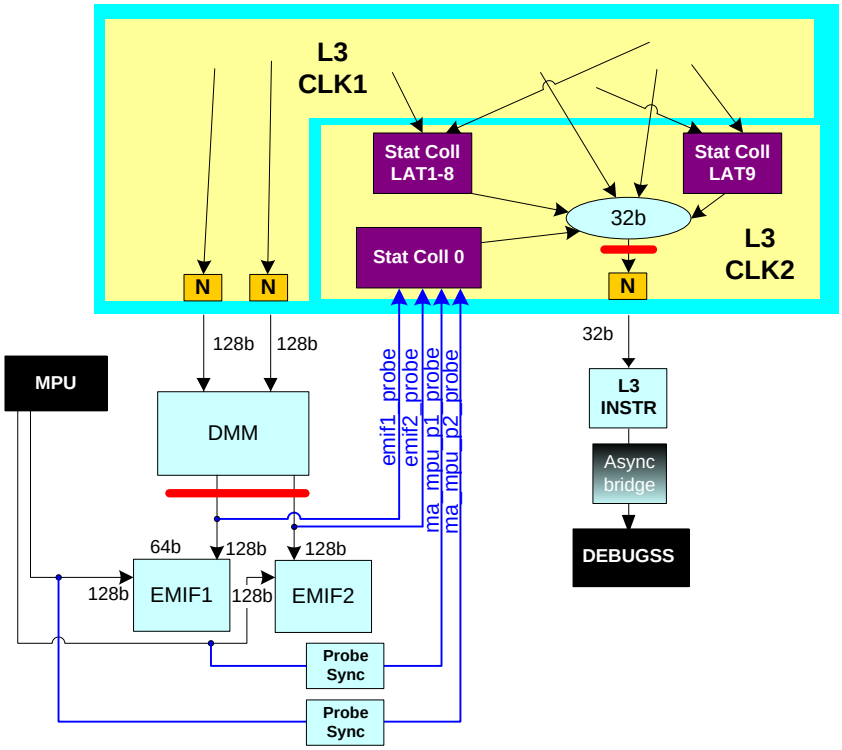


Figure 54. L3 STATCOLL EMIF1 and EMIF2 PROBE Mechanism

The ideal throughput for DDR3 running at 532 MHz is:

- For the non-Interleaved case, 532 MHz × 4 bytes × 2 = 4256 MB/s
- For the interleaved case, (Non-Interleaved value × 2) = 8512 MB/s

Table 91 shows that when the EMIF FIFOs are fully occupied then the system can expect a 10-15% boost in performance by configuring the EMIFs in interleaved mode versus a non-interleaved mode.

Also note that this is a synthetic test to measure the maximized out EMIF throughput. When executing a real system use-case, you can expect around 55-60% DDR utilization.

Table 91. Impact of Interleaved vs Non-Interleaved DDR3 for Multiple Initiators

	Non-Interleaved (One 32-Bit Memory)			Interleaved (Two 32-Bit Memory)		
	EMIF1 Bandwidth	EMIF2 Bandwidth (No Traffic to EMIF2)	Total	EMIF1 Bandwidth	EMIF2 Bandwidth	Total
Maximum Bandwidth (MB/s)	3121.60	NA	3121.60	3650.88	3642.88	7293.76
Average Bandwidth (MB/s)	2458.25	NA	2458.25	3035.51	3045.00	6080.51
Maximum DDR Utilization	73%	NA	73%	86%	86%	86%
Average DDR Utilization	58%	NA	58%	71%	72%	71%

19 DDR3 vs DDR2 Performance

The EMIF module provides connectivity between DDR2 or DDR3 types of memories and manages data bus read/write accesses between external memories and device subsystems which have master access to the L3_MAIN interconnect and DMA capability. EMIF supports JEDEC standard-compliant DDR2-SDRAM and DDR3-SDRAM memory types.

Supported CAS latencies are:

- DDR3: 5, 6, 7, 8, 9, 10, and 11
- DDR2: 2, 3, 4, 5, 6, and 7

This section compares the performance of DDR2 and DDR3 for a single initiator (System EDMA) and multiple initiators that try to simulate an actual use-case scenario.

The following configurations were used to test the DDR2 performance versus the DDR3 performance:

- DDR3 Configuration:
  - MT41K128M16: 16 Meg × 16 × 8 banks DDR part
  - Page size is 1024-cells. This makes the effective page size = 1024 × 16 bits × 2 = 32768 bits = 4096 Bytes = 4KB.
  - Number of Banks = 8
  - CAS Latency: 6 cycles
  - DPLL Frequency set to 532 MHz and 400 MHz
- DDR2 Configuration:
  - Page size is 1024-cells. This makes the effective page size = 1024 × 16 bits × 2 = 32768 bits = 4096 Bytes = 4KB.
  - Number of Banks = 8
  - CAS Latency: 6 cycles
  - DPLL Frequency set to 400 MHz

19.1 Impact of DDR2 vs DDR3 for a Single Initiator

Figure 55 shows the impact on the bandwidth of single transfer controller System EDMA while using DDR2 at 400 MHz, DDR3 at 532 MHz, and DDR3 at 400 MHz. As can be seen, the DDR2 gives better efficiency than DDR3 at the same frequency of operation. This is mainly because the CAS latency of DDR2 is lower than the CAS latency of DDR3.

DDR	CAS Write Latency (Cycles)	CAS Latency (Cycles)
DDR3 at 532 MHz	6	7
DDR3 at 400 MHz	6	7
DDR2 at 400 MHz	NA	6

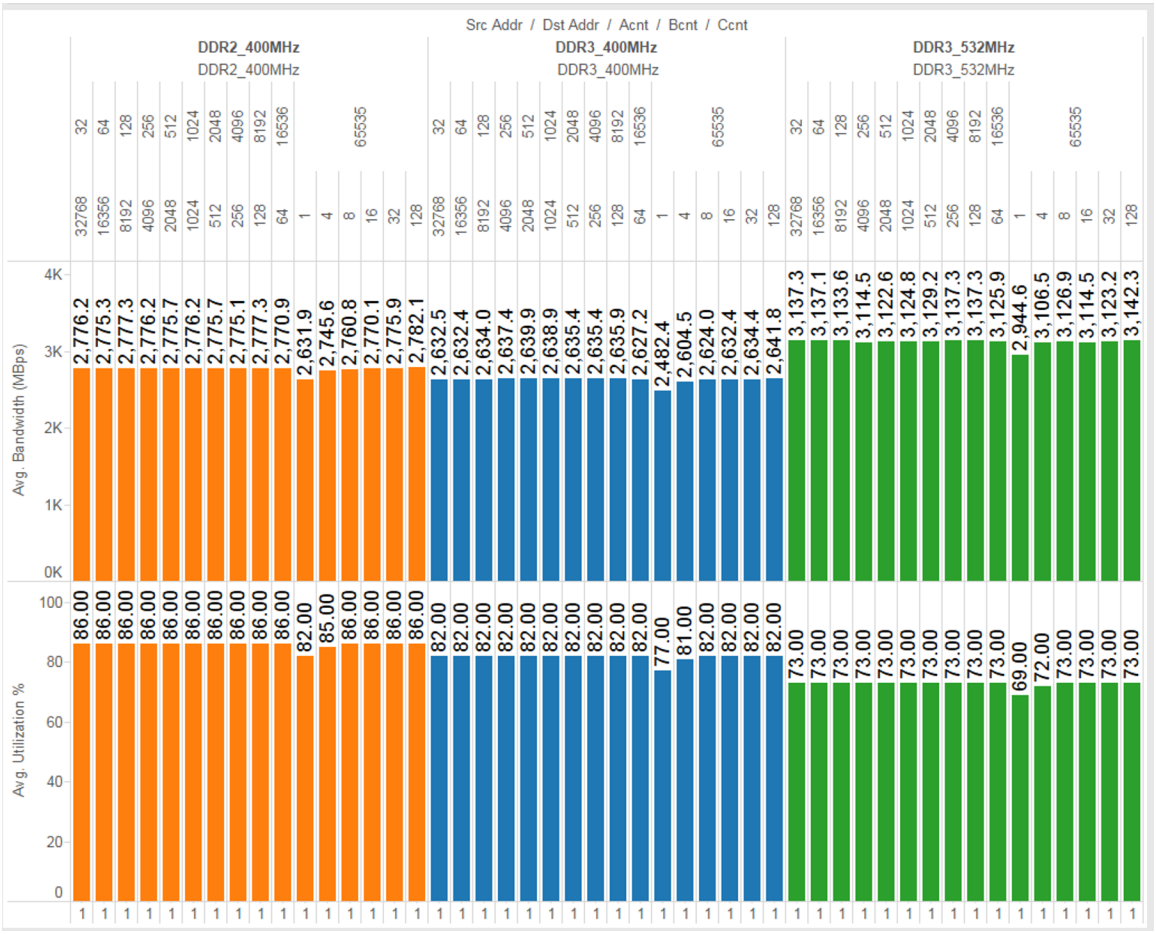


Figure 55. DDR2 versus DDR3 Performance and Efficiency for Single Initiator

19.2 Impact of DDR2 vs DDR3 for Multiple Initiators

To understand the total available DDR bandwidth of the system DDR2 versus DDR3, it is required to have a multi-initiator test. In this synthetic test, the following initiators are used for DDR2 and DDR3. The DSS reporting an Underflow error was used as a reference point to know when the DDR bandwidth is maximized out in the system.

NOTE: In the current experiment, the EMIF is configured in non-interleaved and the system is assumed to be a single 32-bit DDR system.

Non-Interleaved initiators:

- IVAHD: Executing 1080p60 decode
- BB2D: Executing 1280x720 NV12 3 frame overlay
- DSP1 EDMA single TC
- DSS 3 VID + 1 GFX pipe: 2 VID pipes 720p ARGB8888, 1VID + 1 GFX RGB888

Table 92. Impact of DDR3 versus DDR2 for Multiple Initiators

	EMIF1 Bandwidth		
	DDR3 Non-Interleaved (532 MHz)	DDR2 Non-Interleaved (400 MHz)	DDR3 Non-Interleaved (400 MHz)
Maximum Bandwidth (MB/s)	3121.60	2748.97	2288.32
Average Bandwidth (MB/s)	2458.25	2566.12	1734.47
Maximum DDR Utilization	73%	86%	72%
Average DDR Utilization	58%	80%	54%

20 Boot Time Profile

Boot-time, that is, the time taken by the system to show its "availability" since the power button was pushed on, is becoming a key differentiator in the usability vector.

The definition of availability varies across the devices. For example:

- Appearance of home screen for devices containing a display, for example, cellphone or media player
- An audible tone / LED turning on or changing color for devices without display
- Appearance of shell prompt on development systems with console

In the current context, Boot time refers to time taken until the appearance of a Linux shell prompt.

Complete Boot time can be measured in two stages:

1. ROM Boot time profile
2. System Boot time profile

20.1 ROM Boot Time Profile

The time taken from ROM until booting the initial software (ISW) can be measured with probing signals. A GPIO bit is toggled high in the initial software. The time difference between the PORZ signal and GPIO signal is measured to get the boot time. This time measured includes ROM Boot time and the ISW loading time. This procedure is repeated for different ISW image sizes.

NOTE: In case of QSPI, ROM copies the binary to SRAM and the ISW runs from SRAM. The loading time varies based on the ISW image size.

Table 93. ROM Boot Time With Different Image Sizes

	Image Size				
	13.1 KB	**64 KB**	**128 KB**	**256 KB**	**480 KB**
QSPI1	24 ms	63 ms	111 ms	207 ms	374 ms
QSPI4	16 ms	20 ms	25 ms	36 ms	55 ms

20.2 System Boot Time Profile

System Boot time process consists of operations such as authentication, hashing, loading the binaries, and so on. Execution time can be measured by placing timestamps at the start and end of these operations. Auditing the timestamps helps in calculating actual time spent in processing. This procedure is repeated for both GP and HS samples. The HS boot flow has a few extra steps of authentication over GP flow.

The stages involved in booting are:

- ROM Code
- Public Protected Application (PPA) [Only in HS boot flow]
- Authenticate Initial Software (ISW)
  - Hash Initial Software (ISW)
- 1st Initial Software (ISW)
  - Load U-Boot
  - Authenticate U-Boot [Only in HS boot flow]
    - Hash U-Boot [Only in HS boot flow]
- U-Boot
  - Load DTB
  - Authenticate DTB [Only in HS boot flow]
    - Hash DTB [Only in HS boot flow]
  - Load Kernel
  - Authenticate Kernel [Only in HS boot flow]
    - Hash Kernel [Only in HS boot flow]
- Kernel

NOTE: In production boot flow, U-Boot is a single stage bootloader.

The boot time can vary based on the type of peripheral or memory device set by the SYSBOOT configuration. The values measured here for QSPI.

The results were captured using the following binaries:

- PPA: 11.6 Kb
- ISW: 59.8 Kb
- U-Boot: 251.23 Kb
- Device Tree (DTB): 37.79 Kb
- Kernel: 4.81 Mb

Table 94. System Boot Time Profile for GP and HS Samples for SYSBOOT Production

Operation	QSPI4				QSPI1			
	GP		HS		GP		HS	
	Time (ms)	Elapsed (ms)	Time (ms)	Elapsed (ms)	Time (ms)	Elapsed (ms)	Time (ms)	Elapsed (ms)
PPA init			21.53	9.05			31.82	9.40
Authenticate ISW cert			39.01	2.72			89.81	2.69
Hash ISW			39.04	2.69			89.81	2.68
1st ISW	19.84	434.82	41.61	536.29	20.17	434.84	92.38	536.32
Load Kernel	157.68	290.14	179.58	290.18	158.02	290.1	230.32	290.14
Load DTB	450.66	3.8	472.55	3.79	450.99	3.8	523.30	3.80
Authenticate DTB			481.80	1.59			526.58	1.59
Hash DTB			481.80	0.70			532.56	0.71
Authenticate Kernel			491.56	92.88			536.35	92.98
Hash Kernel			491.58	92.08			537.85	92.14
Start Kernel	804.72	–	923.00	–	805.13	–	973.80	–

Table 95. System Boot Time Profile for GP and HS Samples for SYSBOOT Development

Operation	QSPI4				QSPI1			
	GP		HS		GP		HS	
	Time (ms)	Elapsed (ms)	Time (ms)	Elapsed (ms)	Time (ms)	Elapsed (ms)	Time (ms)	Elapsed (ms)
PPA init			21.52	9.06			31.82	8.94
Authenticate ISW cert			38.84	2.69			89.26	2.70
Hash ISW			38.87	2.66			89.28	2.68
1st ISW	19.84	110.26	41.53	120.32	20.17	110.26	91.97	120.43
Load U-Boot	95.49	29.57	117.18	29.63	95.83	29.57	167.61	29.63
Authenticate U-Boot			151.85	5.58			202.45	5.58
Hash U-Boot			151.85	4.70			202.45	4.73
1st U-Boot	132.39	4848.88	164.15	4989.90	132.72	4850.54	214.67	4998.57
Load DTB	770.48	13.92	802.70	14.50	771.61	13.93	854.18	14.50
Load Kernel	790.71	889.55	823.48	890.22	792.16	889.54	874.97	890.23
Authenticate DTB			2621.06	1.59			2659.48	1.60
Hash DTB			2621.06	0.71			2659.49	0.71
Authenticate Kernel			2629.94	93.04			2668.39	92.96
Hash Kernel			2629.94	92.21			2668.40	92.24
Start Kernel	5331.59	–	5499.08	–	5333.7	–	5558.31	–

21 L3 Statistics Collector Programming Model

Following are APIs that are used to configure statistics collector, setup timer, and get statistics on regular interval of 100 µs.

```
Initialize Statcoll: statCollectorInit();

void statCollectorInit()
{
    gStatColState.stat0_filter_cnt = 0;
    gStatColState.stat1_filter_cnt = 0;
    gStatColState.stat2_filter_cnt = 0;
    gStatColState.stat3_filter_cnt = 0;
}
Enable Statcoll and Read Statcoll registers:
/** \brief statCollectorControl
 *     Description: API to enable statcoll. Same API can be used to read the
 *     statcoll register values as well.
 *      Inputs:
 *      inst_name : Statcoll Instance Name. eg: STATCOL_EMIF_SYS,
 *                  STATCOL_DSP1_MDMA etc. defined in STATCOL_ID enumeration.
 *      cur_stat_filter_cnt : This value is ignored when calling this function
 *                  to enable the statcoll. When trying to read the statcoll
 *                  this value is used to determine the filter number used.
 *      mode: Used to indicate whether the function is being called for reading
 *                  or enabling the statcoll as defined by :
 *                  #define ENABLE_MODE      0x0
 *                  #define READ_STATUS_MODE 0x1
 *      Return : In the enable mode the function returns the filter number
 *                  assigned. In the read mode the function * returns the value
 *                  read (BW/Latency etc) from the statcoll registers.
*/
UInt32 statCollectorControl(UInt32 inst_name, UInt32 cur_stat_filter_cnt, UInt32 mode)
{
  switch (inst_name)
  {
      case STATCOL_EMIF_SYS:     cur_base_address = stat_coll0_base_address;
      cur_event_mux_req = 0;
      cur_event_mux_resp = 1;
      if(mode == ENABLE_MODE) {gStatColState.stat0_filter_cnt = gStatColState.stat0_filter_cnt +
1;}
      if(mode == ENABLE_MODE) {cur_stat_filter_cnt = gStatColState.stat0_filter_cnt;}
      break;
      case <NEXT_STATCOLL> :
      ...
   }
   if(mode == ENABLE_MODE)
  {
   if ( cur_stat_filter_cnt > 4 )
   {
     printf("We have exhausted filters/counters.....\n");
   }
   // Global Enable Stat Collector
   wr_stat_reg(cur_base_address+0x8,0x1);

   // Soft Enable Stat Collector
   wr_stat_reg(cur_base_address+0xC,0x1);

   wr_stat_reg(cur_base_address+0x18,0x5);
   // Operation of Stat Collector / RespEvt => Packet
   wr_stat_reg(cur_base_address+0x1C,0x5);

   // Event Sel
   wr_stat_reg(cur_base_address+0x20+4*(cur_stat_filter_cnt-1),cur_event_mux_req);

   // Op is EventInfo
```

```
    wr_stat_reg(cur_base_address+0x1FC+(0x158*(cur_stat_filter_cnt-1)),2);

    // Event Info Sel Op -> packet length
    wr_stat_reg(cur_base_address+0x1F8+(0x158*(cur_stat_filter_cnt-1)),0);

    // Filter Global Enable
    wr_stat_reg(cur_base_address+0xAC+(0x158*(cur_stat_filter_cnt-1)),0x1);

    // Filter Enable
    wr_stat_reg(cur_base_address+0xBC+(0x158*(cur_stat_filter_cnt-1)),0x1);

    // Manual dump
    wr_stat_reg(cur_base_address+0x54,0x1);
    // use send register to reset counters
   }
   else
   {
    wr_stat_reg(cur_base_address+0xC,0x0);
    cur_stat_filter_cnt = rd_stat_reg(cur_base_address+0x8C+((cur_stat_filter_cnt-1)*4));
    wr_stat_reg(cur_base_address+0xC,0x1);
   }
   return cur_stat_filter_cnt;
}
Usage (Dummy code):
void main()
{
   statCollectorInit();
   counterIdISSNTR1 = statCollectorControl(STATCOL_ISS_NRT1, 0, ENABLE_MODE);
   DMTIMER_prcmenable(TIMER_NUM);
   DMTIMER_Start(TIMER_NUM);

   // Dummy Read
   statCollectorControl(STATCOL_ISS_NRT1, counterIdISSNTR2, READ_STATUS_MODE);

   while(statCountIdx < TOTAL_COUNT)
   {
     while (DMTIMER_Read(TIMER_NUM) <= SYS_CLK_FREQ/10000) // for 100 us {;}
     statCountISSNRT1[statCountIdx++] = statCollectorControl(STATCOL_ISS_NRT1, counterIdISSNTR1,
READ_STATUS_MODE);
     DMTIMER_Stop(TIMER_NUM);
     DMTIMER_Start(TIMER_NUM);
   }
}
```

```
Timer APIs:
ReturnCode_t DMTIMER_Start (UWORD8 timer_num)
{
   ReturnCode_t checkReturn = RET_OK;

   /* Counter clear and auto reload enable */
   if (timer_num < 1 || timer_num > 16)
            return RET_FAIL;

   switch(timer_num) {
           case 1:
        /* Clear the counter value */
        WR_REG_32(TIMER1, DMTIMER__TCRR, 0x0);
        /* Triggering the timer load */
        WR_REG_32(TIMER1, DMTIMER__TTGR, 0x1);
        /* Start timer and reload enable: bit[0] start,
        bit[1] autoreload enable */
        WR_REG_32(TIMER1, DMTIMER__TCLR, 0x1);
        break;
           case 2:
        ...
           default:
        checkReturn = RET_FAIL;
        break;
   }
   return checkReturn;
}

ReturnCode_t DMTIMER_Stop(UWORD8 timer_num)
{
   ReturnCode_t checkReturn = RET_OK;

   /* Counter clear and auto reload enable */
   if (timer_num < 1 || timer_num > 16)
            return RET_FAIL;

   switch(timer_num) {
    case 1:
        /* Bit[0]: 0, counter is frozen */
        WR_REG_32(TIMER1, DMTIMER__TCLR, 0x0);
        break;
    case 2:
        ...
    default:
        checkReturn = RET_FAIL;
        break;
   }
   return checkReturn;
}
```

```
UWORD32 DMTIMER_Read(UWORD8 timer_num)
{
   volatile UWORD32 read_value = 0;

   if (timer_num < 1 || timer_num > 16)
            return RET_FAIL;

   switch(timer_num) {
    case 1:
        read_value = RD_REG_32(TIMER1, DMTIMER__TCRR);
        break;
    case 2:
        ...
    default:
        read_value = 0;
        break;
   }
   return read_value;
}
ReturnCode_t DMTIMER_prcmenable(UWORD8 timer_num)
{
   ReturnCode_t checkReturn = RET_OK;

   if (timer_num < 1 || timer_num > 16)
            return RET_FAIL;

   switch(timer_num) {
    case 1:
        checkReturn = (ReturnCode_t )prcm_enable_module(prcm_timer1);
        break;
    case 2:
        ...
    default:
        checkReturn = RET_FAIL;
        break;
    }
    return checkReturn;
}
```

22 Reference

- *TI Vision SDK, Optimized Vision Libraries for ADAS Systems* (SPRY260)
- *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x* (SPRZ397)
- *TDA2x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.x Technical Reference Manual* (SPRUHK5)
- *TDA2Ex SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.0 Technical Reference Manual* (SPRUI00)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265