Application Note

Understanding TDA4VM or DRA829 Memory for Optimal Performance

Karan Saxena, Kip Broadhurst, Junbok You, Richard Woodruff, and Ganesh Chelliah

ABSTRACT

The Jacinto 7 family of devices support multiple internal memories to achieve the best in class system latency, bandwidth, and functional safety requirements for a large number of automotive and industrial applications. The device-specific Data Sheet along with the Technical Reference Manual (TRM) can provide a detailed overview of the different kinds of memories in the device and their intended usage.

This application note is specific to TDA4VM and gives a brief overview of the different memory types and a typical use case for which it may be used along with the Software Development Kit (SDK) careabouts.

TDA4VM	
Data sheet	https://www.ti.com/lit/gpn/tda4vm
TRM	https://www.ti.com/lit/zip/spruil1
SDK	https://www.ti.com/tool/PROCESSOR-SDK-J721E

For details more than what is covered in this application note, see the *TDA4VM Jacinto™ Processors for ADAS and Autonomous Vehicles Sil Revs 1.0 and 1.1 Data Sheet* and *DRA829/TDA4VM Technical Reference Manual*. If further clarifications are required, post on the TI's Processor's E2E forum.

Table of Contents

Trademarks

Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All trademarks are the property of their respective owners.

1 Different Types of Memories on the TDA4VM

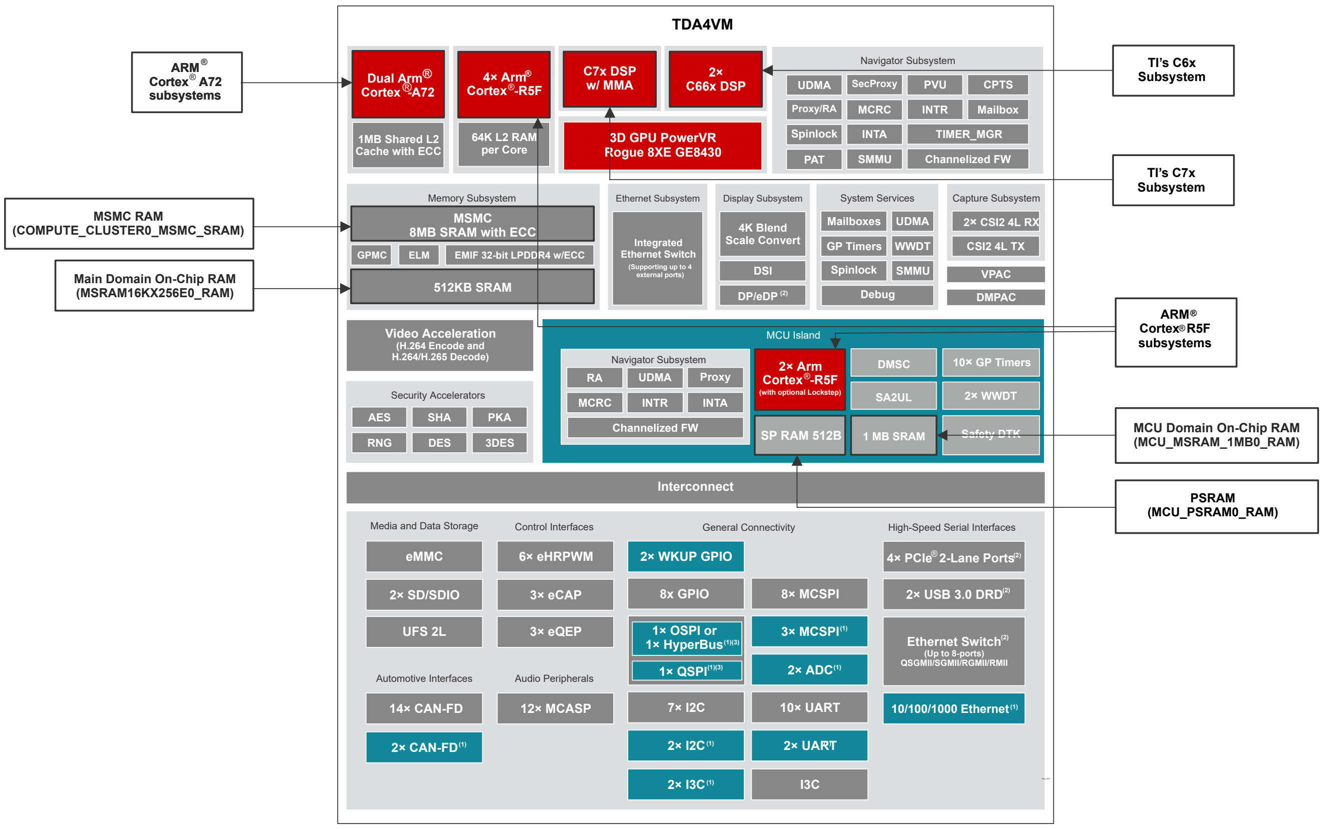

Figure 1-1. Device Block Diagram

1. The interface is located on the MCU Island but is available for the full system to access.
2. DP, SGMII, USB3.0, and PCIE[3:0] share a total of twelve SerDes lanes.
3. Two simultaneous flash interfaces configured as OSPI0 and OSPI1, or HyperBus™ and OSPI1.

This document covers the below memories and subsystems on the TDA4VM:

1. PSROM
2. PSRAM
3. MSMC RAM
4. MSRAM
5. Memories in Arm® Cortex® A72 subsystem
6. Memories in Arm Cortex R5F subsystem
7. Memories in TI's C6x Subsystem
8. Memories in TI's C7x Subsystem
9. DDR Subsystem

2 Memory Overview and Intended Usage

This section provides details on the different types of memories on the device along with their typical use cases. This information is key in designing the memory requirements for the end system.

2.1 PSROM

The PSROM module provides a memory mapped region that may be used for accessing ROM. The module has no registers, and maps the bus interface address, control, and read data signals to the address, control, and read data of the ROM. For more information, see the *Memory Map of the Device* chapter in *DRA829/TDA4VM Technical Reference Manual*.

2.1.1 Typical Use Cases

Usage of PSROM is for Boot ROM. The Boot ROM is a non-volatile memory that is used during the startup of the device to load a customer defined image such as a secondary boot loader. The device has two ROM codes operating in tandem (in TDA4VM): the MCU ROM code and the DMSC ROM code. The boot ROM code executes when the MCU processor is released from reset. This reset is generated from the DMSC subsystem. The ROM executes only on initial device power up or after wakeup from a low power mode.

2.2 PSRAM

The PSRAM block provides a memory mapped region that may be used by the instantiating system and other masters in the system to store various types of information. This region behaves as a standard byte-accessible static RAM. The PSRAM is a vbusp bus interface connected to a RAM. It has no control registers, and maps the bus interface address, control, and data signals to the address, control, and data of the RAM. The VBUSP protocol is a single-issue (i.e. pended), strongly ordered protocol that emphasizes simplicity over the highest possible performance.

MCU_PSRAM0_RAM, of size 512 Bytes, will survive any reset (MCU_PORz, PORz or Warm Reset) as long as the MCU domain is powered through the reset.

2.2.1 Typical Use Cases

- Scratchpad RAM
  - Used for storing debug and context information across chip resets
  - After a (warm/powered) reset, the events leading up to a reset can be analyzed and software can be debugged.

2.3 MSMC RAM

The Multicore Shared Memory Controller (MSMC) forms the heart of the compute cluster (COMPUTE_CLUSTER0) providing high-bandwidth resource access both to and from all of the connected processing elements and the rest of the system. MSMC serves as the data-movement backbone of the compute cluster.

The MSMC subsystem supports an on chip MSMC SRAM, which on TDA4VM is 8MB (4 banks x 2MB) SRAM with ECC. This memory is:

- Shared coherent level 2 / level 3 memory-mapped SRAM
- Shared coherent level 3 cache

The MSMC SRAM can be configured either as SRAM or as L3 Cache, or a combination of both. This memory can be used for performance improvement of a particular module.

To see how to split the MSMC as Cache and SRAM, see Section 4.4.2

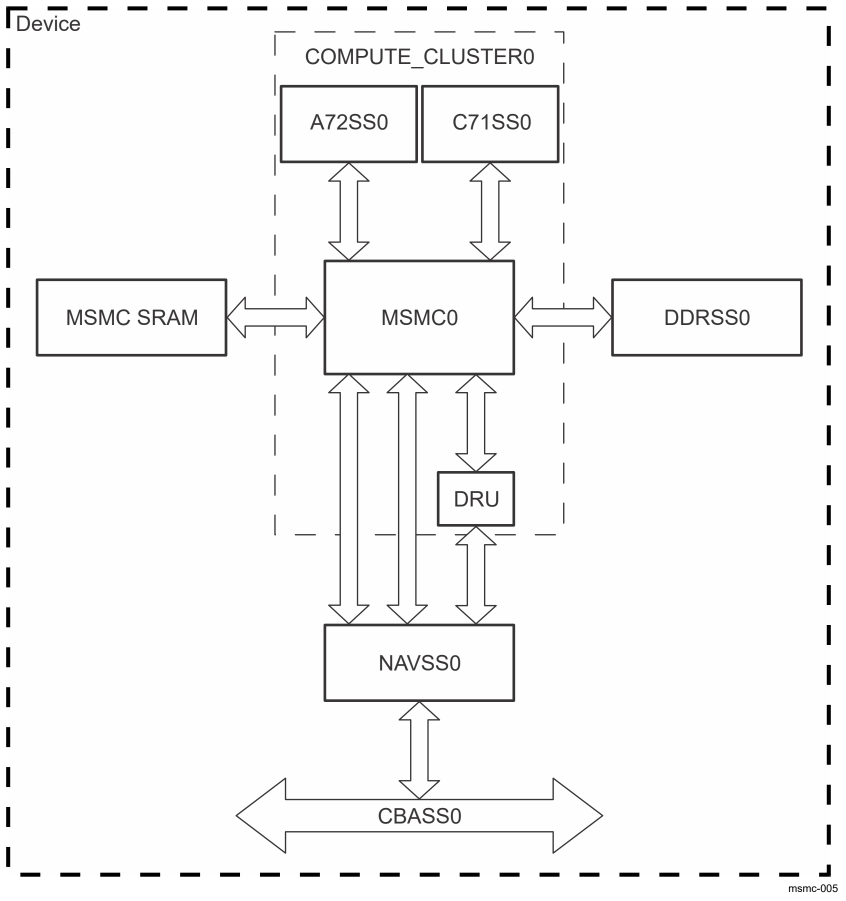

Figure 2-1. MSMC Overview

2.3.1 Typical Use Cases

- L3 Cache for Compute Cluster (A72 and C7x)
- SRAM C7x to use for running deep learning algorithms

2.3.2 Relevant Links

For more information, see the *Multicore Shared Memory Controller (MSMC)* chapter in the *DRA829/TDA4VM Technical Reference Manual*.

2.4 MSRAM

The MSRAM block provides a memory mapped region that may be used to store various types of information. It uses a higher transaction issue rate bus interface with full support for multiple outstanding transactions and simultaneous completions of read and write transactions. For more information, see the *Navigator Subsystem (NAVSS)* chapter of the device-specific TRM

There are two MSRAMs on the device: 512 KB sized in the Main Domain (*MSRAM16KX256E0_RAM*) and the other of size 1 MB in the MCU Domain (*MCU_MSRAM_1MB0_RAM*).

2.4.1 Typical Use Cases

- Main Domain On-Chip SRAM or Main OCRAM (*MSRAM16KX256E0_RAM*)
  - Provides additional on chip ECC protected SRAM to be used for any purpose
- MCU Domain On-Chip RAM or MCU OCRAM (*MCU_MSRAM_1MB0_RAM*)
  - Provides RAM for program/data storage
  - Local RAM accessible to R5F and DMSC/SMS and external masters (if enabled in their initiator firewalls)
  - On-chip RAM memory used by ROM code during boot and also for loading the booting image

2.5 ARM Cortex A72 Subsystem

The device implements one dual-core Arm Cortex-A72 MPU, which is integrated inside the Compute Cluster, along with other modules. The Cortex-A72 cores are general-purpose processors that can be used for running customer applications.

2.5.1 L1/L2 Cache Memory

- 48KB L1 Instruction Cache per processor with parity protection
- 32KB L1 Data Cache per processor with ECC protection
- 1MB Shared L2 Cache with ECC protection

2.5.2 L3 Memory

Optional L3 cache via MSMC available, see Section 2.3.

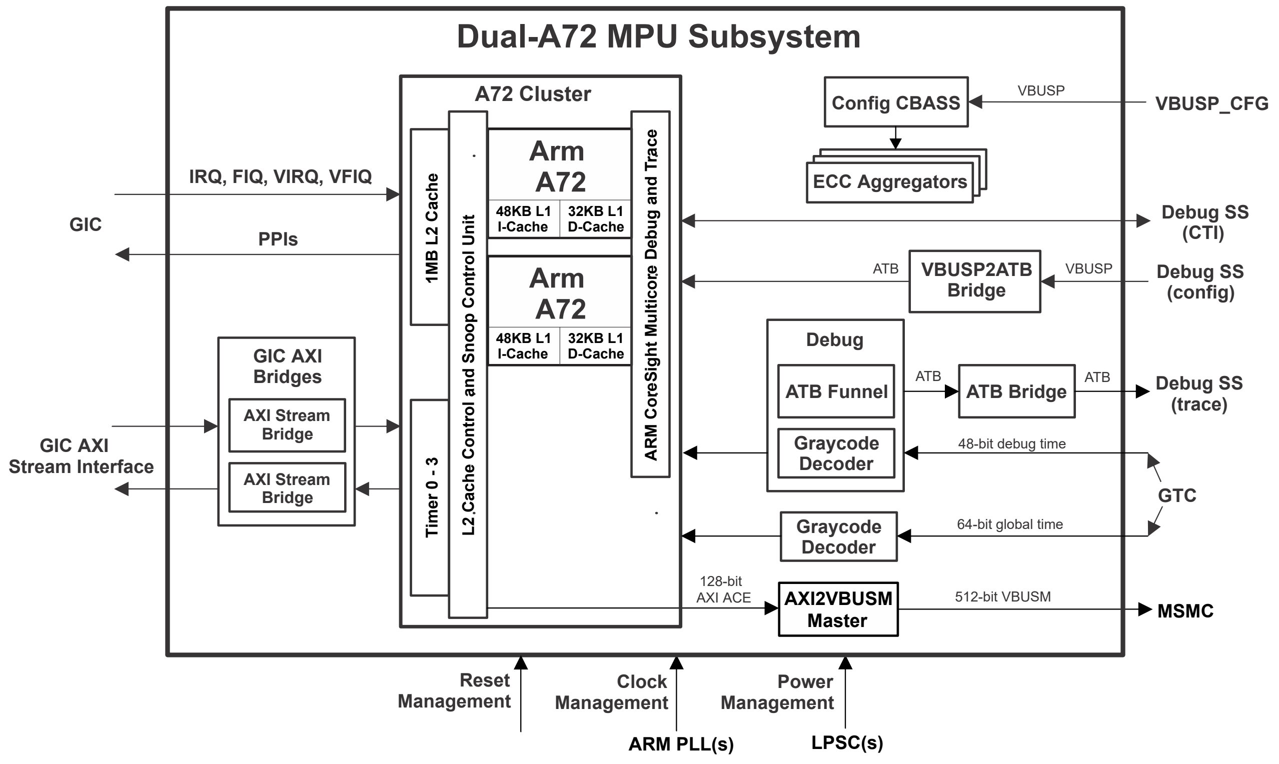

Figure 2-2. Dual-A72 MPU Subsystem

2.5.3 Relevant Links

- Arm Cortex-A72 Home Page: Cortex-A72 – Arm®
- Arm Cortex-A72 MPCore Processor Technical Reference Manual r0p3: https://developer.arm.com/documentation/100095/0003
- Arm Cortex-A Series Programmer's Guide for ARMv8-A: https://developer.arm.com/documentation/den0024/a/
- For more information, see the *Dual-A72 MPU Subsystem* chapter in the *DRA829/TDA4VM Technical Reference Manual*.

2.6 ARM Cortex R5F Subsystem

The R5FSS is a dual-core implementation of the Arm Cortex-R5F processor configured for split/lock operation. It also includes accompanying memories (L1 caches and tightly-coupled memories), standard Arm CoreSight™ debug and trace architecture, integrated vectored interrupt manager (VIM), ECC aggregators, and various other modules for protocol conversion and address translation for easy integration into the SoC.

2.6.1 L1 Memory System

Caches and Tightly Coupled Memory (TCM) are both L1 memories to R5F.

2.6.2 Cache

The R5F has a Harvard cache architecture, which means it has an independent instruction and data cache.

- 16KB instruction cache
  - 4x4KB ways
  - SECDED ECC protected per 64 bits
- 16KB data cache
  - 4x4KB ways
  - SECDED ECC protected per 32 bits

2.6.3 Tightly Coupled Memory (TCM)

There is 64KB of tightly coupled memory per CPU in the cluster. The latency from the TCMs in single cycle latency, same as landing a transaction in the Cache.

- SECDED ECC protected per 32 bits
- Readable/writable from system
- Split into A and B banks (with B further splitting into B0 and B1 interleaved banks)
  - 32KB TCMA (ATCM)
  - 16KB TCMB0 (B0TCM)
  - 16KB TCMB1 (B1TCM)

2.6.4 Typical Use Case

Hot data and code with poor locality can be placed in the TCM for the best performance. The TCMs are sometimes loaded with some boot-up code to initialize the R5F Memory Protection Unit (MPU) to further execute code out of DDR. Because some applications do not cache well, there are two TCM interfaces that permit connection to configurable memory blocks of *Tightly-Coupled Memory* (ATCM and BTCM). ATCM is usually used for code and the dual ported BTCM for data.

- An ATCM typically holds interrupt or exception code that must be accessed at high speed, without any potential delay resulting from a cache miss.
- A BTCM typically holds a block of data for intensive processing, such as audio or video processing. BTCM is dual ported to increase bandwidth.
- The TCMs are external to the processor. This provides flexibility in optimizing the TCM subsystem for performance, power, and RAM type.

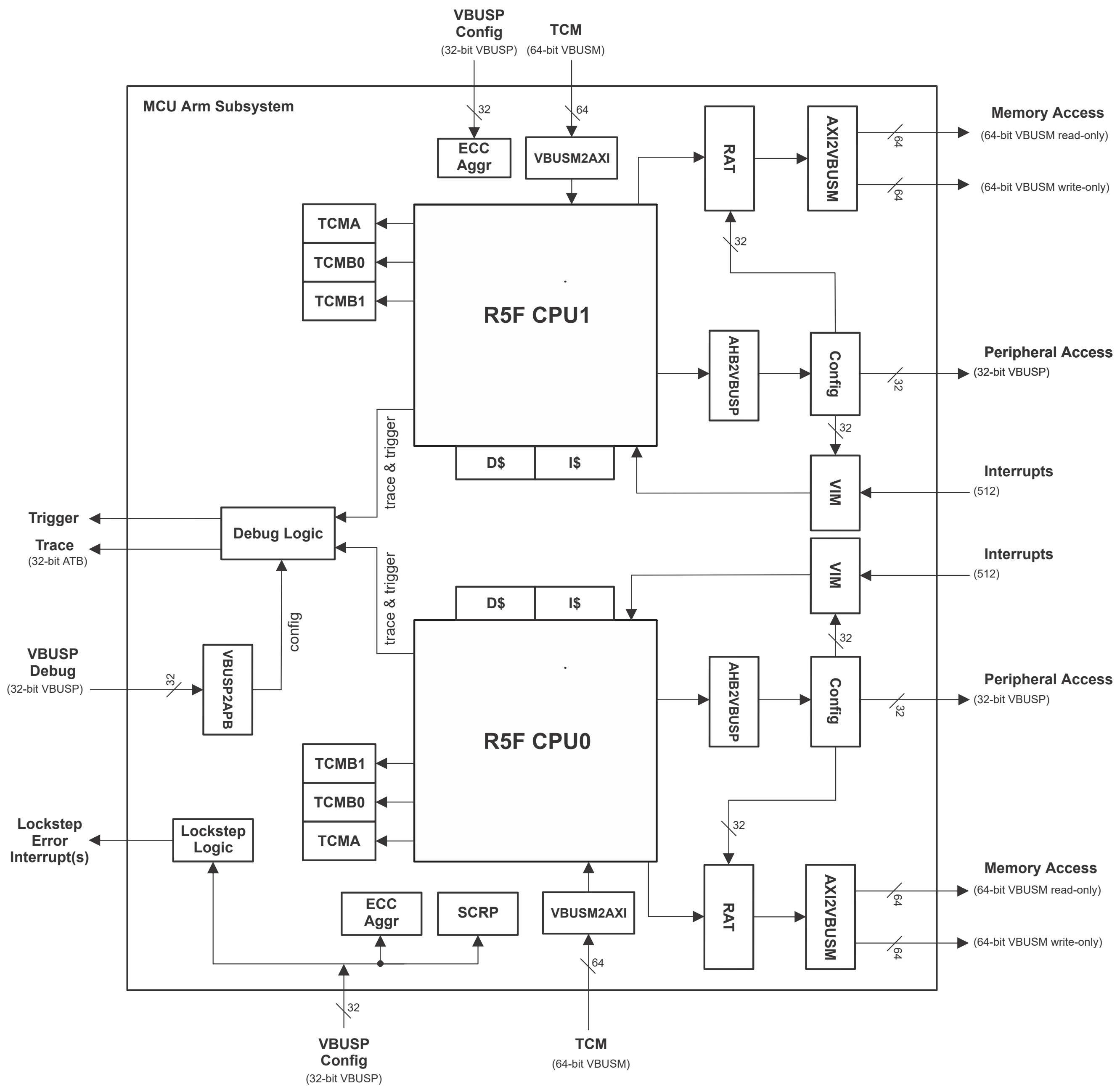

Figure 2-3. R5FSS Block Diagram

2.6.5 Relevant Links

- Arm Cortex-R5F Home Page: Cortex-R5 – ARM
- Cortex-R5 Technical Reference Manual r1p2: https://developer.arm.com/documentation/ddi0460/d
- Arm Cortex-R Series Programmer's Guide: https://developer.arm.com/documentation/den0042/a/Coding-for-Cortex-R-Processors
- For more information, see the *Dual-R5F MCU Subsystem* chapter in the *DRA829/TDA4VM Technical Reference Manual*.

2.7 TI's C6x Subsystem

The C66x subsystem is based on the TI's standard TMS320C66x DSP CorePac module. It includes subsystem logic to ease the C66x CorePac integration into the SoC, while maximizing software reuse from previous devices.

2.7.1 Memory Layout

- Program Memory Controller (PMC): 32KB L1 program memory (L1P), configured as 32 byte line cache, 2KB page
- Data Memory Controller (DMC): 32KB L1 data memory, configurable as cache and/or SRAM
- Unified Memory Controller (UMC): 288KB L2 memory
  - 256 KB configurable as cache or SRAM
  - 32KB always SRAM

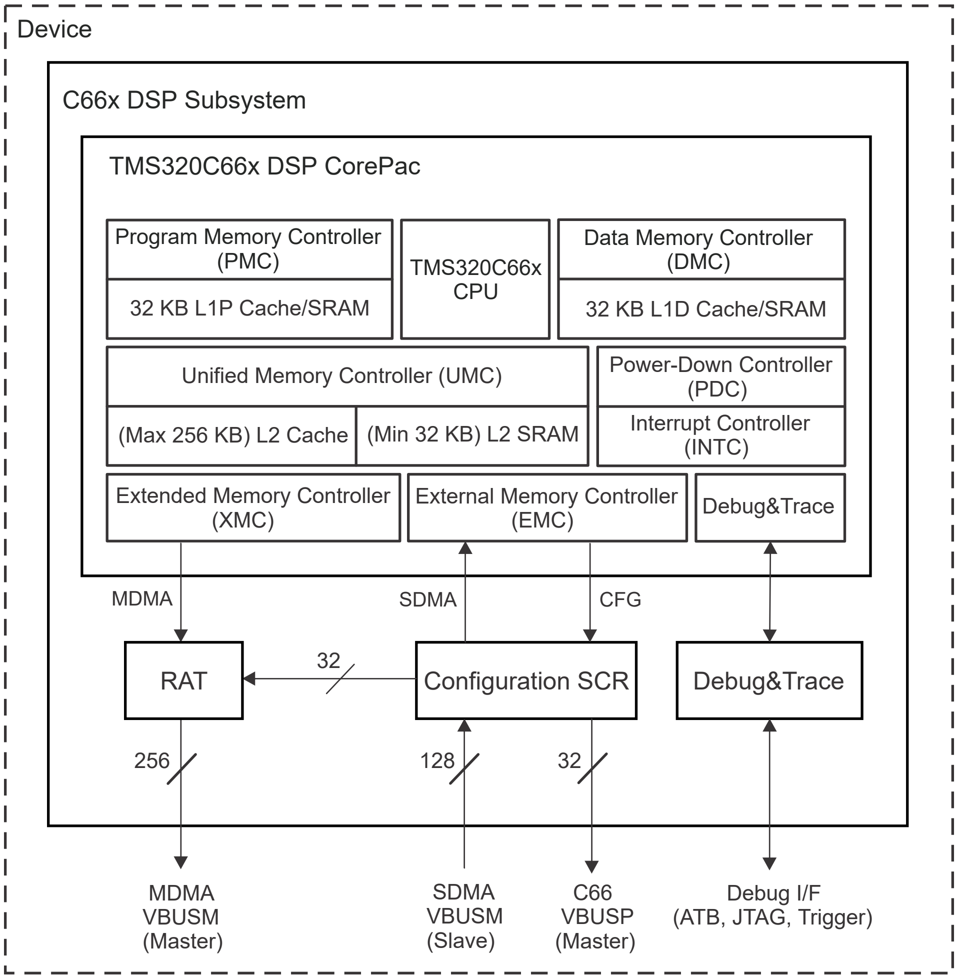

Figure 2-4. C66SS Overview

2.7.2 Relevant Links

- *TMS320C66x DSP CorePac User Guide*
- *TMS320C66x DSP Cache User Guide*
- For more information, see the *C66x DSP Subsystem* chapter of the *DRA829/TDA4VM Technical Reference Manual*.

2.8 TI's C7x Subsystem

The TMS320C71x is the next-generation fixed and floating-point DSP platform. The C71x DSP is a new core in the Texas Instruments' DSP family. The C71x DSP supports vector signal processing, providing significant lift in DSP processing power over a broad range of general signal processing tasks in comparison to the C6x DSP family. In addition, the C71x provides several specialized functions which accelerate targeted functions by more than 30 times. Besides expanding vector processing capabilities, the new C71x core also incorporates advanced techniques to improve control code efficiency and ease of programming such as branch prediction, protected pipeline, precise exception and virtual memory management.

2.8.1 Memory Layout

- Level 1 (L1):
  - L1 program memory controller (PMC) with 32KB L1P memory, all cache (no support for L1P SRAM)
  - L1 data memory controller (DMC) with 48KB L1D memory, configurable as cache and/or SRAM. Example, for TDA4VM, 32KB is cache, remaining 16KB will be SRAM for Look up Table implementation.
- Level 2 (L2):
  - L2 unified memory controller (UMC) with 512KB L2 memory, configurable as cache and/or SRAM
  - In SDK by default it is configured as 64KB cache and 448 KB SRAM.
- Full coherence between L1D cache, SE, L2 SRAM, MSMC SRAM and DDR
- MSMC memory can be assigned to the C71x for improved performance. SDK defaults to assigning maximum available MSMC SRAM to C71x.

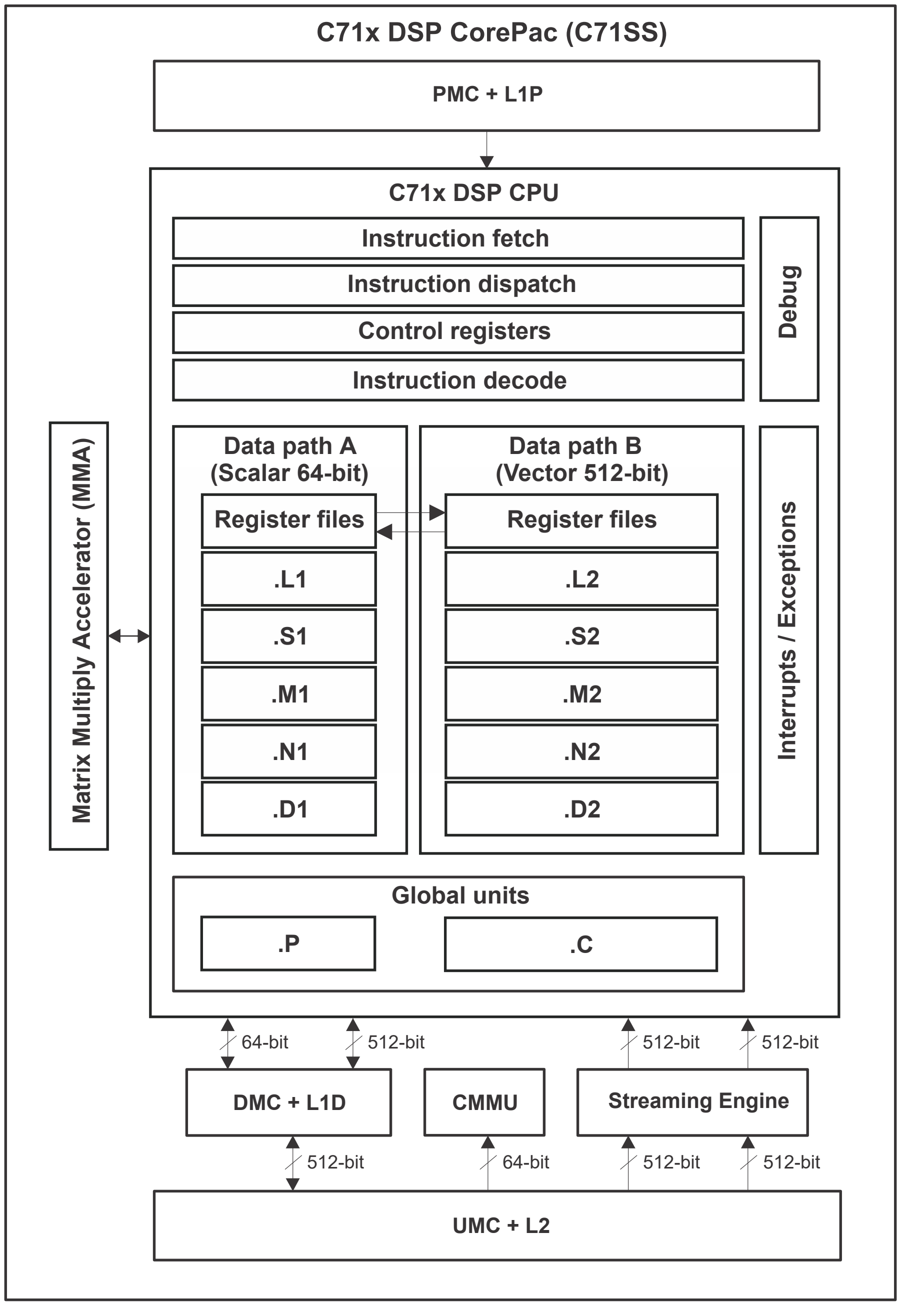

Figure 2-5. C71SS Overview

2.8.2 Relevant Links

- For more information, see the *C71x DSP Subsystem* chapter in the *DRA829/TDA4VM Technical Reference Manual*.
- *C7000 C/C++ Optimizing Compiler User's Guide*
- *C7000 Optimization Guide*
- *C7000 Host Emulation Guide*
- *C6000 to C7000 Migration User's Guide*
- Learning Software Pipeline Feedback Information - video

2.9 DDR Subsystem

The DDR subsystem in this device comprises DDR controller, DDR PHY and wrapper logic to integrate these blocks in the device. The DDR subsystem is referred to as DDRSS0 and is used to provide an interface to external SDRAM devices that can be utilized for storing program or data. DDRSS0 is accessed via MSMC, and not directly through the system interconnect.

2.9.1 Relevant Links

- For more information, see the *DDR Subsystem (DDRSS)* chapter in the *DRA829/TDA4VM Technical Reference Manual*.

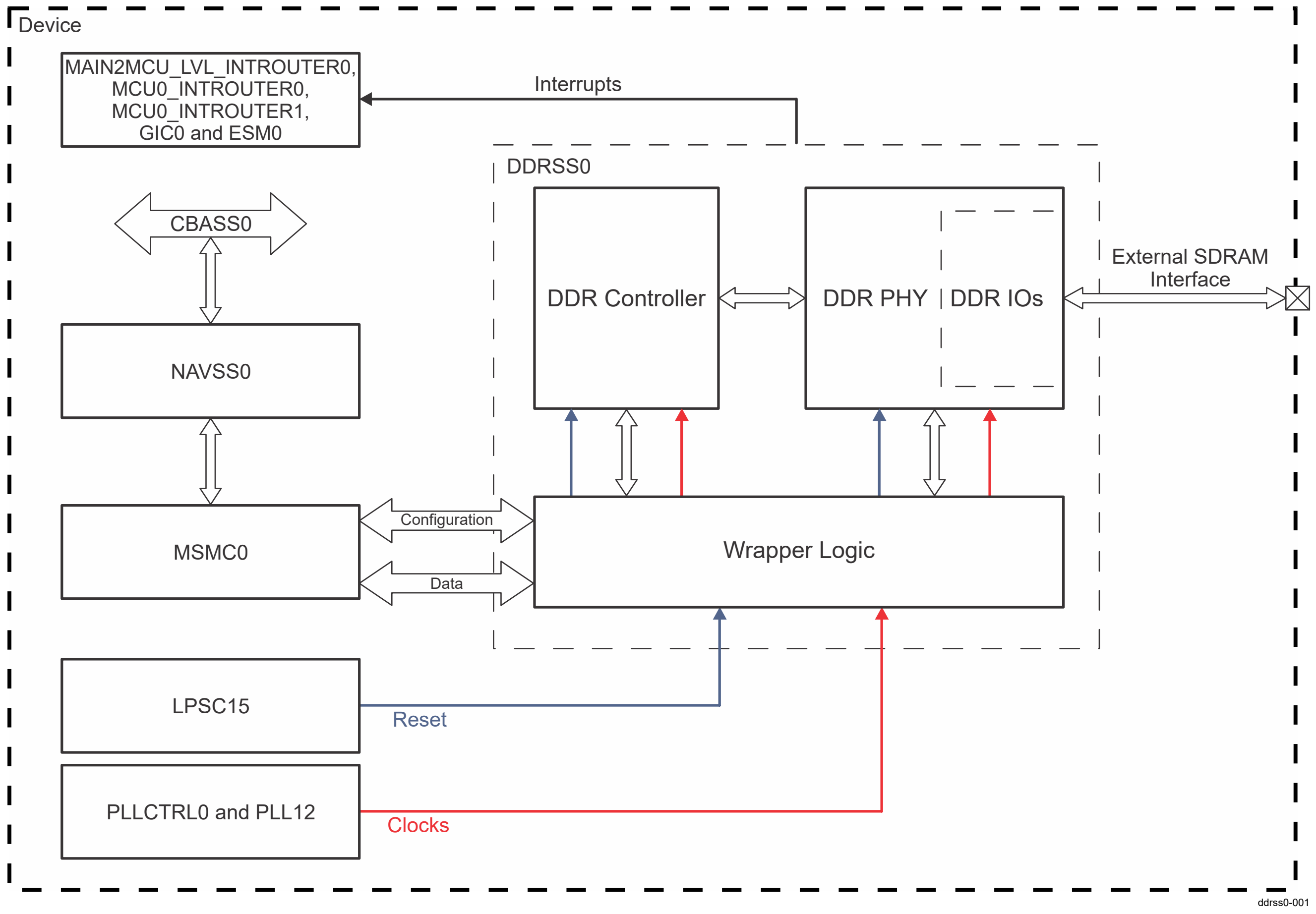

Figure 2-6. DDRSS Overview

3 Performance numbers

3.1 SDK Data Sheet

SDK release from TI has some benchmarks published as a part of the SDK documentation. These can be referred for details on SDK performance.

- Linux SDK performance guide: link
- RTOS SDK data sheet: link

Note that these links may change release on release but can be navigated to from the PROCESSOR-SDK-J721E on ti.com.

3.2 Memory Access Latency

Table 3-1 and Table 3-2 illustrate the relative memory access latency (read) from each core to primary memory end-points in the SoC system. Table 3-1 shows the latency for a core to access different memory end-points with access to DDR as a baseline. For example, in the first row of Table 3-1, the A72 core has a latency to access MSMC that is 33% of the latency to access DDR. Latencies to MCU OCRAM and MAIN OCRAM are both greater than the latency to DDR. Table 3-2 compares the latency from each core to DDR, with the A72 core's access to DDR as the baseline. The MCU R5 core takes 2.55x longer to access DDR than the A72 core.

It should be noted that DDR access latency is typically not constant due to factors such as SDRAM refresh cycles and periodic retraining. To provide an accurate comparison, this analysis uses optimal DDR access latency by excluding non-deterministic factors. Other SRAM-based memory end-points do not have variance in access latency.

Overall, these tables provide insight into the relative memory access latency for each core in the system, with implications for system performance and optimizations.

Note that the relative performance in Table 3-2 is for the worst case latency, such that, in cases when the data is read from the physical memory rather than the cache. If data being processed has a lot of locality, then the caches will hide most of the latency. The performance boost of using the different memories starts to show up as the cache miss rate increases. Observed performance from a source to a destination can vary depending on caching and contention. The SDK data sheet compares the cached performance using benchmarks like LMBench for Linux running on A72 and a memory benchmarking application for FreeRTOS running on R5F.

Table 3-1. Relative Latency From Each Core to Memory End-Point

	DDR	MSMC	C7x L2SRAM	C6x L2SRAM	MCU OCRAM	MAIN OCRAM
A72	1.00x	0.33x	0.38x	1.30x	1.59x	1.13x
C7x	1.00y	0.36y	0.03y	1.24y	1.50y	1.08y
C6x	1.00z	0.63z	0.67z	0.01z	0.52z	0.28z
MCU R5F	1.00a	0.73a	0.78a	0.54a	0.20a	0.47a
MAIN R5F	1.00b	0.71b	0.75b	0.51b	0.53b	0.42b

Table 3-2. Comparison of DDR Access Latency From Each Core

	A72	C7x	C6x	MCU R5F	MAIN R5F
DDR	1.00c	1.12c	1.98c	2.55c	2.37c

From above tables, items which should be considered during system design:

- Using memory local to the processor where software is running, results in reduced memory access latency, when compared to DDR
- When using memory to share data between cores (A72/R5/C6/C7)
  - The memory access time will be different depending on the core the software is running on.
  - Care should be taken when selecting which memory will be used to share the data
- MSMC memory, has low access time from all cores, when compared to DDR. Use of MSMC memory can lead to increased performance in many use cases. The most efficient use of this memory for the overall system should be reviewed.

4 Software Careabouts When Using Different Memories

4.1 How to Modify Memory Map for RTOS Firmwares

MCU R5F, MAIN R5Fs, C66x, C7x run FreeRTOS in the default SDK. Memory map for these FreeRTOS firmwares can be altered using Linker Command files in the SDK. See an example linker command file here along with the include files here and here. Note, after altering the memory map, always do an application clean build.

Changes made from linker command files get reflected in the generated map files (*.map). Map files can be located in the generated binary folder of the build environment. On a Linux system, you can run “readelf –l” on the elf image to generate a map file view as well.

Note, all system shared memories should be used such that the different cores do not corrupt each others memory space. It is recommended to create a system memory map design in a spreadsheet or similar tool and see if there are overlaps.

4.2 DDR Sharing Between RTOS Core and HLOS

DDR is a shared resource between all the cores (R5F, M3, C6x, C7x, A72) in the system and needs to be partitioned for usage from different cores.

DDR usage:

- Code / Data for firmwares from different core
- HLOS for its dynamic memory requirements
- IPC between cores

To understand the system memory map in case of the vision_apps + Linux/QNX use case, see this developer note. A System generated memory map can also be seen in an autogenerated file at path: *vision_apps/platform/j721e/rtos/system_memory_map.html*

DDR memory ranges being used for RTOS firmwares should be non-overlapping with each other and should also be reserved from the HLOS side.

- In a Linux system, the device tree “reserved_memory” node can be used. See example here.
- If using QNX, documentation on making updates to system memory map can see here.

If DDR region used for MAIN R5F (say) is not reserved from HLOS, then the HLOS can use that space leading to memory corruption, as the MAIN R5F Software is running with the understanding that the DDR memory range is dedicated for its use.

4.3 MCU On-Chip RAM Usage by Bootloader

The MCU_MSRAM or OCMC is used by the bootloader SPL/SBL and hence there are some constrains on its usage by the application.

For more information, see this developer note for the OCMC usage. Table 4-1 captures these details.

Table 4-1. Memory Map Considerations for Using MCU On-Chip RAM

Address	Size	Purpose	Persistence
0x41C00000	512 KB	Hold SPL/SBL Image	Only during SPL/SBL execution. Can be claimed once MCU1_0 app is jumped to.
0x41C80000	8 KB	Hold the board configuration passed between SPL/SBL and SCISERVER APP.	Can be claimed once MCU R5F app executes Sciclient_init() on MCU R5F.
0x41C82000	502 KB	Loadable applications on MCU On-Chip RAM (including SCISERVER APP)	Throughout MCU R5F app execution.
0x41CFFB00	~1.2 KB	Common header used in ROM combined image format as well.	Can be claimed once MCU R5F app executes Sciclient_init() on MCU R5F.

4.4 MSMC RAM Default SDK Usage

4.4.1 MSMC RAM Reserved Sections

For TDA4VM (J721E) 8MB of MSMC RAM is present and mapped at 0x7000_0000. 128KB from start and 64KB from end is reserved by ATF and DMSC (for IPC) respectively and hence should not be used.

Table 4-2. MSMC RAM Reserved Sections

Name	Start Address	End Address	Size	Attributes	Description
MSMC_MPU1	0x70000000	0x7001FFFF	128.00 KB	RWIX	MSMC reserved for ATF (Arm Trusted Firmware) on MPU1
Usable MSMC (Default SDK assigns this to MSMC_C7x_1)	0x70020000	0x707E7FFF	7.78 KB	RWIX	MSMC for C7x_1
MSMC_DMSC	0x707F0000	0x707FFFFF	64.00 KB	RWIX	MSMC reserved for DMSC IPC

4.4.2 MSMC RAM Configuration as Cache and SRAM

The configuration of the MSMC RAM as Cache or SRAM is dictated by the common board configuration passed to the Device Manager. For details on how to do this, see the SDK documentation's developer note on How to configure K3 MSMC memory for use as SRAM or L3 cache?.

Default SDK configures the whole of MSMC as SRAM and gives this to the C7x DSP for running deep learning algorithms. This configuration can be changed based on the customer use cases.

4.5 Usage of ATCM from MCU R5F

In the special case of MCU R5F, the core is taken out of reset by the ROM code. This is different than how other cores in the system are booted. The other cores MAIN R5F or DSPs are booted by the software bootloader running on the MCU R5F or the A72.

The ATCM (or TCM-A) is not enabled at power on for the R5F and the ROM code does not explicitly enable it either. The R5 SPL that comes up after the ROM code does not alter the state of the R5F and as such, does not enable the ATCM. This means the ATMC should not be used from the MCU R5F application unless it has been explicitly enabled. For this reason, BTCM is used to store the reset vector in for the MCU R5F firmware.

SBL however, unlike R5 SPL, explicitly enables the ATCM and it can be used if SBL is being used as the bootloader instead of the R5 SPL and u-boot.

4.6 Usage of DDR to Execute Code from R5F

Upon PORz, R5F's default MPU (memory protection unit) is not setup to have execute permissions for DDR, see Table 7-1 of the Arm Cortex R5F TRM.

To execute code from DDR, the MPU needs to be re-configured in the MPU init (__mpu_init()) function from the startup code, giving executable permissions to DDR mapped to 0x8000_0000. Once this is done, code can be executed from DDR.

5 Summary

There are many memories available to software running on the Jacinto7 family of devices. Knowing where these memories are located, and how they are being used is important to system design.

While DDR memory is readily available to all cores, it is not always the most efficient selection. Amount of memory available must be weighed against performance of that memory, when choosing where to store information. These software architecture decisions can be unique for a particular core, but may also have an overall system impact.

Reviewing entire systems use of memory is recommended as part of design phase. TI's Processor's E2E forum, is a great resource for any conversation on this subject.