Application Note

Tuning QoS and CoS Settings for DDR Bandwidth Optimization on TDA4x and AM6x Devices

Jared McArthur

ABSTRACT

The TDA4x and AM6x devices all contain a quality of service (QoS) scheme to give priority to specific transactions and balance loads across different applications and IP. When the QoS settings are not tuned for the device and applications, this can cause unexpected or undesirable behavior. The case study covered within this document demonstrates the undesirable behavior that is caused by this lack of tuning. The display suffers from sync lost errors when the auto valet parking (AVP) demo is running. This is due to a lack of proper QoS and class of service (CoS) settings to ensure that the display has priority. Once the CoS settings within the DDR controller are set to the correct values, the sync lost issue is no longer observed.

Table of Contents

Trademarks

1 Data Movement within the TDA4VH

Note

Read section **3. System Interconnect** of the TDA4VH TRM for more details.

To understand how transactions are balanced and prioritized, it's important to have a general grasp on the system interconnects and how the data flows. In the following case study, we will focus primarily on the data routing of the display subsystem (DSS) and C7x to the DDR subsystem (DDRSS), but it's good to have a general understanding of all the initiators and targets.

Figure 1-1 (taken from the TDA4VH TRM) shows a high level diagram of initiators and targets and the direction of their requests. The DSS falls within the "Initiator" block. The section **3.2.6 Initiator-Target Connections** within the TDA4VH TRM contains **Connectivity Matrixes** that further elaborate the relationships between the initiators and targets within the system.

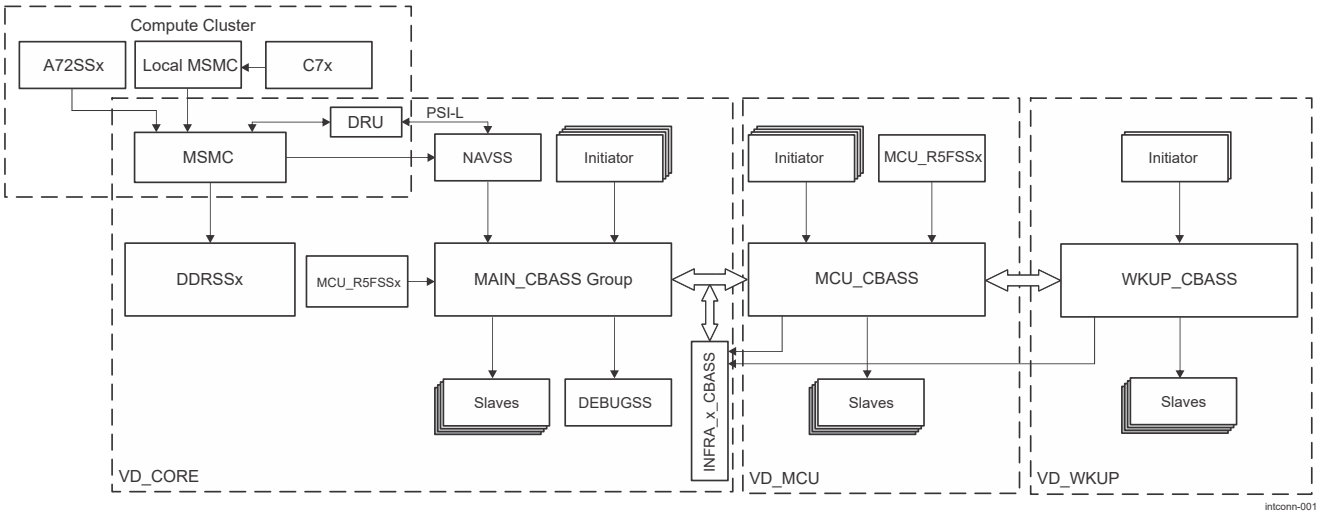

Figure 1-1. Device System Interconnect Overview

A C7x to DDR request largely follows the following path: C7x → MSMC → DDRSS

A DSS to DDR request largely follows the following path: DSS → Main CBASS → NAVSS → MSMC → DDRSS

1.1 Common Bus Architecture Subsystem (CBASS)

The system All modules and subsystems in the device communicate with each other through the system interconnect. It is partitioned into the following sections:

- CBASS0 interconnect
- INFRA_CBASS0 interconnect
- MCU_CBASS0 interconnect
- WKUP_CBASS0 interconnect

Most modules are connected to the CBASS0 interconnect within the MAIN domain. The CBASS contains quality of service mechanisms for some modules that are explained in later sections.

1.2 Navigator Subsystems (NAVSS)

Note

Read section **10.2.10 NAVSS North Bridge (NB)** of the TDA4VH TRM for more details.

CAUTION

This section will discuss the Main NAVSS (NAVSS0), not the MCU NAVSS.

The NAVSS0 consists of the following components:

- Unified DMA subsystem (UDMASS)
- Module subsystem (MODSS)
- North bridge subsystem (NBSS)
- Virtualization subsystem (VirtSS)
- ECC aggregators

Figure 1-2 (taken from the TDA4VH TRM) shows the NAVSS0 hardware components and their integration.

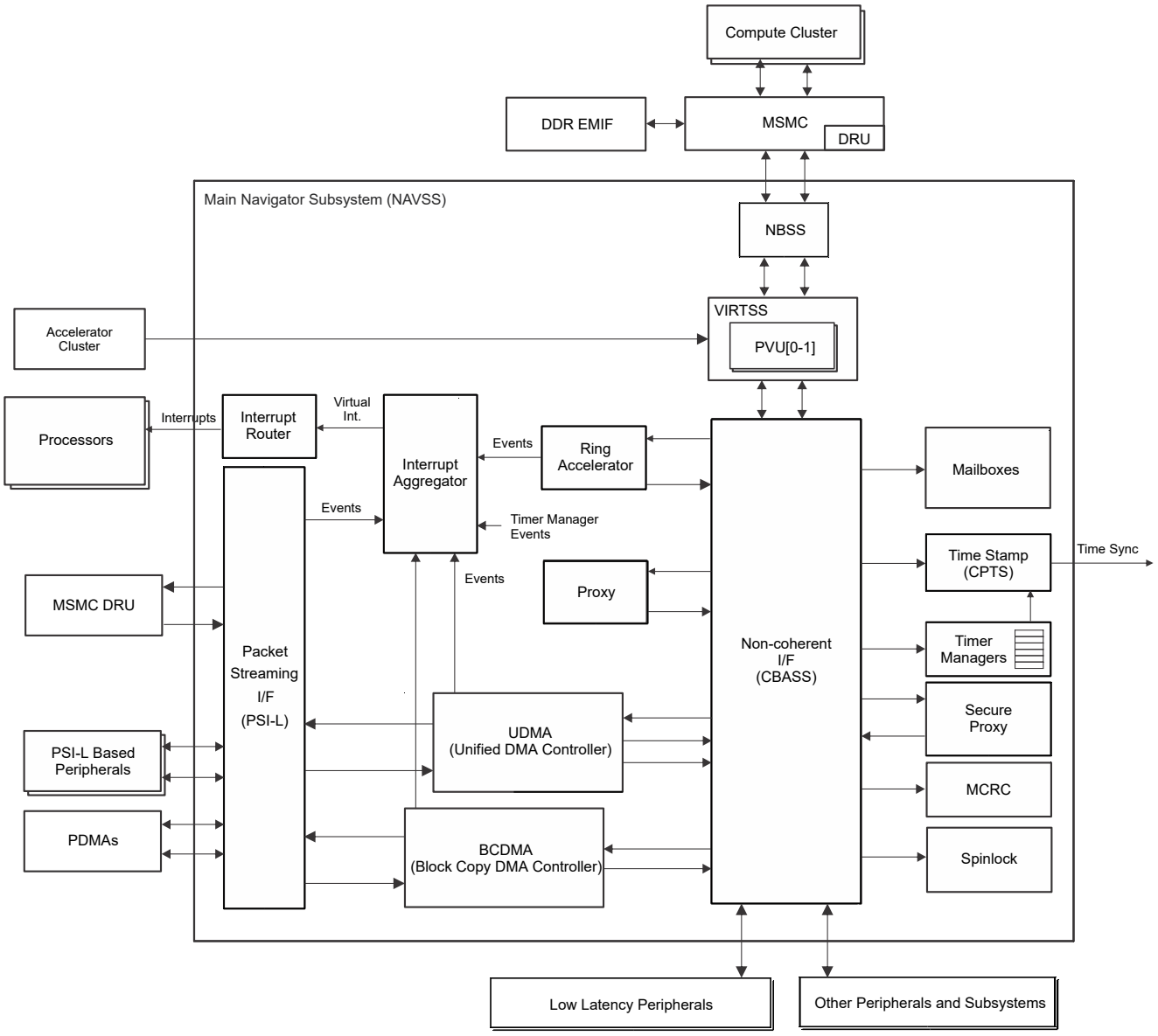

Figure 1-2. NAVSS0 Top-Level Block Diagram

1.2.1 NAVSS North Bridge (NB)

Note

Read section **10.2.10 NAVSS North Bridge (NB)** of the TDA4VH TRM for more details.

The NB bridges between VBUSM interfaces and a VBUSM.C interface. In other words, it bridges the CBASS (VBUSM) and MSMC (VBUSM.C). The NAVSS contains 2 North Bridges: NB0 and NB1.

Figure 1-3 (taken from the DRA829/TDA4VM TRM) displays the high level structure of the NAVSS0's NB0 and NB1.

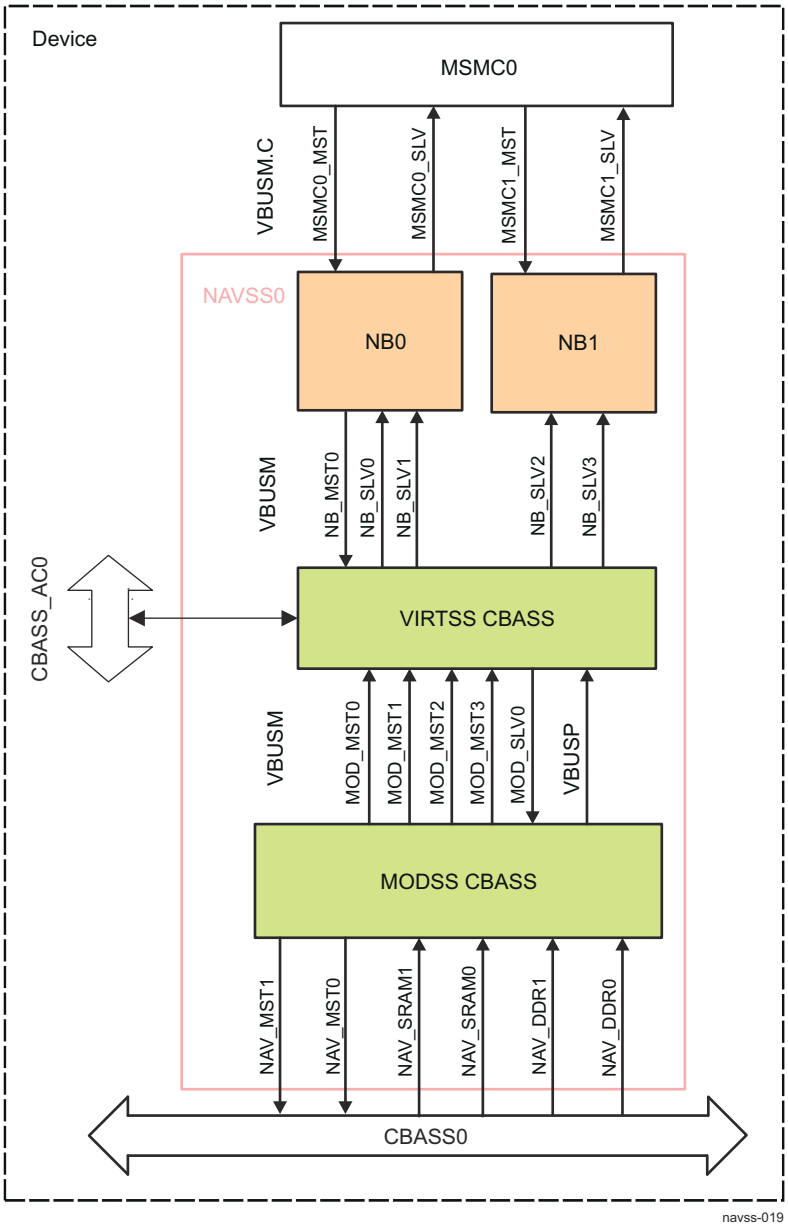

Figure 1-3. NB Overview

The NB's quality of service mechanisms will be explained in later sections.

1.3 Multicore Shared Memory Controller (MSMC)

Note

Read section **8.1 Multicore Shared Memory Controller (MSMC)** of the TDA4VH TRM for more details.

The MSMC provides high-bandwidth data-movement and resource access to and from the internal processing elements of the compute cluster and the rest of the system.

Figure 1-4 (taken from the TDA4VH TRM) shows an overview of the MSMC and its surrounding modules.

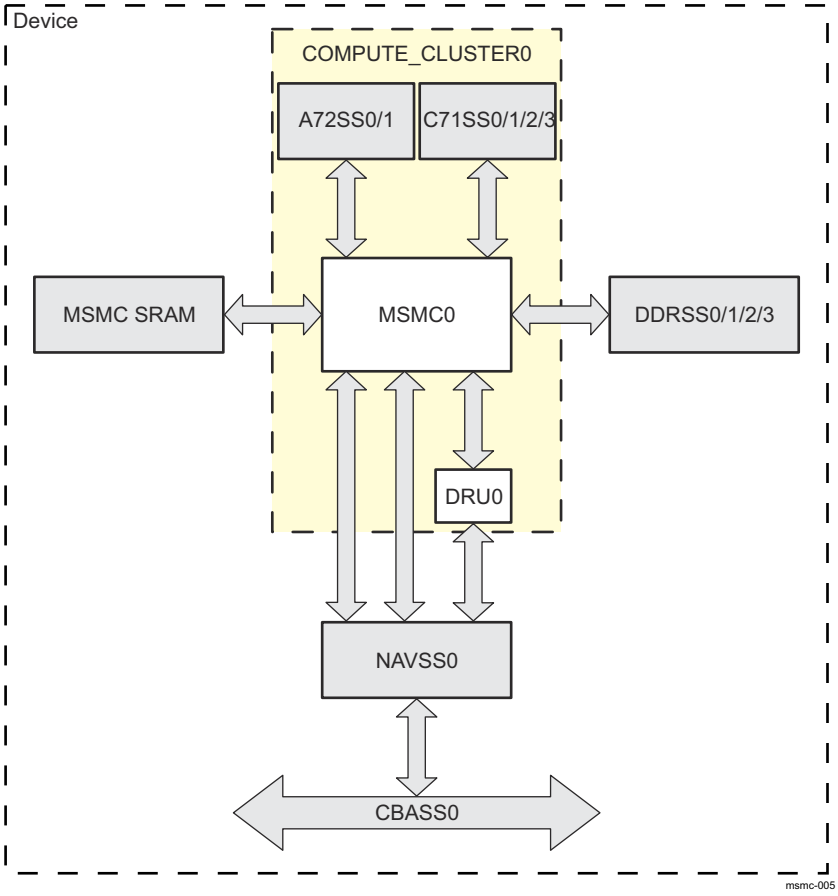

Figure 1-4. MSMC Overview

The MSMC's quality of service mechanisms is explained in later sections.

2 Quality of Service (QoS)

Note

Read section **3.2.1 Quality of Service (QoS)** of the TDA4VH TRM for more details.

Quality of service is the use of mechanisms to control traffic within a network. In this case, that network is the TDA4VH device. There are two (system-wide) methods of quality of service offered: order ID and arbitration by priority. Order ID controls the mapping of a transaction's master and that master's channels (if there are multiple); this allows for balancing traffic across different paths. Priority offers arbitration capabilities to improve latency and bandwidth.

Priority of transactions is relatively simple to understand. Priorities range from 0 to 7 with 0 equating to the highest priority and 7 equating to the lowest priority. Transactions with higher priorities are handled before transactions with lower priorities. The way to set the priority of a request varies from IP to IP, but there is usually a CTRL MMR or register within the IP to set the priority. For example, the DSS' DSS_DISPC_0_COMMON_M_DSS_CBA_CFG register controls the priority level of DSS requests.

Table 2-1. DSS Priority Register

Register	Field	Bits	Description
DSS_DISPC_0_COMMON_M_DSS_CBA_CFG	PRI_HI	5:3	The value sent out on the PRI_HI bus from DSS to CBA Indicates the priority level for high-priority [MFLAG] transactions. • Value of 0x0 indicates highest priority • Value of 0x7 indicates lowest priority
	PRI_LO	2:0	The value sent out on the PRI_LO bus from DSS to CBA Indicates the priority level for normal [non-MFLAG] transactions. • Value of 0x0 indicates highest priority • Value of 0x7 indicates lowest priority

Transactions' order IDs can be set within the initiators' registers or CBASS registers. The effects of order IDs are less simple than priorities, and different subsystems within the data movement architecture use order ID to route and balance loads through different paths.

Note

By default, all masters send transactions with order ID = 0.

2.1 NAVSS0

The interconnect inside NAVSS0 uses order ID to provide multiple (at least two) parallel paths to DDR and a separate set of multiple (at least two) parallel paths to SRAM. NAVSS0 also provides multiple (at least two) parallel paths for the DMA traffic to SoC level, which can provide isolated DMA traffic paths.

Within the NAVSS0, the two North Bridges route SRAM and DDR traffic to and from the MSMC. North Bridge 0 routes SRAM traffic and North Bridge 1 routes DDR traffic.

2.1.1 NAVSS0 North Bridge

Note

Read section **10.2.10.2.10 Quality of Service** of the TDA4VH TRM for more details.

Each North Bridge receives multiple sources that are separated by order ID. For North Bridge 0: source 0 receives all transactions with order IDs 0-7 and source 1 receives all transactions with order IDs 8-15. For North Bridge 1: source 0 receives all transactions with order IDs 0-4, source 1 receives order IDs 5-9, and source 2 receives order IDs 10-15. These parallel paths spread transaction loads by order ID. The order IDs that each source receives are not programmable by the user.

CAUTION

The order IDs that each source receives can change between different devices. For example, the TDA4VM only routes two sources for both North Bridge 0 and 1.

Order ID also influences the ordering of commands. Each read command received from the VBUSM interface with a particular order ID value will have their read data returned in exactly the same order, even if the commands are from different masters. If the reads use different order ID values then that read data can be returned in any order, whichever is received first on the VBUSM.C interface.

To route return traffic from the VBUSM.C back to the correct VBUSM source, the order ID is used. Due to this, the order IDs of the sources cannot overlap. This is not an issue due to the sources inherently routing different order IDs.

2.1.1.1 Normal vs Real-Time Traffic

The north bridge uses 3 threads to separate traffic to the MSMC:

- Thread 0: commands to VBUSM.C
- Thread 1: commands from VBUSM.C
- Thread 2: real-time commands to VBUSM.C

To support quality of service, the North Bridge provides registers to map the sources to either the normal thread (0) or real-time thread (2). Any source mapped to the real-time thread will be arbitrated before the normal thread. If there are multiple sources mapped to the same thread, then they are arbitrated based on priority, and if they are the same priority, then by round robin. In this way, the North Bridge is able to split traffic and assign priority by order ID (and in a way expand the number of possible priorities).

Programming whether an order ID is mapped to the normal or real-time thread is done within the NAVSS North Bridge MMR registers, specifically within the NAVSS_NORTH_x_NBSS_NBx_MMRS_threadmap (x signifies 0 or 1) register.

Table 2-2. NAVSS North Bridge Threadmap Registers

Register	Field	Bit(s)	Description
NAVSS_NORTH_0_NBSS_NB0_MMRS_threadmap	RESERVED	31:3	Reserved
	THREADMAP	1	Maps order IDs 8-15 to a VBUSM.C thread number: • 0: VBUSM.C thread 0 (non-real time traffic) • 1: VBUSM.C thread 2 (real time traffic)
		0	Maps order IDs 0-7 to a VBUSM.C thread number: • 0: VBUSM.C thread 0 (non-real time traffic) • 1: VBUSM.C thread 2 (real time traffic)

Table 2-2. NAVSS North Bridge Threadmap Registers (continued)

Register	Field	Bit(s)	Description
NAVSS_NORTH_1_NBSS_NB1_MMRS_threadmap	RESERVED	31:3	Reserved
	THREADMAP	2	Maps order IDs 10-15 to a VBUSM.C thread number: • 0: VBUSM.C thread 0 (non-real time traffic) • 1: VBUSM.C thread 2 (real time traffic)
		1	Maps order IDs 5-9 to a VBUSM.C thread number: • 0: VBUSM.C thread 0 (non-real time traffic) • 1: VBUSM.C thread 2 (real time traffic)
		0	Maps order IDs 0-4 to a VBUSM.C thread number: • 0: VBUSM.C thread 0 (non-real time traffic) • 1: VBUSM.C thread 2 (real time traffic)

CAUTION

The NAVSS_NORTH_x_NBSS_NBx_MMRS_threadmap register's fields vary between devices. The above table is representative of the TDA4VH.

2.2 Multicore Shared Memory Controller (MSMC)

Note

Read section **8.1.2.11 MSMC Quality-of-Service** of the TDA4VH TRM for more details.

The MSMC provides two classes of traffic: real-time (RT) and non-real-time (NRT). These two classes of traffic correspond to the real-time thread (2) and normal thread (0) within the North Bridge. The MSMC provides a dedicated buffering at each arbitration point that can only be consumed by RT traffic, so NRT traffic cannot completely starve out RT requests.

There is no software control over the MSMC QoS hardware. Interfaces which do not support QoS features denote all traffic as non-real-time.

2.3 DDR Subsystem (DDRSS)

The DDRSS contains quality of service mechanisms through the MSMC2DDR bridge while also containing it's own unique mechanism: class of service (CoS). The DDR controller utilizes its CoS mechanism to map VBUSM.C threads and priorities to its internal threads and priorities.

2.3.1 MSMC2DDR Bridge

Note

Read sections **8.2.3.1 DDRSS MSMC2DDR Bridge** and **8.2.3.1.1 VBUSM.C Threads** of the TDA4VH TRM for more details

The MSMC2DDR bridge supports 2 threads:

- High Priority Thread (HPT): traffic received on VBUSM.C thread 2 belongs to HPT

- Low Priority Thread (LPT): traffic received on VBUSM.C thread 0 belongs to LPT

HPT has priority over LPT, and execution of commands from the command queue can be out-of-order. This ensures the HPT is guaranteed execution even when the LPT is blocked.

Because the MSMC2DDR bridge maintains data coherency across threads, priority inversion is possible. Any HPT transactions that depend on LPT transactions due to address conflicts are blocked until execution of those corresponding LPT transactions.

2.3.2 Class of Service (CoS)

Note

Read section **8.2.3.1.2 Class of Service (CoS)** of the TDA4VH TRM for more details.

Class of service is specific to the DDRSS, and controls how the system (VBUSM.C) priorities map to the DDRSS internal priorities. The MSMC2DDR bridge has the following registers to map VBUSM.C priorities to DRR controller priorities:

- Range match registers:
  - DDRSS_V2A_R1_MAT_REG
  - DDRSS_V2A_R2_MAT_REG
  - DDRSS_V2A_R3_MAT_REG
- Priority map registers:
  - DDRSS_V2A_LPT_DEF_PRI_MAP_REG
  - DDRSS_V2A_LPT_R1_PRI_MAP_REG
  - DDRSS_V2A_LPT_R2_PRI_MAP_REG
  - DDRSS_V2A_LPT_R3_PRI_MAP_REG
  - DDRSS_V2A_HPT_DEF_PRI_MAP_REG
  - DDRSS_V2A_HPT_R1_PRI_MAP_REG
  - DDRSS_V2A_HPT_R2_PRI_MAP_REG
  - DDRSS_V2A_HPT_R3_PRI_MAP_REG

Figure 2-1 (taken from the TDA4VH TRM) shows how priority map registers map incoming priorities to the appropriate DDR priorities.

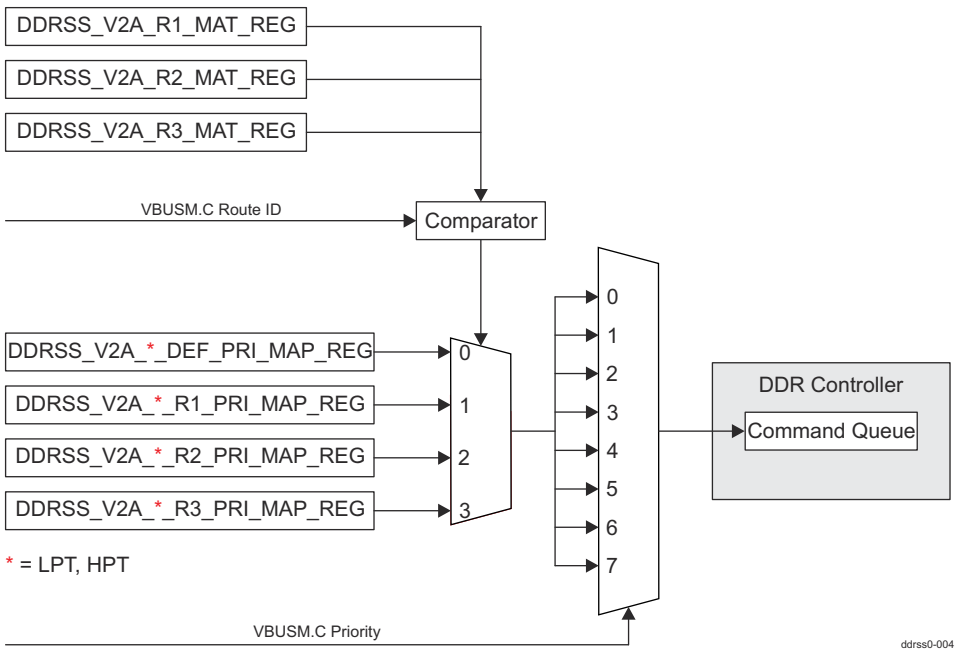

Figure 2-1. DDRSS CoS Mapping Diagram

2.4 QoS Summary

Now that we've discussed the mechanisms and IPs that are responsible for controlling QoS, it's important to put all of the details together, and understand what it means from a high level point of view.

As said before, QoS is achieved through two mechanisms: order ID and priority. The order ID is a programmable field for requests and impacts how requests are routed between different IPs. The order ID's primary focus is to provide a mechanism to balance how data flows between different paths. Priority controls the order that requests are serviced in.

The data movement architecture subsystems also introduced the concept of non-real-time (NRT) and real-time (RT) threads. The name for the NRT and RT attribute varies IP to IP.

Table 2-3. Non-Real-Time and Real-Time Naming Across IPs

IP	Non-Real Time (NRT)	Real Time (RT)
MSMC	Non-real-time	Real-time
NAVSS North Bridge	Thread 0 (normal)	Thread 2 (real time)
DDR Controller	Low priority thread (LPT)	High priority thread (HPT)

The RT thread is arbitrated before the NRT thread, therefore giving transactions on the RT thread a higher priority. Whether a transaction falls into the NRT or RT thread is decided by the order ID and how the North Bridges are programmed. When you combine the concept of the NRT and RT threads with priorities, you can view of the hierarchy of requests as the following table.

Table 2-4. Priority Levels Across Non-Real-Time and Real-Time Requests

NRT or RT Request	Priority	Priority Level
Real-time	0	Highest priority
	1	Priority decreases down the list
	2	
	3	
	4	
	5	
	6	
	7	
Non-real-time	0	
	1	
	2	
	3	
	4	
	5	
	6	
	7	Lowest priority

3 Case Study: Display Sync Lost Issue

Let's begin looking at a case study which can demonstrate how QoS mechanisms can balance loads across a device.

3.1 Problem Statement

When running the AVP (auto valet parking) demo with a 4k display plugged in, the display suffers from frequent sync losts.

3.2 Setup and Recreation

3.2.1 Requirements

Table 3-1. Hardware and Software Requirements

Item	Link	Comments
J784S4XEVM	https://www.ti.com/tool/J784S4XEVM	N/A
SD Card	N/A	N/A
4K 30fps monitor	N/A	Connect the monitor to Display Port1
PROCESSOR-SDK-RTOS-J784S4	ti-processor-sdk-rtos-j784s4-evm-09_02_00_05.tar.gz	Version 09.02
SOC generic TI sample input data set	psdk_rtos_ti_data_set_09_02_00.tar.gz	Listed within the PROCESSOR-SDK-RTOS-J784S4 page
SOC specific tidl models	psdk_rtos_ti_data_set_09_02_00_j784s4.tar.gz	Listed within the PROCESSOR-SDK-RTOS-J784S4 page
PROCESSOR-SDK-LINUX-J784S4	ti-processor-sdk-linux-adas-j784s4-evm-09_02_00_05-Linux-x86-Install.bin	Version 09.02
RTOS patches	rtos-patches.tar.xz • 0001-vision_apps-Remove-the-DSS-application-from-MCU2_0.patch • 0002-vision_apps-Remove-display-use-from-the-AVP-demo.patch	Patches to apply to the vision_apps repository
Linux patches	linux-patches.tar.xz • 0001-arm64-dts-ti-k3-j784s4-vision-apps-Re-enable-DSS-for.patch	Patches to apply to the ti-linux-kernel repository

3.2.1.1 RTOS Patches

3.2.1.1.1 0001-vision_apps-Remove-the-DSS-application-from-MCU2_0.patch

```
From d5bd0778612110390ed7a20e7bb9afb4c95f0c25 Mon Sep 17 00:00:00 2001
From: Jared McArthur <j-mcarthur@ti.com>
Date: Tue, 28 Jan 2025 11:14:48 -0600
Subject: [PATCH 1/2] vision_apps: Remove the DSS application from MCU2_0

Remove the DSS application from MCU2_0 to give Linux control of the
display driver.

Signed-off-by: Jared McArthur <j-mcarthur@ti.com>
---
 platform/j784s4/rtos/common/app_cfg_mcu2_0.h | 13 +++++++++----
 1 file changed, 9 insertions(+), 4 deletions(-)

diff --git a/platform/j784s4/rtos/common/app_cfg_mcu2_0.h b/platform/j784s4/rtos/common/
app_cfg_mcu2_0.h
index a18c41b..8a96d2e 100755
--- a/platform/j784s4/rtos/common/app_cfg_mcu2_0.h
+++ b/platform/j784s4/rtos/common/app_cfg_mcu2_0.h
@@ -78,7 +78,7 @@
 #ifdef BUILD_MCU_BOARD_DEPENDENCIES

     #define ENABLE_CSI2RX
-    #define ENABLE_CSI2TX
```

```
+    // #define ENABLE_CSI2TX
     #undef  ENABLE_DSS_HDMI

     /* IMPORANT NOTE:
@@ -86,8 +86,8 @@
      * - When ENABLE_DSS_SINGLE is defined, only one of ENABLE_DSS_DSI or ENABLE_DSS_EDP should be
defined
      * - When ENABLE_DSS_DUAL is defined, ENABLE_DSS_DSI and ENABLE_DSS_EDP are not used, both EDP
and DSI are enabled unconditionally
      */
-    #define ENABLE_DSS_SINGLE
-    #undef  ENABLE_DSS_DUAL
+    // #define ENABLE_DSS_SINGLE
+
 #if defined(ENABLE_DSS_DUAL)
     #undef  ENABLE_DSS_SINGLE
@@ -102,7 +102,12 @@
     #undef  ENABLE_CSI2TX
 #endif
     #define ENABLE_I2C
-    #define ENABLE_BOARD
+    // #define ENABLE_BOARD
+    #undef  ENABLE_DSS_DUAL
+    #undef  ENABLE_DSS_SINGLE
+    #undef  ENABLE_DSS_DSI
+    #undef  ENABLE_DSS_EDP
+    #undef  ENABLE_DSS_HDMI
 #else

     #undef ENABLE_CSI2RX
--
2.34.1
```

3.2.1.1.2 0002-vision_apps-Remove-display-use-from-the-AVP-demo.patch

```
From c8059ede6e8a5cb38933f01b6c275a84539cd267 Mon Sep 17 00:00:00 2001
From: Jared McArthur <j-mcarthur@ti.com>
Date: Tue, 28 Jan 2025 11:29:07 -0600
Subject: [PATCH 2/2] vision_apps: Remove display use from the AVP demo

Remove display calls from the auto valey parking (AVP) demo. The demo
traditionally outputs to a display; remove this functionality so Linux
can own the display driver.

Disabling the display within the AVP demo allows for testing Linux's
display driver while the C7x cores are loaded.

Signed-off-by: Jared McArthur <j-mcarthur@ti.com>
---
 .../app_tidl_avp/avp_img_mosaic_module.c      |  73 ++++++++++-
 apps/dl_demos/app_tidl_avp/concerto.mak       |   2 +-
 apps/dl_demos/app_tidl_avp/main.c             | 119 ++++++---------------
 3 files changed, 97 insertions(+), 97 deletions(-)

diff --git a/apps/dl_demos/app_tidl_avp/avp_img_mosaic_module.c b/apps/dl_demos/app_tidl_avp/
avp_img_mosaic_module.c
index 21ace33..d43f7ad 100644
--- a/apps/dl_demos/app_tidl_avp/avp_img_mosaic_module.c
+++ b/apps/dl_demos/app_tidl_avp/avp_img_mosaic_module.c
@@ -61,7 +61,14 @@
  */

 #include "avp_img_mosaic_module.h"
-
+#include <fcntl.h>
+#include <linux/fb.h>
+#include <stdio.h>
+#include <stdlib.h>
+#include <string.h>
+#include <sys/ioctl.h>
+#include <sys/mman.h>
+#include <unistd.h>

 vx_status app_init_img_mosaic(vx_context context, ImgMosaicObj *imgMosaicObj, vx_int32 bufq_depth)
 {
@@ -143,7 +150,68 @@ void app_create_graph_img_mosaic(vx_graph graph, ImgMosaicObj *imgMosaicObj,
```

```
vx_

     return;
 }
+/*
+#include <fcntl.h>
+#include <linux/fb.h>
+#include <stdio.h>
+#include <stdlib.h>
+#include <string.h>
+#include <sys/ioctl.h>
+#include <sys/mman.h>
+#include <unistd.h>
+
+int main(int argc, char *argv[]) {
+int fb_fd = open("/dev/fb0", O_RDWR);
+if (fb_fd == -1) {
+perror("Error: cannot open framebuffer device");
+return 1;
+}
+
+struct fb_var_screeninfo vinfo;
+struct fb_fix_screeninfo finfo;
+
+// Get fixed screen information
+if (ioctl(fb_fd, FBIOGET_FSCREENINFO, &finfo)) {
+perror("Error reading fixed information");
+return 2;
+}
+
+// Get variable screen information
+if (ioctl(fb_fd, FBIOGET_VSCREENINFO, &vinfo)) {
+perror("Error reading variable information");
+return 3;
+}

+int screensize = vinfo.yres_virtual * finfo.line_length;
+
+// Map framebuffer to user memory
+char *fbp = (char *)mmap(0, screensize, PROT_READ | PROT_WRITE, MAP_SHARED, fb_fd, 0);
+if ((intptr_t)fbp == -1) {
+perror("Error: failed to map framebuffer device to memory");
+return 4;
+}
+
+// Open the image file (raw RGB data)
+FILE *img = fopen("image.rgb", "rb");
+if (!img) {
+perror("Error: cannot open image file");
+munmap(fbp, screensize);
+close(fb_fd);
+return 5;
+}
+
+// Read image data into framebuffer memory
+fread(fbp, 1, screensize, img);
+
+// Cleanup
+fclose(img);
+munmap(fbp, screensize);
+close(fb_fd);
+
+return 0;
+
+}
+*/
 vx_status writeMosaicOutput(char* file_name, vx_image out_img)
 {
     vx_status status;
@@ -194,6 +262,8 @@ vx_status writeMosaicOutput(char* file_name, vx_image out_img)
                 data_ptr += image_addr.stride_y;
             }

+
+
             if(num_bytes != (img_width*img_height))
             {
                 printf("Luma bytes written = %d, expected = %d\n", num_bytes,
```

```
img_width*img_height);
@@ -224,6 +294,7 @@ vx_status writeMosaicOutput(char* file_name, vx_image out_img)
                printf("CbCr bytes written = %d, expected = %d\n", num_bytes,
img_width*img_height/2);
             }

+
             vxUnmapImagePatch(out_img, map_id);

         }
diff --git a/apps/dl_demos/app_tidl_avp/concerto.mak b/apps/dl_demos/app_tidl_avp/concerto.mak
index fab60bc..b4b4677 100644
--- a/apps/dl_demos/app_tidl_avp/concerto.mak
+++ b/apps/dl_demos/app_tidl_avp/concerto.mak
@@ -3,7 +3,7 @@ ifeq ($(TARGET_CPU),$(filter $(TARGET_CPU), x86_64 A72 A53))
 include $(PRELUDE)

 TARGET      := vx_app_tidl_avp
-CSOURCES    := main.c avp_scaler_module.c avp_pre_proc_module.c avp_tidl_module.c
avp_post_proc_module.c fisheye_angle_table.c avp_img_mosaic_module.c avp_draw_detections_module.c
avp_display_module.c
+CSOURCES    := main.c avp_scaler_module.c avp_pre_proc_module.c avp_tidl_module.c
avp_post_proc_module.c fisheye_angle_table.c avp_img_mosaic_module.c avp_draw_detections_module.c

 ifeq ($(HOST_COMPILER),GCC_LINUX)
 CFLAGS += -Wno-unused-function
diff --git a/apps/dl_demos/app_tidl_avp/main.c b/apps/dl_demos/app_tidl_avp/main.c
index 9d23faf..5b0a673 100644
--- a/apps/dl_demos/app_tidl_avp/main.c
+++ b/apps/dl_demos/app_tidl_avp/main.c
@@ -78,7 +78,6 @@
 #include "avp_post_proc_module.h"
 #include "avp_draw_detections_module.h"
 #include "avp_img_mosaic_module.h"
-#include "avp_display_module.h"
 #include "avp_test.h"

 #ifndef x86_64
@@ -108,8 +107,6 @@ typedef struct {

     ImgMosaicObj imgMosaicObj;

-    DisplayObj displayObj;
-
     vx_char input_file_path[APP_MAX_FILE_PATH];
     vx_char output_file_path[APP_MAX_FILE_PATH];
     vx_char input_file_list[APP_MAX_FILE_PATH];
@@ -188,9 +185,7 @@ static void app_update_param_set(AppObj *obj);
 static void add_graph_parameter_by_node_index(vx_graph graph, vx_node node, vx_uint32
node_parameter_index);
 static void app_pipeline_params_defaults(AppObj *obj);
 static void app_find_object_array_index(vx_object_array object_array[], vx_reference ref, vx_int32
array_size, vx_int32 *array_idx);
-#ifndef x86_64
-static void app_draw_graphics(Draw2D_Handle *handle, Draw2D_BufInfo *draw2dBufInfo, uint32_t
update_type);
-#endif
+
 #ifdef AVP_ENABLE_PIPELINE_FLOW
 static vx_status app_run_graph_for_one_frame_pipeline(AppObj *obj, vx_int32 frame_id);
 #else
@@ -228,7 +223,7 @@ static void app_run_task(void *app_var)
     AppObj *obj = (AppObj *)app_var;
     vx_status status = VX_SUCCESS;

-    while(!obj->stop_task && (status == VX_SUCCESS))
+    while((status == VX_SUCCESS))
     {
         status = app_run_graph(obj);
     }
@@ -626,15 +621,6 @@ static void app_parse_cfg_file(AppObj *obj, vx_char *cfg_file_name)
                 }
             }
             else
-            if(strcmp(token, "display_option")==0)
-            {
-                token = strtok(NULL, s);
-                if(token != NULL)
```

```
-                {
-                    obj->displayObj.display_option = atoi(token);
-                }
-            }
-            else
             if(strcmp(token, "delay_in_msecs")==0)
             {
                 token = strtok(NULL, s);
@@ -718,7 +704,6 @@ static void app_parse_cfg_file(AppObj *obj, vx_char *cfg_file_name)

     if (obj->test_mode == 1)
     {
-        obj->displayObj.display_option = 1;
         obj->is_interactive = 0;
         obj->num_iterations = 1;
         /* if testing, just run the number of frames that are found in the expected
@@ -770,7 +755,6 @@ static void app_parse_cmd_line_args(AppObj *obj, vx_int32 argc, vx_char *argv[])
     if (set_test_mode == vx_true_e)
     {
         obj->test_mode = 1;
-        obj->displayObj.display_option = 1;
         obj->is_interactive = 0;
         obj->num_iterations = 1;
         /* if testing, just run the number of frames that are found in the expected
@@ -780,7 +764,6 @@ static void app_parse_cmd_line_args(AppObj *obj, vx_int32 argc, vx_char *argv[])
     }

     #ifdef x86_64
-    obj->displayObj.display_option = 0;
     obj->is_interactive = 0;
     #endif

@@ -876,7 +859,6 @@ static void add_graph_parameter_by_node_index(vx_graph graph, vx_node node, vx_u
 static vx_status app_init(AppObj *obj)
 {
     vx_status status = VX_SUCCESS;
-    app_grpx_init_prms_t grpx_prms;

     /* Create OpenVx Context */
     obj->context = vxCreateContext();
@@ -994,21 +976,10 @@ static vx_status app_init(AppObj *obj)
     {
         status = app_init_img_mosaic(obj->context, &obj->imgMosaicObj, AVP_BUFFER_Q_DEPTH);
     }
-    if(status == VX_SUCCESS)
-    {
-        status = app_init_display(obj->context, &obj->displayObj, "display_obj");
-    }
+
     appPerfPointSetName(&obj->total_perf , "TOTAL");
     appPerfPointSetName(&obj->fileio_perf, "FILEIO");

-    #ifndef x86_64
-    if(obj->displayObj.display_option == 1)
-    {
-        appGrpxInitParamsInit(&grpx_prms, obj->context);
-        grpx_prms.draw_callback = app_draw_graphics;
-        appGrpxInit(&grpx_prms);
-    }
-    #endif

     return status;
 }
@@ -1041,14 +1012,6 @@ static void app_deinit(AppObj *obj)

     app_deinit_img_mosaic(&obj->imgMosaicObj, AVP_BUFFER_Q_DEPTH);

-    app_deinit_display(&obj->displayObj);
-
-    #ifndef x86_64
-    if(obj->displayObj.display_option == 1)
-    {
-        appGrpxDeInit();
-    }
-    #endif

     tivxTIDLUnLoadKernels(obj->context);
     tivxImgProcUnLoadKernels(obj->context);
```

```
@@ -1086,7 +1049,6 @@ static void app_delete_graph(AppObj *obj)

     app_delete_img_mosaic(&obj->imgMosaicObj);

-    app_delete_display(&obj->displayObj);

     vxReleaseGraph(&obj->graph);
 }
@@ -1180,10 +1142,6 @@ static vx_status app_create_graph(AppObj *obj)

     app_create_graph_img_mosaic(obj->graph, &obj->imgMosaicObj, NULL);

-    if(status == VX_SUCCESS)
-    {
-        status = app_create_graph_display(obj->graph, &obj->displayObj, obj-
>imgMosaicObj.output_image[0]);
-    }

 #ifdef AVP_ENABLE_PIPELINE_FLOW
     /* Scalar Node - input is in Index 0 */
@@ -1547,8 +1505,11 @@ static vx_status app_run_graph_for_one_frame_pipeline(AppObj *obj, vx_int32
fram
                 APP_PRINTF("App Writing Outputs Start...\n");

                 snprintf(output_file_name, APP_MAX_FILE_PATH, "%s/
mosaic_output_%010d_1920x1080.yuv", obj->output_file_path, (frame_id - AVP_BUFFER_Q_DEPTH));
+                // printf("output_file_name=%s\n",output_file_name);
                 status = writeMosaicOutput(output_file_name, mosaic_output_image);
-
+                /*Kangjia get perception algorithm result*/
+                // printf("--------------------------------------\n");
+
                 APP_PRINTF("App Writing Outputs Done!\n");
             }
             /* Enqueue output */
@@ -1629,10 +1590,10 @@ static vx_status app_run_graph(AppObj *obj)
             APP_PRINTF("app_avp: Frame ID %d of %d ... Done.\n", frame_id, obj->start_frame + obj-
>num_frames);

             /* user asked to stop processing */
-            if((obj->stop_task) || (status == VX_FAILURE))
-            {
-                break;
-            }
+            //if((obj->stop_task) || (status == VX_FAILURE))
+            //{
+            //    break;
+            //}
         }
         printf("app_avp: Iteration %d of %d ... Done.\n", x, obj->num_iterations);
         appPerfPointPrintFPS(&obj->total_perf);
@@ -1644,17 +1605,17 @@ static vx_status app_run_graph(AppObj *obj)
         }

         /* user asked to stop processing */
-        if((obj->stop_task) || (status == VX_FAILURE))
-        {
-          break;
-        }
+        //if((obj->stop_task) || (status == VX_FAILURE))
+        //{
+        //  break;
+        //}
     }

 #ifdef AVP_ENABLE_PIPELINE_FLOW
     vxWaitGraph(obj->graph);
 #endif

-    obj->stop_task = 1;
+    //obj->stop_task = 1;

     return status;
 }
@@ -1838,8 +1799,8 @@ static void set_img_mosaic_defaults(AppObj *obj, ImgMosaicObj *imgMosaicObj)
 {
     vx_int32 idx = 0;
     vx_int32 in = 0;
```

```
-    imgMosaicObj->out_width    = 1920;
-    imgMosaicObj->out_height   = 1080;
+    imgMosaicObj->out_width    = 800;
+    imgMosaicObj->out_height   = 600;
     imgMosaicObj->num_inputs   = obj->enable_psd + obj->enable_vd + obj->enable_sem_seg;

     tivxImgMosaicParamsSetDefaults(&imgMosaicObj->params);
@@ -1848,8 +1809,8 @@ static void set_img_mosaic_defaults(AppObj *obj, ImgMosaicObj *imgMosaicObj)
     /* Right camera - PSD output */
     if(obj->enable_psd == 1)
     {
-        imgMosaicObj->params.windows[idx].startX  = 840;
-        imgMosaicObj->params.windows[idx].startY  = 200;
+        imgMosaicObj->params.windows[idx].startX  = 100;
+        imgMosaicObj->params.windows[idx].startY  = 50;
         imgMosaicObj->params.windows[idx].width   = 512;
         imgMosaicObj->params.windows[idx].height  = 512;
         imgMosaicObj->params.windows[idx].input_select   = in++;
@@ -1860,8 +1821,8 @@ static void set_img_mosaic_defaults(AppObj *obj, ImgMosaicObj *imgMosaicObj)
     /* Right camera - PSD output */
     if(obj->enable_vd == 1)
     {
-        imgMosaicObj->params.windows[idx].startX  = 1380;
-        imgMosaicObj->params.windows[idx].startY  = 200;
+        imgMosaicObj->params.windows[idx].startX  = 100;
+        imgMosaicObj->params.windows[idx].startY  = 50;
         imgMosaicObj->params.windows[idx].width   = 512;
         imgMosaicObj->params.windows[idx].height  = 512;
         imgMosaicObj->params.windows[idx].input_select   = in++;
@@ -1872,8 +1833,8 @@ static void set_img_mosaic_defaults(AppObj *obj, ImgMosaicObj *imgMosaicObj)
     /* Front camera - semantic segmentation output */
     if(obj->enable_sem_seg == 1)
     {
-        imgMosaicObj->params.windows[idx].startX  = 40;
-        imgMosaicObj->params.windows[idx].startY  = 250;
+        imgMosaicObj->params.windows[idx].startX  =20;
+        imgMosaicObj->params.windows[idx].startY  = 100;
         imgMosaicObj->params.windows[idx].width   = 768;
         imgMosaicObj->params.windows[idx].height  = 384;
         imgMosaicObj->params.windows[idx].input_select   = in++;
@@ -1887,10 +1848,6 @@ static void set_img_mosaic_defaults(AppObj *obj, ImgMosaicObj *imgMosaicObj)
     imgMosaicObj->params.clear_count  = 4;
 }

-static void set_display_defaults(DisplayObj *displayObj)
-{
-    displayObj->display_option = 0;
-}

 static void app_pipeline_params_defaults(AppObj *obj)
 {
@@ -1918,7 +1875,6 @@ static void app_default_param_set(AppObj *obj)
     obj->vdDrawDetectionsObj.params.num_classes = 1;
     obj->vdDrawDetectionsObj.params.class_id[0] = 1;

-    set_display_defaults(&obj->displayObj);

     app_pipeline_params_defaults(obj);

@@ -1972,31 +1928,4 @@ static void app_find_object_array_index(vx_object_array object_array[],
vx_refer
         vxReleaseImage(&img_ref);
     }
 }
-#ifndef x86_64
-static void app_draw_graphics(Draw2D_Handle *handle, Draw2D_BufInfo *draw2dBufInfo, uint32_t
update_type)
-{
-    appGrpxDrawDefault(handle, draw2dBufInfo, update_type);
-
-    if(update_type == 0)
-    {
-        Draw2D_FontPrm sHeading;
-        Draw2D_FontPrm sAlgo1;
-        Draw2D_FontPrm sAlgo2;
-        Draw2D_FontPrm sAlgo3;
-
-        sHeading.fontIdx = 0;
```

```
-        Draw2D_drawString(handle, 560, 5, "Analytics for Auto Valet Parking", &sHeading);
-
-        sAlgo1.fontIdx = 2;
-        Draw2D_drawString(handle, 270, 640, "Semantic Segmentation", &sAlgo1);

-        sAlgo2.fontIdx = 2;
-        Draw2D_drawString(handle, 920, 720, "Parking Spot Detection", &sAlgo2);
-
-        sAlgo3.fontIdx = 2;
-        Draw2D_drawString(handle, 1490, 720, "Vehicle Detection", &sAlgo3);
-    }
-
-  return;
-}
-#endif
--
2.34.1
```

3.2.1.2 Linux Patches

3.2.1.2.1 0001-arm64-dts-ti-k3-j784s4-vision-apps-Re-enable-DSS-for.patch

```
From 1245202b0656ae2d2f52b76951c4f2341c8290fb Mon Sep 17 00:00:00 2001
From: Jared McArthur <j-mcarthur@ti.com>
Date: Tue, 28 Jan 2025 14:04:34 -0600
Subject: [PATCH 1/1] arm64: dts: ti: k3-j784s4-vision-apps: Re-enable DSS for
 Linux

Re-enable DSS within the Linux domain when running vision_apps
applications.

The display driver is usually controlled by the MCU2_0 core when
vision_apps applications are run, and the Linux display driver is
disabled. Revert this behavior.

Signed-off-by: Jared McArthur <j-mcarthur@ti.com>
---
 .../boot/dts/ti/k3-j784s4-vision-apps.dtso    | 20 --------------------
 1 file changed, 20 deletions(-)

diff --git a/arch/arm64/boot/dts/ti/k3-j784s4-vision-apps.dtso b/arch/arm64/boot/dts/ti/k3-j784s4-
vision-apps.dtso
index 83b500405..ba226e32c 100644
--- a/arch/arm64/boot/dts/ti/k3-j784s4-vision-apps.dtso
+++ b/arch/arm64/boot/dts/ti/k3-j784s4-vision-apps.dtso
@@ -10,36 +10,16 @@

 #include <dt-bindings/mux/ti-serdes.h>

-&main_r5fss0_core0_shared_memory_queue_region {
-    status = "disabled";
-};
-
-&main_r5fss0_core0_shared_memory_bufpool_region {
-    status = "disabled";
-};
-
 #include "k3-j784s4-rtos-memory-map.dtsi"

 &main_i2c1 {
     status = "disabled";
 };

-&main_i2c4 {
-    status = "disabled";
-};
-
 &main_i2c5 {
     status = "disabled";
 };

-&dss {
-    status = "disabled";
-};
-
-&serdes_wiz4 {
```

```
-    status = "disabled";
-};
-
 &ti_csi2rx0 {
     status = "disabled";
 };
--
2.34.1
```

3.2.2 Host Setup

The following commands are executed on the host PC.

Install all dependencies.

```
# install the PROCESSOR-SDK-RTOS-J784S4
# install the PROCESSOR-SDK-LINUX-J784S4
# download the data set tar files
# download and untar the patch tars
# insert SD card
```

Export build variables.

```
export PSDKR_PATH=<path-to-rtos-sdk>
export PSDKL_PATH=<path-to-linux-sdk>
export DATA_SET_PATH=<path-to-directory-where-data-sets-are-stored>
export RTOS_PATCHES=<path-to-rtos-patches>
export LINUX_PATCHES=<path-to-linux-patches>
```

Set up the PSDK RTOS.

```
# set up PSDK RTOS
cd $PSDKR_PATH
./sdk_builder/scripts/setup_psdk_rtos.sh
```

Set up the SD card. The example assumes the SD card is at /dev/sdb.

```
umount /dev/sdb?*
cd $PSDKR_PATH
sudo sdk_builder/scripts/mk-linux-card.sh /dev/sdb
./sdk_builder/scripts/install_to_sd_card.sh
cd /media/$USER/rootfs/
mkdir -p opt/vision_apps
cd opt/vision_apps
tar --strip-components=1 -xf $DATA_SET_PATH/psdk_rtos_ti_data_set_09_02_00.tar.gz
tar --strip-components=1 -xf $DATA_SET_PATH/psdk_rtos_ti_data_set_09_02_00_j784s4.tar.gz
sync
```

Edit, build, and install the demo application.

```
# edit and build demo app
cd $PSDKR_PATH/vision_apps
git init
git add -A
git commit -m "SDK 09.02.00.05 release"
git am $RTOS_PATCHES/*.patch
cd ../sdk_builder
./make_sdk.sh
make linux_fs_install_sd
```

Edit, build, and install the device-tree.

```
export PATH=$PATH:$PSDKL_PATH/linux-devkit/sysroots/x86_64-arago-linux/usr/bin/aarch64-oe-linux/
cd $PSDKL_PATH/board-support/ti-linux-kernel-6.1.80+gitAUTOINC+2e423244f8-ti
git add -A
git commit -m "SDK 09.02.00.05 release"
git am $LINUX_PATCHES/*.patch
make ARCH=arm64 CROSS_COMPILE=aarch64-oe-linux- defconfig ti_arm64_prune.config
make ARCH=arm64 CROSS_COMPILE=aarch64-oe-linux- DTC_FLAGS=-@ ti/k3-j784s4-vision-apps.dtbo
sudo mv /media/$USER/rootfs/boot/dtb/ti/k3-j784s4-vision-apps.dtbo /media/$USER/rootfs/boot/dtb/ti/
```

```
k3-j784s4-vision-apps.dtbo.old
sudo cp arch/arm64/boot/dts/ti/k3-j784s4-vision-apps.dtbo /media/$USER/rootfs/boot/dtb/ti/
```

3.2.3 Target Setup

The following commands are executed on the target.

```
cd /opt/vision_apps
source ./vision_apps_init.sh
./run_app_tidl_avp.sh
```

3.2.4 Recreation

The following photo was taken with kmstest running in tandem with the AVP demo.

```
systemctl stop weston
kmstest & ./run_app_tidl_avp.sh
```

The screen suffers sync losts when running the AVP demo. This results in unpredictable artifacts on the display.

Figure 3-1. Sync Losts when Running kmstest and the AVP Demo

3.3 Debugging QoS

3.3.1 CPTracer

CCS (and Lauterbach) offer a tool called CPTracer, which allows developers to profile traffic within the system. In this case, we'll profile the DDR traffic.

3.3.1.1 Setup

1. Follow the steps for setting up CCS within section 7.4. Debugging with HLOS running on A72 (Linux / QNX) of the Processor SDK RTOS.

CAUTION

The specific version of the PSDK RTOS mentioned here is: PSDK RTOS J784S4 10.01.00

2. The traffic profiling window can be opened from the SoC analysis tab.

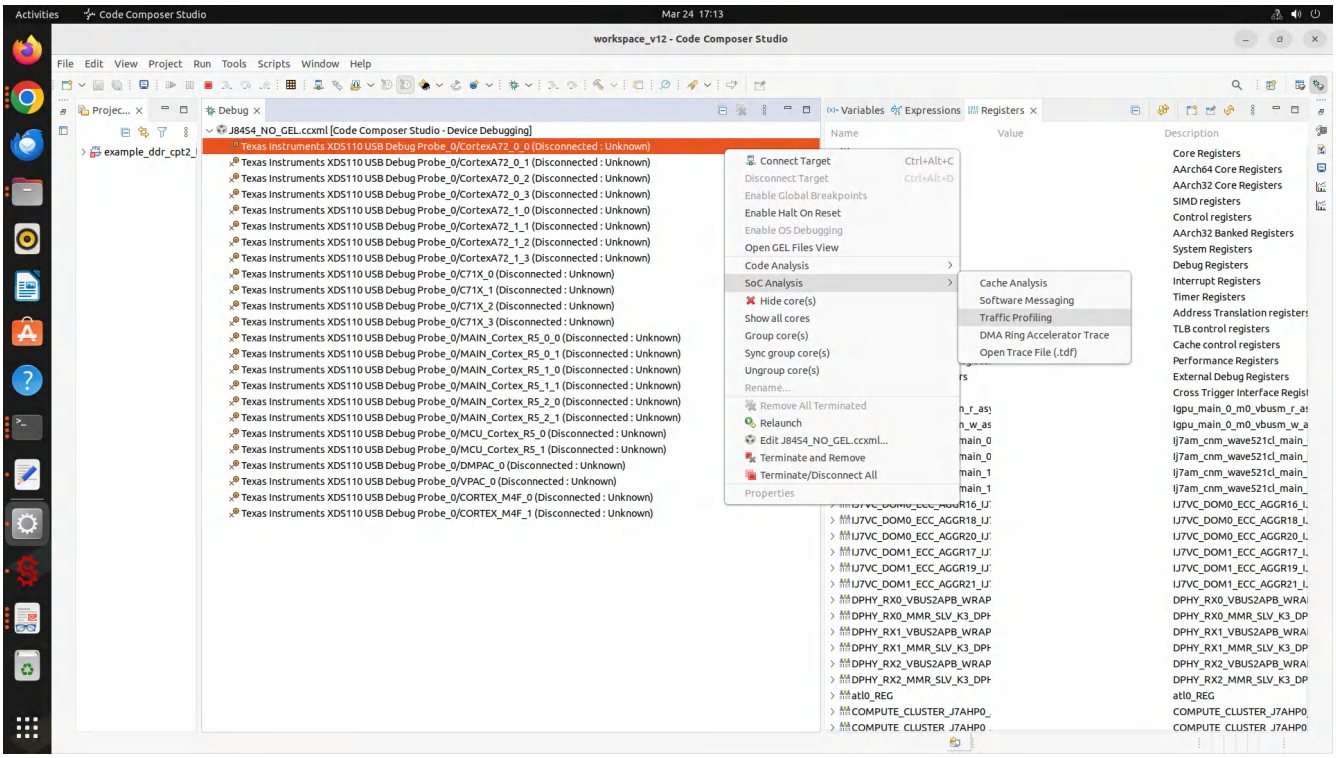

Figure 3-2. Traffic Profiling Selection in CCS

3. This is the window that should appear when opening CPTracer:

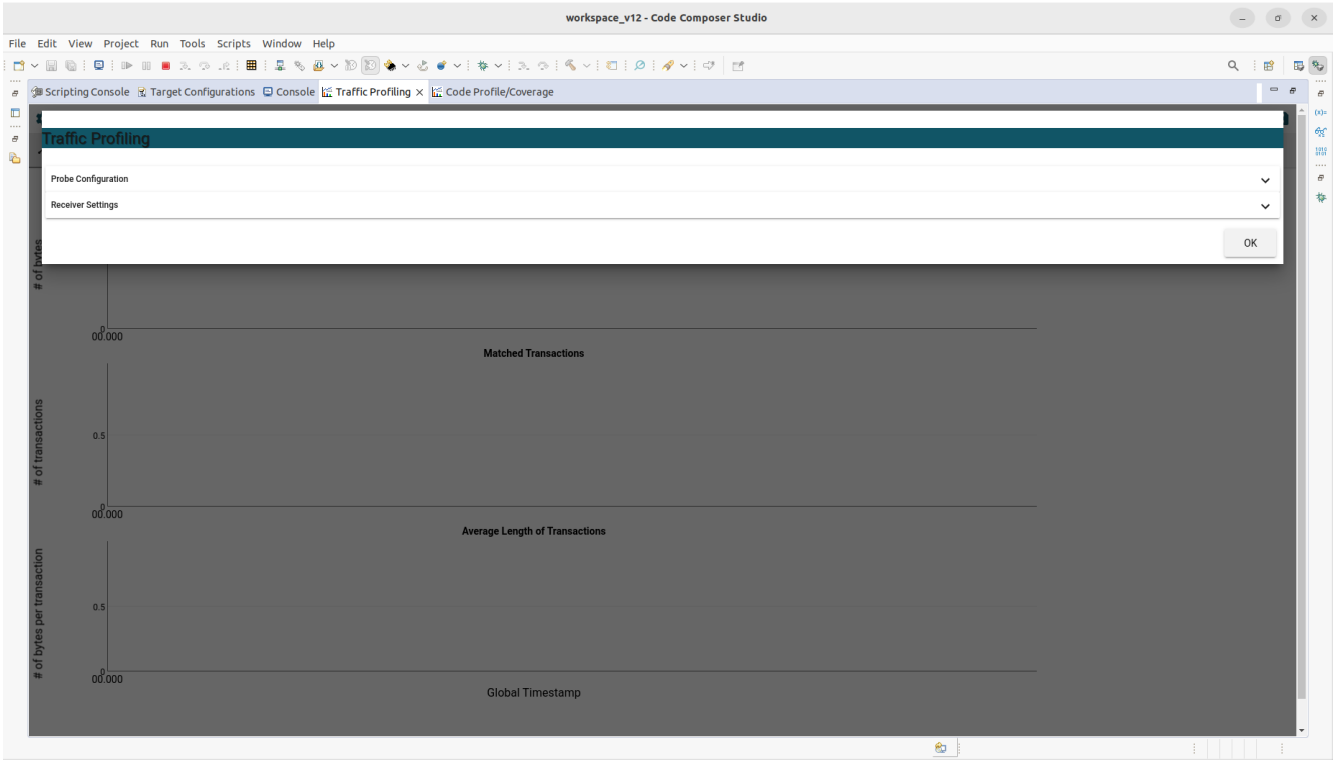

Figure 3-3. CPTracer Settings in CCS

4. There are many filters available to prune which transactions are profiled, and they are editable by clicking the gear icon of a corresponding initiator.

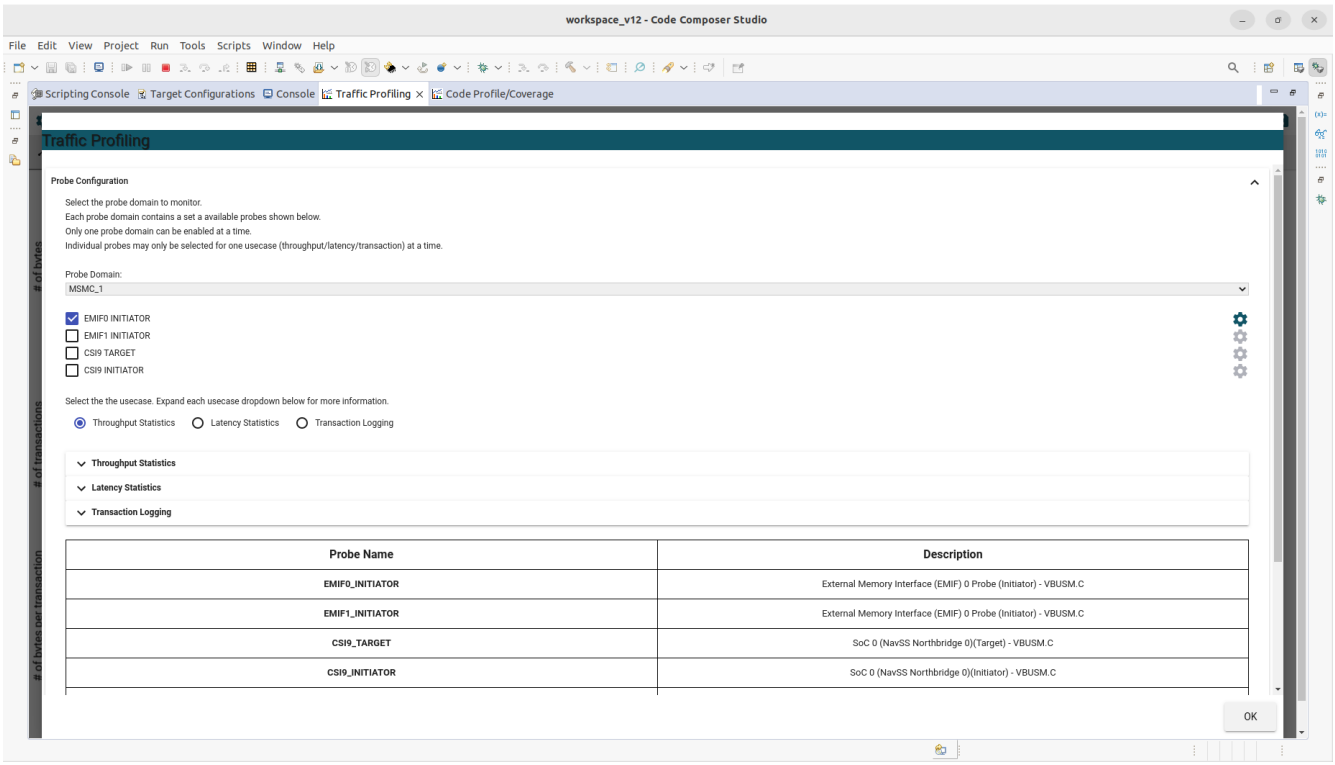

Figure 3-4. CPTracer Filters in CCS

3.3.1.2 Profiling Throughput

CPTracer offers a way to profile the total throughput of transactions within a period. The following is output when a csv is exported:

- **Master ID**: ID of the initiator making the transaction
- **Master Name**: name of the initiator (corresponds with the Master ID)
- **Data Message**: description of the row (contains the period in clock cycles)
- **Global Timestamp**: time of sample window's closing (based on GTC (200MHz clock))
- **Trace Status**: notes when the trace starts and ends
- **Byte Transactions**: total number of bytes (sent within a period) observed at the probe that match the set filters
- **Matched**: total number of transactions (within a period) observed at the probe that match the set filters
- **Avg. Length**: average number of bytes (sent within a period) per transaction observed at the probe that match the set filters

3.3.1.3 Profiling Latency

CPTracer also offers a way to profile the latency of transactions within a period. The following is output when a csv is exported:

- **Master ID**: ID of the initiator making the transaction
- **Master Name**: name of the initiator (corresponds with the Master ID)
- **Data Message**: description of the row (contains the period in clock cycles)
- **Global Timestamp**: time of sample window's closing (based on GTC (200MHz clock))
- **Trace Status**: notes when the trace starts and ends
- **Tracked**: total number of transactions within a period from the initiator
- **Matched Transactions**: total number of transactions (within a period) observed at the probe that match the set filters
- **Max Wait**: maximum latency measurement of a single transaction (within a period) observed at the probe that match the set filters
- **Total Wait**: total latency (all latency measurements summed together) (within a period) observed at the probe that match the set filters
- **Credit Wait**: total credit latency (within a period) observed at the probe for credit-based buses that match the set filters

3.3.1.4 Profiling Transactions

CPTracer gives detailed information on transactions that are observed. The following are output when a csv is exported (this is a pared down list of relevant columns):

- **Route ID**: route ID of the transaction
- **Byte Count**: burst size of the transaction
- **Priority**: the priority of the transaction
- **QOS**: the quality of service of the transaction
- **Order ID**: the order ID of the transaction

Note

The complete list of columns can be found in the Advanced Probe Filters section of the CPTracer documentation

3.3.1.5 Profiling Relevant Routes

The issue that we're viewing is occurring due to the C7x disrupting the DSS, therefore we want to profile the C7x and DSS routes.

The route IDs for DSS and C7x transactions are as follows:

Table 3-2. Relevant Initiators and Route IDs

Initiator	Route ID
C7x_1 Core	0x20

Table 3-2. Relevant Initiators and Route IDs (continued)

Initiator	Route ID
C7x_1 DRU0	0x21
C7x_1 DRU1	0x22
C7x_1 CMMU	0x23
C7x_2 Core	0x24
C7x_2 DRU0	0x25
C7x_2 DRU1	0x26
C7x_2 CMMU	0x27
C7x_3 Core	0x28
C7x_3 DRU0	0x29
C7x_3 DRU1	0x2A
C7x_3 CMMU	0x2B
C7x_4 Core	0x2C
C7x_4 DRU0	0x2D
C7x_4 DRU1	0x2E
C7x_4 CMMU	0x2F
DSS_INST0_VBUSM_DMA	0xA20
DSS_INST0_VBUSM_FBDC	0xA21

Note

Route IDs can be found within the appendixes of the TDA4VH TRM

The CCS version of CPTracer only contains EMIF0 and EMIF1 (missing EMIF2 and 3) within the MSMC_1 probe domain, but you're able to approximate the total throughput by multiplying one EMIF's throughput by four.

3.3.1.6 Profiling DSS Throughput

The filters used to profile only DSS transactions are the following:

- **Route ID Value**: 0xA20
- **Route ID Mask**: 0xFFE
- **Sampling Window**: 0x4000

Only EMIF0 was profiled.

The total bytes sent per frame were calculated with: **dss-frame-thru-calc.py**

```
# Author: Jared McArthur

import csv
import matplotlib.pyplot as plt
from argparse import ArgumentParser
from textwrap import dedent

def main():
    total_bytes = []
    time_stamps = []

    parser = ArgumentParser(prog="dss-frame-thru-calc.py")
    parser.add_argument("file", type=str)
    parser.add_argument("frame_start", type=float)
    parser.add_argument("frame_end", type=float)

    args = parser.parse_args()
    start = args.frame_start
```

```
    end = args.frame_end

    with open(args.file, "r") as csvfile:
        data = csv.DictReader(csvfile)

        first_stamp = 0

        for row in data:
            if row.get("Master ID") != "":
                total_byte = int(row.get("Byte Transactions"), base=16)
                time_stamp = int(row.get("Global Timestamp"), base=16)

                if len(time_stamps) == 0:
                    first_stamp = time_stamp

                total_bytes.append(total_byte)
                time_stamps.append((time_stamp - first_stamp) / (1000000000 / 5))

    stamps_len = len(time_stamps)
    for index, stamp in enumerate(reversed(time_stamps)):
        if stamp < start or stamp > end:
            time_stamps.pop(stamps_len - 1 - index)
            total_bytes.pop(stamps_len - 1 - index)

    bytes_in_frame = 0
    for val in total_bytes:
        bytes_in_frame += val

    print(dedent(f"""
        Num periods in segment: {len(time_stamps)}
        Time elapsed in segment: {time_stamps[-1] - time_stamps[0]}
        Bytes sent in segment: {bytes_in_frame}"""))

    plt.plot(time_stamps, total_bytes)
    plt.show()

if __name__ == "__main__":
    main()
```

3.3.1.6.1 Theoretical DSS Throughput

The theoretical DSS throughput (**265420800** bits per frame) is calculated for a **3840x2160@30fps XR32-888** display.

Table 3-3. Theoretical DSS Throughput

Parameter	Value
Height	3840
Width	2160
fps	30
Bits per Pixel	32
Bits per Frame	265420800
Mib per Frame	253.125
Data Rate (bps)	7962624000
Data Rate (Mibps)	7593.75

3.3.1.6.2 Normal DSS Throughput

The following image displays the throughput of DSS transactions without the AVP demo running.

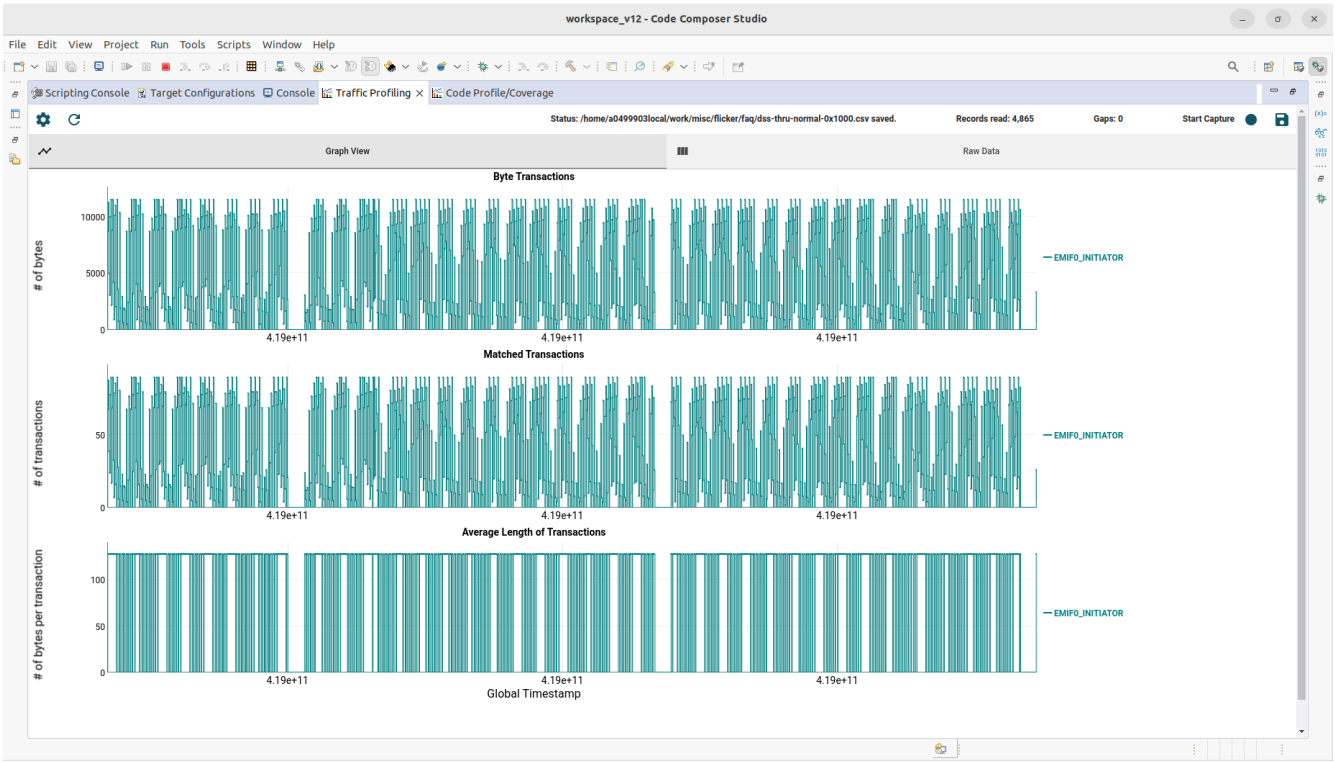

Figure 3-5. Normal DSS Throughput

The throughput measurements were taken by adding up all of periods' throughputs that fell within a single frame. The measurements were gathered by saving the throughput CSV and feeding them through a python script.

Table 3-4. Normal Measured DSS Throughput

	DSS Throughput of 1 EMIF	Corrected Throughput for 4 EMIFs
Bytes per Frame	8294400	33177600
Bits per Byte	8	
fps	30	
Number of EMIFs Measured	1	4
Bits per Frame	66355200	265420800
Mib per Frame	63.28125	253.125
Data Rate (bps)	1990656000	7962624000
Data Rate (Mibps)	1898.4375	7593.75

The total throughput of a single frame matches 1/4 of the expected throughput for a **3840x2160@30fps** screen (**66355200** bits per frame). Only one out of four EMIFs was profiled, so this is the expected result. Once the measurement is corrected, the throughput equals the theoretical throughput (**265420800** bits per frame).

3.3.1.6.3 DSS Throughput with the AVP Demo Running

The following image displays the throughput of DSS transactions with the AVP demo running.

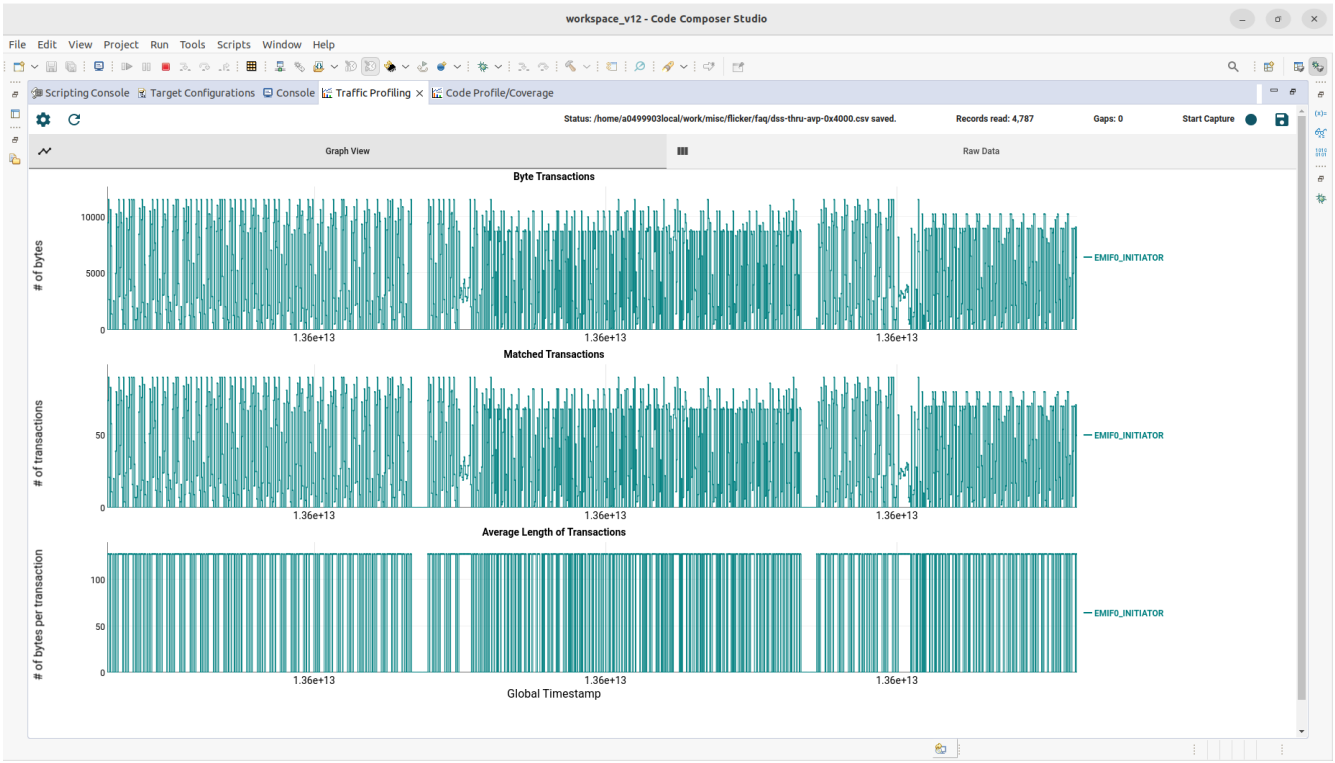

Figure 3-6. AVP Demo DSS Throughput

Table 3-5. Measured DSS Throughput with AVP Demo

	DSS Throughput of 1 EMIF	Corrected Throughput for 4 EMIFs
Bytes per Frame	8257536	33030144
Bits per Byte	8	
fps	30	
Number of EMIFs Measured	1	4
Bits per Frame	66060288	264241152
Mib per Frame	63	252
Data Rate (bps)	1981808640	7927234560
Data Rate (Mibps)	1890	7560

The total throughput falls short of what is required to display **3840x2160@30fps**. This is what causes the sync losts. Only **264241152** bits were sent in a frame instead of **265420800** bits.

3.3.1.7 Profiling DSS Latency

Hypothetically, the DSS transactions are being stalled and causing a large spike in the total latency. To verify this hypothesis, we profile the latency of the DSS with and without the AVP demo running.

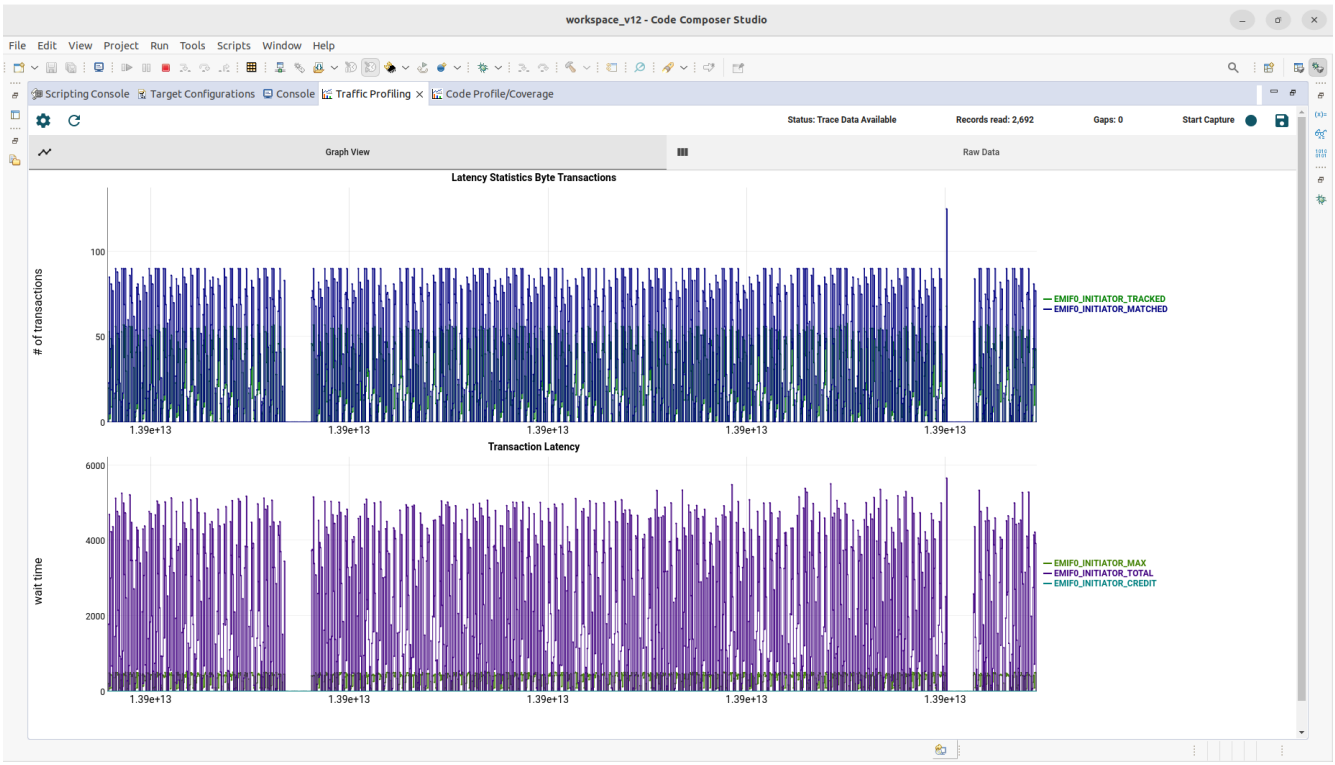

Figure 3-7. Normal DSS Latency

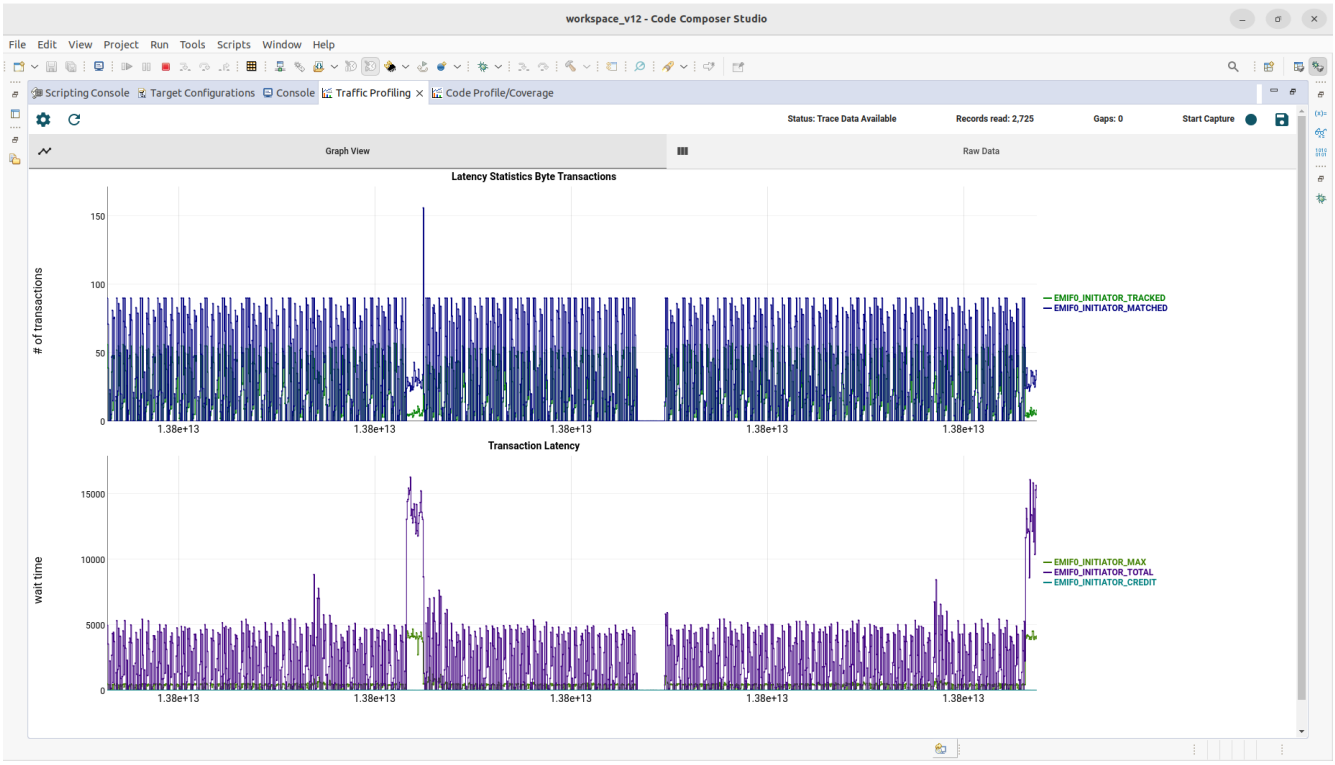

Figure 3-8. AVP Demo DSS Latency

Looking at the latency graphs, it's obvious that the DSS transactions are being stalled. We can now profile the C7x transactions and try to narrow down the source of the stalls.

3.3.1.8 Profiling C7x Throughput

Profiling the throughput of each separate C7x core can show what exactly is causing the DSS stalls.

The high throughput and high number of transactions sections can be causing the stalls.

Table 3-6 contains the settings required to filter for the separate C7x cores.

Table 3-6. C7x Route ID Values and Masks

C7x Core	Route IDs	Route ID Value	Route ID Mask
All cores	0x20 to 0x2F	0x020	0xFF0
1	0x20 to 0x23	0x020	0xFFC
2	0x24 to 0x27	0x024	0xFFC
3	0x28 to 0x2B	0x028	0xFFC
4	0x2C to 0x2F	0x02C	0xFFC

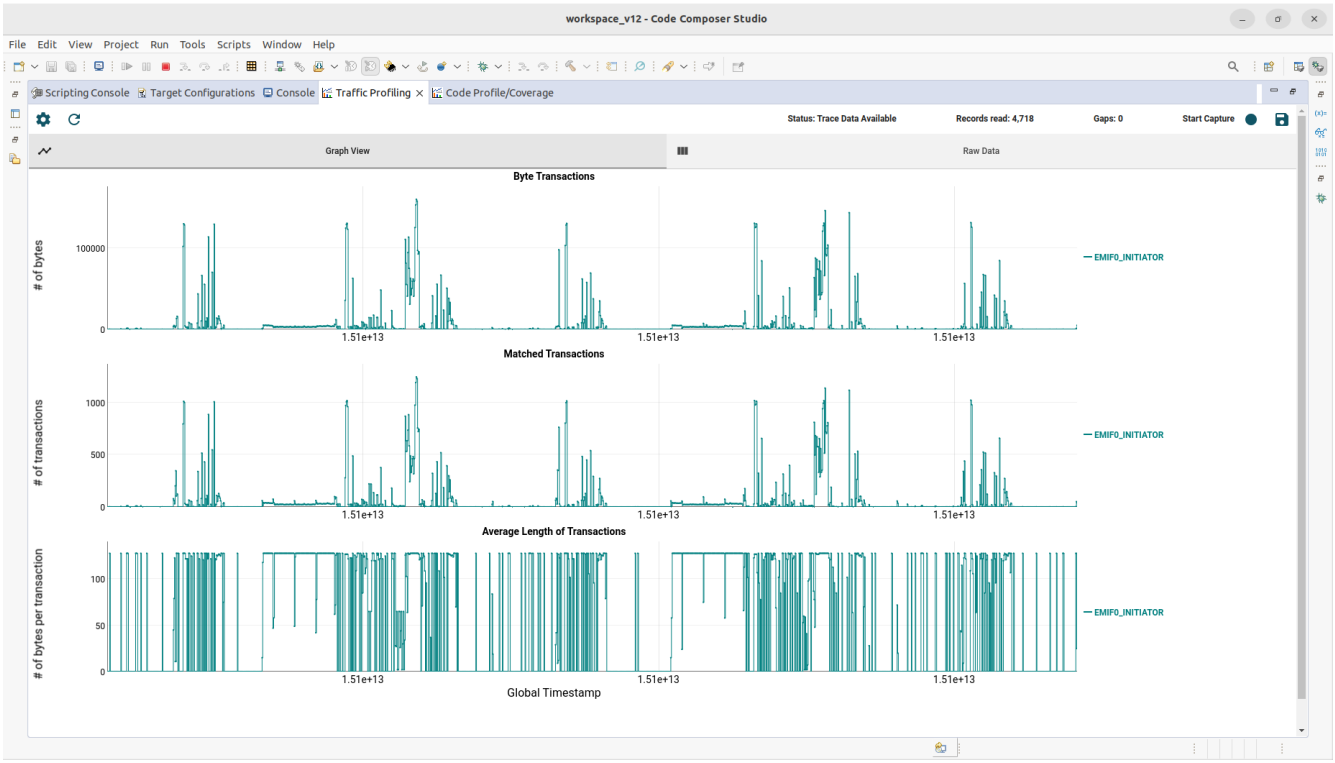

Figure 3-9. All C7x Throughput

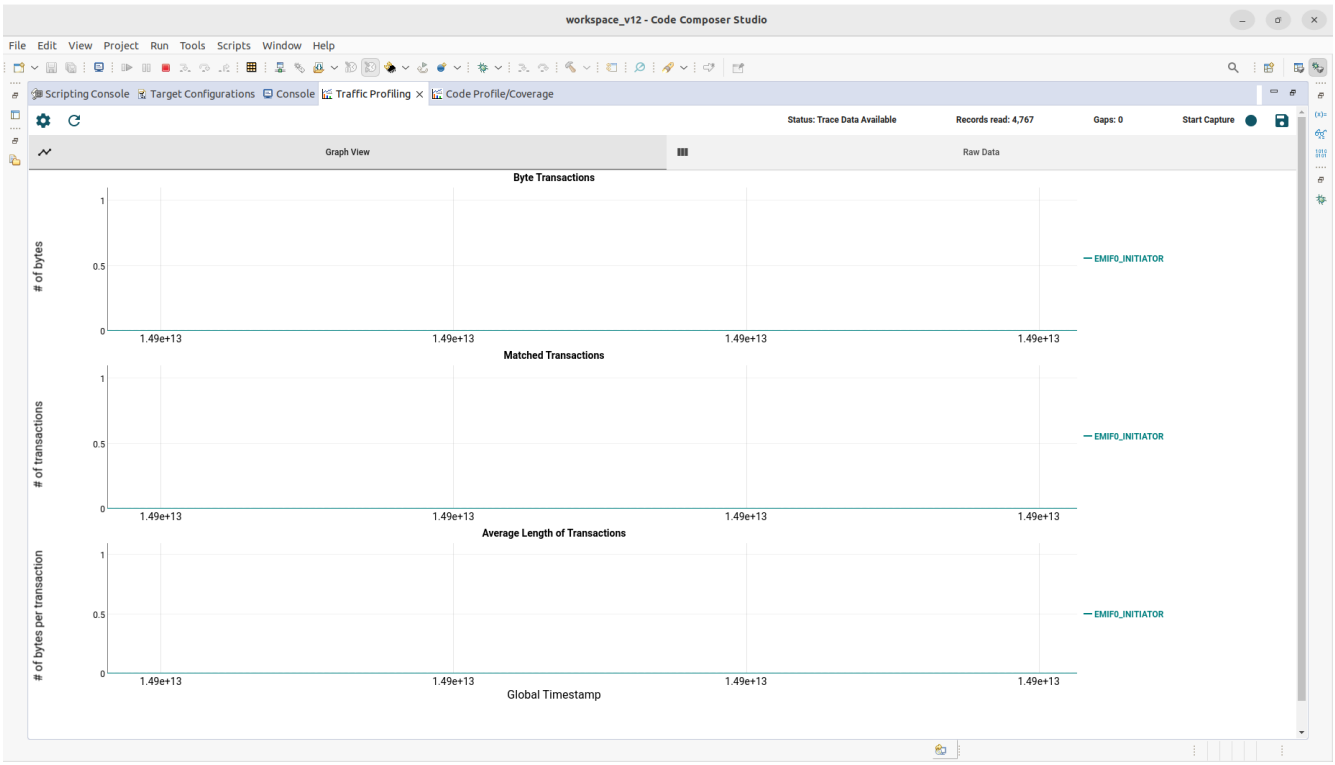

Figure 3-10. C7x_1 Throughput

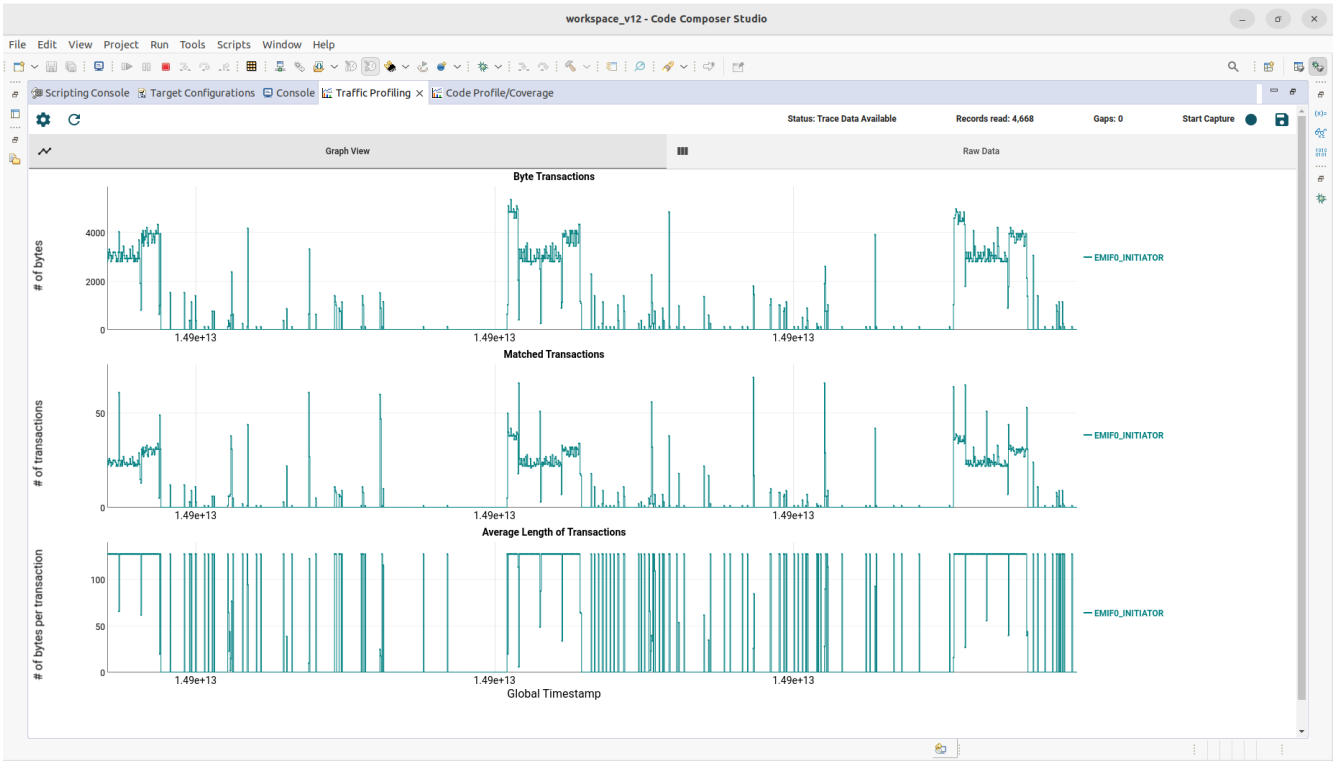

Figure 3-11. C7x_2 Throughput

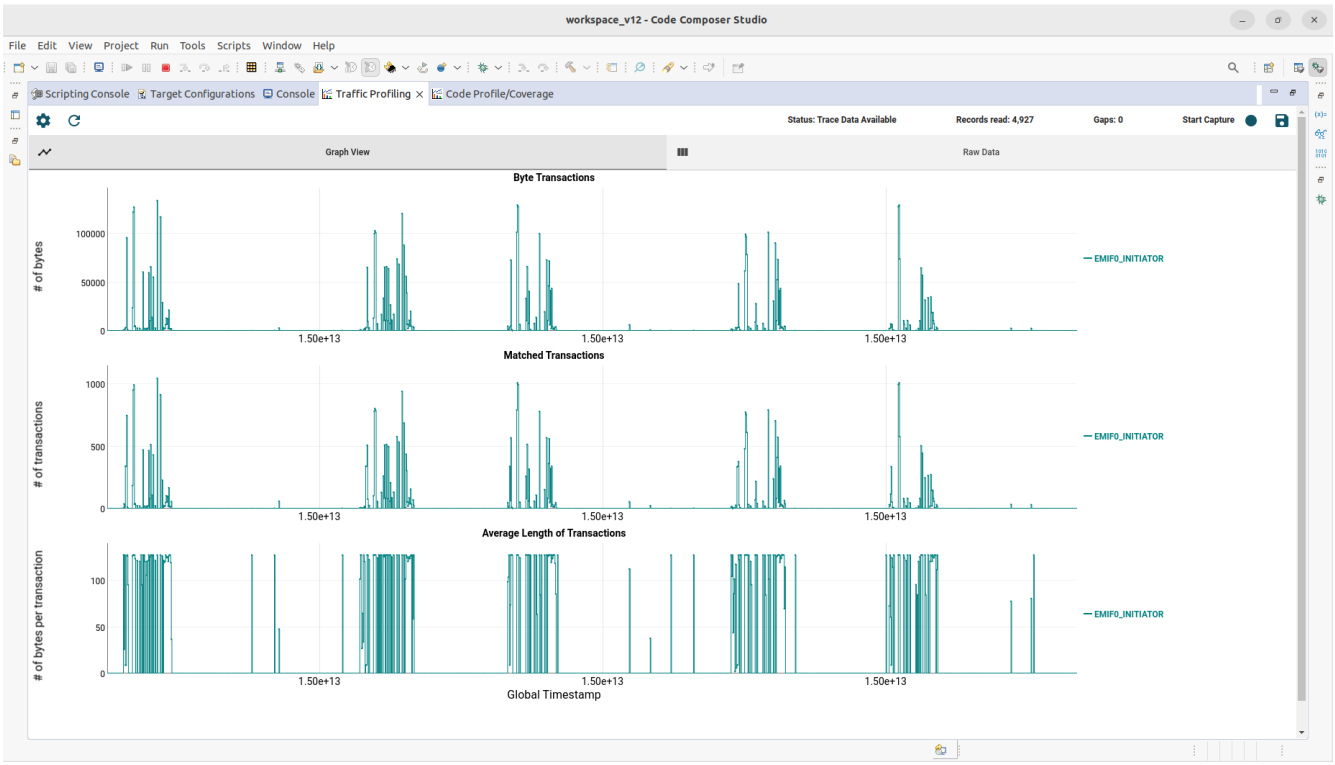

Figure 3-12. C7x_3 Throughput

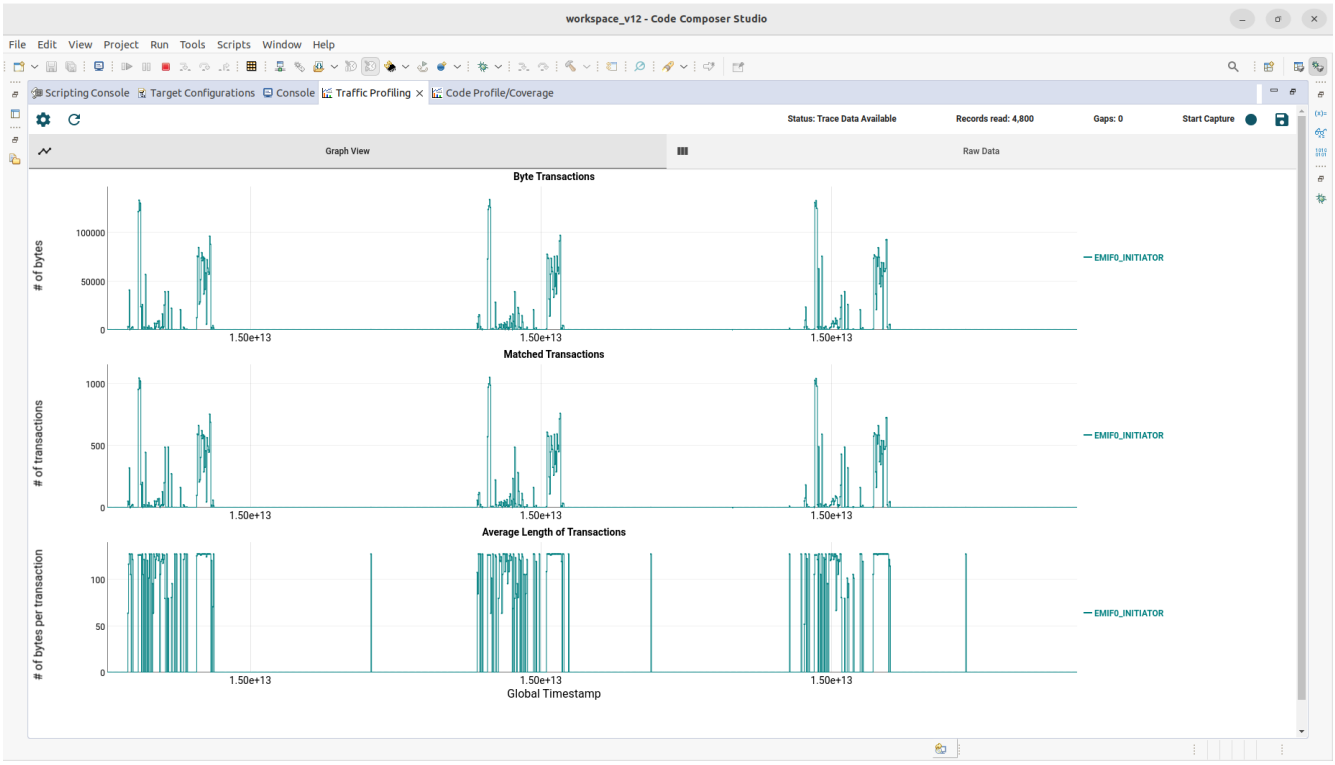

Figure 3-13. C7x_4 Throughput

Looking at the above plots, it appears that C7x_4 is causing the stalls in the DSS. The high throughput sections appear to inversely match when the DSS throughput pattern.

Because it appears that the DSS is stalling due to the high throughput transactions sent from the C7x_4, let's take a closer look at C7x_4.

3.3.1.9 Profiling C7x Throughput vs DSS Latency

Unfortunately, CCS' CPTracer doesn't allow you to visualize transaction throughputs and transaction latencies at the same time. Lauterbach's CPTracer does, however.

You can profile throughput on EMIF0 and latency on EMIF1 to view both the throughput and latency (of the same or different routes) at the same time.

Using this method, we can verify which specific C7x_4 route has a non-zero throughput while the DSS latency is high and DSS throughput is bottlenecked.

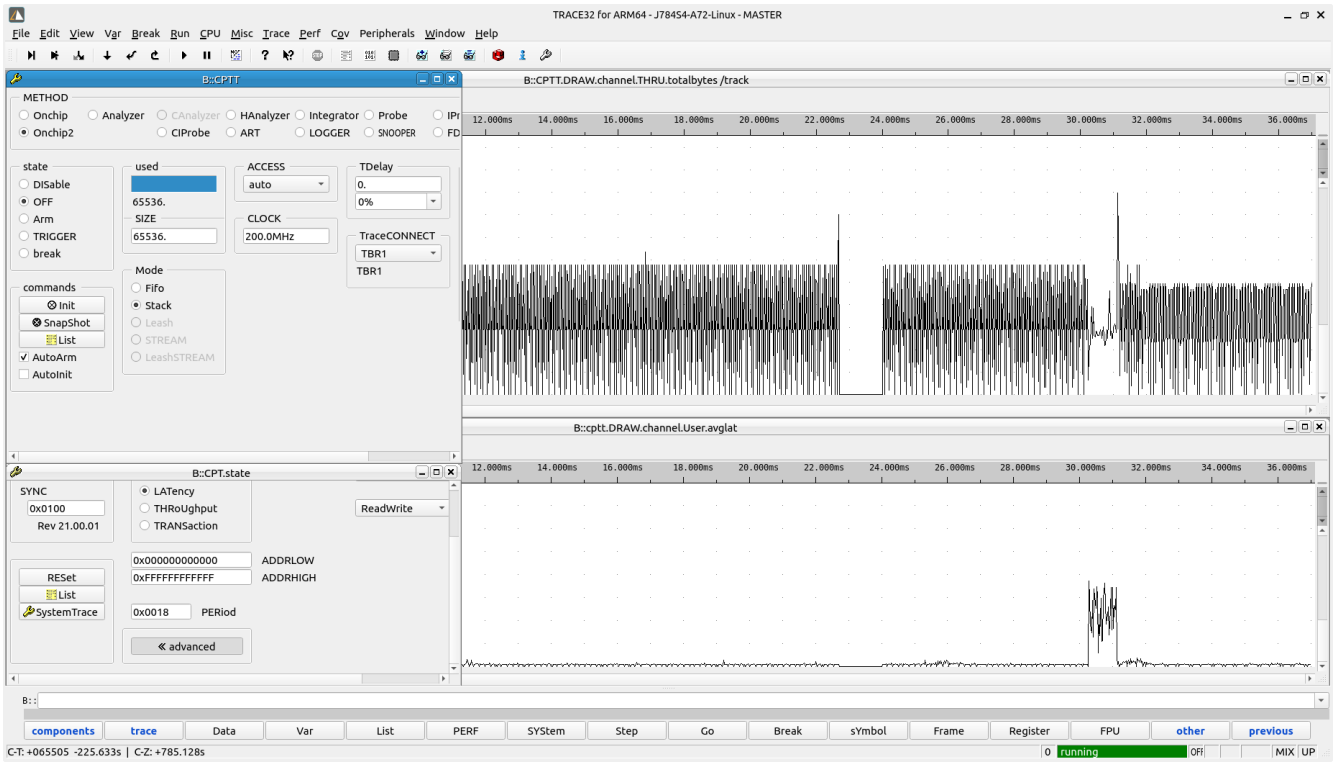

Figure 3-14. DSS Throughput vs DSS latency

C7x_4 core (route 0x2C) transactions' compared to DSS transactions' latency.

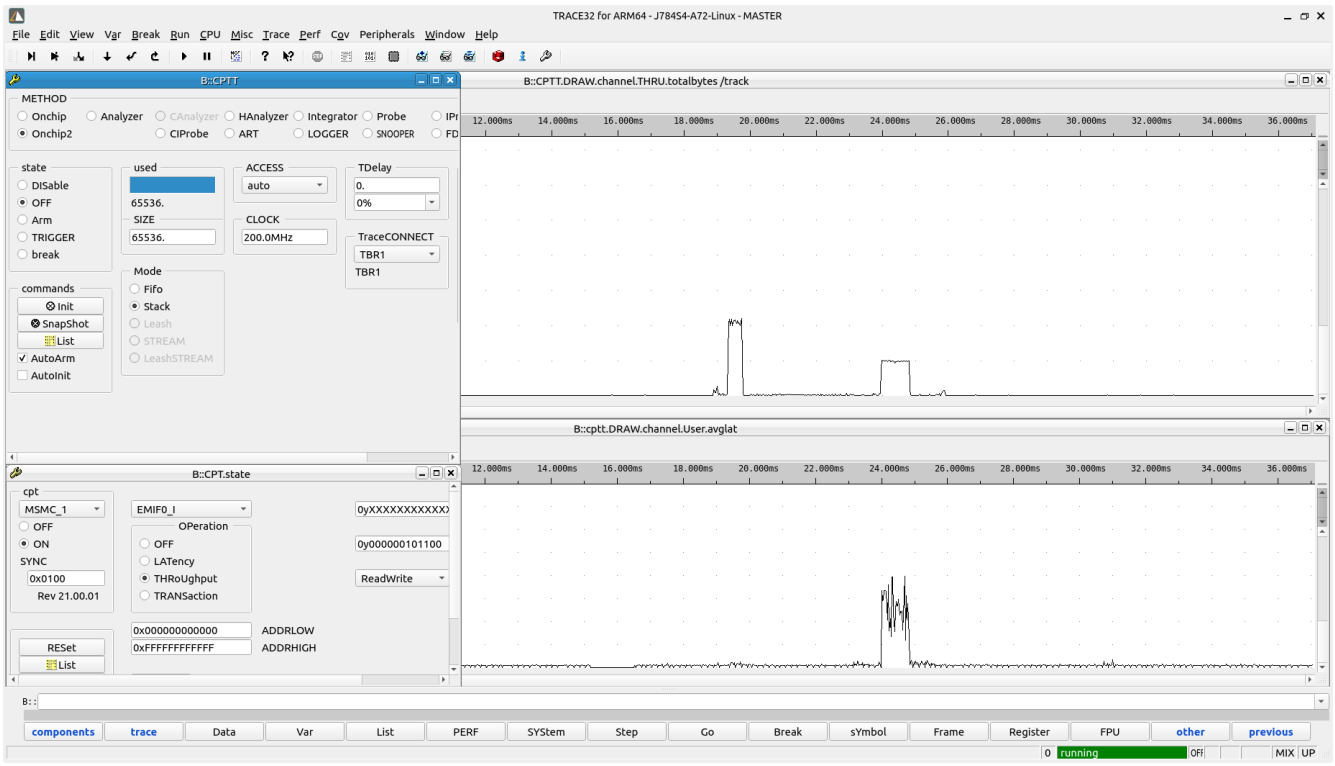

Figure 3-15. C7x_4 Core Throughput vs DSS Latency

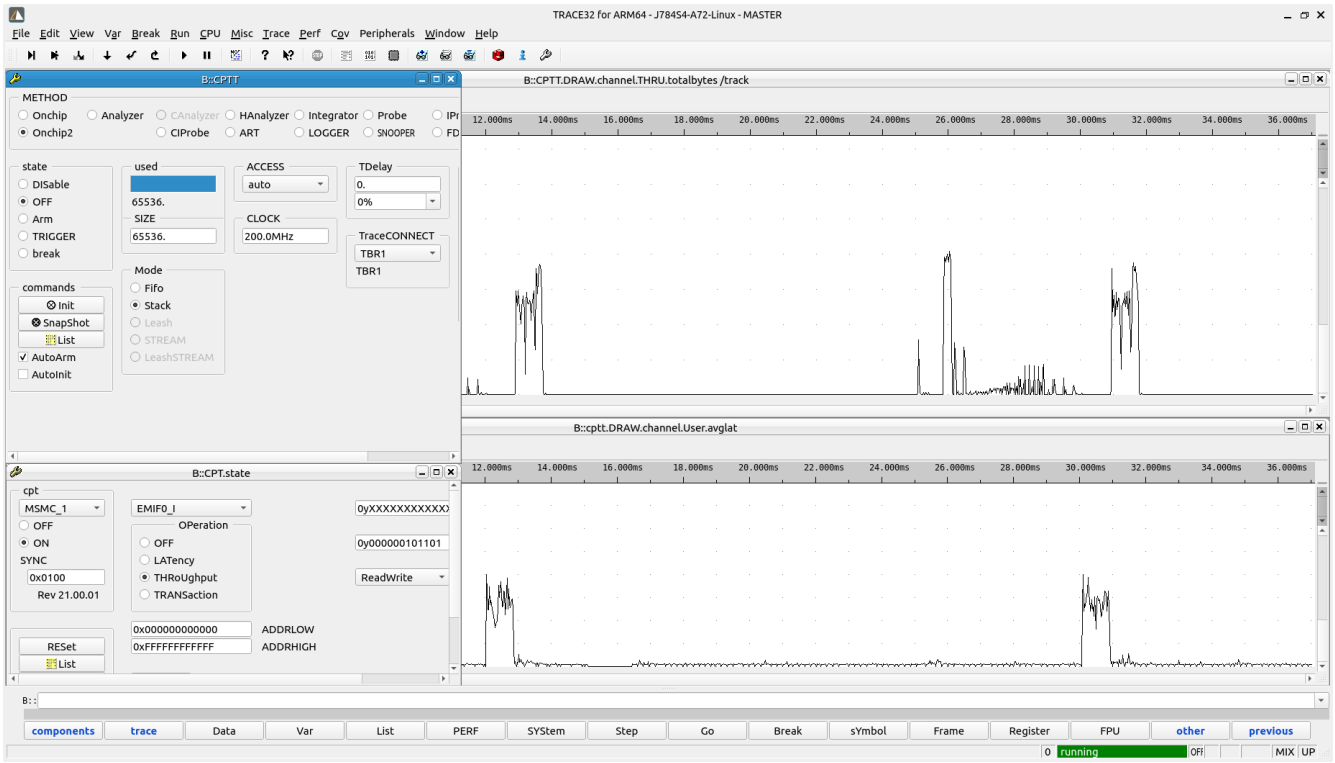

Figure 3-16. C7x_4 DRU0 Throughput vs DSS Latency

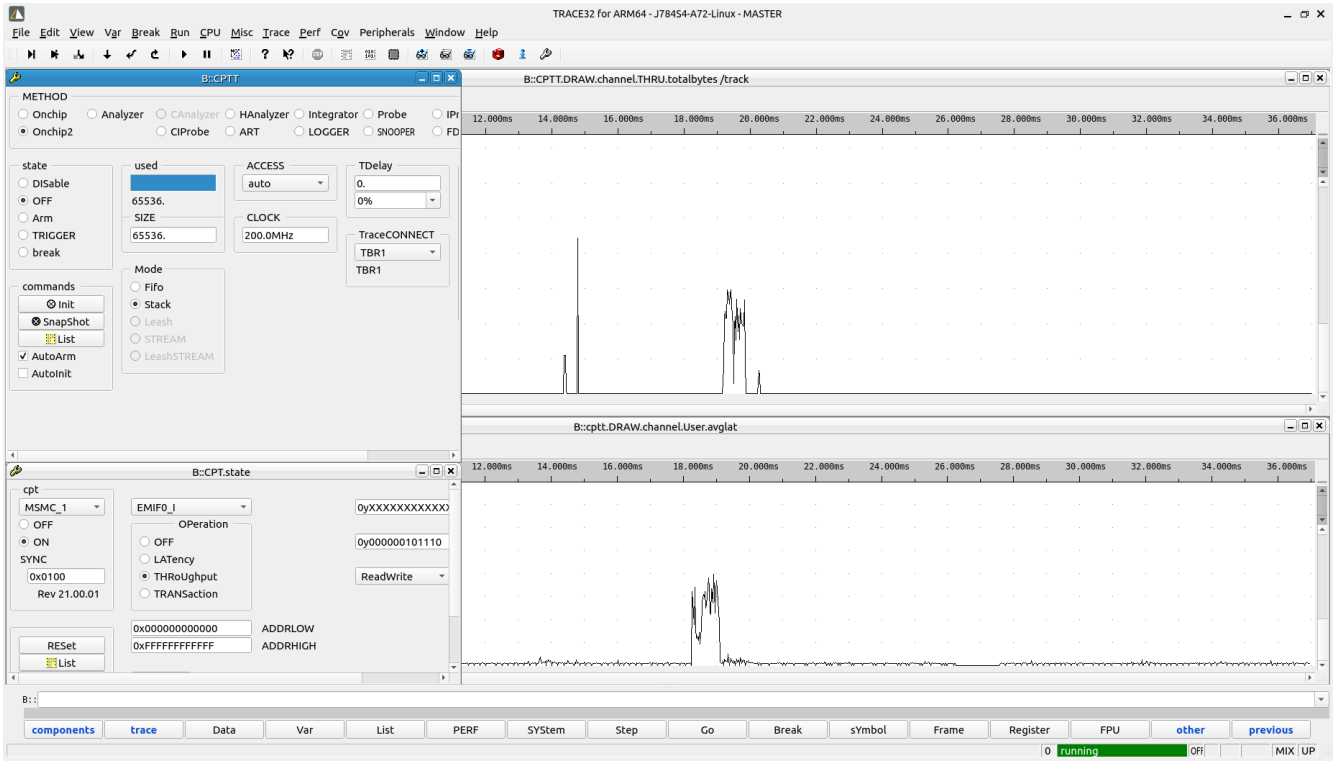

Figure 3-17. C7x_4 DRU1 Core Throughput vs DSS Latency

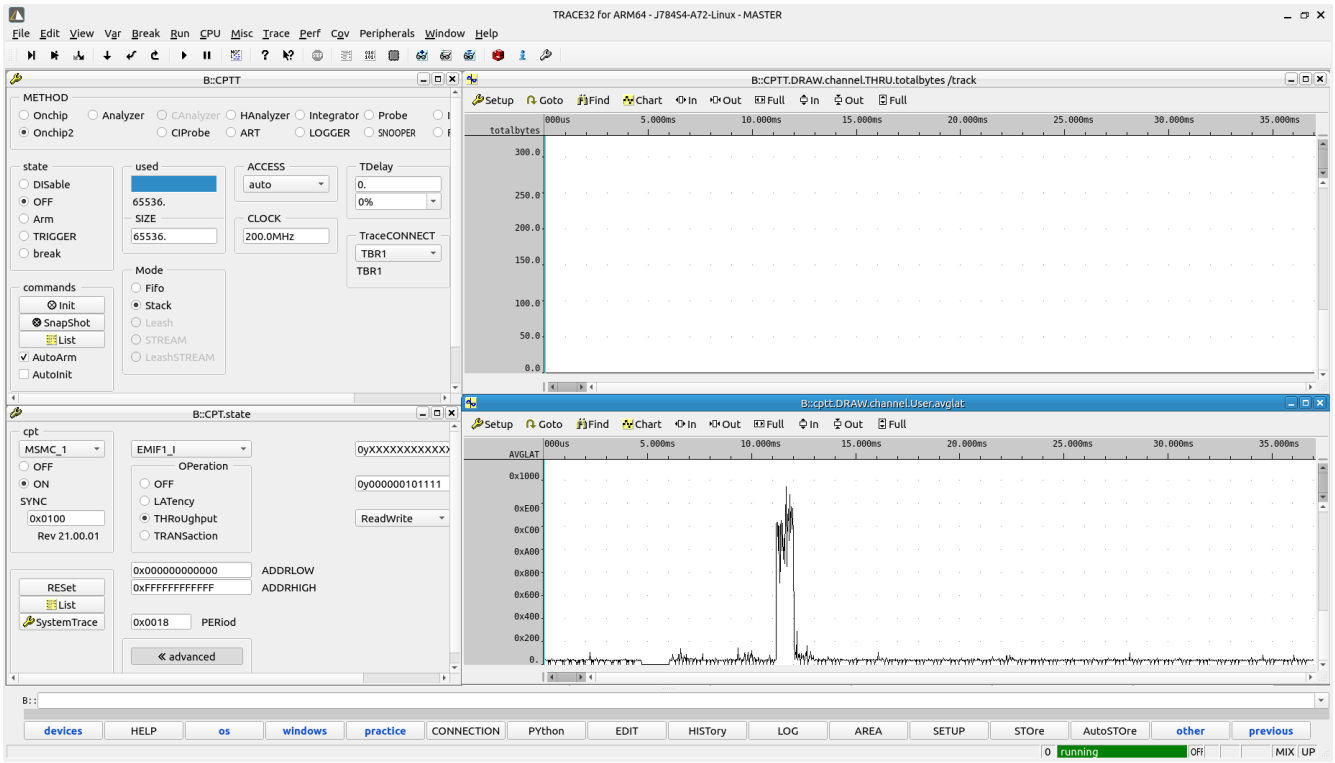

Figure 3-18. C7x_4 CMMU Throughput vs DSS Latency

Looking at the previous plots, the issue appears to be with the C7x_4 core (0x2D) transactions.

3.3.1.10 Profiling C7x_4 Core Transactions

Using CPTracer's transaction profiling capabilities, we can look at the QoS settings for the C7x_4 core and DSS transactions.

C7x_4 core transactions:

- **Route ID**: 0x002D
- **Priority**: 0x03
- **QoS**: 0x00
- **Order ID**: 0x00

DSS transactions:

- **Route ID**: 0xA20
- **Priority**: 0x00 or 0x01
- **QoS**: 0x00
- **Order ID**: 0x0F

Looking at the priorities, the C7x_4 core transactions must not be superseding the DSS transactions.

3.3.2 Editing QoS Settings

Although all of the QoS settings appear to be correct, for posterity's sake, we can work on how to set and change them. Users also need to verify that the DSS transactions are being routed to the RT thread instead of the NRT thread.

After setting the relevant QoS settings, a user can verify changes with the transaction profiling feature of CPTracer.

Note

Code to edit QoS settings can be generated with the **Keystone3 Resource Partitioning Tool** that comes packaged with SYSCONFIG.

3.3.2.1 Editing Order ID

Changing the order ID of specific transactions allows you to route them to the NRT or RT thread. This is done within CBASS configuration registers and within the independent IPs (depending on the IP).

3.3.2.1.1 DSS Order ID

The order ID for the DSS is set within the CBASS configuration registers. There are registers to control the order ID for each channel within the DSS and registers to map those order IDs to VBUSM order IDs. The DSS contains both a primary port (signified with "DMA") and secondary port (signified with "frame buffer decompression (FBDC)"). If compression is enabled for a particular channel, then all transactions of that channel go through the secondary port.

Table 3-7. DSS Order ID Registers

Register	Address	Bits	Field	Description
CBASS_AC_NONSAFE_QOS_lk3_dss_main_0_dss_inst0_vbusm_dma_slv_linkgrp_1_grp_map1	0x45DC2000	N/A	N/A	The Group Map Register defines the final orderid for the initiator lk3_dss_main_0.dss_inst0_vbusm_dma for group slv_linkgrp_1.
		31:28	ORDERID7	Order ID signal for 7
		27:24	ORDERID6	Order ID signal for 6
		23:20	ORDERID5	Order ID signal for 5
		19:16	ORDERID4	Order ID signal for 4
		15:12	ORDERID3	Order ID signal for 3
		11:8	ORDERID2	Order ID signal for 2
		7:4	ORDERID1	Order ID signal for 1
		3:0	ORDERID0	Order ID signal for 0
CBASS_AC_NONSAFE_QOS_lk3_dss_main_0_dss_inst0_vbusm_dma_slv_linkgrp_1_grp_map2	0x45DC2004	N/A	N/A	The Group Map Register defines the final orderid for the initiator lk3_dss_main_0.dss_inst0_vbusm_dma for group slv_linkgrp_1.
		31:28	ORDERID15	Order ID signal for 15
		27:24	ORDERID14	Order ID signal for 14
		23:20	ORDERID13	Order ID signal for 13
		19:16	ORDERID12	Order ID signal for 12
		15:12	ORDERID11	Order ID signal for 11
		11:8	ORDERID10	Order ID signal for 10
		7:4	ORDERID9	Order ID signal for 9
		3:0	ORDERID8	Order ID signal for 8

Table 3-7. DSS Order ID Registers (continued)

Register	Address	Bits	Field	Description
CBASS_AC_NONSAFE_QOS_lk3_dss_main_0_dss_inst0_vbusm_dma_map x	0x45DC2100 + (x * 4); where x = 0 to 9	7:4	ORDERID	Order ID signal for channel x. Selects route for load balancing (0-7 uses one route, 8-15 another). Also used by DDR4/ LPDDR4 re-ordering to maximize throughput. Order of transactions is only guaranteed with the same order ID
CBASS_AC_NONSAFE_QOS_lk3_dss_main_0_dss_inst0_vbusm_fbdc_slv_linkgrp_1_grp_map1	0x45DC2404	N/A	N/A	The Group Map Register defines the final orderid for the initiator lk3_dss_main_0.dss_inst0_vbusm_fbdc for group slv_linkgrp_1.
		31:28	ORDERID7	Order ID signal for 7
		27:24	ORDERID6	Order ID signal for 6
		23:20	ORDERID5	Order ID signal for 5
		19:16	ORDERID4	Order ID signal for 4
		15:12	ORDERID3	Order ID signal for 3
		11:8	ORDERID2	Order ID signal for 2
		7:4	ORDERID1	Order ID signal for 1
		3:0	ORDERID0	Order ID signal for 0
CBASS_AC_NONSAFE_QOS_lk3_dss_main_0_dss_inst0_vbusm_fbdc_slv_linkgrp_1_grp_map2	0x45DC2408	N/A	N/A	The Group Map Register defines the final orderid for the initiator lk3_dss_main_0.dss_inst0_vbusm_fbdc for group slv_linkgrp_1.
		31:28	ORDERID15	Order ID signal for 15
		27:24	ORDERID14	Order ID signal for 14
		23:20	ORDERID13	Order ID signal for 13
		19:16	ORDERID12	Order ID signal for 12
		15:12	ORDERID11	Order ID signal for 11
		11:8	ORDERID10	Order ID signal for 10
		7:4	ORDERID9	Order ID signal for 9
		3:0	ORDERID8	Order ID signal for 8

Table 3-7. DSS Order ID Registers (continued)

Register	Address	Bits	Field	Description
CBASS_AC_NONSAFE_QOS_lk3_dss_main_0_dss_inst0_vbusm_fbdc_map x	0x45DC2500 + (x * 4); where x = 0 to 9	7:4	ORDERID	Order ID signal for channel x. Selects route for load balancing (0-7 uses one route, 8-15 another). Also used by DDR4/ LPDDR4 re-ordering to maximize throughput. Order of transactions is only guaranteed with the same order ID

The code to set the order IDs of the DSS channels is found in the ti-u-boot repository at arch/arm/mach-k3/r5/j784s4/j784s4_qos_uboot.c. This file sets general QoS settings and can be generated with SYSCONFIG.

The excerpt contains the settings for the video plane 1 (VID1), but the file also contains the settings for video plane 2 (VID2), video lite plane 1 (VIDL1), and video lite plane 2 (VIDL2). Each plane is associated with two channels.

```
...

struct k3_qos_data qos_data[] = {
        /* DSS_PIPE_VID1 - 2 endpoints, 2 channels */
        {
                .reg = K3_QOS_REG(K3_DSS_MAIN_0_DSS_INST0_VBUSM_DMA, 0),
                .val = K3_QOS_VAL(0, 15, 0, 0, 0, 0),
        },
        {
                .reg = K3_QOS_REG(K3_DSS_MAIN_0_DSS_INST0_VBUSM_DMA, 1),
                .val = K3_QOS_VAL(0, 15, 0, 0, 0, 0),
        },
        {
                .reg = K3_QOS_REG(K3_DSS_MAIN_0_DSS_INST0_VBUSM_FBDC, 0),
                .val = K3_QOS_VAL(0, 15, 0, 0, 0, 0),
        },
        {
                .reg = K3_QOS_REG(K3_DSS_MAIN_0_DSS_INST0_VBUSM_FBDC, 1),
                .val = K3_QOS_VAL(0, 15, 0, 0, 0, 0),
        },

...

        /* Following registers set 1:1 mapping for orderID MAP1/MAP2
         * remap registers. orderID x is remapped to orderID x again
         * This is to ensure orderID from MAP register is unchanged
         */

        /* K3_DSS_MAIN_0_DSS_INST0_VBUSM_DMA - 1 groups */
        {
                .reg = K3_QOS_GROUP_REG(K3_DSS_MAIN_0_DSS_INST0_VBUSM_DMA, 0),
                .val = K3_QOS_GROUP_DEFAULT_VAL_LOW,
        },
        {
                .reg = K3_QOS_GROUP_REG(K3_DSS_MAIN_0_DSS_INST0_VBUSM_DMA, 1),
                .val = K3_QOS_GROUP_DEFAULT_VAL_HIGH,
        },

        /* K3_DSS_MAIN_0_DSS_INST0_VBUSM_FBDC - 1 groups */
        {
                .reg = K3_QOS_GROUP_REG(K3_DSS_MAIN_0_DSS_INST0_VBUSM_FBDC, 0),
                .val = K3_QOS_GROUP_DEFAULT_VAL_LOW,
        },
        {
                .reg = K3_QOS_GROUP_REG(K3_DSS_MAIN_0_DSS_INST0_VBUSM_FBDC, 1),
                .val = K3_QOS_GROUP_DEFAULT_VAL_HIGH,
        },
```

```
...
```

K3_QOS_REG, K3_QOS_VAL, K3_QOS_GROUP_REG, and K3_QOS_GROUP_DEFAULT_VAL_LOW/HIGH are defined in arch/arm/mach-k3/include/mach/k3-qos.h:

```
...

/* K3_QOS_REG: Registers to configure the channel for a given endpoint */

#define K3_QOS_REG(base_reg, i)     (base_reg + 0x100 + (i) * 4)

#define K3_QOS_VAL(qos, orderid, asel, epriority, virtid, atype) \
    (qos        << 0  | \
     orderid    << 4  | \
     asel       << 8  | \
     epriority  << 12 | \
     virtid     << 16 | \
     atype      << 28)

/*
 * K3_QOS_GROUP_REG: Registers to set 1:1 mapping for orderID MAP1/MAP2
 * remap registers.
 */
#define K3_QOS_GROUP_REG(base_reg, i)   (base_reg + (i) * 4)

#define K3_QOS_GROUP_DEFAULT_VAL_LOW    0x76543210
#define K3_QOS_GROUP_DEFAULT_VAL_HIGH   0xfedcba98
struct k3_qos_data {
    u32 reg;
    u32 val;
};

...
```

3.3.2.1.2 C7x Order ID

The order ID of each C7x is configurable using the DRU queue configuration registers.

Table 3-8. C7x Order ID Registers

C7x Subsystem	Address	Register	Bits	Field	Description
COMPUTE_CLUSTER0_C71SS0	0x68A08000 + (j * 8); where j = 0 to 6	DRU_QUEUE_cfg_j	7:4	ORDERID	This configures the order ID for QUEUE.
COMPUTE_CLUSTER0_C71SS1	0x69A08000 + (j * 8); where j = 0 to 6	DRU_QUEUE_cfg_j	7:4	ORDERID	This configures the order ID for QUEUE.
COMPUTE_CLUSTER0_C71SS2	0x6AA08000 + (j * 8); where j = 0 to 6	DRU_QUEUE_cfg_j	7:4	ORDERID	This configures the order ID for QUEUE.
COMPUTE_CLUSTER0_C71SS3	0x6BA08000 + (j * 8); where j = 0 to 6	DRU_QUEUE_cfg_j	7:4	ORDERID	This configures the order ID for QUEUE.

The code that sets the DRU queue configuration registers and order IDs is found in the respective main.c in the vision_apps repository (platform/j784s4/rtos/c7x_4/main.c).

```
...

/* DRU configuration */
uint32_t gDruQoS_Enable    = 1;
uint32_t gQoS_DRU_Priority = 3;
uint32_t gQoS_DRU_OrderID  = 0;

void setup_dru_qos(void)
{
   uint64_t DRU_BASE = CSL_COMPUTE_CLUSTER0_MMR_DRU7_MMR_CFG_DRU_BASE;
   volatile uint64_t* queue0CFG     = (uint64_t*)(DRU_BASE + 0x8000);

   if(gQoS_DRU_Priority > 7 || (gDruQoS_Enable == 0))
   {
```

```
    gQoS_DRU_Priority = 0;
  }
  if(gQoS_DRU_OrderID > 15 || (gDruQoS_Enable == 0))
  {
    gQoS_DRU_OrderID = 0;
  }

  uint64_t queue0CFG_VAL = 0x0;
  queue0CFG_VAL |= ((uint64_t)gQoS_DRU_OrderID)<<4;
  queue0CFG_VAL |= ((uint64_t)gQoS_DRU_Priority);

  *queue0CFG = queue0CFG_VAL;
}

...
```

3.3.2.2 NRT and RT Routing

Looking at the order IDs for the C7x_4 core (0) and DSS (15) transactions, it is possible to set DSS transactions as RT while keeping the C7x_4 core transactions as NRT. Drder IDs 0-4 should be routed to the NRT thread, while order IDs 10-15 should be routed to the RT thread.

It's simple to check whether DSS transactions are routed to the RT thread; we can read the NAVSS_NORTH_x_NBSS_NBx_MMRS_threadmap registers (using devmem2).

Table 3-9. North Bridge Thread Mapping Registers

Register	Address	Value
NAVSS_NORTH_0_NBSS_NB0_MMRS_threadmap	0x03702010	0x00000002
NAVSS_NORTH_1_NBSS_NB1_MMRS_threadmap	0x03703010	0x00000004

These values mean SRAM and DDR transactions with order IDs 0-7 are mapped to the NRT thread, while SRAM transactions with order IDs 8-15 and DDR transactions with order IDs 10-15 are mapped to the RT thread. In other words, C7x_4 (order ID 0) transactions are mapped to the NRT thread, while DSS (order ID 15) transactions are mapped to the RT thread. Therefore, DSS transactions should have greater priority than C7x_4 transactions at all times.

3.3.2.2.1 NRT and RT Routing in U-Boot

For the J784S4, the RT and NRT mapping was added into the ti-u-boot-2025.01 branch/release. In previous releases, the code was added through uncommitted edits packaged within the SDK. The relevant code can be found within arch/arm/mach-k3/j784s4/j784s4_init.c.

The code below maps SRAM transactions with order IDs 8-15 and DDR transactions with order IDs 10-15 to the RT thread.

```
...

/* NAVSS North Bridge (NB) */
#define NAVSS0_NBSS_NB0_CFG_MMRS                0x03702000
#define NAVSS0_NBSS_NB1_CFG_MMRS                0x03703000
#define NAVSS0_NBSS_NB0_CFG_NB_THREADMAP        (NAVSS0_NBSS_NB0_CFG_MMRS + 0x10)
#define NAVSS0_NBSS_NB1_CFG_NB_THREADMAP        (NAVSS0_NBSS_NB1_CFG_MMRS + 0x10)
/*
 * Thread Map for North Bridge Configuration
 * Each bit is for each VBUSM source.
 * Bit[0] maps orderID 0-3 to VBUSM.C thread number
 * Bit[1] maps orderID 4-9 to VBUSM.C thread number
 * Bit[2] maps orderID 10-15 to VBUSM.C thread number
 * When bit has value 0: VBUSM.C thread 0 (non-real time traffic)
 * When bit has value 1: VBUSM.C thread 2 (real time traffic)
 */
#define NB_THREADMAP_BIT0                               BIT(0)
#define NB_THREADMAP_BIT1                               BIT(1)
#define NB_THREADMAP_BIT2                               BIT(2)

...
```

```
/* Setup North Bridge registers to map ORDERID 10-15 to RT traffic */
static void setup_navss_nb(void)
{
        writel(NB_THREADMAP_BIT1, (uintptr_t)NAVSS0_NBSS_NB0_CFG_NB_THREADMAP);
        writel(NB_THREADMAP_BIT2, (uintptr_t)NAVSS0_NBSS_NB1_CFG_NB_THREADMAP);
}

...
```

3.3.2.3 Editing Priority

The priority for the separate IPs are set within their respective drivers.

3.3.2.3.1 DSS Priority

The DSS contains two possible priorities for transactions: a high priority for MFLAG transactions and a low priority for non-MFLAG transactions. The MFLAG mechanism allows for the DSS to increase the priority of its traffic when its DMA read buffers are close to underflow.

Table 3-10. DSS Priority Registers

Register	Address	Bits	Field	Description
DSS_DISPC_0_COMMON_M_DSS_CBA_CFG	0x04A000A4	5:3	PRI_HI	The value sent out on the PRI_HI bus from DSS to CBA Indicates the priority level for high-priority [MFLAG] transactions. Value of 0x0 indicates highest priority Value of 0x7 indicates lowest priority
DSS_DISPC_0_COMMON_M_DSS_CBA_CFG	0x04A000A4	2:0	PRI_LO	The value sent out on the PRI_LO bus from DSS to CBA Indicates the priority level for normal [non-MFLAG] transactions. Value of 0x0 indicates highest priority Value of 0x7 indicates lowest priority

The code that sets the priority for the DSS is found in its Linux driver (drivers/gpu/drm/tidss/tidss_dispc.c). This code sets the priority level of MFLAG transactions to 0 and non-MFLAG transactions to 1.

```
...

        u32 cba_lo_pri = 1;
        u32 cba_hi_pri = 0;

        dev_dbg(dispc->dev, "%s()\n", __func__);

        REG_FLD_MOD(dispc, DSS_CBA_CFG, cba_lo_pri, 2, 0);
        REG_FLD_MOD(dispc, DSS_CBA_CFG, cba_hi_pri, 5, 3);

...
```

3.3.2.3.2 C7x Priority

Like the order ID, the priority of each C7x is configurable using the DRU queue configuration registers.

Table 3-11. C7x Priority Registers

C7x	Register	Register	Bits	Field	Description
COMPUTE_CLUSTER0_C71SS0	0x68A08000 + (j * 8); where j = 0 to 6	DRU_QUEUE_cfg_j	2:0	PRI	This configures the priority for QUEUE0. This will be the priority that will be presented on the External bus for all commands from this queue.
COMPUTE_CLUSTER0_C71SS1	0x69A08000 + (j * 8); where j = 0 to 6	DRU_QUEUE_cfg_j	2:0	PRI	This configures the priority for QUEUE0. This will be the priority that will be presented on the External bus for all commands from this queue.
COMPUTE_CLUSTER0_C71SS2	0x6AA08000 + (j * 8); where j = 0 to 6	DRU_QUEUE_cfg_j	2:0	PRI	This configures the priority for QUEUE0. This will be the priority that will be presented on the External bus for all commands from this queue.
COMPUTE_CLUSTER0_C71SS3	0x6BA08000 + (j * 8); where j = 0 to 6	DRU_QUEUE_cfg_j	2:0	PRI	This configures the priority for QUEUE0. This will be the priority that will be presented on the External bus for all commands from this queue.

Like the order ID, the code that sets the priority level is found in the respective main.c in the vision_apps repository (platform/j784s4/rtos/c7x_4/main.c).

```
...

/* DRU configuration */
uint32_t gDruQoS_Enable    = 1;
uint32_t gQoS_DRU_Priority = 3;
uint32_t gQoS_DRU_OrderID  = 0;

void setup_dru_qos(void)
{
   uint64_t DRU_BASE = CSL_COMPUTE_CLUSTER0_MMR_DRU7_MMR_CFG_DRU_BASE;
   volatile uint64_t* queue0CFG     = (uint64_t*)(DRU_BASE + 0x8000);

   if(gQoS_DRU_Priority > 7 || (gDruQoS_Enable == 0))
   {
     gQoS_DRU_Priority = 0;
   }
   if(gQoS_DRU_OrderID > 15 || (gDruQoS_Enable == 0))
   {
     gQoS_DRU_OrderID = 0;
   }

   uint64_t queue0CFG_VAL = 0x0;
```

```
    queue0CFG_VAL |= ((uint64_t)gQoS_DRU_OrderID)<<4;
    queue0CFG_VAL |= ((uint64_t)gQoS_DRU_Priority);

    *queue0CFG = queue0CFG_VAL;
}

...
```

3.3.3 Editing CoS Mappings

Since all of the QoS settings are correct and give preference to the DSS transactions (rather than the C7x_4 core transactions), the issue must lie within the DDR controller's priority mappings.

3.3.3.1 CoS Mapping Registers

The DDRSS contains a series of muxes to map VBUSM.C priorities to AXI priorities. The registers controlling the mappings are the following:

- **Route ID filters**:
  - emif_ew_sscfg_V2A_R1_MAT_REG: allows for filtering and routing of route IDs to range 1 mappings
  - emif_ew_sscfg_V2A_R2_MAT_REG: allows for filtering and routing of route IDs to range 2 mappings
  - emif_ew_sscfg_V2A_R3_MAT_REG: allows for filtering and routing of route IDs to range 3 mappings
- **Priority mappings**:
  - **LPT (low priority thread)**:
    - emif_ew_sscfg_V2A_LPT_DEF_PRI_MAP_REG: default VBUSM.C to AXI priority mappings
    - emif_ew_sscfg_V2A_LPT_R1_PRI_MAP_REG: range 1 VBUSM.C to AXI priority mappings
    - emif_ew_sscfg_V2A_LPT_R2_PRI_MAP_REG: range 2 VBUSM.C to AXI priority mappings
    - emif_ew_sscfg_V2A_LPT_R3_PRI_MAP_REG: range 3 VBUSM.C to AXI priority mappings
  - **HPT (high priority thread)**:
    - emif_ew_sscfg_V2A_HPT_DEF_PRI_MAP_REG: default VBUSM.C to AXI priority mappings
    - emif_ew_sscfg_V2A_HPT_R1_PRI_MAP_REG: range 1 VBUSM.C to AXI priority mappings
    - emif_ew_sscfg_V2A_HPT_R2_PRI_MAP_REG: range 2 VBUSM.C to AXI priority mappings
    - emif_ew_sscfg_V2A_HPT_R3_PRI_MAP_REG: range 3 VBUSM.C to AXI priority mappings

Note

The exact register addresses and fields can be found within the TDA4VH TRM

Take a hypothetical **LPT** transaction. If its route ID falls within a filter, say **range 1**, it will have its priority mapped using the **emif_ew_sscfg_V2A_LPT_R1_PRI_MAP_REG** mappings. If it doesn't fall within any filter, it will have its priority mapped using the **emif_ew_sscfg_V2A_LPT_DEF_PRI_MAP_REG** register. This muxing is represented in Figure 2-1.

3.3.3.2 Checking CoS Mappings

Let's reiterate the QoS settings for the DSS and C7x_4 core transactions:

- **DSS**:
  - Route ID: 0xA20
  - Order ID: 0x0F
  - NRT or RT: RT
  - Priority: 0x00 or 0x01
- **C7x_4 core**:
  - Route ID: 0x02D
  - Order ID: 0x00
  - NRT or RT: NRT
  - Priority: 0x03

The following table contains the values of the CoS registers (the register groups are terms I came up with to categorize the registers):

Table 3-12. CoS Registers

Register Group	Register	Value
Route ID filters	emif_ew_sscfg_V2A_R1_MAT_REG	0x00000000
	emif_ew_sscfg_V2A_R2_MAT_REG	0x00000000
	emif_ew_sscfg_V2A_R3_MAT_REG	0x00000000
LPT priority mappings	emif_ew_sscfg_V2A_LPT_DEF_PRI_MAP_REG	0x00000000
	emif_ew_sscfg_V2A_LPT_R1_PRI_MAP_REG	0x23456677
	emif_ew_sscfg_V2A_LPT_R2_PRI_MAP_REG	0x23456677
	emif_ew_sscfg_V2A_LPT_R3_PRI_MAP_REG	0x23456677
HPT priority mappings	emif_ew_sscfg_V2A_HPT_DEF_PRI_MAP_REG	0x00000000
	emif_ew_sscfg_V2A_HPT_R1_PRI_MAP_REG	0x00112345
	emif_ew_sscfg_V2A_HPT_R2_PRI_MAP_REG	0x00112345
	emif_ew_sscfg_V2A_HPT_R3_PRI_MAP_REG	0x00112345

None of the **Route ID filters** are enabled. This means that all priorities will be mapped using the default priority mappings.

The default priority mappings for both the LPT and HPT transactions are equalized to 0. This means that all transactions have the same priority within the DDRSS. Therefore, both the DSS and C7x_4 core transactions have the same priority within the DDR; this explains how the C7x_4 core transactions are stalling the DSS transactions.

3.4 Fixing the DSS Sync Losts

Now that the issue has been root caused, we need to decide how to fix it.

Having all of the priorities equalized within the DDR controller has benefits. Namely, no threads are evicted when a higher priority thread enters the queue, which helps prevent page thrashing. In this case however, not honoring priorities results in greater detriments than benefits.

There are a couple options to fix the sync lost issue:

1. Remap C7x_4 core transactions
2. Honor priority across all transactions

Remapping exclusively C7x_4 core transactions will resolve the sync lost issue, while leaving all other transactions able to supersede the DSS transactions. Honoring priority across all transactions will force the DDRSS to treat all transactions accordingly with their VBUSM.C priorities.

Due to there only being eight different AXI priority settings, there is some overlap between the HPT and LPT when trying to honor priority across all transactions.

Table 3-13. HPT and LPT to AXI Priority Mapping

Real-Time or Non-Real-Time	VBUSM.C Priority	AXI Priority
Real-time	0	0
Real-time	1	0
Real-time	2	1

Table 3-13. HPT and LPT to AXI Priority Mapping (continued)

Real-Time or Non-Real-Time	VBUSM.C Priority	AXI Priority
Real-time	3	1
Real-time	4	2
Non-real-time	0	2
Real-time	5	3
Non-real-time	1	3
Real-time	6	4
Non-real-time	2	4
Real-time	7	5
Non-real-time	3	5
Non-real-time	4	6
Non-real-time	5	6
Non-real-time	6	7
Non-real-time	7	7

The CoS mapping is added to U-Boot during the board initialization. Patches are written for both ti-u-boot-2023.04 and ti-u-boot-2025.01 branches.

3.4.1 Remap C7x_4 Core Transactions

The following U-Boot patches map the C7x_4 core transactions to the LPT range 1 priority mappings. This avoids changing the priorities of any other transactions.

Note

If you wanted to set all C7x transactions to the LPT range 1 priority mappings, you would replace 0xf02d0000 with 0x80208028

3.4.1.1 ti-u-boot-2023.04

The following patch applies on top of commit: 2bedcd265ca6 ("configs: am57xx_hs_evm_defconfig: Enable configs needed for Early Boot")

0001-arm-mach-k3-j784s4-Remove-priority-equalization-for-.patch

```
From 6a8d46d01e482cff6678189087155f2dd2ddafc9 Mon Sep 17 00:00:00 2001
From: Jared McArthur <j-mcarthur@ti.com>
Date: Thu, 28 Aug 2025 18:33:22 -0500
Subject: [PATCH 1/1] arm: mach-k3: j784s4: Remove priority equalization for
 C7x_4 core transactions

Add C7x_4 core transactions to the low priority thread (LPT) R1 mux
within the DDR controller. By default, the J784S4's MSMC2DDR bridge
maps all VBUSM priorities to 0 and all transactions travel through the
default mux [0].

By default, the LPT R1 mux honors VBUSM priorities. Move C7x_4 core
transactions to the LPT R1 mux, so they will honor VBUSM priorities.
All other transactions priorities remain unchanged from default
behavior.

[0] https://www.ti.com/lit/zip/spruj52

Signed-off-by: Jared McArthur <j-mcarthur@ti.com>
---
 arch/arm/mach-k3/j784s4_init.c | 33 +++++++++++++++++++++++++++++++++
 1 file changed, 33 insertions(+)

diff --git a/arch/arm/mach-k3/j784s4_init.c b/arch/arm/mach-k3/j784s4_init.c
index af0f46e2ab5..4bcc83dadaa 100644
--- a/arch/arm/mach-k3/j784s4_init.c
+++ b/arch/arm/mach-k3/j784s4_init.c
@@ -22,6 +22,26 @@
```

```
 #include <mmc.h>
 #include <remoteproc.h>

+/* DDRSS Config */
+#define DDRSS0_EMIF_EW_SSCFG    0x02980000
+#define DDRSS1_EMIF_EW_SSCFG    0x029A0000
+#define DDRSS2_EMIF_EW_SSCFG    0x029C0000
+#define DDRSS3_EMIF_EW_SSCFG    0x029E0000
+#define DDRSS0_V2A_R1_MAT_REG    (DDRSS0_EMIF_EW_SSCFG + 0x24)
+#define DDRSS1_V2A_R1_MAT_REG    (DDRSS1_EMIF_EW_SSCFG + 0x24)
+#define DDRSS2_V2A_R1_MAT_REG    (DDRSS2_EMIF_EW_SSCFG + 0x24)
+#define DDRSS3_V2A_R1_MAT_REG    (DDRSS3_EMIF_EW_SSCFG + 0x24)
+
+/*
+ * Move the C7x_4 core transactions to the LPT range 1 priority mappings.
+ * This decreases the priority of specifically C7x core transactions,
+ * but doesn't impact the priority of any other transactions.
+ */
+#define DDRSS0_V2A_R1_MAT    0xf02d0000
+#define DDRSS1_V2A_R1_MAT    0xf02d0000
+#define DDRSS2_V2A_R1_MAT    0xf02d0000
+#define DDRSS3_V2A_R1_MAT    0xf02d0000
+
 struct fwl_data infra_cbass0_fwls[] = {
     { "PSC0", 5, 1 },
     { "PLL_CTRL0", 6, 1 },
@@ -139,6 +159,17 @@ static void store_boot_info_from_rom(void)

 #define J784S4_MAX_CONTROLLERS    4

+/* Setup DDRSS CoS (Class of Service) registers to remove priority equalization
+ * for C7x_4 core transactions
+ */
+static void setup_ddrss_cos(void)
+{
+    writel(DDRSS0_V2A_R1_MAT, (uintptr_t)DDRSS0_V2A_R1_MAT_REG);
+    writel(DDRSS1_V2A_R1_MAT, (uintptr_t)DDRSS1_V2A_R1_MAT_REG);
+    writel(DDRSS2_V2A_R1_MAT, (uintptr_t)DDRSS2_V2A_R1_MAT_REG);
+    writel(DDRSS3_V2A_R1_MAT, (uintptr_t)DDRSS3_V2A_R1_MAT_REG);
+}
+
 void board_init_f(ulong dummy)
 {
     struct udevice *dev;
@@ -241,6 +272,8 @@ void board_init_f(ulong dummy)
     }

     spl_enable_dcache();
+
+    setup_ddrss_cos();
 }

 u32 spl_mmc_boot_mode(struct mmc *mmc, const u32 boot_device)
--
2.34.1
```

3.4.1.2 ti-u-boot-2025.01

The following patch applies on top of commit: f3f8c664b300 ("PSTREAM: board: ti: common: Kconfig: add CMD_CACHE")

0001-arm-mach-k3-j784s4-Remove-priority-equalization-for-.patch

```
From 734130b26838b77759e7bcbb0f8af79330e48ec7 Mon Sep 17 00:00:00 2001
From: Jared McArthur <j-mcarthur@ti.com>
Date: Thu, 28 Aug 2025 17:35:44 -0500
Subject: [PATCH 1/1] arm: mach-k3: j784s4: Remove priority equalization for
 C7x_4 core transactions

Add C7x_4 core transactions to the low priority thread (LPT) R1 mux
within the DDR controller. By default, the J784S4's MSMC2DDR bridge
maps all VBUSM priorities to 0 and all transactions travel through the
default mux [0].

By default, the LPT R1 mux honors VBUSM priorities. Move C7x_4 core
transactions to the LPT R1 mux, so they will honor VBUSM priorities.
All other transactions priorities remain unchanged from default
```

```
behavior.

[0] https://www.ti.com/lit/zip/spruj52

Signed-off-by: Jared McArthur <j-mcarthur@ti.com>
---
 arch/arm/mach-k3/j784s4/j784s4_init.c | 32 ++++++++++++++++++++++++++++
 1 file changed, 32 insertions(+)

diff --git a/arch/arm/mach-k3/j784s4/j784s4_init.c b/arch/arm/mach-k3/j784s4/j784s4_init.c
index 9897f4fb921..8557d3b8ddf 100644
--- a/arch/arm/mach-k3/j784s4/j784s4_init.c
+++ b/arch/arm/mach-k3/j784s4/j784s4_init.c
@@ -45,6 +45,26 @@
 #define NB_THREADMAP_BIT1               BIT(1)
 #define NB_THREADMAP_BIT2               BIT(2)

+/* DDRSS Config */
+#define DDRSS0_EMIF_EW_SSCFG    0x02980000
+#define DDRSS1_EMIF_EW_SSCFG    0x029A0000
+#define DDRSS2_EMIF_EW_SSCFG    0x029C0000
+#define DDRSS3_EMIF_EW_SSCFG    0x029E0000
+#define DDRSS0_V2A_R1_MAT_REG    (DDRSS0_EMIF_EW_SSCFG + 0x24)
+#define DDRSS1_V2A_R1_MAT_REG    (DDRSS1_EMIF_EW_SSCFG + 0x24)
+#define DDRSS2_V2A_R1_MAT_REG    (DDRSS2_EMIF_EW_SSCFG + 0x24)
+#define DDRSS3_V2A_R1_MAT_REG    (DDRSS3_EMIF_EW_SSCFG + 0x24)
+
+/*
+ * Move the C7x_4 core transactions to the LPT range 1 priority mappings.
+ * This decreases the priority of specifically C7x core transactions,
+ * but doesn't impact the priority of any other transactions.
+ */
+#define DDRSS0_V2A_R1_MAT    0xf02d0000
+#define DDRSS1_V2A_R1_MAT    0xf02d0000
+#define DDRSS2_V2A_R1_MAT    0xf02d0000
+#define DDRSS3_V2A_R1_MAT    0xf02d0000
+
 struct fwl_data infra_cbass0_fwls[] = {
     { "PSC0", 5, 1 },
     { "PLL_CTRL0", 6, 1 },
@@ -123,6 +143,17 @@ static void setup_navss_nb(void)
     writel(NB_THREADMAP_BIT2, (uintptr_t)NAVSS0_NBSS_NB1_CFG_NB_THREADMAP);
 }

+/* Setup DDRSS CoS (Class of Service) registers to remove priority equalization
+ * for C7x_4 core transactions
+ */
+static void setup_ddrss_cos(void)
+{
+    writel(DDRSS0_V2A_R1_MAT, (uintptr_t)DDRSS0_V2A_R1_MAT_REG);
+    writel(DDRSS1_V2A_R1_MAT, (uintptr_t)DDRSS1_V2A_R1_MAT_REG);
+    writel(DDRSS2_V2A_R1_MAT, (uintptr_t)DDRSS2_V2A_R1_MAT_REG);
+    writel(DDRSS3_V2A_R1_MAT, (uintptr_t)DDRSS3_V2A_R1_MAT_REG);
+}
+
 /* Execute and check results of BIST executed on MCU1_x and MCU4_0 */
 static void run_bist_j784s4(struct udevice *dev)
 {
@@ -328,6 +359,7 @@ void board_init_f(ulong dummy)
         setup_navss_nb();

     setup_qos();
+    setup_ddrss_cos();
 }

 u32 spl_mmc_boot_mode(struct mmc *mmc, const u32 boot_device)
--
2.34.1
```

3.4.2 Honor All Priorities

The following U-Boot patches change the default LPT and HPT priority mappings to the same as the range 1-3 mappings. This is the default behavior for the TDA4VM/J721E.

3.4.2.1 ti-u-boot-2023.04

The following patch applies on top of commit: 2bedcd265ca6 ("configs: am57xx_hs_evm_defconfig: Enable configs needed for Early Boot")

0001-arm-mach-k3-j784s4-Remove-priority-equalization-and-.patch

```
From 6ebd1448e30ed1861492738759680b90209fcc22 Mon Sep 17 00:00:00 2001
From: Jared McArthur <j-mcarthur@ti.com>
Date: Thu, 28 Aug 2025 18:33:22 -0500
Subject: [PATCH 1/1] arm: mach-k3: j784s4: Remove priority equalization and
 honor VBUSM priorities

Give the DDR controller's high priority thread (HPT) priority over the
low priority thread (LPT) and give weight to transactions' individual
priorities as well.

By default, the J784S4's MSMC2DDR bridge maps all VBUSM priorities to
0. This is done with priority mapping registers.

DDRSSX_V2A_LPT_DEF_PRI_MAP_REG: default VBUSM to DDR controller
priority mapping for LPT

DDRSSX_V2A_HPT_DEF_PRI_MAP_REG: default VBUSM to DDR controller
priority mapping for HPT

Set DDRSSX_V2A_LPT_DEF_PRI_MAP_REG as 0x23456677 and
DDRSSX_V2A_HPT_DEF_PRI_MAP_REG as 0x00112345. The values are taken
from the default values for the J721E [0] and are also the default values
for the J784S4 priority map range muxes [1].

[0] https://www.ti.com/lit/zip/spruil1
[1] https://www.ti.com/lit/zip/spruj52

Signed-off-by: Jared McArthur <j-mcarthur@ti.com>
---
 arch/arm/mach-k3/j784s4_init.c | 36 ++++++++++++++++++++++++++++++++++++
 1 file changed, 36 insertions(+)

diff --git a/arch/arm/mach-k3/j784s4_init.c b/arch/arm/mach-k3/j784s4_init.c
index af0f46e2ab5..eb45ccd0cb3 100644
--- a/arch/arm/mach-k3/j784s4_init.c
+++ b/arch/arm/mach-k3/j784s4_init.c
@@ -22,6 +22,27 @@
 #include <mmc.h>
 #include <remoteproc.h>

+/* DDRSS Config */
+#define DDRSS0_EMIF_EW_SSCFG        0x02980000
+#define DDRSS1_EMIF_EW_SSCFG        0x029A0000
+#define DDRSS2_EMIF_EW_SSCFG        0x029C0000
+#define DDRSS3_EMIF_EW_SSCFG        0x029E0000
+#define DDRSS0_V2A_LPT_DEF_PRI_MAP_REG    (DDRSS0_EMIF_EW_SSCFG + 0x30)
+#define DDRSS0_V2A_HPT_DEF_PRI_MAP_REG    (DDRSS0_EMIF_EW_SSCFG + 0x4C)
+#define DDRSS1_V2A_LPT_DEF_PRI_MAP_REG    (DDRSS1_EMIF_EW_SSCFG + 0x30)
+#define DDRSS1_V2A_HPT_DEF_PRI_MAP_REG    (DDRSS1_EMIF_EW_SSCFG + 0x4C)
+#define DDRSS2_V2A_LPT_DEF_PRI_MAP_REG    (DDRSS2_EMIF_EW_SSCFG + 0x30)
+#define DDRSS2_V2A_HPT_DEF_PRI_MAP_REG    (DDRSS2_EMIF_EW_SSCFG + 0x4C)
+#define DDRSS3_V2A_LPT_DEF_PRI_MAP_REG    (DDRSS3_EMIF_EW_SSCFG + 0x30)
+#define DDRSS3_V2A_HPT_DEF_PRI_MAP_REG    (DDRSS3_EMIF_EW_SSCFG + 0x4C)
+
+/*
+ * New thread priority mapping to remove complete priority equalization
+ * of threads coming from VBUSM space to DDRSS space.
+ */
+#define DDRSS_V2A_LPT_DEF_PRIMAP    0x23456677
+#define DDRSS_V2A_HPT_DEF_PRIMAP    0x00112345
+
 struct fwl_data infra_cbass0_fwls[] = {
     { "PSC0", 5, 1 },
     { "PLL_CTRL0", 6, 1 },
@@ -139,6 +160,19 @@ static void store_boot_info_from_rom(void)

 #define J784S4_MAX_CONTROLLERS    4

+/* Setup DDRSS CoS (Class of Service) registers to remove priority equalization */
+static void setup_ddrss_cos(void)
```

```
+{
+    writel(DDRSS_V2A_LPT_DEF_PRIMAP, (uintptr_t)DDRSS0_V2A_LPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_HPT_DEF_PRIMAP, (uintptr_t)DDRSS0_V2A_HPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_LPT_DEF_PRIMAP, (uintptr_t)DDRSS1_V2A_LPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_HPT_DEF_PRIMAP, (uintptr_t)DDRSS1_V2A_HPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_LPT_DEF_PRIMAP, (uintptr_t)DDRSS2_V2A_LPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_HPT_DEF_PRIMAP, (uintptr_t)DDRSS2_V2A_HPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_LPT_DEF_PRIMAP, (uintptr_t)DDRSS3_V2A_LPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_HPT_DEF_PRIMAP, (uintptr_t)DDRSS3_V2A_HPT_DEF_PRI_MAP_REG);
+}
+
 void board_init_f(ulong dummy)
 {
     struct udevice *dev;
@@ -241,6 +275,8 @@ void board_init_f(ulong dummy)
     }

     spl_enable_dcache();
+
+    setup_ddrss_cos();
 }

 u32 spl_mmc_boot_mode(struct mmc *mmc, const u32 boot_device)
--
2.34.1
```

3.4.2.2 ti-u-boot-2025.01

The following patch applies on top of commit: f3f8c664b300 ("PSTREAM: board: ti: common: Kconfig: add CMD_CACHE")

0001-arm-mach-k3-j784s4-Remove-priority-equalization-and-.patch

```
From 7ac3b82eca22d6d28ca767d400c8aa5830704a59 Mon Sep 17 00:00:00 2001
From: Jared McArthur <j-mcarthur@ti.com>
Date: Thu, 28 Aug 2025 17:35:44 -0500
Subject: [PATCH 1/1] arm: mach-k3: j784s4: Remove priority equalization and
 honor VBUSM priorities

Give the DDR controller's high priority thread (HPT) priority over the
low priority thread (LPT) and give weight to transactions' individual
priorities as well.

By default, the J784S4's MSMC2DDR bridge maps all VBUSM priorities to
0. This is done with priority mapping registers.

DDRSSX_V2A_LPT_DEF_PRI_MAP_REG: default VBUSM to DDR controller
priority mapping for LPT

DDRSSX_V2A_HPT_DEF_PRI_MAP_REG: default VBUSM to DDR controller
priority mapping for HPT

Set DDRSSX_V2A_LPT_DEF_PRI_MAP_REG as 0x23456677 and
DDRSSX_V2A_HPT_DEF_PRI_MAP_REG as 0x00112345. The values are taken
from the default values for the J721E [0] and are also the default values
for the J784S4 priority map range muxes [1].

[0] https://www.ti.com/lit/zip/spruil1
[1] https://www.ti.com/lit/zip/spruj52

Signed-off-by: Jared McArthur <j-mcarthur@ti.com>
---
 arch/arm/mach-k3/j784s4/j784s4_init.c | 35 +++++++++++++++++++++++++++++++++
 1 file changed, 35 insertions(+)

diff --git a/arch/arm/mach-k3/j784s4/j784s4_init.c b/arch/arm/mach-k3/j784s4/j784s4_init.c
index 9897f4fb921..f66a8fd1774 100644
--- a/arch/arm/mach-k3/j784s4/j784s4_init.c
+++ b/arch/arm/mach-k3/j784s4/j784s4_init.c
@@ -45,6 +45,27 @@
 #define NB_THREADMAP_BIT1               BIT(1)
 #define NB_THREADMAP_BIT2               BIT(2)

+/* DDRSS Config */
+#define DDRSS0_EMIF_EW_SSCFG        0x02980000
+#define DDRSS1_EMIF_EW_SSCFG        0x029A0000
+#define DDRSS2_EMIF_EW_SSCFG        0x029C0000
```

```
+#define DDRSS3_EMIF_EW_SSCFG        0x029E0000
+#define DDRSS0_V2A_LPT_DEF_PRI_MAP_REG    (DDRSS0_EMIF_EW_SSCFG + 0x30)
+#define DDRSS0_V2A_HPT_DEF_PRI_MAP_REG    (DDRSS0_EMIF_EW_SSCFG + 0x4C)
+#define DDRSS1_V2A_LPT_DEF_PRI_MAP_REG    (DDRSS1_EMIF_EW_SSCFG + 0x30)
+#define DDRSS1_V2A_HPT_DEF_PRI_MAP_REG    (DDRSS1_EMIF_EW_SSCFG + 0x4C)
+#define DDRSS2_V2A_LPT_DEF_PRI_MAP_REG    (DDRSS2_EMIF_EW_SSCFG + 0x30)
+#define DDRSS2_V2A_HPT_DEF_PRI_MAP_REG    (DDRSS2_EMIF_EW_SSCFG + 0x4C)
+#define DDRSS3_V2A_LPT_DEF_PRI_MAP_REG    (DDRSS3_EMIF_EW_SSCFG + 0x30)
+#define DDRSS3_V2A_HPT_DEF_PRI_MAP_REG    (DDRSS3_EMIF_EW_SSCFG + 0x4C)
+
+/*
+ * New thread priority mapping to remove complete priority equalization
+ * of threads coming from VBUSM space to DDRSS space.
+ */
+#define DDRSS_V2A_LPT_DEF_PRIMAP    0x23456677
+#define DDRSS_V2A_HPT_DEF_PRIMAP    0x00112345
+
 struct fwl_data infra_cbass0_fwls[] = {
     { "PSC0", 5, 1 },
     { "PLL_CTRL0", 6, 1 },
@@ -123,6 +144,19 @@ static void setup_navss_nb(void)
     writel(NB_THREADMAP_BIT2, (uintptr_t)NAVSS0_NBSS_NB1_CFG_NB_THREADMAP);
 }

+/* Setup DDRSS CoS (Class of Service) registers to remove priority equalization */
+static void setup_ddrss_cos(void)
+{
+    writel(DDRSS_V2A_LPT_DEF_PRIMAP, (uintptr_t)DDRSS0_V2A_LPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_HPT_DEF_PRIMAP, (uintptr_t)DDRSS0_V2A_HPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_LPT_DEF_PRIMAP, (uintptr_t)DDRSS1_V2A_LPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_HPT_DEF_PRIMAP, (uintptr_t)DDRSS1_V2A_HPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_LPT_DEF_PRIMAP, (uintptr_t)DDRSS2_V2A_LPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_HPT_DEF_PRIMAP, (uintptr_t)DDRSS2_V2A_HPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_LPT_DEF_PRIMAP, (uintptr_t)DDRSS3_V2A_LPT_DEF_PRI_MAP_REG);
+    writel(DDRSS_V2A_HPT_DEF_PRIMAP, (uintptr_t)DDRSS3_V2A_HPT_DEF_PRI_MAP_REG);
+}
+
 /* Execute and check results of BIST executed on MCU1_x and MCU4_0 */
 static void run_bist_j784s4(struct udevice *dev)
 {
@@ -328,6 +362,7 @@ void board_init_f(ulong dummy)
         setup_navss_nb();

     setup_qos();
+    setup_ddrss_cos();
 }

 u32 spl_mmc_boot_mode(struct mmc *mmc, const u32 boot_device)
--
2.34.1
```

4 Summary

This application note covers an overview of the different QoS mechanisms within Jacinto devices (specifically the TDA4VH) and how to use those mechanisms for load balancing across different IPs. Order IDs impact the paths that transactions take and their real-time and non-real-time allocations. Asigning transactions different priorities and real-time/non-real-time allocations impacts the order in which they are serviced, and serves to balance and optimize the performance of all of the applications and their respective hardware.

In the case study, it was observed that C7x_4 core transactions stalled DSS transactions and caused sync lost errors within the display sub-system. The DSS transactions had priority over the C7x_4 core transactions within the VBUSM and VBUSM.C space, but within the DDRSS, all priorities were equalized. Removing this equalization and honoring the VBUSM.C priorities within the DDR controller removed the sync lost errors.

5 References

1. Texas Instruments, *J784S4 J742S2 Technical Reference Manual*, Technical Reference Manual
2. Texas Instruments, *J721E DRA829/TDA4VM Processors Silicon Revision 2.0, 1.1 Technical Reference Manual*, technical reference manual.
3. Texas Instruments, *PROCESSOR-SDK-LINUX-J784S4*, software development kit.
4. Texas Instruments, *PROCESSOR-SDK-RTOS-J784S4*, software development kit.
5. Texas Instruments, *Traffic Profiling with CP Tracers V2*, TI software downloads.
6. Texas Instruments, *0001-arm-mach-k3-j784s4-Remove-priority-equalization-for-.patch*, ti-u-boot-2023.04.y patch
7. Texas Instruments, *0001-arm-mach-k3-j784s4-Remove-priority-equalization-for-.patch*, ti-u-boot-2023.04.y patch
8. Texas Instruments, *0001-arm-mach-k3-j784s4-Remove-priority-equalization-and-.patch*, ti-u-boot-2025.01.y patch
9. Texas Instruments, *0001-arm-mach-k3-j784s4-Remove-priority-equalization-and-.patch*, ti-u-boot-2025.01.y patch

6 Revision History

Changes from Revision * (March 2026) to Revision A (March 2026)	Page
• Updated title	0
• Added table title to Table 3-13	44
• Updated the numbering format for tables, figures, and cross-references throughout the document	52

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025