EDMA3LLD Overview
(Enhanced DMA Gen 3 Low Level Driver)

ADAS Driver Team
7th Jan 2015
Version 1.1

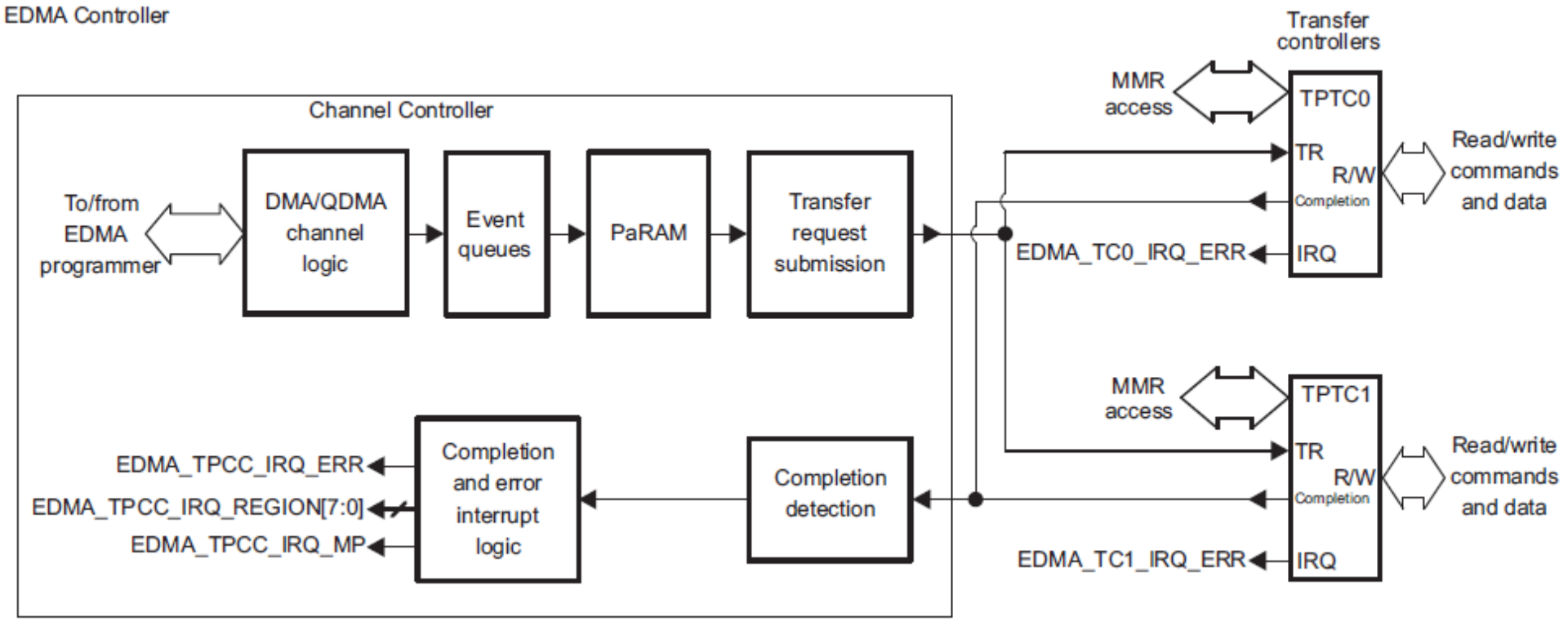
Agenda

- EDMA3 IP Overview
- EDMA3 S/W Architecture
  - Directory Structure and Code walkthrough
  - Build & Run Examples

EDMA Module Overview

- Performs high-performance data transfers between two slave points, memories and peripheral devices
- EDMA controller is based on two major principal blocks
  - EDMA third-party channel controller (EDMA_TPCC)
  - EDMA third-party transfer controller (EDMA_TPTC)
- TPCC serves as an user interface and an event interface and submits transfer requests to TPTC
- TPTC performs read and write transfers by EDMA ports to the slave peripherals

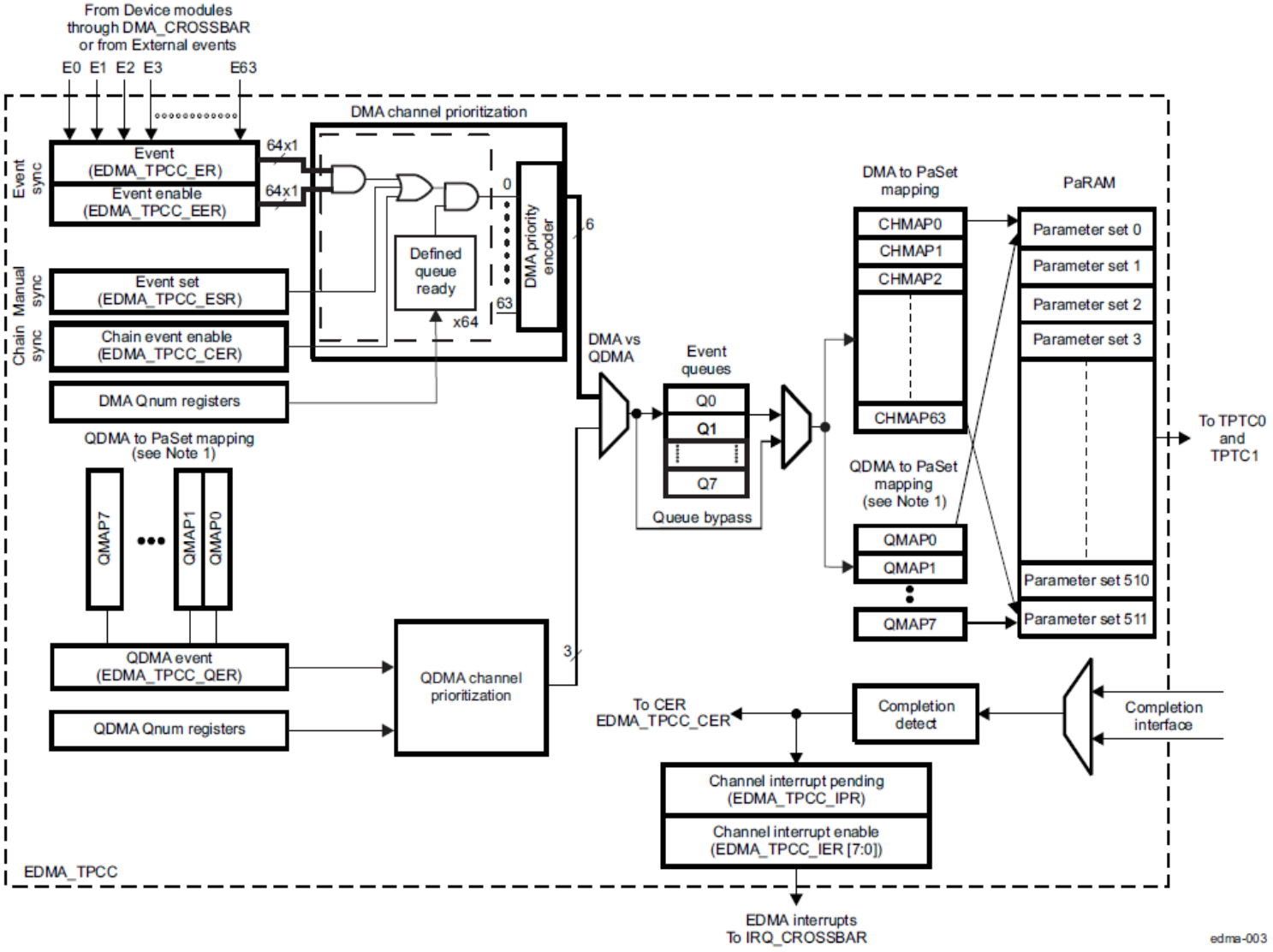
EDMA Controller Block Diagram



Components of TPCC

- Parameter RAM (PaRAM)
  - Contains data transfer parameter sets for DMA channels or for reload
- EDMA event and interrupt processing registers
  - Mapping of events to parameter sets, enable/disable events, enable/disable interrupt conditions, and clearing interrupts.
- Completion detection
  - Detects completion of transfers, can be used optionally to chain trigger new transfers or to assert interrupts
- Event queues
  - Interface between the event detection logic and the transfer request submission logic

TPCC Block Diagram

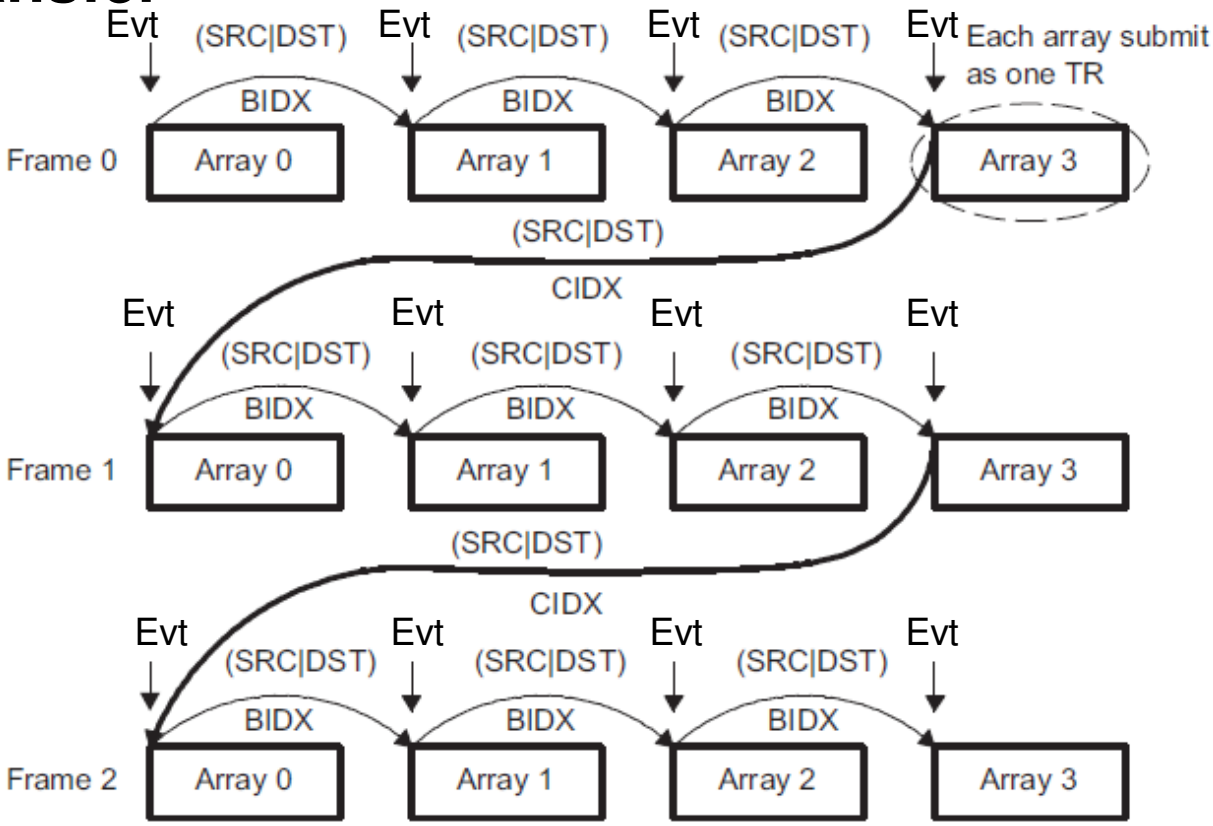


Types of EDMA controller Transfers

- An EDMA transfer is always defined in terms of three dimensions
  - 1st Dimension or Array (A) in a transfer consists of ACNT contiguous bytes
  - 2nd Dimension or Frame (B) in a transfer consists of BCNT arrays of ACNT bytes each
  - 3rd Dimension or Block (C) in a transfer consists of CCNT frames of BCNT arrays of ACNT bytes
- Difference between start of each array in a frame or each frame in a block for Source and Destination is separately programmable

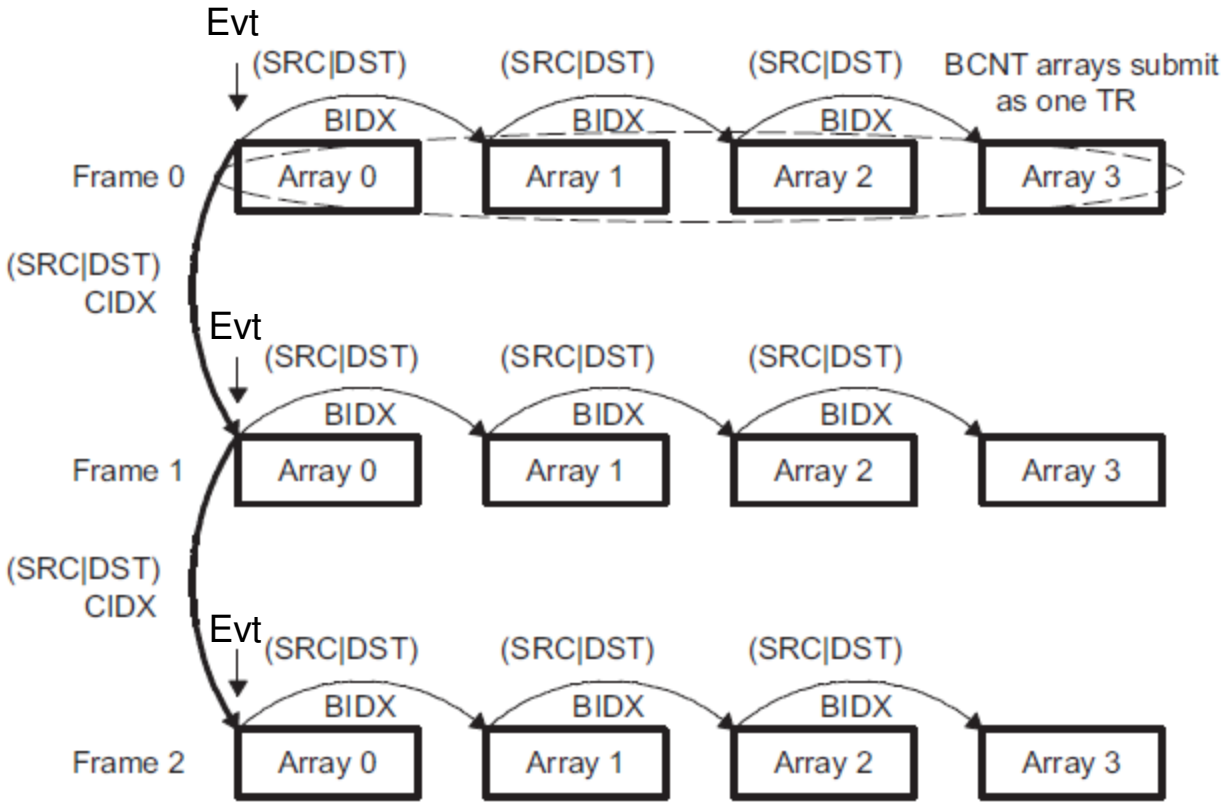
A- Sync Transfer

- **Each trigger will transfer ACNT Bytes.**
- **Totally BCNT * CCNT triggers required for complete transfer**



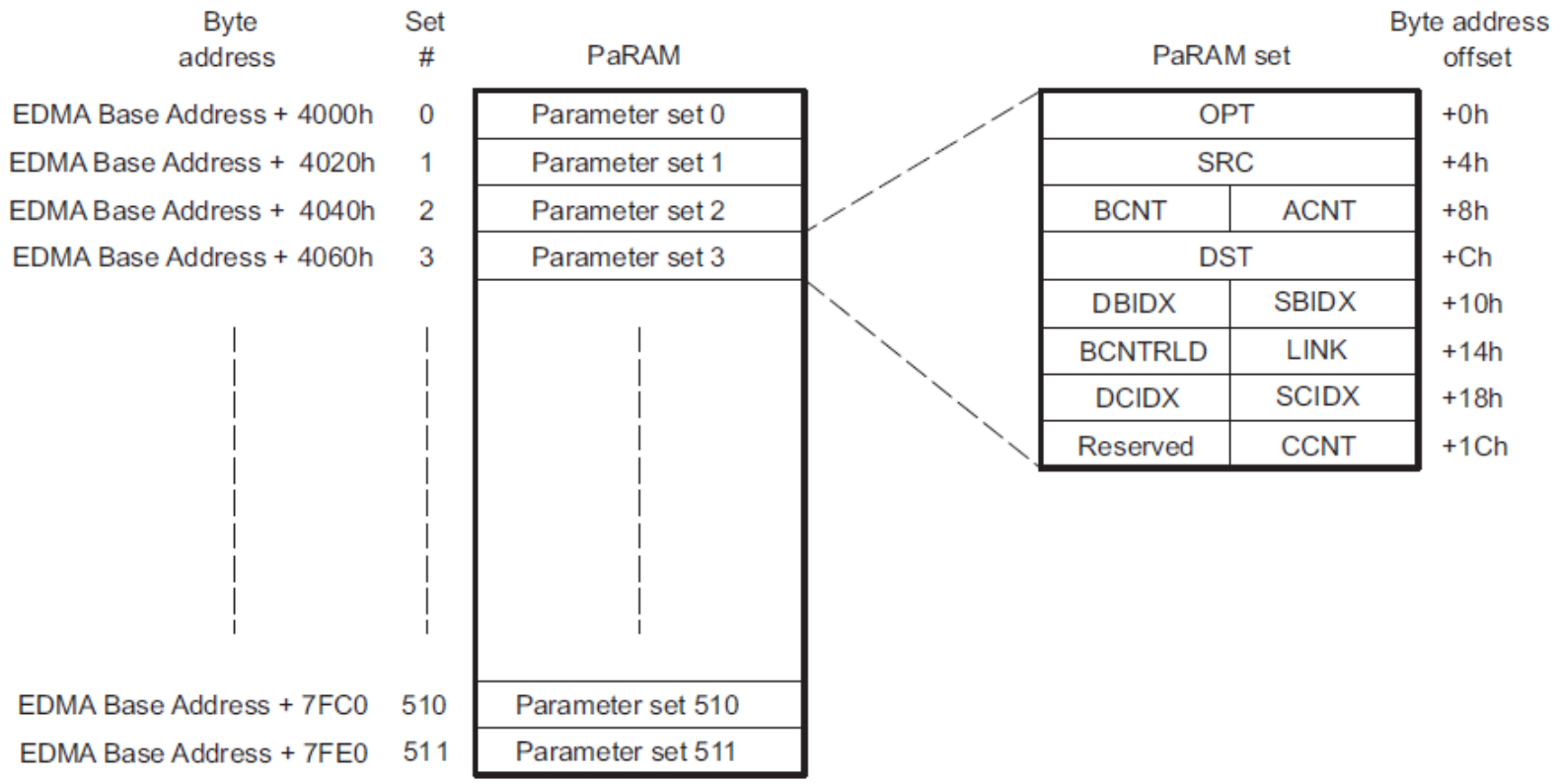
AB- Sync Transfer

- **Each trigger will transfer ACNT * BCNT Bytes.**
- **Totally CCNT triggers required for complete transfer**



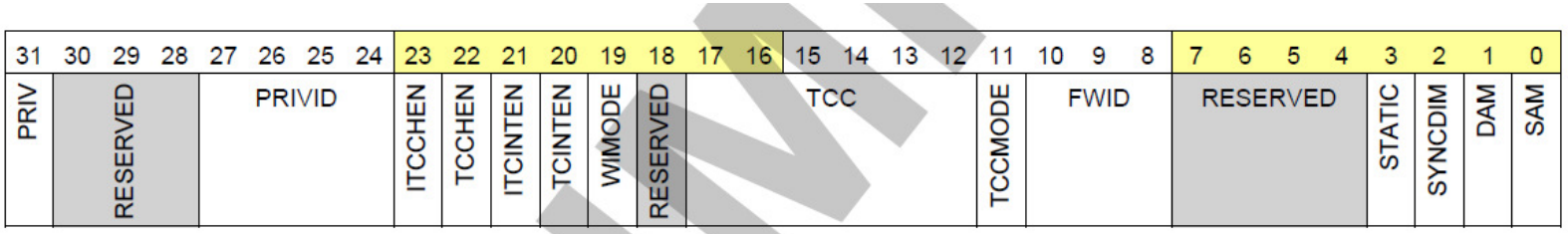
PaRAM Set

- There are 512 PaRAMs that contain configuration information about a transfer



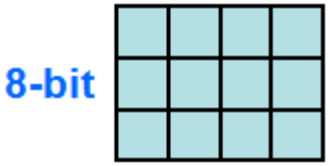
PaRAM Set – OPT Fields

- SAM/DAM – Source/Destination Addressing Mode
- SYNCDIM – A or AB Sync
- FWID – FIFO Width (used if SAM/DAM is set)
- TCC – Transfer Completion Code
- TCINTEN – TC Interrupt Enable
- TCCHEN – TC Chaining Enable



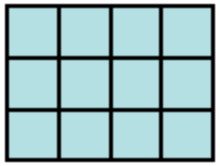
Example – Viewing the transfer

- Lets take an example to transfer 12 bytes from one location to other.

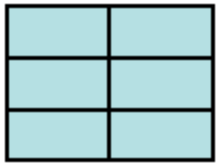


Note: these are contiguous memory locations

- What is ACNT, BCNT and CCNT?
- You can “view” the transfer several ways:



ACNT = 1
BCNT = 4
CCNT = 3



ACNT = 2
BCNT = 2
CCNT = 3

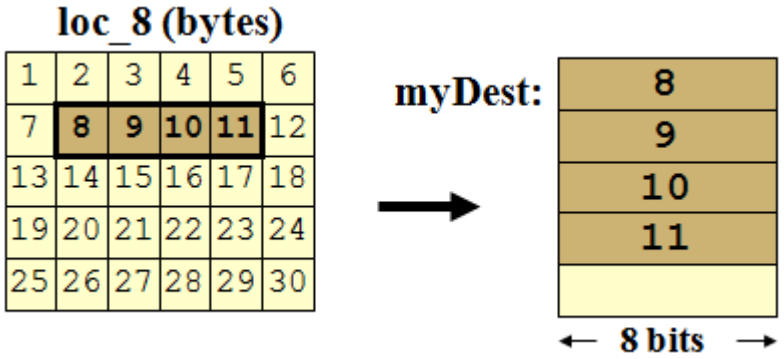
ACNT = 12
BCNT = 1
CCNT = 1
= 12

EDMA Example : Simple (Horizontal Line)

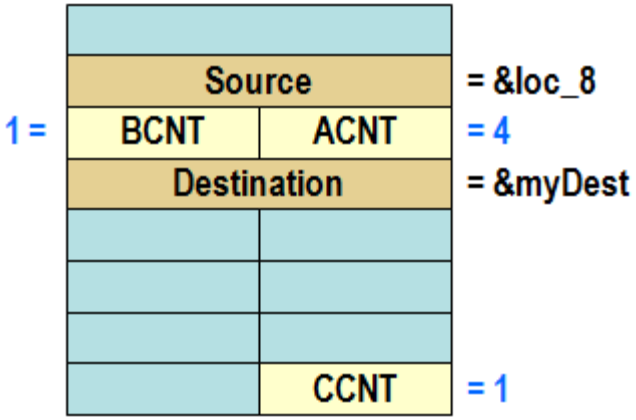
Goal:

Transfer 4 elements

from loc_8 to myDest



- DMA always increments across ACNT fields
- B and C counts must be 1 (or more) for any actions to occur



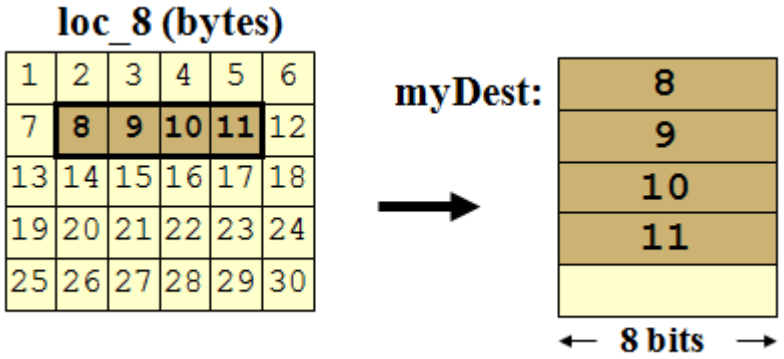
Is there another way to set this up?

EDMA Example : Simple (Horizontal Line)

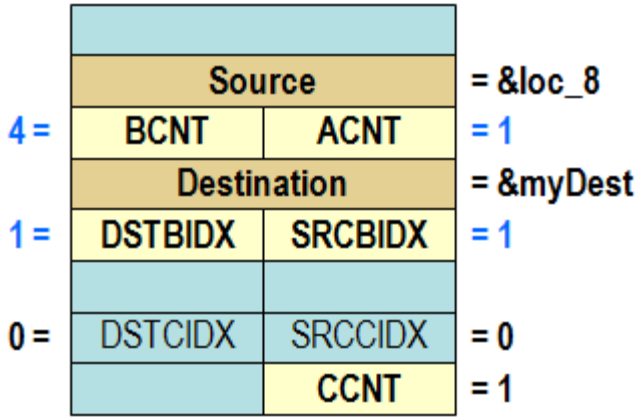
Goal:

Transfer 4 elements

from loc_8 to myDest



- Here, ACNT was defined as element size : 1 byte
- Therefore, BCNT will now be framesize : 4 bytes
- B indexing must now be specified as well



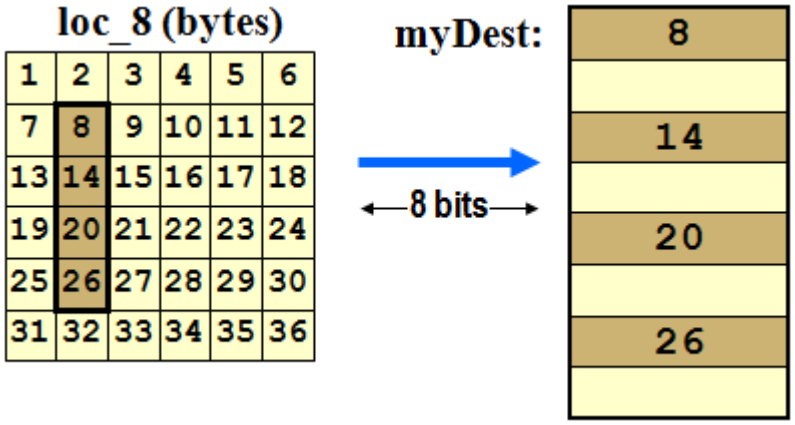
Note: Less efficient version

EDMA Example : Indexing (Vertical Line)

Goal:

Transfer 4 vertical elements

from loc_8 to myDest



- Here, ACNT was defined as element size : 1 byte
- Therefore, BCNT will now be framesize : 4 bytes
- B indexing must now be specified as well

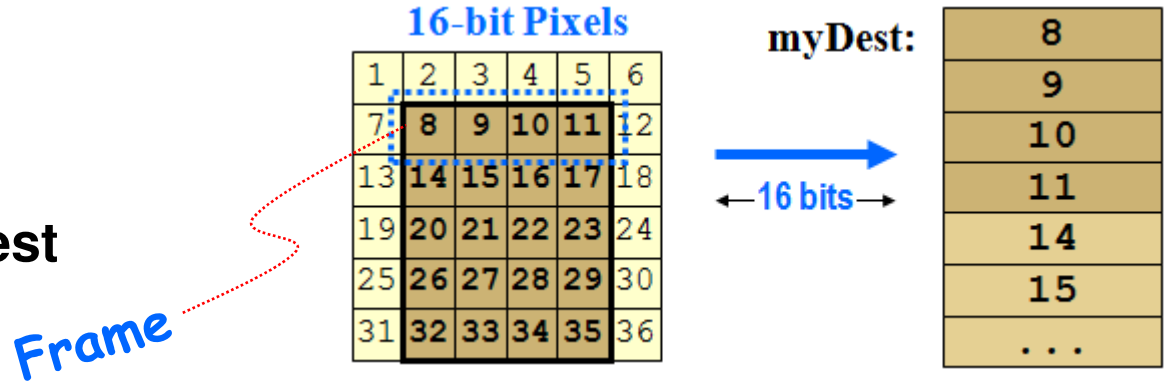
	Source		= &loc_8
4 =	BCNT	ACNT	= 1
	Destination		= &myDest
2 =	DSTBIDX	SRCBIDX	= 6
0 =	DSTCIDX	SRCCIDX	= 0
		CCNT	= 1

EDMA Example : Block Transfer

Goal:

Transfer a 5x4 subset

from loc_8 to myDest



- ACNT is defined here as 'short' element size : 2 bytes
- BCNT is again framesize : 4 elements
- CCNT now will be 5 – as there are 5 frames
- SRCCIDX skips to the next frame

	Source		= &loc_8
4 =	BCNT	ACNT	= 2
	Destination		= &myDest
2 =	DSTBIDX	SRCBIDX	= 2 (2 bytes going from block 8 to 9)
2 =	DSTCIDX	SRCCIDX	= 6 (3 elements from block 11 to 14)
		CCNT	= 5

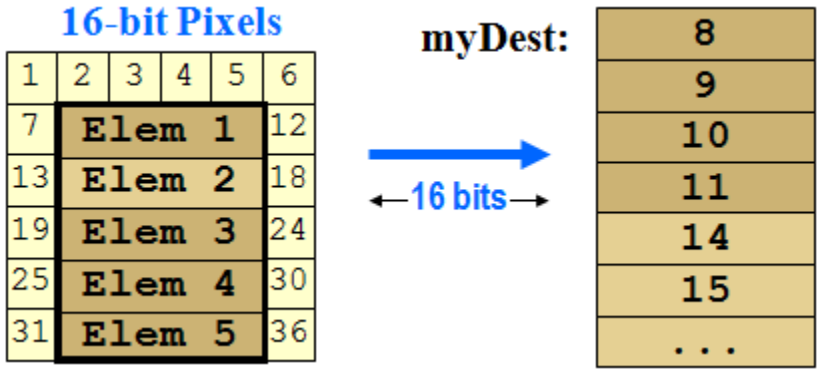
Note: Less efficient version

EDMA Example : Block Transfer

Goal:

Transfer a 5x4 subset

from loc_8 to myDest



- ACNT is defined here as the entire frame : 4 * 2 bytes
- BCNT is the number of frames : 5
- CCNT now will be 1
- SRCBIDX skips to the next frame

	Source		= &loc_8
5 =	BCNT	ACNT	= 8
	Destination		= &myDest
(4*2) is 8 =	DSTBIDX	SRCBIDX	= 12 is (6*2) (from block 8 to 14)
0 =	DSTCIDX	SRCCIDX	= 0
		CCNT	= 1

Initiating a DMA Transfer

- **Event-triggered transfer request**
  - A peripheral, system, or externally-generated event triggers a transfer request (typical use case)
- **Manually-triggered transfer request**
  - CPU manually triggers a transfer on a particular by writing a 1 to the event set registers
- **Chain-triggered transfer request**
  - A transfer is triggered on the completion of another transfer or sub-transfer.

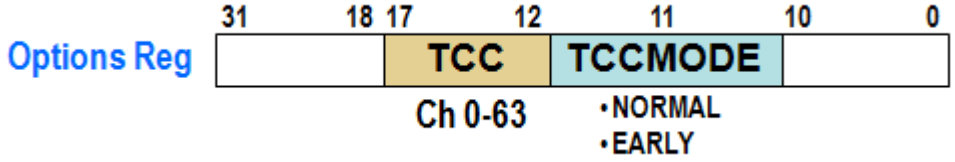
EDMA Channel Controller Regions

- Address space of TPCC is divided into eight regions
- Individual channel resources are assigned to a specific region
- Each region is typically assigned to a specific device or CPU.
- In a typical use case each region is associated to a core(M4, DSP, etc) the required DMA channels are assigned to the region and programmed for doing transfer.

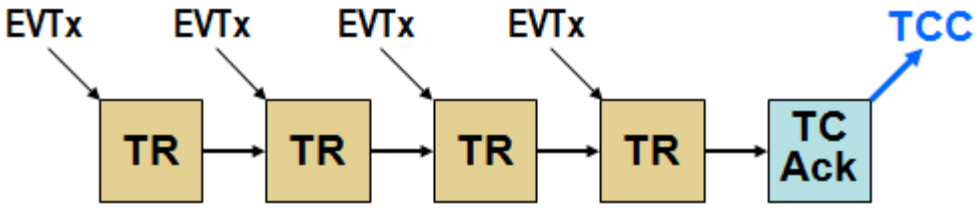
EDMA Interrupts

- There are two kinds of interrupts, Completion Interrupt and Error Interrupt.
- From Each of the region there is one completion interrupt and few error interrupts(not region specific)
- These interrupts are inputs to IRQ Crossbar and can be routed to any of the CPUs.

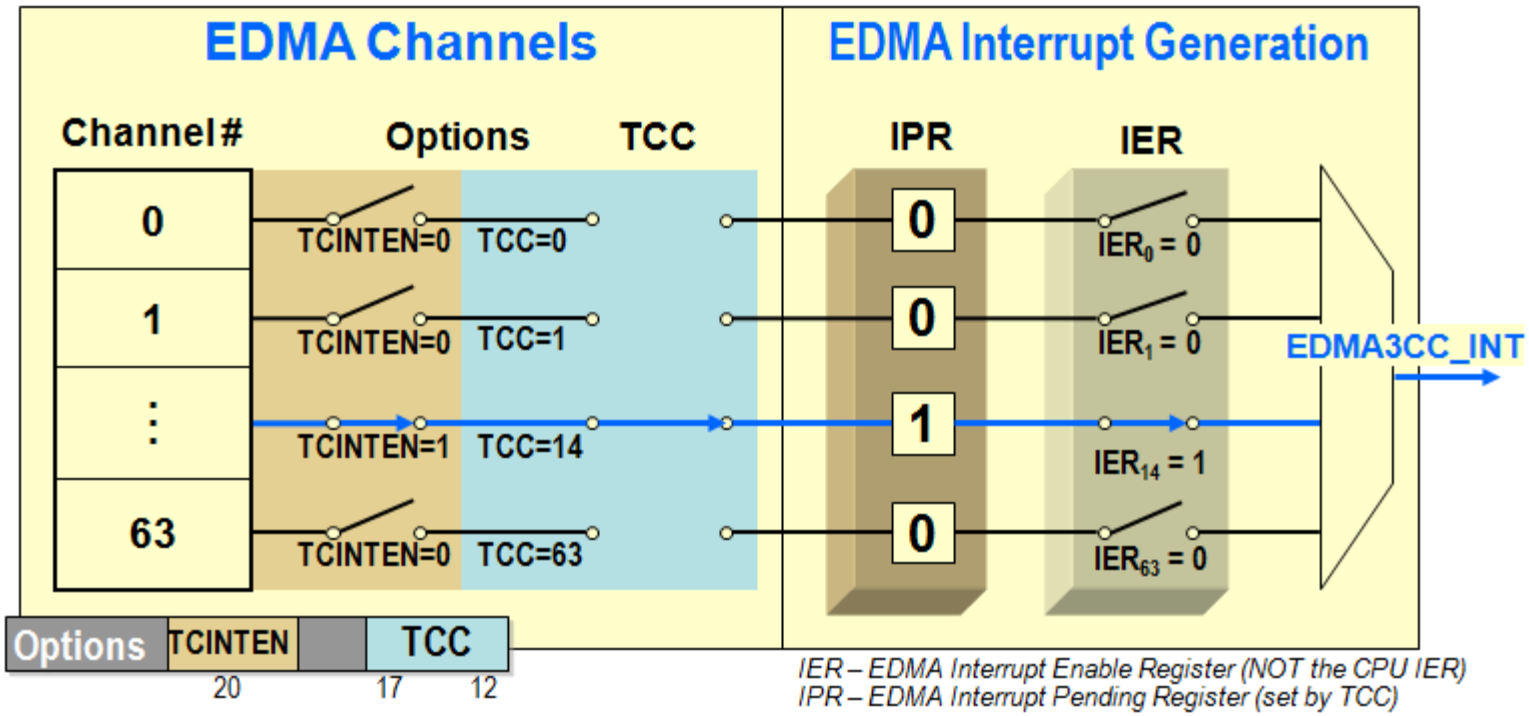
Transfer Complete Code (TCC)



- TCC is generated when a transfer completes. This is referred to as the “Final TCC”.
- TCC can be used to trigger an EDMA interrupt and/or another transfer (chaining)
- Each TR below is a “transfer request” which can be either ACNT bytes (A-sync) or ACNT * BCNT bytes (AB-sync). Final TCC only occurs after the LAST TR.



Generate EDMA Interrupt

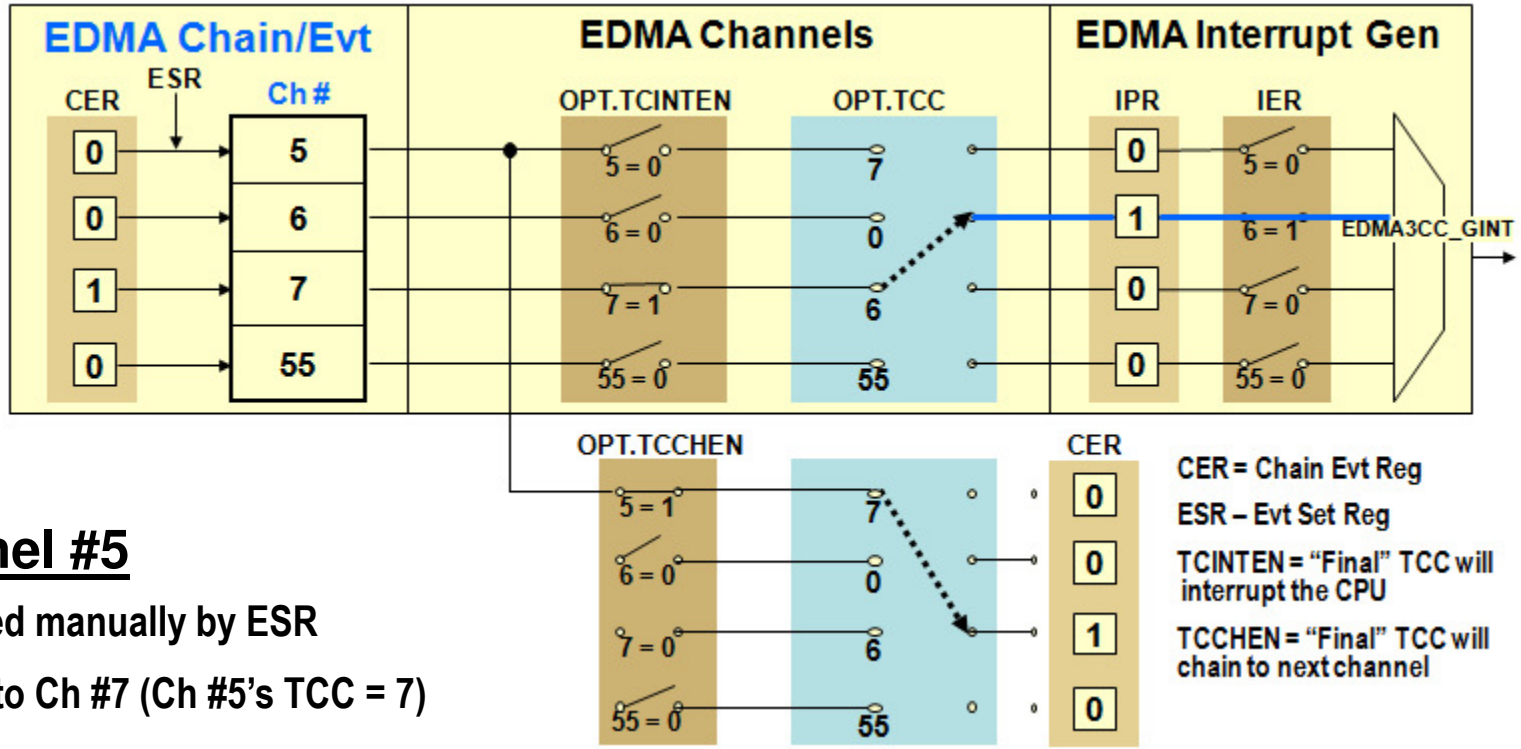


- When Final TR is finished and options register is set up to generate interrupt, interrupt is generated.
- TCC Field in Options register of PaRAM is used to set corresponding bit in IPR Register, which triggers interrupt to CPU.

Linking and Chaining Transfer

- Linking is a mechanism which allows the entire PaRAM set (Associated with DMA or QDMA Channel) to be reloaded from a location within the PaRAM memory map.
  - useful for maintaining ping-pong buffers, circular buffering, and repetitive/continuous transfers
- Chaining is a mechanism by which the completion of one transfer automatically sets the event for another channel.

Example – Simple Chaining



Channel #5

- Triggered manually by ESR
- Chains to Ch #7 (Ch #5's TCC = 7)

Channel #7

- Triggered by chaining from Ch #5
- Interrupts the CPU when finished (sets TCC = 6)
- ISR checks IPR (TCC=6) to determine which channel generated the interrupt

Notes:

- Any Ch can chain to any other Ch by enabling OPT.TCCHEN and specifying the next TCC
- Any Ch can interrupt the CPU by enabling its OPT.TCINTEN option (and specifying the TCC)
- IPR bit set depends on previous Ch's TCC setting

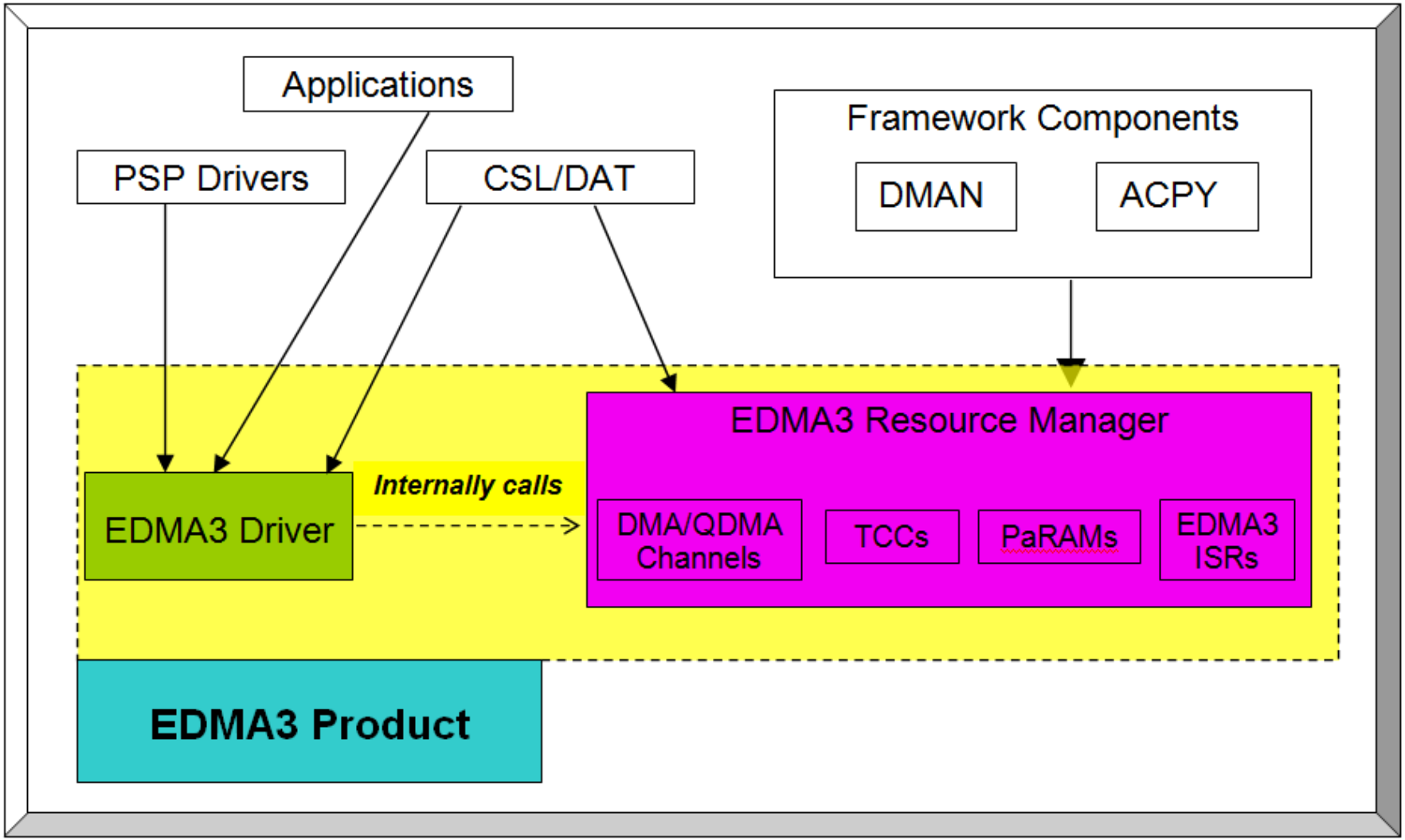
Mapping events to DMA Channels

- DMA Crossbar module (Part of Control module) is used to map the Events from peripherals to DMA Channels.
- DMA Events from peripherals are connected to the input of DMA Crossbar, and output of DMA Crossbar is connected to EDMA Events.
- DMA Crossbar can be programmed to connect any of its inputs to any of its outputs.

Agenda

- EDMA3 IP Overview
- **EDMA3 S/W Architecture**
  - Directory Structure and Code walkthrough
  - Build & Run Examples

EDMA3LLD Architecture



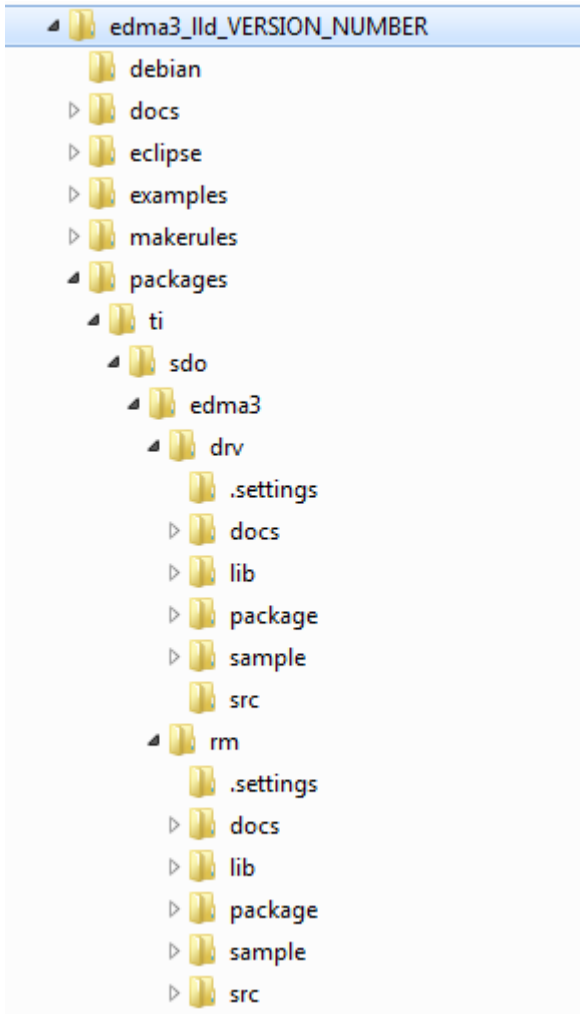
EDMA3LLD Overview

- The EDMA3LLD (Low Level Driver) is designed to be OS agnostic
- The LLD is divided mainly in to Driver (Drv) and Resource Manager (RM) and OS abstraction layers
- RM is used mainly for allocating and managing the resources like DMA Channels, PaRAM, TCC etc. and Interrupt handling
- Driver is used for providing uniform interface for application for all types of transfers and error checking

EDMA3LLD Overview Contd.

- DRV is dependent on RM for managing EDMA resources
- DRV/RM exposes few APIs for registering interrupts and Semaphores for protecting global data structures and expects the proper implementation from application.
- Each of Drv and RM have a sample OS abstraction library which has the implementation for Interrupt and semaphores using TI RTOS (SYSBIOS)

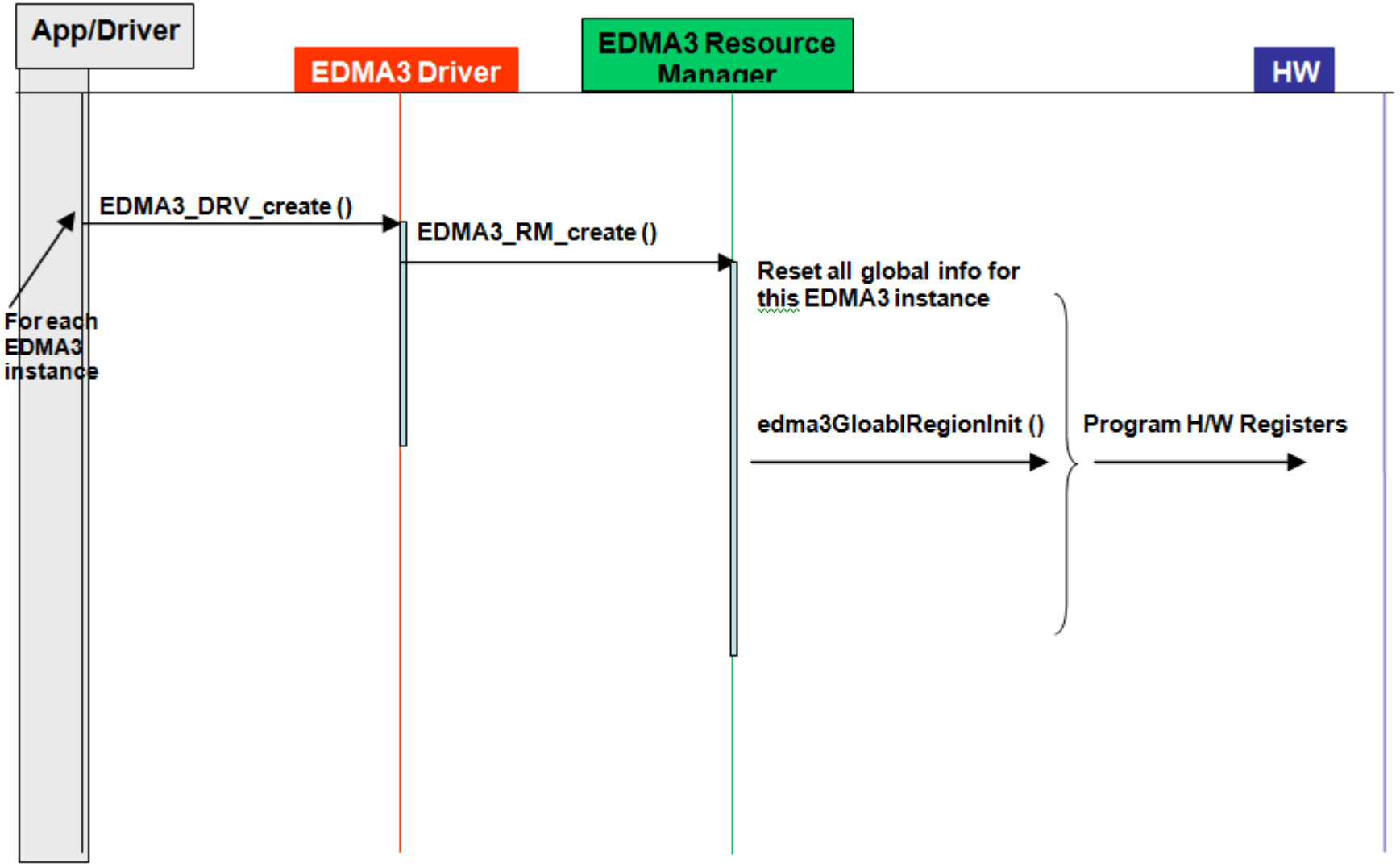
EDMA3LLD Directory Structure



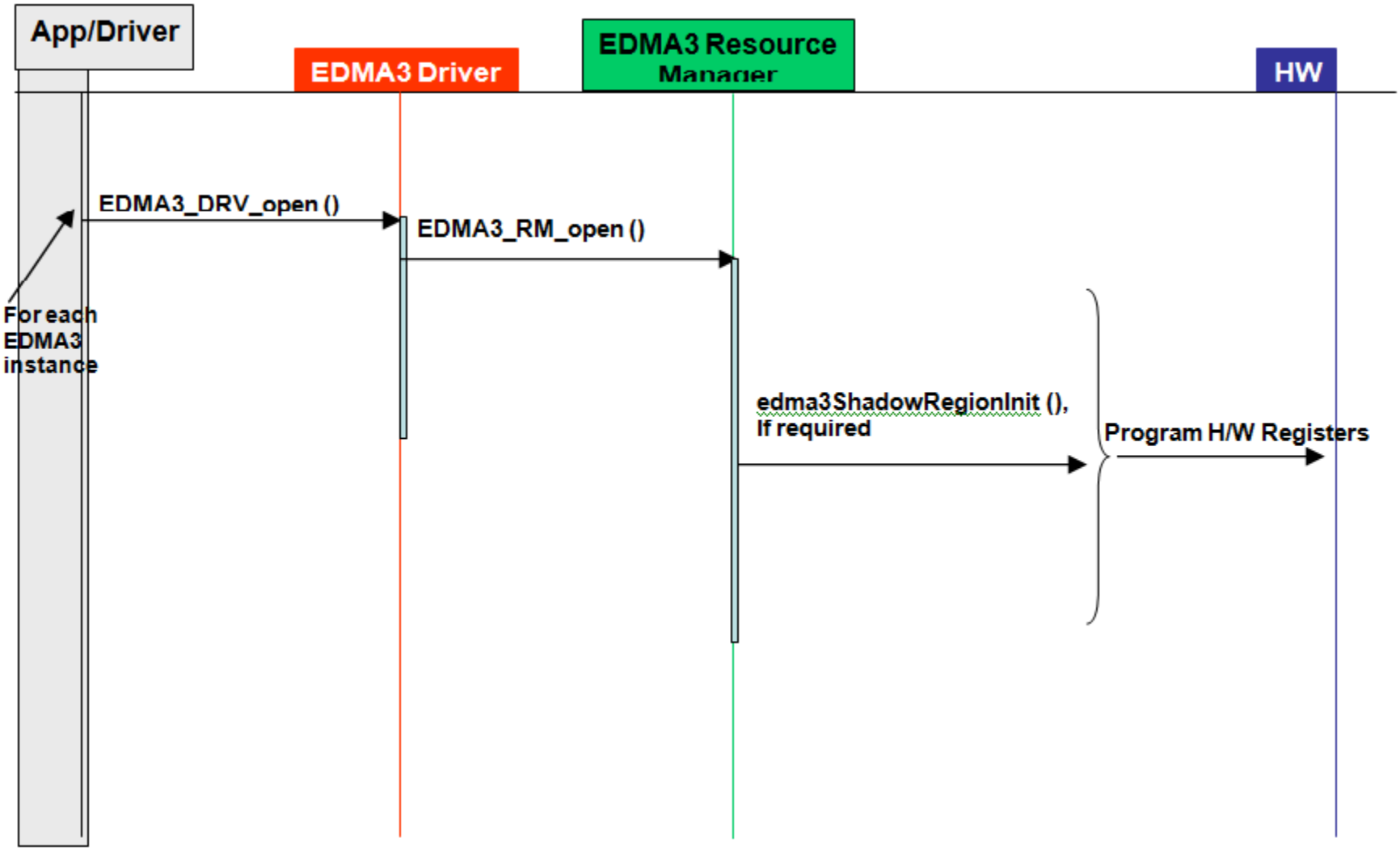
EDMA3 Driver Initialization

- EDMA3 Driver should be initialized first before it can be used by the peripheral drivers or application
- Call *EDMA3_DRV_create()* to create EDMA3 driver object and provide global configuration parameters
- Call *EDMA3_DRV_open()* to open a region specific EDMA3 driver instance. This returns the handle which should be used for further APIs

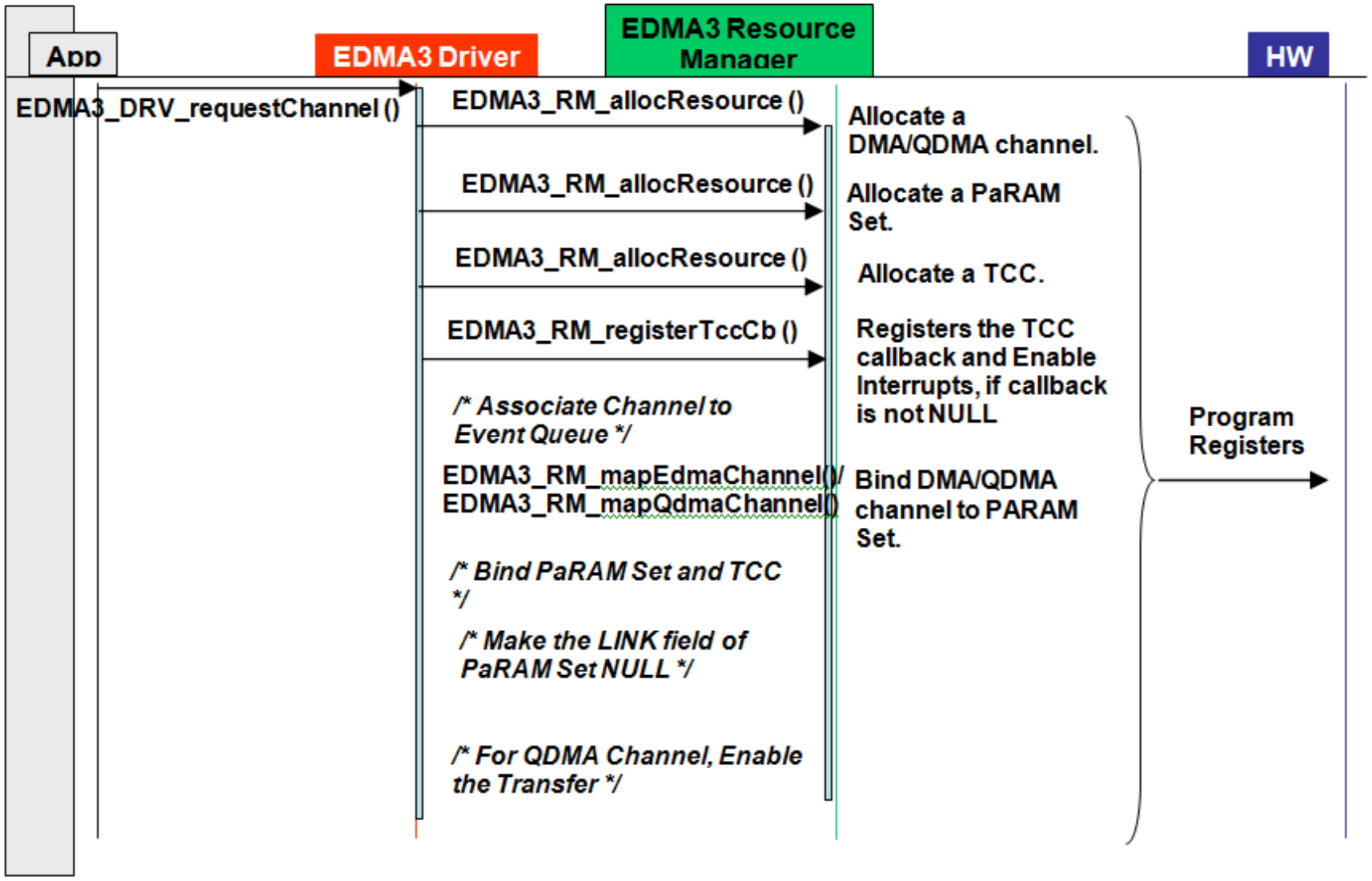
API Flow Driver Creation



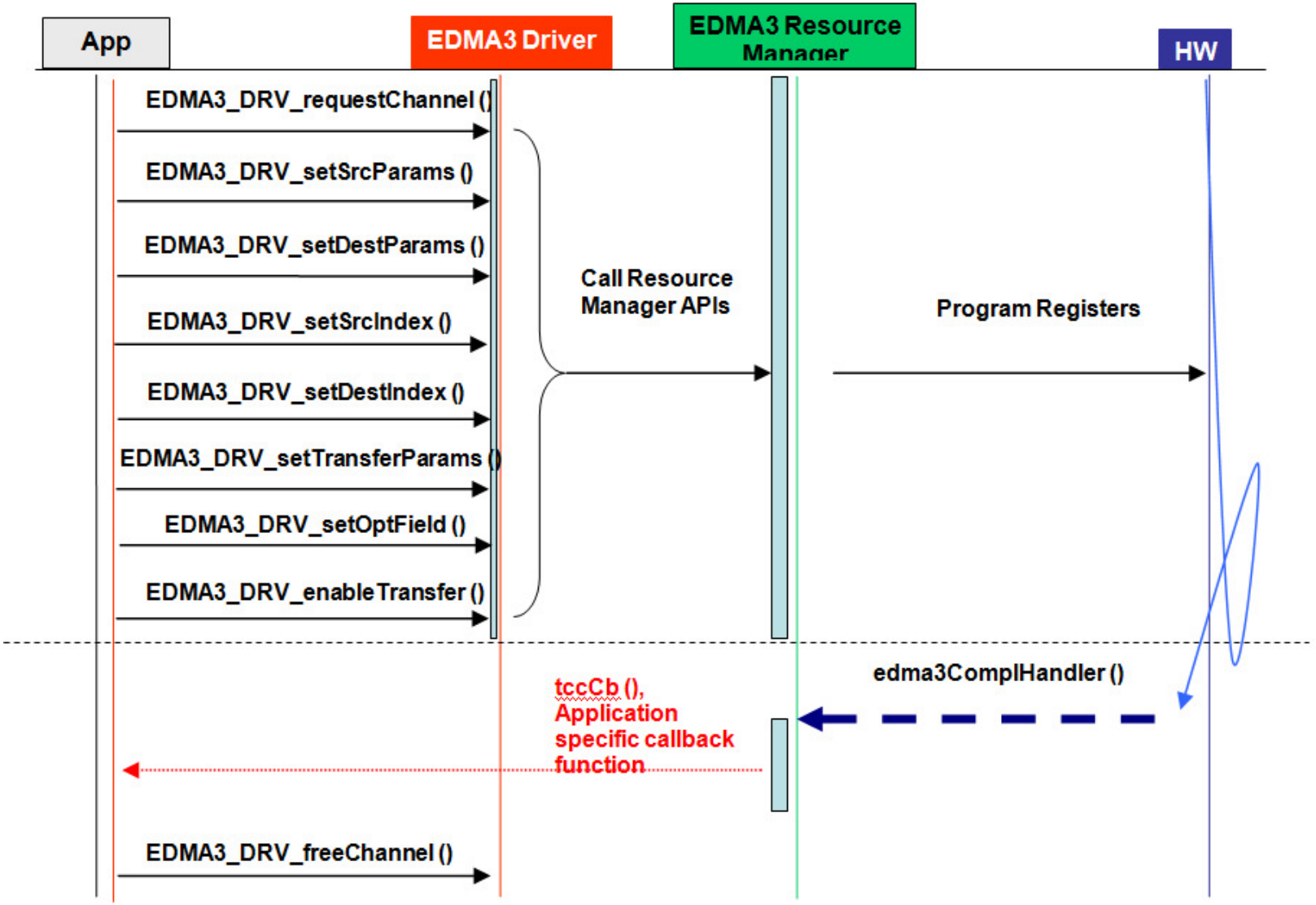
API Flow Open Driver Instance



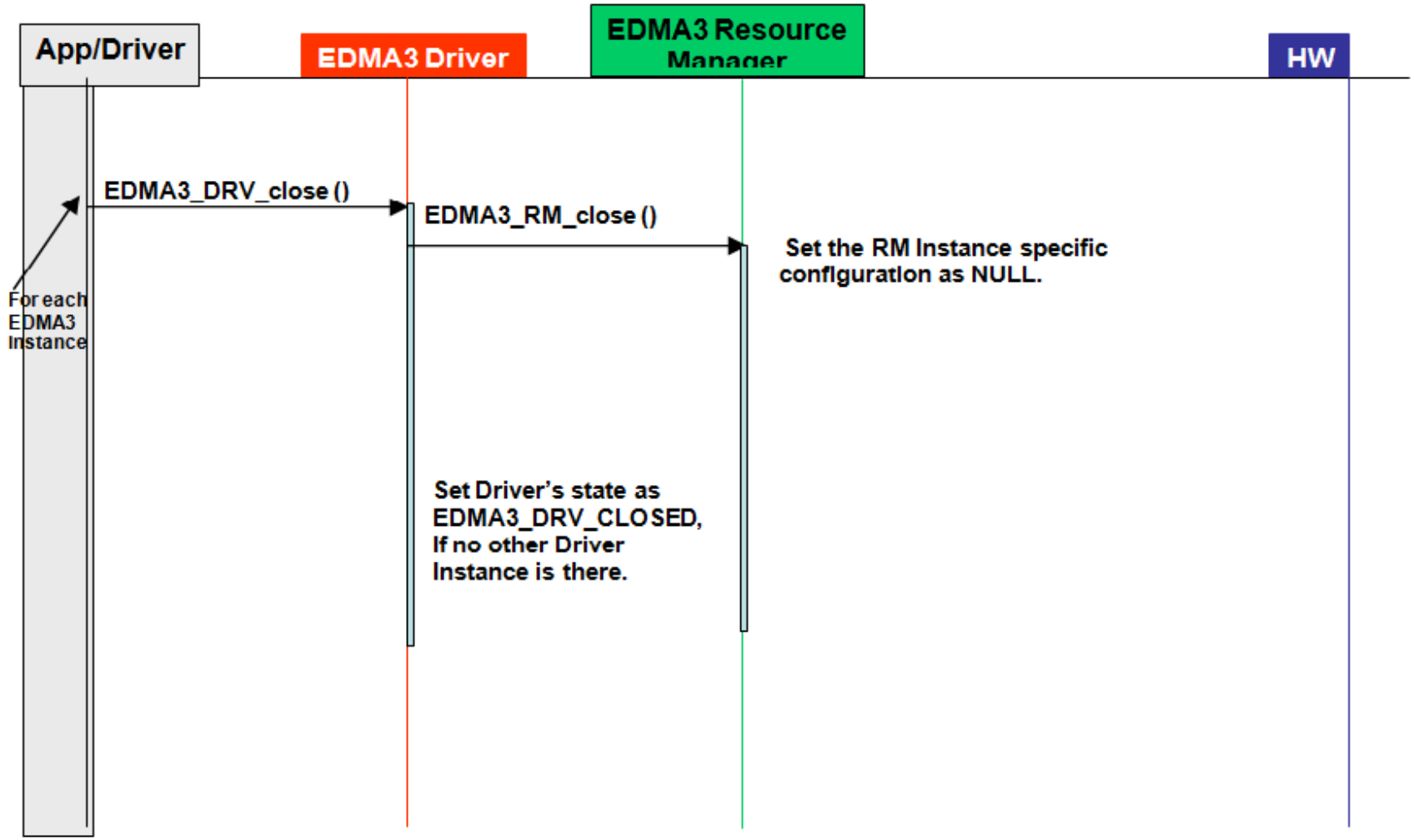
API Flow Request Channel



API Flow DMA Transfer



API Flow Driver Close



Building EDMA3 Libs and Examples

- Updated makerules\env.mk with tool chain paths
- Set Env Variable “ROOTDIR” with edma3lld install path
- Set Env Variable “PATH” with xdc tools which has the make binary
- To build libs issue following command
  - gmake libs FORMAT=ELF
- To build examples issue following command
  - gmake examples FORMAT=ELF

Building EDMA3 Libs and Examples

- optionally *PLATFORM=<platform name>* can be provided in the build command to build only for a specific platform.
  - gmake examples PLATFORM=tda2xx-evm
- Examples provided in the package performs memory to memory (DDR) transfers with features provided by EDMA3 like linking, chaining etc.
- Connect to CCS load program on to the core and run the sample examples. This prints test status on the CCS Console.

Questions?
Thank You