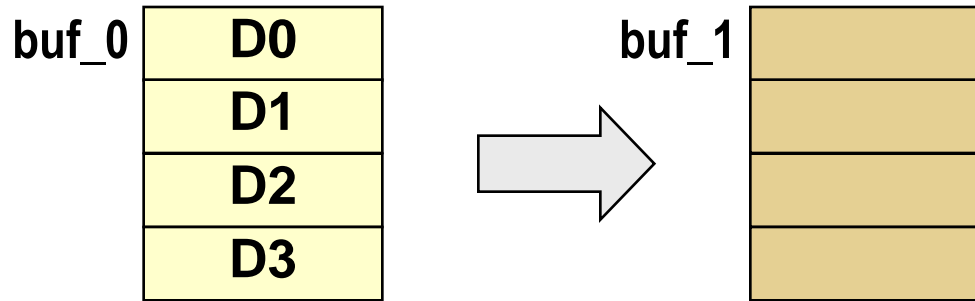


# **EDMA3, QDMA and IDMA for the Keystone Platform**

# Outline

- ◆ **Introduction to EDMA3**
- ◆ **Example 1: Single Block Transfer**
- ◆ **Programming EDMA3 with CSL 3.0**
- ◆ **Example 2: Multiple Block Transfer**
- ◆ **Linking vs. Chaining**
- ◆ **QDMA**
- ◆ **IDMA**

# Why Use DMA?



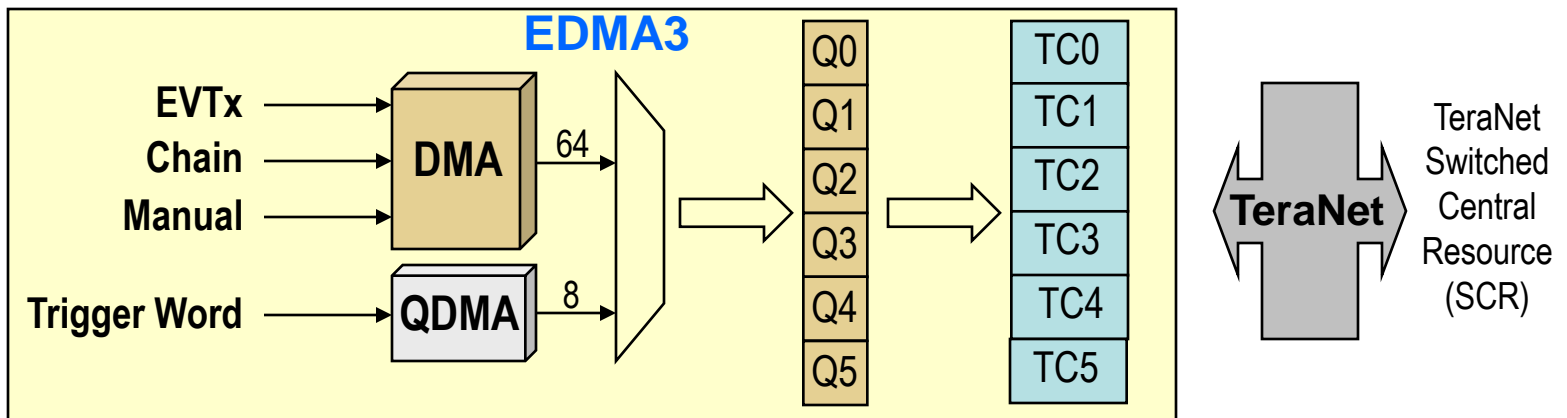
- ◆ The primary function of DMA is to move data without direct CPU involvement
- ◆ What information does a DMA controller need to perform a transfer?
  - Source address
  - Destination address
  - Length (or size)
- ◆ What options might be useful to perform the transfer?
  - Do you want to interrupt the CPU when the transfer is complete?
  - Is this transfer synchronized to an event (like the McBSP RCV buffer is full)?
  - How do the source and destination addresses update? (same, +1, -1, +4 ?)

# What are DMA and EDMA3 ?

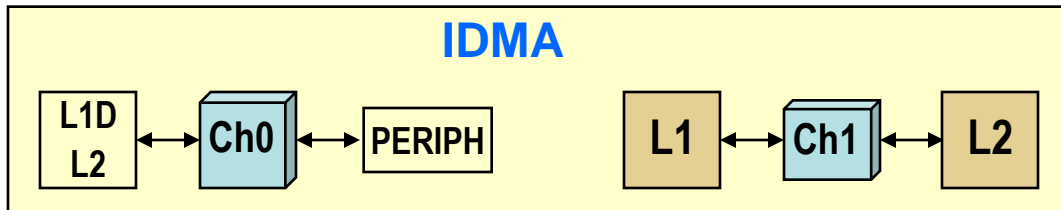
➤ When we say “DMA”, what do we mean? Well, there are MANY forms of “DMA” (Direct Memory Access) on this device:

- **EDMA3** – “Enhanced” DMA handles 64 DMA CHs and 8 QDMA CHs

- ✓ DMA – 64 channels that can be triggered manually or by events/chaining
- ✓ QDMA – 8 channels of “Quick” DMA triggered by writing to a “trigger word”



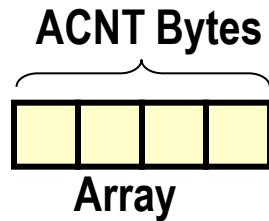
- **IDMA** – 2 CHs of “Internal” DMA (Periph Cfg, Xfr L1 ↔ L2)



- **Peripheral “DMA”s** – Each master device hooked to the TeraNet Switched Central Resource (SCR) has its own DMA (e.g. SRIO, EMAC, etc.)

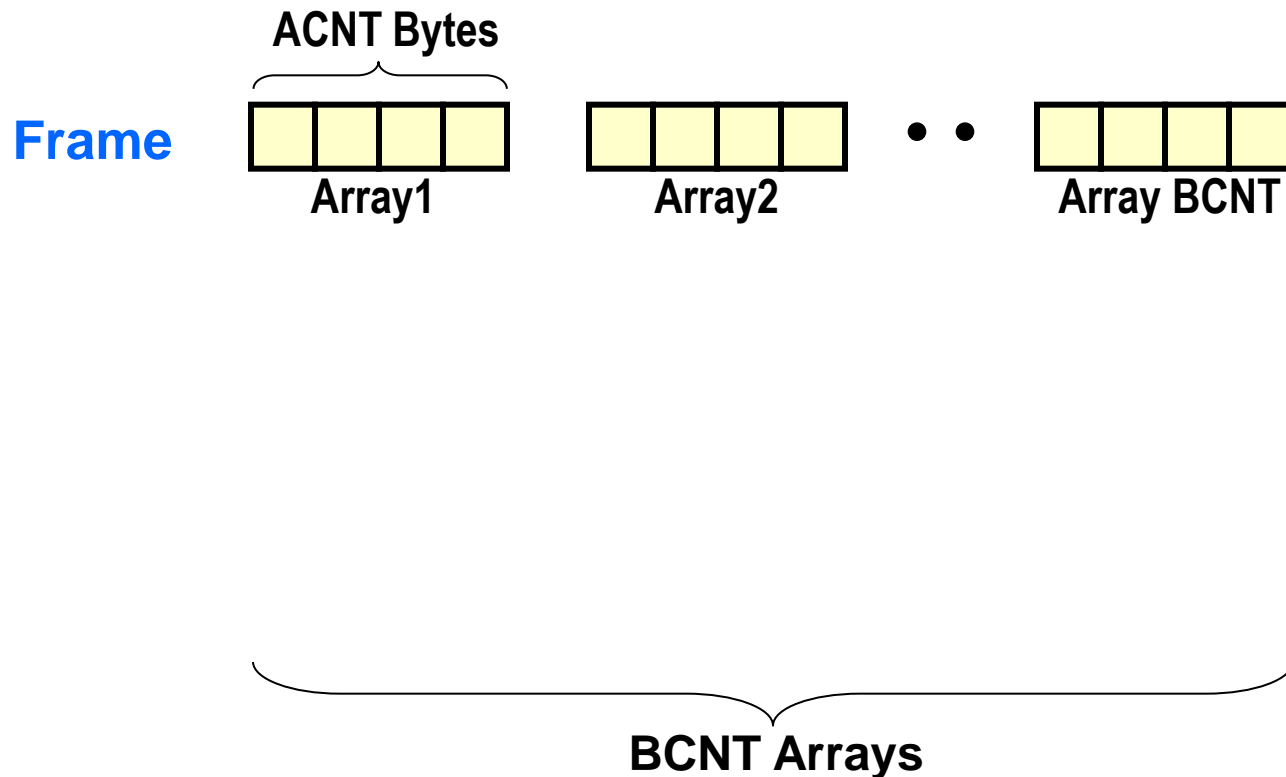
# EDMA3 Terminology

- ◆ 3-dimensional transfer consisting of ACNT, BCNT and CCNT:
  - ACNT = Array = # of contiguous ACNT bytes (16-bit unsigned, 0-65535)
- ◆ Minimum transfer is an array of ACNT bytes



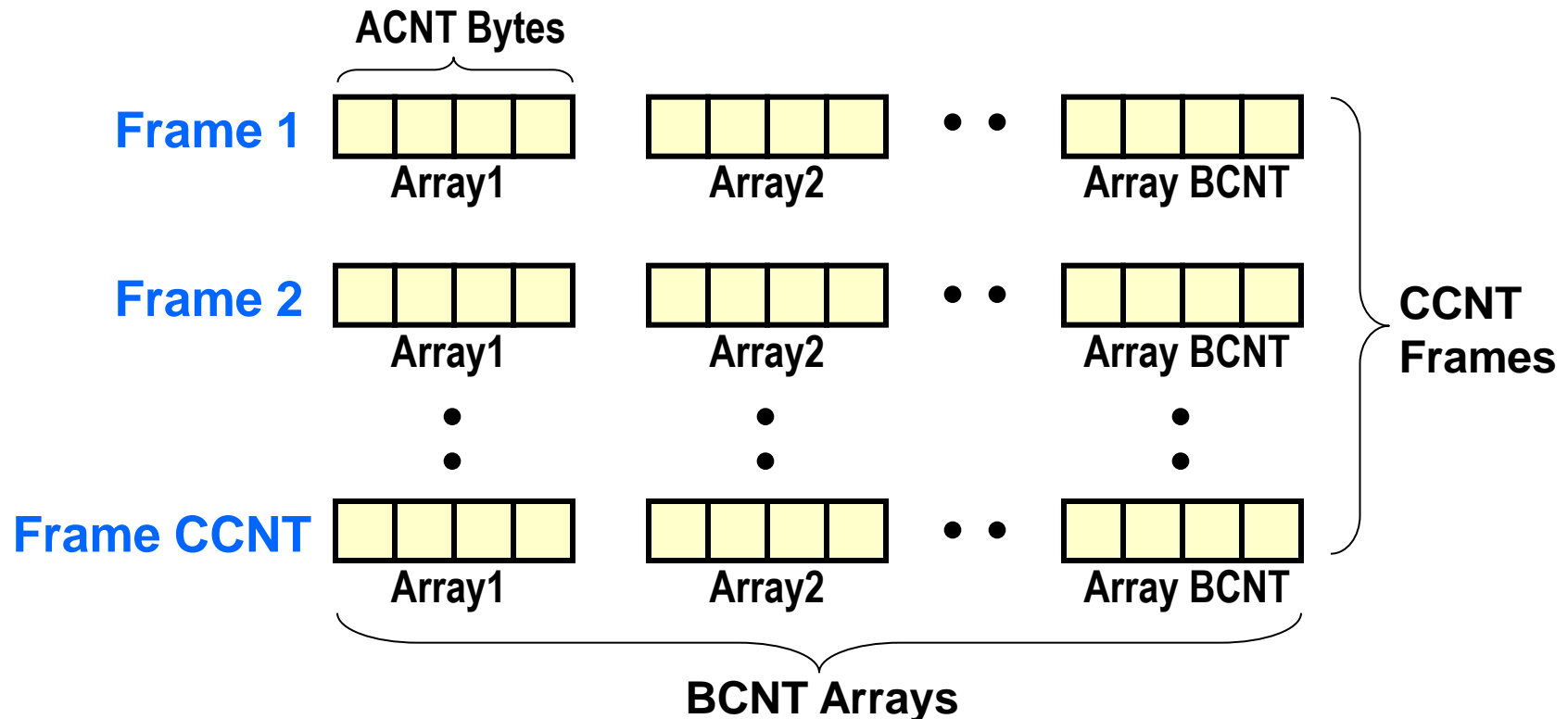
# EDMA3 Terminology

- ◆ 3-dimensional transfer consisting of ACNT, BCNT and CCNT:
  - ACNT = Array = # of contiguous ACNT bytes (16-bit unsigned, 0-65535)
  - BCNT = Frame = # of ACNT arrays (16-bit unsigned, 0-65535)
- ◆ Minimum transfer is an array of ACNT bytes



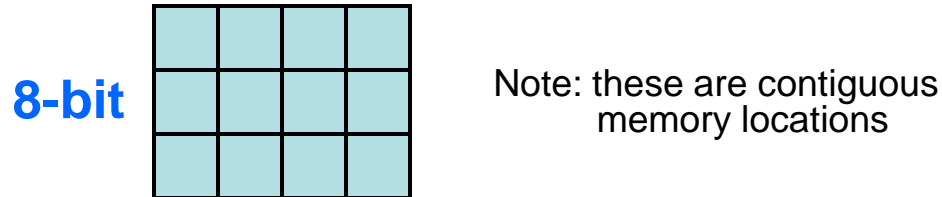
# EDMA3 Terminology

- ◆ 3-dimensional transfer consisting of ACNT, BCNT and CCNT:
  - ACNT = Array = # of contiguous ACNT bytes (16-bit unsigned, 0-65535)
  - BCNT = Frame = # of ACNT arrays (16-bit unsigned, 0-65535)
  - CCNT = Block = # of BCNT frames (16-bit unsigned, 0-65535)
- ◆ Minimum transfer is an array of ACNT bytes
- ◆ Total transfer count =  $ACNT * BCNT * CCNT$



# Example: How Do You VIEW the Transfer?

- ◆ Let's start with a simple example – or is it simple?
- ◆ We need to transfer 12 bytes from “here” to “there”.



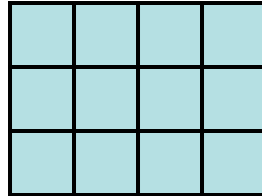
- ◆ What are ACNT, BCNT and CCNT?



# Example: How Do You VIEW the Transfer?

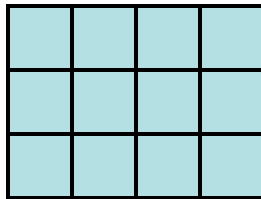
- ◆ Let's start with a simple example – or is it simple?
- ◆ We need to transfer 12 bytes from “here” to “there”.

8-bit



Note: these are contiguous memory locations

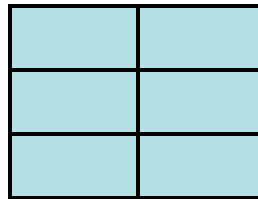
- ◆ What are ACNT, BCNT and CCNT?
- ◆ You can “view” the transfer several ways:



ACNT = 1

BCNT = 4

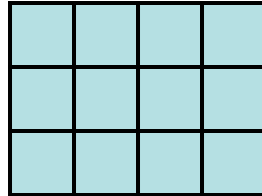
CCNT = 3



# Example: How Do You VIEW the Transfer?

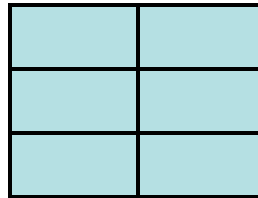
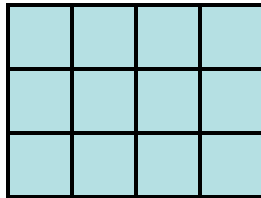
- ◆ Let's start with a simple example – or is it simple?
- ◆ We need to transfer 12 bytes from “here” to “there”.

8-bit



Note: these are contiguous memory locations

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



ACNT = 2

BCNT = 2

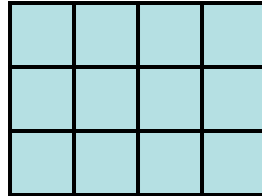
CCNT = 3



# Example: How Do You VIEW the Transfer?

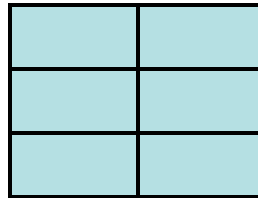
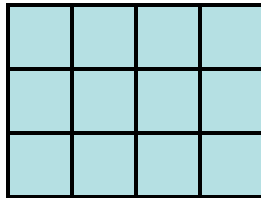
- ◆ Let's start with a simple example – or is it simple?
- ◆ We need to transfer 12 bytes from “here” to “there”.

8-bit



Note: these are contiguous memory locations

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



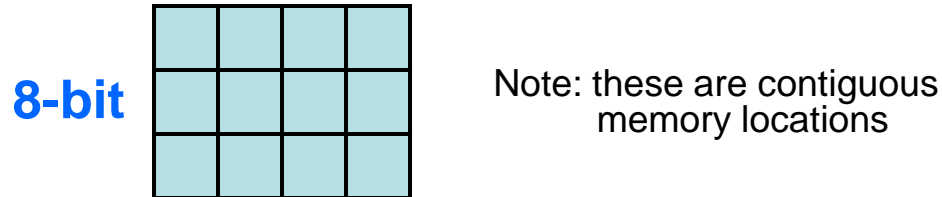
ACNT = 12

BCNT = 1

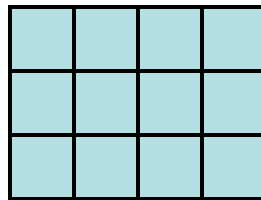
CCNT = 1

# Example: How Do You VIEW the Transfer?

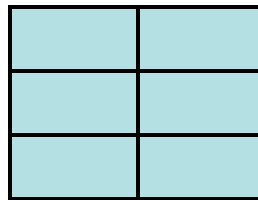
- ◆ Let's start with a simple example – or is it simple?
- ◆ We need to transfer 12 bytes from “here” to “there”.



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



ACNT = 1  
BCNT = 4  
CCNT = 3



ACNT = 2  
BCNT = 2  
CCNT = 3

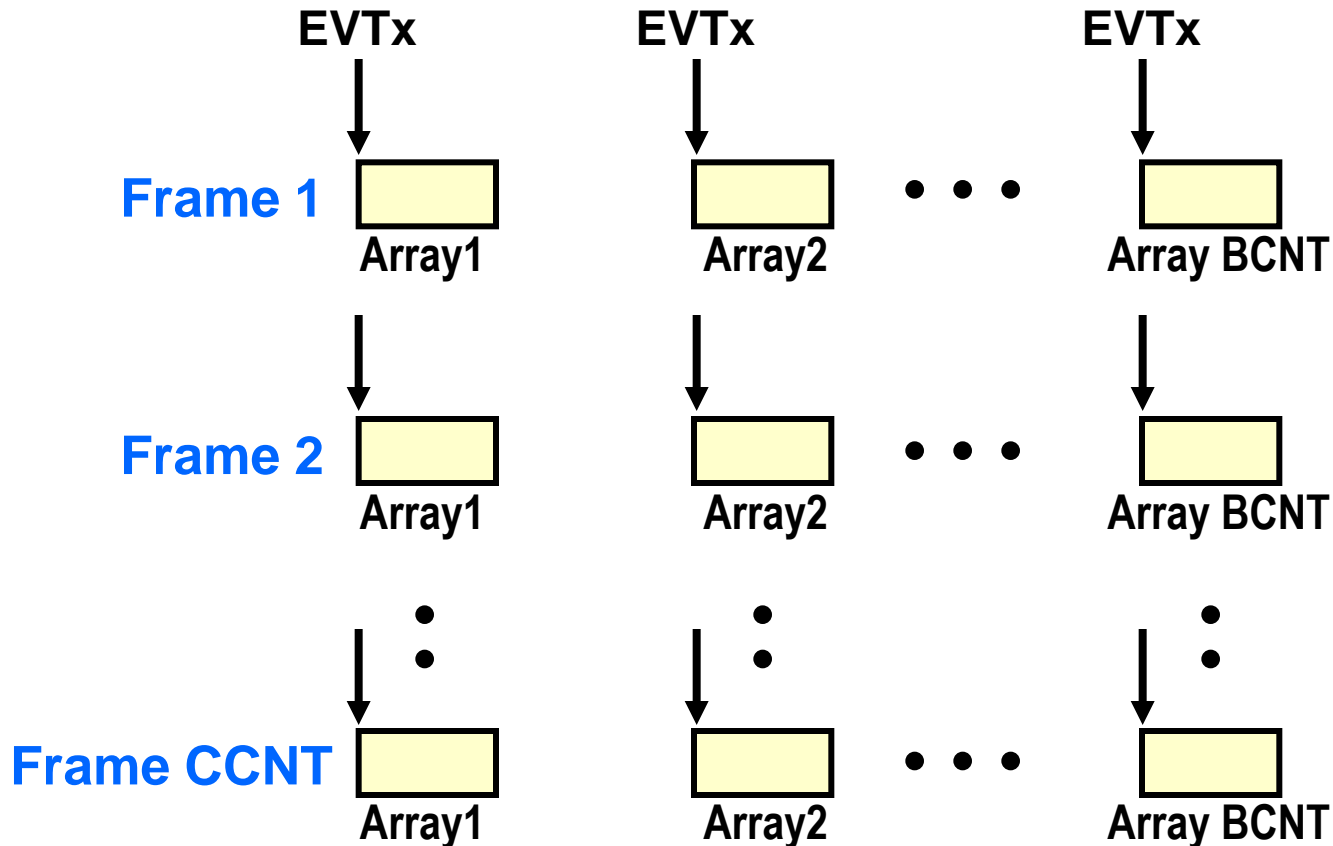


ACNT = 12  
BCNT = 1  
CCNT = 1

- ◆ Which “view” is the best? Well, that depends on what your system needs and the type of synchronization...

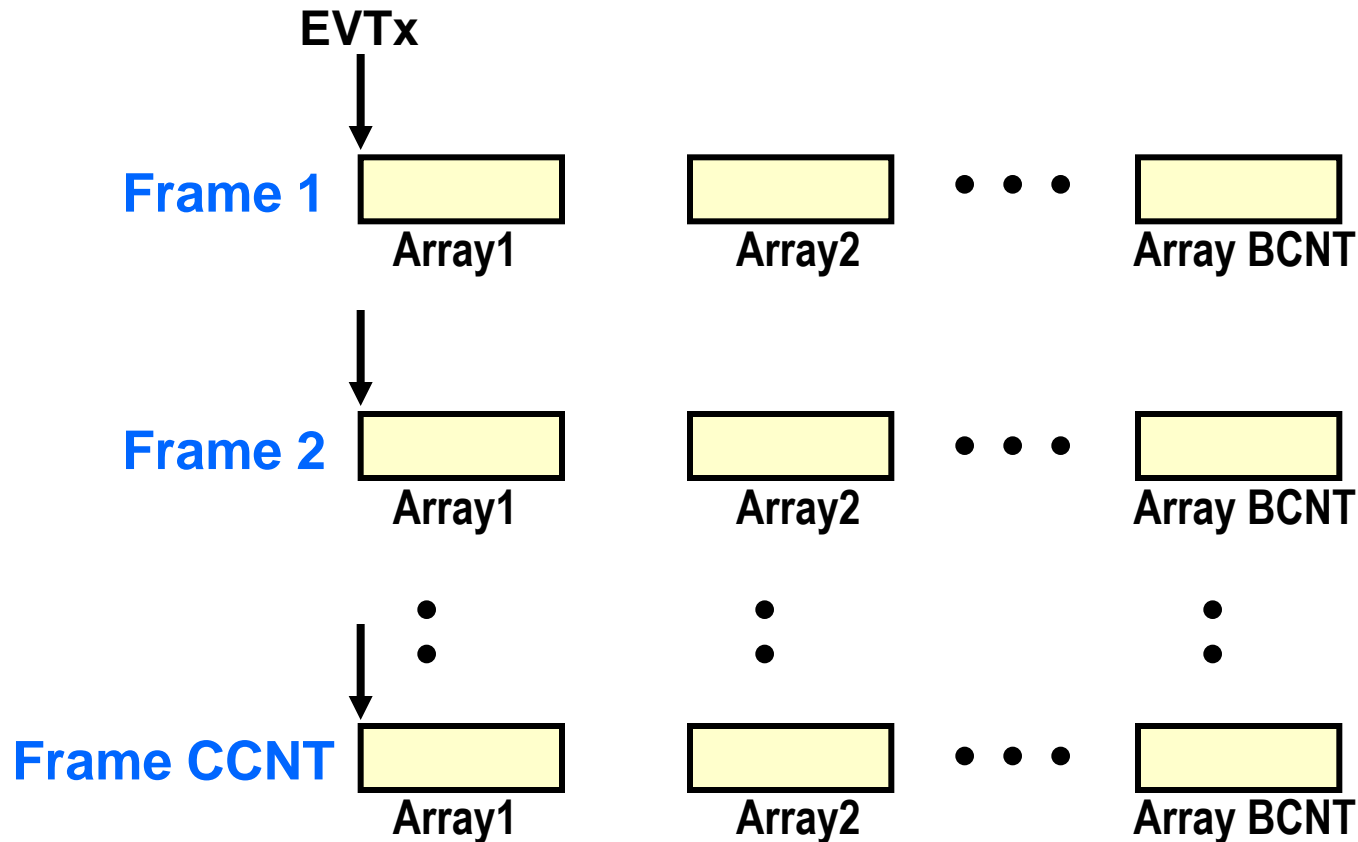
# “A” Synchronization

- ◆ An event (like the McBSP receive register full), triggers the transfer of exactly 1 array of ACNT bytes (2 bytes)
- ◆ Example: McBSP tied to a codec (you want to sync each transfer of a 16-bit word to the receive buffer being full or the transmit buffer being empty).



# “AB” Synchronization

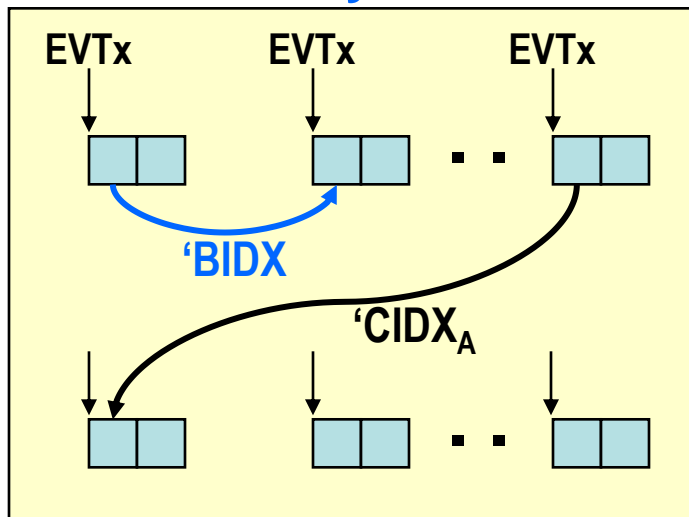
- ◆ An event triggers a two-dimensional transfer of BCNT arrays of ACNT bytes (A\*B)
- ◆ Example: Line of video pixels (each line has BCNT pixels consisting of 3 bytes each – Y, Cb, Cr)



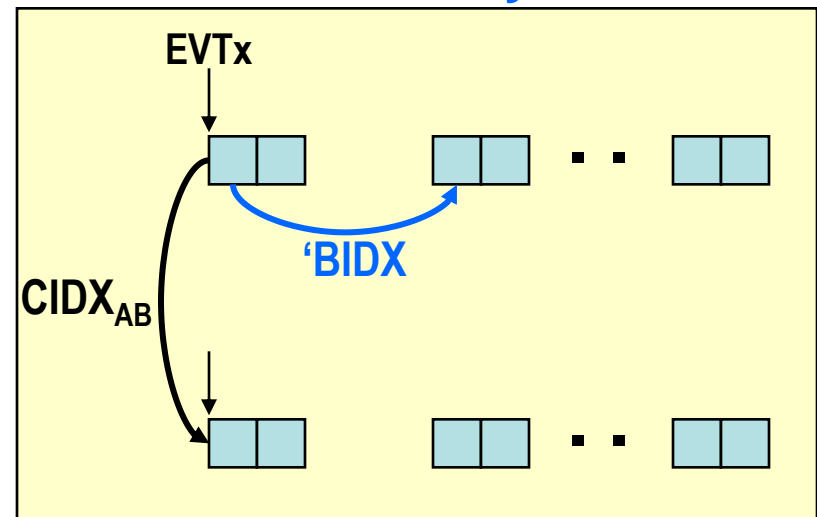
# Indexing: 'BIDX & 'CIDX

- ◆ EDMA3 has two types of indexing: 'BIDX and 'CIDX
- ◆ Each index can be set separately for SRC and DST (next slide...)
- ◆ 'BIDX = index in bytes between ACNT arrays (same for A-sync and AB-sync)
- ◆ 'CIDX = index in bytes between BCNT frames (different for A-sync vs. AB-sync)
- ◆ 'BIDX/'CIDX: signed 16-bit, -32768 to +32767

A-Sync



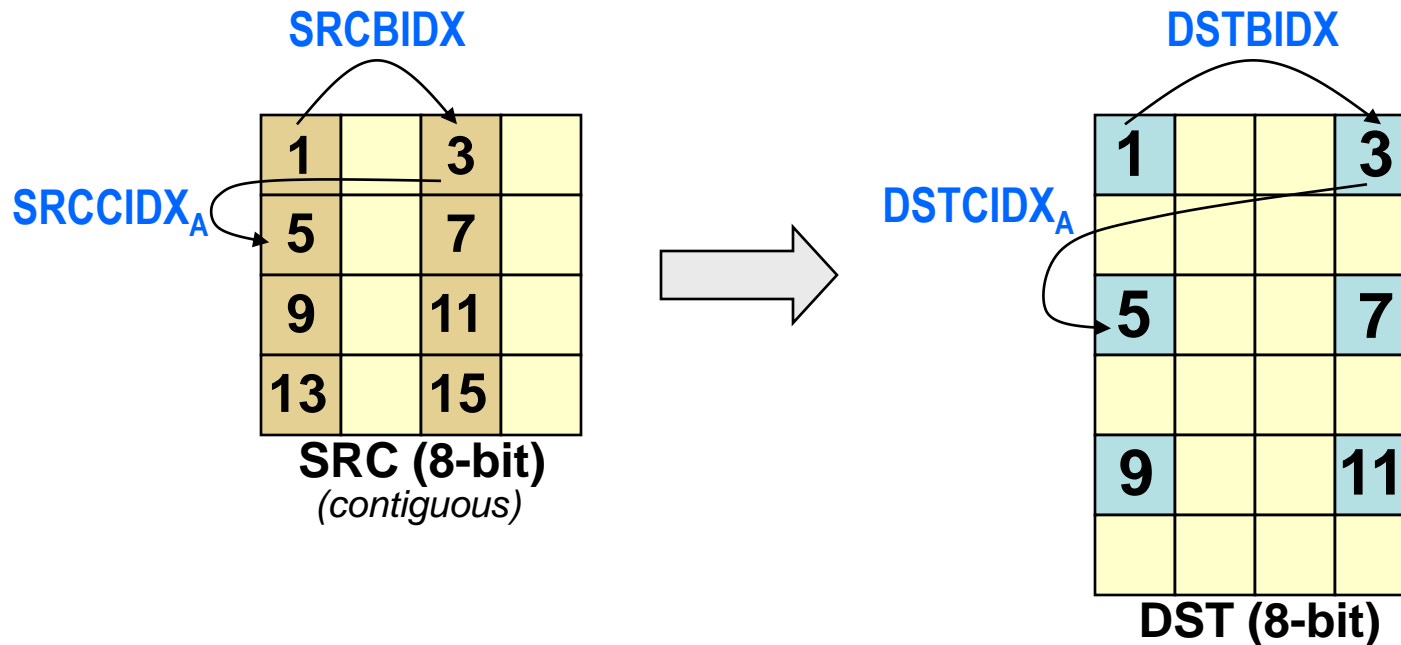
AB-Sync



- ◆ 'CIDX distance is calculated from the starting address of the previously transferred block (array for A-sync, frame for AB-sync) to the next frame to be transferred.

# Indexed Transfers

- ◆ EDMA3 has four indexes allowing higher flexibility for complex transfers:
  - **SRCBIDX** = # bytes between arrays (Ex: SRCBIDX = 2)
  - **SRCCIDX** = # bytes between frames (Ex:  $\text{SRCCIDX}_A = 2$ ,  $\text{SRCCIDX}_{AB} = 4$ )
  - Note: 'CIDX depends on the synchronization used – “A” or “AB”
  - **DSTBIDX** = # bytes between arrays (Ex: DSTBIDX = 3)
  - **DSTCIDX** = # bytes between frames (Ex:  $\text{DSTCIDX}_A = 5$ ,  $\text{DSTCIDX}_{AB} = 8$ )

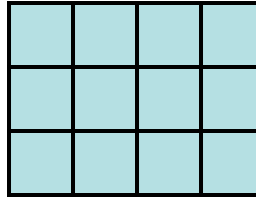




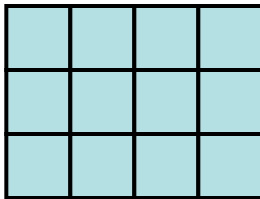
# Example: Using Indexing

- ◆ Remember this example? Fill in the proper SOURCE index values for each “view” below:

8-bit



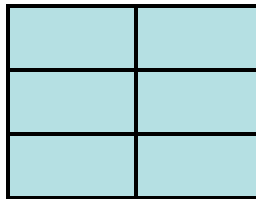
Note: these are contiguous memory locations



ACNT = 1

BCNT = 4

CCNT = 3



ACNT = 2

BCNT = 2

CCNT = 3



ACNT = 12

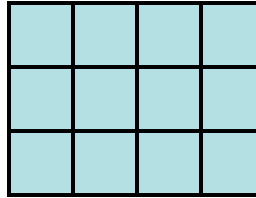
BCNT = 1

CCNT = 1

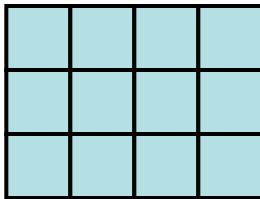
# Example: Using Indexing

- ◆ Remember this example? Fill in the proper SOURCE index values for each “view” below:

8-bit



Note: these are contiguous memory locations



ACNT = 1

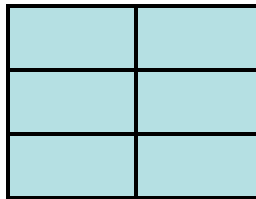
BCNT = 4

CCNT = 3

'BIDX = 1

'CIDX<sub>A</sub> = 1

'CIDX<sub>AB</sub> = 4



ACNT = 2

BCNT = 2

CCNT = 3



ACNT = 12

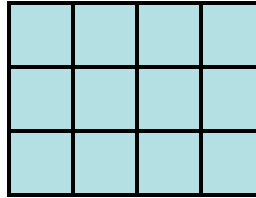
BCNT = 1

CCNT = 1

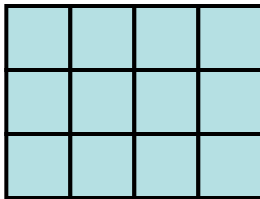
# Example: Using Indexing

- ◆ Remember this example? Fill in the proper SOURCE index values for each “view” below:

8-bit



Note: these are contiguous memory locations



ACNT = 1

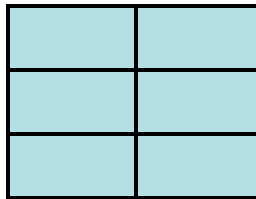
BCNT = 4

CCNT = 3

'BIDX = 1

'CIDX<sub>A</sub> = 1

'CIDX<sub>AB</sub> = 4



ACNT = 2

BCNT = 2

CCNT = 3

'BIDX = 2

'CIDX<sub>A</sub> = 2

'CIDX<sub>AB</sub> = 4



ACNT = 12

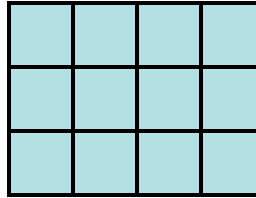
BCNT = 1

CCNT = 1

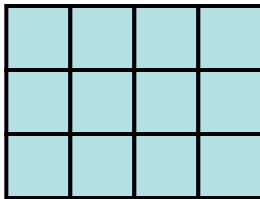
# Example: Using Indexing

- ◆ Remember this example? Fill in the proper SOURCE index values for each “view” below:

8-bit



Note: these are contiguous memory locations



ACNT = 1

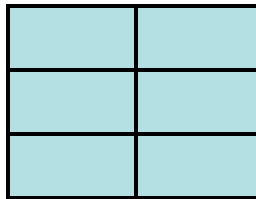
BCNT = 4

CCNT = 3

'BIDX = 1

'CIDX<sub>A</sub> = 1

'CIDX<sub>AB</sub> = 4



ACNT = 2

BCNT = 2

CCNT = 3

'BIDX = 2

'CIDX<sub>A</sub> = 2

'CIDX<sub>AB</sub> = 4



ACNT = 12

BCNT = 1

CCNT = 1

'BIDX = N/A

'CIDX<sub>A</sub> = N/A

'CIDX<sub>AB</sub> = N/A

# EDMA3 Parameter RAM Sets (PSETS)

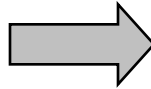
- ◆ EDMA3 has 256 Parameter RAM sets (PSETs) that contain configuration information about a transfer
- ◆ 64 DMA CHs and 8 QDMA CHs can be mapped to any one of the 256 PSETs and then triggered to run (by various methods)

64 DMA CHs

0
⋮
63

8 QDMA CHs

0
⋮
3



PaRAM Set 0
PaRAM Set 1
⋮
PSET 63
PSET 64
⋮
PSET 255

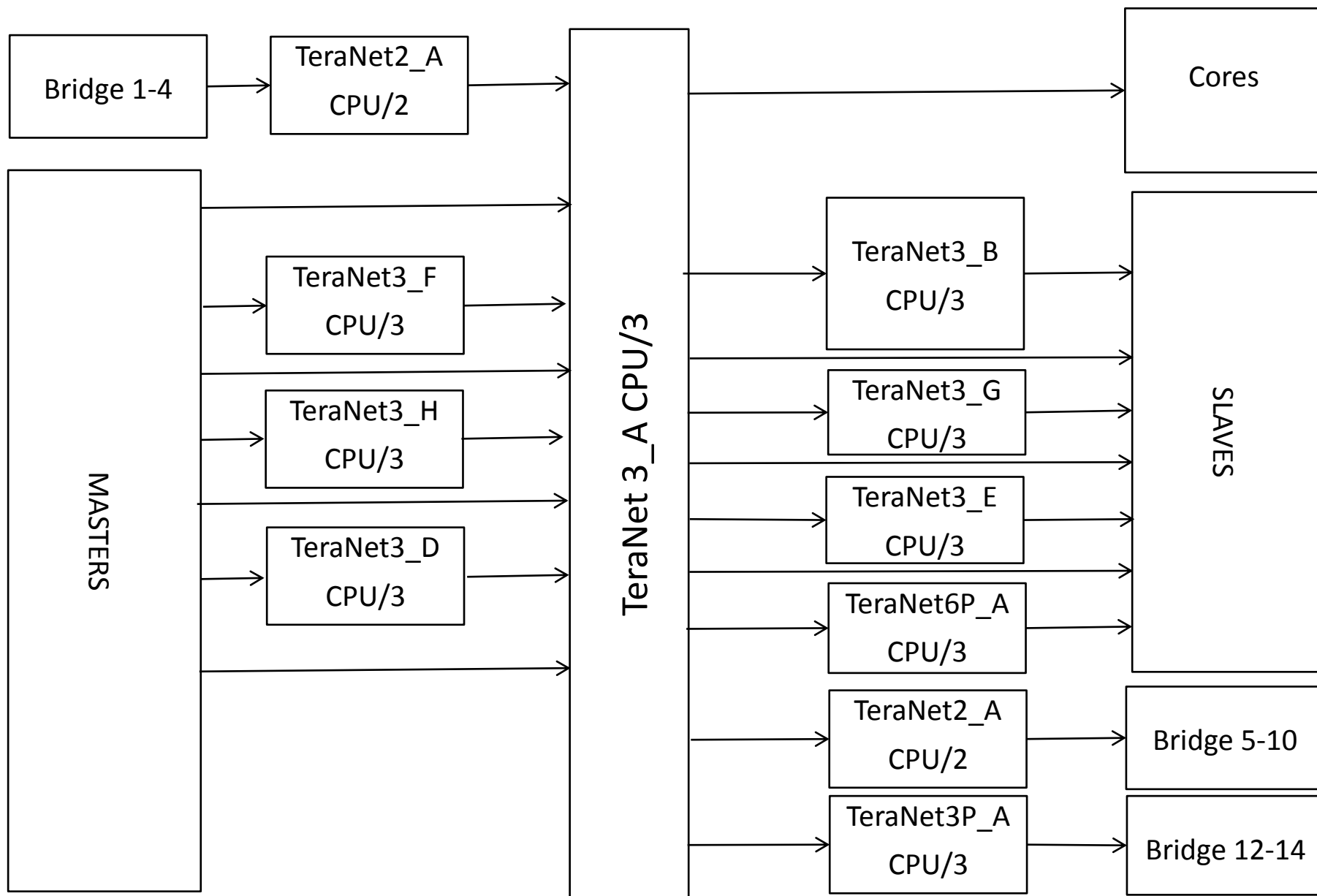
Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT
31	0

- ◆ Each PSET contains 12 register fields:

- Options (interrupt, chaining, sync mode, etc)
- SRC/DST addresses
- ACNT/BCNT/CCNT (size of transfer)
- Four SRC/DST Indexes
- BCNTRLD (BCNT reload for 3D xfrs)
- LINK (pointer to another PSET)

*Note: PSETs are dedicated EDMA RAM (not part of IRAM)*

# TeraNet Switch Fabric Connections



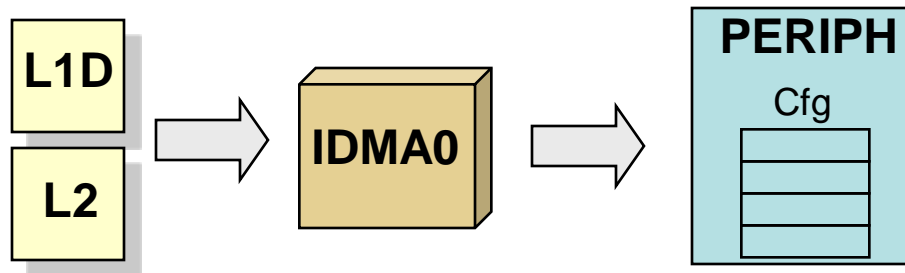
For more information, refer to your device-specific data manual.

# IDMA = Internal DMA

- C64x+ IDMA – Performs background data movement or peripheral programming **WITHOUT** using EDMA bandwidth/resources or TeraNet SCR (internal to CorePac).

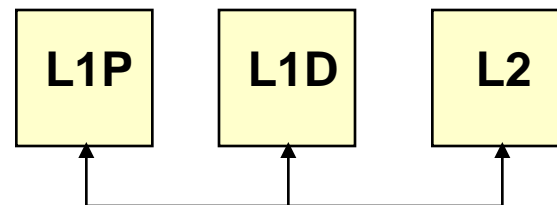
## Channel 0 (IDMA0 – Hi Priority)

- Performs rapid programming of peripheral configuration registers
- Avoids unnecessary wait states through CFG bus vs. traditional use of the CPU copying config structures from L2 to the peripheral registers
- Typically used when new config structures are needed quickly. A copy of the structures can be stored in L1D/L2 and then transferred during run-time.



## Channel 1 (IDMA1 – Lo Priority)

- Rapid block transfers between L1P, L1D, L2



# Outline

- ◆ Introduction to EDMA3
- ◆ Example 1: Single Block Transfer
- ◆ Programming EDMA3 with CSL 3.0
- ◆ Example 2: Multiple Block Transfer
- ◆ Linking vs. Chaining
- ◆ QDMA
- ◆ IDMA



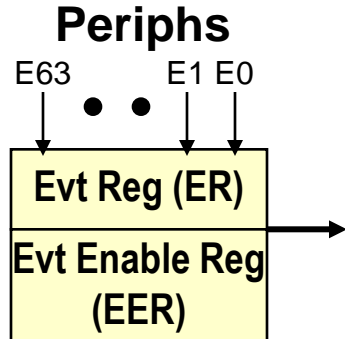
# Single Block Transfer Process

- 1. Trigger the transfer to start**
- 2. EDMA3 executes the transfer**
- 3. Post-transfer actions**
  - ◆ notify the CPU (interrupt)**
  - ◆ start another transfer (chaining)**

# Trigger an EDMA3 Transfer to Start

- ◆ Each of the 64 DMA channels can be triggered by any of the following:

## Event Triggering (from a peripheral) – EER/ER



### Examples

- McBSP 0/1 (REVT0/1, XEVT0/1)
- Timer 0/1 (TEVTLO/HI 0/1)
- GPIO (GPINT[15:5])
- Chip Int Cntlr 3 (CIC3[15:0])
- VCP2 (VCP2REVT/XEVT)
- TCP2 (TCP2REVT/XEVT)
- FSEVT[13:4]
- I2C (ICREVT/XEVT)

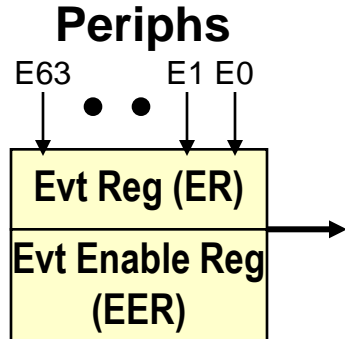
- Each event is tied to a specific DMA channel (e.g. XEVT1 → Ch 14) and can be enabled/disabled via EER register

12	XEVT0	McBSP 0 Transmit Event
13	REVT0	McBSP 0 Receive Event
14	XEVT1	McBSP 1 Transmit Event
15	REVT1	McBSP 1 Receive Event

# Trigger an EDMA3 Transfer to Start

- ◆ Each of the 64 DMA channels can be triggered by any of the following:

## Event Triggering (from a peripheral) – EER/ER



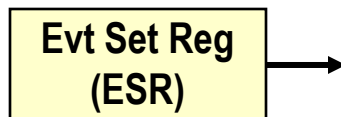
### Examples

- McBSP 0/1 (REVT0/1, XEVT0/1)
- Timer 0/1 (TEVTLO/HI 0/1)
- GPIO (GPINT[15:5])
- Chip Int Cntlr 3 (CIC3[15:0])
- VCP2 (VCP2REVT/XEVT)
- TCP2 (TCP2REVT/XEVT)
- FSEVT[13:4]
- I2C (ICREVT/XEVT)

- Each event is tied to a specific DMA channel (e.g. XEVT1 → Ch 14) and can be enabled/disabled via EER register

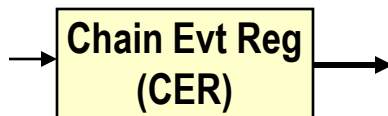
12	XEVT0	McBSP 0 Transmit Event
13	REVT0	McBSP 0 Receive Event
14	XEVT1	McBSP 1 Transmit Event
15	REVT1	McBSP 1 Receive Event

## Manual Triggering - ESR



- CPU writes a “1” to the corresponding bit of the Event Set Register (ESR)

## Chain Triggering - CER



- Used to execute a sequence of TRs after a single event
- Ex: EVT0 triggers Ch0, Ch0 completes and triggers Ch1 (TCC=1)
- Chained events are captured in the Chain Event Register (CER)

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from `&pixel_7` to `&myDest`
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

**8-bit Pixels**

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

(Src: `&pixel_7`)

*Note: data values are in contiguous memory*



`&myDest:`

7
8
9
10
13
14
15
16
19
20
21
22

← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from `&pixel_7` to `&myDest`
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

**8-bit Pixels**

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

(Src: `&pixel_7`)

*Note: data values are in contiguous memory*



`&myDest:`

7
8
9
10
13
14
15
16
19
20
21
22

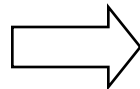
← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0



## Solution


31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from &pixel\_7 to &myDest
- Transfer all pixels as quickly as possible (single EVTx – **xfr all data, AB-sync**)

**8-bit Pixels**

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

(Src: &pixel\_7)

*Note: data values are in contiguous memory*



&myDest:

7
8
9
10
13
14
15
16
19
20
21
22

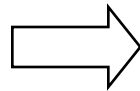
← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0



## Solution


31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from &pixel\_7 to &myDest
- Transfer all pixels as quickly as possible (single EVTx – **xfr all data, AB-sync**)

**8-bit Pixels**

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

(Src: &pixel\_7)

*Note: data values are in contiguous memory*

&myDest:

7
8
9
10
13
14
15
16
19
20
21
22

← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0

## Solution

<b>AB-sync</b>	
3	4
	1

31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from &pixel\_7 to &myDest
- Transfer all pixels as quickly as possible (single EVTx – **xfr all data, AB-sync**)

**8-bit Pixels**

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

(Src: &pixel\_7)

*Note: data values are in contiguous memory*



&myDest:

7
8
9
10
13
14
15
16
19
20
21
22

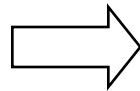
← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0



## Solution

AB-sync	
3	4
	1

31

0



# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from &pixel\_7 to &myDest
- Transfer all pixels as quickly as possible (single EVTx – **xfr all data, AB-sync**)

**8-bit Pixels**

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

(Src: &pixel\_7)

*Note: data values are in contiguous memory*



&myDest:

7
8
9
10
13
14
15
16
19
20
21
22

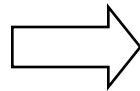
← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0



## Solution

<b>AB-sync</b>	
3	4
	<b>1</b>

31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from &pixel\_7 to &myDest
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

**8-bit Pixels**

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

(Src: &pixel\_7)

*Note: data values are in contiguous memory*

&myDest:

7
8
9
10
13
14
15
16
19
20
21
22

← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0

## Solution

**A-sync?**

	4

31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from &pixel\_7 to &myDest
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

**8-bit Pixels**

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

(Src: &pixel\_7)

*Note: data values are in contiguous memory*

&myDest:

7
8
9
10
13 - 11
14 - 12
15 - 13
16 - 14
19 - 15
20 - 16
21 - 17
22 - 18

← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0

## Solution

A-sync?	
12	

31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from `&pixel_7` to `&myDest`
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

8-bit Pixels

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

(Src: `&pixel_7`)

*Note: data values are in contiguous memory*



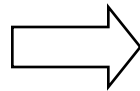
`&myDest:`

7
8
9
10
13
14
15
16
19
20
21
22

← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT



## Solution

AB-sync	
<code>&amp;pixel_7</code>	
3	4
<code>&amp;myDest</code>	
	1

31

0

31

0

# Parameters for a Single Block Transfer

## Goals:

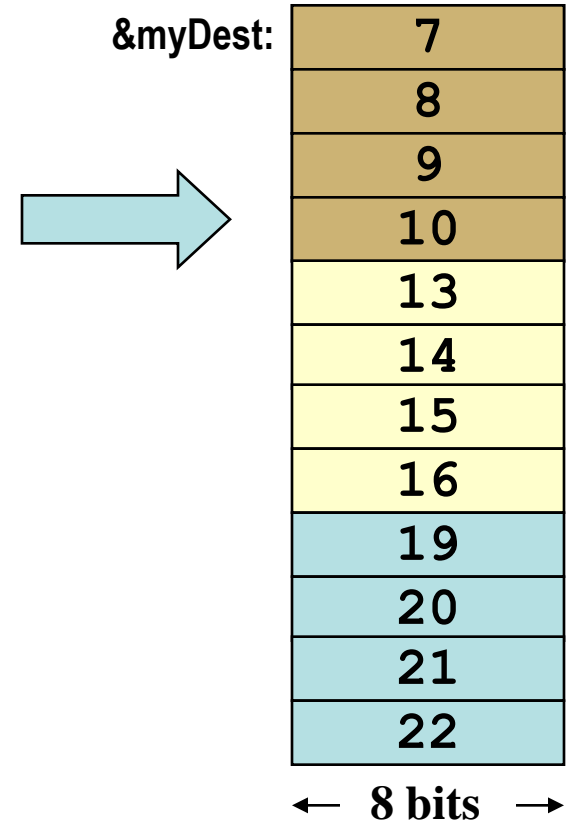
- Transfer a block of 8-bit pixels from `&pixel_7` to `&myDest`
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

8-bit Pixels

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

(Src: `&pixel_7`)

*Note: data values are in contiguous memory*



## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	<b>SRCBIDX</b>
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0

## Solution

AB-sync	
<code>&amp;pixel_7</code>	
3	4
<code>&amp;myDest</code>	
	<b>6</b>
	1

31

0

# Parameters for a Single Block Transfer

## Goals:

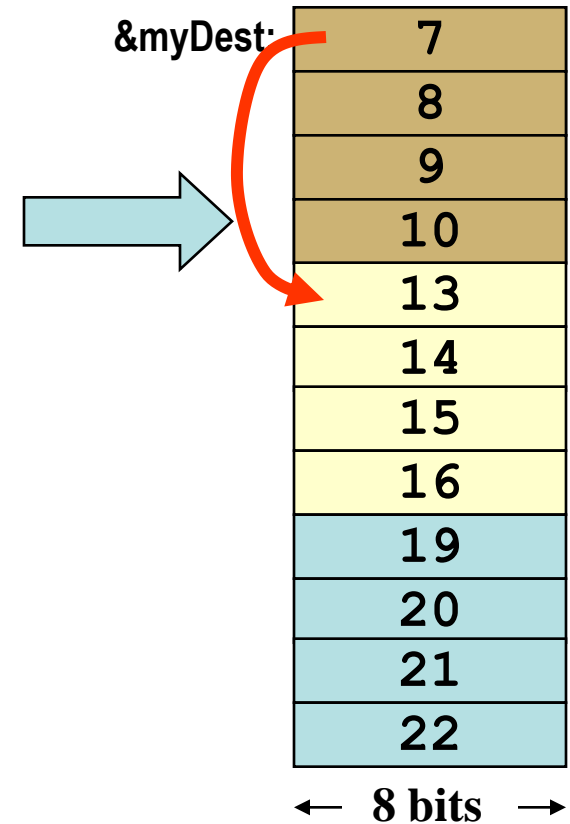
- Transfer a block of 8-bit pixels from `&pixel_7` to `&myDest`
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

**8-bit Pixels**

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

(Src: `&pixel_7`)

*Note: data values are in contiguous memory*

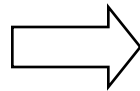


## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
<b>DSTBIDX</b>	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0



## Solution

AB-sync	
<code>&amp;pixel_7</code>	
3	4
<code>&amp;myDest</code>	
<b>4</b>	6
	1

31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from `&pixel_7` to `&myDest`
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

**8-bit Pixels**

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

(Src: `&pixel_7`)

*Note: data values are in contiguous memory*



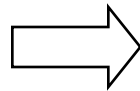
`&myDest:`

7
8
9
10
13
14
15
16
19
20
21
22

← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT



## Solution

AB-sync	
&pixel_7	
3	4
&myDest	
4	6
0	0
	1

31

0

31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from `&pixel_7` to `&myDest`
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

**8-bit Pixels**

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

(Src: `&pixel_7`)

*Note: data values are in contiguous memory*



`&myDest:`

7
8
9
10
13
14
15
16
19
20
21
22

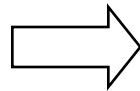
← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0



## Solution

AB-sync	
&pixel_7	
3	4
&myDest	
4	6
BCNT or any	
0	0
	1

31

0



# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from `&pixel_7` to `&myDest`
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

**8-bit Pixels**

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

(Src: `&pixel_7`)

*Note: data values are in contiguous memory*



`&myDest:`

7
8
9
10
13
14
15
16
19
20
21
22

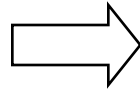
← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	<b>LINK</b>
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0



## Solution

AB-sync	
&pixel_7	
3	4
&myDest	
4	6
3	<b>0xffff</b>
0	0
	1

31

0

# Parameters for a Single Block Transfer

## Goals:

- Transfer a block of 8-bit pixels from `&pixel_7` to `&myDest`
- Transfer all pixels as quickly as possible (single EVTx – xfr all data, AB-sync)

**8-bit Pixels**

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

(Src: `&pixel_7`)

*Note: data values are in contiguous memory*



`&myDest:`

7
8
9
10
13
14
15
16
19
20
21
22

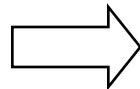
← 8 bits →

## Param Set (active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0



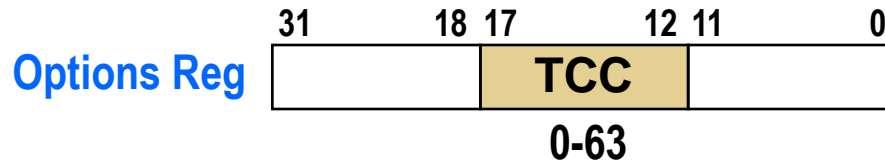
## Solution

AB-sync	
&pixel_7	
3	4
&myDest	
4	6
3	0xffff
0	0
	1

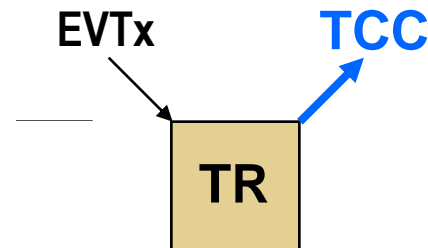
31

0

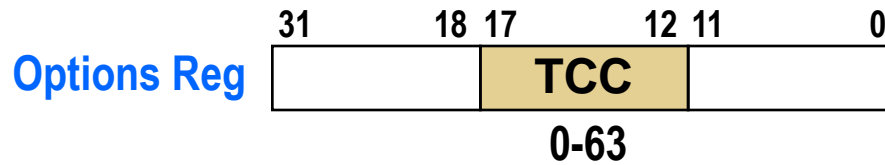
# Transfer Complete Code (TCC)



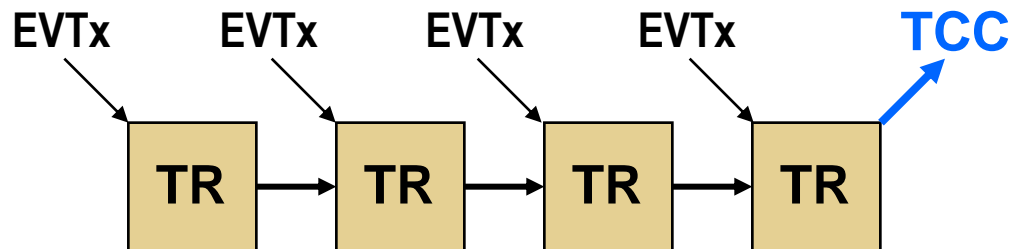
- ◆ TCC is generated when a transfer completes.
- ◆ TCC can be used to trigger a CPU interrupt and/or another transfer (chaining)
- ◆ Each TR below represents one “Transfer Request” which is either ACNT bytes (A-sync) or ACNT \* BCNT bytes (AB-sync).



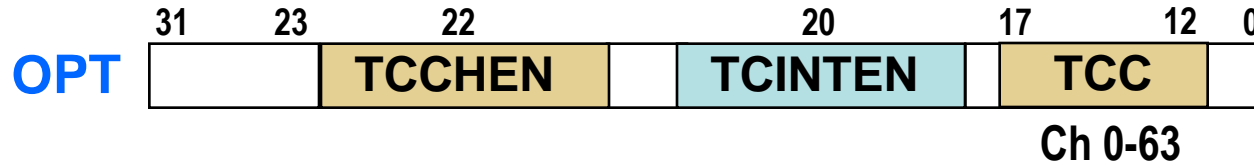
# Transfer Complete Code (TCC)



- ◆ TCC is generated when a transfer completes.
- ◆ TCC can be used to trigger a CPU interrupt and/or another transfer (chaining)
- ◆ Each TR below represents one “Transfer Request” which is either ACNT bytes (A-sync) or ACNT \* BCNT bytes (AB-sync).

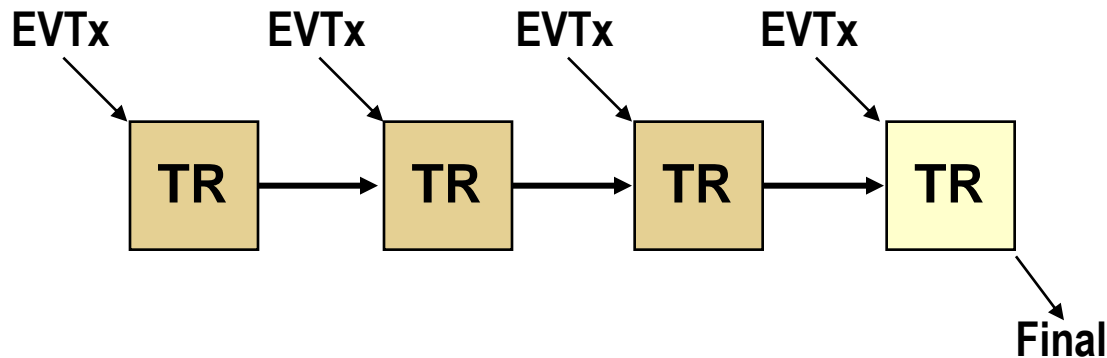


# Transfer Completion



*Transfer Completion* indicates a COMPLETE transfer sequence has been completed.

- ◆ Chain Event Register (CER[TCC]) gets set if selected by TCCHEN (chaining)
- ◆ Interrupt Pending Register (IPR[TCC]) set if selected by TCINTEN (this can interrupt the CPU)



**EVTx =**

- ER (sync)
- ESR (manual)
- CER (chain)

- ◆ Each TR (Transfer Request) can be ACNT bytes (A-sync) or ACNT\*BCNT bytes (AB-sync)
- ◆ This “Final” TCC is for only the LAST TR of a transfer.

# Outline

- ◆ Introduction to EDMA3
- ◆ Example 1: Single Block Transfer
- ◆ Programming EDMA3 with CSL 3.0
- ◆ Example 2: Multiple Block Transfer
- ◆ Linking vs. Chaining
- ◆ QDMA
- ◆ IDMA

# EDMA3 Programming Model

## 1. Initialize EDMA3 Module

## 2. Configure Channel

A. Channel #, Handle

B. Options Register

C. Other Channel Parameters (ACNT, BCNT, etc)

D. Write Config Values to PARAM

## 3. Start the Channel Running (manual, sync, ...)

# Example 1: Single Block Transfer

➤ From the proceeding slides, our goal is to program this example transfer

➤ We need to program:

- Options Register (TCC, Sync: A or AB)
- ACNT, BCNT, CCNT
- 'BIDX, 'CIDX
- Src/Dst Addr

**8-bit Pixels**

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

(Src: &pixel\_7)

*Note: data values are in contiguous memory*



&myDest:

7
8
9
10
13
14
15
16
19
20
21
22

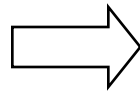
← 8 bits →

## Parameter Set (n)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31

0



## Solution

TCC & AB-sync	
&pixel_7	
3	4
&myDest	
4	6
= BCNT	0xFFFF
0	0
RSVD	1

31

0

Let's look at the CSL code required to program this example transfer...



# Step 1: Initialize EDMA3 Module

## Declarations

```
#include <csl.h>
#include <csl_edma3.h>
CSL_Edma3Context    context;
CSL_Status           status;
CSL_Edma3Handle     hEdmaModule;
CSL_Edma3Obj        edmaObj;
```

## Init EDMA3 Module

```
// Init is a CSL placeholder function for consistency (must be executed first)
status = CSL_edma3Init(&context);
```

## Get Handle to EDMA3 Module

```
// Open populates the Object and returns the Module handle
hEdmaModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);
```

# Step 2A: Open Channel

## Declarations

```
CSL_Edma3ChannelObj      chObj;  
CSL_Edma3ChannelAttr     chAttr;  
CSL_Edma3ChannelHandle   hChannel;
```

## Ch Selection

```
chAttr.regionNum = CSL_EDMA3_REGION_GLOBAL;
```

```
chAttr.chaNum = CSL_EDMA3_CHA_4;           // Channel w/ no event tied to it
```

## Open Ch

```
hChannel = CSL_edma3ChannelOpen(&chObj, CSL_EDMA3, &chAttr, &status);
```

- ◆ **CSL\_edma3ChannelOpen() is similar to <mod>Open. In this case, it populates the CHANNEL object and returns a handle to the opened CHANNEL.**
- ◆ **In the following code, we can use this handle (hChannel) to write to the channel's register set.**

Let's first review the **OPTIONS** register...

# Channel OPTions Register

- ◆ The Options register contains bit fields that configure how the channel operates
- ◆ Each field has a corresponding description in the Param Setup code comments

Figure 2-8. Channel Options Parameter (OPT)

31	30	28	27	24	23	22	21	20	19	18	17	16
PRIV	Reserved	PRIVID		ITCCHEN	TCCHEN	ITCINTEN	TCINTEN	Reserved		TCC		
R-0	R-0	R-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R/W-0		
15	12	11	10	8	7	4			3	2	1	0
TCC		TCCMOD	FWID	Reserved				STATIC	SYNCDIM		DAM	SAM
R/W-0		R/W-0	R/W-0	R/W-0				R/W-0	R/W-0		R/W-0	R/W-0

**TCC = Transfer Complete Code to signal completion**

**SYNCDIM = A-sync or AB-sync**

# Step 2B: Configure Options

```
CSL_Edma3ParamSetup    myParamSetup = {  
  
    CSL_EDMA3_OPT_MAKE (  
        CSL_EDMA3_ITCCH_DIS,  
        CSL_EDMA3_TCCH_DIS,  
        CSL_EDMA3_ITCINT_DIS,  
        CSL_EDMA3_TCINT_DIS,  
        CSL_EDMA3_CHA_4,           // TCC (ex., match ch)  
        CSL_EDMA3_TCC_NORMAL,  
        CSL_EDMA3_FIFOWIDTH_NONE,  
        CSL_EDMA3_STATIC_DIS,  
        CSL_EDMA3_SYNC_AB,       // Sync mode (A or AB)  
        CSL_EDMA3_ADDRMODE_INCR,  
        CSL_EDMA3_ADDRMODE_INCR ),  
  
        ■ ■ ■  
    }
```

# Step 2C: Configure Channel Params

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT
31	0

Options	
&pixel_7	
3	4
&myDest	
4	6
= BCNT	0xFFFF (later)
0	0
RSVD	1
31	0

```

&pixel_7,
CSL_EDMA3_CNT_MAKE(4, 3),
&myDest,
CSL_EDMA3_BIDX_MAKE(6, 4),
CSL_EDMA3_LINKBCNTRLD_MAKE(0xFFFF, 3),
CSL_EDMA3_CIDX_MAKE(0, 0),
1
};

```

```

// Source Addr
// aCntbCnt - (ACNT, BCNT)
// Dest Addr
// srcDstBidx - (SRCBIDX, DSTBIDX)
// linkBcntrl - (LINK, BCNTRLD)
// srcDstCidx - (SRCCIDX, DSTCIDX)
// cCnt - CCNT

```

# Step 2D: Write Channel Params to PSET

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

31 0

Options	
&pixel_7	
3	4
&myDest	
4	6
= BCNT	0xFFFF (later)
0	0
RSVD	1

31 0

// write the PaRAM setup values to PaRAM – *this gets the handle to the PSET (e.g. #249)*

```
PsetNum = 249;
```

```
hParam = CSL_edma3GetParamHandle(hChannel, PsetNum, NULL);
```

```
status = CSL_edma3ParamSetup(hParam, &myParamSetup);
```

// map the channel (#4) to the PSET (#249)

```
CSL_edma3HwChannelSetupParam(hChannel, PsetNum)
```

// map the channel (#4) to a queue

```
CSL_edma3HwChannelSetupQue(hChannel, CSL_EDMA3_QUE_1)
```

# Step 3: Enable and Start Channel

## ◆ Start the Channel Running (3 options)

- Event Sync from peripheral (Event Enable Register – set bit in EER, next example)

```
CSL_edma3HwChannelControl(hChannel, CSL_EDMA3_CMD_CHANNEL_ENABLE, NULL);
```

- Chain Event from another channel (Chain Event Register – CER)
- Manually Trigger the channel to Run (Event Set Register – ESR) (shown below)

```
CSL_edma3HwChannelControl(hChannel, CSL_EDMA3_CMD_CHANNEL_SET, NULL);
```

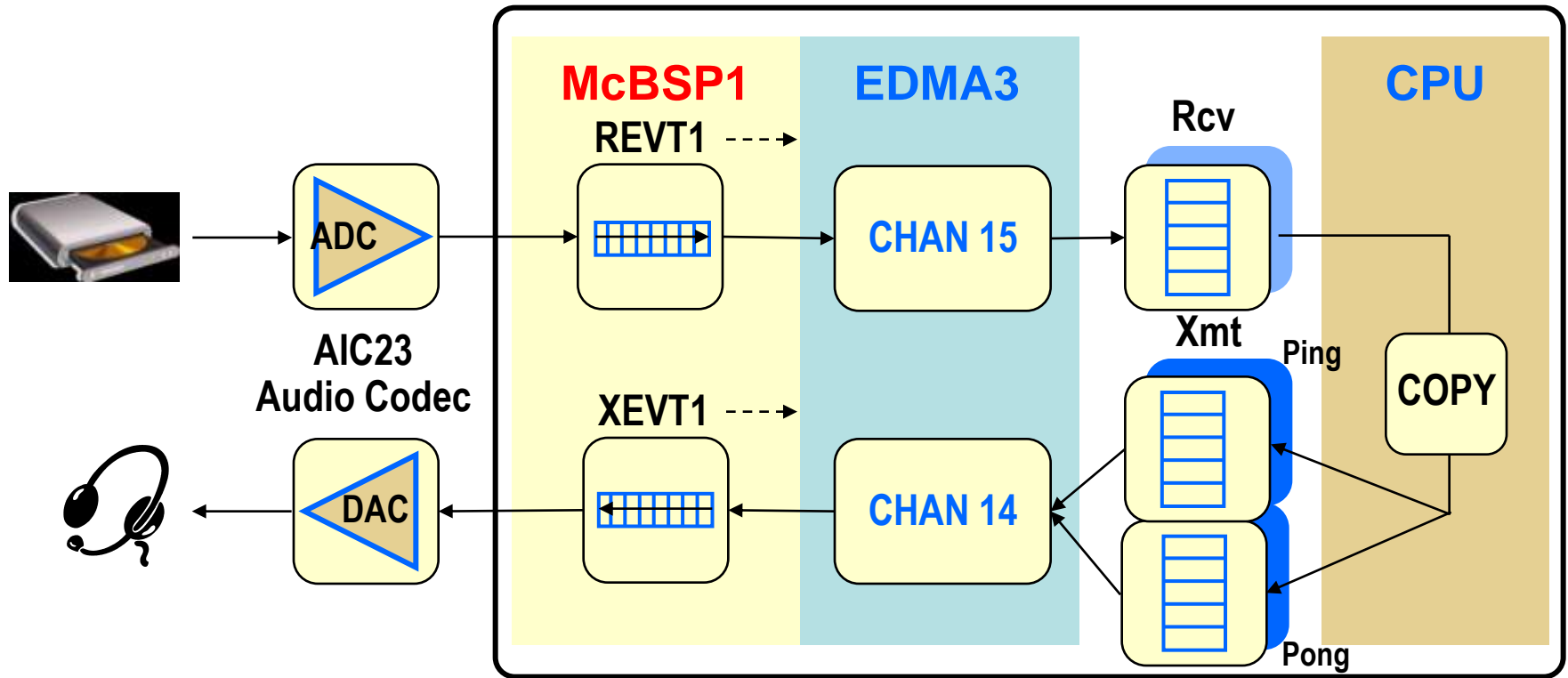
- ## ◆ Notice both call **CSL\_edma3HwChannelControl()**. This is used to enable the channel or to start it manually, i.e. it controls the Ch's operation.

# Outline

- ◆ Introduction to EDMA3
- ◆ Example 1: Single Block Transfer
- ◆ Programming EDMA3 with CSL 3.0
- ◆ Example 2: Multiple Block Transfer
- ◆ Linking vs. Chaining
- ◆ QDMA
- ◆ IDMA

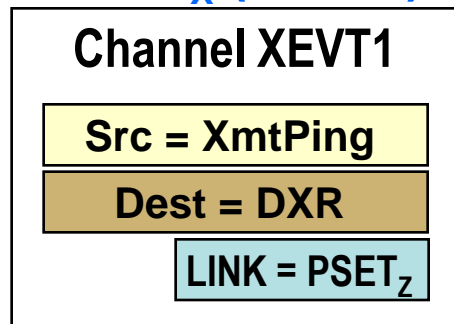


# Linking Ping → Pong → Ping → Etc.

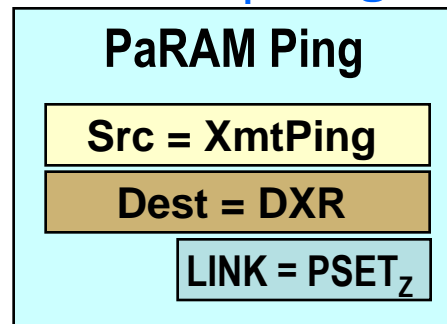


How do we link transfers for ping and pong?

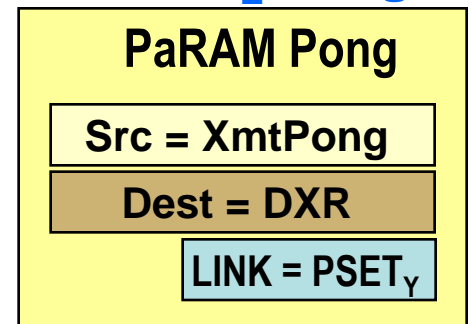
## PSET<sub>x</sub> (Active)



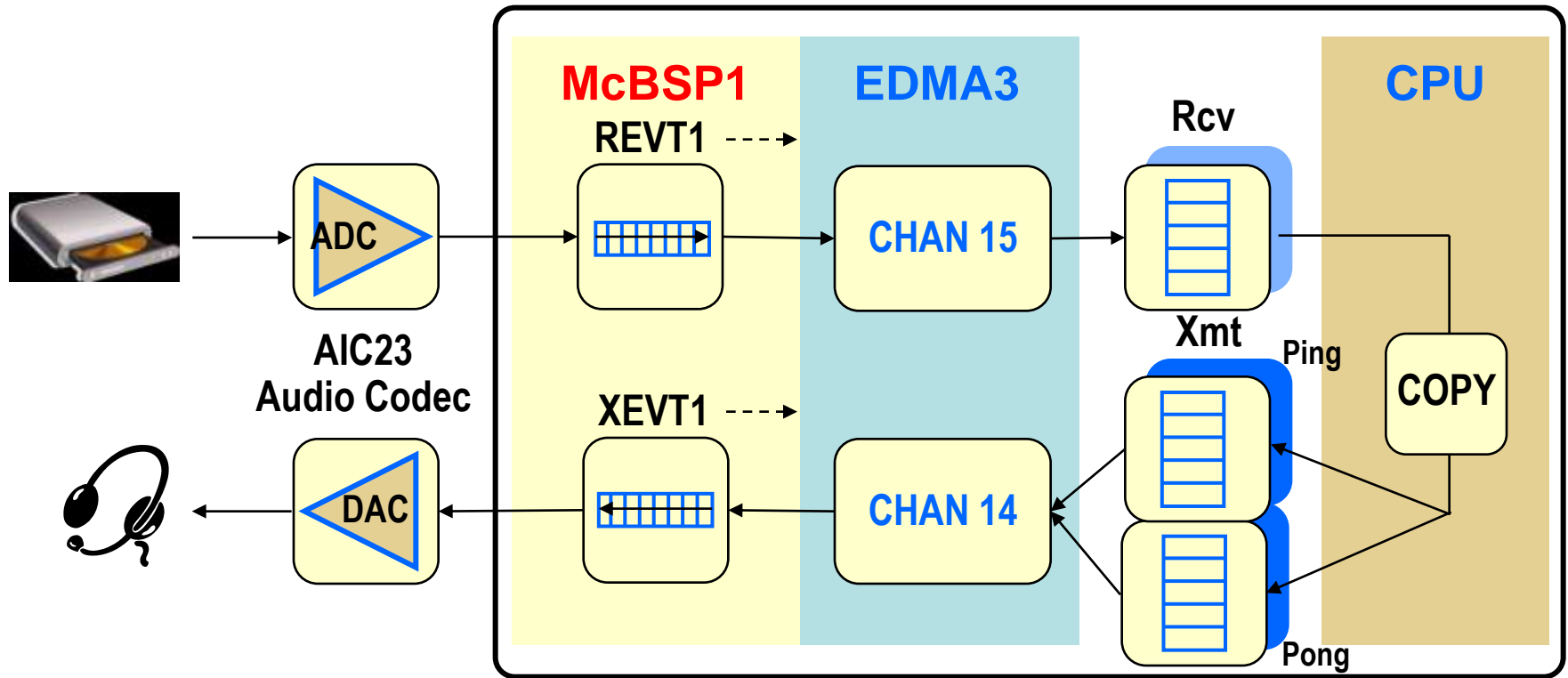
## PSET<sub>y</sub> Ping



## PSET<sub>z</sub> Pong

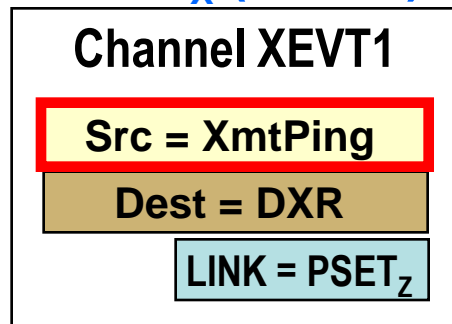


# Linking Ping → Pong → Ping → Etc.

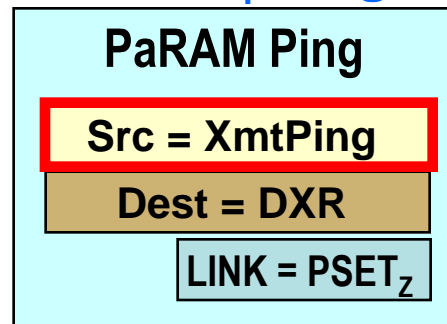


How do we link transfers for ping and pong? Use the Active PSET plus two Link PSETs. Assign different **Src addresses** to use the desired buffer.

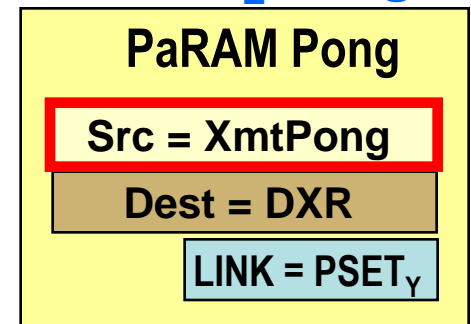
## PSET<sub>x</sub> (Active)



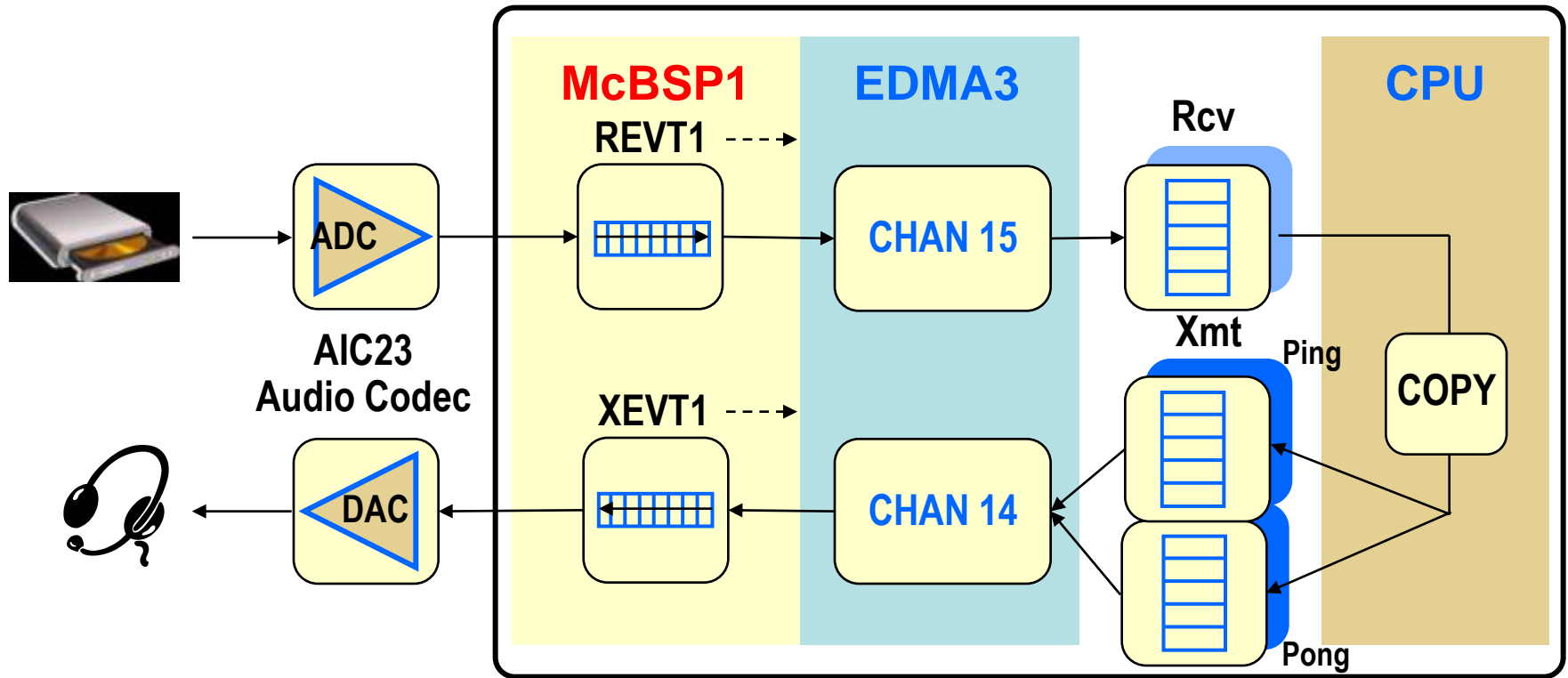
## PSET<sub>y</sub> Ping



## PSET<sub>z</sub> Pong

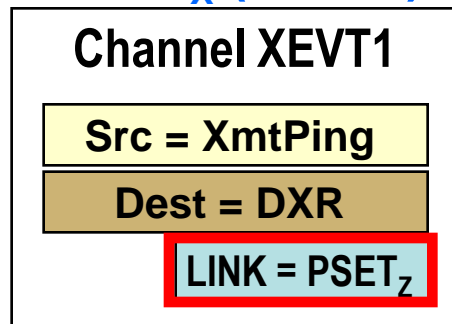


# Linking Ping → Pong → Ping → Etc.

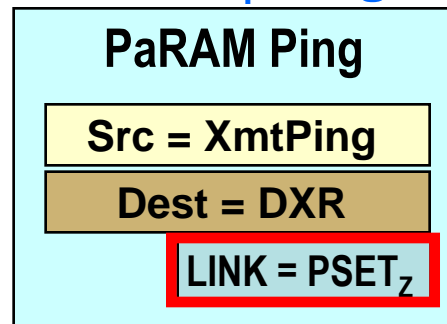


How do we link transfers for ping and pong? Use the Active PSET plus two Link PSETs. Assign different Src addresses to use the desired buffer. Set **LINK** field to point to the **NEXT PSET to use**.

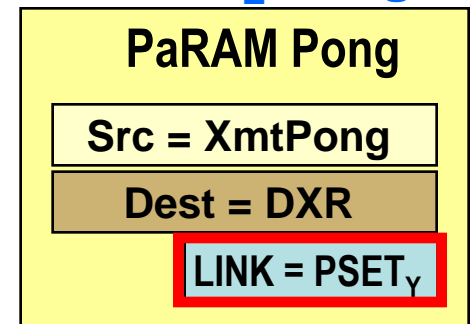
## PSET<sub>x</sub> (Active)



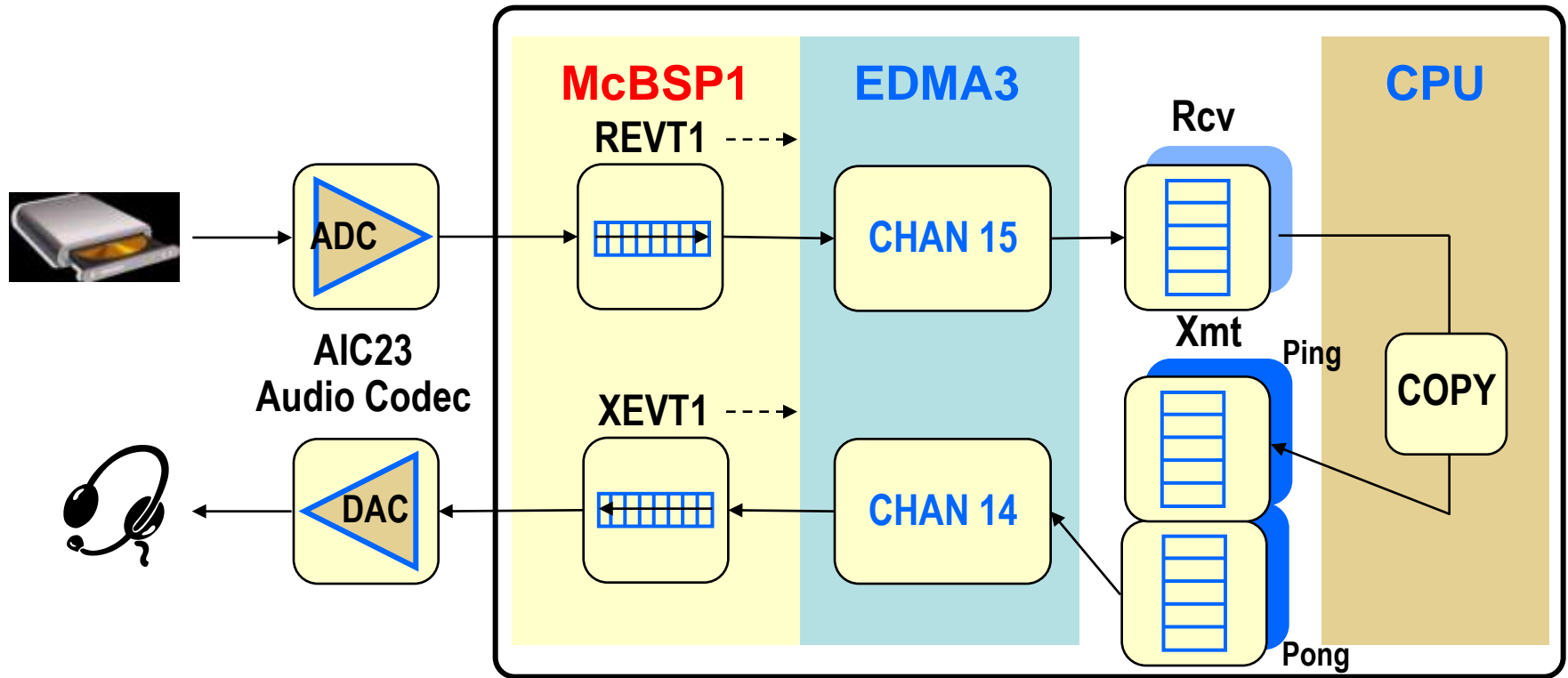
## PSET<sub>y</sub> Ping



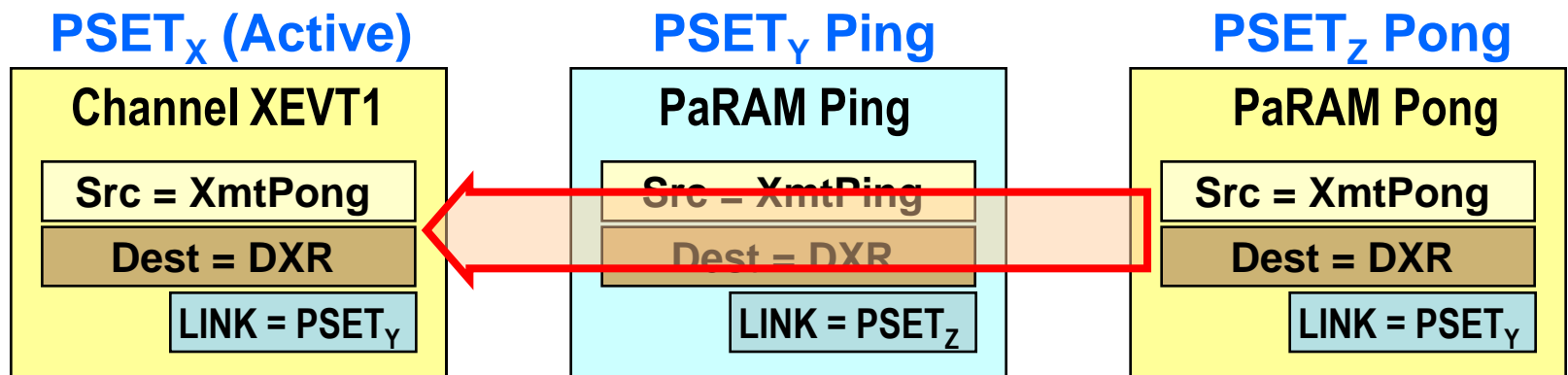
## PSET<sub>z</sub> Pong



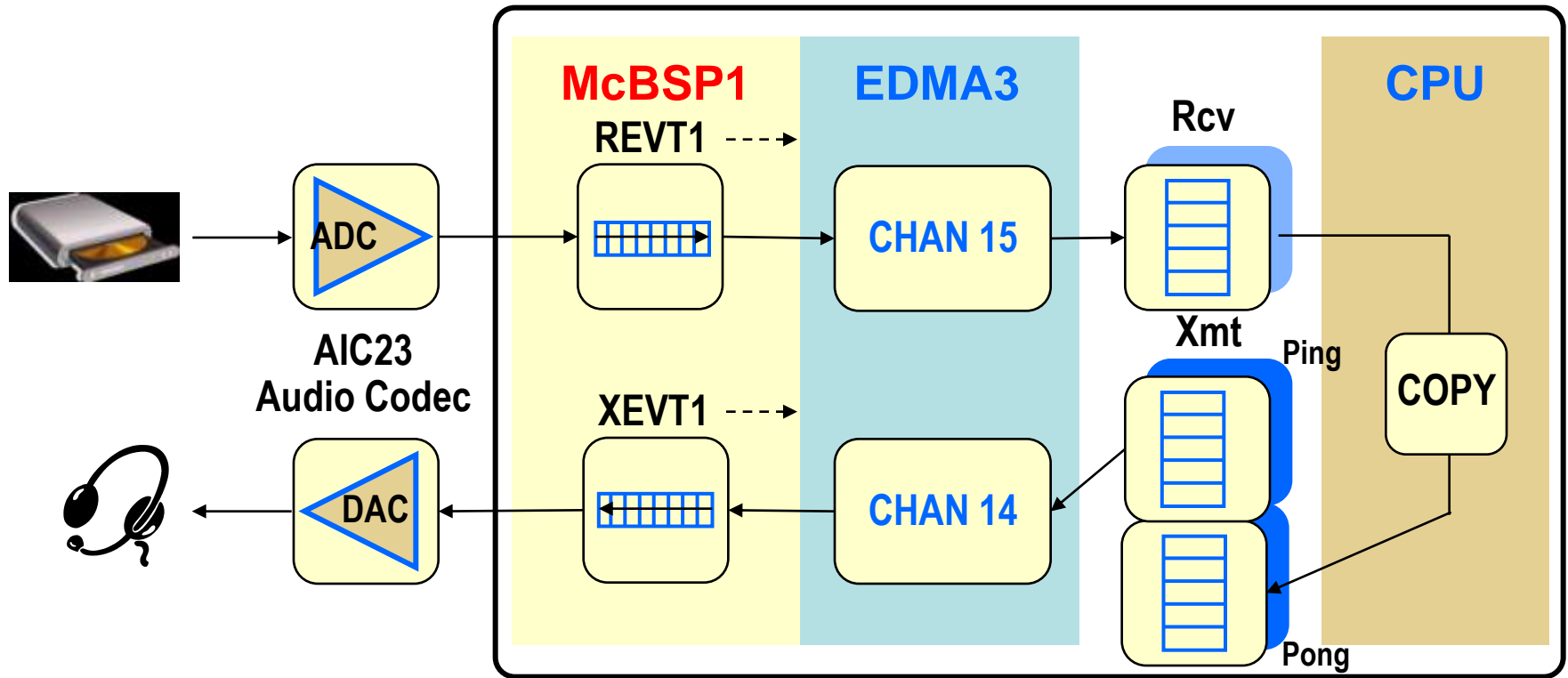
# Linking Ping → Pong → Ping → Etc.



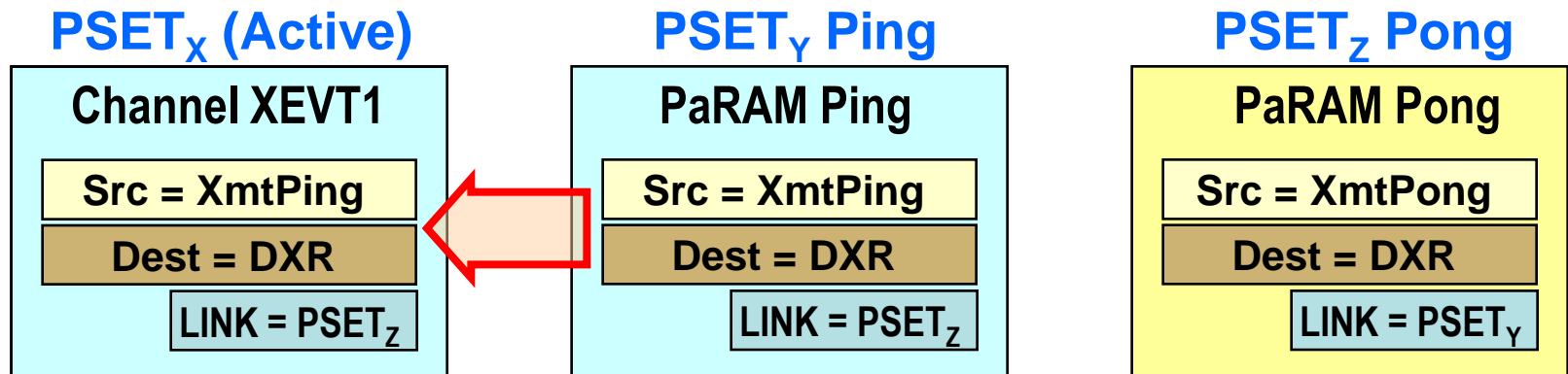
How do we link transfers for ping and pong? When Active Ch  $PSET_x$  is complete,  $PSET_z$  Pong is COPIED to Active Ch  $PSET_x$ .



# Linking Ping → Pong → Ping → Etc.



**How do we link transfers for ping and pong?** When Active Ch  $PSET_x$  is complete,  $PSET_z$  Pong is COPIED to Active Ch  $PSET_x$ . When Pong is done,  $PSET_y$  Ping is COPIED to Active Ch  $PSET_x$ .



# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



**McBSP1**

Dst: &DXR

Pong Src: &audio\_37

## PSET<sub>x</sub> (Active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync? (A or AB)

### 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



**McBSP1**

Dst: &DXR

Pong Src: &audio\_37

### PSET<sub>x</sub> (Active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

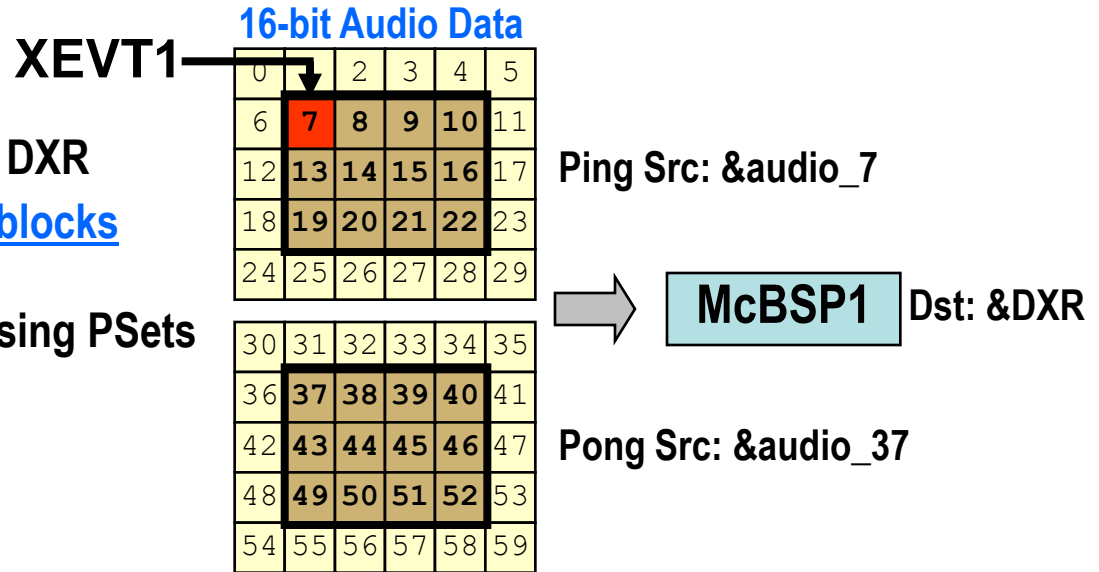
# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync? (A or AB)



## PSET<sub>x</sub> (Active)

Options	
Source	
BCNT	ACNT
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT



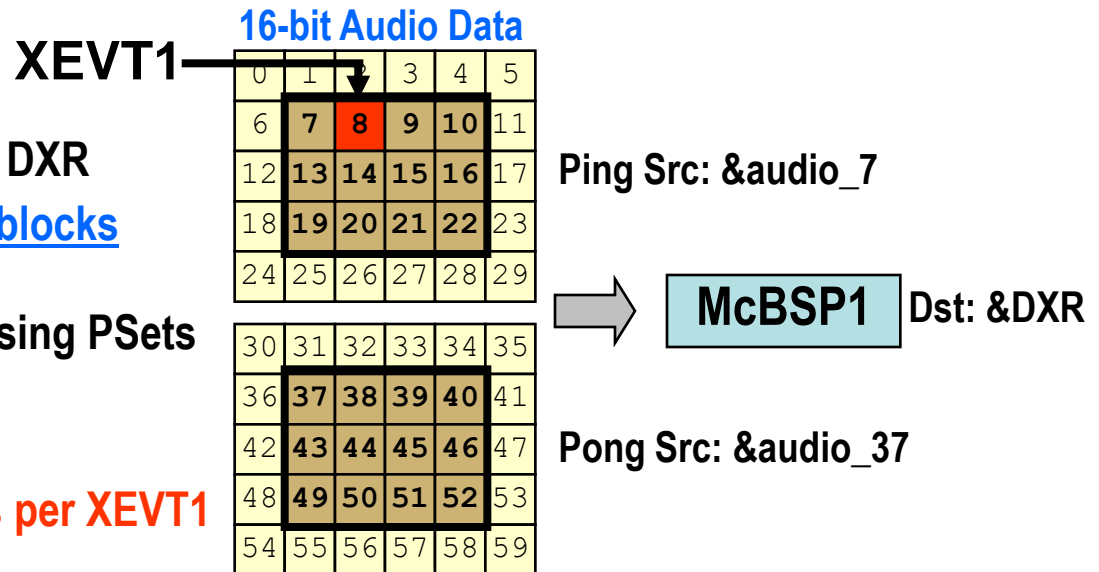
# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync? **A-sync for 2 bytes per XEVT1**



## PSET<sub>x</sub> (Active)

### Options – A-sync

#### Source

BCNT

ACNT

#### Destination

DSTBIDX

SRCBIDX

BCNTRLD

LINK

DSTCIDX

SRCCIDX

RSVD

CCNT

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- **Sizes of ACNT, BCNT, CCNT and indexes?**

### 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



McBSP1

Dst: &DXR

Pong Src: &audio\_37

### PSET<sub>x</sub> (Active)

Options – A-sync

Source

BCNT

ACNT

Destination

DSTBIDX

SRCBIDX

BCNTRLD

LINK

DSTCIDX

SRCCIDX

RSVD

CCNT

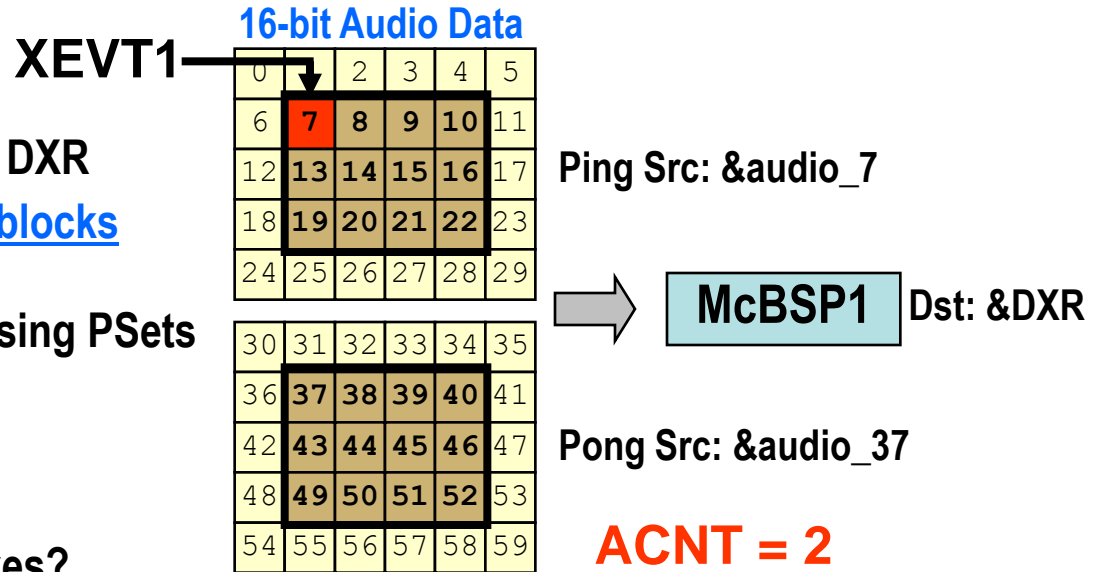
# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT and indexes?



## PSET<sub>x</sub> (Active)

Options – A-sync	
Source	
BCNT	2
Destination	
DSTBIDX	SRCBIDX
BCNTRLD	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT and indexes?

### 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7

→ **McBSP1** Dst: &DXR

Pong Src: &audio\_37

**ACNT = 2**

**BCNT = 4**

### PSET<sub>x</sub> (Active)

Options – A-sync	
Source	
4	2
Destination	
DSTBIDX	SRCBIDX
4	LINK
DSTCIDX	SRCCIDX
RSVD	CCNT

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT and indexes?

## PSET<sub>x</sub> (Active)

Options – A-sync	
Source	
4	2
Destination	
DSTBIDX	SRCBIDX
4	LINK
DSTCIDX	SRCCIDX
RSVD	3

## 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7

→ **McBSP1** Dst: &DXR

Pong Src: &audio\_37

**ACNT = 2**

**BCNT = 4**

**CCNT = 3**

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT **and indexes?**

### 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



McBSP1

Dst: &DXR

Pong Src: &audio\_37

### PSET<sub>x</sub> (Active)

Options – A-sync	
Source	
4	2
Destination	
DSTBIDX	SRCBIDX
4	LINK
DSTCIDX	SRCCIDX
RSVD	3

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT and indexes?

### 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7

→ **McBSP1** Dst: &DXR

Pong Src: &audio\_37

**SRCBIDX = 2 = ACNT**

**DSTBIDX = 0 (DXR)**

### PSET<sub>x</sub> (Active)

Options – A-sync	
Source	
4	2
Destination	
0	2
4	LINK
DSTCIDX	SRCCIDX
RSVD	3

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT and indexes?

### PSET<sub>x</sub> (Active)

Options – A-sync	
Source	
4	2
Destination	
0	2
4	LINK
0	6
RSVD	3

### 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7

→ **McBSP1** Dst: &DXR

Pong Src: &audio\_37

**SRCBIDX = 2 = ACNT**

**DSTBIDX = 0 (DXR)**

**SRCCIDX = 6**

**DSTCIDX = 0 (DXR)**



# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT and indexes?
- Which channel should we use and why?

## PSET<sub>x</sub> (Active)

Options – A-sync	
Source	
4	2
Destination	
0	2
4	LINK
0	6
RSVD	3

## 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



McBSP1

Dst: &DXR

Pong Src: &audio\_37

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT and indexes?
- Which channel should we use and why?

**16-bit Audio Data**  
**XEVT1** →

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



**McBSP1** Dst: &DXR

Pong Src: &audio\_37

**XEVT1 event**

**XEVT1 = 14**

## PSET<sub>x</sub> (Active)

Options – A-sync	
Source	
4	2
Destination	
0	2
4	LINK
0	6
RSVD	3

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT and indexes?
- Which channel should we use and why?
- **Src/Dst addresses?**

**PSET<sub>x</sub> (Active)**

Options – A-sync	
Source	
4	2
Destination	
0	2
4	LINK
0	6
RSVD	3

16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



**McBSP1** Dst: &DXR

Pong Src: &audio\_37

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

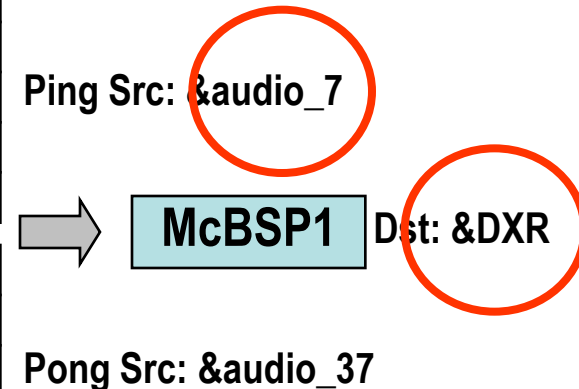
- What kind of Sync?
- Sizes of ACNT, BCNT, CCNT and indexes?
- Which channel should we use and why?
- Src/Dst addresses?

### PSET<sub>x</sub> (Active)

Options – A-sync	
&audio_7	
4	2
&DXR	
0	2
4	LINK
0	6
RSVD	3

### 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59



# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- How do we transfer the second block?

### 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



McBSP1

Dst: &DXR

Pong Src: &audio\_37

### PSET<sub>x</sub> (Active)

#### Options – A-sync

&audio\_7

4

2

&DXR

0

2

4

LINK

0

6

RSVD

3

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- How do we transfer the second block?

### 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



McBSP1 Dst: &DXR

Pong Src: &audio\_37

### PSET<sub>x</sub> (Active)

Options – A-sync	
&audio_7	
4	2
&DXR	
0	2
4	PSET <sub>y</sub>
0	6
RSVD	3



### PSET<sub>y</sub> (Pong)

Options – A-sync	
&audio_37	
4	2
&DXR	
0	2
4	NULL
0	6
	3

# Example 2: Multiple Block Transfer

## Goals:

- Transfer two blocks of 16-bit audio data from &audio\_7 & \_37 to McBSP1 DXR
- Trigger an interrupt to CPU after both blocks have been transferred.
- Link between ping (\_7) & pong (\_37) using PSets

## Questions:

- How do we transfer the second block?
- **How do we generate an interrupt?**

## 16-bit Audio Data

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Ping Src: &audio\_7



McBSP1 Dst: &DXR

Pong Src: &audio\_37

## PSET<sub>x</sub> (Active)

Options – A-sync	
&audio_7	
4	2
&DXR	
0	2
4	PSET <sub>y</sub>
0	6
RSVD	3



## PSET<sub>y</sub> (Pong)

Options – A-sync	
&audio_37	
4	2
&DXR	
0	2
4	NULL
0	6
	3

# Interrupt: EDMA Channels

## EDMA Channels

Channel # Options TCC

0	TCINTEN=0	TCC=0	○
1	TCINTEN=0	TCC=1	○
⋮	TCINTEN=1	TCC=14	○
63	TCINTEN=0	TCC=63	○

Options

TCINTEN

20

TCC

17

12



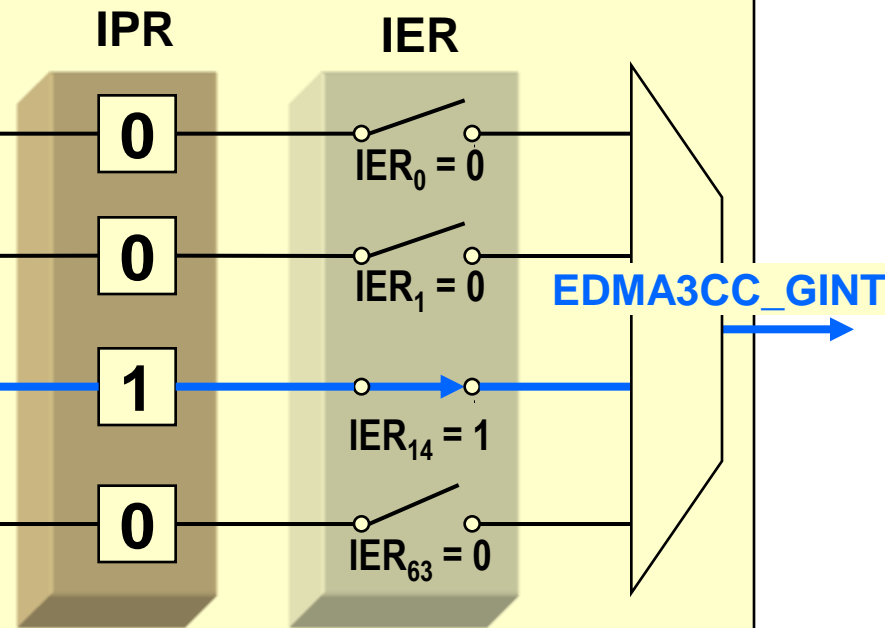
# Generate an EDMA Interrupt

## EDMA Channels

Channel #	Options	TCC
0	TCINTEN=0	TCC=0
1	TCINTEN=0	TCC=1
⋮		
63	TCINTEN=0	TCC=63



## EDMA Interrupt Generation



IER – EDMA Interrupt Enable Register (NOT the CPU IER)  
IPR – EDMA Interrupt Pending Register (set by TCC)

# Generate an EDMA Interrupt

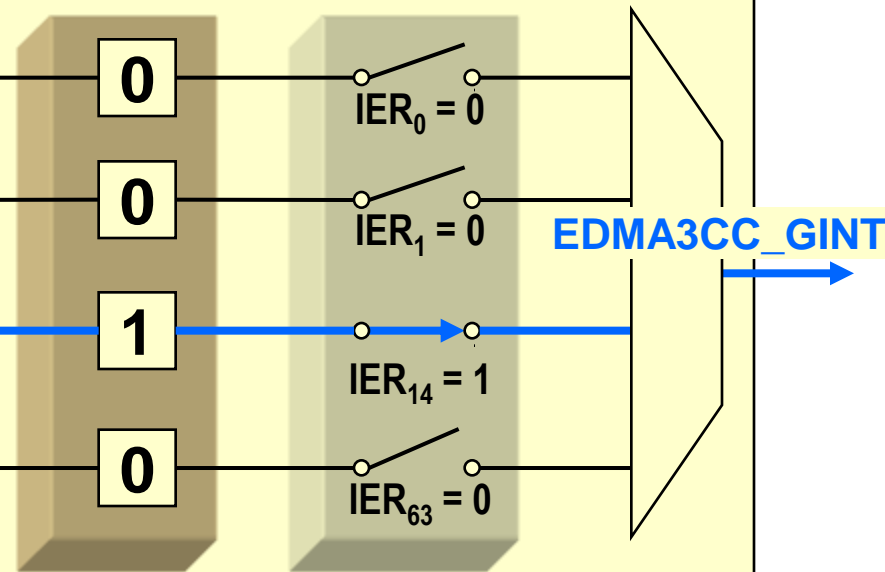
## EDMA Channels

Channel #	Options	TCC
0	TCINTEN=0	TCC=0
1	TCINTEN=0	TCC=1
⋮		
63	TCINTEN=0	TCC=63



## EDMA Interrupt Generation

IPR IER



IER – EDMA Interrupt Enable Register (NOT the CPU IER)  
IPR – EDMA Interrupt Pending Register (set by TCC)

**To set the proper EDMA IER bit for XEVT1:**

```
edmaIntr.region = CSL_EDMA3_REGION_GLOBAL ;
edmaIntr.intrh = 1 << (CSL_EDMA3_CHA_XEVT1-32); // high 32 bits
edmaIntr.intr  = 1 << (CSL_EDMA3_CHA_XEVT1);    // low 32 bits
CSL_edma3HwControl(hModule, CSL_EDMA3_CMD_INTR_ENABLE, &edmaIntr);
```

# Check the IPR<sub>bit</sub>

- ◆ If there are 64 channels, 64 IPR bits and only ONE EDMA interrupt (EDMA3CC\_GINT), how do you know which IPR got set?

# Check the IPR<sub>bit</sub>

- ◆ If there are 64 channels, 64 IPR bits and only ONE EDMA interrupt (EDMA3CC\_GINT), how do you know which IPR got set?
- ◆ You check the appropriate IPR bit. In this example, to check the proper EDMA IPR bit for XEVT1, you could use:

```
void edmaHwi(void) {  
    Uint32 intr;  
    intr = *pEdmaChannelPendReg;    // Set intr = EDMA IPR  
  
    if (intr & (0x01 << CSL_EDMA3_CHA_XEVT1)) {    // Check IPR to see if XEVT1 is set  
        SEM_post(&xmtBuffReady);  
    }  
  
    *pEdmaChannelClearReg = intr;    // Clear EDMA IPR – user must clear this bit  
}
```

# Check the IPR<sub>bit</sub>

- ◆ If there are 64 channels, 64 IPR bits and only ONE EDMA interrupt (EDMA3CC\_GINT), how do you know which IPR got set?
- ◆ You check the appropriate IPR bit. In this example, to check the proper EDMA IPR bit for XEVT1, you could use:

```
void edmaHwi(void) {  
    Uint32 intr;  
    intr = *pEdmaChannelPendReg;    // Set intr = EDMA IPR  
  
    if (intr & (0x01 << CSL_EDMA3_CHA_XEVT1)) { // Check IPR to see if XEVT1 is set  
        SEM_post(&xmtBuffReady);  
    }  
  
    *pEdmaChannelClearReg = intr;    // Clear EDMA IPR – user must clear this bit  
}
```

- ◆ Or you can use the **EDMA Interrupt Dispatcher...**

# EDMA Interrupt Dispatcher

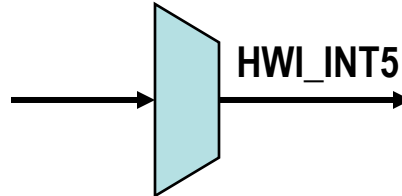
➤ Here's the interrupt chain from beginning to end:

## 1. An interrupt occurs



EDMA3CC\_GINT (#24)

## 2. Interrupt Selector



HWI\_INT5

## 3. HWI\_INT5 Properties

HWI\_INT5 Properties

General Dispatcher

comment: defines the INT5 Interrupt

interrupt selection number: 24

function: \_edma\_int\_dispatcher

## 4. HWI Dispatcher (ON + Arg)

HWI\_INT5 Properties

General Dispatcher

☒ Use Dispatcher

Arg: \_hEdmaModule

## 5. EDMA Interrupt Dispatcher

Read IPR bits  
Determine which one is set  
Call corresponding handler (ISR) in Fxn Table

## 6. ISR (interrupt handler)

```
void edma_xmt_isr (void)
{
    SEM_post (&semaphore);
}
```

➤ How does the ISR Fxn Table (in #5 above) get loaded with the proper handler Fxn names?

```
edma_int_hook(TCC_EDMA_XEVT1, (EdmaTccHandler)&edma_xmt_isr);
```

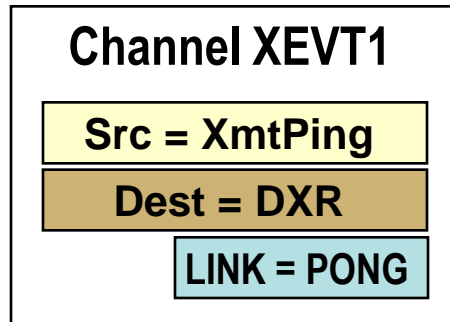
# Outline

- ◆ Introduction to EDMA3
- ◆ Example 1: Single Block Transfer
- ◆ Programming EDMA3 with CSL 3.0
- ◆ Example 2: Multiple Block Transfer
- ◆ Linking vs. Chaining
- ◆ QDMA
- ◆ IDMA

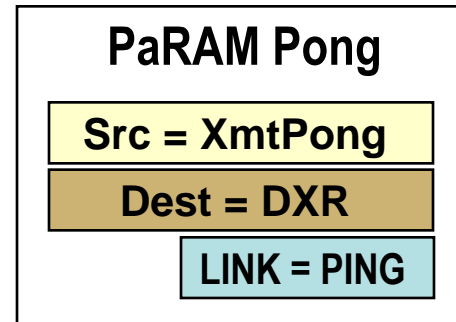
# Linking

- **Linking** – When a channel is done with its transfer, it uses the LINK field to determine which PSET will be used to re-load the channel's PSET register set.
- **Linking** does NOT cause a trigger to occur.

## Channel's PSET



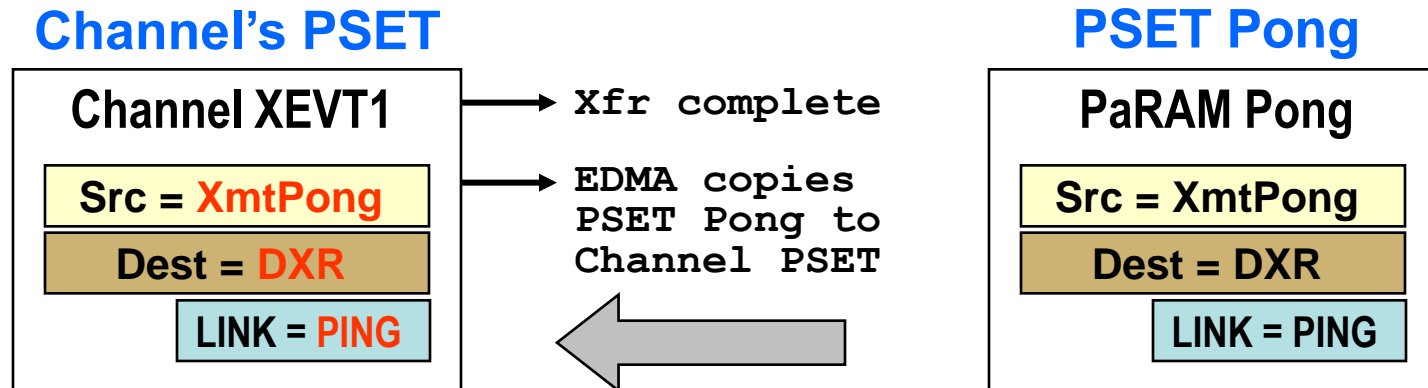
## PSET Pong





# Linking

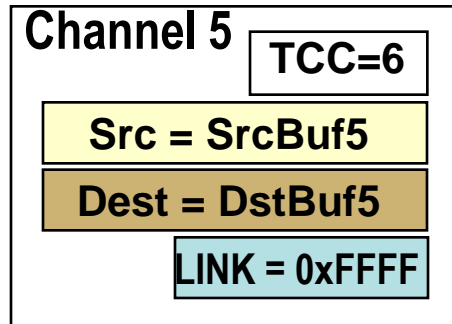
- **Linking** – When a channel is done with its transfer, it uses the LINK field to determine which PSET will be used to re-load the channel's PSET register set.
- **Linking** does NOT cause a trigger to occur.



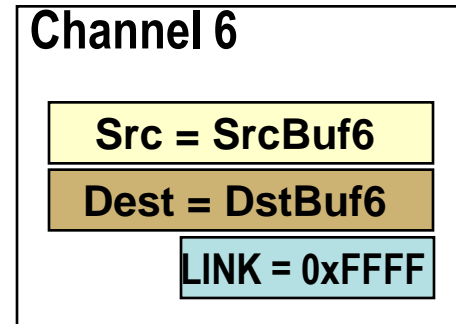
# Chaining

- **Linking** – When a channel is done with its transfer, it uses the LINK field to determine which PSET will be used to re-load the channel's PSET register set.
- **Linking** does NOT cause a trigger to occur.

## Channel 5's PSET



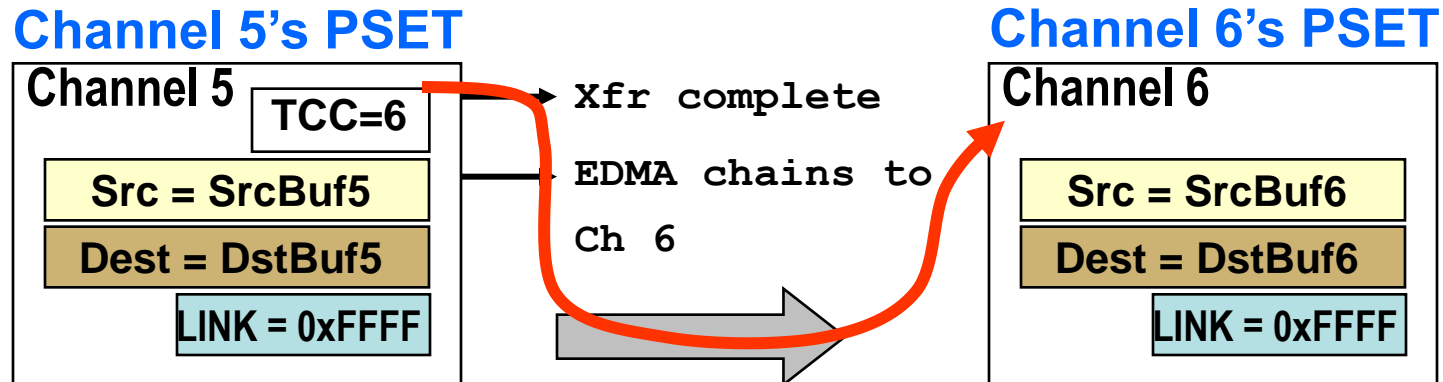
## Channel 6's PSET



- **Chaining** – The TCC of one channel is set to trigger any channel to run when the current channel is finished. For example, Ch #5 has OPT.TCC=6 which can trigger Ch #6 to run via the CER (Chain Event Register).

# Chaining

- **Linking** – When a channel is done with its transfer, it uses the LINK field to determine which PSET will be used to re-load the channel's PSET register set.
- **Linking** does NOT cause a trigger to occur.

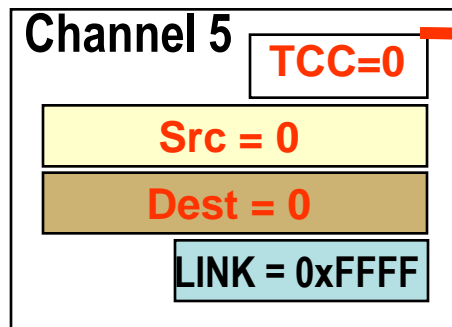


- **Chaining** – The TCC of one channel is set to trigger any channel to run when the current channel is finished. For example, Ch #5 has OPT.TCC=6 which can trigger Ch #6 to run via the CER (Chain Event Register).

# Chaining

- **Linking** – When a channel is done with its transfer, it uses the LINK field to determine which PSET will be used to re-load the channel's PSET register set.
- **Linking** does NOT cause a trigger to occur.
- LINK = 0xFFFF = Link-to-NULL. The PSET will be set to all 0's.

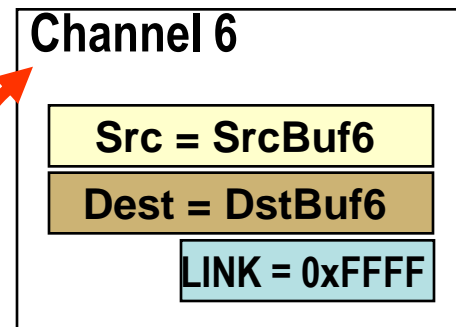
## Channel 5's PSET



Xfr complete

EDMA chains to  
Ch 6

## Channel 6's PSET

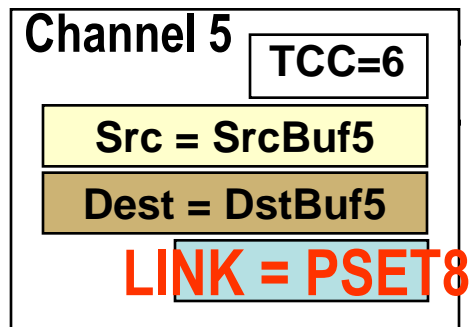


- **Chaining** – The TCC of one channel is set to trigger any channel to run when the current channel is finished. For example, Ch #5 has OPT.TCC=6 which can trigger Ch #6 to run via the CER (Chain Event Register).
- Linking will also be performed along with chaining.

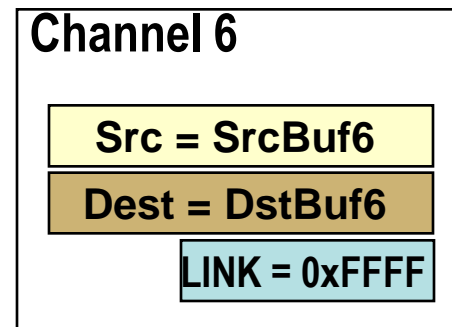
# Linking & Chaining Combined

- **Linking** – When a channel is done with its transfer, it uses the LINK field to determine which PSET will be used to re-load the channel's PSET register set.
- **Linking** does NOT cause a trigger to occur.
- LINK = 0xFFFF = Link-to-NULL. The PSET will be set to all 0's.

## Channel 5's PSET



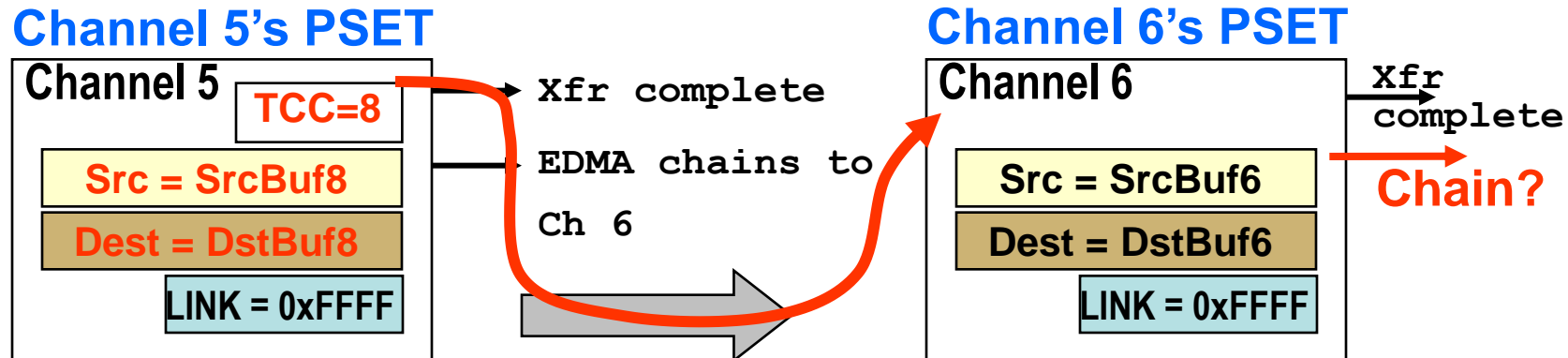
## Channel 6's PSET



- **Chaining** – The TCC of one channel is set to trigger any channel to run when the current channel is finished. For example, Ch #5 has OPT.TCC=6 which can trigger Ch #6 to run via the CER (Chain Event Register).
- Linking will also be performed along with chaining. For example, when Ch #5 is done, it links (copies) PSET #8 and at the same time triggers (OPT.TCC=6) Ch #6 to run. When Ch #6 is done, it can link to restore its PSET and also chain to a third channel or back to Ch #5.

# Linking & Chaining Combined

- **Linking** – When a channel is done with its transfer, it uses the LINK field to determine which PSET will be used to re-load the channel's PSET register set.
- **Linking** does NOT cause a trigger to occur.
- LINK = 0xFFFF = Link-to-NULL. The PSET will be set to all 0's.

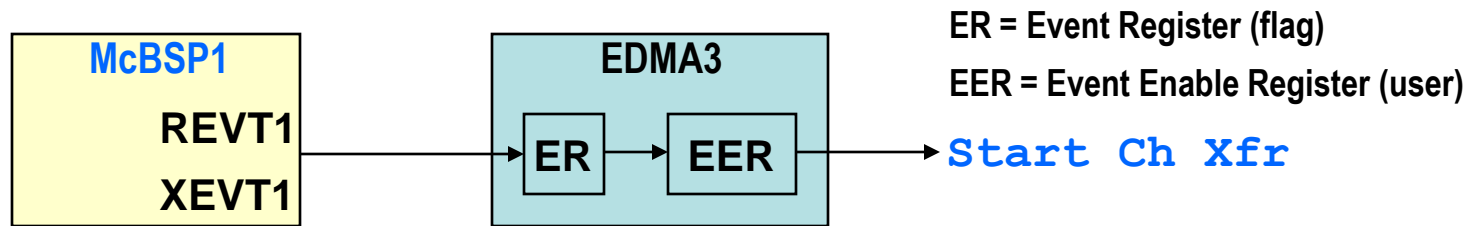


- **Chaining** – The TCC of one channel is set to trigger any channel to run when the current channel is finished. For example, Ch #5 has OPT.TCC=6 which can trigger Ch #6 to run via the CER (Chain Event Register).
- Linking will also be performed along with chaining. For example, when Ch #5 is done, it links (copies) PSET #8 and at the same time triggers (OPT.TCC=6) Ch #6 to run. When Ch #6 is done, it can link to restore its PSET and also chain to a third channel or back to Ch #5.

# Reminder: Triggering Transfers

➤ There are three ways to trigger an EDMA transfer:

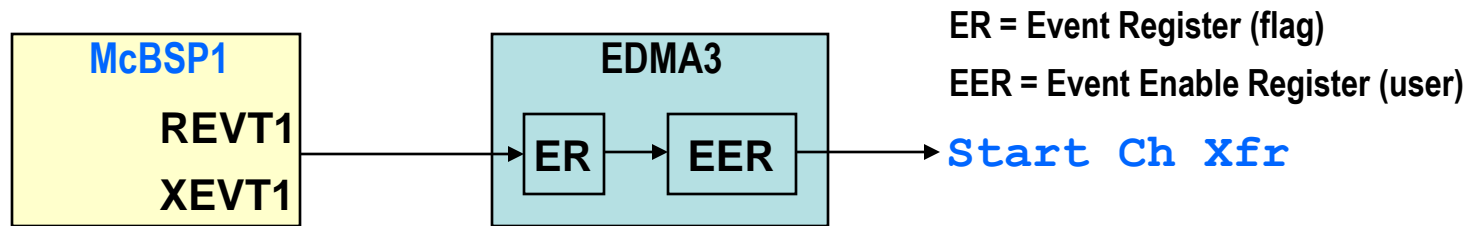
## 1 Event Sync from peripheral



# Reminder: Triggering Transfers

➤ There are three ways to trigger an EDMA transfer:

## 1 Event Sync from peripheral



## 2 Manually Trigger the Channel to Run

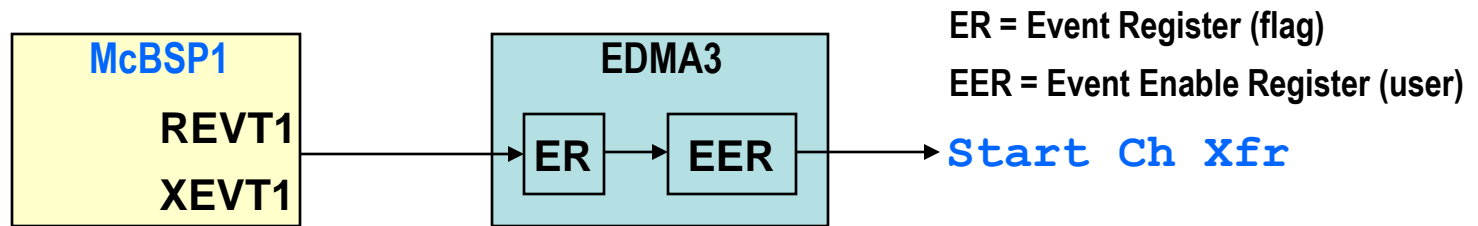




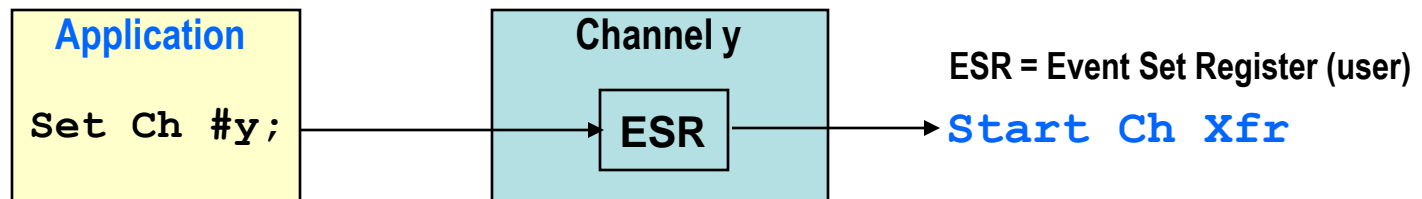
# Reminder: Triggering Transfers

➤ There are three ways to trigger an EDMA transfer:

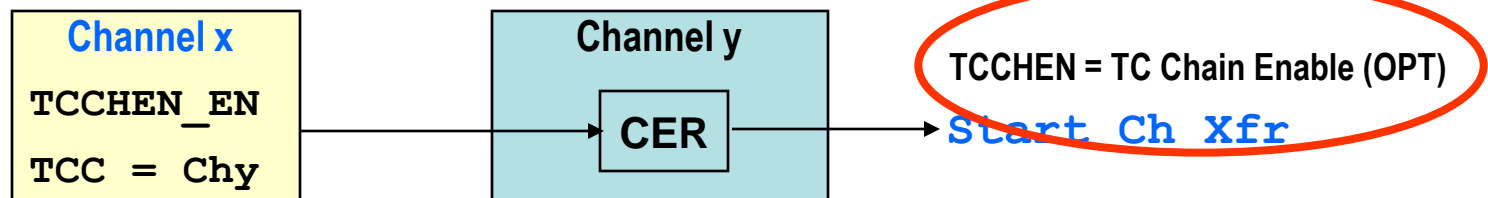
## 1 Event Sync from peripheral



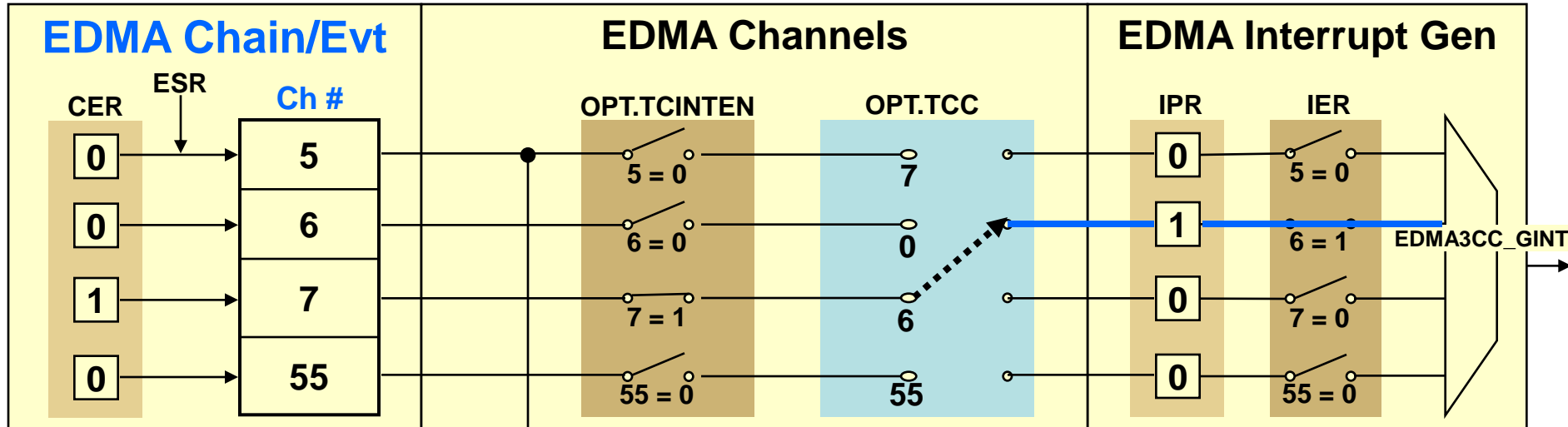
## 2 Manually Trigger the Channel to Run



## ✓ 3 Chain Event from another channel (next example...)



# Chaining Example Overview

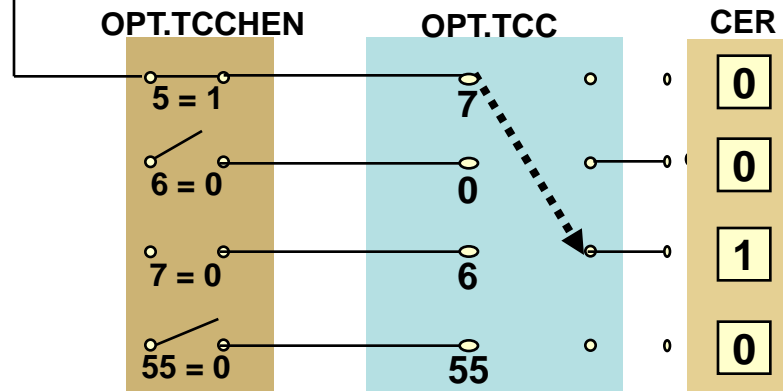


## Channel #5

- 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



CER = Chain Evt Reg

ESR – Evt Set Reg

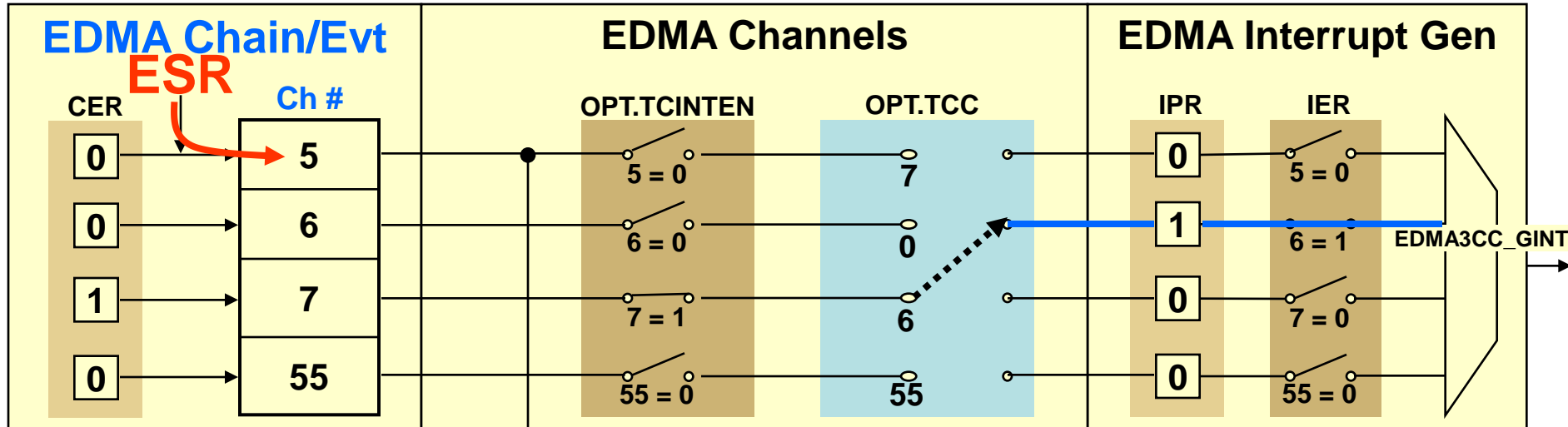
TCINTEN = “Final” TCC will interrupt the CPU

TCCHEN = “Final” TCC will chain to next channel

## 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 completed Ch's TCC setting

# Chaining Example 1

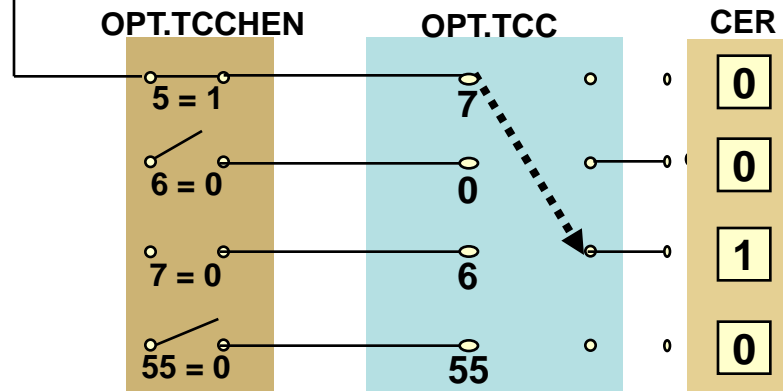


## Channel #5

- 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



CER = Chain Evt Reg

ESR – Evt Set Reg

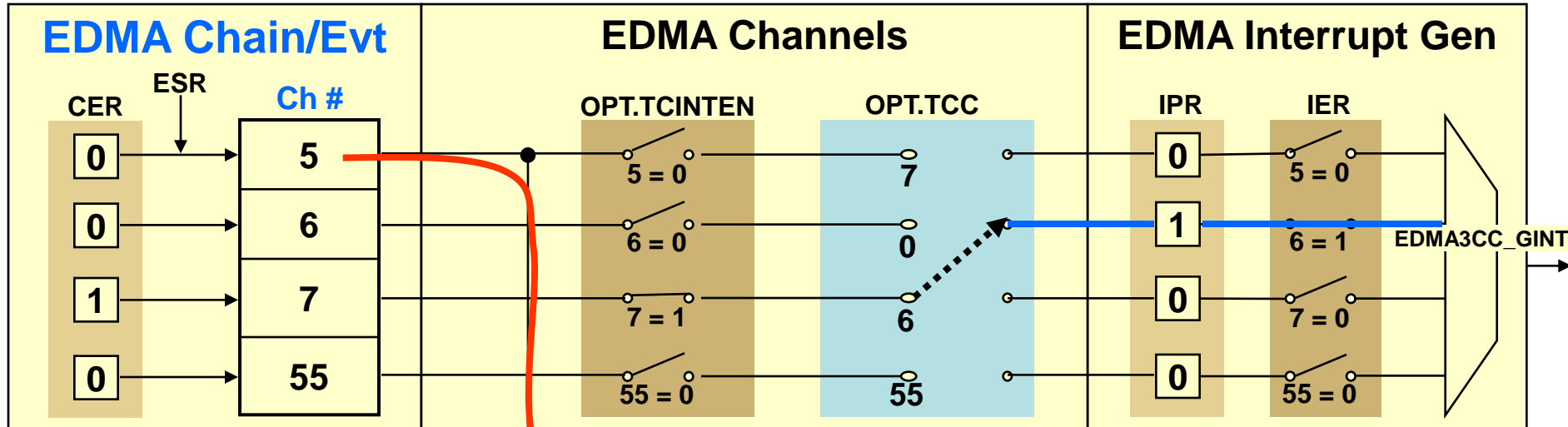
TCINTEN = “Final” TCC will interrupt the CPU

TCCHEN = “Final” TCC will chain to next channel

## 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 completed Ch's TCC setting

# Chaining Example 1

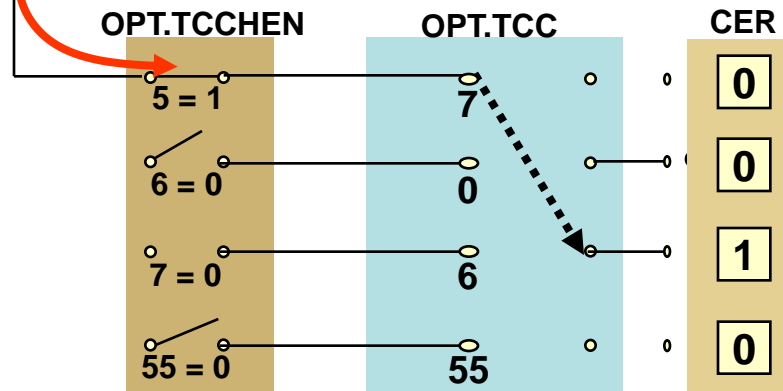


## Channel #5

- 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



CER = Chain Evt Reg

ESR – Evt Set Reg

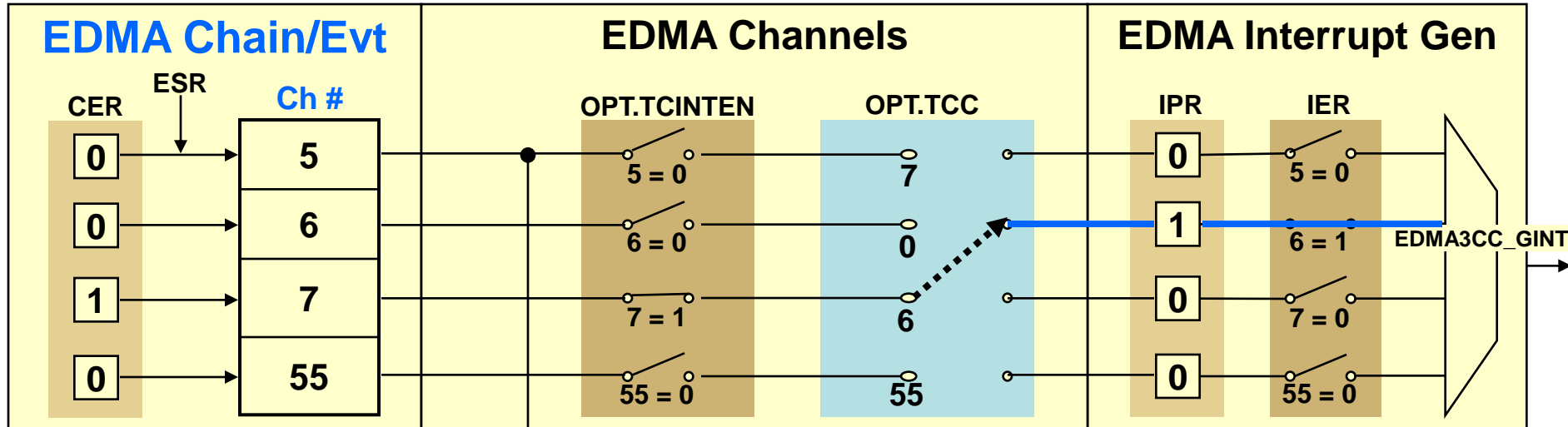
TCINTEN = “Final” TCC will interrupt the CPU

TCCHEN = “Final” TCC will chain to next channel

## 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 completed Ch's TCC setting

# Chaining Example 1

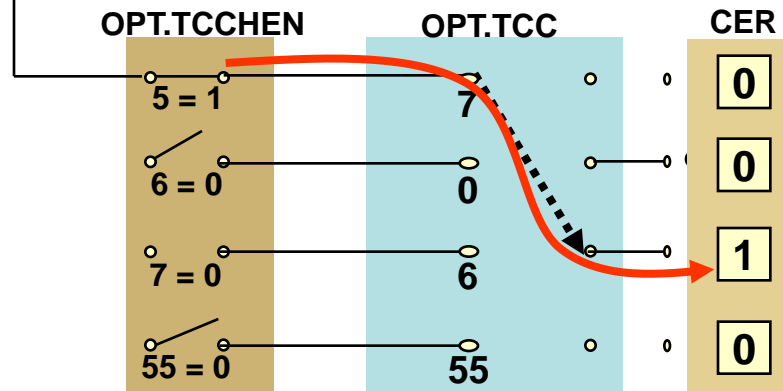


## Channel #5

- 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



CER = Chain Evt Reg

ESR – Evt Set Reg

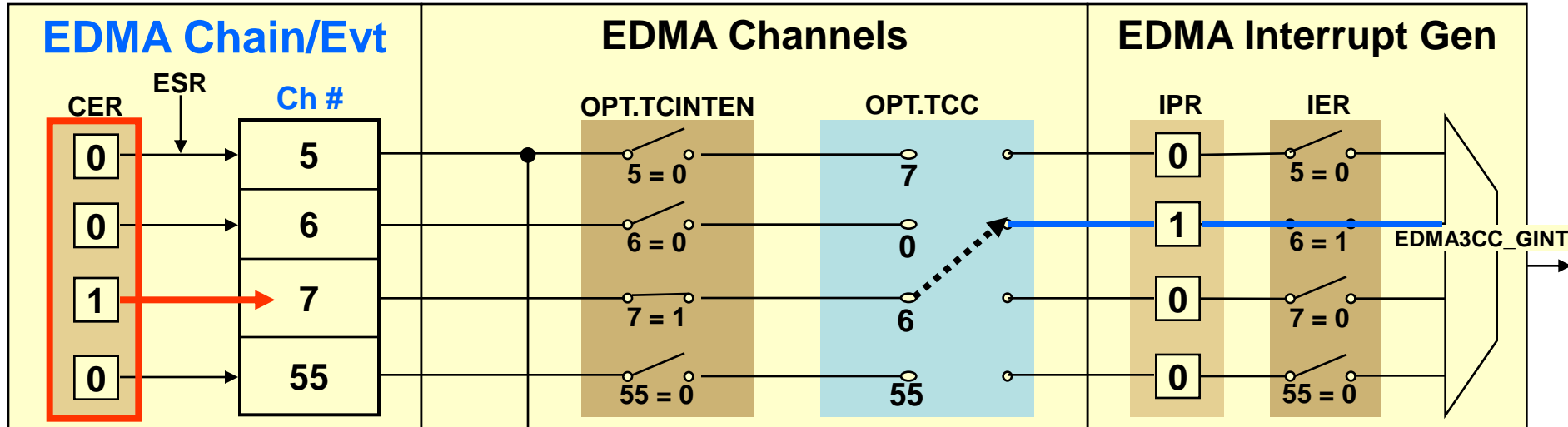
TCINTEN = “Final” TCC will interrupt the CPU

TCCHEN = “Final” TCC will chain to next channel

## 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 completed Ch's TCC setting

# Chaining Example 2

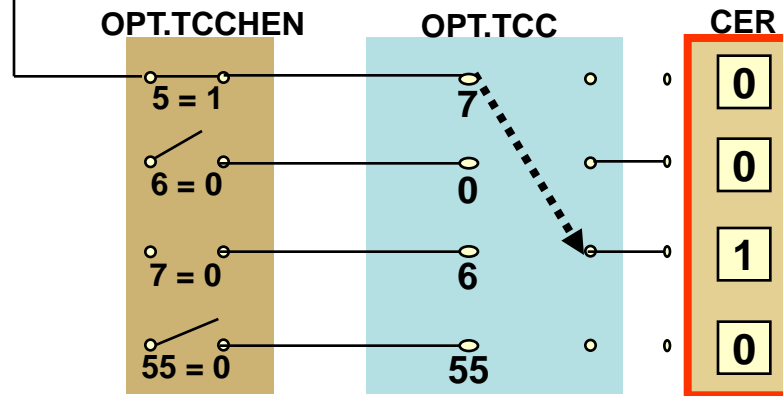


## Channel #5

- 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



CER = Chain Evt Reg

ESR – Evt Set Reg

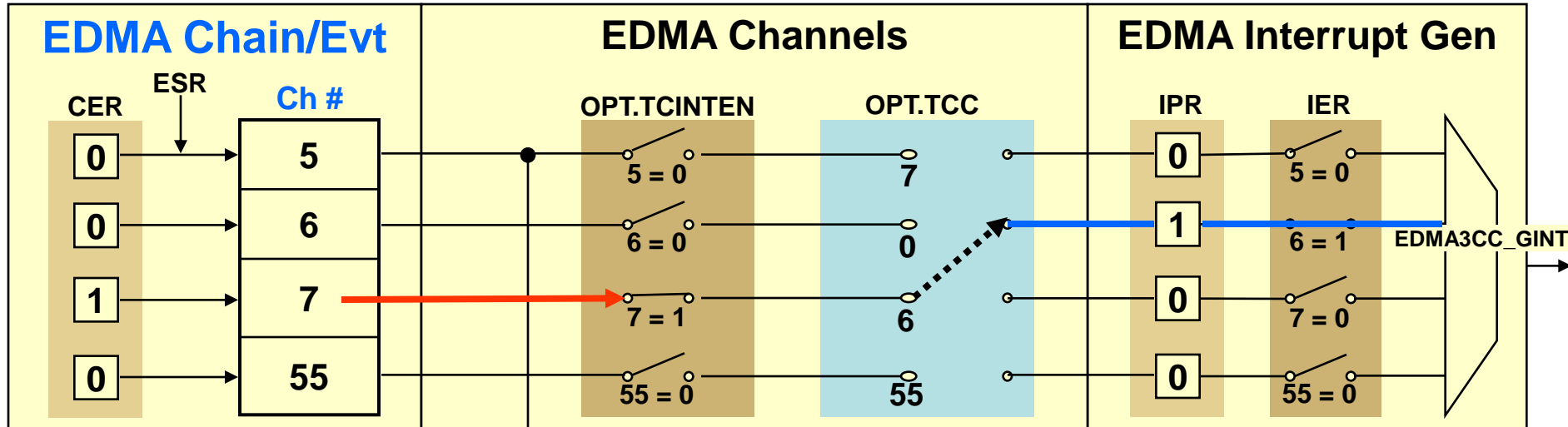
TCINTEN = “Final” TCC will interrupt the CPU

TCCHEN = “Final” TCC will chain to next channel

## 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 completed Ch's TCC setting

# Chaining Example 2

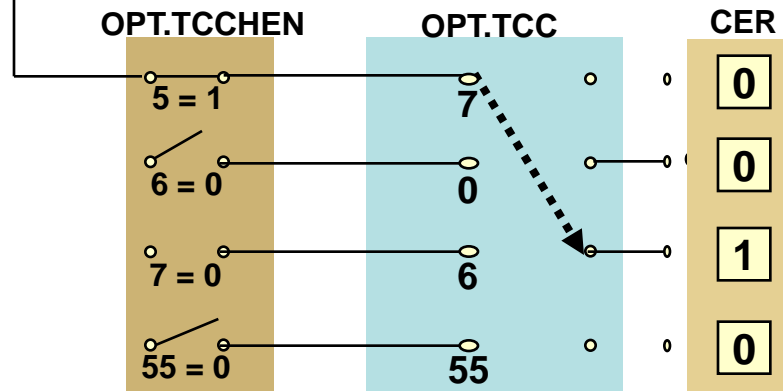


## Channel #5

- 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



CER = Chain Evt Reg

ESR – Evt Set Reg

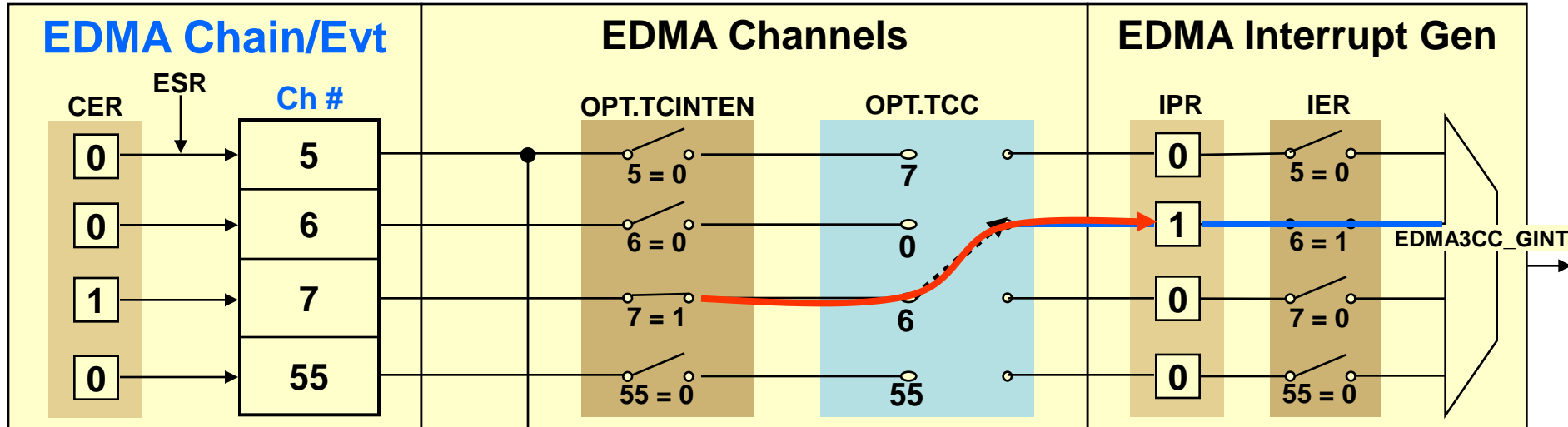
TCINTEN = “Final” TCC will interrupt the CPU

TCCHEN = “Final” TCC will chain to next channel

## 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 completed Ch's TCC setting

# Chaining Example 2

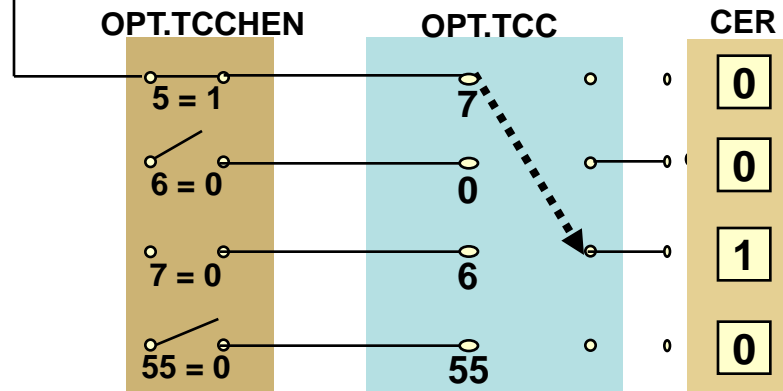


## Channel #5

- 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



CER = Chain Evt Reg

ESR – Evt Set Reg

TCINTEN = “Final” TCC will interrupt the CPU

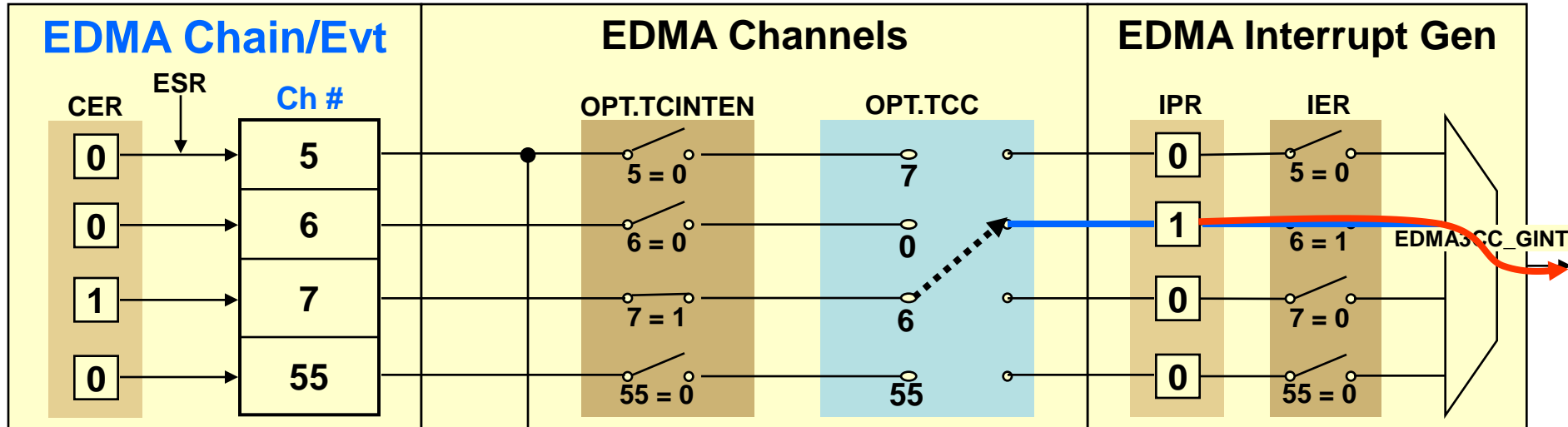
TCCHEN = “Final” TCC will chain to next channel

## 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 completed Ch's TCC setting



# Chaining Example 2

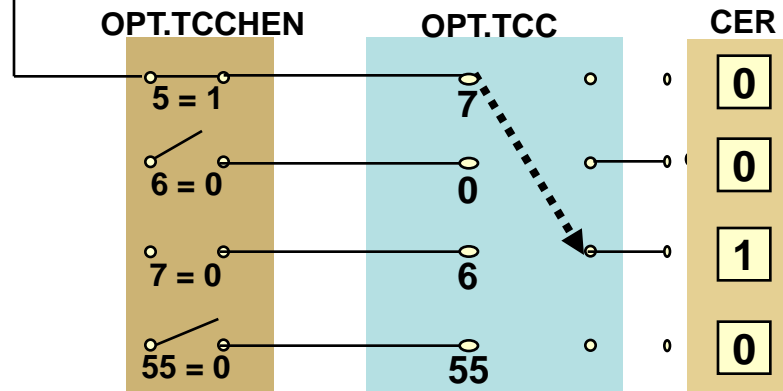


## Channel #5

- 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



CER = Chain Evt Reg

ESR – Evt Set Reg

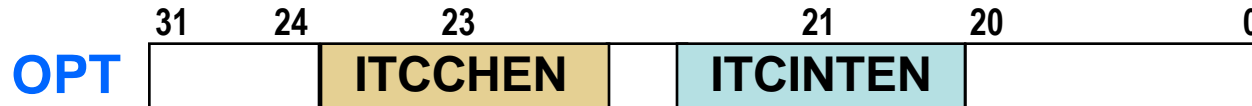
TCINTEN = “Final” TCC will interrupt the CPU

TCCHEN = “Final” TCC will chain to next channel

## Notes:

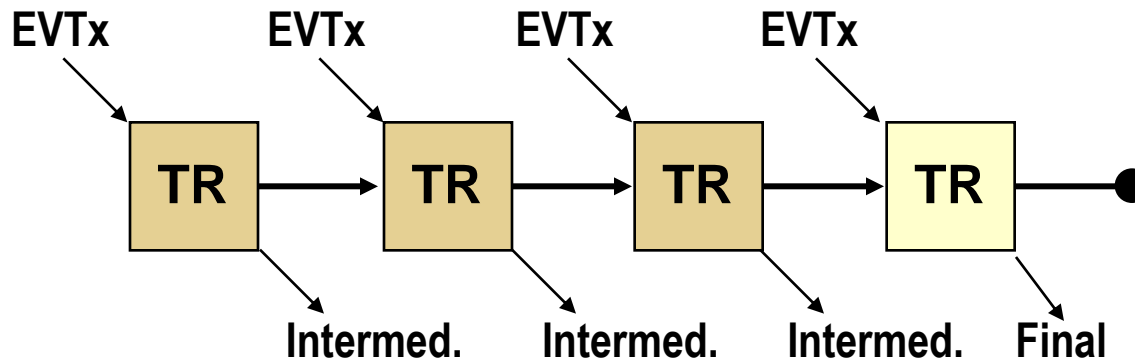
- Any Ch can chain to any other Ch by enabling OPT.TCCHEN and specifying the next TCC
- Any Ch can interrupt the CPU by enabling its OPT.TCINTEN option (and specifying the TCC)
- IPR bit set depends on completed Ch's TCC setting

# Intermediate Transfer Completion



*Intermediate transfer completion* indicates a TR has been completed EXCEPT THE LAST

- ◆ Chain Event Register (CER[TCC]) set if selected by ITCCHEN (“intermediate” chaining)
- ◆ Interrupt Pending Register (IPR[TCC]) set if selected by ITCINTEN (this will interrupt the CPU)



EVTx =

- ER (sync)
- ESR (manual)
- CER (chain)

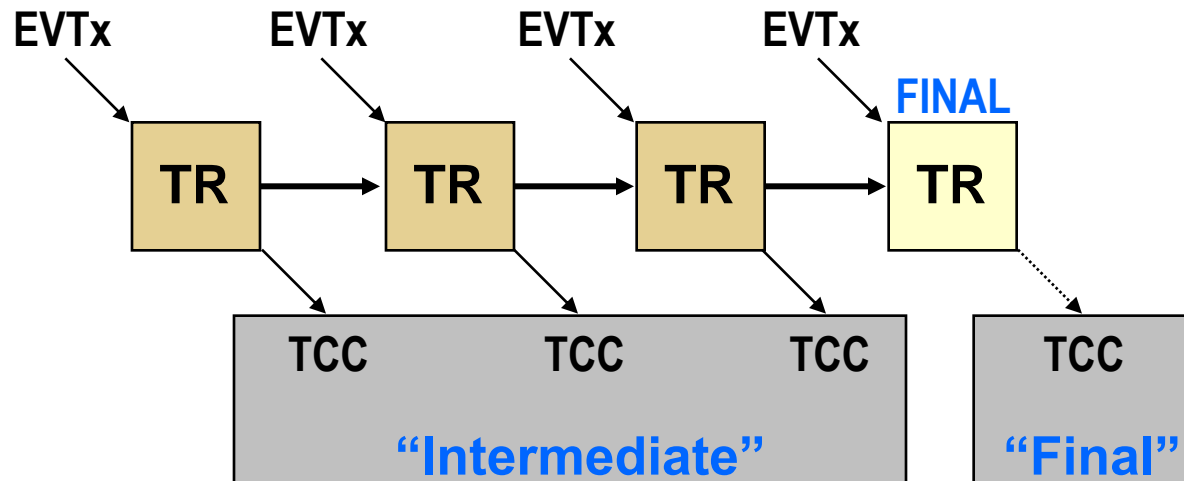
- ◆ Reminder: A TR (transfer request) can either be by ACNT bytes (A-sync) or A\*B bytes (AB-sync)
- ◆ “Intermediate” completion is for all TRs of a transfer EXCEPT the LAST. “Final” TCC is for only the LAST TR of a transfer.

# Intermediate vs. Final Completion

Figure 2-8. Channel Options Parameter (OPT)

31	30	28	27	24	23	22	21	20	19	18	17	16
PRIV	Reserved	PRIVID	ITCCHEN	TCCHEN	ITCINTEN	TCINTEN	Reserved	TCC				
R-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0				
15	12	11	10	8	7			4	3	2	1	0
TCC	TCCMOD	FWID	Reserved				STATIC	SYNCDIM	DAM	SAM		
R/W-0	R/W-0	R/W-0	R/W-0				R/W-0	R/W-0	R/W-0	R/W-0		

- ◆ In the example below, BOTH “Intermediate” and “Final” Completion are being used in the same transfer.
- ◆ If a transfer has multiple TRs (as shown below), “Intermediate” completion will generate a TCC code after every TR – EXCEPT THE LAST. “Intermediate” completion is configured by setting the OPT.ITCCHEN bit.
- ◆ “Final” completion is generated only on the LAST (FINAL) TR. Depending on your system, you could enable OPT.TCCHEN to chain after the last TR or send a CPU interrupt by enabling OPT.TCINTEN, or both.



# Outline

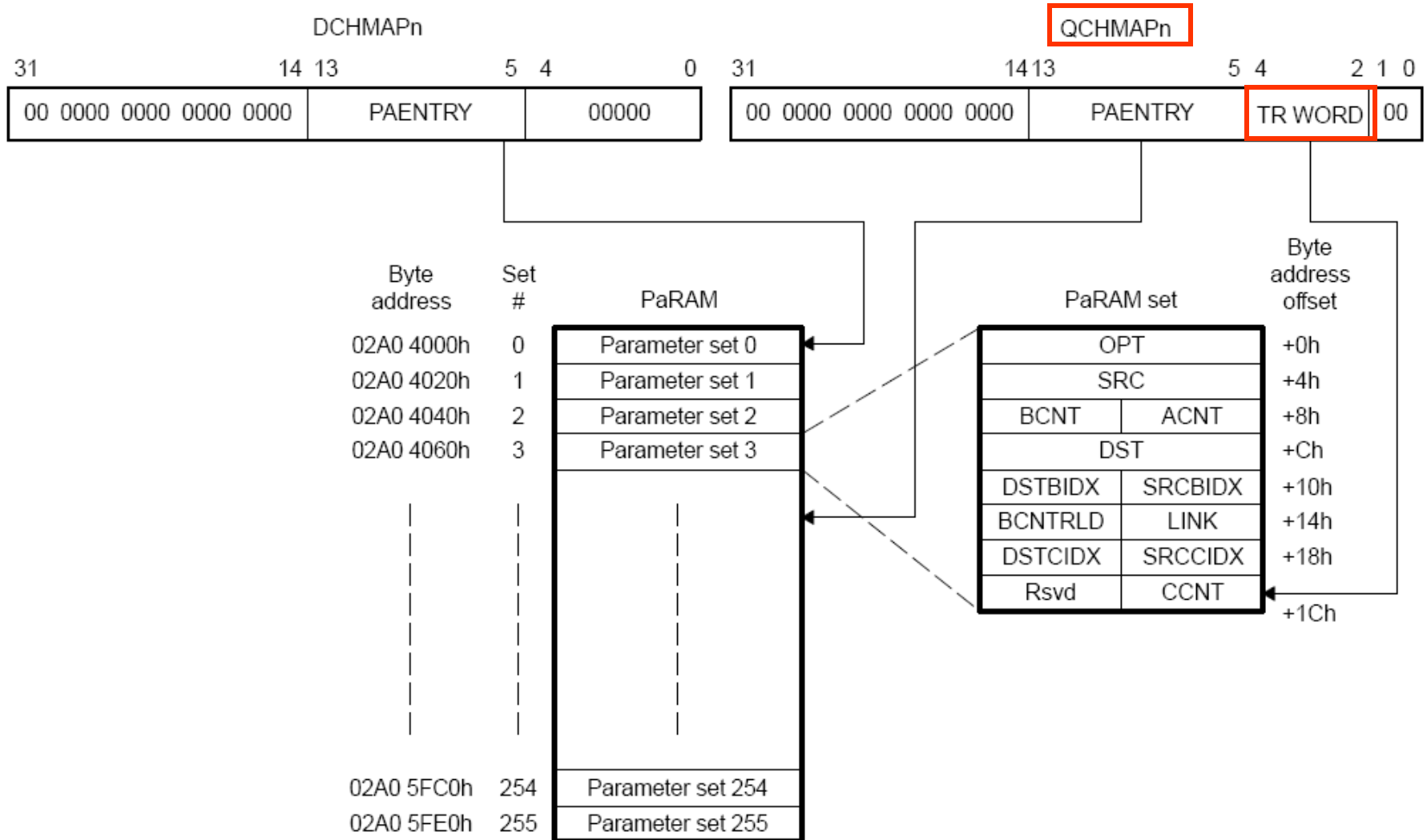
- ◆ Introduction to EDMA3
- ◆ Example 1: Single Block Transfer
- ◆ Programming EDMA3 with CSL 3.0
- ◆ Example 2: Multiple Block Transfer
- ◆ Linking vs. Chaining
- ◆ QDMA
- ◆ IDMA

# QDMA = Quick DMA

- ◆ QDMA is used for simple transfers where syncing to an event is not required. Address/count updates and linking are not performed. CCNT = 1 (single event transfer).
- ◆ A transfer can be triggered by two methods:  
(1) writing to a trigger word; (2) using the CSL DAT module.
- ◆ It's “quick” because the CPU can initiate a transfer with as few as ONE write to a channel register
- ◆ How does it work?
  - QDMA channel is “auto-triggered” when CPU writes to the “trigger” word
  - Eliminates the need to write to PSET and kick off transfer w/ separate write to ESR
  - Selection of the trigger word allows CPU to modify only words of interest in a PSET
  - Assumes OPT.STATIC = 1. Count and address updates and linking NOT performed.
- ◆ Example:
  - If ACNT/BCNT/CCNT are typically static for a given algorithm, but SRC is different for each transfer, then SRC could be defined as the trigger word. CPU can initiate a transfer with a single write to the SRC address for the specified PSET.

# QDMA Mapping

## DMA Channel and QDMA Channel to PaRAM Mapping



# DAT Module: QDMA Made Easy

```
#include <csl_dat.h>                                     // DAT Module header file
DAT_Setup datSetup;                                       // use for QDMA example
int32_t id;
uint32_t fillVal;

datSetup.qchNum = CSL_DAT_QCHA_0;                        // pick a QDMA channel 0-7
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;
datSetup.tccNum = 0;                                     // pick a TCC
datSetup.paramNum = 0;                                   // pick a PSET
datSetup.priority = CSL_DAT_PRI_1;                       // pick a queue/TC (0-5)
DAT_open(&datSetup);

// Fill a linear block of memory with the specified fillVal using QDMA
fillVal = 0;
id = DAT_fill(gBufXmt, sizeof(gBufXmt), &fillVal);      // similar to memset()
id = DAT_fill(gBufRcv, sizeof(gBufRcv), &fillVal);
DAT_wait(id);                                            // optional

DAT_copy(gBufRcv, gBufXmt, BUFFSIZE);                  // similar to memcpy()
```

# Outline

- ◆ Introduction to EDMA3
- ◆ Example 1: Single Block Transfer
- ◆ Programming EDMA3 with CSL 3.0
- ◆ Example 2: Multiple Block Transfer
- ◆ Linking vs. Chaining
- ◆ QDMA
- ◆ IDMA



# IDMA = Internal DMA

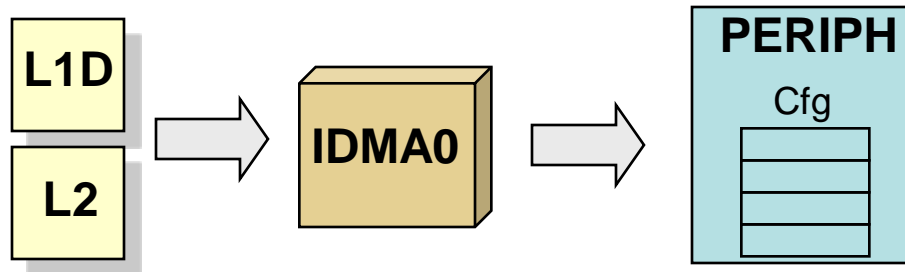
- **C64x+ IDMA – Performs background data movement or peripheral programming WITHOUT using EDMA bandwidth/resources or TeraNet SCR (internal to CorePac).**

# IDMA = Internal DMA

- C64x+ IDMA – Performs background data movement or peripheral programming **WITHOUT** using EDMA bandwidth/resources or TeraNet SCR (internal to CorePac).

## Channel 0 (IDMA0 – Hi Priority)

- Performs rapid programming of peripheral configuration registers
- Avoids unnecessary wait states through CFG bus vs. traditional use of the CPU copying config structures from L2 to the peripheral registers
- Typically used when new config structures are needed quickly. A copy of the structures can be stored in L1D/L2 and then transferred during run-time.

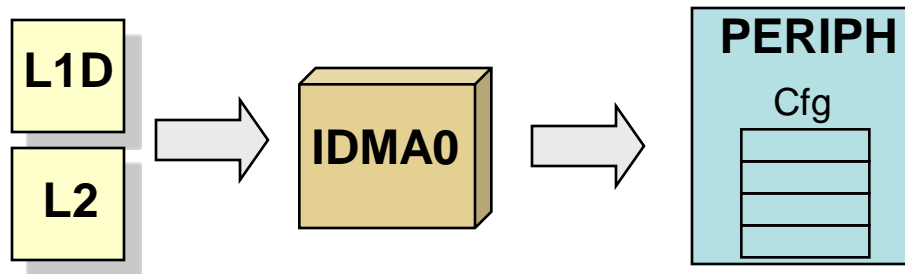


# IDMA = Internal DMA

- C64x+ IDMA – Performs background data movement or peripheral programming **WITHOUT** using EDMA bandwidth/resources or TeraNet SCR (internal to CorePac).

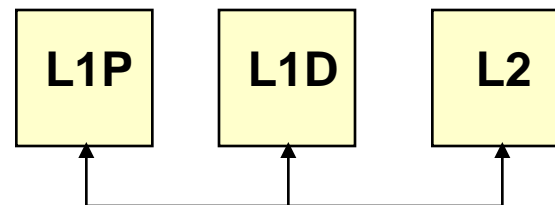
## Channel 0 (IDMA0 – Hi Priority)

- Performs rapid programming of peripheral configuration registers
- Avoids unnecessary wait states through CFG bus vs. traditional use of the CPU copying config structures from L2 to the peripheral registers
- Typically used when new config structures are needed quickly. A copy of the structures can be stored in L1D/L2 and then transferred during run-time.



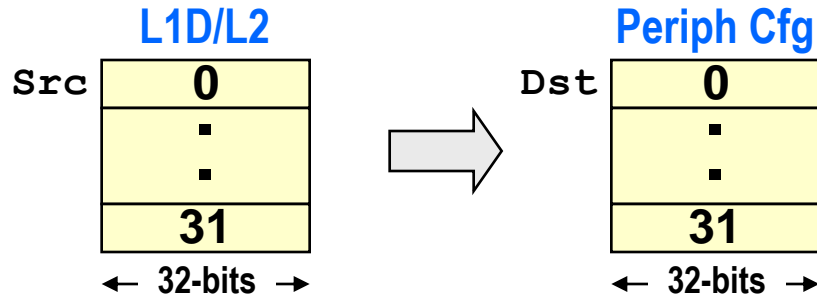
## Channel 1 (IDMA1 – Lo Priority)

- Rapid block transfers between L1P, L1D, L2



# IDMA0: Programming Details

- IDMA0 operates on a block of 32 contiguous 32-bit registers (both src/dst blocks must be aligned on a 32-word boundary). Optionally generate CPU interrupt if needed.
- User provides: Src, Dst, Count and “mask” (Reference: SPRU871)

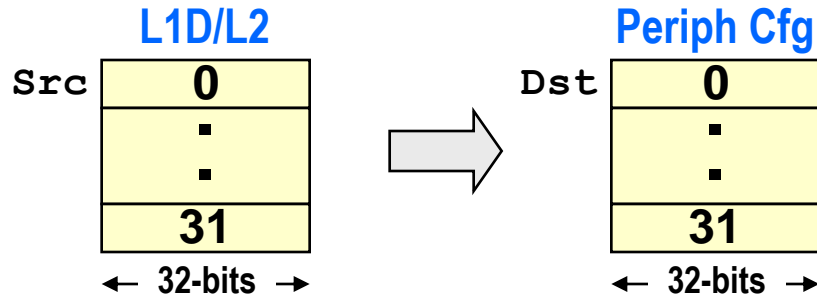


Count = # of 32-register blocks to xfr (up to 16)

Mask = 32-bit mask determines WHICH registers to transfer (“0” = xfr, “1” = NO xfr)

# IDMA0: Programming Details

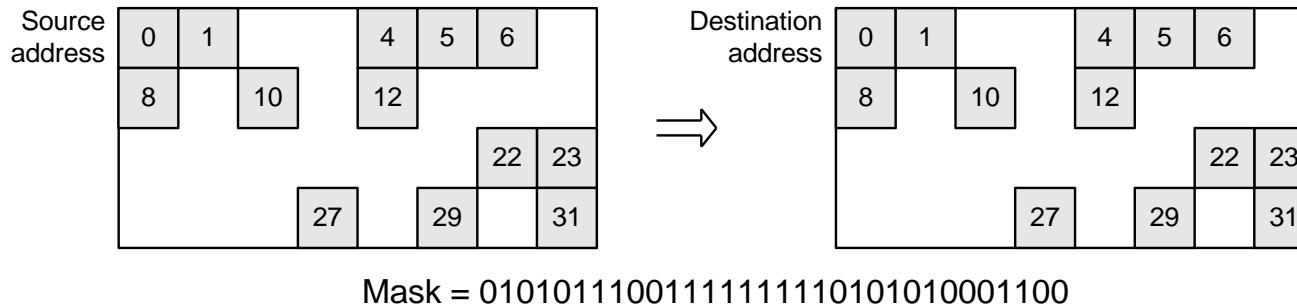
- IDMA0 operates on a block of 32 contiguous 32-bit registers (both src/dst blocks must be aligned on a 32-word boundary). Optionally generate CPU interrupt if needed.
- User provides: Src, Dst, Count and “mask” (Reference: SPRU871)



Count = # of 32-register blocks to xfr (up to 16)

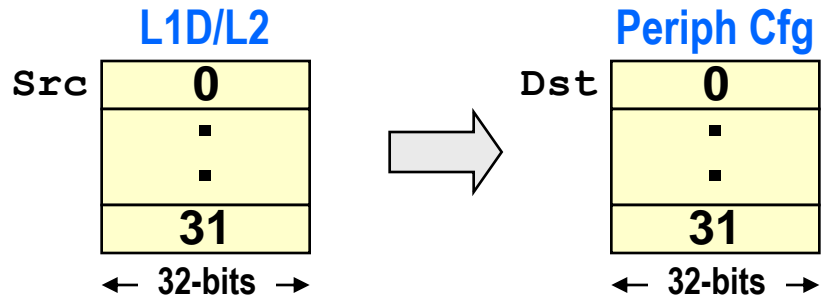
Mask = 32-bit mask determines WHICH registers to transfer (“0” = xfr, “1” = NO xfr)

- Example Transfer using MASK (not all regs typically need to be programmed):



# IDMA0: Programming Details

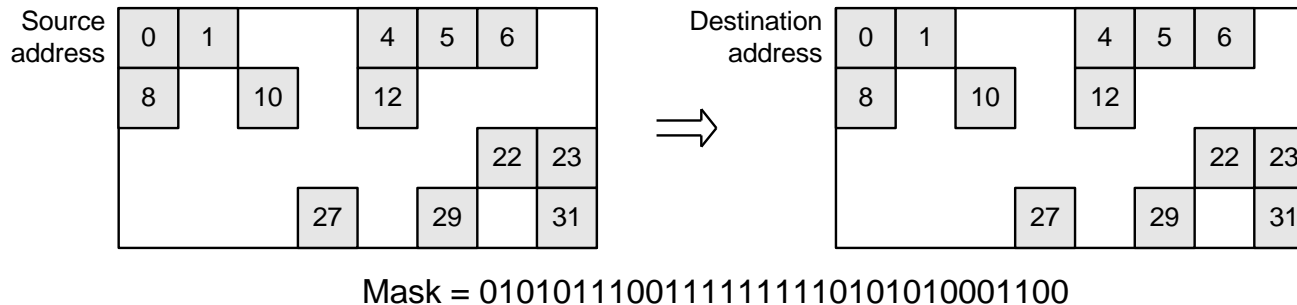
- IDMA0 operates on a block of 32 contiguous 32-bit registers (both src/dst blocks must be aligned on a 32-word boundary). Optionally generate CPU interrupt if needed.
- User provides: Src, Dst, Count and “mask” (Reference: SPRU871)



Count = # of 32-register blocks to xfr (up to 16)

Mask = 32-bit mask determines WHICH registers to transfer (“0” = xfr, “1” = NO xfr)

- Example Transfer using MASK (not all regs typically need to be programmed):

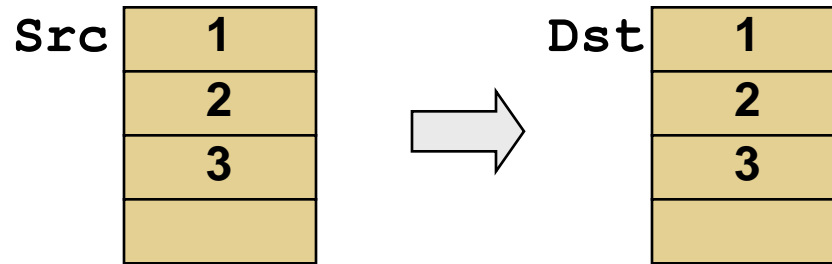


- User must write to IDMA0 registers in the following order (COUNT written – triggers transfer):

```
IDMA0_MASK = 0x573FEA8C;    //set mask for 13 regs above
IDMA0_SOURCE = reg_ptr;      //set src addr in L1D/L2
IDMA0_DEST = MMR_ADDRESS;    //set dst addr to config location
IDMA0_COUNT = 0;             //set mask for 1 block of 32 registers
```

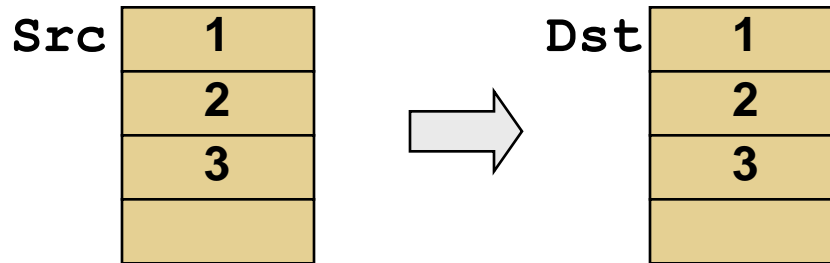
# IDMA1: Programming Details

- IDMA1 is optimized for contiguous burst transfers between L1P, L1D and L2



# IDMA1: Programming Details

- IDMA1 is optimized for contiguous burst transfers between L1P, L1D and L2

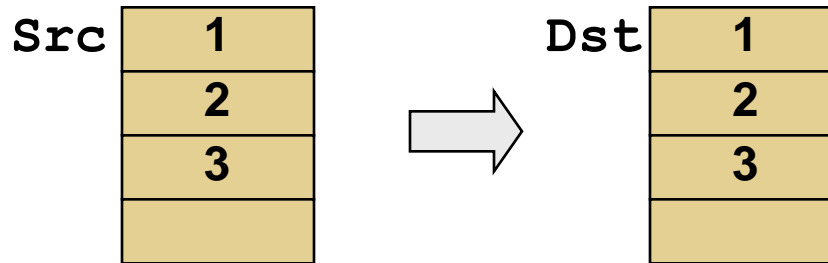


- Cannot access CFG port registers (only used for internal memory transfers)
- User provides: Src, Dst, Count (Reference: SPRU871)
- All src/dest addresses increment linearly throughout the transfer
- IDMA1\_COUNT = #bytes to transfer



# IDMA1: Programming Details

- IDMA1 is optimized for contiguous burst transfers between L1P, L1D and L2



- Cannot access CFG port registers (only used for internal memory transfers)
- User provides: Src, Dst, Count (Reference: SPRU871)
- All src/dest addresses increment linearly throughout the transfer
- IDMA1\_COUNT = #bytes to transfer
- Example:

```
IDMA1_SOURCE = outBuffFast;           //set src addr in L1D
IDMA1_DEST = outBuff;                 //set dst addr to L2
IDMA1_COUNT = 7 << IDMA_PRI_SHIFT |    //PRI low vs. cache/EDMA
              1 << IDMA_INT_SHIFT |    //interrupt CPU on completion
              bufsize;                 //set count to buffer size (bytes)
```