TI Developer Conference
March 7-9, 2007 • Dallas, TX

Minds in Motion

Harness the Power of EDMA3

Randy Preskitt
Sr. Member Technical Staff
DSP Field Apps
Dallas, Texas

Harness the Power of EDMA3

- Does your DSP have EDMA3?
- Introduction to EDMA3
- Use CSL 3 to program EDMA3
- Tips, tricks, tools, techniques

Harness the Power of EDMA3

- Does your DSP have EDMA3?
- Introduction to EDMA3
- Use CSL 3 to program EDMA3
- Tips, tricks, tools, techniques

Which DSPs Have EDMA3?

- All C64x+ devices:
  - C6451, C6452, C6454, C6455
  - TCI6482
  - DM6441, DM6443, DM6446
  - DM6431, DM6433, DM6435, DM6437
  - DM647, DM648
  - C6421, C6424
  - LP64x
  - Any new C64x+ DSPs

Harness the Power of EDMA3

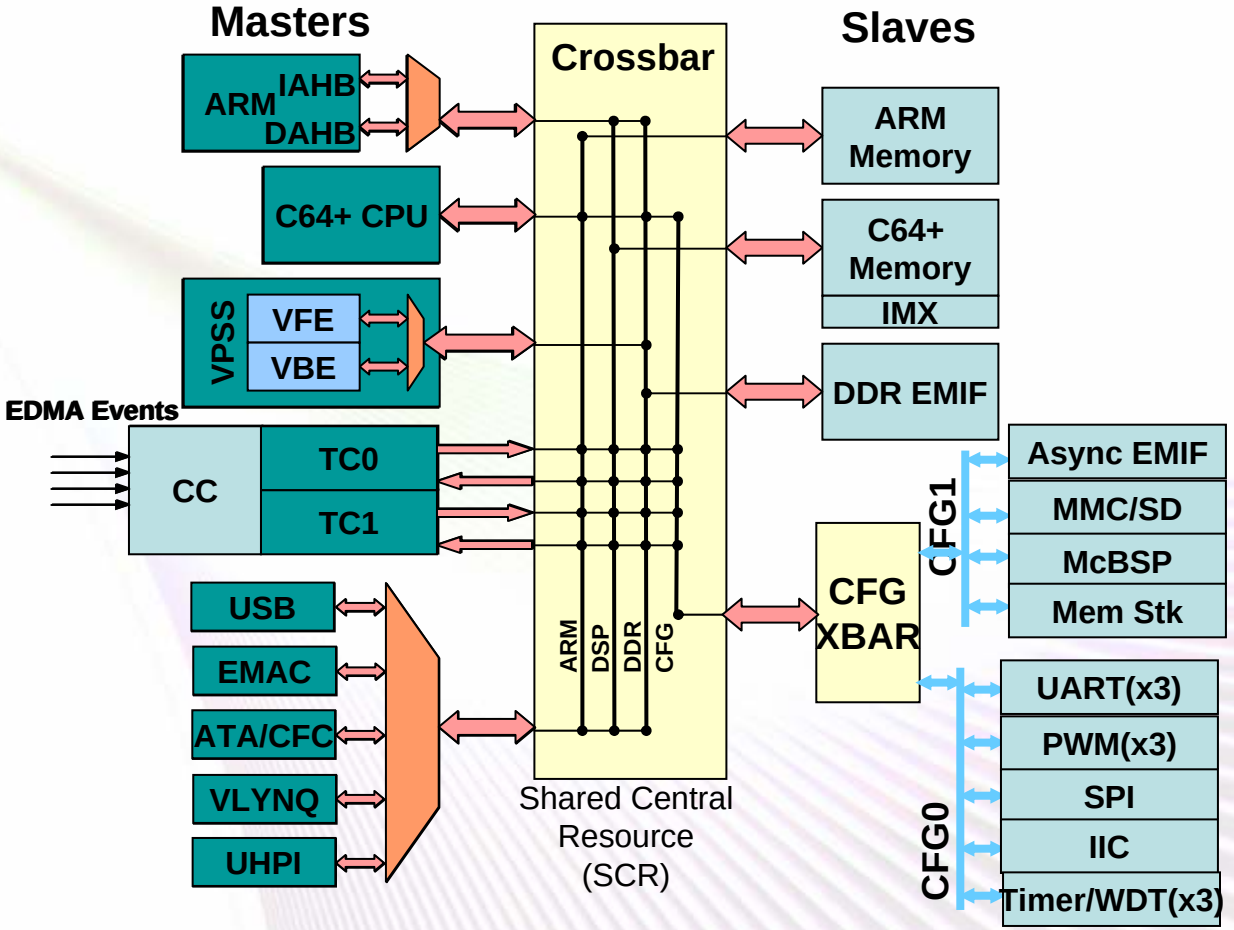
- Does your DSP have EDMA3?
- Introduction to EDMA3
- Use CSL 3 to program EDMA3
- Tips, tricks, tools, techniques

Introduction to EDMA3

- Channel Controller (CC)
  - Accepts and coordinates events
  - Sends Transfer Requests (TRs) to TC
- Transfer Controller (TC)
  - Handles actual transfers
  - Optimizes transfers for both source and destination
- Can do all the data movement for the CPU

System Architecture: DM644x

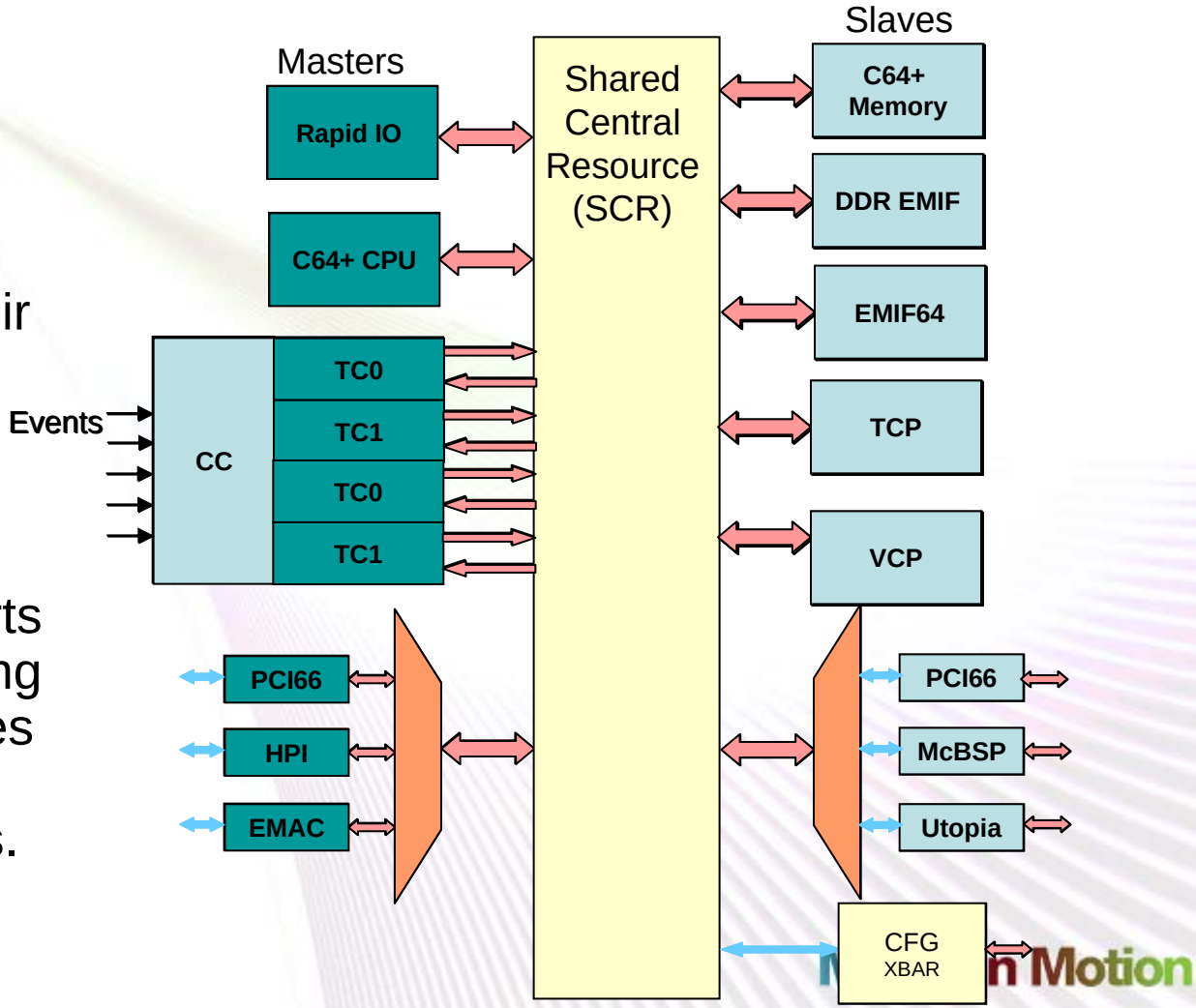
- ARM, DSP and VPSS have direct access to crossbar switch.
- Transfer Controllers have their own crossbar ports, too.
- Other master peripherals share a crossbar port.
- Slave peripherals are accessed by masters via the crossbar, aka Shared Central Resource (SCR).



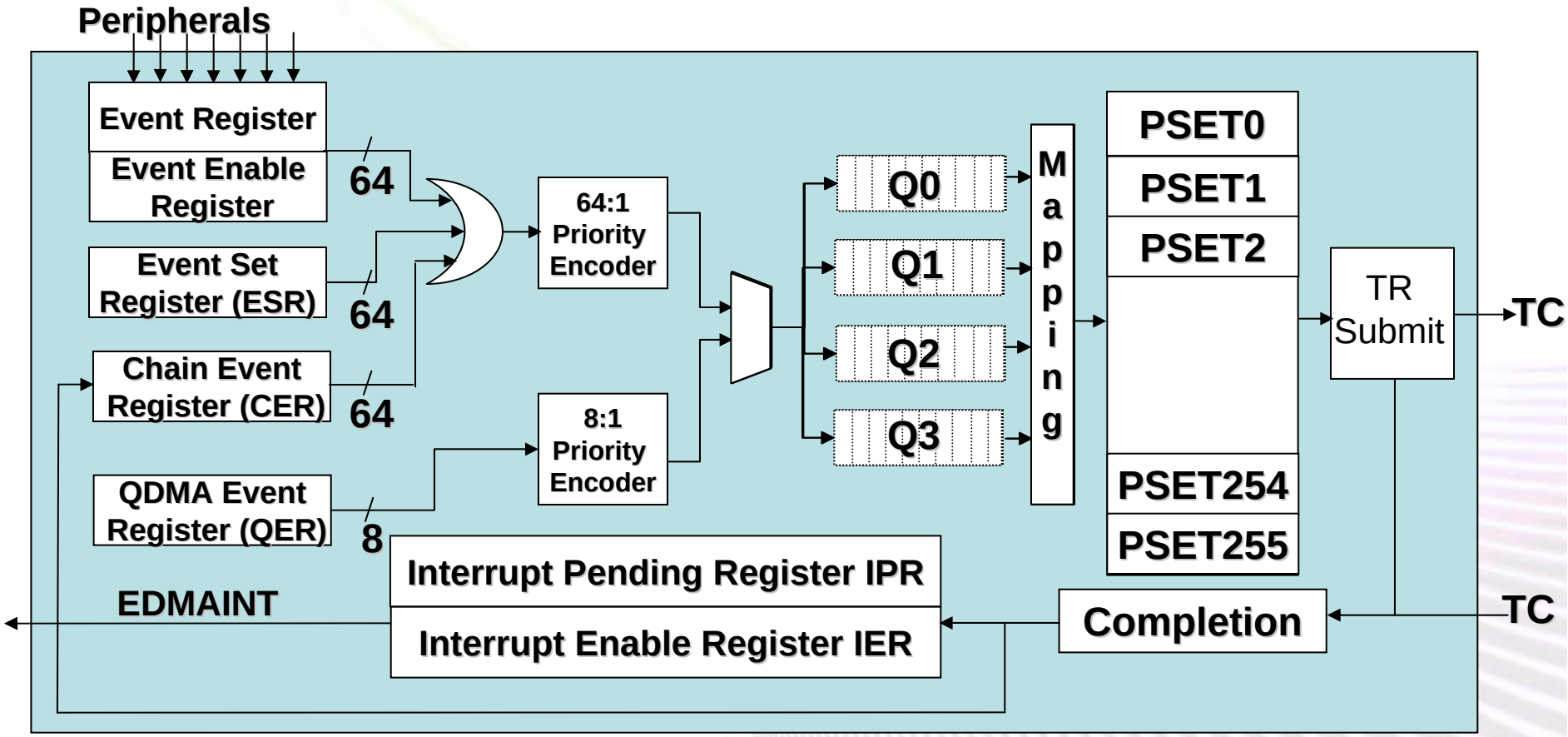
Minds in Motion

System Architecture: C645x

- C64x CPU and RapidIO have their own SCR ports.
- Four Transfer Controllers have their own SCR ports.
- Lower-bandwidth master peripherals share a port.
- Slave peripheral ports are shared depending on VBUS frequencies and widths. EMIFs have their own ports.



Channel Controller Architecture



EDMA3 Channel Trigger Types

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

- Event Triggering:
  - Used for peripheral driven transfers, such as McBSP, Utopia, etc.
  - "Triggered" by assertion of enabled Event Input signal.
  - Events captured in Event Register (ER[63:0]).
  - Events enabled in Event Enable Register (EER[63:0]).
- Chain Triggering:
  - Used to execute multiple Transfer Requests upon receipt of a single event. For example:
    - Ch0 is "Event Triggered" by Event0.
    - Ch0 completion can "chain-trigger" to Ch1 by programming OPT.TCC=1.
  - Chain Events captured in Chain Event Register (CER[63:0]).
  - Event does not need to be enabled.
- Manual Triggering:
  - Used to initiate a Transfer Request under CPU control.
  - Each Active Channel PaRAM entry can be "manually triggered" by the CPU writing a "1" to the corresponding bit of the "Event Set Register" (ESR[63:0]).
  - Event does not need to be enabled.

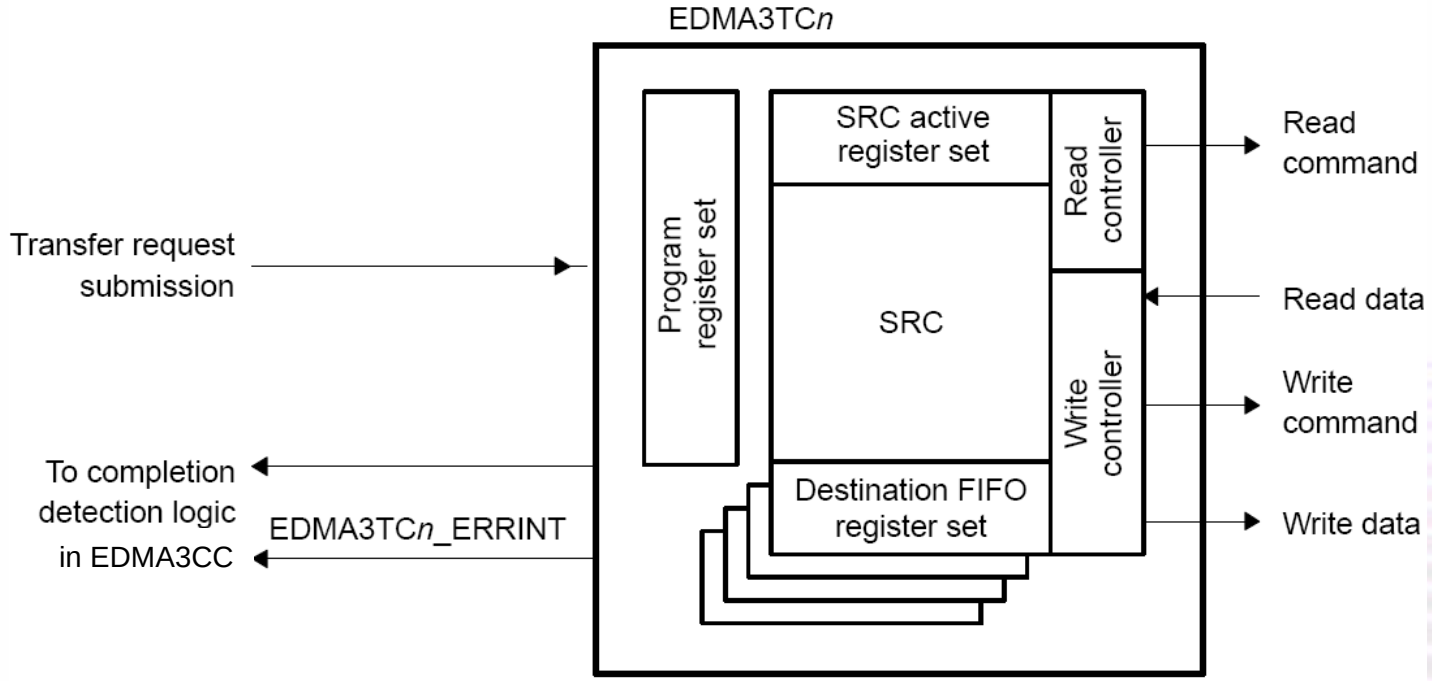
C6455 EDMA3 Events

- McBSP XEVT0/1
- McBSP REVT0/1
- UREVT
- UXEVT
- RIOINT1
- VCP2REVT
- VCP2XEVT
- TCP2REVT
- TCP2XEVT
- TEVTLO0
- TEVTHI0
- TEVTLO1
- TEVTHI1
- GPINT[15:0]
- DSP_EVT
- ICREVT
- ICXEVT

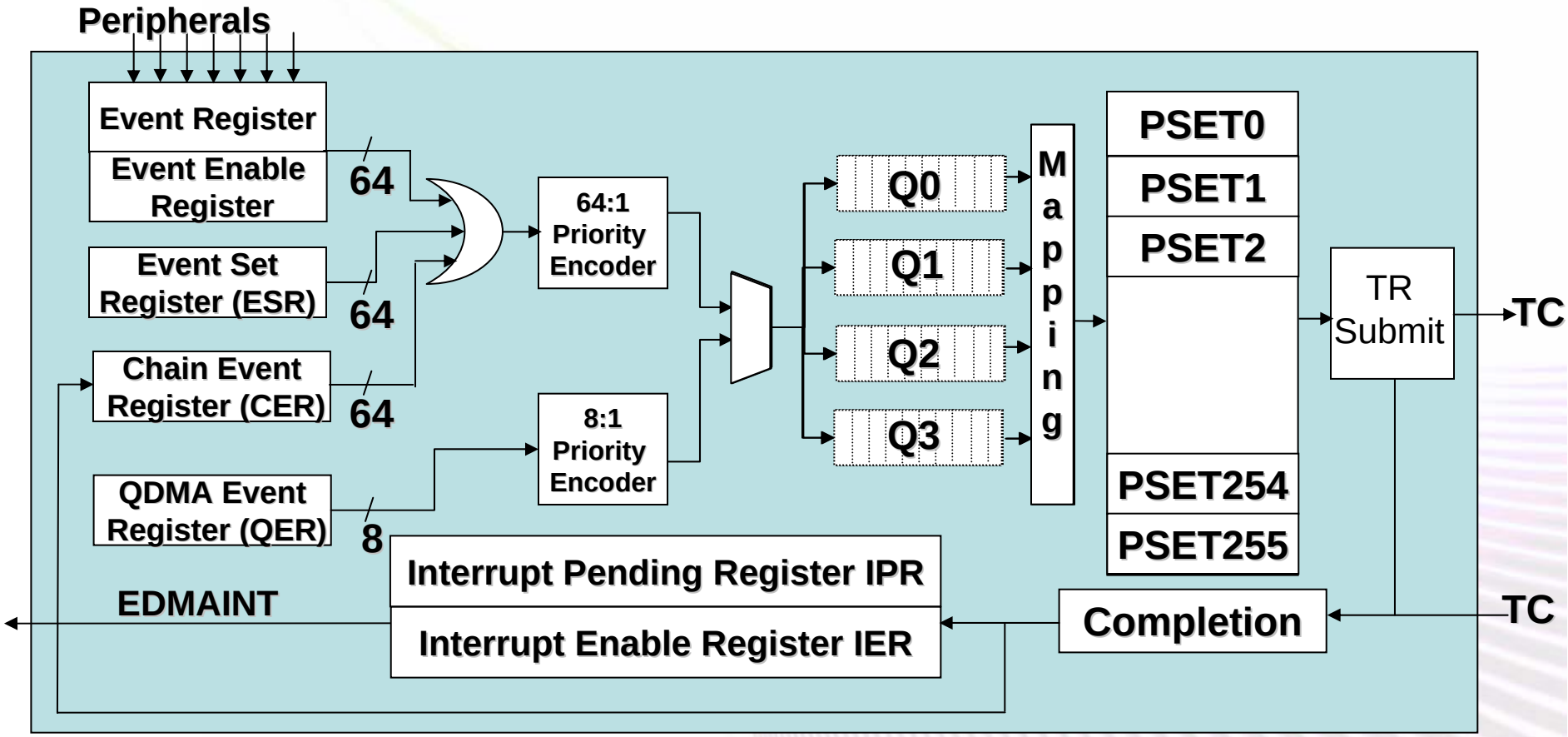
Transfer Controller Architecture

- CC sends a Transfer Request (TR).
- New TR waits in prog reg set.
- Current TR runs in active register set.
- Read ctlr pulls data into FIFOs.
- Write ctlr optimizes writes for dest.
- Multiple TR reads can be buffered in FIFOs waiting to write.

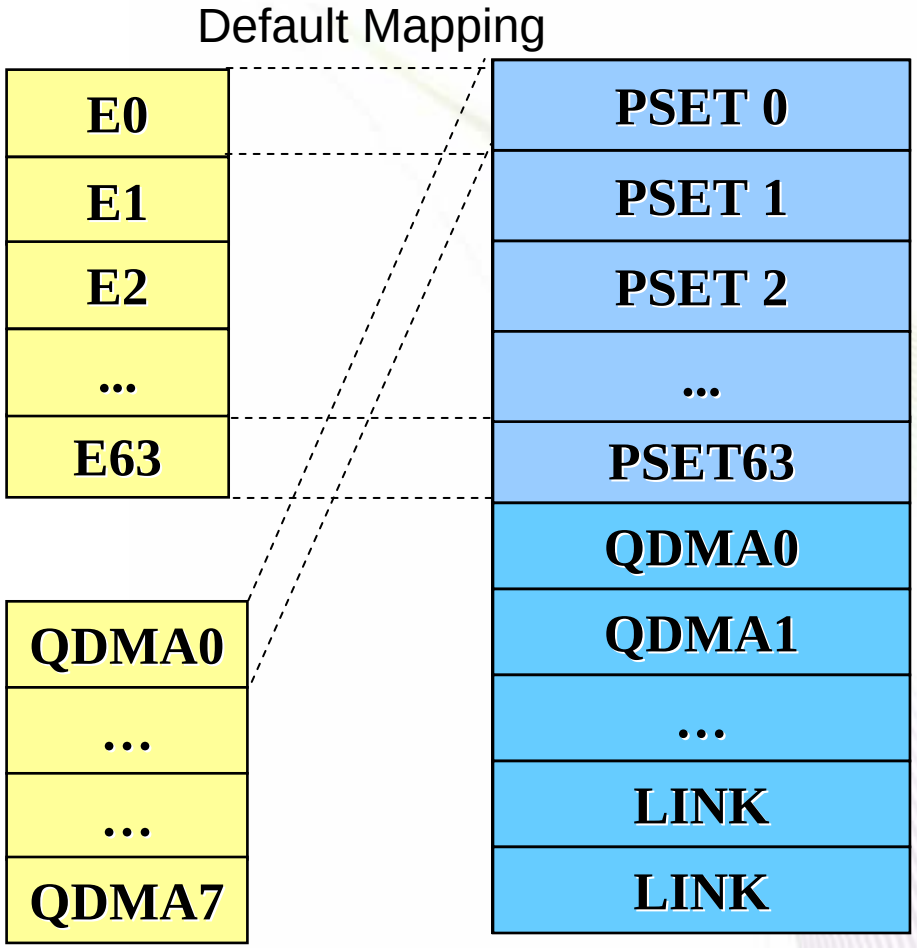
SPRU966a Figure 2-3. EDMA3 Transfer Controller (EDMA3TC) Block Diagram



Channel Controller Architecture

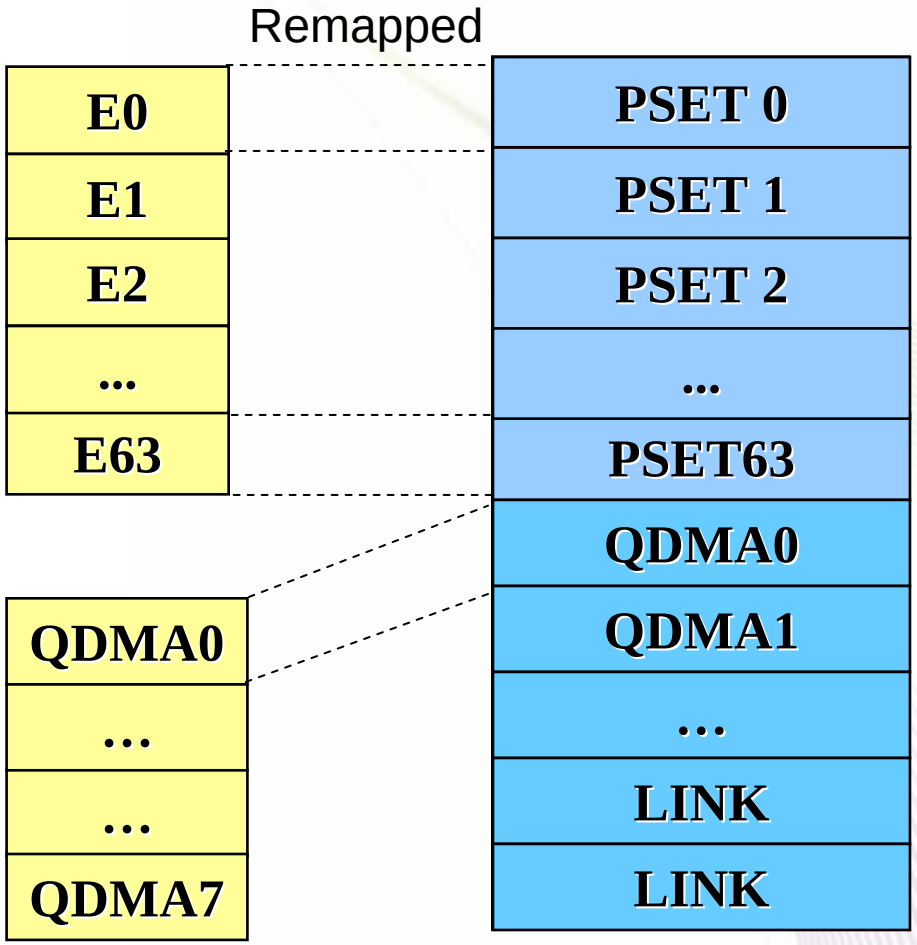


Event Mapping



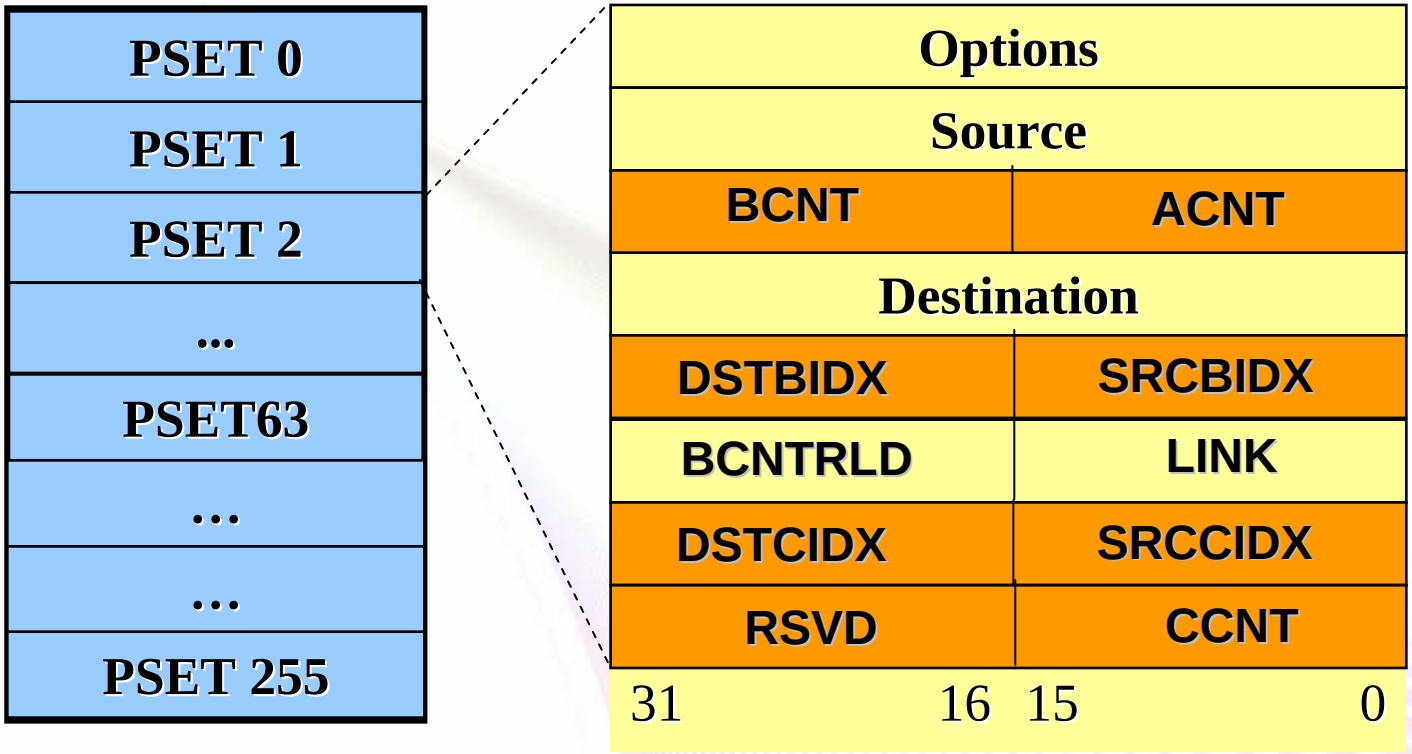
- Default mapping:
  - E0 – PSET0
  - QDMA0 – PSET0
- Mapping of events to PSET controlled by registers.

Event Mapping



- Default mapping:
  - E0 – PSET0
  - QDMA0 – PSET0
- Mapping of events to PSET controlled by registers.

Parameter RAM Sets



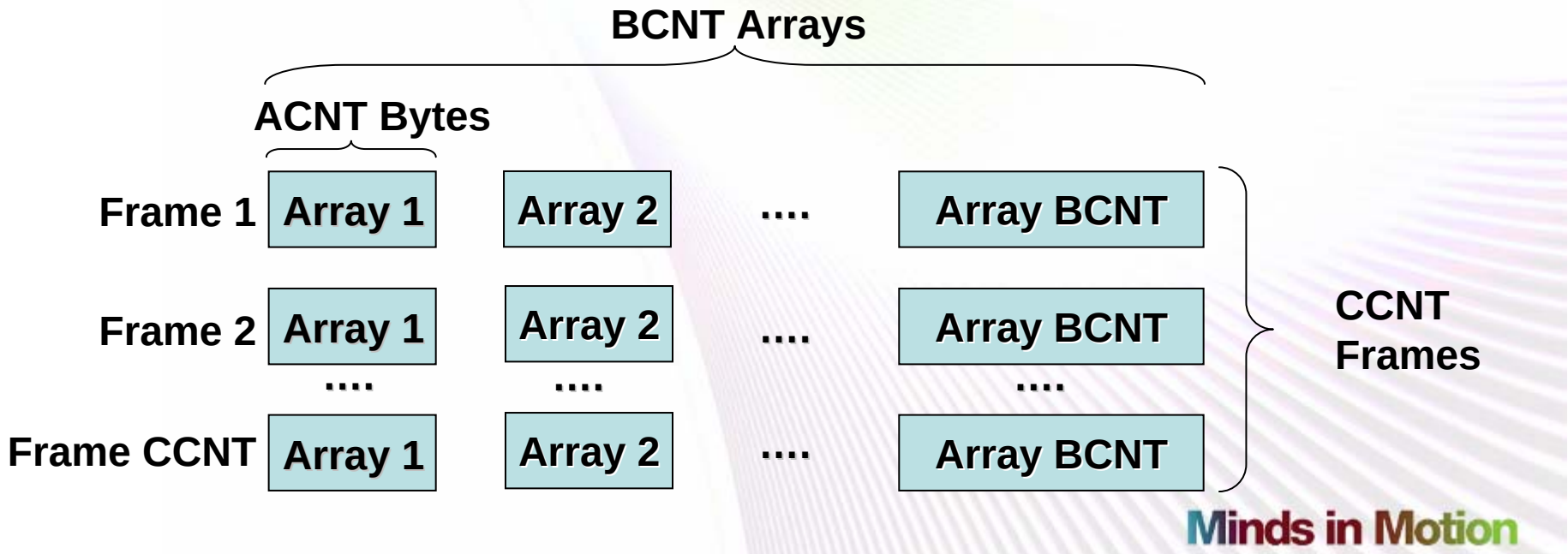
- Up to 512 PSETs available (dependent on device, 256 in C6455).
- Each PSET can be used for DMA, QDMA or link sets.

Parameter RAM Entries

- Options
- Source and Destination addresses
- ACNT/BCNT/CCNT
  - Number of bytes to transfer for each of the three dimensions.
- SRCBIDX/DSTBIDX
  - Number of bytes from start of one array to start of the next array, the second dimension.
- BCNTRLD
  - Parameter entry used to reload BCNT during three-dimensional transfers.
- SRCCIDX/DSTCIDX
  - Number of bytes from start of one frame to start of the next frame, the third dimension.
- LINK
  - Pointer to another PSET to be copied when this one is exhausted.

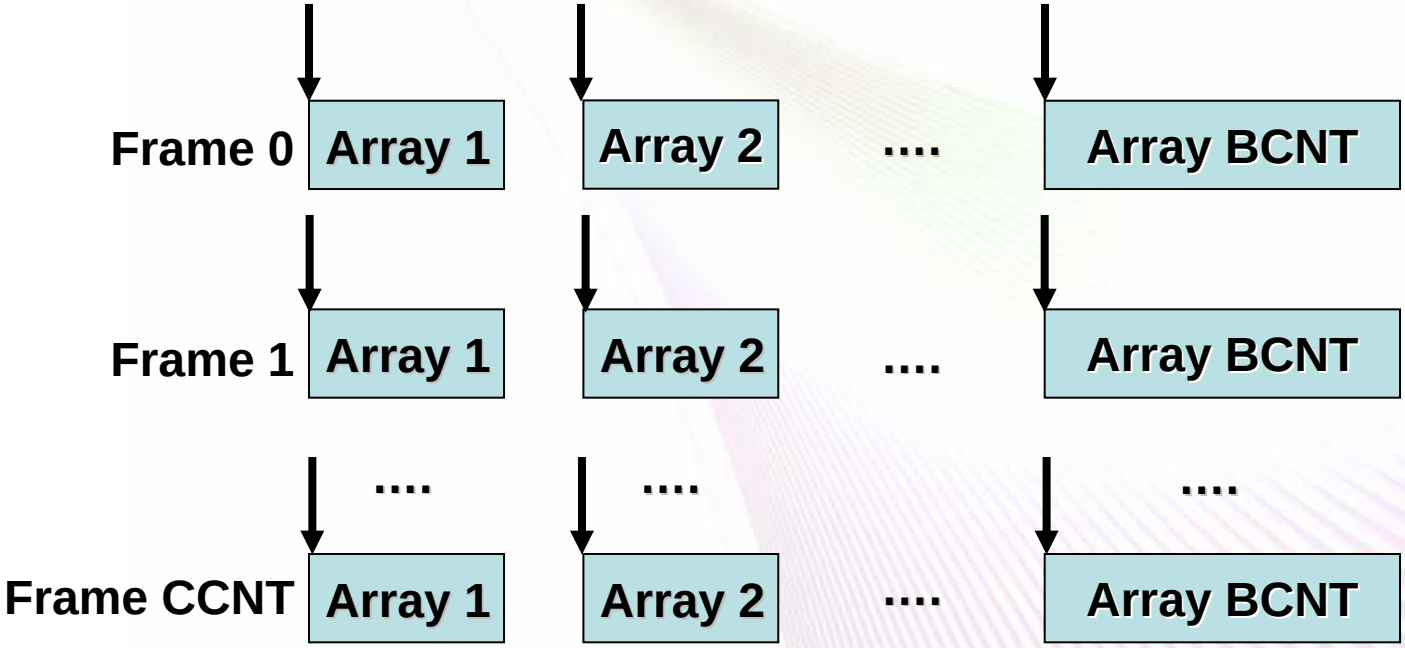
EDMA Terminology

- Truly orthogonal transfer defined in three dimensions.
- Array 1st dimension that consists of contiguous ACNT bytes.
- Frame 2nd dimension of BCNT arrays of ACNT bytes.
- Block 3rd dimension of CCNT frames of BCNT arrays of ACNT bytes
- Minimum transfer is one frame of one array of ACNT bytes.



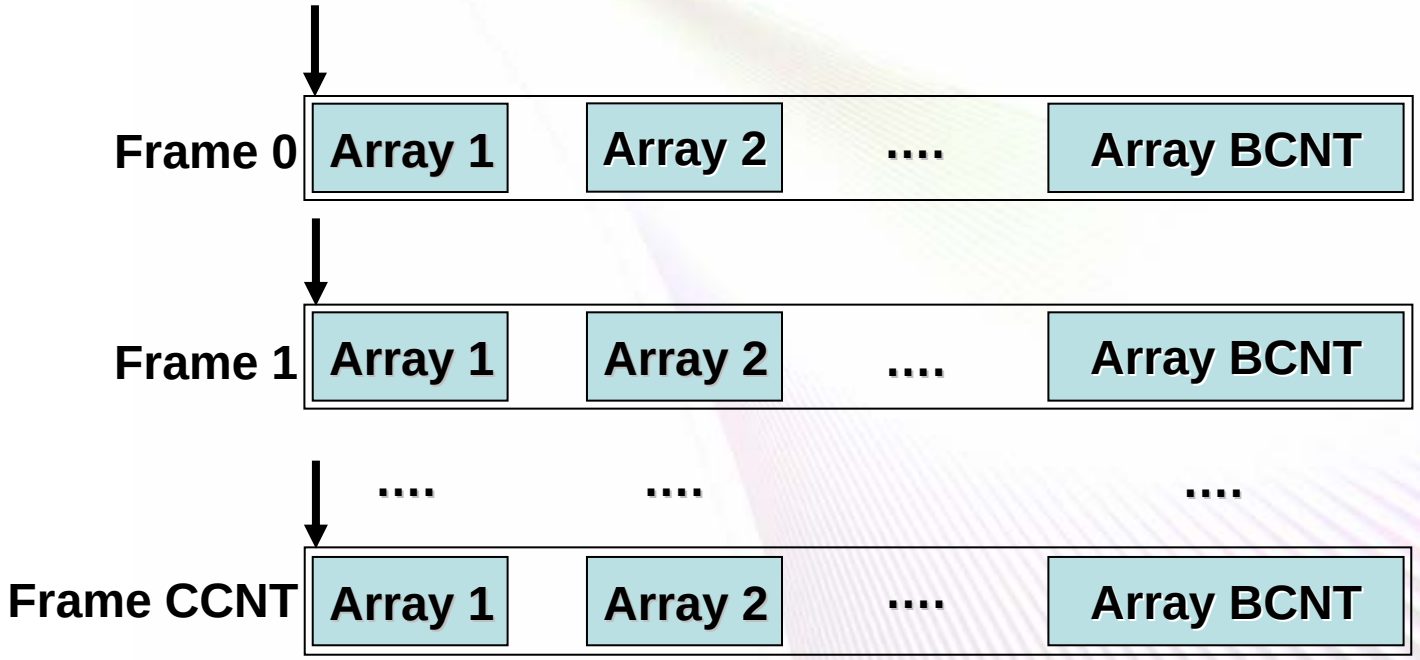
“A” Synchronized Transfer

“A” synchronized transfers: Each event triggers the transfer of exactly one array of ACNT bytes.



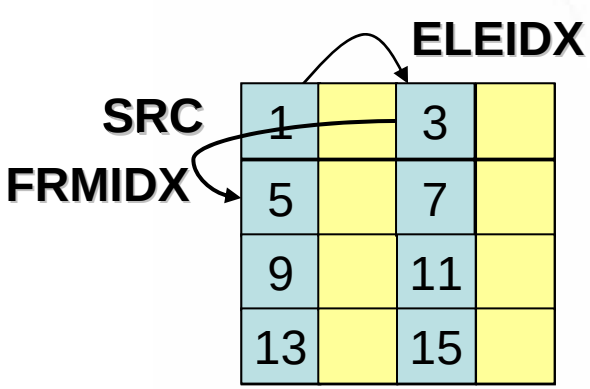
“AB” Synchronized Transfer

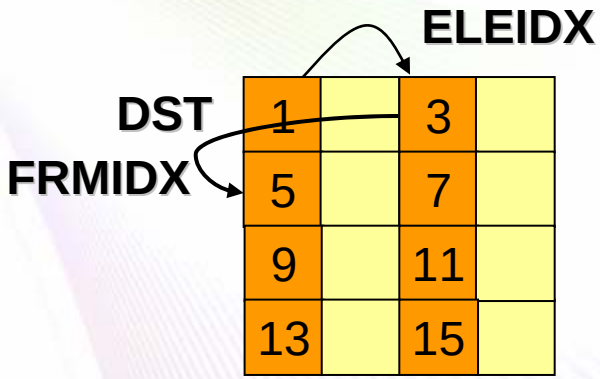
“AB” synchronized transfer: Each event triggers a two-dimensional transfer of BCNT arrays of ACNT bytes.



Indexed Transfers

EDMA2 used a single set of element index (ELEIDX) and frame index (FRMIDX) for both source and destination.

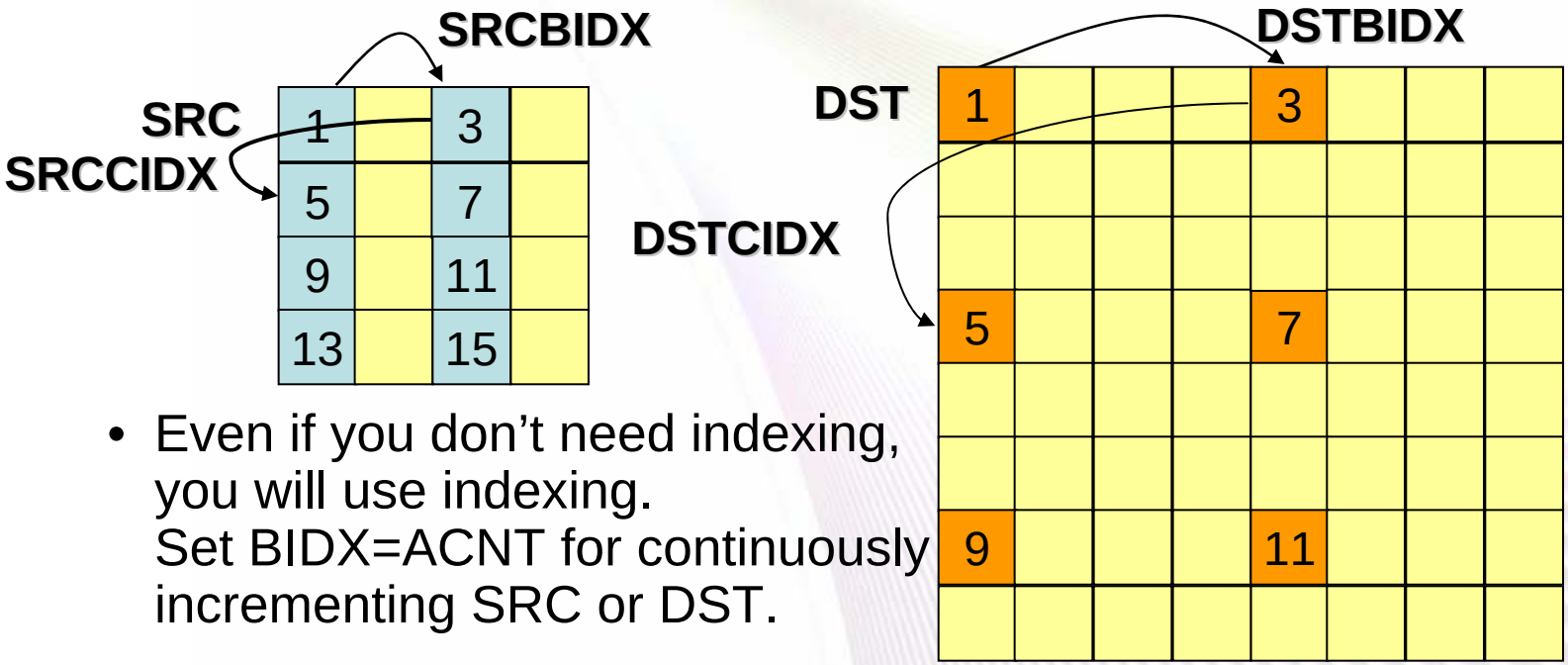




Indexed Transfers

EDMA3 is more flexible, and indexing is always “on.”

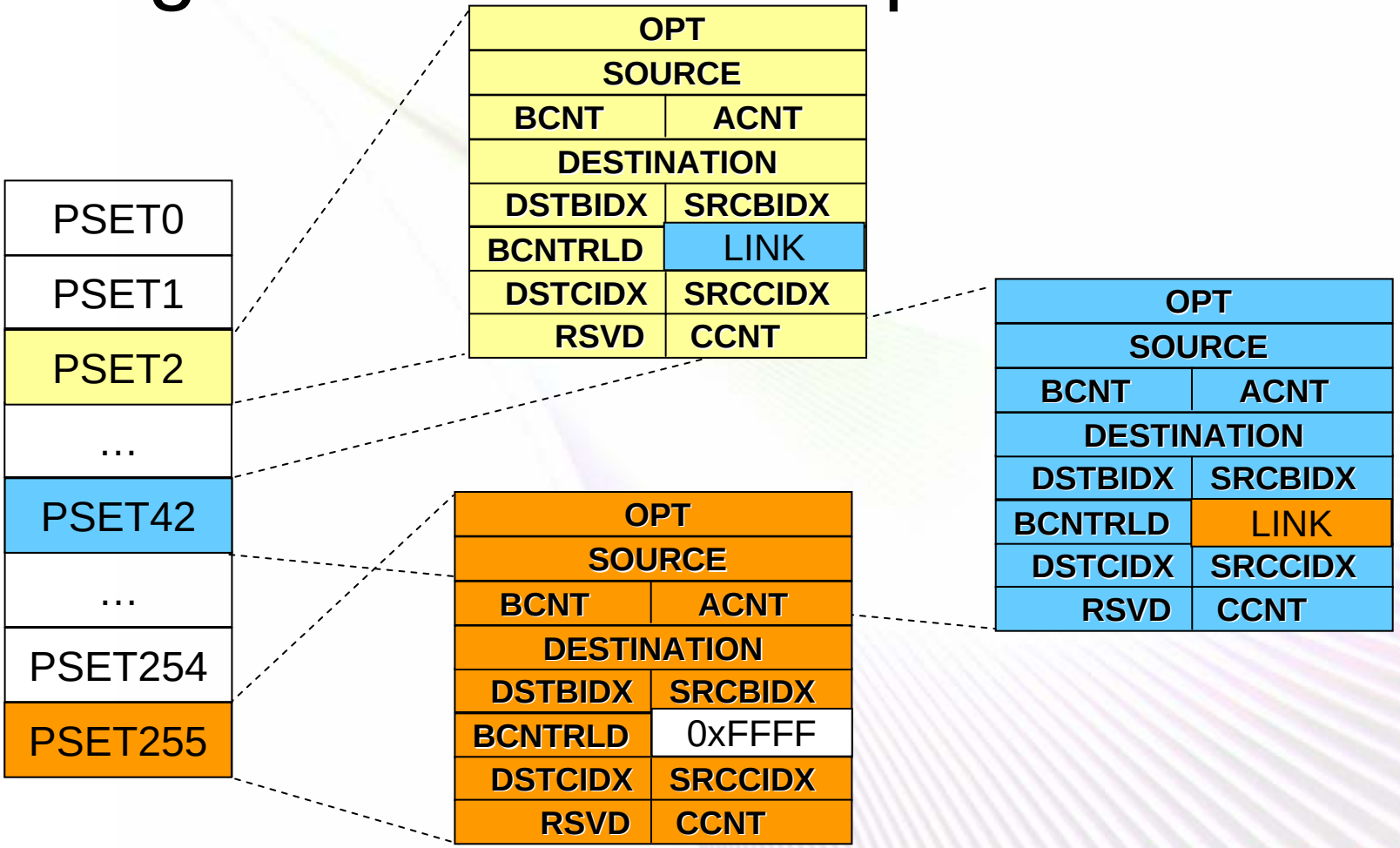
- Four index fields for source and destination indexing in B and C dimensions



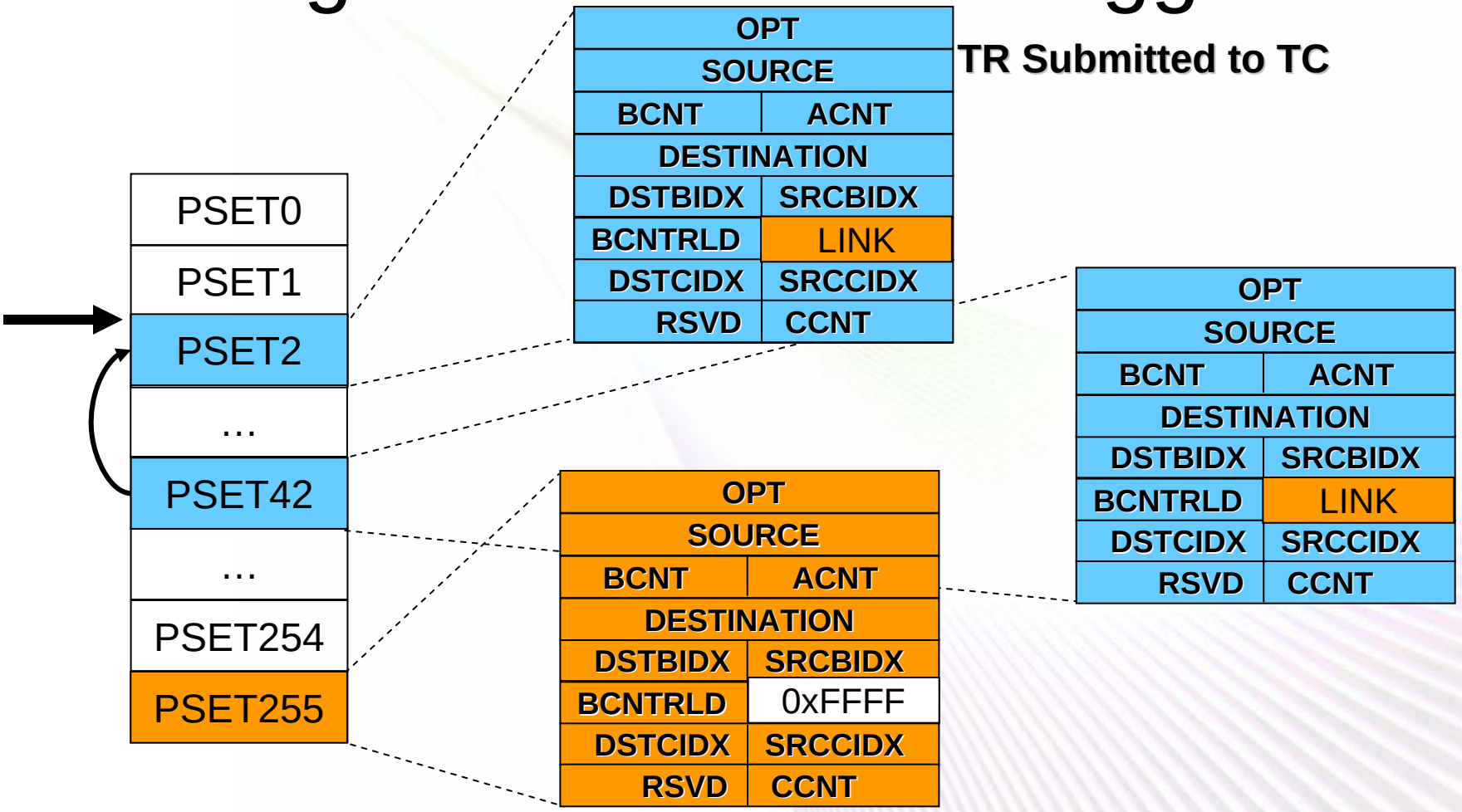
- Even if you don’t need indexing, you will use indexing.
  Set BIDX=ACNT for continuously incrementing SRC or DST.

Minds in Motion

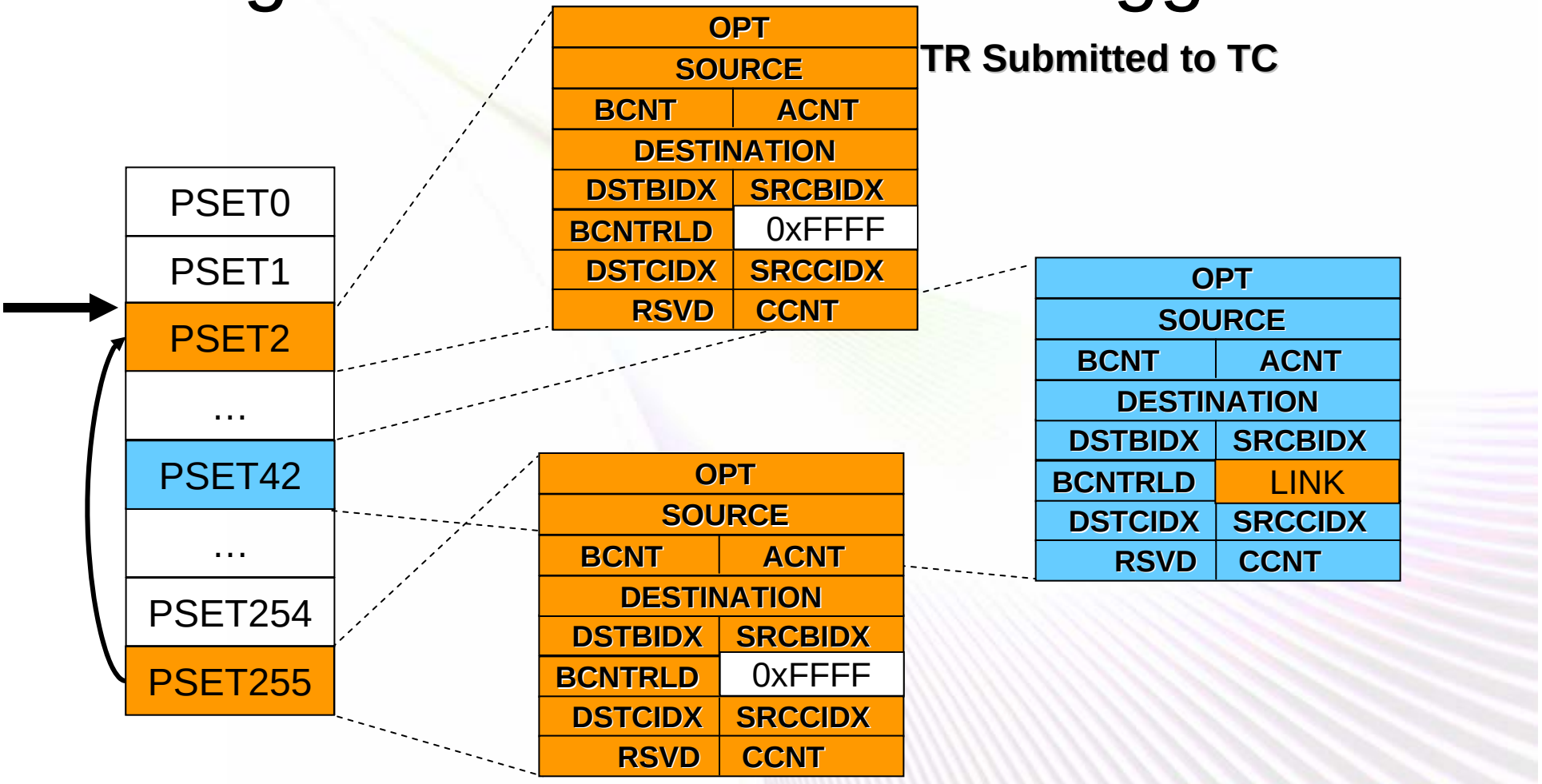
Linking Transfers: Set Up Three PSETs



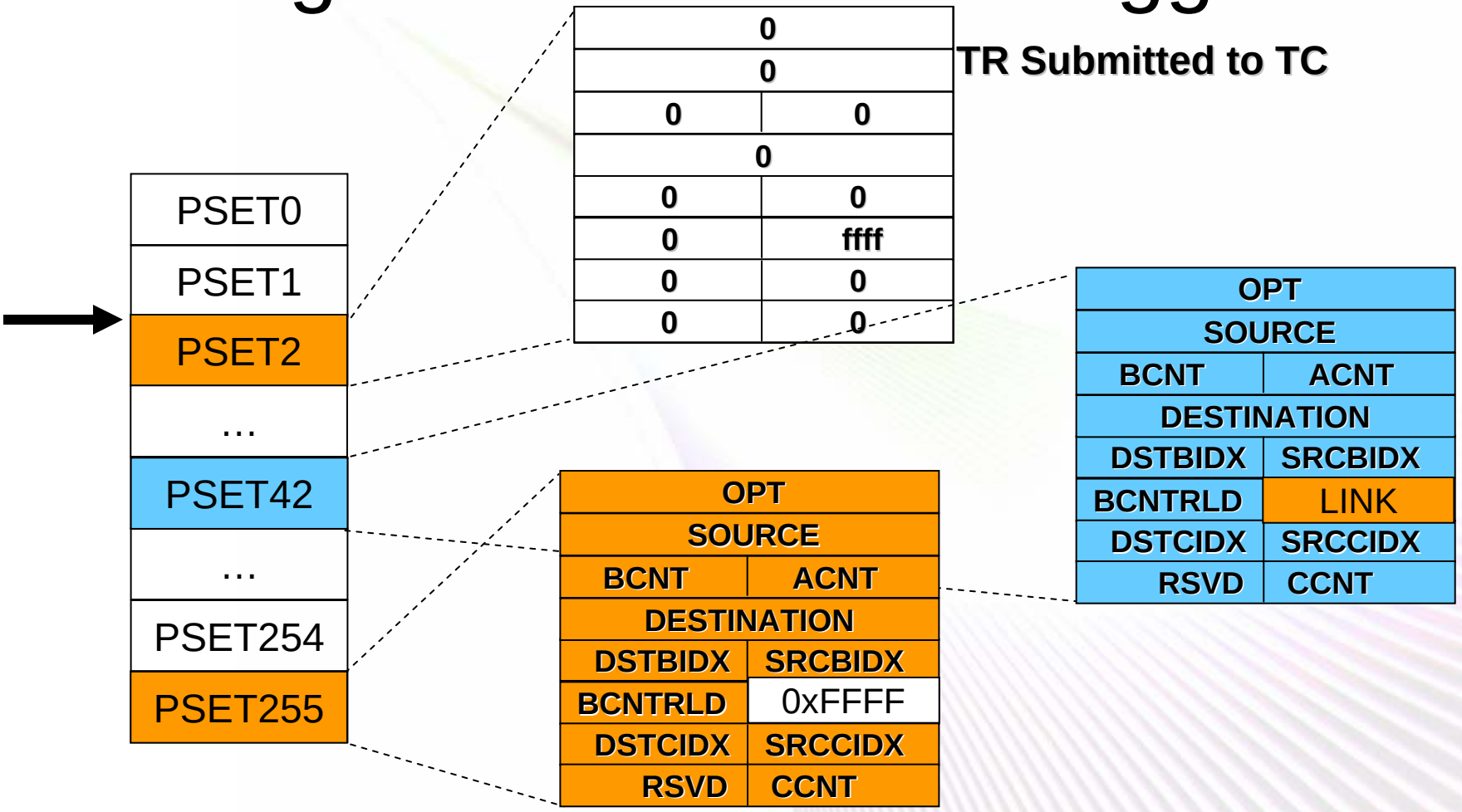
Linking Transfers: First Trigger Event



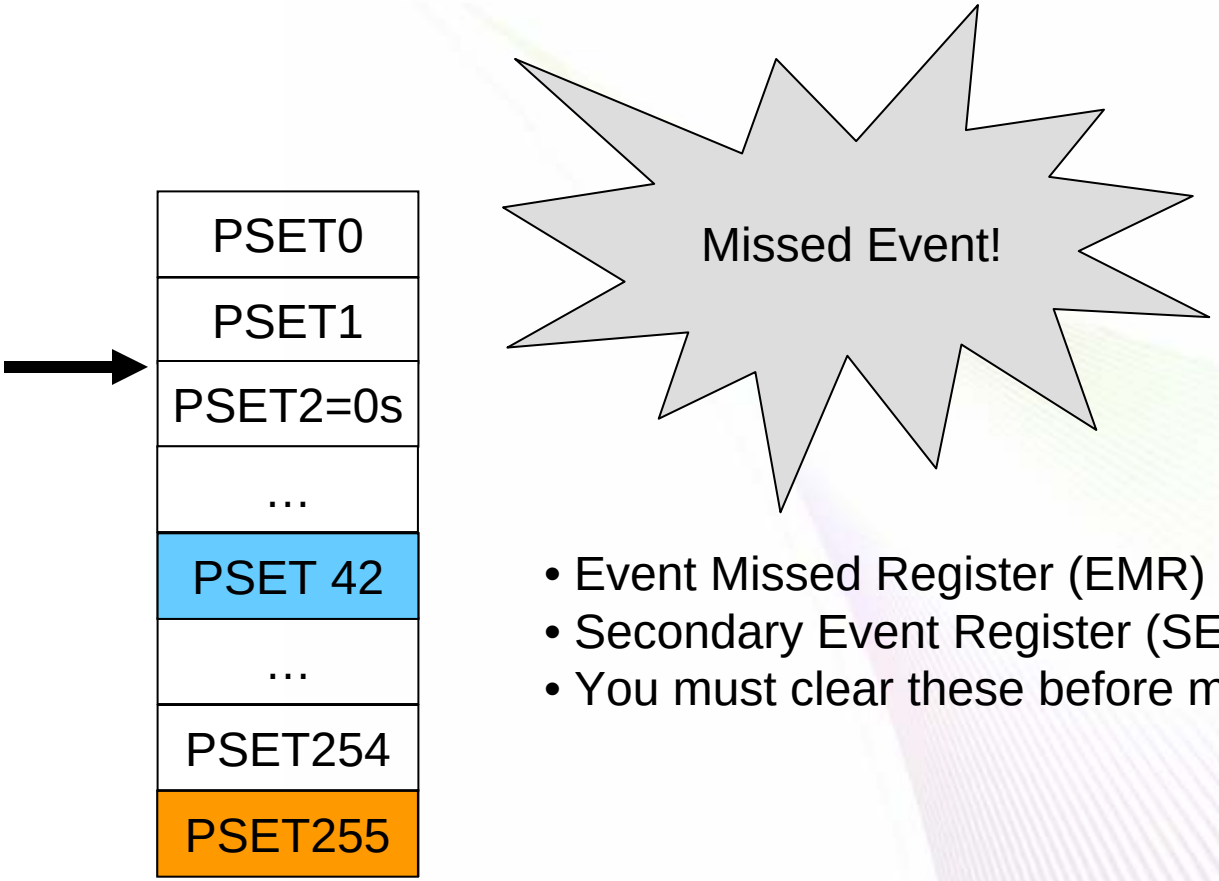
Linking Transfers: Second Trigger Event



Linking Transfers: Third Trigger Event

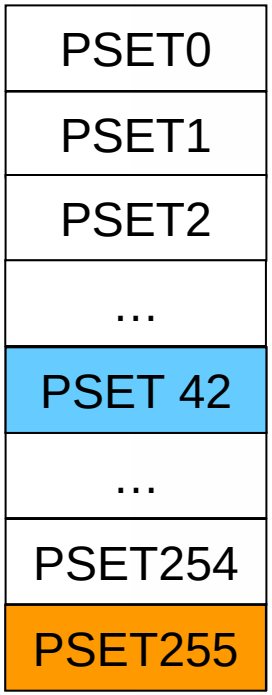


Linking Transfers: If Another Event?



- Event Missed Register (EMR) set.
- Secondary Event Register (SER) remains set.
- You must clear these before more events on this channel.

Linking Transfers



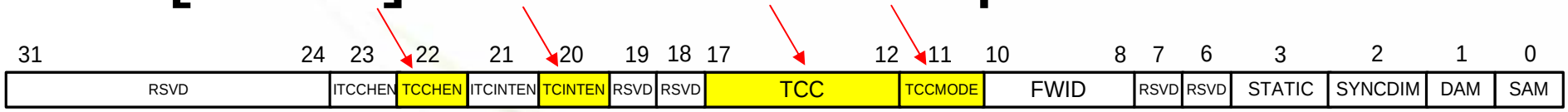
- Linking updates event parameters to re-initialize the PSET.
- Linking is always enabled in EDMA3. All transfers must link to either:
  - Valid link entry
  - NULL entry
- LINK entry points to PSET from which to load the next configuration.
- If linking is not desired, LINK field needs to be set to 0xFFFF, which causes CC to automatically generate a NULL entry. (User does not need to create a NULL entry in the PSET.) All fields are 0x0000 (LINK remains 0xFFFF).
- Link update occurs when event is extracted from event queue and submitted as TR packet to TC and the parameter configuration is exhausted.

Transfer Completion Code (TCC)

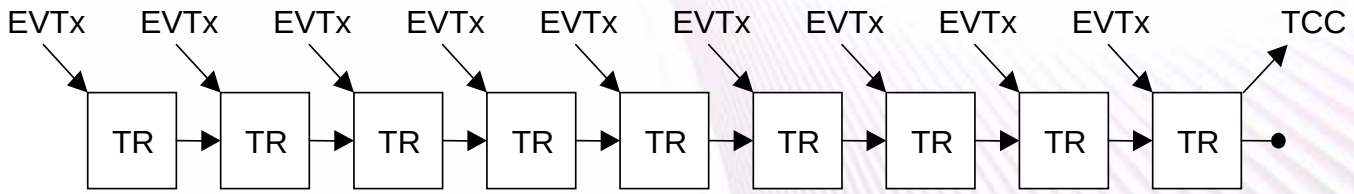
The transfer completion code (TCC) provides a signaling mechanism to generate an interrupt to the CPU and/or trigger another channel (chaining) when a transfer is completed.

- Several fields in the OPT register define how and when a TCC is generated.
- Generated by the Channel Controller when a TR is submitted (early mode) or by the Transfer Controller when a transfer is completed (normal mode).

[Final] Transfer Complete Code

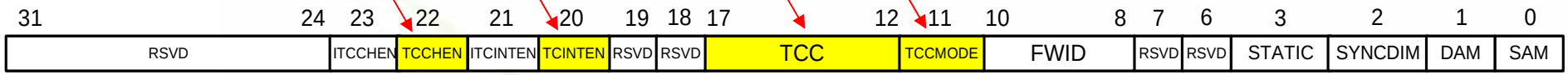


- Transfer complete code (TCC)
  - Indicates that the whole transfer has completed.
  - Can be sent after submitting the last TR to TC (Early mode) or peripheral acknowledgement (Normal mode).
  - CER is set if selected by TCCHEN for chaining.
  - IPR is set if selected by TCINTEN for interrupting.

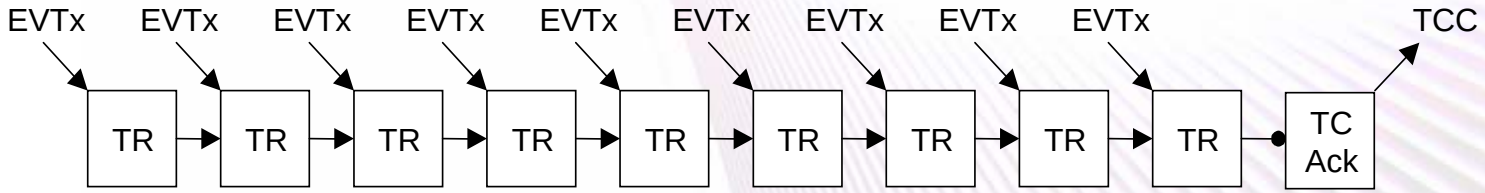


Early Completion mode (TCC from Channel Controller)

[Final] Transfer Complete Code

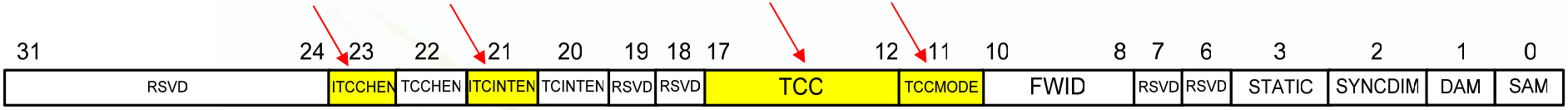


- Transfer complete code (TCC)
  - Indicates that the whole transfer has completed.
  - Can be sent after submitting the last TR to TC (Early mode) or peripheral acknowledgement (Normal mode).
  - CER is set if selected by TCCHEN for chaining.
  - IPR is set if selected by TCINTEN for interrupting.

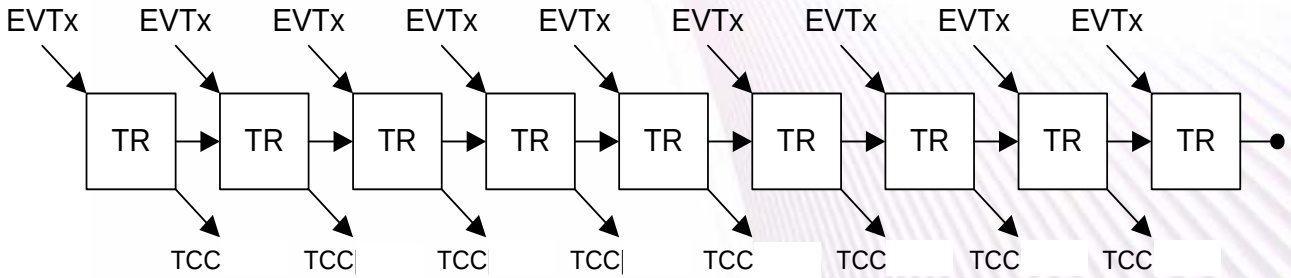


Normal Completion mode (TCC from Transfer Controller)

Intermediate TCC

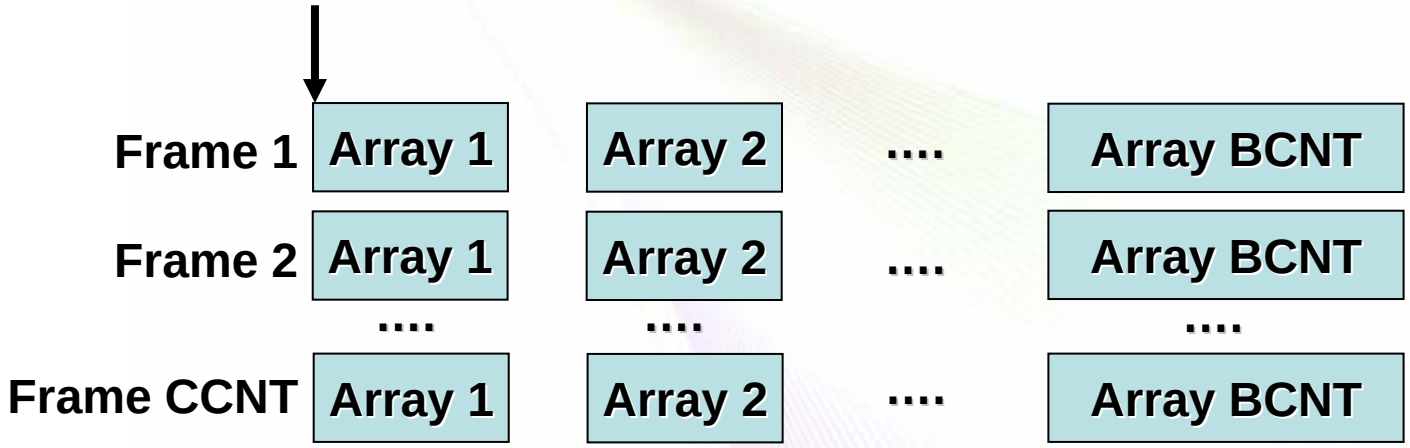


- Intermediate transfer complete code (ITCC)
  - Indicates a TR has been submitted (except the last).
  - Can be sent after submitting all but the last TR to TC (Early mode) or all but the last peripheral acknowledgement through TC (Normal mode).
  - CER is set if selected by ITCCHEN for chaining.
  - IPR is set if selected by ITCINTEN for interrupting.



"ABC" Synchronized Transfer?

- **"ABC" synchronized transfer:** Would like a single event to trigger the whole three-dimensional transfer.

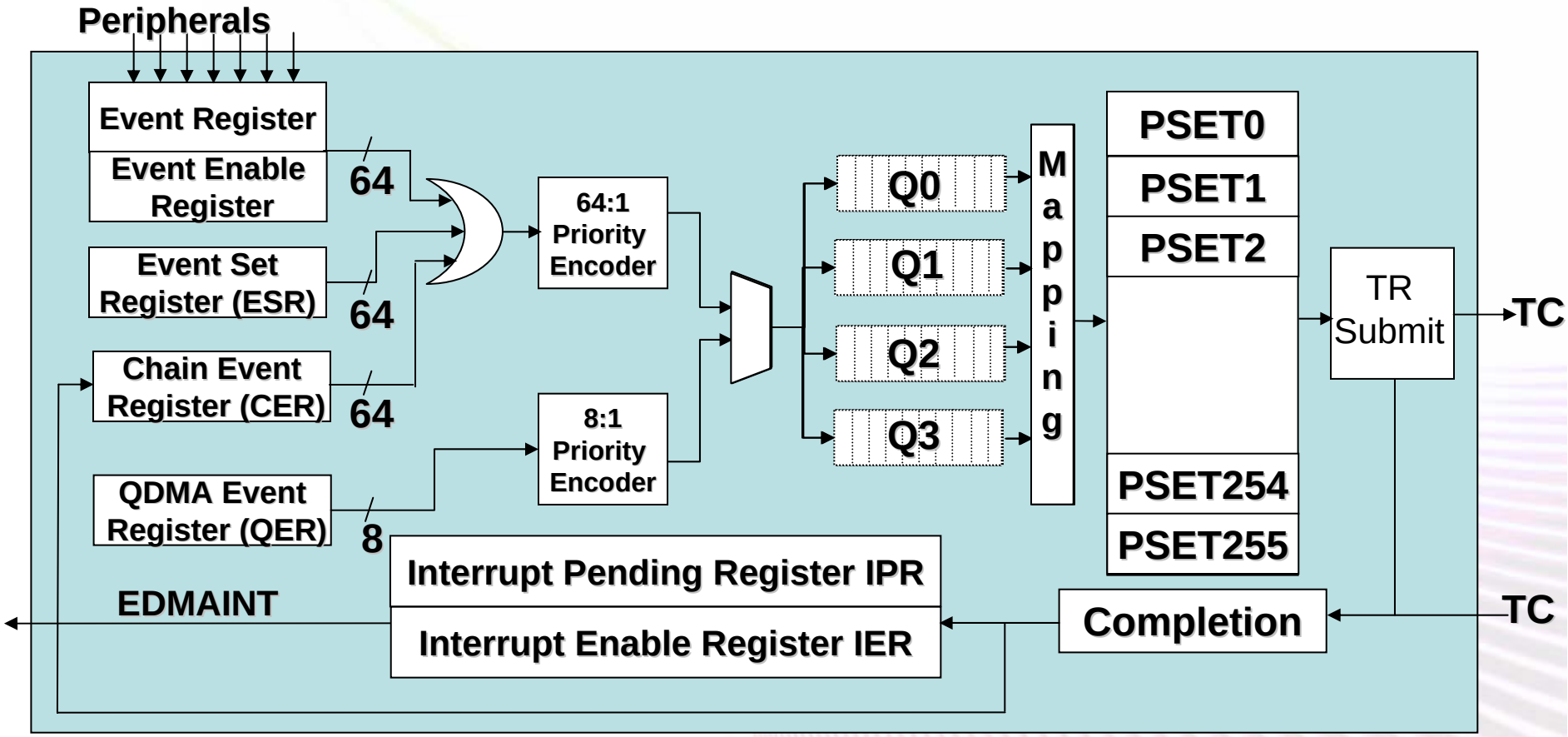


- AB-sync + chaining to itself will work.
- At the end, link to something new and chain to it, too!

DMA, QDMA, IDMA

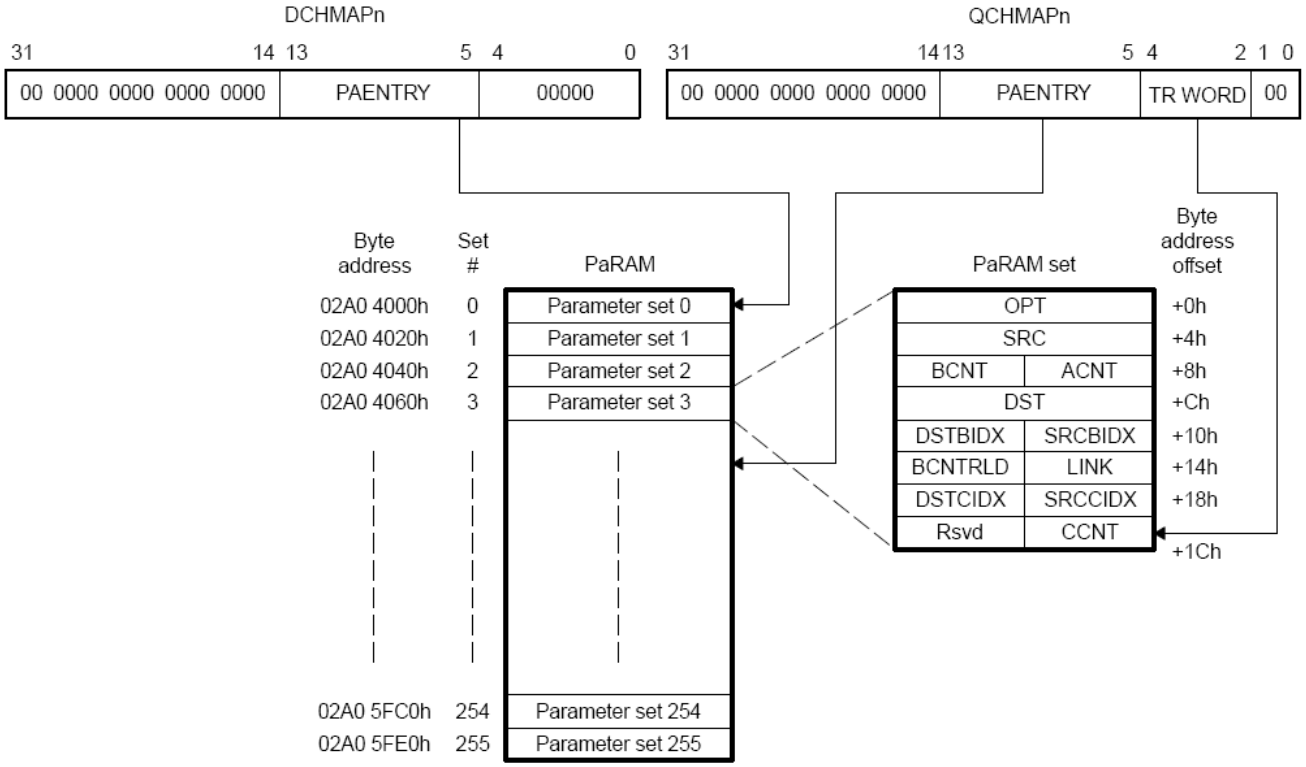
- DMA
  - Usually runs in background (pgm not waiting).
  - Peripheral/event driven.
  - Chaining and linking relieve CPU loading.
- QDMA
  - Can run in background (pgm not waiting).
  - Often runs in foreground (pgm waiting).
  - CSL DAT module uses QDMA for copy and fill operations.
  - Four QDMA channels on C6455.
- IDMA
  - Specialized for “internal” transfers.
  - IDMA0: L1/L2<->Config Bus
    - Source and destination must be aligned to a 32-byte boundary.
    - Note: SPRU871d mistakenly says 32-word alignment.
  - IDMA1: L2<->L1

QDMA Path Through CC



QDMA Channel Mapping

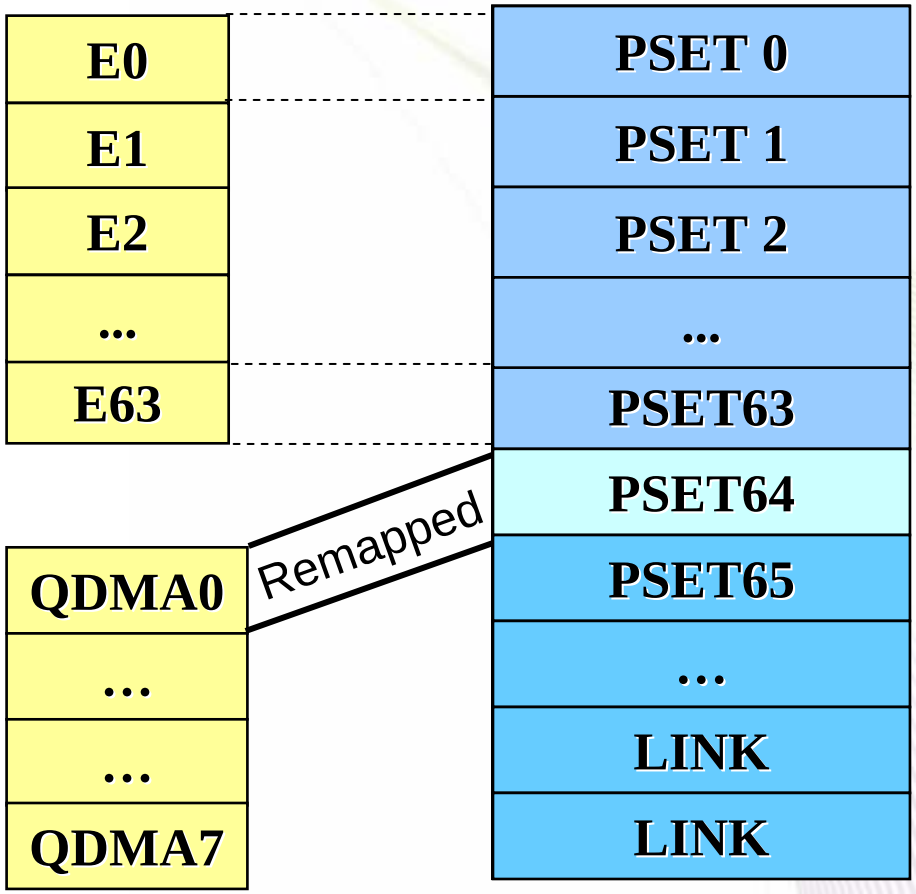
SPRU871d Figure 2-10. DMA Channel and QDMA Channel to PaRAM Mapping



QDMA Channel Trigger Types

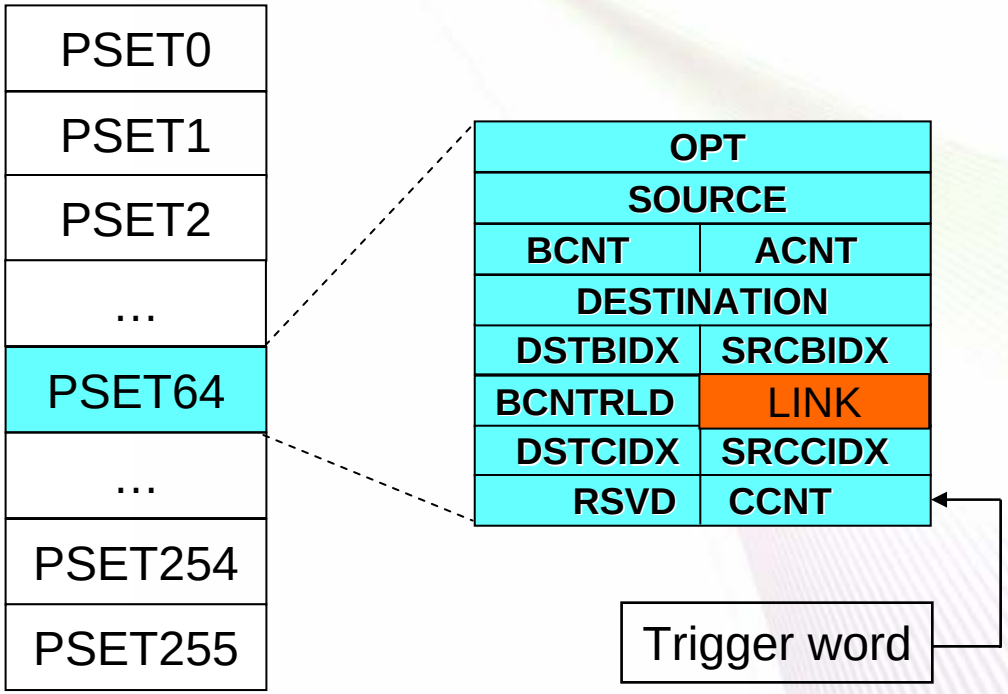
- Auto-Triggering
  - QDMA Channels allow for CPU to initiate DMA transfers with a minimum number of writes.
  - QDMA Channel is “auto-triggered” when CPU writes to the “trigger” word of specified QDMA Channel Entry.
  - Eliminates need to write to PaRAM entry and kick off transfer with separate write to ESR (Event Set Register).
  - Trigger Word is user-programmed for each QDMA Channel to any location in PaRAM address space (QCHMAP register).
  - Selection of Trigger Word allows CPU to modify only words of interest in a PaRAM Set. For example, if CNT is typically static for a given algorithm, but SRC is different from transfer to transfer, then SRC could be defined as the trigger word. CPU can initiate a DMA transfer with as few as one write to SRC for the specified PaRAM Set.
  - Assumes OPT.STATIC = 1. Count and address updates or linking not performed. Otherwise it could trigger unexpected transfer requests.
- LINK Triggering
  - If OPT.STATIC=0, copying the LINK PSET will write to the trigger word.
  - Should choose Trigger Word=7 (CCNT) so everything is copied before triggering.
- Optimized for C64x IDMA
  - C64x+ IDMA transfers to a linear range of memory with a mask to enable/disable transfers to specific words within the linear range.
  - IDMA can stride across multiple QDMA Channels to perform several QDMA submissions.

QDMA Auto-Trigger: Set Channel Mapping



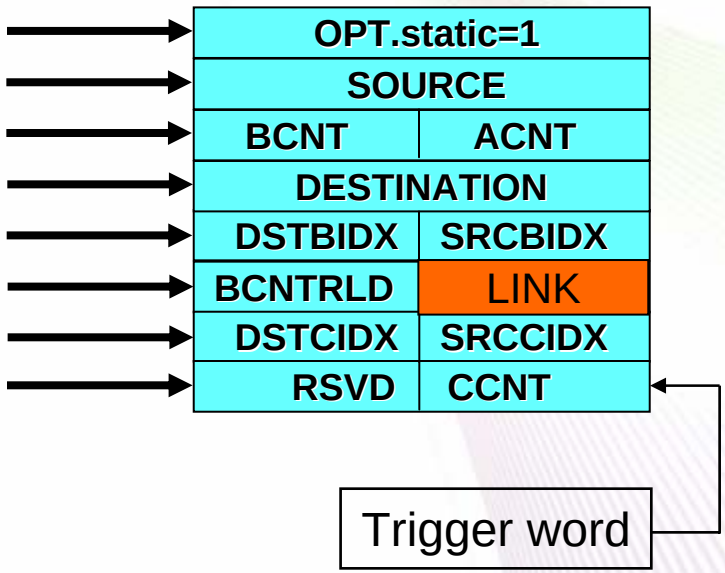
- QDMA channels remapped
  - QDMA0 => PSET64

QDMA Auto-Trigger: Select Trigger Word



QDMA Auto-Trigger: Write 8 Words

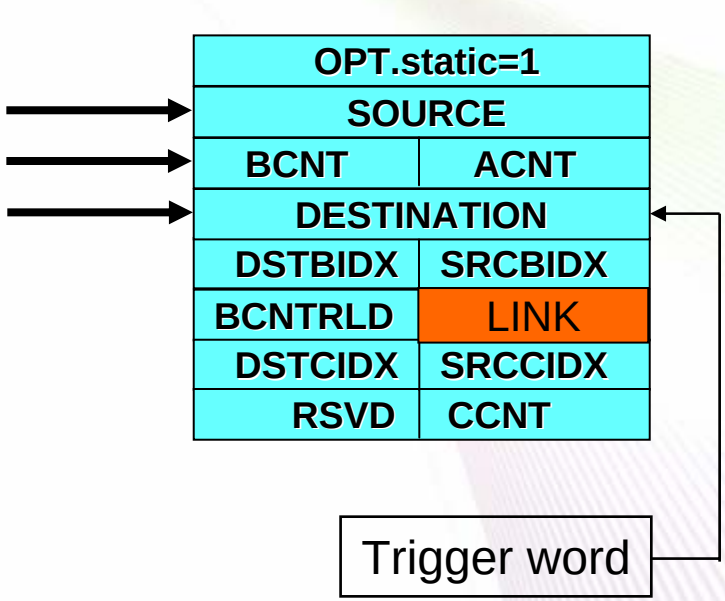
CPU writes:



Request to CC queue
then
TR submitted to TC

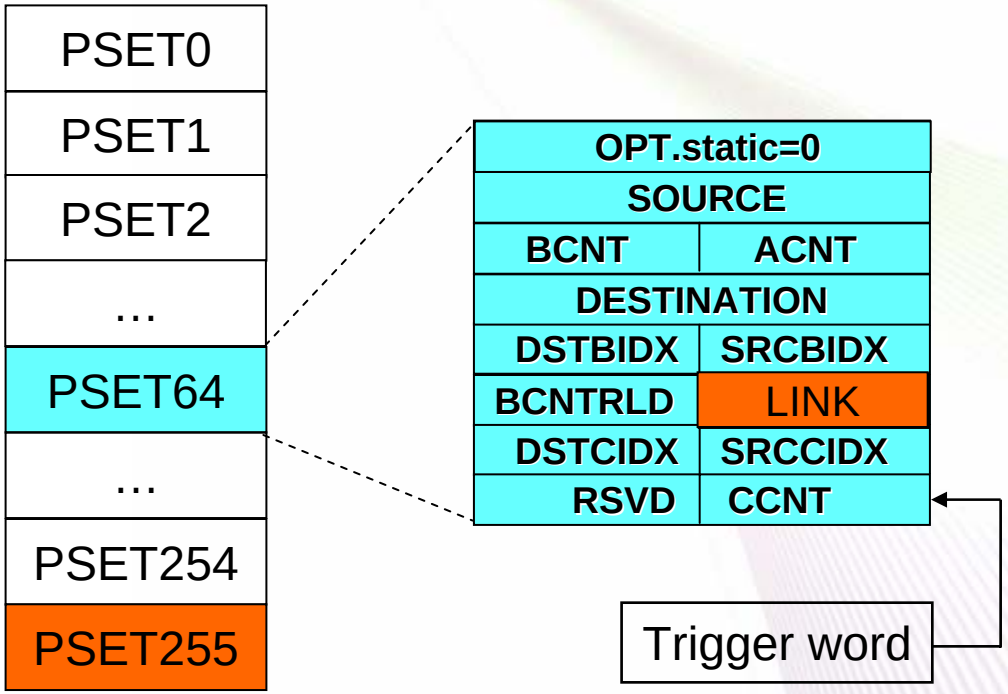
QDMA Auto-Trigger: Write 3 Words

CPU writes:



Request to CC queue
then
TR Submitted to TC

QDMA Auto-Trigger: Link Triggering 1



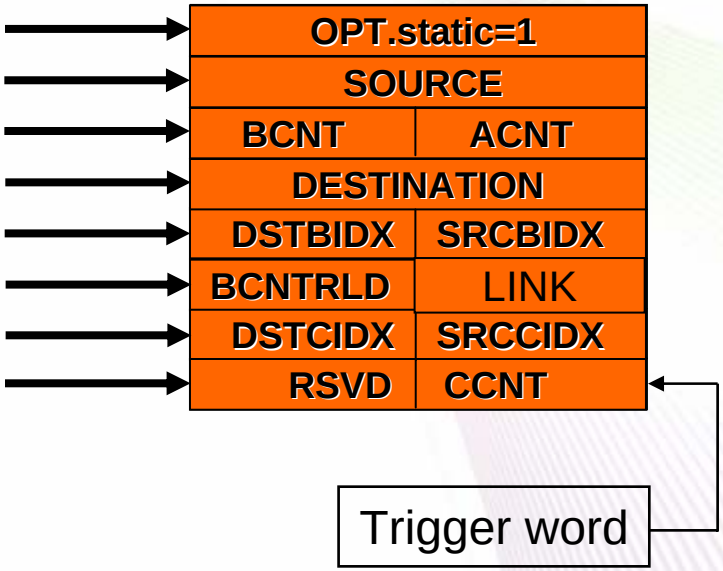
STATIC=0 so link will occur

Link to any filled PSET

Trigger must be 7 because all 8 words will be written

QDMA Auto-Trigger: Link Triggering 2

CPU writes:

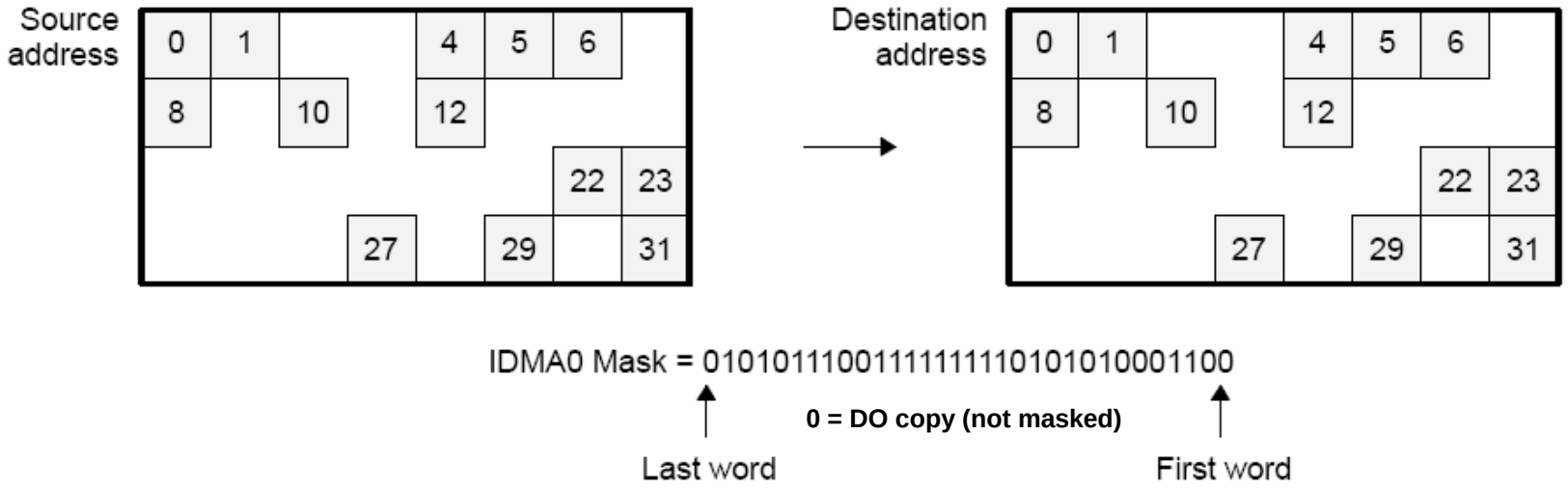


Request to CC queue
…then
TR submitted to TC
…then
new PSET copied to PSET64,
including writing to trigger
…then
Request to CC queue
…then
TR submitted to TC

IDMA0

```
#include <csl_idma.h>
IDMA0_init( IDMA_INT_EN );                    // enable interrupt, or not
IDMA0_configArgs( Mask, Src, Dst, BlockCnt-1 );
IDMA0_wait();                                 // optional
```

SPRU871d Figure 5-1. IDMA Channel 0 Transaction



IDMA1

```
#include <csl_idma.h>

IDMA1_init( IDMA_PRI_7, IDMA_INT_EN ); // set priority, enable interrupt

IDMA1_copy( Src, Dst, ByteCnt );

IDMA1_fill( Dst, ByteCnt, FillWord );

IDMA1_wait();                                   // optional
```

Comparison of ‘C6455 and ‘LP64x Channel Controller Features

Channel Controller Feature	‘C6455	‘LP64 *
Number of DMA Channels	64	64
Number of QDMA Channels	4	8
Number of Interrupt Channels (TCC)	64	64
Number of PaRAM Entries	256	128
Number of Queues	4	3
Number of Transfer Controllers	4	3
Number of Regions	8	2
DMA Channel Mapping	Yes	No
Memory Protection Support	Yes	No

* Preliminary Info – Subject to Change

Comparison of ‘C6455 and ‘LP64x Transfer Controller Features

Transfer Controllers	Configuration	‘C6455	‘LP64 *
TC0	FIFOSIZE	128 bytes	128 bytes
	BUSWIDTH	16 bytes	8 bytes
	DSTREGDEPTH	2 entries	4 entries
	DBS	64 bytes	16 bytes**
TC1	FIFOSIZE	128 bytes	256 bytes
	BUSWIDTH	16 bytes	8 bytes
	DSTREGDEPTH	4 entries	4 entries
	DBS	64 bytes	32 bytes**
TC2	FIFOSIZE	256 bytes	128 bytes
	BUSWIDTH	16 bytes	8 bytes
	DSTREGDEPTH	4 entries	4 entries
	DBS	64 bytes	64 bytes**
TC3	FIFOSIZE	256 bytes	x
	BUSWIDTH	16 bytes	x
	DBS	64 bytes	x

* Preliminary Info – Subject to Change

** DBS is programmable on LP (16, 32 or 64 bytes). Defaults are shown.

Harness the Power of EDMA 3

- Does your DSP have EDMA3?
- Introduction to EDMA3
- Use CSL 3 to program EDMA3
- Tips, tricks, tools, techniques

Use CSL 3 to Program EDMA3

- A simple transfer example using a global channel region
- The changes to use a shadow region
- The changes to cause an interrupt
  - Use EDMA interrupt dispatcher provided with CSL 3
- DAT CSL for QDMA transfers
- QDMA CSL

A Single Transfer Using CSL 3

```
CSL_edma3Init();
CSL_edma3Open(&edmaObj,CSL_EDMA3,,);
CSL_edma3HwSetup(hModule,);
```

Initialize EDMA3

```
CSL_edma3ChannelOpen(&chObj, CSL_EDMA3,,);
CSL_edma3GetHwChannelSetupParam(hChannel,);
CSL_edma3GetParamHandle(hChannel,,);
CSL_edma3ParamSetup(hParam,);
CSL_edma3HwChannelControl(hChannel,,);
```

Prepare channel

```
CSL_edma3HwChannelControl(hChannel,,);
CSL_edma3GetHwStatus(hModule,,);
CSL_edma3HwControl(hModule,,);
```

Run

```
CSL_edma3ChannelClose(hChannel);
CSL_edma3Close(hModule);
```

Close, if needed

A Single Transfer: Initialize EDMA3

```
#include <csl_edma3.h>

CSL_Edma3Context        context;
CSL_Status              status;
CSL_Edma3Handle         hModule;
CSL_Edma3Obj            edmaObj;

// Init is a CSL placeholder function for consistency.
status = CSL_edma3Init(&context);

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

A Single Transfer: Initialize EDMA3

```
CSL_Edma3HwSetup           hwSetup;
CSL_Edma3HwDmaChannelSetup dmahwSetup [CSL_EDMA3_NUM_DMACH]
                                  = CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Edma3HwQdmaChannelSetup qdmahwSetup[CSL_EDMA3_NUM_QDMACH]
                                  = CSL_EDMA3_QDMACHANNELSETUP_DEFAULT;

dmahwSetup[CSL_EDMA3_CHA_DSP_EVT].paramNum
                                                  = CSL_EDMA3_CHA_DSP_EVT;
dmahwSetup[CSL_EDMA3_CHA_DSP_EVT].que
                                                  = CSL_EDMA3_QUE_0;
hwSetup.dmaChaSetup   = dmahwSetup;
hwSetup.qdmaChaSetup = qdmahwSetup;

// HwSetup maps DMA/QDMA channels to CC PaRAM sets and to CC queues.
status = CSL_edma3HwSetup(hModule,&hwSetup);
```

A Single Transfer: Prepare Channel

```
CSL_Edma3ChannelObj     chObj;
CSL_Edma3ChannelAttr    chAttr;

// for global region, just open the channel and continue:
chAttr.regionNum = CSL_EDMA3_REGION_GLOBAL;
chAttr.chaNum = CSL_EDMA3_CHA_DSP_EVT;
hChannel = CSL_edma3ChannelOpen(&chObj, CSL_EDMA3, &chAttr, &status);
```

A Single Transfer: Prepare Channel

```
CSL_Edma3ChannelObj          chObj;
CSL_Edma3ChannelAttr         chAttr;

// for shadow region, enable shadow region access, then open the channel.
// set bits in DRAEn/DRAEHn (OR function),  use _DISABLE to clear bits.
regionAccess.region = CSL_EDMA3_REGION_5 ;
regionAccess.drae =   (1 << CSL_EDMA3_CHA_DSP_EVT) // must have access
                     |(1 << CSL_EDMA3_CHA_8);          // to channel and tcc
regionAccess.draeh = 0;                                // if different
status = CSL_edma3HwControl(hModule,
                  CSL_EDMA3_CMD_DMAREGION_ENABLE, &regionAccess);

/* Open the channel in context of shadow region 5 */
chAttr.regionNum = CSL_EDMA3_REGION_5;
chAttr.chaNum = CSL_EDMA3_CHA_DSP_EVT;
hChannel = CSL_edma3ChannelOpen(&chObj, CSL_EDMA3, &chAttr, &status);
```

A Single Transfer: Prepare Channel

```
CSL_Edma3ParamSetup        myParamSetup;

myParamSetup.option = CSL_EDMA3_OPT_MAKE(
                                CSL_EDMA3_ITCCH_DIS,
                                CSL_EDMA3_TCCH_EN,
                                CSL_EDMA3_ITCINT_DIS,
                                CSL_EDMA3_TCINT_EN,
                                CSL_EDMA3_CHA_8,              // TCC
                                CSL_EDMA3_TCC_NORMAL,
                                CSL_EDMA3_FIFOWIDTH_NONE,
                                CSL_EDMA3_STATIC_DIS,
                                CSL_EDMA3_SYNC_AB,
                                CSL_EDMA3_ADDRMODE_INCR,
                                CSL_EDMA3_ADDRMODE_INCR
                        );
```

A Single Transfer: Prepare Channel

```
myParamSetup.srcAddr = (Uint32)srcBuf;

// note the order of the parameters is like the name: (ACNT, BCNT)
//    not like the PaRAM word: BCNT : ACNT
myParamSetup.aCntbCnt = CSL_EDMA3_CNT_MAKE(512,1);

myParamSetup.dstAddr = (Uint32)dstBuf;

// B/CIDX don't matter in this case since BCNT = 1 and CCNT = 1
myParamSetup.srcDstBidx = CSL_EDMA3_BIDX_MAKE(0,0);
myParamSetup.linkBcntrld =
            CSL_EDMA3_LINKBCNTRLD_MAKE(CSL_EDMA3_LINK_NULL,0);
myParamSetup.srcDstCidx = CSL_EDMA3_CIDX_MAKE(0,0);
myParamSetup.cCnt = 1;
```

A Single Transfer: Prepare Channel

```
myParamSetup.srcAddr = (Uint32)srcBuf;

// note the order of the parameters is like the name: (ACNT, BCNT)
//    not like the PaRAM word: BCNT : ACNT
myParamSetup.aCntbCnt = CSL_EDMA3_CNT_MAKE(512,2);

myParamSetup.dstAddr = (Uint32)dstBuf;

// BIDX does matter since BCNT > 1
myParamSetup.srcDstBidx = CSL_EDMA3_BIDX_MAKE(512, 512);
myParamSetup.linkBcntrld =
            CSL_EDMA3_LINKBCNTRLD_MAKE(CSL_EDMA3_LINK_NULL,0);
myParamSetup.srcDstCidx = CSL_EDMA3_CIDX_MAKE(0,0);
myParamSetup.cCnt = 1;
```

A Single Transfer: Prepare Channel

```
// write the PaRAM setup values to PaRAM.
status = CSL_edma3ParamSetup(hParamBasic, &myParamSetup);

// enable this channel by writing to EER via EESR.
status = CSL_edma3HwChannelControl(hChannel,
                                   CSL_EDMA3_CMD_CHANNEL_ENABLE, NULL);
```

A Single Transfer: Run

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

do {  /* Poll for IPR bit set */

  CSL_edma3GetHwStatus(hModule,
                       CSL_EDMA3_QUERY_INTRPEND,&regionIntr);
  regionIntr.intr &= 1 << CSL_EDMA3_CHA_8;      // avoids clearing other IPR bits
} while ( regionIntr.intr == 0 );

regionIntr.intrh = 0;  // avoids clearing other IPR bits
status = CSL_edma3HwControl(hModule,
                       CSL_EDMA3_CMD_INTRPEND_CLEAR, &regionIntr);
```

A Single Transfer: Close

```
// if you are done with a channel, close it.
status = CSL_edma3ChannelClose(hChannel);

// if you are done with the EDMA3, close it.
//    (but who does except for example code?)
status = CSL_edma3Close(hModule);
```

Cause an Interrupt When Done

```
CSL_intcInit(&intcContext);
CSL_intcGlobalNmiEnable();
CSL_intcGlobalEnable(&state);
```

} Initialize INTC

```
CSL_intcOpen (&intcObjEdma, …);
CSL_intcPlugEventHandler(hIntcEdma,…);
CSL_intcHwControl(hIntcEdma, …);
```

} Prepare interrupt path from EDMA

```
EdmaEventHook(tcc, isrFunctionName);
```

} Set up EDMA dispatcher

```
CSL_edma3HwControl(hModule,
        CSL_EDMA3_CMD_INTR_ENABLE, …);
```

} Enable intr from this channel in EDMA

[A good EDMA dispatcher is provided with CSL 3 examples.]

Interrupt: Initialize INTC

```
#include <csl_intc.h>

CSL_IntcContext                 intcContext;
CSL_IntcEventHandlerRecord      EventHandler[10];
CSL_IntcGlobalEnableState       state;

/* Intc module initialization */
intcContext.eventhandlerRecord = EventHandler;
intcContext.numEvtEntries = 10;
CSL_intcInit(&intcContext);

/* Enable NMIs */
CSL_intcGlobalNmiEnable();

/* Enable global interrupts */
CSL_intcGlobalEnable(&state);
```

Interrupt: Prepare Path

```
/* Create an INTC object, tie the event to a CPU interrupt */
vectId = CSL_INTC_VECTID_4;                          // CPU interrupt
hIntcEdma = CSL_intcOpen ( &intcObjEdma,
                    CSL_INTC_EVENTID_EDMA3CC_INT1,   // interrupt event
                    &vectId , NULL );

/* Associate the EDMA dispatcher (event handler) with the INTC object */
EventRecord.handler = &eventEdmaHandler;  // eventEdmaHandler from CSL 3
EventRecord.arg = (void*)( hModule );     //                examples folder
CSL_intcPlugEventHandler( hIntcEdma, &EventRecord );

/* Enable this interrupt within the INTC module */
CSL_intcHwControl( hIntcEdma, CSL_INTC_CMD_EVTENABLE, NULL );
```

Interrupt: Set Up EDMA Dispatcher

```
unsigned long long llTemp;

/* Enable interrupt in the EDMA module */
regionIntr.region = CSL_EDMA3_REGION_GLOBAL;
llTemp = (unsigned long long)1 << tccNumber;
regionIntr.intr = _loll(llTemp);   regionIntr.intrh = _hill(llTemp);
status = CSL_edma3HwControl(hModule, CSL_EDMA3_CMD_INTR_ENABLE,
                                                    &regionIntr);

/* Hook up the EDMA event with an interrupt handler */
EdmaEventHook(tccNumber, tccFxn);

void tccFxn(void)
{
    intFlag = 1;
}
```

Interrupt: Enable Channel Interrupt

```
unsigned long long llTemp = (unsigned long long)1 << tcc;

regionIntr.region = CSL_EDMA3_REGION_1;
regionIntr.intr = _loll(llTemp);          //    regionIntr.intr   = 0x2 ;
regionIntr.intrh = _hill(llTemp);         //    regionIntr.intrh = 0x0 ;

/* Enable interrupts by setting a bit in EDMA3's IER/IERH */
status = CSL_edma3HwControl(hModule,CSL_EDMA3_CMD_INTR_ENABLE,
                                    &regionIntr);

// make sure the channel will cause an interrupt
myParamSetup.option = CSL_EDMA3_OPT_MAKE(
                                                ...
                                                CSL_EDMA3_TCINT_EN,
                                                ...
```

DAT = QDMA Made Easy

```
// open the DAT module, this gets one QDMA channel ready to use.
DAT_open(&datSetup);

// fill function copies a single byte multiple times.
DAT_fill(destAddr,byteCnt,(Uint32*)&fillVal);

// copy function does 1D copy.
DAT_copy(&src1d,&dst1d,byteCnt);

// copy2d can do 1D2D, 2D1D, 2D2D copies.
DAT_copy2d(DAT_1D2D,src1d,dst2d,16,8,20);

// wait, when needed.
DAT_wait(tcc);
```

DAT Details: It's This Easy

```
#include <csl_dat.h>
DAT_Setup datSetup;
Uint8 fillVal = 0;

// initialize the DAT module
datSetup.qchNum     = CSL_DAT_QCHA_3;                  // pick a channel
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;           // pick a region
datSetup.tccNum      = tccDat;                         // pick a channel
datSetup.paramNum = 255;                               // pick a PSET
datSetup.priority       = CSL_DAT_PRI_1;               // pick a CC/TC que
DAT_open(&datSetup);

DAT_fill(destAddr,byteCnt,(Uint32*)&fillVal);

DAT_wait(tccDat);
```

QDMA CSL: Start From a Single EDMA3 Transfer

```
CSL_edma3Init();
CSL_edma3Open(&edmaObj,CSL_EDMA3,,);
CSL_edma3HwSetup(hModule,);
```

Initialize EDMA3

```
CSL_edma3ChannelOpen(&chObj, CSL_EDMA3,,);
CSL_edma3GetHwChannelSetupParam(hChannel,);
CSL_edma3GetParamHandle(hChannel,,);
CSL_edma3ParamSetup(hParam,);
CSL_edma3HwChannelControl(hChannel,,);
```

Prepare channel

```
CSL_edma3HwChannelControl(hChannel,,);
CSL_edma3GetHwStatus(hModule,,);
CSL_edma3HwControl(hModule,,);
```

Run

```
CSL_edma3ChannelClose(hChannel);
CSL_edma3Close(hModule);
```

Close, if needed

QDMA via EDMA3 CSL: Changes

```
#include <csl_edma3.h>
CSL_edma3Init(&context);
CSL_edma3Open(&edmaObj,CSL_EDMA3,NULL,&status);

qdmahwSetup[3].que = CSL_EDMA3_QUE_1;
qdmahwSetup[3].triggerWord = 7;
hwSetup.qdmaChaSetup = qdmahwSetup;
CSL_edma3HwSetup(hModule,&hwSetup);

chAttr.chaNum = CSL_EDMA3_QCHA_3;
CSL_edma3ChannelOpen(&chObj, CSL_EDMA3, &chAttr, &status);

CSL_edma3GetHwChannelSetupParam(hChannel,&paramNum);
CSL_edma3GetParamHandle(hChannel,paramNum,NULL);
```

QDMA via EDMA3 CSL: Changes

```
// this would cause the channel to run, if it were enabled.
CSL_edma3ParamSetup(hParam, &myParamSetup);

// after this enable, the QDMA is ready to run.
CSL_edma3HwChannelControl(hChannel,
                          CSL_EDMA3_CMD_CHANNEL_ENABLE, NULL);

// now any write to trigger word will trigger the channel.
IDMA0_configArgs(0x000000ff, Src, hParam, 0);
            or
for ( i = 0; i < 8; i++ )
    ((int*)hParam)[i] = ((int*)&myParamSetup)[i];
```

Which is the better way to write eight words to PaRAM?

Harness the Power of EDMA3

- Does your DSP have EDMA3?
- Introduction to EDMA3
- Use CSL 3 to program EDMA3
- Tips, tricks, tools, techniques

Tips, Tricks, Tools, Techniques

- Comparison of IDMA0, DAT, QDMA, CPU
- RAM-based long transfer chains
- McBSP loopback using “self-linking”
- Debug capabilities
- Ending with DUMMY vs. NULL
- Clearing error conditions
- PSET allocation with PSET0 = DUMMY

Fill PaRAM: IDMA/CPU/QDMA

# of PSets =	1	2	3	4	64	254	255
IDMA no wait	139	139	141	141	621	19221	19221
IDMA and wait	285	395	485	595	6817	25217	25327
DAT fill no wait	166	166	166	166	168	168	166
DAT fill and wait	621	869	963	1197	35825	140705	141257
QDMA fill no wait (I)	138	138	140	140	140	140	140
QDMA fill and wait (I)	573	683	761	831	5241	35535	35675
QDMA fill no wait (C)	138	138	138	138	138	138	138
QDMA fill and wait (C)	457	567	645	715	5125	35425	35565
CPU write	94	190	288	384	6144	24384	24480

Use IDMA to write PSETs, don't or do wait.
Use CSL DAT_fill, don't or do wait.
Use QDMA to copy word to PSETs, don't or do wait
(write to QDMA PSET using IDMA or CPU).
Use CPU to write PSETs.

Fill PaRAM: IDMA/CPU/QDMA

# of PSets =	1	2	3	4	64	254	255
IDMA and wait	285	395	485	595	6817	25217	25327
DAT fill and wait	621	869	963	1197	35825	140705	141257
QDMA fill and wait(I)	573	683	761	831	5241	35535	35675
QDMA fill and wait(C)	457	567	645	715	5125	35425	35565
CPU write	94	190	288	384	6144	24384	24480

Use IDMA to write PSETs, don't or do wait.
Use CSL DAT_fill, don't or do wait.
Use QDMA to copy word to PSETs, don't or do wait
(write to QDMA PSET using IDMA or CPU).
Use CPU to write PSETs.

These are all cases where the writes to PaRAM will run to completion. [–o3 optimization used for all cases]

Fill PaRAM: IDMA/CPU/QDMA

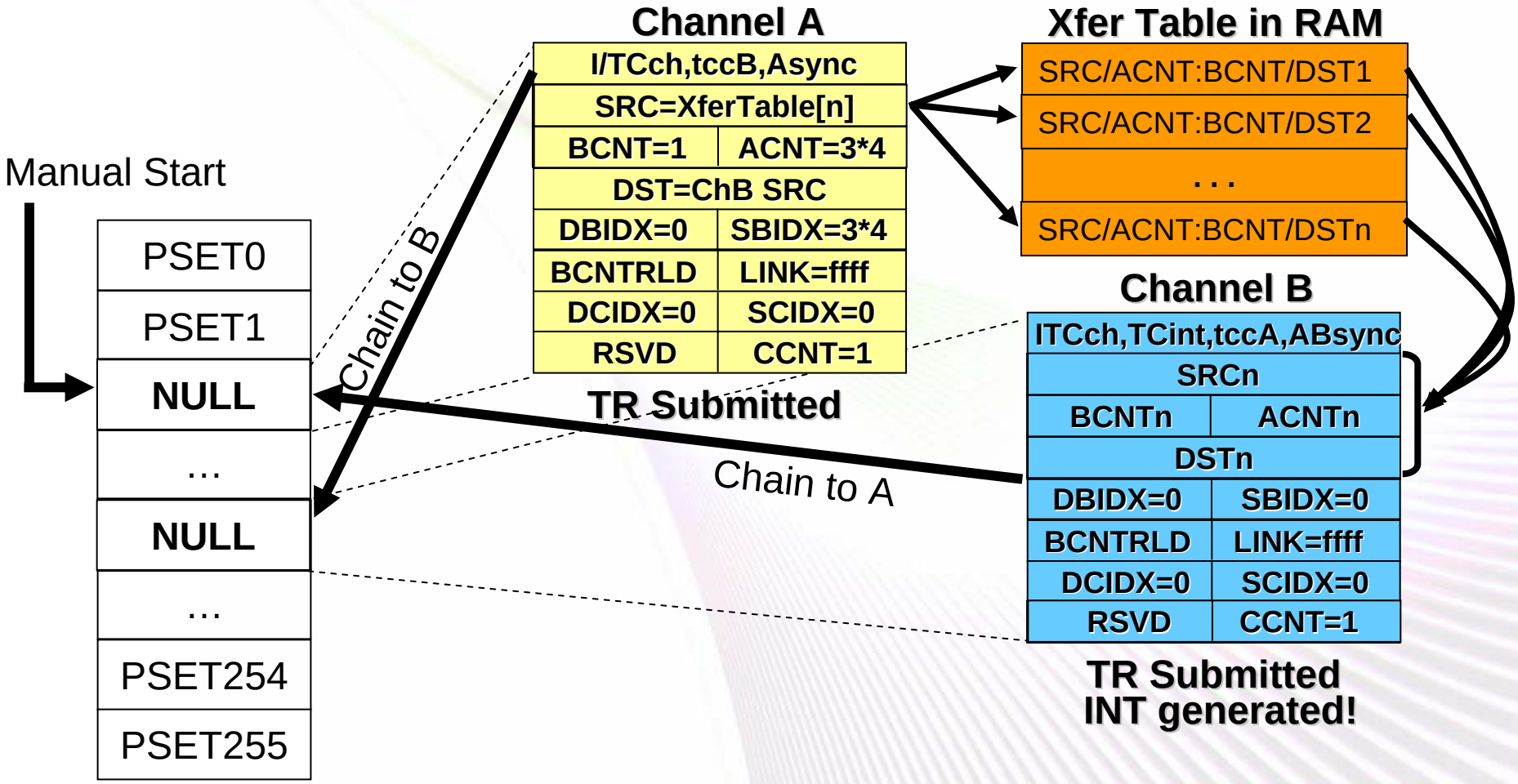
# of PSets =	1	2	3	4	64	254	255
IDMA no wait	139	139	141	141	621	19221	19221
DAT fill no wait	166	166	166	166	168	168	166
QDMA fill no wait (I)	138	138	140	140	140	140	140
QDMA fill no wait (C)	138	138	138	138	138	138	138
CPU write	94	190	288	384	6144	24384	24480

Use IDMA to write PSETs, don't or do wait.
Use CSL DAT_fill, don't or do wait.
Use QDMA to copy word to PSETs, don't or do wait
(write to QDMA PSET using IDMA or CPU).
Use CPU to write PSETs.

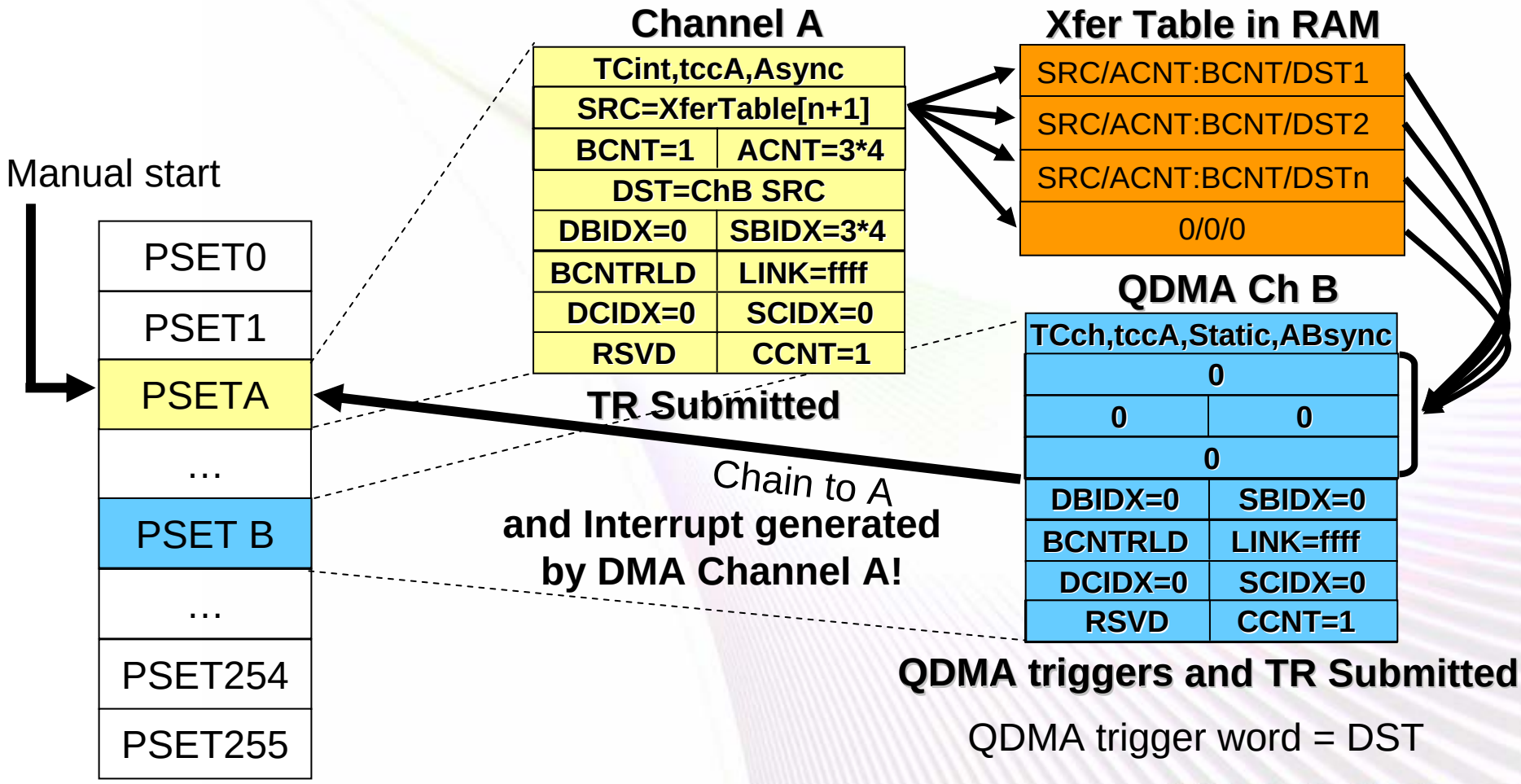
These are all cases where the writes to PaRAM will run in background and complete later (except for "CPU write").

Minds in Motion

RAM-based Long Transfer Chain

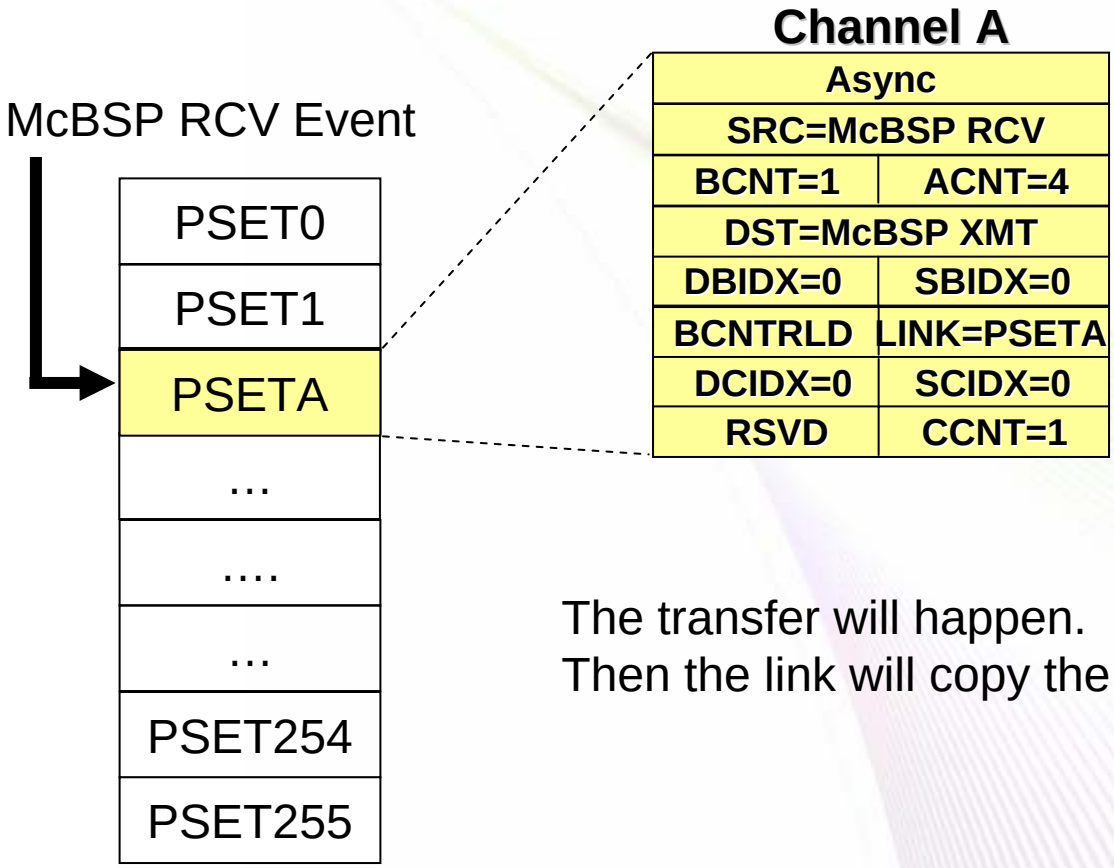


DMA/QDMA Transfer Chain



Minds in Motion

McBSP Loopback: “Self-linking”



The transfer will happen.
Then the link will copy the un-updated values back.

EDMA3 Debug Tools

- CC queue thresholds and watermarking
  - Threshold = how full the queue can get.
    - Sets error condition flag, can cause error interrupt.
  - Watermark = how full the queue has been.
- CC Queue visibility
  - Can see the last 16 queued TRs.
- TC visibility
  - Program register set status.
  - Source active registers are memory mapped.
  - Destination FIFO registers are memory mapped.
- Error detection flags

DUMMY vs. NULL

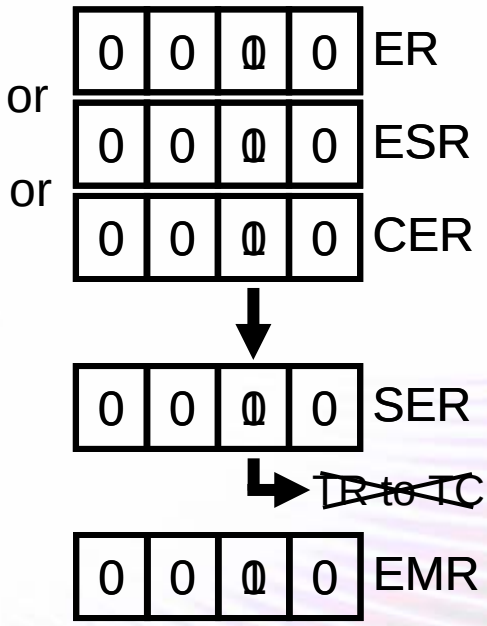
- Valid PSET
  - ACNT >= 1, BCNT >= 1, CCNT >= 1
- DUMMY PSET
  - One or two of xCNT = 0
  - Others >= 1
- NULL PSET
  - ACNT = 0, BCNT = 0, CCNT = 0

DUMMY vs. NULL

- Link to valid PSET
  - Ready for next event.
- Link to DUMMY PSET
  - Next event will not send TR (nothing to do).
  - Next event will not be recorded or cause error.
- Link to NULL PSET
  - Next event sets EMR and SER.
  - No more TRs.

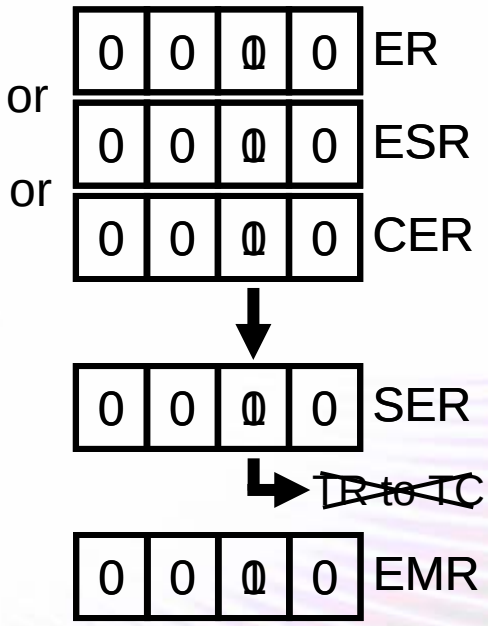
EDMA3 Event Error Conditions

- If ER or ESR or CER is set to trigger an event, the bit ‘moves’ to SER.
  - ER = Event Register
  - ESR = Event Set Register
  - CER = Chained Event Register
  - SER = Secondary Event Register
- If the PSET is not NULL, SER gets cleared and a TR is sent if PSET is not DUMMY.
- If the PSET is NULL, SER stays set and EMR gets set.
  - EMR = Event Missed Register



EDMA3 Event Error Conditions

- If the PSET was NULL when a trigger occurred, SER and EMR got set.
- If another trigger occurs, nothing happens.
  - ER or ESR or CER stays set.
  - SER and EMR stay set.
  - No TR goes out.
- How do you clear the SER and EMR to allow events to occur again?

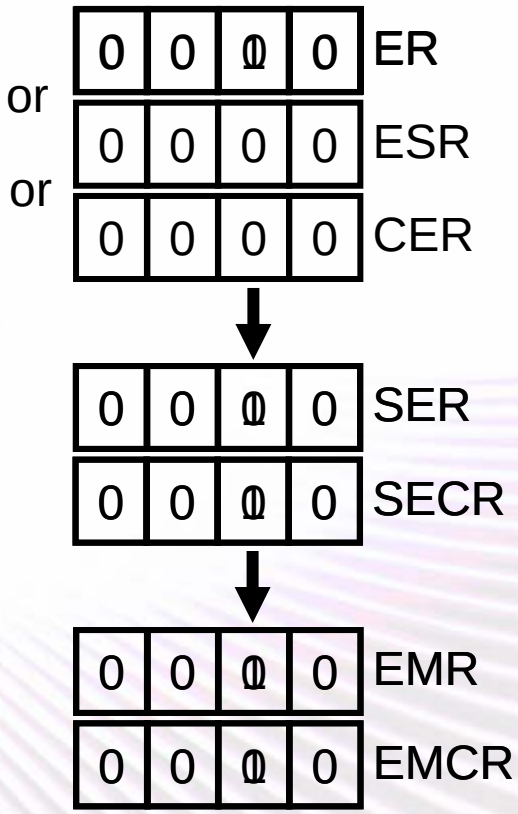


No more TRs for this channel!

Minds in Motion

EDMA3 Event Error Conditions

- Set EMCR to clear a bit in EMR.
  - EMCR = Event Missed Clear Register
- Set SECR to clear a bit in SER.
  - SECR = Secondary Event Clear Register
- But there is a pending ER bit.
  - This ‘moves’ to SER immediately, trying to cause a new TR.
  - If the PSET is still NULL, EMR gets set again.
  - Neither the SER nor EMR bits are cleared.
- Solution: Clear SER until it stays cleared, then clear EMR.



Minds in Motion

PSET Allocation, PSET0 = DUMMY

- Make PSET0 be a DUMMY set for linking.
- Initially map all channels to PSET0.
- When you open a channel, check if its paramnum is 0. Already used if not.
- If you are done with a channel, set its paramnum back to 0.

EDMA3

Questions?

Source files are available for IdmaEval and EDMA_long_chain.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

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

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

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

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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