Chapter 5

Enhanced Direct Memory Access (EDMA)

Learning Objectives

- The need for a DMA (EDMA).
- Terms and definitions (with examples).
- EDMA functionality, including:
  - Transfer modes and synchronisation.
  - EDMA interrupt.
  - Quick DMA (QDMA).
- Programming the EDMA, including:
  - Using the Chip Support Library (CSL).
  - Example “inout” program using Ping-Pong EDMA.

The Need for a DMA

- There are two methods for transferring data from one part of the memory to another, these are using:
  (1) CPU.
  (2) DMA.
- If a DMA is used then the CPU only needs to configure the DMA. Whilst the transfer is taking place the CPU is then free to perform other operations.

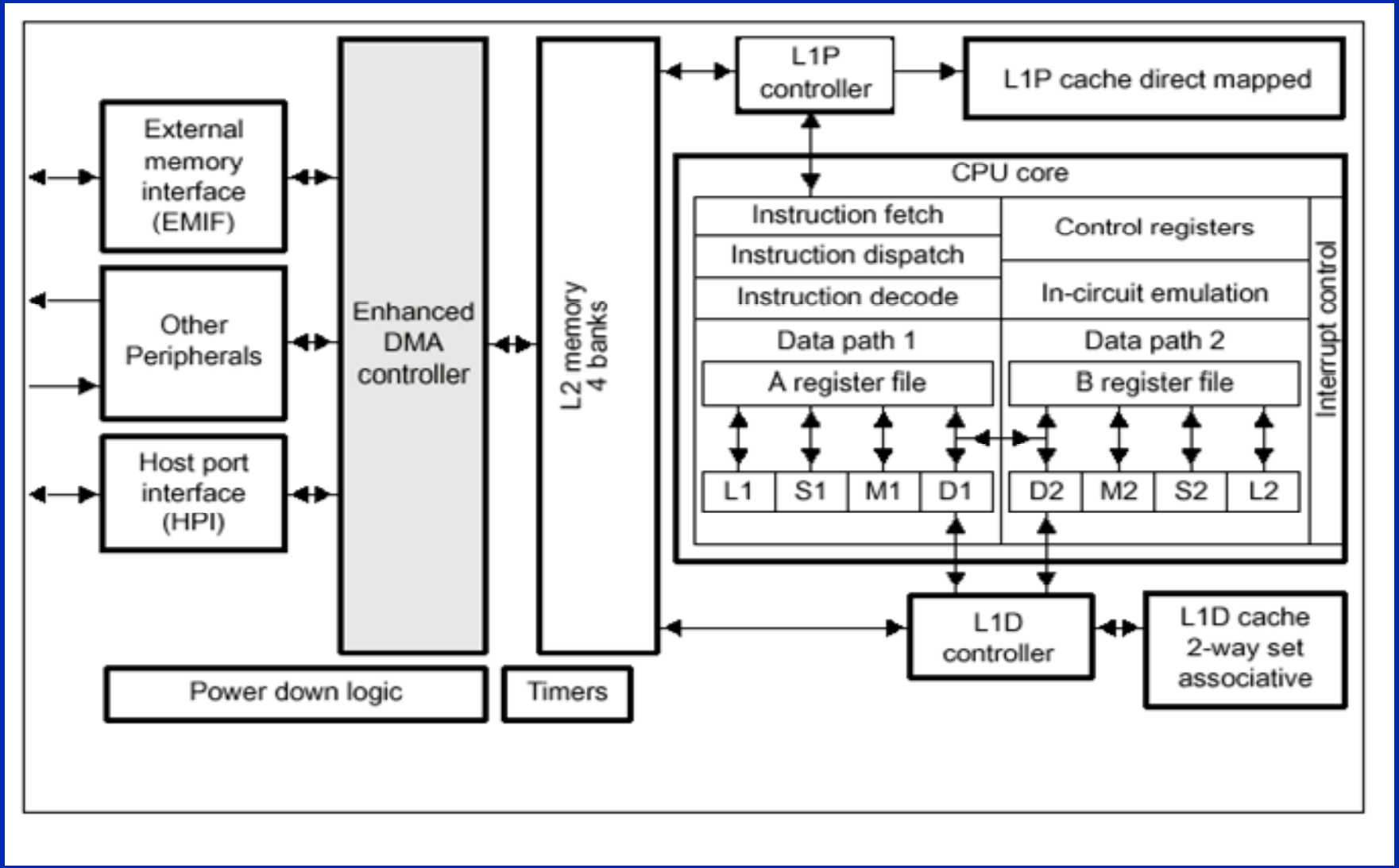
Introduction to the EDMA

- The ‘C6211/C6711 on-chip EDMA controller allows data transfers between the level two (L2) cache memory controller and the device peripherals.
- These transfers include:
  - Cache servicing.
  - Non-cacheable memory accesses.
  - User programmed data transfers.
  - Host accesses.

EDMA Interface

- **The C621x/C671x/C641x Block diagram.**

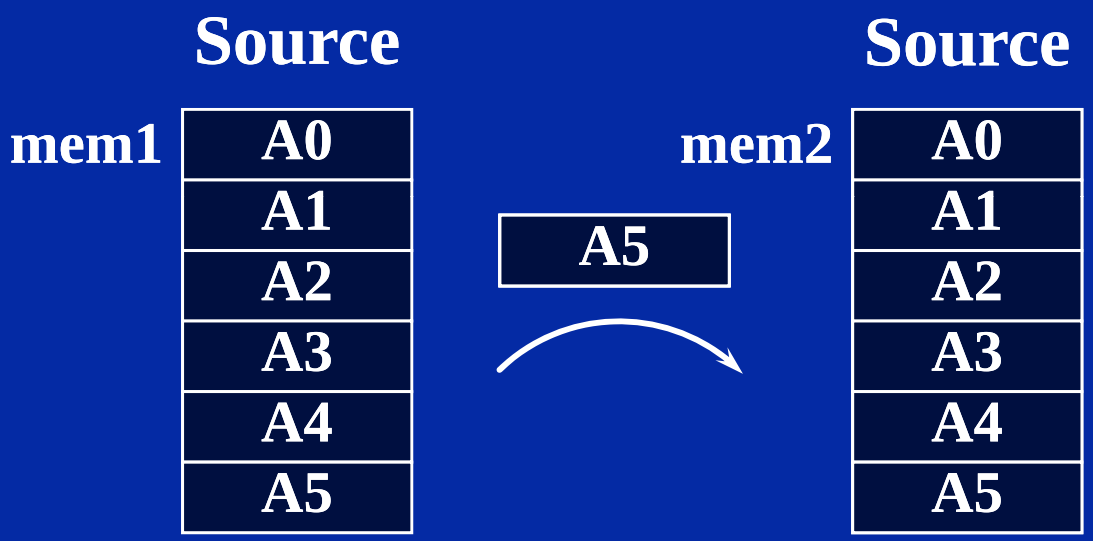
The EDMA allows data transfer to/from any addressable memory spaces.



EDMA Functionality

- The data transfer is performed with zero overhead.
- It is transparent to the CPU which means that the EDMA and CPU operations can be independent.
- However, if the EDMA and CPU both try to access the same memory location arbitration will be performed by the program memory controller.

EDMA Functionality



EDMA Features

- The ‘C6211/C6711 on-chip EDMA controller has the following features:
  - 16 channels.
  - 1 auxiliary channel dedicated for the HPI (not accessible to the user).
  - 1 Quick DMA (QDMA).

EDMA Channel Priorities

- The ‘C6211/C6711 EDMA channels have two programmable levels of priority (Level 0 reserved only for the L2).

Options (PRI 31:39)	Priority Level	Requestors
000b	Level 0: Urgent	L2 Controller *
001b	Level 1: High	EDMA, QDMA, HPI
010b	Level 2: Low	EDMA, QDMA
011-111b	Reserved	

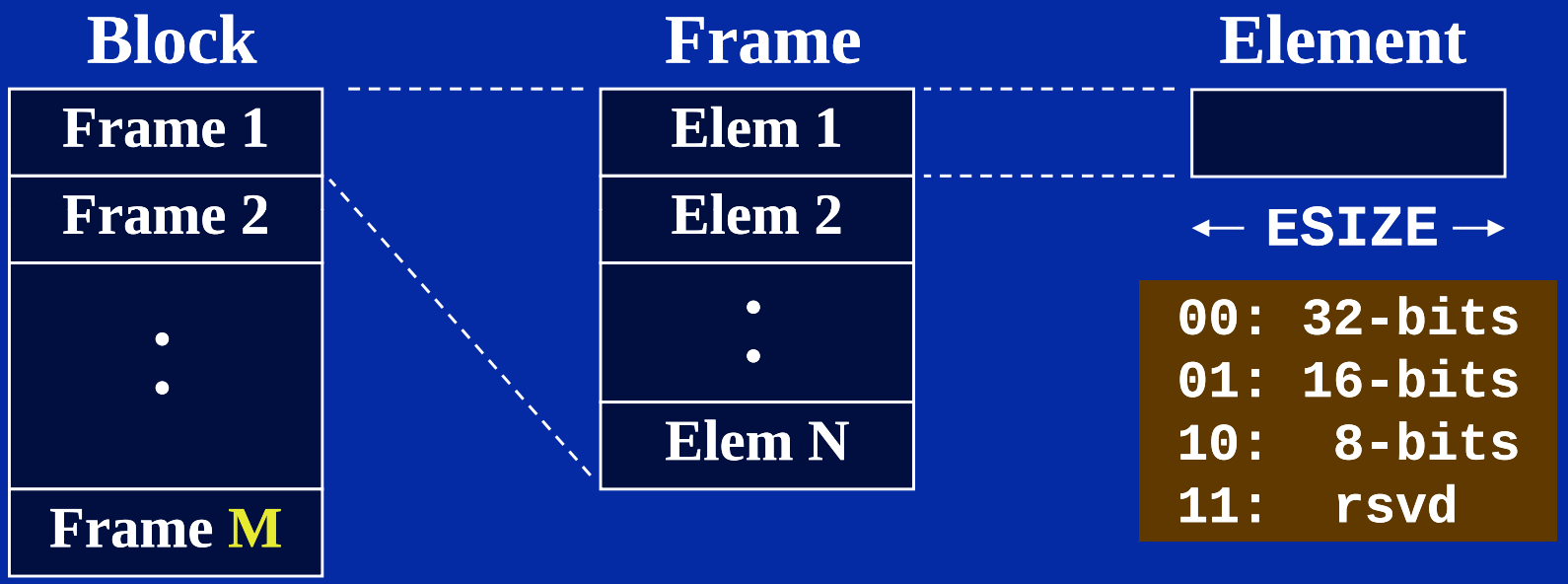
* Requests from CPU/L1 and L2 controller

EDMA Performance

- The ‘C6211/C6711 EDMA can perform element transfers with single-cycle throughput provided there is no conflict.
- The following conditions can limit the performance:
  - EDMA stalls when there are multiple transfer requests on the same priority level.
  - EDMA accesses to L2 SRAM with lower priority than the CPU.

Some Definitions

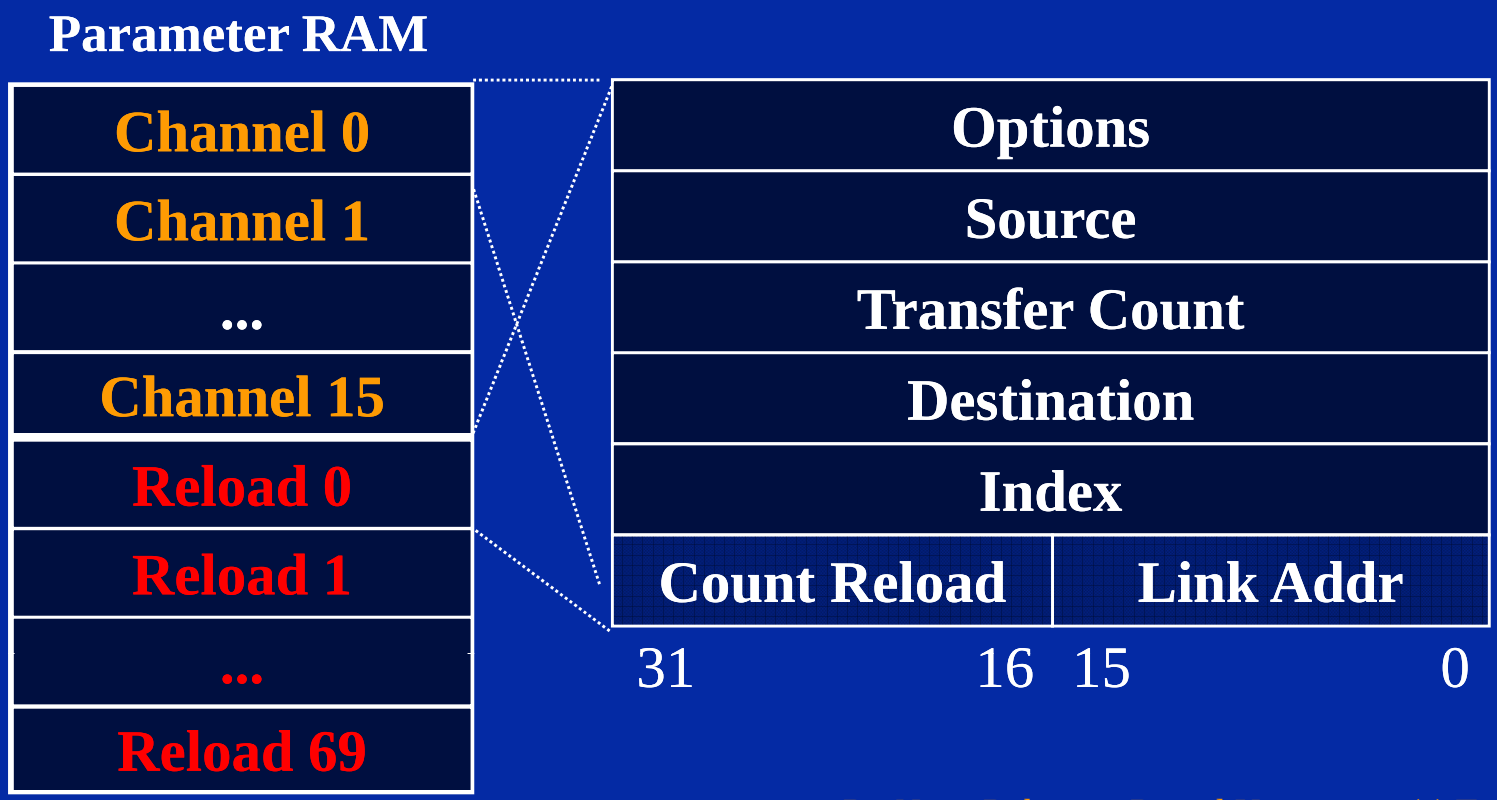
- The relation between a block, frame and element is shown below:



- N = Element count (ELECNT).
- M = Frame count (FRMCNT).
- See SPRU190 page 6-5 for more.

How the EDMA Works

- The EDMA has a parameter RAM composed of:
  - Channel parameters.
  - Reload channel parameters.

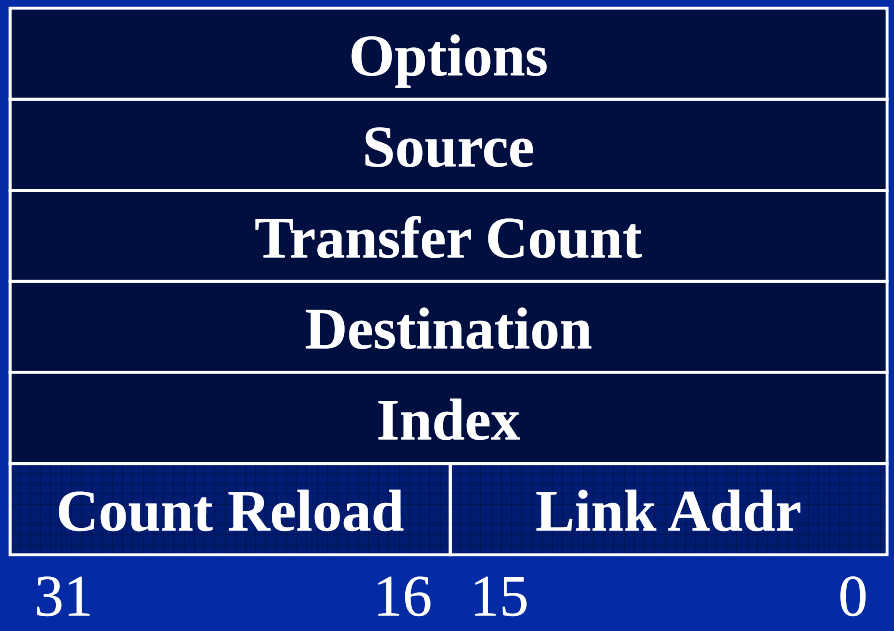


How the EDMA Works

- The EDMA has a parameter RAM composed of:
  - Channel parameters.
  - Reload channel parameters.
- The user programs both channel and reload channel parameters.
- The channel parameters contain all the information needed for the EDMA in order to perform a transfer.
- When a transfer is complete the channel parameters are reloaded from the corresponding reload channel.

EDMA Parameters

- **The parameters in the parameter table need to be determined before the EDMA can be programmed.**



EDMA Parameters (Options)

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	RSVD	LINK	FS

EDMA Channel Options Register

Bit Field	Label	Description
31:29	PRI	Priority levels for the EDMA event
28:27	ESIZE	Element size (32/16/8-bit)
26	2DS	Source dimension
25:24	SUM	Source address update mode
23	2DD	Destination dimension
22:21	DUM	Destination address update mode
20	TCINT	Transfer complete interrupt enable
19:16	TCC	Transfer complete code
1	LINK	Link
0	FS	Frame synchronisation

EDMA Parameters (Options)

Bit No.	Field	Description	Section
31–29	PRI	Priority levels for EDMA events PRI=000b; Reserved; Urgent priority. For C621x/C671x, this level is reserved ONLY for L2 requests and not valid for EDMA transfer requests. For C64x, this level is available for CPU and EDMA transfer requests. PRI=001b; High priority EDMA transfer PRI=010b; Medium priority EDMA transfer (C64x); Low priority EDMA transfer (C621x/C671x). PRI=011b; Low priority EDMA transfer (C64x); reserved (C621x/C671x) PRI=100b to 111b; reserved	6.17
28–27	ESIZE	Element size ESIZE=00b; 32-bit word ESIZE=01b; 16-bit half-word ESIZE=10b; 8-bit byte ESIZE=11b; reserved	6.9
26	2DS	Source dimension 2DS = 0; 1-dimensional source. 2DS = 1; 2-dimensional source.	6.8 and 6.11
25–24	SUM	Source address update mode SUM = 00b; Fixed address mode. No source address modification SUM = 01b; Source address increment depends on 2DS, and FS bit-fields SUM = 10b; Source address decrement depends on 2DS, and FS bit-fields SUM = 11b; Source address modified by the element index/frame index depending on 2DS, and FS bits.	6.11

EDMA Parameters (Options)

23	2DD	Destination dimension 2DD = 0; 1-dimension destination. 2DD = 1; 2-dimensional destination.	6.8 and 6.11
22–21	DUM	Destination address update mode DUM = 00b; Fixed address mode. No destination address modification DUM = 01b; Destination increment depends on 2DD, and FS bit-fields DUM = 10b; Destination decrement depends on 2DD, and FS bit-fields DUM = 11b; Destination modified by the element index/frame index depending on 2DD, and FS bits.	6.11
20	TCINT	Transfer complete interrupt TCINT=0; Transfer complete indication disabled. CIPR bits are not set upon completion of a transfer. TCINT=1; The relevant CIPR bit is set on channel transfer completion. The bit (position) set in the CIPR is the TCC value specified.	6.14 and 6.15
19–16	TCC	Transfer complete code TCC=0000b to 1111b; 4-bit code is used to set the relevant bit in CIPR (i.e. CIPR[TCC] bit) provided TCINT=1, when the current set is exhausted. For C64x, the 4-bit TCC code is used in conjunction with bit field TCCM for a 6-bit transfer complete code.	6.14 and 6.15
1	LINK	Link LINK=0; Linking of event parameters disabled. Entry not reloaded. LINK=1; Linking of event parameters enabled. After the current set is exhausted, the channel entry is reloaded with the parameter set specified by the link address. The link address must be on a 24-byte boundary and within the EDMA PaRAM. The link address is a 16-bit address offset from the PaRAM base address.	6.6.7 and 6.12
0	FS	Frame synchronization FS=0; Channel is element/array synchronized. FS=1; Channel is frame synchronized. The relevant event for a given EDMA channel is used to synchronize a frame.	6.7

EDMA Parameters

- **Source:** Start address of the source.
- **Transfer Count:**
  - Upper 16 bits [31:16]: Frame count.
  - Lower 16 bits [15:0]: Element count.
- **Destination:** Start address of the destination.

EDMA Parameters

- **Index:**
  - **Upper 16 bits [31:16]: Frame index.**
  - **Lower 16 bits [15:0]: Element index.**
- **Count reload: Value to be reloaded during into the element count when a frame is complete (only used in 1-D mode).**
- **Link address: Specifies the address from where the parameters are reloaded. The 16-bit value is added to 0x01A0 xxxx to form the 32-bit address.**

EDMA Synchronisation

- **Two methods for initiating a transfer:**
  - **(1) CPU initiated.**

This is known as unsynchronised EDMA. With this method the CPU writes to the Event Register (ER) through the Event Set Register (ESR) in order to start the EDMA transfer (this can be used to simulate an event).

EDMA Synchronisation

- **Two methods for initiating a transfer:**
  - **(1) CPU initiated.**
  - **(2) Event triggered.**

In this case the event is latched in the Event Register (ER) which then triggers the transfer.

The events that can trigger a transfer are given on the following slide.

EDMA Events

Channel	Event	Event Description
0	DSPINT	Host port host to DSP interrupt
1	TINT0	Timer 0 interrupt
2	TINT1	Timer 1 interrupt
3	SD_INT	EMIF SDRAM timer interrupt
4	EXT_INT4	External interrupt pin 4
5	EXT_INT5	External interrupt pin 5
6	EXT_INT6	External interrupt pin 6
7	EXT_INT7	External interrupt pin 7
8	EDMA_TCC8	EDMA transfer complete code 1000b interrupt
9	EDMA_TCC9	EDMA TCC 1001b interrupt
10	EDMA_TCC10	EDMA TCC 1010b interrupt
11	EMDA_TCC11	EDMA TCC 1011b interrupt
12	XEVT0	McBSP0 transmit event
13	REVT0	McBSP0 receive event
14	XEVT1	McBSP1 transmit event
15	REVT1	McBSP1 receive event

- **An event can be cleared using the CPU by writing to the Event Clear Register (ECR).**

Transfer Synchronisation

- The synchronisation mode depends on whether or not the transfer is two dimensional.
- Therefore first specify that the transfer is either 1-D or 2-D.

- 0b = 1 dimensional
- 1b = 2 dimensional

1-D Synchronisation

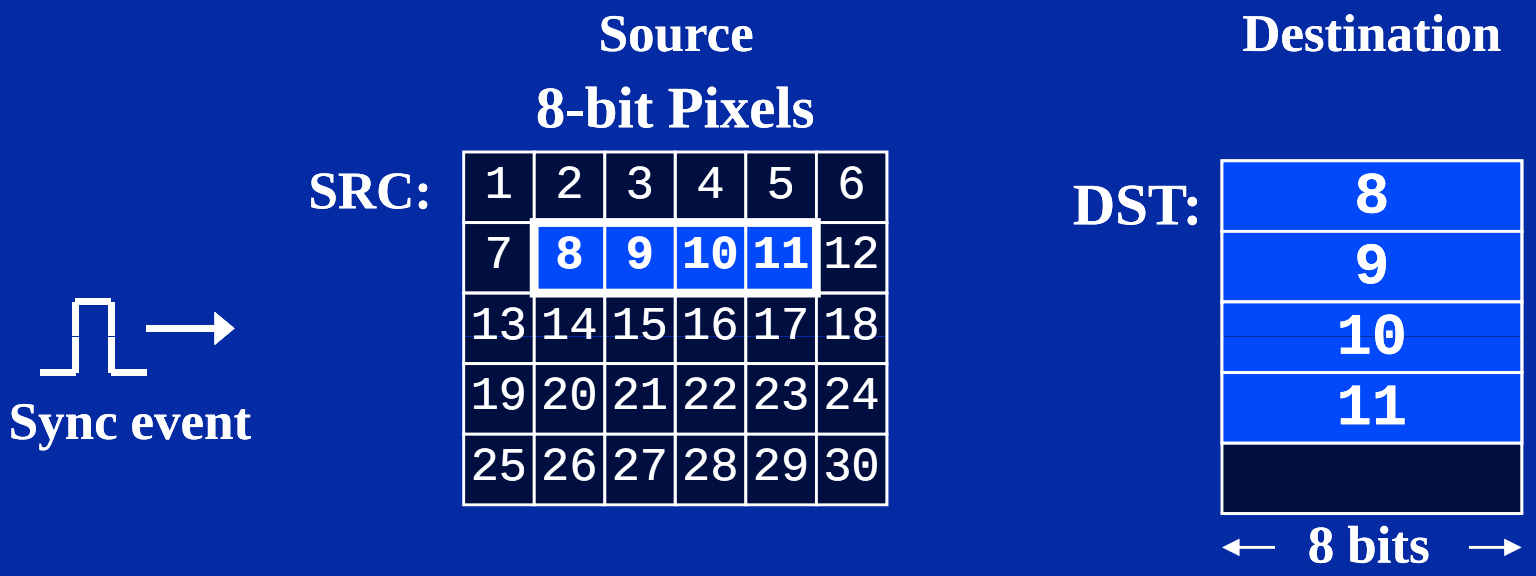
- There are two modes of synchronisation in the 1-D transfer mode.
- These are:
  - Element synchronised: each event causes one element to be transferred.
  - Frame synchronised: each event causes a whole frame to be transferred.
- The FS bit is used to specify the synchronisation mode.

1-D Transfer Synchronisation

- **Element synchronised (FS=0):**
  - **In this case the EDMA transfers each element after receiving the synchronisation event until the whole frame is transferred.**
  - **The element count is then reloaded and the frame count decremented.**
  - **The frame index is then added to the last element address to calculate the next frame start address.**
  - **If the link is enabled (LINK=1b) then the transfer parameters are reloaded.**

Options Register

Example: 1-D Transfer with FS=0



- ESIZE = 10b
- ELECNT = 4
- FRMCNT = 0
- SUM = 01b

- 00b = 32-bits
- 01b = 16-bits
- 10b = 8-bits
- 11b = Reserved

- FRMCNT = M - 1

- 01b = Increment
- 10b = Decrement
- 11b = Modified by INDEX

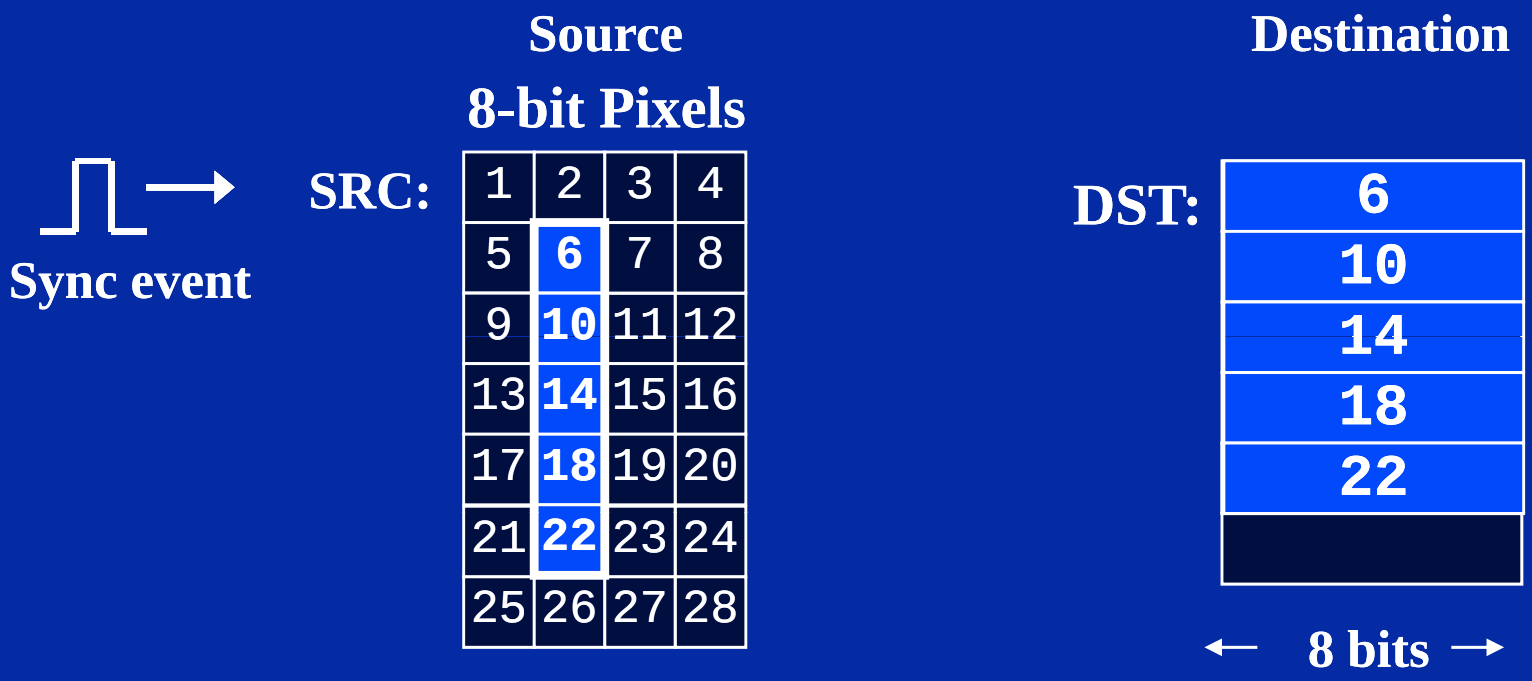
Example: 1-D Transfer with FS=0

ESIZE

	10	Options
Source = SRC+7		
FRMCNT = 0		ELECNT = 4
Destination = DST		
FRMIDX		ELEIDX
Count Reload		Link Addr

31 16 15 0

Example: 1-D Transfer with FS=0



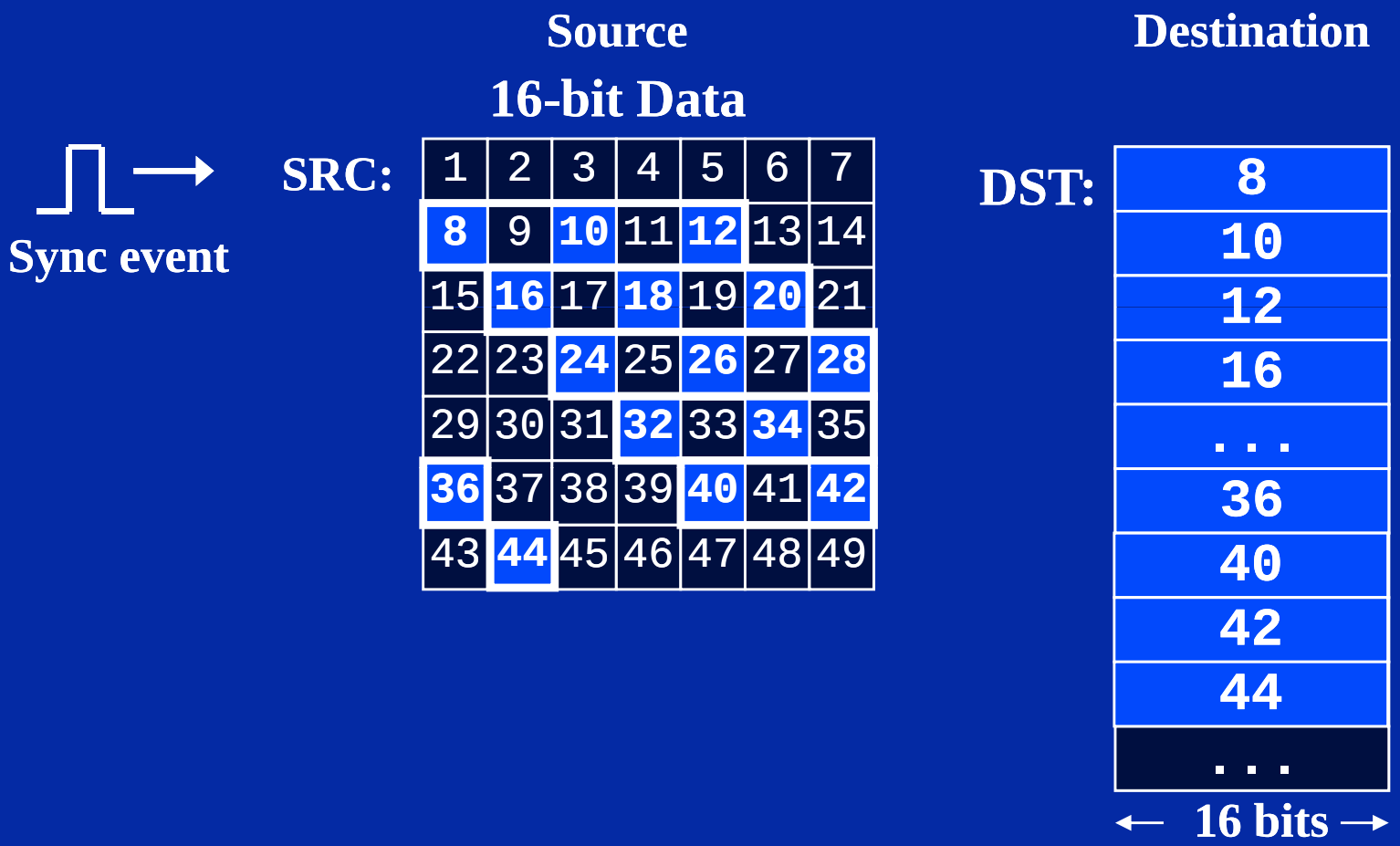
- ESIZE = 10b (8-bits)
- ELECNT = 5, FRMCNT = 0
- SUM = 11b
- ELEIDX = 4

In this mode:
Next frame addr = Last element addr + FRMIDX

1-D Transfer Synchronisation

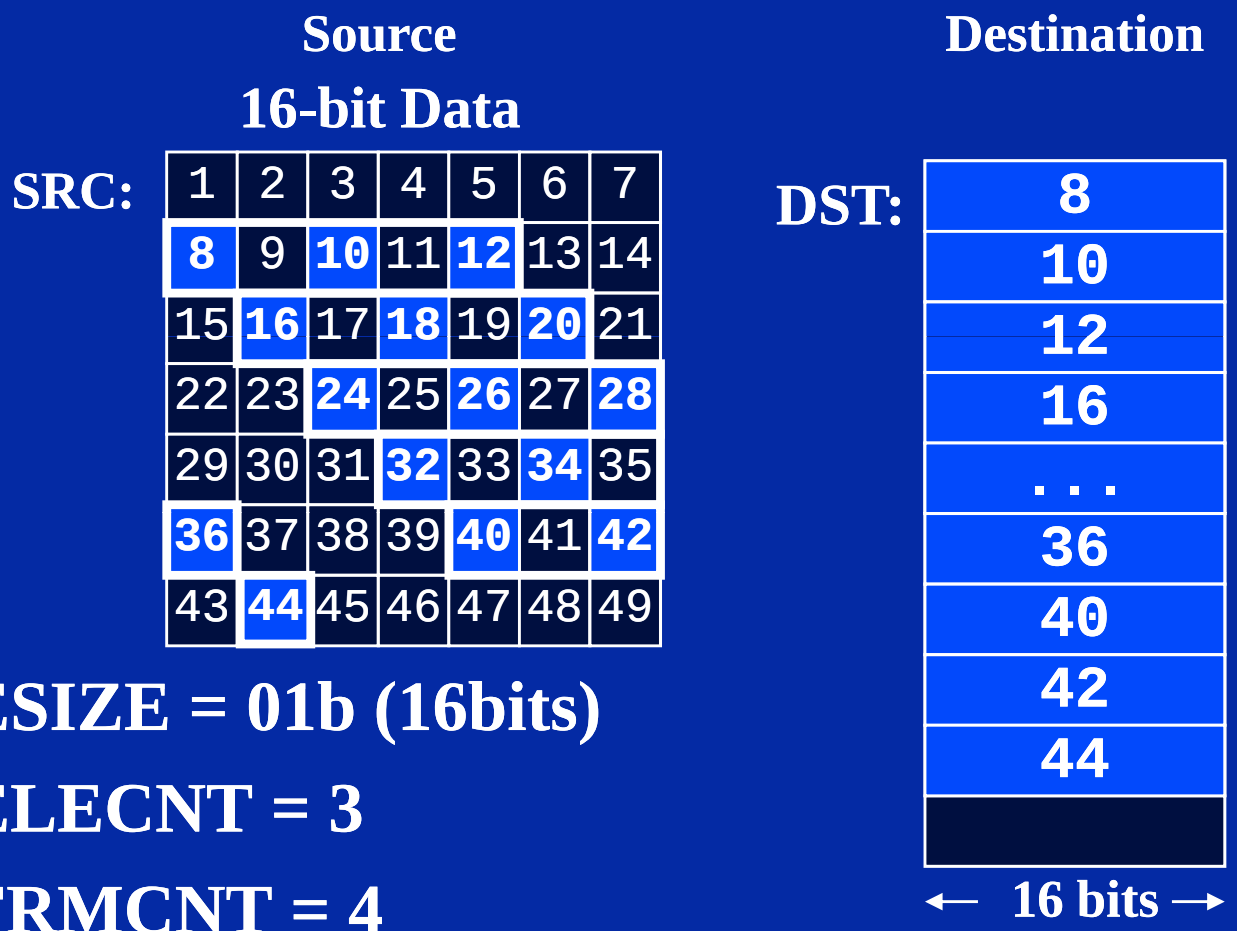
- **Frame synchronised (FS=1):**
  - **In this case the EDMA transfers a whole frame after receiving the synchronisation event.**
  - **In this mode the frame index no longer represents the difference between the address of the last element of the frame and the start address of the next frame.**
  - **The frame index is added to the start address of the frame to derive the start address of the next frame.**
  - **See the example on the following slide.**

Example: 1-D Transfer with FS=1



- All elements in a frame are offset by ELEIDX bytes.
- All frames in a block are offset by FRMIDX bytes.

Example: 1-D Transfer with FS=1



- ESIZE = 01b (16bits)
- ELECNT = 3
- FRMCNT = 4
- SUM = 11b, 2DS = 0, FS = 1
- FRMIDX = 16, ELEIDX = 4

In this mode:
Next frame addr = First element addr + FRMIDX

Example: 1-D Transfer with FS=1

Sync event

Source

8-bit Pixels

SRC:

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

Destination

DST:

2
4
14
16
26
28

← 8 bits →

- ESIZE = 10b
- ELECNT = 2
- FRMCNT = 2
- SUM = 01b, 2DS =0b, FS = 1
- FRMIDX = 12, ELEIDX = 2

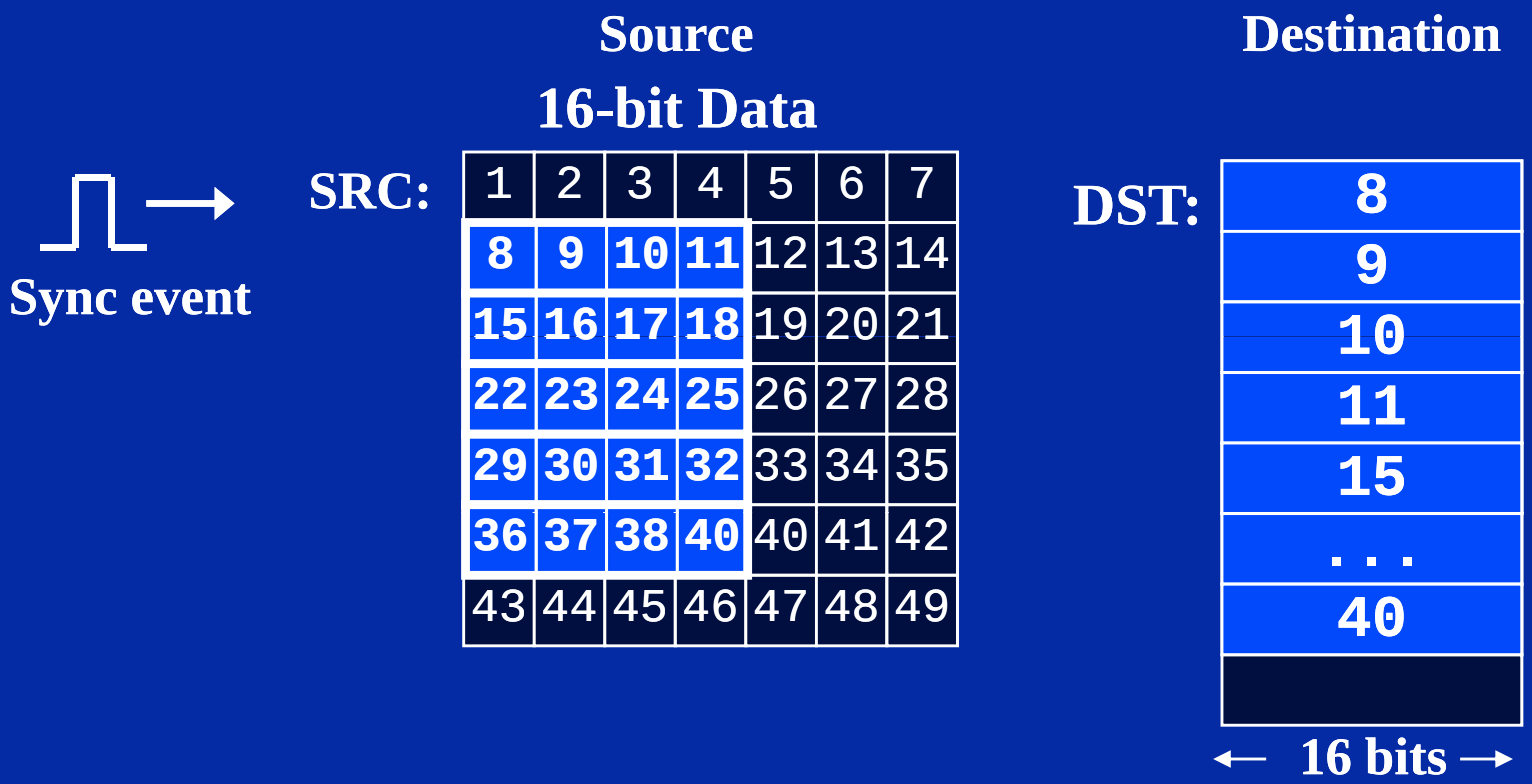
2-D Synchronisation

- There are two modes of synchronisation in the 2-D transfer mode.
- These are:
  - Array synchronised: each event causes one line of the array to be transferred.
  - Block synchronised: each event causes a the entire block to be transferred.
- The FS bit is used to specify the synchronisation mode.

2-D Transfer Synchronisation

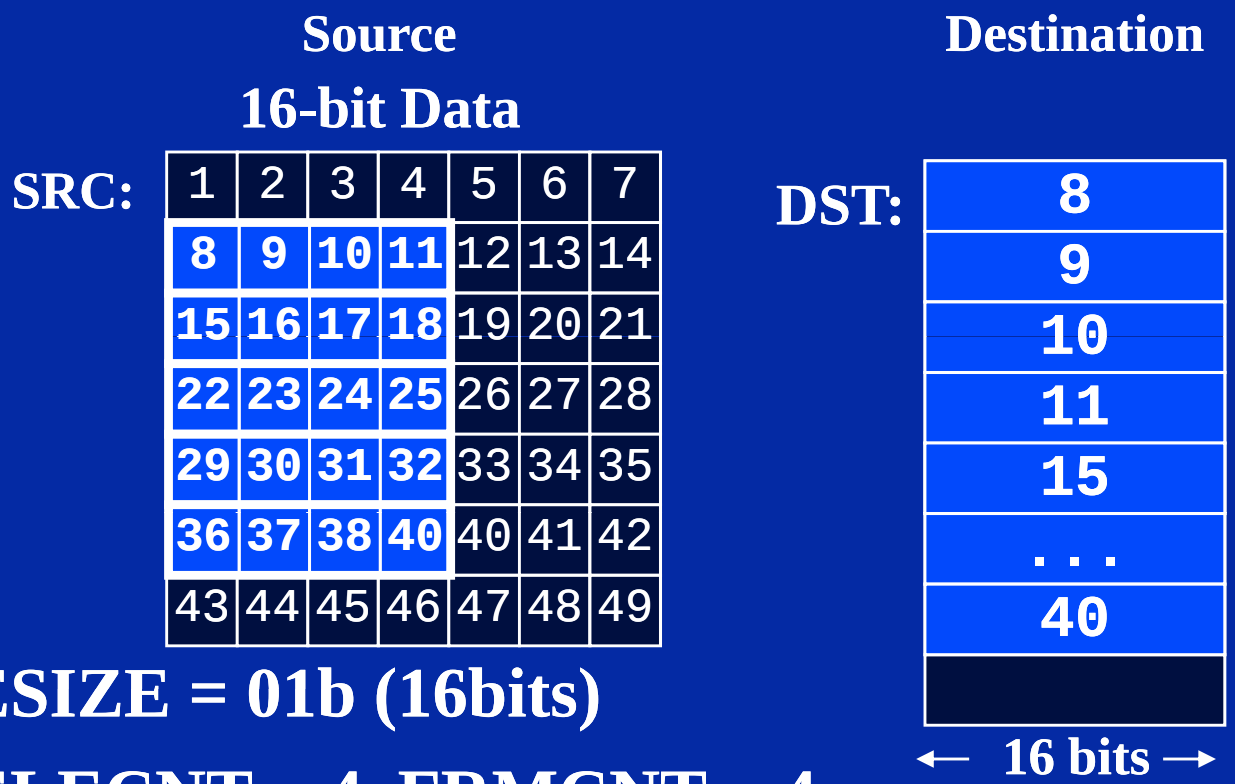
- **Array synchronised (FS=0):**
  - **This is the same as the 1-D frame synchronisation mode except that the elements are all contiguous.**
  - **One array is transferred per synchronisation event.**
  - **The frame count is equal to the number of frames minus one because when the FRMCNT=0 the complete transfer parameters are reloaded after sending the last transfer request to the address generation hardware.**
  - **The frame index is added to the start address of the frame to derive the next frame address.**

Example: 2-D Transfer with FS=0



- With 2D transfers there is no indexing between elements, therefore ELEIDX is not used in 2D transfers.
- To specify a 2D source set 2DS in channel options.

Example: 2-D Transfer with FS=0



- ESIZE = 01b (16bits)
- ELECNT = 4, FRMCNT = 4
- SUM = 01b, 2DS = 1b
- FRMIDX = 14, ELEIDX = N/A

In this mode:
Next frame addr = First element addr + FRMIDX

2-D Transfer Synchronisation

- **Block synchronised (FS=1):**
  - **This is the same as the 1-D frame synchronisation mode except that the elements are all contiguous.**
  - **The entire block is transferred following the synchronisation event.**
  - **At the end of each frame the frame index is added to the last element address to calculate the next frame start address.**

Example: 2-D Transfer with FS=1

Sync event

Source

16-bit Data

SRC:

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	40	40	41	42
43	44	45	46	47	48	49

Destination

DST:

8
9
10
11
15
. . .
40

← 16 bits →

- ESIZE = 01b (16bits)
- ELECNT = 4, FRMCNT = 4
- SUM = 01b, 2DS = 1b
- FRMIDX = 8, ELEIDX = N/A

In this mode:
Next frame addr = Last element addr + FRMIDX

EDMA Interrupt Generation

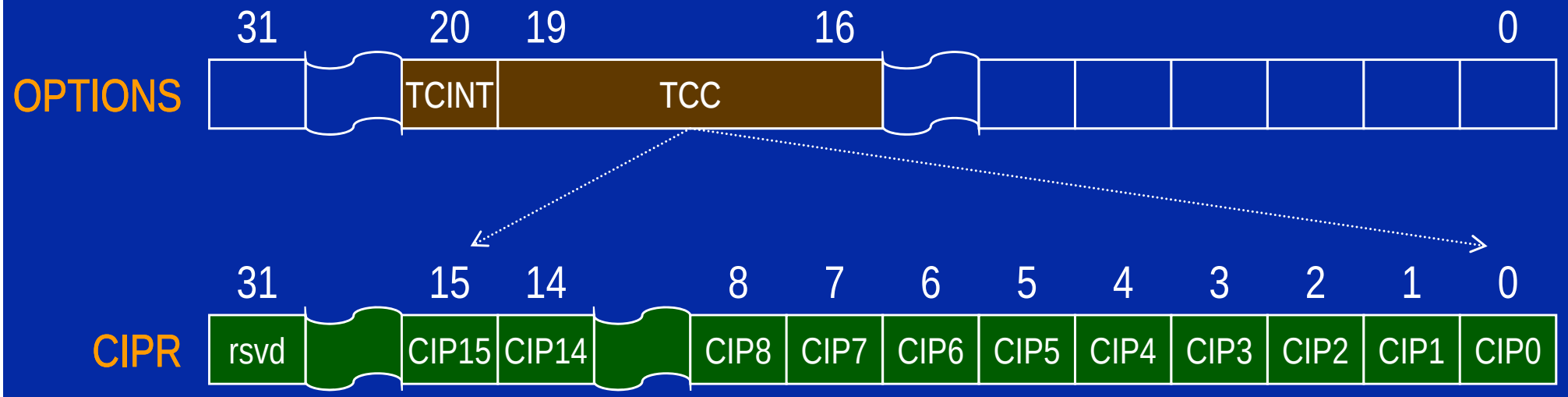
- The EDMA controller is responsible for generating transfer completion interrupts to the CPU.
- The EDMA generates a single interrupt (EDMA_INT) to the CPU on behalf of all 16 channels.
- The programmer has to read the CIPR register to determine which channel caused the interrupt or which interrupts are pending while the ISR is being serviced.

EDMA Interrupt Generation

- **Each channel has an OPTIONS register**
- **OPTIONS contains:**
  - **TCINT - do you want to interrupt CPU?**
  - **TCC - 4 bit completion code (your choice)**

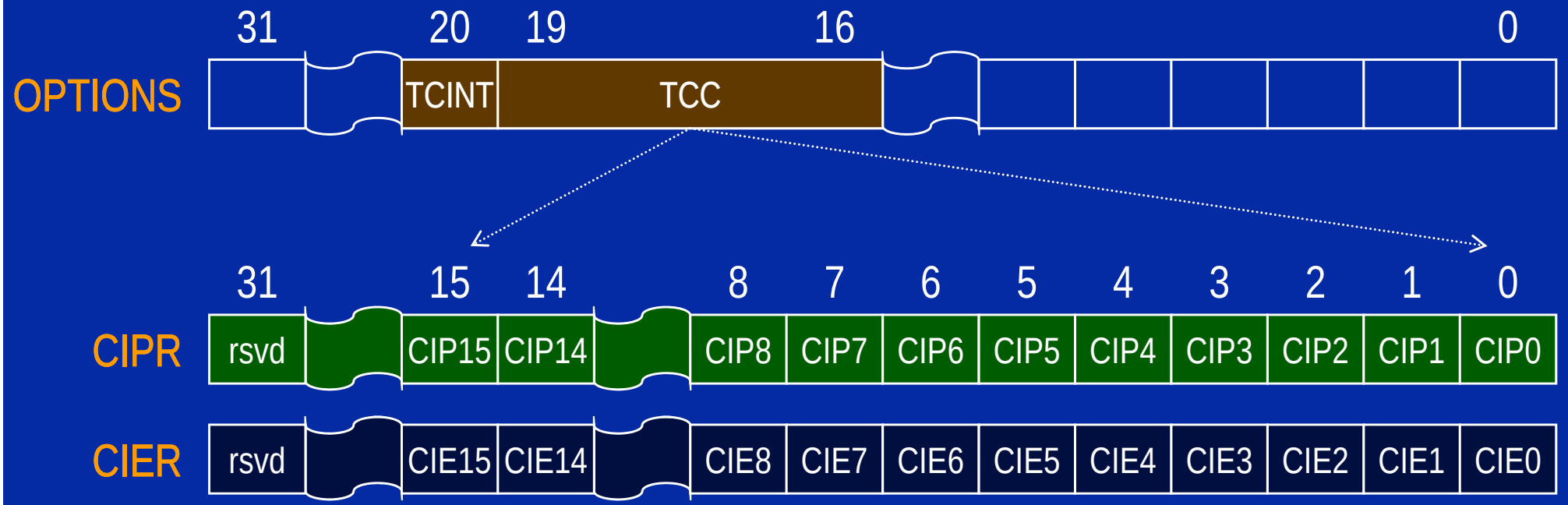
EDMA Interrupt Generation

- **Each channel has an OPTIONS register**
- **OPTIONS contains:**
  - **TCINT - do you want to interrupt CPU?**
  - **TCC - 4 bit completion code (your choice)**
- **Upon completion: If you set TCINT = 1, then CIPR bit equal to TCC value (you set) is set to one**



EDMA Interrupt Generation

- Each channel has an OPTIONS register
- OPTIONS contains:
  - TCINT - do you want to interrupt CPU?
  - TCC - 4 bit completion code (your choice)
- Upon completion: If you set TCINT = 1, then CIPR bit equal to TCC value (you set) is set to one
- CIER bit must be set for CPU to be interrupted
- Only 1 EDMA interrupt to CPU. Upon int, CPU should service all pending channels set in CIPR



Chaining EDMA Transfers

- After completion of an EDMA channel transfer another EDMA channel transfer can be triggered.
- This triggering mechanism is similar to event triggering.
- However this method can only be used to trigger EDMA channels 8 to 11.

Chaining EDMA Transfers

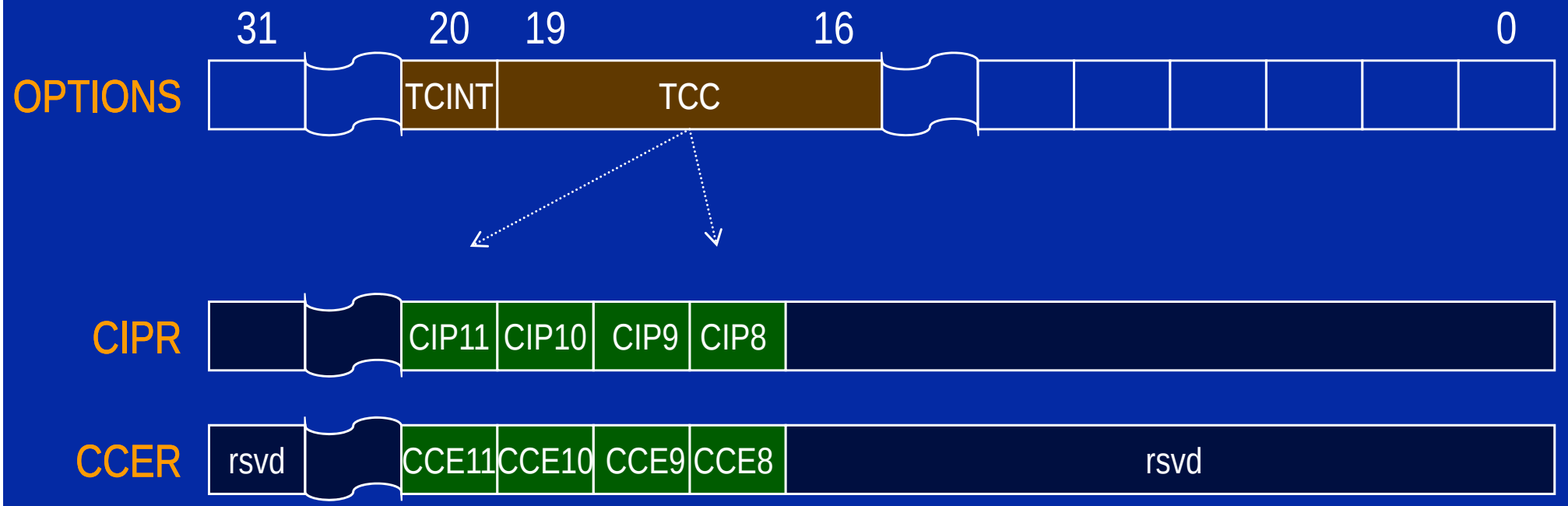
- **In addition to setting the TCINT and TCC bits there is also another register, the Channel Chain Enable Register (CCER) that must be set.**

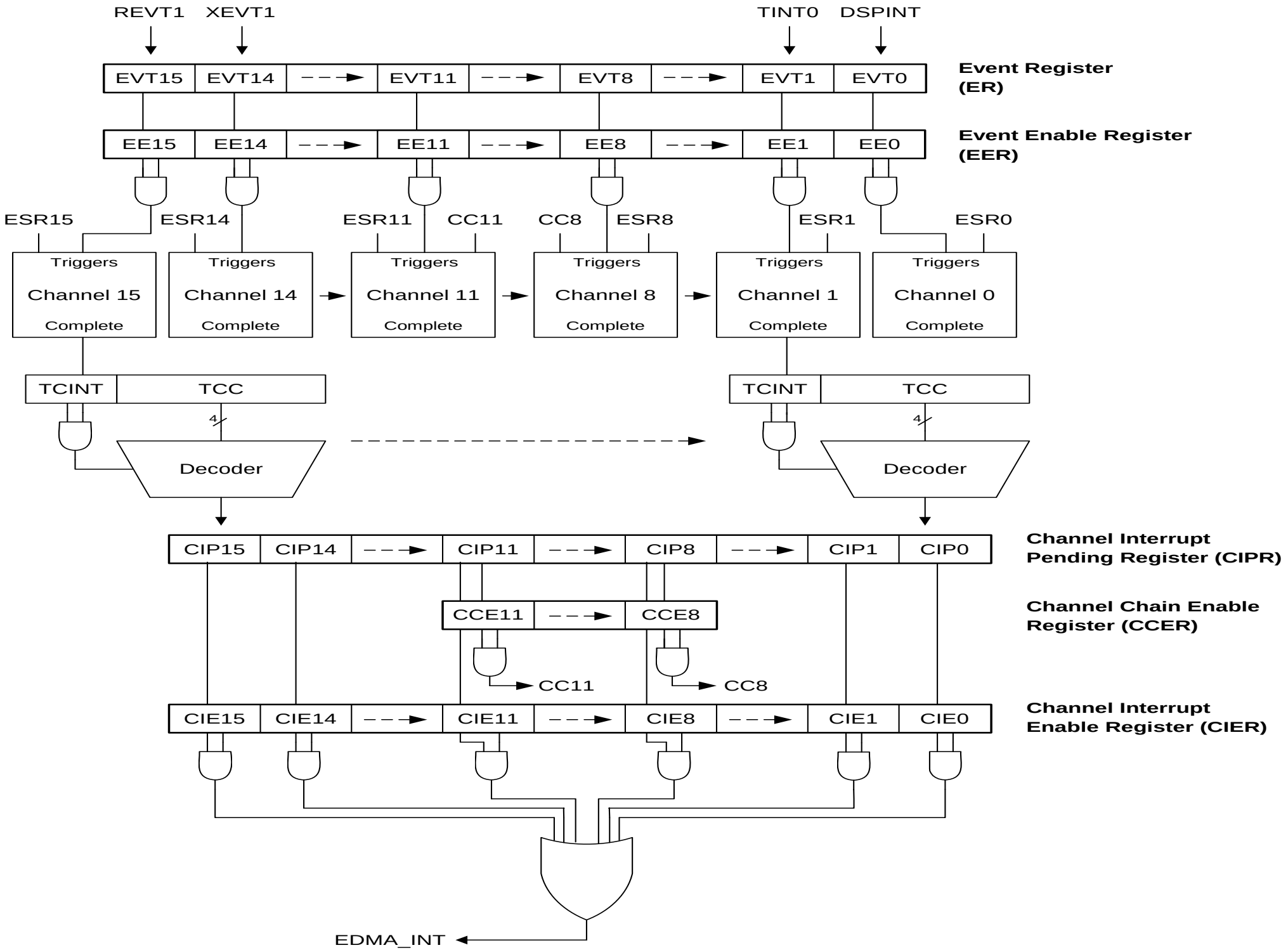
Channel Chain Enable Register (CCER)

Chaining EDMA Transfers

Channel Changing … ah, that is, Channel Chaining

- Upon completion, one EDMA channel can kick-off another
- Rather than setting CIER bit (to enable CPU int), set CCER to enable channel 8-11 to start



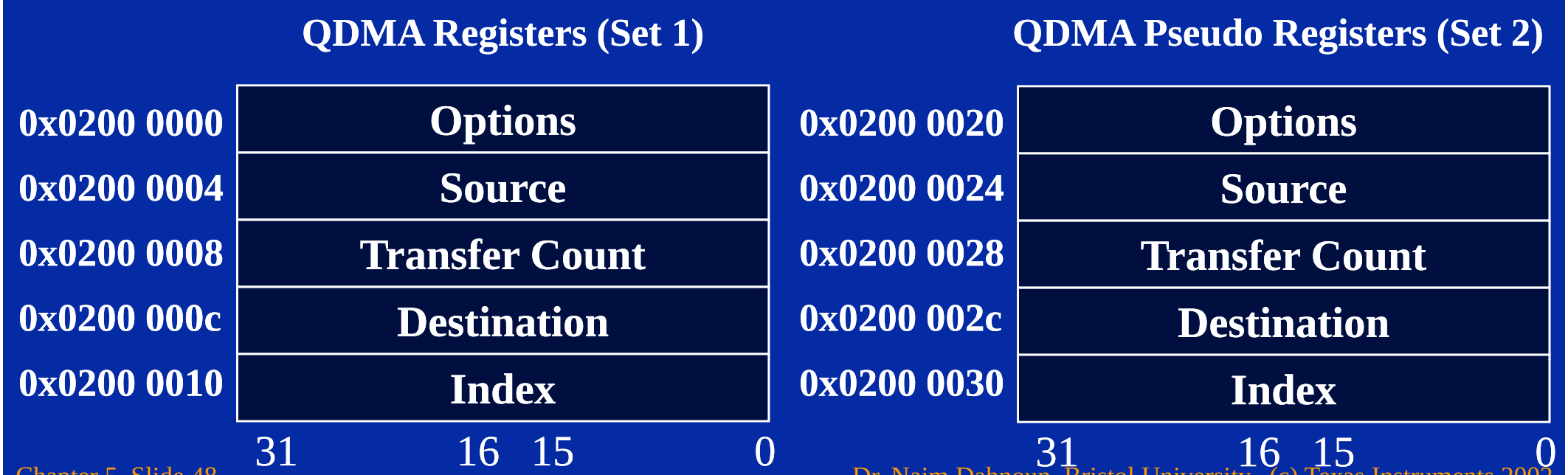

REVT1
XEVT1
TINT0
DSPINT
EVT15
EVT14
EVT11
EVT8
EVT1
EVT0
Event Register (ER)
EE15
EE14
EE11
EE8
EE1
EE0
Event Enable Register (EER)
ESR15
ESR14
ESR11
CC11
CC8
ESR8
ESR1
ESR0
Triggers
Channel 15 Complete
Channel 14 Complete
Channel 11 Complete
Channel 8 Complete
Channel 1 Complete
Channel 0 Complete
TCINT
TCC
4
Decoder
CIP15
CIP14
CIP11
CIP8
CIP1
CIP0
Channel Interrupt Pending Register (CIPR)
CCE11
CCE8
Channel Chain Enable Register (CCER)
CC11
CC8
CIE15
CIE14
CIE11
CIE8
CIE1
CIE0
Channel Interrupt Enable Register (CIER)
EDMA_INT

Introduction to the Quick DMA (QDMA)

- The QDMA provides a very efficient way of moving data.
- It supports nearly all the same modes as the EDMA however transfer requests are submitted faster.
- However it does not support reload of a count or link.
- Therefore the QDMA is suited to one-off moves of blocks of data internally whereas the EDMA is suited to moving data between peripherals and the memory.

QDMA Registers

- The QDMA is programmed via two sets of five write-only memory mapped registers:
  - Writing to the first set of registers configures the QDMA but does not submit a request.
  - Writing to the second set of registers configures the QDMA and submits a transfer request.



QDMA Options Register

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	RSVD	RSVD	FS

EDMA Channel Options Register

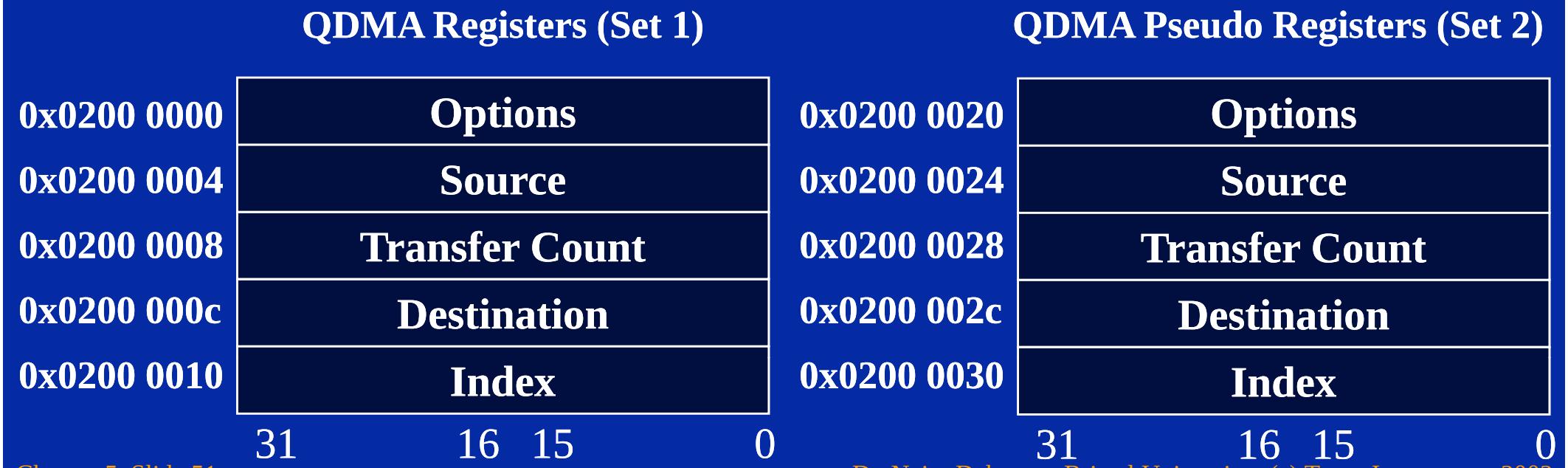
Bit Field	Label	Description
31:29	PRI	Priority levels for the QDMA event
28:27	ESIZE	Element size (32/16/8-bit)
26	2DS	Source dimension
25:24	SUM	Source address update mode
23	2DD	Destination dimension
22:21	DUM	Destination address update mode
20	TCINT	Transfer complete interrupt enable
19:16	TCC	Transfer complete code
0	FS	Frame synchronisation

Other QDMA Registers

- **Source:** Start address of the source.
- **Transfer Count:**
  - Upper 16 bits [31:16]: Frame count.
  - Lower 16 bits [15:0]: Element count.
- **Destination:** Start address of the destination.
- **Index:**
  - Upper 16 bits [31:16]: Frame index.
  - Lower 16 bits [15:0]: Element index.

Configuring the QDMA

- **Five writes are required to submit a request:**
  - **The first four registers are configured by writing to the corresponding QDMA registers.**
  - **The fifth register is configured by writing to the corresponding pseudo register causing the request to be submitted.**



Features of the QDMA

- All transfers are frame synchronised.
- Although linking is not supported, completion interrupts and channel chaining are supported.
- The values held in the registers are not changed by the hardware hence the same transfer can be repeated by a single write to one of the pseudo registers.

Programming the EDMA

- There are three methods available for programming the EDMA:
  - (1) Writing directly to the EDMA registers.
  - (2) Using the Chip Support Library (CSL).
  - (3) Graphically using the DSP/BIOS GUI interface.

Programming the EDMA - Direct

(1) Writing directly to the EDMA registers:

- Although this method is straightforward, it relies on a good understanding of the EDMA and the DSP memory map.
- This method is tedious and prone to errors.

```
#include <intr.h>
#include <regs.h>
#include <c6211dsk.h>

void EDMA_setup (void)
{
        *(unsigned volatile int *) ECR = 0xffff;
        *(unsigned volatile int *) EER = 0xffff;
        *(unsigned volatile int *) CIPR = 0xffff;
        *(unsigned volatile int *) CIER = 0xffff;
        ...
}
```

Programming the EDMA - CSL

(2) Using the Chip Support Library:

- The CSL provides a C language interface for configuring and controlling the on-chip peripherals, in this case the EDMA.
- The library is modular with each module corresponding to a specific peripheral. This has the advantage of reducing the code size.
- Some modules rely on other modules also being included, for example the IRQ module is required when using the EDMA module.

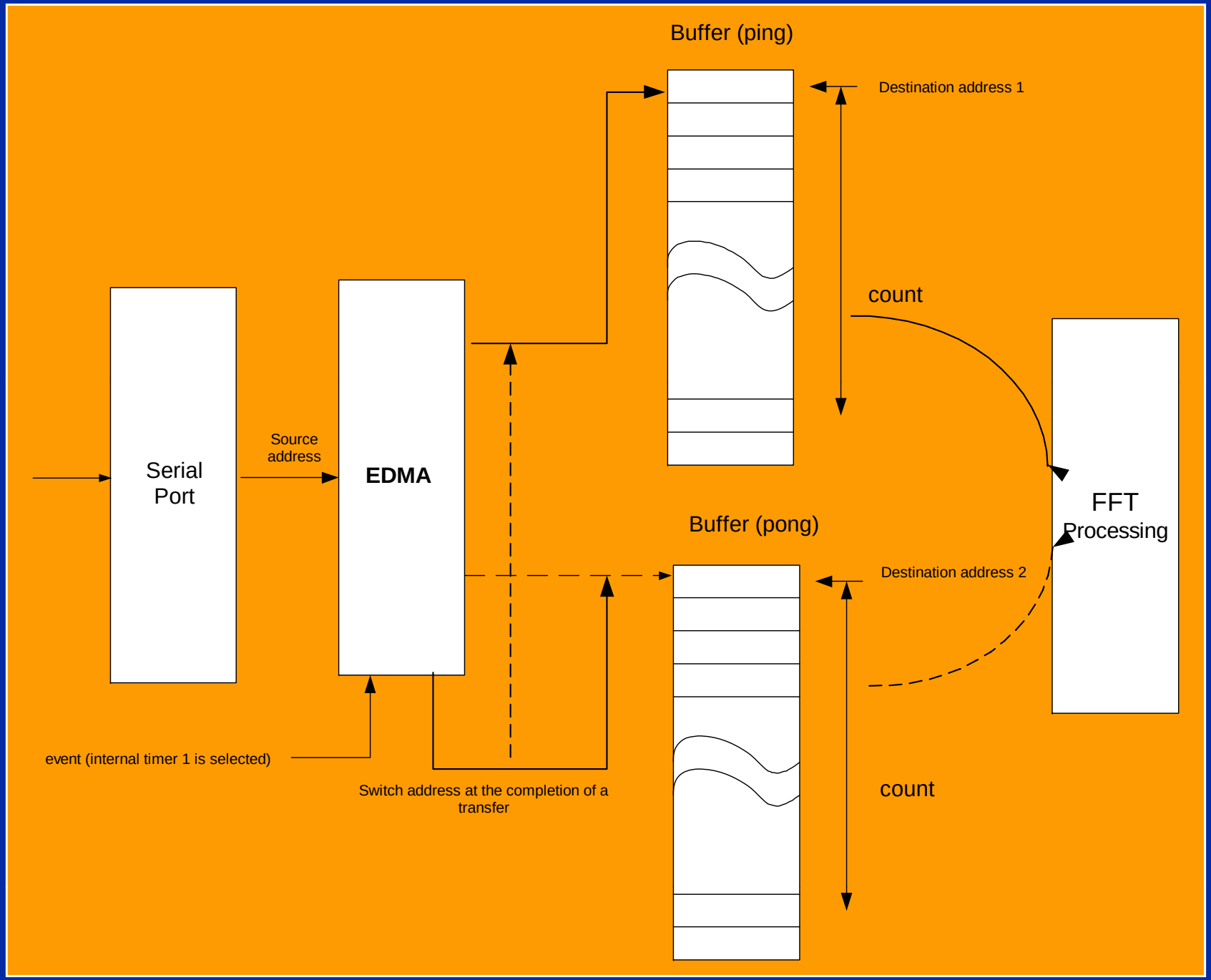
Programming the EDMA - CSL

- The CSL can be divided into four sections:

Constants	Functions	Macros	Structure
EDMA_CHA_CNT	EDMA_allocTable	EDMA_ADDR (<REG>)	EDMA_Config
EDMA_SUPPORT	EDMA_allocTableEx	EDMA_RGET(<REG>)	
EDMA_TABLE_CNT	EDMA_chain	EDMA_RSET(<REG>,X)	
	EDMA_clearChannel	EDMA_FGET(<REG>,<FIELD>)	
	EDMA_clearParm	EDMA_FSET(<REG>,<FIELD>,fieldval)	
	EDMA_close	EDMA_FSETS(<REG>,<FIELD>,<SYM>)	
	EDMA_config	EDMA_RGETA(addr, <REG>)	
	EDMA_configArgs	EDMA_RSETA(addr,<REG>,x)	
	EDMA_disableChaining	EDMA_FGETA(addr,<REG>,<FIELD>)	
	EDMA_enableChaining	EDMA_FSETA(addr,<REG>,<FIELD>,fieldval)	
	EDMA_disableChannel	EDMA_FSETSA(addr,<REG>,<FIELD>,<SYM>)	
	EDMA_enableChannel	EDMA_ADDRH(h,<REG>)	
	EDMA_freeTable	EDMA_RGETH(h,<REG>)	
	EDMA_freeTableEx	EDMA_RSETH(h,<REG>,x)	
	EDMA_getChannel	EDMA_FGETH(h,<REG>,<FIELD>)	
	EDMA_getConfig	EDMA_FSETH(h,<REG>,<FIELD>,fieldval)	
	…		

- See Code Composer Studio help for more details.

Programming the EDMA - CSL Example



Programming the EDMA - CSL Example

- **CSL programming procedure:**
  - **(1) Create handles for the EDMA channel and reload parameters:**

```
EDMA_Handle hEdma;
```

  - **(2) Create the EDMA configuration:**

```
EDMA_Config cfgEdma;
```

Programming the EDMA - CSL Example

- **CSL programming procedure (cont):**
  - **(3) Create the configuration structures for the ping and pong channels:**

```
EDMA_Config cfgEdmaPong = {0x28720002,
                           EDMA_SRC_OF(McBSP0_DRR),
                           EDMA_CNT_OF(BUFF_SZ),
                           EDMA_DST_OF((unsigned int)cin_data),
                           EDMA_IDX_OF(0x00000004),
                           EDMA_RLD_OF(0x00000000)};

EDMA_Config cfgEdmaPing = {0x28720002,
                           EDMA_SRC_OF(McBSP0_DRR),
                           EDMA_CNT_OF(BUFF_SZ),
                           EDMA_DST_OF((unsigned int)in_data),
                           EDMA_IDX_OF(0x00000004),
                           EDMA_RLD_OF(0x00000000)};
```

Programming the EDMA - CSL Example

- **CSL programming procedure (cont):**
  - **(4) Map the event to a physical interrupt (see Interrupt section):**

```
IRQ_map (IRQ_EVT_EDMAINT, 8);
```

This maps the EDMA_INT interrupt to CPU_INT8.

  - **(5) Set the interrupt dispatcher configuration structure (see Interrupt section):**

```
IRQ_configArgs (IRQ_EVT_EDMAINT,
                EdmaIsr,
                0x00000000,
                IRQ_CCMASK_DEFAULT,
                IRQ_IEMASK_ALL);
```

Programming the EDMA - CSL Example

- CSL programming procedure (cont):

(6) Open up an EDMA channel associated with the Timer 1 (remember each EDMA is associated with a specific event):

```
hEdma = EDMA_open (EDMA_CHA_TINT1, EDMA_OPEN_RESET);
```

(7) Allocate the EDMA reload parameters:

```
hEdmaPing = EDMA_allocTable (-1);
hEdmaPong = EDMA_allocTable (-1);
```

Note: -1 means allocate at any available location.

(8) Copy the first reload configuration structure to the EDMA configuration structure:

```
cfgEdma = cfgEdmaPing;
```

Programming the EDMA - CSL Example

- **CSL programming procedure (cont):**

  (9) Configure the link fields of the configuration structure:

```
cfgEdmaPing.rld = EDMA_RLD_RMK(0,hEdmaPong);
cfgEdmaPong.rld = EDMA_RLD_RMK(0,hEdmaPing);
cfgEdma.rld     = EDMA_RLD_RMK(0,hEdmaPong);
```

  This can be done at stage 3 but in this way we do not know the numerical value of the reload address.

  (10) Setup the EDMA channel using the configuration structure:

```
EDMA_config (hEdmaPing, &cfgEdmaPing);
EDMA_config (hEdmaPong, &cfgEdmaPong);
```

Programming the EDMA - CSL Example

- **CSL programming procedure (cont):**

  (11) Finally initialise all the EDMA registers:

```
EDMA_RSET (ECR, 0xffff);      // clear all events
EDMA_enableChannel(hEdma);
EDMA_RSET (EER, 0x4);         // set the timer 1 event enable bit
EDMA_RSET (CIPR, 0xffff);
EDMA_RSET (CIER, 0x4);        // make the timer 1 event generate
                              // an EDMA_INT interrupt
```

- **An example CCS project is included in:**
  - **Code\Chapter 05 - EDMA\Edma_Inout_Staticcfg**

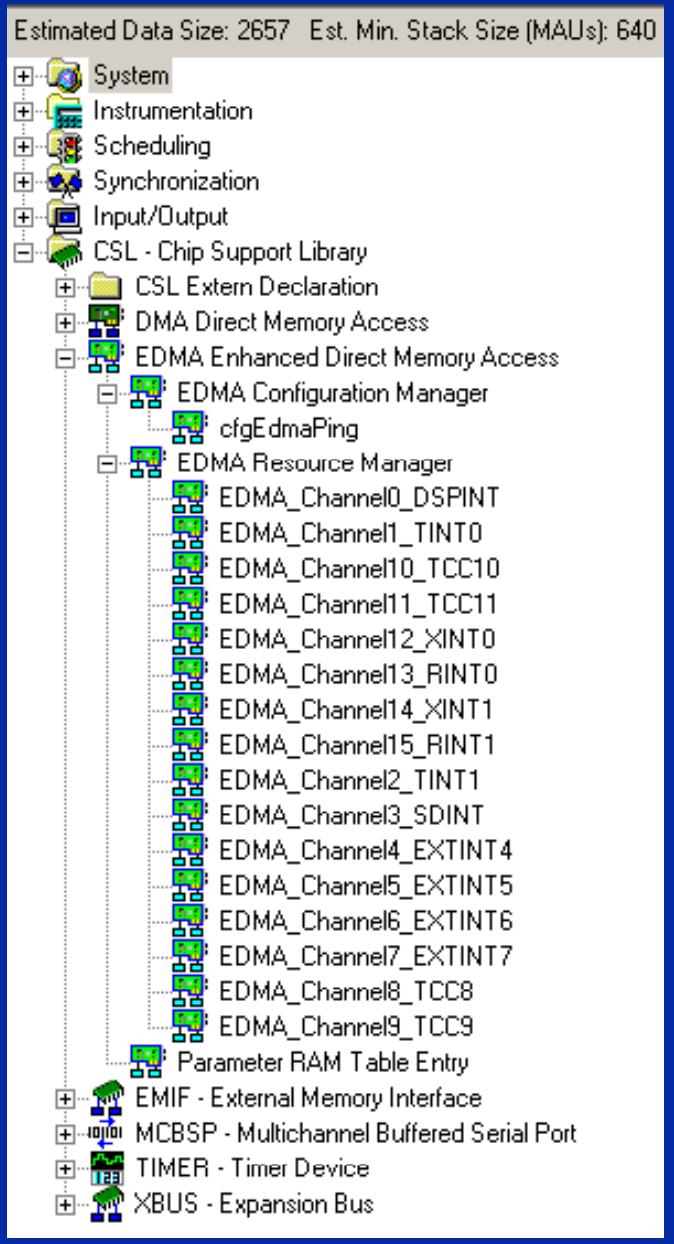
Programming the EDMA - DSP/BIOS GUI

(3) DSP/BIOS GUI Interface

- With this method the configuration structure is created graphically and the setup code is generated automatically.

Programming the EDMA - DSP/BIOS GUI

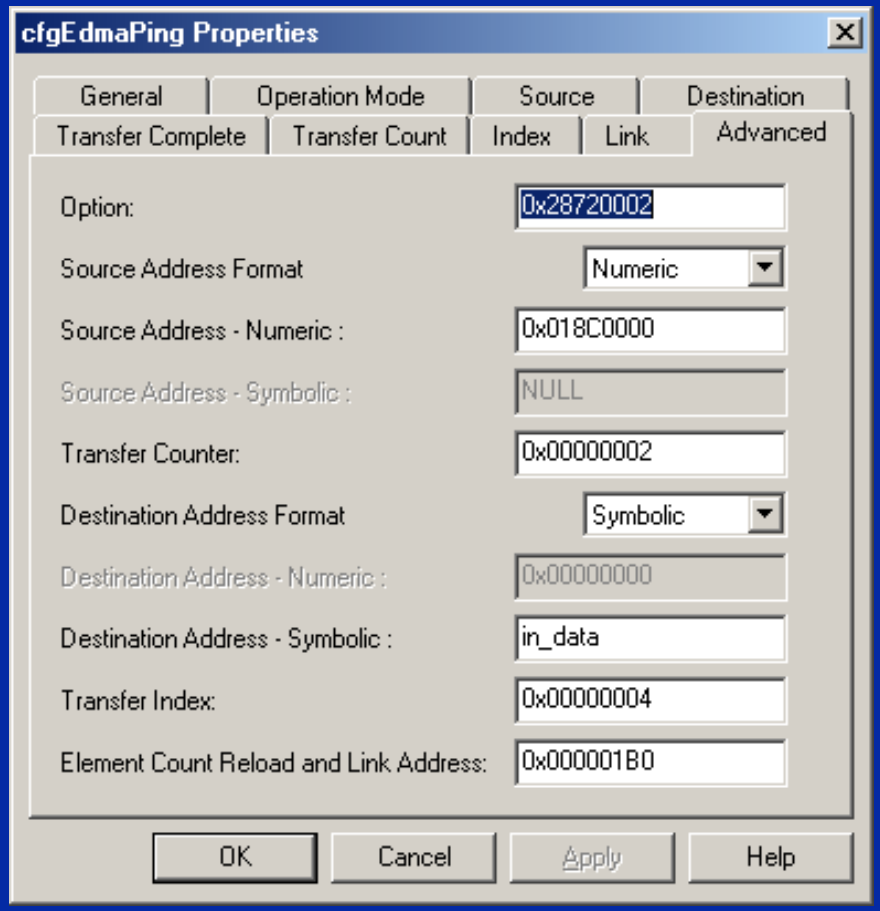
- **Procedure:**
  - **(1) Create a configuration using the EDMA configuration manager.**



Programming the EDMA - DSP/BIOS GUI

- **Procedure:**

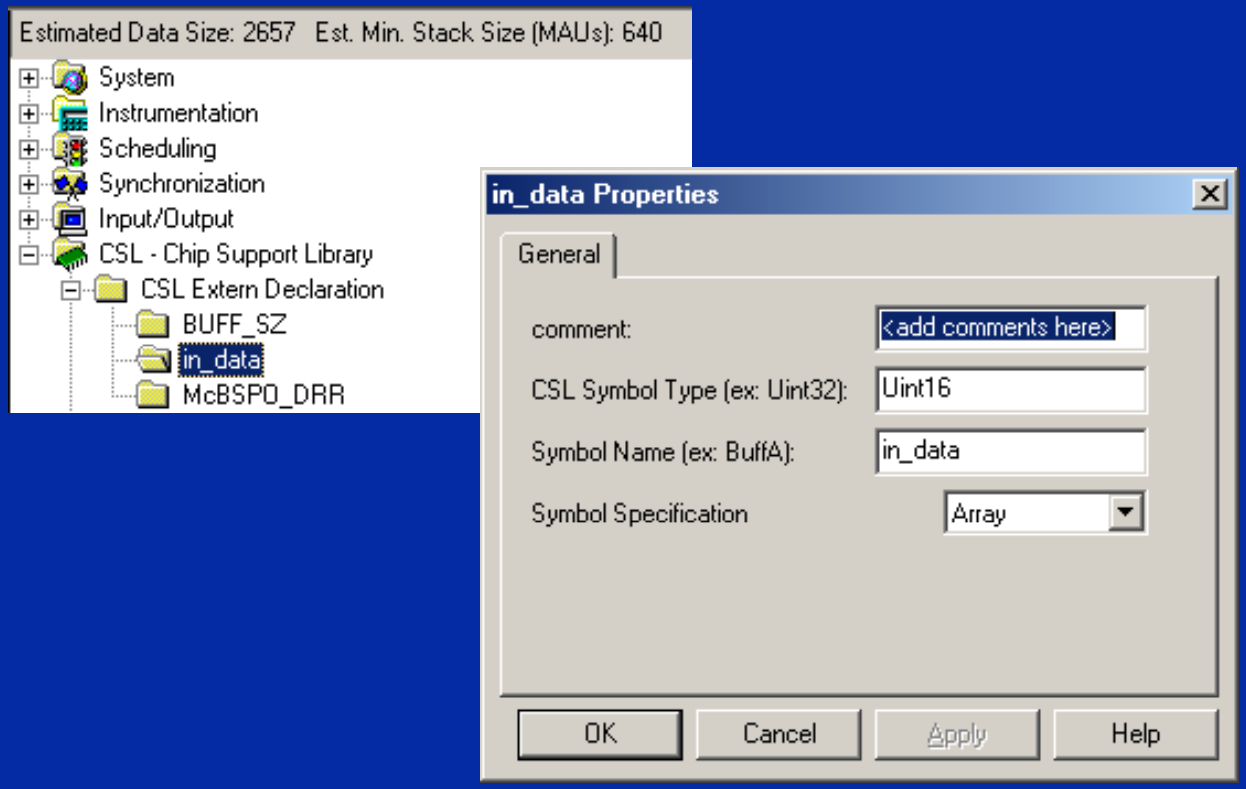
  (2) Right click and select “Properties”, see the figure below, and then select “Advanced” and fill all parameters as shown below.



Programming the EDMA - DSP/BIOS GUI

- **Procedure:**

  (3) If you are using symbolic parameters such as "in_data" you need to declare it in the "CSL Extern Declaration", see below figure.



Programming the EDMA - DSP/BIOS GUI

- **Procedure:**
  - **(4) A file is then generated that contains the configuration code. The file generated for this example is shown on the next slide.**

Programming the EDMA - DSP/BIOS GUI

```
/*   Do *not* directly modify this file.  It was    */
/*   generated by the Configuration Tool; any  */
/*   changes risk being overwritten.                */

/* INPUT edma_inout_csl.cdb */

/*  Include Header File  */
#include "edma_inout_cslcfg.h"

extern far Uint16 McBSP0_DRR;
extern far Uint16 in_data[];
extern far Uint16 BUFF_SZ;

/*  Config Structures */
EDMA_Config cfgEdmaPing = {
    0x28720002,         /*  Option  */
    0x018C0000,         /*  Source Address - Numeric   */
    0x00000002,         /*  Transfer Counter  */
    (Uint32) in_data,   /*  Destination Address - Symbolic   */
    0x00000004,         /*  Transfer Index  */
    0x000001B0          /*  Element Count Reload and Link Address  */
};

/*  Handles  */

/*
 *  ======== CSL_cfgInit() ========
 */
void CSL_cfgInit()
{
}
```

Chapter 5

Enhanced Direct Memory Access (EDMA)

- End -