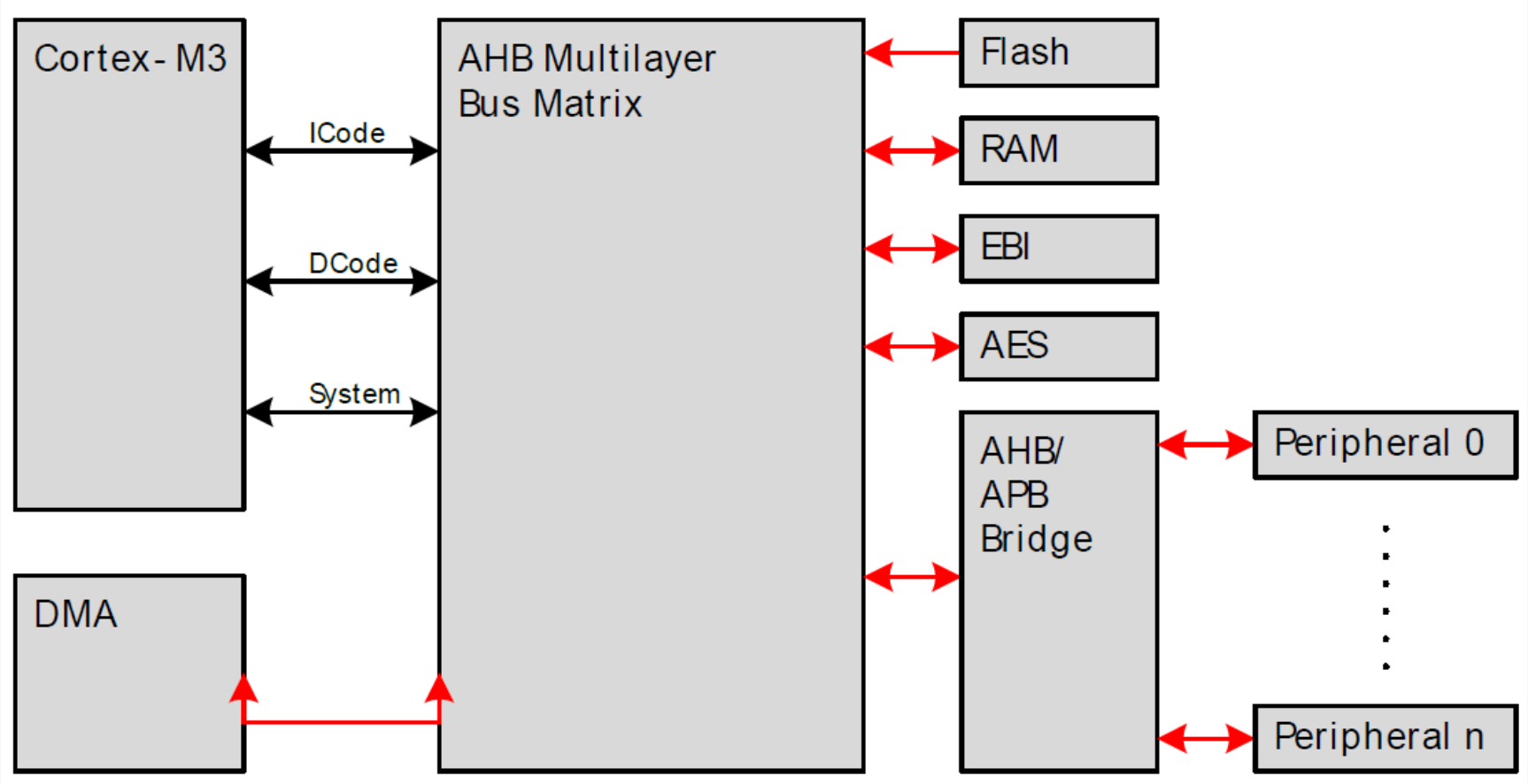
EFM32 Series 0: DMA (ARM PrimeCell µDMA PL230)

EFM32 - DMA

- DMA has read/write access to most of the EFM32 memory map
  - Flash writes can not be done in memory map, but through sequenced writes to peripheral registers
  - RAM and EBI secondary mapping in code space not accessible by DMA



DMA – Configuration Overview

- Registers in source/destination peripherals
  - Configure peripherals to send correct DMA requests and/or behave correctly when a DMA access is performed
- DMA registers
  - Configure trigger sources for each channel
  - Set up interrupts, check status etc.
- Each channel has primary and alternate structure in RAM
  - Source address
  - Destination address
  - DMA cycle number (N)
  - Arbitration rate(2^R)
  - Cycle type
    - Basic
    - Auto
    - Ping-pong
    - Scatter-gather (Memory or peripheral)
  - Data size and increment

Channel Configuration

8.7.29 DMA_CHx_CTRL - Channel Control Register

Offset	Bit Position																													
0x1100	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset											0x00																		0x0		
Access											RW																		RW		
Name											SOURCESEL																		SIGSEL		

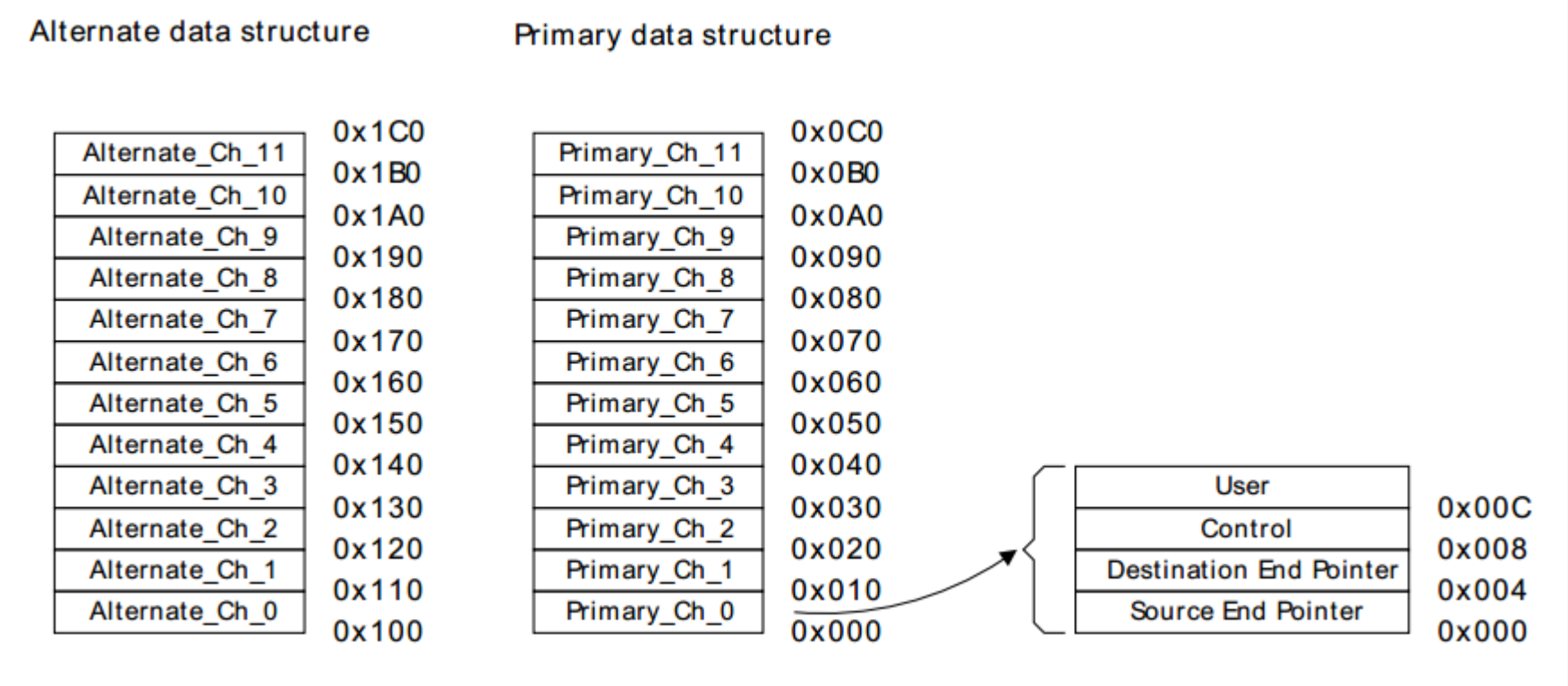
Bit	Name	Reset	Access	Description
31:22	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
21:16	SOURCESEL	0x00	RW	**Source Select**

Select input source to DMA channel.

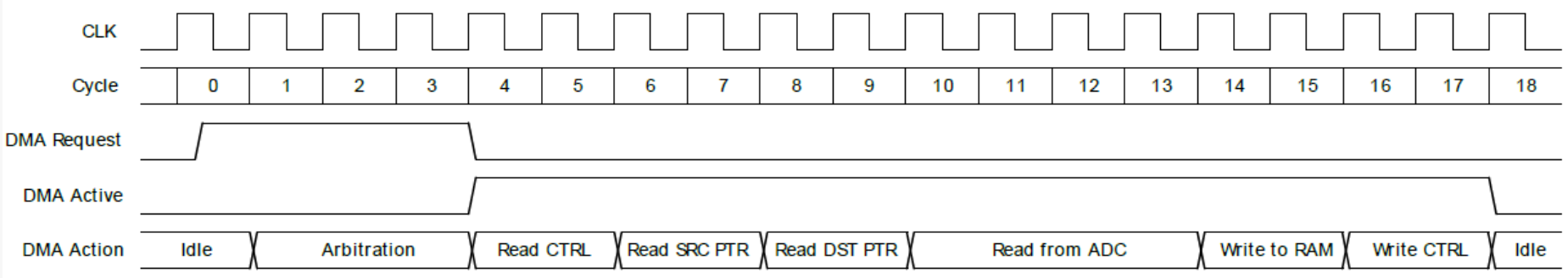
Value	Mode	Description
0b000000	NONE	No source selected
0b001000	ADC0	Analog to Digital Converter 0
0b001010	DAC0	Digital to Analog Converter 0
0b001100	USART0	Universal Synchronous/Asynchronous Receiver/Transmitter 0
0b001101	USART1	Universal Synchronous/Asynchronous Receiver/Transmitter 1
0b001110	USART2	Universal Synchronous/Asynchronous Receiver/Transmitter 2
0b010000	LEUART0	Low Energy UART 0
0b010001	LEUART1	Low Energy UART 1
0b010100	I2C0	I2C 0
0b010101	I2C1	I2C 1
0b011000	TIMER0	Timer 0
0b011001	TIMER1	Timer 1

Channel Descriptor

- Stored in RAM
- Updated for every transfer
- Only need to allocate memory for descriptors that are used



DMA cycle timing

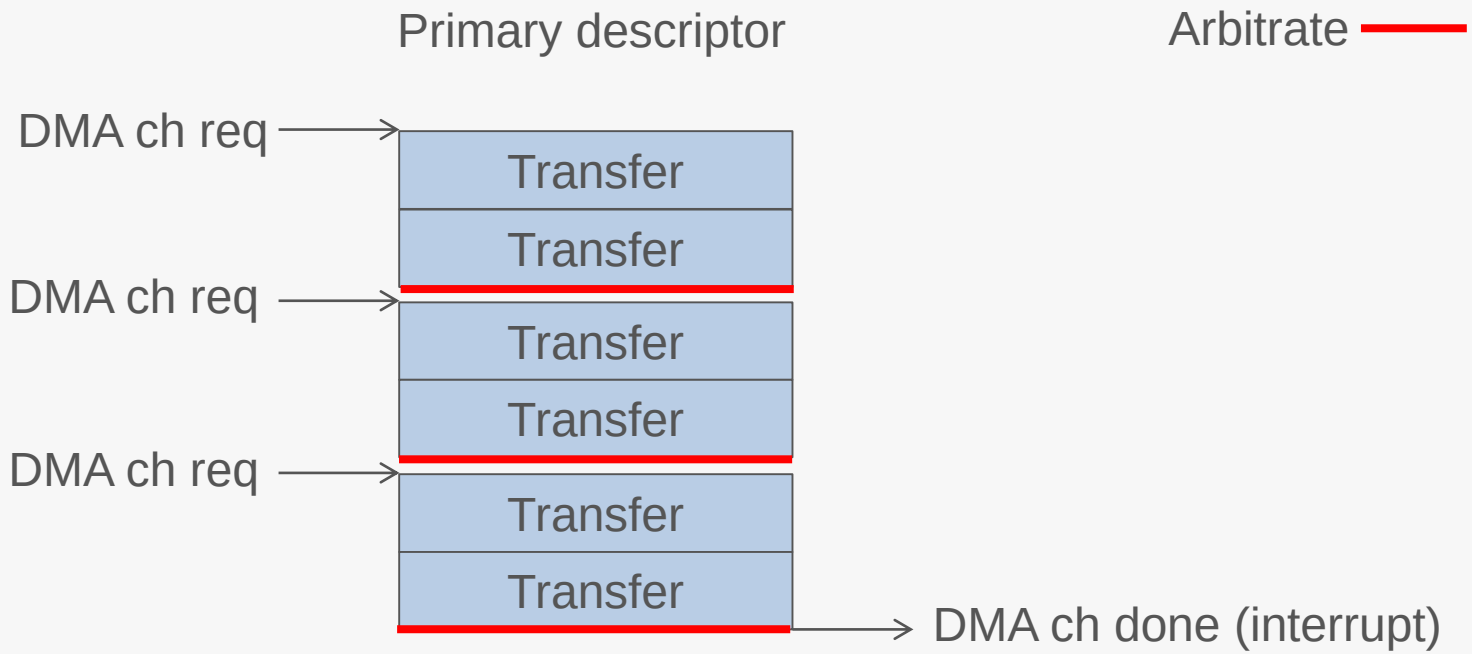


Example DMA timing

- DMA transfer timing affected by CPU activity on the bus
  - Timing of DMA transfers is not predictable
  - Margins in timing must be included to ensure reliable operation
  - Error handlers should be implemented in-case of overflow/underflow
- Bus matrix arbitration settings (only in GG, LG and WG)
  - CPU priority (default): CPU priority (0 wait-state access)
  - DMA priority: DMA priority (0 wait-state access)
  - DMAEM1 priority: DMA priority when sleeping, CPU priority when awake
- Lower numbered DMA channels have priority
  - Priority can be increased individually for each channel

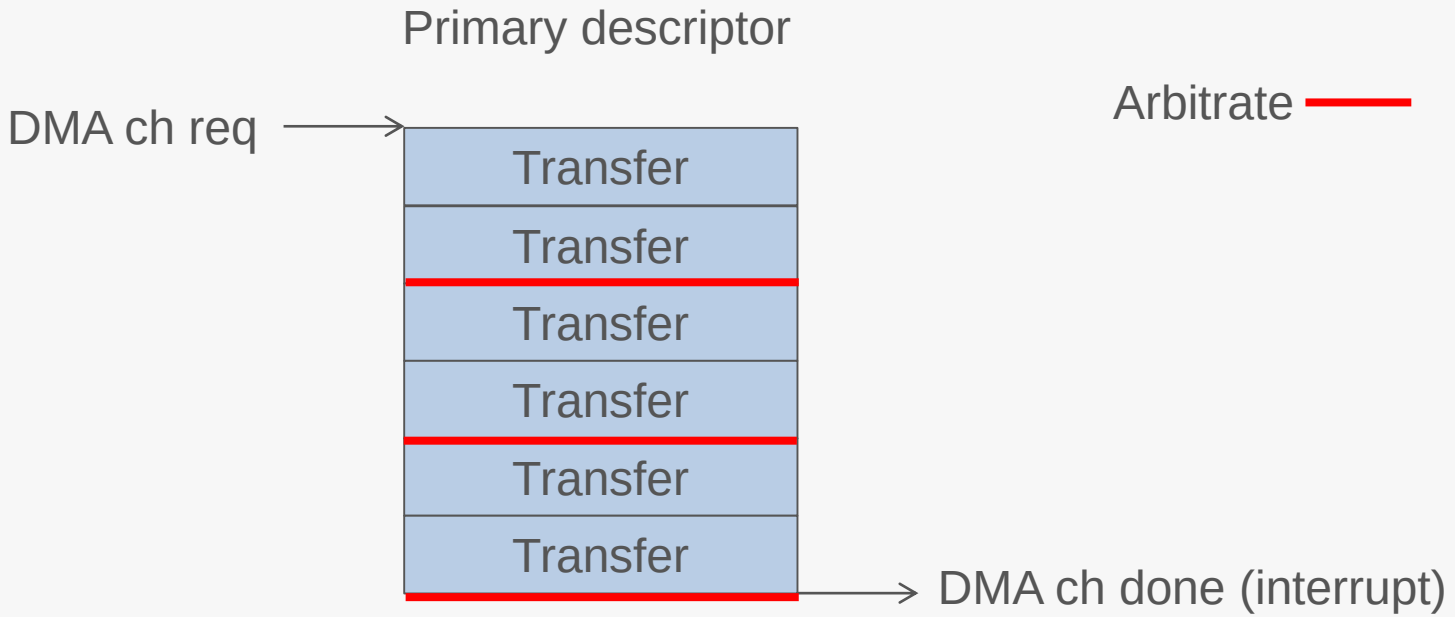
DMA – Basic Mode

- Performs 2^R transfers per request until N transfers have been done. Arbitrates after 2^R
- Normal with 2^R=1 in EFM32
- Example with 2^R=2, N=6:



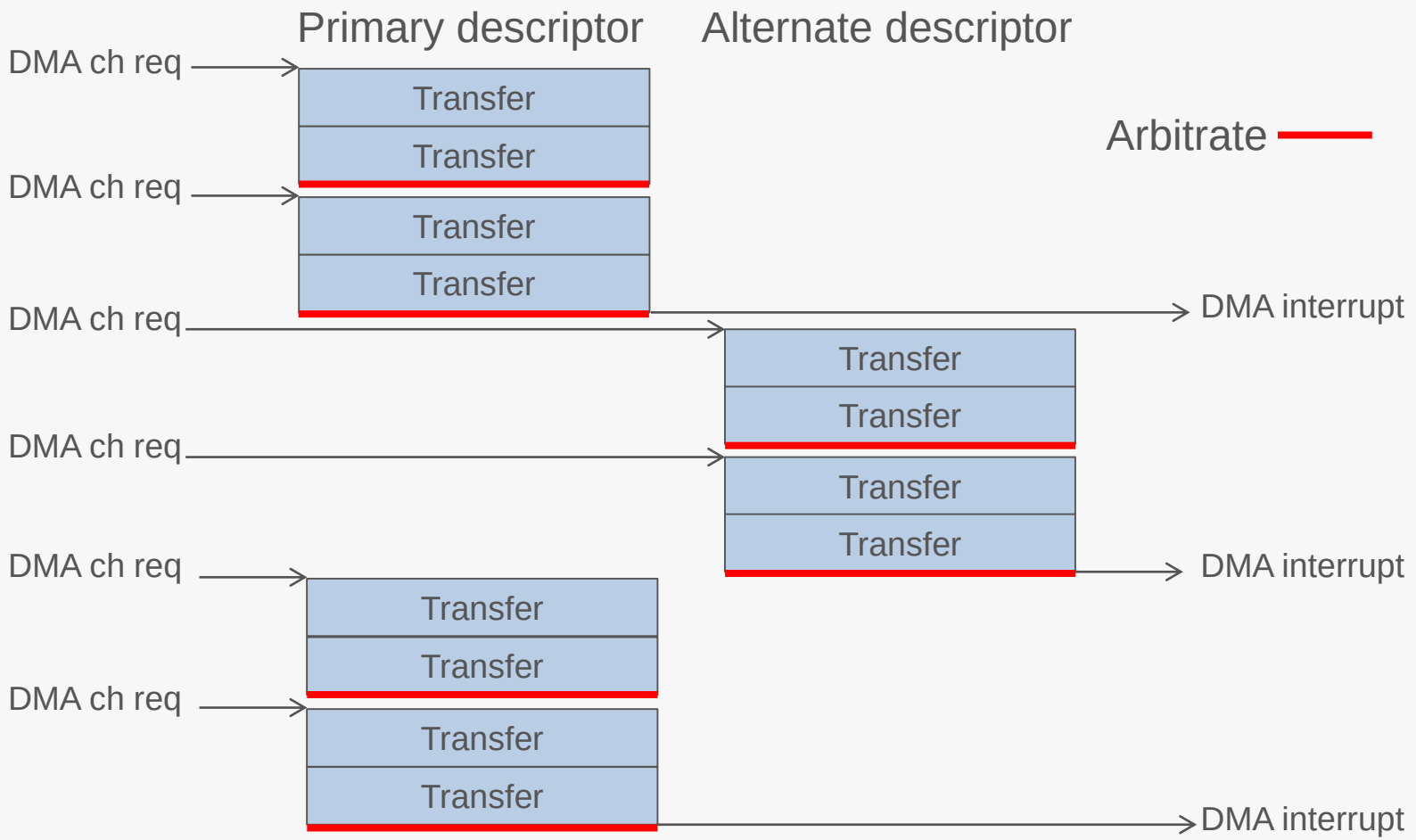
DMA – Auto Mode

- Performs N transfers per request. Arbitrates for every 2^R transfers
- Example with 2^R=2, N=6



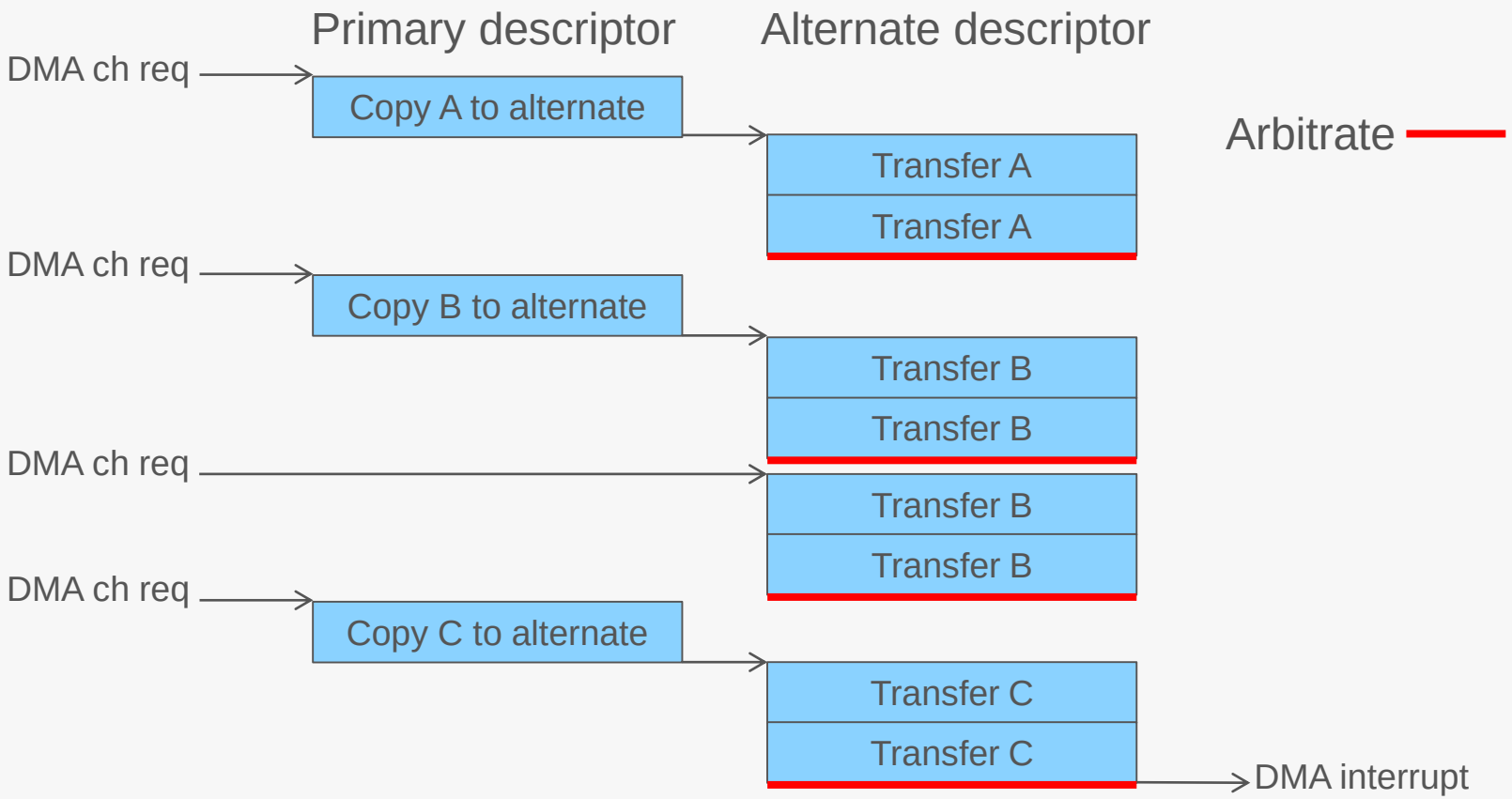
DMA – Ping-pong Mode

- Auto ：Ping-Pong: N requests on Primary, then N requests on Alternate, then Primary... Arbitration for every 2^R transfers
  - Continues as long as CPU reconfigures the descriptors while the other is running
- Example: 2^R=2, N=4:



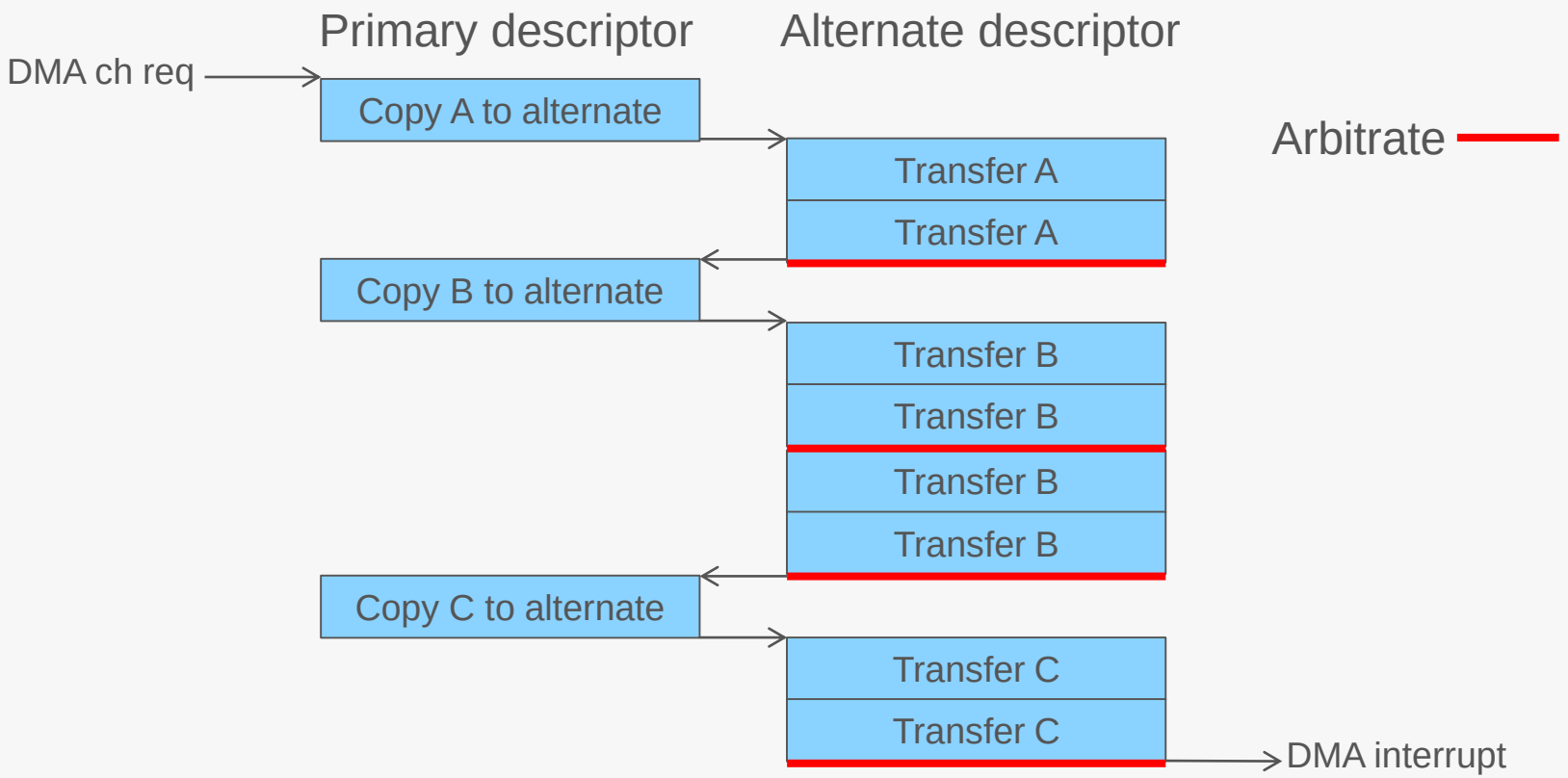
DMA – Peripheral Scatter-gather Mode

- Scather-gather: Uses Primary to modify tasks in Alternate
  - 1) Primary structure copies task to Alternate structure
  - 2) N transfers completed on Alternate
  - 3) If Primary N>0 goto 1)
- One DMA request for each 2^R transfers
- Example: Task A (2^R=2, N=2), Task B (2^R=2, N=4), Task C (2^R=2, N=2)



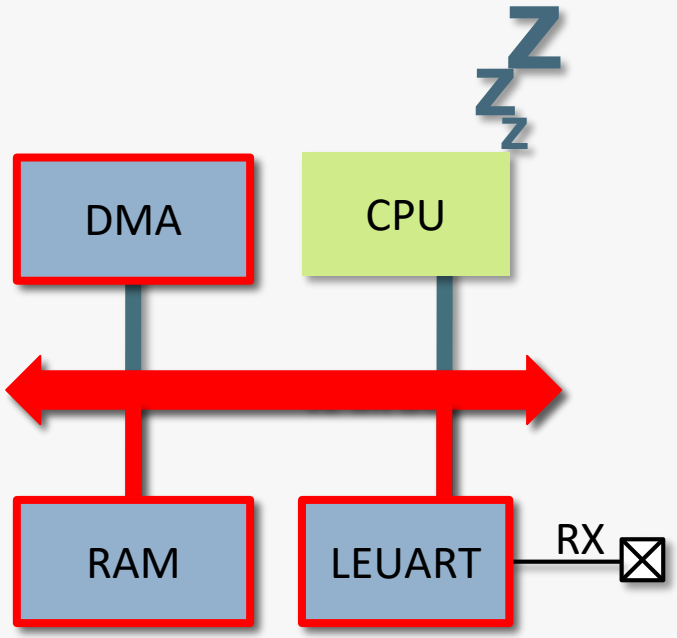
DMA – Memory Scatter-gather Mode

- Scather-gather: Uses Primary to modify tasks in Alternate
  1) Primary structure copies task to Alternate structure
  2) N transfers completed on Alternate
  3) If Primary N>0 goto 1)
- One request initiates whole sequence
- Example: Task A (2^R=2, N=2), Task B (2^R=2, N=4), Task C (2^R=2, N=2)



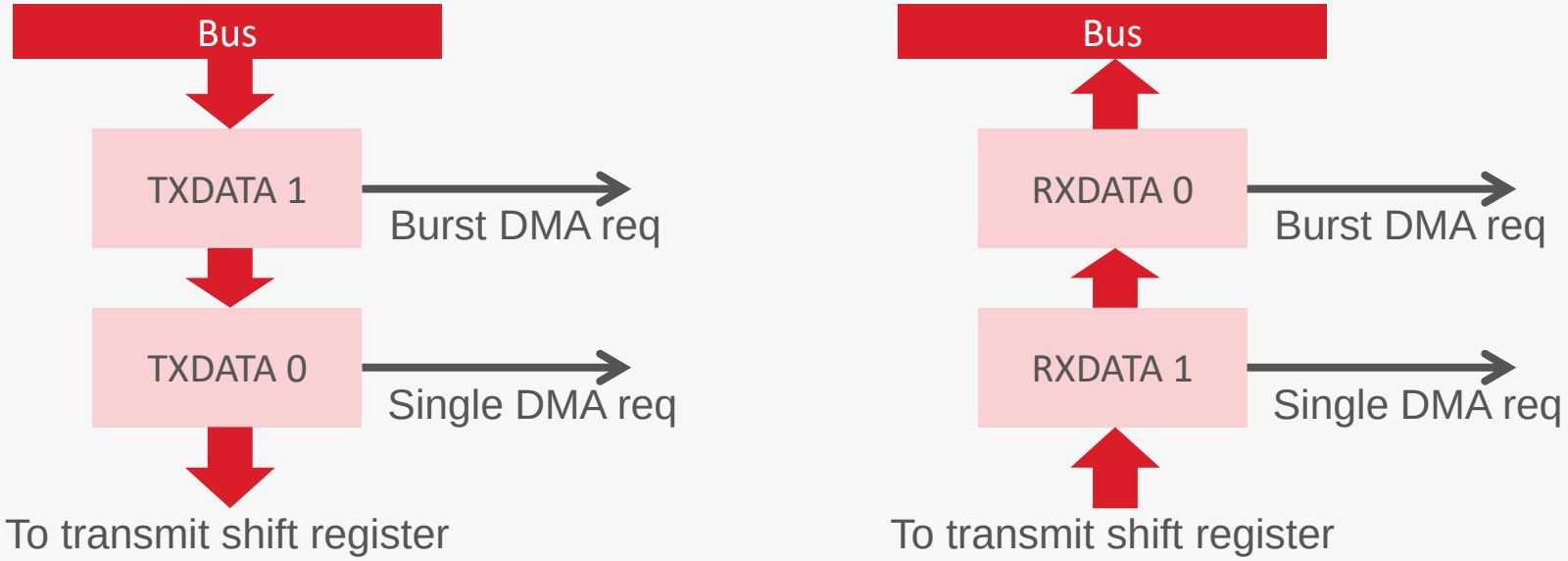
DMA – LEUART/LESENSE DMA in EM2

- LEUART and LESENSE can use DMA from «EM2»
- What is actually happening:
  1. LEUART/LESENSE DMA requests wakes up:
     - Regulated power domain
     - HFRCO
     - Bus system
     - RAM
     - DMA
  2. DMA transfers the data to or from RAM
  3. The additional resources are shut off
- Functionally the same as EM2
  - Other peripherals like TIMERs etc are still clock gated
- Power consumption during DMA transfer the same as EM1
  - Usually short and seldom



DMA – Single vs burst requests

- Most peripherals support only burst DMA requests
  - 2^R transfers done for each request
- UART and USART support combined single and burst DMA requests
  - Single and/or burst requests are sent depending on TX/RX FIFO fill level
  - Default:
    - DMA performs single transfers until TX buffer is full or RX buffer is empty
  - DMA_CHUSEBURSTS = 1
    - DMA waits until TX buffer is full or RX buffer is empty before transferring everything in one burst
    - Both buffer elements can be read/written in one transfer as one combined 32-bit word.

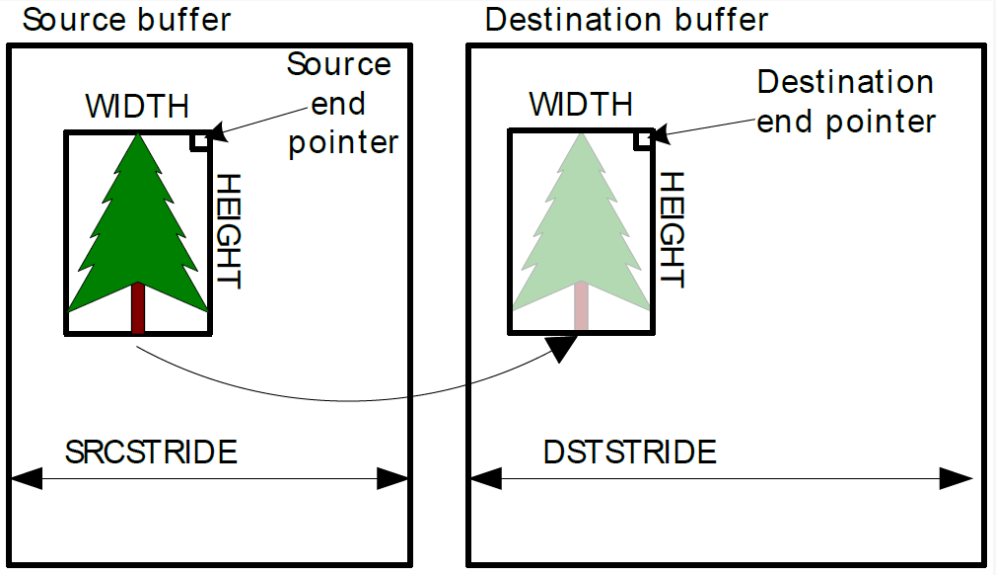


DMA Debugging Techniques

- DMA debugging is challenging for a number of reasons:
  - DMA is not stopped with CPU breakpoints
  - Race conditions easily created between CPU and DMA
  - Bus contention creates non-predictable timing
  - DMA source and destination locations have poor debug visibility
- Useful techniques:
  - Check DMA_CHREQSTATUS to check that DMA requests are set as expected. Do not enable DMA channel for this.
  - Split multi-step DMA transfers to include an intermediate RAM buffer debugging. Use IDE watchpoints to check RAM buffer contents.
  - Use IDE watchpoints to check that DMA descriptors in RAM are updated correctly (especially N value)
  - Trigger DMA request with from SW (DMA_CHSWREQ) to check that DMA channel behaves correctly before enabling peripheral DMA request.
  - Enable peripheral underflow/overflow interrupts to get notified if the DMA is not transferring data fast enough
  - Check DMA_ERR flag to see if DMA is accessing unmapped memory region.
  - Use peripheral PRS signal output to GPIO pins to check timing

DMA Feature Extensions in GG, LG and WG

- Retain descriptor state between transfers
  - Speed up execution of consequtive DMA requests on same channel by not re-loading DMA descriptor from RAM for every DMA request
- Looped transfers for ch 0 and 1
  - Automatic relead of N value a configurable number of times
  - Useful to create a ring buffer
- 2D copy on ch 0
  - Copy of rectangular memory area between memory locations
  - Useful for graphics update



DMA EFM32 Family variations

Family	Channels	Descriptor retention	Looping	2D copy
ZG	4	No	No	No
TG	8	No	No	No
G	8	No	No	No
LG	12	Yes	Yes	Yes
WG	12	Yes	Yes	Yes
GG	12	Yes	Yes	Yes

DMA Configuration with emlib

`DMA_Init()` – Set up common DMA config registers

`DMA_CfgChannel()` – Set up DMA channel config registers

`DMA CfgDescr()/DMA_CfgDescrScatterGather()` – Configure alternate and primary descriptors in RAM

Once channel has been configured it can be activated in desired mode:

```
DMA_ActivateBasic()/* Basic mode */
DMA_ActivateAuto() /* Auto mode */
DMA_ActivatePingPong() /* Ping-pong mode */
DMA_ActivateScatterGather()
DMA_RefreshPingPong()
```

DMA interrupt handler are implemented in emlib, but callbacks can be registered by application

- emlib DMA config includes DMA interrupt handler
  - Callback functions registered during DMA config

Hands-on task 1 - Basic Mode

1. Open an\fae_training\iar\dma.eww and got to adc_basic project
2. Run code and check that DMA->CHREQSTATUS[0] is set to 1
3. Uncomment the DMA_ActivateBasic() function
4. Set a breakpoint in ADC0_IRQHandler()
5. Run code and verify that success is reached without triggering ADC IRQ breakpoint and that ramBufferAdcData is filled with samples
6. Decrease ADC_CLOCK_DIV to 10 and run code
7. Find lowest clock divisor that does not trigger ADC IRQ breakpoint
8. Set compiler optimization to high/speed
9. Find lowest clock divisor that does not trigger ADC IRQ breakpoint

10. Expand example by changing to ping-pong mode

Hands-on task 2 – Ping-pong Mode

1. Open an\fae_training\iar\dma.eww and got to adc_ping_pong project
2. Set a breakpoint in ADC0_IRQHandler()
3. Run code and check that both ramBufferAdcDatax buffers are filled
4. Find lowest clock divisor that does not trigger ADC IRQ breakpoint
5. Set ADC_SAMPLES = 5 and run code. Breakpoint should hit.
6. Find lowest ADC_SAMPLE that does not trigger breakpoint

Examples in DMA appnote

- Flash transfer
  - Memory transfer from flash to RAM using auto mode
- ADC transfer
  - Basic transfer from ADC to RAM
- ADC transfer (ping-pong)
  - Transfer from ADC to RAM using ping-pong
- SPI Master
  - RX and TX DMA channels handling SPI master operation
- SPI TX (ping-pong)
  - Send SPI data at max frequency using when using only TX DMA
- UART RX and TX
  - RX and TX DMA channels handling UART operation
- Scatter-gather transfer
  - 3 transfers between different memory segments executed automatically
- GPIO trigger
  - Transition on GPIO input triggers DMA transfer from RAM to a set of GPIO pins
- I2C master
  - Read and write of an I2C slave using DMA

More information in
AN0013 EFM32 DMA