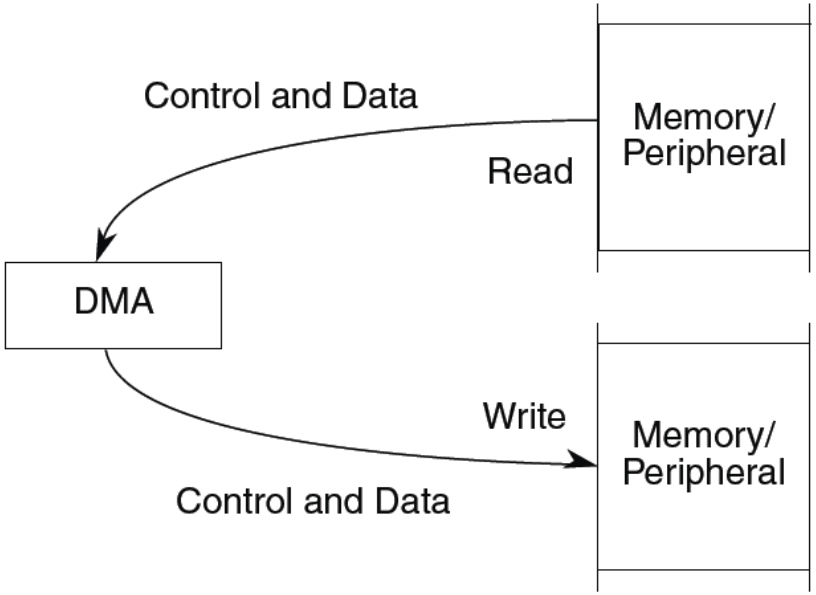
DIRECT MEMORY ACCESS OVERVIEW

Transferring data with DMA

- **For high-bandwidth devices, the transfers consist primarily of relatively large blocks of data**
  - overhead could still be intolerable, since it could consume a large fraction of the processor time
- **direct memory access (DMA) is a mechanism that provides the ability to transfer data directly to/from the memory without involving the processor**
  - Provided by the device controller
  - Provided by the central DMA device
- **DMA device must be a bus master**

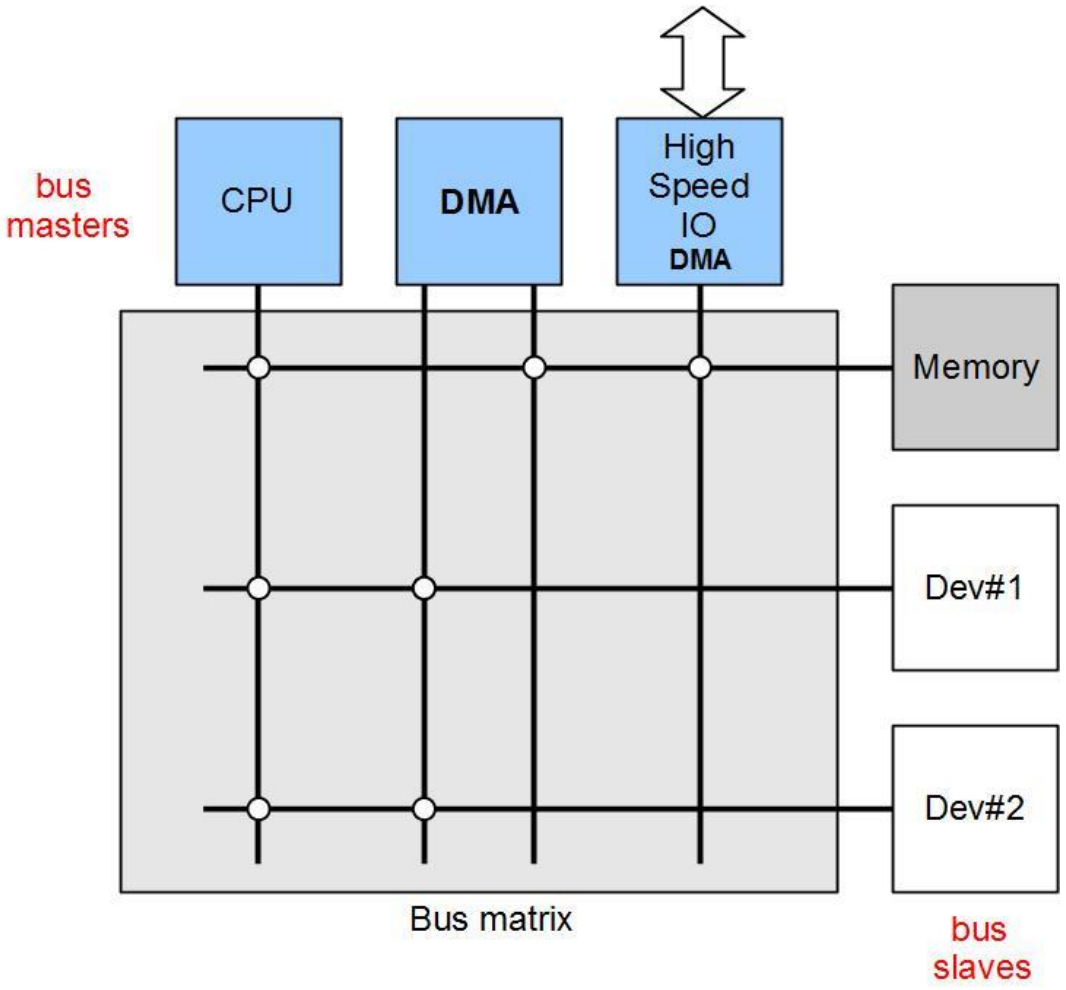
Basic Concepts

- **Special hardware to read data from a source and write it to a destination**
- **Various configurable options**
  - Number of data items to copy
  - Source and destination addresses can be fixed or change (e.g. increment, decrement)
  - Size of data item
  - When transfer starts
- **Operation**
  - Initialization: Configure controller
  - Transfer: Data is copied
  - Termination: Channel indicates transfer has completed



Example System with DMA engine

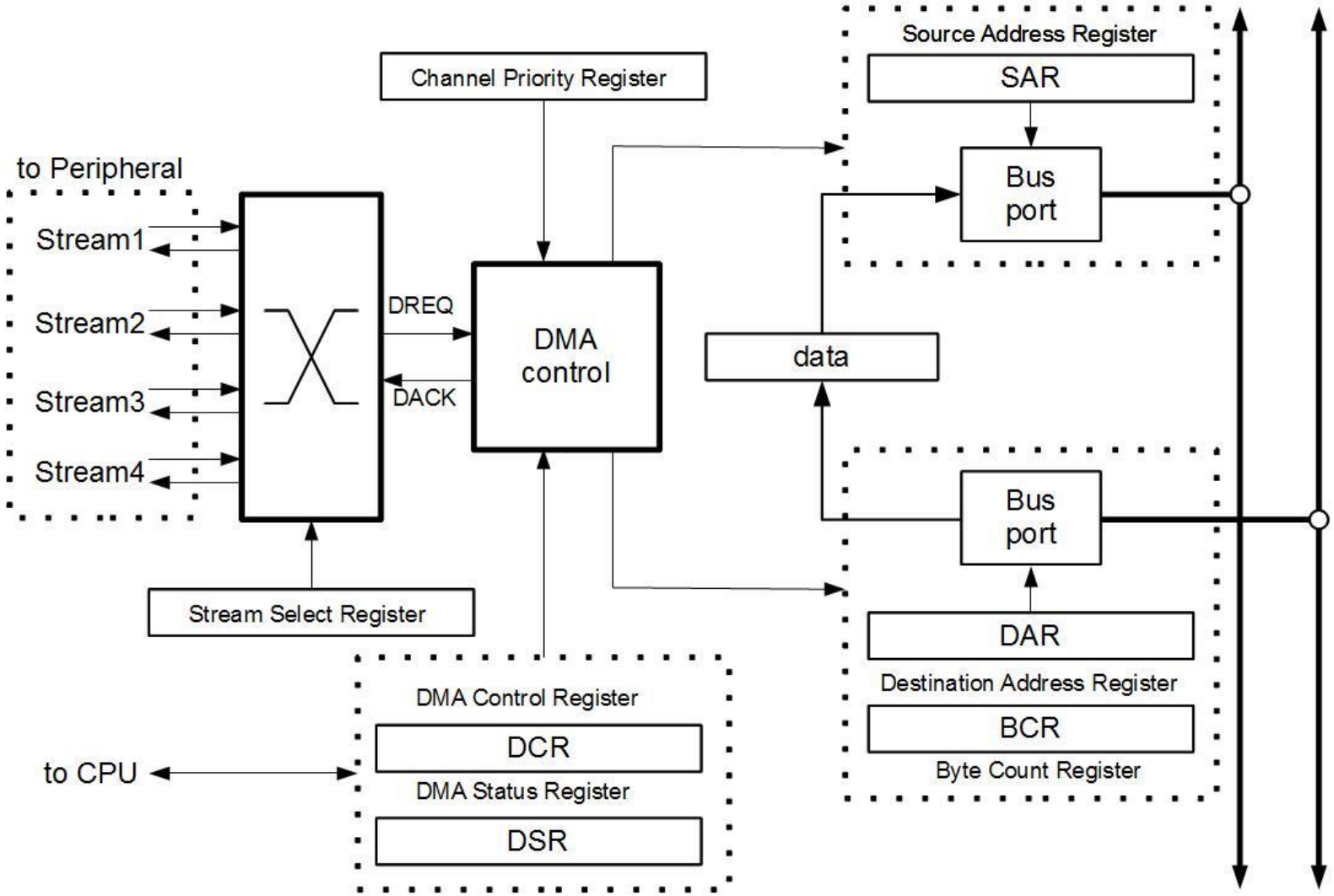
- Central or IO DMA engines are possible



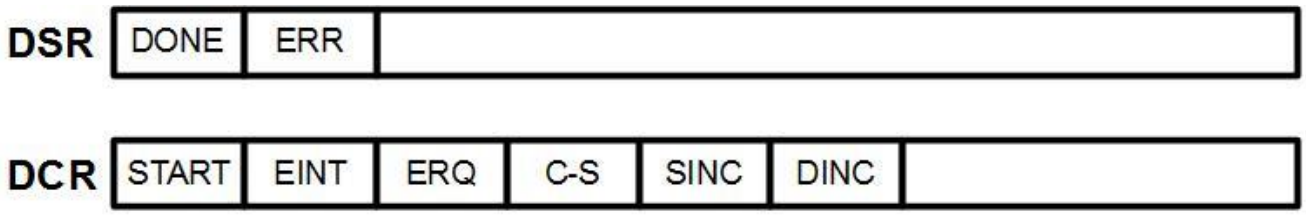
DMA basics

- **Direct memory access (DMA) is used in order to provide high-speed peripherals - memory and memory- memory data transfer.**
- **Each DMA transfer must have Transfer Control Descriptor (TCD) assigned**
- **TCD consist of**
  - **SAR** – Source Address Register, **DAR** - Destination Address Register, **DSR** - Status Register, **BCR** -Byte Count Register
- **DMA controller can provide multiple channels and it can service multiple sources / stereams**
  - Each channel has seperate **TDC**
  - Channels have different priority
  - Each peripheral is connected to DMA module using separate stream
  - Each channel request can be selected among possible stream requests

Central DMA structure example



DMA Status and Control flags



DONE - Transactions done. Set when all DMA controller transactions complete as determined by transfer count
When BCR reaches zero, DONE is set when the final transfer completes successfully.

ERR - Error occured.

START - Start transfer.

EINT - Enable interrupt on completion of transfer.
Determines whether an interrupt is generated by completing a transfer or by the occurrence of an error
Condition.

ERQ - Enable peripheral request.
Enables peripheral request to initiate transfer.

C-S - Cycle steal
0 DMA continuously makes read/write transfers until the BCR decrements to 0.
1 Forces a single read/write transfer per DMA request.

SINC - Source increment
Controls whether the source address increments after each successful transfer.

DINC - Destination increment

DMA procedure

- **The processor sets up the DMA by supplying:**
  - the identity of the device,
  - the operation to perform on the device,
  - the memory address that is the source or destination of the data to be transferred,
  - and the number of bytes to transfer
- **The DMA starts the operation on the device and arbitrates for the bus.**
  - DMA unit can complete an entire transfer, which may be thousands of bytes in length, without bothering the processor
- **Once the DMA transfer is complete, the controller interrupts the processor**

Transfer Requests

- **The DMA channel supports software-initiated or peripheral-initiated requests.**
  - A software request is issued by setting **DCR:START**
  - Peripheral request are initiated by asserting DMA Request (DREQ) signal when **DCR:ERQ** is set. Setting **DCR:ERQ** enables recognition of the peripheral DMA.
- **The hardware can be programmed to automatically clear DCR:ERQ, disabling the peripheral request, when BCR reaches zero.**

Cycle-steal and continuous modes

- **Cycle-steal mode (DCR:CS = 1)**
  - Only one complete transfer from source to destination occurs for each request.
- **Continuous mode (DCR:CS = 0)**
  - After a software-initiated or peripheral request, the DMA continuously transfers data until BCR reaches zero.
  - The DMA performs the specified number of transfers, then retires the channel.

Channel prioritization

- **Many DMA channels can be prioritized based on number, with channel 0 having the highest priority and the last channel having the lowest priority.**
- **Another scenario can assume the priority register to allow a programmers to set channel priorities according their need.**
  - Priorities between DMA stream requests are software-programmable (e.g. 4 levels consisting of very high, high, medium, low) or hardware in case of equality (request 0 has priority over request 1, etc.)

DMA termination

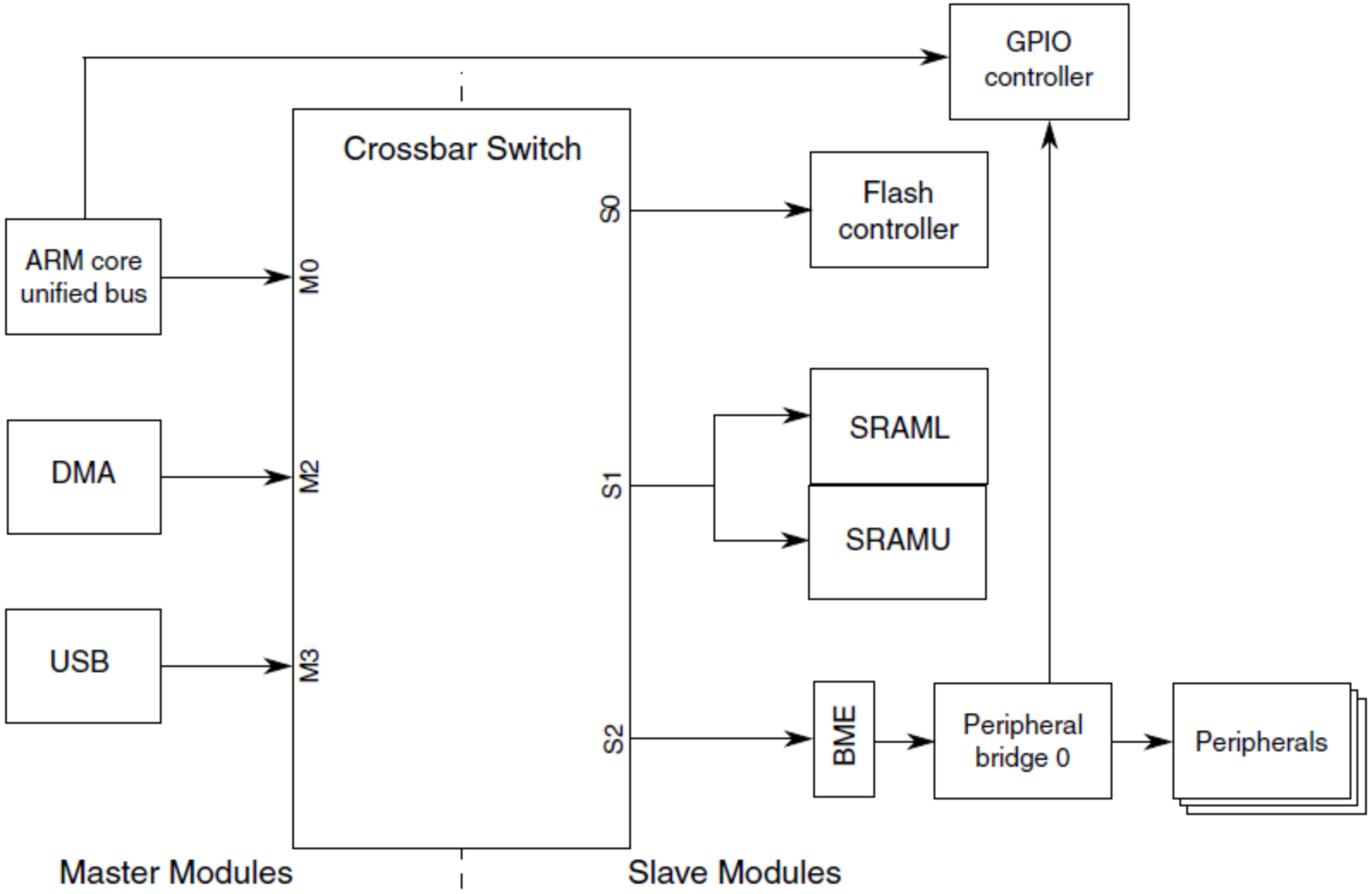
- **Interrupt**
  - If DCR:EINT is set, the DMA drives the appropriate interrupt request signal. The processor can read DSR to determine whether the transfer terminated successfully or with an error.

DMA MULTIPEXER AND MODULE IN KINETIS L

DMA Controller Features

- **4 independent channels**
  - Channel 0 has highest priority
- **8-, 16- or 32-bit transfers, data size can differ between source and destination**
- **Modulo addressable**
- **Can trigger with hardware signal or software**
- **Can run continuously or periodically (“cycle-stealing”)**
- **Hardware acknowledge/done signal**

Kinetis L family Crossbar Switch



DMA request multiplexer

- **The direct memory access multiplexer (DMAMUX) routes DMA sources, called slots, to any of the four DMA channels**
  - allows up to 63 DMA request signals

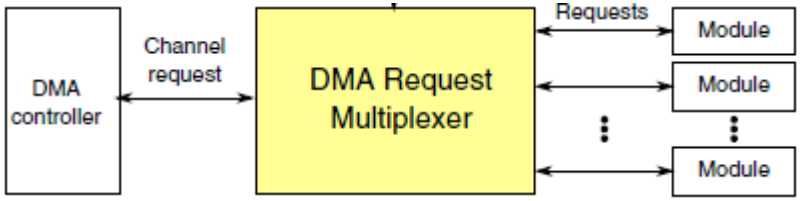
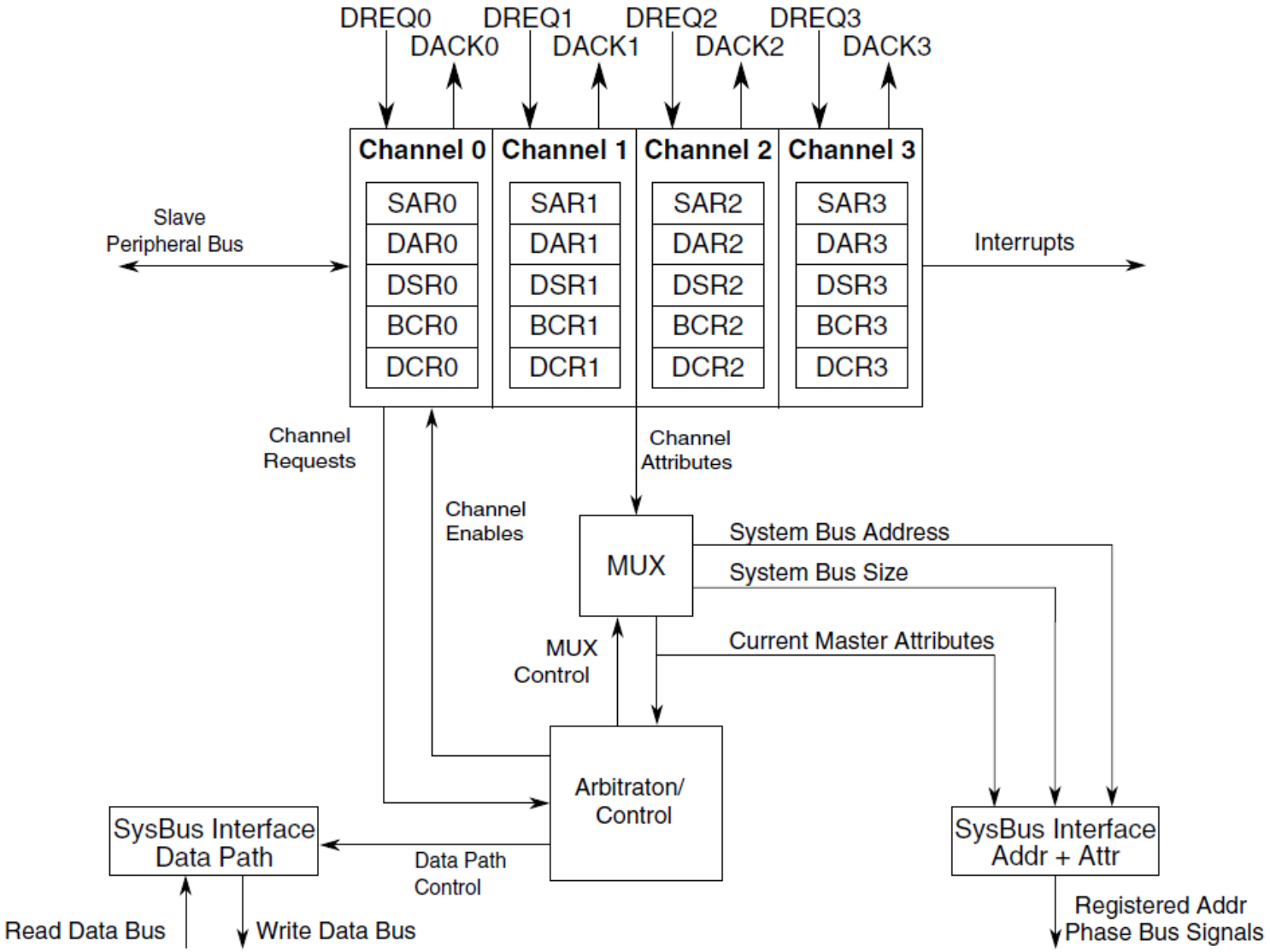


Table 3-20. DMA request sources - MUX 0

Source number	Source module	Source description	Async DMA capable
0	—	Channel disabled[1]	
1	Reserved	Not used	
2	UART0	Receive	Yes
3	UART0	Transmit	Yes
4	UART1	Receive	
5	UART1	Transmit	
6	UART2	Receive	
7	UART2	Transmit	

Table continues ...

Kinteis L DMA Controller

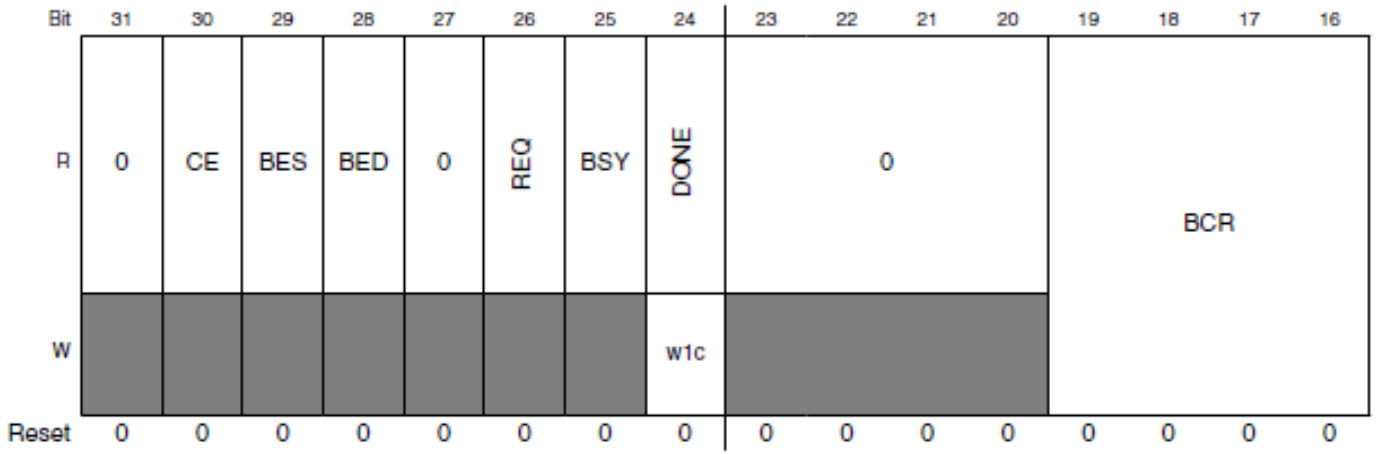


Registers

- **DMA_SARn**
  - Source address register,
  - Valid values 0 to 0x000f ffff

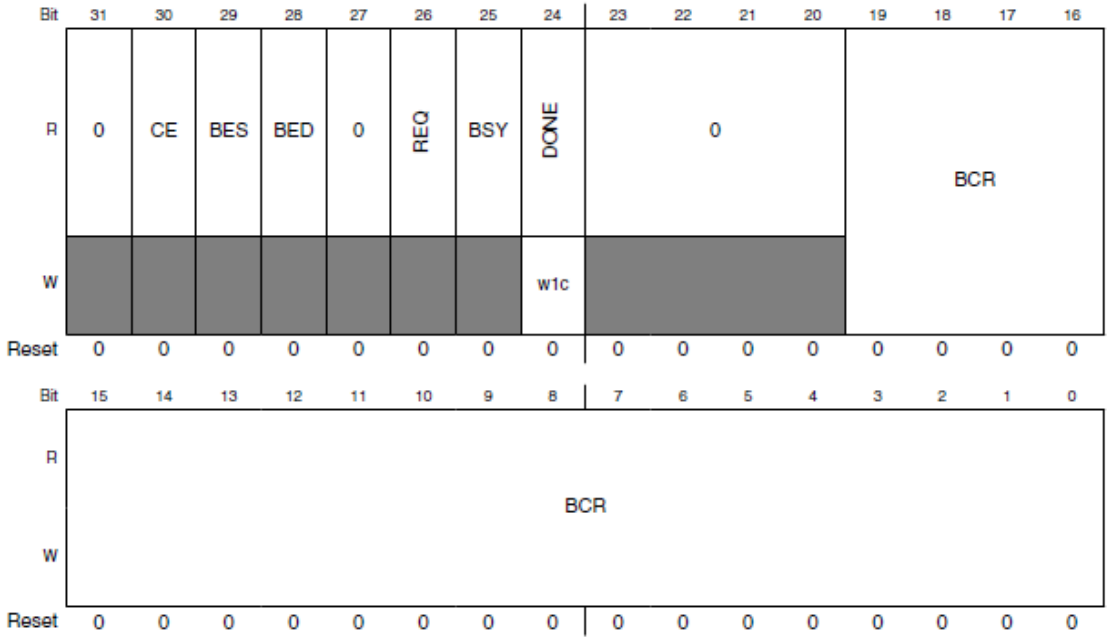
- **DMA_DARn**
  - Destination address register
  - Valid values 0 to 0x000f ffff

Status Register/Byte Count Register DMA_DSR_BCRn



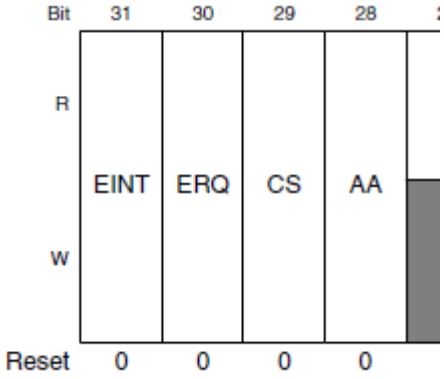
- **Status flags: 1 indicates error**
  - CE: Configuration error
  - BES: Bus error on source
  - BED: Bus error on destination
  - REQ: A transfer request is pending (more transfers to perform)
  - BSY: DMA channel is busy
  - DONE: Channel transfers have completed or an error occurred. Clear this bit in an ISR.

Byte Count Register



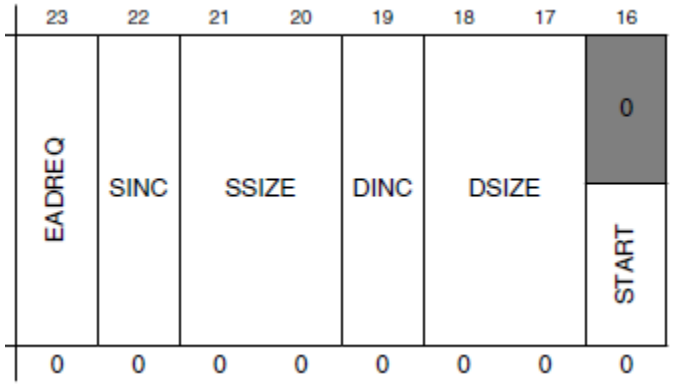
- **BCR: Bytes remaining to transfer**
- **Decremented by 1, 2 or 4 after completing write (determined by destination data size)**

DMA Control Register (DMA_DCRn)



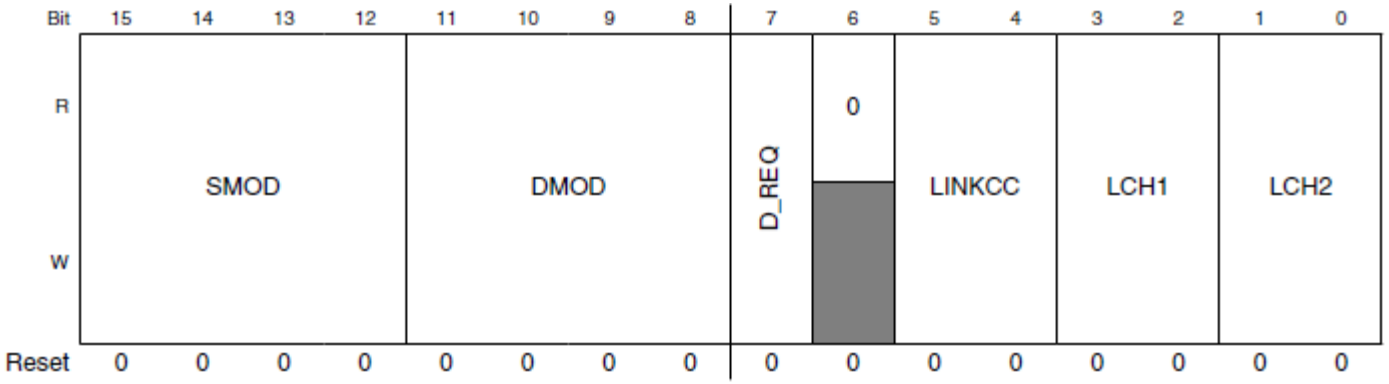
- **EINT: Enable interrupt on transfer completion**
- **ERQ: Enable peripheral request to start transfer**
- **CS: Cycle steal**
  - 0: Greedy - DMA makes continuous transfers until BCR == 0
  - 1: DMA shares bus, performs only one transfer per request
- **AA: Autoalign**

DMA Control Register (DMA_DCRn)



- **EADRQ – Enable asynchronous DMA requests when MCU is in Stop mode**
- **SINC/DINC – Increment SAR/DAR (by 1,2 or 4 based on SSIZE/DSIZE)**
- **SSIZE/DSIZE – Source/Destination data size.**
  - Don't need to match – controller will perform extra reads or writes as needed (e.g. read one word, write two bytes).
  - 00: longword (32 bits)
  - 01: byte (8 bits)
  - 10: word (16 bits)
- **START – Write 1 to start transfer**

DMA Control Register (DMA_DCRn)



- **SMOD, DMOD – Source/Destination address modulo**
  - When non-zero, supports circular data buffer – address wraps around after 2^{n+3} bytes (16 bytes to 64 kilobytes)
  - When zero, circular buffer is disabled
- **D_REQ: If 1, then when BCR reaches zero ERQ bit will be cleared**
- **LINKCC: Enables this channel to trigger another channel**
  - 00: Disabled
  - 01: Two stages:
    - Link to channel LCH1 after each cycle-steal transfer
    - Link to channel LCH2 after BCR reaches 0
  - 10: Link to channel LCH1 after each cycle-steal transfer
  - 11: Link to channel LCH1 after BCR reaches 0
- **LCH1, LCH2: Values 00 to 11 specify linked DMA channel (0 to 3)**

Basic Use of DMA

- **Enable clock to DMA module (in SIM register SCGC7)**
- **Initialize control registers**
- **Load SARn with source address**
- **Load DARn with destination address**
- **Load BCRn with number of bytes to transfer**
- **Clear DSRn[DONE]**
- **Start transfer by setting DCRn[START]**
- **Wait for end of transfer**
  - Interrupt generated if DCRn[EINT] is set (DMAn_IRQHandler)
  - Poll DSRn[DONE]

Demonstration: Memory Copy

- **Software-triggered**
- **void Copy_Longwords(uint32_t * source, uint32_t * dest, uint32_t count)**

- **Could use as a fast version of memcpy function, but need to handle all cases**
  - Alignment of source and destination
  - Data size
  - Detecting overlapping buffers

Memory to memory DMA Initialization

```
#include <stdint.h>
#include <MKL25Z4.h>

void Init_DMA_To_Copy(void) {
  SIM->SCGC7 |= SIM_SCGC7_DMA_MASK;
  DMA0->DMA[0].DCR =   DMA_DCR_SINC_MASK |
                       DMA_DCR_SSIZE(0) |
                       DMA_DCR_DINC_MASK |
                       DMA_DCR_DSIZE(0);
            // Size: 0 = longword, 1 = byte, 2 = word

}
```

Memory to memory copy using DMA

```
void Copy_Longwords( uint32_t *source,
                     uint32_t * dest,
                     uint32_t count) {
  // initialize source and destination pointers
  DMA0->DMA[0].SAR = DMA_SAR_SAR((uint32_t) source);
  DMA0->DMA[0].DAR = DMA_DAR_DAR((uint32_t) dest);
  // byte count
  DMA0->DMA[0].DSR_BCR = DMA_DSR_BCR_BCR(count*4);
  // verify done flag is cleared
  DMA0->DMA[0].DSR_BCR &= ~DMA_DSR_BCR_DONE_MASK;

  // start transfer
  DMA0->DMA[0].DCR |= DMA_DCR_START_MASK;
  // wait until it is done
  while (!(DMA0->DMA[0].DSR_BCR & DMA_DSR_BCR_DONE_MASK));
}
```