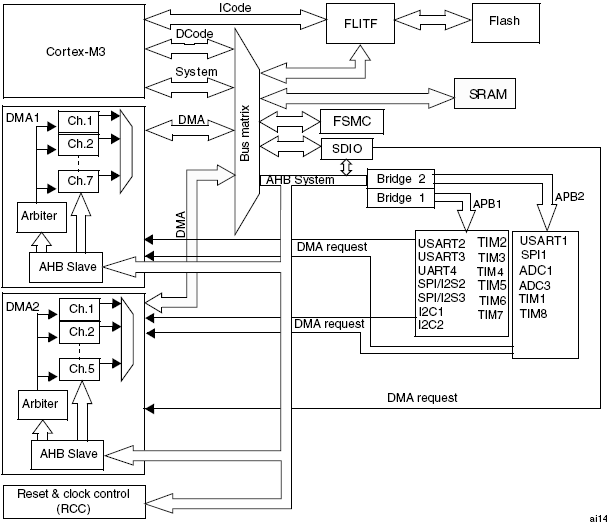
**Introduction**

* high-speed data transfer between peripherals and memory & memory to memory without CPU action
* two DMA controllers with 12 channels (7 for DMA1 & 5 for DMA2)
* arbiter for handling priority between DMA requests
* 12- independently configurable channels/requests
* each of 12 channels is connected to dedicated hardware DMA request
* software trigger is also supported on each channel
* 4-level (very high, high, medium) programmable priorities between request
* Independent source and destination transfer size
* circular buffer management
* 3 event flags are ORed to provide single interrupt request for each channel
  + DMA Half Transfer
  + DMA Transfer complete
  + DMA Transfer Error
* Memory-to-Memory, Memory-to-peripheral, Peripheral-to-memory, peripheral-to-peripheral
* programmable number of data transfers – up to 65536



**Functional Description**

* DMA performs direct memory transfer by sharing the system bus with cortex M3 core
* DMA request can stop CPU access to system bus for some bus cycles if the CPU and DMA are targeting the same destination

**DMA transactions**

* After an event, the peripheral sends request signals to DMA controller
* DMA controller serves the request depending on channel priority
* Acknowledge is sent to peripheral by DMA controller as soon as DMA controller access the peripheral
* Peripheral release its request as soon as it gets the acknowledge from DMA controller
* When request is DE asserted by the peripheral, the DMA controller releases the Acknowledge

**DMA transfer**

1. Loading of data from peripheral data register or location in memory
   * addressed through internal current peripheral/memory address register
   * start address – base peripheral/memory address programmed in ***DMA\_CPARx*** or ***DMA\_CMARx*** register
2. Storing of data loaded to peripheral data register or location in memory
   * addressed through internal current peripheral/memory address register
   * start address – base peripheral/memory address programmed in ***DMA\_CPARx*** or ***DMA\_CMARx*** register
3. post-decrement of ***DMA\_CNDTRx*** register – number of transaction to be performed

**Arbiter**

* Manages channel request based on their priority and launches peripheral/memory access sequence.
* Priorities are managed in two stages.
  + **Software -** each channel priority can be configured in ***DMA\_CCRx*** register into four levels
  + **Hardware –** if two request have same software priority level, then channel with the lowest number will get priority
* **DMA1** has higher priority over **DMA2**

**DMA channels**

* each channel can handle DMA transfer between peripheral register located at fixed address and memory address
* register which contains the amount of data items to be transferred is decremented after each transaction
* Transfer data sizes of peripheral and memory are selected by ***PSIZE*** and ***MSIZE*** bits in register ***DMA\_CCRx***
* Peripheral and memory pointes can be optionally be automatically post-increment after each transaction depending on ***PINC*** and ***MINC*** bits in ***DMA\_CCRx*** registers.
* In **increment** mode, the address of next transfer will be address of previous one increment by 1,2,4 depending on chosen data size
* first data transfer address is in ***DMA\_CPARx/DMA\_CMARx***
* In **non-circular** mode, no DMA request is served after last transfer.
* the DMA channel must be disabled to give new number of data items in ***DMA\_CNDTRx*** register
* In **circular mode**, DMA\_CNDTRx register is auto reloaded with initially programmed value

**Circular Mode**

* handle circular buffers and continuous data flows – ADC scan mode
* enabled by ***CIRC*** bit in ***DMA\_CCRx***
* when activated the number of data to be transferred in automatically reloaded to initially programmed value
* the DMA requests continue to be served

**Memory-to-Memory Mode**

* **DMA** can work without trigger by request from peripheral --- called Memory to memory mode
* ***MEM2MEM*** bit in ***DMA\_CCRx*** is set
* transaction is initiated as soon as ***EN*** bit in ***DMA\_CCRx***  is set
* and transfer stops when ***DMA\_CNDTRx*** register becomes zero
* Can’t be used in combination with circular mode

**Channel configuration procedure**

1. Set peripheral register address in ***DMA\_CPARx*** registers – the data will be moved from/to this address to/from memory after peripheral event.
2. Set memory address in ***DMA\_CMARx*** register – the data will be written to or read from this memory after peripheral event
3. Configure total number of data to be transferred in ***DMA\_CNDTRx*** register – after each peripheral event, this value is decremented
4. Configure the channel priority using ***PL[1:0]*** bits in ***DMA\_CCRx***
5. Configure in ***DMA\_CCRx***
   1. **Data** **transfer direction**
   2. **Circular mode**
   3. **Peripheral & memory incremented mode**
   4. **Peripheral & memory data size**
   5. **Interrupt after half and/or full transfer**
6. Activate channel by setting ***ENABLE*** bit in ***DMA\_CCRx***

**Interrupts**

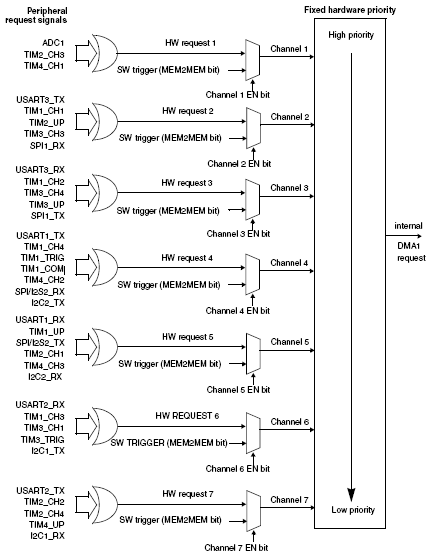
* Interrupt can be on Half-Transfer, Transfer Complete or transfer Error for each DMA channel

|  |  |  |
| --- | --- | --- |
| **Interrupt event** | **Event flag** | **Enable Control bit** |
| Half-transfer | HTIF | HTIE |
| Transfer Complete | TCIF | TCIE |
| Transfer Error | TEIF | TEIE |

**DMA Request Mapping**

**DMA1 controller**

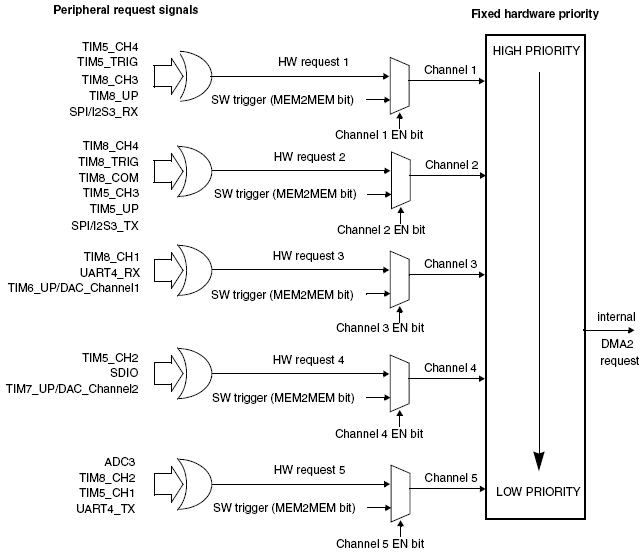
* the 7 request from peripherals are simply logically ORed – meaning one request must be enabled
* **TIMx[1,2,3,4], ADC1, SPI1, SPI/I2S2, I2Cx[1,2], USARTx[1,2,3]**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Peripherals** | **C1** | **C2** | **C3** | **C4** | **C5** | **C6** | **C7** |
| **ADC1** | ADC1 | - | - | - | - | - | - |
| **SPI/I2S** | - | SPI  RX | SPI1  TX | SPI2/I2S2  RX | SPI2/I2S2  TX | - | - |
| **USART** | - | USART3\_TX | USART3\_RX | USART1  TX | USART1  RX | USART2\_RX | USART2  TX |
| **I2C** | - | - | - | I2C2 TX | I2C2 RX | I2C1 TX | I2C1 RX |
| **TIM1** | - | TIM1  CH1 | - | TIM1 CH4 TIM1 TRIG TIM1 COM | TIM1 UP | TIM1  CH3 | - |
| **TIM2** | TIM  CH3 | TIM2  UP | - | - | TIM2  CH1 | - | TIM2 CH2 TIM2 CH4 |
| **TIM3** | - | TIM3  CH3 | TIM3  CH4 TIM3  UP | - | - | TIM3  CH1 TIM3  TRIG | - |
| **TIM4** | TIM4\_CH1 | - | - | TIM4 CH2 | TIM4  CH3 | - | TIM4  UP |

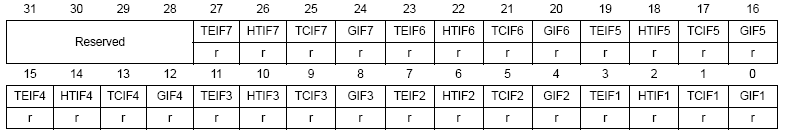
**DMA2 controller**

* the 5 request from peripherals are simply logically ORed – meaning one request must be enabled
* **TIMx[5,6,7,8], ADC3, SPI/I2S3, UART4, DAC\_Channel[1,2] SDIO**

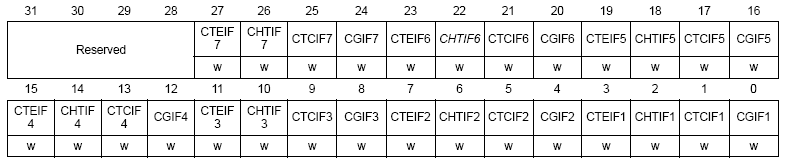


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Peripherals** | **Channel 1** | **Channel 2** | **Channel 3** | **Channel 4** | **Channel 5** |
| ADC3 | - | - | - | - | ADC3 |
| SPI/I2S3 | SPI/I2S3\_RX | SPI/I2S3 TX | - | - | - |
| UART4 | - | - | UART4 RX | - | UART4 TX |
| SDIO | - | - | - | SDIO | - |
| TIM5 | TIM5 CH4 TIM5 TRIG | TIM5CH3 TIM5 UP | - | TIM5 CH2 | TIM5 CH1 |
| TIM6/ DAC1 | - | - | TIM6 UP/ DAC1 | - | - |
| TIM7 | - | - | - | TIM7 UP/ DAC2 | - |
| TIM8 | TIM8 CH3 TIM8 UP | TIM8 CH4 TIM8 TRIG TIM8 COM | TIM8 CH1 | - | TIM8 CH2 |

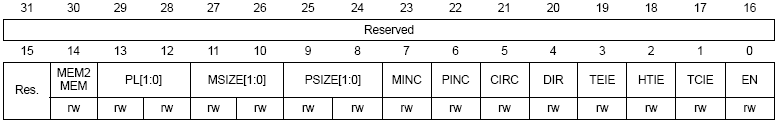
**DMA registers**

**DMA interrupt status register (DMA\_ISR)**

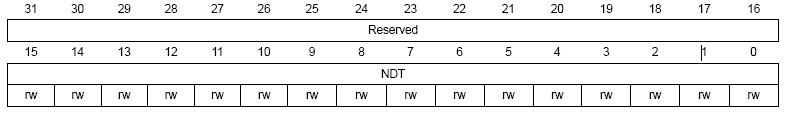
|  |  |  |  |
| --- | --- | --- | --- |
| **TEIFx** | **HTIFx** | **TCIFx** | **GIFx** |
| Channel x transfer error flag | Channel x half transfer flag | Channel x transfer complete flag | Channel x global interrupt flag |
| Set by hardware cleared by software by writing 1 in DMA\_IFCR | Set by hardware cleared by software by writing 1 in DMA\_IFCR | Set by hardware cleared by software by writing 1 in DMA\_IFCR | Set by hardware cleared by software by writing 1 in DMA\_IFCR |
| 1 – Transfer Error (TE) | 1 – Half transfer Event (HT) | 1 – Transfer complete event (TC) | 1 – TE, HT, TC event |

**DMA interrupt flag clear register (DMA\_IFCR)**

|  |  |  |  |
| --- | --- | --- | --- |
| **CTEIFx** | **CHTIFx** | **CTCIFx** | **CGIFx** |
| transfer error flag clear | half transfer flag clear | transfer complete flag clear | global interrupt flag clear |
| 1 – clear TEIF flag | 1 – clear HTIF flag | 1 – clear TCIF flag | 1 - clear GIF,TEIF, HTIF, TCIF flag |

**DMA channel x configuration register (DMA\_CCRx) (x = 1...7, x =Channel number)**

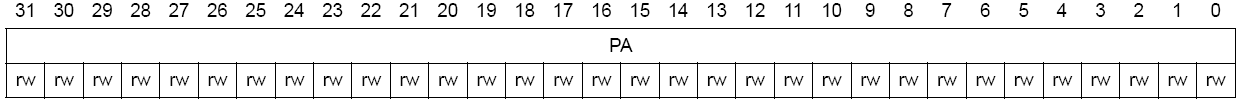
|  |  |
| --- | --- |
| **Bits** | **Description** |
| **MEM2MEM - Memory to memory mode** | 1 – enable Memory to Memory mode |
| **PL[1:0] – Channel Priority level** | 00 – low, 01 – medium  10 – high, 11 – very high |
| **MSIZE[1:0] – Memory size** | 00 – 8-bits, 01 – 16-bits, 10 – 32bits |
| **PSIZE[1:0] – Peripheral size** | 00 – 8-bits, 01 – 16-bits, 10 – 32bits |
| **MINC – Memory increment mode** | 1 – enable Memory increment mode |
| **PINC – Peripheral increment mode** | 1 – enable Peripheral increment mode |
| **CIRC – Circular Mode** | 1 – enable Circular Mode |
| **DIR – Data transfer direction** | 0 – read from peripheral  1- read from memory |
| **TEIE – Transfer error interrupt enable** | 1 – TE interrupt enabled |
| **HTIE – Half Transfer complete interrupt enable** | 1 – HT interrupt enabled |
| **TEIE – Transfer complete interrupt enable** | 1 – TC interrupt enabled |
| **EN – Channel enable** | 1 – Channel enabled |

**DMA channel x number of data register (DMA\_CNDTRx) (x = 1...7, where x = channel number)**

* Number of data to be transferred (0 up to 65535).
* can be written only when the channel is disabled
* register decrements after each DMA transfer
* channel becomes zeros or reloaded depending on mode

**DMA channel x peripheral address register (DMA\_CPARx) (x = 1..7, where x = channel number)**

* Base address of peripheral data register from/to which data will be read/written



**DMA channel x memory address register (DMA\_CMARx) (x = 1..7, where x = channel number)**

* Base address of memory area from/to which data will be read/written

