Section 5. Direct Memory Access

High Speed and Direct Data Transfer Between Memory and Peripherals

Basic Concepts of DMA

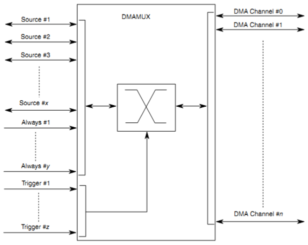
- Limitations of Interrupt Processing
  - CPU involvements
  - Good for discrete events with small amount of data
  - Inefficient for large data transfers
- Needs fo High Speed Data Transfer between
  - disk and RAM;
  - NIC and RAM;
  - more
- An analogous example: Program of Study
  - Student asks dean for advising and signature → Request
  - Dean directs student to the advisor → address, tasks, and go
  - Student talks with the advisor → communication/data tranfser
  - Advisor signed the program of study after completing advising and send student back o dean for final signature→ report completion

General Procedure of DMA

- DMA Request
  - Request from peripheral through hardware
  - Explicit software initiation
  - Channel to channel linking for continual transfer
- Source/Destination and amount of Data Transfer
  - CPU write registers in DMA controller to define
    - Source address, destination address, and byte count
- Direct and Continuous Data Transfer
  - Data transfer is done directly between memory and peripheral device without CPU involvement
- Report Completion
  - When transfer is done, reporting completion through interrupt

Introduction of DMAMUX

The DMA Mux routes up to 63 DMA sources (called slots) to be mapped to any of the 16 DMA channels. This is illustrated in the following figure.



Features and Modes of Operation

The DMA channel MUX provides these features:

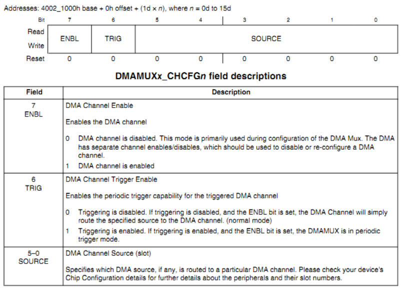
- 52 peripheral slots + 10 always-on slots can be routed to 16 channels.
- 16 independently selectable DMA channel routers.
  - The first 4 channels additionally provide a trigger functionality.
- Each channel router can be assigned to one of the 52 possible peripheral DMA slots or to one of the 10 always-on slots.

Operating modes :

- Disabled mode
  - The DMA channel is disabled, used mainly as the reset state for a DMA channel in the DMA channel MUX
- Normal mode
  - A DMA source (such as DSPI transmit or DSPI receive) is routed directly to the specified DMA channel.
- Periodic trigger mode
  - A DMA source may only request a DMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically.

Channel Configuration Register (DMAMUXx_CHCFGn)

- Enable/Disabled and associated with DMA slots



Addresses: 4002_1000h base + 0h offset + (1d × *n*), where *n* = 0d to 15d

Bit	7	6	5	4	3	2	1	0
Read Write	ENBL	TRIG	SOURCE					
Reset	0	0	0	0	0	0	0	0

DMAMUXx_CHCFG*n* field descriptions

Field	Description
7 ENBL	DMA Channel Enable Enables the DMA channel 0 DMA channel is disabled. This mode is primarily used during configuration of the DMA Mux. The DMA has separate channel enables/disables, which should be used to disable or re-configure a DMA channel. 1 DMA channel is enabled
6 TRIG	DMA Channel Trigger Enable Enables the periodic trigger capability for the triggered DMA channel 0 Triggering is disabled. If triggering is disabled, and the ENBL bit is set, the DMA Channel will simply route the specified source to the DMA channel. (normal mode) 1 Triggering is enabled. If triggering is enabled, and the ENBL bit is set, the DMAMUX is in periodic trigger mode.
5–0 SOURCE	DMA Channel Source (slot) Specifies which DMA source, if any, is routed to a particular DMA channel. Please check your device's Chip Configuration details for further details about the peripherals and their slot numbers.

Enabling and configuring sources

Enabling a source with periodic triggering

1. Determine with which DMA channel the source will be associated. Note that only the first 4 DMA channels have periodic triggering capability
2. Clear the CHCFG[ENBL] and CHCFG[TRIG] bits of the DMA channel
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point
4. Configure the corresponding timer
5. Select the source to be routed to the DMA channel. Write to the corresponding CHCFG register, ensuring that the CHCFG[ENBL] and CHCFG[TRIG] bits are set

Example:

Configure source #5 transmit for use with DMA channel 2, with periodic triggering capability

1. Write 0x00 to CHCFG2 (base address + 0x02)
2. Configure channel 2 in the DMA, including enabling the channel
3. Configure a timer for the desired trigger interval
4. Write 0xC5 to CHCFG2 (base address + 0x02)

The following code example illustrates steps #1 and #4 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);

In File main.c:
#include "registers.h"
:
:
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xC5;
```

Enabling a source without periodic triggering

Enabling a source without periodic triggering

1. Determine with which DMA channel the source will be associated. Note that only the first 4 DMA channels have periodic triggering capability
2. Clear the CHCFG[ENBL] and CHCFG[TRIG] bits of the DMA channel
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point
4. Select the source to be routed to the DMA channel. Write to the corresponding CHCFG register, ensuring that the CHCFG[ENBL] is set while the CHCFG[TRIG] bit is cleared

Disabling a source

A particular DMA source may be disabled by not writing the corresponding source value into any of the CHCFG registers

Switching the source of a DMA channel

1. Disable the DMA channel in the DMA and re-configure the channel for the new source
2. Clear the CHCFG[ENBL] and CHCFG[TRIG] bits of the DMA channel
3. Select the source to be routed to the DMA channel. Write to the corresponding CHCFG register, ensuring that the CHCFG[ENBL] and CHCFG[TRIG] bits are set

Example:

Switch DMA channel 8 from source #5 transmit to source #7 transmit

1. In the DMA configuration registers, disable DMA channel 8 and re-configure it to handle the transfers to peripheral slot 7. This example assumes channel 8 doesn’t have triggering capability
2. Write 0x00 to CHCFG8 (base address + 0x08)
3. Write 0x87 to CHCFG8 (base address + 0x08). (In this example, setting the CHCFG[TRIG] bit would have no effect, due to the assumption that channels 8 does not support the periodic triggering functionality).

The following code example illustrates steps #2 and #3 above:

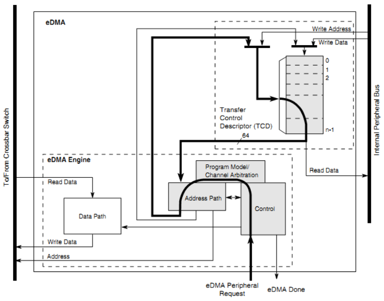
```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);

In File main.c:
#include "registers.h"
:
:
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x87;
```

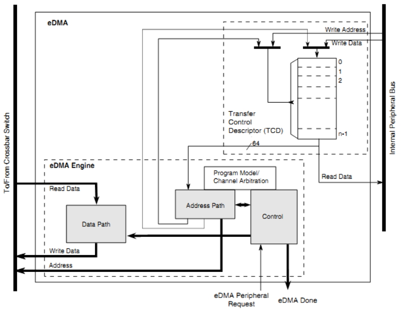
DMA Controller of K70

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source and destination addresses and transfer size
  - Support for enhanced addressing modes
- 32-channel implementation that performs complex data transfers with minimal intervention from a host processor
  - Internal data buffer, used as temporary storage to support 16-byte burst transfers
  - Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
  - 32-byte TCD stored in local memory for each channel
  - An inner data transfer loop defined by a minor byte transfer count
  - An outer data transfer loop defined by a major iteration count
- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
  - Peripheral-paced hardware requests, one per channel
- Fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Optional error terminations per channel and logically summed together to form one error interrupt to the interrupt controller
- Optional support for scatter/gather DMA processing
- Support for complex data structures
- Support to cancel transfers via software

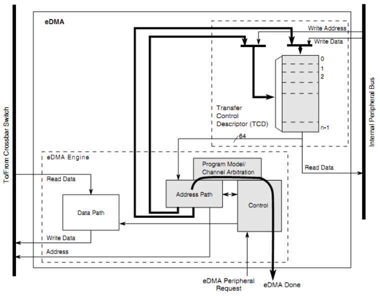
3 Segments Operation Procedure: 1st Channel Activation



2nd Data Transfer



3rd: Completion



Configurable DMA Registers

- TCDn: 32 transfer control descriptors, each has:
- Source address: DMA_TCDn_SADDR, 32 bits
- Signed Source Address Offset: DMA_TCDn_SOFF, 16 bits
- Transfer Attributes: DMA_TCDn_ATTR: 16 bits
  - SMOD, SSIZE,DMOD, DSIZE
- Minor Byte Count: DMA_TCDn_NBYTES_MLNO
- Signed Minor Loop Offset: DMA_TCDn_NBYTES_MLOFFNO
- Last Source Address Adjustment: DMA_TCDn_SLAST
- Destination Address: DMA_TCDn_DADDR
- Signed Destination Address Offset: DMA_TCDn_DOFF
- Current Minor Loop Link, Major Loop Count: DMA_TCDn_CITER_ELINKNO
- Last Destination Address Adjustment: DMA_TCDn_DLASTSGA
- Control and Status Register: TCDn_CSR
- Beginning Minor Loop Link, Major Loop Count: DMA_TCDn_BITER_ELINKYES
  - ELINK, LINKCH,BITER

Example Registers of TCDn

- Control and Status Register

Addresses: 4000_8000h base + 101Ch offset + (32d × *n*), where *n* = 0d to 31d

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	BWC		0	MAJORLINKCH					DONE	ACTIVE	MAJORELINK	ESG	DREQ	INTHALF	INTMAJOR	START
Write			▨													
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:
- x = Undefined at reset.

- BWC: BW control:
  - 10: DMA stalls for 4 cycle after each R/W,
  - 11 stalls for 8 cycles
- Bit 5 enable channel linking to MAJOR_LINKCH after Major Loop Counter is exhausted
- Done: [7] : completion flag
- ACTIVE: signaling DMA active
- Bits 1 and 2 enable IRQ when MLC done and half done, resp
- START: writing a 1 to this bit explicitly start DMA

Transfer Attributes Register: ATTR

- Defines several different transfer attributes;

Addresses: 4000_8000h base + 1006h offset + (32d × *n*), where *n* = 0d to 31d

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	SMOD					SSIZE			DMOD					DSIZE		
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:
- x = Undefined at reset.

- SMOD
  - 5 bits address modulo for implementing a circular queue buffer
  - Value here specifies # of lower order address bits that changes
  - Src offset is typically set to the transfer size, post-imcrement
- SSIZE
  - Source data transfer size: 0:8-bit, 1:16-bit, 2:32-bit, 4:16B
- DMOD, DSIZE

eDMA initialization

To initialize the eDMA:

1. Write to the CR if a configuration other than the default is desired.
2. Write the channel priority levels to the DCHPRIn registers if a configuration other than the default is desired.
3. Enable error interrupts in the EEI register if so desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the ERQ register.
6. Request channel service via either:

- Software: setting the TCDn_CSR[START]
- Hardware: slave device asserting its eDMA peripheral request signal

Single Request Example:

```
TCDn_CITER = TCDn_BITER = 1
TCDn_NBYTES = 16
TCDn_SADDR = 0x1000
TCDn_SOFF = 1
TCDn_ATTR[SSIZE] = 0
TCDn_SLAST = -16
TCDn_DADDR = 0x2000
TCDn_DOFF = 4
TCDn_ATTR[DSIZE] = 2
TCDn_DLAST_SGA= -16
TCDn_CSR[INT_MAJ] = 1
TCDn_CSR[START] = 1 (Should be written last after all other fields have been initialized)
All other TCDn fields = 0
```

This generates the following event sequence:

1. User write to the TCDn_CSR[START] bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCDn_CSR[DONE] = 0, TCDn_CSR[START] = 0, TCDn_CSR[ACTIVE] = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source-to-destination transfers are executed as follows:
   - a. Read byte from location 0x1000, read byte from location 0x1001, read byte from 0x1002, read byte from 0x1003.
   - b. Write 32-bits to location 0x2000 → first iteration of the minor loop.
   - c. Read byte from location 0x1004, read byte from location 0x1005, read byte from 0x1006, read byte from 0x1007.
   - d. Write 32-bits to location 0x2004 → second iteration of the minor loop.
   - e. Read byte from location 0x1008, read byte from location 0x1009, read byte from 0x100A, read byte from 0x100B.
   - f. Write 32-bits to location 0x2008 → third iteration of the minor loop.
   - g. Read byte from location 0x100C, read byte from location 0x100D, read byte from 0x100E, read byte from 0x100F.
   - h. Write 32-bits to location 0x200C → last iteration of the minor loop → major loop complete.
6. The eDMA engine writes: TCDn_SADDR = 0x1000, TCDn_DADDR = 0x2000,TCDn_CITER = 1 (TCDn_BITER).
7. The eDMA engine writes: TCDn_CSR[ACTIVE] = 0, TCDn_CSR[DONE] = 1, INT[n] = 1.
8. The channel retires and the eDMA goes idle or services the next channel.