

F²MC16LX FAMILY

16-BIT MICROCONTROLLER

ALL SERIES

EIOS & μ DMA

APPLICATION NOTE

Revision History

Date	Issue
01/08/2003	1.0, MST Initial version

This document contains 33 pages.

Warranty and Disclaimer

To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH restricts its warranties and its liability for **all products delivered free of charge** (eg. software include or header files, application examples, target boards, evaluation boards, engineering samples of IC's etc.), its performance and any consequential damages, on the use of the Product in accordance with (i) the terms of the License Agreement and the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials. In addition, to the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH disclaims all warranties and liabilities for the performance of the Product and any consequential damages in cases of unauthorised decompiling and/or reverse engineering and/or disassembling. **Note, all these products are intended and must only be used in an evaluation laboratory environment.**

1. Fujitsu Microelectronics Europe GmbH warrants that the Product will perform substantially in accordance with the accompanying written materials for a period of 90 days from the date of receipt by the customer. Concerning the hardware components of the Product, Fujitsu Microelectronics Europe GmbH warrants that the Product will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.
2. Should a Product turn out to be defect, Fujitsu Microelectronics Europe GmbH's entire liability and the customer's exclusive remedy shall be, at Fujitsu Microelectronics Europe GmbH's sole discretion, either return of the purchase price and the license fee, or replacement of the Product or parts thereof, if the Product is returned to Fujitsu Microelectronics Europe GmbH in original packing and without further defects resulting from the customer's use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to Fujitsu Microelectronics Europe GmbH, or abuse or misapplication attributable to the customer or any other third party not relating to Fujitsu Microelectronics Europe GmbH.
3. To the maximum extent permitted by applicable law Fujitsu Microelectronics Europe GmbH disclaims all other warranties, whether expressed or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the Product is not designated.
4. To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH's and its suppliers' liability is restricted to intention and gross negligence.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES

To the maximum extent permitted by applicable law, in no event shall Fujitsu Microelectronics Europe GmbH and its suppliers be liable for any damages whatsoever (including but without limitation, consequential and/or indirect damages for personal injury, assets of substantial value, loss of profits, interruption of business operation, loss of information, or any other monetary or pecuniary loss) arising from the use of the Product.

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect

Contents

REVISION HISTORY	2
WARRANTY AND DISCLAIMER	3
CONTENTS	4
1 INTRODUCTION	6
2 EI²OS	7
2.1 Series	7
2.2 EI ² OS Function	7
2.3 EI ² OS set-up	8
2.3.1 EI ² OS Descriptor	8
2.3.1.1 Data Counter (DCT)	9
2.3.1.2 I/O Register Address Pointer (IOA)	9
2.3.1.3 EI ² OS status register (ISCS)	9
2.3.1.4 Buffer address pointer (BAP)	10
2.3.2 Interrupt Control Register (ICR)	11
2.3.3 Operation-flow of EI ² OS	12
2.3.4 Execution time of EI ² OS	13
2.3.5 Restrictions	14
3 μDMA USING DESCRIPTOR IN RAM	15
3.1 Series	15
3.2 μ DMA Function	15
3.2.1 μ DMA Register	15
3.2.1.1 DMA enable register (DER)	16
3.2.1.2 DMA stop status register (DSSR)	16
3.2.1.3 DMA status register (DSR)	17
3.2.2 μ DMA Descriptor in RAM	17
3.2.2.1 Data Counter (DCT)	18
3.2.2.2 I/O Register Address Pointer (IOA)	18
3.2.2.3 μ DMA status register (DMACS)	19
3.2.2.4 Buffer address pointer (BAP)	20
3.3 μ DMA processing procedure	20
3.4 μ DMA Processing Time	21
3.4.1 Continuous data transfer	21
3.4.2 Transfer performance	22

3.4.2.1	Minimum transfer speed	22
3.4.2.2	Maximum transfer speed	22
3.4.3	End of transfer (request of resource) processing time.....	22
3.5	Restrictions	22
4	SUMMARY.....	23
4.1	EI ² OS	23
4.2	μ DMA using Descriptor in RAM.....	23
5	APPENDIX.....	24
5.1	Series Overview	24
5.2	Example Descriptor set-up	25
5.2.1	EI ² OS.....	25
5.2.1.1	Isd.h.....	26
5.2.2	μ DMA using Descriptor in RAM	29
5.2.2.1	μ dma.c.....	29
5.2.2.2	μ dma.h	30
5.3	Figures	33
5.4	Tables	33

1 Introduction

In general Resource Interrupts are used to react to external Events.

In the case of high frequency Interrupts, using the standard Interrupt mechanism the system performance will be slow down.

For such kind of Interrupt cases (e.g. UART data block transfer, periodically ADC scanning) the Extended Intelligence I/O Service (EI²OS) transfer can be used.

In new devices the enhanced version of EI²OS, now called μ DMA , is used.

This Application Note shows how to set-up the different macros and restrictions.

2 EI²OS

Extended intelligent I/O service (EI²OS)

The EI²OS function automatically transfers data between input / output and memory.

2.1 Series

The EIIOS is implemented in following series:

- MB90385
- MB90390
- MB90420/425
- MB90435
- MB90440
- MB90455
- MB90460
- MB90470
- MB90495
- MB90520
- MB90540/545
- MB90550
- MB90560/565
- MB90570
- MB90580
- MB90590
- MB90595

2.2 EI²OS Function

Instead of using a 'normal' interrupt the EI²OS can be used to handle the interrupts in the background.

The EI²OS transfer automatically data between peripheral resources (I/O) and memory.

- No internal register is used for transfer, eliminating the need for register saving and increasing the transfer speed.
- Transfer can be terminated from I/O, preventing unnecessary data from being transferred. (Stop request)
- Incrementing, decrementing, or no update can be selected for the buffer address.
- Incrementing, decrementing, or no update can be selected for the I/O register address (if the buffer address is updated).
- Generates EI²OS end of transfer interrupt

2.3 EI²OS set-up

The ICR register and the EI²OS Descriptor must be set-up.

The upper 4 bits of ICR register (ICS3-0) selects the used EI²OS channel.

Bit 3 (ISE) enables EI²OS. Because an end of EI²OS transfer-interrupt is generated, the lower 3 bits (IL0-3) define the interrupt level.

There are 16 EI²OS channels available. Channel 0 Descriptor starts at address 0x100, channel 16 Descriptor at address 0x178.

The EI²OS Descriptors are located in RAM, using 8 Bytes of memory.

2.3.1 EI²OS Descriptor

The EI²OS Descriptor consists of following Registers

- Data Counter (DCT)
- I/O Register Address Pointer (IOA)
- EI²OS status register (ISCS)
- Buffer address pointer (BAP)

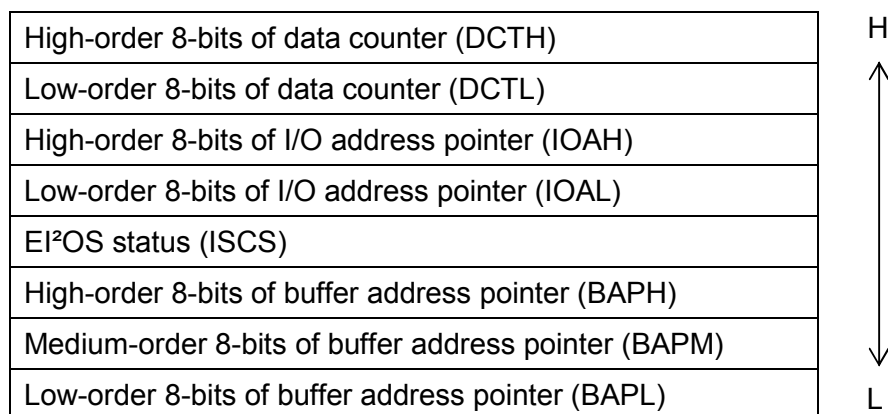


Figure 1: Extended Intelligence I/O Service Descriptor Configuration

The Descriptor must be linked to correct address by the Application.

An example how to do this is shown via our isd.c and isd.h files.

See chapter Appendix.

2.3.1.1 Data Counter (DCT)

The 16-bit Data counter register works as a counter. It is decremented by one before each data transfer. EI²OS is terminated when the counter reaches '0'.

Data counter (upper) (DCTH)

Bit	15	14	13	12	11	10	9	8
	B15	B14	B13	B12	B11	B10	B9	B8

Data counter (lower) (DCTL)

Bit	7	6	5	4	3	2	1	0
	B7	B6	B5	B4	B3	B2	B1	B0

2.3.1.2 I/O Register Address Pointer (IOA)

This 16-bit register indicates the low-order address (A15-0) of the buffer and I/O register used for data transfer. The high-order address (A23-A16) is zero. Any I/O between 0x0000 – 0xFFFF addresses can be specified.

I/O Register Address Pointer (upper) (IOAH)

Bit	15	14	13	12	11	10	9	8
	A15	A14	A13	A12	A11	A10	A9	A8

I/O Register Address Pointer (lower) (IOAL)

Bit	7	6	5	4	3	2	1	0
	A7	A6	A5	A4	A3	A2	A1	A0

2.3.1.3 EI²OS status register (ISCS)

The ISCS register is an 8-bit register. It indicates the update direction (increment/decrement), transfer data format (Byte/word) and transfer direction.

EI²OS status Register (ISCS)

Bit	7	6	5	4	3	2	1	0
	Res.	Res.	Res.	IF	BW	BF	DIR	SE

IF: Specify if whether I/O register address pointer is updated or not

IF	Function
0	After data transfer, the I/O register address pointer is updated
1	After data transfer, the I/O register address pointer is not updated

BW: Specify the transfer data length

BW	Function
0	Byte
1	Word

BF: Specify if whether memory address pointer is updated or not

BF	Function
0	After data transfer, the buffer address pointer is updated
1	After data transfer, buffer address pointer is not updated

DIR: specify transfer direction

DIR	Function
0	I/O -> buffer
1	Buffer -> I/O

SE: Specify whether transfer can be terminated via resource request

SE	Function
0	Not terminated by a resource request
1	Terminated by a resource request

2.3.1.4 Buffer address pointer (BAP)

The 24-bit register holds the address used for next EI²OS transfer. With the 24-bit addressing, the buffer can be anywhere within 16Mbyte space.

2.3.2 Interrupt Control Register (ICR)

ICR-Register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ICS3	ICS2	ICS1	ICS0	ISE	IL2	IL1	IL0

ICS3	ICS2	ICS1	ICS0	Channel	Descriptor address
0	0	0	0	0	0x100
0	0	0	0	1	0x108
...
1	1	1	1	15	0x178

ISE:

0	EI ² OS disabled (default)
1	EI ² OS enabled

Interrupt Level Bits (ILx)

IL2	IL1	IL0	Interrupt Level
0	0	0	0 (lowest priority)
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6 (lowest priority)
1	1	1	7 (interrupt disabled)

Be aware, the ICR register have different meaning for writing or reading.

Check the Hardware Manual for details.

2.3.3 Operation-flow of EI²OS

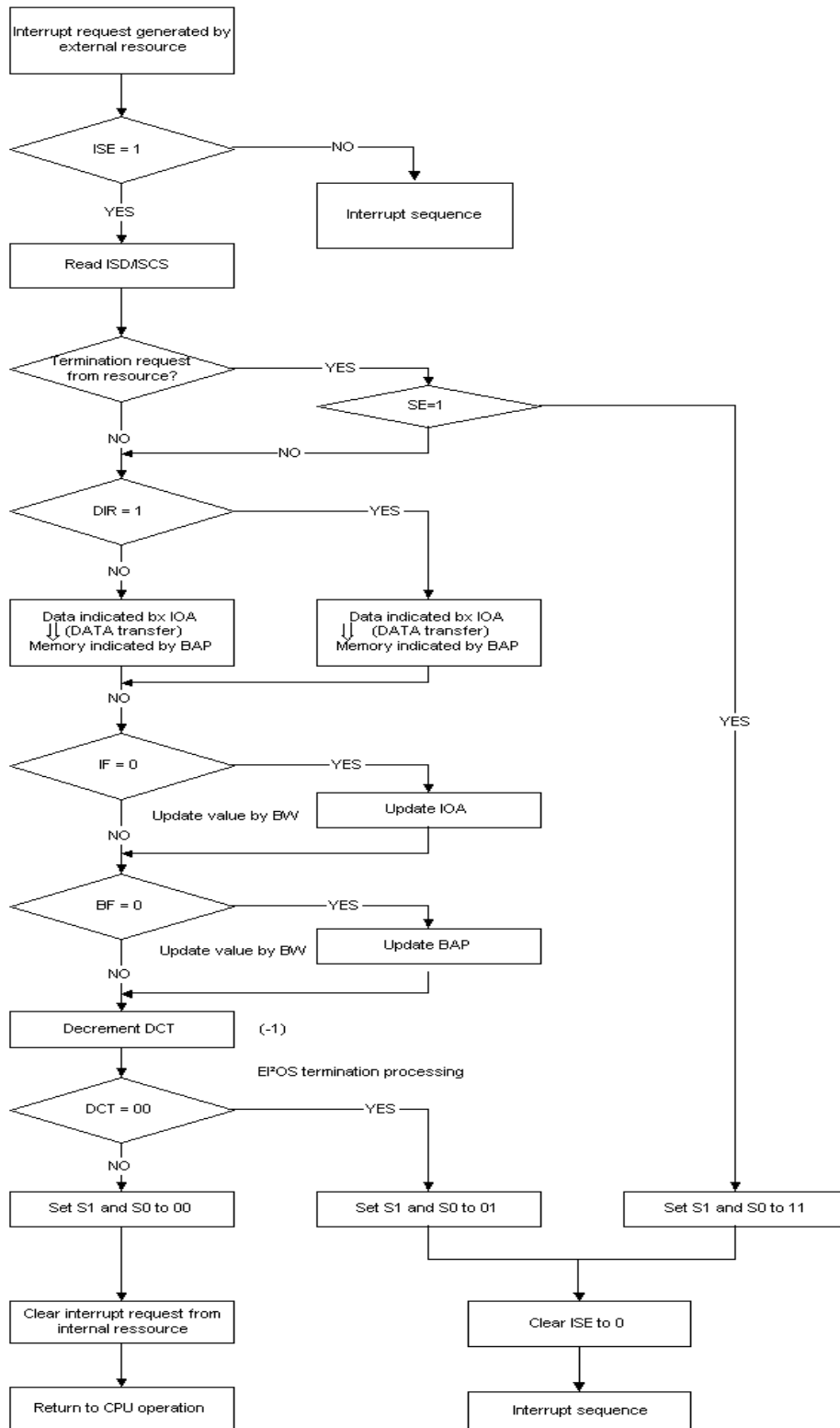


Figure 2: EI²OS Operation Flow

ISD : IEI2OS descriptor

ISCS : EI2OS status register

IF : IOA update/fixed selection bit in the EI2OS status register (ISCS)

BW : Transfer data length specification bit in the EI2OS status register (ISCS)

BF : BAP update/fixed selection bit in the EI2OS status register (ISCS)

DIR : Data transfer direction specification bit in the EI2OS status register (ISCS)

SE : EI2OS termination control bit in the EI2OS status register (ISCS)

DCT : Data counter

IOA : I/O register address pointer

BAP : Buffer address pointer

ISE : EI2OS enable bit in the interrupt control register (ICR)

S1,S0 : EI2OS status in the interrupt control register (ICR)

2.3.4 Execution time of EI²OS

Refer also the corresponding Hardware Manual of used series.

- when data transfer continues
Table 1 + Table 3 = machine cycles
- When a Stop request is issued from resource
 $36 + (6 \times Z) =$ machine cycles
Z: See Table 3
- When the counting is complete
Table 1 + Table 3 + Table 2 = machine cycles

ISCS SE bit		Set to 0		Set to 1		
		Fixed	Updated	Fixed	Updated	
I/O Address pointer	Buffer address pointer	Fixed	32	34	33	35
		Updated	34	36	35	37

Table 1: Execution Time of EI²OS

I/O Address pointer			Internal access		External access	
			B/E	O	B/E	8/O
Buffer address pointer	Internal access	B/E	0	+2	+1	+4
		O	+2	+4	+3	+6
	External access	B/E	+1	+3	+2	+5
		8/O	+4	+6	+5	+8

Table 2: Data Transfer Compensation Values for EI²OS Execution Time

B: Byte data transfer

8: 8-bit external bus transfer

E: Even address word transfer

O: Odd address word transfer

Address indicated by the stack pointer	Cycle count compensation value
External area, 8-bit data bus	+4
External area, even-numbered address	+1
External area, odd-numbered address	+4
Internal area, even-numbered address	0
Internal area, odd-numbered address	+2

Table 3: Compensation values for Interrupt Processing cycle count

2.3.5 Restrictions

Each ICR Register is used to set the interrupt level of two resources.

When using EI²OS with one resource, the other resource cannot be used with interrupts.

Otherwise this resource will activate the EI²OS transfer, too.

Descriptors using RAM, (0x100 – 0x17F), be aware of this issue. Do not link other program parts into this area when using EI²OS.

3 μ DMA using Descriptor in RAM

The μ DMA is an improved version of EI²OS. Enabling and channel selection have to be set-up in special μ DMA Registers. The ICR Registers are not used therefore. The Descriptor area is in RAM area 0x100-0x17F. How to set-up such kind of Descriptor is shown in chapter Appendix.

3.1 Series

The μ DMA using Descriptors in RAM is implemented in following series:

- MB90470
- MB90480

3.2 μ DMA Function

The μ DMA consists of 16 channels. Every channel is fixed to a resource.

The μ DMA registers are used to enable/disable transfer, status, etc.

The Descriptors for each channel are located in the RAM.

The corresponding ICR register set the interrupt level of μ DMA end-of-transfer interrupt.

The μ DMA transfer automatically data between peripheral resources (I/O) and memory.

3.2.1 μ DMA Register

The μ DMA is using 3 internal Registers:

- DMA enable Register (DER)
- DMA stop status register (DSSR)
- DMA status register (DSR)

3.2.1.1 DMA enable register (DER)

The DER Register is a 16-bit Register.

DERH:

Bit	15	14	13	12	11	10	9	8
	EN15	EN14	EN13	EN12	EN11	EN10	EN9	EN8

DERL:

Bit	7	6	5	4	3	2	1	0
	EN7	EN6	EN5	EN4	EN3	EN2	EN1	EN0

μ DMA enable bit

ENx bit	Function
0 (initial value)	Outputs an interrupt request from a resource to the interrupt controller. (An interrupt request from a resource is not used as a DMA start request).
1	An interrupt request output from a resource is used as a DMA start request. Cleared to "0" when the DMA transfer byte count reaches 0.

3.2.1.2 DMA stop status register (DSSR)

The DSSR register is a 8-bit register. Resource can stop transfer if enabled.

Bit	7	6	5	4	3	2	1	0
	STP7	STP6	STP5	STP4	STP3	STP2	STP1	STP0

STPx bit	Function
0 (initial value)	No STOP request is accepted in a DMA transfer.
1	STOP request is accepted in a DMA transfer to stop DMA operation. STOP request is accepted only the UART receive (channels 7). The bits other than the bit 7 are not valid. Writing "1" by running software is not valid.

3.2.1.3 DMA status register (DSR)

Displays if μ DMA transfer is ongoing or finished.

DSRH

Bit	15	14	13	12	11	10	9	8
	DE15	DE14	DE13	DE12	DE11	DE10	DE9	DE8

DSRL

Bit	7	6	5	4	3	2	1	0
	DE7	DE6	DE5	DE4	DE3	DE2	DE1	DE0

Bit description:

DEx bit	Function
0 (initial value)	DMA transfer is not finished.
1	If a DMA transfer ends, an interrupt request is output to the interrupt controller.

3.2.2 μ DMA Descriptor in RAM

The μ DMA is using dedicated channels for each resource.

So, each Descriptor is dedicated to a resource/channel.

The Descriptor must be located to the correct memory address.

Check the Hardware Manual of series for Details.

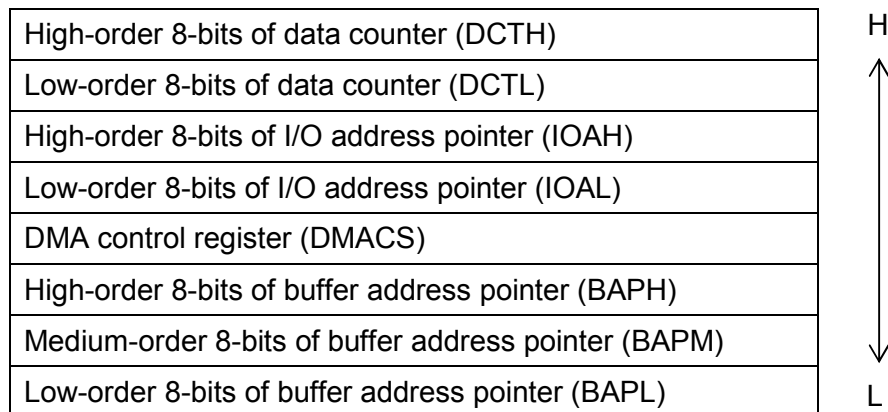


Figure 3 μ DMA Descriptor Configuration

3.2.2.1 Data Counter (DCT)

The 16-bit Data counter register works as a counter. It is decremented by one before each data transfer. μ DMA is terminated when the counter reaches '0'.

Data counter (upper) (DCTH)

Bit	15	14	13	12	11	10	9	8
	B15	B14	B13	B12	B11	B10	B9	B8

Data counter (lower) (DCTL)

Bit	7	6	5	4	3	2	1	0
	B7	B6	B5	B4	B3	B2	B1	B0

3.2.2.2 I/O Register Address Pointer (IOA)

This 16-bit register indicates the low-order address (A15-0) of the buffer and I/O register used for data transfer. The high-order address (A23-A16) is zero. Any I/O between 0x0000 – 0xFFFF addresses can be specified.

I/O Register Address Pointer (upper) (IOAH)

Bit	15	14	13	12	11	10	9	8
	A15	A14	A13	A12	A11	A10	A9	A8

I/O Register Address Pointer (lower) (IOAL)

Bit	7	6	5	4	3	2	1	0
	A7	A6	A5	A4	A3	A2	A1	A0

3.2.2.3 μ DMA status register (DMACS)

The DMACS register is an 8-bit register. It indicates the update direction (increment/decrement), transfer data format (Byte/word) and transfer direction.

μ DMA status Register (DMACS)

Bit	7	6	5	4	3	2	1	0
	Res.	Res.	Res.	IF	BW	BF	DIR	SE

IF: Specify if whether I/O register address pointer is updated or not

IF	Function
0	After data transfer, the I/O register address pointer is updated
1	After data transfer, the I/O register address pointer is not updated

BW: Specify the transfer data length

BW	Function
0	Byte
1	Word

BF: Specify if whether memory address pointer is updated or not

BF	Function
0	After data transfer, the buffer address pointer is updated
1	After data transfer, buffer address pointer is not updated

DIR: specify transfer direction

DIR	Function
0	I/O -> buffer
1	Buffer -> I/O

SE: Specify whether transfer can be terminated via resource request

SE	Function
0	Not terminated by a resource request
1	Terminated by a resource request

3.2.2.4 Buffer address pointer (BAP)

The 24-bit register holds the address used for next μ DMA transfer. With the 24-bit addressing, the buffer can be anywhere within 16Mbyte space.

3.3 μ DMA processing procedure

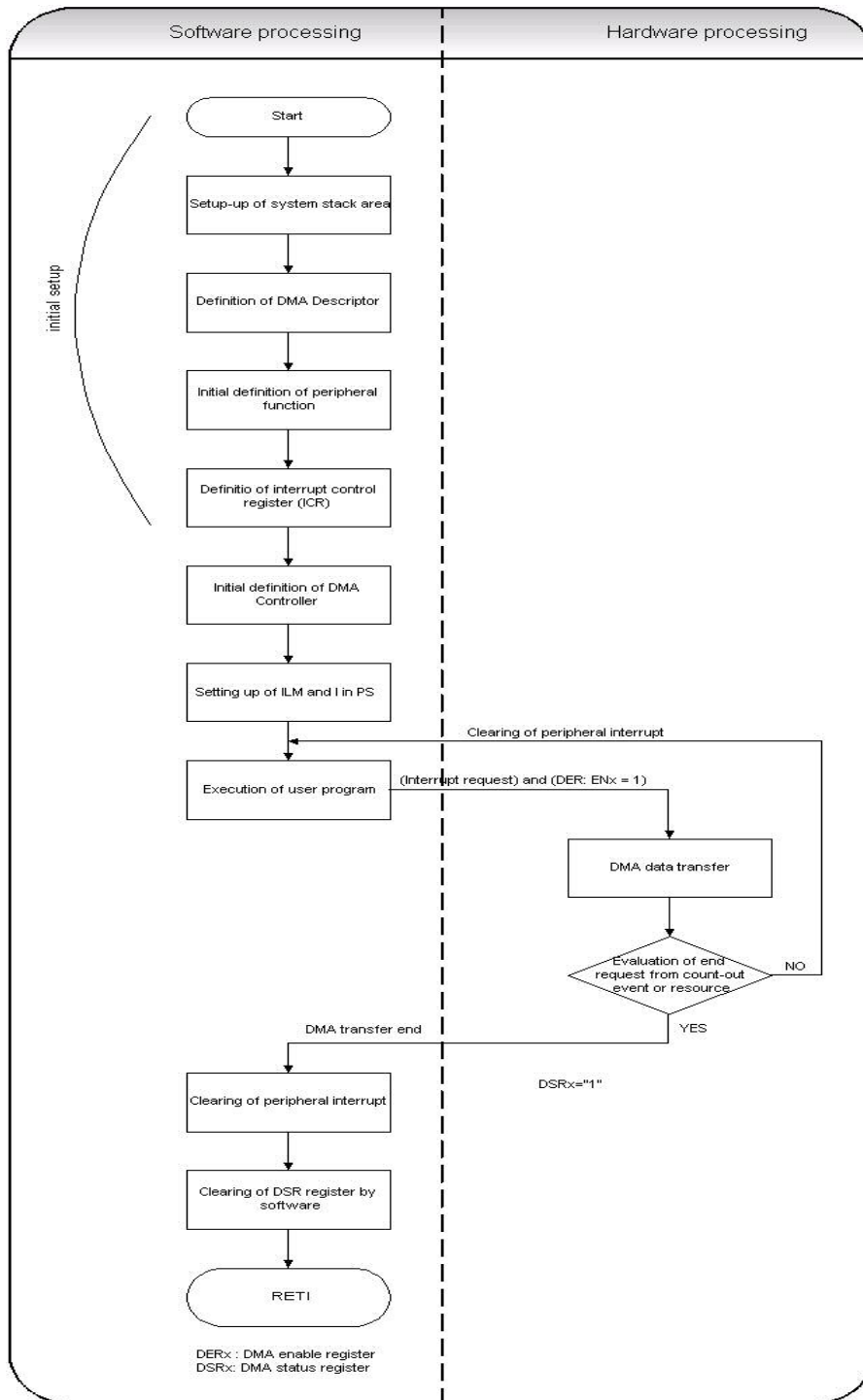


Figure 4: μ DMA processing procedure

3.4 μ DMA Processing Time

The processing time of μ DMA varies with following factors:

- Settings of μ DMA control status registers (DMACS)
- Address (area) indicated by I/O register address pointer (IOA)
- Address (area) indicated by the buffer address pointer (BAP)
- External data bus width for external access
- Data length of transfer data

3.4.1 Continuous data transfer

The μ DMA processing time during continuous data transfer depends of the settings of the BAP register.

See Table 4

Depending of the μ DMA -execution condition a correction is required.

See Table 5

Processing time during continuous data transfer:

Table 4 + Table 5 = machine cycles

Setting of IOA update/fixed selection bit		Fixed	Update
BAP address update/fixed Setting of selection bit (BF)	Fixed	17	19
	Update	19	21

Table 4: μ DMA execution time

I/O register address pointer			Internal access		External access	
			B / even	Odd	B / even	8 / odd
Buffer address pointer	Internal access	B/even	0	+2	+1	+4
		Odd	+2	+4	+3	+6
	External access	B / even	+1	+3	+2	+5
		8 / odd	+4	+6	+5	+8

Table 5: Correction values of data transfer for μ DMA execution time

B: Byte data transfer

8: Word transfer with external bus width of 8-bit

Even: Word transfer of an even-numbered address

Odd: Word transfer of an odd-numbered address

3.4.2 Transfer performance

3.4.2.1 Minimum transfer speed

1.7 μ s / 10 MHz (machine clock)

1.07 μ s / 16 MHz (Machine clock)

For following cases:

- Built-in I/O -> built-in RAM; or built-in RAM -> built-in I/O without address increment
- Even numbered address -> even numbered address or 8-bit access

3.4.2.2 Maximum transfer speed

2.8 μ s / 10 MHz (machine clock)

1.75 μ s / 16 MHz (machine clock)

3.4.3 End of transfer (request of resource) processing time

When a transfer-end is requested of the peripheral function (I/O)

If a μ DMA data transfer ends partway (DEx = 1) because of an end request by a peripheral function (I/O), the data transfer fails and the hardware interrupt starts. The μ DMA processing time in this case is calculated with the following formula. Z in the formula indicates a correction value for interrupt processing time. (See Table 6)

$36 + (6 \times Z)$ machine cycle

Address indicated by stack pointer	Correction value (Z)
External 8bit	+4
External even-numbered address	+1
External odd-numbered address	+4
Internal even-numbered address	0
Internal odd-numbered address	+2

Table 6: Correction values (Z) for interrupt handling time

3.5 Restrictions

Descriptors using RAM area (0x100 – 0x17F), be aware of this issue. Do not link other program parts into this area when using μ DMA.

Every resource is using a dedicated channel, be sure to link the Descriptor Register to the correct address. (Check Hardware Manual of used series)

4 Summary

4.1 EI²OS

- 16 channels available
- Channel free selectable to resource (in ICR register)
- Descriptor located in RAM
- EI²OS Channel selection via ICR register
- Two resources sharing one ICR register -> using EI²OS -> second resource cannot use interrupts
- End of transfer interrupt is a 'normal' Interrupt, so in the corresponding ICR register the interrupt level must be additionally set.

4.2 μ DMA using Descriptor in RAM

- 16 channels available
- Each channel is dedicated to one resource
- Descriptor located in RAM
- The use of μ DMA is independent of ICR register -> both resources sharing one ICR register can be used with μ DMA or μ DMA and 'normal' Interrupts.
- End of transfer interrupt is a 'normal' Interrupt, the corresponding ICR register must be used to set the interrupt level

5 Appendix

5.1 Series Overview

Series	EI ² OS	μ DMA (Descriptor in RAM)
MB90385	Yes	--
MB90390	Yss	--
MB90420/425	Yes	--
MB90435	Yes	--
MB90440	Yes	--
MB90455	Yes	--
MB90460	Yes	--
MB90470	Yes	Yes
MB90480	--	Yes
MB90495	Yes	--
MB90520	Yes	--
MB90540/545	Yes	--
MB90550	Yes	--
MB90560/565	Yes	--
MB90570	Yes	--
MB90580	Yes	--
MB90590	Yes	--
MB90595	Yes	--

Figure 5: Series Overview

5.2 Example Descriptor set-up

5.2.1 EI²OS

Isd.c

```
/******  
*  
* DESCRIPTION:  
*  
*     Extended Intelligent I/O Service Descriptor Definition  
*  
* AUTHOR: Fujitsu Mikroelektronik GmbH, HL  
*  
* HISTORY:  
*     Version 1.0: original version  
*     Version 2.0: macros added  
*           Verison 3.0: - isd.h and isd.c splitted  
*                       - section name for SWB used  
*                       - default number set to 1  
*                       - underscore from "_isd" removed  
*                       - name changed to Isd  
*     Version 4.0: - error in defines in isd.h removed  
*****/  
  
#include "isd.h"  
  
#pragma section DATA=ISD,attr=DATA,locate=0x000100  
  
/* reserve area */  
ISDSTR Isd[_NUM_ISD];
```

5.2.1.1 Isd.h

```

/*****
*
* DESCRIPTION:
*
* Extended Intelligent I/O Service Descriptor Declaration, limited - to save internal
RAM - to 5 EIIOS channels.
*
* If more descriptors are needed, they can be added here easily be the user.
*
* AUTHOR: Fujitsu Mikroelektronik GmbH, HL
*
* HISTORY:
* Version 1.0: original version
* Version 2.0: macros added
* Verison 3.0: - isd.h and isd.c splitted
* - section name for SWB used
* - default number set to 1
* - underscore from "_isd" removed
* - name changed to Isd
* Version 4.0: error in "Defines" removed, tka
*
*****/

#ifndef __ISD_H
#define __ISD_H

#ifndef _NUM_ISD
# define _NUM_ISD 1
#endif

#if _NUM_ISD < 1 || _NUM_ISD > 16
# error Invalid number of EIIOS descriptors
#endif

/* structure of ISD */

```

```
typedef union {
    struct
    {
        unsigned char  BAPL;    /* lower 8 Bit of Buffer Address Pointer */
        unsigned char  BAPM;    /* middle 8 Bit of Buffer Address Pointer */
        unsigned char  BAPH;    /* upper 8 Bit of Buffer Address Pointer */
        unsigned char  ISCS;    /* Interrupt Service Control and Status */
        unsigned short IOA;     /* IO-Address in bank zero */
        unsigned short DCT;     /* current Data Counter */
    } reg;
    struct
    {
        unsigned short BAPML;   /* use this when assigning __near pointers */
    } regc;
} ISDSTR;
```

```
extern ISDSTR Isd[];
```

```
#if _NUM_ISD >= 1
# define ISD0_BAPML  (Isd[0].regc.BAPML)
# define ISD0_BAPL   (Isd[0].reg.BAPL)
# define ISD0_BAPM   (Isd[0].reg.BAPM)
# define ISD0_BAPH   (Isd[0].reg.BAPH)
# define ISD0_ISCS   (Isd[0].reg.ISCS)
# define ISD0_IOA    (Isd[0].reg.IOA)
# define ISD0_DCT    (Isd[0].reg.DCT)
# if _NUM_ISD == 1
# define ISD_BAPML   (Isd[0].regc.BAPML)
# define ISD_BAPL    (Isd[0].reg.BAPL)
# define ISD_BAPM    (Isd[0].reg.BAPM)
# define ISD_BAPH    (Isd[0].reg.BAPH)
# define ISD_ISCS    (Isd[0].reg.ISCS)
# define ISD_IOA     (Isd[0].reg.IOA)
# define ISD_DCT     (Isd[0].reg.DCT)
# endif
#endif
#if _NUM_ISD >= 2
```

```
# define ISD1_BAPML (Isd[1].regc.BAPML)
# define ISD1_BAPL (Isd[1].reg.BAPL)
# define ISD1_BAPM (Isd[1].reg.BAPM)
# define ISD1_BAPH (Isd[1].reg.BAPH)
# define ISD1_ISCS (Isd[1].reg.ISCS)
# define ISD1_IOA (Isd[1].reg.IOA)
# define ISD1_DCT (Isd[1].reg.DCT)
#endif

#if _NUM_ISD >= 3
# define ISD2_BAPML (Isd[2].regc.BAPML)
# define ISD2_BAPL (Isd[2].reg.BAPL)
# define ISD2_BAPM (Isd[2].reg.BAPM)
# define ISD2_BAPH (Isd[2].reg.BAPH)
# define ISD2_ISCS (Isd[2].reg.ISCS)
# define ISD2_IOA (Isd[2].reg.IOA)
# define ISD2_DCT (Isd[2].reg.DCT)
#endif

#if _NUM_ISD >= 4
# define ISD3_BAPML (Isd[3].regc.BAPML)
# define ISD3_BAPL (Isd[3].reg.BAPL)
# define ISD3_BAPM (Isd[3].reg.BAPM)
# define ISD3_BAPH (Isd[3].reg.BAPH)
# define ISD3_ISCS (Isd[3].reg.ISCS)
# define ISD3_IOA (Isd[3].reg.IOA)
# define ISD3_DCT (Isd[3].reg.DCT)
#endif

#if _NUM_ISD >= 5
# define ISD4_BAPML (Isd[4].regc.BAPML)
# define ISD4_BAPL (Isd[4].reg.BAPL)
# define ISD4_BAPM (Isd[4].reg.BAPM)
# define ISD4_BAPH (Isd[4].reg.BAPH)
# define ISD4_ISCS (Isd[4].reg.ISCS)
# define ISD4_IOA (Isd[4].reg.IOA)
# define ISD4_DCT (Isd[4].reg.DCT)
#endif

#endif /* __ISD_H */
```

5.2.2 μ DMA using Descriptor in RAM

5.2.2.1 μ dma.c

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.  
FUJITSU */
```

```
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY  
ERRORS OR */
```

```
/* ELIGIBILITY FOR ANY PURPOSES. */
```

```
/* (C) Fujitsu Microelectronics Europe GmbH */
```

```
/******
```

```
*
```

```
* DESCRIPTION:
```

```
*
```

```
* uDMA Descriptor Definition
```

```
*
```

```
* AUTHOR: Fujitsu Microelectronics Europe GmbH, MSt
```

```
* HISTORY:
```

```
* Version 1.0: original version, adapted from ISD.c
```

```
*
```

```
*****/
```

```
#include "uDMA.h"
```

```
/* check Hardware Manual of series to get the correct address for the descriptor */
```

```
#pragma section DATA=ISD,attr=DATA,locate=0x000178 /* ADC descriptor area */
```

```
/* reserve area */
```

```
ISDSTR Isd[_NUM_ISD];
```

5.2.2.2 μ dma.h

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.
FUJITSU */
```

```
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY
ERRORS OR */
```

```
/* ELIGIBILITY FOR ANY PURPOSES. */
```

```
/* (C) Fujitsu Microelectronics Europe GmbH */
```

```
/******
```

```
*
```

```
* DESCRIPTION:
```

```
*
```

```
* uDMA Descriptor Declaration, limited - to save internal RAM - to 5 uDMA channels.
```

```
* If more descriptors are needed, they can be added here easily by the user.
```

```
*
```

```
* AUTHOR: Fujitsu Mikroelektronik GmbH, HL
```

```
*
```

```
* HISTORY:
```

```
* Version 1.0: original version (adapted from ISD.h)
```

```
*
```

```
*****/
```

```
#ifndef __ISD_H
```

```
#define __ISD_H
```

```
#ifndef _NUM_ISD
```

```
# define _NUM_ISD 1
```

```
#endif
```

```
#if _NUM_ISD < 1 || _NUM_ISD > 16
```

```
# error Invalid number of EIIOS descriptors
```

```
#endif
```

```
/* structure of ISD */
```

```
typedef union {
```

```
struct
{
    unsigned char  BAPL;    /* lower 8 Bit of Buffer Address Pointer */
    unsigned char  BAPM;    /* middle 8 Bit of Buffer Address Pointer */
    unsigned char  BAPH ;   /* upper 8 Bit of Buffer Address Pointer */
    unsigned char  DMACS ;  /* Interrupt Service Control and Status */
    unsigned short IOA ;    /* IO-Address in bank zero */
    unsigned short DCT ;    /* current Data Counter */
} reg;
struct
{
    unsigned short BAPML;   /* use this when asigning __near pointers */
} regc;
} ISDSTR;

extern ISDSTR Isd[];

#if _NUM_ISD >= 1
# define ISD0_BAPML (Isd[0].regc.BAPML)
# define ISD0_BAPL (Isd[0].reg.BAPL)
# define ISD0_BAPM (Isd[0].reg.BAPM)
# define ISD0_BAPH (Isd[0].reg.BAPH)
# define ISD0_DMACS (Isd[0].reg.DMACS)
# define ISD0_IOA (Isd[0].reg.IOA)
# define ISD0_DCT (Isd[0].reg.DCT)
# if _NUM_ISD == 1
# define ISD_BAPML (Isd[0].regc.BAPML)
# define ISD_BAPL (Isd[0].reg.BAPL)
# define ISD_BAPM (Isd[0].reg.BAPM)
# define ISD_BAPH (Isd[0].reg.BAPH)
# define ISD_DMACS (Isd[0].reg.DMACS)
# define ISD_IOA (Isd[0].reg.IOA)
# define ISD_DCT (Isd[0].reg.DCT)
# endif
#endif
#if _NUM_ISD >= 2
# define ISD1_BAPML (Isd[1].regc.BAPML)
```

```
# define ISD1_BAPL    (Isd[1].reg.BAPL)
# define ISD1_BAPM    (Isd[1].reg.BAPM)
# define ISD1_BAPH    (Isd[1].reg.BAPH)
# define ISD1_DMACS   (Isd[1].reg.DMACS)
# define ISD1_IOA     (Isd[1].reg.IOA)
# define ISD1_DCT     (Isd[1].reg.DCT)
#endif

#if _NUM_ISD >= 3
# define ISD2_BAPML   (Isd[2].regc.BAPML)
# define ISD2_BAPL    (Isd[2].reg.BAPL)
# define ISD2_BAPM    (Isd[2].reg.BAPM)
# define ISD2_BAPH    (Isd[2].reg.BAPH)
# define ISD2_DMACS   (Isd[2].reg.DMACS)
# define ISD2_IOA     (Isd[2].reg.IOA)
# define ISD2_DCT     (Isd[2].reg.DCT)
#endif

#if _NUM_ISD >= 4
# define ISD3_BAPML   (Isd[3].regc.BAPML)
# define ISD3_BAPL    (Isd[3].reg.BAPL)
# define ISD3_BAPM    (Isd[3].reg.BAPM)
# define ISD3_BAPH    (Isd[3].reg.BAPH)
# define ISD3_DMACS   (Isd[3].reg.DMACS)
# define ISD3_IOA     (Isd[3].reg.IOA)
# define ISD3_DCT     (Isd[3].reg.DCT)
#endif

#if _NUM_ISD >= 5
# define ISD4_BAPML   (Isd[4].regc.BAPML)
# define ISD4_BAPL    (Isd[4].reg.BAPL)
# define ISD4_BAPM    (Isd[4].reg.BAPM)
# define ISD4_BAPH    (Isd[4].reg.BAPH)
# define ISD4_DMACS   (Isd[4].reg.DMACS)
# define ISD4_IOA     (Isd[4].reg.IOA)
# define ISD4_DCT     (Isd[4].reg.DCT)
#endif

#endif /* __ISD_H */
```

5.3 Figures

Figure 1: Extended Intelligence I/O Service Descriptor Configuration	8
Figure 2: EI ² OS Operation Flow	12
Figure 3 μ DMA Descriptor Configuration.....	17
Figure 4: μ DMA processing procedure.....	20
Figure 5: Series Overview	24

5.4 Tables

Table 1: Execution Time of EI ² OS	13
Table 2: Data Transfer Compensation Values for EI ² OS Execution Time.....	14
Table 3: Compensation values for Interrupt Processing cycle count.....	14
Table 4: μ DMA execution time	21
Table 5: Correction values of data transfer for μ DMA execution time	21
Table 6: Correction values (Z) for interrupt handling time.....	22

-- END --