M16C/26

Using the DMAC with a Forward Destination

1.0 Abstract

The following article introduces and shows an example of how to use the DMAC function of the M16C/26 with a fixed source address and forward counting destination address.

2.0 Introduction

The Renesas M30262 is a 16-bit MCU based on the M16C/60 series CPU core. The MCU features include up to 64K bytes of Flash ROM, 2K bytes of RAM, and 4K bytes of virtual EEPROM. The peripheral set includes 10-bit A/D, UARTs, Timers, DMA, and GPIO. The MCU has two DMAC (Direct Memory Access Controller) channels that allow data to be transferred from a source memory location to a destination memory location without using the CPU. The DMAC utilizes the same internal address and data busses as the CPU yet is given a higher priority to the data bus than the CPU. This method of DMAC and CPU bus arbitration is termed “cycle stealing”.

Each DMAC controller is capable of transferring data to or from a fixed address to any other address within the 1Mbyte address space. The DMAC controllers can automatically transfer 128k bytes of data, using word (16bit) transfers, or 64k bytes of data using byte (8-bit) transfers. The source or destination address can also be auto-incremented. DMAC transfers can be initiated by an interrupt request signal or by manually writing to the software DMA request bit. When requests are initiated by an interrupt request signal, neither the interrupt enable flag (I flag) nor the interrupt priority level affects the DMA transfers.

3.0 DMAC with Fixed Source, Forward Destination Description

In the fixed source address, forward counting address mode, the DMAC controller will transfer bytes or words from a fixed source address to an incrementing destination address (increments after each transfer). The transfers can either be bytes or words. Loading a value into the transfer count register controls the number of automated transfers. Transfers will continue to occur each time the DMAC trigger event occurs until the transfer register underflows. Therefore, the number loaded into the transfer register should be 1 less than the number of transfers desired. A control register bit determines whether each transfer is a byte or word of data.

The DMAC controller can be configured to perform a single transfer cycle, in which case, the transfers stop after the transfer register has underflowed. In repeat mode, the Destination Pointer register and the Transfer Counter register are reloaded after the Transfer Counter register has underflowed. In repeat mode, transfers will occur each time a trigger event occurs until the DMA enable bit is set inactive ("0").

4.0 Configuring the DMAC for Fixed Source, Forward Destination

To configure a DMAC channel, the following choices must be configured (the configuration for this example are shown in parentheses):

1. Select the DMA request cause (UART0 receive interrupt request) by setting DM0SL register to 0x0b.
2. Select fixed or forward source (fixed source) by setting bit-4 of DM0CON register to 0.
3. Select fixed or forward destination (forward destination) by setting bit-5 of DM0CON register to 1.
4. Select 8 or 16-bit transfers (8-bit transfers) by setting bit-0 of DM0CON register to 1.
5. Select a single transfer or multiple transfers (single transfer) by setting bit-1 of DM0CON register to 0.
6. Select source address for the transfer (UART0 receive buffer) by specifying SAR0.
7. Select the destination address for the transfer (Buffer address in RAM) by specifying DAR0.
8. Select the number of bytes to be transferred (10) by writing (9) in the Transfer Counter register.

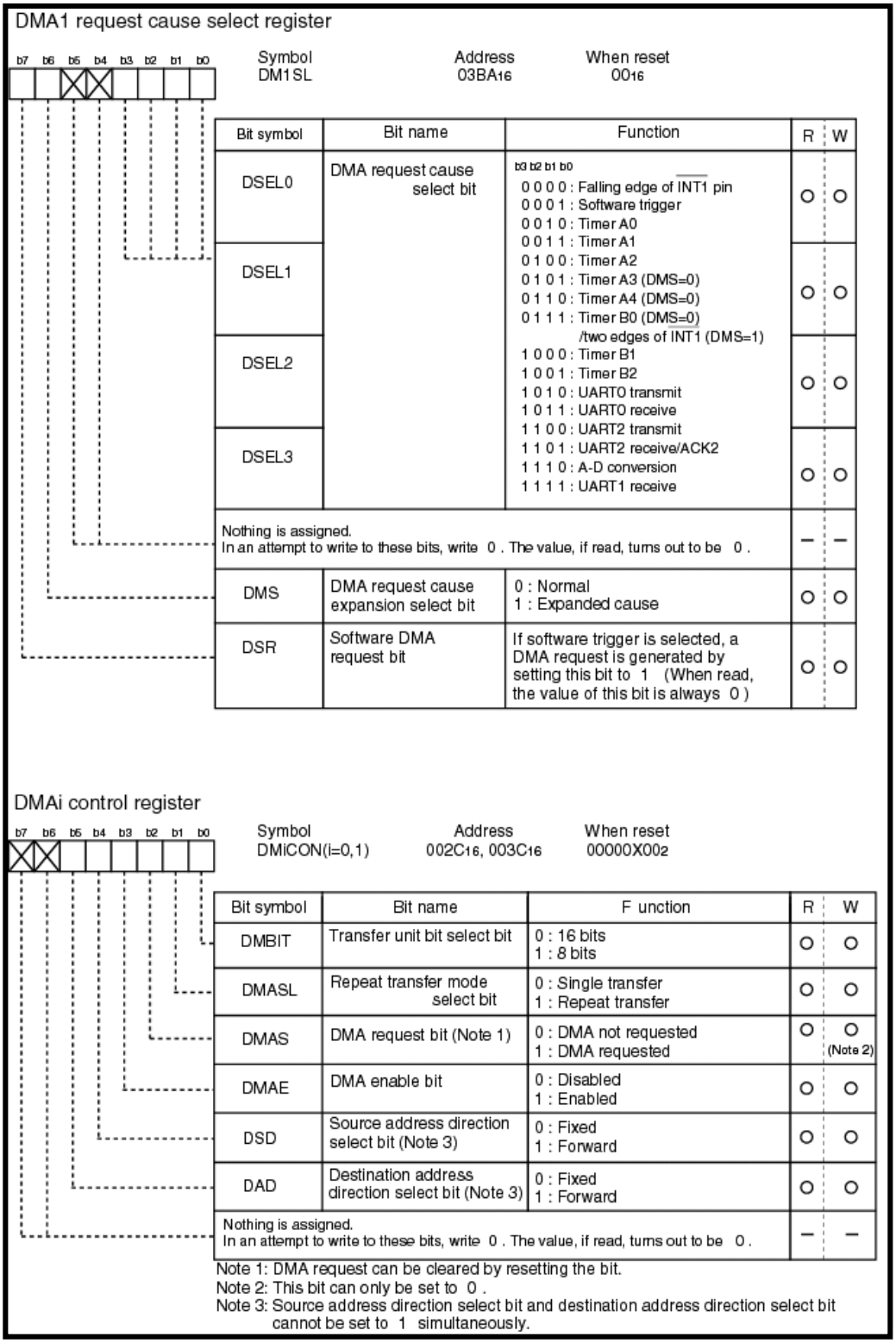
The registers that are used to configure and control the DMAC channels are shown in Figure 1 and Figure 2.

DMA0 request cause select register

Symbol: DM0SL | Address: 03B8₁₆ | When reset: 00₁₆

Bit symbol	Bit name	Function	R	W
DSEL0	DMA request cause select bit	b3 b2 b1 b0 0 0 0 0 : Falling edge of $\overline{\text{INT0}}$ pin 0 0 0 1 : Software trigger 0 0 1 0 : Timer A0 0 0 1 1 : Timer A1 0 1 0 0 : Timer A2 0 1 0 1 : Timer A3 0 1 1 0 : Timer A4 (DMS=0) /two edges of $\overline{\text{INT0}}$ pin (DMS=1) 0 1 1 1 : Timer B0 (DMS=0) 1 0 0 0 : Timer B1 (DMS=0) 1 0 0 1 : Timer B2 (DMS=0) 1 0 1 0 : UART0 transmit 1 0 1 1 : UART0 receive 1 1 0 0 : UART2 transmit 1 1 0 1 : UART2 receive 1 1 1 0 : A-D conversion 1 1 1 1 : UART1 transmit	○	○
DSEL1			○	○
DSEL2			○	○
DSEL3			○	○
Nothing is assigned. In an attempt to write to these bits, write 0 . The value, if read, turns out to be 0 .			–	–
DMS	DMA request cause expansion select bit	0 : Normal 1 : Expanded cause	○	○
DSR	Software DMA request bit	If software trigger is selected, a DMA request is generated by setting this bit to 1 (When read, the value of this bit is always 0)	○	○

Figure 1 DMA0 Request Cause Select Register



DMA1 request cause select register

b7 b6 b5 b4 b3 b2 b1 b0

Symbol	Address	When reset
DM1SL	03BA16	0016

Bit symbol	Bit name	Function	R	W
DSEL0	DMA request cause select bit	b3 b2 b1 b0 0 0 0 0 : Falling edge of INT1 pin 0 0 0 1 : Software trigger 0 0 1 0 : Timer A0 0 0 1 1 : Timer A1 0 1 0 0 : Timer A2 0 1 0 1 : Timer A3 (DMS=0) 0 1 1 0 : Timer A4 (DMS=0) 0 1 1 1 : Timer B0 (DMS=0) /two edges of INT1 (DMS=1) 1 0 0 0 : Timer B1 1 0 0 1 : Timer B2 1 0 1 0 : UART0 transmit 1 0 1 1 : UART0 receive 1 1 0 0 : UART2 transmit 1 1 0 1 : UART2 receive/ACK2 1 1 1 0 : A-D conversion 1 1 1 1 : UART1 receive	○	○
DSEL1			○	○
DSEL2			○	○
DSEL3			○	○
Nothing is assigned. In an attempt to write to these bits, write 0 . The value, if read, turns out to be 0 .			–	–
DMS	DMA request cause expansion select bit	0 : Normal 1 : Expanded cause	○	○
DSR	Software DMA request bit	If software trigger is selected, a DMA request is generated by setting this bit to 1 (When read, the value of this bit is always 0)	○	○

DMAi control register

b7 b6 b5 b4 b3 b2 b1 b0

Symbol	Address	When reset
DMiCON(i=0,1)	002C16, 003C16	00000X002

Bit symbol	Bit name	F unction	R	W
DMBIT	Transfer unit bit select bit	0 : 16 bits 1 : 8 bits	○	○
DMASL	Repeat transfer mode select bit	0 : Single transfer 1 : Repeat transfer	○	○
DMAS	DMA request bit (Note 1)	0 : DMA not requested 1 : DMA requested	○	○ (Note 2)
DMAE	DMA enable bit	0 : Disabled 1 : Enabled	○	○
DSD	Source address direction select bit (Note 3)	0 : Fixed 1 : Forward	○	○
DAD	Destination address direction select bit (Note 3)	0 : Fixed 1 : Forward	○	○
Nothing is assigned. In an attempt to write to these bits, write 0 . The value, if read, turns out to be 0 .			–	–

Note 1: DMA request can be cleared by resetting the bit.
Note 2: This bit can only be set to 0 .
Note 3: Source address direction select bit and destination address direction select bit cannot be set to 1 simultaneously.

Figure 2 DMA Control Registers

5.0 Reference

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com

E-mail Support

support_apl@renesas.com

Data Sheets

- M16C/26 datasheets, M30262eds.pdf

User's Manual

- M16C/20/60 C Language Programming Manual, 6020c.pdf
- M16C/20/60 Software Manual, 6020software.pdf
- Application Note: Writing interrupt handlers in C for the M16C
- MSV30262-SKP Quick start guide, Quick_Start_Guide_MSB30262.pdf
- MSV30262-SKP Users Manual, Users_Manual_MSV30262.pdf
- MDECE30262 Schematics, Schematics_MDECE30262_RevA.pdf

6.0 Software Code

The example program was written to run on the MSV30262 Starter Kit but could be modified to implement in a user application. The program is written in C and compiled with the KNC30 compiler. The program demonstrates using the DMA0 channel to transfer data from a UART receive buffer to a buffer established in RAM. The program performs a single transfer of 10 bytes from the UART0 receive buffer to memory. At the completion of the transfer a DMA0 interrupt request is generated. UART0 on the starter kit board is connected to a 9-pin D-sub connector that can be used to connect to a PC running a terminal program, such as HyperTerminal. With the program running, keys typed into the terminal program will be transferred by the DMAC to the memory buffer.

To run program perform the following steps:

1. Load program "dma_fwd_des.x30" using KD30.
2. Set up COM port of PC and configure HyperTerminal to operate at 9600 BAUD, 1 Stop Bit, and No Parity. Connect serial cable from COM port of PC to UART0 of starter kit board.
3. Execute program by pressing GO button on KD30.
4. Type 10 characters into the HyperTerminal window. Using Ram Monitor window of KD30, view received data starting at address 0x0410.

```
/***********************************************************************
*      File Name:   dma_fwd_dest.c
*
*      Content: DMA fixed source to forward destination
***********************************************************************

=======================================================================*
*      $Log:$
*=======================================================================*/

#include "sfr262.h"         /* SFR register definition */

// prototypes
#pragma Interrupt dma0_isr
void mcu_init (void);
void uart_init (void);
void dma_init (void);

// declare buffer
unsigned char buffer[10];

/***********************************************************************
Name:         main
Parameters:   None
Returns:      None
Description: Initializes the system and then loops forever.
***********************************************************************/
void main()
{
       mcu_init();                 // initialize mcu to full Xin system clock
                                   // 20 MHz in MSV30262 board

       uart_init ();               // initialize UART0 which is source of data
       dma_init ();                // initialize DMA registers
       dmae_dm0con = 1;            // enable DMA transfers
       asm ("fset I");             // enable interrupts
       re_u0c1 = 1;                // enable UART0 receive

       while (1);                  //loop forever
 }

/***********************************************************************
Name:         DMA_init
Parameters:   None
Returns:      None
Description: Initializes DMA for transfer from single source to multiple
             destinations set to transfer 16 bytes.  Transfers each time there is
             an interrupt request from UART0.
***********************************************************************/
```

```
void dma_init(void)
{
       dm0sl =       0x0b;

       /*       00001011;  DMA0 trigger select UART0 receive
                ||||||||------(DSEL0) the four bits (DSEL3-DSEL0) set the DMA
                |||||||-------(DSEL1) request cause.  set for UART0 receive
                ||||||--------(DESEL2)
                |||||---------(DSEL3)
                ||||---------- not used set to 0
                |||----------- not used set to 0
                ||------------(DMS) DMA request cause expansion bit to normal
                |-------------(DSR) set to 1 to generate DMA request
                               if software trigger selected */

    dm0con = 0X21;

    /*  00100001;  DMA0 trigger select UART0 receive
               ||||||||------(DMBIT) transfer unit bit select bit  1 = 8 bits
               |||||||-------(DMASL) repeat transfer mode  0 = single transfer
               ||||||--------(DMAS) DMA request bit can only be set to 0
               |||||---------(DMAE)DMA enable bit  0= disabled
               ||||----------(DSD)source address direction  0 = fixed
               |||-----------(DAD)destination address direction 1 = forward
               ||------------ not used set to 0
               |------------- not used set to 0 */

       sar0 = (unsigned long)&u0rb;     //  set source to address of uart0
                                        //  receiving buffer

       dar0 = (unsigned long)&buffer[0];//  set destination register to
                                        //  beginning of buffer

       tcr0 = 0x9;                      //  set transfer counter to transfer 10
                                        //  bytes
                                        //  (number of transfers desired -1)
       dm0ic = 0x04;                    //  set interrupt priority for DMA to 4

}

/************************************************************************
Name:         dma0_isr
Parameters:   None
Returns:      None
Description: This service routine is entered after the completion of the DMA
             transfer.
************************************************************************/
void dma0_isr(void)
{


}
```

```
/*****************************************************************************
Name:         uart_init
Parameters:   None
Returns:      None
Description: Initializes uart for 9600 baud, 1 stop bit no parity.
*****************************************************************************/
void uart_init(void)
{
   int  dummy;

   //  Configure Uart0 for 9600 baud, 8 data bits, 1 stop bit, no parity

       u0mr = 0x05;               // set mode register
       u0c0 = 0x10;               // set control register
       u0brg = 0x81;              // set bit rate generator
                                  // (20Mhz/16/9600)-1

       dummy = u0rb;              // clear receive buffer by reading

       s0tic = 0x00;              // disable UART0 interrupts
}

/*****************************************************************************
Name:         mcu_init
Parameters:   None
Returns:      None
Description: Initializes mcu for full Xin system clock - 20 MHz in MSV30262 board
*****************************************************************************/
void mcu_init(void){ //Initialize mcu for sull speed (20MHz) operation

   prc0 = 1;          /* Unlock CM0 and CM1 */
   cm0 = 0x08;        /* Enable divider selected by CM1 */
   cm1 = 0x20;        /* Select no division, high Xin drive */
   cm2 = 0x0;         /* disable stop detection, main clock - Xin */
   prc0 = 0;          // Lock the System Clock Control Register
}
```

Keep safety first in your circuit designs!

- Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
  Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
- Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
  The information described here may contain technical inaccuracies or typographical errors.
  Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
  Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
  Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.