M16C/62

Using the M16C/62 DMAC in Forward Destination Mode

1.0 Abstract

The following article introduces and shows an example of how to use the DMAC function of the M16C/62 with a fixed source address and forward counting destination address.

2.0 DMAC

The Renesas M16C/62 is a 16-bit MCU based on the M16C CPU core, with 256 KB of user Flash. 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 a cycle stealing method.

Each DMAC controller is capable of transferring data to or from a fixed address to any other address in the 1Mbyte address space. The DMAC controllers can automatically transfer 128k bytes of data, using word (16-bit) 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

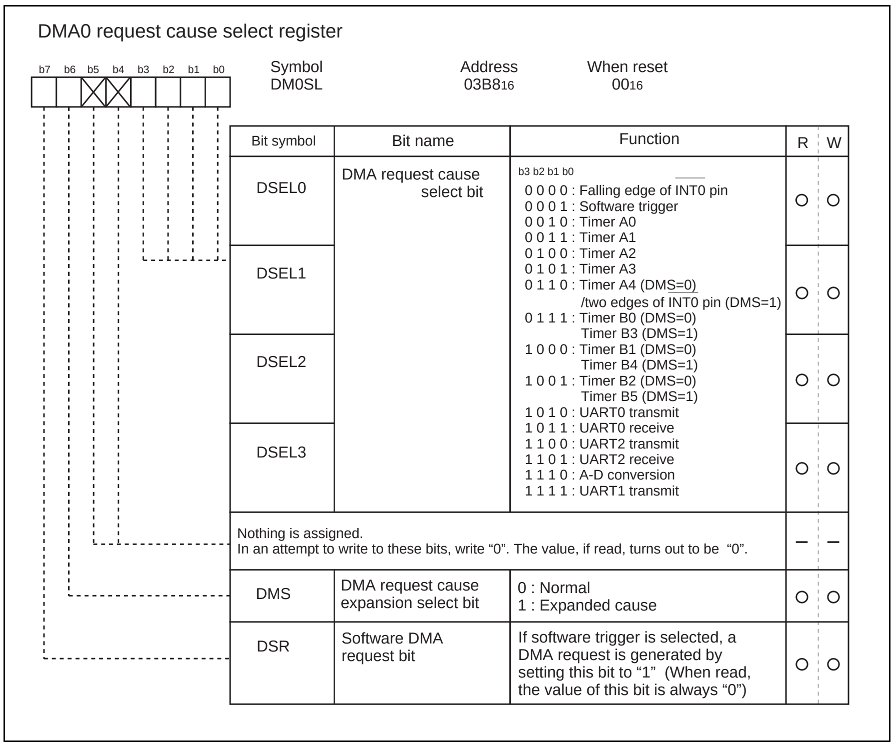
In the fixed source address, forward counting address mode, the DMAC controller will transfer bytes or words from a fixed source address to an incrementing destination address (that increments after each transfer). The transfers can be either bytes or words. Loading a value into the transfer count register controls the number of automated transfers. Transfers will continue to occur each time the DMAC triggers events until the transfer register underflows, therefore the number loaded into the 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 the repeat mode the Destination Pointer register and the Transfer Counter register are reloaded after the Transfer Counter register has underflowed. In the 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 configurations for this example are shown in parentheses):

1. Select the DMA request cause (UART0 receive interrupt request).
2. Select fixed or forward source (fixed source).
3. Select fixed or forward destination (forward destination).
4. Select 8 or 16-bit transfers (8-bit transfers).
5. Select a single transfer or multiple transfers (single transfer).
6. Select source address for the transfer (UART0 receive buffer).
7. Select the destination address for the transfer (Buffer address in RAM).
8. Select the number of bytes to be transferred (10).

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



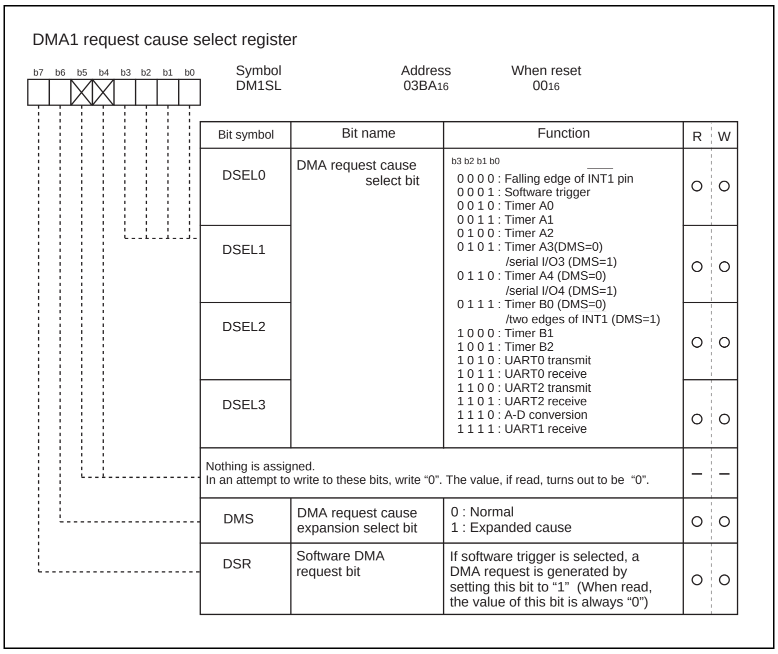
DMA0 request cause select register

b7 b6 b5 b4 b3 b2 b1 b0

Symbol	Address	When reset
DM0SL	$03B8_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W
DSEL0	DMA request cause select bit	b3 b2 b1 b0 0 0 0 0 : Falling edge of 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 INT0 pin (DMS=1) 0 1 1 1 : Timer B0 (DMS=0) Timer B3 (DMS=1) 1 0 0 0 : Timer B1 (DMS=0) Timer B4 (DMS=1) 1 0 0 1 : Timer B2 (DMS=0) Timer B5 (DMS=1) 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

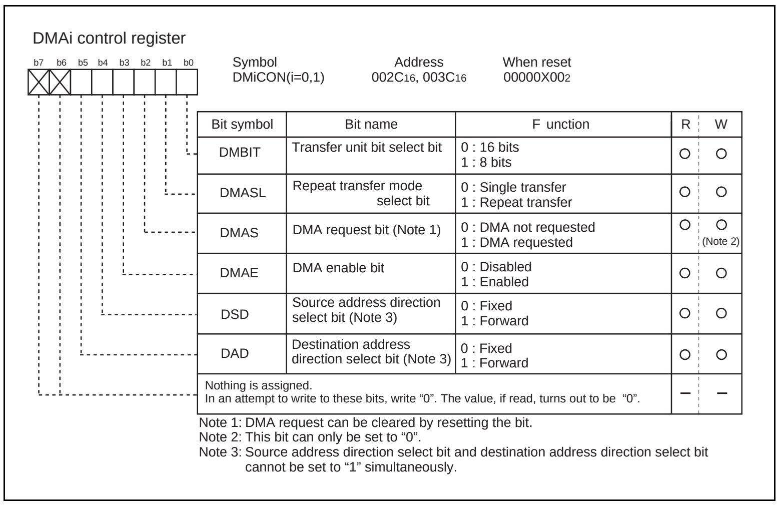


DMA1 request cause select register

b7 b6 b5 b4 b3 b2 b1 b0

Symbol DM1SL　Address $03BA_{16}$　When reset 00_{16}

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) /serial I/O3 (DMS=1) 0 1 1 0 : Timer A4 (DMS=0) /serial I/O4 (DMS=1) 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 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 DMiCON(i=0,1)　Address $002C_{16}$, $003C_{16}$　When reset $00000X00_2$

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

6.0 Reference

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com

E-mail Support

support_apl@renesas.com

Data Sheets

- M16C/62 datasheets, 62aeds.pdf

7.0 Software Code

The example program was written to run on the MSV1632 Starter Kit but could be modified to implement in a user application. The program is written in C (the NC30 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 DB 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.

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

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

#include "sfr62a.h"                /* SFR register definition */

// prototypes
#pragma Interrupt dma0_isr

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()
{
        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
                                            //       transmit 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 int
                                          //       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 = 0x67;                        // set bit rate generator
                                              //  (16Mhz/16/9600)-1
         dummy = u0rb;                        // clear receive buffer by reading
         s0tic = 0x00;                        // disable UART0 interrupts
}
```