To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

M16C/26

Using the DMAC with a Forward Source

1.0 Abstract

The following article introduces and shows an example of how to use the DMAC function of the M16C/26 with a forward counting source address and fixed 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 (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 Forward Source, Fixed Destination Description

In the forward source counting address, fixed destination address mode, the DMAC controller will transfer bytes or words from an incrementing source address (increments after each transfer) to a fixed destination address. 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 trigger event occurs 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.

When the DMAC controller is configured to perform a single transfer cycle, the DMAC becomes disabled after the transfer register underflows. In repeat mode, the Source Pointer register and the Transfer Counter register are reloaded with their initial values after the Transfer Counter register underflows and the DMAC remains active. Therefore, 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 Forward Source, Fixed 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 by setting DM0SL register to 0x0a (UART0 receive interrupt request).
2. Select fixed or forward source (forward source) by setting bit-4 of DMCON register to 1.
3. Select fixed or forward destination (fixed destination) by setting bit-5 of DM0CON register to 0.
4. Select 8 or 16-bit transfers (8 bit transfers) by setting bit-0 of DM0CON register to 1.
5. Select a single transfer (single transfer) by setting bit-1 of DM0CON register to 0.
6. Select the source address for the transfer (Buffer address in RAM) by specifying SAR0.
7. Select the destination address (UART0 transmit buffer address) for the transfer 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

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 $\overline{INT0}$ 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 $\overline{INT0}$ 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	O	O
DSEL1			O	O
DSEL2			O	O
DSEL3			O	O
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	O	O
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)	O	O

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	$03BA_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W
DSEL0 DSEL1 DSEL2 DSEL3	DMA request cause select bit	b3 b2 b1 b0 0 0 0 0 : Falling edge of $\overline{INT1}$ 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 $\overline{INT1}$ (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	O	O
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	O	O
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)	O	O

DMAi control register

b7 b6 b5 b4 b3 b2 b1 b0

Symbol	Address	When reset
DMiCON(i=0,1)	$002C_{16}$, $003C_{16}$	$00000X00_2$

Bit symbol	Bit name	Function	R	W
DMBIT	Transfer unit bit select bit	0 : 16 bits 1 : 8 bits	O	O
DMASL	Repeat transfer mode select bit	0 : Single transfer 1 : Repeat transfer	O	O
DMAS	DMA request bit (Note 1)	0 : DMA not requested 1 : DMA requested	O	O (Note 2)
DMAE	DMA enable bit	0 : Disabled 1 : Enabled	O	O
DSD	Source address direction select bit (Note 3)	0 : Fixed 1 : Forward	O	O
DAD	Destination address direction select bit (Note 3)	0 : Fixed 1 : Forward	O	O
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 or MSV-Mini26-SKP Quick start guide
- MSV30262-SKP or MSV-Mini26-SKP Users Manual
- MDECE30262 or MSV-Mini26-SKP Schematic

6.0 Software Code

The example program was written to run on the MSV3062 Starter Kit but could be modified to implement in a user application. The program is written in C and compiled using the KNC30 compiler. The program demonstrates using the DMA0 channel to transfer data from a memory buffer to the UART0 transmit buffer. The program performs a single transfer of 10 bytes to UART0. 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, the data contained in the DATA array will be sent to the terminal program and appear on the PC screen.

To run program perform the following steps:

1. Load program "dma_fs.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 and "0123456789" will be displayed in the HyperTerminal program window.

```
/******************************************************************************
*
*       File Name: dma_fwd_src.c
*
*       Content:  DMAC from a memory buffer to UART transmit buffer
*==============================================================================
*
*       $Log:$
*==============================================================================
*/

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

#pragma Interrupt  dma0_isr

// prototypes
void mcu_init (void);
void uart_init (void);
void DMA_init (void);

unsigned const char *ptr_string;
unsigned const char data[]= "0123456789" ;

/****************************************************************************
Name:         main
Parameters:   None
Returns:      None
Description: Initializes the system and then loops forever.
*************************************************************************/
void main()
       {
       ptr_string= &data[0];

       mcu_init();           // initialize mcu to full Xin system clock
                             // 20 MHz in MSV30262 board
       uart_init ();         // initialize UART0 and pre-load first character
                             // in transmit buffer
       DMA_init ();          // initialize DMA registers
       dmae_dm0con = 1;      // enable DMA transfers
       asm ("fset I");       // enable interrupts
       te_u0c1 = 1;          // enable UART0 transmit

       while (1);            //loop forever
 }
/****************************************************************************
Name:         DMA_init
Parameters:   None
Returns:      None
Description: Initializes DMA for transfer from forward source to fixed destinations

       Set DMAC0 for 10 byte transfers from memory to UART0  transmit buffer
****************************************************************************/
```

```
void DMA_init(void)
{
       dm0sl =  0x0a;   /* DMA0 trigger select UART0 transmit
                00001010;
                ||||||||------(DSEL0) the four bits (DSEL3-DSEL0) the DMA
                |||||||-------(DSEL1) request cause set for UART0 transmit
                ||||||--------(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 = 0X11;       /* DMA0 single transfer, 8 bit mode, forward source,
                                                  fixed destination /*
               00001011;
               ||||||||------(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  1 = forward
               |||-----------(DAD)destination address direction 0 = fixed
               ||------------not used set to 0
               |-------------not used set to 0 */

       dar0 = (unsigned long)&u0tb;       //  set destination register to address
                                          //  of uart0 transmit buffer

       sar0 = (unsigned long)ptr_string;  //  set source register to address of
                                          //  beginning of data buffer
       tcr0 = 0x9;                        //  set transfer counter for 10 transfers
                                          //  (number of transfers -1)

       dm0ic = 0x04;                      //  set interrupt priority for DMA0
                                          //  interrupt 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
*****************************************************************************/
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

       u0tb = 0x20;                // Place data in the transmit buffer
                                   // so when the transmit enable bit is set
                                   // later on, this first byte will be sent
                                   // out and cause a UART transmit interrupt
                                   // to occurs triggering the DMAC
       s0tic = 0x00;               // Disable UART0 receive interrupt,

}

/*****************************************************************************
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
}
```