M16C/62

Using the M16C/62 Timer in PWM Mode

1.0 Abstract

PWMs, or Pulse Width Modulators, are useful in DC motor control, actuator control, synthesized analog output, piezo transducers, etc. PWMs produce a signal of (typically) fixed frequency and vary the width of the pulse to control a peripheral. The following article describes how to use the M16C/62 A Timers as Pulse Width Modulators, referred to as Pulse Width Modulation Mode.

2.0 Introduction

The M16C/62 is a 16-bit MCU, based on the M16C CPU core, with features including 10-bit A/D, D/A, UARTS, timers, DMA, etc., and up to 256KB of user flash. The MCU has 5 timer A's, all of which can operate as PWMs. Timer A has the following additional modes of operation:

- Event Counter Mode
- Timer Mode
- One-Shot Mode

Figure 1 illustrates the operation of timer A. The remainder of this document will focus on setting up timers A0 and A1 in PWM Mode.

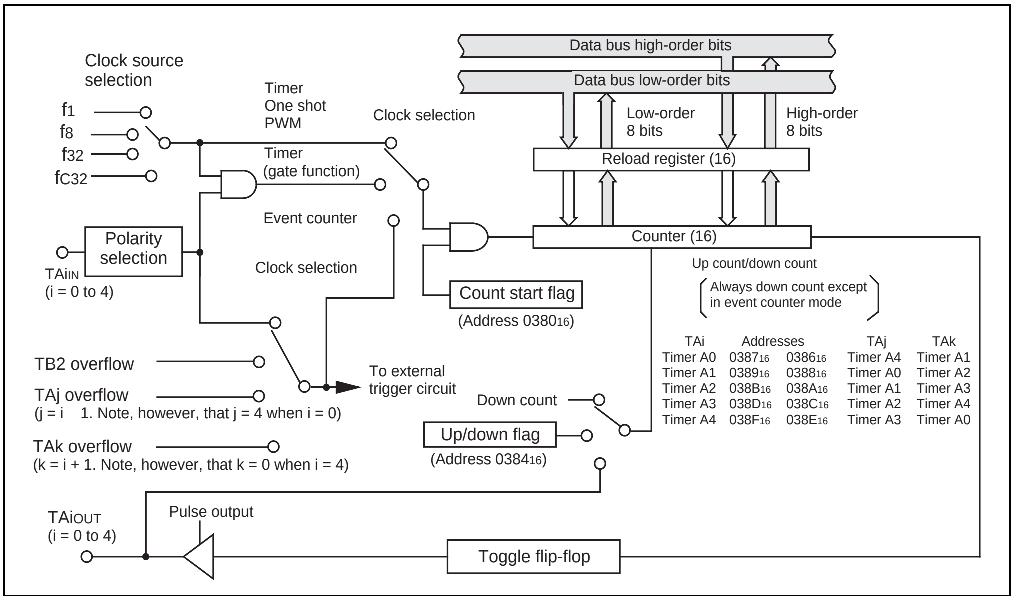

Figure 1 Block Diagram of Timer A

3.0 PWM Mode Description

PWM Mode has two "sub" modes: 16 bit and 8 bit. In 16-bit mode, the value of the 16-bit Ai Timer register determines the pulse width and the frequency is fixed to f_{in} / 65535. Therefore the maximum frequency of the 16-bit PWM (assuming 16 MHz at Xin) is approximately 244 Hz. In 8-bit mode, the high order 8 bits are used to determine the pulse width and the lower 8 bits the frequency, where the frequency is f_{in} / 255 or a maximum frequency of approximately 62,745 Hz. Note that the PWM output is free running and interrupts need not be enabled or serviced. Also the user has the option of triggering the start of the PWM output via the timer's Tai_{IN} pin.

The pulse width (high level) of the 16-bit PWM is:

pulse width (high) = n/ f_{in} , or % duty = n / 65535 * 100,

where n is the value loaded into the Ai counter register.

The pulse width (high level) of the 8-bit PWM is:

pulse width (high) = n * (m+1) / f_{in} , or n/ (255 * f_{PWM}), % duty = n / 255 * 100,

where n is the value loaded into the Ai counter register's high order address and m is loaded into the Ai counter register's low order address.

The pulse width can be changed at any time by writing to the Ai counter register, but during counting, the write affects only the reload register, and the counter register is updated on the next cycle.

Figure 2 and Figure 3 illustrate the timing for the PWMs.

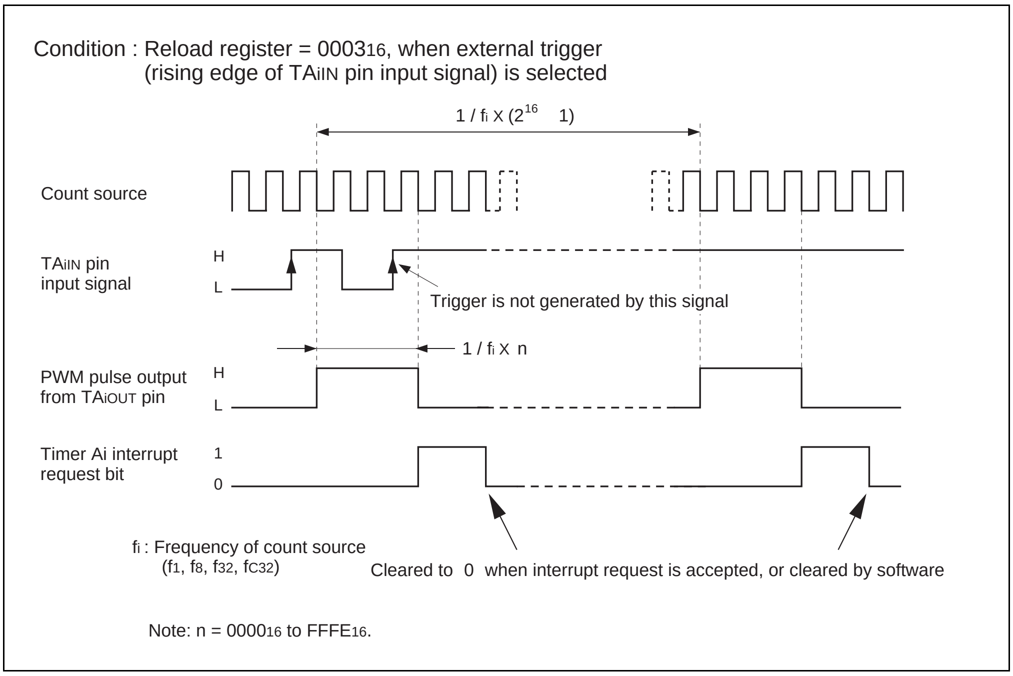

Figure 2 Example of How a 16-bit Pulse Width Modulator Operates

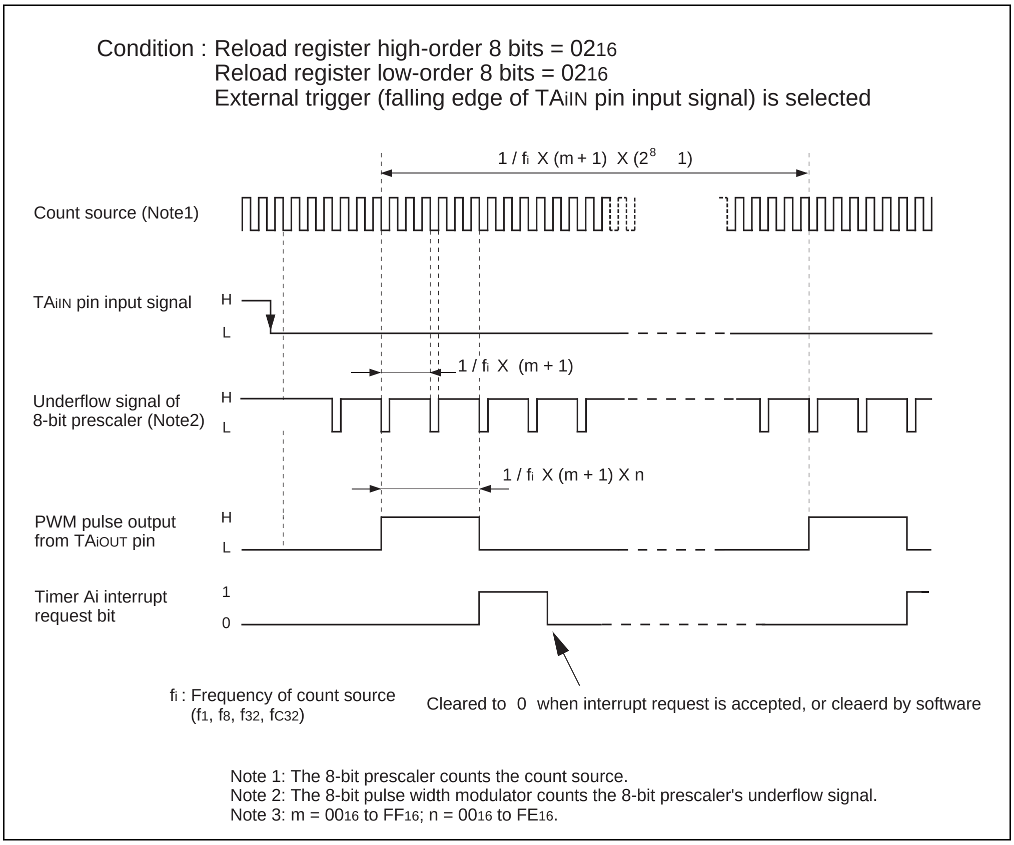

Figure 3 Example of How an 8-bit Pulse Width Modulator Operates

4.0 Configuring PWM Mode

To configure a timer for PWM mode:

1. Load the Timer Ai register with the pulse width value (16-bit) or frequency and pulse width value (8-bit).

Note: For 16-bit 65535 is not valid. For 8-bit 255 is not valid.

---

2. Load the Timer Ai Mode register, TaiMR:
   - Select PWM mode: bits TMOD0 and TMOD1 = 1.
   - Set the MR0 bit = 1 for PWM Mode.
   - Clear the MR1 bit for a falling edge external trigger, or set it for rising edge.
   - Clear the MR2 bit to use the ‘count start flag’ as a trigger, or set it for external trigger.
   - Clear the MR3 bit for 16-bit PWM, and set it for 8-bit PWM.
   - Select the clock source (f1, f/8, f/32, or fc/32): bits TCK0,TCK1 register.
3. Load the Timer Interrupt Control register (TAiIC) with an interrupt priority level, (ILVL) (load zero if interrupts are not required)..
4. Enable interrupts if required (I flag set).
5. Set the ‘start count’ flag bit, TaiS in the ‘count start flag’ register, TABSR.

It is not necessary to perform these steps in the order listed, but the count register should be loaded before the 'start count' flag is set. Also, the priority level should not be modified when there is a possibility of an interrupt occurring.

The required registers are shown in Figure 4 through Figure 7.

Timer Ai mode register

b7	b6	b5	b4	b3	b2	b1	b0
					1	1	1

Symbol	Address	When reset
TAiMR(i=0 to 4)	0396_{16} to $039A_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W
TMOD0	Operation mode select bit	b1 b0 1 1 : PWM mode	O	O
TMOD1			O	O
MR0	1 (Must always be "1" in PWM mode)		O	O
MR1	External trigger select bit (Note 1)	0: Falling edge of TAi IN pin's input signal (Note 2) 1: Rising edge of TAi IN pin's input signal (Note 2)	O	O
MR2	Trigger select bit	0: Count start flag is valid 1: Selected by event/trigger select register	O	O
MR3	16/8-bit PWM mode select bit	0: Functions as a 16-bit pulse width modulator 1: Functions as an 8-bit pulse width modulator	O	O
TCK0	Count source select bit	b7 b6 0 0 : f_1 0 1 : f_8 1 0 : f_{32} 1 1 : f_{C32}	O	O
TCK1			O	O

Note 1: Valid only when the TAiIN pin is selected by the event/trigger select bit (addresses 0382_{16} and 0383_{16}). If timer overflow is selected, this bit can be "1" or "0".
Note 2: Set the corresponding port direction register to "0".

Figure 4 Timer Ai Mode Register in Pulse Width Modulation Mode

Timer Ai register (Note)

(b15) b7	(b8) b0 b7	b0

Symbol	Address	When reset
TA0	0387_{16}, 0386_{16}	Indeterminate
TA1	0389_{16}, 0388_{16}	Indeterminate
TA2	$038B_{16}$, $038A_{16}$	Indeterminate
TA3	$038D_{16}$, $038C_{16}$	Indeterminate
TA4	$038F_{16}$, $038E_{16}$	Indeterminate

Function	Values that can be set	R	W
• Timer mode Counts an internal count source	0000_{16} to $FFFF_{16}$	O	O
• Event counter mode Counts pulses from an external source or timer overflow	0000_{16} to $FFFF_{16}$	O	O
• One-shot timer mode Counts a one-shot width	0000_{16} to $FFFF_{16}$	X	O
• Pulse width modulation mode (16-bit PWM) Functions as a 16-bit pulse width modulator	0000_{16} to $FFFF_{16}$	X	O
• Pulse width modulation mode (8-bit PWM) timer low-order address functions as an 8-bit prescaler and high-order address functions as an 8-bit pulse width modulator	00_{16} to FE_{16} (Both high-order and low-order addresses)	X	O

Note: Read and write data in 16-bit units

Figure 5

Count start flag

b7 b6 b5 b4 b3 b2 b1 b0

Symbol	Address	When reset
TABSR	0380_{16}	00_{16}

Bit Symbol	Bit Name	Function	R	W
TA0S	Timer A0 count start flag	0 : Counting stops 1 : Counting starts	O	O
TA1S	Timer A1 count start flag		O	O
TA2S	Timer A2 count start flag		O	O
TA3S	Timer A3 count start flag		O	O
TA4S	Timer A4 count start flag		O	O
TB0S	Timer B0 count start flag		O	O
TB1S	Timer B1 count start flag		O	O
TB2S	Timer B2 count start flag		O	O

Figure 6

Interrupt control register (Note 2)

b7 b6 b5 b4 b3 b2 b1 b0

Symbol	Address	When reset
TBiIC (i = 0 to 4)	0055_{16} to 0059_{16}	$XXXXX000_2$

Bit Symbol	Bit Name	Function	R	W
ILVL0	Interrupt priority level select bit	b2 b1 b0 0 0 0 : Level 0 (interrupt disabled) 0 0 1 : Level 1 0 1 0 : Level 2 0 1 1 : Level 3 1 0 0 : Level 4 1 0 1 : Level 5 1 1 0 : Level 6 1 1 1 : Level 7	O	O
ILVL1			O	O
ILVL2			O	O
IR	Interrupt request bit	0 : Interrupt not requested 1 : Interrupt requested	O	O Note 2
Nothing is assigned. Write "0" when writing to these bits. The value is "0" if read.			–	–

Note 1: Rewrite the interrupt control register only if it will not generate an interrupt request for that register. See precautions listed at end of the Interrupts chapter.
Note 2: This bit can only be accessed for reset (=0) Set (=1) cannot be accessed.

Figure 7

5.0 Reference

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com

E-mail Support

support_apl@renesas.com

Data Sheet

- M16C/62 datasheets, 62aeds.pdf

User's Manual

- NC30 Ver. 4.0 User's Manual, NC30UE.PDF
- M16C/60 and M16C/20 C Language Programming Manual, 6020EC.pdf
- M16C/62 User's Manual, 62eum.pdf

6.0 Software Code

Following is a simple program written for Renesas' NC30 compiler to illustrate how to set up a 16-bit PWM Mode on timer A0, and an 8-bit PWM on timer A1. This program runs on the MSV1632/62 Starter Kit Board.

To become familiar with the PWM, try changing the clock source or switch to a different timer (e.g., TA2, TA3 etc.).

```
/*****************************************************************************
*
*       File Name: pwm.c
*
*       Content: Example program using Timer A in "PWM mode" .This program
*             is written for the 'Timer A PWM Mode' application note. Timer
*             A0 is set up as an 8bit PWM (output on TA0out or P7.0),
*             timer A1 set up as a 16bit PWM (output on TA1out or P7.2), and
*             the output varied. The frequency of the 16bit PWM is set to 244Hz,
*             and the 8bit PWM is set to 7843Hz.  The outputs can be viewed
*             on a scope. This program works with the MSV1632/62 starter kit board,
*             but should work with any M16C/62 system with P7.0 and P7.2
*             available.
*
*        Compiled with NC30 ver. 3.20.00.
*
*        All timing based on 16 Mhz Xtal
*
*       Copyright, 2003 Renesas Technology Corporation, Inc.
*=============================================================================
*       $Log:$
*===========================================================================*/
```

```
#include "sfr62.h"

#define  PWM8_CONFIG 0x67  /*  01100111 value to load into timer A0 mode register
                               ||||||||_ TMOD0,TMOD1: PWM MODE SELECTED
                               ||||||____ MR0:         = 1 FOR PWM MODE
                               |||||_____ MR1,MR2:     EXT TRIGGER NOT SELECTED
                               |||_______ MR3:         SET TO 1 FOR 8BIT PWM
                               ||________ TCK0,TCK1:   F DIVIDED BY 8 SELECTED */

#define  PWM16_CONFIG 0x07  /* 00000111 value to load into timer A1 mode register
                               ||||||||_ TMOD0,TMOD1: PWM MODE SELECTED
                               ||||||____ MR0:         = 1 FOR PWM MODE
                               |||||_____ MR1,MR2:     EXT TRIGGER NOT SELECTED
                               |||_______ MR3:         SET TO 0 FOR 16BIT PWM
                               ||________ TCK0,TCK1:   F DIVIDED BY 1 SELECTED */

#define  CNTR_IPL 0x00   // TA0 AND TA1 interrupt priority level
int time_cnt;  // loop counter

//prototypes
 void init(void);


/******************************************************************************
Name:     main()
Parameters: none
Returns:  nothing
Description: initializes variables, then goes into an infinite loop. A
             simple delay loop is used to "slowly" increase the pulse widths
**************************************************************************** */

void main (void)
{ int pwm16;
  char pwm8;

  pwm16 = 100;       // 16bit PWM changes slowly, so give it a reasonable width to
                     // start
  pwm8 = 0;
  time_cnt = 0;
  init();
  while (1)
  {
       while(time_cnt <10000)
              time_cnt++;        // delay loop
```

```
      time_cnt = 0;
       pwm8++;                   // cannot read-modify-write on timer counter
       pwm16+=10;                // registers: while counting, value is
                                 // indeterminate.
       if (pwm8 > 0xfe)          // the 8bit PWM value cannot exceed 0xFE
              pwm8 = 0;
       if (pwm16 > 0xfffe)       // the 16bit PWM value cannot exceed 0xFFFE
              pwm16 = 100;
       ta0h = pwm8;
       ta1 =pwm16;
  }

}
/***************************************************************************
Name:  initial()
Parameters: none
Returns:  nothing
Description: Timer TA0 and TA1 setup for PWM mode.
************************************************************************** */
 void init()
  {
   ta0l = 0x00;    // PWM TA0 frequency set to 7,843 Hz

/* the following procedure for writing an Interrupt Priority Level follows that as
described in the M16C
   data sheets under 'Interrupts' */

// intialize TA0
   _asm ("   fclr i") ;          // turn off interrupts before modifying IPL
   ta0ic |= CNTR_IPL;        // use read-modify-write instruction to write IPL
   ta0mr = PWM8_CONFIG;
   _asm ("   fset i");

   ta0s = 1;  //start PWM


// intialize TA1
   _asm ("   fclr i") ;      // turn off interrupts before modifying IPL
   ta1ic |= CNTR_IPL;        // use read-modify-write instruction to write IPL
   ta1mr = PWM16_CONFIG;
   _asm ("   fset i");

   ta1s = 1;  //start PWM

  }
```