**Freescale Semiconductor**
Application Note

Document Number: EB651
Rev. 1.2, 10/2006

# MPC5500 eMIOS

## Avoiding Unexpected Module Operation

by: Bill Terry
MCD 32-Bit Applications Engineering
John West
MCD 32-Bit Systems Engineering

## 1 Introduction

The enhanced modular input/output subsystem (eMIOS) found in the MPC5500 family of devices provides multiple modes for each unified channel (UC), any of which may be selected depending on the requirements of the customer application. This bulletin provides a supplementary reference for programming and using certain modes.

Special attention is given to possible problems that may occur if the details of the operation of the module hardware are not fully understood. Where appropriate, pseudocode examples illustrate the recommended usage.

In addition to this bulletin, see the reference manual for the device being used.

**Contents**





freescale™ semiconductor

# 2 Modes of Operation

These sections contain information necessary for successful implementation of certain modes of operation. Generally, there are three types of software action (or inaction) that can cause problems:

- Counter wrap — Updating a register match value to a value less than the current value of the selected time base, can result in a UC output state that is static until the selected time base rolls over and the new match value is reached. Depending on the time base selected, this roll-over time can be as much as the time required for a 24-bit counter to roll over. In this document the time base roll-over problem is referenced as a counter wrap.
- Non-coherent A and B register values — Reading the A match register on the same clock cycle that the eMIOS hardware is attempting to update the A match register delays the update, resulting in the application obtaining incoherent data from the A and B registers.
- Volatile register/time base data — In certain continuous input modes, the application has a finite amount of time to read a value from the A and/or B registers or the UC internal counter before the data is lost due to a subsequent update by the eMIOS hardware.

**NOTE**

The MPC5553 and some later devices offer buffered modes for OPWM, OPWFM, MC, and OPWMC modes (see Section 2.3, “Output Pulse-Width Modulation (OPWM),” Section 2.4, “Output Pulse-Width Modulation and Frequency Modulation (OPWFM),” Section 2.7, “Modulus Counter (MC),” and Section 2.8, “Output Pulse-Width Modulation Center Aligned (OPWMC)”). The buffered modes can eliminate some or all of the software requirements described in the related sections and should be used if available. Refer to the reference manual for your device.

## 2.1 Input Pulse-Width Measurement (IPWM)

The IPWM mode allows the measurement of the width of a positive or negative pulse by capturing the counter value at the leading edge in register B and the counter value at the trailing edge in register A. The pulse-width is obtained by subtracting B from A.

When an application reads the A register on exactly the same clock cycle as a trailing edge event occurs, there is contention for access to the A register between the CPU and the eMIOS hardware. The CPU has higher priority and will access the A register, and the eMIOS update to the A register is delayed. Figure 1 shows the behavior in the error condition.

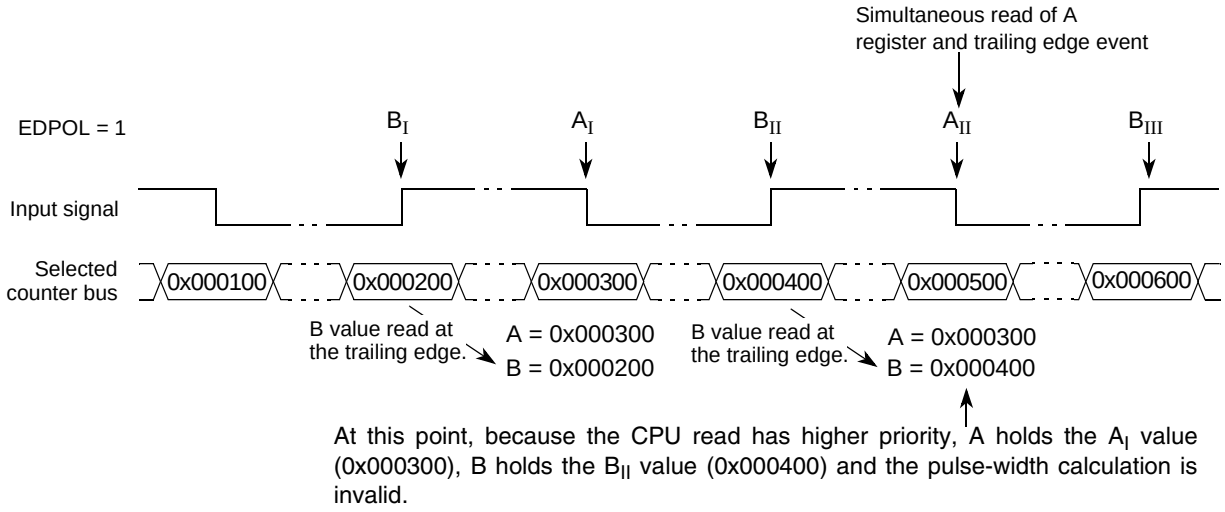



**Figure 1. IPWM Mode**

## 2.1.1 Pseudocode

This section contains a pseudocode example to avoid the error condition shown in Figure 1. The software monitors the UC flag and performs multiple reads of the A and B registers to ensure coherent values are obtained.

```
GetIPWM:
        if flag not set
                return;
        save MSR (machine state register)
        disable interrupts
        set b1 = B register
        set a1 = A register
        set a2 = A register
        set b2 = B register
        clear UC flag
                if a1 != a2
                        if b1 != b2
                                pulse width = a2 - b2
                        else
                                pulse width = a1 - b1
                else
                        pulse width = a2 - b2;
        if pulse width < 0 then a rollover occurred
                correct pulse width by adding 0x1000000 to

        restore old MSR value
```

## 2.2 Input Period Measurement (IPM)

The IPM mode allows the measurement of the period of an input signal by capturing the counter value on two consecutive rising edges or two consecutive falling edges. When an application reads A on exactly the same clock cycle as an edge event occurs, there is contention for access to the A register between the CPU and the eMIOS hardware. The CPU has higher priority and will access the A register, and the eMIOS update to the A register is delayed. Figure 2 shows the behavior in the error condition.

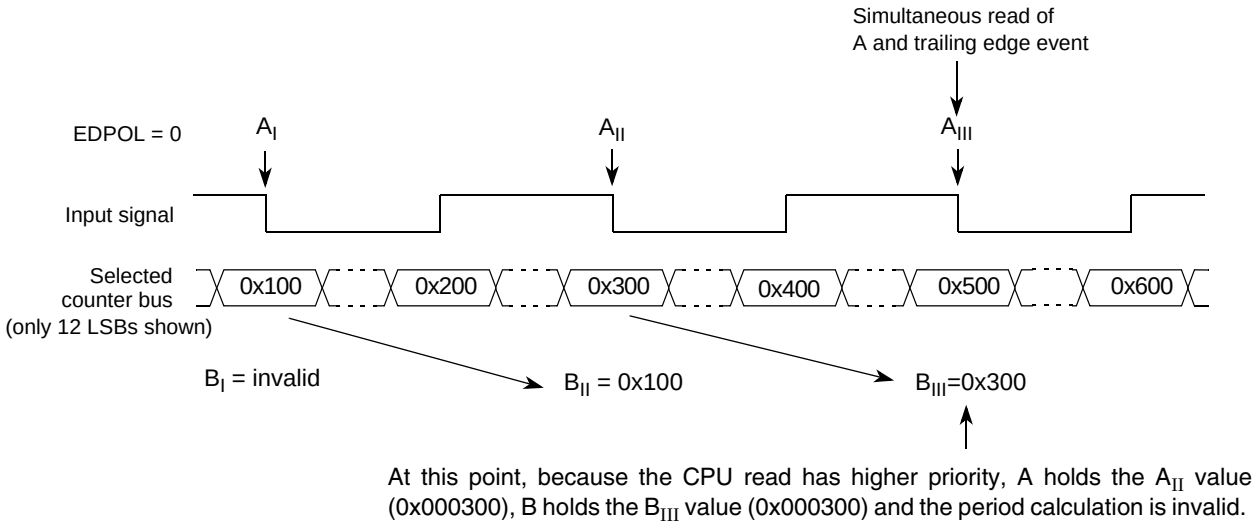



**Figure 2. IPM Mode**

### 2.2.1 Pseudocode

Use the pseudocode example to avoid the error condition shown in Figure 2. The software monitors the UC flag and performs multiple reads of the A and B registers to ensure coherent values are obtained.

```
GetIPM:
        if flag not set
                return;
        save MSR (machine state register)
        disable interrupts
        set b1 = B register
        set a1 = A register
        set a2 = A register
        set b2 = B register
        clear UC flag
                if a1 != a2
                        if b1 != b2
                                pulse width = a2 - b2
                        else
                                pulse width = a1 - b1
                else
                        pulse width = a2 - b2;
        if pulse width < 0 then a rollover occurred
                correct pulse width by adding 0x1000000
        restore old MSR value
```

## 2.3 Output Pulse-Width Modulation (OPWM)

The output pulse-width mode (OPWM) is typically used to generate a PWM output where the A match register determines the leading edge, and the B match register determines the trailing edge. OPWM mode may be configured for immediate update, where A and B values are updated as soon as they are written, or next period update mode, where the B update value is buffered until the next period.

This mode requires an ordered match to operate properly. That is, an A match must always occur before a B match can occur. Consequently, a match value written to the A (or B register if using immediate mode) that is greater than the current counter value will cause a counter-wrap condition (with output either asserted or unasserted) until the next match occurs. The application software should ensure that any immediate updates to the A or B register are compared to the current UC counter value and force the appropriate match if required.

If a next period update for the B register can be allowed by the application, the test for the B update value against the current counter value is not required. Figure 3 and Figure 4 illustrate the behavior for immediate updates to leading edge and trailing edge, respectively.

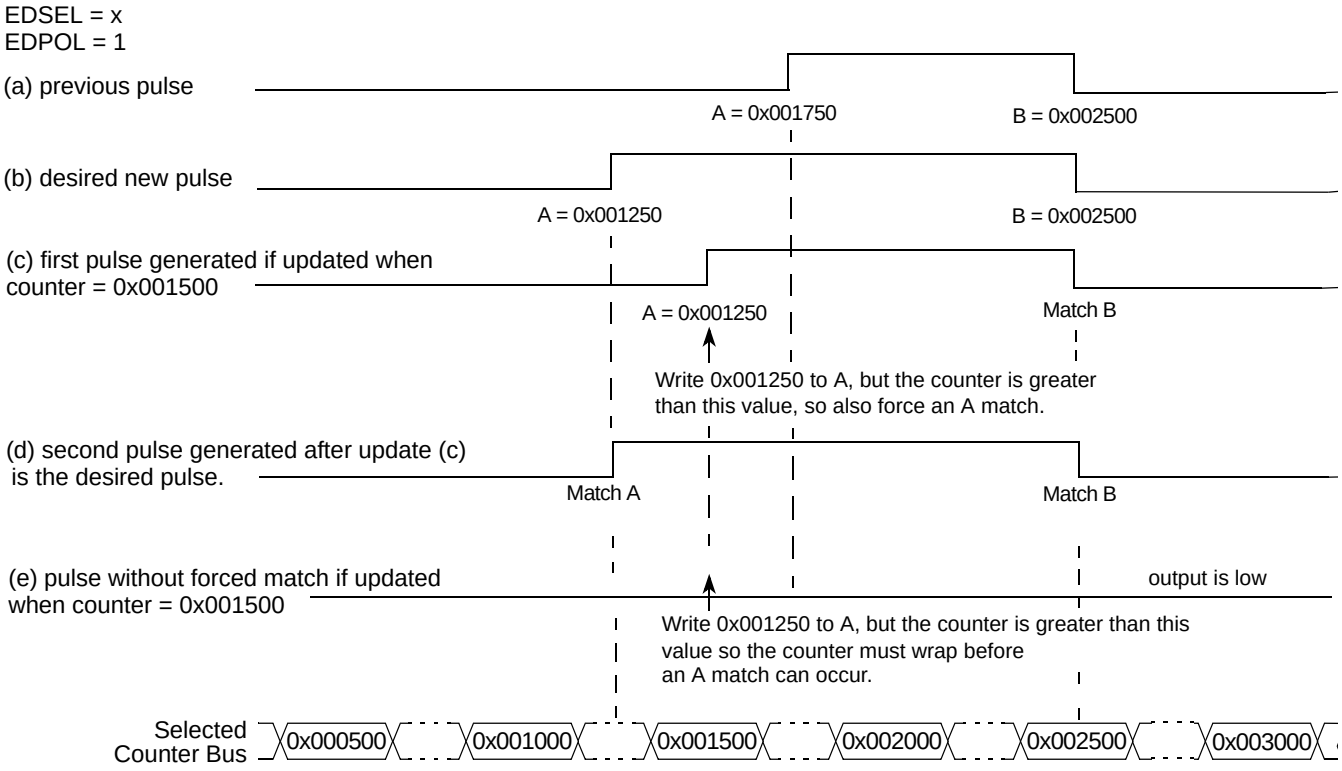



**Figure 3. OPWM Mode — Leading Edge Update (Immediate Update)**

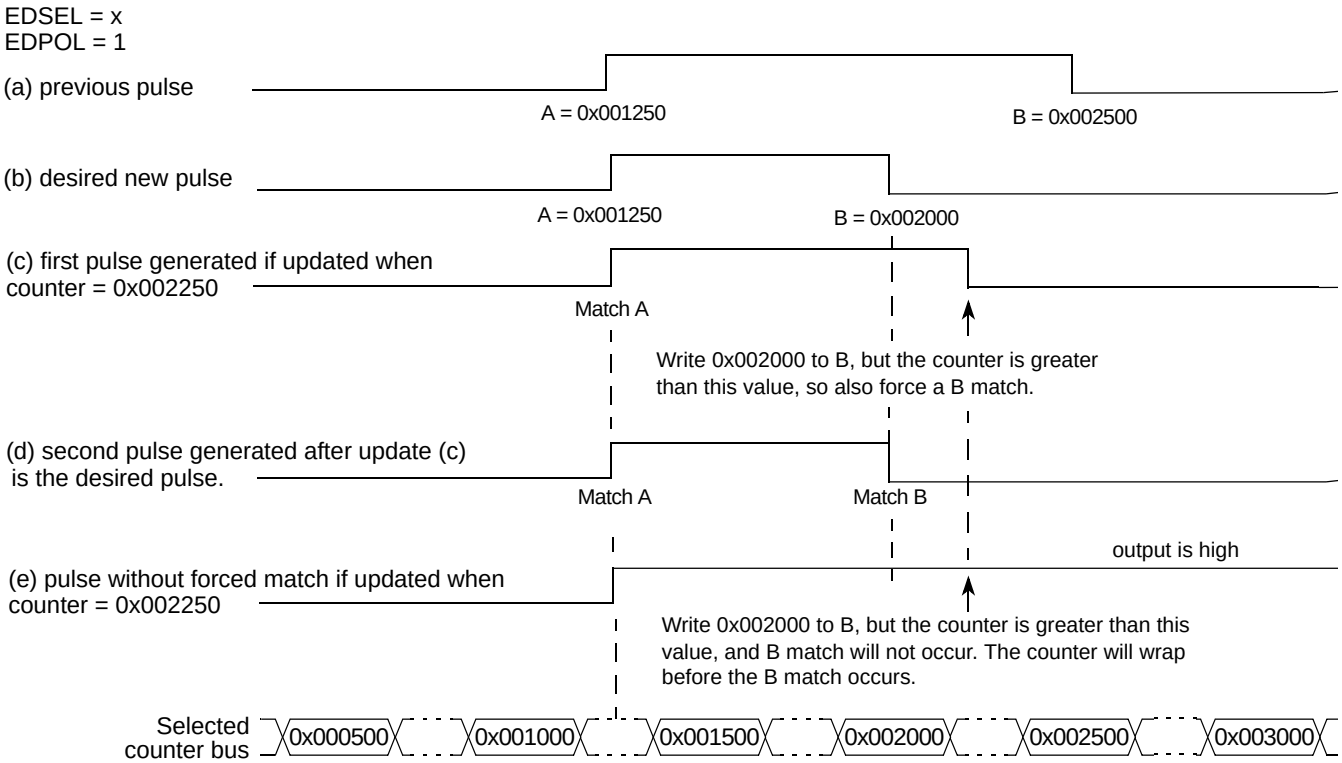



**Figure 4. OPWM Mode — Trailing Edge Update (Immediate Update)**

## 2.3.1 Pseudocode

This section contains a pseudocode example to avoid the error conditions shown in Figure 3 and Figure 4. Updates to both registers could be combined in a single function.

```
update OPWM leading edge:
        save MSR (machine state register)
        disable interrupts
        write the new A Register value
        if new A value < current counter value
                force an A match
        restore MSR

update OPWM trailing edge:
        save MSR (machine state register)
        disable interrupts
        write the new B Register value
        if new B value < current counter value
                force a B match
        restore MSR
```

## 2.4 Output Pulse-Width Modulation and Frequency Modulation (OPWFM)

The output pulse-width and frequency modulation mode (OPWFM) is typically used to generate a PWM output that may be dynamically controlled for variable period and pulse-width. In this mode, the A match register is used to determine the pulse-width, and the B match register controls the period. The UC counter resets at the end of every period (B match event). OPWFM mode may be configured for immediate update, where A and B values are updated as soon as they are written, or next period update mode, where the B update value is buffered until the next period.

This mode requires an ordered match to operate properly. That is, an A match must always occur before a B match can occur. Consequently, a match value written to the A (or B register if using immediate mode) that is less than the current counter value will cause a counter-wrap condition (with output either asserted or unasserted) until the next match occurs (See Figure 5 and Figure 8). The application software should ensure that immediate updates to the A or B register are compared to the current UC counter value and force the appropriate match if required.

### 2.4.1 Pulse-Width Updates

Figure 5 illustrates how the counter-wrap condition can occur with pulse-width updates, and how it can be avoided.

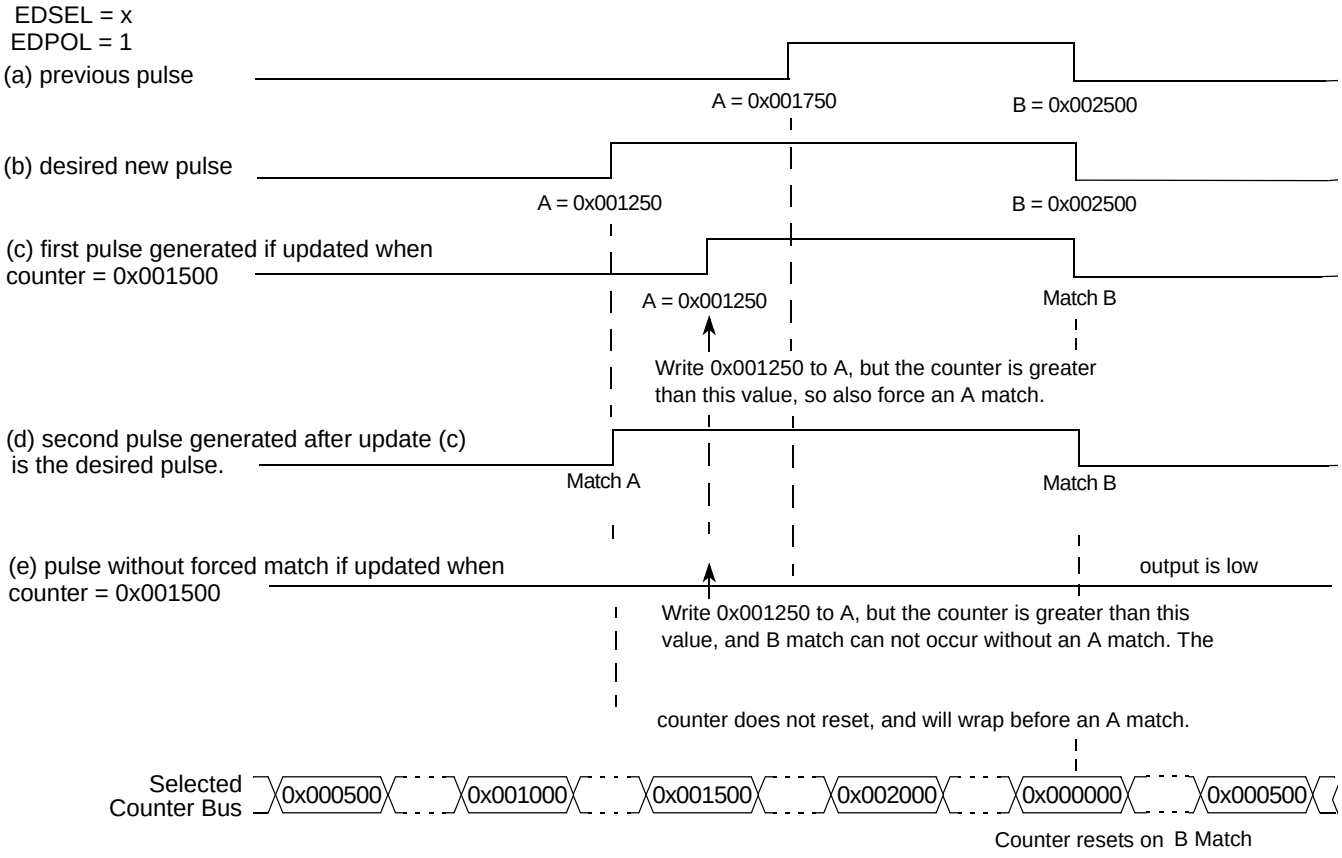
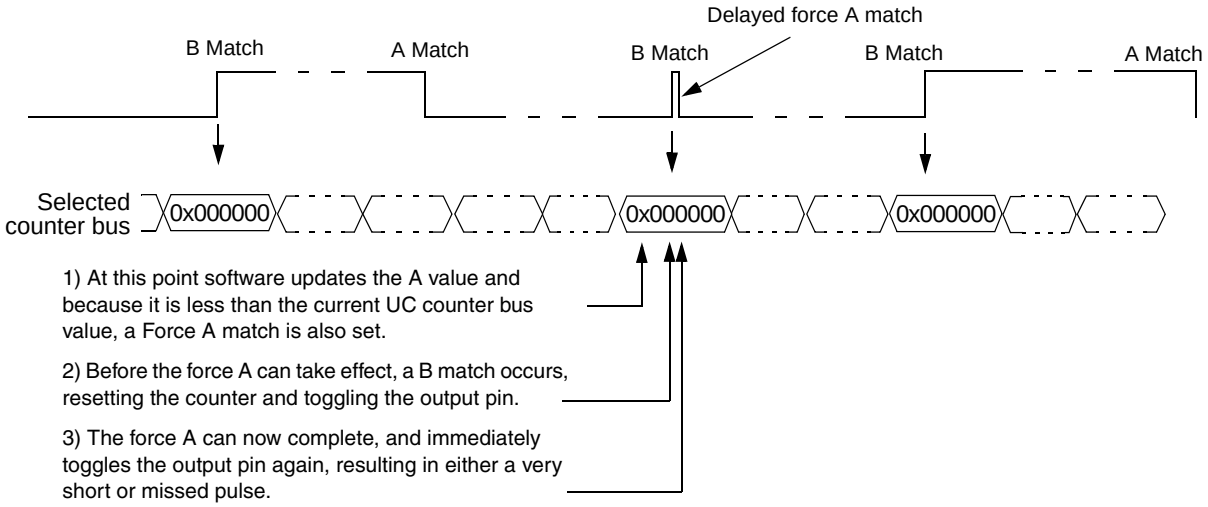
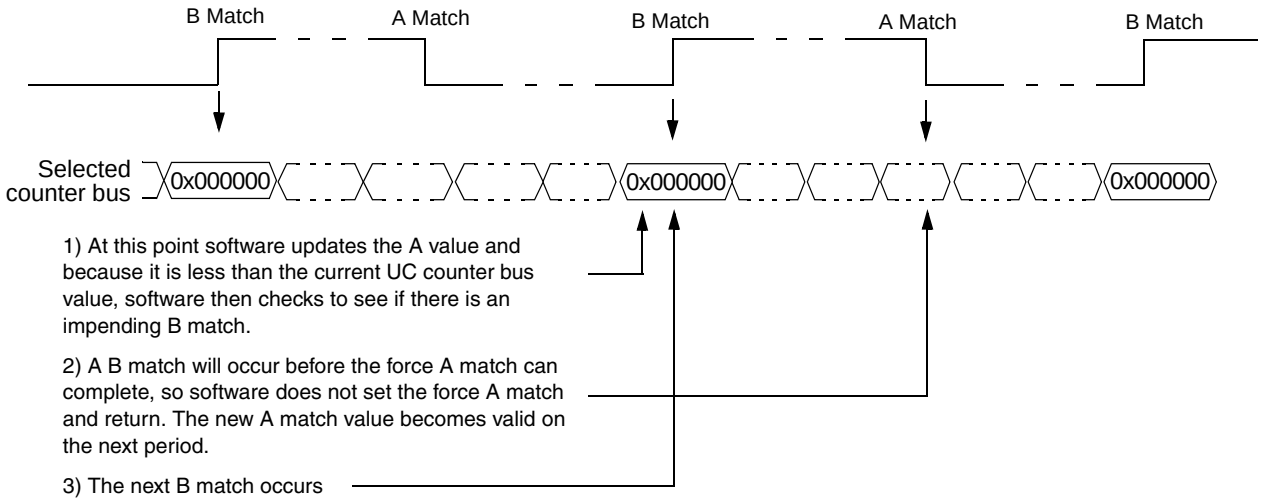



**Figure 5. OPWFM Mode — Asynchronous Pulse-Width Update (Part One)**

In case an update to the A register requires a forced match, care must also be taken that an impending B match will not occur before the forced A match has time to complete, resulting in a missed, or very short, pulse. Figure 6 illustrates how a missed pulse can occur and how it can be avoided.




a) Missed pulse due to a delayed force A match




b) Correct force A match procedure

**Figure 6. OPWFM Asynchronous Pulse-Width Update (Part Two)**

To avoid the problem illustrated in Figure 6a, the time in UC time base clocks that is required for the code executing the Force A match to complete must be known. Section 2.4.1.1, “Calculating Code Execution Time,” contains example code illustrating the software method and the necessary calculations.

#### 2.4.1.1 Calculating Code Execution Time

The subsequent code is an example of how a typical application may asynchronously update the OPWFM pulse-width.

Example Code[1] (UC time base is running at 20MHz, Fsys = 80MHz):

```
 1 #define LOOKAHEAD_BUF 8
 2
 3 update_pulse_width( uint32_t new_A ){
 4
 5     uint32_t temp;
 6
 7     /* disable irqs if enabled */
 8     asm(" wrteei 0");
 9
10     /* write the new A value */
11     EMIOS.CH[21].CADR.R = new_A;
12
13     /* read the current counter value */
14     temp =  EMIOS.CH[21].CCNTR.R;
15
16     /* if the A match has been missed */
17     if( temp > new_A ){
18 /*>>>> start counting system clks here.
19      this is the code that will only run
20      on a possible Force A match.  >>>>>>*/
21          /* if too close to upcoming B match */
22          if( temp + LOOKAHEAD_BUF > EMIOS.CH[21].CBDR.R){
23                /* just return */
24                return;
25          }
26          else{
27                /* else force match A */
28                EMIOS.CH[21].CCR.R |= 0x2000;
29          }
30     }
31     /* reenable irqs if they were disabled in line 9*/
32     asm(" wrteei 1");
33 /*>>>> stop counting system clks here >>>>>> */
34 }
```

**Figure 7. OPWFM Pulse-Width Update — Example Code**

In the code example, LOOKAHEAD_BUF represents the time in UC time base clocks required for the force match A code to complete execution. Line 23 compares the current counter value plus the LOOKAHEAD_BUF value to the next B match value to see if the force A match will complete before the next B match occurs. If it will complete, the force A match is issued, but if it will not, the code returns and the new A match value will be in effect at the next period.

1. This code example is not intended to replace user application code. It is provided only as an illustration. Actual application code may require additional functionality not included in this code.

### 2.4.1.2 Calculating LOOKAHEAD_BUF

LOOKAHEAD_BUF represents the required time for a force match A to be completed. The LOOKAHEAD_BUF value is not in measured in system clocks, but rather counter ticks for the selected channel time base. The value depends on how the update function code is written (number of system clocks required to execute), which compiler is used and how the compiler optimization options are set, and how the prescalers in the eMIOS are configured. A change in any of the above dependencies requires a recalculation of the LOOKAHEAD_BUF value.

This formula can be used to calculate the LOOKAHEAD_BUF value, where:

Fsys = system clock frequency
Fchan = UC counter time base frequency
funcClks = number of Fsys clocks for the force match A code to execute (see code example).

***Eqn. 1***

$$\text{LOOKAHEAD\_BUF} = \frac{funcClks \cdot Fchan}{Fsys}$$

Referencing the example code in Figure 7, measurement showed that 26 system clocks were required to execute the `else` case with the force match A (line19 through line 34). Rounding up to 30 clks to provide a small safety margin and calculating the LOOKAHEAD_BUF time using Equation 1:

LOOKAHEAD_BUF = (30 * 20 MHz)/ 80 MHz = 7.5

Rounding up to an integer value results in a LOOKAHEAD_BUF of 8.

### 2.4.1.3 Maximum pulse-width

The size of the LOOKAHEAD_BUF will cause a limit on the maximum pulse duration:

PulseWidth < (B register value – LOOKAHEAD_BUF)

Examples of the effect of the workaround on maximum pulse-width at various PWM frequencies are shown in Table 1.

**Table 1. Maximum Pulse-Width — UC Counter Running at 20 MHz**

| | | Channel PWM Frequency | | |
|---|---|---|---|---|
| **Fsys** | **LOOKAHEAD_BUF** | **100Hz** | **1kHz** | **100kHz** |
| 80 MHz | 8 | (B=200,000)<br>Max PW = 99.996% | (B=20,000)<br>Max PW = 99.96% | (B=200)<br>Max PW = 96% |
| 132 MHz | 5 | (B=1,320,000)<br>Max PW = 99.9996% | (B=132,000)<br>Max PW = 99.996% | (B=1320)<br>Max PW = 99.6% |

### 2.4.2 Period Updates

Figure 8 illustrates how the counter-wrap condition can occur with period updates, and how it can be avoided.

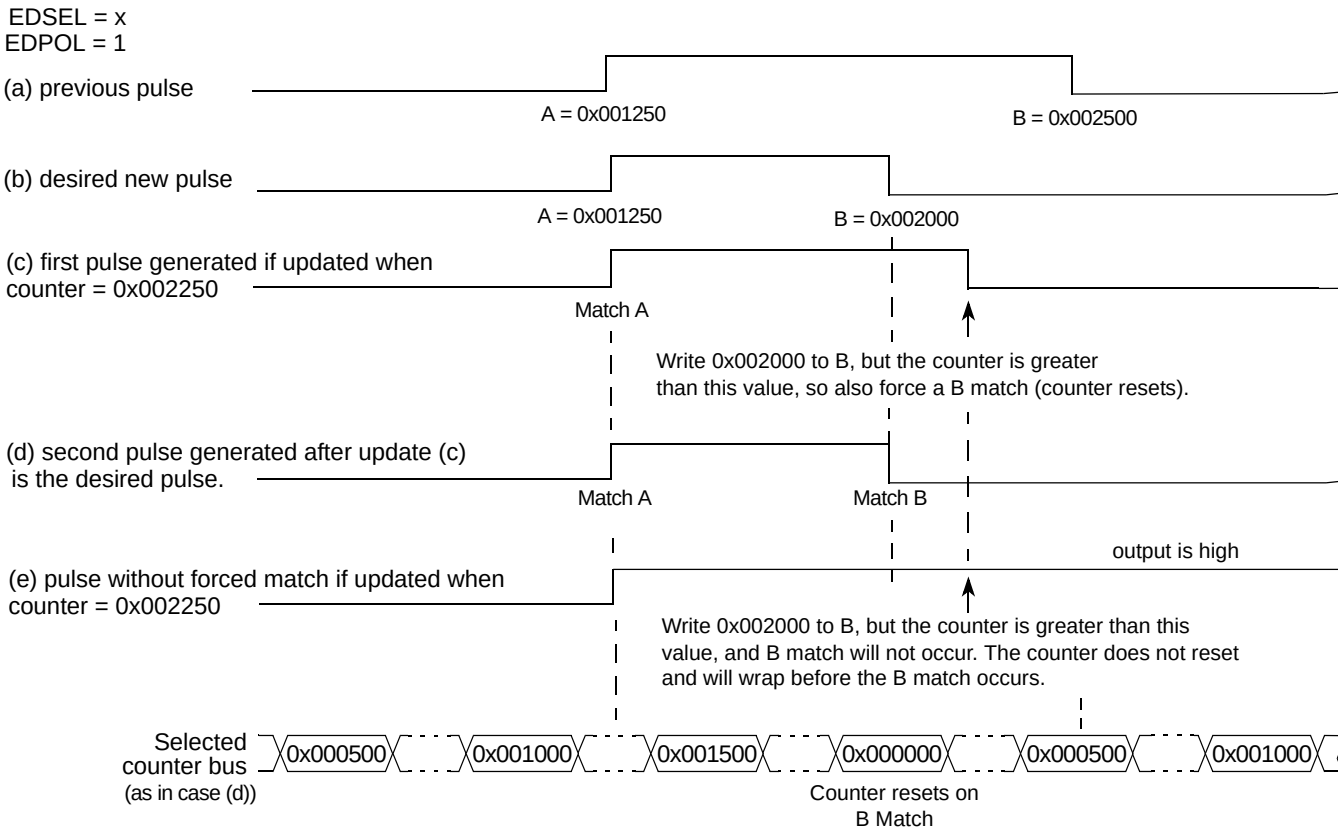



**Figure 8. OPWFM Mode — Asynchronous Period Update**

If a next period update for the B register can be allowed by the application, the test for the B update value against the current counter value is not required.

#### 2.4.2.1 Pseudocode

This section contains a pseudocode example to avoid the error conditions shown in Figure 5, Figure 6, and Figure 8. Updates to both registers may be combined in a single function.

```
update OPWFM pulse width:
        save MSR (machine state register)
        disable interrupts
        write the new A Register value
        if new A value < current counter value
                if counter value + buffer > B value
                        just return
                else force an A match
        restore MSR

update OPWFM period:
        save MSR (machine state register)
```

```
        disable interrupts
        write the new B Register value
        if new B value < current counter value
                force a B match
        restore MSR
```

## 2.4.3 Hazards Associated with 0% Duty Cycle

A UC running OPWFM mode may be configured to generate a 0% duty cycle by setting the A and B registers to the same value. In this mode, the output of the channel is controlled by the EDPOL bit in the UC channel control register. If EDPOL = 0, the output will be continuously low, and conversely if EDPOL = 1 the output will be continuously high. The subsequent sections describe timing hazards that should be avoided when using OPFWM configured for 0% duty cycle.

### 2.4.3.1 Setting the A and B Registers to Create 0% Duty Cycle

A potential timing hazard occurs when updating a UC to start 0% duty cycle and a new A and B value are written (so that A now equals B). Under certain timing sequences associated with the update of the registers, there may be a final pulse with an indeterminate duty cycle before the UC begins the 0% duty cycle.

### 2.4.3.2 Updating the Period During a 0% Duty Cycle

Even though a UC may be running OPWFM in a 0% duty cycle mode, the application may require that the period of the PWM be updated in preparation for the next time the UC is enabled at some non-zero duty cycle. If A and B registers are updated asynchronously, while A = B and the channel is in OPWFM mode, there is a possibility that a spurious pulse of indeterminate length may be generated.

#### 2.4.3.2.1 Pseudocode

There are two methods that avoid the hazards described in Section 2.4.3.1, “Setting the A and B Registers to Create 0% Duty Cycle,” and Section 2.4.3.2, “Updating the Period During a 0% Duty Cycle.” One may be used when the UC time base may be cleared and disabled and interrupts or flags are not needed, and one may be used when the UC must generate interrupts or flags and the internal counter must continue to increment.

- When the internal counter must continue to increment, and/or flags or interrupts must be generated:

  After it is configured for A = B, the channel is returned to OPWFM mode so that matches will continue to generate flags and/or interrupts. Note that in addition to the force match A, the physical pin is disconnected from the channel during the update. Also note that the internal counter is cleared and stopped when GPIO output mode is entered, and then restarts when OPWFM mode is entered.

```
set_duty_0( x )
        Put UC in GPIO output mode
        Set A = B = x
        Using SIU[1], disconnect UC from output pin
        Set UC for OPWFM mode (flags/interrupts are now generated if enabled)
```

1. SIU is System Integration Unit, See MPC5500 Reference Manual for your device.

```
        Force an A Match
        Using SIU, connect UC to output pin
```

**NOTE**

When using the SIU to disconnect and re-connect the UC to the output pin, the pin state while the UC is disconnected is determined by the PCR WPE and WPS bits and the state of the WKPCFG pin during system boot. The PCR must be configured so the output assumes the desired state while the UC is disconnected.

- Pseudocode (when flags or interrupts are not needed)

  In this pseudocode example, the output of the UC is set to 0% duty cycle by putting the UC in GPIO output mode and setting the EDPOL bit to the desired state. The internal counter is cleared and disabled and no flags and/or interrupts are generated while the UC is in GPIO output mode. After it is in GPIO output mode, updates may be made to A and B match registers at any time before the UC is returned to OPWFM mode.

```
set_duty_0_no_flags( x )
        Put UC in GPIO output mode
        Set EDPOL to desired output state
```

## 2.5 Single-Action Output Compare (SAOC)

This mode allows a UC to be configured to generate a defined output based on the match of a value in the A register. The selected UC time-base is compared to the A register.

If the software updates the A register to a value that is less than the internal counter value, the counter will continue counting until it wraps before finally hitting the new register A value. This will result in a stuck pin state. In most cases, it is preferred to immediately trigger a match event if a value written to match register A is less than the current counter value.

Figure 9 illustrates the possible output behavior under the conditions discussed.

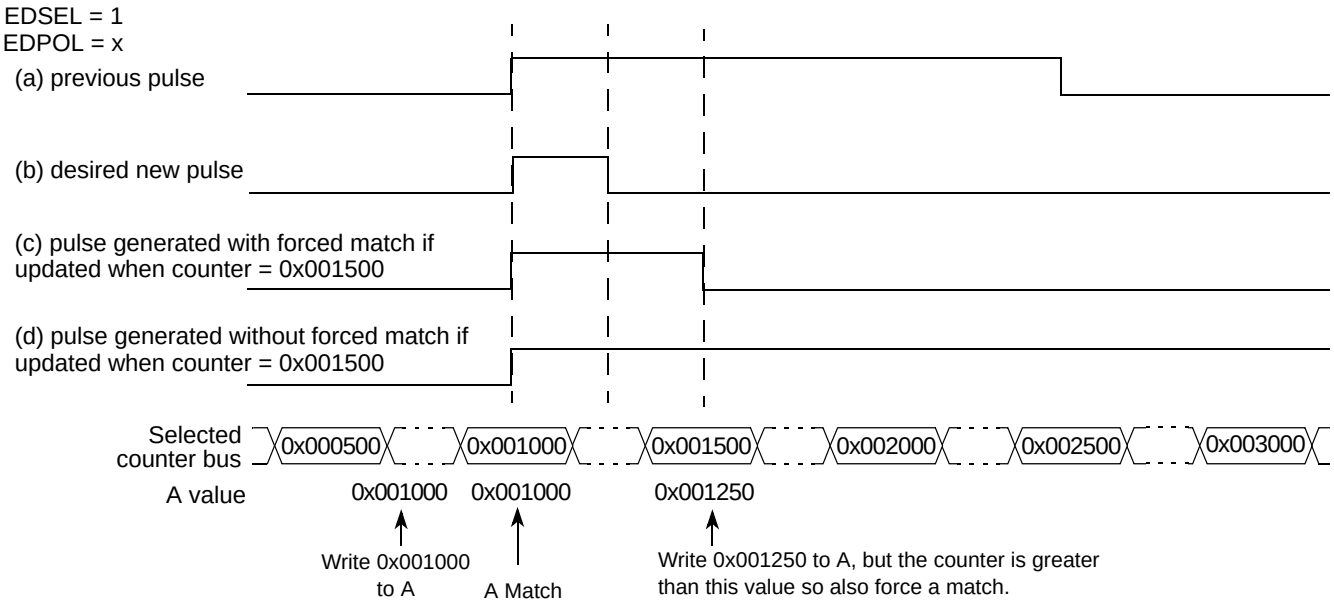



**Figure 9. SAOC Mode**

### 2.5.1 Pseudocode

This section contains a pseudocode example to perform the updated match value and counter value check illustrated in Figure 9.

```
update_saoc:
        save MSR (machine state register)
        disable interrupts
        write new match value to A register
        if (new value < the counter value)
                force a match now
        restore MSR
```

## 2.6 Double-Action Output Compare (DAOC)

The double-action output compare (DAOC) mode is used to generate a one-shot pulse, with pulse-width and timing determined by the A and B match registers. If either of the match registers are updated with a value less than the current UC counter value, a match will not occur until the counter has wrapped and again reaches the programmed match value.

Figure 10 illustrates the possible output behavior under the conditions discussed.

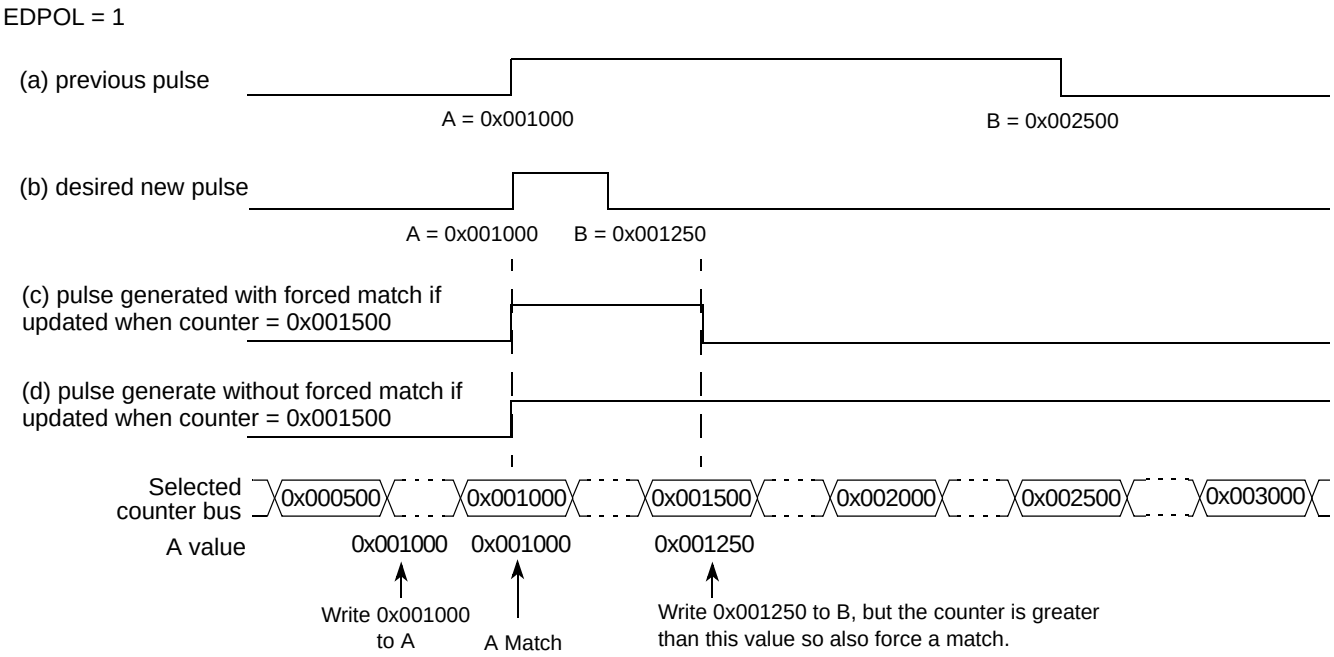



**Figure 10. DAOC Mode**

### 2.6.1 Pseudocode

The pseudocode example illustrates a method for updating the A registers. A similar routine should be used to update B or the two updates can be combined into one function.

```
update DAOC A:
        save MSR (machine state register)
        disable interrupts
        write the new A Register value
        if new value < current counter value
                force an A match
        restore MSR
```

## 2.7 Modulus Counter (MC)

The MC mode can be used to provide a time base for a counter bus or as a general-purpose timer. The internal counter counts up from the current value until it matches the value in register A. Register B is cleared and is not accessible to the MCU. Up mode or up/down mode may be selected.

If the software updates the A register to a value less than the current value of the internal counter, and the counter remains counting up, it will continue counting up until it wraps and finally hits the new register A value. This creates errant behavior in any UC that is using the MC as a time base.

To avoid this problem, the A match flag event should be configured to generate an interrupt that updates the A register before the next A match occurs. Asynchronous writes to match register A are not

recommended. If the A value does not need to be updated on a regular basis, the FEN bit may be used to trigger an update as needed. This avoids the overhead of servicing the interrupt once every MC period.

Figure 11 shows the suggested method of updating to the A register in MC mode. There is limit to how short the programmed period can be and allow enough time for the interrupt-driven update to occur. The software designer must ensure that this minimum period time is known and any period selected is greater than this minimum.

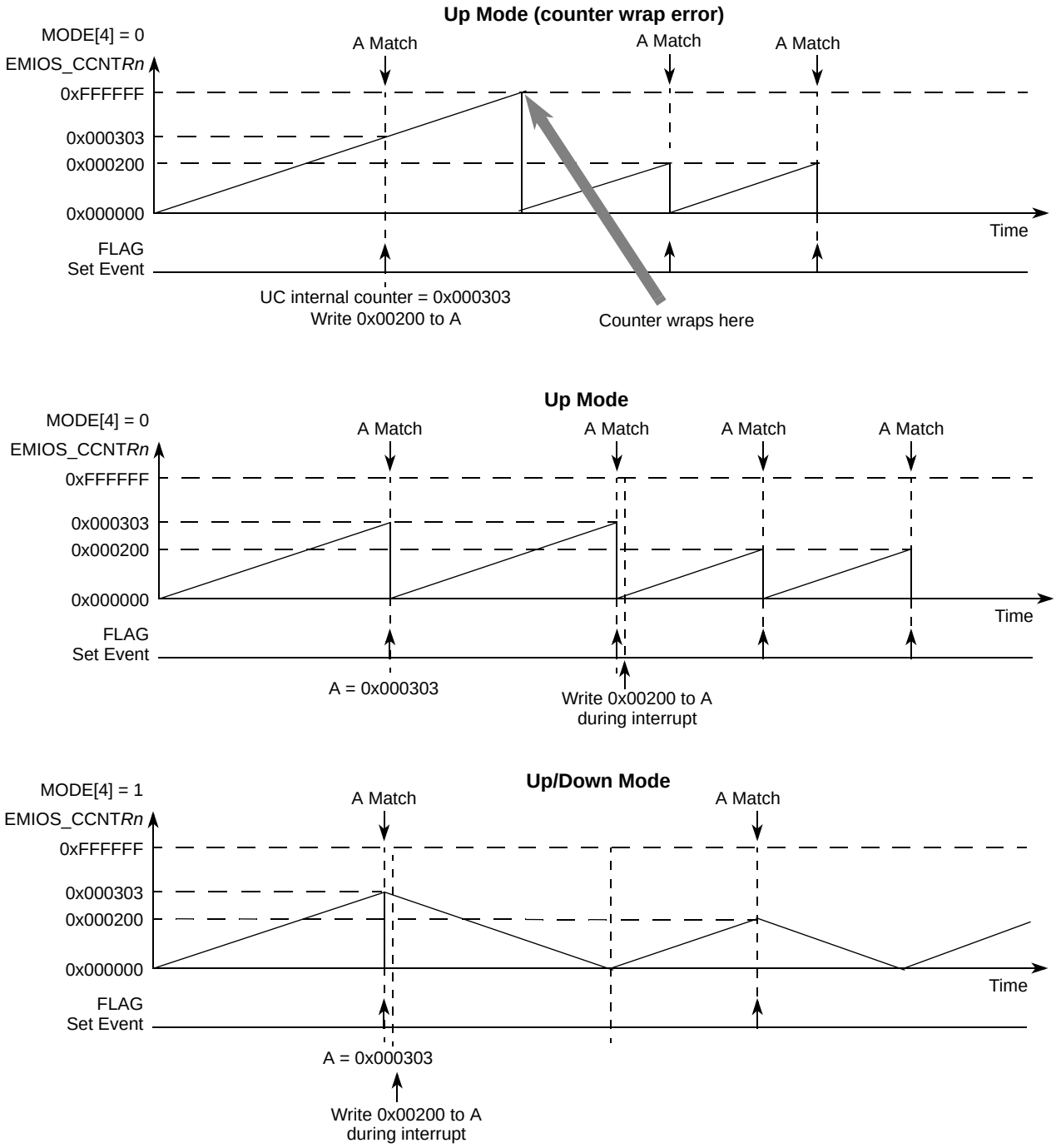



**Figure 11. MC Modes**

### 2.7.1 Pseudocode

The pseudocode illustrates a method for updating the A register as needed.

```
if (update_needed)
         load var with new A value
         write the FEN bit to enable irq
...
on match A irq:
         write var to A register
         clear FEN bit to disable further irq
         clear UC Flag
```

## 2.8 Output Pulse-Width Modulation Center Aligned (OPWMC)

The output pulse-width modulation center aligned (OPWMC) operating mode generates a center-aligned PWM with dead-time insertion at the leading or trailing edge. The selected counter bus must be running in up/down mode. Register A contains the ideal duty cycle for the PWM signal and is compared with the selected time base. Register B contains the dead-time value and is compared with the internal counter.

If an update to the A value is written to a value less than the current counter value, while the counter is counting up, a counter wrap occurs before the next match can take place. Additionally, if the A register value is updated to a value greater than the current time base count while the counter is counting down, the trailing edge will not be generated on the current down count, but rather on the next up count, so a trailing edge becomes a leading edge. The results of asynchronous writes to the A register can create erroneous counter behavior.

To avoid these potential problems, the A value (duty cycle) should only be updated during an interrupt at the trailing edge of the pulse, and must be greater than the current counter value to avoid an early match see Figure 12). Ensure that the update will complete before the next A match will occur.

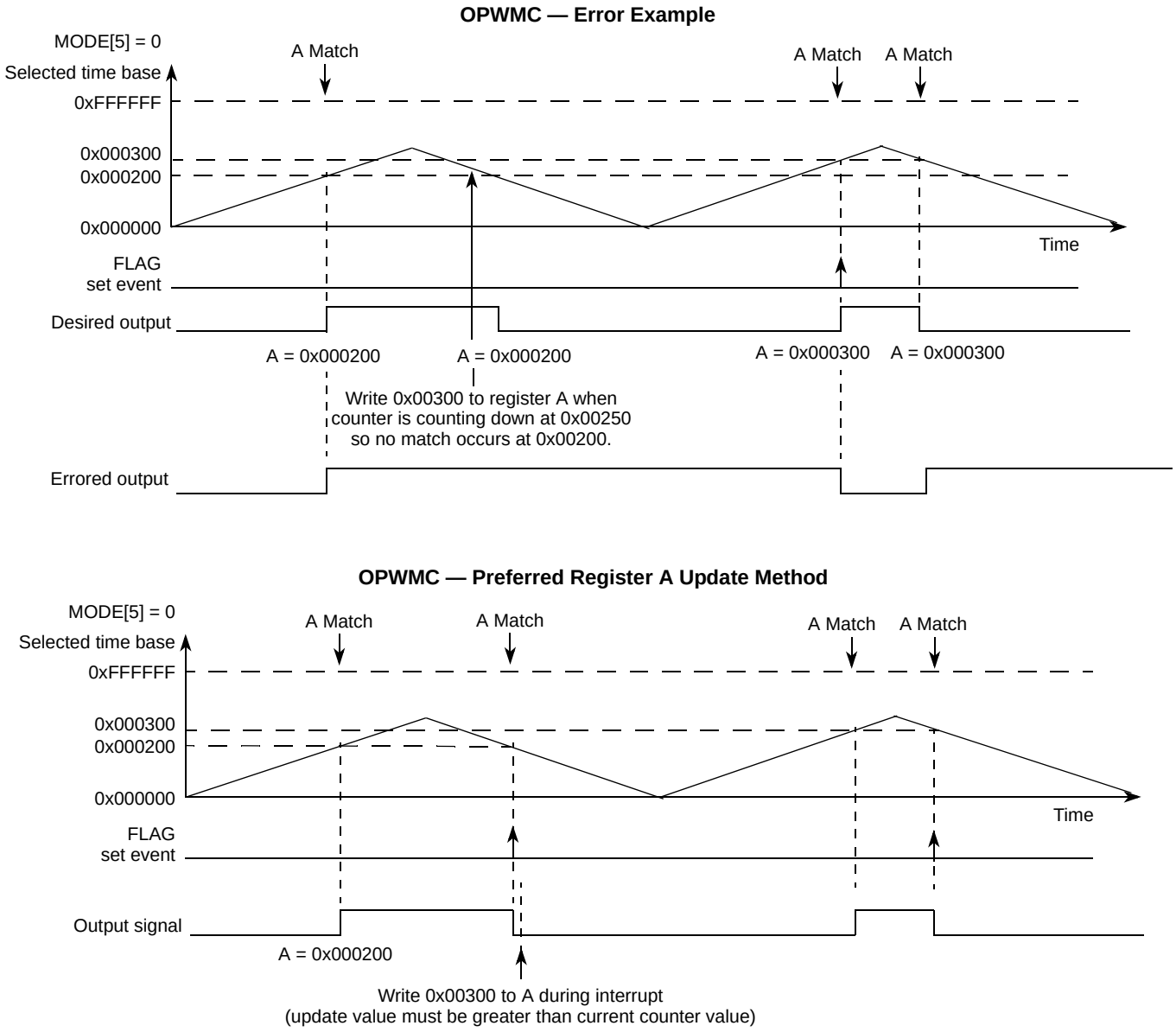



**Figure 12. OPWMC Mode**

## 2.8.1 Pseudocode

```
if (update_needed)
        load var with new A value
        write the FEN bit to enable irq
...
on match A irq:
        if var is >= to UC time base count
                write var to A register
                clear FEN bit to disable further irq
                clear UC Flag
        else leave irq enabled and return
```

## 2.9 Pulse Edge Count (PEC)

The PEC mode is used to count the number of pulses or edges detected on the input over a desired time window, as defined by the A and B register values programmed by software. The mode may be set to collect data continuously, or in a single shot operation.

If continuous operation is selected, the B match flag should be configured to generate an interrupt so that the service routine will read the count value from the UC counter register before the next A match occurs (see Figure 13). You must ensure that the interrupt service routine has time to run to completion before the next A match event. Asynchronous reads of the event counter are not recommended.

Updating the A and/or B registers while the time window is active (after a Match A, but before a Match B) will cause the time window change to take place immediately and may result in an erroneous time count in the internal counter. Asynchronous writes to the A and B registers are not recommended, and could be incorporated into the interrupt handler shown in the pseudocode below.

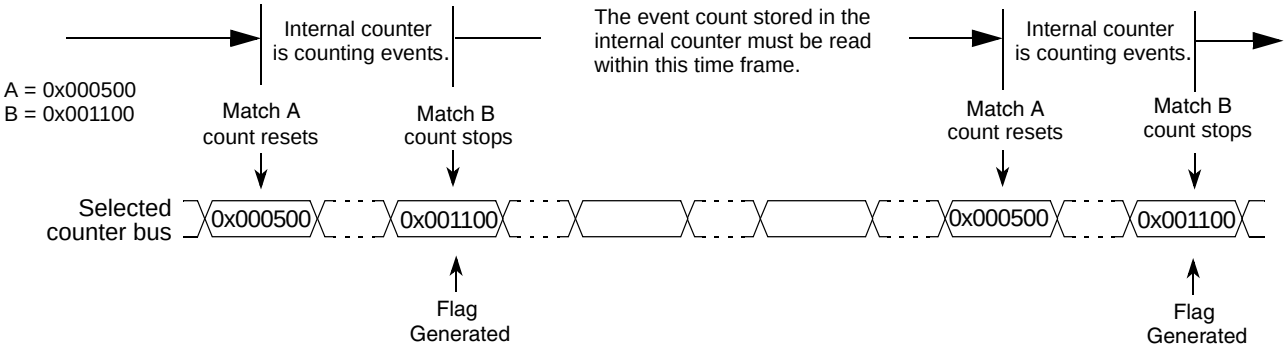



**Figure 13. PEC Continuous Mode**

### 2.9.1 Pseudocode

The pseudocode illustrates an interrupt-based method to ensure valid reading of the edge count value when in PEC continuous mode as illustrated in Figure 13.

```
if edge count is needed
        write the FEN bit to enable irq on B match
...
on match B irq:
        read UC internal counter
        optional: update A and B if needed
        clear FEN bit to disable further irq
        clear UC Flag
```

## 2.10 Pulse Edge Accumulate (PEA)

The PEA mode is used to measure a time required for a defined number of events to occur. Like PEC mode, PEA mode may be configured for continuous or single- shot mode. The measured time is available by reading the A and B registers, where the B register holds the beginning time at zero event counts, and the A register holds the ending time when the desired event count is reached.

There are two potential problems when using PEA in continuous mode. If a value is written to the A register while a measurement is in progress, that is, the internal counter is already counting events, and the new A register value is less than the current internal counter value, an A match will not occur and the counter wrap condition will exist. Additionally, reading the A register on exactly the same clock cycle that the eMIOS hardware is attempting to update it can result in non-coherent A and B register values.

To avoid the counter wrap problem above, asynchronous writes to the A register are not recommended. To avoid the coherency issue, the methods shown in Section 2.1.1, “Pseudocode,” and Section 2.1.1, “Pseudocode,” could be used, however with those methods it could be possible to miss a measurement period while polling the UC flag. Instead, the A match flag event must be configured to generate an interrupt so that the service routine will read the time base counter values from the A and B registers coherently before the next A match event occurs. This ensures true continuous mode operation. Asynchronous reads of the A and B registers are not recommended.

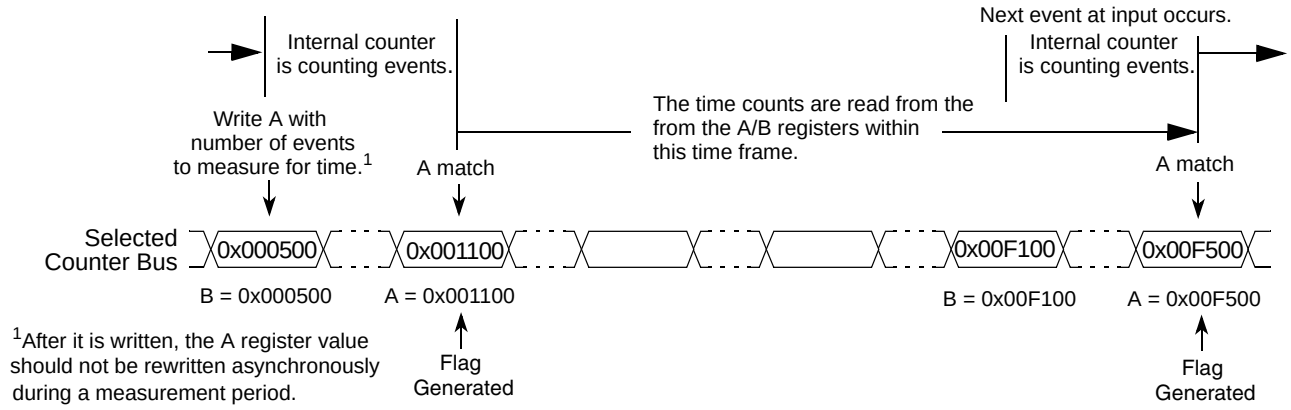



**Figure 14. PEA Continuous Mode**

## 2.10.1 Pseudocode

The pseudocode illustrates an interrupt-based method to ensure valid reading of the time base count values from the A and B registers when in PEA continuous mode. If the input events are occurring at a rapid rate, there may be a very short time to read A and B values coherently. The software designer must ensure the interrupt service routine will have time to run to completion before the next A match occurs.

```
if time count is needed
        write the FEN bit to enable irq on A match
...
on match A irq:
        read UC A and B registers
        (A register could be updated here)
        clear FEN bit to disable further irq
        clear UC Flag
```

# 2.11 Windowed Programmable Time Accumulation (WPTA)

The WPTA mode accumulates the sum of the total high time or low time of an input signal over a programmable interval (time window) set by the A and B register values. WPTA mode is always

continuous, so the B match flag should be configured to generate an interrupt so that a service routine will read the count value from the UC counter register before the next A match occurs (see Figure 13). The software designer must ensure the interrupt service routine will have time to run to completion before the next A match event.

Updating the A and/or B registers while the time window is active (after a Match A, but before a Match B) will cause the time window change to take place immediately and may result in an erroneous time count in the internal counter. Asynchronous writes to the A and B registers are not recommended, and could be incorporated into the interrupt handler shown in the pseudocode below.

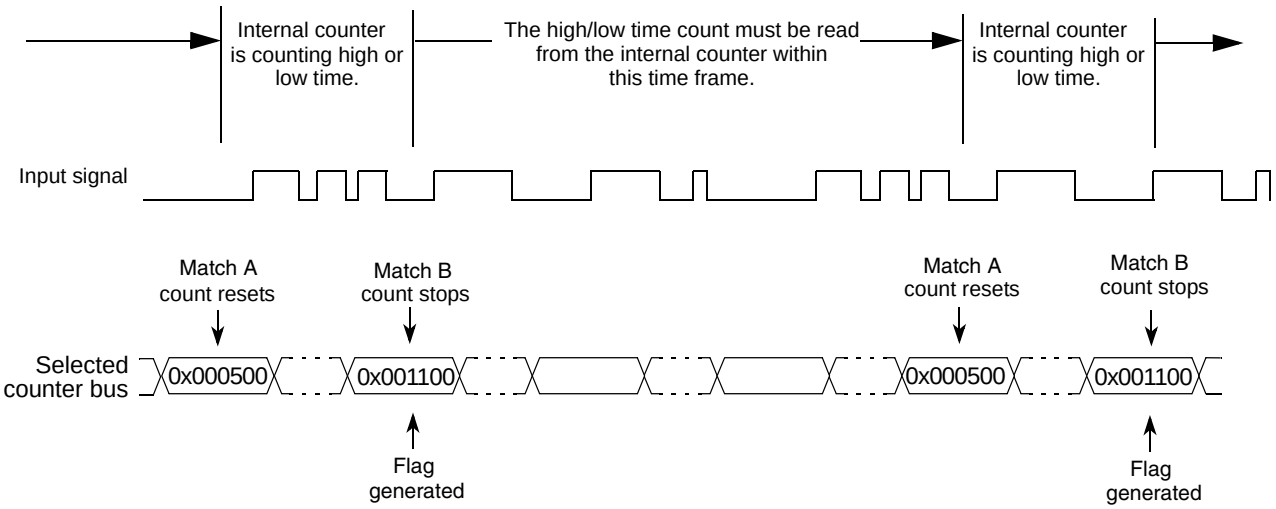



**Figure 15. WPTA Mode**

## 2.11.1 Pseudocode

This pseudocode illustrates an interrupt based method to ensure valid reading of the UC internal counter value when in WPTA mode. The interrupt service routine must have time to run to completion before the next A match event occurs.

```
if hi or low time count is needed
        write the FEN bit to enable irq on B match
...
on match B irq:
        read UC internal counter value
        optional: update A and B if needed
        clear FEN bit to disable further irq
        clear UC Flag
```

# 3 Document Revision History

**Table 2. Document Revision History**

| Rev. No. | Substantive Change(s) | Date |
|---|---|---|
| 1.0 | Initial release. | 07-21-05 |
| 1.1 | Updated OPWFM pulse-width update method to avoid missed pulses. | 07-15-06 |
| 1.2 | Added Section 2.4.3 - OPWFM, 0% duty cycle cases | 10-06-06 |

## *How to Reach Us:*

**Home Page:**
www.freescale.com

**E-mail:**
support@freescale.com

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

**Europe, Middle East, and Africa:**
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

**Japan:**
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only:***
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: EB651
Rev. 1.2
10/2006



RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

freescale™
semiconductor