Freescale Semiconductor
Application Note

Document Number: AN4424
Rev. 0, 12/2011

Synchronize Analog Modules and Timers with PDB Modules in MC9S08MP16

by: Gang Chen

Contents

1 Introduction

The synchronization of the analog modules and timers in the MC9S08MP16 series is one of the best features of this microcontroller. The hardware synchronization can help to reduce the CPU processing time and can be used in various applications, such as motor control and medical solutions. The programmable delay block (PDB) is a versatile module integrated in MC9S08MP16 series MCU to implement the synchronization easily and automatically. This application note explains how to synchronize the analog modules in all the different combinations, such as real-time counter (RTC) with PDB, flextimer (FTM) with analog-to-digital converter (ADC), and so on.

The MC9S08MP16 series has two PDB modules, referred to as PDB1 and PDB2. Figure 1 shows the block diagrams of and the interconnections between the timers, the analog modules, and the PDB modules.

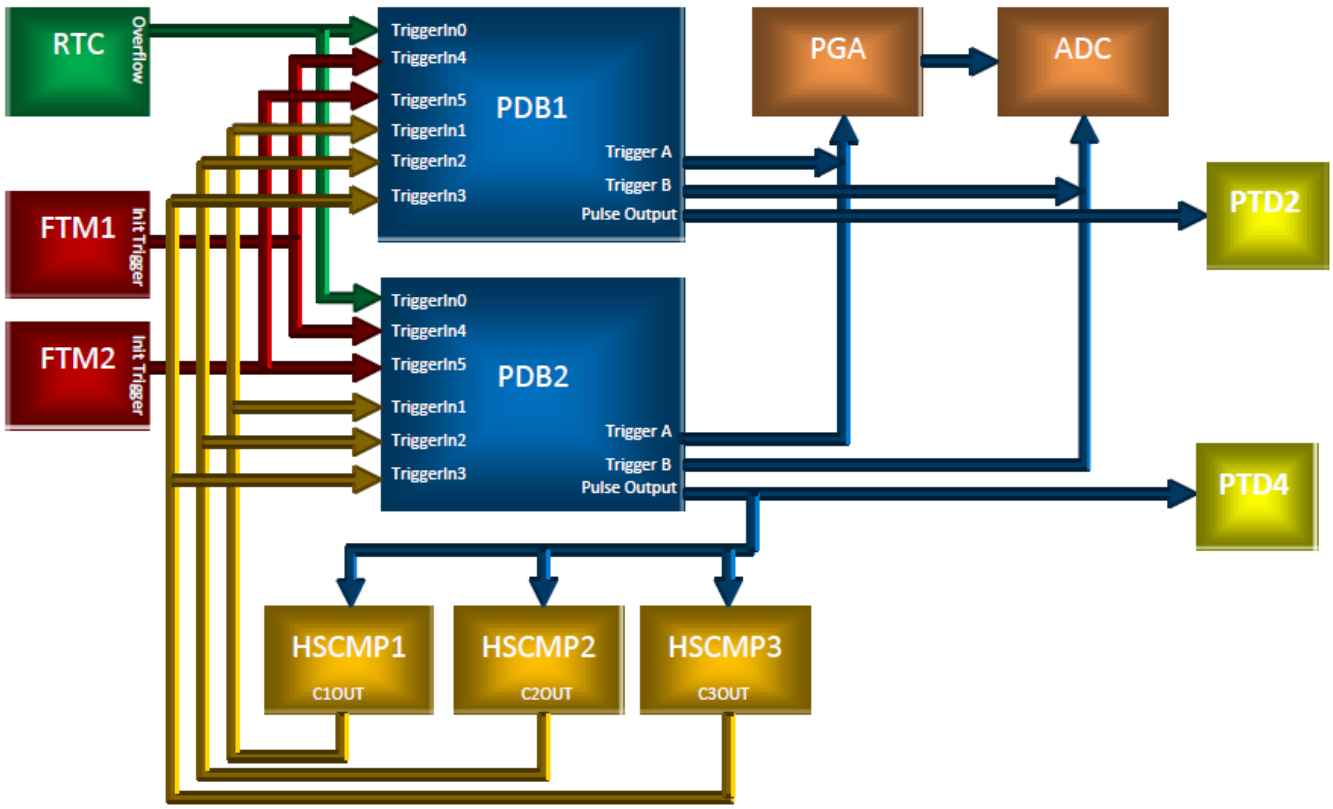

Figure 1. Connections between timers, analog modules, and the PDB modules in MP16

The primary function of the PDB is to provide:

- A controllable delay from the FTM's SYNC output to the sample trigger input of the programmable gain amplifiers and ADC.
- A controllable window that is synchronized with PWM pulses for analog comparators to compare the analog signals in a defined window.

An alternate function of the PDB is to generate PWM pulses that are synchronized to the FTM, comparator's output, and RTC. The trigger signal, which can be selected for the FTM's initial trigger, comparator's output, RTC overflow, or software, starts PBD or resets the PBD counter, if the PDB is in continuous mode. A single one-shot pulse or pulse string can be generated in result to a response trigger signal.

Each of the PDB has three available outputs as shown in Figure 1. Trigger A and Trigger B are versatile outputs, each of them can be configured to output the single signal of Trigger A or Trigger B, double signals of both Trigger A and Trigger B, or the pulse output. Only the pulse output signal could be output to the external pin of the module.

The pulse output of the PDB modules is generated by Trigger A and Trigger B together. Trigger A and Trigger B are used to precisely schedule the rising and falling edges for the output waveform. Figure 2 shows the relationship among Trigger A, Trigger B, and the pulse output. Delay time A and B are controlled by registers PDB1DLYA and PDB1DLYB. The clock source of the PDB module is the system bus clock. The width of the pulse is determined by the difference between Delay A and Delay B.

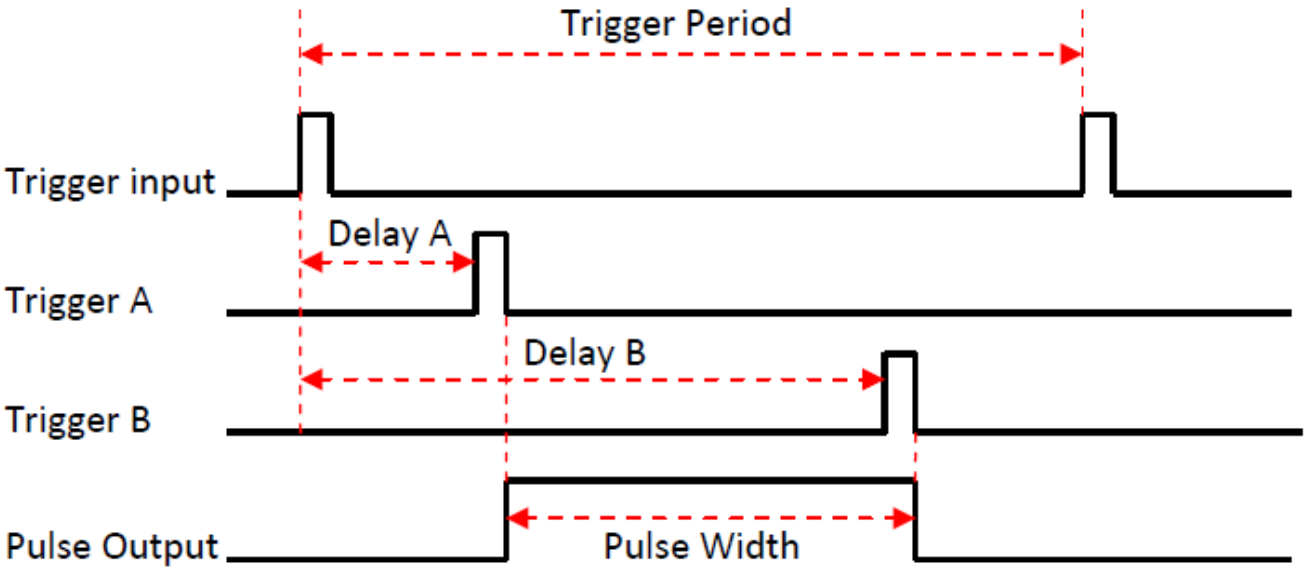

Figure 2. Pulse output of the PDB module

Hereafter, some application examples are discussed to introduce how to use the PDB modules in the MS9S08MP16 series MCU.`

2 Example 1 - Trigger PDB with RTC and output pulse/PWM signal to the external pin

2.1 Use One-Shot mode to output pulse signal

The simplest way to use the PDB is to trigger it with the RTC, then output pulse or PWM signals to its external pin. We can watch clearly how it works with the visible pulse waveforms on the external pin. The block diagram of this example is shown in Figure 3. The PDB1 selects the RTC overflow signal as the input trigger signal, and outputs the pulse signal to the external pin of PTD2.

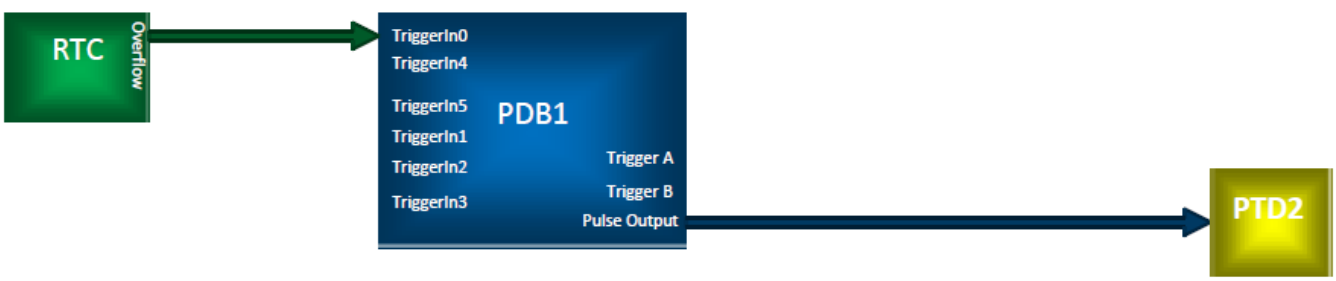

Figure 3. RTC triggers PDB1 to output pulse to PDT2

Figure 2 has explained how to generate pulse/PWM signals within the PDB module. As the RTC is selected as the trigger input, the PDB will be triggered periodically by RTC overflows. Therefore, if the PDB module is set to work on One-Shot mode, the output period will be same as the overflow period of the RTC.

Example 1 - Trigger PDB with RTC and output pulse/PWM signal to the external pin

Below are example codes to initialize RTC and PDB module to make them work.

```
void rtc_init(void) {
       /* Select the 1-KHz low power oscillator (LPO) as the clock source, prescaler is 10
and modulo is 2, so that it overflows every 20 ms */
        RTCMOD = 0x01U;
               /* RTCMOD(b7:b0): Modulo = 1 + 1 = 2 */
        RTCSC = 0x0BU;
               /* RTCSC: RTIF(b7)= 0 */
               /* RTCLKS(b6:b5)= 0, select the 1-KHz clock source */
               /* RTIE(b4)= 0, real-time interrupt is disalbed */
               /* RTCPS(b3:b0)= 11 (0xB), RTC prescaler is 10 */
}
```

Function rtc_init(void) initializes registers RTCMOD and RTCSC to make the RTC run and overflow every 20 ms. The PDB1 uses this overflow signal to trigger its delay counters.

```
void pdb1_init(void) {
   PDB1DLYA = 0x01U;
       /* PDB1DLYA(b15:b0) = 0x01, PDB Delay A value */
   PDB1DLYB = 0xFFFFU;
       /* PDB1DLYB(b15:b0) = 0xFFFF, PDB Delay B value */
   PDB1CTRL1_LDOK = 1U;
       /* LDOK(b0) = 1, set to load delay and modulo registers */
   PDB1SCR = 0xC0U;
       /* PADEN(b7) = 1, pulse output is driven on external pin */
       /* PDBEN(b6) = 1, enable PDB module */
       /* COF(b5) = 0, counter overflow flag, write 1 to clear */
       /* COIE(b4) = 0, counter overflow interrupt disabled */
       /* DBF(b3) = 0, Delay B successful compare flag, write 1 to clear */
       /* DBIE(b2) = 0, Delay B successful compare interrupt disabled */
       /* DAF(b1) = 0, Delay A successful compare flag, write 1 to clear */
       /* DAIE(b0) = 0, Delay A successful compare interrupt disabled */
}
```

Function pdb1_init(void) needs to write three registers only to get the PDB1 run and output pulse/PWM to the external pin. All the other registers can keep their default values. These three registers are: PDB1DLYA, PDB1DLYB, and PDB1SCR.

Register PDB1CTRL1 keeps its default value of 0x00, so that both Trigger A and Trigger B are bypassed and their internal outputs are disabled, keep low, because in this example we only send the pulse to an external pin, no internal signals are used. However, the counter of PDB1 and the delay time comparators will work when the module is enabled, that is, bit PDBEN of PDB1SCR is set.

When the PDB1 pulse output is enabled, bit PADEN of PDB1SCR is set, the PTD2 pin is controlled by the PDB1 module. On default mode, if no pulse is output on the PTD2 pin, it keeps low level. Trigger A controls the rising edge of the pulse, and Trigger B controls falling edge. Therefore, if Delay A value is less than Delay B, PTD2 will output a positive pulse; and if Delay A value is larger than Delay B, a negative pulse will be the output.

In this example, the Delay A value is set as 1, which is the minimum value, and Delay B is set as the maximum value of 0xFFFF. The PDB counter counts from 1 to 0xFFFF. If a delay value is set as 0, no trigger event will happen. As the system bus clock is about 5 MHz, or 0.2 μs, the Delay A and Delay B time is:

Delay A time = 0 Delay B time = 0.2 μs × (0xFFFF – 1) = 13.1 ms

Therefore the pulse width is:

Pulse Width = Delay B time – Delay A time = 13.1 ms

Figure 4 is the output pulse waveform on the PTD2 pin.

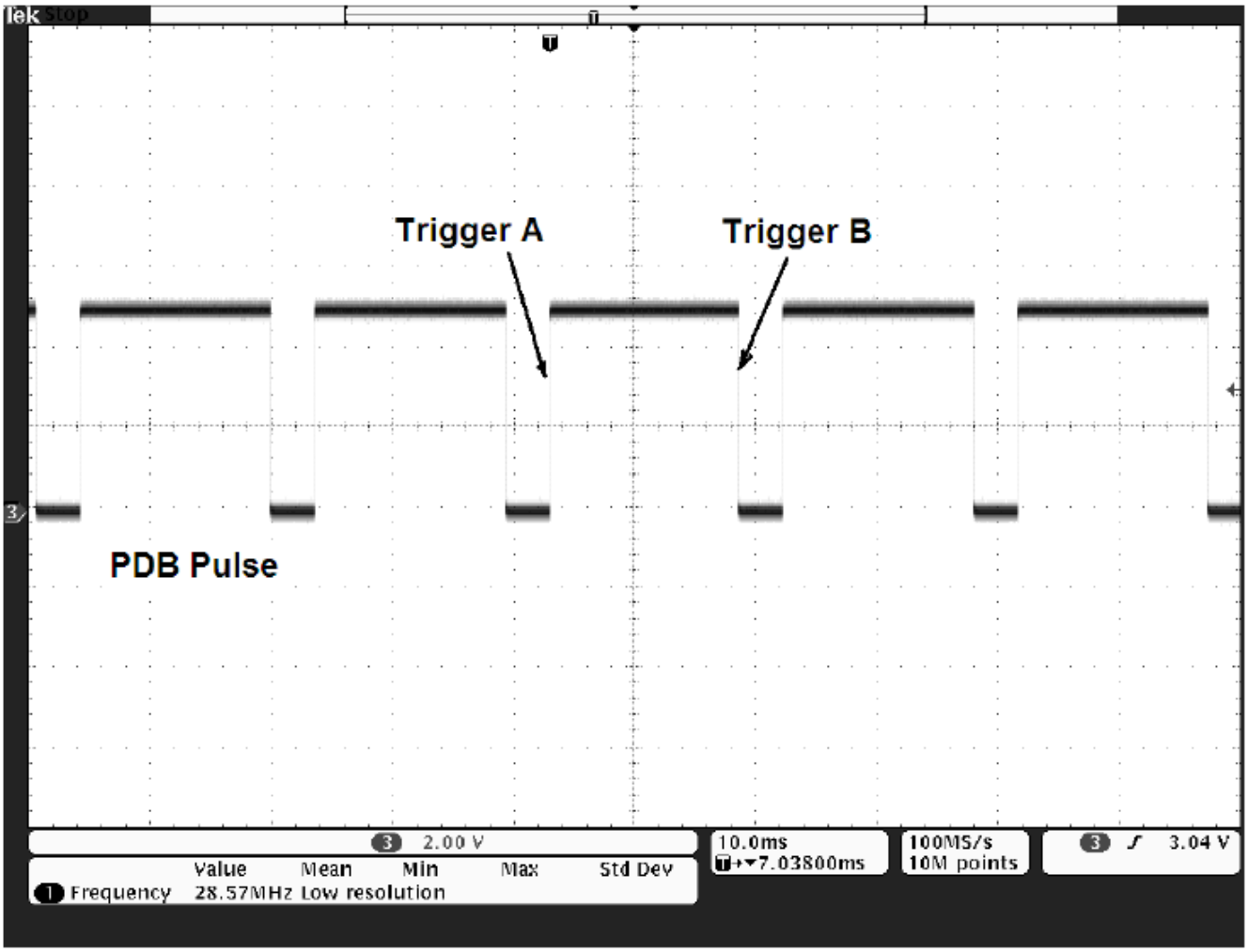

Figure 4. Output pulse on PTD2 (PDB1DLYA = 1, PDB1DLYB = 0xFFFF)

It is very easy to change the pulse's polarity by switching the delay time between Delay A and Delay B register. By setting PDB1DLYA as 0xFFFF, and setting PDB1DLYB as 1, the pulse waveform will change as shown in Figure 5.

Example 1 - Trigger PDB with RTC and output pulse/PWM signal to the external pin

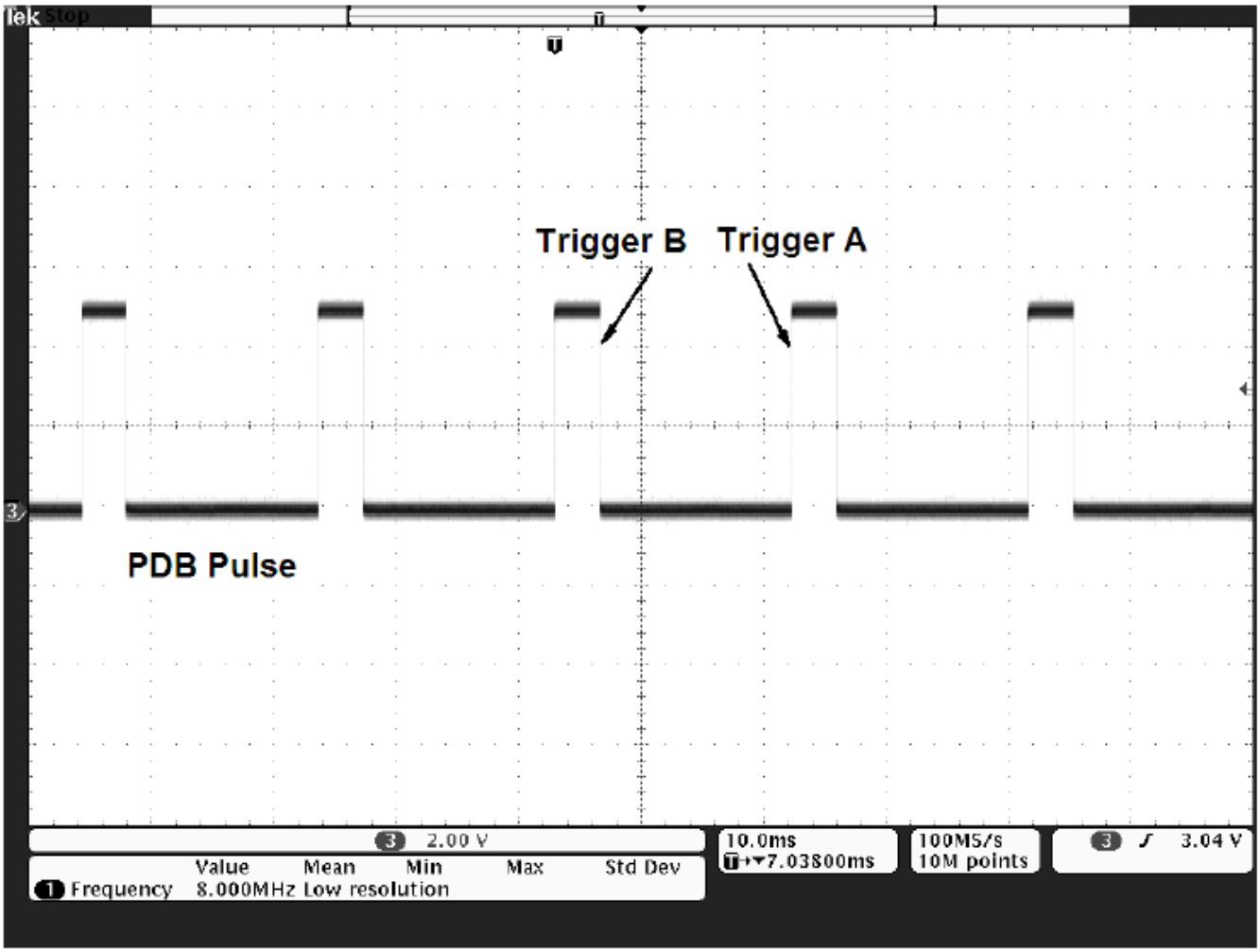

Figure 5. Output pulse on PTD2 (PDB1DLYA = 0xFFFF, PDB1DLYB = 1)

2.2 Use Continuous mode to output pulse string

Another application of the pulse/PWM output function is to generate a group of pulses every time when the PDB is triggered. This can be implemented by using Continuous mode of the PDB module. When bit CONT of PDBxCTRL2 is set, the Continuous mode is enabled, and that means every time when the PDB counter reaches the modulo value or overflows, it will restart to count again from 1.

Below is the example code to initialize the PDB1 to generate a group of pulses and use its counter overflow interrupts to control the number of pulse.

```
void pdb1_init(void) {
     PDB1CTRL1 = 0x00U;
        /* LDMOD(b7) = 0, Load mode selection bit */
        /* BOS(b6:b5) = 00, counter B delay is bypassed */
        /* ENB(b4) = 0, Trigger B outputs are disabled */
        /* AOS(b3:b2) = 00, counter A delay is bypassed */
        /* ENA(b1) = 0, Trigger A outputs are disabled */
        /* LDOK(b0) = 0, set to load delay and modulo registers */
     PDB1CTRL2 = 0x02U;
        /* PRESCALER(b7:b5) = 000, peripheral clock prescaler is 1 */
        /* TRIGSEL(b4:b2) = 00, select RTC overflow signal as PDB trigger */
        /* CONT(b1) = 1, module is in Continuous mode */
```

```
      /* SWTRIG(b0) = 0, software trigger bit */
    PDB1MOD = 0x2000U;
      /* PDB1MOD(b15:b0) = 0x2000, PDB modulo value */
    PDB1DLYA = 0x01U;
      /* PDB1DLYA(b15:b0) = 0x01, PDB Delay A value */
    PDB1DLYB = 0x1000U;
      /* PDB1DLYB(b15:b0) = 0x1000, PDB Delay B value */
    PDB1CTRL1_LDOK = 1U;
      /* LDOK(b0) = 1, set to load delay and modulo registers */
    PDB1SCR = 0xF0U;
      /* PADEN(b7) = 1, pulse output is driven on external pin */
      /* PDBEN(b6) = 1, enable PDB module */
      /* COF(b5) = 1, counter overflow flag, write 1 to clear */
      /* COIE(b4) = 1, counter overflow interrupt enabled */
      /* DBF(b3) = 0, Delay B successful compare flag, write 1 to clear */
      /* DBIE(b2) = 0, Delay B successful compare interrupt disabled */
      /* DAF(b1) = 0, Delay A successful compare flag, write 1 to clear */
      /* DAIE(b0) = 0, Delay A successful compare interrupt disabled */
}
```

To output a pulse string every time the PDB module is triggered, an interrupt should be generated for every trigger period to control the pulse number. The PDB has three interrupt sources: counter overflow, Delay A match and Delay B match. Each interrupt source has an enable bit that can block the interrupt request from being recognized by the interrupt controller. In this example, the counter overflow interrupt is used to control the pulse number and output 6 pulses for every trigger event. Below is the interrupt service routine code.

```
char pdb1_cont_times = 0;

void interrupt pdb1_isr(){
    if(PDB1SCR_COF == 1){
      PDB1SCR_COF = 1;          //Clear counter overflow flag
      if(PDB1CTRL2_CONT)
      {
         if(++pdb1_cont_times >= 5)    //Output 6 pulses every time
         {
            PDB1CTRL2_CONT = 0;//Disable Continuous mode after 6 pulses
            pdb1_cont_times =0;
         }
    }else
    {
          PDB1CTRL2_CONT = 1; //Enable Continuous mode to output pulses
          pdb1_cont_times = 0;
    }
   }
     if(PDB1SCR_DAF == 1)
        PDB1SCR_DAF = 1;          //Clear Delay A match flag
     if(PDB1SCR_DBF == 1)
        PDB1SCR_DBF = 1;          //Clear Delay B match flag
}
```

In this example code, the modulo value of the PDB counter is set as 0x2000, the Delay B value is 0x1000, and the Delay A value is 1. Therefore, the pulse period is 0x2000, bus clock time, and the pulse width is 0x1000, bus clock time, so that the duty cycle of the pulse is 50%. Figure 6 is the pulse string waveform output on PTD2.

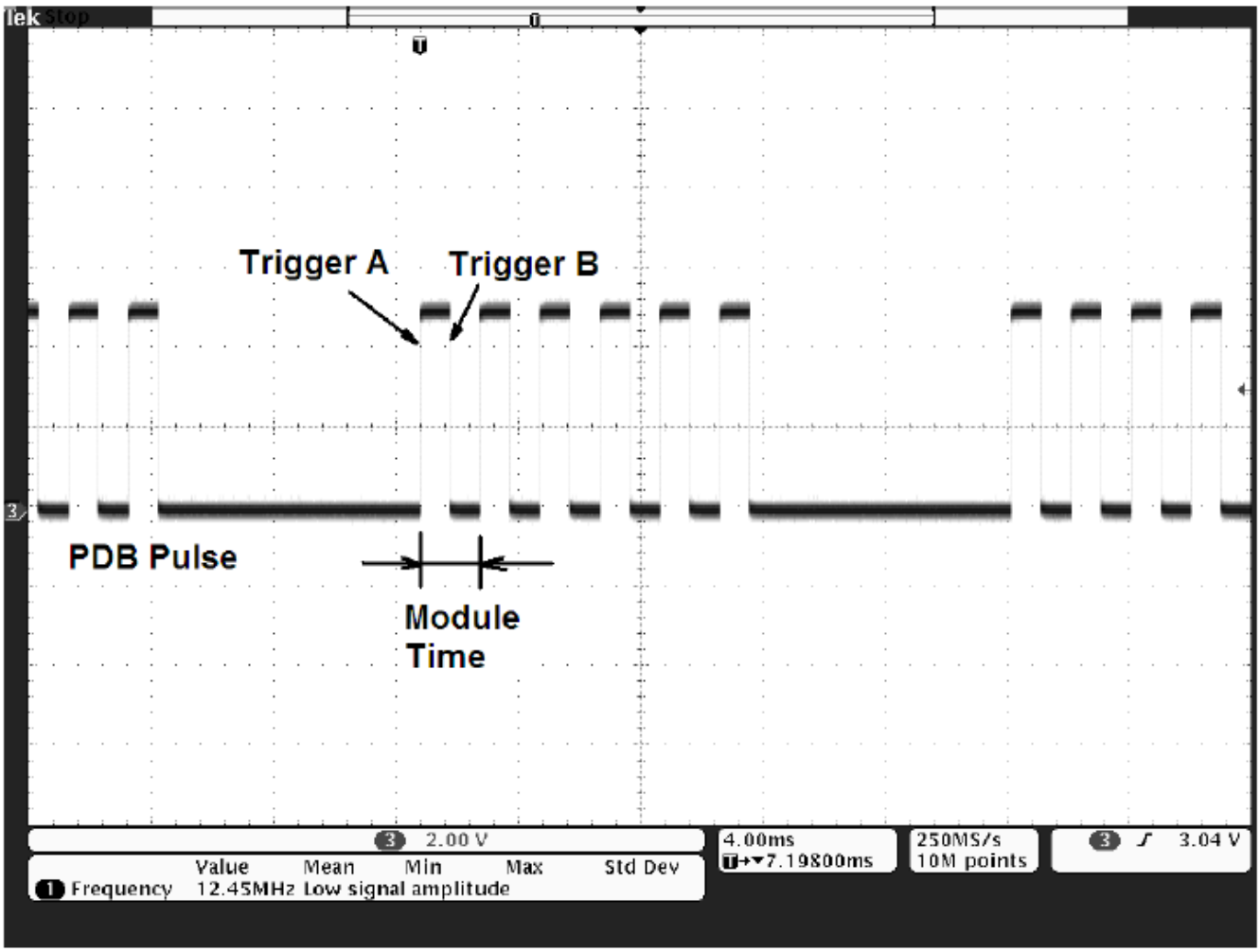

Figure 6. Output pulses with a Continuous mode of PDB

3 Example 2 - synchronize FTM and ADC with PDB

In some applications, the ADC must work synchronously with another module. For example, when controlling a motor with PWM signals, the ADC module must sample at a specific moment of the PWM signal so that the right current or voltage signal is sampled. With the PDB module, the synchronization between various modules could be implemented automatically by hardware.

Figure 7 shows the internal connection among PDB1, PGA, FTM, and ADC modules. The ADC module can be triggered directly by RTC if a periodic sampling and AD conversion is required. However, if the ADC needs to work synchronously with FTM1 or FTM2, it could be implemented with PDB1 or PDB2. FTM1 or FTM2 triggers PDB1/PDB2 with the initialization signal. After a time delay, the PDB modules will trigger PGA with Trigger A or Trigger ADC directly with Trigger B. PGA will trigger ADC after it finishes sampling and conversion. The PDB modules can also trigger ADC directly with Trigger A if the PGA is disabled.

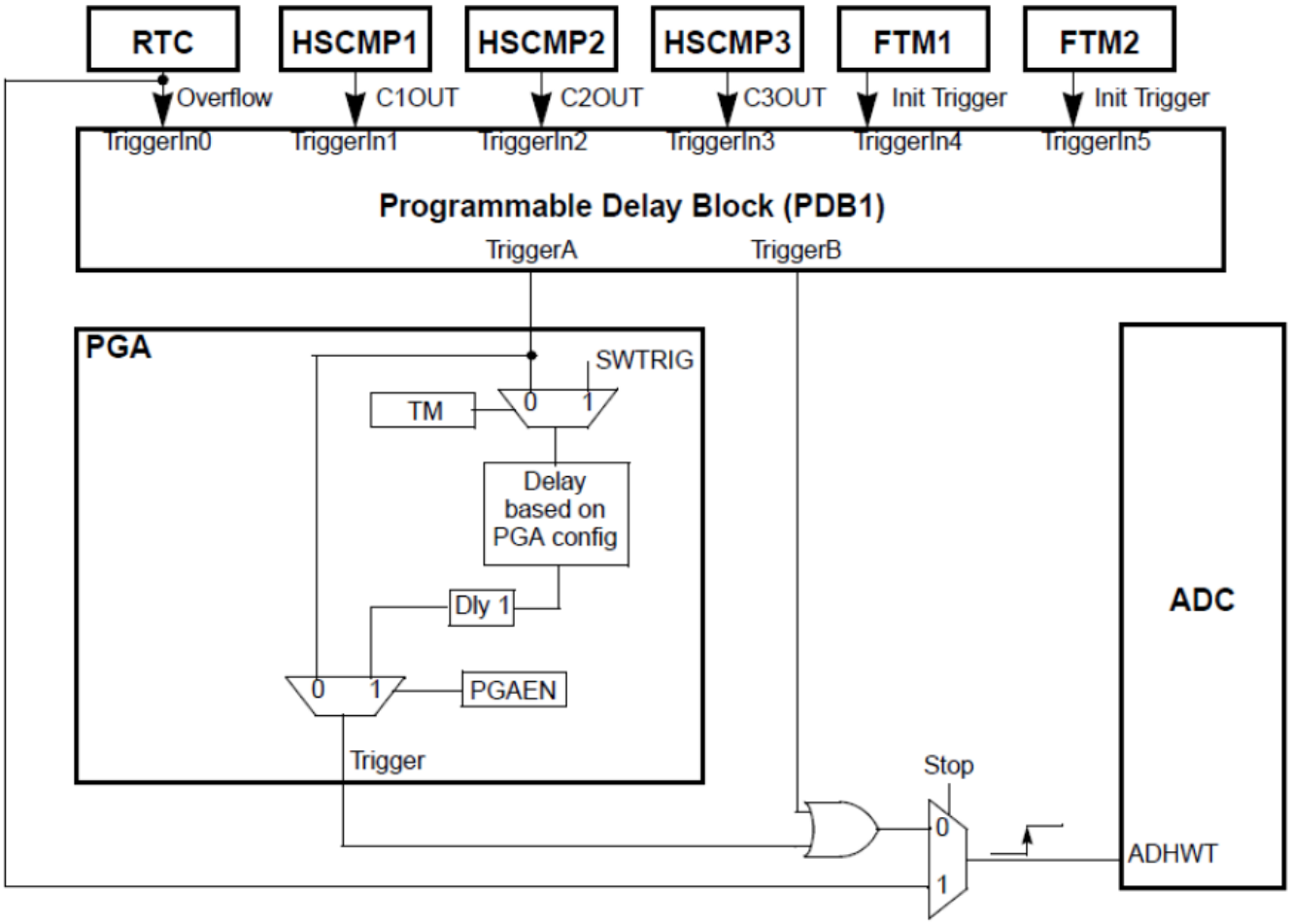

Figure 7. ADC hardware trigger block diagram

3.1 Trigger ADC with Trigger B only

Here is a simple example showing how the ADC module works synchronously with the FTM1. Figure 8 shows the block diagram of modules that are used in this application example. In this example, only Trigger B of PDB1 is enabled to trigger the ADC directly, both Trigger A of PDB1 and the PGA are disabled.

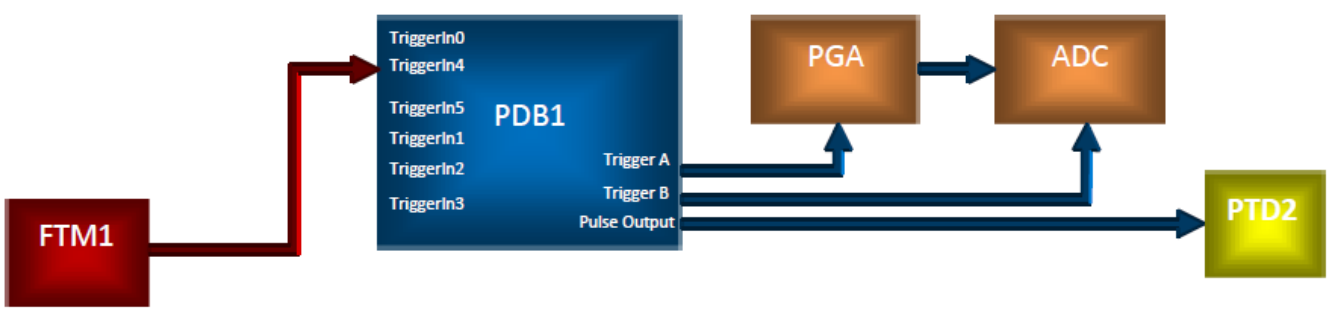

Figure 8. Modules used to synchronize FTM1 and ADC with PDB

The FTM1 module is initialized to output a center-aligned PWM signal, as shown in Figure 9. Its modulo value is set as 300 and the channel value is set as 200. Therefore, the PWM period is 600, bus clock time, and the duty cycle is 1/3. Channel 0 of FTM1 is selected so that the PWM signal outputs on pin PTA2.

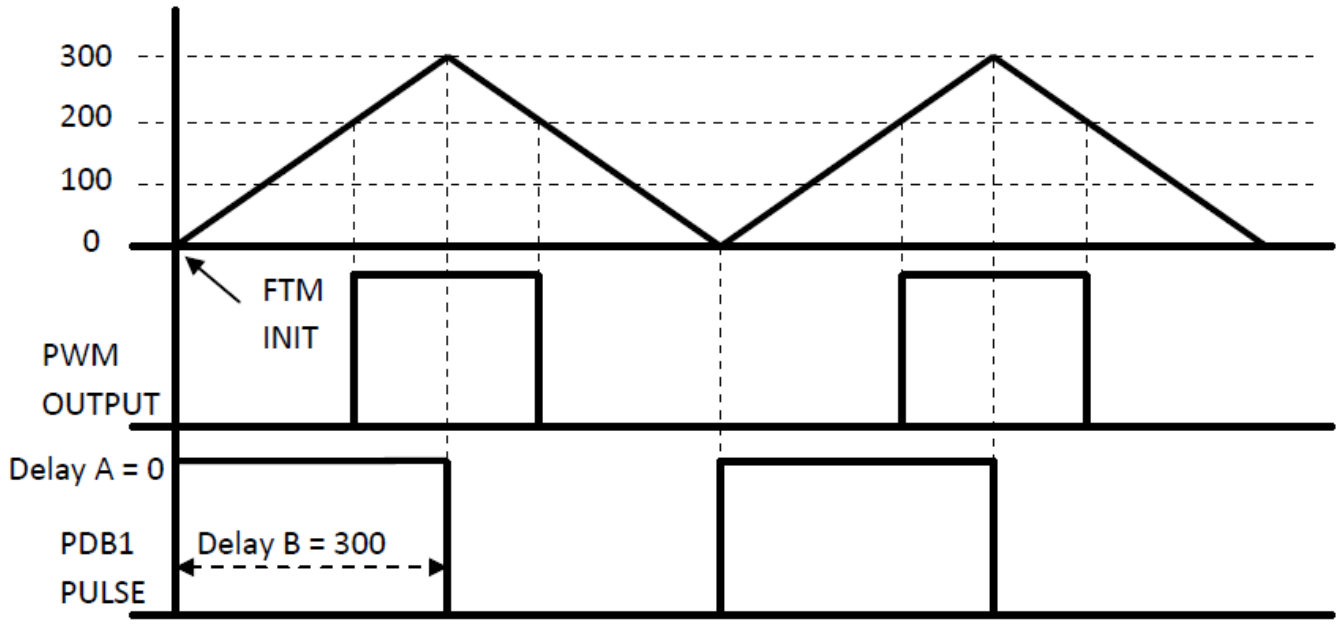

Figure 9. Synchronize FTM1 and ADC with PDB

Below is the FTM1 initialization code:

```
void ftm1_init(void){
    FTM1SC = 0x28U;
        /* TOF(b7): Timer overflow flag */
        /* TOIE(b6) = 0, Timer overflow interrupt disabled */
        /* CPWMS(b5) = 1, Center-aligned PWM selected */
        /* CLKS(b4:b3) = 01, System clock is selected */
        /* PS(b2:b0) = 0, Prescaler factor */
    FTM1C0SC = 0x04U;
        /* CH0F(b7) = 0, Channel 0 flag; */
        /* CH0IE(b6) = 0, Channel 0 interrupt is disabled */
        /* MS0B(b5) = 0, Channel 0 mode select B */
        /* MS0A(b4) = 0, Channel 0 mode select A */
        /* ELS0B(b3) = 1, Channel 0 edge/level select B */
        /* ELS0A(b2) = 0, Channel 0 edge/level select A */
     FTM1MOD = 300;
        /* FTM1MOD(b15:b0), FTM1 counter modulo value */
     FTM1C0V = 200;
        /* FTM1C0V(b15:b0), FTM1 channel 0 value */
     FTM1CNTIN = 0;
        /* FTM1CNTIN(b15:b0), FTM1 counter initial value */
     FTM1MODE = 0x00U;
        /* FAULTIIE(b7) = 0, Fault interrupt disabled */
        /* FALUTM(b6:b5) = 00, Fault Control mode */
        /* CAPTEST(b4) = 0, Capture Test mode is disabled */
        /* PWMSYNC(b3) = 0, PWM Synchronization mode */
        /* WPDIS(b2) = 0, Write protection enabled */
        /* INIT(b1) = 0, Don't initialize the output channels */
        /* FTMEN(b0) = 1, All the FTM registers are available */
     FTM1EXTTRIG = 0x40U;
        /* TRIGF(b7) = 0, Channel trigger flag */
        /* INITTRIGEN(b6) = 1, Initialization trigger enabled */
        /* CHnTRIG(b5:b0) = 0, Channel n trigger disabled */
}
```

To trigger the PDB with FTM1, INITTRIGEN bit of FTMnEXTTRIG must be set to enable the trigger signal output to TriggerIn4 of PDB1 and PDB2.

For the PDB, it selects the TriggerIn4 as the trigger input, and enables Trigger B to trigger the ADC module. In addition, it must set correct delay time. In this example, Delay A is set as 1 and Delay B is set as 300, bus clock time, so that the ADC starts to sample at the middle of the PWM positive pulse. Below is the PDB module's initialization code:

```
void pdb1_init(void) {
     PDB1CTRL1 = 0x30U;     // Trigger B is enabled and function of Delay B only
     PDB1CTRL2 = 0x10U;     // Input selects FTM1 init trigger
     PDB1DLYA= 1;           // Trigger A at the beginning
     PDB1DLYB = 300;        // Trigger B at the middle of the PWM
     PDB1CTRL1_LDOK = 1U;   // Load delay values
     PDB1SCR = 0xC0U;       // Enable pulse output on PTD2
}
```

Next step is to initialize the ADC module. What is needed to do is to enable the hardware trigger, select an ADC channel and the Conversion mode. Below is the code:

```
void adc_init(void){
    ADCSC1 = 0x4CU;
           /* COCO(b7) = 0, Conversion Complete Flag */
           /* AIEN(b6) = 1, Conversion complete interrupt enabled */
           /* ADCO(b5) = 0, Continuous conversion disabled */
           /* ADCH(b4:b0) = 01100, Input channel selects AD12 */
    ADCSC2 = 0x40U;
           /* ADACT(b7) = 0, read only conversion active flag */
           /* ADTRG(b6) = 1, hardware trigger selected */
           /* ADFE(b5) = 0, Compare function disabled */
           /* ACFGT(b4) = 0, Compare triggers when input is less than compare value */
     ADCCFG = 0U;
           /* ADLPC(b7) = 0, High speed */
           /* ADIV(b6:b5) = 00, Clock divider */
           /* ADLSMP(b4) = 1, Long sample time */
           /* MODE(b3:b2) = 00, Conversion mode selects 8-bit */
           /* ADICLK(b1:b0) = 00, Input clock select bus clock */
     APCTL1 = 0x00U;
     APCTL2 = 0x10U;         // Select channel of AD12
            /* ADPCn(b7:b0) = 1, ADn pin I/O control disabled */
}
```

To observe the synchronization relationship between the FTM and ADC, the conversion complete interrupt is enabled. A pulse is output to an external pin, in this example PTD3, every time when an ADC conversion is completed. Below is the interrupt service routine code:

```
void interrupt adc_isr(void){
     PTDD_PTDD3 = 1;
     asm nop;
     asm nop;
     PTDD_PTDD3 = 0;
     ADCRL;       // Read to clear COCO flag
}
```

Figure 10 shows the waveforms of PWM output, ADC conversion complete signal, and PDB pulse output.

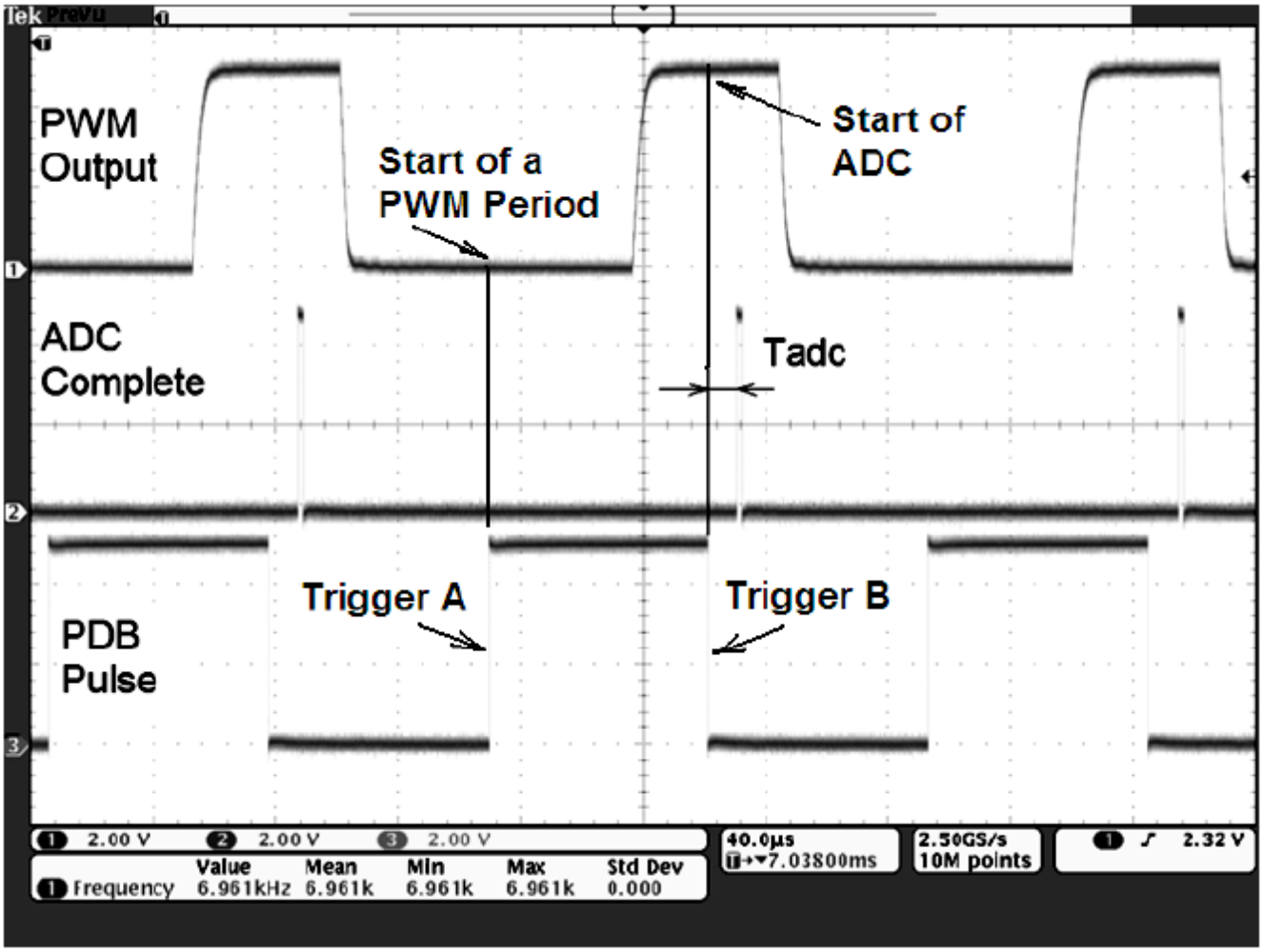

Figure 10. Synchronize FTM and ADC with PDB Trigger B

In Figure 10, the waveform of channel 1 is the PWM output of FTM1; channel 2 is the signal of ADC complete interrupt; and channel 3 is the PDB pulse output. Delay A of the PDB module is set as 1, so that Trigger A occurs almost at the same time when the FTM is reinitialized and starts a new PWM period. Trigger B is delayed to occur at the middle of the PWM signal, and Tadc in this figure represents the sampling and conversion time of the ADC.

3.2 Trigger ADC with Trigger A and Trigger B in the meantime

If both Trigger A and Trigger B are enabled in the meantime, and the PGA is disabled, the ADC will be triggered twice during one PWM period, so that the selected ADC channel would be sampled and converted two times on every period. Modify the PDB initialization codes as shown below:

```
void pdb1_init(void) {
        PDB1CTRL1 = 0x36U;      // Trigger A and Trigger B are enabled
        PDB1CTRL2 = 0x10U;      // Input selects FTM1 init trigger
        PDB1DLYA= 1;            // Trigger A at the beginning
        PDB1DLYB = 300;         // Trigger B at the middle of the PWM
        PDB1CTRL1_LDOK = 1U;    // Load delay values
        PDB1SCR = 0xC0U;        // Enable pulse output on PTD2
}
```

Keep the other initialization codes for other modules unchanged, the waveform will be changed as shown in Figure 11.

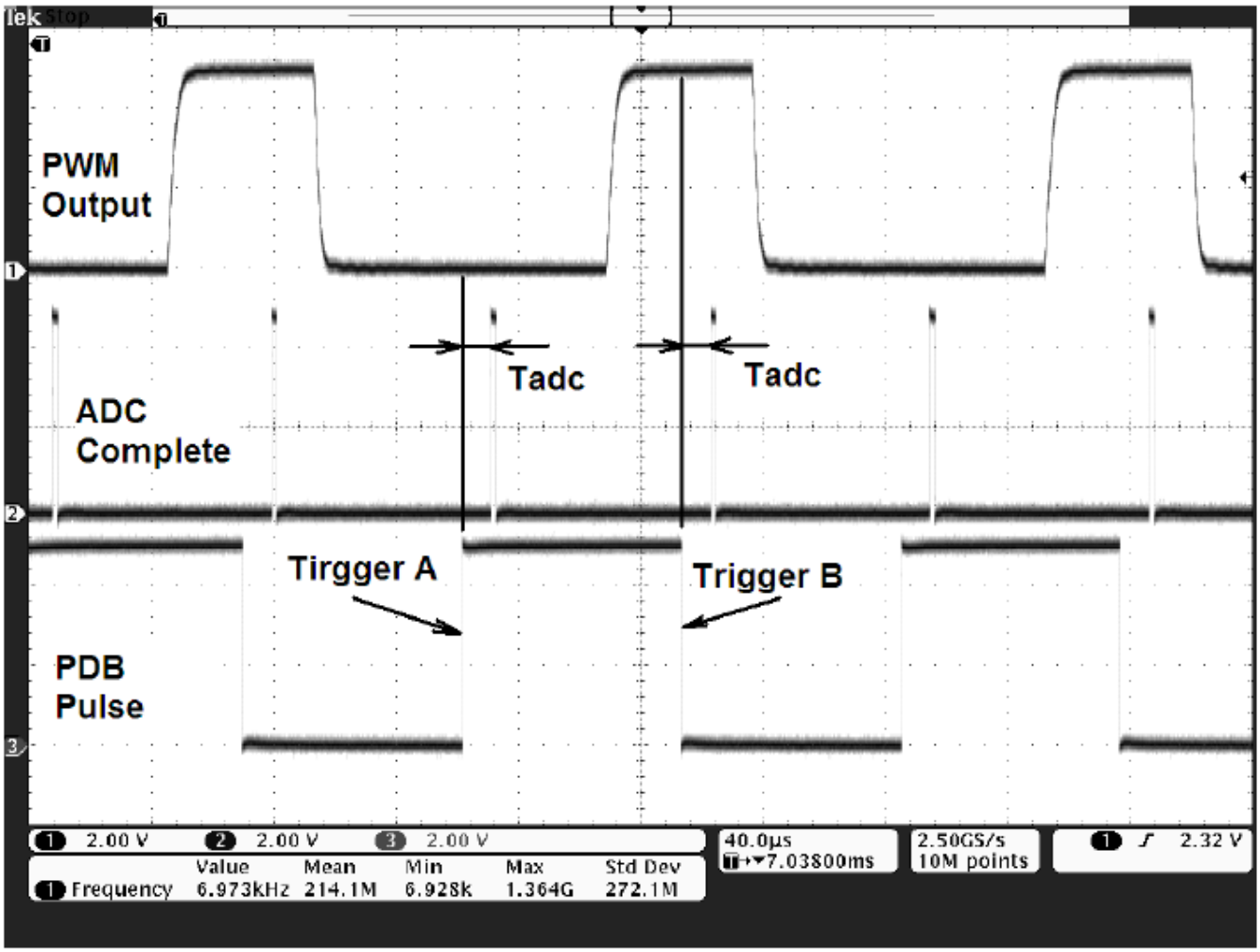

Figure 11. Trigger ADC with both Trigger A and Trigger B

3.3 Trigger PGA with Trigger A and Trigger ADC with Trigger B in the meantime

In some cases the analog signal needs to be amplified before sampled by the ADC module, then the PGA module on MP16 could be used. The PGA module is the programmable gain amplifier that performs differential-to single-ended conversion of analog signals. It has the following features:

- Software and hardware triggers are available
- PGA outputs driven to on-chip ADC input channels
- 1x, 2x, 4x, 8x, 16x, or 32x gain
- •Integrated Sample/Hold circuit

As shown in Figure 8, the PGA module can only be triggered by Trigger A of the PDB. The PGA output connects to channel 13 of the ADC module, and the PGA module will trigger ADC automatically after it completes sampling and conversion. Below is the initialization code to start up the PGA module:

```
void pga_init(void){
     PGACNTL1 = 0x02U;
        /* CALMODE(b4:b3) = 00, Mission mode */
        /* CPD(b2:b0) = 010, Charge Pump frequency divisor */
```

```
    PGACNTL2 = 0x00U;
    PGACNTL0 = 0x05U;
       /* TM(b7) = 0, Hardware Trigger mode */
       /* GAINSEL(b6:b2) = 0001, Gain value is 2 */
       /* LP(b1) = 0, High-power mode */
       /* EN(b0) = 1, PGA enabled */
}
```

Trigger A of the PDB module needs to be enabled to trigger the PGA. Another part needs to be modified because the PGA output is internally connected with channel 13 of the ADC module, so that the user program must switch sampling channels every time when the ADC conversion complete interrupt occurs. Figure 12 is the waveform picture, and it is almost same as Figure 11. But if watched carefully, it is found that Tpga_adc is a little longer than Tadc, because it includes the sampling and conversion time of the PGA.

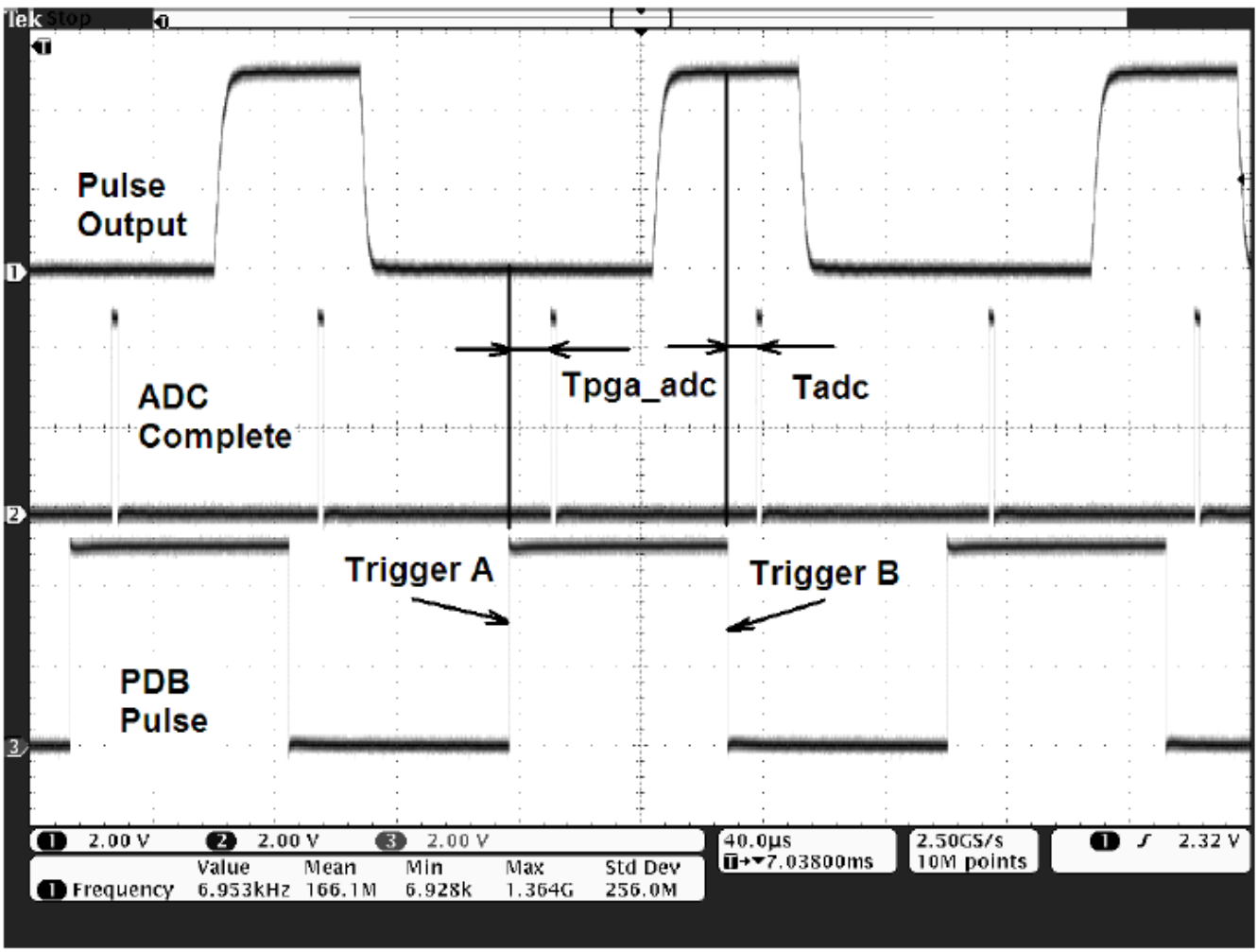

Figure 12. Trigger PGA with Trigger A and Trigger ADC with Trigger B in the meantime

4 Example 3 - synchronize HSCMP and ADC with PDB

In MC9S08MP16 series, there are three integrated high speed comparator modules (HSCMP), referred to as HSCMP1, HSCMP2, and HSCMP3. Each of the HSCMP modules can be configured for Windowing or Sampling modes of operation. The PDB2 PulseOut output is used as the Window/Sample input to all three HSCMP modules. Figure 13 shows the configuration used for PDB2.

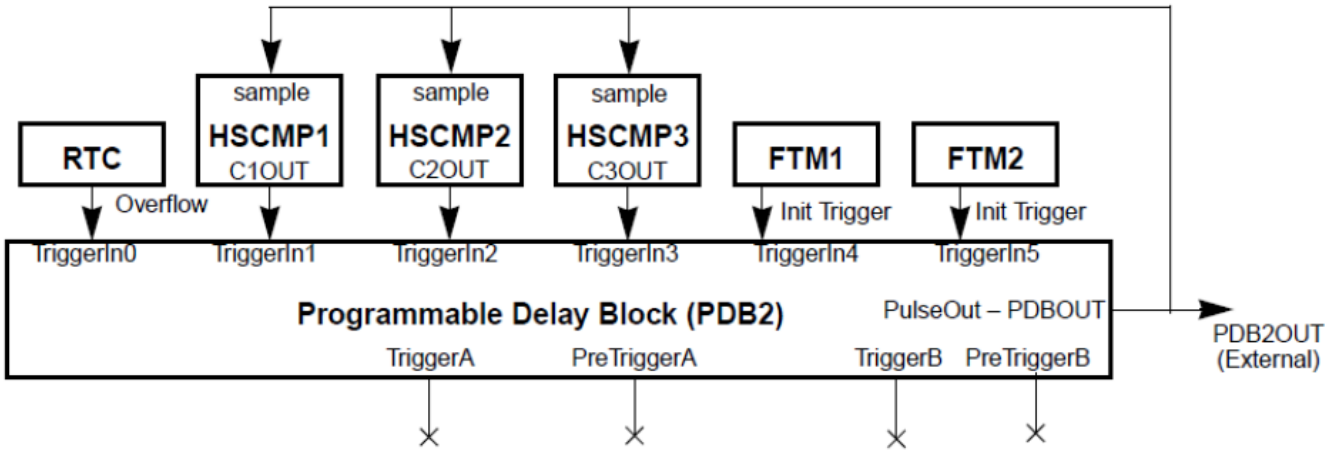

Figure 13. PDB2 interface to HSCMP1/HSCMP2/HSCMP3 Windowingx

In MC9S08MP16 series, the CxOUT output from each of the HSCMP modules can be configured to generate trigger events for the PDB1 and PDB2 modules. HSCMP1 is configured to the TriggerIn1 input of both PDB1 and PDB2 modules. HSCMP2 is configured to the TriggerIn2 input of both PDB1 and PDB2 modules. HSCMP3 is configured to the TriggerIn3 input of both PDB1 and PDB2 modules.

An application example is to synchronize the HSCMP module and the ADC module with the PDB module. In some applications, the ADC module needs to sample at the moment when an analog signal crosses a threshold and the comparator outputs a rising or falling edge, for example the zero-cross signal. With the PDB module, this function can be implemented automatically by hardware. Figure 14 shows the block diagram of modules used to synchronize HSCMP and ADC with PDB.

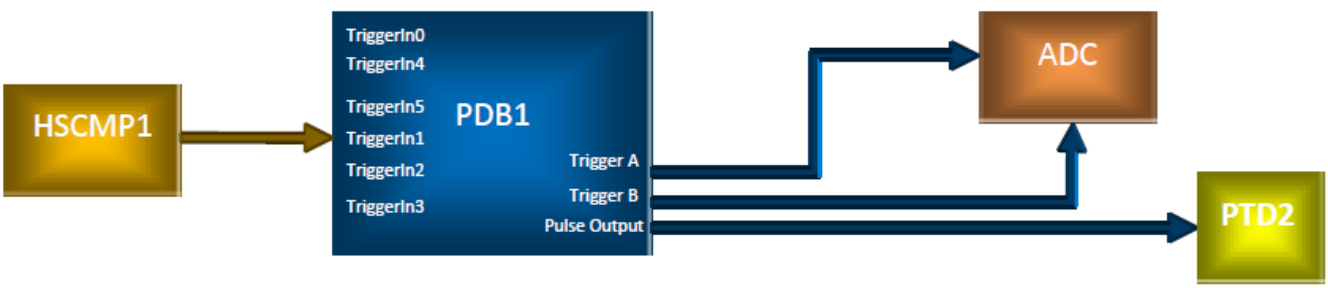

Figure 14. Block diagram of modules used to synchronize HSCMP and ADC with PDB

The output of the HSCMP1 module is selected as the trigger input of the PDB1. Trigger A and/or Trigger B are/is used to trigger the ADC module. If both Trigger A and Trigger B are enabled, the ADC module will be triggered twice every period. Pulse output on PTD2 is enabled to observe the relationship between the trigger signals and the ADC complete signals.

First, initialize the HSCMP1 to work on Continuous mode. Below is the code:

```
void hscmp1_init(void){
     HSCMP1CR1 = 0x02;
         /* SE(b7) = 0, Sampling mode enabled */
         /* WE(b6) = 0, Window mode disabled */
         /* b5 = 0, Reserved and must be cleared */
         /* PMODE(b4) = 0, Power Saving mode */
         /* INV(b3) = 0, Do not invert the comparator output */
         /* COS(b2) = 0, Set COMPO equal to COUT (filtered output) */
         /* OPE(b1) = 1, The comparator output pin enabled */
         /* EN(b0) = 0, Analog comparator disabled */
     HSCMP1SCR = 0x00;
         /* b7:b5 = 000, Reserved and must be cleared */
         /* IER(b4) = 0, Rising edge interrupt disabled */
         /* IEF(b3) = 0, Falling edge interrupt disabled */
         /* CFR(b2) = 0, Rising edge interrupt flag */
```

Example 3 - synchronize HSCMP and ADC with PDB

```
        /* CFF(b1) = 0, Falling edge interrupt flag */
        /* COUT(b0) = 0, Analog comparator output, read only */
    HSCMP1CR0 = 0x0E;
        /* b7 = 0, Reserved and must be cleared */
        /* FLT_CNT(b6:b4) = 000, Filter sample count is 1 */
        /* PMC(b3:b2) = 11, Positive input selects P4, or PTE4 */
        /* MMC(b1:b0) = 10, Minus input selects M3, or PTE3 */
    HSCMP1PCR = 0x84;
        /* INPPE(b7:b4) = 1000, P4 is required by the HSCMP - PTE4 is selected */
        /* INMPE(b7:b4) = 0100, M3 is required by the HSCMP - PTE3 is selected */
     HSCMP1CR1 |= 0x01;
        /* EN(b0) = 1, Analog comparator enabled */
}
```

Second, initialize the ADC module with the same code as the previous example.

The third step is to initialize the PDB. The output of the HSCMP1 is selected as the input of TriggerIn1, and the Trigger A is enabled to trigger the ADC module. Below is the code:

```
void pdb1_init(void) {
    PDB1CTRL1 = 0x06U;// Trigger A is enabled and function of Delay A only
    PDB1CTRL2 = 0x04U;// Select C1OUT as the input trigger
    PDB1DLYA= 1;
    PDB1DLYB = 300;
    PDB1CTRL1_LDOK = 1U;
    PDB1SCR = 0xC0U;// Enable pulse output on PTD2
}
```

When all the above initialization functions are executed, the ADC module will be triggered every time when the HSCMP module output a rising edge. Figure 15 is the waveform record.

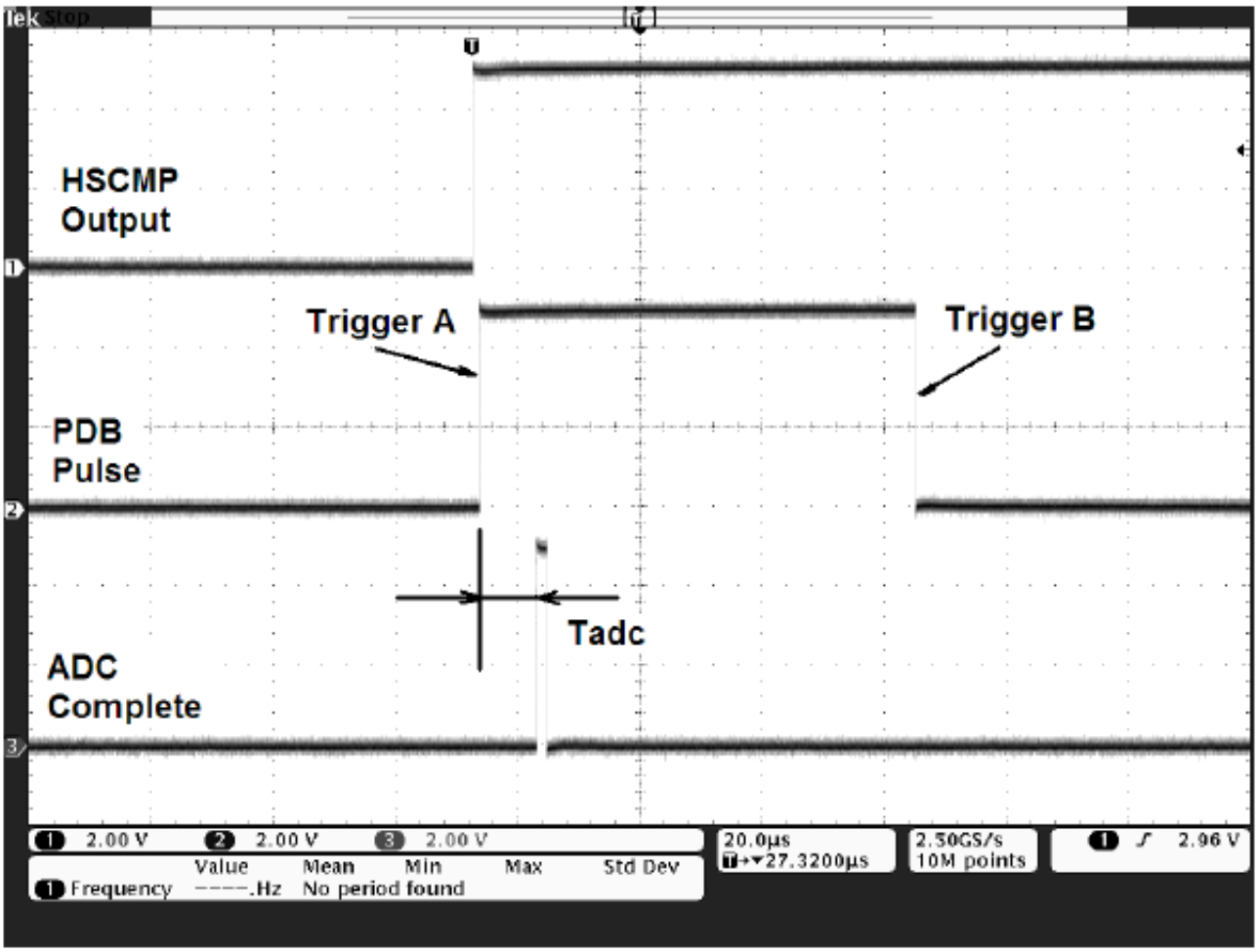

Figure 15. Synchronize HSCMP and ADC with PDB

5 Example 4 - synchronize FTM, HSCMP, and ADC with PDB

In MC9S08MP16 series, each of the HSCMP modules can be configured for Windowing or Sampling modes of operation. The PDB2 PulseOut output is used as the Window/Sample input to all three HSCMP modules. Figure 11 shows the configuration used for PDB2.

Here is an application scenario: the ADC module must sample during the period when PWM is outputting high level, and it also needs to sample at the moment of a rising or falling edge, such as at the zero-cross moment, of the comparator output in the sampling window. This could be implemented with the windowing modes of the HSCMP module. However, it needs to use two PDB modules, as shown in Figure 16.

In Figure 16, the HSCMP1 module outputs the rising or falling edge, and triggers the ADC module with the PDB1 module. The sampling window is generated by FTM1 and PDB2 modules. The PDB2 module is triggered by the Init-Trigger from FTM1, so that it can output a pulse signal which is synchronized with the PWM signal generated by FTM1. The pulse signal is used as the windowing signal for HSCMP1. Only rising or falling edges in the window period would be sampled by HSCMP1 and generate the corresponding C1OUT to trigger PDB1.

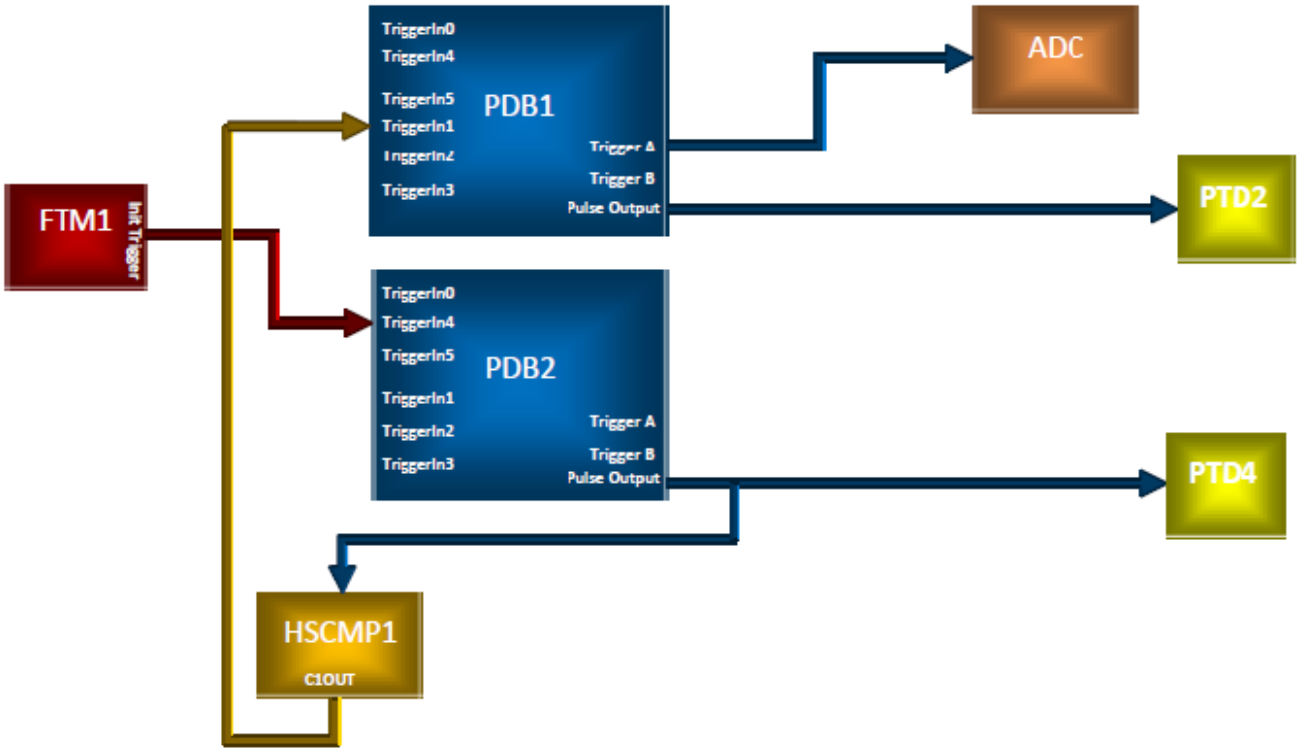

Figure 16. Block diagram of connections to synchronize FTM, HSCMP, and ADC with PDB

For this application example, five modules are used and need to be initialized, they are: FTM1, HSCMP1, PDB2, PDB1, and ADC.

As the previous example, FTM1 is initialized to generate a center-aligned PWM output with period of 600, bus clock time, and duty cycle of 1/3. The initialization code is exactly same as the previous one.

```
void hscmp1_init(void){
    HSCMP1CR1 = 0x42;
       /* SE(b7) = 0, Sampling mode enabled */
       /* WE(b6) = 1, Window mode enabled */
       /* b5 = 0, Reserved and must be cleared */
       /* PMODE(b4) = 0, Power Saving mode */
       /* INV(b3) = 0, Do not invert the comparator output */
       /* COS(b2) = 0, Set COMPO equal to COUT (filtered output) */
       /* OPE(b1) = 1, The comparator output pin enabled */
       /* EN(b0) = 0, Analog comparator disabled */
    HSCMP1SCR = 0x00;
       /* b7:b5 = 000, Reserved and must be cleared */
       /* IER(b4) = 0, Rising edge interrupt disabled */
       /* IEF(b3) = 0, Falling edge interrupt disabled */
       /* CFR(b2) = 0, Rising edge interrupt flag */
       /* CFF(b1) = 0, Falling edge interrupt flag */
       /* COUT(b0) = 0, Analog comparator output, read only */
    HSCMP1CR0 = 0x0E;
       /* b7 = 0, Reserved and must be cleared */
       /* FLT_CNT(b6:b4) = 000, Filter sample count is 1 */
       /* PMC(b3:b2) = 11, Positive input selects P4, or PTE4 */
       /* MMC(b1:b0) = 10, Minus input selects M3, or PTE3 */
     HSCMP1PCR = 0x84;
       /* INPPE(b7:b4) = 1000, P4 is required by the HSCMP - PTE4 is selected */
       /* INMPE(b7:b4) = 0100, M3 is required by the HSCMP - PTE3 is selected */
```

```
      HSCMP1CR1 |= 0x01;
        /* EN(b0) = 1, Analog comparator enabled */
}
```

The PDB2 module needs to be initialized to select the Init-Trigger signal from FTM1 as the trigger input. It needs to set the Delay A and Delay B value correctly to supply a precise time window for HSCMP1 to sample. Below is the code:

```
void pdb2_init(void){
       PDB2CTRL1 = 0x00U;       // Trigger A and Trigger B are disabled
       PDB2CTRL2 = 0x10U;       // Input selects FTM1 init trigger
       PDB2DLYA= 250;
       PDB2DLYB = 350;
       PDB2CTRL1_LDOK = 1U;
       PDB2SCR = 0xC0U;         // Enable pulse output on PTD2
}
```

As shown in Figure 17, the modulo register of FTM1 is set as 300 while FTM1 is set to output center-aligned PWM signals. So the FTM1 counter counts up from 0 to 300, then counts down from 300 to 0. The channel 0 register of FTM1 is set as 200, so that the PWM signal has period of 600, and its duty cycle is 200/600, or 1/3. For the PDB2, if Delay A is set as 250, and Delay B as 350, it would output a pulse with width of 100, center-aligned and exactly synchronized with the PWM signal. This pulse signal would be used as the sampling window for the HSCMP1 module. In other words, only analog input inside this window would be sampled by HSCMP1.

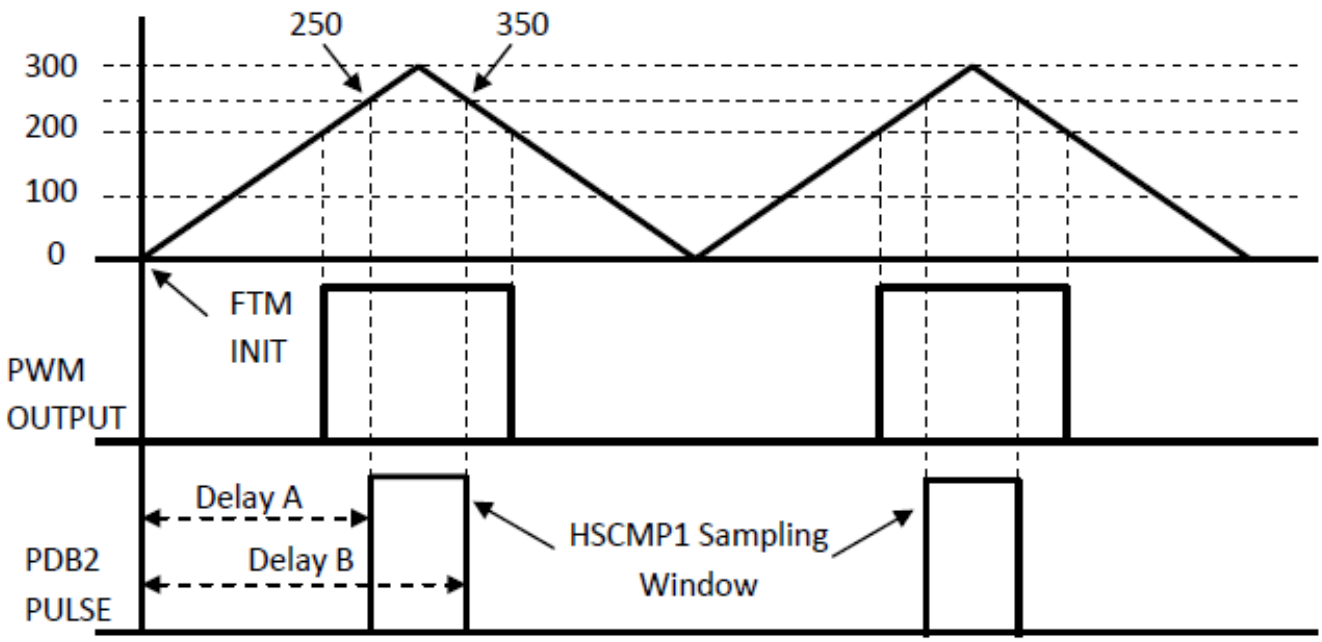

Figure 17. Generate a synchronized pulse as windowing signal for HSCMP1 with PDB2

As shown in Figure 17, if the analog input values of HSCMP1 changed and cause its output a rising edge, it will trigger PDB1, then PDB1 will trigger ADC after a delay. The PDB1 module needs to be initialized to select the C1OUT from HSCMP1 as the input trigger, and then output the Trigger A signal to the ADC module. Below is the initialization code:

```
void pdb1_init(void) {
        PDB1CTRL1 = 0x06U;// Trigger A is enabled and function of Delay A only
        PDB1CTRL2 = 0x04U;// Input select C1OUT trigger
        PDB1DLYA= 1;
        PDB1DLYB = 300;
        PDB1CTRL1_LDOK = 1U;
        PDB1SCR = 0xC0U;// Enable pulse output on PTD2
}
```

The initialization code for the ADC module is same as that of the previous example. Figure 18 shows the waveforms.

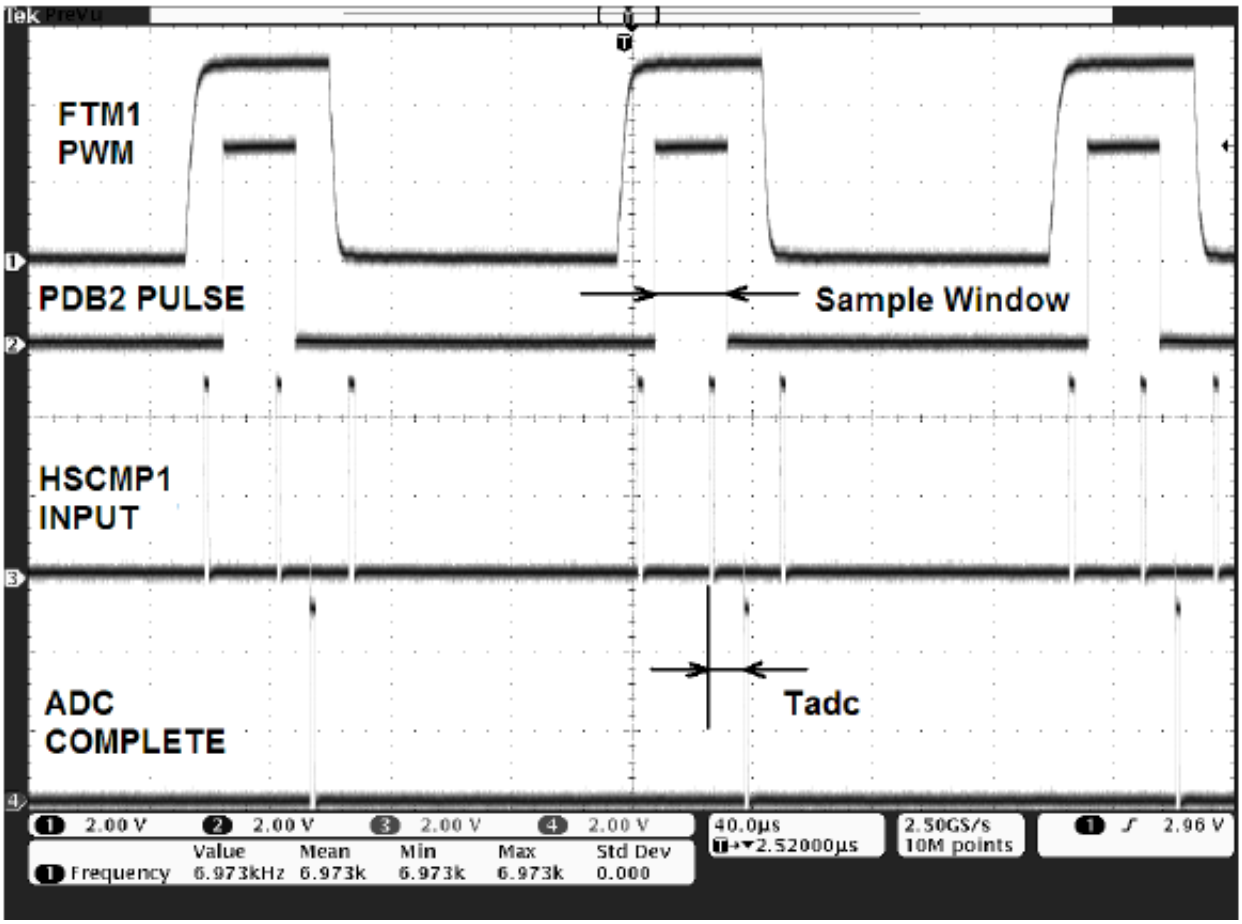

Figure 18. HSCMP1 samples in window generated by PDB2

Figure 18 shows that the PDB2 module outputs the pulses that are precisely synchronized with the PWM signals generated by FTM1. In every PWM period there are three positive pulses input to HSCMP1, but only the middle one inside the sampling window would trigger the ADC. All the other inputs outside the sampling window are ignored. That is a powerful feature for many applications such as motor control, and it is implemented by hardware, without any software processing.

6 Conclusions

The PDB modules are very powerful and versatile to synchronize timers and analog modules, such as ADC, PGA, and HSCMP. In this application note, some basic application examples are discussed to introduce how to use the PDB modules to implement the synchronization. In fact, the PDB modules can achieve more complicated functions if more modules are connected and triggered to one another through them. Users need to set them according to actual application requirements, and software processing may be required in some sophisticated application cases.

7 References

1. Reference Manual: *MC9S08MP16RM.pdf*, available at http://www.freescale.com
2. Datasheet: *MC9S08MP16DS.pdf*, available at http://www.freescale.com
3. Demo Board Schematic: *DEMO9S08MP16_RevB_Schematic.pdf*, available at http://www.freescale.com

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4424
Rev. 0, 12/2011