CHAPTER 8

Counter/Timers

Counters count rising edges of a pulse train. The pulses may come from the internal peripheral bus clock or external sources. If a fixed frequency clock produces the pulses, counters become timers (the count represents a time). Therefore, the words "counter" and "timer" are often used interchangeably. Because Microchip's documentation refers to these devices as "timers," we adopt that terminology. Timers can generate interrupts after a preset number of pulses has been counted or on the falling edge of an external pulse whose duration is being timed. These timers differ from the core timer introduced in Chapter 5 because they are peripherals rather than part of the MIPS32 CPU.

8.1 Overview

The PIC32 is equipped with five 16-bit peripheral timers named Timerx, where x is 1 to 5. A timer increments on the rising edge of a clock signal, which may come from the PBCLK or from an external source of pulses. The input for an external source for Timerx is pin TxCK. For the 64-pin PIC32MX795F512H on the NU32 board, only Timer1 is equipped with a pin for an external input (T1CK). The 100-pin version of this chip (the PIC32MX795F512L) has an external input pin for all five timers.

A prescaler $N \geq 1$ determines how many clock pulses must be received before the timer increments. If the prescaler is set to $N = 1$, the timer increments on every clock rising edge; if it is set to $N = 8$, it increments every eighth rising edge. The clock source type (internal or external) and the prescaler value is chosen by setting the value of the SFR TxCON.

Each 16-bit timer can count from 0 up to a period $P \leq 2^{16} - 1 = 65{,}535 = 0\text{xFFFF}$. The current count is stored in the SFR TMRx and the value of P can be chosen by writing to the period register SFR PRx. When the timer reaches the value P, a *period match* occurs, and after N more pulses are received, the counter "rolls over" to 0. If the input to the timer is the 80 MHz PBCLK, with 12.5 ns between rising edges, then the time between rollovers is $T = (P + 1) \times N \times 12.5$ ns. Choosing $N = 8$ and $P = 9,999$, we get $T = 1$ ms, and changing N to 64 gives $T = 8$ ms. By configuring the timer to use the PBCLK as input and to generate an interrupt when a period match occurs, the timer can implement a function that runs at a fixed frequency (a controller, for example) (see Figure 8.1).

Embedded Computing and Mechatronics with the PIC32 Microcontroller. http://dx.doi.org/10.1016/B978-0-12-420165-1.00008-1

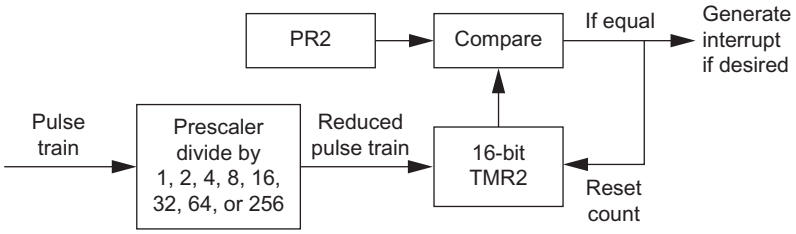

Figure 8.1
Simplified block diagram for a typical use of the 16-bit Timer2. The pulse train feeds a prescaler, which produces one output pulse for every N input pulses ($N = 1, 2, 4, 8, 16, 32, 64$, or 256). The TMR2 SFR stores the count of these pulses. TMR2 resets to zero on the first pulse in the reduced pulse train after TMR2 matches the period register PR2. By default, PR2 is 0xFFFF, so TMR2 counts up to $2^{16} - 1$ before rolling over to zero. Timers 3, 4, and 5 are similar to Timer 2, but Timer 1 can only have prescaler values $N = 1, 8, 64$, or 256.

If the period P is zero, then once the count reaches zero it will never increment again (it keeps rolling over). No interrupt can be generated by a period match if $P = 0$.

The PIC32 has two types of timers: Type A and Type B, each with slightly different features (explained shortly). Timer1 is Type A and Timer2 to Timer5 are Type B. The timers can be used in the following modes, chosen by the SFR TxCON:

Counting PBCLK pulses. In this mode, the timer counts PBCLK pulses, so the count corresponds to an elapsed time. This mode is often used to generate interrupts at a desired frequency by appropriate setting of N and P. It can also be used to time the duration of code, like how we used the core timer in Chapter 5. A peripheral timer, however, can increment once every N PBCLK cycles, including $N = 1$, not just every 2 SYSCLK cycles.

Synchronous counting of external pulses. For Timer1, an external pulse source is connected to the pin T1CK. The timer count increments after each rising edge of the external source. This mode is called "synchronous" because timer increments are synchronized to the PBCLK; the timer does not actually increment until the first rising edge of PBCLK after the rising edge of the external source. If the external pulses are too fast, the timer will not accurately count them. According to the Electrical Characteristics section of the Data Sheet, the duration of the high and low portions of a pulse should be at least 37.5 ns each.

Asynchronous counting of external pulses (Type A Timer1 only). The Type A pulse counting circuit can be configured to increment independently of the PBCLK, allowing it to count even when the PBCLK is not operating, such as in the power-saving Sleep mode. If a period match occurs, Timer1 can generate an interrupt and wake up the PIC32. When used in asynchronous mode, the timer can count pulses with high and low durations as low as 10 ns each.

Timing the duration of an external pulse. Also called "gated accumulation mode." For Timer1, when the input on external pin T1CK goes high, the counter starts incrementing according to the PBCLK and the prescaler *N*. When the input drops low, the count stops. The timer can also generate an interrupt when the input drops low.

Important differences between Timer1 (Type A) and Timer2 to Timer5 (Type B) are:

- Only Timer1 can count external pulses on the PIC32MX795F512H.
- Timer1 can have prescalers $N = 1$, 8, 64, and 256, while Timer2 to Timer5 can have prescalers $N = 1$, 2, 4, 8, 16, 32, 64, and 256.
- Timer2 and Timer3 can be chained together to make a single 32-bit timer, called Timer23. Timer4 and Timer5 can similarly be used to make a single 32-bit timer, called Timer45. These combined timers allow counts of up to $2^{32} - 1$, or over 4 billion. When two timers are used to make Timerxy (x < y), Timerx is called the "master" and Timery is the "slave"—only the prescaler and mode information in TxCON are relevant, while those fields in TyCON are ignored. When Timerx rolls over from $2^{16} - 1$ to 0, it sends a clock pulse to increment Timery. In Timerxy mode, the 16 bits of TMRy are also stored in the most significant 16 bits of the SFR TMRx, so the 32 bits of TMRx contain the full count of Timerxy. Similarly, the 32 bits of PRx contain the 32-bit period match value *P*xy. The interrupt associated with a period match (or a falling input in gated accumulation mode) is actually generated by Timery, so interrupt settings should be chosen for Timery's IRQ and vector.

Timers are used in conjunction with digital waveform generation by the Output Compare peripheral (Chapter 9) and in timing digital input waveforms by the Input Capture peripheral (Chapter 15). A timer can also be used to repeatedly trigger analog to digital conversions (Chapter 10).

8.2 Details

The following SFRs are associated with the timers. All SFRs default to 0x0000, except the PRx SFRs, which default to 0xFFFF. First, we describe settings common to both Type A and Type B timers.

TxCON, x = 1 to 5 The Timerx control SFR configures the behavior of Timerx. Important bits common to both Type A and Type B timers include

TxCON⟨15⟩, or TxCONbits.ON: Enables and disables the timer.

1 Timerx enabled.

0 Timerx disabled.

TxCON⟨7⟩, or TxCONbits.TGATE: Sets gated accumulation mode, which can be used to time the duration of an external pulse. In gated accumulation mode the timer starts counting when an external signal goes high and stops when it goes low.

1 Gated accumulation mode enabled.
0 Gated accumulation mode disabled.

Gated accumulation also requires TxCONbits.TCS = 0 (below).

TxCON⟨1⟩, or TxCONbits.TCS: Determines whether the timer uses an external clock source or PBCLK. On the PIC32MX795F512H, only Timer1 has a pin for an external clock source.

1 Timerx uses the signal on TxCK as an external pulse source.
0 PBCLK provides the pulse source.

The Type A timer, Timer1, also has the relevant fields:

T1CON The control register for Timer1. Has the same fields as above; here we describe the fields specific to Timer1.

T1CON⟨5:4⟩, or T1CONbits.TCKPS: Sets the prescaler ratio. The prescaler determines how many pulses are required to increment the timer count. Type A timers have fewer prescaler ratios than Type B timers.

0b11 Prescaler of 1:256.
0b10 Prescaler of 1:64.
0b01 Prescaler of 1:8.
0b00 Prescaler of 1:1.

T1CON⟨2⟩, or T1CONbits.TSYNC: Determines whether external clock inputs are synchronized to PBCLK. When synchronized, the timer counts external rising edges on the PBCLK ticks. When asynchronous, every external pulse is registered immediately (subject to timing requirements specified in the Data Sheet). Only Type A timers can count pulses asynchronously.

1 External counting is synchronized.
0 External counting is asynchronous.

The Type B timers, Timer2 to Timer5, have their own type-specific TxCON fields.

TxCON, x = 2 to 5 Here we describe the fields in TxCON specific to Type B timers.

TxCON⟨6:4⟩, or TxCONbits.TCKPS: Sets the prescaler ratio. There are more choices than for Type A timers.

0b111 Prescaler of 1:256.
0b110 Prescaler of 1:64.
0b101 Prescaler of 1:32.
0b100 Prescaler of 1:16.
0b011 Prescaler of 1:8.
0b010 Prescaler of 1:4.
0b001 Prescaler of 1:2.
0b000 Prescaler of 1:1.

TxCON⟨3⟩, or TxCONbits.T32: This bit is only relevant for x = 2 and 4 (Timer2 and Timer4). When set, Timerx and Timery are chained together to make the 32 bit

timer called Timerxy (y = x + 1). When in 32-bit mode, TyCON settings are ignored, TMRy is enabled, and its clock value comes from the rollover of TMRx after it hits 0xFFFF. Interrupts are generated by Timery, but the timer's full 32-bit value and 32-bit rollover count are accessed via TMRx and PRx. When TxCONbits.T32 is zero, Timerx and Timery operate as independent 16-bit timers.

1 Use Timer23 (if x = 2) or Timer45 (if x = 4) as 32-bit timers.

0 User Timerx as a 16-bit timer.

The following SFRs apply to each Timerx, x = 1 to 5.

TMRx, x = 1 to 5 TMRx⟨15:0⟩ stores the 16-bit count of Timerx. TMRx resets to 0 on the next count after TMRx reaches the number stored in PRx. This rollover process is called a period match. In Timerxy 32-bit mode, TMRx contains the 32-bit value of the chained timer, and period match occurs when TMRx = PRx.

PRx, x = 1 to 5 PRx⟨15:0⟩ contains the maximum value of the count of TMRx before it resets to zero on the next count. An interrupt can be generated on this period match. In Timerxy 32-bit timer mode, PRx contains the 32-bit value of the period *P*xy. Interrupts are generated as if Timery triggered the interrupt.

The timer can generate an interrupt on the falling edge of the gate input when it is in gated mode (TxCONbits.TCS = 0 and TxCONbits.TGATE = 1). Otherwise, it can generate an interrupt whenever a period match occurs.

The relevant interrupt flags are shown in Table 8.1. To enable the interrupt for Timerx, the interrupt enable bit IEC0bits.TxIE must be set. The interrupt flag bit IFS0bits.TxIF should be cleared and the priority and subpriority bits IPCxbits.TxIP and IPCxbits.TxIS must be written. In 32-bit Timerxy mode, interrupts are generated by Timery; interrupt settings for Timerx are ignored.

Table 8.1: Vectors and bits relevant to timer interrupts

IRQ Source	Vector	Flag	Enable	Priority	Subpriority
Timer1	4 _TIMER_1_VECTOR	IFS0⟨4⟩ IFS0bits.T1IF	IEC0⟨4⟩ IEC0bits.T1IE	IPC1⟨4:2⟩ IPC1bits.T1IP	IPC1⟨1:0⟩ IPC1bits.T1IS
Timer2	8 _TIMER_2_VECTOR	IFS0⟨8⟩ IFS0bits.T2IF	IEC0⟨8⟩ IEC0bits.T2IE	IPC2⟨4:2⟩ IPC2bits.T2IP	IPC2⟨1:0⟩ IPC2bits.T2IS
Timer3	12 _TIMER_3_VECTOR	IFS0⟨12⟩ IFS0bits.T3IF	IEC0⟨12⟩ IEC0bits.T3IE	IPC3⟨4:2⟩ IPC3bits.T3IP	IPC3⟨1:0⟩ IPC3bits.T3IS
Timer4	16 _TIMER_4_VECTOR	IFS0⟨16⟩ IFS0bits.T4IF	IEC0⟨16⟩ IEC0bits.T4IE	IPC4⟨4:2⟩ IPC4bits.T4IP	IPC4⟨1:0⟩ IPC4bits.T4IS
Timer5	20 _TIMER_5_VECTOR	IFS0⟨20⟩ IFS0bits.T5IF	IEC0⟨20⟩ IEC0bits.T5IE	IPC5⟨4:2⟩ IPC5bits.T5IP	IPC5⟨1:0⟩ IPC5bits.T5IS

8.3 Sample Code

8.3.1 A Fixed Frequency ISR

To create a 5 Hz ISR with an 80 MHz PBCLK, the interrupt must be triggered every 16 million PBCLK cycles. The highest a 16-bit timer can count is $2^{16} - 1$. Instead of wasting two timers to make a 32-bit timer with a prescaler $N = 1$, let us use a single 16-bit timer with a prescaler $N = 256$. We shall use Timer1. We should choose PR1 to satisfy

```
16,000,000 = (PR1 + 1) x 256
```

that is, PR1 = 62,499. The ISR in the following code toggles a digital output at 5 Hz, creating a 2.5 Hz square wave (a flashing LED on the NU32).

Code Sample 8.1 TMR_5Hz.c**. Timer1 Toggles RF0 Five Times a Second (LED1 on the NU32 Flashes).**

```
#include "NU32.h"          // constants, functions for startup and UART

void __ISR(_TIMER_1_VECTOR, IPL5SOFT) Timer1ISR(void) {  // INT step 1: the ISR
  LATFINV = 0x1;                  // toggle RF0 (LED1)
  IFS0bits.T1IF = 0;              // clear interrupt flag
}

int main(void) {
  NU32_Startup(); // cache on, min flash wait, interrupts on, LED/button init, UART init

  __builtin_disable_interrupts(); // INT step 2: disable interrupts at CPU
                                  // INT step 3: setup peripheral
  PR1 = 62499;                    //             set period register
  TMR1 = 0;                       //             initialize count to 0
  T1CONbits.TCKPS = 3;            //             set prescaler to 256
  T1CONbits.TGATE = 0;            //             not gated input (the default)
  T1CONbits.TCS = 0;              //             PCBLK input (the default)
  T1CONbits.ON = 1;               //             turn on Timer1
  IPC1bits.T1IP = 5;              // INT step 4: priority
  IPC1bits.T1IS = 0;              //             subpriority
  IFS0bits.T1IF = 0;              // INT step 5: clear interrupt flag
  IEC0bits.T1IE = 1;              // INT step 6: enable interrupt
  __builtin_enable_interrupts();  // INT step 7: enable interrupts at CPU
  while (1) {
      ;                           // infinite loop
  }
  return 0;
}
```

8.3.2 Counting External Pulses

The following code uses the 16-bit Timer1 to count the rising edges on the input T1CK. The 32-bit Timer45 creates an interrupt at 2 kHz to toggle a digital output, generating a 1 kHz

pulse train on RD1 that acts as input to T1CK. Although a 16-bit timer can certainly generate a 2 kHz interrupt, we use a 32-bit timer just to show the configuration. In Chapter 9 we will learn about the Output Compare peripheral, a better way to use a timer to create more flexible waveforms.

To create an IRQ every 0.5 ms (2 kHz), we use a prescaler $N = 1$ and a period match PR4 = 39,999, so

```
(PR4 + 1) x N x 12.5 ns = 0.5 ms
```

The code below displays to your computer's screen the amount of time that has elapsed since the PIC32 was reset, in milliseconds. If you wait 65 s, you will see Timer1 roll over.

Code Sample 8.2 TMR_external_count.c**. Timer45 Creates a 1 kHz Pulse Train on RD1, and These External Pulses Are Counted by Timer1. The Elapsed Time Is Periodically Reported Back to the Host Computer Screen.**

```
#include "NU32.h"          // constants, functions for startup and UART

void __ISR(_TIMER_5_VECTOR, IPL4SOFT) Timer5ISR(void) {  // INT step 1: the ISR
  LATDINV = 0x02;                                        // toggle RD1
  IFS0bits.T5IF = 0;                                     // clear interrupt flag
}

int main(void) {
  char message[200] = { };
  int i = 0;

  NU32_Startup();                 // cache on, interrupts on, LED/button init, UART init
  __builtin_disable_interrupts(); // INT step 2: disable interrupts

  TRISDbits.TRISD1 = 0;           // make D1 an output. connect D1 to T1CK (C14)!

                                  // configure Timer1 to count external pulses.
                                  // The remaining settings are left at their defaults
  T1CONbits.TCS = 1;              // count external pulses
  PR1 = 0xFFFF;                   // enable counting to max value of 2^16 - 1
  TMR1 = 0;                       // set the timer count to zero
  T1CONbits.ON = 1;               // turn Timer1 on and start counting

                                  // 1 kHz pulses with 2 kHz interrupt from Timer45
  T4CONbits.T32 = 1;              // INT step 3: set up Timers 4 and 5 as 32-bit Timer45
  PR4 = 39999;                    //             rollover at 40,000; 80MHz/40k = 2 kHz
  TMR4 = 0;                       //             set the timer count to zero
  T4CONbits.ON = 1;               //             turn the timer on
  IPC5bits.T5IP = 4;              // INT step 4: priority for Timer5 (int goes with T5)
  IFS0bits.T5IF = 0;              // INT step 5: clear interrupt flag
  IEC0bits.T5IE = 1;              // INT step 6: enable interrupt
  __builtin_enable_interrupts();  // INT step 7: enable interrupts at CPU

  while (1) {
                                  // display the elapsed time in ms
    sprintf(message,"Elapsed time:  %u ms\r\n", TMR1);
    NU32_WriteUART3(message);
```

```
    for(i = 0; i < 10000000; ++i){// loop to delay printing
      _nop();                     // include nop so loop is not optimized away
    }
  }
  return 0;
}
```

8.3.3 Timing the Duration of an External Pulse

In this last example we modify our previous code to use Timer45 to toggle the digital output RD1 every 100 ms, creating a 5 Hz square wave. These pulses are timed by Timer1 in gated accumulation mode. The accumulated count begins when the T1CK input from RD1 goes high and stops when the T1CK input drops low. The falling edge triggers an ISR that displays the Timer1 count to the screen and resets the timer. You should find that the measured time is very close to 100 ms, as expected.

Code Sample 8.3 TMR_pulse_duration.c**. Timer45 Creates a Series of 100 ms Pulses on RD1. These Pulses Are Input to T1CK and Timer1 Measures Their Duration in Gated Accumulation Mode.**

```
#include "NU32.h"          // constants, functions for startup and UART

void __ISR(_TIMER_5_VECTOR, IPL4SOFT) Timer5ISR(void) {  // INT step 1: the ISR
  LATDINV = 0x02;                 // toggle RD1
  IFS0bits.T5IF = 0;              // clear interrupt flag
}

void __ISR(_TIMER_1_VECTOR, IPL3SOFT) Timer1ISR(void) { // INT step 1: the ISR
  char msg[100] = { };
  sprintf(msg,"The count was %u, or %10.8f seconds.\r\n", TMR1, TMR1/312500.0);
  NU32_WriteUART3(msg);
  TMR1 = 0;                       // reset Timer1
  IFS0bits.T1IF = 0;              // clear interrupt flag
}

int main(void) {
  NU32_Startup();                 // cache on, interrupts on, LED/button init, UART init

  __builtin_disable_interrupts(); // INT step 2: disable interrupts

  TRISDbits.TRISD1 = 0;           // make D1 an output. connect D1 to T1CK (C14)

                                  // INT step 3
  T1CONbits.TGATE = 1;            //             Timer1 in gated accumulation mode
  T1CONbits.TCKPS = 3;            //             1:256 prescale ratio
  T1CONbits.TCS = 0;
  PR1 = 0xFFFF;                   //             use the full period of Timer1
  T1CONbits.TON = 1;              //             turn Timer1 on

  T4CONbits.T32 = 1;              // for T45:    enable 32 bit mode Timer45
  PR4 = 7999999;                  //             set PR so timer rolls over at 10 Hz
  TMR4 = 0;                       //             initialize count to 0
  T4CONbits.TON = 1;              //             turn Timer45 on
```

```
    IPC5bits.T5IP = 4;              // INT step 4: priority for Timer5 (int for Timer45)
    IPC1bits.T1IP = 3;              //             priority for Timer1
    IFS0bits.T5IF = 0;              // INT step 5: clear interrupt flag for Timer45
    IFS0bits.T1IF = 0;              //             clear interrupt flag for Timer1
    IEC0bits.T5IE = 1;              // INT step 6: enable interrupt for Timer45
    IEC0bits.T1IE = 1;              //             enable interrupt for Timer1
    __builtin_enable_interrupts();  // INT step 7: enable interrupts at the CPU

    while (1) {
      ;
    }
    return 0;
  }
```

8.4 Chapter Summary

- The PIC32 timers can be used to generate fixed-frequency interrupts, count external pulses, and time the duration of external pulses. Additionally, the Type A Timer1 can asynchronously count external pulses even when the PIC32 is in Sleep mode, while the Type B timers Timer2 and Timer3 can be chained to make the 32-bit timer Timer23. Similarly, Timer4 and Timer5 can be chained to make the 32-bit timer Timer45.
- For a 32-bit timer Timerxy, the timer configuration information in TxCON is used (TyCON is ignored), and the interrupt enable, flag status, and priority bits are configured for Timery (this information for Timerx is ignored). The 32-bit Timerxy count is held in TMRx and the 32-bit period match value is held in PRx.
- A timer can generate an interrupt when either the external pulse being timed falls low (gated accumulation mode) or the count reaches a value stored in a period register (period match).

8.5 Exercises

1. Assume PBCLK is running at 80 MHz. Give the four-digit hex values for T3CON and PR3 so that Timer3 is enabled, has a 1:64 prescaler, and rolls over (generates an interrupt) every 16 ms.
2. Using a 32-bit timer (Timer23 or Timer45) to count rising edges from the 80 MHz PBCLK, what is the longest duration you can time, in seconds, before the timer rolls over? (Use the prescaler that maximizes this time.)

Further Reading

PIC32 family reference manual. Section 14: Timers. (2013). Microchip Technology Inc.