



Using the Z8 Encore! XP[®] Timer

AN013104-1207

Abstract

Zilog's Z8 Encore! XP[®] microcontroller consists of four 16-bit reloadable timers that can be used for timing, event counting or for generating pulse-width modulation (PWM) signals.

This Application Note highlights the Z8 Encore! XP Timer functionality and provides a set of routines to use the Timer in different modes, such as the ONE-SHOT, CONTINUOUS, COUNTER, PWM, CAPTURE, COMPARE, GATED, and the CAPTURE/COMPARE modes.

The Timer functionality is configured through API functions that initialize, enable, disable, reload, start and reset the Timer, and set the PWM, Timer interrupt priority and Timer output.

► **Note:** *The source code file associated with this Application Note, AN0131-SC01.zip is available for download on www.zilog.com.*

Z8 Encore! XP[®] Flash MCU Overview

Zilog's Z8 Encore! XP products are based on the new eZ8 CPU and introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows for faster development time and program changes in the field. The high-performance register-to-register based architecture of the eZ8 core maintains backward compatibility with Zilog's popular Z8[®] microcontroller unit (MCU).

Featuring eZ8 CPU, the new Z8 Encore! XP microcontrollers combine a 20 MHz core with Flash memory, linear-register SRAM, and an

extensive array of on-chip peripherals. These peripherals make the Z8 Encore! XP suitable for a variety of applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Overview of Timers

Timers are useful in a variety of day-to-day applications. They can be used to maintain an accurate time-of-day clock, set an alarm to turn OFF after some elapsed time, measure elapsed time between two externally occurring events, count moving objects on a conveyor belt or, generate a melody or a DTMF dialing, etc. In control system applications, Timers can be used to perform tasks at regular intervals, like event captures, and generation of events and pulses of different frequencies with varying duty cycles.

Timers can be on-chip (that is, hardware implemented) or can be implemented in software. With software, Timers can be implemented using for-loops or counters. However, such timing depends on the number and type of instructions, which may not make efficient use of the CPU as it prevents the CPU from working on other useful processing activity. It is desirable to free the CPU to perform other useful activities while waiting for an amount of time to elapse, rather than spend it in a do-nothing loop.

Microprocessors and microcontrollers usually come with one or more on-chip Timers that can be used in different operating modes. The Z8 Encore! XP Timer operating modes are described in the [Z8 Encore! XP[®] Timer Operation](#) on page 2.

Z8 Encore! XP[®] Timer Features

There are four 16-bit, reloadable Z8 Encore! XP Timers that can be used for timing, event counting, or for generating pulse-width modulated signals.

The features of Z8 Encore! XP Timer's include:

- 16-bit, reload counter
- Programmable prescaler with prescale values from 1 to 128
- PWM output generation

- Capture, compare and capture/compare capability
- External input pin for Timer input, clock gating, or capture signal
- Timer output pin
- Timer interrupt

Figure 1 displays the Z8 Encore! XP Timer architecture.

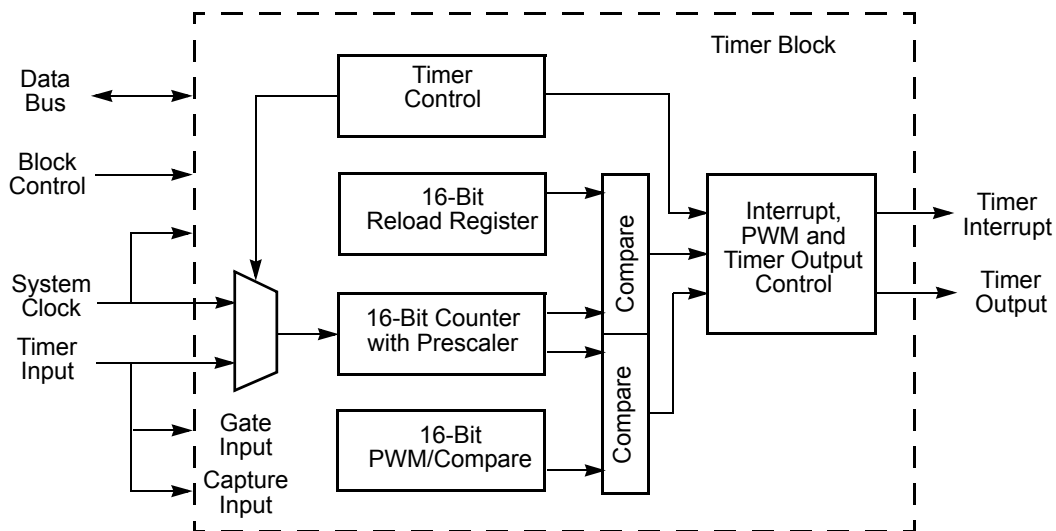


Figure 1. Architecture of the Z8 Encore! XP[®] Timer

Z8 Encore! XP[®] Timer Operation

Z8 Encore! XP Timers are 16-bit, up-counter timers. Table 1 consists of the reload, prescale, and

duration values that can be set for the smallest and largest time-out delays.

Table 1. Timer Values for Type of Time-Out Delays

Type of Time-Out Delay	Timer Reload Values	Prescale Values	Duration = (Prescale Value ÷ Clock Period) * Timer Reload Value
Smallest	0001H	1	Duration = 1 system clock frequency period
Largest	0000H	128	Duration = system clock frequency period * 128 * 65536

► **Note:** When the Timer reaches a value of FFFFH, it rolls over to 0000H and continues counting.

Z8 Encore! XP[®] Timer Modes

Z8 Encore! XP Timers can be configured to operate in the following modes.

CONTINUOUS Mode

In the CONTINUOUS mode, the time-out period is determined by the following equation:

$$\text{CONTINUOUS Mode time-out period(s)} = \frac{\text{Reload Value} \times \text{Prescale}}{\text{System clock frequency(Hz)}}$$

In this mode, the counter resets to 0001 after reaching the reload value and then resumes counting. [Figure 2](#) displays the output and the interrupt generated during CONTINUOUS mode of operation.

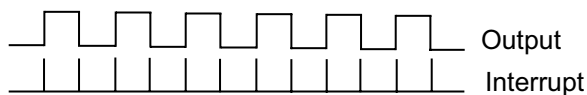


Figure 2. Output Generated in CONTINUOUS Mode

CONTINUOUS mode can generate a square wave that can develop any real time clock application. However, it is not limited to such applications.

ONE-SHOT Mode

In the ONE-SHOT mode, the time-out period is determined by the following equation:

$$\text{ONE-SHOT Mode time-out period(s)} = \frac{(\text{Reload Value} - \text{Start Value}) \times \text{Prescale}}{\text{System clock frequency(Hz)}}$$

In this mode, the reload counter resets to 0001 after reaching the reload value and the Timer is disabled automatically. The ONE-SHOT operating mode can generate a delayed trigger pulse.

COUNTER Mode

In the COUNTER mode, the number of Timer input transitions that occur from start of the Timer is determined by the following equation:

$$\text{COUNTER Mode Timer Input Transitions} = \text{Current Count Value} - \text{Start Value}$$

In this mode, the counter counts the external events up to the reload value, then resets to 0001 and resumes counting. [Figure 3](#) displays the output generated during counter operating mode.

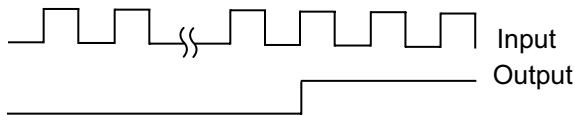


Figure 3. Output Generated in COUNTER Mode

The COUNTER mode counts external pulses/events. The input to the Timer is an external pulse.

PWM Mode

The PWM period is determined by the following equation:

$$\text{PWM period(s)} = \frac{\text{Reload Value} \times \text{Prescale}}{\text{System clock frequency(Hz)}}$$

If TPOL (a control bit that selects the port I/O polarity in the timer control register) is set to 0, the ratio of the PWM Output High Time to the total period is calculated by the following equation:

$$\text{PWM Output High Time Ratio(\%)} = \frac{\text{Reload Value} - \text{PWM Value} \times 100}{\text{Reload Value}}$$

► **Note:** *The PWM value must be less than the reload value.*

If TPOL is set to 1, the ratio of the PWM Output High Time to the total period is calculated by the following equation:

$$\text{PWM Output High Time Ratio(\%)} = \frac{\text{PWM Value} \times 100}{\text{Reload Value}}$$

[Figure 4](#) displays the output generated during PWM mode of operation.

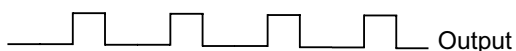


Figure 4. Output Generated In PWM Mode

The PWM mode is widely used in control system applications. For example, in the Servo control system, changing the duty cycle of the PWM pulse can change the speed of the motor.

CAPTURE Mode

In the CAPTURE mode, the elapsed time from Timer Start to Timer Capture events can be calculated by the following equation:

$$\text{Capture Elapsed Time(s)} = \frac{(\text{Capture Value} - \text{Start Value}) \times \text{Prescale}}{\text{System clock frequency(Hz)}}$$

The CAPTURE mode can be used to calculate the time interval between two successive events, which can be used to develop a time interval meter.

COMPARE Mode

In the COMPARE mode, the input signal is continuously compared with a dynamic specified event such that it throws an interrupt whenever the values match. The Timer resumes counting after it is reset.

The Compare time is determined by the following equation:

$$\text{COMPARE Mode Time(s)} = \frac{(\text{Compare Value} - \text{Start Value}) \times \text{Prescale}}{\text{System clock frequency(Hz)}}$$

Figure 5 displays the output and interrupt generated during the COMPARE operating mode.

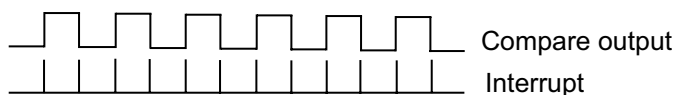


Figure 5. Output Generated In COMPARE Mode



GATED Mode

In the GATED mode, the timer counts only when the timer input signal is in its active state. When the timer reaches the reload value, it resets to 0001H and resumes counting. Also, if the timer output alternate function is enabled, the Timer output pin changes state at timer reset. The input to the Timer in this mode is the gating pulse. This mode can be used for measuring the ON/OFF time, which in turn can be used to measure the frequency of the gated input pulse.

CAPTURE/COMPARE Mode

In the CAPTURE/COMPARE mode, the elapsed time from Timer Start to Timer Capture events can be calculated by the following equation:

$$\text{Capture Elapsed Time(s)} = \frac{(\text{Capture Value} - \text{Start Value}) \times \text{Prescale}}{\text{System clock frequency(Hz)}}$$

Z8 Encore! XP[®] Register Descriptions

The Z8 Encore! XP Timer registers are briefly described in this section. There are four Timer registers.

Timer Control Register (TxCTL)

The Timer Control Register (TxCTL) selects the TIMER mode, sets the prescale value, defines the polarity of input/output pulse and disables/enables the timer. [Table 2](#) displays the Timer Control Register.

Table 2. Timer Control Register (TxCTL)

BITS	7	6	5	4	3	2	1	0
FIELD	TEN	TPOL	PRES			TMODE		
RESET	0	0	0	0	0	0	0	0

TEN—Timer Enable

TPOL—Timer Polarity

PRES—Prescale value

TMODE—TIMER Mode



Timer Byte Register (TxH, TxL)

The Timer Byte Register (TxH, TxL) provides a delay in the first cycle count of the timer after the timer is activated. [Table 3](#) displays the Timer High Byte Register and [Table 4](#) displays the Timer Low Byte Register.

Table 3. Timer High Byte Register (TxH)

BITS	7	6	5	4	3	2	1	0
FIELD	TH							
RESET	0	0	0	0	0	0	0	0

TH—Start Timer High value

Table 4. Timer Low Byte Register (TxL)

BITS	7	6	5	4	3	2	1	0
FIELD	TL							
RESET	0	0	0	0	0	0	0	1

TL—Start Timer Low value

Timer Reload Register (TxRH, TxRL)

The Timer Reload Register (TxRH, TxRL) loads the terminal count of the timer. The pulse period depends on the register value. [Table 5](#) displays the Timer Reload High Byte Register and [Table 6](#) displays the Timer Reload Low Byte Register.

Table 5. Timer Reload High Byte Register (TxRH)

BITS	7	6	5	4	3	2	1	0
FIELD	TRH							
RESET	1	1	1	1	1	1	1	1

TRH—Timer Reload High value

**Table 6. Timer Reload Low Byte Register (TxRL)**

BITS	7	6	5	4	3	2	1	0
FIELD	TRL							
RESET	1	1	1	1	1	1	1	1

TRL—Timer Reload Low value

Timer PWM Byte Register (TxPWMH, TxPWML)

The Timer PWM Byte Register (TxPWMH, TxPWML) modulates the width in a given pulse period. The duty cycle of the modulated pulse is directly proportional to the value of the register. [Table 7](#) displays the Timer PWM High Byte Register and [Table 8](#) displays the Timer PWM Low Byte Register.

Table 7. Timer PWM High Byte Register (TxPWMH)

BITS	7	6	5	4	3	2	1	0
FIELD	PWMH							
RESET	0	0	0	0	0	0	0	0

PWMH—PWM High value

Table 8. Timer PWM Low Byte Register (TxPWML)

BITS	7	6	5	4	3	2	1	0
FIELD	PWML							
RESET	0	0	0	0	0	0	0	0

PWML—PWM Low value



Using the Z8 Encore! XP[®] Timer

The Z8 Encore! XP Timer is configured using the APIs described in this application note. There are ten APIs that perform the following tasks:

1. Initialize the Timer
2. Enable Timer
3. Disable Timer
4. Set Timer control
5. Start the Timer
6. Reload the Timer
7. Set the PWM value for the Timer in PWM mode
8. Set the Timer priority
9. Set the Timer output
10. Reset the Timer

The `init_timer()` API is used to initialize the Z8 Encore! XP Timer in different modes. All the other APIs are called within this API, during initialization. The following examples describes the usage of the `init_timer()` API. For more information on description of this API, see [Appendix B—API Description](#) on page 14.

```
init_timer()

void      init_timer(timer_num,
                    timer_cont, s_high, s_low, r_high,
                    r_low, pm_high, pm_low)
```

Example 1

Consider the example where you need to configure a Timer, TIMER0, in a CONTINUOUS mode to generate 1 ms pulses. Follow the steps below to achieve this configuration:

1. Disable TIMER0
2. Set the mode in CONTINUOUS mode and timer number to TIMER0
3. Set the timer start value to 0x0000

4. Set the timer reload value to 0x4800
5. Set the timer priority
6. Enable TIMER0

All the above tasks is accomplished by calling the `init_timer()` API and entering the required parameters, as displayed below.

```
main()
{
    init_timer(TIMER0, CONTINUOUS_M
              ODE, 0x00, 0x00, 0x48, 0x00, 0x00, 0
              x00);
    /*call the function to display the
    real time clock*/
}
```

Follow the steps below in the interrupt routine to update the real time clock:

```
isr_timer0()
{
    static int sec_timer = 0;
    if(sec_timer++ > 1000)
    /*call a function to increment the
    real time clock*/
}
```

Example 2

Follow the steps below to generate a PWM signal:

1. Disable the TIMER1
2. Set the mode in PWM mode and timer number to TIMER1
3. Set the timer start value to 0x0000
4. Set the timer reload value to 0x4800 (500 Hz)
5. Set the timer PWM value to 0x1200 (25%)
6. Set the timer priority
7. Enable TIMER1

All these operations is accomplished by calling the `init_timer()` API and entering the required parameters as displayed below.

```
main()
{
    init_timer(TIMER1,PWM_MODE,0x0
              0,0x00,0x48,0x00,0x12,0x00);
}
```

Example 3

Follow the steps below to change the duty cycle of the PWM pulse for the timer (Timer#), with the new PWM values as *high value* and *low value*:

1. Disable the timer, Timer#
2. Set the timer PWM values, high value and low value
3. Enable the timer, Timer#

To perform this operation, the following three APIs are called in the sequence given below:

```
{
    disable_timer(Timer#);

    set_timer_pwm(Timer#,high
                 value,low value);

    enable_timer(Timer#);
}
```

Testing the Z8 Encore! XP[®] Timer in Different Modes

The API routines developed for Z8 Encore! XP Timer are tested for different modes. [Figure 6](#) displays the setup to test the Timer functionality in different modes. The function generator generates the required input for the Timer. The Logic Analyzer/CRO captures the timer output.

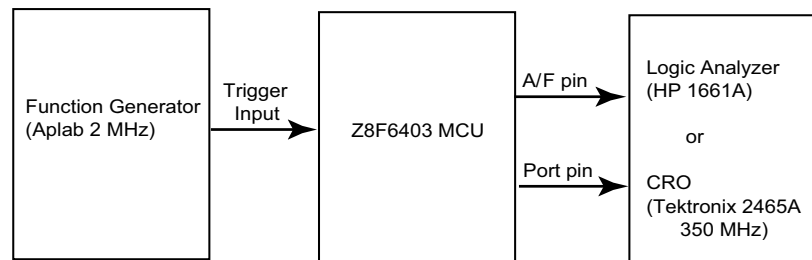


Figure 6. Setup to Test TIMER Mode Functionality

The function generator is used in the modes where external input is required, such as the COUNTER, CAPTURE, and the CAPTURE/COMPARE modes.

For testing the ONE-SHOT mode, one pulse is generated just before enabling the Timer, and the second pulse is generated in the interrupt service routine (ISR) of the Timer. [Figure 7](#) through [Figure 9](#) display the Timer output waveforms, at the appropriate Port pins, for different timer configurations.

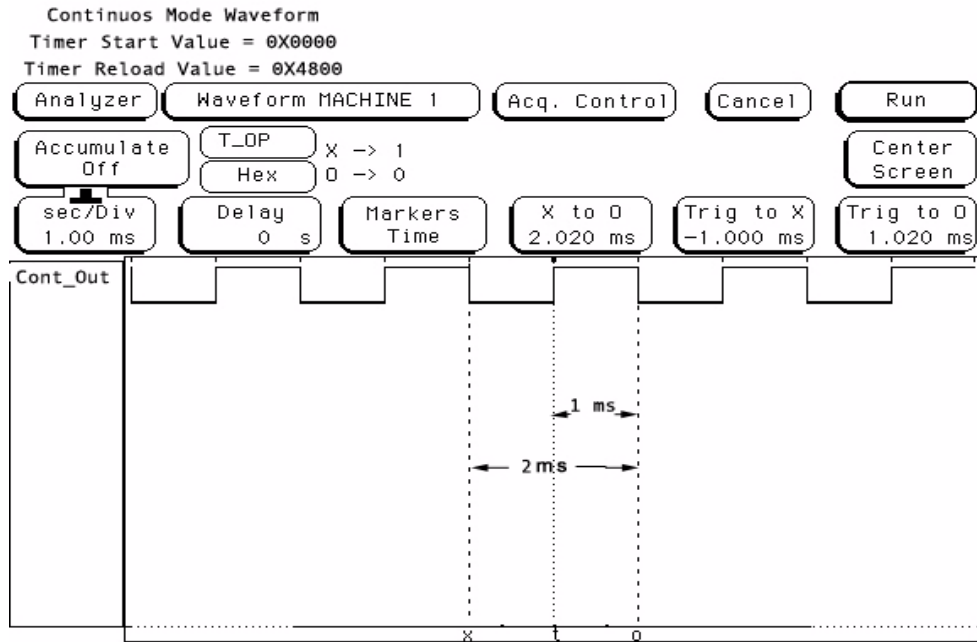


Figure 7. CONTINUOUS Mode Waveform

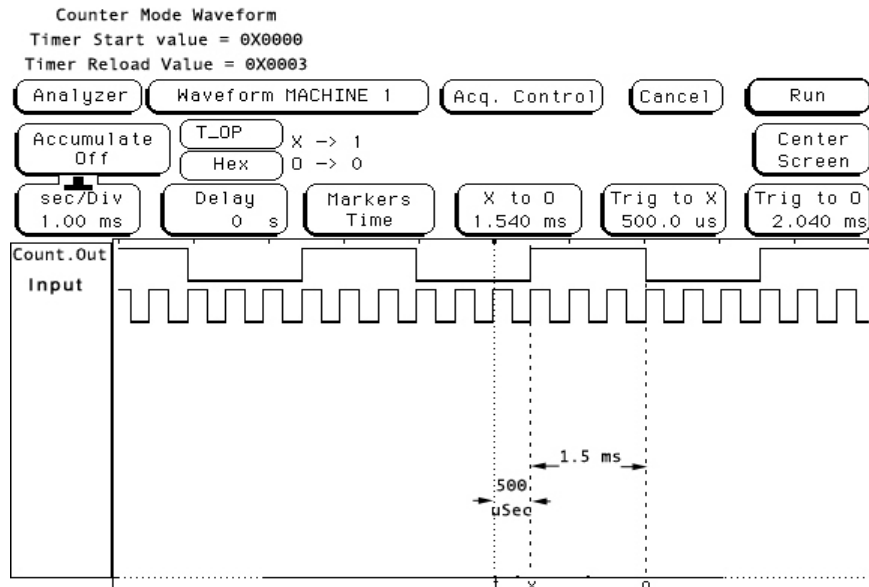


Figure 8. COUNTER Mode Waveform

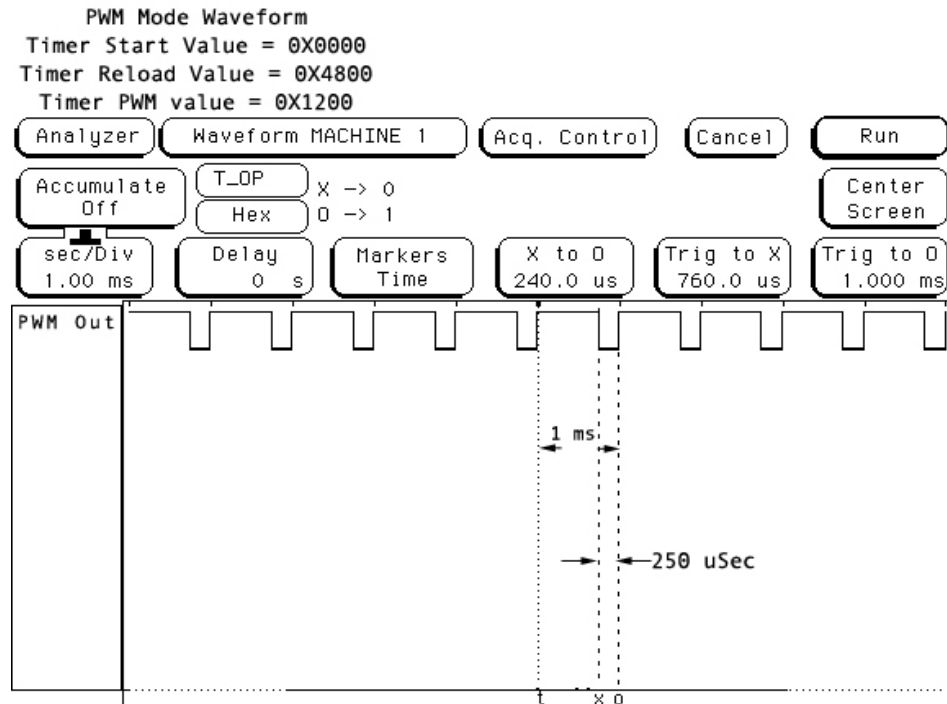


Figure 9. PWM Mode Waveform

Summary

There are different methods to configure the Z8 Encore! XP[®] Timers to operate in different modes, for different applications. This Application Note provides a set of routines that offer a ready-to-use solution to configure the Z8 Encore! XP Timer dynamically for all modes, so that product design time and cost is reduced.

The solution provided in this Application Note can be used for, but is not limited to, developing products like real-time clocks, PWM generation, timing generation, and a host of other timer applications.

References

The documents associated with eZ8 CPU and Z8 Encore! XP available on www.zilog.com are provided below:

- eZ8 CPU Core User Manual (UM0128)
- Z8 Encore![®] Flash Microcontroller Development Kit User Manual (UM0146)
- Z8 Encore! XP[®] 64K Series Flash Microcontrollers Product Specification (PS0199)

Appendix A—Flowcharts

This Appendix displays the flowchart to configure the Z8 Encore! XP[®] Timer (see [Figure 10](#)).

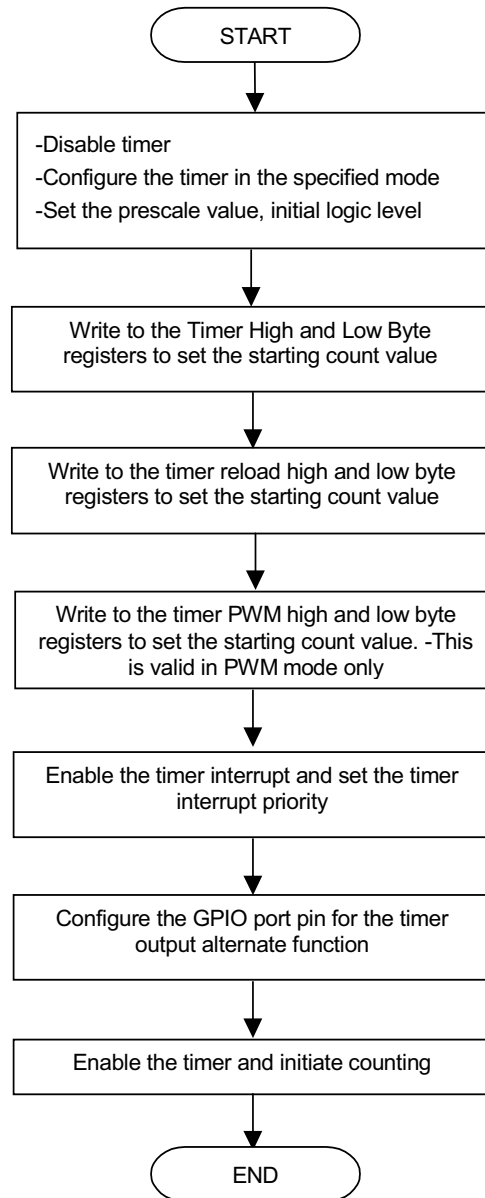


Figure 10. Flowchart to Configure the Timers



Appendix B—API Description

This Appendix contains the detailed description of the `init_timer()` API.

init_timer()

Prototype

```
void init_timer(timer_num, timer_cont, s_high, s_low, r_high,
r_low, pm_high, pm_low);
```

Description

This API initializes all the on-chip timer parameters for required functionality. The `timer_cont` parameter's variables are defined before the function is called. The `timer_cont` parameter is expressed as:

```
unsigned char timer_control = enable_disable|polarity|pres-
cale|timer_mode;
```

Argument (s)

<code>timer_num</code>	'0' for Timer0.
<code>timer_cont</code>	This parameter defines the TIMER mode, disables/enables the timer, sets the prescale value and polarity.
<code>s_high, s_low</code>	Sets the desired timer start value.
<code>r_high, r_low</code> :	Sets the desired timer reload value.
<code>pm_high, pm_low</code>	Sets the desired PWM value. This argument is valid for PWM mode only.

Return Value (s)

Nil

Example 1

To set the Timer for CONTINUOUS Mode

```
init_timer(0, timer_control, 00, 00, 48, 00, 00, 00)
timer_cont = DISABLE_TIMER | POLARITY_HIGH | PRESCL_1 | CONTINU-
OUS;
```

This API invokes Timer0 for CONTINUOUS mode of operation. The output pulse width is 1 ms. `DISABLE_TIMER`, `POLARITY_HIGH`, `PRESCL_1`, and `CONTINUOUS` are # defined in the program.



Example 2

To set the Timer for PWM mode with High Polarity.

```
init_timer(1, timer_control, 00, 00, 48, 00, 12, 00)
timer_control = DISABLE_TIMER | POLARITY_HIGH | PRESC1_1 | PWM;
```

This API invokes Timer1 for PWM mode of operation. The PWM pulse width is 250 μ s. `DISABLE_TIMER`, `POLARITY_HIGH`, `PRESC1_1`, and `PWM` are # defined in the program.

Example 3

To set the Timer for PWM Mode with Low Polarity

```
init_timer(1, timer_control, 00, 00, 48, 00, 12, 00)
timer_control = DISABLE_TIMER | POLARITY_LOW | PRESC1_1 | PWM;
```

This API invokes Timer1 for PWM mode of operation. Polarity Low inverts the out pulse. In this case, the PWM pulse width is 750 μ s. `DISABLE_TIMER`, `POLARITY_LOW`, `PRESC1_1`, and `PWM` are # defined in the program.

Example 4

To set the Timer for SINGLE-SHOT/ONE-SHOT mode

```
init_timer(2, timer_control, 00, 00, 48, 00, 00, 00)
timer_control = DISABLE_TIMER | POLARITY_HIGH | PRESC1_1 |
ONE_SHOT;
```

This API invokes Timer2 for SINGLE-SHOT mode of operation. Interrupts are generated after 1 ms. `DISABLE_TIMER`, `POLARITY_HIGH`, `PRESC1_1`, and `ONE_SHOT` are # defined in the program.



Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

©2007 by Zilog, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

Z8, Z8 Encore!, and Z8 Encore! XP are registered trademarks of Zilog, Inc. All other product or service names are the property of their respective owners.