



Application Note

AN2365

PSoC Express™ Timing Application Examples

Author: Dave Funston

Associated Project: Yes

Associated Part Family: CY8C21xxx, CY8C24x23A, CY8C24794, CY8C27x43, CY8C29x66

PSoC Express Version: 2.0

Associated Application Notes: NA

Abstract

PSoC Express priority encoder transfer function valuator and interface valuator can be combined to implement low-speed timers.

Introduction

PSoC Express makes timing interval generation available in two different ways for low-speed timing. First, the structure of a PSoC Express application has built-in timing so that counting control loop iterations can serve as a way to implement a timing function. Second, PSoC Express contains an interval generator input driver whose pulses can be captured and counted to implement timing functionality. This Application Note shows how PSoC Express applications can use these capabilities in five different timing functions.

Interval Counting and Generation Implementation

The priority encoder transfer function valuator makes an excellent counter. In its simplest form, it increments itself using the main control loop iteration as the timing interval generation mechanism. Each of the examples in this note implements the interval counting function in a priority encoder with a variation of the example code shown in Code 1.

```
If 1 Then Timer+1
```

Code 1.

Code 1 is a simple, self-incrementing counter implemented in a priority encoder valuator named "Timer."

The interval time is the inverse of the Sampling Rate option selected at build time. Selection of Free Run for the Sampling Rate results in an unknown interval time. However, in this case, an interval generator input driver running in Pulse mode can generate the timing interval. In this case, the priority encoder code looks similar to Code 2.

```
If (Interval==Interval_Triggered) Then Timer+1
```

Code 2.

Code 2 is a simple, self-incrementing counter implemented in a priority encoder valuator named "Timer" that uses an interval generator named "Interval" as input.

1. One-Time, Power-Up Delay Timer

The power-up delay timer takes advantage of the fact that PSoC Express initializes all valuator values to 0 upon power-up. Upon power-up, the priority encoder (Timer) self-increments until its value is equal to the value of an interface valuator (DelayTime) that specifies the delay length in intervals. The delay length must be set as desired at design time. In order to implement a delay, other design elements are coded to wait until $Timer \geq DelayTime$ before doing something.

```
If Timer<DelayTime Then Timer+1
```

Code 3.

Code 3 is priority encoder code for a power-up delay timer using loop iterations for interval generation. Refer to *startup_delay_timer.cmx*.

For the interval generator sourced timer, pulses from an interval generator (PulseGenerator) running in Pulse mode are counted.

```
If (PulseGenerator==PulseGenerator__Triggered)
&& (Timer<DelayTime) Then Timer+1
```

Code 4.

Code 4 is priority encoder code for a power-up delay timer using an interval generator driver (PulseGenerator) for interval generation. Refer to *startup_pg_delay_timer.cmx*.

2. One-Time, Event-Triggered Delay Timer

The power-up delay timer can be altered to start when an event occurs by adding a valuator (TimerTriggered) that sets itself to 1 upon the occurrence of the event. The TimerTriggered transfer function type can be selected from the available types as long as it is able to latch in a value upon occurrence of the event that it is monitoring. For this example, TimerTriggered is a priority encoder.

The Timer monitors the value of TimerTriggered, so the interval counting mechanism increments after the event occurs rather than upon power-up. A second difference from the power-up delay timer is that DelayTime can be configured prior to triggering through the use of an interface driver.

```
If (TimerTriggered==1) && (Timer<DelayTime)
Then Timer+1
```

Code 5.

Code 5 is priority encoder code for an event-triggered delay timer using loop iterations for interval generation. Refer to *event_triggered_delay_timer.cmx*.

```
If (TimerTriggered==1) &&
(PulseGenerator==PulseGenerator__Triggered) &&
(Timer<DelayTime) Then Timer+1
```

Code 6.

Code 6 is priority encoder code for an event-triggered delay timer using an interval generator driver (PulseGenerator) for interval generation. Refer to *event_triggered_pg_delay_timer.cmx*.

3. Resettable, Event-Triggered Delay Timer

In order to change the event-triggered timer from one-time use to multiple uses, a mechanism for resetting the Timer and the triggering valuator is needed. Adding code to the TimerTriggered valuator to clear itself upon the receipt of a reset event partially accomplishes this goal. Adding code to the Timer to clear itself when the TimerTriggered clears completes this task.

```
If TimerTriggered==0 Then 0
ElseIf (TimerTriggered==1) && (Timer<DelayTime)
Then Timer+1
```

Code 7.

Code 7 is priority encoder code for a resettable event-triggered delay timer using loop iterations for interval generation. Refer to *resettable_delay_timer.cmx*.

```
If TimerTriggered==0 Then 0
ElseIf (TimerTriggered==1) &&
(PulseGenerator==PulseGenerator__Triggered) &&
(Timer<DelayTime) Then Timer+1
```

Code 8.

Code 8 is priority encoder code for a resettable event-triggered delay timer using an interval generator driver (PulseGenerator) for interval generation. Refer to *resettable_pg_delay_timer.cmx*.

4. Configurable Cycling Timer

Like the power-up delay timer, the cycling timer starts incrementing upon power-up. Unlike the power-up delay timer, which stops incrementing when it reaches its limit, the cycling timer automatically resets itself when it reaches its limit and then continues incrementing. An interface valuator (CycleTime) holds the cycle time. At any time, CycleTime can be reconfigured through an interface driver.

```
If Timer<CycleTime Then Timer+1
ElseIf 1 Then 0
```

Code 9.

Code 9 is priority encoder code for the cycling timer using loop iterations for interval generation. Refer to *cycling_timer.cmx*.

```
If (PulseGenerator==PulseGenerator__Triggered)
&& (Timer<CycleTime) Then Timer+1
ElseIf
(PulseGenerator==PulseGenerator__Triggered)
Then 0
```

Code 10.

Code 10 is priority encoder code for the cycling timer using an interval generator driver (PulseGenerator) for interval generation. Refer to *cycling_pg_timer.cmx*.

5. Resettable, Event Duration Timer

Duration between two events can be measured using the same techniques. A valuator (TimerRunning) sets itself to 1 when the first event (Start) occurs, and clears itself when the second event (Stop) occurs. The Timer increments when TimerRunning equals 1. A third event (Reset) clears TimerRunning and Timer in order to make the duration timer resettable.

```
If Reset==Reset__High Then 0
ElseIf (TimerRunning==1) && (Timer>32767) Then
Timer+1
```

Code 11.

Code 11 is priority encoder code for the event duration timer using loop iterations for interval generation. Refer to *resettable_event_timer.cmx*.

```
If Reset==Reset__High Then 0
ElseIf (TimerRunning==1) &&
(PulseGenerator==PulseGenerator__Triggered) &&
(Timer>32767) Then Timer+1
```

Code 12.

Code 12 is priority encoder code for the event duration timer using an interval generator driver (PulseGenerator) for interval generation. Refer to *resettable_pg_event_timer.cmx*.

Conclusion

The priority encoder transfer function and interface valuator are useful for implementing timing functions.

About the Author

Name: Dave Funston

Title: Applications Engineer, Staff

Background: Dave Funston supports PSoC Express at Cypress Semiconductor in Lynnwood, WA. He holds a Master's degree in Mechanical Engineering from the University of California at Davis, and has programmed microcontrollers since 1996.

Contact: dfu@cypress.com

Cypress Semiconductor
2700 162nd Street SW, Building D
Lynnwood, WA 98087
Phone: 800.669.0557
Fax: 425.787.4641

<http://www.cypress.com/>

Copyright © 2006 Cypress Semiconductor Corporation. All rights reserved.

PSoC is a registered trademark of Cypress Semiconductor Corp.

"Programmable System-on-Chip," PSoC Designer and PSoC Express are trademarks of Cypress Semiconductor Corp.

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

The information contained herein is subject to change without notice. Made in the U.S.A.