

Implementing Hardware Quadrature Phase Decoders

Author: Edwin Olson

Associated Project: Yes

Associated Part Family: CY8C22xxx, CY8C24xxx, CY8C25xxx, CY8C26xxx, CY8C27xxx

PSoC Designer Version: 4.00

Summary

The PSoC family lacks a user module for decoding quadrature phase signals. However, quadrature phase decoders *can* be implemented on PSoCs by using an unusual arrangement of other user modules. Each decoder uses two digital blocks plus an inverter (or a third digital block on CY8C26xxx parts), is quarter-resolution, and can handle very fast counting rates. Two complete decoders can be implemented in a single device with room to spare.

Introduction

Quadrature phase encoders can be found on many actuators such as DC gearhead motors. They allow a control system to monitor the precise angular position of the output shaft. This, in turn, allows control of either position or velocity.

Optical encoders are typically constructed from a perforated disk and an optical break-beam sensor. The disk rotates with the motor while the break-beam sensor is fixed. As the disk rotates, perforations in the disk pass through the break-beam sensor creating a square wave.

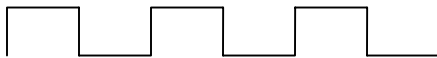


Figure 1. A single channel optical encoder output. A logical "high" indicates that the break-beam sensor is positioned over a perforation in the disk.

This configuration is known as a "single channel" optical encoder and can provide position and velocity information for unidirectional motors. In the case of reversible motors, an ambiguity arises: whether the motor rotates clockwise or counterclockwise, the square wave appears the same. Even worse, if the motor is not rotating, a perforation might be *almost* under the sensor. Environmental vibration may cause the break-beam sensor to trigger and untrigger randomly. The motor would appear to be moving while it is essentially stationary.

Quadrature phase encoders solve the problems of single channel encoders by using two break-beam sensors. The two sensors are positioned 90 degrees out-of-phase, which results in a "Gray" code output. A Gray code is a sequence where only one bit changes on each count; this is essential to avoid glitches.

The two sensors are usually designated A and B. When the motor rotates in one direction, the waveform of A will "lead" signal B -- its rising edges occur first. When the motor rotates in the other direction, the waveform of signal B will lead signal A. This difference is used to determine the direction of rotation and to eliminate the environmental noise problem -- environmental noise will appear as alternating and canceling movement forward and backward.

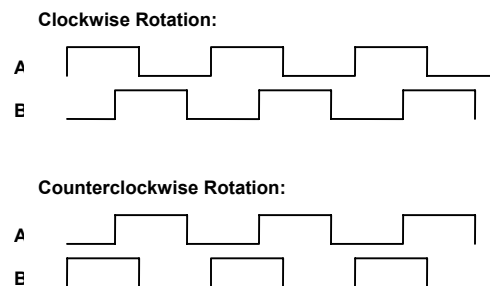
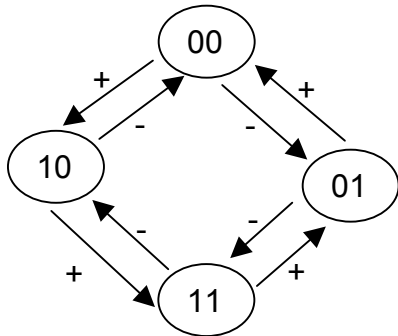
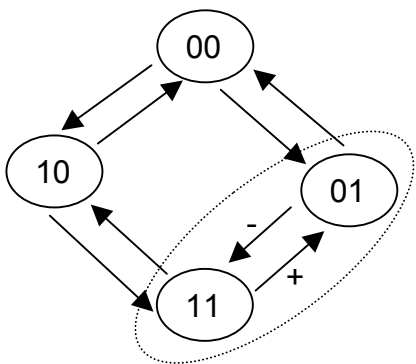


Figure 2. Output of a quadrature phase encoder rotating in both directions.

Quadrature phase signals are typically decoded with a state machine and an up/down counter. A conventional decoder has four states, corresponding to all possible values of the A and B inputs. The state transition diagram is shown below (same-state transitions are not depicted). State transitions marked with a “+” and “-“ indicate increment and decrement operations on the quadrature phase counter.



For each full cycle of the quadrature phase signal, the quadrature phase counter changes by four ticks. Lower resolution counters can also be used by implementing up/down operations on only a subset of the state transitions. A quarter-resolution decoder is shown below.



Decoding on PSoCs

The PSoC hardware supports neither state machines nor up/down counters. There simply isn't enough flexibility to implement a full resolution encoder in hardware. However, implementing a quarter-resolution encoder is possible. One concession is immediately necessary: there are no up/down counters on the PSoC, so we will need to use *two* counters, one which will count “up” events, and the other which will count “down” events. We then simply subtract the “down” counter from the “up” counter to compute the effective value of the up/down counter.

The problem is now reduced to controlling two counters: what should the *clock* and *enable* signals for each counter be?

The transition 01->11 must cause a count in one counter. This is easily done by connecting the A input to the clock and the B input to an enable. The transition 11->01 must cause a count on the other counter. If we invert the A signal, we can use it for the clock and B for the enable, just like before. See the circuit diagram below.

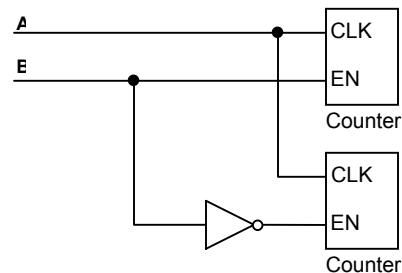


Figure 3. Conceptual schematic of quadrature phase decoder.

Note that we need an inverter to create $\sim A$. On old parts, such as CY8C26xxx, we can use a DigInV User Module, or provide an external inverter. On new parts, including CY8C27xxx, we have the additional option of using the programmable LUTs to create our inverters -- an appealing option since it does not consume additional digital blocks. To use the LUTs as an inverter, the A signal must be routed from an input row through the LUT (which computes $\sim A$), back to the input columns, and finally back to an input row. See the diagram below for an example; it shows two quadrature phase decoders wired up simultaneously. Note that the two counters from each decoder are on different rows in order to accommodate the routing of the clock signal through an inverter.

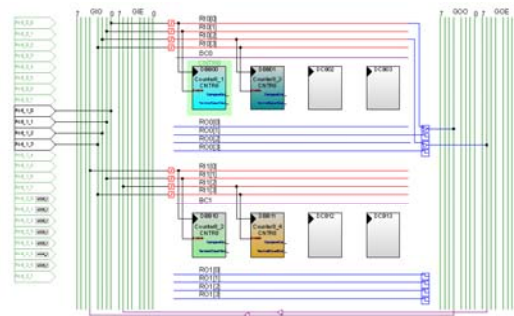


Figure 4. Interconnect view for two decoders.

Whenever we feed unsynchronized inputs into a digital circuit, we must consider the possibility of metastability. However, these problems are easily dealt with on PSoC by the built-in synchronizing circuits. On CY8C26xxx parts, these synchronizing circuits are always enabled, but on CY8C27xxx parts, these will need to be manually enabled.

Design Details

Reading the value of an 8-bit down counter using the provided API can cause ticks to be lost. The counter value cannot be read without overwriting the Compare register. To work around this, the API temporarily disables the counter, makes a copy of the Compare register, performs the counter read, and then restores the value of the Compare register. During the time that the counter is disabled, however, we could miss ticks from the encoder, particularly if a time-consuming interrupt occurs.

However, in this application, we do not need the Compare register and it is okay if its value is overwritten. Thus, it is not necessary for us to disable the timer, and we can reduce the probability of missing an encoder tick if we use the following code instead of the high-level API:

`_ReadDirectCounter1:`

```
M8C_SetBank0
mov  A, reg[Counter8_1_COUNTER_REG]
mov  A, reg[Counter8_1_COMPARE_REG]
ret
```

To read the value of the quadrature phase counter, use code such as:

```
int g_quadPhaseValue=0;
int g_quadPhaseLastUp=0;
int g_quadPhaseLastDown=0;

int UpdateQuadPhase()
{
    int newUp, newDown;
    int deltaUp, deltaDown;

    newUp=255-ReadDirectCounter1();
    newDown=255-ReadDirectCounter2();

    deltaUp=newUp-g_quadPhaseLastUp;
    if (deltaUp<0)
        deltaUp+=256;

    deltaDown=newDown-g_quadPhaseLastDown;
    if (deltaDown<0)
        deltaDown+=256;

    g_quadPhaseValue+=(deltaUp-deltaDown);
    g_quadPhaseLastUp=newUp;
    g_quadPhaseLastDown=newDown;

    return g_quadPhaseValue;
}
```

Several global variables are used to maintain state. We need to keep the last read values of the up and down counters so that we can determine the change in position. We also maintain the position of the encoder. Note that when the counters are read, we convert them into positive ticks by subtracting the counter value from 255. The code also handles the fact that the 8-bit counters wrap every 256 counts.

In order for this code to work correctly, `UpdateQuadPhase` must be called before 256 counts of the encoders occur or ticks will be lost. Assuming a counting rate of around 100 kHz (which represents a very fast quadrature phase signal), this means that `UpdateQuadPhase` must be called at least every 2.5 ms. Alternatively, the terminal count interrupt could be used to simulate a wider counter, or a 16-bit counter could be used which would decrease the rate at which updates would need to occur. In many systems, quadrature phase signals are closer to 1 kHz, which allows 250 ms between updates.

The ability to do quadrature phase decoding with PSoC hardware has the potential of reducing the component count of many systems that would otherwise require a CPLD or other dedicated parts.

This technique has been tested and a project file is included. The project outputs the current value of the encoder to an LCD screen using the LCD User Module. A prototype system can be wired up in a few minutes.

This is a revision of the original note. In the original version, the inverter appeared on the wrong input signal (on the enable, rather than the clock). I apologize for the error.

About the Author

Name: Edwin Olson
Title: Research Assistant, MIT
Background: Edwin is a PhD student at MIT studying robotics, vision, and state estimation. He also teaches a course on autonomous robotics.
Contact: eolson@mit.edu
<http://www.blisstonia.com/eolson>

Cypress MicroSystems, Inc.
2700 162nd Street SW, Building D
Lynnwood, WA 98037
Phone: 800.669.0557
Fax: 425.787.4641

<http://www.cypress.com/> / http://www.cypress.com/aboutus/sales_locations.cfm / support@cypressmicro.com

Copyright © 2003 Cypress MicroSystems, Inc. All rights reserved.

PSoC™ (Programmable System-on-Chip™) is a trademark of Cypress MicroSystems, Inc.

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

The information contained herein is subject to change without notice.