AN101

v1.1

Setting outputs at position

Copley servo drives support several different methods of triggering general purpose output pins based on the motor position. This feature allows an output pin to accurately trigger an external event when the motor moves past a programmed set point, or falls within a window of positions.

These output pin configurations can be broadly divided into two categories; software based triggers and hardware based triggers. Software triggers are managed by the drive firmware and updated at the position loop rate (3 or 4 kHz depending on the drive). For these output configurations, the delay between the motor passing the specified position and the output pin being updated is on the order of several hundred microseconds.

Hardware based triggers are managed by the FPGA on select Copley drives and have a much lower latency then software triggers. The delay between passing a position and setting an output for these types of triggers is dependent on the type of encoder used. For quadrature encoders, the delay is on the order of a few tens of nanoseconds.

Software triggered outputs

All Copley servo drives starting with feature set B support a number of position based output configuration which are managed by the drive firmware. At the position loop update rate, the drive firmware compares the most recent encoder reading to the programmed position set points and either activates or deactivates the output pin based on the results of the comparison.

Each general purpose output pin has an associated drive parameter which is used to configure it. These parameter are documented in the parameter dictionary document. The values set to these parameters consist of a 16-bit configuration code, and one or two additional 32-bit values, the meaning of which depends on the output configuration.

The 16-bit configuration code for these output pin settings is bit mapped as follows:

Bits	Description
0-7	Specify the output function used (see below).
8	If set, the output level is inverted from it's normal state.
9-11, 15	Reserved. Must be set to zero.
12-13	Specify the axis number on multi-axis drives (0 for first axis, 1 for second axis, etc).
14	If set, use the commanded position for output comparison. If clear, use the actual position read from the encoder

Function 4 – Set output while within position window:

For this output function, the two 32-bit values programed with the output pin configuration give the bounds of a position window in which the output pin is active. Each servo cycle the firmware compares the current motor position to these boundary points and sets the output if the motor is within the window. The first position specified should give the lower boundary of the window, and the second position should give the upper boundary.

For example; to configure output 3 (drive parameter 0x72) to go active between actual motor position 10000 and position 20000, the following ASCII command would be used:

```
s r0x72 4 10000 20000
```

Function 5 – Pulse output on low->high position crossing.

This output pin function causes the output to go active for a programmable number of milliseconds when the motor position crosses from below to above a specified position. The first 32-bit value gives the position to compare against. The second 32-bit value gives the length of the output pin pulse in milliseconds.

For example; to configure output 1 (drive parameter 0x70) to pulse for 100ms when the motor position crosses from less then 123 to greater then 123, the following ASCII command would be used:

```
s r0x70 5 123 100
```

Function 6 – Pulse output on high->low position crossing.

This output pin function works the same as function 5, except that the output is pulsed active when the motor position crosses the target value moving from above to below the target.

Function 7 – Pulse output on position crossing in either direction.

This output pin function works the same as functions 5 and 6. The output is pulsed on any crossing of the target position.

Function 9 – Pulse output when crossing a series of positions.

To use this output pin function, a section of the drive's trace memory must first be reserved and loaded with a series of positions sorted from lowest to highest. The output pin can then be configured to pulse for a programmable number of milliseconds each time one of the positions is crossed in either direction.

Loading positions into trace memory can not be done over the ASCII interface. The binary serial protocol supports this as does the CANopen interface. To allocate and load trace memory over CANopen, see objects 0x250A – 0x250C in the CANopen programmer's guide.

When the output pin is configured in this mode, the upper half of the first 32-bit word written to the output pin configuration parameter stores the offset (in 16-bit word units) of the first position stored in trace memory. The lower half of the first 32-bit value gives the total number of positions stored in trace memory. The second 32-bit value gives the pulse duration in milliseconds.

For example, to configure output 2 to generate a 10ms pulse at positions 1000, 3000, 5000, and 6000 via the CANopen or EtherCAT networks, the following steps would be taken:

1. Reserve space in the trace buffer which can be used to hold the positions. Each position requires two 16-bit words of trace buffer memory, so we would need at least 8 words to hold the 4 positions used in this example. If a larger number of positions are required, more space should be reserved. Reserve the space in the trace buffer by writing the number of 16-bit words needed to CANopen object 0x250A.
2. Now, the position data needs to be uploaded into the reserved area of the trace buffer. This is done by first writing 0 to object 0x250B. 0x250B holds the offset into the reserved portion of the trace memory where we want to write data. We're writing to the beginning of the reserved area, so set this object to zero. Then write the position data to object 0x250C. Values written to object 0x250C are copied into the trace memory at the offset location set using object 0x250B. This offset is automatically incremented as data is written to the buffer, so the positions can be written either using a single 16-byte write to object 0x250C, or as a sequence of 4 4-bit writes.
3. Configure the output pin function. For output 2 this would be done by writing to object 0x2193 sub-index 2. The configuration value consists of a total of 10 bytes. The first 2 bytes give the basic configuration function number which is 9 in this case. The next 2 bytes give the number of positions stored in our buffer (4 in this example). The next 2 bytes give the offset into the reserved area of trace memory where the positions start (0 in this example). The next 2 bytes give the pulse duration in milliseconds (10 in this example), and the final two bytes are reserved and should be set to zero.
4. As the motor moves, it should set output 2 for 10ms each time it passes one of the configured points. The points can be changed at any time by uploading a new set using step 2 above. There is no need to reallocate the trace memory or reconfigure the output pin.

Hardware triggered outputs

Starting with firmware version 1.40, Copley drives in the plus family (feature set E) have some additional circuitry which allows an output to be set on position with much lower latency then the software based functions described above.

Single axis drives (such as the AEM) contain one such compare output module. Dual axis drives (such as the AE2) contain two compare modules; one per axis.

Output pins can be assigned to follow the hardware compare output by setting the output pin configuration code to 16. For example, to configure output 2 (parameter 0x71) as a compare output, the following ASCII command would be used:

```
s r0x71 16
```

The compare hardware itself is configured through the use of several additional drive parameters:

Parameter	CANopen object	Description
0x185	0x2160	Compare module configuration. This parameter is bit-mapped as follows: **Bits** **Description** 0 Set to enable module 1 Set to invert active state of output 2 If set, toggle output on compare match. If clear, pulse output for programmable time. 3-4 Define mode of compare module. See below. 5-31 Reserved for future use. Should be set to zero.
0x186	0x2161	Compare module status register. This parameter is bit-mapped as follows: **Bit** **Description** 0 Current value of compare output (read only). 1 Set when position matches compare register 0. Write 1 to clear. 2 Set when position matches compare register 1. Write 1 to clear. 3-31 Reserved.
0x187	0x2162	Compare value 0.
0x188	0x2163	Compare value 1.
0x189	0x2164	Compare increment. Signed 32-bit value used to update compare values in some modes.
0x18A	0x2165	Compare pulse period. The lower 20-bits of this parameter give the period of the compare output pulse in 10ns units.

The behavior of the compare circuit is controlled by the mode set it's configuration register:

Mode 0 – Dual compare mode.

In this mode the output can be pulsed or toggled at a fixed number of encoder counts in either direction. The compare module constantly tests the motor position against the value of both compare values. If the position crosses either value moving in the positive direction, then the other compare value is increased by the increment. If the position crosses either compare value going in the negative direction, then the other compare value is decreased by the increment. At any compare value crossing, the output is pulsed or toggled.

To use this compare mode, value 0 and value 1 should be initialized so that they flank the current motor position, and the increment should be initialized to 2 times the number of counts between desired pulses.

For example, if the goal is to pulse the compare output every 100 encoder counts in either direction, and the current motor position is 1234 encoder counts, then the compare parameters should be configured as follows:

value 0:	1200
value 1:	1300
increment:	200
config:	1

If the position decreases to 1199 counts, then the value 1 parameter will automatically be decreased by the increment to 1100. Value 0 is not changed in this case, so the new compare values continue to flank the position (value 1 = 1100, position = 1199, value 0 = 1200).

Mode 1 – Single compare mode.

This mode can be used to output a sequence of pulses every N encoder counts in a single direction. When the compare module is configured in this mode it will constantly compare the motor position with the value programed in value 0. At every crossing, the increment will be added to the value 0 register.

For example, if the current motor position were 123 counts, and the goal was to pulse the output every 100 counts starting with position 5000, then the following settings would be used:

value 0:	5000
increment:	100
config:	9

This mode can also be used to generate a single pulse at a fixed position when ever that position is crossed in either direction. In this case the value 0 register is configured with the trigger position, and the increment is set to zero.

Mode 2 – Set output while in position window.

In this mode the compare output is active when ever the motor position is greater then value 0, and less then or equal to value 1. The increment setting is not used.

Mode 3 – Single compare mode with end position.

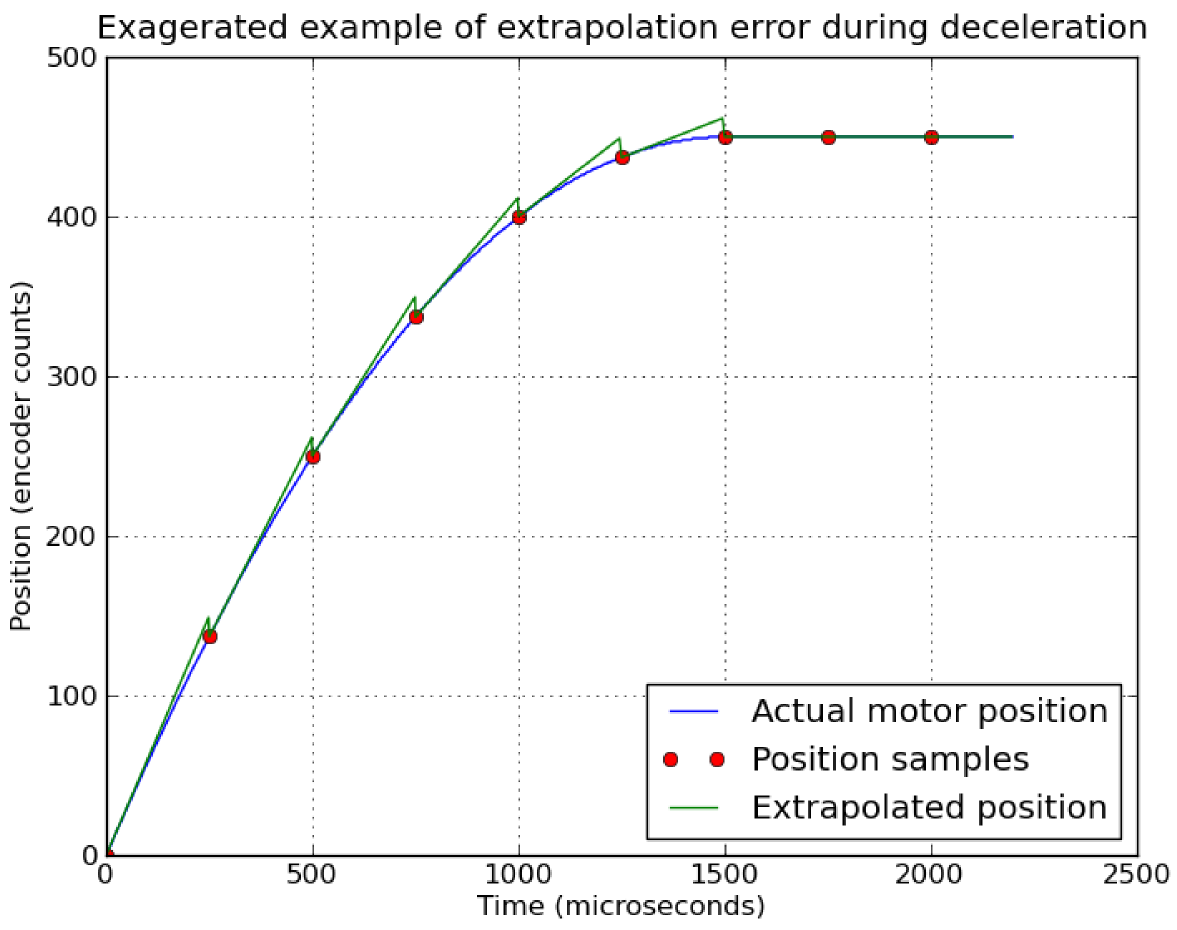
This mode behaves like mode 1 except that the compare module will be automatically disabled if the position stored in value 1 is crossed. This allows a sequence of pulses to be output starting with the value initially programmed in value 0, and ending when the position programmed in value 1 is reached.

Position extrapolation.

The compare module can be used with any type of encoder supported by the drive. If the encoder is a quadrature encoder connected to the primary quadrature input connection of the drive, then the encoder position will be directly used by the hardware to perform the comparison operations. This is the most accurate form of compare supported by the drive.

For other encoder types, or for quadrature encoders connected to the drive's multi-mode port, the drive firmware samples the encoder position at the position loop update rate. The calculated position and velocity are then used to extrapolate the position over the coarse of the next loop period. During this time the compare module works with the extrapolated position data for it's comparison operations. The result is still a very accurate compare output as long as the motor is running at a constant velocity. There is however the possibility of a compare output firing prematurely, or multiple times in dual compare mode if the drive is accelerating during the compare event.

For example, the figure below shows the difference between the actual motor position and the extrapolated position when the motor is decelerating. The blue line shows the actual motor position. The red dots show the position samples made by the drive every 250 microsecond loop update. The green line shows the results of extrapolating these positions for one position loop cycle. The difference between the green and blue lines show the error caused by this extrapolation.



The compare hardware in the drive will use the extrapolated positions to trigger it's output. In this example, the motor came to rest at a final position of 450 encoder counts. The extrapolated positions however ranged up to 461 counts. Had a compare target of 460 been set, then it would have been triggered by the compare hardware even though the actual motor position had never reached that point. In Dual compare mode, a trigger would have occurred when the extrapolated position exceeded 460 counts, and then another trigger would have occurred when the position was updated to it's correct value of 450 counts.

It should be noted that in this example the motor is moving at a very high velocity at time zero (600,000 encoder counts/second), and is decelerating at a very high rate (400,000,000 counts/second/second). In a more realistic application, the error caused by extrapolating motor position over a 250 microsecond period would be much smaller.

There are several ways to work around these potential extrapolation errors. In single compare mode the hardware will only trigger a compare event when moving in one direction. In this case the compare output may be triggered slightly early or late due to extrapolation errors, but won't be triggered multiple times. Running the motor at a constant velocity over the range of positions for which the compare outputs are required will also minimize the error caused by extrapolation. The size of the extrapolation error is directly proportional to the motor acceleration.

Finally, if no extrapolation errors can be tolerated, then a quadrature encoder is the preferred position sensing device. Quadrature encoders can be continuously sampled by the compare hardware and therefore do not use extrapolation. In this case, no error caused by extrapolation is possible.

Revision History

Date	Version	Revision
3/13/2012	1.0	Initial release
10/24/2014	1.1	Added additional information on output type 9.