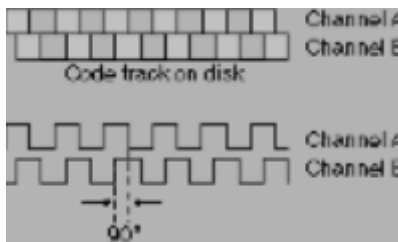


Encoders - Measuring rotation of a wheel or other shaft

Encoders are devices for measuring the rotation of a spinning shaft. Encoders are typically used to measure the distance a wheel has turned which can be translated into the distance the robot has traveled. The distance traveled over a measured period of time represents the speed of the robot, and is another common use for encoders. Encoders can also directly measure the rate of rotation by determining the time between pulses. This article covers the use of quadrature encoders (defined below) For non-quadrature incremental encoders, see the article on counters. For absolute encoders the appropriate article will depend on the input type (most commonly [analog](#), I2C or SPI).

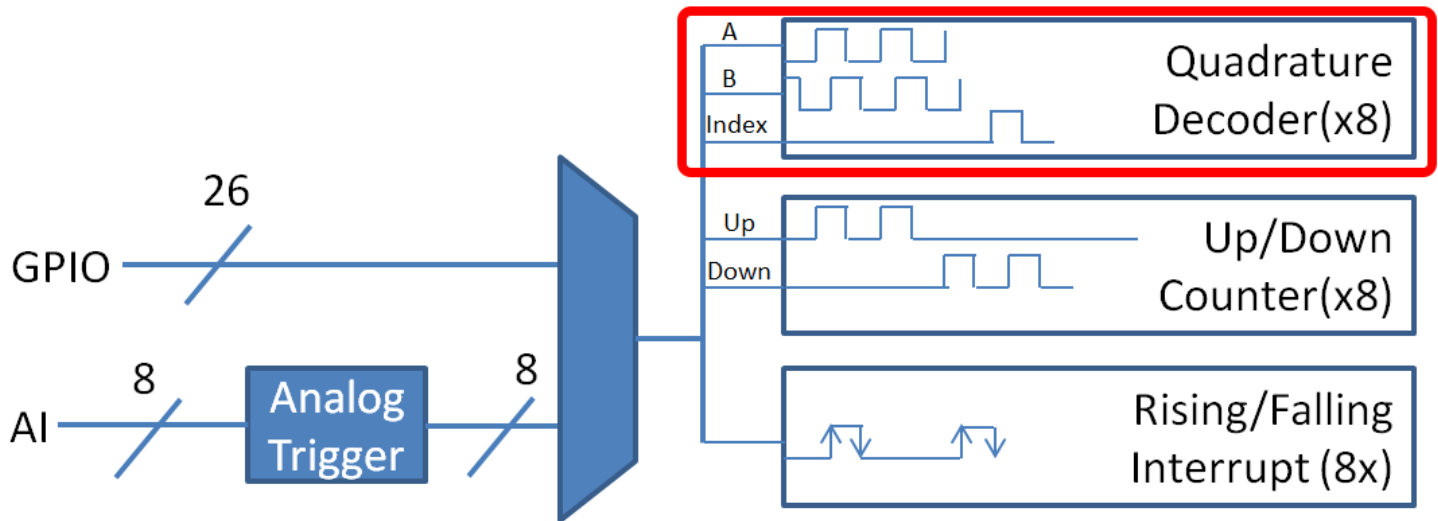
Quadrature Encoder Overview



A quadrature encoder is a device for measuring shaft rotation that consists of two sensing elements 90 degrees out of phase. The most common type of encoder typically used in FRC is an optical encoder which uses one or more light sources (LEDs) pointed at a striped or slit code wheel and two detectors 90 degrees apart (these may be located opposite the LED to detect transmission or on the same side as the LED to measure reflection). The phase difference between the signals can be used to detect the direction of rotation by determining which signal is "leading" the other.

Encoders - Measuring rotation of a wheel or other shaft

Encoders vs. Counters



The FRC FPGA has 8 Quadrature decoder modules which can do 4x decoding of a 2 channel quadrature encoder signal. This means that the module is counting both the rising and falling edges of each pulse on each of the two channels to yield 4 ticks for every stripe on the codewheel. The quadrature decoder module is also capable of handling an index channel which is a feature on some encoders that outputs one pulse per revolution. The counter FPGA modules are used for 1x or 2x decoding where the rising or rising and falling edges of one channel are counted and the second channel is used to determine direction. In either case it is recommended to use the Encoder class for all quadrature encoders, the class will assign the appropriate FPGA module based on the encoding type you choose.

Sampling Modes

The encoder class has 3 sampling modes: 1x, 2x and 4x. The 1x and 2x mode count the rising or the rising and falling edges respectively on a single channel and use the B channel to determine direction only. The 4x mode counts all 4 edges on both channels. This means that the 4x mode will have a higher positional accuracy (4 times as many ticks per rotation as 1x) but will also have more jitter in the rate output due to mechanical deficiencies (imperfect phase difference, imperfect striping) as well as running into the timing limits of the FPGA. For sensing rate, particularly at high RPM, using 1x or 2x decoding and increasing the number of samples to average may substantially help reduce jitter. Also keep in mind that the FPGA has 8 quadrature decoding modules (used for 4x decoding) and 8 counter modules (used for 1x and 2x decoding as well as Counter objects).

Encoders - Measuring rotation of a wheel or other shaft

Constructing an Encoder object

C++

```
Encoder *enc;  
enc = new Encoder(0, 1, false, Encoder::EncodingType::k4X);
```

Java

```
Encoder enc;  
enc = new Encoder(0, 1, false, Encoder.EncodingType.k4X);
```

There are a number of constructors you may use to construct encoders, but the most common is shown above. In the example, 0 and 1 are the port numbers for the two digital inputs and false tells the encoder to not invert the counting direction. The sensed direction could depend on how the encoder is mounted relative to the shaft being measured. The k4X makes sure that an encoder module from the FPGA is used and 4X accuracy is obtained.

Setting Encoder Parameters

C++

```
Encoder *sampleEncoder = new Encoder(0, 1, false, Encoder::EncodingType::k4X);  
sampleEncoder->SetMaxPeriod(.1);  
sampleEncoder->SetMinRate(10);  
sampleEncoder->SetDistancePerPulse(5);  
sampleEncoder->SetReverseDirection(true);  
sampleEncoder->SetSamplesToAverage(7);
```

Java

```
Encoder sampleEncoder = new Encoder(0, 1, false, Encoder.EncodingType.k4X);  
sampleEncoder.setMaxPeriod(.1);  
sampleEncoder.setMinRate(10);  
sampleEncoder.setDistancePerPulse(5);  
sampleEncoder.setReverseDirection(true);  
sampleEncoder.setSamplesToAverage(7);
```

The following parameters of the encoder class may be set through the code:

- Max Period - The maximum period (in seconds) where the device is still considered moving. This value is used to determine the state of the `getStopped()` method and effect the output of the `getPeriod()` and `getRate()` methods. This is the time between pulses on an individual channel (scale factor is accounted for). It is recommended to use the Min Rate parameter

Encoders - Measuring rotation of a wheel or other shaft

instead as it accounts for the distance per pulse, allowing you to set the rate in engineering units.

- Min Rate - Sets the minimum rate before the device is considered stopped. This compensates for both scale factor and distance per pulse and therefore should be entered in engineering units (RPM, RPS, Degrees/sec, In/s, etc)
- Distance Per Pulse - Sets the scale factor between pulses and distance. The library already accounts for the decoding scale factor (1x, 2x, 4x) separately so this value should be set exclusively based on the encoder's Pulses per Revolution and any gearing following the encoder.
- Reverse Direction - Sets the direction the encoder counts, used to flip the direction if the encoder mounting makes the default counting direction unintuitive.
- Samples to Average - Sets the number of samples to average when determining the period. Averaging may be desired to account for mechanical imperfections (such as unevenly spaced reflectors when using a reflective sensor as an encoder) or as oversampling to increase resolution. Valid values are 1 to 127 samples.

Starting, Stopping and Resetting Encoders

C++

```
Encoder *sampleEncoder = new Encoder(0, 1, false, Encoder::EncodingType::k4X);  
sampleEncoder->Reset();
```

Java

```
Encoder sampleEncoder = new Encoder(0, 1, false, Encoder.EncodingType.k4X);  
sampleEncoder.reset();
```

The encoder will begin counting as soon as it is created. To reset the encoder value to 0 call reset().

Getting Encoder Values

C++

```
Encoder *sampleEncoder = new Encoder(0, 1, false, Encoder::EncodingType::k4X);  
int count = sampleEncoder->Get();  
double distance = sampleEncoder->GetRaw();  
double distance = sampleEncoder->GetDistance();  
double period = sampleEncoder->GetPeriod();  
double rate = sampleEncoder->GetRate();  
boolean direction = sampleEncoder->GetDirection();  
boolean stopped = sampleEncoder->GetStopped();
```

Encoders - Measuring rotation of a wheel or other shaft

Java

```
Encoder sampleEncoder = new Encoder(0, 1, false, Encoder.EncodingType.k4X);
int count = sampleEncoder.get();
double distance = sampleEncoder.getRaw();
double distance = sampleEncoder.getDistance();
double period = sampleEncoder.getPeriod();
double rate = sampleEncoder.getRate();
boolean direction = sampleEncoder.getDirection();
boolean stopped = sampleEncoder.getStopped();
```

The following values can be retrieved from the encoder:

- Count - The current count. May be reset by calling reset().
- Raw Count - The count without compensation for decoding scale factor.
- Distance - The current distance reading from the counter. This is the count multiplied by the Distance Per Count scale factor.
- Period - The current period of the counter in seconds. If the counter is stopped this value may return 0. This is deprecated, it is recommended to use rate instead.
- Rate - The current rate of the counter in units/sec. It is calculated using the DistancePerPulse divided by the period. If the counter is stopped this value may return Inf or NaN, depending on language.
- Direction - The direction of the last value change (true for Up, false for Down)
- Stopped - If the counter is currently stopped (period has exceeded Max Period)