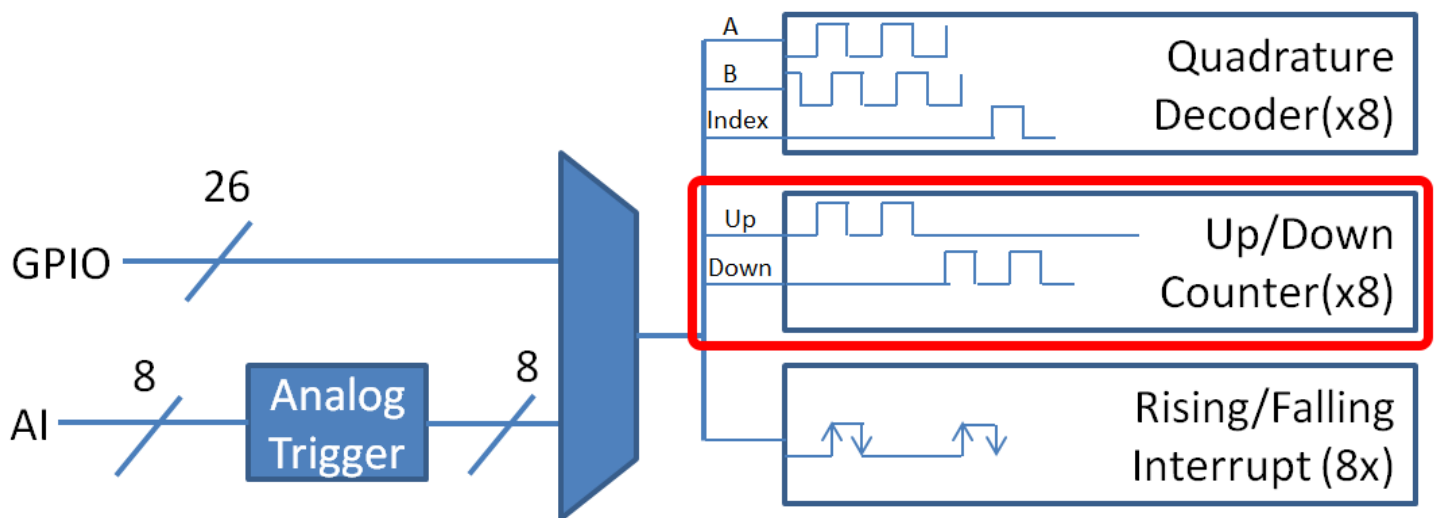


Counters - Measuring rotation, counting pulses and more

Counter objects are extremely flexible elements that can count input from either a digital input signal or an analog trigger.

Counter Overview

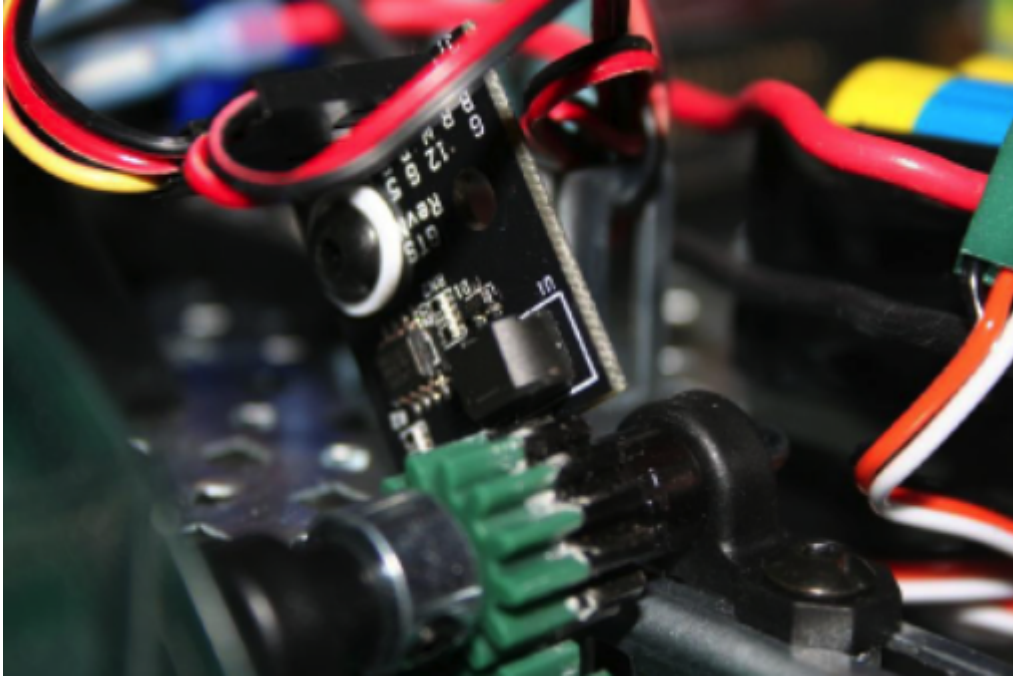


There are 8 Up/Down Counter units contained in the FPGA which can each operate in a number of modes based on the type of input signal:

- Gear-tooth/Pulse Width mode - Enables up/down counting based on the width of an input pulse. This is used to implement the GearTooth sensor class with direction sensing.
- Semi-period mode - Counts the period of a portion of the input signal. This mode is used by the Ultrasonic class to measure the time of flight of the echo pulse.
- External Direction mode - Can count edges of a signal on one input with the direction (up/down) determined by a second input
- "Normal mode"/Two Pulse mode - Can count edges from 2 independent sources (1 up, 1 down)

Counters - Measuring rotation, counting pulses and more

Gear-Tooth Mode and GearTooth Sensors



Gear-tooth sensors are designed to be mounted adjacent to spinning ferrous gear or sprocket teeth and detect whenever a tooth passes. The gear-tooth sensor is a Hall-effect device that uses a magnet and solid-state device that can measure changes in the field caused by the passing teeth. The picture above shows a gear-tooth sensor mounted to measure a metal gear rotation. Notice that a metal gear is attached to the plastic gear. The gear tooth sensor needs a ferrous material passing by it to detect rotation.

The Gear-Tooth mode of the FPGA counter is designed to work with gear-tooth sensors which indicate the direction of rotation by changing the length of the pulse they emit as each tooth passes such as the ATS651 provided in the 2006 FRC KOP.

Semi-Period mode

```
C++
Counter *exampleCounterHi = new Counter(0);
Counter *exampleCounterLow = new Counter(3);
exampleCounterHi->SetSemiPeriodMode(true);
exampleCounterLow->SetSemiPeriodMode(false);
double highPulse = exampleCounterHi->GetPeriod();
double lowPulse = exampleCounterLow->GetPeriod();
```

Counters - Measuring rotation, counting pulses and more

Java

```
Counter exampleCounterHi = new Counter(0);
Counter exampleCounterLow = new Counter(3);
exampleCounterHi.setSemiPeriodMode(true);
exampleCounterLow.setSemiPeriodMode(false);
double highPulse = exampleCounterHi.getPeriod();
double lowPulse = exampleCounterLow.getPeriod();
```

The semi-period mode of the counter will measure the pulse width of either a high pulse (rising edge to falling edge) or a low pulse (falling edge to rising edge) on a single source (the Up Source). Call `setSemiPeriodMode(true)` to measure high pulses and `setSemiPeriodMode(false)` to measure low pulses. In either case, call `getPeriod()` to obtain the length of the last measured pulse (in seconds).

External Direction mode

The external direction mode of the counter counts edges on one source (the Up Source) and uses the other source (the Down Source) to determine direction. The most common usage of this mode is quadrature decoding in 1x and 2x mode. This use case is handled by the Encoder class which sets up an internal Counter object, and is covered in the next article [Encoders - Measuring rotation of a wheel or other shaft](#).

Normal mode

C++

```
Counter *normalCounter = new Counter();
normalCounter->SetUpSource(1);
normalCounter->SetUpDownCounterMode();
```

Java

```
Counter normalCounter = new Counter();
normalCounter.setUpSource(1);
normalCounter.setUpDownCounterMode();
```

The "normal mode" of the counter, also known as Up/Down mode or Two Pulse mode, counts pulses occurring on up to two separate sources, one source for Up and one source for Down. A common use case of this mode is using a single source (the Up Source) with a reflective sensor or hall effect sensor as a single direction encoder. The code example above shows an alternate method of setting up the Counter sources, this method is valid for any of the modes. The method shown in the Semi-Period mode example is also perfectly valid for all modes of the counter including the Normal Mode.

Counters - Measuring rotation, counting pulses and more

Counter Settings

C++

```
Counter *normalCounter = new Counter(1);
normalCounter->SetMaxPeriod(.1);
normalCounter->SetUpdateWhenEmpty(true);
normalCounter->SetReverseDirection(false);
normalCounter->SetSamplesToAverage(10);
normalCounter->SetDistancePerPulse(12);
```

Java

```
Counter normalCounter = new Counter(1);
normalCounter.setMaxPeriod(.1);
normalCounter.setUpdateWhenEmpty(true);
normalCounter.setReverseDirection(false);
normalCounter.setSamplesToAverage(10);
normalCounter.setDistancePerPulse(12);
```

There are a few different parameters that can be set to control various aspects of the counter behavior:

- 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.
- Update When Empty - Setting this to false will keep the most recent period on the counter when the counter is determined to be stalled (based on the Max Period described above). Setting this parameter to True will return 0 as the period of a stalled counter.
- Reverse Direction - Valid in external direction mode only. Setting this parameter to true reverses the counting direction of the external direction mode of the counter.
- 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.
- Distance Per Pulse - Sets the multiplier used to determine distance from count when using the `getDistance()` method.

Resetting the counter

C++

```
Counter *normalCounter = new Counter(1);
normalCounter->Reset();
```

Counters - Measuring rotation, counting pulses and more

Java

```
Counter normalCounter = new Counter(1);  
normalCounter.reset();
```

Counters begin counting as soon as they are instantiated. To reset the counter value to 0 call `reset()`.

Getting Counter Values

C++

```
Counter *normalCounter = new Counter(1);  
int count = normalCounter->Get();  
double distance = normalCounter->GetDistance();  
double period = normalCounter->GetPeriod();  
double rate = normalCounter->GetRate();  
bool direction = normalCounter->GetDirection();  
bool stopped = normalCounter->GetStopped();
```

Java

```
Counter normalCounter = new Counter(1);  
int count = normalCounter.get();  
double distance = normalCounter.getDistance();  
double period = normalCounter.getPeriod();  
double rate = normalCounter.getRate();  
boolean direction = normalCounter.getDirection();  
boolean stopped = normalCounter.getStopped();
```

The following values can be retrieved from the counter:

- Count - The current count. May be reset by calling `reset()`
- 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, depending on the setting of the Update When Empty parameter.
- 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)