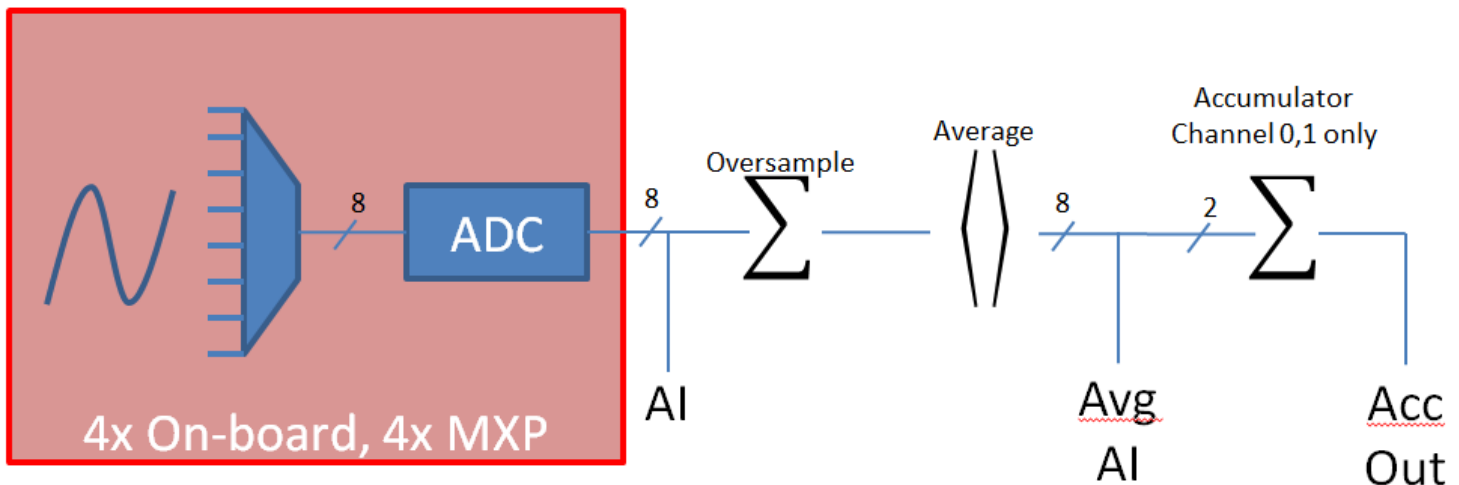


## Analog inputs

# Analog inputs

The roboRIO Analog to Digital module has a number of features not available on simpler controllers. It will automatically sample the analog channels in a round robin fashion, providing a combined sample rate of 500 ks/s (500,000 samples / second). These channels can be optionally oversampled and averaged to provide the value that is used by the program. There are raw integer and floating point voltage outputs available in addition to the averaged values. The diagram below outlines this process.

## Analog System Diagram



When the system averages a number of samples, the division results in a fractional part of the answer that is lost in producing the integer valued result. Oversampling is a technique where extra samples are summed, but not divided down to produce the average. Suppose the system were oversampling by 16 times – that would mean that the values returned were actually 16 times the average. Using the oversampled value gives additional precision in the returned value.

## Constructing an Analog Input

```
C++
AnalogInput *ai;
ai = new AnalogInput(0);
```

```
Java
AnalogInput ai;
ai = new AnalogInput(0);
```

# Analog inputs

To construct an AnalogInput object, simply pass in the channel number for the desired input.

## Oversampling and Averaging

$$Oversample(AI_0) = \sum_0^{2^N-1} AI$$

Where N is the number of Oversample bits

$$Average = \left( \sum_0^{2^M-1} AI_0 \right) / 2^M$$

Where M is the number of Average bits

$$f_{avg} = \frac{f_s}{2^{(M+N)}}$$

Where  $f_s$  is the original sampling frequency

The number of averaged and oversampled values are always powers of two (number of bits of oversampling/averaging). Therefore the number of oversampled or averaged values is two ^ bits, where 'bits' is passed to the methods: SetOversampleBits(bits) and SetAverageBits(bits). The actual rate that values are produced from the analog input channel is reduced by the number of averaged and oversampled values. For example, setting the number of oversampled bits to 4 and the average bits to 2 would reduce the number of delivered samples by 16x and 4x, or 64x total.

## Code example

C++

```
AnalogInput *exampleAnalog = new AnalogInput(0);  
int bits;  
exampleAnalog->SetOversampleBits(4);  
bits = exampleAnalog->GetOversampleBits();  
exampleAnalog->SetAverageBits(2);  
bits = exampleAnalog->GetAverageBits();
```

Java

```
AnalogInput exampleAnalog = new AnalogInput(0);  
int bits;  
exampleAnalog.setOversampleBits(4);  
bits = exampleAnalog.getOversampleBits();
```

# Analog inputs

```
exampleAnalog.setAverageBits(2);  
bits = exampleAnalog.getAverageBits();
```

The above code shows an example of how to get and set the number of oversample bits and average bits on an analog channel

## Sample Rate

C++  
`AnalogInput::SetSampleRate(62500);`

Java  
`AnalogInput.setGlobalSampleRate(62500);`

The sample rate is fixed per analog I/O module, so all the channels on a given module must sample at the same rate. However, the averaging and oversampling rates can be changed for each channel. The use of some sensors (currently just the Gyro) will set the sample rate to a specific value for the module it is connected to. The example above shows setting the sample rate for a module to the default value of 62,500 samples per channel per second (500kS/s total).

## Reading Analog Values

C++  
`AnalogInput *exampleAnalog = new AnalogInput(0);  
int raw = exampleAnalog->GetValue();  
double volts = exampleAnalog->GetVoltage();  
int averageRaw = exampleAnalog->GetAverageValue();  
double averageVolts = exampleAnalog->GetAverageVoltage();`

Java  
`AnalogInput exampleAnalog = new AnalogInput(0);  
int raw = exampleAnalog.getValue();  
double volts = exampleAnalog.getVoltage();  
int averageRaw = exampleAnalog.getAverageValue();  
double averageVolts = exampleAnalog.getAverageVoltage();`

There are a number of options for reading Analog input values from an analog channel:

1. Raw value - The instantaneous raw 12-bit (0-4096) value representing the 0-5V range of the ADC. Note that this method does not take into account the calibration information stored in the module.

## Analog inputs

2. Voltage - The instantaneous voltage value of the channel. This method takes into account the calibration information stored in the module to convert the raw value to a voltage.
3. Average Raw value - The raw, unscaled value output from the oversampling and averaging engine. See above for information on the effect of oversampling and averaging and how to set the number of bits for each.
4. Average Voltage - The scaled voltage value output from the oversampling and averaging engine. This method uses the stored calibration information to convert the raw average value into a voltage.

## Accumulator

The analog accumulator is a part of the FPGA that acts as an integrator for analog signals, summing the value over time. A common example of where this behavior is desired is for a gyro. A gyro outputs an analog signal corresponding to the rate of rotation, however the measurement commonly desired is heading or total rotational displacement. To get heading from rate, you perform an integration. By performing this operation at the hardware level it can occur much quicker than if you were to attempt to implement it in the robot code. The accumulator can also apply an offset to the analog value before adding it to the accumulator. Returning to the gyro example, most gyros output a voltage of 1/2 of the full scale when not rotating and vary the voltage above and below that reference to indicate direction of rotation.

## Setting up an accumulator

### C++

```
AnalogInput *exampleAnalog = new AnalogInput(0);
exampleAnalog->SetAccumulatorInitialValue(0);
exampleAnalog->SetAccumulatorCenter(2048);
exampleAnalog->SetAccumulatorDeadband(10);
exampleAnalog->ResetAccumulator();
```

### Java

```
AnalogInput exampleAnalog = new AnalogInput(0);
exampleAnalog.setAccumulatorInitialValue(0);
exampleAnalog.setAccumulatorCenter(2048);
exampleAnalog.setAccumulatorDeadband(10);
exampleAnalog.resetAccumulator();
```

There are two accumulators implemented in the FPGA, connected to channels 0 and 1. Any device which you wish to use with the analog accumulator must be attached to one of these two

# Analog inputs

channels. There are no mandatory parameters that must be set to use the accumulator, however depending on the device you may wish to set some or all of the following:

1. Accumulator Initial Value - This is the raw value the accumulator returns to when reset. It is added to the output of the hardware accumulator before the value is returned to the code.
2. Accumulator Center - This raw value is subtracted from each sample before the sample is applied to the accumulator. Note that the accumulator is after the oversample and averaging engine in the pipeline so oversampling will affect the appropriate value for this parameter.
3. Accumulator Deadband - The raw value deadband around the center point where the accumulator will treat the sample as 0.
4. Accumulator Reset - Resets the value of the accumulator to the Initial Value (0 by default).

## Reading from an Accumulator

### C++

```
AnalogInput *exampleAnalog = new AnalogInput(0);
long count = exampleAnalog->GetAccumulatorCount();
long value = exampleAnalog->GetAccumulatorValue();
AccumulatorResult *result = new AccumulatorResult();
exampleAnalog->GetAccumulatorOutput(result);
count = result->count;
value = result->value;
```

### Java

```
AnalogInput exampleAnalog = new AnalogInput(0);
long count = exampleAnalog.getAccumulatorCount();
long value = exampleAnalog.getAccumulatorValue();
AccumulatorResult result = new AccumulatorResult();
exampleAnalog.getAccumulatorOutput(result);
count = result.count;
value = result.value;
```

Two separate pieces of information can be read from the accumulator in three total ways:

1. Count - The number of samples that have been added to the accumulator since the last reset.
2. Value - The value currently in the accumulator

## Analog inputs

3. Combined - Retrieve the count and value together to assure synchronization. This should be used if you are going to use the count and value in the same calculation such as averaging.