stamp.peripheral.io.ADC

Class AtoD

```
java.lang.Object
  |
  +--stamp.peripheral.io.ADC.AtoD
```

Direct Known Subclasses:
ADC0831, LTC1298

---

public abstract class **AtoD**
extends Object

This is an abstract class encapsulating the capabilities of a generic Analog to Digital converter (ADC). This class is suitable for ADCs up to 16-bits of resolution.

Includes methods for reading the RAW value from the ADC chip, as well as methods for converting this RAW value to volts, millivolts, and temperature.

```
Revision History:
Ver 1.0 - 11/27/02: Initial release of class submitted to Parallax Inc.
                    by customer Tim Constable of Boston, MA
                    Evaluated and modified by Steve Dill of Parallax Inc.
```

Field Summary

protected int	**lastRaw** last calculated raw ADC value
protected int	**readSize** Bytes to read for specific ADC
protected int	**resolution** Range of the sensor

Constructor Summary

AtoD()

Method Summary

protected int	**calcMV**(int raw) Calculates and stores millivolt value for later retrieval.
String	**calcTemp**(boolean sym) Convert answer into a temperature format for devices measuring the temperature as a voltage F.
int	**lastMV**() Get last calculated millivolt value.
int	**lastRaw**() Get raw DAC value from chip, value depends on resolution of chip.
String	**lastVf**() Get last read value and format it in volts.
abstract int	**read**(int channel) Read value from A to D chip.
int	**readSmooth**(int channel, int times) Read from the ADC chip multiple times, then return the average.
void	**setBitValue**(int bitHigh, int bitLow, int offset) Each ADC has a bit value that represents the millivolt value between each numeric difference in the RAW value received from the ADC.

Methods inherited from class java.lang.Object
equals

Field Detail

readSize

```
protected int readSize
```

Bytes to read for specific ADC

---

resolution

```
protected int resolution
```

Range of the sensor

---

lastRaw

```
protected int lastRaw
```

last calculated raw ADC value

Constructor Detail

AtoD

```
public AtoD()
```

Method Detail

lastVf

```
public String lastVf()
```

Get last read value and format it in volts.

Returns:
voltage in a formatted string

---

lastMV

```
public int lastMV()
```

Get last calculated millivolt value.

Returns:
voltage in millivolts

---

lastRaw

```
public int lastRaw()
```

Get raw DAC value from chip, value depends on resolution of chip.

Returns:
raw value read from ADC

---

setBitValue

```
public void setBitValue(int bitHigh,
```

```
                    int bitLow,
                    int offset)
```

Each ADC has a bit value that represents the millivolt value between each numeric difference in the RAW value received from the ADC. This bit value needs to be calculated for each ADC chip, and then passed into this method so the correct math can be performed to obtain the voltage.

Using the 8-bit ADC (ADC0831) as an example:
Divide the range of the voltage being measured, lets say 5000 mV by the resolution of the chip 256.
5000/256=19.53125
This number 19.53125 is the bit value of this ADC. Since the Javelin does not have native floating point math, we will need to format this value and pass it in as two values, a high bit & low bit. The high bit (`bitHigh`), is simply the whole number to the left of the decimal, in this case 19. The low bit (`bitLow`), needs to follow the algorithm below:
Shift low bit left by 16: *0.53125*65536=34816*
Then if the answer is greater than a signed integer (32767), which in our example it is, you will do the following:
Subtract the answer from 65536: *65536-34816=30720*, then negate it *-30720*. These are the two values that you will pass in 19 & -30720. If your ADC has the capability to begin measuring from anything but 0 volts, you will need to place this amount as millivolts into the offset. For example, if your ADC is set from 2 to 4 volts, then the offset will be 2000 millivolts.

Known ADC values (16-bit max):

8-bit ADC0831 - bit value is 19.53125 `bitHigh` = 19 `bitLow` = -30720 `offset` = -Vin, 0 if tied to ground.
12-bit LTC1298 - bit value is 1.22073125 `bitHigh` = 1 `bitLow` = 14463 `offset` = 0

Parameters:

`bitHigh` - High portion of the bit value
`bitLow` - Low portion of the bit value
`offset` - Offset for bit value, when -Vin is not tied to ground

---

read

```
public abstract int read(int channel)
```

Read value from A to D chip.

This abstract method is used for a variety of A to D chips. The channels are not implemented the same for each chip, if your chip is not listed below, refer to the chip's datasheet as to what 0, 1, 2 will be translated to.

ADC0831: Only has one channel CH0, use 0
LTC1298: 0=CH0, 1=CH1, 2=CH0-CH1, 3=CH1-CH0

Parameters:

`channel` - Specify 0,1,2,3 for specific ADC chip

Returns:

raw value from chip

---

readSmooth

```
public int readSmooth(int channel,
                      int times)
```

Read from the ADC chip multiple times, then return the average. Input values: 1 = 2 reads, 2 = 4 reads, 3 = 8 reads, 4 = 16 reads, or 5 = 32 reads
This abstract method is used for a variety of A to D chips. The channels are not implemented the same for each chip, if your chip is not listed below, refer to the chip's datasheet as to what 0, 1, 2, or 3 will be translated to.

ADC0831: Only has one channel CH0, use 0
LTC1298: 0=CH0, 1=CH1, 2=CH0-CH1, 3=CH1-CH0

Parameters:

`channel` - Specify 0,1,2,3 for specific ADC chip

`times` - number of times to smooth, input 1,2,3,4,5 for 2,4,8,16,32 reads

Returns:

smoothed value

---

calcTemp

```
public String calcTemp(boolean sym)
```

Convert answer into a temperature format for devices measuring the temperature as a voltage F.

Additionally add the "F" character for printing. This routine rounds up the tenths.

Parameters:

`sym` - - set to true to and add "F"

Returns:

Temperature as string

---

calcMV

```
protected int calcMV(int raw)
```

Calculates and stores millivolt value for later retrieval.

Parameters:

`raw` - A to D value

Returns:

millivolt
