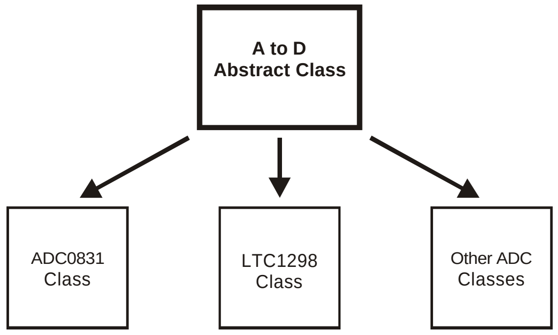
599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office/Tech Support: (916) 624-8333
Fax: (916) 624-8003

Web Site: www.javelinstamp.com
Home Page: www.parallaxinc.com

General: info@parallaxinc.com
Sales: sales@parallaxinc.com
Technical: javelintech@parallaxinc.com



Contents

Introduction to the Analog to Digital Abstract Class

AtoD.java is an abstract class, which contains many useful methods that are common among various analog to digital converters (ADC), such as the ADC0831 or the LTC1298. Each of these classes inherits methods from this abstract class for their specific hardware. This allows for greater flexibility when any enhancements are made to the abstract class. This application note will explain how you can use this abstract class for your own specific ADC.

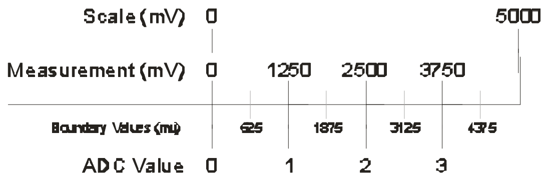
Implementing the Analog to Digital Abstract Class

You will need to create a class, or use one provide by Parallax, for the specific ADC of your choice. This class will interface the Javelin to the ADC chip, it should extend the AtoD.java abstract class. This will allow your programs access to all of the AtoD's methods. Your chip's library will pass the correct pin assignments, initialize, and read from the ADC chip.

How an ADC Works

An ADC chip will measure the voltage of a particular pin, usually from 0 to 5 volts. Then returns a value which corresponds to an interval based upon the scale (bit size) of the ADC. In the AtoD class we call this value *raw*. Let's use a 2-bit ADC for an example, refer to Figure 1.1 for this section.

Figure 1.1
2-bit ADC outputs and corresponding voltage measurements for a 5 volt scale


Scale (mV) 0 5000
Measurement (mV) 0 1250 2500 3750
Boundary Values (mV) 625 1875 3125 4375
ADC Value 0 1 2 3

A 2-bit ADC will have a resolution of 2-bits. Think of a bit as a binary number, since you have a 2-bit binary number you can make 4 different combinations: 00, 01, 10, 11. These *ADC values* will be represented by the number 0,1,2, and 3. Most ADC's will measure voltages from 0 to 5000 mV. Now refer to Table 1.1, there you will see the *calculated measurement* for each of the ADC values.

Table 1.1: 2-bit ADC Calculated Voltage

ADC Value (raw)	0	1	2	3
Calculated Measurement	0	1250	2500	3750

Bit Value

The bit value is the range of voltage represented by each bit. In our 2-bit ADC example, shown in Figure 1.1, the bit value would be (total millivolts/$2^{resolution}$) = 5000/4 = 1250 mV. This value is used below when for the boundaries when reading a measurement.

Measurement Boundaries

Each ADC has measurement boundaries, these boundaries will be calculated using the bit value. Please refer to Table 1.2 for the corresponding ranges for our 2-bit example.

Table 1.2: 2-bit ADC bounderies

ADC Value (raw)	0	1	2	3
Bounderies	0 to 625[1]	625 to 1875	1875 to 3125	3125 to 4375[2]

Note 1: The range of the lowest interval is ½ of the bit value.
Note 2: Due to the resolution, the highest voltage a 2-bit ADC can report is 4375 millivolts.

Differences Between ADC Chips

ADC's vary in their bit size which will determine the precision of the device. Some will have multiple channels, and others can vary their measurement range.

The ADC0831, for example, has the ability to narrow the measurement width by raising -Vin, or adjusting the Vref width. The narrower the width, the more precise the measurement can be. This happens since the 256 measurement intervals will be placed over a smaller range.

The LTC1298, for example, has the ability to measure 2 voltages simultaneously. And can return the difference between both of these measurements. It is a 12-bit chip and therefore can have 4096 intervals, which of course would make it more accurate, compared to the ADC0831.

You will need to do some research to find out which one will best suit your needs. Contact the manufacture for the specific type of features for your project.

Downloads, Parts, and Equipment for the AtoD abstract class

This application note (AppNote006-AnalogToDigital.pdf), the abstract class (AtoD.java), the javadoc file (AtoD.pdf), Program Listing 1.1 (AtoD_Test.java), the demonstration program (AtoD_Demo.java), and Program Listing 1.2 (ADC0831.java from AppNote007) are all available to you for free download from:

http://www.javelinstamp.com/Applications.htm

You can use the AppNote006-AnalogToDigital.exe, to install the files listed below. These files must be located in specific paths within the Javelin Stamp IDE directory. Although the path to this directory can be different, the default root path is: C:\Program Files\Parallax Inc\Javelin Stamp IDE

The file list below is organized by directory, then by filename, verify that your file list is organized in the same way.

<root path>\doc\AppNote006-AnalogToDigital.pdf

```
<root path>\doc\AtoD.pdf
<root path>\lib\stamp\peripheral\io\ADC\AtoD.java
<root path>\Projects\examples\peripheral\io\ADC\AtoD_Test.java
<root path>\Projects\examples\peripheral\io\ADC\AtoD_Demo.java
```

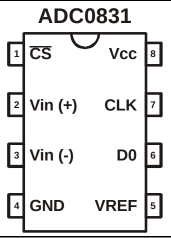
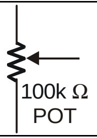
In addition to the files above, the following class file is also required. ADC0831.java can be found in AppNote007 and must be installed in the following directory:

```
<root path>\lib\stamp\peripheral\io\ADC\ADC0831.java
```

This file is very similar to Program Listing 1.2, except Program Listing 1.2 does not have the embedded JavaDocs. You could copy Program Listing 1.2 into the path shown above to test this AppNote, but I would recommend downloading the entire AppNote007.

The ADC0831 chip will be used as an example for testing the AtoD abstract class. Table 1.3 lists the parts you will need for this application note.

Table 1.3: Parts List

Quantity	Part Ordering Info and Part Description	Schematic Symbol
1	National Semiconductor's ADC0831 Parallax Part #ADC0831	 ADC0831 1 CS 2 Vin (+) 3 Vin (-) 4 GND 8 Vcc 7 CLK 6 D0 5 VREF
1	100K Potentiometer Parallax Part #152-01040	

The equipment used to test this example includes a Javelin Stamp, Javelin Stamp Demo Board, 7.5 V, 1000 mA DC power supply, serial cable, and PC with the Javelin Stamp IDE v2.01.

An Example Circuit with the ADC0831

Figure 1.2 is the circuit that will be used within this application note. By turning the potentiometer you will introduce a voltage (0-5 V) to the ADC0831. The ADC0831 will read this voltage and send a numeric number that represents this value to the Javelin Stamp. The Javelin Stamp will utilize methods within the AtoD abstract class and convert this ADC value into the correct voltage and transmit this message to the IDE's *messages from the Javelin window*. This scenario, as well as the circuit measurement, is the same for AppNote006-ADC0831.

Here's how you connect the ADC0831 to the Javelin Stamp (see Figure 1.2).

- Pin 1, the chip select pin (CS) of the ADC0831 is connected to pin 12 (P7) of the Javelin Stamp.
- Pin 2, the positive input voltage pin ($Vin^{(+)}$) of the ADC0831 is connected to the *wiper* of the potentiometer. The other two leads of the potentiometer are connected to ground (Vss) and +5 V (Vdd).
- Pin 3, the negative input voltage pin ($Vin^{(-)}$) of the ADC0831 is connected to ground (Vss).
- Pin 4, the ground pin (GND) of the ADC0831 is connected to ground (Vss).
- Pin 5, the voltage reference pin (Vref) is connected to +5 V (Vdd).
- Pin 6, the data out pin (DO) of the ADC0831 is connected to pin 5 (P0).
- Pin 7, the clock pin (CLK) of the ADC0831 is connected to pin 11 (P6) of the Javelin Stamp.
- Pin 8, the power pin (Vcc) of the ADC0831 is connected to +5 V (Vdd).

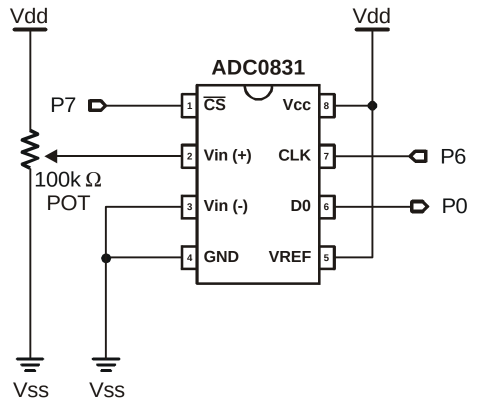


Figure 1.2
Wiring diagram for the ADC0831

Testing the ADC0831 Circuit

Program Listing 1.1 is a short program that will verify that the circuit in Figure 1.2 is working properly. This program will display the current voltage being read from the potentiometer. When the program is executed it will create an ADC0831 object called "**`adc`**". This object contains the methods from both the ADC0831.java and the AtoD.java libraries.

```
ADC0831 adc = new ADC0831(CPU.pin0,CPU.pin6,CPU.pin7); //Create ADC object
```

Next, the program will clear the Javelin's message window by printing the value **`CLS`**.

```
System.out.print(CLS);                 // Clear the display
```

Next, the program will enter a do loop, and then by printing the **HOME** value it will position the cursor to the top-left corner of the screen. This is done so the output data from the Javelin will remain in one area of the screen.

```
System.out.print(HOME);                // position cursor
```

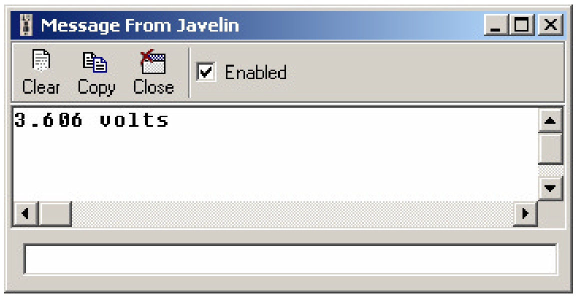
Now the program will read the ADC and store the data within the **adc** object.

```
adc.read(0);                           // read and store values
```

To recall the values from the **adc** object and display them in the message window, the program executes the **lastVf()** method within a print statement. This method will display the last read voltage with the proper formatting in decimal.

```
System.out.print(adc.lastVf());        // display formatted volts
System.out.println(" volts ");
```

The test program (Program Listing 1.1) will display output, to the Javelin's IDE message window, similar to Figure 1.3.



```
3.606 volts
```

Figure 1.3
ADC0831 Test output window

If your output voltage value is zero (or not accurate), try the following:

- ✓ Verify that the potentiometer you are using is rated at 100 K.
- ✓ Carefully verify that each pin on your ADC0831 chip match the circuit shown in Figure 1.2.

Tip Verify with a Voltmeter	You can also use a voltmeter to verify that the output window matches the actual voltage. To do this you would connect the ground lead of your voltmeter (COM) to the ground side of the potentiometer (Vss). Connect the positive lead of your voltmeter to the wiper leg of the potentiometer (Vin$^{(+)}$). Select the proper setting on the voltmeter. If the voltage shown is negative then the connection is backwards. To fix this, simply switch the two leads.

Program Listing 1.1 – The AtoD_Test

```
import stamp.core.*;
import stamp.peripheral.io.ADC.*;

/*
 * Tests the ADC0831 circuit from the Application Note #006.
 * Version 1.0 - 11/29/02
 */

public class AtoD_Test {

  final static char HOME = 0x01;          // Position cursor upper-left
  final static char CLS = '\u0010';       // Clear Screen

  public static void main() {
    ADC0831 adc = new ADC0831(CPU.pin0,CPU.pin6,CPU.pin7);  // create ADC object
    System.out.print(CLS);                                  // Clear the display

    do {
      System.out.print(HOME);             // position cursor
      adc.read(0);                        // read and store values
      System.out.print(adc.lastVf());     // display formatted volts
      System.out.println(" volts ");
    } while (true);                       // do forever

  }// end method: main
}// end class: AtoD_Test
```

Creating a Library for Your ADC chip

The library class for a specific ADC will contain the constructor/methods needed for to your ADC chip. This section will explain how to create your own library class by using the ADC0831 chip as an example. You will need to refer to the data sheet for your particular chip.

The Advantage of an Abstract Class

The bulk of the library class is inherited from the **`AtoD`** abstract class. This is where all the work goes for reading, calculating and formatting the result. The methods within the abstract class are common for all ADC libraries. When a new method is created within this abstract class, it is available for all of the ADC libraries.

You will not need to rewrite each ADC library, just download the latest version of the **`AtoD`** abstract class to obtain the newer methods.

The ADC Library Class Constructors/Methods

This section will explain what is unique to each ADC chip. There are five specifications that you will need to create.

1) The library class
2) calculating the bit value
3) The constructor
4) Unique characteristics of the chip.
5) Unique methods to read the chip.

The Library Class

This class will extend the **`AtoD`** abstract class. Here's how:

```
public class ADC0831 extends AtoD {
```

This class is named **`ADC0831`**, and it **`extends`** the class **`AtoD`**. When an object is created of type **`ADC0831`** from main, it will include *all* the methods from the **`AtoD`** abstract class as well as the **`ADC0831`** class.

The bit Constant

A bit is the smallest amount of measuring accuracy that a particular chip can measure. For an 8-bit ADC you will have 256 intervals across the voltage range. The Javelin will read this value from the ADC and calculate the voltage level it represents. The highest voltage an 8-bit ADC can report is 4.980 volts. Each increment (bit) of an 8-bit ADC is 19.53125 millivolts in width. There are 256 values from 0 to 255. The ADC measures a voltage, determines which value it is closest to and gives you this result. Therefore, there could be an error of ±19.53125/2 or 9.765625 millivolts. Since the Javelin does not support **`real`**, or **`float`**, types natively we need to break the number up into 2 segments, to the left (**`bitHigh`**) and right (**`bitLow`**) of the decimal point. The left side will always be a whole number and will work with the Javelin's math routines as is. The fractional portion will need to be converted so it can be manipulated with the math functions included within the AtoD class. Here's the formula for finding the proper *bit value* of a particular ADC.

The formula below needs to be calculated with a calculator.

Instructions	**Example based on the ADC0831**
1) Take the bit size of the ADC	8-bit
2) Convert this the number of intervals	$2^8 = 256$
3) Get maximum voltage ADC can read	5 volts
4) Convert this into millivolts	5 volts * 1000 mV = 5000 mV
5) Divide the max millivolts by the number of intervals of the ADC	5000/256

6)	This is your “bit value”	19.53125
7)	`bitHigh` equals the left side of the decimal	19
8)	Subtract `bitHigh` from the bit value to give you the fractional portion of the bit value	0.53125
9)	Multiply this value by 65536	0.53125*65536= 34816
10)	`bitLow` equals this value only if it is 32767 or less.	34816 is NOT less than 32767
11)	If this value is greater than 32767	Yes, it is
12)	Then subtract 65536	34816-65536= -30720
13)	This is your `bitLow` value	-30720

If your `bitLow` value is negative this is fine, it’s simply being represented in a signed integer format. Here are the two lines of code which set the bit value.

```
final static int bitHigh  = 19;          // High bit of 19.53125
final static int bitLow   = -30720;      // Low bit of 19.53125
```

The Constructor

The constructor will contain parameters which will represent the data pins of the Javelin Stamp that interacts with your ADC chip. The ADC chip’s protocol used will determine how many communication pins are needed. This information can be found in the datasheet for your particular chip. Here is an example of how this would be implemented for the ADC0831 which requires three communication pins:

```
public ADC0831(int datapin, int clockpin, int enablepin){
```

Within your class to use the same variable names that are passed in, use the following snippet of code.

```
this.datapin   = datapin;
this.clockpin  = clockpin;
this.enablepin = enablepin;
```

Each ADC will have a particular read size, as well as a size in which the ADC will partition the voltage range. This information can be found in the data sheet. For the ADC0831 it has a read size of 9 bits and a resolution of 256 (8 bits). This resolution is what we used with the formula above to find the bit value.

```
readSize   = 9;                 // Bits to Read on the ADC0831
resolution = 256;               // Range of the ADC0831
```

Pass the bit value into the AtoD method `setBitValue` of the AtoD abstract *(also called super)* class.

```
super.setBitValue(bitHigh,bitLow,0);  // Set bit value size in super class
```

The last section of the constructor initializes the ADC chip. In our example we set the enable pin low, then set the clock pin low. After a brief pause, we set the enable pin high, and then low. Initializing the pins in this fashion will guarantee that the first read from the ADC will be accurate. Of course there are other ways to do this, if this method does not work for your type of chip check the data sheet for your chip's startup sequence.

```
CPU.writePin(enablepin,false);   // init the bus
CPU.writePin(clockpin, false);
CPU.delay(100);                  // settle down
CPU.writePin(enablepin,false);
CPU.writePin(enablepin,true);
```

Unique Characteristics of an ADC chip

The ADC0831 has the ability to change the *range of voltage* for which it will measure. Doing so will alter the bit value for each interval of the 256 intervals that this 8-bit chip has. A method is needed to set the bit value and is included here in the ADC0831 library class, since it is specific to the ADC0831 chip. Simply recalculate the bit value constants and insert them here. For example, if you set vRef to 3 volts, then the ADC0831 will have a 0-3 volt measurement. So, when calculating your formula use 3000 millivolts instead of 5000 millivolts in step 4.

```
public void setBitValue(int bitHigh, int bitLow, int offset) {
  super.setBitValue(bitHigh,bitLow,offset);
}
```

Notice that this method includes an offset value. This value is for when you alter the ADC0831's –Vin. If you set –Vin to 1 volt, then Vref to 3 volts, the voltage range will be from 1-4 volts. So you will need a 1000 millivolts (1 volt) offset for the voltage calculations to work properly.

Reading From the ADC chip

Since there can be many differences reading among the various ADC chips. A unique read will need to be created for your specific chip. Although there is a read method within the **AtoD** class it is an empty method. This method needs to be there, and the method you create within your device library must match the same declaration as the read method within the **AtoD** abstract class.

```
public int read(int channel){
```

Since the ADC0831 only has one channel to measure, the **channel** parameter here is insignificant to the read method. But, needs to be present to stay compatible with the declaration for the **AtoD**'s read method. Next, the ADC0831's read method begins by setting the enable pin low.

```
CPU.writePin(enablePin,false);
```

Then by using the **`shiftIn`** command, read the value supplied by the ADC chip.

```
clockIn = CPU.shiftIn(dataPin,clockPin,readSize,CPU.POST_CLOCK_MSB);
```

The **`shiftIn`** command requires four parameters.

1) **`dataPin`** is the data pin number that was read in by the constructor.
2) **`clockPin`** is the clock pin number that was read in by the constructor.
3) **`readSize`** is the readSize variable declared within the constructor.
4) **`CPU.POST_CLOCK_MSB`** specifies post-clock sampling and MSB-first transmission for the shift methods.

Next, the method will set the enable pin to high and then wait for a short amount of time.

```
CPU.writePin(enablePin,true);
CPU.delay(100);
```

The last snippet of code simply sends the data read in from the ADC into the **`calcMV()`**method within the **`AtoD`** abstract class. This method will calculate the millivolts and store the answer within the **`AtoD`** object, until specifically requested from the **`main`** method.

```
lastRaw = clockIn;                              // raw is protected
super.calcV(clockIn);                           // copy to superclass
super.calcMV(clockIn);
return clockIn;
```

Program Listing 1.2 below is the actual ADC0831 library class without JavaDoc comments that was created for AppNote007. For the complete code including the JavaDoc comments please refer to AppNote007.

Program Listing 1.2 – The ADC0831 Library Class

```
/*
 * Copyright © 2002 Parallax Inc. All rights reserved.
 * Author Tim Constable - Boston, MA
 */

package stamp.peripheral.io.ADC;
import stamp.core.*;

public class ADC0831 extends AtoD {

  final static int bitHigh  = 19;           // High bit of 19.53125
  final static int bitLow   = -30720;       // Low bit of 19.53125
  public ADC0831(int dataPin, int clockPin, int enablePin){

    // update protected variables
    this.dataPin   = dataPin;
    this.clockPin  = clockPin;
    this.enablePin = enablePin;
    readSize   = 9;                         // Bytes to Read on the ADC0831
    resolution = 256;                       // Range of the ADC0831

    super.setBitValue(bitHigh,bitLow,0);    // Set bit value size in super class

    // initialize pins -- make low for high-going pulses
    CPU.writePin(enablePin,false);          // init the bus
    CPU.writePin(clockPin, false);
    CPU.delay(100);                         // settle down
    CPU.writePin(enablePin,false);
    CPU.writePin(enablePin,true);
  }// end constructor: ADC0831

  public void setBitValue(int bitHigh, int bitLow, int offset) {
    super.setBitValue(bitHigh,bitLow,offset);
  }//end method: setBitValue

  public int read(int channel){
    int clockIn;
    CPU.writePin(enablePin,false);
    clockIn = CPU.shiftIn(dataPin,clockPin,readSize,CPU.POST_CLOCK_MSB);
    CPU.writePin(enablePin,true);
    CPU.delay(100);
    lastRaw = clockIn;                               // raw is protected
    super.calcMV(clockIn);
    return clockIn;
  }// end method: read

//=============================================================================
// Methods and fields below this point are private.
//=============================================================================

  // Pin Constants unique to chip
  private int enablePin;                             // ADC0831 enable pin #1
  private int dataPin;                               // ADC0831 data pin #6
  private int clockPin;                              // ADC0831 clock pin #7

}// end class: ADC0831
```

The Demonstration Program

AtoD_Demo.java is a detailed, step-by-step, fully comprehensive demonstration of many of the methods available within the **`AtoD`** abstract class. The circuit shown in Figure 1.2 will be needed, verify that it is properly connected. This demonstration is best utilized when you follow the comments that are included within the class.

Published Resources – for More Information

This class was developed to allow the Javelin to interact with the ADC0831 chip. Much care has been taken in creating this class. Below you will find useful information that was used in creating this document.

"AN007 - Using the ADC0831 8-bit Serial I/O A/D Converter", Application Note, Parallax Inc., Nov. 2002
This document explains how to create the ADC0831 circuit, requirements for the ADC0831 chip, and how to interact the Javelin Stamp to the ADC0831 class files.

"AN008 - Using the LTC1298 12-bit Converter", Application Note, Parallax Inc., Nov. 2002
This document explains how to create the LTC1298 circuit, requirements for the LTC1298 chip, and how to interact the Javelin Stamp to the LTC1298 class files.

"ADC0831/ADC0832/ADC0834/ADC0838 8-Bit Serial I/O A/D Converters with Multiplexer Options", Data Sheet, National Semiconductor, 2002
This document will give you detailed information about the internal working of the ADC0831.

"LTC1286/LTC1298", Data Sheet, Linear Technology Corporation, 1994
This document will give you detailed information about the internal working of the LTC1298.

Javelin Stamp Discussion Forum – Questions and Answers

The Parallax, Inc. Javelin Stamp Discussion Forum is a searchable repository of design questions and answers for the Javelin Stamp. To view the Javelin Stamp Forum, go to www.javelinstamp.com and follow the Discussion link. You can also join this forum and post your own questions. The Parallax technical staff, and Javelin Stamp users who monitor the list, will see your questions and reply with helpful tips, part numbers, pointers to useful web pages, etc.