Analog Input Functions

Analog input is a way to read a voltage at one of the Arduino analog input pins. One thing that differentiates analog input from the other I/O mechanisms is that you do not need to set the pin direction. Analog input pins are always used exclusively for analog input.

**analogRead()**

Analog input on an Arduino is done using the `analogRead()` function. `analogRead()` takes a single parameter, the analog pin number, and returns an int that is proportional to the voltage at the pin. The int returned will always be in the range 0 to 1023. It is important to know what the default reference voltage is for the particular Arduino hardware being used is.

**NOTE:** For Arduino Unos and Megas, the reference voltage is 5V. For Arduino Dues and some other models, the reference voltage is 3.3V. Make sure to check what your reference voltage is prior to using `analogRead()`, or your code might behave in unexpected ways!

Example Code:

```cpp
int analogValue = analogRead(0);
```

**map()**

The `map()` function can shift a value in a given range into a value in a different range. If, for example, you wanted to read the voltage, in mV, rather than the normalized range given, you could write:

```cpp
int analogValue = analogRead(0);

// Map the normalized analog value to millivolts.
// NOTE: Assumes the reference voltage is 5V.
int milliVolts = map(analogValue, 0, 1023, 0, 5000);
```

**analogRead() Example**

We will extend the example used in Digital Input by replacing the button with a potentiometer which will control the rate at which the LED blinks. A potentiometer is also called a variable resistor. Like a resistor, there are two pins for each end. Unlike a resistor, there is a third pin connected to a wiper that slides across the internal resistor. This wiper effectively shorts out part of the resistor, allowing the resistance to change as the wiper moves back and forth.
These schematics are attached and can be viewed with Fritzing.

Blinking Light with Variable Rate Arduino Sketch

```cpp
#define BUTTON_PIN 2
// Pin definitions
#define LED_PIN 13
#define POT_PIN 0

// How often should we read the potentiometer state?
#define ANALOG_READ_INTERVAL_MS 100

//
// Global Variables
//

// Flag indicating if the LED is on
bool g_fLedOn = true;

// When was the last time we flipped the LED state?
unsigned long g_uiLastFlipMs = 0;
```

analogRead() Example
// When was the last time we checked the potentiometer state?
unsigned long g_uiLastPollMs = 0;

// Current LED blinking half-period
unsigned long g_uiFlipPeriodMs = 1000;

void setup()
{
    // Initialize LED pin as an output
    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
    // Get the current time on the microcontroller
    unsigned int uiTimeMs = millis();

    // If enough time has elapsed, try to read the potentiometer state
    if ((uiTimeMs - g_uiLastPollMs) >= ANALOG_READ_INTERVAL_MS)
    {
        int iAnalogVal = analogRead(POT_PIN);

        // Since the analog pin is at the center of a (0-10k)/10k voltage divider, the input will be between 2.5V and 5V.
        // This means that analogRead() will return a value between 512 and 1023.
        iAnalogVal = constrain(iAnalogVal, 512, 1023);

        // Map the analog value (512 - 1023) to a half-period length (50ms - 1000ms)
        int iFlipPeriod = map(iAnalogVal, 512, 1023, 50, 1000);
        g_uiFlipPeriodMs = (unsigned int)iFlipPeriod;

        // Record the last time we polled the potentiometer state
        g_uiLastPollMs = uiTimeMs;
    }

    // If enough time has elapsed, flip the LED state
    if ((uiTimeMs - g_uiLastFlipMs) >= g_uiFlipPeriodMs)
    {
        if (g_fLedOn)
        {
            // Turn LED off
            digitalWrite(LED_PIN, LOW);
        }
        else
        {
            // Turn LED on
            digitalWrite(LED_PIN, HIGH);
        }
    }
}
// Flip LED state
g_fLedOn = !g_fLedOn;

// Record the last time we flipped the LED state
g_uiLastFlipMs = uiTimeMs;
}
}

This code can be downloaded [here](#).