SparkFun Blocks for Intel® Edison - Arduino Block

Introduction

The Arduino Block for Edison provides the Intel® Edison with a direct, serial link to an Arduino-compatible ATmega328 microprocessor.

Why would you need an Arduino connected to your Edison? Isn't it powerful enough to handle anything that may be thrown at it? That's the problem – it's almost too powerful. Because it's running an operating system, it's incapable of real-time processing – the bread-and-butter of smaller microcontrollers like the ATmega328. Components which require precise timing – like WS2812 LEDs or servo motors – may be incompatible with the Edison as it can't reliably generate clock signals.

The Arduino block allows the Edison to offload those lower-level hardware tasks. Additional to that, if you've already written Arduino code for an external component, you don't have to port that code to the Edison – just run it on an Arduino block!

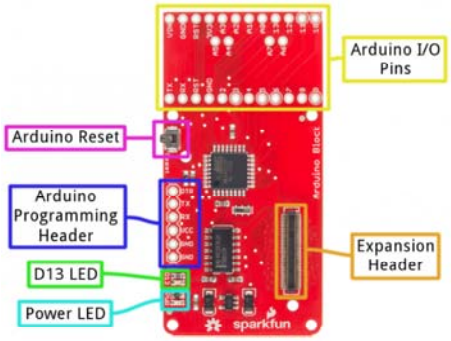
Suggested Reading

If you are unfamiliar with Blocks, take a look at the General Guide to Sparkfun Blocks for Intel Edison.

Other tutorials that may help you on your Arduino Block adventure include:

- Edison Getting Started Guide
- Powering Your Project
- What is an Arduino?
- Using the Arduino Pro Mini 3.3V

Board Overview



- **Expansion Header** – The 70-pin Expansion header breaks out the functionality of the Intel Edison. This header also passes signals and power throughout the stack. These function much like an Arduino Shield.
- **Arduino I/O Pins** – All of the Arduino's I/O pins are broken out to a pair of headers (plus a couple in between). This header footprint exactly matches that of the Arduino Pro Mini – if you have any Mini shields they should mate exactly to this header.
- **Arduino Programming Header** – The standard 6-pin FTDI header is used to program the Arduino's serial bootloader. Plug a 3.3V FTDI Basic in to program your Arduino.
- **D13 LED** – Every good Arduino needs an LED! This small, green LED is tied to the Arduino's pin 13. Great for blinking "Hello, world" or debugging.
- **Power LED** – The Arduino block has an on-board 3.3V regulator, and this LED is tied to the output of that regulator.
- **Arduino Reset Button** – This reset button is tied to the Arduino's reset line. It will only reset the Arduino; it has no effect on the Edison.

Schematic Overview

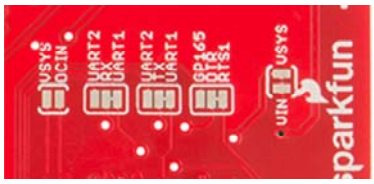
The Arduino block pairs the ATmega328 to your Edison via one of two UARTs. The board defaults to connecting the Arduino to Edison via UART1. Jumpers (see more below) allow you to select UART2, if your application requires. Take care using UART2, though, it's default utility is for console access to the Edison.

The Arduino Block has an on-board 3.3V voltage regulator, which takes its input from the Edison's VSYS bus. Since the Arduino is running at 3.3V, its **clock speed is limited to 8MHz**.

If you want to take a closer look at the schematic, download the PDF here.

Jumpers

On the back-side of the Arduino block, there are a handful of jumpers, which lend extra utility to the board.

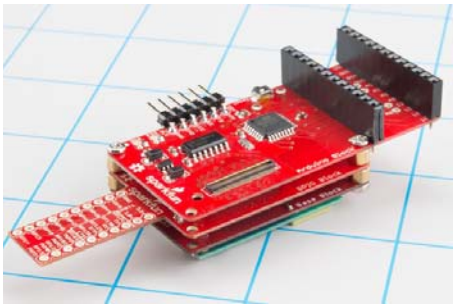


Three two-way jumpers – for RX, TX, and DTR – allow you to select between UART1 (default) and UART2. To switch these jumpers, grab a hobby knife, cut the default traces, and drop a solder blob between the middle pad and outer pad of your choice.

The jumper labeled **VIN/VSYS** allows you to remove the VSYS line from powering the Arduino block. This is handy if you need to isolate the Arduino block's power source from the Edison. In this case, you'll need to supply power (3.3-12V) externally via the "VIN" pin.

Using the Arduino Block

To use the Arduino Block, attach it to either an Edison or add it to a stack of other SparkFun Block's.



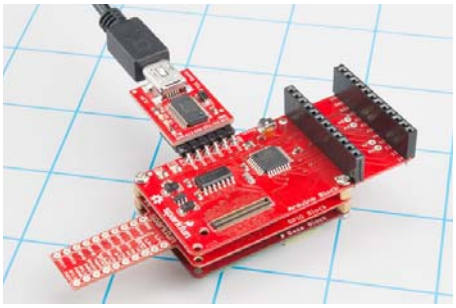
Arduino block stacked on top of a GPIO Block and a Base Block.

In order to supply power to your Edison, you'll need at least one additon Block in your stack. You can use a Base Block or Battery Block, for example.

Programming the Arduino

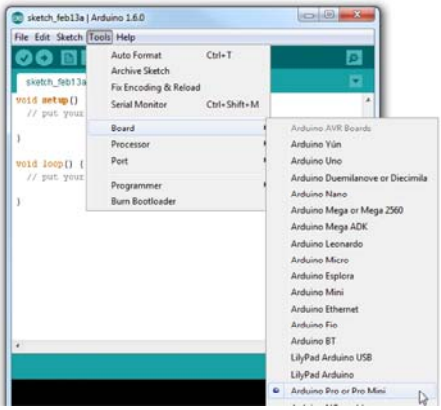
The Arduino on the Arduino Block can be programmed while it's either on or off the Edison. Depending on your application, though, it's recommended that you load code on the Arduino while it's disconnected from your Edison stack, before adding it to the rest of the system.

If you've ever uploaded an Arduino sketch to an Arduino Pro or Pro Mini, you're already familiar with uploading code to the Arduino block. Connect a 3.3V FTDI Basic to the 6-pin FTDI header on the board.

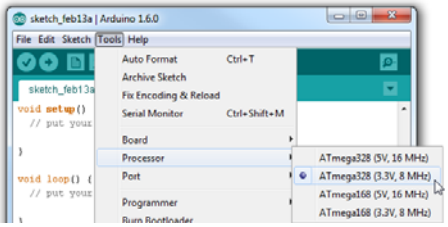


Using a 3.3V FTDI Basic to program the Arduino on the Arduino Block.

In Arduino (the non-Edison version of Arduino!), select "Arduino Pro or Pro Mini 3.3V/8MHz" from the Tools > Board menu. If you're using the latest release of Arduino (1.6 or later), first select **Arduino Pro or Pro Mini** from the "Board" menu.



Then select **ATmega328 (3.3V, 8MHz)** from the “Processor” menu.

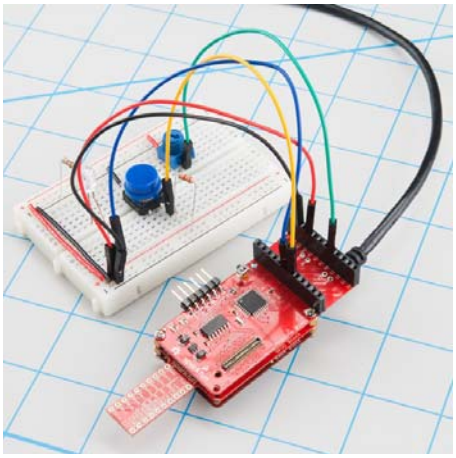


Then upload away!

Using the Arduino Pins

The Arduino’s I/O pins are all broken out to a pair of headers. These headers match up exactly to the Arduino Pro Mini. If you have any shields or piggyback boards for a Pro Mini, it should work seamlessly with the Arduino Block.

You can solder headers, wires, or any other connectors to these pins.



If you’re soldering headers to the pins, take extra care deciding which side to solder to. Depending on the rest of your Edison stackup, those headers might get in the way of connectors on other boards (the USB connectors on the Base and Console Blocks, in particular).

Connecting the Edison to the Arduino

The Arduino Block connects the Arduino to the Edison through a serial (UART) connection. Jumpers on the back of the board allow you select which of the Edison’s two UARTs mate with the Arduino. Unless you can’t avoid it, we recommend leaving the jumpers in the default configuration – the Edison’s UART2 is usually devoted to console access.

To program the Edison to control and interact with the Arduino, you'll need to use the UART to establish a communication protocol between the devices. See the next section for an easy example of UART communication between Arduino and Edison.

Controlling the Arduino Block with Firmata

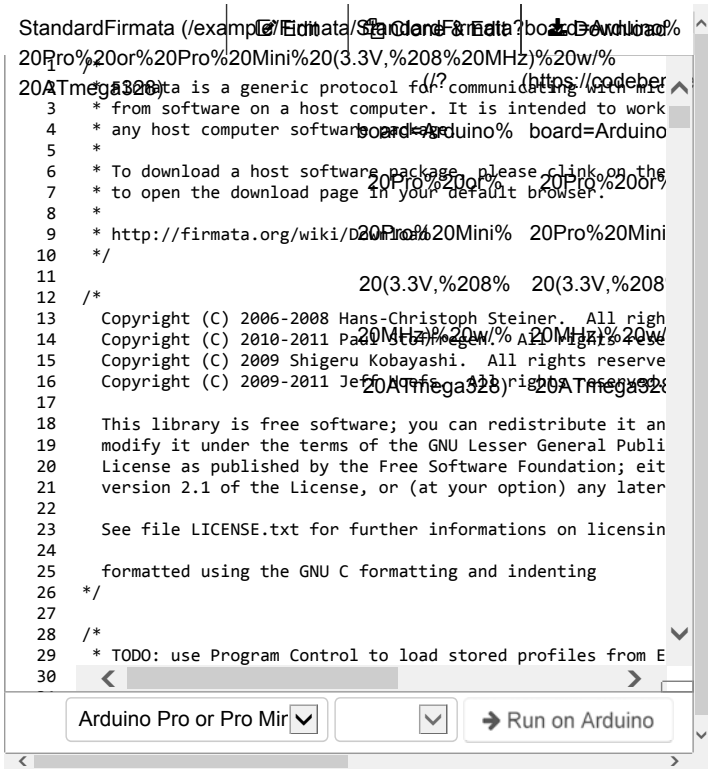
Firmata is an established protocol popular within the Arduino realm for applications that require a separate machine (usually a computer) to control the Arduino. It's a serial-based protocol that uses defined messages to set digital pins, read analog pins, and do everything else you're used to with Arduino.

Firmata is so useful, the standard Arduino IDE even ships with the Firmata library. Here's an example of how an Edison can be used to control and interact with an Arduino running Firmata.

Upload StandardFirmata to the Arduino

Before uploading any code to the Edison, let's load something into the Arduino. Once the Firmata code is running on your Arduino, you may never have to upload code to it again.

Using the standard Arduino IDE (i.e. *not* the IDE built for Edison), load up the "StandardFirmata" sketch by going to **File** > **Examples** > **Firmata** > **StandardFirmata**. If you have the Codebender addon installed, you can use the embed below to upload the code to your Arduino Block.



With the Firmata firmware uploaded, you can disconnect the FTDI Basic, and connect the Arduino Block to your Edison stack.

Edison Firmata for Arduino Client

The harder part of this equation is writing something that executes on the Edison which interacts with our Firmata-running Arduino. There are tons of great client examples in the Firmata GitHub profile, but nothing quite built for the Edison.

Riffing on the Firmata Processing example, we wrote this sketch to enact an Edison Firmata client.

> **Arduino version alert!** This Arduino sketch is intended to run on the Edison. You'll need to download the Edison Arduino IDE, and use that to upload this code to your Edison. For more help programming the Edison in Arduino, check out our Getting Started with Edison tutorial.

Here’s the sketch. Copy/paste from below, or grab the latest version from this Gist:

```
/*************************************************************
***
   Edison Firmata Client
   by: Jim Lindblom @ SparkFun Electronics
   created on: Februrary 12, 2015
   github:

   This is an Firmata client sketch for the Edison. It can
   communicate with an Arduino running Firmata over a Serial
   connection.

   Support for the following functions is written:
     firmata_init() -- set up firmata and pin reporting
     firmata_pinMode([pin], [0, 1, 2, 3, 4, 5, 6])
     firmata_digitalWrite([pin], [LOW/HIGH])
     firmata_analogWrite([pin], [0-255])
     firmata_analogRead([0-7])
     firmata_digitalRead([pin])
     firmata_servoWrite([pin], [value])

   Development Environment Specifics:
   Arduino 1.5.3 (for Edison)
   Intel Edison rev C
   Arduino Block for Edison
     Arduino should be running StandardFirmata

   This sketch is based on Firmata's processing client:
   https://github.com/firmata/processing
   As such, it is released under the same, free license. You c
an
   redistribute it and/or modify it under the terms of the GN
U
   Lesser General Public License as published by the Free
   Software Foundation; either version 2.1 of the License, or
   (at your option) any later version.

   Distributed as-is; no warranty is given.
*************************************************************
**/
// SerialEvent1 isn't defined in the Edison core (I think).
// To get some form of interrupt-driven Serial input, we'll re
ad
// serial in on a timer.
#include <TimerOne.h>

#define MAX_DATA_BYTES 4096
#define MAX_PINS 128

// Pin Mode definitons:
// Use any of these six values to set a pin to INPUT, OUTPUT,
// ANALOG, PWM, SERVO, SHIFT, or I2C.
enum const_pin_mode {
  MODE_INPUT,  // 0
  MODE_OUTPUT, // 1
  MODE_ANALOG, // 2
  MODE_PWM,    // 3
  MODE_SERVO,  // 4
  MODE_SHIFT,  // 5
  MODE_I2C     // 6
};

// Message Types
// Used by the low-level Firmata functions to set up the
// Firmata messages.
```

```
#define ANALOG_MESSAGE  0xE0
#define DIGITAL_MESSAGE 0x90
#define REPORT_ANALOG   0xC0
#define REPORT_DIGITAL  0xD0
#define START_SYSEX     0xF0
#define SET_PIN_MODE    0xF4
#define END_SYSEX       0xF7
#define REPORT_VERSION  0xF9
#define SYSTEM_RESET    0xFF

// Extended Commands:
// Used by the low-level Firmata functions to set up the
// Firmata messages.
#define SERVO_CONFIG            0x70
#define STRING_DATA             0x71
#define SHIFT_DATA              0x75
#define I2C_REQUEST             0x76
#define I2C_REPLY               0x77
#define I2C_CONFIG              0x78
#define EXTENDED_ANALOG         0x6F
#define PIN_STATE_QUERY         0x6D
#define PIN_STATE_RESPONSE      0x6E
#define CAPABILITY_QUERY        0x6B
#define CAPABILITY_RESPONSE     0x6C
#define ANALOG_MAPPING_QUERY    0x69
#define ANALOG_MAPPING_RESPONSE 0x6A
#define REPORT_FIRMWARE         0x79
#define SAMPLING_INTERVAL       0x7A
#define SYSEX_NON_REALTIME      0x7E
#define SYSEX_REALTIME          0x7F

// Flags and variables to keep track of message reading statu
s:
boolean parsingSysex = false;
int waitForData = 0;
int storedInputData[MAX_DATA_BYTES];
int sysexBytesRead = 0;
int executeMultiByteCommand = 0;
int multiByteChannel = 0;
// Variable arrays to keep track of pin values read in.
int digitalInputData[] = {0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0};
int analogInputData[] = {0, 0, 0, 0, 0, 0, 0, 0,
                         0, 0, 0, 0, 0, 0, 0, 0};
int analogChannel[MAX_PINS];
boolean blinkFlag = false;

void setup()
{
  // Debug messages are sent out Serial. Use the Serial monito
r
  // at 9600 bps to read pin values.
  Serial.begin(9600);

  // firmata_init sets up our firmata client. It tells the
  // Firmata device to begin streaming analog values and any
  // digital pin value changes.
  firmata_init();

  // Use firmata_pinMode([pin], [value]) to set up pins on the
  // Firmata host:
  firmata_pinMode(13, MODE_OUTPUT); // LED tied to pin 13
  firmata_pinMode(4, MODE_INPUT);   // Digital input on pin 4
  firmata_pinMode(A0, MODE_ANALOG); // Analog input on pin 0
  firmata_pinMode(3, MODE_PWM);     // PWM LED on pin 3
```

```
}

void loop()
{
  // Print the value of our Firmata Arduino's A0 pin:
  int a0Value = firmata_analogRead(0);
  Serial.print("A0: ");
  Serial.println(a0Value);


  // Print the value of our digital input on pin 4:
  int d4Value = firmata_digitalRead(4);
  Serial.print("Pin 4: ");
  Serial.println(d4Value);

  if (d4Value == LOW)
  {
    // Scale the value of A0 to write a PWM output on pin 3:
    firmata_analogWrite(3, firmata_analogRead(0) / 4);
  }
  else
  {
    // Scale the value of A0 to write a PWM output on pin 3:
    firmata_analogWrite(3, 0);
  }

}

////////////////////////////////////
// Upper Level Firmata Functions //
////////////////////////////////////
// Firmata functions that you should use in your sketch above.
// If this was a class, these'd be public functions.

// firmata_init() --
// - Initialize our Firmata Serial port.
// - Set up a timer to read in Serial messages outside of loop
().
// - Configure our Firmata Arduino to report all digital outpu
ts
// - Configure our Firmata Arduino to report all analog output
s
void firmata_init()
{
  Serial1.begin(57600);
  // set a timer of length 100,000 microseconds ( 0.1 sec - o
r 10Hz)
  Timer1.initialize(1000);
  Timer1.attachInterrupt( checkSerial ); // attach the servic
e routine here

  // Turn on reporting for all digital ports
  for (int i=0; i<16; i++)
  {
    Serial1.write(REPORT_DIGITAL | i);
    Serial1.write(1);
  }
  // This function will check for analog channels and set them
  // to REPORTING
  firmata_queryAnalogMapping();
}

// firmata_digitalRead([pin]) --
// - Returns the latest digital input value we've read on the
//   requested pin.
```

```
// - digitalInputData[] is updated in firmata_processInput()
//   as serial messages come in.
int firmata_digitalRead(int pin)
{
  return (digitalInputData[pin >> 3] >> (pin & 0x07)) & 0x01;
}

// firmata_analogRead([pin])
// - Returns the latest analog value we've read on the request
ed
//   pin.
// - analogInputData[] is updated in firmata_processInput()
//   as serial messages come in.
int firmata_analogRead(int pin)
{
  return analogInputData[pin];
}

// firmata_pinMode([pin], [mode])
// - Set an Arduino pin to input, output, analog in, PWM, serv
o,
//   shift register, or i2c.
// - [pin] - can be any Arduino pin 0-13, A0-A7
// - [mode] - should be one of these defined values:
//          - MODE_INPUT  - Digital input
//          - MODE_OUTPUT - Digital output
//          - MODE_ANALOG - Analog input
//          - MODE_PWM    - Analog output
//          - MODE_SERVO  - Servo output
//          - MODE_SHIFT  - Shift register output
void firmata_pinMode(int pin, int mode)
{
  Serial1.write(SET_PIN_MODE);
  Serial1.write(pin);
  Serial1.write(mode);
}

// firmata_digitalWrite([pin], [value])
// - Set an Arduino digital pin to HIGH or LOW
// - [pin] - Any digital pin 0-18
// - [value] - LOW or HIGH
void firmata_digitalWrite(int pin, int value)
{
  int port = (pin >> 3) & 0x0F;
  int data;

  if (value)
    data |= (1 << (pin & 0x07));
  else
    data &= ~(1 << (pin & 0x07));

  Serial1.write(DIGITAL_MESSAGE | port); // Digital data
  Serial1.write(data & 0x7F); // Digital pins 0-6 bitmask
  Serial1.write(data >> 7);   // Digital pin 7 bitmask
}

// firmata_analogWrite([pin], [value])
// - Set an Arduino pin - CONFIGURED AS PWM (!) - to an
//   analog output value.
// - [pin] - Any analog output capable pin (3, 5, 6, 9, 10, 11
// - [value] - 0-255
void firmata_analogWrite(int pin, int value)
{

  Serial1.write(ANALOG_MESSAGE | (pin& 0x0F)); // Analog pin
```

```
  Serial1.write(value & 0x7F); // Analog LS 7 bits
  Serial1.write(value >> 7);   // Analog MS 7 bits
}

// firmata_servoWrite([pin], [value])
// - Set an Arduino pin - CONFIGURED AS SERVO (!) - to output
//   a servo signal.
void firmata_servoWrite(int pin, int value)
{
  Serial1.write(ANALOG_MESSAGE | (pin & 0x0F));
  Serial1.write(value & 0x7F);
  Serial1.write(value >> 7);
}

////////////////////////////////
// Low level Firmata functions //
////////////////////////////////
// Firmata helper functions you probably won't need to call in
// your sketch. If this was a class, these'd be private functi
ons.

// checkSerial()
// Interrupt-recurring function. Checks for available serial d
ata
// and processes any serial messages that come in.
void checkSerial()
{
  while (Serial1.available())
  {
    //Serial.write(Serial1.read());
    firmata_processInput((unsigned char) Serial1.read());
  }
  if (blinkFlag)
  {
    firmata_digitalWrite(13, HIGH); // Tell Arduino to write 1
3 HIGH
    blinkFlag = false;
  }
  else
  {
    firmata_digitalWrite(13, LOW); // Tell Arduino to write 1
3 HIGH
    blinkFlag = true;
  }
}

// firmata_processInput([inputData])
// Handles all Firmata messages - everything from version chec
ks
// to analog and digital readings.
void firmata_processInput(unsigned char inputData)
{
  int command;

  if (parsingSysex) // If we're parsing a system message
  {
    if (inputData == END_SYSEX)
    { // Received end of system message, process it
      parsingSysex = false;
      firmata_processSysexMessage();
    }
    else
    { // In the system message, add to it
      storedInputData[sysexBytesRead] = inputData;
      sysexBytesRead++;
```

```
    }
  }
  else if (waitForData > 0 && inputData < 128)
  { // Else waiting for data
    waitForData--; // Decrement wait for data
    storedInputData[waitForData] = inputData;

    if (executeMultiByteCommand != 0 && waitForData == 0)
    {
      switch(executeMultiByteCommand)
      {
      case DIGITAL_MESSAGE:
        firmata_setDigitalInputs(multiByteChannel,
                 (storedInputData[0] << 7) + storedInputData
[1]);
        break;
      case ANALOG_MESSAGE:
        firmata_setAnalogInput(multiByteChannel,
                 (storedInputData[0] << 7) + storedInputData
[1]);
        break;
      case REPORT_VERSION:
        firmata_setVersion(storedInputData[1], storedInputData
[0]);
        break;
      }
    }
  }
  else // Beginning of a message
  {
    if (inputData < 0xF0)
    {
      command = inputData & 0xF0;
      multiByteChannel = inputData & 0x0F;
    }
    else
    {
      command = inputData;
    }
    switch (command)
    {
    case DIGITAL_MESSAGE:
    case ANALOG_MESSAGE:
    case REPORT_VERSION:
      waitForData = 2;
      executeMultiByteCommand = command;
      break;
    case START_SYSEX:
      parsingSysex = true;
      sysexBytesRead = 0;
      break;
    }
  }
}

// firmata_processSysexMessage()
// - Process a system message.
// - Mostly just handles defining which pins are analog channe
ls.
void firmata_processSysexMessage()
{
  switch (storedInputData[0])
  {
  case ANALOG_MAPPING_RESPONSE:
    // Begin by setting every channel to NOT analog (127)
```

```
    for (int pin = 0; pin < sizeof(analogChannel); pin++)
      analogChannel[pin] = 127;
    // Enumerate analog channels:
    for (int i = 1; i < sysexBytesRead; i++)
      analogChannel[i - 1] = storedInputData[i];
    // Set each analog output to reporting
    for (int pin = 0; pin < sizeof(analogChannel); pin++)
    {
      if (analogChannel[pin] != 127)
      {
        Serial1.write(REPORT_ANALOG | analogChannel[pin]);
        Serial1.write(1);
      }
    }
    break;
  }
}

// firmata_queryAnalogMapping()
// - Send the ANALOG_MAPPING_QUERY request message.
// - Called in init to keep track of which pins are analog ins
void firmata_queryAnalogMapping()
{
  Serial1.write(START_SYSEX);
  Serial1.write(ANALOG_MAPPING_QUERY);
  Serial1.write(END_SYSEX);
}

// firmata_setDigitalInputs()
// - Set a value in the digitalInputData array to portData
void firmata_setDigitalInputs(int portNumber, int portData)
{
  digitalInputData[portNumber] = portData;
}

// firmata_setAnalogInput
// - Set an analog pin value to [value].
void firmata_setAnalogInput(int pin, int value)
{
  analogInputData[pin] = value;
}

void firmata_setVersion(int majorVersion, int minorVersion)
{
  //! Todo
}
```

After uploading that sketch to your Edison, your Arduino Block should begin blinking the D13 pin. You can also open up the Serial Monitor to see what values your Arduino Block is reading in on D4 and A0.

This serves as a simple Firmata client for the Edison, but it should be easily expandable. Try using any of these functions (defined in lower portions of the sketch) to add more features to your Edison Firmata Client:

- **`firmata_pinMode([pin], [mode])`** – As with any Arduino sketch, setting the pin's mode is critical. Firmata requires an extra bit of information. Use any Arduino pin for the `[pin]` variable. For the `[mode]` variable use either `MODE_INPUT`, `MODE_OUTPUT`, `MODE_ANALOG`, `MODE_PWM`, `MODE_SERVO`, or `MODE_SHIFT`.
- **`firmata_digitalRead([pin])`** – Read in the digital value of a pin. This function will return a value between 0 and 1. That pin should be set as `MODE_INPUT` before calling this function.

- **`firmata_analogRead([pin])`** – Read in the value of an anlog pin. This function will return a value between 0 and 1023. The pins should be set as `MODE_ANALOG` before being read.
- **`firmata_digitalWrite([pin], [value])`** – Write a digital pin either HIGH or LOW. The pin should be set as `MODE_OUTPUT` before calling the function.
- **`firmata_analogWrite([pin], [value])`** – Write an analog-output pin to a value between 0 and 255. The pin must be PWM-capable – that means either pin 3, 5, 6, 9, 10, or 11. And it should be configured as a `MODE_PWM` before hand.
- **`firmata_servoWrite([pin], [value])`** –Set a pin to output a servo signal. The value here is an angle between 0 and whatever the high-end angle of your servo is. Remember to set the pin to `MODE_SERVO` before calling this function!

Don't feel like you have to user Firmata when using the Arduino block with your Edison. It's a great place to start, though, if you're looking for a simple communication protocol over the serial interface.

Resources and Going Further

Now that you have had a brief overview of the Arduino Block, take a look at some of these other tutorials. These tutorials cover programming, Block stacking, and interfacing with the Intel Edison ecosystems.

Edison General Topics:

- General Guide to Sparkfun Blocks for Intel Edison
- Edison Getting Started Guide
- Loading Debian (Ubilinix) on the Edison

Check out these other Edison related tutorials from SparkFun:

SparkFun Blocks for Intel® Edison - UART Block

A quick overview of the features of the UART Block.

SparkFun Blocks for Intel® Edison - microSD Block

A quick overview of the features of the microSD Block.

Edison Getting Started Guide

An introduction to the Intel® Edison. Then a quick walk through on interacting with the console, connecting to WiFi, and doing...stuff.

SparkFun Blocks for Intel® Edison - Console Block

A quick overview of the features of the Console Block.